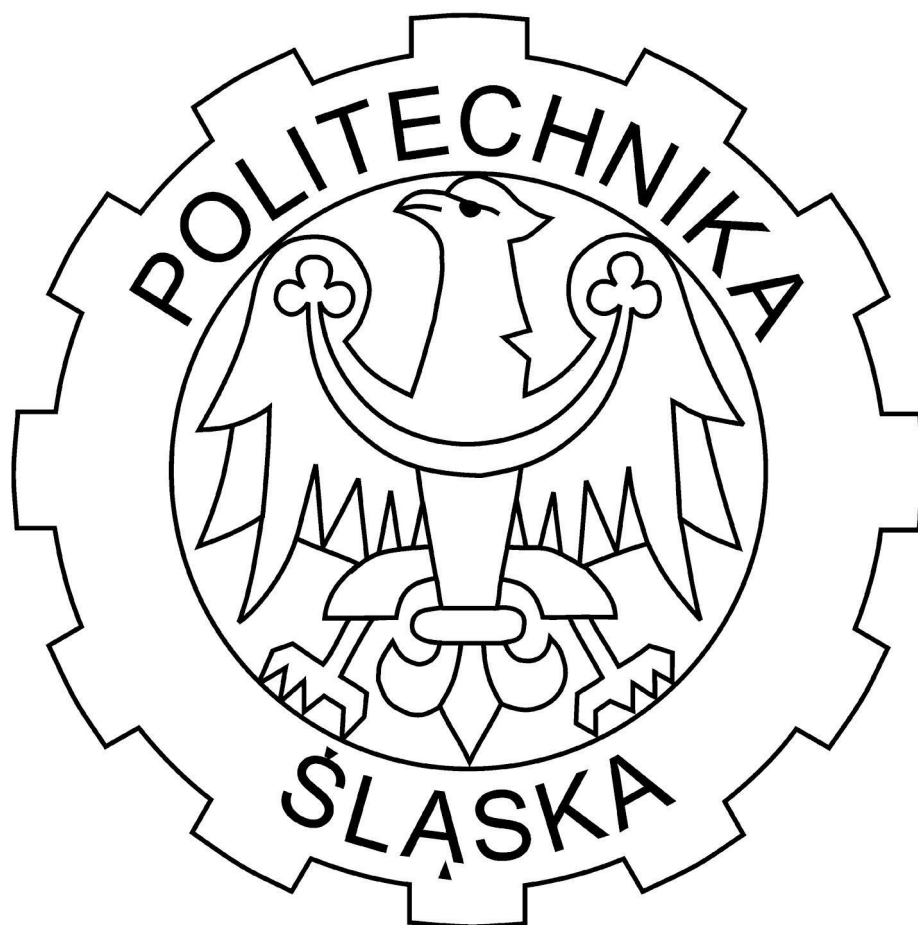


# Programowanie komputerów: Projekt Fourier



**Prowadzący:** dr inż. Tomasz Moroń

**Autor:** Marek Bąk

Teleinformatyka Sem. 3

**Data wykonania (sprawozdania):** 19.01.2021

# Wstęp

Celem mojego projektu była realizacja transformacji fouriera oraz transformacji odwrotnej, sygnału przekazanego na wejście. Program w dużej mierze został napisany z wykorzystaniem kontenerów (wektorów). Zostały w nim zawarte również strumienie (`std::istream`), algorytmy (`std::copy(...)`). Uwzględnione zostały również metody wirtualne oraz obsługa wyjątków. Wycieki pamięci wykrywała biblioteka `<crtdbg.h>`. Dane zawarte w pliku wejściowym są liczbami rzeczywistymi a ich liczba jest potęgą dwójki. Przekazany sygnał musi być zawarty w pliku z rozszerzeniem `“.csv”`. Po wykonaniu programu otrzymujemy dwa pliki z `“.csv”`. Plik uzyskany przez realizację szybkiej transformacji fouriera jest zbiorem liczb zespolonych, które należy przekopiować do odpowiednich kolumn arkusza `“SzablonFFT”` (zawartego w projekcie), aby uzyskać jego reprezentację graficzną. Z kolei plik otrzymany poprzez realizację IFFT na sygnale po przejściu przez FFT, zwraca nam ciąg liczb rzeczywistych, które pokrywają się z tymi wprowadzonymi do programu.

# Teoretyczne podstawy projektu

Cały program dzieli się na poszczególne klasy, zgodnie z diagramem przedstawionym na rysunku 1.



