Digital Signal Processing Lab (ECL333)

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

Department of Electronics and Communication Engineering

Faculty: Jinu Jayachandran, Viji R

Student Name: ____________________ **Roll Number:** ____________________

Instructions

1. Complete the Pre-lab work before the lab session.
2. Write all answers neatly within the space provided. Do not attach additional sheets.
3. Submit your lab sheet before leaving the lab. Late submissions receive zero marks.
4. Ensure to import Python libraries ‘numpy‘, ‘matplotlib.pyplot‘, ‘scipy.signal‘, ‘pandas‘, and ‘time‘.
5. The post-lab questions are indicated by **PTQ**.
6. Make sure the ‘noisy_temperature_data.csv‘ file (uploaded in ETLAB) is in the same directory as your lab code.

Objective

- To understand the principles and implementations of Overlap-Add and Overlap-Save methods for efficient linear convolution.
- To apply these methods to filter a long noisy data sequence (e.g., sensor data).
- To compare the computational efficiency of Overlap-Add, Overlap-Save, and linear convolution of the entire sequence.
- To appreciate the practical benefits of block convolution techniques in real-world DSP applications.

1 Theoretical Background

1.1 Linear Convolution vs. Circular Convolution

Linear Convolution

For two finite-length discrete-time sequences $x[n]$ of length N and $h[n]$ of length M, their linear convolution $y[n] = x[n] * h[n]$ is given by:

$$y[n] = \sum_{k=0}^{M-1} x[n-k]h[k]$$

The length of the resulting sequence $y[n]$ is $N + M - 1$.

Circular Convolution

For two finite-length discrete-time sequences $x[n]$ and $h[n]$ of the **same length** L, their circular convolution $y[n] = x[n] \circledast h[n]$ is given by:

$$y[n] = \sum_{k=0}^{L-1} x[k]h[(n-k)_L], \quad n = 0, 1, \dots, L-1$$

where $(n-k)_L$ denotes modulo L operation. The length of the result is L.

Relation between Linear and Circular Convolution

Linear convolution can be computed using circular convolution if the sequences are appropriately zero-padded. Specifically, if $x[n]$ has length N and $h[n]$ has length M, their linear convolution $y[n]$ has length $N + M - 1$. If both $x[n]$ and $h[n]$ are zero-padded to a length $L \geq N + M - 1$ before computing their circular convolution, the result will be identical to their linear convolution. This property is fundamental for frequency-domain convolution, as the DFT inherently performs circular convolution.

1.2 Limitations of Direct Frequency-Domain Convolution for Long Signals

For very long input signals, computing the DFT of the entire signal (which could be millions of samples long) and then performing multiplication in the frequency domain becomes computationally expensive and memory-intensive. The Fast Fourier Transform (FFT) algorithm, while efficient for individual DFTs, still scales roughly as $O(N \log N)$ where N is the length of the signal. For extremely large N, this can be prohibitive. Block convolution methods, such as Overlap-Add and Overlap-Save, circumvent this by breaking the long input signal into smaller, manageable blocks.

1.3 Block Convolution Methods

These methods break a long input signal $x[n]$ into shorter blocks, convolve each block with the filter's impulse response $h[n]$ using the efficient FFT-based circular convolution, and then combine the results to yield the overall linear convolution output.

Let $h[n]$ be the impulse response of length M. We choose an FFT length L_{FFT} such that $L_{FFT} \geq L_x + M - 1$, where L_x is the length of the input block. For computational efficiency, L_{FFT} is usually chosen as a power of 2.

1. Overlap-Add (OLA) Method

Principle: The input signal $x[n]$ is segmented into non-overlapping blocks of length L. Each block is then linearly convolved with the impulse response $h[n]$ of length M. Since the linear convolution of an L-length block with an M-length filter produces an output of length $L + M - 1$, these outputs will overlap. The overlapping portions are simply added together to form the final output.

