Efficient Basis Pursuit DeNoising via Active Sets and Homotopy

Suely P. OLIVEIRA [a] David E. STEWART [b]
[a] *Department of Computer Science, University of Iowa*
[b] *Department of Mathematics, University of Iowa*

Abstract. The Basis Pursuit DeNoising problem is a variant of the well-known lasso method for obtaining sparse least-squares approximations, which can be represented as the minimization problem $\min_{\boldsymbol{\beta}} \frac{1}{2} \|A\boldsymbol{\beta} - \mathbf{y}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1$. In this paper the problem is treated as a parametric quadratic program with parameter $\lambda \geq 0$. By this technique, very large scale problems can be essentially solved exactly for a range of values of λ, as long as the number of non-zero components of $\boldsymbol{\beta}$ is small to modest.

Keywords. Basis pursuit denoising, active set method, homotopy method

1. Introduction

The Basis Pursuit DeNoising (BPDN) problem is a convex optimization problem related to the lasso [21] that can be used to identify the a sparse solution to a problem of representing given data. This problem generally has the form

$$\min_{\boldsymbol{\beta}} \frac{1}{2} \|A\boldsymbol{\beta} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1, \tag{1}$$

with variants where there is a constraint on either $\|A\boldsymbol{\beta} - \mathbf{y}\|_2$ or $\|\boldsymbol{\beta}\|_1$. This problem is related to ideas involved in "compressed sensing" [6] and sparse representation or approximation [4]. By contrast, the lasso is the problem

$$\min_{\boldsymbol{\beta}} \|A\boldsymbol{\beta} - \boldsymbol{y}\|_2 \quad \text{subject to } \|\boldsymbol{\beta}\|_1 \leq \tau. \tag{2}$$

The lasso and BPDN problems are closely related in that $\boldsymbol{\beta}$ is a solution for (1) if and only if it is a solution of (2) for an appropriate value of τ. The computational methods described here complement the methods of Osborne, Presnell and Turlach [15] for the lasso, which rely on a homotopy approach for the τ parameter. Related algorithms for related problems can be found in [7,16].

Computationally, the challenge is with very large data sets where A is $m \times n$ with mn so large that the explicit dense matrix A does not fit into main memory. For such large problems, the matrix and its transpose are typically represented as functions $\boldsymbol{x} \mapsto A\boldsymbol{x}$ and $\boldsymbol{z} \mapsto A^T\boldsymbol{z}$ respectively [1] as are used for large-scale numerical methods for solving systems of equations, such as conjugate gradient methods, LSQR [18], GMRES [19], QMR [11], etc. For small BPDN problems, standard quadratic programming (QP) methods [14, Chap. 16] can be used. For large problems, most researchers use projected gradient type or other 1st order methods [3,17]. These methods take a considerable number of iterations, but the cost per iteration is relatively small. Some researchers try to reduce the computational cost of these methods by reducing the number of variables via dual optimization problems [8], which is known as "screening" the variables in $\boldsymbol{\beta}$. Newton methods can converge rapidly, at least in terms of the number of iterations, but the cost of a single step can be prohibitive.

BPDN is an application of parametric quadratic programming [9] and has been extended and refined [2,13] with [10] dealing with degeneracy issues in the context of ℓ_1 terms. This paper, however, provides a complete and efficient algorithm for BPDN problems provided the active set $\mathcal{I} := \{\, i \mid \beta_i \neq 0 \,\}$ is of modest size, $|\mathcal{I}| \sim 10^3$, say, on current architectures. This algorithm is an active set algorithm and uses a homotopy in λ. Note that this active set refers to the variables that are active rather than to active constraints. The algorithm is exact, and not iterative, giving the solution as a piecewise linear function of $\lambda \in [\lambda_{min}, \infty)$. Moreover, each step involves multiplication by A (once) and multiplication by A^T (once), and the solution of a single symmetric positive definite $|\mathcal{I}| \times |\mathcal{I}|$ linear system, plus an additional $\mathcal{O}(|\mathcal{I}| + m + n)$ floating point operations. Experimental results suggest that the number of steps is close to $|\mathcal{I}|$, at least where $|\mathcal{I}|$ is modest. The linear system solve has a cost of $\mathcal{O}(|\mathcal{I}|^3)$ if done independently, but QR or Cholesky updating and downdating [12] can reduce the cost per step for solving the linear system to $\mathcal{O}(|\mathcal{I}|^2)$. This indicates an overall cost of $\mathcal{O}(|\mathcal{I}|^3 + |\mathcal{I}|\,(m+n))$ floating point operations plus the cost of $\mathcal{O}(|\mathcal{I}|)$ matrix–vector multiplications. That is, the algorithm presented can efficiently handle some quite large BPDN problems.

