Experiment 4: Implementation and Performance Analysis of Overlap-Add and Overlap-Save for Signal Filtering

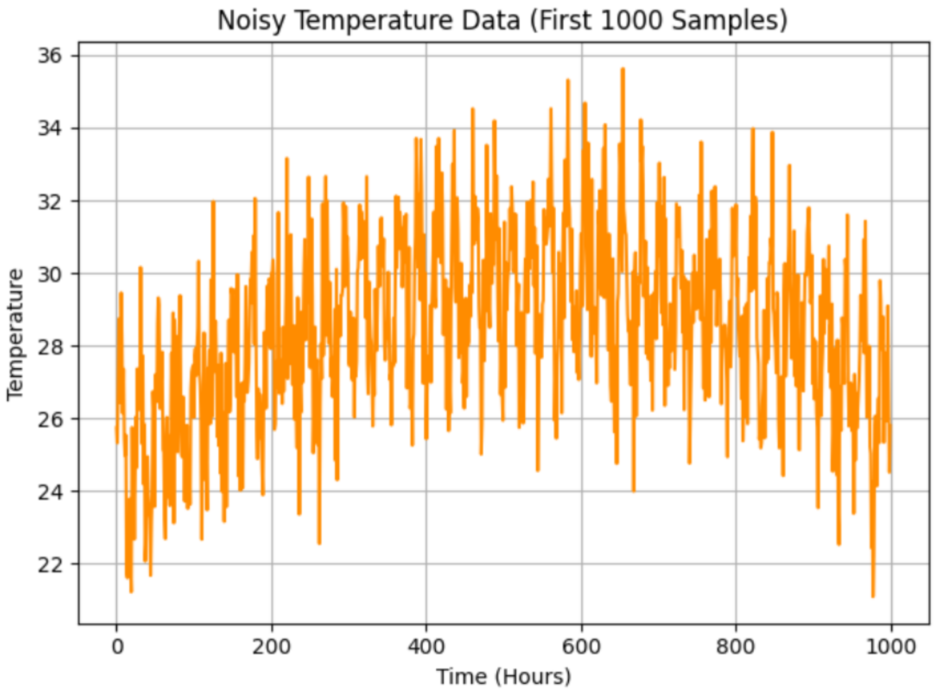
Data Loading

Program Code

```
# Step 1: Import Required Libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import firwin, freqz
import pandas as pd
import time

# Step 2: Load Noisy Temperature Data from CSV
df = pd.read_csv('/content/noisy_temperature_data.csv')

# Extract time and temperature values
Time_Hours = df['Time_Hours'].values
noisy_temperature_data = df['Noisy_Temperature'].values

# Sampling rate (1 sample per hour)
Fs = 1
total_samples = len(noisy_temperature_data)

# Step 3: Display Basic Information about the Data
print("Length of data =", total_samples)
print(f"Duration in days = {total_samples / 24:.3f}")
print("Maximum Temperature =", np.max(noisy_temperature_data))
print("Minimum Temperature =", np.min(noisy_temperature_data))

# Step 4: Plot a Segment of the Noisy Temperature Data
plt.plot(noisy_temperature_data[:1000], color='darkorange')
plt.xlabel('Time (Hours)')
plt.ylabel('Temperature')
plt.title('Noisy Temperature Data (First 1000 Samples)')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output

