University of California San Diego

Jacob School of Engineering

ECE 269: Linear Algebra

Orthogonal Matching Pursuit and Sparse Signal Recovery

December 25, 2019

Author	*Student ID*
Xingyi Yang	A53316455

Contents

List of Figures

Abstract

Sparse approximation is the problem to find the sparsest linear combination for a signal from a redundant dictionary, which is widely applied in signal processing and compressed sensing. In this project, I manage to implement the Orthogonal Matching Pursuit (OMP) algorithm for sparse signals and images approximation. Meanwhile, comprehensive experiments are conducted to analyses its performance under noiseless and noisy environments. The project's result shows the superiority of OMP algorithm under noiseless situation as well as revealing its vulnerability under noisy environments.

Keywords— Orthogonal Matching Pursuit (OMP), Sparse Approximation, Signal Recovery

1 Introduction

In the field of signal processing, it is the usual case to represent signal with a linear combination of set of m unit vectors. When those vectors are orthonormal, this approximation can be regarded as projection of signal onto the subspace spanned by elements of this m orthonormal basis. However, when those vectors are linear dependent, less than m vectors is needed since $m - r$ non-independent elements can be represent by the other r linear independent vectors.

It is easy to imagine that if m orthonormal basis can perfectly approximate a signal, surely $m + 1$ should not be worse. Though orthogonal basis is efficient in most case, a dictionary with linear dependent vectors can do even better. That's why sparse approximation is of great significance in signal processing.Under the assumption of a linear dependent set, or in other words, redundant dictionaries, finding the sparsest representation is called the sparse approximation problem.

One of the the biggest challenge for sparse approximation is the selection of atoms for the sparsest representation, which make sparse approximation a NP-hard problem[4].

In this project, I implement the Orthogonal Matching Pursuit (OMP) algorithm to recover the signal with random generated dictionary. I also show its robustness under noiseless and noisy situation. Section 2 will clarify some basic concept I use in the experiment, mainly focusing on the process of OMP algorithm. Section 3 covers the experiment setup, also demonstrating the statistical result of experiments. The python code are attached in the Appendix part for reference.

2 Methodology

2.1 Basic Concepts

Definition 2.1. *Dictionary*[2] A dictionary $D = \{\varphi\}_{l=1}^{N}$ in $\Re^M$ is a collection $D \subset \Re^M$ of unit-norm vectors. A dictionary has the following properties:

- The subscript of D is $\Omega = \{1, 2, \ldots, N\}$
- $\forall l \in \Omega, \|\varphi_l\|_2 = 1$
- Elements of D are called atoms.
- If D is linearly dependent, the dictionary is redundant.
- Cardinality refer to the the size of set, denoted by $|\cdot|$. Clearly $|D| = |\Omega| = N$.

Definition 2.2. *Sparse Approximation Problems* Given a dictionary $D = \{\varphi\}_{l=1}^{N}$ and $N > s > 1$. Assume $A = \begin{bmatrix}\varphi_1 & \varphi_2 & \ldots & \varphi_N\end{bmatrix}$, solve

$$\begin{aligned} &\arg\min_x \|y - Ax\|_2 \\ \text{s.t.} \quad &\|x\|_0 = s \end{aligned} \tag{1}$$

where y is the measurement, $A \in R^{M\times N}$ is the measurement matrix. $\|\cdot\|_0$ and $\|\cdot\|_2$ refers to the L_0 and L_2 norm respectively.

The indices of the non-zero entries of x is denoted by S, where $\forall i \in S, x_i \neq 0$. It is obvious that $|S| = \|x\|_0 = s$.

2.2 Matching Pursuit

If D contains a large number of vectors (N is large), solving for the most sparse representation of x is computationally unacceptable for practical applications.

In 1993, Mallat and Zhang [3] proposed a greedy solution address the sparse approximation problem, called Matching Pursuit(MP). For a given measurement y and a dictionary D, MP algorithm iteratively generates a sorted list of atom indices that correlates most strongly with the residual measurement, which form the suboptimal solution to the problem of sparse signal representation.

Algorithm 1 Matching Pursuit

Input: Measurement y, measurement matrix $A = \begin{bmatrix}\varphi_1 & \varphi_2 & \dots & \varphi_N\end{bmatrix}$
Output: Recovered signal x, indices for corresponding atoms indexes S
Initialization: $r_1 \leftarrow y$, $m \leftarrow 1$, $S \leftarrow \{\}$, $x \leftarrow zeroList(N)$

function MP
  while stop rule == False **do**
    Find $i_m^* \in \Omega$ that $i_n^* = \arg\max_i |\langle r_m, \varphi_i \rangle|$
    $S \leftarrow S \cup i_m^*$
    $x_{i_m^*} \leftarrow \langle r_m, \varphi_{i_m^*} \rangle$
    $r_{m+1} \leftarrow r_m - \langle r_m, \varphi_{i_m^*} \rangle \varphi_{i_m^*}$
    $m \leftarrow m + 1$
    return x, S

In Algorithm 1, r_m is the residual measurement at step n. $x_{i_n^*}$ is the coefficient corresponding to the $\varphi_{i_n^*}$ for the final linear combination term. MP is known as a Greedy Algorithm. At each step, MP only select the i_n^* atom with the greatest inner product value with the residual measurement. Th value of $x_{i_n^*}$ is merely relied the atom $\varphi_{i_n^*}$ with no respect on the global optimal solution.

2.3 Orthogonal Matching Pursuit

In this section we give a detailed description of the orthogonal matching pursuit (OMP) algorithm. For any subset $S \subset \Omega$, denote by $A(S)$ a submatrix of measurement matrix A consisting of the columns φ_i with $i \in S$. $x(S)$ is the subvector corresponding to $A(S)$. Thus, the OMP algorithm can be stated as follows.

Algorithm 2 Orthogonal Matching Pursuit

Input: Measurement y, measurement matrix $A = \begin{bmatrix}\varphi_1 & \varphi_2 & \dots & \varphi_N\end{bmatrix}$
Output: Recovered signal x, indices for corresponding atoms indexes S
Initialization: $r_1 \leftarrow y$, $m \leftarrow 1$, $S \leftarrow \{\}$, $x \leftarrow zeroList(N)$

function OMP
  while stop rule == False **do**
    Find $i_m^* \in \Omega$ that $i_n^* = \arg\max_i |\langle r_m, \varphi_i \rangle|$
    $S \leftarrow S \cup i_m^*$
    $x \leftarrow (A(S)^T A(S))^{-1} A(S)^T y$
    $r_{m+1} \leftarrow r_m - Ax$
    $m \leftarrow m + 1$
    return x, S

The OMP is also a stepwise forward selection algorithm and is easy to implement. Followed by the same atom selection criteria as MP method, OMP differ from MP in that, at each step, OMP computes the least square solution for $y = A(S)x$ that $x = (A(S)^T A(S))^{-1} A(S)^T y$. This least-squares minimization aims to

obtain the best approximation over the atoms that have already been chosen. A superiority of OMP is that it never selects the same atom twice because the residual is orthogonal to the atoms that have already been chosen.

