Digital Signal Processing/Discrete Fourier Transform

As the name implies, the **Discrete Fourier Transform** (DFT) is purely discrete: discrete-time data sets are converted into a discrete-frequency representation. This is in contrast to the DTFT that uses discrete time, but converts to continuous frequency. Since the resulting frequency information is discrete in nature, it is very common for computers to use DFT calculations when frequency information is needed.

Using a series of mathematical tricks and generalizations, there is an algorithm for computing the DFT that is very fast on modern computers. This algorithm is known as the **Fast Fourier Transform** (FFT), and produces the same results as the normal DFT, in a fraction of the computational time as ordinary DFT calculations. The FFT will be discussed in detail in later chapters.

Contents

DFT

The Discrete Fourier Transform is a numerical variant of the Fourier Transform. Specifically, given a vector of *n* input amplitudes such as $\{f_0, f_1, f_2, \ldots, f_{n-2}, f_{n-1}\}$, the Discrete Fourier Transform yields a set of *n* frequency magnitudes.

The DFT is defined as such:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}$$

here, k is used to denote the frequency domain ordinal, and n is used to represent the time-domain ordinal. The big "N" is the length of the sequence to be transformed.

The **Inverse DFT** (IDFT) is given by the following equation:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn}$$

Where W_N is defined as:

$$W_N = e^{-j2\pi/N}$$

Again, "N" is the length of the transformed sequence.

Matrix Calculations

The DFT can be calculated much more easily using a matrix equation:

$$\mathbf{X} = \mathbf{D}_N \mathbf{x}$$

The little "x" is the time domain sequence arranged as a Nx1 vertical vector. The big "X" is the resulting frequency information, that will be arranged as a vertical vector (as per the rules of matrix multiplication. The "D_N" term is a matrix defined as such:

$$\mathbf{D}_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{(N-1)} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}$$

In a similar way, the IDFT can be calculated using matrices as follows:

$$\mathbf{x} = \mathbf{D}_N^{-1}\mathbf{X}$$

And we will define $\mathbf{D}_N^{-1}$ as the following:

$$\mathbf{D}_N^{-1} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^{-1} & W_N^{-2} & \cdots & W_N^{-(N-1)} \\ 1 & W_N^{-2} & W_N^{-4} & \cdots & W_N^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{-N-1} & W_N^{-2(N-1)} & \cdots & W_N^{-(N-1)(N-1)} \end{bmatrix}$$

Visualization of the Discrete Fourier Transform

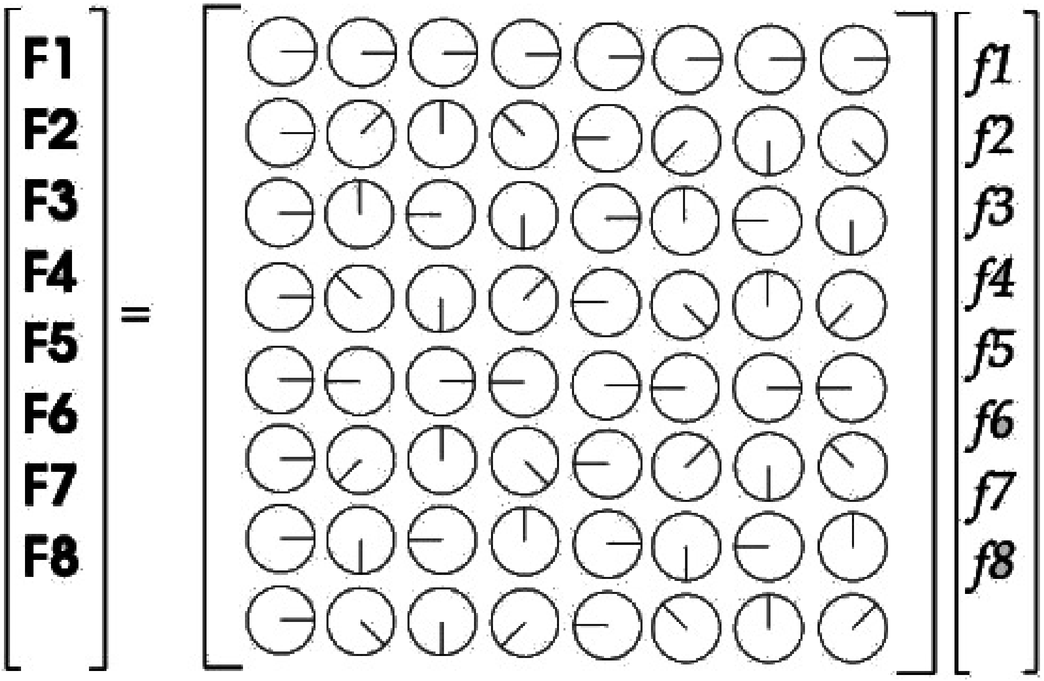
It may be easily seen that the term

$$e^{-\pi t}$$

represents a unit vector in the complex plane, for any value of j and k. The angle of the vector is initially 0 radians (along the real axis) for $j = 0$ or $k = 0$. As j and k increase, the angle is increased in units of 1/nth of a circle. The total angle therefore becomes $2\pi * jk/n$ radians.

