Metody numeryczne

Sprawozdanie 11

Szybka transformata Fouriera

Mateusz Górczany

Maj 2020

1 Wstęp teoretyczny

1.1 Opis

Transformata Fouriera to proces, który przenosi daną funkcję z dziedziny czasu do dziedziny częstotliwości. Taki zabieg może być użyteczny do:

- interpolacji, aproksymacji
- szybkiego mnożenia wielomianów
- cyfrowego przetwarzania sygnałów (zaprezentowane w sekcji "Opis zadania")
- kompresja danych
- analiza sygnałów czasowych
- rozwiązywanie równań różniczkowych
- całkowanie.

1.2 Szybka transformata Fouriera (FFT) - algorytm *radix-2*

Niech x_k będzie funkcją w dziedzinie czasu, natomiast X_k jej transformatą w dziedzinie częstotliwości, określoną się w następujący sposób:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = 0, \dots, N-1, \tag{1}$$

natomiast n jest potęgą dwójki do k-tej potęgi. Rozdzielając równanie (1) na sumę rozwiązań parzystych i nieparzystych:

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k} \quad (*) \tag{2}$$

$$= \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT elementów parzystych } x_m} + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT elementów nieparzystych } x_m} = E_k + e^{-\frac{2\pi i}{N}k} O_k, \quad (*) \tag{3}$$

otrzymano formułę:

$$X_k = \begin{cases} E_k + e^{-\frac{2\pi i}{N}k} O_k & \text{if } k < N/2 \\ E_{k-N/2} - e^{-\frac{2\pi i}{N}(k-N/2)} O_{k-N/2} & \text{if } k \geqslant N/2. \quad (*) \end{cases} \tag{4}$$

Algorytm *radix-2* zmniejsza złożoność obliczeniową Transformaty Fouriera z $O(n^2)$ do $O(nlog(n))$.

2 Opis rozwiązywanego zadania na laboratoriach

Zadanie polegało na zaszumieniu funkcji $y_0(i)$, a następnie za pomocą transformaty Fouriera, pozbycie się szumów o wartości poniżej 25% maksymalnej wartości funkcji.
Zabieg przeprowadzono dla następujących $k = \{6, 8, 10\}$.

$$y_0(i) = sin(\omega i) + sin(\omega i) + sin(3\omega i), \tag{5}$$

gdzie $\omega = 2\frac{2\pi}{N}$, $i = \{1, 2, 3, ..., N = 2^k\}$

$$y(i) \;=\; y_0(i) + a \tag{6}$$

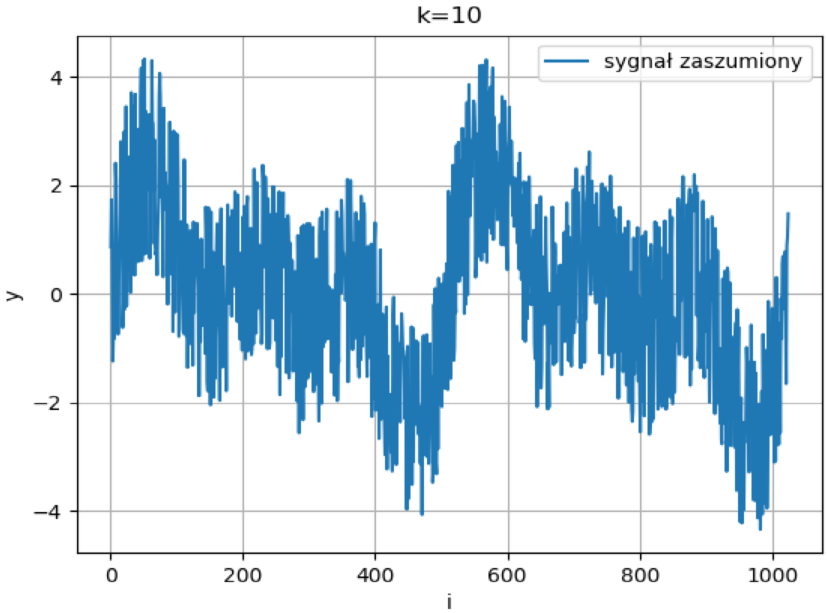
$$a = 2sign \cdot X \qquad\qquad X = \frac{rand()}{RANDMAX + 1.0} \tag{7}$$

$$\text{sign} = \begin{cases} +1, & Y > \frac{1}{2} \\ -1, & Y \leqslant \frac{1}{2} \end{cases} \tag{8}$$

