Music Transcription

Project with G. Kumar

October 26, 2018

1 Introduction

The high level objective of this project is to extract information from an audio signal produced by music. The information include: (i) the type of musical instrument being played, (ii) the key note being played, (iii) the tempo or the beat of the music, and so on. The challenging part of the project is that: (i) the signal may be corrupted with background noise, (ii) multiple instruments may be played at the same time, (iii) multiple notes may be played at the same time, and so on. In this project we only consider one instrument, piano, and focus on the problem of detecting which piano key is being played at each time instant

Problem statement: Consider an audio signal produced with piano key notes. At each time instant, detect which key is being played (in real-time).

This problem belongs to the field of signal processing and machine learning. It is an old problem with rich literature. It seems that the common approaches to solve the problem involve computing the spectrogram of the signal and performing feature extraction on the output from the spectrogram.

Our methodology is to formulate the problem in a probabilistic setting, and use Bayesian inference to solve the problem. In particular, we construct a model for each note, and apply filtering algorithms. Our model is based on an oscillator model and the filtering approach is coupled oscillator feedback particle filter.

Next, I describe our plan to approach this problem.

2 Oscillator model

Let $\theta \in [0, 2\pi]$ be an angle, and ω be a frequency. Consider the following dynamic model:

$$\text{Oscillator:} \quad \theta_{k+1} = (\theta_k + \omega \Delta t) \mod 2\pi \tag{1}$$

for $k = 0, 1, 2, \ldots, K$ where Δt is time step-size. We call model 1 an *oscillator*. Assume the initial angle $\theta_0 \in \text{uniform}([0, 2\pi])$ is a random number generated from uniform distribution on $[0, 2\pi]$. Assume we have access to a noisy signal given by:

$$\text{Observation signal:} \quad Y_k = h(\theta_k) + \sigma_w W_k, \tag{2}$$

where $h(\theta)$ is observation function, W_k is independent normal Gaussian noise, and $\sigma_w \geq 0$ is the noise strength. Assume the observation function is of the form:

$$\text{Observation function:} \quad h(\theta) = c\sin(\theta) \tag{3}$$

where c is a constant.

Task 1: Simulate system (1) and the signal (2) with parameters $\omega = 2\pi$, $\Delta t = 0.01$, $\sigma_w = 0.1$, $c = 1.0$, and $K = 100$. Produce the following figures: (i) a figure that shows the trace of the θ_k with k; (ii) a figure that shows the trace of Y_k with k. Note that the initial condition θ_0 is random. Simulate the system for $M = 100$ runs and create a histogram of θ_0 and θ_{50}. Can you also plot a histogram of the pair (θ_0, θ_{50})? Can you say anything about their correlation?

Next: Next we are going to assume we are not given $\{\theta_0, \theta_1, \ldots, \theta_K\}$. We only see the observation signal $\{Y_1, \ldots, Y_K\}$ and we are asked to estimate $\{\theta_0, \ldots, \theta_K\}$. But more on that later!

3 Coupled oscillator feedback particle filter

In this section, I describe the feedback particle filter (FPF) algorithm. The objective of the FPF algorithm is to estimate the state of a dynamical system based on noisy observation signal. For example, consider the oscillator model (1). Assume θ_0 is not known. Then the objective of FPF is to estimate θ_k based on the observation signal $\{Y_1, \ldots, Y_k\}$.

The objective is achived by computing what we call *posterior* probability distribution of θ_k. The posterior is a probability distribution that gives the probability of θ_k conditioned on the observation signal $\{Y_1, \ldots, Y_k\}$.

The posterior distribution is represented by empirical distribution of N oscillators $\{\theta^1, \ldots, \theta^N\}$. Each oscillator is assigned a frequency $\omega^i \sim \text{unif}([\omega - \delta, \omega + \delta])$ and is initilized uniformly $\theta_0^i \sim \text{unif}([0, 2\pi])$. Each oscillator evolves according to

$$\text{(FPF)} \quad \theta_{k+1}^i = \theta_k^i + \omega^i \Delta t + \mathsf{K}^i (Y_k - \frac{h(\theta_k^i) + \hat{h}}{2})\Delta t \mod 2\pi, \tag{4}$$

where $\hat{h} = \frac{1}{N}\sum_{i=1}^N h(\theta_k^i)$, and K^i is the *gain*. The gain is computed from Algorithm 1.

Algorithm 1 Algorithm for computing the gain in the (coupled oscillator) FPF

Input: $\{\theta^i\}_{i=1}^N$, $\{h(\theta^i)\}_{i=1}^N$
Output: $\{\mathsf{K}^i\}_{i=1}^N$

1: $\hat{h} = \frac{1}{N}\sum_{i=1}^N h(\theta^i)$
2: $A_{11} = \frac{1}{N}\sum_{i=1}^N \sin^2(\theta^i)$;1
3: $A_{22} = \frac{1}{N}\sum_{i=1}^N \cos^2(\theta^i)$;
4: $A_{12} = A_{21} = -\frac{1}{N}\sum_{i=1}^N \sin(\theta^i)\cos(\theta^i)$;
5: $b_1 = \frac{1}{N}\sum_{i=1}^N \cos(\theta^i)(h(\theta^i) - \hat{h})$
6: $b_2 = \frac{1}{N}\sum_{i=1}^N \sin(\theta^i)(h(\theta^i) - \hat{h})$
7: $c = A^{-1}b$
8: $\mathsf{K}^i = -c_1\sin(\theta^i) + c_2\cos(\theta^i)$ for $i = 1, \ldots, N$.

