



UNIVERSITY OF CALIFORNIA,
SANTA BARBARA

SENIOR THESIS

Bregman Algorithms

Author:
Jacqueline BUSH

Supervisor:
Dr. Carlos
GARCÍA-CERVERA

June 9, 2011

Acknowledgments

I could not have done this without the help and support of my advisor
Dr. Carlos Garcia-Cervera.

Abstract

In this paper we investigate methods for solving the Basis Pursuit problem and the Total Variation denoising problem. The methods studied here are based on the Bregman iterative regularization, and efficient algorithm for convex, constraint optimization problems. We study two different versions of the original Bregman iterative algorithm: the Linearized Bregman algorithm, and the Split Bregman algorithm. We find that the original Bregman Algorithm has good convergence properties, but while these properties hold for the linearized version, they do not always hold for the Split Bregman Algorithm.

Contents

1	Introduction	1
2	Background	4
3	Bregman Iterative Algorithm	8
3.1	Convergence properties of the Bregman Iterative Algorithm . . .	10
4	Linearized Bregman Iterative Algorithm	14
4.1	Linearized Bregman Iterative Algorithm with <i>Kicking</i>	17
5	Split Bregman Algorithm	20
5.1	Anisotropic TV Denoising	23
5.2	Isotropic TV Denoising	26
6	Numerical Results	29
6.1	Basis Pursuit Problem	29
6.2	Image Denoising Problem	32
7	Further Study	37
8	Matlab Code	38
8.1	Linearized Bregman	42
8.2	Linear Bregman with kicking	42
8.3	Anisotropic TV Denoising	44
8.4	Isotropic TV Denoising	47

List of Figures

1	Linearized Bregman Algorithm	18
2	Linearized Bregman Algorithm	20
3	Linearized Bregman Algorithm, $m=10$, $n=30$, $\ u\ _0 = 0.1n$	31
4	Linearized Bregman Algorithm, $m=50$, $n=200$, $\ u\ _0 = 0.05n$	31
5	Split Bregman Results using Fluid Jet Image	34
6	Split Bregman Error Results using the Fluid Jet Image	34
7	Split Bregman Results using Clown Image	35
8	Split Bregman Error Results using the Clown Image	35
9	Split Bregman Results using Durer Detail Image	36
10	Split Bregman Error Results using the Durer Detail Image	36

1 Introduction

In this paper we study Bregman Iterative algorithms, and their ability to solve constrained optimization problems such as the Basis Pursuit problem and the Total Variation (TV) denoising problem.

In the Basis Pursuit problem we are interested in finding a solution $\mathbf{u} \in \mathbb{R}^n$ to the linear system of equations $\mathbf{A}\mathbf{u} = \mathbf{f}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $m \ll n$, and $\mathbf{f} \in \mathbb{R}^m$. We assume that the rows of \mathbf{A} are linearly independent. As a result, the system is underdetermined, and therefore it has infinitely many solutions. In the Basis Pursuit problem we look for a solution to the linear system of equations with minimal l_1 norm, i.e., we want to solve the constrained problem

$$\min_{\text{subject to } \mathbf{A}\mathbf{u}=\mathbf{f}} \|\mathbf{u}\|_1, \quad (1.1)$$

where

$$\|\mathbf{u}\|_1 = \sum_{i=1}^n |u_i|. \quad (1.2)$$

Imposing the constraint in (1.1) leads to a number of difficulties, so instead, we relax the constraint, and solve the unconstrained basis pursuit problem

$$\min_{\mathbf{u} \in \mathbb{R}^n} \mu \|\mathbf{u}\|_1 + \frac{1}{2} \|\mathbf{A}\mathbf{u} - \mathbf{f}\|_2^2, \quad (1.3)$$

where $\mu \in \mathbb{R}$ is a positive constant. One can prove that when $\mu \rightarrow 0$, the solution to the unconstrained problem converges to a solution of the constrained problem.

The basis pursuit problem appears in applications of compressed sensing, where a signal assumed to be sparse is reconstructed from incomplete data on it. Using this principle one can encode a sparse signal $\bar{\mathbf{u}}$ using a linear transformation $\mathbf{A}\bar{\mathbf{u}} = \mathbf{f} \in \mathbb{R}^m$, and then recover $\bar{\mathbf{u}}$ from \mathbf{f} using (1.3). Recent applications of the basis pursuit problem include compressive imaging, and computer vision

tasks to name a few [10].

In our second problem, the TV denoising problem, we want to recover an image that has been affected by noise. This leads to a problem of the form

$$\min_{u \in BV(\Omega)} \|u\|_{BV} + H(u), \quad (1.4)$$

where $H(u)$ is a convex function. The functional in (1.4) is minimized among functions of bounded variation (BV): Given $\Omega \subset \mathbb{R}^n$ and $u : \Omega \rightarrow \mathbb{R}$, we define its bounded variation as [4]

$$\|u\|_{BV} = \sup \left\{ \int_{\Omega} u \operatorname{div}(g) \, dx \mid g = (g_1, g_2, \dots, g_n) \in C_0^1(\Omega, \mathbb{R}^n) \right. \\ \left. \text{and } |g(x)| \leq 1 \text{ for } x \in \Omega \right\},$$

where

$$\operatorname{div}(g) = \sum_{i=1}^n \frac{\partial g_i}{\partial x_i}. \quad (1.5)$$

The space of functions of bounded variation is defined as $BV(\Omega) = \{u : \Omega \rightarrow \mathbb{R} \mid \|u\|_{BV} < +\infty\}$, and it is a Banach space. Notice that if $u \in C^1(\Omega)$, then integration by parts gives,

$$\|u\|_{BV} = \int_{\Omega} |\nabla u| \, dx = \|\nabla u\|_1. \quad (1.6)$$

Therefore, the Sobolev space $W^{1,1}(\Omega)$ of functions in $L^1(\Omega)$ whose gradient is in $L^1(\Omega)$ is contained in $BV(\Omega)$: $W^{1,1}(\Omega) \subset BV(\Omega)$. However, $BV(\Omega) \neq W^{1,1}(\Omega)$: Consider for instance a set $E \subseteq \Omega$ with a smooth boundary, and $u = \chi_E$, its characteristic function. Then,

$$\|\chi_E\|_{BV} = \mathcal{H}^1(\partial E),$$

the surface area of ∂E . However, since χ_E is discontinuous at the interface ∂E , $\nabla \chi_E \notin L^1(\Omega)$. This example, which might seem artificial at first, is the typical situation when one tries to recover an image, since images are best represented by a collection of disjoint sets, i.e., separated by sharp interfaces. Thus $BV(\Omega)$, and not the Sobolev space $W^{1,p}(\Omega)$ of functions in $L^p(\Omega)$ whose gradient is also in $L^p(\Omega)$, seems to be the natural space in which to consider the TV denoising problem. However, evaluating the BV -norm of a function can be costly, and it is sometimes replaced by the L^1 -norm of the gradient. Since the focus in this thesis is the application of Bregman-type algorithms to this problem, this is the approach that we will take.

Thus, in the TV denoising problem considered here, we are given as data a function $\mathbf{f} \in L^2(\Omega)$, which is the representation of the noisy image. We try to recover the original image by minimizing

$$\min_{u \in BV(\Omega)} \|\nabla u\|_1 + \frac{\mu}{2} \|u - f\|_2^2, \quad (1.7)$$

where $\mu > 0$ is a constant. Reducing the noise from obtained images can be applied to many fields. For example, if we can reduce the noise on images coming in from a satellite one can save the millions of dollars that it would require to send an astronaut to repair it. It can also be applied to reduce noise from an medical test such as an MRI or CT scan.

This thesis is organized as follows: In section 2 we introduce some of the definitions and concepts to be used throughout the remainder of the thesis. In section 3 the Bregman iterative algorithm is introduced and its convergence properties are studied. A linearized version of the algorithm is derived in section 4. One drawback of the Linearized Bregman algorithm is that it can reach periods of stagnation, where progress toward the solution is slow. A method called *Kicking*

that reduces the number of iterations that the algorithm spends in these periods of stagnation is also described in section 4. In section 5 the Split Bregman algorithm is introduced and studied. It is shown in section 6 that the Linearized Bregman algorithm solves the basis pursuit problem quickly and accurately. It is also shown that the Split Bregman algorithm is not monotonic, unlike the iterative Bregman algorithm introduced in section 3.

2 Background

Several algorithms have been introduced to solve (1.3), such as the l_1l_s algorithm developed by S-J. Kim, K. Koh, and S. Boyd [7]. The authors applied an interior-point method to a log-barrier formulation of (1.3). Each interior point iteration involved solving a system of linear equations. These iterations were accelerated using a preconditioned conjugate gradient method, for which S-J. Kim, K. Koh, and S. Boyd developed an efficient preconditioner [7]. While this algorithm does solve (1.3), it can be expensive in time and computer memory. For medium sized problems it can require up to a hundred iterations, but for very large problems the algorithm could take several hundred iterations to compute a solution with relative tolerance 0.01 [7]. The Bregman Iterative Algorithms described in the next sections are proposed as a faster, less computationally expensive alternative.

In what follows, we consider a normed space X , with norm $\|\cdot\|$.

Definition A function $J : X \rightarrow \mathbb{R}$ is said to be *convex* if $\forall x, y \in X$ and any $t \in [0, 1]$,

$$J(tx + (1 - t)y) \leq tJ(x) + (1 - t)J(y). \quad (2.1)$$

Definition We say that $J : X \rightarrow \mathbb{R}$ is *lower semicontinuous* at $u \in X$ if

$$\liminf_{x \rightarrow u} J(x) \geq J(u). \quad (2.2)$$

or, equivalently, if for all $\lambda \in \mathbb{R}$, the set $\{x : J(x) \leq \lambda\}$ is closed.

Both the l_1 -norm and the BV-norm are lower semicontinuous functions. To see this, consider the set

$$S = \{\mathbf{u} \in \mathbb{R}^n : \|\mathbf{u}\|_1 \leq \lambda\}. \quad (2.3)$$

If $\lambda \leq 0$ then S is the empty set, which is closed by definition. Now suppose that $\lambda > 0$, and assume $\{\mathbf{u}_m\}$ is a sequence in S that converges to $\mathbf{u} \in \mathbb{R}^n$. It follows that

$$\|\mathbf{u}\|_1 \leq \limsup_{m \rightarrow \infty} (\|\mathbf{u}_m\|_1 + \|\mathbf{u} - \mathbf{u}_m\|_1) \leq \lambda. \quad (2.4)$$

Hence $\mathbf{u} \in S$, and S is a closed set in \mathbb{R}^n .

