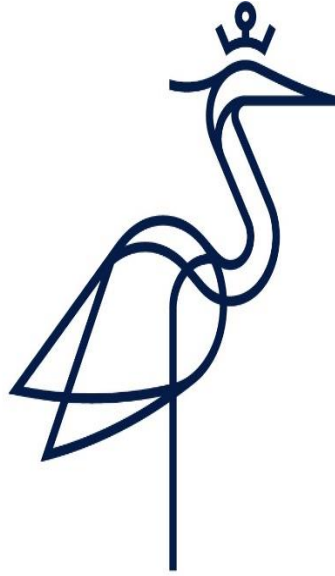


ANGLIA RUSKIN UNIVERSITY

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT



**a.r.u. | Anglia Ruskin
University**

MSc ELECTRONIC/ELECTRICAL ENGINEERING

ADVANCED DIGITAL SIGNAL PROCESSING

MOD007721

Analysis of Speech Vocoding and Digital Filter Design Using MATLAB

(TECHNICAL REPORT)

NAME: HAKEEM ISSAH

STUDENT ID: 2452525

Submitted on 5th DECEMBER, 2025

TABLE OF CONTENTS

Introduction	5
Working Principles of Vocoder.....	5
Technical Description of the Vocoder Design	6
Technical Specifications of the Vocoder Design (MATLAB Implementation)	6
Graphs of Output and Analysis	11
The use of FFT (Fast Fourier Transform).....	15
The use of Rectification	15
Vocoder in Music	15
Vocoder vs. Autotune.....	16
Applications of Vocoder.....	16
Conclusion	16
PART 2: DIGITAL FILTER DESIGN	17
Introduction.....	18
Design Methodology	18
Task 1	19
Task 2	22
Task 3	25
Task 4	28
Task 5	31
Task 6	34
Task 7	37
Task 8	39
Task 9	40
REFERENCES.....	42
APPENDIX.....	43
Vocode Design Code.....	43
Filter Code	44

TABLE OF FIGURES

Figs. 1(a)-(b): Original Sound Clips (time and frequency spectrum).....	12
Figs. 2: Rectified signal (time and spectrum).....	12
Fig. 3: Low-pass filtered signal	13
Figs. 4(a)-(b): Modulated with noise signal (time and frequency spectrum)	13
Figs. 5(a)-(b): Bandpass-filtered signal (time and frequency spectrum).....	14
Figs. 6(a)-(b): Full Vocoder Output (time and frequency spectrum)	14
Figs. 7(a)-(c): FIR LPF delay, magnitude, and phase at order 50 and minimum.....	19
Figs. 8(a)-(c): FIR LPF magnitude, phase, and delay at density factors 50 and 20	20
Figs. 9(a)-(c): FIR HPF of magnitude, phase, and delay at order 100 and minimum	22
Figs. 10(a)-(b): FIR HPF of magnitude, phase, and delay at density factors 20 and 50	23
Figs. 11(a)-(c): FIR BPF of delay, magnitude, and phase at order 100 and minimum.....	25
Figs. 12(a)-(c): FIR BPF magnitude, phase, and delay at density factors 20 and 50	26
Figs. 13(a)-(c): FIR BPF delay, magnitude, and phase response at order 100 and minimum.....	28
Figs. 14(a)-(c): FIR BPF of magnitude, phase, and delay at density factors 20 and 50	29
Figs. 15(a)-(c): IIR BPF Magnitude, phase, and delay waveforms at minimum order	32
Figs. 16(a)-(c): IIR BPF magnitude, phase, and delay waveforms at order 100 (filterDesigner).....	32
Fig. 17: MATLAB Code error message in order 100 IIR design	32
Fig. 18: filterDesigner specifications for order 100 IIR filter design	33
Figs. 19(a)-(c): IIR BPF magnitude, phase, and delay waveforms at minimum order	34
Figs. 20(a)-(c): IIR BPF delay, magnitude, and phase waveforms at order 100 (filterDesigner).....	35
Fig. 21: MATLAB Code Error message of order 100 IIR filter design	35
Fig. 22: No Density Factor specification in IIR design	36
Figs. 23(a)-(c): FIR BPF of magnitude, phase, and delay at orders 100 and 130	37
Figs. 24(a)-(c): Elliptic LPF magnitude, phase, and delay at orders 50 and a minimum	of 40
Fig. 25: No density factor specification	41

PART 1: VOCODER IMPLEMENTATION

Introduction

A vocoder is an audio processor that takes the human voice and reshapes it into something artificial, often giving it a metallic or robotic quality. It does this by analysing the speech signal, breaking it into frequency bands, and then using that information to modulate another sound source, effectively re-sculpting the spectral content of the voice. Vocoder was developed first as a communication technology, but later became a common tool in electronic music and hip-hop, used by producers and musicians. (Cychosz et al., 2024; Chakraborty & Krishnan, 2023). In the 1930s, a scientist named Homer Dudley was the first to develop the design and concept of the vocoder. At that moment, the aim was to compress the speech signal, minimising its bandwidth requirements for transmission along communication lines. Vocoder technology was critical during World War II, as the military and high-ranking officials communicated securely over these systems.

It was used in the SIGSALY system, an encrypted speech transmission system that enabled secure conversations between Allied leaders. Notable figures like Franklin D. Roosevelt and Winston Churchill communicated via SIGSALY, which is often cited as one of the earliest secure voice systems (Bennett, P., 2008).

Working Principles of Vocoder

A vocoder operates using two main signals: the **modulator** and the **carrier**.

- The **modulator signal** (typically speech) is first passed through a bank of analysis filters, usually a set of band-pass filters.
- These band-pass filters split the input into multiple **frequency bands**, each capturing energy in a specific region of the spectrum.
- The output of each band is then measured by an **analysis envelope detector** (often called an analysis meter). A level analyser determines the amplitude or energy of each band over time.

Meanwhile, the **carrier signal** (such as noise or a synthesizer sound) is sent through a similar set of **output band-pass filters**. These are aligned with the analysis filters in terms of their frequency ranges. Still, they do not carry the detailed harmonic structure of the original speech, only a generic version of the modulator's spectral layout.

The envelopes extracted from the modulator bands are then used to control the amplitude of the carrier signal's corresponding bands. In other words, each carrier band is “shaped” by the time-varying envelope of the matching modulator band.

- If the vocoder has **more band-pass filters**, the resulting vocoded sound more closely resembles the original speech (modulator) in terms of intelligibility and articulation. (Mushtaq et al., 2021)
- In practice, we supply the voice as the **modulator** and choose an appropriate **carrier**; the result is a speech signal with a characteristically robotic or synthetic tone.

The choice of carrier affects the final sound significantly:

- A simple waveform (e.g., a sine wave) tends to produce a very robotic or “alien” voice.
- A richer carrier (e.g., a complex synthesizer pad or chord) yields a more musical, stylized, or “processed” vocal quality.

Technical Description of the Vocoder Design

1. Bands

The modulator (voice) signal is divided into separate **frequency bands** by passing it through multiple band-pass filters. The spectrum allows only a limited number of bands. In the past, vocoder systems had fewer bands, resulting in a more classic or vintage sound. Intuitively, closer attention to detail and greater intelligibility would be evident in modern designs, and they are. If we desire a more retro sound, fewer bands are selected.

2. Frequency Range

The **frequency range** analysed by the vocoder determines how much harmonic information it captures from the input signal.

- A **wider frequency range** preserves more of the original spectral content and can produce a more detailed, natural-sounding output.
- A **narrower frequency range** focuses on certain parts of the spectrum (for instance, the midrange), which can emphasize particular vocal characteristics while disregarding others.

3. Voiced and Unvoiced Sounds

Speech consists of **voiced** (periodic, pitch-based) and **unvoiced** (noise-like) components.

- **Voiced sounds** (like vowels) must preserve both their pitch and formant structure to remain intelligible.
- **Unvoiced sounds** (like “s”, “f”, “sh”) are more noise-like and must be handled carefully so their transient characteristics and clarity are not lost or overly smoothed.

A well-designed vocoder treats these two categories appropriately to maintain intelligibility and naturalness.

4. Formants

Formants are resonant frequency bands of the human vocal tract that largely determine the perceived timbre of a voice. For the vocoder output to sound understandable and recognisably “vocal,” it must preserve these formants as far as possible. Techniques such as formant shifting can be used creatively to alter the timbre without changing pitch, but the basic formant structure should remain coherent for the speech to be intelligible.

Technical Specifications of the Vocoder Design (MATLAB Implementation)

In line with the assignment specification, this section outlines the technical realisation of a common vocoder in MATLAB. We would focus on a speech that still remains intelligible after processing, called 'Shannon Speech,' despite high spectral manipulation (Cychosz et al., 2024). This system follows the structure of a classical vocoder:

- Filtering (Bandpass) of the input into the channels
- Low-pass filtering and rectification to extract envelopes
- Modulation of white noise

- Final band-pass filtering and summation of channels

This implementation allows key digital signal processing (DSP) concepts to be explored, while demonstrating both basic and more advanced aspects of vocoder technology.

1. Loading and Saving the Audio File *bkbe2114.wav*

The first step is to **read the input audio file**. The MATLAB script and the file *bkbe2114.wav* must be stored in the same directory to avoid path errors. If they are not in the exact location, MATLAB will be unable to find the file, leading to an error when attempting to read it.

We use:

- `[y, Fs] = audioread('bkbe2114.wav');`

Here:

- `Fs` is the **sampling frequency** in Hz,
- `y` contains the **sampled audio data**.

The variable `y` can be inspected in the workspace to examine the waveform samples.

2. Listening to the Audio File

It is useful to audition the original audio before processing:

- The program can display a prompt in the Command Window asking the user to press Enter to hear the file.
- The command `soundsc(y, Fs)` then plays the audio. `soundsc` first scales the signal so that it plays as loudly as possible without clipping, and `Fs` specifies the playback sample rate.

This step helps confirm that the file has been read correctly and allows the user to compare the original and vocoded outputs later.

3. Plotting the Audio Signal

Next, the time-domain representation of the audio signal is plotted.

- A time axis `t` is constructed from the sample indices and sampling frequency.
- The command `subplot` is used to display multiple plots in a single figure window.
- The 16-bit audio samples in `y` are plotted against time, with time on the x-axis and amplitude on the y-axis.

This visualisation shows the overall shape and duration of the speech waveform and forms a reference for later stages.

4. Plotting the Spectrum

To analyse the frequency content, the **Fast Fourier Transform (FFT)** is applied:

- `Y = fft(y);`
- `abs(fft(y))` returns the **magnitude spectrum**, but because FFT values are often complex and may contain harmful components, `abs` is used to obtain the absolute magnitude.

A normalised magnitude spectrum can be plotted by dividing by the maximum value. A **frequency axis** is generated using, for example, `linspace`, and the data are plotted using `plot`.

A subplot is used again to display multiple spectra in the same figure window.

- `xlabel` and `ylabel` label the frequency and magnitude axes.
- Axis limits are set to make the relevant parts of the spectrum clearly visible.

This step provides insight into the distribution of energy across frequencies in the modulator signal.

5. Generating Random Noise

The vocoder uses white noise as the **carrier signal**, so a random noise vector of the same length as the modulator is generated.

- This white noise has a flat spectral distribution across the band of interest.
- It acts as the “blank canvas” onto which the speech envelopes will be imposed.

The noise must match the length of the speech (`y`) so that time-domain operations can be performed sample-by-sample.

6. Calculation of Cut-Off Frequencies and System Parameters

The system uses **12 channels**, with an input bandwidth of 0 to 11 kHz and a sampling frequency of 22,000 Hz (twice the maximum frequency).

Firstly, the 0-35 interval would give us a single maximum cut-off frequency (f) that exceeds half the Nyquist frequency (22,000 Hz), i.e., 11 kHz, which is not ideal. We find a new interval with the cut-off formula:

$$f = 165.4(10^{0.06x} - 1) \text{----- (1)}$$

where $f=f_{\text{max}}(\text{ideal}) = 11000 \text{ Hz}$

Solving this equation (1), we get: $X = 30.5$

Next, we find the x values within the interval, 0-30.5;

`x-values = (0, 2.54, 5.08, 7.62, 10.16, 12.70, 15.24,, 17.78, 20.32, 22.86, 25.4, 27.94, 30.50)`

They are then mapped into frequency cut-offs using the same formula as equation (1). The resulting approximate cut-off frequencies from the formula are as follows:

$F = (0, 69.53, 168.28, 308.55, 507.79, 790.77, 1192.71, 1763.61, 2574.51, 3726.27, 5362.19, 7685.79, 11000)$

Hence:

Channel 1 = 0-69.53 Hz, Channel 2 = 69.53-168.28 Hz, Channel 3 = 168.28-308.55 Hz, Channel 4 = 308.55-507.79 Hz, etc. Every one of these ranges defines a frequency bandwidth.

e.g.:

- Channel 1 bandwidth: about 70 Hz (0–69.5 Hz)
- Channel 2 bandwidth: about 99 Hz (69.5–168.3 Hz)

The sampling frequency, $F_s = 22,000$ Hz, is used both for processing and for normalising the cut-off frequencies (by dividing by $F_s/2$) when designing the filters.

7. Modulator Signal Passing Through Band-Pass Filters

In the **analysis stage**, the voice (modulator) signal is passed through a set of elliptical IIR band-pass filters—one per channel.

- A loop runs over channels $k = 1$ to N (where $N = 12$).
- For each channel, an elliptic band-pass filter is designed using something like:

```
(b, a) = ellip(n, Rp, Rs, Wp, 'bandpass');
```

where:

- n is the filter order,
- R_p is the passband ripple,
- R_s is the stopband attenuation,
- W_p contains the **normalised** passband edge frequencies, derived from the x values via the earlier formula and scaled by $F_s/2$.

The modulator is filtered channel by channel, and the spectrum of each filtered output can then be plotted. As the loop iterates over channels, we observe the signal being split into narrowband components. The purpose of this stage is to **analyse** the speech signal by splitting it into multiple frequency bands, enabling the extraction of the temporal envelope in each band.

8. Rectification

After band-pass filtering, each channel's output is **rectified**.

- Rectification converts the bipolar waveform (containing both positive and negative amplitudes) into a unipolar one by removing or flipping the harmful components.
- In the implementation, a loop runs over all samples of the filtered signal:
 - If a sample is greater than zero, it is kept,
 - If it is less than or equal to zero, it is set to zero.

This half-wave rectification step ensures that only positive values remain, making it easier to track the signal's amplitude envelope in each band. Rectification is therefore a key step in envelope extraction.

9. Passing Through the Low-Pass Filter

Rectification can introduce rapid fluctuations (“ripples”) at the signal’s fundamental frequency. To obtain a smooth envelope, the rectified signal is passed through a **low-pass filter**.

- A 3rd-order Butterworth IIR filter with a cut-off frequency of 160 Hz can be used, for example:

```
[bb, aa] = butter(3, 160/(Fs/2), 'low');
```

This third-order low-pass Butterworth filter:

- Smooths out the rectified waveform,
- Extracts the slower-varying envelope that contains the essential modulation characteristics of speech,
- Reduces high-frequency fluctuations while retaining the temporal shape of the intensity envelope.

10. Multiplying with the Carrier Signal

The smoothed envelope from each channel is then **multiplied by the white noise carrier** signal.

- This operation imposes the **low-frequency envelope** of the speech band onto the broadband carrier.
- In effect, according to the speech envelope, the noise amplitude in that frequency band is shaped.

The core modulation step states that the speech envelopes modulate the amplitude of the noise, producing a signal that carries the rhythm and speech envelope, but retains the spectral characteristics of the noise.

11. Final Band-Pass Filtering

Through each channel, the modulated signals are then passed into another group of bandpass filters, the **output filters**.

- These have the **same number and frequency specifications** as the analysis filters.
- For example, analysis bands 1–4 are mirrored by output bands 1–4 with the same cut-off frequencies.

By matching each analysis band to a corresponding output band, the vocoder reconstructs a speech-like structure on the noise-based carrier. The result is a signal whose spectral shape is controlled by the airline but whose envelopes follow those of the original speech.

12. Final Vocoder Output

To obtain the final vocoded signal, the outputs from all channels are **summed**:

- A variable, for example, y_{voc} , is initialised as a zero vector of the same size as y .

- Each channel's output (`y_bp_out`) is added to `y_voc`:

```
y_voc = y_voc + y_bp_out;
```

After all channels are accumulated, the program prompts the user (using `disp`) and then uses:

- `soundsc(y_voc, Fs);`

to play the vocoded output at full scale.

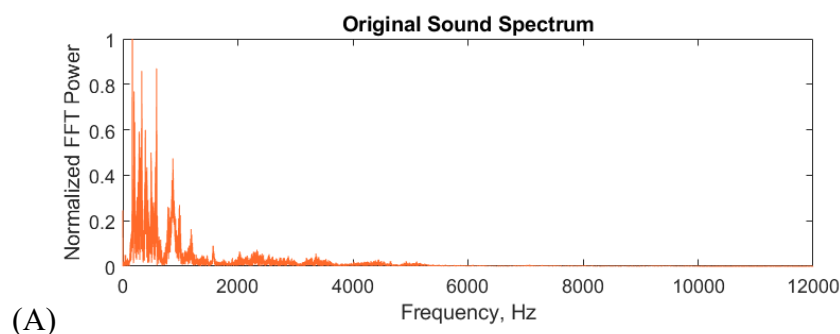
At this point, the original sentence (e.g., “Kettle boils quickly”) is heard with a **robotic or synthetic** character. All of the earlier steps—analysis filtering, rectification, low-pass filtering, modulation, and output filtering—combine to produce the classic vocoder effect.

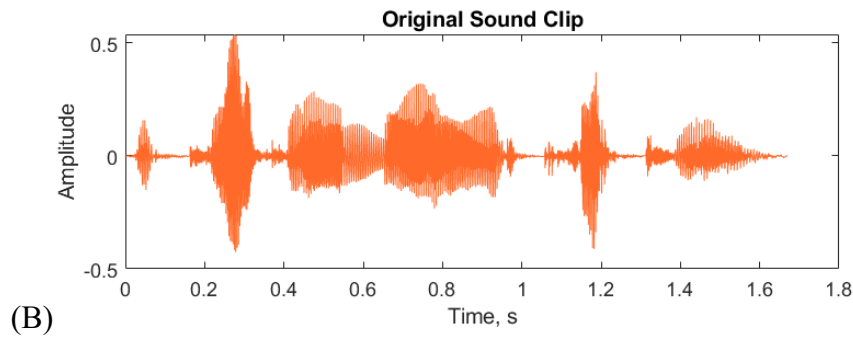
Graphs of Output and Analysis

Original Input Signal

In the time domain, the original sound clip shows clear speech-like bursts: short, high-amplitude oscillations grouped into syllables and words, separated by quieter intervals where the signal is near zero. These quick oscillations inside each burst are complemented by vocal-cord vibration and formants. For the frequency spectrum, this shows as strong energy concentrated at very low frequencies, formant bands and peaks normally below 4-5 kHz and quickly reducing energy as frequency increases to about 11-12 kHz

Together, these plots show a natural-voiced signal with structured periodicity and a low-frequency-dominated spectrum.

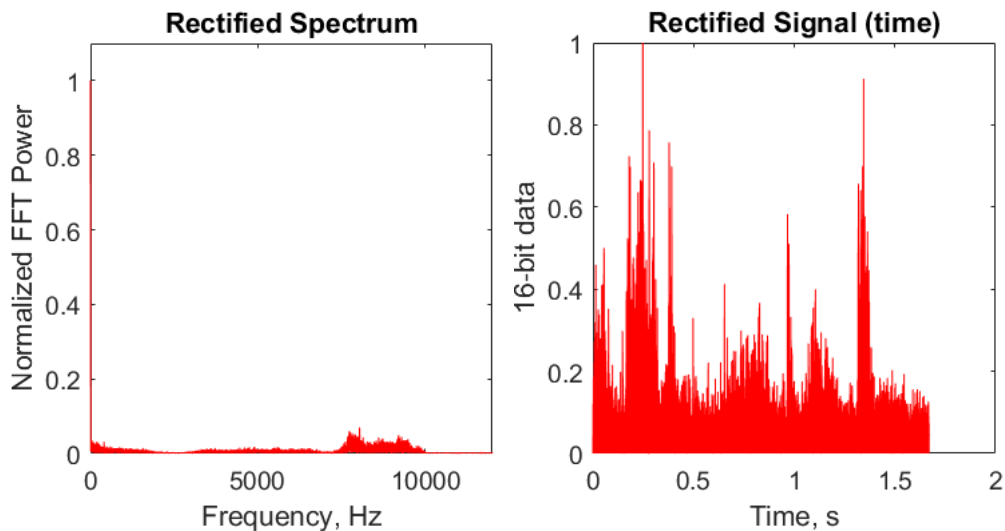




FIGS. 1(A)-(B): ORIGINAL SOUND CLIPS (TIME AND FREQUENCY SPECTRUM)

Rectification Process

After the rectification of half-wave, negative samples are clipped, meaning the time-domain signal doesn't cross zero. So you have a unipoar waveform where peaks show spikes of high energy in the band and valleys sit cose to zero. The distortion introduces a DC component and slow variations in the frequency domain, which appear as a concentration near 0 Hz (DC and very low frequencies), along with reduced but still visible higher-frequency content inherited from the original band-pass signal. So, together, these plots show how rectification converts a symmetric, band-limited, oscillatory signal into a positive waveform whose low-frequency components now carry information about the amplitude envelope.



