Exploiting Structure in a Basis Pursuit Solver

Stephen Pinto

June 6, 2015

Abstract

This document details the analysis of two solvers for the canonical basis pursuit denoising problem [CS01, Tib96] – one is a log barrier interior point solver and the other uses the Alternating Direction Method of Multipliers [SBE11, NP13]. Fitting into the broader context of applying Compressive Sampling (CS) to audio compression, the basis pursuit denoising problem in question features a large amount of structure. Exploiting this structure reduces the iteration cost in the log barrier method from a naive $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ and brings the iteration cost in the ADMM method down to $\mathcal{O}(n \log n)$. An implementation in Python showed a clear tradeoff between the log barrier method and the ADMM solver as speed vs accuracy – the ADMM solver was fast while the log barrier method was accurate.

Contents

1 Background

High-quality perceptual audio coding's commercial success changed forever the way we experience music. The high compression ratios achieved without significant audio quality degradation allowed for a massive expansion of digital audio technology into portable devices and low-bandwith applications. High-quality audio coding formats such as mp3 [BJ:90], Dolby Digital [LDFV96], MPEG Advanced Audio Coding (AAC) and other low-bit-rate audio coding formats are now found in many billions of devices at home and in commercial settings as these audio compression systems are built into DVD and other portable media players, mobile phones, TV sets, internet streaming tools, and other digital media devices.

The main approach to high-quality perceptual audio coding systems has been based on analysis/synthesis filterbanks, separating spectral components followed by a quantization/dequantization and coding process guided by models of human hearing. In the past few years, a different approach to data compression and reconstruction, Compressive Sampling (CS), has being gaining traction in a number of non-audio applications including in the fields of radar technology and medical imaging which suggests that CS may also hold promise for audio coding applications.

The CS process is based on the numerical solution to a convex optimization problem that exploits sparseness in some signal representations [CW08]. Under certain conditions, a signal that has been compressed using the CS process can be perfectly reconstructed from a compressed set of samples. CS has seen a few applications in the world of audio [CS11, COJ09, DMD$^+$13] but has not yet been applied to audio coding (at least to the author's knowledge). The author is currently working on that project with Prof. Marina Bosi at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford. This paper details the development of an Log-Barrier Interior Point solver and an Alternating Direction Method of Multipliers (ADMM) Solver for the aforementioned convex optimization problem.

2 Problem Statement

Figure 1 shows a block diagram of the overall CS coding process. In the encoding stage, overlapping blocks of an audio signal are windowed with a sine window to create a sample vector, f. The CS paradigm dictates that f be compressed to some vector $y = \Phi f$ with some sensing matrix, Φ, with orthogonal rows, where $\Phi \in \mathbf{R}^{m \times n}$, $m < n$. In this case, the rows of Φ are a randomly chosen subset of the standard basis of $\mathbf{R}^n$. This compression is augmented for this application with a perceptual matrix $H \in \mathbf{R}^{n \times n}$ having eigenvalues derived equal to the inverse of the masked threshold values [BG03] of the windowed signal to create a perceptually compressed vector y from f by

$$y = \Phi H f.$$

Represented in the frequency domain, the masked threshold indicates the necessary magnitude a signal must contain at a certain frequency for humans to hear it (the interested