```
Length of data = 20000
Duration in days = 833.333
Maximum Temperature = 38.10836649441515
Minimum Temperature = 12.516180899476904
```



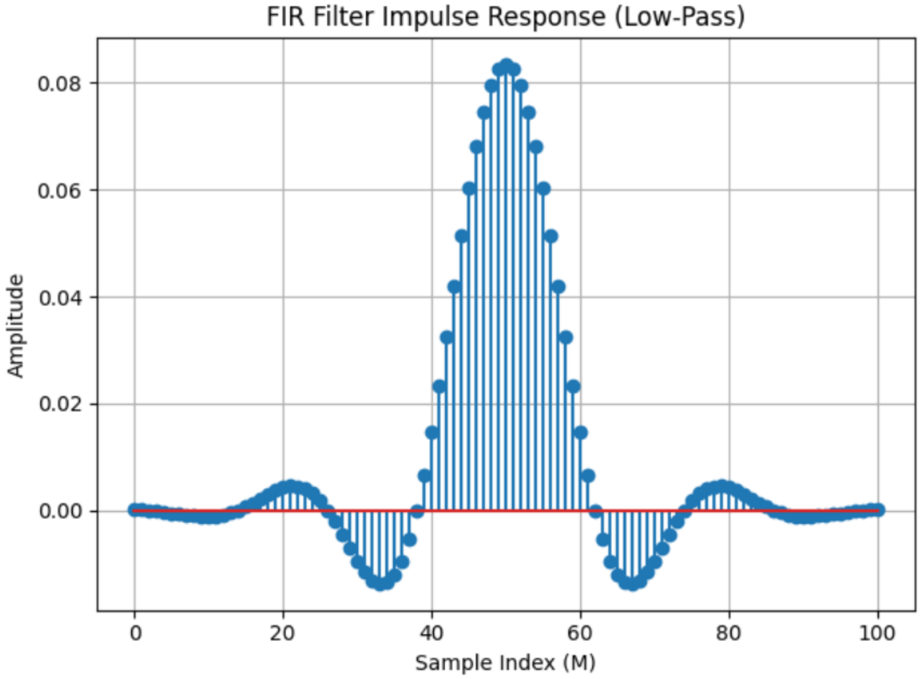
Filter Design

Program Code

```
# Design a Low-Pass FIR Filter using firwin

from scipy.signal import firwin

# Filter parameters
filter_order = 100                # Filter order (M-1)
Fs = 1                            # Sampling frequency in Hz (1 sample/
    hour)
cutoff_freq_hz = 1 / (2 * 24)    # Cutoff frequency = 1 cycle per 48
    hours
cutoff_freq_norm = cutoff_freq_hz / (Fs / 2)  # Normalized cutoff (0 to
     1)

# Design the FIR filter (low-pass)
h = firwin(filter_order + 1, cutoff_freq_norm, pass_zero=True, fs=Fs)
M = len(h)  # Actual length of the filter

print(f"Designed FIR filter of length M = {M}")

#  Plot the Impulse Response of the FIR Filter
plt.stem(h)
plt.xlabel('Sample Index (M)')
plt.ylabel('Amplitude')
plt.title("FIR Filter Impulse Response (Low-Pass)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output

```
Designed FIR filter of length M = 101
```