FIGS. 2: RECTIFIED SIGNAL (TIME AND SPECTRUM)

Passing Through the Low-Pass Filter

The rectified signal, at 160 Hz, is low-pass filtered. When done the time waveform turns into a slowly varying, smooth curve that always becomes positive and tracks the total loudness of that band over time: it rises during speech segments (Chakraborty & Krishnan, 2023). It falls in pauses, with all rapid oscillations removed. The corresponding spectrum is confined almost entirely below 160 Hz, with most of the energy near DC and gradually decaying with frequency. Taken together, they show a clean amplitude envelope: a purely slow variation that preserves the timing and strength of speech events while discarding fine harmonic and pitch structure.

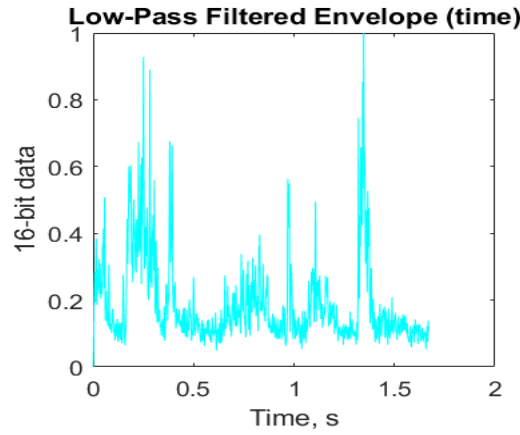
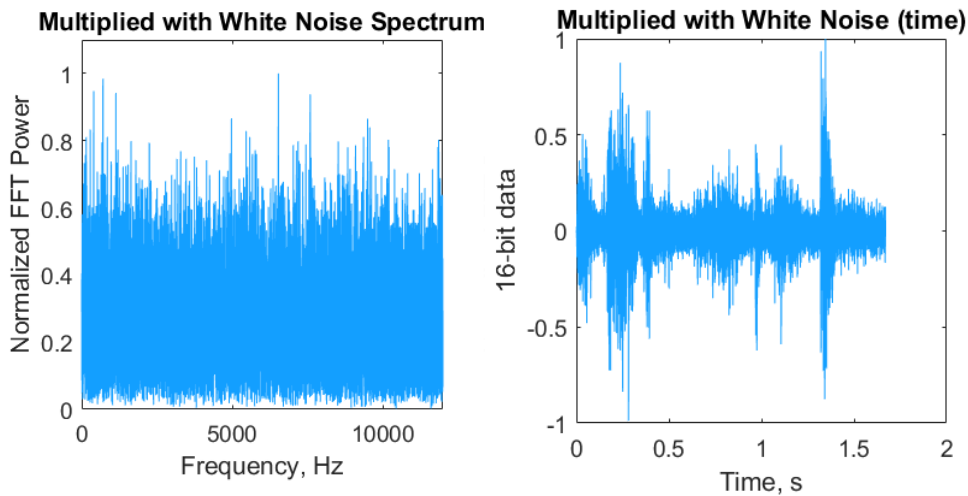


FIG. 3: LOW-PASS FILTERED SIGNAL

Multiplication with the Carrier Signal

When this smooth envelope is multiplied by white noise, the time-domain signal becomes jagged and noisy again. However, its overall shape still follows the envelope: the noise bursts are stronger where the envelope is high and weaker where it is low, so you can still see “clouds” of noise aligned with syllables. In the spectrum, the white noise spreads energy broadly across almost the entire 0–11 kHz band, giving a relatively flat magnitude response, but modulated slightly by the envelope’s slow changes. Together, these plots show a signal whose fine structure is random like noise, yet whose time-varying amplitude pattern still encodes the speech rhythm.

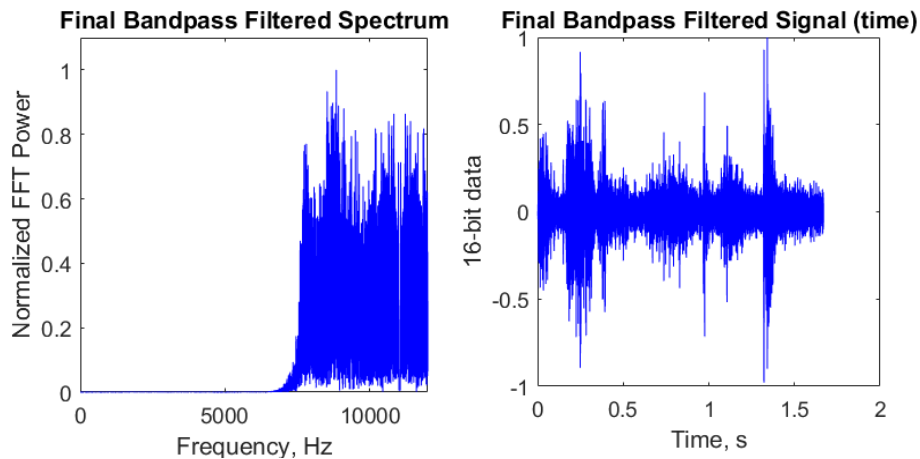


FIGS. 4(A)-(B): MODULATED WITH NOISE SIGNAL (TIME AND FREQUENCY SPECTRUM)

Final Bandpass Filtering

When the noise-modulated signal is passed through the same bandpass filter used in analysis, the noise-like feature still remains in the time waveform but looks narrower and smoother in bandwidth: you still see bursts aligned with the original speech envelope, but the rapid fluctuations are now constrained to a specific frequency range for this channel. In the frequency domain, the spectrum is no longer flat: energy is concentrated between the lower and upper cutoff frequencies of that band, with sharp attenuation outside that range. Together, these plots demonstrate how each channel produces a band-limited noise signal shaped in

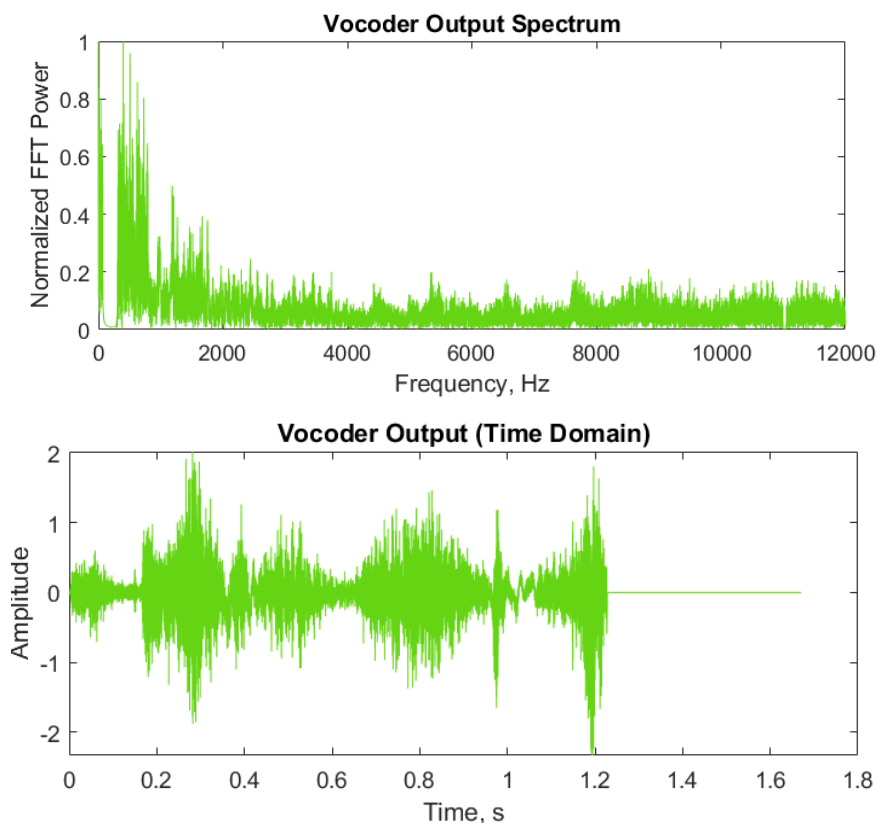
time by the speech envelope, ready to be summed with other channels to reconstruct a full-band vocoded signal.



FIGS. 5(A)-(B): BANDPASS-FILTERED SIGNAL (TIME AND FREQUENCY SPECTRUM)

Final Vocoder Output.

After adding all the channels, the output in the time domain looks like a noisy, dense waveform, yet keeps a clear, large-scale envelope that closely mimics the original speech, that is for regions where the talker displays high-variance noise, while pauses appear as low-amplitude sections. Looking at the spectrum, the signal is richer and noisier than the original, with the energy distributed across many frequencies. Yet, the overall spectral envelope still shows stronger energy at lower frequencies and characteristic bumps where formant bands from different channels overlap. Taken together, these final plots illustrate the central effect of the vocoder: it replaces the detailed harmonic structure of speech with band-limited noise while preserving the temporal and spectral envelopes that carry intelligibility, yielding a robotic but understandable voice.



FIGS. 6(A)-(B): FULL VOCODER OUTPUT (TIME AND FREQUENCY SPECTRUM)

The use of FFT (Fast Fourier Transform)

FFT is used in the vocoder implementation primarily because of its **efficiency** in computing the Discrete Fourier Transform (DFT). The Discrete Fourier Transform performs $O(N^2)$ operations for given N samples, while the FFT shrinks this to $O(N\log N)$, which is much faster for larger N (Takahashi, 2019).

For instance, for 2048 samples:

- A direct DFT would need the order of about four million complex multiplications,
- while an FFT would require around 11,000 operations.

This produces a speed improvement factor of approximately 370, with much reduced memory usage. As a result, FFT is ideal for spectrum analysis and visualisation in vocoder applications, allowing fast, practical analysis of speech and intermediate signals. (Abood, 2020)

The use of Rectification

Rectification is an essential step in the vocoder because it prepares the modulator signal (voice) for envelope detection. When the signal comes out of the bandpass filters, it is bipolar – it has both positive and negative parts. However, to understand how the signal's strength (amplitude) changes over time in each band, we only need the overall shape, or envelope, not the exact positive and negative swings. Rectification removes the negative part of the signal and converts it to unipolar form, making it easier for the low-pass filter to smooth it and extract the envelope. This envelope carries the critical speech information, such as syllable shapes and formant movements. By first rectifying and then low-pass filtering, the vocoder can capture the main speech characteristics in each band and apply them to the carrier signal, helping produce a clear and intelligible robotic voice.

Vocoders in Music

Vocoders are used in music because they allow artists to reshape the human voice into something new and expressive. They can make vocals sound robotic, futuristic, or “alien,” which fits electronic and experimental genres. At the same time, vocoders can thicken the sound by adding harmonies and smooth chords under the melody. They also help blend the voice with synths and pads, so the singer becomes part of the overall instrumental texture.

They appear in many styles, including electronic, pop, R&B, and even gospel and experimental music. Below are some examples of how vocoders have shaped modern sound:

Daft Punk – “Harder, Better, Faster, Stronger” (2001): The French electronic duo Daft Punk used heavy vocoder processing on the main vocals. The words are clearly understood, but the voice sounds robotic and machine-like, which fits their electronic, robot-themed image and stage persona perfectly.

Imogen Heap – “Hide and Seek” (2005): In this song, the voice is processed through a vocoder-like system with harmonies created from the chords. There are no traditional instruments at the start; the entire harmony feels like a choir made from one voice. This shows how vocoders can turn a single vocal line into rich, layered harmonies.

Bon Iver – “715 – CREΣKS” (2016): Here, the vocoder is used not just for a robotic effect, but to create an emotional, textured vocal sound. The processed voice feels both human and synthetic, adding depth and atmosphere to the song.

Vocoders vs. Autotune

Despite the confusion between vocoders and autotune to the general populace, they serve different purposes.

- The **vocoder** primarily changes and refine the **spectral envelope** of the sound. Through its group of bandpass filters, it changes the shape of the frequency content of a sound using information derived from another signal (modulator). The pitch doesn't change, but instead, there is transformation of the timbre, that frequently produces the robotic effects.
- But for auto-tune, it is mainly concerned with **the correction** of the pitch. It analyses the singer's voice and changes it by shifting frequencies down or up to the nearest wanted note
-
-
- The frequency content is manipulated specifically to change pitch, resulting in vocals that are more in-tune or deliberately "stepped" between notes for creative effect.

In summary, vocoders are used to **reshape the timbre** into something robotic or synthetic, while Auto-Tune is used to **adjust pitch** and correct or stylise tuning in vocal performances. (Splice, 2025; Hayes & Engel, 2024)

Applications of Vocoders

Vocoders have a broad range of applications, including:

- **Speech synthesis** – Generating artificial speech or transforming spoken audio into new timbres.
- **Music production** – Creating robotic, harmonised, or stylised vocal effects in various genres.
- **Artificial intelligence** – Assisting in speech coding, feature extraction, and experiments involving speech perception.
- **Broadcasting and audio engineering** – Efficient coding and transmission of speech signals.
- **Film and media** – Designing character voices and sound effects in movies, TV, and games.
- **Telecommunications** – Early secure communication systems and low-bit-rate speech transmission.

Conclusion

Vocoders are potent tools in audio and speech processing, enabling the creation of unique, expressive, and often highly synthetic sounds. The MATLAB-based speech vocoder described here demonstrates the whole chain of operations—analysis filtering, rectification, envelope extraction, noise modulation, and output filtering—while allowing detailed examination of spectra at each stage.

With the implementation of this system, we gain the technical know-how to operate vocoders and maintain speech intelligibility with fewer channels. In practice, these algorithms are reliable for secure voice transmission and for unique vocal effects in modern music production. Their ability to analyse, reshape, and reconstruct speech makes them an essential and enduring technology across many fields.

PART 2: DIGITAL FILTER DESIGN

Introduction

Part 2 of this report presents the design, implementation, and analysis of a series of digital filters using MATLAB. Both **finite impulse response (FIR)** and **infinite impulse response (IIR)** structures are investigated, with a focus on:

- Low-pass, high-pass, band-pass and band-stop frequency responses
- The impact of filter order on magnitude, phase, and group delay
- The role of the density factor in FIR equiripple design
- Practical limitations when attempting high-order IIR elliptic filters

All designs assume a frequency(sampling) $f_s = 48$ kHz. Frequency responses are examined through magnitude (in dB), phase (in radians), and group delay (in samples) plots generated in MATLAB and, for IIR designs, via the combination of code and the Filter Designer App.

Design Methodology

Two primary design methods are used:

1. FIR equiripple filters

- ✓ It is designed with the Parks-McClellan algorithm using `firpmord` and `firpm`
- ✓ Passband and stopband edge frequencies are determined, as well as desired amplitudes (0 or 1), and weights for the bands.
- ✓ The frequency-sampling grid for the optimisation is done with the density factor
- ✓ Two orders are contrasted in each task: a *minimum order* and a *fixed higher order* (50 or 100).

2. IIR elliptic filters

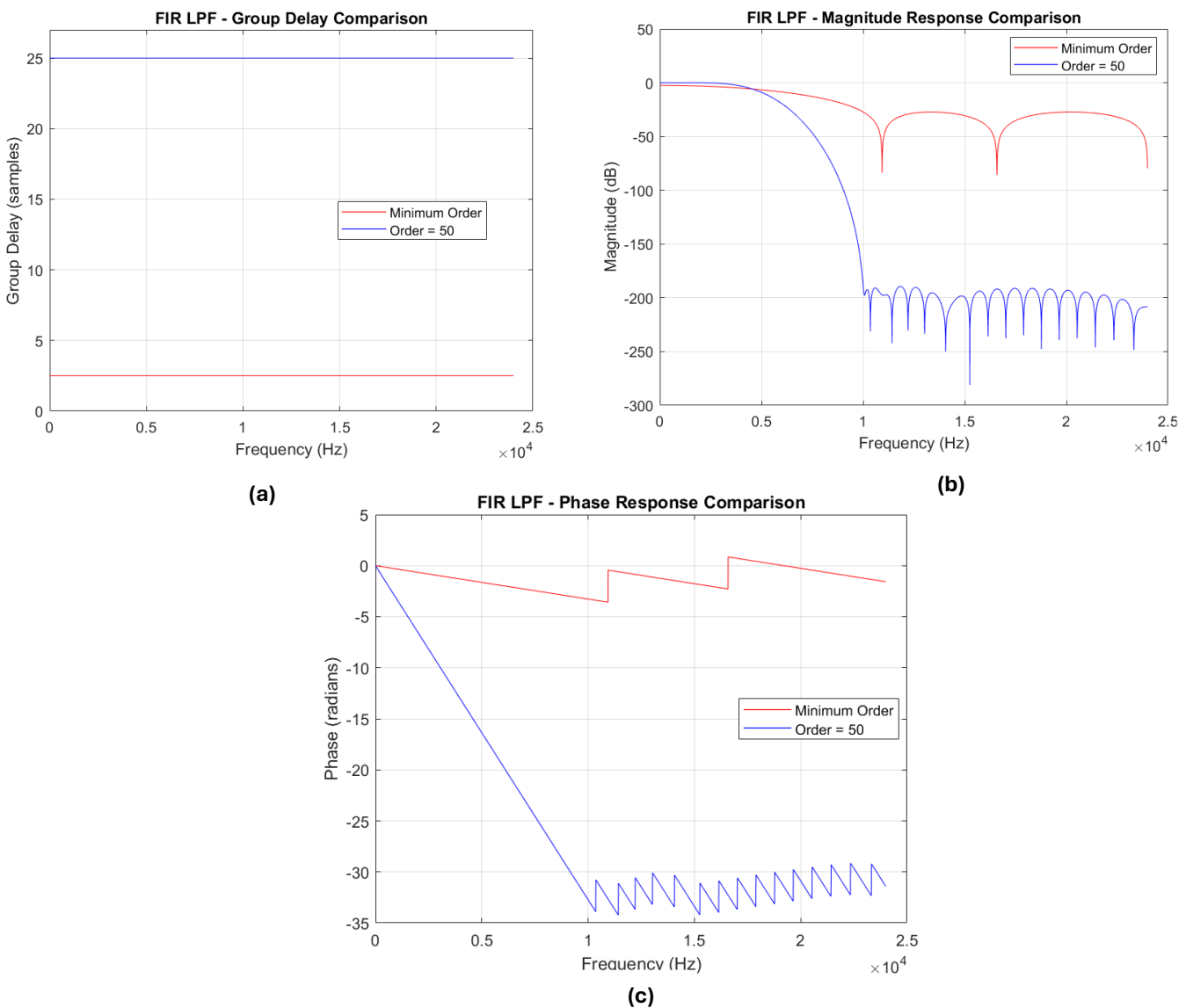
- ✓ They are designed using `ellipord` and `ellip` in MATLAB and also reproduced in the **Filter Designer App**.
- ✓ Elliptic design is picked for its acute transitions and high stopband weakening at low orders.
- ✓ A minimum order solution is designed for its specification, and then a higher order is attempted to explore limitations.
- ✓ For these types of filters, there is no density factor parameter. This concept applies only to the design of the FIR equiripple. (MathWorks, 2024a; NI, 2023; Wanhammar & Saramäki, 2020)

Task 1

Design Process

In this design, a Finite Impulse Response (FIR) Low-Pass Filter (LPF) is implemented in MATLAB using the Parks–McClellan method. The filter is designed for a sampling frequency of 48 kHz, with a passband from 300 Hz to 10 kHz, 1 dB passband ripple and 40 dB stopband attenuation. A minimum-order solution is first obtained, then a higher fixed order $N = 50$ is designed. Two density factors (20 and 50) are also used to examine their effect on the frequency-sampling grid. For each case, the magnitude, phase, and group-delay responses are plotted using a custom `plot_responses` function.

