آزمایشگاه پردازش بی درنگ سیگنال های دیجیتال

گزاش کار آزمایش شماره 3 و 4

استاد شاه منصوری

فاطمه جلیلی

شماره دانشجویی : 810199398

تاریخ تحویل : 1402/10/8

آزمایش شماره 3 :

محاسبه ی تبدیل فوریه ی $x[n] = \sin\left(2\pi \times 2000 \times \frac{n}{16000}\right)$ به صورت تئوری با فرض مقدار داشتن بین 0 تا : $N-1$:

$$DFT(x[n]) = X(k) = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi nk}{N}} = \sum_{n=0}^{N-1} \sin\left(2\pi \times 2000 \times \frac{n}{16000}\right) e^{\frac{-j2\pi n}{N}k}$$

$$= \sum_{n=0}^{N-1} \sin\left(\frac{n\pi}{4}\right) e^{\frac{-j2\pi n}{N}k} = \frac{1}{2j} \sum_{n=0}^{N-1} \left(e^{j\frac{n\pi}{4}} - e^{-j\frac{n\pi}{4}}\right) e^{\frac{-j2\pi n}{N}k}$$

$$= \frac{1}{2j} \sum_{n=0}^{N-1} e^{\frac{-j2\pi n}{N}\left(k-\frac{N}{8}\right)} - e^{\frac{-j2\pi n}{N}\left(k+\frac{N}{8}\right)}$$

$$= \frac{1}{2j}\left(\frac{e^{-j2\pi n\left(k-\frac{N}{8}\right)} - 1}{e^{\frac{-j2\pi n}{N}\left(k-\frac{N}{8}\right)} - 1}\right) - \frac{1}{2j}\left(\frac{e^{-j2\pi n\left(k+\frac{N}{8}\right)} - 1}{e^{\frac{-j2\pi n}{N}\left(k+\frac{N}{8}\right)} - 1}\right)$$

$$= \frac{1}{2j}\left(\frac{e^{-j\pi n\left(k-\frac{N}{8}\right)} \sin\left(2\pi n\left(k-\frac{N}{8}\right)\right)}{e^{\frac{-j\pi n}{N}\left(k-\frac{N}{8}\right)} \sin\left(\frac{2\pi n}{N}\left(k-\frac{N}{8}\right)\right)}\right) - \frac{1}{2j}\left(\frac{e^{-j\pi n\left(k-\frac{N}{8}\right)} \sin\left(2\pi n\left(k+\frac{N}{8}\right)\right)}{e^{\frac{-j\pi n}{N}\left(k-\frac{N}{8}\right)} \sin\left(\frac{2\pi n}{N}\left(k+\frac{N}{8}\right)\right)}\right)$$

صورت کسر ها مگر در حالتی که $k = \frac{N}{8}$ و یا $k = -\frac{N}{8}$ که همان $k = N - \frac{N}{8}$ است صفر می شود ؛ برای این دو حالت با جایگذاری مستقیم قبل از محاسبه مجموع سری هندسی داریم :

$$X(k) = \begin{cases} \frac{N}{2j} & k = \frac{N}{8} \text{ or } k = N - \frac{N}{8} \\ 0 & otherwise \end{cases}$$

که سیگنالی محدود است.

اگر فرض کنیم $x[n]$ خودش متناوب است :

$$DTFT(x[n]) = DTFT\left(\sin\left(\frac{n\pi}{4}\right)\right) = \frac{1}{j}\left(\delta\left(\omega - \frac{\pi}{4}\right) - \delta\left(\omega + \frac{\pi}{4}\right)\right)$$

که سیگنالی نامحدود و متناوب با 2π است.