To understand the meaning of the Discrete Fourier Transform, it becomes effective to write the transform in matrix form, depicting the complex terms pictorally.

For example, with $n = 8$, the fourier transform can be written:



The top row (or the left-hand column) changes not at all; therefore, F0 is the resultant when the amplitudes augment each other. The second row changes by 1/nth of a circle, going from left to right. The third row changes by 2/nths of a circle; the fourth row changes by 3/nths of a circle; the fifth row, here, changes by 4/nths = 4/8 = 1/2 a circle. Therefore, F4 is the resultant when the even terms are set at odds with the odd terms.

We therefore see from the square matrix that the DFT is the resultant of all 2-D vectoral combinations of the input amplitudes.

Relation to DTFT

The DFT and the DTFT are related to each other in a very simple manner. If we take the DTFT of a given time sequence, x[n], we will get a continuous-frequency function called $X(e^{j\omega})$. If we sample $X(e^{j\omega})$ at N equally-spaced locations, the result will be the DFT, X[k]

Circular Time Shifting

Circular Time Shifting is very similar to regular, linear time shifting, except that as the items are shifted past a certain point, they are looped around to the other end of the sequence. This subject may seem like a bit of a tangent, but the importance of this topic will become apparent when we discuss the Circular Convolution operation in the next chapter.

Let's say we have a given sequence, x[n], as such:

```
x[n] = [1& 2 3 4]
```

As we know, we can time-shift this sequence to the left or the right by subtracting or adding numbers to the argument:

```
x[n+1] = [1 2& 3 4 0]
```

```
x[n-1] = [0& 1 2 3 4]
```

Here, we've padded both sequences with zeros to make it obvious where the next item is coming in from. Now, let's say that when we shift a set, instead of padding with zeros, we loop the first number around to fill the hole. We will use the notation x[<n+A>] to denote an A-point circular shift:

```
x[<n+1>] = [2& 3 4 1]
```

Notice how the length of the sequence started as 4, and ended as 4, and we didn't need to pad with zeros. Also notice that the 1 "fell off" the left side of the set, and "reappeared" on the right side. It's as if the set is a circle, and objects that rotate off one direction will come back from the other direction. Likewise, let us rotate in the other direction:

```
x[<n-1>] = [4& 1 2 3]
```

If we rotate around by N points, where N is the length of the set, we get an important identity:

```
x[<n+N>] = x[<n-N>] = x[n]
```

Time Inversion

Now, let's look at a more complicated case: mixing circular shifting with time inversion. Let's define our example set x[n]:

```
x[n] = [1& 2 3 4]
```

we can define our time-inverted set as such:

```
x[n] = [4 3 2 1&]
```

notice that the "4" went from n=3 to n=-3, but the "1" stayed at the zero point. Now, let's try to combine this with our circular shift operation, to produce a result x[<-n>]. We know that the "1" (in the zero position) has not moved in either direction, nor have we shifted to the left or right, because we aren't adding or subtracting anything to our argument (n). So we can put 1 back in the same place as it was. All the rest of the numbers (2 3 and 4) all shifted to the right of the sequence, and therefore we will loop them around to the other side:

```
x[<-n>] = [1& 4 3 2]
```

Here is a general rule for circular time-inversion:

$$x[< -n >] = \begin{cases} x[0], & \text{if } n = 0 \\ x[< N - n >], & \text{if } n \neq 0 \end{cases}$$

Circular Convolution

Circular Convolution is an important operation to learn, because it plays an important role in using the DFT.

Let's say we have 2 discrete sequences both of length N, x[n] and h[n]. We can define a circular convolution operation as such:

$$y[n] = \sum_{m=0}^{N-1} x[m]h[< n - m >_N]$$

notice how we are using a circular time-shifting operation, instead of the linear time-shift used in regular, linear convolution.

Circular Convolution Theorem

The DFT has certain properties that make it incompatible with the regular convolution theorem. However, the DFT does have a similar relation with the Circular Convolution operation that is very helpful. We call this relation the **Circular Convolution Theorem**, and we state it as such:

Circular Convolution Theorem

> Multiplication in the discrete-time domain becomes circular convolution in the discrete-frequency domain. Circular convolution in the discrete-time domain becomes multiplication in the discrete-frequency domain.

Relation to Linear Convolution

Circular Convolution is related to linear convolution, and we can use the circular convolution operation to compute the linear convolution result. Let's say we have 2 sets, x[n] and h[n], both of length N:

1. pad both sets with N-1 zeros on the right side.
2. perform the circular convolution on the new sets of length N+(N-1)
3. The result is the linear convolution result of the original two sets.

Fast Fourier Transform (FFT)

The Fast Fourier Transform refers to algorithms that compute the DFT in a numerically efficient manner.Algorithms like:-decimation in time and decimation in frequency. This is a algorithm for computing the DFT that is very fast on modern computers.

Retrieved from "https://en.wikibooks.org/w/index.php?title=Digital_Signal_Processing/Discrete_Fourier_Transform&oldid=3023321"