AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI
INSTYTUT ELEKTRONIKI

PROJEKT DYPLOMOWY

Aplikacja do analizy częstotliwościowej sygnału

Application for the frequency signal analysis

Autor: **Dominik Duchnik**
Kierunek studiów: Elektronika
Opiekun pracy: dr inż. Rafał Frączek

Kraków, 2024

Spis treści

1. Wstęp

1.1. Wprowadzenie

Cyfrowe przetwarzanie sygnałów jest jedną z najważniejszych gałęzi współczesnej inżynierii. Obszary takie jak telekomunikacja, medycyna, akustyka zostały zrewolucjonizowane przez własne, skomplikowane matematycznie algorytmy dotyczące przetwarzania sygnałów. Głównym jego celem jest efektywne manipulowanie sygnałami wykorzystując algorytmy numeryczne, które korzystają z cyfrowych układów elektronicznych. W przeciwieństwie do sygnałów analogowych, które są ciągłe w czasie, sygnał cyfrowy charakteryzuje się dyskretyzacją w dziedzinie czasu – jest reprezentowany przez skończony ciąg liczb. Sygnały w postaci cyfrowej posiadają wiele zalet, takich jak odporność na zakłócenia i straty informacji w porównaniu do sygnałów analogowych, co przekłada się na poprawę jakości i niezawodności ich przetwarzania. Ponadto dane w postaci cyfrowej są łatwiejsze w przechowywaniu i przesyłaniu. Jednym z kluczowych zagadnień w aspekcie cyfrowego przetwarzania sygnałów jest analiza częstotliwościowa, zwana również analizą fourierowską. Jest to matematyczne narzędzie umożliwiające przekształcenie sygnału z dziedziny czasu do częstotliwości, co znacznie ułatwia analizę sygnału i znajduje swoje zastosowanie m.in. w kompresji danych. Poza transformacją Fouriera w praktyce wykorzystywana jest również transformacja kosinusowa, która przekształca sygnał na sumę funkcji cosinus o współczynnikach rzeczywistych. Swoje zastosowanie znalazła w szczególności w kompresji obrazów JPEG.[1]

1.2. Geneza

Genezą powstania aplikacji, która będzie opisywana w niniejszej pracy jest powszechnie znany problem zrozumienia przez uczniów oraz studentów podstawowych zagadnień związanych z przetwarzaniem sygnałów. Z obserwacji przeprowadzonych podczas studiów można wywnioskować, iż abstrakcja oraz skomplikowany aparat matematyczny analizy sygnałów stanowią dużą przeszkodę w zrozumieniu materiału, który wiele wnosi do dziedziny elektroniki. Interaktywne narzędzia do wizualizacji mają potencjał do znacznego ułatwiania procesu nauki poprzez możliwość eksperymentowania z różnymi parametrami oraz obserwowania natychmiastowych efektów. Było to motywacją do opracowania innowacyjnego narzędzia edukacyjnego w postaci aplikacji, które ma służyć do ilustracji procesów zachodzących w obszarze, jakim jest analiza częstotliwościowa.

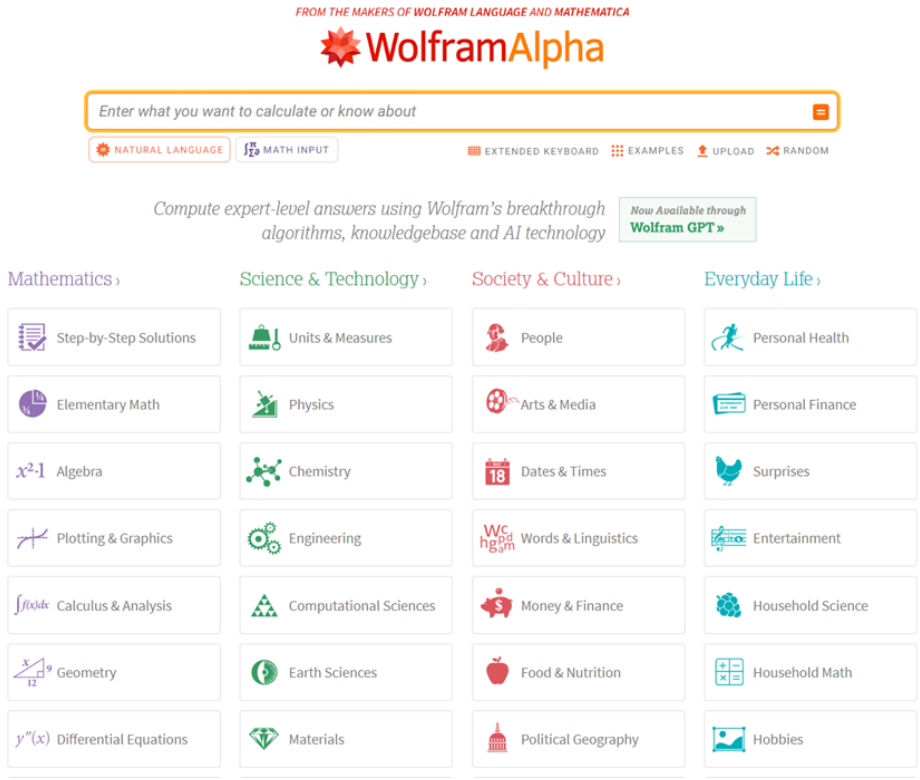
1.3. Cel pracy

Celem pracy jest opracowanie, zaprojektowanie oraz implementacja aplikacji desktopowej, którą można uruchomić na każdym komputerze z zainstalowanym środowiskiem języka programowania Python. Zakres prac obejmuje proces od projektowania interfejsu użytkownika po zastosowanie algorytmów obliczających transformaty dostarczonych przez biblioteki. Głównym elementem aplikacji jest graficzny interfejs użytkownika stworzony za pomocą biblioteki PyQt5. Elementy interaktywne, takie jak przyciski czy pola tekstowe, pozwalają na dynamiczną modyfikacje wybranych parametrów w czasie rzeczywistym, co pozwala na lepsze zrozumienie wpływu różnych ustawień na wynik. Aplikacja została zaimplementowana z wykorzystaniem popularnych bibliotek. NumPy w celu manipulacji danymi, SciPy, która dostarcza gotowe algorytmy do obliczania transformat oraz Matplotlib, służąca do wizualizacji wyników analizy. Uniwersalność oraz prostota obsługi sprawią, że aplikacja będzie cennym wsparciem w procesie nauczania.

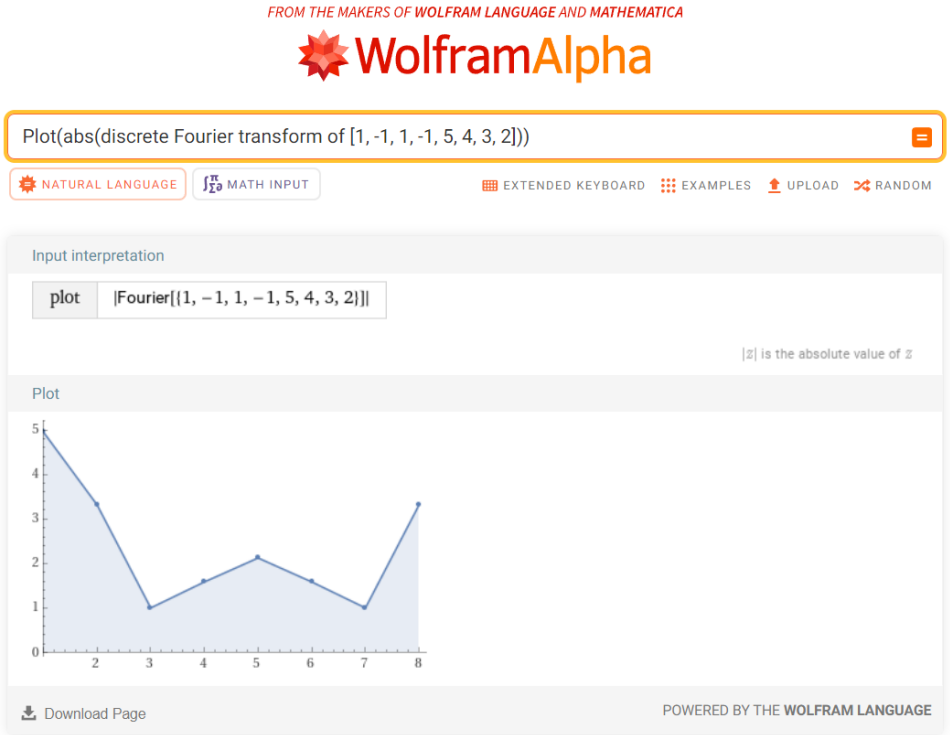
1.4. Przegląd istniejących rozwiązań

1.4.1. Wolfram Alpha

Wolfram Alpha (rys. 1.1) to program obliczeniowy, służący do analizy matematycznej oraz obliczeń numerycznych. Dostępny jest zarówno jako strona internetowa oraz aplikacja mobilna. Wynaleziony został przez Stephena Wolframa w 2005 roku i jest stale rozwijany. Silnik oparty jest na oprogramowaniu *Mathematica*. Za jego pomocą można wykonywać zarówno proste jak i złożone obliczenia, takie jak macierze, zagadnienia związane z trygonometrią oraz liczbami zespolonymi [2]. W kontekście analizy widmowej, program umożliwia obliczanie dyskretnej transformaty Fouriera danego ciągu próbek (rys. 1.2) oraz jej wizualizacje w postaci wykresu, a także obliczanie ciągłej transformaty Fouriera wybranej funkcji.



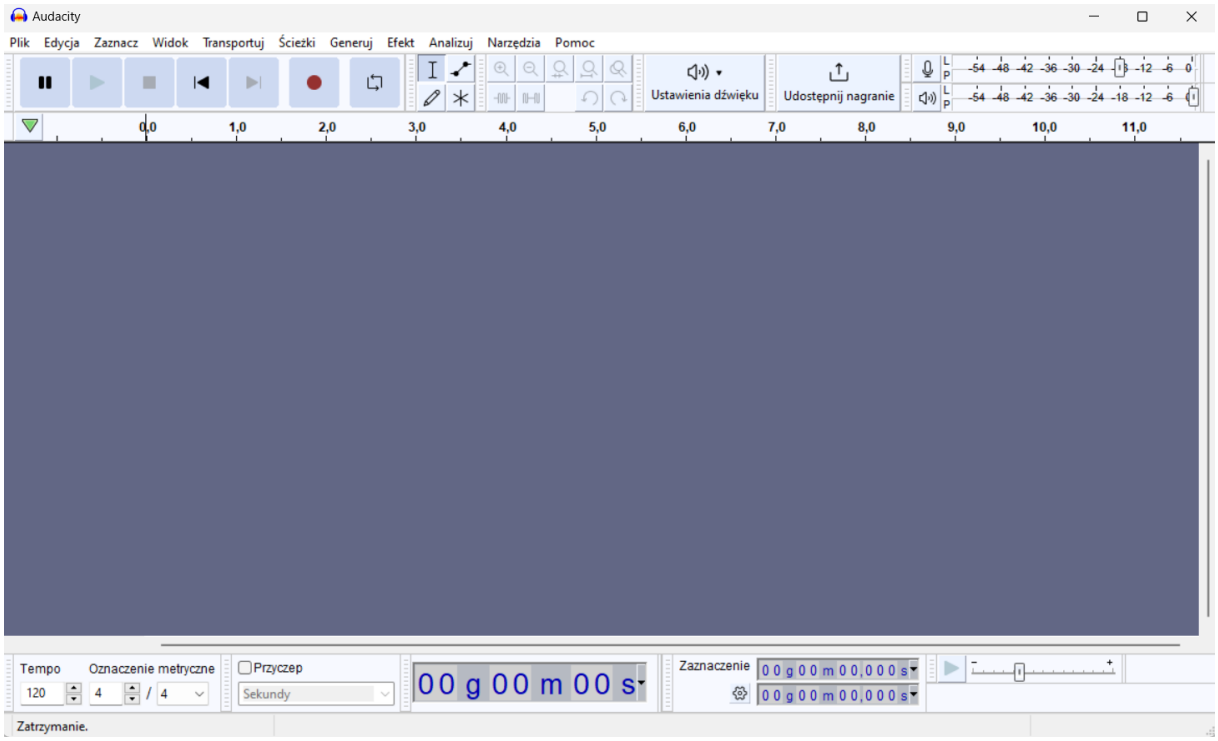
Rys 1.1 Widok główny strony Wolfram Alpha



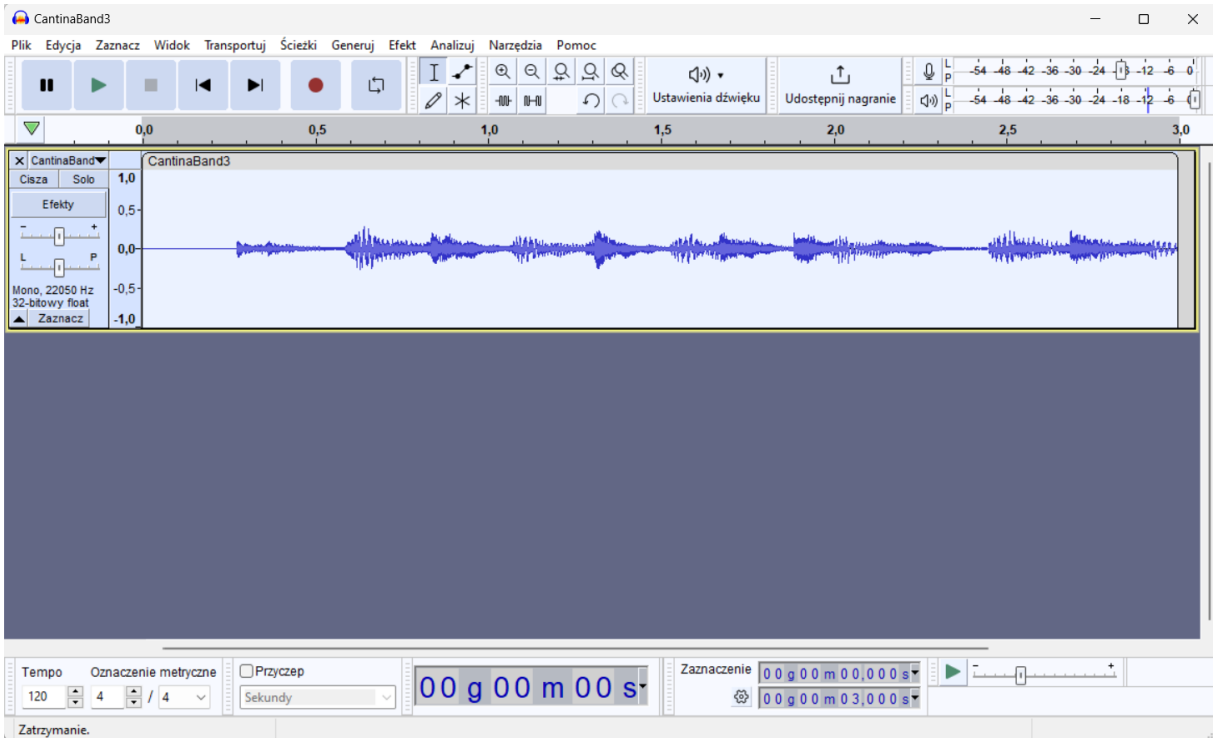
Rys 1.2 Przykładowy wykres modułu DFT

1.4.2. Audacity

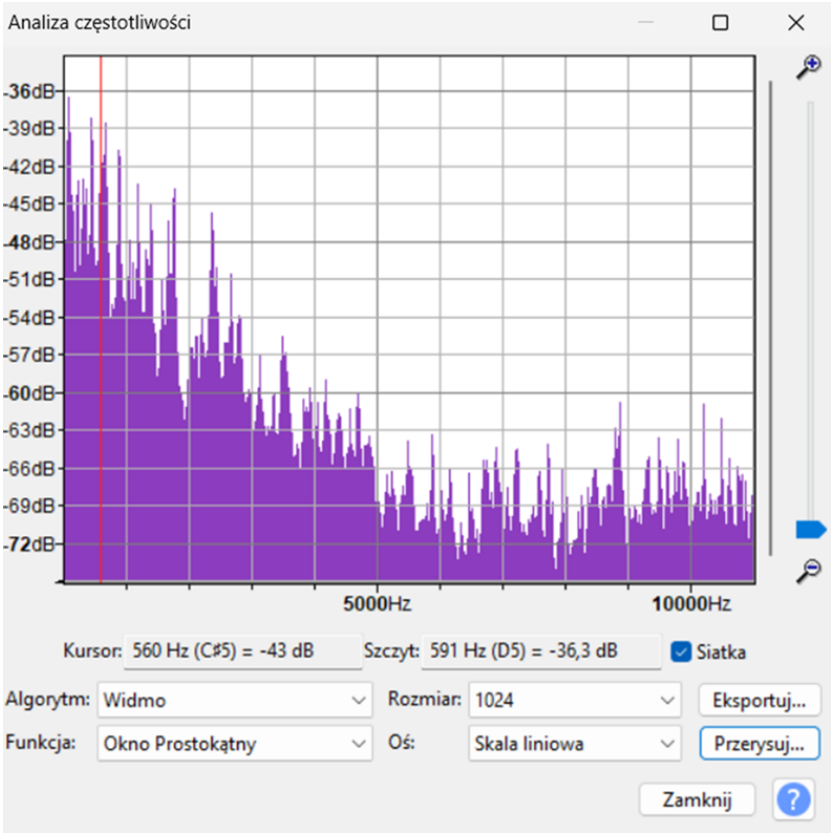
Aplikacja Audacity (rys. 1.3) to darmowe oprogramowanie służące do edycji oraz nagrywania dźwięku. Opracowany został przez zespół wolontariuszy w 2000 roku, stając się podstawowym narzędziem zarówno dla początkujących, ale także i dla profesjonalistów zajmujących się inżynierią akustyczną. Jest dostępny dla użytkowników systemów operacyjnych takich jak Window, macOS oraz Linux. Oferuje on szereg funkcji wykorzystywanych w manipulacji dźwiękiem, od prostej edycji (rys. 1.4) i nagrywania po bardziej złożone procesy takie jak analiza widmowa (rys. 1.5). Ponadto program oferuje szereg wtyczek i rozszerzeń, dzięki którym można dostosować oprogramowanie do indywidualnych wymagań [3].



