מטלה מסכמת- חלק 1

מגישים:

סמי נחמד

נועם רהט

צחי טחן

הנחיות

- התרגיל יבוצע ביחידים/זוגות/שלשות.**על כל אחד מהסטודנטים להגיש את התרגיל במודל!**
- גם אם לא מצוין, יש לצרף לכל סעיף את קוד המטלב שכתבתם בתוספת הערות (בגוף הקוד ומעבר).
- בתרגיל זה נשתמש במספר תעודות הזהות של הסטודנטים המבצעים להגדרות שונות.

נסמן:

d_1=סכום הספרות של מספר תעודת זהות של סטודנט א'

d_2=סכום הספרות של מספר תעודת זהות של סטודנט ב'

d_3=סכום הספרות של מספר תעודת זהות של סטודנט ג'

ונגדיר בנוסף $d = d_1 + d_2 + d_3$

בתרגיל זה אסור להשתמש בפקודות המטלב: conv, filter, fft2, ifft2, fftn, ifftn, xcorr, conv2, xcorr2, fft2, fftn, fftfilt.

בחלק א' אין להשתמש גם בפקודות fft, ifft.

חלק א' - מימוש FFT

בחלק זה תכתבו שיגרה המממשתFFT ושיגרה נוספת המממשת IFFT על ידי שימוש בשגרה הראשונה.

יש לממש את השיגרות באחד מ-2 אופנים: א. ללא שימוש ברקורסיה. ב. באופן רקורסיבי.

שימו לב – אופן המימוש תלוי בזוג המבצע את התרגיל.

אם d זוגי המימוש צריך להתבצעללא שימוש ברקורסיה.

אם d אי זוגי המימוש יתבצע באופן רקורסיבי.

בכדי לבדוק את עבודתכם, השוו לפונקציות FFT ו IFFT של מטלב. צרפו את קוד המטלב שפיתחתם בתוספת הערות (בגוף הקוד).

קוד מטלב:

```
function r = bitrev(k, bits)
    % Compute the bit-reversed value of an integer
    % Input:
    %   k - integer to be bit-reversed
    %   bits - number of bits in the binary representation
```

```
    % Output:
    %   r - bit-reversed value of k

    r = 0;
    for i = 0:bits-1
        r = bitor(bitshift(r, 1), bitand(bitshift(k, -i), 1));
    end

    return;  % Explicit return statement
end

function x = bitrevorder(x)
    % Reorder the input array according to bit-reversed order
    % Input:
    %   x - input signal (vector of complex numbers)
    % Output:
    %   x - bit-reversed ordered signal

    N = length(x);
    bits = log2(N);
    for i = 0:N-1
        rev_i = bitrev(i, bits);
        if rev_i > i
            % Swap elements to achieve bit-reversed order
            temp = x(i+1);
            x(i+1) = x(rev_i+1);
            x(rev_i+1) = temp;
        end
    end

    return;  % Explicit return statement
end

function X = iterativeFFT(x)
    % Iterative FFT implementation using the Cooley-Tukey algorithm
    % Input:
    %   x - input signal (vector of complex numbers)
    % Output:
    %   X - FFT of the input signal

    N = length(x);

    % Check if N is a power of 2
    if bitand(N, N - 1) ~= 0
        error('Length of input signal must be a power of 2.');
    end

    % Bit-reversed order permutation
    X = bitrevorder(x);

    % Perform the FFT using the iterative approach
    for len = 2:2:N
        halfLen = len / 2;
        W = exp(-2i * pi * (0:halfLen-1) / len);  % Twiddle factors for
current stage
        for start = 1:len:N
            for k = 0:halfLen-1
                index1 = start + k - 1;  % Adjusted for MATLAB's 1-based
indexing
```

```
                index2 = start + k + halfLen - 1;  % Adjusted for MATLAB's
1-based indexing

                % Combine the results of smaller transforms
                t = W(k + 1) * X(index2 + 1);  % Adjusted for MATLAB's 1-
based indexing
                X(index2 + 1) = X(index1 + 1) - t;  % Adjusted for MATLAB's
1-based indexing
                X(index1 + 1) = X(index1 + 1) + t;  % Adjusted for MATLAB's
1-based indexing
            end
        end
    end
end

function X = inverseFFT(X)
    % Inverse FFT implementation using the iterative FFT function
    % Input:
    %   X - input signal (vector of complex numbers)
    % Output:
    %   x - inverse FFT of the input signal

    N = length(X);
    X = conj(X);  % Conjugate the input signal
    X = iterativeFFT(X);  % Apply the FFT
    X = conj(X);  % Conjugate the result
    X = X / N;  % Normalize by dividing by the length

    return;  % Explicit return statement
end

% Main function to run the test
function iterativeFFT_test
    % Test the function
    x = [1, 2, 3, 4];  % Sample input signal
    X_iterative = iterativeFFT(x);  % Call the iterative FFT function

    % Compare with MATLAB's built-in FFT function
    X_builtin = fft(x);

    % Display the results
    disp('Iterative FFT:');
    disp(X_iterative);

    disp('Built-in FFT:');
    disp(X_builtin);

    % Test the inverse FFT function
    x_inverse = inverseFFT(X_iterative);

    % Compare with MATLAB's built-in IFFT function
    x_builtin_inverse = ifft(X_builtin);

    % Display the results
    disp('Inverse FFT (custom):');
    disp(x_inverse);

    disp('Inverse FFT (builtin):');
    disp(x_builtin_inverse);
```

```
end
```

>> iterativeFFT_test

Iterative FFT:

10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 + 0.0000i -2.0000 - 2.0000i

Built-in FFT:

10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 + 0.0000i -2.0000 - 2.0000i

Inverse FFT (custom):

1.0000 + 0.0000i   2.0000 + 0.0000i   3.0000 + 0.0000i   4.0000 - 0.0000i

Inverse FFT (builtin):

1   2   3   4

כמובן שתוצאה יצאה נכונה כמצופה

חלק ב' – סינון דיגיטלי

נתון האות

$$r(t)=\underbrace{cos(2\pi 5t)}_{s(t)}+\underbrace{cos(2\pi 10t)}_{v(t)}=s(t)+v(t)$$

כמו כן, נתון בקובץ filter_0.25_101.mat מסנן ספרתי, המסנן הינו בעל אורך סופי של 102 דגימות.

