PROJECT REPORT

Digital Signal Conditioning & Noise Reduction Framework

Project 1

A Comprehensive Study on Gaussian Pulse Generation, Noise Modeling, and FIR Filter Design

Contents

Abstract

This project report presents a comprehensive implementation of digital signal conditioning and noise reduction techniques applied to Gaussian pulse signals. The work encompasses the generation of Gaussian pulses with controlled parameters, the addition of realistic noise models including both Gaussian and Bernoulli-Gaussian impulse noise, spectral analysis using FFT and PSD methods, and the design and implementation of various windowed FIR filters for noise suppression. The study demonstrates the effectiveness of different window functions (Rectangular, Hamming, Blackman, and Hanning) in filtering applications and introduces matched filtering techniques for optimal signal detection. Performance evaluation is conducted through SNR comparison across different filtering approaches, providing quantitative insights into the efficacy of each method.

Chapter 1

Introduction

Signal processing plays a fundamental role in modern communication systems, radar applications, medical imaging, and various other domains where signals are corrupted by noise during acquisition or transmission. The ability to accurately reconstruct original signals from noisy observations is critical for reliable system performance.

This project focuses on the complete signal processing pipeline, from signal generation through noise addition, analysis, and filtering. A Gaussian pulse serves as the test signal due to its importance in pulse radar systems, ultra-wideband communications, and detector physics. The Gaussian pulse provides a controlled, analytically tractable signal with known spectral properties, making it ideal for evaluating filter performance.

The project addresses realistic noise scenarios by implementing both Gaussian noise (representing thermal noise and other random processes) and impulse noise (modeling switching transients, electromagnetic interference, and cosmic ray disturbances). The combination of these noise types creates challenging conditions for signal recovery, requiring sophisticated filtering strategies.

1.1 Objectives

The primary objectives of this project are:

- Generate Gaussian pulses with controlled temporal characteristics
- Implement realistic noise models including Gaussian and Bernoulli-Gaussian impulse noise
- Perform spectral analysis to characterize signal and noise components
- Design and implement windowed FIR filters for noise reduction
- Develop matched filtering techniques for optimal signal detection
- Compare filter performance through quantitative SNR analysis

Chapter 2

Gaussian Pulse Generation

The Gaussian pulse serves as the fundamental signal in this study. Its mathematical formulation ensures complete control over pulse width, amplitude, and temporal characteristics.

2.1 Mathematical Formulation

The Gaussian pulse is defined by the equation:

$$x(n) = A \cdot \exp\left[-\frac{(n-n_0)^2}{2\sigma^2}\right] \tag{2.1}$$

where:

- A = amplitude of the pulse
- σ = standard deviation controlling pulse width
- N = number of samples
- n = time index (0 to $N-1$)
- n_0 = pulse center ($N/2$)
- f_s = sampling frequency = 1000 Hz

The temporal relationships are given by:

$$\sigma_{\text{time}} = \frac{\sigma_{\text{samples}}}{f_s} \tag{2.2}$$

$$t = \frac{n-n_0}{f_s} \tag{2.3}$$

2.2 Design Parameters

The number of samples is chosen as a power of two to ensure efficient FFT computation, sufficient frequency resolution, and adequate capture of the Gaussian pulse without truncation. Values such as 256 or 512 represent a practical balance between accuracy and computational efficiency. Powers of two are not mathematically required, but they are computationally optimal and system-aligned in FFT-based digital signal processing.

The design follows these critical rules:

1. Choose σ first based on pulse physics requirements

2. Ensure $N \geq 6\sigma$ to capture the pulse without significant truncation
3. Round N up to the nearest power of 2 for FFT efficiency

2.3 Implementation

For this project, the following parameters were selected:

- $A = 1$ (normalized amplitude)
- $\sigma = 30$ samples
- $N = 512$ samples (next power of 2 after $6\sigma = 180$)
- $f_s = 1000$ Hz
- $n_0 = 256$ (pulse centered in the window)

The pulse width is approximately $6\sigma \approx 180$ samples, where most of the signal energy is concentrated.

