CIE 442: Digital Signal Processing Wed., Dec. 02, 2020

Project Part 1 Report

By: Abdelrahman Elaraby (201700556)
Alaa Alajmy (201700095),
Moemen Gaafar (201700873)

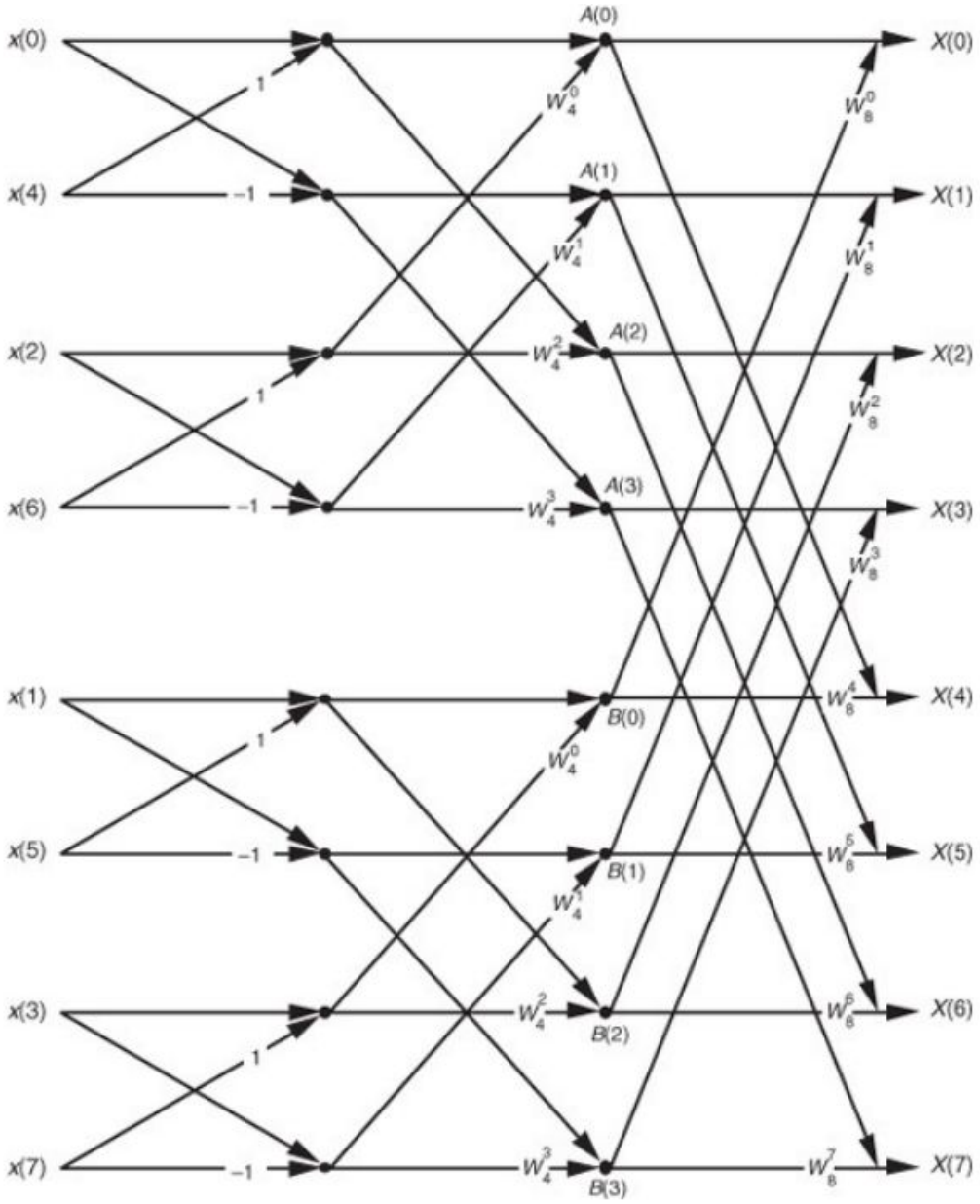
Called the most important mathematical algorithm yet discovered, the fast fourier transform (FFT) serves all fields from astronomy to chemistry to digital communication. Although the FFT was first discovered by Carl Gauss more than 200 years ago, scientists still race to develop a version of the same algorithm that could handle modern amounts of data, like those coming from the Laser Interferometer Gravitational-wave Observatory (LIGO), possibly using modern devices, like quantum computers. [1]

In this project, we will implement the FFT and the inverse FFT for 1-dimensional and 2-dimensional data.

