Techniki programowania projekt 3

Mateusz Dadura 203156
Karol Betlejewski 203232

wykorzystane biblioteki:

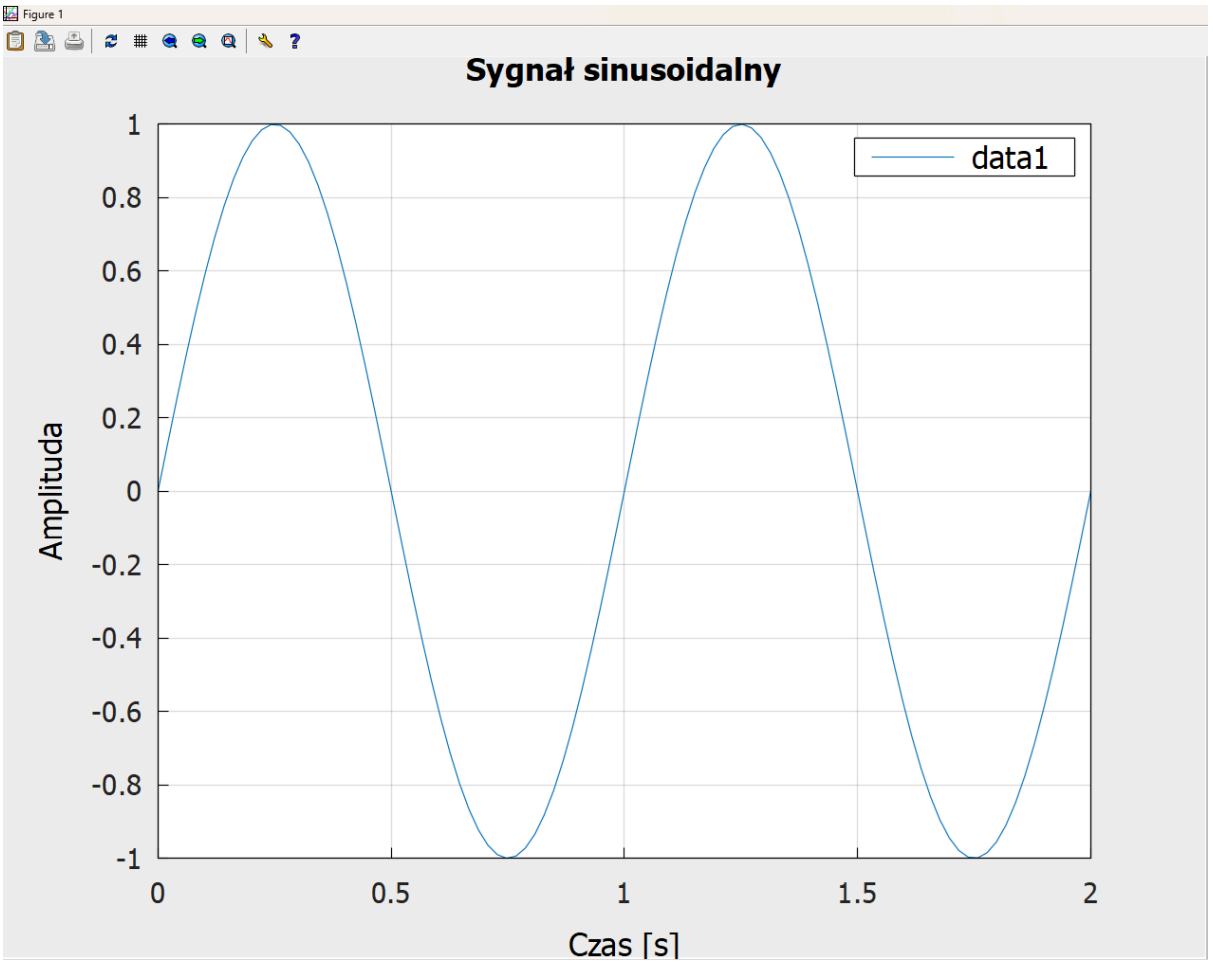
- matplotplusplus
- pybind11

w projekcie zaimplementowano w bibliotece:
-generowanie syganałów o zadanej częstotliwości
-wizualizacje sygnałów z wykorzystaniem biblioteki matplotplusplus
-DFT i odwrotną DFT
-filtracje 1D i 2D

Rodzaje generowanych sygnałów (razem z wizualizacją matplot):

