Assignment 10
Linear and Circular Convolution

April 3, 2019

One of the main uses of the DFT is to implement Convolution. As you know from your course in DSP, the convolution is defined as

$$y[n] = \sum_{k=-\infty}^{\infty} x[n-k]h[k] \tag{1}$$

In fourier space, this becomes

$$Y[m] = X[m]H[m] \tag{2}$$

which is a major simplification. We will only look at convolution of causal signals, so the summation in Eq. 1 is from $k = 0$ to $k = \infty$.

Linear Convolution using Direct Summation

The summation in Eq. 1 can be implemented directly. We assume that the filter, $h[k]$, has coefficients from $k = 0$ to $k = N-1$. So we can have an algorithm that implements Eq. 1 as follows:

```
for n corresponding to x: # iterate over indices in x
  y[n]=0;                 # initialize y to zero
  for k in range(K):      # iterate over indices in h
    y[n] += x[n-k]*h[k]   # compute the convolution sum
  #end for
#end for
```

We initialize y to zero since the sum started at $k = 0$. If y had any initial value, it would have decayed by the time we reached $t = nT$.

Computational Cost

How many computations are required? One multiplication and one addition per index in h. So the total cost is

$$\text{cost} = N \times K \text{ multiplications and } N \times K \text{ additions}$$

Typical FIR filters have 50 taps, and so the cost can be quite prohibitive.

In python, how would you optimize this computation? Can you eliminate one or both for loops?

There is a library function, `np.convolve`, that implements this operation.

Start and End of sequence

The above algorithm is fine for an infinite sequence $x[n]$. What happens if $x[n]$ was a finite sequence? Let us assume that x is present from n_1 to n_2, $n_1 \leq n_2$

- Consider any n. Then,
$$x[n-k]*h[k]$$
has terms for $0 \leq k < N$ and for $n_1 \leq n-k \leq n_2$, i.e., for $0 \leq k < N$ and $n_1 + k \leq n \leq n_2 + k$. Since we assume k to be non-negative, the summation has terms for
$$n \geq n_1 \text{ and } n \leq n_2 + N - 1$$
- For which n does $y[n]$ use the full set of terms in the summation? Clearly this is for $n_1 + N - 1 \leq n \leq n_2$
- If we pad $x[n]$ with zeros on both sides, the original algorithm will continue to work (though with some unnecessary additional calculations)

Circular Convolution using the DFT

You will have learnt that the DFT also supports a convolution. This is defined by (using Oppenheim's notation)

$$\begin{aligned}\tilde{x}[n] &= \frac{1}{N}\sum_{k=0}^{N-1}\tilde{X}[k]e^{j(2\pi/N)kn}\\ \tilde{X}[k] &= \sum_{n=0}^{N-1}\tilde{n}[n]e^{-j(2\pi/N)kn}\end{aligned}$$

where the˜in $\tilde{x}$ means the sequence of N values has been extended into an infinite periodic sequence, so that

$$\tilde{x}[n] = \begin{cases} x[n] & 0 \leq n < N \\ x[n-N] & N \leq n < 2N \\ x[n+N] & -N \leq n < 0 \end{cases}$$

The filter $h[k]$ does not need to be extended, as it is used without shifting. But we can use $\tilde{h}[k]$ without any harm. Then, the convolution of $x[n]$ and $h[k]$ is given by

$$\begin{aligned}\tilde{y}[n] &= \sum_{m=0}^{N-1}\tilde{x}[n-m]\tilde{h}[m]\\ &= \sum_{m=0}^{N-1}x[(n-m)\text{modulo}N]h[m] \end{aligned} \quad (3)$$

Taking the DFT, this becomes

$$\tilde{Y}[k] = \tilde{X}[k]\tilde{H}[k] \quad (4)$$

How long a sequence is needed for $\tilde{Y}$ to be correct? The first window is moved over the second (according to our definition. Oppenheim uses the opposite convention and moves the second window over the first). The length of the resulting sequence is the sum of the two lengths less by one. For the above case, it is $2N-1$.

What if length of $\tilde{x}$ plus length of $\tilde{h}$ is larger than $N+1$?

While this is nice, it is not very practical, since we rarely want to convolve two periodic sequences. What we do have are filters with finite impulse responses (FIR filters) and infinite sequences to be filtered by these filters. That is covered in the next section.

Computational Speed

Equation 3 costs N^2 multiplications and a similar number of addtions. But Eq. 4 Costs $N\log_2 N$ to get the DFT of $x[n]$, another $N\log_2 N$ to convert back to $y[n]$ (We must do the same for $h[k]$ but this has to be done only once). Finally we need N operations to compute the convolution in frequency domain. The total cost is only $2N\log_2 N + N$ multiplications and additions, which is much less than the N^2 cost for Eq. 3.

