Akademia Górniczo-Hutnicza w Krakowie

Wydział Fizyki i Informatyki Stosowanej

Metody Numeryczne

Laboratorium 11: Aproksymacja sygnału okresowego przy użyciu FFT

Andrzej Świętek

21.05.2024

Contents

1 Wstęp teoretyczny **2**
- 1.1 Transformata Fouriera . . . 2
- 1.2 Szybka Transformata Fouriera . . . 2
  - 1.2.1 Opis . . . 2
  - 1.2.2 Algorytm FFT . . . 2
- 1.3 Aproksymacja sygnału okresowego przy użyciu FFT . . . 3
  - 1.3.1 Opis . . . 3
  - 1.3.2 Algorytm aproksymacji sygnału przy użyciu FFT . . . 4

2 Problem **4**

3 Implementacja **4**
- 3.1 Inicjalizacja . . . 4
- 3.2 Wnętrze głównej pętli - ciąg dalszy . . . 5
- 3.3 Funckje generujące sygnał i jego zakłucenia . . . 6
- 3.4 Implementacja funkcji filtrującej sygnał . . . 7

4 Wyniki **8**
- 4.1 Wyniki dla $k = 8$ . . . 8
- 4.2 Wyniki dla $k = 10$ . . . 12
- 4.3 Wyniki dla $k = 12$ . . . 16

5 Analiza wyników **19**

6 Wnioski **21**

1. Wstęp teoretyczny

1.1. Transformata Fouriera

Transformata Fouriera jest narzędziem matematycznym, które przekształca funkcję czasu (lub przestrzeni) na funkcję częstotliwości. Dzięki temu możliwe jest analizowanie widma częstotliwościowego sygnału. Transformata Fouriera dla sygnału ciągłego $x(t)$ jest definiowana jako:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

gdzie $X(f)$ jest funkcją częstotliwości f, a j to jednostka urojona.

Dla sygnałów dyskretnych, takich jak sygnały cyfrowe, używa się Dyskretnej Transformaty Fouriera (DFT), która jest zdefiniowana jako:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}$$

gdzie $x(n)$ jest sygnałem w dziedzinie czasu, $X(k)$ jest sygnałem w dziedzinie częstotliwości, a N to liczba próbek.

1.2. Szybka Transformata Fouriera

1.2.1. Opis

Szybka Transformata Fouriera (FFT) jest efektywnym algorytmem do obliczania Dyskretnej Transformaty Fouriera (DFT). Algorytm FFT redukuje złożoność obliczeniową DFT z $O(N^2)$ do $O(N \log N)$, gdzie N jest liczbą próbek sygnału. Dzięki temu możliwe jest szybkie przekształcanie sygnałów na dużych zbiorach danych.

1.2.2. Algorytm FFT

Algorytm FFT opiera się na zasadzie "dziel i zwyciężaj". Główne kroki algorytmu to: 1. Podział sygnału na podsygnały parzyste i nieparzyste. 2. Rekursywne zastosowanie FFT na podsygnałach. 3. Łączenie wyników za pomocą operacji motylkowych.

Schematyczny opis algorytmu FFT:

Algorithm 1 FFT

```
1: procedure FFT(x)
2:     N ← length(x)
3:     if N = 1 then
4:         return x
5:     end if
6:     X_even ← FFT(x[0 : 2 : N − 1])
7:     X_odd ← FFT(x[1 : 2 : N])
8:     for k = 0 to N/2 − 1 do
9:         t ← e^{−2πik/N} · X_odd[k]
10:        X[k] ← X_even[k] + t
11:        X[k + N/2] ← X_even[k] − t
12:    end for
13:    return X
14: end procedure
```

1.3. Aproksymacja sygnału okresowego przy użyciu FFT

Aproksymacja sygnału okresowego za pomocą FFT polega na przekształceniu sygnału z dziedziny czasu na dziedzinę częstotliwości, a następnie na odtworzeniu sygnału z ograniczoną liczbą składowych częstotliwościowych. Dzięki FFT możemy skompresować sygnał poprzez eliminację mniej istotnych częstotliwości i zachowanie jedynie tych, które mają największy wkład w oryginalny sygnał. Proces ten można podzielić na kilka kroków:

1.3.1. Opis

1. Pobranie próbek sygnału: Zbieramy dyskretne próbki sygnału w dziedzinie czasu.

2. Zastosowanie FFT: Przekształcamy sygnał z dziedziny czasu na dziedzinę częstotliwości za pomocą FFT.

3. Filtracja częstotliwości: Eliminujemy częstotliwości o niskiej amplitudzie, które mają niewielki wpływ na oryginalny sygnał.

4. Zastosowanie odwrotnej FFT: Przekształcamy sygnał z powrotem na dziedzinę czasu, zachowując jedynie istotne składowe częstotliwościowe.

