MathTool HW6

2024-12-11

Question 1 Fitting a 2AFC psychometric function

Part a)

```
% function nll = nloglik(mu, sigma, lambda, I, T, B)
%     %mu: mean
%     %sigma: standard deviation
%     %lambda: lapse rate
%     %I: stimulus intensities
%     %T: trials per intensity
%     %B: num of correct response
%
%     p_correct = lambda / 2 + (1 - lambda) * normcdf(I, mu, sigma);
%     p_correct = max(min(p_correct, 1 - 1e-6), 1e-6);
%     likelihoods = B .* log(p_correct) + (T - B) .* log(1 - p_correct);
%     nll = -np.sum(likelihoods);
% end
```

Part b)

```
T = ones(1, 7) * 100;     %trials per intensity
I = 1:7; %stimulus intensities
lambda_true = 0.05; %lapse rate
mu_true = 4; %mean
sigma_true = 1; %standard deviation

%p correct in this case
p_correct_b = lambda_true / 2 + (1 - lambda_true) * normcdf(I, mu_true,
sigma_true);

%B: num of correct response
B = binornd(T, p_correct_b);

nll_nloglik = @(x) nloglik(x(1), x(2), x(3), I, T, B);
start_point = [2, 2, 0.05];

[params, nll] = fminsearch(nll_nloglik, start_point);
```

```
disp('Estimated parameters:');
```

```
Estimated parameters:
```

```
disp(['mu: ', num2str(params(1))]);
```

```
mu: 3.9492
```

```
disp(['sigma: ', num2str(params(2))]);
```

```
sigma: 1.0787
```

```
disp(['lambda: ', num2str(params(3))]);
```

```
lambda: 0.024181
```

```
disp('True parameters:');
```

```
True parameters:
```

```
disp(['mu: ', num2str(mu_true)]);
```

```
mu: 4
```

```
disp(['sigma: ', num2str(sigma_true)]);
```

```
sigma: 1
```

```
disp(['lambda: ', num2str(lambda_true)]);
```

```
lambda: 0.05
```

For mean and standard deviation, the estimations were close to the true value. However, the lapse rate is not very close.

Part c)

```
%from part b
p_correct_c = lambda_true / 2 + (1 - lambda_true) * normcdf(I, mu_true,
sigma_true);
B = binornd(T, p_correct_c);
nll_nloglik = @(x) nloglik(x(1), x(2), x(3), I, T, B);
start_point = [2, 2, 0.05];

%optimizations
options = optimoptions('fminunc', 'Algorithm', 'quasi-newton', 'Display',
'iter');
[params, nll, exitflag, output, grad, hessian] = fminunc(nll_nloglik,
start_point, options);
```

```
 Iteration  Func-count       f(x)        Step-size       First-order optimality
     0          4          404.309                             154
     1         12          378.015     0.00190608              81.2
     2         24          307.069        6.91627              114
     3         32          293.198            0.1              361
     4         44          274.375     0.0331341              289
     5         48          272.266              1              80.8
     6         52          266.578              1              56.6
```

```
     7          56          263.198             1            148
     8          60          260.698             1             44
     9          64          259.233             1           17.6
    10          68          258.872             1           4.51
    11          72          258.783             1           6.38
    12          80          250.368             2           15.7
    13          96          245.405        23.286             93
    14         104           244.55      0.448279             89
    15         112          244.237           0.1           80.3
    16         116          242.165             1           24.6
    17         120          239.309             1           28.4
    18         128          237.582      0.299312           10.4
    19         132          237.212             1            3.5
                                                       First-order
 Iteration  Func-count       f(x)        Step-size       optimality
    20         136          237.163             1            0.348
    21         140          237.163             1           0.0653
    22         144          237.163             1           0.0102
    23         148          237.163             1         0.000326
    24         152          237.163             1         1.34e-05

Computing finite-difference Hessian using objective function.

Local minimum found.

Optimization completed because the size of the gradient is less than
the value of the optimality tolerance.

<stopping criteria details>
```

```
mu_est = params(1);
sigma_est = params(2);
lambda_est = params(3);

%covariance matrix as the inverse of the hessian matrix
cov_matrix = inv(hessian);

%compute sd
std_mu = sqrt(cov_matrix(1, 1));
std_sigma = sqrt(cov_matrix(2, 2));
std_lambda = sqrt(cov_matrix(3, 3));

%95% CI
CI_mu = [mu_est - 1.96 * std_mu, mu_est + 1.96 * std_mu];
CI_sigma = [sigma_est - 1.96 * std_sigma, sigma_est + 1.96 * std_sigma];
CI_lambda = [lambda_est - 1.96 * std_lambda, lambda_est + 1.96 *
std_lambda];
```

```
fprintf('Estimated parameters:\n');
```

```
Estimated parameters:
```

```
fprintf('mu: %.4f, 95%% CI: [%.4f, %.4f]\n', mu_est, CI_mu(1), CI_mu(2));
```

```
mu: 4.0799, 95% CI: [3.9096, 4.2501]
```

```
fprintf('sigma: %.4f, 95%% CI: [%.4f, %.4f]\n', sigma_est, CI_sigma(1),
CI_sigma(2));
```

```
sigma: 1.0931, 95% CI: [0.8699, 1.3162]
```

```
fprintf('lambda: %.4f, 95%% CI: [%.4f, %.4f]\n', lambda_est, CI_lambda(1),
CI_lambda(2));
```

```
lambda: 0.0482, 95% CI: [0.0033, 0.0930]
```

```
fprintf('\nTrue parameters:\n');
```

```
True parameters:
```

```
fprintf('mu: %.4f, sigma: %.4f, lambda: %.4f\n', mu_true, sigma_true,
lambda_true);
```

```
mu: 4.0000, sigma: 1.0000, lambda: 0.0500
```

```
%Checking whether true parameters fall in the confidence interval
in_CI_mu = (mu_true >= CI_mu(1) && mu_true <= CI_mu(2));
in_CI_sigma = (sigma_true >= CI_sigma(1) && sigma_true <= CI_sigma(2));
in_CI_lambda = (lambda_true >= CI_lambda(1) && lambda_true <= CI_lambda(2));
fprintf('\nDo the true parameters fall within the confidence intervals?\n');
```

```
Do the true parameters fall within the confidence intervals?
```

```
fprintf('mu: %d, sigma: %d, lambda: %d\n', in_CI_mu, in_CI_sigma,
in_CI_lambda);
```

```
mu: 1, sigma: 1, lambda: 1
```

Thus, mu and sigma fall with in the confidence internal but not the lamda.

Which is similar to what we observed in part b),where the lapse rate is not close to true parameter.

Part d)

```
nBootstrap = 500;
paramsBootstrap = zeros(nBootstrap,3);

for i = 1:nBootstrap
    B_resampled = arrayfun(@(b, t) binornd(t, b ./ t, [1, 1]), B, T);
    nll_resampled = @(x) nloglik(x(1), x(2), x(3), I, T, B_resampled);
    paramsResampled = fminsearch(nll_resampled, start_point);
    paramsBootstrap(i, :) = paramsResampled;
end

bootstrap_mu = paramsBootstrap(:, 1);
bootstrap_sigma = paramsBootstrap(:, 2);
```

```
bootstrap_lambda = paramsBootstrap(:, 3);

CI_mu_bootstrap = prctile(bootstrap_mu, [2.5, 97.5]);
CI_sigma_bootstrap = prctile(bootstrap_sigma, [2.5, 97.5]);
CI_lambda_bootstrap = prctile(bootstrap_lambda, [2.5, 97.5]);

fprintf('Bootstrap confidence intervals:\n');
```

```
Bootstrap confidence intervals:
```

```
fprintf('mu: [%.4f, %.4f]\n', CI_mu_bootstrap(1), CI_mu_bootstrap(2));
```

```
mu: [3.8989, 4.2536]
```