Another key component of OMP is the stopping rule which depends on the noise structure. In the noiseless case the natural stopping rule is $r_m = 0$. In this project, we shall consider several different noise structures[1]. To be more specific, two types of noise condition are considered. One is that when the sparsity $s = |S|$ is known, the iteration stops when $n \geq s$. Another situation stands for unknown sparsity but known norm bound. Under the second condition, the searching for S stops when $\|r_m\|_2 \leq t$ where $r_m = y - Ax$. In addition, we would mainly consider the important case of Gaussian noise where $n \sim N(0, \sigma^2)$. The stopping rule for each case and the properties of the resulting procedure will be discussed in Section 3.

3 Experiment

3.1 Evaluation Metrics

Normalized Error Let $\hat{x}$ be the estimate of x obtained from OMP. To measure the performance of OMP, we consider the Normalized Error defined as

$$NormalizedError = \frac{\|x - \hat{x}\|_2}{\|x\|_2}$$

The average Normalized Error is obtained by averaging the Normazlized Error over 2000 Monte Carlo runs.

Peak Signal-to-noise Ratio Peak Signal-to-Noise ratio(PSNR) is widely used as a image quality metric. Given a noise-free $m \times n$ grayscale image I and its approximation $\hat{I}$, MSE is defined as:

$$MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - \hat{I}(i,j)]^2$$

The PSNR (in dB) is defined as:

$$PSNR = 10 \log_{10}(\frac{MAX_I^2}{MSE})$$

Here, MAX_I is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255.

Exact Recovery for OMP Without proof, a sufficient condition for OMP to recover the sparsest representation of the input signal is that

$$\max_{l \notin S} \|A(S)^+ \varphi_l\|_1 < 1$$

where $A(S)^+ = (A(S)^H A(S))^{-1} A(S)^H$

3.2 Experimental Setup

In this project, we explore the performance and robustness of OMP algorithm under noiseless and noisy situation. Thus we design 6 set of experiments. The experiment details are elaborated below.

- Noiseless case
    1. **Noiseless phase transition plot with Exact Support Recovery probability**

With each $N \in \{20, 50, 100\}$, I vary M and s_{max} and calculate the probability of Exact Support Recovery by averaging over 2000 random realizations of $A \in \Re^{M \times N}$ and $x \in \Re^N$ with s non-zero elements, where $s \sim U(1, s_{max})$. The measurement y is computed as

$$y = Ax$$

In my implementation, $M \in \{10, 15, 20, \ldots, 95\}$ and $s_{max} \in \{\lfloor \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rceil + 3, \lfloor \frac{N}{2} \rceil + 6, \ldots\} (\max s_{max} < N)$. $\lfloor \cdot \rceil$ refers to the round operation. Stop iteration when $\|r_m\|_2 \leq \epsilon = 0.001$.

2. **Noiseless phase transition plots with Normalized Error**

With fixes $N \in \{20, 50, 100\}$, I vary M and s_{max} and calculate the Normalized Error by averaging over 2000 random realizations of $A \in \Re^{M \times N}$ and $x \in \Re^N$ with s non-zero elements, where $s \sim U(1, s_{max})$. The measurement y is computed as

$$y = Ax$$

In my implementation, $M \in \{10, 15, 20, \ldots, 95\}$ and $s_{max} \in \{\lfloor \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rceil + 3, \lfloor \frac{N}{2} \rceil + 6, \ldots\} (\max s_{max} < N)$. Stop iteration when $\|r_m\|_2 \leq \epsilon = 0.001$.

- Noisy case

1. **Noisy phase transition with Normalized Error, known sparsity**

With each $N \in \{20, 50, 100\}$, I vary M and s_{max} and calculate the probability of Successful Recovery ($NormalizedError < 0.001$) by averaging over 2000 random realizations of $A \in \Re^{M \times N}$ and $x \in \Re^N$ with s non-zero elements, where $s \sim U(1, s_{max})$. The noise vector is drawn form $N(0, \sigma^2)$, with $\sigma \in \{0.001, 0.01\}$. The measurement y is computed as

$$y = Ax + n$$

In my implementation, $M \in \{10, 15, 20, \ldots, 95\}$ and $s_{max} \in \{\lfloor \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rceil + 3, \lfloor \frac{N}{2} \rceil + 6, \ldots\} (\max s_{max} < N)$. $\lfloor \cdot \rceil$ refers to the round operation. Stop iteration when $m = s$.

2. **Noisy phase transition with Normalized Error, known noise norm**

With each $N \in \{20, 50, 100\}$, I vary M and s_{max} and calculate the probability of Successful Recovery ($NormalizedError < 0.001$) by averaging over 2000 random realizations of $A \in \Re^{M \times N}$ and $x \in \Re^N$ with s non-zero elements, where $s \sim U(1, s_{max})$. The noise vector is drawn form $N(0, \sigma^2)$, with $\sigma \in \{0.01, 0.2\}$. The measurement y is computed as

$$y = Ax + n$$

In my implementation, $M \in \{10, 15, 20, \ldots, 95\}$ and $s_{max} \in \{\lfloor \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rceil + 3, \lfloor \frac{N}{2} \rceil + 6, \ldots\} (\max s_{max} < N)$. $\lfloor \cdot \rceil$ refers to the round operation. Stop iteration when $\|r_m\|_2 \leq \|n\|_2$.

- Image case

1. **Noiseless image recovery**

The original image is processed by Discrete Cosine Transform(DCT) with 8×8 blocks. Each 8×8 DCT block is vectorize to a 64-dimension vector $x^{(i)}$ with zigzag pattern. Assume $x^{(i)} \in \Re^{64}$ as the ground-truth signal ($N = 64$). I vary M and randomly construct a $A \in \Re^{M \times 64}$ for the whole image at a time. The measurement $y^{(i)}$ for each vector is computed as

$$y^{(i)} = Ax^{(i)}$$

Each reconstructed signal $\hat{x}^{(i)}$ is reshaped into 8×8 and inverse discrete cosine transformed (IDCT) to a image block. I repeat the image reconstruction over 100 times and calculate the Normalized Error and PSNR to evaluate the recovery performance of OMP algorithm. In my implementation, $M \in \{10, 15, 20, \ldots, 95\}$ and the Stop rule is $\|r_m\|_2 \leq \epsilon = 0.001$.

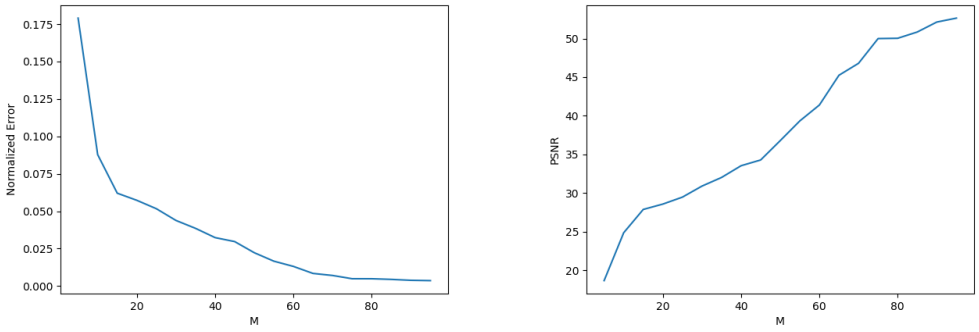
2. **Noisy image recovery**

   The original image is processed by Discrete Cosine Transform(DCT) with 8×8 blocks. Each 8×8 DCT block is vectorize to a 64-dimension vector $x^{(i)}$ with zigzag pattern. Assume $x^{(i)} \in \Re^{64}$ as the ground-truth signal ($N = 64$). I vary M and randomly construct a $A \in \Re^{M \times 64}$ for the whole image at a time. The noise vector $n^{(i)}$ is drawn form $N(0, \sigma^2)$, with $\sigma \in \{0.001, 0.01\}$. The measurement $y^{(i)}$ for each vector is computed as

