Algorytmy 2
Laboratorium: szybka transformacja Fouriera (FFT)

Przemysław Klęsk

21 stycznia 2021

1 Cel

Transformacja (przekształcenie) Fouriera odgrywa ważną rolę w informatyce. Praktyczne zastosowania tej transformacji obejmują m.in.: przetwarzanie sygnałów i obrazów, analizę widma, filtrowanie, aproksymację, kompresję danych audio–wideo, szybkie mnożenie dużych liczb całkowitych.

Przedmiotem zadania jest dyskretna transformacja Fouriera (DFT, ang. *Discrete Fourier Transform*) obliczana na dwa sposoby: sposobem bezpośrednim (według definicji) oraz sposobem szybkim (FFT, ang. *Fast Fourier Transform*). Zasadniczym celem zadania jest wykonanie implementacji FFT według klasycznego algorytmu Cooleya–Tukeya opartego na podejściu „dziel i zwyciężaj", w którym liczba próbek sygnału (N) jest pewną potęgą dwójki (radix-2 FFT). Algorytm ten składa wynikową transformatę na podstawie dwóch transformat FFT wykonanych osobno dla parzystych i nieparzystych próbek sygnału (lub funkcji). Takie podejście prowadzi do złożoności obliczeniowej $\Theta(N \log N)$, podczas gdy obliczenia realizowane według definicji mają złożoność $\Theta(N^2)$.

Dla przypomnienia: jeżeli $f_0, f_1, \ldots, f_{N-1}$ oznacza zestaw próbek pewnego sygnału (lub funkcji), to zgodnie z definicją — *transformatą*, czyli wynikiem dyskretnej transformacji Fouriera, jest zestaw N współczynników zespolonych określonych wzorem

$$c_k = \sum_{n=0}^{N-1} f_n e^{-2\pi i k n/N}, \quad k = 0, 1, \ldots, N-1, \tag{1}$$

gdzie i oznacza jednostkę urojoną ($i^2 = -1$).

2 Instrukcje, wskazówki, podpowiedzi

1. W realizowanym programie należy wykorzystać typ reprezentujący **liczby zespolone**, w którym części rzeczywista i urojona są liczbami zmiennoprzecinkowymi w precyzji 64-bitowej — `std::complex<double>`. W tym celu należy wykonać dołączenie: `#include <complex>`. Liczby zespolone można wówczas powoływać do życia na kilka sposobów, na przykład:

```
std::complex<double> z1 = 2.0 + 3.0i;
std::complex<double> z2(2.0, 3.0);
```

2. Niektóre przydatne funkcje związane z liczbami zespolonymi to: `real(...)` (oddaje część rzeczywistą z liczby zespolonej), `imag(...)` (część urojona), `abs(...)` (moduł), `conj(...)` (sprzężenie zespolone), `exp(...)` (eksponenta zespolona).

3. Przydatne może być także umieszczenie w nagłówku programu linii:
```
#define _USE_MATH_DEFINES
#include "math.h"
```
w celu dołączenia biblioteki matematycznej oraz stałych, m.in. `M_PI` reprezentującej pewne przybliżenie liczby π.

4. Drugą ewentualnością jest samodzielne zdefiniowanie w programie stałych dla π oraz i (zamiast używania literału `i`), np. w taki sposób:
```
const double pi = std::acos(-1);
const std::complex<double> i(0, 1);
```

5. Należy dostarczyć implementacje dwóch funkcji **DFT** i **FFT** wyznaczających transformatę Fouriera odpowiednio wg definicji i w sposób szybki (argumenty: tablica liczb f_n, jej rozmiar N; wynik: tablica współczynników zespolonych c_k; gdzie k, $n = 0, 1, \ldots, N-1$).

6. **Uwaga** Realizacja funkcji FFT nie odbywa się *w miejscu*. W związku z tym należy w ramach rekurencji zadbać o odpowiednie powoływanie i uwalnianie pamięci, zarówno dla wydzielanych próbek parzystych / nieparzystych, jak i dla chwilowych rezultatów cząstkowych. Nie wolno dopuścić do wycieków pamięci.

7. Poprawność odpowiedzi zwracanych przez przygotowane funkcje (DFT i FFT) należy sprawdzić z istniejącymi gotowymi funkcjami w dowolnej bibliotece numerycznej (np. MATLAB lub `numpy` języka Python).

