Demo Project 1: Orthogonal Matching Pursuit (OMP) and Sparse Signal Recovery

Xuehai He

November 2019

1 Performance Metrics:

See code.

2 Experimental setup:

See code.

3 Noiseless case: (n = 0)

3.1 Implement OMP (you may stop the OMP iterations once $\left\|\mathbf{y}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2$ is close to 0) and evaluate its performance. Calculate the probability of Exact Support Recovery (i.e. the fraction of runs when $\hat{\mathcal{S}}=\mathcal{S}$) by averaging over 2000 random realizations of $\mathbf{A}$, as a function of M and $s_{\max}$ (for different fixed values of N). For each N, the probability of exact support recovery is a two dimensional plot (function of M and $s_{\max}$) and you can display it as an image. The resulting plot is called the "noiseless phase transition" plot, and it shows how many measurements (M) are needed for OMP to successfully recover the sparse signal, as a function of $s_{\max}$. Do you observe a sharp transition region where the probability quickly transitions from a large value (close to 1) to a very small value(close to 0)? Generate different phase transition plots for the following values of N : 20, 50 and 100. Regenerate phase transition plots for average Normalized Error (instead of probability of successful recovery). Comment on both kinds of plots.

We generated different phase transition plots for the following values of N: 20, 50, 80 and 100. And the result of "noisy phase transition" plots as a function of M and s_{max} for different fixed values of N are shown as follows:

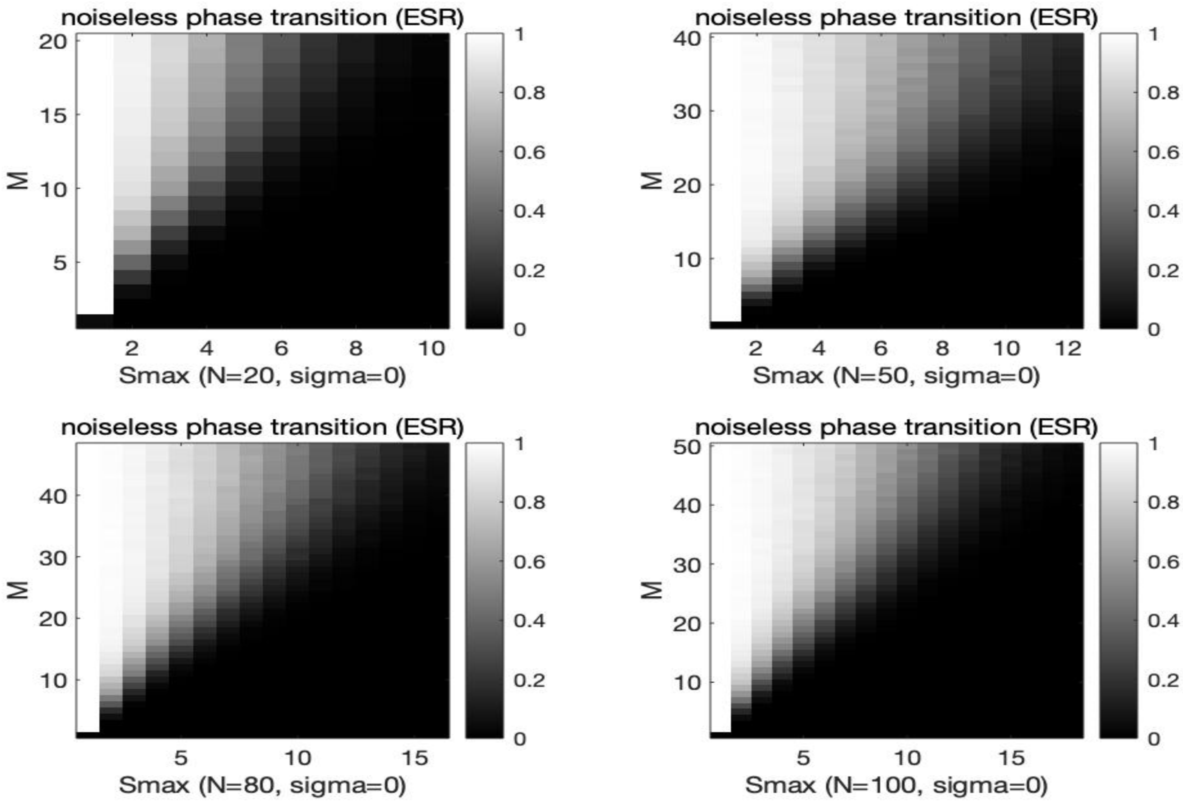

Figure 1: Exact support recovery

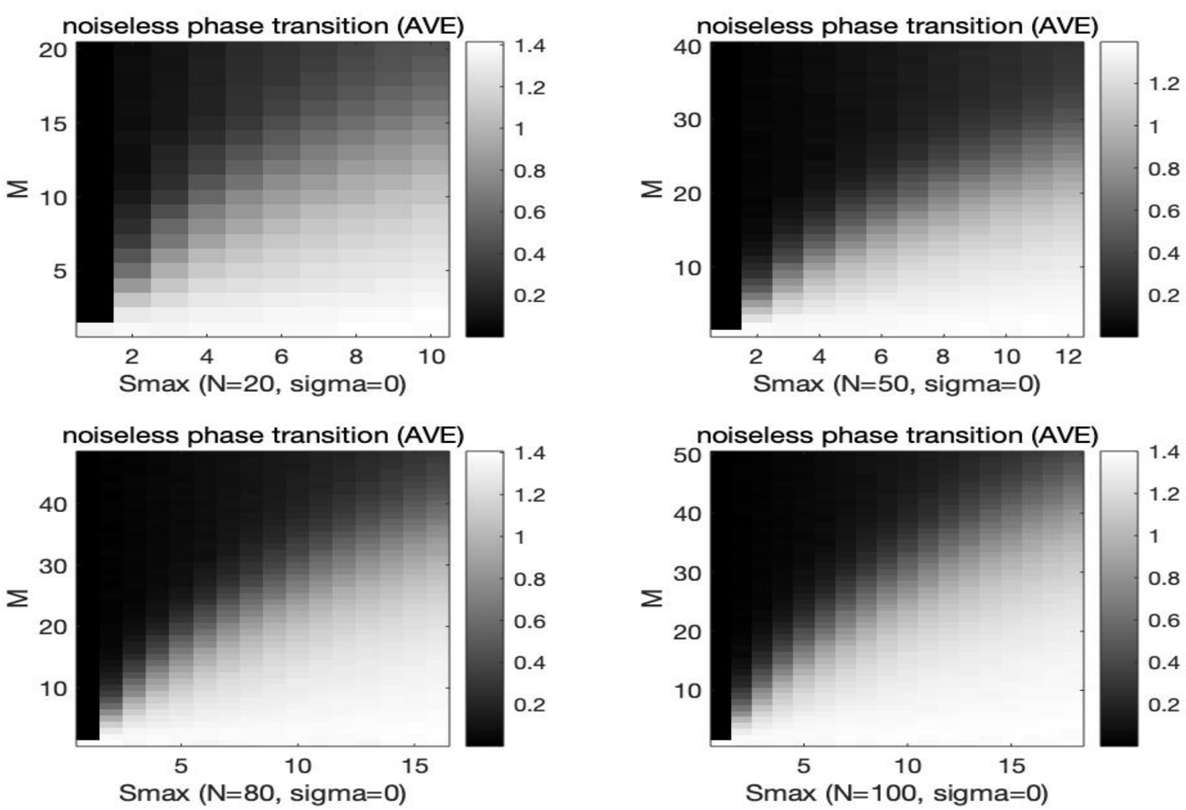

Figure 2: Average Normalized Error

From the “noiseless phase transition” plot of the probability of the Exact Support Recovery which is shown in Figure 1, we can observe a sharp transition region where the probability quickly transitions from a large value (close to 1) to a very small value (close to 0). It is when s_{max} is about 1 and M is about 1. And when after that when s_{max} and M continully grows, the transition region become blurred and we can not tell it very clearly. We can also see that as M increases, which means our measurement rate increases when we fix N, we tend to achieves higher probability of Exact Support Recover under the same sparsity rate. On the contrary, as s_{max} increases, which means our sparsity grows when we fix N, we tend to achieves lower probability of Exact Support Recover under the same measurement rate. These results are very obvious and clear to see. Because in compressed sensing, the more information we could get, we are more likely to recover the original signal [1]. Meanwhile, we could also notice that when we increase N, or in other words, the dimension of the input vector, we still get a similar phase transition curve. In addition, we still have the relation that the larger measurement rate, the higher the probability of exact recovery. And at the same time, the blurred region also tends to move to a larger value. But the shape of our boundary and the whole graph is still very similar. As the principle behind it is still the same.

For the second plot of average normalized error, the similar idea with the ESR, except that we focused on the NMSE (I usually refer to average Normalized Error as normalized mean square error when I was doing my previous research in compressed sensing area) of it. The sharp transition region is still when s_{max} is about 1 and M is about 1. And when after that when s_{max} and M continue grows, the transition region become blurred and we can not tell it very clearly. Quite obviously, the higher ESR, the lower aNE we tends to get. Because our recovery tends to become more accurate.

In a nutshell, Phase transition curve of aNE and probability exact support recovery have nearly the same form and shape as a whole. While from the definition it is quite clear that when the probability of exact support recovery is high, the value of average normalized error would be low.

4 Noisy case: (n $\neq$ 0)

4.1 (a)Assume that sparsity s is known. Implement OMP (terminate the algorithm after first s columns of A are selected). Generate "noisy phase transition" plots (for fixed N and σ) where success is defined as the event that the Normalized Error is less than 10^{-3}. Repeat the experiment for two values of σ (one small and one large) and choose N as $20, 50$ and 100. Comment on the results.