Chapter 3

Noise Generation and Modeling

Realistic noise modeling is essential for evaluating filter performance under practical conditions. This project implements two distinct noise types that commonly occur in real-world signal acquisition systems.

3.1 Gaussian Noise

Gaussian noise represents thermal noise, receiver noise, and other random processes that follow a normal distribution. The noise is generated with power controlled by the desired input Signal-to-Noise Ratio (SNR).

3.1.1 Generation Procedure

The Gaussian noise generation procedure follows these steps:

1. Decide desired input SNR (set to 0.25 dB for this project)
2. Compute signal power:
$$P_s = \frac{1}{N}\sum_{n=0}^{N-1} |x(n)|^2 \tag{3.1}$$
3. Compute required noise power:
$$P_n = \frac{P_s}{10^{\mathrm{SNR}/10}} \tag{3.2}$$
4. Generate noise:
$$w(n) = \sqrt{P_n} \cdot \mathrm{randn}(\mathrm{size}(x_n)) \tag{3.3}$$

3.2 Impulse Noise

Impulse noise is modeled as Bernoulli-Gaussian noise, where impulse occurrences follow a Bernoulli process and impulse amplitudes are Gaussian distributed. Classical impulse noise consists of sparse, fixed-magnitude bipolar spikes, whereas the generated impulse noise uses sparse impulses with Gaussian-distributed amplitudes, resulting in a heavy-tailed noise distribution.

The difference is not merely the magnitude of the impulses but whether the impulse amplitudes are deterministic or randomly distributed. Classical impulse noise uses fixed-magnitude spikes, whereas the generated impulse noise has Gaussian-distributed amplitudes, leading to heavy-tailed behavior.

3.2.1 Physical Representation

Impulsive noise consists of:

- Rare events
- Very high amplitude
- Short duration (often 1–2 samples)

Physically, it represents:

- Switching transients in power supplies
- Electromagnetic interference (EMI) spikes
- Cosmic ray-like disturbances in electronics

Mathematically, it is not Gaussian.

3.2.2 Mathematical Model

The impulse noise is defined as:

$$w_i(n) = \begin{cases} A_i & \text{with probability } p \\ 0 & \text{with probability } (1-p) \end{cases} \tag{3.4}$$

3.2.3 Generation Parameters

- Impulse Probability: $p = 0.005$ (0.5% of samples corrupted, representing sparse events)
- Noise Standard Deviation: $\sigma_n = \sqrt{P_n}$
- Impulse Amplitude: $A_i = 15 \cdot \sigma_n \cdot \text{randn}(1, N)$ (has both polarities)
- Bernoulli Process: $b = \text{rand}(1, N) < p$
- Impulsive Noise Sequence: $w_i = b \cdot A_i$

3.2.4 Amplitude Selection Rationale

Practical rules of thumb for impulse noise amplitude selection (assuming signal/noise has standard deviation σ):

- **Mild impulse noise ($\approx 5\sigma$):** Barely distinguishable from large Gaussian excursions, often too weak for testing impulse-noise robustness
- **Typical/realistic impulse noise ($\approx 10\sigma$):** Clearly non-Gaussian, common choice in signal processing simulations
- **Strong/stress-test impulse noise ($\approx 20\sigma$ or more):** Dominates locally, used to test filters, denoisers, and outlier rejection

For this project, 15σ was selected to create challenging but realistic conditions for filter evaluation.

3.3 Combined Noise Model

The final noisy signal is formed by:

$$y(n) = x(n) + w(n) + w_i(n) \tag{3.5}$$

This represents the clean Gaussian pulse corrupted by both continuous Gaussian noise and sparse impulsive disturbances.

Chapter 4

Spectral Analysis of Noisy Signal

FFT shows spectral components, but PSD quantifies power distribution, which is essential for noise characterization and detector signal analysis. Magnitude spectra were used for filter specification and visualization, while PSD analysis can be employed for statistical noise characterization.