1.3.2. Algorytm aproksymacji sygnału przy użyciu FFT

Algorithm 2 Aproksymacja sygnału przy użyciu FFT

```
1: procedure APROKSYMACJA FFT(x, threshold)
2:     N ← length(x)
3:     X ← FFT(x)        ▷ Przekształcenie sygnału do dziedziny częstotliwości
4:     for k = 0 to N − 1 do
5:         if |X[k]| < threshold then
6:             X[k] ← 0          ▷ Eliminacja małych składowych częstotliwości
7:         end if
8:     end for
9:     x_approx ← IFFT(X)                    ▷ Odwrotna FFT do dziedziny czasu
10:    return x_approx
11: end procedure
```

2. Problem

Naszym zadaniem będzie zastosowanie FFT do odszumienia sygnału periodycznego. Sygnał zaszumiony generujemy zgodnie z poniższym algorytmem:

1. Wygenerować zaszumiony sygnał (część rzeczywista) i zapisać go do wekotra typu double. Długość wektora wynosi $N = 2k$, kolejno dla $k = 8, 10, 12$.
2. Wyznaczyć transformatę sygnału korzystając z biblioteki GSL.
3. Dla k = 8 sporządzić rysunek pokazujący część rzeczywistą i urojoną transformaty oraz rysunek pokazujący wartości modułów współczynników transformaty.
4. Dla każdego k przeprowadzić dyskryminację sygnału na poziomie max $|ck|/2$ tj. wyzerować te współczynniki transformaty (części rzeczywiste i urojone) które nie przekraczają tego progu.
5. Po dyskryminacji wyznaczyć transformatę odwrotną a otrzymany sygnał unormować dzieląc go przez N.
6. Dla każdego k wykonać po dwa rysunki:
   (a) sygnału zaburzonego i odszumionego oraz
   (b) sygnału niezaburzonego i odszumionego
7. W sprawozdaniu proszę przeanalizować uzyskane wyniki i określić wpływ częstości próbkowania na końcowy wynik.

3. Implementacja

3.1. Inicjalizacja

Na początku inicjalizujemy generator liczb losowych oraz definiujemy wartości k, dla których będziemy obliczać liczbę próbek N. Następnie generujemy czysty sygnał periodyczny, dodajemy do niego szum i zapisujemy oba sygnały.

```
 srand(static_cast<unsigned int>(time(0)));

// Define N = 2^k for k = 8, 10, 12
const int K_VALUES[] = {8, 10, 12};
const int NUM_K_VALUES = sizeof(K_VALUES) / sizeof(K_VALUES[0]);

    for (int k_idx = 0; k_idx < NUM_K_VALUES; ++k_idx) {
        int k = K_VALUES[k_idx];
        int N = 1 << k; // N = 2^k

        // CLEAN SIGNAL
        std::vector<double> signal(N);
        generate_periodic_signal(signal, N);
        save_signal(signal, "clean_signal_"+to_string(k));

        // NOISED SIGNAL
        add_noise(signal, N);
        save_signal(signal, "noiced_signal_"+to_string(k));

        // ... Ciag dalszy ponizej

    } // KONIEC GLOWNEJ PETLI
```

3.2. Wnętrze głównej pętli - ciąg dalszy

W tej części programu przygotowany zostaje wektor o rozmiarze $2N$ celem przechowania sygnału dla liczb zespolonych. Uzyskanie n-tej wartości dyskretnego sygnału odbywa się według następującego wzoru:

$$\mathfrak{Re}(signal[n]) = 2 \cdot i$$

$$\mathfrak{Im}(signal[n]) = 2 \cdot i + 2$$

Mając to na uwadze funkcja $prepare_data$ zapisuje wektor w taki własnie sposób - jednak części urojonej przypisuje wartość 0.
Mając tak przygotowane dane przekazujemy je do funkcji z biblioteki GSL aby dokonać transformaty w przód.

```
        // Prepare data array for FFT REAL + IMAGINARY
        std::vector<double> data(2 * N); // 2x bigger size (2N)
        prepare_data_array(signal, data);

        // Perform FFT
        gsl_fft_complex_radix2_forward(data.data(), 1, N);
        save_result(data, N, "transformed_"+to_string(k));
```

Listing 1: Ostatnie przygotowanie danych i wykonanie Szybkier Transfornaty Fouriera w Przód

Po dokonaniu transformaty można przystąpić do filtrowania sygnału. Aby tego dokonać należy najpierw znaleźć wartość graniczną - próg - poniżej której sygnał uznawany będzie za szum i będzie można go usunąć. Zmienna magnitude reprezentujemodół liczby zepolonej odpowiadającej wartości sygnału w danym momencie.

$$|z| = \sqrt{Re(z)^2 + Im(z)^2}, \qquad z \in \mathbb{C}$$

```
        // Apply the threshold filter
        double max_magnitude = 0.0;
        for (int i = 0; i < N; ++i) {
            double real_part = data[2 * i];
            double imag_part = data[2 * i + 1];
            double magnitude = std::sqrt(
                real_part * real_part + imag_part * imag_part
            );

            // Zapis wartosci magnitude do pliku

            if (magnitude > max_magnitude) {
                max_magnitude = magnitude;
            }
        }
        double threshold = max_magnitude / 2.0;
        filter_signal(data, N, threshold);
```

Listing 2: Znalezienie progu i oczyszczenie szumy

Wartość progu zostaje ustanowiona na połowę wartości maksymalnego modółu a następnie zostaje przekazana do funkcji która iterując przez elementy wektora porównuje czy wartość sygnału jest większa od progu. W przypadku jęsi nie wartość zostaje wyzerowana, ponieważ znaczy to dokładnie tyle że jest to szum.

