NC STATE UNIVERSITY

Charles David Richardson

FFT Mini Project

ECE 513: Digital Signal Processing

10/26/2022

1. Introduction

The Discrete Fourier Transform(DFT) is one of the most important principles in digital signal processing. The process of Discrete Fourier Transform(DFT) converts a signal from its discrete time domain to a signal in the frequency domain. There are many methods to employ the DFT such as the Fast Fourier Transform(FFT) algorithm. The FFT is an algorithm that computes the Discrete Fourier Transform of a sequence by inherently dividing the DFT into even and odd samples. Since, computation of the DFT may be slow and require more time to execute, methods been established to speed up this computation.

2. Methods and Algorithms

As mentioned previously, the FFT is used to speed up DFT calculations, in which even and odd samples are taken from a specific discrete-time sequence x(n) and computed recursively, to construct the resulting signal in the frequency domain. This can be illustrated by the butterfly algorithm shown in Figure 1.

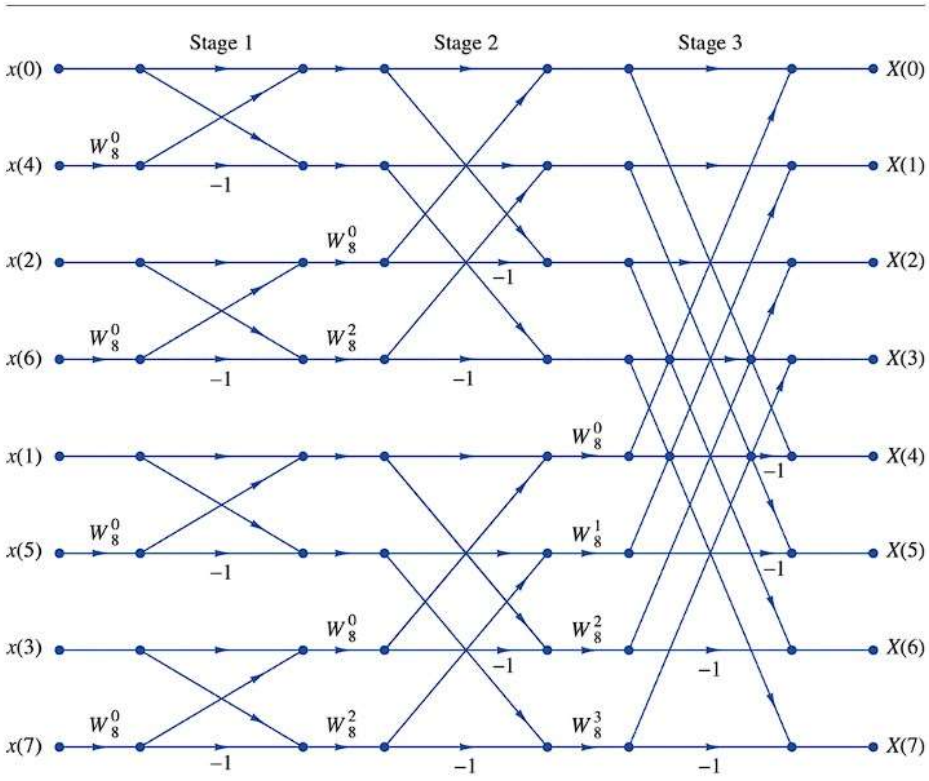

Figure 1: Eight Point Decimation In Time FFT(Butterfly) Algorithm

Each butterfly computation can be represented in the format shown in figure 2.

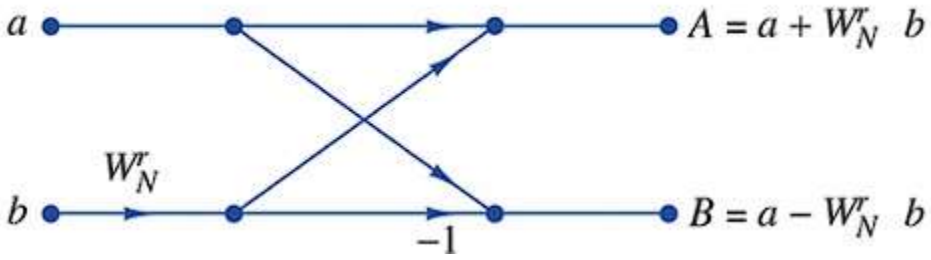

Figure 2: 2-Point DFT Butterfly Algorithm

3. MATLAB Design

MATLAB Code was designed for the DFT Algorithm as well as the FFT(Decimation in Time/Butterfly) Algorithm. The DFT Code is shown in Figure 3 and the FFT Code is shown in Figure 4.