Contents:

I. Introduction

The FFT is an algorithm that computes the discrete fourier transform or the inverse discrete fourier transform of a sequence at a complexity of $O(NlogN)$ rather than the traditional $O(N^2)$. In brief, the FFT works by decomposing the input sequence into smaller sequences of 2-points, computing the 2-point DFT of each 2-point sequence, then using the resultant to compute the 4-point spectrum of each adjacent 4 points, then computing the 8-point spectrum, and so on. A diagram of the 8-point FFt is shown below where the twiddle factor $W_N = e^{-2\pi j/N}$



II. Methods

The frame of work holds 6 main functions:

a. **zero_pad:**
   Input: a 1D or 2D input array and the dimension of the input.
   Output: the 1D or 2D input array zero-padded to the next 2^n and 2^m lengths.
   This function applies zero-padding on the input arrays before they are passed to the FFT or IFFT functions. To make the plotted time-domain representation plots more aesthetic, padding is applied on both ends across each dimension, rather than just on one end.

b. **myfft:**
   Input: a 1D time-domain array of length 2^n.
   Output: the frequency-domain spectrum of the input array with the same length.
   This function implements the FFT algorithm for a 1-dimensional array. Details of the implementation are found in the function's comments in the following pages.

c. **myifft:**
   Input: a 1D frequency-domain array of length 2^n.
   Output: the time-domain array of the input array with the same length.
   This function implements the IFFT algorithm for a 1-dimensional array.

d. **myfft_2D:**
   Input: a 2D spatial image with dimensions 2^n and 2^m.
   Output: the frequency-domain spectrum of the input array with the same size.
   This function implements the 2D-FFT algorithm by first implementing the 1D FFT of each column to produce an intermediate matrix, then computing the 1D FFT of each row of the intermediate matrix.

e. **myifft_2D:**
   Input: a 2D frequency-domain array with dimensions 2^n and 2^m
   Output: the spatial image of the input array with the same size.
   This function implements the 2D-IFFT algorithm by first implementing the 1D IFFT of each column to produce an intermediate matrix, then computing the 1D IFFT of each row of the intermediate matrix.

f. **fast_fourier:**
   Input: the number of dimensions used (1 or 2), the input 1D or 2D array, the transform to be applied (fft or ifft), and the sampling frequency.
   Output: the computed FFT/IFFT and a plot of both the time and frequency domains.
   This function takes in the input array, passes it to zero_pad to become of a size suitable for the FFT and IFFT functions, then passes it as required to myfft, myifft, myfft_2D, or

myifft_2D. At the end, fast_fourier plots the time and frequency domains on the input array.

III. References

[1] “Fast Fourier transform,” *Wikipedia*, 24-Nov-2020. [Online]. Available: https://en.wikipedia.org/wiki/Fast_Fourier_transform. [Accessed: 02-Dec-2020].

IV. Code

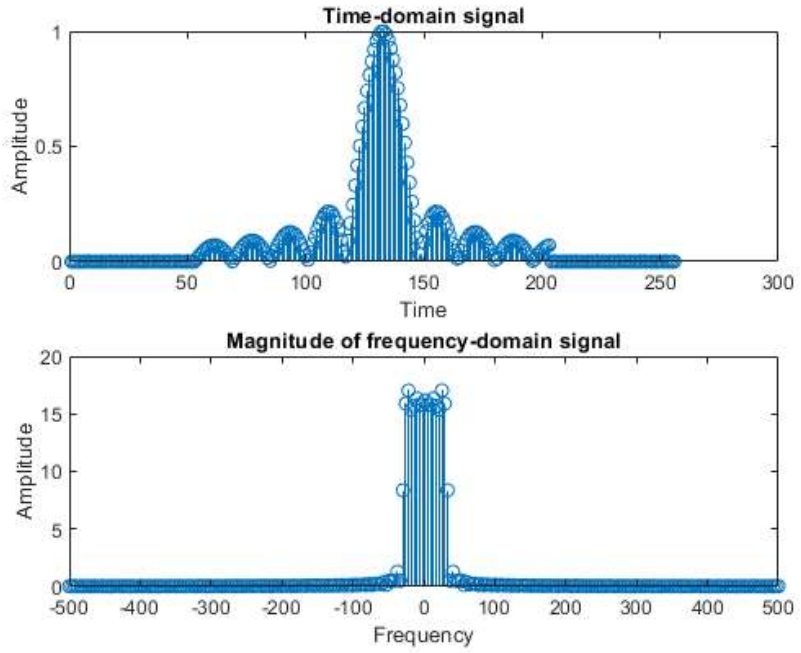
Our created functions along with a script that reads an image and a 1D signal then uses them to test the 1D and 2D FFT and IFFT algorithms are attached in the following pages.

