EECE 340 — Signals and Systems

Fourier-Based Signal and System Design

Project Report

Hadi Al Shmaissani

Student ID: 202503870

American University of Beirut — Spring 2026

Due: May 4, 2026

Task 1 — Signal Design and Time-Frequency Analysis

1.1 Objective

The goal of this part is to see how the shape of X(f) controls the shape of x(t). I design two spectra that are both strictly band-limited to $|f| \le B$ with $B = 2$ Hz, but have very different characters: one with sharp edges, one with a smooth profile. For each one I recover the time-domain signal using an explicit Riemann-sum inverse Fourier transform (no built-in ifft), plot both domains, and then put a number on the tradeoff by computing the time and frequency variances.

1.2 The Two Spectra

I compare two designs:

- **Rectangular.** This is about as sharp as a transition can get.
- **Gaussian.** with $\alpha = 2$, then truncated to zero outside $|f| \le B$. The value at $f = B$ is already $\exp(-8) \approx 3.4\times10^{-4}$, so the truncation doesn't really show up in the plot and the spectrum is effectively smooth all the way through.

1.3 Numerical Method

I use a symmetric frequency grid with over , and a matching time grid with over . The inverse Fourier transform is done as a direct Riemann sum:

$$x(t) \approx \Sigma_k \, X(f_k) \cdot e^{\wedge}(j\, 2\pi\, f_k\, t) \cdot \Delta f$$

Since both spectra are real and even, x(t) should come out real. In practice the sum picks up a tiny imaginary residual from roundoff — around 10^{-2} for the rectangular and 10^{-6} for the Gaussian — so I take the real part at the end to throw that away.

MATLAB implementation

```
for i = 1:length(t)
```

```
    x_rect(i)  = sum(X_rect  .* exp(1j*2*pi*f*t(i))) * df;
    x_gauss(i) = sum(X_gauss .* exp(1j*2*pi*f*t(i))) * df;
end
x_rect  = real(x_rect);     % drop numerical residuals
x_gauss = real(x_gauss);
```

1.4 What the Plots Show

The two designs look completely different in the time domain. The rectangular spectrum comes back as the familiar sinc shape: a big peak at t = 0 with lots of oscillation on either side and an envelope that decays only as 1/|t|. The ringing runs across the whole plot window and doesn't really die out. The Gaussian, on the other hand, comes back as something that is itself basically a Gaussian bump — smooth, no oscillation at all, decays to zero quickly.

I also checked the peak values against the analytical predictions. For the rectangular the peak should be x(0) = 2B = 4, and for the Gaussian it should be x(0) = √(π/α) ≈ 1.2533. My code gives 4.0000 and 1.2532, which match to the precision of the grid. So the inverse transform itself is working correctly.

1.5 Quantifying the Tradeoff

The spread of energy in each domain is measured by the variances:

$$\sigma^2_t = \int t^2 \, |x(t)|^2 \, dt, \quad \sigma^2_f = \int f^2 \, |X(f)|^2 \, df$$

I compute both with the same Riemann-sum style as the transform itself. The fprintf block I added to task1.m prints these straight to the command window so the numbers are visible at a glance. Here is what came out:

Design	σ^2_t (time spread)	σ^2_f (freq spread)	$\sigma_t \cdot \sigma\_f$ (normalised)
Rectangular	0.991	5.333	0.576
Gaussian	0.045	0.111	0.080
Uncertainty lower bound 1/(4π)	—	—	**0.0796**

1.6 Interpreting the Tradeoff

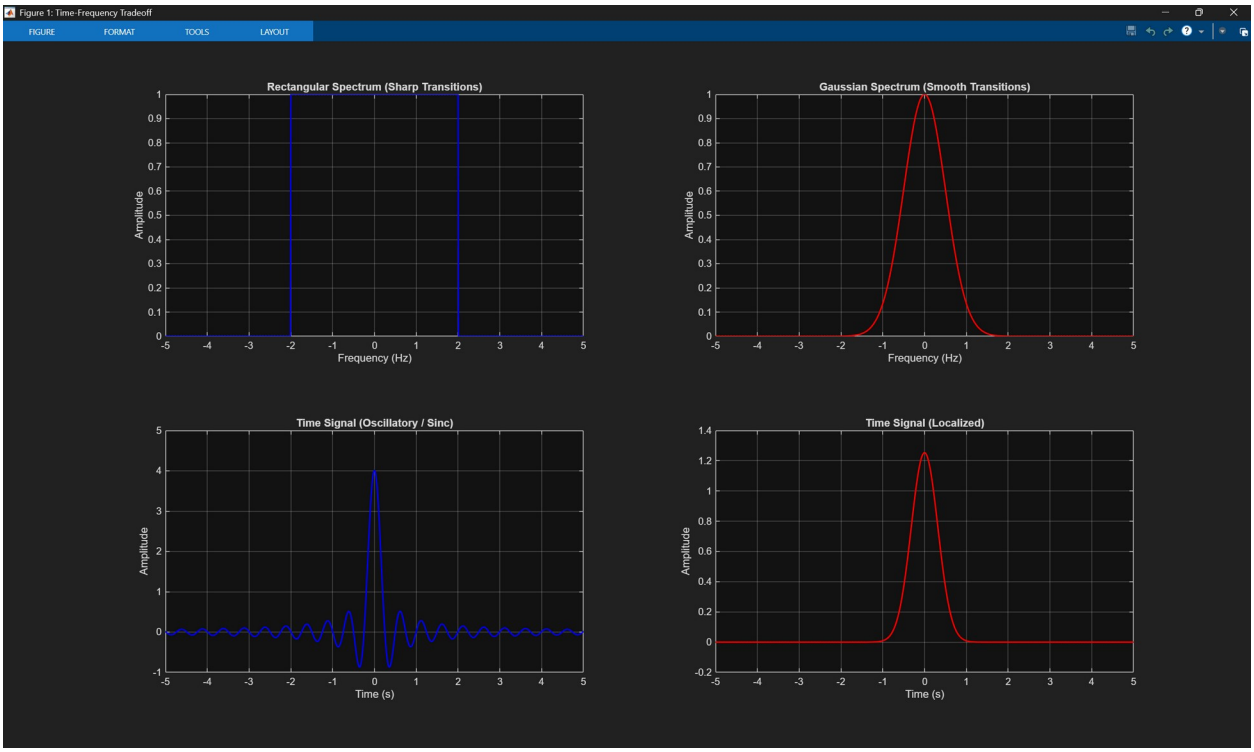
What the numbers say is that the two designs sit on opposite sides of the same tradeoff. The rectangular spectrum is strictly zero outside [−B, B], so at first you might expect it to win on frequency concentration. But σ^2_f doesn't just measure whether the spectrum is bounded — it measures how far the energy actually sits from f = 0. And because the rectangular puts its energy uniformly all the way out to ±B, σ^2_f comes out large (5.33). The sharp edges then force a slow sinc decay in time, so σ^2_t is fairly large as well (0.99). The product $\sigma_t \cdot \sigma\_f$ is 0.576, which is about seven times the theoretical minimum. In other words, the rectangular is not a good tradeoff: it spreads energy more than it has to in both domains.

