COMPRESSIVE SENSING: APPLICATIONS FROM 1-D TO N-D

Kenneth V. Domingo

kdomingo@nip.upd.edu.ph

An Undergraduate Thesis Submitted to

National Institute of Physics

College of Science

University of the Philippines

Diliman, Quezon City

In Partial Fulfillment of the Requirements

for the Degree of

BACHELOR OF SCIENCE IN APPLIED PHYSICS

June 2020

Acknowledgments

Abstract

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This study explores the use of compressive sensing (CS), an emergent sampling theorem that allows reconstruction of signals from much fewer samples than required by the Nyquist-Shannon sampling theorem (NST) . In this framework, the computational burden of encoding/decoding a signal is shifted from the sampling device to the device performing reconstruction, decompression, or other modes of post-processing. As such, there exist many ways to reconstruct a signal from compressive measurements.

CS has found its applications in simple audio signals containing stable frequencies (such as pure tones [1, 2]) and dynamic frequencies (such as speech [3–5]), images [6–8], and grayscale videos [9, 10]. The formulation of a sensing matrix in CS requires a basis conforming to some uniform uncertainty principle, and most common starting points would be partial discrete cosine transform (DCT) matrices or partial discrete wavelet transform (DWT) matrices.

1.1 Related literature

In 2004, Candès, Romberg, Tao [11], and Donoho [12] both asked and answered the questions that birthed the field which we now know as compressive sensing. The methods in CS apply concepts from time-frequency uncertainty principles [13] and sparse representations, which were studied rigorously by Donoho and Elad [14].

Linh-Trung et al. [15] demonstrated the use of deterministic chaos filters to acquire samples instead of random distributions. Normally, a deterministic chaotic function needs one or more initial value parameters, and the sequence produced by different combinations of initial values rapidly diverge from each other. This phenomenon led to investigation of the use of CS as an encryption algorithm. Simultaneous compression and encryption was achieved by [7], and it was found that the produced sequences were sensitive to initial value perturbations on the order of 10^{-15}. Their image compression-encryption model via CS was shown to have a key space on the order of 10^{83}, making it extremely resistant to brute force and other types of attacks. In this study, a logistic map was used to encode and construct the sensing matrix. The encryption system exhibited similar key sensitivity and robustness characteristics mentioned in the former. In the methods above, including those in this study, sampling was performed in the signal domain (i.e., temporal domain for audio, spatial domain for images), and the reconstruction was performed in the frequency domain.

Whereas images are not naturally bandlimited and rather, are dependent on the spatial resolution and bit depth of the imaging device, audio size scales proportionally with time and takes on a wider range of values. The accepted frequency range of human hearing is from 20 Hz to 20 kHz, so by the NST, a

sampling frequency greater than 40 kHz is needed to ensure that an audio sample is recorded completely. Any meaningful audio recording, especially those containing speech, will certainly have a significant duration, so one cannot straightforwardly apply methodologies used for images . The first challenge this would pose for electronic systems is insufficient memory to process the entire signal all at once. Low circumvented this problem [3, 4] by transforming the signal to the modulation domain, essentially raising a one-dimensional signal to N-dimensions, where N is dependent on the desired spectrogram resolution and number of subbands. This method was adopted in this study, and additionally, each subband was multiplied with a window function to suppress potential boundary artifacts when reconstructing. In earlier experiments, reconstruction would often completely fail when windowing was not used. In the cases, however, that were successful, the reconstruction exhibited severe boundary artifacts in the form of distortion, aliasing, or noise.

1.2 Novelty

This study aims to provide a generalization for applying CS techniques to signals of arbitrary dimensions, for applications such as compression, encryption, and enhancement. Contemporary CS research work exclusively on either audio or image signals, and—due to the computational demands—and focus on constructing effective sensing matrices, optimizing the computational complexity for real-time applications, and improving reconstruction quality. In the establishment of CS methods, two different general frameworks arise for image and audio signals.

Furthermore, current research tend to evaluate signal reconstruction quality using statistical metrics, such as mean-squared error (MSE) and its variants. Arguably, the final interpreter of all signals are humans, and it is important to be able to tell how well any compressive/reconstructive algorithm performs just by looking at the metrics without directly observing the signal contents. In light of this, the study also aims to evaluate the reconstruction quality of CS algorithms using perceptually accurate metrics. This class of objective metrics are usually built upon now-obsolete subjective scoring systems, and allows human observers to make an informed estimate of the signal quality without directly accessing the signal itself.

Finally, this study is conducted in order to lay out a unified, standardized workflow for similar applications of CS on signals with arbitrary content. This includes signals containing a combination of audio and images, such as color videos and hyperspectral images.

1.3 Thesis overview

The next chapter establishes the relevant mathematical concepts and notation to be used throughout this study, algorithms used in signal reconstruction, and appropriate metrics per type of signal. Chapter 3 establishes basic workflows and studies the effect of random sampling on CS reconstruction. Chapters 4 & 5 respectively focus on image-based CS and audio-based CS. Each of these chapters are self-contained methodologies, results, and discussions to emphasize that the methods can work independently of each other. Conclusions of the study and recommendations for future studies are presented in Chapter 6.

Chapter 2

Preliminaries

The trend of both curiosity and profit-driven human development has caused a surge in the amount of openly accessible raw data. More often than not, the data is generated much faster than it can be processed into something interpretable or useful. In the endeavor of keeping up with the inflow of information, there are two major factors that significantly hinder our progress. First, Moore's law implicitly sets a physical limit to the number of transistors that can be placed on a chip, consequently limiting how powerful and how fast electronic systems can become (barring a paradigm shift in the fundamental design of semiconductors). The second is the Nyquist-Shannon sampling theorem (NST), which limits the range of frequencies a recording device can successfully capture. This states that given that you know a signal's highest frequency component f_B, sampling it at a rate f_S that is greater than twice this frequency is sufficient to capture all of the pertinent information regarding that signal: that is $f_S > 2f_B$, where f_B is known as the Nyquist frequency or the bandwidth; twice this value is the Nyquist rate [16]. For signals that are not naturally bandlimited, such as images, the ability

reproduce a signal is dependent on the device's resolution and still follows the same principle: there should be at more than twice the number of pixels codimensional with the image's highest spatial frequency. For practical day-to-day use, the NST will suffice. However, issues arise when bandwidth and storage are at a premium. Typically, after sensing a signal, not all of the raw data is stored. Rather, this data is converted to a compressed format by systematically discarding values such that the loss of information is virtually imperceptible. Thus, the process of acquiring massive amounts of data followed by compression is extremely wasteful. CS aims to directly acquire the parts of the signal that would otherwise survive this compression stage in the classical sampling scheme.

2.1 Sampling paradigms

Consider a signal $\mathbf{x} \in \mathbb{R}^n$; this notation indicates that $\mathbf{x}$ is a vector of cardinality n, containing elements over the field of real numbers ($\mathbf{x}$ can also easily be a complex vector, but for the purposes of this chapter, it is sufficient to emphasize that we are working with real-valued signals). The process of acquisition or sensing this signal can be modeled as a linear system, where the physical signal properties we wish to capture are transformed into digital values by applying a linear transformation

$$\mathbf{y} = \mathbf{A}\mathbf{x} \tag{2.1}$$

or in the literature of signal processing [17], by correlating them with a waveform basis

$$y_k = \langle \mathbf{x} | \mathbf{a}_k \rangle , \quad k \in \mathbb{N} \leq n \tag{2.2}$$

In conventional sampling, $\mathbf{a}_k$ are Dirac basis vectors which turn $\mathbf{y}$ into a vector containing samples of $\mathbf{x}$ in the temporal or spatial domain; if $\mathbf{a}_k$ are Fourier basis vectors (i.e., sinusoids), then $\mathbf{y}$ is a vector of Fourier coefficients. If the signal has been sampled sufficiently in the sense that the number of measurements m is equal to the dimension n of the signal, then $\mathbf{A}$ is a square matrix, and the original signal $\mathbf{x}$ can be reconstructed from the information vector $\mathbf{y}$ by inversion of (2.1). However, the process of recovering $\mathbf{x} \in \mathbb{R}^n$ from $\mathbf{y} \in \mathbb{R}^m$ becomes ill-posed when we consider the undersampled case ($m \ll n$), as the sensing matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$—whose row vectors are denoted as $\mathbf{a}_m$—causes the system to become underdetermined: there exist infinitely many candidate solutions $\hat{\mathbf{x}}$ which satisfy (2.1). To add to this, we also consider the possibility that the measurements are not perfect, and are contaminated with noise. How then do we recover a signal from measurements which are incomplete and most likely inaccurate? The answer lies in enforcing constraints based on models of natural signals, as well as constraints based on optimization techniques. CS can be viewed as a strategic undersampling method: the signal is sampled at random points, and the ratio of the number of indices where it is sampled to the size of the signal can be associated with some quasi-frequency which may be lower than the Nyquist rate.

2.2 Sparsity

Most natural signals, especially those with some underlying periodicity, can be represented sparsely when expressed in the appropriate basis. This process of "sparsifying" can be expressed as

$$f = \langle \mathbf{x} | \boldsymbol{\psi}(k) \rangle \tag{2.3}$$

Similar to (2.2), this involves correlating the signal with the appropriate basis function to yield a representation in the sparse domain. Image information, for example, are commonly expressed in the DCT domain by

$$f_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right], \quad 0 \leq k < N \tag{2.4}$$

and its corresponding inverse is

$$x_k = \frac{1}{2} f_0 + \sum_{n=1}^{N-1} f_n \cos\left[\frac{\pi}{N}\left(k + \frac{1}{2}\right)n\right], \quad 0 \leq k < N \tag{2.5}$$

where the cosine term corresponds to $\boldsymbol{\psi}(k)$ in (2.3). We can express (2.4) more conveniently as $\mathbf{f} = \boldsymbol{\Psi}\mathbf{x}$, where $\boldsymbol{\Psi} \in \mathbb{R}^{n \times n}$ is the sparsifying matrix. Figure 2.1 shows this sparsifying process in action: given a test image, taking its Daubechies-8 discrete wavelet transform (D8 DWT) and zooming into a random subset shows that most of the signal energy is concentrated in just a few of the coefficients. All the other coefficients, when compared to the k highest coefficients, are practically zero; such a signal is referred to as k-sparse. The compressed image resulting from discarding all but the 25,000 highest coefficients and performing the inverse transform shows that any difference from the original image is virtually imperceptible. A similar concept is used in JPEG compression, wherein an image is divided into 8×8 blocks, and in each block, a certain number of DCT coefficients are discarded depending on the desired quality factor Q [18].

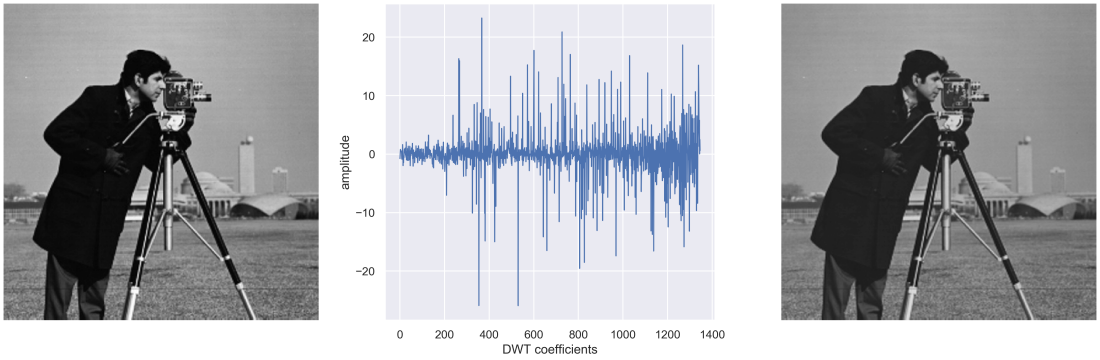


Figure 2.1: Original 512×512, 8-bit image (left), and a random subset (for better visibility) of its D8 DWT coefficients (middle). Most of the signal energy is concentrated in just a few terms. By discarding all but the 25,000 highest coefficients and performing the inverse transform, the resulting image (right) is perceptually no different from the original.

