

Part 1

Report Biomedical Signal Processing

Ardalan Gerami 99102112

6/2/2024



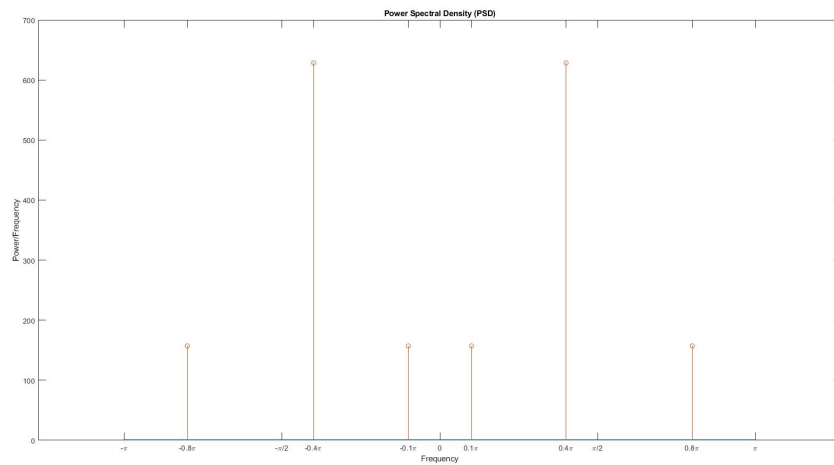
Contents

1	Question 1	3
1.1	part a	3
1.2	part b	3
1.3	part c	4
1.4	part d	4
1.5	part e	5
1.6	part f	6
1.7	part g	6
2	Question 2	19
2.1	part a	19
2.2	part b	21
2.3	part c	23
2.4	part d	23
2.5	part e	25
2.6	part f	26
3	Question 3	35
3.1	part a	35
3.2	part b	36
3.3	part c	36
3.4	part d	36

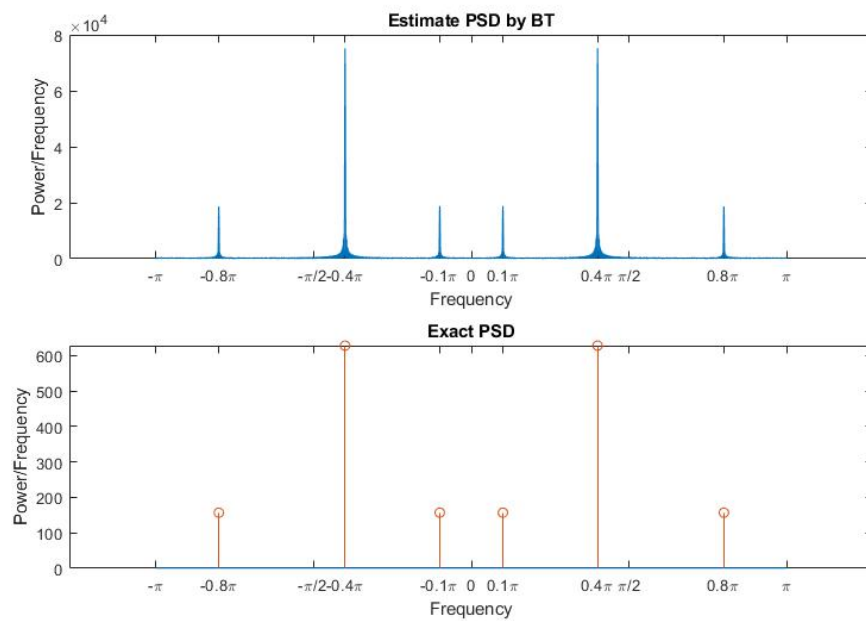
1 Question 1

1.1 part a

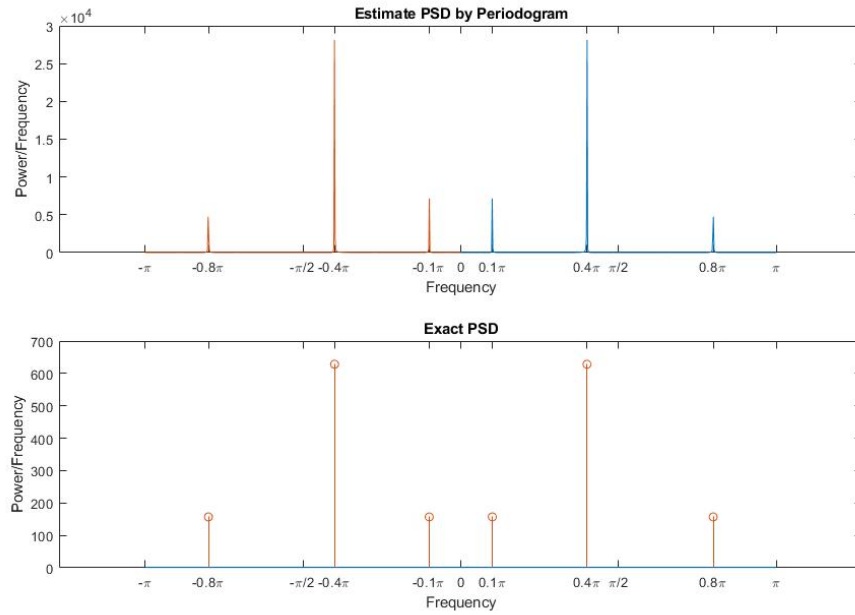
As we know for PSD of $A\cos(\omega_0 n + \phi)$ is $A^2/2 * \pi(\delta(\omega - \omega_0) + \delta(\omega + \omega_0))$ and the PSD of white noise is σ^2 .



