A MULTILEVEL DECOMPOSITION METHOD FOR SPARSE SIGNAL RECOVERY IN A FOURIER BASIS

TOM GOLDSTEIN

ABSTRACT. We introduce a multi-level decomposition scheme for solving basis pursuit problems in a Fourier basis. The iterates generated by this scheme are equivalent to the coordinate-descent (CD) method for basis pursuit. However, unlike standard CD methods, the new algorithm computes each iterate using only $O(n \log n)$ operations. Using four different test problems, the CD algorithm is compared to two other common algorithms for basis pursuit. For the problems considered here, runtimes for the CD algorithm are approximately 5-10 times faster than conventional methods.

1. Introduction

In this manuscript, we consider a fast multi-level decomposition method for reconstructing sparse signals from samples taken in the Fourier domain. In particular, we wish to solve problems of the form

$$\min_{u \in \mathbb{R}^N} \sum_{i=0}^{N-1} \phi(u_i) + \sum_{i=0}^{N-1} \frac{\mu}{2} (R_i \mathcal{F}_N u_i - s_i)^2$$

where $\{R_i\}$ is some sequence on real numbers, $s_i$ is the observed data, $\phi$ is some regularizing function, and $\mathcal{F}_N$ represents the discrete Fourier transform operator defined on vectors of size N. This energy can be written more compactly using vector norm notation as

$$\min_{u \in \mathbb{R}^N} \Phi(u) + \frac{\mu}{2} \| R \mathcal{F}_N u - s \|^2$$

where $R$ denotes the diagonal matrix having the sequence $\{R_i\}$ as its diagonal entries, and $\| \cdot \|$ denotes the discrete 2-norm.

Problems of this form arise frequently in compressed sensing (CS), where we wish to reconstruct $u$ from a small subset of its Fourier coefficients [6, 7, 8, 13, 33, 40]. If the $k$th Fourier coefficient is known, we take $s_k$ to be the coefficient, and $R_k = 1$. If the $k$th mode is unknown, we take $s_k = R_k = 0$. Problems of this form also arise in signal processing [10], analog-to-digital conversion [41, 21] and statistical regression [37].

Another class of problems that can be represented in the form (2) are sparse deconvolution problems. Sparse deconvolution problem have the form

$$\min_u |u| + \frac{\mu}{2} \| K u - s \|^2$$

where $K$ is a convolution matrix. These problems arise, for example, in heat-source identification, seismology, and medical imaging applications [25, 27, 28, 29, 34, 19, 16]. Problems of the form (3) can be written in the form (4) by noting that
\( K = \mathcal{F}_N R \mathcal{F}_N \) for some diagonal matrix \( R \). Because of the unitary nature of the Fourier transform, we have

\[
\| Ku - s \|^2 = \| \mathcal{F}_N^* R \mathcal{F}_N u - s \|^2 = \| R \mathcal{F}_N u - \mathcal{F}_N s \|^2
\]

where the rightmost form of this energy is the same as that appearing in (4).

Many different choices of \( \phi \) have been suggested in order to enforce sparsity of the recovered signal. One of the most common choices is the L1 regularizer \( \phi(u_i) = |u_i| \). This regularizer has the advantage of being (weakly) convex, making it possible to efficiently compute a global minimizer.

Another technique for enforcing sparsity is to force the recovered signal to take on only positive values. This leads to the “non-negative least squares” technique. Non-negative least squares corresponds to problems of the form (2) where we choose

\[
\phi^+(u) = \begin{cases} 
0 & \text{if } u \geq 0 \\
\infty & \text{if } u < 0
\end{cases}
\]

The uniqueness and compressed sensing properties of non-negative least squares are studied in [5, 14].

Problems of the form (2) are traditionally solved using techniques of the gradient-descent type. This is because fast algorithms can be used to evaluate the Fourier transform of \( u \). In this manuscript, we propose to solve problems of the form (2) using a multi-level decomposition scheme which is equivalent to coordinate descent.

The organization of this paper is as follows: We first review some common techniques for basis-pursuit problems including the coordinate descent method. We then show how the coordinate descent method can be efficiently applied to problems involving the Fourier transform. We also describe how the coordinate descent method can incorporate inequality constraints (such as non-negativity) which would be difficult to incorporate into gradient based methods. Finally, we show numerical results demonstrating the efficiency of coordinate descent based methods for these problems.

