# Lab 4 – Przestrzenie wektorowe i algebra linowa

Arkadiusz Kurnik, Jan Cichoń

Zadanie 1:

```python
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.integrate import quad

# Funkcja e^x
def f(x):
    return np.exp(x)

# Przybliżenie Taylora dla e^x (n = 5)
def taylor_approximation(x, n):
    approximation = np.zeros_like(x)
    for i in range(n+1):
        approximation += (x**i) / math.factorial(i)
    return approximation

# Aproksymacja szeregiem Fouriera
def fourier_series_approximation(func, n_terms, samples=1000, interval=(-np.pi, np.pi)):
    a, b = interval
    L = (b - a) / 2
    x_values = np.linspace(a, b, samples)

    a0 = (1/L) * quad(lambda x: func(x), a, b)[0]

    an = []
    bn = []
    for n in range(1, n_terms+1):
        an_coeff = (1/L) * quad(lambda x: func(x) * np.cos(n*np.pi*x/L), a, b)[0]
        bn_coeff = (1/L) * quad(lambda x: func(x) * np.sin(n*np.pi*x/L), a, b)[0]
        an.append(an_coeff)
        bn.append(bn_coeff)

    approximation = a0/2 * np.ones_like(x_values)
    for n in range(1, n_terms+1):
        approximation += an[n-1] * np.cos(n*np.pi*x_values/L) + bn[n-1] * np.sin(n*np.pi*x_values/L)

    return x_values, approximation
```

```python
# Wielomiany Czebyszewa
def chebyshev_polynomials(x, degree):
    T = np.zeros((degree + 1, len(x)))
    T[0] = np.ones_like(x)
    if degree > 0:
        T[1] = x
    for n in range(2, degree + 1):
        T[n] = 2 * x * T[n - 1] - T[n - 2]
    return T

# WLS z Czebyszewem
def wls_approximation_chebyshev(x, y, degree):
    T = chebyshev_polynomials(x, degree).T
    W = np.diag(np.ones_like(x))
    A = T.T @ W @ T
    b = T.T @ W @ y
    coeffs = np.linalg.solve(A, b)
    y_approx = T @ coeffs
    return y_approx

# WLS z klasycznymi wielomianami
def wls_approximation_polynomial(x, y, degree):
    X = np.vander(x, degree + 1, increasing=True)
    W = np.diag(np.ones_like(x))
    A = X.T @ W @ X
    b = X.T @ W @ y
    coeffs = np.linalg.solve(A, b)
    y_approx = X @ coeffs
    return y_approx
```

```python
# Funkcje wagowe
def constant_weight(x):
    return np.ones_like(x)

def triangle_weight(x):
    return (1 - np.abs(x)) * 1000

def v_shape_weight(x):
    return 1000 - (1 - np.abs(x)) * 1000

def wls_with_weight(x, y, degree, weight_func, basis='polynomial'):
    W = np.diag(weight_func(x))

    if basis == 'polynomial':
        X = np.vander(x, degree + 1, increasing=True)
    elif basis == 'chebyshev':
        X = chebyshev_polynomials(x, degree).T
    else:
        raise ValueError("basis must be 'polynomial' or 'chebyshev'")

    A = X.T @ W @ X
    b = X.T @ W @ y
    coeffs = np.linalg.solve(A, b)
    y_approx = X @ coeffs
    return y_approx

weights = {
    "Stała": constant_weight,
    "Trójkątna": triangle_weight,
    "Trójkątna odwrotna": v_shape_weight
}

# Zakres x w [-π, π]
x = np.linspace(-np.pi, np.pi, 400)

# Obliczenie funkcji i przybliżeń
y_exact = f(x)
y_taylor = taylor_approximation(x, 5)
```

```python
# Fourier
n_terms = 10
samples = 1000
a, b = -np.pi, np.pi
x_fourier, y_fourier = fourier_series_approximation(f, n_terms, samples, (a, b))

# WLS - przeskalowanie x do [-1, 1]
x_scaled = x / np.pi
y_scaled = f(x)

# WLS: Chebyshev i klasyczny wielomian
y_wls_chebyshev = wls_approximation_chebyshev(x_scaled, y_scaled, degree=5)
y_wls_poly = wls_approximation_polynomial(x_scaled, y_scaled, degree=5)

# Błędy
error_taylor = np.abs(y_exact - y_taylor)
error_fourier = np.abs(y_exact - np.interp(x, x_fourier, y_fourier))
error_wls_chebyshev = np.abs(y_exact - y_wls_chebyshev)
error_wls_poly = np.abs(y_exact - y_wls_poly)

# Wykres funkcji i aproksymacji
plt.figure(figsize=(10, 6))
plt.plot(x, y_exact, label='e^x', linewidth=2)
plt.plot(x, y_taylor, label="Taylor (n=5)", color="red", linestyle="--", linewidth=2)
plt.plot(x_fourier, y_fourier, label=f"Fourier (n = {n_terms})", color="green", linestyle=":", linewidth=2)
plt.plot(x, y_wls_poly, label="$x^n$ (n=5)", color="orange", linestyle="dashdot", linewidth=2)
plt.plot(x, y_wls_chebyshev, label="Chebyshev (n=5)", color="purple", linestyle="-.", linewidth=2)
plt.title("Funkcja $e^x$ i jej aproksymacje")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()

# --- Wykresy błędów na wspólnym rysunku z 2 subplotami ---
fig, axs = plt.subplots(1, 2, figsize=(16, 6), sharex=True)
```

```python
# Pełny zakres błędów
axs[0].plot(x, error_taylor, label="Taylor (n=5)", color="red", linestyle="--")
axs[0].plot(x, error_fourier, label="Fourier (n=10)", color="green", linestyle=":")
axs[0].plot(x, error_wls_poly, label="$x^n$ (n=5)", color="orange", linestyle="dashdot")
axs[0].plot(x, error_wls_chebyshev, label="Chebyshev (n=5)", color="purple", linestyle="-.")
axs[0].set_title("Błąd aproksymacji – pełny zakres")
axs[0].set_xlabel("x")
axs[0].set_ylabel("Błąd bezwzględny")
axs[0].legend()
axs[0].grid(True)

# Ograniczony zakres błędów
axs[1].plot(x, error_taylor, label="Taylor (n=5)", color="red", linestyle="--")
axs[1].plot(x, error_fourier, label="Fourier (n=10)", color="green", linestyle=":")
axs[1].plot(x, error_wls_poly, label="$x^n$ (n=5)", color="orange", linestyle="dashdot")
axs[1].plot(x, error_wls_chebyshev, label="Chebyshev (n=5)", color="purple", linestyle="-.")
axs[1].set_title("Błąd aproksymacji – przycięty do [-1, 1]")
axs[1].set_xlabel("x")
axs[1].set_ylim(-1, 1)
axs[1].legend()
axs[1].grid(True)

plt.tight_layout()
plt.show()

# --- Wykresy wielomianów bazowych w jednej figurze z 2 subplotami ---
x_base = np.linspace(-1, 1, 400)
degree = 4

fig, axs = plt.subplots(1, 2, figsize=(16, 6), sharex=True, sharey=True)

# x^n
for i in range(degree + 1):
    axs[0].plot(x_base, x_base**i, label=f"$x^{i}$")
axs[0].set_title("Bazowe wielomiany potęgowe $x^n$")
axs[0].set_xlabel("x")
axs[0].set_ylabel("Wartość")
axs[0].grid(True)
```