Another way of viewing the computational complexity of the algorithm is to look at the number of "passes" that are needed over a given data set. Every multiplication by A or A^T is essentially a pass over the data, and so it is the number of matrix-vector multiplications that is to be minimized. Thus the cost of the algorithm is $\mathcal{O}(|\mathcal{I}|)$ passes over the data.

2. Algorithm for BPDN

The homotopy active set algorithm presented in this paper tracks $\mathcal{I} = \{\, i \mid \beta_i \neq 0 \,\}$, $\lambda > 0$ and $\boldsymbol{\beta}$ which minimize (1). Note that for each $i \in \mathcal{I}$,

$$(A^T(A\boldsymbol{\beta} - \boldsymbol{y}))_i + \lambda \operatorname{sign} \beta_i = 0.$$

For an interval in which the $s_i = \operatorname{sign}\beta_i$ does not change as we change λ, we get a system of equations

$$A_{\mathcal{I}}^T A_{\mathcal{I}} \boldsymbol{\beta}_{\mathcal{I}} = (A^T \boldsymbol{y})_{\mathcal{I}} - \lambda \boldsymbol{s}_{\mathcal{I}} \tag{3}$$

where $A_{\mathcal{I}}$ is the matrix formed by the columns $A_{\bullet i}$ for $i \in \mathcal{I}$, and $\boldsymbol{u}_{\mathcal{I}}$ consists of the components u_i where $i \in \mathcal{I}$ for any vector $\boldsymbol{u}$. Provided the columns $A_{\bullet i}$ for $i \in \mathcal{I}$ are linearly independent, $A_{\mathcal{I}}^T A_{\mathcal{I}}$ is invertible and so $\boldsymbol{\beta}_{\mathcal{I}}$ is uniquely determined by (3). Furthermore, under these conditions $\boldsymbol{\beta}_{\mathcal{I}}$ (and thus $\boldsymbol{\beta}$) is affine in λ. So under the assumption that any set of k columns of A is linearly indpendent, $\boldsymbol{\beta} = \boldsymbol{\beta}(\lambda)$ is a piecewise affine function of λ.

The task then is to find the "break-points" of $\boldsymbol{\beta}(\lambda)$. We can start with λ large and positive; if $\lambda \geq \left\|A^T\boldsymbol{y}\right\|_\infty$ then $\boldsymbol{\beta}(\lambda) = 0$. The first break-point is then at $\lambda_0 = \left\|A^T\boldsymbol{y}\right\|_\infty$. At this point, the active set changes: for the i where $\left|(A^T\boldsymbol{y})_i\right| = \left\|A^T\boldsymbol{y}\right\|_\infty$, we add i to $\mathcal{I}$ as λ is reduced below λ_0.

As λ is reduced further and crosses more break-points, indexes are added to $\mathcal{I}$ or removed from $\mathcal{I}$ with each break-point. For example, if $i \in \mathcal{I}$ for $\lambda_{k+1} < \lambda < \lambda_k$, and the affine function $\beta_i(\lambda) \to 0$ as $\lambda \downarrow \lambda_{k+1}$ we remove i from $\mathcal{I}$. If $j \notin \mathcal{I}$ for $\lambda_{k+1} < \lambda < \lambda_k$ and $\left|(A^T(A\boldsymbol{\beta}(\lambda_{k+1}) - \boldsymbol{y}))_j\right| = \lambda_{k+1}$ then we add j to $\mathcal{I}$.