Steps:

1. Choose block length L.
2. Set FFT length $P = L + M - 1$. Usually, P is chosen as the smallest power of 2 greater than or equal to $L + M - 1$.
3. Pad $h[n]$ with zeros to length P to get $H[k] = \text{DFT}(h[n])$.
4. Divide $x[n]$ into non-overlapping blocks $x_i[n]$ of length L.
5. For each block $x_i[n]$:
   - Pad $x_i[n]$ with zeros to length P.
   - Compute $X_i[k] = \text{DFT}(x_i[n])$.
   - Compute $Y_i[k] = X_i[k]H[k]$.
   - Compute $y_i[n] = \text{IDFT}(Y_i[k])$. This $y_i[n]$ has length P.
6. The linear convolution $y[n]$ is obtained by summing the overlapping parts of $y_i[n]$. Specifically, the last $M - 1$ samples of $y_i[n]$ overlap with the first $M - 1$ samples of $y_{i+1}[n]$.

2. Overlap-Save (OLS) Method

Principle: The input signal $x[n]$ is segmented into overlapping blocks of length P. Each block contains L new samples and $M - 1$ samples from the end of the previous block (the "overlap" part). Each block is circularly convolved with the impulse response $h[n]$ (padded to length P). The first $M - 1$ samples of the circular convolution output are typically discarded (as they are "corrupted" by the circularity due to the overlap), and the remaining L samples are saved.

Steps:

1. Choose FFT length P. This P is also the block length.
2. The number of "new" samples processed in each block is $L = P - (M - 1)$.
3. Pad $h[n]$ with zeros to length P to get $H[k] = \text{DFT}(h[n])$.
4. Divide $x[n]$ into overlapping blocks $x_i[n]$ of length P. Each block $x_i[n]$ consists of the last $M - 1$ samples of the previous block, followed by L new samples. For the first block, the initial $M - 1$ samples can be zero-padded.

5. For each block $x_i[n]$:
   - Compute $X_i[k] = \text{DFT}(x_i[n])$.
   - Compute $Y_i[k] = X_i[k]H[k]$.
   - Compute $y_i[n] = \text{IDFT}(Y_i[k])$. This $y_i[n]$ has length P.
6. The first $M - 1$ samples of each $y_i[n]$ are discarded. The remaining L samples are concatenated to form the final output $y[n]$.

1.4 Comparison of OLA and OLS

Both methods convert linear convolution into a series of FFT-based circular convolutions, offering significant computational advantages for long input signals when compared to direct time-domain convolution or a single large FFT.

- **Overlap-Add:** Requires adding overlapping segments in the time domain. Requires zero-padding each input block.
- **Overlap-Save:** Requires discarding initial samples from each output block and copying end samples to the beginning of the next input block. Requires no zero-padding of input blocks (beyond the initial zero-padding for the first block). It generally involves less memory manipulation as samples are simply discarded/saved, not summed.

The choice between OLA and OLS often depends on implementation specifics, but both are highly efficient.

Pre-lab Work

Q1. Find the output of a system with impulse response $h[n] = \{1, -1\}$ for an input $x[n] = \{4, 3, 2, 1, 0, 1, 2, 3, 4\}$ using linear convolution. How many addition and multiplication operations are required to find the output?[5 Marks]

Q2. Find the output of a system with impulse response $h[n] = \{1, -1\}$ for an input $x[n] = \{4, 3, 2, 1, 0, 1, 2, 3, 4\}$ using **overlap-save** method. Use block size $N = 6$ [5 Marks]

Q3. Find the output of a system with impulse response $h[n] = \{1, 1\}$ for an input $x[n] = \{4, 3, 2, 1, 0, 1, 2, 3, 4\}$ using **overlap-add** method. Use block size $N = 6$ [5 Marks]

Q4 Write a python function `linear_convolution(x,h)` which finds the linear convolution of input arguments `x` and `h`. [5 Marks]

2 Procedure

In this experiment, you will filter a long noisy temperature data sequence using Overlap-Add, Overlap-Save, and a direct linear convolution approach, then compare their performance.

