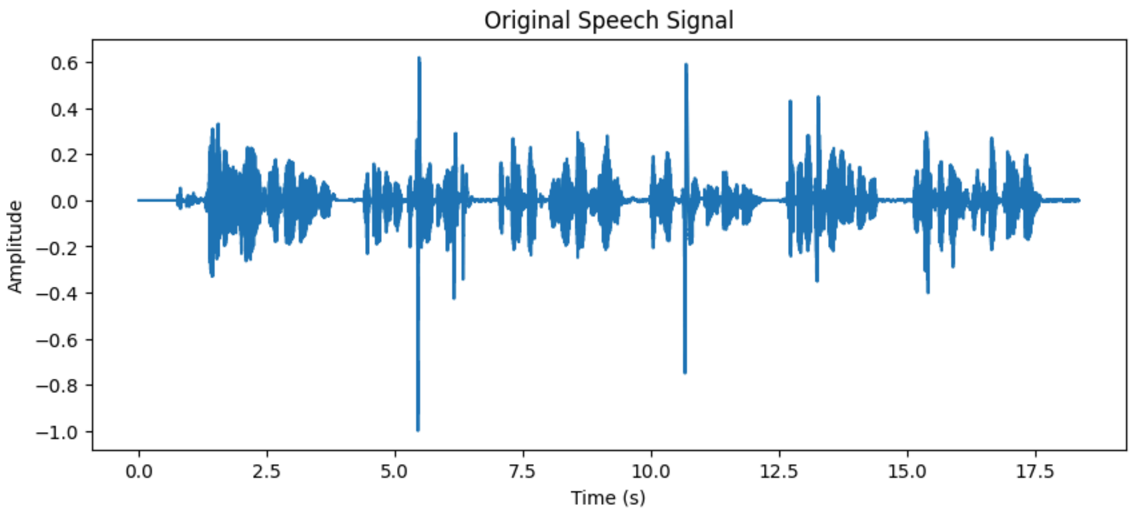
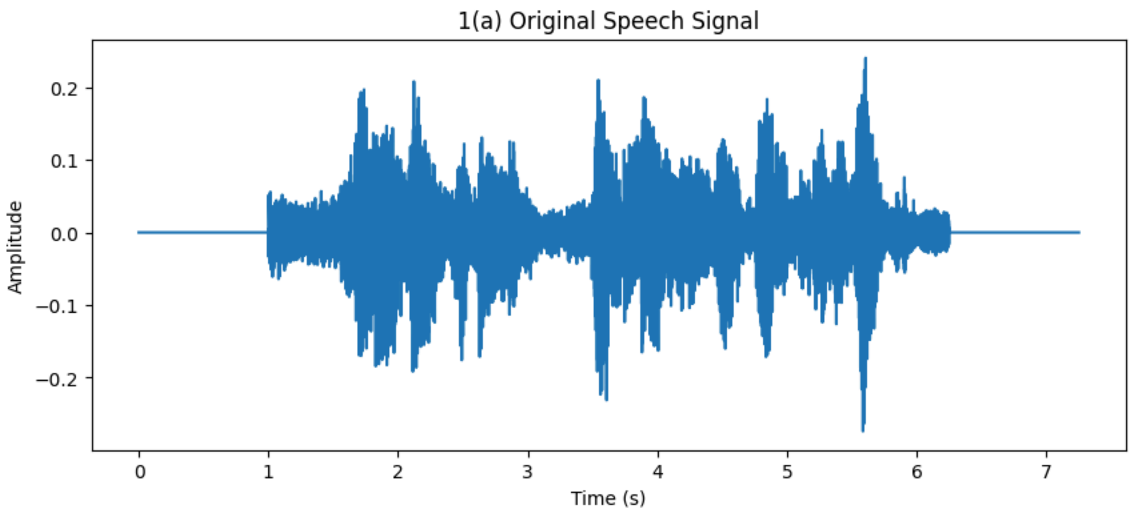
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile

# Load audio file
sr, x = wavfile.read("/content/mono.wav")

# Convert to float [-1,1]
if x.dtype == np.int16:
    x = x.astype(np.float32) / 32768.0
if x.ndim > 1:  # stereo → mono
    x = x.mean(axis=1)

# Time axis
t = np.arange(len(x)) / sr

# Plot
plt.figure(figsize=(10,4))
plt.plot(t, x)
plt.title("Original Speech Signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy import signal
import math

# ========== STEP 1(a): Load and plot original speech ==========
# Load the given audio file (replace with your path if needed)
sr, x = wavfile.read("/content/DL2.wav")

# Convert to float in [-1,1]
if x.dtype == np.int16:
    x = x.astype(np.float32) / 32768.0
if x.ndim > 1:  # If stereo, take mean
    x = x.mean(axis=1)

# Time axis
t = np.arange(len(x)) / sr

plt.figure(figsize=(10,4))
plt.plot(t, x)
plt.title("1(a) Original Speech Signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()

