Project 2 Writeup

Question 1: Beran's Cadenza

This problem was pretty satisfying to complete.

We are tasked with using a piecewise Gaussian model to find all note boundaries $0 = n_0 < n_1 < ... < n_K = 40000$

The given Gaussian distribution is:

$$\mathbb{P}(s|[n_{k-1}, n_k]) = \frac{1}{Z} e^{-\sum_{n_{k-1}}^{n_k - p_k}(s(n+p_k)-s(n))^2}$$

Which means the likelihood of a sequence of $[n_{k-1}, n_k]$ intervals will be

$$\prod_{k=1}^{K} \frac{1}{Z} e^{-\sum_{n_{k-1}}^{n_k - p_k}(s(n+p_k)-s(n))^2}$$

In order to include a prior for the number K of notes, we can multiply the likelihood by a simple additional exponential term $\mathbb{P}(K = k) = \frac{1}{Z}e^{-ak}$

$$\prod_{k=1}^{K} \frac{1}{Z} e^{-\sum_{n_{k-1}}^{n_k - p_k}(s(n+p_k)-s(n))^2} \times e^{-ak}$$

In order to simplify this probability, we take the log, ignoring constant terms:

$$\log \mathbb{P} = -\sum_{k=1}^{K} (\sum_{n_{k-1}}^{n_k - p_k} (s(n+p_k) - s(n))^2) - ak$$

We can justify the approximation given on p.105 of the textbook by expanding the components of the equation:

$$\sum_{k=1}^{K} \sum_{n_{k-1}}^{n_k - p_k} (s(n+p_k) - s(n))^2 = \sum_{k=1}^{K} \sum_{n_{k-1}}^{n_k - p_k} s(n+p_k)^2 + s(n)^2 - 2s(n)s(n+p_k)$$

Because $s(n+p_k)^2$ can be approximated as $s(n)^2$:

$$s(n+p_k)^2 + s(n)^2 \approx 2 \sum_{n=1}^{40000} s(n)^2$$

Given the pre-computed C matrix which is defined as:

$$C(n,p) = \sum_{k=1}^{n} s(k)s(k+p)$$

All together now, the expanded double sum can be approximated as

$$\approx 2\sum_{n=1}^{40000} s(n)^2 - 2\sum_{k=1}^{K}(C(n_k,p_k) - C(n_{k-1},p_k))$$

In order to maximize the probability rather than minimize the negative, we reverse the sign in the f_k function. Brute forcing the problem would be impossible, so we must use dynamic programming to efficiently compute the most likely segmentation of the music. We can use the Bellman algorithm because the sum we are maximizing can be written in the form:

$$F(x_1,...,x_n) = f_1(x_1,x_2) + f_2(x_2,x_3) + ... + f_{n-1}(x_{n-1},x_n)$$

Our f_k function takes a tuple of $[n_k, p_k]$ values:

$$f_k([n_{k-1},p_{k-1}],[n_k,p_k]) = (C(n_k,p_k) - C(n_{k-1},p_k))$$

Written in the Bellman form:

$$F([n_1,p_1],...,[n_K,p_K]) = f_1([n_1,p_1],[n_2,p_2]) + ... + f_{K-1}([n_{k-1},p_{k-1}],[n_k,p_k])$$

First, we establish that we will build 3 matrices with N rows and 30 columns (22 colums initially because K=22 is fixed):

h(n,k): The log likelihood of note index **k** ending at timestep **n**

p_argmax(n,k): The most likely pitch of note index **k** ending at timestep **n**

n_argmax(n,k): The most likely previous note boundary given the note index **k** ends at timestep **n**

We then find the minimum over the 2D range of h, iterating through all p_k and n_k. For the first step:

$$h_2([n_2,p_2]) = min[f_1([n_1,p_1],[n_2,p_2])]$$
$$\text{p\_argmax}(n_2,2) = argmax_{p_2}(f_1([n_1,p_1],[n_2,p_2]))$$
$$\text{n\_argmax}(n_2,2) = argmax_{n_1}(f_1([n_1,p_1],[n_2,p_2]))$$

And for subsequent steps:

$$h_k([n_k,p_k]) = \min(\mathbf{f\_1}([n_1,p_1],[n_2,p_2]) + ... + f_{k-1}([n_{k-1},p_{k-1}],[n_k,p_k])$$

We must note that we have already calculated the bolded terms in the $k-1$ step. Therefore, at each step, we maximize over n and p using the function:

$$f_k([n_{k-1}, p_{k-1}], [n_k, p_k]) = (C(n_k, p_k) - C(n_{k-1}, p_k))$$

summed with $h(n, k-1)$

Once we have filled the h(n,k) matrix, we backtrack through it to find the optimal path by finding $max_K(h_40000)$ because we know the last note boundary must end at n=40000.