2. Background

In this section we review commonly used methods for finding sparse solutions to systems of equations. These algorithms apply to problems of the form

\[
(4) \quad \min_{u \in R^N} \Phi(u) + \frac{\mu}{2} \| Au - s \|^2
\]

where \( A : R^N \rightarrow R^M \) is an arbitrary linear operator.

**FPC.** A very common approach to solving problems of the form (4) is to use a gradient-descent-based method such as Fixed-Point Continuation (FPC) [20]. This technique is based on the concept of forward-backward splitting. Algorithms of this form were originally proposed for differentiable problems by Lions and Mercier [23] and Passty [31], and later studied extensively by others [9, 26, 45]. Rigorous results for L1-regularized problems were first proposed by Hale, Yin, and Zheng in [20]. Techniques for accelerating this simple iteration scheme have also been proposed and analyzed in [3, 1], although these variants will not be considered here.

Like other forward-backward splitting techniques, FPC is a two stage algorithm that operates on some initial guess \( u^k \). During the first stage, we obtain \( u^{k+1} \) using
A gradient descent step on the differentiable term in (4).
\[ u^k = u^k - tA^T(Au^k - s). \]
During the second stage, we update the value of \( \tilde{u}^k \) by solving the “proximal” problem
\[ u_{k+1} = \arg\min_u \Phi(u) + \frac{\mu}{2t} \| u - \tilde{u}^k \| \]

The overall algorithm can be written

**Algorithm 1** Fixed-Point Continuation (FPC)

1: Initialize: \( u^0 \in R^N \)
2: for \( k = 0, 1, \cdots \) do
3: \( \tilde{u}^k = u^k - tA^T(Au^k - s) \)
4: \( u_{k+1} = \arg\min_u \Phi(u) + \frac{\mu}{2t} \| u - \tilde{u}^k \| \)
5: end for

For the problems considered here, the minimization in step 4 of the above algorithm has a simple closed form solution. For \( L^1 \) regularized problems, we take
\[ u_{k+1} = \arg\min |u| + \frac{\mu}{2t} \| u - \tilde{u}^k \| = \frac{u}{|u|} \max\{u - \frac{t}{\mu}, 0\}. \]
In the case of non-negative least squares, we take
\[ u_{k+1} = \arg\min \Phi^+ + \frac{\mu}{2t} \| u - \tilde{u}^k \| = \max\{u, 0\}. \]

Note that the FPC algorithm (1) only requires that we be able to evaluate the linear operator \( A \) and its adjoint. In case the operator \( A \) involves a Fourier transform, step 3 of the FPC algorithm can be evaluated quickly using the fast Fourier transform (FFT). This makes FPC advantageous for problems involving the Fourier transform (and other fast transforms). Another advantage of the FPC method is its extremely simple implementation.

**Orthogonal Matching Pursuit.** One way to enforce sparsity is to explicitly penalize non-zero entries using a penalty of the form
\[ \Phi^0(u) = \| u \|_0, \]
where the “\( L^0 \) norm” counts the number of non-zero entries of \( u \). Because such a regularizer is non-convex, the resulting minimization problem is not in general computationally tractable. However, it is possible to attempt to solve the problem using heuristic methods, such as orthogonal matching pursuit (OMP) [24, 35, 36, 42].

OMP is a greedy algorithm in which elements are added to the support of \( u \) one at a time until a suitable approximation to the sparse signal is reached. The algorithm is initialized with \( \text{support}(u^0) = \emptyset \). Then, a single element is added to the support of \( u^1 \) in order to minimize the resulting residual error, \( \| s - Au^1 \|_2^2 \). On each successive iteration, the support of the signal is expanded by one element, so that \( |\text{support}(u^k)| = k \). This is formalized in algorithm (2). Note that we have used \( a_i \) to denote the \( i \)th column of \( A \).
Algorithm 2 Orthogonal Matching Pursuit (OMP)