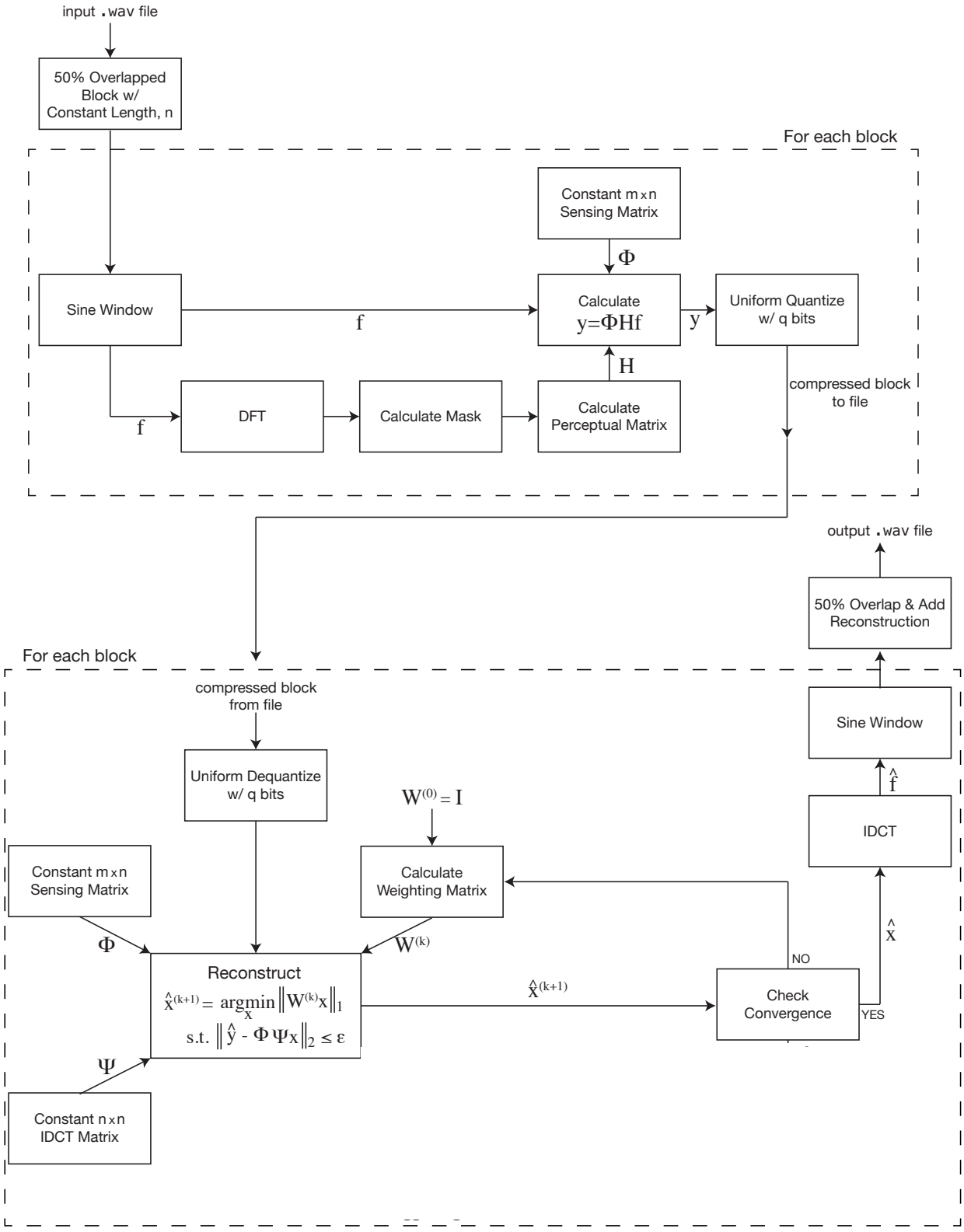

Figure 1: Overall block diagram of the whole coding and decoding system.

reader is referred to [BG03] for details on calculating the masked threshold.) As such, Hf is a vector with certain frequency components de-emphasized based on the ability of humans to hear them which, when multiplied by sensing matrix, Φ, leads to the compressed vector y. The sensing matrix does not change from block to block while the perceptual matrix, H, does. The encoder then quantizes y and writes the result to a binary file.

The decoder reads the quantized and compressed blocks, dequantizes them, and then reconstructs each block. The reconstruction process [CWB08] iteratively solves a series of convex optimization problems of the form

$$\hat{x}^{(k+1)} = \underset{x}{\operatorname{argmin}} \quad \lambda \|W^{(k)} x\|_1 + \frac{1}{2} \|\Phi\Psi x - y\|_2^2 \tag{1}$$

where

$$W^{(k)} = \operatorname{diag}\left(\frac{1}{|\hat{x}_1^{(k-1)}| + \delta}, ..., \frac{1}{|\hat{x}_n^{(k-1)}| + \delta} \right)$$