2.3 Incoherence

Suppose we have two matrices $\mathbf{\Phi}$ and $\mathbf{\Psi}$ which are involved in the sensing of a signal. As before, $\mathbf{\Psi}$ is the sparsifying matrix which converts the signal into a sparse representation, and $\mathbf{\Phi}$ is the actual sensing matrix. The coherence between these two bases is expressed as

$$\mu(\mathbf{\Phi}, \mathbf{\Psi}) = \sqrt{n} \max_{0 \leq i,j < n} |\langle \boldsymbol{\varphi}_i | \boldsymbol{\psi}_j \rangle| \tag{2.6}$$

In other words, the coherence is the measure of the largest correlation between the column vectors of $\mathbf{\Phi}$ and $\mathbf{\Psi}$. In compressive sensing, we are interested in low-coherence basis pairs (i.e., basis pairs for which $\mu \to 1$). For example, in the classical sampling scheme, $\mathbf{\Phi}$ is the Dirac basis $\boldsymbol{\varphi}_k(t) = \delta(t - k)$, and $\mathbf{\Psi}$ is the DCT basis (2.4). This basis pair in particular is also called a time-frequency pair and achieves maximum incoherence ($\mu = 1$) regardless of the number of dimensions [13]. Additionally, any orthonormal basis $\mathbf{\Phi}$ containing independent identically

distributed (i.i.d.) entries are also largely incoherent with a fixed basis $\mathbf{\Psi}$ [17]. The consequence of this is that CS performs most efficiently when sensing with incoherent and random systems.

2.4 Reconstruction strategies

Another measure of the sparsity of a signal is its ℓ_0 norm, denoted $\|\mathbf{x}\|_0$, which simply counts the number of non-zero coefficients of $\mathbf{x}$. As such, the goal of the reconstruction stage in CS is to find the sparsest representation of the vector $\mathbf{x}$ in terms of the sensing matrix $\mathbf{\Phi}$ by solving the combinatorial optimization problem

$$\min_{\mathbf{x}} \|\mathbf{x}\|_0 \quad \text{subject to} \quad \mathbf{y} = \mathbf{\Phi x} \tag{2.7}$$

which, as the name implies, requires one to enumerate all possible k-element combinations of the columns of $\mathbf{\Phi}$, and determining the smallest combination which approximates the signal the closest. However, this process quickly becomes intractable even for a modestly-sized signal. This requirement is therefore relaxed by instead minimizing the ℓ_1 norm

$$\min_{\mathbf{x}} \|\mathbf{x}\|_1 \quad \text{subject to} \quad \mathbf{y} = \mathbf{\Phi x} \tag{2.8}$$

where the ℓ_1 norm is defined as

$$\|\mathbf{x}\|_1 = \sum_{i=0}^{N-1} |x_i| \tag{2.9}$$

and is commonly called the taxicab or Manhattan distance. Most signals encountered in practical situations, however, are not sparse but rather,

approximately sparse. As mentioned earlier, any signal measurement will inevitably include some form of noise. Though ℓ_1 minimization can definitely still be used (by casting it as a convex problem, as in the case of [19, 20]), other algorithms opt for an ℓ_1-regularized least squares approach as in the case of LASSO [21], whose objective is

$$\min_{\mathbf{x}} \frac{1}{2m} \|\mathbf{y} - \mathbf{\Phi}\mathbf{x}\|_2^2 + \alpha \|\mathbf{x}\|_1 \tag{2.10}$$

where $0 \leq \alpha \leq 1$ is the ℓ_1 regularization parameter. Greedy algorithms are also a popular approach in this problem, the most common being the sparsity-constrained orthogonal matching pursuit (OMP) [22], which has the objective

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{\Phi}\mathbf{x}\|_2^2 \quad \text{subject to} \quad \|\mathbf{x}\|_0 \leq k \tag{2.11}$$

This method enforces the constraint that the reconstructed signal should be, at most, k-sparse in the selected coding dictionary $\mathbf{\Phi}$. There exist a plethora of algorithms dedicated to the decoding phase of CS. The ones mentioned above are primarily used in this study.

Chapter 3

Random sampling-based compressive sensing

In this chapter, I lay out the groundwork for performing basic compressive sensing techniques which will be repeatedly used and built upon in the following chapters. I also investigate various random properties that take place in the construction of sensing matrices and their potential effect on the reconstruction quality. In order to quantify these properties, I focus primarily on one-dimensional sinusoids, better visualized as audio. In particular, these are signals containing a few known frequency components that do not vary appreciably, if at all, through time.

3.1 Test case: Sinusoid

For the signals of interest, I use the Fourier domain as the sparse representation. I synthesized a C_5 piano note (523 Hz)—corresponding to a Nyquist rate of 1046 Hz—using Guitar Pro, with the standard sampling rate of 44.1 kHz and a duration of 1 second. Due to the number of samples, I only worked with the first 1/8th

second, corresponding to 5512 samples. This will be the original signal; let's call this signal $\mathbf{x}$. I then compressively sampled this portion by taking 300 uniformly distributed random measurements, equivalent to a 5% compression ratio; this will be our compressed vector $\mathbf{y}$. Figure 3.1 visualizes how these measurements are distributed in time.

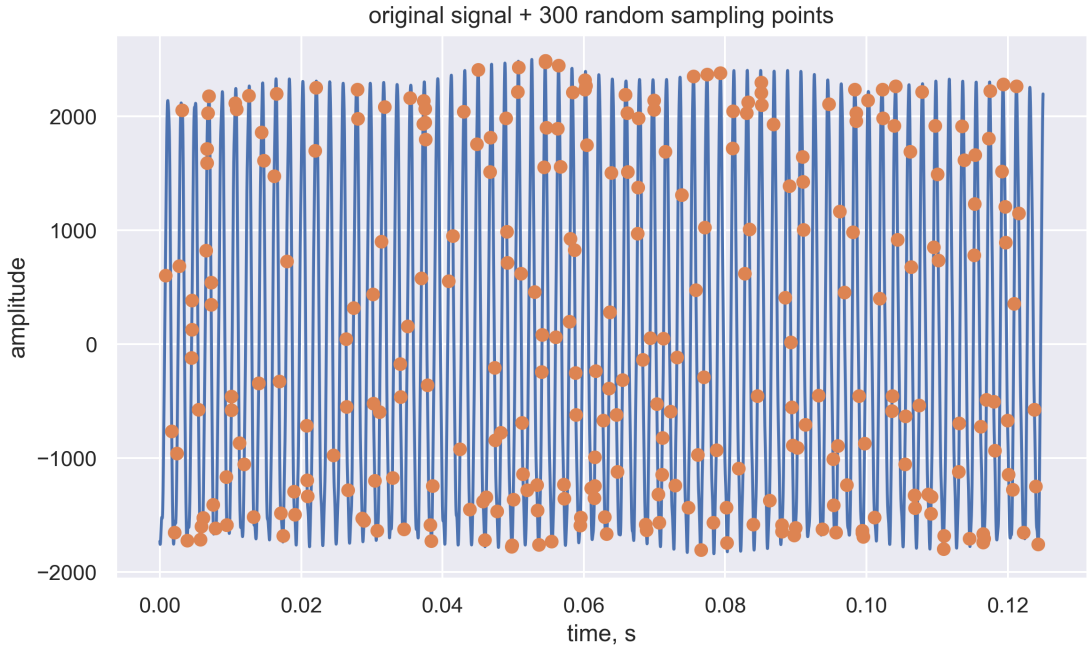


Figure 3.1: Original test signal (blue) and random measurements (orange).

In this model, sampling points consist of a discrete set of indices containing the signal amplitude corresponding to an instantaneous point in time. The random measurements actually point to these indices, which subset the chosen sparsifying basis: in this case, the DCT domain, for simplicity. The same random indices are used to select the rows of the DCT matrix of shape $n \times n$, where n is the signal dimension. I then stack these rows to form the sensing matrix $\mathbf{A}$; essentially a partial DCT matrix of shape $m \times n$, where m is the number of random measurements.

Essentially, I am simulating a sensing device that not only purposely undersamples, but also samples at random, intermittent points in time.

In Chapter 2, I discussed an overview of popular algorithms used in CS. For the purposes of comparison in this chapter, I will be focusing on a gradient-based method (LASSO), and a convex optimization-based method (CVXPY). The optimization objectives for these two become

$$\text{LASSO}: \quad \min_{\hat{\mathbf{x}}} \frac{1}{2m}\|\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}\|_2^2 + \alpha\|\hat{\mathbf{x}}\|_1 \tag{3.1}$$

$$\text{CVXPY}: \quad \min_{\hat{\mathbf{x}}} \|\hat{\mathbf{x}}\|_1 \quad \text{subject to} \quad \mathbf{A}\hat{\mathbf{x}} = \mathbf{y} \tag{3.2}$$

where $\hat{\mathbf{x}}$ is the candidate solution, and the optimum value of α was automatically determined via 10-fold cross validation. A detailed implementation is shown in Appendix A. Figure 3.2 shows a comparison of the original signal with the reconstructions from the two algorithms in the time and frequency domains. In the time domain, both algorithms appear to have been able to successfully reconstruct the signal, though the CVXPY recovery shows many artifacts. The LASSO algorithm yields a mean-squared error (MSE) of 0.002, while CVXPY yields an MSE of 0.074. In the frequency domain, however, many frequencies are erroneously being recovered by the LASSO method, and more so with the CVXPY method. Because LASSO's α is a hyperparameter, it will need additional tuning to yield a more optimal value.

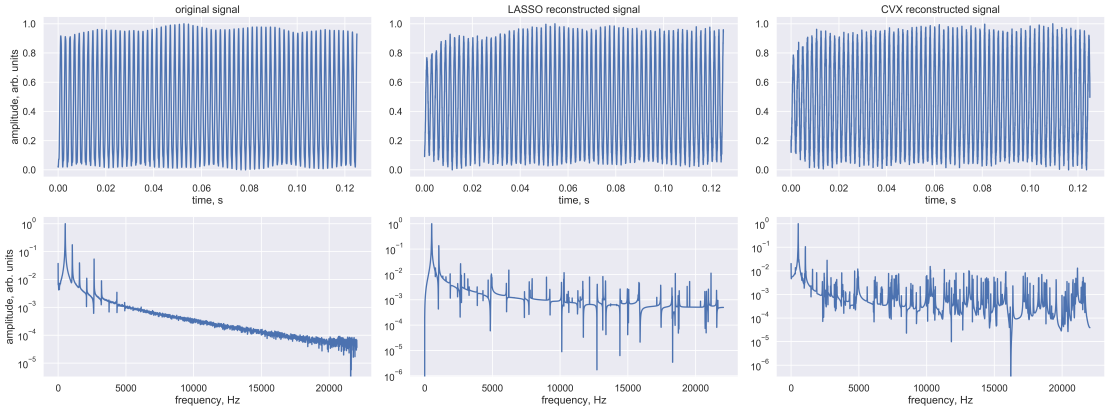


Figure 3.2: Original signal (leftmost), LASSO reconstruction (middle), and CVXPY reconstruction (rightmost). The top row shows the time domain representation, while the bottom row shows the frequency domain representation.

3.2 Effect of random distribution on reconstruction error

So far, I have worked solely with uniformly-distributed random sampling. Here, I investigate and compare the quality of reconstruction (in MSE) in terms of the random distribution. I will be working with two common distributions: the Gaussian and Poisson distributions, as well as the triangular distribution, which is commonly used in audio and image dithering. For each distribution, I generate i.i.d. random variables and use these to compressively sample the signal. I then evaluate the reconstruction MSE and take the average over 10 iterations to obtain error bars.

3.2.1 Uniform

The uniform distribution is given by

$$U(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b, \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where a and b are the lower and upper bounds, respectively.

3.2.2 Gaussian

The Gaussian/normal distribution can be generated by

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \tag{3.4}$$

where the parameters $\mu : \mu \in \mathbb{R}$ & $\sigma^2 : \sigma > 0$ are the distribution's mean and variance, respectively.

3.2.3 Poisson

The Poisson distribution can be generated by

$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!} \tag{3.5}$$

where the parameter $\lambda : \lambda > 0$ is the distribution's mean and variance.

