Szybka Transformacja Fouriera w jednym i dwóch wymiarach

Vittoria Condorelli

Wydział Matematyki i Informatyki
Uniwersytet Jagielloński

Kraków
23 czerwca 2023

Streszczenie

Tematem pracy jest dyskretna transformata Fouriera (DFT), ze szczególnym uwzględnieniem optymalnych metod obliczania jej. W początkowych rozdziałach zawarta jest motywacja, jak i potrzebne definicje i dowody związane z dyskretną transformatą Fouriera w jednym wymiarze. Następnie zaprezentuję algorytm pozwalający na efektywne obliczanie DFT - Szybką Transformację Fouriera (FFT) w wersji Cooleya-Tukeya. W następnych rozdziałach podobnej analizie poddany jest przypadek dwuwymiarowy. Zaproponowane są tam dwa algorytmy - użycie jednowymiarowej FFT do przypadku dwuwymiarowego i uogólnienie algorytmu Cooleya-Tukeya do dwóch wymiarów. W dalszej części znajduje się opis przykładowych zastosowań FFT. Dla funkcji jednowymiarowych jest to szybkie mnożenie wielomianów oraz dużych liczb całkowitych, w przypadku dwuwymiarowym - kompresji obrazów. Praca jest zilustrowana dołączonym plikiem `fast_fourier_transform.ipynb` zawierającym implementację i porównanie czasów wykonania zaprezentowanych metod i wizualizację ich wyników wykresami.

Spis treści

1 Wprowadzenie

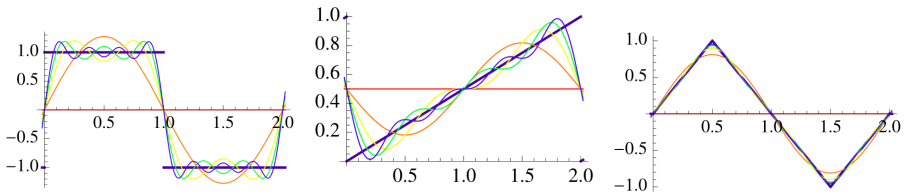
Szereg Fouriera to sposób przedstawienia funkcji okresowej jako nieskończonej sumy funkcji trygonometrycznych o różnych częstotliwościach. Dla funkcji f o okresie P jest on dany wzorem:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos(\frac{2\pi nx}{P}) + b_n \sin(\frac{2\pi nx}{P})) \quad \text{dla}$$
$$a_n = \frac{2}{P}\int_{-P/2}^{P/2} s(x)\cos(\frac{2\pi nx}{P})dx$$
$$b_n = \frac{2}{P}\int_{-P/2}^{P/2} s(x)\sin(\frac{2\pi nx}{P})dx$$

Lub alternatywnie, w formie eksponencjonalnej:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{2\pi inx}{P}} \quad \text{dla} \quad c_n = \frac{1}{P}\int_{-P/2}^{P/2} f(x)e^{-\frac{2\pi inx}{P}}dx$$

Motywacją wprowadzenia takiej reprezentacji przez Josepha Fouriera było rozwiązanie problemu z dziedziny równań różniczkowych cząstkowych - równania przewodnictwa cieplnego. Szereg Fouriera dał możliwość korzystania z dobrze zbadanych własności funkcji trygonometrycznych przy działaniach na funkcjach, które na pierwszy rzut oka nie mają z nimi nic wspólnego. Dzięki temu powstała nowa dziedzina analizy matematycznej - analiza harmoniczna, inaczej nazywana analizą Fouriera.



(a) (b) (c)

Rysunek 1: Wykresy kolejnych sum częściowych szeregu Fouriera przykładowych funkcji. [Wei]

W praktycznych zastosowaniach często spotykamy się z próbkowaniem funkcji. Może to wystąpić na przykład podczas dyskretnego odczytu sygnału lub w metodach numerycznych. Wtedy wszystkie nasze informacje na temat funkcji są przedstawione jako ciąg jej kolejnych wartości w punktach próbkowania. W takim przypadku, aby analogicznie do szeregu Fouriera przybliżać funkcję za pomocą funkcji trygonometrycznych możemy użyć dyskretnej transformaty Fouriera opisanej w następnym rozdziale.

2 Dyskretna Tranformata Fouriera

Definicja 1 (Dyskretna Transformata Fouriera)**.**
Dla ciągu $(a_0, a_1, \ldots, a_{N-1})$ *takiego, że* $, a_k = f(x_k)$
gdzie f *jest funkcją* $f : \mathbb{R} \to \mathbb{C}$
i gdzie $x_k = \frac{2\pi}{N}k$ *(jest to* N *punktów rozłożonych w równych odległościach na odcinku* $[0, 2\pi)$*)*

Dyskretną transformatą Fouriera ciągu $(a_0, a_1, \ldots, a_{N-1})$ *nazywamy ciąg* $(A_0, A_1, \ldots, A_{N-1}), A_i \in \mathbb{C}$ *postaci:*

$$A_k = \sum_{n=0}^{N-1} a_n e^{-ix_k n}$$

gdzie: $0 \leqslant k < N$

Wygodnym w niektórych przypadkach zapisem powyższego wzoru jest postać, gdzie użyty jest N-ty pierwiastek prymitywny jedynki:

$$\omega_N = e^{i\frac{2\pi}{N}}$$

Wtedy:

$$A_k = \sum_{n=0}^{N-1} a_n e^{-i\frac{2\pi}{N}kn} = \sum_{n=0}^{N-1} a_n \omega_N^{-kn}$$

Twierdzenie 1 (Odwzorowanie odwrotne)**.**
Przekształcenie odwrotne dla dyskretnej transformaty Fouriera dane jest wzorem:

$$a_n = f(x_n) = \frac{1}{N} \sum_{k=0}^{N-1} A_k e^{ix_n k}$$

gdzie: $0 \leqslant n \leqslant N-1$
Alternatywny zapis:

$$a_n = \frac{1}{N} \sum_{k=0}^{N-1} A_k \omega_N^{kn}$$

Dowód.

