

GRADIENT POLYTOPE FACES PURSUIT FOR LARGE SCALE SPARSE RECOVERY PROBLEMS

Aris Gretsistas, Ivan Damnjanovic and Mark D. Plumbley

Queen Mary University of London
Centre for Digital Music
Mile End Road, E1 4NS, London, UK

ABSTRACT

Polytope Faces Pursuit is a greedy algorithm that performs Basis Pursuit with similar order complexity to Orthogonal Matching Pursuit. The algorithm adds one basis vector at a time and adopts a path-following approach based on the geometry of the polar polytope associated with the dual Linear Program. Its initial implementation uses the method of Cholesky factorization to update the solution vector at each step, which can be computationally expensive for solving large scale problems as it requires the successive storage of large matrices. In this paper, we present a different approach using directional updates to estimate the solution vector at each time. The proposed method uses the gradient descent method, reducing the memory requirements and computational complexity. We demonstrate the application of this Gradient Polytope Faces Pursuit algorithm to a source separation problem.

Index Terms— Sparse representation, greedy algorithm, Polytope Faces Pursuit, Gradient descent, Source Separation.

1. INTRODUCTION

Many Signal Processing applications ranging from compressed sensing to source separation require that the signal of interest has been initially transformed in a domain, where it can be expressed as a linear combination of a small number of significant coefficients. This signal expansion is known as a *sparse representation* and the obtained transformed signal \mathbf{s} of length N is said to be K -sparse, when it is expressed by only K non-zero entries, with $K \ll N$. More specifically, we suppose that the given observed signal or vector $\mathbf{y} = [y_1, \dots, y_N]^T$, which we need to decompose, can be accurately represented in a basis \mathbf{A} (known as a dictionary). The sparse representation problem is then formulated:

$$\mathbf{y} = \mathbf{A}\mathbf{s} \quad (1)$$

where \mathbf{A} is a full rank $M \times N$ matrix and \mathbf{s} is the unknown sparse sequence of coefficients. The basis \mathbf{A} can be either a fixed transformation matrix (FFT, DCT, wavelets etc.) or a dictionary that has been identified using some dictionary learning technique. In the special case where $M < N$, i.e. the number of basis vectors is larger than the dimensionality of the observed signals space, we refer to \mathbf{A} as an overcomplete dictionary. The resulting system of linear equations is said to be an underdetermined system, as the number of unknowns or variables is larger than the number of equations. Such a system yields an infinite number of solutions. From all of the existing solutions, we are interested in obtaining the sparsest one, namely the vector with the fewest possible non-zero components. Put another way, we need to solve the optimization problem

$$\min_{\mathbf{s}} \|\mathbf{s}\|_0 \quad \text{such that} \quad \mathbf{y} = \mathbf{A}\mathbf{s} \quad (2)$$

where $\|\cdot\|_0$ is the ℓ_0 norm, but this problem is known to be combinatorial i.e. NP-hard. Alternatively, according to the method of *Basis Pursuit* [1] one can attempt to solve the convex optimization problem:

$$\min_{\mathbf{s}} \|\mathbf{s}\|_1 \quad \text{such that} \quad \mathbf{y} = \mathbf{A}\mathbf{s}. \quad (3)$$

The solution to this problem can be obtained using Linear Programming (LP), e.g. interior-point methods.

Nevertheless, the convergence to the solution using the Basis Pursuit (BP) method is rather slow and approximate greedy algorithms such as *Matching Pursuit* (MP) [2] and *Orthogonal Matching Pursuit* (OMP) [3] can be used instead. These algorithms will find the sparsest solution in certain cases. Our approach to this problem is based on the polytope geometry of the dual linear program, which is equivalent to problem (3). Therefore, the so-called method of *Polytope Faces Pursuit* (PFP) [4] will perform Basis Pursuit in a greedier and faster way.

In this paper, we propose an implementation of the Polytope Faces Pursuit algorithm by following a gradient descent based approach for the estimation of the pseudo-inverse matrix which is used to update the solution vector at each step

This research is supported by ESPRC Leadership Fellowship EP/G007144/1 and Platform Grant EP/045235/1, and EU FET-Open Project FP7-ICT-225913 “SMALL”.

of the algorithm. Therefore, we overcome the storage and memory limitations of our earlier Cholesky-based algorithm [4]. Subsequently, we develop an efficient and fast greedy algorithm suitable for large-scale sparse recovery problems. In the following sections we will review the existing Polytope Faces Pursuit algorithm and develop the new gradient descent implementation. The performance of the algorithm is tested on problems generated using the benchmarking toolbox SPARCO [5], including an audio source separation problem and the results will then be compared with the earlier version of Stagewise Polytope Faces Pursuit (SPFP) [6] and the Basis Pursuit Solver.

