Лабораторная работа 2: Быстрое преобразование Фурье. Вычисление дискретной свертки.

Третьяк Илья Дмитриевич (Tretyak0ID@gmail.com)

Номер в списке = 20 => Вариант 20+1 (mod 4) = 1

Цель работы: Изучить алгоритм быстрого преобразования Фурье (БПФ) и убедиться в ускорении вычислений при его использовании.

Задание 1

> Реализовать на C или C++ алгоритмы непосредственного вычисления ДПФ и ОДПФ по формулам (1) и (2) для комплексного входного сигнала с двойной точностью (double). Входные данные загружать из текстового файла (разделитель – пробел), сгенерированного, например, в MATLAB.

Листинг методов ДПФ на C++

```
//-----------------------------------------------DFT-METHODS---------------------------
--------------------
//Discrate Fourier transford for complex number vector X
vector<complex<double>> dft(vector<complex<double>> &X){
    double N = size(X);

    vector<complex<double>> Y(N);

    //Measuring-time----------------------------------
    auto start = chrono::high_resolution_clock::now();
    //Measuring-time----------------------------------

    for (double k=0; k < N; k++){
        for (double j=0; j < N; j++){
            Y[k] = Y[k]+1/sqrt(N)*X[j]*exp(-2*PI*complex<double>(0,1)*k*j/N);
        }
    }

    //Measuring-time--------------------------------
    auto end = chrono::high_resolution_clock::now();
  chrono::duration<float> duration = end - start;
    cout << "Time: " << duration.count() << "\n";
    //add result in file
    ofstream fin("data/time_measure.txt", ios::app);
    fin << "Time DFT for N=" << N << " " << duration.count() << "\n";
    fin.close();
```

```
    //Measuring-time---------------------------------

    return Y;
};

//Inverse discrate Fourier transform of a complex vector
vector<complex<double>> odft(vector<complex<double>> &Y){
    double N = size(Y);

    vector<complex<double>> X(N);

    for (double k=0; k < N; k++){
        for (double j=0; j < N; j++){
            X[k] = X[k]+1/sqrt(N)*Y[j]*exp(2*PI*complex<double>(0,1)*k*j/N);
        }
    }

    return X;
};
```

Листинг кода MATLAB

```
n = 4; %Размерность векторов для преобразований Фурье и сверток

%ГЕНЕРАЦИЯ-СИГНАЛОВ-ДЛЯ-ФУРЬЕ-ПРКОБРАЗОВАНИЯ------------
t = linspace(-5,5,2^n);
st = -1000; en = 1000;
x = randi([st en], 1, length(t))/100;
y = randi([st en], 1, length(t))/100;
%сохранение в файл
fid = fopen('data/real.txt', 'wt');
fprintf(fid, '%f ', x);
fclose(fid);
type('data/real.txt');

fid = fopen('data/imag.txt', 'wt');
fprintf(fid, '%f ', y);
fclose(fid);
type('data/imag.txt');
```

В общих словах

В MATLAB генерируем случайный два случайных вещественных сигнала, соханяем в 2 файла `real.txt` для вещественной и `imag.txt` мнимой части соответственно.

Далее в C++ читаем полученные файлы в 2 `vector<double>` `r` и `i` , из которых в последствии делаем один `vector<complex<double>>` , как вещественную и мнимую части его элементов.

ДПФ и ОДПФ реализованы непосредственно (в листинге выше) `dft` и `odft`.

Также реализованы методы считывания из 2х файлов в вещественными числами комплексного массива, один из файлов служит основой вещественных частей элементов массива, а второй - мнимой. Метод `complex_vector_read`.

А также метод для печати массива комплексных чисел на экран `print_complex_vector`.

Пример работы

Произведем 3 преобразования: прямое дискретное исходного вектора, обратное дискретное исходного вектора и обратное преобразование прямого преобразования.

(Компиляцию производил при помощи gcc)

```
(base) ilatretak@MacBook-Pro Lab2 % ./DFT.o
-----Complex-vector-X------
7.61 + 3.6i
4.13 + 6.36i
-2.04 + -0.22i
6.8 + 0.82i
4.39 + -2.72i

-----Complex-vector-dft(X)------
9.005 + -4.365i
1.90277 + 2.79766i
0.47163 + 1.31225i
3.67462 + 2.79154i
2.33 + -6.855i

-----Complex-vector-odft(dft(X))------
7.61 + 3.6i
4.13 + 6.36i
-2.04 + -0.22i
6.8 + 0.82i
4.39 + -2.72i
```

Для читаемости выведены первые 5 компонент векторов, вообще они имеют размерность 2^4.

Задание 2

> Реализовать на C или C++ алгоритмы прямого и обратного БПФ для комплексного входного сигнала длиной 2^n , n – любое натуральное число:
> а) с прореживанием по времени и двоично-инверсными перестановками

