ZJU-UIUC INSTITUTE
Zhejiang University-University of Illinois at Urbana-Champaign Institute
浙江大学伊利诺伊大学厄巴纳香槟校区联合学院

ECE 459

COMMUNICATIONS SYSTEMS

Final Project (Fall 2023)

PERFORMANCE ANALYSIS OF AMPLITUDE AND FREQUENCY MODULATION IN NOISE

Group 5

Name	Student ID	Grade
Yiqin LI	3200112322	
Peidong YANG	3200115555	
Zhuohao XU	3200115233	
Rongjian CHEN	3200110745	
Tiantian ZHONG	3200110643	

Instructor

Prof. Said MIKKI and Prof. Juan ALVAREZ

January 10, 2024

Abstract

This paper reports the methodology and results of the simulation project. The team simulated amplitude modulation and frequency modulation communication systems, plotted the signals and spectrums and computed the signal-noise ratio. The simulation verifies the functionalities of both the communications systems and compares the performance of the performance of the systems in noise.

Keywords Amplitude Modulation, Frequency Modulation, Noise Analysis

Contents

1 Introduction

1.1 The Background

The history of modulation techniques dates back to the early days of radio communication when Amplitude Modulation emerged as the pioneering method. Over time, Frequency Modulation gained prominence due to its resilience against noise and superior audio quality, particularly in broadcasting and mobile communication [1, p. 152].

Amplitude Modulation (AM) is a communication technique that transmits messages by modulating the amplitude of a radio frequency (RF) wave. This modulation is achieved through the combination of the message signal with a high-frequency carrier wave. The resulting modulated waves can be demodulated using either coherent detectors or envelope detectors.

Frequency Modulation (FM), classified as a form of Angle Modulation, involves integrating the message into the phase of an RF signal. Demodulation of the FM signal can be accomplished through the utilization of differentiators or slope circuits.

Both AM and FM present distinct advantages and trade-offs, necessitating a comprehensive performance analysis. Such an analysis is crucial for a thorough understanding of their strengths and limitations within contemporary communication systems.

1.2 Objectives and Purposes

This project is designed to conduct a comprehensive analysis of the performance of FM and AM communication systems in the presence of noise. The methodology involves constructing an envelope-modulated AM and narrow-band FM communication system, with the subsequent application of Additive White Gaussian Noise (AWGN) to the system. Specifically, the demodulation of AM signals is to be carried out using envelope detectors.

The team is tasked with simulating the modulation and demodulation processes for both systems and subsequently comparing the spectra and waveforms of the message signals and demodulated signals. The chosen message signals encompass both a multi-tone message and a Text-to-Speech (TTS)-generated voice recording. The pre- and post-detection signal-to-noise ratios (SNRs) are to be obtained to facilitate a comprehensive performance analysis.

Furthermore, the team is required to meticulously observe disparities between input and output signals, as well as their spectra, in order to conclude the distinctive characteristics of AM and FM modulation. A comparative analysis of anti-noise performance is also imperative, involving the measurement of the signal-to-noise ratio (SNR). The evaluation of simulation performance itself is to be conducted by comparing theoretical and experimental pre- and post-detection SNRs.

This project aims to equip the team with an in-depth understanding of both the modulation and demodulation processes, enabling a nuanced comprehension of the intricacies involved in implementing communication systems using Python. Specifically, the team is expected to demonstrate proficiency in performing Fourier Transform and Hilbert Transform, as well as implementing filters and envelope detectors.

1.3 Literature Review

The textbook by Haykin and Moher [1, Sec. 3.1] presents an intuitive method of envelope modulation. In this approach, the modulated signal is derived by combining a carrier wave with an amplified

and DC-shifted message signal. For the demodulation process, the team references Haykin's work, particularly [1, Fig. 9.8]. Though Ulrich [2] introduces an alternative method of envelope detection utilizing Hilbert Transformation, this non-causal operation is not implementable in analog circuits. A more common implementation in real circuit design is to apply a diode cascaded by an RC filter.

This project applies the Direct Method of FM signal generation as illustrated in [1, Fig. 4.7], which involves an integrator, a controllable oscillator and filters. [1, Fig. 9.13] also elucidated the demodulation process that could be applied to the project.

The measurement of pre- and post-detection SNRs are mentioned in [1, Sec. 9.5 & 9.7], where the theoretical SNRs are also give. This allows the team to evaluate both the performance of the channels and the quality of the simulation.

Ideal filters are not physically implementable as they are non-causal [3, p. 428]. However, the Butterworth filter offers a practical solution for simulating real-world filters, as discussed in the works of Storr [4] and Khetarpal et al. [5]. Implementation of the Butterworth filter can be achieved using the `scipy.signal` package [7], [8].

In simulating an analog process with a digital tool, sampling is critical. [6, pp. 296–297] mentioned Nyquist theorem which ensures no aliasing effects, and the theorem is an important guidance for choosing sampling frequency. The Butterworth filter functions also requires the cut-off frequencies to be less than the Nyquist frequency [7], [8].

2 Methodology

This chapter presents the methodology and the relevant theories employed in the project. The simulations were executed within the Jupyter Notebook environment, leveraging essential packages for numerical computation, signal analysis, and plotting, namely `numpy`, `scipy`, and `matplotlib`.

2.1 Determination of Signal Bandwidth

It is of significance to measure the bandwidth of message signals. As is stated in [3, Sec. 7.2.3] and illustrated in Figure 2.1, the bandwidth B_W of a band-limited signal is the minimum frequency component that the signal has almost zero magnitude at any $f > B_W$, which means

$$S(f) = \begin{cases} \not\approx 0, & \text{for } |f| = B_W \\ \approx 0, & \text{for any } |f| > B_W \end{cases}. \tag{2.1}$$

In the experiment, the bandwidth of a band-limited signal can be determined by visually observing the spectrum or by calculating the maximum frequency with non-zero frequency response.

2.2 Additive White Gaussian Noise

Additive White Gaussian Noise (AWGN) refers to a Gaussian process that can be directly added to a signal to approximate a noise-distorted signal in a real-life communication channel.

A Gaussian process denoted as $\mathrm{N}\left[\mu, \sigma^2\right]$ has the probability density function as

$$f_{Gaussian}(u) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(u-v)^2}{2\sigma^2}\right]. \tag{2.2}$$

It has mean μ and standard deviation σ^2.

According to [1, Sec. 8.10], a noise process is called white noise if it has zero-mean and its power spectral density satisfies

$$S_W(f) = \frac{N_0}{2}. \tag{2.3}$$

The power spectrum of a Python-generated white Gaussian noise process is shown in Figure 2.2.

In this project, the AWGN process will be simulated using `numpy.random.normal()` function. Figure 2.3 shows a Python-generated Gaussian noise process and its frequency spectrum.

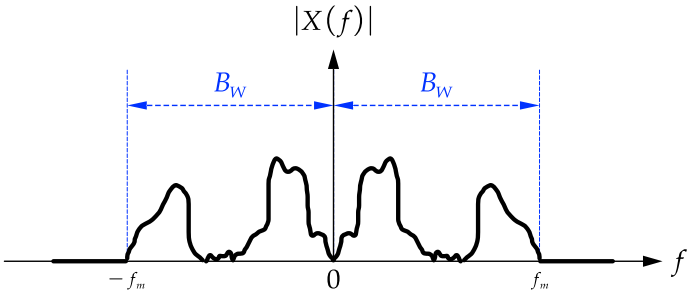

Figure 2.1 The bandwidth of the signal is B_W.

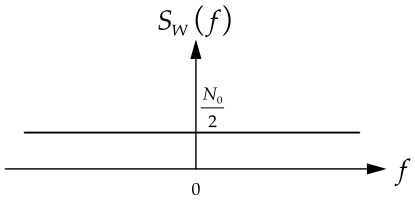

Figure 2.2 The power spectrum of a white noise process.

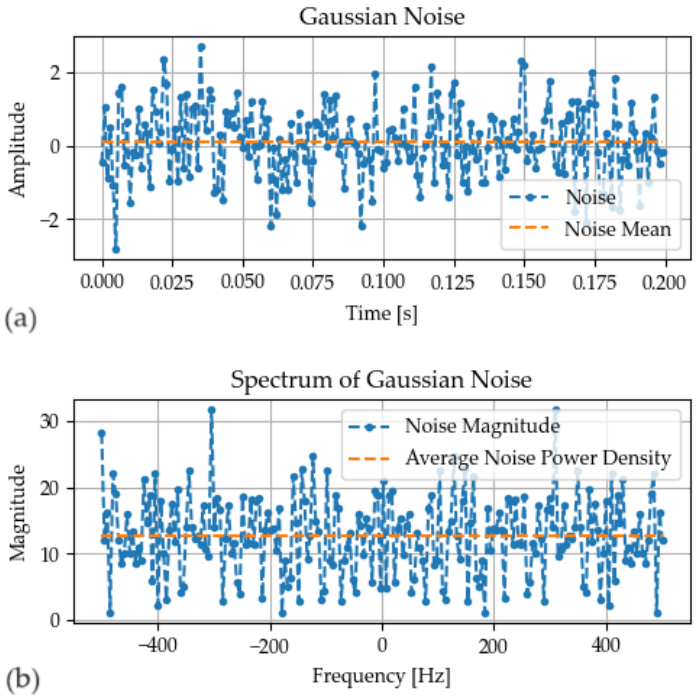

Figure 2.3 The Python-generated Gaussian process N $\left[\mu = 0, \sigma^2 = 1\right]$ and its spectrum.

2.3 AM Simulation

2.3.1 Envelope Modulation

Consider a message signal $m(t)$ and carrier wave $A_c \cos\left(2\pi f_c t + \phi\right)$ where ϕ is the phase delay of the local oscillator. In this project, we pick $\phi = 0$ for simplicity. The modulation of a message signal, denoted as $m(t)$, can be achieved through envelope modulation, represented by the equation:

$$s(t) = A_c \left[1 + k_a m(t)\right] \cos\left(2\pi f_c t + \phi\right) \tag{2.4}$$

In this expression, k_a denotes the modulation sensitivity, A_c corresponds to the amplitude of the carrier wave, and f_c represents the frequency of the carrier wave. It is imperative to ensure that the carrier wave frequency f_c significantly surpasses the highest frequency component, denoted as W, of the message signal $m(t)$ to prevent aliasing. This condition can be expressed as $f_c \gg W$. Moreover, the choice of the modulation sensitivity, k_a, needs to adhere to the constraint which is crucial to prevent envelope distortion, as outlined by Haykin and Moher [1, pp. 101–102]:

$$|k_a m(t)| < 1, \quad \text{for all } t. \tag{2.5}$$

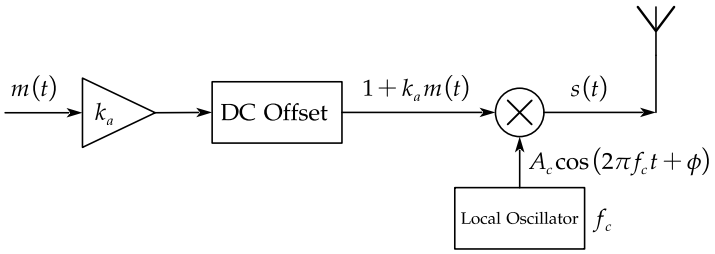

Figure 2.4 Block diagram illustrating the process of envelope modulation.

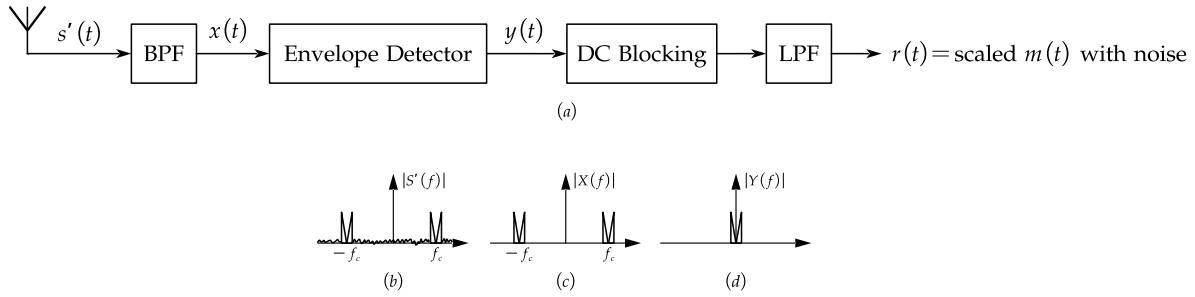

Figure 2.5 The envelope demodulation process. (a) The block diagram of the demodulation process. (b) The spectrum of the received noisy signal $s'(t)$. (c) The spectrum of the band-pass filtered signal $x(t)$. (d) The spectrum of envelope detected signal $y(t)$.