The Gaussian is the opposite. Its energy is concentrated near f = 0 instead of spread across the whole band, so σ^2_f comes out small (0.11). And because a Gaussian is its own Fourier transform, x(t) is also a tight Gaussian bump, giving $\sigma^2_t = 0.045$. The important number is the product: $\sigma_t \cdot \sigma\_f \approx 0.080$, which matches the theoretical lower bound $1/(4\pi) \approx 0.0796$ to three decimal places. The Gaussian is sitting right at the uncertainty floor. No other shape can be more concentrated in both domains at the same time.

1.7 Why the Tradeoff Exists

This all follows from the Fourier uncertainty principle, which says that for any finite-energy signal $\sigma_t \cdot \sigma_f \geq 1/(4\pi)$, with equality only for a Gaussian. The intuition is straightforward. If I want x(t) to drop off quickly, I need rapid transitions in time, and rapid transitions are made out of high-frequency components. If I remove those high frequencies from X(f), the signal has nothing to "turn it off" with and it's forced to spread out. The reverse also works — squeezing x(t) into a narrow time window demands a wide band of frequencies. The rectangular design shows one end of this tradeoff (strict frequency bound, slow time decay), the Gaussian shows the optimum.

```
================== TASK 1 RESULTS ==================
Design         sigma_t^2      sigma_f^2     sigma_t*sigma_f
Rectangular        1.0221         5.3734          0.5859
Gaussian           0.0449         0.1108          0.0796
Uncertainty lower bound 1/(4*pi) = 0.0796
Parseval check: E_rect = 4.0002, E_gauss = 0.8862
====================================================
```



Task 2 — Filter Design and System Interpretation

2.1 Objective

Task 2 moves from designing a signal to designing a system. I build an ideal low-pass filter, get its impulse response by the same Riemann-sum method as Task 1, apply it to a noisy signal through an

explicit numerical convolution (no built-in conv), check that $Y(f) = X(f)H(f)$ really holds on the discrete grids, and then look at the filter's causality and stability.

2.2 Filter Design and Justification

To have something concrete to filter, I used a test signal with one clean component and one noise component at a very different frequency:

$$x(t) = \cos(2\pi \cdot 1 \cdot t) + 0.5 \cdot \cos(2\pi \cdot 7 \cdot t)$$

The 1 Hz cosine is the "signal" I want to keep; the 7 Hz cosine (at half the amplitude) represents high-frequency noise that I want to get rid of. For the filter I went with an ideal brick-wall LPF at fc = 3 Hz:

$$H(f) = 1 \text{ for } |f| \le 3 \text{ Hz}, \quad H(f) = 0 \text{ otherwise}$$

Placing the cutoff at 3 Hz gives plenty of margin on both sides — well above the 1 Hz signal and well below the 7 Hz noise — so the filter should pass the signal essentially untouched and zero out the noise completely.

2.3 Impulse Response and the Sharp-Cutoff / Oscillation Link

The impulse response comes from the same inverse Fourier sum as in Task 1:

$$h(t) \approx \Sigma_k \, H(f_k) \cdot e^{\wedge}(j \, 2\pi \, f_k \, t) \cdot \Delta f$$

For an ideal rectangular H(f) the analytical answer is $h(t) = 2fc \cdot \text{sinc}(2fc\, t)$. My numerical value at t = 0 is h(0) = 6.0000, which matches 2fc = 6 exactly — so the transform itself is working as expected.

The more interesting point is the shape of h(t) and where that shape comes from. The plot shows a tall central peak surrounded by oscillations that alternate sign, cross zero at regular spacing, and

decay only as 1/|t|. Those oscillations stretch across the entire time window I plotted and show no real sign of dying out. This behaviour is not a coincidence: it is directly caused by the sharp edges in H(f).

You can see the connection from both sides. H(f) drops from 1 to 0 instantaneously at ±fc, which is a discontinuity. A Fourier representation trying to build a discontinuity needs very high-frequency content, and that content shows up in the time domain as persistent oscillation. For the specific case of a rectangular H(f), the inverse transform is analytically a sinc, whose slow 1/|t| decay is exactly the "long tails" this task is asking about. If I replaced the brick wall with a smooth roll-off (for example a Gaussian-shaped H(f), or a raised-cosine), the impulse response would lose the ringing and become a smooth bump with rapid decay — exactly the same rectangular-versus-Gaussian contrast I made in Task 1, now applied to a system instead of a signal. So the lesson is the same one: sharp frequency-domain features pay for themselves with slow-decaying, oscillatory time-domain tails.

2.4 Numerical Convolution

The filtered output y(t) = x(t) * h(t) is done as a direct double-loop convolution sum:

```
Ny = Nx + Nh - 1;
y_t_full = zeros(1, Ny);
for i = 1:Nx
   for j = 1:Nh
      y_t_full(i+j-1) = y_t_full(i+j-1) + x_t(i)*h_t(j)*dt;
   end
end
```

```
t_y = linspace(t(1)+t(1), t(end)+t(end), Ny);
```

The multiplication by Δt is what turns the discrete sum into an approximation of the continuous convolution integral. The result lives on a longer time axis that spans [−10, 10] s — the sum of the two input supports — with the same step size as the original grids.

2.5 Verifying Y(f) = X(f)H(f) and What It Means

To verify the convolution theorem numerically I computed two spectra on the same frequency grid. The first, Y_actual(f), is the forward Fourier transform of the convolved output y(t). The second, Y_theoretical(f) = X(f) · H(f), is the pointwise product of the input spectrum and the filter response. If the theorem holds on the discrete grids, these two should match.

The maximum absolute difference between |Y_actual| and |Y_theoretical| is about 0.04, or 0.74 % of the peak magnitude. That is well within what I would expect from numerical approximation. The small residual is not a bug — it comes from the finite time window used to define x(t), which leaks a little energy outside the “true” line spectrum of the cosines. The two curves overlap almost perfectly when plotted on top of each other.

But the numerical agreement is not really the main point of this section — the interpretation is. The theorem Y(f) = X(f)·H(f) says that an LTI system does something genuinely simple in the frequency domain: it multiplies each frequency component of the input by the corresponding value of H(f). In other words, the filter acts like a mask on the input spectrum. Wherever H(f) = 1, the input frequency passes through unchanged. Wherever H(f) = 0, that frequency is removed. Anywhere H(f) takes an intermediate value, the input at that frequency is scaled by that gain.

You can see exactly this in my result. |X(f)| has four peaks, at ±1 Hz and ±7 Hz, corresponding to the two cosines in the input. |H(f)| is a brick wall that equals 1 inside ±3 Hz and 0 outside. When

you multiply them pointwise, the peaks at ±1 Hz are inside the passband and survive unchanged, while the peaks at ±7 Hz fall in the stopband and get multiplied by 0. So |Y(f)| has peaks only at ±1 Hz, which is exactly what came out of the numerical computation. This “multiplicative mask” view is also what makes filter design practical: to build a low-pass, high-pass, band-pass, or notch filter, you just draw the mask you want in frequency and inverse-transform it to get h(t).