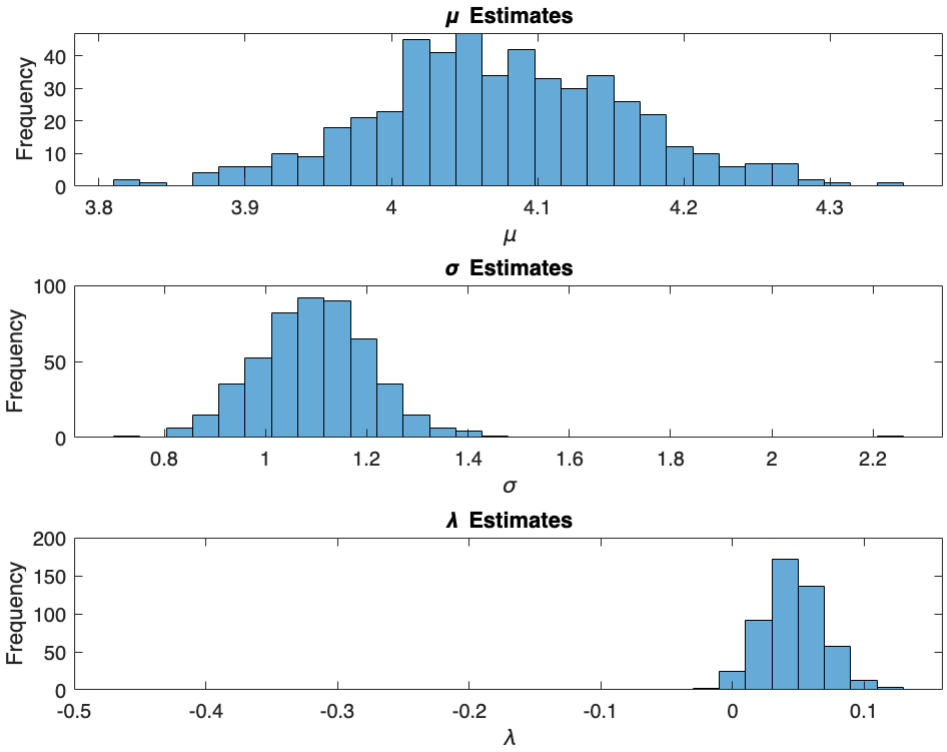
```
fprintf('sigma: [%.4f, %.4f]\n', CI_sigma_bootstrap(1),
CI_sigma_bootstrap(2));
```

```
sigma: [0.8788, 1.3227]
```

```
fprintf('lambda: [%.4f, %.4f]\n', CI_lambda_bootstrap(1),
CI_lambda_bootstrap(2));
```

```
lambda: [0.0043, 0.0922]
```

```
figure;
subplot(3, 1, 1);
histogram(bootstrap_mu, 30);
title('\mu Estimates');
xlabel('\mu');
ylabel('Frequency');

subplot(3, 1, 2);
histogram(bootstrap_sigma, 30);
title('\sigma Estimates');
xlabel('\sigma');
ylabel('Frequency');

subplot(3, 1, 3);
histogram(bootstrap_lambda, 30);
title('\lambda Estimates');
xlabel('\lambda');
ylabel('Frequency');
```




```
%To compare with precious part:
%Comparison between confidence intervals

fprintf('\nComparison of confidence intervals:\n');
```

```
Comparison of confidence intervals:
```

```
fprintf('mu:\n');
```

```
mu:
```

```
fprintf('  Hessian-based CI: [%.4f, %.4f]\n', CI_mu(1), CI_mu(2));
```

```
  Hessian-based CI: [3.9096, 4.2501]
```

```
fprintf('  Bootstrap CI:     [%.4f, %.4f]\n', CI_mu_bootstrap(1),
CI_mu_bootstrap(2));
```

```
  Bootstrap CI:     [3.8989, 4.2536]
```

```
fprintf('sigma:\n');
```

```
sigma:
```

```
fprintf('  Hessian-based CI: [%.4f, %.4f]\n', CI_sigma(1), CI_sigma(2));
```

```
  Hessian-based CI: [0.8699, 1.3162]
```

```
fprintf('  Bootstrap CI:     [%.4f, %.4f]\n', CI_sigma_bootstrap(1),
CI_sigma_bootstrap(2));
```

```
  Bootstrap CI:     [0.8788, 1.3227]
```

```
fprintf('lambda:\n');
```

```
lambda:
```

```
fprintf('  Hessian-based CI: [%.4f, %.4f]\n', CI_lambda(1), CI_lambda(2));
```

```
  Hessian-based CI: [0.0033, 0.0930]
```

```
fprintf('  Bootstrap CI:     [%.4f, %.4f]\n', CI_lambda_bootstrap(1),
CI_lambda_bootstrap(2));
```

```
  Bootstrap CI:     [0.0043, 0.0922]
```

```
%Compute significance of mu
mu_overlap_length = max(0, min(CI_mu(2), CI_mu_bootstrap(2)) -
max(CI_mu(1), CI_mu_bootstrap(1)));
mu_avg_length = (CI_mu(2) - CI_mu(1) + CI_mu_bootstrap(2) -
CI_mu_bootstrap(1)) / 2;
mu_overlap_percentage = (mu_overlap_length / mu_avg_length) * 100;
```

```
%Compute significance of sigma
sigma_overlap_length = max(0, min(CI_sigma(2), CI_sigma_bootstrap(2)) -
max(CI_sigma(1), CI_sigma_bootstrap(1)));
sigma_avg_length = (CI_sigma(2) - CI_sigma(1) + CI_sigma_bootstrap(2) -
CI_sigma_bootstrap(1)) / 2;
sigma_overlap_percentage = (sigma_overlap_length / sigma_avg_length) * 100;
```

```
%Compute significance of lambda
lambda_overlap_length = max(0, min(CI_lambda(2), CI_lambda_bootstrap(2)) -
max(CI_lambda(1), CI_lambda_bootstrap(1)));
lambda_avg_length = (CI_lambda(2) - CI_lambda(1) + CI_lambda_bootstrap(2) -
CI_lambda_bootstrap(1)) / 2;
lambda_overlap_percentage = (lambda_overlap_length / lambda_avg_length) *
100;
```

```
%Overall overlapping percentages
fprintf('\nOverlap Percentages:\n');
```

```
Overlap Percentages:
```

```
fprintf('mu: %.2f%%\n', mu_overlap_percentage);
```

```
mu: 97.97%
```

```
fprintf('sigma: %.2f%%\n', sigma_overlap_percentage);
```

```
sigma: 98.27%
```

```
fprintf('lambda: %.2f%%\n', lambda_overlap_percentage);
```

```
lambda: 98.99%
```

```
% Interpret significant overlap
fprintf('\nSignificant Overlap Assessment:\n');
```

```
Significant Overlap Assessment:
```

```
if mu_overlap_percentage >= 50
    fprintf('mu: Significant overlap.\n');
else
    fprintf('mu: Insufficient overlap.\n');
end
```

```
mu: Significant overlap.
```

```
if sigma_overlap_percentage >= 50
    fprintf('sigma: Significant overlap.\n');
else
    fprintf('sigma: Insufficient overlap.\n');
end
```

```
sigma: Significant overlap.
```

```
if lambda_overlap_percentage >= 50
    fprintf('lambda: Significant overlap.\n');
else
    fprintf('lambda: Insufficient overlap.\n');
end
```

```
lambda: Significant overlap.
```

Overall, we can see that CI with 2.5th and 97.5th percentiles significantly agrees with those from previous part.

```
%Function from Part a)
```

```
function nll = nloglik(mu, sigma, lambda, I, T, B)
    %Psychometric function
    p_correct = lambda / 2 + (1 - lambda) * normcdf(I, mu, sigma);
    p_correct = max(min(p_correct, 1 - 1e-6), 1e-6);
    likelihoods = B .* log(p_correct) + (T - B) .* log(1 - p_correct);
    nll = -sum(likelihoods);
end
```

MathTool HW6

2024-12-13

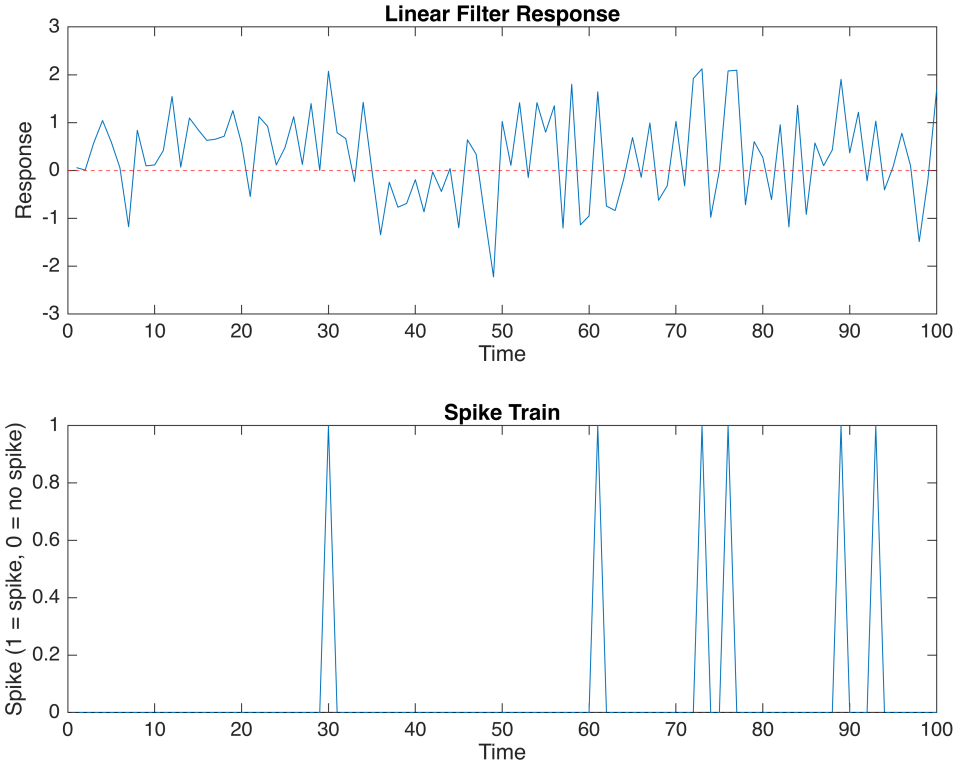
Question 2 Reverse Correlation

```
clear all; clc; close all;
addpath("hw6-files/");
```

Part a)

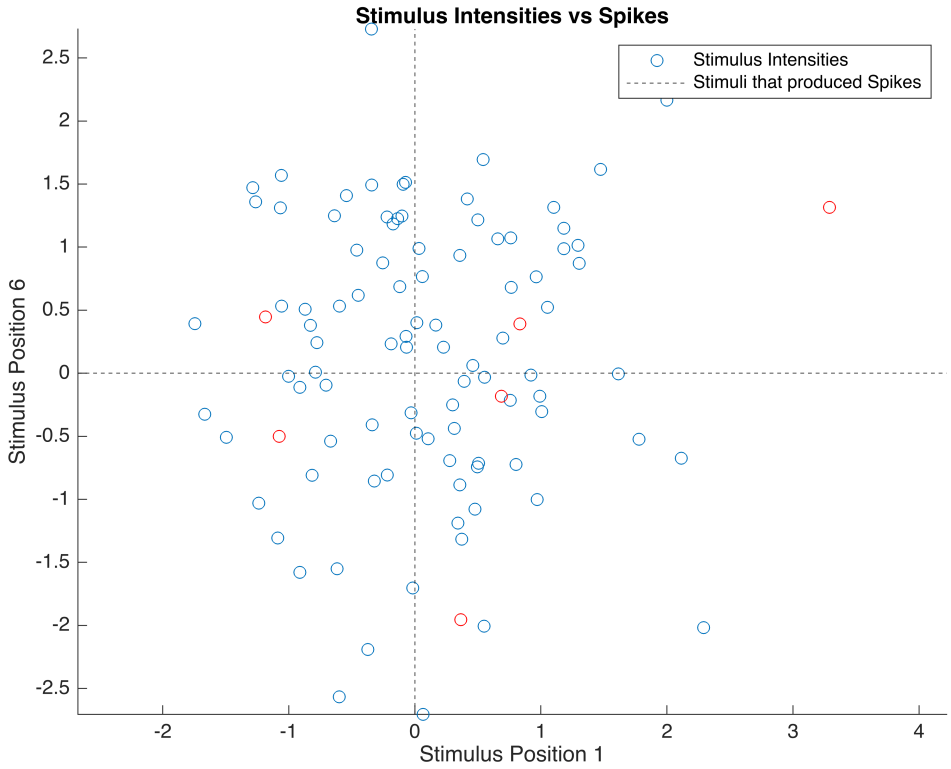
```
kernel = [1 2 1; 2 4 2; 1 2 1] / 6;
kernel_flat = kernel(:);

%Stimuli
duration = 100;
[spikes, stimuli] = runGaussNoiseExpt(kernel_flat, duration);
filter_response = stimuli * kernel_flat;
```

```
figure;
subplot(2,1,1);
plot(filter_response);
yline(0, 'r--');
title('Linear Filter Response');
xlabel('Time');
ylabel('Response');

subplot(2,1,2);
plot(spikes);
yline(0, 'k--');
title('Spike Train');
xlabel('Time');
ylabel('Spike (1 = spike, 0 = no spike)');
```



The spike tend to occur when the linear filter response is relatively peaking. Because spikes are more likely to be generated when the stimulus produces a large positive response after being filtered by the kernel, which we can interpret it as exceeding some threshold.

```
% 2D Scatter Plot
stimulus_1_6 = stimuli(:, [1, 6]);
figure;
scatter(stimulus_1_6(:,1), stimulus_1_6(:,2), 'o');
xline(0, 'k--');
yline(0, 'k--');
hold on;
spike_positions = stimuli(spikes == 1, [1, 6]);
scatter(spike_positions(:,1), spike_positions(:,2), 'ro');
axis equal;
title('Stimulus Intensities vs Spikes');
xlabel('Stimulus Position 1');
ylabel('Stimulus Position 6');
legend({'Stimulus Intensities', 'Stimuli that produced Spikes'});
```



It does look like a 2D Gaussian since the points are spread symmetrically around the origin. Also, the density points gradually decreases as moving farther away from the center.

