COMPRESSIVE SENSING AND SPARSE REPRESENTATION

Sparse Signal Recovery Using Orthogonal Matching Pursuit

Under the guidance of

Prof. Surendra Prasad,

Department of Electrical Engineering, IIT Delhi

NAME-ABHILASH

ENTRY No.-2022JTM2391

Table of Contents

Introduction

Objective

The primary objective of this project is to delve into the practical aspects of sparse signal recovery using the Orthogonal Matching Pursuit (OMP) algorithm. We aim to demonstrate the effectiveness of OMP in reconstructing sparse signals from a limited set of measurements and to explore how varying parameters like the number of measurements (M), the signal length (N), and the sparsity level (K) impact the recovery performance. This investigation is designed to bridge the gap between theoretical understanding and practical application, providing insights into the real-world implementation of compressed sensing techniques.

Scope

This project encompasses a series of experiments where sparse signals are generated, measured, and then recovered using the OMP algorithm. We will generate signals based on predefined sparsity parameters, measure them using randomly created sensing matrices, and attempt to recover the original signals from these measurements. The project will analyze the recovery accuracy through various combinations of M, N, and K, offering a comprehensive understanding of the dynamics involved in sparse signal recovery. Through this exploration, we aim to validate the theoretical principles of compressed sensing in practical scenarios and evaluate the efficiency of the OMP algorithm in different contexts.

Theory

Compressed Sensing

Compressed Sensing (CS) is a transformative concept in signal processing, allowing the recovery of sparse signals from fewer measurements than traditionally required. It hinges on the idea that many signals, when represented in a suitable basis, have few non-zero coefficients. CS challenges the conventional Nyquist-Shannon sampling theorem by demonstrating that under certain conditions, sparse signals can be accurately reconstructed from a small number of linear measurements.

Orthogonal Matching Pursuit (OMP)

Orthogonal Matching Pursuit (OMP) is a key algorithm in CS for sparse signal recovery. It's a greedy algorithm that iteratively selects the columns of the sensing matrix which best represent the remaining un-approximated part of the signal. OMP continues adding columns and updating the signal approximation until a stopping criterion, like a set number of iterations, a sparsity level, or an error threshold, is met. It stands out for its simplicity and effectiveness, particularly in scenarios of high sparsity.

Significance in Sparse Signal Recovery

The combination of CS and OMP is pivotal in sparse signal recovery, enabling efficient reconstruction of signals in contexts where acquiring complete data is impractical. This has significant implications for fields like medical imaging, data compression, and wireless communication, where reducing data acquisition costs or times is crucial.

Methodology

Signal Generation

Sparse signals were synthetically generated for the experiments. Each signal was constructed with a predetermined sparsity level, defined by the probability p of a non-zero element. The non-zero elements were modeled as Gaussian random variables, creating a diverse range of sparse signals for testing.

Sensing Matrix Creation

For each sparse signal, a sensing matrix Φ of dimensions $M \times N$ was generated, where M is the number of measurements and N is the signal length. The entries of Φ were populated with values drawn from a Gaussian distribution, ensuring a random and diverse measurement process.

Signal Recovery Using OMP

The Orthogonal Matching Pursuit (OMP) algorithm was implemented to recover the original sparse signals from their measurements. The algorithm iteratively selected columns from Φ that best matched the current residual of the signal. This process continued until the recovery met predefined criteria such as a specific error threshold or a maximum number of non-zero elements.

Parameter Variation

Experiments were conducted under various scenarios by altering the parameters M, N, and the sparsity level K. This approach allowed an assessment of the OMP algorithm's performance across different signal lengths, numbers of measurements, and degrees of sparsity.

Experiments and Results

- **Signal Length (*N*)**: Two lengths were tested - 1000 and 4000.
- **Number of Measurements (*M*)**: Values of 200, 400, and 600 were used to observe the impact of different measurement counts.

