Introduction

For this assignment, I implemented three functions in MATLAB®:

1. myDFT, a brute-force implementation of the DFT,
2. myFFT_0, a decimation-in-time implementation of the FFT, and
3. butterfly, a decimation-in-time butterfly called by myFFT_0.

A code listing for these functions is provided in the appendix.

Verification

I used the script below to verify the main functions. Results are provided in Table 1 and Figure 1.

Table 1. Verification results: sum of error magnitudes relative to MATLAB® fft.

Function	**N=8**	**N=256**
myDFT	1.6314e-13	5.6040e-08
myFFT_0	7.4733e-15	8.4525e-10

```
% script to verify myFFT by comparing to
MATLAB fft
x = [1:256];

N = 8
err1 = sum(abs( fft(x,N) - myDFT(x,N) ))
err2 = sum(abs( fft(x,N) - myFFT_0(x,N) ))

N = 256
err1 = sum(abs( fft(x,N) - myDFT(x,N) ))
err2 = sum(abs( fft(x,N) - myFFT_0(x,N) ))

N=16
x = ones(1,N);
X = myFFT_0(x,N);
X = fftshift(X);
xlist = [-N/2:N/2-1]/N;
plot(xlist,abs(X),'-o')
xlabel('f (cycles/sample)')
ylabel('magnitude FFT')
```

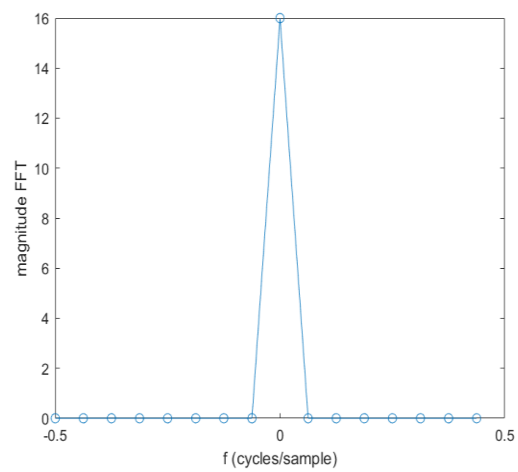

Figure 1. Output of FFT from verification.

Benchmarking

I used the following script to benchmark the time efficiency of the code.

```
% benchmarking script for myDFT, mlfft, and myFFT
x = [1:1024];        % test data
N=1024;              % length of DFT, FFT
ntimes = 10;       % number of times to compute DFT/FFT
```

```
profile on

% myDFT
for time = 1:ntimes
    X = myDFT(x,N);
end

% myFFT
for time = 1:ntimes
    X = myFFT_0(x,N);
end

% MATLAB FFT
for time = 1:ntimes
    X = mlfft(x,N);
end

profile off
profile viewer
 %-------------------------
function X = mlfft(x,N)
X = fft(x,N);
end
```

Results are provided in Table 2. Observe that my FFT performed much quicker than my DFT. Still, the MATLAB ® FFT was much faster than mine.

Table 2. Benchmark results: total time from MATLAB® profiler.

Function	Total Time (s)
myDFT	9.839
myFFT_0	0.494
MATLAB® FFT	0.001

Appendix

In this appendix a listing of the code I wrote is provided.

```
function X = myDFT(x,N)
%---------------------------------------------
% function: myDFT
% purpose: brute-force computation of DFT
% pgmmer:  GEB
% date:    fall 2022
% inputs:
%    x     x(n), row-vector of samples, implicitly n=0 to L-1
%    N     DFT order
% Note: if L > N, only first  N points of x used
% outputs:
%    X     X(k), row-vector of DFT values, implicitly k=0 to N-1
```

```
%--------------------------------------------
xlen = length(x);       % length of sample list
xzp = [x zeros(1,N-xlen)];  % zero pad x so length N
W = exp(-j*2*pi/N);     % precompute W_N
X = zeros(1,N);         % initialize DFT values to zero (for accum.)
for kindex=1:N          % loop over DFT indices starting at 1
    k = kindex-1;       % DFT indices starting at k=0
    for nindex = 1:N    % loop over sample indices starting at 1
        n = nindex-1;   % sample indices starting at n=0
        X(kindex) = X(kindex) + xzp(nindex)*W^(k*n);   %accum. DFT terms
    end
end

end
```

```
function X = myFFT_0(x,N)
%--------------------------------------------
% function: myFFT8
% purpose: decimation-in-time FFT for N=8
% pgmmer:  GEB
% date:    fall 2022
% inputs:
%    x     x(n), row-vector of samples, implicitly n=0 to L-1
% outputs:
%    X     X(k), row-vector of DFT values, implicitly k=0 to N-1
%--------------------------------------------
nstages = log2(N);          % number of stages
nbf = N/2;                  % butterflies per stage
xlen = length(x);           % length of sample list
if xlen > N                 % truncate if x longer than N
    xzp = x(1:N);
else
    xzp = [x zeros(1,N-xlen)];  % zero pad x so length N
end
W = exp(-j*2*pi/N);         % precompute W_N
temp = 1;                   % precompute powers of W_N needed
for m = 1:N/2
    Wlist(m) = temp;
    temp = W*temp;
end

xout = bitrevorder(xzp);       % order x in bit-reversed order, buffer

for stage = 1:nstages       % loop over stages
    xin = xout;             % use data from previous stage
    xout = [];              % empty buffer for stage output
    ngroup = 2^(nstages-stage);  % number of smaller FFTs
    for group = 1:ngroup    % loop over groups
        gindex1 = 1 + (group-1)*2^stage;  %first point into group
        npts = 2^stage;      % number of points into group
        gin = xin(gindex1:gindex1+npts-1);  %points into stage
        nbf = npts/2;          % bufferflies per group
```

```
        index1 = 1;             % index of first point into first butterfly
        jump = 2^(stage-1);     % index2 = index1 + jump
        index2 = 1 + jump;      % index of second pt into first buttfly
        tindex = 1;             % index for twiddle factor, first bf
        for bf = 1:nbf          % loop over butterflies in group
            %fprintf('stage=%d, group=%d, bf = %d\n',stage,group,bf); %debug
            twiddle = Wlist(tindex);     % twiddle factor, first butterfly
            bfin = [gin(index1),gin(index2)];   %input to bufferfly
            bfout = butterfly(bfin,twiddle);    %bufferfly
            gout(index1) = bfout(1);            % store results in gout
            gout(index2) = bfout(2);
            index1 = index1 + 1;    % update point indices for next bf
            index2 = index1 + jump;
            tindex = tindex + ngroup; % update twiddle factor index
        end % butterfly loop per group
        xout = [xout gout];   % concatenate group outputs
    end %group loop
end %stage loop
X = xout; % last xout is FFT result
end

function bfout = butterfly(bfin,twiddle)
%-----------------------------------------------
% function: butterfly
% purpose: decimate-in-time bufferfly
% pgmmer:  GEB
% date:    fall 2022
% inputs:
%    bfin  length-2 row vector of input values
%    twiddle  twiddle factor
% outputs:
%    bfout  lenght-2 row vector of output valudes
%-----------------------------------------------
prod = bfin(2)*twiddle;
bfout(1) = bfin(1) + prod;
bfout(2) = bfin(1) - prod;

end
```