Raport Projekt 3

Aleksander Rorbach 203749 ACiR 3B

Edie Kulik 203578 ACiR 3A

1) Wizualizacja sygnału (plot_test.py)

```cpp
//implementacja klasy signal
Signal::Signal(double frequency, double faza, std::string signal_name)
    : f(frequency), name(signal_name) {
    samples = generate(f, faza, name);
}
Signal::Signal(std::string signal_name, const Fourier& transformata)
    : name(signal_name) {
    samples = idft(transformata);
}
Signal::Signal(const std::vector<double>& samples)
    : name("generated"), samples(samples) {
}
Signal::~Signal() {
    std::cout << "Signal destructor called for " << name << std::endl;
}
```

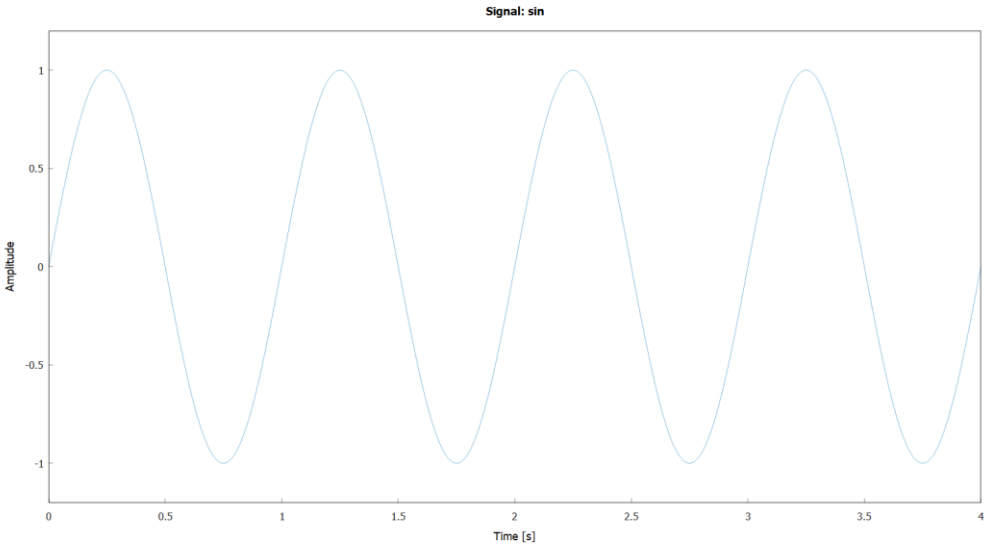
.

```cpp
// generuje sygnal, czestotliwosc, przesuniecie w fazie, typ sygnalu
std::vector<double> generate(double f, double faza, std::string name) {
    int sf = N;
    int n = N * T;
    std::vector<double> signal(n);

    for (int i = 0; i < n; i++) {
        double phase = (2 * PI * f * i) / sf + faza * PI / 180;
        if (name == "sin") signal[i] = sin(phase);
        else if (name == "cos") signal[i] = cos(phase);
        else if (name == "kwadrat") signal[i] = sin(phase) >= 0 ? 1 : -1;
        else if (name == "pila") signal[i] = (1 / PI) * (phase - PI * floor(phase / PI));
        else return {};
    }
    return signal;
}
```

```cpp
// rysowanie funkcji sygnalu w dziedzinie czasu
void plot_signal(Signal signal) {
    using namespace matplot;
    std::filesystem::create_directory("../raport");
    std::vector<double> t(signal.samples.size());
    double dt = T / signal.samples.size();
    for (size_t i = 0; i < t.size(); i++) t[i] = i * dt;
    xlim({ 0, T - 1 });
    ylim({ -1.2, 1.2 });
    plot(t, signal.samples);
    xlabel("Time [s]");
    ylabel("Amplitude");
    title("Signal: " + signal.name);
    save("../raport/" + signal.name + "_signal.png");
    show();
}
```

Anaconda Prompt - python plot_test.py

```
(base) C:\Users\aleks>activate cmake_env

(cmake_env) C:\Users\aleks>cd C:\Users\aleks\Desktop\TP-3\CMakeProject1\out\src\Debug

(cmake_env) C:\Users\aleks\Desktop\TP-3\CMakeProject1\out\src\Debug>python plot_test.py
Podaj częstotliwość sygnału: 1
Podaj fazę sygnału (w stopniach): 0
Podaj typ sygnału (sin, cos, kwadrat, pila): sin
Press ENTER to continue...
```