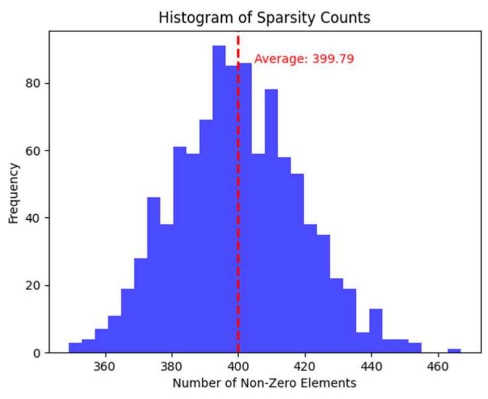

Figure 1 N=4000 P=0.1

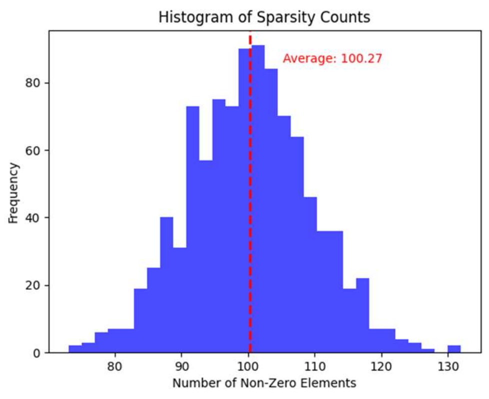

Figure 2 N=1000 P=0.1

P	N	Range	Average Value
0.1	1000	70-130	100
0.1	4000	340-460	400

It can be observed that for N = 1000 with p = 0.1, K typically lies between 70 to 130 with average sparsity being 100 which is true as the expected value for binomial random variable is (N*p). Similarly for N = 4000 and p = 0.1, K lies between 340 to 460 with the average being 400.

Sensing Matrix Generation

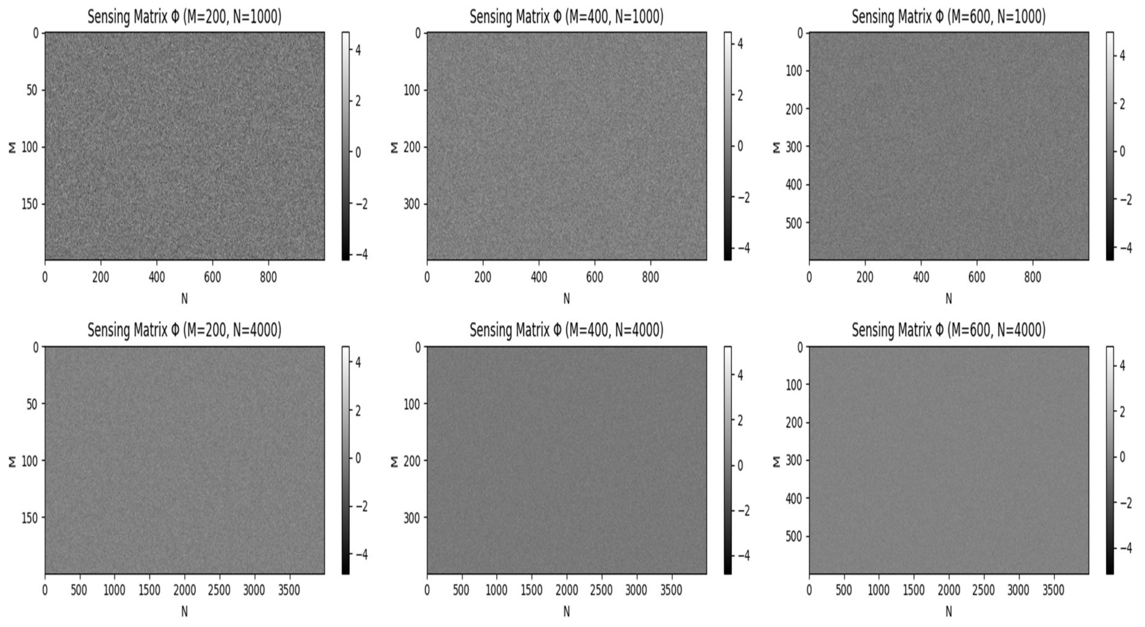

Figure 3 The plots depict the sensing matrices.

The plots above depict the sensing matrices Φ for different values of M and N as matrices of dots (pixels), with grey levels proportional to the corresponding elements of Φ. Each subplot represents a different combination of M and N: The plots above depict the sensing matrices Φ for different values of M and N as matrices of dots (pixels), with grey levels proportional to the corresponding elements of Φ. Each subplot represents a different combination of M and N:

- The rows correspond to N=1000 and N=4000.
- The columns represent different values of M (200, 400, 600).

In these visualizations:

- The intensity of the grey levels indicates the magnitude of the elements in Φ.