3.1. Discrete Fourier Transform

The DFT Function was constructed with the following steps. I first calculated the length of the sequence using the length function. Then I truncated/zero padded the sequence depending on if N was greater than or equal to or less than L. I then initialized a temporary storage element for the calculation of the DFT and pre-allocated the output array X(k). Then I do the Discrete Fourier Transform Calculation for each k value and then add it to the array.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Filename    : my_DFT.m
%   Author      : Chuck Richardson
%   UnityID     : cdricha5
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[X_k] = myDFT(x,N)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculate the Length of the Series
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    L=length(x);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If N is less than L than truncate the sequence
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(N<L)
        x_n=x(1:N);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If N is greater than or equal to L than pad with zeros
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    elseif(N >= L)
        x_n=[x, zeros(1,N-L)];
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Intialize Temporary and Output Arrays
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    y=0;
    X_k=[];
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate DFT
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for k=0:1:N-1
        for n=0:1:N-1
            y=y+x_n(n+1)*exp(-i*2*pi*n*k/N);
        end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Add resulting X(k) and Reinitialize y to zero
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        X_k=[X_k y];
        y=0;
    end
end
```

Figure 3: DFT MATLAB Code

3.2. Fast Fourier Transform

Like in the DFT Code, I first calculated the length of the sequence using the length function. Then I truncated/zero padded the sequence depending on if N was greater than or equal to or less than L. Then I reverse the bit order of the sequence and calculate the number of stages of the butterfly. Then I precompute the twiddle factors to result in less computations for the loop. I then index the Twiddle Factor Array to get the necessary Twiddle Factors for each stage.

I then perform the Butterfly FFT Calculation by computing $F_1(k)$ and $F_2(k)$ and use the butterfly function to get my resulting $X(k)$ and $X(k + N/2)$ for each respective butterfly. This is then added to the resulting array X when it is finished computation for all the stages. I have three loops one loop to loop over stages, one loop to loop over the butterfly groups, and then one loop to loop over butterflies. This is shown in Figure 4.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Filename    : my_FFT_273.m
%   Author      : Chuck Richardson
%   UnityID     : cdricha5
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function X = myFFT_273(x,N)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculate the Length of the Series
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    L=length(x);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Truncate/Zero-Pad
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(N<L)
        xin=x(1:N);
    elseif(N>=L)
        xin=[x, zeros(1,N-L)];
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Compute Number of Stages
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    NUM_STAGES=log2(N);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Reverse Order for Butterfly
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    xin=bitrevorder(xin);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Pre-Compute the Twiddle Factors
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Twiddle_Factors = zeros(0,N/2-1); %Pre-Allocate Twiddle Factors
    for n = 0:(N/2-1)
        Twiddle_Factors(n+1)=cos(2*pi/N*n)-1i*sin(2*pi/N*n);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Initialize Butterfly Arrays/Output Array
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Input_Butterfly = zeros(1,2);
    Get_Butterfly   = zeros(1,2);
    X               = zeros(1,N);
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Going through N-Point DFTs
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for stages = 1:NUM_STAGES
        N_Stage = 2^stages;
        N_Stage_Div_2 = N_Stage/2;
        Twiddle_Factor = Twiddle_Factors(1:N/N_Stage:(N/2));
        N_minus_Nstage = N-N_Stage;
        for k = 0:N_Stage:N_minus_Nstage
            for n=0:N_Stage_Div_2 -1
                Input_Butterfly          = [xin(n+k+1) xin(n+k+N_Stage_Div_2+1)];
                Get_Butterfly            = butterfly(Input_Butterfly,
Twiddle_Factor(n+1));
                xin(n+k+1)                = Get_Butterfly(1);
                xin(n+k+N_Stage_Div_2+1)  = Get_Butterfly(2);
            end
        end
    end
    X=xin;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Butterfly Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function bout = butterfly(bin, twiddle)
    bout(1) =   bin(1) + (bin(2).*twiddle);
    bout(2) =   bin(1) - (bin(2).*twiddle);
end
```

Figure 4: FFT(Butterfly/Decimation in Time) MATLAB Code

4. Results

Verification and validation of results was conducted using the provided MATLAB verify.m script. Shown below in Table 1 is the margin of error for both the DFT and FFT respectively for the expected minus the actual.