The fact that the BV-norm is lower semicontinuous follows from the following results, whose proof can be found in [4], and that will be used in the following section:

Proposition 2.1. (*Semicontinuity*): Let $\Omega \in \mathbb{R}^n$ be an open set and $\{f_j\}$ a sequence of functions in $BV(\Omega)$ which converge in $L^1_{loc}(\Omega)$ to a function f . Then

$$\|f\|_{BV} \leq \liminf_{j \rightarrow \infty} \|f_j\|_{BV}. \quad (2.5)$$

Proposition 2.2. (*Compactness*): Let Ω be a bounded open set in \mathbb{R}^n . Then the sets of functions uniformly bounded in the BV-norm are relatively compact in $L^1(\Omega)$.

Definition The function $J : X \rightarrow \mathbb{R}$ is *coercive* if

$$\lim J(u) = +\infty \text{ for } \|u\| \rightarrow \infty.$$

The importance of this definition can be seen from the following

Proposition 2.3. *Let V be a reflexive Banach space, and X a non-empty closed, convex subset of V . Assume that $J : X \rightarrow \mathbb{R}$ is convex, lower semi-continuous, and coercive. Then, the problem*

$$\inf_{u \in X} J(u)$$

has at least one solution.

The poof can be found in [3].

Definition The *dual space* of X , denoted by X^* , is defined to be the set of functions $l : X \rightarrow \mathbb{R}$ such that l is linear and continuous.

Given an element $l \in X^*$, we denote $l(u) = \langle l; u \rangle$ for all $u \in X$.

Definition Suppose $J : X \rightarrow \mathbb{R}$ is a convex function and $u \in X$. An element $p \in X^*$ is called a *subgradient of J at u* if $\forall v \in X$

$$J(v) - J(u) - \langle p, v - u \rangle \geq 0. \tag{2.6}$$

The set of all subgradients of J at u is called the *subdifferential of J at u* , and it is denoted by $\partial J(u)$.

Notice that the subdifferential extends the notion of gradient of a function. For example, consider $f(x) = \|x\|_2$ in \mathbb{R}^n , the Euclidean norm. This function is not differentiable at $x = 0$, however it has a subdifferential at $x = 0$, and in fact $\partial f(0)$ is the closed ball (in the Euclidean norm) around 0 with radius 1, $\bar{B}_1(0)$.

Definition Consider a function $J : X \rightarrow \mathbb{R}$. We call the limit as $\lambda \rightarrow 0_+$, if it exists, of

$$\frac{J(u + \lambda v) - J(u)}{\lambda}$$

the *directional derivative of J at u in the direction v* , and denote it by $J'(u; v)$.

If there exists $p \in X^*$ such that

$$J'(u, v) = \langle p, v \rangle \quad \forall v \in X,$$

we say that J is *Gateaux-differentiable at u* , and call p the *Gateaux-differential of J at u* , and denote it by $J'(u)$.

One can prove that if $J : X \rightarrow \mathbb{R}$ is Gateaux-differentiable at $u \in X$, then $\partial J(u) = \{J'(u)\}$. And conversely, if J is continuous at $u \in X$ and has only one subgradient, then J is Gateaux-differentiable at u and $\partial J(u) = \{J'(u)\}$. In addition notice that $J : X \rightarrow \mathbb{R}$ has a minimum at $u \in X$ if and only if $0 \in \partial J(u)$, for in that case

$$J(v) - J(u) \geq \langle 0, v - u \rangle = 0.$$

Finally, all the algorithms discussed in this paper rely on the Bregman distance, introduced by L. M. Bregman in 1966 [1].

Definition Suppose $J : X \rightarrow \mathbb{R}$ is a convex function, $u, v \in X$ and $p \in \partial J(v)$. Then the Bregman Distance between points u and v is defined by

$$D_J^p(u, v) = J(u) - J(v) - \langle p, u - v \rangle. \quad (2.7)$$

The Bregman distance has several nice properties that make it an efficient tool to solve l_1 regularization problems, such as,

Property 2.4. For all $u, v \in X$, and $p \in \partial J(v)$, $D_J^p(u, v) \geq 0$.

Proof. Since $p \in \partial J(v)$, this property follows directly from the definition of subdifferentials. □

Property 2.5. $D_J^p(v, v) = 0$

Proof. Observe, $D_J^p(v, v) = J(v) - J(v) - \langle p, 0 \rangle = 0$. □

Note that in general the Bregman Distance is not symmetric. For instance if $p \in \partial J(v) \cap \partial J(u)$ then

$$\begin{aligned} D_J^p(u, v) &= J(u) - J(v) - \langle p, u - v \rangle \\ &= J(u) - J(v) + \langle p, v - u \rangle \\ &= -(J(v) - J(u) - \langle p, v - u \rangle) \\ &= -D_J^p(v, u) \end{aligned}$$

3 Bregman Iterative Algorithm

In [8], S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin proposed the Bregman Iterative Algorithm as an efficient algorithm for solving problems of the form

$$\min_u \{J(u) + H(u, f)\}, \tag{3.1}$$

where for a closed and convex set X both $J : X \rightarrow \mathbb{R}$ and $H : X \rightarrow \mathbb{R}$ are convex nonnegative functions with respect to $u \in X$, for a fixed f . In addition $H(u, f)$ is assumed to be differentiable. Recall f is the vector or matrix, depending on the problem, that u was encoded into. Osher, Burger, Goldfarb, Xu, and Yin defined the Bregman iterative algorithm as follows

Bregman Iterative Algorithm:

Initialize: $k=0, u^0 = 0, p^0 = 0$

while “ u^k not converge” **do**

$$u^{k+1} = \operatorname{argmin}_u D_J^{p^k}(u, u^k) + H(u)$$

$$p^{k+1} = p^k - \nabla H(u^{k+1}) \in \partial J(u^{k+1})$$

$$k = k + 1$$

end while

It is easy to see that the first iteration yields

$$u^1 = \min_{u \in X} (J(u) + H(u)). \quad (3.2)$$

Hence the first iteration of this algorithm solves (3.1). However to solve our initial problem we need the residual term to be minimal. This is why the Bregman Iterative Algorithm continues until the residual term converges.

While working with basis pursuit problems we let $J(u) = \|u\|_1$ and $H(u) = \frac{1}{2} \|Au - f\|_2^2$. When studying image denoising we let $J(u) = \|u\|_{BV}$ or $J(u) = \|\nabla u\|_1$, and $H(u) = \|u - f\|_2^2$.

Proposition 3.1. *Let X be a reflexive Banach space. The iterative procedure in the above algorithm is well defined if $H : X \rightarrow \mathbb{R}$ and $J : X \rightarrow \mathbb{R}$ are convex, J is lower semicontinuous and coercive, H is bounded below and Gateaux-differentiable.*

Proof. Since H is bounded below, we can assume that H is nonnegative without loss of generality. Otherwise, define $\tilde{H} = H - \inf H$. Since H and \tilde{H} differ by a constant, the algorithm applied to $J + \tilde{H}$ generates the same sequence $\{u^k\}$.

The above algorithm is well defined if for each k ,

$$Q_k(u) = D_J^{p^{k-1}}(u, u^{k-1}) + H(u) \quad (3.3)$$

has a minimizer, u^k , and $p^k = p^{k-1} - \nabla H(u^k) \in \partial J(u^k)$. Recall that the

Bregman Distance and $H(u)$ are nonnegative. Hence,

$$\begin{aligned}
Q_k(u) &= J(u) - J(u^{k-1}) - \langle p^{k-1}, u - u^{k-1} \rangle + H(u) \\
&\geq J(u) - J(u^{k-1}) - \langle p^{k-1}, u - u^{k-1} \rangle \\
&= D_J^{p^{k-1}}(u, u^{k-1}) \geq 0
\end{aligned}$$

It follows that $Q_k(u)$ has a lower bound for all k . From the coercivity of J and the continuity of H , it follows by Prop. 2.3 that $Q_k(u)$ has a minimizer, u^k , for each k . Since $Q_k(u)$ has a minimum at u^k , it follows that $0 \in \partial Q_k(u^k)$. Hence,

$$\begin{aligned}
0 &\in \partial J(u^k) - p^{k-1} + \nabla H(u^k), \\
\Rightarrow p^{k-1} - \nabla H(u^k) &\in \partial J(u^k), \\
\Rightarrow p^k &\in \partial J(u^k) \quad \forall k \geq 1.
\end{aligned}$$

□

Note that if $J(u) = \|u\|_{BV}$, then minimizing sequences are precompact, by Proposition 2.2, and J is lower semicontinuous by Proposition 2.1. The existence of minimizers follows then from the Direct Method in the Calculus of Variations [2].

3.1 Convergence properties of the Bregman Iterative Algorithm

The Bregman Iterative Algorithm has been applied to many problems including image denoising and basis pursuit because it has some very nice convergence properties. These properties include monotonic decrease in the residual term, convergence to the original image or signal that we are trying to recover in the residual term with exact data, and convergence in terms of Bregman distance

to the original image or signal with noisy data. We assume that X , J , and H satisfy the same assumptions as in Proposition 3.1.

In the first result, we show that the sequence generated by the algorithm monotonically decreases H :

Proposition 3.2. *Monotonic decrease in H :*

$$H(u^{k+1}) \leq H(u^{k+1}) + D_J^{p^k}(u^{k+1}, u^k) \leq H(u^k). \quad (3.4)$$

Proof. Recall that the Bregman distance is nonnegative and that u^{k+1} minimizes $D_J^{p^k}(u, u^k) + H(u)$. Hence

$$H(u^{k+1}) \leq H(u^{k+1}) + D_J^{p^k}(u^{k+1}, u^k) \leq H(u^k) + D_J^{p^k}(u^k, u^k) = H(u^k).$$

□

Next we show that the sequence of residual terms $\{H(u^k)\}$ converges to the minimum value of H :

Proposition 3.3. *If \tilde{u} minimizes $H : X \rightarrow \mathbb{R}$ and $J(\tilde{u}) < \infty$, then*

$$H(u^k) \leq H(\tilde{u}) + J(\tilde{u})/k. \quad (3.5)$$

Proof. Observe that

$$\begin{aligned}
D^{p^k}(u, u^k) &+ D^{p^{k-1}}(u^k, u^{k-1}) - D^{p^{k-1}}(u, u^{k-1}) = \\
&= J(u) - J(u^k) + \langle p^k, u^k - u \rangle + J(u^k) - J(u^{k-1}) \\
&+ \langle p^{k-1}, u^{k-1} - u^k \rangle - J(u) + J(u^{k-1}) - \langle p^{k-1}, u^{k-1} - u \rangle \\
&= \langle p^k, u^k - u \rangle + \langle p^{k-1}, u^{k-1} - u^k \rangle - \langle p^{k-1}, u^{k-1} - u \rangle \\
&= \langle p^k, u^k - u \rangle - \langle p^{k-1}, u^k - u \rangle \\
&= \langle p^k - p^{k-1}, u^k - u \rangle.
\end{aligned}$$

