IMPERIAL COLLEGE of SCIENCE, TECHNOLOGY & MEDICINE

DEPARTMENT of ELECTRICAL & ELECTRONIC ENGINEERING

MSc in Communications and Signal Processing

Formal Report No. 1

Name	Yibing Liu
Experiment Code	WD
Title of Experiment	Recover Sparse Signals from Under-Sampled Observations
Date of Submission	27/11/2020
Supervisor of Experiment	Dr. Wei Dai
Grade	

Communications and Signal Processing Laboratory

Recover Sparse Signals from Under-Sampled Observations

Yibing Liu
Electrical & Electronic Engineering
Imperial College London
yibing.liu20@imperial.ac.uk

Aims— Provide a comprehensive and complete understanding to the concept of sparse signals. Emphasis will be given by the use of different algorithms to realize the recovery of the sparse signals from under-sampled observations.

I. BACKGROUND

The sparse signal is the signal which contains only a small number of non-zero elements compared to its dimension. It has a good preformation in the compression,

Sparse signal processing has become a major component in modern signal processing theory, underpinning compressed sensing, linear regression, machine learning, big data processing, etc. It is based on the observation that most signals, under certain transform or dictionary, only contain a few significant components. Based on this sparsity, efficient ways for data acquisition, processing, and analysis can be developed.

Sparse recovery is a fundamental problem in the fields of compressed sensing, signal de-noising, statistical model selection, and more. The key idea of sparse recovery lies in that a suitably high dimensional sparse signal can be inferred from very few linear observations.

We can use different algorithm to realize the sparse recovery, such as the Orthogonal Matching Pursuit (OMP) Algorithm, the Subspace Pursuit (SP) Algorithm and the Iterative Hardthresholding (IHT) Algorithm.

II. INTRODUCTION

Linear System

For the linear system:

$$y = Ax$$

Where $y \epsilon \mathbb{R}^m$ denotes the m observation vector, $A \epsilon \mathbb{R}^{m \times n}$ is the given linear transform, and $x \epsilon \mathbb{R}^n$ is the n variables input vector.

Hence, linear transformation:

$$T_A: \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$n - vectors\ input \rightarrow m - vectors\ output$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & \cdots & \cdots & a_{2n} \\ \vdots & \cdots & \cdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

There are three conditions:

① m=n: a square system, A is a square matrix,

If A is invertible, $x = A^{-1}y$: the exact solution

② m>n: overdetermined system, A: tall matrix,

x has least-square solution

③ m<n: underdetermined system, A: fat matrix

There are infinite many solutions, x has basic

solution with m-sparse.

Sparse Signal

The general signals can be transformed into sparse signals through simple operation:

$$x = Tc$$

$$c \in \mathbb{R}^l: original\ signal$$

$$x \in \mathbb{R}^n: \text{sparse signal}$$

$$T \in \mathbb{R}^{n\times l}: transformation\ matrix$$

For a sparse signal, its support set is:

$$\text{Supp}(x)= \text{index of non-zero entries}$$

$$s = \{i, x_i \neq 0\}$$

For example, $x = [5,6,0,7,0]$, $s = [1,2,4]$

The sparsity of x is the size of s

For $I = supp(x)$, A_I is the compact matrix constructed by I and x_I is the compact vector constructed by I, then

$$y = Ax = A_I x_I$$

Note: A does not need to be sparse, but x should be hard thresholding function $H_s(x)$

$(x \in \mathbb{R}^n, s < n)$

Remain the S largest entries(in magnitude) and set all other entries to 0.

$$[H_s(x)]_i = \begin{cases} x_i, i \in supp(x) \\ 0, i \notin supp(x) \end{cases}$$

For example, s=1, $x = [1,2,3]$, then $[H_s(x)]$=[0,0,3].

Projection

Linear subspace

A=[$a_1, a_2, a_3 \cdots a_n$] $\in \mathbb{R}^m$ contains linearly independent vectors, thus the linear span of A is

$$span(A) = \left\{\sum_{i=1}^{n} \lambda_i a_i,\ \lambda_i \in \mathbb{R}\right\}$$

A is the basis for the linear subspace, $s = \langle A \rangle$

dim(s) = n: the number of vectors in the basis.

