Real-Time Audio Analysis/Visualization with the Sliding Discrete Fourier Transform (SDFT) algorithm - 2014.09.14

Shikhir Arora

shikhir@sarajar.com

Synopsis: [DRAFT - IN PROCESS] The goal of this paper is to present the SDFT (Sliding Discrete Fourier Transform) in a concise but mathematical way as an alternative algorithm for applications in the real-time audio processing domain. Specifically, the goal is to present the intuition behind the very widely used Fast Fourier Transform (FFT) and show the reasoning as well as theory behind the SDFT. The application of the SDFT is then discussed with the goal of application towards real-time analysis of audio for the purpose of visualization that is automated and accessible to users using capable hardware in 2014. Our application of real-time audio processing has specific requirements with regards to the transform data needed for efficient end result (that is accurate visualization through display patterns or through third party hardware such as lightbulbs). The efficiency of the SDFT towards this goal is discussed and compared with traditional algorithms. This paper aims to match the SDFT algorithm to the stated goal, showing that the gains from SDFT are directly beneficial for this type of analysis. The negatives of SDFT are also discussed and analyzed to show that, for the real-time purpose intended for this project, there are minimal *statistically significant* downsides as the application mainly utilizes the dynamic design of the SDFT while omitting needs for the invertibility and large data output of traditional FFT's. The importance of characteristics in real-time processes such as latency are discussed for the efficient application and design with the algorithm. This document pulls from the Introduction to SDFT published by the University of Toronto's Communications Department (linked at the end of this paper) and the 2003 (updated 2011) publication by E. Jacobsen and R. Lyons: "The Sliding DFT" (also linked at the end of this paper). The construction of the paper is aimed to be mathematically sound but also intuitive and accessible, focusing on the theory of the SDFT and resultant description in higher-level of potential real-time audio visualization projects. [DRAFT - IN PROCESS]

1 Intuition

Our goal is to perform a Discrete Fourier Transform, specifically a Sliding DFT to model a time-frequency relationship. Furthermore, this paper discusses an intuitive approach towards using SDFT in real-time audio processing. The FFT (Fast Fourier Transform) is one of the most popular algorithms utilized by many for performing a computationally quick and accurate output of sample data. It is very important within specific fields (such as signal processing) to generate approximations. I focus on real-time analysis through an external audio source which could be used in further applications (such as advanced, microphone enabled visualization), Traditional methods for such a task involve FFT and IFFT (inverse FFT) functions to construct signals and perform a relatively quick, accepted form of the spectrum analysis.

A quick overview of FFT is needed to understand the incentive for an alternate method of signal processing. The FFT algorithm takes advantage of patterns within exponential functions

such as our Discrete Fourier Transform (DFT). While not absolutely required, an efficient and fast FFT algorithm such as the widely used Cooley-Turkey FFT rely on bases of 2 for their sample. In a traditional FFT application, one chooses a quantity of samples to analyze. For example, 1,024 (a multiple of 2) is a popular sample size when using a traditional FFT. Why the number 2? Remember that the FFT is just an algorithm to compute the DFT. It works by taking a sample of data and utilize repetitive symmetry within the equation. This recursion eventually reduces the large sample [which would demand **many** (N^2) resources with a standard DFT applied to each value] to a computationally negligible amount. A standard DFT has a computational efficiency of $\mathcal{O}(N^2)$ while the FFT reduces this to $\mathcal{O}(N * log(N))$. Let us briefly take a look at the DFT from definition:

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-i2\pi kn/N} \qquad \text{(1 - Discrete Fourier Transform/DFT)}$$

There are several ways one could interpret this (single sided) DFT. To get an intuitive sense of how the FFT was developed we first utilize Euler's Identity where $exp[i * n * 2\pi]$ equals 1 for all n. Therefore if we compute X_{N+k} (or X_{N*i+k}) we find that it is equal to X_k. This is crucial: we can now split the summation into two parts, even and odd, and compute the DFT from the new formula:

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} * e^{-i2\pi k(2m)/N} + \sum_{m=0}^{N/2-1} x_{2m+1} * e^{-i2\pi k(2m+1)/N} \qquad \text{(2 - FFT)}$$

We must also note that the range of k is $0 \leq k < N$ and the range of n is $0 \leq n < N/2$. Taking this symmetry into consideration we only have to perform *half* of the resulting computations. And this is recursive: we can keep dividing by powers of 2 (in the case of the discussed Cooley-Turkey FFT, which is the most popular implementation) until we get to a sample size that is low enough to be computationally negligible. The FFT algorithm now has a computational efficiency of $\mathcal{O}(N * log(N))$ compared to the $\mathcal{O}(N^2)$ standard DFT.

The FFT is very efficient and enjoys widespread use in digital signal processing. For real-time applications such as audio analyzation from a microphone, the use of a Sliding Discrete Fourier Transform is more efficient. The SDFT is based on a sliding window where the prior sample point is used to construct the next DFT and the oldest value truncated in an ordered chain of set value. In proper implementation the SDFT achieves a computational speed of $\mathcal{O}(N)$ - faster than that of a FFT or standard DFT. In addition, it makes intuitive sense to use a SDFT in applications where small samples are ideal and DFT's are calculated more often.