2)DFT i IDFT (idft_test.py)

- Implementacja klasy Signal (kontener danych sygnałowych) - jak wyżej

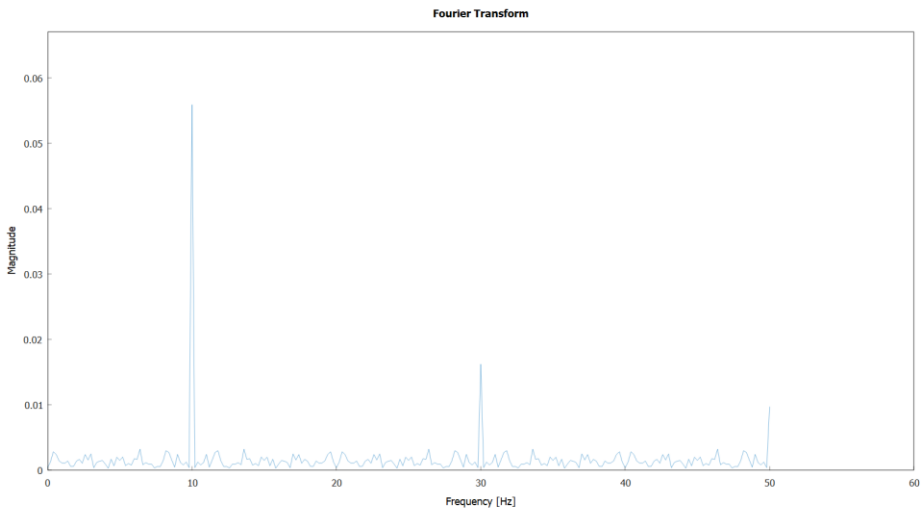
```cpp
//implementacja klasy fourier
Fourier::Fourier(const Signal& signal) {
    X = dft(signal);
}
Fourier::~Fourier() {}
```

```cpp
//dft
std::vector<std::complex<double>> dft(Signal signal) {
    std::vector<std::complex<double>> X(signal.samples.size());
    const double scale = 1.0 / std::sqrt(signal.samples.size());
    for (int k = 0; k < signal.samples.size(); k++) {
        X[k] = std::complex<double>(0.0, 0.0);
        for (int n = 0; n < signal.samples.size(); n++) {
            double angle = 2.0 * PI * k * n / signal.samples.size();
            X[k] += signal.samples[n] * std::polar(1.0, -angle);
        }
        X[k] *= scale;
    }
    return X;
}

//idft
std::vector<double> idft(Fourier fourier) {
    if (fourier.X.empty()) return {};
    const std::vector<std::complex<double>> X = fourier.X;
    std::vector<double> samples(X.size());
    for (int n = 0; n < X.size(); n++) {
        std::complex<double> sum(0.0, 0.0);
        for (int k = 0; k < X.size(); k++) {
            sum += X[k] * std::polar(1.0, 2.0 * PI * k * n / X.size());
        }
        samples[n] = std::real(sum);
    }
    double max_val = *std::max_element(samples.begin(), samples.end());
    for (double& val : samples) val /= max_val;
    return samples;
}
```

```cpp
//rysowanie fouriera
void plot_fourier(Fourier fourier) {
    using namespace matplot;
    std::filesystem::create_directory("../raport");
    size_t M = fourier.X.size();
    double fs = N, df = fs / M;
    size_t H = M / 2 + 1;
    std::vector<double> f(H), mag(H);
    for (size_t k = 0; k < H; ++k) {
        f[k] = k * df;
        mag[k] = 2.0 * std::abs(fourier.X[k]) / M;
    }
    xlim({ 0, *std::max_element(f.begin(), f.end()) * 1.2 });
    ylim({ 0, *std::max_element(mag.begin(), mag.end()) * 1.2 });
    plot(f, mag);
    xlabel("Frequency [Hz]");
    ylabel("Magnitude");
    title("Fourier Transform");
    save("../raport/Fourier_transform.png");
    show();
}
```