Niech macierz D będzie postaci:

$$D = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \ldots & \omega^0 \\ \omega^0 & \omega^{-1} & \omega^{-2} & \ldots & \omega^{-(N-1)} \\ \omega^0 & \omega^{-2} & \omega^{-4} & \ldots & \omega^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^0 & \omega^{-(N-1)} & \omega^{-2(N-1)} & \ldots & \omega^{-(N-1)^2} \end{bmatrix}$$

gdzie $\omega = e^{i\frac{2\pi}{N}}$.

Wtedy dyskretna transformata Fouriera $A = (A_0, A_1, \ldots, A_{N-1})$ ciągu $a = (a_0, a_1, \ldots, a_{N-1})$ jest dana wzorem: $A = Da$. Wtedy $a = D^{-1}A$.
Jeśli wzór z Twierdzenia 1 jest poprawny, zachodzi $a = \frac{1}{N}\overline{D}A$, gdzie $\overline{D}$ to macierz D po wykonianiu sprzężenia na każdym elemencie.
Aby pokazać prawdziwość twierdzenia, musimy więc pokazać, że $D^{-1} = \frac{1}{N}\overline{D}$, czyli $\overline{D}D = IN$, gdzie I to macierz identycznościowa.

$$(\overline{D}D)_{kl} = \sum_{n=0}^{N-1} \omega^{kn}\omega^{-ln} = \sum_{n=0}^{N-1} \omega^{n(k-l)} \quad \text{dla } k, l = 0, \ldots, N-1$$

dla $k = l$:

$$(\overline{D}D)_{kl} = \sum_{n=0}^{N-1} \omega^0 = \sum_{n=0}^{N-1} 1 = N$$

dla $k \neq l$:

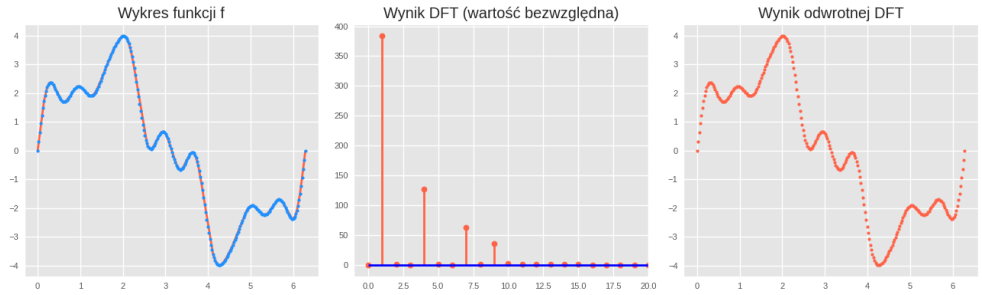
$$(\overline{D}D)_{kl} = \sum_{n=0}^{N-1} \omega^{n(k-l)} = \omega^0 \cdot \frac{1 - \overbrace{\omega^{(k-l)N}}^{=1}}{1 - \underbrace{\omega^{(k-l)}}_{\neq 1}} = 0$$

Jak widać z powyższych równości $\overline{D}D = IN$. □

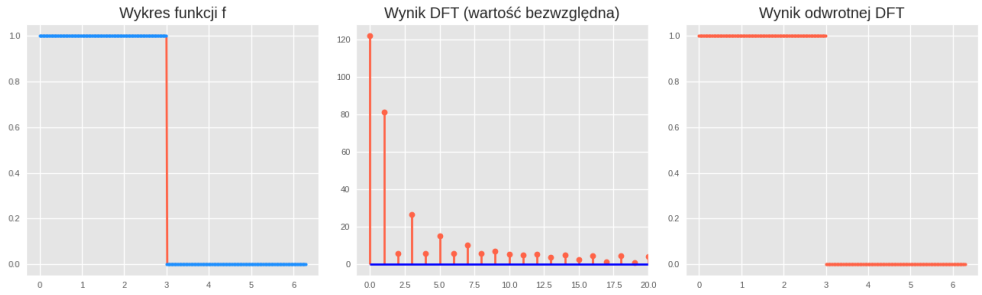
Z wzoru z Twierdzenia 1 widać, że dyskretna transformata Fouriera daje nam współczynniki dla przybliżenia funkcji f za pomocą funkcji eksponencjonalnych - interpolację trygonometryczną. Ze względu na wzór Eulera: $e^{i\alpha} = \cos\alpha + i\sin\alpha$ możemy na nią patrzeć jako na przybliżanie funkcji sumą sinusoid o różnych częstotliwościach. Wtedy dla zespolonego współczynnika A_k przy jego postaci wykładniczej $A_k := re^{i\phi}$ możemy następująco przekształcić fragment wzoru z Twierdzenia 1:

$$A_k e^{ix_n k} = r \cdot e^{i\phi} e^{ix_n k} = re^{i(\phi + x_n k)} = rcos(\phi + x_n k) + irsin(\phi + x_n k)$$

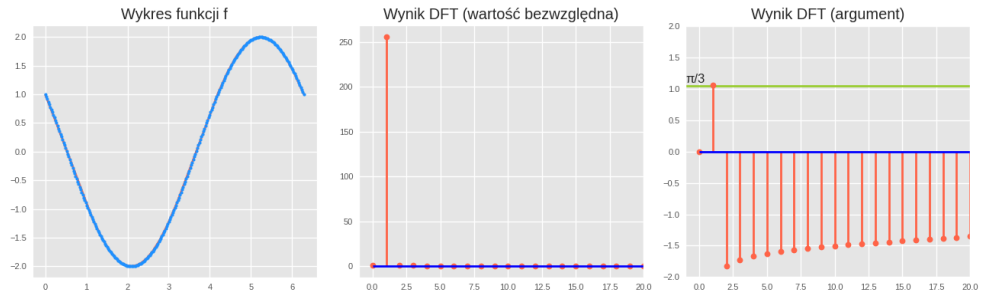
Pokazuje to, że r (wartość bezwzględna A_k) ma wpływ na wysokość w najwyższym punkcie (amplitudę) sinusoidy o częstotliwości k (liczbie powtórzeń w przedziale $[0, 2\pi]$), a ϕ (argument A_k) na jej przesunięcie.



