TSKS15 Computer Laboratory Exercise 2

Fall 2020

Version of this document: September 4, 2020

The objective of this computer project is to implement a classifier that can detect music melodies. More specifically, the task of the classifier is to take a noisy signal produced by a signal generator and determine which melody, among a predetermined set of melodies, this signal contains.

You will implement two different classifiers based on different signal models, and use Monte-Carlo simulation to analyze how these classifiers perform for different noise levels.

1 Background on Music Theory

We start with some introduction to how music and melodies can be described in mathematical terms. The audio signals (sound pressure) generated by many real music instruments, such as a piano, are approximately periodic. These signals can then be expanded in a Fourier series consisting of a superposition of sinusoids. The lowest frequency in this expansion, say f_0, for a particular note is called the *fundamental frequency*. The other frequencies in the expansion are integer multiples of f_0 and called *harmonics* (*overtones*).

The simplest possible model of a tone from an instrument would be a single sinusoid with fundamental frequency f_0. A more realistic model includes also (some of) the harmonics. In this project, we will be considering the 1st, 3rd, and 5th harmonic (equivalently the fundamental tone, the 2nd overtone, and the 4th overtone). Hence, the three tones present in the signal have frequencies f_0, $3f_0$, and $5f_0$, respectively.

An *octave* refers to a factor-of-two separation between two tones. Each octave is divided into 12 semitones, such that the frequency ratio between two subsequent semitones is $2^{1/12}$. The

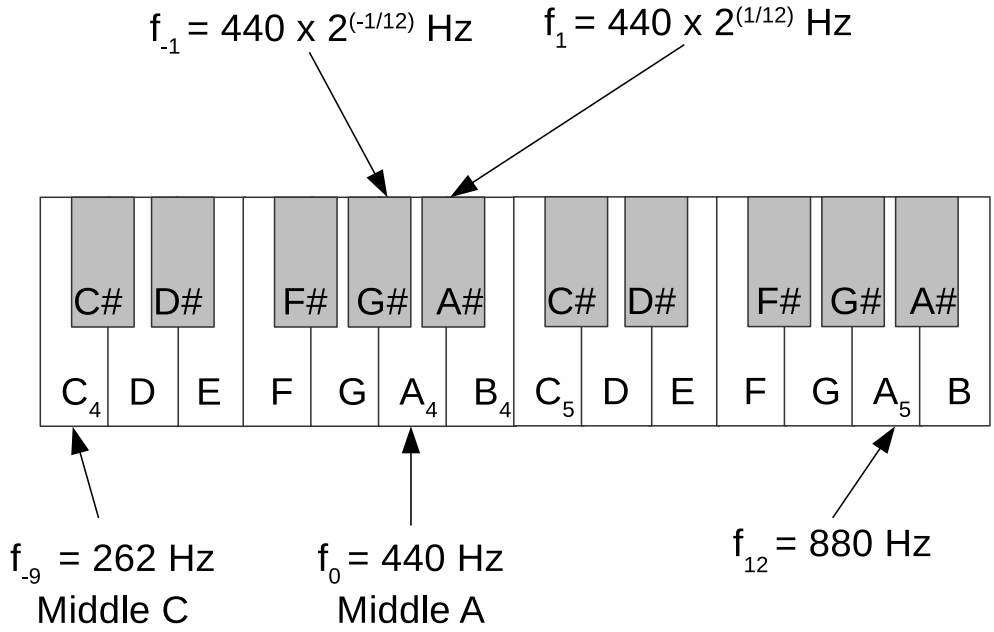

Figure 1: A piano keyboard, and the fundamental frequency of some of the keys (notes). The illustration shows all the notes in the 4th and 5th octave.

semitones within an octave are C, C#, D, D#, E, F, F#, G, G#, A, A#, B, C where the last C has a fundamental frequency equal to twice that of the first C. If the fundamental frequency of the first C is f_0, then the fundamental frequencies of the subsequent 12 notes are

$$2^{1/12} f_0, 2^{2/12} f_0, ..., 2^{11/12} f_0, 2 f_0.$$

For unambiguous notation, whenever it is not obvious from the context, each note may be amended by an integer that designates its octave number, for example, C4, G4, C5, ... The difference in fundamental frequency, for example, between the same note in two neighboring octaves (e.g. C4 and C5) is then a factor of two.

