Spectral Audio Processing using the STFT and Cross Synthesis

1.0 Aim

Understand the principles of operation of spectral audio processing using the short time Fourier transform (STFT) and implement Cross Synthesis using STFT

2.0 Learning Outcomes

You will be able to:

- Understand the principles of signal buffering and its use in STFT analysis
- Understand the conditions for perfect reconstruction using ISTFT.
- Implement STFT and ISTFT analysis-synthesis in MATLAB.
- Understand the principles of spectral modifications using STFT
- Understand the concept of spectral envelope and its estimation by linear prediction (LPC)
- Implement Cross Synthesis algorithm using STFT and LPC

A. Introduction

Short Time Fourier Transform performs FFT analysis on short windows in time. This is also called “sliding-window” FFT. The results of the FFT represent the contents of the audio signal in terms of time-frequency information. STFT processing has two main components: analysis and synthesis. We analyze sound in terms of spectral models primarily because:

- Time-Frequency (T-F) analysis is what the human brain does
- We may synthesize sound in terms of STFT models
- It allows processing and signal modification directly in the T-F domain
- It allows time-varying, adaptive and other non-linear signal modifications.

Understanding the concept of STFT requires firm understanding of the relations between FFT and filtering, as well as concepts of phasor amplitude and phase. We will experiment with different modifications of amplitudes and phases to demonstrate its acoustic “meaning”.

A. Short Time Fourier Transform

Short-time analysis and synthesis enables us to represent the spectra of signals with spectral profiles that change over time (which is the case for most “interesting” 1-dimensional signals such as speech and music). We can think of STFT as multiplying the signal x[n] by a short-time window that is centered around the time frame n. The segment of the signal contained in the window is analyzed using the DFT, which implies the evaluation of the Time-Frequency representation at a set of discrete frequencies where

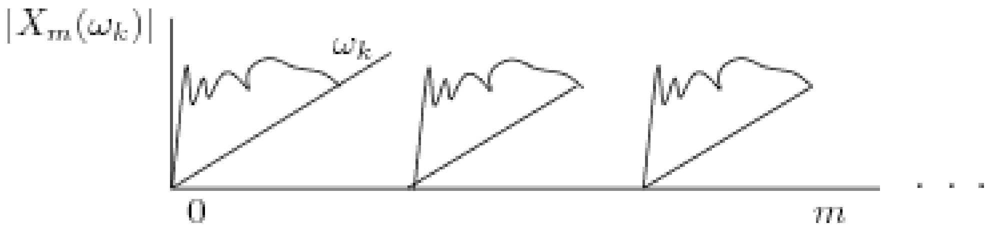
$$X[n,k] = \sum_{m=-\infty}^{\infty} x[m]w[n-m]e^{-j\omega_k m} = \sum_{m=-\infty}^{\infty} x[m]w[n-m]e^{-j2\pi nk/N}$$

A. Impact of window size and shape

The window serves several functions:

- It allows controlling the tradeoff between frequency resolution and side-lobe suppression (i.e. how sharp a peak in frequency is versus how high are the sidelobes)
- Using band-limited window allows better "localization" in time-frequency. Note that square window is not band-limited.
- To allow perfect reconstruction the windows must sum up to 1 (so called COLA condition, to be explained later)

In the OLA interpretation of the STFT, we apply a time-shifted window w[n-m] to our signal x[n], selecting data near time m, and compute the Fourier-transform to obtain the spectrum of the m-th frame. STFT is viewed as a time-ordered sequence of spectra, one per frame, with the frames overlapping in time.