Rysunek 2: Od lewej: wykres funkcji i jej próbek, wartości DFT i wynik odwzorowania odwrotnego dla funkcji $f(x) = 3\sin x + \sin(4x) + 0.5\sin(7x) + 0.3\sin(9x)$. Na wykresie widoczne jest, że wartości bezwzględne dla poszczególnych częstotliwości odpowiadają amplitudom składowych sinusoid (mnożnikom przy sinusach).



Rysunek 3: Analogiczne wykresy dla funkcji schodkowej.



Rysunek 4: Wykresy funkcji, wartości bezwzględnej i argumentu wyniku DFT dla funkcji $f(x) = 2\cos(x + \frac{\pi}{3})$. Widać, że jedyną częstotliwością dla której wartość bezwzględna nie jest bliska 0 jest 1, a jej argument odpowiada przesunięciu i jest równy $\frac{\pi}{3}$.

3 Szybka Transformacja Fouriera

Aby policzyć dyskretną transformatę Fouriera ciągu $(a_0, a_1, \ldots, a_{N-1})$ z definicji, potrzebujemy wykonać $\mathcal{O}(N)$ operacji arytmetycznych podczas obliczania każdego z wyrazów A_k. Obliczenie wartości całego ciągu $A_1, A_2, \ldots, A_{N-1}$ wymaga więc złożoności rzędu $\mathcal{O}(N^2)$.

Jesteśmy w stanie uzyskać wynik dyskretnej transformaty Fouriera w czasie $\mathcal{O}(n \log n)$ przy użyciu algorytmu Szybkiej Transformacji Fouriera (Fast Fourier Transform, w skrócie FFT). Najbardziej popularnym algorytmem FFT jest algorytm Cooleya-Tukeya. Jest on rekurencyjny, w każdym kroku rozbija DFT ciągu długości $N = N_1 N_2$ na sumę N_1 DFT krótszych ciągów długości N_2. Najpierw omówię przypadek, w którym postawa (po angielsku radix) N_1 jest równa 2. Przypadek ten nazywany jest Radix-2 DIT (Decimation in Time).

3.1 Algorytm Cooleya-Tukeya o podstawie 2

Dla uproszczenia załóżmy, że N jest potęgą liczby 2. Poniżej pokażę dwie własności, które pozwolą sformułować algorytm.

Lemat 1.
Niech $A_0, \ldots, A_{N-1}$ będzie wynikiem DFT dla ciągu $(a_0, \ldots, a_{N-1})$, gdzie $a_k = f(x_k)$, $x_k = \frac{2\pi}{N}k$.

Niech E_k i O_k będą k-tymi wyrazami wyniku DFT dla ciągu elementów z $(a_0, \ldots, a_{N-1})$ o kolejno parzystych i nieparzystych indeksach. Wtedy zachodzi:

$$A_k = E_k + e^{-i\frac{2\pi}{N}k} O_k \quad \forall_{k \in \{1,\ldots,\frac{N}{2}\}}$$

Dowód.

$$A_k = \sum_{n=0}^{N-1} a_n e^{-i\frac{2\pi}{N}kn} = \left(\sum_{n=0}^{N/2-1} a_{2n} e^{-i\frac{2\pi}{N}k(2n)}\right) + \left(\sum_{n=0}^{N/2-1} a_{2n+1} e^{-i\frac{2\pi}{N}k(2n+1)}\right)$$
$$= \underbrace{\sum_{n=0}^{N/2-1} a_{2n} e^{-i\frac{2\pi}{N/2}kn}}_{E_k} + e^{-i\frac{2\pi}{N}k} \underbrace{\sum_{n=0}^{N/2-1} a_{2n+1} e^{-i\frac{2\pi}{N/2}kn}}_{O_k}$$

□

Lemat 2.

$$A_{k+\frac{N}{2}} = E_k - e^{-i\frac{2\pi}{N}k}O_k$$

Dowód.

$$\begin{aligned}
A_{k+\frac{N}{2}} &= \sum_{n=0}^{N/2-1} a_{2n}e^{-i\frac{2\pi}{N/2}(k+\frac{N}{2})n} + e^{-i\frac{2\pi}{N}(k+\frac{N}{2})}\sum_{n=0}^{N/2-1} a_{2n+1}e^{-i\frac{2\pi}{N/2}(k+\frac{N}{2})n} \\
&= \sum_{n=0}^{N/2-1} a_{2n}e^{-i\frac{2\pi}{N/2}k}e^{-i2\pi n} + e^{-i\frac{2\pi}{N}k}e^{i\pi}\sum_{n=0}^{N/2-1} a_{2n+1}e^{-i\frac{2\pi}{N/2}k}e^{-i2\pi n} \\
&= \sum_{n=0}^{N/2-1} a_{2n}e^{-i\frac{2\pi}{N/2}k} - e^{-i\frac{2\pi}{N}k}\sum_{n=0}^{N/2-1} a_{2n+1}e^{-i\frac{2\pi}{N/2}k} = E_k - e^{i\frac{2\pi}{N}k}O_k \\
&\text{bo} \quad e^{-i2\pi n} = 1, e^{i\pi} = -1
\end{aligned}$$

□

Na podstawie lematów 1 i 2 widać, że dyskretną transformatę Fouriera ciągu o długości N możemy uzyskać przez obliczenie DFT dwóch ciągów długości $\frac{N}{2}$. Poniżej przedstawiam algorytm:

Algorytm 1 Cooleya-Tukeya, Radix 2-DIT

```
function FFT(a)
    even ← elementy tablicy a o parzystych indeksach
    odd ← elementy tablicy a o nieparzystych indeksach
    E ← FFT(even)
    O ← FFT(odd)
    A ← pusta tablica wielkości tej samej co a
    for i = 0, ..., N/2 do
        A[k] = E[k] + e^{-i 2π/N k} O[k]
        A[k + N/2] = E[k] - e^{-i 2π/N k} O[k]
    end for
    return A
end function
```

Twierdzenie 2.
Zaproponowany Algorytm 1 obliczania dyskretnej transformaty Fouriera ma złożoność $\mathcal{O}(N \log N)$.

Dowód.