$$y^{(i)} = Ax^{(i)} + n^{(i)}$$

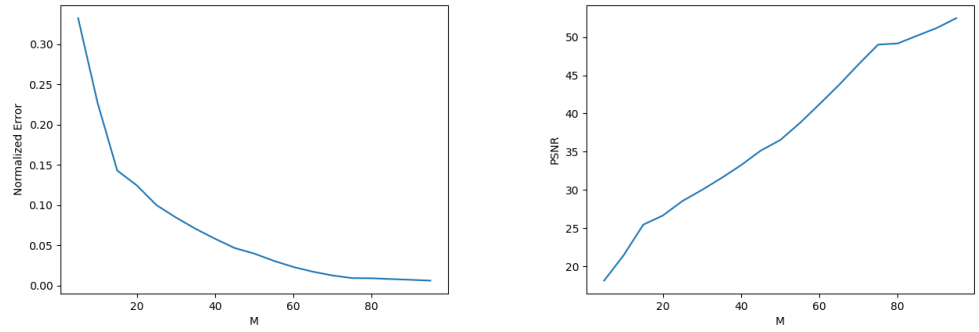
   Each reconstructed signal $\hat{x}^{(i)}$ is reshaped into 8×8 and inverse discrete cosine transformed (IDCT) to a image block. I repeat the image reconstruction over 100 times and calculate the Normalized Error and PSNR to evaluate the recovery performance of OMP algorithm. In my implementation, $M \in \{10, 15, 20, \ldots, 95\}$ and the Stop rule is $\|r_m\|_2 \leq \|n\|_2$.

The two experiment under Noiseless Case study the relation between recovery performance and various M and s_{max}. Two Noisy experiments compare the robustness of OMP algorithm under different noise degree and with different stopping rule. Image recovery tests the ability of OMP method in terms of practical application of image compression and reconstruction.



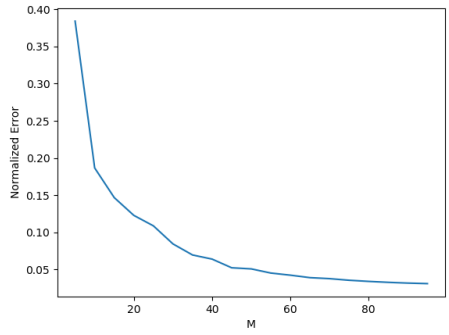
(a) Normalized Error (b) PNSR

Figure 1: Dog: Noiseless Normalized Error curve and PNSR curve

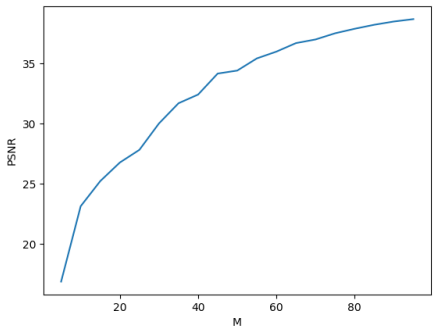


(a) Normalized Error (b) PNSR

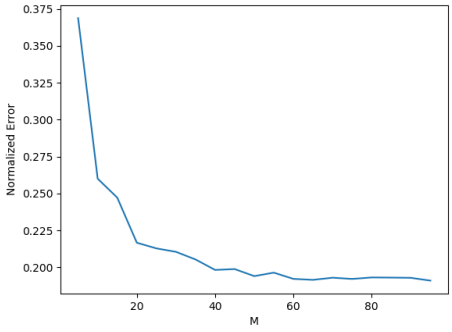
Figure 2: Marilyn Monroe: Noiseless Normalized Error curve and PNSR curve



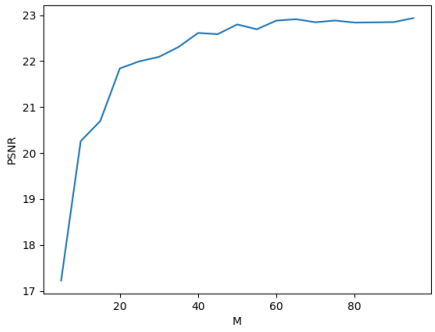
(a) Normalized Error, $\sigma = 0.01$



(b) PNSR, $\sigma = 0.01$



(c) Normalized Error, $\sigma = 0.2$



(d) PNSR, $\sigma = 0.2$

Figure 3: Marilyn Monroe: Noisy Normalized Error curve and PNSR curve

3.3 Main Result

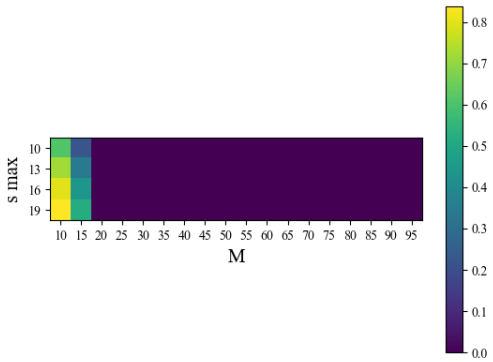
3.3.1 Noiseless Case

The Noiseless phase transition plot with $N \in \{20, 50, 100\}$ are displayed in Fig.4. Under the noiseless situation, we can see that Normalized Error sharply drops with the increase of both M and s_{max}, where as the the success probability of Exact Support Recovery gradually reach to 1. We may observe a sharp transition region where both the probability of ESR and average Normalized Error quickly changes. This transition approximately happens around $M = N$. When $N > M$ the normalized error tends to be large and the successful probability is quite low. This phenomenon results in that, for Fig.4(a)4(b) where $N = 20$, most of the experiments are of good performance. However from Fig.4(e)4(f), The recovery result for large N is poor.

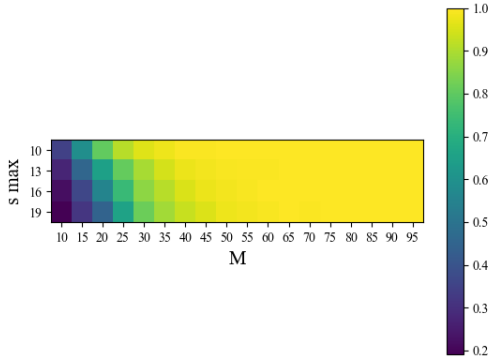
3.3.2 Noisy Case

It is the usual case that that a signal may suffer from unwanted noise during capture, storage, transmission, processing, or conversion. In this section, we intentionally add noise to the measurement with different assumptions to testify the robustness of the OMP algorithm.