We choose σ with the small value 10^{-3} and the large value 10^{-2}, and in these cases the SNR is about 80dB and 60dB respectively (not a accurate value, because I followed the Experimental setup section and it only asked us to normalize $\boldsymbol{A}$ but did not ask us to create a normalized input sparse vector $\boldsymbol{x}$. While I myself used to normalize the vector so that I can get a quantitative SNR to make my experiment more comparable and evaluable [1]). BTW, the result of “noisy phase transition” plots for different N (I have tested N =20, 50, 80 and 100 four cases) and σ is shown as follows:

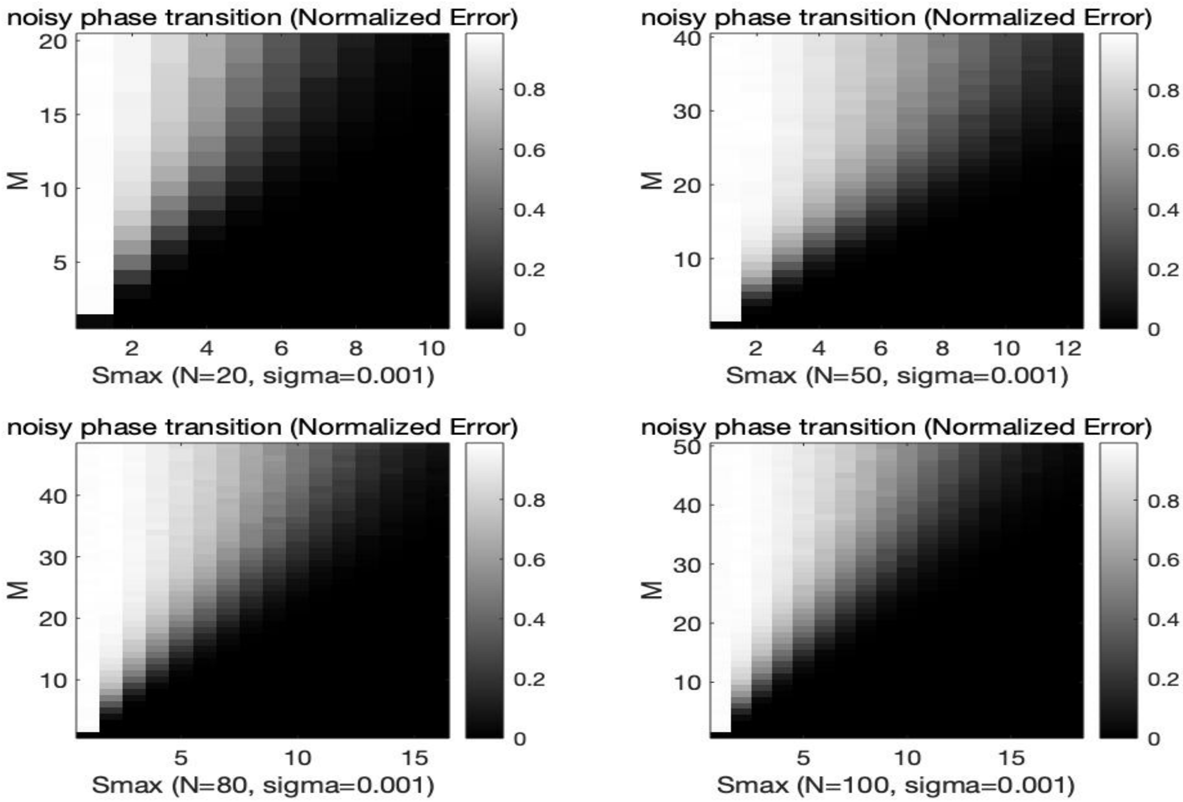

Figure 3: Success rate defined when Normalized Error is less than 10^{-3} ($\sigma = 0.001$)

