Taking Advantage of Structure in an Interior Point QP Solver

Stephen Pinto

EE364b: Convex Optimization II Class Project

Introduction

Compressive Sampling (CS) is a computational technique for data compression and reconstruction which centers around a convex optimization problem. Under certain conditions, a signal that is sparse in some domain can be perfectly reconstructed from a compressed set of samples. CS is perhaps most well known for its application in the field of medical imaging. Reconstruction from the compressed set involves solving the mixed weights basis pursuit problem

$$\min_x \ \|Wx\|_1 + \lambda \|y - \Phi\Psi x\|_2^2$$

where $W \in \mathbf{R}^{n\times n}$ is diagonal, $\Phi \in \mathbf{R}^{m\times n}$ with $m < n$, and $\Psi \in \mathbf{R}^{n\times n}$ is orthogonal. Processes that involve solving a large number of these problems iteratively present a computational challenge that makes CS an untenable option. However, taking advantage of the structure present will cut back on that computation time. This project deals with an example of two common selections for Φ and Ψ – specifically Φ has rows randomly selected from the standard basis of $\mathbf{R}^n$ and Ψ is the orthogonal IDCT (AKA DCT-Type 3) matrix.

Reformulation as QP with linear constraint

The ℓ_1 norm in the objective function can be cast into the equivalent problem

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

Forming this into a $2n$ dimensional problem gives

$$\begin{array}{ll} \min\limits_{z} & \begin{bmatrix}\mathbf{0}^T & \mathbf{1}^T\end{bmatrix} z + \lambda \left(z^T \begin{bmatrix} \Psi^T\Phi^T\Phi\Psi & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix} z - \begin{bmatrix} 2y^T Z & \mathbf{0}^T \end{bmatrix} z + y^T y \right) \\ \text{subject to} & Az \preceq \mathbf{0}. \end{array}$$

This is a straightforward quadratic problem with linear constraints. As such, this project implements a log-barrier interior point method.

Log Barrier Centering Step

Labeling our objective $f_0(z)$, our $2n$ inequality constraints $f_i(z)$, and the barrier function $\phi(z)$, the centering step solves

$$\min_z \ g(z) = tf_0(z) + \phi(z) = tf_0(z) - \sum_{i=1}^{2n} \log(-f_i(z))$$

In using Newton's Method to solve each centering step, the most expensive operation is $\nabla^2 g(z)^{-1} \nabla g(z)$ with

$$\nabla g(z) = t\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} + 2\lambda \left(\begin{bmatrix} \Psi^T\Phi^T\Phi\Psi & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix} z - \begin{bmatrix} \Psi^T\Phi^T y \\ \mathbf{0} \end{bmatrix} \right)\right) - \sum_1^{2n} \frac{1}{f_i(z)} \nabla f_i(z)$$

and

$$\nabla^2 g(z) = 2t\lambda \begin{bmatrix} \Psi^T\Phi^T\Phi\Psi & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix} + \sum_1^{2n} \frac{1}{f_i(z)^2} \nabla f_i(z) \nabla f_i(z)^T$$

Inverting the Hessian

The Hessian, $\nabla^2 g(z)$ is a sum of a rank $2n$ matrix and a rank m matrix,

$$\nabla^2 g(z) = \nabla^2\phi(z) + 2t\lambda UU^T = G_z + 2t\lambda UU^T$$

where $U^T = \begin{bmatrix}\Phi\Psi & \mathbf{0}\end{bmatrix}$. The matrix inversion lemma says

$$\nabla^2 g(z)^{-1} = G_z^{-1} - G_z^{-1} U \left(\frac{1}{2t\lambda} I_m + U^T G_z^{-1} U \right)^{-1} U^T G_z^{-1}.$$

Note that G_z contains only three non-zero diagonals and can be conveniently represented as the block matrix

$$G_z = \begin{bmatrix} D_1 & D_2 \\ D_2 & D_3 \end{bmatrix}$$

where each block is a diagonal matrix. The matrix inversion lemma thus becomes

$$\nabla^2 g(z)^{-1} = G_z^{-1} - G_z^{-1} U \left(\frac{1}{2t\lambda} I_m + \Phi\Psi D_1 \Psi^T \Phi^T \right)^{-1} U^T G_z^{-1}.$$

Calculating the interior rank-m matrix, $\Psi D_1 \Psi^T$ is the single most expensive operation if considered dense but has a good deal of structure to take advantage of. Recall Ψ is the orthogonal DCT-Type 3 matrix, which diagonalizes the sum of the Toeplitz and Hankel matrices

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

where $\omega \in \mathbf{R}^{N+1}$ is the IDCT-Type 1 of $\mathbf{diag}(D_1)$ with a single zero padding the end. This means creating $\Psi D_1 \Psi^T$ is a $\mathcal{O}(n^2)$ operation instead of $\mathcal{O}(n^3)$ if assumed dense.