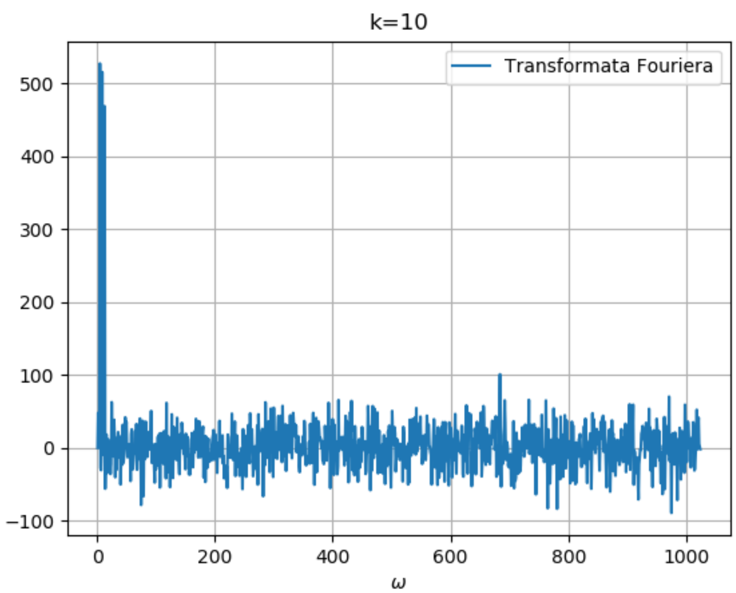
Na początku wyznaczono wartości $y_0(i)$ i zapisano do wektora, następnie obliczono funkcję zaszumioną $y(i)$, dodając zakłócenie a. Wynik także zapisano. W kolejnym kroku użyto szybkiej transformaty Fouriera.

Na uzyskanej transformacie dokonano, powyżej opisanej, dyskryminacji wartości uznawanych za szum. Na końcu użyto transformaty odwrotnej i przeskalowano. Operację powtórzono dla każdego k. Wyniki są widoczne poniżej.