3.2.4 Triangular

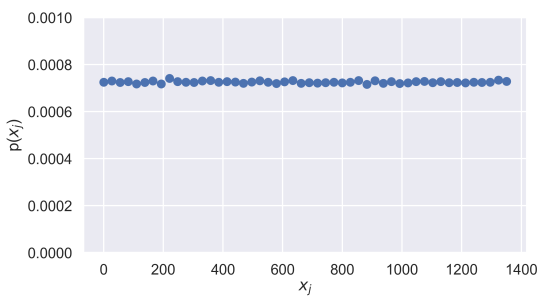
The triangular distribution is generated by

$$T(x) = \begin{cases} 0 & x < a, \\ \frac{2(x-a)}{(b-a)(c-a)} & a \leq x < c, \\ \frac{2}{b-a} & x = c, \\ \frac{2(b-x)}{(b-a)(b-c)} & c < x \leq b, \\ 0 & x > b \end{cases} \tag{3.6}$$

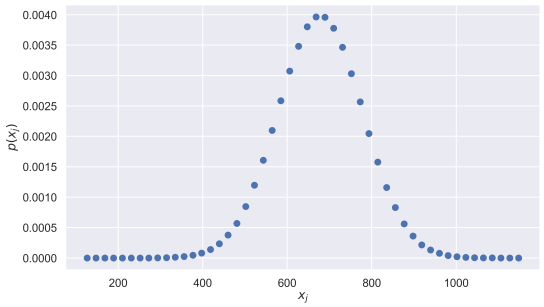
where $a : a \in (-\infty, +\infty)$ is the lower bound, $b : b > a$ is the upper bound, and $c : a \leq c \leq b$ is the mode.

3.2.5 Results & discussion

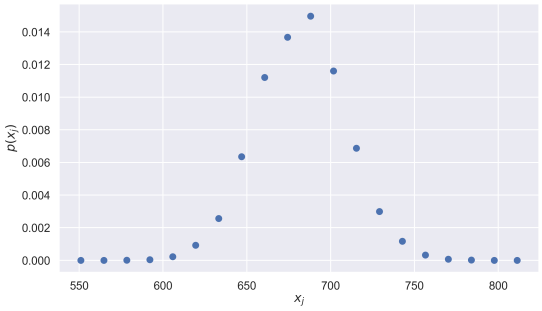
Due to the computational requirements, I will only work with the first 1/32 seconds of the signal, corresponding to 1378 samples. Figure 3.3 shows the probability density for each distribution.



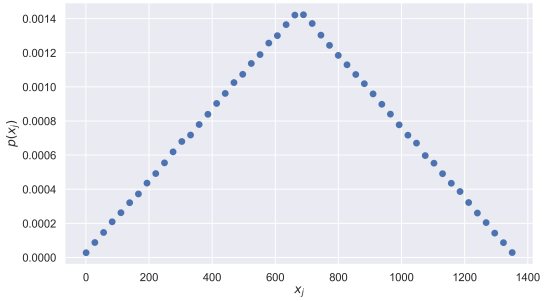
(a) Uniform



(b) Gaussian ($\mu = n/2$, $\sigma = 100$)



(c) Poisson ($\lambda = n/2$)



(d) Triangular ($a = 0$, $b = n/2$, $c = n$)

Figure 3.3: Probability densities of the different random distributions used in this section, corresponding to the signal indices.

Figure 3.4 shows the MSE evaluated for each random distribution as a function of the fraction of total samples, more aptly referred to as the compression ratio. From this, we can observe that the uniform and triangular distributions give the lowest reconstruction error, but the latter has a more consistent performance

across a wide range of compression ratios. They are followed, in order, by the Poisson and Gaussian distributions. One reason for the former's performance is that they are able to completely span the signal with appreciable probability near the bounds, while the latter's probability near the bounds are quickly approaching zero.

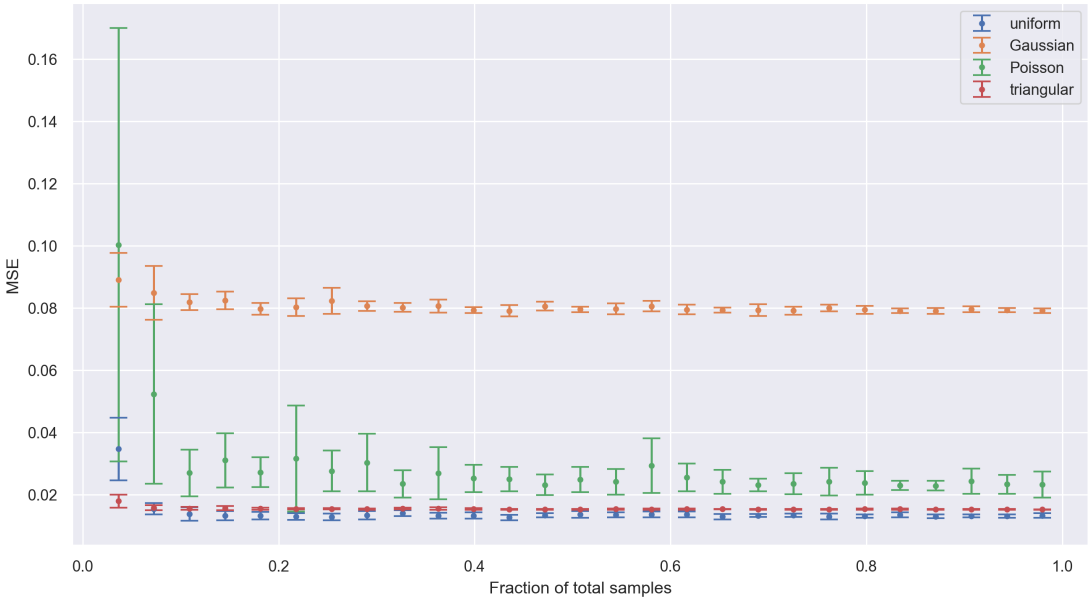


Figure 3.4: Evaluated MSE for each random distribution as a function of compression ratio, average over 10 iterations.

In line with these findings, I will be using uniformly-distributed random variables throughout this study unless otherwise stated. In a later chapters, I will be exploring more on recovering exact frequencies and their harmonics beyond the Nyquist rate in real signals, as well as signals with multiple frequencies.

Chapter 4

Image compressive sensing

One of the more intuitive applications of CS lies in spatial signals as it is easier to visualize. In this scheme, the process can be simplified either by flattening it to one dimension and processing it in its entirety, or maintaining its dimensionality and processing it by patches. The general workflow that arises from image CS is as follows:

1. Define the compression ratio m/n, where n is the signal size, and m is the desired size of the compressed signal.

2. Draw m random indices from the signal without replacement and store this as a sample sequence $\boldsymbol{\xi}$.

3. Extract the row vectors of the desired $n \times n$ sparsifying basis $\boldsymbol{\Psi}$ indexed by $\boldsymbol{\xi}$, and stack these to form the sensing matrix $\boldsymbol{\Phi}$ (i.e., $\boldsymbol{\Phi} = \boldsymbol{\Psi}_{\boldsymbol{\xi}}$)

4. With the desired reconstruction algorithm, perform the optimization (2.8) to obtain the reconstructed signal $\hat{\mathbf{x}}$.

In the case of high-definition images (whose shortest side is at least 720 pixels), it is usually more practical and yields better results if the image is processed in patches.

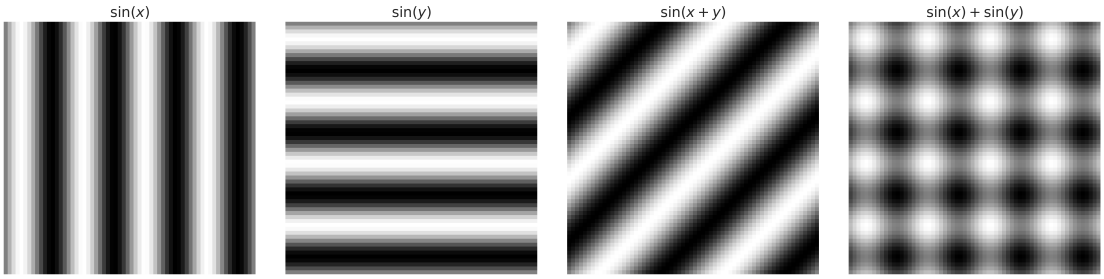
4.1 Test case: Sinusoidal pattern

As mentioned in Chapter 2, the most commonly used sparse representation domain for images is the Fourier domain, referred to in some fields as k-space. In this space, signals are represented as a linear superposition of a finite number of sinusoidal patterns. In Fig. 4.1a, 64×64 pixel sinusoidal patterns are generated, corresponding to sine waves traveling horizontally, vertically, and diagonally, as well as an egg tray pattern. In each case, all frequency components are 4 Hz. Figure 4.1b visualizes the compressed image when a random sample of 5% is taken from the signal. The actual compressed signal that is seen by the reconstruction algorithm is a one-dimensional sequence containing only the information from the points being sampled. Orthogonal matching pursuit (OMP) was used for reconstruction, which is a greedy algorithm that finds the combination of basis vectors which best represents the signal (similar to matching pursuit), but in addition, the residual at each iteration is recomputed using an orthogonal projection on the set of previously selected basis vectors [23]. Its objective function is

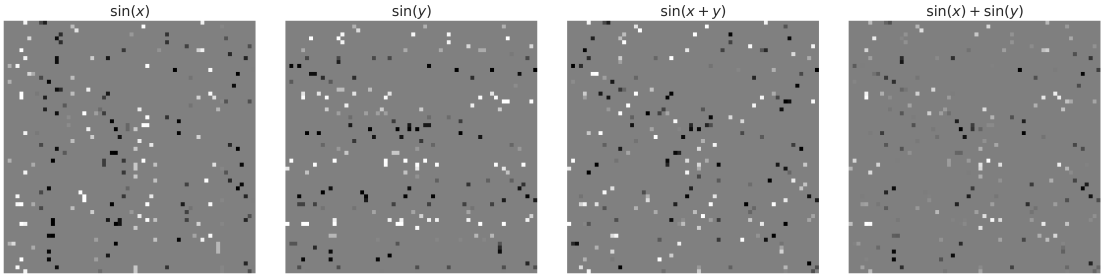
$$\arg\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{\Phi x}\|_2^2 \quad \text{subject to} \quad \|\mathbf{x}\|_0 \leq \gamma \tag{4.1}$$

where γ is a hyperparameter which controls the maximum allowable number of non-zero coefficients. The `Scikit-learn` implementation sets this value to 10% of the number of samples by default [21]. Evaluation of the mean-squared error

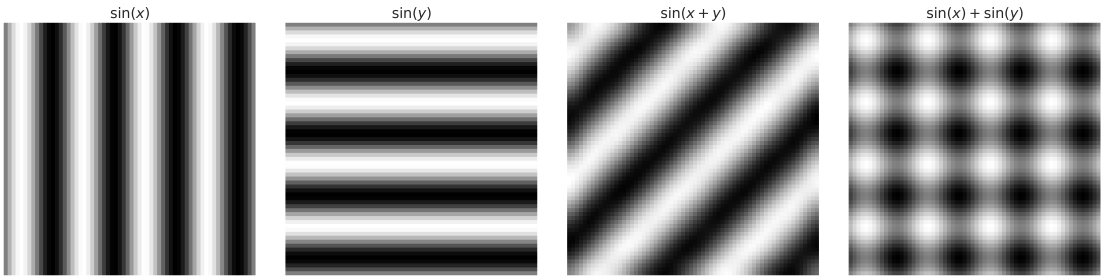
(MSE) for the pure horizontal and pure vertical sine waves, as well as the egg tray pattern yields a value that is practically negligible ($\approx 10^{-31}$); the reconstruction is exact. On the other hand, the reconstructed diagonal sine wave yields an MSE of 10^{-3}—still quite small, but mild distortion can be observed at the image boundaries. This is due to the fact that the information at hand is finite, and so is the window size which, in this case, is the same size as the signal itself.



(a) Test 2D sinusoid patterns.



(b) Visualization of sinusoid patterns with compression ratio of 5%.



(c) Reconstruction from 5% of samples.

Figure 4.1: Test 64×64 pixel 2D sinusoid patterns corresponding to vertical sinusoids, horizontal sinusoids, diagonal sinusoids, and egg tray pattern. All frequency components are 4 Hz.

4.2 Image with multiple sinusoids

4.2.1 Pre-processing

For this section, the image used is M.C. Escher's *Relativity*, an example of a more complex image but consisting of dominant sinusoidal patterns that are made apparent when you zoom in. The original image has dimensions of 1600×981 pixels, for a total of 1,569,600 pixels. Following the procedure with the previous section, this would require the construction of a 1,569,600 $\times$ 1,569,600 sparsifying matrix containing $\approx 2 \times 10^{12}$ entries. Assuming that the matrix would be stored as 32-bit floating point numbers, this process alone would take up ≈ 8 GB of memory, and it would be highly impractical to process similarly-sized images as a whole. The workaround is to split it into smaller, manageable patches.

4.2.2 Processing

For this image in particular, it was first resized to 1600×976 pixels so that it could be equally divided into a grid of 16×16, each with a dimension of 100×61 pixels. After compressively sampling each patch at 40% compression ratio, reconstruction was performed using the Embedded Conic Solver (ECOS) of the Convex Optimization Python library (CVXPY), which recasts (2.8) as a convex problem and directly minimizes the ℓ_1 norm [19, 20, 24] and thus, is significantly slower compared to OMP.