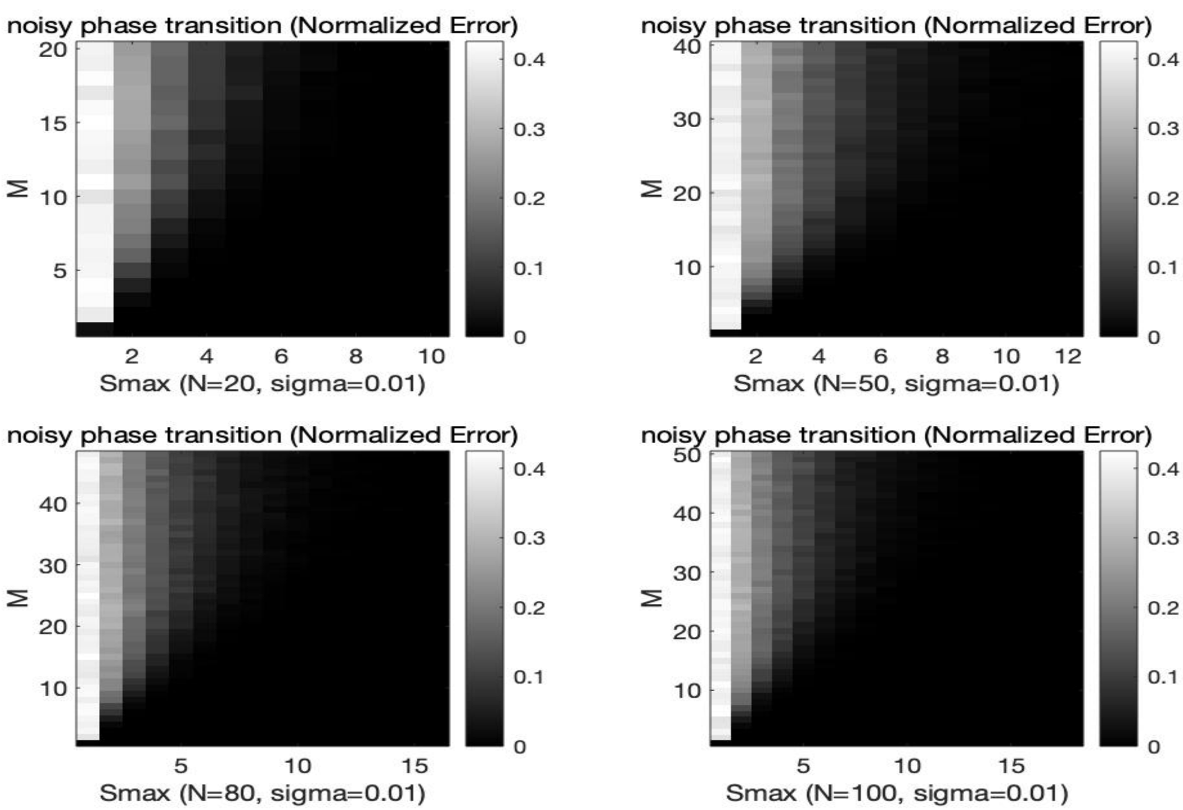

Figure 4: Success rate defined when Normalized Error is less than 10^{-3} ($\sigma = 0.01$).

Compared to the results we get in the last question, we can see that the existence of noise affects the performance of the recovery. As the noise can corrupt the signal that we measured. This leads to more blurred regions especially around the boundary between the successes part and the unsuccess part. Apat from that, the highest probability of success we could get in Figure 4 where $\sigma = 0.01$ is only about 0.5. And the larger the noise value, the harder that we can recover the vector, as we can see from Figure 3 and Figure 4 that the larger noise, the lower success rate generally. Last but not least, the relation between N, M and S_{max} is also similar as what we analyzed under the noiseless case question.

4.2 (b)Assume the sparsity s is NOT known, but $\|\mathbf{n}\|_2$ is known. Implement OMP where you may stop the OMP iterations once $\left\|\mathbf{y} - \mathbf{A}\mathbf{x}^{(k)}\right\|_2 \leq \|\mathbf{n}\|_2$). Generate phase transition plots using the same criterion for success as the previous part. Comment on the results.

Similar as the previous question, we choose σ with the small value 10^{-3} and the large value 10^{-4}. And the result of "noisy phase transition" plots for different N (we have tested 20, 50, 80 and 100) and σ is shown as follows:

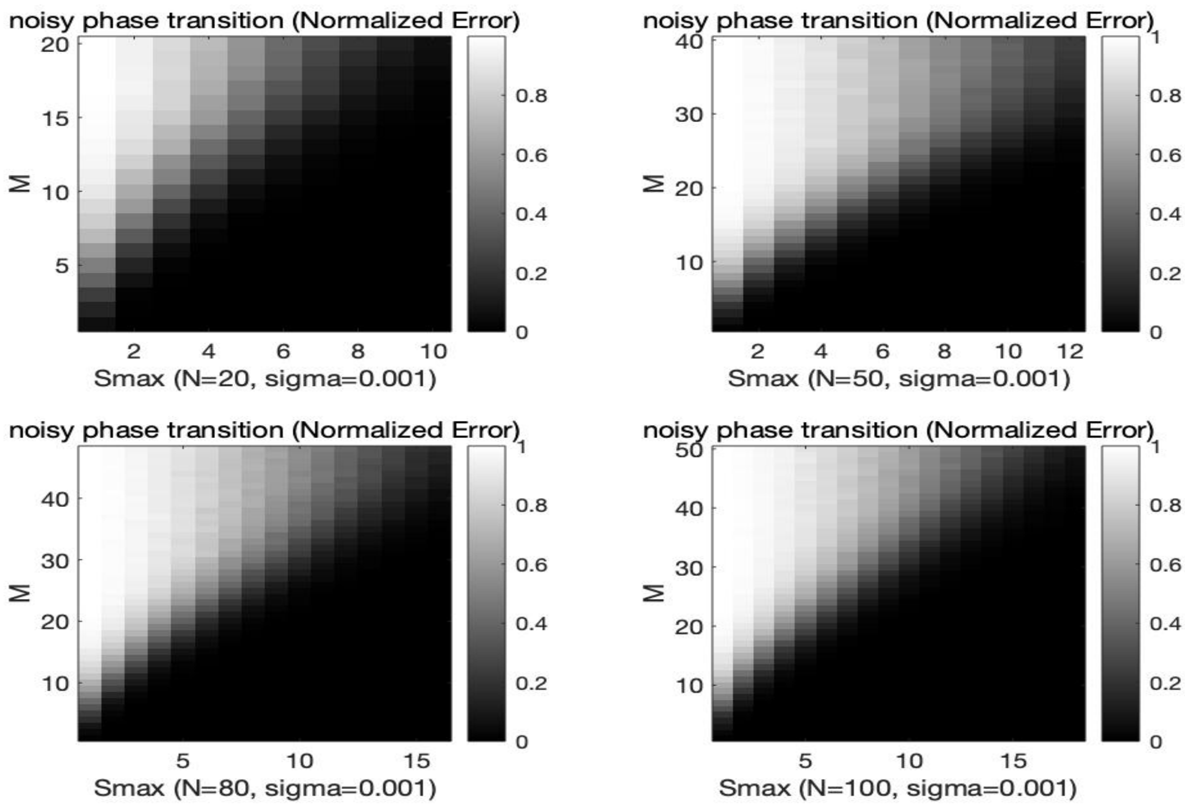

Figure 5: Success rate defined when Normalized Error is less than 10^{-3} ($\sigma = 0.001$)