Contents


```
clear; clc;
```

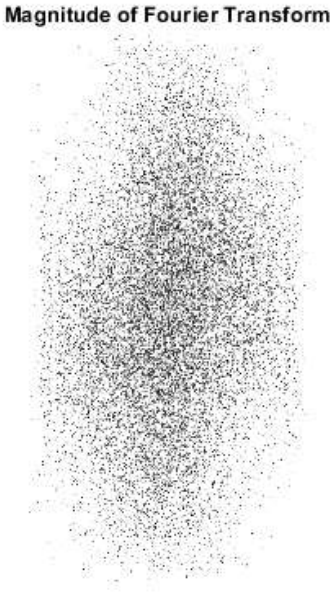
Testing 1D FFT

```
n = 1:150;
x = sinc(2*pi*n/100 - 5);
fast_fourier(1,x,"fft", 1000);
```



Testing 2D FFT

```
x = double(imread('pout.tif'));
fast_fourier(2,x,"fft");
```



1D FFT

```
function temp = myfft(x)
% This function performs 1D FFT.
% x must have 2^n elements where n is a positive integer.

N = length(x);
n = log2(N);
temp = zeros(1, N);

% Apply bit inversion to get correct order of FFT input
for i = 1: N
    j = dec2bin(i-1, n); % Convert decimal number to binary
    j = flip(j); % Invert bit order
    k = bin2dec(j); % Convert back to decimal
    temp(k+1) = x(i); % Re-insert original number into its new position
end

% Apply 2-point DFT on each adjacent pair of points
for i = 1:2:N-1
    x(i) = temp(i) + temp(i+1);
    x(i+1) = temp(i) - temp(i+1);
end

% Recursively compute the 4-point till N-point FFT
p = 4; % Point number to be used
while p <= N
    w = 0;
    % Apply algorithm for each p points of the array
    while w < N
        for k = 1:p/2
             temp(w + k)= x(w + k) + exp(-2i*pi*(k-1)/p)*x(w + k + p/2);
        end
        for k = p/2+1:p
             temp(w + k)= exp(-2i*pi*(k-1)/p)*x(w + k) + x(w + k - p/2);
        end
        w = w + p; % Increment w to compute next p points
    end
    p = p * 2; % Double p to compute next layer
    x = temp;
end

temp = x;
end
```

1D IFFT

```
function temp = myifft(x)
% This function performs 1D IFFT.
% x must have 2^n elements where n is a positive integer.

N = length(x);
n = log2(N);
temp = zeros(1, N);

% Apply bit inversion to get correct order of IFFT input
for i = 1: N
    j = dec2bin(i-1, n); % Convert decimal number to binary
    j = flip(j);  % Invert bit order
    k = bin2dec(j); % Convert back to decimal
    temp(k+1) = x(i)/N; % Re-insert original number into its new position
                        % and scale by 1/N
end

% Apply 2-point IDFT on each adjacent pair of points
for i = 1:2:N-1
    x(i) = temp(i) + temp(i+1);
    x(i+1) = temp(i) - temp(i+1);
end

% Recursively compute the 4-point till N-point iFFT
p = 4;
while p <= N
    w = 0;
    % Apply algorithm for each p points of the array
    while w < N
        for k = 1:p/2
             temp(w + k)= x(w + k) + exp(2i*pi*(k-1)/p)*x(w + k + p/2);
        end
        for k = p/2+1:p
             temp(w + k)= exp(2i*pi*(k-1)/p)*x(w + k) + x(w + k - p/2);
        end
        w = w + p; % Increment w to compute next p points
    end
    p = p * 2;  % Double p to compute next layer
    x = temp;
end

temp = x;
end
```

2D FFT

```
function X = myfft_2D(x)
% This function performs 2D FFT.
% x must have a size of 2^n by 2^m where n,m are positive integers

[n, m] = size(x);
Int = zeros(n, m);
X = zeros(n,m);

% Perform 1D FFT on each column of x then store the results in Int
for i = 1:m
    Int(:,i) = myfft(x(:,i)')';
end

% Perform 1D FFT on each row of Int then store the results in X
for i = 1:n
    X(i,:) = myfft(Int(i,:));
end

end
```