The product of modulation sensitivity k_a and amplitude of message signal A_m

$$\mu = k_a A_m \tag{2.6}$$

is called the modulation factor, or as stated by [9], the modulation index, which can be alternatively expressed as

$$\mu = \frac{A_m}{A_c}. \tag{2.7}$$

In this project, the AM modulation index is specified as $\mu = 0.3$.

The implementation of amplitude modulation can be illustrated through the block diagram depicted in Figure 2.4. The message signal is amplified by a gain k_a and is added a DC offset. The carrier wave is then multiplied with the scaled and shifted message signal, which produces the envelope-modulated wave that is to be transmitted through the channel.

2.3.2 Envelope Detection

Diverging from coherent detection, envelope detection dispenses with the necessity of multiplying the received signal by the carrier wave. Instead, the received signal undergoes filtration by a Band Pass Filter (BPF) to eliminate noise beyond the desired bandwidth, and subsequently, an envelope detector facilitates the recovery of the message signal. This procedural sequence is depicted in Figure 2.5.

A common way to build the envelope detector in Python is to apply Hilbert transform [2], [10]. The envelope of an envelope-modulated signal $s\,(t)$ can be derived by calculating the magnitude of the Hilbert transform of the signal,

$$\text{envelop of } s\,(t) = |\tilde{s}\,(t)| \tag{2.8}$$

where $\tilde{s}(t)$ is the Hilbert transform of $s\,(t)$.

However, as is mentioned in [3, p. 428], the Hilbert Transform is not a causal process, so is not

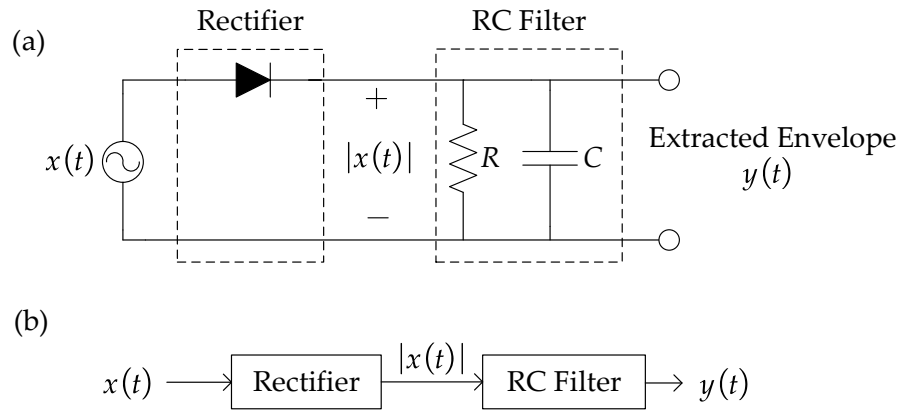

Figure 2.6 The diode detector. (a) The circuit of the typical diode detector. (b) The block diagram of the diode detector.

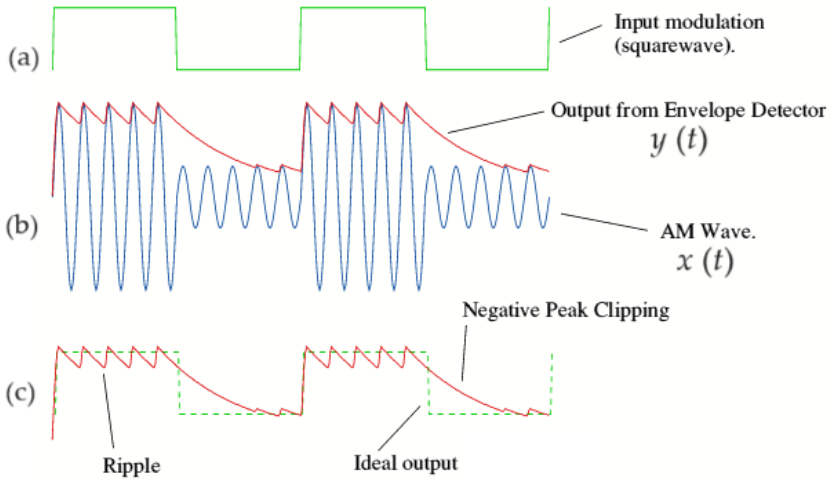

Figure 2.7 Ripple and peak clipping effects [12, Fig. 9.3].

able to be realized using analog circuits. Instead, in physical circuitry, the envelope detector typically comprises a diode and an LPF [11].

Figure 2.6 illustrates a typical diode detector circuit with a full-wave rectifier. As is discussed in [1, pp. 111–112], on the positive segment of the signal, the diode is forward-biased and the capacitor is charged quickly. On the negative segment, the diode becomes reverse-biased and the capacitor discharges slowly through the resistor. When the input signal is greater than the output signal, the capacitor charges again; when the input signal is smaller, the capacitor discharges. In this way, the RC circuit can filter out the high-frequency carrier wave component and approximate the message signal. The time constant of the RC filter, $\tau = RC$, should satisfy

$$RC \ll \frac{1}{f_c}. \tag{2.9}$$

The charging and discharging process is visualized in Figure 2.7 which takes a square wave as an example message signal. The output from the envelope detector is a series of saw-tooth wave that approximates the input square-wave message signal.

2.3.3 SNR Calculation and Measurement

As is stated in Haykin and Moher [1, Eq. (9.26)], the theoretical pre-detection SNR in an envelope modulation receiver can be calculated by

$$\mathrm{SNR}^{\mathrm{AM}}_{\text{pre, theoretical}} = \frac{A_c^2(1+k_a^2)P}{2N_0B_T} \tag{2.10}$$

where A_c is the carrier amplitude, k_a is the modulation sensitivity, P is the power of message signal, N_0 is twice the power spectral density of white noise and B_T is the noise bandwidth of the BPF.

The actual pre-detection SNR can be obtained by measuring the power of the obtained signal and that of noise. Since the filtering process will filter both the useful signal and the noise, the SNR calculation cannot simply apply the specified noise power used in AWGN generation. Instead, there are two ways of measuring the noise in this simulation:

1. Calculating as the power of difference between the resulting signal under an ideal channel and that under a noisy channel.
2. Calculating the power of noise that passes all the filters.

Following the notation in Figure 2.5, the actual pre-detection SNR can be measured as

$$\mathrm{SNR}^{\mathrm{AM}}_{\text{pre, measured}} = \frac{\mathbb{E}[x^2(t)]}{\text{filter noise power}}. \tag{2.11}$$

The "filter noise power" is measured by sending the white noise to the demodulator without any modulated signal and computing its power.

The textbook [1, Eq. (9.23)] states that the theoretical post-detection SNR in an envelope modulation receiver is expressed as

$$\mathrm{SNR}^{\mathrm{AM}}_{\text{post, theoretical}} = \frac{A_c^2k_a^2P}{2N_0W} \tag{2.12}$$

which is a valid approximation only for high SNR and $0 < k_a < 100\%$.

The actual post-detection SNR can be expressed as, if following the notation in Figure 2.5,

$$\mathrm{SNR}^{\mathrm{AM}}_{\text{post, measured}} = \frac{\mathbb{E}\left[r^2(t)\right]}{\text{filter noise power}}. \tag{2.13}$$

Theoretically, according to the calculation in [1, Sec. 9.7], the pre- and post-detection SNR should have a relation similar to Figure 2.8.

2.4 FM Simulation

2.4.1 Narrow-Band FM Modulation

The textbook [1, Sec. 4.1] states that the message signal $m(t)$ will be phase-modulated with a carrier signal $A_c\cos(2\pi f_c t)$, which obtains the frequency modulated wave

$$s(t) = A_c\cos\left[2\pi f_c t + 2\pi k_f\int_0^t m(\tau)\,\mathrm{d}\tau\right] \tag{2.14}$$

where the instantaneous frequency is

$$f_i(t) = f_c + k_f m(t). \tag{2.15}$$

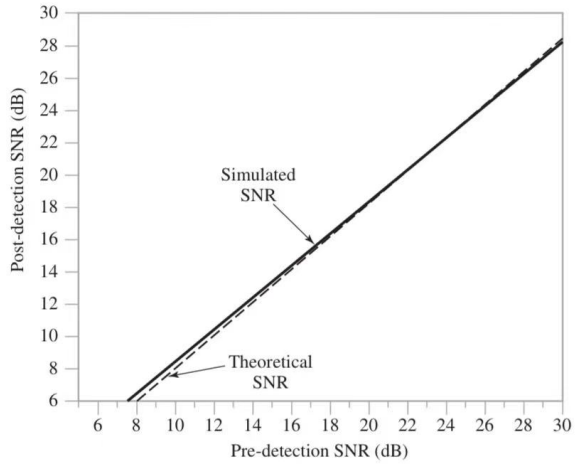

Figure 2.8 The relation between AM pre- and post-detection SNRs [1, Fig. 9.12].

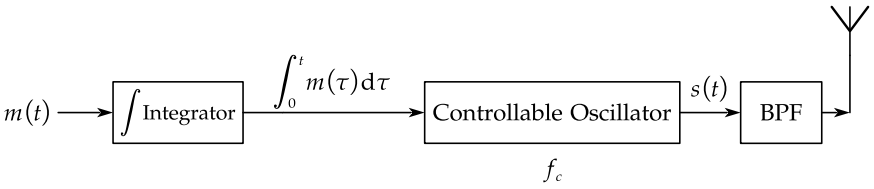

Figure 2.9 The direct method of narrow-band FM modulation.

Here k_f denotes the modulation sensitivity which determines the frequency deviation by

$$\Delta f = k_f A_m. \tag{2.16}$$

By Carson's rule [1, Sec. 4.6], [13], the transmission bandwidth of an FM wave for a frequency-modulated signal is estimated as

$$B_T = 2\Delta f + 2f_{m,\max} \tag{2.17}$$

where $f_{m,\max}$ is the highest modulating frequency. The ratio of Δf and f_m is defined as modulation index,

$$\beta = \frac{\Delta f}{f_m}. \tag{2.18}$$

In this project, the modulation index is specified as $\beta = 0.3$.

Such modulation can be done using a direct method [1, Sec. 4.7], where the frequency modulator contains an oscillator directly controllable by the message signal. Figure 2.9 illustrates this process: the message signal is taken integral and sent to the controllable oscillator to produce a phase-variant signal, which is the modulated wave.

2.4.2 Narrow-Band FM Demodulation

The key to FM demodulation is to extract the phase of the received signal. Considering an ideal channel, the received signal should equal the modulated signal $s(t)$ in Eq. (2.14). Taking the derivative of $s(t)$ yields

$$v(t) = \frac{\mathrm{d}}{\mathrm{d}t} s(t) = -A_c \left[2\pi f_c + 2\pi k_f m(t)\right] \sin\left[2\pi f_c t + 2\pi k_f \int_0^t m(\tau)\,\mathrm{d}\tau\right]. \tag{2.19}$$

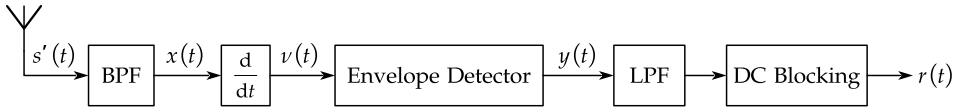

Figure 2.10 The demodulation process of frequency-modulated signals.