4.2.3 Reconstruction evaluation

To quantify the reconstruction quality, the Structural Similarity Index (SSIM) [25] was used. This is a perception-based model that takes into account perceptual

factors such as luminance, contrast, and structure. SSIM is calculated on windows in the image, and is defined as

$$\text{SSIM}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{(2\mu_{\mathbf{x}}\mu_{\hat{\mathbf{x}}} + c_1)(2\sigma_{\mathbf{x}\hat{\mathbf{x}}} + c_2)}{(\mu_{\mathbf{x}}^2 + \mu_{\hat{\mathbf{x}}}^2 + c_1)(\sigma_{\mathbf{x}}^2 + \sigma_{\hat{\mathbf{x}}}^2 + c_2)} \tag{4.2}$$

where $\mathbf{x}, \hat{\mathbf{x}}$ are the original and reconstructed signals, respectively, μ are the image means, σ are the image standard deviations, and c are constants to stabilize division with a small denominator. SSIM values of 0.8 and above are considered acceptable.

After stitching all patches at the end, the reconstructed image is shown in Fig. 4.3. Evaluation of SSIM yields a value of 0.88, way above the acceptable threshold. Selected patches with the aforementioned dominant patterns are shown with their reconstructed counterparts in Fig. 4.4, corresponding to patches dominated by horizontal sinusoids, vertical sinusoids, diagonal sinusoids, multiple sinusoids, and patches with no dominant pattern.

We can observe that at this compression rate, the patches with a single apparent sinusoidal pattern (Figs. 4.4a-4.4c) are successfully recovered, with some noise present especially for the patch with a dominant diagonal pattern (similar to the previous section). The patch with multiple sinusoid patterns (Fig. 4.4d), although still recognizable, is laden with a lot of noise. Lastly, the patch with no apparent pattern (Fig. 4.4e) is barely recognizable, except for the portions where a dominant sinusoidal pattern is partially present in the frame.

From this, the following information can be gleaned. First, reconstruction performs better on smaller patches, and when the patch in question contains as few frequency components as possible (such is the case with the patches with only one dominant pattern). Second, the patch with no dominant pattern—upon closer

visual inspection—can be classified as being successfully recovered; however, the reconstruction noise is almost at the same level as the signal itself, which makes them indistinguishable. This can be attributed to the fact that the patches with no apparent dominant pattern are actually composed of a superposition of sinusoids residing primarily in the high-frequency region of k-space. Since the sampling points are uniformly distributed throughout the spatial domain, so are they in the frequency domain. Thus, the information in the high-frequency region is not sufficiently captured, and a higher compression ratio is required to be able to better recover these high-frequency regions. Another solution would be, as mentioned earlier, to make the patches smaller so that lesser frequencies are captured in one patch.

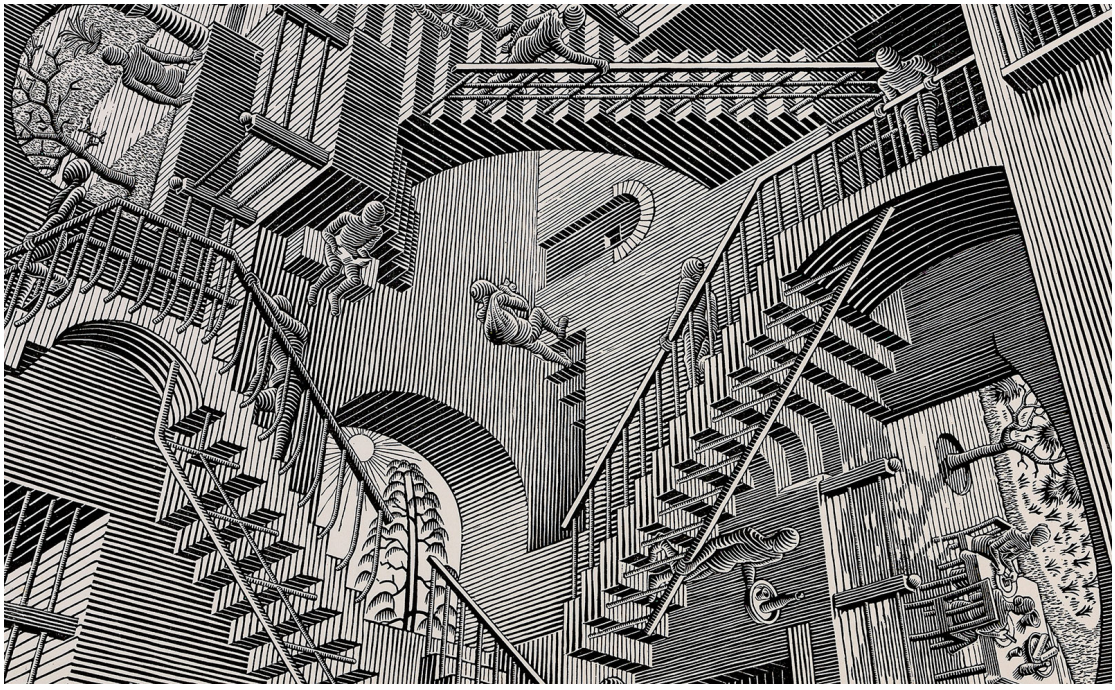


Figure 4.2: *Relativity* by M.C. Escher, a complex image consisting of various sinusoidal patterns.

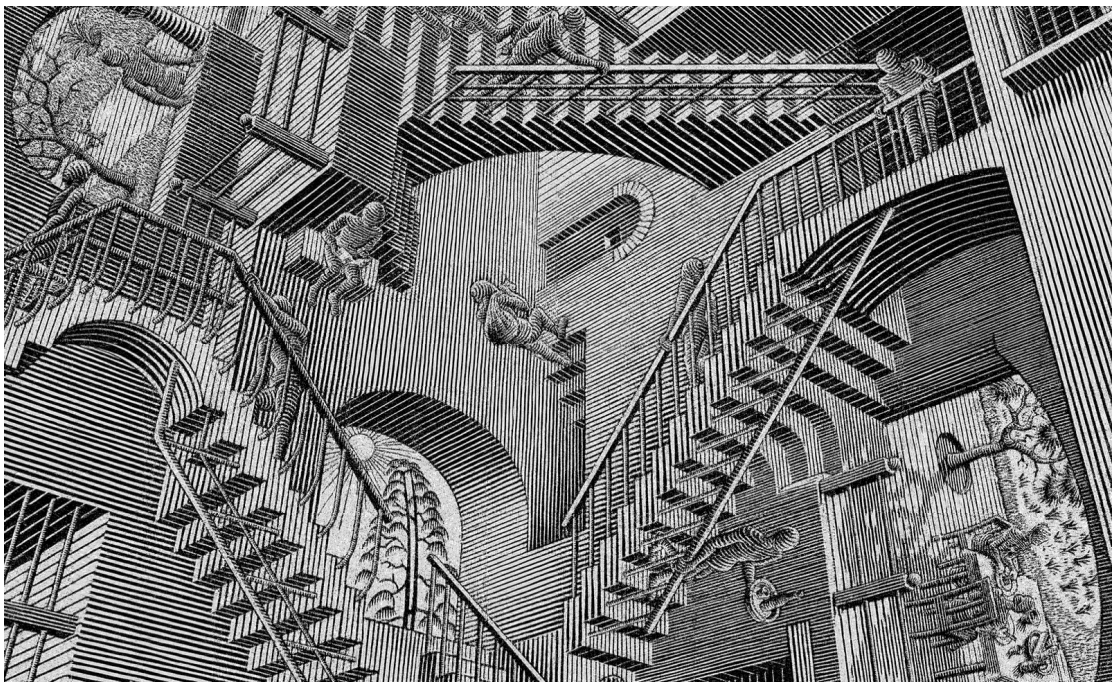


Figure 4.3: Reconstructed *Relativity* from 50% of samples from each patch.

4.3 Simultaneous compression & encryption

Because of the way compressively sampled images are coded with the sensing matrix, the use of CS as an encryption algorithm arises naturally. Consider the logistic map

$$x_{n+1} = r x_n (1 - x_n) \tag{4.3}$$

which is often used as an archetypal example of deterministic chaotic behavior for values of $r \in [3.57, 4)$. In this regime, the sequences produced by varying the initial parameter x_0 rapidly diverge from each other. Thus, this can become an encryption system by treating the parameters r and x_0 as an encryption key set, and (4.3) as the hash function. One key set is then needed for each signal dimension. Thus,

two key sets are needed for grayscale image applications (six, in the case of an RGB image and a unique key set for each color channel).

4.3.1 Methodology

The construction of the sensing matrix $\mathbf{\Phi}$ differs from the general workflow, and is as follows:

1. From (4.3), generate a sequence of length $2m$ with the initial key pair r_1 and x_{01}. Discard the first n elements to avoid the transient response and store the latter n elements as a sequence $\mathbf{s}$.

2. Explicitly generate the index sequence of $\mathbf{s}$ and store it as sequence $\mathbf{p} = [0, 1, ..., m-1]$.

3. Sort $\mathbf{p}$ according to ascending values of $\mathbf{s}$.

4. Generate the first sensing matrix $\mathbf{\Phi}_1$ by extracting and stacking rows of a Hadamard matrix $\mathbf{H}$ of order N indexed by the first m elements of $\mathbf{p}$, i.e.,

$$\mathbf{\Phi}_1 = \begin{bmatrix} \mathbf{H}_{p_1} & \mathbf{H}_{p_2} & \cdots & \mathbf{H}_{p_m} \end{bmatrix}^\top \tag{4.4}$$

   where $\mathbf{H}_{p_i}$ denotes the p_ith row vector of $\mathbf{H}$.

5. Any other sensing matrices can be constructed in the same way.

The usage of a Hadamard matrix implies that the image must first be reshaped to have dimensions that are integer multiples of 4. The original image is first reshaped to 256×256 pixels, and is sparsified by transforming it to the DCT domain. The desired compressed dimension is set to $m = 192$, corresponding

to a compression ratio of 75%, and the keys are set to values of $r_1 = r_2 = 3.99$, $x_{01} = 0.11$, and $x_{02} = 0.24$.

4.3.2 Results & discussion

Figure 4.5 shows the application of this to the Lena test image. Visual inspection of the encrypted representation shows horizontal and vertical bands distributed throughout the representation space, and is indicative that a simple inverse Fourier transform will not recover any meaningful information. Assuming that the receiver knows the encryption scheme, recovery of the original message is successful if the same keys r_1, r_2, x_{01}, x_{02} as the encryption stage are used, which will allow the receiver to construct the exact same sensing matrices $\mathbf{\Phi}_1, \mathbf{\Phi}_2$ and perform the inverse operation on the encrypted message. In the decrypted image, encryption artifacts can be observed, as indicated by some visible banding, but is nonetheless recognizable; evaluation of the MSE and SSIM yields values of 0.02 and 0.82, respectively.

4.3.3 Correlation analysis

Correlation is a statistical measure of obtaining useful information from a given signal by analyzing adjacent pixels. The correlation coefficients can be obtained by

$$C = \frac{\sum_{i=1}^{N}(x_i - \langle \mathbf{x} \rangle)(y_i - \langle \mathbf{y} \rangle)}{\sqrt{\sum_{i=1}^{N}(x_i - \langle \mathbf{x} \rangle)^2 \sum_{i=1}^{N}(y_i - \langle \mathbf{y} \rangle)^2}} \tag{4.5}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the original and encrypted signals, respectively. For the purposes of encryption, a correlation coefficient closer to 0 is better. Table 4.1 shows the computed correlations for the original and encrypted Lena images in the horizontal,

Table 4.1: Correlation coefficients of test Lena image.

	horizontal	vertical	diagonal
original	0.94	0.97	0.91
encrypted	0.42	0.02	−0.05

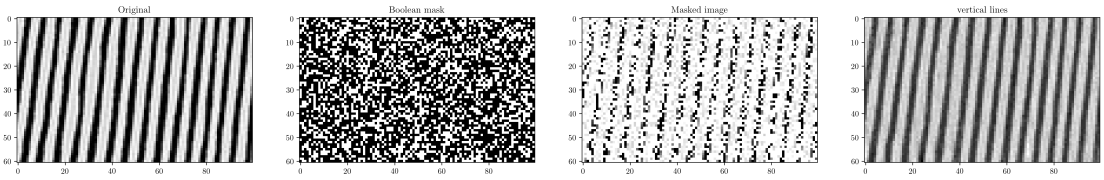
vertical, and diagonal directions. The correlation of the encrypted signal is much lower than that of the original, which means that a potential attacker cannot gain any useful information by employing statistical analyses.