Aby uzyskać odszumioną formę naszego pierwotnego sygnału należy posłużyć się transformatą odwrotną. W tym celu ponownie posługujemy się biblioteką GSL, która posiada już zaimplementowaną funkcję $gsl\_fft\_complex\_radix2\_backward()$. Po dokonaniu samej transformaty odwrotnej sygnał nalęzy znormalizować dzieląc zarówno część rzeczywistą jak i urojoną przez N - długość sygnału.

```
        // INVERSE FFT
        gsl_fft_complex_radix2_backward(data.data(), 1, N);

        // NORMALIZATION OF THE SIGNAL
        for (int i = 0; i < N; ++i) {
            data[2 * i] /= (double)N;
            data[2 * i + 1] /= (double)N;
        }

        save_result(data, N, "wynik"+to_string(k));

    } // KONIEC GLOWNEJ PETLI
```

Listing 3: Wykonanie transformaty odwrotnej na przefiltrowanym sygnale

3.3. Funckje generujące sygnał i jego zakłucenia

```
double signal_fn(double i, double omega) {
    return sin(omega*i) + sin(2*omega*i) + sin(3*omega*i);
}


void generate_periodic_signal(std::vector<double>& signal, int N) {
     double omega = TWO_PI / (double)N;
    for (int i = 0; i < N; ++i) signal[i] = signal_fn(i, omega);
}
```

```cpp
void add_noise(std::vector<double>& signal, int N) {
    for (int i = 0; i < N; ++i) {
        double delta = 2 * (
                static_cast<double>(rand()) / RAND_MAX - 0.5
            );
        signal[i] += delta;
    }
}
```

3.4. Implementacja funkcji filtrującej sygnał

```cpp
void prepare_data_array(
    const std::vector<double>& signal,
    std::vector<double>& data
) {
    int N = signal.size();
    for (int i = 0; i < N; ++i) {
        data[2 * i] = signal[i]; // Czesc Rzeczywista Re
        data[2 * i + 1] = 0.0;   // Czesc Im -> zero
    }
}
```

```cpp
void filter_signal(std::vector<double>& data, int N, double threshold) {
    for (int i = 0; i < N; ++i) {
        double real_part = data[2 * i];
        double imag_part = data[2 * i + 1];
        double magnitude = std::sqrt(
            real_part * real_part + imag_part * imag_part
        );
        if (magnitude < threshold) { // ponizej progu
            data[2 * i] = 0.0;       // usuwanie szumu -> Re
            data[2 * i + 1] = 0.0;   // usuwanie szumu -> Im
        }
    }
}
```

4. Wyniki

4.1. Wyniki dla $k = 8$

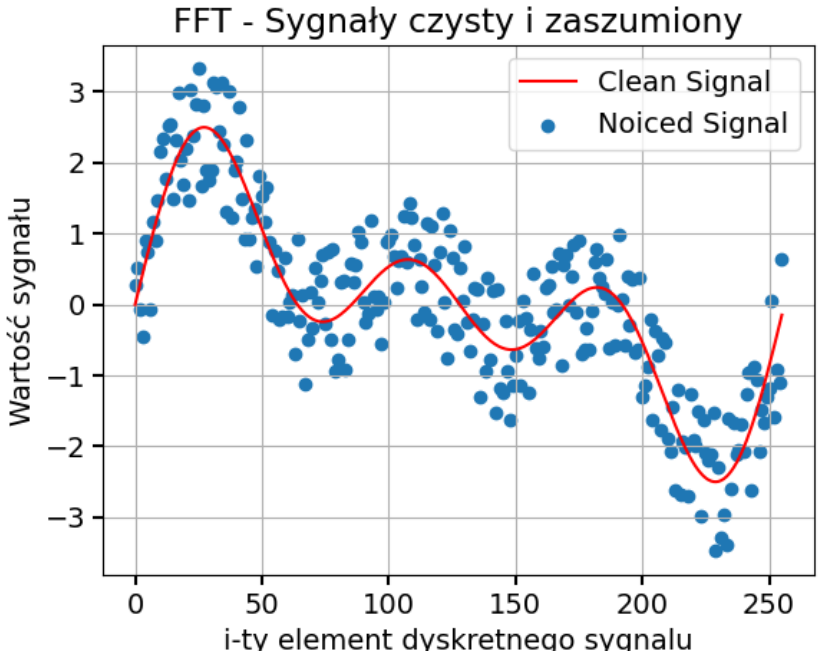

Figure 1: Wizualizacja sygnału bez zakłuceń (oryginalnego) i sygnału z wprowadzonym szumem dla $k = 8, N = 2^8$

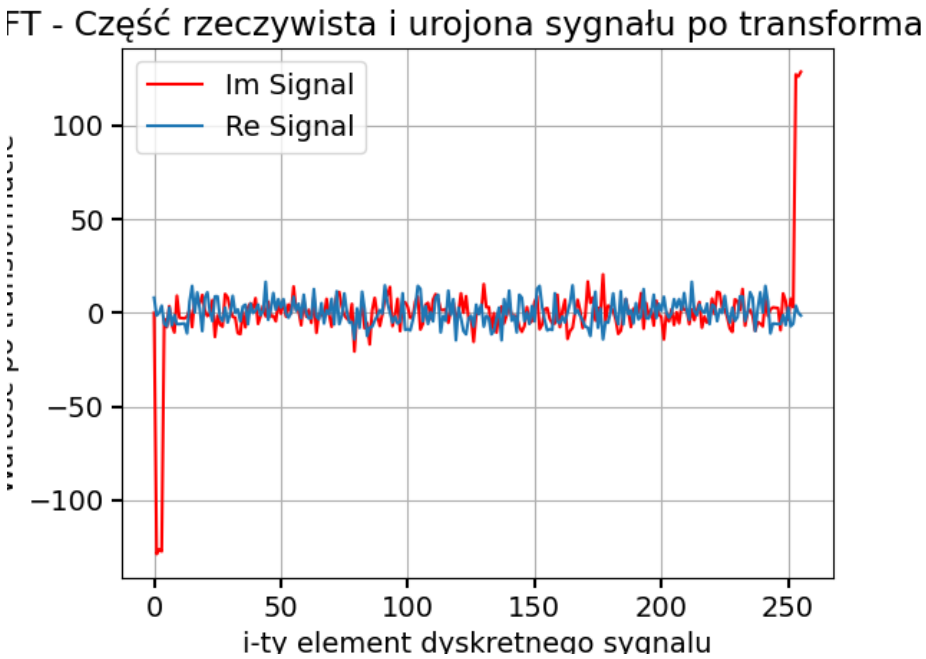

Figure 2: Wykres prezętujący część rzeczywistą i urojąną po dokonaniu transformaty Fouriera $k = 8$