until convergence is reached (which typically requires fewer than five iterations). The sparsifying basis, Ψ, ensures that problem 1 is minimizing a representation of the block in a sparse domain - i.e. $f = \Psi x$. In this case, Ψ is the unitary IDCT matrix. The weighting constant λ is at most

$$\lambda_{max} = \|\Psi^T \Phi^y\|_\infty$$

and serves to prioritize either the sparsity (as measured by the ℓ_1 norm) or the least squares difference between the compressed vector, y, and the compressed version of the reconstruction, $\Psi\Phi\hat{x}$. Problem 1 is called the Basis Pursuit Denoising Problem [**?**]. It minimizes the sparsity of $\hat{x}$ while maintaining a certain ℓ_2 fidelity. For the first decoder block, $W^{(0)}$ is initialized as the identity matrix. In successive blocks, the iterations begin with the previous blocks final $W^{(k)}$. The sensing matrix, Φ, is the same sensing matrix used by the encoder.

Finally, the signal output from the iterative procedure is windowed again with a sine window and overlapped and added with the output of the prior block to create the full reconstructed audio signal.

Decoding a single audio file is a costly process as reconstructing each block requires solving up to five basis pursuit denoising problems and there could thousands of blocks in a single file. For example, a naive interior point solver is bounded by $\mathcal{O}(n^3)$ for each iteration – such an implementation would get nowhere fast. However, the problem has a great deal of structure – W is diagonal, Φ is sparse (each row having only a single nonzero value), and Ψ is unitary. This paper details the minimum computational complexity bound for two different solvers – a log barrier interior point solver and an ADMM solver – and shows numerical results for an implementation of each in Python.

3 Log Barrier Solver

To frame problem 1 in a context appropriate for an interior point solver, we must turn the ℓ_1 norm into an LP

$$\begin{array}{ll} \text{minimize} & \lambda \mathbf{1}^T s + (1/2)\|\Phi\Psi x - y\|_2^2 \\ \text{subject to} & \begin{bmatrix} W & -I \\ -W & -I \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} \preceq \mathbf{0}. \end{array} \tag{2}$$

Expressed as a problem with an optimization variable $z \in \mathbf{R}^{2n}$,

$$\begin{array}{ll} \text{minimize} & \lambda c^T z + (1/2)\|Pz - b\|_2^2 \\ \text{subject to} & Az \preceq \mathbf{0}, \end{array} \tag{3}$$

and expressed in cleaner notation for future reference

$$\begin{array}{ll} \text{minimize} & f_0(z) = \lambda l(z) + q(z) \\ \text{subject to} & f_1(z) \preceq \mathbf{0}. \end{array} \tag{4}$$

Algorithm 1 shows log barrier interior point method exactly as described in [BV04]. It refers to the log barrier function, which is defined as

$$\phi(z) = -\sum_{i=1}^{2n} \log\left(-a_i^T\right)$$

where a_i are the rows of A. Solving the centering step is equivalent to solving an uncon-

Algorithm 1 Barrier Method

1: **given** strictly feasible x, $t := t^{(0)} > 0$, $\mu > 1$, tolerance $\epsilon > 0$.
2: **repeat**
3: *Centering step*: $x^*(t) = \text{argmin } g(z) = tf_0(z) + \phi(z)$.
4: *Update*: $x := x^*(t)$.
5: *Stopping Criterion*: **quit** if $m/t < \epsilon$.
6: *Increase* t: $t := \mu t$.

strained minimization problem – this is accomplished with Newton's Method, described in algorithm 2. Finally, algorithm 3 describes the backtracking line search utilized in Newton's Method.

