Audio Denoising using Sparse Approximation Techniques in Time-Frequency Domain

Ajin Tom
Music Technology Department
McGill University - Montreal, CA
ajin.tom@mail.mcgill.ca

ABSTRACT

This research project considers the problem of finding sparse approximations of audio signals using techniques like Lasso-based regression as well as Orthogonal Matching Pursuit for applications like audio denoising. This involves an implementation, evaluation and exploration of greedy as well as relaxation methods approaches to solve the sparse approximation problem.

Keywords

denoising, sparse approximations, time-frequency, orthogonal matching pursuit, lasso based regression

1. Introduction

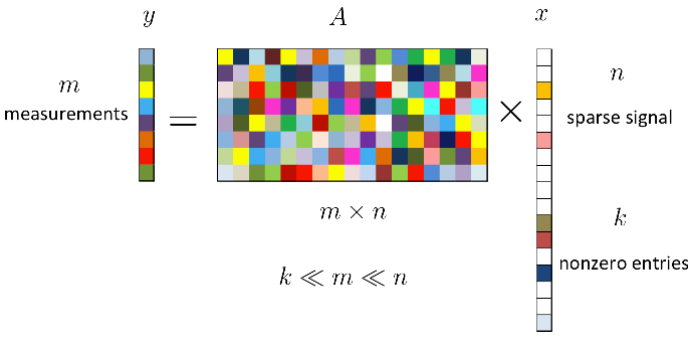
In the recent years there has been a growing interest in sparse approximations involving time-frequency domain. This is mainly associated to the large number of applications these techniques bring, especially in the audio domain. The task of finding sparse approximations is not trivial as there is no general method guaranteed to work in every situation. Even if we get them to work, there is always various tuning methods to make the algorithm work efficiently.

This project considers the problem of finding sparse approximations of audio signals using two common techniques Lasso-based regression as well as Orthogonal Matching Pursuit for applications like audio denoising. The lasso replaces the sparse approximation problem by a convex optimization problem so that the algorithms can effectively find solutions. The Orthogonal matching pursuit is a greedy method for solving the sparse approximation problem. This algorithm involves an iteration process in which the column vectors of a dictionary is chosen based on resemblance with the required vectors to build the solution.

2. Sparse approximations

Finding a sparse approximation is more than just an abstract mathematical problem. Sparse approximations have a wide range of practical applications. Vectors are often used to represent large amounts of data which can be difficult to store or transmit. Thus, by using a sparse approximation, the amount of space needed to store the vector would be reduced to a fraction of what was originally needed. Sparse approximations can also be used to analyze data by showing how column vectors in a given basis come together to produce the data.

Sparse representations of signals end up giving clear interpretations when the appropriate basis can be found. Comparing various models, sparser models are preferred since each variable then holds more meaning.



In the context of this project, the sparse measurement vector is defined as follows:

$$\boldsymbol{y = Ax} \qquad \textit{where,}$$

y - observation vector (size m)
A - dictionary matrix (size m x n)
x - sparse signal with fewest number of non-zero entries (size n)

Assuming m << n, number of observations is much smaller than dimension of source signal;

Aim is to recover the sparsest solution by solving

$$\textbf{arg min} \; \| \mathbf{x} \|_0 \; \text{subject to} \; \mathbf{y} = \mathbf{Ax}$$

where $\| \mathbf{x} \|_0$ is the number of non-zero entries of x

An exhaustive search method for this problem involves:

- Fix support of x: determine which entry of x is zero and non-zero
- Check if corresponding x has a solution for y = Ax
- Solve all 2^n equations, to find an optimal solution

Better strategies would involve branch and bound methods and that could end up being time consuming. So it is natural to seek approximate solutions.

There are 2 main strategies that have been explored in this project to recover sparse solutions. The first one is to use a cost function that can traceably minimized, the lasso. The second one is to use greedy algorithms which build up a solution by selecting one coefficient in x per iteration.

In the following sections, both the algorithms and their performances along with their solutions are discussed. Numerical results are also provided to compare the different algorithms and to discuss suitability for the different audio applications.

