| 04 | **Cyfrowe Przetwarzanie Sygnałów**  FFT – szybka transformata Fouriera prof. dr hab. inż. Tomasz Zieliński, dr inż. Jarosław Bułat | 2019.04.01 |
| --- | --- | --- |

# 1a. FFT składane od drugiego poziomu „motylków” (2pkt)

Wykonaj zadanie 1a jeżeli Twój numer indeksu jest „parzysty”, w przeciwnym wypadku wybierz 1b.

Złożoność obliczeniowa *N*-punktowej transformaty DFT określonej zależnością:

, , (1)

to *O*(*N2*). Jedno zespolone *N*-punktowe DFT można wykonać w następujący sposób:

(2)

, (3)

(4)

, . (5)

Zauważ, że obliczenia w nawiasach z (5) to transformata DFT (1) o długości *N*/2 więc zależność można zapisać:

,   
 , (6)

Złożoność obliczeniowa (1) to *O*(*Np*) natomiast złożoność (6) jest mniejsza i wynosi *2\*O*((*N*/2)*2*). Dokładny opis znajdziesz w TZ, podrozdział 9.5.1, równania (9.35)-(9.40).



I teraz wszystko staje się jasne ;-) Schemat przedstawiony od (2) do (6) można zastosować do (6) i dalej, głębiej, aż do momentu, gdy wektor wejściowy **x** będzie składał się tylko dwóch próbek.

Wygeneruj sygnał **x**, losowy, o długości 1024 próbek. Oblicz **X** za pomocą funkcji DFT(...). Następnie wyznacz **Xfft** za pomocą (6): **Xfft=X1+cX2**, dodając do siebie osobno obliczone widma DFT(...) próbek parzystych i nieparzystych (te drugie z korektą **c**). Następnie widma **X1** oraz **X2** wyznacz ponownie za pomocą (2): **X1=X11+cX12**, **X2=X21 +cX22** (czyli podziel próbki o numerach parzystych na te o numerach parzystych i nieparzystych, podobnie zrób z próbkami o numerach nieparzystych). Zauważ, że **Xfft** oraz **X** ma długość 1024, **X1** i **X2** to wektory o długości 512 natomiast **X11**, **X12** , **X21**, **X22**  mają długość 256 próbek.

Porównaj czy wynik uzyskany we wszystkich 3 sposobach jest taki sam.

# 1b. FFT za pomocą rekurencji (2 pkt)

Złożoność obliczeniowa transformacji DFT *N*-punktowej to *O*(*Np*). Jedno zespolone *N*-punktowe DFT można wykonać jako złożenie dwóch *N*/2-punktowych DFT (plus pewne obliczenia związane ze ,,składaniem'') co skutkuje obniżeniem złożoności do 2\**O*((*N*/2)2). Następnie można rozbić obliczenia na cztery *N*/4-punktowe DFT. Dla *N =*2*p* można zejść w ten sposób do wielu DFT o dłguości *N =*2.

Poniższa funkcja realizuje zespoloną transformację Fouriera poprzez podział w dziedzinie czasu DIT (ang. *Decimation i Time*).

function X = dit( x )

N = length(x);

x = x(:); % macierz wertykalna

X1 = fft( x(1:2:N) ); % próbki parzyste

X2 = fft( x(2:2:N) ); % próbki nieparzyste

X = zeros( size(x) );

k = (0:N/2-1)';

X(1:N/2) = X1 + exp( j\*2\*pi/N .\*-k ) .\* X2;

X(N/2+1:N) = X1 + exp( j\*2\*pi/N .\* -(k+N/2) ) .\* X2;

Działanie funkcji można zweryfikować programem (powyższą funkcje zapisz do pliku dit.m):

clear all;

close all;

N = 256;

x = randn( N, 1 );

X1 = fft(x); % oryginalne DFT

X2 = dit(x); % DFT ,,sklejane'' z dwóch połówek

mean( abs(X1-X2) ) % błąd

Funkcja dit(...) wykonuje tylko pierwszy z *p* etapów podziału. Każdy następny etap powinien być wykonany na zmiennej X1 i X2. Wykorzystując dit(...) zaimplementuj algorytm radix-2 DIT FFT dla długości *N =*2*p*. Wykorzystaj rekurencję.

# 2. Transformata Fouriera sygnałów rzeczywistych (3 pkt)

Transformata DFT jest zespolona, jednak często transformacji poddaje się sygnały rzeczywiste np.: dźwięk, zdjęcia cyfrowe, etc... Dlatego, aby jeszcze bardziej przyspieszyć działanie tego algorytmu można wykorzystać symetrię widma i:

1. wykonać dwie N-puntkowe transformacje rzeczywistego sygnału za pomocą jednej N-punktowej transformaty zespolonej lub
2. wykonać jedną N-punktową transformacje rzeczywistego sygnału za pomocą N/2-punktowej transformaty zespolonej.

Jeżeli przedostatnia cyfra numeru Twojej legitymacji studenckiej jest liczbą nieparzystą wykonaj pierwszy punkt, jeżeli jest parzysta wybierz punkt drugi. Przyjmij N=1024, wygeneruj losowe dane o rozkładzie normalnym, wykonaj transformatę rzeczywistą a następnie porównaj otrzymane wyniki do transformaty zespolonej. Jakie przyspieszenie algorytmu uzyskałeś tą metodą?