The spikes tend to appear around moderate positive values for **Stimulus Position 1** and close-to-zero values for **Stimulus Position 6,** showing that it's aligning with the kernel.

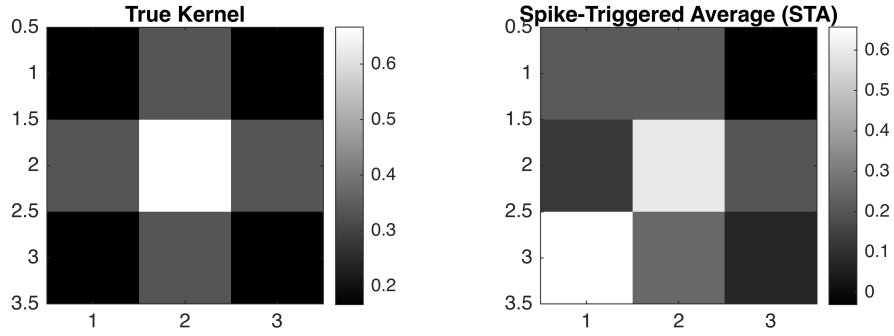
This does match our expectation, since:

- The kernel acts as a spatial filter, and spikes occur when the stimulus matches or correlates well with the kernel.
- Most of the stimulus intensities are near the center, with fewer points farther from the origin, which is because of the stimuli are drawn from a Gaussian distribution.
- Spikes in Gaussian space is well-aligned with pattern in linear filter response space.

Part b)

```
kernel = [1 2 1; 2 4 2; 1 2 1] / 6;
kernel_flat = kernel(:);
duration = 100;
[spikes, stimuli] = runGaussNoiseExpt(kernel_flat, duration);
if sum(spikes) > 0
    STA = mean(stimuli(spikes == 1, :), 1);
else
    STA = zeros(1, length(kernel_flat));
end
```

```
STA_mean=mean(STA);

STA = STA / norm(STA);
STA_matrix = reshape(STA, size(kernel));

figure;
subplot(1, 2, 1);
imagesc(kernel);
colormap gray;
colorbar;
axis equal tight;
title('True Kernel');

subplot(1, 2, 2);
imagesc(STA_matrix);
colormap gray;
colorbar;
axis equal tight;
title('Spike-Triggered Average (STA)');
```




```
similarity = corr(kernel_flat, STA');
```

```
disp(['Mean of STA: ', num2str(STA_mean)]);
```

```
Mean of STA: 0.62531
```

```
disp(['Similarity (correlation) between STA and true kernel: ',
num2str(similarity)]);
```

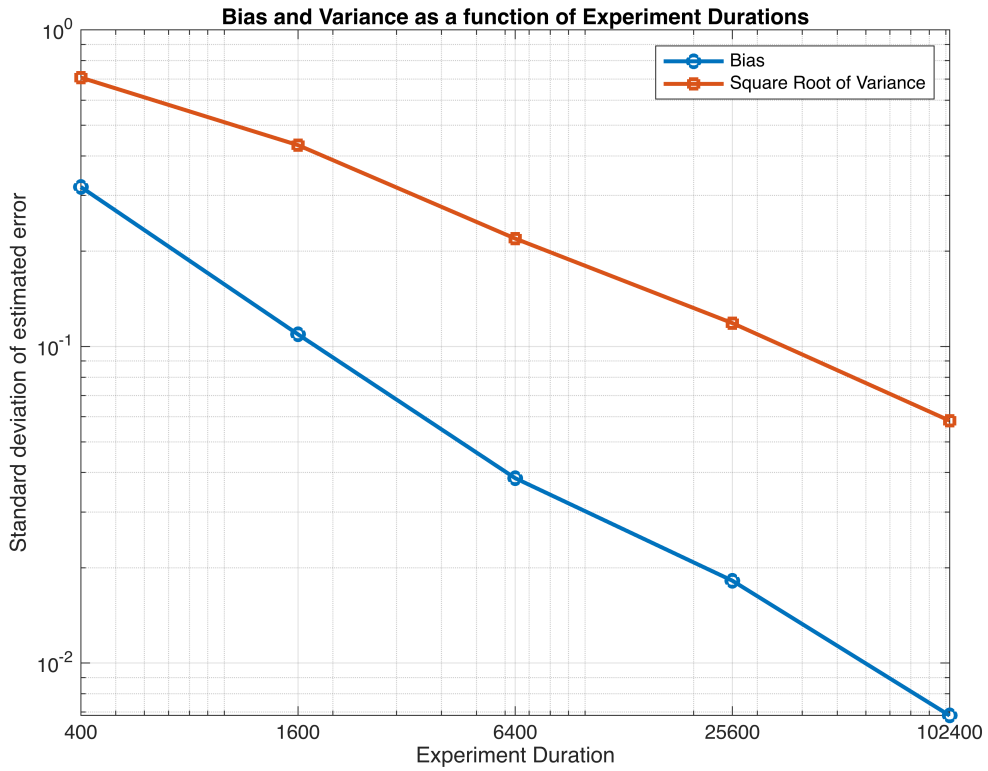
```
Similarity (correlation) between STA and true kernel: 0.44563
```

It's around 73% similar between the true kernel and the STA, with center being most activated in both.

```
durations = [100, 400, 1600, 6400, 25600, 102400];
nRuns = 100;

biases = zeros(1, length(durations));
variances = zeros(1, length(durations));

for d = 1:length(durations) %durations
    duration_b = durations(d);
    STAs = zeros(length(kernel_flat), nRuns); %empty holder

    for r = 1:nRuns %runs
        [spikes, stimuli] = runGaussNoiseExpt(kernel_flat, duration_b);

        spike_indices = find(spikes == 1);
        if isempty(spike_indices)
            STA_b = zeros(size(kernel_flat));
        else
            spike_b = length(spike_indices);
            STA_b = zeros(spike_b, length(kernel_flat));
            for i = 1:spike_b
                if spike_indices(i) < 6
                    STA_b(i, :) = mean(stimuli(1:spike_indices(i), :), 1);
                else
                    STA_b(i, :) =
mean(stimuli(spike_indices(i)-5:spike_indices(i), :), 1);
                end
            end
            STA_b = mean(STA_b, 1);
        end

        STA_b = STA_b(:) / norm(STA_b);
        STAs(:, r) = STA_b;
    end %runs

    %mean
    mean_STA = mean(STAs, 2);

    %bias
    bias_kernel = mean_STA - kernel_flat;
    biases(d) = norm(bias_kernel);
```

```
    %variance
    differences = STAs - mean_STA;
    variances(d) = mean(sum(differences.^2, 1));
end %durations
```

```
figure;
loglog(durations, biases, 'o-', 'LineWidth', 2);
hold on;
loglog(durations, sqrt(variances), 's-', 'LineWidth', 2);
legend('Bias', 'Square Root of Variance');
xlabel('Experiment Duration');
ylabel('Standard deviation of estimated error');
title('Bias and Variance as a function of Experiment Durations');
grid on;

xticks(durations);
xticklabels(arrayfun(@num2str, durations, 'UniformOutput', false));
```



From the graph we can see that **bias** decreases proportionally as the experiment duration increases. Longer durations allow for better averaging of the stimuli that caused spikes, leading to an STA that better approximates the true kernel. This indicates that STA estimate becomes more accurate as more data is gathered, which is overall decreasing estimated error.

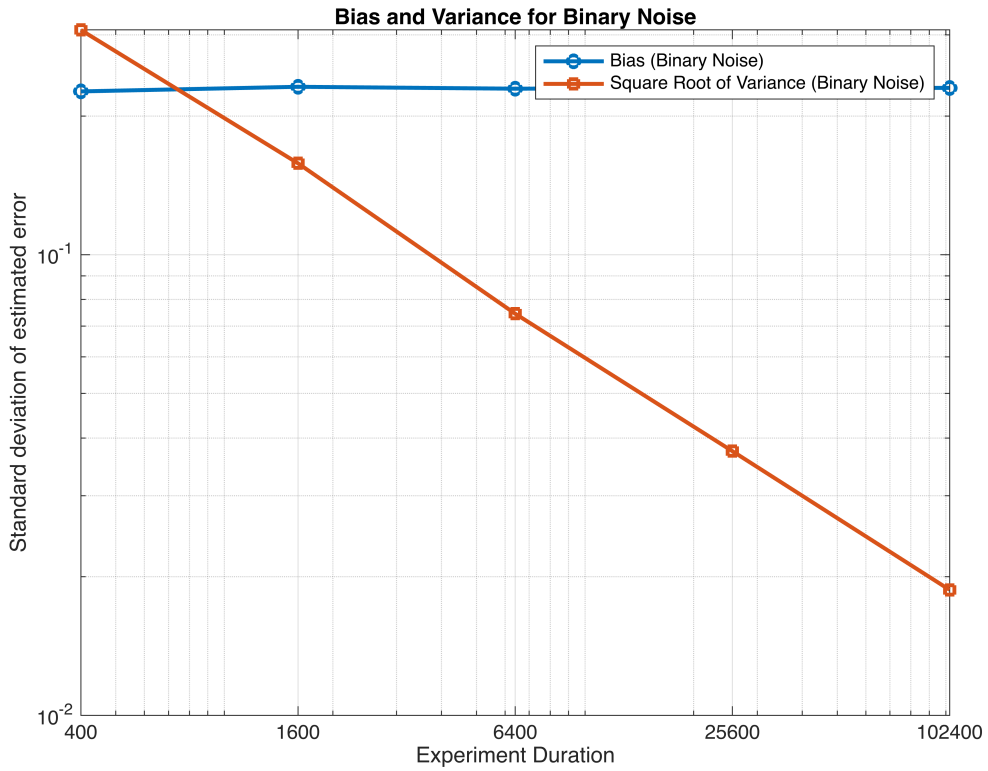
The **square root of the variance** also decreases as the experiment duration increases. Increasing the number of trials reduces the effect of random variability in STA. This reflects the fact that random noise in the STA estimates reduces with more data.

Overall, we can caonclude that both systematic and random errors reduce with increasing sample size.