Using the DFT to implement Linear Convolution

Suppose now we have a low pass filter. We want to apply it to a speech signal. The speech signal is very long (say 5 minutes worth at 44 Ksmp/sec, or $13,200,000$ samples). The filter is an FIR filter, with 32 taps.

If we do linear convolution this is going to cost us $1.32\times 10^7 \times 32$ operations, or 422 million operations in all.

If somehow we could do this using FFTs, we could do it in less than 150 million operations.

But how to do it?

Circular Convolution as Linear Convolution with Aliasing

- Suppose $h[]$ fits into a 2^m window. We zero pad the $h[k]$ if required.
- Divide $x[]$ into sections 2^m long.
- Apply circular convolution to each section of $x[]$.
- Stitch the outputs back together.

The problem with this approach is that each convolution is longer than the section of $x[]$. So, we actually have to have $x[]$ sliced into smaller sections so that if L is the length of the $x[]$ sections, and P is the length of $h[]$, then $N = L + P - 1$.

This now gives us N samples of $y[]$, the samples corresponding to the convolution of $h[]$ with a section of $x[]$ that is L samples in size. So which of these samples would agree with the $y[]$ obtained from linear convolution?

The linear convolution corresponds to moving $h[]$ over $x[]$ and summing the portion that overlaps. i.e.,

$$y[n] = \sum_{k=0}^{P-1} x[n-k]h[k]$$

So the first $P-1$ samples of $y[]$ are incorrect since some of the $x[]$ values don't exist in the slice we used. Similarly the last $P-1$ samples are also wrong for the same reason (they need future $x[n]$). **The middle samples are correct.**

So what can we do? We first save the last $P-1$ samples (call them $w[0]$ to $w[P-2]$) of the previous convolution. The current slice (of length $L = N+1-P$) is now convolved with $h[]$.

There are $P-1$ zeros prior to the first $x[n]$ value in the slice. In the convolution, these do not correspond to any $x[n]$, since we have L values per slice of $x[]$ and have $P-1$ extra places that are loaded with zeros. Call the output of the circular convolution for these places as $u[0]$ to $u[P-2]$.

Then, the last $P-1$ values of the previous convolution are given by

$$y[m + (L-P+1) + k] = w[k] + u[k]$$

Following these, we have $L-P$ correct y values. And finally we have $P-1$ more entries that are incorrect, waiting for the next circular convolution to correct them.

In each circular convolution, we use L new x values, and get L new y values.

The best way to understand this is to draw diagrams which will be done in the Lab Lecture. (You can also go through the DFT chapter of Oppenheim and Schafer, though that is a little obscure).

Computational Cost

We do one N-point DFT, N multiplications and additions, and an N-point inverse DFT. After that we add w and u values ($P-1$ of them). So the total computational cost (only counting multiplications) is

$$\text{cost to get } L \text{ values} = 2N\log_2 N + N + (P-1)$$

Direct linear convolution of L samples is $L \times P = (N-P+1)\times P$

As long as $2\log_2 N + 1 < P$ the circular convolution is faster than linear convolution.

Assignment

1. ✓ Download h.csv from Moodle. The file contains coefficients for an FIR filter.
2. ✓ Plot the magnitude and phase response of the filter and infer the properties of the filter.

   Read up `scipy.signal.freqz()` to plot the magnitude and phase response of a digital filter.
3. ✓ Generate the following sequence and plot:

   $$x = cos(0.2*pi*n) + cos(0.85*pi*n) \tag{5}$$

   Where n is 1, 2, 3, ... 2^{10}
4. ✓ Pass the sequence x through the filter using linear convolution and plot the output.

   What do you observe?
5. ✓ Now repeat process using circular convolution. Plot the output.
   What is difference do you observe?
   Note that computational cost of circular convolution is less in frequency domain.
   Use the DFT technique discussed to do the convolution.
   Note: You will have to zero-pad the filter appropriently.
   ```
   y1=np.fft.ifft(np.fft.fft(x) * np.fft.fft( np.concate(( h,
   np.zeros(1,len(x)-len(h)) )))
   ```
6. ✓ Now, do the linear convolution using circular convolution.
   Refer page 3.
7. **Circular Correlation**
   In this section, we will take an example sequence which is widely used in communications called as Zadoff-Chu sequences. They are named after Solomon A.Zadoff and D.C.Chu.
   Read the Zadoff–Chu sequence from x1.csv.
   Properties of Zadoff-Chu sequence:

   (a) It is a complex sequence.

   (b) It is a constant amplitude sequence.

(c) The auto correlation of a Zadoff–Chu sequence with a cyclically shifted version of itself is zero.

(d) Correlation of Zadoff–Chu sequence with the delayed version of itself will give a peak at that delay.

Correlation can be written as conv(a(n),b(-n)) which in frequency domain is A(jw).*conj(B(jw)).
Correlate x1 with a cyclic shifted version of x1, with a shift of 5 to the right.
Plot the correlation output. Note that this is a complex sequence hence you need to plot the magnitude.
What do you observe?