As is typical of using Newton's Method, computing

$$\nabla^2 g(z)^{-1} \nabla g(z)$$

is the most expensive operation – $\mathcal{O}(n^3)$ if the Hessian is assumed to be dense. However, analytically considering $g(z)$ and its derivatives shows significant gains can be made. The derivatives of our unconstrained objective function, $g(z)$, are

$$\nabla g(z) = t\nabla f_0(z) + \nabla\phi(z) = t\left(\lambda c + P^T P z - y^T P z\right) - \sum_{i=1}^{2n+1} \frac{1}{a_i^T z} a_i \tag{5}$$

Algorithm 2 Newton's Method

given feasible $z \in \mathbf{dom}\ g(z) = tf_0(z) + \phi(z)$, tolerance $\epsilon > 0$.
repeat
    Compute Newton Step & Decrement:
        $\Delta z_{nt} := -\nabla^2 g(z)^{-1} \nabla g(z)$; $\lambda^2 := \nabla g(z)^T \nabla^2 g(z)^{-1} \nabla g(z)$.
    Stopping Criterion: **quit** if $\lambda^2/2 < \epsilon$.
    Line Search: Choose step size t by backtracking line search.
    Update: $z := z + t\Delta z_{nt}$.

Algorithm 3 Backtracking Line Search

given a descent direction Δz for g at $z \in \mathbf{dom}\ g$, $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$.
$t := 1$.
while $g(z + t\Delta z) > g(z) + \alpha t \nabla g(z)^T \Delta z$ **do**
    $t := \beta t$.

$$\nabla^2 g(z) = t\nabla^2 f_0(z) + \nabla^2 \phi(z) = tP^T P + \sum_{i=1}^{2n+1} \frac{1}{(a_i^T z)^2} a_i a_i^T. \tag{6}$$

The most expensive part of calculating $\nabla g(z)$ is finding the product

$$P^T P z = \begin{bmatrix} Z^T Z & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix},$$

where $Z = \Phi\Psi$. Left multiplying a vector by Z costs $\mathcal{O}(n \log n)$. This is because left multiplying by Ψ is equivalent to calculating the IDCT of x ($\mathcal{O}(n \log n)$ with the FFT algorithm) and left multiplying by Φ, which is equivalent to just picking out $m < n$ elements of a vector, costs $\mathcal{O}(m)$. Similar analysis for left multiplication by Z^T yields a cost of $\mathcal{O}(n \log n)$, giving the total bound on calculating $\nabla g(z)$ as $\mathcal{O}(n \log n)$.

Inverting $\nabla^2 g(z)$ is more expensive. The matrix

$$G_a = \sum_{i=1}^{2n+1} \frac{1}{(a_i^T z)^2} a_i a_i^T$$

is rank $2n$ and tri-diagonal since W is diagonal. This means calculating G_a^{-1} and left multiplying by that inverse are both bounded as $\mathcal{O}(n)$ operations. The matrix

$$P^T P = \begin{bmatrix} Z^T Z & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} Z^T \\ 0 \end{bmatrix} \begin{bmatrix} Z & 0 \end{bmatrix} = UU^T$$

is rank m. The matrix inversion lemma, detailed in §C.4.3 of [BV04], states

$$(A + BC)^{-1} = A^{-1} - A^{-1}B(I_m + CA^{-1}B)^{-1}CA^{-1}, \tag{7}$$

where $A \in \mathbf{R}^{n\times n}$ is nonsingular, $B \in \mathbf{R}^{n\times m}$, and $C \in \mathbf{R}^{m\times n}$. Assuming A^{-1} is fast to calculate, applying equation 7 reduces the cost of calculating the inverse of the sum to $\mathcal{O}(m^3)$ – the time needed to calculate the dense $m \times m$ inverse inside the parentheses. This implies $\nabla^2 g(z)^{-1}$ is equivalent to the expression

$$(G_a + tUU^T)^{-1} = G_a^{-1} - G_a^{-1}U\left(\frac{1}{t}I_m + U^T G_a^{-1}U\right)^{-1} U^T G_a^{-1}.$$