Part c)

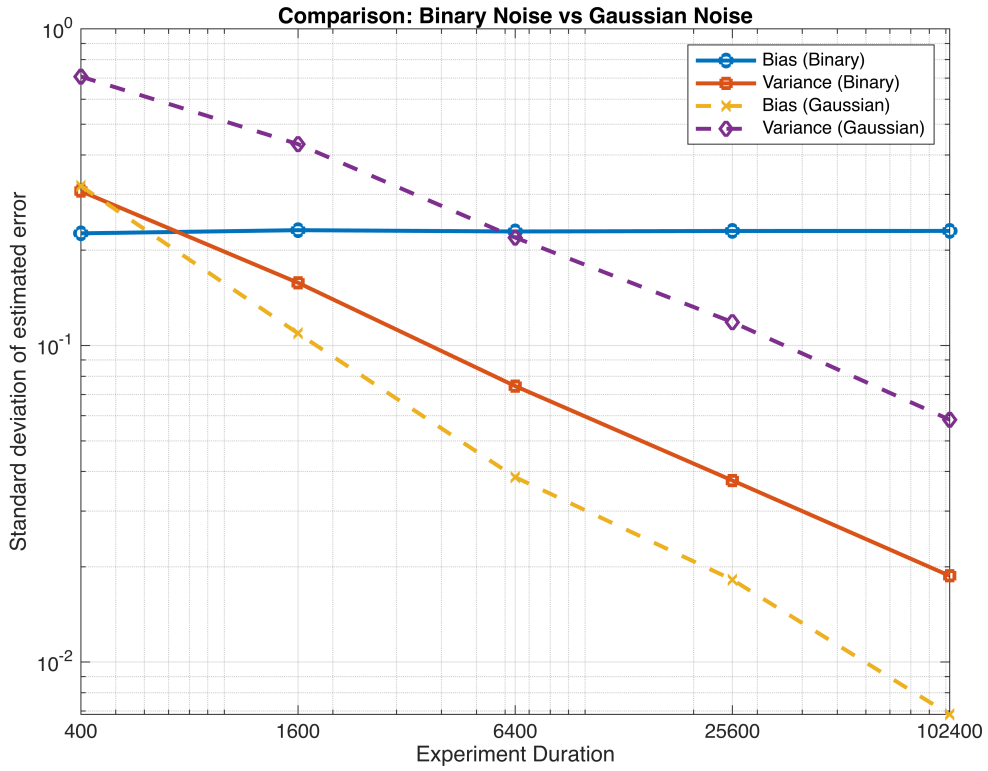
```
kernel_c = [1 2 1; 2 4 2; 1 2 1] / 6;
kernel_flat_c = kernel_c(:);

durations = [100, 400, 1600, 6400, 25600, 102400];
nRuns = 100;

biases_bin = zeros(1, length(durations));
variances_bin = zeros(1, length(durations));

for dIdx = 1:length(durations)
    duration_c = durations(dIdx);
    STAs_bin = zeros(length(kernel_flat_c), nRuns);

    for r = 1:nRuns
        [spikes, stimuli] = runBinNoiseExpt(kernel_flat_c, duration_c);

        spike_indices = find(spikes == 1);
        if isempty(spike_indices)
            STA_c = zeros(size(kernel_flat_c));
        else
            STA_c = mean(stimuli(spike_indices, :), 1);
            STA_c = STA_c(:);
        end

        STA_c = STA_c / norm(STA_c);
        STAs_bin(:, r) = STA_c;
    end

    mean_STA_bin = mean(STAs_bin, 2);

    %bias
    bias_kernel_bin = mean_STA_bin - kernel_flat_c;
    biases_bin(dIdx) = norm(bias_kernel_bin);

    %variance
    diffs_bin = STAs_bin - mean_STA_bin;
    variances_bin(dIdx) = mean(sum(diffs_bin.^2, 1));
end
```

```
figure;
loglog(durations, biases_bin, 'o-', 'LineWidth', 2);
hold on;
loglog(durations, sqrt(variances_bin), 's-', 'LineWidth', 2);
legend('Bias (Binary Noise)', 'Square Root of Variance (Binary Noise)');
xlabel('Experiment Duration');
ylabel('Standard deviation of estimated error');
title('Bias and Variance for Binary Noise');
grid on;

xticks(durations);
xticklabels(arrayfun(@num2str, durations, 'UniformOutput', false));
```




```
% Compare with Gaussian noise (overlaid plot)
figure;
loglog(durations, biases_bin, 'o-', 'LineWidth', 2);
hold on;
loglog(durations, sqrt(variances_bin), 's-', 'LineWidth', 2);
loglog(durations, biases, 'x--', 'LineWidth', 2);  %G
loglog(durations, sqrt(variances), 'd--', 'LineWidth', 2); %G
legend('Bias (Binary)', 'Variance (Binary)', 'Bias (Gaussian)', 'Variance
(Gaussian)');
xlabel('Experiment Duration');
ylabel('Standard deviation of estimated error');
title('Comparison: Binary Noise vs Gaussian Noise');
grid on;
```

```
xticks(durations);
xticklabels(arrayfun(@num2str, durations, 'UniformOutput', false));
```



For binary noise, since it's discrete of binary stimuli restricts their ability to approximate the kernel's continuous profile, so it has a fixed systematic error (representing relatively constant bias). On the otherhand, Gaussian noise is continous, so it introduces more variability.

For variance, the binary noise exhibits lower variance compared to Gaussian noise, especially for short experiment durations, because binary stimuli have limited variability. Gaussian noise has higher initial variance but reduces both bias and variance more effectively with increasing data.

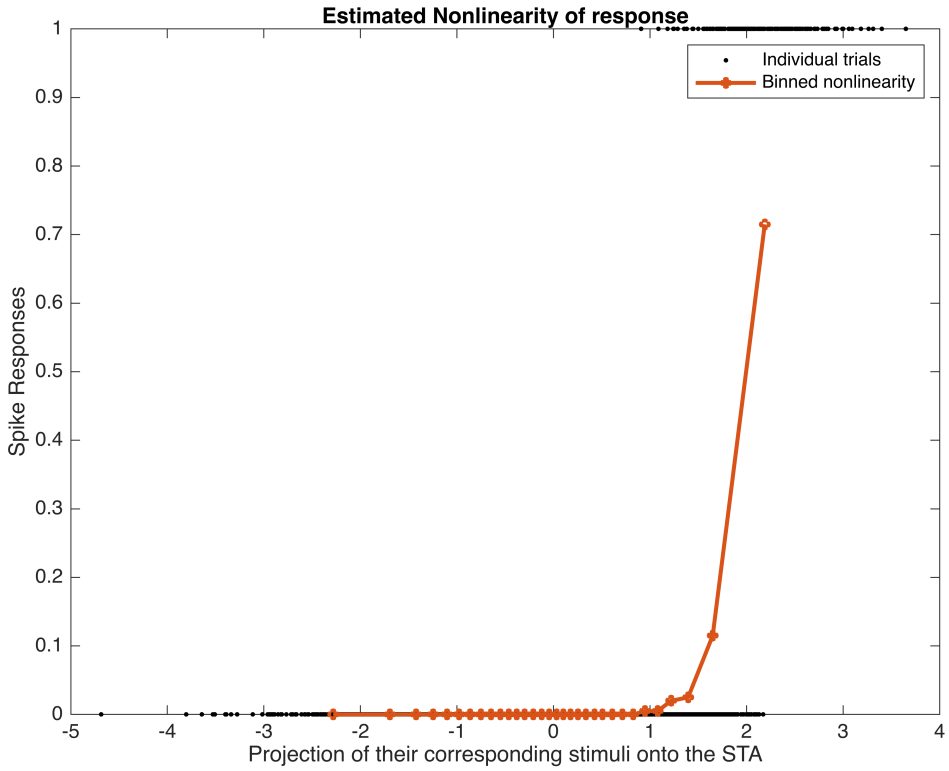
Overall, we can see that binary noise shows a constant bias due to its discrete nature, while Gaussian noise reduces bias with more data. Binary noise has lower variance initially but both noise types reduce variance with more data.

Afterall, I would say that Gaussian noise is better for capturing the true kernel with enough data, while binary noise provides more stable but systematically **biased** estimates.

Part d)

```
duration = 6400;
kernel_d = [1 2 1; 2 4 2; 1 2 1] / 6;
kernel_flat_d = kernel_d(:);

[spikes, stimuli] = runGaussNoiseExpt(kernel_flat_d, duration);
```

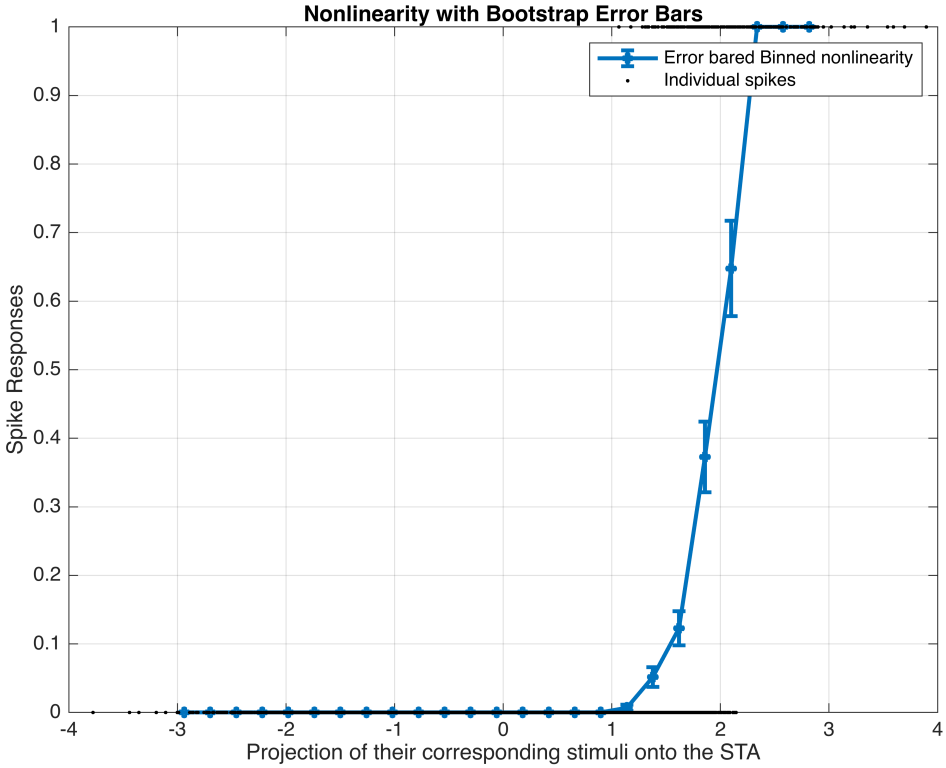
```
%Single run STA
spike_indices = find(spikes == 1);
if ~isempty(spike_indices)
    STA_d = mean(stimuli(spike_indices, :), 1);
else
    STA_d = zeros(1, length(kernel_flat_d));
end
STA_d = STA_d(:) / norm(STA_d);

%projection
projections = stimuli * STA_d;

[projectionsNew, sort_indices] = sort(projections, 'descend');
spikesNew = spikes(sort_indices);


sizeBin = 200;
nBins = duration / sizeBin;
meanProjection = zeros(nBins, 1);
meanSpike = zeros(nBins, 1);

%mean projection values and spike probabilities in each bin
for bin = 1:nBins
    bin_range = (bin-1)*sizeBin + 1 : bin*sizeBin;
    meanProjection(bin) = mean(projectionsNew(bin_range));
    meanSpike(bin) = mean(spikesNew(bin_range));
end


figure;
plot(projectionsNew, spikesNew, 'k.', 'MarkerSize', 8);
hold on;
plot(meanProjection, meanSpike, 'o-', 'LineWidth', 2, 'MarkerSize', 4);

xlabel('Projection of their corresponding stimuli onto the STA');
ylabel('Spike Responses');
title('Estimated Nonlinearity of response');
legend('Individual trials', 'Binned nonlinearity');
```