On the piano, the tone A4 (the A key in the middle of the piano) has the fundamental frequency 440 Hz. The piano (and other instruments) normally require regular tuning to ensure that, for example, a played note A4 generates a tone with fundamental frequency (very close to) 440 Hz. Figure 1 shows the location of notes on a piano keyboard, the octave numbering, and some of the corresponding fundamental frequencies.

2 The Melody Dataset and Signal Generator

On the course webpage there is a zip-file with all files needed to solve the lab. Make sure to download these files before you start working. All computers in the computer labs Olympen and Asgård are equipped with both Python3 and Matlab. All scripts we have provided have been tested on those computers.

We will be working with a dataset consisting of 10 melodies. These melodies are defined in the file `melodies.txt`. Your solution will require this file to map the melodies to notes, and it is also used by the signal generator (see below). The notes of each melody is specified on one of the lines in the file. (The file has 10 lines.) Each melody consist of 12 notes of equal duration. For example, the first line reads `C4, C4, G4, G4, A4, A4, G4, G4, F4, F4, E4, E4`, which are the first 12 notes of the song "Twinkle Twinkle".

To generate signals, use the signal generator (`SignalGenerator.py` or `SignalGenerator.m`). Each time this signal generator is invoked, it produces one of the ten melodies, selected uniformly at random. The signal produced is slightly off pitch, such that the fundamental frequency of the note is multiplied by a small factor (close to 1), simulating an instrument tuning error. For example, when the signal generator would ideally generate the A4 note, corresponding to $f_0 = 440$ Hz, it instead generates αf_0, where α is the tuning/pitch mismatch ($\alpha \approx 1$, but $\alpha \neq 1$). This tuning mismatch is unknown, and needs to be determined by your classifier. In addition to the tuning mismatch, the signal is also corrupted by additive, white, Gaussian noise with variance σ^2.

The output of the signal generator is

- a signal containing a randomly selected melody (noisy and with a tuning mismatch), sampled with frequency 8820 Hz
- the index of the selected melody, m, (i.e. the line number of `melodies.txt`),
- the pitch mismatch, $\alpha \in \mathcal{A}$ (with equal probability); $\mathcal{A} = \{0.975, 1.025\}$

Importantly, your classifier must only use the signal. The two last outputs (melody index and the pitch mismatch) are provided for validation only, to be able for you to check whether or not your classifier made the correct decision.

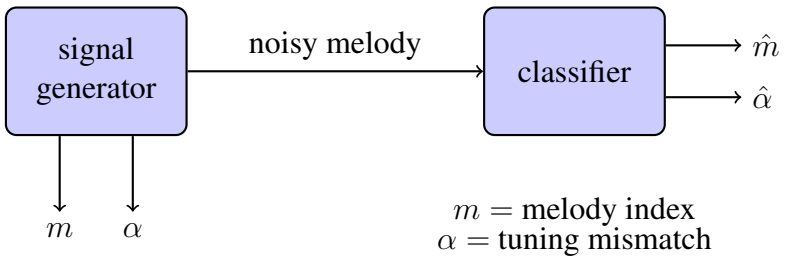

Figure 2: Interplay between the signal generator, and the classifier you will construct. The classifier takes a noisy signal as input, and determines which melody the signal contains (along with the tuning mismatch). The classifier output $\hat{m}, \hat{\alpha}$ is then compared to the ground truth m, α, in order to estimate its performance.

3 Your Task

Your task is to implement classifier algorithms that take a noisy signal and determines what melody the signal contains, and then evaluate the performance of these classifiers. The interplay between the signal generator and your classifier is illustrated in Figure 2.

There are 10 possible melodies and 2 possible pitch mismatches, yielding in total 20 possible combinations. All these combinations are equally likely. Each melody consists of 12 notes.

In the language of hypothesis testing, the classifier's task is then to select among 20 hypotheses $\{\mathcal{H}_j\}$ corresponding to these 20 combinations. Let m be the index of the melody, $m = 1, ..., 10$. Let l be the index of the pitch mismatch, $l = 1, 2$. If we enumerate the 20 hypotheses $\{\mathcal{H}_j\}$, there will be a one-to-one correspondence between j, m, and l: $j = 2(m-1) + l$.

