Problem 1

(a) Perfect noise cancellation is achieved if $y[n] = \tilde{r}[n]$, where $\tilde{r}[n]$ is the noise $r[n]$ filtered by $H(z)$. Therefore, the transfer function of the adaptive filter $F(z)$ is simply

$$F(z) = H(z) \tag{1}$$

(b) If $x[n]$ (the desired response) and $r[n]$ (the input) are uncorrelated, then the vector P is the zero vector, since the ith entry of P is simply $P_i = \mathbb{E}(x[n]r[n-i]) = 0$. Therefore,

$$W^\star = R^{-1}P = 0. \tag{2}$$

That is, the Wiener solution is zero, which implies that all the weights of the adaptive filter are zero. As a result, for any input, the adaptive filter produces a zero output.

(c) Note that the R matrix is simply $R = \sigma_r^2 I_{L+1}$, where I_{L+1} is the $(L+1) \times (L+1)$ identity matrix. Therefore,

$$\mu = \frac{0.001}{\text{trace}(R)} = \frac{0.001}{(L+1)\sigma_r^2} = 0.01 \tag{3}$$

(d) Since $R = \sigma_r^2 I_{L+1}$, all eigenvalues of R are equal to $\lambda = \sigma_r^2$. The time constant of the LMS algorithm is given by

$$\tau = \frac{1}{4\mu\lambda} = 12,500 \text{ samples} \tag{4}$$

$$T = \tau / F_s = 0.63 \text{ seconds} \tag{5}$$

Therefore, it takes roughly $4T = 2.27$ seconds for the LMS algorithm to converge. This convergence time could be substantially shortened by increasing the adaptation constant μ.

(e) Assuming that μ still satisfies

the stability condition $\mu < \text{trace}(R)$, we can make the following conclusions:

- The convergence time or learning curve time constants are inversely proportional to the adaptation constant. Therefore, the convergence time decreases by increasing μ.
- The minimum mean square error does not depend on the adaptation constant. Hence, it remains the same.
- The excess MSE is proportional to the misadjustment, which in turn is proportional to μ. Hence, the excess MSE increases by increasing μ.

(f) The code is attached at the end of this file. The requested plots are shown below.

Ideally the adaptive filter would converge to $H(z)$. However, since $H(z)$ is IIR the adaptive FIR filter of order $L = 49$ cannot match it perfectly. Note that the first 20 coefficients of the adaptive

filter are zero because of the 20-sample delay in $H(z)$.

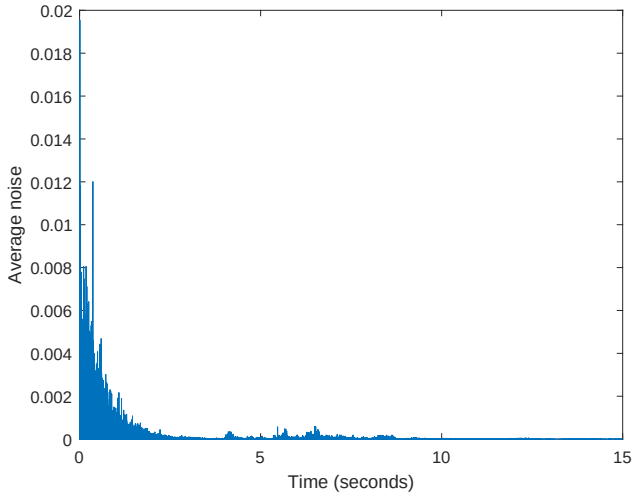
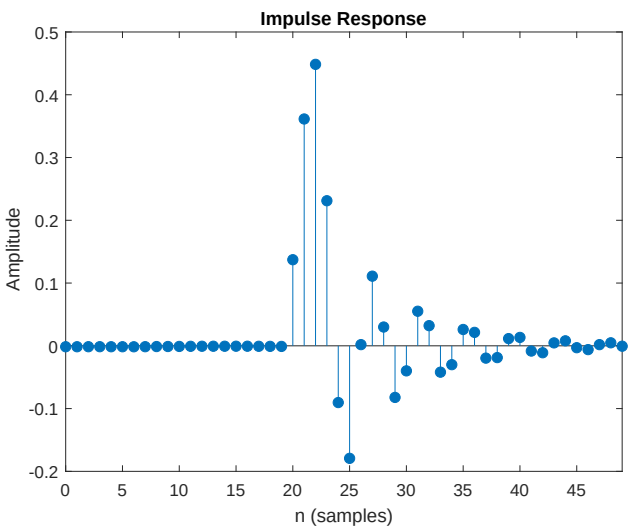
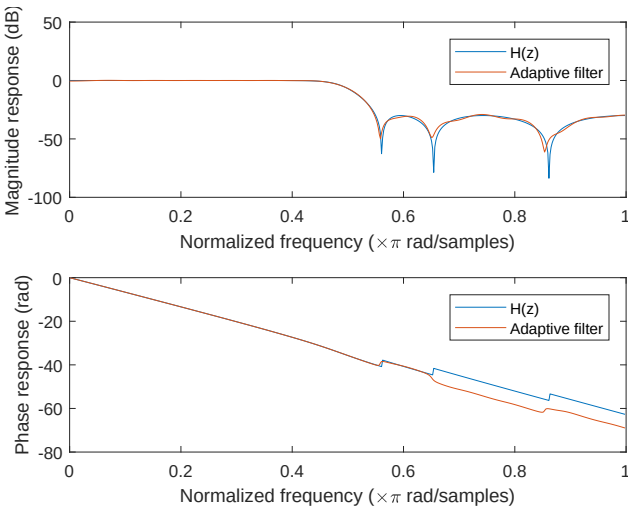