B. Inverse STFT

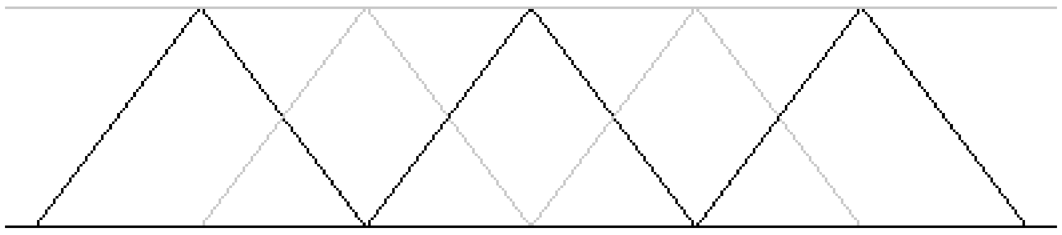
Because of the FFT and IFFT duality it is possible to reconstruct the original signal by summing the FFT.

x[n] ←-----------→ X[k]
x[m]w[n-m] ←-----------→ X[n,k]

Note that IFFT will recreate windowed signals, which then need to be summed together in a fashion the reminds the overlap-add method. The only difference is that instead of using rectangular windows we use tapering of the signal segments by the windowing function. If the windows sum up to one, this is called COLA condition

$$\boxed{\sum_{m\in\mathbf{Z}} w(n-mR) = 1, \ \forall n \in \mathbf{Z}.}$$

Summation of triangular windows give a constant level:



C. Spectral Modification using the STFT

It seems very natural to modify the sound properties by modifying parameters in the spectral domain. Actually we tend to think about filters as operations on frequency contents, so modifications such as low pass, high pass or bandpass are immediately translated to suppression of appropriate frequencies in the time-frequency domain. Using the relation between multiplication in frequency and convolution in time such modifications are possible, but there is an important caveat – since we are dealing with discrete time and discrete frequency signals, multiplication between DFT's is equivalent to circular-convolution and NOT linear convolution. We have investigated this topic in the previous set of notes in relation to fftfilt. We need to assure that the equivalent time domain impulse response for our desired frequency modification is time-limited so that we can use a long FFT with zero padding to turn the circular convolution into a linear (regular) convolution. Unless we can assure that, the result of manipulating the spectral contents of STFT will result in so called time-aliasing artifact, which is just another name for circular convolution.
Let us give an example: Assume you want to create a bandpass filter with W = [0.2 0.3]. We are using here a MATLAB notation for specification of filter response in frequency a common in filter design, such as the W parameter in fir1 command. W specifies the passband in terms of frequency range [0 1] that corresponds to [0 SamplingFreq/2].
A straightforward but incorrect way of doing such filtering is letting through only the FFT bins (rows in the STFT matrix) that correspond to these frequencies, which correspond to the STFT rows between

```
Passband = [round(0.2*nfft/2): round(0.3*nfft/2)]
```

and their counterpart in the negative frequencies (the frequencies above Nyquist).
In general when manipulating the STFT we will consider only the positive frequency part of the FFT bins that fall on bins numbers 1:nfft/2+1, and then replicate the design of that part in the negative frequencies range by copying the filter symmetrically. So if H is the frequency design in the positive frequency range (1:nfft/2+1), we complete it to the whole frequency range (1:nfft) by `H = [H; conj(H(end-1:-1:2))];`
Note: we use one more bin in the positive frequency range (fft bins (1:nfft/2+1)) since bin 1 corresponds to frequency of 0 Hz (DC) and the other bins are symmetric in the following fashion: bin 2 correpond to nfft, bin 3 to nfft-1 and so on. If the frequency design contains a phase specification, or in other words if H is complex, then the filter

response in the upper (negative) frequencies should be a conjugate of the lower part, to assure that the impulse response is real.

If we perform a filtering operation by cutting off the frequencies outside of the passband, as described above and go directly to synthesize the signal by ITFT, we actually implement an ideal passband filter, which has an impulse response nfft long . This means in fact that in terms of fftfilt operation we are doing the wrong thing since both the signal window and the filter impulse response are same length as the fft size. Doing such modification followed by ISTFT will create time-aliasing.