پیاده سازی در C :

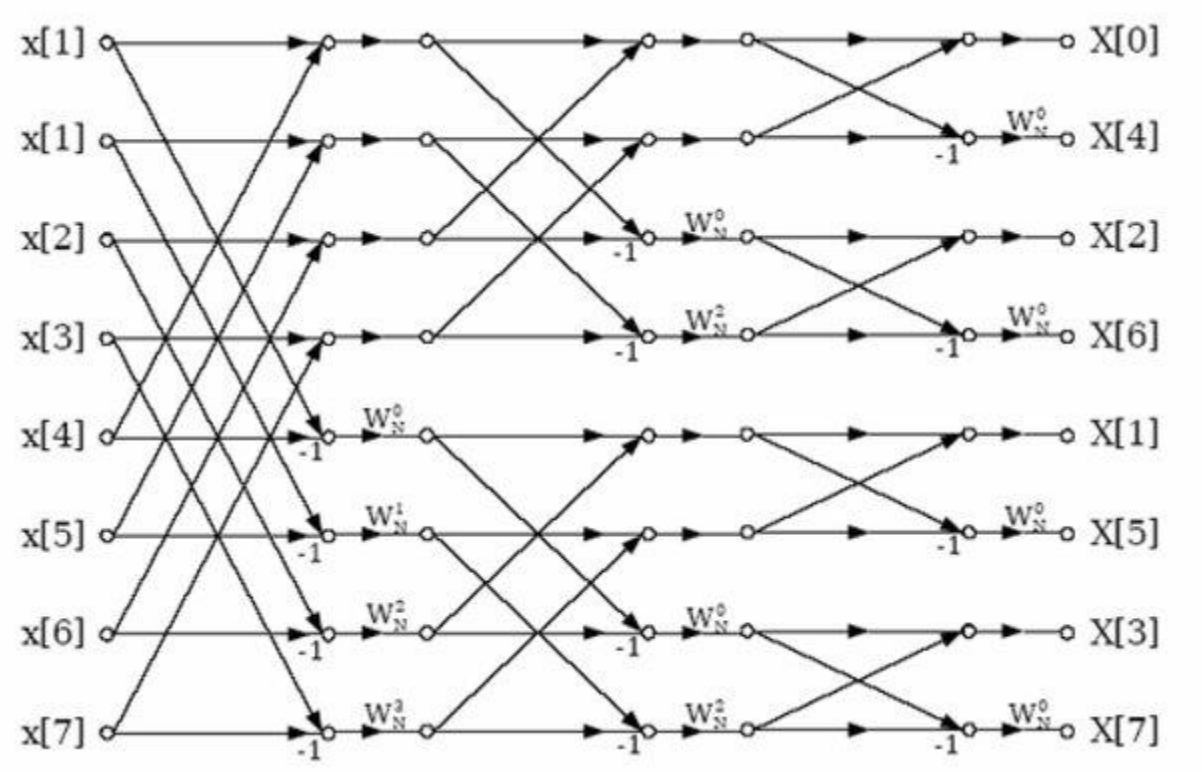
```c
#include <stdio.h>
#include <math.h>
#include "fft.c"
#include "Complex.c"

#define PI 3.14159

int main(){
    int i, j;
    float f[3] = {2000.0, 4000.0, 5000.0};
    float fs[3] = {16000.0, 20000.0, 24000.0};
    int N = 1024;

    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            float w = 2 * PI * f[i];
            float fs_val = fs[j];

            complex sine[N];

            for (int k = 0; k < N; k++){
                sine[k].x = sin(w * k / fs_val);
                sine[k].y = 0;
            }

            fft(sine, N, 1/fs_val);

            char filename[40];
            sprintf(filename, "../sine_fft_f%.1f_fs%.1f.txt", f[i], fs_val);
            FILE* fft_file = fopen(filename, "w+");

            for (int k = 0; k < N; k++){
                fprintf(fft_file, "%f ", ccabs(sine[k]));
            }

            fclose(fft_file);
        }
    }

    return 0;
}
```

مطابق خواسته سوال 3 تا f و 3 تا fs تعریف می کنیم و با گردش در دو لوپ تو در تو مقادیر fft سیگنال سینوسی به ازای این ترکیب های مختلف این فرکانس ها حساب می کنیم و روی فایل txt ذخیره می کنیم.

برای این کار ابتدا فرکانس زاویه ای را بر حسب f محاسبه می کنیم و سپس سیگنال سینوسی را با استفاده از تابع sin از کتابخانه math تعریف می کنیم (قسمت موهومی سیگنال را صفر قرار می دهیم).

سپس از کد تابع fft که در اختیارمان قرار داده شده استفاده می کنیم و با دادن ورودی های سیگنال سینوسی که بدست آوردیم تا حاصل fft روی خود آن ذخیره شود ، تعداد نقاط تبدیل و dt که برابر 1/fs است ، تبدیل فوریه را محاسبه می کنیم .