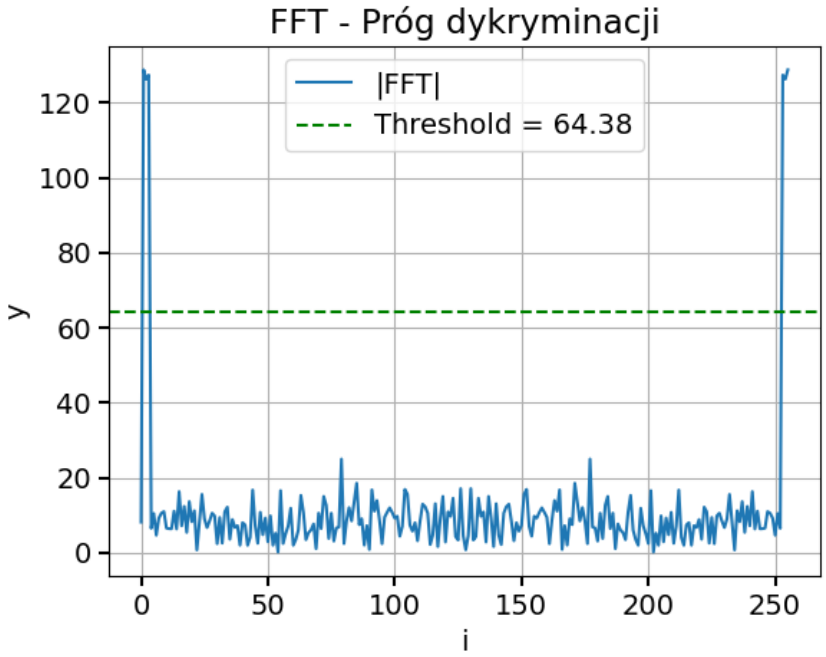

Figure 3: Wykres prezentujący wartości modółu wartości sygnału jako liczby zespolonej wraz wartością progu dyskryminacji $k = 8$

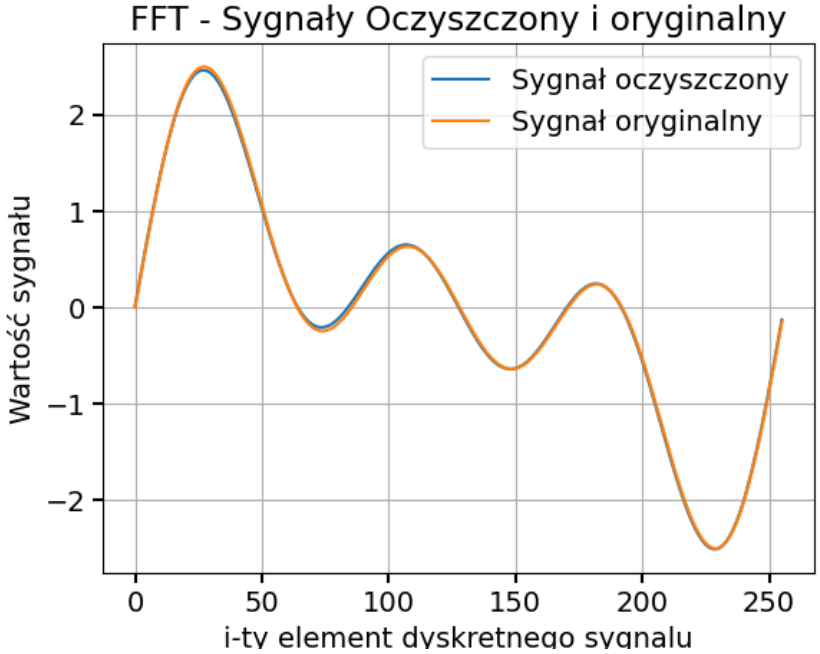

Figure 4: Porównanie sygnału oryginalnego (przed wprowadzeniem zakłuceń) z sygnałem oczyszczonym dla $k = 8$