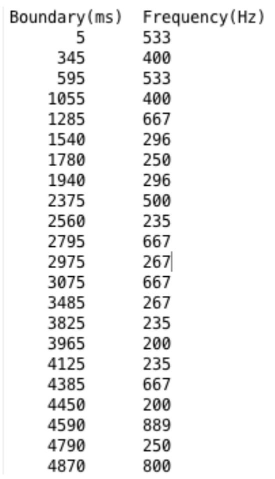
Once we have K, we repeatedly access the n_argmax(n, k) to find the optimal note boundaries and note frequencies:

$$n_{\bar{K}-1} = \text{n\_argmax}(40000, K)$$

$$n_{\bar{K}-2} = \text{n\_argmax}(n_{\bar{K}-1}, K-1)$$

$$\ldots$$

As we backtrack through note boundaries, we also access p_argmax(n, k) to find the optimal note at that boundary.



```
Boundary(ms)  Frequency(Hz)
     5          533
   345          400
   595          533
  1055          400
  1285          667
  1540          296
  1780          250
  1940          296
  2375          500
  2560          235
  2795          667
  2975          267
  3075          667
  3485          267
  3825          235
  3965          200
  4125          235
  4385          667
  4450          200
  4590          889
  4790          250
  4870          800
```

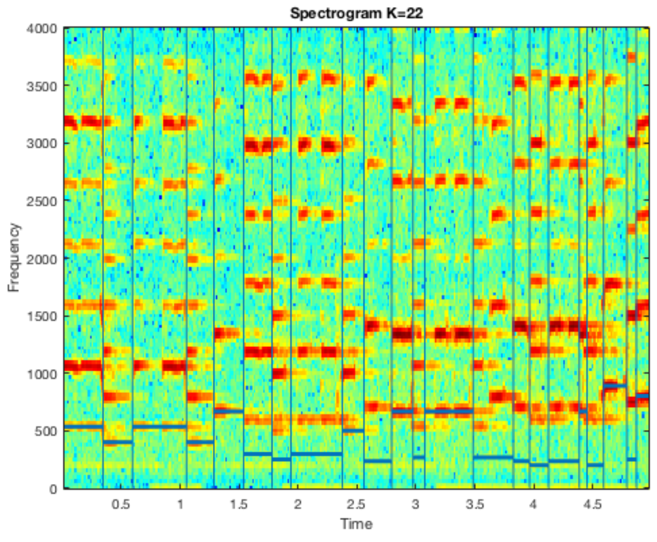


Figure 1: It can be observed that the frequencies match the spectrogram's high amplitude areas

In order to find the optimal a, we include the ak term in the summed f_k term. This acts to progressively decrease likelihood as k increases. Because we know that K=22 is correct, we can iterate over a to find the point where K=22.

At this point, we can remove the constraint that $K = 22$, and let the algorithm choose K within the boundaries for any audio sample.

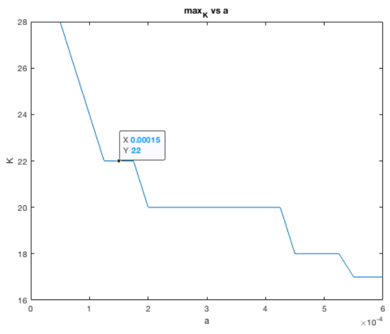


Figure 2: $a = 0.00015$ is found to be the best a

Question 2: Differentiability of Lacunary Series

A lacunary trigonometric series is defined as:

$$g(x) = \sum_{n=1}^{N=\infty} (a^n \sin(b^n x))$$

In order to make this feasible for Matlab to numerically compute, we must estimate ∞ as a large $N = 100$. Although this selection of N does not seem very large, the exponential sums ensures that anything greater will overflow a 64-bit double whose maximum is 10^{308}.

Given the constraint that $0 < a < 1 < b$, we select equally spaced a and b.

a	b
0.1	2.1
0.3	2.3
0.5	2.5
0.7	2.7
0.9	2.9

To get an idea for what the function will look like, we can think about what happens as b^n increases past 1. We can observe that for b terms larger than 2, the $sin(b^n x)$ wraps around and quickly becomes discontinuous.

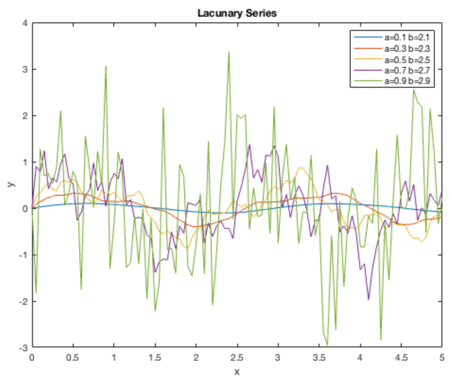


Figure 3: As b increases, the $sin(b^n)$ term quickly becomes unrecognizable and discontinuous

As one would expect, the derivative of this series is just as unrecognizable:

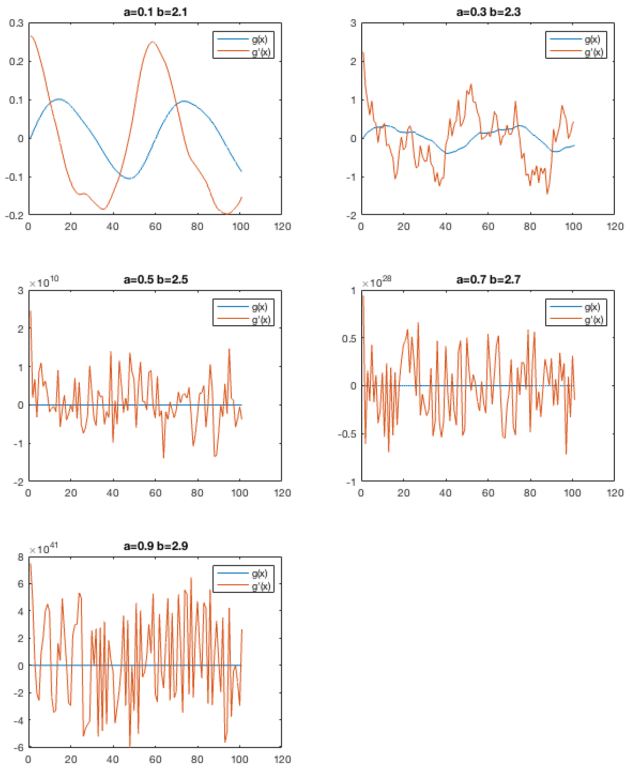


Figure 4: $g'(x)$ spikes into overflow territory, close to 10^{40}, when either n or b is slightly increased, it will quickly overflow

Question 3: Brownian and Ornstein-Uhlenbeck Processes

In this problem, we are tasked with modeling 6 Brownian Motion processes as described below. For each problem, the following notation will be used:

Brownian motion is a random sample continuously distributed with standard normal pdf

$$\beta = N(0, 1)$$

An Ornstein-Uhlenbeck process simply includes a term which nudges the derivative back to zero.

a) A sample of Brownian motion and an Ornstein-Uhlenbeck driven by the same random sample: Y_B is a Brownian random sequence, Y_{OU} is a Ornstein-Uhlenbeck sequence

$$dY_B = \beta$$

$$dY_{OU} = \beta - Y_{OU}$$

b) 2D Brownian motion:

$$dY = \beta$$

$$dX = \beta$$

c) A curve where the direction Θ is Brownian: Δ is a constant which determines the size of each step

$$d\Theta = \beta$$

$$dX = \Delta\cos(\Theta)$$

$$dY = \Delta\sin(\Theta)$$

d) A curve whose x and y velocities are Ornstein-Uhlenbeck processes: λ is a constant which tunes the rate which the Velocity is pulled back to 0

$$dV_X = \beta - \lambda V_X$$

$$dV_Y = \beta - \lambda V_Y$$

e) A curve whose curvature κ is Brownian Motion:

Δ is a constant which determines the size of each step

$$d\kappa = \beta$$

$$d\Theta = \kappa$$

$$dX = \Delta\cos(\Theta)$$

$$dY = \Delta\sin(\Theta)$$

f) A curve whose curvature is an Ornstein-Uhlenbeck process:
λ is a constant which tunes the rate which the curvature is pulled back to 0
Δ is a constant which determines the size of each step