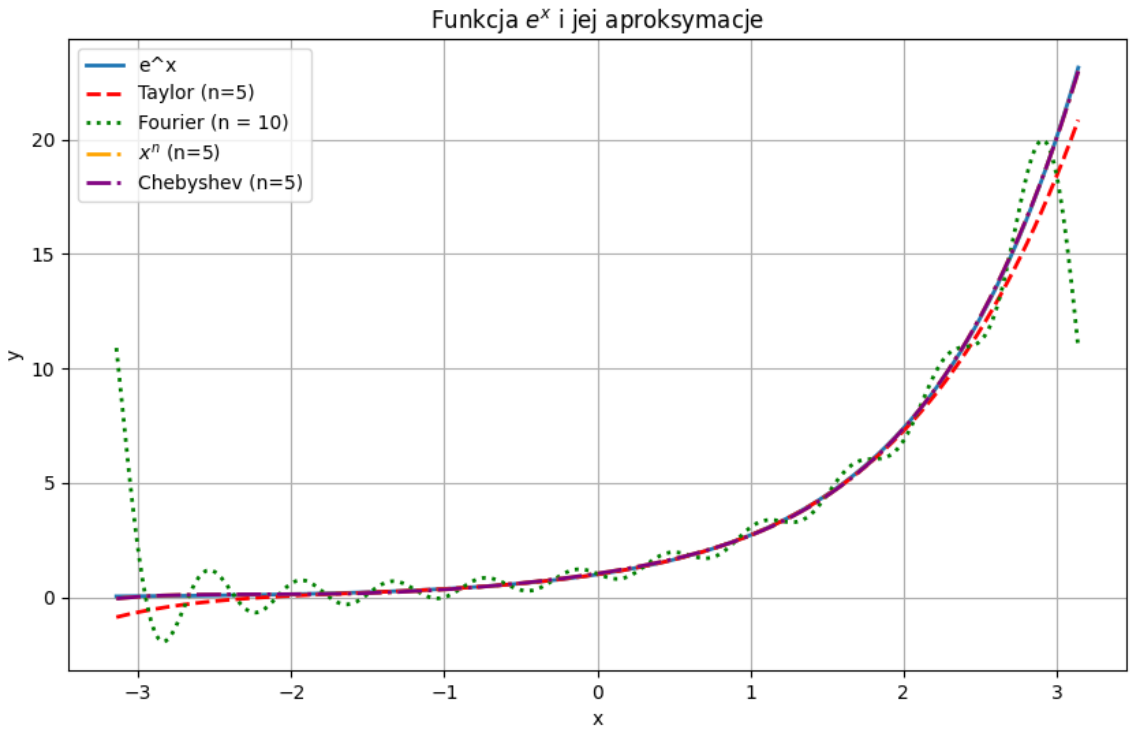
```python
# Czebyszewa
T_vals = chebyshev_polynomials(x_base, degree)
for i in range(degree + 1):
    axs[1].plot(x_base, T_vals[i], label=f"T_{i}(x)")
axs[1].set_title("Wielomiany Czebyszewa")
axs[1].set_xlabel("x")
axs[1].grid(True)

plt.tight_layout()
plt.show()

fig, axs = plt.subplots(len(weights), 2, figsize=(12, 12))

for idx, (name, weight_func) in enumerate(weights.items()):
    w_vals = weight_func(x_scaled)

    # Rysuj funkcję wagową
    axs[idx, 0].plot(x_scaled, w_vals)
    axs[idx, 0].set_title(f"Funkcja wagowa: {name}")
    axs[idx, 0].set_xlabel("x")
    axs[idx, 0].set_ylabel("W(x)")
    axs[idx, 0].grid(True)

    # WLS z daną wagą (na wielomianach potęgowych)
    y_wls_weighted = wls_with_weight(x_scaled, y_scaled, degree=5, weight_func=weight_func, basis='polynomial')
    error_weighted = np.abs(y_exact - y_wls_weighted)

    # Wykres błędu
    axs[idx, 1].plot(x, error_weighted, label="WLS z wagą", color="blue")
    axs[idx, 1].plot(x, error_taylor, label="Taylor (n=5)", color="red", linestyle="--")
    axs[idx, 1].plot(x, error_fourier, label="Fourier (n=10)", color="green", linestyle=":")
    axs[idx, 1].plot(x, error_wls_poly, label="$x^n$ (n=5)", color="orange", linestyle="dashdot")
    axs[idx, 1].plot(x, error_wls_chebyshev, label="Chebyshev (n=5)", color="purple", linestyle="-.")
    axs[idx, 1].set_title(f"Funkcja $e^x$ - błąd aproksymacji z wagą: {name}")
    axs[idx, 1].set_xlabel("x")
    axs[idx, 1].set_ylabel("Błąd")
    axs[idx, 1].set_ylim(-1, 1)
    axs[idx, 1].legend()
    axs[idx, 1].grid(True)

plt.tight_layout()
plt.show()
```

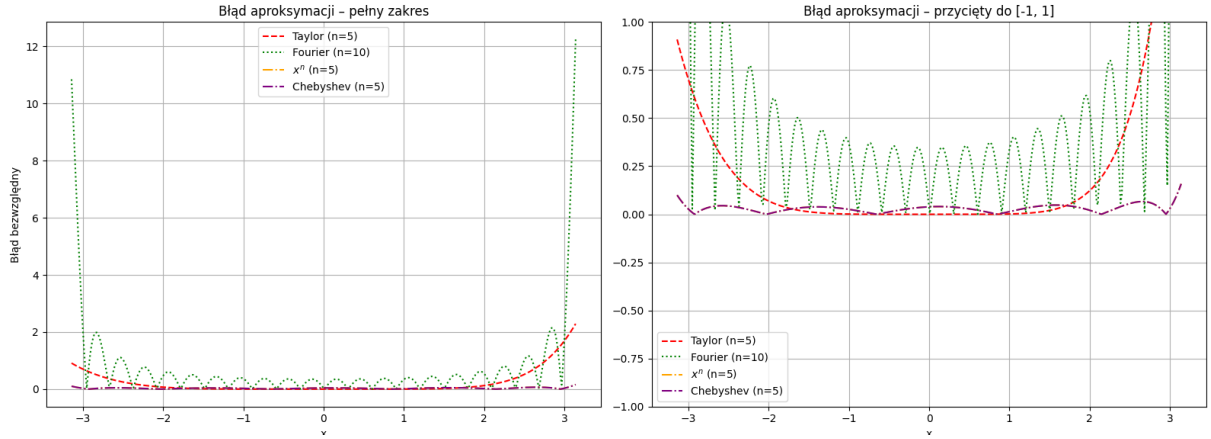
```python
#=======================================================================================================================
#=======================================================================================================================

# Funkcja |sin(x)|
def f_abs_sin(x):
    return np.abs(np.sin(x))

# Przybliżenie Taylora dla |sin(x)| (n = 7)
def taylor_approximation_abs_sin(x, n):
    approximation = np.zeros_like(x)
    for i in range(n+1):
        approximation += ((-1)**i) * (x**(2*i+1)) / math.factorial(2*i+1)
    return np.abs(approximation)

# Aproksymacja szeregiem Fouriera dla |sin(x)| (n = 10)
def fourier_series_approximation_abs_sin(func, n_terms, samples=1000, interval=(-np.pi, np.pi)):
    a, b = interval
    L = (b - a) / 2
    x_values = np.linspace(a, b, samples)

    a0 = (1/L) * quad(lambda x: func(x), a, b)[0]

    an = []
    bn = []
    for n in range(1, n_terms+1):
        an_coeff = (1/L) * quad(lambda x: func(x) * np.cos(n*np.pi*x/L), a, b)[0]
        bn_coeff = (1/L) * quad(lambda x: func(x) * np.sin(n*np.pi*x/L), a, b)[0]
        an.append(an_coeff)
        bn.append(bn_coeff)

    approximation = a0/2 * np.ones_like(x_values)
    for n in range(1, n_terms+1):
        approximation += an[n-1] * np.cos(n*np.pi*x_values/L) + bn[n-1] * np.sin(n*np.pi*x_values/L)

    return x_values, np.abs(approximation)
```

```python
# WLS z Czebyszewem dla |sin(x)| (n = 15)
def wls_approximation_chebyshev_abs_sin(x, y, degree):
    T = chebyshev_polynomials(x, degree).T
    W = np.diag(np.ones_like(x))
    A = T.T @ W @ T
    b = T.T @ W @ y
    coeffs = np.linalg.solve(A, b)
    y_approx = T @ coeffs
    return y_approx

# WLS z klasycznymi wielomianami dla |sin(x)| (n = 15)
def wls_approximation_polynomial_abs_sin(x, y, degree):
    X = np.vander(x, degree + 1, increasing=True)
    W = np.diag(np.ones_like(x))
    A = X.T @ W @ X
    b = X.T @ W @ y
    coeffs = np.linalg.solve(A, b)
    y_approx = X @ coeffs
    return y_approx

# Funkcje wagowe i wls
def wls_with_weight_abs_sin(x, y, degree, weight_func, basis='polynomial'):
    W = np.diag(weight_func(x))

    if basis == 'polynomial':
        X = np.vander(x, degree + 1, increasing=True)
    elif basis == 'chebyshev':
        X = chebyshev_polynomials(x, degree).T

    A = X.T @ W @ X
    b = X.T @ W @ y
    coeffs = np.linalg.solve(A, b)
    y_approx = X @ coeffs
    return y_approx

# Obliczenie funkcji i przybliżeń dla |sin(x)|
y_exact_abs_sin = f_abs_sin(x)
y_taylor_abs_sin = taylor_approximation_abs_sin(x, 7)

# Fourier dla |sin(x)|
x_fourier_abs_sin, y_fourier_abs_sin = fourier_series_approximation_abs_sin(f_abs_sin, 10, samples, (a, b))
```