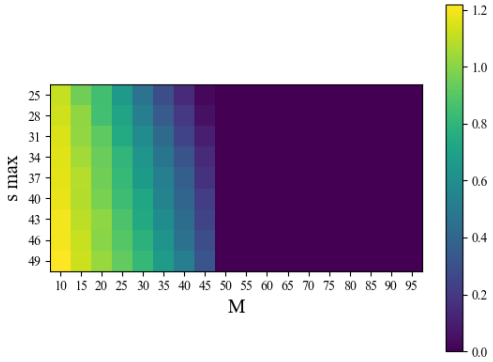
Noisy measurement with known sparsity Under the assumption of known sparsity, the iterations for atom selection stop when $m \geq s$. The probability of successful recovery under different M and s_{max} is shown in Fig.5. With small $\sigma = 0.001$, the OMP algorithm still performs considerably robustly, resembling the noiseless situation. In Fig.5(a), when $M >> N$, most signals can be successfully reconstructed with probability above 0.9. However, when the noise's variance grows to 0.01, the recovery results are messed



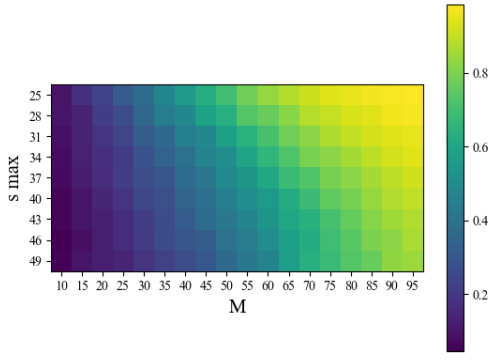
(a) Normalized Error N=20



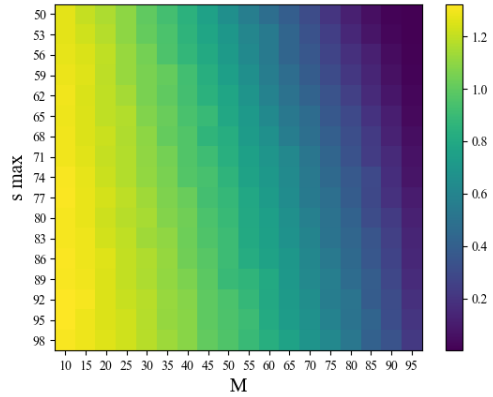
(b) Probability of Exact Support Recovery N=20



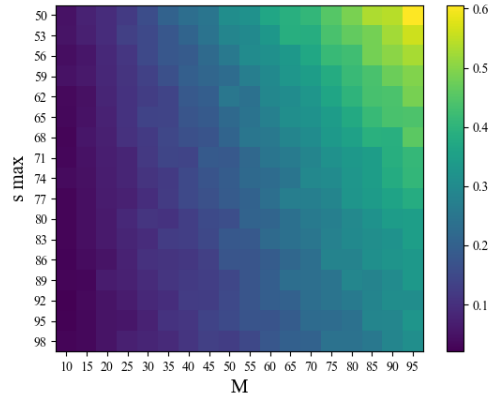
(c) Normalized Error N=50



(d) Probability of Exact Support Recovery N=50



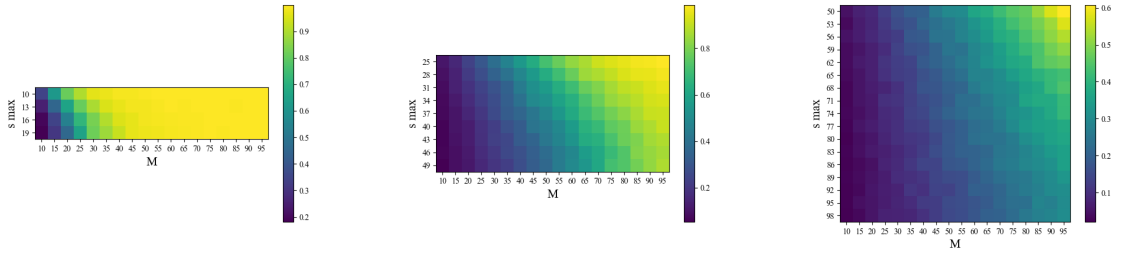
(e) Normalized Error N=100



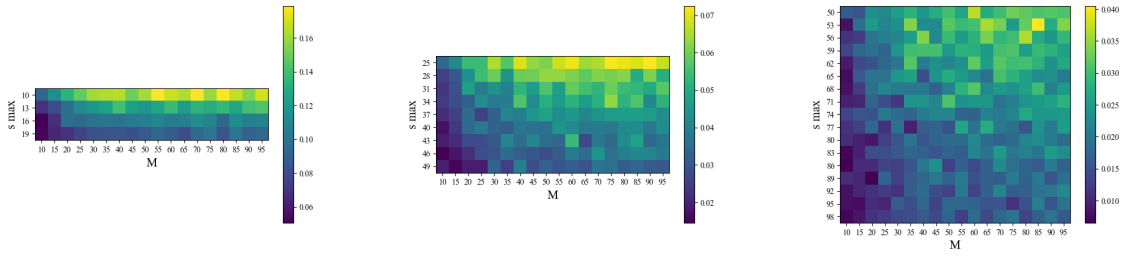
(f) Probability of Exact Support Recovery N=100

Figure 4: Noiseless phase transition plot with different N

up. As we may notice in Fig.5(d)5(e)5(f), no matter how M increases, the successful rate is under 0.2. This shows that OMP is vulnerable in severe noisy environment.

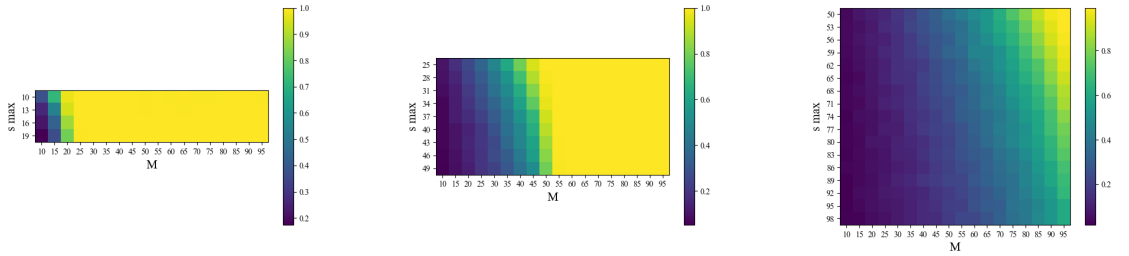


(a) $\sigma = 0.001$ and $N = 20$ (b) $\sigma = 0.001$ and $N = 50$ (c) $\sigma = 0.001$ and $N = 100$

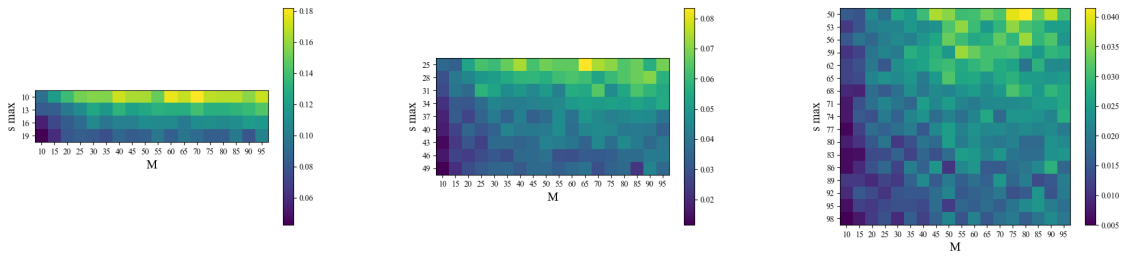


(d) $\sigma = 0.01$ and $N = 20$ (e) $\sigma = 0.01$ and $N = 50$ (f) $\sigma = 0.01$ and $N = 100$