3. The Lasso and Basis Pursuit

3.1. Concept

The idea of Basis Pursuit is to replace the difficult sparse problem with an easier optimization problem. The formal definition of the sparse problem can be represented as follows:

$$\textbf{min} \; \|\mathbf{x}\|_0 \; \text{subject to} \; \mathbf{Ax} = \mathbf{y}$$

The difficulty with the above problem is the L0 norm. Basis pursuit replaces the L0 norm with the L1 to make the problem easier to work with:

$$\textbf{min} \; \|\mathbf{x}\|_1 \; \text{subject to} \; \mathbf{Ax} = \mathbf{y}$$

The solution to the above problem is relatively easier if the right conditions are chosen following which the sparsest solution can be obtained. This is mainly because L1 norm is usually concerned with the value of the entries than the quantity. A vector with a small L1 could have very small valued non-zero entries in every position which would give it a large L0 norm.

The Lasso is similar to Basis Pursuit and is a commonly used for de-noising techniques. The Lasso rather than trying to minimize the L1 norm like Basis Pursuit, places a restriction on its value, this is commonly referred to as regularization:

$$\textbf{min} \; \|\mathbf{Ax} - \mathbf{b}\|_2 \; \text{subject to} \; \|\mathbf{x}\|_1 < \mathbf{e}$$

The Lasso allows us to find approximations rather than just representations and like Basis Pursuit, can be guaranteed to find the sparsest solution under the right conditions.

3.2. Algorithm

The Lasso is an optimization principle rather than an algorithm. There are numerous algorithms to solve the problems above involving L1 norm.

The most basic algorithm for this problem starts with a solution $\mathbf{x_0}$ where $\mathbf{Ax_0} = \mathbf{y}$. Then goes through an iteration process changing the entries in $\mathbf{x_{k-1}}$ to form a new solution $\mathbf{x_k}$ while maintaining the condition $\mathbf{Ax_k} = \mathbf{y}$. A transformation is then applied to $\mathbf{x_k}$ which effectively sparsifies $\mathbf{x_k}$. Eventually a vector is

reached that meets the preset stopping conditions and by forcing all extreme small entries to zero the final solution is obtained. It should be noted that there are some more general interior point methods which do not require the condition $\mathbf{Ax_k = y}$ to be met during the iteration process as long as the final solution meets the condition.

Expanding on these concepts, we now introduce the regularization parameter with an aim to identify a sparse expansion into an over complete dictionary with M < N, by minimizing the functional:

$$\Psi(x) = \left\| y - \sum_k x_k \varphi_k \right\|_2^2 + \lambda \|x\|_1$$

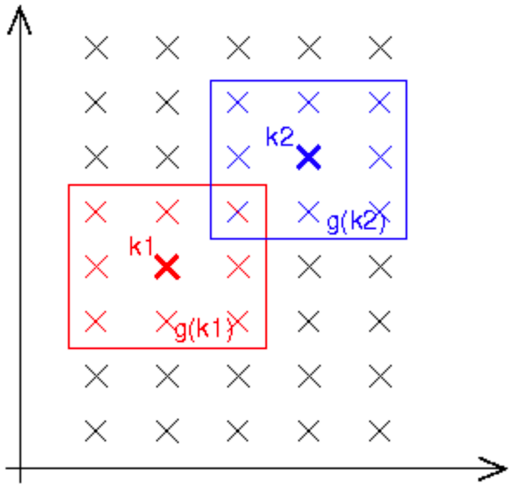
The L1 penalty on the synthesis coefficients **xk** yields sparsity. The regression problem with L2,1 penalty is known as Group-Lasso (G-Lasso) regression, which is basically regression with multiple measurement vectors.

One main shortcoming in this technique is the independence across the groups, ie; a given coefficient cannot belong to two different groups. So in this case we can consider a smoothening across the groups by using the concept of neighbourhoods. This technique is called Windowed-Group Lasso (WG-Lasso).

In G-Lasso the hierarchy is fixed once for all and given number cannot belong to several groups. To relax this constraint, a single index k is used to label the analysis coefficients yk. Then we associate with any index k a family of neighbourhood indices g(k) = { m ; m belongs to neighbourhood k }, and compute the shrinkage coefficient as follows:

$$\nu_k(\underline{y}) = \frac{\lambda}{\sqrt{\sum_{m \in g(k)} |y_m|^2}} = \frac{\lambda}{\|\underline{y}_{g(k)}\|_2}$$

When the neighbourhoods are disjoint, WG-Lasso reduces to G-Lasso since there will not be any overlap of groups. The following figure shows a good illustration of 2 overlapping groups in a WG-Lasso. The neighbourhood of the coefficient k1 is given by red window, and the neighbourhood of k2 by the blue one. Both the neighbourhoods share one coefficient.