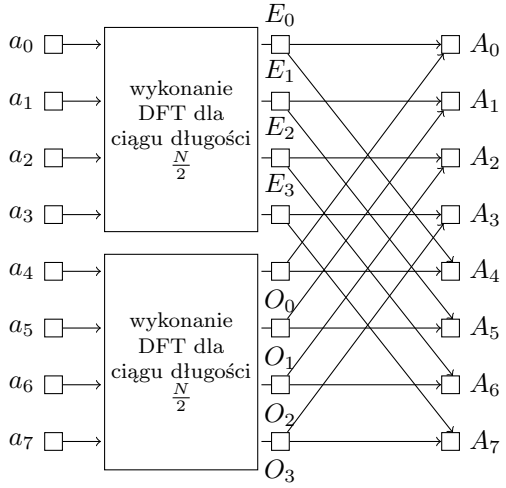
Połączenie wyników dwóch mniejszych DFT, aby uzyskać każdy z elementów DFT długości N można wykonać w czasie stałym, więc obliczenie całego ciągu wymaga złożoności $\mathcal{O}(N)$. Poniżej pokażę, że złożoność całego algorytmu FFT wynosi $\mathcal{O}(N \log N)$.

Niech $N = 2^k$ będzie długością ciągu.
Oznaczmy przez $T(N)$ złożoność algorytmu FFT dla ciągu długości N.

$$T(N) = 2T\left(\frac{N}{2}\right) + \mathcal{O}(N) \leqslant 2T\left(\frac{N}{2}\right) + CN \leqslant$$
$$2 \cdot \left(2T\left(\frac{N}{4}\right) + C \cdot \frac{N}{2}\right) + CN = 4T\left(\frac{N}{4}\right) + 2CN \leqslant$$
$$4 \cdot \left(2T\left(\frac{N}{8}\right) + C \cdot \frac{N}{4}\right) + 2CN = 8T\left(\frac{N}{8}\right) + 3CN \leqslant$$
$$\vdots$$
$$\leqslant 2^k T(1) + kCN = NA + CN \log_2 N = \mathcal{O}(N \log N)$$

gdzie C, A to stałe □

To w których miejscach używamy których wartości wyników DFT krótszych ciągów, aby uzyskać DFT dłuższego ciągu w algorytmach szybkiej transformacji Fouriera można przedstawić za pomocą diagramu. Ze względu na jego kształt nazywa się go motylem (butterfly diagram). Diagram dla algorytmu opisanego wyżej, dla ciągu 8 wartości wygląda następująco:



3.2 Algorytm Cooleya-Tukeya o postawie r

Zakładając, że r jest dzielnikiem N możemy uzyskać analogiczne rozbicie na r DFT wielkości $\frac{N}{r}$ przez wykonanie podstawienia:

$$n := rp + q \qquad \text{dla } p = 0, \ldots, \frac{N}{r} - 1,\ q = 0, \ldots, r-1$$
$$k := u + v\frac{N}{r} \qquad \text{dla } u = 0, \ldots, \frac{N}{r} - 1,\ v = 0, \ldots, r-1$$

Wtedy wzór na dyskretną transformatę Fouriera ma postać:

$$A_{u+v\frac{N}{r}} = \sum_{q=0}^{r-1} \sum_{p=0}^{\frac{N}{r}-1} a_{rp+q} \omega_N^{-(u+v\frac{N}{r})(rp+q)} = \sum_{q=0}^{r-1} \sum_{p=0}^{\frac{N}{r}-1} a_{rp+q} \omega_N^{-urp} \omega_N^{-uq} \omega_N^{-vNp} \omega^{-v\frac{N}{r}q} = (*)$$

a skoro:

$$\omega_N^r = e^{i\frac{2\pi}{N}r} = \omega_{\frac{N}{r}} \quad \omega_N^{\frac{N}{r}} = e^{i\frac{2\pi}{N}\frac{N}{r}} = \omega_r \quad \omega_N^N = 1$$

to:

$$(*) = \sum_{q=0}^{r-1} \underbrace{\left(\sum_{p=0}^{\frac{N}{r}-1} a_{rp+q} \omega_{\frac{N}{r}}^{-up} \right)}_{\text{DFT rozmiaru } r} \omega_N^{-uq} \omega_r^{-vq}$$

Analogicznie do przypadku z postawą 2 wykonujemy r mniejszych DFT na elementach ciągu podzielonych na podzbiory o tej samej reszcie z dzielenia przez r (q w powyższym wzorze). Złożoność takiego algorytmu to również $\mathcal{O}(N \log N)$.

3.3 Obliczanie odwrotnej DFT za pomocą FFT

Ze względu na podobieństwo wzorów dyskretnej transformacji Fouriera i jej odwzorowania odwrotnego naturalne wydaje się użycie algorytmu FFT także do obliczania odwrotnej DFT. Dzięki temu moglibyśmy uzyskać złożoność $\mathcal{O}(N \log N)$ również w tym przypadku.

Zauważmy, że dla liczby zespolonej $x = a + bi$ zachodzi:

$$\text{swap}(x) := i \cdot \overline{x} = i(a - bi) = ia - i^2 b = b + ia$$

Przemnożenie przez i sprzężenia x daje jako wynik liczbę x po zamianie części zespolonej i rzeczywistej. Wtedy też:

$$x = \text{swap}(\text{swap}(x)) = i\overline{(i \cdot \overline{x})}$$

Zastosujmy powyższą równość do wzoru na odwzorowanie odwrotne dyskretnej transformaty Fouriera:

$$a_n = i\overline{(i \cdot \overline{a_n})} = i\overline{\left(i \cdot \overline{\frac{1}{N} \sum_{k=0}^{N-1} A_k e^{ix_n k}} \right)} = i\overline{\left(i \cdot \frac{1}{N} \sum_{k=0}^{N-1} \overline{A_k} e^{-ix_n k} \right)} =$$

$$= \frac{1}{N} \cdot i\overline{\left(\frac{1}{N} \sum_{k=0}^{N-1} (i \cdot \overline{A_k}) e^{-ix_n k} \right)} = \frac{1}{N} \cdot \text{swap}\left(\sum_{k=0}^{N-1} \text{swap}(A_k) e^{-ix_n k} \right)$$

Z przekształconego wzoru powyżej widać, że odwrotną dyskretną transformatę Fouriera można policzyć przez wykonanie następujących kroków:

1. Zamianę części zespolonej i rzeczywistej wyrazów ciągu
2. Policzenie dyskretnej transformaty Fouriera za pomocą algorytmu FFT
3. Ponowną zamianę części zespolonej i rzeczywistej wyniku
4. Przemnożenie wartości przez $\frac{1}{N}$

4 Wielowymiarowa Dyskretna Transformata Fouriera

W poprzednich rozdziałach omówiłam zagadnienie dyskretnej transformaty Fouriera dla ciągu jednowymiarowego. Możliwe jest analogiczne zdefiniowanie jej również dla ciągów wielowymiarowych. Szczególną uwagę skupię na ciągach dwuwymiarowych ze względu na ich zastosowania w przetwarzaniu obrazów. Jako ciąg dwuwymiarowy możemy traktować tablice pikseli reprezentujące obrazy.

