باسمه تعالی

گزارش تمرین کامپیوتری سری ۳

پردازش علائم بیولوژیک

دانشکده مهندسی برق

استاد: محمدباقر شمس اللهی

گردآورنده: رادین خیّام

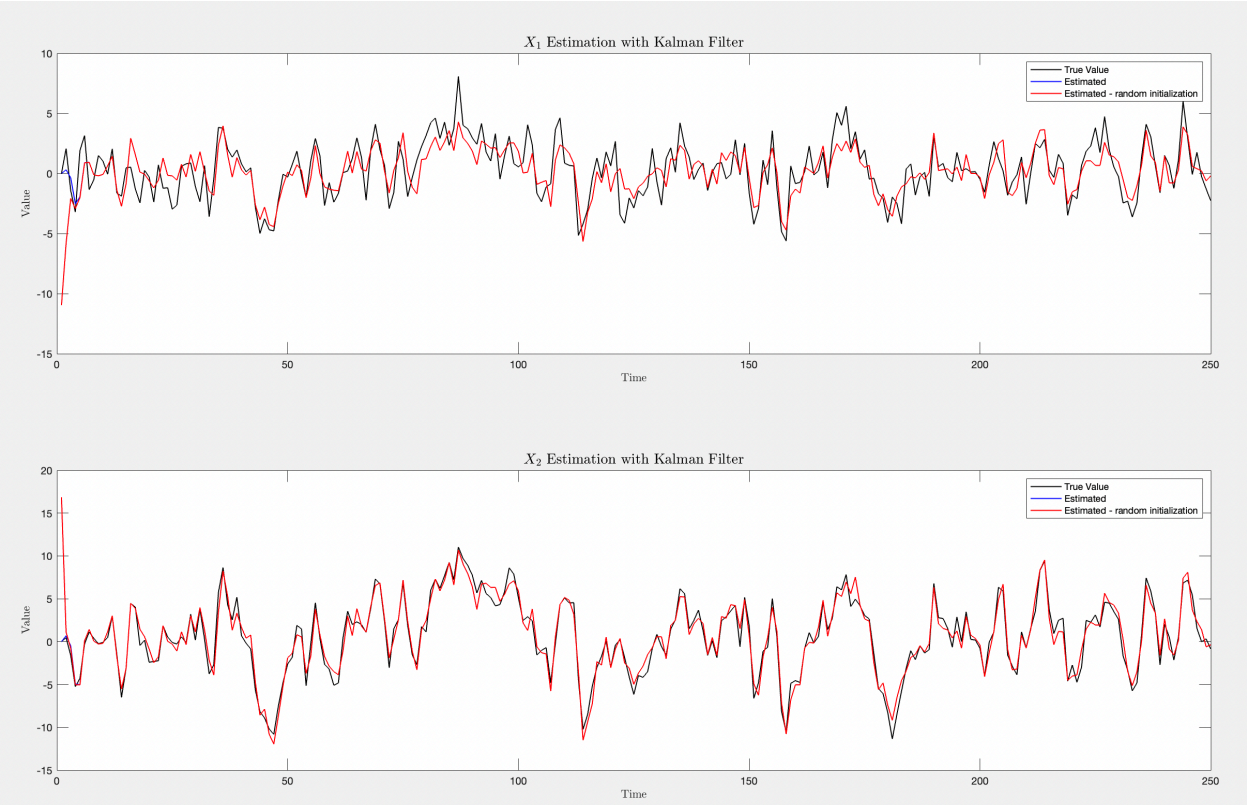
سوال ۱-

الف)

مقادیر اولیه درنظرگرفته شده:

```
sigma_v = 1;
sigma_1 = 2;
sigma_2 = 2;
a = 0.5;
b = 0.5;

N = 250;  % Number of samples
```

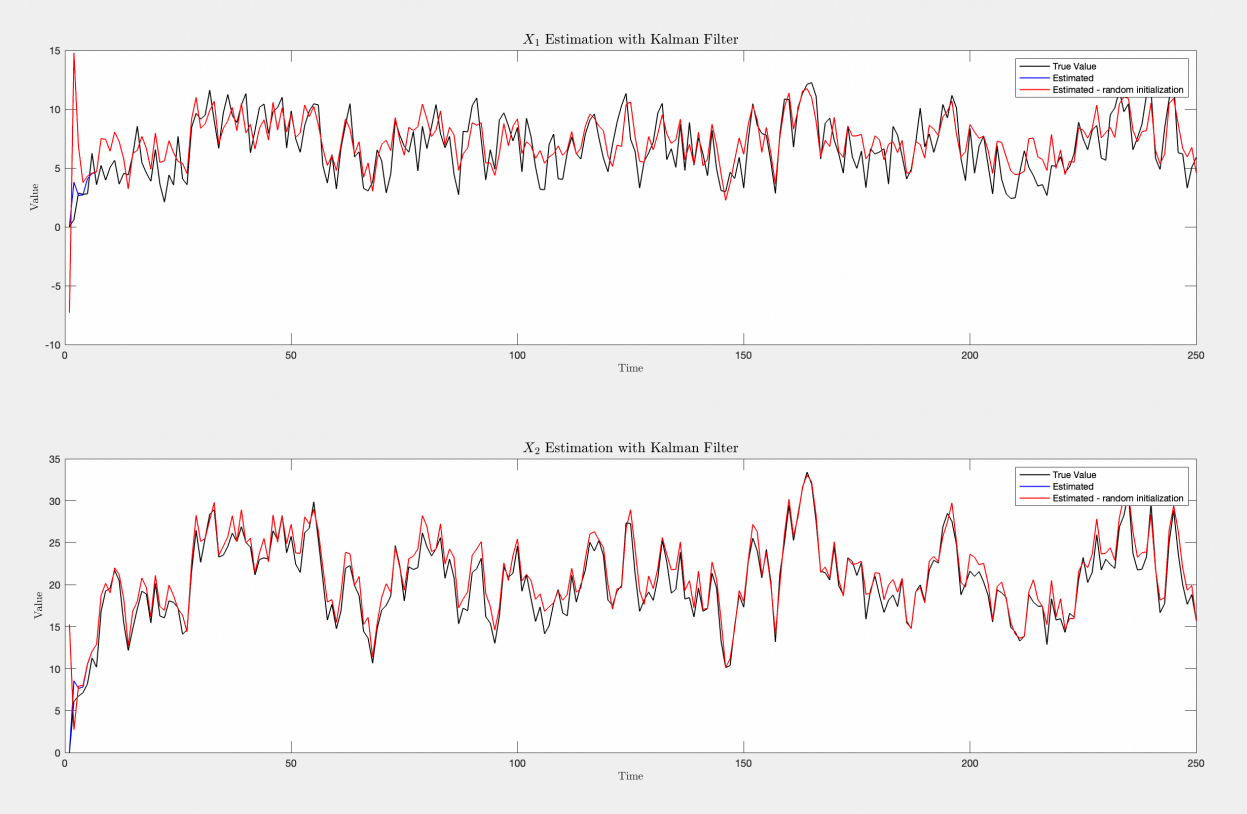



```
Pearson Correlation Coeficient for X_1 = 0.79
Pearson Correlation Coeficient for X_2 = 0.97618
Pearson Correlation Coeficient for X_1 with random initialization = 0.71237
Pearson Correlation Coeficient for X_2 with random initialization = 0.94628
```

ب)

به جای استفاده از توزیع گوسی برای تولید نویزها از توزیع یکنواخت استفاده کردیم.

```
V = sqrt(12) * sigma_v * rand(1, N); % generating noise with uniform distribution and variance sigma_v
U_1 = sqrt(12) * sigma_1 * rand(1, N);
U_2 = sqrt(12) * sigma_2 * rand(1, N);
```



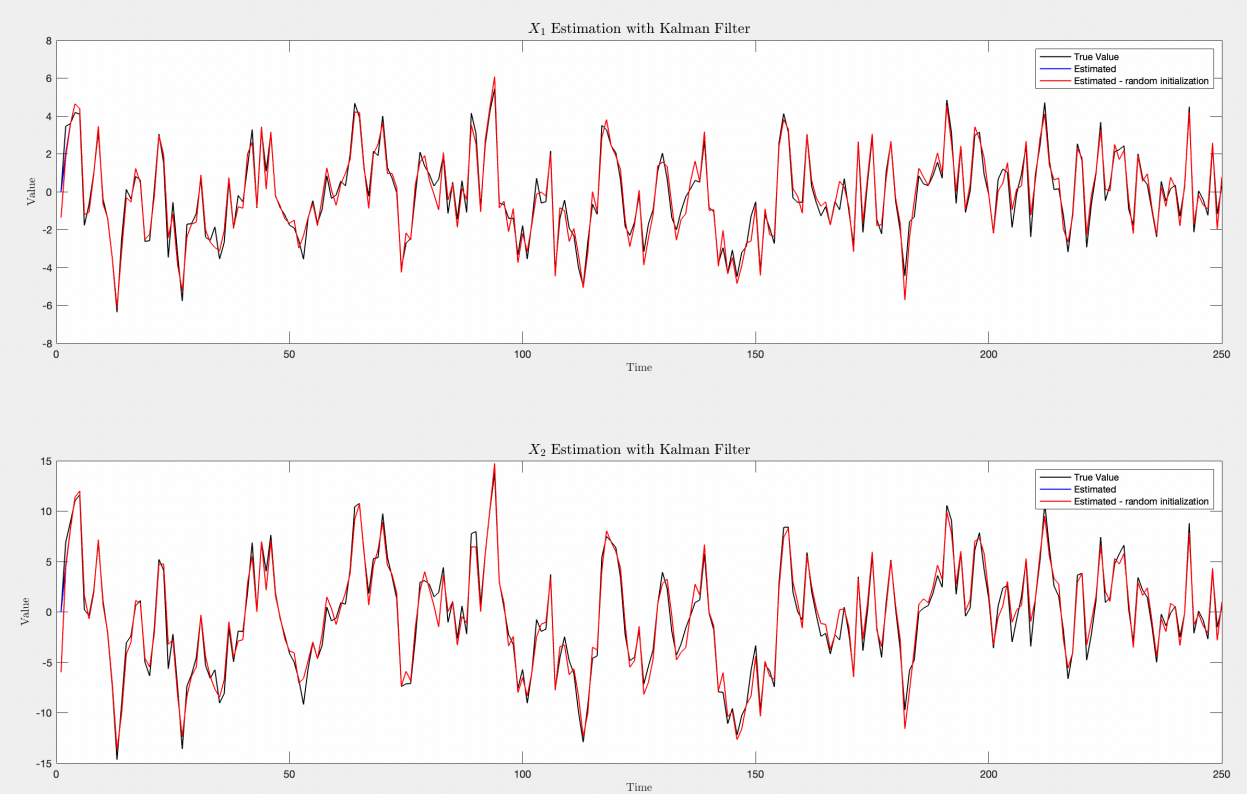

```
Pearson Correlation Coeficient for X_1 = 0.77988
Pearson Correlation Coeficient for X_2 = 0.97882
Pearson Correlation Coeficient for X_1 with random initialization = 0.68609
Pearson Correlation Coeficient for X_2 with random initialization = 0.95948
```

نتایج خیلی متفاوت نشده است.