تابع fft الگوریتم پروانه ای مطابق شکل زیر را برای محاسبه fft پیاده سازی می کند:



در انتها اندازه تبدیل فوریه را با استفاده از تابع ساده ccabs که توان دو قسمت حقیقی و موهومی را جمع می کند روی فایل txt با نام شامل f و fs مربوطه ذخیره می کنیم.

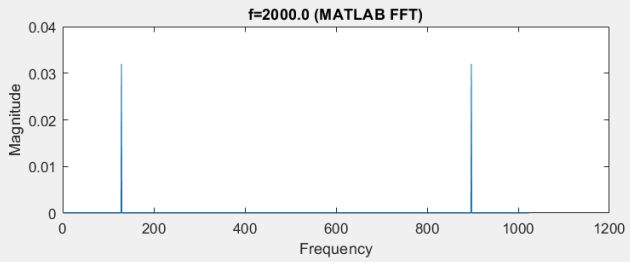
دیدن نتایج در متلب :

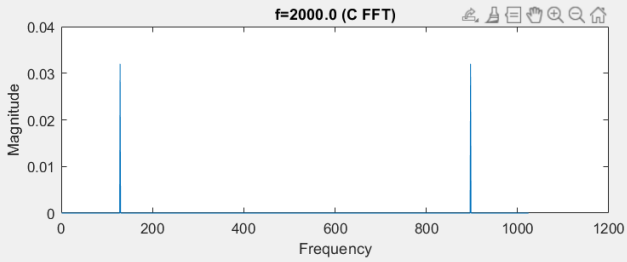
فایل های ذخیره شده را می خوانیم و رسم می کنیم

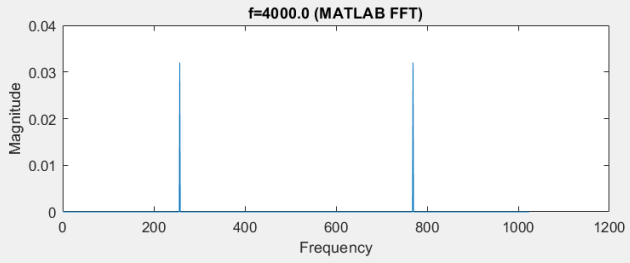
برای مطمین شدن از نحوه صحیح عملکرد کد در کنار fft محاسبه شده توسط C ، با استفاده از متلب هم fft را محاسبه می کنیم و پس از نرمال کردن در کنار هم رسم می کنیم:

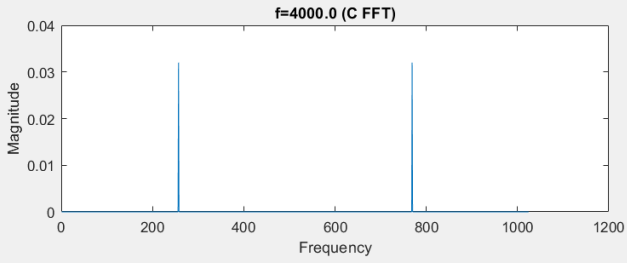
```
%%
f = [2000.0, 4000.0, 5000.0];
fs = [16000.0, 20000.0, 24000.0];
N = 1024;

for j = 1:length(fs)
    figure;
    sgtitle(sprintf('FFT Results for fs=%.1f', fs(j)));

    for i = 1:length(f)

        t = (0:N-1) / fs(j);
        x = sin(2*pi*f(i)*t);

        % Calculate FFT using MATLAB
        fft_matlab = fft(x);

        % Read FFT data calculated by C code
        filename = sprintf('sine_fft_f%.1f_fs%.1f.txt', f(i), fs(j));
        fft_c = dlmread(filename);

        subplot(length(f), 2, (i-1)*2 + 1);
        plot(abs(fft_matlab)/ fs(j));
        title(sprintf('f=%.1f (MATLAB FFT)', f(i)));
        xlabel('Frequency');
        ylabel('Magnitude');
        subplot(length(f), 2, (i-1)*2 + 2);
        plot(fft_c);
        title(sprintf('f=%.1f (C FFT)', f(i)));
        xlabel('Frequency');
        ylabel('Magnitude');
    end
end
```