Листинг методов БПФ и ОБПФ на языке C++

```
//----------------------------------------------FFT-METHODS----------------------------
--------------------
//Calculus W coefficient
complex<double> W(int l, int k){
  complex<double> i = complex<double>(0, 1);
```

```
  complex<double> w = exp((-2*M_PI*l/pow(2, k))*i);
  return w;
}

//Calculus binary invert index
int invert(int i, int n){
  int u = 0;
  int q;
  for (int k = 0; k < n; ++k)
  {
    q = i % 2;
    if (q == 1) u = u + pow(2, n - k - 1);
    i = i / 2;
  }
  return u;
}

//Fast Fourier transford for complex number vector Z
vector<complex<double>> fft(vector<complex<double>> Z){

    int N = size(Z);
    int n = log2(N);
    vector<complex<double>> X = Z;

    for (int i = 0; i < N; ++i) {
      int u = invert(i, n);
      if (u >= i) {
        complex<double> r = X[i];
        X[i] = X[u];
        X[u] = r;
    }
    }

    //Measuring-time-----------------------------------
    auto start = chrono::high_resolution_clock::now();
    //Measuring-time-----------------------------------

    for (int k=1; k <= n; k++){

        for (int j=0; j < pow(2,n-k); j++){

            for (int l=0; l < pow(2, k-1); l++){
                complex<double>a = X[j * (pow(2, k)) + l];
                X[j*pow(2, k) + l]               = X[j*pow(2, k) + l] + W(l,
k)*X[j*pow(2, k) + l + pow(2, k-1)];
                X[j*pow(2, k) + l + pow(2, k-1)] = a - W(l, k)*X[j*pow(2, k) + l +
pow(2, k-1)];

            };
```

```
        };
    };

    for (int i = 0; i < N; i++){
    X[i] = X[i]/sqrt(N);
  }

    //Measuring-time--------------------------------------------------
    auto end = chrono::high_resolution_clock::now();
  chrono::duration<float> duration = end - start;
    cout << "Time: " << duration.count() << "\n";
    //add result in file
    ofstream fin("data/time_measure.txt", ios::app);
    fin << "Time FFT for N=" << N << " " << duration.count() << "\n";
    fin.close();
    //Measuring-time---------------------------------------------------

    return X;
}

//Inverse Fast Fourier transford for complex number vector Z
vector<complex<double>> offt(vector<complex<double>> Z) {
    int N = size(Z);
    int n = log2(N);

  vector<complex<double>> X(N);
  vector<complex<double>> Y(N);

  for (int i = 0; i < pow(2, n); i++) {
    Y[i] = conj(Z[i]);
  }

  X = fft(Y);

  for (int i = 0; i < pow(2, n); i++) {
    X[i] = conj(X[i]);
  }

    Y.clear();
  return X;
}
```

Методы `W` и `invert` вычисляют коэффциенты матрицы ω преобразования Фурье и двоично-инвертированный индекс для исходного.

Далее в методе `fft` реализовано прямое быстрое преобразование Фурье с пропеживаем по времени с перестановками. В `offt` обратное.

Загрузка преобразуемого вектора происходит точно так же, как это сделано выше

Пример работы

Данные и условия как выше, только вместо ДПФ/ОДПФ используем БПФ/ОБПФ

```
-----Complex-vector-X------
7.61 + 3.6i
4.13 + 6.36i
-2.04 + -0.22i
6.8 + 0.82i
4.39 + -2.72i

-----Complex-vector-fft(X)------
9.005 + -4.365i
1.90277 + 2.79766i
0.47163 + 1.31225i
3.67462 + 2.79154i
2.33 + -6.855i

-----Complex-vector-offt(fft(X))------
7.61 + 3.6i
4.13 + 6.36i
-2.04 + -0.22i
6.8 + 0.82i
4.39 + -2.72i
```

Задание 3

> Убедиться в корректности работы алгоритмов:
>
> а) проверить выполнение равенства X = ОДПФ(ДПФ(X)), а также равенства X = (ОБПФ(БПФ(X));
>
> б) сравнить результаты ДПФ(X) и БПФ(X);
> в) сравнить результаты работы реализованного алгоритма, например, с результатами процедуры fft, встроенной в MATLAB.
>
> (рекомендуется для сравнения использовать значение ошибки)

а.1) Равенство X=ОДПФ(ДПФ(X))

убедимся в этом, используя метод вычисление максимального обсолютного отклонения 2х элементов комплексного вектора `max_absolut_error`

```
//-----------------------------------METHODS-FOR-EVALUTING-EFFECTIVNESS---------------
---------------------
//Calculus maximum absolut value error
double max_absolut_error(vector<complex<double>> X, vector<complex<double>> Y, string
messege){
    double maximum = 0;
```

```
    for (int i=0; i < size(X); i++){
        if(abs(X[i]-Y[i]) > maximum){
            maximum = abs(X[i]-Y[i]);
        }
    }

    cout <<"maximum absolut difference "<< messege << ": " << maximum << "\n";
    return maximum;
};
```

Получаем результат:

```
maximum absolut difference ODFT(DFT(X)): 3.57513e-14
```

Видим, что погрешность близка к машинному нулю. В этом можно убедиться и посмотрев на векторы выше, это равенство выполняется с очень большой точностью.

а.2) Равенство X=ОБПФ(БПФ(X))

Аналогично:

```
maximum absolut difference DFT(ODFT(X)): 2.22045e-15
```

б) сравнить результаты ДПФ(X) и БПФ(X);

Для этого совместим 2 метода вычисления `fft` и `dft` для одного вектора и методом `max_absolut_error` вычислим максимальную погрешность

```
maximum absolut difference DFT and FFT: 4.974e-14
```

Видим, что разница крайне мала.

в) сравнить результаты работы реализованного алгоритма, например, с результатами процедуры fft, встроенной в MATLAB.