Margin of Error			
N=8	Length = 256	DFT(Discrete Fourier Transform) Error	7.8208e-14
N=8	Length = 256	FFT(Fast Fourier Transform) Error	3.6575e-15
N=256	Length = 256	DFT(Discrete Fourier Transform) Error	1.9808e-8
N=256	Length = 256	FFT(Fast Fourier Transform) Error	6.6854e-11

Table 2: Margin of Error Calculations

Shown in Figure 5 is the magnitude output from our FFT for N=16 and L=256.

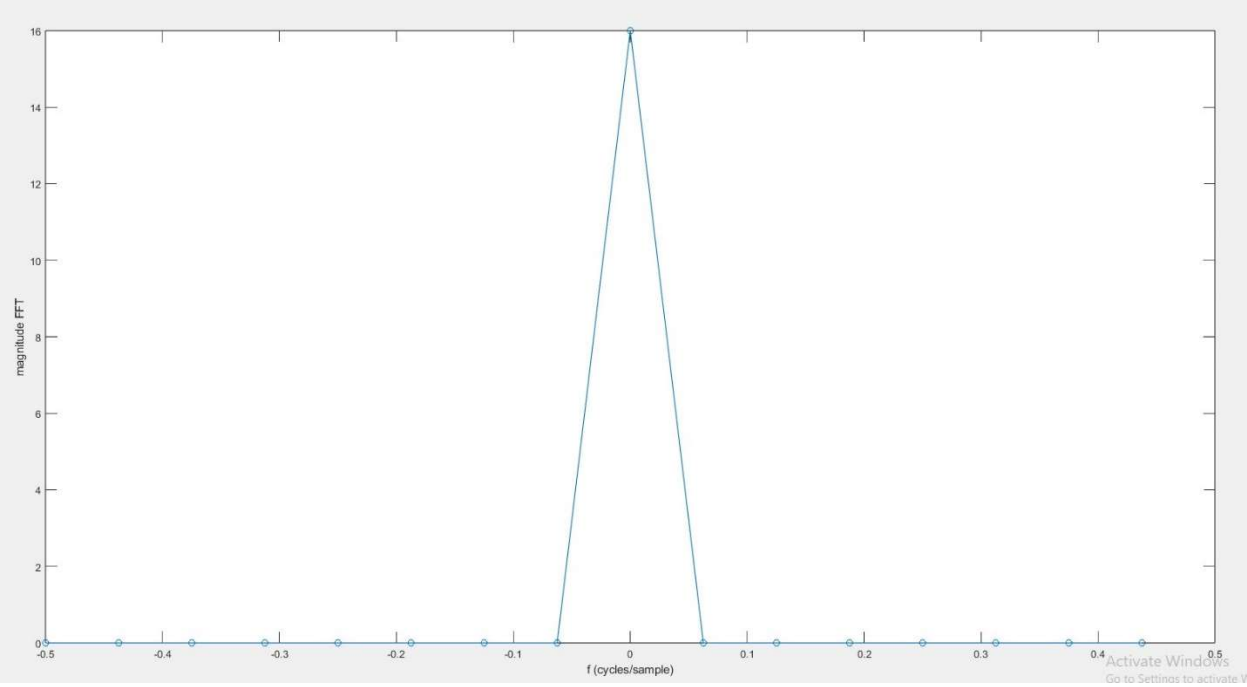

Figure 5: Magnitude Output FFT/Butterfly

Benchmarking was also conducted on my DFT and FFT code. Shown in Table 2 are the results of the benchmark. It is important to note that the FFT was much faster than that of the DFT. This can be attributed to the fact that the FFT calculates even and odd samples as opposed to calculating one large sample. Thusly, resulting in less memory used and faster execution time.

Function Name	**Calls**	**Total Time(s)**	**Self Time(s)**
myDFT	10	2.256	2.256
MyFFT_273	10	.280	0.093
MyFFT_273>butterfly	51200	0.163	0.163

Table 2: Execution Parameters

5. Conclusion

In conclusion, the Fast Fourier Transform remains superior to the Discrete Fourier Transform. It was very interesting to see the run time metrics of the Fast Fourier transform in comparison to Discrete Fourier Transform. By means of breaking up the samples into even and odd samples it is much more efficient to compute the Discrete Fourier Transform without a large overhead of memory and execution time. The results achieved during this project proved valid, with the actual matching that of the expected results. I look forward to using the FFT in industry as well as hopefully more of my educational endeavors.