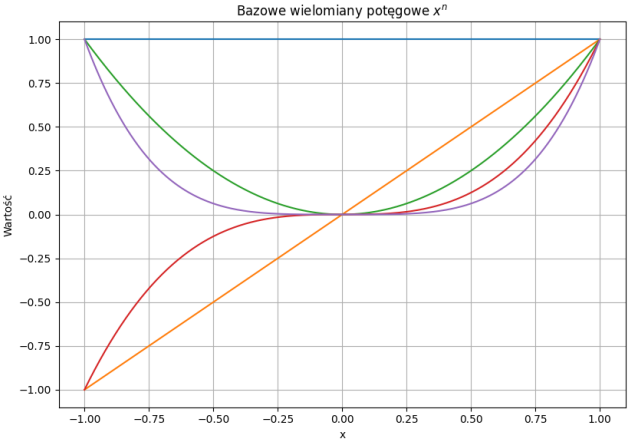
```python
# WLS - przeskalowanie x do [-1, 1]
x_scaled_abs_sin = x / np.pi
y_scaled_abs_sin = f_abs_sin(x)

# WLS: Chebyshev i klasyczny wielomian dla |sin(x)| (n = 15)
y_wls_chebyshev_abs_sin = wls_approximation_chebyshev_abs_sin(x_scaled_abs_sin, y_scaled_abs_sin, degree=15)
y_wls_poly_abs_sin = wls_approximation_polynomial_abs_sin(x_scaled_abs_sin, y_scaled_abs_sin, degree=15)

# --- Wykres funkcji i aproksymacji dla |sin(x)| ---
plt.figure(figsize=(10, 6))
plt.plot(x, y_exact_abs_sin, label='|sin(x)|', linewidth=2)
plt.plot(x, y_taylor_abs_sin, label="Taylor (n=7)", color="red", linestyle="--", linewidth=2)
plt.plot(x_fourier_abs_sin, y_fourier_abs_sin, label=f"Fourier (n = 10)", color="green", linestyle=":", linewidth=2)
plt.plot(x, y_wls_poly_abs_sin, label="$x^n$ (n=15)", color="orange", linestyle="dashdot", linewidth=2)
plt.plot(x, y_wls_chebyshev_abs_sin, label="Chebyshev (n=15)", color="purple", linestyle="-.", linewidth=2)
plt.title("Funkcja $|sin(x)|$ i jej aproksymacje")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()

# Funkcje wagowe
weights = {
    "Stała": constant_weight,
    "Trójkątna": triangle_weight,
    "Trójkątna odwrotna": v_shape_weight
}
```

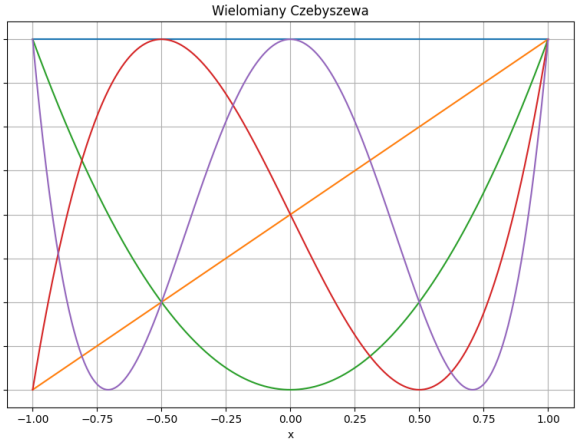
```python
# --- Wykresy funkcji wagowych i aproksymacji WLS dla |sin(x)| ---
fig, axs = plt.subplots(len(weights), 2, figsize=(12, 12))

for idx, (name, weight_func) in enumerate(weights.items()):
    w_vals = weight_func(x_scaled_abs_sin)

    # Rysuj funkcję wagową
    axs[idx, 0].plot(x_scaled_abs_sin, w_vals)
    axs[idx, 0].set_title(f"Funkcja wagowa: {name}")
    axs[idx, 0].set_xlabel("x")
    axs[idx, 0].set_ylabel("W(x)")
    axs[idx, 0].grid(True)

    # WLS z daną wagą (na wielomianach potęgowych)
    y_wls_weighted_abs_sin = wls_with_weight_abs_sin(x_scaled_abs_sin, y_scaled_abs_sin, degree=15, weight_func=weight_func, basis='polynomial')

    # Rysuj WLS z wagą
    axs[idx, 1].plot(x, np.abs(y_exact_abs_sin - y_wls_weighted_abs_sin), label="WLS z wagą", color="blue")
    axs[idx, 1].plot(x, np.abs(y_exact_abs_sin - y_taylor_abs_sin), label="Taylor (n=7)", color="red", linestyle="--")
    axs[idx, 1].plot(x, np.abs(y_exact_abs_sin - np.interp(x, x_fourier_abs_sin, y_fourier_abs_sin)), label="Fourier (n=10)", color="green", linestyle=":")
    axs[idx, 1].plot(x, np.abs(y_exact_abs_sin - y_wls_poly_abs_sin), label="$x^n$ (n=15)", color="orange", linestyle="dashdot")
    axs[idx, 1].plot(x, np.abs(y_exact_abs_sin - y_wls_chebyshev_abs_sin), label="Chebyshev (n=15)", color="purple", linestyle="-.")

    # Ustawienie zakresu osi y
    axs[idx, 1].set_title(f"Funkcja $|sin(x)|$ - błąd aproksymacji z wagą: {name}")
    axs[idx, 1].set_xlabel("x")
    axs[idx, 1].set_ylabel("Błąd")
    axs[idx, 1].set_ylim(-0.1, 0.2)  # Ustawienie zakresu y
    axs[idx, 1].legend()
    axs[idx, 1].grid(True)

plt.tight_layout()
plt.show()
```

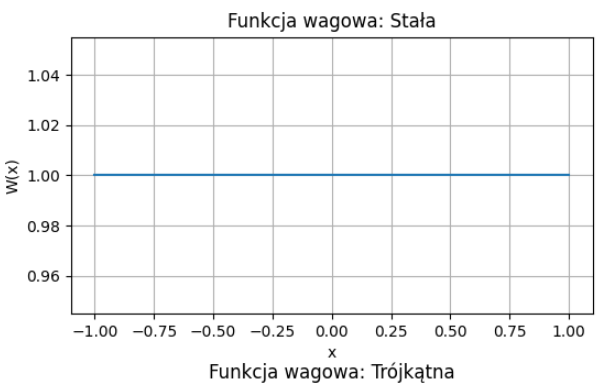


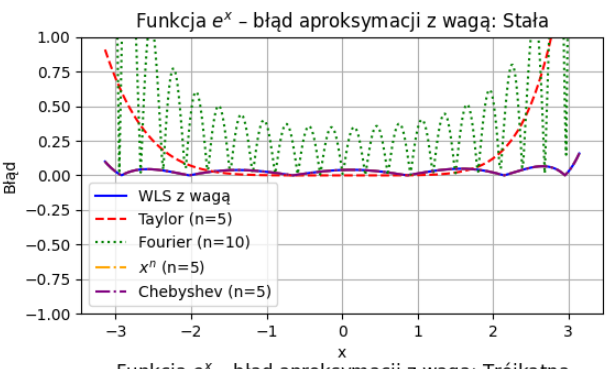

Błąd aproksymacji – pełny zakres
Błąd bezwzględny
x
Taylor (n=5)
Fourier (n=10)
$x^n$ (n=5)
Chebyshev (n=5)

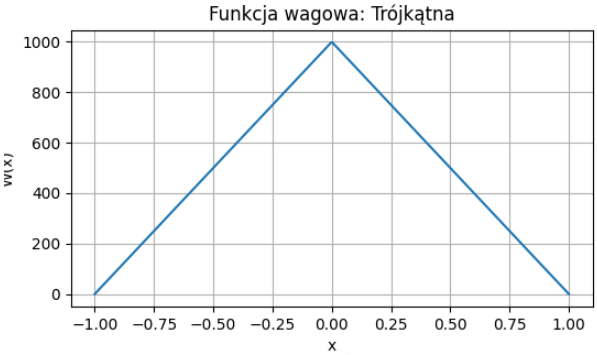

Błąd aproksymacji – przycięty do [-1, 1]
x
Taylor (n=5)
Fourier (n=10)
$x^n$ (n=5)
Chebyshev (n=5)