W projekcie zaimplementowano funkcje generujące cztery podstawowe typy sygnałów jednowymiarowych (1D), które są często używane w analizie i przetwarzaniu sygnałów:

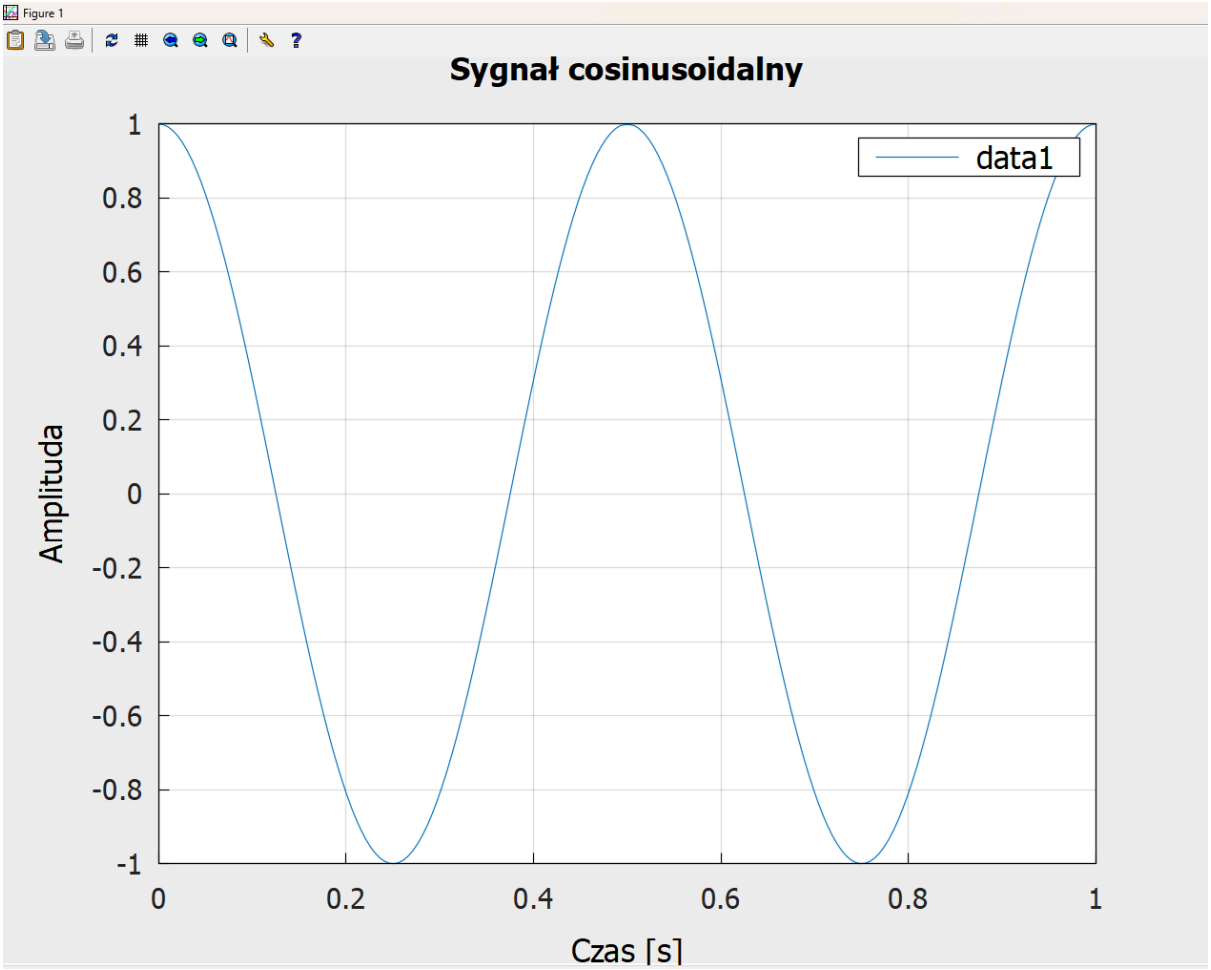
1. Sygnał sinusoidalny (`Sinus`)



```
void Sinus(double frequency, double start_time, double end_time, int num_samples) {
    using namespace matplot;

    if (num_samples <= 1 || end_time <= start_time) {
        cerr << "Nieprawidłowe dane wejściowe.\n";
        return;
    }

    vector<double> t(num_samples);
    double dt = (end_time - start_time) / (num_samples - 1);
    for (int i = 0; i < num_samples; ++i)
        t[i] = start_time + i * dt;

    vector<double> y(num_samples);
    for (int i = 0; i < num_samples; ++i)
        y[i] = sin(2 * pi * frequency * t[i]);

    plot(t, y);
    title("Sygnał sinusoidalny");
    xlabel("Czas [s]");
    ylabel("Amplituda");
    grid(on);
    show();
}
```