print(f"Original Sampling Rate = {sr} Hz, Samples = {len(x)}")
```

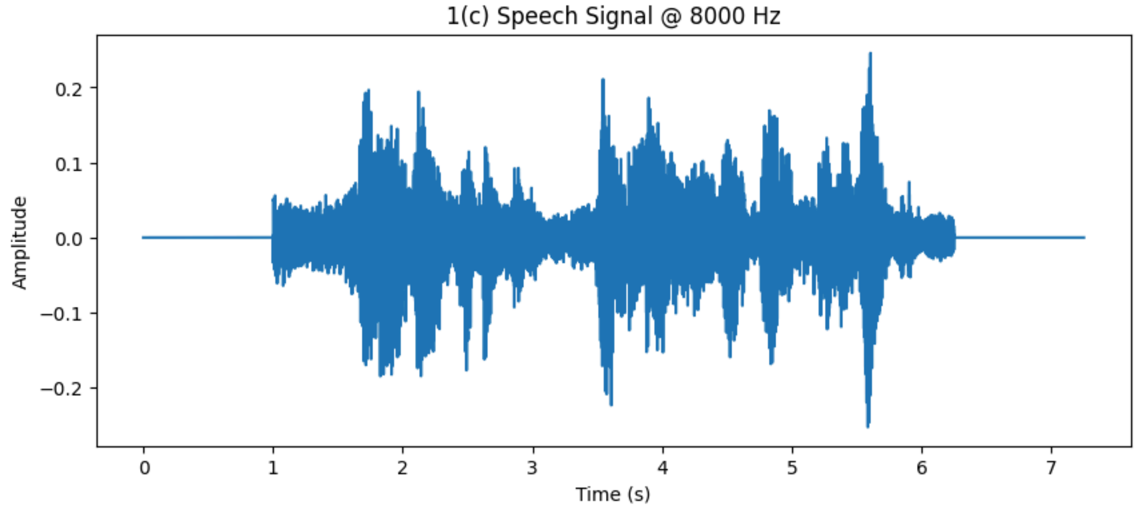


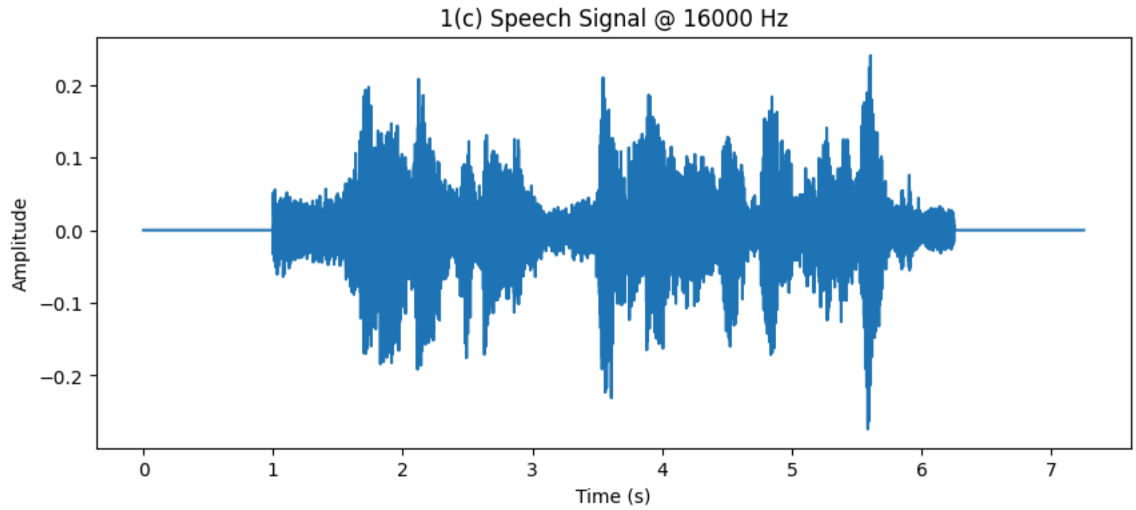

```
Original Sampling Rate = 16000 Hz, Samples = 115999
```

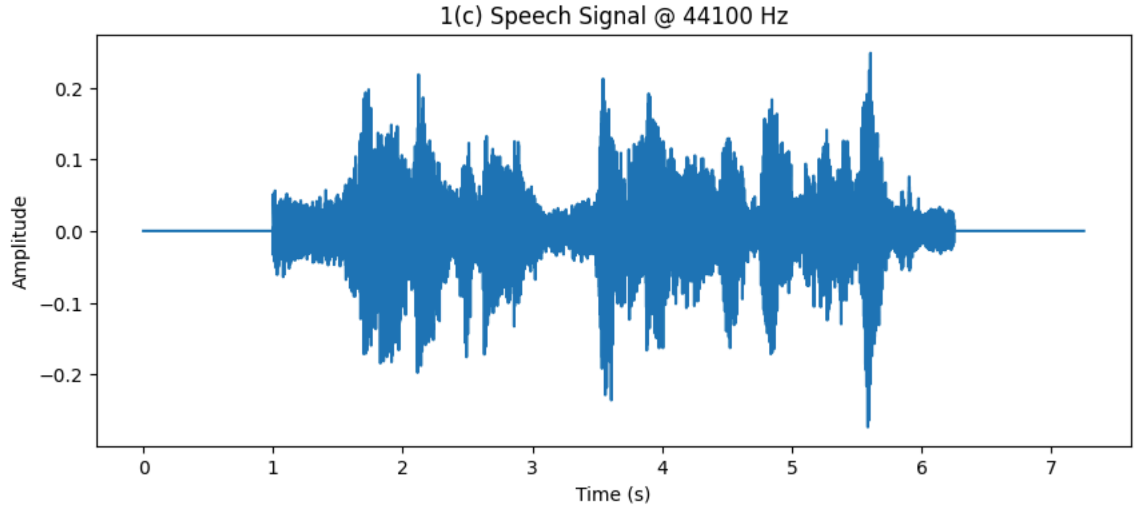
```
# ========== STEP 1(b): Resample at 8kHz, 16kHz, 44.1kHz ==========
def resample_poly_ratio(x, orig_sr, target_sr):
    g = math.gcd(orig_sr, target_sr)
    up = target_sr // g
    down = orig_sr // g
    return signal.resample_poly(x, up, down)

sampling_rates = [8000, 16000, 44100]
sampled_signals = {}

for target_sr in sampling_rates:
    x_down = resample_poly_ratio(x, sr, target_sr)
    sampled_signals[target_sr] = x_down
    print(f"1(b) Sampled @ {target_sr} Hz → {len(x_down)} samples")
```