ניתן לקרב את תגובת התדר של המסנן באופן הבא

$$H(e^{j\omega})=\begin{cases}1 & |\omega|\le\frac{\pi}{4}\\ A & \frac{\pi}{4}\le|\omega|\le\pi\end{cases}$$

כאשר A הינו קבוע כלשהו המקיים $A\ll 1$.

ברצוננו לדגום את $r(t)$ בקצב F_s, כאשר לסיגנל הדגום נקרא $r[n]$, ולאחר מכן נסנן את $r[n]$ באמצעות $H(e^{j\omega})$.

א. קבעו את תדר הדגימה כך שבמוצא המסנן נקבל את $s[n]$ וננחית את $v[n]$ פי A.

נרצה לדגום את האות הרציף r(t) בקצב Fs: כפי שלמדנו עבור אות שנראה כמו $\cos(2\pi f_0 t)$ אם נדגום בקצב Fs האות הדגום- הבדיד, ייראה כמו: $x[n]=\cos\left(2\pi\frac{f_0}{Fs}n\right)$ (מתיחת הציר פי $\frac{1}{Fs}$ והכפלה כל 2π ואצלנו:

$$r[n]=\underbrace{\cos\left(2\pi\frac{5}{Fs}n\right)}_{=s[n]}+\underbrace{\cos\left(2\pi\frac{10}{Fs}n\right)}_{=v[n]}$$

כעת, אם נרצה שהאות שיעבור *s[n]* דרך הסנן יעבור בטווח של $|w|\le\frac{\pi}{4}$ ואילו שהאות *v[n]* יונחת פי *A* יעבור בטווח של $\frac{\pi}{4}\le|w|\le\pi$

נזכור תחילה ש $\cos(w_0 t)$ מקיים בתדר $\delta(w\pm w_0)$ ונקבל:

$$r[e^{jw}]=\delta(w\pm 2\pi 5)+\delta(w\pm 2\pi 10)$$

ומכיוון שדגמנו את האות-התדר בבדיד הוא הכפלת הציר פי $Ts(=\frac{1}{Fs})$ ושכפול כל 2π -סכ"ה נקבל ש:

$$r[e^{jw}] = \delta\left(w \pm 2\pi\frac{5}{Fs}\right) + \delta(w \pm 2\pi\frac{10}{Fs})$$

ועל מנת לקבל את הטווחים שרצינו נדרוש: $\frac{5}{Fs}\pi < \frac{\pi}{4}, \frac{\pi}{4} < \frac{10\pi}{Fs} < \pi \rightarrow 40 < Fs < 80$

(נשים לב שזה גם יעמוד בתנאי ניקוייסט כי $2 * \frac{10}{F} < 2\pi \rightarrow \frac{20}{40} < 2\pi$ כלומר גם במקרה הכי גרוע של בחירת *F* לא נחרוג מהתנאי...)

ב. כמה דגימות יש ליטול מ-$r(t)$ כדי לקבל במוצא המסנן 2048 דגימות מסוננות?

כפי שלמדנו, בקונבלוציה לינארית בין אות באורך *N* לאות באורך *M* סך האיברים במוצא יהיו *L=N+M-1* ואצלנו נבצע קונבלוציה בין אות *r* באורך *N* לבין המסנן שנתון שאורכו הוא *M=102* ולכן אם נרצה שאורך האות במוצא יהיה 2048: $N + 102 - 1 = 2048 \rightarrow N = 1947$

ג. נסמן ב- $\hat{S}[n]$ את מוצא המסנן וב-$\hat{S}[k]$ את התמרת ה-DFT של 2048 הדגימות של $\hat{S}[n]$. שרטטו בעזרת מטלב את $|\hat{S}[k]|$ כאשר ציר האיקס הינו ציר תדר אנלוגי בתחום $[0, F_s]$.

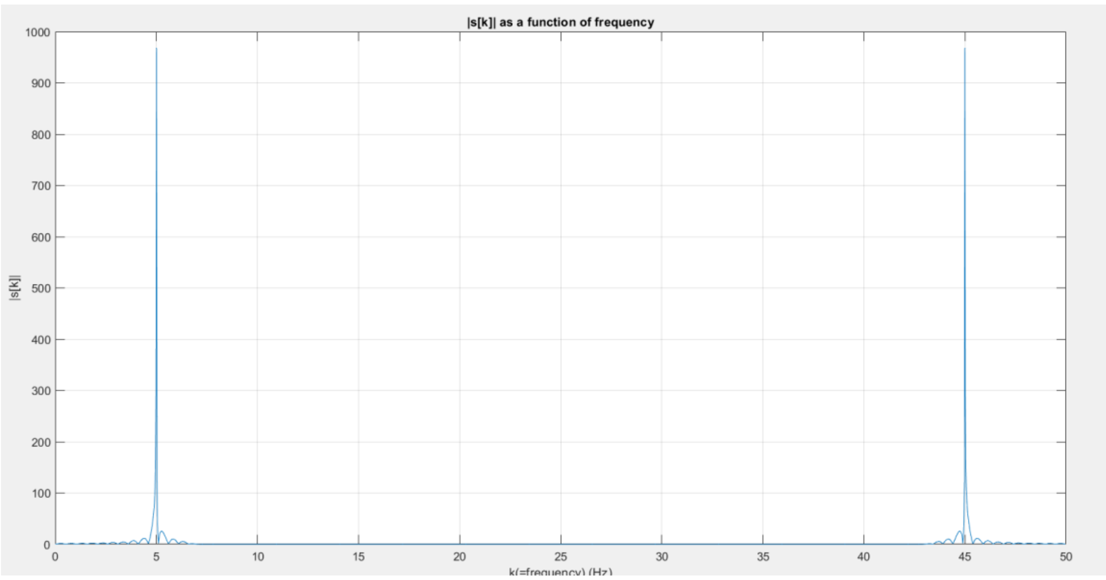
נרצה לשרטט במטלב את התמרת ה*DTF* של מוצא המסנן שלנו-לא מוגדר עבורנו מה הערך של *Fs* שאיתו נעבוד ולכן נחליט לעבוד עם תדר דגימה של $50[Hz]$;