4.1 FFT Implementation

The Fast Fourier Transform (FFT) was computed with zero-padding to increase frequency resolution:

- FFT length: $N_{\text{fft}} = 1024$ (zero-padded from 512 samples)
- Frequency axis: $f = \left(-\frac{N_{\text{fft}}}{2} : \frac{N_{\text{fft}}}{2} - 1\right) \cdot \frac{f_s}{N_{\text{fft}}}$
- Magnitude spectrum: $|Y(k)|/N_{\text{fft}}$ (normalized)

The computation is performed as:

$$Y(k) = \text{FFT}(y(n), N_{\text{fft}}) \tag{4.1}$$

$$Y_{\text{shift}}(k) = \text{fftshift}(Y(k)) \tag{4.2}$$

$$\text{Mag}(k) = \frac{|Y_{\text{shift}}(k)|}{N_{\text{fft}}} \tag{4.3}$$

4.2 Filtering Objective

The objective of filtering is to remove high-frequency noise components introduced by additive noise while preserving the low-frequency spectral content of the Gaussian pulse. Since the useful signal energy is concentrated at low frequencies, a low-pass FIR filter is selected to achieve effective noise suppression without distorting the signal shape.

The filtering objective is defined based on spectral separation between signal-dominated and noise-dominated frequency regions. Analysis of the magnitude spectrum reveals that the Gaussian pulse energy is primarily concentrated below 100 Hz, while noise components extend across the entire frequency range.

Chapter 5

Median Filtering for Impulse Noise Suppression

Before applying linear FIR filters, median filtering is employed as a preprocessing step to remove impulse noise. Median filters are nonlinear filters that replace each sample with the median value within a sliding window, making them highly effective at removing isolated outliers while preserving signal edges.

5.1 Window Size Selection

For the Gaussian pulse with $\sigma = 30$ samples, the pulse width is approximately $6\sigma \approx 180$ samples where most energy is concentrated. The window size must be much smaller than the pulse width to avoid smoothing away the signal itself.

Median filters must use odd window sizes $(3, 5, 7, 9, \ldots)$ so there is a clear center sample to replace. The formula for window size selection is:

$$\text{Window Size} = 2 \times (\text{max impulse width}) + 1 \tag{5.1}$$

For impulses with width of approximately 1–2 samples:

$$\text{Minimum window} = 2 \times 1 + 1 = 3 \tag{5.2}$$

$$\text{Safer choice} = 2 \times 2 + 1 = 5 \tag{5.3}$$

A window size of 5 provides the optimal balance between impulse removal and signal preservation. This size requires at least 3 "good" samples in the window to outvote the spike.

5.2 Window Size Guidelines

Window Size	**Effect**	**Best For**
3	Minimal smoothing	Very sparse impulses, preserve fine details
5	Good balance	Typical impulse noise (1–2 sample width)
7	More aggressive	Dense impulses or wider spikes
9+	Heavy smoothing	Very noisy, but risks distorting signal

Table 5.1: Median filter window size selection guidelines

5.3 Implementation

A 5-point median filter was applied to the noisy signal before FIR filtering:

$$y_{\text{med}}(n) = \text{medfilt1}(y(n), 5) \tag{5.4}$$

Chapter 6

FIR Filter Design and Implementation

Finite Impulse Response (FIR) filters provide linear-phase characteristics essential for preserving signal shape. This project implements low-pass FIR filters using the window method with various window functions to evaluate their performance characteristics.

6.1 Design Specifications

- Sampling Frequency: $f_s = 1000$ Hz
- Cutoff Frequency: $f_c = 100$ Hz
- Filter Order: $N_f = 51$
- Normalized Cutoff: $f_{cn} = \frac{f_c}{f_s/2} = 0.2$

6.2 Windowed-Sinc Method

The ideal low-pass filter impulse response is given by:

$$h(n) = 2f_{cn} \cdot \mathrm{sinc}(2f_{cn} \cdot (n - \alpha)) \tag{6.1}$$

where $\alpha = 0.5(N_f - 1)$ centers the filter for linear phase. The sinc function is defined as:

$$\mathrm{sinc}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} & x \neq 0 \\ 1 & x = 0 \end{cases} \tag{6.2}$$