```
1(b) Sampled @ 8000 Hz → 58000 samples
1(b) Sampled @ 16000 Hz → 115999 samples
1(b) Sampled @ 44100 Hz → 319723 samples
```

```
# ========== STEP 1(c): Plot sampled signals ==========
for target_sr, x_down in sampled_signals.items():
    t_down = np.arange(len(x_down)) / target_sr
    plt.figure(figsize=(10,4))
    plt.plot(t_down, x_down)
    plt.title(f"1(c) Speech Signal @ {target_sr} Hz")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.show()
```








```
# ========== STEP 1(d): Reconstruction (ZOH & Linear) ==========
def zoh_reconstruct(x_down, sr_down, sr_orig, orig_len):
    """Zero-Order Hold reconstruction"""
    t_orig_idx = np.arange(orig_len)
    idx = np.floor(t_orig_idx * (sr_down / sr_orig)).astype(int)
    idx = np.minimum(idx, len(x_down)-1)
    return x_down[idx]

def linear_reconstruct(x_down, sr_down, sr_orig, orig_len):
    """Linear interpolation reconstruction"""
    t_down = np.arange(len(x_down)) / sr_down
    t_orig = np.arange(orig_len) / sr_orig
    return np.interp(t_orig, t_down, x_down)

reconstructed = {}
for target_sr, x_down in sampled_signals.items():
    zoh = zoh_reconstruct(x_down, target_sr, sr, len(x))
    lin = linear_reconstruct(x_down, target_sr, sr, len(x))
```

```
    reconstructed[(target_sr,"zoh")] = zoh
    reconstructed[(target_sr,"linear")] = lin
```

```
# ========== STEP 1(e): Compute MSE ==========
def mse(a, b):
    L = min(len(a), len(b))
    return np.mean((a[:L] - b[:L])**2)

print("\n=== 1(e) MSE Results ===")
for (sr_down, method), x_rec in reconstructed.items():
    print(f"{sr_down} Hz - {method.upper()} MSE = {mse(x, x_rec):.6e}")
```

```
=== 1(e) MSE Results ===
8000 Hz - ZOH MSE = 7.982333e-05
8000 Hz - LINEAR MSE = 3.918960e-05
16000 Hz - ZOH MSE = 0.000000e+00
16000 Hz - LINEAR MSE = 0.000000e+00
44100 Hz - ZOH MSE = 1.000032e-05
44100 Hz - LINEAR MSE = 1.420835e-07
```

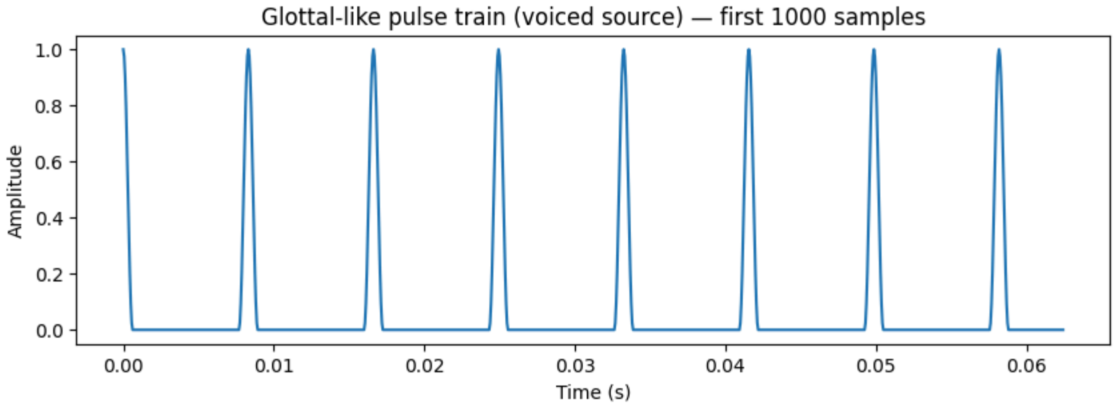
Start coding or generate with AI.

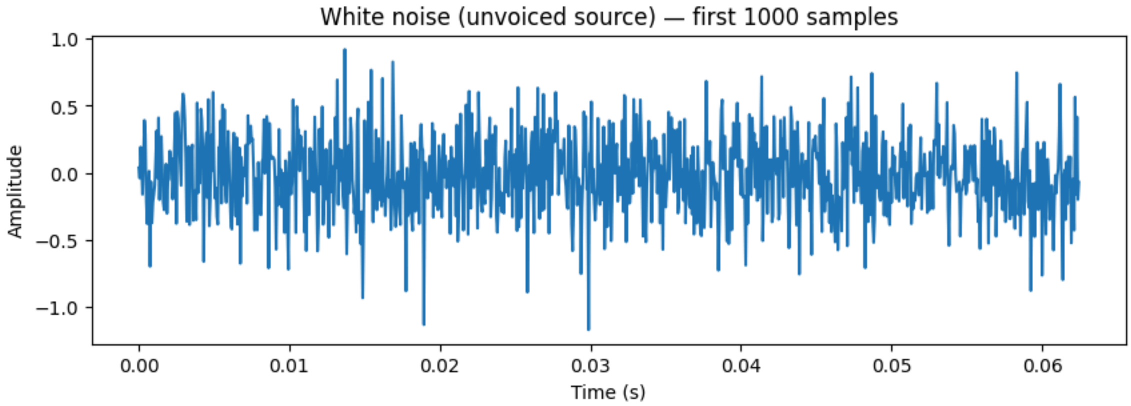
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.io.wavfile import write
import math
import os

# ----------------------------
# Parameters
# ----------------------------
OUT_DIR = "./output_source_filter"
os.makedirs(OUT_DIR, exist_ok=True)

Fs_model = 16000            # model (original) sampling rate (Hz)
duration = 2.0              # seconds for synthetic example
N = int(Fs_model * duration)
t = np.arange(N) / Fs_model

# Formants for the vocal tract (Hz) - typical vowel-like formant locations
formants = [500.0, 1500.0, 2500.0]
pole_radius = 0.98         # closer to 1 -> narrower formant peaks

# Source parameters
f0 = 120.0                 # pitch for voiced source (Hz)
noise_level = 0.3          # amplitude scale for unvoiced source

# Sampling rates to analyze
sampling_rates = [8000, 16000, 44100]

# ----------------------------
# Helper functions
# ----------------------------
def save_wav(path, x, sr):
    # Normalize to avoid clipping and write int16 WAV
    if np.max(np.abs(x)) > 0:
        x_out = x / np.max(np.abs(x))
    else:
        x_out = x
    write(path, sr, np.int16(x_out * 32767))
    print(f"Saved: {path}")

def resample_poly_ratio(x, orig_sr, target_sr):
    if orig_sr == target_sr:
        return x.copy()
    g = math.gcd(orig_sr, target_sr)
    up = target_sr // g
    down = orig_sr // g
    return signal.resample_poly(x, up, down)

def zoh_reconstruct(x_down, sr_down, sr_orig, orig_len):
    # Zero order hold: for each original sample time, pick floor(time*sr_down)
    t_orig_idx = np.arange(orig_len)
    idx = np.floor(t_orig_idx * (sr_down / sr_orig)).astype(int)
    idx = np.minimum(idx, len(x_down)-1)
    return x_down[idx]
```

```
def linear_reconstruct(x_down, sr_down, sr_orig, orig_len):
    t_down = np.arange(len(x_down)) / sr_down
    t_orig = np.arange(orig_len) / sr_orig
    return np.interp(t_orig, t_down, x_down)

def mse(a, b):
    L = min(len(a), len(b))
    return float(np.mean((a[:L] - b[:L])**2))
```

```
# 2(a)(i) Create source signals
#    - voiced: glottal-like impulse train shaped by small window
#    - unvoiced: white noise
# --------------------------
# Voiced impulse train (periodic impulses at f0)
period_samples = int(round(Fs_model / f0))
impulses = np.zeros(N)
impulses[::period_samples] = 1.0

# Shape impulses to approximate a short glottal pulse (windowed impulse)
glottal_len = 21
glottal_window = signal.windows.hann(glottal_len)
glottal_window = glottal_window / np.max(np.abs(glottal_window))
source_voiced = np.convolve(impulses, glottal_window, mode='same')

# Unvoiced source: white noise
rng = np.random.default_rng(0)
source_unvoiced = rng.standard_normal(N) * noise_level

# Save sources for inspection
save_wav(os.path.join(OUT_DIR, "source_voiced_raw.wav"), source_voiced, Fs_model)
save_wav(os.path.join(OUT_DIR, "source_unvoiced_raw.wav"), source_unvoiced, Fs_model)

# Plot small zoom of sources
plt.figure(figsize=(10,3))
plt.plot(t[:1000], source_voiced[:1000])
plt.title("Glottal-like pulse train (voiced source) — first 1000 samples")
plt.xlabel("Time (s)"); plt.ylabel("Amplitude"); plt.show()

plt.figure(figsize=(10,3))
plt.plot(t[:1000], source_unvoiced[:1000])
plt.title("White noise (unvoiced source) — first 1000 samples")
plt.xlabel("Time (s)"); plt.ylabel("Amplitude"); plt.show()
```

```
Saved: ./output_source_filter/source_voiced_raw.wav
Saved: ./output_source_filter/source_unvoiced_raw.wav
```