Figure 1: Average noise averaged over 20 independent noise realizations. Note that convergence happens approximately after 4.5 seconds, which is consistent with part 1(d)



(a) Adaptive filter coefficients after adaptation.



(b) Magnitude and phase response of adaptive filter after adaptation compared to $H(z)$.

Figure 2: Coefficients and frequency response of adaptive filter.

Code for part (1)

```
%% Noise canceling
clear, clc, close all

[s, Fs] = audioread('guitartune.wav'); % Fs is sampling frequency

Nruns = 20; % number of independent runs to average learning curve
var_r = 2e-3; % variance of white Gaussian noise

L = 49; % adaptive filter order. L+1 coefficients

```

```
% H(z) filter coefficients
% Although H(z) is "unknown", for developing your answers you can use this
% filter. It is a eighth-order IIR Chebyshev type II filter with bandwidth
% 0.55*Fs. Chebyshev type II filters have constant gain at the passband,
% but they exhibit ripples in the stopband.
[hb, ha] = cheby2(6, 30, 0.55);
hb = [zeros(1, 20) hb]; % Add a delay of 20 samples to H(z)
% Hint: for filtering, use the function filter, e.g., output = filter(hb, ha, input)

%% Your solutions go here
% Matlab hints:
% - The function randn() generates zero-mean, unit-variance,
% Gaussian-distributed numbers
% - Fixed filters can be easily implement using the function filter()
% - Adaptive filters are can be implemented in a for loop
% - To plot the coefficients of an FIR filter, use the function impz
% (impulse response)
% - To plot the magnitude and phase response, use the function freqz()
% - To play a signal use the function sound(x, Fs). Don't forget to include
% the sampling frequency when calling sound, otherwise Matlab will assume
% Fs = 8 kHz, which is not the correct value.
sound(s, Fs) % example


%% Your solutions go here
% (C) adaptation constant
lamb = var_r; % all eigenvalues are equal to var_r since R = var_r*I
mu = 0.001/((L+1)*var_r)

% (D) convergence time
tau = 1/(4*mu*lamb); % learning curve time constants
T_conv = 4*tau/Fs % convergence time

% (F) Implementation
% Generate input white noise
r = sqrt(var_r)*randn(size(s));
rf = filter(hb, ha, r);

% Generate signal x
x = s + rf; % from diagram in Fig. 2

avg_noise = zeros(size(s));
for q = 1:Nruns
    y = zeros(size(s)); % output of adaptive filter
    e = zeros(size(s)); % error signal
```

```
    W = zeros(1, L+1); % initial weight vector
    for k = L+1:length(s)
        y(k) = W*r(k:-1:k-L);
        e(k) = x(k) - y(k);
        W = W + 2*mu*e(k)*r(k:-1:k-L).';
    end

    avg_noise = avg_noise + (s-e).^2;
end

avg_noise = avg_noise/Nruns;

figure, box on
plot((0:length(s)-1)/Fs, avg_noise)
xlabel('Time (seconds)')
ylabel('Average noise')
saveas(gca, '../figs/part1_average_noise', 'epsc')

figure, impz(W)
saveas(gca, '../figs/part1_coeff', 'epsc')

[H1, w] = freqz(hb, ha);
H2 = freqz(W, 1, w);

figure
subplot(211), hold on, box on
plot(w/pi, 20*log10(abs(H1)))
plot(w/pi, 20*log10(abs(H2)))
xlabel('Normalized frequency (\times\pi rad/samples)')
ylabel('Magnitude response (dB)')
legend('H(z)', 'Adaptive filter')

subplot(212), hold on, box on
plot(w/pi, unwrap(angle(H1)))
plot(w/pi, unwrap(angle(H2)))
xlabel('Normalized frequency (\times\pi rad/samples)')
ylabel('Phase response (rad)')
legend('H(z)', 'Adaptive filter')
saveas(gca, '../figs/part1_freqz', 'epsc')
```

Problem 2

(a) The DFT of a constant signal is an impulse scaled by the d.c. value:

$$X[k] = \{8, 0, 0, 0, 0, 0, 0, 0\} \tag{6}$$

(b) This signal can be represented as $x[n] = e^{j\pi n}, n = 0, \ldots, 7$. Start with the result from part (a) and use the circular time shift property of the DFT. Note that for $N = 8$,

$$(-1)^n = e^{j\pi n} = W_N^{-nN/2} \tag{7}$$

Using the circular frequency shift property, the DFT is a shift of the DFT from part (a) by $N/2 = 4$: $X[k] = \{0, 0, 0, 0, 8, 0, 0, 0\}$