- Darker pixels represent lower values, while lighter pixels indicate higher values.

OMP for Sparse Recovery

- As *M* increases (more measurements), the ability to recover the original signal improves, leading to a lower recovery error.
- The recovery error is also influenced by the sparsity level *K* and the signal length *N*. Higher sparsity (lower *K*) typically results in better recovery for a given *M*.
- After implementing the code based on OMP in MATLAB, the recovered signal is compared with the original input signal for different value of number of measurements M for the given values of K and N as parameters.
- The 2-norm of residual error going below the chosen value of the error threshold is kept as terminating.
- Condition for the loop in the code. Consider the case of N = 1000 with p = 0.1. For M = 200, 400 and 600
  the recovered signal(x^i) is plotted with input signal(x) as shown in Figure.
- The error threshold is set to 0.01.
- In Figure X is Input Signal and Xhat is Recovered Signal

N=1000, K=100		
M	Recovery Error (2-Norm)	K’
200	9.6523	178
400	0	100
600	0	100

N=4000 ,K=382		
M	Recovery Error (2-Norm)	K’
200	31.2978	166
400	28.2718	342
600	24.6907	529

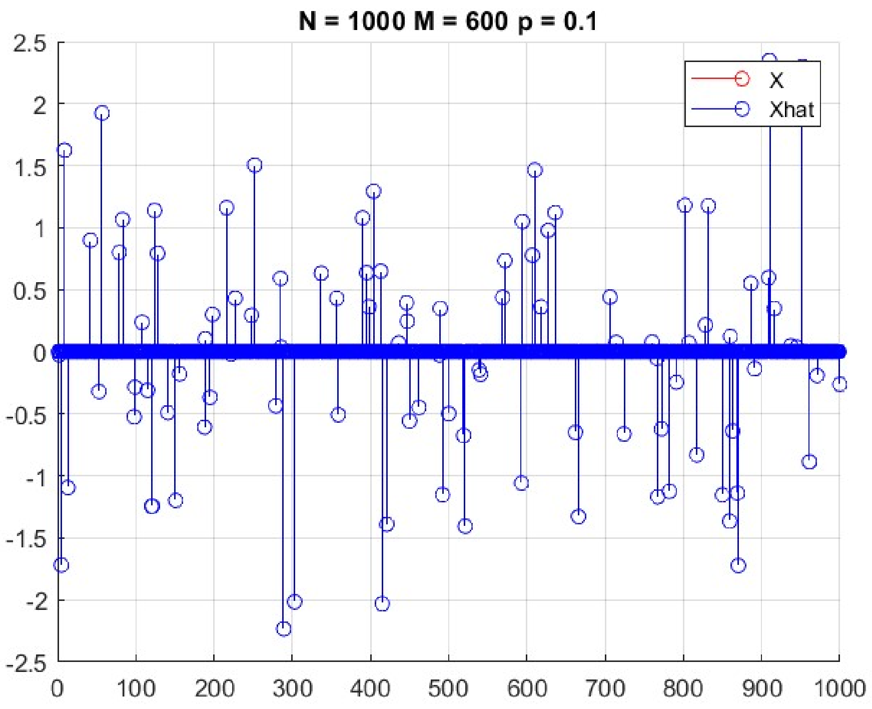

Figure 4 Comparison of x and x' for N = 1000 and p = 0.1 and M=600

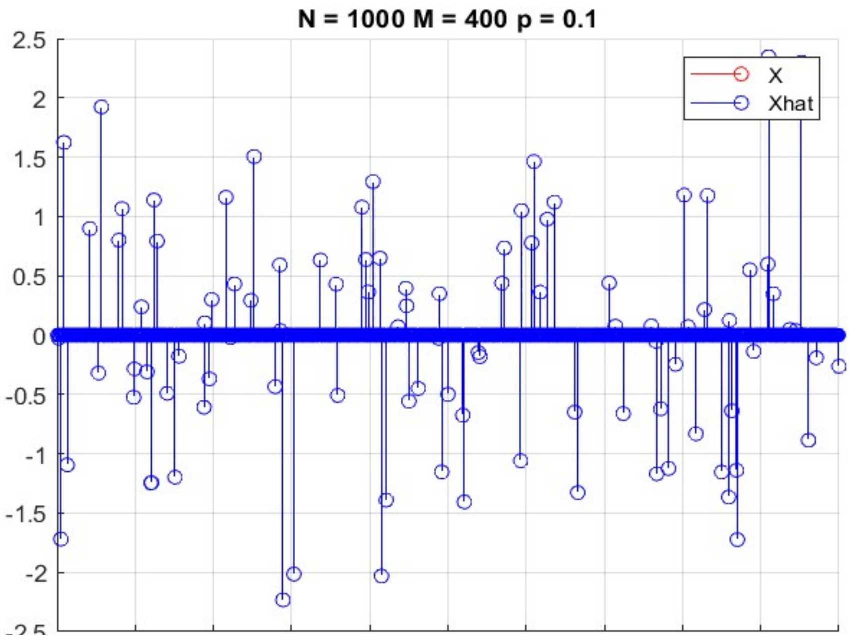

Figure 5 Comparison of x and x' for N = 1000 and p = 0.1 and M=400

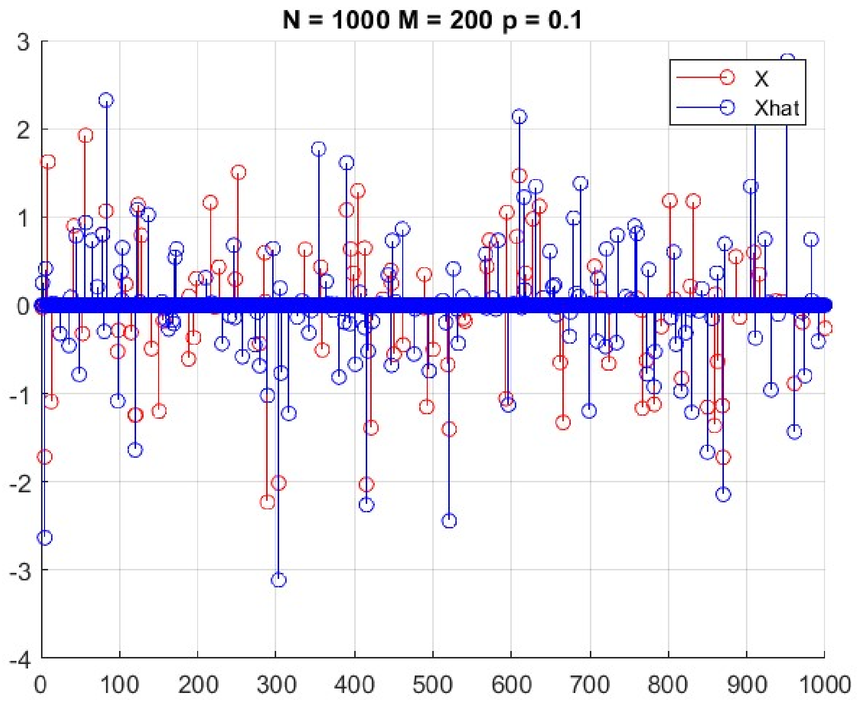