Rys. 1.3 Główny panel aplikacji



Rys. 1.4 Przykładowa analiza sygnału dźwiękowego



Rys. 1.5 Przykładowa transformata sygnału dźwiękowego

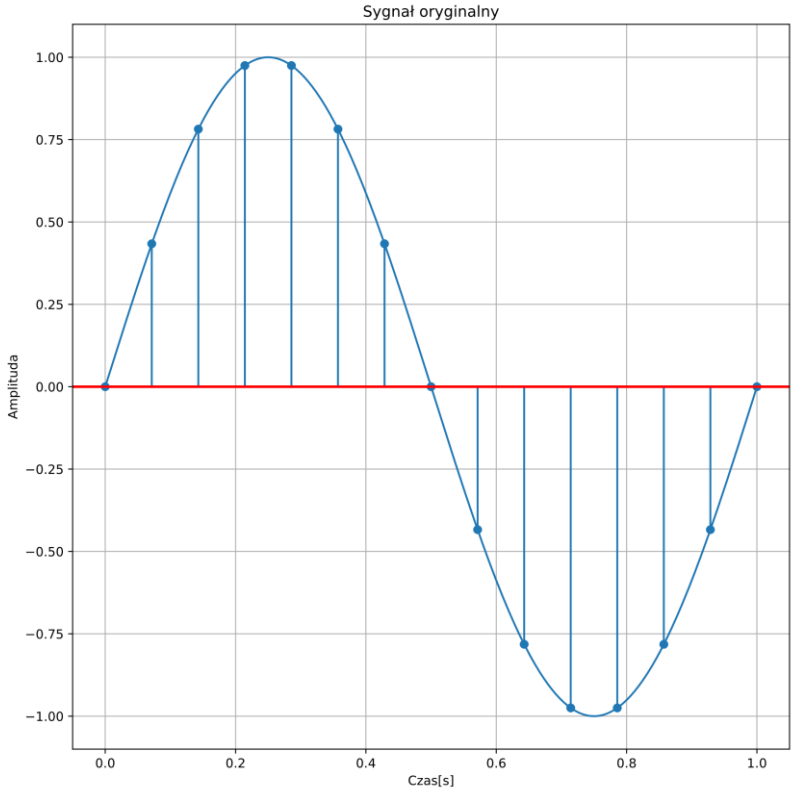
2. Analiza częstotliwościowa sygnałów jednowymiarowych

2.1. Sygnały jednowymiarowe

Sygnał jednowymiarowy to ciąg wartości, które zmieniają się w zależności od jednej zmiennej, którą w przypadku analizy częstotliwościowej, jest czas. Jednym z rodzajów takiego sygnału jest fala dźwiękowa. Dźwięk jest falą mechaniczną opisywaną jako sygnał jednowymiarowy za pomocą wielkości takich jak amplituda czy częstotliwość [7].

2.2. Próbkowanie sygnału

Próbkowanie to operacja, która polega na pobieraniu wartości sygnału w określonych momentach czasowych – tak zwanych próbek. Odstęp czasowy pomiędzy próbkami nazywany jest okresem próbkowania, zaś jego odwrotność częstotliwością próbkowania. Celem takiego procesu jest przekształcenie sygnału z dziedziny analogowej do postaci cyfrowej [1].



Rys. 2.1 Sygnał i jego dyskretna reprezentacja

2.2.1. Twierdzenie o próbkowaniu

Twierdzenie o próbkowaniu, znane także jako twierdzenie Nyquista-Shannona to jedna z fundamentalnych zasad w cyfrowym przetwarzaniu sygnałów wyrażona wzorem(2.1).

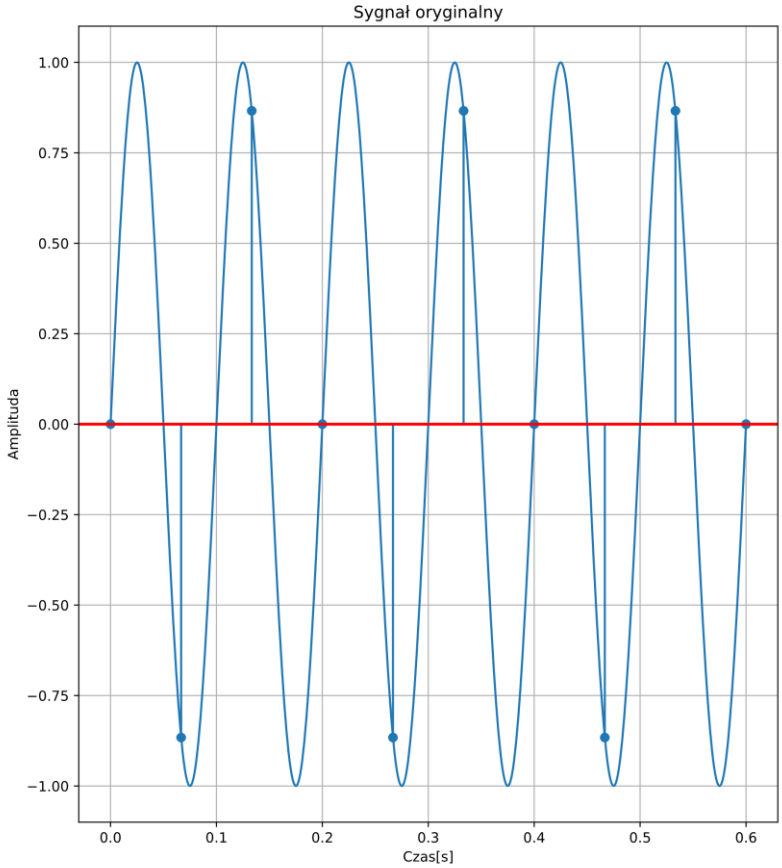
$$f_s \geq 2 * f_{max} \, , \tag{2.1}$$

gdzie:

f_s – częstotliwość próbkowania,

f_{max} – najwyższa częstotliwość w sygnale.

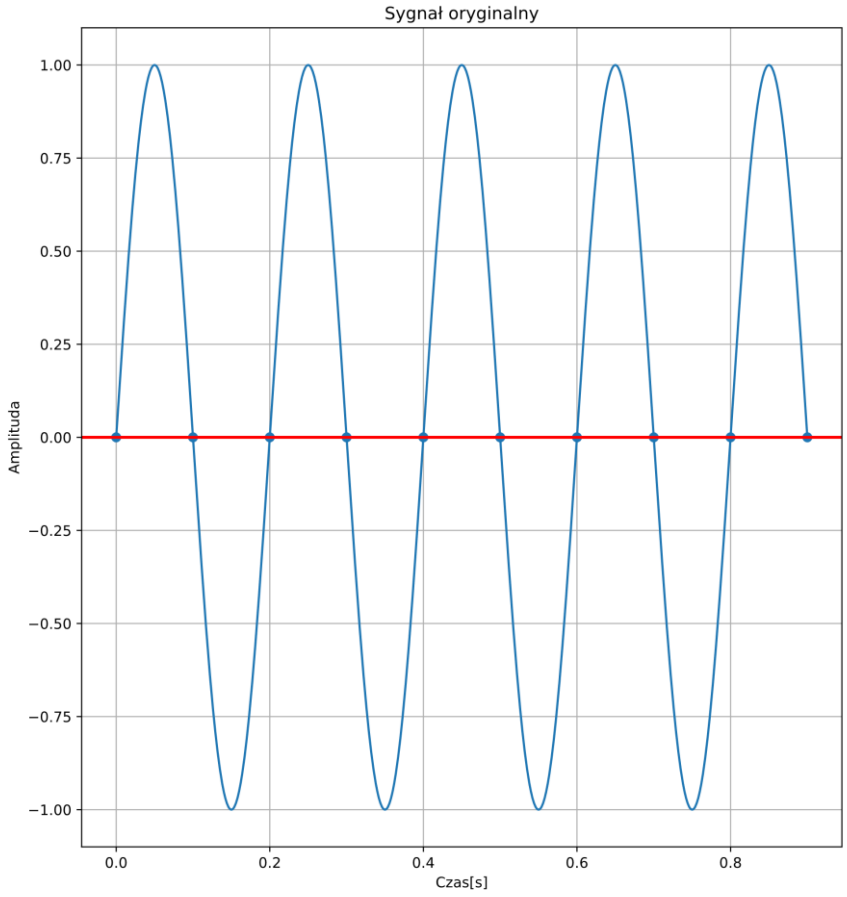
Twierdzenie to głosi, że aby poprawnie odtworzyć sygnał analogowy z jego próbek, próbkowanie należy przeprowadzić z częstotliwością próbkowania będącą przynajmniej dwukrotnością najwyższej częstotliwości harmonicznej sygnału analogowego. W przeciwnym wypadku wystąpi efekt aliasingu, który spowoduje, że częstotliwości wyższe niż częstotliwość Nyquista (połowa częstotliwości próbkowania) zostaną fałszywie zinterpretowane jako częstotliwości niższe niż w rzeczywistości, co prowadzi do zniekształceń sygnału [7].



Rys 2.2 Sygnał spróbkowany bez spełnienia twierdzenia o próbkowaniu

2.2.2. Próbkowanie krytyczne

Próbkowaniem krytycznym nazywa się przypadek, gdy próbkowanie wykonywane jest z częstotliwością równą dokładnie dwukrotności częstotliwości najwyższej harmonicznej sygnału. W tym wypadku, mimo spełnienia warunków wspomnianego twierdzenia, nie zawsze można poprawnie odtworzyć sygnał z jego próbek. Rozważając przypadek sygnału sinusoidalnego o częstotliwości równej 5Hz, którego próbki zostały pobrane z częstotliwością próbkowania równą 10Hz (rys. 2.3) można zauważyć, że widoczne są w miejscach, gdy wartość sygnału wynosi zero.[10]



Rys 2.3 Wizualizacja próbkowania krytycznego

W tym przypadku poprawna rekonstrukcja sygnału jest niemożliwa, pomimo spełnienia kryteriów twierdzenia o próbkowaniu.

2.3. Transformacja Fouriera

Ważnym narzędziem analizy częstotliwościowej sygnału, zwanej inaczej również analizą fourierowską, jest transformacja Fouriera. Jest to koncept matematyczny, który pozwala na przedstawienie dowolnej funkcji jako sumę składowych sinusoid o różnych częstotliwościach. Nazwa pochodzi od francuskiego matematyka oraz fizyka Jean Baptiste Joseph'a Fouriera, który pracując nad teorią przepływu ciepła odkrył, że każdy ciągły w czasie, okresowy sygnał (funkcja matematyczna), może zostać przedstawiony jako suma dobranych fal sinusoidalnych. Celem tej operacji jest przekształcenie funkcji ciągłej z dziedziny czasu do dziedziny częstotliwości, czego wynikiem jest transformata Fouriera [1]. Zdefiniowana jest przez równanie (2.2).

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{j2\pi ft}, \tag{2.2}$$

gdzie:

e – liczba Eulera,

j – jednostka urojona,

f – częstotliwość[Hz],

t – wartości chwil czasu[s].

$X(f)$ nazywamy zespolonym widmem Fouriera sygnału $x(t)$. Jego zadaniem jest przenoszenie informacji o częstotliwości zawartej w sygnale. Informuje nas o zawartości w sygnale zespolonej składowej harmonicznej zgodnej z równaniem (2.3).

$$e^{j2\pi ft} = cos(2\pi ft) + jsin(2\pi ft), \tag{2.3}$$

Widmo jest funkcją zespoloną, ma więc część rzeczywistą R ($(X(f))$ oraz część urojoną (jI), a także moduł oraz fazę$\left(\angle X(f)\right)$. Przedstawiają to wzory(2.4, 2.5, 2.6).

$$X(f) = R\left(X(f)\right) + jI\left(X(f)\right) = X(f) \vee e^{j\angle X(f)}, \tag{2.4}$$

$$|X(f)| = \sqrt{R\left(X(f)\right)^2 + I\left(X(f)\right)^2}, \tag{2.5}$$

$$\angle X(f) = arctg\left(\frac{I\left(X(f)\right)}{R\left(X(f)\right)}\right) \tag{2.6}$$

2.4. Dyskretna transformacja Fouriera

W przetwarzaniu komputerowym sygnały na ogół nie występują w postaci ciągłej, ale mamy do czynienia ze skończonym ciągiem próbek, które powstały w procesie próbkowania. Dla takich sygnałów zdefiniowana została dyskretna transformacja Fouriera. Jeżeli sygnał został spróbkowany zgodnie z twierdzeniem o próbkowaniu, to równanie dyskretnej transformaty Fouriera możemy zapisać w postaci (2.6).

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}}, \quad 0 \leq k \leq N-1 \tag{2.7}$$

gdzie:

$X(k)$ - wartość transformaty dla k-tej częstotliwości

$x(n)$ – wartość n-tej próbki sygnału

N – całkowita ilość próbek w sygnale

Znając widmo sygnału jesteśmy w stanie odnaleźć pierwotną sekwencje próbek sygnału. Proces odczytywania oryginalnego sygnału z widma reprezentowanego przez współczynniki transformaty Fouriera sygnału nazywamy syntezą. Wykorzystujemy w nim odwrotną transformację Fouriera IDFT (ang. *Inverse Discrete Fourier Transform*) wyrażoną równaniem(2.8) [1].

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{\frac{j2\pi kn}{N}}, 0 \leq n \leq N-1 \tag{2.8}$$

gdzie:

$X(k)$ - wartość transformaty dla k-tej częstotliwości

$x(n)$ – wartość n-tej próbki sygnału

N – całkowita ilość próbek w sygnale

2.4.1. Własności DFT

1. Liniowość – własność, zgodnie z którą transformata sumy dwóch sygnałów wejściowych jest równa sumie ich transformat, co opisuje zależność (2.9) [7].

$$\sum_{n=0}^{N-1} [ax(n) + by(n)] e^{\frac{-j2\pi kn}{N}} = a \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}} + b \sum_{n=0}^{N-1} y(n) e^{\frac{-j2\pi kn}{N}}, \tag{2.9}$$

gdzie:

a, b – dowolne liczby rzeczywiste

$x(n), y(n)$ – n-ta wartość danego sygnału

2. Niewrażliwość na przesunięcie – przesunięcie w dziedzinie czasu prowadzi do przesunięcia fazowego w dziedzinie częstotliwości. Rozważając sygnał dyskretny $x(n)$ oraz jego transformatę $X(k)$, jeżeli przesuniemy sygnał o pewną liczbę próbek, to jego transformata będzie przesunięta w fazie, co opisuje równanie(2.10).

$$\sum_{n=0}^{N-1} x(n-n_0)e^{\frac{-j2\pi kn}{N}} = \sum_{n=0}^{N-1} x(m)e^{\frac{-j2\pi k(m+n_0)}{N}} = e^{\frac{-j2\pi kn_0}{N}}X(k), m = n - n_0 \qquad (2.10)$$

gdzie:

n_0 – liczba próbek o którą przesuwamy sygnał x(n)

Dyskretna transformata Fouriera jest niewrażliwa na przesunięcie czasowe, co znaczy, że nie zmienia się informacja o amplitudzie widma, a wprowadza jedynie przesunięcie fazowe w wyniku [7].

2.5. Szybka transformacja Fouriera

Klasyczna metoda obliczania dyskretnej transformaty Fouriera polegająca na rozwiązywaniu równań liniowych w praktycznych zastosowaniach może okazać się niewystarczająca ze względu na czas, jaki potrzebuje komputer na przeprowadzenie obliczeń. Szybka transformacja Fouriera (ang. *Fast Fourier Transform*) jest metodą, która pozwala na znaczne skrócenie czasu obliczeń. Najczęściej stosowanym jest algorytm DIT (ang. *Decimation-in-Time*). Używany jest do obliczania dyskretnej transformaty Fouriera poprzez liczenie mniejszych transformat i wykorzystywanie właściwości funkcji zespolonych takich jak symetria oraz okresowość.[11] Proces algorytmu jest następujący:

1.Obliczanie współczynników transformaty X(k)

W przypadku, gdy liczba próbek sygnału N jest potęgą liczby 2, możliwe jest rozdzielenie sygnału dyskretnego x(n) na dwie sekwencje, tak aby jedna zawierała próbki o numerach parzystych, a druga nieparzystych(2.11).

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)W_N^{k2r} \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)W_N^{k(2r+1)} \qquad (2.11)$$

gdzie:

$X(k)$ – współczynnik transformaty o indeksie k

$x(2r)$ – ciąg parzystych próbek

$x(2r+1)$ – ciąg nieparzystych próbek

$W_N^{k2r} = e^{\frac{-j2\pi k2r}{N}}$

Stosując własności funkcji zespolonych, możemy to samo równanie zapisać w postaci(2.12).

$$X(k) = X_{0(k)} + W_N^k X_1(k) \tag{2.12}$$

gdzie:

$X_{0(k)}$ – transformata dla próbek parzystych

$X_{1(k)}$ – transformata dla próbek nieparzystych

Dla N = 8, możemy zdefiniować osiem równań wiążących transformatę $X(k)$ z transformatami obliczonymi dla połówek sygnału(2.13).

$$\begin{cases} X(0) = X_0(0) + W_N^0 X_1(0) \\ X(1) = X_0(1) + W_N^1 X_1(1) \\ X(2) = X_0(2) + W_N^2 X_1(2) \\ X(3) = X_0(3) + W_N^3 X_1(3) \\ X(4) = X_0(0) + W_N^4 X_1(0) \\ X(5) = X_0(1) + W_N^5 X_1(1) \\ X(6) = X_0(2) + W_N^6 X_1(2) \\ X(7) = X_0(3) + W_N^7 X_1(3) \end{cases} \tag{2.13}$$

Całkowita liczba operacji mnożenia jest równa $\frac{N^2}{2} + N$, natomiast dodawania $\frac{N^2}{2}$. Redukuje to liczbę potrzebnych operacji o prawie połowę.

2. Obliczanie współczynników transformaty $X_0(k)$ i $X_1(k)$

Jeżeli liczba próbek sygnału po rozdzieleniu nadal będzie potęgą liczby 2, transformatę można dalej podzielić na taką, która będzie miała $\frac{N}{4}$ punktów(2.14, 2.15).

$$X_0[k] = X_{00}\left[k_{\frac{N}{4}}\right] + W_{\frac{N}{2}}^k X_{01}\left[k_{\frac{N}{4}}\right] \tag{2.14}$$

$$X_1[k] = X_{10}\left[k_{\frac{N}{4}}\right] + W_{\frac{N}{2}}^k X_{11}\left[k_{\frac{N}{4}}\right] \tag{2.15}$$

gdzie:

$X_{00}, X_{01}, X_{10}, X_{11}$ – transformaty dla $\frac{N}{4}$ punktów.