پ)

نویز U2 را ضریبی از نویز U1 در نظر گرفتیم (به صورتی که واریانسش سیگما ۲ بشود)

```
V = sigma_v * randn(1, N);
U_1 = sigma_1 * randn(1, N);
U_2 = U_1 * sigma_2/sigma_1; % make correlation between U_1 and U_2
```




```
Pearson Correlation Coeficient for X_1 = 0.97524
Pearson Correlation Coeficient for X_2 = 0.98372
Pearson Correlation Coeficient for X_1 with random initialization = 0.9742
Pearson Correlation Coeficient for X_2 with random initialization = 0.98099
```

نتایج بهبود پیدا کرده است. در فیلتر کالمن نیازی نبود که حتما نویزهای سیستم نسبت بهم مستقل باشند بلکه لازم بود که نویز مشاهدات و نویز سیستم از هم مستقل باشند.

ت)

نتایج با هر دو حالت مقادیر اولیه مطابق با درس و با حالت اولیه رندوم در قسمت‌های الف، ب و پ گزارش شده است.

ث)

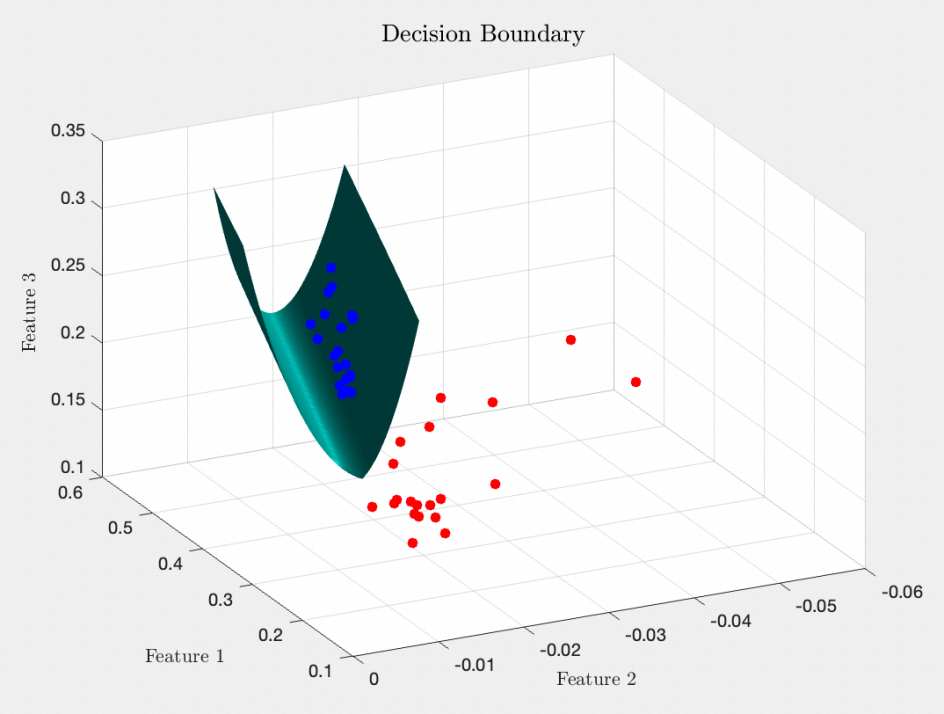
معیار ارزیابی ضریب همبستگی پیرسون در قسمت‌های الف، ب و پ گزارش شده است.

سوال ۲-

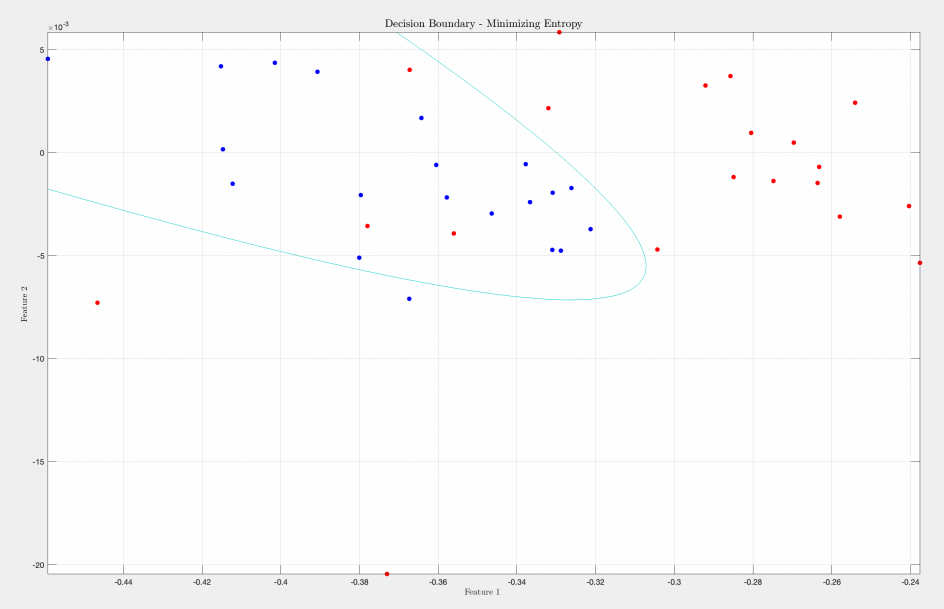
الف)

ویژگی‌ها استخراج شدند.

ب)

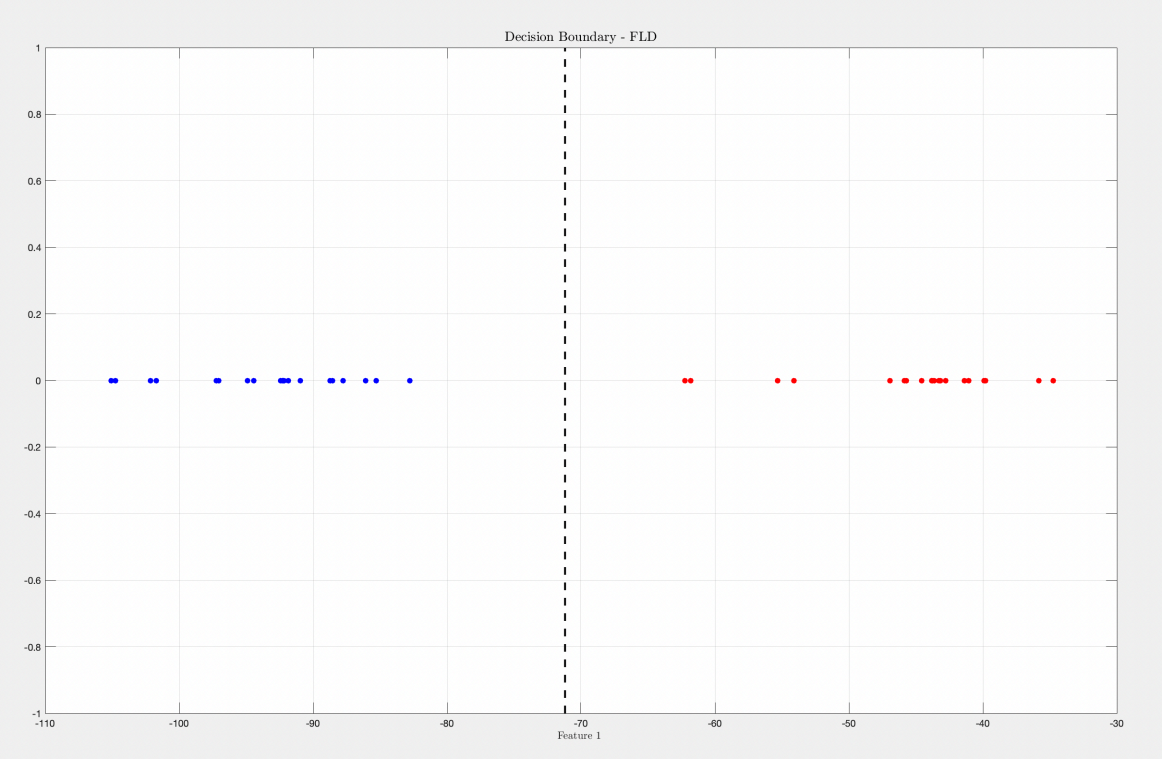


پ)



جدایی پذیری کاهش پیدا کرده است اما اگر به آن مفهومی که در درس مطرح شد (یعنی اینکه اگر میانگین دوکلاس روی هم نیفتد یعنی جدایی پذیرند) بله جدایی پذیری حفظ شده است.

ت)



بله جدایی پذیری کاملا حفظ شده است.

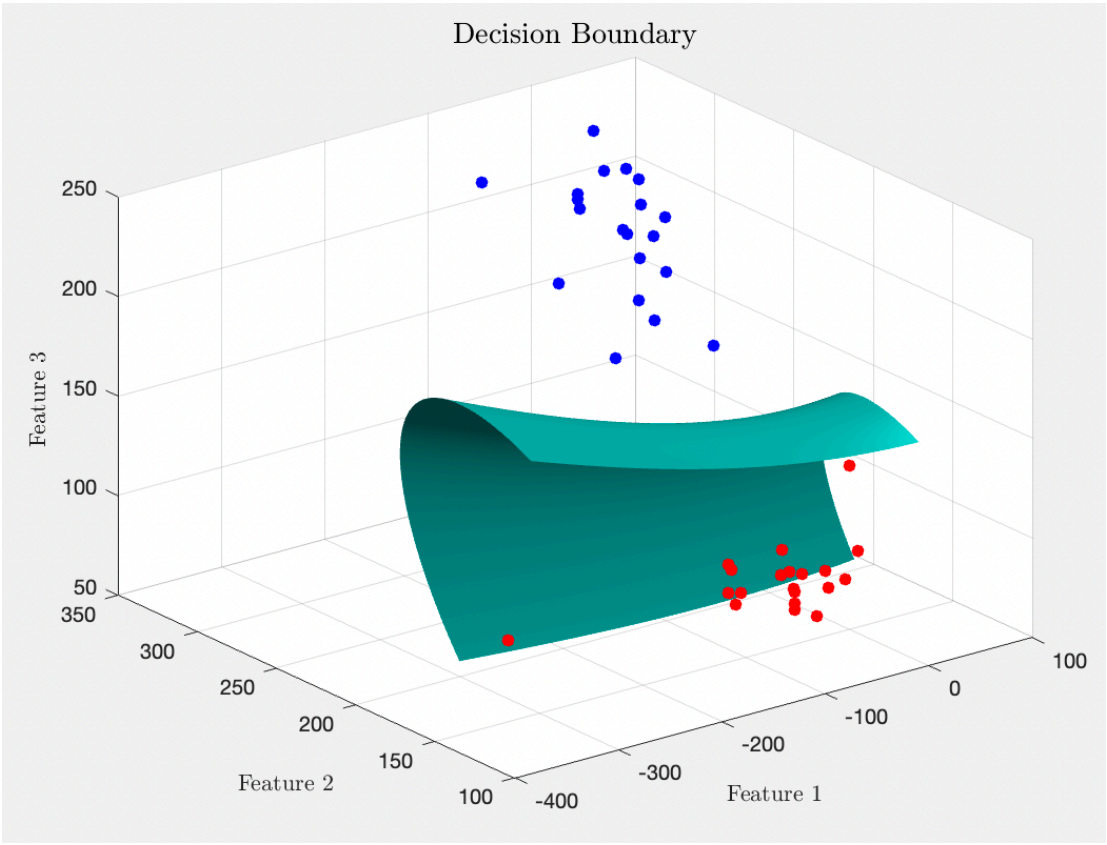
ث)