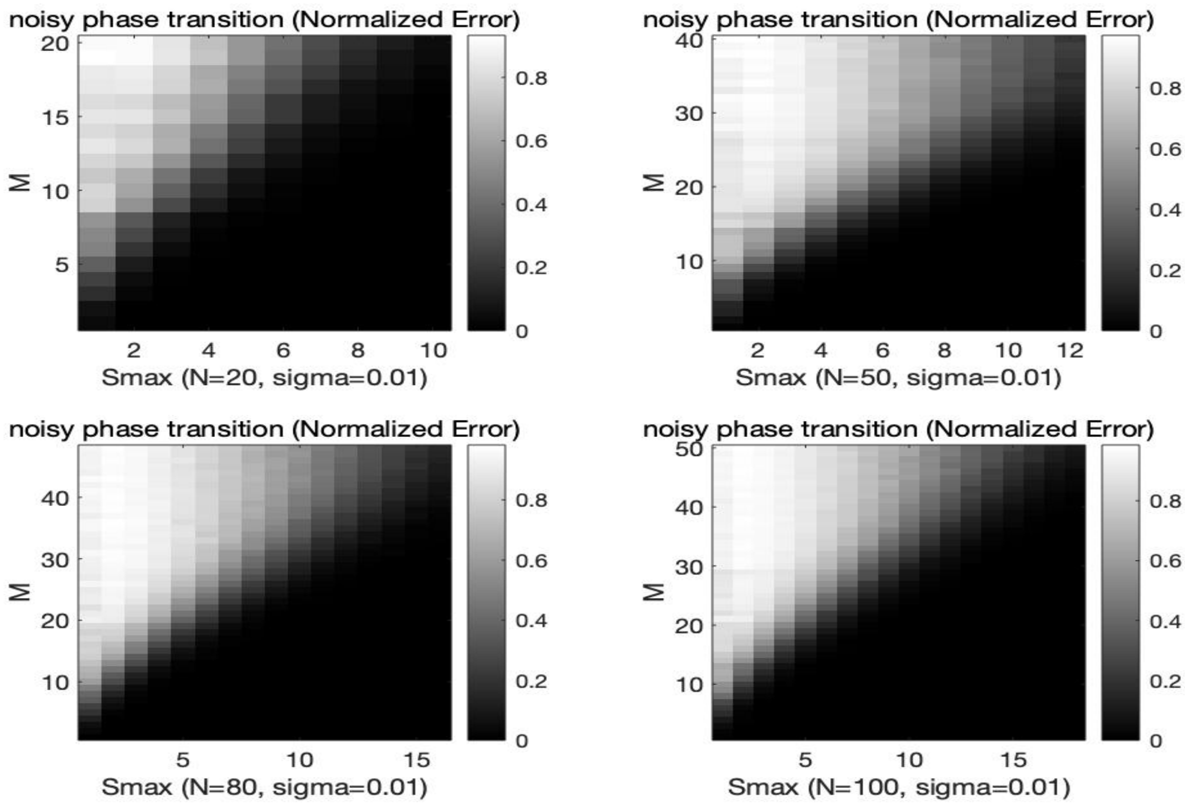

Figure 6: Success rate defined when Normalized Error is less than 10^{-3} ($\sigma = 0.01$)

In this case, when we have already known the prior information of the noise, that is $norm(\boldsymbol{n})$, then the effect of noise on the final performance of our algorithm will be reduced a lot. Compare Figure 5 and Figure 6, they look very similar, the probability of success recovery is very close. Similarly, it also shows a very alike shape and trend with plots in the last question. But the performance of the algorithm under different noise situation does not change so much. Last but not least, the relation between N, M and S_{max} is also similar as what we analyzed under the noiseless case question.

4.3 Design a numerical experiment to test OMP on real images. Describe your approach in detail about how you generate the measurement model, and comment on the quality of reconstructed images as you vary the number of measurements. What is the maximum compression (i.e the ratio of M/N) that still leads to (visually) satisfactory reconstruction? Show the reconstructed image for different values of M to justify your answer.

To simplify the model and avoid too much numerical experiment, we do not consider the existence of noise under this real image question. But it can be found in my previous paper [1] (I did simulations for CRPCA problem, even not exactly the same as CS, but for the sparse matrix part they have a lot in common) or refer to other works done by my lab [2], [3]. We did tons of simulations and compare under different noise condition, sparsity and measurement rates.