(c) This is similar to part (b). Note that for $N = 8$

$$e^{j\pi/2n} = e^{j2\pi/NN/4n} = W_N^{-N/4n}. \tag{8}$$

Using the circular frequency shift property, the DFT is a shift of the DFT from part (a) by $N/4 = 2$: $X[k] = \{0, 0, 8, 0, 0, 0, 0, 0\}$

(d) Start with the the result of part (c). Note that $\sin(\pi n/2) = -j/2(e^{j\pi n/2} - e^{-j\pi n/2})$. Using the circular frequency shift property, we find that $X[k] = \{0, 0, -4j, 0, 0, 0, 4j, 0\}$

(e) The DFT of an impulse is constant in frequency: $X[k] = \{1, 1, 1, 1, 1, 1, 1, 1\}$.

(f) Circularly shifting a signal in time corresponds to multiplication by a complex exponential in frequency. Using part (e), we find that: $X[k] = \{1, -j, -1, j, 1, -j, -1, j\}$.

Problem 3

Signal	DFT	Justification
1	d	$x[n]$ is real and $x[n] = x[((-n))_N]$ ($\tilde{x}[n]$ is even), so $X[k]$ is purely real, thus $\angle X[k] \in \{0, \pi\}$. Also, $\sum_{n=0}^{N-1} x[n] = X[0] = 5$.
2	a	$x[n]$ is real and $x[n] = x[((-n))_N]$ ($\tilde{x}[n]$ is even), so $X[k]$ is purely real, thus $\angle X[k] \in \{0, \pi\}$. Also, $\sum_{n=0}^{N-1} x[n] = X[0] = 1$.
3	c	$x[n]$ is real and $x[n] = -x[((-n))_N]$ ($\tilde{x}[n]$ is odd), so $X[k]$ is purely imaginary, thus $\angle X[k] \in \{-\pi/2, \pi/2\}$. Also, $\sum_{n=0}^{N-1} x[n] = X[0] = 0$.
4	f	$x[n]$ is delayed impulse, so $\lvert X[k]\rvert = 1 \; \forall \; k$
5	b	Signal 5 is a delayed version of signal 1, so $\lvert X[k]\rvert$ same as for signal 1. But signal 5 lacks symmetry, so $\angle X[k]$ is more complicated than for signal 1.
6	e	Signal 6 is a delayed version of signal 3, so $\lvert X[k]\rvert$ same as for signal 3. But signal 6 lacks symmetry, so $\angle X[k]$ is more complicated than for signal 3.

Problem 4: Linear convolution and circular convolution

(a)

We will have $x[n] * h[n] = x[n] \textcircled{N} h[n]$ as long as

$$N \geq L + P - 1 \tag{9}$$

(b)

$$\begin{aligned} c_{xy}[m] &= x[m] * y^*[-m] \\ &= x[m] \textcircled{N} y^*[-m] \qquad \text{(as long as } N \geq 2L-1\text{)} \\ &= \text{IFFT}\{\text{FFT}\{x[m]\}(\text{FFT}\{y[m]\})^*\}, \end{aligned}$$

where the last equality follows since circular convolution in time domain corresponds to product of DFTs in frequency domain. Moreover, we've used the conjugate and time reversal properties of the DFT i.e., $\text{FFT}\{y^*[-m]\} = (\text{FFT}\{y[m]\})^*$. The FFT/IFFT involved in this calculation are $N = 2L - 1$.

As mentioned in the hint, this computation will provide $c_{xy}[m]$ indexed from 0 up to $N-1$. We would need to use the function `fftshfit`, for instance, to produce $c_{xy}[m]$ in the conventional representation from $-N/2+1$ up to $N/2-1$.

(c)

If $x[n] = y[n]$, then

$$c_{xx}[m] = \text{IFFT}\{|\text{FFT}\{x[m]\}|^2\} \tag{10}$$

Hence this computation requires one fewer FFT.

Problem 5: Overlap-add or overlap-save

(a)

The code for for part (a) is attached at the end of this question. The implementation assumes FFT length $N = 256$, and $M = 52$.

The Figures below show the results of comparison for overlap-add and overlap-save methods. We obtain the same results as with direct implementation of the FIR filter. The disagreement at the end of the sequence is simply because we did not have an entire block to perform block convolution.