כפי שהסברנו לעייל $s[n] = r[n] * h[n]$ וכפי שהראנו בסעיף קודם-באורך של 2048 איברים, לאחר מכן נמצא את *s[k]* דרך ה*DFT* של *s[n]* ולביטוי הזה את הערך המוחלט:

נבחר *Fs=50* -נצפה שהרכיב *v* ייסנן והרכיב *s* יישאר-נצפה לקבל:

$$s[k] = \delta\left(w - 2\pi\frac{5}{Fs}\right) + \delta\left(w - 2\pi\frac{N-5}{Fs}\right)$$

(מכיוון שזו התמרת *DFT* נקבל את הדלתאות שלנו ב*5,N-5* ולא ב± 5)כלומר נצפה לקבל שתי דלתאות סביב $2\pi\frac{5}{Fs}, 2\pi\frac{45}{Fs}$, אבל מכיוון שאנחנו עובדים בציר תדר אנלוגי-הפתרון יהיה סביב 5,45 כי כפי שלמדנו מתקיים $w = \frac{2\pi f}{F_s}$ כאשר *w* הוא התדר בבדיד וה*f* הוא התדר בציר אנלוגי(הרצים) ואנחנו עובדים בהרצים ולכן $f = w * \frac{F_s}{2\pi}$ ונקבל את 5,45



ואכן קיבלנו 2 דלתאות סביב5,45

קוד מטלב:

```
% Function to perform linear convolution
function [y, num_mult, num_add] = linear_conv(x, h)
    % Linear convolution implementation
    Nx = length(x);
    Nh = length(h);
    Ny = Nx + Nh - 1;
    y = zeros(1, Ny);

    num_mult = 0;
    num_add = 0;

    % Perform convolution
    for i = 1:Nx
        for j = 1:Nh
            y(i + j - 1) = y(i + j - 1) + x(i) * h(j);
            num_mult = num_mult + 1;  % Count multiplications
            if j > 1
                num_add = num_add + 1;  % Count additions (excluding first 
in each row)
            end
        end
        %fprintf('Number of multiplications : %d\n', num_mult);

    end
end

close all;
clc;
clear;

% Load the filter
data = load('filter_0.25_101.mat'); % This loads a struct with field names 
matching the variables in the .mat file

% Correctly access the filter coefficients using the actual field name 'h'
h = data.h; % Use struct field access to get the correct filter variable
Fs=50;
t=(0:1946)/Fs;% in order to creat a 1947 lenght vector

% c/d transform-will mention that we allready included the /Fs in the
% t vector
r_n=cos(2*pi*5*t)+cos(2*pi*10*t);

bpf_length=length(h);

s_n=linear_conv(r_n,h);% linear convelution-will result a 2048 output 
lenght signal

S_k=abs(fft(s_n));%s[k] is the dft of s[n]-finding its abs:

f=(0:2047)*(Fs/2048);% Frequency,after resolutiong like Fs/N
plot(f,S_k);
xlabel('k(=frequency) (Hz)');
ylabel('|s[k]|');
title('|s[k]| as a function of frequency');
```

```
grid on;
```

ד. לסעיף זה בלבד, נתון $F_s = 25\text{ Hz}$. האם ניתן להשתמש במסנן הנ"ל לצורך סינון $v[n]$? אם כן, הסבירו כיצד.

נתון לנו כי התדר הוא *Fs=25* ולכן:

$$r[n] = \underbrace{\cos\left(2\pi \frac{5}{25} n\right)}_{=s[n]} + \underbrace{\cos\left(2\pi \frac{10}{25} n\right)}_{=v[n]}$$

$Fs < 40$ ולכן שתי האותות יעברו במסנן דרך המכפלה ב1:ע"מ לסנן את *v* שוב-נרצה ש:

$$40 < Fs < 80$$

נעשה העלאת קצב:

נבחר *Fs=50* כלומר נרצה שהאות-שנדגם בתדר *Fs* ייראה כאילו נדגם בתדר $L * Fs$ כאשר *L=2*

בעצם נרצה להגיע לאות שייראה כמו:

$$r'[n] = r\left[\frac{n}{2}\right]$$

זה אומר שנרפד באפס את האות בזמן כך:

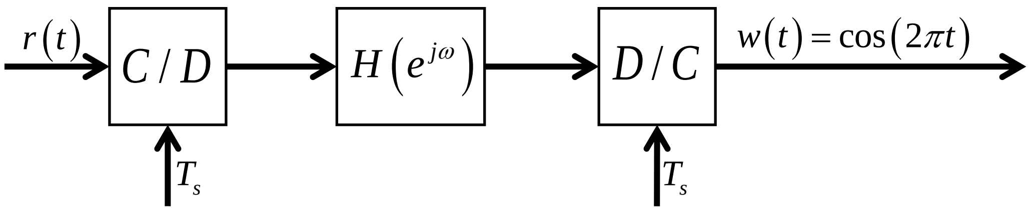
$$x_L[n] = \begin{cases} x\left[\frac{n}{L}\right] & n = 0, \pm L, \pm 2L, \dots \\ 0 & \text{else} \end{cases} \longleftrightarrow X_L\left(e^{j\omega}\right) = X\left(e^{j\omega L}\right)$$