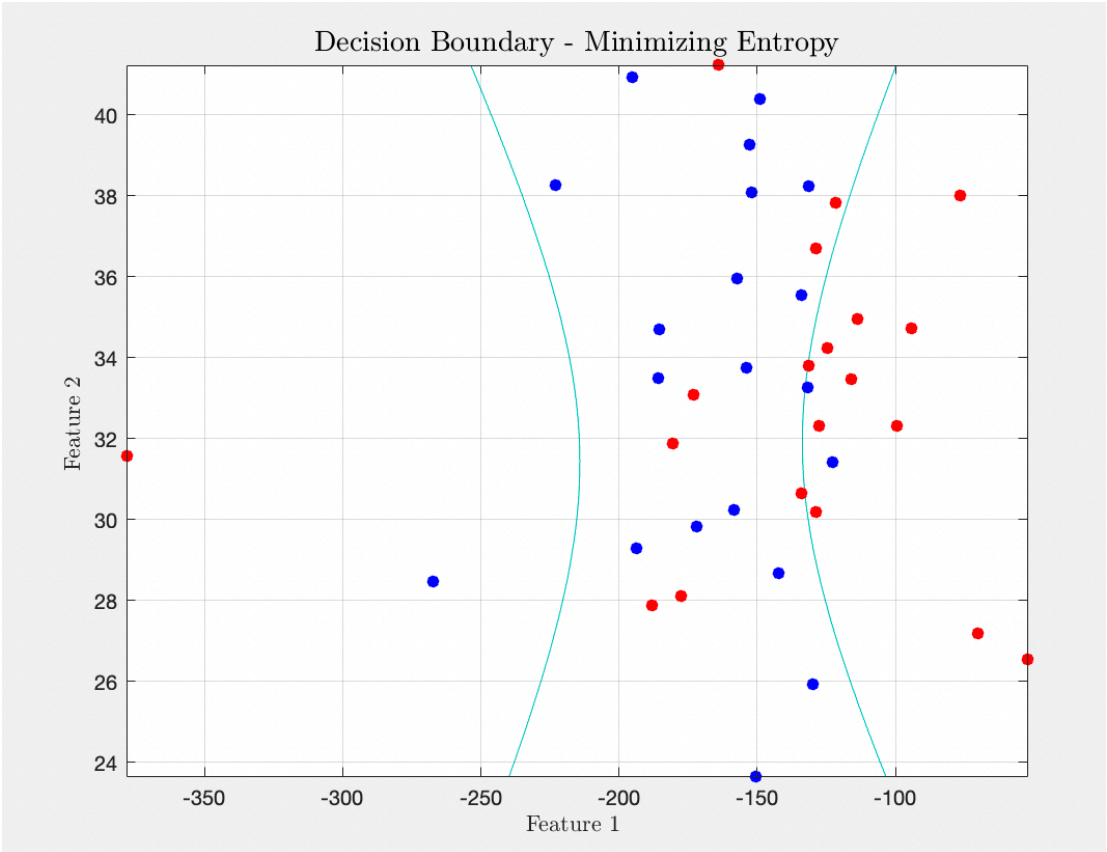
نتایج بدین صورت بدست آمده است.

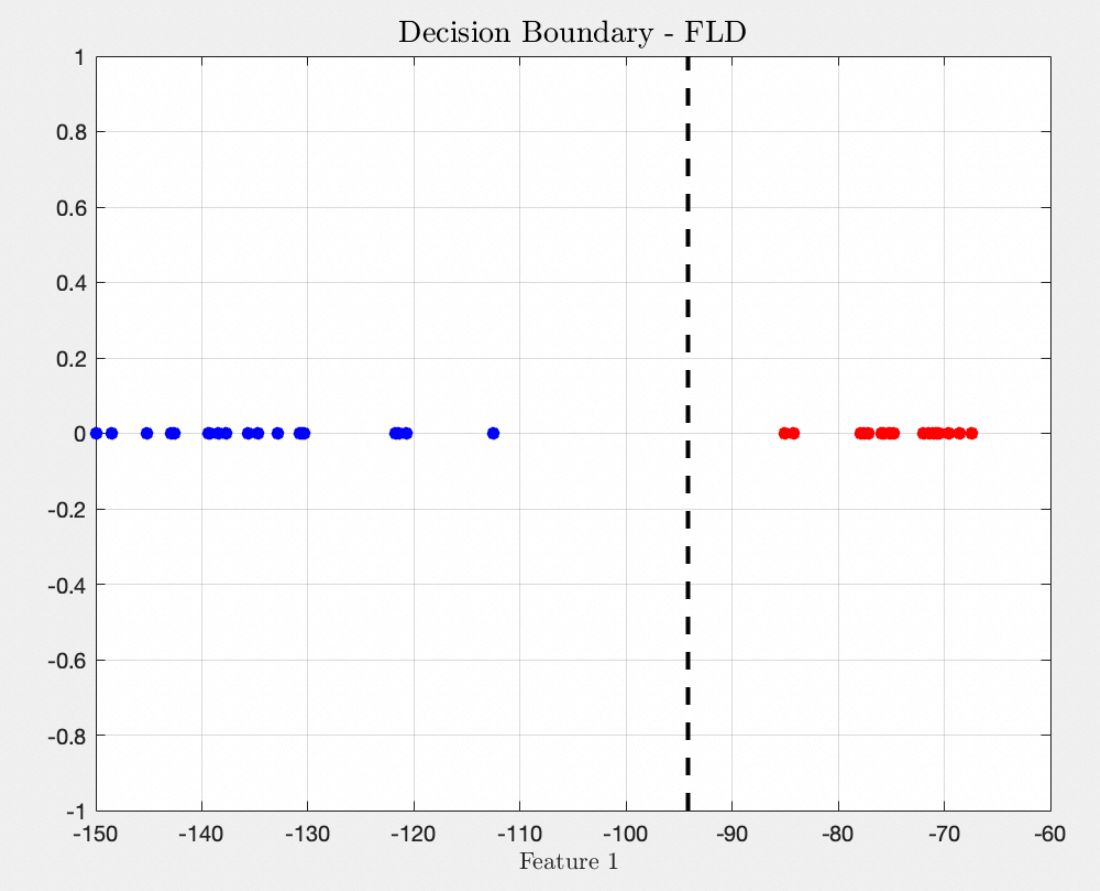
```
Part b Results:
Accuracy of Mahalanobis Distance = 0.975
Accuracy of Euclidean Distance = 0.975
Accuracy of Bayes Classifier = 0.975
============================================================
Part c Results:
Accuracy of Mahalanobis Distance = 0.875
Accuracy of Euclidean Distance = 0.825
Accuracy of Bayes Classifier = 0.925
============================================================
Part d Results:
Accuracy of Mahalanobis Distance = 0.975
Accuracy of Euclidean Distance = 0.975
Accuracy of Bayes Classifier = 0.975
============================================================
```

ج)

ویژگی‌های جدیدی که بدست آوریم، ویژگی‌های آماری میانگین سیگنال، انحراف معیار آن و Mean Absolute Deviation (MAD) بوده است.





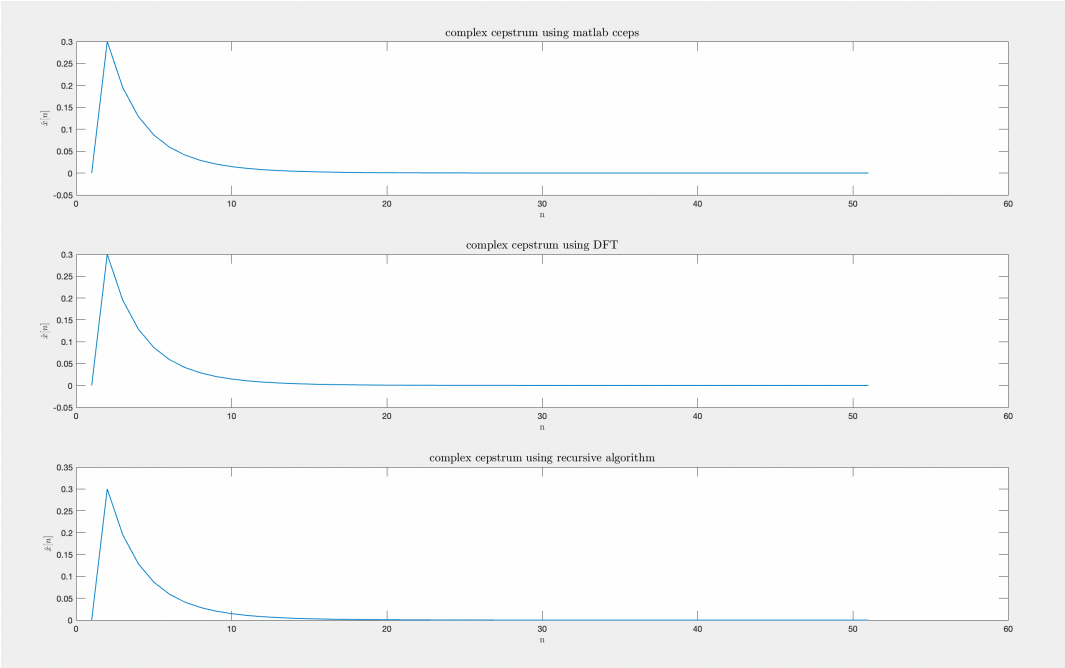



```
Part b Results:
Accuracy of Mahalanobis Distance = 1
Accuracy of Euclidean Distance = 1
Accuracy of Bayes Classifier = 1
============================================================
Part c Results:
Accuracy of Mahalanobis Distance = 0.575
Accuracy of Euclidean Distance = 0.675
Accuracy of Bayes Classifier = 0.7
============================================================
Part d Results:
Accuracy of Mahalanobis Distance = 0.975
Accuracy of Euclidean Distance = 1
Accuracy of Bayes Classifier = 0.975
============================================================
```

می‌بینیم که نتایج در حالت استفاده از سه ویژگی و همچنین حالت تک ویژگی کمی بهبود پیدا کرده است، اما در حالت کاهش بعد به دو ویژگی نتایج بد تر شده است.