2. SPARSE RECOVERY VIA POLYTOPE FACES PURSUIT

For the Polytope Faces Pursuit method we convert the traditional ℓ_1 -minimization problem to its *standard form* using nonnegative coefficients

$$\min_{\tilde{\mathbf{s}}} \mathbf{1}^T \tilde{\mathbf{s}} \quad \text{such that} \quad \mathbf{y} = \tilde{\mathbf{A}} \tilde{\mathbf{s}}, \tilde{\mathbf{s}} \geq \mathbf{0} \quad (4)$$

where $\mathbf{1}$ is a column vector of ones, $\tilde{\mathbf{A}} = [\mathbf{A}, -\mathbf{A}]$ and $\tilde{\mathbf{s}}$ is the $2N$ nonnegative vector

$$\tilde{s}_i = \begin{cases} \max(s_i, 0) & 1 \leq i \leq N \\ \max(-s_{i-N}, 0) & N+1 \leq i \leq 2N. \end{cases} \quad (5)$$

The above linear program has a corresponding *dual linear program*

$$\max_{\mathbf{c}} \mathbf{y}^T \mathbf{c} \quad \text{such that} \quad \tilde{\mathbf{A}}^T \mathbf{c} \leq \mathbf{1} \quad (6)$$

which has an optimum solution \mathbf{c}^* associated with the optimum solution \mathbf{s}^* of (4). Thus, we can initially look for a solution \mathbf{c}^* to (6) and use the Karush-Kuhn-Tucker (KKT) [7] conditions to solve the resulting system for \mathbf{s}^* . The full Polytope Faces Pursuit algorithm is given in [4] and illustrated in Algorithm 1.

As it has been mentioned in [6], the most computationally expensive task of this algorithm is the calculation of the Moore-Penrose pseudo-inverse $(\tilde{\mathbf{A}}^k)^\dagger$, and therefore the estimation of the solution vector $\tilde{\mathbf{s}}^k$ and the corresponding \mathbf{c}^k at each iteration. A similar problem arises in many greedy algorithms as MP and OMP. Most of these adopt a Cholesky factorization method to compute the pseudoinverse. According to this method, which has been used in our earlier PFP implementation, if \mathbf{A} is a full column rank matrix then its pseudoinverse is given by $\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T = (\mathbf{R}^T \mathbf{R})^{-1} \mathbf{A}^T$, where \mathbf{R} is an upper triangular matrix [8]. Although this method can be very efficient for small-scale problems it requires storage of the upper triangular matrix \mathbf{R} , which grows in size at each stage by one column corresponding to the selected atom of the dictionary. This storage requirement for \mathbf{R} can be undesirable for large-scale problems, as it increases

Algorithm 1 Polytope Faces Pursuit (original version)