4.3.4 Key sensitivity analysis

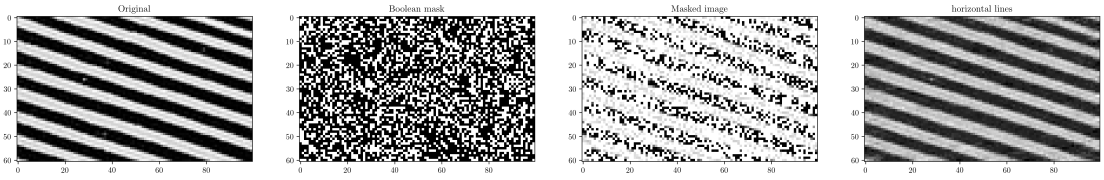
With the knowledge that the hash function (4.3) is chaotic, the sensitivity of the encryption system to the keys can be tested by perturbing the initial keys with small values. Figure 4.6 shows the decryption results when all the correct keys are used, except for x_{01}, which is perturbed by a tiny value $\approx 10^{-15}$ (third image), and similarly when the correct x_{01} is used but x_{02} is perturbed by the same amount (last image).

Additionally, Fig. 4.7 shows the MSE curves evaluated by varying values of the perturbation Δx_{01} and Δx_{02}. Perturbations are generated as 100 equally-spaced values in the range $\pm 10^{-14}$, plus zero. This specific value was chosen because the resolution of a double precision floating point number in Python on a 64-bit CPU architecture is 10^{-16}. The MSE generally oscillates at some high value, and exhibits a sharp dip when the MSE is evaluated for the correct keys ($\Delta x_0 = 0$). The same is done for Fig. 4.8, which shows the SSIM curves. Similarly, values oscillate at a low value, corresponding to recovered signals with no meaningful

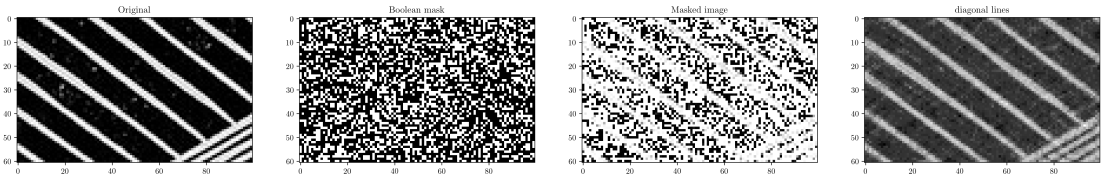
data. Maximum SSIM of 0.83 is achieved only for the correct keys. This shows that brute force attacks are intractable against this kind of encryption system.



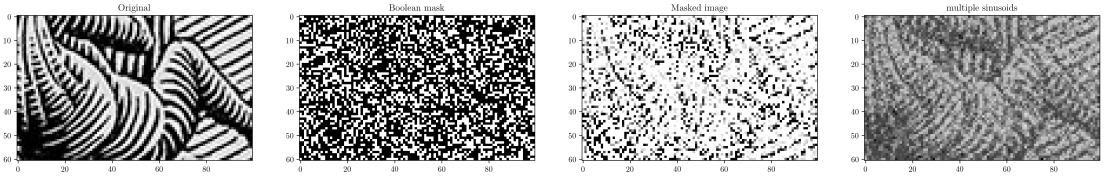
(a) Dominant vertical



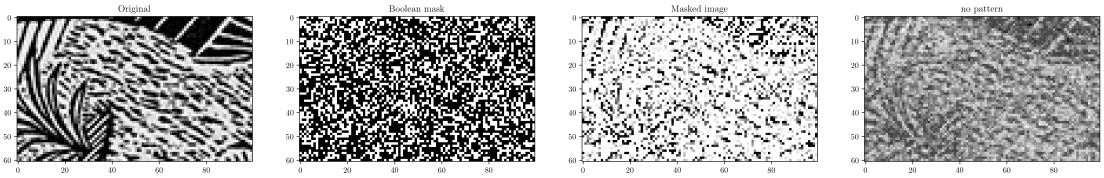
(b) Dominant horizontal



(c) Dominant diagonal



(d) Multiple dominant sinusoids



(e) No dominant pattern

Figure 4.4: Extracted and reconstructed patches from *Relativity* using 40% of samples.

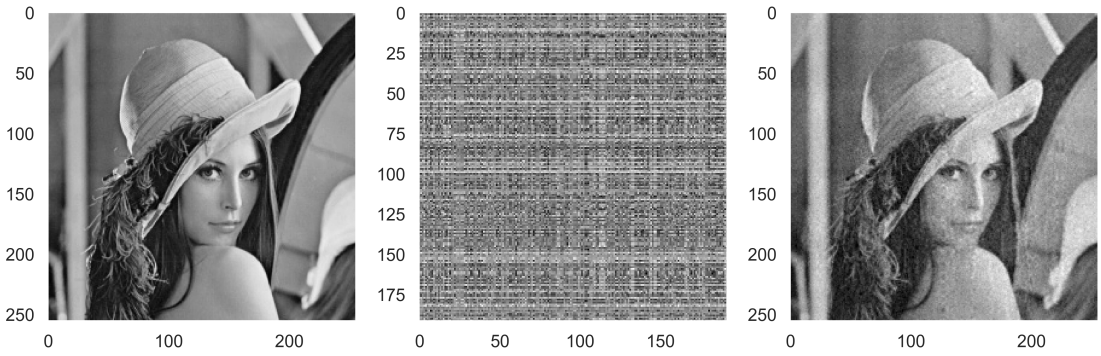


Figure 4.5: Simultaneous compression and encryption achieved with compressive sensing: original image (left), encrypted image (middle), and decrypted/reconstructed image (right).

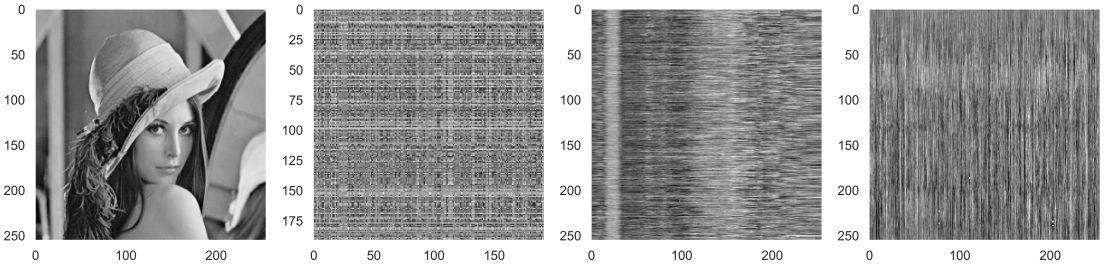
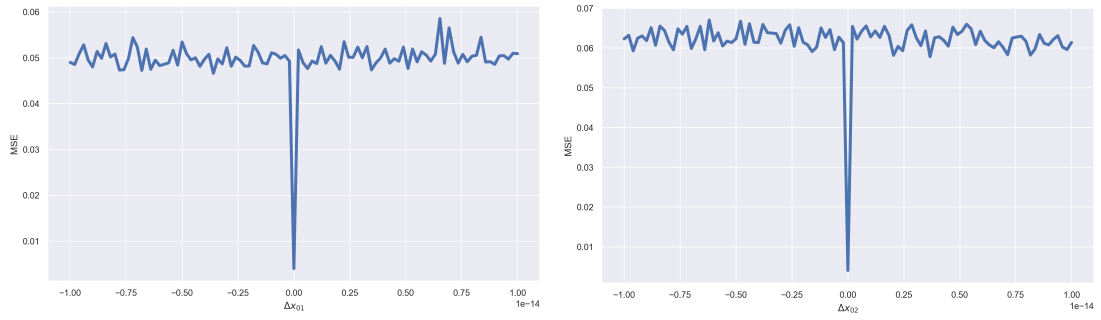


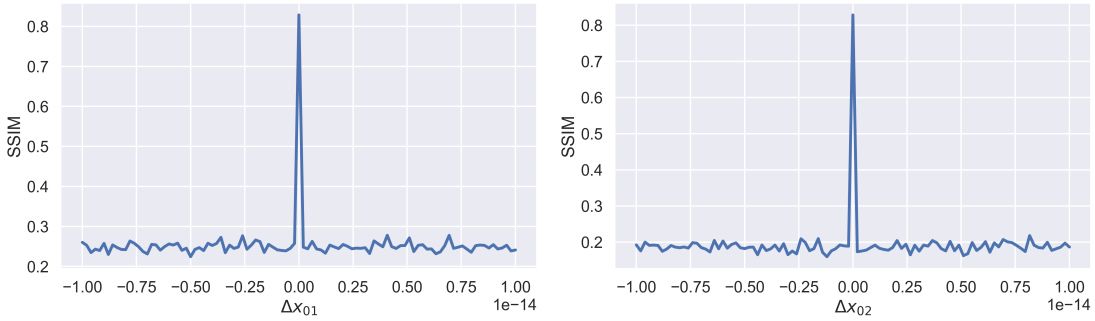
Figure 4.6: Test image Lena (first) with the encrypted representation (second), the decryption result when the correct keys are used but x_{01} is perturbed by a value of 10^{-15} (third), and the decryption result when the correct keys are used but x_{02} is perturbed by a value of 10^{-15}.



(a) Perturbation of Δx_{01}

(b) Perturbation of Δx_{02}

Figure 4.7: MSE curves resulting from evaluation of reconstruction error for tiny perturbations in the initial values Δx_{01} and Δx_{02}.



(a) Perturbation of Δx_{01}

(b) Perturbation of Δx_{02}

Figure 4.8: SSIM curves resulting from evaluation of reconstruction error for tiny perturbations in the initial values Δx_{01} and Δx_{02}.

Chapter 5

Audio compressive sensing

In this chapter, I apply CS to audio signals. These type of signals act as the bridge to N-dimensional CS as they are one-dimensional when represented in the time domain, but are projected to higher dimensions when represented in another domain, such as the spectrogram/modulation domain. Unlike images, audio signals are a tad harder to compressively sample. Due to their relatively higher information density, the effects of undersampling are easily observed.

5.1 Test case: Sinusoid redux

In this test case, I recorded a guitar playing a single E_4 (330 Hz) note at the standard 44.1 kHz sampling rate for 4 seconds. Since the Nyquist rate of the actual signal is 660 Hz, the recording can be downsampled to a practical 8 kHz for processing. The signal waveform and frequency content is shown in Fig. 5.1a. The base frequency is dominant in the frequency spectrum, and several harmonics can be observed. The goal here is to be able to recover the harmonics that have a frequency higher than the compressive sampling rate.

The compressed signal is shown in Fig. 5.1b, which was compressively sampled with a quasi-frequency of 1000 Hz (1000 i.i.d. random samples per second), corresponding to a 12.5% compression ratio. The waveform envelope still resembles that of the original, but due to the random nature of sampling, the periodicity is not preserved, and is reflected in the seemingly random frequency content.

Following a similar process shown in Chapter 3, I chose DCT to be the sparse representation domain, and LASSO as the optimization algorithm. The reconstructed signal is shown in Fig. 5.1c.For this case, I am concerned only with the frequency components that are recovered, and not so much with the magnitude. Thus, the reconstruction quality can be quantified using the cosine similarity

$$\text{similarity} = \cos\theta = \frac{\mathbf{x} \cdot \hat{\mathbf{x}}}{\|\mathbf{x}\|_2 \|\hat{\mathbf{x}}\|_2} \tag{5.1}$$

which allows us to compare two signals' frequency content directly in the time domain. A cosine similarity value of 0.8 and above indicates acceptable quality; a value of 1.0 indicates perfect reconstruction.

5.2 Comparison of algorithms

Following the same procedure as the previous section, my aim now is to compare the performance of three different reconstruction algorithms in terms of average runtime and reconstruction quality. The algorithms used are LASSO and OMP, which were described in Chapter 2. Additionally, the Smoothed L_0 Norm (SL0) [26] is used, which approximates the ℓ_0 norm using a Gaussian of the form

$$\lim_{\sigma \to 0} x \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{5.2}$$

While all algorithms have polynomial time complexity [27–29], OMP shows the worst scaling with respect to time; LASSO and SL0 show similar performance over time (Fig. 5.2a). In terms of reconstruction quality (cosine similarity), LASSO is able to breach the 0.8 threshold at 30% compression ratio, while SL0 achieves this at 50% compression. On the other hand, OMP shows a nonlinear trend with a large error, which is indicative of unstable performance for low compression ratios (Fig. 5.2b).