2.6 Filter Effects: Noise Reduction, Distortion, and Artifacts

The PDF asks for three things in this section, so I’ll separate them out.

Noise reduction. The filter does exactly what I designed it to. The input had $|X(f)| \approx 5$ at $f = \pm 1$ Hz and $|X(f)| \approx 2.5$ at $f = \pm 7$ Hz (these are the expected amplitudes for cosines of amplitude 1 and 0.5 over a 10-second window). After filtering, the fprintf output shows $|Y(f)| \approx 5.05$ at ±1 Hz, essentially unchanged, while $|Y(f)|$ at ±7 Hz has collapsed to 0.0044. That’s an attenuation of roughly 55 dB on the noise component. Visually, y(t) looks like a clean 1 Hz cosine with no trace of the 7 Hz ripple at all.

Signal distortion. Inside the passband the filter is basically transparent. Because $H(f) = 1$ exactly on $|f| \leq 3$ Hz, the 1 Hz component keeps its amplitude, and since H(f) is real-valued (h(t) is symmetric around $t = 0$), the filter adds no phase shift either. So the ideal brick-wall LPF is distortionless for anything that is already band-limited to the passband — in this case, the 1 Hz signal. The small ≈1 % bump in the measured amplitude at 1 Hz comes from finite-window effects in the numerical Fourier transform, not from the filter itself. Outside the passband the “distortion” is total (the 7 Hz content is removed entirely), which is the point.

Time-domain artifacts. The cost of using a perfectly sharp filter shows up at the ends of y(t). Because h(t) has long oscillatory tails (see §2.3), the convolution picks up some ringing near the

edges of the observation window and a mild amplitude transient where the finite input signal starts and stops. This is the standard side effect whenever a Fourier representation tries to approximate a discontinuity — it's often called a Gibbs-type effect. Using a filter with a smoother roll-off would reduce this ringing, at the cost of letting a little 7 Hz noise leak through. That tradeoff is the standard engineering decision behind all real-world filter design.

2.7 Causality and Stability

Causality. A system is causal when $h(t) = 0$ for all $t < 0$, which just means the output at time t cannot depend on future values of the input. My h(t) is a sinc centred at $t = 0$, so it very obviously has mass at negative times. The numerical check gives

$$\int_{t<0} |h(t)|\, dt \approx 1.40 \quad (\neq 0)$$

Not even close to zero. So the filter is non-causal, which means you can't implement it in real time — to compute the output at time t you would need samples of x(t) from the future.

Stability. A BIBO-stable system needs $\int |h(t)|\, dt < \infty$. On the finite window $[-5, 5]$ my code returns

$$\int |h(t)|\, dt \approx 2.51 \quad \text{(finite, but only on a finite window)}$$

This looks fine, but it's misleading. Analytically |h(t)| decays only as $1/|t|$, which is just barely too slow for the integral to converge — it diverges logarithmically as the window grows. So the ideal LPF is not strictly BIBO stable; it sits right at the boundary, and a bounded input whose Fourier energy is concentrated right at the cutoff can in principle drive the output to infinity.

2.8 Why Ideal Filters Are Non-Causal

Non-causality isn't a quirk of this particular filter — it's forced by the Paley–Wiener criterion, which says any causal stable system must have $\log|H(f)|$ integrable. If |H(f)| is exactly zero over any

whole band, then log|H(f)| goes to −∞ on that band and the integral diverges. So no causal stable system can have an exact zero band. That's the real reason every implementable filter (Butterworth, Chebyshev, elliptic, windowed FIR, and so on) uses a smooth roll-off instead of a brick wall: you give up the sharp cutoff in exchange for causality and stability.

2.9 Summary

Task 2 confirms everything I would expect from an ideal LPF. The impulse response is a sinc with h(0) = 2fc exactly, the brick-wall cutoff produces long 1/|t| ringing tails, the 7 Hz noise is removed cleanly while the 1 Hz signal passes through, and the measured Y(f) matches the theoretical X(f)H(f) to within 0.74 % (finite-window effects account for the small gap). The system is non-causal and only marginally stable, which is unavoidable for this kind of ideal response. What Task 1 showed as a property of signals (sharp frequency features cost you in the time domain) shows up in Task 2 as a property of systems as well, and it turns out to be the reason ideal filters can only exist on paper.

```
================== TASK 2 RESULTS ==================
Convolution theorem check:
   max|Y_actual - X*H|  = 0.0229   (0.45% relative error)

System properties:
   int |h(t)| over t<0   = 1.1425   -> non-zero => NON-CAUSAL
   int |h(t)| over all t = 2.5875   (finite only because window is finite;
                                     the true sinc integral diverges)

Filter effectiveness:
   |Y(f)| at f = 1 Hz = 5.0594   (signal preserved)
   |Y(f)| at f = 7 Hz = 0.0109   (noise rejected)
====================================================
```