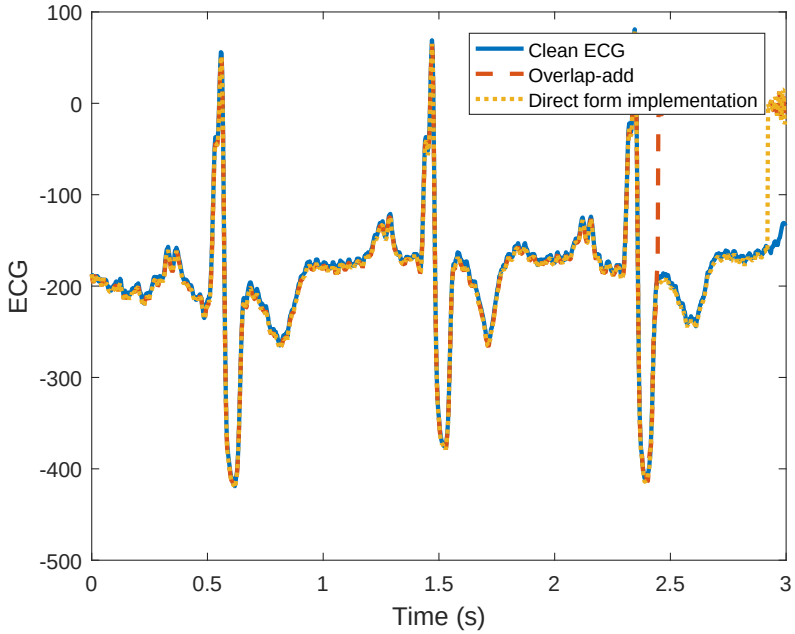


Figure 3: Result using the overlap-add method

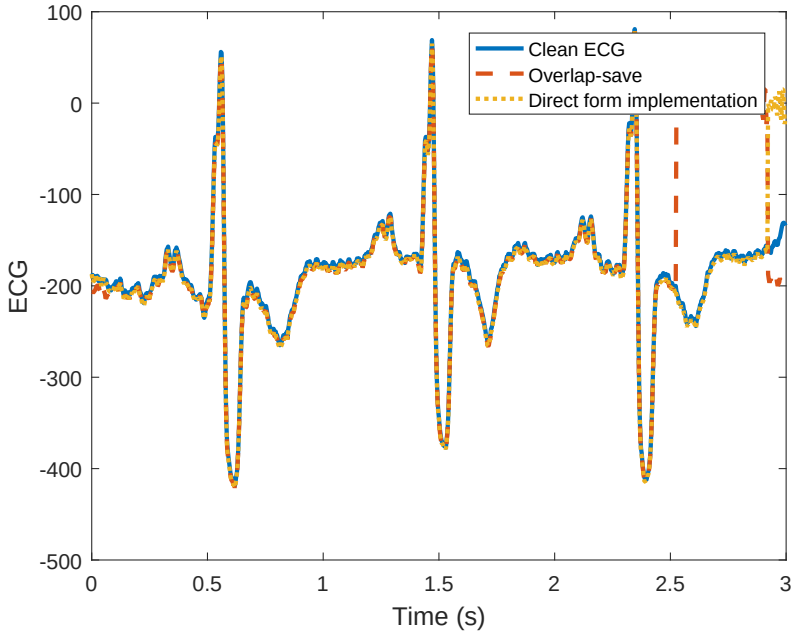


Figure 4: Result using the overlap-save method.

(b)

Without any simplification, the block convolution requires 3 FFTs (including IFFT) are necessary to produce $N - M$ useful output samples. Therefore,

$$C = \frac{(3 \times (2N(\log_2(N) - 1)) + N)}{N - M} = 53.4369 \qquad \text{(block convolution)}$$

for $N = 256$ and $M = 52$.

The direct form implementation of a Mth-order FIR filter (without any symmetry) requires $M + 1$ multiplications per useful output. Thus,

$$C = M + 1 = 53 \qquad \text{(FIR direct implementation)}$$

Simplifications:

Note that the FFT to compute the filter coefficients may be computed in advance and stored in a memory since the filter is assumed constant. Thus, we could avoid 1 FFT:

$$C = \frac{(2 \times (2N(\log_2(N) - 1)) + N)}{N - M} = 36.038 \qquad \text{(block convolution)}$$

assuming $N = 256$ and $M = 52$.

An linear phase FIR filter requires $\lfloor (M+1)/2 \rfloor$ multiplications per useful output, since equal coefficients can be combined. Therefore,

$$C = \lfloor (M + 1)/2 \rfloor = 26 \qquad \text{(linear phase FIR direct implementation)}$$

For $M = 52$.

Solutions with or without these simplifications are accepted.

(c)

For both overlap-add and overlap-save methods, we need 3 FFTs (including IFFT), and an N-point multiplication. Moreover, for each block of length L only $N - M$ outputs are useful. In the overlap-save method there are M unusable outputs per block. In the overlap-add method the M outputs will only be useful at the next block.

$$C = \frac{3 \times (2N(\log_2(N) - 1)) + N}{N - M} \qquad \text{(block convolution)}$$

Assuming that the DFT of the filter coefficients is computed in advance:

$$C = \frac{2 \times (2N(\log_2(N) - 1)) + N}{N - M} \qquad \text{(block convolution 2 FFTs)}$$

Both results for C are accepted. The figure below assumes that C was computed in accordance with the equation above, assuming that the DFT of the filter coefficients was computed in advance:

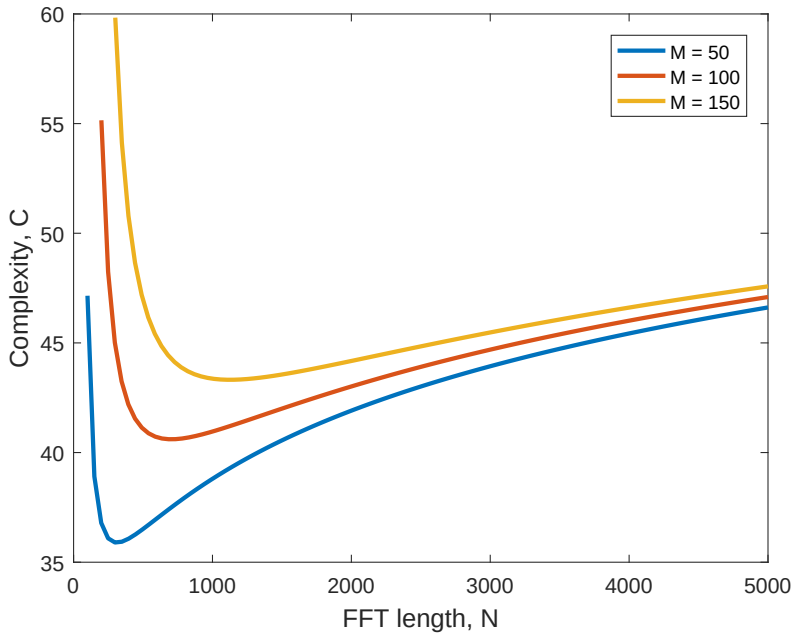


Figure 5: Complexity of overlap-save or overlap-add method as a function of the FFT length. These curves assumed that the DFT of the filter coefficients was computed in advance, hence saving one FFT in the complexity calculations.

Note that for each filter length there is an optimal FFT length.

Moreover, note that as the filter-order becomes large, the FFT implementation becomes more attractive even if we assume that there's symmetry in the impulse response of the filter. For $M = 100$, the block convolution method has complexity close to $C = 40$, while a direct implementation assuming symmetry would have complexity $C = 50$.

Note: the particular formula for the number of real multiplications of N-point FFT considered in this problem is only valid when N is a power of 2. The actual number of multiplications in a FFT depend on whether the FFT is radix-2, radix-4, prime radix, etc. The actual values will vary slightly, but the main conclusions from this problem remain the same.

(d) (optional)

See Matlab code below for solution. Plots are shown in part (a).

Matlab code for Problem 4

```
%% Overlap and save or overlap and add
clear, clc, close all

load('notch_filter_coeff'); % loads filter coefficients
load('ecg_recording.mat')  % loads ECG data

% Definitions
Nfft = 256;     % FFT length
M = length(hls) - 1; % M = 52
```

```
Loa = Nfft - M; % block length for overlap-add
Los = Nfft;     % block length for overlap-save
H = fft(hls, Nfft); % FFT of filter coefficients

t = 0:T:(N-1)*T; % time vector
ecg_fir = filter(hls, 1, ecg_60Hz);
ecg_fir = circshift(ecg_fir, [0, -M/2]); % remove group delay

%% Overlap-add
yoa = zeros(size(ecg_60Hz));
select = (1:Loa);
for r = 0:floor(length(ecg_60Hz)/Loa)-1
    xr = ecg_60Hz(select);           % non-overlapping segments of length Loa
    yr = ifft(H.*fft(xr, Nfft));    % compute Nfft-point circular convolution
    yoa(select(1):(select(1)+Nfft-1)) = yoa(select(1):(select(1)+Nfft-1)) + yr; % add
    select = select + Loa;
end

figure, hold on, box on
n = 0:length(ecg_60Hz)-1;
yoa = circshift(yoa, [0, -M/2]); % remove group delay
plot(t, ecg_clean, 'LineWidth', 2, 'DisplayName', 'Clean ECG')
plot(t, yoa, '--', 'LineWidth', 2, 'displayname', 'Overlap-add')
plot(t, ecg_fir, ':', 'LineWidth', 2, 'displayname', 'Direct form implementation')
legend('-dynamiclegend')
xlabel('Time (s)', 'FontSize', 12)
ylabel('ECG', 'FontSize', 12)
saveas(gca, '../figs/fir_notch_overlap_add', 'epsc')

%% Overlap-save
yos = zeros(size(ecg_60Hz));
select = (1:Los);
for r = 0:floor(length(ecg_60Hz)/Los)
    xr = ecg_60Hz(select);           % non-overlapping segments of length Loa
    yr = ifft(H.*fft(xr, Nfft));    % compute Nfft-point circular convolution
    yoa(select(M+1:end)) = yr(M+1:end); % save
    select = select + Los - M;
end

figure, hold on, box on
n = 0:length(ecg_60Hz)-1;
yoa = circshift(yoa, [0, -M/2]); % remove group delay
plot(t, ecg_clean, 'LineWidth', 2, 'DisplayName', 'Clean ECG')
plot(t, yoa, '--', 'LineWidth', 2, 'displayname', 'Overlap-save')
plot(t, ecg_fir, ':', 'LineWidth', 2, 'displayname', 'Direct form implementation')
```

```
55 legend('-dynamiclegend')
56 xlabel('Time (s)', 'FontSize', 12)
57 ylabel('ECG', 'FontSize', 12)
58 saveas(gca, '../figs/fir_notch_overlap_save', 'epsc')
59
60 %% Complexity calculation
61 M = 50:50:150;
62
63 C = @(N, M) (2*2*N.*(log2(N)-1) + N)./(N-M);
64
65 figure, hold on, box on
66 for k = 1:length(M)
67     N = linspace(M(k)*2, 5e3);
68     plot(N, C(N, M(k)), 'LineWidth', 2, 'DisplayName', sprintf('M = %d', M(k)))
69 end
70 xlabel('FFT length', 'FontSize', 12)
71 ylabel('Complexity', 'FontSize', 12)
72 legend('-dynamiclegend')
73 saveas(gca, '../figs/block_conv_complexity', 'epsc')
```