Effects of Different Orders on the Filter's Performance



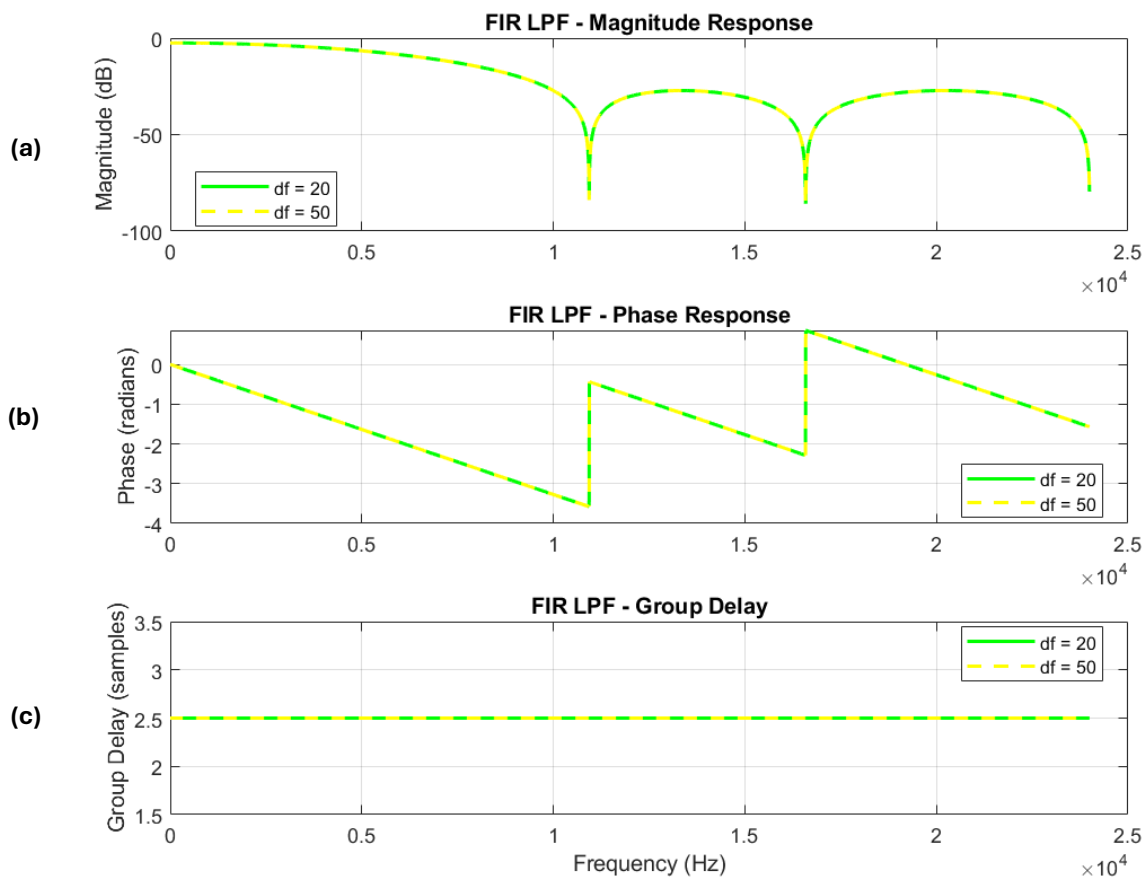
FIGS. 7(A)-(C): FIR LPF DELAY, MAGNITUDE, AND PHASE AT ORDER 50 AND MINIMUM

Observations: In the minimum-order design, the passband is flat at 0 dB, the stopband attenuation lies between -70 dB and -80 dB, and there is a gradual transition between the passband and stopband. For an order of 50, the passband remains flat, but there is steepening in the transition band, and the stopband reaches -200

dB, accompanied by small ripples, resulting in a sharper cutoff. The phase response is nearly linear in both cases, but the 50-order filter shows a larger overall phase shift. There is also a constant group delay of 2.5 samples and 25 samples for the minimum-order and 50-order cases, respectively; hence, both preserve the shape of their waveforms, but the higher-order cases introduce a longer delay.

Implications and Conclusion: It is clear that increasing the order significantly deepens the stopband attenuation and sharpens the transition, thereby suppressing unwanted high-frequency components. But there is a cost to this enhanced selectivity: a higher computational load and a larger time delay. The minimum-order design offers a simple, fast implementation, while the 50-order filter provides enhanced frequency-selective performance but is less appropriate for low-latency applications.

Effects of Different Density Factors on System Performance



FIGS. 8(A)-(C): FIR LPF MAGNITUDE, PHASE, AND DELAY AT DENSITY FACTORS 50 AND 20

Observations: In these two density factor scenarios, the magnitude displays a flat passband at around 0 dB and a stopband attenuation of about -80 dB. Also, the $df=20$ waveform looks less continuous, especially in the transition region, while the $df=50$ curve gives finely-resolved, smoother curve. The phase response for both filters is essentially the same linear trend, but again appears more jagged for $df = 20$ and more continuous for $df = 50$. Group delay remains constant at about 2.5 samples in both cases, with the $df = 50$ plot showing a cleaner, more uniform line.

Implications and Conclusion: Changing the density factor from 20 to 50 does **not** alter the filter's performance: passband flatness, stopband attenuation, linear phase, and constant group delay remain identical. The only effect is on the **visual resolution** of the plots. A higher density factor provides a denser frequency

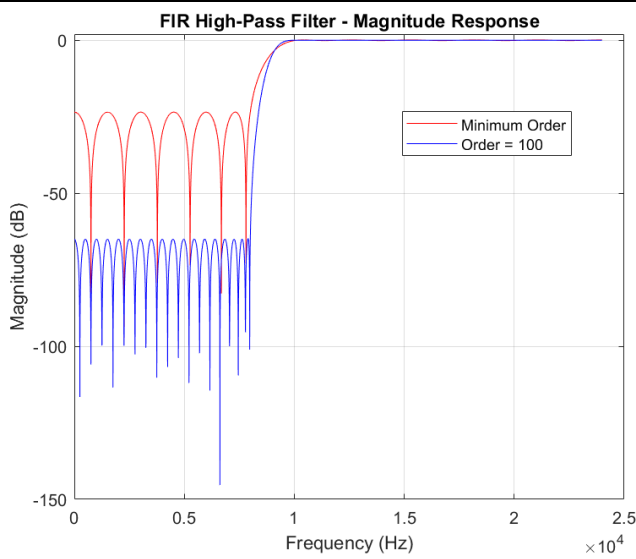
grid, producing smoother, more accurate magnitude, phase, and group-delay curves. This makes the filter behaviour easier to inspect and present, even though the underlying response remains unchanged.

Task 2

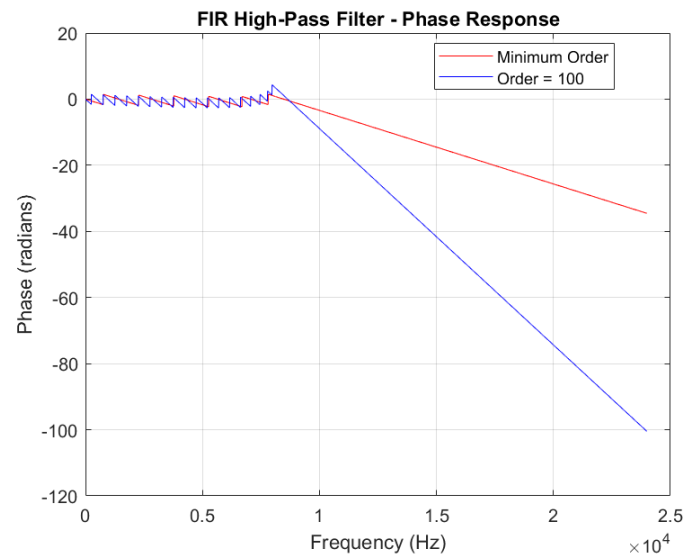
Design Process

In Task 2, a Finite Impulse Response (FIR) High-Pass Filter (HPF) was designed in MATLAB using the Parks–McClellan algorithm (`firpm`) with `firpmord` to estimate the minimum order. The sampling frequency was $f_s = 48$ kHz, with a stopband edge at $f_{\text{stop}} = 8$ kHz and a passband edge at $f_{\text{pass}} = 10$ kHz. Two main designs were created: a **minimum-order design** with $N = 34$, and a **fixed higher-order design** with $N = 100$. Both used a density factor (DF) of 50 and are shown in the plots labelled “FIR HPF – min order = 34 dens = 50” and “FIR HPF – N = 100 dens = 50”. To study the density factor separately, the $N = 100$ filter was re-designed with DF = 20 and DF = 50 and plotted together. All responses were generated using the `plot_responses` utility.

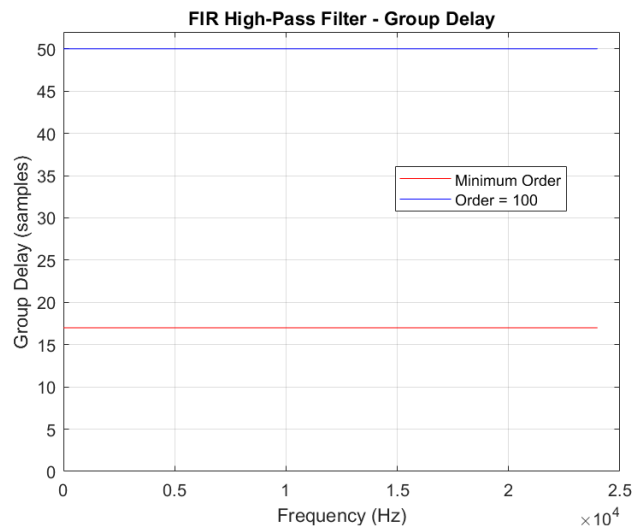
Effects of Different Orders on System Performance



(a)



(b)



(c)

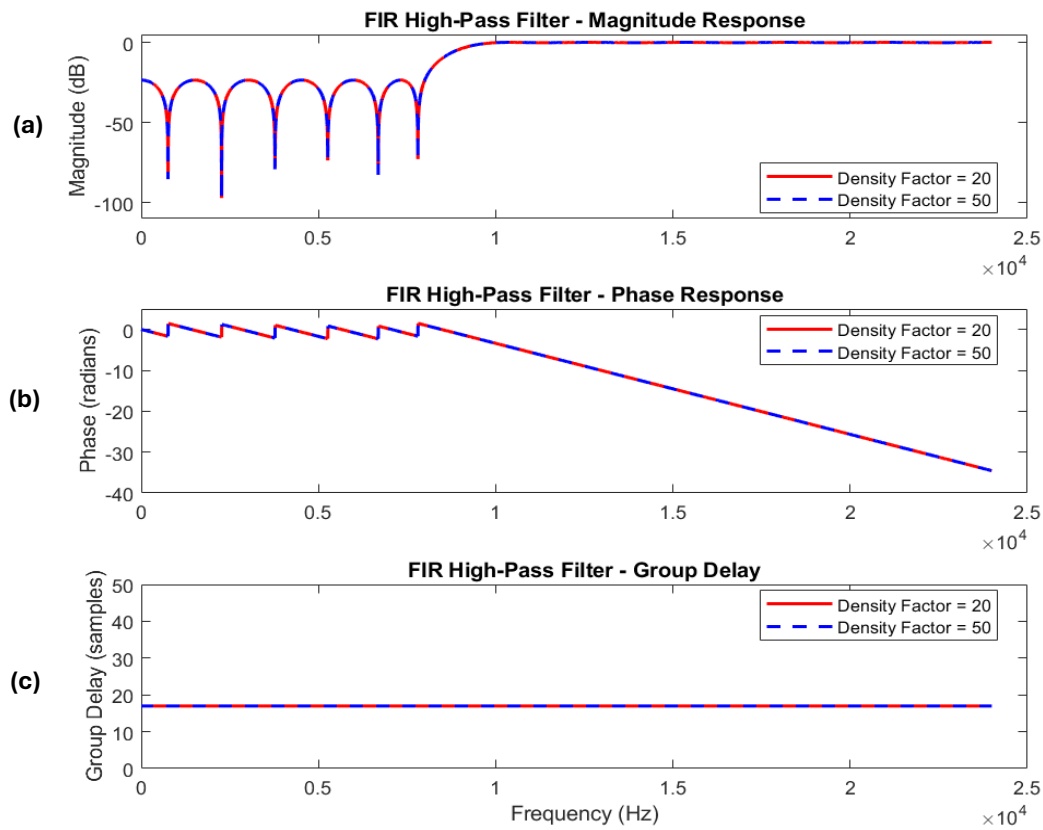
FIGS. 9(A)-(C): FIR HPF OF MAGNITUDE, PHASE, AND DELAY AT ORDER 100 AND MINIMUM

Observations: For the **minimum** order design, there is a visible attenuation in the stopband(0-8kHz) between -70 to -80 dB and a relatively wider transition from 8 kHz to 10 kHz. The higher order design provides a much deeper attenuation stopband (down to about -100 to -120 dB) and a noticeably sharper transition.

In both cases, the phase response is almost perfectly linear, but the total phase shift is much larger for $N = 100$. The group delay is constant at approximately 17 samples for $N = 34$ and 50 samples for $N = 100$, which, for linear-phase FIR filters, is consistent with $\tau_g = N/2$.

Implications and Conclusion: Stepping up the order from minimum to 100 greatly enhances the selectivity, meaning low-frequency components are strongly shrunk, and the passband is better isolated. In retrospect, there is a higher computational load and increased delay cost. The minimum-order design offers a low-delay, compact solution that meets basic requirements, while the $N = 100$ design is considered when a sharp cutoff and maximum stopband attenuation are more relevant than real-time responsiveness.

Effects of Different Density Factors on System Performance



FIGS. 10(A)-(B): FIR HPF OF MAGNITUDE, PHASE, AND DELAY AT DENSITY FACTORS 20 AND 50

Observations: For fixed $N = 100$, changing DF from 20 to 50 leaves the magnitude, phase, and group-delay responses essentially unchanged. Both designs show deep attenuation up to 8 kHz, a steep transition around 8–10 kHz, and a flat passband above 10 kHz. The phase responses overlap almost exactly, and the group delay remains a flat line at 50 samples. The only visible difference is that DF = 50 yields smoother, more finely sampled curves.

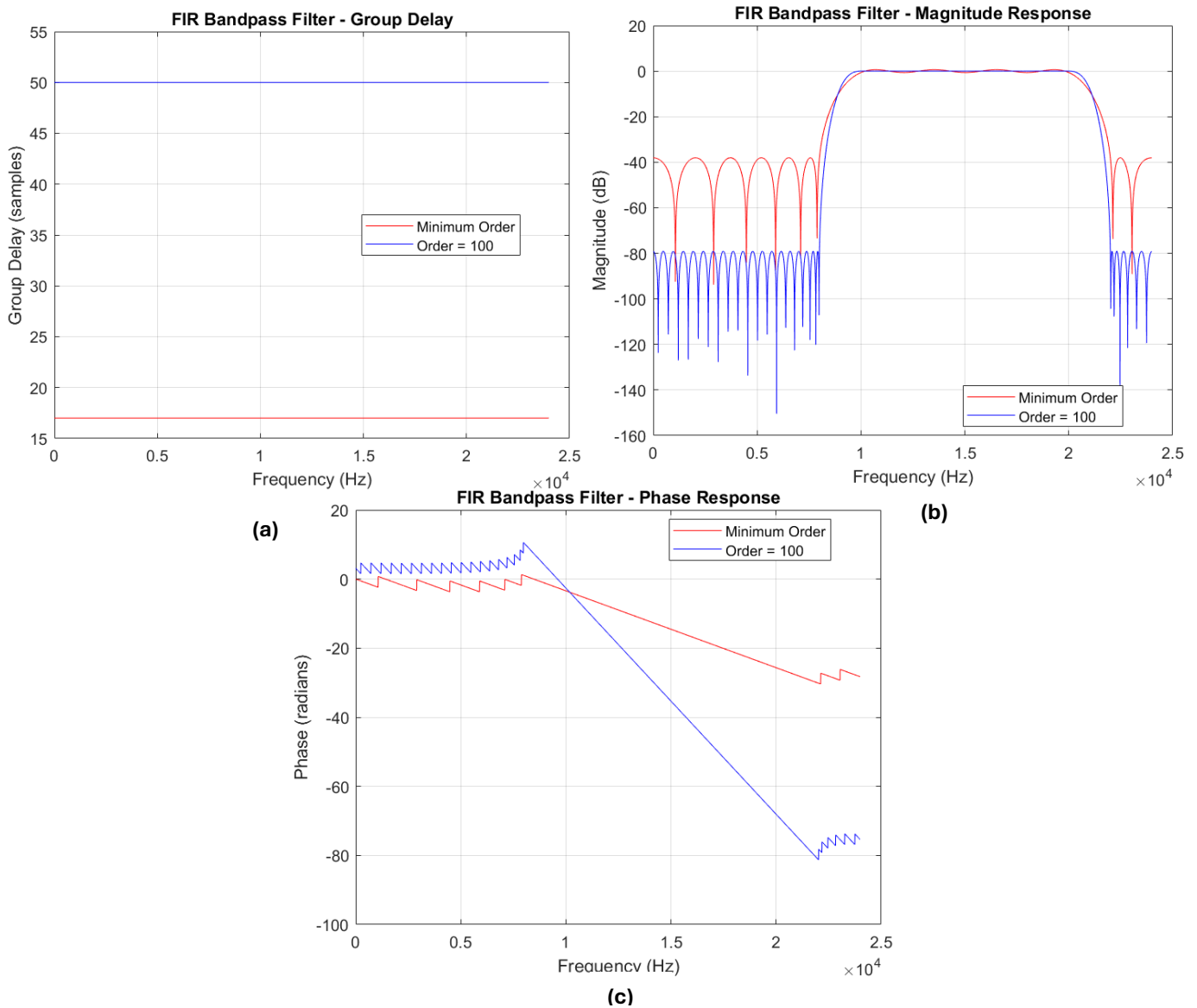
Implications and Conclusion: The density factor controls the frequency grid resolution used during design and plotting, thereby improving the smoothness and visual detail of the response, but it does not alter the filter's actual behaviour. Stopband attenuation, transition steepness, passband flatness, linear phase, and constant group delay are determined by the order, not by DF. Thus, DF affects graphical clarity and numerical precision, while the filter's real performance is governed by N .

Task 3

Design Process

In Task 3, a Finite Impulse Response (FIR) Band-Pass Filter (BPF) was designed in MATLAB using the Parks–McClellan algorithm (`firpm`) together with `firpmord` to estimate the minimum order. The sampling frequency was $f_s = 48$ kHz, with a passband between 10–20 kHz and stopbands below 8 kHz and above 22 kHz. Two main designs were created. First, a **minimum-order BPF** with $N \approx 34$, designed for density factors $DF = 20$ and $DF = 50$. Second, a **higher-order BPF** with fixed $N = 100$ and $DF = 50$ using the same band edges and tolerances. Additional plots compare the minimum order and $N = 100$ based on the magnitude, phase, and group delay characteristics.

Effects of Different Orders on System Performance

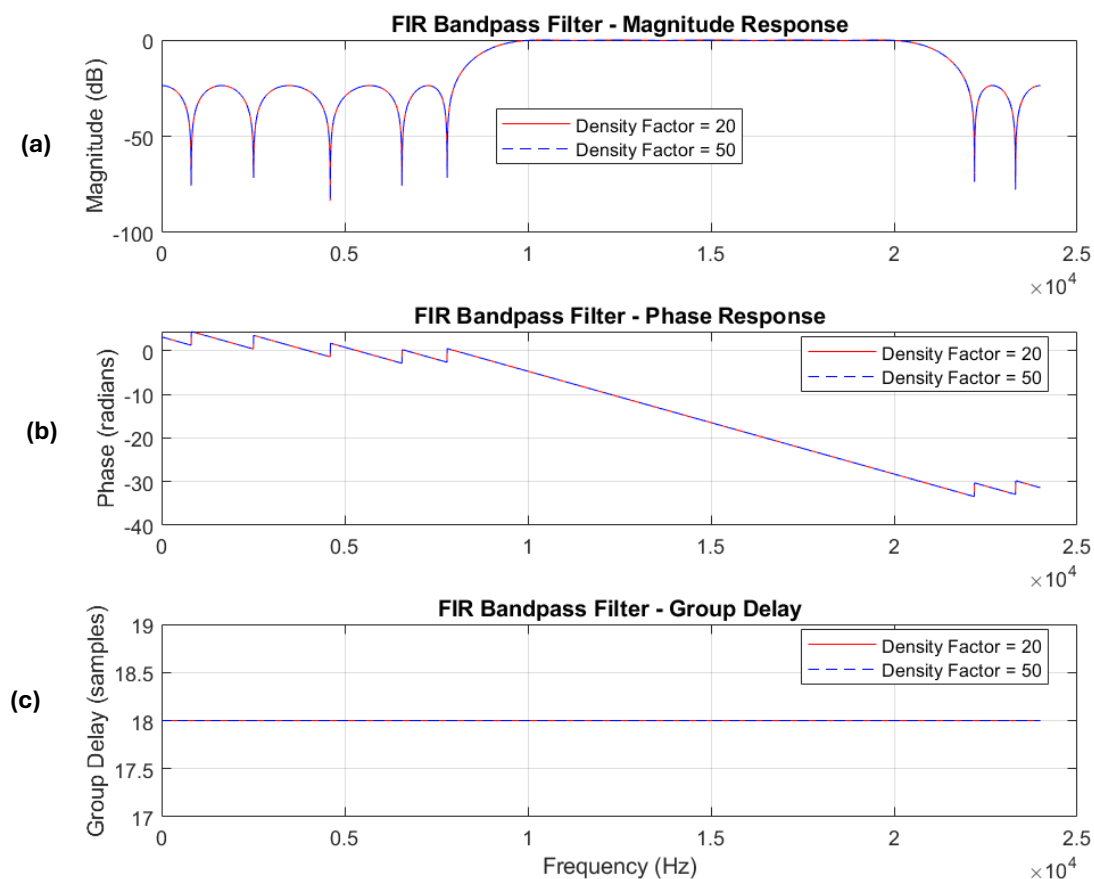


FIGS. 11(A)-(C): FIR BPF OF DELAY, MAGNITUDE, AND PHASE AT ORDER 100 AND MINIMUM

Observations: In this BPF minimum-order design, the stopbands of 0-8 kHz and 22-24 kHz are attenuated to approximately -70 to -80 dB, with broad transitions around 8–10 kHz and 20–22 kHz. The 10–20 kHz passband is close to flat, within a few dB of 0 dB, with modest ripple. For $N = 100$, stopband attenuation is much deeper (around -120 to -140 dB), with more, narrower lobes, and the transitions are much steeper, giving a more rectangular band-pass. The passband remains essentially flat, with tighter ripple control. In both cases, the phase response is almost linear. The 100 filter has a steeper slope and a larger total phase shift. Group delay is constant at about 17 samples for the minimum order, and about 50 samples for $N = 100$.