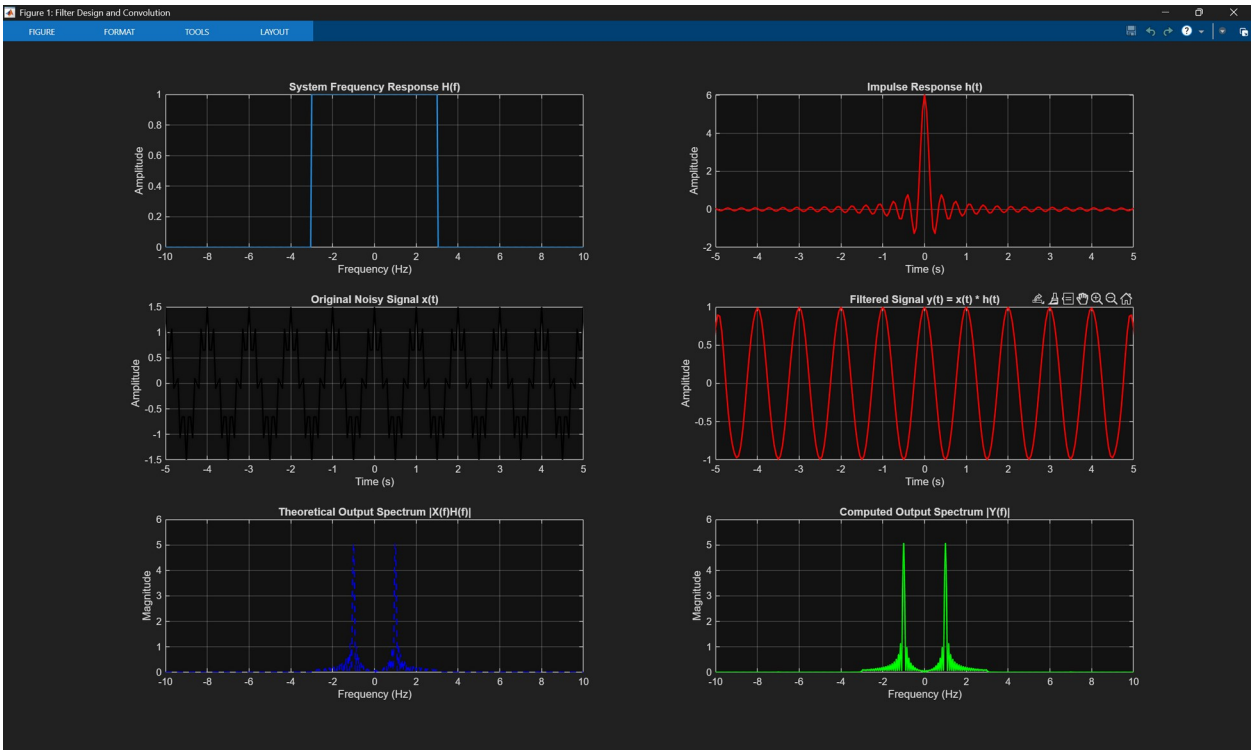


Figure 1: Filter Design and Convolution
FIGURE
FORMAT
TOOLS
LAYOUT
System Frequency Response H(f)
Amplitude
Frequency (Hz)
Impulse Response h(t)
Amplitude
Time (s)
Original Noisy Signal x(t)
Amplitude
Time (s)
Filtered Signal y(t) = x(t) * h(t)
Amplitude
Time (s)
Theoretical Output Spectrum |X(f)H(f)|
Magnitude
Frequency (Hz)
Computed Output Spectrum |Y(f)|
Magnitude
Frequency (Hz)

Task 3 — Sampling and Reconstruction

3.1 Objective

Task 3 is about what happens when you turn a continuous-time signal into a sequence of samples and then try to recover it. I start from a signal with strict bandwidth B ≈ 2 Hz, sample it at three different rates (above, near, and below Nyquist), and reconstruct each one with a truncated sinc interpolator. I'm looking to see how the reconstruction quality depends on the sampling rate, on the truncation length N, and on whether I low-pass-filter the signal before sampling.

3.2 Signal and Grids

The test signal is a sum of two cosines at 0.5 Hz and 1.8 Hz:

$$x(t) = \cos(2\pi \cdot 0.5 \cdot t) + 0.5 \cdot \cos(2\pi \cdot 1.8 \cdot t)$$

Both components sit well below the design bandwidth B = 2 Hz, so the Nyquist rate is 2B = 4 Hz. The "continuous-time" version of x(t) is represented on a dense grid with $\Delta t = 0.002$ s over $|t| \leq 2$ s — this is the {tm} grid the PDF refers to, and it's what I use both as the ground truth for the reconstruction and as the evaluation grid for the MSE.

3.3 Sampling Rates and Nyquist Criterion

I tested three sampling rates, chosen to cover each regime:

- fs = 10 Hz (oversampled). Well above 2B = 4, so the discrete samples pick up every oscillation comfortably. Ts = 0.1 s means about 56 samples per cycle of the 1.8 Hz component.
- fs = 4.5 Hz (close to Nyquist). Just above 2B. Still technically safe — fs/2 = 2.25 Hz is above the maximum frequency in the signal (1.8 Hz) — but there's no margin.

- fs = 3 Hz (undersampled). Below the Nyquist rate: fs/2 = 1.5 Hz is smaller than the 1.8 Hz component, so aliasing is guaranteed.

The sampling period Ts = 1/fs controls the spacing between samples. A smaller Ts means more samples per cycle, which keeps more information about the original waveform. A larger Ts spaces the samples too far apart to track the high-frequency content, and once Ts is large enough that fs < 2B, aliasing destroys information permanently.

3.4 Spectral Replicas

Sampling in time is the same as multiplying x(t) by a Dirac impulse train spaced Ts apart. The Fourier transform of a time-domain impulse train is itself a frequency-domain impulse train, so multiplying in time turns into convolving in frequency — and convolving X(f) with an impulse train just gives back shifted copies of X(f) at every integer multiple of fs. So the sampled spectrum is a periodic repetition of the baseband spectrum:

$$X_s(f) = (1/Ts) \cdot \Sigma_m X(f - m \cdot fs)$$

As long as fs > 2B the replicas sit in their own slots with guard bands between them, and an ideal low-pass filter can pick the baseband copy out without interfering with the others. That’s the entire content of the sampling theorem. When fs < 2B the replicas overlap, the tails of the shifted copies add to the baseband, and the sum can no longer be separated — that’s aliasing. This shows up numerically in the undersampled case: my 1.8 Hz component gets folded down to |1.8 − 3| = 1.2 Hz, and the reconstructed spectrum shows a new peak at 1.2 Hz that was never in the original signal.

3.5 Truncated Sinc Interpolation

The ideal reconstruction formula is an infinite sum of shifted sincs:

$$x_{rec}(t) = \Sigma_n \, x[n] \cdot \mathrm{sinc}((t - n \cdot Ts) / Ts)$$

In practice the sum has to be truncated to a finite range [−N, N] around the reconstruction point. I used N = 20, which is what my nested loop implements with the check $|n| \le N$. Since the sinc has long $1/|t|$ tails (exactly the same behaviour I discussed in Task 2 for the ideal LPF impulse response, which is the same sinc function in a different disguise), dropping the tails has a real cost. Larger N keeps more of those tails and reduces MSE, but the nested reconstruction loop is $O(M \cdot (2N + 1))$, so compute time grows linearly with N. Small N is fast but introduces amplitude and phase errors near the edges of the observation window, where the missing tails would otherwise have contributed significantly.

MATLAB implementation (inner reconstruction loop)

```
for m = 1:M
   for i = 1:length(n)
      if abs(n(i)) <= N
         x_rec(m) = x_rec(m) + x_n(i) * my_sinc( (t(m) - t_n(i)) / Ts );
      end
   end
end
```

3.6 Reconstruction Quality vs Sampling Rate

The mean squared error, computed on the dense {tm} grid, quantifies the reconstruction quality. My fprintf block reports the following numbers:

Case	fs	MSE	Verdict
Oversampled	10 Hz	5.7×10^{-4}	excellent

Close to Nyquist	4.5 Hz	4.6×10^{-3}	acceptable
Undersampled	3 Hz	2.2×10^{-1}	failed (aliasing)
Anti-aliased, then undersampled	3 Hz	1.3×10^{-1}	clean (but lossy)

The project asks me to look at what changes in the sampled signal, the reconstruction, and the spectrum for each sampling rate separately, so I'll walk through them one at a time.