1: Initialize: $u^0 = 0$
2: $S^k = \text{support}(u^k) = \emptyset$
3: for $k = 0, 1, \cdots$ do
4: \hspace{1em} $r^k = s - Au^k$
5: \hspace{1em} for $i = 1, 2, \cdots, N$ do
6: \hspace{2em} $\epsilon_i = \min_{z \in \mathbb{R}} \|r^k - a_i z\|$
7: \hspace{1em} end for
8: \hspace{1em} Choose $i^*$ such that $\forall i, \epsilon_{i^*} \leq \epsilon_i$
9: \hspace{1em} $S^{k+1} = S^k \cup \{i^*\}$
10: \hspace{1em} $u^{k+1} = \arg \min_{\text{support}(u) \in S^{k+1}} \|s - Au\|^2$
11: \hspace{1em} end for

The outer loop in algorithm (2) is usually terminated when $\|s - Au^k\|$ falls below a preset tolerance. Although the $L_0$ penalized problem is non-convex, it has been shown that, in the context of compressed sensing problems with sufficiently sparse signals, this algorithm will find a global minimum with reasonably high probability [24, 42]. For such problems it has also been shown that the solutions to the $L_0$ penalized problem and the $L_1$ penalized problem will coincide with reasonably high probability. For this reason, the OMP algorithm can be considered as a substitute for (or at least an approximation to) the $L_1$ regularized problem.

Coordinate Descent. Fixed Point Continuation is a gradient descent based minimization technique. For general basis pursuit problems (e.g. problems not involving the Fourier transform or other fast transforms), faster methods are available. For unconstrained problems, coordinate descent (CD) is frequently the most efficient approach. These techniques work by minimizing (2) with respect to each individual element $u_i$ in sequence.

Algorithm 3 Coordinate Descent (CD)

1: Initialize: $u^0 \in \mathbb{R}^N$
2: for $k = 0, 1, \cdots$ do
3: \hspace{1em} for $i = 1, 2, \cdots, N$ do
4: \hspace{2em} $u_i^k \leftarrow \arg \min_{u_i} \Phi(u) + \frac{\mu}{2} \|Au - s\|$
5: \hspace{1em} end for
6: \hspace{1em} end for

Methods of this type were proposed very early by Fu [18], and were later popularized by Daubechies et al. [12]. Variants of this algorithm have been studied extensively for various applications including the elastic net [48, 46], non-negative least squares [4], grouped regression [47], and many other applications [22, 44, 43]. Variants of this algorithm have also been proposed for TV regularized problems (also called the “fused lasso”) by Tibshirani and others [38, 39, 2, 30]. A detailed review of many of these techniques can be found in [17].

Not only does CD have a faster convergence rate than FPC, but for dense $A$ the cost of a CD iteration is lower than the cost of an FPC iteration. These two factors make CD a superior solver when speed is the primary consideration.
Unfortunately, CD is generally not available for basis pursuit problems involving
the Fourier transform. The reason for this is that CD iterations require $O(n^2)$
computations. FPC, on the other hand, require only matrix multiplications, which
can be accomplished in $O(N \log N)$ operations using the Fast Fourier Transform.

In this manuscript, we introduce an algorithm that efficiently performs CD iter-
ations on problems of the form (2) without using any explicit Fourier transforms.
After introducing the algorithm, we will present numerical results demonstrating
that this approach outperforms conventional methods by a considerable margin for
a wide variety of applications.

3. The Cooley-Tukey FFT

One of the most efficient schemes for computing the FFT is the Cooley-Tukey
factorization [11, 15, 32]. Although this type of procedure can be performed on
vectors of arbitrary composite size, we will assume for simplicity that the signal
length is a power of 2. In this case, we get a radix 2 decimation-in-time algorithm
in which the Fourier transform of a length $N$ signal is represented as a linear
combination of smaller Fourier transforms of size $n = N/2$.

If we let $\omega_N = e^{-2\pi i/N}$ denote the $N$th root of unity and $F_N$ denote the FFT of
size $N$, then this decomposition can be written in summation form as

$$F_N u(k) = \sum_{m=0}^{N-1} u_m \omega_N^{mk} = \sum_{m=0}^{n-1} u_{2m} \omega_N^{(2m)k} + \sum_{m=0}^{n-1} u_{2m+1} \omega_N^{(2m+1)(k)}$$

$$= F_n u_e(k) + e^{-2\pi i k/N} F_n u_o(k)$$

where $u_e$ represents the even-indexed components of the $u$, and $u_o$ represents the
odd-indexed components of $u$. To be more precise

$$u_e(m) = u(2m) \quad \text{and} \quad u_o(m) = u(2m + 1).$$

This structure of this algorithm is more intuitive when it is written as a matrix
factorization. We write

$$F_n u = \begin{pmatrix} I_n & D_N \\ I_n & -D_N \end{pmatrix} \begin{pmatrix} F_n u_e \\ F_n u_o \end{pmatrix}.$$