There are three subtasks:

1. Implement a single-tone classifier.
2. Implement a three-tone classifier.
3. Use Monte-Carlo simulation to evaluate the performances of these classifiers.

These three subtasks are described in more detail in the next three subsections.

3.1 Single-Tone (Sinusoid) Classifier

This is the simplest possible classifier. It assumes that each melody consists of a series of concatenated sinusoids, one for each note. Each note in turn consists of a single sinusoid with an unknown phase and amplitude.

Let $f_{n,m}$ denote the fundamental frequency of the nth note in the mth melody.[1] The note produced by the signal generator is denoted $\bar{f}_{n,j} = f_{n,m}\alpha_l$, where $\alpha_l \in \mathcal{A}$ is the lth pitch mismatch. (The mapping between j, m and l is as given above.) The nth note of the output of the signal generator under $\mathcal{H}_j$ can be modeled as

$$A_n \cos(2\pi \bar{f}_{n,j} k + \phi_n),$$

where A_n is a random, unknown attenuation, ϕ_n is a random, unknown phase shift, and k is (discrete) time.

Here we will use a Bayesian approach to the detection problem. The Bayesian framework allows us to derive a classifier that is optimal (in the minimum-probability-of-error sense) for given statistical distributions of A_n and ϕ_n. For analytical tractability, we will assume that A_n is Rayleigh distributed and ϕ_n is uniformly distributed between 0 and 2π.[2]

An important point is that the classifier that results from this assumption is optimal only if the signal is actually generated by a model for which A_n and ϕ_n are Rayleigh respectively uniformly distributed. But A_n and ϕ_n in the signal generated by the signal generator, albeit random, do not have these distributions.[3] However, this is not a problem: The classifier that we build is still useful (though not optimal) for data generated from a different model than the one assumed in its derivation. In fact, in practice in algorithm design it is very common that one does not know the exact statistical distributions of the unknown parameters, and simply assumes a distribution that is either physically reasonably, or leads to simple mathematics, or, preferably, both.

The melody is represented by a vector of samples:

$$\boldsymbol{y} \triangleq \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_N \end{bmatrix},$$

[1] All frequencies in this description are discrete-time frequencies. The continuous-time frequency corresponding to $f_{n,m}$ is $f_{n,m} \cdot f_s$, where $f_s = 8820$ Hz is the sampling frequency.

[2] This is a commonly used model known as "Rayleigh fading".

[3] You may see this by inspecting the code of the signal generator.

where $\boldsymbol{y}_n \in \mathbb{R}^K$ is the vector of samples associated with the nth note, N is the number of notes per melody, and K is the number of samples per note.

Under the assumptions made on the statistical distributions of A_n and ϕ_n, a sampled tone can be modeled as [1, Example 5.5]

$$\boldsymbol{y}_n = \begin{bmatrix} 1 & 0 \\ \cos(2\pi \bar{f}_{n,j}) & \sin(2\pi \bar{f}_{n,j}) \\ \vdots & \vdots \\ \cos(2\pi \bar{f}_{n,j}(K-1)) & \sin(2\pi \bar{f}_{n,j}(K-1)) \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{K-1} \end{bmatrix},$$

or equivalently

$$\boldsymbol{y}_n = \boldsymbol{H}_{n,j}\boldsymbol{\theta}_n + \boldsymbol{w}_n,$$

where

$$\boldsymbol{\theta}_n \sim \mathcal{N}(\boldsymbol{0}, \rho \boldsymbol{I}_2)$$

and

$$\boldsymbol{w}_n \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}_K).$$

(The observation matrix $\boldsymbol{H}_{n,j}$ for each note is of dimension $K \times 2$.)

Based on this model, derive the optimal detector by following these steps:

(a) Write up the (conditional) probability density function for the nth note signal under the jth hypothesis, $p(\boldsymbol{y}_n|j)$

(b) Simplify the formula for the probability density function by using

- Sylvester's determinant identity
- The approximation
$$\frac{1}{K}\boldsymbol{H}_{n,j}^T \boldsymbol{H}_{n,j} \approx \frac{1}{2}\boldsymbol{I},$$
for large K (also explain why this is a legitimate approximation)
- The matrix inversion lemma

Your final expression for the probability density should be very short and simple, and it will contain an exponential function and a squared Euclidean norm.