6.3 Coefficient Normalization

The FIR coefficients are normalized after windowing to ensure unity DC gain, compensating for amplitude distortion introduced by truncation. When you design an FIR filter using a window, the raw coefficients $h[n]$ from the windowed-sinc method may not sum to 1. This can cause:

- The filtered signal amplitude to change (gain $\neq 1$)
- SNR calculations to be inaccurate

Normalization ensures the DC gain of the filter equals 1 (passband gain is unity):

$$h_{\text{norm}}(n) = \frac{h(n)}{\sum_{k=0}^{N_f-1} h(k)} \tag{6.3}$$

Improper indexing and cutoff normalization can lead to complete gain cancellation in FIR filters; therefore, filter indices and window lengths must be strictly matched.

6.4 Window Functions

Four different window functions were implemented to compare their frequency-domain characteristics.

6.4.1 Rectangular Window

$$w_{\text{rect}}(n) = 1, \quad 0 \leq n \leq N_f - 1 \tag{6.4}$$

Provides the sharpest transition band but exhibits significant ripple in both passband and stopband due to spectral leakage. Best main lobe width but poorest sidelobe suppression.

6.4.2 Hamming Window

$$w_{\text{hamm}}(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N_f - 1}\right) \tag{6.5}$$

Offers improved sidelobe suppression compared to rectangular window with moderate transition bandwidth. Provides good balance between main lobe width and sidelobe attenuation.

6.4.3 Blackman Window

$$w_{\text{black}}(n) = 0.42 - 0.5 \cos\left(\frac{2\pi n}{N_f - 1}\right) + 0.08 \cos\left(\frac{4\pi n}{N_f - 1}\right) \tag{6.6}$$

Provides excellent sidelobe suppression at the cost of wider main lobe. Best overall frequency selectivity for applications requiring high stopband attenuation.

6.4.4 Hanning Window

$$w_{\text{hann}}(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N_f - 1}\right) \tag{6.7}$$

Provides moderate performance between Hamming and Blackman windows. Good general-purpose window with reasonable sidelobe suppression and transition bandwidth.

6.5 Filter Implementation

For each window function:

$$h_{\text{windowed}}(n) = h(n) \cdot w(n) \tag{6.8}$$

$$h_{\text{final}}(n) = \frac{h_{\text{windowed}}(n)}{\sum_{k=0}^{N_f-1} h_{\text{windowed}}(k)} \tag{6.9}$$

Chapter 7

Signal Filtering and Convolution

Time-domain linear convolution is used as it directly represents the physical operation of an LTI FIR filter. FFT-based filtering is discussed as an efficient alternative for long sequences, with appropriate zero-padding to preserve linear convolution.

7.1 Linear Convolution

Linear convolution is implemented by sliding the FIR impulse response across the input signal, multiplying overlapping samples, and summing them. The output length is the sum of individual signal lengths minus one, and indexing must reflect correct signal alignment.

For a signal of length L and filter of length N_f, the convolution is:

$$y_{\text{conv}}(n) = \sum_{k=0}^{N_f-1} h(k) \cdot y_{\text{med}}(n-k) \tag{7.1}$$

The output length is: $L_{\text{out}} = L + N_f - 1$

7.2 Group Delay Compensation

The apparent left shift in the filtered signal occurs due to FIR group delay, which must be compensated during trimming. The amplitude variation is caused by non-unity DC gain of the windowed FIR coefficients. The Blackman window happens to preserve amplitude because its coefficient sum is closest to unity. Normalizing the filter coefficients and compensating for group delay resolves both effects.

The group delay for linear-phase FIR filters is:

$$\tau = \frac{N_f - 1}{2} \tag{7.2}$$

The filtered outputs were aligned by compensating for FIR group delay through proper trimming of the convolution result.

7.3 Valid Output Extraction

To obtain the valid portion of the filtered signal aligned with the original:

$$y_{\text{valid}} = y_{\text{conv}}(N_f : N_f + L - 1) \tag{7.3}$$

This indexing removes the group delay and extracts samples corresponding to the original signal length.