An envelope detector could remove the sinusoidal term and get

$$y(t) \approx -A_c \left[2\pi f_c + 2\pi k_f m(t)\right] \tag{2.20}$$

which is an approximation that only contains the scaled version of $m(t)$ and some DC offset. Hence by applying a DC blocking circuit and a proper gain to $y(t)$, the message signal can be concluded.

In practice where the channel is noisy, a BPF will be applied before differentiating and an LPF will be applied before concluding the output, both of which suppress noise introduced by the channel. The complete demodulation process is shown in Figure 2.10.

2.4.3 SNR Calculation and Measurement

As discussed in [1, Sec. 9.7], the theoretical pre- and post-detection SNR is expressed as

$$\mathrm{SNR}^{\mathrm{FM}}_{\text{pre, theoretical}} = \frac{A_c^2}{2N_0 B_T} \tag{2.21}$$

$$\mathrm{SNR}^{\mathrm{FM}}_{\text{post, theoretical}} = \frac{3A_c^2 k_f^2 P}{2N_0 W} \tag{2.22}$$

where P is the power of the message signal, B_T is the noise bandwidth of the BPF, and W is the message bandwidth.

The actual SNRs can be calculated by measuring the power of the signal and noise, which can be concluded as

$$\mathrm{SNR}^{\mathrm{FM}}_{\text{pre, measured}} = \frac{\mathbb{E}\left[x^2(t)\right]}{\text{filter noise power}} \tag{2.23}$$

$$\mathrm{SNR}^{\mathrm{FM}}_{\text{post, measured}} = \frac{\mathbb{E}[r^2(t)]}{\text{filter noise power}}. \tag{2.24}$$

Theoretically, according to the calculation in [1, Sec. 9.8], the pre- and post-detection SNR should have a relation similar to Figure 2.11 where the pre-detection SNR is linear with post-detection SNR for high pre-detection SNR.

2.5 Filter Design

2.5.1 Ideal Filters

The filters remove the unnecessary frequency components of its input signals. Figure 2.12 shows the impulse response of an ideal LPF and BPF.

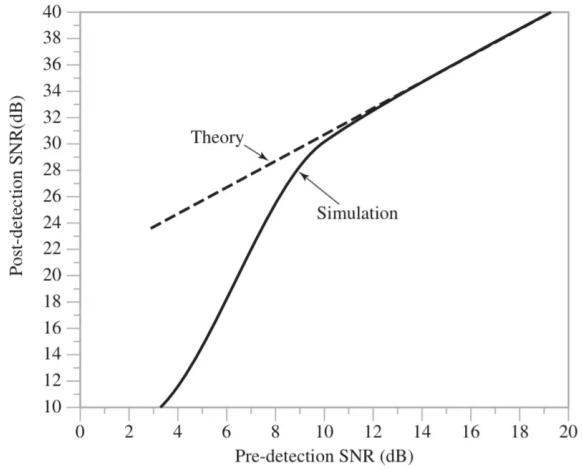

Figure 2.11 The relation between FM pre- and post-detection SNRs [1, Fig. 9.17].

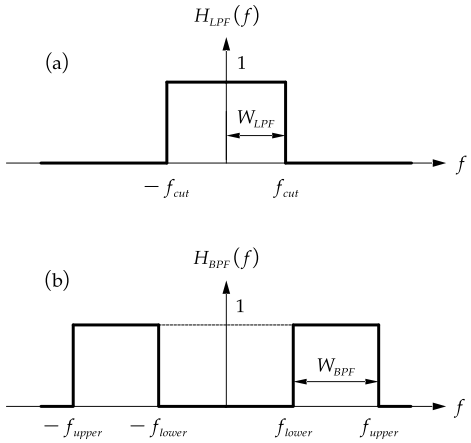

Figure 2.12 The impulse response of ideal LPF and BPF. (a) The impulse response of an ideal LPF. W_{LPF} shows the bandwidth of the LPF. (b) The impulse response of an ideal BPF. W_{BPF} shows the bandwidth of the BPF.

The expressions for ideal LPF and BPF in the frequency domain are

$$H_{LPF}(f) = \text{rect}\left(\frac{f}{2f_{cut}}\right) \tag{2.25}$$

$$H_{BPF}(f) = \text{rect}\left(\frac{f - f_{cut}}{2f_{cut}}\right) + \text{rect}\left(\frac{f + f_{cut}}{2f_{cut}}\right) \tag{2.26}$$

By inverse Fourier transformation, their time-domain impulse response can be obtained as

$$h_{LPF}(t) = 2f_{cut}\text{sinc}(2f_{cut}t) \tag{2.27}$$

$$h_{BPF}(t) = 2f_{cut}\text{sinc}(2f_{cut}t)\,\text{e}^{j2\pi f_{cut}t} + 2f_{cut}\text{sinc}(2f_{cut}t)\,\text{e}^{-j2\pi f_{cut}t} \tag{2.28}$$

which contains non-periodic and infinitely expanding sinc terms. This indicates the impulse response to be associated with the future input, which violates the causality. Thus the ideal LPF and BPF are not causal and are not physically implementable.

2.5.2 Butterworth Filter in Python

Both Amplitude Modulation (AM) and Frequency Modulation (FM) communication systems necessitate the incorporation of filters. In practical scenarios, however, the realization of ideal filters poses inherent challenges. Nevertheless, the Butterworth Filter exhibits characteristics closely approximating those of an ideal filter. Figure 2.13 illustrates an analog circuit design of a first-order Butterworth LPF which is composed of an operational amplifier. According to [14], the general n-th order Butterworth LPF has a transfer function

$$H_n(j\omega) = \frac{1}{\sqrt{1+\epsilon^2\left(\frac{\omega}{\omega_c}\right)^{2n}}} \tag{2.29}$$

where ϵ is the maximum pass-band gain and ω_c is the cut-off frequency. Figures 2.14 and 2.15 depict the frequency responses of Butterworth LPF and BPF with varying orders, illustrating that an increased order (n) results in a more pronounced roll-off slope. As expounded by [3], [4], a Butterworth Low Pass Filter manifests a maximally flat frequency response within its passband, swiftly attenuating beyond the cut-off frequency. This advantageous attribute empowers the design of filters with minimal distortion, albeit at the expense of an indeterminate phase delay induced by the inherent properties of Infinite Impulse Response (IIR) filters.

In the digital realm, Python facilitates the implementation of filters as digital IIR filters, with each filter in the z-domain expressed as the quotient of two polynomials:

$$H_{\text{digital}}(z) = \frac{\sum_{p=0}^{n} a_p z^{-p}}{\sum_{q=0}^{n} b_q z^{-q}}. \tag{2.30}$$

The response of the filter is uniquely determined by coefficients a_i and b_j ($0 \le i, j \le n$), where these coefficients can be computed using the `scipy.signal.butter()` function, as documented by [7], [8], [15].

2.5.3 Nyquist Sampling Theorem

The sampling frequency is of significance in the design of the Butterworth filter. As stated in [6, pp. 296–297], the sampling frequency f_s should satisfy

$$f_s \ge 2B_W \tag{2.31}$$

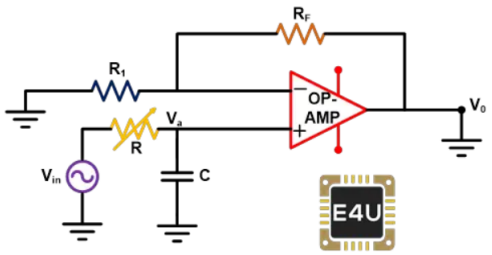

Figure 2.13 A first-order Butterworth LPF circuit [14].

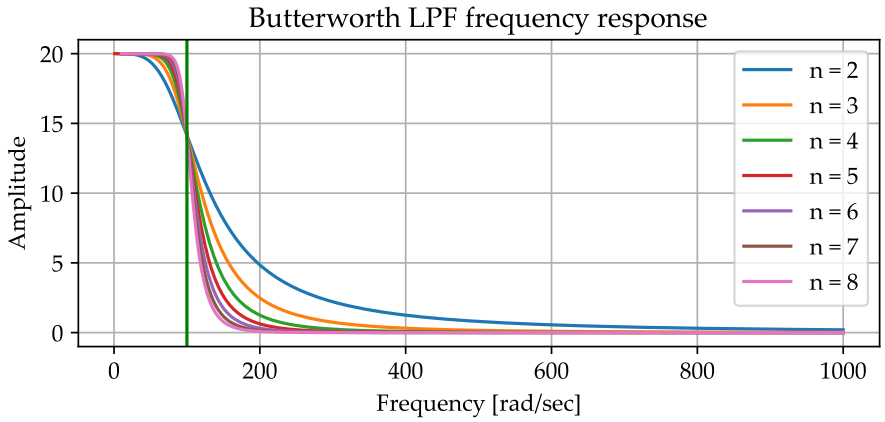

Figure 2.14 The frequency response of a Butterworth LPF with cut-off frequency $\omega_c = 100\,\text{rad/s}$.

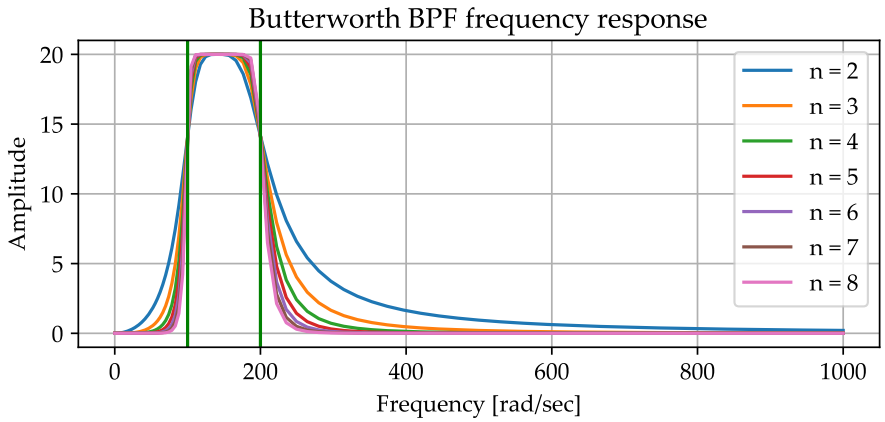

Figure 2.15 The frequency response of a Butterworth BPF with pass-band from 100 Hz to 200 Hz.

where B_W is the bandwidth of the signal to sample. Hence in filter design, the frequency parameters should also follow the sampling theorem,

$$f_{cut,LPF} \leq \frac{f_s}{2} \tag{2.32}$$

$$f_{lower,BPF} < f_{upper,BPF} \leq \frac{f_s}{2}. \tag{2.33}$$

3 Results

3.1 Experiment Settings

3.1.1 The Message Signal

Considering the wide application of AM/FM technologies in radio communication where the data to transmit is mainly human voice, this project applies a TTS-generated monaural male voice recording which samples at 48 kHz and lasts around 3.5 s.

The bandwidth of the TTS-generated recording can be determined by its spectrum illustrated in Figure 3.1 using Python. Note that the recording is generated using the male's voice, which typically ranges from 80 Hz to 200 Hz. It is observed that the majority of power is concentrated within 5 kHz, which will be regarded as the bandwidth of the message signal. To ensure a more precise bandwidth, an LPF is added to remove unnecessary frequency components beyond 5 kHz.