Для этого сгенерируем в матлабе сигнал, преобразуем его быстрым преобразование Фурье `fft`, сохраним в файл

```
%МАТЛАБОВСКОЕ-ФУРЬЕ-ПРЕОБРАЗОВАНИЕ---------------------
vect1     = x + j.*y;
fft_vect1 = fft(vect1)./sqrt(length(vect1));

fid = fopen('data/matlab_fft_real.txt', 'wt');
fprintf(fid, '%f ', real(fft_vect1));
fclose(fid);
type('data/matlab_fft_real.txt');
fid = fopen('data/matlab_fft_imag.txt', 'wt');
fprintf(fid, '%f ', imag(fft_vect1));
fclose(fid);
type('data/matlab_fft_real.txt');
```

Затем загрузим в cpp-файл и вычислим `max_absolut_error` для `fft` в C++ и `fft` в MATLAB

```
//Task_3
vector<complex<double>> MATLAB_X_FFT  = complex_vector_read("data/matlab_fft_real.txt",
"data/matlab_fft_imag.txt");

double dif_matlab_fft_fft   = max_absolut_error(FFT_X, MATLAB_X_FFT, "MATLAB fft
error");
```

Получаем результат:

```
maximum absolut difference MATLAB fft error: 6.59124e-07
```

Задание 4

> Проанализировать зависимость времени выполнения БПФ и непосредственного вычисления ДПФ от длины N преобразования.

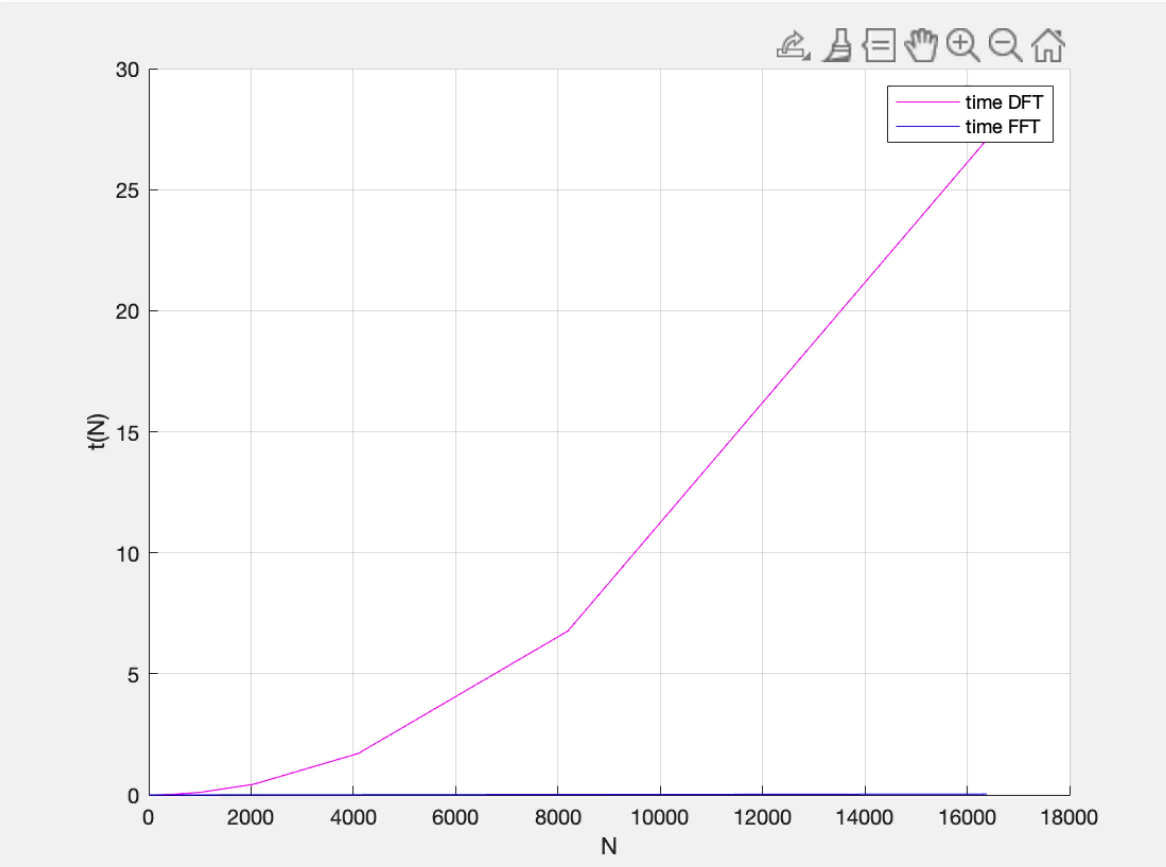
Для этого проведем генерацию в MATLAB массивыв различной размерности: $2^2 - 2^{14}$, будем замерять время работы `fft` и `dft`, причем будем сохранять эту статистику в текстовый фал `time_measure.txt`.