Problem 6: Spectrograms

(a) Spectrograms (a) and (c) were computed with the rectangular window. This can be infered from the amount of leakage in the spectrogram. This is a result from the large sidelobes of the rectangular window.

(b) Spectrograms (a) & (b) have approximately the same frequency resolution, as do spectrograms (c) & (d). Note that the the lines in these spectrograms have approximately the same thickness

(c) Spectrogram (c) has the shortest time window. Although spectrograms (c) and (d) have virtually the same frequency resolution i.e., same main-lobe width, the Hamming window has to be longer to achieve the same main-lobe width of a rectangular window. This is the reason that, for instance, a filter designed by window using the Hamming window would have slower roll-off than a filter designed using a rectangular window of same length.

(d) The discrete-time signal has three frequency components

$$x[n] = \begin{cases} A_1\cos(0.7\pi n + \phi_1) + A_2\cos(0.4\pi n + \phi_2), & 0 \le n \le 1000 \\ A_2\cos(0.4\pi n + \phi_2), & 1000 < n \le 2000 \\ A_2\cos(0.4\pi n + \phi_2) + A_3\cos(0.5\pi n + \phi_3), & 2000 < n \le 3000 \end{cases} \tag{11}$$

We cannot specify the amplitudes $\{A_1, A_2, A_3\}$ and the phases $\{\phi_1, \phi_2, \phi_3\}$.

To obtain the continuous-time signal, we simply replace n by t/T

$$x_c(t) = \begin{cases} A_1\cos(7000\pi t + \phi_1) + A_2\cos(4000\pi t + \phi_2), & 0 \le n \le 1000 \\ A_2\cos(4000\pi t + \phi_2), & 1000 < n \le 2000 \\ A_2\cos(4000\pi t + \phi_2) + A_3\cos(5000\pi t + \phi_3), & 2000 < n \le 3000 \end{cases} \tag{12}$$

Note that the anplitudes should be scaled by T, but we simply redefined the amplitudes for simplicity.

Problem 7: Frequency modulation

(a)

$$\Omega_i(t) = \frac{d\theta(t)}{dt} = \Omega_c + \beta\Omega_m\cos(\Omega_m t) \tag{13}$$

Hence, the maximum frequency deviation from Ω_c is $\beta\Omega_m$.

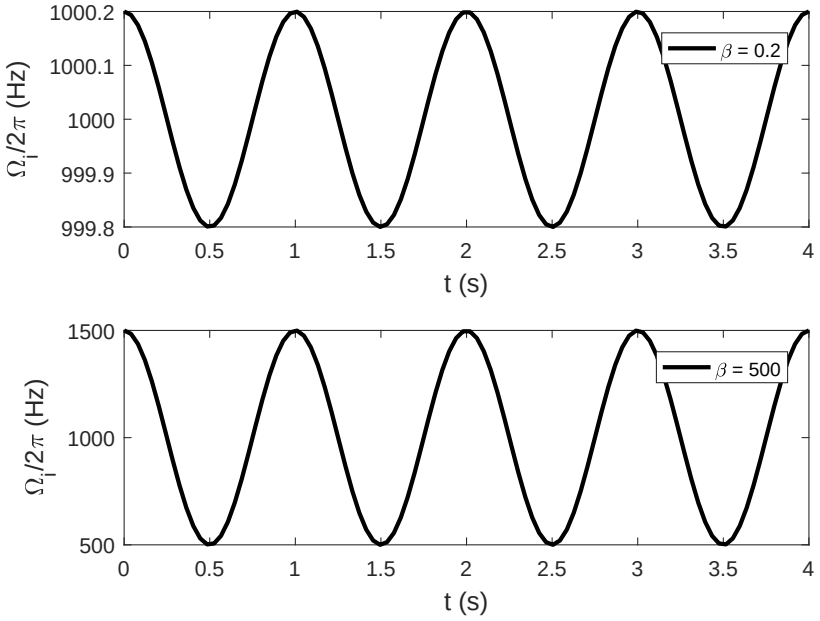


Figure 6: $\Omega_i(t)/(2\pi)$ for $\beta = 0.2$ (top) and $\beta = 500$ (bottom).