5.3 Speech

In order to show its practical merits, we will inevitably have to deal with increasingly large and complex signals. Audio recordings containing speech will encompass a wide range of frequencies, so such signals can only be downsampled so much before essential information is lost to aliasing. Unlike images, large audio signals cannot simply be chopped into smaller, manageable pieces. The effects of aliasing are amplified due to the high information density, and CS' violation of periodic constraints introduce artifacts in the vicinity of where the signal was sliced. This is the motivation for transforming the signal first into the modulation domain (spectrogram).

5.3.1 Sparse transformation

In obtaining the spectrogram representation, first define a short length sampling window, typically only a few milliseconds in duration, as well as the overlap between adjacent frames. The latter is crucial in suppressing boundary artifacts as it ensures that some information from the current measurement is carried over to the next

measurement. The signal is then divided into frames by sliding this window across the entire signal. Each frame is multiplied with a window function; in this case, I used the Hann window, defined as

$$w[n] = \sin^2\left(\frac{\pi n}{N}\right) \tag{5.3}$$

where $N + 1$ is the length of the window, and $n : 0 \leq n \leq N$ is the frame index. Finally, each frame undergoes a Fourier transformation. The entire process is also called a short-time Fourier transform, and is summarized as

$$X(\omega, p) = \sum_{p=0}^{P-1} x[p]w[p - kR]e^{-i\omega p} \tag{5.4}$$

where $x[p]$ is the pth signal frame, $w[p - kR]$ is the window function with hop size R and time index k, and ω is the angular frequency.

5.3.2 Pre-processing

Test signals were obtained from the TIMIT Acoustic-Phonetic Continuous Speech Corpus [30], which contains speech recordings in `WAV` format. The recordings are of English speakers grouped by region, sex, and unique spoken sentence. All files have a sampling rate of 16 kHz and are, on average, 3 seconds long. I chose a test signal at random, specifically the `DR8/MJLN0/SA1.wav` file. This indicates that the speaker was from dialect region 8 (nomadic), was male with speaker code `JLN0`, and spoke unique sentence `SA1`, which reads

> She had your dark suit in greasy wash water all year.

Before proceeding, I downsampled the file to 8 kHz. The representation of the signal in the time and modulation domains are shown in Fig. 5.3a.

5.3.3 Processing

I compressively sampled the signal with a compression ratio of 40%, using 1024 frames and 75% frame overlap. Following the results from Sec. 5.2, I used the LASSO algorithm for reconstruction, once again obtaining the optimal regularization parameter α by 5-fold cross validation.

5.3.4 Reconstruction evaluation

The reconstruction quality was quantified using the International Telecommunication Standardization Sector (ITU-T) recommendation P.862 [31], otherwise known as the Perceptual Evaluation of Speech Quality (PESQ). This metric is a full-reference, perceptually intuitive scoring system which models the now-obsolete mean opinion scores (MOS). This algorithm performs a series of standardized tests modeled after qualitative metrics, analyzes and compares the original and reconstructed signals, and returns a value from 1.0 (bad) to 5.0 (perfect). Because real reconstructed signals are rarely exactly the same as the original, the PESQ values are usually thresholded up to 4.5 (excellent). PESQ values of 3.0 and above indicate acceptable quality.

For a more quantitative test, I also used the average segmental signal-to-noise ratio (SNR_{seg}) [32], defined as

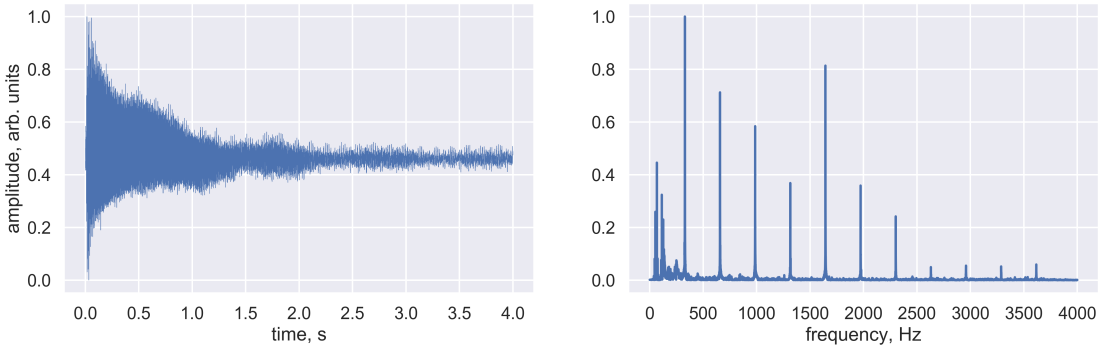
$$\text{SNR}_{\text{seg}} = \frac{10}{B}\sum_{b=0}^{B-1}\log_{10}\frac{\sum_{i=Nb}^{Nb+N-1}x_i^2}{\sum_{i=Nb}^{Nb+N-1}(x_i-\hat{x}_i)^2} \tag{5.5}$$

where N is the frame length, B is the number of frames, x_i are the original signal samples, and $\hat{x}_i$ are the reconstructed signal samples.

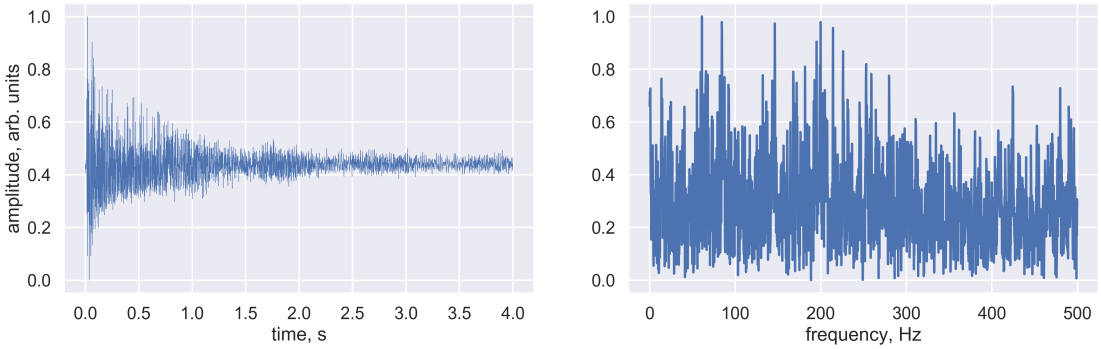
Figure 5.3b shows the reconstructed signal. Qualitative comparison in the time domain shows that the original and reconstructed waveforms are structurally similar. In the modulation domain, the dynamic range of the latter seems to have diminished, but the dominant frequencies can still be observed. Evaluation of the PESQ and SNR_{seg} yields values of 2.50 and 0.07, respectively. At face value, I can immediately tell from the PESQ that the reconstructed signal quality is slightly below average; listening to the reconstructed recording reveals a noticeable level of noise in the background. However, the same distinction cannot be made for the SNR_{seg} since its bounds are not well-defined.

5.3.5 Error space analysis

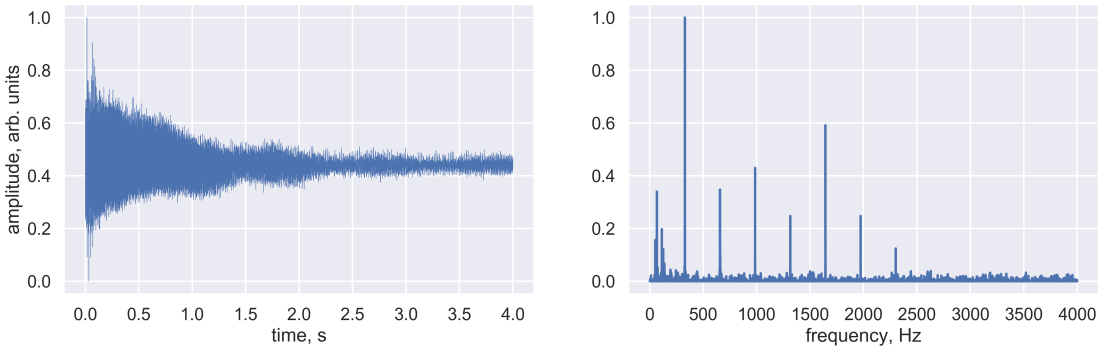
Using the same test signal, I generated the error space maps by compressively sampling the signal and evaluating the metrics for varying compression ratios $\in [0.1, 0.9]$ in increments of 0.1, and varying number of frames $\in \{128, 256, 512, 1024\}$, while keeping the frame overlap constant at 75%. Figure 5.4 shows the PESQ and SNR_{seg} maps. The former exhibits a sensitivity to the compression ratio, and achieves the acceptable threshold of 3.0 at around 60% compression. The latter shows sensitivity towards the number of frames (as it is an *average* metric) with some additional degradation below 40% compression ratio. It achieves a maximum value of 0.08 at around 1024 frames.



(a) Original

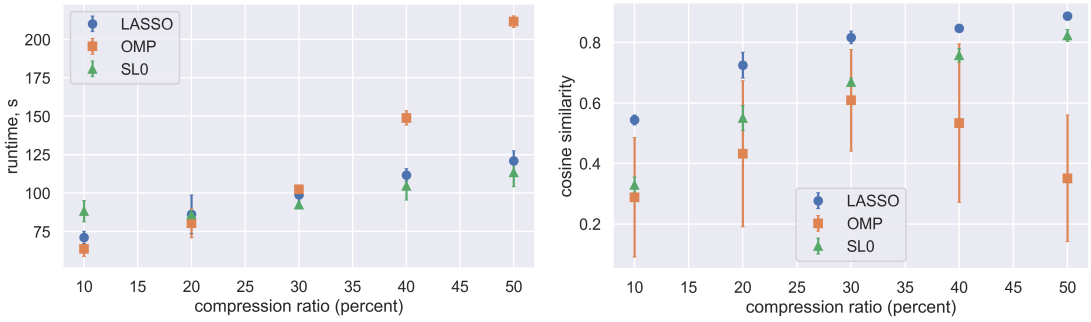


(b) Compressed



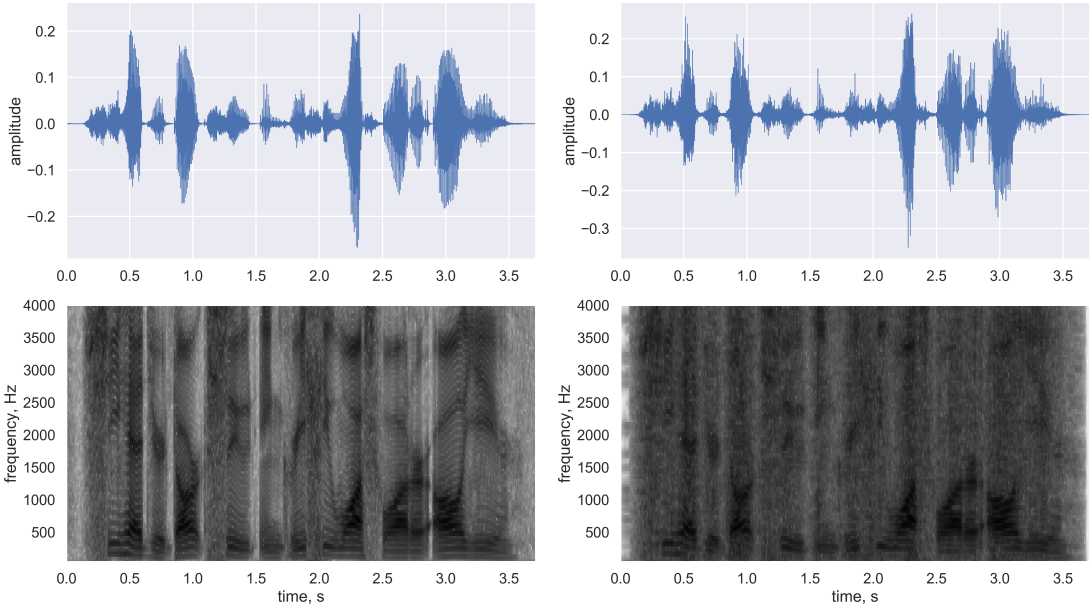
(c) Recovered

Figure 5.1: 330 Hz guitar signal representation in the time domain (left column) and frequency domain (right column).



(a) Runtime (b) Reconstruction quality

Figure 5.2: Comparison of the performance of LASSO, OMP, and SL0.