Part e)

```
duration_e = 6400;
kernel_e = [1 2 1; 2 4 2; 1 2 1] / 6;
kernel_flat_e = kernel_e(:);

[spikes, stimuli] = runGaussNoiseExpt(kernel_flat_e, duration_e);

spike_indices = find(spikes == 1);
STA_e = zeros(1, length(kernel_flat_e));
if ~isempty(spike_indices)
    STA_e = mean(stimuli(spike_indices, :), 1);
end
STA_e = STA_e(:) / norm(STA_e);

projections_e = stimuli * STA_e;

nBins_e = 32;
binEdges = linspace(min(projections_e), max(projections_e), nBins_e + 1);
binCenters = (binEdges(1:end-1) + binEdges(2:end)) / 2;

mean_spike_in_bin = arrayfun(@(b) ...
    mean(spikes((projections_e >= binEdges(b)) & (projections_e <
binEdges(b+1)))), ...
    1:nBins_e)';
```

```
nbootstrap = 100;
bootstrap_spike_bins = zeros(nbootstrap, nBins_e);

for i = 1:nbootstrap
    rand_indices = randi(duration_e, [duration_e, 1]);
    proj_resampled = projections_e(rand_indices);
    spikes_resampled = spikes(rand_indices);

    bootstrap_spike_bins(i, :) = arrayfun(@(b) ...
        mean(spikes_resampled((proj_resampled >= binEdges(b)) &
(proj_resampled < binEdges(b+1)))), ...
        1:nBins_e);
end


meanBootstrap = mean(bootstrap_spike_bins, 1);
sdBootstrap = std(bootstrap_spike_bins, 0, 1);

figure;
errorbar(binCenters, meanBootstrap, sdBootstrap, 'o-', 'LineWidth', 2,
'MarkerSize', 4);
hold on;
plot(projections_e, spikes, 'k.', 'MarkerSize', 5);
xlabel('Projection of their corresponding stimuli onto the STA');
ylabel('Spike Responses');
title('Nonlinearity with Bootstrap Error Bars');
legend('Error bared Binned nonlinearity', 'Individual spikes');
grid on;
```


Nonlinearity with Bootstrap Error Bars
Error bared Binned nonlinearity
Individual spikes
Spike Responses
Projection of their corresponding stimuli onto the STA
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-4
-3
-2
-1
0
1
2
3
4

MathTool HW6

2024-12-20

Question 3 Classification (decision) in a 2-dimensional space

```
clear all; clc; close all;
addpath("hw6-files/");
load('fisherData.mat');
```

Part a)

```
% Load the data
load fisherData.mat

%means and discriminant vector
mean1 = mean(data1, 1);
mean2 = mean(data2, 1);
discriminant = mean2 - mean1;
discriminant_unit = discriminant / norm(discriminant);

%for plotting
x_min = min([data1(:,1); data2(:,1)]) - 1;
x_max = max([data1(:,1); data2(:,1)]) + 1;
y_min = min([data1(:,2); data2(:,2)]) - 1;
y_max = max([data1(:,2); data2(:,2)]) + 1;
[x, y] = meshgrid(linspace(x_min, x_max, 100), linspace(y_min, y_max, 100));
grid_points = [x(:), y(:)];

%Gaussian likelihood
pdf1 = exp(-0.5 * sum((grid_points - mean1).^2, 2)) / (2 * pi);
pdf2 = exp(-0.5 * sum((grid_points - mean2).^2, 2)) / (2 * pi);
pdf1_image = reshape(pdf1, size(x));
pdf2_image = reshape(pdf2, size(x));

%do binary classification
binary_image = pdf2_image > pdf1_image;

figure;
imagesc([x_min, x_max], [y_min, y_max], binary_image);
set(gca, 'YDir', 'normal');
hold on;
scatter(data1(:,1), data1(:,2), 'r', 'filled');
scatter(data2(:,1), data2(:,2), 'b', 'filled');
colormap([1 0.8 0.8; 0.8 0.8 1]);
```

HW 6 Part a) Math

$$\begin{cases} \text{class 1 (dogs)} : x \sim N(\mu_1, \sigma^2 I) \\ \text{class 2 (cats)} : x \sim N(\mu_2, \sigma^2 I) \end{cases}$$

→ covariance matrices

⇒ Gaussian probability density function

class 1: $p(x|\text{class 1}) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{1}{2\sigma^2}\|x-\mu_1\|^2\right)$

class 2: $p(x|\text{class 2}) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{1}{2\sigma^2}\|x-\mu_2\|^2\right)$

⇒ To make decisions, we need to see:

$$-\frac{1}{2\sigma^2}\|x-\mu_1\|^2 \text{ comparing to } -\frac{1}{2\sigma^2}\|x-\mu_2\|^2$$

$$\therefore \|x-\mu_1\|^2 \text{ v.s. } \|x-\mu_2\|^2$$

if $\|x-\mu_1\|^2 < \|x-\mu_2\|^2$, then <u>class 1</u>.

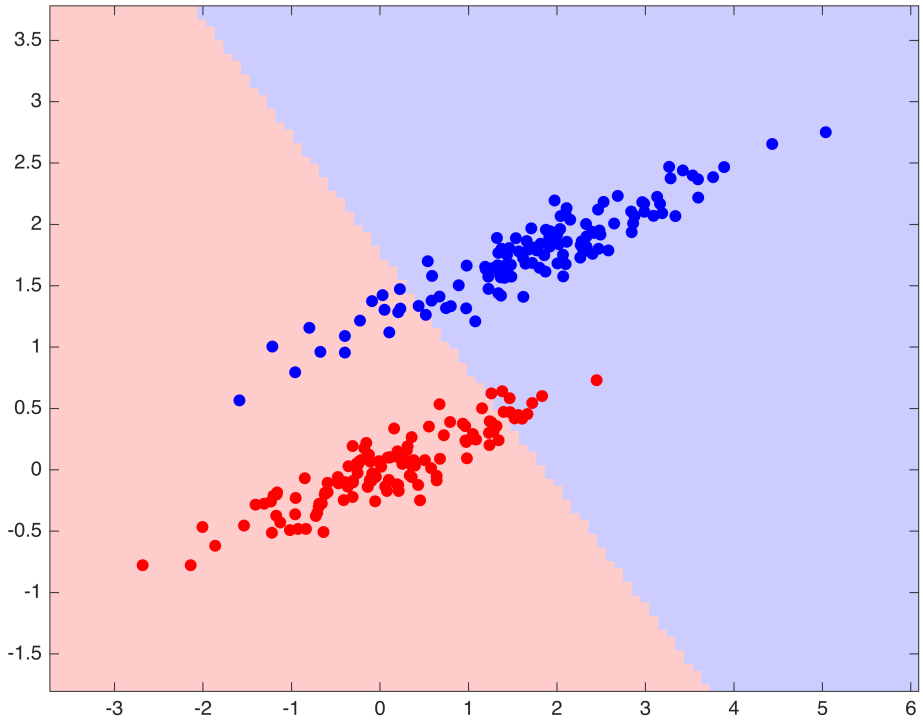
$$\Rightarrow \|x-\mu_1\|^2 = x^T x - 2x^T\mu_1 + \mu_1^T\mu_1$$

$$\|x-\mu_2\|^2 = x^T x - 2x^T\mu_2 + \mu_2^T\mu_2$$

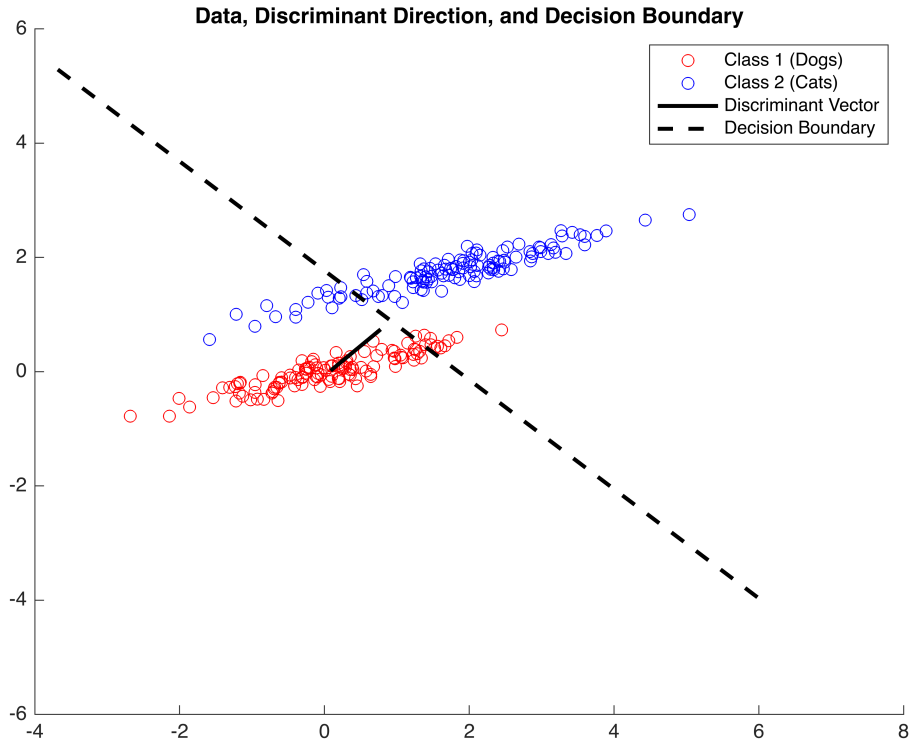
$$\therefore 2x^T(\mu_2-\mu_1) = \mu_2^T\mu_2 - \mu_1^T\mu_1$$