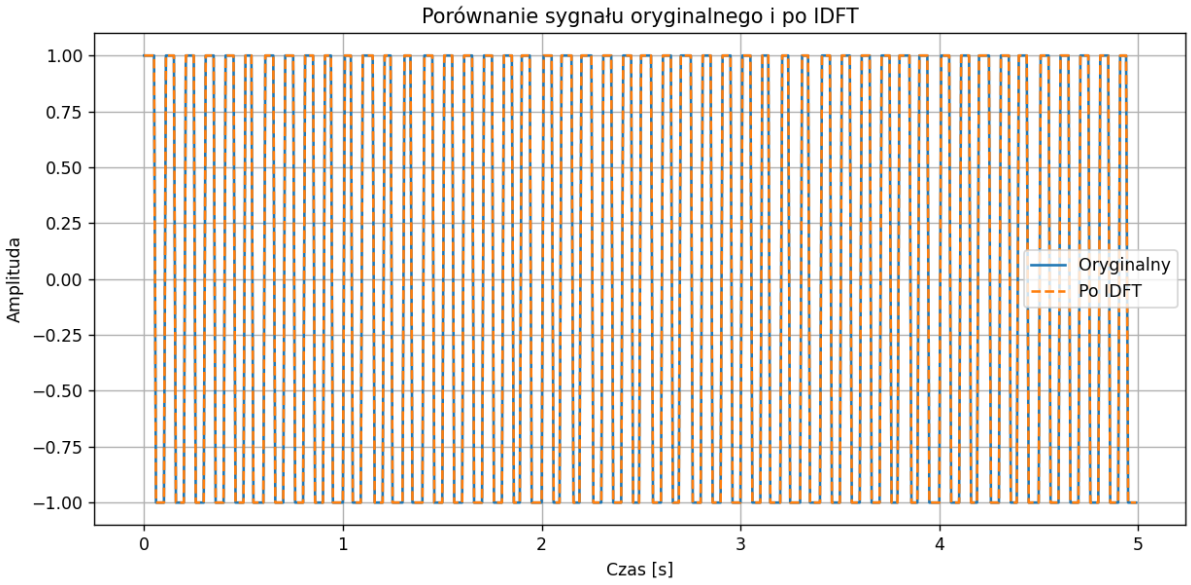
DFT (plot_test.py) - DFT dla sygnału prostokątnego 10Hz

```
Anaconda Prompt - python  plot_test.py

(base) C:\Users\aleks>activate cmake_env

(cmake_env) C:\Users\aleks>cd C:\Users\aleks\Desktop\TP-3\CMakeProject1\out\src\Debug

(cmake_env) C:\Users\aleks\Desktop\TP-3\CMakeProject1\out\src\Debug>python plot_test.py
Podaj częstotliwość sygnału: 10
Podaj fazę sygnału (w stopniach): 0
Podaj typ sygnału (sin, cos, kwadrat, pila): kwadrat
Press ENTER to continue...
Signal destructor called for kwadrat
Czy chcesz wykonać filtrację sygnału? (t/n): n
Czy chcesz wykonać transformatę Fouriera? (t/n): t
Signal destructor called for kwadrat
Signal destructor called for kwadrat
Press ENTER to continue...
```



IDFT (idft_test.py) - IDFT dla tego samego sygnału

```
(cmake_env) C:\Users\aleks\Desktop\TP-3\CMakeProject1\out\src\Debug>python idft_test.py
Podaj częstotliwość (Hz): 10
Podaj typ sygnału (sin/cos/kwadrat/pila): kwadrat
Signal destructor called for kwadrat
Signal destructor called for kwadrat
```