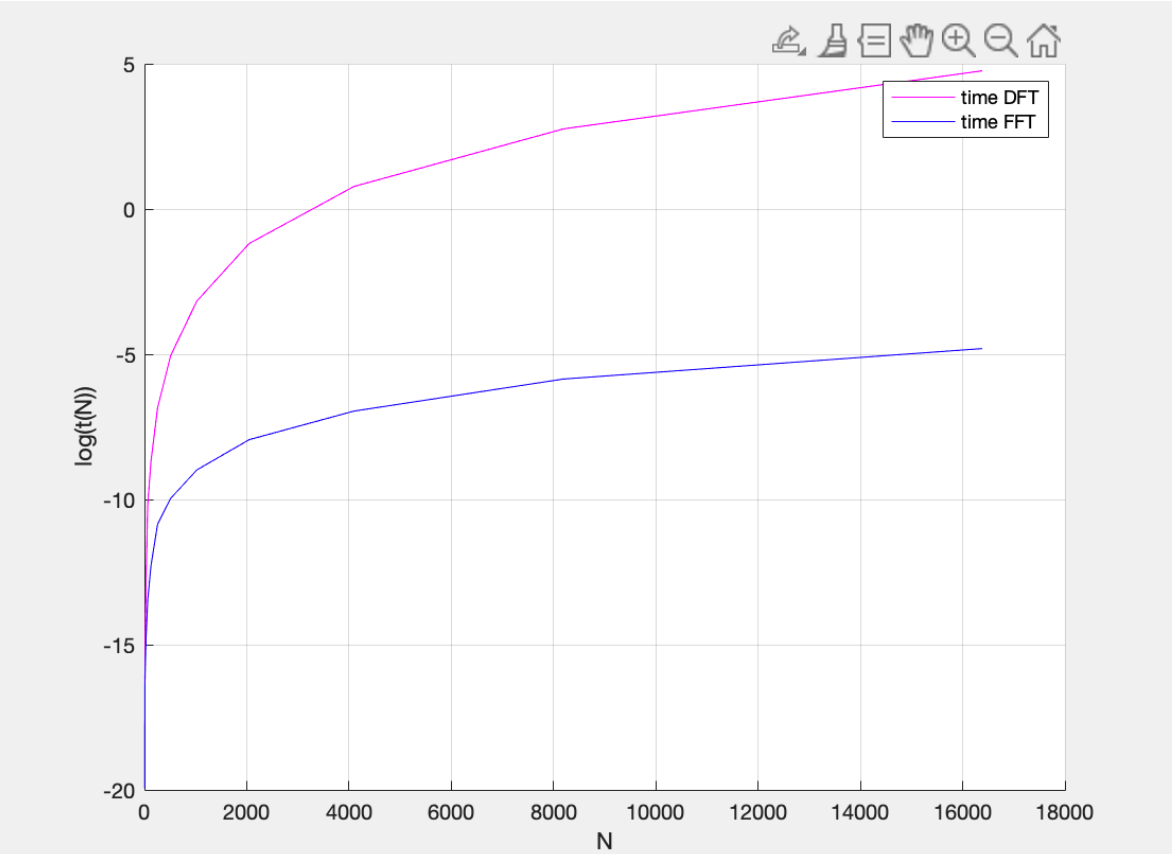
Далее в MATLAB занесем полученные данные и построим графики зависимости времени вычисления БПФ и ДПФ от длины вектора N. Также для наглядности представим результаты на логорифмической шкале.

```
DFT = [1e-06, 1.3e-05, 4e-05, 0.00015, 0.000872, 0.002446, 0.008645, 0.030223,
0.111084, 0.439266, 1.71226, 6.76674, 27.0718];
FFT = [1e-06, 1.1e-05, 1.8e-05, 3.7e-05, 8.9e-05, 0.000202, 0.000539, 0.001002,
0.001976, 0.004069, 0.008045, 0.017287, 0.035784];
N   = zeros(1,13);

for i=1:13
    N(i) = 2^(i+1);
```

```
end

hold on; grid on;
plot(N, DFT, 'm')
plot(N, FFT, 'b')
xlabel('N'); ylabel('t(N)'); legend('time DFT', 'time FFT')

figure
hold on; grid on;
plot(N, log2(DFT), 'm')
plot(N, log2(FFT), 'b')
xlabel('N'); ylabel('t(N)'); legend('time DFT', 'time FFT')
```

Получаем результат:





Как видно, быстрое преобразование Фурье работает куда эффективнее по времени, нежели обычное дискретное преобразование, это отличие становится существенным при N>500, при N<=500 время работы алгоритмов примерно сравнимо.

Задание 5

> Реализовать на C или C++ процедуру прямого вычисления свертки двух последовательностей по формуле (3). Входные данные загружать из текстового файла (разделитель – пробел), сгенерированного, например, в MATLAB.

Векторы для свертки будем генерировать в MATLAB и аналогичным образом сохранять в текстовые файлы

```
%ГЕНЕРАЦИЯ-СИГНАЛОВ-ДЛЯ-СВЕРТОК-------------------------
t = linspace(-5,5,2^n);
st = -1000; en = 1000;
x1 = randi([st en], 1, length(t))/100;
y1 = randi([st en], 1, length(t))/100;
x2 = randi([st en], 1, length(t))/100;
y2 = randi([st en], 1, length(t))/100;
%сохранение в файл
fid = fopen('data/real_convolution_x.txt', 'wt');
fprintf(fid, '%f ', x1);
fclose(fid);
type('data/real_convolution_x.txt');
```

```
fid = fopen('data/imag_convolution_x.txt', 'wt');
fprintf(fid, '%f ', y1);
fclose(fid);
type('data/imag_convolution_x.txt');

fid = fopen('data/real_convolution_y.txt', 'wt');
fprintf(fid, '%f ', x2);
fclose(fid);
type('data/real_convolution_y.txt');

fid = fopen('data/imag_convolution_y.txt', 'wt');
fprintf(fid, '%f ', y2);
fclose(fid);
type('data/imag_convolution_y.txt');
```

Загружаем из текстовых файлов массивы комплексных чисел, выполяем преобразование методом `convolution` в файле `convolution.cpp`

```
//-----------------------------------------------CONVOLUTION---------------------------
--------------------
//Analitic convilution
vector<complex<double>> convolution(vector<complex<double>> X, vector<complex<double>>
Y){
    vector<complex<double>> out_vector(size(X)+size(Y)-1);

    //Measuring-time----------------------------------
    auto start = chrono::high_resolution_clock::now();
    //Measuring-time----------------------------------

    for (int n=0; n<size(X)+size(Y)-1; n++){
        for (int k=0; k<size(X); k++){
            if (((n - k) >= 0) && ((n - k) < size(Y)) && (k < size(X))) {
                out_vector[n] = out_vector[n] + X[k]*Y[n-k];
            }
        }
    }

    //Measuring-time----------------------------------------------------------------
    auto end = chrono::high_resolution_clock::now();
  chrono::duration<float> duration = end - start;
    cout << "Time: " << duration.count() << "\n";
    //add result in file
    ofstream fin("data/time_measure.txt", ios::app);
    fin << "Time analitic convolution for N=" << size(X) << " " << duration.count() <<
"\n";
    fin.close();
    //Measuring-time-----------------------------------------------------------------

    return out_vector;
```

```
}
```