By construction,

$$p^k - p^{k-1} = -\nabla H(u^k). \quad (3.6)$$

Since H is convex it follows that

$$\langle p^{k-1} - p^k, u^k - u \rangle = \langle \nabla H(u^k), u^k - u \rangle \leq H(u) - H(u^k). \quad (3.7)$$

Then we get the following equation:

$$D^{p^j}(u, u^j) + D^{p^{j-1}}(u^j, u^{j-1}) - D^{p^{j-1}}(u, u^{j-1}) \leq H(u) - H(u^j) \quad \forall j. \quad (3.8)$$

Now if we sum in j in equation (3.8) we get,

$$\begin{aligned}
&\sum_{j=1}^k D^{p^j}(u, u^j) + D^{p^{j-1}}(u^j, u^{j-1}) - D^{p^{j-1}}(u, u^{j-1}) \leq \sum_{j=1}^k H(u) - H(u^j) \\
\Rightarrow D^{p^k}(\tilde{u}, u^k) &+ \sum_{j=1}^k \left[D^{p^{j-1}}(u^j, u^{j-1}) - H(\tilde{u}) + H(u^j) \right] \leq D^0(\tilde{u}, u^0) = J(\tilde{u}).
\end{aligned}$$

From proposition 3.2 and since the Bregman distance is never negative we can

rewrite the above equation as follows,

$$D^{p^k}(\tilde{u}, u^k) + k[H(u^k) - H(\tilde{u})] \leq J(\tilde{u}). \quad (3.9)$$

Hence

$$H(u^k) \leq H(\tilde{u}) + \frac{J(\tilde{u})}{k}. \quad (3.10)$$

□

Next we show that as long as the residual term $H(u^k; f) > \delta^2$ then the Bregman distance between the original signal or image and the current iteration is decreasing.

Proposition 3.4. *For an open set X let $H : X \rightarrow \mathbb{R}$ suppose $H(\tilde{u}; f) \leq \delta^2$ and $H(\tilde{u}; g) = 0$ (f, g, \tilde{u}, δ , represent noisy data, noiseless data, perfect recovery, and noise level, respectively). Then $D_J^{p^{k+1}}(\tilde{u}, u^{k+1}) < D_J^{p^k}(\tilde{u}, u^k)$ as long as $H(u^{k+1}) > \delta^2$.*

Proof. Suppose $H(\tilde{u}) < \delta^2$ and insert it into equation (3.8), then

$$\begin{aligned} D^{p^j}(\tilde{u}, u^j) + D^{p^{j-1}}(u^j, u^{j-1}) - D^{p^{j-1}}(\tilde{u}, u^{j-1}) &\leq \delta^2 - H(u^j) \\ \Rightarrow D^{p^j}(\tilde{u}, u^j) - D^{p^{j-1}}(\tilde{u}, u^{j-1}) &\leq \delta^2 - H(u^j). \end{aligned}$$

It follows that if $H(u^j) > \delta^2$ then

$$D^{p^j}(\tilde{u}, u^j) \leq D^{p^{j-1}}(\tilde{u}, u^{j-1}).$$

□

4 Linearized Bregman Iterative Algorithm

The Bregman algorithm is a good tool to solve the basis pursuit problem explained in the introduction. However, at each step the algorithm requires the minimization of

$$D^p(u, u^k) + H(u), \quad (4.1)$$

which can be computationally expensive. A linearized version of the Bregman Iterative Algorithm was introduced by Yin, Osher, Goldfarb and Darbon in [10]. The main advantage of the linearized algorithm is that the minimization of (4.1) is replaced by a minimization step that can be solved exactly, which allows for efficient computation. We begin by linearizing $H(u)$: Given u^k we approximate $H(u)$ by,

$$H(u) = H(u^k) + \langle \nabla H(u^k), u - u^k \rangle. \quad (4.2)$$

Since this approximation is only accurate for u close to u^k , Yin, Osher, Goldfarb, and Darbon added the penalty term $\frac{1}{2\delta} \|u - u^k\|_2^2$. The original problem is then replaced by

$$u^{k+1} = \arg \min_u D_J^p(u, u^k) + H(u^k) + \langle \nabla H(u^k), u - u^k \rangle + \frac{1}{2\delta} \|u - u^k\|_2^2. \quad (4.3)$$

Notice that the penalty term also makes the objective function bounded below, and

$$\|(u - u^k) + \delta \nabla H(u^k)\|_2^2 = \|u - u^k\|_2^2 + 2\delta \langle \nabla H(u^k), u - u^k \rangle + \delta^2 \|H(u^k)\|_2^2.$$

In addition, observe that $\|H(u^k)\|_2^2$ and $H(u^k)$ are constants with respect to u . It follows that the iteration (4.3) is equivalent to the iteration

$$u^{k+1} = \arg \min_u D_J^p(u, u^k) + \frac{1}{2\delta} \|u - (u^k - \delta \nabla H(u^k))\|_2^2. \quad (4.4)$$

We apply this method to the Basis Pursuit problem (1.3), and let $H(u) = \frac{1}{2}\|Au - f\|_2^2$. Plugging this H into (4.4) we get

$$u^{k+1} = \arg \min_u D_J^k(u, u^k) + \frac{1}{2\delta}\|u - (u^k - \delta A^T(Au^k - f))\|_2^2. \quad (4.5)$$

Now we derive a special case of (4.5). The case when $J(u) = \mu\|u\|_1$ and $\mu > 0$. In section 3 we defined $p^{k+1} \in \partial J(u^{k+1})$ by

$$p^{k+1} = p^k - \nabla H(u^k) \quad (4.6)$$

$$= p^k - \frac{1}{\delta}(u^{k+1} - (u^k - \delta A^T(Au^k - f))). \quad (4.7)$$

Hence,

$$p^{k+1} = p^k - A^T(Au^k - f) - \frac{(u^{k+1} - u^k)}{\delta} = \dots = \sum_{j=0}^k A^T(f - Au^j) - \frac{u^{k+1}}{\delta}. \quad (4.8)$$

Let

$$v^k = \sum_{j=0}^k A^T(f - Au^j). \quad (4.9)$$

Then,

$$\delta v^k - \delta p^{k+1} = u^{k+1}, \quad (4.10)$$

$$\Rightarrow p^k = v^{k-1} - \frac{u^k}{\delta}. \quad (4.11)$$

Then we get

$$\begin{aligned}
u^{k+1} &= \min_u \mu \|u\|_1 - \mu \|u^k\|_1 - \langle u - u^k, p^k \rangle + \frac{1}{2\delta} \|u - (u^k - \delta A^T(Au^k - f))\|_2^2 \\
&= \min_u \mu \sum_{i=1}^n |u_i| - \mu \|u^k\|_1 - \langle u - u^k, v^{k-1} - \frac{u^k}{\delta} \rangle \\
&\quad + \frac{1}{2\delta} \|u - (u^k - \delta A^T(Au^k - f))\|_2^2 \\
&= \min_u \mu \sum_{i=1}^n |u_i| - \langle u, v^{k-1} - \frac{u^k}{\delta} \rangle + \frac{1}{2\delta} \|u - (u^k - \delta A^T(Au^k - f))\|_2^2 + C
\end{aligned}$$

Where C denotes constant terms with respect to $u \in X$. Notice that $u \in X$ is componentwise separable. Hence we can minimize each component of $u \in X$ separately. Doing this we get,

$$\begin{aligned}
0 &= \begin{cases} \mu - v_i^{k-1} + \frac{u_i^k}{\delta} + \frac{u_i}{\delta} - \frac{u_i^k}{\delta} - (A^T(f - Au^k))_i & u_i > 0 \\ 0 & u_i = 0 \\ \mu + v_i^{k-1} - \frac{u_i^k}{\delta} - \frac{u_i}{\delta} + \frac{u_i^k}{\delta} + (A^T(f - Au^k))_i & u_i < 0 \end{cases} \\
0 &= \begin{cases} \mu - v_i^k + \frac{u_i}{\delta} & u_i > 0 \\ 0 & u_i = 0 \\ v_i^k + \mu - \frac{u_i}{\delta} & u_i < 0 \end{cases} \\
\Rightarrow u_i^{k+1} &= \begin{cases} \delta(v_i^k - \mu) & v_i^k \in (\mu, \infty) \\ 0 & v_i^k \in [-\mu, \mu] \\ \delta(v_i^k + \mu) & v_i^k \in (-\infty, -\mu) \end{cases}
\end{aligned}$$

For convenience of notation we define the shrinkage function as follows: For $a \geq 0$,

$$\text{shrink}(y, a) = \begin{cases} y - a, & y \in (a, \infty) \\ 0, & y \in [-a, a] \\ y + a, & y \in (-\infty, -a) \end{cases}$$

It follows that

$$u_i^{k+1} = \delta \text{shrink}(v_i^k, \mu). \quad (4.12)$$

The linearized algorithm for this special case can then be written as

Linearized Bregman Algorithm:

```

Initialize: u = 0, v=0
while “||f - Au|| does not converge” do
    vk+1 = vk + AT(f - Auk)
    uk+1 = δ shrink(vik+1, μ)
end while

```

4.1 Linearized Bregman Iterative Algorithm with *Kicking*

When running the Linearized Bregman iterative algorithm sometimes there exist periods of stagnation, where the residual stays almost constant for a large number of iterations. This is illustrated in figure 1, where we solve the Basis Pursuit problem (1.3) using the Linearized Bregman algorithm. We chose a random matrix **A** of size 50 × 200, and the number of nonzero elements in the original signal was 20.