3.3. Implementation

As far as implementation is concerned, the Linear Time-Frequency Analysis Toolbox in MATLAB is used for performing various algorithmic tasks like Gabor transform.

The major challenge in the implementation involved creating a sliding window to define the neighbourhood before choosing the coefficients. In the iterative process the shrinkage operates only on the original analysis coefficients to give corresponding synthesis coefficients.

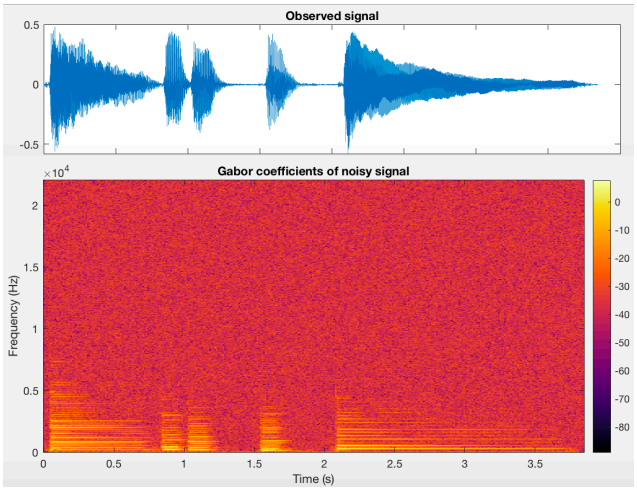
Following is the implementation of WG-Lasso:

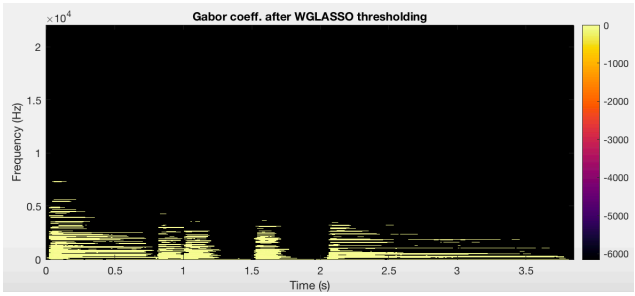
```
ltfatstart;
%load signal
[s,Fs]  = audioread('piano.wav');

T = length(s);
Time = linspace(0,T/Fs,T);

figure;
plot(Time, s);
title('Observed signal');
```

```
%induce noise
snrlevel = 10;
% sn = awgn(s,snrlevel,'measured');
% disp(snr(s,sn));

sigma_noise = norm(s)^2*10^(-snrlevel/10)/T;
noise = sqrt(sigma_noise)*randn(size(s));
sn = s + noise;
disp(snr(s,sn));
% Gabor parameters: window length, type,
overlap
M = 1024;
a = M/2;
g = gabwin({'tight', 'hann'}, a, M);

% Gabor transform
G_sn = dgtreal(sn, g, a, M);
figure;
plotdgtreal(G_sn,a,M,Fs);
title('Gabor coefficients of noisy
signal');

lambda = sqrt(sigma_noise);
G_sd = G_sn.*max(0,1-lambda./abs(G_sn));

% WG Lasso

%size of the neighborhood
K = 5;
neigh = ones(1,K);
neigh = neigh/norm(neigh(:),1);

% center of the window
c = ceil(K/2);

% matrix to stock the local energy of
each neighborhood
% centralize gabor squared coefficients,
mirror left and right borders

[MG,NG] = size(G_sn);
W = zeros(MG, NG+K-1);
W(:, c: NG+c-1) = abs(G_sd).^2;
W(:, 1:c-1) =  fliplr(W(:, c :
2*(c-1)));       % left border
W(:, NG+c:end) = fliplr(W(:, NG - K
+2*c: NG+c-1));  % right border

% neighborhood energy
W = (conv2(W, neigh, 'same'));
W = W(:, c : NG + c -1);

% thresholding
W = sqrt(W);
G_sd = G_sn.*max(0,1-(lambda./W));

figure;
plotdgtreal(G_sd,a,M,Fs);
title('Gabor coeff. after WGLASSO
thresholding');

% snr
sd = idgtreal(G_sd,g,a,M,T);
disp(snr(s,sd));
```

3.4. Results and Discussion

The WG-LASSO with structured shrinkage implementation was able to remove the noise successfully. Various signals were used for testing the performance of the algorithm. Out of which here are the results for a short piano melody:





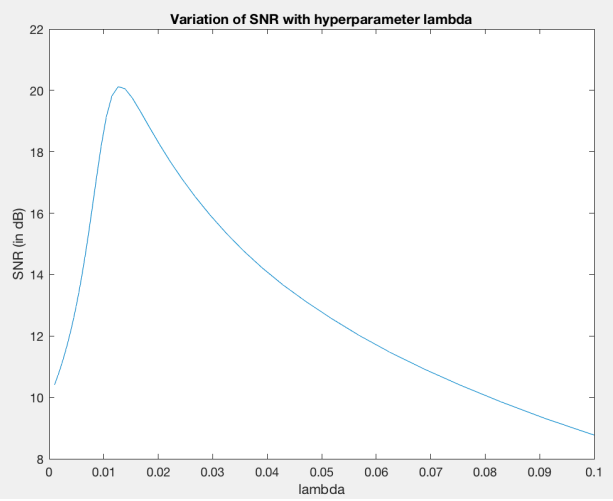
It can be clearly seen the white noise components have even removed without much loss of the original signal components. As far as sound output is concerned, perceptually the output seems to have denoised successfully (tested across 4 subjects informally).

Here are some of the input hyper-parameters as well as observed output values:

Original SNR	9.9966 dB
SNR after denoising	16.4479 dB
Lambda (hyperparameter)	0.0275
Neighbourhood size	5

Following is the trend for SNR observed for increasing values of lambda:

lambda	SNR (in dB)
0.0012	**10.5238**
0.0016	**10.7191**
0.0021	**10.9962**
0.0028	**11.3994**
0.0037	**12.007**
0.0049	**12.9677**
0.0066	**14.5676**
0.0087	**17.1672**
0.0115	**19.8223**
0.0153	**19.7477**
0.0202	**18.2435**
0.0268	**16.5291**
0.0356	**14.7908**
0.0471	**13.1176**
0.0625	**11.473**



4. Orthogonal Matching Pursuit

4.1. Concept

The Orthogonal Matching Pursuit (OMP) constructs an approximation of an observed signal by going through an iterative process using a defined dictionary. MP and OMP are both greedy algorithms used to obtain sparse signal approximations. OMP is known to offer better performance and the difference from MP is that in each iteration the decomposition y is calculated by projecting the signal x orthogonally onto all the selected atoms. OMP therefore calculates the best signal approximation possible with these atoms. This benefit comes at the cost of orthogonal projection.

The iterative process involves calculation of the locally optimum solution. This is done by finding the column vector in **A** whim most closely resembles a residual vector **r**. The residual vector starts becoming equal to the vector that is required to be approximated, ie: **r** = **b** , and is adjusted at every iteration to take into account the vector previously chosen. It is the hope that this sequence of locally optimum solutions will lead to the global optimum solution. MP simply removes the selected column vector from the residual vector at each iteration.

4.2. Algorithm

Input:

- signal **y** : observed signal
- Matrix **A** : dictionary
- Stopping criterion until level of accuracy is reached: sparsity **k**

Output:

- Approximation vector **x**

Algorithm:

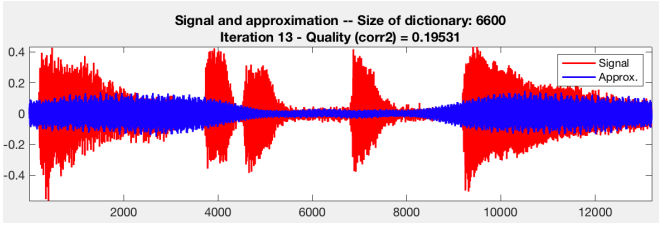
1. Initialize residual **r** = **y**, time = 0
2. Initialize an array, **vk** to store dictionary atoms
3. The best atom is chosen by computing the solution of max < **r, ak** > where **ak** are the column vectors of **A**
4. The index of the chosen atom is appended to the **vk** array of atom indices.
5. Solve the least square problem by computing **x**, the solution to under/overestimated system of equations **y** = **A** * **x**
6. Residual is calculated by performing **r** = **y** - **Ax**
7. The crucial step in OMP involves an iteration in which the component of the atom just chosen is removed from all the column vectors in the dictionary matrix **A.**
8. These steps continue as the iterations carry on until the stopping criteria is reached : *in this implementation - reaching desired SRR*