Definicja 2 (Wielowymiarowa Dyskretna Transformata Fouriera)**.**
Dla d-wymiarowego ciągu $(a_{n_1,n_2,\ldots,n_d})_{n_j=0,1,\ldots,N_j-1}$*,*
takiego, że $a_{j_1,\ldots,j_d} = f(x_{j_1,\ldots,j_d})$*, gdzie* f *jest funkcją* $f : \mathbb{R}^d \to \mathbb{C}$
i gdzie $x_{j_1,\ldots,j_d} = (x_{1,j_1}, x_{2,j_2}, \ldots, x_{d,j_d})$*, przy* $x_{k,j} = \frac{2\pi}{N_k} j$

Dyskretną transformatą Fouriera ciągu a nazywamy ciąg $(A_{k_1,k_2,\ldots,k_d})_{k_j=0,1,\ldots,N_j-1}$ *postaci:*

$$A_{k_1,\ldots,k_d} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_d=0}^{N_d-1} a_{n_1,\ldots,n_d} e^{-ix_{1,k_1}n_1} \cdots e^{-ix_{d,k_d}n_d}$$

gdzie: $0 \leqslant k_j < N_j$
W zapisie z n-tym pierwiastkiem jedynki:

$$A_{k_1,k_2,\ldots,k_d} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_d=0}^{N_d-1} a_{n_1,n_2,\ldots,n_d} \omega_{N_1}^{-k_1 n_1} \omega_{N_2}^{-k_2 n_2} \cdots \omega_{N_d}^{-k_d n_d}$$

Twierdzenie 3 (Odwzorowanie odwrotne)**.**
Przekształcenie odwrotne dla wielowymiarowej dyskretnej transformaty Fouriera dane jest wzorem:

$$a_{n_1,\ldots,n_d} = f(x_{1,n_1}, \ldots, x_{d,n_d}) = \frac{1}{\prod_{j=0}^{d} N_j} \sum_{k_1=0}^{N_1-1} \cdots \sum_{k_d=0}^{N_d-1} A_{k_1,\ldots,k_d} e^{ix_{1,n_1}k_1} \cdots e^{ix_{d,n_d}k_d}$$

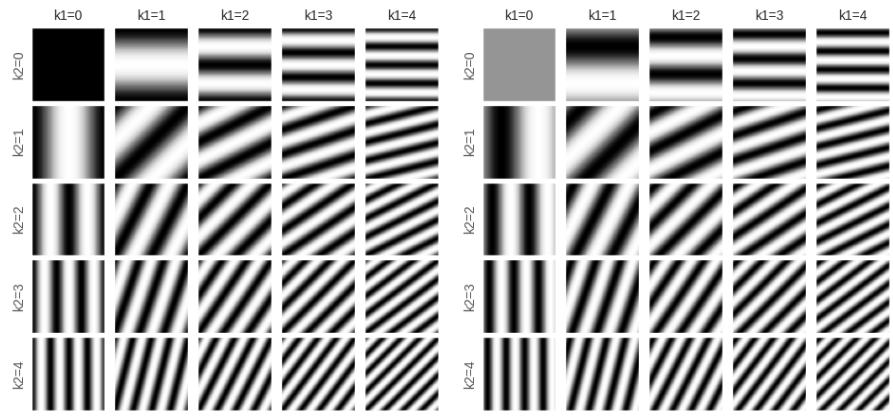
Alternatywny zapis:

$$a_{n_1,\ldots,n_d} = \frac{1}{\prod_{j=0}^{d} N_j} \sum_{k_1=0}^{N_1-1} \cdots \sum_{k_d=0}^{N_d-1} A_{k_1,\ldots,k_d} \omega_{N_1}^{k_1 n_1} \cdots \omega_{N_d}^{k_d n_d}$$

4.1 Intuicja w przypadku dwuwymiarowym

W przypadku jednowymiarowym możemy patrzeć na DFT jako na przedstawienie funkcji jako sumy sinusoid, ponieważ część urojona i rzeczywista wyrażenia e^{ixn} to sinus i cosinus. Analogicznie w dwuwymiarowym przypadku możemy rozważyć jak będą wyglądały funkcje na które DFT rozkłada funkcję wejściową.

Z odwzorowania odwrotnego dwuwymiarowej dyskretnej transformaty Fouriera widać, że w tym przypadku bazą są funkcje postaci $e^{ixk_1}e^{ixk_2}$ dla kolejnych $k_1 = 0, \ldots, N_1, k_2 = 0, \ldots, N_2$. Biorąc ich część rzeczywistą i urojoną możemy zobaczyć, że są to sinusoidalnie wygięte płaszczyzny o różnych częstotliwościach i nachyleniu (po angielsku nazywane *sinusoidal gratings*).



(a) (b)

Rysunek 5: Części rzeczywiste (a) i urojone (b) funkcji postaci $e^{ixk_1}e^{ixk_2}$ dla $k_1, k_2 = 0, \ldots, 4$. Kolor biały oznacza wartość -1, a czarny 1. Funkcje stanowiłyby bazę w dwuwymiarowym DFT wielkości 5×5.

5 Dwuwymiarowe algorytmy FFT

Przez $N := N_1 N_2$ oznaczmy ilość punktów w których liczymy DFT. Obliczenie dwuwymiarowej DFT z definicji wymaga wykonania $\mathcal{O}(N)$ operacji podczas obliczania każdego z wyrazów A_{k_1,k_2}, więc cały algorytm wymaga $\mathcal{O}(N^2)$ operacji. Analogicznie do przypadku jednowymierowego istnieją algorytmy pozwalające na obliczenie dwuwymierowej DFT w czasie $\mathcal{O}(N \log N)$.