For comparison with the clean signal, the same operation is performed:

$$x_{\text{conv}}(n) = \sum_{k=0}^{N_f-1} h(k) \cdot x(n-k) \tag{7.4}$$

$$x_{\text{valid}} = x_{\text{conv}}(N_f : N_f + L - 1) \tag{7.5}$$

Chapter 8

Matched Filter Design

The matched filter is designed to maximize the signal-to-noise ratio at the filter output for a known signal waveform. For the Gaussian pulse, the matched filter impulse response is the time-reversed and conjugated version of the signal itself.

8.1 Matched Filter Theory

For a known signal $s(t)$ in additive white Gaussian noise, the matched filter has impulse response:

$$h_{\text{match}}(t) = s^*(T - t) \tag{8.1}$$

For real signals, this simplifies to time reversal:

$$h_{\text{match}}(n) = x(N - 1 - n) \tag{8.2}$$

The matched filter maximizes the output SNR at the sampling instant T when the signal and noise are uncorrelated.

8.2 Implementation

The matched filter was implemented by:

1. Time-reversing the clean Gaussian pulse: $h_{\text{match}} = \text{fliplr}(x_n)$
2. Normalizing to unit energy: $h_{\text{match}} = \frac{h_{\text{match}}}{\sum h_{\text{match}}}$
3. Convolving with the median-filtered noisy signal

The filtering operation is:

$$y_{\text{match}}(n) = h_{\text{match}} * y_{\text{med}}(n) \tag{8.3}$$

$$y_{\text{match,valid}} = y_{\text{match}}(N_h : N_h + L - 1) \tag{8.4}$$

where $N_h = \text{length}(h_{\text{match}})$ accounts for group delay.

8.3 Characteristics

The matched filter provides optimal detection performance in white Gaussian noise but may not be optimal for colored noise or impulse noise scenarios. Its frequency response is matched to the signal spectrum, providing maximum energy collection at the decision instant.

The matched filter output exhibits:

- Maximum SNR at the peak (center of filtered pulse)
- Autocorrelation-like shape (pulse convolved with itself)
- Enhanced pulse width due to self-convolution

Chapter 9

Performance Evaluation and SNR Analysis

Quantitative performance evaluation was conducted by computing the Signal-to-Noise Ratio (SNR) before and after filtering for each window function and the matched filter.

9.1 SNR Computation Methodology

The SNR was computed using the following procedure:

1. Estimate the noise component:

$$n(n) = y(n) - x(n) \tag{9.1}$$

2. Compute signal power:

$$P_s = \frac{1}{L}\sum_{n=0}^{L-1} x^2(n) \tag{9.2}$$

3. Compute noise power:

$$P_n = \frac{1}{L}\sum_{n=0}^{L-1} n^2(n) \tag{9.3}$$

4. Calculate SNR in dB:

$$\text{SNR} = 10\log_{10}\left(\frac{P_s}{P_n}\right) \tag{9.4}$$

For filtered signals, the noise is estimated as:

$$n_{\text{after}}(n) = y_{\text{filtered}}(n) - x_{\text{filtered}}(n) \tag{9.5}$$

where x_{filtered} is the clean signal passed through the same filter.

9.2 Results Summary

The SNR comparison provides quantitative insight into the effectiveness of each filtering approach. Results demonstrate the trade-offs between different window functions in terms of noise suppression and signal preservation.

Expected outcomes:

- **Blackman window:** Highest noise suppression due to best stopband attenuation

- **Rectangular window:** Moderate performance with potential passband ripple effects
- **Hamming and Hanning windows:** Intermediate performance balancing selectivity and suppression
- **Matched filter:** Optimal SNR for Gaussian noise component, though affected by impulse noise

9.3 Performance Metrics

The key metrics for evaluation include:

- Input SNR before filtering
- Output SNR after each filtering method
- SNR improvement (output SNR - input SNR)
- Visual assessment of signal shape preservation
- Frequency domain characteristics

Chapter 10

Conclusion

This project successfully implemented a comprehensive digital signal conditioning and noise reduction framework. The work demonstrated the complete signal processing pipeline from controlled signal generation through realistic noise modeling, spectral analysis, and sophisticated filtering techniques.