In the above factorization, $D_N$ represents the diagonal matrix of “twiddle factors,”

$$D_{N, ii} = e^{-2\pi i k/N}$$

Using the Cooley-Tukey concept, we have written the Fourier matrix as a prod-
uct of a block diagonal “butterfly matrix,” and a set of smaller Fourier matrices.
Because of the block-diagonal structure of the butterfly matrix, multiplication by
this matrix can be performed in $O(N)$ operations. The Fourier transforms of the
smaller matrices are performed by recursively applying the Cooley-Tukey formula.
A simple induction argument shows that the total cost of computing the $N$-point
FFT using this scheme is $O(N \log N)$.

4. CD Minimization for Problems Involving the Fourier Transform

In this section, we discuss how to perform fast CD minimization for the basis
pursuit problem (2). We will assume for simplicity that the signal length is a power
of 2. Our method will work by subdividing this problem into two smaller problems,
solving these small problems, and then recombining the solutions. In this sense, the method is a divide-and-conquer algorithm much like the FFT itself.

We begin by re-writing the energy $E_n$ in terms of the even and odd-indexed components of $u$. The element-wise decoupled L1 regularizing term can be easily decomposed this way. When we perform this decomposition, the energy (2) becomes

$$E_N(u) = E_N(u_e, u_o) = |u_e| + |u_o| + \frac{\mu}{2} F(u_e, u_o)$$

where

$$F(u_e, u_o) = \|R_N u - s\|^2.$$ 

To proceed with CD minimization, we must find a mechanism to decouple the even- and odd-indexed components of $u$ in the energy (7). This is provided by the following theorem.

**Theorem.** The energy (7) can be written in the following two equivalent forms

(8) Form 1: $F(u_e, u_o) = \|R_0 F u_e - s_e\|^2 + C_e$

(9) Form 2: $F(u_e, u_o) = \|R_0 F u_o - s_o\|^2 + C_o$

where

$$R_0 = (R_1^* R_1 + R_2^* R_2)^{1/2}$$

$$s_e = R_1^* r_d^{-1} s_1 + R_2^* r_d^{-1} s_2 + (R_1^* R_2 - R_1^* R_1) r_d^{-1} D_N F_{N/2} u_o$$

$$s_o = D_N^* R_1^* R_2^* r_d^{-1} s_1 - D_N^* R_2^* R_1^* r_d^{-1} s_2 + (R_2^* R_2 - R_1^* R_1) R_1 r_d^{-1} D_N F_{N/2} u_e$$

and $C_e$ does not depend on $u_e$, while $C_o$ does not depend on $u_o$. We also have that $R_d$ is the diagonal matrix

$$(R_d)_{ii} = \begin{cases} (R_0)_{ii} & \text{if } (R_0)_{ii} \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

**Proof.** To show that (8) is equivalent to (7), we will show that both (8) and (7) have the same subgradient with respect to $u_e$. Using the Cooley-Tukey factorization (5), we can expand the energy (7) in terms of $u_e$ and $u_o$. We get

(10) $F(u_e, u_o) = \left\| \begin{pmatrix} I & D_N \\ I & -D_N \end{pmatrix} \begin{pmatrix} F_{n} u_e \\ F_{n} u_o \end{pmatrix} - \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} \right\|^2$.

Equation (10) can be written

$$F(u_e, u_o) = \|R_1 F_{N/2} u_e + R_1 D_N F_{N/2} u_o - s_1\|^2 + \|R_2 F_{N/2} u_e - R_2 D_N F_{N/2} u_o - s_2\|^2$$

We then differentiate with respect to $u_e$, and rearrange the result:

(11) $\partial_e F = F_1^* R_1^* (R_1 F_{n} u_e + R_1 D_N F_{n} u_o - s_1)$

(12) $+ F_2^* R_2^* (R_2 F_{n} u_e - R_2 D_N F_{n} u_o - s_1)$

(13) $= F_1^* (R_1^* R_1 + R_2^* R_2) F_{n} u_e + (F_1^* R_1^* R_1 D_N F_{n} - F_2^* R_2^* R_2 D_N F_{n}) u_o$