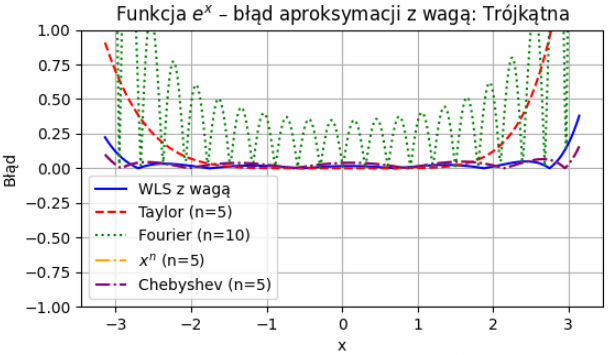

Bazowe wielomiany potęgowe $x^n$
Wartość
x

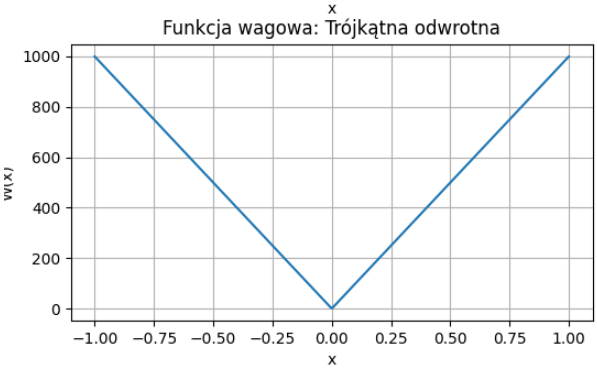

Wielomiany Czebyszewa
x

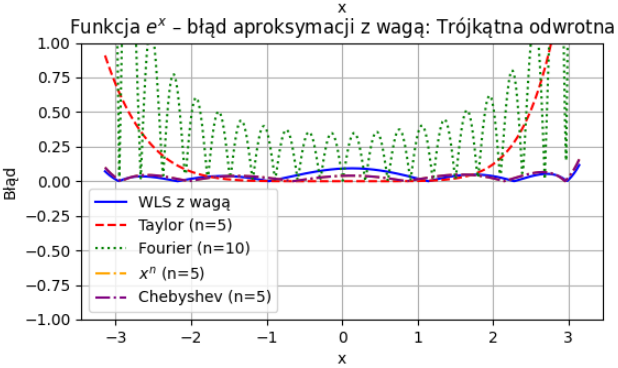

Funkcja wagowa: Stała
W(x)
x

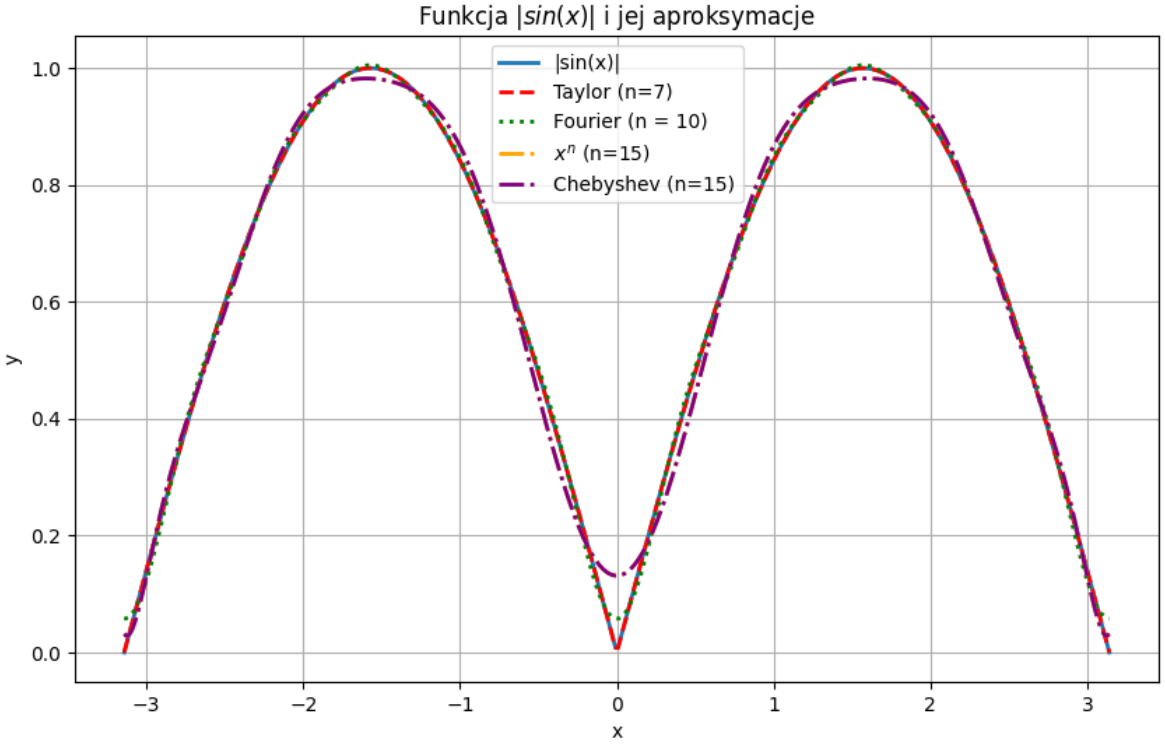

Funkcja $e^x$ – błąd aproksymacji z wagą: Stała
Błąd
x
WLS z wagą
Taylor (n=5)
Fourier (n=10)
$x^n$ (n=5)
Chebyshev (n=5)

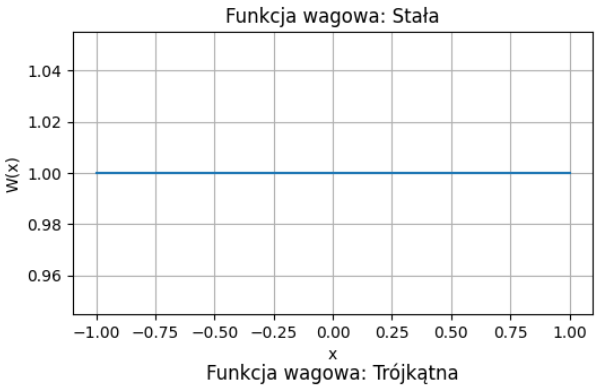

Funkcja wagowa: Trójkątna
x

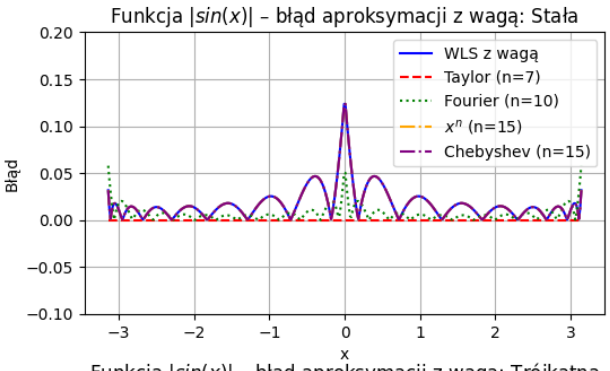

Funkcja $e^x$ – błąd aproksymacji z wagą: Trójkątna
Błąd
x
WLS z wagą
Taylor (n=5)
Fourier (n=10)
$x^n$ (n=5)
Chebyshev (n=5)

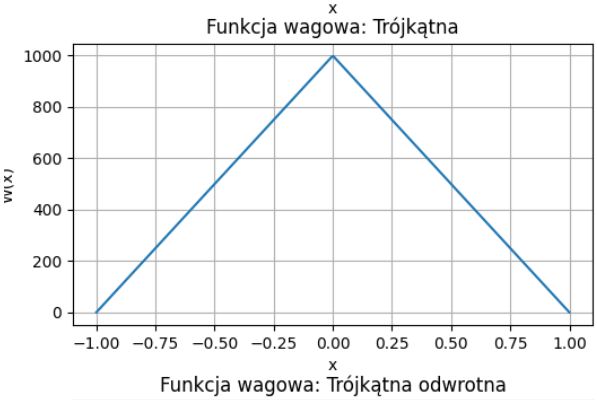

Funkcja wagowa: Trójkątna odwrotna
x

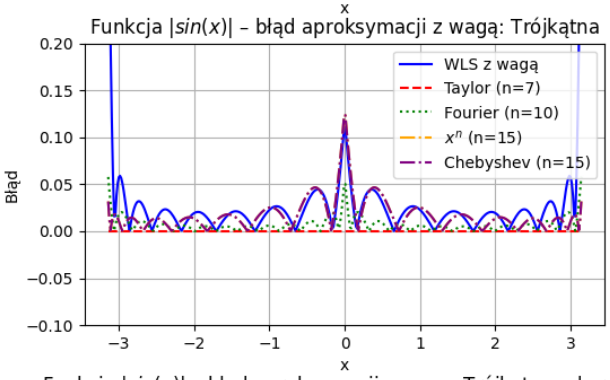

Funkcja $e^x$ – błąd aproksymacji z wagą: Trójkątna odwrotna
Błąd
x
WLS z wagą
Taylor (n=5)
Fourier (n=10)
$x^n$ (n=5)
Chebyshev (n=5)

