Class Project: EE2800 – Digital Signal Processing

January-May 2024

Instructor: Sundar Vanka

Guest Instructor: N. R. Sanjeev

Due: Friday, April 19, 2024, 11:59 pm

Talk and Demo: Saturday, April 20, 2024 in A-220 (10 am - 1 pm)

INSTRUCTIONS

1. **General**
   - There are **TWO problem statements** in this document. Your team is expected to pick **EXACTLY ONE** of these to work on for the class project.
   - **This project contributes 15% to your overall score**.
2. **Project Workflow and Tutorial Classes**
   - The problems involve the **design and implementation of DSP algorithms** to solve real-life problems, albeit in a vastly simplified form.
   - You are expected to **implement your algorithms only in Matlab.**
   - As in practical engineering, concepts and insights from the the class and computer tutorials are useful, but don't expect to get results by plugging in some formula or using a known result: **there may be no single correct way to achieve the design objective, although you will realize there are several wrong ways to avoid :-)**
   - **Students are free to look up any additional material they may require and cite the same in their slides.**
   - **Teams can approach the instructor/guest-lecturer any time to get inputs about the project and/or their design. The usual means of contacting apply. No penalties for such advice.**
3. **Project Deliverables:**
   - A **two-slide deck** summarizing your solution approach, design and any limitations.
   - **Matlab code that implements this design** (may include one or more m-files, should work on institute-supplied Matlab installations)
   - **The material above should be uploaded as a SINGLE tarball per team**. The name of the tarball should be named as `TEAM<team number>.tar`. For example, team 1's submission should appear as `TEAM1.tar`.

4. **Evaluation:**
   - Time & Venue: **10 am - 1 pm on April 20, (Sat) in A-220.**
   - Each team gets 10 min:
     - **Presentation + Q&A: 3 min. + 2 min.**
     - **Demo (on TA's computer running institute-supplied Matlab): 5 min.**
   - Slots listed in the xls shared with this project on Google Classroom. **Submissions of teams that do not show up during the evaluation slot will not be graded.**

<u>Project Theme: DSP For Signals with Time-Varying Spectrum</u>

A classic usecase of DFT is to analyze and/or synthesize signals with *time-varying* spectrum, which is very useful where a *joint* time-frequency representation of a signal is necessary. The Short-Time Fourier Transform (STFT) that you saw in the class is one such example where the DFT size N can be thought of as the length of a *window* over which the spectrum of the signal is to be calculated.

In this project we look at two cases with time-varying spectrum:

1. **Natural Signals:** Signals such as speech, music, seismic signals, bio medical signals etc. where such variation is produced naturally. For example, in uttering the word "shallow", the sound at beginning will have a spectrum that comes from the "sh" sound (called a *fricative*), followed by the vowel sound "a". Intuitively, the fricatives sound like noise, so their spectrum is somewhat like radio static, i.e., all frequencies are roughly equally strong. On the other hand, vowels are known to have peaks at the resonant frequencies of the vocal tract (poles? :-)). One problem connected with natural signals is presented in **Problem 1**.

2. **Synthetic Signals:** Signals *synthesized* by manmade objects, e.g., communication systems, radars etc. where the frequency content of the signal is precisely engineered to achieve a certain design objective (e.g., greater range, greater noise immunity etc.). The wireless signal from your Bluetooth headset is a great example ("frequency hopping"). One such problem connected with synthetic signals is presented in **Problem 2**.

Problem 1: Pre-processing for Speech Recognition

Speech Recognition is ubiquitous: just ask Google Assistant :-) To improve feature extraction and classification accuracy by their ML algorithms, pretty much all speech recognition systems must remove unwanted signals from the user's speech using DSP algorithms. These *pre-processing* algorithms help the app function with background noise from traffic, appliances, etc.

The objective of this problem is to design and implement a simple denoising algorithm in Matlab, using the concepts of STFT and DFT from signals of the form $y(n) = x(n) + w(n)$ where $x(n)$ is the speech signal, and $w(n)$ represents different models for undesirable signals, that we will just call "noise".

1. **Part 1 (Suppressing noise of a known type)** You are given a noisy recording of speech sampled at 8 kHz that is a speech recording where $w(n)$ consists of bursts of tones. You are required to design an algorithm that will locate the tone bursts and attenuate them.
2. **Part 2 (Suppressing noise from unknown bands)** In this second part the speech signal, $x(n)$, is additively corrupted with an undesirable noise signal which could lie in spectral bands of 200Hz starting at 0Hz ("DC"). You are required to identify the noisy band(s) in the speech signal using a suitable method and suppress it so that the overall perceptual quality of speech improves.
3. In each case, your code will need to play the corrupted and the cleaned up speech signals.