4.2. Wyniki dla $k = 10$

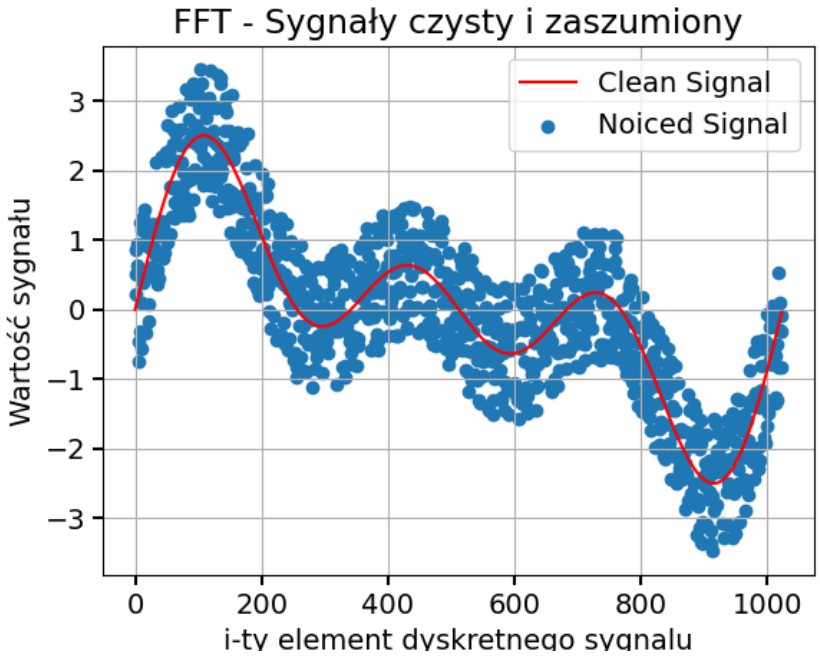

Figure 5: Wizualizacja sygnału bez zakłuceń (oryginalnego) i sygnału z wprowadzonym szumem dla $k = 10, N = 2^{10}$

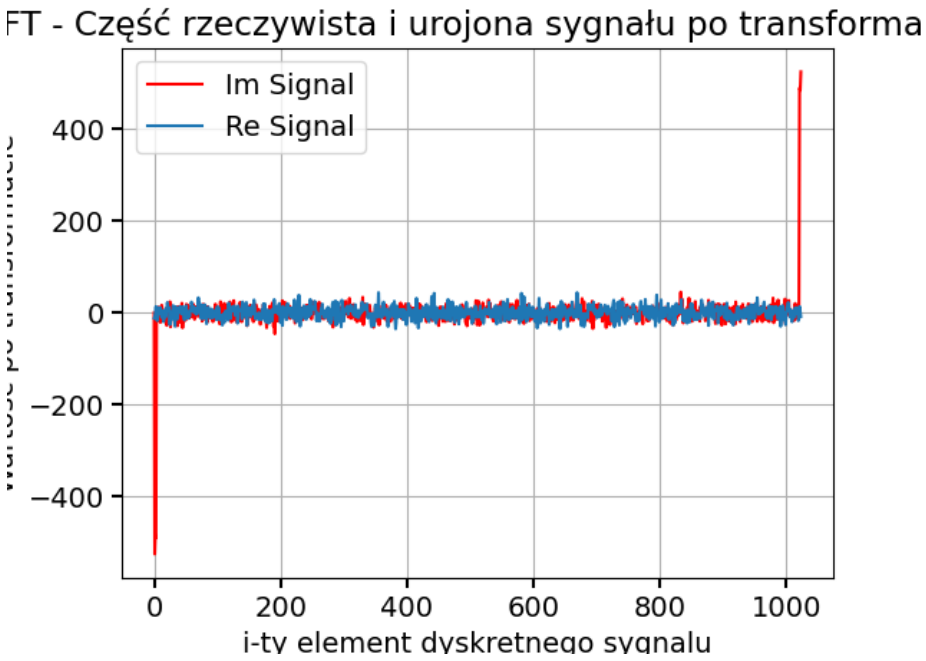

Figure 6: Wykres prezętujący część rzeczywistą i urojąną po dokonaniu transformaty Fouriera dla $k = 10$

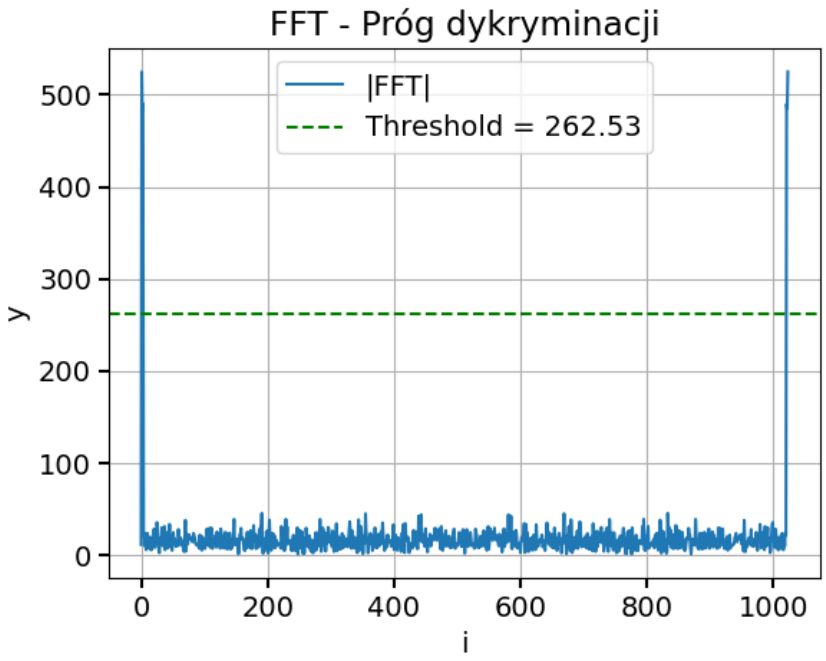

Figure 7: Wykres prezentujący wartości modółu wartości sygnału jako liczby zespolonej wraz wartością progu dyskryminacji dla $k = 10$