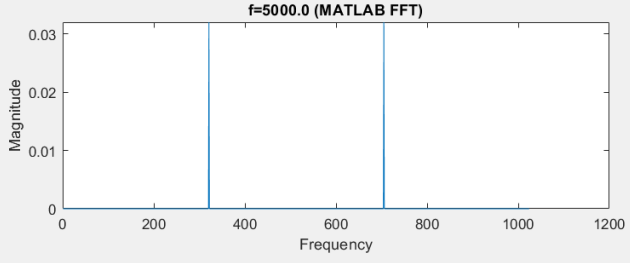
FFT Results for fs=16000.0

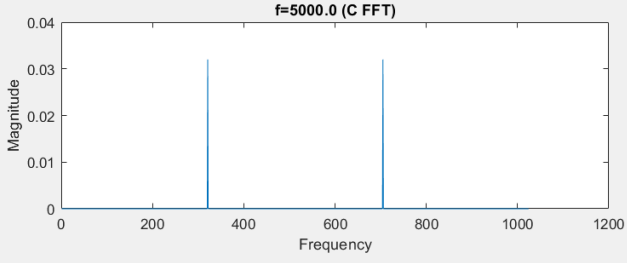




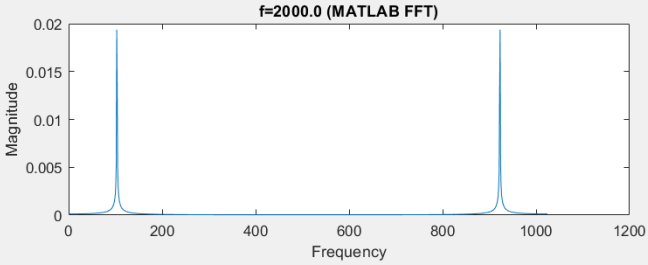


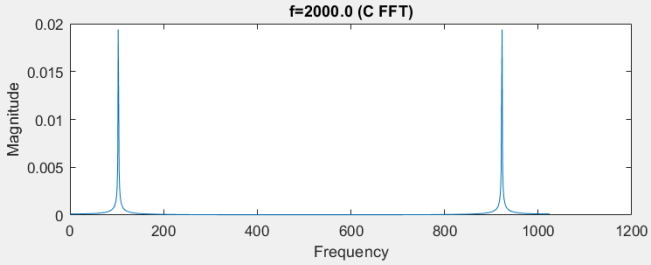


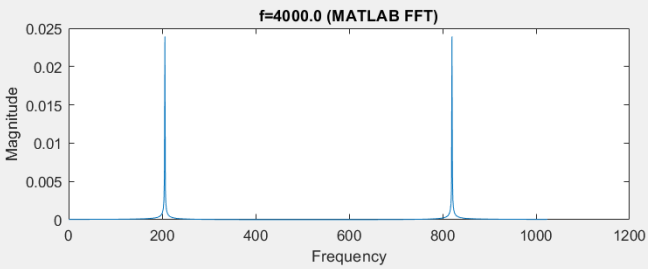


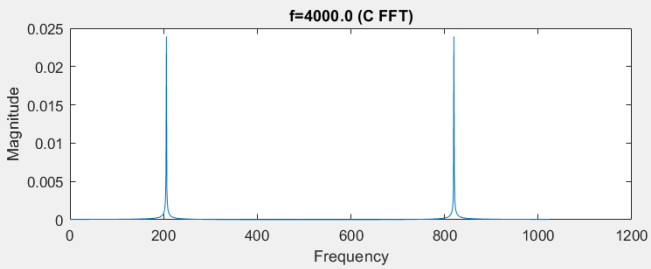


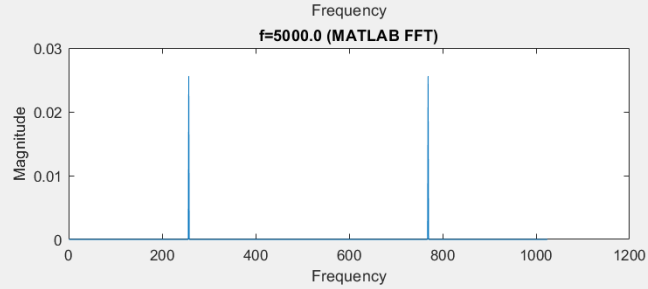
FFT Results for fs=20000.0

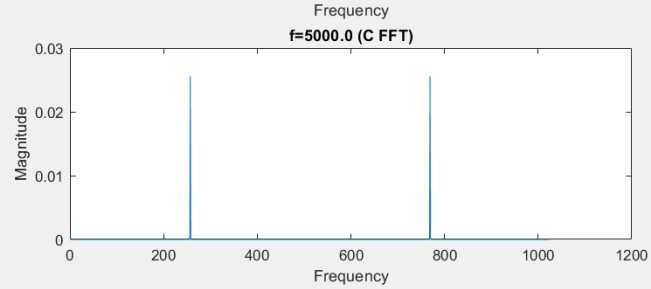


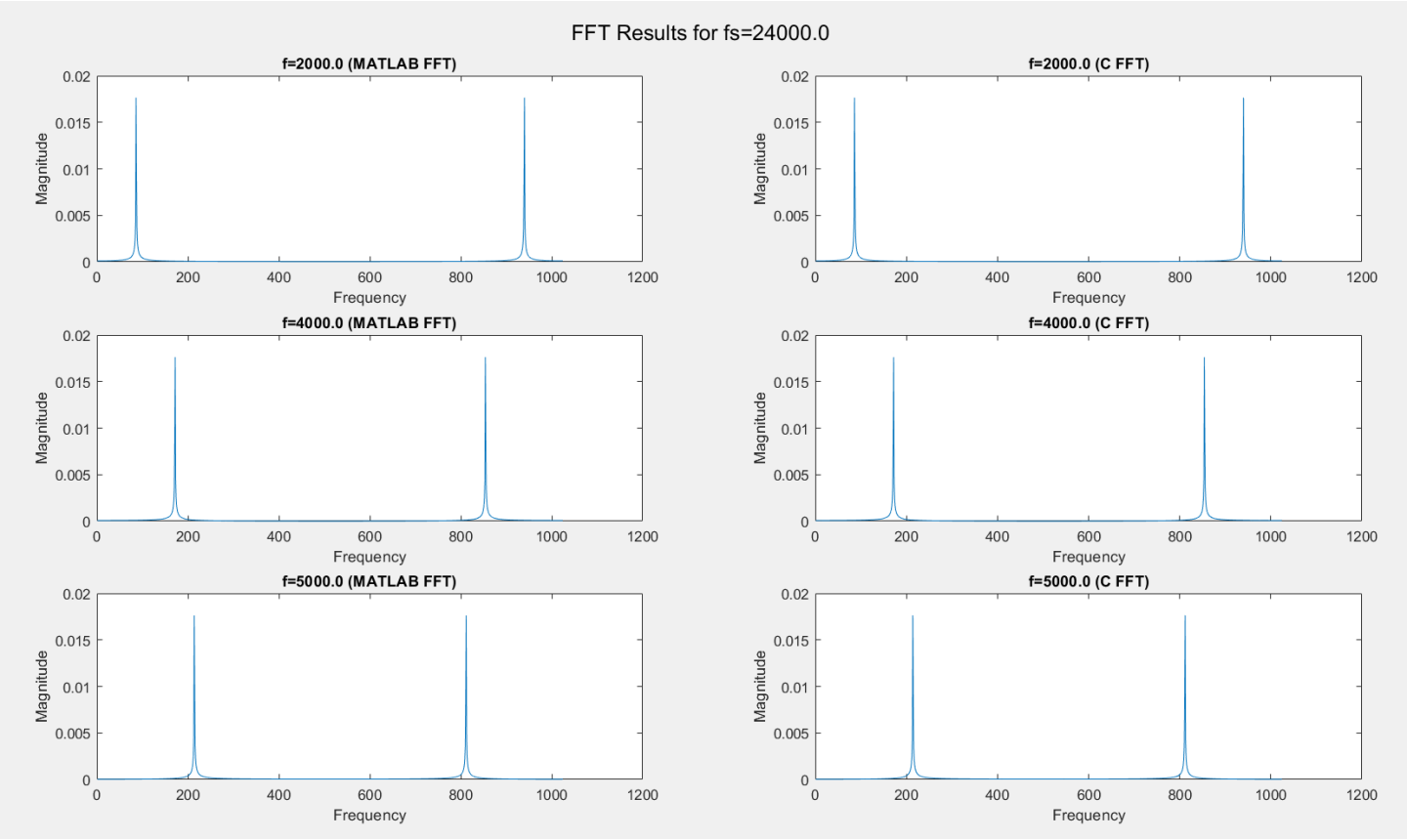












مطابق اشکال فوق می بینیم که fft توسط کد C مشابه آنچه توسط متلب محاسبه شده است و به درستی پیاده سازی شده.

آزمایش شماره 4 :

اثبات رابطه تئوری :

داریم :

$$Re[c_n] = a_n \xrightarrow{DFT} A_k$$

$$Im[c_n] = b_n \xrightarrow{DFT} B_k$$

می دانیم :

$$Re[x_n] \xrightarrow{DFT} \frac{1}{2}[X_k + X^*_{N-k}]$$

$$Im[c_n] \xrightarrow{DFT} \frac{1}{2}[X_k - X^*_{N-k}]$$

بنابراین داریم :

$$|A_k|^2 + |B_k|^2 = \frac{1}{4}|C_k + C^*_{N-k}|^2 + \frac{1}{4}|C_k - C^*_{N-k}|^2$$