2.1 Data Loading

1. **Load Noisy Temperature Data:** Load the ‘noisy_temperature_data.csv‘ file into a NumPy array. This simulates long-term temperature readings with noise.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import firwin, freqz
import pandas as pd
import time

# Load the noisy temperature data from CSV
df = pd.read_csv('noisy_temperature_data.csv')
noisy_temperature_data = df['Noisy_Temperature'].values
Fs = 1 # Sampling rate is 1 sample per hour, from data generation
total_samples = # The length of total number of samples of temperature
    data.

#---------------------------------------------
# Plot 1000 samples of noisy_temperature_data. What should be the units
    of x-axis and y-axis?
#---------------------------------------------
```

PTQ1. Draw the plot of the noisy temperature data segment.

PTQ2. The number of samples in the temperature data is ____________ which corresponds to ____________ days.

PTQ3. The minimum and maximum temperatures are ________ and ____________ respectively.

2.2 Filter Design

1. **Setting Filter Parameters and finding filter coefficients (taps) $h[n]$:**

   Design a low-pass FIR filter to remove high-frequency noise (e.g., daily variations and random noise) while preserving the slower, seasonal trend. A cutoff frequency around 1 cycle per 2 days (or 0.5 cycles/day) would be reasonable for smoothing.

   Note: Read the comments along with the code to understand more about the steps in filter design.

```
#-----------------------------------------------
# Filter parameters
#-----------------------------------------------
# This sets the order of the FIR filter to 100. For an FIR filter, the
    number of taps (or coefficients) is M = filter_order + 1 = 101. Each
    tap represents one coefficient (or value) of h[n] and the filter will
    have 101 coefficients.
filter_order = 100 # M-1

# Fs = 1 is the sampling frequency (1 sample per hour). You want the
    cutoff frequency to be 1 cycle every 48 hours (i.e., 0.0208 Hz).This
    is normalized by the Nyquist frequency (which is Fs/2=0.5Hz). This
    means the low-pass filter will preserve variations slower than 1 per
    48 hours, effectively removing daily noise.
cutoff_freq_norm = 1 / (2 * 24) / (Fs / 2)
#-----------------------------------------------
# Design FIR low-pass filter using firwin
#------------------------------------------------
# FUNCTION
# firwin - designs a windowed-sinc FIR filter.

# INPUTS
        # filter_order + 1 = 101 specifies the number of taps.
        # cutoff_freq_norm is the cutoff frequency (in Hz if fs=Fs is
            specified).
        # pass_zero=True means it's a low-pass filter.
        # fs=Fs makes the cutoff frequency interpretable in Hz, not
            normalized units.
# OUTPUTS
        # h - filter coefficients (or impulse repsonse of the filter)
h = firwin(filter_order + 1, cutoff_freq_norm, pass_zero=True, fs=Fs)
M = len(h) # Actual filter length

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

#-----------------------------------------------
# Plot filter's impulse response here.
```

```
#--------------------------------------------------
# Use 'stem' function as this is a discrete time filter. The x-axis
    should be values from 0 to M-1 and y-axis should be h.
```

PTQ4: Draw the impulse response.

2. **Frequency response of filter:** Here we are going to find the frequency response of the designed filter and plot its magnitude response.