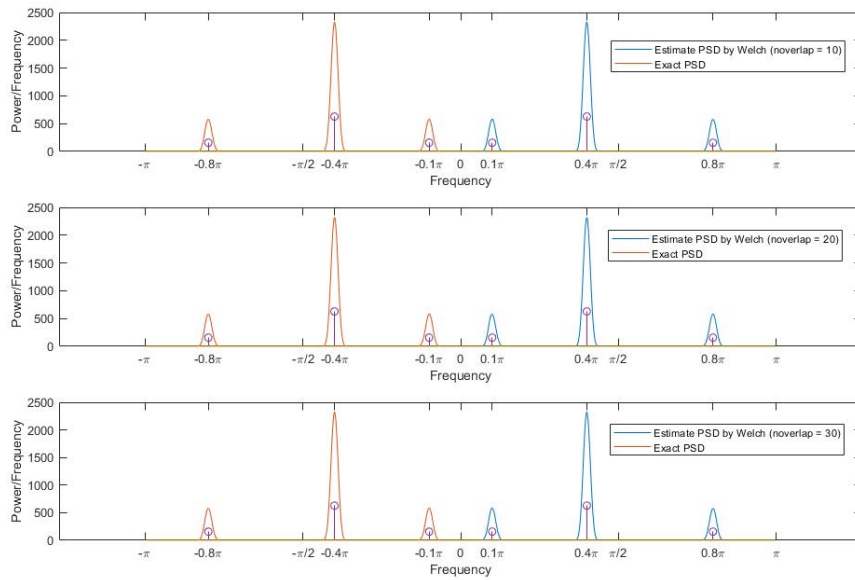
1.2 part b



1.3 part c



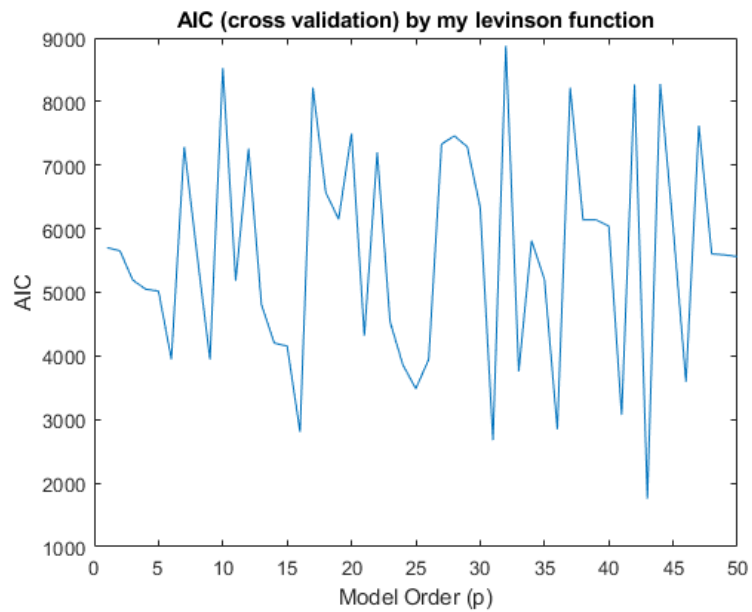
1.4 part d



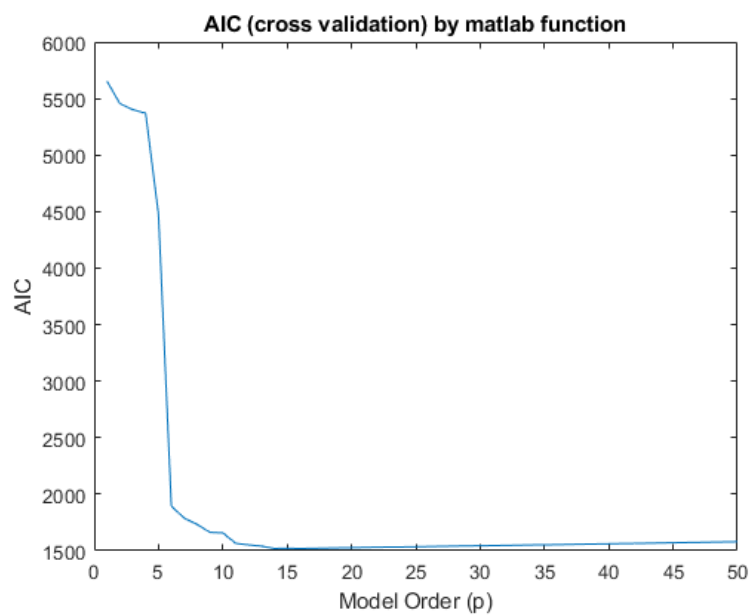
1.5 part e

- I have implemented my levinson function and matlab levinson and these are the results :

- By my function:



- By matlab function:



- optimal order:

Optimal AR coeffs:

-0.1486
-0.0970
0.0584
-0.1814
-0.1533
0.0999
-0.0627
-0.1252
0.1621
0.1041
-0.0832
0.1649
0.1900
-0.0650
0.0728
-0.8610

-coeffs for optimal order:

Optimal AR Order (p) by Levinson-Durbin:

18

Optimal AR Order (p) by AIC:

16

1.6 part f

1.7 part g

Matlab Code

```
1 %% our information
2 samples = 1000;
3
4 alpha1 = 2*pi*rand;
5 alpha2 = 2*pi*rand;
6 alpha3 = 2*pi*rand;
7
8 n = 0:samples-1;
9 x = 10*cos(0.1*pi*n + alpha1) + 20*cos(0.4*pi*n +
    alpha2) + 10*cos(0.8*pi*n + alpha3) + randn(1,
    samples);
10
11
12 %% part a
13
14 f = linspace(-pi, pi, 1000);
15 PSD = ones(size(f));
16 add_delta = @(psd, freq, amp) psd + amp * (f == freq);
17 PSD = add_delta(PSD, 0.1*pi, pi * 50);
18 PSD = add_delta(PSD, -0.1*pi, pi * 50);
19 PSD = add_delta(PSD, 0.4 * pi, pi * 200);
20 PSD = add_delta(PSD, -0.4 * pi, pi * 200);
21 PSD = add_delta(PSD, 0.8 * pi, pi * 50);
22 PSD = add_delta(PSD, -0.8 * pi, pi * 50);
23 figure;
24 plot(f, PSD);
25 title('Power Spectral Density (PSD)');
26 xlabel('Frequency');
27 ylabel('Power / Frequency');
28 hold on;
```

```

29 stem([0.1*pi, -0.1*pi, 0.4 * pi, -0.4 * pi, 0.8 * pi,
      -0.8 * pi], ...
30      [pi * 50, pi * 50, pi * 200, pi * 200, pi * 50, pi
      * 50]);
31 xticks([-pi -0.8*pi -pi/2 -0.4*pi -0.1*pi 0 0.1*pi
      0.4*pi pi/2 0.8*pi pi]);
32 xticklabels({'-\pi', '-0.8\pi', '-\pi/2', '-0.4\pi',
      '-0.1\pi', '0', '0.1\pi', '0.4\pi', '\pi/2', '0.8\
      pi', '\pi'});
33
34
35
36 %% part b
37
38 estimated_corr = zeros(1, samples/2);
39 BT_PSD = zeros(1, 20000);
40 f1 = linspace(-pi, pi, 20000);
41
42 for i=1:samples/2
43     for j = 1:(samples-i)
44         estimated_corr(i) = x(j)*x(j+i) +
45             estimated_corr(i);
46     end
47     estimated_corr(i) = estimated_corr(i) / samples;
48 end
49 for w=1:20000
50     temp = 0;
51     for j=1:(samples/2)
52         temp = estimated_corr(j) * exp(-1i*f1(w)*j) +
53             temp;
54     end