5.1 Algorytm "row-column"

Na dwuwymiarową dyskretną transformatę Fouriera możemy patrzeć jak na wykonanie jednowymiarowej DFT na wszystkich kolumnach macierzy wejściowej, a potem ponownie na wszystkich wierszach wyniku. Można to rozpisać w następujący sposób:

$$\mathrm{DFT1}_{k_1,k_2} := \sum_{n_1=0}^{N_1-1} a_{n_1,k_2}\omega_{N_1}^{-k_1 n_1}$$

$$\mathrm{DFT2}_{k_1,k_2} := \sum_{n_2=0}^{N_2-1} \mathrm{DFT1}_{k_1,n_2}\omega_{N_2}^{-k_2 n_2} = \sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1} a_{n_1,n_2}\omega_{N_1}^{-k_1 n_1}\omega_{N_2}^{-k_2 n_2} = A_{k_1,k_2}$$

Dzięki tej obserwacji możemy użyć jednowymiarowego algorytmu FFT Cooleya-Tukeya do obliczenia wielowymiarowej dyskretnej transformaty Fouriera. Konieczne będzie wtedy obliczenie N_1 razy DFT wielkości N_2, a następnie N_2 razy DFT wielkości N_1. Złożoność takiego rozwiązania będzie wynosiła $\mathcal{O}(N \log N)$ dla $N = N_1 N_2$ co pokazuję poniżej.

Dowód.

$$\mathcal{O}(N_1 \log N_1)N_2 + \mathcal{O}(N_2 \log N_2)N_1 = \mathcal{O}(N_1 N_2 \log N_1) + \mathcal{O}(N_1 N_2 \log N_2) =$$
$$\mathcal{O}(N \log N_1) + \mathcal{O}(N \log N_2) = \mathcal{O}(N \max\{\log N_1, \log N_2\}) =$$

co możemy dalej ograniczyć przez:

$$= \mathcal{O}(N \log N)$$

□

5.2 Algorytm "vector-radix"

Kolejny algorytm jest generalizacją pomysłu z algorytmu Cooleya-Tukeya na więcej wymiarów. W jednowymiarowej wersji podstawą była liczba, która dzieliła długość ciągu, w wielowymiarowym algorytmie podstawą jest wektor. Dla dwuwymiarowego przypadku z podstawą (r_1, r_2) użyję podstawienia:

$$\text{dla } j \in \{1,2\}:$$
$$n_j := r_j p_j + q_j \qquad \text{dla } p_j = 0,\ldots,\frac{N_j}{r_j}-1,\ q_j = 0,\ldots,r_j-1$$
$$k_j := u_j + v_j \frac{N_j}{r_j} \qquad \text{dla } u_j = 0,\ldots,\frac{N_j}{r_j}-1,\ v_j = 0,\ldots,r_j-1$$

Wtedy wzór na dyskretną transformatę Fouriera ma postać:

$$A_{k_1,k_2} = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} a_{n_1,n_2}\omega_{N_1}^{-k_1 n_1}\omega_{N_2}^{-k_2 n_2} =$$
$$\sum_{q_1=0}^{r_1-1}\sum_{q_2=0}^{r_2-1}\sum_{p_1=0}^{N_1/r_1-1}\sum_{p_2=0}^{N_2/r_2-1} a_{\substack{r_1p_1+q_1,\\ r_2p_2+q_2}}\omega_{N_1}^{-(u_1+v_1\frac{N_1}{r_1})(r_1p_1+q_1)}\omega_{N_2}^{-(u_2+v_2\frac{N_2}{r_2})(r_2p_2+q_2)} =$$
$$\sum_{q_1=0}^{r_1-1}\sum_{q_2=0}^{r_2-1}\sum_{p_1=0}^{N_1/r_1-1}\sum_{p_2=0}^{N_2/r_2-1} a_{\substack{r_1p_1+q_1,\\ r_2p_2+q_2}}\omega_{N_1}^{-u_1r_1p_1}\omega_{N_1}^{-v_1N_1p_1}\omega_{N_1}^{-u_1q_1}\omega_{N_1}^{-v_1\frac{N_1}{r_1}q_1}\omega_{N_2}^{-u_2r_2p_2}\cdot$$
$$\cdot\,\omega_{N_2}^{-v_2N_2p_2}\omega_{N_2}^{-u_2q_2}\omega_{N_2}^{-v_2\frac{N_2}{r_2}q_2} =$$
$$\sum_{q_1=0}^{r_1-1}\sum_{q_2=0}^{r_2-1}\underbrace{\left(\sum_{p_1=0}^{N_1/r_1-1}\sum_{p_2=0}^{N_2/r_2-1} a_{\substack{r_1p_1+q_1,\\ r_2p_2+q_2}}\omega_{N_1/r_1}^{-p_1u_1}\omega_{N_2/r_2}^{-p_2u_2}\right)}_{\mathrm{DFT}(q_1,q_2)_{u_1,u_2}}\omega_{N_1}^{-q_1u_1}\omega_{N_2}^{-q_2u_2}\omega_{r_1}^{-q_1v_1}\omega_{r_2}^{-q_2v_2}$$

Powyżej jako $\mathrm{DFT}(q_1,q_2)$ oznaczyłam dyskretną transformatę Fouriera elementów o indeksach i,j, gdzie reszta z dzielenia i przez r_1 wynosi q_1, a reszta z dzielenia j przez r_2 wynosi q_2. Jak widac z powyższego wzoru, w tym algorytmie potrzebne jest policzenie $r_1 \cdot r_2$ dwuwymiarowych DFT wielkości $\frac{N_1}{r_1} \times \frac{N_2}{r_2}$. Aby uzyskać DFT całego ciągu sumujemy ich wartości po przemnożeniu przez pewne współczynniki, co wymaga ilości operacji $\mathcal{O}(N_1N_2)$ dla stałych r_1, r_2. Złożoność całej dwuwymiarowej wersji algorytmu Cooleya-Tukeya to $\mathcal{O}(M \log M)$ dla $M = N_1N_2$. Poniżej pokażę dowód złożoności dla przypadku, gdzie $r_1 = r_2 = 2$ (Radix-2 DIT), analogiczny do dowodu wersji jednowymiarowej.