Zdj. 1. Diagram klas

W projekcie zostały wykorzystane następujące wzory/algorytmy pozwalające na wyliczenie FFT oraz IFFT:

```

$$X_0, \dots, N-1 \leftarrow \text{ditfft2}(X, N, s):$$

$$\text{if } N = 1 \text{ then}$$

$$X_0 \leftarrow X_0$$

$$\text{else}$$

$$X_0, \dots, N/2-1 \leftarrow \text{ditfft2}(X, N/2, 2s)$$

$$X_{N/2}, \dots, N-1 \leftarrow \text{ditfft2}(X+s, N/2, 2s)$$

$$\text{for } k = 0 \text{ to } N/2-1$$

$$t \leftarrow X_k$$

$$X_k \leftarrow t + \exp(-2\pi i \cdot k/N) X_{k+N/2}$$

$$X_{k+N/2} \leftarrow t - \exp(-2\pi i \cdot k/N) X_k$$

$$\text{endfor}$$

$$\text{endif}$$

```

DFT of  $(X_0, X_s, X_{2s}, \dots, X_{(N-1)s})$ :

przypadek bazowy trywialnego rozmiaru DFT = 1

DFT  $(X_0, X_{2s}, X_{4s}, \dots)$

DFT  $(X_s, X_{s+2s}, X_{s+4s}, \dots)$

łączenie DFT dwóch połówek do pełnego DFT:

Zdj.2 Pseudokod algorytmu Cooleya Tukeya

$$x(n) = \frac{1}{N} \left[ \sum_{k=0}^{N-1} X^*(k) W_N^{kn} \right]^*$$

Zdj. 3 Wzór na wyliczenie IFFT

W programie zostały zastosowane również następujące klasy (struktury danych):

*file\_read* - odpowiedzialna za odczyt danych z pliku .csv

*save\_to\_file* - odpowiedzialna za zapis wyników do poszczególnych plików

*IFFT* - dziedzicząca publicznie po *FFT*, klasa odpowiadająca za obliczenie IFFT na podstawie zdjęcia 3

*ditfft2* - dziedzicząca publicznie po *FFT*, klasa odpowiadająca za obliczenie FFT. Algorytm na obliczanie takowej transformaty zamieszczony jest na zdjęciu 2

*FFT* - Jest klasą bazową dla IFFT oraz *ditfft2*. Odpowiedzialna jest za połączenie tych dwóch klas razem ze sobą, ponieważ na FFT składa się zarówno wyliczenie samej transformaty, jak i transformaty odwrotnej.

**Dokumentacja wewnętrzna:**

**FourierProjekt Marek Bąk**

AUTHOR  
Version  
Tue Jan 19 2021

## **Table of Contents**

Table of contents



# Hierarchical Index

## Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- FFT

  - IFFT

  - ditfft2

- file\_read

- save\_to\_file

# Class Index

## Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- ditfft2** (Klasa ditfft2 dziedzicząca publicznie po FFT odpowiedzialna jest za wyliczenie FFT )
- FFT** (Klasa bazowa FFT )
- file\_read** (Klasa odpowiedzialna za odczyt danych z pliku z rozszerzeniem .csv )
- IFFT** (Klasa IFFT dziedzicząca publicznie po FFT odpowiedzialna jest za obliczenie odwrotnej transformacji fouriera )
- save\_to\_file** (Klasa odpowiedzialna za zapis danych bezpośrednio do pliku )



## **Class Documentation**

## ditfft2 Class Reference

Klasa **ditfft2** dziedziczaca publicznie po **FFT** odpowiedzialna jest za wyliczenie **FFT**

`#include <FFT.h>`

Inheritance diagram for ditfft2:

### Public Member Functions

- **ditfft2** (std::vector< double > insert, double ilosc\_danych, int s)
- std::vector< std::complex< double > > **CooleyTukeyAlgorithm** (const std::vector< double > x, double N, int s)

*Funkcja realizująca algorytm Cooley'a Tukey'a do wyliczenia **FFT***

- std::vector< std::complex< double > > **fftwektor** () override  
*Metoda przeciążająca metodę wirtualna w klasie **FFT**, służy do wyliczenia **FFT***

### Additional Inherited Members

---

### Detailed Description

Klasa **ditfft2** dziedziczająca publicznie po **FFT** odpowiedzialna jest za wyliczenie **FFT**

---

### Member Function Documentation

**std::vector< std::complex< double > > ditfft2::CooleyTukeyAlgorithm** (const std::vector< double > x, double N, int s)

Funkcja realizująca algorytm Cooley'a Tukey'a do wyliczenia **FFT**

#### Returns

Zwraca wynik w postaci wektora liczb zespolonych typu double

**std::vector<std::complex<double> > ditfft2::fftwektor** () [inline], [override], [virtual]

Metoda przeciążająca metodę wirtualna w klasie **FFT**, służy do wyliczenia **FFT**

Reimplemented from **FFT** (p.).