(a) Original (b) Recovered

Figure 5.3: Test speech signal in the time domain (top row) and modulation domain (bottom row).

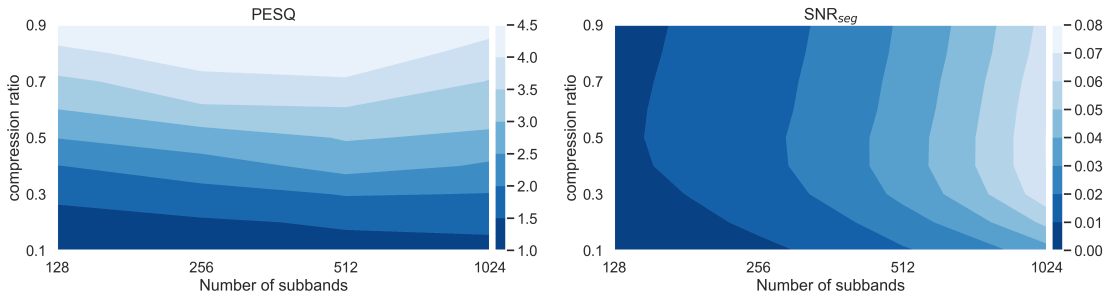


Figure 5.4: PESQ and SNR_{seg} error space maps as a function of compression ratio and number of subbands.

Chapter 6

Conclusions and recommendations

In this study, I investigated the use of compressive sensing for various applications, and as a viable alternative sampling theorem. In the compressive sampling of signals, there are similar workflows that can be followed separately for image-based and audio-based signals. Both follow a common workflow in the encoding stage, but differ slightly in the decoding stage. Depending on the size of the signal, both types can be processed in one go, but more often than not, the signal has to be processed in several manageable slices. Image-based signals can be sliced into patches with no overlap, while audio-based signals can be divided into subbands with significant overlap in order to suppress reconstruction artifacts.

Due to the sparsity and incoherence requirements of CS, the ideal starting point is by using partial DCT matrices encoded by randomly distributed samples. As for the random distribution, uniform distribution achieves the lowest

reconstruction error, while triangular distribution achieves more consistent results throughout a wide range of compression ratios.

Bibliography

[1] M. R. Mathew and B. Premanand, Sub-Nyquist Sampling of Acoustic Signals Based on Chaotic Compressed Sensing, *Procedia Technol.* **24**, 941 (2016), ISSN 22120173.

[2] I. Andráš, P. Dolinský, L. Michaeli, and J. Šaliga, A time domain reconstruction method of randomly sampled frequency sparse signal, *Meas. J. Int. Meas. Confed.* **127**, 68 (2018), ISSN 02632241.

[3] S. Y. Low, D. S. Pham, and S. Venkatesh, Compressive speech enhancement, *Speech Commun.* **55**, 757 (2013), ISSN 01676393.

[4] S. Y. Low, Compressive speech enhancement in the modulation domain, *Speech Commun.* **102**, 87 (2018), ISSN 01676393.

[5] V. Abrol, P. Sharma, and A. K. Sao, Voiced/nonvoiced detection in compressively sensed speech signals, *Speech Commun.* **72**, 194 (2015), ISSN 01676393.

[6] Y. Mo, A. Zhang, F. Zheng, and N. Zhou, An image compression-encryption algorithm based on 2-D compressive sensing, *J. Comput. Inf. Syst.* **9**, 10057 (2013), ISSN 15539105.

[7] N. Zhou, S. Pan, S. Cheng, and Z. Zhou, Image compression-encryption scheme based on hyper-chaotic system and 2D compressive sensing, *Opt. Laser Technol.* **82**, 121 (2016), ISSN 00303992.

[8] R. A. Romero, G. A. Tapang, and C. A. Saloma, in *Proceedings of the Samahang Pisika ng Pilipinas Physics Conference* (University of the Philippines Visayas, Iloilo City, Philippines, 2016), vol. 34, SPP–2016–PA–14.

[9] S. Liu, M. Gu, Q. Zhang, and B. Li, Principal component analysis algorithm in video compressed sensing, *Optik (Stuttg).* **125**, 1149 (2014), ISSN 00304026.

[10] J. Chen, K. X. Su, W. X. Wang, and C. D. Lan, Residual distributed compressive video sensing based on double side information, *Zidonghua Xuebao/Acta Autom. Sin.* **40**, 2316 (2014), ISSN 02544156.

[11] E. J. Candès, J. K. Romberg, and T. Tao, Stable signal recovery from incomplete and inaccurate measurements, *Commun. Pure Appl. Math.* **59**, 1207 (2006), ISSN 00103640, arXiv:0503066v2.

[12] D. L. Donoho, Compressed sensing, *IEEE Trans. Inf. Theory* **52**, 1289 (2006), ISSN 00189448.

[13] D. Donoho and X. Huo, Uncertainty principles and ideal atomic decomposition, *IEEE Trans. Inf. Theory* **47**, 2845 (2001), ISSN 00189448.

[14] D. L. Donoho and M. Elad, Optimally sparse representation in general (nonorthogonal) dictionaries via L1 minimization, *Proc. Natl. Acad. Sci. U. S. A.* **100**, 2197 (2003), ISSN 00278424.

[15] N. Linh-Trung, D. Van Phong, Z. M. Hussain, H. T. Huynh, V. L. Morgan, and J. C. Gore, Compressed sensing using chaos filters, *Proc. 2008 Australas. Telecommun. Networks Appl. Conf. ATNAC 2008* 219–223 (2008).

[16] C. E. Shannon, Communication in the presence of noise, *Proceedings of the Institute of Radio Engineers* **37**, 10 (1949).

[17] E. J. Candes and M. B. Wakin, An introduction to compressive sampling: A sensing/sampling paradigm that goes against the common knowledge in data acquisition, *IEEE Signal Process. Mag.* **25**, 21 (2008), ISSN 10535888.

[18] CCITT Study Group VIII and Joint Photographics Expert Group, *T.81 – Digital compression and coding of continuous-tone still images – Requirements and guidelines* (1982).

[19] S. Diamond and S. Boyd, CVXPY: A Python-embedded modeling language for convex optimization, *Journal of Machine Learning Research* **17**, 1 (2016).

[20] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, A rewriting system for convex optimization problems, *Journal of Control and Decision* **5**, 42 (2018).

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research* **12**, 2825 (2011).

[22] R. Rubinstein, M. Zibulevsky, and M. Elad, Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit, *CS Tech.* 1–15 (2008).

[23] S. G. Mallat and Z. Zhang, Matching Pursuits With Time-Frequency Dictionaries (1993).

[24] A. Domahidi, E. Chu, and S. Boyd, in *European Control Conference (ECC)* (2013), 3071–3076.

[25] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, Image Quality Assessment: From Error Visibility to Structural Similarity, *IEEE Trans. Image Process.* **13**, 600 (2004), ISSN 1057-7149.

[26] H. Mohimani, M. Babaie-Zadeh, and C. Jutten, A fast approach for overcomplete sparse decomposition based on smoothed L0 norm, *IEEE Trans. Signal Process.* **57**, 289 (2009), ISSN 1053587X, arXiv:arXiv:0809.2508v2.

[27] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, Least angle regression, *The Annals of Statistics* **32**, 407 (????).

[28] B. L. Sturm and M. G. Christensen, in *20th European Signal Processing Conference (EUSIPCO 2012)* (2012), 220–224.

[29] J. Xiang, H. Yue, X. Yin, and L. Wang, A new smoothed L0 regularization approach for sparse signal recovery, *Mathematical Problems in Engineering* (2019).

[30] J. S. Garafolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue, *TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1* (Linguistic Data Consortium, 1993).

[31] Telecommunication Standardization Sector of ITU, Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs, *ITU-T Recommendation P.862 (02/01)* (2001).

[32] P. C. Loizou, *Speech Enhancement: Theory and Practice* (CRC Press, 2013), 2nd ed.

Appendix A

Codes and Implementations

Listed in this section are the core codes used in the study. Complete source codes are available at my private GitHub repository. Access can be requested via email at kvdomingo@up.edu.ph.

Listing A.1: Code for compressive sensing of 1D test sinusoids.

```
import numpy as np
import numpy.random as rand
import scipy.fftpack as fft

# load recording
signal = np.loadtxt("piano.txt").astype("float32")

# define parameters
samprate = 44.1e3
duration = 1/8
N = int(duration*samprate)
M = 300
t = np.linspace(0, duration, N)

# extract short portion of recording
sig_start = 40000
x = signal[sig_start:sig_start + N]

# simulate compressive measurements
yi = rand.randint(0, N, M)
yi = np.sort(yi)
```

```
y = x[yi]

# L1 optimization using CVX ECOS
import cvxpy as cvx
xhat_cvx = cvx.Variable(N)
objective = cvx.Minimize(cvx.Norm(xhat_cvx, 1))
constraints = [A*xhat_cvx == y]
prob = cvx.Problem(objective, constraints)
result = prob.solve(verbose=True, solver="ECOS")
x_cvx = np.array(xhat_cvx.value)
x_cvx = np.squeeze(x_cvx)
x_cvx = fft.dct(x_cvx, norm="ortho", axis=0)

# L1-regularized L2 optimization using LASSO
from sklearn.linear_model import Lasso, LassoCV
lasso = LassoCV(cv=10, random_state=0, verbose=True, n_jobs=-1)
lasso.fit(A, y)
x_lasso = fft.idct(lasso.coef_)
```

Listing A.2: Code for implementation of GPU-accelerated SL0, translated from original MATLAB version.

```
import numpy as np
import numpy.random as rand
import tensorflow as tf
import scipy.fftpack as fft

# define L0 norm estimator function
def Fsigma(x_i, sigma):
    x = tf.abs(x_i)
    return x_i * tf.exp(-x**2/(2*sigma**2))

def SL0(A, b, Fsigma=Fsigma, sigma_min=1e-12, mu_0=2, L=3, sdf=0.5):
    A = tf.convert_to_tensor(A, dtype=tf.float32)
    b = tf.convert_to_tensor(b, dtype=tf.float32)
    A_plus = tf.linalg.pinv(A)
    x = tf.linalg.matmul(A_plus, b)
    sigma = 2 * np.max(np.abs(x.numpy()))
    while sigma > sigma_min:
        for i in range(L):
            delta = Fsigma(x, sigma)
            x -= mu_0 * delta
            x -= tf.linalg.matmul(A_plus, tf.linalg.matmul(A, x) - b)
        sigma *= sdf
    return x.numpy()
```

Listing A.3: Simultaneous compression-encryption with SL0.

```
import numpy as np
```

```
2   import numpy.random as rand
3   import scipy.fftpack as fft
4   import cv2 as cv
5
6   def encrypt(filename, compression, key1, key2, key3):
7       if isinstance(filename, str):
8           X = cv.imread(filename, 0)
9       if isinstance(filename, np.ndarray):
10          X = filename.copy()
11      N = len(X)
12      M = int(compression * N)
13
14      lamda1 = [key1]
15      lamda2 = [key2]
16      for i in range(1, 2*N):
17          lamda1.append(key3*lamda1[i-1]*(1 - lamda1[i-1]))
18          lamda2.append(key3*lamda2[i-1]*(1 - lamda2[i-1]))
19
20      s1 = np.array(lamda1[N:])
21      s2 = np.array(lamda2[N:])
22      n = np.arange(N)
23      sorty1 = np.array([s1, n]).T
24      sorty2 = np.array([s2, n]).T
25      sorty1 = sorty1[sorty1[:, 0].argsort()]
26      sorty2 = sorty2[sorty2[:, 0].argsort()]
27      l1 = sorty1.T[1].astype(np.uint8)
28      l2 = sorty2.T[1].astype(np.uint8)
29
30      H = np.linalg.hadamard(2**(np.round(np.log2(N)).astype(int)))
31      Phi1 = H[l1[:M], :N]
32      Phi2 = H[l2[:M], :N]
33
34      Psi = fft.dct(np.identity(N))
35      beta1 = Phi1.dot(Psi.T.dot(X))
36      beta = Psi.T.dot(X.T.dot(Psi))
37      beta2 = Psi.T.dot(beta1.T)
38      Y = Phi2.dot(beta.dot(Phi1.T))
39      return (Y, compression)
40
41  def decrypt(signal_in, decompress, key1, key2, key3):
42      M = len(signal_in)
43      N = int(M/decompress)
44      lamda1 = [key1]
45      lamda2 = [key2]
46      for i in range(1, 2*N):
47          lamda1.append(key3*lamda1[i-1]*(1 - lamda1[i-1]))
48          lamda2.append(key3*lamda2[i-1]*(1 - lamda2[i-1]))
49
50      s1 = np.array(lamda1[N:])
```

```
    s2 = np.array(lamda2[N:])
    n = np.arange(N)
    sorty1 = np.array([s1, n]).T
    sorty2 = np.array([s2, n]).T
    sorty1 = sorty1[sorty1[:, 0].argsort()]
    sorty2 = sorty2[sorty2[:, 0].argsort()]
    l1 = sorty1.T[1].astype(np.uint8)
    l2 = sorty2.T[1].astype(np.uint8)

    H = np.linalg.hadamard(2**(np.round(np.log2(N)).astype(int)))
    Phi1 = H[l1[:M], :N]
    Phi2 = H[l2[:M], :N]

    y1 = SL0(Phi2, signal_in)
    y2 = SL0(Phi1, y1.T)
    Y = idct2(ydir2)
    return (Y, decompress)
```

