

## Investigation of Gaussian Processes

### Parsing Beran's Cadenza

This problem was pretty satisfying to complete.

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

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:

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

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:

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

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:

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

First, we establish that we will build 3 matrices with N rows and 30 columns (22 columns 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:

$$\begin{aligned} h_2([n_2, p_2]) &= \min[f_1([n_1, p_1], [n_2, p_2])] \\ \mathbf{p\_argmax}(n_2, 2) &= \operatorname{argmax}_{p_2}(f_1([n_1, p_1], [n_2, p_2])) \\ \mathbf{n\_argmax}(n_2, 2) &= \operatorname{argmax}_{n_1}(f_1([n_1, p_1], [n_2, p_2])) \end{aligned}$$

And for subsequent steps:

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

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  $\text{n\_argmax}(n, k)$  to find the optimal note boundaries and note frequencies:

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

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

...

As we backtrack through note boundaries, we also access  $\text{p\_argmax}(n, k)$  to find the optimal note at that boundary.

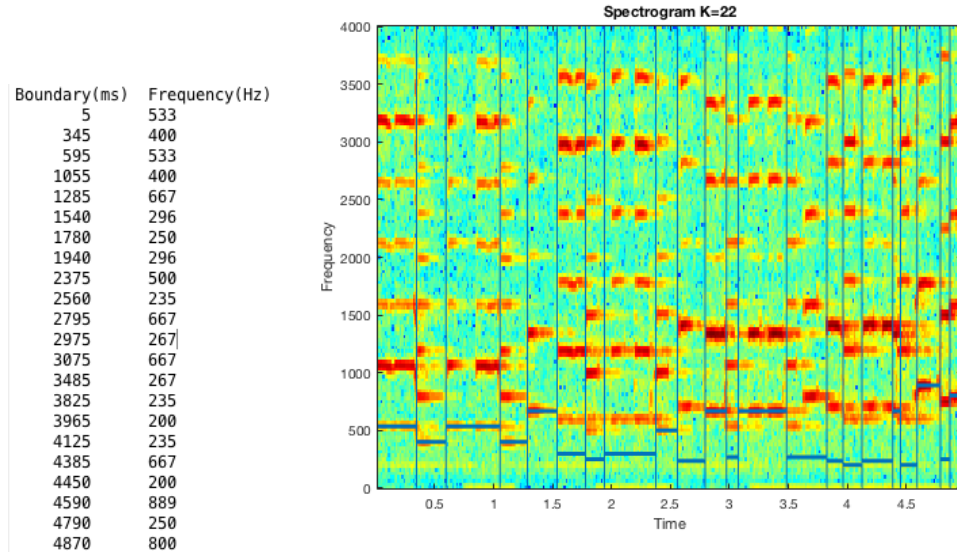


Figure 1: It can be observed that the frequencies match the spectrogram's high amplitude areas

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.

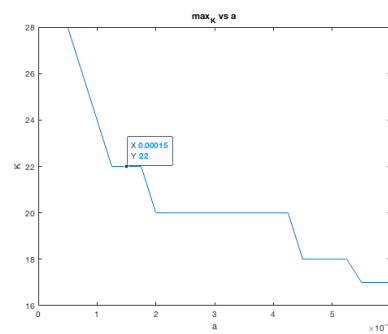


Figure 2:  $a = 0.00015$  is found to be the best  $a$

## 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  $\infty$  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.

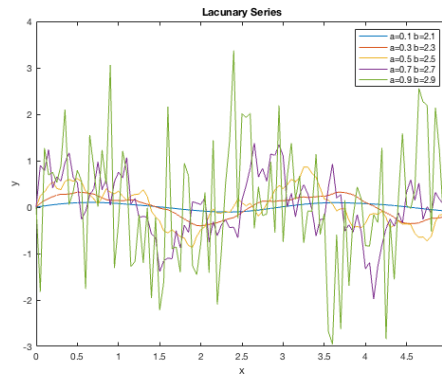


Figure 3: As  $b$  increases, the  $\sin(b^n)$  term quickly becomes unrecognizable and discontinuous