One remedy to the situation is to use an FIR spectral design methods instead of directly manipulating the STFT. As a matter of fact, a very crude method of filter design is going back to time from the frequency specification by inverse fft – this will create a finite impulse response of the length of fft, and then going back to frequency with a higher fft size that does zero the needed zero padding. This requires doing first an STFT with short nfft size for specifying the design in frequency, then going back to time to create a short impulse response, and finally going again into frequency domain with high STFT to achieve the modification with enough zero padding to avoid time-aliasing. This requires two extra fft steps, and it effectively performs two separate steps – STFT analysis as visul help for filter design, and second step that is equivalent to fftfilt to do the actual filtering. Of coure the filtering can be done directly in time domain using other filtering methods, including IIR filtering that can not be done by STFT.
The effect of different filter designs and time-aliaing in STFT domain will be demonstrated in class…

D. Spectral Envelope Estimation using LPC

Spectral envelope is a smooth curve that “envelopes” the graph of FFT magnitudes. It can be imagined as a string hanging of the peaks of FFT, giving an overall shape of the spectrum but not tracing the actual shape of the individual FFT bins. In fact, there are various ways to define and measure spectral envelope, also depending on the intended application. One of the most common usages of spectral envelopes is describing the so-called formant structure in speech. Formants are broad amplification areas in the frequency domain that are attributed to resonances of the vocal tract. Apparently our ability to understand speech and recognize phonemes of speech can be described by the tracing the frequencies of 2 or 3 lower formants. These formants are the peaks of a spectral envelope, disregarding or being insensitive to fine details of the spectrum. As a matter of fact, we can still recognize the vowel “a” even when the pitch changes and the actual partials or the details of the FFT are very different. What is common to all different vocalizations of a particular phoneme is that its spectral envelope, and it corresponding formant peaks, remain relatively constant, independent of pitch, or even whispered voice.
A lot of speech recognition deals with recognition of phonemes, and thus estimation of spectral envelopes. We will borrow one such method called “linear prediction”

What linear prediction tries to do is match a low order recursive filter to a given waveform. It does so by trying to create a predictor (this is where the name comes from) so that the next sample can be predicted from a short past. So prediction is in fact done by finding a short FIR filter that is operating on past samples that outputs an estimate of what is going to be the next sample. The difference between the predicted next sample and the true sample is called "prediction error" or "residual".
The reason for explaining this is to understand how prediction turns into signal approximation or modeling. In our application we want to use the modeling aspect of LPC, rather then prediction, in the following way:
Consider the following prediction equation
y'(n) = a1*y(n-1) + a2*y(n-2) + …+ap*y(n-p)
y'(n) is the prediction of y(n), so the error is e(n) = y(n)-y'(n)
Plugging this into the prediction equation allows us to write
y(n) = -a1*y(n-1) - a2*y(n-2) - …-ap*y(n-p) + e(n)
If we consider the error e(n) as an "input", and y(n) as an "output", what we have here is a recursive filter that creates the current output y(n) from a combination of previous outputs y(n-1), …., y(n-p) and an input signal that is the residual "noise". In other words, we created a recursive filter the is driven by noise that tries to recreate the original signal. IIR filters that have only recursive part are also called All-pole filters or Auto-regressive (AR) filters. (Note- FIR filters are sometimes called Moving Average (MA) filters, so a general IIR filter that has both poles and zeros is sometimes called ARMA).

Spectral envelope estimation using LPC:
Linear prediction is realized in matlab using lpc command. The output of lpc is a set of filter coefficients (1, a1, a2, …, ap) where p is the order of lpc that is specified by the user. The letter "c" in lpc stands for "coefficients".
If we want to describe the filter in terms of transfer function in frequency, we consider Y(z) as the output (using the z- method to translate time to frequency using the delay operator), and white noise of as input. Since, for reasons not explained here, the spectral amplitude of white noise is flat in frequency, we assume that the input is identically 1 for all frequencies. Since e(n) (out noise input in the signal approximation approach) might be of some power e that is not unit, we actually assume that e(n)=e*u(n) where u(n) is a unit variance white nose. Doing all that gives an expression for Y(z) as

$$Y(z) = \frac{e}{A(z)},$$
$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_p z^{-p}$$

In our Cross synthesis application we will consider the amplitude of Y(z), i.e. abs(Y(z)) as a model of the spectral envelope of the modulator speech signal. We will use this spectral envelope as a time-varying filter that will modify the STFT of another signal, in our case a musical "carrier" signal, to create a TalkBox.
It should be noted that lpc is done on short frames, so that the filter A(z) is re-estimated adaptively over time. Applying such time-varying envelope to a musical filter creates a time varying filter that creates the talking effect.