Dla N = 8, możemy zdefiniować cztery równania wiążących transformatę obliczoną dla $\frac{N}{2}$ próbek i dla $\frac{N}{4}$ próbek(2.16, 2.17).

$$\begin{cases} X_0(0) = X_{00}(0) + W_{\frac{N}{2}}^{0} X_{01}(0) \\ X_0(1) = X_{00}(1) + W_{\frac{N}{2}}^{1} X_{01}(1) \\ X_0(2) = X_{00}(0) + W_{\frac{N}{2}}^{2} X_{01}(0) \\ X_0(3) = X_{00}(1) + W_{\frac{N}{2}}^{4} X_{01}(1) \end{cases} \quad (2.16)$$

$$\begin{cases} X_1(0) = X_{10}(0) + W_{\frac{N}{2}}^{0} X_{11}(0) \\ X_1(1) = X_{10}(1) + W_{\frac{N}{2}}^{1} X_{11}(1) \\ X_1(2) = X_{10}(0) + W_{\frac{N}{2}}^{2} X_{11}(0) \\ X_1(3) = X_{10}(1) + W_{\frac{N}{2}}^{4} X_{11}(1) \end{cases} \quad (2.17)$$

3.Dwupunktowa DFT

Dla N=8, $\frac{N}{4}$ punktowa DFT jest dwupunktową DFT daną przez równania(2.18, 2.19, 2.20).

$$X_{00}(k) \sum_{n=0}^{N-1} x_{00}(n) W_N^{kn} = \sum_{n=0}^{1} x_{00}(n) W_2^{kn}, 0 \le k \le 1 \quad (2.18)$$

$$X_{00}(0) = x_{00}(0) + x_{00}(1) = x(0) + x(4) \quad (2.19)$$

$$X_{00}(1) = x_{00}(0) + x_{00}(1) = x(0) - x(4) \quad (2.20)$$

Liczba takich etapów jest zależna od liczby próbek i jest równa $log_2 N$. Wpływ algorytmu na liczbę operacji oraz szybkość działania został przedstawiony w tabeli 2.1. Dla dużych wartości N różnica jest istotna w porównaniu do małych i sięga redukcji operacji o ponad 90%.[1]

Tabela 2.1. Porównanie efektywności przed i po zastosowaniu algorytmu DIT[1]

Liczba próbek N	Liczba operacji dodawania/mnożenia(N^2)	Liczba operacji po zastosowaniu algorytmu($N log_2 N$)	Redukcja[%]
8	64	24	62.5
64	4096	384	90.6
512	262144	4608	98.2

2.6. Dyskretna transformacja kosinusowa

Podobnie jak w przypadku transformacji Fouriera, DCT (ang. *Discrete Cosine Transform*) ma za zadanie przekształcenie sygnału z dziedziny czasu do dziedziny częstotliwości. Jednak w odróżnieniu od DFT, które dekomponuje sygnał na sumę funkcji sinus oraz cosinus – DCT wykorzystuje jedynie szereg funkcji kosinusowych. Ta właściwość powoduje, że ma ona jedynie współczynniki rzeczywiste [7]. Istnieje kilka wariantów DCT, dla których można zastosować przekształcenie odwrotne – odwrotną transformację kosinusową IDCT (ang. *Inverse Discrete Cosine Transform*). Ich równania zostały przedstawione w tabeli 2.2.

Tabela 2.2. Rodzaje dyskretnej transformacji kosinusowej[1]

Rodzaj DCT	$\boldsymbol{X(k)}$	**x(n)**
DCT-I	$\sqrt{\frac{2}{N}} * c(k) * \sum_{n=0}^{N} c(n) * x(n) * cos\left[\frac{\pi k n}{N}\right]$ $k, n = 0,1,2, \dots, N$	$\sqrt{\frac{2}{N}} * c(n) * \sum_{k=0}^{N} c(k) * X(k) * cos\left[\frac{\pi k n}{N}\right]$ $k, n = 0,1,2, \dots, N$
DCT-II	$\sqrt{\frac{2}{N}} * c(k) \sum_{n=0}^{N-1} x(n) * cos\left[\frac{(2n+1)k\pi}{2N}\right]$ $k, n = 0,1,2, \dots, N-1$	$\sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} c(k) * X(k) * cos\left[\frac{(2n+1)k\pi}{2N}\right]$ $k, n = 0,1,2, \dots, N-1$
DCT-III	$\sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} c(n) * x(n) * cos\left[\frac{(2k+1)n\pi}{2N}\right]$ $k, n = 0,1,2, \dots, N-1$	$\sqrt{\frac{2}{N}} * c(n) \sum_{n=0}^{N-1} X(k) * cos\left[\frac{(2k+1)n\pi}{2N}\right]$ $k, n = 0,1,2, \dots, N-1$

DCT-IV	$\sqrt{\frac{2}{N}}\sum_{n=0}^{N-1} x(n)$ $* cos\left[\frac{(2n+1)(2k+1)\pi}{4N}\right]$ $k,n = 0,1,2,\dots,N-1$	$\sqrt{\frac{2}{N}}\sum_{n=0}^{N-1} X(k)$ $* cos\left[\frac{(2n+1)(2k+1)\pi}{4N}\right]$ $k,n = 0,1,2,\dots,N-1$

gdzie:

$$c(i) = \begin{cases} \frac{1}{\sqrt{2}} dlai = 0 lubi = N \\ 1 dla 0 < i < N \end{cases}$$

Transformacja typu DCT-II jest stosowana głównie w kompresji obrazów nieruchomych i ruchomych, czyli w standardach odpowiednio JPEG oraz MPEG. Natomiast transformacja DCT-III znalazła zastosowanie w modulowanych zespołach filtrów standardu MPEG audio [1].

2.7. Przeciek widma

Podczas próbkowania sygnału otrzymamy określoną liczbę próbek, co prowadzi do uzyskania również określonej liczby wartości widma w zakresie od 0 do f_s – częstotliwości próbkowania. Odstęp pomiędzy dwoma wartościami widma nazywamy rozdzielczością częstotliwościową transformaty(2.21).

$$d_f = \frac{f_s}{N}, \tag{2.21}$$

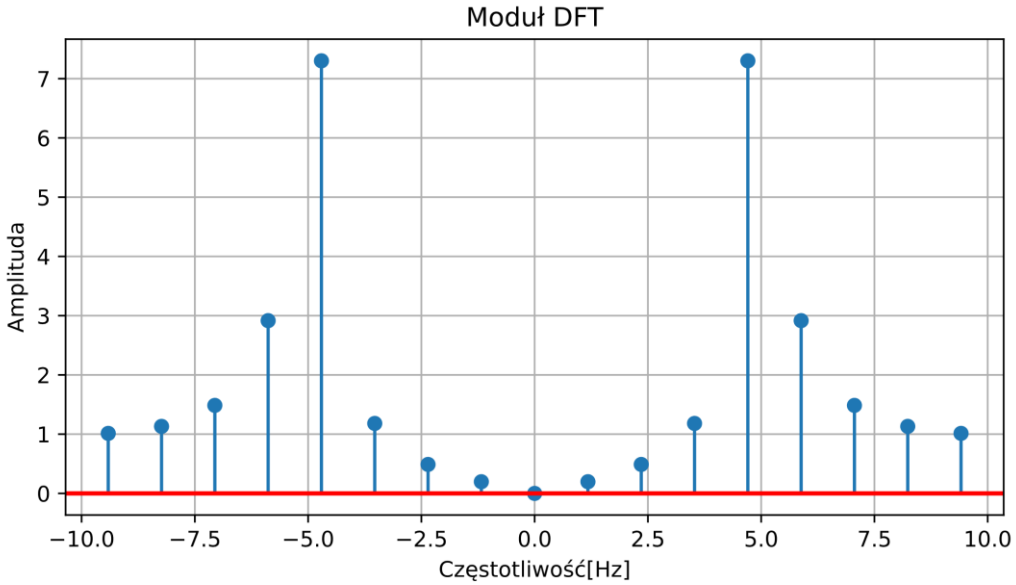
gdzie:

d_f – rozdzielczość widma

f_s – częstotliwość próbkowania

N – liczba wartości widma

W skutek skończonej rozdzielczości na ogół nie dostaniemy dokładnej wartości prążka widma. Zawsze będzie ona za mała lub za duża. Czynnikiem decydującym o tej różnicy jest liczba próbek sygnału wejściowego.



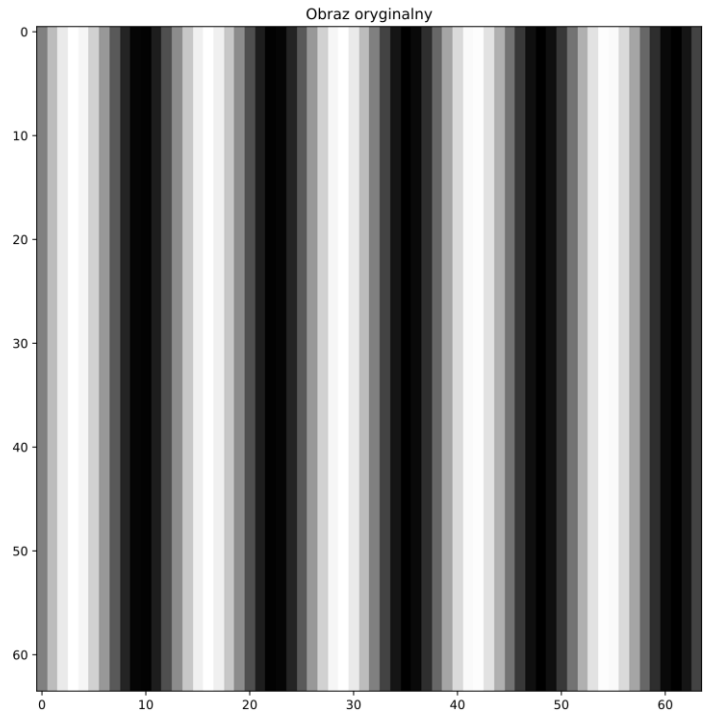
Rys 2.3 Wizualizacja zjawiska przecieku widma

Dyskretna transformata Fouriera obliczana jest dla sygnałów okresowych. W przypadku, gdy obliczeniom poddawana jest część sygnału, która stanowi jego okres, widmo zostanie zniekształcone. Zjawisko to nazywamy przeciekiem widma (rys 2.3). Jeżeli całkowita liczba N próbek sygnału obejmie cały okres sygnału, wówczas jako widmo na wykresie ukaże nam się jeden ostry prążek. Jeśli natomiast sygnał zostanie przerwany pomiędzy okresami, to niektóre częstotliwości „wycieką” do okolicznych składowych, co spowoduje zniekształcenia w dziedzinie częstotliwości [8].

3. Analiza częstotliwościowa sygnałów dwuwymiarowych

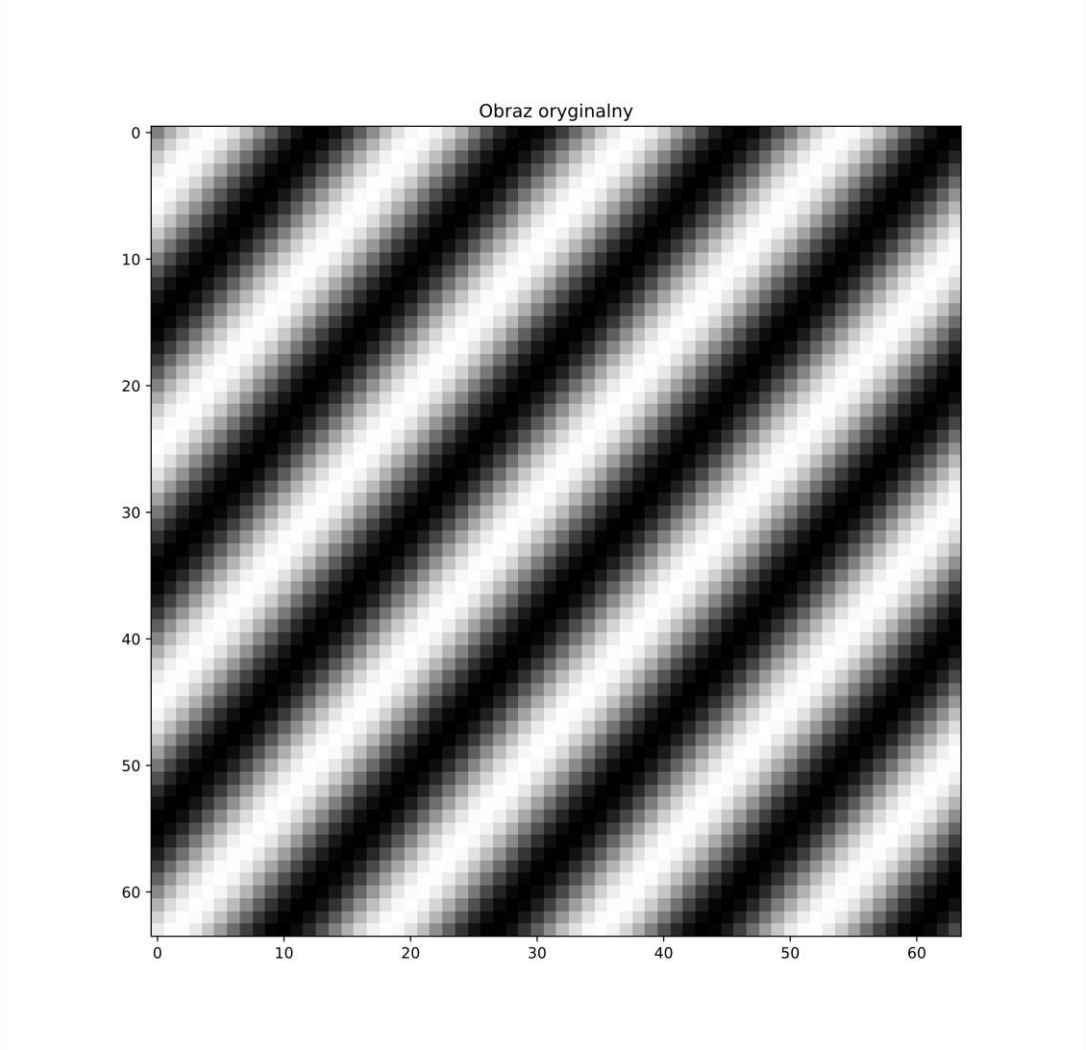
3.1. Sygnały dwuwymiarowe

Sygnał dwuwymiarowy to taki, którego, w przeciwieństwie do sygnału jednowymiarowego, wartość zależy od dwóch zmiennych. Przykładem takiego sygnału jest obraz, który reprezentuje wartość piksela w dwuwymiarowej siatce. Pierwszym wymiarem obrazu jest położenie wzdłuż osi poziomej (szerokość), drugi zaś odpowiada za położenie wzdłuż osi pionowej (wysokość). Wartość piksela z kolei reprezentuje kolor (jasność) obrazu w danym punkcie. Analiza częstotliwościowa sygnałów dwuwymiarowych znajduje szerokie zastosowanie w kompresji obrazów. Sygnały dwuwymiarowe, które następnie są wyświetlane w postaci obrazu można tworzyć na podstawie funkcji sinus. Jednym z parametrów, które mają wpływ na taki sygnał jest częstotliwość. Odpowiada ona za to, ile fal mieści się w tym samym obszarze przestrzeni (im wyższa częstotliwość, tym więcej fali zmieści się w danym obszarze). W odróżnieniu od sygnałów jednowymiarowych, w tym wypadku odnosimy się do częstotliwości przestrzennej [5]. Przykładowy obraz opisujący wpływ częstotliwości przestrzennej został przedstawiony na rysunku 3.1.



Rys 3.1 Sygnał dwuwymiarowy o częstotliwości przestrzennej 5Hz

Kolejnym istotnym parametrem jest faza sygnału dwuwymiarowego. Odpowiada ona za obrót pasków widocznych na obrazie.



Rys 3.2 Sygnał dwuwymiarowy o częstotliwości przestrzennej 5Hz i fazie 40°

3.2. Dwuwymiarowa dyskretna transformacja Fouriera

Podobnie jak w przypadku sygnałów jednowymiarowych, sygnały dwuwymiarowe, takie jak obrazy, możemy poddać różnym przekształceniom. Jednym z nich jest dwuwymiarowa transformacja Fouriera (2D DFT), zdefiniowana równaniem (3.1).

$$X(k,l) = \sum_{m=0}^{M-1}\left(\sum_{n=0}^{N-1} x(m,n) * e^{-\frac{j2\pi nl}{N}}\right) * e^{-\frac{j2\pi mk}{M}}, \tag{3.1}$$

$$0 \leq m,k \leq M-1; 0 \leq n,l \leq N-1$$

gdzie:

$X(k,l)$ – wartości transformaty o indeksach k, l,

$x(m,n)$ – wartości sygnału dwuwymiarowego o indeksach m, n,

k, l – indeksy dyskretnych częstotliwości,

m, n – indeksy dyskretnego położenia przestrzennego.