At each stage there is the possibility that at a break-point there is more than one candidate i to remove from $\mathcal{I}$ or more than one candidate j to remove from $\mathcal{I}$, or both possibilities occur at a particular break-point. In such cases, the question of which i to add or j to remove can be resolved through a degeneracy resolution procedure. This can be done using a formal expansion replacing $A^T\boldsymbol{y}$ by $A^T\boldsymbol{y}+\epsilon\boldsymbol{\zeta}_1+\epsilon^2\boldsymbol{\zeta}_2+\cdots$ and considering the perturbed problem as $\epsilon \downarrow 0$ with suitable (usually random) choices for $\boldsymbol{\zeta}_1$, $\boldsymbol{\zeta}_2$, $\ldots$. These $\boldsymbol{\zeta}_k$ vectors are only generated when needed. With a random choice of $\boldsymbol{\zeta}_1$, the probability that the tie is not broken is zero. In the rare case that the tie is not broken by this choice, we need to generate $\boldsymbol{\zeta}_2$ and the probability that this then does not break the tie is again zero. Once $\boldsymbol{\zeta}_1$ vector is generated, it must be stored to ensure consistency with later steps. This approach to degeneracy resolution has been used in other problems, such as linear programming. The perturbation approach to handling degeneracy in linear programming is due to Charnes [5].

The full algorithm is shown in [20].

3. Numerical results

The algorithm was implemented in Matlab^{TM} in two forms; one taking A as an explicit matrix, and another which accepts functions representing $\mathbf{x} \mapsto A\mathbf{x}$ and $\mathbf{w} \mapsto A^T\mathbf{w}$. This latter form is particularly useful for large-scale problems as memory is needed for neither A nor A^T.

3.1. Random matrices

There are artificially generated problems in which the matrix A is a pseudo-randomly generated matrix; if the underlying pseudo-random number generator produced independent random values, the entries of the $m \times n$ matrix A would be independent and uniformly distributed according to the standard normal distribution. A sparse vector $\widehat{\mathbf{x}} \in \mathbb{R}^n$ is also generated as follows: for a given value of r the set $\mathcal{S} = \{\, i \mid \widehat{x}_i \neq 0 \,\}$ is first generated pseudo-randomly from the set of subsets of $\{1, 2, \ldots, n\}$ of size r, and the entries $\widehat{x}_i$, $i \in \mathcal{S}$ set to independent normally distributed values with the standard mean (zero) and variance (one). The vector $\mathbf{y} = A\widehat{\mathbf{x}} + \boldsymbol{\epsilon}$ where the entries ϵ_i are independently and normally distributed values with mean zero and variance σ^2.

The above algorithm was applied and stopped when $|\mathcal{I}_k| = r$. The resulting sets $\mathcal{I}_k$ were then compared with $\mathcal{I}$, and the norm $\left\|\boldsymbol{\beta}^k - \widehat{\mathbf{x}}\right\|_\infty$ computed. The value $\sigma = 1 \times 10^{-2}$ for the test results shown in Table 1. The actual number of iterations is $2k$.

The Matlab^{TM} profiler was used to identify where most of the time was spent in these tests overall: about 77% of the time spent in or for the BPDN function was actually spent on the matrix–vector multiplication functions $\mathbf{z} \mapsto A\mathbf{z}$ and $\mathbf{z} \mapsto A^T\mathbf{z}$.

test #	m	n	k	$\lvert\mathcal{I}_k \Delta \mathcal{S}\rvert$	$\lVert\boldsymbol{\beta}^k - \widehat{\mathbf{x}}\rVert_\infty$	time (s)
1	100	300	10	2	0.0437	0.0418
2	100	300	10	3	0.0097	0.0198
3	100	300	10	1	0.0202	0.0199
4	100	300	10	0	0.0114	0.0177
5	100	300	10	4	0.3191	0.0313
6	300	100	10	0	0.0067	0.0172
7	300	100	10	0	0.0042	0.0172
8	300	100	10	1	0.0047	0.0199
9	300	100	10	0	0.0051	0.0188
10	300	100	10	0	0.0090	0.0170
11	1000	3000	50	1	0.0063	0.7212
12	1000	3000	50	0	0.0164	0.7261
13	1000	3000	50	0	0.0070	0.6874
14	1000	3000	50	2	0.0040	0.7237
15	1000	3000	50	1	0.0057	0.6931
16	3000	1000	50	1	0.0026	0.7099
17	3000	1000	50	1	0.0023	0.7321
18	3000	1000	50	0	0.0024	0.7118
19	3000	1000	50	2	0.0020	0.7317
20	3000	1000	50	2	0.0019	0.7324