It took Matlab 0.438609 seconds to run the Linearized Bregman iterative algorithm and reduce the initial residual to 0.000932. While the Linearized Bregman algorithm is already fast, we can make it faster by removing the stagnation periods seen in figure 1. To get rid of unnecessary iterations we estimate the number of steps required to leave the stagnation period. Observe that during a stagnation period of, say, *m* steps,

$$\begin{cases} u^{k+j} = u^{k+1}, \\ v^{k+j} = v^k + jA^T(f - Au^{k+1}) \quad j = 1, \dots, m. \end{cases}$$

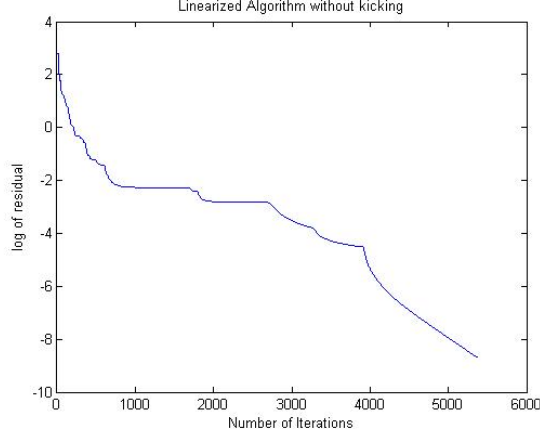


Figure 1: Linearized Bregman Algorithm

Now $u^{k+1} = \delta\text{shrink}(v^k, \mu)$ and during a period of stagnation u^k moves only slightly. It follows that v_i^k will remain fairly constant if $u_i^k \neq 0$. In other words v_i^k will only keep changing if $u_i^k = 0$. Let I_0 be all the indices where $u_i^k = 0$. It follows that

$$\begin{cases} u_i^{k+j} \equiv u_i^{k+1} & \forall i \\ v_i^{k+j} = v_i^k + j(A^T(f - Au^{k+1}))_i & i \in I_0 \\ v_i^{k+j} \equiv v_i^{k+1} & i \notin I_0 \end{cases}$$

The stagnation ends when for some $i \in I_0$, v_i^k leaves the interval $[-\mu, \mu]$. We can estimate the number of steps required for some v_i^k to leave the interval $[-\mu, \mu]$ by

$$s_i = \left\lceil \frac{\mu \text{sign} \cdot ((A^T(f - Au^{k+1}))_i) - v_i^{k+1}}{(A^T(f - Au^{k+1}))_i} \right\rceil \quad \forall i \in I_0, \quad (4.13)$$

and

$$s = \min_{i \in I_0} \{s_i\} \quad (4.14)$$

Using s we can predict the end of the stagnation period. We define the next update by

$$\begin{cases} u^{k+s} \equiv u^{k+1} \\ v^{k+s} = v^k + sA^T(f - Au^{k+1}). \end{cases}$$

This is the concept of *Kicking*, introduced by Osher, Moa , Dong, and Yin [9].

The Linearized Bregman iterative algorithm with Kicking is:

Linearized Bregman Iteration with Kicking:

```

Initialize:  $u = 0, v = 0$ 
while " $\|f - Au\|$  does not converge" do
     $u^{k+1} = \delta\text{shrink}(v^k, \mu)$ 
    if " $u^{k+1} \approx u^k$ " then
        calculate  $s$  from (4.13) and (4.14)
         $v_i^{k+1} = v_i^k + s(A^T(f - Au^{k+1}))_i, \quad \forall u \in I_0$ 
         $v_i^{k+1} = v_i^k \quad \forall u \notin I_0$ 
    else
         $v^{k+1} = v^k + A^T(f - Au^{k+1})$ 
    end if
end while

```

We ran this algorithm for the same matrix \mathbf{A} and original signal $\bar{\mathbf{u}}$ to see if the *kicking* method really reduced the number of iterations required to solve (1.3).

It took Matlab 0.065734 seconds to run this algorithm and reduce the initial residual to 0.000943 for this \mathbf{A} and $\bar{\mathbf{u}}$. Notice that the linearized Bregman algorithm took less than half of the time the linearized Bregman algorithm without kicking. The number of iterations required to solve (1.3) also decreased from 5369 iterations to 665 iterations when we applied kicking.

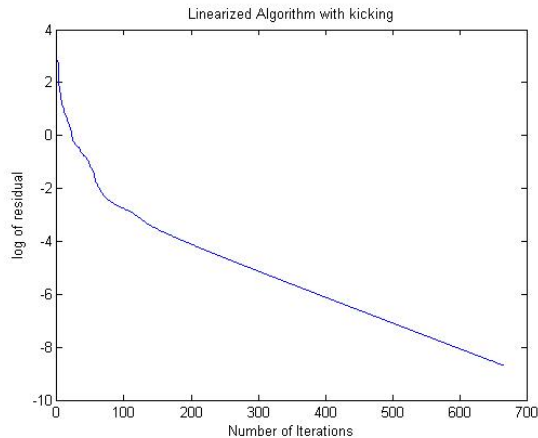


Figure 2: Linearized Bregman Algorithm

5 Split Bregman Algorithm

In [5] T. Goldstein, and S. Osher introduced the Split Bregman method to solve the general optimization problem of the form

$$\min_{u \in X} \|\Phi(u)\|_1 + H(u), \tag{5.1}$$

where X is a closed, convex set, and both $\Phi : X \rightarrow \mathbb{R}$, and $H : X \rightarrow \mathbb{R}$ are convex functions. We rewrite (5.1) in terms of the equivalent constrained minimization problem,

$$\min_{u \in X, d \in \mathbb{R}} \|d\|_1 + H(u) \quad \text{such that } d = \Phi(u). \tag{5.2}$$

As in the Basis Pursuit problem, we relax the constraints and work with the unconstrained problem

$$\min_{u \in X, d \in \mathbb{R}} \|d\|_1 + H(u) + \frac{\lambda}{2} \|d - \Phi(u)\|_2^2, \tag{5.3}$$

where $\lambda > 0$ is a constant. By defining

$$J(u, d) = |d|_1 + H(u), \quad (5.4)$$

we can write (5.3) as

$$\min_{u \in X, d \in \mathbb{R}} J(u, d) + \frac{\lambda}{2} \|d - \Phi(u)\|_2^2. \quad (5.5)$$

Notice that this is the same problem that we addressed in section 3 with the iterative Bregman algorithm. We can thus solve (5.5) by using the iterative Bregman algorithm, which generates the following sequences:

$$(u^{k+1}, d^{k+1}) = \min_{u \in X, d \in \mathbb{R}} D_J^p(u, u^k, d, d^k) + \frac{\lambda}{2} \|d - \Phi(u)\|_2^2, \quad (5.6)$$

$$p_u^{k+1} = p_u^k - \lambda(\nabla\Phi)^T(\Phi(u^{k+1}) - d^{k+1}), \quad (5.7)$$

$$p_d^{k+1} = p_d^k - \lambda(d^{k+1} - \Phi(u^{k+1})). \quad (5.8)$$

Since the iterative Bregman algorithm is well defined we know that (5.6) has a minimum and that $(p_u^{k+1}, p_d^{k+1}) \in \partial J(u^{k+1}, d^{k+1})$. Expanding (5.6) we get,

$$\begin{aligned} (u^{k+1}, d^{k+1}) = \min_{u \in X, d \in \mathbb{R}} & J(u, d) - J(u^k, d^k) - \langle p_u^k, u - u^k \rangle - \langle p_d^k, d - d^k \rangle \\ & + \frac{\lambda}{2} \|d - \Phi(u)\|_2^2. \end{aligned} \quad (5.9)$$

In addition notice that,

$$p_u^{k+1} = p_u^k - \lambda(\nabla\Phi)^T(\Phi(u^{k+1}) - d^{k+1}) \quad (5.10)$$

$$= -\lambda(\nabla\Phi)^T \sum_{i=1}^{k+1} (\Phi(u^i) - d^i), \quad (5.11)$$

and

$$p_d^{k+1} = p_d^k - \lambda(d^{k+1} - \Phi(u^{k+1})) = \lambda \sum_{i=1}^{k+1} (\Phi(u^i) - d^i). \quad (5.12)$$

Define

$$b^{k+1} = b^k + (\Phi(u^{k+1}) - d^{k+1}) = \sum_{i=1}^{k+1} (\Phi(u^i) - d^i). \quad (5.13)$$

Then it follows that

$$p_u^k = -\lambda(\nabla\Phi)^T b^k \quad \forall k \geq 0, \quad (5.14)$$

and

$$p_d^k = \lambda b^k \quad \forall k \geq 0. \quad (5.15)$$

Applying (5.14) and (5.15) to (5.9) we get

$$\begin{aligned} (u^{k+1}, d^{k+1}) &= \min_{u \in X, d \in \mathbb{R}} J(u, d) - J(u^k, d^k) + \lambda \langle b^k, \Phi u - \Phi u^k \rangle \\ &\quad - \lambda \langle b^k, d - d^k \rangle + \frac{\lambda}{2} \|d - \Phi(u)\|_2^2 \\ &= \min_{u \in X, d \in \mathbb{R}} J(u, d) - J(u^k, d^k) - \lambda \langle b^k, d - \Phi u \rangle \\ &\quad - \lambda \langle b^k, d^k - \Phi u^k \rangle + \frac{\lambda}{2} \|d - \Phi(u)\|_2^2 \\ &= \min_{u \in X, d \in \mathbb{R}} J(u, d) - \lambda \langle b^k, d - \Phi u \rangle + \frac{\lambda}{2} \|d - \Phi(u)\|_2^2 + C_1 \\ &= \min_{u \in X, d \in \mathbb{R}} J(u, d) + \frac{\lambda}{2} \|d - \Phi(u) - b^k\|_2^2 + C_2, \end{aligned} \quad (5.16)$$

where C_1 and C_2 are constants. This is the Split Bregman method introduced by Goldstein and Osher [5], which we write more compactly in the following form:

Split Bregman Algorithm:

Initialize: $k=0$, $u^0 = 0$, $b^0 = 0$

while $\|u^k - u^{k-1}\|_2^2 > \text{tol}$ **do**
 $u^{k+1} = \min_u H(u) + \frac{\lambda}{2} \|d^k - \Phi(u) - b^k\|_2^2$
 $d^{k+1} = \min_d |d| + \frac{\lambda}{2} \|d - \Phi(u^{k+1}) - b^k\|_2^2$
 $b^{k+1} = b^k + (\Phi(u^{k+1}) - d^{k+1})$
 $k = k + 1$
end while

By using the idea in the Split Bregman algorithm, the Bregman algorithm can be applied to more general optimization problems of the form

$$\min_{u \in X} J(u) + \frac{1}{2} \|u - f\|_2^2. \quad (5.17)$$

We apply the Split Bregman Algorithm to the problems of anisotropic TV denoising and isotropic TV denoising.