Projection

The projection of $x \in \mathbb{R}^n$ on the subspace span(A) is:

$$x_p = Proj(x, A) = AA^{\dagger}x = A(A^TA)^{-1}A^{-1}y$$

The residue is:

$$x_{\mathrm{r}} = Resid(x, A) = x - x_p$$

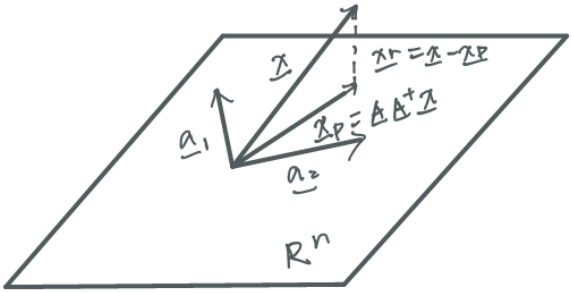

Figure 1 the projection and residue

Note: for linearly independent columns:

$$A^{\dagger} = (A^{\mathsf{T}}A)^{-1}A^{\mathsf{T}}$$

for linearly independent rows

$$A^{\dagger} = A^{\mathsf{T}}(AA^{\mathsf{T}})^{-1}$$

Greedy Algorithm

A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage. In many problems, a greedy strategy does not usually produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.

The Orthogonal Matching Pursuit(OMP) Algorithm

The OMP selects one column that is most correlated to the current residue from the dictionary matrix A at each step. Then add this column to the current selected column set. Then use the current selected set to approach the optimal sparse signal x. The current residue is obtained by projecting the observation y onto the linear subspace spanned by the current selected columns and the algorithm iterate. When the stop condition satisfies, the y can be generated by linearly combining the selected columns and optimal sparse signal x. Note that the residue after each step I the OMP is orthogonal to all selected columns, so no columns will be chosen twice.

The advantage of AMP is simple to implement and has a high speed with fast implementation.

The Subspace Pursuit(SP) Algorithm

The SP is an algorithm that improves the OMP. SP selects k columns in each iteration, while OMP selects one column. Also, in SP, the selected column in each step may be abandoned in next iteration, however, the selected column in OMP will be kept. Therefore, the SP has backtracking.

SP has low computation complexity, especially for reconstruction of relatively sparse signal. SP also has similar reconstruction accuracy to the linear LP algorithm .

The Iterative Hardthresholding (IHT) Algorithm

The IHT is to find the optimal sparse x by continuously iterating with the hardthresholding equations. It learns from residue and optimize the result in each iteration.

III. PROCEDURE

OMP:

Input: Transformation or dictionary matrix A, sparsity S and the observation y.

Output: $\hat{x}$ – the S-sparse approximation of x

Initialization: residue $y_r = y$, $\hat{x} = 0$,index set $s = \Phi$.

Step : iterate steps 1 to 3 for s times or the exit criteria are true.

① Find the index of the largest entries in the product of residue y_r and A. Then update the selection set, as

$$\mathcal{S} = \mathcal{S} \bigcup \text{supp}\left(H_1\left(\boldsymbol{A}^T \boldsymbol{y}_r\right)\right).$$

② Obtain the optimal $\hat{x}$ in the current step with:

$$\hat{\boldsymbol{x}}_{\mathcal{S}} = \boldsymbol{A}_{\mathcal{S}}^{\dagger} \boldsymbol{y} \text{ and } \hat{\boldsymbol{x}}_{\mathcal{S}^c} = \boldsymbol{0}.$$

③ Update the residue with $y_r = y - A\hat{x}$. Also judge whether the step condition is satisfied, if it is, jump out of the iteration.

SP:

Input: Transformation or dictionary matrix A, sparsity S and the observation y.

Output: $\hat{x}$ – the S-sparse approximation of x

Initialization: selection set $S = supp\big(H_s(A^T y)\big)$ and $y_r = resid(y, As)$

Step : iterate steps 1 to 5 until the exit criteria are true.