Frequency response of filter

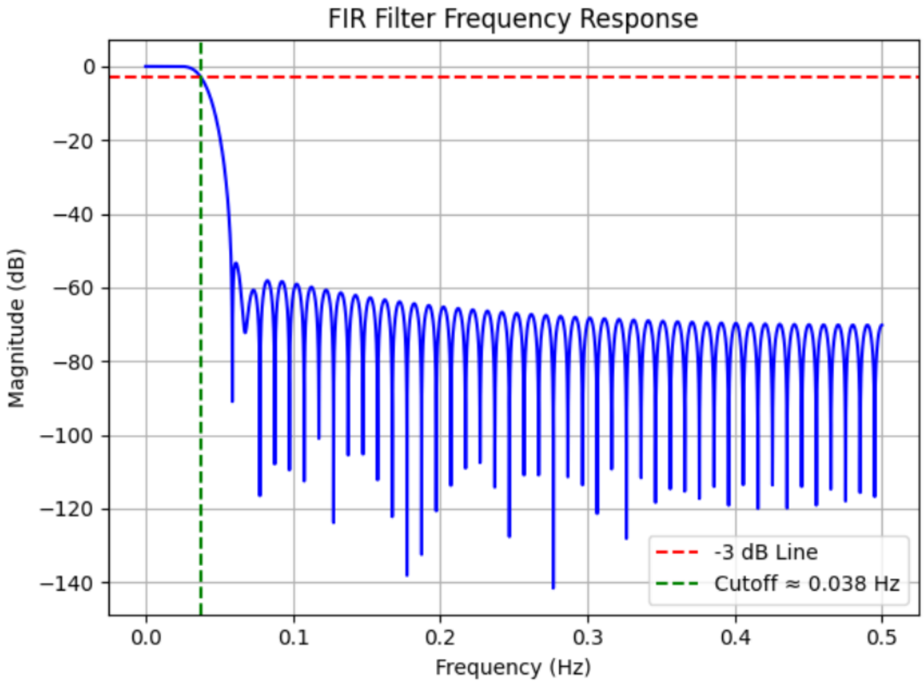
Program Code

```
#  Frequency Response of the FIR Filter
from scipy.signal import freqz

# Compute frequency response of the FIR filter
w, H = freqz(h, worN=8192, fs=Fs)  # w in Hz since fs is specified

# Convert magnitude to dB
y = 20 * np.log10(np.abs(H))

#  Determine the -3 dB Cutoff Frequency
ymax = np.max(y)
print(f"Maximum magnitude = {ymax:.3f} dB")

# Find frequency where gain drops 3 dB below max
ymax_3db = ymax - 3
print(f"Maximum magnitude 3 dB below max = {ymax_3db:.3f} dB")

# Index of the first frequency where the response drops below -3 dB
ymax_3db_index = np.where(y <= ymax_3db)[0][0]
print(f"Index at -3 dB = {ymax_3db_index}")

# Get cutoff frequency in Hz
cutoff_freq = w[ymax_3db_index]
print(f"Estimated cutoff frequency = {cutoff_freq:.3f} Hz")
```

```

# Plot the Magnitude Frequency Response (in dB)
plt.plot(w, y, color='blue')
plt.axhline(ymax_3db, color='red', linestyle='--', label='-3 dB Line')
plt.axvline(cutoff_freq, color='green', linestyle='--', label=f'Cutoff
        {cutoff_freq:.3f} Hz')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.title('FIR Filter Frequency Response')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output

```
Maximum magnitude = 0.032 dB
Maximum magnitude 3 dB below max = -2.968 dB
Index at -3 dB = 617
Estimated cutoff frequency = 0.038 Hz
```



Overlap-Add (OLA) Implementation

Program Code

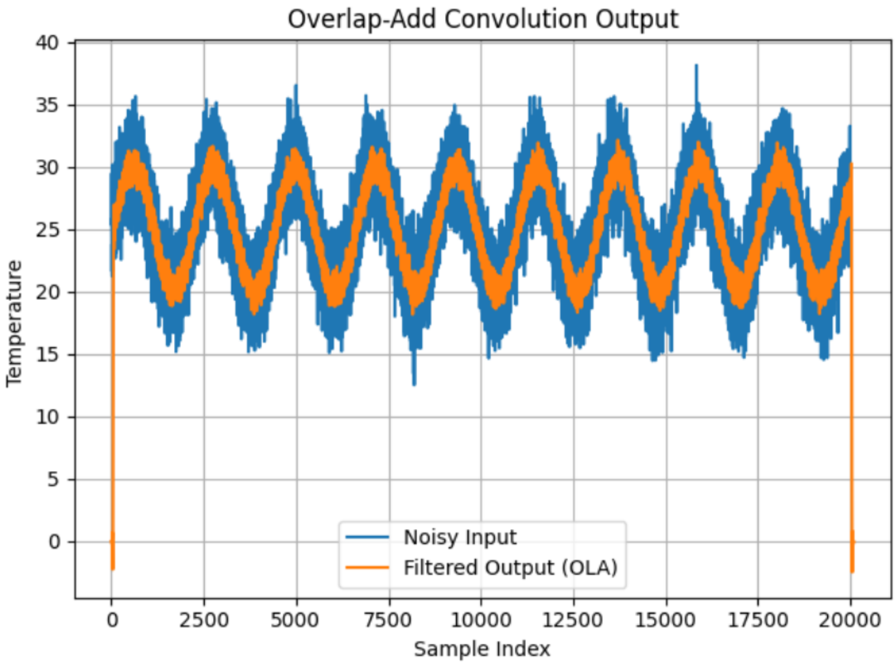
```
L = 1024  # Block length
M = len(h)  # Length of impulse response
N = L + M - 1  # Output length of linear convolution for one block
P = int(2 ** np.ceil(np.log2(N)))  # FFT length (power of 2)