---

The documentation for this class was generated from the following files:

- D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/FFT.h
- D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/ditfft2.cpp



# FFT Class Reference

Klasa bazowa **FFT**

```
#include <FFT.h>
```

Inheritance diagram for FFT:

## Public Member Functions

- **FFT** (std::vector< double > data, double rozmiar, int s)
- virtual std::vector< std::complex< double > > **fftwektor** ()  
*Metoda wirtualna zwracająca pusty wektor liczb zespolonych typu double*
- virtual std::vector< double > **IFFTwektor** ()  
*Metoda wirtualna zwracająca pusty wektor liczb typu double*
- virtual ~**FFT** ()  
*Wirtualny destruktor*

## Protected Attributes

- std::vector< double > **dane\_wejscowe**
- double **N**  
*Rozmiar danych wejściowych*
- int **s**  
*Krok*

---

## Detailed Description

Klasa bazowa **FFT**

---

## Constructor & Destructor Documentation

**virtual FFT::~~FFT ()** [*inline*], [*virtual*]

Wirtualny destruktor

---

## Member Function Documentation

**virtual std::vector<std::complex<double> > FFT::fftwektor ()** [*inline*], [*virtual*]

Metoda wirtualna zwracająca pusty wektor liczb zespolonych typu double

## Returns

Reimplemented in **ditfft2** (*p.*).

**virtual std::vector<double> FFT::IFFTwektor () [inline], [virtual]**

Metoda wirtualna zwracająca pusty wektor liczb typu double

## Returns

Reimplemented in **IFFT** (*p.*).

---

## Member Data Documentation

**double FFT::N [protected]**

Rozmiar danych wejściowych

**int FFT::s [protected]**

Krok

---

**The documentation for this class was generated from the following file:**

- D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/FFT.h

## file\_read Class Reference

Klasa odpowiedzialna za odczyt danych z pliku z rozszerzeniem .csv  
`#include <file_read.h>`

### Public Member Functions

- `std::vector< double > odczytplik (const std::string nazwa)`  
*Funkcja odczytująca dane z pliku*

### Private Attributes

- `std::fstream file`

---

### Detailed Description

Klasa odpowiedzialna za odczyt danych z pliku z rozszerzeniem .csv

---

### Member Function Documentation

`std::vector< double > file_read::odczytplik (const std::string nazwa)`

Funkcja odczytująca dane z pliku

#### Returns

dane w postaci wektora liczb typu double

---

The documentation for this class was generated from the following files:

- `D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/file_read.h`
- `D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/file_read.cpp`

## IFFT Class Reference

Klasa **IFFT** dziedzicząca publicznie po **FFT** odpowiedzialna jest za obliczenie odwrotnej transformacji fouriera  
`#include <FFT.h>`

Inheritance diagram for IFFT:

### Public Member Functions

- **IFFT** (std::vector< std::complex< double >> dane\_wejsciowe, double rozmiar)
- std::vector< double > **IFFTwyliczenie** (double **N**, const std::vector< std::complex< double >> **dane**)  
*Funkcja odpowiedzialna za obliczenie **IFFT***
- std::vector< double > **IFFTwektor** () override  
*Metoda przeciążająca metodę wirtualna w klasie **FFT**, służy do wyliczenie **IFFT***

### Private Attributes

- std::vector< std::complex< double > > **dane**  
*Dane w postaci sygnału po **FFT***
- double **N**  
*Rozmiar danych*

### Additional Inherited Members

---

### Detailed Description

Klasa **IFFT** dziedzicząca publicznie po **FFT** odpowiedzialna jest za obliczenie odwrotnej transformacji fouriera

---

### Member Function Documentation

**std::vector<double> IFFT::IFFTwektor ()** [`inline`], [`override`], [`virtual`]

Metoda przeciążająca metodę wirtualna w klasie **FFT**, służy do wyliczenie **IFFT**

#### Returns

Reimplemented from **FFT** (*p.*).

**std::vector< double > IFFT::IFFTwyliczenie** (double **N**, const std::vector< std::complex< double >> **dane**)

Funkcja odpowiedzialna za obliczenie **IFFT**

---

## Member Data Documentation

**std::vector<std::complex<double> > IFFT::dane [private]**

Dane w postaci sygnału po **FFT**

**double IFFT::N [private]**

Rozmiar danych

---

**The documentation for this class was generated from the following files:**

- D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/FFT.h
- D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/iff.cpp



## save\_to\_file Class Reference

Klasa odpowiedzialna za zapis danych bezpośrednio do pliku

```
#include <file_save.h>
```

### Public Member Functions

- `std::string Complex_toString (const std::complex< double > &dane)`  
*Funkcja inline, zamieniająca liczbę zespoloną na string'a*
- `void saveVector (const std::vector< double > wejscie, const std::string name, int N)`  
*Funkcja zapisująca dane po **IFFT** do pliku*
- `void saveVectorComplex (const std::vector< std::complex< double >> wejscie, const std::string name, int N)`  
*Funkcja zapisująca dane po **FFT** do pliku*