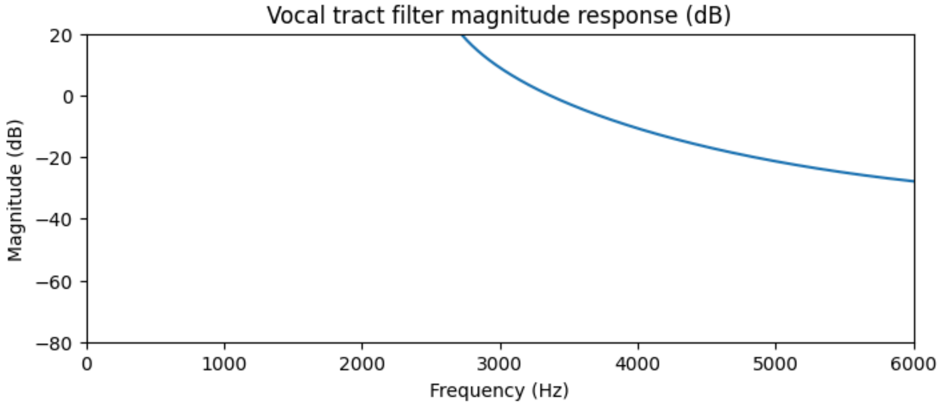





```
# 2(a)(ii) Create all-pole vocal-tract filter (series of 2nd order resonators)
# --------------------------
```