Dodatkowe przyspieszenie transformcji Fouriera sygnałów rzeczywistych wykorzystuje właściwość symetrii widma: dla rzeczywistego sygnału *x*(*n*), *n=*0,1,2,…,*N-*1, jego transformata *X*(*k*), k*=*0,1,2,…,*N-*1 ma następującą właściwość:

czyli: Re{*X*(*k*)} *=* Re{*X*(*N-k*)}*,* Im{*X*(*k*))*=-*Im(*X*(*N-k*))*, k=*1,2,...*,N-*1*.*

Dlatego, dla dwóch niezależnych sygnałów (**punkt 1**) *x1*(*n*) i *x2*(*n*)*, n=*0,1,2,...,*N-*1*,* można uzyskać ich transformaty Fouriera w następujący sposób:

*y*(*n*)*=x1*(*n*)*+jx2*(*n*)

**Y=**fft(**y**)

wykorzystując symetrię widma względem *k=N*/2 można ,,odzyskać'' transformaty Fouriera sygnałów **x1** i **x2** w następujący sposób:

*X1r*(*k*) *=* 0.5\*(*Yr*(*k*)*+Yr*(*N-k*))*,* *k=*1,2,3*,...,N-*1, *X1i*(*k*) *=* 0.5\*(*Yi*(*k*)*-Yi*(*N-k*))*,* *k=*1,2,3*,...,N-*1

*X2r*(*k*) *=* 0.5\*(*Yi*(*k*)*+Yi*(*N-k*))*,* *k=*1,2,3*,...,N-*1, *X2i*(*k*) *=* 0.5\*(*Yr*(*N-k*)*-Yr*(*k*))*,* *k=*1,2,3*,...,N-*1

gdzie **Yr** i **Yi** to odpowiednio część rzeczywista i urojona wektora **Y,** podobnie jak **X1r**, **X1i**, **X2r** i **X2i** są odpowiednio częścią rzeczywistą i urojoną wektoów **X1** i **X2**. Zerowe prążki widma odzyskuje się następująco:

*X1r*(0)=*Yr*(0), *X1i*(0)=0

*X2r*(0)=*Yi*(0), *X2i*(0)=0

Ostatecznie, transformaty Fouriera wektorów **x1** i **x2** uzyskuje się łącząc wektory:

**X1***=***X1r***+j***X1i**

**X2***=***X2r***+j***X2i**

Wersja alternatywna (**punkt 2**) polega na wyznaczeniu N-punktowego sygnału rzeczywistego za pomocą pojedynczej N/2-punktowej FFT:

niech *y*(*n*)*=x*(2*n*)*+jx*(2*n+*1)*, n=*0,1,2,...,*N*/2-1 a **Y** to transformata Fouriera **y** o dlugości *N*/2. Wtedy operację **X***=*fft(**x**) można otrzymać w dwóch krokach:

1) , k=1,2,3,...,N/2-1.

Widmo dla *k=N*/2*,N*/2*+*1*,...,N-*1 jest sprzężone, symetryczne do pierwszej połówki. Załóż, że *Y*(*513*)*=Re*(*Y*(*0*))*-Im*(*Y*(*0*)).

2)

# 3. Implementacja algorytmu radix-2 (opcja, +1 pkt)

Wygeneruj w Matlabie zespolony wektor **x** o rozmiarze 1x1024, charakterze losowym (rozkład normalny) oraz nieskorelowanej części rzeczywistej i urojonej. Zapisz go w formacie Matlaba do pliku x.mat oraz w formacie w który odczytasz w języku C/C++ (plik xcpp.dat).

Napisz program (w języku Matlab) o nazwie myFFT, który wczyta x.mat oraz xcpp.dat wykona transformatę Fouriera obu sygnałów uzyskując **X1** oraz **X2**. Porównaj oba wyniki. Jeżeli są istotnie różne to oznacza, że utraciłeś część informacji podczas zapisu do pliku xcpp.dat – skoryguj ten błąd.

Napisz program w języku C/C++ implementujący szybką transformatę Fouriera (FFT) za pomocą algorytmu *radix-2* w dwóch wersjach – na zmiennych typu float oraz double. Następnie wczytaj sygnał **x** z pliku xcpp.dat, wykonaj transformację sygnału w precyzji float oraz double, zapisz wyniki w oddzielnych plikach, w formacie który będziesz mógł odczytać w środowisku Matlab.

W języku Matlab, wczytaj transformacje wykonane w języku C/C++ i porównaj wynik do wzorcowej implementacji fransformaty Fouriera wykonanej w języku Matlab (wektor **X1**).

Wykorzystaj opis algorytmu *radix-2* z wykładu.

# 4. Porównanie implementacji C/C++ z biblioteką fftw (opcja, +0.5 pkt)

Wykorzystując dane z punktu 3 wykonaj dodatkowo transformację FFT za pomocą biblioteki *fftw* ([www.fftw.org](http://www.fftw.org/)). Sprawdź poprawność wyników w wersji float i double. Załóż algorytm Matlaba jako wzorcowy (pamiętaj o potencjalnych różnicach w skalowaniu wyniku transformaty!).

Oblicz czas wykonywania Twojej implementacji i implementacji z biblioteki *fftw*. Do obliczeń użyj funkcji gettimeofday(...) pod systemem Linux. Dyskusja na temat wyznaczania czasu krótkich procedur oraz optymalizacji kodu zawarta została w konspekcie do laboratorium 02 przedmiotu TowNiT. Do czasu wykonania procedury FFT z biblioteki *fftw* nie wliczaj tzw. „planu”.