Dwuwymiarowa DFT przekształca obraz z dziedziny przestrzennej do dziedziny częstotliwości, co umożliwia analizę zawartości częstotliwości obrazu. Do implementacji wykorzystywany jest algorytm szybkiej transformacji Fouriera FFT (ang. *Fast Fourier Transform*) w celu efektywnego obliczania transformaty dla dużych obrazów.[9]

Odwrotna dwuwymiarowa dyskretna transformata Fouriera zdefiniowana jest równaniem(3.2).

$$x(m,n) = \frac{1}{N}\sum_{l=0}^{N-1}\left(\frac{1}{M}\sum_{k=0}^{M-1} X(k,l) * e^{\frac{j2\pi mk}{M}}\right) * e^{\frac{j2\pi nl}{N}} \tag{3.2}$$

$$0 \le m,k \le M-1; 0 \le n,l \le N-1$$

Zarówno ciągłe jak i dyskretne dwuwymiarowe transformacje Fouriera sprowadzają się do wykonania dwóch serii jednowymiarowych transformacji, gdzie druga seria działa na wyniku pierwszej, lecz wzdłuż innej osi. Na początek obliczane są transformaty wierszy, a następnie kolumn. Kolejność może być zmieniona.[1]

3.3. Dwuwymiarowa dyskretna transformacja kosinusowa

Dwuwymiarowa transformacja kosinusowa to technika przetwarzania sygnałów szeroko stosowana w analizie oraz kompresji obrazów i plików wideo. Tak jak jej jednowymiarowy odpowiednik, wykorzystuje funkcje cosinus w celu przekształcenia sygnału do dziedziny częstotliwości. Szczególne zastosowanie znajduje w kompresji obrazów w standardzie JPEG (ang. *Joint Photographic Experts Group*). Tak samo jak w przypadku jednowymiarowej transformacji, istnieją różne rodzaje 2D DCT, sprowadzające do wykonania dwóch serii jednowymiarowych transformacji. W praktyce stosowana jest transformata DCT-II, zdefiniowana wzorem (3.3) oraz jej proces odwrotny – odwrotna dwuwymiarowa transformacja kosinusowa (3.4).[1]

$$X(k,l) = \sum_{m=0}^{M-1}\left(\sum_{n=0}^{N-1} x(m,n) * \beta(l) * \cos\left(\frac{\pi l}{N}\left(n+\frac{1}{2}\right)\right) * \alpha(k)C * \cos\left(\frac{\pi k}{M}\left(m+\frac{1}{2}\right)\right)\right) \tag{3.3}$$

$$x(m,n) = \sum_{l=0}^{N-1} \sum_{k=0}^{M-1} X(k,l) * \alpha(k) * cos\left(\frac{\pi k}{M}\left(m+\frac{1}{2}\right)\right) * \beta(l) * cos\left(\frac{\pi l}{N}\left(n+\frac{1}{2}\right)\right) \tag{3.4}$$

gdzie:

$X(k,l)$ – wartości transformaty o indeksach k, l,

$x(m,n)$ – wartości sygnału dwuwymiarowego o indeksach m, n,

k, l – indeksy dyskretnych częstotliwości,

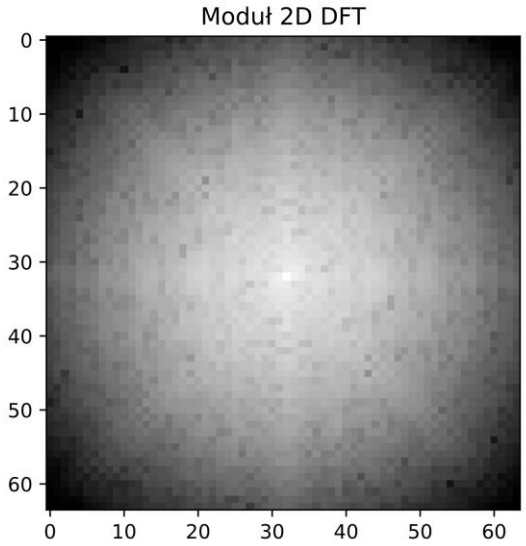
m, n – indeksy dyskretnego położenia przestrzennego.

$$\alpha(k) = \begin{cases} \sqrt{\frac{1}{M}}, k = 0 \\ \sqrt{\frac{2}{M}}, k = 1 \dots M-1 \end{cases} \qquad \beta(l) = \begin{cases} \sqrt{\frac{1}{N}}, l = 0 \\ \sqrt{\frac{2}{N}}, l = 1 \dots N-1 \end{cases}$$

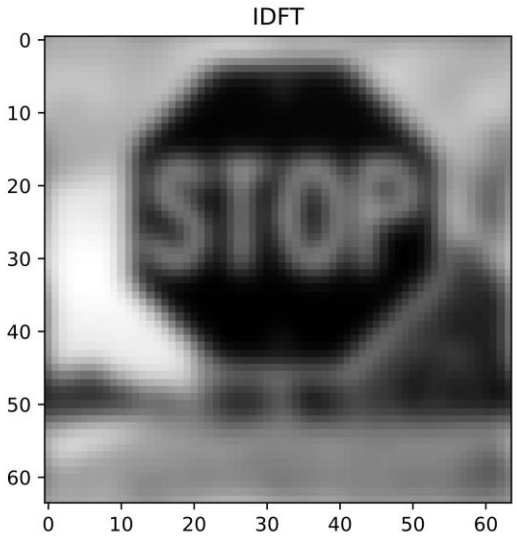
Największą zaletą w porównaniu z dwuwymiarową transformacją Fouriera jest fakt, że funkcje bazowe 2D DCT nie zawierają wartości zespolonych, a jedynie rzeczywiste. Ma to znaczenie w przypadku stopnia komplikacji obliczeń.

3.4. Filtr Gaussa

W celu przetwarzania sygnałów dwuwymiarowych, takich jak obrazy, stosowane są filtry dwuwymiarowe. Na początku stosujemy wybraną transformatę, następnie filtr jest nakładany w dziedzinie częstotliwości, co po zastosowaniu odwrotnej transformaty pozwoli na odczytanie obrazu po odfiltrowaniu pewnych składowych. Najpopularniejszym filtrem stosowanym dla dwuwymiarowej transformacji Fouriera jest filtr Gaussa (rys 3.3 oraz 3.4).



Rys 3.3 Dwuwymiarowa transformata Fouriera, filtracja Gaussa



Rys 3.4 Rozmycie Gaussa

Filtracja polega na wymnożeniu wartości transformaty z maską Gaussa, wyrażoną wzorem(3.5).[13]

$$G = \frac{1}{2\pi f^2} e^{\frac{-(x^2+y^2)}{2f^2}} \tag{3.5}$$

gdzie:

f – częstotliwość graniczna

x – pozioma współrzędna środka obrazu

y – pionowa współrzędna środka obrazu

4. Wymagania projektowe

4.1. Zasady SOLID

W miarę rozwoju aplikacji, stopień komplikacji kodu zwiększa się. Aby w procesie rozwoju aplikacji zachować spójną strukturę oraz możliwość łatwej modyfikacji, należy stosować dobre praktyki programowania obiektowego. Jedną z takich praktyk są zasady SOLID [4].

4.1.1. Zasada pojedynczej odpowiedzialności

Zasada pojedynczej odpowiedzialności mówi o tym, że każda klasa w kodzie programu powinna być odpowiedzialna za realizacje jednego, konkretnego zadania lub funkcji. Zastosowanie tej zasady prowadzi do rozdzielenia kodu na mniejsze, bardziej zrozumiałe części, co ułatwia rozwój w przyszłości kosztem liczby klas w programie [4].

4.1.2. Zasada otwarte/zamknięte

Zasada otwarte/zamknięte oznacza, że klasy powinny być otwarte na rozszerzenia, ale zamknięte na modyfikacje. Dodanie nowych funkcjonalności powinno być możliwe bez zmieniania dotychczasowego kodu, ale poprzez dodawanie nowych fragmentów. W implementacji tej zasady często używany jest polimorfizm oraz dziedziczenie, co umożliwia elastyczne dostosowywanie funkcji bez ingerencji w istniejący kod. Jest to jeden z fundamentów programowania obiektowego, który umożliwia rozbudowę klasy w przyszłości [4].

4.1.3. Zasada podstawienia Liskov

Nazwa tej zasady pochodzi od amerykańskiej programistki Barbary Liskov. Polega ona na tym, że musi zostać zachowana zgodność interfejsu oraz wszystkich metod w klasie, aby można było zastąpić obiekt jednego typu innym obiektem pochodnym bez wpływu na zachowanie programu.[4]

4.1.4. Zasada segregacji interfejsów

Zasada segregacji interfejsów to najprostsza w zrozumieniu zasada, według której nie należy tworzyć interfejsów z niepotrzebnymi metodami – powinny być one jak najmniejsze i spełniać konkretne funkcje. Rozbicie interfejsów na mniejsze części, które spełniają określone funkcje, sprawia, że będą one łatwiejsze w zrozumieniu [4].

4.1.5. Zasada odwrócenia zależności

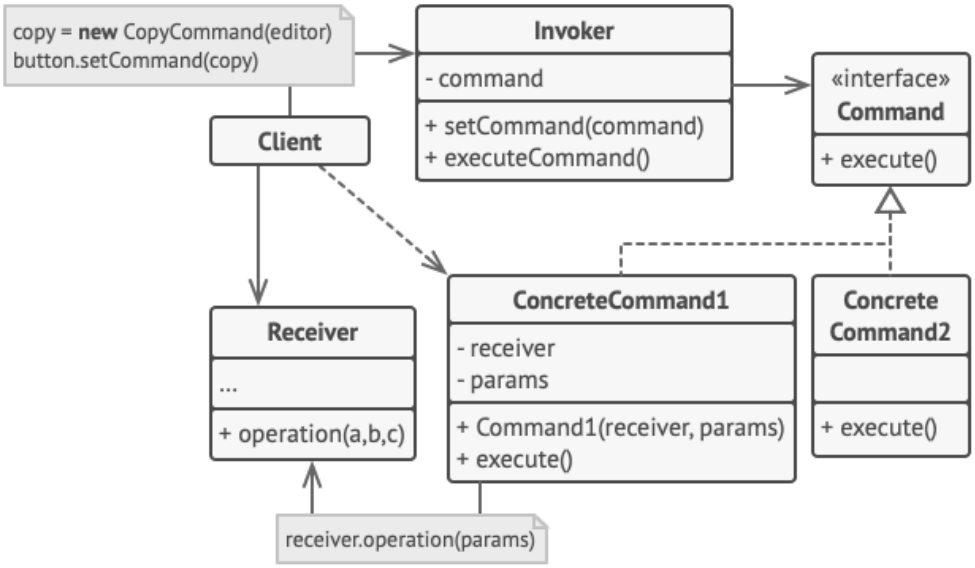
Zasada odwrócenia zależności polega na tym, że wysokopoziomowe moduły nie powinny zależeć bezpośrednio od modułów niskopoziomowych. Obydwa powinny być zależne od interfejsów lub klas abstrakcyjnych [4].

4.2. Wzorzec projektowy Command

Polecenie (ang. *Command*) jest wzorcem projektowym z kategorii wzorców behawioralnych.

Polega on na zamianie żądania w obiekt, który zawiera wszystkie potrzebne informacje na jego temat. Umożliwia to ich przekazywanie jako argumentów metody, opóźnianie lub umieszczanie w kolejce. Jest szczególnie przydatny, gdy chcemy uniezależnić wywołanie operacji od jej wykonania. Swoje zastosowanie znalazł w aplikacjach zawierających interfejs użytkownika (np. obsługa przycisków). Pomaga zwiększyć elastyczność oraz łatwość utrzymania kodu [6]. Struktura (rys. 4.1) wzorca Command składa się z czterech elementów:

- Polecenie (Command) – jego zadaniem jest reprezentacja konkretnego żądania jako obiekt, który zawiera wszystkie niezbędne informacje do jego wykonania
- Odbiorca (Receiver) – odpowiada za realizację konkretnej operacji powiązanej z danym poleceniem
- Wywoływacz (Invoker) – inicjuje żądania oraz przekazuje je do odpowiednich obiektów poleceń do wykonania
- Klient (Client) – tworzy obiekty poleceń oraz przekazuje je do wywoływacza



Rys 4.1 Struktura wzorca projektowego Command [6]

5. Wykorzystane narzędzia

5.1. Język programowania Python

Język programowania Python jest jednym z najbardziej wszechstronnych oraz powszechnie używanych narzędzi programistycznych. Prostota, czytelność kodu oraz ogromna społeczność programistyczna sprawiają, że jest popularnym wyborem wśród wielu projektów. Jest on wysokopoziomowym i wieloparadygmatowym językiem programowania. Posiada bogatą kolekcję bibliotek, które znajdują zastosowanie niemal w każdej dziedzinie. Jednym z kluczowych atutów języka Python jest dynamiczne typowanie, co oznacza, że nie musimy deklarować typu zmiennej, sprawiając, że kod staje się bardziej elastyczny. Niewątpliwie jego zaletą jest również automatyczne zarządzanie pamięcią, które zwalnia programistę z konieczności manualnego jej alokowania, co zwiększa wygodę oraz skraca czas poświęcony rozwojowi aplikacji.[12]

5.2. Biblioteki

5.2.1. NumPy

NumPy(ang. *Numerical Python*) to biblioteka dla języka Python, zaprojektowana głównie do przeprowadzania obliczeń numerycznych. Jest ona jedną z najpopularniejszych biblitotek ze względu na jej wszechstronne zastosowanie w pracy z wielowymiarowymi tablicami oraz macierzami, które znajdują swoje zastosowanie m.in. w inżynierii.

5.2.2. SciPy

SciPy(ang. *Scientific Python*) dostarcza zestawu zaawansowanych narzędzi i funkcji do przeprowadzania przede wszystkim obliczeń naukowych. Oferowany przez SciPy zestaw funkcji znajduje zastosowanie w wielu dziedzinach, takich jak analiza danych i przetwarzanie sygnałów.

5.2.3. Matplotlib

Nazwa biblioteki Matplotlib pochodzi od słów „Matrix”, co oznacza macierze, na których operuje oraz od słowa „Plot” oznaczającego wykres. Przeznaczona do tworzenia wykresów oraz wizualizacji danych. W jej ofercie znajdziemy do wyboru wykresy m.in. liniowe oraz punktowe. Jej największą zaletą jest niewątpliwie duża kontrola nad wyglądem wyświetlanych danych. Umożliwia również zintegrowanie okna z wykresem z różnymi bibliotekami służącymi do projektowania interfejsów graficznych, takich jak PyQt5.

5.2.4. PyQt5

Jako narzędzie służące do zaprojektowania graficznego interfejsu użytkownika wykorzystana została biblioteka PyQt5. Jest to zbiór modułów umożliwiających integrację interfejsu graficznego z aplikacjami w języku Python. Oparta jest ona na technologii Qt wykorzystywanej do projektowania aplikacji desktopowych oraz mobilnych. Swoją popularność zawdzięcza wszechstronności oraz prostocie użytkowania. Niewątpliwym atutem jest integracja z narzędziem Qt Designer, które umożliwia w szybki sposób zaprojektowanie całego interfejsu w sposób graficzny, bez konieczności tworzenia kodu w sposób manualny. Z programu, możemy wyekstrahować plik o rozszerzeniu *.ui*, który następnie możemy wczytać bezpośrednio w kodzie z wykorzystaniem dołączonej funkcji *loadUi*.

5.2.5. OpenCV

Biblioteka OpenCV(ang. *Open Source Computer Vision*) to biblioteka służąca do przetwarzania obrazów i ich analizy za pomocą języka Python. Umożliwia ona odczytanie danych z plików obrazowych o rozszerzeniu m.in. JPEG i PNG. Oferuje również funkcje takie jak zmiana rozmiaru obrazu, obrót, korekcja jasności lub kontrastu.

6. Implementacja

6.1. Plan realizacji

Kroki, które zostały zaplanowane obejmują analizę wymagań, projektowanie, implementację oraz testowanie.

Podczas analizy wymagań wzięto pod uwagę założenia, które ma spełniać aplikacja w celu opracowania jak najbardziej optymalnego rozwiązania opisywanego problemu.

Projektowanie polegało na zaplanowaniu, jakie elementy interfejsu graficznego będą niezbędne oraz na wstępnym planie architektury aplikacji, w tym użytych bibliotekach oraz potrzebnych klasach.

Implementację rozpoczęto od stworzenia pierwszej kluczowej funkcjonalności aplikacji – wyboru sygnału oraz rysowania wykresu. Następnie utworzono oddzielne okna do analizy transformacji DFT oraz DCT. Po przeprowadzeniu testów kolejnym krokiem było stworzenie zakładki dla sygnałów 2D, w której zawarte są przyciski do otworzenia okien analizy transformat dwuwymiarowych.

Po stwierdzeniu poprawnego działania opisanych funkcji, dodano funkcjonalność projektowania filtrów dla danej transformaty. Aplikacja była regularnie testowana na etapach implementacji poszczególnych funkcji. Informacje szczegółowe zostały opisane w rozdziale 8.

6.2. Założenia aplikacji

Aplikacja ma służyć jako narzędzie do analizy sygnałów jednowymiarowych oraz dwuwymiarowych pod kątem zawartości częstotliwości, wykorzystując transformację DFT oraz DCT. Przeznaczona jest dla uczniów i studentów w celu wizualizacji zjawisk dotyczących analizy częstotliwościowej.

Na etapie projektowania przyjęto, że aplikacja ma udostępniać następujące funkcjonalności:

- Wybór sygnału do analizy – funkcja matematyczna, plik dźwiękowy lub obraz
- Możliwość obserwacji wpływu parametrów sygnału na wynik analizy częstotliwościowej
- Zastosowanie filtrów w dziedzinie częstotliwości dla wybranych transformat

6.3. Wymagania aplikacji

Do uruchomienia aplikacji wymagany jest dowolny komputer z zainstalowanym środowiskiem języka Python. Zainstalowane muszą być również biblioteki opisane w rozdziale 5.

6.4. Wybór sygnału

Podstawą działania aplikacji jest wybór sygnału, który zostanie poddany analizie częstotliwościowej. Parametry, z których obliczana jest funkcja pobierane są z interfejsu graficznego poprzez klasy znajdujące się w pliku `Controllers.py`. Do każdego okna została przypisana osobna klasa zawierająca zestaw metod odpowiedzialnych za odczytanie informacji z konkretnego elementu interfejsu.

6.4.1. Sygnał jednowymiarowy

W przypadku sygnału jednowymiarowego, za stworzenie danej harmonicznej odpowiedzialna jest klasa `Sine`, która implementuje interfejs `Signal`. Składa się ona z metody umożliwiającej stworzenie funkcji sinus, przyjmując jako argument wektor czasu (oś OX na wykresie) oraz takiej, która pozwala na konwersje informacji o sygnale na tekst, który następnie jest wyświetlany w polu tekstowym interfejsu graficznego. Za obsługę stworzonych funkcji odpowiedzialna jest klasa `SineHandler`, która implementuje interfejs `SignalsHandler`. Zawiera on pola przechowujące informacje o istniejących sygnałach w postaci listy oraz metody pozwalające na dodanie wybranego sygnału do listy i wygenerowanie z nich fali sinusoidalnej.

6.4.2. Sygnał dwuwymiarowy

Klasa `Sine2D` odpowiadająca za sygnał dwuwymiarowy implementuje ten sam interfejs co w przypadku sygnału jednowymiarowego. Za pomocą metody `get_wave` zwracana obliczony sygnał.

Metoda jako argument przyjmuje rozmiar obrazu w pikselach. W celu stworzenia siatki za pomocą funkcji `meshgrid` z pakietu NumPy tworzymy listę wartości dopasowanych do wartości parametru `size`. Pozwala nam to na stworzenie dwuwymiarowej funkcji sinusoidalnej wyrażonej wzorem (6.1).

$$y(X,Y) = sin\,[2\pi \frac{X*cos(\varphi)*f+Y*sin(\varphi)*f}{size}] \qquad (6.1)$$

gdzie:

X – poziome współrzędne siatki,

Y – pionowe współrzędne siatki,

φ – kąt określający orientację sinusoidy,

f – częstotliwość przestrzenna,

$size$ – rozmiar siatki w pikselach.