(c) Write out an explicit, simplified expression for the optimal classifier. Hint: recall that all $\boldsymbol{y}_n$ are mutually independent, conditioned on j.

Your final expression should be in terms of finding the index j for which a certain sum of squared Euclidean norms is the largest.

3.2 Three-tone classifier

This classifier assumes that each note consists of three sinusoids with unknown phases and amplitudes. The sinusoids have frequencies equal to the fundamental frequency $f_{n,m}$, and its second- and fourth-order harmonics ($3f_{n,m}$ and $5f_{n,m}$). The phases and amplitudes for all sinusoids are modeled as random and mutually independent.

Part of your task is to generalize the single-tone detector derived above to this three-tone model. (Hint: the observation matrix for each note, $\boldsymbol{H}_{n,j}$, will now be $K \times 6$.)

3.3 Performance Evaluation by Monte-Carlo Simulation

You will conduct a numerical study (Monte-Carlo simulation) to determine which classifier that works best. Try each of the two classifiers on two different kinds of signals produced by the signal generator, resulting in a total of four different scenarios:

1. Single-tone classifier on a single-tone melody
2. Single-tone classifier on a three-tone melody
3. Three-tone classifier on a single-tone melody
4. Three-tone classifier on a three-tone melody

To see how the SNR (ρ/σ^2) affects performance, try each one of these in the presence of noise. Plot the probability of error (misclassification) as function of the SNR.

4 Hints

- Octave/Matlab or Python may be used for the programming. (The signal generator supports both choices.)
- To gain some insights into the problem, listen to the melodies generated by the signal generator (`.wav` files) by using `aplay` in Linux, or the program `tsks15audio.py` in Python, or the commands `audioread` respectively `soundsc` in Octave/Matlab.

- If the SNR is too high, both classifiers perform well for both single-tone and three-tone sinusoidal melodies. Reduce the SNR such that you can see a difference between the classifiers.
- To speed up the classifier, it is recommended to decrease the number of samples processed for each note. For example, even though the note consists of K samples, your classifier may only consider the first $K/10$. To compensate for fewer samples, you will need to increase the SNR.
- To get stable curves, count at least 50 misclassification errors for each SNR.
- The interesting SNR range is somewhere between -50 dB and -10 dB, when using the full signal (all samples). If using fewer samples, you need to compensate this by increasing the SNR.

4.1 For the Octave/Matlab User

All files required for the lab can be obtained from the course webpage. You may study the files `tsks15_example1.m` and `tsks15_example2.m` to see examples of how the signal generator can be called.

4.2 For the Python User

The package `numpy` has to be installed. You will have to import SignalGenerator class (contained in `SignalGenerator.py`), in order to use its functions. The most relevant function for you is `generate_random_melody`. You may study the example files `tsks15_example1.py` and `tsks15_example2.py` to see how this works.

5 Examination

- Individual oral examination takes place in class (in the computer lab). Please see the course webpage for exact dates.

  To pass the lab examination, you must present

  - a single-tone classifier that achieves 100% success rate when the unknown signal is a one-tone melody and the SNR ρ/σ^2 is large enough;
  - a three-tone classifier that achieves 100% success rate when the unknown signal is a three-tone melody and the SNR ρ/σ^2 is large enough;
  - a graph consisting of four curves, one for each scenario, showing the impact of SNR on the performance. The horizontal axis should show the SNR, and the vertical axis should show probability of error (misclassification).

  You must be able to answer questions regarding your implementation and program code, and questions on your derivation of the three-tone classifier.

  When plotting the graphs, use $K/10$ samples and enough Monte-Carlo runs.

- Collaboration on this homework in small groups, is encouraged, but each student should write up her/his own program code. Copying of program code from other students, or from previous years' students, or from the Internet is strictly prohibited.

- The program code you have written should be submitted to Urkund via email: `erik.g.larsson.liu@analys.urkund.se`.

References

[1] S. Kay, *Fundamentals of Statistical Signal Processing, Volume II: Detection Theory*, 1st ed. Englewood Cliffs, N.J: Prentice Hall, Feb. 1998.