$$= \frac{1}{4}(|C_k|^2 + |C_{N-k}|^2 + 2C_k C^*_{N-k} + |C_k|^2 + |C_{N-k}|^2 - 2C_k C^*_{N-k})$$

$$= \frac{1}{2}(|C_k|^2 + |C_{N-k}|^2)$$

پیاده سازی در C :

```
# define N 1024
# define L 8
# define len 8 * 1024
```

طول تبدیل فوریه را 1024 ، طول فریم را 8 و طول سیگنال ها را 8*1024 تعریف می کنیم که به ازای 8 فریم تبدیل fft را انجام دهیم.

```
double spectrum[N / 2 + 1] = {0};
double buffer[N / 2 + 1] = {0};
float a[N] = {0};
float b[N] = {0};
float signal[len] = {0};
complex ping[N];
complex pong[N];
```

با توجه به اینکه ورودی یک سیگنال حقیقی است طیق متقارن است و ما فقط قسمت مثبت را محاسبه و رسم می کنیم بنابراین اندازه طیف و بافر که در مراحل بعدی به ازای هر پنجره مقادیر طیف را ذخیره می کند و سپس پس از نرمالایز شدن به مقدار طیف اصلی اضافه می شود ، N/2+1 یا همان 513 تعیین می کنیم.

a,b همان A_k, B_k در رابطه ی تئوری هستند که ضرایب سری فوریه قسمت حقیقی و موهومی سیگنال هستند و طول آن ها برابر طول تبدیل فوریه و برابر N است.

اندازه سیگنال ورودی هم مطابق توضیحات قبل 1024*8 تعیین می شود.