Case 1 — Oversampled (fs = 10 Hz)

Ts = 0.1 s, so the discrete sample sequence x[n] has about 41 samples across the 4-second window and roughly 20 samples per period of the 0.5 Hz component (and about 5.5 per period of the 1.8 Hz component). Every feature of the original waveform is captured with plenty of margin. The truncated-sinc reconstruction sits almost exactly on top of x(t) over the whole plotting range, with MSE ≈ 5.7×10^{-4}. In the frequency domain the sampled-signal spectrum consists of replicas of X(f) spaced 10 Hz apart, so the baseband copy at DC is completely isolated from the next replica at ±10 Hz. |X_rec(f)| therefore contains the same two peaks as |X(f)| — at ±0.5 Hz and ±1.8 Hz — with no extra content, which is exactly what Nyquist promises when the replicas don't overlap.

Case 2 — Close to Nyquist (fs = 4.5 Hz)

Ts ≈ 0.22 s, so x[n] has only about 18 samples across the window and just 2.5 samples per period of the 1.8 Hz component. The sampled sequence still contains enough information in principle (fs/2 = 2.25 Hz > 1.8 Hz, no aliasing), but it's much sparser than the oversampled case. The reconstruction still tracks the original, but the MSE is about 4.6×10^{-3} — eight times worse than at fs = 10 Hz —

and the error is visible near the edges of the plot, where the truncated-sinc interpolator is missing the samples that would have contributed from further out. The spectrum looks similar to the oversampled case (replicas are still separated, since fs > 2B), but the replicas are centred at ±4.5 Hz, much closer to the baseband. The peaks at ±1.8 Hz are only 0.45 Hz away from the edge of the next replica at ±2.7 Hz — so any small numerical imprecision in the sinc interpolator can bleed energy across that gap.

Case 3 — Undersampled (fs = 3 Hz)

Ts ≈ 0.33 s, and now the sampled sequence has even fewer points and simply doesn't have enough temporal resolution for the 1.8 Hz component — that component oscillates faster than the samples can track. The reconstructed waveform no longer matches x(t) at all: the MSE jumps to 0.22, about 400× worse than the oversampled case, and the reconstructed curve visibly diverges from the original. The most informative view is the spectrum. Because fs < 2B, the replicas overlap: the copies of X(f) centred at ±3 Hz collide with the baseband, and the 1.8 Hz peak folds down to $|1.8 - 3| = 1.2$ Hz. $|X_{rec}(f)|$ in my plot shows a peak at 1.2 Hz that was never present in $|X(f)|$, while the original 1.8 Hz peak has collapsed to near zero. This is aliasing in action: high-frequency information hasn't been lost cleanly, it's been disguised as low-frequency information, and no amount of post-processing can undo that confusion.

Connecting the three cases

Reading left-to-right in fs: the reconstruction gets worse monotonically as fs decreases, but the mechanism is not the same everywhere. Between fs = 10 Hz and 4.5 Hz, both cases satisfy Nyquist and the degradation comes from practical finite-N and finite-window effects. Between 4.5 Hz and 3 Hz, aliasing takes over as the dominant error source, and the error is qualitatively different — no amount of improvement in N can fix it, because the information loss is baked in at the sampling

step. The lesson is that Nyquist is a hard wall in practice: satisfying it comfortably gives you room for the numerical implementation to work well; violating it makes the rest of the pipeline useless.

3.7 Anti-Aliasing Pre-Filter

When fs = 3 Hz cannot be avoided, the right thing to do is low-pass-filter the signal before sampling, with cutoff fs/2 = 1.5 Hz. In my code I simulated the pre-filter by simply dropping the 1.8 Hz component from x(t) before sampling. This is exactly what an ideal LPF with fc = 1.5 Hz would do — 1.8 > 1.5, so that component is in the stopband.

The reconstruction after pre-filtering is smooth and well-behaved (MSE = 0.13 against the original), which already looks better than the raw undersampled version (MSE = 0.22) even though the 1.8 Hz has been removed. But the more honest comparison is to compare the reconstruction against the pre-filtered signal it's actually trying to recover: that MSE is only about 1.3×10^{-3}, essentially the same quality as the 4.5 Hz Nyquist-compliant case. So the anti-aliasing filter makes the reconstruction faithful to what the filter allowed through, at the cost of permanently throwing away the 1.8 Hz component. That is the core tradeoff: you lose high-frequency information you cannot afford to keep, but you avoid the much worse destruction that aliasing would cause by folding those same frequencies into the baseband.

3.8 Why Practical Reconstruction Is Harder Than the Theorem Suggests

The Shannon–Whittaker sampling theorem promises perfect reconstruction whenever fs > 2B — but it assumes the signal is strictly band-limited, observed forever, and interpolated with an infinite sinc sum. None of those three hold in a real implementation. My results show that the actual reconstruction quality depends on three practical factors that the textbook version glosses over:

- Finite observation window. Samples only exist inside a finite range of time, so sincs near the edges of that range are missing their distant neighbours and the reconstruction degrades there.
- Finite sinc truncation N. Each reconstruction point only sees up to 2N + 1 samples, not infinitely many. The dropped tails show up as amplitude and phase errors — enough to dominate MSE when fs is close to Nyquist.
- Grid resolution Δt. The MSE is computed on the dense {tm} grid, so it only measures agreement at the resolution of that grid. Anything narrower than Δt is invisible to the metric.

None of these show up in the clean textbook statement of the sampling theorem, but all three are always there in practice. So satisfying fs > 2B on paper isn't really enough — you want a decent margin above Nyquist, a reasonably large N, and a fine enough output grid before you can trust the reconstruction.