Figure 5: Noisy phase transition plot, known sparsity



(a) $\sigma = 0.001$ and $N = 20$ (b) $\sigma = 0.001$ and $N = 50$ (c) $\sigma = 0.001$ and $N = 100$



(d) $\sigma = 0.01$ and $N = 20$ (e) $\sigma = 0.01$ and $N = 50$ (f) $\sigma = 0.01$ and $N = 100$

Figure 6: Noisy phase transition plot, known noise norm

Noisy measurement with known noise norm Under the assumption of known noise norm, the iterations for atom selection stop when $\|r_m\|_2 \leq \|n\|_2$. The general consequence is similar with the result obtained with the known sparsity assumption. When σ is small in Fig.6(a)6(b)6(c), the OMP method is robust enough to recover the slightly noisy signal. But large σ could disable the algorithm completely.

Another difference is that, the successful possibility when noise norm is known marginally outperform result when sparsity is known. This prove that stop rule with respect to the noise norm can take advantage of the noise information, which is beneficial for the sparse approximation problem using OMP.

3.3.3 Image Case

In addition to the experiments with randomly generated 1D signals, the sparse approximation problem with real image is of great practicality. With real images divided into 8×8 image blocks, we adopt DCT and vectorize each block to 64-dimension vector. With different choice of M, the quality of the recovered image varies. We study the image quality under noiseless and noisy situation.

Noiseless image recovery The recovered images without random noise are visualized in Fig.7 and Fig.8. And the Normalized Error Curve and PSNR curve are shown in Fig.1 and Fig.2. For both example images of "Dog" and "Marilyn Monroe", some common conclusion can be drawn.

1. The image quality promotes with the the increase of M. When $5 \leq M \leq 35$, the visual quality improvement in term of both Normalized Error and PSNR is very rapid. When $45 \leq M \leq 65$, the improvement of Normalized Error becomes marginal. When $M > 65$, the Normalized Error curve stabilize and hardly improves.
2. From my experiments, the maximum compression ratio is approxmately $20/64$, when $M = 20$.

Noisy image recovery Without loss generality, we explore the image reconstruction performance with noisy DCT vectors under Gaussian noise. The visualization of reconstructed "Marilyn Monroe" is shown in Fig.9 ($\sigma = 0.01$) and Fig.10 ($\sigma = 0.2$). Their corresponding Normalized Error Curve and PNSR Curve are displayed in Fig.3. Some finding can be reached through the experiments:

1. In general, the tendency of image reconstruction with noisy data resembles that with noiseless data. The Normalized Error decreases and PNSR increased with the growth of M.
2. The recovered image quality is severely influence by the degree of the noise. For $\sigma = 0.01$, the PSNR up-bound for the reconstructed image is about 40 dB and reconstructed image still looks clear and sharp. However, the limit of the image PSNR under $\sigma = 0.2$ dramatically drops to 23 dB. The visualization give the same result that under the situation of $\sigma = 0.2$, the facial details can be vague and indistinguishable for human.

4 Conclusion

With comprehensive experiments and statistical analysis of the performance of Orthogonal Matching Pursuit algorithm under (1) Noiseless and (2) Noisy condition and on (1) Artifical Signals and (2) Real Images, we can draw the conclusion that OMP is generally a algorithm of great performance when the compression ratio M/N is large enough. Under noisy condition, good choice of stopping rules can further improves the robustness of the algorithm. However, OMP is not a universal solution for sparse approximation problem under severely noisy environments. Large Gaussian Noise with big L_2 norm and large variance could lead to failure of signal recovery. Further works should focus on improving the robustness of OMP algorithm.

References

[1] T Tony Cai and Lie Wang. Orthogonal matching pursuit for sparse signal recovery with noise. Institute of Electrical and Electronics Engineers, 2011.

[2] Anna C. Gilbert. An introduction to sparse approximation. Department of Mathematics University of Michigan.

[3] Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415, 1993.

[4] Joel A Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information theory*, 50(10):2231–2242, 2004.

Appendix

```python
import numpy as np
import random
import scipy

def define_A(M,N):
    '''Define the measurement matrix
    :args
    M: number of row
    N: number of column
    :returns
    A: M x N matrix with normalized columns drawn from N(0,1)
    '''
    A = []
    for i in range(N):
        col = np.random.randn(M)
        col /= norm2(col)

        A.append(col)
    A = np.array(A).T
    return A

def define_x(s,N):
    '''Define the real signal
    :args
    s: random support of cardinality
    N: size of x
    :return
    x: real signal
    '''
    # Uniformly sample the index

    index = random.sample(range(N),s)
    x = np.zeros(N)
    for idx in index:
        # For each index, draw a number from [-10,-1]||[1,10]
        r = np.random.rand()
        if r >0.5:
            x[idx]=(r-0.5)*18+1
        else:
            x[idx]=(r-0.5)*18-1
    return x.T,index

def OMP(A,y,N,stop=np.infty,r_thresh=0.01):
    '''Orthogonal Matching pursuit algorithm
    :args
    A: measurement matrix
    y:
    '''
    r = y
    x_pre = np.zeros(N)
    Lambdas = []
    i = 0
    # Control stop interation with norm thresh or sparsity
    while norm2(r)>r_thresh and i<stop:

        # Compute the score of each atoms
        scores = A.T.dot(r)

        # Select the atom with the max score
        Lambda = np.argmax(abs(scores))
        # print(Lambda)
```

```
        Lambdas.append(Lambda)

        # All selected atoms form a basis
        basis = A[:,Lambdas]

        # Least square solution for y=Ax
        x_pre[Lambdas] = np.linalg.inv(np.dot(basis.T,basis)).dot(basis.T).dot(y)

        # Compute the residual
        r = y - A.dot(x_pre)

        i += 1
    return x_pre.T,Lambdas

def norm2(x):
    return np.sqrt(np.sum([i**2 for i in x]))

def norm1(x):
    return np.sum([abs(i) for i in x])

def Normalized_Error(x,x_pre):
    return norm2(x-x_pre)/norm2(x)

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

def dct2(a):
    return scipy.fftpack.dct( scipy.fftpack.dct( a, axis=0, norm='ortho' ), axis=1, norm='ortho' )

def idct2(a):
    return scipy.fftpack.idct( scipy.fftpack.idct( a, axis=0 , norm='ortho'), axis=1 , norm='ortho')
```