Funkcja `Sinus` generuje i wizualizuje dyskretny sygnał sinusoidalny w języku C++ z użyciem biblioteki Matplot++. Przyjmuje cztery parametry: częstotliwość sygnału (`frequency`), czas początkowy (`start_time`), czas końcowy (`end_time`) oraz liczbę próbek (`num_samples`). Po sprawdzeniu poprawności danych wejściowych (czy liczba próbek jest większa niż 1 i czy czas końcowy jest większy od początkowego), funkcja tworzy wektor `t`, który reprezentuje kolejne momenty czasowe, równomiernie rozłożone w zadanym przedziale czasu. Następnie dla każdego punktu w czasie obliczana jest wartość funkcji sinusoidalnej `y[i] = sin(2πft[i])`, tworząc w ten sposób próbki sygnału sinusoidalnego. Na końcu funkcja używa Matplot++ do narysowania wykresu sygnału: na osi X przedstawiony jest czas, a na osi Y – amplituda. Dodane są także etykiety osi, tytuł wykresu i siatka pomocnicza, a całość zostaje wyświetlona za pomocą `show()`. Funkcja służy do demonstracji podstawowych właściwości sygnału sinusoidalnego w domenie czasu.

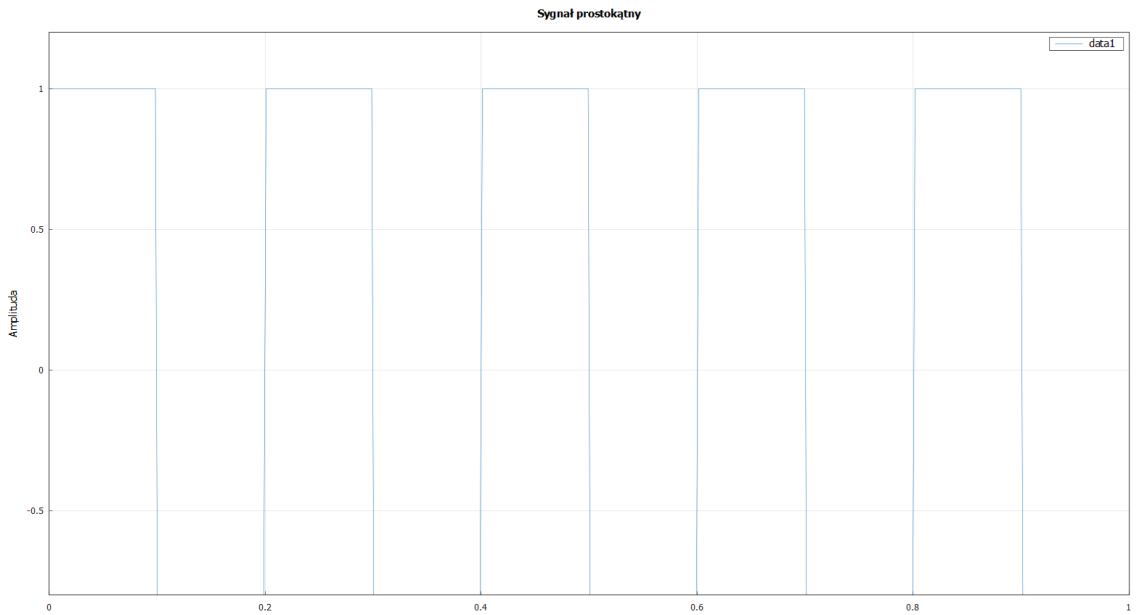
2. Sygnał cosinusoidalny (`Cosinus`)