Implications and Conclusion: In the higher-order BPF, the 10-20 kHz band is better isolated, while frequencies below 8 kHz and above 22 kHz are strongly suppressed. Both filters preserve the shape of the waveform due to the linear-phase response, but the $N = 100$ filter a bigger fixed delay (50 to 17 samples). Hence, the higher-order filter is preferred when we require sharp band isolation and very high stopband rejection, and when extra delay is acceptable; the minimum-order filter is more suitable when lower latency and a more straightforward implementation are priorities. The minimum-order design is appropriate when a simple implementation and low latency are a priority.

Effects of Different Density Factors on System Performance



FIGS. 12(A)-(C): FIR BPF MAGNITUDE, PHASE, AND DELAY AT DENSITY FACTORS 20 AND 50

Observations: With a fixed filter order, changing the density factor from 20 to 50 has no technical impact on the design responses. They both yield the same bandpass shape, strong attenuation in both stopbands, and a near-flat passband of 0 dB. The density factor of 50 produces fine waveforms, whereas $df=20$ appears slightly rougher.

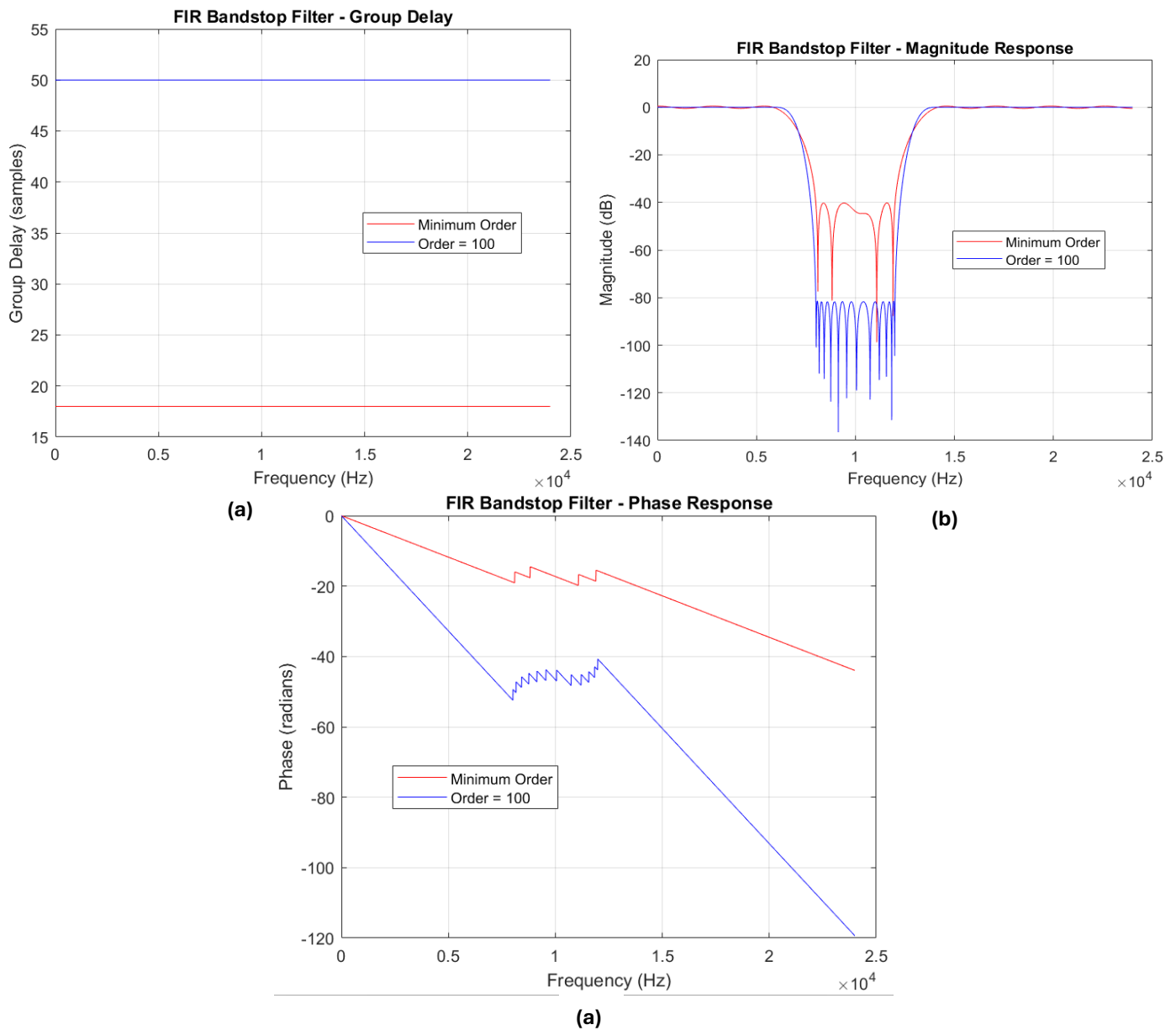
Implications and Conclusion: Density factor controls the resolution of the frequency grid used in `firpm` and in plotting, not the actual filter behaviour. A higher DF improves the visual smoothness and precision of the magnitude, phase, and group-delay plots, at the cost of slightly higher design time, but does not change stopband attenuation, passband shape, phase linearity, or delay. In summary, order N determines real performance; the density factor mainly affects graphical and numerical resolution.

Task 4

Design Process

In this task, a Finite Impulse Response (FIR) Band-Stop Filter (BSF) was designed in MATLAB using the Parks–McClellan algorithm (`firpm`) with `firpmord` to estimate the minimum order. The sampling frequency was $f_s = 48$ kHz, and the filter was required to reject frequencies between 8–12 kHz while passing signals below 6 kHz and above 14 kHz. First, a **minimum-order** filter (≈ 36 taps) was designed. The exact specification was then implemented with a **higher order**, $N = 100$ to investigate the effect of increasing filter length. Both designs were repeated with density factors $DF = 20$ and $DF = 50$ to realise the role of density in graphical smoothness. The magnitude, phase, and delay responses are produced using the plotting function, as in previous tasks.

Effects of Different Orders on System Performance

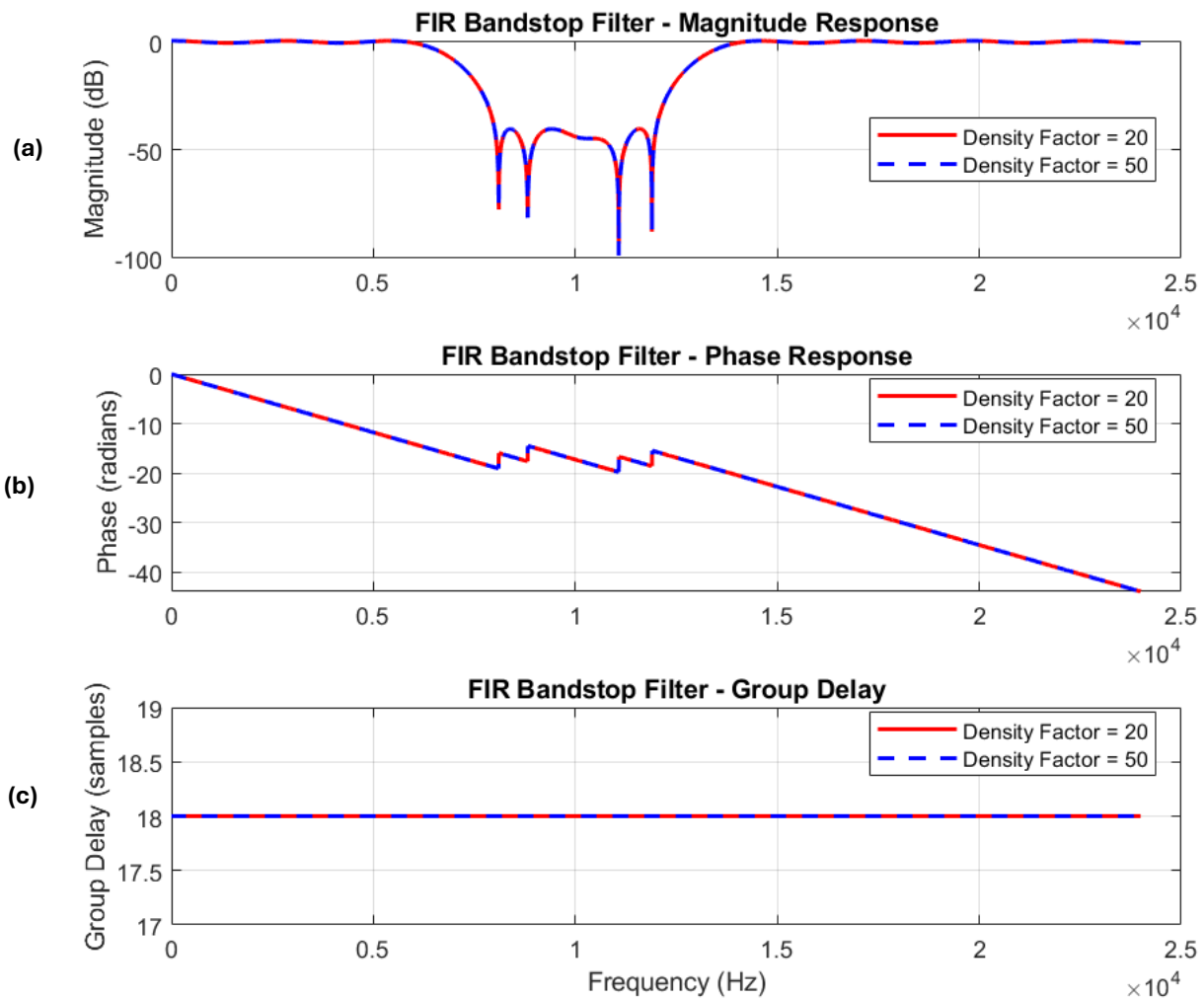


FIGS. 13(A)-(C): FIR BPF DELAY, MAGNITUDE, AND PHASE RESPONSE AT ORDER 100 AND MINIMUM.

Observations: Considering the minimum-order design, the 8-12 kHz region, which is the stopband attenuation, reaches between -60 dB and -80 dB, with relatively broad transitions around 6–8 kHz and 12–14 kHz. The passbands remain close to 0 dB with modest ripple. With $N = 100$, the notch becomes much deeper (around -110 to -130 dB), with more and sharper lobes in the stopband, and the transitions at both edges are significantly steeper. The passbands are also flatter with reduced ripple. In these two cases, the phase response is almost linear. The higher-order is characterised by a much steeper slope and a significant phase shift. There is a constant group delay of about 18 samples for the minimum-order and 50 samples for *the* $N = 100$ Filter, which is in line with $\tau_g = N/2$.

Implications and Conclusion: Increasing the order greatly improves stopband selectivity and notch sharpness, making the BSF much better in attenuating the 8-12 kHz band. Meanwhile, this comes at the cost of greater delay and higher computational complexity. The minimum-order design is efficient computationally with broad transitions and moderate attenuation, while the higher-order is rather considered when strong rejection of the stopband is prioritized over latency.

Effects of Different Density Factors on System Performance



FIGS. 14(A)-(C): FIR BPF OF MAGNITUDE, PHASE, AND DELAY AT DENSITY FACTORS 20 AND 50

Observations: For density factors at 20 and 50, the magnitude responses are essentially identical in terms of cutoff location, attenuation level, and transition width. $DF = 50$ produces smoother, more finely detailed curves, whereas $DF = 20$ appears slightly coarser. The phase responses overlap almost exactly, both retaining linear-phase characteristics, and the group delay remains constant for a given order regardless of DF .

Implications and Conclusion: The Density factor affects only the resolution and smoothness of the design and plotted responses, not the underlying filter behaviour. $DF = 50$ provides clearer, more publication-ready graphs, while $DF = 20$ is computationally lighter but still yields the same frequency characteristics. Attenuation, phase linearity, and group delay are determined by the filter order, not by the density factor.

Task 5

Design Process (MATLAB Code and Filter Designer App)

MATLAB Code

The IIR band-pass filter was designed using an **elliptic prototype**. The edge frequencies were normalised by the Nyquist frequency (24 kHz):

$\Omega_{\text{stop1}} = 4000/24000$, $\Omega_{\text{pass1}} = 6000/24000$, $\Omega_{\text{pass2}} = 20000/24000$, $\Omega_{\text{stop2}} = 22000/24000$. With passband ripple $A_p \approx 1$ dB and stopband attenuation $A_s \approx 80$ dB, `ellipord` was used to compute the **minimum order** and critical band edges. Then `ellip` generated the digital IIR BPF:

$$[b_{\min}, a_{\min}] = \text{ellip}(N_{\min}, A_p, A_s, W_n, 'bandpass').$$

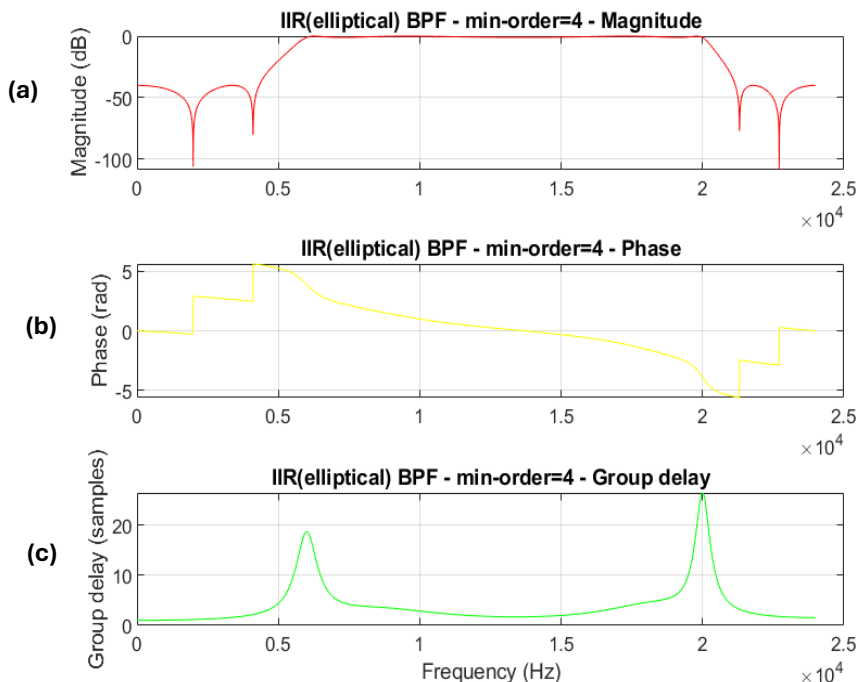
When the order was forced to $N = 100$ MATLAB reported that it **could not safely design** the filter due to numerical issues, and no valid coefficients or plots were generated.

Filter Designer App

The same specification was implemented in the Filter Designer App (Response Type: Bandpass, Method: IIR–Elliptic, $F_s = 48$ kHz, $F_{\text{stop1}} = 4$ kHz, $F_{\text{pass1}} = 6$ kHz, $F_{\text{pass2}} = 20$ kHz, $F_{\text{stop2}} = 22$ kHz, $A_{\text{pass}} = 1$ dB, $A_{\text{stop}} = 80$ dB).

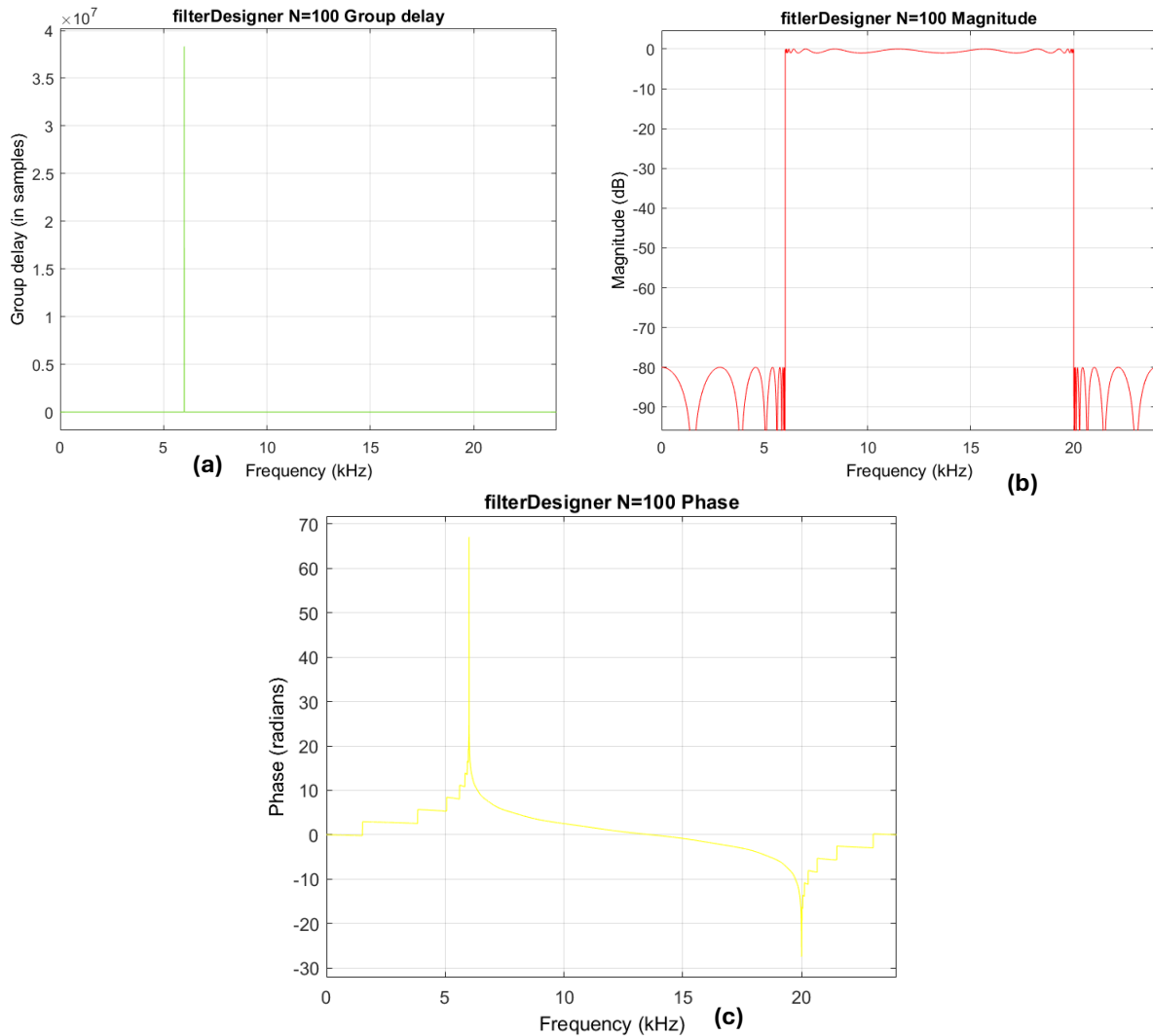
For the minimum-order, the filterDesigner produces responses that same as that of the MATLAB code. The order is later set to 100 and the Designer generated the required waveforms but MATLAB code produces something different

Effects of Changing Order on Performance



FIGS. 15(A)-(C): IIR BPF MAGNITUDE, PHASE, AND DELAY WAVEFORMS AT MINIMUM ORDER

This elliptic BPF of **minimum order** displays sharp transitions at 4 to 6 kHz and 20 to 22 kHz. Stopbands below 4 kHz and above 22 kHz are attenuated to about -80 to -100 dB, while the 6–20 kHz passband stays near 0 dB with small ripple. The phase is strongly nonlinear with wrapping, and the group delay is frequency-dependent, with peaks (about 20–25 samples) near the passband edges. This strengthens its selectivity at a low computational cost, but introduces some phase and delay distortion.



FIGS. 16(A)-(C): IIR BPF MAGNITUDE, PHASE, AND DELAY WAVEFORMS AT ORDER 100 (FILTERDESIGNER)

When implementing the MATLAB code, **forcing $N = 100$** fails because these high-order elliptic IIR filters become numerically ill-conditioned: poles coalesce near the unit circle, making the denominator highly sensitive and potentially unstable. The App, using the SOS implementation, can realise an order-100 filter, but its behaviour is impractical. The magnitude improves only marginally over the minimum-order case. At the same time, the phase becomes highly distorted, and the group delay exhibits huge spikes (tens of millions of samples), indicating extreme, unusable delay at specific frequencies.