8. Nie jest wymagane przygotowanie implementacji transformacji odwrotnej (IDFT).

9. W celu weryfikacji zgodności współczynników zespolonych zwracanych przez DFT i FFT należy przygotować funkcję obliczającą błąd pomiędzy tymi wynikami (średni moduł różnic). Np. niech c oraz c' oznaczają wyniki zwracane odpowiednio przez DFT i FFT, wówczas błąd powinien być obliczony jako: $1/N \sum_{k=0}^{N-1} |c_k - c'_k|$, gdzie $|\cdot|$ oznacza moduł z liczby zespolonej.

10. W końcowym programie należy przewidzieć możliwość wypisania na ekran otrzymanych współczynników (np. z pomocą przełącznika typu `bool`).

3 Zawartość funkcji `main()`

Główny eksperyment zawarty w funkcji `main()` ma polegać na wielokrotnym wykonywaniu przygotowanych funkcji DFT i FFT dla coraz większych rozmiarów tablicy wejściowej z próbkami sygnału (kolejne

rzędy wielkości o podstawie 2, od 2^1 aż do 2^{13}), wraz z raportowaniem pomiarów czasowych. Dostarczone próbki mogą pochodzić z dowolnej wybranej funkcji, np. liniowej $f_n = n/N$ lub funkcji kwadratowej $f_n = (n/N)^2$, $n = 0, 1, \ldots N-1$ (lub jeszcze innej).

Poniższy listing pokazuje schemat eksperymentu (proszę traktować go jako poglądowy przykład):

```
int main()
{
    const int MAX_ORDER = 13; // maksymalny rzad wielkosci danych (w ramach bazy 2)
    const bool PRINT_COEFS = false; // przelacznik do ewentualnego wypisu na ekran rezultatow DFT, FFT

    for (int o = 1; o <= MAX_ORDER; o++)
    {
        const int N = 1 << o; // rozmiar problemu (potega dwojki - przesuniecie bitowe w lewo)
        printf("N: %i\n", N);

        double* f = new double[N];
        for (int n = 0; n < N; n++)
            f[n] = n / (double)N; // przykladowe dane (tu akurat: probki funkcji liniowej)

        clock_t t1 = clock();
        complex<double>* cDFT = dft(f, N);
        clock_t t2 = clock();
        double dft_time = (t2 - t1) / (double)CLOCKS_PER_SEC * 1000.0;
        printf("DFT time [ms]: %f\n", dft_time);

        t1 = clock();
        complex<double>* cFFT = fft(f, N);
        t2 = clock();
        double fft_time = (t2 - t1) / (double)CLOCKS_PER_SEC * 1000.0;
        printf("FFT time [ms]: %f\n", fft_time);

        printf("mean error: %f\n", err(cDFT, cFFT, N));

        if (PRINT_COEFS)
            for (int k = 0; k < N; k++)
                ... // wypis na ekran wspolczynnikow obu transformat (czesci rzeczywiste i urojone)
        printf("------\n", N);

        delete[] f;
        delete[] cDFT;
        delete[] cFFT;
    }

    return 0;
}
```

4 Sprawdzenie antyplagiatowe — przygotowanie wiadomości e-mail do wysłania

1. Kod źródłowy programu po sprawdzeniu przez prowadzącego zajęcia laboratoryjne musi zostać przesłany na adres *algo2@zut.edu.pl*.

2. Plik z kodem źródłowym musi mieć nazwę wg schematu: `nr_albumu.algo2.nr_lab.main.c` (plik

może mieć rozszerzenie `.c` lub `.cpp`). Przykład: `123456.algo2.lab06.main.c` (szóste zadanie laboratoryjne studenta o numerze albumu 123456). Jeżeli kod źródłowy programu składa się z wielu plików, to należy stworzyć jeden plik, umieszczając w nim kody wszystkich plików składowych.

3. Plik musi zostać wysłany z poczty ZUT (zut.edu.pl).

4. Temat maila musi mieć postać: `ALGO2 IS1 XXXY LAB06`, gdzie `XXXY` to numer grupy (np. `ALGO2 IS1 210C LAB06`).

5. W pierwszych trzech liniach pliku z kodem źródłowym w komentarzach muszą znaleźć się:

   - informacja identyczna z zamieszczoną w temacie maila (linia 1),
   - imię i nazwisko autora (linia 2),
   - adres e-mail (linia 3).

6. Mail nie może zawierać żadnej treści (tylko załącznik).

7. W razie wykrycia plagiatu, wszytkie uwikłane osoby otrzymają za dane zadanie ocenę 0 punktów (co jest gorsze niż ocena 2 w skali $\{2, 3, 3.5, 4, 4.5, 5\}$).