As discussed earlier, left multiplying by Z (and thus U) and G_a^{-1} are both operations with low bounds – $\mathcal{O}(n\log n)$ and $\mathcal{O}(n)$ respectively while left multiplying by the $m \times m$ inverse in the center costs $\mathcal{O}(n^2)$. Forming and inverting the $m \times m$ matrix

$$\frac{1}{t}I_m + U^T G_a^{-1}U = \frac{1}{t}I_m + \Phi\Psi D_1 \Psi^T \Phi^T$$

is the most expensive task at hand. The diagonal matrix D_1 is the upper left $n \times n$ block diagonal matrix within G_a. Utilizing the fact that D_1 is diagonal and that multiplying by Ψ is $\mathcal{O}(n\log n)$, calculating $\Psi D_1 \Psi^T$ could be done in $\mathcal{O}(n^2 \log n)$. However, since Ψ is the orthogonal, type 3 IDCT matrix, $\Psi D_1 \Psi^T$ is, in fact, bounded as $\mathcal{O}(n^2)$. Just as $\mathcal{F}\Lambda\mathcal{F}^*$, where $\mathcal{F}$ is the unitary DCT matrix and Λ is diagonal, is a circulant matrix that corresponds to circular convolution, $\Psi D_1 \Psi^T$ also forms a known, structured matrix. Specifically, it forms a matrix corresponding to symmetric convolution which is a sum of a Toeplitz and a Hankel matrix [**?**]

$$\Psi D_1 \Psi^T = \Omega_T + \Omega_H = \frac{1}{2}\left(\begin{bmatrix} \omega_0 & \omega_1 & \dots & \omega_{N-1} \\ \omega_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \omega_1 \\ \omega_{N-1} & \dots & \omega_1 & \omega_0 \end{bmatrix} + \begin{bmatrix} \omega_1 & \omega_2 & \dots & \omega_N \\ \omega_2 & \dots & \dots & \vdots \\ \vdots & \dots & \dots & \omega_2 \\ \omega_N & \dots & \omega_2 & \omega_1 \end{bmatrix}\right)$$

where $\omega \in \mathbf{R}^{N+1}$ is the IDCT-Type 1 of $\mathbf{diag}(D_1)$ with a single zero padding the end. Forming this matrix is bounded as $\mathcal{O}(n^2)$ since tacking on the Φ and Φ^T matrices to the left and right of the expression respectively introduces a $\mathcal{O}(m^2)$ operation since Φ is sparse.

In summary, the structure in the problem allows us to drop the cost of a single Newton Iteration from the naive $\mathcal{O}(n^3)$ to the significantly faster $\mathcal{O}(n^2)$. However, alternative methods for solving problem 1, such as ADMM, do not require a matrix inversion each iteration and offer considerable speed benefits.

4 ADMM Solver

As shown by Boyd et al. in [SBE11, NP13], ADMM offers a fast method for solving problem 1. Algorithm 4 shows the general algorithm for ADMM and for this specific problem, each iteration consists of the scaled updates

$$x^{(k+1)} = \left(\Psi^T\Phi^T\Phi\Psi + \rho W^T W\right)^{-1}\left(\Psi^T\Phi^T y + \rho W^T\left(z^{(k)} - u^{(k)}\right)\right),$$

Algorithm 4 Alternating Direction Method of Multipliers

given $x^{(0)}$, $u^{(0)} = 0$, $z^{(0)} = 0$, $\rho > 0$, absolute tolerance $\epsilon_A > 0$, relative tolerance $\epsilon_R > 0$.
repeat
    x update.
    z update.
    u update.
    Calculate primal residual, r, and primal tolerance, ϵ_p.
    Calculate primal residual, s, and dual tolerance, ϵ_d.
until $\|r\|_2 < \epsilon_p$ **and** $\|s\|_2 < \epsilon_d$

$$z^{(k+1)} = S_{\lambda/\rho}\left(Wx^{(k+1)} + u^{(k)}\right),$$

and

$$u^{(k+1)} = u^{(k)} + Wx^{(k+1)} - z^{(k+1)}$$

where $S_{\lambda/\rho}$ is the soft thresholding operator

$$S_{\lambda/\rho}(\nu) = \left(\nu - \frac{\lambda}{\rho}\right)_+ - \left(-\nu - \frac{\lambda}{\rho}\right)_+,$$