```
#-----------------------------------------------------
# Frequency response of filter
#-----------------------------------------------------
# Compute the frequency response of a digital filter, specifically your
    FIR filter with impulse response h.
#--------------------------------------------------
# FUNCTION
# freqz - computes the frequency response of the filter. For an FIR
    filter, this is equivalent to evaluating the Discrete-Time Fourier
    Transform (DTFT) of h.
#INPUTS
        # h - This is the impulse response (the filter coefficients).
        # worN=8192 - This sets the number of frequency points at which
            the frequency response is computed. A higher value (like 8192)
            gives a very smooth and high-resolution plot of the magnitude
            and phase response.
        # fs=Fs - This tells freqz to return the frequency values w in Hz
            instead of radians/sample. Since Fs = 1 (1 sample/hour), w will
             range from 0 to 0.5 Hz (the Nyquist frequency for 1 Hz
            sampling).
# OUTPUTS:
        # w: A NumPy array of frequencies in Hz, from 0 to Nyquist (i.e.,
            0.5 Hz here)
        # H: The complex frequency response of the filter at each of the w
             points.
w, H = freqz(h, worN=8192, fs=Fs) # worN is number of points, fs is
    sample rate

#-------------------------------------------------
# Plot filter's magnitude response here
#-------------------------------------------------
# Plot w in x-axis and 20log10(abs(H)) in y-axis. Give appropriate labels
     for x and y-axis. Use plot function as frequency response of a
    discrete time system is continuous.
```

PTQ5: Draw the magnitude response plots of your designed filter. Mark the pass band and stop band.

PTQ6: From the magnitude response, the designed filter is a ______________ filter.

PTQ7: The pass band cut-off frequency is defined as the frequency at which the magnitude response of the filter drops 3dB below the maximum value in the pass band. The 3dB cut-off frequency of the filter is ______________________. *(You might need to write a small python code to find this.)*

2.3 Overlap-Add (OLA) Implementation

Implement the Overlap-Add method to filter the 'noisy_temperature_data'.

1. **Setup Parameters and Pre-compute FFT of Filter:**

```
#-------------------------------------------------
# Parameters for OLA/OLS
#-------------------------------------------------
L = 1024 # Input block length (adjust for performance, e.g., 2^N)
```

```
N = # FFT length. This length should be the length of output of linear
    convolution of L samples of input and h (which is of length M).
# Ensure P is a power of 2 for faster FFT
P = int(2**np.ceil(np.log2(N)))

#---------------------------------------------------
# Print the values of L, N, M and P here.
#---------------------------------------------------
# Pre-compute DFT of filter impulse response (padded to P)
H_fft = # FFT of h

# Initialize output array
filtered_ola_output = # Initialize final output array with zeros. What
    should be the size of this array?
```

2. **Segment, Convolve, and Add:** Here we are going to do Overlap-Add using FFT and IFFT instead of circular convolution. Note that computations using FFT/IFFT are efficient because of low complexity of the FFT algorithm.