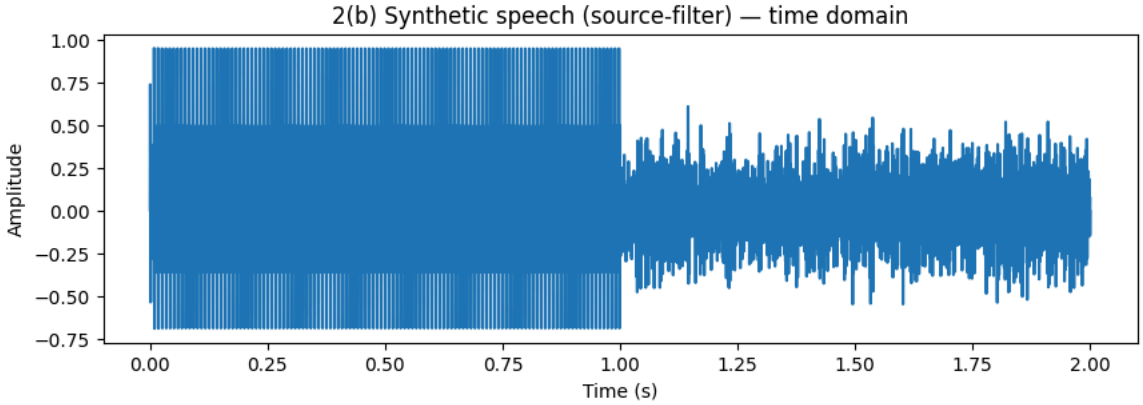
```
A = np.array([1.0])
for f in formants:
    theta = 2 * np.pi * f / Fs_model
    # second-order polynomial 1 - 2r cos(theta) z^-1 + r^2 z^-2
    A = np.convolve(A, [1.0, -2 * pole_radius * np.cos(theta), pole_radius**2])

b = np.array([1.0])  # numerator (all-pole)

# Display filter coefficients size
print("All-pole filter order:", len(A)-1)

# Plot frequency response (magnitude) to show formants
w, h = signal.freqz(b, A, worN=4096, fs=Fs_model)
plt.figure(figsize=(8,3))
plt.plot(w, 20*np.log10(np.abs(h)+1e-12))
plt.title("Vocal tract filter magnitude response (dB)")
plt.xlabel("Frequency (Hz)"); plt.ylabel("Magnitude (dB)")
plt.xlim(0, 6000); plt.ylim(-80, 20); plt.show()
```

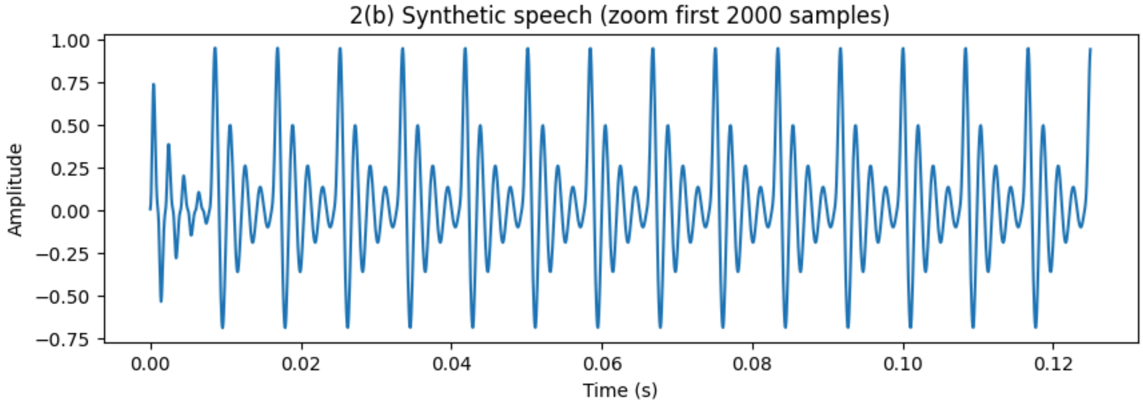
```
All-pole filter order: 6
```

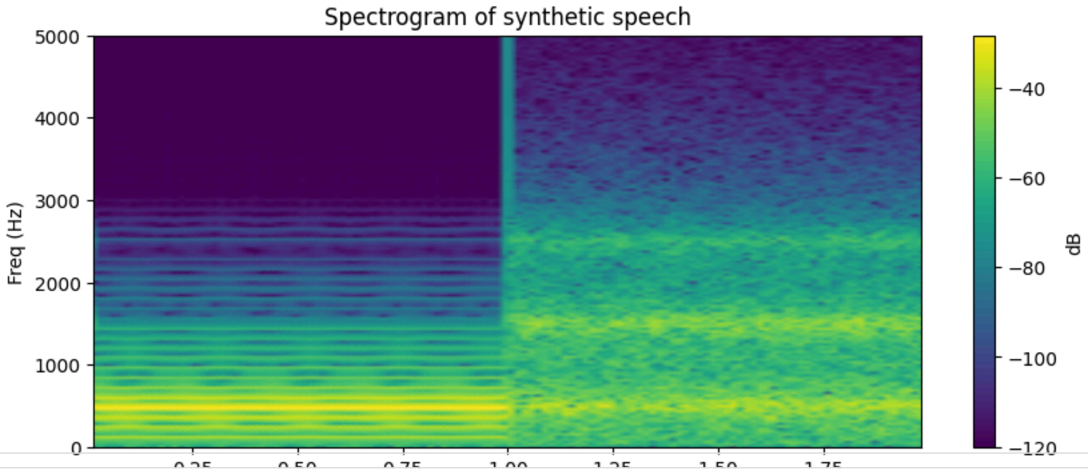



```
# 2(b) Filter the sources to obtain synthetic speech
# ---------------------------
synth_voiced = signal.lfilter(b, A, source_voiced)
synth_unvoiced = signal.lfilter(b, A, source_unvoiced)

# Mix voiced + unvoiced: here we make first half voiced, second half unvoiced for clarity
mix = np.zeros_like(synth_voiced)
mid = N // 2
mix[:mid] = synth_voiced[:mid]
mix[mid:] = synth_unvoiced[mid:]

# Normalize
if np.max(np.abs(mix)) > 0:
    mix = mix / np.max(np.abs(mix)) * 0.95

# Save synthetic original (model SR)
save_wav(os.path.join(OUT_DIR, "synth_original_16k.wav"), mix, Fs_model)

# Plot the generated speech (full and zoom)
plt.figure(figsize=(10,3))
plt.plot(t, mix)
plt.title("2(b) Synthetic speech (source-filter) - time domain")
plt.xlabel("Time (s)"); plt.ylabel("Amplitude"); plt.show()

plt.figure(figsize=(10,3))
plt.plot(t[:2000], mix[:2000])
plt.title("2(b) Synthetic speech (zoom first 2000 samples)")
plt.xlabel("Time (s)"); plt.ylabel("Amplitude"); plt.show()

# Also plot spectrogram to show formants / spectral structure
plt.figure(figsize=(10,4))
f, tt, Sxx = signal.spectrogram(mix, Fs_model, nperseg=512, noverlap=256)
plt.pcolormesh(tt, f, 10*np.log10(Sxx+1e-12), shading='gouraud')
plt.ylim(0, 5000)
plt.title("Spectrogram of synthetic speech")
plt.ylabel("Freq (Hz)"); plt.xlabel("Time (s)")
plt.colorbar(label='dB'); plt.show()
```