Funkcja `Cosinus` w języku C++ generuje i wyświetla sygnał cosinusoidalny przy użyciu biblioteki Matplot++. Przyjmuje cztery argumenty: częstotliwość sygnału (`frequency`), czas początkowy (`start_time`), czas końcowy (`end_time`) oraz liczbę próbek (`num_samples`). Na początku funkcja sprawdza poprawność danych wejściowych – liczba próbek musi być większa od 1, a czas końcowy musi przewyższać początkowy. Następnie tworzy wektor `t` zawierający równomiernie rozłożone punkty czasowe w zadanym przedziale, z krokiem obliczanym jako `(end_time - start_time) / (num_samples - 1)`. Dla każdego punktu czasu wyznaczana jest odpowiadająca wartość funkcji cosinusoidalnej zgodnie ze wzorem `y[i] = cos(2πft[i])`, tworząc próbki sygnału. Za pomocą Matplot++ funkcja rysuje wykres amplitudy w funkcji czasu, nadaje tytuł wykresowi („Sygnał cosinusoidalny”), oznacza osie jako „Czas [s]” i „Amplituda”, włącza siatkę pomocniczą i wyświetla wynikowy wykres. Funkcja ta służy do analizy zachowania sygnału cosinusoidalnego w dziedzinie czasu.

```
void Cosinus(double frequency, double start_time, double end_time, int num_samples) {
    using namespace matplot;

    if (num_samples <= 1 || end_time <= start_time) {
        cerr << "Nieprawidłowe dane wejściowe.\n";
        return;
    }

    vector<double> t(num_samples);
    double dt = (end_time - start_time) / (num_samples - 1);
    for (int i = 0; i < num_samples; ++i)
        t[i] = start_time + i * dt;

    vector<double> y(num_samples);
    for (int i = 0; i < num_samples; ++i)
        y[i] = cos(2 * pi * frequency * t[i]);

    plot(t, y);
    title("Sygnał cosinusoidalny");
    xlabel("Czas [s]");
    ylabel("Amplituda");
    grid(on);
    show();
}
```

3. Sygnał prostokątny (`Rectangular`)

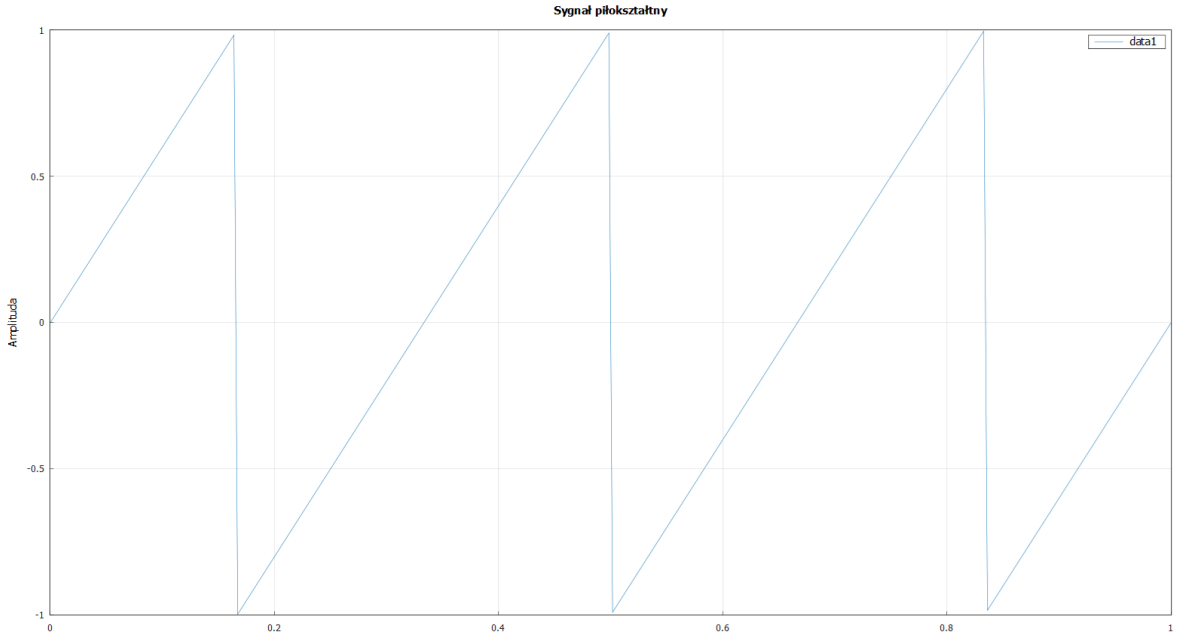


Funkcja `Rectangular` w języku C++ generuje i wyświetla wykres sygnału prostokątnego przy użyciu biblioteki Matplot++. Działa w oparciu o sinusoidę, przekształcając jej wartości w przebieg binarny: dodatnie i zerowe wartości przyjmują wartość 1, a ujemne – wartość -1.