① Find the index of S largest entries in the product of A and y_r, update the selection set by adding these S index with

$$\tilde{\mathcal{S}} = \mathcal{S} \bigcup \text{supp}\left(H_S\left(\boldsymbol{A}^T \boldsymbol{y}_r\right)\right).$$

② Set $\boldsymbol{b}_{\tilde{\mathcal{S}}} = \boldsymbol{A}_{\tilde{\mathcal{S}}}^{\dagger} \boldsymbol{y}$ and $\boldsymbol{b}_{\tilde{\mathcal{S}}^c} = \boldsymbol{0}$.

③ Find the largest S entries in $b_{\tilde{S}}$ and then generate the selection set with

$$S = supp\left(H_s(b)\right)$$

④ Then obtain the optimal sparse signal $\hat{x}$ at the current step with

$$\hat{\boldsymbol{x}} = \boldsymbol{A}_{\mathcal{S}}^{\dagger} \boldsymbol{y} \text{ and } \hat{\boldsymbol{x}}_{\mathcal{S}^c} = \boldsymbol{0}.$$

⑤ Update the residue with $y_r = y - A\hat{x}$, if the current residue is larger than the last residue or the normalized error is below certain value, then skip the iteration. As $\|y_r\|_2 > \|y_r - pre\|_2$ or $\|y_r\|_2/\|y\|_2$<1e-6.

IHT:

Input: Transformation or dictionary matrix A, sparsity S and the observation y.

Output: $\hat{x}$ – the S-sparse approximation of x

Initialization: residue $y_r = y$, $\hat{x} = 0$.

Step : iterate steps 1 to 2 for s times or the exit criteria are true.

① Execute the optimization function:

$$\hat{\boldsymbol{x}} = H_S\left(\hat{\boldsymbol{x}} + \boldsymbol{A}^T\left(\boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}}\right)\right).$$

② When the current residue is larger than the last one or the normalized residue is in the acceptable range, as $\|y_r\|_2 > \|y_r - pre\|_2$ or $\|y_r\|_2/\|y\|_2$<1e-6, then exit the iteration.

Experiment requirement:

Exercise 1:

Compare the vectors x, $\hat{x}_1$and $\hat{x}_2$:

```
clc;
m=128;
n=256;
A=normc(rand(m,n));
x=randn(n,1);
y=A*x;
x1=pinv(A)*y;
x2=A\y;
```

Exercise 2:

```
close all;
clc;

m=128;
n=256;
S=12;
A=normc(randn(m,n));
x=zeros(n,1);
suppSet = randsample(n,S);
x(suppSet,:)=randn(length(suppSet),1);
y=A*x;
x1=pinv(A)*y;
x2=A\y;
%OMP Algorithm
xs=zeros(n,1);
yr=y;
index=zeros(S,1);
for i = 1:1:S
    proj=A'*yr;
    pos=find(abs(proj)==max(abs(proj)));

    pos=pos(1);
    index(i)=pos;
    xs=zeros(n,1);
    xs(index(1:i))=A(:,index(1:i))\y;
    yr=y-A*xs;
end
x_OMP=xs;
%SP Algorithm
proj = A'*y;
[var,pos]=sort(abs(proj),'descend');
T=[];
T=union(T,pos(1:S));
yr=y-A*A'*y;
while(1)
    proj=A'*yr;
    [~,pos]=sort(abs(proj),'descend');
    T_add=union([],pos(1:S));
    T=union(T,T_add);
    bs=zeros(n,1);
    bs(T)=A(:,T)\y;
    [~,pos]=sort(abs(bs),'descend');
    T=union([],pos(1:S));
    xs=zeros(n,1);
    xs(T)=A(:,T)\y;
    yr_n=y-A*xs;
    if (norm(yr)^2/norm(y)^2<1e-
6||norm(yr_n)>norm(yr))
        break;
    end
    yr=yr_n;
end
x_SP=xs;
%HIT Algorithm
xs=zeros(n,1);
yr=y-A*xs;
while(1)
    xs=xs+A'*(y-A*xs);
    [val,pos]=sort(abs(xs),'descend');
    xs(abs(xs)<min(val(S)))=0;
    yr_n=y-A*xs;
    if ((norm(yr_n,2)/norm(y,2)<1e-
6)||norm(yr_n,2)>norm(yr,2))
        break;
```

```
    end
    yr=yr_n;
end
x_IHT=xs;
```