Listing 1: Orthogonal Matching pursuit Algorithm

```
from OMP import *
import numpy as np
from tqdm import tqdm
import time
# ! Problem 1, OMP with no noise

class Experiment:
    def __init__(self,M=range(10,100,5),N=[20,50,100]):
        self.M = M
        self.N = N

    def define_all(self,M,N,s):
        A = define_A(M,N)
        x,index = define_x(s,N)
        y = A.dot(x)
        return A,x,y,index

    def run_exp_NormError(self,times = 2000):

        for N in self.N:
            s_max_range = range(N//2,N,3)
            len_M = len(self.M)
            len_s_max = len(s_max_range)
            transition_map = np.zeros((len_s_max,len_M))
            print(transition_map.shape)
            for M_id in range(len_M):
                M = self.M[M_id]
                for s_max_id in range(len_s_max):
                    s_max = s_max_range[s_max_id]
                    e = 0
                    for e_id in tqdm(range(times)):
                        s = np.random.randint(low=1,high = s_max)
                        A,x,y,index = self.define_all(M,N,s)
                        x_pre,Lambdas = OMP(A,y,N)
```

```
                        e += Normalized_Error(x,x_pre)
                    e /= times
                    transition_map[s_max_id,M_id]=e
                    print("Normalized_Error for M={} and N={} {}".format(M,N,e))

            np.save("N{}.npy".format(N),transition_map)
            print("N={} finished".format(N))

    def run_exp_ESR(self,times = 2000):

        for N in self.N:
            s_max_range = range(N//2,N,3)
            len_M = len(self.M)
            len_s_max = len(s_max_range)
            succes_map = np.zeros((len_s_max,len_M))
            print(succes_map.shape)
            for M_id in range(len_M):
                M = self.M[M_id]
                for s_max_id in range(len_s_max):
                    s_max = s_max_range[s_max_id]
                    t = 0
                    for e_id in tqdm(range(times)):
                        s = np.random.randint(low=1,high=s_max)
                        A,x,y,true_index = self.define_all(M,N,s)
                        x_pre,pred_index = OMP(A,y,N,r_thresh=0.001)

                        if set(true_index)==set(pred_index):
                            t+=1
                    t=t/times
                    succes_map[s_max_id,M_id] = t
                    print("ESR rate for M={} and N={} {}".format(M,N,t))

            np.save("N{}_ESR_rate.npy".format(N),succes_map)
            print("N={} finished".format(N))

    def run_exp_Normtrans(self,times = 2000):

        for N in self.N:
            s_max_range = range(N//2,N,3)
            len_M = len(self.M)
            len_s_max = len(s_max_range)
            succes_map = np.zeros((len_s_max,len_M))
            print(succes_map.shape)
            for M_id in range(len_M):
                M = self.M[M_id]
                for s_max_id in range(len_s_max):
                    s_max = s_max_range[s_max_id]
                    t = 0
                    for e_id in tqdm(range(times)):
                        s = np.random.randint(low=1,high = s_max)
                        A,x,y,index = self.define_all(M,N,s)
                        x_pre,Lambdas = OMP(A,y,N)
                        e = Normalized_Error(x,x_pre)
                        if e < 0.001:
                            t += 1
                    t /= times
                    succes_map[s_max_id,M_id] = t
                    print("Normalized_Error_propability for M={} and N={} {}".format(M,N,t))

            np.save("N{}_Normalized_Error_propability_trans_plot.npy".format(N),succes_map)
            print("N={} finished".format(N))

if __name__ == "__main__":
    exp = Experiment()
    exp.run_exp_ESR()
```

Listing 2: Noiseless Experiment

```python
from baseExp import Experiment
from OMP import *
import numpy as np
from tqdm import tqdm

# ! Problem 2, OMP with noise

class NoiseExperiment(Experiment):
    def __init__(self,sigma=0.001,M=range(10,100,5),N=[20,50,100]):
        '''
        :args
        sigma: the std of the noise

        '''
        super(NoiseExperiment,self).__init__
        self.sigma = sigma
        self.M = M
        self.N = N


    def define_noise_all(self,M,N,s):
        A = define_A(M,N)
        x,index = define_x(s,N)
        y = A.dot(x)
        n = np.random.normal(0, self.sigma, M)
        y += n.T

        return A,x,y,index,norm2(n)

    def run_exp_Normtrans1(self,times = 2000):
        '''Noise experiment 1: known sparsity'''
        for N in self.N:
            s_max_range = range(N//2,N,3)
            len_M = len(self.M)
            len_s_max = len(s_max_range)
            succes_map = np.zeros((len_s_max,len_M))
            print(succes_map.shape)
            for M_id in range(len_M):
                M = self.M[M_id]
                for s_max_id in range(len_s_max):
                    s_max = s_max_range[s_max_id]
                    t = 0
                    for e_id in tqdm(range(times)):
                        s = np.random.randint(low=1,high = s_max)
                        A,x,y,index,norm = self.define_noise_all(M,N,s)
                        x_pre,Lambdas = OMP(A,y,N,stop=s)
                        temp_e = Normalized_Error(x,x_pre)

                        if temp_e<0.001:
                            t+=1
                    t = t/times
                    succes_map[s_max_id,M_id] = t
                    print("Normalized_Error_propability for M={} and N={} {}".format(M,N,t))

            np.save("Noise1_N{}_sigma{}.npy".format(N,self.sigma),succes_map)
            print("N={} finished".format(N))

    def run_exp_Normtrans2(self,times = 2000):
        '''Noise experiment 2: unknown sparsity, known noise norm'''
        for N in self.N:
            s_max_range = range(N//2,N,3)
            len_M = len(self.M)
            len_s_max = len(s_max_range)
            succes_map = np.zeros((len_s_max,len_M))
            print(succes_map.shape)
            for M_id in range(len_M):
                M = self.M[M_id]
                for s_max_id in range(len_s_max):
```

```
                    s_max = s_max_range[s_max_id]
                    t = 0
                    for e_id in tqdm(range(times)):
                        s = np.random.randint(low=1,high = s_max)
                        A,x,y,index,norm = self.define_noise_all(M,N,s)
                        x_pre,Lambdas = OMP(A,y,N,r_thresh=norm)
                        temp_e = Normalized_Error(x,x_pre)

                        if temp_e<0.001:
                            t+=1
                    t = t/times
                    succes_map[s_max_id,M_id] = t
                    print("Normalized_Error_propability for M={} and N={} {}".format(M,N,t))

            np.save("Noise2_N{}_sigma{}.npy".format(N,self.sigma),succes_map)
            print("N={} finished".format(N))



if __name__ == "__main__":
    exp = NoiseExperiment()
    exp.run_exp_Normtrans2()
```