# Print parameters
```

```
print(f"L = {L}, N = {N}, M = {M}, P = {P}")

# Overlap-Add function
def overlap_add(x, h, L, P):
    Nx = len(x)
    M = len(h)
    N_blocks = int(np.ceil(Nx / L))

    # Step 1: FFT of zero-padded h[n]
    H_fft = np.fft.fft(h, P)

    # Step 2: Initialize output array
    y_ola = np.zeros(Nx + M - 1)

    # Step 3: Time measurement start
    start_time = time.time()

    # Step 4: Process each block
    for i in range(N_blocks):
      start_idx = i * L
      end_idx_x = min(start_idx + L, Nx)

      # Extract and pad block
      x_block = x[start_idx:end_idx_x]
      x_block_padded = np.pad(x_block, (0, P - len(x_block)))

      # FFT, IFFT
      X_block = np.fft.fft(x_block_padded)
      Y_block = X_block * H_fft
      y_block = np.real(np.fft.ifft(Y_block))

      # Avoid overrun in final block
      end_idx_y = min(start_idx + P, len(y_ola))
      valid_length = end_idx_y - start_idx
      y_ola[start_idx:end_idx_y] += y_block[:valid_length]


    # Step 5: Time measurement end
    end_time = time.time()
    exec_time = end_time - start_time

    print(f"Overlap-Add Execution Time: {exec_time:.6f} seconds")

    return y_ola


# Apply the Overlap-Add Convolution
filtered_ola_output = overlap_add(noisy_temperature_data, h, L, P)

# Plot the result
plt.plot(noisy_temperature_data, label='Noisy Input')
plt.plot(filtered_ola_output, label='Filtered Output (OLA)')
plt.title('Overlap-Add Convolution Output')
plt.xlabel('Sample Index')
plt.ylabel('Temperature ')
plt.legend()
plt.grid(True)
plt.tight_layout()
```

```
plt.show()
```

Output

```
L = 1024, N = 1124, M = 101, P = 2048
Overlap-Add Execution Time: 0.009515 seconds
```



Overlap-Save (OLS) Implementation

Program Code

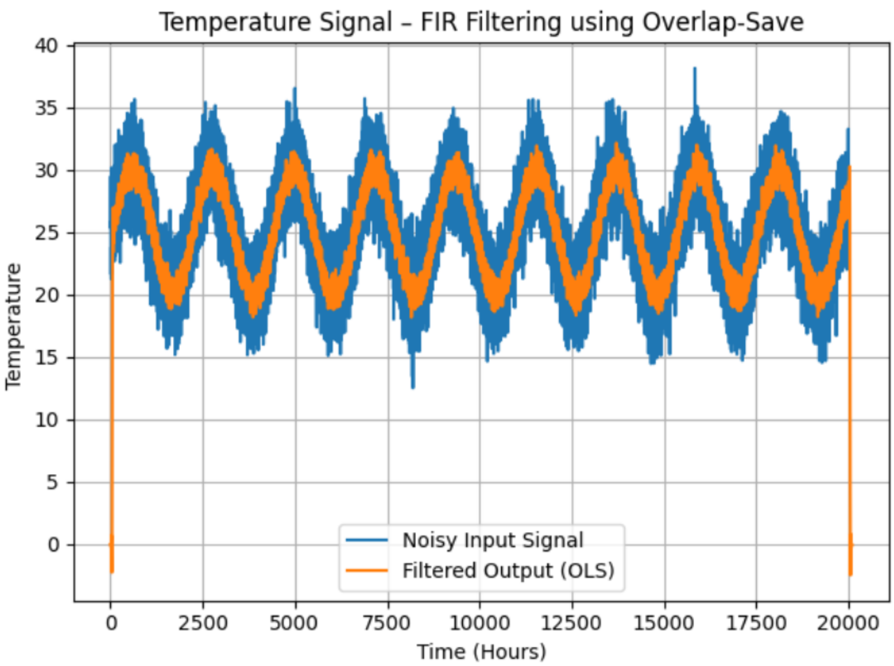
```
import numpy as np
import matplotlib.pyplot as plt
import time