Table 1. Results for random matrix problems

3.2. Identifying letters

Two problems were created based on the task of identifying letters against a noisy and cluttered background. An image was created by the following steps: letters were chosen at random and their bitmaps scaled and added to an empty image after translating by random amounts in both x and y-coordinates. After N letters were added to the image in this way, scaled independent pseudo-random Gaussian noise was added. The two tasks are: (1) identify the locations of a given letter in the image, and (2) identify type and location of up to M of the letters in the image. For the specific test, the image is 150×150 monochromatic pixels, and the letters are bold Helvetica (the "a" is 15×15 pixel bitmap) generated from the Xfig drawing tool and exported as JPEG files, read into Matlab^{TM} via the function `imread()`. A total of $4 \times 26 = 104$ letters are selected at random and translated by a random amount using uniform probability distributions chosen to ensure that the entire letter remains within the image. However, letters can and do overlap significantly. A typical example is shown in Figure 1.

The dimension of $\boldsymbol{\beta}$ for finding a single letter is essentially the number of translations that can be performed on a letter, which is close to $150^2 = 22\,500$ for a 150×150 pixel image. For the task of identifying some letter from the image, the dimension of $\boldsymbol{\beta}$ must be multiplied by the number of letters, giving a dimension of $\approx 26 \times 150^2 = 585\,000$. For locating a given letter, the matrix A is represented by a function that forms a two-dimensional convolution of $\boldsymbol{\beta}$ with the bitmap of the letter to locate. This convolution is computed via Fast Fourier Transforms (FFTs); the matrix A^T is represented by a similar function computed via FFTs. For locating and identify-

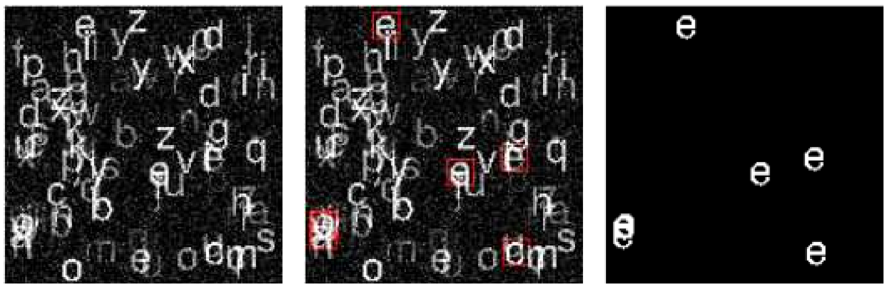
(a) Test image (b) Boxed letters (c) Identified letters

Figure 1. Identification of a given letter

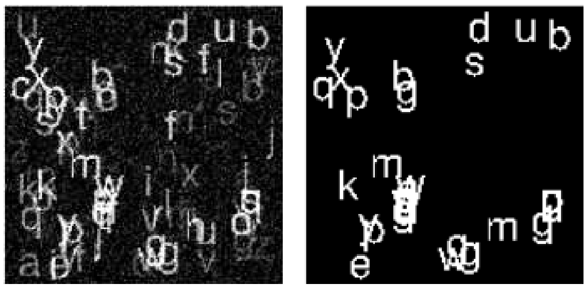
(a) Test image (b) Identified letters

Figure 2. Identification of 25 out of 100 letters in image

ing letters, $\boldsymbol{\beta}$ is formed by $\boldsymbol{\beta}^{(p)}$ where p represents a letter. The matrix A is then given by $A\boldsymbol{\beta} = \sum_{p=1}^{26} A^{(p)}\boldsymbol{\beta}^{(p)}$ where $A^{(p)}$ is the matrix representing convolution with the bitmap for the pth letter.