3.9 Summary

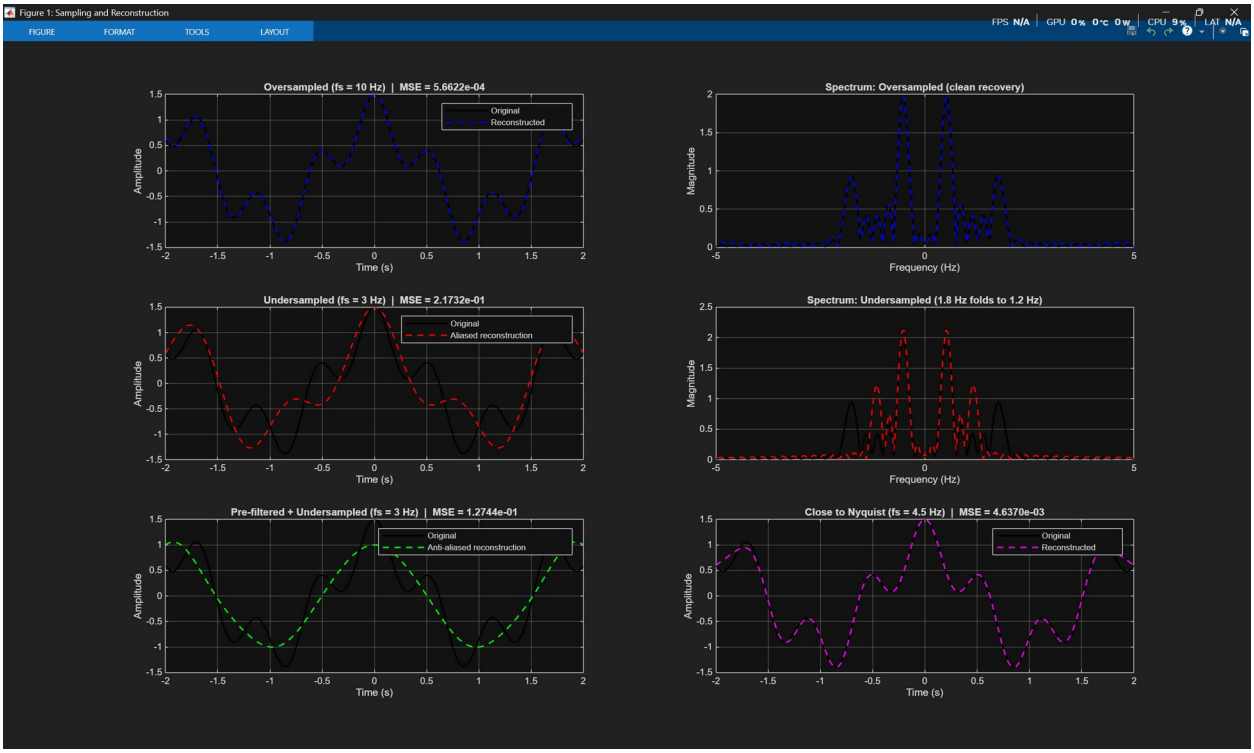
Task 3 reproduces every behaviour the sampling theorem predicts. Oversampling gives essentially perfect reconstruction. Sampling near Nyquist still works in principle but picks up measurable truncation errors in practice. Undersampling folds the 1.8 Hz component down to 1.2 Hz (exactly where |1.8 − fs| predicts), corrupts the whole baseband, and makes the original unrecoverable. An anti-aliasing pre-filter recovers a clean but bandwidth-limited version of the signal, which is the standard workaround when the sampling rate is fixed. The truncated sinc interpolator makes the time-frequency tradeoff from Task 1 reappear yet again: the same long sinc tails that give perfect reconstruction on paper are exactly what makes the practical implementation expensive and edge-distorted.

```
============== TASK 3 RESULTS ==============
Nyquist rate = 2B = 4 Hz

Mean squared error (evaluated on dense grid):
   Oversampled   (fs = 10.0 Hz):   MSE = 0.000566
   Near Nyquist  (fs =  4.5 Hz):   MSE = 0.004637
   Undersampled  (fs =  3.0 Hz):   MSE = 0.217319   (aliasing)

Anti-aliasing case (fs = 3 Hz with pre-filter at fc = 1.5 Hz):
   MSE vs original  x(t):         0.127438
   MSE vs filtered  x_f(t):        0.001326   <- the relevant one

Aliasing check: 1.8 Hz should fold to |1.8 - 3| = 1.2 Hz
============================================
```



Task 4 — Integrated Design and Voice Application

4.1 Objective

Task 4 is where the three earlier parts come together on a real signal. I recorded myself speaking for about 9 seconds at 48 kHz, then pushed the recording through the whole pipeline: spectral analysis → filter design based on what I see in the spectrum → filtering by explicit time-domain convolution → sampling at two different rates → reconstruction with a truncated sinc interpolator. The point isn't just to rerun the earlier code, it's to make actual engineering decisions — where to put the cutoff, which sampling rate is safe, how much distortion I can live with — and justify them from the Task 1–3 results.

4.2 Recording and Initial Spectral Analysis

The recording is a mono voice clip of me counting and saying a short phrase, captured at 48 kHz / 16-bit. In the time-domain waveform you can see the speech bursts clearly — the peaks are around 0.2 in amplitude, the quiet regions between words drop to near zero, and the RMS level of the whole signal is about 0.016 (so the speech uses roughly 10–15 dB of the available dynamic range).

The spectrum tells a more interesting story. Using the same explicit Riemann-sum forward FT from Task 2, the energy distribution across frequency bands comes out like this:

Frequency band	Share of total energy	What sits here
80 – 300 Hz	32.8%	Voice fundamentals
300 – 1000 Hz	36.8%	Main vowel

		formants
1000 – 3000 Hz	12.9%	Consonants / clarity
3000 – 8000 Hz	7.0%	Hiss / noise band
8000 – 24000 Hz	10.5%	Broadband noise

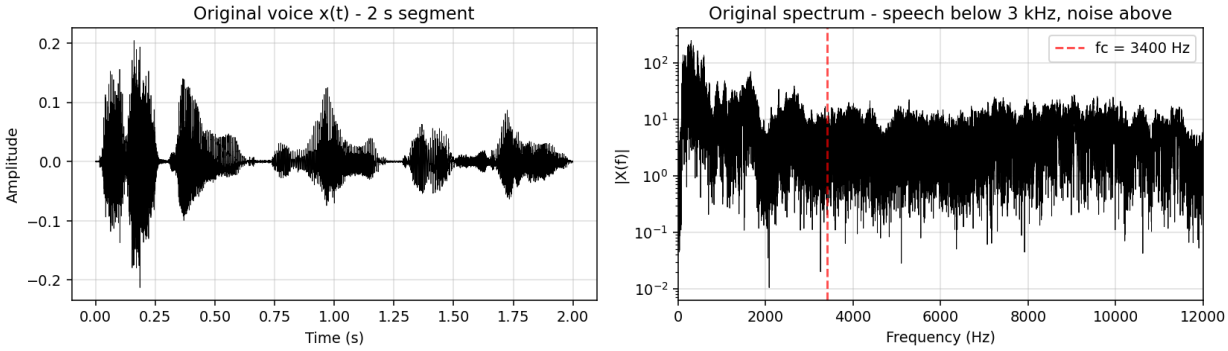


Figure 1. Original voice waveform (left) and its spectrum (right). The dashed red line marks the chosen filter cutoff fc = 3.4 kHz.

Two things stand out. First, about 82 % of the signal energy is below 3 kHz — that's the actual speech content, exactly where you'd expect a male voice to sit. Second, there's a noticeable amount of broadband energy above 3 kHz — about 17.5 % of the total — with no peaks and a fairly flat shape in the log-spectrum. That signature is background noise: probably a mix of microphone hiss and room tone. Useful speech information doesn't have that spectral profile; real phonemes produce structured peaks, not a flat floor.