(b)

$$\begin{aligned} x_{FM}(t) &= \cos(\Omega_c t + \beta\sin(\Omega_m t)) \\ &= \cos(\Omega_c t)\cos(\beta\sin(\Omega_m t)) - \sin(\Omega_c t)\sin(\beta\sin(\Omega_m t)) \\ &\approx \cos(\Omega_c t) - \beta\sin(\Omega_m t)\sin(\Omega_c t) \qquad \text{(small-angle approximation when } \beta << \pi/2) \end{aligned}$$

By inspecting the equation above, we can see that $\mathcal{F}\{x_{FM}(t)\}$ has impulses at frequencies $\pm\Omega_c$, $\pm\Omega_c + \Omega_m$, and $\pm\Omega_c - \Omega_m$.

(c)

We simply replace $t = nT$ in (3) of the assignment,

$$x_{FM}[n] = x_{FM}(nT) = \cos(2\pi(1000)n/8000 + \beta \sin(2\pi n/8000))$$
$$\cos(\pi/4n + \beta \sin(\pi n/4000)) \tag{14}$$

(d)

Code for this part is attached at the end of this problem

(i)

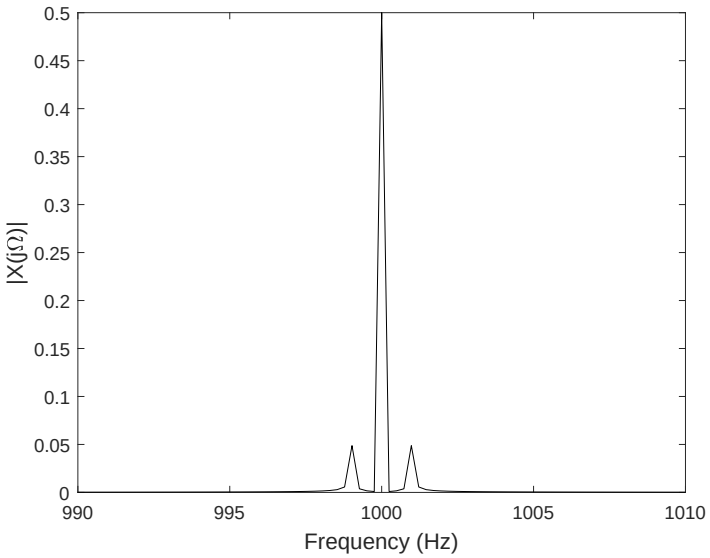


Figure 7: DFT of $x_{FM}(t)$ for $\beta = 0.2$.

(ii)

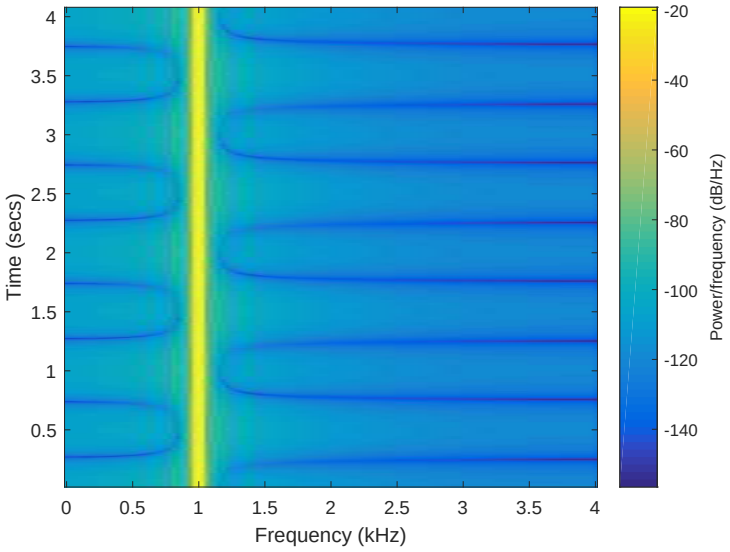


Figure 8: Spectrogram of $x_{FM}(t)$ for $\beta = 0.2$.

The spectrogram does not show the frequency variation observed in part (a) because the window does not have enough resolution to resolve the frequency variation when $\beta = 0.2$.

(e)

Code for this part is attached at the end of this problem

(i)

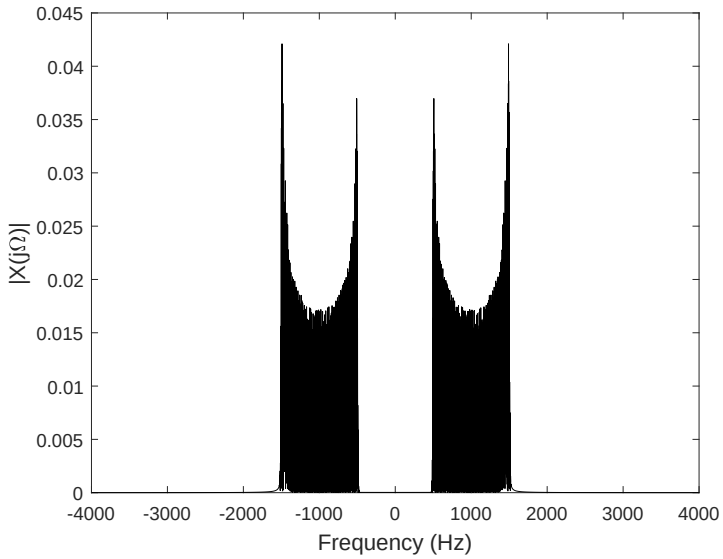


Figure 9: DFT of $x_{FM}(t)$ for $\beta = 500$.