Task 5: Could not safely design order 100 elliptic IIR due to numerical issues: The

Usually, IIR filters work well at low orders; very high orders are unstable

FIG. 17: MATLAB CODE ERROR MESSAGE IN ORDER 100 IIR DESIGN

Conclusion: The minimum-order elliptic BPF already meets the design requirements with low complexity, at the price of non-linear phase and moderate group-delay variation. Pushing the order to 100 leads to numerical problems in MATLAB and to extreme phase and delay behaviour in the App, with little gain in magnitude response. In practice, small minimum orders are preferred, and very high-order elliptic IIR BPFs should be avoided.

Effects of Changing Density Factors

The screenshot shows the Filter Designer app interface with the following specifications:

- Response Type:** Bandpass (selected)
- Design Method:** IIR Elliptic (selected)
- Filter Order:** Specify order: 100
- Frequency Specifications:**
 - Units: Hz
 - Fs: 48000
 - Fpass1: 6000
 - Fpass2: 14000
- Magnitude Specifications:**
 - Units: dB
 - Astop: 80
 - Apass: 1

A "Design Filter" button is located at the bottom right of the interface.

FIG. 18: FILTERDESIGNER SPECIFICATIONS FOR ORDER 100 IIR FILTER DESIGN

Observation: For this IIR design, there is no density factor option in `ellip/ellipord` or in the Filter Designer App. The density factor used in earlier FIR tasks belongs to the Parks–McClellan equiripple algorithm (`firpmord`, `firpm`), where a discrete frequency grid is used. ω_k is sampled more densely as density increases. Elliptic IIR filters, however, are not obtained by frequency-grid optimisation. They are derived from an analog elliptic prototype $H_a(s)$, transformed to band-pass form and mapped to the z-plane using the bilinear transform. The poles and zeros are determined analytically; there is no parameter corresponding to the density factor, and changing it would not affect. $H(z)$.

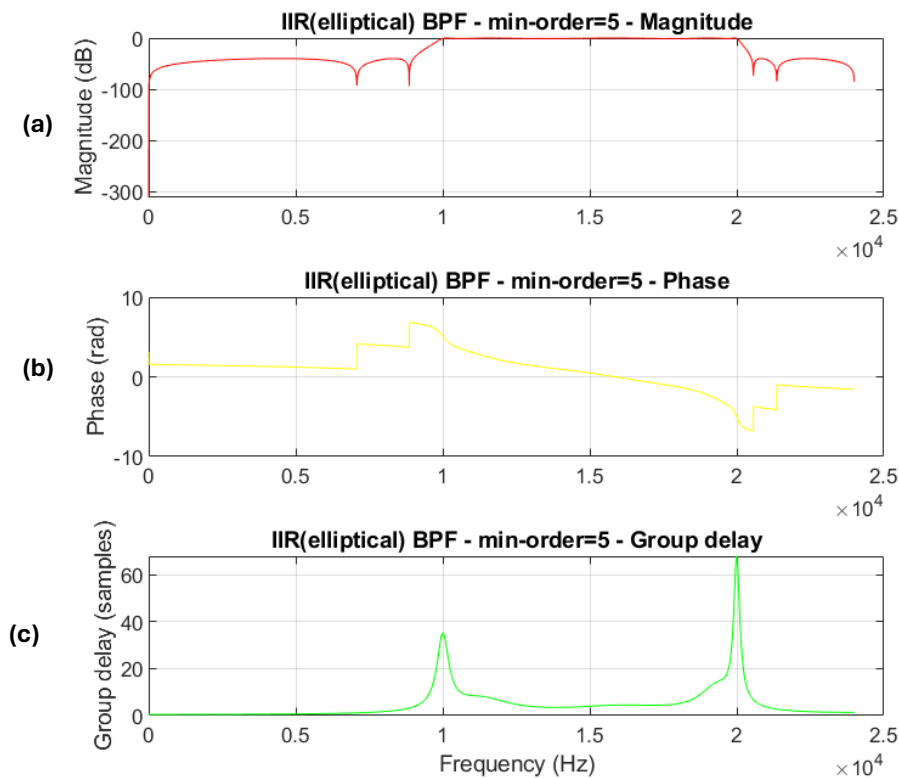
Conclusion: The Density factor is only meaningful for FIR equiripple designs and does **not** apply to IIR elliptic BPFs. Consequently, there is no change in magnitude, phase, or group delay when “changing” density factor in this task, because it is not a valid design parameter for IIR filters.

Task 6

Design Process (Code and App)

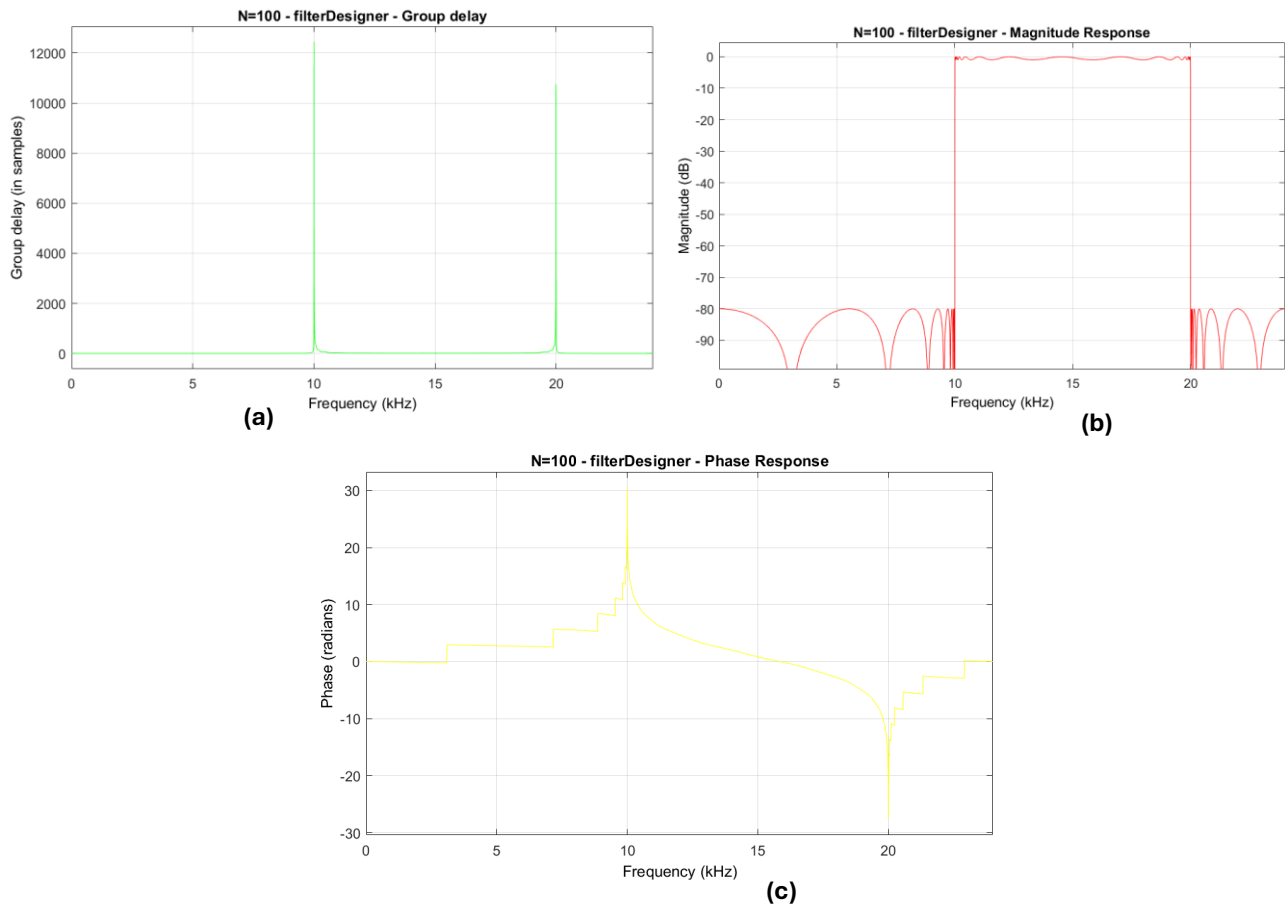
For this task, we designed an IIR elliptic bandpass filter with a passband between 10-20 kHz and with stopbands above 22 kHz and below 8 kHz at *sampling frequency*, 48 kHz. MATLAB code was used to determine minimum order and obtain the characteristic responses. In parallel, the filterDesigner App was applied with the exact specifications (bandpass, IIR-elliptic, SOS structure) to verify the design and to provide characteristic response plots.

Effects of Changing Orders on Performance



FIGS. 19(A)-(C): IIR BPF MAGNITUDE, PHASE AND DELAY WAVEFORMS AT MINIMUM ORDER

At **minimum order**, the elliptic BPF shows intense attenuation in the stopbands, with the passband (10-20 kHz) near 0 dB and sharp transitions. The phase is visibly non-linear, with sudden jumps; the waveform shape is not preserved compared to that of FIR filters. Group delay is frequency-dependent, with peaks near 10 kHz and 20 kHz (from a few samples up to 60), showing poles close to the unit circle. The Filter Designer App reproduces the same behaviour, confirming that minimum-order elliptic designs are efficient and stable but inherently non-linear in phase with variable delay. These are well-suited to frequency-selective tasks (e.g., communications) but not ideal for applications that require time-domain waveform fidelity.



FIGS. 20(A)-(C): IIR BPF DELAY, MAGNITUDE, AND PHASE WAVEFORMS AT ORDER 100 (FILTERDESIGNER)

```

16
17 % Attempt N = 100 as in task description
18 try
19     [b_iir2_100, a_iir2_100] = ellip(N5, Rp, Rs, Wn2, 'bandpass');
20     plot_responses(b_iir2_100, a_iir2_100, fs, nfft, sprintf('IIR(elliptical) BPF - min-order=%d', N5));
21 catch ME
22     fprintf('Task 6: Could not safely design order %d elliptic IIR: %s\n', N5, ME.message);
23 end

```

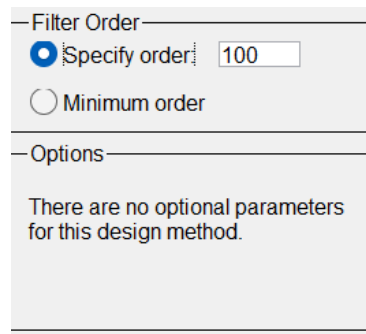
Task 6: Could not safely design order 100 elliptic IIR: The filter order is too large. Use a smaller value.
Use caution: high-order IIRs are rarely necessary.

FIG. 21: MATLAB CODE ERROR MESSAGE OF ORDER 100 IIR FILTER DESIGN

When the order was forced to $N = 100$ MATLAB reported that the filter “could not be safely designed” because the order is too large: many poles crowd near the unit circle, coefficients become ill-conditioned, and the filter risks instability. The Filter Designer App can realize an order-100 elliptic BPF using SOS and scaling, but the result is not practically usable. The magnitude response shows only a minor improvement over the minimum-order case. At the same time, the phase becomes highly irregular, and the group delay exhibits large spikes (on the order of 10,000–12,000 samples) near the band edges. This extreme and uneven delay makes the filter unsuitable for real systems.

Conclusion (Order Effects): The elliptic IIR BPF of minimum order is efficient and stable. It provides very sharp transitions with typical group-delay peaks. In contrast, the 10-order design is numerically unstable in MATLAB and produces a non-uniform phase and excessive delay in the App, with no meaningful gain in the magnitude response. High-order elliptic IIR designs should therefore be avoided; minimum-order designs are preferred.

Effects of Changing Density Factors



The image shows a MATLAB Filter Designer dialog box. It has two main sections: 'Filter Order' and 'Options'. In the 'Filter Order' section, there are two radio buttons: 'Specify order' (which is selected) and 'Minimum order'. Next to 'Specify order' is a text box containing the value '100'. The 'Options' section below it contains the text: 'There are no optional parameters for this design method.'

FIG. 22: NO DENSITY FACTOR SPECIFICATION IN IIR DESIGN

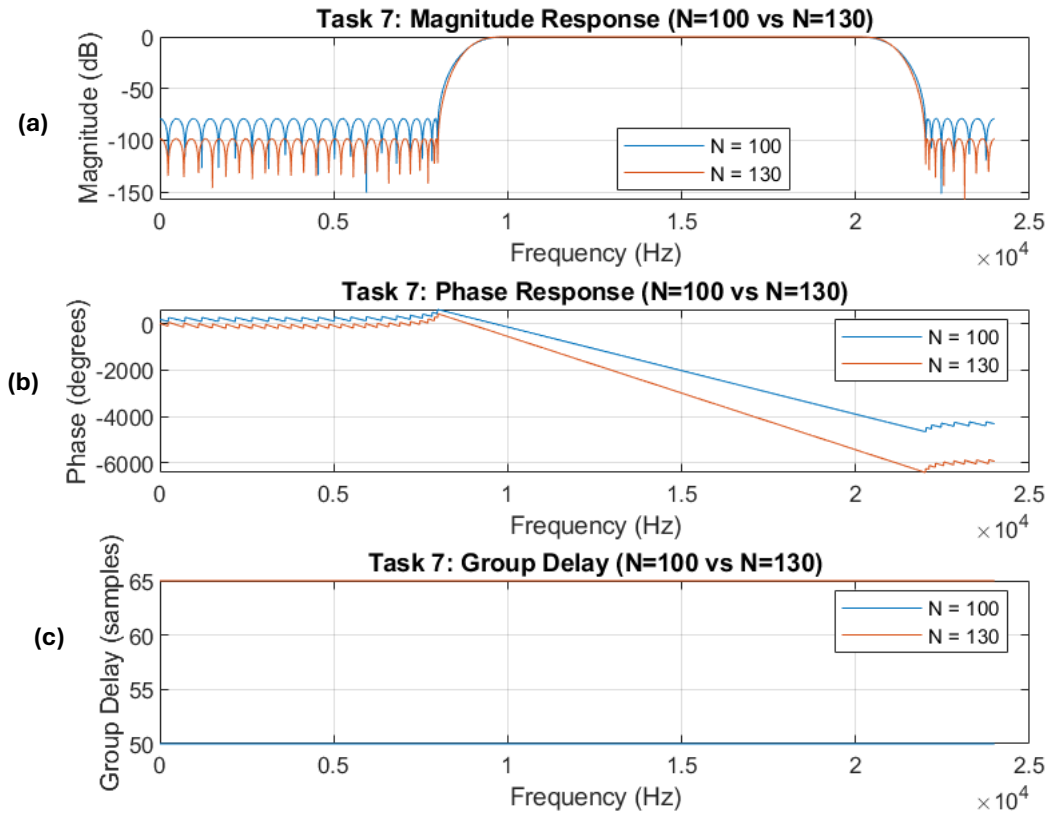
The density factor cannot be applied to IIR filters. It is specific to FIR equiripple filters designed with `firpmord` and `firpm` (Parks–McClellan), in which the frequency grid is sampled more densely as `df` increases. Elliptic IIR filters are instead derived from analog prototypes and mapped to the z -plane; their transfer function, $H(z)$ does not depend on a frequency-sampling grid. Consequently, MATLAB and Filter Designer neither accept nor expose a density-factor option for IIR designs, and changing `df` does not affect magnitude, phase or delay. In summary, the density factor is meaningful only for FIR equiripple filters and cannot be used or analysed for elliptic IIR BPFs.

Task 7

Characteristic Comparisons at $N=100$ and $N=130$ (FIR Filter)

In Task 7, two FIR equiripple band-pass filters with identical specifications were compared using the Parks–McClellan algorithm (`firpm`): sampling frequency $f_s = 48$ kHz, a passband of 10–20 kHz, and stopbands below 8 kHz and above 22 kHz. Two filters were designed: one of order $N = 100$ yielding coefficients b_{100} , and another of order 130, giving b_{130} . The magnitude responses displayed shows that both filters behave as we expect, each with an equiripple passband around 0 dB. The passband, between 10–20 kHz is identical for both designs, proving that increasing the order does not essentially improve the flatness of the passband once a proper equiripple solution has been reached.

Differences arise mainly in the stopband and transition regions: the $N = 100$ filter achieves roughly -80 dB attenuation, while the $N = 130$ filter reaches around -100 dB with steeper roll-offs at 8–10 kHz and 20–22 kHz, demonstrating the classical benefit of additional taps—reduced approximation error and deeper rejection.



FIGS. 23(A)-(C): FIR BPF OF MAGNITUDE, PHASE, AND DELAY AT ORDERS 100 AND 130

However, the phase and group-delay plots reveal essential consequences of this increased filter length. Because both filters are linear-phase FIR designs, their phase responses decrease linearly with frequency, but the slope is steeper for $N = 130$, indicating a greater inherent delay. The group-delay curves quantify this: a linear-phase FIR of order N has a constant group delay of $N/2$ samples. Accordingly, the *order 100* filter shows a delay of about 50 samples while *order 130* shows about 65 samples. The phase plots supports this, the unwrapped of the *order 130* filter is proportionally more negative, showing the extra delay the filtered signal output is aligned with the input. This increased delay does not alter the waveform due to the linear phase, but it does introduce latency. This latency is not relevant in offline processing but problematic in real-time systems.

From a computational standpoint, the increase from 100 to 130 taps implies a proportional rise in per-sample processing cost—30 additional multiplications and additions per output sample, roughly a 30% overhead. Thus, while the $N = 130$ filter provides deeper stopbands and slightly sharper transitions, the improvements are incremental compared to the cost: more phase delay, more memory, and more operations per sample. Task 7, therefore, demonstrates that while higher FIR order improves magnitude performance, it also increases latency and computational demands, underscoring the balance designers must achieve between filtering performance and implementation efficiency.

Task 8

When comparing FIR and IIR filters in terms of efficiency and cost, the key question is what is being optimised. In general, **IIR filters are more efficient** for meeting a given magnitude response (specified passband ripple, stopband attenuation, and transition width), because they require far fewer coefficients than FIR filters. In the previous tasks, our FIR filters used orders like 36, 50, 100, and also 130 to achieve steep transitions and 60-100 dB attenuation. In contrast, elliptic IIR filters attained similar or better attenuation with orders of only 4 or 5.

Since each coefficient corresponds to a multiply–accumulate operation and a stored delay element, an FIR of order 100 requires about 101 multiplications per sample. In contrast, an IIR of order 4 needs far fewer. This means that, for the same selectivity, an IIR filter uses less CPU time and memory and introduces less delay, which is why IIR is often preferred in real-time, resource-limited systems such as embedded DSP, radio receivers, and control applications.

FIR filters are therefore more “expensive” computationally, because they typically need a much higher order to match IIR stopband attenuation and transition steepness. This translates into more operations per sample, more memory for delay lines, and higher power consumption. However, FIR filters are “cheaper” and safer from a design perspective. These FIR filters can be generated with the exact linear phase, so they do not distort waveform shapes—an essential advantage in audio, data communications, and pulse-shaping applications where phase distortion is critical.

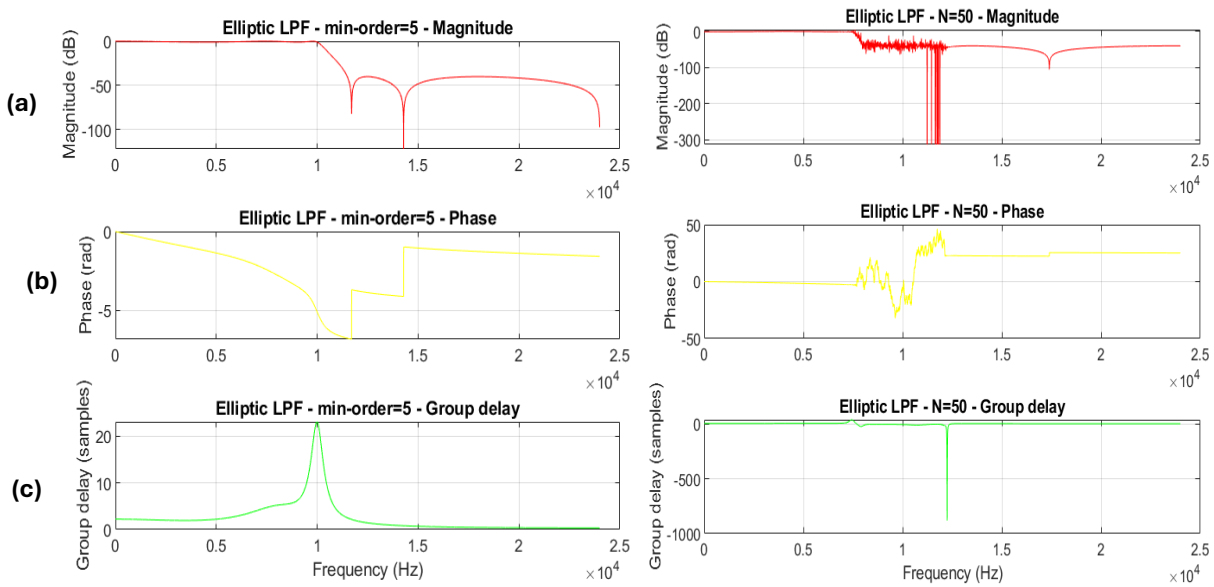
This translates into many operations per sample, delay lines with more memory, and very high power consumption. However, FIR filters are pretty 'cheap' and safer from the perspective of design. These filters are generated with the same linear phase, so waveform shapes are not distorted, an essential advantage in pulse-shaping applications where phase distortion is important.