כתוצאה מכך האות יכווץ פי *L* בתדר ונקבל שכפולים של האות הבדיד הנ"ל-במקום כל 2π כל $\frac{2\pi}{2} =$ π ולאחר מכן פשוט נסנן שכפולים מיותרים על ידי:

$$H\left(e^{j\omega}\right) = \begin{cases} L & |\omega| < \frac{\pi}{L} \\ 0 & \frac{\pi}{L} < |\omega| < \pi \end{cases}$$

לאחר מכן נבצע דמנציה כלומר הורדת קצב כפול 2 על מנת לקבל את האות המקורי בחזרה ונקבל את הדרוש...

ה. נתונה המערכת הבאה



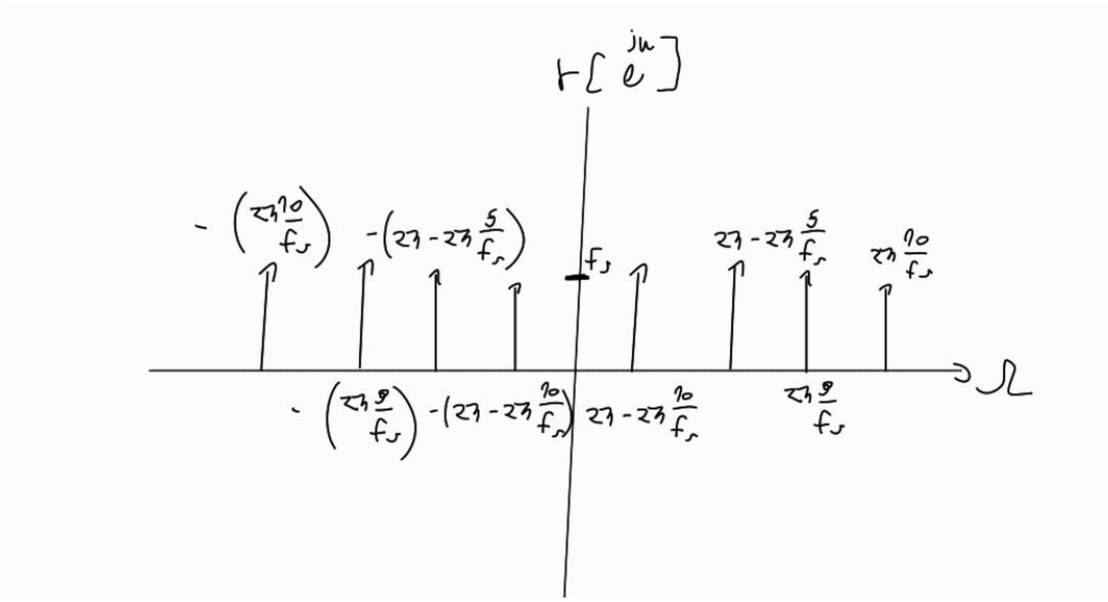
הסבירו בפירוט כיצד ניתן לממש מערכת זו בהינתן המסנן ואות הכניסה הנ"ל? (מה צריך להיות קצב הדגימה ומדוע?)

נזכור שבנקודה Y נקבל שהאות הבדיד יהיה: $r[n] = \cos\left[2\pi\frac{5}{F_s}n\right] + \cos\left[2\pi\frac{10}{F_s}n\right]$

כעת,במישור התדר-$r\left[e^{jw}\right] = \delta(\pm 2\pi 5) + \delta(\pm 2\pi 10)$

כאשר לאחר כיווץ הציר והכפלת האיברים פי Fs וביצוע שכפולים כל 2π נקבל:

$$F_s\left(\delta\left(\pm 2\pi\frac{5}{F_s}\right) + \delta\left(\pm 2\pi\frac{10}{F_s}\right) + \delta\left(\pm\left(2\pi - 2\pi\frac{5}{F_s}\right)\right) + \delta\left(\pm\left(2\pi - 2\pi\frac{10}{F_s}\right)\right)\right)$$



את זה נכפיל במסנן $H\left(e^{jw}\right)$ שלנו ונקבל את X

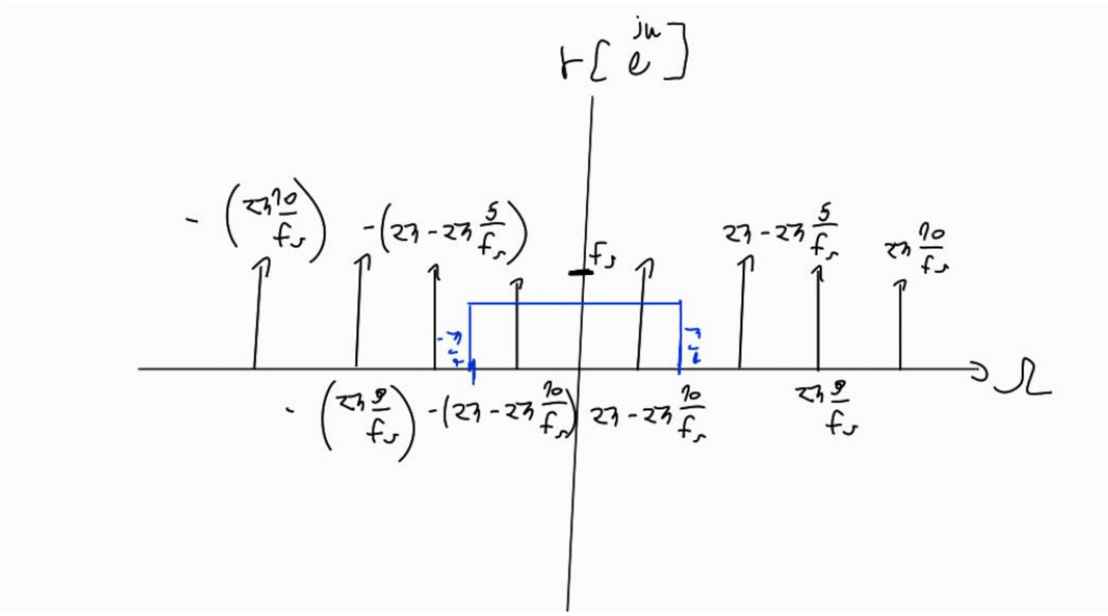
כעת,נזכור שבמוצא נצטרך לקבל $\cos(2\pi t)$ שבמישור התדר-בשלב הX נצטרך לקבל:

$$\cos(2\pi t) \rightarrow_{d\backslash c} F_s \cos\left(2\pi\frac{1}{F_s}n\right) \rightarrow_{e}^{jw} F_s\delta\left(\pm\frac{2\pi}{F_s}\right)$$

סכ"ה במישור התדר נדרוש $X = F_s\delta\left(\pm\frac{2\pi}{F_s}\right)$

נזכור כעת שנוכל להגיע לכך אם נסנן את כל האיברים חוץ מ2 האיברים הכי קטנים-

$$\delta\left(\pm\left(2\pi - 2\pi\frac{10}{F_s}\right)\right)$$



על מנת להגיע לכך נדרוש שיתקיים: $\frac{2\pi}{F_s} = 2\pi - 2\pi\frac{10}{F_s}$ על מנת שהדלתאות האילו יהיו שוות לדלתאות שנרצה-נציב ונקבל $F_s = 11$

בנוסף נשים לב שאכן רק הרכיב הזה ייעבור בטווח המסנן של $\frac{\pi}{4}$:

$$2\pi - \frac{2\pi 10}{11} = \frac{2}{11}\pi < \frac{\pi}{4}$$

ושהאיברים האחרים לא ייעברו: $2\pi - \frac{2\pi 5}{11} = \frac{12\pi}{11} > \frac{\pi}{4}$

$$2\pi\frac{5}{11} = \frac{10\pi}{11} > \frac{\pi}{4}$$

ואכן נקבל את הדרוש.

חלק ג' – OVA

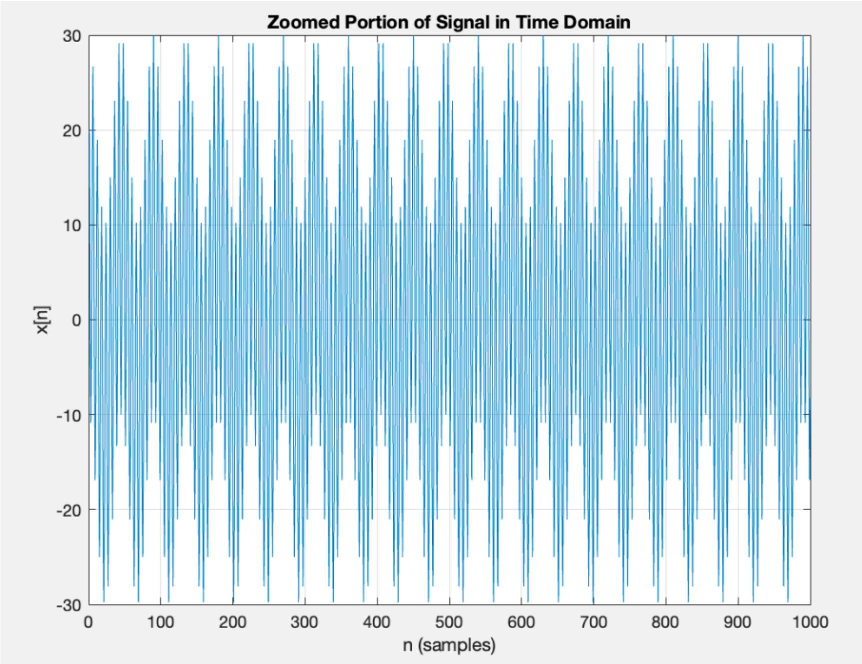
בחלק זה עליכם לממש את שיטת OVA עליה דיברנו בכיתה.

נתון אות ממשי $\{x[n] \in \mathbb{R}: n = 0, \dots, N\}$ ושני מסננים $h_1[n], h_2[n]$. קצב הדגימה הינו 18000 לשנייה. ברצוננו לממש את הקונבולוציה הלינארית:

$$y[n] = x[n] * h[n]$$

בכדי לבצע את הקונבולוציה הלינארית בצורה יעילה אנו נשתמש בשיטת *OVA*.

א. על מנת לטעון את האות $x[n]$, הורידו את הקובץ *sig_x.mat* מאתר הקורס. כיצד נראה הסיגנל? מהם התדרים הפעילים?