(14) $- F_1^* R_1^* s_1 - F_2^* R_2^* s_2$

We next differentiate (8), which gives us

(15) $\partial_e F = F_1^* R_0^* R_0 F_{n} u_e - F_1^* R_1^* R_1^* (R_1^* R_1 - R_2^* R_2) D_N F_{n} u_o$

(16) $- F_1^* R_0^* R_1^* R_1 s_1 - F_2^* R_0^* R_2^* R_2 s_2$
Now, note that \((R_0)_{ii} = 0 \iff (R_1)_{ii} = (R_2)_{ii} = 0\). From this, and the definition of \(R_0\) we get the following identities
\[
R_0^* R_0 = R_1^* R_1 + R_2^* R_2 \quad R_0^* R_0^{-1} R_1 = R_1 \quad R_0^* R_0^{-1} R_2 = R_2.
\]
When these identities are applied to (15-16), we see that it is equivalent to (13-14).

The proof of (9) follows from a similar argument.

We now describe how the two forms of (10) can be used to perform element-wise minimization. Suppose we first want to perform element-wise minimization on the even-indexed elements of \(u\). During this minimization step, the elements of \(u_o\) are fixed. The minimization problem we wish to solve can thus be written
\[
\arg \min_{u_e \in \mathbb{R}^n} E(u) = \arg \min_{u_e \in \mathbb{R}^n} |u_e| + |u_o| + \frac{\mu}{2} F(u_e, u_o) = \arg \min_{u_e \in \mathbb{R}^n} |u_e| + \frac{\mu}{2} F(u_e, u_o)
\]
Because \(u_o\) is held constant, we choose to expand the energy \(F\) in the form (8) above. This gives us the equivalent problem
\[
\arg \min_{u_e \in \mathbb{R}^n} |u_e| + \frac{\mu}{2} \|R_0 F u_e - s_e\|^2.
\]
Performing element-wise minimization of the energy (2) on the even index elements of \(u\) is thus equivalent to performing element-wise minimization on (4). Note that the constant factor \(C_e\) has been omitted here because it depends only on the constant vector \(u_o\), and thus has no effect of the solution of (4).

Similarly, to perform element-wise minimization of the odd-indexed components of \(u\), we need only perform a minimization sweep on the following problem of size \(n = N/2\):
\[
\arg \min_{u_o \in \mathbb{R}^n} |u_o| + \frac{\mu}{2} \|R_0 F u_o - s_o\|^2.
\]
The minimization of the small (size \(n = N/2\)) problem is performed recursively using the same decomposition that we used for problems of size \(N\). On each stage of the recursion, the problem size gets reduced by a factor of 2. The recursion terminates when the problem size has been reduced to 1, and we must solve the resulting problem analytically. The length 1 problem has the form
\[
\arg \min_{u \in \mathbb{R}} |u| + \frac{\mu}{2} \|R F_1 u - s\|^2
\]
If we note that \(F_1 = I_1\), we can see that this problem is easily solvable for many choices of \(\phi\). In particular, we have
\[
\arg \min_{u \in \mathbb{R}} |u| + \frac{\mu}{2} \|Ru - s\|^2 = \frac{x}{|x|} \max\{|x| - \frac{1}{\mu|R|^2}, 0\}.
\]

5. Implementation of the Element-Wise method

Following the arguments above, element-wise minimization on (2) can thus be achieved by the following recursive algorithm:
Algorithm 4: cdfft\(v, R, s, \mu, N\) (slow version)

1: if \(N = 1\) then
2: return \(\text{shrink}(s, 1/\mu|R|^2)\)
3: end if
4: for \(m = 0\) to \(N/2 - 1\) do
5: \(u_e(m) = u(2m)\), and \(u_o(m) = u(2m + 1)\)
6: end for
7: Let \(s_e = R_1^{-1}R_d^{-1}s_1 + R_2^{-1}R_d^{-1}s_2 + (R_2^*R_2 - R_1^*R_1)R_d^{-1}D_N\mathcal{F}_{N/2}u_o\)
8: Recursive call: \(u_e = \text{cdfft}(u_e, R_0, s_e, \mu, N/2)\)
9: \(s_o = D_N^*R_1^{-1}s_1 - D_N^*R_2^{-1}s_2 + (R_2^*R_2 - R_1^*R_1)R_d^{-1}D_N^*\mathcal{F}_{N/2}u_e\)
10: Recursive call: \(u_o = \text{cdfft}(u_o, R_0, s_o, \mu, N/2)\)
11: for \(m = 0\) to \(N/2 - 1\) do
12: \(u(2m) = u_e(m)\), and \(u(2m + 1) = u_o(m)\)
13: end for
14: return \(u\)

