Experiment 6: FIR Filter Design for Audio Denoising

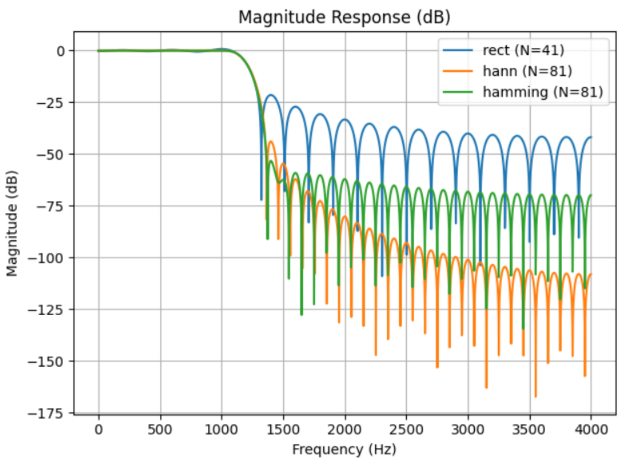
FIR Low-Pass Filter Design (Window Method)

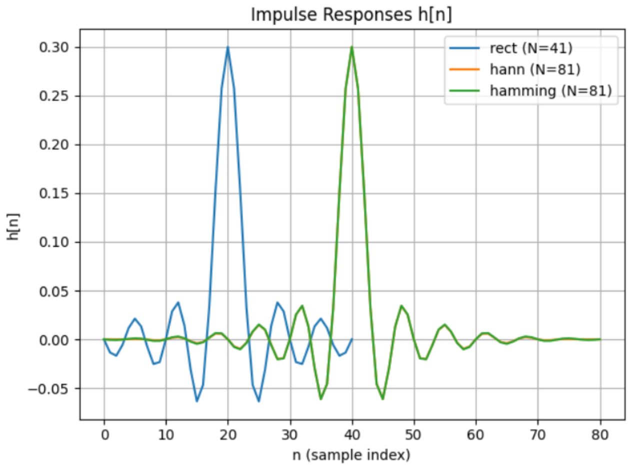
Program Code

```
import numpy as np
import matplotlib.pyplot as plt
from math import pi, ceil
from scipy.signal import freqz, windows

# Functions
def estimate_N(window_name, Dw):
    if window_name == 'rect':
        Nf = (4 * pi) / Dw
    elif window_name in ('hann', 'hamming'):
        Nf = (8 * pi) / Dw

    N = int(ceil(Nf))
    if N % 2 == 0:
        N += 1
    return N

def ideal_lp(wc, N):
    M = (N - 1) // 2
    n = np.arange(N)
    k = n - M
    hd = np.zeros(N)
    for i, kv in enumerate(k):
        if kv == 0:
            hd[i] = wc / pi
        else:
            hd[i] = np.sin(wc * kv) / (pi * kv)
    return hd

def apply_window(hd, window_name):
    N = len(hd)
    if window_name == 'rect':
        w = np.ones(N)
    elif window_name == 'hann':
        w = windows.hann(N, sym=True)
    elif window_name == 'hamming':
        w = windows.hamming(N, sym=True)

    return hd * w

def design_lpf(fs, fp, fsb, window_name):
    wp = 2 * pi * fp / fs
    wsb = 2 * pi * fsb / fs
    Dw = wsb - wp
    N = estimate_N(window_name, Dw)
    fc = (fp + fsb) / 2.0
    wc = 2 * pi * fc / fs
    hd = ideal_lp(wc, N)
```

```
    h = apply_window(hd, window_name)
    return h, N, wc, Dw


# Main script
fs = 8000
fp = 1000
fsb = 1400
windows_list = ['rect', 'hann', 'hamming']
results = []

for wname in windows_list:
    h, N, wc, Dw = design_lpf(fs, fp, fsb, wname)
    w, H = freqz(h, worN=4096, fs=fs)
    results.append((wname, N, w, H, h, wc, Dw))

# Plot magnitude responses
for (wname, N, w, H, h, wc, Dw) in results:
    mag_db = 20 * np.log10(np.maximum(np.abs(H), 1e-10))
    plt.plot(w, mag_db, label=f"{wname} (N={N})")
plt.title("Magnitude Response (dB)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude (dB)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot impulse responses
for (wname, N, w, H, h, wc, Dw) in results:
    n = np.arange(len(h))
    plt.plot(n, h, label=f"{wname} (N={N})")
plt.title("Impulse Responses h[n]")
plt.xlabel("n (sample index)")
plt.ylabel("h[n]")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output





Audio Denoising with Hamming FIR Low-Pass Filter

Program Code

```
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
from scipy.signal import lfilter, spectrogram

# Sub-algorithms
def estimate_N_hamming(Dw):

    N = np.ceil(8 * np.pi / Dw)
    if N % 2 == 0:
        N += 1
    return int(N)

def ideal_lp(wc, N):
    M = (N - 1) // 2
    hd = np.zeros(N)
    for i in range(N):
        k = i - M
        if k == 0:
            hd[i] = wc / np.pi
        else:
            hd[i] = np.sin(wc * k) / (np.pi * k)
    return hd

def apply_hamming_window(hd):
    N = len(hd)
    n = np.arange(N)
    w = 0.54 - 0.46 * np.cos(2 * np.pi * n / (N - 1))
    return hd * w

def design_hamming_lpf(fs, fp, fsb):
    wp = 2 * np.pi * fp / fs
    wsb = 2 * np.pi * fsb / fs
    Dw = wsb - wp
```

```
    N = estimate_N_hamming(Dw)
    wc = wp
    hd = ideal_lp(wc, N)
    h = apply_hamming_window(hd)
    return h, N

#Main
fs = 8000.0
fp = 1000.0
fsb = 1400.0

x, fs_file = sf.read("noisy_speech_8k.wav")

# 2. FIR Filter Design
h, N = design_hamming_lpf(fs, fp, fsb)
print(f"Hamming FIR Low-pass Filter designed with length N={N}")

# 3. Signal Filtering
y = lfilter(h, 1.0, x)

# 4(a). Spectral and Waveform Comparison     Spectrograms
f1, t1, Sx = spectrogram(x, fs=fs, nperseg=256, noverlap=128)
f2, t2, Sy = spectrogram(y, fs=fs, nperseg=256, noverlap=128)

plt.figure(figsize=(8, 5))
plt.semilogy(f1, np.mean(Sx, axis=1) + 1e-12, label="Before (noisy)")
plt.semilogy(f2, np.mean(Sy, axis=1) + 1e-12, label="After (denoised)")
plt.title("Average Power Spectral Density")
plt.xlabel("Frequency (Hz)")
plt.ylabel("PSD (linear, semilog plot)")
plt.grid(True)
plt.legend()
plt.tight_layout()

# 4(b). Waveforms comparison (short segment)
plt.figure(figsize=(10, 6))
time = np.arange(len(x)) / fs
segment = slice(0, int(0.05 * fs))

plt.subplot(2, 1, 1)
plt.plot(time[segment], x[segment])
plt.title("Original Noisy Signal (time domain)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid(True)

plt.subplot(2, 1, 2)
plt.plot(time[segment], y[segment])
plt.title("Filtered (Denoised) Signal (time domain)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid(True)

plt.tight_layout()

plt.show()

# 5. Optional Audio File Saving
```

```python
try:
    sf.write("/content/denoisy_speech_8k.wav", y, int(fs))
    print("Filtered audio saved as 'denoised_hamming.wav'")
except Exception as e:
    print("Could not save file:", e)
```

Output