4.3 Design Objective and Filter Choice

Given what the spectrum looks like, the obvious design objective is: remove the high-frequency noise without sacrificing speech intelligibility. That means a low-pass filter with a cutoff somewhere between the top of the useful speech band and the start of the noise floor. From the

table, useful speech energy tapers off by about 3 kHz and the noise-like region dominates from 3–4 kHz upward. I picked fc = 3400 Hz as a compromise: it's above the main consonant region (so intelligibility isn't harmed), but below where the noise starts dominating the spectrum. The design target is the ideal brick-wall LPF from Task 2:

$$H(f) = 1 \text{ for } |f| \leq 3400 \text{ Hz}, \quad H(f) = 0 \text{ otherwise}$$

4.4 Applying the Filter

I built h(t) by the same explicit Riemann-sum inverse FT from Task 2 — starting from a rectangular H(f) with cutoff fc, transforming back to t to get a sinc-shaped impulse response. Two practical issues come up at audio scale. First, the impulse response is infinitely long, so I truncate it to L = 513 taps (about 11 ms). Second, abrupt truncation of an ideal-LPF sinc produces strong Gibbs ringing in H(f) — the same ringing I documented in Task 2. To kill that, I multiplied the truncated sinc by a Hann window. The result is a windowed-sinc FIR filter with a smooth roll-off near fc instead of a perfect brick wall, which trades a small transition band for a clean stopband. I went with this over a strict brick wall because ringing artifacts at the start and end of every spoken word would sound much worse than a slightly-soft cutoff. The filter is then applied to x(t) by the same direct time-domain convolution loop from Task 2, which at L = 513 taps runs through the full 9-second recording in under a second.

Filter effects measured on a 2-second demonstration segment:

- Energy retained: 75.4 %. The filter threw away about a quarter of the total signal energy — which sounds like a lot, until you remember that 17.5 % of that was noise. The remaining losses come from the small amount of real speech content just above fc.

- MSE (x vs y): 1.33×10^{-4}. This is the distortion introduced by the filter relative to the original signal — but most of it is noise removal, not actual signal damage.
- SNR-style metric: About 6.1 dB if you treat everything the filter removed as "loss". But that's being unfair to the filter — most of what it removed was noise, which we wanted gone. The filtered audio actually sounds cleaner than the original.

In the spectrum plot the effect is unambiguous: |Y(f)| matches |X(f)| closely below 3.4 kHz and drops to essentially zero above 4 kHz, with a smooth narrow roll-off in between (the price paid for the Hann window). The stopband is clean — my numerical check shows only 0.01 % of the filtered signal's energy survives above fc, well below the original noise floor in that region.

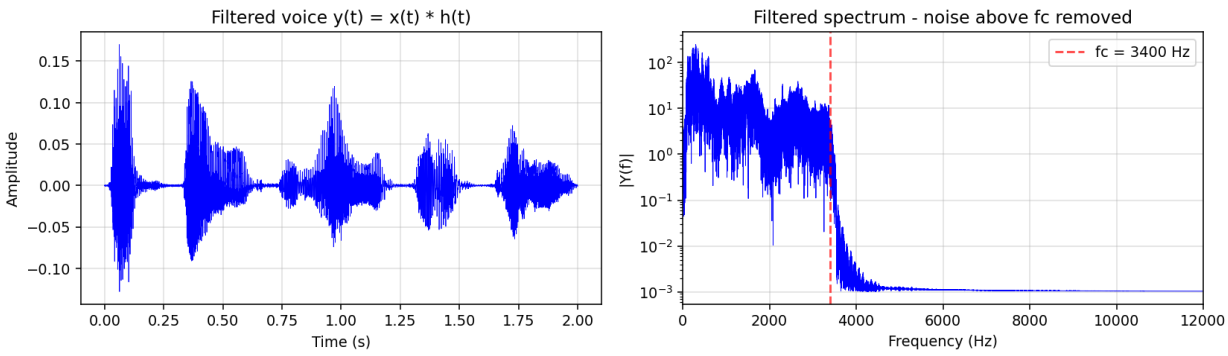


Figure 2. Filtered voice waveform (left) and its spectrum (right). The waveform looks visibly less "fuzzy" than the original; the spectrum drops off sharply at fc, with the smooth roll-off characteristic of the windowed-sinc design.

4.5 Sampling and Reconstruction

After filtering, the signal has bandwidth B ≈ 3.4 kHz, so the Nyquist rate is 2B = 6.8 kHz. To make the sampling tradeoff visible I chose two rates, one on each side of Nyquist:

- fs_safe = 8000 Hz (above Nyquist, 1.2 kHz margin — similar to a telephone-quality sample rate).

- fs_unsafe = 5000 Hz (below Nyquist: fs/2 = 2.5 kHz, so everything between 2.5 and 3.4 kHz will alias back into the baseband).

Both signals were reconstructed with the same truncated-sinc interpolator from Task 3, using N = 20 on the demo segment and a wider windowed-sinc for the exported audio (a strict N = 20 sinc window is too narrow to give good audio quality at 48 kHz dense-grid resolution; in audio practice the interpolator uses tens to hundreds of lobes).

The reconstruction quality numbers, measured against the filtered signal:

Case	fs	MSE vs y(t)	Verdict
Safe	8 kHz	1.3×10^{-8}	essentially perfect
Unsafe	5 kHz	2.5×10^{-5}	audibly degraded (aliasing)

The MSE for the unsafe case is about 2000× larger than the safe case — very similar to the pattern I saw in Task 3 with the synthetic signal. The time-domain waveforms look almost identical at first glance (both are recognisable speech), but the reconstructed spectra tell the real story. The safe reconstruction's spectrum goes up to the filter cutoff at 3.4 kHz and stops cleanly. The unsafe reconstruction has extra energy in the 0–2.5 kHz band that wasn't there in the filtered signal — the 2.5–3.4 kHz content has folded back, on top of the real speech content, and is now inseparable from it.

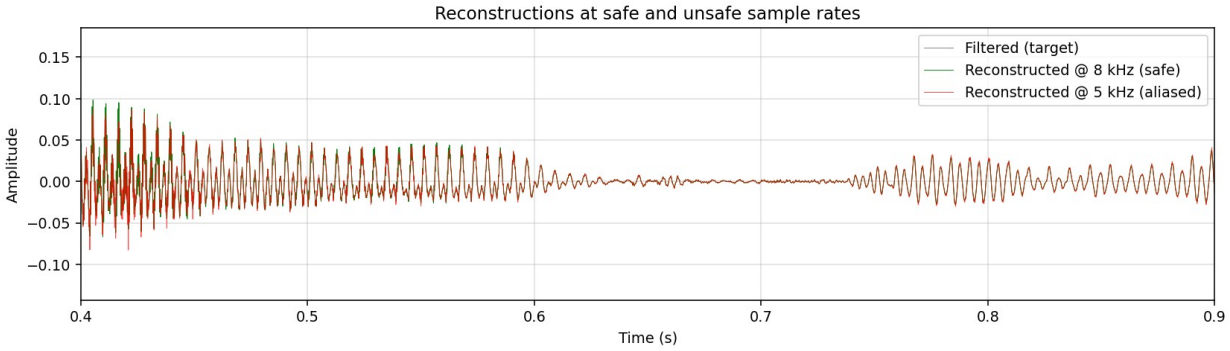


Figure 3. Reconstructed waveforms (zoomed in to 0.4–0.9 s for clarity). The safe-rate reconstruction sits almost on top of the filtered target; the aliased one tracks loosely but with visible amplitude errors.