Final Project: TALKBOX

Your final "big" programming project will be implementing Cross Synthesis between voice and musical signal. To do that you need to implement the following stages of the algorithm:

STFT:
Input: audio signal
1. Dividing a signal into windowed frames to create a "buffer" matrix
2. Performing FFT on every windowed signal segment
Output: Matrix of FFT vectors

ISTFT:
Input: Matrix of FFT vectors from STFT
1. Performing IFFT on the input matrix
2. Summing the resulting signals in overlap add fashion
Output: audio signal

Envelope:
Input: Audio signal
Output: A matrix of spectral envelopes

Cross synthesis is achieved by the following procedure:
Using two signals, called source and modulator (source is the music, modulator is speech) do the following:
1. Perform STFT analysis of the source
2. Perform Envelope analysis of the modulator
3. Multiply the modulator STFT matrix by speech envelope matrix
4. Perform ISTFT on the resulting matrix

Details to be considered:
The envelope has to be "smooth enough" so that its impulse response is short enough to avoid time-aliasing, as described in the Spectral Modifications section.

Steps to fulfill the project:

1. You will be provided a first sketch of the windowed buffering function winbuf.m The function creates a matrix where every column is a windowed segment of the original waveform. The signal contents in the columns overlap according to parameters specified in winbuf function.
2. This function can be modified to create an OLA procedure that reconstructs the original sound samples from the buffer matrix. The input is the buffer and hop size. Name the function ola.m
3. FFT can be used on the results of 1, or FFT can be performed on every column of the buffer in the winbuf loop. Write a new function called STFT can be written by modifying the function from 1 so that both buffering and FFT are performed in that one function. The size of the fft is an additional parameter.

4. IFFT can be done on the output of step 3 (FFT) before step 2 (OLA) or can be written inside a function where IFFT is done first and OLA are done second. In such a case the size of the fft can be determined from the size of the columns of the input FFT matrix, but the parameters of the window size and overalp (or hop) need to be provided in order to produce successfully the OLA. This function should be named ISTFT.
5. Read the help of lpc in MATLAB. When lpc function is applied to the buffer matrix (the output on winbuf) of the modulator signal, it produces a matrix of lp coefficients, where each column is an lpc of the corresponding windowed segment (a column of the buffer matrix).
6. The spectrum of the lp filter is be obtained by taking the absolute value of the FFT of the lp coefficient matrix (again, FFT will operate on the columns). Since we are interested in spectral envelope that describes the amplitudes only, we remove the phase by taking an absolute value of the complex matrix that we got from the fft
7. Calculate the spectral envelope as 1/(lp spectrum). Explanation: In previous step we got the spectrum of the prediction filter. As explained in the notes, this is the reciprocal of the spectral envelope. Write a function that combines step 5-7 into a one function Envelope. It receives signal as an input, window size, hop, nfft and size of lpc analysis. This function outputs a matrix of spectral envelopes. Call this output matrix ENV. This envelope operation is performed on the "modulator" signal.
8. The ENV matrix is multiplied pointwise (.*) with the STFT of the "carrier signal"
9. The resulting modified signal is inversed FFT's (use ISTFT from step 4).
10. Done!

Submission instructions:
1 - write the ola.m function and test it to show that winbuf and ola are inverse one of the other.
2. write stft.m and istft.m and show that they are inverse of each other
3. write envelope.m that performs lpc analysis
4. make an xsynth program that calls stft, lpc, makes the modification and does istft

Please zip and email the following to cdwarren@ucsd.edu by [due date] at midnight.
[yourname]diary <---- diary of testing the different function and running xsynth
[yourname]ola.m <---- your cross synthesis code
[yourname]stft.m <---- your stft code
[yourname]istft.m <---- your istft code
[yourname]envelope.m <---- your envelope code
[yourname]xsynth.m <---- your cross synthesis code
[yourname]xsynth.wav <---- a musical example using this code
[yourname]sourcesound.wav <---- source sounds for xsynth
[yourname]modulatorspeech.wav

Please zip and email the following to cdwarren@ucsd.edu by May 1 at midnight.