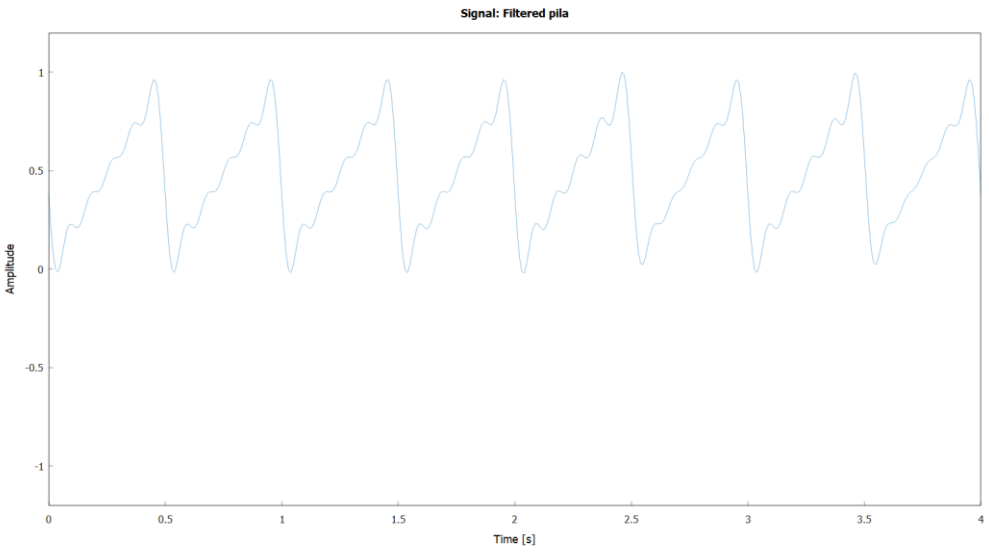


3)Filtracja (plot_test.py)

```cpp
//filtracja 1d
Signal filter(Signal signal, double cutoff_hz) {
    Fourier transformata(signal);
    int M = signal.samples.size();
    double fs = N;
    double df = fs / M;
    for (int k = 0; k < M; ++k) {
        double freq = (k >= M / 2) ? (k - M) * df : k * df;
        if (std::abs(freq) > cutoff_hz) transformata.X[k] = { 0.0, 0.0 };
    }
    return Signal("Filtered " + signal.name, transformata);
}
```

```cpp
//filtracja 2d z uzyciem jadra macierzy kernel
std::vector<std::vector<double>> filter2D(const std::vector<std::vector<double>>& data, const std::vector<std::vector<double>>& kernel) {
    int rows = data.size();
    int cols = data[0].size();
    int krows = kernel.size();
    int kcols = kernel[0].size();
    int kr = krows / 2;
    int kc = kcols / 2;

    std::vector<std::vector<double>> output(rows, std::vector<double>(cols, 0.0));

    for (int i = kr; i < rows - kr; ++i) {
        for (int j = kc; j < cols - kc; ++j) {
            double sum = 0.0;
            for (int m = 0; m < krows; ++m) {
                for (int n = 0; n < kcols; ++n) {
                    int x = i + m - kr;
                    int y = j + n - kc;
                    sum += data[x][y] * kernel[m][n];
                }
            }
            output[i][j] = sum;
        }
    }
    return output;
}
```

```
(cmake_env) C:\Users\aleks\Desktop\TP-3\CMakeProject1\out\src\Debug>python plot_test.py
Podaj częstotliwość sygnału: 1
Podaj fazę sygnału (w stopniach): 0
Podaj typ sygnału (sin, cos, kwadrat, pila): pila
Press ENTER to continue...
Signal destructor called for pila
Czy chcesz wykonać filtrację sygnału? (t/n): t
Podaj częstotliwość odcięcia [Hz]: 10
```

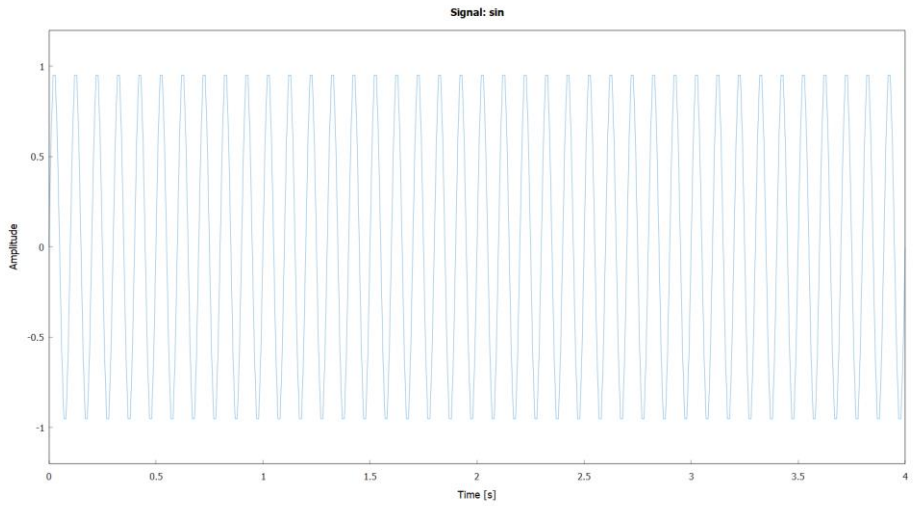


4)Generowanie sygnałów o zadanej częstotliwości (plot_test.py)

- implementacja klasy signal,
- generowanie sygnału o zadanej częstotliwości, fazie i kształcie
- wizualizacja sygnału

Jak w podpunkcie 1.

a) Sin 10Hz



b) Cos 0.1Hz