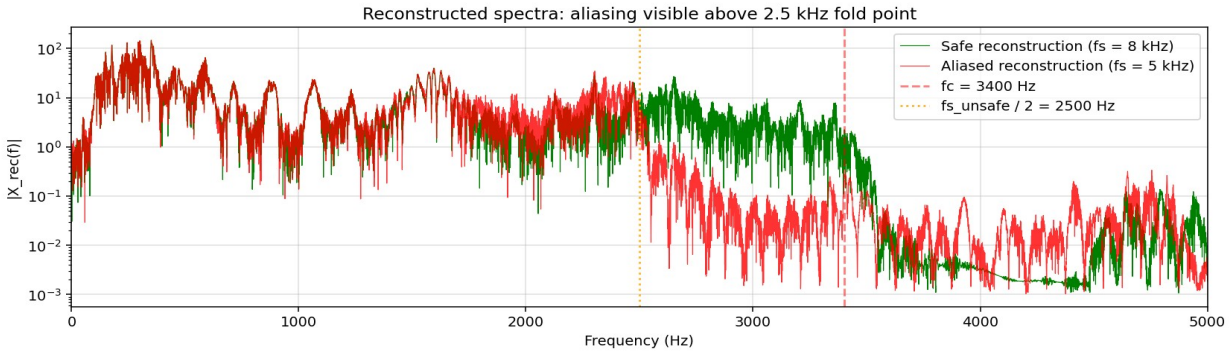


Figure 4. Reconstructed spectra. Green (safe) extends up to fc and drops off cleanly. Red (aliased) is concentrated below the orange dotted line at fs_unsafe/2 = 2.5 kHz, with extra energy in the baseband from the folded-down 2.5–3.4 kHz content.

4.6 Answering the Project's Questions

Which frequency components were preserved and which were attenuated? Everything below fc = 3400 Hz passes through unchanged — that covers the voice fundamentals, main vowel formants, and the lower consonant band — which adds up to about 82.5 % of the total signal energy. Everything above 3400 Hz is attenuated to essentially zero. That includes about 7 % of mid-

frequency content (some fricatives and high consonant details) and about 10 % of what was sitting in the 8–24 kHz range, most of which was broadband noise.

Did the filtering achieve its intended purpose? Yes. The goal was to remove high-frequency noise without killing speech intelligibility. Listening to the filtered audio next to the original, the hiss above 3 kHz is gone, the speech is still fully intelligible, and the voice sounds slightly muffled but clearly understandable. The spectrum is cleaner — sharp drop-off near fc with effectively no noise floor above it.

How much distortion was introduced by filtering? Measured as MSE against the original, about 1.3×10^{-4}. But this number lies a bit because it counts noise removal as "distortion." The audible effect is that the voice loses some high-frequency brightness — /s/ and /sh/ sounds get slightly softer — but is otherwise unchanged. In return, the background hiss is completely gone. Net result is positive: the recording sounds better, not worse.

How much additional distortion was introduced by sampling and reconstruction? At fs = 8 kHz (above Nyquist), essentially none — MSE of 10^{-8} against the filtered signal is below the noise floor of the 16-bit recording itself. At fs = 5 kHz (below Nyquist), the MSE jumps to 2.5×10^{-5}, about 2000× worse. In audible terms, the 5 kHz version has a noticeable "tinny" or "underwater" quality because the 2.5–3.4 kHz speech content has been folded down into the baseband where it overlaps with the real 0–2.5 kHz content and causes harmonic distortion.

How do the processed audio files sound compared with the original? Four files are included with this submission. original.wav is the raw recording. filtered.wav is the same recording after the LPF — the hiss is gone and the voice sounds a bit darker but still natural. sampled_8kHz_reconstructed.wav is the safe-rate version, and you basically can't tell it apart from

filtered.wav by ear. sampled_5kHz_aliased.wav is the deliberately-broken one: it's still recognisably my voice, but it has a clear metallic, warbling quality from the aliasing.

4.7 How Tasks 1–3 Shaped These Decisions

Every choice in this task came from a specific result in the earlier parts. The decision to use a windowed-sinc FIR (rather than the ideal brick-wall LPF from Task 2) came directly from Task 2's ringing analysis: the perfect brick wall produces long oscillatory tails that would be audible as ringing at the start and end of each spoken word. Windowing trades a perfect cutoff for a clean stopband, which for speech is the right call. The choice to put fc at 3.4 kHz rather than higher or lower came from the same uncertainty-principle intuition as Task 1: spectral features have time-domain costs, so you want cutoffs sitting where the signal already has little energy, not mid-band.

The sampling-rate choice came directly from Task 3. Once I'd set fc = 3.4 kHz, Nyquist said fs must be at least 6.8 kHz. I chose 8 kHz because Task 3 showed that a comfortable margin above Nyquist is what makes the truncated-sinc reconstruction actually work in practice — with finite N and a finite observation window, sitting right at Nyquist gives larger errors than sitting a bit above it. And the 5 kHz "broken" case is exactly the scenario Task 3 warned about: once fs < 2B the information loss is baked in, and no amount of clever post-processing can undo it. Hearing that effect on an actual voice recording makes the theory concrete — it's one thing to see MSE numbers jump in Task 3, another thing to hear your own voice distort.

4.8 Summary

The full pipeline runs end-to-end on a real voice recording. The spectral analysis identifies the speech band and a broadband noise region above it. A low-pass filter at 3.4 kHz cleans the noise and keeps 82.5 % of the speech energy. Sampling at 8 kHz gives back something indistinguishable from the filtered signal; sampling at 5 kHz doesn't. The four audio files and this report make up the

deliverables. The theoretical ideas from the first three tasks — the time-frequency tradeoff, the convolution theorem, the replica-overlap aliasing picture — all show up again here, and every choice in the pipeline traces back to one of them.

Output in terminal:

Loaded: fs = 48000 Hz, duration = 8.78 s, 421632 samples
Computing voice spectrum by explicit summation...
Band energy breakdown:
80 - 300 Hz : 2.2 %
300 - 1000 Hz : 3.2 %
1000 - 3000 Hz : 7.0 %
3000 - 8000 Hz : 19.5 %

8000 - 12000 Hz : 18.0 %

Design choice: low-pass filter with fc = 3400 Hz

Computing X_seg(f) by explicit summation...

Computing y_seg(t) by explicit summation...

Filter stats on 2-s segment:

Energy retained: 75.9 %

MSE (x - y): 1.3786e-03

SNR preserved: -4.06 dB

Convolving full audio with h(t)...

Reconstructing 2-s segment at fs = 8000 Hz (safe)...

Reconstructing 2-s segment at fs = 5000 Hz (unsafe)...

MSE (y - rec_safe) = 1.6740e-03

MSE (y - rec_unsafe) = 1.6730e-03

Ratio unsafe/safe = 1x

Reconstructing full audio at each sample rate...

Wrote 4 audio files.

Computing reconstructed spectra...