Funkcja przyjmuje cztery argumenty: częstotliwość sygnału, czas początkowy i końcowy oraz liczbę próbek, które mają zostać wygenerowane. Po sprawdzeniu poprawności parametrów wejściowych tworzony jest wektor `t` reprezentujący chwile czasowe, równomiernie rozłożone między czasem początkowym a końcowym. Następnie w pętli obliczana jest wartość funkcji sinusoidalnej dla każdej chwili czasowej i na tej podstawie przypisywana jest wartość 1 lub -1, w zależności od znaku tej funkcji. Wyniki zapisywane są w wektorze `y`, który reprezentuje wartości sygnału prostokątnego. Na końcu wykorzystywana jest biblioteka Matplot++ do narysowania wykresu: na osi X umieszczony jest czas, a na osi Y wartość amplitudy sygnału. Dodawane są także tytuł wykresu, podpisy osi oraz siatka, po czym wykres jest wyświetlany. Cały proces umożliwia wygenerowanie i wizualizację dyskretnego sygnału prostokątnego o zadanej częstotliwości i czasie trwania.

```
void Rectangular(double frequency, double start_time, double end_time, int num_samples) {
    using namespace matplot;

    if (num_samples <= 1 || end_time <= start_time) {
        cerr << "Nieprawidłowe dane wejściowe.\n";
        return;
    }

    vector<double> t(num_samples);
    double dt = (end_time - start_time) / (num_samples - 1);
    for (int i = 0; i < num_samples; ++i)
        t[i] = start_time + i * dt;

    vector<double> y(num_samples);
    for (int i = 0; i < num_samples; ++i) {
        double val = sin(2 * pi * frequency * t[i]);
        y[i] = (val >= 0 ? 1.0 : -1.0);
    }

    plot(t, y);
    title("Sygnał prostokątny");
    xlabel("Czas [s]");
    ylabel("Amplituda");
    grid(on);
    show();
}
```

4. Sygnał piłokształtny (`Sawtooth`)



Funkcja `Sawtooth` generuje i wizualizuje dyskretny sygnał piłokształtny (ang. *sawtooth wave*) przy użyciu biblioteki Matplot++ w języku C++. Przyjmuje cztery parametry: częstotliwość sygnału, czas początkowy, czas końcowy oraz liczbę próbek. Najpierw sprawdzana jest poprawność danych wejściowych — jeśli liczba próbek jest mniejsza lub równa 1 albo czas końcowy jest mniejszy bądź równy początkowemu, funkcja wypisuje komunikat o błędzie i kończy działanie. Następnie tworzony jest wektor `t`, w którym obliczane są chwile czasowe dla równomiernie rozmieszczonych próbek na zadanym przedziale czasu. Główna część funkcji oblicza wartości sygnału piłokształtnego według wzoru:

`2.0 * (frequency * t[i] - floor(frequency * t[i] + 0.5))`,

co generuje znormalizowany kształt piły o wartościach w przedziale od -1 do 1, zgodnie z klasyczną definicją funkcji piłokształtnej. Ostatecznie obliczone wartości y są rysowane względem czasu `t` na wykresie. Funkcja dodaje tytuł wykresu, podpisy osi oraz siatkę, a na końcu wyświetla wynik. Całość umożliwia analizę kształtu sygnału piłokształtnego w dziedzinie czasu dla zadanych parametrów.

```
void Sawtooth(double frequency, double start_time, double end_time, int num_samples) {
    using namespace matplot;

    if (num_samples <= 1 || end_time <= start_time) {
        cerr << "Nieprawidłowe dane wejściowe.\n";
        return;
    }

    vector<double> t(num_samples);
    double dt = (end_time - start_time) / (num_samples - 1);
    for (int i = 0; i < num_samples; ++i)
        t[i] = start_time + i * dt;

    vector<double> y(num_samples);
    for (int i = 0; i < num_samples; ++i) {
        double val = 2.0 * (frequency * t[i] - floor(frequency * t[i] + 0.5));
        y[i] = val;
    }

    plot(t, y);
    title("Sygnał piłokształtny");
    xlabel("Czas [s]");
    ylabel("Amplituda");
    grid(on);
    show();
}
```

