Tutorial — March 28, 2005

The Sliding DFT

1 Motivation

Given a discrete-time signal, which can be considered as a sequence

$$\ldots, x(-2), x(-1), x(0), x(1), x(2), \ldots, x(n), x(n+1), \ldots \tag{1}$$

In some applications, one may wish to analyze the signal using a fixed-length sliding window. For example, suppose at the time instant n, a windowed sequence $\mathbf{x}(n)$ consisting of N samples, is formed as follows

$$\mathbf{x}(n) = \{x(n-N+1), x(n-N+2), \ldots, x(n-1), x(n)\} \tag{2}$$

and an N-DFT operation is performed on $\mathbf{x}(n)$ to yield

$$\mathbf{X}(n) = \{X_0, X_1, \ldots, X_{N-2}, X_{N-1}\}. \tag{3}$$

Hence, at each new time instant n, a new DFT-sequence $\mathbf{X}(n)$ is computed (compare to the previously examined short-term Fourier transform method). Evidently, an FFT operation can be used to evaluate the N-sequence $\mathbf{X}(n)$.

What is important to note is that the FFT is "fast" or computationally efficient when ALL the N values of $\mathbf{X}(n)$ are needed. But suppose, in a certain scenario, we are only interested in the kth value of the DFT, i.e., X_{k-1}, then the FFT seems to be wasteful. Essentially, with the FFT, we have to compute the entire DFT-sequence and discard the unwanted values. Is there a better solution? In fact, there exists a more desirable method, known as the Goertzel algorithm, to obtain an isolated X_k (see Sec. 9.2 in your Oppenheim text). In addition, when we are considering a sliding-window scenario described above, an algorithm known as the sliding DFT (SDFT) is particularly efficient.

Basically, the SDFT comes from the observation that for two successive time instants, say $n-1$ and n, the windowed sequences $\mathbf{x}(n-1)$ and $\mathbf{x}(n)$ contain essentially identical elements. For example, if the window length N=16, then at the time instant $n = 15$, the sequence

$$\mathbf{x}(15) = \{x(0), x(1), \ldots, x(14), x(15)\} \tag{4}$$

is fed to the 16-DFT. And at $n = 16$,

$$\mathbf{x}(16) = \{x(1), x(2), \ldots, x(15), x(16)\} \tag{5}$$

is used in the 16-DFT computation. The striking similarity between $\mathbf{x}(n-1)$ and $\mathbf{x}(n)$ is exploited in the SDFT for computational efficiency.

2 Method

Before presenting the SDFT, we recall the following DFT property. If N-DFT of $x[n]$ is $X[k]$ then

$$x[((n-m))_N] \quad \overset{N-DFT}{\longleftrightarrow} \quad W_N^{km} X[k] \tag{6}$$

where $W_N = e^{\frac{-j2\pi}{N}}$. Thus, (6) shows the N-DFT of a circularly shifted sequence. In particular, if a sequence is circularly shifted by one sample (to the left), then the DFT value X_k becomes

$$X_k \rightarrow X_k e^{j2\pi k/N}. \tag{7}$$

Now let us consider more closely how to relate two successive windowed sequences, such as those in (4) and (5). In fact, regardless of the value of N, one can obtain $\mathbf{x}(n)$ from $\mathbf{x}(n-1)$ by the following simple 2-step process:

1. Replace the first element of $\mathbf{x}(n-1)$ with the last element of $\mathbf{x}(n)$, e.g., start with (4) and replace x_0 with x_{16}.
2. Perform circular shift on the replaced sequence by one sample (to the left).

These two steps lead to simple effects in the DFT domain.

2.1 Effect of Step #1:

Observe the contribution of the first element (i.e., $x(0)$) in the DFT:

$$X_k = \sum_{n=0}^{N-1} x(n) W_N^{nk} = x(0) + x(1)W_N^k + \ldots + x(N-1)W_N^{(N-1)k} \tag{8}$$

which shows that the first element $x(0)$ appears unmodified in the DFT formula. Hence, the effect of step #1 on the DFT is simple. For example, perform step 1 on (4) to get an intermediate sequence

$$\widehat{\mathbf{x}(15)} = \{x(16), x(1), \ldots, x(14), x(15)\}. \tag{9}$$

Denoting by $\mathbf{X}_k(15)$ and $\widehat{\mathbf{X}_k(15)}$ the DFTs of respectively $\mathbf{x}(15)$ and $\widehat{\mathbf{x}(15)}$, we get from (8)

$$\widehat{X_k(15)} = X_k(15) - x(0) + x(16) \tag{10}$$

2.2 Effect of Step #2:

Denote by $\mathbf{X}_k(16)$ the DFT of $\mathbf{x}(16)$, then we simply apply property (7) to (10) to get

$$X_k(16) = \widehat{X_k(15)} e^{j2\pi k/N} \tag{11}$$

2.3 The SDFT Algorithm

Hence, combining the effects of both steps, we can relate the DFTs of two successive windowed sequences, each of length N, $\mathbf{x}(n-1)$ and $\mathbf{x}(n)$ as:

$$\boxed{X_k(n) = \left[X_k(n-1) - x(n-N) + x(n)\right] e^{j2\pi k/N}} \tag{12}$$

3 Analysis

In terms of efficiency, the SDFT is impressive because, regardless of N, it requires a constant number of operations to compute a successive DFT output: two real adds and one complex multiply (from (12)). However, this computation assumes that, for the current time instant, the DFT of the previous time instant is available. A simple solution for algorithm initialization is to obtain the DFT of the first time instant simply using the direct DFT (8), or apply the FFT (when more than one X_k's are required).