(ii)

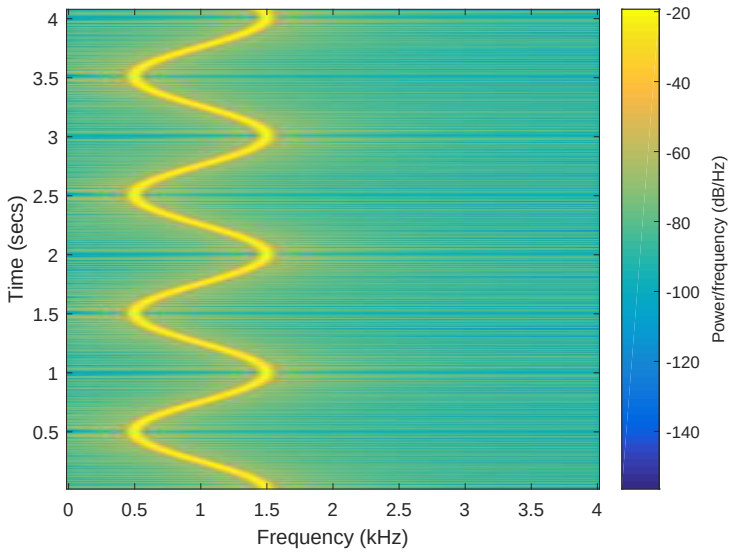


Figure 10: Spectrogram of $x_{FM}(t)$ for $\beta = 500$.

Now the frequency variation is large enough that we can observe it in the spectrogram.

(iii)

Increasing the window length improves frequency resolution. However, by making the window length too large may lead to excessive leakage, since the sidelobes area may become significant.

If Ω_m is small, we must make the window longer in order to improve the frequency resolution.

The window length does not depend on Ω_c provided that the frequency variations $\Omega_c + \beta\Omega_m$ does not cause aliasing. That is, $\Omega_c + \beta\Omega_m < \Omega_s/2$.

Code for Problem 6

```
%% Problem 6: Frequency modulation
%% a)
t = linspace(0,4);
Wc = 2*pi*1e3;
Wm = 2*pi;

Wi = @(beta, t) Wc + beta*Wm*cos(Wm*t);

figure
subplot(211)
plot(t, Wi(0.2, t)/(2*pi), 'k', 'LineWidth', 2);
xlabel('t (s)', 'FontSize', 12);
ylabel('\Omega_i/2\pi (Hz)', 'FontSize', 12);
legend('\beta = 0.2')

subplot(212)
plot(t, Wi(500, t)/(2*pi), 'k', 'LineWidth', 2)
```

```
xlabel('t (s)', 'FontSize', 12);
ylabel('\Omega_i/2\pi (Hz)', 'FontSize', 12);
legend('\beta = 500')
saveas(gca, '../figs/hw6_fm_a', 'epsc')

%% d)
beta = 0.2;
N = 32768;
n = 0:N-1;
%
x = cos(pi/4*n) - beta*sin(pi/4e3*n).*sin(pi/4*n);

% soundsc(x)

% (i)
dt = 1/8e3;
df = 1/(N*dt);
f = -1/(2*dt):df:1/(2*dt)-df;
X = fftshift(fft(x, N)/N);

figure
plot(f, abs(X), 'k')
ylabel('|X(j\Omega)|', 'FontSize', 12)
xlabel('Frequency (Hz)', 'FontSize', 12)
axis([990 1010 0 0.5])
saveas(gca, '../figs/hw6_fm_di', 'epsc')

% (ii)
figure
spectrogram(x, 256, 250, 256, 8e3);
saveas(gca, '../figs/hw6_fm_dii', 'epsc')

%% e)
beta = 500;
N = 32768;
n = 0:N-1;

x = cos(pi/4*n + beta*sin(pi*n/4e3));
% soundsc(x)

% (i)
dt = 1/8e3;
df = 1/(N*dt);
f = -1/(2*dt):df:1/(2*dt)-df;
X = fftshift(fft(x, N)/N);
```

```
63
64 figure
65 plot(f, abs(X), 'k')
66 ylabel('|X(j\Omega)|', 'FontSize', 12)
67 xlabel('Frequency (Hz)', 'FontSize', 12)
68 saveas(gca, '../figs/hw6_fm_ei', 'epsc')
69
70 % (ii)
71 figure
72 spectrogram(x, 256, 250, 256, 8e3);
73 saveas(gca, '../figs/hw6_fm_eii', 'epsc')
```