Dowód.

Dla uproszczenia załóżmy, że $N_1 = N_2 = N, N = 2^k$

$$T(N\times N) = 4T\left(\frac{N}{2}\times\frac{N}{2}\right)+\mathcal{O}(N^2) \leqslant 4T\left(\frac{N}{2}\times\frac{N}{2}\right)+CN^2 \leqslant$$
$$4\cdot\left(4T\left(\frac{N}{4}\times\frac{N}{4}\right)+C\cdot\frac{N^2}{4}\right)+CN^2 = 16T\left(\frac{N}{4}\times\frac{N}{4}\right)+2CN^2 \leqslant$$
$$16\cdot\left(4T\left(\frac{N}{8}\times\frac{N}{8}\right)+C\cdot\frac{N^2}{16}\right)+2CN^2 = 64T\left(\frac{N}{8}\times\frac{N}{8}\right)+3CN^2 \leqslant$$
$$\vdots$$
$$\leqslant 4^kT(1\times1)+kCN^2 = (*)$$
$$N^2 = 4^k \quad 2k = \log_2 N^2 \quad \text{więc:}$$
$$(*) = N^2T(1\times1)+\frac{1}{2}CN^2\log N^2 = \mathcal{O}(N^2\log N^2)$$

□

6 Zastosowania

6.1 Zastosowania jednowymiarowe

Algorytm FFT ma najszersze zastosowania w cyfrowym przetwarzaniu sygnałów. Rozbijanie sygnału dyskretnego na poszczególne sinusoidy pozwala na modyfikowanie lub filtrowanie go w dziedzinie częstotliwości. Na przykład przy takiej reprezentacji sygnału dźwiękowego możliwe jest filtrowanie szumów przez usunięcie niechcianych częstotliwości z DFT sygnału i uzyskanie sygnału wynikowego dzięki odwzorowaniu odwrotnemu DFT.

6.1.1 Szybkie mnożenie wielomianów

Niech $A(x) = a_0 + a_1 x + \cdots + a_N x^N$ i $B(x) = b_0 + b_1 x + \cdots + b_N x^N$ będą wielomianami stopnia N. Aby uzyskać wynik ich mnożenia $C(x) = A(x)B(x)$ jako ich reprezentacji możemy użyć dwóch tablic A i B długości $N + 1$ zawierających współczynniki tych wielomianów i użyć poniższego algorytmu.

Algorytm 2 Algorytm naiwny mnożenia wielomianów

```
function MULTIPLY(A, B, N)
    C ← tablica wielkości N + 1 wypełniona zerami
    for i = 0, ..., N do
        for j = 0, ..., N do
            C[i + j] ← C[i + j] + A[i] · B[j]
        end for
    end for
    return C
end function
```

Ze względu na dwie pętle przechodzące od 0 do N w powyższym algorytmie jego złożoność dla wielomianów stopnia N to $\mathcal{O}(N^2)$. Jesteśmy w stanie znaleźć algorytm o lepszej złożoności niż powyższy dzięki zmianie sposobu reprezentowania wielomianu.

Dla dowolnych różnych od siebie punktów $x_0, x_1, \ldots, x_N$ i wielomianu A stopnia N ciąg par $\{(x_0, A(x_0)), (x_1, A(x_1)), \cdots, (x_N, A(x_N))\}$ daje jednoznaczną reprezentację A. Wynika to z tego, aby przejść z niej do postaci współczynników wystarczy rozwiązać układ $N + 1$ równań o $N + 1$ niewiadomych.

Dwa wielomiany A i B podane w takiej reprezntacji jesteśmy w stanie wymnożnyć w czasie $\mathcal{O}(N)$ wykonując $N+1$ mnożeń wartości wielomianów w punktach, ponieważ reprezentacja wielomianu $C = A \cdot B$ to:

$$\{(x_0, A(x_0) \cdot B(x_0)), (x_1, A(x_1) \cdot B(x_1)), \ldots, (x_N, A(x_N) \cdot B(x_N))\}$$

Zamiana z reprezentacji współczynnikowej na reprezentację wartościami punktów wymaga ewaluacji wielomianu w $N + 1$ punktach, a złożoność pojedynczej takiej operacji to $N+1$, więc mimo powyżej opisanej zalety tej reprezentacji nadal złożoność wymnażania wielomianów w ten sposób to $\mathcal{O}(N^2)$.

Aby zredukować złożoność zamiany do $\mathcal{O}(N \log N)$ możemy użyć algorytmu FFT. Dla $a_0, \ldots, a_N$ będących współczynnikami wielomianu A dyskretna transformata Fouriera tego ciągu to ciąg $\sum_{n=0}^{N} a_n \omega_{N+1}^{-kn}$ dla $k = 0, \ldots, N$. Jest to więc ciąg wartości wielomianu w punktach $\omega_{N+1}^0, \omega_{N+1}^{-1}, \ldots, \omega_{N+1}^{-N}$. Użycie tych punktów do reprezentowania wielomianów daje nam następujący algorytm:

Algorytm 3 Szybkie mnożenie wielomianów z użyciem FFT

```
function MULTIPLY(A, B, N)
    A' ← FFT(A)                          ▷ zmiana reprezentacji
    B' ← FFT(B)
    C ← pusta tablica wielości N + 1
    for i = 0, ..., N do
        C[i] = A[i] · B[i]
    end for
    C' ← InverseFFT(C)                   ▷ powrót do reprezentacji współczynnikami
    return C'
end function
```

Złożoność powyższego algorytmu to $2\mathcal{O}(N \log N) + \mathcal{O}(N) = \mathcal{O}(N \log N)$.

Algorytmu tego możemy użyć także do mnożenia dużych liczb całkowitych. Na liczby w systemie dziesiętnym możemy patrzeć jak na wartości w punkcie 10 wielomianu, którego kolejne współczynniki to cyfry danej liczby. Z tego powodu aby wykonać szybkie mnożenie dwóch liczb wystarczy policzyć DFT ciągu cyfr obu liczb, przemnożyć wartości na tych samych indeksach i uzyskać cyfry wyniku odwzorowaniem odwrotnym DFT.