10.1 Key Achievements

- Generated Gaussian pulses with precise control over temporal characteristics using mathematically rigorous parameter selection
- Implemented realistic Bernoulli-Gaussian impulse noise model that accurately represents physical disturbances in electronic systems
- Performed comprehensive spectral analysis to identify signal-dominated and noise-dominated frequency regions
- Designed and implemented multiple windowed FIR filters with proper coefficient normalization and group delay compensation
- Developed matched filtering approach for optimal signal detection in white Gaussian noise
- Conducted quantitative performance evaluation through SNR comparison across different filtering methods

10.2 Technical Insights

Several critical technical insights emerged from this implementation:

- Median filtering provides essential preprocessing for impulse noise removal before linear filtering
- Window function selection represents a fundamental trade-off between transition bandwidth and stopband attenuation
- Proper coefficient normalization is essential to maintain signal amplitude and enable accurate SNR calculations
- Group delay compensation ensures temporal alignment of filtered signals with original waveforms

- The difference between classical impulse noise and Bernoulli-Gaussian impulse noise lies in whether amplitudes are deterministic or randomly distributed, leading to different statistical properties

10.3 Applications

The techniques developed in this project have direct applications in:

- Pulse radar signal processing
- Ultra-wideband communication systems
- Detector physics and particle detection
- Biomedical signal processing (ECG, EEG analysis)
- Industrial instrumentation and measurement systems

10.4 Future Work

Potential extensions to this work include:

- Adaptive filtering techniques for non-stationary noise environments
- Wavelet-based denoising for multi-resolution analysis
- Machine learning approaches for automatic filter parameter optimization
- Real-time implementation on embedded DSP platforms
- Extension to multi-channel signal processing scenarios

Appendix A

MATLAB Implementation Code

The complete MATLAB implementation is provided below for reference and reproducibility.

A.1 Gaussian Pulse Generation

```
% ------------------------------
% Generate Gaussian Pulse
% ------------------------------

A = 1;
sigma = 30;
N = 512;
n = 0:N-1;
no = N/2;
fs = 1000;

x_n = A * exp(- (n - no).^2 / (2 * sigma^2));

figure;
plot(n, x_n);
xlabel('Sample Index');
ylabel('Amplitude');
title('Gaussian Pulse');
grid on;
```

A.2 Noise Generation

```
% ------------------------------
% Adding Noise
% ------------------------------

% Gaussian Noise
ISNR = 0.25;
Ps = 1/N * sum(x_n.^2);
Pn = Ps/ 10^(ISNR/10);
w_n = sqrt(Pn)*randn(size(x_n));

% Impulse Noise
p = 0.005;
```

```
b = rand(1,N) < p;
sigma_n = sqrt(Pn);
Ai = 15 * sigma_n * randn(1,N);
w_i = b .* Ai;

% Combined Noise
y_n = x_n + w_n + w_i;

figure;
plot(n, y_n);
xlabel('Sample Index');
ylabel('Amplitude with Noise');
title('Noisy Gaussian Pulse');
grid on;
```

A.3 Spectral Analysis

```
% ------------------------------
% Analysis of Noisy Signal
% ------------------------------

N_fft = 1024;
Y_k = fft(y_n,N_fft);
f = (-N_fft/2 : N_fft/2 - 1)* fs/N_fft;
Y_k_shift = fftshift(Y_k);
Mag_Y_k = abs(Y_k_shift)/N_fft;

figure;
plot(f, (Mag_Y_k));
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum of Noisy Signal');
grid on;
```

A.4 Median Filtering

```
% Median Filtering
y_n_med = medfilt1(y_n, 5);
```

A.5 FIR Filter Design

```
% Design FIR Filters
fs = 1000;
fc = 100;
Nf = 51;
```

```
n_f = 0:Nf-1;
a = 0.5*(Nf-1);
L = length(x_n);

fcn = fc / (fs/2);

hn = 2*fcn * sinc(2*fcn * (n_f - a));

% Rectangular Window
w_rect = rectwin(Nf)';
hn_rect = hn .* w_rect;
hn_rect = hn_rect/sum(hn_rect);

% Hamming Window
w_hamm = hamming(Nf)';
hn_hamm = hn .* w_hamm;
hn_hamm = hn_hamm/sum(hn_hamm);

% Blackman Window
w_black = blackman(Nf)';
hn_black = hn .* w_black;
hn_black = hn_black/sum(hn_black);

% Hanning Window
w_hann = hanning(Nf)';
hn_hann = hn .* w_hann;
hn_hann = hn_hann/sum(hn_hann);
```