```



```

53     BT_PSD(w) = estimated_corr(1) + 2 * real(temp);
54 end
55
56 figure;
57 subplot(2,1,1);
58 plot(f1,abs(BT_PSD))
59 title('Estimate PSD by BT');
60 xlabel('Frequency');
61 ylabel('Power/Frequency');
62 xticks([-pi -0.8*pi -pi/2 -0.4*pi -0.1*pi 0 0.1*pi
        0.4*pi pi/2 0.8*pi pi]);
63 xticklabels({'-\pi', '-0.8\pi', '-\pi/2', '-0.4\pi',
        '-0.1\pi', '0', '0.1\pi', '0.4\pi', '\pi/2', '0.8\pi', '\pi'});
64
65 subplot(2,1,2);
66 plot(f, PSD);
67 hold on;
68 title('Exact PSD');
69 xlabel('Frequency');
70 ylabel('Power/Frequency');
71 stem([0.1*pi, -0.1*pi, 0.4 * pi, -0.4 * pi, 0.8 * pi,
        -0.8 * pi], ...
72      [pi * 50, pi * 50, pi * 200, pi * 200, pi * 50, pi
        * 50]);
73 xticks([-pi -0.8*pi -pi/2 -0.4*pi -0.1*pi 0 0.1*pi
        0.4*pi pi/2 0.8*pi pi]);
74 xticklabels({'-\pi', '-0.8\pi', '-\pi/2', '-0.4\pi',
        '-0.1\pi', '0', '0.1\pi', '0.4\pi', '\pi/2', '0.8\pi', '\pi'});
75

```

```

76 %% part c
77
78 [pxx, f2] = periodogram(x,[],[],2*pi);
79
80 figure;
81 subplot(2,1,1);
82 plot(f2, pxx); % Adjust frequency range to [-pi, pi]
83 hold on;
84 plot(-f2, pxx)
85 title('Estimate PSD by Periodogram');
86 xlabel('Frequency');
87 ylabel('Power/Frequency');
88 xticks([-pi -0.8*pi -pi/2 -0.4*pi -0.1*pi 0 0.1*pi
          0.4*pi pi/2 0.8*pi pi]);
89 xticklabels({'-\pi', '-0.8\pi', '-\pi/2', '-0.4\pi',
              '-0.1\pi', '0', '0.1\pi', '0.4\pi', '\pi/2', '0.8\pi', '\pi'});
90
91 subplot(2,1,2);
92 plot(f, PSD);
93 hold on;
94 stem([0.1*pi, -0.1*pi, 0.4 * pi, -0.4 * pi, 0.8 * pi,
        -0.8 * pi], ...
95      [pi * 50, pi * 50, pi * 200, pi * 200, pi * 50, pi
        * 50]);
96 title('Exact PSD');
97 xlabel('Frequency');
98 ylabel('Power/Frequency');
99 xticks([-pi -0.8*pi -pi/2 -0.4*pi -0.1*pi 0 0.1*pi
          0.4*pi pi/2 0.8*pi pi]);
100 xticklabels({'-\pi', '-0.8\pi', '-\pi/2', '-0.4\pi',

```

```

    '-0.1\pi', '0', '0.1\pi', '0.4\pi', '\pi/2', '0.8\pi', '\pi'});

101
102
103
104
105
106 %% part d
107
108
109 [pxx_welch_10, f3] = pwelch(x, 100, 10, samples, 2*pi)
    ;
110 [pxx_welch_20, f4] = pwelch(x, 100, 20, samples, 2*pi)
    ;
111 [pxx_welch_30, f5] = pwelch(x, 100, 30, samples, 2*pi)
    ;
112
113 figure;
114 subplot(3,1,1);
115 plot(f3, pxx_welch_10);
116 hold on;
117 plot(-f3, pxx_welch_10)
118 hold on;
119 plot(f, PSD);
120 hold on;
121 stem([0.1*pi, -0.1*pi, 0.4 * pi, -0.4 * pi, 0.8 * pi,
    -0.8 * pi], ...
122      [pi * 50, pi * 50, pi * 200, pi * 200, pi * 50, pi
    * 50]);
123 xlabel('Frequency');
124 ylabel('Power / Frequency');

```

```

125 legend('Estimate PSD by Welch (noverlap = 10)', 'Exact
    PSD')
126 xticks([-pi -0.8*pi -pi/2 -0.4*pi -0.1*pi 0 0.1*pi
    0.4*pi pi/2 0.8*pi pi]);
127 xticklabels({'-\pi', '-0.8\pi', '-\pi/2', '-0.4\pi',
    '-0.1\pi', '0', '0.1\pi', '0.4\pi', '\pi/2', '0.8\
    pi', '\pi'});

128
129 %%%%%%%%%%
130
131 subplot(3,1,2);
132 plot(f4, pxx_welch_20);
133 hold on;
134 plot(-f4, pxx_welch_20)
135 hold on;
136 plot(f, PSD);
137 hold on;
138 stem([0.1*pi, -0.1*pi, 0.4 * pi, -0.4 * pi, 0.8 * pi,
    -0.8 * pi], ...
139     [pi * 50, pi * 50, pi * 200, pi * 200, pi * 50, pi
    * 50]);
140 xlabel('Frequency');
141 ylabel('Power/Frequency');
142 legend('Estimate PSD by Welch (noverlap = 20)', 'Exact
    PSD')
143 xticks([-pi -0.8*pi -pi/2 -0.4*pi -0.1*pi 0 0.1*pi
    0.4*pi pi/2 0.8*pi pi]);
144 xticklabels({'-\pi', '-0.8\pi', '-\pi/2', '-0.4\pi',
    '-0.1\pi', '0', '0.1\pi', '0.4\pi', '\pi/2', '0.8\
    pi', '\pi'});

145

```

```

146 %%%
147
148
149 subplot(3,1,3);
150 plot(f5, pxx_welch_30);
151 hold on;
152 plot(-f5, pxx_welch_30)
153 hold on;
154 plot(f, PSD);
155 hold on;
156 stem([0.1*pi, -0.1*pi, 0.4 * pi, -0.4 * pi, 0.8 * pi,
      -0.8 * pi], ...
157      [pi * 50, pi * 50, pi * 200, pi * 200, pi * 50, pi
      * 50]);
158 xlabel('Frequency');
159 ylabel('Power/Frequency');
160 legend('Estimate PSD by Welch (noverlap = 30)', 'Exact
      PSD')
161 xticks([-pi -0.8*pi -pi/2 -0.4*pi -0.1*pi 0 0.1*pi
      0.4*pi pi/2 0.8*pi pi]);
162 xticklabels({'-\pi', '-0.8\pi', '-\pi/2', '-0.4\pi',
      '-0.1\pi', '0', '0.1\pi', '0.4\pi', '\pi/2', '0.8\
      pi', '\pi'});
163
164
165
166
167
168 %% part e
169
170