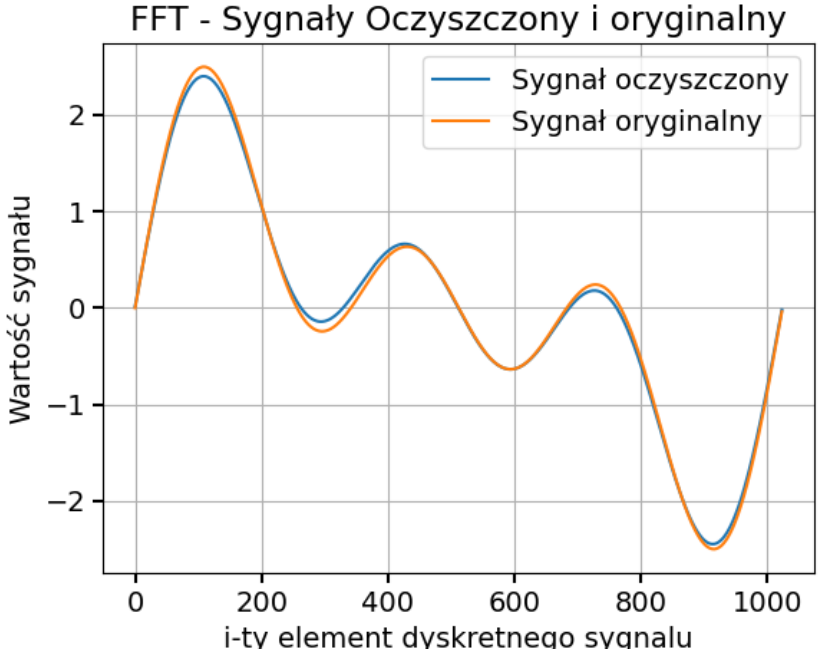

Figure 8: Porównanie sygnału oryginalnego (przed wprowadzeniem zakłuceń) z sygnałem oczyszczonym dla $k = 10$

4.3. Wyniki dla $k = 12$

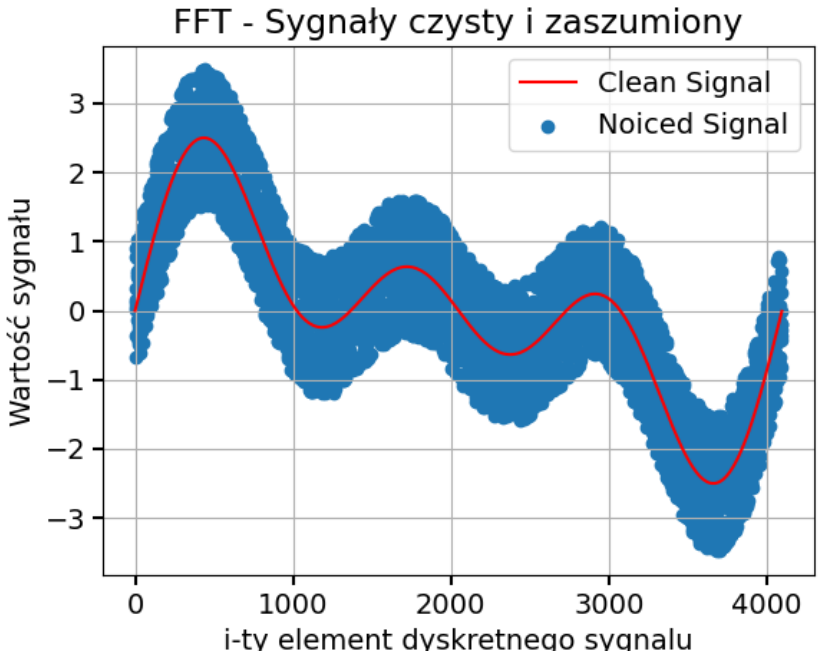

Figure 9: Wizualizacja sygnału bez zakłuceń (oryginalnego) i sygnału z wprowadzonym szumem dla $k = 12, N = 2^{12}$

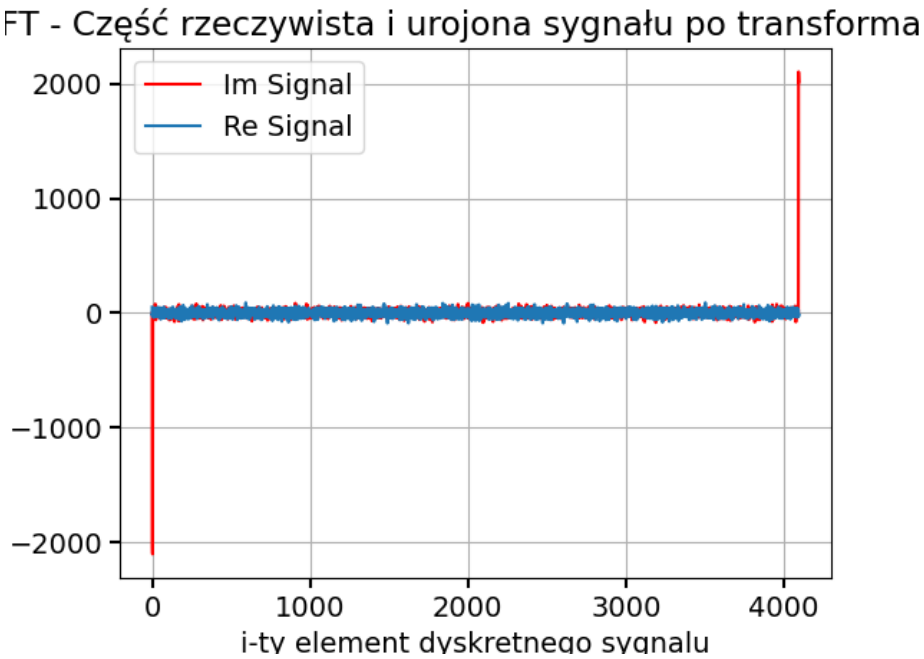

Figure 10: Wykres prezętujący część rzeczywistą i urojąną po dokonaniu transformaty Fouriera dla $k = 12$