Listing A.4: Code for reading from an audio file and performing CS.

```
import numpy as np
import numpy.random as rand
import scipy.fftpack as fft
import scipy.io.wavfile as wav
import IPython.display as disp

class compsenseFromFile:
    def __init__(self, filename, downsample=False, downrate=None):
        self.rate, self.data = wav.read(filename)
        self.filename = filename
        self.name = filename[:-4]
        if len(self.data.shape) > 1 and self.data.shape[1] > 1:
            self.data = self.data.mean(axis=1)
        self.N = len(self.data)
        self.dur = self.N/self.rate
        self.t = np.linspace(0, self.dur, self.N)
        self.coef = fft.fft(self.data)
        self.coefshift = fft.fftshift(self.coef)
        if downsample:
            self.downrate = downrate
            self.Nd = int(downrate * self.dur)
            nd = np.round(np.linspace(0, self.N-1, self.Nd)).astype(int)
            self.data = self.data[nd]
            self.t = self.t[nd]
            self.coef = fft.fft(self.data)
            self.coefshift = fft.fftshift(self.coef)
            self.rate = downrate
        else:
            self.Nd = self.N
```

```
30              self.t = np.linspace(0, self.dur, self.Nd)
31
32      def getDominantFrequency(coef, rate):
33          return np.argmax(abs(coef))/len(coef)*rate
34
35      def displayOriginal(self, save=False):
36          fig = mp.figure(figsize=(5*16/9*2, 5))
37
38          ax = fig.add_subplot(121)
39          ax.plot(self.t, self.data, lw=0.4)
40          ax.set_xlabel("time␣s")
41          ax.set_ylabel("amplitude")
42          ax.set_title("original␣signal,␣$f_s␣=␣%i$Sa/s"%(self.rate))
43
44          f = np.linspace(0, self.rate, self.Nd)
45          ax = fig.add_subplot(122)
46          ax.plot(f[:self.Nd//2], abs(self.coef)[:self.Nd//2])
47          ax.set_xlabel("frequency,␣Hz")
48          ax.set_ylabel("density")
49          ax.set_title("Frequency␣spectrum")
50
51          if save:
52              mp.savefig(self.name + "_original.png", dpi=300, bbox_inches="tigh
53
54          mp.show()
55          disp.Audio(self.data, rate=self.rate)
56
57      def sampleCompressive(self, mode, rate):
58          self.subrate = rate
59          self.M = int(self.subrate*self.dur)
60
61          if mode == "random":
62              m = np.sort(rd.randint(0, self.Nd, self.M))
63              elif mode == "subnyquist":
64              m = np.round(np.linspace(0, self.Nd-1, self.M)).astype(int)
65          else:
66              raise ValueError("Specified␣mode␣is␣invalid")
67
68          self.y = self.data[m]
69          self.tm = self.t[m]
70          self.compressedCoef = fft.fft(self.y)
71          self.compressedCoefShift = fft.fftshift(self.compressedCoef)
72          self.m = m
73
74      def displayCompressed(self, save=False):
75          y = self.to_int16(self.y)
76
77          fig = mp.figure(figsize=(5*16/9*2, 5))
78
```

```
79          ax = fig.add_subplot(121)
80          ax.plot(self.tm, y, lw=0.4)
81          ax.set_xlabel("time␣s")
82          ax.set_ylabel("amplitude")
83          ax.set_title("compressively␣sampled␣signal")
84
85          f = np.linspace(0, self.subrate, self.M)
86          ax = fig.add_subplot(122)
87          ax.plot(f[:self.M//2], abs(self.compressedCoef)[:self.M//2])
88          ax.set_xlabel("frequency␣Hz")
89          ax.set_ylabel("density")
90          ax.set_title("Frequency␣spectrum")
91
92          if save:
93              mp.savefig(self.name + "_comp.png", dpi=300, bbox_inches="tight")
94              wav.write(self.name + "_comp.wav", self.subrate, y)
95
96          mp.show()
97          disp.Audio(y, rate=self.subrate)
98
99      def recovery(self, method, **method_kwargs):
100         self.method = method
101         d = fft.dct(np.identity(self.Nd))
102         A = d[self.m]
103
104         if method == "lasso":
105             prob = skl.Lasso(**method_kwargs)
106         elif method == "lassocv":
107             prob = skl.LassoCV(**method_kwargs)
108         elif method == "omp":
109             prob = skl.OrthogonalMatchingPursuit(**method_kwargs)
110         elif method == "sl0":
111             prob = SL0(A, self.y)
112             self.recoveredCoef = prob
113             return
114         else:
115             raise ValueError("Specified␣method␣is␣invalid")
116
117         prob.fit(A, self.y)
118         self.recoveredCoef = prob.coef_
119
120     def displayRecovered(self, save):
121         xhat = fft.idct(self.recoveredCoef)
122         xhat = self.to_int16(xhat)
123         fig = mp.figure(figsize=(5*16/9*2, 5))
124
125         ax = fig.add_subplot(121)
126         ax.plot(self.t, xhat, lw=0.4)
127         ax.set_xlabel("time␣s")
```

```
        ax.set_ylabel("amplitude")
        ax.set_title("reconstructed signal")

        f = np.linspace(0, self.rate/2, self.Nd)
        ax = fig.add_subplot(122)
        ax.plot(f[:self.Nd//2], abs(self.recoveredCoef)[:self.Nd//2])
        ax.set_xlabel("frequency, Hz")
        ax.set_ylabel("density")
        ax.set_title("Frequency spectrum")

        if save:
            mp.savefig(self.name + "_recon_" + self.method + ".png", dpi=300,
            wav.write(self.name + "_recon_" + self.method + ".wav", self.rate,

        mp.show()
        disp.Audio(self.y, rate=self.rate)
        self.xhat = xhat

    def to_int16(self, sig):
        return np.round(sig/abs(sig).max() * ((2**16 - 1)//2)).astype("int16")

    def get_psnr(self):
        return 20*np.log10((2**16/2 - 1)/np.sqrt(mse(self.data, self.xhat)))

    def run_all(self, mode, rate, save, method, **method_kwargs):
        self.displayOriginal(save)
        self.sampleCompressive(mode, rate)
        self.displayCompressed(save)
        self.recovery(method, **method_kwargs)
        self.displayRecovered(save)
```

Listing A.5: Code for parsing speech from multiple sources and performing CS.

```
class SpeechCS:
    def __init__(self, filename, rate=None, downsample=False, downrate=Non
        source_type = type(filename)
        if source_type == str:
            if filename.endswith(".wav"):
                self.data, self.rate = sf.read(filename)
            elif filename.endswith(".csv"):
                self.data = np.genfromtxt(filename, delimiter=",")
                if rate is not None:
                    self.rate = rate
                else:
                    self.rate = int(input("Enter sample rate: "))
            else:
                return "Unsupported file type"
        elif source_type == np.ndarray:
            self.data = filename
```

```
17                  if rate is not None:
18                      self.rate = rate
19                  else:
20                      self.rate = int(input("Enter␣sample␣rate:␣"))
21
22              self.dur = self.data.size/self.rate
23              self.name = filename[:-4]
24              self.filename = filename
25
26              if len(self.data.shape) > 1 and self.data.shape[1] > 1:
27                  self.data = self.data.mean(axis=1)
28
29              if downsample:
30                  self.rate = downrate
31                  downsize = int(self.dur * downrate)
32                  down_idx = np.round(np.linspace(0, self.data.size-1, downsize)
33                  self.data = self.data[down_idx]
34
35              self.t = np.linspace(0, self.dur, self.data.size)
36
37          def displayData(self, seglen, pc_overlap, save=False, savename=None, *
38              fig = plt.figure(figsize=(5*16/9, 5*2))
39
40              ax = fig.add_subplot(211)
41              ax.plot(self.t, self.data, lw=0.75, **plot_kwargs)
42              ax.set_ylabel("amplitude")
43
44              ax = fig.add_subplot(212, sharex=ax)
45              f, tx, Zxx = sig.stft(self.data, self.rate, nperseg=seglen, noverl
46              powspecdens = np.zeros_like(Zxx, float)
47              powspecdens[0] = abs(Zxx[0])**2
48              for w in range(1, len(f)):
49                  powspecdens[w] = (abs(Zxx[w])**2)
50              powspecdens = 10*np.log10(powspecdens)
51              ax.pcolormesh(tx, f, powspecdens, cmap="gray_r", shading="flat", z
52              ax.set_ylim(f[1], f[-1])
53              ax.set_ylabel("frequency,␣Hz")
54              ax.set_xlabel("time,␣s")
55
56              plt.tight_layout()
57              if save:
58                  plt.savefig(savename, dpi=300, bbox_inches='tight')
59              plt.show()
60              disp.Audio(self.data, rate=self.rate)
61
62          def sampleCompressive(self, comp_ratio, seglen, percent_overlap, windo
63              hop_size = int(seglen * percent_overlap)
64              starts = np.arange(0, self.data.size, seglen - hop_size, dtype=int
65              starts = starts[starts + seglen < self.data.size]
```

```
66              w = sig.get_window(window, seglen + 1)[:-1]
67              comp_size = int(seglen * comp_ratio)
68              comp_rate = int(self.rate * comp_ratio)
69              xhat = np.zeros_like(self.data, complex)
70              wsum = np.zeros_like(xhat)
71              rand_idx = np.zeros((len(starts), comp_size), int)
72              for i in range(len(rand_idx)):
73                  rand_idx[i] = rd.choice(seglen, size=comp_size, replace=False)
74              spars_basis = fft.dct(np.identity(seglen))
75              for i,n in enumerate(starts):
76                  x = w * self.data[n : n + seglen]
77                  y = x[rand_idx[i]]
78                  A = spars_basis[rand_idx[i]]
79
80                  if i == 0:
81                      prob = skl.LassoCV(cv=10, random_state=0, n_jobs=3)
82                      prob.fit(A, y)
83                      alpha = prob.alpha_
84
85                  prob = skl.Lasso(alpha=alpha)
86                  with warnings.catch_warnings():
87                      warnings.simplefilter("ignore")
88                      prob.fit(A, y)
89
90                  xhat[n : n + seglen] += w * fft.idct(prob.coef_)
91                  wsum[n : n + seglen] += w**2
92
93              self.seglen = seglen
94              self.comp_ratio = comp_ratio
95              self.percent_overlap = percent_overlap
96              self.hop_size = hop_size
97              self.starts = starts
98              self.w = w
99              self.xhat = xhat
100
101         def displayRecovered(self, seglen, pc_overlap, save=False, savename=No
102             fig = plt.figure(figsize=(5*16/9, 5*2))
103
104             ax = fig.add_subplot(211)
105             ax.plot(self.t, self.xhat.real, lw=0.75, **plot_kwargs)
106             ax.set_ylabel("amplitude")
107
108             ax = fig.add_subplot(212, sharex=ax)
109             f, tx, Zxx = sig.stft(self.xhat.real, self.rate, nperseg=seglen, n
110             powspecdens = np.zeros_like(Zxx, float)
111             powspecdens[0] = abs(Zxx[0])**2
112             for w in range(1, len(f)):
113                 powspecdens[w] = (abs(Zxx[w])**2)
114             powspecdens = 10*np.log10(powspecdens)
```

```
115             ax.pcolormesh(tx, f, powspecdens, cmap="gray_r", shading="flat", z
116             ax.set_ylim(f[1], f[-1])
117             ax.set_ylabel("frequency␣Hz")
118             ax.set_xlabel("time␣s")
119
120             plt.tight_layout()
121             if save:
122                 plt.savefig(savename, dpi=300, bbox_inches='tight')
123             plt.show()
124             disp.Audio(self.xhat.real, rate=self.rate)
```