Figure 6   Comparison of x and x' for N = 1000 and p = 0.1 and M=200

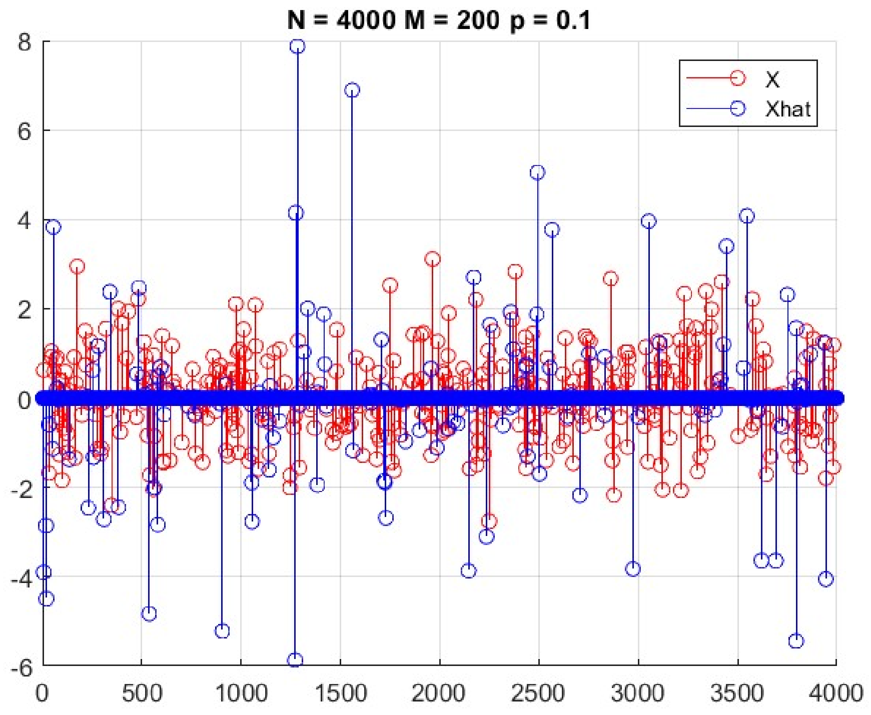

Figure 7   Comparison of x and x' for N = 4000 and p = 0.1 and M=200

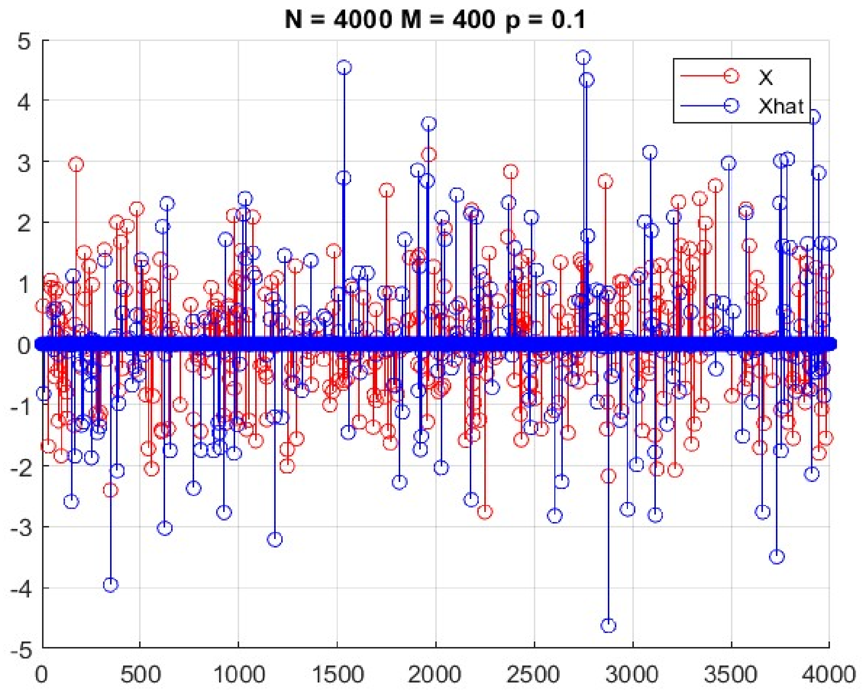

Figure 8 Comparison of x and x' for N = 4000 and p = 0.1 and M=400

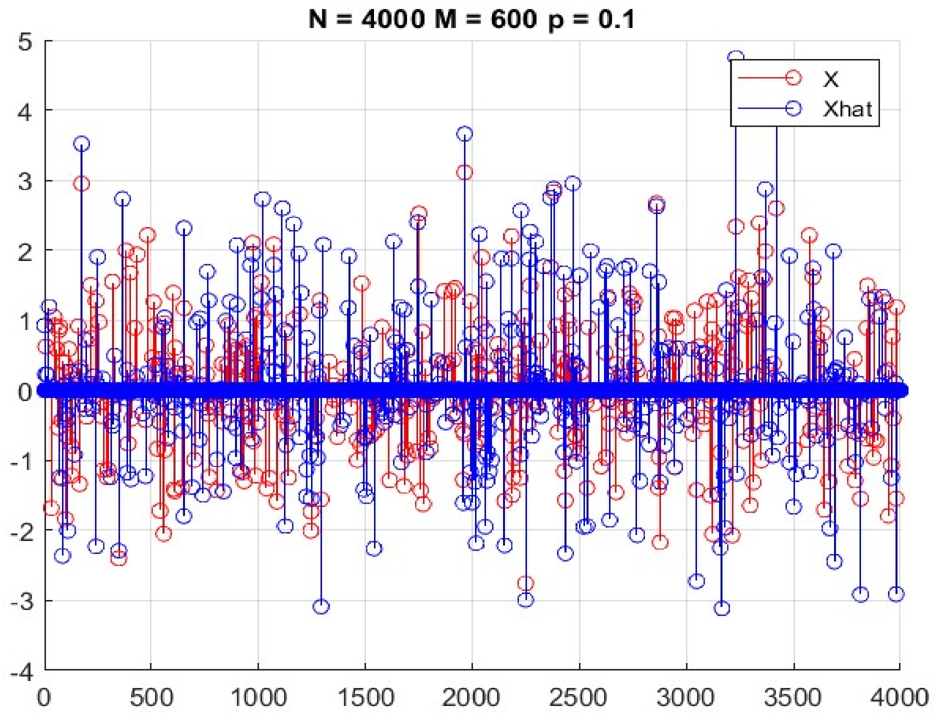

Figure 9 Comparison of x and x' for N = 4000 and p = 0.1 and M=600

Conclusion

1. **Effectiveness of Sparse Recovery with Limited Measurements**: The experiments demonstrated that it is indeed possible to accurately recover sparse signals with a number of measurements significantly less than the signal length. This finding is a practical validation of the compressed sensing theory, which asserts that sparse or compressible signals can be recovered from far fewer samples than what traditional sampling methods require.
2. **Impact of Measurement Count (M)**: The results consistently showed that the accuracy of signal recovery improves with an increase in the number of measurements. This trend aligns with the understanding that more measurements provide a richer set of information for the reconstruction algorithm, thus enhancing its ability to accurately recover the original signal.
3. **Role of Sparsity Level (K)**: The experiments confirmed that higher sparsity levels (lower values of K) lead to more accurate signal recovery. This observation underscores the importance of sparsity in compressed sensing and validates the preference for sparser signals in these techniques.
