OMNIS 2

Otwartość. Modernizacja. Nowoczesność. Integracja. Społeczność.

Podstawy Elektroniki
Ćwiczenia komputerowe

Pomiary w dziedzinie częstotliwości. Szereg Fouriera. Twierdzenie o próbkowaniu

Autor: Dariusz Tefelski

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
Politechnika Warszawska

Plac Politechniki 1
00-661 Warszawa

www.pw.edu.pl

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
Politechnika Warszawska

Plac Politechniki 1
00-661 Warszawa
www.pw.edu.pl

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z koncepcją analizy sygnałów okresowych w dziedzinie częstotliwości za pomocą szeregów Fouriera. Poznanie sposobów wyznaczenia współczynników rozkładu sygnału, przekształcenia ich do formy amplitudowo-fazowej, stosowania szybkiej transformaty Fouriera FFT oraz badania procesu próbkowania i idealnej rekonstrukcji sygnałów za pomocą środowiska Jupyter Lab i bibliotek języka Python.

2 Wstęp teoretyczny

2.1 Czym jest szereg Fouriera?

Wyobraźmy sobie, że każdy dźwięk instrumentu lub sygnał elektryczny, który jest powtarzalny (okresowy), możemy "rozebrać na części pierwsze". Te części to najprostsze fale: sinusy i cosinusy o różnych częstotliwościach i amplitudach. Jean-Baptiste Joseph Fourier udowodnił, że każdą funkcję okresową $f(t)$ spełniającą warunki Dirichleta można zapisać jako sumę nieskończonego szeregu funkcji trygonometrycznych.

Dla sygnału o okresie T i częstotliwości podstawowej $f_0 = 1/T$ (częstość kołowa $\omega_0 = 2\pi/T$), szereg Fouriera przyjmuje postać:

$$f(t) = a_0 + \sum_{k=1}^{\infty} [a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t)] \tag{1}$$

Gdzie:

- $a_0 = \frac{1}{T}\int_0^T f(t)dt$ – składowa stała (średnia wartość sygnału).
- $a_k = \frac{2}{T}\int_0^T f(t)\cos(k\omega_0 t)dt$ – amplitudy składowych cosinusoidalnych.
- $b_k = \frac{2}{T}\int_0^T f(t)\sin(k\omega_0 t)dt$ – amplitudy składowych sinusoidalnych.

2.2 Postać amplitudowo-fazowa

Choć zapis z a_k i b_k jest matematycznie wygodny, w technice częściej operujemy amplitudą i fazą danej harmonicznej. Możemy to zapisać jako:

$$f(t) = c_0 + \sum_{k=1}^{\infty} c_k \cos(k\omega_0 t - \varphi_k) \tag{2}$$

Zależności między współczynnikami:

- $c_k = \sqrt{a_k^2 + b_k^2}$ – amplituda k-tej harmonicznej.
- $\varphi_k = \text{atan2}(b_k, a_k)$ – przesunięcie fazowe k-tej harmonicznej.
- $c_0 = a_0$.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
Politechnika Warszawska

Plac Politechniki 1
00-661 Warszawa
www.pw.edu.pl

Zauważ, że postać ta jest bardziej intuicyjna: mówi nam "jak silny"jest dany sygnał o częstotliwości $k \cdot f_0$ i o ile jest "przesunięty"w czasie.

Warto wiedzieć

Zauważ, że do wyznaczenia przesunięcia fazowego wykorzystano funkcję $atan2(b_k, a_k)$, a nie standardowy $atan(\frac{b_k}{a_k})$. Przeciwdziedzina funkcji $atan$ to przedział $(-\frac{\pi}{2}, \frac{\pi}{2})$, nie obejmuje pełnego kąta $2 \cdot \pi$. Aby jednoznacznie określić kąt, trzeba rozpatrzyć znaki przy współczynnikach a_k i b_k, które wskażą w której ćwiartce układu współrzędnych na płaszczyźnie, np. zespolonej znajduje się punkt, a nie tylko ich iloraz.

2.3 Próbkowanie i twierdzenie Shannona-Kotielnikowa

W systemach cyfrowych sygnał ciągły zamieniamy na ciąg próbek w odstępach T_s.

Definicja

Aby sygnał był w pełni odtwarzalny, częstotliwość próbkowania f_s musi być ponad dwukrotnie większa od najwyższej częstotliwości sygnału f_{max}. $(f_s > 2f_{max})$.

Złamanie tej zasady powoduje **aliasing** – błędne nakładanie się widm.

2.4 Transformata Fouriera i FFT

Dla sygnałów nieokresowych stosujemy Transformatę Fouriera. W komputerach używamy **Dyskretnej Transformaty Fouriera (DFT)**, a jej szybki wariant **FFT** (Fast Fourier Transform) pozwala na błyskawiczne obliczenia dzięki metodzie "dziel i zwyciężaj", redukując złożoność z $O(N^2)$ do $O(N \log N)$.