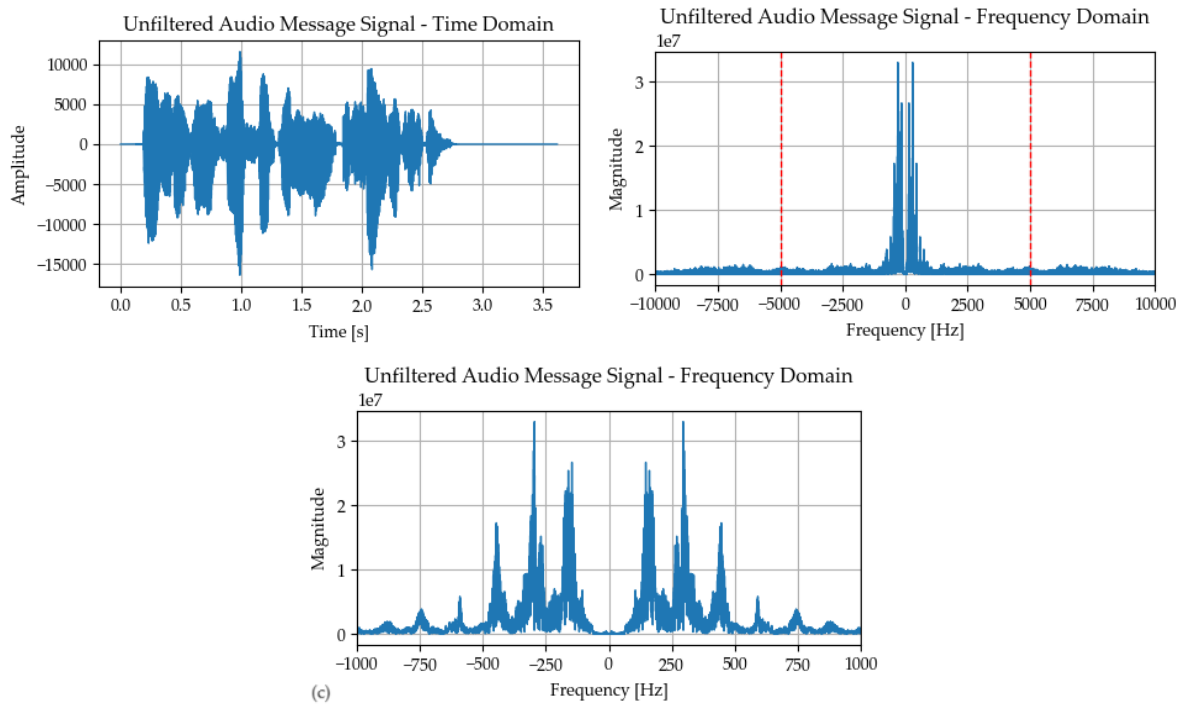

Figure 3.1 The waveforms of the unfiltered audio recording. (a) The signal in time domain. (b) The spectrum of the signal. The majority of the response is concentrated within 1 kHz, but the team chooses 5 kHz for balancing better audio quality and bandwidth saving. (c) The spectrum of the signal within 1 kHz. The major frequency components are among 125 Hz to 500 Hz.

3.1.2 Simulation Parameters

AM Simulation

The parameters for AM simulation are listed as follows:

- Message Bandwidth $f_m = 5\,\text{kHz}$.
- Message Amplitude $A_m = 11604$.
- Sampling Frequency $f_s = 48\,\text{kHz}$.

- Modulation Index $\mu = 0.3$.
- Carrier Wave Frequency $f_c = 12\,\text{kHz}$.
- Carrier Wave Amplitude $A_c = 1$.
- Variance of AWGN $\sigma^2 = 1 \times 10^{-5}$.

Using the parameters above, the following parameters can be calculated:

- Modulation Sensitivity $k_a = 1.83 \times 10^{-5}$.

FM Simulation

The parameters for FM simulation are listed as follows:

- Message Bandwidth $f_m = 5\,\text{kHz}$.
- Message Amplitude $A_m = 11604$.
- Sampling Frequency $f_s = 480\,\text{kHz}$.
- Modulation Index $\beta = 0.3$.
- Carrier Wave Frequency $f_c = 180\,\text{kHz}$.
- Carrier Wave Amplitude $A_c = 1$.
- Variance of AWGN $\sigma^2 = 1 \times 10^{-7}$ ($\sigma^2 = 1 \times 10^{-5}$ is also tested).

Using the parameters above, the following parameters can be determined:

- Maximum Frequency Deviation $\Delta f = \beta f_m = 1.5\,\text{kHz}$.
- Transmission Bandwidth $B_T = 2\,(\Delta f + f_m) = 13\,\text{kHz}$.
- Modulation Sensitivity $k_f = 0.129$.

3.2 AM Simulation

To maximally eliminate the distortion at the beginning and end of the time domain signal brought by the filters, the message signal is padded with zeros at both the beginning and the end. The signal is shown in Figure 3.2.

The zero-padded message signal is then modulated with the carrier wave, and the modulated signal is shown in Figure 3.3. Note that the spectrum of the modulated signal has frequency components around the carrier frequency 12 kHz.

The noise is then added to the signal and produces a noisy FM signal. The signal is illustrated 3.4.

At the receiver side, a BPF is applied to the signal to suppress noise. The spectrum of the filtered signal is shown in Figure 3.5. Then the envelope detector is applied to the signal, which produces the demodulated signal shown in Figure 3.6.

The SNRs are calculated to be:

- Pre-Detection SNR: 46.99 dB.
- Post-Detection SNR: 119.96 dB.

The team also tried different pre- and post-detection SNRs. The visual diagram of the relation between pre- and post-detection SNRs is shown in Figure 3.7. They are almost linear with each other, which matches the theoretical computation in [1, Sec. 9.7]

3.3 FM Simulation

The same audio file is applied as the input to the system, whose time and frequency domain waveforms are shown in Figure 3.8. In order to reach the target sampling frequency 48 kHz, the team first did an upsample by a factor of 10 to the audio by `scipy.signal.resample()`. This allows the team to have better flexibility in designing filters. The upsampling expands the frequency spectrum by the properties of the Fast Fourier Transform (FFT) Algorithm. The spectrum of resampled and filtered signal is shown in Figure 3.8. Note that compared to Figure 3.1(b), the magnitude for each frequency component is multiplied by a factor of 10.

The integrator takes an integral to the message signal, and the controllable oscillator produces the frequency-modulated signal. The waveform and the spectrum of the signal are shown in Figure 3.9 and 3.10. The two main lobes present around the carrier frequency $f_c = 180\,\text{kHz}$.

The noise is the same as the AM simulation, and the noisy frequency-modulated signal is shown in Figure 3.11. After receiving the signal, the signal is band-pass filtered and the signal is shown in Figure 3.12. The pass-band of the filter is designed to be from $f_c - B_T$ to $f_c + B_T$, where B_T is the transmission bandwidth. The filtered signal is then taken differentiation and envelope detection (using ideal envelope detector realised by Hilbert transformation), and the results are shown in Figure 3.13 and Figure 2.6 respectively.

After steps above, we use low pass filter at first, and then minus the DC value bias directly to get the fully recovered signal (see Figure 3.15).

The final result we get for the SNR is like this:

- Pre-Detection SNR: 66.99 dB.
- Post-Detection SNR: 258.64 dB.

As we can see, the post-SNR is obviously much higher than the post one. Because while the lowpass filter does not damage the message signal, the power for the noise is significantly decreased.

In order to compare the anti-noise performance with AM, the team tested the noise variance $\sigma^2 = 1 \times 10^{-5}$. The pre-detection SNR in this case is 46.99 dB, and the post-detection SNR is 239.792 dB. The post-detection SNR is much higher than AM, which indicates FM has better anti-noise performance.

The relationship between pre- and post-detection SNRs is tested and plotted in Figure 3.16.

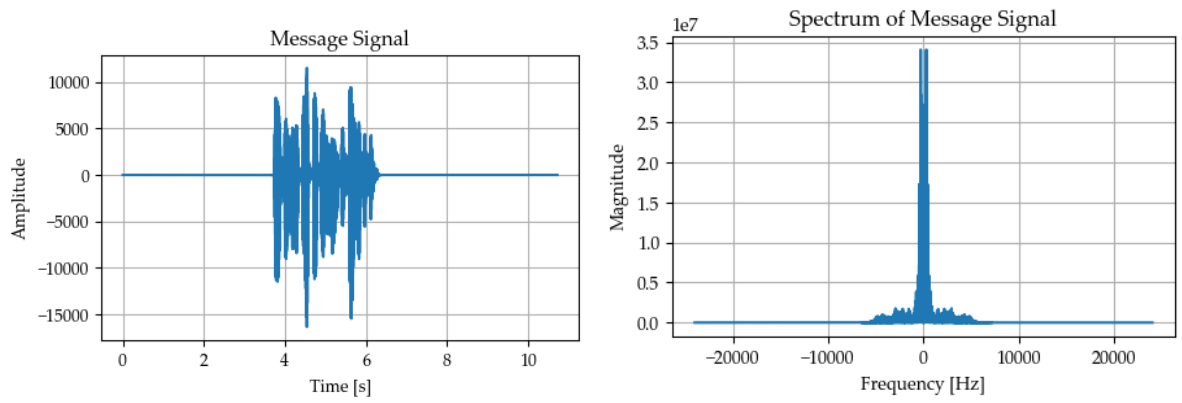

Figure 3.2 The zero-padded message signal. It is triple the length of the original message signal.

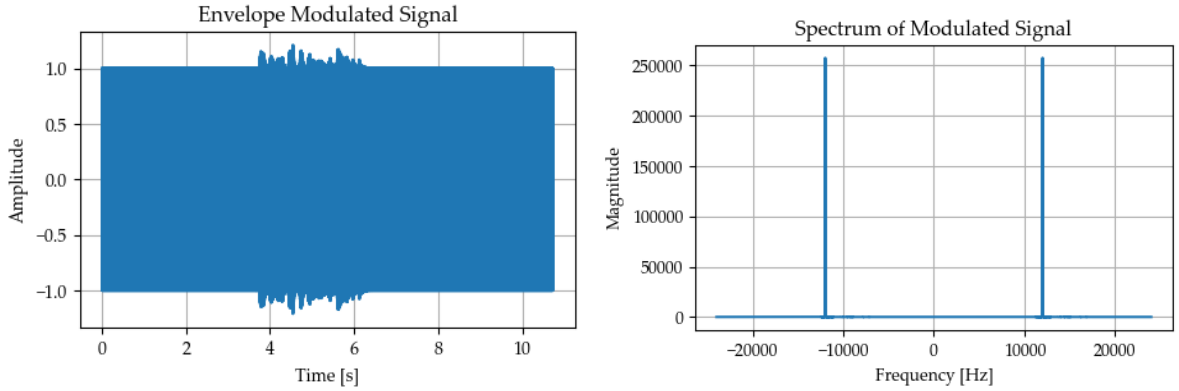

Figure 3.3 The envelope-modulated signal.

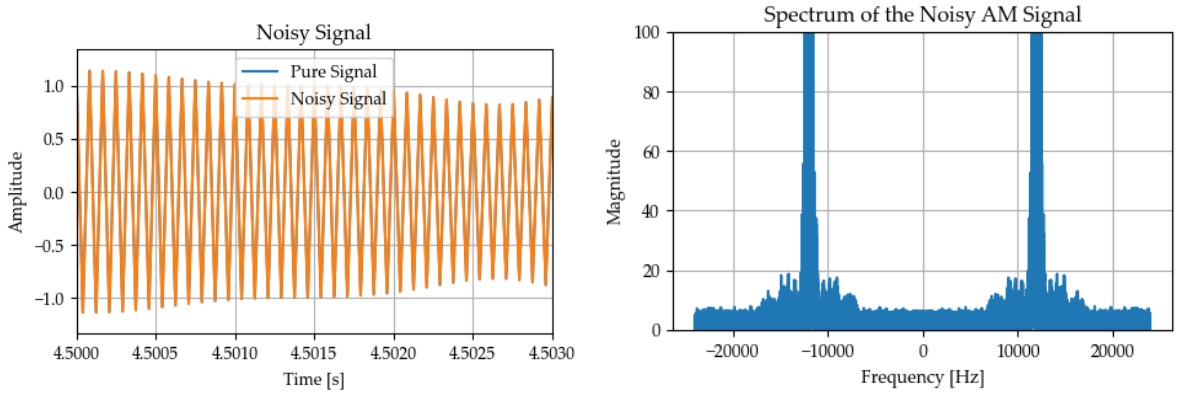

Figure 3.4 The noisy AM signal. Note that the plotting magnitude range of the spectrum is limited in order to show the noise clearly.