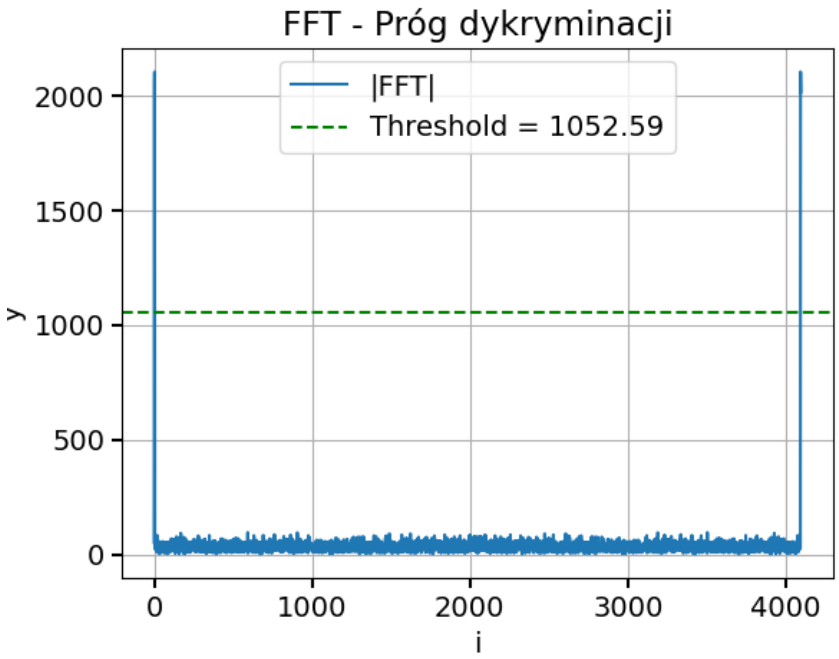

Figure 11: Wykres prezentujący wartości modółu wartości sygnału jako liczby zespolonej wraz wartością progu dyskryminacji dla $k = 12$

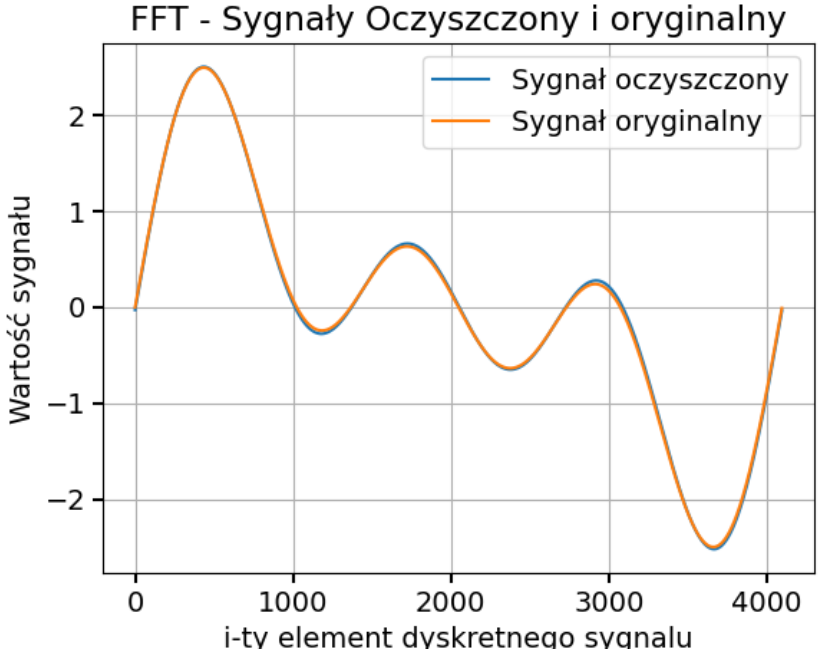

Figure 12: Porównanie sygnału oryginalnego (przed wprowadzeniem zakłuceń) z sygnałem oczyszczonym dla $k = 12$

5. Analiza wyników

W tej sekcji przeanalizujemy wyniki eksperymentów dotyczących aproksymacji sygnałów okresowych przy użyciu Szybkiej Transformaty Fouriera (FFT). Wyniki te obejmują zarówno sygnały oryginalne, zaszumione, jak i oczyszczone dla różnych wartości k.

1. **Kształt sygnału i jego dokładność**:
   - k=8: Sygnał oczyszczony zachowuje ogólny kształt oryginalnego sygnału, jednak widać pewne zniekształcenia i utratę drobniejszych szczegółów. Zakłócenia w sygnale są widoczne, ale są one znacząco zmniejszone w porównaniu do sygnału zaszumionego.
   - k=10: Przy większej liczbie próbek, sygnał oczyszczony jest bardziej precyzyjny. Detale sygnału są lepiej zachowane, a ilość szumów jest mniejsza w porównaniu do przypadku $k = 8$
   - k=12: Największa liczba próbek pozwala na uzyskanie sygnału oczyszczonego, który jest niemalże identyczny z sygnałem oryginalnym. Zakłócenia są minimalne, a jakość aproksymacji jest najwyższa spośród analizowanych przypadków.