This algorithm is slow, however, because it requires the computation of FFT’s at each level (an FFT is involved in the definition of \(s_e\) and \(s_o\)). A close analysis of (4) shows that these FFT’s can be eliminated by operating on the Fourier transform of \(u\) rather than \(u\) itself.

In order to do this, we make the substitution \(v = \mathcal{F}_n u\). Our goal is to re-write algorithm (4) in terms of the variable \(v\) so that all computations can be done in the Fourier domain. For this purpose, we decompose \(v\) into its upper and lower halves

\[
v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}
\]

Now, using the Cooley-Tukey factorization (5), it is trivial to derive the following identities:

\[
\begin{align*}
v_e &= \mathcal{F}_n u_e = \frac{1}{2}(v_1 + v_2) \\
v_o &= \mathcal{F}_n u_o = \frac{1}{2}D_N^*(v_1 - v_2) \\
v &= \begin{pmatrix} v_e + D_N v_o \\ v_e - D_N v_o \end{pmatrix}
\end{align*}
\]

Using these identities, we can re-write the above algorithm so that we only operate in the Fourier domain:
Algorithm 5: cdfft($v, R, s, \mu, N$)

1: if $N = 1$ then
2: \hspace{1em} return $\text{shrink}(s, 1/\mu |R|^2)$
3: end if
4: $v_e = \frac{1}{2}(v_1 + v_2)$
5: $v_o = \frac{1}{2}D_N^*(v_1 - v_2)$
6: $s_e = R_1^dR_d^{-1}s_1 + R_2^dR_d^{-1}s_2 + (R_2^dR_2 - R_1^dR_1)R_d^{-1}D_Nv_o$
7: Recursive call: $v_e = \text{cdfft}(v_e, R_0, s_e, \mu, N/2)$
8: $s_o = D_N^*R_1^dR_d^{-1}s_1 - D_N^*R_2^dR_d^{-1}s_2 + (R_2^dR_2 - R_1^dR_1)R_d^{-1}D_N^*v_e$
9: Recursive call: $v_o = \text{cdfft}(v_o, R_0, s_o, \mu, N/2)$
10: $v = \begin{pmatrix} v_e + D_Nv_o \\ v_e - D_Nv_o \end{pmatrix}$
11: return $v$

The above algorithm requires no explicit FFT’s, and runs in $O(n \log N)$ operations. Almost all of the computation takes place in steps 7 and 9, where we build the vectors $s_e$ and $s_o$. Note that, while these formulas look bulky, they actually represent only a small amount of computation. Each of $s_e$ and $s_o$ are formed using a simple linear combination of 3 vectors, and the coefficients of this linear combination are precomputed only once.

6. Numerical Experiments

To demonstrate the performance of the CD algorithm, and compare it to FPC and OMP, we use four simple test problems: two problems from compressed sensing, and two deconvolution problems. These problems are summarized in table 1.

The first two test problems are conventional compressed sensing problems. We wish to recover a sparse signal from a subset of its Fourier modes. For each trial, a sparse signal of length 256 is generated by randomly choosing 5 non-zero elements of unit intensity. Problem CS1 is to recover the signal using only 32 of its 256 Fourier coefficients, chosen at random. Problem CS2 recovers the signal using 128 randomly chosen Fourier coefficients. For both problems, we enforce sparsity using an L1-regularized problem of the form (4), with $\mu = 20$. The FPC and CD iterations were stopped when the condition $\|u^{k+1} - u^k\| < 10^{-8}$ was met. The OMP algorithm was terminated with the criteria $\|Au^k - s\| < 0.1$.