- 1: Input: $\tilde{\mathbf{A}} = [\tilde{\mathbf{a}}_i]$, \mathbf{y} {If required, set $\tilde{\mathbf{A}} \leftarrow [\mathbf{A}, -\mathbf{A}]$ }
 - 2: Set stopping conditions l_{\max} and θ_{\min}
 - 3: Initialize: $k \leftarrow 0$, $\mathcal{I}^k \leftarrow \emptyset$, $\tilde{\mathbf{A}}^k \leftarrow \emptyset$, $\mathbf{c}^k \leftarrow \mathbf{0}$, $\tilde{\mathbf{s}}^k \leftarrow \emptyset$, $\hat{\mathbf{y}}^k \leftarrow \mathbf{0}$, $\mathbf{r}^k \leftarrow \mathbf{y}$
 - 4: **while** $|\mathcal{I}^k| < l_{\max}$ and $\max_i \tilde{\mathbf{a}}_i^T \mathbf{r}^{k-1} > \theta_{\min}$ **do** {Find next face}
 - 5: $k \leftarrow k + 1$
 - 6: Find face:
 $i^k \leftarrow \arg \max_{i \notin \mathcal{I}^{k-1}} \{(\tilde{\mathbf{a}}_i^T \mathbf{r}^{k-1}) / (1 - \tilde{\mathbf{a}}_i^T \mathbf{c}^{k-1}) \mid \tilde{\mathbf{a}}_i^T \mathbf{r}^{k-1} > 0\}$
 - 7: Optionally: $\lambda^k \leftarrow (1 - \tilde{\mathbf{a}}_i^T \mathbf{c}^{k-1}) / (\tilde{\mathbf{a}}_i^T \mathbf{r}^{k-1})$
 - 8: Add constraints:
 $\tilde{\mathbf{A}}^k \leftarrow [\tilde{\mathbf{A}}^{k-1}, \mathbf{a}_{i^k}]$, $\mathcal{I}^k \leftarrow \mathcal{I}^{k-1} \cup \{i^k\}$
 - 9: $\tilde{\mathbf{s}}^k \leftarrow (\tilde{\mathbf{A}}^k)^\dagger \mathbf{x}$
 - 10: **while** $\tilde{\mathbf{s}}^k \not\geq \mathbf{0}$ **do** {Release retarding constraints}
 - 11: Select some $j \in \mathcal{I}^k$ such that $\tilde{s}_j^k < 0$; remove column \mathbf{a}_j from $\tilde{\mathbf{A}}^k$
 - 12: Update: $\mathcal{I}^k \leftarrow \mathcal{I}^k \setminus \{j\}$, $\tilde{\mathbf{s}}^k \leftarrow (\tilde{\mathbf{A}}^k)^\dagger \mathbf{y}$
 - 13: **end while**
 - 14: $\mathbf{c}^k \leftarrow (\tilde{\mathbf{A}}^k)^\dagger \mathbf{1}$, $\hat{\mathbf{y}}^k \leftarrow \tilde{\mathbf{A}}^k \tilde{\mathbf{s}}^k$, $\mathbf{r}^k \leftarrow \mathbf{y} - \hat{\mathbf{y}}^k$
 - 15: **end while**
 - 16: Output: $\mathbf{c}^* = \mathbf{c}^k$,
 $\tilde{\mathbf{s}}^* \leftarrow \mathbf{0} + \text{corresponding entries from } \tilde{\mathbf{s}}^k$
{If required, get $s_i^* \leftarrow (\tilde{s}_i^* - \tilde{s}_{i+n}^*)$, $1 \leq i \leq n$ }
-

memory requirements that usually affect the convergence speed of the algorithm. To overcome these Cholesky factorization limitations we can use a gradient descent method, which does not require the storage of any matrices.

3. GRADIENT POLYTOPE FACES PURSUIT

The Gradient Descent or Steepest Descent method is an iterative method that can be used to solve the linear system of equations $\mathbf{A}\mathbf{s} = \mathbf{y}$, when \mathbf{A} is a symmetric, positive definite and real $N \times N$ matrix. However, the problem we wish to solve is unsymmetric and the matrix $\tilde{\mathbf{A}}$ is non-square. Blumensath and Davies [9] has shown that this is possible, deriving a gradient descent implementation of Orthogonal Matching Pursuit. Therefore, in a similar manner we can adopt a gradient descent based approach to estimate $\tilde{\mathbf{s}}$ iteratively, by calculating the new direction and then updating the solution vector.

Let us now derive the gradient descent Polytope Faces Pursuit algorithm. At the k -th step of PFP, the algorithm should have selected k atoms from the given overcomplete dual dictionary $\tilde{\mathbf{A}}$ and removed $0 \leq l < k$ atoms. Thus, in order to estimate the coefficients vector we need to solve the non square system $\mathbf{y} = \tilde{\mathbf{A}}^k \tilde{\mathbf{s}}^k$, where $\tilde{\mathbf{A}}^k$ is an $M \times (k-l)$ matrix. Instead of inverting the matrix, we can attempt to minimize to cost function $\frac{1}{2} \mathbf{s}^T \tilde{\mathbf{A}}^T \tilde{\mathbf{A}} \tilde{\mathbf{s}} - (\tilde{\mathbf{A}} \tilde{\mathbf{s}})^T \mathbf{y}$, which is equivalent to solving the system $\tilde{\mathbf{A}}^T \tilde{\mathbf{A}} \tilde{\mathbf{s}} = \tilde{\mathbf{A}}^T \mathbf{y}$. Consequently, the al-

gorithm at the next iteration will move towards the direction \mathbf{d}^k :

$$\tilde{\mathbf{s}}^k = \tilde{\mathbf{s}}^k + \alpha^k \mathbf{d}^k. \quad (7)$$