6.2 Zastosowania dwuwymiarowe

Na obrazy rastrowe możemy patrzeć jako na dwuwymiarowe macierze wartości oznaczających kolory poszczególnych pikseli. Analogicznie do przetwarzania jednowymiarowych sygnałów FFT 2D pozwala na modyfikowanie obrazów w dziedzinie częstotliwości. Dzięki temu główne zastosowania dwuwymiarowego FFT są związane z przetwarzaniem obrazów. Poniżej pokażę w jaki sposób możemy użyć dyskretnej transformaty Fouriera do kompresji stratnej bitmapy.

6.2.1 Kompresja obrazów

Da się zauważyć, że w rozkładzie obrazów przez FFT występuje wiele małych wartości, które w niewielki sposób wpływają na odbiór obrazu przez człowieka. Są one często związane z bardzo wysokimi częstotliwościami. Zamiana takich wartości na zera nie zmieni znacznie tego jak wygląda obraz, a pozwala często na usunięcie większości wartości z DFT. Taka operacja pozwoli na zapisanie wartości DFT na znacznie mniejszej ilości pamięci, ze względu na niesienie mniejszej liczby informacji. Możemy to zrobić przez wybór małej stałej i zamianę na zera wszystkich elementów mniejszych od niej. Poniżej prezentuję wyniki testu takiego algorytmu kompresji, którego implementacja znajduje się w pliku `fast_fourier_transform.ipynb`.

Rysunek 6: Porównanie obrazu oryginalnego z obrazami uzyskanymi przez odwrotne FFT po zamianie kolejno 66% i 95% najmniejszych wartości jego DFT na zera.

Jeden z najczęściej stosowanych algorytmów kompresji stratnej obrazów - JPEG - używa podejścia podobnego do uproszczonej metody opisanej powyżej. Użyta tam transformata to dyskretna transformata cosinusowa (DCT) analogicznia do DFT, działająca na ciągach liczb rzeczywistych.

7 Implementacja i wyniki

Do pracy dołączam plik Jupyter Notebook `fast_fourier_transform.ipynb`. Zawarłam w nim przykładową implementację metod obliczania dyskretnej transformaty Fouriera, które opisałam powyżej. Są to dla przypadku jednowymiarowego - algorytm Cooleya-Tukeya o podstawie 2, a dla dwuwymiarowego "row-column" i "vector-radix" również przy podstawie 2.

Znajdują się w nim również porównania czasów wykonania tych algorytmów z wersjami z definicji i wizualizacja ich wyników za pomocą wykresów. W przypadku jednowymiarowym podczas porównywania czasów użyłam ciąg długości 2048. Wyniki testu były następujące.

Algorytm	Czas wykonania
Obliczanie z definicji	2.633160687 s
Szybka Transformata Fouriera	0.080158529 s

Zgodnie z oczekiwaniami czas wykonania FFT był znacznie krótszy niż wersji obliczanej z definicji ze względu na złożoność $\mathcal{O}(N \log N)$.

W przypadku dwuwymiarowym wykonałam dwa testy. W pierwszym użyłam danych rozmiaru 32×32 i porównałam dwa omówione algorytmy FFT z obliczaniem z definicji. Poniżej znajdują się wyniki testu.

Algorytm	Czas wykonania
Obliczanie z definicji	9.858511204999985 s
FFT w wersji "row-column"	0.068875935999983 s
FFT w wersji "vector-radix"	0.030879343000009 s

Z wyników testu widać, że nawet dla tak małego rozmiaru obrazu czas obliczania dwuwymiarowej DFT z definicji drastycznie różni się od czasów obu algorytmów szybkiej transformaty Fouriera. Aby dokładniej porównać ich czasy wykonania przeprowadziłam kolejny test bez zawierania w nim wersji obliczanej z definicji. Użyłam danych rozmiaru 1024×1024. Wyniki testu były następujące.

Algorytm	Czas wykonania
FFT "row-column"	74.921213618 s
FFT "vector-radix"	32.930432062 s

Z powyższego testu widać, że pomimo tego, że oba algorytmy mają złożoność $\mathcal{O}(N \log N)$ czas wykonania algorytu "vector-radix" okazał się mniejszy. Można pokażać, że algorytm ten wymaga w ogólności wykonania mniejszej ilości operacji niż wersja "row-column".

8 Podsumowanie

Powyższa praca zawiera wprowadzenie do zagadnień związanych z dyskretną transformatą Fouriera funkcji jedno i wielowymiarowych. Zgodnie z tematem zaprezentowane są w niej metody obliczania jej algorytmami Szybkiej Transformaty Fouriera. W jednowymiarowym przypadku jest to algorytm Cooleya-Tukeya, a w dwuwymiarowym

- "row-column" i "vector-radix". W pracy zawarte są uzasadnienia poprawności tych metod oraz dowody złożoności, dzięki którym widać zyski możliwe do uzyskania dzięki ich używaniu. Podane są również konkretne przykłady zastosowań dyskretnej transformaty Fouriera, w których można wykorzystać algorytmy FFT. Kod ilustrujący pracę jest zawarty w dołączonym pliku `fast_fourier_transform.ipynb`.

9 Bibliografia

Źródła

[T H89] R. L. Rivest T. H. Cormen C. E. Leiserson. *"Wprowadzeie do algorytmów"*. 1989.

[Sza08] J. Szabatin. *"Podstawy teorii sygnałów"*. 2008.

[Pud17] S. Pudasaini. *"Computation of IDFT through forward DFT"*. 2017. URL: https://www.researchgate.net/publication/319226292_Computation_of_IDFT_through_forward_DFT.

[Jia20] Yan-Bin Jia. *Polynomial Multiplication and Fast Fourier Transform*. 2020. URL: https://faculty.sites.iastate.edu/jia/files/inline-files/polymultiply.pdf.

[Bou] A. Bounchaleun. *"An Elementary Introduction to Fast Fourier Transform Algorithms"*. URL: https://math.uchicago.edu/~may/REU2019/REUPapers/Bounchaleun.pdf.

Grafiki

[Wei] Eric W. Weisstein. *"Fourier Series." MathWorld–A Wolfram Web Resource.* URL: https://mathworld.wolfram.com/FourierSeries.html.