Exercise 3:

```
close all;
clc;

m=128;
n=256;
N=500;  %Number of tests
S=3:3:63;
succ_omp=zeros(1,length(S));
succ_sp=zeros(1,length(S));
succ_iht=zeros(1,length(S));
for Sn=1:1:length(S)
    succ_numo=0;
    succ_numsp=0;
    succ_numiht=0;

    for t=1:1:N
        A=normc(randn(m,n));
        x=zeros(n,1);
        suppSet = randsample(n,S(Sn));
        x(suppSet,:)=randn(length(suppSet),1);
        y=A*x;
        %OMP Algorithm
        xs=zeros(n,1);
        yr=y;
        index=zeros(S(Sn),1);
        for i = 1:1:S(Sn)
            proj=A'*yr;
            pos=find(abs(proj)==max(abs(proj)));
            pos=pos(1);
            index(i)=pos;
            xs=zeros(n,1);
            xs(index(1:i))=A(:,index(1:i))\y;
            yr=y-A*xs;
        end
        xomp=xs;
        %SP Algorithm
        proj = A'*y;
        [var,pos]=sort(abs(proj),'descend');
        T=[];
        T=union(T,pos(1:S(Sn)));
        yr=y-A*A'*y;
        while(1)
            proj=A'*yr;
            [~,pos]=sort(abs(proj),'descend');
            T_add=union([],pos(1:S(Sn)));
            T=union(T,T_add);
            bs=zeros(n,1);
            bs(T)=A(:,T)\y;
            [~,pos]=sort(abs(bs),'descend');
            T=union([],pos(1:S(Sn)));
            xs=zeros(n,1);
            xs(T)=A(:,T)\y;
            yr_n=y-A*xs;
            if (norm(yr_n,2)/norm(y,2)<1e-6||norm(yr_n,2)>=norm(yr,2))
                break;
            end
            yr=yr_n;
        end
        xsp=xs;
        %IHT Algorithm
        xs=zeros(n,1);
        yr=y-A*xs;
        while(1)
            xs=xs+A'*(y-A*xs);
            [val,pos]=sort(abs(xs),'descend');
            xs(pos(S(Sn)+1:n))=0;
            yr_n=y-A*xs;
            if ((norm(yr_n,2)/norm(y,2)<1e-6)||norm(yr_n,2)>norm(yr,2))
                break;
            end
            yr=yr_n;
        end
        xiht=xs;

        if ((norm(xomp-x,2)/norm(x))<1e-6)
            succ_numo=succ_numo+1;
        end
        if (norm(xsp-x,2)/norm(x)<1e-6)
            succ_numsp=succ_numsp+1;
        end
        if (norm(xiht-x,2)/norm(x)<1e-6)
            succ_numiht=succ_numiht+1;
        end
    end
    succ_omp(Sn)=succ_numo/N;
    succ_sp(Sn)=succ_numsp/N;
    succ_iht(Sn)=succ_numiht/N;
end

plot(S,succ_omp);
hold on;
plot(S,succ_sp);
hold on;
plot(S,succ_iht);
xlabel('S');
ylabel('Success Rate');
legend('OMP','SP','IHT');
```

IV. RESULT & CONCLUSIONS

For exercise 1,

1	2	3
0.6946	0.2674	5.1412
-0.6430	-1.4605	0
-0.0251	0.0777	0
0.9979	0.1380	0
-0.1316	-0.7790	0
1.2251	0.6618	5.2171
-1.8754	-1.1361	-1.1633
-0.1294	-0.4055	-4.4725
1.0607	0.6364	0
1.5432	-0.1976	-3.4886
0.5505	0.5735	-2.3633
1.2345	0.3634	0
0.7782	0.8911	0
-0.9516	-0.8254	0
1.1571	0.8246	0
-0.2781	0.2015	0
1.2140	0.2681	0

Figure 2 the values of x, $\hat{x}_1$ and $\hat{x}_2$

We can see that, x, $\hat{x}_1$ and $\hat{x}_2$ are totally different.