6.5. Wczytywanie plików

Ważną funkcjonalnością aplikacji jest możliwość dokonania analizy na sygnałach odczytanych z plików. W przypadku sygnałów jednowymiarowych będzie to plik dźwiękowy o formacie wav, a dwuwymiarowych – obraz o formacie jpg. Funkcje te zostały zaimplementowane w pliku `FileHandler.py`, w którym znajduje się interfejs `FileHandler`. Jako atrybut przyjmuje on ścieżkę do wybranego pliku pobraną z interfejsu graficznego. Klasy `WavFileHandler` oraz `JpgFileHandler` implementują wspomniany interfejs nadpisując metodę `generate_data`, która zwraca dane odczytane z pliku. Sygnał dźwiękowy obsługiwany jest przez metodę `read` z modułu scipy.io. Zwraca ona kolejno częstotliwość próbkowania oraz wektor danych do wykreślenia wykresu. Sygnał dwuwymiarowy w postaci obrazu odczytywany jest za pomocą biblioteki `cv2` z wykorzystaniem metody `imread`, a następnie w celu ustandaryzowania wyników obliczeń zostaje przeskalowany do rozmiaru 64x64px na potrzeby aplikacji za pomocą metody `resize`.

6.6. Algorytmy

6.6.1. Transformacje

Algorytmy do obliczeń transformat zarówno jednowymiarowych jak i dwuwymiarowych zostały zaimplementowane z wykorzystaniem biblioteki SciPy. W pliku `TransformAnalyzer.py` znajdują się klasy odpowiedzialne za zastosowanie odpowiednich metod z pakietu `scipy.fft`. Implementują one interfejs `TransformAnalyzer`, którego atrybutem jest funkcja, która zostanie poddana wybranej transformacie poprzez zastosowanie metody `calculate`. W tym celu zostały zastosowane funkcje przedstawione w tabeli 6.1.

Tabela 6.1 Opis funkcji zastosowanych do obliczeń transformat

Nazwa funkcji	Zastosowanie
`fft`	Obliczanie DFT wykorzystując algorytm FFT
`ifft`	Proces odwrotny do FFT
`fft2`	Obliczanie 2D DFT wykorzystując algorytm FFT
`ifft2`	Proces odwrotny do 2D FFT
`dct`	Obliczanie DCT
`idct`	Proces odwrotny do DCT
`dctn`	Obliczanie n-wymiarowej DCT
`idctn`	Proces odwrotny do DCTN

Każdej jednowymiarowej wielkości odpowiada oś pozioma, dla sygnału ciągłego – oś czasu, dla sygnału dyskretnego – dyskretna oś czasu, a dla transformat – osie częstotliwości. Zostały one zaimplementowane w pliku `Axis.py`.

Znajduje się w nim interfejs `Axis`, w którym zdefiniowane zostały atrybuty takie jak liczba próbek i częstotliwość próbkowania. Wykorzystuje je metoda `generate`, która zwraca wartości danej osi.

Klasa `TimeAxis` odpowiada za oś czasu dla sygnału ciągłego. Wymaga dodatkowego parametru jakim jest krok czasowy. W celu wygenerowania wartości osi czasu wykorzystana została funkcja z biblioteki NumPy – `arange`. Generuje ona wartości od parametru start do stop z odstępem równym parametrowi krok czasowy.

Funkcją klasy `DiscreteTimeAxis` jest wygenerowanie wartości dyskretnych osi czasu. W metodzie `generate` wykorzystana została funkcja `linspace`, która generuje ciąg punktów (próbek) od wartości do stop. Ilość punktów definiuje parametr `num`.

Klasa `FFTAxis` reprezentuje oś dla dyskretnej transformaty Fouriera. Wykorzystana została funkcja `fftfreq` z pakietu SciPy, która w sposób automatyczny generuje wartości częstotliwości dla podanych parametrów, takich jak liczba próbek oraz częstotliwość próbkowania.

Klasa `DCTAxis` reprezentuje oś dla dyskretnej transformaty kosinusowej. Za pomocą funkcji `arange` generowane są wartości częstotliwości wykorzystując parametry takie jak liczba próbek i częstotliwość próbkowania.

6.6.2. Filtracja

Jednym z założeń aplikacji jest możliwość stosowania filtracji w dziedzinie częstotliwości. Funkcjonalność ta została zaimplementowana w plikach `Filters.py` – dla filtracji sygnałów jednowymiarowych oraz `GaussianFilters.py` – dla dwuwymiarowych.

W przypadku jednowymiarowej transformaty DFT oraz DCT do wyboru są cztery rodzaje filtrów:

Funkcją filtrów dolnoprzepustowego oraz górnoprzepustowego jest wyzerowanie współczynników transformaty odpowiadającym częstotliwościom większym(dolnoprzepustowy) lub mniejszym(górnoprzepustowy) od częstotliwości granicznej. W tym celu zastosowano funkcję `where` z biblioteki NumP. Zwraca ona indeksy tablicy, które spełniają określone warunki. W tym przypadku sprawdzamy, gdzie wartości osi częstotliwości są większe lub mniejsze niż częstotliwość graniczna filtru.

Filtr pasmowo-przepustowy ma za zadanie wyzerować współczynniki transformaty odpowiadające częstotliwościom mniejszym niż dolna częstotliwość graniczna oraz większym niż górna częstotliwość graniczna. Efekt ten został osiągnięty poprzez zastosowanie operatora logicznego *lub*(|) pozwalającego na połączenie dwóch warunków.

Filtr pasmowo-zaporowy zeruje współczynniki transformaty w miejscach, gdzie częstotliwość należy do przedziału od dolnej do górnej częstotliwości granicznej. Efekt osiągnięto poprzez użycie operatora logicznego *i(&)*.

W przypadku transformat dwuwymiarowych do wyboru są dwa rodzaje filtrów:

Filtry dolnoprzepustowy i górnoprzepustowy dla dwuwymiarowej dyskretnej transformaty Fouriera zostały zaimplementowane poprzez przemnożenie współczynników transformaty i maski Gaussa, opisanej w rozdziale 3.4. Wykorzystano w tym celu parametry takie jak koordynaty środka filtru oraz przestrzenna częstotliwość graniczna.

W przypadku dwuwymiarowej transformaty kosinusowej, filtry dolnoprzepustowy ustawia wszystkie elementy maski powyżej wybranemu indeksowi wiersza i po lewej stronie od wybranego indeksu kolumny na 1.

Filtr górnoprzepustowy ustawia wszystkie elementy maski poniżej wybranemu indeksowi wiersza i po prawej stronie od wybranego indeksu kolumny na 1.

6.7. Wykresy

Wszystkie obliczenia przeprowadzane w aplikacji są wizualizowane poprzez wykresy. Za integrację interfejsu użytkownika z oknem wykresu jest odpowiedzialna klasa `Canvas` poprzez wykorzystanie narzędzi dostarczonych przez bibliotekę Matplotlib. Tworzy płótno (ang. *canvas*) oraz pasek narzędzi nawigacyjnych. Ponadto, tworzy obiekt osi, który później jest wykorzystany do rysowania wykresu. Zawiera również metodę `clear` służącą do usuwania zawartości z obecnej osi. Obiekty tej klasy są tworzone bezpośrednio w klasie odpowiedzialnej za dane okno aplikacji.

Wyświetlanie wykresów obsługiwane jest w pliku `Graph.py`, który zawiera interfejs definiujący potrzebne atrybuty oraz metodę `create_on_canvas`. Każdy rodzaj wykresu implementuje ten interfejs.

Klasa `Plot` służy do obsługi wykresów funkcji ciągłych w czasie. Tworząc obiekt tej klasy wymagane są parametry takie jak dane osi zarówno poziomej jak i pionowej, opisy osi oraz tytuł wykresu. Nadpisywana jest metoda `create_on_canvas`, która rysuje wykres korzystając z funkcji biblioteki Matplotlib, `plot`.

Wykresy sygnałów dyskretnych w czasie obsługuje klasa `StemPlot`. Jedyną różnicą w porównaniu do wykresów ciągłych jest użycie funkcji `stem`, która zamiast linii ciągłej rysuje wykres w postaci punktów.

Rysowaniem wykresów dwuwymiarowych w postaci obrazów zajmuje się klasa `ImagePlot`. Do narysowania takiego wykresu, jako parametr potrzebujemy jedynie wektora danych osi pionowej oraz tytuł wykresu. Funkcją, z której korzystamy w tym przypadku jest `imshow`, która rysuje obraz z tablicy dwuwymiarowej.

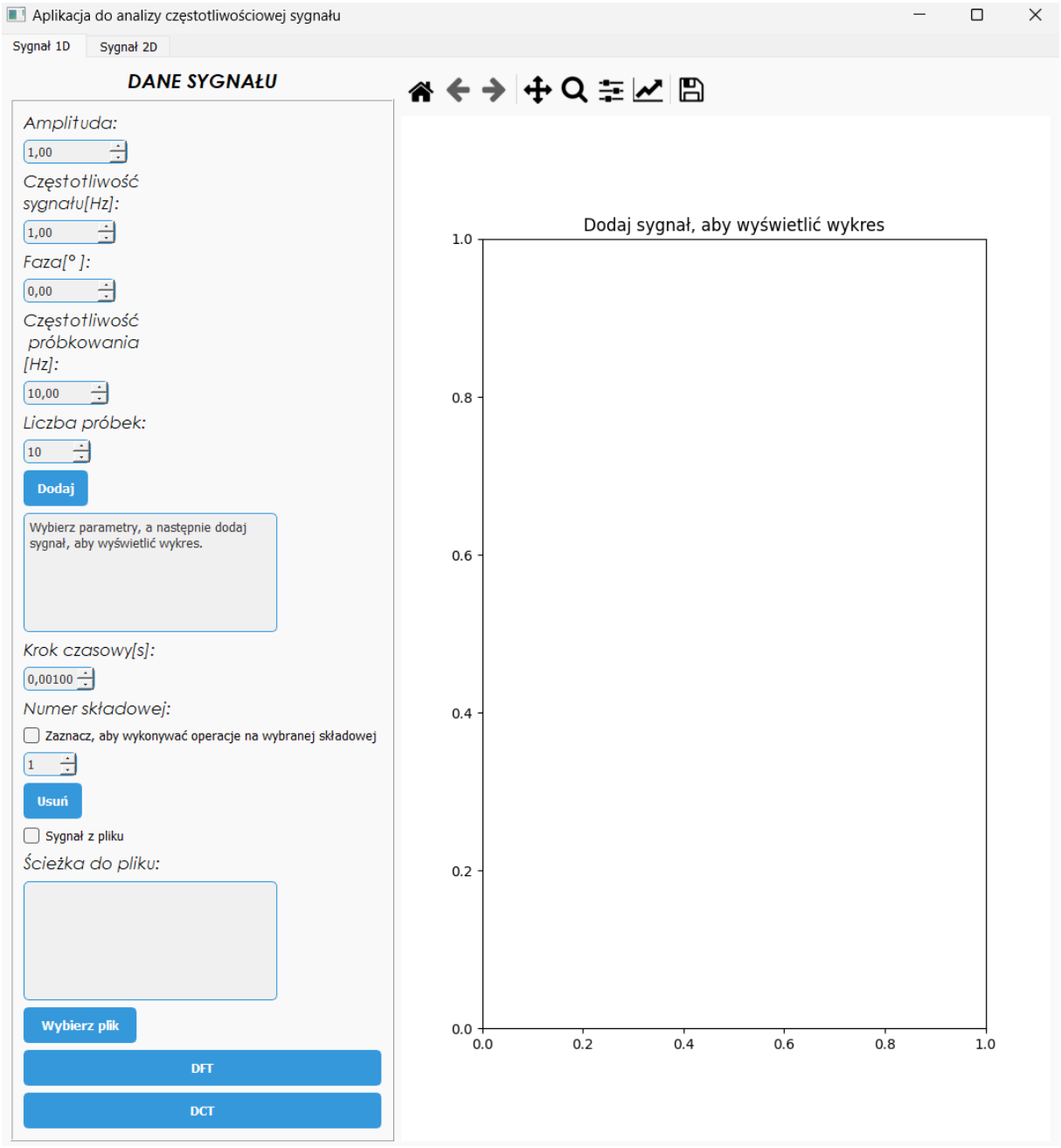
Dodatkowo stworzona została klasa `EmptyPlot`, której nadpisana metoda wyświetla w formie wykresu pusty wykres z informacją o braku wystarczających danych. Została zaimplementowana z myślą o zwiększeniu intuicyjności aplikacji, wyświetlając powiadomienie, gdy aplikacja została świeżo uruchomiona lub nie został wybrany żaden sygnał.

Klasy odpowiadające za sygnały jednowymiarowe zostały ze sobą powiązane w klasie `Receiver`, która następnie przekazuje konkretne informacje do klas implementujących interfejs `Command`, zgodnie ze wzorcem projektowym opisanym w rozdziale 4.2. Dla każdego przycisku wchodzącego w interakcje z użytkownikiem przypisany jest obiekt klasy `Invoker`, którego zadaniem jest przechowywanie poleceń, które zostaną wykonane przy wciśnięciu

przycisku przez użytkownika. Dla sygnałów dwuwymiarowych taką funkcję pełni klasa `Receiver2D`.

7. Interfejs graficzny

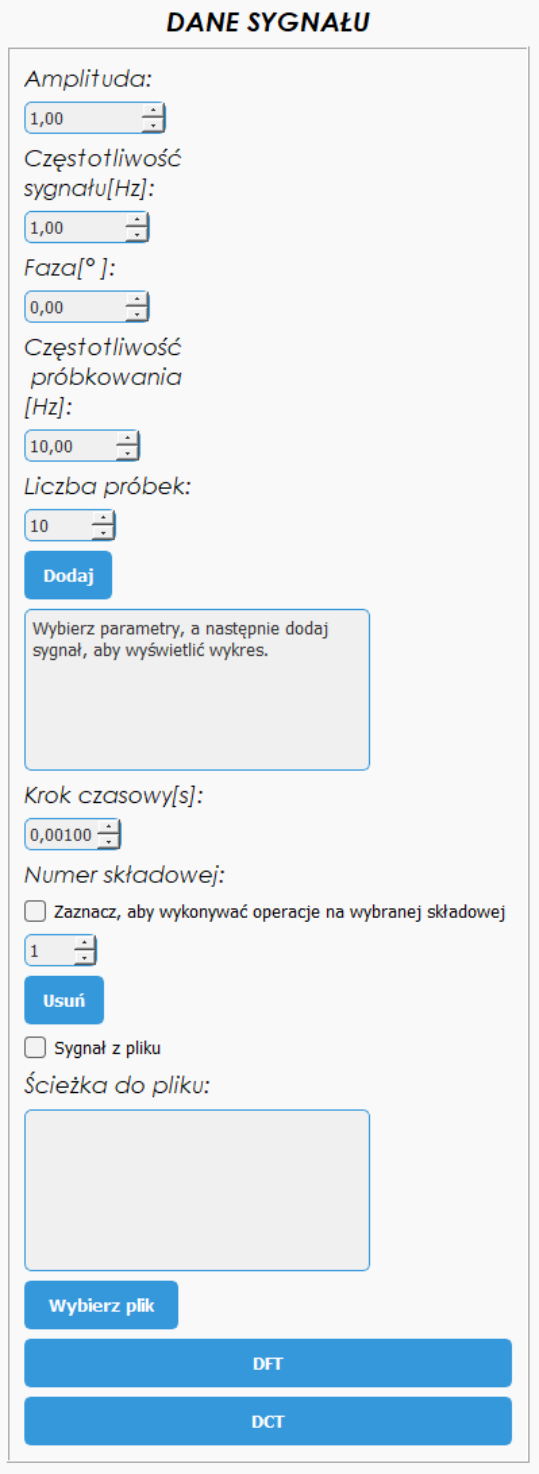
7.1. Okno główne



Rys 7.1 Zakładka dla sygnałów jednowymiarowych

Po uruchomieniu aplikacji (rys. 7.1) pierwszą rzeczą, którą zobaczymy będzie zakładka stworzona do analizy sygnałów jednowymiarowych. Składa się ona z panelu do wyboru parametrów sygnału oraz okna z miejscem, w którym rysowany będzie wykres stworzonego sygnału. W górnej części okna znajdują się zakładki przełączające pomiędzy sygnałem jednowymiarowym a dwuwymiarowym.

7.1.1. Panel do wyboru parametrów analizy



Rys 7.2 Panel do wyboru parametrów

Kluczowym elementem interfejsu użytkownika aplikacji jest panel (rys. 7.2) z elementami, które są odpowiedzialne za interakcje z użytkownikiem. Do dyspozycji użytkownika są parametry opisane w tabeli 7.1.

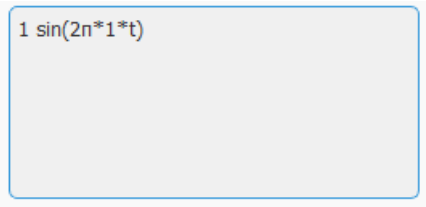
Tabela 7.1. Parametry analizy

Parametr	**Funkcja**
Amplituda	największa wartość danej składowej sygnału
Częstotliwość[Hz]	ilość okresów w czasie jednej sekundy trwania sygnału
Faza[°]	przesunięcie sygnału względem osi OX
Częstotliwość próbkowania[Hz]	liczba próbek pobranych z sygnału w czasie jednej sekundy
Liczba próbek	ilość próbek danego sygnału, która ma zostać narysowana na wykresie
Krok czasowy[s]	odstęp pomiędzy kolejnymi wartościami na osi czasu w przypadku sygnału ciągłego
Numer składowej	po dodaniu składowej sygnału, korzystając z tego pola możemy wybrać, na której składowej chcemy dokonywać operacji takich jak zmiana amplitudy czy częstotliwości i obserwować zmiany w czasie rzeczywistym lub też ją usunąć.