```

```

171 R_x = xcorr(x, 'biased');
172
173 % implementing levinson by myself and AIC cross
    valiadtation
174 E = [];
175 K=[];
176 AIC = zeros(50,1);
177 temp = 0;
178 R_xx = R_x(1000:1999);
179 for i = 1:50
180     if i == 1
181         E(i) = R_xx(i);
182     elseif i == 2
183         k(i) = -R_xx(i) / E(i-1);
184         a(i,i) = k(i);
185         E(i) = (1 - k(i)^2) * E(i-1);
186     else
187         for j = 1:i-1
188             temp = temp + a(j,i-1) * R_xx(i-j);
189         end
190         k(i) = -(R_xx(i) + temp) / E(i-1);
191         a(i,i) = k(i);
192         for j = 1:i-1
193             a(j,i) = a(j,i-1) + k(i) * a(i-j,i-1);
194         end
195         E(i) = (1 - k(i)^2) * E(i-1);
196     end
197     AIC(i) = samples * log(E(i)) + 2 * (i);
198     temp = 0;
199 end
200 [~, p_opt_AIC] = min(AIC);

```

```

201 p_opt_LD = 1;
202 disp('Optimal AR Order (p) by Levinson-Durbin:');
203 disp(p_opt_LD);
204 disp('Optimal AR Order (p) by AIC:');
205 disp(p_opt_AIC);
206 figure;
207 plot(1:50, AIC);
208 title('AIC (cross validation)');
209 xlabel('Model Order (p)');
210 ylabel('AIC');
211 disp('Optimal AR coeffs (p) by Levinson-Durbin:');
212 disp(a(:,p_opt_LD));
213 disp('Optimal AR coeffs (p) by AIC:');
214 disp(a(:,p_opt_AIC));
215 % implementing levinson by matlab function and AIC
    cross validation
216
217 e = zeros(51,1);
218 aic = zeros(50,1);
219 [a1, e(1)] = levinson(R_xx, 0); % Order 0 model
220 for p = 1:50
221     [a1, e(p+1)] = levinson(R_xx, p);
222     aic(p) = p*log(e(p+1))+2*p;
223 end
224 [~, p_opt_aic] = min(AIC);
225 p_opt_LD_m = 1;
226 disp('Optimal AR Order (p) by Levinson-Durbin:');
227 disp(a(:,p_opt_LD_m));
228 disp('Optimal AR Order (p) by AIC:');
229 disp(a(:,p_opt_aic));
230 figure;

```

```

231 plot(1:50, AIC);
232 title('AIC (cross validation)');
233 xlabel('Model Order (p)');
234 ylabel('AIC');
235
236
237
238 %% part f
239
240
241 % Maximum MA order to consider
242 max_q = 20;
243
244 % Preallocate arrays for criteria
245 aic = zeros(max_q, 1);
246 bic = zeros(max_q, 1);
247 fpe = zeros(max_q, 1);
248
249 % Loop through different MA orders
250 for q = 1:max_q
251     try
252         % Estimate MA model of order q
253         model = arima('Constant', 0, 'MA', q, '
                Variance', 1);
254         fit = estimate(model, x, 'Display', 'off', '
                EnforceInvertibility', true);
255
256         % Get the log-likelihood, number of parameters
                , and variance of residuals
257         logL = loglikelihood(fit);
258         numParams = q + 1; % q MA parameters + 1

```



```

                variance parameter
259         variance = fit.Variance;
260
261         % Calculate criteria
262         aic(q) = -2 * logL + 2 * numParams;
263         bic(q) = -2 * logL + log(N) * numParams;
264         fpe(q) = variance * (1 + 2 * numParams / N) /
                (1 - 2 * numParams / N);
265     catch ME
266         % If estimation fails, set criteria to Inf
267         aic(q) = Inf;
268         bic(q) = Inf;
269         fpe(q) = Inf;
270         disp(['Order ', num2str(q), ' estimation
                failed: ', ME.message]);
271     end
272 end
273
274 % Find the order with minimum AIC, BIC, and FPE
275 [~, min_aic_order] = min(aic);
276 [~, min_bic_order] = min(bic);
277 [~, min_fpe_order] = min(fpe);
278
279 % Display results
280 disp(['Optimal order by AIC: ', num2str(min_aic_order)
        ]);
281 disp(['Optimal order by BIC: ', num2str(min_bic_order)
        ]);
282 disp(['Optimal order by FPE: ', num2str(min_fpe_order)
        ]);
283

```

```

284 % Plot AIC, BIC, and FPE
285 figure;
286 subplot(2,3,1);
287 plot(1:max_q, aic, 'LineWidth', 2);
288 title('AIC for Different MA Model Orders');
289 xlabel('MA Order');
290 ylabel('AIC');
291 legend('AIC');
292 grid on;
293
294 subplot(2,3,2);
295 plot(1:max_q, bic, 'LineWidth', 2);
296 title('BIC for Different MA Model Orders');
297 xlabel('MA Order');
298 ylabel('BIC');
299 legend('BIC');
300 grid on;
301
302 subplot(2,3,3);
303 plot(1:max_q, fpe, 'LineWidth', 2);
304 title('FPE for Different MA Model Orders');
305 xlabel('MA Order');
306 ylabel('FPE');
307 legend('FPE');
308 grid on;
309
310
311
312
313
314

```

```

315
316
317 %% part g
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333 %% part h

```

2 Question 2

2.1 part a

```

1 %% a
2 ECG_dataset = load('test.mat');
3 fs = 360;
4 s = ECG_dataset.val;
5 t = 0:1/fs:(length(s)-1)/fs;
6 phi1 = 2*pi*rand;
7 phi2 = 2*pi*rand;
8 N1 = 2*cos(100*pi*t+phi1);