5.1 Anisotropic TV Denoising

We first consider the Anisotropic TV Denoising problem,

$$\min_u \left\| \frac{\partial u}{\partial x} \right\|_1 + \left\| \frac{\partial u}{\partial y} \right\|_1 + \frac{\mu}{2} \|u - f\|_2^2, \quad (5.18)$$

where f is the noisy image. We will denote $\frac{\partial u}{\partial x}$ by u_x and $\frac{\partial u}{\partial y}$ by u_y . We consider the equivalent constrained problem to (5.18)

$$\min_u \|d_x\|_1 + \|d_y\|_1 + \frac{\mu}{2} \|u - f\|_2^2, \text{ such that } d_x = u_x \text{ and } d_y = u_y. \quad (5.19)$$

As in the previous section to solve (5.19) we actually solve the unconstrained version

$$\min_{u, d_x, d_y} \|d_x\|_1 + \|d_y\|_1 + \frac{\mu}{2} \|u - f\|_2^2 + \frac{\lambda}{2} \|d_x - u_x\|_2^2 + \frac{\lambda}{2} \|d_y - u_y\|_2^2. \quad (5.20)$$

Note that this is the same problem that we addressed when deriving the split Bregman algorithm. We can thus solve (5.20) by using the split Bregman algorithm

$$(u^{k+1}, d_x^{k+1}, d_y^{k+1}) = \min_{u, d_x, d_y} \|d_x\|_1 + \|d_y\|_1 + \frac{\mu}{2} \|u - f\|_2^2 + \frac{\lambda}{2} \|d_x - u_x - b_x^k\|_2^2 + \frac{\lambda}{2} \|d_y - u_y - b_y^k\|_2^2, \quad (5.21)$$

$$b_x^{k+1} = b_x^k + (u_x^{k+1} - d_x^{k+1}), \quad (5.22)$$

$$b_y^{k+1} = b_y^k + (u_y^{k+1} - d_y^{k+1}). \quad (5.23)$$

Notice that the functional being minimized in the first step is smooth with respect to u . To minimize it we simply set its first variational derivative equal to zero:

$$0 = \mu(u^{k+1} - f) - \lambda \nabla_x^T (d_x^k - u_x^{k+1} - b_x^k) - \lambda \nabla_y^T (d_y^k - u_y^{k+1} - b_y^k), \quad (5.24)$$

$$= \mu u^{k+1} - \mu f - \lambda \nabla_x^T (d_x^k - b_x^k) + \lambda u_{xx}^{k+1} + \lambda u_{yy}^{k+1} - \lambda \nabla_y^T (d_y^k - b_y^k), \quad (5.25)$$

$$\Rightarrow (\mu I + \lambda \Delta) u^{k+1} = \mu f + \lambda \nabla_x^T (d_x^k - b_x^k) + \lambda \nabla_y^T (d_y^k - b_y^k). \quad (5.26)$$

This problem is solved using Dirichlet boundary conditions. For our examples we choose Ω to be a rectangular domain, and discretize the equations using a uniform grid. We approximate the partial derivatives using second order, centered finite differences. The terms of the form $\|d\|_p^p$ are approximated by

$$\|d\|_p^p \approx \Delta x \Delta y \sum |d_{i,j}|^p.$$

Following [5], we solve the system of equations using the Gauss-Seidel iteration. Faster methods could also be used, such as the Conjugate Gradient Method, or Multigrid [6]. The Gauss-Seidel solution can be written componentwise as follows:

$$\begin{aligned}
u_{i,j}^{k+1} = G_{i,j}^k &= \frac{\lambda}{\mu + 4\lambda} (u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k + d_{x,i-1,j}^k - d_{x,i,j}^k \\
&\quad + d_{y,i,j-1}^k - d_{y,i,j}^k - b_{x,i-1,j}^k + b_{x,i,j}^k - b_{y,i,j-1}^k + b_{y,i,j}^k) \\
&\quad + \frac{\mu}{\mu + 4\lambda} f_{i,j}. \tag{5.27}
\end{aligned}$$

At the boundaries of the domain we used one-sided finite differences instead of the centered finite differences. Next we solve the problem with respect to d_x . Since the function $f(x) = |x|$ is differentiable in $\mathbb{R} \setminus \{0\}$, when we minimize with respect to d_x , we get:

$$\begin{aligned}
0 &= \begin{cases} 1 + \lambda d_x^k - \lambda u_x^{k+1} - \lambda b_x^k & d_x^k > 0 \\ 0 & d_x^k = 0 \\ 1 - \lambda d_x^k + \lambda u_x^{k+1} + \lambda b_x^k & d_x^k < 0 \end{cases} \\
\Rightarrow d_x^k &= \begin{cases} (u_x^{k+1} + b_x^k) - \frac{1}{\lambda} & (u_x^{k+1} + b_x^k) \in (\frac{1}{\lambda}, \infty) \\ 0 & (u_x^{k+1} + b_x^k) \in [-\frac{1}{\lambda}, \frac{1}{\lambda}] \\ \frac{1}{\lambda} + u_x^{k+1} + b_x^k & (u_x^{k+1} + b_x^k) \in (-\infty, -\frac{1}{\lambda}) \end{cases} \\
&= \text{shrink} \left(u_x^{k+1} + b_x^k, \frac{1}{\lambda} \right).
\end{aligned}$$

Now observe that the equation for d_y is the same as the equation for d_x , and can be obtained by replacing all the x 's by y 's. It follows then that

$$d_y^k = \text{shrink} \left(u_y^{k+1} + b_y^k, \frac{1}{\lambda} \right). \tag{5.28}$$

Therefore the Split Bregman Anisotropic TV Denoising algorithm can be written as follows:

Split Bregman Anisotropic TV Denoising:

Initialize: $k=0, u^0 = 0, b^0 = 0$

while $\|u^k - u^{k-1}\|_2^2 > \text{tol}$ **do**

$u^{k+1} = G^k$ where G is the Gauss-Seidel function defined earlier.

$d_x^{k+1} = \text{shrink}(\nabla_x u^{k+1} + b_x^k, \frac{1}{\lambda})$

$d_y^{k+1} = \text{shrink}(\nabla_y u^{k+1} + b_y^k, \frac{1}{\lambda})$

$b_x^{k+1} = b_x^k + (\nabla_x u^{k+1} - d_x^{k+1})$

$b_y^{k+1} = b_y^k + (\nabla_y u^{k+1} - d_y^{k+1})$

$k = k + 1$

end while

5.2 Isotropic TV Denoising

In the Isotropic TV model we solve the following problem:

$$\min_u \|\nabla u\|_2 + \frac{\mu}{2} \|u - f\|_2^2. \quad (5.29)$$

We rewrite this problem as a constrained problem,

$$\min_u \|(d_x, d_y)\|_2 + \frac{\mu}{2} \|u - f\|_2^2, \text{ such that } d_x = u_x \text{ and } d_y = u_y, \quad (5.30)$$

and as in the previous section we relax the constraints and solve the unconstrained problem

$$\min_{u, d_x, d_y} \|(d_x, d_y)\|_2 + \frac{\mu}{2} \|u - f\|_2^2 + \frac{\lambda}{2} \|d_x - u_x\|_2^2 + \frac{\lambda}{2} \|d_y - u_y\|_2^2. \quad (5.31)$$

Note that this is the same problem that we addressed when deriving the split Bregman algorithm. We can thus solve (5.31) by using the split Bregman algorithm

$$(u^{k+1}, d_x^{k+1}, d_y^{k+1}) = \min_{u, d_x, d_y} \|(d_x, d_y)\|_2 + \frac{\mu}{2} \|u - f\|_2^2 + \frac{\lambda}{2} \|d_x - u_x - b_x^k\|_2^2 + \frac{\lambda}{2} \|d_y - u_y - b_y^k\|_2^2, \quad (5.32)$$

$$b_x^{k+1} = b_x^k + (u_x^{k+1} - d_x^{k+1}), \quad (5.33)$$

$$b_y^{k+1} = b_y^k + (u_y^{k+1} - d_y^{k+1}). \quad (5.34)$$

The minimization problem with respect to u in the first step is the same as in the Anisotropic problem. Hence as before we let $u^{k+1} = G^k$ where G^k is the Gauss-Seidel solution given in equation (5.27). The difference between the Anisotropic problem and the isotropic problem lies in how we calculate d_x and d_y . Unlike the anisotropic problem, in the isotropic problem d_x and d_y are coupled together. Consider the minimization problem with respect to d_x ,

$$0 = \frac{d_x^{k+1}}{\|(d_x^k, d_y^k)\|_2} + \lambda(d_x^{k+1} - u_x^k - b_x^k). \quad (5.35)$$

$$(5.36)$$

Define

$$s^k = \sqrt{|u_x^k + b_x^k|^2 + |u_y^k + b_y^k|^2}. \quad (5.37)$$

We approximate $\|(d_x^k, d_y^k)\|_2$ by s^k , which leads to

$$\begin{aligned} 0 &= \frac{d_x^{k+1}}{s^k} + \lambda(d_x^{k+1} - u_x^k - b_x^k) \\ \Rightarrow d_x^{k+1} \left(\lambda + \frac{1}{s^k} \right) &= \lambda(u_x^k + b_x^k) \\ &= \frac{s^k \lambda (u_x^k + b_x^k)}{s^k \lambda + 1}. \end{aligned}$$

As in the previous algorithm d_y has the same formula as d_x expect the x 's become y 's, i.e.,

$$d_y^{k+1} = \frac{s^k \lambda (u_y^k + b_y^k)}{s^k \lambda + 1}.$$

From this we get the algorithm.

Split Bregman Isotropic TV Denoising:

Initialize: $k=0$, $u^0 = 0$, $b^0 = 0$

while $\|u^k - u^{k-1}\|_2^2 > \text{tol}$ **do**

$u^{k+1} = G^k$ where G is the Gauss-Seidel function defined earlier.

$$d_x^{k+1} = \frac{s^k \lambda (u_x^k + b_x^k)}{s^k \lambda + 1}$$

$$d_y^{k+1} = \frac{s^k \lambda (u_y^k + b_y^k)}{s^k \lambda + 1}$$

$$b_x^{k+1} = b_x^k + (u_x^{k+1} - d_x^{k+1})$$

$$b_y^{k+1} = b_y^k + (u_y^{k+1} - d_y^{k+1})$$

$$k = k + 1$$

end while

6 Numerical Results

6.1 Basis Pursuit Problem

In our numerical experiments we used random matrices generated in Matlab using `randn(m,n)`. We had Matlab generate random sparse vectors \bar{u} . We chose those vectors such that the number of nonzero entries were equal to $0.1n$ or $0.05n$ which were obtained using the routine calls `round(0.1n)` and `round(0.05n)` in MATLAB, respectively. We then defined $\mathbf{f} = \mathbf{A}\bar{\mathbf{u}}$. The stopping criterion

$$\frac{\|\mathbf{A}u^k - \mathbf{f}\|_2}{\|\mathbf{f}\|_2} < 10^{-5} \quad (6.1)$$

was used. In the following tables we show the size of the random matrix A , the l_1 norm of the original vector \bar{u} , the l_1 norm of our solution, the residual error, the number of iterations required, and the time needed to perform the algorithm:

Results of linearized Bregman- l_1 without kicking						
		$\ \bar{u}\ _1$	$\ u\ _1$	$\ Au - f\ _2$	Num. of iter.	Time required
m	n	Number of nonzero entries in \bar{u} : $0.1n$				
10	30	1.193162	1.177883	0.000084	17472	0.869771 s
50	100	5.020869	5.020832	0.000712	2334	0.144548 s
50	200	8.173409	6.962439	0.000674	4425	0.306456 s
75	150	6.774425	6.774392	0.000995	296	0.031358 s
m	n	Number of nonzero entries in \bar{u} : $0.05n$				
10	30	0.549661	0.549658	0.000030	820	0.031979 s
50	100	2.130546	2.130546	0.000484	312	0.026888 s
50	200	4.417850	4.417829	0.000808	993	0.068624 s
75	150	4.318850	4.318850	0.000762	74	0.008143 s

Results of linearized Bregman- l_1 with kicking						
		$\ \bar{u}\ _1$	$\ u\ _1$	$\ Au - f\ _2$	Num. of iter.	Time required
m	n	Number of nonzero entries in \bar{u} : $0.1n$				
10	30	1.193162	1.178426	0.000083	3970	0.219367 s
50	100	5.020869	5.020938	0.000786	1273	0.104327 s
50	200	8.173409	7.017613	0.000656	786	0.075189 s
75	150	6.774425	6.774426	0.001033	570	0.051384 s
m	n	Number of nonzero entries in \bar{u} : $0.05n$				
10	30	0.549661	0.549660	0.000032	733	0.039978 s
50	100	2.130546	2.130540	0.000338	49	0.006380 s
50	200	4.417850	4.473151	0.000860	11909	1.103843 s
75	150	4.318850	4.318843	0.000829	34	0.005658 s

From the table we can conclude that in general the linearized Bregman algorithm with kicking yields a result that is almost as good as the result yielded with the linearized Bregman algorithm but with fewer iterations. The following should be observed: Both the linearized Bregman algorithm, and the linearized Bregman algorithm with kicking are accurate with a small residual error. The linearized Bregman algorithm always returns u such that $\|u\|_1$ is smaller than $\|\bar{u}\|_1$, but in three of the eight tests the linearized Bregman algorithm with kicking yielded a result u with $\|u\|_1$ larger than $\|\bar{u}\|_1$. Furthermore observe that in most of the tests, the algorithm with kicking solved the problem in significantly less iterations.

In figures 3 and 4, we plot the residual in the corresponding iterations.

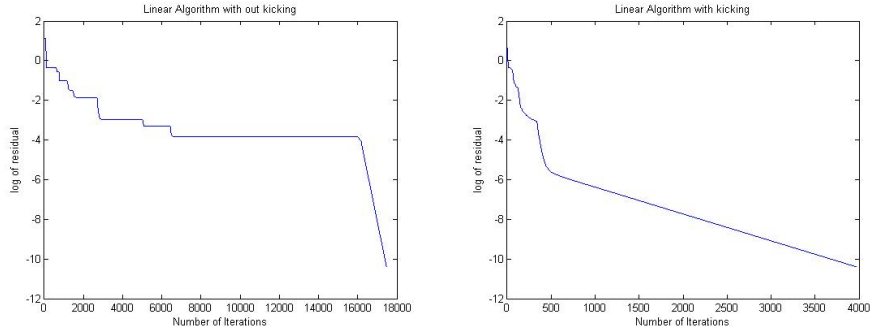


Figure 3: Linearized Bregman Algorithm, $m=10$, $n=30$, $\|u\|_0 = 0.1n$

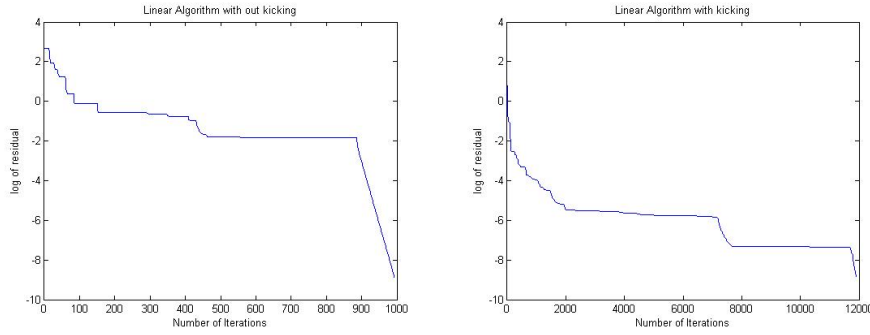


Figure 4: Linearized Bregman Algorithm, $m=50$, $n=200$, $\|u\|_0 = 0.05n$

Notice that the graphs printed in figure 3 are the graphs corresponding to the first row of both of the above tables. Also notice that the graphs printed in figure 4 are the graphs corresponding to seventh row of both of the above tables. In the first case we see that the Linearized Bregman algorithm with kicking has fewer iterations than the linearized Bregman algorithm because it successfully removes the stagnation periods. The second case shows us that when the linearized Bregman algorithm with kicking had more iterations it was because the algorithm not only failed to remove the stagnation periods but increased the number and length of the periods.

6.2 Image Denoising Problem

To test the algorithms in the TV denoising problem, we considered several images, added noise to it, and tried to recover from it the original image. We obtained our images from a list of images already available in Matlab. Suppose that X is the matrix representation of an image, of size $m \times n$. To add noise to X we created a random matrix N using the Matlab command `rand(m,n)`. We then defined our noisy image to be $f = X + cN$, where $c > 0$ is a constant. For our experiments we used $\mu = 0.1$, $\lambda = 0.2$, $c = 20$, and tolerance = 10^{-3} . In tables 1 and 2 we show the results for the Anisotropic and Isotropic TV denoising algorithms, respectively. The relative error is measured by $\|u - X\|_2^2$.

Image	Size		Anisotropic TV Denoising		
	n	m	Number of Iterations	Relative Error	Time (s)
Fluid Jet	400	300	15	0.066469	45.265365 s
Bone	367	490	13	0.050428	57.263076 s
Gatlinburg	480	640	9	0.034703	70.699119 s
Durer	648	509	6	0.024618	53.494445 s
Durer Detail	359	371	10	0.084235	33.669978 s
Cape Cod	360	360	14	0.061195	44.381875 s
Clown	200	320	11	0.086670	17.690584 s
Earth	257	250	10	0.071599	16.257617 s
Mandrill	480	500	5	0.030040	33.262180 s

Table 1: Results for the Anisotropic TV denoising algorithm

Image	Size		Isotropic TV Denoising		
	n	m	Number of Iterations	Relative Error	Time (s)
Fluid Jet	400	300	9	0.066773	8.016607 s
Bone	367	490	8	0.050665	11.391311 s
Gatlinburg	480	640	7	0.034764	17.688556 s
Durer	648	509	5	0.023862	14.446595 s
Durer Detail	359	371	9	0.080530	9.333276 s
Cape Cod	360	360	9	0.061358	9.138667 s
Clown	200	320	8	0.086626	4.106909 s
Earth	257	250	8	0.071367	4.008097 s
Mandrill	480	500	5	0.029832	10.545355 s

Table 2: Results for the Isotropic TV denoising algorithm

From the tables we see that the isotropic TV denoising is faster and just as accurate as the anisotropic denoising version of the algorithm. In our tests we found that for the fluid jet, bone, Gatlinburg, and cape cod images, the sequence of residuals converged monotonically towards a value that was lower than that of the original noisy image. We found that the relative error in the iteration associated to the clown image was oscillatory, and converged to a value higher than the value for the original noisy image. The last four images showed a decrease in the error in the first iteration, after which the errors monotonically increased and converged to a point of relative error higher than the relative error of the original noisy image. This can be seen from the following images and graphs.