The second test problem we consider is a sparse deconvolution problem. In this case, the vector $u^\ast$ represents the summation of 5 delta-function “sources” with unknown locations. The locations of these sources is chosen at random on each trial. The observed data, $s$, represents the summation of these delta functions after blurring with a Gaussian kernel. The goal of this problem is to “reverse the heat equation,” and find the location of the unknown sources. This deconvolution problem is regularized with an L1 penalty to ensure sparse results. Problem D1 is a deconvolution problem with a Gaussian kernel of variance 10 pixels. Problem D2 is blurred with a kernel of variance 0.5. Both the FPC and CD algorithms were terminated when the condition $\|u^{k+1} - u^k\| < 10^{-4}$ was met.

Results averaged over 100 trials are reported in table 2. In addition to reporting the number of iterations and total runtime (in milliseconds) for each algorithm, the
Table 1. Four Numerical Test Problems

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>Type</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS1</td>
<td>Compressed Sensing</td>
<td>32/256 Fourier Coefficients</td>
</tr>
<tr>
<td>CS2</td>
<td>Compressed Sensing</td>
<td>128/256 Fourier Coefficients</td>
</tr>
<tr>
<td>D1</td>
<td>Deconvolution</td>
<td>Gaussian Blur, $\sigma^2 = 10$</td>
</tr>
<tr>
<td>D2</td>
<td>Deconvolution</td>
<td>Gaussian Blur, $\sigma^2 = 0.5$</td>
</tr>
</tbody>
</table>

The table also identifies the number of wrong atoms (i.e. non-zero elements) detected. The OMP method is not well suited for deconvolution problems, and it was found to generate excessively large numbers of wrong atoms for these problems. For this reason, we do not report OMP results for problems D1 and D2.

Table 2. Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>Iterations</th>
<th>Runtime (ms)</th>
<th>Wrong Atoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS1</td>
<td>OMP</td>
<td>5.46</td>
<td>6.7</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>FPC</td>
<td>167</td>
<td>19.0</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>CD</td>
<td>17.6</td>
<td><strong>1.2</strong></td>
<td>0.53</td>
</tr>
<tr>
<td>CS2</td>
<td>OMP</td>
<td>4.96</td>
<td>5.5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>FPC</td>
<td>20.5</td>
<td>2.4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CD</td>
<td>8.53</td>
<td><strong>0.60</strong></td>
<td>0</td>
</tr>
<tr>
<td>D1</td>
<td>OMP</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>FPC</td>
<td>3217</td>
<td>251.7</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>CD</td>
<td>942</td>
<td><strong>55.3</strong></td>
<td>0.85</td>
</tr>
<tr>
<td>D2</td>
<td>OMP</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>FPC</td>
<td>26.4</td>
<td>2.2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CD</td>
<td>2.15</td>
<td><strong>0.18</strong></td>
<td>0</td>
</tr>
</tbody>
</table>

From the results in table 2, it is clear that one advantage of the coordinate descent (CD) method is speed. For the test problems considered here, it was observed that the CD method was approximately one order of magnitude faster than the OMP and FPC methods. Furthermore, for the ill-conditioned problem CS1, the L1-regularized methods tend to identify fewer wrong atoms then the L0 regularized OMP method. Another significant advantage of the CD method is that, unlike the FPC algorithm, it does not require the user to choose a time step and it has no stability restriction.

One notably disadvantage of the CD scheme is that its implementation is relatively complex when compared to FPC. The FPC method is extremely easy to implement in Matlab, and its implementation can take advantage of extremely well-optimized implementations of the fast Fourier transform, such as FFTW. In order to compete with such efficient codes, the CD algorithm described above must be implemented in a low-level language such as C/C++.

7. Conclusion

We introduce a multi-level decomposition scheme for solving basis pursuit problems in a Fourier basis. The iterates generated by this scheme are equivalent to
A MULTILEVEL DECOMPOSITION METHOD FOR SPARSE SIGNAL RECOVERY IN A FOURIER BASIS

Figure 1. Convergence Curves for the four problems considered. Curves show log(error) vs. iteration number for the FPC and CD algorithms. Error is defined in the L2 sense, i.e. error = \|u^k - u^*\|. The CD algorithm is depicted by the solid blue line. The dotted green line represents the FPC algorithm.

The CD algorithm is compared to two other common algorithms for basis pursuit. For the problems considered here, runtimes for the CD algorithm are approximately 5-10 times faster than conventional methods.

REFERENCES