$$d\kappa = \beta - \lambda\kappa$$

$$d\Theta = \kappa$$
$$dX = \Delta\cos(\Theta)$$
$$dY = \Delta\sin(\Theta)$$

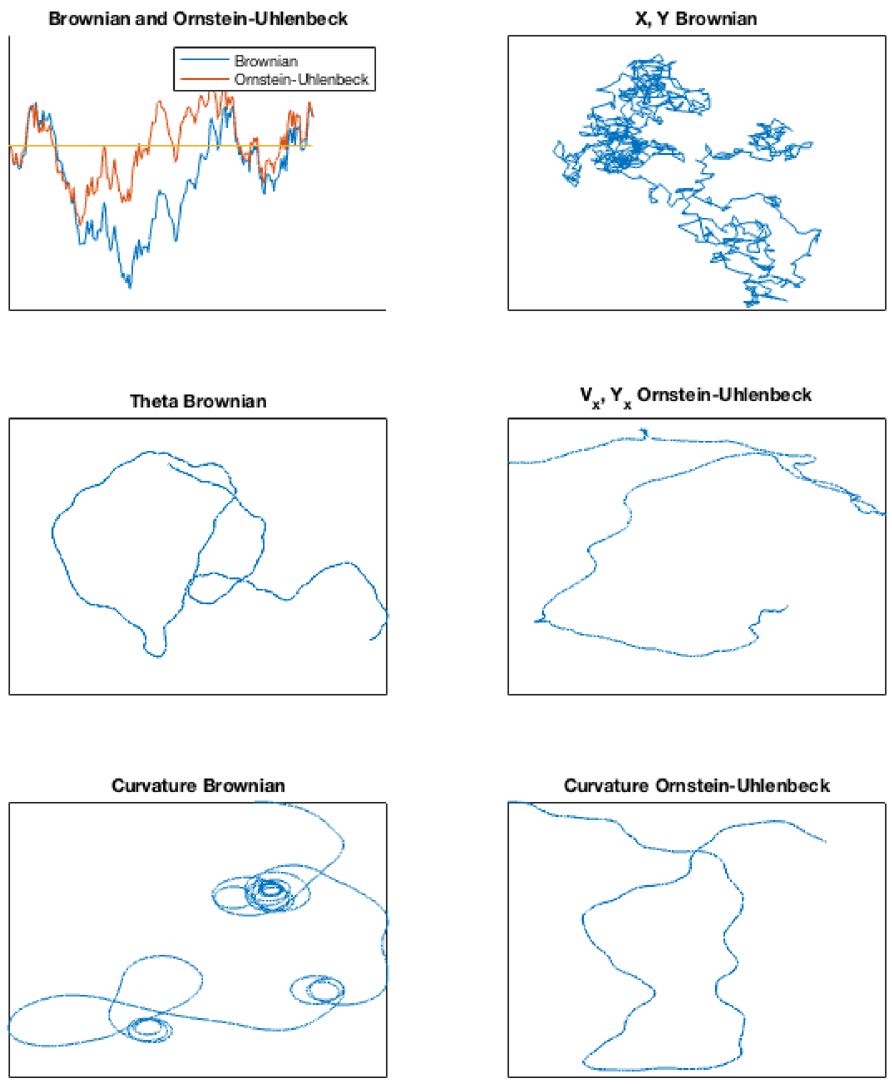
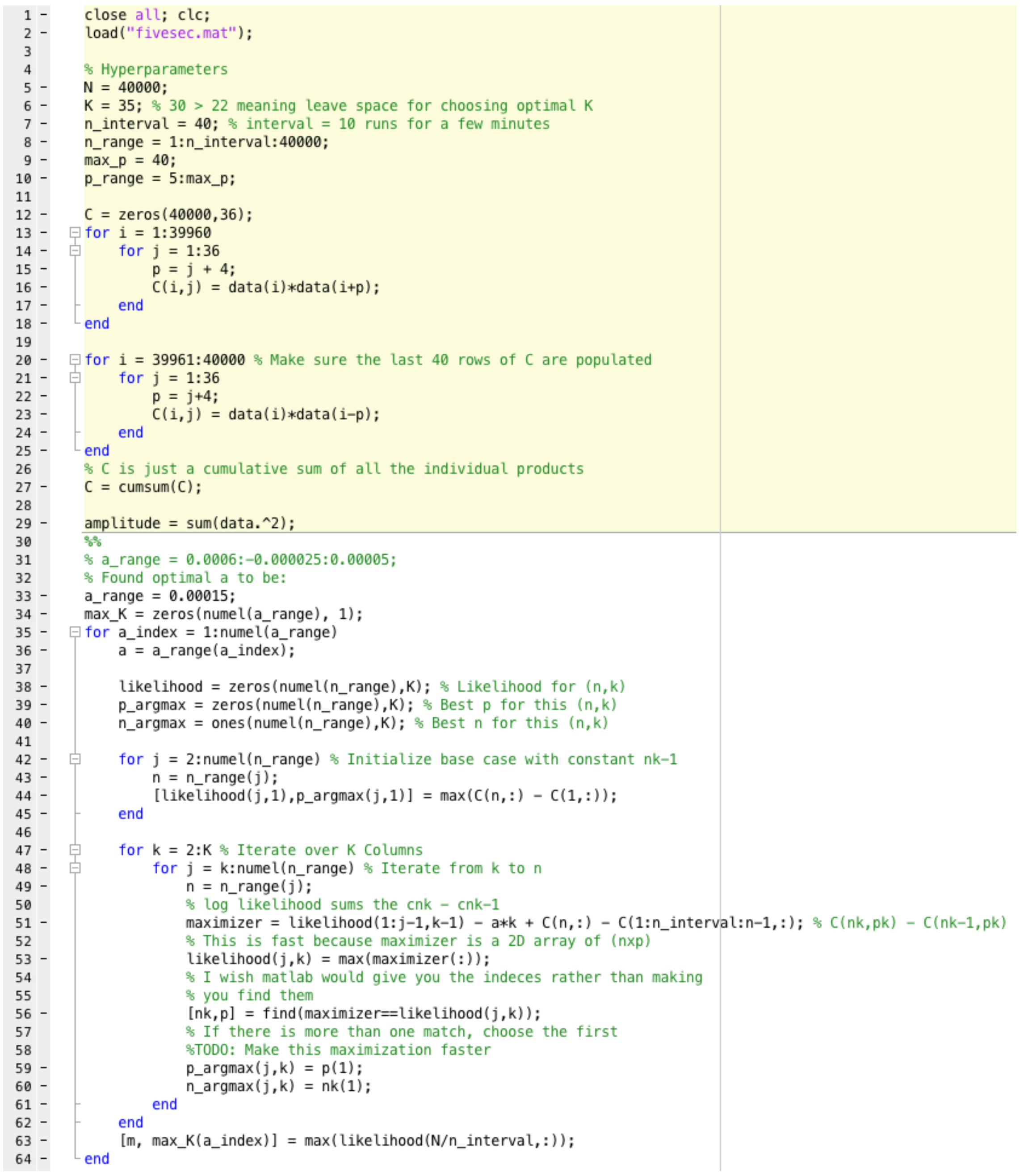