Task II: Consider the same parameters as Task I. Use the observation signal form Task I to implement the FPF algorithm 4. Choose $N = 100$ and $\delta = 0.1$. (i) Plot the trajectory of the oscillators $\{\theta_k^1, \ldots, \theta_k^N\}$ along with the trajectory of θ_k. (ii) Plot the histogram of the oscillators at $k = 0$, $k = 10$, and $k = 50$.

In [8]:
```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import sympy
import plotly.plotly as py
import plotly.tools as tls
```

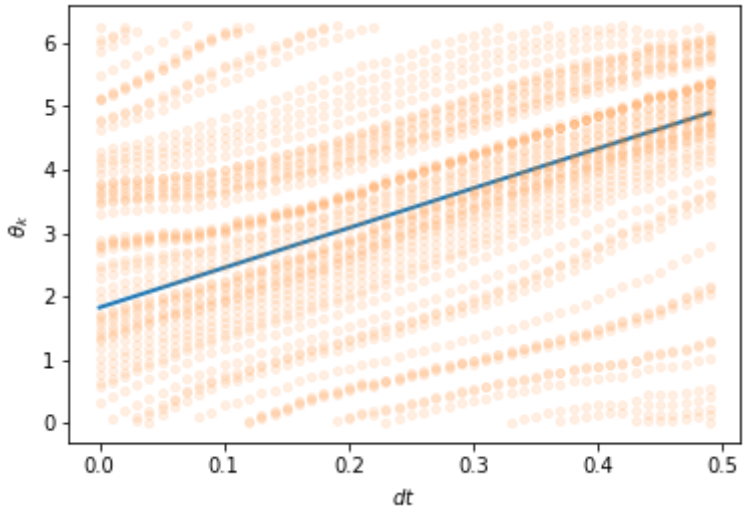
In [9]:
```
import random
g = np.random # random number genrator
```

In [10]:
```
delta = 0.1
dt = 0.01
omega = 2*np.pi
sigma_w = 0.5
```

In [19]:
```
K = 50
N = 50
t_k = np.arange(0,K*dt,dt)
```

In [20]:
```
omega_i = g.uniform(omega - delta, omega + delta, N) # you can use this
 too

#Kalman Gain computation
def FPF_gain(theta_i):
    h_k   = np.sin(theta_i)
    h_hat = (1./N)*sum(h_k)
    A11 = (1./N)*sum(np.sin(theta_i)*np.sin(theta_i))
    A22 = (1./N)*sum(np.cos(theta_i)*np.cos(theta_i))
    A12 = (1./N)*sum(np.sin(theta_i)*np.cos(theta_i))
    A   = np.array([[A11, A12],
               [A12, A22]])
    B_1 = (1./N)*sum(np.cos(theta_i)*(h_k - h_hat))
    B_2 = (1./N)*sum(np.sin(theta_i)*(h_k - h_hat))
    B   = np.array([[B_1],
                [B_2]])
    #Inverse matrix of A
    A_inverse = np.linalg.inv(A)
    #C = A^-1*B
    C = np.dot(A_inverse, B)
    c1 = C[0]
    c2 = C[1]
    #Kalman Gain with theta_i for i = 1,..., N
    K_i = -c1*np.sin(theta_i) + c2*np.cos(theta_i)
    return K_i


def osc():
    theta_k = np.zeros(K)
    theta_k[0] = g.uniform(0,2*np.pi)
    for k in range(K-1):
        theta_k[k+1] = (theta_k[k] + omega*dt) % (2*np.pi)
    return theta_k #defining theta_k (might be irrelevant)


#FPF Function
def FPF(Y_k):
    theta_ik = np.zeros([N,K])
    theta_ik[:,0] = g.uniform(0, 2*np.pi, N)
    for k in range(K-1):
        h_k = np.sin(theta_ik[:,k])
        K_i = FPF_gain(theta_ik[:,k])
        h_hat = np.mean(h_k)
        ykhat = Y_k[k] - (h_k + h_hat)*.5
        theta_ik[:,k+1] = (theta_ik[:,k] + omega_i*dt + K_i*ykhat*0.01/(
sigma_w*sigma_w)) % (2*np.pi)
    return theta_ik

theta_k = osc()
Y_k = np.sin(theta_k) + sigma_w*np.random.randn(K)
theta_ik = FPF(Y_k)
```

In [21]:
```
#trajectory of the oscillators
plt.plot(t_k,theta_k,lw=2, label = 'true oscillator',color='C0')
for i in range(N):
    plt.plot(t_k,theta_ik[i,:],lw=2,alpha=0.1,color='C1', ls='none',ms=4
,marker='o')
plt.ylabel(r'$\theta_k$')
plt.xlabel(r'$dt$')
plt.show()
```



In [22]:
```
#hist of the oscillators
plt.hist(theta_ik,normed=1)
for i in range(N):
    plt.hist(theta_ik[i,:],lw=2,alpha=0.1,color='C1')
plt.show()
```