2020 ASAS Homework 2: Convolution and Linear Time-invariant Filtering

Prof. Yi-Wen Liu

March 12, 2020

NOTE: This week I chose Max MSP as the software/programming environment to complete the homework, mainly for its real-time signal processing and imaging capabilities. Max MSP is “free” if you only want to run programs (no saving or editing). You can download the latest and greatest, but the programs I created are tested on version 7. <https://cycling74.com/downloads/older>

1. **Averaging and “differencing”** (Remarks: Yes, here we are going to use “difference” as a verb). Please first find a piece of music or any sound and read it into MATLAB. Denote the signal as 𝑥[𝑛].
2. Check the sampling rate, the number of channels, and the number of bits per sample.
3. The simplest low-pass filter is to average over consecutive samples; that is, let

𝑦[𝑛] =

1 𝑝

∑ 𝑥[𝑛 − 𝑚 + 1] ,

𝑝 𝑚=1

where 𝑝 is the order of the filter, and 𝑦[𝑛] is the output of the filter. Implement this filter and listen to the output. Increase 𝑝 from 1 to 10 to see if you can hear the difference.

(a)-(b) See program 1a-d.maxpat. I have chosen to implement the simple low-pass averaging filter over a buffer updated in real-time. That is, the buffer is the block of m signals averaged through x[m:n]. Though simple, the filter works very well, offering a smooth roll-off of higher frequencies through p=10 and well beyond. A fun trick is to send the filtered signal to one ear and the unfiltered signal to the other ear on a pair of stereo headphones. As we increase m, the unfiltered signal is perceived as being panned towards the ear receiving its signal, whereas the filtered signal is more and more masked by its full spectrum sibling.

1. Check your result in (b) against the following results. They should essentially be identical.

y\_conv = conv(x, 1/p\*ones(p,1)); Unfortunately, I went to Academia Sinica to install Matlab (the key they provided was invalid) and for some reason they still need more time to “prepare” the installation. Apologies, but I will have it hopefully for certain next assignment.

1. In fact, the array ones(p,1) can be regarded as the impulse response of this **FIR** (finite impulse-response) filter. Denote h = 1/p\*ones(p,1) and use freqz(h) to plot the frequency response of the filter.

See program 1a-d, 1e-f and 1e-f2. I have plotted the real-time frequency response to the signal in 1a-d. In 1e-f and 1e-f2, you may select between H(p,1) and H(y-m,1) to convolve the signal. 1e-f and 1e-f2 also have a static plot of the frequency response, which should compare with freqz(h).

1. Now, define 𝑦𝑑 = 𝑥[𝑛] − 𝑥[𝑛 − 1]. Of course this can be done with a simple for loop, but alternatively, please use conv() to do the job.

In my implementations in 1e-f (autocorrelation?) and 1e-f2 (convolution), I chose to implement dynamically resizable convolution windows that follow and precede x[n], respectively. This I did to experiment with the effect, lower the cpu usage from the very expensive full 2N-1 overlap, and also naively implement correlation1. I am not very familiar with designing correlation filters, but when mixing Guassian noise with higher deviations with a periodic phasor signal, the phasor signal emerges strongly with larger windows and is fairly obscured at the lowest window (block n = 1). This was interesting for me. (Note: I did include the full overlap in file 1e-f2\_full, but the result was also less pleasant in addition to being so cpu intensive. Different real-time DSP convolutions2 also seem to deviate from the textbook convolution3).

1. Set x = 0.1\*randn(A\_CERTAIN\_LENGTH, 1) so it is an instance of Gaussian white noise with mean zero and standard deviation 𝜎 = 0.1. Listen and compare the result before and after differencing. Does it feel more unpleasant before or after averaging or differencing? Describe how you feel about it and explain, perhaps, the reason why.

I found the simple averaging filter to be the most pleasant, as it very smoothly rolls off the noisy high frequencies. This gives the sensation of moving the noise source to a distance and ‘view’ that is pleasant. Recalling the waterfall demonstration in class, it is as if one is too close for comfort, just averaging more is like walking backwards some until the intensity is just right. The convolution and autocorrelation are sharper (the latter somewhat moreso), with some jagged edges to the noise, likely due to the displacement between samples caused by multiplication. There is also a slightly more musical quality to the convolution filter, as it allows shaped bands of noise at different window sizes.

**Remark**: When you listening to a signal stored in a vector, make sure that its range is between ±1 to avoid clipping effect. Or, alternatively, use soundsc() to avoid clipping. Always remember to specify the sampling rate.

1. **Infinite impulse response (IIR)**. Take any audio signal 𝑥[𝑛] and for any 𝑛 > 1

implement the following by a for loop,

𝑦[𝑛] = 𝛼𝑦[𝑛 − 1] + 𝑥[𝑛],

Where 0 < |𝛼| < 1 is a constant. You can assume 𝑦[1] = 0.

1. Check your result against y = filter([1], [1 -alpha], x). Next time!
2. Create an instance of Gaussian white noise for about 1 second long. Listen and compare the result before and after filtering.

See program 2a. Similar to the other programs, I implemented this with a resizable window, allowing the filter to also serve as an averaging filter, if desirable, with the infinite term. I found that 𝛼𝑦 mainly adds energy to the percept that tails back to the original input. As the window, or chain, is enlarged, this ‘tail’ is further tapered. To my ears, it adds a third, intermediary spectrum between the attenuated noise towards the floor and the pitched band.

1. Create a periodic impulse train:

𝑥[𝑛] = {0.5, if 𝑛 = 80𝑚

0, otherwise.

Where 𝑚 is an integer. In other words, 𝑥[𝑛] is non-zero at every 80th sample. Listen and compare the result before and after filtering.

Sound-wise, the effect is not terribly different than what was previously described, with the exception of the source being pitched at around 275.6 hz. Being impulses, it is not a generally pleasant sound, and thus it is nicer to roll it off, though it never quite gets nice. What is interesting is the plot, which shows the effect on the higher frequency impulses as 𝛼𝑦 and the window are manipulated. It seems to show what I described previously, that 𝛼𝑦 adds an energy and ‘tail’ back towards the source impulse.

1. Discuss how to restore 𝑥[𝑛] from 𝑦[𝑛]. (Hint: if 𝛼 is known, it is quite easy. However, if 𝛼 is unknown, you may need to make assumptions.)

As intimated in the question, and already shown in class, if 𝛼 is known, restoring x[n] becomes a matter of rewriting the equation as a function of X. In the case of an unknown 𝛼, our group, led by a member of your lab whose name I cannot grasp, came up with an intriguing solution in which the average of x[n] is assumed to be zero, and 𝛼 is held to be the scalar responsible for the average difference between |y[n] – y[n-1|. Though I contributed nothing to this idea, I will hazard a few points on the assumptions: one is that we should seem to gain more by stating we expect the mean of x[n] to be 0, less the signal be a strictly positive or negative signal. Another is, if we know x[n] to be a periodic binary impulse, for example, then we know that the average of x[n] will never be zero, but approach zero as the period between impulses becomes larger.

References

1. Gareth Loy and John Chowning. 2007. Musimathics, Volume 2: The Mathematical Foundations of Music. The MIT Press.
2. <https://cycling74.com/forums/convolution-in-gen-again>
3. <https://www.mathworks.com/help/matlab/ref/conv.html#d118e219932>