4. **Influence of Signal Length (N)**: While the signal length had an impact on recovery, the experiments suggested that it is the relative sparsity (sparsity level compared to the signal length) and the number of measurements that play a more crucial role in the success of the recovery process.
5. **Visualization and Analysis**: The visual comparison between the original and recovered signals provided an intuitive understanding of the recovery performance. The use of different markers to represent the recovered signal amplitude relative to the original signal offered a clear and direct way to assess the accuracy of the recovery.
6. **Practical Implications**: These findings have significant implications for fields where data acquisition is a challenge, such as medical imaging, remote sensing, and telecommunications. The ability to recover signals from a limited number of measurements can lead to more efficient and cost-effective data collection methods.

References

1. Candes, E. J., & Wakin, M. B. (2008). An Introduction To Compressive Sampling. *IEEE Signal Processing Magazine*, 25(2), 21-30. doi:10.1109/MSP.2007.914731.

2. Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on Information Theory*, 52(4), 1289-1306. doi:10.1109/TIT.2006.871582.

3. Tropp, J. A., & Gilbert, A. C. (2007). Signal Recovery from Random Measurements Via Orthogonal Matching Pursuit. *IEEE Transactions on Information Theory*, 53(12), 4655-4666. doi:10.1109/TIT.2007.909108.