получаем результат:

```
-----Complex-vector-convolution------
-45.783 + 47.8698i
98.9731 + -19.9328i
-25.8132 + 25.9631i
88.6355 + 99.0108i
-130.256 + -109.19i
172.303 + 156.926i
-60.3587 + -39.8822i
29.0227 + 75.6387i
81.6745 + -195.187i
-35.1897 + 150.164i
20.1476 + 10.9808i
129.885 + 20.1577i
119.108 + -240.373i
-12.9767 + 34.4767i
146.36 + -243.501i
-28.0972 + 296.187i
-17.3002 + 23.5061i
-56.5316 + 40.6601i
76.0823 + -263.82i
26.738 + 255.829i
276.81 + 216.084i
-245.852 + 222.521i
22.6703 + -217.388i
108.401 + -62.0505i
45.0745 + -129.159i
-204.759 + 244.497i
62.499 + 29.8669i
-118.427 + -77.5855i
101.784 + 52.6526i
17.1185 + -83.9522i
-61.6846 + 33.1874i
```

Сказать об этом пока нечего, продолжим исследование дальше...

Задание 6

> Реализовать процедуру нахождения дискретной свертки, основанную на БПФ. При вычислении БПФ использовать результаты п. 2 задания.

При помощи метода `fft_convolution` выполним свертку на основе быстрого преобразования Фурье тех же самых векторов.

```
//FFT convolution
vector<complex<double>> fft_convolution(vector<complex<double>> x,
vector<complex<double>> y){
    int L = max(2*size(x), 2*size(y));
    vector<complex<double>> out(L);
    vector<complex<double>> X(L);
    vector<complex<double>> Y(L);

    //Measuring-time----------------------------------
    auto start = chrono::high_resolution_clock::now();
    //Measuring-time----------------------------------

    for (int i = 0; i < L; i++) {
    if (i < size(x)) X[i] = x[i];
    else X[i] = 0;

    if (i < size(y)) Y[i] = y[i];
    else Y[i] = 0;}

    X = fft(X);
    Y = fft(Y);

    for (int i = 0; i < L; i++) {
        out[i] = sqrt(L)*X[i]*Y[i];
    }

    out = offt(out);

    //Measuring-time--------------------------------------------------
    auto end = chrono::high_resolution_clock::now();
  chrono::duration<float> duration = end - start;
    cout << "Time: " << duration.count() << "\n";
    //add result in file
    ofstream fin("data/time_measure.txt", ios::app);
    fin << "Time FFT convolution for Nx=" << size(x) << " and Ny=" << size(y) << " " <<
duration.count() << "\n";
    fin.close();
    //Measuring-time--------------------------------------------------

    return out;
}
```

Получаем результат:

```
-----Complex-vector-convolution------
-45.783 + 47.8698i
98.9731 + -19.9328i
-25.8132 + 25.9631i
```

```
88.6355 + 99.0108i
-130.256 + -109.19i
172.303 + 156.926i
-60.3587 + -39.8822i
29.0227 + 75.6387i
81.6745 + -195.187i
-35.1897 + 150.164i
20.1476 + 10.9808i
129.885 + 20.1577i
119.108 + -240.373i
-12.9767 + 34.4767i
146.36 + -243.501i
-28.0972 + 296.187i
-17.3002 + 23.5061i
-56.5316 + 40.6601i
76.0823 + -263.82i
26.738 + 255.829i
276.81 + 216.083i
-245.852 + 222.521i
22.6703 + -217.388i
108.401 + -62.0505i
45.0745 + -129.159i
-204.758 + 244.497i
62.499 + 29.8669i
-118.427 + -77.5855i
101.784 + 52.6526i
17.1185 + -83.9522i
-61.6846 + 33.1874i
```

Можно заметить, что векторы очень похожи...

Задание 7

Посмотрим насколько же похожи эти векторы:

> Убедится в корректности работы процедуры из п. 5 и п. 6 задания, сравнив полученные результаты с результатами работы встроенной функций MATLAB conv.
>
> (рекомендуется для сравнения использовать значение ошибки)

Для этого сравним результаты как друг с другом (аналогичным методом сравнения векторов косплксных чисел `max_absolut_error`)

```
maximum absolut difference analitic and fft convolution: 1.42153e-13
```

Видим, что различие в результатах близко к машинному нулю.

Также свернем эти сигналы в MATLAB фукнцией `conv`, загрузим результаты в txt-файл, выгрузим в cpp и произведем сравнение с написаными нами методами:

```
MATLAB_CONV = complex_vector_read("data/matlab_conv_real.txt",
"data/matlab_conv_imag.txt");

double dif_matlab_and_fft_conv    = max_absolut_error(MATLAB_CONV, FFT_CONV,    "matlab
and fft convolution");
double dif_matlab_and_analit_conv = max_absolut_error(MATLAB_CONV, ANALIT_CONV, "matlab
and analitic convolution");
```

Имеем благоприятные результаты - вычисление свертки в MATLAB и написанными нами методоми почти не отличимы.

```
maximum absolut difference matlab and fft convolution: 1.10263e-13
maximum absolut difference matlab and analitic convolution: 8.53391e-14
```