Moreover, FIR filters with real, symmetric coefficients are always stable, regardless of order, so there is no need to worry about poles leaving the unit circle or numerical instability. In summary, IIR filters are usually more efficient in implementation because they achieve the same magnitude response with a much lower order. Still, they are more demanding to design correctly due to issues of stability, non-linear phase, and numerical sensitivity. FIR filters are computationally heavier but offer linear phase, guaranteed stability, and highly predictable behaviour, making them the safer choice when these properties outweigh the cost of extra computation.

Task 9

Design Process

In this task, an elliptic IIR low-pass filter (LPF) is designed to pass frequencies up to about 10 kHz (the useful band from 300 Hz) and attenuate those above it. The sampling frequency is $f_s = 48\text{kHz}$, so the Nyquist frequency is 24 kHz. In MATLAB, the design is specified using normalised edge frequencies and a passband ripple of about 1 dB with a stopband attenuation of about 80 dB. `ellipord` is first used to estimate the minimum order that satisfies these specifications, which returns $N \approx 5$. `ellip` then generates the corresponding IIR coefficients. The resulting magnitude, phase, and group-delay plots show a flat passband up to about 10 kHz and a sharp transition into the stopband. The exact requirement is implemented in the App. Contrasting the **minimum-order** to the **higher-order design** ($N = 50$) show their behaviour as the order increases.

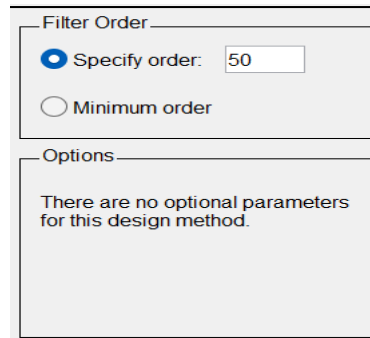


FIGS. 24(A)-(C): ELLIPTIC LPF MAGNITUDE, PHASE, AND DELAY AT ORDERS 50 AND MINIMUM

Observations: At minimum, the magnitude passband stays close to 0 dB, with few ripples, and there is a sharp drop of the stopband from -80 to -100 dB. There is also smooth decrease in phase with one wrap, while the group delay usually flat and low with 20-22 sample peak near the cutoff. When there is an order increase of 50, the magnitude develops very deep

narrow notches (down to about -300 dB), and the passband near the cutoff becomes heavily rippled. The phase response becomes highly irregular around the transition band, and the group-delay plot exhibits a significant negative spike of several hundred samples above the cutoff region. This indicates a numerically ill-conditioned design with zeroes and poles clustered near the unit circle.

Conclusion: The results show that this minimum-order design already meets the design specifications with a manageable delay and low complexity. The 50-order filter does not deliver a similar benefit; the extra stopband depth is rarely needed and is achieved at the cost of severe phase distortion and pathological group-delay behaviour, with a real risk of instability under coefficient quantisation. Thus, minimum-order elliptic designs are both sufficient and safer.



Filter Order

☒ Specify order: 50

☐ Minimum order

Options

There are no optional parameters for this design method.

FIGURE 25: NO DENSITY FACTOR SPECIFICATION

Lastly, though this task mentions density factors of 20 and 50, they **do not apply to elliptic IIR filters**. It is a Parks–McClellan FIR parameter and does not appear in the elliptic design equations, which are based on analog prototypes and bilinear transforms. The density factor is normally ignored for LPF IN MATLAB and Filter Designer, changing it does not have an effect on the magnitude, phase or group delay. The only essential design trade-offs for this filter are passband ripple, stopband attenuation, and order, not the density factor.

REFERENCES

- Abood, S.I., 2020.** *Digital signal processing: a primer with MATLAB*. Aalborg: River Publishers.
- Chakraborty, S. and Krishnan, S., 2023.** A review of methods for coding of speech signals. *Journal of Signal Processing Systems*. [online] Available at: Publisher platform (Accessed: 1 December 2025).
- Chhetri, G.D., Tiwari, N. and Adhikary, B.B., 2025.** Impacts of vocoder selection on Tacotron-based Nepali text-to-speech. *SN Computer Science*, 6(1), Article 12.
- Cychosz, M., Winn, M.B. and Goupell, M.J., 2024.** How to vocode: using channel vocoders for cochlear-implant research. *Journal of the Acoustical Society of America*, 155(1), pp. 465–487.
- Hayes, B. and Engel, J., 2024.** Differentiable digital signal processing for music and speech generation: a review. *Neural Computing and Applications*. [online] Available at: Publisher platform (Accessed: 1 December 2025).
- MathWorks, 2024a.** *firpm – Parks–McClellan optimal FIR filter design*. Signal Processing Toolbox documentation. Natick, MA: MathWorks. [online] Available at: MathWorks Help Center (Accessed: 1 December 2025).
- MathWorks, 2024b.** *ellipord and ellip – elliptic IIR filter order and design*. Signal Processing Toolbox documentation. Natick, MA: MathWorks. [online] Available at: MathWorks Help Center (Accessed: 1 December 2025).
- Mehrish, A., Raza, H. and Tyagi, V., 2023.** A review of deep learning techniques for speech processing. *Information Fusion*, 91, pp. 316–348.
- Mushtaq, F. et al., 2021.** Investigating cortical responses to noise-vocoded speech in children with normal hearing using functional near-infrared spectroscopy. *Brain Sciences*, 11(4), p. 432.
- Sarpal, S., 2020.** Difference between IIR and FIR filters: a practical design guide. *ASN Filter Designer Blog*, 28 April. [online] Available at: ASN Filter Designer website (Accessed: 1 December 2025).
- Splice, 2025.** *What is a vocoder? How to use vocoders in your music*. Splice Blog. [online] Available at: Splice website (Accessed: 1 December 2025).
- Takahashi, N., 2019.** Fast Fourier transform algorithms for parallel computers. In: *High-performance computing for signal and image processing*. Cham: Springer.

APPENDIX

Vocode Design Code

```
1 % ***** Load test sound:
2 [y, Fs] = audioread('bkbe2114.wav');
3 disp('Press Enter for original sound');
4 pause
5 soundsc(y, Fs);
6 % Create time vector representation of y
7 t = (0:length(y)-1)/Fs;
8
9 % ***** Plot the original sound wave:
10 figure;
11 subplot(2,4,1)
12 plot(t, y, 'b');
13 title('Original Sound Clip (time)');
14 ylabel('16-bit data');
15 xlabel('Time, s')
16 % ***** Calculate the FFT spectrum of the original sound:
17 y_fft_raw = abs(fft(y));
18 max_y_fft = max(y_fft_raw(:));
19 if max_y_fft > 0
20     y_fft = y_fft_raw / max_y_fft;
21 else
22     y_fft = y_fft_raw;
23 end
24 % Normalize with small epsilon to avoid division by zero
25 Faxis = linspace(0, Fs, length(y)); % Frequency axis array
26 % ***** Plot the spectrum of the original sound:
27 subplot(2,4,5)
28 plot(Faxis, y_fft, 'b')
29 title('Original Sound Clip (Freq)');
30 xlabel('Frequency, Hz')
31 ylabel('Normalized FFT Power')
32 axis([0 12000 0 1.1]); % [x_start x_end y_min y_max]
33 % ***** Initialize output variable based on the size of y:
34 if size(y, 2) > 1
35     y_voc = zeros(size(y)); % Stereo output if y is stereo
36 else
37     y_voc = zeros(size(y, 1), 1); % Mono output if y is mono
38 end
39 % ***** Generate white noise for multiplication:
40 wnoise = randn(size(y));
41 max_w = max(abs(wnoise(:)));
42 if max_w > 0
43     wnoise = wnoise / max_w;
44 end
45 % Normalize with small epsilon to avoid division by zero
46 % ***** Choose number of channels:
47 N = 12;
48 % ***** Calculate the cutoff frequencies:
49 f = [0, 69.53, 168.28, 308.55, 507.79, 790.77, 1192.71,...
50     1763.61, 2574.51, 3726.27, 5362.19, 7685.79, 11000]; % in Hz
51 % ***** Normalize the cutoff frequencies:
52 w = f / (Fs / 2);
53
54 % ***** Plot original signal and its spectrum in a separate figure:
55 figure;
56 subplot(2,1,1)
57 plot(t, y, 'b');
58 title('Original Sound Clip');
59 ylabel('Amplitude');
60 xlabel('Time, s');
61
62 subplot(2,1,2)
63 plot(Faxis, y_fft, 'b');
64 title('Original Sound Spectrum');
65 xlabel('Frequency, Hz');
66 ylabel('Normalized FFT Power');
67 xlim([0 12000]);
68
69
70
71 % ***** STAGE 1 starts here:
72 for k = 1:N
73 % ***** STEP 0: BANDPASS FILTERING through filter bank:
74 Rp = 3; % Passband Ripple
75 Rs = 60; % Stopband Attenuation
76 if k == 1 % First channel uses a low-pass filter
77 [b, a] = ellip(7, Rp, Rs, w(2), 'low');
78 else % Subsequent channels use band-pass filters
79 [b, a] = ellip(5, Rp, Rs, [w(k) w(k+1)]);
80 end
81 y_bp = filter(b, a, y); % Bandpass filter output
82 % ***** Plot the bandpass filtered signal:
83 figure(1)
84 subplot(2,4,2)
85 plot(t, y_bp, 'b');
86 title(['Bandpass Filtered Output (Channel ', num2str(k), ')']);
87 ylabel('16-bit data');
88 xlabel('Time, s')
89 % ***** Calculate the spectrum of bandpass filtered signal:
90 yf_fft_raw = abs(fft(y_bp));
91 max_yf_fft = max(yf_fft_raw(:));
92 if max_yf_fft > 0
93     yf_fft = yf_fft_raw / max_yf_fft;
94 else
95     yf_fft = yf_fft_raw;
96 end
97 % ***** Plot the spectrum:
98 subplot(2,4,6)
99 plot(Faxis, yf_fft, 'b')
100 title('Bandpass Filtered Spectrum')
101 xlabel('Frequency, Hz')
102 ylabel('Normalized FFT Power')
103 axis([0 12000 0 1.1]);
104 % ***** STEP 2: Half-wave Rectification:
105 yr = max(y_bp, 0);
106 max_yr = max(abs(yr(:)));
107 if max_yr > 0
108     yr = yr / max_yr;
109 end
110 % Normalize
111 % ***** Plot the rectified sound wave:
112 subplot(2,4,3)
113 plot(t, yr, 'r');
114 title('Rectified Signal (time)');
115 ylabel('16-bit data');
116 xlabel('Time, s')
```

```

117 % ***** Calculate the spectrum of rectified signal:
118 yr_fft_raw = abs(fft(yr));
119 max_yr_fft = max(yr_fft_raw(:));
120 if max_yr_fft > 0
121     yr_fft = yr_fft_raw / max_yr_fft;
122 else
123     yr_fft = yr_fft_raw;
124 end
125
126 % ***** Plot the spectrum of the rectified signal:
127 subplot(2,4,7)
128 plot(Faxis, yr_fft, 'b')
129 title('Rectified Spectrum')
130 xlabel('Frequency, Hz')
131 ylabel('Normalized FFT Power')
132 axis([0 12000 0 1.1]);
133
134 % ***** STEP 3: Low-Pass Filtering (160 Hz) to Extract Envelope:
135 [bb, aa] = butter(3, 160/(Fs/2), 'low');
136 yf = filter(bb, aa, yr);
137 max_yf = max(abs(yf(:)));
138 if max_yf > 0
139     yf = yf / max_yf;
140 end
141 % Normalize
142 % ***** Plot the low-pass filtered signal (envelope):
143 subplot(2,4,4)
144 plot(t, yf, 'b');
145 title('Low-Pass Filtered Envelope (time)');
146 ylabel('16-bit data');
147 xlabel('Time, s')
148
149 % ***** STEP 4: Multiply Envelope with White Noise:
150 if length(yf) ~= length(wnoise)
151     wnoise = randn(size(y));
152 max_w = max(abs(wnoise(:)));
153 if max_w > 0
154     wnoise = wnoise / max_w;
155 end
156 % Normalize
157 end
158 y_out = yf .* wnoise;
159 max_y_out = max(abs(y_out(:)));
160 if max_y_out > 0
161     y_out = y_out / max_y_out;
162 end
163 % Normalize
164 % ***** Plot the multiplied signal:
165 subplot(2,4,1)
166 plot(t, y_out, 'b');
167 title('Multiplied with White Noise (time)');
168 ylabel('16-bit data');
169 xlabel('Time, s')
170 % ***** Calculate the spectrum of the multiplied signal:
171 y_out_fft_raw = abs(fft(y_out));
172 y_out_fft = y_out_fft_raw / (max(y_out_fft_raw) + eps);
173 % ***** Plot the spectrum:
174 subplot(2,4,5)
175 plot(Faxis, y_out_fft, 'b')
176 title('Multiplied with White Noise Spectrum');
177 xlabel('Frequency, Hz')
178 ylabel('Normalized FFT Power')
179 axis([0 12000 0 1.1]);
180
181 % ***** STEP 5: Final Bandpass Filtering:
182 y_bp_out = filter(b, a, y_out);
183 max_y_bp_out = max(abs(y_bp_out(:)));
184 if max_y_bp_out > 0
185     y_bp_out = y_bp_out / max_y_bp_out;
186 end
187 % Normalize
188 % ***** Plot the final bandpass filtered signal:
189 subplot(2,4,2)
190 plot(t, y_bp_out, 'b');
191 title('Final Bandpass Filtered Signal (time)');
192 ylabel('16-bit data');
193 xlabel('Time, s')
194 % ***** Calculate the spectrum of the final bandpass filtered signal:
195 yc_fft_raw = abs(fft(y_bp_out));
196 yc_fft = yc_fft_raw / (max(yc_fft_raw) + eps);
197 % ***** Plot the spectrum:
198 subplot(2,4,6)
199 plot(Faxis, yc_fft, 'b')
200 title('Final Bandpass Filtered Spectrum');
201 xlabel('Frequency, Hz')
202 ylabel('Normalized FFT Power')
203 axis([0 12000 0 1.1]);
204
205 % ***** STEP 6: Add this channel to output:
206 y_voc = y_voc + y_bp_out;
207 end
208 % After the for k = 1:N loop
209 if any(isnan(y_voc(:)))
210     disp('NaNs detected in y_voc, replacing with zeros');
211     y_voc(~isfinite(y_voc)) = 0;
212 end
213 disp('Press Enter for vocoder output.');
```

% Time-domain vocoder output

```

214 figure;
215 subplot(2,1,1);
216 plot(t, y_voc);
217 title('Vocoder Output (Time Domain)');
218 xlabel('Time, s');
219 ylabel('Amplitude');
```

% Spectrum of vocoder output

```

220 Yvoc_fft_raw = abs(fft(y_voc));
221 Yvoc_fft = Yvoc_fft_raw / max(Yvoc_fft_raw(:));
222 Faxis = linspace(0, Fs, length(y_voc));
223
224 subplot(2,1,2);
225 plot(Faxis, Yvoc_fft);
226 title('Vocoder Output Spectrum');
227 xlabel('Frequency, Hz');
228 ylabel('Normalized FFT Power');
229 xlim([0 12000]);
230
231 pause
232 % ***** Play vocoder output:
233 soundsc(y_voc, Fs);
234
```

Filter Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Task 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fs = 48000;           % sampling frequency (Hz)
nfft = 4096;         % for freq response plotting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task 1: FIR LPF (design using firpm & firpmord)
% Interpretation: Lowpass with passband edge fp = 10 kHz (0 - 10k pass)
%               stopband edge fsb = 12 kHz (attenuate beyond 12k)
%               If you actually want a bandpass 300-10k
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('--- Task 1: FIR LPF ---\n');
% Design specs (change these if instructor gave different edges)
fp = 300;             % passband edge Hz
fsb = 10000;          % stopband edge Hz
% allowed deviations (example ripples)
Rp = 1;               % passband ripple (dB)
As = 40;              % stopband attenuation (dB)
% Convert ripple/atten to linear deviations for firpmord
dev = [(10^(Rp/20)-1)/(10^(Rp/20)+1), 10^(-As/20)]; % approx [delta_p, delta_s]

% Frequency vector for firpmord: [fstop fpass] or [fpass fstop] depends on filter type
% For lowpass: f = [fp fsb]; a = [1 0];
f = [fp fsb]; a = [1 0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1. FOR DENSITY FACTOR = 20 %%%%%%%%%%
dens1 = 20;
% 1a) Design with Minimum order
[n_min, fo, ao, w] = firpmord(f, a, dev, fs, dens1);
fprintf('Task 1: firpmord min order (dens=%d) = %d\n', dens1, n_min);
b_min = firpm(n_min, fo, ao, w); % Parks-McClellan design
a_min = 1;
plot_responses(b_min, a_min, fs, nfft, sprintf('FIR LPF - min order=%d df=%d', n_min,
dens1));

% 1b) Design with Order = 50
N_user = 50;
% Need frequency grid normalized to [-1 1] for firpm? Using firpm with frequency in Hz via
fo from firpmord
% We'll use the same fo/ao weights but set order to N_user
b50 = firpm(N_user, fo, ao, w);
plot_responses(b50, a_min, fs, nfft, sprintf('FIR LPF - N=%d df=%d', N_user, dens1));

%% =====
% Comparison Plots: Min Order vs Order = 50
% (Using existing variables: b_min, b50, a_min, fs, nfft)
% =====

% Frequency response for both filters
```

```

[H_min, W] = freqz(b_min, a_min, nfft, fs);
[H_50, ~] = freqz(b50, a_min, nfft, fs);

% Magnitude (in dB)
mag_min = 20*log10(abs(H_min) + eps);
mag_50 = 20*log10(abs(H_50) + eps);

% Phase (unwrap to remove discontinuities)
phase_min = unwrap(angle(H_min));
phase_50 = unwrap(angle(H_50));

% Group Delay
[gd_min, w_gd] = grpdelay(b_min, a_min, nfft, fs);
[gd_50, ~] = grpdelay(b50, a_min, nfft, fs);

%% ===== Magnitude Comparison =====
figure('Name','Magnitude Comparison','NumberTitle','off');
plot(W, mag_min, 'r', 'LineWidth', 0.6); hold on;
plot(W, mag_50, 'b', 'LineWidth', 0.6);
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('FIR LPF - Magnitude Response Comparison');
legend('Minimum Order', 'Order = 50', 'Location', 'Best');

%% ===== Phase Comparison =====
figure('Name','Phase Comparison','NumberTitle','off');
plot(W, phase_min, 'r', 'LineWidth', 0.6); hold on;
plot(W, phase_50, 'b', 'LineWidth', 0.6);
grid on;
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
title('FIR LPF - Phase Response Comparison');
legend('Minimum Order', 'Order = 50', 'Location', 'Best');

%% ===== Group Delay Comparison =====
figure('Name','Group Delay Comparison','NumberTitle','off');
plot(w_gd, gd_min, 'r', 'LineWidth', 0.6); hold on;
plot(w_gd, gd_50, 'b', 'LineWidth', 0.6);
grid on;
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
title('FIR LPF - Group Delay Comparison');
legend('Minimum Order', 'Order = 50', 'Location', 'Best');

xlim([0 25000])
ylim([0.0 27.0])

```

Activate Win
Go to Settings to

```

%% 2. FOR DENSITY FACTOR = 50 %%%
dens2 = 50;
% 2a) Design with min order
[n_min2, fo2, ao2, w2] = firpmord(f, a, dev, fs, dens2);
fprintf('Task 1: firpmord min order (dens=%d) = %d\n', dens2, n_min2);
b_min2 = firpm(n_min2, fo2, ao2, w2);
plot_responses(b_min2, a_min, fs, nfft, sprintf('FIR LPF, min order=%d, df=%d', n_min2,
dens2));

% 2b) Design with Order = 50
new_order1 = 50;
[~, fo2, ao2, w2] = firpmord(f, a, dev, fs, dens2);
fprintf('Task 1: firpmord order (dens=%d) = %d\n', dens2, new_order1);
new_N1 = firpm(new_order1, fo2, ao2, w2);
plot_responses(new_N1, a_min, fs, nfft, sprintf('FIR LPF, Order=%d df=%d', new_order1,
dens2));

fprintf('Observations Task 1:\n');
fprintf(' - Increasing N (order) produces a sharper transition and usually lower ripple in
the passband.\n');
fprintf(' - Changing firpmord ''density'' can change the estimated minimum N because it
uses a denser frequency grid to estimate transitions (higher density -> more accurate but
sometimes larger N).\n\n');