---

### Detailed Description

Klasa odpowiedzialna za zapis danych bezpośrednio do pliku

---

### Member Function Documentation

`std::string save_to_file::Complex_toString (const std::complex< double > & dane) [inline]`

Funkcja inline, zamieniająca liczbę zespoloną na string'a

`void save_to_file::saveVector (const std::vector< double > wejscie, const std::string name, int N)`

Funkcja zapisująca dane po **IFFT** do pliku

`void save_to_file::saveVectorComplex (const std::vector< std::complex< double >> wejscie, const std::string name, int N)`

Funkcja zapisująca dane po **FFT** do pliku

---

**The documentation for this class was generated from the following files:**

- `D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/file_save.h`
- `D:/Studia/Repo/7275b5c1-gr01-repo/Projekt/FourierProj/file_save.cpp`

# **Index**

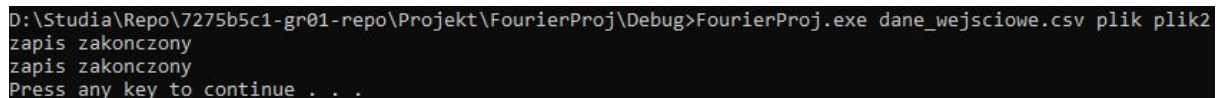
INDEX

## Dokumentacja zewnętrzna:

Będziemy potrzebowali pliku z sygnałem składającym się z liczb rzeczywistych, którego długość danych jest jakąś potęgą dwójki, w innym przypadku program nie zadziała i wyświetli odpowiedni komunikat. Plik ten musi zostać skopiowany do folderu z plikiem FourierProj.exe. Musi on zawierać sygnał w postaci próbek ułożonych w kolejnych kolumnach. Nie należy zapominać o formatowaniu tego pliku, musi być ono ustawione na UTF-8, aby uniknąć ewentualnych problemów związaną z konwersją danych wejściowych na wektor liczb typu double. W przypadku sygnału składającego się z liczb rzeczywistych separator dziesiętny musi zostać ustawiony na “.” W tym celu wchodzimy do ustawień excel’a, następnie do zakładki zaawansowane/advanced i zmieniamy wartość “Decimal separator” na “.”. Następnie klikamy ctrl+h i zamieniamy “,” na “.”.

Uruchomienie programu polega na wskazaniu w konsoli systemu windows odpowiedniej ścieżki do programu, oraz wpisanie odpowiednich zmiennych. Są to kolejno, nazwa pliku z rozszerzeniem.exe, plik .csv z którego wczytywane będą dane, nazwa pliku, do którego zapisane zostaną wyniki IFFT oraz nazwa pliku do którego zostaną zapisane wyniki FFT.

Przykładowe uruchomienie programu z wiersza poleceń przedstawione jest na poniższym obrazku:



```
D:\Studia\Repo\7275b5c1-gr01-repo\Projekt\FourierProj\Debug>FourierProj.exe dane_wejscowe.csv plik plik2
zapis zakonczony
zapis zakonczony
Press any key to continue . . .
```

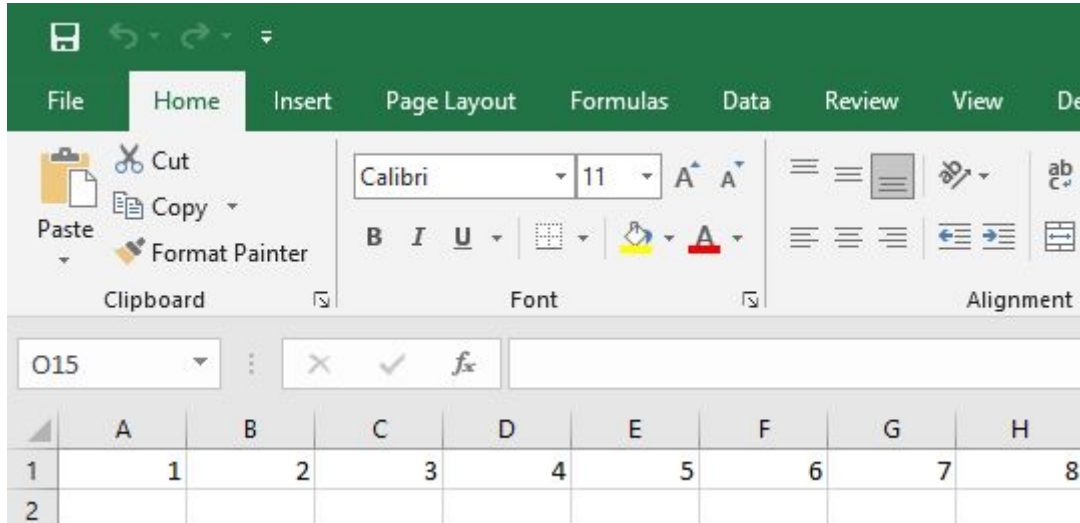
Napis “zapis zakończony”, jest w tym wypadku powtórzony dwa razy, ponieważ do każdego z plików udało się zapisać wyniki.