```

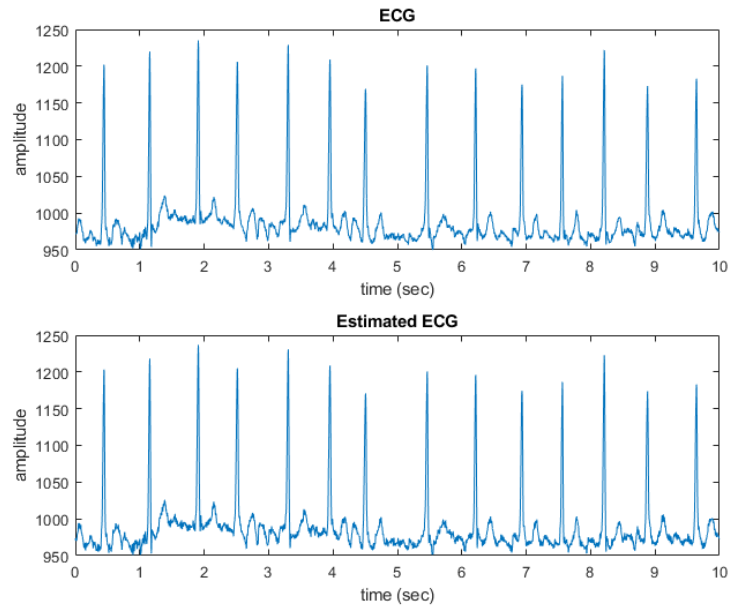
```

9  N2 = 2*cos(100*pi*t+phi2);
10 reference_signal = N1;
11 primary_signal = s + N2;
12 w = zeros(length(s),1);
13 y = zeros(length(s),1);
14 e = zeros(length(s),1);
15 mu = 0.000001;
16 p = y-w;
17 [e,y] = adaptivefilter(w,reference_signal ,
    primary_signal ,mu);

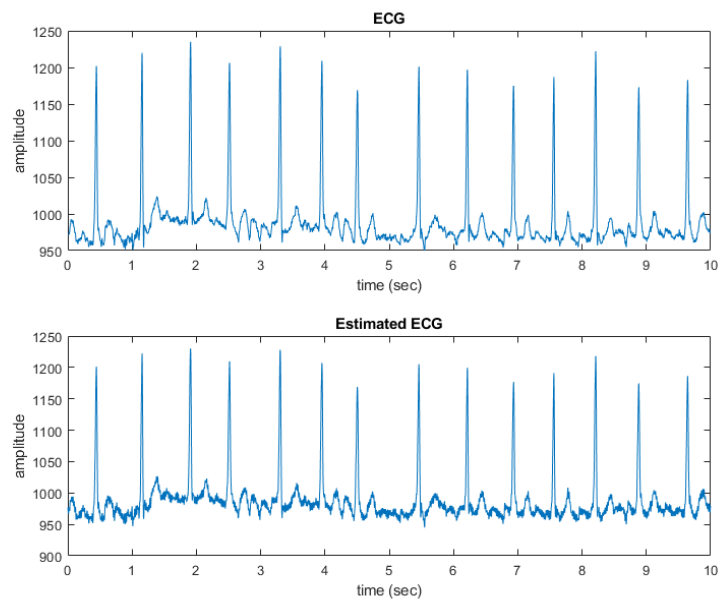
```

2.2 part b

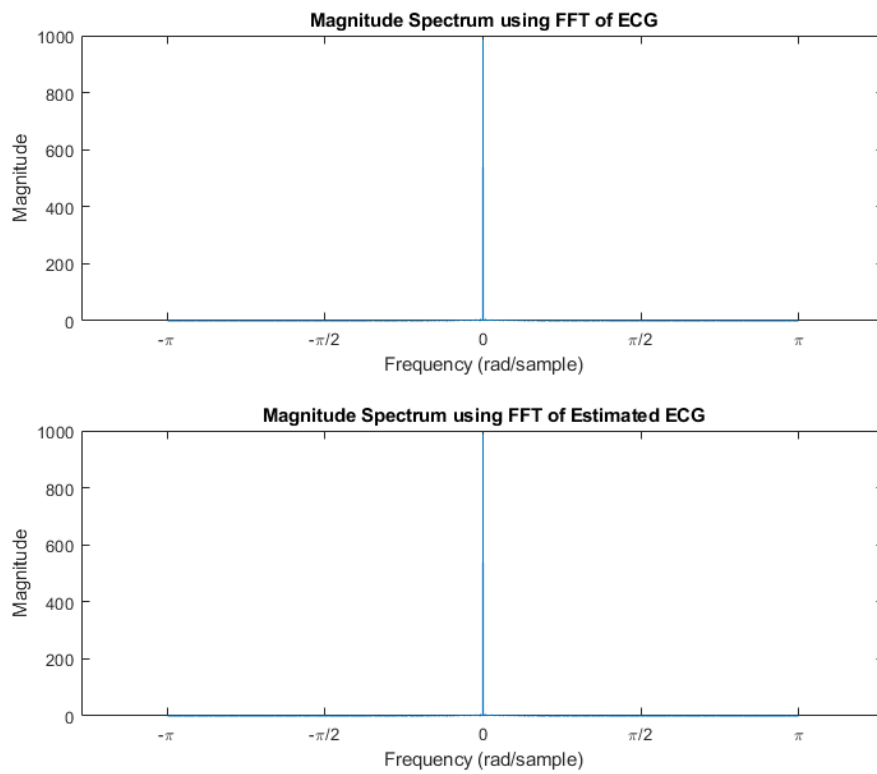
with Amplitude = 5 for noises :



with Amplitude = 20 for noises :



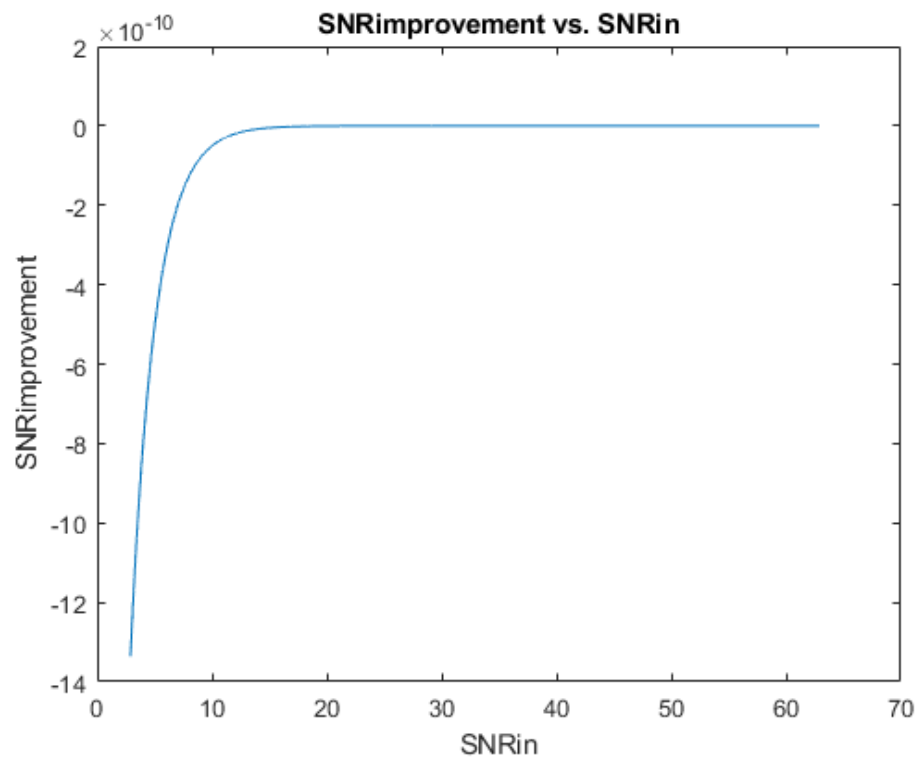
* As we can see still we have a little amount of effect from the noise , which added to the ECG signal, but we have estimated the ECG signal properly. Also we have to say that the effect of noise relate to the amplitude directly.



* As we can see there is almost no difference in spectrum of the ECG and estimated ECG.