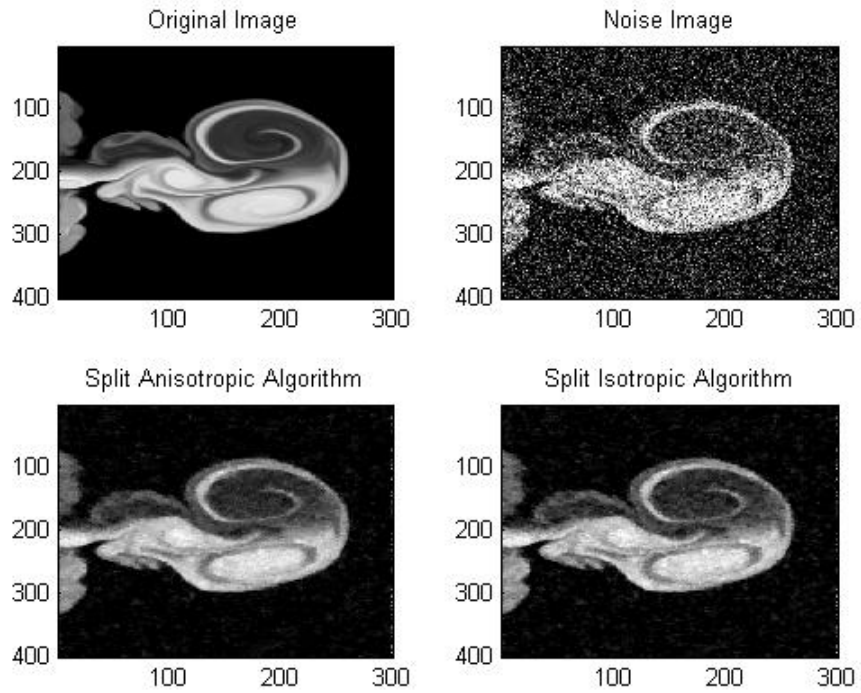


Figure 5: Split Bregman Results using Fluid Jet Image

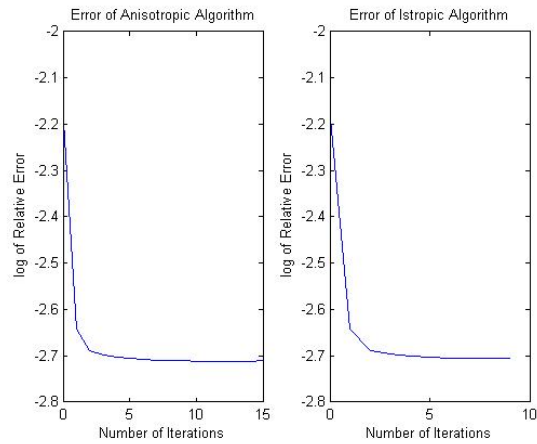


Figure 6: Split Bregman Error Results using the Fluid Jet Image

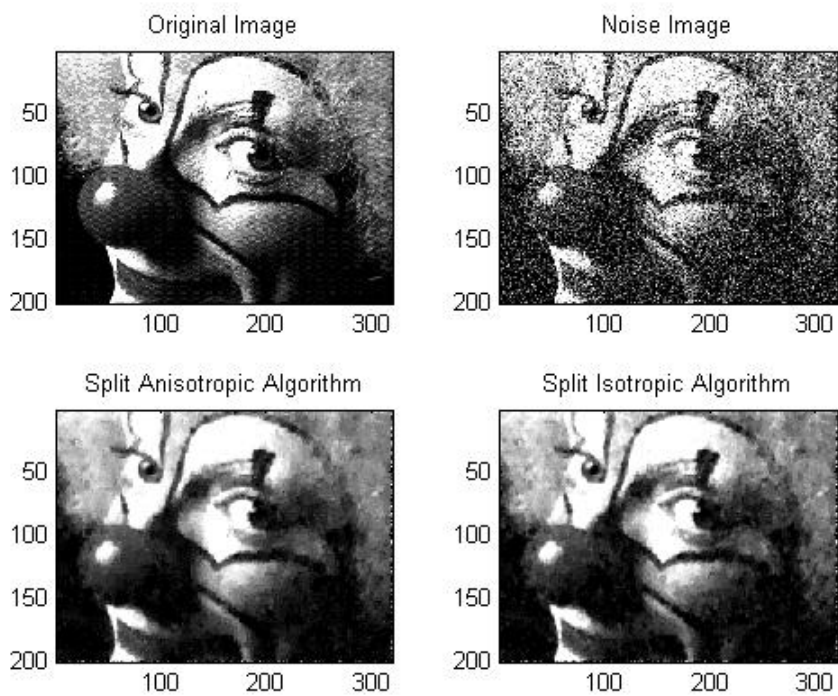


Figure 7: Split Bregman Results using Clown Image

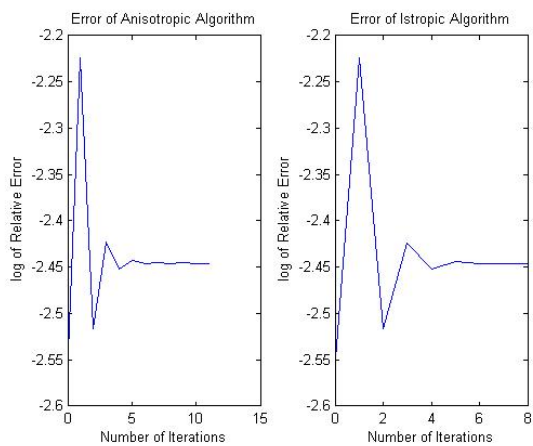


Figure 8: Split Bregman Error Results using the Clown Image

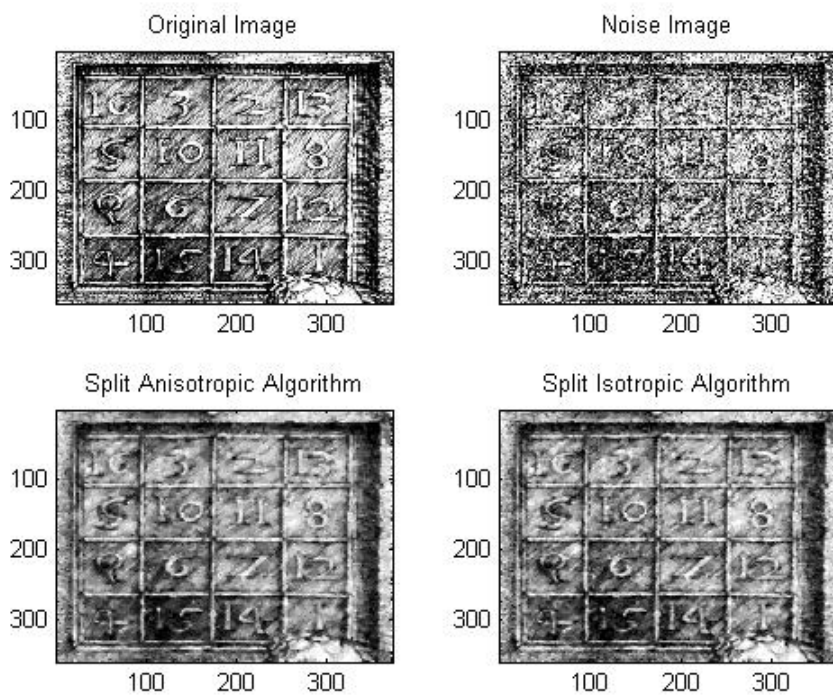


Figure 9: Split Bregman Results using Durer Detail Image

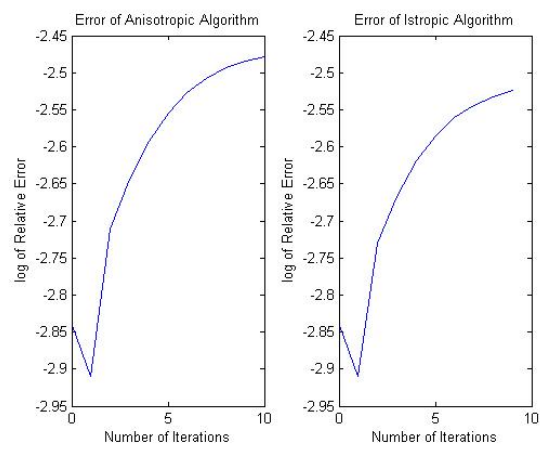


Figure 10: Split Bregman Error Results using the Durer Detail Image

We showed in section 3.1 that the relative error of the iterative Bregman algorithm decreased monotonically. However the last two images displayed in this paper show that the relative error of the Split Bregman algorithm with both isotropic and anisotropic filters does not necessarily converge monotonically. We do not currently have a satisfactory explanation for this behavior, and it would be interesting to explore it further. We would also like to see if other properties of convergence discussed in section 3.1 hold for the Split Bregman algorithm.

7 Further Study

When experimenting with these algorithm we observed several surprising results. In the linearized algorithms we noticed that occasionally the linearized algorithm without kicking performed better than the algorithm with kicking. It would be interesting to run more tests and experiment with different conditions to figure out why this happens, and if there was a way to correct this problem. In addition we would like to investigate why in two of our tests the kicking version of the linearized algorithm yielded long stagnation periods.

In the image denoising problem the logarithm of the relative error did not always converge to a point below the initial error. Five of the nine images tested had this problem. In the future we would like to run more tests and see if by changing μ and λ we can fix this problem. We would also want to investigate mathematically why the split Bregman algorithm proved not to be strictly monotonic.

8 Matlab Code

The following Matlab codes were written to implement the Linearized Bregman Iterative Algorithm, and the Linearized Bregman Iterative Algorithm with kicking.

```
function lineartest
clc; clear all;
m=50; n=200;
tic
%Calculate random matrix
A = randn(m,n);
%Code for Sparse Vector (u bar)
nz=round(0.05*n); %Number of nonzero numbers
z=n-nz; %Number of zero numbers
ub= rand(n,1); %Vector of Random Elements
b=randperm(n); %Random Permutation Vector of m elements.
ub(b(1:z))=0; %Uses the permutation vector to find z random places,
               %and sets them equal to zero.
%Calculate f
f = A*ub;

fprintf('Linear Algorithm \n')
[k,u,r] = LinAlgorithm(A,f,n);
fprintf('Number of Iterations=%f\n',k);
fprintf('||u||_1=%f\n', norm(u,1));
fprintf('||ub||_1= %f\n', norm(ub,1));
fprintf('||Au-f||_1 = %f\n', norm(A*u-f,1));
figure(1)
```

```

i=(1:k);
plot(i,r(i))
title('Linear Algorithm with out kicking')
xlabel('Number of Iterations')
ylabel('log of residual')

fprintf('Linear Algorithm with Kicking\n');
[l,k,u,r]=LinAlgKicking(A,f,n);
fprintf('Number of times going into if statement=%.0f\n',l);
fprintf('Number of Iterations=%.0f\n',k);
fprintf('||u||_1=%f\n', norm(u,1));
fprintf('||ub||_1= %f\n', norm(ub,1));
fprintf('||Au-f||_1 = %f\n', norm(A*u-f,1));
figure(2)
i=(1:k);
plot(i,r(i))
title('Linear Algorithm with kicking' )
xlabel('Number of Iterations')
ylabel('log of residual')
end

```

I used the following code to run the Anisotropic and Isotropic TV denoising codes,

```

function imagetest
    clc; clf; clear all;
clear X map;
imglist = {'flujet', ... Fluid Jet
           'spine', ... Bone

```

```

    'gatlin', ... Gatlinburg
    'durer', ... Durer
    'detail', ... Durer Detail
    'cape', ... Cape Cod
    'clown', ... Clown
    'earth', ... Earth
    'mandrill', ... Mandrill
    'spiral'};

load(imglist{8},'X','map');
c=20; %As c increases the noise in the image gets worse.
noise=randn(size(X));
f=X+c*noise;
mu=.1;lambda=.2; Tol=10^(-3);
size(X)

fprintf('Split Anisotropic Algorithm \n')
[u1,l,k]=SplitAnisotropic2(f,X, Tol,lambda, mu);
fprintf('Number of iterations %0.f \n',k-1)
p = norm(u1-X,2)/norm(X,2);
fprintf('Relative error, ||u-X||_2/||X||_2 = %f \n', p )

fprintf('Split Isotropic Algorithm \n')
[u2,ll,kk] = SplitIsotropic2(f,X,Tol,lambda,mu);
pp = norm(u2-X,2)/norm(X,2);
fprintf('Number of iterations %.0f \n', kk-1)
fprintf('Relative error, ||u-X||_2/||X||_2 = %f \n', pp)

```

```

figure(1)
colormap('gray')
subplot(2,2,1)
image(X)
title('Original Image')
subplot(2,2,2)
image(f)
title('Noise Image')
subplot(2,2,3)
image(u1)
title('Split Anisotropic Algorithm')
subplot(2,2,4)
image(u2)
title('Split Isotropic Algorithm')

figure(2)
subplot(1,2,1)
j=(1:k);
plot(j-1, log(1(j)))
xlabel('Number of Iterations')
ylabel('log of Relative Error')
title('Error of Anisotropic Algorithm')
subplot(1,2,2)
i=[1:kk];

```

```

plot( i-1, log(ll(i)))
xlabel('Number of Iterations')
ylabel('log of Relative Error')
title('Error of Istropic Algorithm')

```

8.1 Linearized Bregman

The following code implements the linear bregman iterative algorithm,

```

function [k,u,r]=LinAlgorithm(A,f,n)
tic
%Initialize
u=zeros(n,1); v = zeros(n,1);
delta=.01; mu=1000; epsilon=10^(-5);
%Algorithm
k=0;
while (norm(f-A*u,2)/norm(f,2))>epsilon %stopping criterion
k=k+1;
v = v + A'*(f-A*u);
u = delta*shrink(v,mu);
r(k) = log(norm(f-A*u,2));
end
toc
end