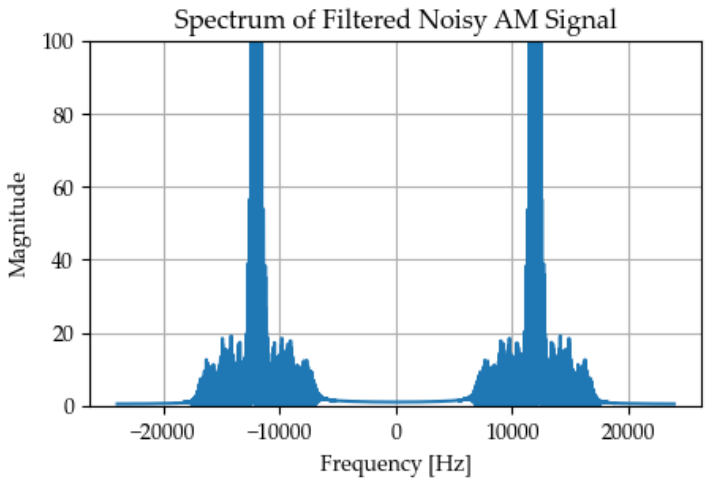

Figure 3.5 The filtered noisy AM signal.

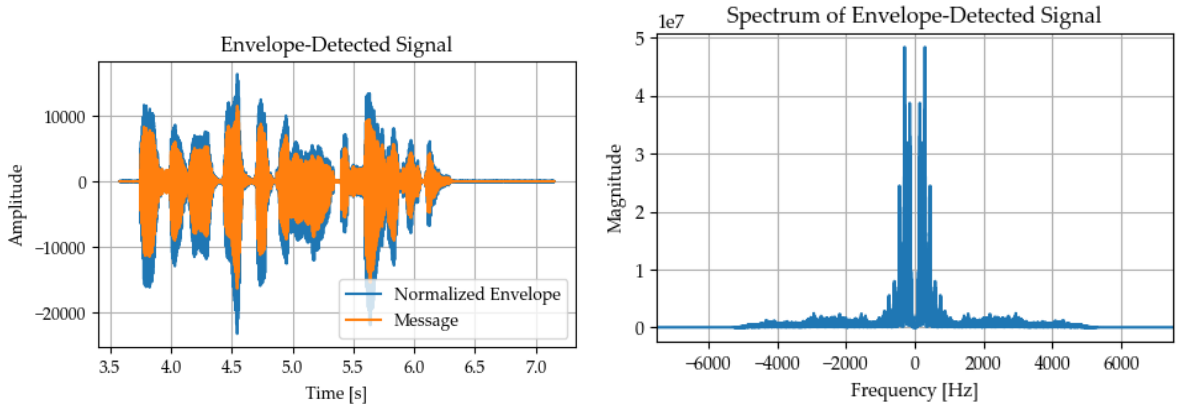

Figure 3.6 The demodulated signal of AM system.

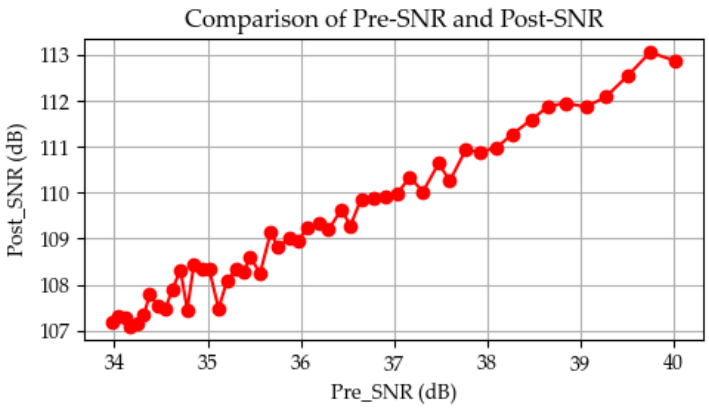

Figure 3.7 The relationship between AM pre- and post-detection SNRs.

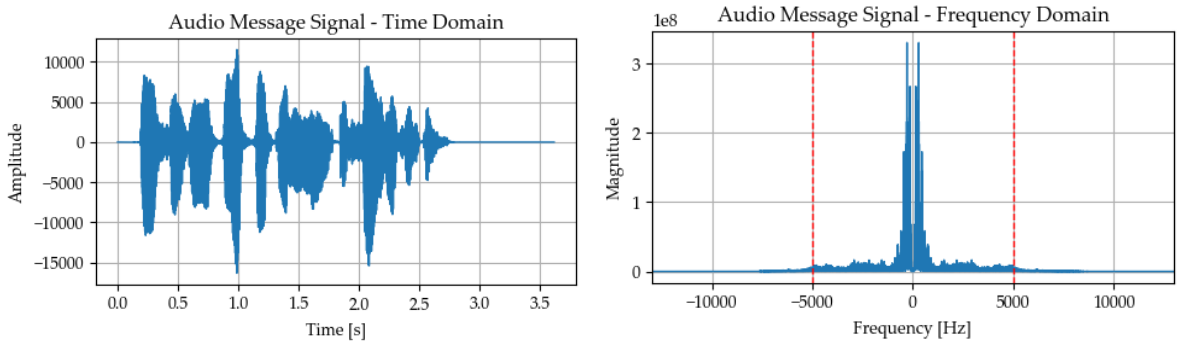

Figure 3.8 The waveforms of the resampled 5 kHz low-pass filtered audio recording. (a) The signal in time domain. (b) The spectrum of the signal.

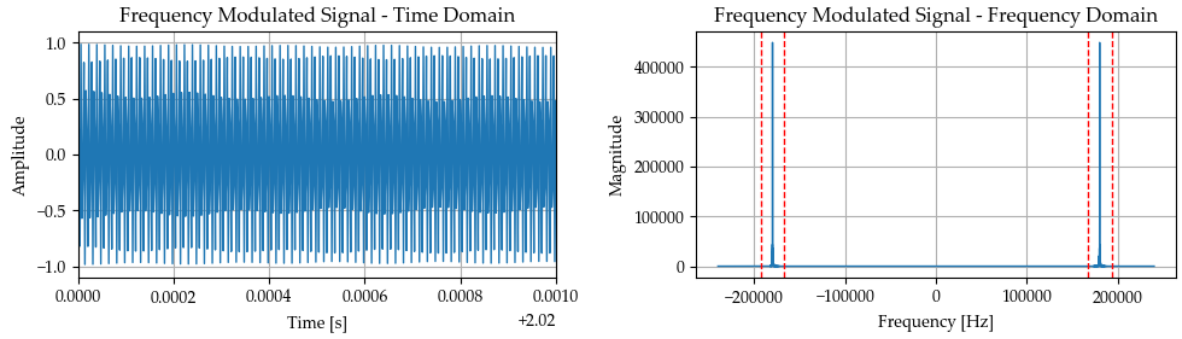

Figure 3.9 The waveform and spectrum of the frequency-modulated signal.

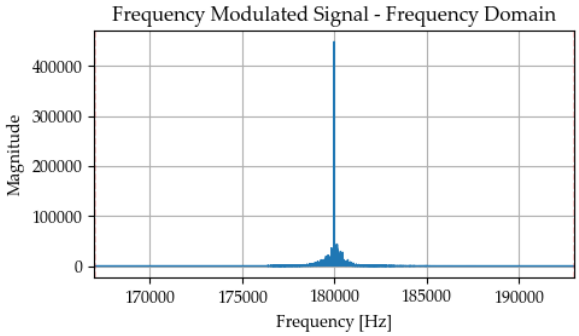

Figure 3.10 The spectrum of the frequency-modulated signal within the transmission bandwidth.

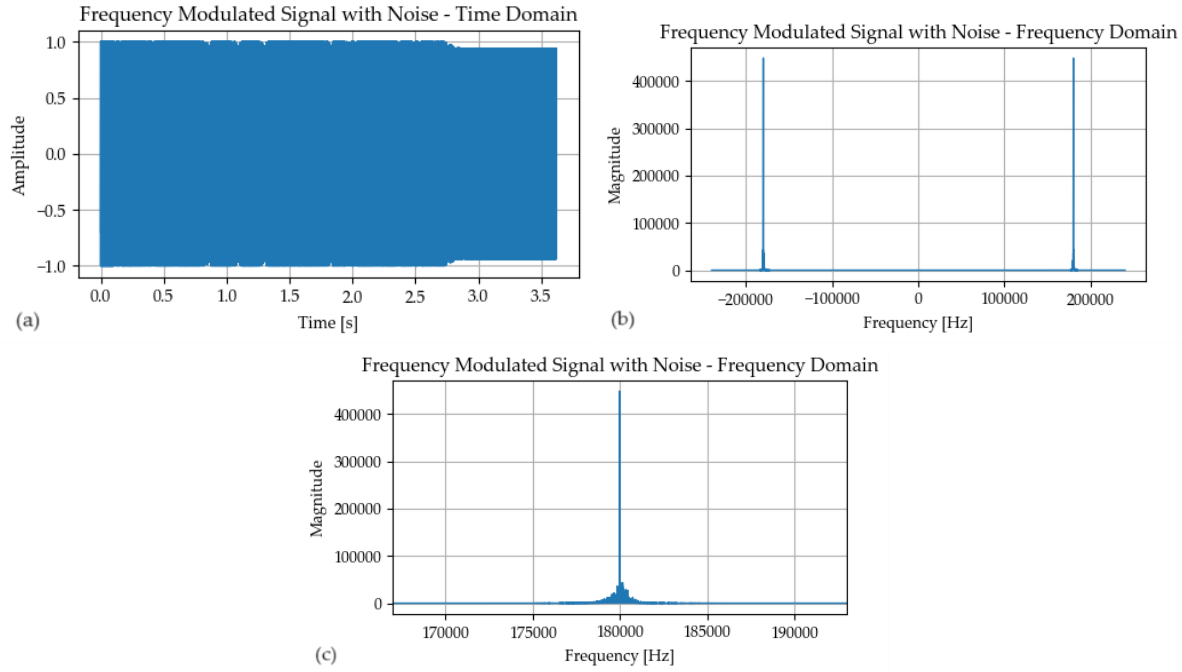

Figure 3.11 The waveforms of the noisy frequency-modulated signal. (a) The time-domain signal. (b) The frequency-domain signal. (c) The frequency-domain signal within the transmission bandwidth. Note that the response is non-zero beyond the transmission bandwidth.

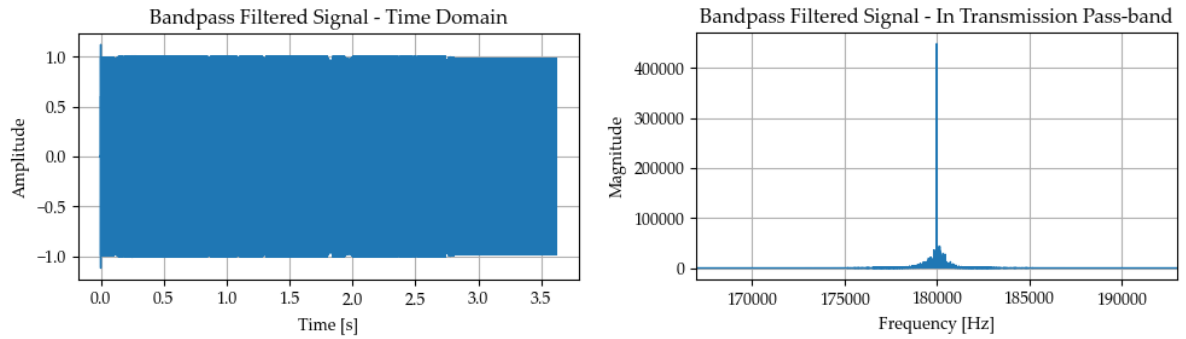

Figure 3.12 The band-pass filtered frequency-modulated signal. Note that some distortion happens at the beginning of the signal, which is brought about by filtering a non-periodic signal using a digital filter.