λ is our ℓ_1 norm weight, and ρ is a constant with values ranging from roughly 0.1 to 100 and effects the necessary number of iterations to convergence. Each iteration also entails calculating two residuals

$$r^{(k)} = Wx^{(k)} - z^{(k)}$$
$$s^{(k)} = \rho W\left(z^{(k)} - z^{(k-1)}\right)$$

and two tolerances

$$\epsilon_p^{(k)} = \sqrt{n}\epsilon_A + \epsilon_R \, \mathbf{max}\left(\|Wx^{(k)}\|_2, \|z^{(k)}\|_2\right)$$
$$\epsilon_d^{(k)} = \sqrt{n}\epsilon_A + \epsilon_R\rho\|Wu^{(k)}\|_2$$

all of which are cheap to calculate – they are bounded by $\mathcal{O}(n)$. The x update features a $n \times n$ matrix inversion however this inversion is not iteration dependent – this means we can calculate the difficult parts of the inverse once and cache them for future use. Similarly, the constant term

$$\left(\Psi^T\Phi^T\Phi\Psi + \rho W^T W\right)^{-1}\Psi^T\Phi^T y$$

can be calculated once and cached for use each iteration. Using the same ideas from section 3, calculating the inversion is bounded as $\mathcal{O}(n^2)$ and left multiplying by the result is bounded as either $\mathcal{O}(m^2)$ or $\mathcal{O}(n \log n)$, whichever is larger. Since we don't need to calculate the actual inverse each iteration, the x updates for each iteration is bounded by either $\mathcal{O}(m^2)$ or $\mathcal{O}(n \log n)$ – much faster than the $\mathcal{O}(n^2)$ of the log barrier method. Clearly the z and u updates are both $\mathcal{O}(n)$.

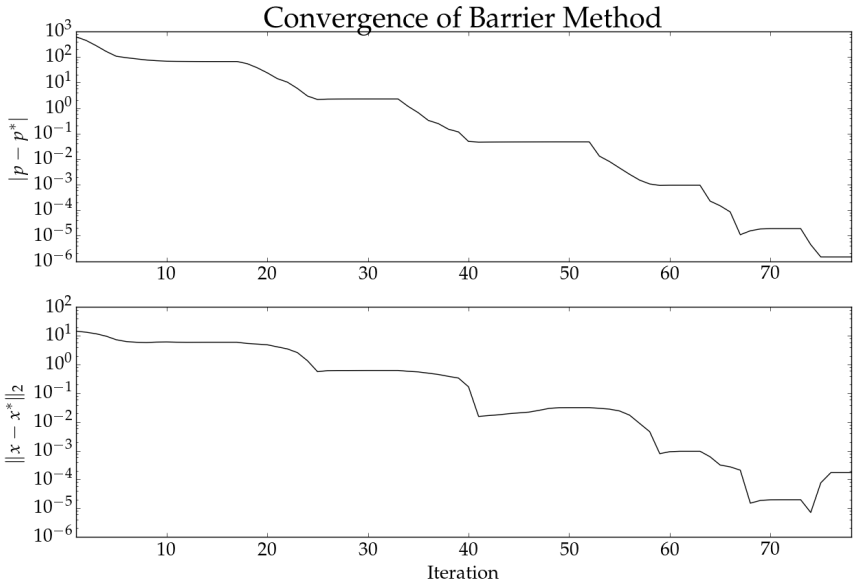

Figure 2: Convergence of the log barrier method for $n = 512$ and $m = 64$.

5 Numerical Examples

The author implemented both an log barrier interior point method and ADMM for the basis pursuit denoising of problem 1 in Python and used CVXPY [DCB14] as a verification tool. For a specific example of $n = 512$, $m = 64$, and randomly generated data, figures 2 and 3 show the convergence of the log barrier and ADMM solvers respectively. For both examples, the ℓ_1 weight was $\lambda = 0.1\lambda_{max}$. The log barrier method used $\alpha = 0.1$, $\beta = 0.5$, $\mu = 50$, $\epsilon = 10^{-5}$ while the ADMM solver used $\rho = 0.1$, $\epsilon_A = 10^{-4}$, and $\epsilon_R = 10^{-2}$.

Being assured they converge, the next step is to catalogue each of their time performances and accuracy with respect to the true x^* value, as calculated by CVXPY. The standard deviation

$$\frac{1}{n}\|x^* - x^{CVX}\|_2$$

serves as the metric of accuracy. All processes were run in Python on a 2.8 GHz i7 Apple MacBook Pro with 16 GB of RAM.