# Overlap-Save FIR Filtering Function
def overlap_save(noisy_temperature_data, h, L_ols_valid, P):
    Nx = len(noisy_temperature_data)
    M = len(h)

    if P < M - 1:
        raise ValueError("FFT length P must be >= M - 1")

    # Pre-compute FFT of the FIR filter
    H_fft_ols = np.fft.fft(h, P)

    # Initialize output array
    y_ols = np.zeros(Nx + M - 1)

    # Initial overlap buffer (M-1 zeros)
```

```
    overlap_buffer = np.zeros(M - 1)

    # Start timing
    start_time = time.time()
    num_blocks = int(np.ceil(Nx / L_ols_valid))

    # Process each block
    for i in range(num_blocks):
        start_idx = i * L_ols_valid
        end_idx = min(start_idx + L_ols_valid, Nx)

        # Get current block and prepend overlap
        current_samples = noisy_temperature_data[start_idx:end_idx]
        x_block = np.concatenate((overlap_buffer, current_samples))

        # Zero-pad if block length < P
        if len(x_block) < P:
            x_block = np.pad(x_block, (0, P - len(x_block)))

        # FFT-based filtering
        X_block = np.fft.fft(x_block)
        Y_block = X_block * H_fft_ols
        y_circular = np.fft.ifft(Y_block).real

        # Extract valid part of output
        valid_output = y_circular[M - 1:M - 1 + L_ols_valid]

        # Store into output array
        output_start_idx = i * L_ols_valid
        output_end_idx = min(output_start_idx + len(valid_output), len(
            y_ols))
        y_ols[output_start_idx:output_end_idx] = valid_output[:
            output_end_idx - output_start_idx]

        # Update overlap buffer for next block
        overlap_buffer = x_block[P - (M - 1):P]

    end_time = time.time()
    print(f"OLS filtering completed in {end_time - start_time:.4f}
        seconds")

    return y_ols[:Nx + M - 1]


# Set FFT Length and Compute L_ols_valid
# Ensure P is a power of 2 and >= len(h)
P = 512
M = len(h)
L_ols_valid = P - (M - 1)


# Apply Overlap-Save Filtering
filtered_ols_output = overlap_save(noisy_temperature_data, h,
    L_ols_valid, P)


# Plot the Filtered and Noisy Temperature Data
plt.plot(noisy_temperature_data, label='Noisy Input Signal')
```

```
plt.plot(filtered_ols_output, label='Filtered Output (OLS)')
plt.xlabel("Time (Hours)")
plt.ylabel("Temperature ")
plt.title("Temperature Signal     FIR Filtering using Overlap-Save")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output

```
OLS filtering completed in 0.0048 seconds
```



Linear Convolution

Program Code

```
# Define Manual Linear Convolution Function
def linear_convolution(x, h):
    N = len(x) + len(h) - 1
    y = np.zeros(N)

    # Manual convolution implementation
    for n in range(N):
        for k in range(len(h)):
            if 0 <= n - k < len(x):
                y[n] += x[n - k] * h[k]

    return y

# Apply Linear Convolution
```

```
# Reuse the previously defined FIR filter h
h = firwin(filter_order + 1, cutoff_freq_norm , pass_zero=True , fs=Fs)

# Measure execution time
start_time = time.time()
system_out_linear_conv = linear_convolution(noisy_temperature_data , h)
end_time = time.time()

print(f"Linear Convolution Execution Time: {end_time - start_time:.4f}
    seconds")
```

Output

```
Linear Convolution Execution Time: 1.4711 seconds
```

Performance Comparison

Program Code

```
# Calculate Mean Squared Error
error_ols= np.mean(np.abs(system_out_linear_conv - filtered_ols_output)
    **2)
error_ola= np.mean(np.abs(system_out_linear_conv - filtered_ola_output)
    **2)

# Display errors
print(f"Mean Squared Error (OLS vs Linear Conv) = {error_ols}")
print(f"Mean Squared Error (OLA vs Linear Conv) = {error_ola}")
```

Output

```
Mean Squared Error (OLS vs Linear Conv) = 6.571623401358339e-29
Mean Squared Error (OLA vs Linear Conv) = 7.562060024164471e-29
```