سوال ۳-

الف)



ب)

ازهمان دیتاست موجود برای سوال ۲ استفاده کردیم.

سوال ۴-

فیلتر کالمن)

Driving fatigue detection based on brain source activity and ARMA model, Fahimeh Nadalizadeh, Mehdi Rajabioun, Amirreza Feyzi, Medical & Biological Engineering & Computing (2024) 62:1017–1030 , https://doi.org/10.1007/s11517-023-02983-z

مقاله‌ای که مورد بررسی قرارگرفت، روشی نوین برای تشخیص خستگی رانندگان با استفاده از سیگنال‌های الکتروانسفالوگرافی (EEG) ارائه داده است. این روش شامل استخراج فعالیت‌های مغزی از طریق روش‌های مکان‌یابی منبع و برآورد دینامیکی فعالیت منابع با استفاده از یک فیلتر دوگانه کالمن است. فیلتر کالمن دوگانه برای برآورد فعالیت منبع و روابط بین منابع به کار رفته و از این اطلاعات برای پیش‌بینی خستگی راننده استفاده می‌شود. مدل‌سازی زمانی فعالیت‌های مغزی و سیگنال‌های EEG نیز با استفاده از مدل‌های ARMA انجام شده است و از ویژگی‌های این مدل‌سازی برای تشخیص خستگی استفاده می‌کند. در نهایت، از روش‌های ترکیبی برای بهبود دقت تشخیص استفاده شده و نتایج نشان دهنده موفقیت این روش در مقایسه با روش‌های دیگر است.

طبقه بندی)

Analysis of ECG-based arrhythmia detection system using machine learning, Shikha Dhyani, Adesh Kumar, Sushabhan Choudhury, 2023, https://doi.org/10.1016/j.mex.2023.102195

مقاله مورد بررسی، به تجزیه و تحلیل سیستم تشخیص آریتمی قلبی با استفاده از سیگنال‌های الکتروکاردیوگرام (ECG) و روش‌های یادگیری ماشینی می‌پردازد. در این مطالعه، از تبدیل ویولت سه‌بعدی (3D Discrete Wavelet Transform) برای پیش‌پردازش و استخراج ویژگی‌های سیگنال ECG و همچنین استفاده از طبقه بند ماشین بردار پشتیبان (SVM) برای طبقه‌بندی ضربان‌های قلبی استفاده شده است. نتایج مقاله نشان می‌دهد که این روش‌ها قادر به تشخیص دقیق آریتمی‌های مختلف با دقت بالایی هستند. دقت کلی مدل SVM در طبقه‌بندی این داده‌ها بسیار بالا بوده و به بیش از ٪99 رسیده است. این نشان‌دهنده کارایی بالای ترکیب تبدیل موجک و SVM در تشخیص و طبقه‌بندی آریتمی‌ها می‌باشد. از این رو، این مدل می‌تواند به عنوان یک ابزار موثر در تشخیص زودهنگام و دقیق آریتمی‌های قلبی در بیماران مورد استفاده قرارگیرد.

آنالیز کپستروم)

Cepstral Analysis-Based Artifact Detection, Recognition, and Removal in Prefrontal EEG. Issa, M. F., Tuboly, G., Kozmann, G., & Juhasz, Z. (2023). *IEEE Xplore*. DOI: 10.1109/MSR-2019-0016.

متاسفانه مقاله‌ای که از سال ۲۰۲۳ به بعد باشه و به صورت رایگان هم در دسترس باشه پیدا نکردم.

مدل مخفی مارکوف)

N. J. Trujillo-Barreto, D. A. Galvez, A. Astudillo and W. El-Deredy, "Explicit Modeling of Brain State Duration Using Hidden Semi Markov Models in EEG Data," in *IEEE Access*, vol. 12, pp. 12335-12355, 2024, doi: 10.1109/ACCESS.2024.3354711.

در این مقاله، استفاده از مدل‌های نیمه‌مارکوف مخفی (HSMM) برای تجزیه و تحلیل داده‌های الکتروانسفالوگرافی (EEG) مورد بررسی قرار گرفته است. نویسندگان بر محدودیت‌های مدل‌های مارکوف مخفی (HMM) که زمان‌های ماندگاری حالات را با توزیع هندسی مدل‌سازی می‌کنند تاکید و پیشنهاد داده‌اند که HSMM با امکان مدل‌سازی دقیق‌تر مدت زمان حالات مغزی می‌تواند این نقص را رفع کند. در این مطالعه، با استفاده از روش‌های بیزین و برآورد بیز متغیر، تعداد حالات مغزی و توزیع‌های مدت زمان حالات تعیین و مدل‌ها بر روی داده‌های شبیه‌سازی شده و واقعی ارزیابی شده‌اند. نتایج نشان می‌دهند که HSMM در مقایسه با HMM عملکرد بهتری دارد و مدل‌سازی دقیق مدت زمان حالات برای دستیابی به پیش‌بینی‌های معتبر ضروری است.