- תדרים פעילים באות (מעל לסף):

3000 400

- הייצוג המתמטי של האות:

$$A_1 cos(\frac{2\pi \cdot 400n}{18000 + 0.14})$$

$$A_2 cos(\frac{2\pi \cdot 3000n}{18000 + 1.05})$$

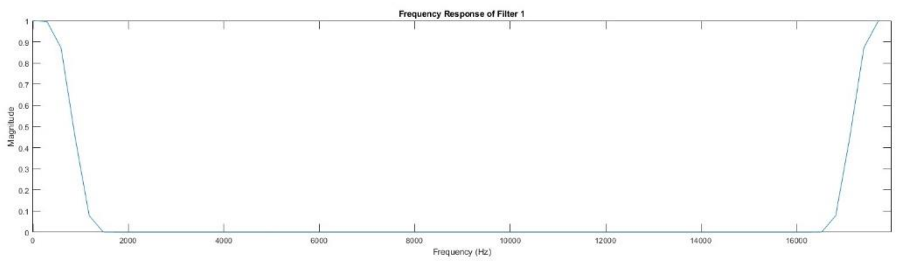
קוד מטלב:

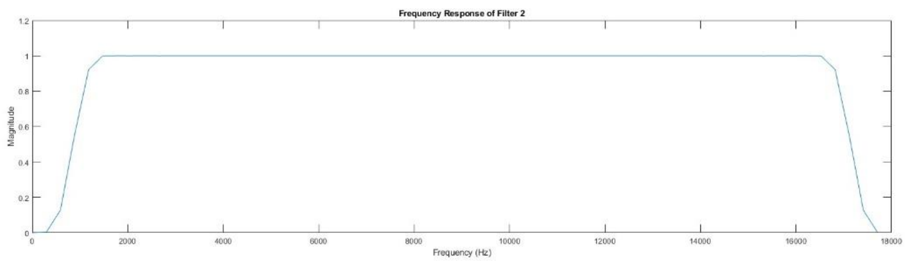
```
%% Q1:
%% What does the signal "sig_x.mat" look like? What are the active
frequencies?

% Load the signal from the .mat file and define initial parameters
load('sig_x.mat');  % Loads the variable 'x'
fs = 18000;  % Sampling rate in Hz
N = length(x);  % Length of the signal

% Define the time vector for the entire signal and for the zoomed portion
n = 0:N-1;  % Time vector for the entire signal
zoomRange = 1:min(N, 300);  % Zoomed range

% Plot the zoomed portion of the signal in the time domain
figure;
plot(n(zoomRange), x(zoomRange));
title('Zoomed Portion of Signal in Time Domain');
xlabel('n (samples)');
ylabel('x[n]');
grid on;

% Compute the FFT of the signal and define the frequency axis
```

```
X = fft(x);
f = (-N/2:N/2-1) * (fs / N);  % Frequency axis centered around 0

% Compute and plot the magnitude spectrum centered around 0
magnitude = abs(fftshift(X));
figure;
plot(f, magnitude);
title('Magnitude Spectrum of the Signal (Centered)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% Detect active frequencies using a threshold
threshold = max(magnitude) * 0.1;
active_indices = find(magnitude > threshold);
active_frequencies = f(active_indices);

disp(active_frequencies);

fprintf('Mathematical representation of the signal:\n');
phases = angle(fftshift(X));  % Assign to a variable first
for i = 1:length(active_frequencies)
    fprintf('A%d * cos(2 * pi * %.0f * n / %.0f + %.2f)\n', i, 
abs(active_frequencies(i)), fs, phases(active_indices(i)));
end
```

ב. על מנת ליצור את המסננים הורידו את הקבצים *filter_1.mat, filter_2.mat* מאתר הקורס. מהם סוגי המסננים?

המסנן הראשון "*filter_1.mat*" הוא *LPF*, ואילו המסנן השני "*filter_2.mat*" הוא *HPF*:





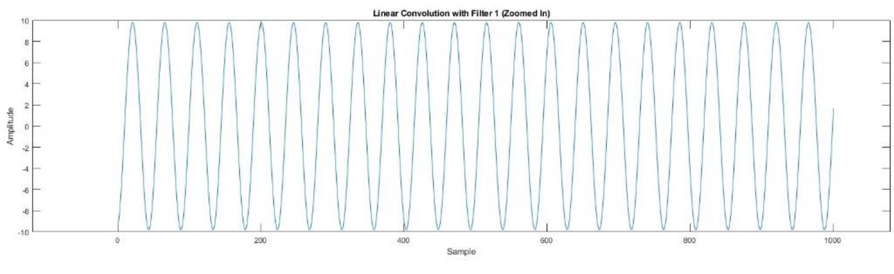
קוד מטלב:

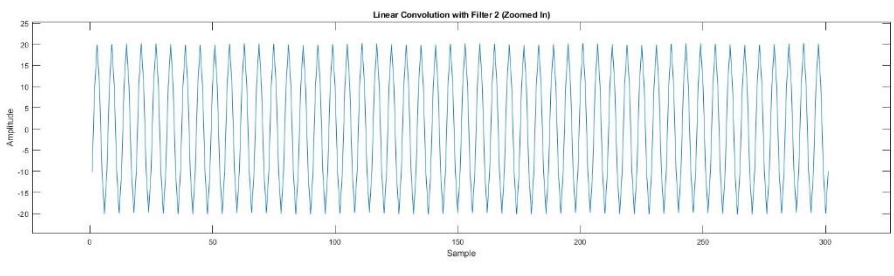
```
%% Q2:
%% What are the types of filters filter_1.mat and filter_2.mat?
```

```
filter_1 = load('filter_1.mat');
filter_2 = load('filter_2.mat');
filter_1_varname = fieldnames(filter_1);
filter_2_varname = fieldnames(filter_2);
h1 = filter_1.(filter_1_varname{1}); % Access the first variable in
filter_1.mat
h2 = filter_2.(filter_2_varname{1}); % Access the first variable in
filter_2.mat

% Compute and plot the frequency response of each filter
h1_fft = fft(h1);
h2_fft = fft(h2);
freq = (0:length(h1_fft)-1) * (18000 / length(h1_fft));

figure;
subplot(2,1,1);
plot(freq, abs(h1_fft));
title('Frequency Response of Filter 1');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(2,1,2);
plot(freq, abs(h2_fft));
title('Frequency Response of Filter 2');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

ג. ממשו באופן ישיר קונבולוציה לינארית בין הסיגנל $x[n]$ לכל אחד מהמסננים. השוו בין התוצאות והסבירו אותם. מהו זמן הריצה?

תוצאות הקונבולוציה מראות את ההשפעה של כל מסנן על הסיגנל $x[n]$. המסננים מעבירים תדרים שונים ומנחיתים אחרים, כפי שניתן לראות מהגרפים של הקונבולוציה הלינארית:





זמני הריצה הקצרים מראים יעילות בביצוע החישוב, עם זמני ריצה כמעט זהים לשני המסננים.

Linear convolution with Filter 1 (BSF) took 0.115626 seconds.

Number of multiplications with Filter 1: 16470000

Number of additions with Filter 1: 16200000

Linear convolution with Filter 2 (BPF) took 0.065639 seconds.

Number of multiplications with Filter 2: 16470000

Number of additions with Filter 2: 16200000

קוד מטלב:

```
%% Q3:
%% Implement a Directly Linear Convolution between the signal x[n] and each
of the filters.
%% Compare the results and explain them. What is the running time?
% Compute linear convolution with each filter and count operations
tic;
[y1_linear, mult1, add1] = linear_conv(x, h1);
time_y1_linear = toc;

tic;
[y2_linear, mult2, add2] = linear_conv(x, h2);
time_y2_linear = toc;

% Display results
fprintf('Linear convolution with Filter 1 (LPF) took %.6f seconds.\n',
time_y1_linear);
fprintf('Number of multiplications with Filter 1: %d\n', mult1);
fprintf('Number of additions with Filter 1: %d\n', add1);

fprintf('Linear convolution with Filter 2 (HPF) took %.6f seconds.\n',
time_y2_linear);
fprintf('Number of multiplications with Filter 2: %d\n', mult2);
fprintf('Number of additions with Filter 2: %d\n', add2);

% Plot the results of linear convolution
figure;
subplot(2,1,1);
plot(y1_linear(1200:1500));
title('Linear Convolution with Filter 1 (Zoomed In)');
xlabel('Sample');
ylabel('Amplitude');

subplot(2,1,2);
plot(y2_linear(1200:1500));
title('Linear Convolution with Filter 2 (Zoomed In)');
xlabel('Sample');
ylabel('Amplitude');
```

ד. ממשו קונבולוציה לינארית על ידי *OVA*. הסבירו כיצד קבעתם את פרמטרי האלגוריתם. מהו זמן הריצה האופטימלי? הציגו זאת בגרף כתלות בגודל המסגרת. הסבירו באופן מפורט כיצד עובדת השיטה.

קוד מטלב:

```
%% Q4:
%% Implament linear convolution by OVA.
%% Explain how you determined the parameters of the algorithm.
%% What is the optimal running time? Show this in a graph as a function of
frame size.
%% Explain in detail how the method works.

% Define the segment length (L)
L = 1024;  % Adjust the segment length as needed
tic;
% Perform the convolution using the overlap_and_add function
[y1, num_mult, num_add] = overlap_and_add(x, h1, L);
time = toc;

fprintf('Linear convolution with Filter 1 (BPF) took %.6f seconds.\n',
time);
fprintf('Number of multiplications with Filter 1: %d\n', num_mult);
fprintf('Number of additions with Filter 1: %d\n', num_add);

tic;
% Perform the convolution using the overlap_and_add function
[y2, num_mult, num_add] = overlap_and_add(x, h2, L);
time = toc;

fprintf('Linear convolution with Filter 2 (BPF) took %.6f seconds.\n',
time);
fprintf('Number of multiplications with Filter 2: %d\n', num_mult);
fprintf('Number of additions with Filter 2: %d\n', num_add);
% Plot the result in the time domain (zoomed in to improve visibility)
figure;
n = 0:length(y1)-1;  % Time vector

% Define the zoom range (adjust as needed)
zoom_start = 200;
zoom_end = 500; % Plot first 5000 samples for better visibility

subplot(2,1,1);
plot(n(zoom_start:zoom_end), y1(zoom_start:zoom_end));
title('Zoomed Result of Convolution using Overlap and Add - filter1');
xlabel('n (samples)');
ylabel('y1[n]');