دو آرایه ی مختلط ping , pong هم برای دخیره a,b در قسمت حقیقی و موهومی آن ها تعریف می کنیم که به تابع fft ورودی می دهیم.

```
for (int n = 0; n < N * L; n++){
    signal[n] = 10000 * cos( X: n * 100 * 2 * PI / 1024);
}
```

```
for (int n = 0; n < N * L; n++){
    signal[n] = 10000 * sin(n * 100 * 2 * PI / 512);
}
```

```
for (int n = 0; n < N * L; n++) {
    if ((n % 512) < 256) {
        signal[n] = 1;
    } else {
        signal[n] = 0;
    }
}
```

سیگنال های خواسته شده را در مطابق کد فوق تعریف می کنیم . برای اینکار برای توابع سینوس و کسنوس از کتابخانه math استفاده می کنیم و مطابق صورت آزمایش سیگنال به طول N*L را تعریف می کنیم ، برای سیگنال مربعی هم از یک if استفاده می کنیم که در صورتی باقی مانده n بر 512 از 256 کم تر باشد مقدار 1 و در صورتی که بیش تر باشد مقدار 0 را اختیار می کند.

```
    FILE* file;
    fopen_s(&file, Filename: "../cos_output.txt", Mode: "w");
//    fopen_s(&file, "../sin_output.txt", "w");
//    fopen_s(&file, "../rect_output.txt", "w");
```