Design criteria:

- Your approach must avoid altering the speech content as much as possible.
- The speech files `speech_with_beeps.txt` and `speech_noisy.txt` are only *examples* to help with the design. These should *not* be construed as the only test vectors. Both the tone suppression and denoising should work with a wide variety of typical usecases and together, with some variation in the tone type/amplitude as well as the type of noise.
- It would be ideal if you made no assumptions about the speakers: their number, gender, and the language and content of their speech. If you do, such assumptions should be clearly stated in your slides.
- To produce each output sample for the tone suppression algorithm (**Part 1**) you should require no more than 20 ms of samples (i.e., 160 samples at 8 kHz). In other words you are allowed to look no more than 20 ms into the future to suppress the noise in the current sample. This buffering mimics real time operation as it puts a limit on the time-lag("latency") between the signal input and signal output.
- It's okay to process denoising in **Part 2** offline and replay the speech.
- Note that the design should be able to suppress a variety of undesirable signals from typical environments – i.e., not restricted to the specific signal in `speech_noisy.txt`. Also, it should not assume only a specific number and gender of the speaker. These may be verified during the demo.

Problem 2: Simple Digital Communication System

Digital (as opposed to *analog* – AM, FM) communication techniques map a string of bits called a *message* (e.g., from a file) to a specially designed signal (*waveform*) for transmission over a noisy *communication channel* (e.g., optical fiber, free space). Received signals are typically corrupted by noise and hardware imperfections. We will design and implement simple DSP algorithms using DFT for such signals with time-varying spectrum.

1. **Part 1 (Detection).** Design a *detector* for the following simple digital communication system. The message is a string containing characters each with UTF-8 code between 0 to 127. The transmitter maps the (decimal) UTF-8 code $c \in \{0, \ldots, 127\}$ to a unique waveform $\exp(j(c + 128) \times 2\pi m/512)$, $m = 0, 1, \ldots, 511$. The sequences are sent in the order of the characters in the message without gaps. For example, a message 'AB' is mapped to

$$x(n) = x_A(n) + x_B(n - 512),$$

where the sequence $x(n)$ is 2×512 samples long and each of the sequences $x_A(m) = \exp(j(65 + 128) \times 2\pi m/512)$, $x_B(m) = \exp(j(66 + 128) \times 2\pi m/512)$, for $m = 0, 1, ..., 511$ and zero otherwise, stitched together as above (using a rectangular window in STFT terminology). The numbers 65, 66 come from the UTF-8 codes for 'A' and 'B' respectively. Note that each character (incl. spaces, full stops etc.) in the message is to be faithfully sent. An example of a message-$x()$ pair is provided in `example-prob2.mat`.

The **detector input** is a noisy version of transmit sequence given by $y(n) = x(n) + w(n)$ which is to be generated by the Matlab command `y = awgn(x,snr,'measured')`, where `snr` can take values from -20 to 10. The **detector output** is the character string that the algorithm detects from $y(n)$, without the knowledge of $x(n)$. The detector knows the *structure* of $x(n)$ above, but not the actual characters.

Part 2 (Frequency Synchronization). In practical scenarios the carrier frequencies for up and downconversion do not match: e.g., the transmitter (tx) may use 1000 MHz the receiver may use 999.95 MHz, resulting in a *carrier frequency offset* (CFO) of 50 kHz. A unknown CFO $\Delta\omega$ results in $y(n) = \exp(j\Delta\omega n)x(n) + w(n)$ where $w(n)$ was defined as above.

One way to design a receiver to account for CFO follows: (a) Estimate the unknown offset $\Delta\omega$ (b) Use this estimate to compensate $y(n)$ for the CFO to create a *CFO-compensated sequence* $z(n)$ and (c) input $z(n)$ to the detector from Part 1. A common way to simplify (a) is to **prepend** every message with a known *training sequence* at the tx. Suppose the training sequence 'TTTT' is prepended to the message sequence and transmitted as in Part 1 (i.e., e.g., a message 'AB C' with a training sequence becomes 'TTTTAB C'). The receiver knows this training sequence but not the message.

Design Criteria:

- Design algorithms to implement **Part 1** and **Part 2**. Input: complex sequence $y(n)$ (as a complex array, with or without CFO). Output: detected Unicode codes converted to characters.
- $\leq 10\%$ of the detected should be incorrect when `snr`=-5 in Part 2 and `snr`=-15 in Part 1.