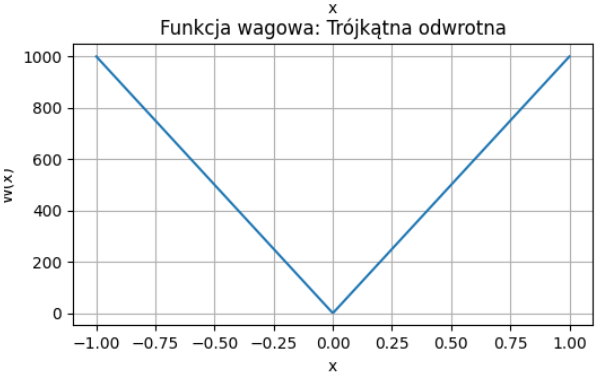

Funkcja |sin(x)| i jej aproksymacje
|sin(x)|
Taylor (n=7)
Fourier (n = 10)
$x^n$ (n=15)
Chebyshev (n=15)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3

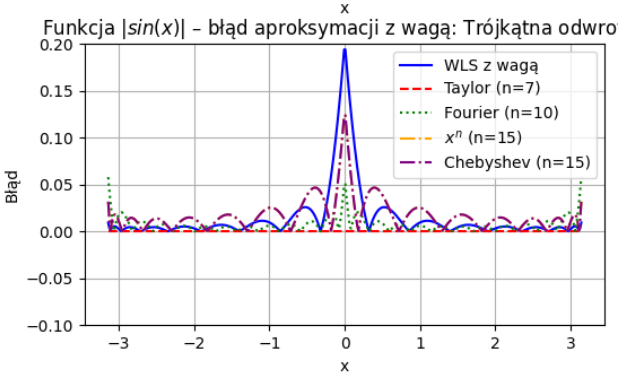

Funkcja wagowa: Stała
W(x)
x


Funkcja |sin(x)| – błąd aproksymacji z wagą: Stała
Błąd
x
WLS z wagą
Taylor (n=7)
Fourier (n=10)
$x^n$ (n=15)
Chebyshev (n=15)


Funkcja wagowa: Trójkątna
x


Funkcja |sin(x)| – błąd aproksymacji z wagą: Trójkątna
Błąd
x
WLS z wagą
Taylor (n=7)
Fourier (n=10)
$x^n$ (n=15)
Chebyshev (n=15)


Funkcja wagowa: Trójkątna odwrotna
x

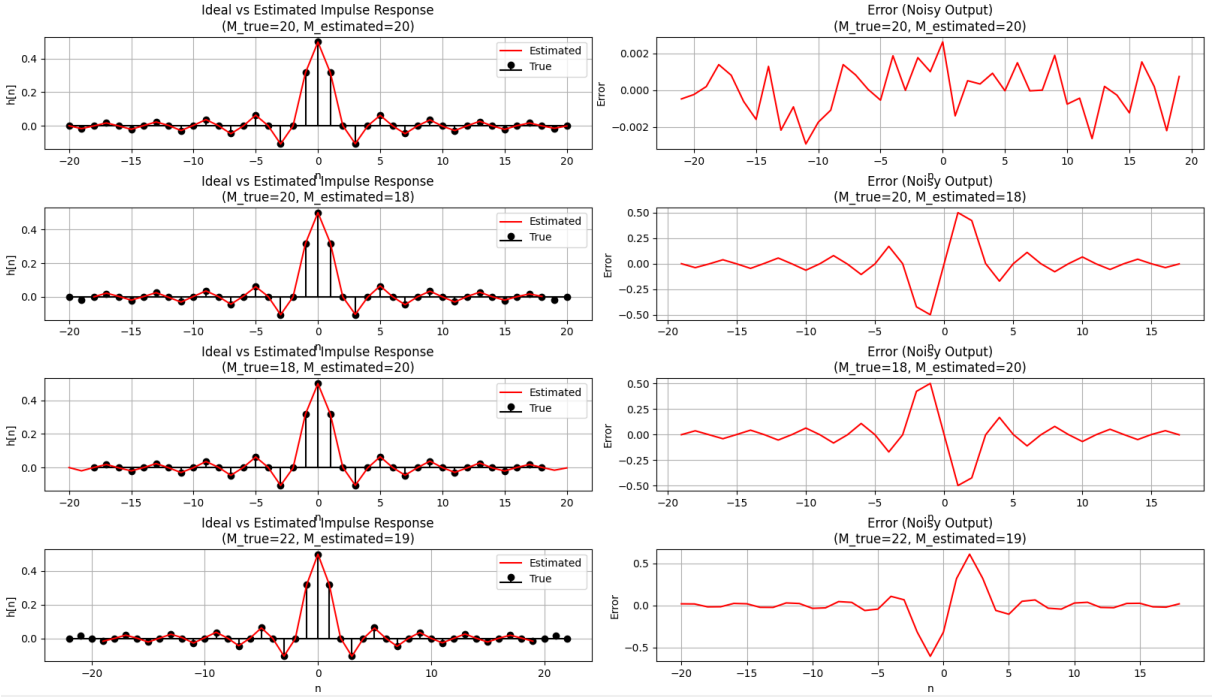

Funkcja |sin(x)| – błąd aproksymacji z wagą: Trójkątna odwro
Błąd
x
WLS z wagą
Taylor (n=7)
Fourier (n=10)
$x^n$ (n=15)
Chebyshev (n=15)