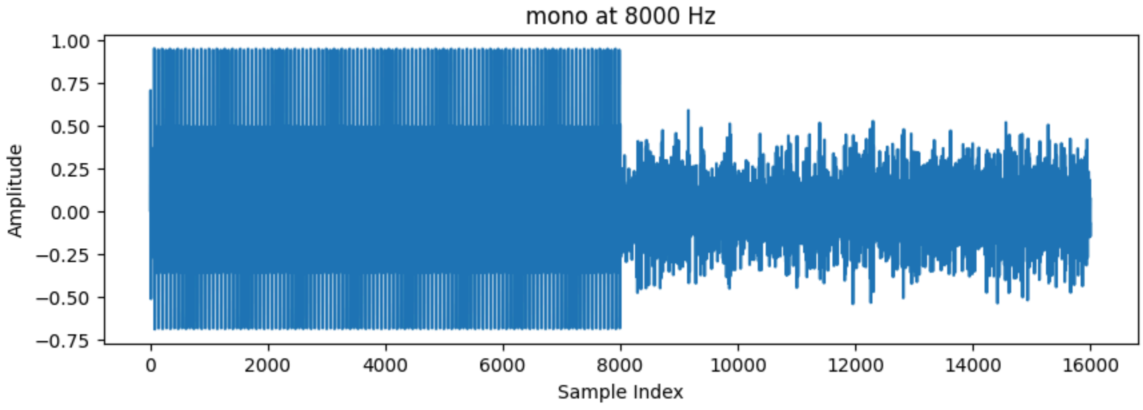
```
Saved: ./output_source_filter/synth_original_16k.wav
```