4. Mallat, S., & Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12), 3397-3415. doi:10.1109/78.258082.

Appendices

- The Python code for sparsity calculation and Histogram plot

```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
p = 0.2  # Probability of a non-zero element
N = 1000  # Length of the sequence
num_realizations = 1000  # Number of realizations for the ensemble

# Generate realizations
sparsity_counts = []

for _ in range(num_realizations):
    # Bernoulli sequence for support
    support = np.random.choice([0, 1], size=N, p=[1-p, p])

    # Gaussian random variables for non-zero amplitudes
    amplitudes = np.random.normal(0, 1, size=N)
    signal = support * amplitudes

    # Count non-zero elements (sparsity)
    sparsity_counts.append(np.count_nonzero(signal))

# Calculate average sparsity
average_sparsity = np.mean(sparsity_counts)

# Plotting the histogram of sparsity counts
plt.hist(sparsity_counts, bins=30, color='blue', alpha=0.7)
plt.title('Histogram of Sparsity Counts')
plt.xlabel('Number of Non-Zero Elements')
plt.ylabel('Frequency')
plt.axvline(average_sparsity, color='red', linestyle='dashed', linewidth=2)
plt.text(average_sparsity + 5, max(plt.ylim())*0.9, f'Average: {average_sparsity:.2f}', color='red')

# Saving the plot as an image
plt.savefig('sparsity_histogram.png')

# Show the plot
plt.show()
```