Listing 3: Noisy Experiment

```
from OMP import *
import numpy as np
import os
from tqdm import tqdm
import time
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from scipy import fftpack
from skimage.transform import resize
from skimage.metrics import peak_signal_noise_ratio as PSNR
# ! Problem 3, image compression

class ImageExperiment:
    def __init__(self,M=range(5,100,5)):
        self.sigma = 0.2
        self.M = M
        self.zigzag=np.array(open('Zig-Zag Pattern.txt').read().split()).astype(np.int).reshape(8,8)
        self.expname = "marilynmonroe210-2"
        self.imsize = (400, 320)
        self.norm_error =[]
        self.psnrs = []


    def dct8x8(self,img):
        '''Do 8x8 DCT on image (in-place)'''
        dct = np.zeros_like(img)
        for i in range(0,img.shape[0],8):
            for j in range(0,img.shape[1],8):
                dct[i:(i+8),j:(j+8)] = dct2(img[i:(i+8),j:(j+8)])
        return dct

    def idct8x8(self,dct):
        '''8x8 DCT to image'''
        dct_img = np.zeros_like(dct)

        for i in range(0,dct.shape[0],8):
            for j in range(0,dct.shape[1],8):
                dct_img[i:(i+8),j:(j+8)] = idct2(dct[i:(i+8),j:(j+8)])
        return dct_img

    def block2zigzag(self,block):
        vector = np.zeros(64)
        vector[self.zigzag] = block
```

```
        return vector

    def zigzag2block(self,vector):
        block = np.zeros((8,8))
        block = vector[self.zigzag]
        return block

    def dct2zigzag(self,dct):
        h,w = dct.shape
        block_h,block_w = (h//8),(w//8)
        zigzag_vector = np.zeros((block_h*block_w,64))

        for i in range(0,h,8):
            for j in range(0,w,8):
                zigzag_vector[(i//8)*block_w+j//8] = self.block2zigzag(dct[i:(i+8),j:(j+8)])
        return zigzag_vector

    def zigzag2dct(self,zigzag_vector,imsize):
        h,w = imsize
        block_h,block_w = (h//8),(w//8)
        dct = np.zeros((h,w))
        for i in range(0,h,8):
            for j in range(0,w,8):
                dct[i:(i+8),j:(j+8)] = self.zigzag2block(zigzag_vector[(i//8)*block_w+j//8])
        return dct

    def run_image(self):
        foldername = "experiment/exp_{}/".format(self.expname)
        if os.path.exists(foldername)==False:
            os.mkdir(foldername)
        img = plt.imread('images/menglu.jpg')/255

        img = rgb2gray(img)
        img = resize(img, self.imsize)

        # Do 8x8 DCT on image and vectorized with zigzag pattern
        dct = self.dct8x8(img)
        zigzag_vector = self.dct2zigzag(dct)
        zigzag_shape = zigzag_vector.shape
        # OMP with dct zigzag vectors
        N = 64

        for M in self.M:
            pre_zigzag_vector = np.zeros(zigzag_shape)
            A = define_A(M,N)
            for i in range(zigzag_shape[0]):
                x = zigzag_vector[i].T
                y = A.dot(x)
                x_pre,Lambdas = OMP(A,y,N)
                pre_zigzag_vector[i] = x_pre.T


            pre_dct = self.zigzag2dct(pre_zigzag_vector,self.imsize)
            pre_img = self.idct8x8(pre_dct)
            norm_e =Normalized_Error(img,pre_img)
            self.norm_error.append(norm_e)
            psnr = PSNR(img,pre_img)
            self.psnrs.append(psnr)
            print("M={} Normalized Error={} PSNR={}".format(M,norm_e,psnr))
            plt.imsave(os.path.join(foldername,"preimageM={}.png".format(M)),pre_img,cmap="gray")

    def run_image_noise(self):
        foldername = "experiment/exp_{}/".format(self.expname)
        if os.path.exists(foldername)==False:
            os.mkdir(foldername)
        img = plt.imread('images/menglu.jpg')/255

        img = rgb2gray(img)
```

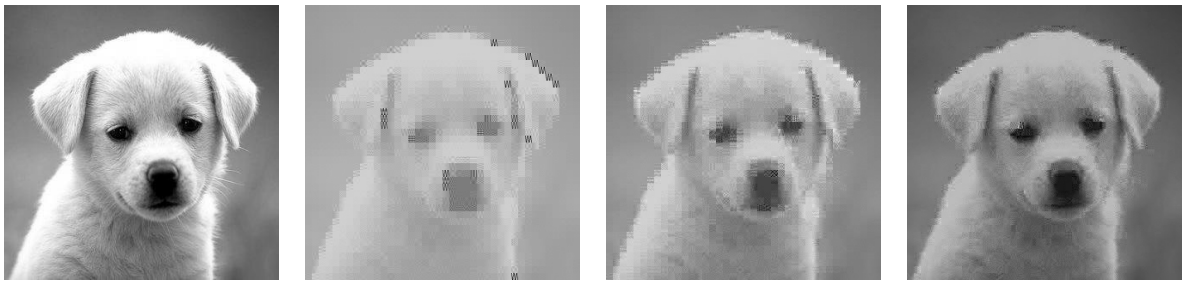
```
        img = resize(img, self.imsize)

        # Do 8x8 DCT on image and vectorized with zigzag pattern
        dct = self.dct8x8(img)
        zigzag_vector = self.dct2zigzag(dct)
        zigzag_shape = zigzag_vector.shape
        # OMP with dct zigzag vectors
        N = 64

        for M in self.M:
            pre_zigzag_vector = np.zeros(zigzag_shape)
            A = define_A(M,N)
            for i in range(zigzag_shape[0]):
                x = zigzag_vector[i].T
                n = np.random.normal(0,self.sigma,M).T
                y = A.dot(x) + n
                x_pre,Lambdas = OMP(A,y,N,r_thresh=norm2(n))
                pre_zigzag_vector[i] = x_pre.T


            pre_dct = self.zigzag2dct(pre_zigzag_vector,self.imsize)
            pre_img = self.idct8x8(pre_dct)
            norm_e =Normalized_Error(img,pre_img)
            self.norm_error.append(norm_e)
            psnr = PSNR(img,pre_img)
            self.psnrs.append(psnr)
            print("Noise M={} Normalized sigma={} Error={} PSNR={}".format(M,self.sigma,norm_e,psnr))

            plt.imsave(os.path.join(foldername,"preimageM={}.png".format(M)),pre_img,cmap="gray")

    def plot(self):
        plotpath = "plot/"
        plt.figure()
        plt.plot(self.M,self.psnrs)
        plt.xlabel("M")
        plt.ylabel("PSNR")
        plt.savefig(os.path.join(plotpath,"{}PSNR.png".format(self.expname)))

        plt.figure()
        plt.plot(self.M,self.norm_error)
        plt.xlabel("M")
        plt.ylabel("Normalized Error")
        plt.savefig(os.path.join(plotpath,"{}NormalizedError.png".format(self.expname)))




if __name__ == "__main__":
    exp = ImageExperiment()
    # exp.run_image()
    exp.run_image_noise()
    exp.plot()
```

Listing 4: Image Experiment

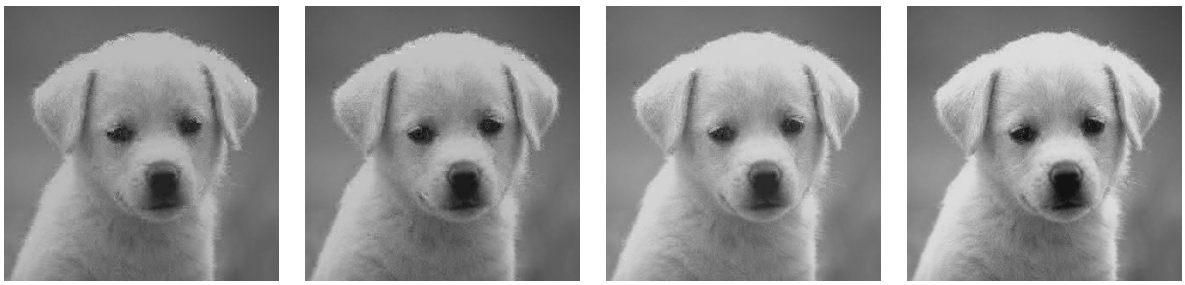
```
import matplotlib.pyplot as plt
import numpy as np
import os
plt.rcParams["font.family"] = "Times New Roman"
Ns=[20,50,100]
# os.mkdir('plot/Noise2/')
for N in Ns:
    # image = np.load('Noise2_N{}_sigma0.001.npy'.format(N))
    # name = 'N{}_ESR_rate'.format(N)
    name = 'Noise2_N{}_sigma0.001'.format(N)
    image = np.load('{}.npy'.format(name))

    plt.figure()
```

```
    print(image.shape)
    plt.imshow(image)
    ytick = range(N//2,N,3)
    xtick = range(10,100,5)
    plt.xticks(np.arange(len(xtick)),xtick)
    plt.yticks(np.arange(len(ytick)),ytick)
    plt.xlabel('M',fontsize=15)
    plt.ylabel('s max',fontsize=15)
    plt.colorbar()
    plt.savefig('plot/Noise2/{}.png'.format(name))
```