کد:

```
%% CHW3 - Radin Khayyam - 99101579
%% Q1
clc; clear; close all;
% sigma_1 = input('Enter sigma_1: ');
% sigma_2 = input('Enter sigma_2: ');
% sigma_v = input('Enter sigma_v: ');
% a = input('Enter a: ');
% b = input('Enter b: ');
sigma_v = 1;
sigma_1 = 2;
sigma_2 = 2;
a = 0.5;
b = 0.5;

N = 250;  % Number of samples
%% Q1 - Part a
clc;
V = sigma_v * randn(1, N);
U_1 = sigma_1 * randn(1, N);
```

```
U_2 = sigma_2 * randn(1, N);


% system real outputs
f1 = zeros(2,N);
f3 = zeros(1,N);
for k = 2:N
    f1(1,k) = a * f1(1,k-1) + U_1(k);
    f1(2,k) = b * f1(2,k-1) + f1(1,k) + U_2(k);
    f3(k) = f1(2,k) + V(k);
end


X_hat = myKalmanFilter(a,b,sigma_1,sigma_2,sigma_v,f3,N,0);
X_hat_rand_initial = myKalmanFilter(a,b,sigma_1,sigma_2,sigma_v,f3,N,1);


plotResults(f1, X_hat, X_hat_rand_initial);


PCC_X1 = corrcoef(f1(1,:),X_hat(1,:));
PCC_X2 = corrcoef(f1(2,:),X_hat(2,:));
PCC_X1_rand_initial = corrcoef(f1(1,:),X_hat_rand_initial(1,:));
PCC_X2_rand_initial = corrcoef(f1(2,:),X_hat_rand_initial(2,:));
disp(['Pearson Correlation Coeficient for X_1 = ',num2str(PCC_X1(1,2))]);
disp(['Pearson Correlation Coeficient for X_2 = ',num2str(PCC_X2(1,2))]);
disp(['Pearson Correlation Coeficient for X_1 with random initialization =
',num2str(PCC_X1_rand_initial(1,2))]);
disp(['Pearson Correlation Coeficient for X_2 with random initialization =
',num2str(PCC_X2_rand_initial(1,2))]);
%% Q1 - Part b
clc;
V = sqrt(12) * sigma_v * rand(1, N); % generating noise with uniform distribution and
variance sigma_v
U_1 = sqrt(12) * sigma_1 * rand(1, N);
U_2 = sqrt(12) * sigma_2 * rand(1, N);
```

```
% system real outputs
f1 = zeros(2,N);
f3 = zeros(1,N);
for k = 2:N
    f1(1,k) = a * f1(1,k-1) + U_1(k);
    f1(2,k) = b * f1(2,k-1) + f1(1,k) + U_2(k);
    f3(k) = f1(2,k) + V(k);
end

X_hat = myKalmanFilter(a,b,sigma_1,sigma_2,sigma_v,f3,N,0);
X_hat_rand_initial = myKalmanFilter(a,b,sigma_1,sigma_2,sigma_v,f3,N,1);

plotResults(f1, X_hat, X_hat_rand_initial);

PCC_X1 = corrcoef(f1(1,:),X_hat(1,:));
PCC_X2 = corrcoef(f1(2,:),X_hat(2,:));
PCC_X1_rand_initial = corrcoef(f1(1,:),X_hat_rand_initial(1,:));
PCC_X2_rand_initial = corrcoef(f1(2,:),X_hat_rand_initial(2,:));
disp(['Pearson Correlation Coeficient for X_1 = ',num2str(PCC_X1(1,2))]);
disp(['Pearson Correlation Coeficient for X_2 = ',num2str(PCC_X2(1,2))]);
disp(['Pearson Correlation Coeficient for X_1 with random initialization =
',num2str(PCC_X1_rand_initial(1,2))]);
disp(['Pearson Correlation Coeficient for X_2 with random initialization =
',num2str(PCC_X2_rand_initial(1,2))]);
%% Q1 - Part c
clc;
V = sigma_v * randn(1, N);
U_1 = sigma_1 * randn(1, N);
U_2 = U_1 * sigma_2/sigma_1; % make correlation between U_1 and U_2

% system real outputs
f1 = zeros(2,N);
```

```
f3 = zeros(1,N);
for k = 2:N
    f1(1,k) = a * f1(1,k-1) + U_1(k);
    f1(2,k) = b * f1(2,k-1) + f1(1,k) + U_2(k);
    f3(k) = f1(2,k) + V(k);
end

X_hat = myKalmanFilter(a,b,sigma_1,sigma_2,sigma_v,f3,N,0);
X_hat_rand_initial = myKalmanFilter(a,b,sigma_1,sigma_2,sigma_v,f3,N,1);

plotResults(f1, X_hat, X_hat_rand_initial);

PCC_X1 = corrcoef(f1(1,:),X_hat(1,:));
PCC_X2 = corrcoef(f1(2,:),X_hat(2,:));
PCC_X1_rand_initial = corrcoef(f1(1,:),X_hat_rand_initial(1,:));
PCC_X2_rand_initial = corrcoef(f1(2,:),X_hat_rand_initial(2,:));
disp(['Pearson Correlation Coeficient for X_1 = ',num2str(PCC_X1(1,2))]);
disp(['Pearson Correlation Coeficient for X_2 = ',num2str(PCC_X2(1,2))]);
disp(['Pearson Correlation Coeficient for X_1 with random initialization =
',num2str(PCC_X1_rand_initial(1,2))]);
disp(['Pearson Correlation Coeficient for X_2 with random initialization =
',num2str(PCC_X2_rand_initial(1,2))]);

%% Q2
clc; clear;
load('ECG.mat');

train_data = {ECG01, ECG02, ECG03, ECG04, ECG05, ECG06, ECG07, ECG08, ECG09, ECG10, ...
              ECG11, ECG12, ECG13, ECG14, ECG15, ECG16, ECG17, ECG18, ECG19, ECG20, ...
              ECG21, ECG22, ECG23, ECG24, ECG25, ECG26, ECG27, ECG28, ECG29, ECG30, ...
              ECG31, ECG32, ECG33, ECG34, ECG35, ECG36, ECG37, ECG38, ECG39, ECG40};

test_data = {ECG41, ECG42, ECG43, ECG44, ECG45, ECG46, ECG47, ECG48, ECG49, ECG50, ...
```

```
                ECG51, ECG52, ECG53, ECG54, ECG55, ECG56, ECG57, ECG58, ECG59, ECG60, ...
                ECG61, ECG62, ECG63, ECG64, ECG65, ECG66, ECG67, ECG68, ECG69, ECG70, ...
                ECG71, ECG72, ECG73, ECG74, ECG75, ECG76, ECG77, ECG78, ECG79, ECG80};
%% Q2 - Part a
clc;
num_train = length(train_data);
features_train = zeros(3,num_train);
labels_train = zeros(1,num_train);

for i = 1:num_train
    signal = train_data{i};
    features_train (:, i) = featureExtractor(signal);

    if(i>20)
        labels_train(i) = 1;
    end

end

% % normalization - z score
% for i = 1:3
%     features_train(i,:) = (features_train(i,:) - mean(features_train(i,:)))/
var(features_train(i,:));
% end

%% Q2 - Part b
clc;

normal_features = features_train(:, labels_train == 0);
abnormal_features = features_train(:, labels_train == 1);

mu_normal = mean(normal_features,2);
```

```
mu_abnormal = mean(abnormal_features,2);


cov_normal = cov(normal_features');

cov_abnormal = cov(abnormal_features');


prior_normal = 0.5;

prior_abnormal = 0.5;


feature1_range = linspace(min(features_train(1,:)), max(features_train(1,:)), 100);

feature2_range = linspace(min(features_train(2,:)), max(features_train(2,:)), 100);

feature3_range = linspace(min(features_train(3,:)), max(features_train(3,:)), 100);


[f1, f2, f3] = meshgrid(feature1_range, feature2_range, feature3_range);


g_normal = zeros(size(f1));

g_abnormal = zeros(size(f1));


f1_flat = f1(:);

f2_flat = f2(:);

f3_flat = f3(:);


for i = 1:length(f1_flat)


    x = [f1_flat(i); f2_flat(i); f3_flat(i)];


    g_normal(i) = -0.5 * (x - mu_normal)' * inv(cov_normal) * (x-mu_normal) +
log(prior_normal) - 0.5 * log(det(cov_normal));

    g_abnormal(i) = -0.5 * (x - mu_abnormal)' * inv(cov_abnormal) * (x-mu_abnormal) +
log(prior_abnormal) - 0.5 * log(det(cov_abnormal));

end


g_normal = reshape(g_normal, size(f1));

g_abnormal = reshape(g_abnormal, size(f1));
```

```
g12 = g_normal - g_abnormal;


figure;
isosurface(f1, f2, f3, g12, 0,'k');
hold on;
scatter3(normal_features(1,:), normal_features(2,:), normal_features(3,:), 'r','filled');
hold on;
scatter3(abnormal_features(1,:), abnormal_features(2,:), abnormal_features(3,:),
'b','filled');
xlabel('Feature 1 ','Interpreter','latex');
ylabel('Feature 2 ','Interpreter','latex');
zlabel('Feature 3 ','Interpreter','latex');
title('Decision Boundary','Interpreter','latex','FontSize',14);
grid on;


%% Q2 - Part c
clc;


cov_all = cov(features_train');
[V, D] = eig(cov_all);
eigenvalues = diag(D);
[sortedEigenvalues, ind] = sort(eigenvalues, 'descend');
sortedEigenvectors = V(:, ind);


u1 = sortedEigenvectors(:,2:3);


features_train_reduced = u1' * features_train ;


normal_features = features_train_reduced(:, labels_train == 0);
abnormal_features = features_train_reduced(:, labels_train == 1);
```

```
mu_normal = mean(normal_features,2);

mu_abnormal = mean(abnormal_features,2);


cov_normal = cov(normal_features');

cov_abnormal = cov(abnormal_features');


prior_normal = 0.5;

prior_abnormal = 0.5;


feature1_range = linspace(min(features_train_reduced(1,:)),
max(features_train_reduced(1,:)), 100);

feature2_range = linspace(min(features_train_reduced(2,:)),
max(features_train_reduced(2,:)), 100);



[f1, f2] = meshgrid(feature1_range, feature2_range);


g_normal = zeros(size(f1));

g_abnormal = zeros(size(f1));


f1_flat = f1(:);

f2_flat = f2(:);



for i = 1:length(f1_flat)


    x = [f1_flat(i); f2_flat(i)];


    g_normal(i) = -0.5 * (x - mu_normal)' * inv(cov_normal) * (x-mu_normal) +
log(prior_normal) - 0.5 * log(det(cov_normal));

    g_abnormal(i) = -0.5 * (x - mu_abnormal)' * inv(cov_abnormal) * (x-mu_abnormal) +
log(prior_abnormal) - 0.5 * log(det(cov_abnormal));

end
```

```
g_normal = reshape(g_normal, size(f1));
g_abnormal = reshape(g_abnormal, size(f1));

g12 = g_normal - g_abnormal;

figure;
contour(f1, f2, g12, [0 0]);
hold on;
scatter(normal_features(1,:), normal_features(2,:), 'r','filled');
hold on;
scatter(abnormal_features(1,:), abnormal_features(2,:), 'b','filled');
xlabel('Feature 1 ','Interpreter','latex');
ylabel('Feature 2 ','Interpreter','latex');
zlabel('Feature 3 ','Interpreter','latex');
title('Decision Boundary - Minimizing Entropy','Interpreter','latex','FontSize',14);
grid on;

%% Q2 - Part d
clc;

normal_features = features_train(:, labels_train == 0);
abnormal_features = features_train(:, labels_train == 1);

mu_normal = mean(normal_features,2);
mu_abnormal = mean(abnormal_features,2);

cov_normal = cov(normal_features');
cov_abnormal = cov(abnormal_features');

prior_normal = 0.5;
prior_abnormal = 0.5;
```

```
S_w = prior_normal * cov_normal + prior_abnormal * cov_abnormal;

w = inv(S_w)*(mu_normal - mu_abnormal);

features_train_reduced = w' * features_train;


normal_features = features_train_reduced(:, labels_train == 0);

abnormal_features = features_train_reduced(:, labels_train == 1);


mu_normal = mean(normal_features);

mu_abnormal = mean(abnormal_features);


var_normal = var(normal_features);

var_abnormal = var(abnormal_features);


feature1_range = linspace(min(features_train_reduced), max(features_train_reduced),
1000);


f1 = feature1_range;


g_normal = zeros(size(f1));

g_abnormal = zeros(size(f1));


f1_flat = f1(:);




for i = 1:length(f1_flat)


    x = [f1_flat(i)];


    g_normal(i) = -0.5 * (x - mu_normal)^2 / var_normal + log(prior_normal) - 0.5 *
log(var_normal);

    g_abnormal(i) = -0.5 * (x - mu_abnormal)^2 / var_abnormal + log(prior_abnormal) - 0.5
* log(var_abnormal);

end
```

```
g_normal = reshape(g_normal, size(f1));
g_abnormal = reshape(g_abnormal, size(f1));

g12 = g_normal - g_abnormal;

boundry = find(g12 == min(abs(g12))) / 1000 + feature1_range(1);