The step size α^k can be calculated using the following equation:

$$\alpha^k = \frac{(\mathbf{r}^k)^T \tilde{\mathbf{A}}^k \mathbf{d}^k}{(\tilde{\mathbf{A}}^k \mathbf{d}^k)^T \tilde{\mathbf{A}}^k \mathbf{d}^k}. \quad (8)$$

In order to estimate the corresponding \mathbf{c}^k for the current solution vector $\tilde{\mathbf{s}}^k$ we follow a different strategy. It can be shown that we can estimate basis vertex \mathbf{c} at step k using the iterative formula $\mathbf{c}^k = \mathbf{c}^{k-1} + \lambda^k (\mathbf{r}^{k-1} - \mathbf{r}^k)$, where λ^k is the atom selection criterion given by the following equation:

$$\lambda^k = \arg \min_{\mathbf{a}_i \notin \tilde{\mathbf{A}}^k} \frac{1 - \mathbf{a}_i^T \mathbf{c}^{k-1}}{\mathbf{a}_i^T \mathbf{r}^{k-1}}. \quad (9)$$

To see this, we can calculate convenient update formulae for several of the quantities involved in the PFP algorithm, including the quantities $\mathbf{B} \triangleq \mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$ and $\mathbf{P} \triangleq \mathbf{A}^\dagger \mathbf{A} = \mathbf{A} (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$. Using $\mathbf{A}^k = [\mathbf{A}^{k-1} \mid \mathbf{a}^k]$, it is straightforward to show that

$$\mathbf{B}^k = \left[\frac{\mathbf{B}^{k-1} (\mathbf{I} - \mathbf{a}^k (\mathbf{b}^k)^T)}{(\mathbf{b}^k)^T} \right]$$

where $(\mathbf{b}^k)^T = (\mathbf{I} - \mathbf{P}^{k-1}) \mathbf{a}^k / |(\mathbf{I} - \mathbf{P}^{k-1}) \mathbf{a}^k|^2$, provided that $(\mathbf{I} - \mathbf{P}^{k-1}) \mathbf{a}^k \neq \mathbf{0}$. Furthermore, under the same condition, we can also show that $\mathbf{r}^k = \mathbf{r}^{k-1} - \mathbf{b}^k (\mathbf{a}^k)^T \mathbf{r}^{k-1} = (\mathbf{I} - \mathbf{b}^k (\mathbf{a}^k)^T) \mathbf{r}^{k-1}$ and $\mathbf{c}^k = \mathbf{c}^{k-1} + \mathbf{b}^k (1 - \mathbf{a}^k \mathbf{c}^{k-1})$. Therefore for the switching in point for \mathbf{a}^k , we find that the current point on the path through the polytope is given by $\mathbf{h} = \mathbf{c}^k + \lambda^k \mathbf{r}^k = \mathbf{c}^{k-1} + \lambda^k \mathbf{r}^{k-1}$ and hence $\mathbf{c}^k - \mathbf{c}^{k-1} = \lambda^k (\mathbf{r}^{k-1} - \mathbf{r}^k)$.

Using this iterative formula for the estimation of the basis vertex \mathbf{c}^k we can gain additional computational savings, as we no longer need to update the vector \mathbf{c} by solving the computationally expensive linear system $\mathbf{1} = (\tilde{\mathbf{A}}^k)^T \mathbf{c}$. The resulting algorithm of Gradient Polytopes Faces Pursuit (GFPF) is summarized in Algorithm 2.

Comparing the new version of GFPF with the existing Stepwise Polytope Faces Pursuit (Algorithm 1) one can notice that the release of the retarding constraints, has also been omitted in this implementation. Due to the iterative nature of the gradient descent algorithm the switching out of the negative $\tilde{\mathbf{s}}^k$'s will add computational cost to the algorithm as every time a release occurs the algorithm will have to reestimate the current solution vector starting at $k = 0$. According to the standard implementation of PFP this step is necessary to provide exact sparse solutions. However, there may be circumstances where less sparse solutions are acceptable if these allow for faster convergence. Thus, GFPF algorithm stands as a faster alternative to PFP algorithm that can approximate the BP solution.