When it comes to the real image used in this question, the famous "lena" image with the size of 200×200 which is always used to do image processing is adopted. Then as for the way to generate the measurement model, at first we need to find a sparse representation of the input image to be estimated and reconstructed. This is what CS normally do. And actually there are many ways [2]. In this mini project, to simplify the work, we just took the DCT transformation. In detail, for an input image x, we have $x = \boldsymbol{D}s$ where $\boldsymbol{D}$ is the DCT matrix. After we obtain the matrix s, we zero those elements with small absolute values which are less than 0.2 to create the sparse matrix, which contribute to the final 0.13023 sparsity. Then the image x which obtained by multiplying $\boldsymbol{D}$ with s is shown below in Figure 7. As for how we generate the measurement model, we first of all corrupt the image with a measurement matrix $\boldsymbol{A}$ like what we did in the Experimental setup section ($\boldsymbol{A}$ is generated by with its independent and identically distributed entries drawn from the standard Gaussian distribution). Then we just use the OMP to reconstruct the representation of it s under DCT matrix, then multiply by the DCT matrix $\boldsymbol{D}$ we could obtain the reconstructed image.

From quality of reconstructed images as we vary the number of measurements, it can be seen that the larger M we measure, the better reconstructed image quality we can get. As for the maximum compression (i.e the ratio of M/N) that still leads to (visually) satisfactory reconstruction, we can seen from Figure 7 and 8 which show the input sparse image and the reconstructed image for different values of measurement rate respectively that it is about 0.39 measurement rate, which is like a experienced boundary point that I found in my previous work [1]. And since our input image is 200×200, so M is $200 \times 0.39 = 78$. Meanwhile, it could be seen that when M is larger than this boundary value, the image quality will not change too much because it will only affect the image details that we human are hard to tell. This also coincides to what we did in previous questions.

Figure 7: Original sparse image.