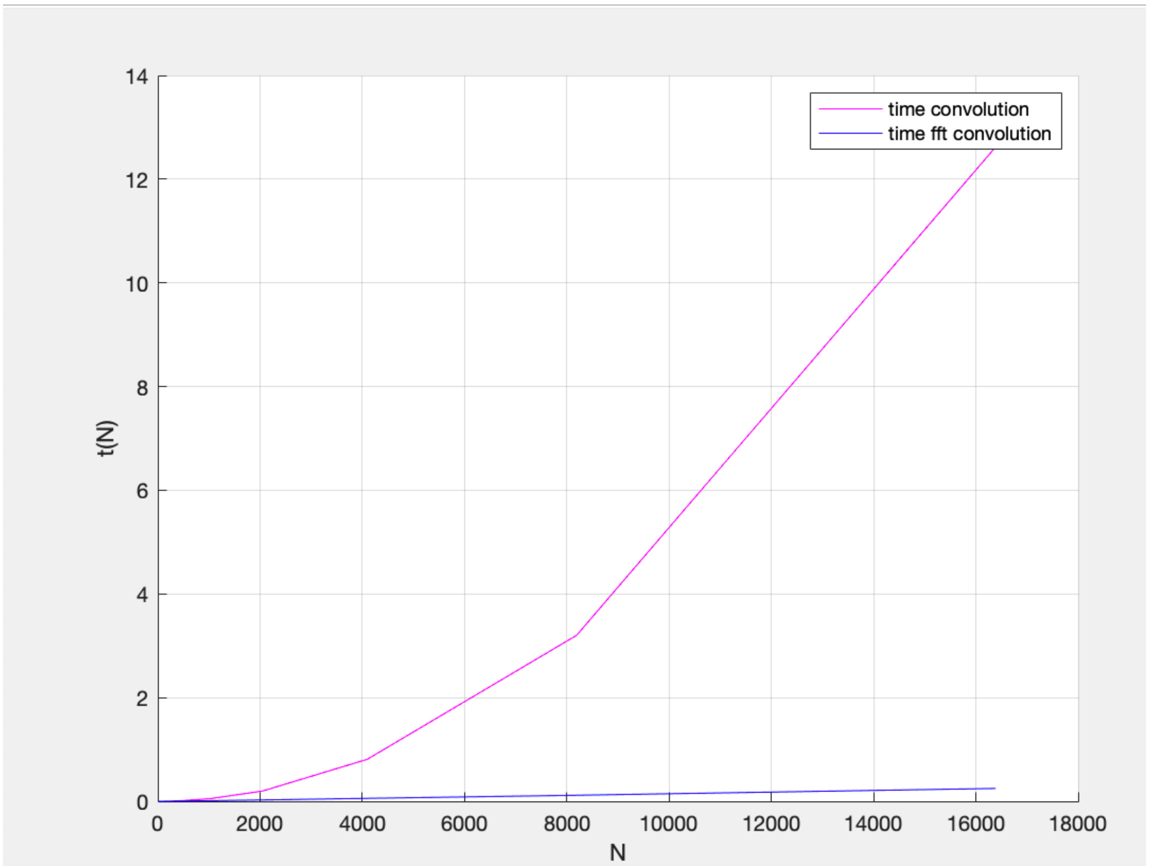
Задание 8

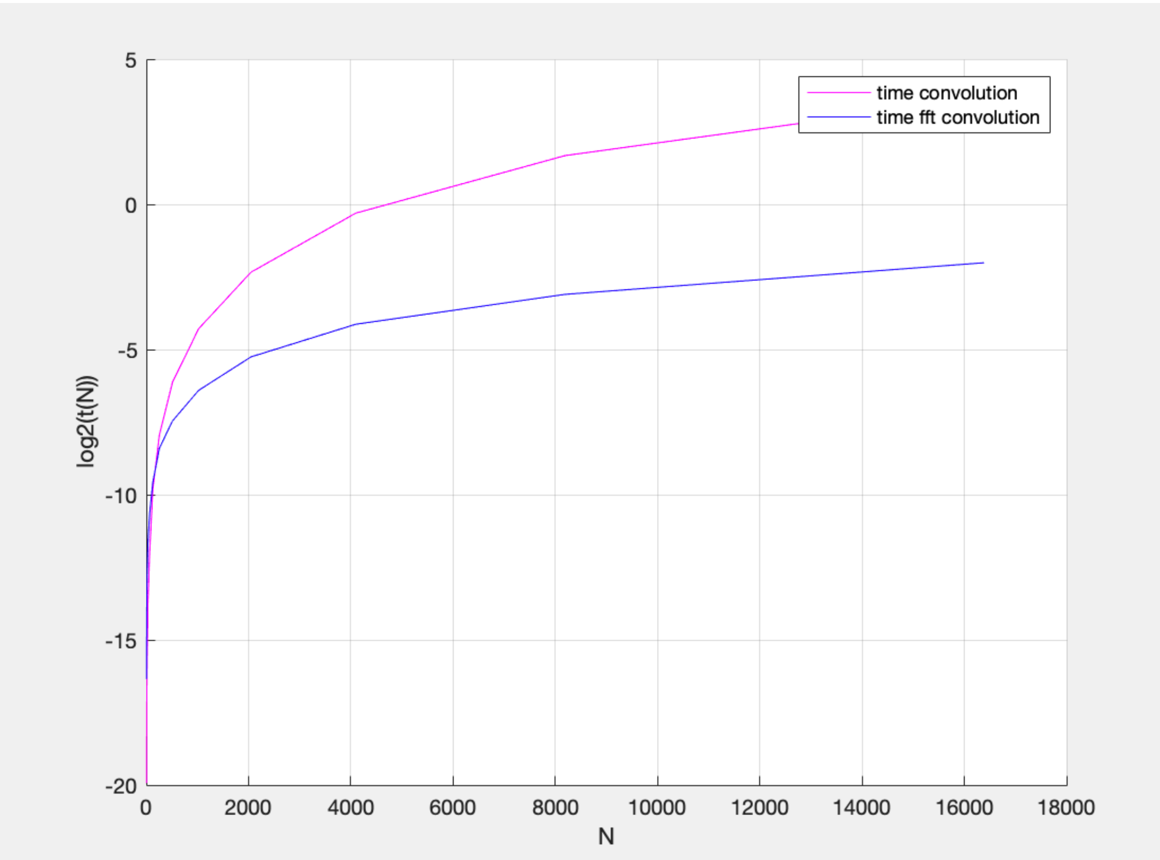
> Сравнить производительность алгоритмов вычисления свертки по
>
> определению (3) и с помощью БПФ в двух случаях: когда размер одной из последовательностей фиксирован, и когда меняются длины обеих последовательностей.