%% =====
% Comparison of FIR LPF with Different Density Factors
% (df = 20 vs df = 50)
% =====
% ===== Recreate Filters at Density Factor = 50 =====
dens2 = 50;

% Minimum order (df = 50)
[n_min_df50, fo2, ao2, w2] = firpmord(f, a, dev, fs, dens2);
b_min_df50 = firpm(n_min_df50, fo2, ao2, w2);

% Order = 50 (df = 50)
N_user = 50;
b50_df50 = firpm(N_user, fo2, ao2, w2);

% ===== Frequency Responses =====
% For df = 20 filters (already created earlier)
[H_min_df20, W] = freqz(b_min, a_min, nfft, fs);
[H_50_df20, ~] = freqz(b50, a_min, nfft, fs);

% For df = 50 filters (newly created)
[H_min_df50, ~] = freqz(b_min_df50, a_min, nfft, fs);
[H_50_df50, ~] = freqz(b50_df50, a_min, nfft, fs);

```

```

% Magnitude
mag_min_df20 = 20*log10(abs(H_min_df20) + eps);
mag_min_df50 = 20*log10(abs(H_min_df50) + eps);

% Phase
phase_min_df20 = unwrap(angle(H_min_df20));
phase_min_df50 = unwrap(angle(H_min_df50));

% Group Delay
[gd_min_df20, w_gd] = grpdelay(b_min, a_min, nfft, fs);
[gd_min_df50, ~] = grpdelay(b_min_df50, a_min, nfft, fs);

%% =====
% Combined Comparison Plots: Density Factor 20 vs 50
% =====
figure('Name','FIR LPF Comparison (Density Factors 20 vs 50)', 'NumberTitle','off','Position',[100 100 900 700]);

% ===== 1. Magnitude Subplot =====
subplot(3,1,1);
plot(W, mag_min_df20, 'g', 'LineWidth', 1.5); hold on;
plot(W, mag_min_df50, 'y--', 'LineWidth', 1.5);
grid on;
ylabel('Magnitude (dB)');
title('FIR LPF - Magnitude Response');
legend('df = 20', 'df = 50', 'Location', 'Best');

% ===== 2. Phase Subplot =====
subplot(3,1,2);
plot(W, phase_min_df20, 'g', 'LineWidth', 1.5); hold on;
plot(W, phase_min_df50, 'y--', 'LineWidth', 1.5);
grid on;
ylabel('Phase (radians)');
title('FIR LPF - Phase Response');
legend('df = 20', 'df = 50', 'Location', 'Best');

% ===== 3. Group Delay Subplot =====
subplot(3,1,3);
plot(W_gd, gd_min_df20, 'g', 'LineWidth', 1.5); hold on;
plot(W_gd, gd_min_df50, 'y--', 'LineWidth', 1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
title('FIR LPF - Group Delay');
legend('df = 20', 'df = 50', 'Location', 'Best');

settitle('FIR Low-Pass Filter Comparison: Density Factors 20 vs 50','FontWeight','bold');

```

Helper function for plotting (magnitude (dB), phase, group delay)

```
function plot_responses(b, a, fs, nfft, titleStr)
% b: numerator (FIR b, IIR b)
% a: denominator (IIR a or 1 for FIR)
[H, W] = freqz(b, a, nfft, fs);
magdb = 20*log10(abs(H) + eps);
ph = unwrap(angle(H));
figure('Name', titleStr, 'NumberTitle', 'off');
subplot(3,1,1);
plot(W, magdb, 'r'); grid on;
ylabel('Magnitude(dB)');
title([titleStr ' - Magnitude']);
subplot(3,1,2);
plot(W, ph, 'b'); grid on;
ylabel('Phase (rad)');
title([titleStr ' - Phase']);
subplot(3,1,3);
% group delay
[gd, w_gd] = grpdelay(b, a, nfft, fs);
plot(w_gd, gd, 'g'); grid on;
ylabel('Group delay(samples)'); xlabel('Frequency(Hz)');
title([titleStr ' - Group delay']);
end
```

%%%

Task 2

%%%

%%%

% Task 2: FIR HPF: fstop = 8000, fpass = 10000

%%%

fprintf('--- Task 2: FIR HPF ---\n');

fstop_h = 8000;

fpass_h = 10000;

% For highpass, frequency vector for firpmord is [fstop fpass], and desired a = [0 1]

f_hp = [fstop_h fpass_h]; a_hp = [0 1];

% deviations (use same Rp, As)

dev_hp = dev;

% Get minimum order, dens = 20

dens = 20;

[n_hp_min, fo_hp, ao_hp, w_hp] = firpmord(f_hp, a_hp, dev_hp, fs, dens);

fprintf('Task 2: firpmord min order (dens=%d) = %d\n', dens, n_hp_min);

```

b_hp_min = firpm(n_hp_min, fo_hp, ao_hp, w_hp);
plot_responses(b_hp_min, 1, fs, nfft, sprintf('FIR HPF - min order=%d dens=%d', n_hp_min,
dens));

% Now design with N = 100, DF = 20
N2 = 100;
b_hp_100 = firpm(N2, fo_hp, ao_hp, w_hp);
plot_responses(b_hp_100, 1, fs, nfft, sprintf('FIR HPF - N=%d dens=%d', N2, dens));

% Now design with N = 100, DF = 50
N2 = 100;
dens1 = 50;
b_hp_100 = firpm(N2, fo_hp, ao_hp, w_hp);
plot_responses(b_hp_100, 1, fs, nfft, sprintf('FIR HPF - N=%d dens=%d', N2, dens1));

% Get minimum order, dens = 50
dens1 = 50;
[n_hp_min, fo_hp, ao_hp, w_hp] = firpmord(f_hp, a_hp, dev_hp, fs, dens);
fprintf('Task 2: firpmord min order (dens=%d) = %d\n', dens, n_hp_min);
b_hp_min = firpm(n_hp_min, fo_hp, ao_hp, w_hp);
plot_responses(b_hp_min, 1, fs, nfft, sprintf('FIR HPF - min order=%d dens=%d', n_hp_min,
dens1));

fprintf('Observations Task 2:\n');
fprintf(' - Minimum N produces acceptable ripple but transition width is determined by
spec. N=100 gives much sharper transition and better attenuation in stopband.\n - Density
factor affects order estimate and how well the algorithm samples the desired response when
estimating order.\n\n');

```

```

%% =====
% Task 2 - FIR HPF Comparison (Min Order vs N = 100)
% Separate plots for Magnitude, Phase, and Group Delay
% =====

% --- Compute frequency responses for both filters ---
[H_hp_min, w] = freqz(b_hp_min, 1, nfft, fs);
[H_hp_100, ~] = freqz(b_hp_100, 1, nfft, fs);

% Magnitude (dB)
mag_hp_min = 20*log10(abs(H_hp_min) + eps);
mag_hp_100 = 20*log10(abs(H_hp_100) + eps);

% Phase (radians)
phase_hp_min = unwrap(angle(H_hp_min));
phase_hp_100 = unwrap(angle(H_hp_100));

% Group Delay (samples)

```

Activate V
Go to Setting

```

[gd_hp_min, w_gd] = grpdelay(b_hp_min, 1, nfft, fs);
[gd_hp_100, ~] = grpdelay(b_hp_100, 1, nfft, fs);

%% =====
% 1. Magnitude Response Comparison
% =====
figure('Name','FIR HPF Magnitude Comparison','NumberTitle','off');
plot(W, mag_hp_min, 'r', 'LineWidth', 0.5); hold on;
plot(W, mag_hp_100, 'b', 'LineWidth', 0.5);
xlabel('Frequency (Hz)');
grid on;
ylim([-150 2.0]);
ylabel('Magnitude (dB)');
title('FIR High-Pass Filter - Magnitude Response');
legend('Minimum Order', 'Order = 100', 'Location','Best');

%% =====
% 2. Phase Response Comparison
% =====
figure('Name','FIR HPF Phase Comparison','NumberTitle','off');
plot(W, phase_hp_min, 'r', 'LineWidth', 0.5); hold on;
plot(W, phase_hp_100, 'b', 'LineWidth', 0.5);
xlabel('Frequency (Hz)');
grid on;
ylabel('Phase (radians)');
title('FIR High-Pass Filter - Phase Response');
legend('Minimum Order', 'Order = 100', 'Location','Best');

%% =====
% 3. Group Delay Comparison
% =====
figure('Name','FIR HPF Group Delay Comparison','NumberTitle','off');
plot(W_gd, gd_hp_min, 'r', 'LineWidth', 0.5); hold on;
plot(W_gd, gd_hp_100, 'b', 'LineWidth', 0.5);
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
grid on;
ylim([0.0 52.0]);
title('FIR High-Pass Filter - Group Delay');
legend('Minimum Order', 'Order = 100', 'Location','Best');

```

```

% ===== Design HPF with Density Factor = 20 =====
dens20 = 20;
[n_hp_min_df20, fo_hp_df20, ao_hp_df20, w_hp_df20] = firpmord(f_hp, a_hp, dev_hp, fs,
dens20);
b_hp_min_df20 = firpm(n_hp_min_df20, fo_hp_df20, ao_hp_df20, w_hp_df20);

```

```

% ===== Design HPF with Density Factor = 50 =====
dens50 = 50;
[n_hp_min_df50, fo_hp_df50, ao_hp_df50, w_hp_df50] = firpmord(f_hp, a_hp, dev_hp, fs,
dens50);
b_hp_min_df50 = firpm(n_hp_min_df50, fo_hp_df50, ao_hp_df50, w_hp_df50);

% ===== Frequency Responses =====
[H_df20, W] = freqz(b_hp_min_df20, a_min, nfft, fs);
[H_df50, ~] = freqz(b_hp_min_df50, a_min, nfft, fs);

% Magnitude
mag_df20 = 20*log10(abs(H_df20) + eps);
mag_df50 = 20*log10(abs(H_df50) + eps);

% Phase
phase_df20 = unwrap(angle(H_df20));
phase_df50 = unwrap(angle(H_df50));

% Group Delay
[gd_df20, w_gd] = grpdelay(b_hp_min_df20, a_min, nfft, fs);
[gd_df50, ~] = grpdelay(b_hp_min_df50, a_min, nfft, fs);

%% =====
% Combined Comparison Figure (df = 20 vs df = 50)
% =====
figure('Name','FIR HPF Comparison ', ...
'NumberTitle','off','Position',[100 100 900 700]);

% ----- 1. Magnitude -----
subplot(3,1,1);
plot(W, mag_df20, 'r', 'linewidth', 1.5); hold on;
plot(W, mag_df50, 'b--', 'linewidth', 1.5);
ylabel('Magnitude (dB)');
ylim([-110 5.0]);
title('FIR High-Pass Filter - Magnitude Response');
legend('Density Factor = 20', 'Density Factor = 50', 'location','best');

% ----- 2. Phase -----
subplot(3,1,2);
plot(W, phase_df20, 'r', 'linewidth', 1.5); hold on;
plot(W, phase_df50, 'b--', 'linewidth', 1.5);
ylabel('Phase (radians)');
ylim([-40 5.0]);
title('FIR High-Pass Filter - Phase Response');
legend('Density Factor = 20', 'Density Factor = 50', 'location','best');

% ----- 3. Group Delay -----
subplot(3,1,3);

```

Activate V
Go to Setting

```

plot(w_gd, gd_df20, 'r', 'linewidth', 1.5); hold on;
plot(w_gd, gd_df50, 'b--', 'linewidth', 1.5);
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
ylim([0.0 50.0]);
title('FIR High-Pass Filter - Group Delay');
legend('Density Factor = 20', 'Density Factor = 50', 'location','best');

sgtitle('FIR High-Pass Filter Comparison: Density Factors 20 vs 50', 'FontWeight','bold');

```

Helper function for plotting (magnitude (dB), phase, group delay)

```

function plot_responses(b, a, fs, nfft, titleStr)
% b: numerator (FIR b, IIR b)
% a: denominator (IIR a or 1 for FIR)
[H, W] = freqz(b, a, nfft, fs);
magdb = 20*log10(abs(H) + eps);
ph = unwrap(angle(H));
figure('Name', titleStr, 'NumberTitle', 'off');
subplot(3,1,1);
plot(W, magdb); grid on;
ylabel('Magnitude (dB)'); title([titleStr ' - Magnitude']);
subplot(3,1,2);
plot(W, ph); grid on;
ylabel('Phase (rad)'); title([titleStr ' - Phase']);
subplot(3,1,3);
% group delay
[gd, w_gd] = grpdelay(b,a,nfft,fs);
plot(w_gd, gd); grid on;
ylabel('Group Delay (samples)'); xlabel('Frequency (Hz)');
title([titleStr ' - Group delay']);
end

```

%%%

Task 3

%%%

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task 3: FIR Bandpass: fstop1=8000, fpass1=10000, fpass2=20000, fstop2=22000
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

fprintf('--- Task 3: FIR BPF ---\n');
fstop1 = 8000; fpass1 = 10000; fpass2 = 20000; fstop2 = 22000;
% For bandpass case, vector of edges and desired amplitudes
f_bp = [fstop1 fpass1 fpass2 fstop2]; a_bp = [0 1 0]; % 0->stop, 1->pass, 0->stop
dev_bp = [dev(2) dev(1) dev(2)]; % approx: stop, pass, stop deviations

```

```

% with gf=20, N=min-order
dens = 20;
[n_bpf_min, fo_bpf, ao_bpf, w_bpf] = firpmord(f_bp, a_bp, dev_bp, fs, dens);

```

```

fprintf('Task 3: firpmord min order (dens=%d) = %d\n', dens, n_bpf_min);
b_bpf_min = firpm(n_bpf_min, fo_bpf, ao_bpf, w_bpf);
plot_responses(b_bpf_min, 1, fs, nfft, sprintf('FIR BPF - min order=%d dens=%d',
n_bpf_min, dens));

% with df=50, N=min-order
dens1 = 50;
[n_bpf_min, fo_bpf, ao_bpf, w_bpf] = firpmord(f_bp, a_bp, dev_bp, fs, dens1);
fprintf('Task 3: firpmord min order (dens=%d) = %d\n', dens1, n_bpf_min);
b_bpf_min = firpm(n_bpf_min, fo_bpf, ao_bpf, w_bpf);
plot_responses(b_bpf_min, 1, fs, nfft, sprintf('FIR BPF - min order=%d dens=%d',
n_bpf_min, dens1));

% With df=20, N = 100
N3 = 100;
b_bpf_100 = firpm(N3, fo_bpf, ao_bpf, w_bpf);
plot_responses(b_bpf_100, 1, fs, nfft, sprintf('FIR BPF - N=%d dens=%d', N3, dens));

% with df=50, N=100
N3 = 100;
b_bpf_100 = firpm(N3, fo_bpf, ao_bpf, w_bpf);
plot_responses(b_bpf_100, 1, fs, nfft, sprintf('FIR BPF - N=%d dens=%d', N3, dens1));

fprintf('Observations Task 3:\n');
fprintf(' - BPF minimum order gives acceptable response but transition regions are wide if
min order is low.\n - N=100 yields much steeper band edges and deeper stopband
attenuation.\n - Changing density affects the estimated min order and slight shape of
response; higher density may increase N estimate.\n\n');

%% =====
% Task 3 - FIR BPF Comparison (Min Order vs N = 100)
% Separate plots for Magnitude, Phase, and Group Delay
% =====

% --- Compute frequency responses for both filters ---
[H_bpf_min, W] = freqz(b_bpf_min, 1, nfft, fs);
[H_bpf_100, ~] = freqz(b_bpf_100, 1, nfft, fs);

% Magnitude (dB)
mag_bpf_min = 20*log10(abs(H_bpf_min) + eps);
mag_bpf_100 = 20*log10(abs(H_bpf_100) + eps);

% Phase (radians)
phase_bpf_min = unwrap(angle(H_bpf_min));
phase_bpf_100 = unwrap(angle(H_bpf_100));

% Group Delay (samples)

```

Activate Windows
Go to Settings

```

[gd_bpf_min, w_gd] = grdelay(b_bpf_min, 1, nfft, fs);
[gd_bpf_100, ~] = grdelay(b_bpf_100, 1, nfft, fs);

%% =====
% 1. Magnitude Response Comparison
% =====
figure('Name','FIR BPF Magnitude Comparison','NumberTitle','off');
plot(W, mag_bpf_min, 'r', 'LineWidth', 0.5); hold on;
plot(W, mag_bpf_100, 'b', 'LineWidth', 0.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('FIR Bandpass Filter - Magnitude Response');
legend('Minimum Order', 'Order = 100', 'Location','Best');

%% =====
% 2. Phase Response Comparison
% =====
figure('Name','FIR BPF Phase Comparison','NumberTitle','off');
plot(W, phase_bpf_min, 'r', 'LineWidth', 0.5); hold on;
plot(W, phase_bpf_100, 'b', 'LineWidth', 0.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
title('FIR Bandpass Filter - Phase Response');
legend('Minimum Order', 'Order = 100', 'Location','Best');

%% =====
% 3. Group Delay Comparison
% =====
figure('Name','FIR BPF Group Delay Comparison','NumberTitle','off');
plot(W_gd, gd_bpf_min, 'r', 'LineWidth', 0.5); hold on;
plot(W_gd, gd_bpf_100, 'b', 'LineWidth', 0.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
ylim([15 55]);
title('FIR Bandpass Filter - Group Delay');
legend('Minimum Order', 'Order = 100', 'Location','Best');

%% =====
% Task 3 - FIR BPF Comparison (Density Factor = 20 vs 50)
% Using subplots for Magnitude, Phase, and Group Delay
% =====

% ===== Design BPF with Density Factor = 20 =====
dens20 = 20;
[n_bpf_min_df20, fo_bp_df20, ao_bp_df20, w_bp_df20] = firpmord(f_bp, a_bp, [dev_bp(2)
dev_bp(1) dev_bp(2)], fs, dens20);

```

```

b_bpf_df20 = firpm(n_bpf_min_df20, fo_bp_df20, ao_bp_df20, w_bp_df20);

% ===== Design BPF with Density Factor = 50 =====
dens50 = 50;
[n_bpf_min_df50, fo_bp_df50, ao_bp_df50, w_bp_df50] = firpmord(f_bp, a_bp, [dev_bp(2)
dev_bp(1) dev_bp(2)], fs, dens50);
b_bpf_df50 = firpm(n_bpf_min_df50, fo_bp_df50, ao_bp_df50, w_bp_df50);

% ===== Frequency Responses =====
[H_df20, W] = freqz(b_bpf_df20, a_min, nfft, fs);
[H_df50, ~] = freqz(b_bpf_df50, a_min, nfft, fs);

% Magnitude
mag_df20 = 20*log10(abs(H_df20) + eps);
mag_df50 = 20*log10(abs(H_df50) + eps);

% Phase
phase_df20 = unwrap(angle(H_df20));
phase_df50 = unwrap(angle(H_df50));

% Group Delay
[gd_df20, w_gd] = grpdelay(b_bpf_df20, a_min, nfft, fs);
[gd_df50, ~] = grpdelay(b_bpf_df50, a_min, nfft, fs);

%% =====
% Combined Comparison Figure (df = 20 vs df = 50)
% =====
figure('Name','FIR BPF Comparison (Density Factor 20 vs 50)', ...
'NumberTitle','off','Position',[100 100 900 700]);

% ----- 1. Magnitude -----
subplot(3,1,1);
plot(W, mag_df20, 'r', 'LineWidth', 0.5); hold on;
plot(W, mag_df50, 'b--', 'LineWidth', 0.5);
grid on;
ylabel('Magnitude (dB)');
title('FIR Bandpass Filter - Magnitude Response');
legend('Density Factor = 20', 'Density Factor = 50', 'Location','Best');

% ----- 2. Phase -----
subplot(3,1,2);
plot(W, phase_df20, 'r', 'LineWidth', 0.5); hold on;
plot(W, phase_df50, 'b--', 'LineWidth', 0.5);
grid on;
ylabel('Phase (radians)');
title('FIR Bandpass Filter - Phase Response');
legend('Density Factor = 20', 'Density Factor = 50', 'Location','Best');

% ----- 3. Group Delay -----

```

Activate Windows
Go to Settings

```

subplot(3,1,3);
plot(w_gd, gd_df20, 'r', 'LineWidth', 0.5); hold on;
plot(w_gd, gd_df50, 'b--', 'LineWidth', 0.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
title('FIR Bandpass Filter - Group Delay');
legend('Density Factor = 20', 'Density Factor = 50', 'Location','Best');

settitle('FIR Bandpass Filter Comparison: Density Factors 20 vs 50', 'FontWeight','bold');

