Assignment 3 COMP 546

Instructor: Mike Langer

Posted: Friday, March 29, 2019.
Due: Sunday, April 14, 2019 at midnight

Instructions

- Submit a single zip file containing your PDF, audio files, and any code that you wrote.
- The assignment has a total of 100 points.
- Late penalty 20 % of your score per day, up to three days.
- For all figures that you submit, label the axes. Be sure to give meaningful definitions of frequency.

Warmup (not graded)

Before starting the questions below, you should spend some time examining at how Matlab computes Fourier transforms. You will use the function `fft` ("fast Fourier transform") which computes the Fourier transform quickly.
First, define vectors `v` of length say 4 or 8 and examine the Fourier transform of these vectors, e.g.

- `[1 0 0 0 0 0 0 0]`, `[0 1 0 0 0 0 0 0]`, etc
- `[1 1 1 1 1 1 1 1]`

Observe the following for your examples:

- the conjugacy property
- the amplitude spectrum of these functions, which is computed by `abs(fft(v))`
- the phase spectrum, which is computed by `angle(fft(v))`

Next example: synthetize a "white noise" sound that is defined by independently and identically distributed Gaussian noise. (Use Matlab `randn`). You can play it on your laptop with the Matlab function `sound(y,Fs)` where `Fs` will be the sampling rate, typically 44100 per second. Compute the amplitude spectrum of this sound and observe that it is very bumpy, but on average it is flat, i.e. if you were to blur the amplitude spectrum, then it would be flat rather than decreasing or increaseing. Stated another way, if you were to generate many such random white noise functions and average the amplitude spectra together, the average would be quite flat. Verify this for yourself.

For further Hints on using Fourier transforms in Matlab using `fft`, see the end of this document.

Questions

1. **(30 points)**

   Consider the local averaging function from Exercises 16,

$$B(x) = \frac{1}{4}\delta(x+1) + \frac{1}{2}\delta(x) + \frac{1}{4}\delta(x-1)$$

   The function $B(x)$ has coefficients that sum to 1 and the coefficients are positive. Verify for yourself that if you convolve $B(x)$ with itself, then you get a new function whose coefficients are still positive and again sum to 1. (To understand why, see Exercise 16 Q 6.)

   One can show that convolving B with itself m times gives a function that is approximately a Gaussian,

$$G(x, \sigma_m) \;\approx\; B * \cdots * B(x)$$

   where the standard deviation σ_m grows with $\sqrt{m}$, and the approximation improves for larger m. Verify this for yourself, e.g. use

```
B = [.5 .25 0 0 0 0 0 0 0 0 0 0 0 0 0 0.25];
cconv(B,B, length(B))
```

   where `B(0)` is at index 1 and the vector is considered to be periodic.

   Recall the local differencing function $D(x)$ – again Exercises 16 – and define it similarly so `D(0)` is at index 1 and the vector is considered to be periodic.

   Your task:

   Let functions $B(x)$ and $D(x)$ both be defined on a much larger set of samples than in the above example, namely $N = 256$. Compute the functions:

$$B * \cdots * B, \qquad D * B * \cdots * B, \qquad D * D * B * \cdots * B$$

   where the number of convolutions m is 2^i, where $i = 0, \ldots, 5$. These three functions are approximately a Gaussian, a first and second derivative of a Gaussian, respectively. Give a table showing:

   - the standard deviation σ_x of the Gaussian
   - the frequency $k \in 0, \ldots, \frac{N}{2}$ at which the amplitude spectra of first and second derivatives of a Gaussian are maximal.
   - the octave bandwidth at half height

2. **(30 points)**

Typical audio signals are not periodic, but the Fourier transform treats them as if they are. This can lead to artifacts when the signal at the left and right boundaries of a window ($t = 0$ and $T - 1$ respectively) have very different values, as this essentially produces an "edge" in the signal. This is a problem when one performs the Fourier transform on a short duration signal where T is relatively small.

As an example of the sort of problem that can occur, consider a signal that is a linear ramp from $t = 0$ to $T - 1$. We would intuitively expect such a signal to consist of very low frequency components. Yet the discontinuity at the "edge" going from $t = T - 1$ back to 0 would introduce high frequency components.

A common way to reduce this edge artifact is to *multiply* the signal by a weighting function that goes to 0 at $t = 0$ and at $t = T - 1$. One often refers to this weighting function simply as a "window". For example, in 1D image with T samples, the window might be defined:

$$W(x) = 1 - \cos(\frac{2\pi}{T}\ t).$$

Windowing introduces it own artifacts, however. To understand them for a particular signal $I(t)$, one can use the following property of the Fourier transform which is similar to the convolution theorem:

$$\mathbf{F}\{I(t)\ h(t)\} = \hat{I}(\omega) * \hat{h}(\omega).$$

Your task:

(a) For the specific case of the raised cosine window $W(t)$ above, what is the effect of multiplying this window on the Fourier transform of the signal? To answer this question, you need to compute the Fourier transform of the window.

Demonstrate this effect for the following two specific signals, using Matlab:

(b) a sine function whose wavelength is much larger than the window width T (e.g. the sine function's wavelength is $8T$); for your example, place the window over a region of the sine that is increasing roughly linearly.