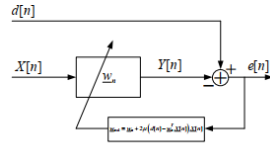
2.3 part c

- For different values of amplitude we have below figure, As we can see after some amplitudes the SNRimprovement will be constant even by increasing SNRin and converge.



2.4 part d

* As we had in slides :



مثال

- مثال ۳- حذف نویز برق شهر
- رفتار یک فیلتر میان‌گذر (notch filter)
- یک ثبت فقط شامل نویز برق شهر
- تفاوت دامنه و فاز دو سینوسی در دو ثبت

$$\begin{cases} d[n] = S[n] + A_1 \cos(\omega_b n + \varphi_1) \\ X[n] = A_1 \cos(\omega_b n + \varphi_1) \end{cases} \Rightarrow \begin{cases} Y[n] = A_1 \cos(\omega_b n + \varphi_1) \\ e[n] = S[n] \end{cases}$$

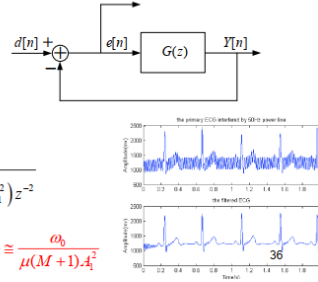
$$w_{n+1} = w_n + 2\mu Z[n] (d[n] - Z[n]^T w_n)$$

$$G(z) = \frac{Y(z)}{E(z)} \cong \frac{\mu(M+1)A_1^2}{2} \frac{1 - \cos \omega_b z^{-1}}{1 - 2\cos \omega_b z^{-1} + z^{-2}} \quad \frac{\mu(M+1)A_1^2}{2} \ll 1$$

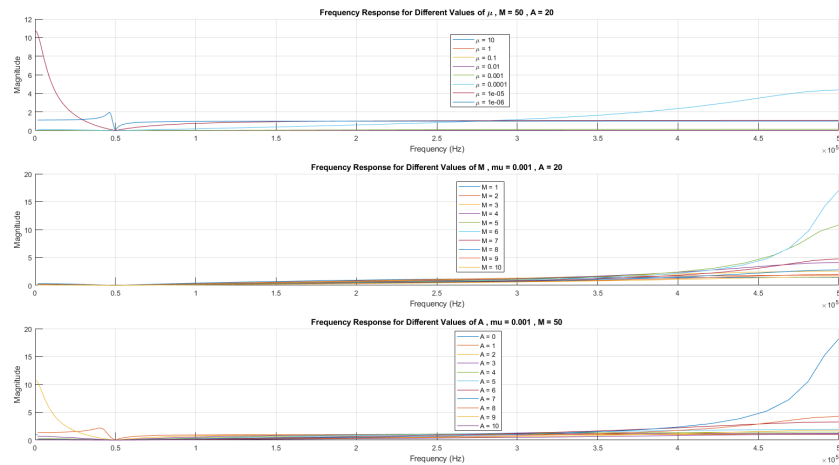
$$H(z) = \frac{E(z)}{D(z)} = \frac{1}{1 + G(z)} \cong \frac{1 - 2\cos \omega_b z^{-1} + z^{-2}}{1 - 2\left(1 - \frac{\mu(M+1)A_1^2}{4}\right)\cos \omega_b z^{-1} + (1 - \mu(M+1)A_1^2)z^{-2}}$$

$$\text{zeros: } 1e^{\pm j\omega_b} \quad \text{poles: } \left(1 - \frac{\mu(M+1)A_1^2}{2}\right)e^{\pm j\omega_b} \quad \Delta\omega \cong \mu(M+1)A_1^2 \quad Q = \frac{\omega_b}{\Delta\omega} \cong \frac{\omega_b}{\mu(M+1)A_1^2}$$

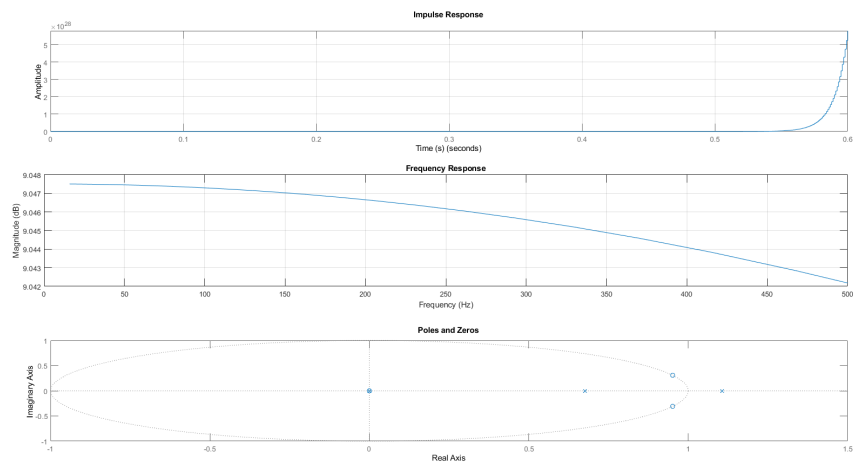
دانشگاه صنعتی شریف- دانشکده برق



- Now for different values of mu,A and M I've plotted frequency responses:

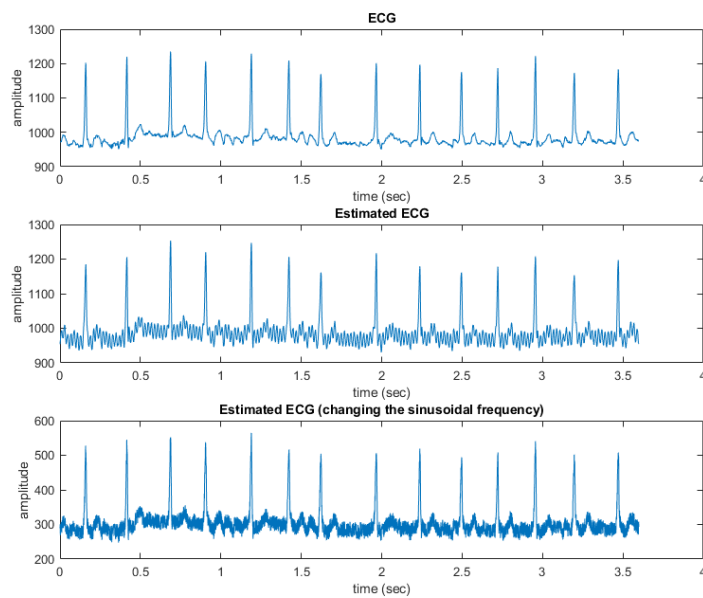


* Now for mu = 0.0001 , A = 5 , M = 100 :