if isempty(boundry == 1)
    boundry = find(g12 == -1*min(abs(g12))) / 1000 * (feature1_range(end) -
feature1_range(1)) + feature1_range(1) ;
end

figure;
plot([boundry boundry], [-1 1], 'k--', 'LineWidth', 2);
hold on;
scatter(normal_features, 0, 'r','filled');
hold on;
scatter(abnormal_features, 0, 'b','filled');
xlabel('Feature 1 ','Interpreter','latex');
title('Decision Boundary - FLD','Interpreter','latex','FontSize',14);
grid on;

%% Q2 - Part e
clc;

% part a
num_test = length(test_data);
features_test = zeros(3,num_test);
labels_test = zeros(1,num_test);

for i = 1:num_test
    signal = test_data{i};
```

```
    features_test (:, i) = featureExtractor(signal);
    if(i>20)
        labels_test(i) = 1;
    end
end

% part b
normal_features_train = features_train(:, labels_train == 0);
abnormal_features_train = features_train(:, labels_train == 1);

mu_normal = mean(normal_features_train,2);
mu_abnormal = mean(abnormal_features_train,2);

cov_normal = cov(normal_features_train');
cov_abnormal = cov(abnormal_features_train');

predicted_labels_mah = zeros(1, num_test);
predicted_labels_euc = zeros(1, num_test);
predicted_labels_bayes = zeros(1, num_test);
for i=1:num_test

    d_normal_mah = (features_test(:,i)-mu_normal)'*inv(cov_normal)*(features_test(:,i)-
mu_normal);
    d_abnormal_mah = (features_test(:,i)-
mu_abnormal)'*inv(cov_abnormal)*(features_test(:,i)-mu_abnormal);

    d_normal_euc = (features_test(:,i)-mu_normal)'*(features_test(:,i)-mu_normal);
    d_abnormal_euc = (features_test(:,i)-mu_abnormal)'*(features_test(:,i)-mu_abnormal);

    g_normal_bayes = -0.5 * (features_test(:,i) - mu_normal)' * inv(cov_normal) *
(features_test(:,i)-mu_normal) + log(prior_normal) - 0.5 * log(det(cov_normal));
    g_abnormal_bayes = -0.5 * (features_test(:,i) - mu_abnormal)' * inv(cov_abnormal) *
(features_test(:,i)-mu_abnormal) + log(prior_abnormal) - 0.5 * log(det(cov_abnormal));
```

```
    if (d_abnormal_mah < d_normal_mah)
        predicted_labels_mah(i) = 1;
    elseif (d_abnormal_mah == d_normal_mah)
        if (randn > 0)
            predicted_labels_mah(i) = 1;
        end
    end

    if (d_abnormal_euc < d_normal_euc)
        predicted_labels_euc(i) = 1;
    elseif (d_abnormal_euc == d_normal_euc)
        if (randn > 0)
            predicted_labels_euc(i) = 1;
        end
    end

    if (g_abnormal_bayes - g_normal_bayes > 0)
        predicted_labels_bayes(i) = 1;
    elseif (g_abnormal_bayes - g_normal_bayes == 0)
        if (randn > 0)
            predicted_labels_bayes(i) = 1;
        end
    end

end

acc_mah = sum(labels_test == predicted_labels_mah)/num_test;
acc_euc = sum(labels_test == predicted_labels_euc)/num_test;
acc_bayes = sum(labels_test == predicted_labels_bayes)/num_test;
disp('Part b Results:')
disp(['Accuracy of Mahalanobis Distance = ',num2str(acc_mah )]);
disp(['Accuracy of Euclidean Distance = ',num2str(acc_euc )]);
```

```
disp(['Accuracy of Bayes Classifier = ',num2str(acc_bayes )]);
disp('============================================================')

% part c
cov_all_train = cov(features_train');
[V, D] = eig(cov_all_train);
eigenvalues = diag(D);
[sortedEigenvalues, ind] = sort(eigenvalues, 'descend');
sortedEigenvectors = V(:, ind);

u1 = sortedEigenvectors(:,2:3);

features_train_reduced = u1' * features_train ;
features_test_reduced = u1' * features_test ;

normal_features = features_train_reduced(:, labels_train == 0);
abnormal_features = features_train_reduced(:, labels_train == 1);

mu_normal = mean(normal_features,2);
mu_abnormal = mean(abnormal_features,2);

cov_normal = cov(normal_features');
cov_abnormal = cov(abnormal_features');

predicted_labels_mah = zeros(1, num_test);
predicted_labels_euc = zeros(1, num_test);
predicted_labels_bayes = zeros(1, num_test);
for i=1:num_test

    d_normal_mah = (features_test_reduced(:,i)-
mu_normal)'*inv(cov_normal)*(features_test_reduced(:,i)-mu_normal);
    d_abnormal_mah = (features_test_reduced(:,i)-
mu_abnormal)'*inv(cov_abnormal)*(features_test_reduced(:,i)-mu_abnormal);
```

```
    d_normal_euc = (features_test_reduced(:,i)-mu_normal)'*(features_test_reduced(:,i)-
mu_normal);

    d_abnormal_euc = (features_test_reduced(:,i)-
mu_abnormal)'*(features_test_reduced(:,i)-mu_abnormal);


    g_normal_bayes = -0.5 * (features_test_reduced(:,i) - mu_normal)' * inv(cov_normal) *
(features_test_reduced(:,i)-mu_normal) + log(prior_normal) - 0.5 * log(det(cov_normal));

    g_abnormal_bayes = -0.5 * (features_test_reduced(:,i) - mu_abnormal)' *
inv(cov_abnormal) * (features_test_reduced(:,i)-mu_abnormal) + log(prior_abnormal) - 0.5
* log(det(cov_abnormal));


    if (d_abnormal_mah < d_normal_mah)
        predicted_labels_mah(i) = 1;
    elseif (d_abnormal_mah == d_normal_mah)
        if (randn > 0)
            predicted_labels_mah(i) = 1;
        end
    end


    if (d_abnormal_euc < d_normal_euc)
        predicted_labels_euc(i) = 1;
    elseif (d_abnormal_euc == d_normal_euc)
        if (randn > 0)
            predicted_labels_euc(i) = 1;
        end
    end


    if (g_abnormal_bayes - g_normal_bayes > 0)
        predicted_labels_bayes(i) = 1;
    elseif (g_abnormal_bayes - g_normal_bayes == 0)
        if (randn > 0)
            predicted_labels_bayes(i) = 1;
        end
```

```
    end

end

acc_mah = sum(labels_test == predicted_labels_mah)/num_test;
acc_euc = sum(labels_test == predicted_labels_euc)/num_test;
acc_bayes = sum(labels_test == predicted_labels_bayes)/num_test;
disp('Part c Results:')
disp(['Accuracy of Mahalanobis Distance = ',num2str(acc_mah )]);
disp(['Accuracy of Euclidean Distance = ',num2str(acc_euc )]);
disp(['Accuracy of Bayes Classifier = ',num2str(acc_bayes )]);
disp('============================================================')

% Part d
normal_features = features_train(:, labels_train == 0);
abnormal_features = features_train(:, labels_train == 1);

mu_normal = mean(normal_features,2);
mu_abnormal = mean(abnormal_features,2);

cov_normal = cov(normal_features');
cov_abnormal = cov(abnormal_features');

prior_normal = 0.5;
prior_abnormal = 0.5;

S_w = prior_normal * cov_normal + prior_abnormal * cov_abnormal;
w = inv(S_w)*(mu_normal - mu_abnormal);

features_train_reduced = w' * features_train;
features_test_reduced = w' * features_test;
```

```
normal_features = features_train_reduced(:, labels_train == 0);
abnormal_features = features_train_reduced(:, labels_train == 1);

mu_normal = mean(normal_features);
mu_abnormal = mean(abnormal_features);

var_normal = var(normal_features);
var_abnormal = var(abnormal_features);

predicted_labels_mah = zeros(1, num_test);
predicted_labels_euc = zeros(1, num_test);
predicted_labels_bayes = zeros(1, num_test);
for i=1:num_test

    d_normal_mah = (features_test_reduced(i)-mu_normal)^2 / var_normal;
    d_abnormal_mah = (features_test_reduced(i)-mu_abnormal)^2 / var_abnormal;

    d_normal_euc = (features_test_reduced(:,i)-mu_normal)^2;
    d_abnormal_euc = (features_test_reduced(:,i)-mu_abnormal)^2;

    g_normal_bayes = -0.5 * (features_test_reduced(i) - mu_normal)^2 / var_normal +
log(prior_normal) - 0.5 * log(var_normal);
    g_abnormal_bayes = -0.5 * (features_test_reduced(i) - mu_abnormal)^2 / var_abnormal +
log(prior_abnormal) - 0.5 * log(var_abnormal);

    if (d_abnormal_mah < d_normal_mah)
        predicted_labels_mah(i) = 1;
    elseif (d_abnormal_mah == d_normal_mah)
        if (randn > 0)
            predicted_labels_mah(i) = 1;
        end
    end
```

```
    if (d_abnormal_euc < d_normal_euc)
        predicted_labels_euc(i) = 1;
    elseif (d_abnormal_euc == d_normal_euc)
        if (randn > 0)
            predicted_labels_euc(i) = 1;
        end
    end

    if (g_abnormal_bayes - g_normal_bayes > 0)
        predicted_labels_bayes(i) = 1;
    elseif (g_abnormal_bayes - g_normal_bayes == 0)
        if (randn > 0)
            predicted_labels_bayes(i) = 1;
        end
    end
end

acc_mah = sum(labels_test == predicted_labels_mah)/num_test;
acc_euc = sum(labels_test == predicted_labels_euc)/num_test;
acc_bayes = sum(labels_test == predicted_labels_bayes)/num_test;
disp('Part d Results:')
disp(['Accuracy of Mahalanobis Distance = ',num2str(acc_mah )]);
disp(['Accuracy of Euclidean Distance = ',num2str(acc_euc )]);
disp(['Accuracy of Bayes Classifier = ',num2str(acc_bayes )]);
disp('============================================================')
%% Q2 - Part f - a
clc; clear;
load('ECG.mat');
train_data = {ECG01, ECG02, ECG03, ECG04, ECG05, ECG06, ECG07, ECG08, ECG09, ECG10, ...
              ECG11, ECG12, ECG13, ECG14, ECG15, ECG16, ECG17, ECG18, ECG19, ECG20, ...
              ECG21, ECG22, ECG23, ECG24, ECG25, ECG26, ECG27, ECG28, ECG29, ECG30, ...
              ECG31, ECG32, ECG33, ECG34, ECG35, ECG36, ECG37, ECG38, ECG39, ECG40};
```

```
test_data = {ECG41, ECG42, ECG43, ECG44, ECG45, ECG46, ECG47, ECG48, ECG49, ECG50, ...
               ECG51, ECG52, ECG53, ECG54, ECG55, ECG56, ECG57, ECG58, ECG59, ECG60, ...
               ECG61, ECG62, ECG63, ECG64, ECG65, ECG66, ECG67, ECG68, ECG69, ECG70, ...
               ECG71, ECG72, ECG73, ECG74, ECG75, ECG76, ECG77, ECG78, ECG79, ECG80};


num_train = length(train_data);
features_train = zeros(3,num_train);
labels_train = zeros(1,num_train);

for i = 1:num_train
    signal = train_data{i};
    features_train (:, i) = featureExtractor_new(signal);

    if(i>20)
        labels_train(i) = 1;
    end

end

% % normalization - z score
% for i = 1:3
%     features_train(i,:) = (features_train(i,:) - mean(features_train(i,:)))/
var(features_train(i,:));
% end

%% Q2 - Part f - b
clc;

normal_features = features_train(:, labels_train == 0);
abnormal_features = features_train(:, labels_train == 1);
```

```
mu_normal = mean(normal_features,2);
mu_abnormal = mean(abnormal_features,2);

cov_normal = cov(normal_features');
cov_abnormal = cov(abnormal_features');