Zadanie 2:

```python
import numpy as np
import matplotlib.pyplot as plt

def h_lp(n, wc=np.pi/2):
    if n == 0:
        return wc / np.pi
    else:
        return np.sin(wc * n) / (np.pi * n)

def generate_impulse_response(M, wc=np.pi/2):
    return np.array([h_lp(n, wc) for n in range(-M, M + 1)])

def generate_gaussian_noise(length, mean=0, std=1):
    return np.random.normal(mean, std, length)

def identify_impulse_response(f, d_noisy, M):
    N = len(f)
    L = 2 * M + 1
    X = np.zeros((N, L))
    for i in range(N):
        for j in range(L):
            if 0 <= i - M + j < N:
                X[i, j] = f[i - M + j]
    h_estimated = np.linalg.pinv(X) @ d_noisy
    return h_estimated

def main():
    configurations = [(20, 20), (20, 18), (18, 20), (22, 19)]
    N_true = 5000
    N_estimated = 5000
    wc = np.pi / 2

    plt.figure(figsize=(18, 12))
```

```
plt.figure(figsize=(18, 12))

for idx, (M_true, M_estimated) in enumerate(configurations):
    h_true = generate_impulse_response(M_true, wc)

    f_true = generate_gaussian_noise(N_true)

    d_true = np.convolve(f_true, h_true, mode='same')

    h_estimated_true = generate_impulse_response(M_estimated, wc)

    f_estimated = generate_gaussian_noise(N_estimated)

    noise = generate_gaussian_noise(N_estimated, mean=0, std=0.1)
    d_noisy = np.convolve(f_estimated, h_estimated_true, mode='same') + noise

    h_estimated = identify_impulse_response(f_estimated, d_noisy, M_estimated)

    n_true = np.arange(-M_true, M_true + 1)
    n_estimated = np.arange(-M_estimated, M_estimated + 1)

    plt.subplot(4, 2, 2 * idx + 1)
    plt.title(f"Ideal vs Estimated Impulse Response\n(M_true={M_true}, M_estimated={M_estimated})")
    plt.stem(n_true, h_true, linefmt='k-', markerfmt='ko', basefmt='k', label="True")
    plt.plot(n_estimated, h_estimated, 'r-', label="Estimated")
    plt.xlabel("n")
    plt.ylabel("h[n]")
    plt.legend()
    plt.grid()

    plt.subplot(4, 2, 2 * idx + 2)
    plt.title(f"Error (Noisy Output)\n(M_true={M_true}, M_estimated={M_estimated})")

    min_len = min(len(h_true), len(h_estimated))
    error = h_true[:min_len] - h_estimated[:min_len]
    n_error = np.arange(-min_len // 2, -min_len // 2 + min_len)

    plt.plot(n_error, error, 'r-')
    plt.xlabel("n")
    plt.ylabel("Error")
    plt.grid()
```

```
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```


Ideal vs Estimated Impulse Response
(M_true=20, M_estimated=20)
Error (Noisy Output)
(M_true=20, M_estimated=20)
Ideal vs Estimated Impulse Response
(M_true=20, M_estimated=18)
Error (Noisy Output)
(M_true=20, M_estimated=18)
Ideal vs Estimated Impulse Response
(M_true=18, M_estimated=20)
Error (Noisy Output)
(M_true=18, M_estimated=20)
Ideal vs Estimated Impulse Response
(M_true=22, M_estimated=19)
Error (Noisy Output)
(M_true=22, M_estimated=19)
Estimated
True
h[n]
Error
n

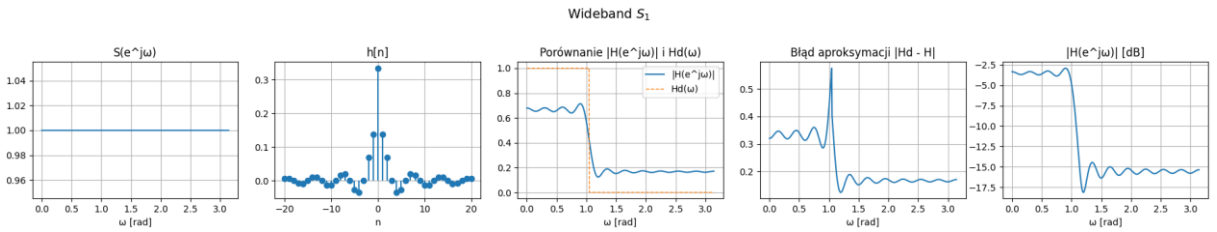
Zadanie 3:

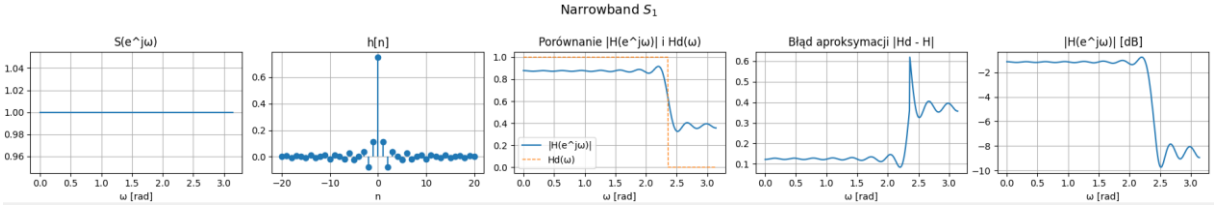
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import freqz

# Parametry filtru
M = 20
N_freqs = 512
w = np.linspace(0, np.pi, N_freqs)

# Macierz A
A = np.zeros((N_freqs, M+1))
for i in range(M+1):
    A[:, i] = 2 * np.cos(w * i)
A[:, 0] = 1

def ideal_response(w, typ='wide'):
    if typ == 'wide':
        wc = np.pi / 3  # Szerokie pasmo przejściowe
    elif typ == 'narrow':
        wc = 3 * np.pi / 4  # Wąskie pasmo przejściowe
    else:
        raise ValueError("Unknown type")
    return np.where(w <= wc, 1, 0), wc

def weight_function(w, typ='S1'):
    if typ == 'S1':
        return np.ones_like(w)
    elif typ == 'S2':
        return np.where(w < 1.5, 20.0, 0.0)
    elif typ == 'S3':
        return np.where(w < 1.5, 0.0, 20.0)
    else:
        raise ValueError("Unknown weighting function")

def plot_filter_response(w, Hd, A, S, title):
    W = np.diag(S)
    c = np.linalg.inv(A.T @ W @ A) @ A.T @ W @ Hd

    # Odpowiedź impulsowa h[n]
    h = np.zeros(2*M+1)
    h[M] = c[0]
    for n in range(1, M+1):
        h[M+n] = h[M-n] = c[n]/2
```

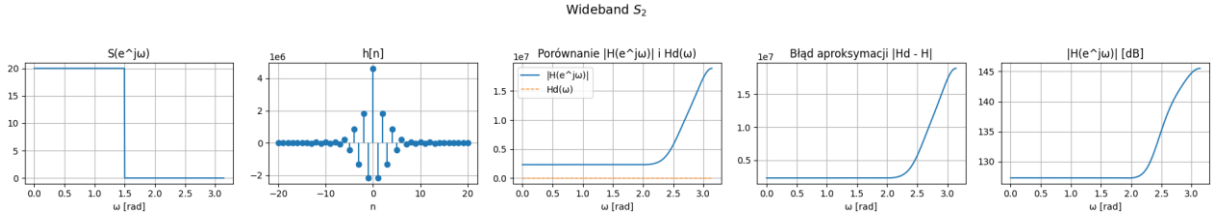
```python
# Odpowiedź częstotliwościowa
w_out, H_out = freqz(h, worN=w)
H_mag = np.abs(H_out)
error = np.abs(Hd - H_mag)

# Rysowanie wykresów
fig, axs = plt.subplots(1, 5, figsize=(22, 4))
fig.suptitle(f"{title}", fontsize=14)

axs[0].plot(w, S)
axs[0].set_title("S(e^jω)")
axs[0].set_xlabel("ω [rad]")
axs[0].grid()

axs[1].stem(np.arange(-M, M+1), h, basefmt=" ")
axs[1].set_title("h[n]")
axs[1].set_xlabel("n")
axs[1].grid()

axs[2].plot(w_out, H_mag, label='|H(e^jω)|')
axs[2].plot(w, Hd, '--', label='Hd(ω)', linewidth=1)
axs[2].set_title("Porównanie |H(e^jω)| i Hd(ω)")
axs[2].set_xlabel("ω [rad]")
axs[2].legend()
axs[2].grid()

axs[3].plot(w, error)
axs[3].set_title("Błąd aproksymacji |Hd - H|")
axs[3].set_xlabel("ω [rad]")
axs[3].grid()

axs[4].plot(w_out, 20 * np.log10(H_mag + 1e-10))
axs[4].set_title("|H(e^jω)| [dB]")
axs[4].set_xlabel("ω [rad]")
axs[4].grid()

plt.tight_layout(rect=[0, 0, 1, 0.93])
plt.show()
```

```python
# === S₁: flat weight ===
S1 = weight_function(w, 'S1')
Hd_wide, _ = ideal_response(w, 'wide')
Hd_narrow, _ = ideal_response(w, 'narrow')
plot_filter_response(w, Hd_wide, A, S1, "Wideband $S_1$")
plot_filter_response(w, Hd_narrow, A, S1, "Narrowband $S_1$")

# === S₂: większa waga w dolnym paśmie ===
S2 = weight_function(w, 'S2')
plot_filter_response(w, Hd_wide, A, S2, "Wideband $S_2$")
plot_filter_response(w, Hd_narrow, A, S2, "Narrowband $S_2$")

# === S₃: większa waga w górnym paśmie ===
S3 = weight_function(w, 'S3')
plot_filter_response(w, Hd_wide, A, S3, "Wideband $S_3$")
plot_filter_response(w, Hd_narrow, A, S3, "Narrowband $S_3$")

def plot_combined_responses(w, Hd, A, weights, labels, title):
    fig, axs = plt.subplots(1, 2, figsize=(16, 5))
    fig.suptitle(title, fontsize=14)

    for S, label in zip(weights, labels):
        W = np.diag(S)
        c = np.linalg.inv(A.T @ W @ A) @ A.T @ W @ Hd

        # Tworzenie odpowiedzi impulsowej
        h = np.zeros(2*M+1)
        h[M] = c[0]
        for n in range(1, M+1):
            h[M+n] = h[M-n] = c[n]/2

        # Odpowiedź częstotliwościowa
        w_out, H_out = freqz(h, worN=w)
        H_mag = np.abs(H_out)
        error = np.abs(Hd - H_mag)

        axs[0].plot(w_out, H_mag, label=label)
        axs[1].plot(w, error, label=label)
```