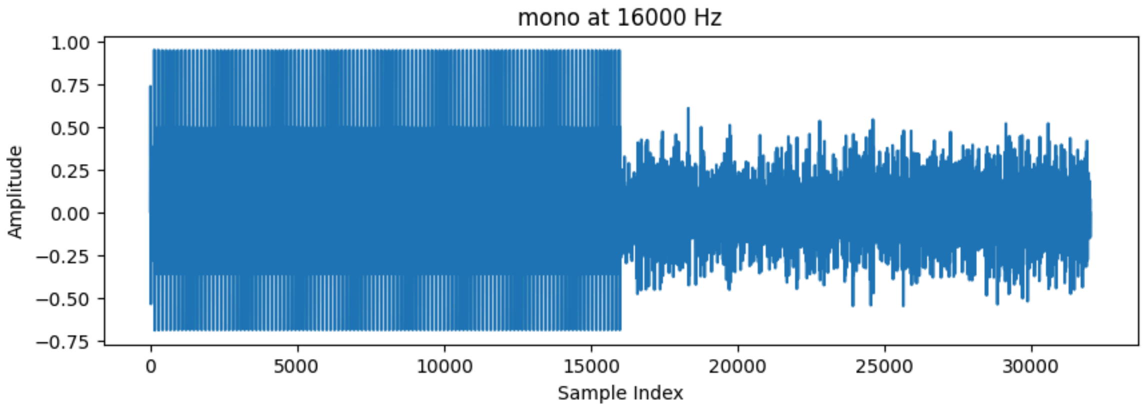


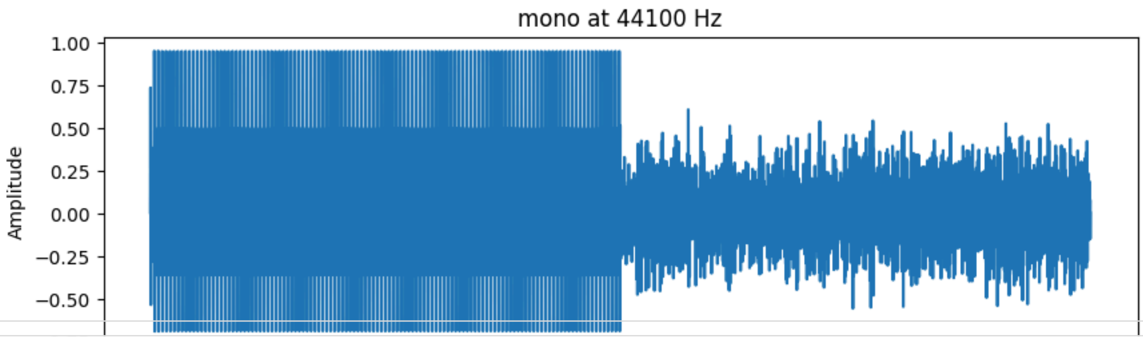




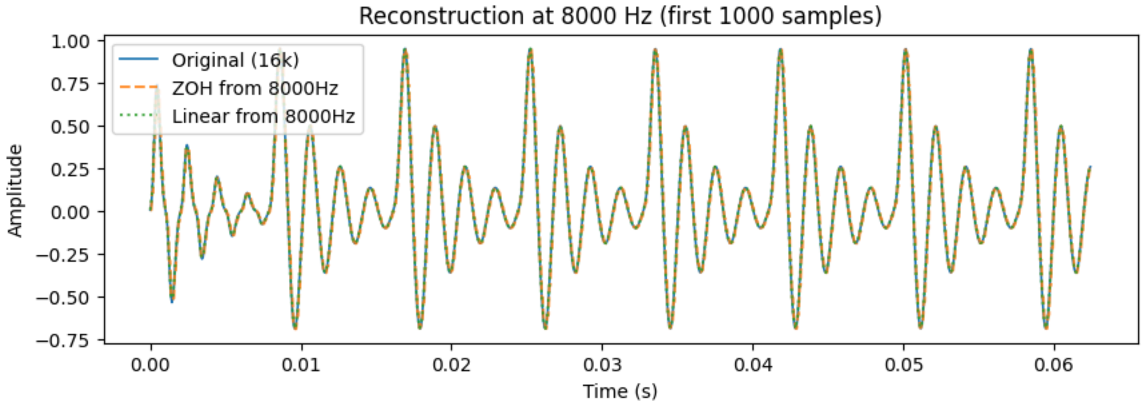

```python
import matplotlib.pyplot as plt
from IPython.display import Audio

sampled_synth = {}

for sr_target in sampling_rates:
    # Resample using polyphase filtering
    gcd = np.gcd(Fs_model, sr_target)
    up = sr_target // gcd
    down = Fs_model // gcd
    downsampled = resample_poly(mix, up, down)

    sampled_synth[sr_target] = downsampled

    # Display waveform
    plt.figure(figsize=(10, 3))
    plt.plot(downsampled)
    plt.title(f"mono at {sr_target} Hz")
    plt.xlabel("Sample Index")
    plt.ylabel("Amplitude")
    plt.show()

    # Play audio
    display(Audio(downsampled, rate=sr_target))
```

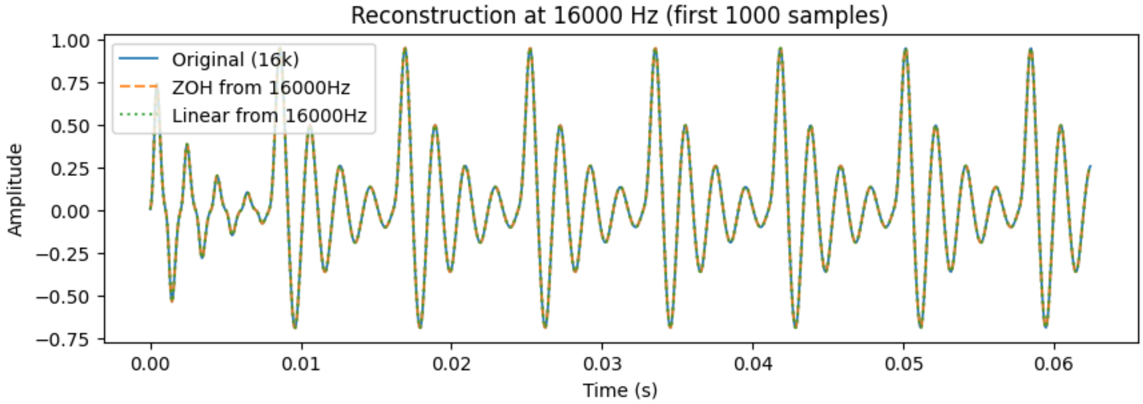


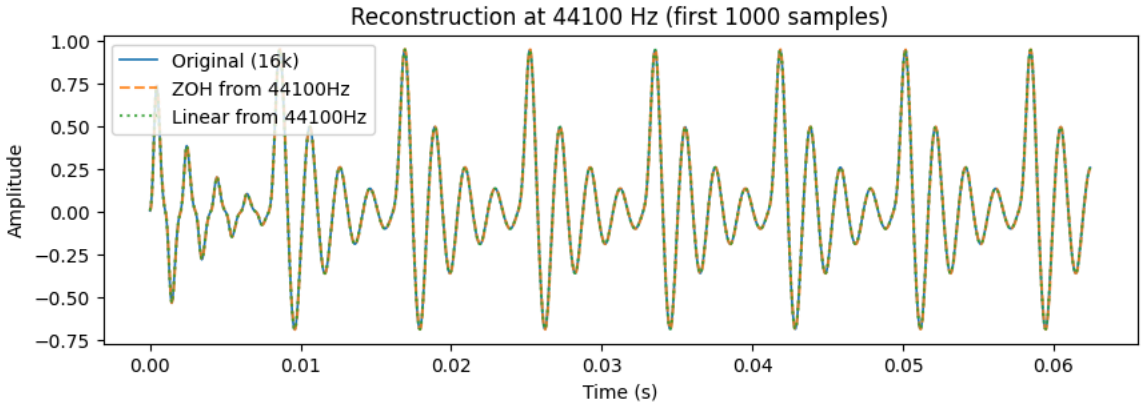





```
# ---------------------------
# 2(d) Reconstruct signals (ZOH and Linear) back to model SR
# ---------------------------
reconstructions = {}   # keys: (sr_target, method)

zoom_samples = 1000  # number of samples to plot for comparison

for sr_target, x_down in sampled_synth.items():
    # Reconstruction
    zoh = zoh_reconstruct(x_down, sr_target, Fs_model, len(mix))
    lin = linear_reconstruct(x_down, sr_target, Fs_model, len(mix))
    reconstructions[(sr_target, 'zoh')] = zoh
    reconstructions[(sr_target, 'linear')] = lin

    # Plot comparison with original (zoomed segment)
    plt.figure(figsize=(10, 3))
    plt.plot(t[:zoom_samples], mix[:zoom_samples], label='Original (16k)', linewidth=1.2)
    plt.plot(t[:zoom_samples], zoh[:zoom_samples], '--', label=f'ZOH from {sr_target}Hz', alpha=0.8)
    plt.plot(t[:zoom_samples], lin[:zoom_samples], ':', label=f'Linear from {sr_target}Hz', alpha=0.8)
    plt.title(f"Reconstruction at {sr_target} Hz (first {zoom_samples} samples)")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.legend()
    plt.show()
```








```
import numpy as np

# Function to compute MSE between two signals
def mse(x, y):
    min_len = min(len(x), len(y))
    return np.mean((x[:min_len] - y[:min_len])**2)

# reconstructions: dictionary {(sr_target, method): reconstructed_signal}
# mix: original synthetic speech
mse_results = {}

print("\nMSE results (original synthetic vs reconstructed):")
for (sr_target, method), x_rec in reconstructions.items():
    val = mse(mix, x_rec)
    mse_results[(sr_target, method)] = val
    print(f"  {sr_target} Hz - {method.upper():6s} : MSE = {val:.6e}")

# Optional: summary table grouped by sampling rate
print("\nSummary by sampling rate:")
for sr_target in sampling_rates:
    print(f" {sr_target} Hz: ZOH = {mse_results[(sr_target,'zoh')]:.6e}  |  Linear = {mse_results[(sr_target,'linea
```

```
MSE results (original synthetic vs reconstructed):
  8000 Hz - ZOH    : MSE = 1.750329e-03
  8000 Hz - LINEAR : MSE = 7.295366e-05
  16000 Hz - ZOH    : MSE = 0.000000e+00
  16000 Hz - LINEAR : MSE = 0.000000e+00
  44100 Hz - ZOH    : MSE = 1.556460e-04
  44100 Hz - LINEAR : MSE = 7.892218e-08
```

```
Summary by sampling rate:
 8000 Hz: ZOH = 1.750329e-03  |  Linear = 7.295366e-05
 16000 Hz: ZOH = 0.000000e+00  |  Linear = 0.000000e+00
 44100 Hz: ZOH = 1.556460e-04  |  Linear = 7.892218e-08
```