prior_normal = 0.5;
prior_abnormal = 0.5;

feature1_range = linspace(min(features_train(1,:)), max(features_train(1,:)), 100);
feature2_range = linspace(min(features_train(2,:)), max(features_train(2,:)), 100);
feature3_range = linspace(min(features_train(3,:)), max(features_train(3,:)), 100);

[f1, f2, f3] = meshgrid(feature1_range, feature2_range, feature3_range);

g_normal = zeros(size(f1));
g_abnormal = zeros(size(f1));

f1_flat = f1(:);
f2_flat = f2(:);
f3_flat = f3(:);

for i = 1:length(f1_flat)

    x = [f1_flat(i); f2_flat(i); f3_flat(i)];

    g_normal(i) = -0.5 * (x - mu_normal)' * inv(cov_normal) * (x-mu_normal) +
log(prior_normal) - 0.5 * log(det(cov_normal));
    g_abnormal(i) = -0.5 * (x - mu_abnormal)' * inv(cov_abnormal) * (x-mu_abnormal) +
log(prior_abnormal) - 0.5 * log(det(cov_abnormal));
end

g_normal = reshape(g_normal, size(f1));
```

```
g_abnormal = reshape(g_abnormal, size(f1));


g12 = g_normal - g_abnormal;


figure;
isosurface(f1, f2, f3, g12, 0,'k');
hold on;
scatter3(normal_features(1,:), normal_features(2,:), normal_features(3,:), 'r','filled');
hold on;
scatter3(abnormal_features(1,:), abnormal_features(2,:), abnormal_features(3,:),
'b','filled');
xlabel('Feature 1 ','Interpreter','latex');
ylabel('Feature 2 ','Interpreter','latex');
zlabel('Feature 3 ','Interpreter','latex');
title('Decision Boundary','Interpreter','latex','FontSize',14);
grid on;


%% Q2 - Part f - c
clc;


cov_all = cov(features_train');
[V, D] = eig(cov_all);
eigenvalues = diag(D);
[sortedEigenvalues, ind] = sort(eigenvalues, 'descend');
sortedEigenvectors = V(:, ind);


u1 = sortedEigenvectors(:,2:3);


features_train_reduced = u1' * features_train ;


normal_features = features_train_reduced(:, labels_train == 0);
abnormal_features = features_train_reduced(:, labels_train == 1);
```

```
mu_normal = mean(normal_features,2);

mu_abnormal = mean(abnormal_features,2);


cov_normal = cov(normal_features');

cov_abnormal = cov(abnormal_features');


prior_normal = 0.5;

prior_abnormal = 0.5;


feature1_range = linspace(min(features_train_reduced(1,:)),
max(features_train_reduced(1,:)), 100);

feature2_range = linspace(min(features_train_reduced(2,:)),
max(features_train_reduced(2,:)), 100);




[f1, f2] = meshgrid(feature1_range, feature2_range);


g_normal = zeros(size(f1));

g_abnormal = zeros(size(f1));


f1_flat = f1(:);

f2_flat = f2(:);




for i = 1:length(f1_flat)


    x = [f1_flat(i); f2_flat(i)];


    g_normal(i) = -0.5 * (x - mu_normal)' * inv(cov_normal) * (x-mu_normal) +
log(prior_normal) - 0.5 * log(det(cov_normal));

    g_abnormal(i) = -0.5 * (x - mu_abnormal)' * inv(cov_abnormal) * (x-mu_abnormal) +
log(prior_abnormal) - 0.5 * log(det(cov_abnormal));

end
```

```
g_normal = reshape(g_normal, size(f1));
g_abnormal = reshape(g_abnormal, size(f1));

g12 = g_normal - g_abnormal;

figure;
contour(f1, f2, g12, [0 0]);
hold on;
scatter(normal_features(1,:), normal_features(2,:), 'r','filled');
hold on;
scatter(abnormal_features(1,:), abnormal_features(2,:), 'b','filled');
xlabel('Feature 1 ','Interpreter','latex');
ylabel('Feature 2 ','Interpreter','latex');
zlabel('Feature 3 ','Interpreter','latex');
title('Decision Boundary - Minimizing Entropy','Interpreter','latex','FontSize',14);
grid on;

%% Q2 - Part f - d
clc;

normal_features = features_train(:, labels_train == 0);
abnormal_features = features_train(:, labels_train == 1);

mu_normal = mean(normal_features,2);
mu_abnormal = mean(abnormal_features,2);

cov_normal = cov(normal_features');
cov_abnormal = cov(abnormal_features');

prior_normal = 0.5;
prior_abnormal = 0.5;
```

```
S_w = prior_normal * cov_normal + prior_abnormal * cov_abnormal;

w = inv(S_w)*(mu_normal - mu_abnormal);

features_train_reduced = w' * features_train;


normal_features = features_train_reduced(:, labels_train == 0);

abnormal_features = features_train_reduced(:, labels_train == 1);


mu_normal = mean(normal_features);

mu_abnormal = mean(abnormal_features);


var_normal = var(normal_features);

var_abnormal = var(abnormal_features);


feature1_range = linspace(min(features_train_reduced), max(features_train_reduced),
1000);


f1 = feature1_range;


g_normal = zeros(size(f1));

g_abnormal = zeros(size(f1));


f1_flat = f1(:);



for i = 1:length(f1_flat)


    x = [f1_flat(i)];


    g_normal(i) = -0.5 * (x - mu_normal)^2 / var_normal + log(prior_normal) - 0.5 *
log(var_normal);

    g_abnormal(i) = -0.5 * (x - mu_abnormal)^2 / var_abnormal + log(prior_abnormal) - 0.5
* log(var_abnormal);
```

```
end


g_normal = reshape(g_normal, size(f1));
g_abnormal = reshape(g_abnormal, size(f1));


g12 = g_normal - g_abnormal;


boundry = find(g12 == min(abs(g12))) / 1000 + feature1_range(1);
if isempty(boundry == 1)
    boundry = find(g12 == -1*min(abs(g12))) / 1000 * (feature1_range(end) -
feature1_range(1)) + feature1_range(1) ;
end


figure;
plot([boundry boundry], [-1 1], 'k--', 'LineWidth', 2);
hold on;
scatter(normal_features, 0, 'r','filled');
hold on;
scatter(abnormal_features, 0, 'b','filled');
xlabel('Feature 1 ','Interpreter','latex');
title('Decision Boundary - FLD','Interpreter','latex','FontSize',14);
grid on;


%% Q2 - Part f - e
clc;


% part a
num_test = length(test_data);
features_test = zeros(3,num_test);
labels_test = zeros(1,num_test);


for i = 1:num_test
```

```
    signal = test_data{i};
    features_test (:, i) = featureExtractor_new(signal);
    if(i>20)
        labels_test(i) = 1;
    end
end

% part b
normal_features_train = features_train(:, labels_train == 0);
abnormal_features_train = features_train(:, labels_train == 1);

mu_normal = mean(normal_features_train,2);
mu_abnormal = mean(abnormal_features_train,2);

cov_normal = cov(normal_features_train');
cov_abnormal = cov(abnormal_features_train');

predicted_labels_mah = zeros(1, num_test);
predicted_labels_euc = zeros(1, num_test);
predicted_labels_bayes = zeros(1, num_test);
for i=1:num_test

    d_normal_mah = (features_test(:,i)-mu_normal)'*inv(cov_normal)*(features_test(:,i)-
mu_normal);
    d_abnormal_mah = (features_test(:,i)-
mu_abnormal)'*inv(cov_abnormal)*(features_test(:,i)-mu_abnormal);

    d_normal_euc = (features_test(:,i)-mu_normal)'*(features_test(:,i)-mu_normal);
    d_abnormal_euc = (features_test(:,i)-mu_abnormal)'*(features_test(:,i)-mu_abnormal);

    g_normal_bayes = -0.5 * (features_test(:,i) - mu_normal)' * inv(cov_normal) *
(features_test(:,i)-mu_normal) + log(prior_normal) - 0.5 * log(det(cov_normal));
    g_abnormal_bayes = -0.5 * (features_test(:,i) - mu_abnormal)' * inv(cov_abnormal) *
(features_test(:,i)-mu_abnormal) + log(prior_abnormal) - 0.5 * log(det(cov_abnormal));
```

```
    if (d_abnormal_mah < d_normal_mah)
        predicted_labels_mah(i) = 1;
    elseif (d_abnormal_mah == d_normal_mah)
        if (randn > 0)
            predicted_labels_mah(i) = 1;
        end
    end

    if (d_abnormal_euc < d_normal_euc)
        predicted_labels_euc(i) = 1;
    elseif (d_abnormal_euc == d_normal_euc)
        if (randn > 0)
            predicted_labels_euc(i) = 1;
        end
    end

    if (g_abnormal_bayes - g_normal_bayes > 0)
        predicted_labels_bayes(i) = 1;
    elseif (g_abnormal_bayes - g_normal_bayes == 0)
        if (randn > 0)
            predicted_labels_bayes(i) = 1;
        end
    end

end

acc_mah = sum(labels_test == predicted_labels_mah)/num_test;
acc_euc = sum(labels_test == predicted_labels_euc)/num_test;
acc_bayes = sum(labels_test == predicted_labels_bayes)/num_test;
disp('Part b Results:')
disp(['Accuracy of Mahalanobis Distance = ',num2str(acc_mah )]);
```

```
disp(['Accuracy of Euclidean Distance = ',num2str(acc_euc )]);

disp(['Accuracy of Bayes Classifier = ',num2str(acc_bayes )]);

disp('============================================================')


% part c

cov_all_train = cov(features_train');

[V, D] = eig(cov_all_train);

eigenvalues = diag(D);

[sortedEigenvalues, ind] = sort(eigenvalues, 'descend');

sortedEigenvectors = V(:, ind);


u1 = sortedEigenvectors(:,2:3);


features_train_reduced = u1' * features_train ;

features_test_reduced = u1' * features_test ;


normal_features = features_train_reduced(:, labels_train == 0);

abnormal_features = features_train_reduced(:, labels_train == 1);


mu_normal = mean(normal_features,2);

mu_abnormal = mean(abnormal_features,2);


cov_normal = cov(normal_features');

cov_abnormal = cov(abnormal_features');


predicted_labels_mah = zeros(1, num_test);

predicted_labels_euc = zeros(1, num_test);

predicted_labels_bayes = zeros(1, num_test);

for i=1:num_test


    d_normal_mah = (features_test_reduced(:,i)-
mu_normal)'*inv(cov_normal)*(features_test_reduced(:,i)-mu_normal);
```

```
    d_abnormal_mah = (features_test_reduced(:,i)-
mu_abnormal)'*inv(cov_abnormal)*(features_test_reduced(:,i)-mu_abnormal);


    d_normal_euc = (features_test_reduced(:,i)-mu_normal)'*(features_test_reduced(:,i)-
mu_normal);
    d_abnormal_euc = (features_test_reduced(:,i)-
mu_abnormal)'*(features_test_reduced(:,i)-mu_abnormal);


    g_normal_bayes = -0.5 * (features_test_reduced(:,i) - mu_normal)' * inv(cov_normal) *
(features_test_reduced(:,i)-mu_normal) + log(prior_normal) - 0.5 * log(det(cov_normal));
    g_abnormal_bayes = -0.5 * (features_test_reduced(:,i) - mu_abnormal)' *
inv(cov_abnormal) * (features_test_reduced(:,i)-mu_abnormal) + log(prior_abnormal) - 0.5
* log(det(cov_abnormal));


    if (d_abnormal_mah < d_normal_mah)
        predicted_labels_mah(i) = 1;
    elseif (d_abnormal_mah == d_normal_mah)
        if (randn > 0)
            predicted_labels_mah(i) = 1;
        end
    end


    if (d_abnormal_euc < d_normal_euc)
        predicted_labels_euc(i) = 1;
    elseif (d_abnormal_euc == d_normal_euc)
        if (randn > 0)
            predicted_labels_euc(i) = 1;
        end
    end


    if (g_abnormal_bayes - g_normal_bayes > 0)