Figures 4 and 5 tell a compelling story of the tradeoff between the two algorithms – for sizeable data, ADMM is three orders of magnitude faster than the log barrier method but the log barrier method is three orders of magnitude more accurate.

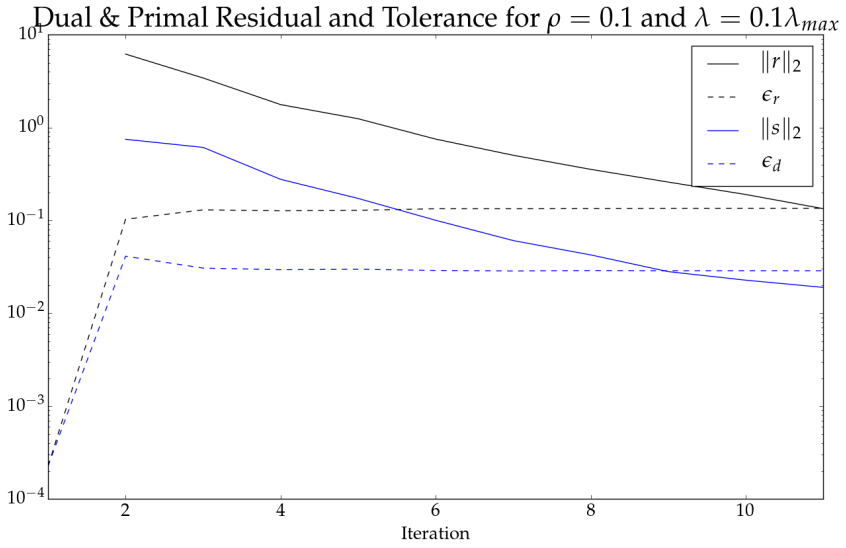

Figure 3: Convergence of ADMM for $n = 512$ and $m = 64$.

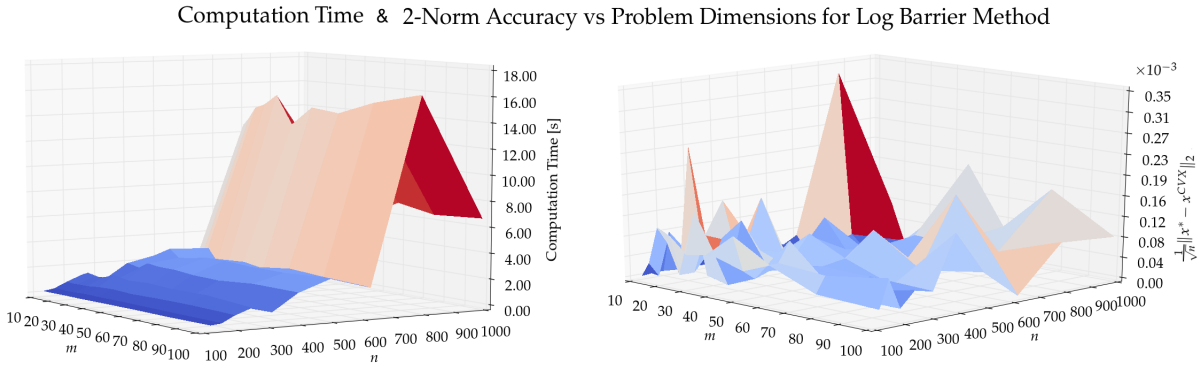

Figure 4: Computation time and ℓ_2 accuracy for the log barrier method.