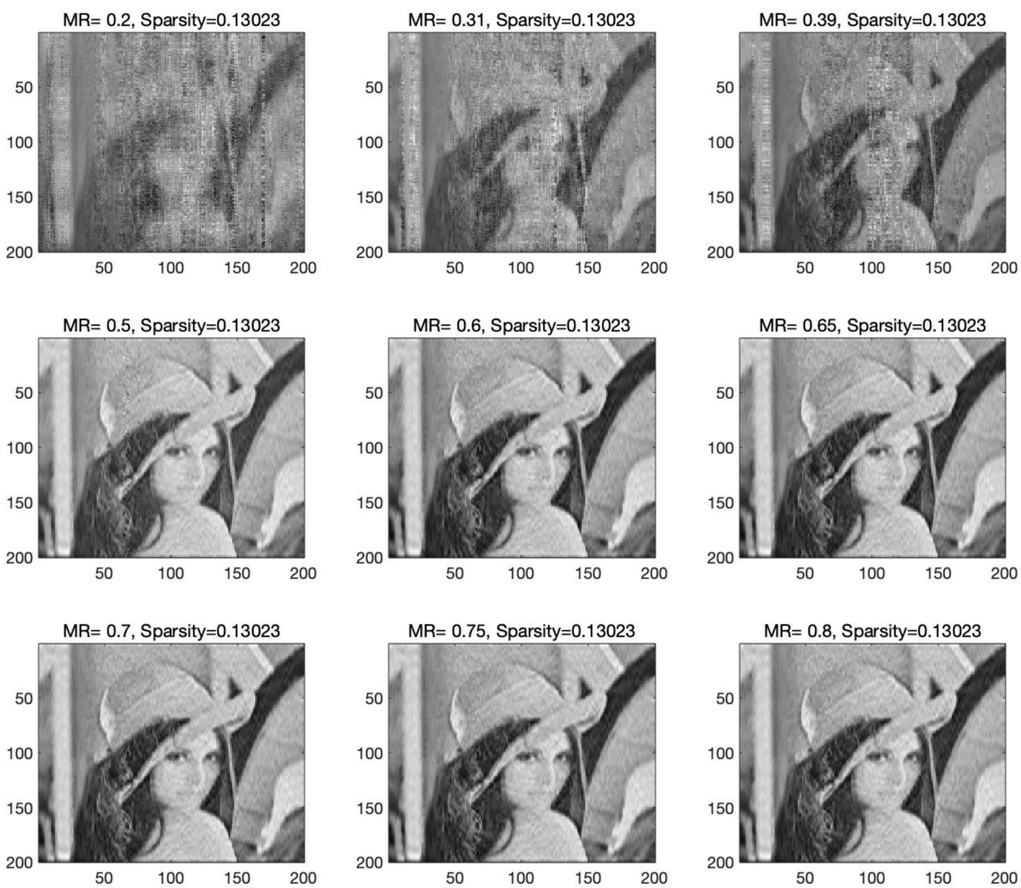

Figure 8: The reconstructed image under different ratio of M/N

5 Code

```
%%%%%%%%%%%%%%%%%%%%%%%%                   XUEHAI HE %%%%%%%%%%%%%%%%%%%%%%%%%%

clc;clear;
times=2000;

%%%%%%%%%%%%%%%%%%%%%%%%% (c) Noiseless case: (n = 0)%%%%%%%%%%%%%%%%%%%%%%%%%%


[ESR_matrix, aNE_matrix]= Monte_Carlo_runs(0, times, 1);


%%%%%%%%%%%%%%%%%%%%%%%%% (d) Noisy case: (n != 0) %%%%%%%%%%%%%%%%%%%%%%%

    sigma_list = [1e-3,1e-2];
    % two values of s (one small and one large)


%% a sparsity s is known

    for sigma= 1:2
        [ESR_matrix, aNE_matrix]= Monte_Carlo_runs(sigma_list(sigma), times, 1);
    end


%% b sparsity s is NOT known
    for sigma= 1:2
        [ESR_matrix, aNE_matrix]= Monte_Carlo_runs(sigma_list(sigma), times, 0);
    end



%% c test OMP on real images
lena_3 = imread('lena.jpg');
%to single channel
lena = lena_3(:,:,1);

%look at the single channel image
% imagesc(lena)

%normalization to [0,1] with double
x = im2double(lena);

%We find the DCT matrix for which the image x has a sparse representation
DCT=dctmtx(size(lena_3,1));

%create the sparse input image
s=DCT\x;
s(abs(s)<0.2) = 0;

figure;
%let's see the sparse input image
x_sparse=DCT*s;
imagesc(x_sparse);
colormap gray;
sparsity = sum(sum(s≠0))/numel(s);


figure;
measurement_rate = [0.2 0.31 0.39 0.5 0.6 0.65 0.7 0.75 0.8];
m = 200*measurement_rate;
for j = 1:length(measurement_rate)
    %corrupt the image with a random matrix A
    A = randn(m(j),200);
    s_hat = zeros(200,200);
    B=A*DCT;
    for i = 1:200
        yy = A*x_sparse(:,i);

        %do not add noise

        [s_hat(:,i),¬] = OMP1(B,yy);
    end
```

```
    subplot(3,3,j);
    imagesc(DCT*s_hat);
%     imagesc(idct2(x_hat));
    title(['MR= ' num2str(measurement_rate(j)) ', Sparsity=',num2str(sparsity)]);
    colormap gray

end



function [ESR_matrix, aNE_matrix]= Monte_Carlo_runs(sigma, times, sparsity_known)
    for n = 1:4
    %% d
        % Generate different phase transition plots for the following values of N: 20, 50 and 100.
        N_list=[20,50,80,100];
        % We generate for N: 20, 50, 80 and 100.
        %the average Normalized Error should be repeating step 1) to step 3) 2000 times and averaging the ...
            results over these 2000 Monte Carlo runs
        measurement_rate = [1,0.8,0.6,0.5];
        M_list=N_list.*measurement_rate;
        Sparsity_max = [0.5,0.24,0.2,0.18];
        S_list = N_list.*Sparsity_max;


        N=N_list(n);
        M_max=M_list(n);
        S_max=S_list(n);
%         M_max=35;
        % Smax is chosen to be a reasonably large integer which is smaller than N
%         S_max=0.3*N;
%         S_max=15;


        ESR_matrix = zeros(M_max,S_max);
        aNE_matrix = zeros(M_max,S_max);
        NE_matrix = zeros(M_max,S_max);

        for M = 1:M_max
            noise = randn(M,1);
            normnoise = norm(sigma*noise);


            for s = 1:S_max

                ESR=0;
                aNE=0;
                NE=0;
                for i=1:times
    %% a

    %%%%%%%%%%%%%%%%%%%%%%%(b)Experimental setup:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                    % Generate A as a random matrix with independent and identically distributed entries ...
                        drawn from the standard normal distribution

                    A = randn(M,N);

                    % Normalize the columns of A, not use in-built library functions.
                    A_vn = sqrt(diag(A'*A))';
                    A = A./A_vn;
                    % norm(A(:,1))

    %% b
                    % Generate the sparse vector x with random support of cardinality s

            %       s indices are
            %       generated uniformly at random from integers 1 to N
                    S = randi([1 N],s,1);
                    S = sort(S); % sort S
            %       non-zero entries drawn as uniform random variables
                    e = zeros(s,1);
                    for j = 1:s
                        U = randi([0 1]);
                        if U==1
                            e(j) = 1 + 9*rand;
```

```
                        elseif U==0
                            e(j) = -1 + (-9)*rand;
                        end
                    end
                    % generate the original signal x
                    x = zeros(N,1);
                    x(S) = e;
                    %sorted signal
                    SNR=norm(x)/sigma;
                    %calculate signal to noise ratio
                    SNR_dB=10*log(SNR);
    %% c

                    % Noiseless case(n = 0)

                    y = A(:,S)*e+sigma*noise;
                    %Noisy case: (n != 0)


                    if sigma==0 && sparsity_known==1
                        [x_hat,S_hat] = OMP1(A,y);

                   %Compute normalized error and exact support recovery for each iteration.
                        if length(S_hat) == s
                            if sum(sort(S_hat)==S)==s
                                ESR = ESR+1;
                            end
                        end
                        % identify ANE(average Normalized Error)
                        aNE = aNE + PM(x,x_hat);