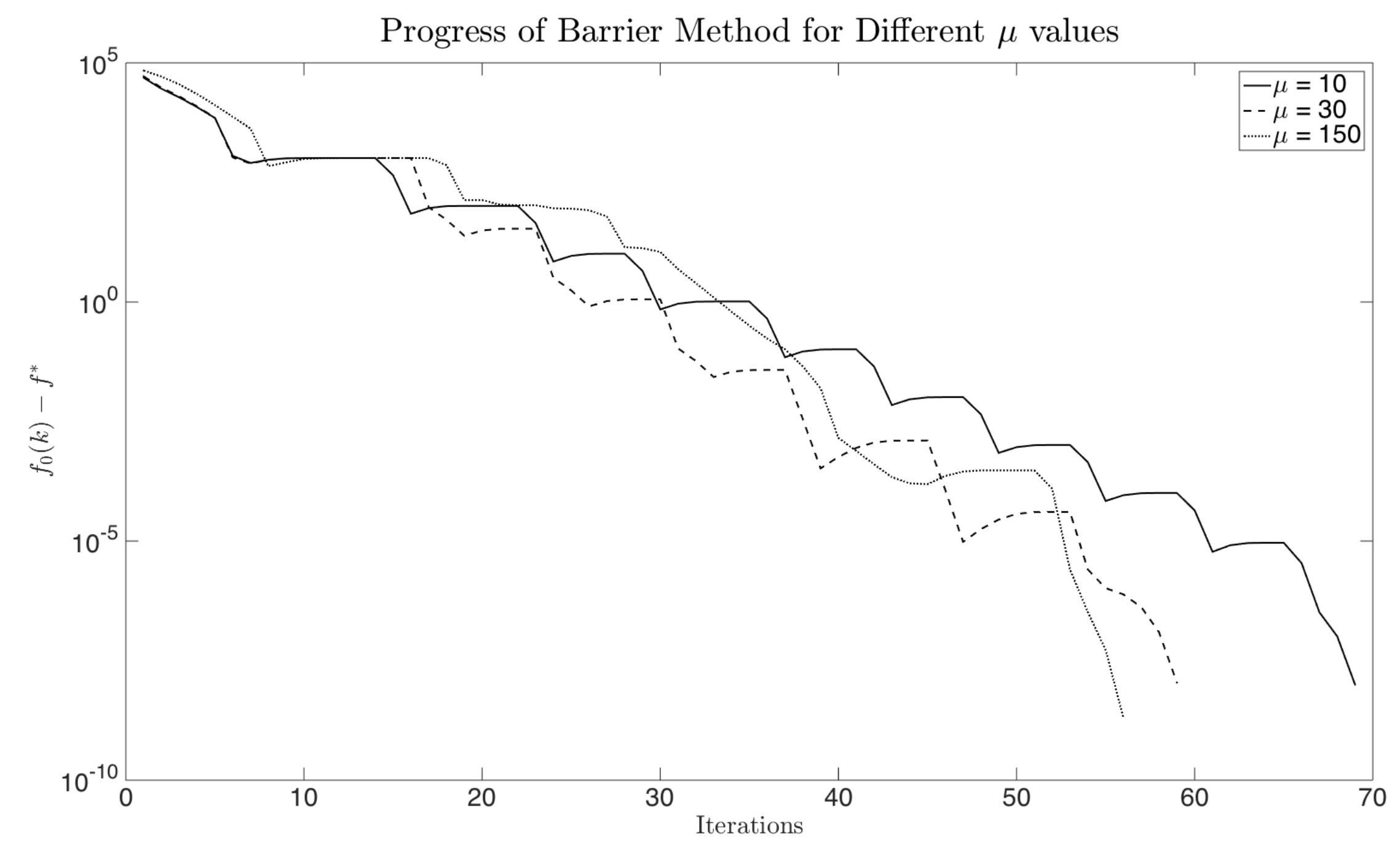
Computation Cost

The expensive product $\nabla^2 g(z)^{-1} \nabla g(z)$ ends up being bounded by the larger of $\mathcal{O}(m^3)$ or $\mathcal{O}(n^2)$.

operation	cost	explanation
Φb_n & $\Phi^T b_n$	$\mathcal{O}(m)$	rows of Φ from standard basis
Ψb_n & $\Psi^T b_n$	$\mathcal{O}(n\log n)$	FFT algorithm can calculate the DCT
find $\nabla g(z)$	$\mathcal{O}(n\log n)$	Ψz is most expensive operation
$G_z^{-1} b_{2n}$	$\mathcal{O}(n)$	G_z consists of four block diagonal matrices
find $\Psi D_1 \Psi^T$	$\mathcal{O}(n^2)$	sum of two $n \times n$ matrices
find $\Phi M \Phi^T$	$\mathcal{O}(m^2)$	creates $m \times m$ matrix via indexing M
invert $m \times m$ matrix	$\mathcal{O}(m^3)$	rank-m matrix is dense
$\nabla^2 g(z)^{-1} b_{2n}$	$\mathcal{O}(m^2)$	mult. by rank-m inv is most expensive

Implementation & Results

I implemented the solver in Python and used CVXPY as a verification tool. For randomly generated data with $n = 512$ and $m = 64$ the solver converges as shown below



For a few values of n and m, the following table lists computation time of a single Newton Iteration in seconds

n	m	without $\Omega_T + \Omega_H$	with $\Omega_T + \Omega_H$
2048	256	0.65	0.1
1024	256	0.14	0.03
512	256	0.03	0.008
512	128	0.03	0.008
256	64	0.004	0.002
256	32	0.004	0.002

and computation time of the entire problem in seconds

n	m	without $\Omega_T + \Omega_H$	with $\Omega_T + \Omega_H$	CVXPY
2048	256	too long	8.7	10.4
1024	256	8.61	2.97	6.12
512	256	1.61	0.79	2.6
512	128	1.97	0.78	0.57
256	64	0.38	0.14	0.07
256	32	0.27	0.16	0.05

Conclusion

It seems the key to getting significant speed gains is to calculate $\Psi D_1 \Psi^T$ as a sum of two matrices. All told the speed gains over CVXPY are modest at best for larger dimensions. Implementing the solver in C or C++ would probably give significant speed gains over this Python implementation.

Acknowledgements

This material fits within a larger project advised by Prof. Marina Bosi at CCRMA.