Transformacja Fouriera (DFT i IDFT)

1. DFT(const vector<double>& input)

Funkcja ta oblicza dyskretną transformatę Fouriera sygnału wejściowego input, będącego wektorem liczb rzeczywistych. Dla każdej częstotliwości k (0 do N-1) obliczana jest suma zespolona, która reprezentuje wpływ tej częstotliwości w sygnale. Wynik zapisywany jest do wektora liczb zespolonych out, który jest wypisywany na standardowe wyjście jako widmo amplitudowe. Składnik urojony bardzo bliski zeru (< 1e-4) jest zerowany dla przejrzystości.

2. I_DFT(const vector<Complex>& spectrum)

Funkcja ta realizuje odwrotną dyskretną transformatę Fouriera na podstawie podanego widma zespolonego. Dla każdej próbki czasu n rekonstruowany jest sygnał w dziedzinie czasu jako suma wszystkich składników widma pomnożonych przez odpowiednią funkcję wykładniczą. Wynikiem jest ciąg wartości rzeczywistych reprezentujący odtworzony sygnał.

3. FiltreLowFrequencies(vector<Complex>& out, double threshold)

Funkcja filtrująca sygnał w dziedzinie częstotliwości. Usuwa (zeruje) składniki widma odpowiadające niskim częstotliwościom na podstawie progu threshold, który określa ułamek długości widma do wyzerowania. Filtracja odbywa się symetrycznie względem środka widma, zgodnie z parzystością DFT dla sygnałów rzeczywistych.

4. I_DFTFiltre(const vector<Complex>& spectrum)

To uproszczona wersja I_DFT, przeznaczona do użytku po filtracji widma. Odtwarza sygnał w dziedzinie czasu na podstawie zmodyfikowanego (przefiltrowanego) widma i wypisuje go.

5. DFTFiltre_Reversed(const vector<double>& signal)

Jest to funkcja kompleksowa, która realizuje pełny proces przetwarzania sygnału:
– oblicza jego DFT,
– filtruje widmo za pomocą FiltreLowFrequencies (domyślnie usuwa 20% najniższych częstotliwości),
– a następnie rekonstruuje sygnał w dziedzinie czasu za pomocą I_DFTFiltre.

Filtracja średnią ruchomą:

Filtracja 1D – funkcja Filter1D

Ta funkcja realizuje filtrację 1D typu "średnia ruchoma" (ang. moving average) na jednowymiarowym sygnale zapisanym jako vector<double> signal.

Sprawdzenie poprawności danych wejściowych

-Rozmiar okna filtracji window_size musi być dodatni i nieparzysty.
-Długość sygnału musi być większa lub równa window_size.

Dla każdej próbki sygnału:

-Obliczana jest średnia wartość sąsiednich próbek w zakresie [-offset, offset], gdzie offset = window_size / 2.

-Jeśli część okna wykracza poza granice sygnału, pomijane są te indeksy (czyli brzegowe punkty traktowane są łagodnie).

-Średnia zapisywana jest do nowego wektora filtered.

Wizualizacja z użyciem Matplot++:

-Rysowany jest wykres oryginalnego sygnału i przefiltrowanego.
-Dodany tytuł, etykiety osi, legenda oraz siatka.

Filtracja 2D – funkcja Filter2D + TestFilter2D

Ta część kodu realizuje filtrację obrazu 2D (np. w formie macierzy) poprzez uśrednianie wartości w sąsiedztwie każdego piksela.

Funkcja Filter2D:

Sprawdzenie poprawności:
-window_size musi być nieparzysty i ≥ 1.
-Obraz (image) nie może być pusty i musi mieć wymiary większe niż okno filtracji.

-Inicjalizacja wynikowej macierzy filtered
O tych samych wymiarach co image, wypełniona zerami.

Dla każdego piksela (i,j):
-Sumowane są wszystkie wartości pikseli znajdujących się w oknie window_size × window_size wokół (i,j).
-Wartość średnia zapisywana jest w filtered[i][j].
-Dla pikseli przy brzegach niepełne okna są uwzględniane selektywnie – czyli nie wykraczają poza ramy obrazu.

Funkcja TestFilter2D:

Tworzy przefiltrowany obraz z użyciem Filter2D,

Wypisuje na konsolę:
-Oryginalny obraz (macierz liczb).
-Obraz po filtracji.