Algorithm 2 Gradient Polytope Faces Pursuit

- 1: Input: $\tilde{\mathbf{A}} = [\tilde{\mathbf{a}}_i]$, \mathbf{y} {If required, set $\tilde{\mathbf{A}} \leftarrow [\mathbf{A}, -\mathbf{A}]$ }
 - 2: Set stopping conditions l_{\max} and θ_{\min}
 - 3: Initialize: $k \leftarrow 0$, $\mathcal{I}^k \leftarrow \emptyset$, $\tilde{\mathbf{A}}^k \leftarrow \emptyset$, $\mathbf{c}^k \leftarrow \mathbf{0}$, $\tilde{\mathbf{s}}^k \leftarrow \emptyset$, $\hat{\mathbf{y}}^k \leftarrow \mathbf{0}$, $\mathbf{r}^k \leftarrow \mathbf{y}$, $\alpha^k \leftarrow 0$, $\mathbf{d}^k \leftarrow \mathbf{0}$
 - 4: **while** $|\mathcal{I}^k| < l_{\max}$ and $\max_i \tilde{\mathbf{a}}_i^T \mathbf{r}^{k-1} > \theta_{\min}$ **do** {Find next face}
 - 5: $k \leftarrow k + 1$
 - 6: Find face:
 $i^k \leftarrow \arg \max_{i \notin \mathcal{I}^{k-1}} \{(\tilde{\mathbf{a}}_i^T \mathbf{r}^{k-1}) / (1 - \tilde{\mathbf{a}}_i^T \mathbf{c}^{k-1}) \mid \tilde{\mathbf{a}}_i^T \mathbf{r}^{k-1} > 0\}$
 - 7: $\lambda^k \leftarrow (1 - \tilde{\mathbf{a}}_{i^k}^T \mathbf{c}^{k-1}) / (\tilde{\mathbf{a}}_{i^k}^T \mathbf{r}^{k-1})$
 - 8: Add constraints:
 $\tilde{\mathbf{A}}^k \leftarrow [\tilde{\mathbf{A}}^{k-1}, \mathbf{a}_{i^k}]$, $\mathcal{I}^k \leftarrow \mathcal{I}^{k-1} \cup \{i^k\}$
 - 9: $\mathbf{d}^k \leftarrow (\tilde{\mathbf{A}}^k)^T \mathbf{r}^{k-1}$
 - 10: $\alpha^k \leftarrow ((\mathbf{r}^k)^T \tilde{\mathbf{A}}^k \mathbf{d}^k) / ((\mathbf{d}^k)^T (\tilde{\mathbf{A}}^k)^T \tilde{\mathbf{A}}^k \mathbf{d}^k)$
 - 11: $\tilde{\mathbf{s}}^k \leftarrow \tilde{\mathbf{s}}^{k-1} + \alpha^k \mathbf{d}^k$, $\mathbf{r}^k \leftarrow \mathbf{r}^{k-1} - \alpha^k \tilde{\mathbf{A}}^k \mathbf{d}^k$
 - 12: $\mathbf{c}^k \leftarrow \mathbf{c}^{k-1} + \lambda^k (\mathbf{r}^{k-1} - \mathbf{r}^k)$
 - 13: **end while**
 - 14: Output: $\mathbf{c}^* = \mathbf{c}^k$,
 $\tilde{\mathbf{s}}^* \leftarrow \mathbf{0}$ + corresponding entries from $\tilde{\mathbf{s}}^k$
 {If required, get $s_i^* \leftarrow (\tilde{s}_i^* - \tilde{s}_{i+n}^*)$, $1 \leq i \leq n$ }
-

4. EXPERIMENTAL EVALUATION

To compare our algorithm to the Basis Pursuit and Stagewise Polytope Faces Pursuit, we use a standard audio source separation problem generated by SPARCO toolbox for testing sparse reconstruction algorithms [5]. SPARCO Source separation example 2 (problem 402) consists of two audio mixtures made of three sources (guitar, piano and voice) mixed using instantaneous mixing.

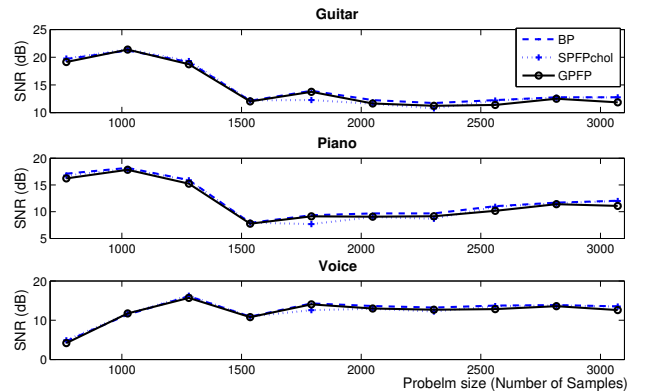


Fig. 1. Signal-to-Noise Ratio against problem size in number of samples

This problem is an undetermined source separation (the number of sources is larger than number of mixtures) and given that audio signals should be sparse in some transform