```

8.2 Linear Bregman with kicking

The following code implements the linear bregman iterative algorithm with kicking,

```

function [l,k,u,r]=LinAlgKicking(A,f,n)

```



```

tic

%Initialize
u=zeros(n,1); u1=zeros(n,1); v = A'*(f-A*u);
delta=.01; mu=100; epsilon=10^(-5);

%Algorithm
k=0; l=0; o=0;
while (norm(f-A*u,2)/norm(f,2))>epsilon %stopping criterion
k=k+1;
u1=u;
u = delta*shrink(v,mu);
if norm(abs(u-u1),2)<10^(-8) ;
l= l +1 ;
x=A'*(f-A*u);
for i=1:n
if abs(u(i)) < 10^(-10)
s(i,1)= ceil((mu*sign(x(i))-v(i))/x(i));
else
s(i,1) = 10^(100);
end
end
ss= min(s);
if ss==10^(100)
ss= 1;
end
for i=1:n
if abs(u(i))<10^(-10)
v(i) = v(i) + ss*x(i);

```

```

        else
            v(i) = v(i) ;
        end
    end
end

else
v = v + A'*(f-A*u);
end

r(k) = log(norm(f-A*u,2));
if k>3 & norm(abs(u-u1),2)<10^(-10)
    break
end
end
toc
end

```

8.3 Anisotropic TV Denoising

The following code implements the split Bregman Algorithm for Anisotropic TV denoising.

```

function [u,p,k]=SplitAnisotropic2(f,X,Tol,lambda,mu)
tic
n=size(f,1); m=size(f,2);
n1=n-1; m1=m-1;
u=f; dx =zeros(n,m); dy=zeros(n,m); bx=zeros(n,m); by=zeros(n,m); k=0;
nn(1) = norm(u,2); p(1) = norm(f-X,2)/norm(X,2);
while nn(k+1) >Tol
    k=k+1;
    u1 = u ;

```

```

%Calculate u.
for i=2:n1
for j=2:m1
u(i,j)= g(u,dx,dy, bx, by,f, lambda, mu, i,j);
end
end

%Compute the ds
for i=2:n1
for j=1:m
dx(i,j) = shrink((u(i+1,j)- u(i-1,j))/2+bx(i,j),1/lambda);
end
end
for j=1:m
dx(1,j) = shrink((u(1+1,j)- u(1,j))+bx(1,j),1/lambda);
end
for j=1:m
dx(n,j) = shrink((u(n,j)- u(n-1,j))+bx(n,j),1/lambda);
end
for j=2:m1
for i=1:n
dy(i,j) = shrink((u(i,j+1) - u(i,j-1))/2 +by(i,j),1/lambda);
end
end
for i=1:n
dy(i,1) = shrink((u(i,2) - u(i,1)) +by(i,1),1/lambda);
end
for i=1:n

```

```

dy(i,m) = shrink((u(i,m) - u(i,m-1)) +by(i,m),1/lambda);
end
%Calculate the b's
for i=2:n1
    for j=1:m
bx(i,j) = bx(i,j) + (u(i+1,j)-u(i-1,j))/2 - dx(i,j) ;
    end
end
    for j=1:m
bx(1,j) = bx(1,j) + (u(1+1,j)-u(1,j)) - dx(1,j) ;
    end
    for j=1:m
bx(n,j) = bx(n,j) + (u(n,j)-u(n-1,j)) - dx(n,j) ;
    end
for i=1:n
    for j=2:m1
by(i,j) = by(i,j) + (u(i,j+1)-u(i,j-1))/2 - dy(i,j) ;
    end
end
    for i=1:n
by(i,1) = by(i,1) + (u(i,1+1)-u(i,1)) - dy(i,1) ;
    end
    for i=1:n
by(i,m) = by(i,m) + (u(i,m)-u(i,m-1)) - dy(i,m) ;
    end
nn(k+1) = norm(u-u1,2)/norm(u,2);
nn(k); nn(k+1);

```

```

    p(k+1)= norm(u-X,2)/norm(X,2);
end
toc
end

```

8.4 Isotropic TV Denoising

The following matlab code implements the split Bregman algorithm for isotropic TV denoising.

```

function [u,p,k]=SplitIsotropic2(f,X,Tol, lambda,mu)
tic
n=size(f,1); m=size(f,2);
n1=n-1; m1=m-1;
u=f; dx =zeros(n,m); dy=zeros(n,m); bx=zeros(n,m); by=zeros(n,m);
k=0; p(1) = norm(f-X,2)/norm(X,2);nn(1) = norm(u,2);
while nn(k+1) >Tol
    k=k+1;
    u1 = u ;
    for i=2:n1
        for j=2:m1
            u(i,j)= g(u,dx,dy, bx, by,f, lambda, mu, i,j);
        end
    end
    %Compute the sx's
    for i=2:n1
        for j=2:m1
            s(i,j) = sqrt( abs( (u(i+1,j)-u(i-1,j))/2 + bx(i,j))^2
            + abs((u(i,j+1)-u(i,j-1))/2 + by(i,j))^2);
        end
    end
end

```

```

        end
    end
    for i=1:n1
        j=1;
        s(i,j) = sqrt( abs( (u(i+1,j)-u(i,j)) + bx(i,j))^2
            + abs((u(i,j+1)-u(i,j)) + by(i,j))^2);
    end
    for j=1:m1
        i=1;
        s(i,j) = sqrt( abs( (u(i+1,j)-u(i,j)) + bx(i,j))^2
            + abs((u(i,j+1)-u(i,j)) + by(i,j))^2);
    end
    for i=2:n
        j=m;
        s(i,j) = sqrt( abs( (u(i,j)-u(i-1,j)) + bx(i,j))^2
            + abs((u(i,j)-u(i,j-1)) + by(i,j))^2);
    end
    for j=2:m
        i=n;
        s(i,j) = sqrt( abs( (u(i,j)-u(i-1,j)) + bx(i,j))^2
            + abs((u(i,j)-u(i,j-1)) + by(i,j))^2);
    end
    for j=1:m1
        i=1;
        s(i,j) = sqrt( abs( (u(i+1,j)-u(i,j)) + bx(i,j))^2
            + abs((u(i,j+1)-u(i,j)) + by(i,j))^2);
    end
end

```

```

%Compute the d's
for i=2:n1
    for j=1:m
        dx(i,j)= (s(i,j)*lambda*((u(i+1,j)-u(i-1,j))/2+bx(i,j)))
            /(s(i,j)*lambda + 1);
    end
end
for j=1:m
    i=1;
    dx(i,j)= (s(i,j)*lambda*((u(i+1,j)-u(i,j))+bx(i,j)))
        /(s(i,j)*lambda + 1);
end
for j=1:m
    i=n;
    dx(i,j)= (s(i,j)*lambda*((u(i,j)-u(i-1,j))+bx(i,j)))
        /(s(i,j)*lambda + 1);
end
for i=1:n
    for j=2:m1
        dy(i,j) = (s(i,j)*lambda*((u(i,j+1)-u(i,j-1))/2 + by(i,j)))
            /(s(i,j)*lambda + 1 );
    end
end
for i=1:n
    j=1;
    dy(i,j) = (s(i,j)*lambda*((u(i,j+1)-u(i,j)) + by(i,j)))
        /(s(i,j)*lambda + 1 );

```

```

end
for i=1:n
    j=m;
    dy(i,j) = (s(i,j)*lambda*((u(i,j)-u(i,j-1))/2 + by(i,j))))
        /(s(i,j)*lambda +1 );
end
%Compute the b's
for i=2:n1
    for j=1:m
        bx(i,j)= bx(i,j) + ((u(i+1,j)-u(i-1,j))/2 - dx(i,j));
    end
end
for j=1:m
    i=1;
    bx(i,j)= bx(i,j) + ((u(i+1,j)-u(i,j)) - dx(i,j));
end
for j=1:m
    i=n;
    bx(i,j)= bx(i,j) + ((u(i,j)-u(i-1,j)) - dx(i,j));
end
for i=1:n
    for j=2:m1
        by(i,j) = by(i,j) + ((u(i,j+1)-u(i,j-1))/2 - dy(i,j));
    end
end
for i=1:n
    j=1;

```



```

        by(i,j) = by(i,j) + ((u(i,j+1)-u(i,j)) - dy(i,j));
    end
    for i=1:n
        j=m;
        by(i,j) = by(i,j) + ((u(i,j)-u(i,j-1)) - dy(i,j));
    end
    nn(k+1) = norm(u-u1,2)/norm(u,2);
    p(k+1)= norm(u-X,2)/norm(X,2);
end
toc
end

```

References

- [1] L. M. Brègman. A relaxation method of finding a common point of convex sets and its application to the solution of problems in convex programming. *Ž. Vyčisl. Mat. i Mat. Fiz.*, 7:620–631, 1967.
- [2] Bernard Dacorogna. *Introduction to the calculus of variations*. Imperial College Press, London, second edition, 2009. Translated from the 1992 French original.
- [3] Ivar Ekeland and Roger Témam. *Convex analysis and variational problems*, volume 28 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, english edition, 1999. Translated from the French.
- [4] Enrico Giusti. *Minimal surfaces and functions of bounded variation*, volume 80 of *Monographs in Mathematics*. Birkhäuser Verlag, Basel, 1984.
- [5] Tom Goldstein and Stanley Osher. The split Bregman method for L_1 -regularized problems. *SIAM J. Imaging Sci.*, 2(2):323–343, 2009.
- [6] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, second edition, 2009.
- [7] Kwangmoo Koh, Seung-Jean Kim, and Stephen Boyd. An interior-point method for large-scale l_1 -regularized logistic regression. *J. Mach. Learn. Res.*, 8:1519–1555 (electronic), 2007.
- [8] Stanley Osher, Martin Burger, Donald Goldfarb, Jinjun Xu, and Wotao Yin. An iterative regularization method for total variation-based image restoration. *Multiscale Model. Simul.*, 4(2):460–489 (electronic), 2005.

- [9] Stanley Osher, Yu Mao, Bin Dong, and Wotao Yin. Fast linearized Bregman iteration for compressive sensing and sparse denoising. *Commun. Math. Sci.*, 8(1):93–111, 2010.
- [10] Wotao Yin, Stanley Osher, Donald Goldfarb, and Jerome Darbon. Bregman iterative algorithms for l_1 -minimization with applications to compressed sensing. *SIAM J. Imaging Sci.*, 1(1):143–168, 2008.