The first task to identify the locations of a single given letter was used as a test problem: eight positions are identified for a given letter. This is clearly more than the number of occurrences of the letter in the image (which is four), but allows for mis-identification of other letters. The mean time for this task for the letters in randomly generated 150×150 images was about 63ms in Matlab. As a method of letter identification in cluttered environments, there are better methods than raw BPDN, but it does achieve good accuracy where letters are not overlapping.

The last task was identification of 100 letters in similar 150×150 images; the time taken to perform this over 10 images ranged from 11.97s to 12.15s with a mean time of 12.06s per image. Figure 2 shows a typical example of the performance of the method; clearly there are difficulties in applying BPDN directly to this problem where multiple letters overlap.

4. Conclusion

An exact method for solving the BPDN as a parametric quadratic program is given, that includes resolution of degenerate linear constraints. This method is capable of handling quite large problems since only small linear systems need to be solved, provided the data matrix A in $\min_{\boldsymbol{\beta}} \frac{1}{2} \|A\boldsymbol{\beta} - \mathbf{y}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1$ and its transpose can be efficiently represented by functions. This algorithm was applied to synthetic data and to the problem of identifying letters in a cluttered environment. While the algorithm performs reasonably well on this task, it is far from being the best at this particular task. But it should be noted that with an efficient implementation, it is possible to exactly solve BPDN problems with up to $\approx 5 \times 10^5$ variables in less than 15 seconds in Matlab.

References

[1] Richard Barrett, Michael Berry, Tony F. Chan, and et al. *Templates for the solution of linear systems: building blocks for iterative methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.

[2] M. J. Best. An algorithm for parametric quadratic programming. In H. Fischer, B. Riedüller, and S. Schäffler, editors, *Applied Mathematics and Parallel Computing—Festschrift for Klaus Ritter*, pages 57–76. Physica-Verlag, Heidelburg, 1996.

[3] Jian-Feng Cai, Stanley Osher, and Zuowei Shen. Convergence of the linearized Bregman iteration for ℓ_1-norm minimization. *Math. Comp.*, 78(268):2127–2136, 2009.

[4] Emmanuel J. Candes and Terence Tao. Near-optimal signal recovery from random projections: universal encoding strategies? *IEEE Trans. Inform. Theory*, 52(12):5406–5425, 2006.

[5] A. Charnes. Optimality and degeneracy in linear programming. *Econometrica*, 20:160–170, 1952.

[6] David L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52(4):1289–1306, 2006.

[7] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Ann. Statist.*, 32(2):407–499, 2004. With discussion, and a rejoinder by the authors.

[8] Laurent El Ghaoui, Vivian Viallon, and Tarek Rabbani. Safe feature elimination in sparse supervised learning. *Pac. J. Optim.*, 8(4):667–698, 2012.

[9] R. Fletcher. A general quadratic programming algorithm. *Journal of the Institute of Mathematics and its Applications*, 7:76–91, 1971.

[10] R. Fletcher. On Wolfe's method for resolving degeneracy in linearly constrained optimization. *SIAM Journal on Optimization*, 24(3):1122–1137, 2014.

[11] Roland W. Freund and Noël M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60(3):315–339, 1991.

[12] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders. Methods for modifying matrix factorizations. *Math. Comp.*, 28:505–535, 1974.

[13] N. I. M. Gould. An algorithm for large-scale quadratic programming. *IMA Journal of Numerical Analysis*, 11(3):299–324, 1991.

[14] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition, 2006.

[15] M. R. Osborne, Brett Presnell, and B. A. Turlach. A new approach to variable selection in least squares problems. *IMA J. Numer. Anal.*, 20(3):389–403, 2000.

[16] M. R. Osborne and B. A. Turlach. A homotopy algorithm for the quantile regression lasso and related piecewise linear problems. *J. Comput. Graph. Statist.*, 20(4):972–987, 2011.

[17] 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.

[18] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8:43–71, 1982.

[19] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 7:856–869, 1986.

[20] David E. Stewart and Suely P. Oliveira. Basis pursuit denoising as a parametric quadratic program. Under preparation, 2018.

[21] Robert Tibshirani. Regression shrinkage and selection via the lasso. *J. Roy. Statist. Soc. Ser. B*, 58(1):267–288, 1996.