هر بار یکی از این سیگنال ها را از کامنت خارج می کنیم و فایل با نام مربوطه را ایجاد و نتایج را در آن ذخیره می کنیم.

```
    int window_num = 0;
    for (window_num; window_num < floor( X: len / (2 * N)); window_num++) {
        coef_generator(signal, a, b, window_num);
        ISR(ping, pong, a, b);
        fft( a: pong,  n: 1024,  dt: 1);

        int j = 0;
        for (j; j < N / 2 + 1; j++) {
            buffer[j] += (ccabs( a: pong[j]) * ccabs( a: pong[j]) + ccabs( a: pong[N - 1 - j]) * ccabs( a: pong[N - 1 - j])) / 2 ;
            if ((window_num - 1) % L == 0){
                spectrum[j] += buffer[j] / (L* N);
                fprintf( stream: file,  format: "%lf\n", spectrum[j]);
                buffer[j] = 0;
            }}

    }
    return 0;
}
```

یک حلقه برای پردازش سیگنال در هر پنجره ایجاد می شود. برای هر پنجره، کد مراحل زیر را انجام می دهد:

- تابع coef_generator را برای تولید ضرایب a و b برای پنجره جاری سیگنال فراخوانی می کند.
- تابع ISR را برای اختصاص ضرایب به آرایه های مختلط پینگ و پنگ فراخوانی می کند.
- تابع fft را برای انجام محاسبه FFT روی آرایه پنگ فراخوانی می کند.
- طبق رابطه ی تئوری اثبات شده مجموع مجذور ضرایب را به استفاده از مجموع مجذور ضریب فوریه مختلط C_k و C_{N-k} محاسبه و در بافر ذخیره می کند.
- بررسی می کند که آیا پنجره فعلی مضربی از L است یا خیر. اگر درست باشد، به این معنی است که بافر برای یک پنجره کامل است و مقادیر بافر انباشته شده را بر (L * N) تقسیم می کند تا نرمالایز شود و آنها را به آرایه طیف اصلی اضافه می کند
- مقادیر طیف را در فایل خروجی می نویسد.

توضیح جزیی تر توابع و مراحل :

```
void coef_generator(float *signal, float *a, float *b, int window_num) {
    int i = 0;
    for (i; i < 2 * N; i++) {
        if (i < N)
            a[i] = signal[window_num * N + i];
        else
            b[i - N] = signal[window_num * N + i];
    }
}
```

تابع coef_generator بر حسب شماره پنجره فعلی حلقه بیرونی ، مقدار 1024 تای اول سیگنال را در پنجره اول که در هر حلقه هست یعنی a و مقدار 1024 تای دوم سیگنال را در پنجره دوم هر حلقه یعنی b ذخیره می کند.