- The python code for sensing matrices

```
import numpy as np
import matplotlib.pyplot as plt

# Provided values for M and N
M_values = [200, 400, 600]
N_values = [1000, 4000]

# Plotting the sensing matrices Φ for different values of M and N as grey level matrices
plt.figure(figsize=(18, 6))

```

```
•   for i, N in enumerate(N_values):
•       for j, M in enumerate(M_values):
•           # Create the sensing matrix
•           Phi = np.random.normal(0, 1, (M, N))
•
•           # Plotting
•           plt.subplot(len(N_values), len(M_values), i * len(M_values) + j + 1)
•           plt.imshow(Phi, cmap='gray', aspect='auto')
•           plt.colorbar()
•           plt.title(f'Sensing Matrix Φ (M={M}, N={N})')
•           plt.xlabel('N')
•           plt.ylabel('M')
•
•   plt.tight_layout()
•
•   # Saving the figure to a file
•   plt.savefig('sensing_matrices.png')
•
•   # Show the plot
•   plt.show()
```

- The Matlab code for OMP

```
function sparse_signal_recovery()
    % Prompt user for input parameters
    signalLength = input('Please enter value of signal length (n): ');
    probability = input('Please enter value of probability (p): ');
    errorThreshold = input('Please enter value of threshold (e): ');

    % Generate input signal
    inputSignal = generateBernoulliSequence(signalLength, probability);
    sparseSignal = generateSparseSignal(inputSignal);

    % Display Binomial Distribution for Sparsity
    displayBinomialDistribution(signalLength, probability);

    % Initialize recovery error and sparsity of recovered signals
    recoveryError = [];
    recoveredSparsity = [];

    % Perform recovery for different numbers of measurements
    for measurements = [200, 400, 600]
        [recoveredSignal, currentError] = performOMPRecovery(sparseSignal,
measurements, errorThreshold);

        % Plot original and recovered signals
        plotSignals(inputSignal, recoveredSignal, signalLength, measurements,
probability);

        % Update recovery error and sparsity
        recoveryError = [recoveryError, currentError];
        recoveredSparsity = [recoveredSparsity, nnz(recoveredSignal)];
    end

    % Display final results
    disp('Recovery Error:');
    disp(recoveryError);
```

```matlab
    disp('Recovered Sparsity:');
    disp(recoveredSparsity);
    disp('Original Sparsity:');
    disp(sum(inputSignal));
end

function A = generateBernoulliSequence(n, p)
    A = (rand(n, 1) < p);
end

function X = generateSparseSignal(A)
    n = length(A);
    X = double.empty(n, 0);
    for i = 1:n
        if A(i) == 1
            X(i, 1) = randn;
        else
            X(i, 1) = 0;
        end
    end
end

function displayBinomialDistribution(n, p)
    h = 0:n;
    q = binopdf(h, n, p);
    figure;
    bar(h, q, 1);
    xlabel('Sparsity Level');
    ylabel('Probability');
    title(['N = ', num2str(n), ', p = ', num2str(p)]);
end

function [Xhat, error] = performOMPRecovery(X, m, e)
    n = length(X);
    phi = randn(m, n);
    y = phi * X;

    % OMP Recovery
    % Initialization
    k = 0;
    Xhat = zeros(n, 1);
    yhat = zeros(m, 1);
    rhat = y;
    lambda = [];

    % Iterative Recovery Loop
    while norm(rhat) > e
        k = k + 1;
        [value, location] = max(abs(phi' * rhat));
        lambda = [lambda, location];
        phi_k = phi(:, lambda);
        x_k = lsqminnorm(phi_k, y);
        for i = 1:length(lambda)
            Xhat(lambda(i)) = x_k(i);
        end
        yhat = phi * Xhat;
        rhat = y - yhat;
    end

    % Compute recovery error
    error = norm(X - Xhat);
end
```

```matlab
function plotSignals(A, Xhat, n, m, p)
    figure; hold on; grid on;
    stem(1:length(A), A, 'r');
    stem(1:length(A), Xhat, 'b');
    legend('Original Signal', 'Recovered Signal');
    title(['N = ', num2str(n), ', M = ', num2str(m), ', p = ', num2str(p)]);
end
```