4.3. Implementation

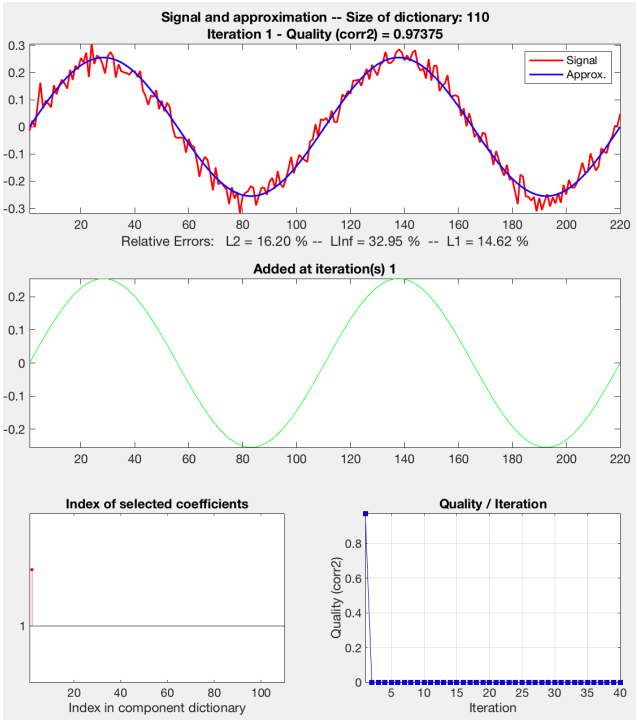
```
function  x = my_omp (k, A, y)

%initialising residue and atoms_array
atom_vector = zeros(size(A,2),1);
ind_atom=[];
res=y;

thresh = 0.33;
res_norm = [];
sig_norm = [];
count = 1;
% preserve original dictionary
A_dup = A;.

% iterations begin
while count < k+1
% choosing best atom
inner_product = abs(A' * res);
[atom, atom_index] = max(inner_product);

% append to atom-indices array
ind_atom = [ind_atom atom_index];

% least-sq solution to y = A * x
xfinal = A_dup(:, ind_atom)\y;

% compute residual
res = y - A_dup(:,ind_atom) * xfinal;

% orthogonlaise dictionary
for i = 1 : size(A,2)
A(:,i) = A(:,i) -
A(:,i)'*A(:,atom_index) *
A(:,atom_index);
end

res_norm = [res_norm norm(res,1)];
sig_norm = [sig_norm norm(y,1)];

% stopping criteria
if (res_norm(count)) < thresh *
sig_norm(count) break;
end

count = count + 1;
end

x = atom_vector;
t = ind_atom';
x(t) = xfinal;
end
```