For exercise 2,

```
>> length(find(x~=0))

ans =

    12

>> length(find(x1~=0))

ans =

   256

>> length(find(x2~=0))

ans =

   128
```

x, $\hat{x}_1$ and $\hat{x}_2$ are totally different, which represent three solutions for $y=Ax$. But the solutions obtained by *pinv(A)*y* and $A\backslash y$ are not the s-sparse signal, where the number of non-zero entries in $\hat{x}_1$ is 256, but in $\hat{x}_2$ is 128

```
>> x(find(x~=0))'
ans =
    0.7928    0.0257   -1.5363   -0.1471   -0.4593    0.0394    1.3952   -0.4365    1.6290   -0.0364   -0.7552    0.4716
>> x_OMP(find(x_OMP~=0))'
ans =
    0.7928    0.0257   -1.5363   -0.1471   -0.4593    0.0394    1.3952   -0.4365    1.6290   -0.0364   -0.7552    0.4716
>> x_SP(find(x_SP~=0))'
ans =
    0.7928    0.0257   -1.5363   -0.1471   -0.4593    0.0394    1.3952   -0.4365    1.6290   -0.0364   -0.7552    0.4716
>> x_IHT(find(x_IHT~=0))'
ans =
   -0.5178   -0.6283    0.6104   -1.1864   -0.7141    0.8483   -0.5145    0.8665   -0.5079    0.4998   -0.5289   -0.5047
```

For *x*, *x_omp*, *x_sp*, *x_iht*, it is clear that *x_omp* and *x_sp* are always equal to *x*. But *x_iht* always has some errors.

From the results above, it is obvious that under the sparsity of 12, OMP and SP algorithm can completely recover the original sparse signal. These two algorithms have an extremely high accuracy, while the IHT algorithm has a much lower accuracy as it often cannot recover the s-sparse signal.

For exercise 3,

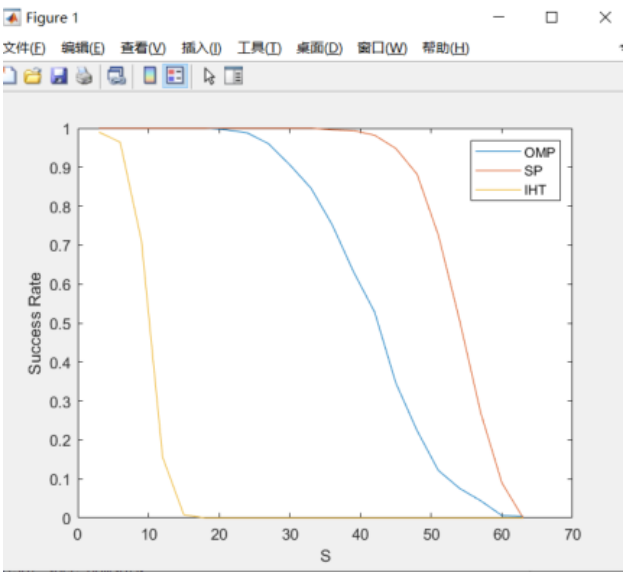

Figure 3 comparison of the performance for the three greedy algorithms

It is clear that SP has kept high accuracy until the sparsity becomes 40. The accuracies of OMP and IHT begin to decrease at about S=3, and the IHT decreases faster than OMP. Therefore, SP has the best performance in accuracy while IHT performs worst.

V. ACKNOWLEGEMENT

The author wishes to acknowledge the help and support given to him during the experiment.

Thanks go to:
Dr. Wei Dai (Academic in charge)
Mr. Yang Zhao (Lab Demonstrate)

VI. REFERENCES

[1].Y. Pati, R. Rezaiifar, and P. Krishnaprasad, “Orthogonal Matching Pursuit : recursive function approximation with application to wavelet decomposition”, in Asilomar Conf. on

Signals, Systems and Comput., 1993

[2].W. Dai and O. Milenkovic, “Subspace Pursuit for Compressive Sensing Signal Reconstruction”, IEEE Trans. Inf. Theory, 2009, 55, 2230-2249.

[3].T. Blumensath and M. Davies, “Iterative hard thresholding for compressed sensing”, Appl. Comput. Harmon. Anal., 2009, 27, 265 - 274.