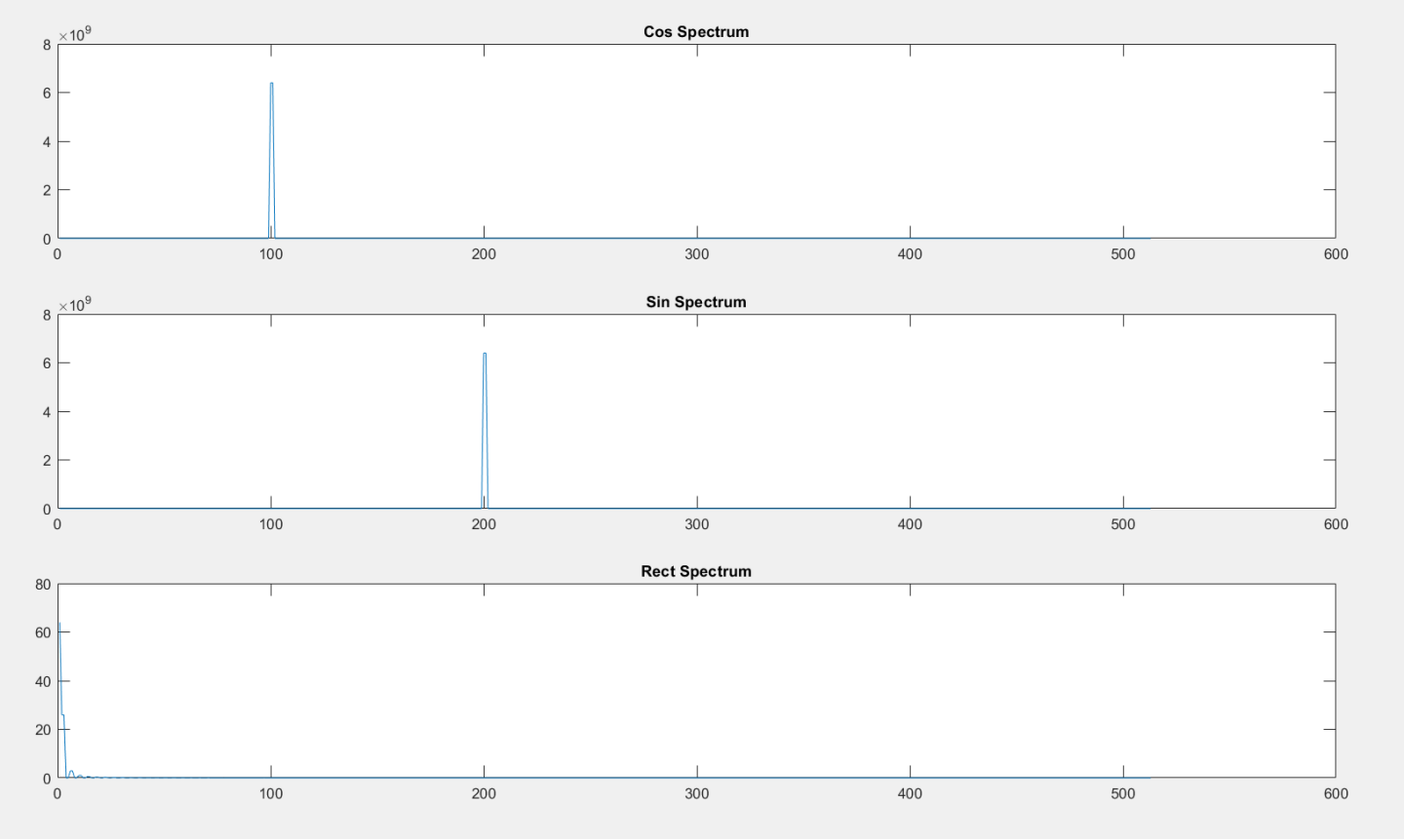
```
void ISR(complex *ping, complex *pong, float *a, float *b) {
    for (int i = 0; i < N; i++) {
        ping[i].x = a[i];
        ping[i].y = b[i];
        pong[i].x = a[i];
        pong[i].y = b[i];
    }
}
```

تابع ISR مقادیر a و b را در قسمت حقیقی و موهومی سیگنال های ping و pong ذخیره می کند ، البته در ادامه کد تنها نیاز به یکی از این دو آرایه برای محاسبه fft سیگنال داریم و می توان به تعریف یکی اکتفا کرد.

دیدن نتایج در متلب :

فایل های ذخیره شده را می خوانیم و رسم می کنیم

```
% Read the data from the C output files
fft_cos = dlmread('cos_output.txt');
fft_sin = dlmread('sin_output.txt');
fft_rect = dlmread('rect_output.txt');

figure;

subplot(3, 1, 1);
plot(fft_cos);
title('Cos Spectrum');

subplot(3, 1, 2);
plot(fft_sin);
title('Sin Spectrum');

subplot(3, 1, 3);
plot(fft_rect);
title('Rect Spectrum');
```



همانطور که انتظار داشتیم برای سیگنال کسینوسی که برابر $10000\cos\left(100n\ \frac{2\pi}{1024}\right)$ بود یک ضربه در فرکانس 100، برای سیگنال سینوسی که برابر $10000sin\left(100n\ \frac{2\pi}{512}\right)$ بود یه ضربه در فرکانس 200 و برای سیگنال مربعی تبدیل فوریه به فرم sinc است.