2.5 Idealna rekonstrukcja - funkcja sinc

Sygnał spróbkowany można idealnie wygładzić do postaci ciągłej sumując funkcje $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$. Każda próbka staje się "wagą"dla funkcji sinc, a ich suma wypełnia przerwy między punktami, tworząc oryginalny przebieg.

2.6 Czym jest aliasing?

Definicja

Aliasing (z ang. alias) to zjawisko „podszywania się" wysokich częstotliwości pod niskie. Dzieje się tak wtedy, gdy próbujemy przetworzyć sygnał analogowy na cyfrowy, ale robimy to zbyt rzadko (zbyt mała częstotliwość próbkowania).

1. **Intuicja**: Efekt „kręcących się kół” w filmach Każdy z nas widział w filmie samochód jadący szybko do przodu, którego felgi wydają się kręcić powoli do tyłu lub stać w miejscu. To jest właśnie aliasing wizualny.
   - Kamera robi np. 24 zdjęcia na sekundę (próbkuje obraz).
   - Jeśli koło obraca się tak szybko, że między jednym a drugim zdjęciem wykona prawie pełny obrót, nasz mózg „połączy kropki” najkrótszą drogą.
   - Zamiast zobaczyć szybki ruch do przodu, widzimy powolny ruch wsteczny. Kamera nas oszukała, bo była za wolna.
2. Mechanizm powstawania: „Łączenie kropek” Wyobraź sobie bardzo szybką sinusoidę (np. 100 Hz – drga 100 razy na sekundę). Jeśli zmierzysz jej wartość (spróbkujesz ją) tylko 110 razy na sekundę, to Twoje punkty pomiarowe będą wypadać w tak niefortunnych miejscach, że gdybyś chciał je połączyć płynną linią, wyjdzie Ci bardzo wolna fala (10 Hz).

   Komputer nie wie, co działo się między próbkami. On widzi tylko punkty i zakłada, że sygnał jest najprostszą możliwą falą, która przez te punkty przechodzi. W ten sposób sygnał o wysokiej częstotliwości „założył maskę” sygnału o niskiej częstotliwości.
3. Dlaczego to jest groźne w pomiarach? Jeśli mierzysz drgania maszyny i występuje tam szum o wysokiej częstotliwości (np. 950 Hz), a Twój przyrząd próbkuje z częstotliwością 1000 Hz, to aliasing sprawi, że na wykresie zobaczysz wyraźny prążek o częstotliwości 50 Hz.
   - Zaczniesz szukać usterki związanej z siecią elektryczną (50 Hz).
   - W rzeczywistości problemem jest zupełnie inny element drgający z częstotliwością 950 Hz.
   - Błąd aliasingu jest nieodwracalny – po nagraniu sygnału nie da się już odróżnić, co było prawdziwą informacją, a co „duchem” powstałym przez zbyt wolne próbkowanie.
4. Jak się przed tym bronić?

   Są dwa sposoby:

   (a) Zasada 2x (Twierdzenie Nyquista): Próbkuj zawsze przynajmniej 2 razy szybciej niż wynosi najwyższa częstotliwość, której się spodziewasz. Jeśli sygnał ma 20 kHz (granica słuchu), płyta CD musi mieć ponad 40 kHz (stąd standard 44.1 kHz).

   (b) Filtr Antyaliasingowy: Zanim zamienisz sygnał na cyfrowy, przepuść go przez filtr dolnoprzepustowy, który „odetnie” wszystkie częstotliwości wyższe niż połowa Twojego próbkowania. Lepiej stracić trochę wysokich tonów, niż pozwolić im „udawać” niskie tony i zepsuć cały pomiar.

3 Część praktyczna – Zadania

Uruchom środowisko Jupyter-Lab, utwórz nowy `Notebook` i wykonaj poszczególne podpunkty.

4 Zadania do wykonania

1. **Zadanie 1**: Wyznacz analitycznie współczynniki a_k, b_k fali prostokątnej (SymPy).
2. **Zadanie 2**: Zrekonstruuj sygnał z c_k, φ_k i porównaj z oryginałem dla różnych N.

Zadania 1 i 2 zostały uwzględnione w jednym kodzie przedstawionym na listingu 4.1.

Listing 4.1: Analityczne wyznaczenie współczynników ak, bk fali prostokątnej oraz rekonstukcja sygnału z ze współczynników ck i ϕk

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

# Definicja symboli
t, T, k = sp.symbols("t T k", real=True, positive=True)
A = 1
T_val = 2 * np.pi
omega0 = 2 * sp.pi / T

# Definicja funkcji prostokatnej (jeden okres)
f_t = sp.Piecewise((A, t < T / 2), (-A, t >= T / 2))

# Obliczanie a0
a0 = (1 / T) * sp.integrate(f_t, (t, 0, T))

# Obliczanie ak i bk (symbolicznie)
ak_expr = (2 / T) * sp.integrate(
    f_t * sp.cos(k * (2 * sp.pi / T) * t), (t, 0, T)
)
bk_expr = (2 / T) * sp.integrate(
    f_t * sp.sin(k * (2 * sp.pi / T) * t), (t, 0, T)
)

print(f"a0 = {a0}")
print(f"ak = {sp.simplify(ak_expr)}")
print(f"bk = {sp.simplify(bk_expr)}")


# Konwersja na funkcje numeryczne
def get_coeffs(n_max):
    coeffs = []
    for i in range(1, n_max + 1):
```

Listing 4.1: Analityczne wyznaczenie współczynników ak, bk fali prostokątnej oraz rekonstukcja sygnału z ze współczynników ck i ϕk (c.d.)