domain (in this case DCT domain), we can use sparse representation algorithms to aid source separation. In our experiment, we ran SolveBP solver from the SparseLab toolbox [10] for 10 iterations, the Stagewise version of Polytope Faces Pursuit [6] selecting 10 faces per iteration and our proposed GPF algorithm. We carried out experiments for 10 different problem sizes (from 768 samples to 3072 sample in 256 samples increments). We compared the Signal to Noise Ratio of reconstruction of three sources and the time spent to reconstruct the sources. As can be seen from Figure 1 Basis Pursuit gives slightly better SNR, but the difference in audio quality is indistinguishable.

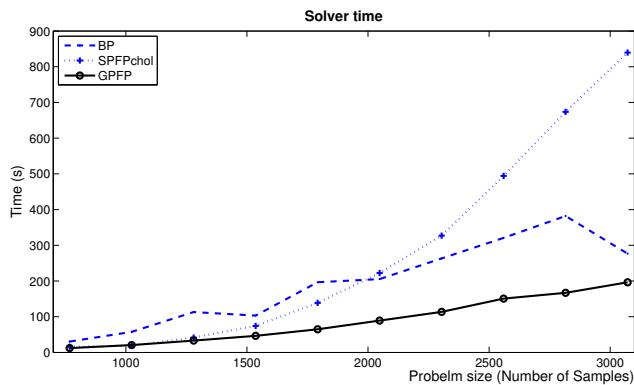


Fig. 2. Elapsed time against problem size in number of samples

The time needed to recover the sources rise almost linearly with problem size for GPF, in contrast with a faster than linear rise of Stagewise PFP algorithm (Figure 2). It can be also seen that the SolveBP implementation of the Basis Pursuit algorithm is in average 2.5 times slower than GPF.

5. CONCLUSIONS

We have introduced a new greedy algorithm, which finds approximate sparse solutions to sparse representations problems, which we call Gradient Polytope Faces Pursuit. We demonstrated that it is suitable for fast convergence especially in the case of large scale problems, as it has less storage and memory requirements compared to standard implementations of Polytope Faces Pursuit and Orthogonal Matching pursuit.

The SPARCO source separation example we used to verify the algorithms robustness, indicates that this new method can act as a suitable alternative to Basis Pursuit. The elapsed time for reconstruction increases linearly with time, when GPF is used. In contrast, Stagewise Polytope Faces Pursuit, based on the Cholesky factorization achieves almost BP reconstruction, while it is slower and the convergence time increases faster than linear with the problem size.

Our future work will focus on the further improvement of the presented algorithm including the investigation of tech-

niques that could lead to a Stagewise version of Gradient Polytope Faces Pursuit.

To download the MATLAB code used in this paper see: <http://www.elec.qmul.ac.uk/digitalmusic/papers/2010/Gretsistas10icassp/>

6. REFERENCES

- [1] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 33–61, 1998.
- [2] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3397–3415, 1993.
- [3] Y. C. Pati, R. Rezaifar, and P. S. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," in *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, 1-3 Nov. 1993*, pp. 40–44.
- [4] M. D. Plumbley, "Recovery of sparse representations by polytope faces pursuit," in *Proceedings of the 6th International Conference on Independent Component Analysis and Blind Source Separation (ICA 2006), Charleston, SC, USA, 5–8 March 2006*, pp. 206–213, LNCS 3889.
- [5] E. van den Berg, M. P. Friedlander, G. Hennenfent, F. Herrmann, R. Saab, and Ö. Yılmaz, "SPARCO: A testing framework for sparse reconstruction," Tech. Rep. TR-2007-20, Dept. Computer Science, University of British Columbia, Vancouver, October 2007.
- [6] M.D. Plumbley and M. Bevilacqua, "Stagewise Polytope Faces Pursuit for Recovery of Sparse Representations," *In Proc DSP 2009*.
- [7] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
- [8] G.H. Golub and C.F. Van Loan, *Matrix computations*, Johns Hopkins Univ Pr, 1996.
- [9] T. Blumensath and ME Davies, "Gradient pursuits," *IEEE Transactions on Signal Processing*, vol. 56, no. 6, pp. 2370–2382, 2008.
- [10] DL Donoho, V. Stodden, and Y. Tsaig, "SparseLab: Sparse solution of underdetermined linear systems of equations," Tech. Rep., Stanford University, <http://sparselab.stanford.edu>, 2007.