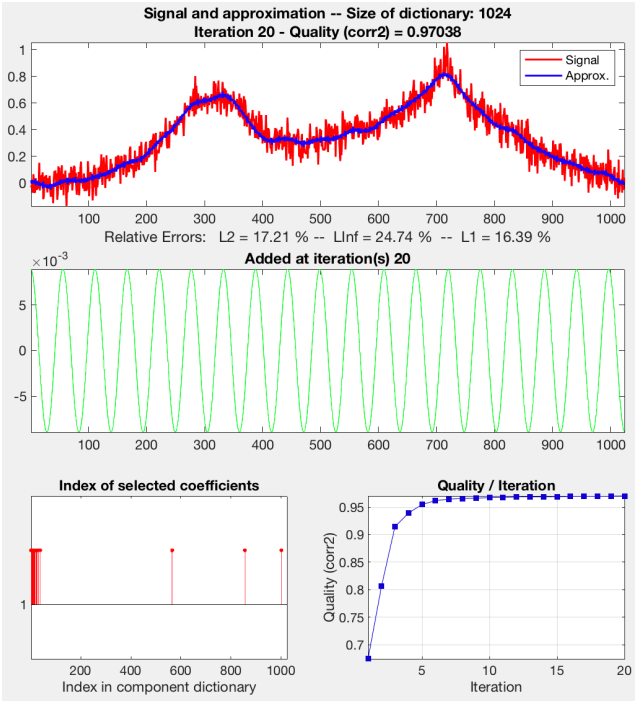
4.4. Results and Discussion

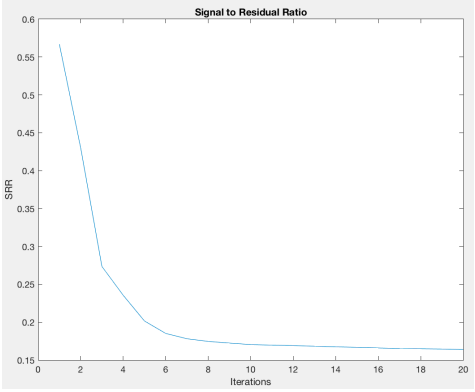
OMP proves to be a relatively fast algorithm for reconstruction problems. But in the context of audio denoising, OMP becomes really tricky as the shrinkage rules are not trivial anymore and the implementation of OMP ends up being susceptible to noise. The choice of dictionary also plays a crucial role for more accurate and faster convergence. For complex signals like the piano melody used in lasso, the algorithm tries to fit the best atom (sine/dct dictionary in this case) in the first few iterations itself; these chosen atoms ends up being the sinusoid that best approximates the envelope of the signal:



Choosing the number of iterations and/or L1 error threshold plays a major role in determining the denoised reconstruction. Here are some of the results from plotting sine tones as well as the cuspmax signal in Matlab:







5. Evaluation & Conclusion

Finally we can compare the properties of OMP and Lasso. The table below shows results of an average of 20 runs over a wide variety of signals: transient, tonal, monophonic, polyphonic and other complex signals:

Lasso			OMP		
Time	Lambda	SNR	Time	Iterations	SNR
0.536	0.002	10.9	0.081	1	7
0.583	0.006	12.07	0.094	2	5
0.6	0.01	12.96	0.1	3	2
0.72	0.05	14.96	0.135	4	1
0.76	0.06	11.8	0.21	5	3
0.8	0.07	10.6	0.29	6	2

Comparison	Lasso	OMP
Computational Cost		✓
Ease of implementation		✓
Effectiveness	✓	
Storage taken up	✓	

This work reviewed two popular sparse atomic approximation techniques for applications like audio denoising. This included an exploration of greedy as well as relaxed approaches to the sparse approximation problem. From these results, we can see that OMP is in general a faster running algorithm than the Lasso, mainly due to its simplicity. The Lasso takes more time to compute as it is a more complicated optimization problem.

Lasso builds up an approximation by taking groups of vectors at a time from the dictionary unlike OMP. This gives relaxation based methods like Lasso a higher accuracy score, while greedy approximation methods like OMP has faster computational speed. My take-away from this project would be to build approximation techniques using groups of vectors at a time, just like in Lasso while maintaining a relatively fast run time like OMP.

6. ACKOWLEDGEMENT

I would like to thank my professor Dr. Philippe Depalle for giving me the opportunity for working on such an interesting project that is very relevant to my research interests. I really appreciate the constant support and help received when I was stuck with implementation aspects as well as interpretation of key concepts. Carrying out this project was a really enriching experience and I would love to use the concepts I learnt throughout this project and seminar for my ongoing research.

7. REFERENCES

1. S. S. Chen, D. L. Donoho, and M. A. Saunders (1999) Atomic decomposition by basis pursuit. SIAM J.Sci. Comp.
2. Tropp, J. A. (2004) Greed is good: Algorithmic results for sparse approximation, IEEE Trans. Inform. Theory, vol 50, no 10, pp. 2231-2242
3. Robert Tibshirani,"Regression Shrinkage and Selection via the LASSO," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.
4. Matthieu Kowalski, Bruno Torrésani. Structured Sparsity: from Mixed Norms to Structured Shrink- age. Rémi Gribonval. SPARS'09 - Signal Processing with Adaptive Sparse Structured Representations, Apr 2009, Saint Malo, France. 2009.
5. Diego, M. Kowalski, K. Siedenburg and M. Dörfler, "Social Sparsity! Neighborhood Systems Enrich Structured Shrinkage Operators," in *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2498-2511, May15, 2013.
6. DavidL.Donoho,"De-noising by soft-thresholding,"*Information Theory, IEEE Transactions on*, vol. 41, no. 3, pp. 613–627, 1995.
7. G. Bhattacharya and P. Depalle, "Sparse denoising of audio by greedy time-frequency shrinkage," *2014 IEEE International Conference on Acoustics,*
8. Kereliuk, Corey, and Philippe Depalle. "Sparse atomic modeling of audio: A review." *Proc. DAFx-11* (2011).
9. Time-Frequency Analysis toolbox: http://ltfat.github.io
10. Link to GitHub repo: https://github.com/ajintom/denoise.git