$$x^T(\mu_2-\mu_1) = \frac{\mu_2^T\mu_2 - \mu_1^T\mu_1}{2}$$

$$x^T \cdot \text{discriminant} = \frac{\mu_2^T\mu_2 - \mu_1^T\mu_1}{2}$$



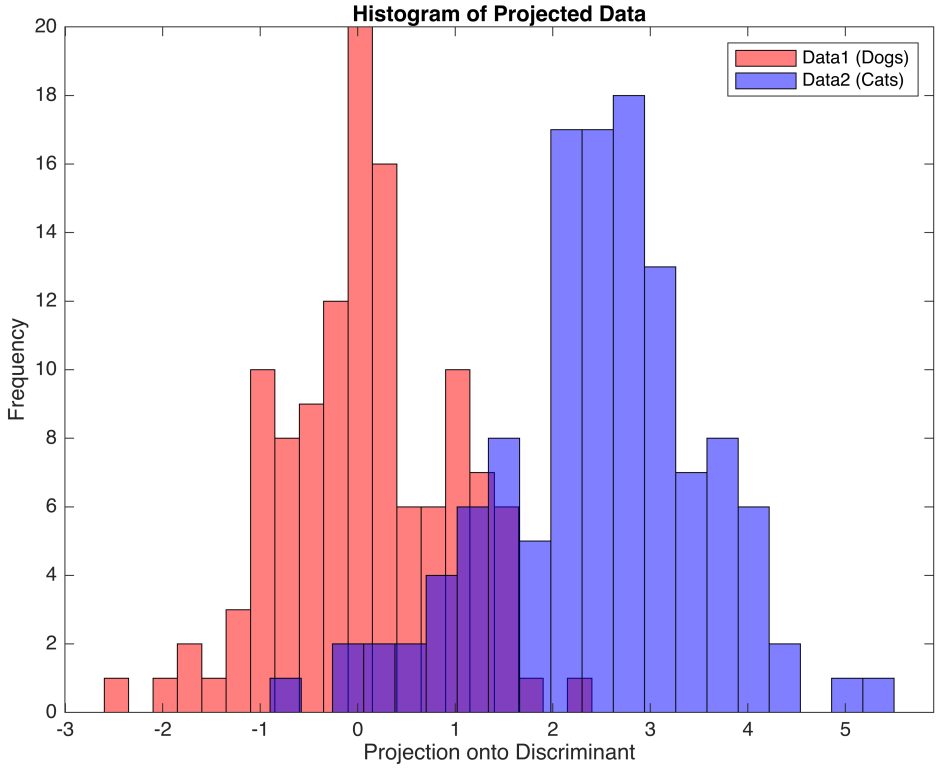

```
thresh = (sum(mean2.^2) - sum(mean1.^2)) / 2;
x_range = linspace(x_min, x_max, 100);
y_range = (thresh - discriminant(1) * x_range) / discriminant(2);
figure;
scatter(data1(:,1), data1(:,2), 'ro'); hold on;
scatter(data2(:,1), data2(:,2), 'bo');
plot([mean1(1), mean1(1) + discriminant_unit(1)], [mean1(2), mean1(2) +
discriminant_unit(2)], 'k-', 'LineWidth', 2);
plot(x_range, y_range, 'k--', 'LineWidth', 2);
title('Data, Discriminant Direction, and Decision Boundary');
legend('Class 1 (Dogs)', 'Class 2 (Cats)', 'Discriminant Vector', 'Decision
Boundary');
```




```
%calculate accuracy
proj_data1 = data1 * discriminant_unit';
proj_data2 = data2 * discriminant_unit';
threshold = (mean(proj_data1) + mean(proj_data2)) / 2;
correct1 = sum(proj_data1 < threshold);
correct2 = sum(proj_data2 > threshold);
accuracy = (correct1 + correct2) / (size(data1, 1) + size(data2, 1));
fprintf('Classification accuracy: %.2f%%\n', accuracy * 100);
```

```
Classification accuracy: 87.92%
```

```
%histograms
figure;
histogram(proj_data1, 20, 'FaceColor', 'r', 'FaceAlpha', 0.5);
hold on;
histogram(proj_data2, 20, 'FaceColor', 'b', 'FaceAlpha', 0.5);
legend('Data1 (Dogs)', 'Data2 (Cats)');
xlabel('Projection onto Discriminant');
ylabel('Frequency');
title('Histogram of Projected Data');
hold off;
```



From the accuracy of classification, we can tell that 87.92% of the points are correctly classified by this classifier.

Part b)

```
N1 = size(data1, 1);
N2 = size(data2, 1);

mean1 = mean(data1, 1);
mean2 = mean(data2, 1);

%calculate scatter matrix
S1 = cov(data1) * (size(data1, 1) - 1);
S2 = cov(data2) * (size(data2, 1) - 1);
Sw = S1 + S2; %within-class scatter matrix

% Fisher's Linear Discriminant
diff_means = (mean2 - mean1)';
discriminant = Sw \ diff_means; %inv(Sw) * (mean2 - mean1)
discriminant_unit = discriminant / norm(discriminant);

x_min = min([data1(:,1); data2(:,1)]) - 1;
x_max = max([data1(:,1); data2(:,1)]) + 1;
y_min = min([data1(:,2); data2(:,2)]) - 1;
y_max = max([data1(:,2); data2(:,2)]) + 1;
```

HW 6 Part b) Math

Fisher's linear discriminant

class 1: $x \sim N(\mu_1, \Sigma)$

$$S_1 = \sum_{i=1}^{N_1} (x_i - \mu_1)(x_i - \mu_1)^T \quad \text{(within class)}$$

class 2: $x \sim N(\mu_2, \Sigma)$

$$S_2 = \sum_{j=1}^{N_2} (x_j - \mu_2)(x_j - \mu_2)^T \quad \text{(within class)}$$

$$S_w = S_1 + S_2$$

$$\Rightarrow \quad S_b = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \quad \text{(between-class)}$$

$$\Rightarrow \quad \text{To maximize: } J(w) = \frac{w^T S_b w}{w^T S_w w}$$

$$w = S_w^{-1}(\mu_2 - \mu_1)$$

$\Rightarrow$ when data are Gaussian with same covariance matrix Σ, log-likelihood ratio:

$$\ln \frac{P(x|\text{class 2})}{P(x|\text{class 1})} = x^T \Sigma^{-1}(\mu_2 - \mu_1) - \frac{1}{2}(\mu_2^T \Sigma^{-1} \mu_2 - \mu_1^T \Sigma^{-1} \mu_1)$$

threshold

$\therefore$ if $w^T x > b$, then point to class 2.

⇒ Fisher's solution maximizes the same linear discriminant direction, making it equivalent to ML.

For covariance matrix:

$$\Sigma_{data} = \frac{S_1 + S_2}{N_1 + N_2}$$

Here, the pooled covariance matrix is:

$$\Sigma_{Pooled} = \frac{(N_1 - 1)S_1 + (N_2 - 1)S_2}{N_1 + N_2 - 2},$$

optimal direction of w would be: $w = \Sigma_{Pooled}^{-1}(\mu_2 - \mu_1)$

⇒ Doing SVD for the covariance matrix:

$$\Sigma_{Pooled} = USV^T$$

$$\therefore \Sigma_{Pooled}^{-1} = VS^{-1}U^T$$

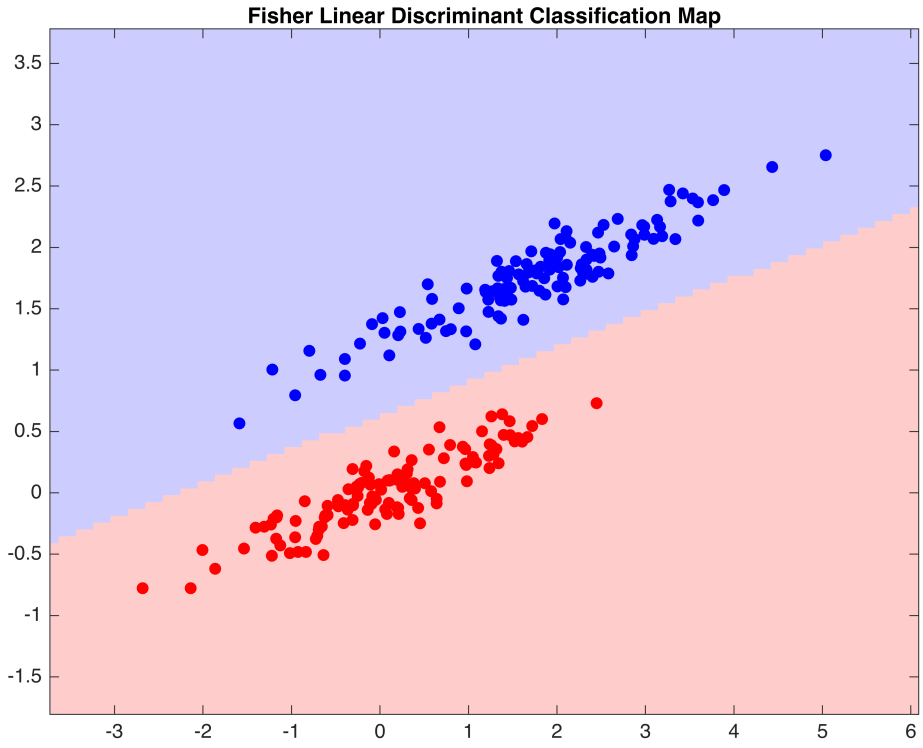
→ diagnol matrix

⇒ Given: $\Sigma_{Pooled} w = (\mu_2 - \mu_1)$

$$\therefore w = VS^{-1}U^T(\mu_2 - \mu_1)$$

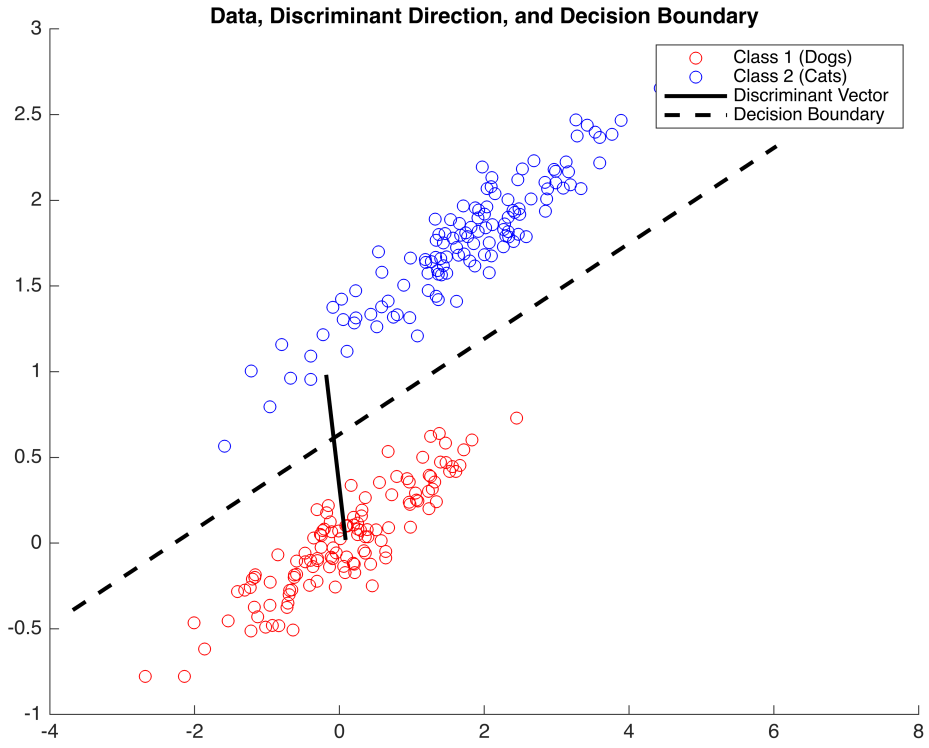
⇒ $$\text{threshold} = \frac{\mu_2^T x_2 - \mu_1^T x_1}{2}$$

```
[x, y] = meshgrid(linspace(x_min, x_max, 100), linspace(y_min, y_max, 100));
grid_points = [x(:), y(:)];

projections = grid_points * discriminant;
threshold = 0.5 * (mean1 * discriminant + mean2 * discriminant); %b
binary_image = reshape(projections > threshold, size(x));

figure;
imagesc([x_min, x_max], [y_min, y_max], binary_image);
set(gca, 'YDir', 'normal');
hold on;
scatter(data1(:,1), data1(:,2), 'r', 'filled');
scatter(data2(:,1), data2(:,2), 'b', 'filled');
colormap([1 0.8 0.8; 0.8 0.8 1]);
title('Fisher Linear Discriminant Classification Map');
```




```
x_range = linspace(x_min, x_max, 100);
y_range = (threshold - discriminant(1) * x_range) / discriminant(2);
figure;
scatter(data1(:,1), data1(:,2), 'ro'); hold on;
scatter(data2(:,1), data2(:,2), 'bo');
plot([mean1(1), mean1(1) + discriminant_unit(1)], [mean1(2), mean1(2) +
discriminant_unit(2)], 'k-', 'LineWidth', 2);
plot(x_range, y_range, 'k--', 'LineWidth', 2);
title('Data, Discriminant Direction, and Decision Boundary');
```

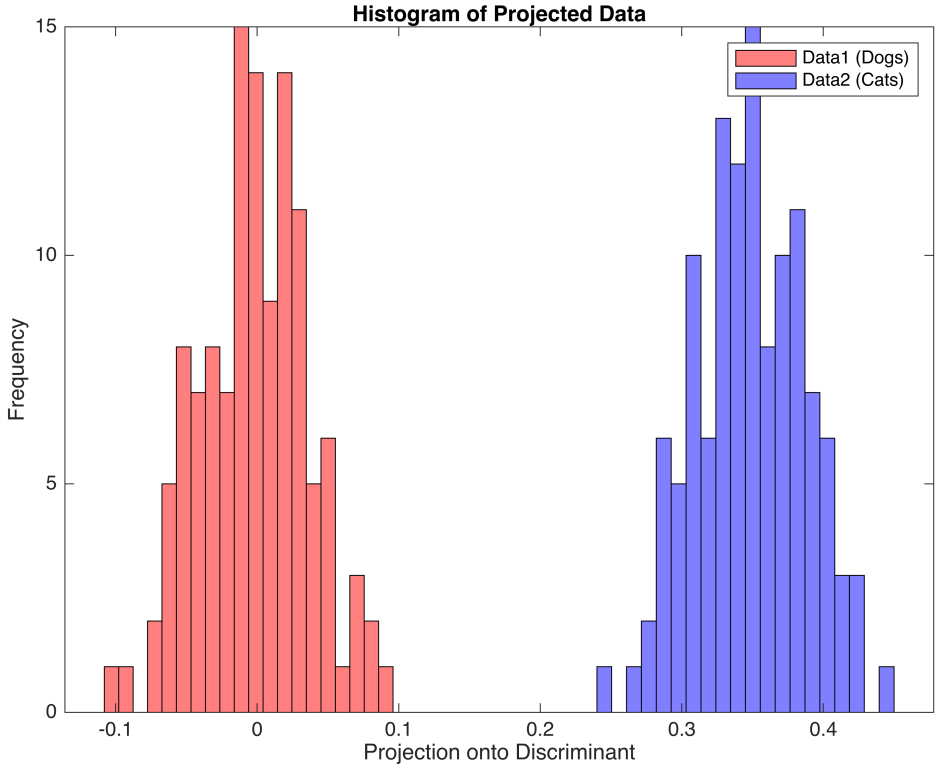
```
legend('Class 1 (Dogs)', 'Class 2 (Cats)', 'Discriminant Vector', 'Decision
Boundary');
```




```
proj_data1 = data1 * discriminant;
proj_data2 = data2 * discriminant;
correct1 = sum(proj_data1 < threshold); %go to class 1
correct2 = sum(proj_data2 > threshold); %go to class 2
accuracy = (correct1 + correct2) / (size(data1, 1) + size(data2, 1));
fprintf('Classification accuracy using FLD: %.2f%%\n', accuracy * 100);
```

```
Classification accuracy using FLD: 100.00%
```

```
figure;
histogram(proj_data1, 20, 'FaceColor', 'r', 'FaceAlpha', 0.5);
hold on;
histogram(proj_data2, 20, 'FaceColor', 'b', 'FaceAlpha', 0.5);
legend('Data1 (Dogs)', 'Data2 (Cats)');
xlabel('Projection onto Discriminant');
ylabel('Frequency');
title('Histogram of Projected Data');
hold off;
```

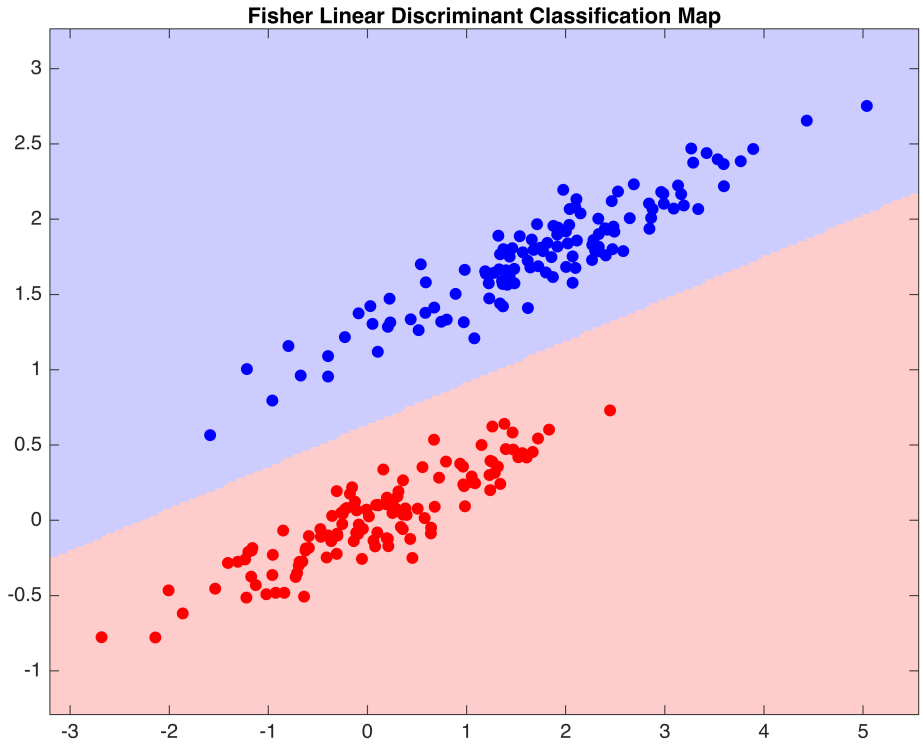


Part b) using SVD

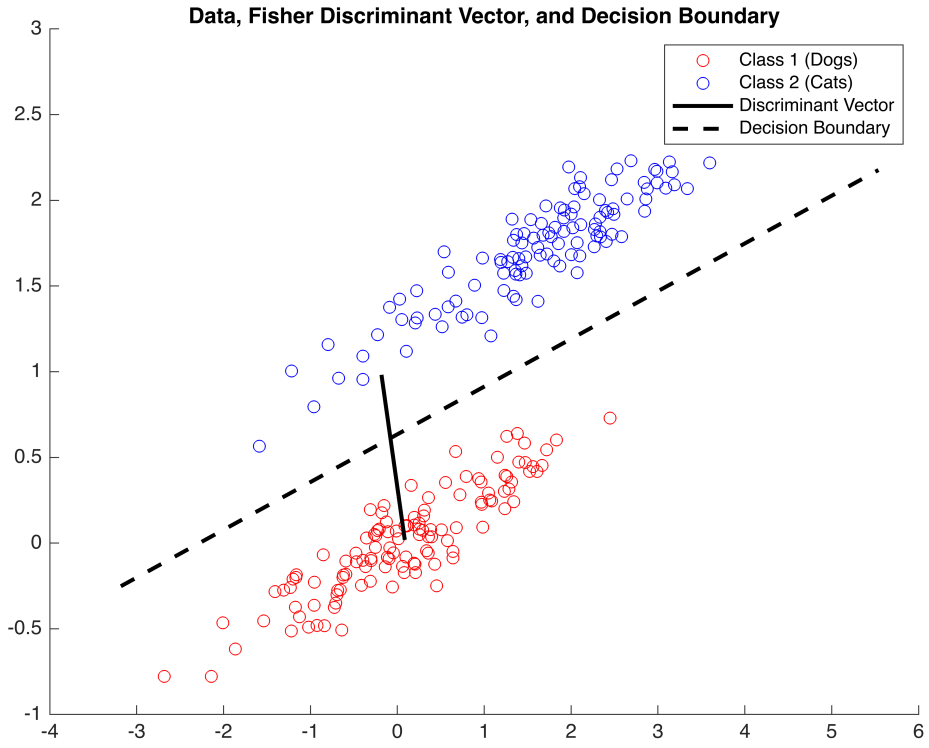
```
pooledCovariance = ((N1 - 1) * S1 + (N2 - 1) * S2) / (N1 + N2 - 2);
[U, S, V] = svd(pooledCovariance);
S_inv = diag(1 ./ diag(S));
discriminant_svd = (V * S_inv * U') * diff_means;
discriminant_svd = discriminant_svd';
discriminant_unit = discriminant_svd / norm(discriminant_svd);
x1 = (V * S_inv * U') * mean1';
x2 = (V * S_inv * U') * mean2';
threshold_svd = (mean2 * x2 - mean1 * x1) / 2;

x_min = min([data1(:, 1); data2(:, 1)]) - 0.5;
x_max = max([data1(:, 1); data2(:, 1)]) + 0.5;
y_min = min([data1(:, 2); data2(:, 2)]) - 0.5;
y_max = max([data1(:, 2); data2(:, 2)]) + 0.5;
[X, Y] = meshgrid(linspace(x_min, x_max, 200), linspace(y_min, y_max, 200));
grid_points = [X(:), Y(:)];
```

```
invPooledCov = V * S_inv * U';
constant = 1 / (2 * pi * sqrt(det(pooledCovariance)));
pdf1 = constant * exp(-0.5 * sum((grid_points - mean1) * invPooledCov .*
(grid_points - mean1), 2));
pdf2 = constant * exp(-0.5 * sum((grid_points - mean2) * invPooledCov .*
(grid_points - mean2), 2));
```

```
pdf1_image = reshape(pdf1, size(X));
pdf2_image = reshape(pdf2, size(X));
binary_image = pdf2_image > pdf1_image; % Classification

figure;
imagesc([x_min, x_max], [y_min, y_max], binary_image);
set(gca, 'YDir', 'normal');
hold on;
scatter(data1(:, 1), data1(:, 2), 'ro', 'filled');
scatter(data2(:, 1), data2(:, 2), 'bo', 'filled');
title('Fisher Linear Discriminant Classification Map');
colormap([1 0.8 0.8; 0.8 0.8 1]);
```