Start coding or generate with AI.

```
import librosa
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from sklearn.metrics import mean_squared_error

# --------------------------
# (a) Load original speech signal
# --------------------------
file_path = "speech.wav"   # <-- replace with your speech file path
original_signal, sr = librosa.load(file_path, sr=44100)  # load at 44.1kHz

# Time axis for original
t_original = np.arange(len(original_signal)) / sr

plt.figure(figsize=(12,4))
plt.plot(t_original, original_signal)
plt.title("Original Speech Signal (Time Domain)")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()

# --------------------------
# (b) Sample at different rates
# --------------------------
sampling_rates = [8000, 16000, 44100]
sampled_signals = {}
time_axes = {}

for new_sr in sampling_rates:
    y_resampled = librosa.resample(original_signal, orig_sr=sr, target_sr=new_sr)
    sampled_signals[new_sr] = y_resampled
    time_axes[new_sr] = np.arange(len(y_resampled)) / new_sr

# --------------------------
# (c) Plot sampled signals
# --------------------------
plt.figure(figsize=(12,8))
for i, new_sr in enumerate(sampling_rates, 1):
    plt.subplot(3,1,i)
    plt.plot(time_axes[new_sr], sampled_signals[new_sr])
    plt.title(f"Sampled Speech at {new_sr/1000:.1f} kHz")
    plt.xlabel("Time [s]")
    plt.ylabel("Amplitude")
    plt.grid(True)
plt.tight_layout()
plt.show()

# --------------------------
# (d) Reconstruction
# --------------------------
reconstructed = {"ZOH": {}, "Linear": {}}
mse_values = {"ZOH": {}, "Linear": {}}

for new_sr in sampling_rates:
    # Time axis of sampled signal
    t_sampled = time_axes[new_sr]
    y_sampled = sampled_signals[new_sr]

    # Target: match original time axis
    t_target = t_original

    # (i) Zero-Order Hold (Nearest Neighbor)
    f_zoh = interp1d(t_sampled, y_sampled, kind="nearest", fill_value="extrapolate")
    y_zoh = f_zoh(t_target)
    reconstructed["ZOH"][new_sr] = y_zoh

    # (ii) Linear Interpolation
    f_linear = interp1d(t_sampled, y_sampled, kind="linear", fill_value="extrapolate")
    y_linear = f_linear(t_target)
    reconstructed["Linear"][new_sr] = y_linear
```

```
        # --------------------------
        # (e) Compute MSE
        # --------------------------
        mse_values["ZOH"][new_sr] = mean_squared_error(original_signal, y_zoh)
        mse_values["Linear"][new_sr] = mean_squared_error(original_signal, y_linear)

    # Print MSE results
    print("Mean Squared Error (MSE):")
    for method in mse_values:
        for new_sr in sampling_rates:
            print(f"{method} at {new_sr/1000:.1f} kHz: {mse_values[method][new_sr]:.6f}")

    # --------------------------
    # Plot one example reconstruction (16 kHz)
    # --------------------------
    plt.figure(figsize=(12,6))
    plt.subplot(2,1,1)
    plt.plot(t_original, original_signal, label="Original", alpha=0.7)
    plt.plot(t_original, reconstructed["ZOH"][16000], label="ZOH Reconstruction", alpha=0.7)
    plt.legend()
    plt.title("Reconstruction using Zero-Order Hold (16 kHz)")

    plt.subplot(2,1,2)
    plt.plot(t_original, original_signal, label="Original", alpha=0.7)
    plt.plot(t_original, reconstructed["Linear"][16000], label="Linear Reconstruction", alpha=0.7)
    plt.legend()
    plt.title("Reconstruction using Linear Interpolation (16 kHz)")
    plt.tight_layout()
    plt.show()
```