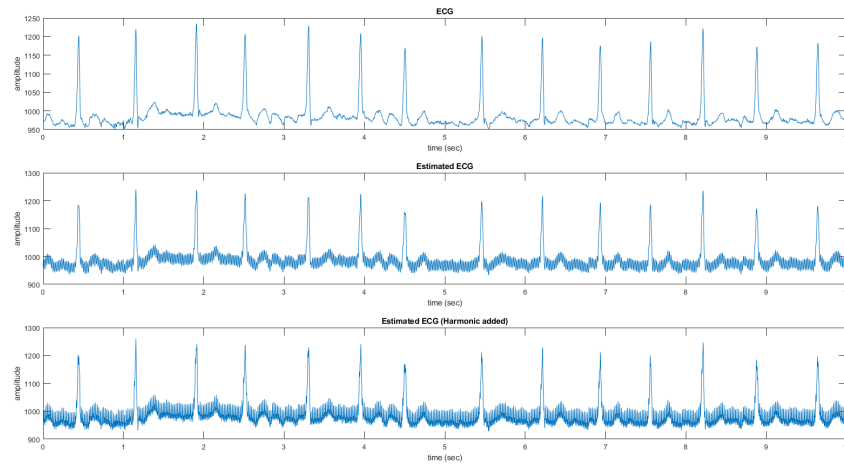
2.5 part e

- For analysis the effect of a little change in sinusoidal frequencies, I have added noise to the frequency and we will have :



* As we can see the output of Adaptive filter would be noisier than previous estimated ECG.

2.6 part f



* As we can see by adding the first harmonic component Adaptive filter can remove less noise from primary signal but still estimation works almost properly.

Matlab Code

```
1 %% a
2 ECG_dataset = load('test.mat');
3 fs = 360;
4 s = ECG_dataset.val;
5 t = 0:1/fs:(length(s)-1)/fs;
6 phi1 = 2*pi*rand;
7 phi2 = 2*pi*rand;
8 N1 = 20*cos(100*pi*t+phi1);
9 N2 = 20*cos(100*pi*t+phi2);
10 reference_signal = N1;
11 primary_signal = s + N2;
12 w = zeros(length(s),1);
13 y = zeros(length(s),1);
14 e = zeros(length(s),1);
```

```

15 mu = 0.000001;
16 p = y-w;
17 [e,y] = adaptivefilter(w,reference_signal ,
    primary_signal ,mu);
18 %% b
19 figure;
20
21 subplot(2,1,1)
22 plot(t,s)
23 title('ECG');
24 xlabel('time (sec)');
25 ylabel('amplitude');
26
27 subplot(2,1,2)
28 plot(t,e)
29 title('Estimated ECG');
30 xlabel('time (sec)');
31 ylabel('amplitude');
32
33 f = (-length(s)/2:length(s)/2-1)*(2*pi/length(s));
34 figure;
35 subplot(2,1,1)
36 plot(f, abs(fftshift(fft(s)))/length(s));
37 title('Magnitude Spectrum using FFT of ECG');
38 xlabel('Frequency (rad/sample)');
39 ylabel('Magnitude');
40 xticks([-pi -pi/2 0 pi/2 pi]);
41 xticklabels({'-\pi', '-\pi/2', '0', '\pi/2', '\pi'})
    ;
42
43 subplot(2,1,2)

```

```

44 plot(f, abs(fftshift(fft(e)))/length(e))
45 title('Magnitude Spectrum using FFT of Estimated ECG')
    ;
46 xlabel('Frequency (rad/sample)');
47 ylabel('Magnitude');
48 xticks([-pi -pi/2 0 pi/2 pi]);
49 xticklabels({'-\pi', '-\pi/2', '0', '\pi/2', '\pi'})
    ;
50
51 %% c
52
53 for i = 0:1000
54     N1 = i*cos(100*pi*t+phi1);
55     N2 = i*cos(100*pi*t+phi2);
56     reference_signal = N1;
57     primary_signal = s + N2;
58     w = zeros(length(s),1);
59     y = zeros(length(s),1);
60     e = zeros(length(s),1);
61     mu = 0.000001;
62     [e,y] = adaptivefilter(w,reference_signal,
        primary_signal,mu);
63     SNRin(i+1) = 10*log10(norm(s)^2/norm(N2)^2);
64     SNRout(i+1) = 10*log10(norm(s)^2/(norm(e-s)^2));
65     SNRimprovement(i+1) = SNRout(i+1) - SNRin(i+1);
66 end
67
68 figure;
69 plot(SNRin,SNRimprovement)
70 title('SNRimprovement vs. SNRin');
71 xlabel('SNRin');

```

```

72 ylabel('SNRimprovement');
73
74
75 %% part d
76
77
78 f = 50;
79 z = tf('z', 1/fs);
80 figure;
81 mu_values = [10 1 0.1 0.01 0.001 0.0001 0.00001
               0.000001];
82 M = 50;
83 A = 20;
84 for i = 1:length(mu_values)
85     mu = mu_values(i);
86     H = (1 - 2*cos(2*pi*f/fs)*z^(-1)+z^(-2)) / ...
87         (1 - 2*(1 - (mu*(M+1)*A^2)/4)*cos(2*pi*f/fs)*z
            ^(-1) + (1 - mu*(M+1)*A^2)*z^(-2));
88     subplot(3, 1, 1);
89     hold on;
90     [mag, phase, w] = bode(H);
91     plot(w*fs/(2*pi), mag(:));
92 end
93 title('Frequency Response for Different Values of \mu'
       ');
94 xlabel('Frequency (Hz)');
95 ylabel('Magnitude');
96 legend(arrayfun(@(x) ['\mu = ', num2str(x)], mu_values,
                  'UniformOutput', false));
97 grid on;
98

```

```

99 % Frequency response for different values of M
100 mu = 0.001;
101 M_values = 1:10;
102 A = 20;
103
104 for M = M_values
105     H = (1 - 2*cos(2*pi*f/fs)*z^(-1)+z^(-2)) / ...
106         (1 - 2*(1 - (mu*(M+1)*A^2)/4)*cos(2*pi*f/fs)*z
107             ^(-1) + (1 - mu*(M+1)*A^2)*z^(-2));
108     subplot(3, 1, 2);
109     hold on;
110     [mag, phase, w] = bode(H);
111     plot(w*fs/(2*pi), mag(:));
112 end
113 title('Frequency Response for Different Values of M');
114 xlabel('Frequency (Hz)');
115 ylabel('Magnitude');
116 legend(arrayfun(@(x) ['M = ' num2str(x)], M_values, '
117     UniformOutput', false));
118 grid on;
119 mu = 0.001;
120 M = 50;
121 A_values = 0:10;
122 for A = A_values
123     H = (1 - 2*cos(2*pi*f/fs)*z^(-1)+z^(-2)) / ...
124         (1 - 2*(1 - (mu*(M+1)*A^2)/4)*cos(2*pi*f/fs)*z
125             ^(-1) + (1 - mu*(M+1)*A^2)*z^(-2));
126     subplot(3, 1, 3);
127     hold on;
128     [mag, phase, w] = bode(H);
129     plot(w*fs/(2*pi), mag(:));