        predicted_labels_bayes(i) = 1;
    elseif (g_abnormal_bayes - g_normal_bayes == 0)
        if (randn > 0)
```

```
            predicted_labels_bayes(i) = 1;
        end
    end

end

acc_mah = sum(labels_test == predicted_labels_mah)/num_test;
acc_euc = sum(labels_test == predicted_labels_euc)/num_test;
acc_bayes = sum(labels_test == predicted_labels_bayes)/num_test;
disp('Part c Results:')
disp(['Accuracy of Mahalanobis Distance = ',num2str(acc_mah )]);
disp(['Accuracy of Euclidean Distance = ',num2str(acc_euc )]);
disp(['Accuracy of Bayes Classifier = ',num2str(acc_bayes )]);
disp('============================================================')

% Part d
normal_features = features_train(:, labels_train == 0);
abnormal_features = features_train(:, labels_train == 1);

mu_normal = mean(normal_features,2);
mu_abnormal = mean(abnormal_features,2);

cov_normal = cov(normal_features');
cov_abnormal = cov(abnormal_features');

prior_normal = 0.5;
prior_abnormal = 0.5;

S_w = prior_normal * cov_normal + prior_abnormal * cov_abnormal;
w = inv(S_w)*(mu_normal - mu_abnormal);

features_train_reduced = w' * features_train;
```

```
features_test_reduced = w' * features_test;


normal_features = features_train_reduced(:, labels_train == 0);

abnormal_features = features_train_reduced(:, labels_train == 1);


mu_normal = mean(normal_features);

mu_abnormal = mean(abnormal_features);


var_normal = var(normal_features);

var_abnormal = var(abnormal_features);


predicted_labels_mah = zeros(1, num_test);

predicted_labels_euc = zeros(1, num_test);

predicted_labels_bayes = zeros(1, num_test);

for i=1:num_test


    d_normal_mah = (features_test_reduced(i)-mu_normal)^2 / var_normal;

    d_abnormal_mah = (features_test_reduced(i)-mu_abnormal)^2 / var_abnormal;


    d_normal_euc = (features_test_reduced(:,i)-mu_normal)^2;

    d_abnormal_euc = (features_test_reduced(:,i)-mu_abnormal)^2;


    g_normal_bayes = -0.5 * (features_test_reduced(i) - mu_normal)^2 / var_normal +
log(prior_normal) - 0.5 * log(var_normal);

    g_abnormal_bayes = -0.5 * (features_test_reduced(i) - mu_abnormal)^2 / var_abnormal +
log(prior_abnormal) - 0.5 * log(var_abnormal);


    if (d_abnormal_mah < d_normal_mah)

        predicted_labels_mah(i) = 1;

    elseif (d_abnormal_mah == d_normal_mah)

        if (randn > 0)

            predicted_labels_mah(i) = 1;

        end
```

```
    end

    if (d_abnormal_euc < d_normal_euc)
        predicted_labels_euc(i) = 1;
    elseif (d_abnormal_euc == d_normal_euc)
        if (randn > 0)
            predicted_labels_euc(i) = 1;
        end
    end

    if (g_abnormal_bayes - g_normal_bayes > 0)
        predicted_labels_bayes(i) = 1;
    elseif (g_abnormal_bayes - g_normal_bayes == 0)
        if (randn > 0)
            predicted_labels_bayes(i) = 1;
        end
    end
end

acc_mah = sum(labels_test == predicted_labels_mah)/num_test;
acc_euc = sum(labels_test == predicted_labels_euc)/num_test;
acc_bayes = sum(labels_test == predicted_labels_bayes)/num_test;
disp('Part d Results:')
disp(['Accuracy of Mahalanobis Distance = ',num2str(acc_mah )]);
disp(['Accuracy of Euclidean Distance = ',num2str(acc_euc )]);
disp(['Accuracy of Bayes Classifier = ',num2str(acc_bayes )]);
disp('========================================================')

%% Q3 - Part a
clc; clear;

n = 0:50;
```

```
x = (0.8.^n) - 0.5 * (0.8.^(n-1)) .* (n > 0);


c_cep_matlab = cceps(x);


fft_x = fft(x);
complex_cepstrum_definition = ifft(log(fft_x));


c_cep_rec = [];
c_cep_rec(1) = log(x(1));
for i=2:length(n)
    a = 0;
    for j=1:i-1
        a = a + (j-1)/(i-1)*c_cep_rec(j)*x(i-(j-1))/x(1);
    end
    c_cep_rec(i) = x(i)/x(1) - a;
end


figure;
subplot(3,1,1)
plot(c_cep_matlab,'LineWidth',1)
title('complex cepstrum using matlab cceps','Interpreter','latex','FontSize',14)
ylabel('$\hat{x}[n]$','Interpreter','latex')
xlabel('n','Interpreter','latex')
subplot(3,1,2)
plot(complex_cepstrum_definition,'LineWidth',1)
title('complex cepstrum using DFT','Interpreter','latex','FontSize',14)
ylabel('$\hat{x}[n]$','Interpreter','latex')
xlabel('n','Interpreter','latex')
subplot(3,1,3)
plot(c_cep_rec,'LineWidth',1)
title('complex cepstrum using recursive algorithm','Interpreter','latex','FontSize',14)
ylabel('$\hat{x}[n]$','Interpreter','latex')
```

```
xlabel('n','Interpreter','latex')
%% part b
clc; clear;
data = load('ECG.mat');
fs = 200;
normal_ECG = data.ECG05;
abnormal_ECG = data.ECG34;
t_normal = linspace(0,length(normal_ECG)/fs,length(normal_ECG));
t_abnormal = linspace(0,length(abnormal_ECG)/fs,length(abnormal_ECG));

normal_ECG_cceps = cceps(normal_ECG);
abnormal_ECG_cceps = cceps(abnormal_ECG);
normal_ECG_rceps = rceps(normal_ECG);
abnormal_ECG_rceps = rceps(abnormal_ECG);

figure;
subplot(3,2,1);
plot(t_normal,normal_ECG,'LineWidth',1);
title('Normal ECG - Time Domain','Interpreter','latex','Fontsize',14);
xlabel('time (s)');
ylabel('Amplitude');

subplot(3,2,2);
plot(t_abnormal,abnormal_ECG,'LineWidth',1);
title('Abnormal ECG - Time Domain','Interpreter','latex','Fontsize',14);
xlabel('time (s)');
ylabel('Amplitude');

subplot(3,2,3);
plot(t_normal(5:end-5),normal_ECG_cceps(5:end-5),'LineWidth',1);
title('Normal ECG - Complex Cepstrum','Interpreter','latex','Fontsize',14);
xlabel('time (s)','Interpreter','latex');
```

```
ylabel('$\hat{x}[n]$','Interpreter','latex');


subplot(3,2,4);
plot(t_abnormal(5:end-5),abnormal_ECG_cceps(5:end-5),'LineWidth',1);
title('Abnormal ECG - Complex Cepstrum','Interpreter','latex','Fontsize',14);
xlabel('time (s)','Interpreter','latex');
ylabel('$\hat{x}[n]$','Interpreter','latex');


subplot(3,2,5);
plot(t_normal(5:end-5),normal_ECG_rceps(5:end-5),'LineWidth',1);
title('Normal ECG - Real Cepstrum','Interpreter','latex','Fontsize',14);
xlabel('time (s)','Interpreter','latex');
ylabel('$C_x[n]$','Interpreter','latex');


subplot(3,2,6);
plot(t_abnormal(5:end-5),abnormal_ECG_rceps(5:end-5),'LineWidth',1);
title('Abnormal ECG - Real Cepstrum','Interpreter','latex','Fontsize',14);
xlabel('time (s)','Interpreter','latex');
ylabel('$C_x[n]$','Interpreter','latex');




%% Function
function [X_hat] = myKalmanFilter(a,b,sigma_1,sigma_2,sigma_v,Z,N,initialization)
    F_k = [a 0; a b];
    H_k = [0 1];
    Q_k = [sigma_1^2 sigma_1^2; sigma_1^2 sigma_1^2+sigma_2^2];
    R_k = sigma_v^2;

    % Initial condition
    if initialization == 0
        X_hat = zeros(2,N);
```

```
        P = [sigma_1^2/(1-a^2) sigma_2^2/((1-a^2)*(1-a*b));
            sigma_2^2/((1-a^2)*(1-a*b)) (sigma_1^2 +(1-a^2)*sigma_2^2)/((1-a^2)*(1-
b^2))];
    else
        X_hat = 10*randn(2,N);
        P = 10*randn(2,2);
    end

    for k = 2:N
        % Prediction
        X_hat(:,k) = F_k * X_hat(:,k-1);
        P = F_k*P*F_k' + Q_k;

        % Update
        G = P * H_k'* inv(H_k*P*H_k'+R_k);
        X_hat(:,k) = X_hat(:,k)+G*(Z(k)-H_k*X_hat(:,k));
        P = P - G*H_k*P;
    end

end

function plotResults(X, X_hat, X_hat_rand_initial)
    figure;
    subplot(2,1,1);
    plot(X(1, :),'k','LineWidth',1);
    hold on;
    plot(X_hat(1, :),'b','LineWidth',1);
    hold on;
    plot(X_hat_rand_initial(1, :),'r','LineWidth',1);
    title('$X_1$ Estimation with Kalman Filter','Interpreter','latex','FontSize',14);
    xlabel('Time','Interpreter','latex');
    ylabel('Value','Interpreter','latex');
```

```
    legend('True Value','Estimated','Estimated - random initialization')

    subplot(2,1,2);
    plot(X(2, :),'k','LineWidth',1);
    hold on;
    plot(X_hat(2, :),'b','LineWidth',1);
    hold on;
    plot(X_hat_rand_initial(2, :),'r','LineWidth',1);
    title('$X_2$ Estimation with Kalman Filter','Interpreter','latex','FontSize',14);
    xlabel('Time','Interpreter','latex');
    ylabel('Value','Interpreter','latex');
    legend('True Value','Estimated','Estimated - random initialization')
end

function [features] = featureExtractor(signal)
    m = mean(signal); % Mean of the signal

    smallerThanMean = signal(signal < m);
    largerThanMean = signal(signal > m);

    % Feature 1: Average ratio of samples smaller than m to the minimum value
    minVal = min(signal);
    if minVal ~= 0
        features(1) = mean(smallerThanMean) / minVal;
    else
        features(1) = NaN; % Handle division by zero
    end

    % Feature 2: Ratio of mean to variance of samples smaller than m
    varSmallerThanMean = var(smallerThanMean);
    if varSmallerThanMean ~= 0
        features(2) = mean(smallerThanMean) / varSmallerThanMean;
```

```
    else
        features(2) = NaN; % Handle division by zero
    end

    % Feature 3: d/D ratio where d is the average difference between samples
    % larger than m and smaller than m, and D is the range of the signal
    maxVal = max(signal);
    D = maxVal - minVal;
    d = mean(largerThanMean) - mean(smallerThanMean);
    if D ~= 0
        features(3) = d / D;
    else
        features(3) = NaN; % Handle division by zero
    end
end

function [features] = featureExtractor_new(ecgSignal)

    meanECG = mean(ecgSignal);
    stdECG = std(ecgSignal);
    madECG = mean(abs(ecgSignal - meanECG));

    features = [meanECG, stdECG, madECG];
end

function [meanECG, stdECG, madECG] = extractSimpleECGFeatures(ecgSignal)
    % Simple ECG Feature Extraction Function
    % Inputs:
    %   ecgSignal - the raw ECG signal
    % Outputs:
    %   meanECG - Mean value of the ECG signal
    %   stdECG - Standard deviation of the ECG signal
```

```
    %   madECG - Mean absolute deviation of the ECG signal

    % Calculate Mean
    meanECG = mean(ecgSignal);

    % Calculate Standard Deviation
    stdECG = std(ecgSignal);

    % Calculate Mean Absolute Deviation
    madECG = mean(abs(ecgSignal - meanECG));

    % Display Results
    disp(['Mean of ECG: ', num2str(meanECG)]);
    disp(['Standard Deviation of ECG: ', num2str(stdECG)]);
    disp(['Mean Absolute Deviation of ECG: ', num2str(madECG)]);
end
```