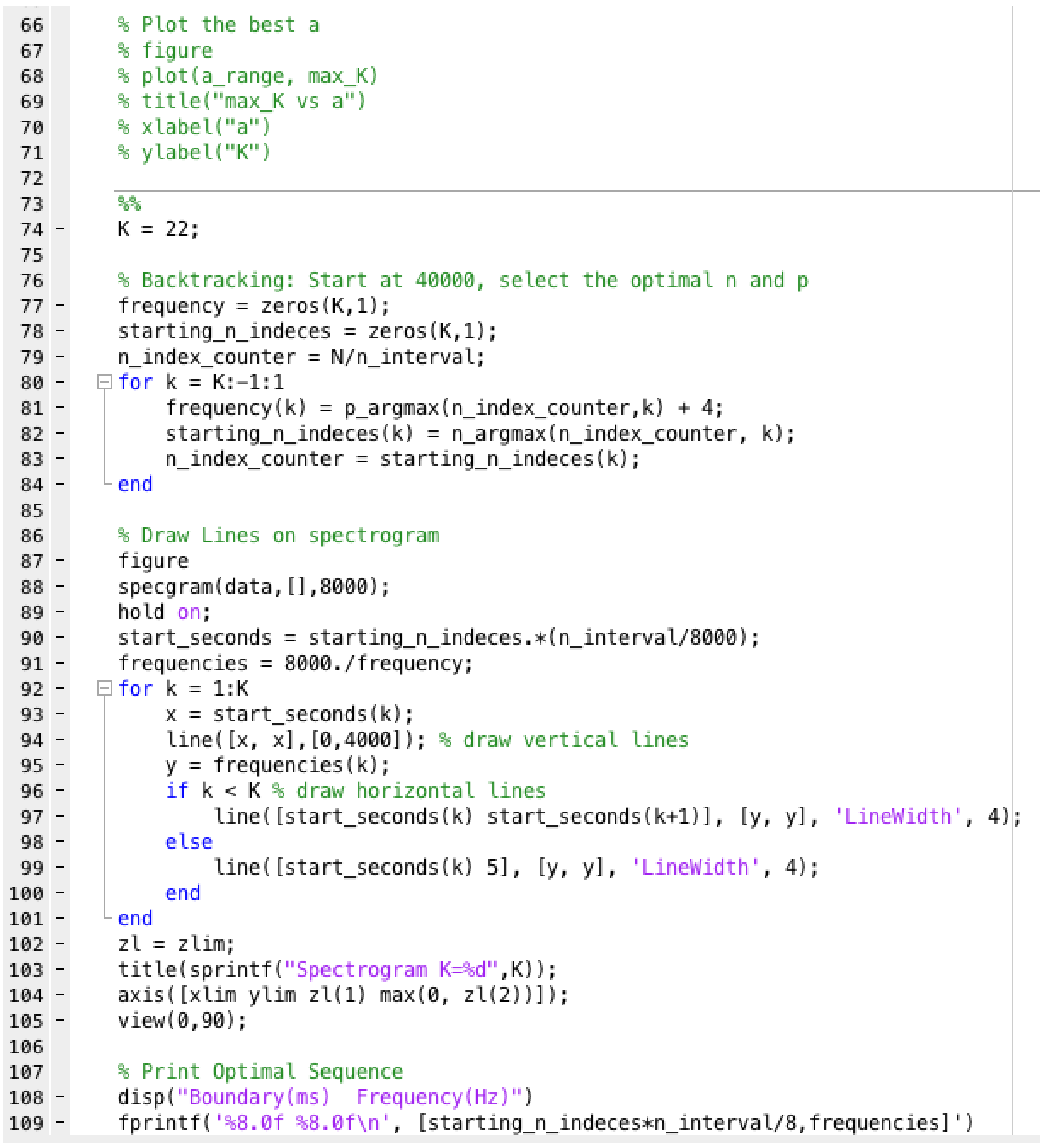
Figure 5: The scale of the random walks doesn't really matter