```

Helper function for plotting (magnitude (dB), phase, group delay)

```

function plot_responses(b, a, fs, nfft, titleStr)
% b: numerator (FIR b, IIR b)
% a: denominator (IIR a or 1 for FIR)
[H, W] = freqz(b, a, nfft, fs);
magdb = 20*log10(abs(H) + eps);
ph = unwrap(angle(H));
figure('Name', titleStr, 'NumberTitle', 'off');
subplot(3,1,1);
plot(W, magdb); grid on;
ylabel('Magnitude (dB)'); title([titleStr ' - Magnitude']);
subplot(3,1,2);
plot(W, ph); grid on;
ylabel('Phase (rad)'); title([titleStr ' - Phase']);
subplot(3,1,3);
% group delay
[gd, w_gd] = grpdelay(b,a,nfft,fs);
plot(w_gd, gd); grid on;
ylabel('Group delay (samples)'); xlabel('Frequency (Hz)');
title([titleStr ' - Group delay']);
end

```

%%%

Task 4

%%%

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task 4: FIR Band Stop (notch bandstop):
% fpass1=6000, fstop1=8000, fstop2=12000, fpass2=14000
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('--- Task 4: FIR Bandstop ---\n');
fpass1 = 6000; fstop1 = 8000; fstop2 = 12000; fpass2 = 14000;
f_bs = [fpass1 fstop1 fstop2 fpass2];

```

```

a_bs = [1 0 1]; % pass, stop, pass
dev_bs = [dev(1) dev(2) dev(1)]; % passband, stopband, passband

%With N=min-order, df=20
dens = 20;
[n_bs_min, fo_bs, ao_bs, w_bs] = firpmord(f_bs, a_bs, dev_bs, fs, dens);
fprintf('Task 4: firpmord min order (dens=%d) = %d\n', dens, n_bs_min);
b_bs_min = firpm(n_bs_min, fo_bs, ao_bs, w_bs);
plot_responses(b_bs_min, 1, fs, nfft, sprintf('FIR BS - min-order=%d dens=%d', n_bs_min,
dens));

%With N=min-order, df=50
dens1 = 50;
[n_bs_min, fo_bs, ao_bs, w_bs] = firpmord(f_bs, a_bs, dev_bs, fs, dens);
fprintf('Task 4: firpmord min order (dens=%d) = %d\n', dens, n_bs_min);
b_bs_min = firpm(n_bs_min, fo_bs, ao_bs, w_bs);
plot_responses(b_bs_min, 1, fs, nfft, sprintf('FIR BS - min-order=%d dens=%d', n_bs_min,
dens1));

% With N = 100, df=20
N4 = 100;
b_bs_100 = firpm(N4, fo_bs, ao_bs, w_bs);
plot_responses(b_bs_100, 1, fs, nfft, sprintf('FIR BS - N=%d dens=%d', N4, dens));

%With N=100, df=50
b_bs_100 = firpm(N4, fo_bs, ao_bs, w_bs);
plot_responses(b_bs_100, 1, fs, nfft, sprintf('FIR BS - N=%d dens=%d', N4, dens1));

fprintf('Observations Task 4:\n');
fprintf(' - Bandstop minimum order may leave shallow stop attenuation; N=100 improves stop
depth.\n');
fprintf(' - Density factor changes order estimate similarly to previous tasks.\n\n');

%% =====
% Task 4 - FIR Bandstop Filter Comparison (Min Order vs N = 100)
% Separate plots for Magnitude, Phase, and Group Delay
% =====
% --- Compute frequency responses for both filters ---
[H_bs_min, W] = freqz(b_bs_min, 1, nfft, fs);
[H_bs_100, ~] = freqz(b_bs_100, 1, nfft, fs);

% Magnitude (in dB)
mag_bs_min = 20*log10(abs(H_bs_min) + eps);
mag_bs_100 = 20*log10(abs(H_bs_100) + eps);

% Phase (radians)
phase_bs_min = unwrap(angle(H_bs_min));
phase_bs_100 = unwrap(angle(H_bs_100));

```

Activate W
Go to Settings

```

% Group Delay (samples)
[gd_bs_min, w_gd] = grpdelay(b_bs_min, 1, nfft, fs);
[gd_bs_100, ~] = grpdelay(b_bs_100, 1, nfft, fs);

%% =====
% 1. Magnitude Response Comparison
% =====
figure('Name','FIR Bandstop Magnitude Comparison','NumberTitle','off');
plot(W, mag_bs_min, 'r', 'linewidth', 0.5); hold on;
plot(W, mag_bs_100, 'b', 'linewidth', 0.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('FIR Bandstop Filter - Magnitude Response');
legend('Minimum Order', 'Order = 100', 'Location','Best');

%% =====
% 2. Phase Response Comparison
% =====
figure('Name','FIR Bandstop Phase Comparison','NumberTitle','off');
plot(W, phase_bs_min, 'r', 'linewidth', 0.5); hold on;
plot(W, phase_bs_100, 'b', 'linewidth', 0.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
title('FIR Bandstop Filter - Phase Response');
legend('Minimum Order', 'Order = 100', 'Location','Best');

%% =====
% 3. Group Delay Comparison
% =====
figure('Name','FIR Bandstop Group Delay Comparison','NumberTitle','off');
plot(W_gd, gd_bs_min, 'r', 'linewidth', 0.5); hold on;
plot(W_gd, gd_bs_100, 'b', 'linewidth', 0.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
ylim([15 55]);
title('FIR Bandstop Filter - Group Delay');
legend('Minimum Order', 'Order = 100', 'Location','Best');

%% =====
% Task 4 - FIR Bandstop Filter Comparison (df = 20 vs df = 50)
% Using subplots for Magnitude, Phase, and Group Delay
% =====
% ===== Design Bandstop Filter with Density Factor = 20 =====
dens20 = 20;
[n_bs_min_df20, fo_bs_df20, ao_bs_df20, w_bs_df20] = firpmord(f_bs, a_bs, [dev_bs(1)
dev_bs(2) dev_bs(1)], fs, dens20);
b_bs_df20 = firpm(n_bs_min_df20, fo_bs_df20, ao_bs_df20, w_bs_df20);

```

```

% ===== Design Bandstop Filter with Density Factor = 50 =====
dens50 = 50;
[n_bs_min_df50, fo_bs_df50, ao_bs_df50, w_bs_df50] = firpmord(f_bs, a_bs, [dev_bs(1)
dev_bs(2) dev_bs(1)], fs, dens50);
b_bs_df50 = firpm(n_bs_min_df50, fo_bs_df50, ao_bs_df50, w_bs_df50);

% ===== Frequency Responses =====
[H_df20, W] = freqz(b_bs_df20, a_min, nfft, fs);
[H_df50, ~] = freqz(b_bs_df50, a_min, nfft, fs);

% Magnitude (dB)
mag_df20 = 20*log10(abs(H_df20) + eps);
mag_df50 = 20*log10(abs(H_df50) + eps);

% Phase (radians)
phase_df20 = unwrap(angle(H_df20));
phase_df50 = unwrap(angle(H_df50));

% Group Delay (samples)
[gd_df20, w_gd] = grpdelay(b_bs_df20, a_min, nfft, fs);
[gd_df50, ~] = grpdelay(b_bs_df50, a_min, nfft, fs);

%% =====
% Combined Comparison Figure (df = 20 vs df = 50)
% =====
figure('Name','FIR Bandstop Filter Comparison (Density Factor 20 vs 50)', ...
'NumberTitle','off','Position',[100 100 900 700]);

% ----- 1. Magnitude -----
subplot(3,1,1);
plot(W, mag_df20, 'r', 'linewidth', 1.5); hold on;
plot(W, mag_df50, 'b--', 'linewidth', 1.5);
grid on;
ylabel('Magnitude (dB)');
title('FIR Bandstop Filter - Magnitude Response');
legend('Density Factor = 20', 'Density Factor = 50', 'Location','Best');

% ----- 2. Phase -----
subplot(3,1,2);
plot(W, phase_df20, 'r', 'linewidth', 1.5); hold on;
plot(W, phase_df50, 'b--', 'linewidth', 1.5);
grid on;
ylabel('Phase (radians)');
title('FIR Bandstop Filter - Phase Response');
legend('Density Factor = 20', 'Density Factor = 50', 'Location','Best');

% ----- 3. Group Delay -----
subplot(3,1,3);
plot(w_gd, gd_df20, 'r', 'linewidth', 1.5); hold on;
plot(w_gd, gd_df50, 'b--', 'linewidth', 1.5);

```

Activate W
Go to Settings

```

grid on;
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
title('FIR Bandstop Filter - Group Delay');
legend('Density Factor = 20', 'Density Factor = 50', 'Location','Best');

settitle('FIR Bandstop Filter Comparison: Density Factors 20 vs 50', 'FontWeight','bold');

```

Helper function for plotting (magnitude (dB), phase, group delay)

```

function plot_responses(b, a, fs, offt, titleStr)
% b: numerator (FIR b, IIR b)
% a: denominator (IIR a or 1 for FIR)
[H, W] = freqz(b, a, offt, fs);
magdb = 20*log10(abs(H) + eps);
ph = unwrap(angle(H));
figure('Name', titleStr, 'NumberTitle', 'off');
subplot(3,1,1);
plot(W, magdb); grid on;
xlabel('Magnitude (dB)'); title([titleStr ' - Magnitude']);
subplot(3,1,2);
plot(W, ph); grid on;
xlabel('Phase (rad)'); title([titleStr ' - Phase']);
subplot(3,1,3);
% group delay
[gd, wgd] = grpdelay(b,a,offt,fs);
plot(wgd, gd); grid on;
xlabel('Group delay (samples)'); xlabel('Frequency (Hz)');
title([titleStr ' - Group delay']);
end

```

%%%

Task 5

%%%

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task 5: IIR BPF: fstop1 = 4000, fpass1=6000, fpass2=20000, fstop2=22000
% Use elliptic IIR (ellipord + ellip), compare min order vs N=100
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

fprintf('--- Task 5: IIR BPF (Elliptic) ---\n');
fstop1 = 4000; fpass1 = 6000; fpass2 = 20000; fstop2 = 22000;
wp = [fpass1 fpass2]/(fs/2); % normalized passband
ws = [fstop1 fstop2]/(fs/2); % normalized stopband
Rp = 1; % dB pass ripple
Rs = 40; % dB stop attenuation

[n_ell_min, wn] = ellipord(wp, ws, Rp, Rs); % minimum order for the IIR
fprintf('Task 5: ellipord min order = %d\n', n_ell_min);
[b_ell_min, a_ell_min] = ellip(n_ell_min, Rp, Rs, wn, 'bandpass');
plot_responses(b_ell_min, a_ell_min, fs, offt, sprintf('IIR(elliptical) BPF - min-order=%d', n_ell_min));

```

```

% Compare with higher order (IIR N=100)
N5 = 100;
% Note: designing a 100th order elliptic IIR bandpass is possible but likely
unstable/won't be necessary.
% Instead, to compare, we design a higher-order FIR BPF (for apples-to-apples) OR cascade
smaller IIRs.
% But as assignment requests N=100: we will show that such a high-order IIR is rarely
needed; still we can
% design an IIR with order 100 (but numeric issues possible).
try
    [b_iir_100, a_iir_100] = ellip(N5, Rp, Rs, Wp, 'bandpass');
    plot_responses(b_iir_100, a_iir_100, fs, nfft, sprintf('IIR(elliptical BPF - min-
order=%d', N5));
    fprintf('Task 5: Successfully designed order %d elliptic IIR (note: high order IIRs
may be numerically sensitive).\n', N5);
catch ME
    fprintf('Task 5: Could not safely design order %d elliptic IIR due to numerical
issues: %s\n', N5, ME.message);
    fprintf('          Usually, IIR filters work well at low orders; very high orders are
unstable or sensitive.\n');
end

fprintf('Observations Task 5:\n');
fprintf(' - IIR minimum order will be small (elliptic gives sharp transitions at low
order).\n - Increasing the order for IIR often gives diminishing returns and may cause
numerical instability.\n - FIR requires larger N for similar transition steepness but is
always stable and has linear phase (if symmetric).\n\n');

%%% Helper function for plotting (magnitude (dB), phase, group delay)

```

```

function plot_responses(b, a, fs, nfft, titleStr)
    % b: numerator (FIR b, IIR b)
    % a: denominator (IIR a or 1 for FIR)
    [H, W] = freqz(b, a, nfft, fs);
    magdb = 20*log10(abs(H) + eps);
    ph = unwrap(angle(H));
    figure('Name', titleStr, 'NumberTitle', 'off');
    subplot(3,1,1);
    plot(W, magdb); grid on;
    ylabel('Magnitude (dB)'); title([titleStr ' - Magnitude']);
    subplot(3,1,2);
    plot(W, ph); grid on;
    ylabel('Phase (rad)'); title([titleStr ' - Phase']);
    subplot(3,1,3);
    % group delay
    [gd, w_gd] = grpdelay(b,a,nfft,fs);
    plot(w_gd, gd); grid on;
    ylabel('Group delay (samples)'); xlabel('Frequency (Hz)');
    title([titleStr ' - Group delay']);
end

```

%%%

Task 6

%%%

% 6. Design a IIR BPF for fstop1 = 8000 Hz, fpass1= 10,000 Hz,
% fpass2= 20,000 Hz and fstop2 = 22,000 Hz. Use the minimum order
% and N = 100 and comment on the difference. Then check the effect
% of the density factor and comment. Make fs = 48000 Hz.

%%%
% Task 6: IIR BPF - fstop1=8000, fpass1=10000, fpass2=20000, fstop2=22000
%%%

fprintf('--- Task 6: IIR BPF (Elliptic) ---\n');

--- Task 6: IIR BPF (Elliptic) ---

fstop1 = 8000; fpass1 = 10000; fpass2 = 20000; fstop2 = 22000;

Wp = [fpass1 fpass2]/(fs/2);

Ws = [fstop1 fstop2]/(fs/2);

[n_iir2_min, Wn2] = ellipord(Wp, Ws, Rp, Rs);

fprintf('Task 6: ellipord min order = %d\n', n_iir2_min);

Task 6: ellipord min order = 5

[b_iir2_min, a_iir2_min] = ellip(n_iir2_min, Rp, Rs, Wn2, 'bandpass');

plot_responses(b_iir2_min, a_iir2_min, fs, nfft, sprintf('IIR(elliptical) BPF - min-order=%d', n_iir2_min));

% Attempt N = 100 as in task description

try

[b_iir2_100, a_iir2_100] = ellip(N5, Rp, Rs, Wn2, 'bandpass');

plot_responses(b_iir2_100, a_iir2_100, fs, nfft, sprintf('IIR(elliptical) BPF - min-order=%d', N5));

catch ME

fprintf('Task 6: Could not safely design order %d elliptic IIR: %s\n', N5,
ME.message);

end

Task 6: Could not safely design order 100 elliptic IIR: The filter order is too large. Use a smaller value.

Use caution: high-order IIRs are rarely necessary.

fprintf('Observations Task 6:\n');

Observations Task 6:

fprintf(' - Same general behavior as Task 5: elliptic IIR gives small order -> sharp
transitions.\n - High IIR order (100) is not recommended in practice.\n\n');

- Same general behavior as Task 5: elliptic IIR gives small order -> sharp transitions.

- High IIR order (100) is not recommended in practice.

Helper function for plotting (magnitude (dB), phase, group delay)

function plot_responses(b, a, fs, nfft, titleStr)

% b: numerator (FIR b, IIR b)

% a: denominator (IIR a or 1 for FIR)

```

[H, W] = freqz(b, a, nfft, fs);
magdb = 20*log10(abs(H) + eps);
ph = unwrap(angle(H));
figure('Name', titleStr, 'NumberTitle', 'off');
subplot(3,1,1);
plot(W, magdb); grid on;
ylabel('Magnitude (dB)'); title([titleStr ' - Magnitude']);
subplot(3,1,2);
plot(W, ph); grid on;
ylabel('Phase (rad)'); title([titleStr ' - Phase']);
subplot(3,1,3);
% group delay
[gd, w_gd] = grpdelay(b,a,nfft,fs);
plot(w_gd, gd); grid on;
ylabel('Group delay (samples)'); xlabel('Frequency (Hz)');
title([titleStr ' - Group delay']);
end

```

Task 7

```

% Task 7: Compare between N = 100 and N = 130 (FIR)
fprintf('--- Task 7: Compare N=100 vs N=130 (FIR example using the bandpass from Task 3) -\n');

N_a = 100;
N_b = 130;

% Design both FIR bandpass filters (same specs as Task 3)
b_a = firpm(N_a, f_a_bpf, a_a_bpf, w_bpf);
b_b = firpm(N_b, f_a_bpf, a_a_bpf, w_bpf);

% -----
% Magnitude + Phase + Group delay comparison
% -----
[Ha, Wa] = freqz(b_a, 1, nfft, fs); % N=100
[Hb, Wb] = freqz(b_b, 1, nfft, fs); % N=130

% Group delay (in samples)
[Ga, wGa] = grpdelay(b_a, 1, nfft, fs);
[Gb, wGb] = grpdelay(b_b, 1, nfft, fs);

```

Activate Windows
Go to Settings

```
figure('Name','Task7: FIR compare N=100 vs N=130','NumberTitle','off');

% ----- Magnitude response -----
subplot(3,1,1);
plot(Wa, 20*log10(abs(Ha) + eps)); hold on;
plot(Wb, 20*log10(abs(Hb) + eps));
grid on;
legend(sprintf('N = %d', Na), sprintf('N = %d', Nb), 'location','Best');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Task 7: Magnitude Response (N=100 vs N=130)');

% ----- Phase response -----
subplot(3,1,2);
plot(Wa, unwrap(angle(Ha))*180/pi); hold on;
plot(Wb, unwrap(angle(Hb))*180/pi);
grid on;
legend(sprintf('N = %d', Na), sprintf('N = %d', Nb), 'location','Best');
xlabel('Frequency (Hz)');
ylabel('Phase (degrees)');
title('Task 7: Phase Response (N=100 vs N=130)');

% ----- Group delay -----
subplot(3,1,3);
plot(Wa, Ga); hold on;
plot(Wb, Gb);
grid on;
legend(sprintf('N = %d', Na), sprintf('N = %d', Nb), 'location','Best');
xlabel('Frequency (Hz)');
ylabel('Group Delay (samples)');
title('Task 7: Group Delay (N=100 vs N=130)');

fprintf('Observations Task 7:\n');
fprintf(' - N=130 gives a steeper transition band and better stopband attenuation than N=100.\n');
fprintf(' - The phase and group delay are smoother but slightly larger (more delay) for N=130.\n');
fprintf(' - Trade-off: higher N improves frequency characteristics at the cost of more computations and latency.\n\n');
```

%%%

Task9

%%%

```
% 9. Design an Elliptical LPF from 300 Hz to 10,000Hz using
% the minimum order and density factor is 20. Use fs = 48000 Hz,
% and use again order of 50. Then, check the effect of Density Factor
% of 50 and comment about the effects of the different order and
% different density gain.
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task 9: Elliptical LPF from 300 Hz to 10,000Hz
% Interpretation: Elliptic lowpass with passband edge 10k (same as Task1, but I use ellip)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('--- Task 9: Elliptic LPF ---\n');
fp = 10000; fsb = 12000;
wp = fp/(fs/2); ws = fsb/(fs/2);
Rp = 1; Rs = 40;
[n_ell_min, wn_ell] = ellipord(wp, ws, Rp, Rs);
fprintf('Task 9: ellipord min order = %d\n', n_ell_min);
[b_ell_min, a_ell_min] = ellip(n_ell_min, Rp, Rs, wn_ell);
plot_responses(b_ell_min, a_ell_min, fs, nfft, sprintf('Elliptic LPF - min-order=%d',
n_ell_min));

```

```

% Elliptic with order 50
N_ell_user = 50;
try
    [b_ell_50, a_ell_50] = ellip(N_ell_user, Rp, Rs, wn_ell);
    plot_responses(b_ell_50, a_ell_50, fs, nfft, sprintf('Elliptic LPF - N=%d',
N_ell_user));
catch ME
    fprintf('Task9: Could not design elliptic of order %d safely: %s\n', N_ell_user,
ME.message);
end

```

```

fprintf('Observations Task 9:\n');
fprintf(' - Elliptic IIR achieves very sharp transitions at low order but has non-linear
phase.\n - Changing density is not relevant for ellip/ellipord (density is for FIR
firrmord).\n\n')

```

```

%% Helper function for plotting (magnitude (dB), phase, group delay)

```

```

function plot_responses(b, a, fs, nfft, titleStr)
% b: numerator (FIR b, IIR b)
% a: denominator (IIR a or 1 for FIR)
[H, W] = freqz(b, a, nfft, fs);
magdb = 20*log10(abs(H) + eps);
ph = unwrap(angle(H));
figure('Name', titleStr, 'NumberTitle', 'off');
subplot(3,1,1);
plot(W, magdb); grid on;
ylabel('Magnitude (dB)'); title([titleStr ' - Magnitude']);
subplot(3,1,2);
plot(W, ph); grid on;
ylabel('Phase (rad)'); title([titleStr ' - Phase']);
subplot(3,1,3);
% group delay
[gd, w_gd] = grndelay(b,a,nfft,fs);

```

Activate W
Go to Settings

```
plot(w_gd, gd); grid on;  
ylabel('Group delay (samples)'); xlabel('Frequency (Hz)');  
title([titleStr ' - Group delay']);  
end
```