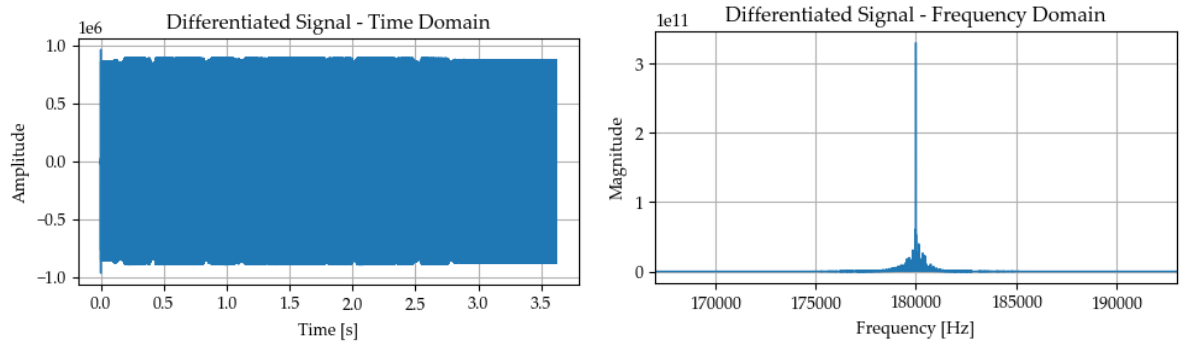

Figure 3.13 The differentiated signal.

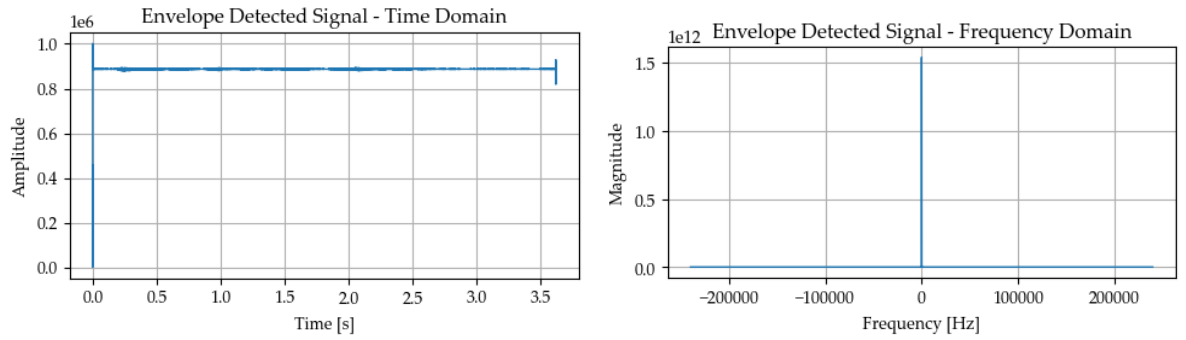

Figure 3.14 The envelope detected differentiated signal. Note that the signal has a non-zero DC offset compared to the message signal.

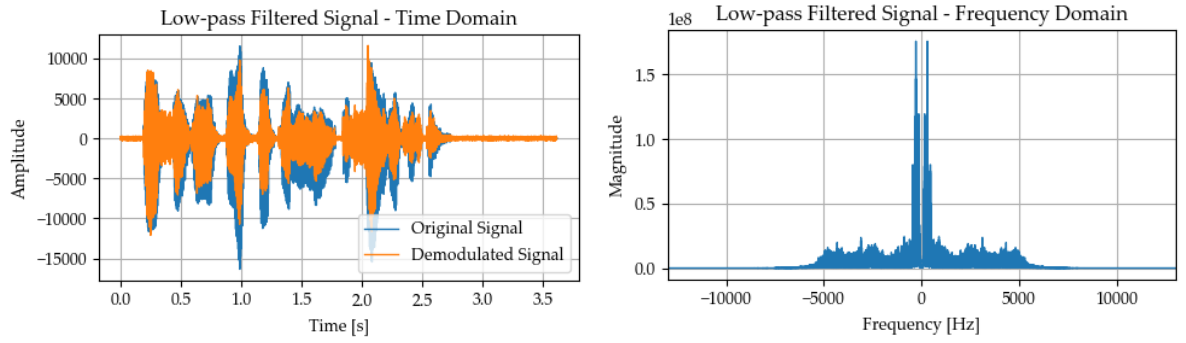

Figure 3.15 The low-pass filtered and DC-blocked signal with maximum amplitude matched with the message signal.

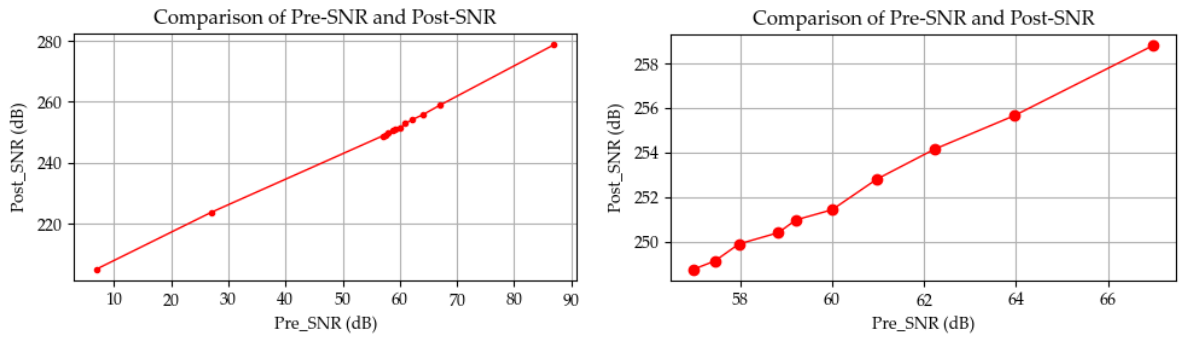

Figure 3.16 The plot of FM pre-detection SNR vs. post-detection SNR.

4 Conclusion

In this project, the team practiced modeling and analyzing data in Python and grabbed some experience in simulating continuous-time systems with digital tools. The team had a deeper understanding of not only analog communication but also digital signal processing, which is of great significance in proceeding into digital communication in career.

Analyzing the results in the previous section, the team confirms the availability of the designed simulation models of the AM and FM communication systems and verified that FM has better frequency-robustness than AM systems.

The quality of the demodulated AM wave sounds better than the FM wave, yet the post-detection SNR for AM is lower than that of FM. This might be due to the distortion brought by the integrator, which does not count into the noise. Compared to AM, the FM system has two extra components, the integrator and

The team observed that by applying filters at proper positions, the noise can be effectively eliminated and the SNR can be significantly increased. The team also noted that filters can bring a significant degree of distortion, as the ideal filters are not implementable. Butterwo rth filter provides a close approximation to ideal filters, which have a flat response within the pass-band and fast rolling-off beyond the cut-off frequency. Eighth-order Butterworth filters are used the most commonly.

The project verifies that pre- and post-SNRs have an almost linear relationship, especially for high pre-detection SNR. This can be concluded from intuition that the filtering process only changes the proportion of noise that passes the systems, hence the ratio of pre- and post-detection SNRs should be almost constant. Theoretical computation in the textbook [1, Sec. 9.6 & 9.7] also shows such approximate linearity.

The project still has some space for improvement. The message signal can be more diversified, from audio signals to analog video signals. The team also came up with an idea about simulating a Slow-Scan Television (SSTV) which is widely used in satellite communication. However, due to the time limitation of the project, those ideas failed to be implemented.

Appendix A Python Scripts of This Project

The scripts of the simulation project, along with the source code of this report which is compiled using LaTeX, is published on GitHub: https://github.com/martinz2002/ece459-project-fa23-zjui.

The Python code for simulation is listed in below sections.

A.1 AM Simulation

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import hilbert, butter, filtfilt
from scipy.fftpack import fft, fftshift
from scipy.io import wavfile

mod_index = 0.3 # modulation index specified by the problem

def calculate_snr(signal, noise):
    signal_power = np.mean(signal ** 2)
    noise_power = np.mean(noise ** 2)
    return 10 * np.log10(signal_power / noise_power)

# 1. Generate Sinusoidal Message Signal
fs, message = wavfile.read('audio.wav')
bw = 5000

t = np.arange(0, len(message)/fs, 1/fs)
plt.figure()
plt.plot(t, message)
plt.title('Original Message Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')


b, a = butter(8, bw, fs=fs, btype='low', analog=False)
message_filtered = filtfilt(b, a, message)
ka = mod_index / np.max(np.abs(message_filtered))

audio_data = np.copy(message_filtered)
audio_data_length = len(message_filtered)
new_message = np.zeros(3*len(message_filtered)) #!! 补长信号，以补偿滤波器的误差
new_message[len(message_filtered):2*len(message_filtered)] = message_filtered
message = new_message
t = np.arange(0, len(message)/fs, 1/fs)
fc = 12000

# print all parameters
print('Sampling rate: {} Hz'.format(fs))
print('Carrier frequency: {} Hz'.format(fc))
```

```
print('Bandwidth: {} Hz'.format(bw))
print('Modulation sensitivity: {}'.format(ka))

# 2. Perform Envelope Modulation
carrier = np.cos(2 * np.pi * fc * t)
message_normalized = new_message
modulated_signal = (1 + ka * message_normalized) * carrier

# 3. Add AWGN Noise
noise_power = 0.0001  # Noise power
noise = np.sqrt(noise_power) * np.random.normal(size=len(t))
noisy_signal = modulated_signal + noise

# 4. Envelope Detection
# bpf to supress noise
b, a = butter(8, [fc - bw, fc + bw], fs=fs, btype='bandpass', analog=False)
noisy_signal_filtered = filtfilt(b, a, noisy_signal)
noisy_signal = np.copy(noisy_signal_filtered)

# use lpf
b, a = butter(8, bw, fs=fs, btype='lowpass', analog=False)
envelope = filtfilt(b, a, np.abs(noisy_signal))
envelope_no_offset = envelope - np.mean(envelope)  # Remove DC offset
envelope_extracted = envelope_no_offset[audio_data_length:2*audio_data_length] # remove padded
    zeros
envelope_extracted_normalized = envelope_extracted / np.max(envelope_extracted) * np.max(np.abs
    (message))  # Normalize

# 5. Plotting
# plot the message signal
plt.figure()
plt.plot(t, message)
plt.title('Message Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')

# plot the spectrum of the message
plt.figure()
f = np.linspace(-fs/2, fs/2, len(t))
message_spectrum = fftshift(fft(message))
plt.plot(f, np.abs(message_spectrum))
plt.title('Spectrum of Message Signal')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Magnitude')

# plot normalized message signal
plt.figure()
plt.plot(t, message_normalized)
plt.title('Normalized Message Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
```

```
90
91  # Plot the envelope modulated signal
92  plt.figure()
93  plt.plot(t, modulated_signal)
94  plt.title('Envelope␣Modulated␣Signal')
95  plt.xlabel('Time␣[s]')
96  plt.ylabel('Amplitude')
97
98  # Plot the spectrum
99  plt.figure()
100 f = np.linspace(-fs/2, fs/2, len(t))
101 modulated_spectrum = fftshift(fft(modulated_signal))
102 plt.plot(f, np.abs(modulated_spectrum))
103 plt.title('Spectrum␣of␣Modulated␣Signal')
104 plt.xlabel('Frequency␣[Hz]')
105 plt.ylabel('Magnitude')
106
107 # Plot the envelope-detected signal after low-pass filtering
108 plt.figure()
109 plt.plot(t[audio_data_length:2*audio_data_length], envelope_extracted_normalized, label='
        Normalized␣Envelope')
110 plt.plot(t[audio_data_length:2*audio_data_length], audio_data, label='Message')
111 plt.title('Envelope-Detected␣Signal')
112 plt.xlabel('Time␣[s]')
113 plt.ylabel('Amplitude')
114 plt.legend()
115 plt.show()
116
117 envelope_extracted_normalized_fft = fftshift(fft(envelope_extracted_normalized))
118 plt.figure()
119 freq = np.linspace(-fs/2, fs/2, len(envelope_extracted_normalized_fft))
120 plt.plot(freq, np.abs(envelope_extracted_normalized_fft))
121 plt.title('Spectrum␣of␣Envelope-Detected␣Signal')
122 plt.xlabel('Frequency␣[Hz]')
123 plt.ylabel('Magnitude')
124 plt.xlim([-1.5*bw, 1.5*bw])
125
126 # write the demodulated signal to a wav file as int16
127 wavfile.write('am_demodulated.wav', fs, envelope_extracted_normalized.astype(np.int16))
128
129 pre_detection_snr = calculate_snr(modulated_signal, noise)
130 post_detection_snr = calculate_snr(envelope_extracted_normalized, noise)
131 pre_detection_snr, post_detection_snr
132
133 print(pre_detection_snr, post_detection_snr)
```

A.2 FM Simuation

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