```
filtered_ola_output =overlap_add(noisy_temperature_data, h, L, P)
```

A brief algorithm for the implementation of the function `overlap_add` is as follows

Input:

- $x[n]$ (`noisy_temperature_data`): The input signal of total length N_x (`total_samples`).
- $h[n]$: The impulse response of length M.
- L: The length of each non-overlapping input block.
- P: The FFT length for block convolution, chosen such that $P \geq L + M - 1$, typically rounded up to the nearest power of 2.

Output:

- $y[n]$ (`filtered_ola_output`): The linearly convolved output signal.

Steps:

1. **Initialization:**
   (a) Calculate the total number of blocks: $N_{\text{blocks}} = \lceil N_x/L \rceil$ (This is the `ceil` function).
   (b) Zero-pad $h[n]$ to length P and compute its DFT: $H_{\text{fft}} = \text{FFT}(h[n], P)$.
   (c) Initialize an array y_{ola} of size $N_x + M - 1$ with zeros.
   (d) Record the start time: `start_time_ola`.
2. **Block-wise Processing Loop:**
   (a) For each block index $i = 0$ to $N_{\text{blocks}} - 1$:

(i) **Extract Input Block:**

- `start_idx` $\leftarrow i \times L$
- `end_idx` $\leftarrow \min($`start_idx` $+ L,\ N_x)$ (why `min` here?)
- Extract $x_{\text{block}} = x[$`start_idx` : `end_idx`$]$

(ii) **Zero-Pad Block:**

- Pad x_{block} to length P to get $x_{\text{block\_padded}}$.

(iii) **Frequency Domain Processing:**

- $X_{\text{block}} \leftarrow \text{FFT}(x_{\text{block\_padded}})$
- $Y_{\text{block}} \leftarrow X_{\text{block}} \cdot H_{\text{fft}}$ (Note: This is sample by sample multiplication)
- $y_{\text{block}} \leftarrow \Re\{\text{IFFT}(Y_{\text{block}})\}$ ($\Re$ means real$(\cdot)$))

(iv) **Overlap-Add Accumulation:**

- Add y_{block} to y_{ola} starting at `start_idx`:

$$y_{\text{ola}}[\texttt{start\_idx} : \texttt{start\_idx} + P] \mathrel{+}= y_{\text{block}}[0 : P]$$

- This step adds overlapping segments from consecutive blocks automatically.

3. **Finalization:**

(a) Record `end_time_ola`.

(b) Compute total execution time: `time_ola = end_time_ola - start_time_ola`.

(c) Trim y_{ola} to length $N_x + M - 1$ (if needed).

(d) Output the execution time and the final result $y[n] = y_{\text{ola}}$.

3. **Plot Filtered Output:** Plot a segment of the `filtered_ola_output` and compare it with the original noisy data.

PTQ8: Draw the plot of original and OLA filtered data segment in the same figure.

PTQ9: The time of execution of overlap add is ____________________

2.4 Part 3: Overlap-Save (OLS) Implementation

Implement the Overlap-Save method to filter the 'noisy_temperature_data'.

1. **Setup Parameters and Pre-compute FFT of Filter:**

```
#------------------------------------------------
# Parameters for OLS
#------------------------------------------------
# P = FFT length (also block length for OLS) - use same P as OLA for fair
    comparison
# L = number of valid output samples per block = P - (M-1)
L_ols_valid = P - (M - 1)
print(f"Using L_ols_valid = {L_ols_valid} valid samples per block for OLS
   .")

# Re-compute H_fft if P changed, otherwise use previous
H_fft_ols = np.fft.fft(h, P)

# Initialize output array
filtered_ols_output = np.zeros(total_samples + M - 1)
overlap_buffer = np.zeros(M - 1) # Buffer to store the last (M-1) samples
    for next block
```

2. **Segment (with overlap), Convolve, and Save:** Let us do overlap-save (OLS) method.