(c) a sine function whose wavelength is much smaller than the window width T e.g. the sine function's wavelength could be $\frac{T}{8}$.

For each case, plot the original signal $I(t)$, the windowed signal $I(t)\ h(t)$, the amplitude spectrum of the original signal, and the amplitude spectrum of the windowed signal. All computations and plots should be over $t = 0$ to $T - 1$ only.

3. (**40 points**)

This question will give you some experience with *spectrograms*. (I did not have time to cover spectrograms at the end of lecture 19. See the slides and the corresponding lecture notes. You will need them for this question as they are part of the course.)

Your Task:

To do this question, you will first need to write a Matlab function or script that creates a spectrogram of a sound. Matlab has a function `spectrogram`, but I do not want you to use it. Instead you will write your own. Just follow the definition given in the slides/notes.

Your spectrogram code should have a variable `blocklength` that specifies how many samples there are in each block. Your signal will be partitioned into blocks of this length. Your frequencies will be defined up to `blocklength`/2, i.e. cycles per block.

A few important notes for your spectrogram plots **(Added March 31)**:

- Sound intensities (amplitudes) are often expressed in a log scale (called decibels or dB). It typicaly be helpful to do this mapping to see structure in spectrograms of real signals, so at least try this.
- The time and frequency axis units should be milliseconds and Hz, respectively, rather than block number and cycles per block. See `https://www.mathworks.com/help/matlab/ref/xticks.html`

Construct or record the sounds described below in (a)-(c), and use your Matlab function or script to make a spectrogram of each of the sounds. Include the spectrograms in your PDF.

Briefy describe the features that you see in each spectrogram, or features that you were expecting to see but don't see. Use two different `blocklength` *values to identify different features for each of the three cases.*

Here are the three sounds:

(a) **Chirp:** Synthesize a sound signal called a *linear chirp*:

$$I(t) = \sin(\frac{2\pi t}{F_s}(\omega_0 + \frac{\omega_1 - \omega_0}{2\ T}t))$$

where F_s is the signal sampling rate (44,100 samples per second) and T is the duration of the signal $I(t)$ in units of samples (not seconds). It can be shown that this function will have continously varying frequency, starting at ω_0 cycles per second at $t = 0$ and finishing at ω_1 cycles per second at the T seconds. Note that there actually are two t's in the argument of the sin, i.e. it is not a typo.[1]

For example, the frequency might fall from $\omega_0 = 15$ kHz at $t = 0$ to $\omega_1 = 1000$ Hz at $t = T$.

[1]You can think of the *instantaneous frequency* at time t as the derivative of the phase of the sine function with respect to time.

(b) **Buzz:** Synthesize a sound signal that consists of a shifted sequence of $\delta(\ )$ functions i.e. pulses such that the duration between consecutive pulses decreases continuously over time, i.e. the pulse frequency rises. There are many ways to define such a sequence, and you will need to come up with one. The resulting sound should start out like a buzzing (like a zipper) and over time the frequency of the buzzing will increase, and eventually it will transition from a buzz into a smoother sounding tone, which gradually increases in frequency. The duration of the whole sound should be no more than ten seconds.

Hint: to play the sound with Matlab's `sound` function, you may need multiply the $\delta()$ functions by some constant.

Briefly describe how you made this sound e.g. give a mathematical formula. Be sure to submit a sound file.

(c) **Vowel:** Record yourself making a *voiced vowel* sound, namely one of a-e-i-o-u.

Note that for this example, most of the energy of the sound will be in lower frequencies, so you may wish to expand that part of your plot in order to see the structure there.

For each of the sounds, be sure to try two different `blocklength` values, and compare results. Also be sure to submit a sound file for each case so that the T.A. can easily listen to it, without running your code. For (a) and (b), use `writesound`. For (c), you need to record a sound e.g. using the sound recording tool on your cell phone or laptop. You can then read it into Matlab and play it e.g.

```
[y,Fs] = audioread('mysound.wma','native');
sound(y,Fs)
```

where `Fs` will be the sampling rate.

Appendix: Hints for using `fft`

The `fft` function takes as input a $1 \times N$ matrix and considers values at indices $1, \ldots, N$ to hold x (or t) values $0, \cdots, N-1$. Note that this leads commonly to programmer "off by one" errors.

Typically we work with filters that are defined not just on positive values of x but also negative values. One can define such a filter in Matlab in two ways:

- Consider indices 1 to $\frac{N}{2}$ to hold the value $x = 0$ to $\frac{N}{2} - 1$, and indices $\frac{N}{2}$ to N to hold values from $x = -\frac{N}{2}$ to -1. For example, suppose `N=8`. You could define your local difference `D` by `[0 -1 0 0 0 0 0 1]`.
- Think of the filter as periodic, and consider indices 1 to N to hold values corresponding to $x = -\frac{N}{2}$ to $\frac{N}{2} - 1$, so that the origin $x = 0$ is at array index $\frac{N}{2} + 1$. Then apply the Matlab function `fftshift` which shifts the array to the scheme above.