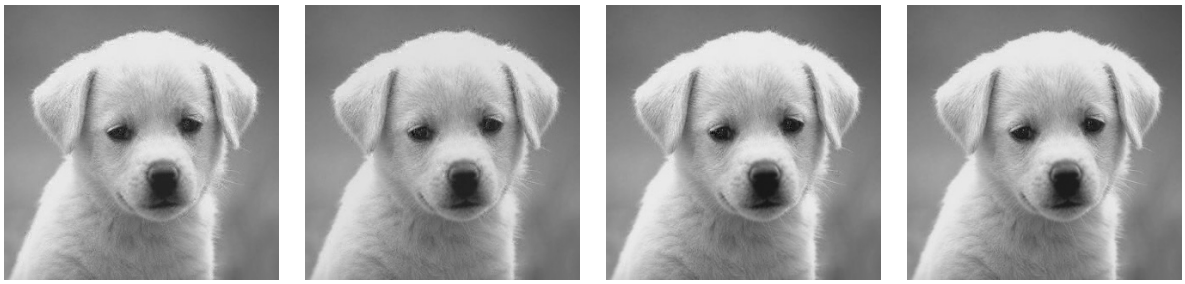
Listing 5: Image plot



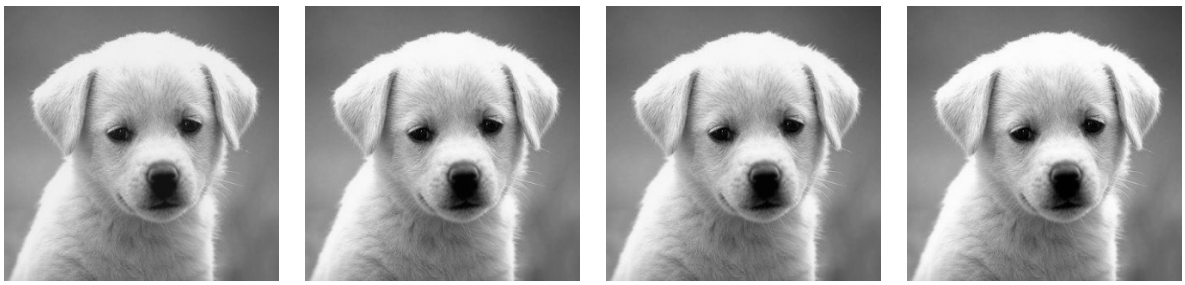
(a) Original (b) M=5 (c) M=10 (d) M=15



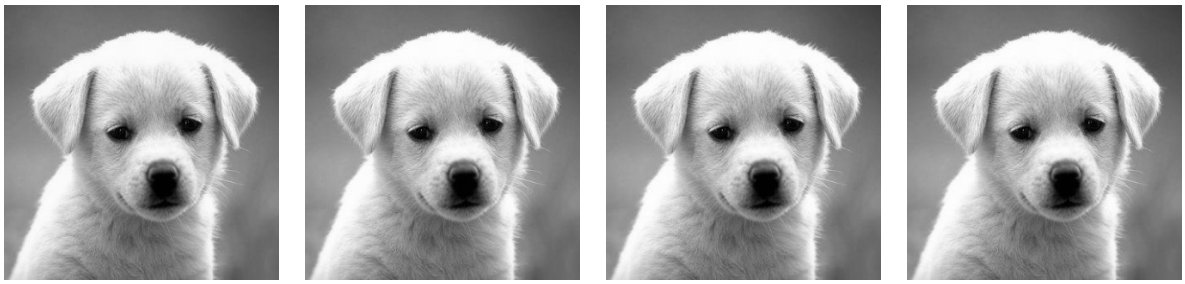
(e) M=20 (f) M=25 (g) M=30 (h) M=35



(i) M=40 (j) M=45 (k) M=50 (l) M=55



(m) M=60 (n) M=65 (o) M=70 (p) M=75



(q) M=80 (r) M=85 (s) M=90 (t) M=95

Figure 7: Dog: Noiseless reconstructed image with different M

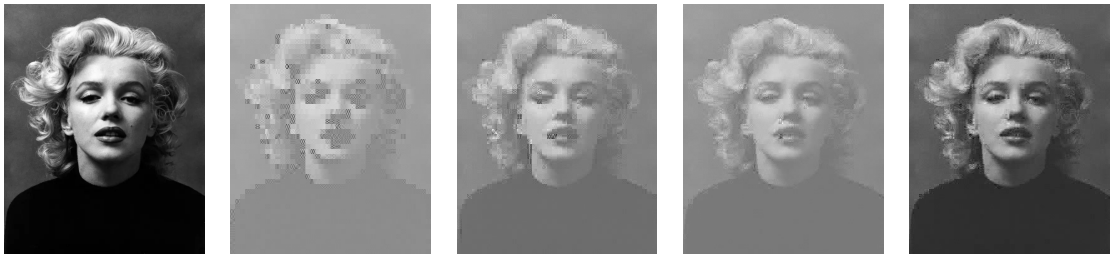
(a) Original (b) M=5 (c) M=10 (d) M=15 (e) M=20

(f) M=25 (g) M=30 (h) M=35 (i) M=40 (j) M=45

(k) M=50 (l) M=55 (m) M=60 (n) M=65 (o) M=70

(p) M=75 (q) M=80 (r) M=85 (s) M=90 (t) M=95

Figure 8: Marilyn Monroe: Noiseless reconstructed image with different M



(a) Original (b) M=5 (c) M=10 (d) M=15 (e) M=20

(f) M=25 (g) M=30 (h) M=35 (i) M=40 (j) M=45

(k) M=50 (l) M=55 (m) M=60 (n) M=65 (o) M=70

(p) M=75 (q) M=80 (r) M=85 (s) M=90 (t) M=95

Figure 9: Marilyn Monroe: Noisy reconstructed image, $\sigma = 0.01$

(a) Original (b) M=5 (c) M=10 (d) M=15 (e) M=20

(f) M=25 (g) M=30 (h) M=35 (i) M=40 (j) M=45

(k) M=50 (l) M=55 (m) M=60 (n) M=65 (o) M=70

(p) M=75 (q) M=80 (r) M=85 (s) M=90 (t) M=95

Figure 10: Marilyn Monroe: Noisy reconstructed image, $\sigma = 0.2$