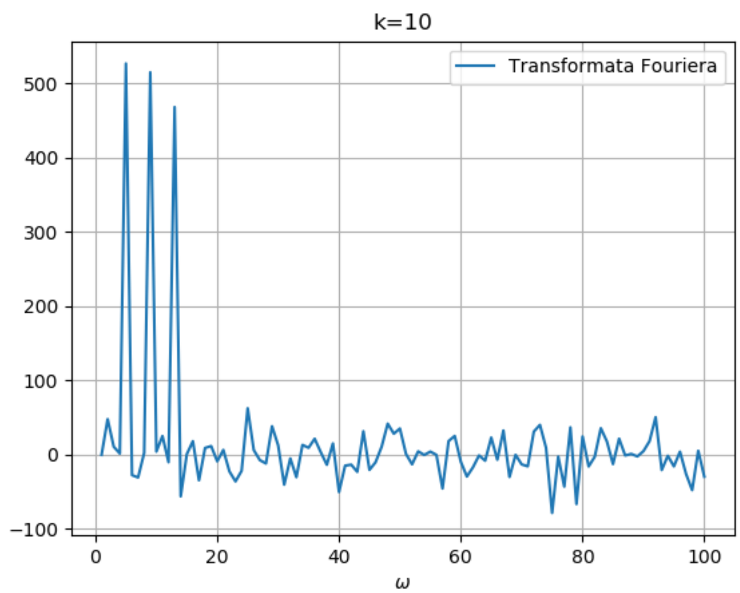
3 Wyniki



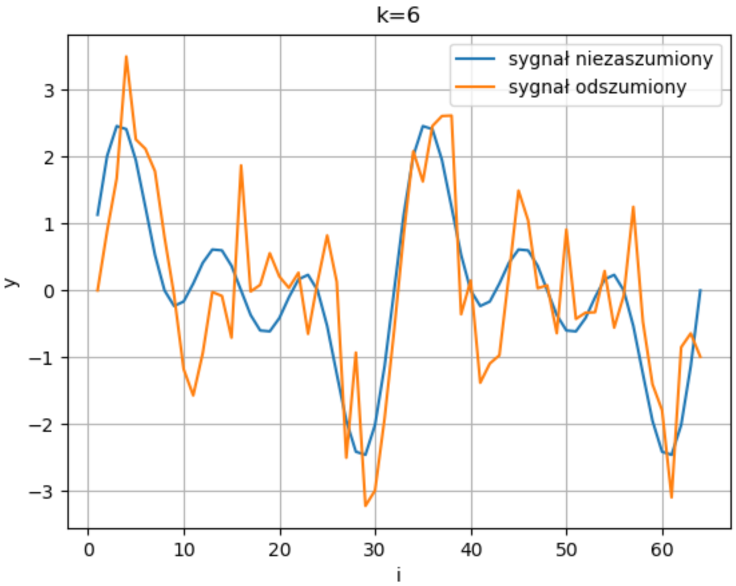
(a) n = 1024



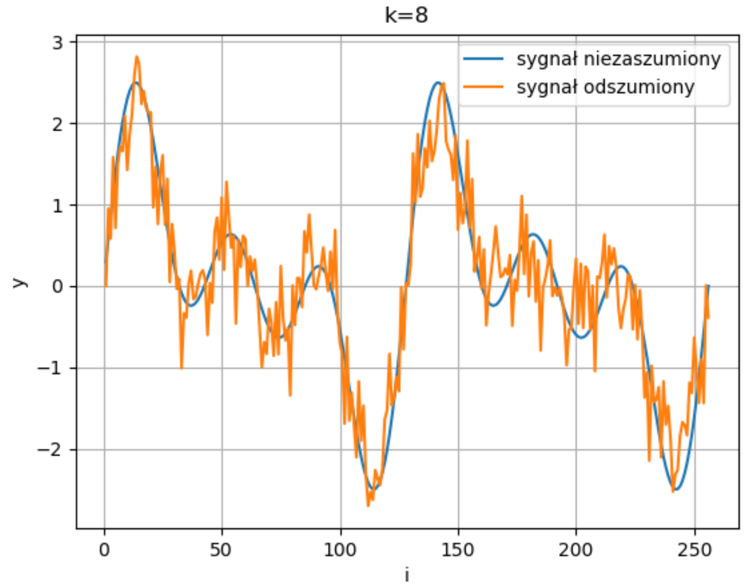
(b) n = 1024



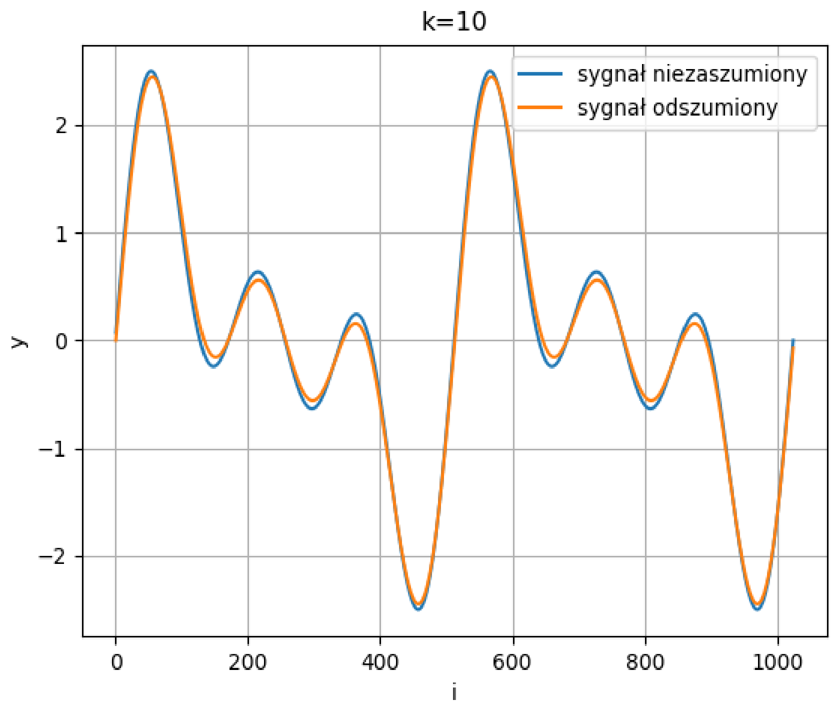
(c) n = 1024



(d) n = 64



(e) n = 256



(f) n = 1024

4 Wnioski

Wyniki uzyskane dla $k = 6 \rightarrow N = 2^6 = 64$ oraz $k = 8 \rightarrow N = 2^8 = 256$, gdzie N to liczba argumentów funkcji, nie były wystarczające do wiernego oddania kształtu funkcji pierwotnej. Dopiero dla $k = 10 \rightarrow N = 2^{10} = 1024$ udało się odzyskać oryginał z zadowalającym efektem. Każdy z uzyskannych wyników w pewien sposób oddawał kształt $y(i)$. Niestety dla małej liczby punktów, program tworzący wykres łączył każdy z nich za pomocą funkcji liniowej, co przekłada się na wygląd wykresu.

Warto zauważyć, że dokonana operacja jest jedynie aproksymacją, a nie interpolacją, gdyż część informacji została utracona podczas dyskryminacji wartości poniżej 25% maksymalnej wartości $y(i)$, a także podczas dodawania szumu do oryginału.

Przypisy:
(*) - wzory zaczerpnięte z wikipedii:
`https://pl.wikipedia.org/wiki/Algorytm_Cooleya-Tukeya`