а) С меняющимися длинами обеих последовательностей

Снова каждый раз, считая свертку для векторов размерности $2^2 - 2^{13}$, будем запоминать время выполнения в txt-файл, затем собранную статистику визуализируем в MATLAB.

```
%Анализ эффективности свертки
convol     = [1e-06, 6e-06, 2e-05, 7.3e-05, 0.000238, 0.001114, 0.004074, 0.014295,
0.051315, 0.197949, 0.81008, 3.20245, 12.618];
fft_convol = [1.2e-05, 6.1e-05, 0.000143, 0.000311, 0.000615, 0.001314, 0.002962,
0.00568, 0.011798, 0.026213, 0.057111, 0.116641, 0.247899];

figure
hold on; grid on;
plot(N, convol, 'm')
plot(N, fft_convol, 'b')
xlabel('N'); ylabel('t(N)'); legend('time convolution', 'time fft convolution')

figure
hold on; grid on;
plot(N, log2(convol), 'm')
plot(N, log2(fft_convol), 'b')
xlabel('N'); ylabel('log2(t(N))'); legend('time convolution', 'time fft convolution')
```





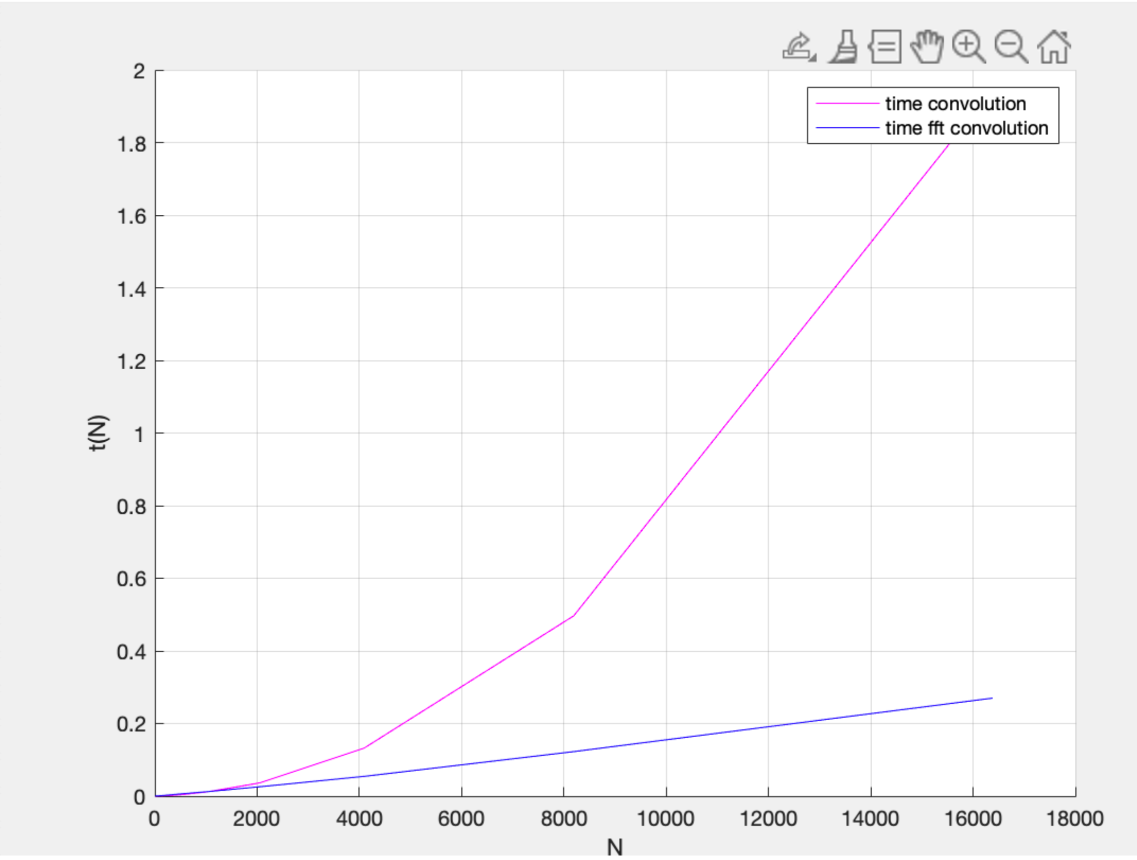
Видим, что при больших N эффективность по времени свертки, вычисляемой на основе быстрого преобразования Фурье куда выше. Для наглядности приведены графики логарифма от анализируемой величины.