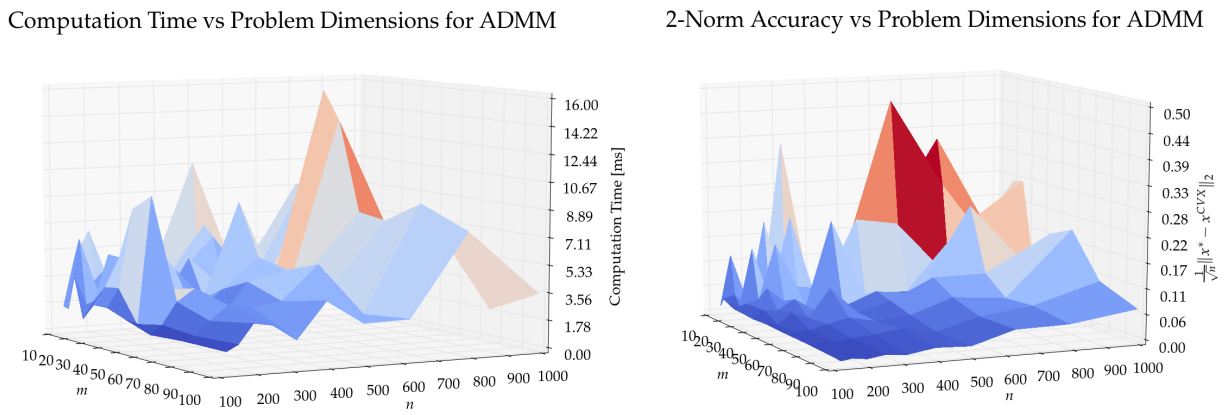

Figure 5: Computation time and ℓ_2 accuracy for ADMM.

6 Conclusion

Large amounts of structure that appear in the relatively common problem of

$$\hat{x}^{(k+1)} = \underset{x}{\text{argmin}} \quad \lambda \|W^{(k)} x\|_1 + \frac{1}{2} \|\Phi \Psi x - y\|_2^2 \tag{8}$$

allow for significant gains in speed. However, there is natural trade off between accuracy and speed for these reconstruction algorithms. In the context of audio applications, this means a tradeoff between an audio codec being tenable in terms of computation time and sounding bad to human ears. A codec taking too long to decompress is just as crippling to its success as poor fidelity. Further explorations into ways of improving the fidelity of ADMM for this solver would be worthwhile in an attempt to create a solver that is both fast and accurate.

Acknowledgments

The author would like to thank Prof. John Duchi for his help and Prof. Marina Bosi at CCRMA for her advisement on the broader project.

References

[BG03] Marina Bosi and Richard E. Goldberg. *Introduction to Digital Audio Coding and Standards*. Kluwer Academic Publishers, 2003.

[BJ:90] *88th AES Convention*, Montreux, 1990.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[COJ09] Mads Graesboll Christensen, Jan Ostergaard, and Soren Holdt Jensen. On Compressed Sensing and its Applications to Speech and Audio signals. pages 356–360, 2009.

[CS01] Donoho David Chen, Scott and Michael Saunders. Atomic Decomposition by Basis Pursuit. *Society for Industrial and Applied Mathematics Review*, 43(1):129–159, 2001.

[CS11] Mads Graesboll Christensen and Bob L. Sturm. A perceptually reweighted mixed-norm method for sparse approximation of audio signals. *Conference Record - Asilomar Conference on Signals, Systems and Computers*, pages 575–579, 2011.

[CW08] Emmanuel J. Candes and Michael B. Wakin. An Introduction To Compressive Sampling. *IEEE Signal Processing Magazine*, 25(March 2008):21–30, 2008.

[CWB08] Emmanuel J. Candes, Michael B. Wakin, and Stephen P. Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier Analysis and Applications*, 14:877–905, 2008.

[DCB14] Steven Diamond, Eric Chu, and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization, version 0.2. cvxpy.org, May 2014.

[DMD+13] B Defraene, N Mansour, S De Hertogh, T van Waterschoot, M Diehl, and M Moonen. Declipping of Audio Signals Using Perceptual Compressed Sensing. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(12):2627–2637, 2013.

[LDFV96] G. Davidson M. Davis C. Todd L. D. Fielder, M. Bosi and S. Vernon. Ac-2 and ac-3: Low-complexity transform-based audio coding, 1996.

[NP13] S. Boyd N. Parikh. Atomic Decomposition by Basis Pursuit. *Proximal Algorithms*, 1(3):123–231, 2013.

[SBE11] E. Chu B. Peleato S. Boyd, N. Parikh and J. Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[Tib96] Robert Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society*, 58(1):267–288, 1996.