Alternatively, we consider an IIR implementation, shown in Fig. 1. This structure is simply a comb and resonator cascade filter. The output is relabeled as $S_k(n)$ to emphasize the transient nature of the filter, i.e., due to the comb filter delay, the output $S_k(n)$ is not equal to $X_k(n)$ for $n < N$. This means that the output is not valid until N input samples have been processed (as is expected, since an N-DFT requires by definition N samples).

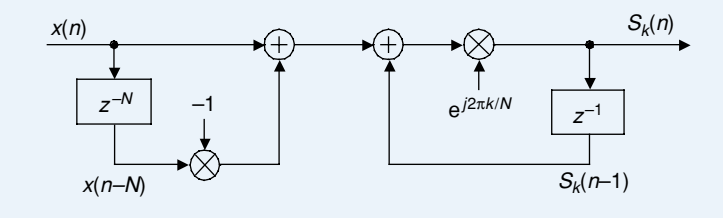

Figure 1: An IIR Implementation of the SDFT: Comb and Resonator Cascade.

Also note that if several DFT elements are required, then a bank of resonators, all driven by a single comb filter can be used. For example, if one is interested in the X_1 and X_4 values of a 16-DFT, then two resonators at frequencies $2\pi/16$ and $2\pi 4/16$ are driven by a single comb filter with a delay $N = 16$.

4 Examples

Example 4.1 (*Circular Shift*) The sequence

$$\{x(n)\}_{n=0}^{4} = [1, 2, 3, 4, 5] \tag{13}$$

has 5-DFT sequence

$$X_k = [15, -2.5 + 3.4410i, -2.5 + 0.8123i, -2.5 - 0.8123i, -2.5 - 3.4410i]. \tag{14}$$

Hence, the circularly shifted sequence $y[n] = (x(n-2))_5$

$$y[n] = (x(n-2))_5 = [4, 5, 1, 2, 3] \tag{15}$$

has 5-DFT sequence

$$Y_k = X_k W_N^{2k} = [15, 4.0451 - 1.3143i, -1.5451 - 2.1266i, -1.5451 + 2.1266i, 4.0451 + 1.3143i] \tag{16}$$

See the following GNU Octave snippet. Obviously with very large N, using the DFT property is more efficient than manually invoking the FFT.

GNU Octave/MATLAB code snippet:

```
x = [1,2,3,4,5], N = length(x); X = fft(x,N)
% do circular shift
m = 2; n = [0:N-1]; n = mod(n-m,N);y = x(n+1)
% manual FFT
Y = fft(y)
% circular shift property
k = [0:N-1]; X.*exp(-j*2*pi*m*k/N)
```

Example 4.2 (*Sliding DFT*) Let $x[n]$ in the previous example be a windowed sequence defined at some instant in time. And suppose the following windowed sequence is the sequence $z[n] = [2, 3, 4, 5, 6]$, which has 5-DFT sequence Z_k. Then the elements of Z_k can be computed using the SDFT as:

$$Z_k = (X_k - 1 + 6)e^{j2\pi k/5} = [20 - 2.5 + 3.4410i, -2.5 + 0.8123i, -2.5 - 0.8123i, -2.5 - 3.4410i] \quad (17)$$

For example,

$$Z_4 = (-2.5 - 3.4410i - 1 + 6)e^{j2\pi 4/5} = -2.5 - 3.4410i \quad (18)$$

GNU Octave/MATLAB code snippet:

```
z = [2,3,4,5,6]
% manual FFT
Z = fft(z)
% sliding DFT for whole sequence, i.e., like filter bank
k = [0:N-1]; (X(k+1)-x(1)+z(end)).*exp(j*2*pi*k/N)
% sliding DFT for single value
k = 4; (X(k+1)-x(1)+z(end)).*exp(j*2*pi*k/N)
```

You can modify the above snippet for very large N, and run over many successive sliding windows, to measure the speed difference between FFT and sliding DFT.

References

[1] E. Jacobsen and R. Lyons. "The Sliding DFT", *IEEE Signal Processing Magazine*, Mar. 2003, pp. 74-80.

[2] E. Jacobsen and R. Lyons. "An update to the Sliding DFT", *IEEE Signal Processing Magazine*, Jan. 2004, pp.110-111.