A.6 Signal Filtering

```
% ------------------------------
% Filter Noisy Signal
% ------------------------------

% Rectangular Window
y_rect = conv(hn_rect, y_n_med);
x_rect = conv(hn_rect, x_n);
x_rect = x_rect(length(hn_rect):length(hn_rect)+L-1);

% Hamming Window
y_hamm = conv(hn_hamm, y_n_med);
x_hamm = conv(hn_hamm, x_n);
x_hamm = x_hamm(length(hn_hamm):length(hn_hamm)+L-1);

% Blackman Window
y_black = conv(hn_black, y_n_med);
x_black = conv(hn_black, x_n);
x_black = x_black(length(hn_black):length(hn_black)+L-1);

```

```
% Hanning Window
y_hann = conv(hn_hann, y_n_med);
x_hann = conv(hn_hann, x_n);
x_hann = x_hann(length(hn_hann):length(hn_hann)+L-1);

% ------------------------------
% Trimming Convolution Output
% ------------------------------

y_rect_valid  = y_rect(length(hn_rect)  : length(hn_rect)  + L -
    1);
y_hamm_valid  = y_hamm(length(hn_hamm)  : length(hn_hamm)  + L -
    1);
y_black_valid = y_black(length(hn_black): length(hn_black) + L -
    1);
y_hann_valid  = y_hann(length(hn_hann)  : length(hn_hann)  + L -
    1);
```

A.7 Matched Filter Design and Filtering

```
% ------------------------------
% Matched Filter Design
% ------------------------------
h_match = fliplr(x_n);
h_match = h_match / sum(h_match);

% ------------------------------
% Filtering
% ------------------------------

y_match = conv(y_n_med, h_match);
x_match = conv(x_n, h_match);

L = length(x_n);
y_match_valid = y_match(length(h_match):length(h_match)+L-1);
x_match_valid = x_match(length(h_match):length(h_match)+L-1);
```

A.8 SNR Comparison

```
% ------------------------------
% SNR Comparison
% ------------------------------

% Noise estimation
n_b = y_n - x_n;

n_a_rect  = y_rect_valid  - x_rect;
```

```
n_a_hamm  = y_hamm_valid  - x_hamm;
n_a_black = y_black_valid - x_black;
n_a_hann  = y_hann_valid  - x_hann;
n_a_match = y_match_valid - x_match_valid;

% Power calculation
Ps = mean(x_n.^2);

Pn_before = mean(n_b.^2);
Pn_rect   = mean(n_a_rect.^2);
Pn_hamm   = mean(n_a_hamm.^2);
Pn_black  = mean(n_a_black.^2);
Pn_hann   = mean(n_a_hann.^2);
Pn_match  = mean(n_a_match.^2);

% SNR calculation
SNR_before = 10*log10(Ps / Pn_before);
SNR_rect   = 10*log10(Ps / Pn_rect);
SNR_hamm   = 10*log10(Ps / Pn_hamm);
SNR_black  = 10*log10(Ps / Pn_black);
SNR_hann   = 10*log10(Ps / Pn_hann);
SNR_match  = 10*log10(Ps / Pn_match);

% Display
fprintf('\nSNR Comparison Results:\n');
fprintf('Before Filtering : %.2f dB\n', SNR_before);
fprintf('Rectangular      : %.2f dB\n', SNR_rect);
fprintf('Hamming          : %.2f dB\n', SNR_hamm);
fprintf('Blackman         : %.2f dB\n', SNR_black);
fprintf('Hanning          : %.2f dB\n', SNR_hann);
fprintf('Matched          : %.2f dB\n', SNR_match);
```