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

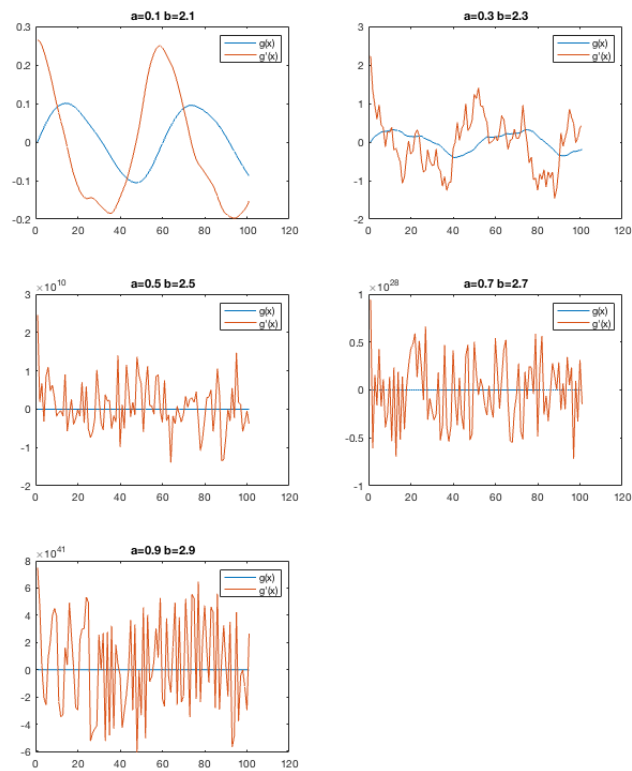


Figure 4:  $g'(x)$  spikes into overflow territory, close to  $10^{40}$ , when either  $n$  or  $b$  is slightly increased, it will quickly overflow

## 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  $\Theta$  is Brownian:  $\Delta$  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:  $\lambda$  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  $\kappa$  is Brownian Motion:

$\Delta$  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:

$\lambda$  is a constant which tunes the rate which the curvature is pulled back to 0

$\Delta$  is a constant which determines the size of each step

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

$$d\Theta = \kappa$$

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

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



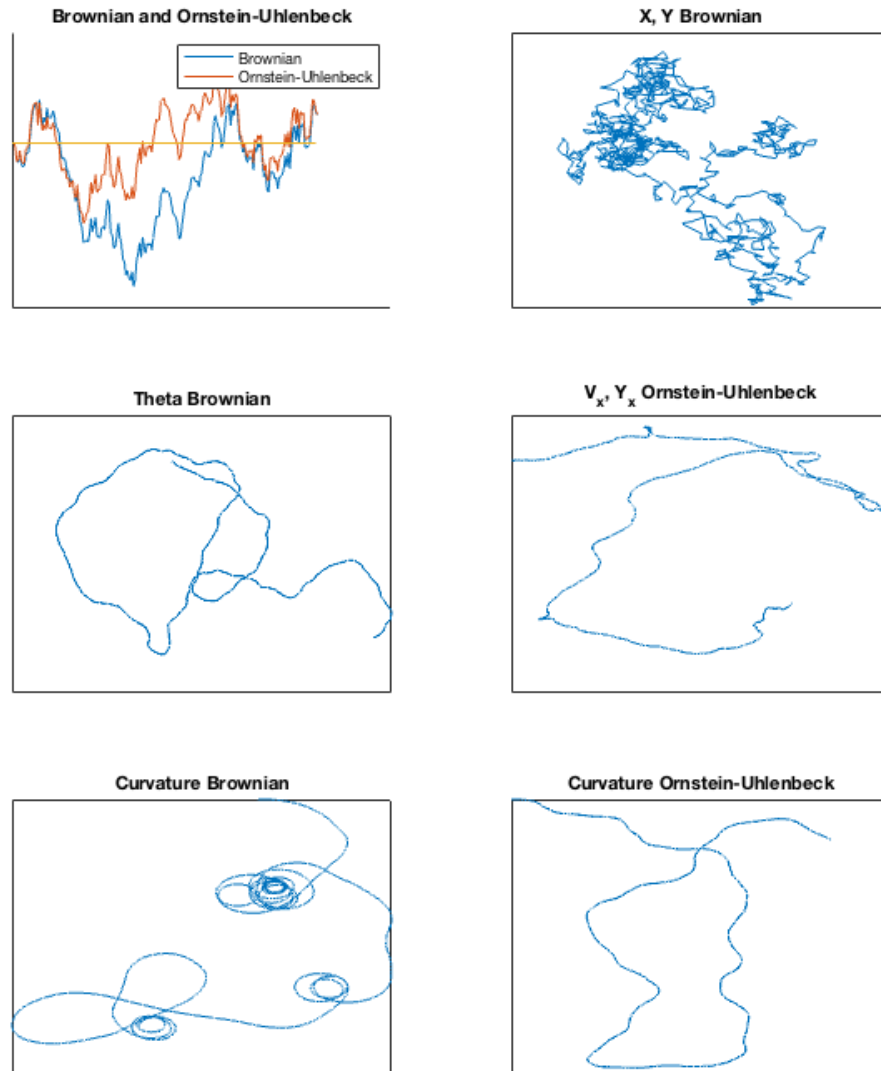


Figure 5: The scale of the random walks doesn't really matter