## Graficzna reprezentacja wyników:

W celu utworzenia reprezentacji graficznej wyników szybkiej transformacji fouriera, musimy otworzyć odpowiedni plik zawierający nasz sygnał po przejściu przez fft. W moim przypadku będzie to “nazwa1.csv”. Należy skopiować wszystkie liczby znajdujące się w nim. W następnym kroku należy otworzyć plik “SzablonFFT” oraz wkleić w odpowiednie pola (część rzeczywista oraz część zespolona) skopiowane wartości. Szablon powinien sam przeliczyć wszystkie wartości i odpowiedni wykres. Jediną modyfikowaną wartością w szablonie jest częstotliwość. Można ją ustawić wedle swojego uznania.

## Efekty działania programu

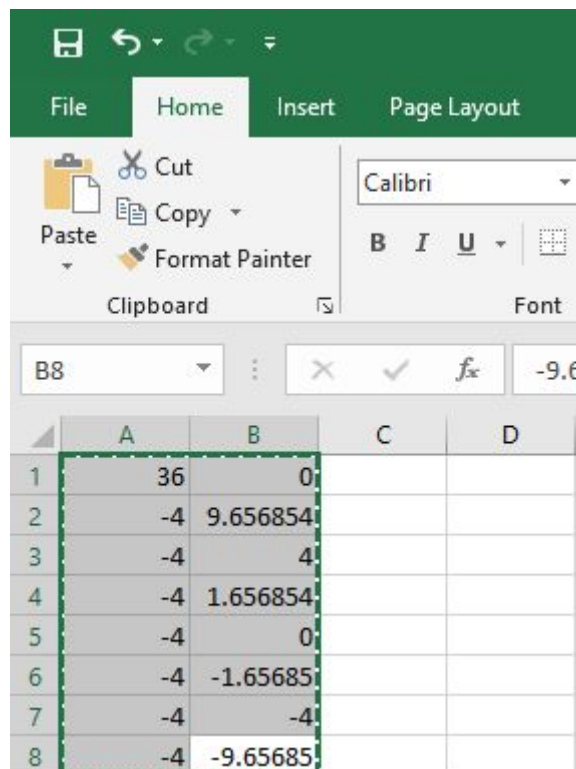
Poniżej zostaną zaprezentowane przykładowe wyniki dla danych wejściowych zaprezentowanych na zdjęciu poniżej:



	A	B	C	D	E	F	G	H
1	1	2	3	4	5	6	7	8
2								

Zdj. 4 Plik z danymi wejściowymi

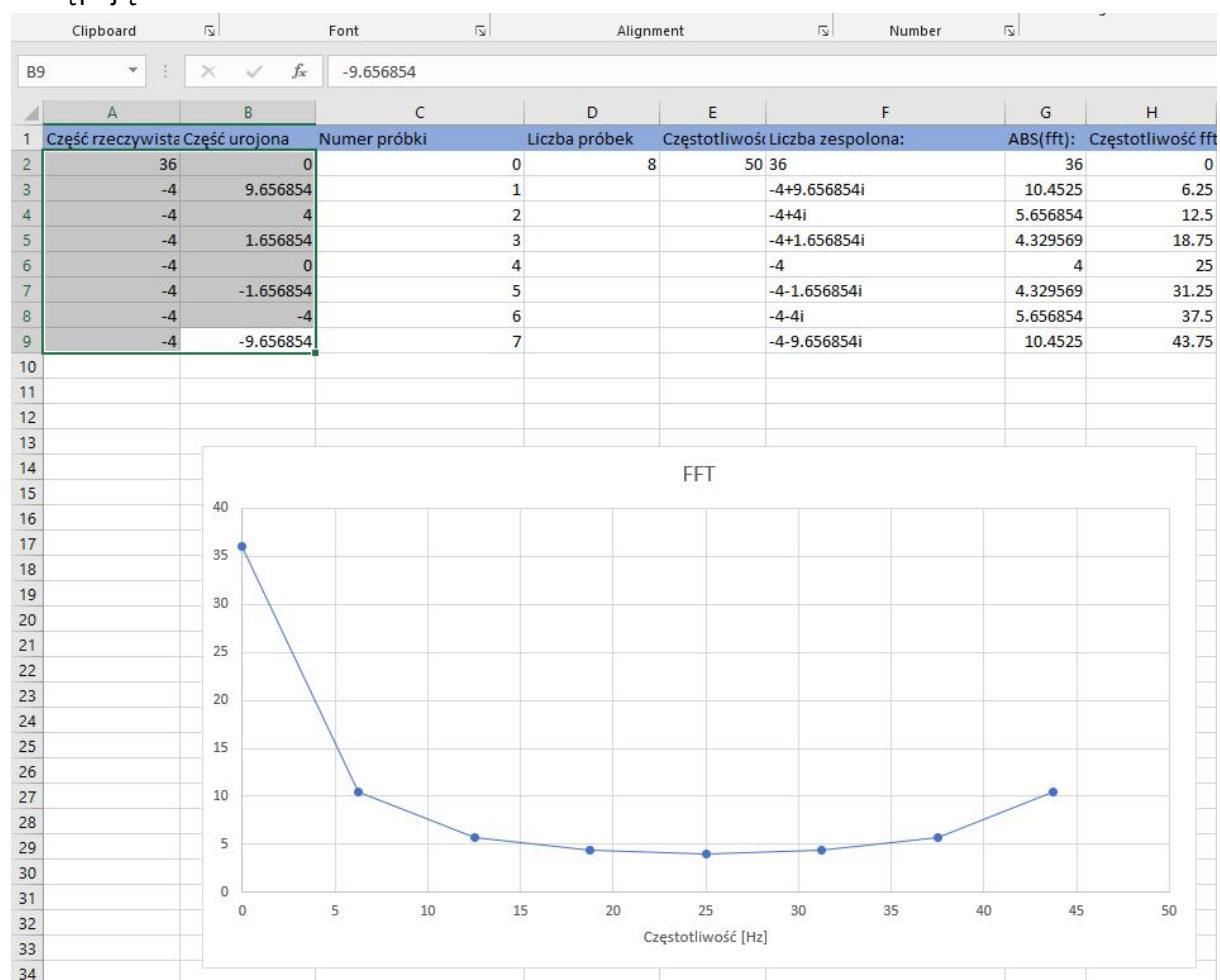
Po wykonaniu programu, w pliku z sygnałem po przejściu przez FFT znajdziemy następujące próbki sygnału:



	A	B	C	D
1	36	0		
2	-4	9.656854		
3	-4	4		
4	-4	1.656854		
5	-4	0		
6	-4	-1.65685		
7	-4	-4		
8	-4	-9.65685		

Zdj. 5 Sygnał po FFT

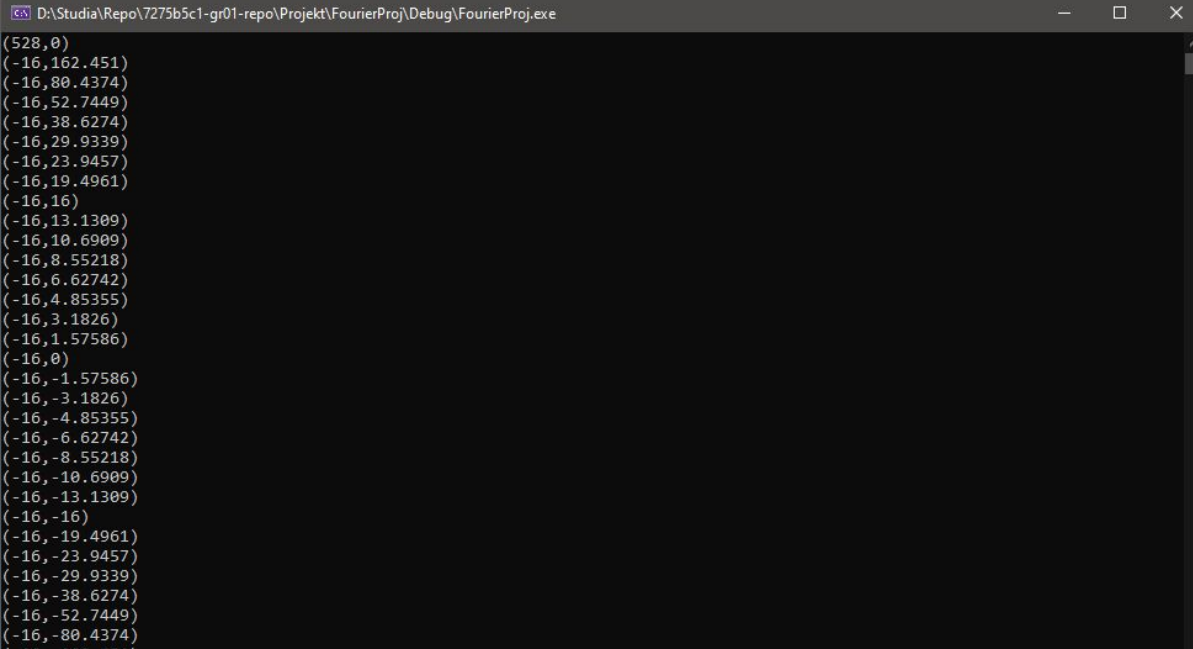
Po odpowiednich modyfikacjach i wklejeniu do szablonu, nasz sygnał prezentuje się następująco:



Zdj. 6 Szablon po wklejeniu wartości

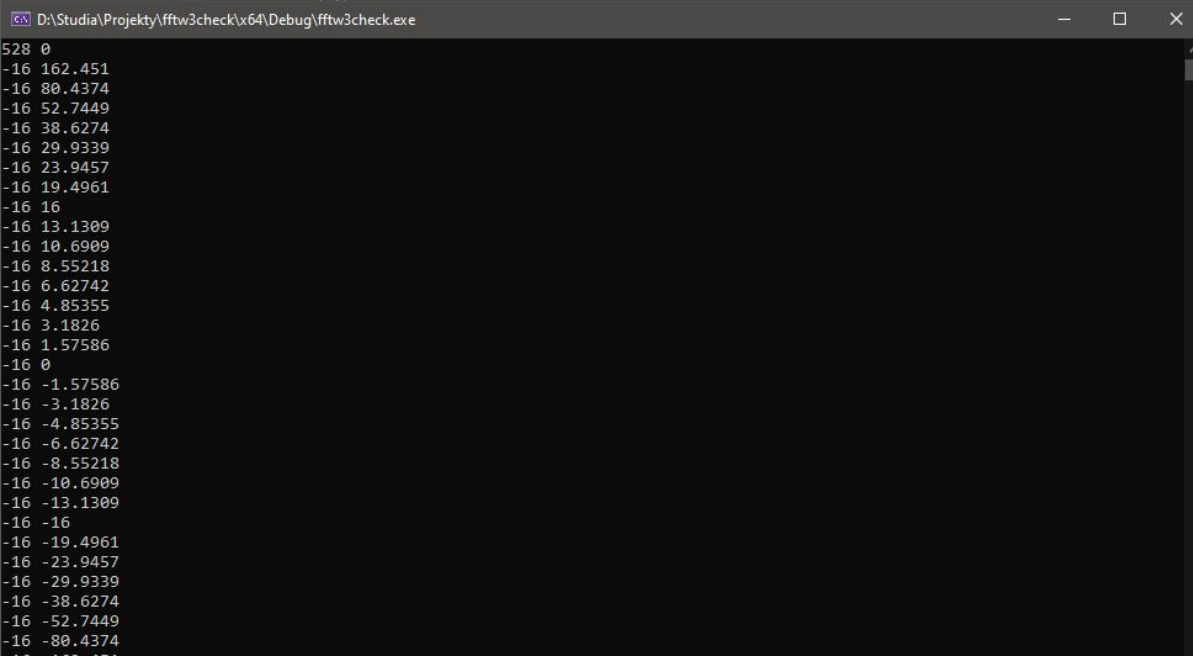
## Porównanie wyników z wynikami z biblioteki fftw3 oraz metody naiwnej.

Porównanie zostało przeprowadzone wielokrotnie, dla różnych sygnałów. Poniżej przedstawię porównanie wyników dla wektora o rozmiarze 32, zawierającego liczby naturalne od 1 do 32 włącznie.



```
D:\Studio\Repo\7275b5c1-gr01-repo\Projekt\FourierProj\Debug\FourierProj.exe
(528,0)
(-16,162.451)
(-16,80.4374)
(-16,52.7449)
(-16,38.6274)
(-16,29.9339)
(-16,23.9457)
(-16,19.4961)
(-16,16)
(-16,13.1309)
(-16,10.6909)
(-16,8.55218)
(-16,6.62742)
(-16,4.85355)
(-16,3.1826)
(-16,1.57586)
(-16,0)
(-16,-1.57586)
(-16,-3.1826)
(-16,-4.85355)
(-16,-6.62742)
(-16,-8.55218)
(-16,-10.6909)
(-16,-13.1309)
(-16,-16)
(-16,-19.4961)
(-16,-23.9457)
(-16,-29.9339)
(-16,-38.6274)
(-16,-52.7449)
(-16,-80.4374)
(-16,-162.451)
```

*Zdj.7 Wyniki FFT z mojego projektu*



```
D:\Studio\Projekty\fftw3check\x64\Debug\fftw3check.exe
528 0
-16 162.451
-16 80.4374
-16 52.7449
-16 38.6274
-16 29.9339
-16 23.9457
-16 19.4961
-16 16
-16 13.1309
-16 10.6909
-16 8.55218
-16 6.62742
-16 4.85355
-16 3.1826
-16 1.57586
-16 0
-16 -1.57586
-16 -3.1826
-16 -4.85355
-16 -6.62742
-16 -8.55218
-16 -10.6909
-16 -13.1309
-16 -16
-16 -19.4961
-16 -23.9457
-16 -29.9339
-16 -38.6274
-16 -52.7449
-16 -80.4374
-16 -162.451
```