```
from scipy.signal import butter, lfilter, hilbert
from scipy.fftpack import fft, fftshift
from scipy.io import wavfile
from scipy.signal import resample

plt.rcParams['font.family'] = 'Palatino␣Linotype'
plt.rcParams['legend.loc'] = 'best'
plt.rcParams['axes.grid'] = True
plt.rcParams['figure.figsize'] = [5,3]
plt.rcParams['lines.linewidth'] = 1

# Function for plotting in time and frequency domain
def plot_signal(t, signal, title, Fs):
    # Time domain
    plt.subplot(2, 1, 1)
    plt.plot(t, signal)
    plt.title(title + '␣-␣Time␣Domain')
    plt.xlabel('Time␣[s]')
    plt.ylabel('Amplitude')

    # Frequency domain
    plt.subplot(2, 1, 2)
    f = np.linspace(-Fs/2, Fs/2, len(signal))
    signal_fft = fft(signal)
    signal_fft_shifted = np.fft.fftshift(signal_fft)
    plt.plot(f, np.abs(signal_fft_shifted))
    plt.title(title + '␣-␣Frequency␣Domain')
    plt.xlabel('Frequency␣[Hz]')
    plt.ylabel('Magnitude')
    plt.tight_layout()
    plt.show()

def lowpass_filter(signal, cutoff, sample_rate, order=5):
    nyq = 0.5 * sample_rate
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='low')
    return lfilter(b, a, signal)

def write_wav(filename, data, sample_rate=44100):
    # resample
    data = resample(data, int(len(data)/sample_rate * 44100))
    wavfile.write(filename, 44100, data.astype(np.int16))

# Bandpass filter design
def bandpass_filter(signal, lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return lfilter(b, a, signal)

```

```
54 def get_envelope(signal, message_frequency=5000, method='hilbert'):
55     if method == 'hilbert':
56         analytic_signal = hilbert(signal)
57         envelope = np.abs(analytic_signal)
58         return envelope
59     b, a = butter(8, message_frequency/(Fs/2), btype='low', analog=False)
60     envelope = lfilter(b, a, np.abs(signal))
61     return envelope
62 
63     # Original single tone signal
64 # original_signal = np.cos(2 * np.pi * f_m * t) + 0.7*np.cos(2 * np.pi * 0.6*f_m * t)
65 f_c = 180000  # Carrier frequency 180 kHz
66 beta = 0.3    # Modulation index
67 Fs, original_signal = wavfile.read('audio.wav')
68 # append some zeros to the end and beginning of the signal
69 original_signal = np.concatenate((np.zeros(1000), original_signal, np.zeros(1000)))
70 t = np.arange(0, len(original_signal)/Fs, 1/Fs)
71 write_wav('original.wav', original_signal, Fs)
72 Am = np.max(original_signal) # Amplitude of the message signal
73 fm = 5e3 # Frequency of the message signal
74 kf = beta*fm/Am # Frequency sensitivity
75 bt = 2*kf*Am + 2*fm # TX Bandwidth of the FM signal, by Carson's rule
76 duration = len(original_signal)/Fs
77 
78 original_signal_fft = fftshift(fft(original_signal))
79 f = np.linspace(-Fs/2, Fs/2, len(original_signal))
80 
81 plt.figure()
82 plt.plot(f, np.abs(original_signal_fft))
83 plt.title('Unfiltered␣Audio␣Message␣Signal␣-␣Frequency␣Domain')
84 plt.xlabel('Frequency␣[Hz]')
85 plt.xlim([-1000, 1000])
86 plt.axvline(x=-fm, color='r', linestyle='--') # mark fm
87 plt.axvline(x=fm, color='r', linestyle='--')
88 plt.ylabel('Magnitude')
89 plt.tight_layout()
90 plt.show()
91 
92 plt.figure()
93 plt.plot(t, original_signal)
94 plt.title('Unfiltered␣Audio␣Message␣Signal␣-␣Time␣Domain')
95 plt.xlabel('Time␣[s]')
96 plt.ylabel('Amplitude')
97 plt.tight_layout()
98 plt.show()
99 
100 # manually upsample to 2*Fs
101 upsample_factor = 10
102 original_signal = resample(original_signal, upsample_factor*len(original_signal))
103 Fs *= upsample_factor
104 t = np.arange(0, duration, 1/Fs)
```

```
105
106 original_signal_fft = fftshift(fft(original_signal))
107 f = np.linspace(-Fs/2, Fs/2, len(original_signal))
108
109 plt.figure()
110 plt.plot(f, np.abs(original_signal_fft))
111 plt.title('Resampled Unfiltered Audio Message Signal - Frequency Domain')
112 plt.xlabel('Frequency [Hz]')
113 plt.xlim([-1000, 1000])
114 plt.axvline(x=-fm, color='r', linestyle='--') # mark fm
115 plt.axvline(x=fm, color='r', linestyle='--')
116 plt.ylabel('Magnitude')
117 plt.tight_layout()
118 plt.show()
119
120 plt.figure()
121 plt.plot(t, original_signal)
122 plt.title('Resampled Unfiltered Audio Message Signal - Time Domain')
123 plt.xlabel('Time [s]')
124 plt.ylabel('Amplitude')
125 plt.tight_layout()
126 plt.show()
127
128 # low pass filter the audio to make it bandlimited to fm
129 original_signal = np.copy(lowpass_filter(original_signal, cutoff=fm, sample_rate=Fs, order=8))
130 write_wav('original_filtered.wav', original_signal, Fs)
131
132 # print the parameters
133 print('Carrier frequency: {} Hz'.format(f_c))
134 print('Modulation index: {}'.format(beta))
135 print('Frequency sensitivity: {} Hz/V'.format(kf))
136 print('TX Bandwidth: {} Hz'.format(bt))
137 print('Duration: {} s'.format(duration))
138 print('Sampling frequency: {} Hz'.format(Fs))
139 print('Number of samples: {}'.format(len(original_signal)))
140 print('Message amp: {}'.format(Am))
141
142 # Plot original signal in time and frequency domain
143 original_signal_fft = fftshift(fft(original_signal))
144 f = np.linspace(-Fs/2, Fs/2, len(original_signal))
145 plt.plot(f, np.abs(original_signal_fft))
146 plt.title('Audio Message Signal - Frequency Domain')
147 plt.xlabel('Frequency [Hz]')
148 plt.xlim([-bt, bt])
149 plt.axvline(x=-fm, color='r', linestyle='--') # mark fm
150 plt.axvline(x=fm, color='r', linestyle='--')
151 plt.ylabel('Magnitude')
152 plt.tight_layout()
153 plt.show()
154
155 plt.figure()
```

```
156 plt.plot(f, np.abs(original_signal_fft))
157 plt.title('Audio␣Message␣Signal␣-␣Frequency␣Domain␣(<␣1␣kHz)')
158 plt.xlabel('Frequency␣[Hz]')
159 plt.xlim([-1000, 1000])
160 plt.axvline(x=-fm, color='r', linestyle='--') # mark fm
161 plt.axvline(x=fm, color='r', linestyle='--')
162 plt.ylabel('Magnitude')
163 plt.tight_layout()
164 plt.show()
165
166 plt.figure()
167 plt.plot(t, original_signal)
168 plt.title('Audio␣Message␣Signal␣-␣Time␣Domain')
169 plt.xlabel('Time␣[s]')
170 plt.ylabel('Amplitude')
171 plt.tight_layout()
172 plt.show()
173
174 # FM Modulation
175 message_integral = np.cumsum(original_signal) / Fs
176
177 # FM modulated signal
178 fm_signal = np.cos(2 * np.pi * f_c * t + 2 * np.pi * kf * message_integral) # carrier amplitude
      = 1 by default
179
180 # Plot FM signal
181 fm_signal_fft = fftshift(fft(fm_signal))
182 f = np.linspace(-Fs/2, Fs/2, len(fm_signal))
183 plt.plot(f, np.abs(fm_signal_fft))
184 plt.title('Frequency␣Modulated␣Signal␣-␣Frequency␣Domain')
185 plt.xlabel('Frequency␣[Hz]')
186 # plt.xlim([-bt, bt])
187 plt.axvline(x=f_c - bt, color='r', linestyle='--') # mark tx bandwidth
188 plt.axvline(x=f_c + bt, color='r', linestyle='--')
189 plt.axvline(x=-f_c - bt, color='r', linestyle='--') # mark tx bandwidth
190 plt.axvline(x=-f_c + bt, color='r', linestyle='--')
191 plt.ylabel('Magnitude')
192 plt.tight_layout()
193 plt.show()
194
195 # Plot FM signal
196 fm_signal_fft = fftshift(fft(fm_signal))
197 f = np.linspace(-Fs/2, Fs/2, len(fm_signal))
198 plt.plot(f, np.abs(fm_signal_fft))
199 plt.title('Frequency␣Modulated␣Signal␣-␣Frequency␣Domain')
200 plt.xlabel('Frequency␣[Hz]')
201 plt.xlim([f_c-bt, f_c+bt])
202 plt.axvline(x=f_c - bt, color='r', linestyle='--') # mark tx bandwidth
203 plt.axvline(x=f_c + bt, color='r', linestyle='--')
204 plt.axvline(x=-f_c - bt, color='r', linestyle='--') # mark tx bandwidth
205 plt.axvline(x=-f_c + bt, color='r', linestyle='--')
```

```
206 plt.ylabel('Magnitude')
207 plt.tight_layout()
208 plt.show()
209
210 plt.figure()
211 plt.plot(t, fm_signal)
212 plt.title('Frequency␣Modulated␣Signal␣-␣Time␣Domain')
213 plt.xlabel('Time␣[s]')
214 plt.ylabel('Amplitude')
215 plt.tight_layout()
216 plt.show()
217
218 plt.figure()
219 plt.plot(t, fm_signal)
220 plt.title('Frequency␣Modulated␣Signal␣-␣Time␣Domain')
221 plt.xlabel('Time␣[s]')
222 plt.xlim([2.02, 2.021])
223 plt.ylabel('Amplitude')
224 plt.tight_layout()
225 plt.show()
226
227 # Add AWGN noise to the FM signal
228 noise_variance = 1e-5
229 noise = np.sqrt(noise_variance) * np.random.randn(len(t))
230 noise_fft = fftshift(fft(noise))
231 f = np.linspace(-Fs/2, Fs/2, len(noise))
232 fm_signal_noisy = fm_signal + noise
233 fm_signal_noisy_fft = fftshift(fft(fm_signal_noisy))
234
235 # plot noise
236 plt.figure()
237 plt.plot(t, noise)
238 plt.title('Noise')
239 plt.xlabel('Time␣[s]')
240 plt.ylabel('Amplitude')
241 plt.tight_layout()
242 plt.show()
243
244 # plot noise in frequency domain
245 plt.figure()
246 plt.plot(f, np.abs(noise_fft))
247 plt.plot(f, np.mean(np.abs(noise_fft))*np.ones(len(f)), 'r--')
248 plt.title('Noise␣-␣Frequency␣Domain')
249 plt.xlabel('Frequency␣[Hz]')
250 plt.ylabel('Magnitude')
251 plt.tight_layout()
252 plt.show()
253
254 # Plot FM signal with noise
255 plt.figure()
256 plt.plot(t, fm_signal_noisy)
```