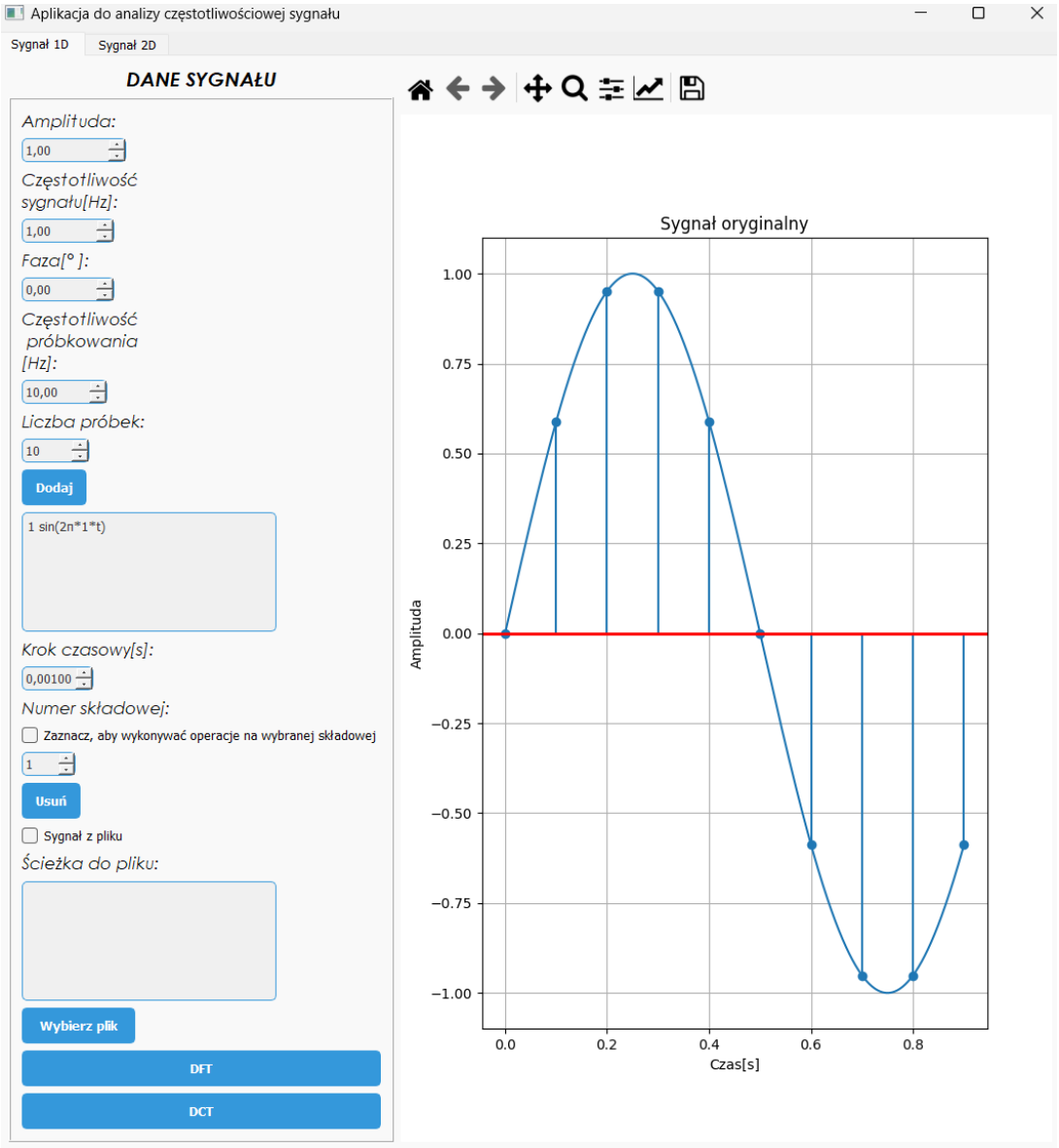
Powyższe pola zawierają również przyciski mające na celu płynną zmianę parametrów, po których wciśnięciu od razu wyświetli jak wpływa dana wielkość na wynik analizy.

Kolejnym elementem wchodzącym w skład panelu służącego do interakcji z użytkownikiem są przyciski, każdy z nich jest odpowiedzialny za inną funkcjonalność.

- *Dodaj* - odczytuje dane z pól odpowiedzialnych za parametry niezbędne do stworzenia sygnału, takie jak amplituda, częstotliwość sygnału i faza, co spowoduje dodanie składowej sygnału o zadanych parametrach i wyświetleniu informacji w polu tekstowym znajdującym się na panelu (rys. 7.3). Pobrane zostaną również parametry obsługujące oś poziomą wykresu. Po dokonaniu obliczeń, zaktualizowane zostaną od razu wszystkie pola z wykresami (rys. 7.4).

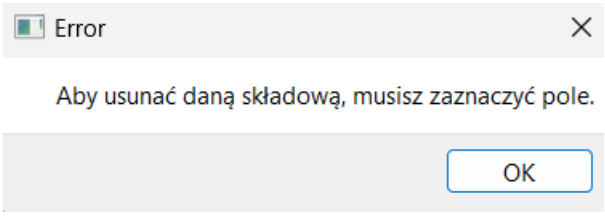


Rys 7.3 Pole z informacjami o przechowywanych składowych



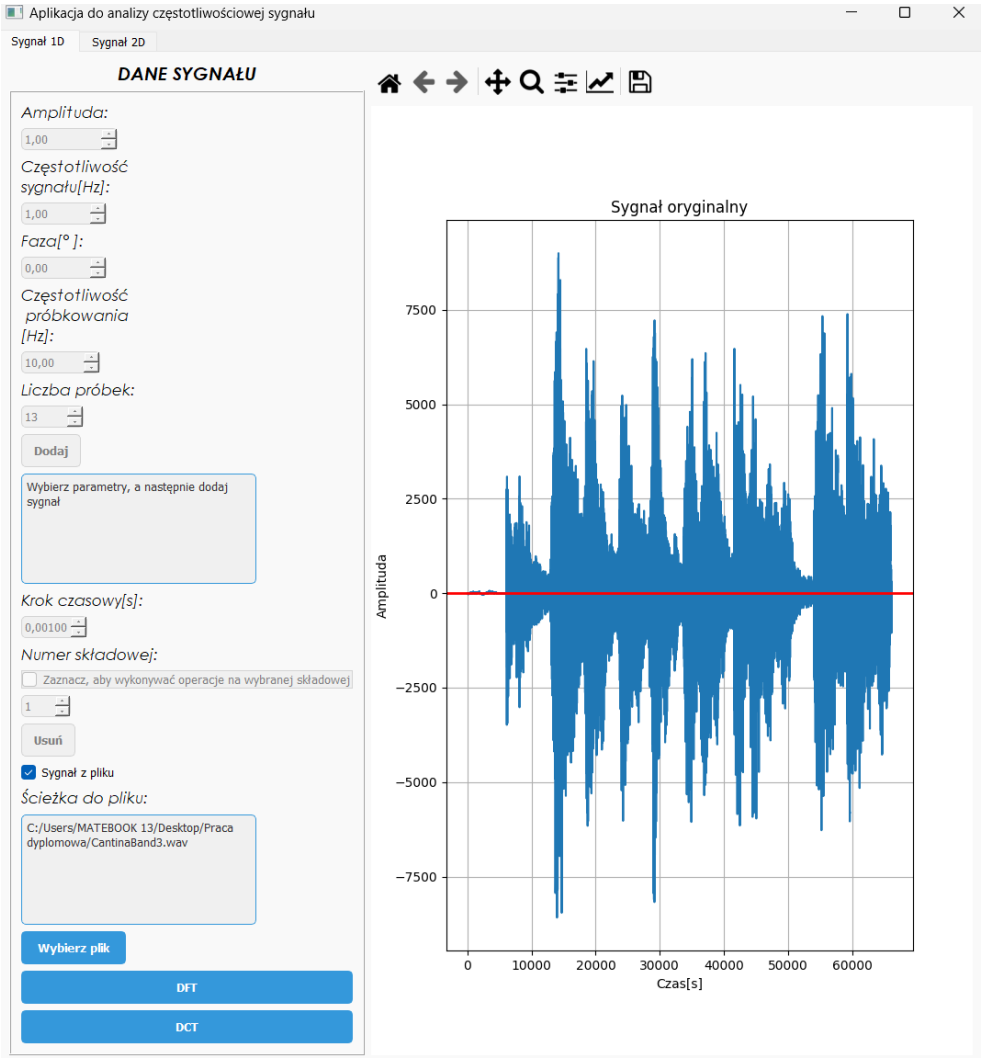
Rys 7.4 Widok aplikacji, po narysowaniu wykresu

- *Usuń* - służy do odczytania wartości numeru składowej oraz następnie jej usunięciu z sygnału. Warunkiem koniecznym do wykonania operacji jest zaznaczenie pola „*Zaznacz, aby wykonywać operacje na wybranej składowej*”. W przeciwnym wypadku wyświetli się komunikat informujący o czynnościach, które musimy podjąć (rys. 7.5).



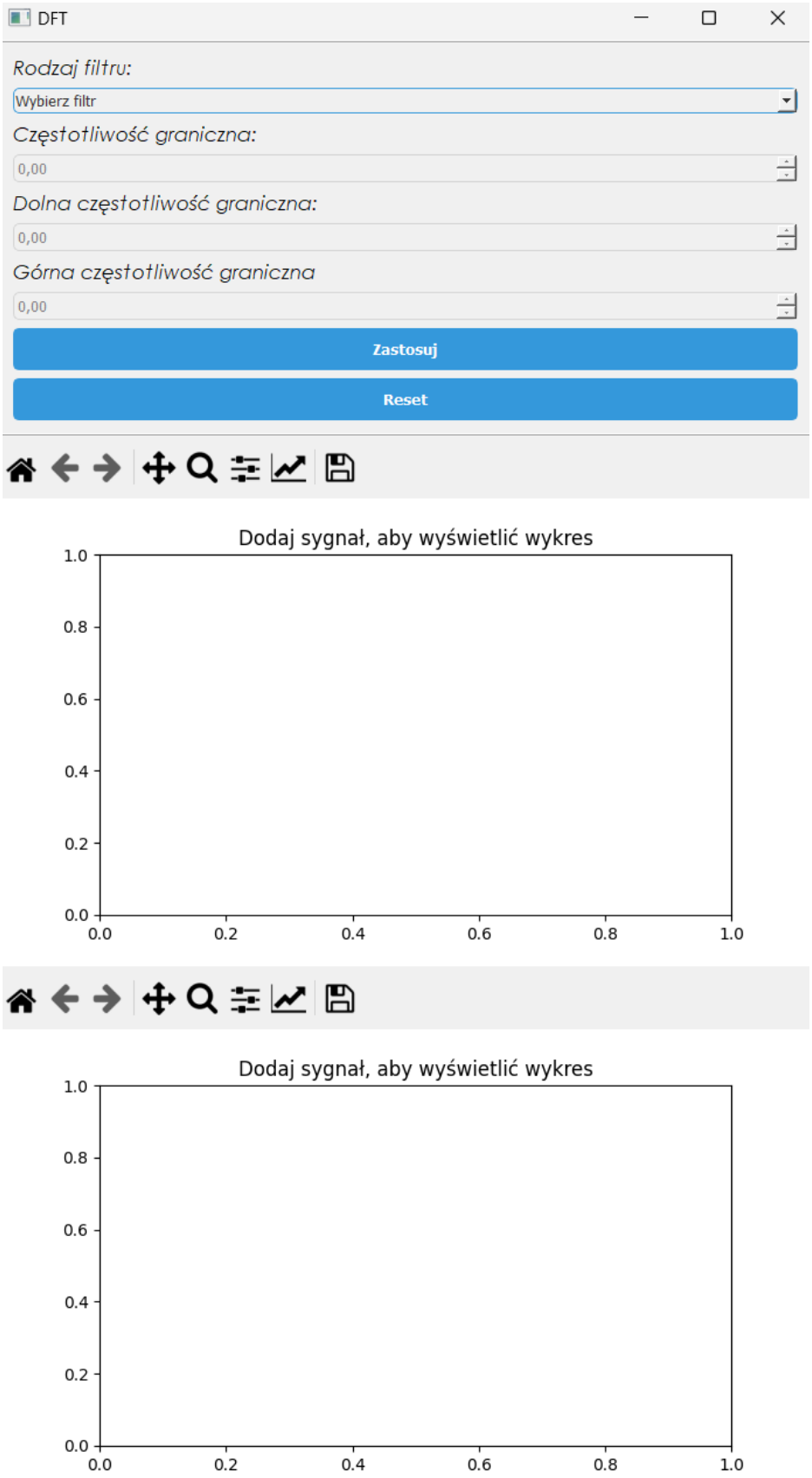
Rys. 7.5 Komunikat o błędzie

Opisane pole musi również zostać zaznaczone, aby można było obserwować zmianę parametrów sygnału na wykresie. W tym przypadku możemy wybrać numer składowej do edycji, pod warunkiem, że składowa o takim indeksie istnieje (numeracja od 1) to możemy zaobserwować, jak zmieniają się wykresy przy jednoczesnej zmianie wybranego parametru.



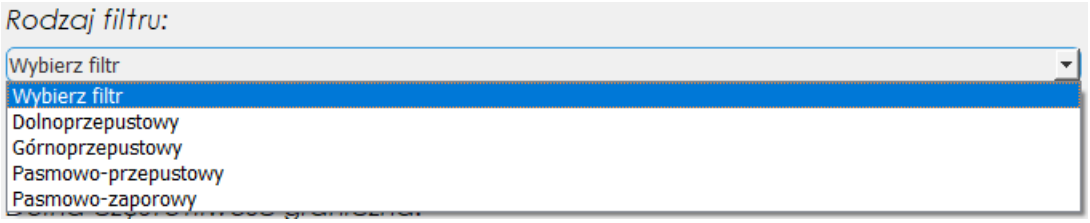
Rys. 7.6 Wykres sygnału z wybranego pliku

- *Wybierz plik* - odpowiada za wybór pliku dźwiękowego o rozszerzeniu *wav* (ang. *Waveform Audio File Format*). Po naciśnięciu przycisku zobaczymy okno z wyborem ścieżki pliku, na którym zostanie dokonana analiza. Następnie, gdy już wybierzemy interesujący nas plik, informacje na jego temat zostaną wyświetlone w polu „*Ścieżka do pliku”*. Ostatnim krokiem do odczytania danych z pliku jest zaznaczenie pola „*Sygnał z pliku”*, dzięki któremu możemy przełączać widok sygnału pomiędzy plikiem, a wybranym sygnałem sinusoidalnym. Gdy zdecydujemy się na sygnał z pliku, wszystkie inne elementy odpowiadające za parametry sygnału zostaną zablokowane, a wyświetlony zostanie wykres danych z pliku (rys 7.6).
- *DFT* - uruchamia oddzielne okno z interfejsem do obsługi czynności związanej z obliczaniem dyskretnej transformaty Fouriera (rys. 7.7).



Rys 7.7 Interfejs do manipulacji DFT

W przypadku interfejsu DFT do dyspozycji użytkownika są pola służące do edycji parametrów związanych z filtracją w dziedzinie częstotliwości. Do wyboru z listy są filtry: dolnoprzepustowy, górnoprzepustowy, pasmowo-przepustowy oraz pasmowo-zaporowy (rys. 4.8). Widoczne są również dwa puste miejsca na wykresy. Na górnym zostanie narysowany wykres modułu DFT, a na dolnym sygnał zrekonstruowany algorytmem IDFT.

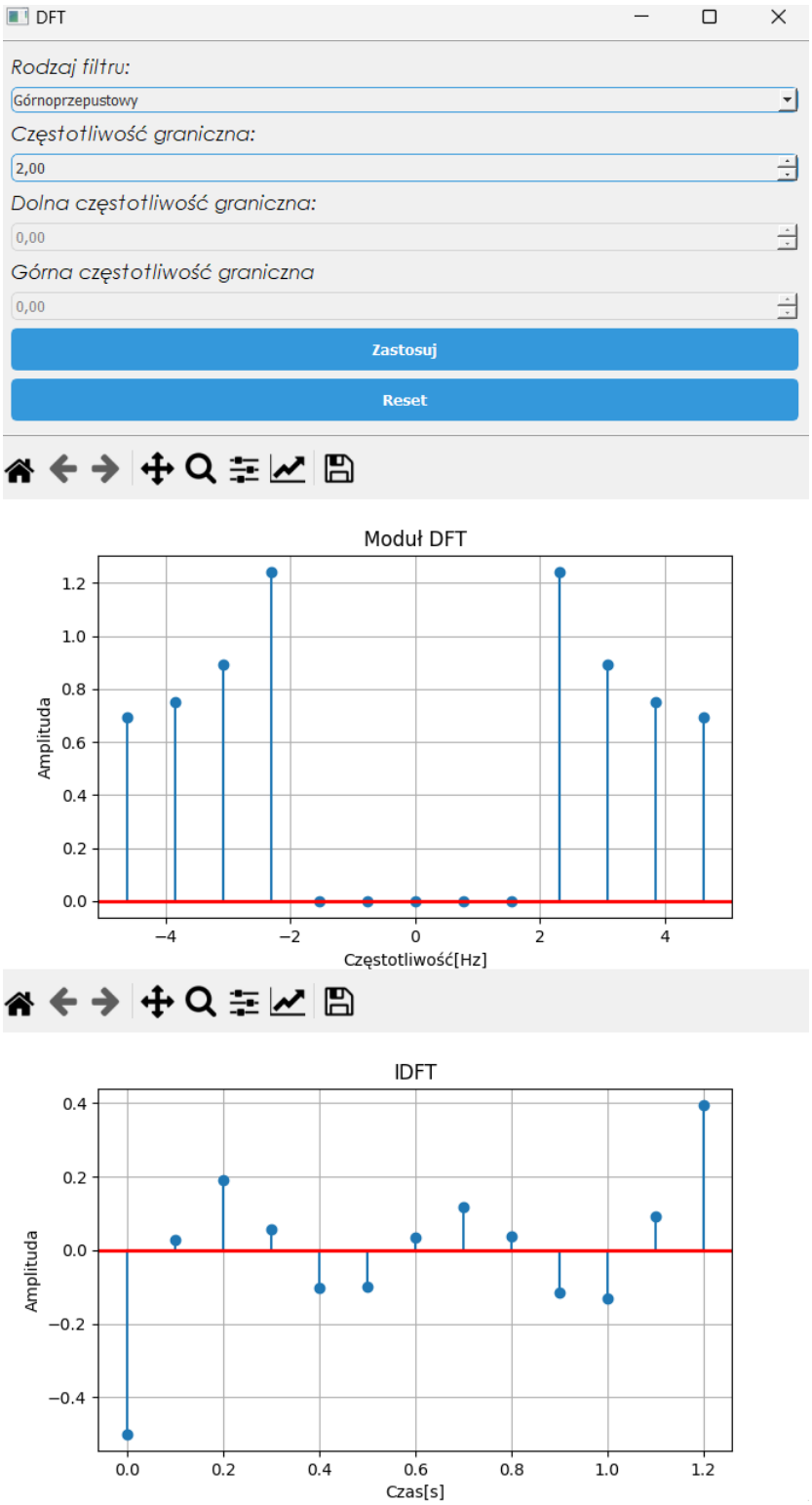


Rys. 7.8 Lista filtrów dla DFT

Po wyborze filtru, w zależności od wybranego rodzaju filtru należy wybrać częstotliwość graniczną dla filtru dolnoprzepustowego i górnoprzepustowego, a dla pasmowo-przepustowego i pasmowo-zaporowego - dolną oraz górną częstotliwość graniczną. W celu zabezpieczenia, w przypadku wyboru filtru, który korzysta z innej liczby parametrów, niepotrzebne pola zostaną zablokowane. Przykład: wybierając filtr górnoprzepustowy, jedynym potrzebnym parametrem jest jedna częstotliwość graniczna, więc pozostałe pola stają się nieaktywne (rys. 7.9).