2 The Sliding DFT

2.1 Basic Idea

Remember that a FFT works in powers of 2 for its efficient application, so a sample of 1,024 would be reduced by recursion to 512, 256, 128, etc. Each recursion is faster and the sample is reduced to 1. This results in the $\mathcal{O}(N * log(N))$ computational speed which is desirable. Another note is to remember that an FFT is simply a method for calculating a DFT, and the results by theory are exactly the same numerically as that of a raw DFT for the sample size. This makes sense as we are just using recursion and the power of symmetry (hence the number 2) to vastly improve our DFT runtime. Given X sample size, FFT will output X data (1:1).

A Sliding DFT is based on the intuition that two successive time windows contain almost all the same information. It is easy to see this by taking two windowed DFT's. Let them be at time instants $n = 5$ and $n = 6$ (any successive time instants work for this; $n, n-1$). In our example with 5 and 6, our fixed window size $N = 6$ would yield the following DFT's:

$$DFT_5 = \{x(0), x(1), x(2), x(3), x(4), x(5)\} \tag{3}$$

and this is used to compute our DFT at $n = 6$ with our window fixed at $N = 6$:

$$DFT_6 = \{x(1), x(2), x(3), x(4), x(5), x(6)\} \tag{4}$$

Comparing the two DFT's we note that they are **very** similar - DFT_{x-1} compared to DFT_x is the basis and direct intuition for a Sliding DFT. As one may already see, real-time audio processing doesn't necessarily need all the outputs in our sample N. Applications that are further one-way (where construction of the original signal is not necessarily needed) such as visualization from an external audio source make the SDFT intuitively but also mathematically faster. In addition, we have the benefit of DFT samples that are likely smaller, not limited by the power of 2 so they can match an application's best requirement for sample size more natively and the data is constantly changing (as is our input with a real-time project described). These benefits will be discussed shortly in a bit more detail but first we should define/present the SDFT from the intuition in equations [3] and [4] as well as general properties of a DFT.

2.2 SDFT

With the above intuition in place we can now define our Sliding DFT algorithm and discuss its application towards real-time audio processing. One final concept is needed to complete our model. It turns out that the DFT sequence follows a *circular shift* pattern. If we take any windowed sequence: N-DFT (where N=fixed window) of $x[n]$ is X_k we can multiply by a linear phase (remember Euler's Identity) $e^{2\pi i/N*nm}$ for some integer m such that $X_k -> X_{k-m}$ modulo N. For our circular shift by one to the left (to construct our Sliding DFT) this is shown by:

$$X_k -> X_k * e^{i2\pi k/N} \tag{5 - Circular Shift}$$

Therefore, to construct our Sliding DFT, we replace the first element of $X(n-1)$ with the last element of $X(n)$ and compute the above one-left circular shift. We can therefore use this theory for any N to get $X(n)$ from $X(n-1)$ (the prior element). This is our SDFT as defined - we use the prior element to compute our DFT for the next sample. Furthermore applying this to our definition of the DFT we can easily see that the first element $x(0)$ is unmodified and therefore we can construct our SDFT formula.

We can now present our SDFT concisely. Using the intuition and the circular shift above, the Sliding DFT (SDFT) of two fixed windowed sequences with successive time instants ($n, n-1$, fixed window N):

$$\boxed{X_k(n) = [X_k(n-1) - x(n-N) + x(n)] * e^{i2\pi k/N}} \tag{6 - SDFT}$$

We will end this section by quickly noting the significance of our SDFT algorithm construction: for *any* N the SDFT model requires an addition, subtraction and complex circular

shift (a singular complex multiplication). This algorithm results in a $\mathcal{O}(N)$ order for each successive sample (compared to a $\mathcal{O}(N * log(N))$ for a FFT). The SDFT assumes that the DFT of the prior time instant $(n-1)$ is available. Depending on the application a single DFT can be taken for our first point, an FFT can be performed or, in the case of real-time audio processing, we can assume that the signal is silent until our first point (as it is) and therefore move sample-by-sample. One can also implement a Infinite impulse response filter where our SDFT is not considered valid until N samples are performed (invalid where $n < N$; per the definition of a windowed DFT with window N). Careful implementation of the SDFT algorithm yields beneficial results in our application for real-time audio as our transform is relatively simple and we are not limited by any powers of 2 for the window size as we are with an efficient FFT.