```
257 plt.title('Frequency␣Modulated␣Signal␣with␣Noise␣-␣Time␣Domain')
258 plt.xlabel('Time␣[s]')
259 plt.ylabel('Amplitude')
260 plt.tight_layout()
261 plt.show()
262
263 # Plot FM signal with noise in frequency domain
264 plt.figure()
265 plt.plot(f, np.abs(fm_signal_noisy_fft))
266 plt.title('Frequency␣Modulated␣Signal␣with␣Noise␣-␣Frequency␣Domain')
267 plt.xlabel('Frequency␣[Hz]')
268 plt.ylabel('Magnitude')
269 plt.tight_layout()
270 plt.show()
271
272 plt.figure()
273 plt.plot(f, np.abs(fm_signal_noisy_fft))
274 plt.title('Frequency␣Modulated␣Signal␣with␣Noise␣-␣Frequency␣Domain')
275 plt.xlabel('Frequency␣[Hz]')
276 plt.xlim([f_c-bt, f_c+bt])
277 plt.ylabel('Magnitude')
278 plt.tight_layout()
279 plt.show()
280
281 print('Noise␣variance:␣{}'.format(np.var(noise)))
282 print('Noise␣mean:␣{}'.format(np.mean(noise)))
283
284 # Apply bandpass filter
285 bp_filtered_signal = bandpass_filter(fm_signal_noisy, f_c - 0.5*bt, f_c + 0.5*bt, Fs)
286 bp_filtered_signal_fft = fftshift(fft(bp_filtered_signal))
287 f = np.linspace(-Fs/2, Fs/2, len(bp_filtered_signal))
288
289 # Plot bandpass filtered signal
290 plt.figure()
291 plt.plot(t, bp_filtered_signal)
292 plt.title('Bandpass␣Filtered␣Signal␣-␣Time␣Domain')
293 plt.xlabel('Time␣[s]')
294 plt.ylabel('Amplitude')
295 plt.tight_layout()
296 plt.show()
297
298 # Plot bandpass filtered signal in frequency domain
299 plt.figure()
300 plt.plot(f, np.abs(bp_filtered_signal_fft))
301 plt.title('Bandpass␣Filtered␣Signal␣-␣Frequency␣Domain')
302 plt.xlabel('Frequency␣[Hz]')
303 plt.ylabel('Magnitude')
304 plt.axvline(x=f_c - 0.5*bt, color='r', linestyle='--') # mark tx bandwidth
305 plt.axvline(x=f_c + 0.5*bt, color='r', linestyle='--') # mark tx bandwidth
306 plt.axvline(x=-f_c - 0.5*bt, color='r', linestyle='--') # mark tx bandwidth
307 plt.axvline(x=-f_c + 0.5*bt, color='r', linestyle='--') # mark tx bandwidth
```

```
plt.tight_layout()
plt.show()

plt.figure()
plt.plot(f, np.abs(bp_filtered_signal_fft))
plt.title('Bandpass␣Filtered␣Signal␣-␣In␣Transmission␣Pass-band')
plt.xlabel('Frequency␣[Hz]')
plt.xlim([f_c-bt, f_c+bt])
plt.ylabel('Magnitude')
plt.tight_layout()
plt.show()

# Take differentiation of the bandpass filtered signal
# Corrected Differentiation of the signal
# The differentiated signal should be one element shorter, so we adjust the time array
    accordingly
differentiated_signal = np.diff(bp_filtered_signal) / np.diff(t)
t_diff = np.arange(0, len(differentiated_signal)/Fs, 1/Fs)

# Plot differentiated signal with corrected time array
plt.figure()
plt.plot(t_diff, differentiated_signal)
plt.title('Differentiated␣Signal␣-␣Time␣Domain')
plt.xlabel('Time␣[s]')
plt.ylabel('Amplitude')
plt.tight_layout()
plt.show()

# Plot differentiated signal with corrected time array
differentiated_signal_fft = fftshift(fft(differentiated_signal))
f = np.linspace(-Fs/2, Fs/2, len(differentiated_signal))
plt.figure()
plt.plot(f, np.abs(differentiated_signal_fft))
plt.title('Differentiated␣Signal␣-␣Frequency␣Domain')
plt.xlabel('Frequency␣[Hz]')
plt.ylabel('Magnitude')
plt.tight_layout()
plt.show()

plt.figure()
plt.plot(f, np.abs(differentiated_signal_fft))
plt.title('Differentiated␣Signal␣-␣Frequency␣Domain')
plt.xlabel('Frequency␣[Hz]')
plt.xlim([f_c-bt, f_c+bt])
plt.ylabel('Magnitude')
plt.tight_layout()
plt.show()


# Envelope detection using the Hilbert transform on the corrected differentiated signal
# any value other than 'hilbert' will use the low-pass filter method
```

```
358 envelope = get_envelope(differentiated_signal, fm, method='hilbert')
359 envelope_fft = fftshift(fft(envelope))
360 f = np.linspace(-Fs/2, Fs/2, len(envelope))
361
362 plt.figure()
363 plt.plot(t_diff, envelope)
364 plt.title('Envelope Detected Signal - Time Domain')
365 plt.xlabel('Time [s]')
366 plt.ylabel('Amplitude')
367 plt.tight_layout()
368 plt.show()
369
370 plt.figure()
371 plt.plot(f, np.abs(envelope_fft))
372 plt.title('Envelope Detected Signal - Frequency Domain')
373 plt.xlabel('Frequency [Hz]')
374 plt.ylabel('Magnitude')
375 plt.tight_layout()
376 plt.show()
377
378 plt.figure()
379 plt.plot(f, np.abs(envelope_fft))
380 plt.title('Envelope Detected Signal - Frequency Domain')
381 plt.xlabel('Frequency [Hz]')
382 plt.xlim([-bt, bt])
383 plt.ylabel('Magnitude')
384 plt.tight_layout()
385 plt.show()
386
387 # Apply low-pass filter on the envelope detected signal
388 lp_filtered_signal = lowpass_filter(envelope, fm, Fs, 8)
389
390 # remove the first and last 1000 samples
391 lp_filtered_signal = lp_filtered_signal[1000:-1000]
392 t = np.arange(0, len(lp_filtered_signal)/Fs, 1/Fs)
393
394 # dc blocking
395 lp_filtered_signal = np.copy(lp_filtered_signal - np.mean(lp_filtered_signal))
396
397 # match gain
398 lp_filtered_signal = np.copy(lp_filtered_signal / np.max(lp_filtered_signal) * Am)
399 t_diff = np.arange(0, len(lp_filtered_signal)/Fs, 1/Fs)
400 write_wav('fm_demodulated.wav', lp_filtered_signal, Fs)
401
402 t_original = np.arange(0, len(original_signal)/Fs, 1/Fs)
403
404 # Plot low-pass filtered signal with corrected time array
405 plt.figure()
406 plt.plot(t_original, original_signal, label = 'Original Signal')
407 plt.plot(t_diff, lp_filtered_signal, label = 'Demodulated Signal')
408 plt.title('Low-pass Filtered Signal - Time Domain')
```

```
409 plt.xlabel('Time␣[s]')
410 plt.ylabel('Amplitude')
411 plt.tight_layout()
412 plt.legend()
413 plt.show()
414
415 # Plot low-pass filtered signal with corrected time array
416 lp_filtered_signal_fft = fftshift(fft(lp_filtered_signal))
417 f = np.linspace(-Fs/2, Fs/2, len(lp_filtered_signal))
418 plt.figure()
419 plt.plot(f, np.abs(lp_filtered_signal_fft))
420 plt.title('Low-pass␣Filtered␣Signal␣-␣Frequency␣Domain')
421 plt.xlabel('Frequency␣[Hz]')
422 plt.ylabel('Magnitude')
423 plt.tight_layout()
424 plt.show()
425
426 plt.figure()
427 plt.plot(f, np.abs(lp_filtered_signal_fft))
428 plt.title('Low-pass␣Filtered␣Signal␣-␣Frequency␣Domain')
429 plt.xlabel('Frequency␣[Hz]')
430 plt.xlim([-bt, bt])
431 plt.ylabel('Magnitude')
432 plt.tight_layout()
433 plt.show()
434
435 # Function to calculate signal power
436 def calculate_power(signal):
437     return np.mean(signal ** 2)
438
439 # Pre-detection snr Calculation
440 fm_signal_power = calculate_power(fm_signal)
441 noise_power = calculate_power(noise)
442 pre_snr = 10 * np.log10(fm_signal_power / noise_power)
443 print('Pre-SNR:␣{:.2f}␣dB'.format(pre_snr))
444
445 # postdetection snr
446 bp_filtered_noise = bandpass_filter(noise, f_c - 0.5*bt, f_c + 0.5*bt, Fs)
447 differentiated_noise = np.diff(bp_filtered_noise) / Fs
448 envelope_noise = get_envelope(differentiated_noise, fm, method='hilbert')
449 lp_filtered_noise = lowpass_filter(envelope_noise, fm, Fs, 8)
450 lp_filtered_noise = lp_filtered_noise[1000:-1000]
451 lp_filtered_noise -= np.mean(lp_filtered_noise)
452 signal_power = np.mean(lp_filtered_signal**2)
453 noise_power = np.mean(lp_filtered_noise**2)
454 post_SNR = 10 * np.log10(signal_power / noise_power)
455 print(post_SNR)
```

References

[1] S. Haykin and M. Moher, *Introduction to Analog & Digital Communications*, 2nd ed. NJ: Wiley, 2007, ISBN: 978-0-471-43222-7.

[2] T. Ulrich. "Envelope calculation from the hilbert transform," ResearchGate. (Mar. 17, 2006), [Online]. Available: https://www.researchgate.net/publication/257547765_Envelope_Calculation_from_the_Hilbert_Transform (visited on 12/30/2023).

[3] E. Kudeki and D. C. Munson, *Analog Signals and Systems* (Illinois ECE Series). NJ: Pearson Prentice Hall, 2009, 512 pp., ISBN: 978-0-13-143506-3.

[4] W. Storr. "Butterworth Filter Design and Low Pass Butterworth Filters," Basic Electronics Tutorials. (Aug. 14, 2013), [Online]. Available: https://www.electronics-tutorials.ws/filter/filter_8.html (visited on 12/31/2023).

[5] V. Khetarpal. "在 Python 中实现低通滤波器 [Implementing low-pass filters in Python]," Delft-Stack. (Dec. 21, 2022), [Online]. Available: https://www.delftstack.com/zh/howto/python/low-pass-filter-python/ (visited on 12/29/2023).

[6] D. Manolakis and V. Ingle, *Applied Digital Signal Processing*. NY: Cambridge University Press, 2011, ISBN: 978-0-521-11002-0.

[7] The SciPy Community. "Scipy.signal.butter," SciPy v1.11.4 Manual. (n.d.), [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html#scipy.signal.butter (visited on 12/27/2023).

[8] The SciPy Community. "Scipy.signal.filtfilt," SciPy v1.11.4 Manual. (n.d.), [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html#scipy.signal.filtfilt (visited on 12/27/2023).

[9] Sasmita. "Modulation Index or Modulation Factor of AM Wave," Electronics Post. (May 19, 2020), [Online]. Available: https://electronicspost.com/modulation-index-or-modulation-factor-of-am-wave/ (visited on 01/04/2024).

[10] 西笑生. "Python 提取信号的包络 [Get envelope of a signal in Python]," CSDN Blog. (Mar. 10, 2023), [Online]. Available: https://blog.csdn.net/flyfish1986/article/details/129444260 (visited on 12/29/2023).

[11] "Analog Communication - AM Demodulators," tutorialspoint. (n.d.), [Online]. Available: https://www.tutorialspoint.com/analog_communication/analog_communication_am_demodulators.htm (visited on 01/02/2024).

[12] J. Lesurf. "The Envelope Detector." (n.d.), [Online]. Available: https://www.winlab.rutgers.edu/~crose/322_html/envelope_detector.html (visited on 01/02/2024).

[13] "Carson's Rule," DAEnotes. (Nov. 12, 2017), [Online]. Available: https://www.daenotes.com/electronics/communication-system/carsons-rule (visited on 01/04/2024).

[14] "Butterworth Filter: What is it? (Design & Applications) | Electrical4U," https://www.electrical4u.com/. (Apr. 16, 2021), [Online]. Available: https://www.electrical4u.com/butterworth-filter/ (visited on 01/08/2024).

[15] The SciPy Community. "Scipy.signal.lfilter," SciPy v1.11.4 Manual. (n.d.), [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html#scipy.signal.lfilter (visited on 12/27/2023).