```

```

127 end
128 title('Frequency Response for Different Values of A');
129 xlabel('Frequency (Hz)');
130 ylabel('Magnitude');
131 legend(arrayfun(@(x) ['A = ' num2str(x)], A_values, '
    UniformOutput', false));
132 grid on;
133 hold off;
134
135 %%%
136
137 mu = 0.0001;
138 A = 5;
139 M = 100;
140 figure;
141 subplot(3, 1, 1);
142 impulse(H);
143 title('Impulse Response');
144 xlabel('Time (s)');
145 ylabel('Amplitude');
146 grid on;
147 subplot(3, 1, 2);
148 [mag, phase, w] = bode(H, {0, pi});
149 mag = squeeze(mag);
150 w = squeeze(w);
151 plot(w*fs/(2*pi), 20*log10(mag));
152 title('Frequency Response');
153 xlabel('Frequency (Hz)');
154 ylabel('Magnitude (dB)');
155 grid on;
156 subplot(3, 1, 3);

```

```

157 pzmap(H);
158 title('Poles and Zeros');
159
160
161 %% part e
162
163
164
165
166
167 %% part f
168 phi1 = 2*pi*rand;
169 phi2 = 2*pi*rand;
170 phi3 = 2*pi*rand;
171 N1 = 20*cos(100*pi*t+phi1);
172 N2 = 20*cos(100*pi*t+phi2);
173 first_harmonic = 20*cos(2*100*pi*t+phi3);
174 reference_signal = N1+first_harmonic;
175 primary_signal = s + N2+first_harmonic;
176 w = zeros(length(s),1);
177 y = zeros(length(s),1);
178 e = zeros(length(s),1);
179 mu = 0.000001;
180 [eh,yh] = adaptivefilter(w,reference_signal,
    primary_signal,mu);
181 [ewh,ywh] = adaptivefilter(w,reference_signal-
    first_harmonic,primary_signal-first_harmonic,mu);
182 figure;
183
184 subplot(3,1,1)
185 plot(t,s)

```



```

186 title('ECG');
187 xlabel('time (sec)');
188 ylabel('amplitude');
189
190 subplot(3,1,2)
191 plot(t,ewh)
192 title('Estimated ECG');
193 xlabel('time (sec)');
194 ylabel('amplitude');
195
196 subplot(3,1,3)
197 plot(t,eh)
198 title('Estimated ECG (Harmonic added)');
199 xlabel('time (sec)');
200 ylabel('amplitude');
201
202
203
204 %% part g
205
206
207 phi1 = 2*pi*rand;
208 N2 = 20*cos(100*pi*t+phi2);
209 primary_signal = s + N2;
210 w = zeros(length(s),1);
211 mu = 0.000001;
212 for i = 1:15
213     [eal(i), yal(i)] = ALEfilter(w, primary_signal,
214                                 primary_signal, mu, i);
215     SNRinal(i) = 10*log10(norm(s)^2/norm(N2)^2);
216     SNRoutal(i) = 10*log10(norm(s)^2/(norm(e(i)-s)^2))

```

```

        ;
216     SNRimprovemental(i) = SNRoutal(i) - SNRinal(i);
217 end
218 figure;
219 plot(SNRin, SNRimprovement)
220 title('SNRimprovement');
221 xlabel('sample');
222 ylabel('SNRimprovement');
223 %% Adaptive filter function :
224
225 function [e,y] = adaptivefilter(w,reference_signal ,
    primary_signal ,mu)
226     for i=1:length(reference_signal)
227         y = reference_signal*w;
228         e = primary_signal - y;
229         w = w + 2*mu*e*reference_signal';
230     end
231 end
232
233
234 %% Adaptive Line Enhancer (ALE)
235
236 function [e, y] = ALEfilter(w, reference_signal ,
    primary_signal , mu, delay)
237     % Initialize variables
238     N = length(primary_signal);
239     y = zeros(1, N); % Filter output
240     e = zeros(1, N); % Error signal
241
242     % Apply delay to the primary signal
243     delayed_primary_signal = [zeros(1, delay),

```

```

primary_signal(1:end-delay)];
244
245 % Adaptive filtering
246 for i = 1:N
247     % Ensure the reference signal is properly
        indexed to avoid overflow
248     if i > delay
249         ref_signal_segment = reference_signal(i-
            delay:i-1)'; % Delayed segment for
                filter input
250         primary_segment = delayed_primary_signal(i
            -delay:i-1); % Delayed primary segment
251         y(i) = ref_signal_segment * w; % Filter
                output
252         e(i) = primary_signal(i) - y(i); % Error
                signal
253         w = w + 2 * mu * e(i) * ref_signal_segment
            '; % Update filter coefficients
254     end
255 end
256 end

```

3 Question 3

3.1 part a

Article: "Model-based Prediction of Heart Rate Variability"

Summary: This study focuses on the use of parametric models for predicting heart rate variability (HRV) in clinical settings. The authors utilize autoregressive models to forecast HRV based on historical data, helping in the early detection of cardiac conditions. The approach allows for real-time monitoring and prediction of heart anomalies, providing valuable insights for preventive healthcare. By lever-

aging parametric modeling, the system adapts to individual patient data, enhancing the accuracy and reliability of predictions.

3.2 part b

Article: "Super-resolution spectral estimation in short-time

non-contact vital sign measurement" Summary: This study applies non-parametric spectral estimation methods to non-contact vital sign measurement using Doppler radar. Techniques such as short-time Fourier transform are employed to analyze the spectral content of radar signals reflected from the body. This allows for accurate extraction of vital signs like heart rate and respiratory rate in real-time, proving beneficial for remote patient monitoring and emergency medical applications.

3.3 part c

Article: "Parametric Spectral Estimation for Sleep Apnea Detection"

Summary: The research discusses the use of parametric spectral estimation techniques, specifically autoregressive (AR) modeling, to detect sleep apnea events from respiratory signals. By estimating the power spectral density of the respiratory signal, the method identifies characteristic patterns associated with apnea episodes. The parametric approach allows for high-resolution spectral analysis, making it possible to detect subtle changes in the signal that are indicative of apnea, thus aiding in the accurate diagnosis and monitoring of sleep disorders.

3.4 part d

Article: "Adaptive Filtering by Non-Invasive Vital Signals Monitoring and

Diseases Diagnosis" Summary: The paper examines the use of adaptive filters, particularly LMS and RLS algorithms, in processing ECG and PPG signals. These filters dynamically adjust to varying noise conditions, effectively removing artifacts and enhancing signal quality. This facilitates accurate measurement of vital parameters such as heart rate and oxygen saturation, improving the reliability of non-invasive monitoring systems in clinical and home settings.