2. **Widmo częstotliwości**:

   - $k = 8$:
     Widmo częstotliwości pokazuje znaczące składowe częstotliwościowe, jednak obecność szumów jest widoczna. Wyznaczenie progu dyskryminacji pozwala na eliminację większości szumów, ale kosztem pewnych składowych sygnału.
   - $k = 10$:
     Widmo częstotliwości jest bardziej wyraźne i mniej zaszumione. Wyższa rozdzielczość widma umożliwia lepsze odfiltrowanie nieistotnych częstotliwości, co przekłada się na dokładniejszy sygnał oczyszczony.
   - $k = 12$:
     Widmo częstotliwości jest najdokładniejsze. Znaczące składowe są wyraźnie widoczne, a szumy są minimalne. Proces filtracji jest najbardziej efektywny, co przekłada się na sygnał oczyszczony o najwyższej jakości.

3. **Wpływ liczby próbek na efektywność filtracji:**

   - **Liczba próbek** N: Wraz ze wzrostem liczby próbek, dokładność aproksymacji sygnału przy użyciu FFT znacząco wzrasta. Większa liczba próbek pozwala na lepsze odwzorowanie szczegółów sygnału oraz skuteczniejszą eliminację szumów.
   - **Próg dyskryminacji**: Wartość progu dyskryminacji była ustalana jako połowa maksymalnej amplitudy w widmie częstotliwości. Przy większej liczbie próbek, próg dyskryminacji jest bardziej precyzyjny, co prowadzi do skuteczniejszego usuwania szumów.

Podsumowując, eksperyment wykazał, że zwiększenie liczby próbek (czyli wyższa wartość k) znacząco poprawia jakość aproksymacji sygnału przy użyciu FFT. Większa liczba próbek umożliwia lepsze odwzorowanie szczegółów sygnału i skuteczniejszą eliminację szumów, co przekłada się na bardziej dokładny sygnał oczyszczony.

6. Wnioski

Przedstawione wyniki obejmowały różne wartości k, co odpowiada różnym liczebnościom próbek N. Po przeprowadzeniu analizy efektywności metody Fast Fourier Transform (FFT) w kontekście aproksymacji sygnałów okresowych oraz eliminacji szumów można wyciągnąć następujące wnioski.

- **Efektywność metody FFT** :
  FFT okazała się skutecznym narzędziem do analizy sygnałów okresowych. W każdym z przeanalizowanych przypadków (dla k=8, k=10 oraz k=12) metoda FFT pozwoliła na skuteczne usunięcie szumu, zachowując jednocześnie istotne składowe częstotliwościowe sygnału. Wyższa liczba próbek (N) przekładała się na dokładniejsze odwzorowanie oryginalnego sygnału.

- **Złożoność obliczeniowa:**
  Metoda FFT charakteryzuje się złożonością obliczeniową $O(NlogN)$ co czyni ją znacznie bardziej efektywną w porównaniu do klasycznej dyskretnej transformaty Fouriera (DFT), której złożoność wynosi $O(N^2)$. Dzięki temu, FFT jest praktycznym narzędziem do analizy sygnałów w czasie rzeczywistym oraz w aplikacjach wymagających dużej szybkości przetwarzania danych.

- **Dokładność i potencjalne zagrożenia:**

  Mimo swojej skuteczności, metoda FFT nie jest wolna od potencjalnych zagrożeń i ograniczeń:

  - **Niepełna dokładność:** Przy dużych poziomach szumu lub gdy sygnał zawiera komponenty o bardzo bliskich częstotliwościach, FFT może mieć trudności z precyzyjnym rozdzieleniem tych składowych. Może to prowadzić do niepełnej eliminacji szumu lub do zniekształceń w oczyszczonym sygnale.
  - **Efekty brzegowe:** Przy analizie sygnałów o ograniczonej długości mogą występować efekty brzegowe, które mogą zniekształcać wyniki transformaty. Rozwiązaniem tego problemu jest stosowanie okienkowania sygnałów przed przystąpieniem do transformacji.
  - **Zakłócenia wynikające z nieliniowości:** W praktycznych zastosowaniach sygnały mogą zawierać nieliniowe zakłócenia, które nie są idealnie reprezentowane w dziedzinie częstotliwości. FFT zakłada liniową superpozycję składowych, co może ograniczać jej dokładność w takich przypadkach.

- **Potencjalne usprawnienia:** Aby poprawić dokładność i skuteczność analizy sygnałów, można rozważyć kilka usprawnień:

  - Zwiększenie liczby próbek: Wzrost liczby próbek N poprawia dokładność odwzorowania sygnału i skuteczność eliminacji szumów.
  - Stosowanie zaawansowanych okienek: Użycie zaawansowanych technik okienkowania (np. okno Hanninga, Hamming, Blackman-Harris) może zmniejszyć efekty brzegowe i poprawić dokładność transformacji.

  - Metody hybrydowe: Połączenie FFT z innymi metodami analizy sygnałów (np. wavelet transform) może zwiększyć skuteczność odszumiania i analizy skomplikowanych sygnałów.

- **Zastosowania praktyczne** Metoda FFT znajduje szerokie zastosowanie w różnych dziedzinach:
  - Inżynieria sygnałowa: Analiza sygnałów audio, obrazów oraz sygnałów radarowych.
  - Telekomunikacja: Kompresja danych, modulacja oraz analiza spektralna.
  - Medycyna: Analiza sygnałów EEG, EKG oraz obrazów medycznych.