Appendix: Matlab Code

Question 1

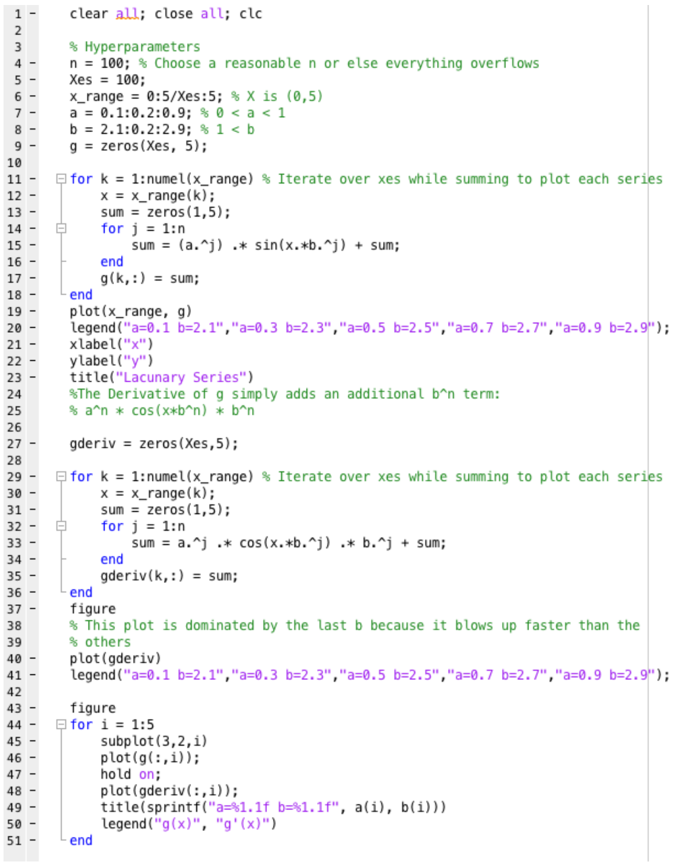


```
close all; clc;
load("fivesec.mat");

% Hyperparameters
N = 40000;
K = 35; % 30 > 22 meaning leave space for choosing optimal K
n_interval = 40; % interval = 10 runs for a few minutes
n_range = 1:n_interval:40000;
max_p = 40;
p_range = 5:max_p;

C = zeros(40000,36);
for i = 1:39960
    for j = 1:36
        p = j + 4;
        C(i,j) = data(i)*data(i+p);
    end
end

for i = 39961:40000 % Make sure the last 40 rows of C are populated
    for j = 1:36
        p = j+4;
        C(i,j) = data(i)*data(i-p);
    end
end
% C is just a cumulative sum of all the individual products
C = cumsum(C);

amplitude = sum(data.^2);
%%
% a_range = 0.0006:-0.000025:0.00005;
% Found optimal a to be:
a_range = 0.00015;
max_K = zeros(numel(a_range), 1);
for a_index = 1:numel(a_range)
    a = a_range(a_index);

    likelihood = zeros(numel(n_range),K); % Likelihood for (n,k)
    p_argmax = zeros(numel(n_range),K); % Best p for this (n,k)
    n_argmax = ones(numel(n_range),K); % Best n for this (n,k)

    for j = 2:numel(n_range) % Initialize base case with constant nk-1
        n = n_range(j);
        [likelihood(j,1),p_argmax(j,1)] = max(C(n,:) - C(1,:));
    end

    for k = 2:K % Iterate over K Columns
        for j = k:numel(n_range) % Iterate from k to n
            n = n_range(j);
            % log likelihood sums the cnk - cnk-1
            maximizer = likelihood(1:j-1,k-1) - a*k + C(n,:) - C(1:n_interval:n-1,:); % C(nk,pk) - C(nk-1,pk)
            % This is fast because maximizer is a 2D array of (nxp)
            likelihood(j,k) = max(maximizer(:));
            % I wish matlab would give you the indeces rather than making
            % you find them
            [nk,p] = find(maximizer==likelihood(j,k));
            % If there is more than one match, choose the first
            %TODO: Make this maximization faster
            p_argmax(j,k) = p(1);
            n_argmax(j,k) = nk(1);
        end
    end
    [m, max_K(a_index)] = max(likelihood(N/n_interval,:));
end
```