```
filtered_ols_output=overlap_save(noisy_temperature_data, h, L_ols_valid,
   P)
```

A brief algorithm for the implementation of the function `overlap_save` is as follows

Input:

- $x[n]$ (`noisy_temperature_data`): Input signal of total length N_x.
- $h[n]$: Impulse response of length M.
- $L_{\text{ols\_valid}}$: Number of valid output samples per block.
- P: FFT length, chosen such that $P \geq L_{\text{ols\_valid}} + M - 1$ (often next power of 2).

Output:

- $y[n]$ (`filtered_ols_output`): The final linearly convolved output.

Steps:

1. **Initialization:**
   (a) Ensure $P > M - 1$ to permit at least one valid output sample.
   (b) Zero-pad $h[n]$ to length P and compute its DFT: $H_{\text{fft\_ols}} = \text{FFT}(h[n], P)$.
   (c) Create output array y_{ols} of size $N_x + M - 1$, initialized to zeros.
   (d) Initialize an `overlap_buffer` of length $M - 1$ with zeros.
   (e) Record the start time: `start_time_ols`.
2. **Block-wise Processing Loop:**
   (a) Compute total number of blocks:
   $$N_{\text{blocks\_ols}} = \left\lceil \frac{N_x}{L_{\text{ols\_valid}}} \right\rceil$$
   (b) For each block index $i = 0$ to $N_{\text{blocks\_ols}} - 1$:
      (i) **Extract new samples:**
      - `start_idx_new` $\leftarrow i \times L_{\text{ols\_valid}}$
      - `end_idx_new` $\leftarrow \min($`start_idx_new` $+ L_{\text{ols\_valid}},\ N_x)$
      - Extract `current_new_samples` $= x[n]$ from `start_idx_new` to `end_idx_new`

      (ii) **Form full block:**
      - Concatenate `overlap_buffer` with `current_new_samples` to get $x_{\text{block\_ols}}$
      - If $\text{len}(x_{\text{block\_ols}}) < P$, zero-pad to length P

      (iii) **Frequency-domain filtering:**
      - $X_{\text{block}} = \text{FFT}(x_{\text{block\_ols}})$
      - $Y_{\text{block}} = X_{\text{block}} \cdot H_{\text{fft\_ols}}$
      - $y_{\text{circular}} = \Re\{\text{IFFT}(Y_{\text{block}})\}$

      (iv) **Discard and retain valid output:**
      - Discard the first $M - 1$ samples of y_{circular}
      - Retain the next $L_{\text{ols\_valid}}$ samples as `valid_output_samples`

      (v) **Write to final output:**
      - `output_start_idx` $\leftarrow i \times L_{\text{ols\_valid}}$
      - Insert `valid_output_samples` into y_{ols} at positions:

        $y_{\text{ols}}[$`output_start_idx` : `output_start_idx`$+\text{len}(\text{valid\_output\_samples})]$

      (vi) **Update overlap buffer:**

- `overlap_buffer` $\leftarrow x_{\text{block\_ols}}[P-(M-1):P]$

3. **Finalization:**
   (a) Record `end_time_ols`.
   (b) Compute time: `time_ols = end_time_ols - start_time_ols`.
   (c) Trim y_{ols} to length $N_x + M - 1$ (if needed).
   (d) Output $y[n] = y_{\text{ols}}$ and report `time_ols`.

3. **Plot Filtered Output:** Plot a segment of the `filtered_ols_output` and compare it with the original noisy data.

PTQ10: Draw the plot of original and OLS filtered data segment in the same figure.

PTQ11: The time of execution of overlap save is ______________________

2.5 Linear Convolution

Use the `linear_convolution(x,h)` function that you defined in the pre-lab to find the output for system with impulse response `h` and input `noisy_temperature_data`. Also, measure the execution time.

```
# Start Time
system_out_linear_conv = linear_convolution(noisy_temperature_data, h)
# End Time
```

PTQ12: The execution time using linear convolution is ______________________

To verify that the output using linear convolution, overlap-add and overlap-save methods are same we find the mean squared error between these values.

```
error_ols=# Find the mean of square of absolute values of difference of
    system_out_linear_conv and filtered_ols_output
error_ola=# Find the mean of square of absolute values of difference of
    system_out_linear_conv and filtered_ola_output
```

PTQ13: The mean squared error obtained between linear convolution output and OLS output is ________

PTQ14: The mean squared error obtained between linear convolution output and OLA output is ________

2.6 Performance Comparison

PTQ15: **Tabulate Execution Times:** Collect the measured execution times for all three methods.

Method	**Execution Time**
Linear Convolution	
Overlap Save	
Overlap Add	

PTQ16: Compare the execution times. Which method is most efficient for this long signal? Explain why block convolution methods are generally preferred for very long sequences compared to linear convolution. Discuss potential factors influencing the exact timing results (e.g., system load, Python overhead).

Work Done During Lab

(List what you have done in the lab.)

Analysis and Inference

(Discuss the effectiveness of filtering based on the visual results. Analyze the computational performance across the three methods. Explain the trade-offs involved in choosing block size L and its impact on efficiency. Discuss real-world applications where these efficient convolution methods are crucial.)

Code Attachment

(Attach your final Python code here.)

Student Signature: ______________________ **Date:** ______________