Rys 7.9 Blokada zbędnych parametrów

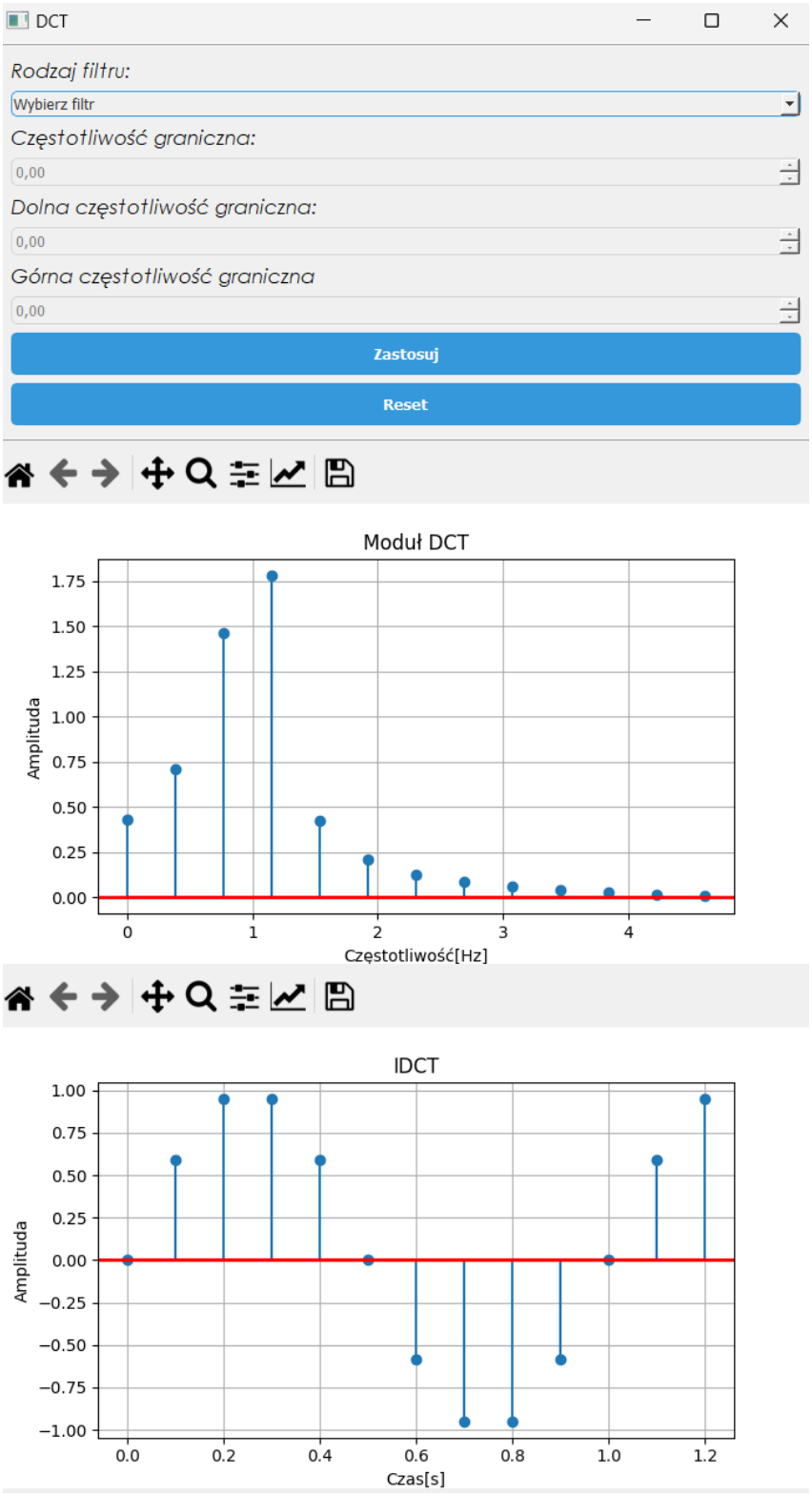
Gdy wybierzemy wszystkie interesujące nas parametry, po wciśnięciu przycisku „*Zastosuj*” okno zostanie zaktualizowane, co spowoduje wyświetlenie wykresów po zastosowaniu wybranego filtru (rys. 7.10).



Rys. 7.10 Wykres DFT i IDFT po zastosowaniu filtru górnoprzepustowego

Ostatnim elementem w interfejsie DFT jest przycisk „*Reset*”, którego naciśnięcie spowoduje cofnięcie zmian - wyświetlenie wykresów przed zastosowaniem filtracji.

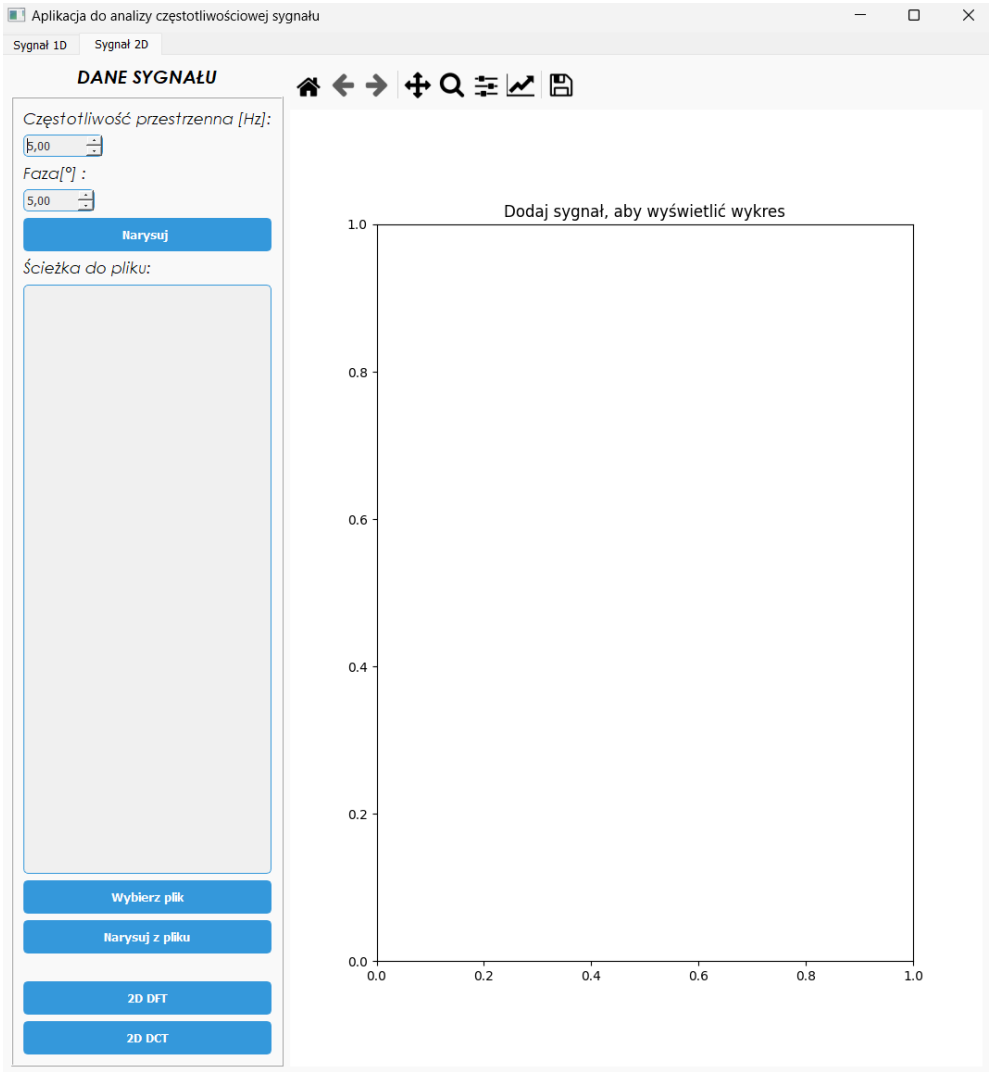
- Przycisk „*DCT*” uruchamia oddzielne okno (rys. 7.11) pozwalające na obserwowanie dyskretnej transformaty kosinusowej, pozostałe funkcje są identyczne jak w przypadku okna DFT.



Rys 7.11 Okno DCT

7.2. Zakładka do analizy sygnałów 2D

Po przełączeniu w lewym górnym rogu zakładki z sygnałów 1D na sygnały 2D zobaczymy następujące okno (rys. 7.12)



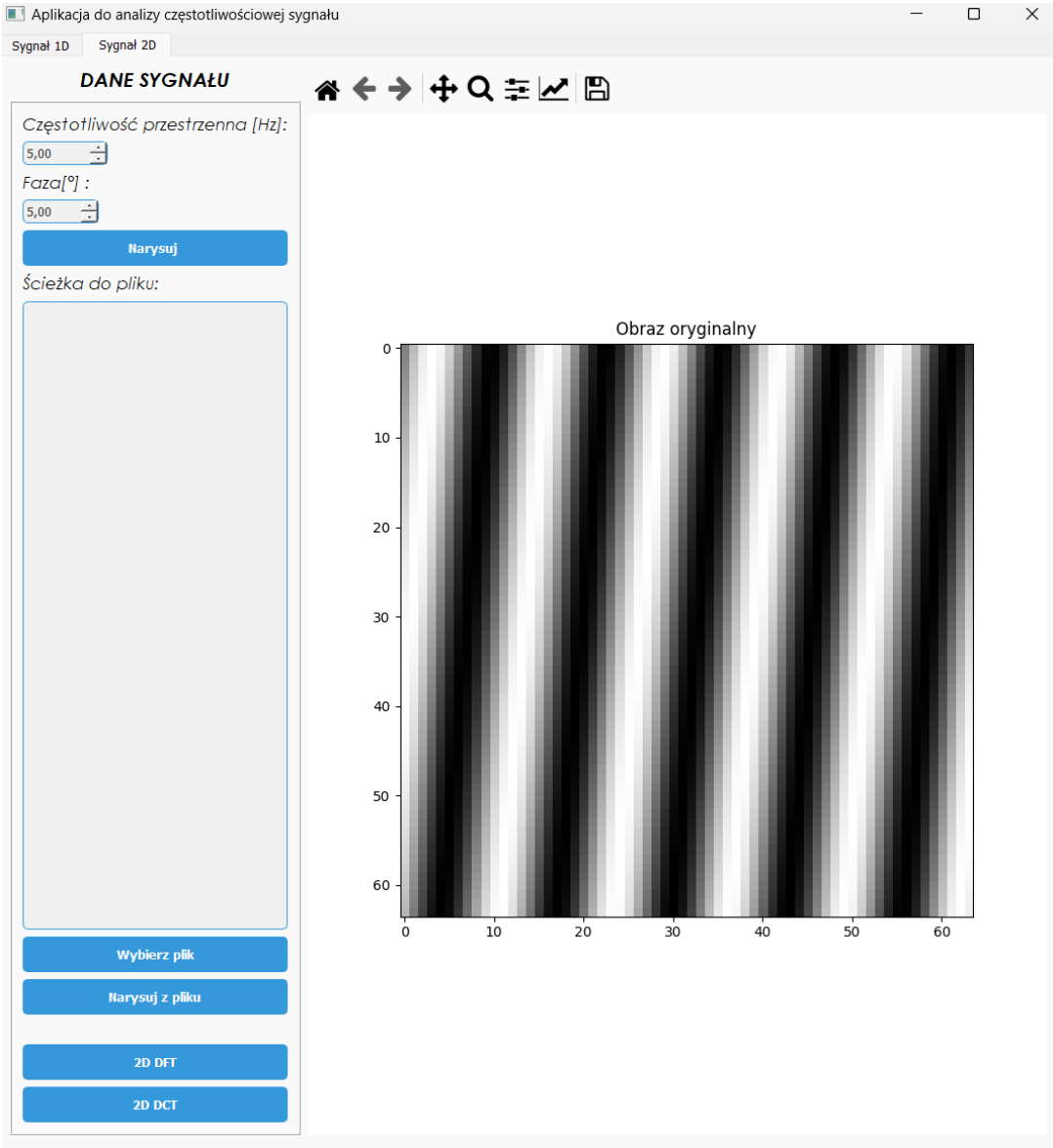
Rys. 7.12 Interfejs do analizy sygnałów dwuwymiarowych

W tej zakładce mamy do wyboru dwa parametry przedstawione w tabeli 7.2.

Tabela 7.2. Parametry potrzebne do stworzenia sygnału 2D

Parametr	**Funkcja**
Częstotliwość przestrzenna[Hz]	odpowiada za to, ile pasków będzie w sygnale dwuwymiarowym
Faza[°]	odpowiada za przesunięcie sygnału

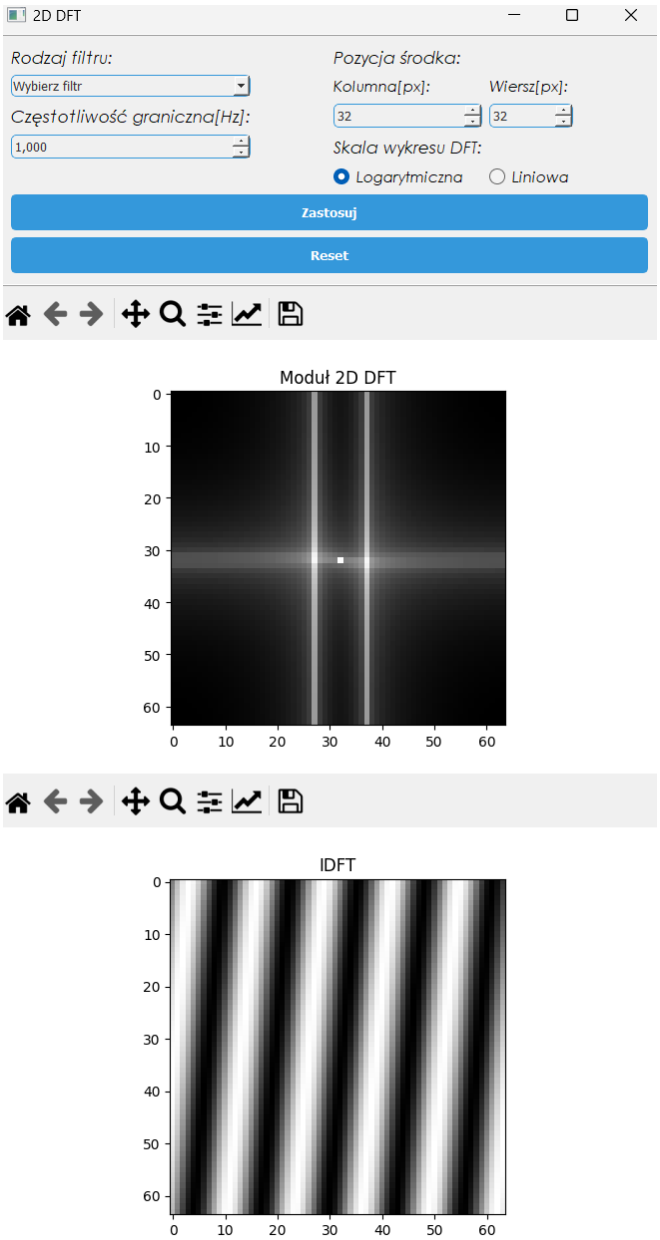
Po dokonaniu wyboru parametrów, w celu narysowania sygnału możemy skorzystać z przycisku *Narysuj*, który ma za zadanie odczytać wybrane parametry i zaktualizować wykresy (rys. 7.13).



Rys. 7.13 Sygnał dwuwymiarowy o częstotliwości 5Hz i fazie 5°

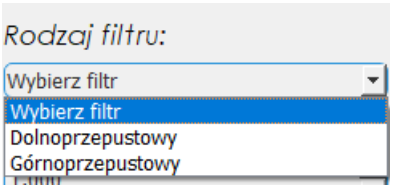
Dodatkową funkcjonalnością jest możliwość wczytania obrazu (sygnału dwuwymiarowego) z pliku o rozszerzeniu *jpg*. Wciskając przycisk *Wybierz plik* będziemy mogli wybrać plik z dysku korzystając z dedykowanego okna. Wykresy zostaną zaktualizowane po naciśnięciu przycisku *Narysuj z pliku*.

Tak samo jak w przypadku sygnałów 1D, możemy użyć dwóch przycisków *2D DFT* oraz *2D DCT* w celu otworzenia okien do analizy odpowiednich transformat.



Rys. 7.14 Widok okna analizy 2D

W przypadku dwuwymiarowej transformaty Fouriera (rys. 7.14) dostępne są filtry dolnoprzepustowy oraz górnoprzepustowy, które możemy wybrać z listy znajdującej się w lewym górnym rogu okna (rys. 7.15).