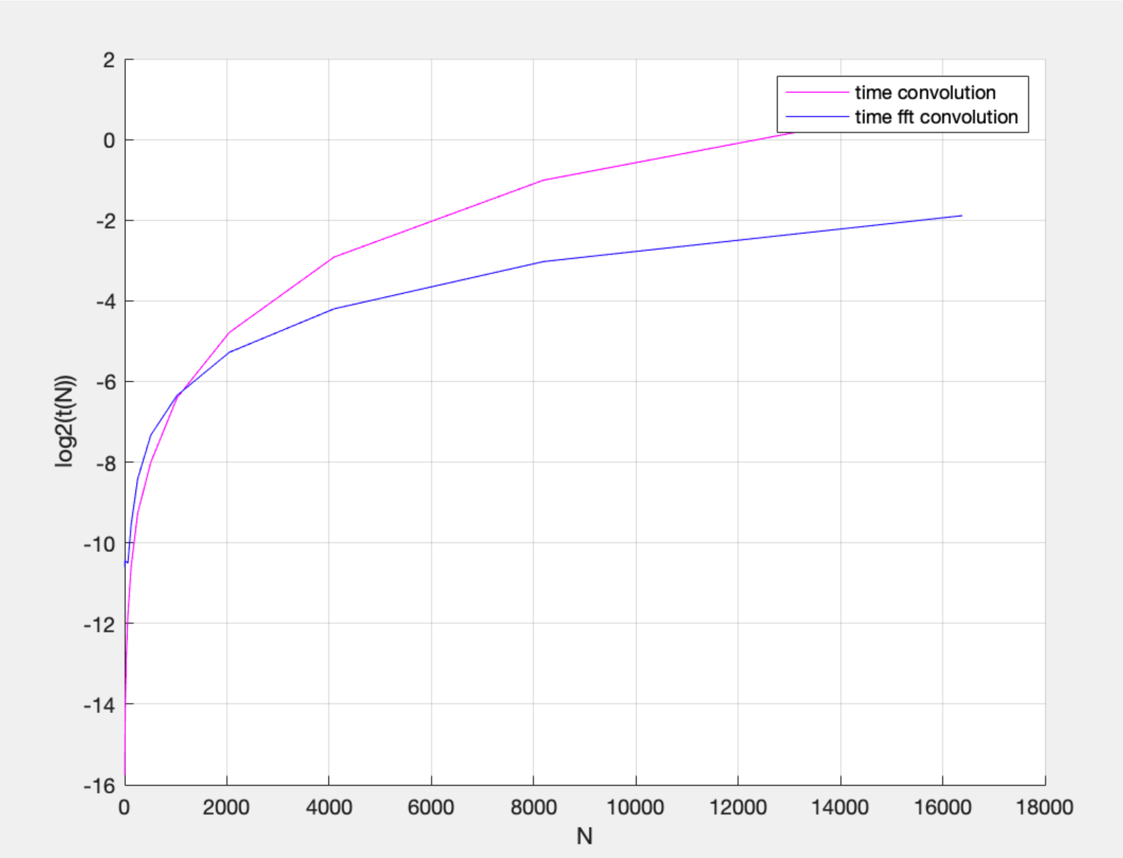
b) С фиксированной длиной одного из векторов

Зафиксируем длину одной из последовательностей $2^6 = 64$ и проведем аналогичные измерения.

```
convol_fix     = [1.8e-05, 3.4e-05, 6.6e-05, 0.000133, 0.000283, 0.000648, 0.001627,
0.003891, 0.011765, 0.036265, 0.132484, 0.496657, 1.94634];
fft_convol_fix = [0.000646, 0.000731, 0.000702, 0.000709, 0.000686, 0.001334, 0.002931,
0.006212, 0.012286, 0.025789, 0.054451, 0.122614, 0.269964];

figure
hold on; grid on;
plot(N, convol_fix, 'm')
plot(N, fft_convol_fix, 'b')
xlabel('N'); ylabel('t(N)'); legend('time convolution', 'time fft convolution')

figure
hold on; grid on;
plot(N, log2(convol_fix), 'm')
plot(N, log2(fft_convol_fix), 'b')
xlabel('N'); ylabel('log2(t(N))'); legend('time convolution', 'time fft convolution')
```

Получаем результат:





Отсюда можем выдвинуть гипотезу о том, что при фиксации длины одной из послдедовательностей скорость вычитсления свертки аналитически (непосредственно) не меняется, а вот fft-свертка оценивает свою эффективность вычисления длиной для наибольшего массива. Видно, чтл до тех пор, пока размер меньшего массива не дойдет до размера фиксированного время вычисления примерно константно и равно времени вычисления сверстки для 2х массивов одного размера с фиксированным, после превышения этой величины скорость вычисления fft-свертки проволжает оцениваться величиной для наибольшего вектора и начинает рости.

Каталог файлов

В основной папке `Lab2` находятся файлы MATLAB:

- `MainLab2.m` для генерации и сохранения векторов, а также вычислкения БПФ и свертки MATLAB
- `result_visualization.m` для визуализации результатов скорости работы алгоритмов в зависимости от размера векторов.

Основыне C++ файлы:

- `DFT.cpp` с реализацией вычисления и анализа ДПФ
- `FFT.cpp` с реализацией вычисления и анализа БПФ
- `convolution.cpp` с реализациями вычисления и анализа сверток

Папка data содержащая все сохраняемые текстовые файлы

- `time_measure.txt` хранящий результаты эффективности работы алгоритмов БПФ, ДПФ и сверток.
- все остальное, генерируемое MATLAB для обмена файлами C++ и MatLab программ.

Вывод

Можно заключить, что использлвание быстрого преобразования Фурье существенным образом виляет на скорости вычисления как самого преобразования Фурье, так и свертки на его основе для болших размерностей векторов.