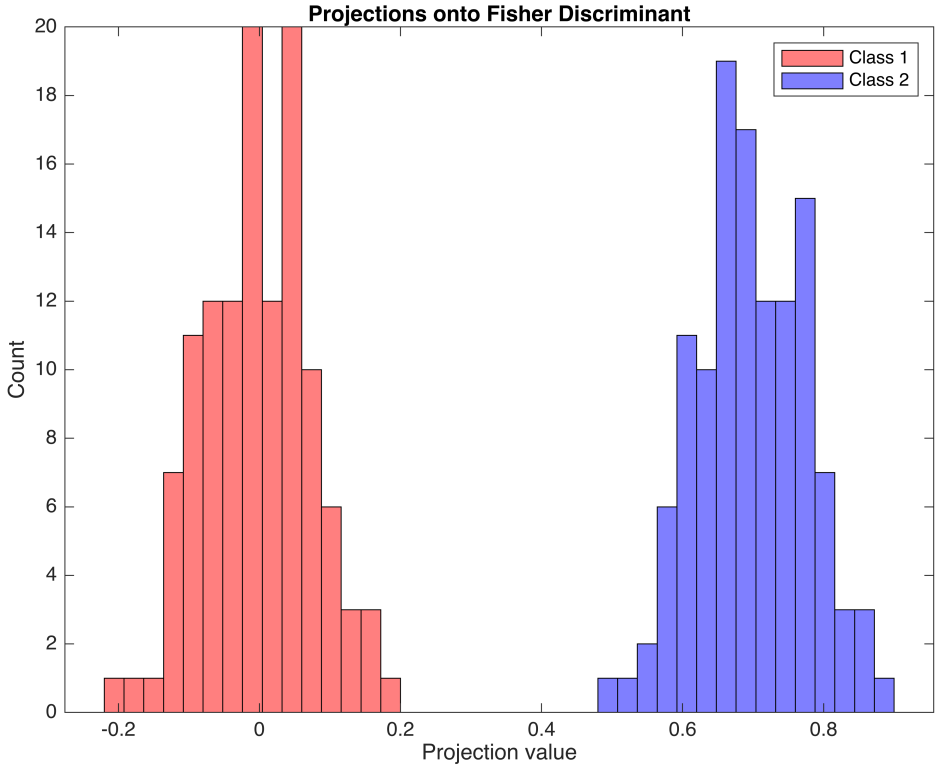

```
x_range = linspace(x_min, x_max, 100);
y_range = (threshold_svd - discriminant_svd(1) * x_range) /
discriminant_svd(2);
figure;
scatter(data1(:, 1), data1(:, 2), 'ro'); hold on;
scatter(data2(:, 1), data2(:, 2), 'bo');
plot([mean1(1), mean1(1) + discriminant_unit(1)], [mean1(2), mean1(2) +
discriminant_unit(2)], 'k-', 'LineWidth', 2); % Discriminant vector
plot(x_range, y_range, 'k--', 'LineWidth', 2); % Decision boundary
title('Data, Fisher Discriminant Vector, and Decision Boundary');
legend('Class 1 (Dogs)', 'Class 2 (Cats)', 'Discriminant Vector', 'Decision
Boundary');
```




```
proj_data1 = data1 * discriminant_svd'; % Projection for class 1
proj_data2 = data2 * discriminant_svd'; % Projection for class 2
correct1 = sum(proj_data1 < threshold); % Correctly classified as class 1
correct2 = sum(proj_data2 > threshold); % Correctly classified as class 2
accuracy = (correct1 + correct2) / (N1 + N2);
fprintf('Classification accuracy using FLD: %.2f%%\n', accuracy * 100);
```

```
Classification accuracy using FLD: 99.58%
```

```
figure;
histogram(proj_data1, 15, 'FaceColor', 'r', 'FaceAlpha', 0.5);
hold on;
histogram(proj_data2, 15, 'FaceColor', 'b', 'FaceAlpha', 0.5);
title('Projections onto Fisher Discriminant');
legend('Class 1', 'Class 2');
xlabel('Projection value');
ylabel('Count');
hold off;
```

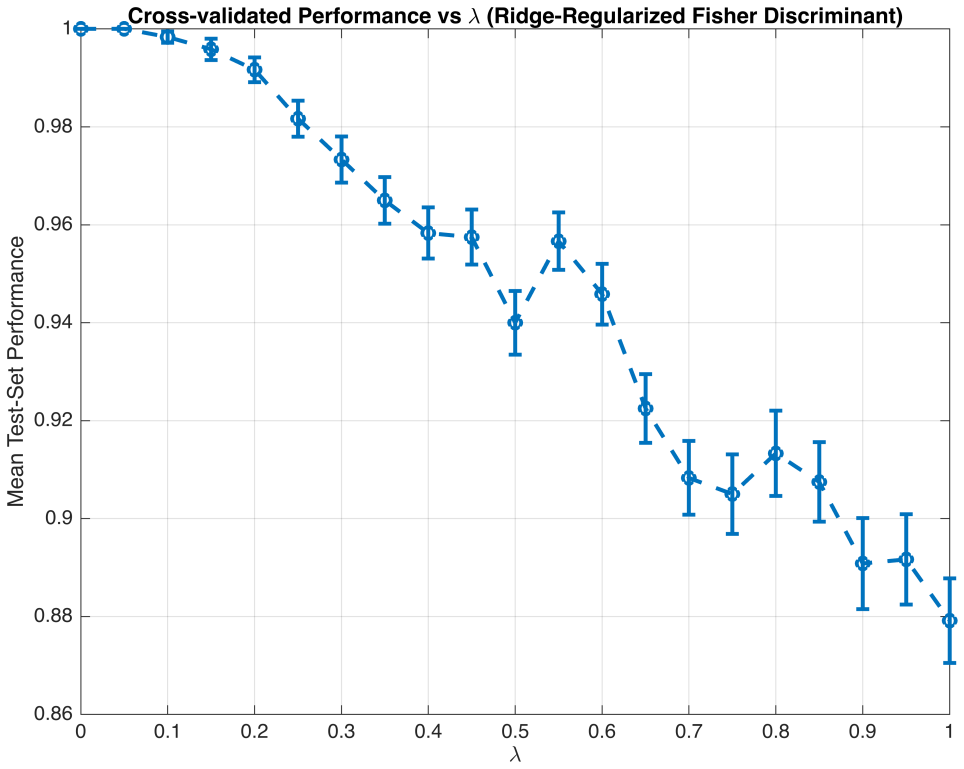


From two ways of computing Fisher's linear discriminant, we can see that 99.58%~100% of points are correctly being calssified.

Part c) Ridge-Regularized Fisher Discriminant

```
lambdas = 0:0.05:1;
nRuns = 100;
test_fraction = 0.05;
performance = zeros(length(lambdas), nRuns);

for inx_lambda = 1:length(lambdas) %lambda
    lambda = lambdas(inx_lambda);

    for run_i = 1:nRuns %runs
        idx1 = randperm(N1);
        idx2 = randperm(N2);
        Ntrain1 = round((1 - test_fraction) * N1);
        Ntrain2 = round((1 - test_fraction) * N2);
        train1 = data1(idx1(1:Ntrain1), :);
        test1 = data1(idx1(Ntrain1+1:end), :);
        train2 = data2(idx2(1:Ntrain2), :);
        test2 = data2(idx2(Ntrain2+1:end), :);

        mean1_train = mean(train1);
```

```
        mean2_train = mean(train2);
        S1_train = cov(train1);
        S2_train = cov(train2);
        Ntrain_total = Ntrain1 + Ntrain2;

        poolCov_train = ((Ntrain1 - 1) * S1_train + (Ntrain2 - 1) *
S2_train) / (Ntrain_total - 2);
        covEstimated = (1 - lambda) * poolCov_train + lambda *
eye(size(poolCov_train));


        [U, S, V] = svd(covEstimated);
        S_inv = diag(1 ./ diag(S));

        %Ridge-regularized Fisher discriminant
        diff_means_ridge = (mean2_train - mean1_train)';
        discriminant_ridge = (V * S_inv * U') * diff_means_ridge;
        discriminant_ridge = discriminant_ridge';

        x1_est = (V * S_inv * U') * mean1_train';
        x2_est = (V * S_inv * U') * mean2_train';
        threshold_ridge = (mean2_train * x2_est - mean1_train * x1_est) / 2;

        dec_test1 = ((test1 * discriminant_ridge') > threshold_ridge);
        dec_test2 = ((test2 * discriminant_ridge') > threshold_ridge);
        correct1 = sum(dec_test1 == 0); %go to class 1
        correct2 = sum(dec_test2 == 1); %go to class 2

        performance(inx_lambda, run_i) = (correct1 + correct2) /
(size(test1, 1) + size(test2, 1));
    end
end

%SEM
mean_perf = mean(performance, 2);
sem_perf = std(performance, 0, 2) / sqrt(nRuns);

figure;
errorbar(lambdas, mean_perf, sem_perf, 'o--', 'LineWidth', 2);
xlabel('\lambda');
ylabel('Mean Test-Set Performance');
title('Cross-validated Performance vs \lambda (Ridge-Regularized Fisher
Discriminant)');
grid on;
```




```
%best lambda
[best_perf, best_idx] = max(mean_perf);
best_lambda = lambdas(best_idx);
disp(['Best lambda: ', num2str(best_lambda), ' with performance: ',
num2str(best_perf)]);
```

```
Best lambda: 0 with performance: 1
```

The best lambda would be 0 since at that point perfromance is the peak.