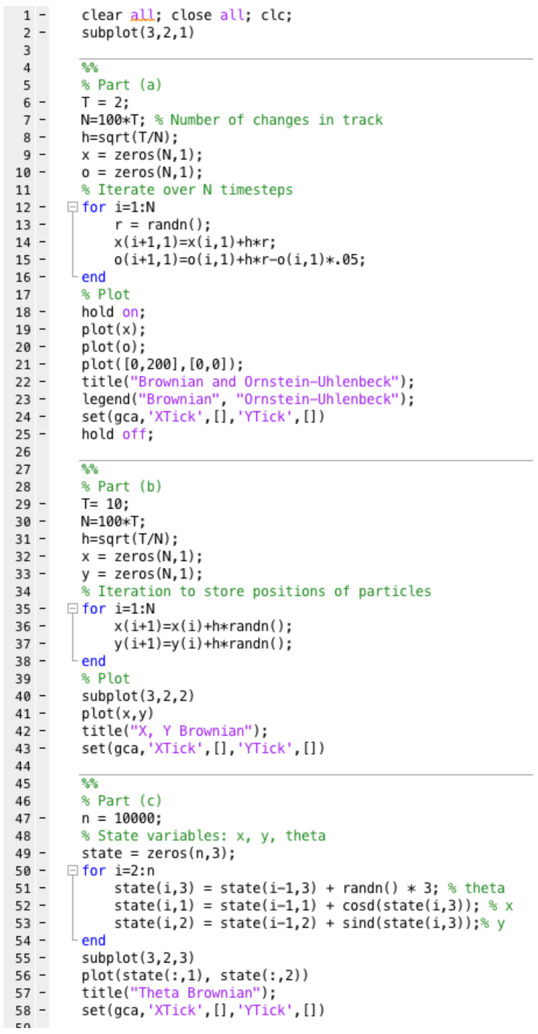
```
% Plot the best a
% figure
% plot(a_range, max_K)
% title("max_K vs a")
% xlabel("a")
% ylabel("K")

%%
K = 22;

% Backtracking: Start at 40000, select the optimal n and p
frequency = zeros(K,1);
starting_n_indeces = zeros(K,1);
n_index_counter = N/n_interval;
for k = K:-1:1
    frequency(k) = p_argmax(n_index_counter,k) + 4;
    starting_n_indeces(k) = n_argmax(n_index_counter, k);
    n_index_counter = starting_n_indeces(k);
end

% Draw Lines on spectrogram
figure
specgram(data,[],8000);
hold on;
start_seconds = starting_n_indeces.*(n_interval/8000);
frequencies = 8000./frequency;
for k = 1:K
    x = start_seconds(k);
    line([x, x],[0,4000]); % draw vertical lines
    y = frequencies(k);
    if k < K % draw horizontal lines
        line([start_seconds(k) start_seconds(k+1)], [y, y], 'LineWidth', 4);
    else
        line([start_seconds(k) 5], [y, y], 'LineWidth', 4);
    end
end
zl = zlim;
title(sprintf("Spectrogram K=%d",K));
axis([xlim ylim zl(1) max(0, zl(2))]);
view(0,90);

% Print Optimal Sequence
disp("Boundary(ms)  Frequency(Hz)")
fprintf('%8.0f %8.0f\n', [starting_n_indeces*n_interval/8,frequencies]')
```

Question 2

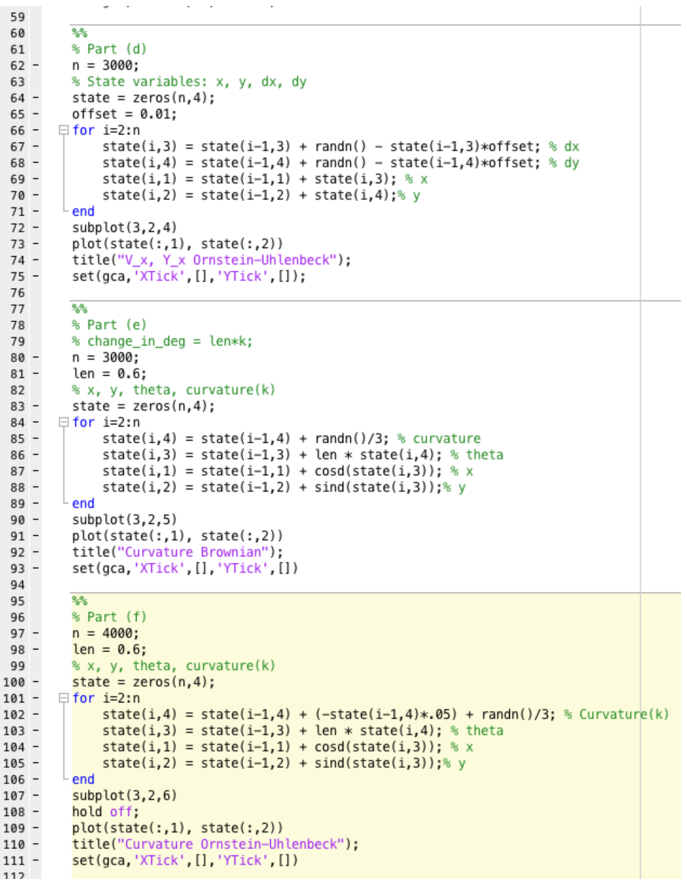


```
clear all; close all; clc

% Hyperparameters
n = 100; % Choose a reasonable n or else everything overflows
Xes = 100;
x_range = 0:5/Xes:5; % X is (0,5)
a = 0.1:0.2:0.9; % 0 < a < 1
b = 2.1:0.2:2.9; % 1 < b
g = zeros(Xes, 5);

for k = 1:numel(x_range) % Iterate over xes while summing to plot each series
    x = x_range(k);
    sum = zeros(1,5);
    for j = 1:n
        sum = (a.^j) .* sin(x.*b.^j) + sum;
    end
    g(k,:) = sum;
end
plot(x_range, g)
legend("a=0.1 b=2.1","a=0.3 b=2.3","a=0.5 b=2.5","a=0.7 b=2.7","a=0.9 b=2.9");
xlabel("x")
ylabel("y")
title("Lacunary Series")
%The Derivative of g simply adds an additional b^n term:
% a^n * cos(x*b^n) * b^n

gderiv = zeros(Xes,5);

for k = 1:numel(x_range) % Iterate over xes while summing to plot each series
    x = x_range(k);
    sum = zeros(1,5);
    for j = 1:n
        sum = a.^j .* cos(x.*b.^j) .* b.^j + sum;
    end
    gderiv(k,:) = sum;
end
figure
% This plot is dominated by the last b because it blows up faster than the
% others
plot(gderiv)
legend("a=0.1 b=2.1","a=0.3 b=2.3","a=0.5 b=2.5","a=0.7 b=2.7","a=0.9 b=2.9");

figure
for i = 1:5
    subplot(3,2,i)
    plot(g(:,i));
    hold on;
    plot(gderiv(:,i));
    title(sprintf("a=%1.1f b=%1.1f", a(i), b(i)))
    legend("g(x)", "g'(x)")
end
```