subplot(2,1,2);
plot(n(zoom_start:zoom_end), y2(zoom_start:zoom_end));
title('Zoomed Result of Convolution using Overlap and Add - filter2');
xlabel('n (samples)');
ylabel('y2[n]');
```

Filter1

Number of multiplications : 16428032

Number of additions: 16158720

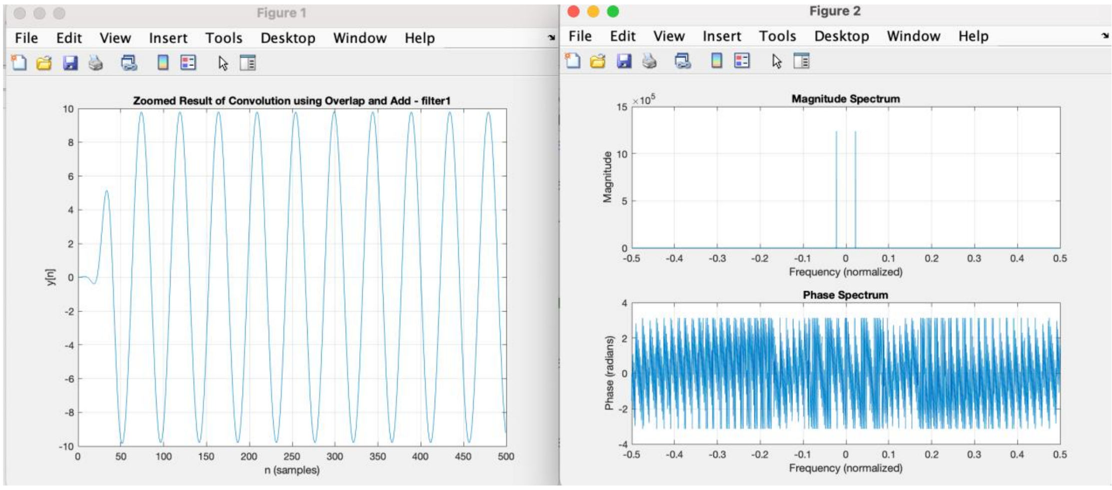
Linear convolution with Filter 1 took 0.042599 seconds.

L = 2048

Number of multiplications : 16365568

L = 4

Number of multiplications : 16469756

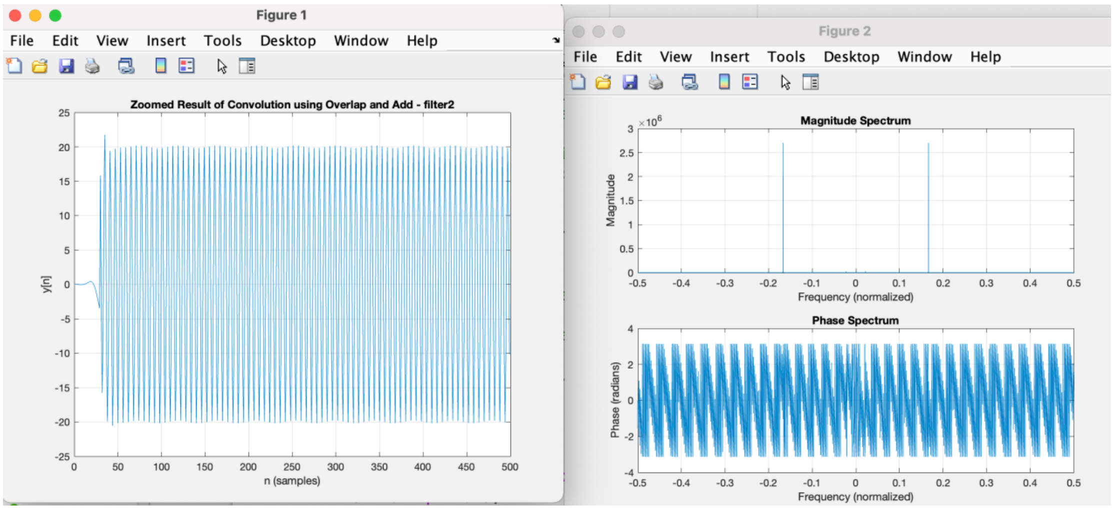


Filter2

Number of multiplications : 16428032

Number of additions: 16158720

Linear convolution with Filter 2 took 0.043691 seconds.



ה. השוו בין זמני הריצה של שתי השיטות על אותו הגרף כפונקציה של גודל המסגרת. לאיזו שיטה ישנה עדיפות מבחינת הביצועים?

אין שיפור משמעותי בכמות הפעולות בשימוש בשיטה *OVA*, אך החשיבות של השיטה היא שניתן לחלק קונבולוציה אחד גדולה מאוד (שיכולה אפילו לגלוש מהזיכרון הזמין) למספר קונבולוציות קטנות יותר, ככה שניתן:
- למקבל את הקונבולוציה (אם יש יותר מליבה אחד)
-לחשב את הקונבולוציה גם אם לכאורה לא היה מספיק זכרון למחשב

קוד מטלב:

```
%% Q5:
%% Compare the running times of the two methods on the same graph as a
function of frame size. Which method has better in terms of performance?
%% Draw on the same graph the output of the 2 types of convolution for each
of the filters and show that you performed OVA correctly.
% Linear Convolution using the provided linear_conv function
[y1_linear, unus1, unus2] = linear_conv(x, h1);
[y2_linear, unus1, unus2] = linear_conv(x, h2);

% Overlap-Add Convolution using the provided overlap_and_add function
L = 2048; % Block size
[y1_ova, unus1, unus2] = overlap_and_add(x, h1, L);
[y2_ova, unus1, unus2] = overlap_and_add(x, h2, L);

% Plotting the results

% Zoomed plot for better visibility
zoom_start = 1;
zoom_end = 500; % Adjust as needed

figure;
subplot(2,1,1);
hold on;
plot(zoom_start:zoom_end, y1_linear(zoom_start:zoom_end), 'r',
'DisplayName', 'Linear Convolution', 'LineWidth', 1.5);
plot(zoom_start:zoom_end, y1_ova(zoom_start:zoom_end), 'b--',
'DisplayName', 'OVA Convolution', 'LineWidth', 1.5);
title('Filter 1 (Zoomed)');
legend('show');
xlabel('n (samples)');
ylabel('y[n]');
grid on;
hold off;

subplot(2,1,2);
hold on;
plot(zoom_start:zoom_end, y2_linear(zoom_start:zoom_end), 'r',
'DisplayName', 'Linear Convolution', 'LineWidth', 1.5);
plot(zoom_start:zoom_end, y2_ova(zoom_start:zoom_end), 'b--',
'DisplayName', 'OVA Convolution', 'LineWidth', 1.5);
title('Filter 2 (Zoomed)');
legend('show');
xlabel('n (samples)');
ylabel('y[n]');
```

```
grid on;
hold off;
```

ו. ציירו על אותו הגרף את המוצא של *2* סוגי הקונבולוציה עבור כל אחד מהמסננים והראו כי ביצעתם *OVA* כראוי.