3 Discussion/Application

With our now presented SDFT algorithm, we can apply it towards our goal of real-time audio processing. The SDFT has various potential applications. Bradford, Dobson and Ffitch's paper published by the University of Bath, England's Computer Science Department discusses an application within the CSound language. [citation] Our goal is specifically real-time audio analysis. In late 2014 computing power continues to grow and the technologies adapted from real-time audio begin to automate tasks such as music visualization. With something like microphone input in real time the SDFT algorithm makes intuitive and mathematical sense. When a $X_{input} -> X_{output}$ (one-to-one) sample result is desired (that is, *all* elements are important or needed in the DFT output from an input) then the FFT is a very efficient algorithm. In real time, however, we aren't necessarily concerned about *every* sample point as the data is constantly changing. In addition, we're concerned with stepwise changes in frequency versus time. A general tool, the spectrogram is used in audio analysis that graphs, for a certain window, frequency with respect to time. In the example of microphone input, a decibel range is chosen by the application (log scale) which represents the minimum dBSL (decibel to unit scale for log) or sensitivity for our input source. The spectrogram then graphs, in real time, the frequency makeup of the signal and represents it on a graph of frequency versus time where more prevalent and harder hitting frequencies are shown as brighter colors. Performed on even modern tablets today such as an Apple iPad Air (64-bit), we are able to get a fast, very accurate frequency makeup of the track through an input source (in our case a standard USB audio microphone). With the mentioned experiment a third party application was used that utilizes traditional FFT's, but the use of a SDFT would be beneficial in the case that this data be fed into further use. The spectrogram alone, for example, has a variety of uses and applications. We're interested in taking data with this principle and using the SDFT to calculate it. Specifically, the potential for visualization on the system as well as to third party platforms is significant.

Taking an application designed with the SDFT algorithm at its core, we can take real time audio from a reliable input source on capable hardware and feed this into a variety of music visualization algorithms, both utilizing current and new designs. The data from the SDFT is fast, robust and changing. Furthermore, we're utilizing the circular shift properties of DFT's to constantly change our DFT within a fixed N window and are not limited by any sample-powers of 2, thereby allowing efficient design of a window sample proper for the application. This can be modified and changed as testing is performed to develop the ideal window for sampling. The old data is not needed as we are only interested in the new (successive) data, quickly interpreting the fed DFT data into visualization algorithms and moving to our next window. This high level description flows **very** well with our SDFT definition and construction. We're only interested in the data a SDFT can provide, and a typical FFT provides extraneous

data for our application. FFT's provide further application with an easy inverse FFT (IFFT) relation that can be constructed very easily mathematically from our FFT definition. This is used in signal processing where reconstruction is needed, such as taking in an input, graphing or collecting FFT outputs and taking those outputs to reconstruct the original signal. Vocoder's, for example, use this deeply to construct algorithms that reconstruct audio signals for applications in many fields. Image processing uses the FFT and IFFT at a base level to construct and reconstruct images with algorithms and compression techniques for various projects. In our case, however, we are not interested in construction of a signal from our transform. As one might imagine, the SDFT algorithm is not easily invertible as the FFT is and cannot be easily used to reconstruct signals from the output. There are methods for doing so with oscillator banks or subsetting (these are discussed in the Bradford, Dobson and Ffitch paper), however our application doesn't require this. It is important to point out that the $\mathcal{O}(N)$ efficiency of the SDFT is beneficial in our case as we're interested in fast changing data. In automated music visualization latency is **key** and it is important to minimize delays. The input data is quickly fed to visualization algorithms that, depending on the front end, display various patterns or send their results as a request to a third party, external source, such as a lightbulb. In the latter scenario, it is very important to minimize latency from our input to our output - that is we are trying to take in real time audio, quickly analyze the transform data, and feed it to an external source (like a lightbulb). The SDFT therefore can quickly feed in time/frequency data and our algorithm(s) for visualization can analyze and map these to the third party output. In the case of one such application, Philips HUE lighting, data from a (traditionally) FFT-output is fed through an algorithm which maps the LED colors over a frequency spectrum. The frequency spectrum is fed into this and assigned RGB-color values that are transmitted to lightbulbs. This is to visualize in real-time or by using waveform's in real time. The benefit of a real-time microphone input signal is that we are able to utilize our SDFT more efficiently. The SDFT works better with a smaller range of window, that is, we're better utilizing certain parts of the frequency spectrum to get a faster, and computationally accurate data field. This turns out to work perfectly as real-time microphone input visualization generally looks at the low frequency changes - these are the ones that are changing with more sensitivity and thus better versatility and likely end result for visualization that is accurate to the time domain of the signal/audio source. In the future, running a distribution function for the ideal sensitivity for both (a) known data that is human hearing is not even and humans hear part of the frequency spectrum "better" as well as (b) experimental results, both hardware and user-input based, where specific genres of music come into play (in certain genres the lower frequencies model the track where, in other genres, mid or high range frequencies may pose a more accurate visualization. The combination of these human inputs, known scientific data with the human ear (for the average adult) as well as experimental/genre based data can lead to very efficient visual algorithms. The scalability and relatively easily computed transformations of our SDFT fed with these algorithms, coupled with increasing and cheaper unit computational power for real-time CPU/GPU based-algorithms and even ASIC (application specific hardware) accessibility allows this model to flourish.