Question 3



```
clear all; close all; clc;
subplot(3,2,1)

%%
% Part (a)
T = 2;
N=100*T; % Number of changes in track
h=sqrt(T/N);
x = zeros(N,1);
o = zeros(N,1);
% Iterate over N timesteps
for i=1:N
    r = randn();
    x(i+1,1)=x(i,1)+h*r;
    o(i+1,1)=o(i,1)+h*r-o(i,1)*.05;
end
% Plot
hold on;
plot(x);
plot(o);
plot([0,200],[0,0]);
title("Brownian and Ornstein-Uhlenbeck");
legend("Brownian", "Ornstein-Uhlenbeck");
set(gca,'XTick',[],'YTick',[])
hold off;

%%
% Part (b)
T= 10;
N=100*T;
h=sqrt(T/N);
x = zeros(N,1);
y = zeros(N,1);
% Iteration to store positions of particles
for i=1:N
    x(i+1)=x(i)+h*randn();
    y(i+1)=y(i)+h*randn();
end
% Plot
subplot(3,2,2)
plot(x,y)
title("X, Y Brownian");
set(gca,'XTick',[],'YTick',[])

%%
% Part (c)
n = 10000;
% State variables: x, y, theta
state = zeros(n,3);
for i=2:n
    state(i,3) = state(i-1,3) + randn() * 3; % theta
    state(i,1) = state(i-1,1) + cosd(state(i,3)); % x
    state(i,2) = state(i-1,2) + sind(state(i,3));% y
end
subplot(3,2,3)
plot(state(:,1), state(:,2))
title("Theta Brownian");
set(gca,'XTick',[],'YTick',[])
```



```
%%
% Part (d)
n = 3000;
% State variables: x, y, dx, dy
state = zeros(n,4);
offset = 0.01;
for i=2:n
    state(i,3) = state(i-1,3) + randn() - state(i-1,3)*offset; % dx
    state(i,4) = state(i-1,4) + randn() - state(i-1,4)*offset; % dy
    state(i,1) = state(i-1,1) + state(i,3); % x
    state(i,2) = state(i-1,2) + state(i,4);% y
end
subplot(3,2,4)
plot(state(:,1), state(:,2))
title("V_x, Y_x Ornstein-Uhlenbeck");
set(gca,'XTick',[],'YTick',[]);

%%
% Part (e)
% change_in_deg = len*k;
n = 3000;
len = 0.6;
% x, y, theta, curvature(k)
state = zeros(n,4);
for i=2:n
    state(i,4) = state(i-1,4) + randn()/3; % curvature
    state(i,3) = state(i-1,3) + len * state(i,4); % theta
    state(i,1) = state(i-1,1) + cosd(state(i,3)); % x
    state(i,2) = state(i-1,2) + sind(state(i,3));% y
end
subplot(3,2,5)
plot(state(:,1), state(:,2))
title("Curvature Brownian");
set(gca,'XTick',[],'YTick',[])

%%
% Part (f)
n = 4000;
len = 0.6;
% x, y, theta, curvature(k)
state = zeros(n,4);
for i=2:n
    state(i,4) = state(i-1,4) + (-state(i-1,4)*.05) + randn()/3; % Curvature(k)
    state(i,3) = state(i-1,3) + len * state(i,4); % theta
    state(i,1) = state(i-1,1) + cosd(state(i,3)); % x
    state(i,2) = state(i-1,2) + sind(state(i,3));% y
end
subplot(3,2,6)
hold off;
plot(state(:,1), state(:,2))
title("Curvature Ornstein-Uhlenbeck");
set(gca,'XTick',[],'YTick',[])
```