2D IFFT

```
function X = myifft_2D(x)
% This function performs 2D IFFT.
% x must have a size of 2^n by 2^m where n,m are positive integers.

[n, m] = size(x);
```

```
Int = zeros(n, m);
X = zeros(n,m);

% Perform 1D IFFT on each column of x then store the results in Int
for i = 1:m
    Int(:,i) = myifft(x(:,i)')';
end

% Perform 1D IFFT on each row of Int then store the results in X
for i = 1:n
    X(i,:) = myifft(Int(i,:));
end

end
```

Zero Padding

```
function y = zero_pad(x,dim)
% This function zero pads the input signal from all sides until its
%dimensions reach a power of 2.
% x is the input signal.
% dim is the the dimension of the matrix (1 or 2).

[n,m] = size(x);

if dim == 1
    % Calculate the number of remaining zeroes to reach a power of 2
    remaining_zeroes = 2^ceil(log2(m)) - m;

    if remaining_zeroes > 0
      % Pad the remaining zeroes from both sides
      if mod(remaining_zeroes,2) == 0
        y = [zeros(1,remaining_zeroes/2) x zeros(1,remaining_zeroes/2)];
      else
        y = [zeros(1,(remaining_zeroes+1)/2) x zeros(1,(remaining_zeroes-1)/2)];
      end
    else
        % In case no padding is required, return x as is
        y = x;
    end
else
    % Calculate the number of remaining zeroes for each dimension to
    %reach a power of 2
    remaining_zeroes_x = 2^ceil(log2(m)) - m;
    remaining_zeroes_y = 2^ceil(log2(n)) - n;

    if remaining_zeroes_x > 0
        % Pad the remaining zeroes from both sides along the horizontal
        % axis
        if remaining_zeroes_x > 0 && mod(remaining_zeroes_x,2) == 0
           y = [zeros(n,remaining_zeroes_x/2) x zeros(n,remaining_zeroes_x/2)];
        else
           y = [zeros(n,(remaining_zeroes_x+1)/2) x zeros(n,(remaining_zeroes_x-1)/2)];
        end
    else
        % In case no padding is required, return x as is
        y = x;
    end

    if remaining_zeroes_y > 0
       % Pad the remaining zeroes from both sides along the horizontal
       % axis
       if mod(remaining_zeroes_y,2) == 0
         y = [zeros(remaining_zeroes_y/2, m+remaining_zeroes_x) y zeros(m+remaining_zeroes_x,remaining_zeroes_y/2)];
       else
         y = [zeros((remaining_zeroes_y+1)/2, m+remaining_zeroes_x); y; zeros((remaining_zeroes_y-1)/2, m+remaining_zeroes_x)];
       end
    end

end
end
```

Layout

```
function X = fast_fourier(dim, x, type, fs)
% This is the layout function.
% dim is the dimension of the FFT and can take values of either 1 or 2.
% x is the input signal.
% type is a flag used to specify whether to perform FFT or IFFT and can take
%values of either 'fft' or 'ifft'.
% fs is the sampling frequency of the input signal.

if (dim ~= 1 && dim ~= 2)
    disp("Error: The dimension of the fft/ifft can either be 1 or 2");
    return;
end

if (type ~= "fft" && type ~= "ifft")
    disp("Error: The type parameter can either be 'fft' or 'ifft'");
    return;
end

x = zero_pad(x,dim);

m = size(x,2);

if dim == 1
    if type == "fft"
        X = myfft(x);
        time = x; fourier = X;
    else
        X = myifft(x);
        time = X; fourier = x;
    end
    subplot(2,1,1);
    stem(1:m, abs(time));
    xlabel("Time");
    ylabel("Amplitude");
    title("Time-domain signal");
    subplot(2,1,2);
    stem(linspace(-fs/2,fs/2,m), abs(fftshift(fourier)));
    xlabel("Frequency");
    ylabel("Amplitude");
    title("Magnitude of frequency-domain signal");
else
    if type == "fft"
        X = myfft_2D(x);
        image = x; fourier = X;
    else
        X = myifft_2D(x);
        image = X; fourier = x;
    end
        subplot(1,2,1);
        imshow(uint8(image));
        title("Image");
        subplot(1,2,2);
        imshow(uint8(abs(fourier)));
        title("Magnitude of Fourier Transform");
end
```

Published with MATLAB® R2019b