                    elseif sigma≠0 && sparsity_known==1
                        x_hat = OMP_s(A,y,s);
                        % success is defined as the event that the Normalized Error is less than 10e-3
                        if PM(x,x_hat)≤1e-3
                            NE = NE + 1;
                        %Success add 1.
                        end

                    elseif sigma≠0 && sparsity_known==0
                        x_hat = OMP_n2(A,y,normnoise);
                        %success is defined as the event that the Normalized Error is less than 10e-3.
                        if PM(x,x_hat)≤1e-3
                            NE = NE + 1;
                        %Success add 1.
                        end

                    end

                end
                ESR_matrix(M,s)=ESR/times;
                aNE_matrix(M,s)=aNE/times;
                NE_matrix(M,s)=NE/times;
            end
        end

        if sigma==0
            if n==1
               figure;
            end
            %noiseless phase transition
            subplot(2,2,n);
            imagesc(ESR_matrix);
            colormap gray;
            title(['noiseless phase transition (ESR)']);
            xlabel(['Smax (N=', num2str(N), ', sigma=' ,num2str(sigma),')']);
            ylabel('M');
            colorbar;
            set(gca,'YDir','normal');

%             if n==1
%                figure;
%             end
%        %Regenerate phase transition plots for average Normalized Error(instead of probability of ...
    successful recovery
%             subplot(2,2,n);
```

```
%               imagesc(aNE_matrix);
%               colormap gray;
%               title('noiseless phase transition (AVE)');
%               xlabel(['Smax (N=', num2str(N), ', sigma=' ,num2str(sigma),')']);
%               ylabel('M');
%               colorbar;
%               set(gca,'YDir','normal');
            if n==1
                saveas(gcf,['ESR_noiseless_Smax (N=', num2str(N), '_sigma=' ,num2str(sigma),').png']);
            end
        else
%               figure;
            if n==1
               figure;
            end
            subplot(2,2,n);
            imagesc(NE_matrix);
            colormap gray;
            title('noisy phase transition (Normalized Error)');
            xlabel(['Smax (N=', num2str(N), ', sigma=' ,num2str(sigma),')']);
            ylabel('M');
            colorbar;
            set(gca,'YDir','normal');
            if n==1
                if sparsity_known==1
                    saveas(gcf,['S_noisy_Smax (N=', num2str(N), '_sigma=' ,num2str(sigma),').png']);
                elseif sparsity_known==0
                    saveas(gcf,['n_noisy_Smax (N=', num2str(N), '_sigma=' ,num2str(sigma),').png']);
                end
            end
        end
    end
end



function [x_hat, S_hat] = OMP1(A,y)
residue = y;
N = size(A,2);
Support = zeros(N,1);
k = 1;
residue_0 = 10000;

while (norm(residue_0)>1e-1) && (k≤N)
    ttt = abs(A'*residue);

    [¬,nn] = max(ttt);
    Support(k) = nn;
    col = A(:,Support(1:k));
    x1 =(col'*col)\(col'*y);
    residue = y - col*x1;
    residue_0 = residue;
    k = k+1;
end

x_hat = zeros(N,1);
x_hat(Support(1:(k-1))) = x1;
S_hat = Support(1:(k-1));

end


% function x_hat = OMP_s(A,y,s)
% % Assume that sparsity s is known
% %initialize support, index, residual
% residue = y;
% N = size(A,2);
% Support = zeros(N,1);
% residue_0 = 1;
% k = 1;
% while (k≤s) && ((residue_0'*residue_0) > 0.001)
%     %terminate the algorithm after first s columns of A are selected
%     ttt = abs(A'*residue);
%     [¬,nn] = max(ttt);
%     Support(k) = nn;
%     col = A(:,Support(1:k));
```

```
%     x1 =(col'*col)\(col'*y);
%     residue = y - col*x1;
%     residue_0 = residue;
%     k = k+1;
% end
%
% x_hat = zeros(N,1);
% x_hat(Support(1:(k-1))) = x1;
%
% end
%
%
% function x_hat = OMP_n2(A,y,norm)
% % the sparsity s is NOT known, but ||n||2 is known
% %initialize support, index, residual
% residue = y;
% N = size(A,2);
% Support = zeros(N,1);
% residue_0 = 20;
% k = 1;
% while residue_0>n
%     %stop the OMP iterations once ||y-Ax(k)||^2≤||n||2.
%     ttt = abs(A'*residue);
%     [¬,nn] = max(ttt);
%     Support(k) = nn;
%     col = A(:,Support(1:k));
%     x1 = linsolve(col'*col,col'*y);
%     residue = y - col*x1;
%     residue_0 = norm(residue);
%     k = k+1;
% end
%
% x_hat = zeros(N,1);
% x_hat(Support(1:(k-1))) = x1;
%
% end

%%%%%%%%%%%%%%%%%%%% (a) Performance Metrics: %%%%%%%%%%%%%%%%%%%%%%%%%%%
function NMSE = PM(x,x_hat)
NMSE = norm(x-x_hat)/norm(x);
end
```

References

[1] X. He, Z. Xue and X. Yuan, "Learned Turbo Message Passing for Affine Rank Minimization and Compressed Robust Principal Component Analysis," in IEEE Access, vol. 7, pp. 140606-140617, 2019.

[2] Z. Xue, J. Ma and X. Yuan, "Denoising-Based Turbo Compressed Sensing," in IEEE Access, vol. 5, pp. 7193-7204, 2017.

[3] J. Ma, X. Yuan and L. Ping, "Turbo Compressed Sensing with Partial DFT Sensing Matrix," in IEEE Signal Processing Letters, vol. 22, no. 2, pp. 158-161, Feb. 2015.

[4] J. A. Tropp, "Greed is good: algorithmic results for sparse approximation," in IEEE Transactions on Information Theory, vol. 50, no. 10, pp. 2231-2242, Oct. 2004.