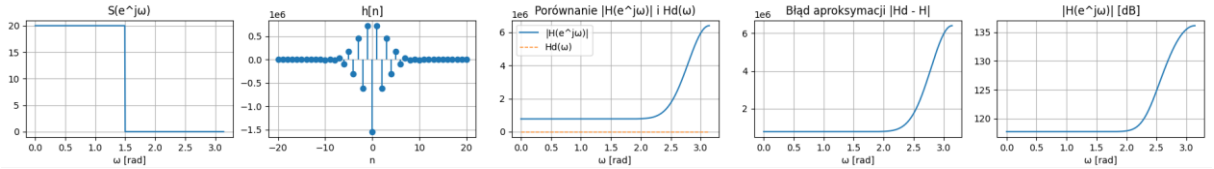
```
    axs[0].plot(w, Hd, 'k--', linewidth=1, label='Hd(ω)')
    axs[0].set_title("Porównanie |H(e^jω)|")
    axs[0].set_xlabel("ω [rad]")
    axs[0].legend()
    axs[0].grid()

    axs[1].set_title("Błąd aproksymacji |Hd - H|")
    axs[1].set_xlabel("ω [rad]")
    axs[1].legend()
    axs[1].grid()

    plt.tight_layout(rect=[0, 0, 1, 0.93])
    plt.show()

# Przygotowanie wag
S1 = weight_function(w, 'S1')
S2 = weight_function(w, 'S2')
S3 = weight_function(w, 'S3')

# Zbiorcze porównanie: wideband
plot_combined_responses(
    w, Hd_wide, A,
    weights=[S1, S2, S3],
    labels=["$S_1$", "$S_2$", "$S_3$"],
    title="Zbiorcze porównanie filtrów – Wideband"
)

# Zbiorcze porównanie: narrowband
plot_combined_responses(
    w, Hd_narrow, A,
    weights=[S1, S2, S3],
    labels=["$S_1$", "$S_2$", "$S_3$"],
    title="Zbiorcze porównanie filtrów – Narrowband"
)
```



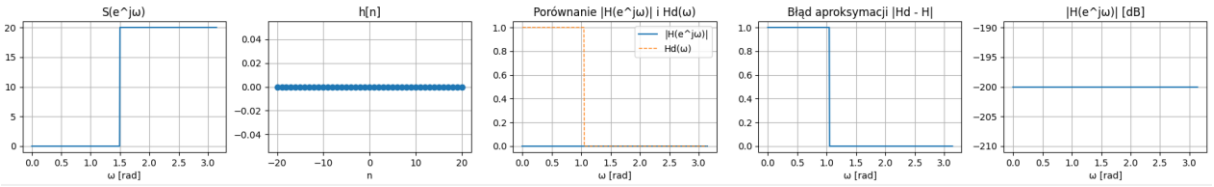



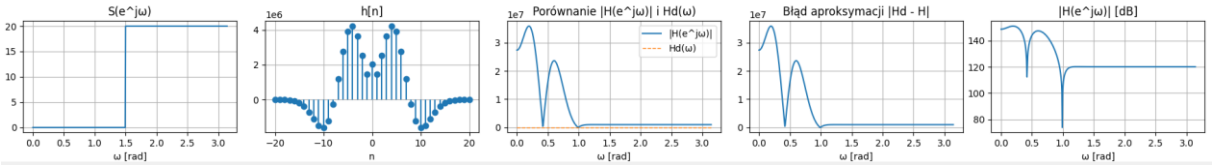
Narrowband $S_2$



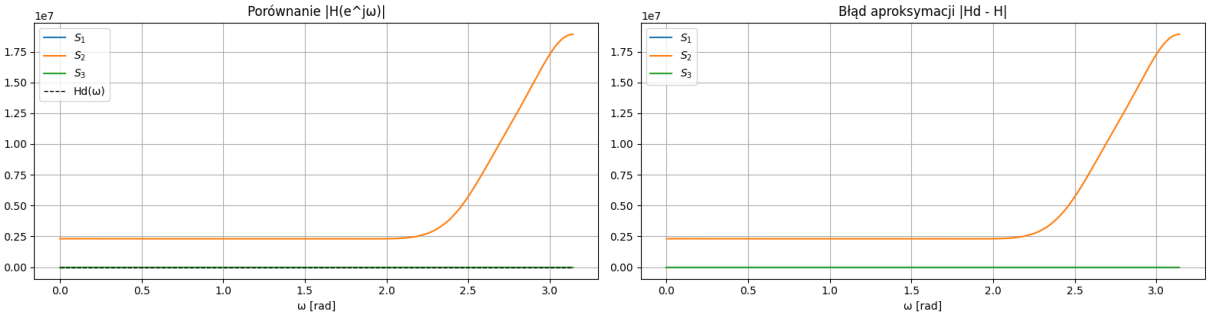

Wideband $S_3$



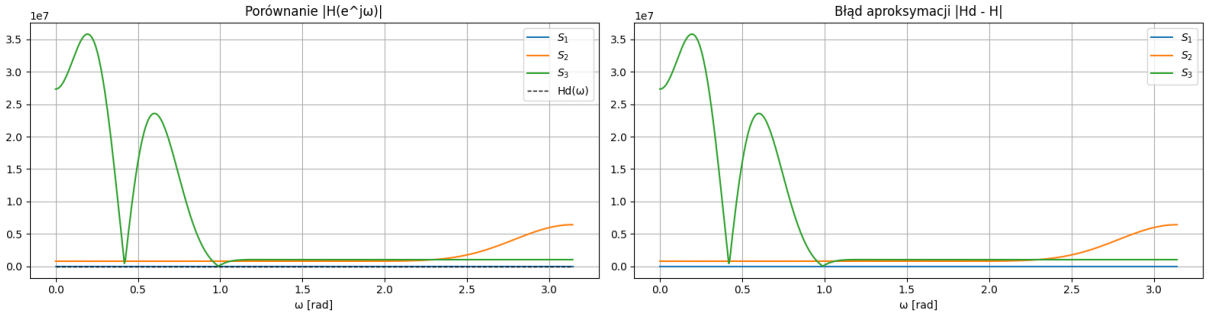

Narrowband $S_3$




Zbiorcze porównanie filtrów – Wideband




Zbiorcze porównanie filtrów – Narrowband

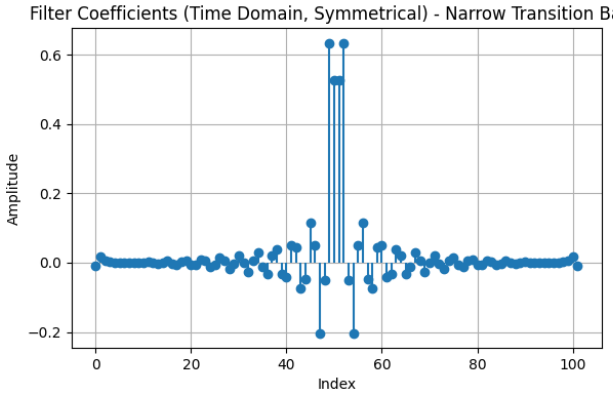


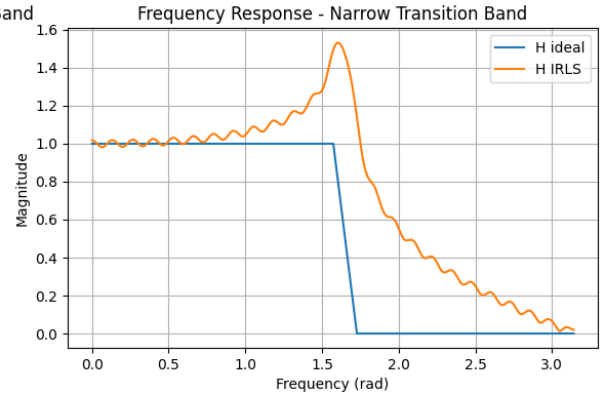
Zadanie 4:

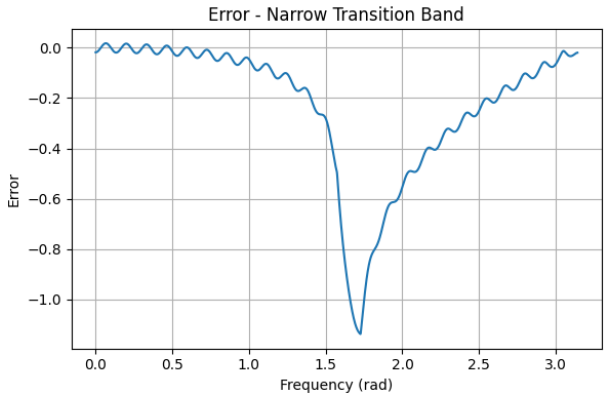
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import toeplitz

def irwls(A, x, p, maxiter=200, stopeps=1e-7):
    pk = 2  # Starting value of p
    c = np.linalg.lstsq(A, x, rcond=None)[0]  # Initial LS solution
    xhat = A @ c
    gamma = 1.5

    for k in range(maxiter):
        pk = min(p, gamma * pk)  # Update p for this iteration
        e = x - xhat  # Estimation error
        s = np.abs(e) ** ((pk - 2) / 2)  # New weights
        WA = np.diag(s) @ A  # Weighted matrix
        chat = np.linalg.lstsq(WA, s * x, rcond=None)[0]  # Weighted LS solution
        lambda_ = 1 / (pk - 1)
        cnew = lambda_ * chat + (1 - lambda_) * c

        if np.linalg.norm(c - cnew) < stopeps:
            c = cnew
            break

        c = cnew
        xhat = A @ c

    return c

def design_fir_filter(N, wc, p, transition_width):
    M = 512
    omega = np.linspace(0, np.pi, M)

    Hd = np.zeros(M)
    Hd[omega <= wc] = 1

    transition_start = wc
    transition_end = min(np.pi, wc + transition_width)
    transition_indices = (omega >= transition_start) & (omega <= transition_end)
    Hd[transition_indices] = np.linspace(1, 0, np.sum(transition_indices))

    k = np.arange(N)
    A = np.cos(np.outer(omega, k))

    h = irwls(A, Hd, p)

    return h, Hd, omega
```

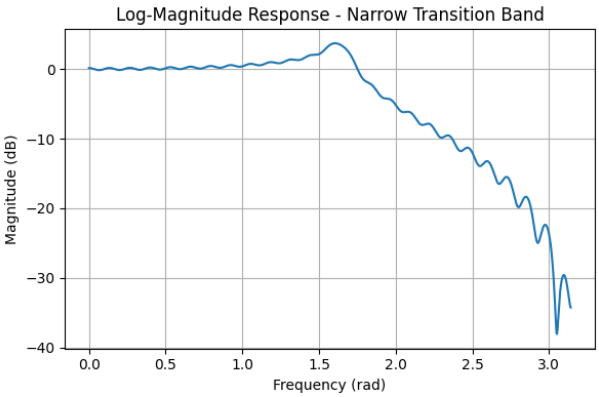
```python
def plot_results(h, Hd, omega, wc, transition_width, title_suffix):
    plt.figure(figsize=(12, 8))
    plt.subplot(2, 2, 1)
    h_symmetric = np.concatenate((h[::-1], h))
    plt.stem(h_symmetric, basefmt=" ", markerfmt="o")
    plt.title(f"Filter Coefficients (Time Domain, Symmetrical) - {title_suffix}")
    plt.xlabel("Index")
    plt.ylabel("Amplitude")
    plt.grid()

    fft_size = 4096
    H = np.fft.fft(h, fft_size)
    H = np.abs(H[:fft_size // 2])
    omega_fine = np.linspace(0, np.pi, fft_size // 2)
    plt.subplot(2, 2, 2)
    plt.plot(omega, Hd, label="H ideal")
    plt.plot(omega_fine, H, label="H IRLS")
    plt.title(f"Frequency Response - {title_suffix}")
    plt.xlabel("Frequency (rad)")
    plt.ylabel("Magnitude")
    plt.legend()
    plt.grid()

    plt.subplot(2, 2, 3)
    Hd_interp = np.interp(omega_fine, omega, Hd)
    plt.plot(omega_fine, Hd_interp - H)
    plt.title(f"Error - {title_suffix}")
    plt.xlabel("Frequency (rad)")
    plt.ylabel("Error")
    plt.grid()

    plt.subplot(2, 2, 4)
    plt.plot(omega_fine, 20 * np.log10(np.maximum(H, 1e-10)))
    plt.title(f"Log-Magnitude Response - {title_suffix}")
    plt.xlabel("Frequency (rad)")
    plt.ylabel("Magnitude (dB)")
    plt.grid()

    plt.tight_layout()
    plt.show()
```

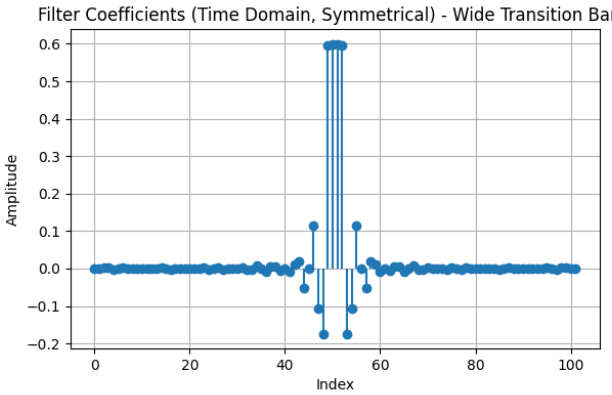
```python
N = 51
wc = np.pi / 2
p = 30

transition_width_narrow = 0.05 * np.pi
h_narrow, Hd_narrow, omega = design_fir_filter(N, wc, p, transition_width_narrow)
plot_results(h_narrow, Hd_narrow, omega, wc, transition_width_narrow, "Narrow Transition Band")

transition_width_wide = 0.2 * np.pi
h_wide, Hd_wide, omega = design_fir_filter(N, wc, p, transition_width_wide)
plot_results(h_wide, Hd_wide, omega, wc, transition_width_wide, "Wide Transition Band")
```

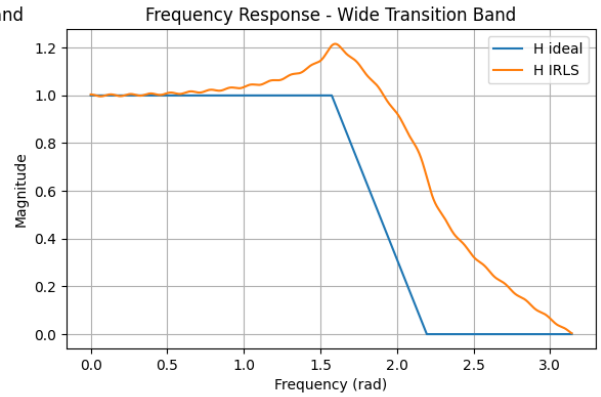

Filter Coefficients (Time Domain, Symmetrical) - Narrow Transition Band
Amplitude
Index

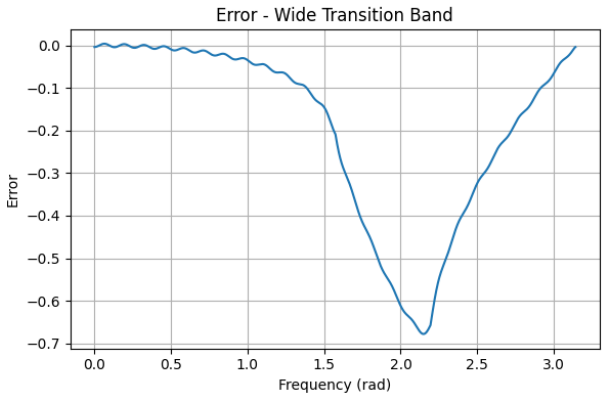

Frequency Response - Narrow Transition Band
H ideal
H IRLS
Magnitude
Frequency (rad)

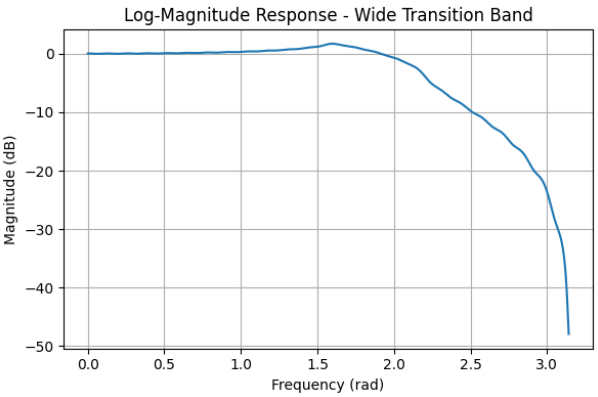

Error - Narrow Transition Band
Error
Frequency (rad)


Log-Magnitude Response - Narrow Transition Band
Magnitude (dB)
Frequency (rad)


Filter Coefficients (Time Domain, Symmetrical) - Wide Transition Band
Amplitude
Index


Frequency Response - Wide Transition Band
H ideal
H IRLS
Magnitude
Frequency (rad)


Error - Wide Transition Band
Error
Frequency (rad)


Log-Magnitude Response - Wide Transition Band
Magnitude (dB)
Frequency (rad)