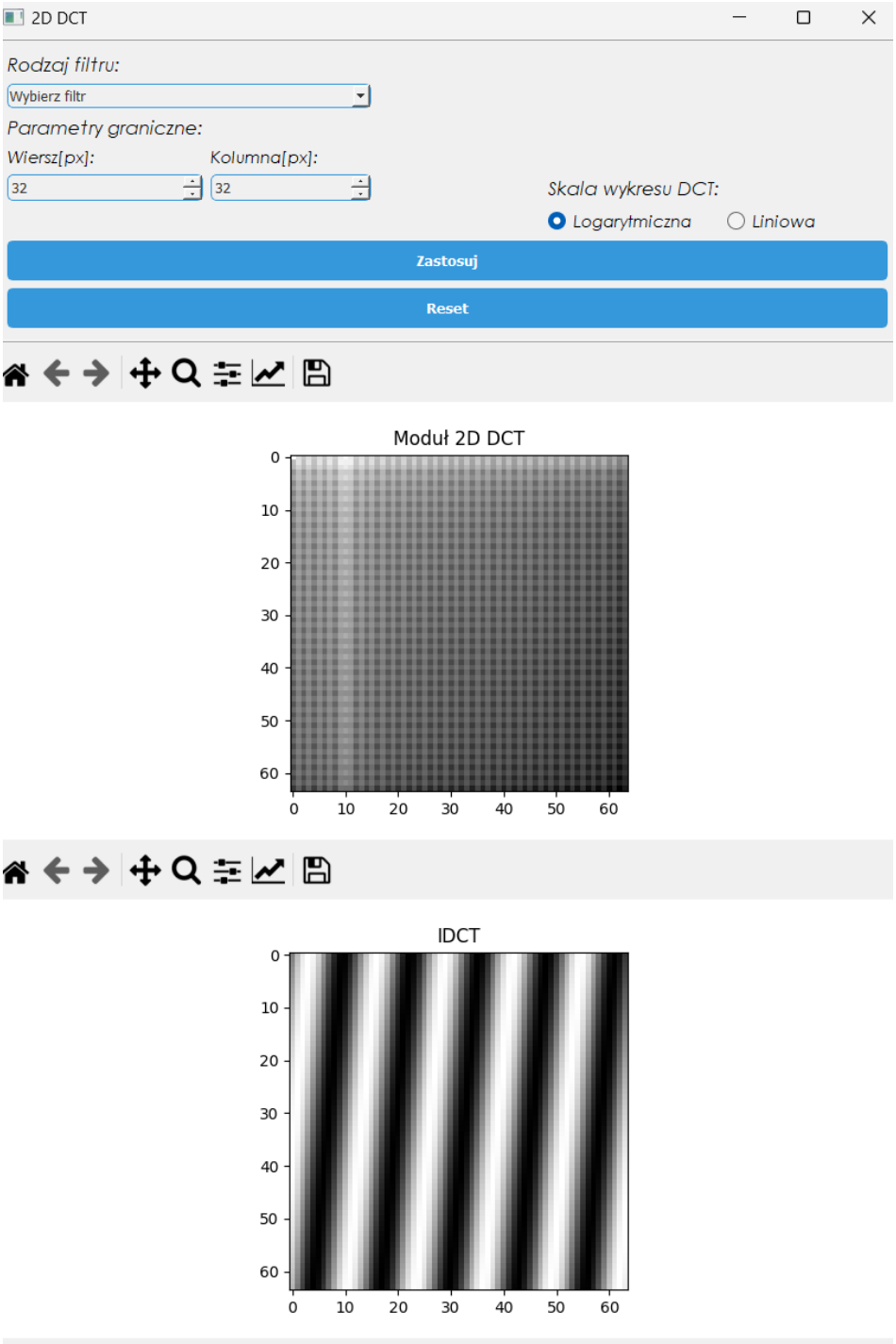


Rys. 7.15 Widok listy z filtrami dla 2D DFT

Parametrem, za pomocą którego możemy manipulować wykresami jest częstotliwość graniczna. Istnieje również możliwość wybrania pozycji środka obrazu, dla którego zostanie

zastosowany filtr. Po wybraniu parametrów, należy skorzystać z przycisku „*Zastosuj*”, który na podstawie danych parametrów zaktualizuje wykresy. Zostaną one domyślnie wykreślone w skali logarytmicznej, a za pomocą pola do zaznaczania możemy wybrać również skalę liniową. Przycisk *Resetuj* służy do cofnięcia zmian.

- Ostatnim elementem jest przycisk „2D DCT”, który otwiera okno do analizy dwuwymiarowej transformaty kosinusowej (rys. 7.16)

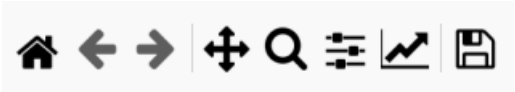


Rys. 7.16 Okno do analizy 2D DCT

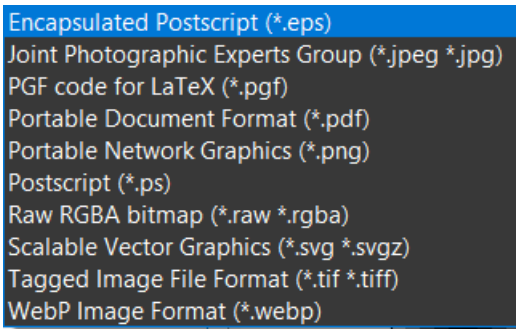
Zaimplementowane zostały tutaj te same funkcjonalności co w przypadku 2D DFT, jednak w tym wypadku parametrami, które są wykorzystywane w filtracji są wiersz oraz kolumna.

7.3. Opcje wykresów

Wszystkie wykresy wykorzystane w projekcie zostały zrealizowane za pomocą biblioteki Matplotlib, która dostarcza szereg funkcji (rys. 7.17) umożliwiających manipulację danymi. Do wyboru mamy funkcje takie jak:

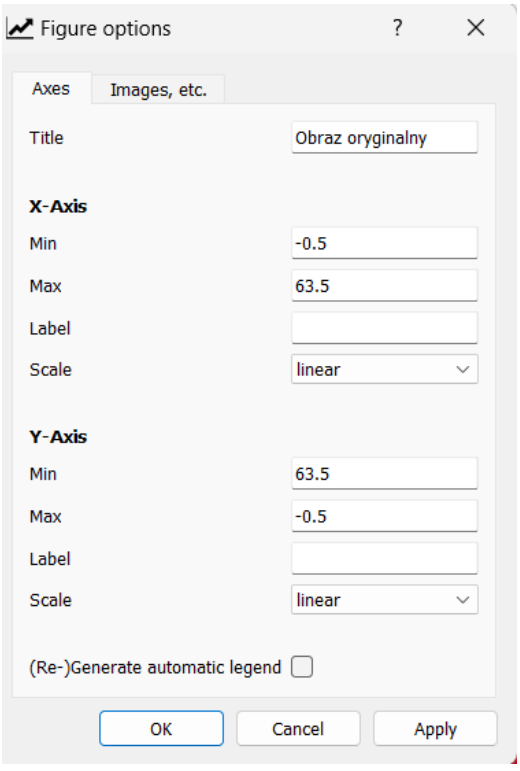


Rys 7.17 Panel z funkcjami wykresu

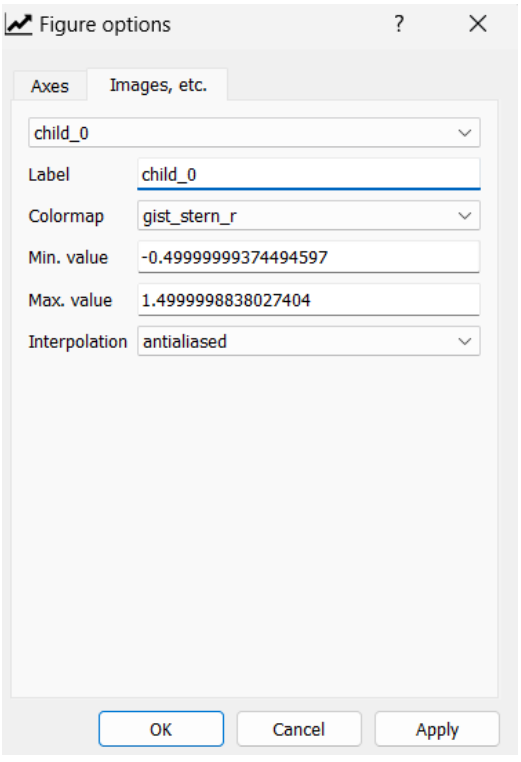


Rys. 7.18 Formaty w których możemy zapisać wykres

- zapis wykresu do pliku – do dyspozycji są formaty widoczne na rysunku (rys. 7.18)

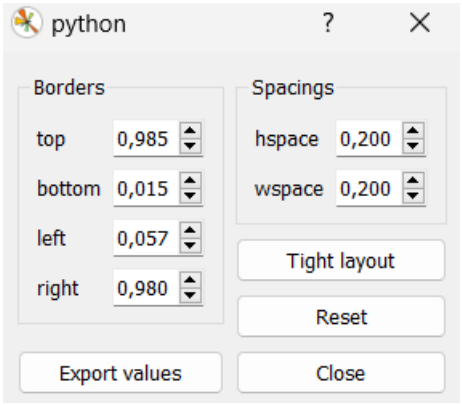


Rys. 7.19 Opcje dotyczące osi wykresu



Rys. 7.20 Opcje dotyczące obrazu

- edycja parametrów opisu osi, krzywej oraz obrazu – w tej sekcji udostępniona została możliwość zmiany tytułu wykresu, wybór minimalnej oraz maksymalnej wartości danej osi, tytuły oraz jej skalowanie (rys. 7.19). Ponadto, w zakładce *Images* (rys. 7.20) możemy manipulować paletą barw wykresu poprzez zmianę parametrów.



Rys. 7.21 Opcje zmiany rozmiaru wykresu

- zmiana rozmiaru wykresu – w tej zakładce (rys. 7.21) możemy edytować marginesy oraz odstępy pomiędzy wykresem a interfejsem graficznym.

  Po zastosowaniu każdej ze wspomnianych zmian możemy je cofnąć w sposób bezproblemowy i intuicyjny korzystając z dedykowanych przycisków.

7.4. Możliwości rozbudowy interfejsu graficznego

Interfejs graficzny został zaprojektowany z myślą o łatwym dodawaniu nowych funkcjonalności. Główne funkcje programu implementowane są poprzez kolejne zakładki, których dodanie nie wiąże się w żaden sposób z edycją istniejących fragmentów kodu ani reorganizacją istniejących elementów interfejsu. Każda podfunkcja programu jest implementowana jako nowe okno, które otwiera się po wciśnięciu odpowiedniego przycisku. Dla każdego okna istnieje oddzielny plik o rozszerzeniu *.ui*, który definiuje układ interfejsu, oraz odpowiadająca mu klasa.

8. Testy aplikacji

Testowanie to kluczowy proces w cyklu projektowania aplikacji. Jego celem jest zapewnienie jakości oraz niezawodności projektu końcowego. Istnieje wiele metod testowania, a każda z nich ma swoje zastosowanie.

Testy regresyjne wykonywane są w celu sprawdzenia, czy dodawanie nowych funkcjonalności nie spowodowało wystąpienia błędów we wcześniej zaimplementowanych funkcjach.[14]

Innym rodzajem testów jest tzw. *monkey-testing,* który polega na wykonywaniu losowych akcji w aplikacji bez konkretnie zaplanowanego scenariusza. Celem jest wykrycie błędów, które mogą wystąpić podczas przypadkowego użytkowania programu.[14]

Testy ponowne (ang. *Retesting*) polegają na ponownym przeprowadzeniu testów danej funkcjonalności, która wcześniej okazała się wadliwa. Ich celem jest sprawdzenie czy poprawki nie wpłynęły negatywnie na inne obszary aplikacji oraz czy błędy zostały pomyślnie naprawione.[14]

8.1. Okno główne

Pierwszy przypadek testowy dotyczył dopasowania aplikacji do różnych rozdzielczości ekranu. Aplikacja została testowana na monitorach o rozdzielczości *2160 x 1440px* oraz *1920 x 1080px.* W obu przypadkach wszystkie elementy interfejsu były widoczne. Dodatkowo, wszystkie pola oraz przyciski automatycznie skalują się wraz z manualną zmianą rozmiaru okna aplikacji.

Kolejnym etapem były testy walidacji danych wejściowych, w tym próby wprowadzenia niepoprawnych danych, takich jak litery zamiast liczb lub nieprawidłowe formaty plików. W przypadku wyboru pliku dźwiękowego program nie akceptuje innych plików niż format *.wav,* a dla plików obrazu jedynie format .jpg, co spełnia założenia projektowe. Wszystkie pola, które oczekują danych liczbowych zostały zabezpieczone przed wprowadzeniem znaków lub liter. Przewidziana została możliwość zostawienia pustych pól danych – w tym wypadku automatycznie zostanie użyta wartość przed usunięciem. Wartości takie jak częstotliwość, liczba próbek, krok czasowy oraz numer składowej powinny być dodatnie – nie jest możliwe zastąpienie tych pól wartościami, które byłyby niezgodne z logiką aplikacji. Pola tekstowe również są niedostępne dla użytkownika – ich zawartość może być edytowana jedynie z poziomu kodu programu.

Program pozwala na edycję parametrów konkretnej składowej sygnału. Zaznaczenie pola znajdującego się pod etykietą numeru składowej sygnału spowoduje to, że wykresy będą aktualizowane po każdej edycji parametrów, z których tworzona jest fala sinusoidalna. Testy tej funkcjonalności polegały na wyborze numeru składowej, której nie ma w sygnale. Nie jest możliwe ustawienie numeru składowej poniżej wartości 1. Natomiast przy próbie wybrania numeru większego niż liczba składowych sygnału zostanie wyświetlony odpowiedni komunikat.

Zaznaczając pole z etykietą *Sygnał z pliku* automatycznie wykresy sygnału zostaną zastąpione tymi obliczonymi z danych z pliku. W przypadku, gdy nie został wybrany żaden plik, wyświetlony zostanie pusty wykres informujący użytkownika, aby dokonał wyboru sygnału. W czasie, gdy wspomniane pole jest zaznaczone, zablokowane została możliwość wybrania parametrów sygnału. Odznaczenie spowoduje narysowanie wykresów na podstawie istniejącego wcześniej sygnału.

W zakładce dla sygnałów 2D zastosowane zostały te same testy z różnicą, że zamiast pola do wyboru pomiędzy sygnałem z pliku, a sygnałem z wybranych parametrów zostały zastosowane dwa różne przyciski. Jeden do rysowania sygnału z parametrów, zaś drugi z pliku. W przypadku, gdy nie wybierzemy żadnego pliku zostanie wyświetlony stosowny komunikat.

8.2. Okna transformat

Dla wszystkich obliczanych transformat w programie istnieje możliwość stosowania filtrów na podstawie wybranych parametrów. W przypadku transformat jednowymiarowych testy zostały wykonane równolegle dla DFT jak i DCT, ze względu na to, że okna te są identyczne. Testy obejmowały próby zastosowania filtrów bez wcześniejszego wyboru sygnału oraz wybór filtrów spoza dostępnych opcji, co skutecznie wyświetliło odpowiednie komunikaty. Po wybraniu sygnału, przetestowane zostały zabezpieczenia przed wprowadzeniem niepoprawnych danych. Tak samo jak w przypadku okna głównego, niemożliwe jest wprowadzenie danych innych niż liczby jak częstotliwości graniczne. Ważny jest fakt, że żadna z częstotliwości, górna lub dolna, nie mogą być ujemne. Aplikacja nie umożliwia ustawienia żadnej z nich jako mniejszej niż zero. Ponadto, przy próbie zastosowania filtru, gdzie dolna częstotliwość graniczna jest wyższa niż dolna – zostaje wyświetlony wyjaśniający komunikat.

Następne testy dotyczą transformat dwuwymiarowych. Dla transformaty 2D DFT pierwszy przypadkiem testowym był wybór filtra bez uprzednio wybranego obrazu. Zarówno dla filtru dolnoprzepustowego jak i górnoprzepustowego wyświetlony został komunikat informujący o

braku danych do narysowania wykresu. Po dodaniu obrazu, ale braku wyboru filtru komunikat informuje nas o konieczności jego wybrania.

W tym wypadku również została sprawdzona reakcja aplikacji na poprawność wprowadzanych danych. Niemożliwym jest wybór ujemnej częstotliwości granicznej. Obraz ma ustawione wymiary 64x64px i w tym samym zakresie możemy manipulować parametrami współrzędnych środka filtru.

Identyczne testy zostały przeprowadzone dla okna transformaty 2D DCT.

9. Podsumowanie

9.1. Podsumowanie pracy

Przedmiotem pracy było stworzenie aplikacji, mającej ułatwić proces nauczania cyfrowego przetwarzania sygnałów a w szczególności analizy częstotliwościowej sygnałów. W tym celu zaprojektowany został interfejs graficzny, który nie tylko jest prosty w obsłudze, ale jest także intuicyjny, aby umożliwić łatwą interakcje użytkownika z aplikacją. Założeniem było umożliwienie wizualizacji wpływu różnych parametrów analizy widmowej w czasie rzeczywistym, co stanowi kluczowy element w procesie dydaktycznym.

Implementacja aplikacji odbyła się w środowisku PyCharm, które jest wydajnym i wszechstronnym narzędziem do programowania w języku Python. Język ten został wybrany ze względu na jego elastyczność, czytelność i szeroką dostępność bibliotek. Wykorzystując wzorzec projektowy *Command* oraz staranne przestrzeganie zasad *SOLID* zapewniono wysoką jakość kodu oraz łatwość jego rozbudowy oraz utrzymania.

Główne cele projektowe zostały osiągnięte, a aplikacja pozwala na obserwację wpływu parametrów sygnału na wynik analizy częstotliwościowej. Dodatkowo, dla każdej transformacji użytkownik ma możliwość zaprojektowania własnego filtru i obserwacji jego działania na odtworzony sygnał za pomocą transformaty odwrotnej. Takie rozwiązanie znacznie ułatwia proces nauki.

9.2. Możliwe kierunki dalszego rozwoju

W przyszłości do aplikacji mogłaby zostać dodana funkcjonalność, która umożliwia wybór okna czasowego do analizy sygnału jednowymiarowego. Zastosowanie okna ma wpływ na wynik analizy częstotliwościowej, zwłaszcza w kontekście minimalizacji przecieku widma.

Kolejną dodatkową funkcjonalnością mogłaby być analiza częstotliwościowa sygnałów o zmiennej częstotliwości. W tym celu konieczne byłoby zaimplementowanie algorytmów obliczających krótkoczasową transformację Fouriera STFT(ang. *Short-Time Fourier Transform*).

Bibliografia

[1]Zieliński T., *Cyfrowe Przetwarzanie Sygnałów – od teorii do zastosowań*, Warszawa, Wydawnictwo Komunikacji i Łączności, 2005

[2]https://en.ryte.com/wiki/Wolfram_Alpha [dostęp 14.02.2024]

[3]https://www.pcworld.com/article/2002140/new-audacity-turns-your-computer-into-a-sound-studio-for-free.html [dostęp 14.02.2024]

[4]https://forreya.medium.com/the-solid-principles-writing-scalable-maintainable-code-13040ada3bca [dostęp 10.01.2024]

[5]https://thepythoncodingbook.com/2021/08/30/2d-fourier-transform-in-python-and-fourier-synthesis-of-images/ [dostęp 15.01.2024]

[6] https://refactoring.guru/design-patterns/command [dostęp 12.02.2024]

[7]Steven W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, California, California Technical Publishing, 1999

[8]https://sound.eti.pg.gda.pl/~greg/dsp/01-AnalizaWidmowa.html [dostęp 01.02.2024]

[9]https://users.cs.cf.ac.uk/dave/Multimedia/node231.html [dostęp 01.02.2024]

[10]https://mikrokontroler.pl/2021/01/13/poradnik-ltspice-tipstricks-7-tajniki-probkowania-aliasing-sygnalow-analogowych/ [dostęp 01.02.2024]

[11]Stanley H. Mneney, *An Introduction to Digital Signal Processing: A Focus on Implementation*, California, River Publishers, 2008

[12]Joakim Sundnes, *Introduction to Scientific Programming with Python*, Lysaker, Simula Research Laboratory, 2020

[13]https://medium.com/jun94-devpblog/cv-2-gaussian-and-median-filter-separable-2d-filter-2d11ee022c66 [dostęp 15.02.2024]

[14]Sacha K., *Inżynieria oprogramowania*, Warszawa, Wydawnictwo Naukowe PWN, 2010

Załączniki

1. Kod aplikacji: https://github.com/dominik200029/Praca-dyplomowa/tree/master