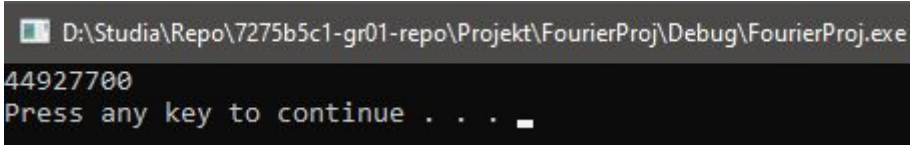
*Zdj.8 Wyniki FFT z biblioteki fftw3*

```
Konsola debugowania programu Microsoft Visual Studio
(528,0)
(-16,-162.451)
(-16,-80.4374)
(-16,-52.7449)
(-16,-38.6274)
(-16,-29.9339)
(-16,-23.9457)
(-16,-19.4961)
(-16,-16)
(-16,-13.1309)
(-16,-10.6909)
(-16,-8.55218)
(-16,-6.62742)
(-16,-4.85355)
(-16,-3.1826)
(-16,-1.57586)
(-16,-5.09849e-14)
(-16,1.57586)
(-16,3.1826)
(-16,4.85355)
(-16,6.62742)
(-16,8.55218)
(-16,10.6909)
(-16,13.1309)
(-16,16)
(-16,19.4961)
(-16,23.9457)
(-16,29.9339)
(-16,38.6274)
(-16,52.7449)
(-16,80.4374)
(-16,162.451)
```

*Zdj. 9 Wyniki FFT z wykorzystaniem metody naiwnej*

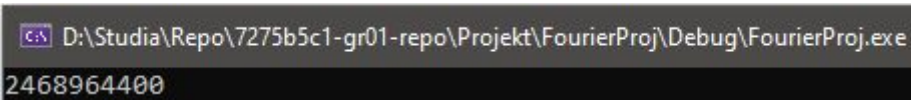
## Czasowe porównanie wydajności programu

Poniższe porównania czasu wykonania programu zostały zrealizowane dla 1024 próbek sygnału od 1:1024.



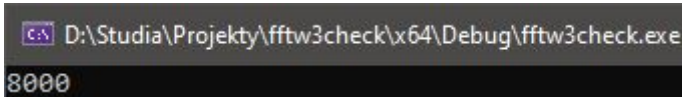
```
D:\Studia\Repo\7275b5c1-gr01-repo\Projekt\FourierProj\Debug\FourierProj.exe
44927700
Press any key to continue . . .
```

*Zdj. 10 Czas wykonania programu w [ns] (mój projekt)*



```
D:\Studia\Repo\7275b5c1-gr01-repo\Projekt\FourierProj\Debug\FourierProj.exe
2468964400
```

*Zdj. 11 Czas wykonania programu w [ns] (metoda naiwna)*



```
D:\Studia\Projekty\fftw3check\x64\Debug\fftw3check.exe
8000
```

*Zdj. 12 Czas wykonania programu w [ns] (z wykorzystaniem biblioteki fftw3)*



## Podsumowanie:

Program został zrealizowany zgodnie z paradygmatami programowania obiektowego, został również sprawdzony pod kątem wycieków pamięci. Wszystkie założenia projektowe udało się zrealizować. Największym problemem było szukanie wycieków pamięci, w moim przypadku program polecony przez prowadzącego definiowanego nagłówkiem `<vld.h>`, nie działał prawidłowo, nie wyświetlał żadnych wycieków pamięci, pomimo celowego wprowadzenia przeze mnie takowych wycieków. W momencie zamykania programu, wyświetlany był niezidentyfikowany błąd. W związku z tym błędy pamięci, były weryfikowane przez funkcję `_CrtDumpMemoryLeaks()`, która jest dostępna po zawarciu biblioteki `<crtdbh.h>`. Kolejnym problemem było nieprawidłowe działanie funkcji `std::stod()`, tak przynajmniej mi się wydawało. Miałem do czynienia z wieloma błędami związanymi z jej użyciem. Jak się później okazało problem tkwił zupełnie gdzie indziej. Okazało się że dane wejściowe były formatowane jako **UTF-8-BOM**, który powodował niezliczone błędy przy zamianie wartości typu `string` na `double`. Rozwiązaniem tego problemu było zmniejszenie formatowania na zwykłe UTF-8. Niewielkim problemem okazało się także porównanie wyników z wynikami metody naiwnej oraz biblioteki FFTW.