```
        a_val = float(ak_expr.subs({k: i, T: T_val}))
        b_val = float(bk_expr.subs({k: i, T: T_val}))
        ck = np.sqrt(a_val**2 + b_val**2)
        phi_k = np.arctan2(b_val, a_val)
        coeffs.append((a_val, b_val, ck, phi_k))
    return coeffs


# Rekonstrukcja
t_np = np.linspace(0, 2 * T_val, 10000)
n_harmonics = 10
coeffs = get_coeffs(n_harmonics)

f_recon = float(a0.subs(T, T_val))
for i, (ak, bk, ck, phik) in enumerate(coeffs):
    # Wykorzystujemy postac amplitudowo-fazowa: ck * cos(k*w0*t -
    ↪  phik)
    # Uwaga: phik z atan2(bk, ak) odpowiada przesunieciu w sin/cos
    k_idx = i + 1
    f_recon += ck * np.cos(k_idx * (2 * np.pi / T_val) * t_np -
    ↪  phik)

# Wykres
plt.figure(figsize=(10, 5))
f_orig = np.where((t_np % T_val) < T_val / 2, 1, -1)
plt.plot(t_np, f_orig, "k--", label="Oryginał", alpha=0.3)
plt.plot(t_np, f_recon, label=f"Rekonstrukcja (N={n_harmonics})")
plt.title("Szereg Fouriera - faza i amplituda")
plt.legend()
plt.grid(True)
plt.show()
```

3. **Zadanie 3**: Wykonaj analizę widmową (FFT) sumy dwóch sinusów o różnych amplitudach i częstotliwościach.

Listing 4.2: Analiza FFT sygnału będącego złożeniem dwóch funkcji harmonicznych o różnych częstotliwościach

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

# Parametry sygnalu
fs = 1000  # Czestotliwosc probkowania [Hz]
T = 1.0  # Czas trwania sygnalu [s]
n = int(fs * T)  # Liczba probek
```

Listing 4.2: Analiza FFT sygnału będącego złożeniem dwóch funkcji harmonicznych o różnych częstotliwościach (c.d.)

```
t = np.linspace(0, T, n, endpoint=False)

# Sygnal: suma dwoch sinusow (50Hz i 120Hz)
f1, f2 = 50, 120
y = 0.6 * np.sin(2 * np.pi * f1 * t) + 1.0 * np.sin(2 * np.pi * f2 *
↪  t)

# Obliczanie FFT
yf = np.fft.fft(y)
xf = np.fft.fftfreq(n, 1 / fs)

# Widmo amplitudowe (normalizacja i branie tylko dodatnich
↪  czestotliwosci)
amplitude_spectrum = 2.0 / n * np.abs(yf[: n // 2])
frequencies = xf[: n // 2]

# Wykresy
plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)
plt.plot(t[:200], y[:200])
plt.title("Sygnał w dziedzinie czasu (fragment)")
plt.xlabel("Czas [s]")
plt.grid(True)

plt.subplot(2, 1, 2)
plt.stem(frequencies, amplitude_spectrum)
plt.title("Widmo amplitudowe (FFT)")
plt.xlabel("Częstotliwość [Hz]")
plt.ylabel("Amplituda")
plt.xlim(0, 200)  # Ograniczenie widoku dla czytelnosci
plt.grid(True)

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

4. **Zadanie 4**: Przeprowadź próbkowanie i rekonstrukcję sinc, wyznaczając błąd.

Listing 4.3: Demonstracja próbkowania sygnału sinus i odtworzenia sygnału z próbek za pomocą funkcji sincus

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

f_sig = 2  # Częstotliwość sygnału badanego
fs = 20  # Częstotliwość próbkowania sygnału
```

Listing 4.3: Demonstracja próbkowania sygnału sinus i odtworzenia sygnału z próbek za pomocą funkcji sincus (c.d.)

```
T_obs = 1.0  # Okres obserwacji
t_fine = np.linspace(0, T_obs, 10000)
y_orig = np.sin(2 * np.pi * f_sig * t_fine)
t_samp = np.arange(0, T_obs, 1 / fs)
y_samp = np.sin(2 * np.pi * f_sig * t_samp)

y_rec = np.zeros_like(t_fine)
for i in range(len(t_samp)):
    y_rec += y_samp[i] * np.sinc(fs * (t_fine - t_samp[i]))

plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(t_fine, y_orig, alpha=0.5)
plt.plot(t_fine, y_rec, "--")
plt.stem(t_samp, y_samp, "r")
plt.title("Rekonstrukcja")
plt.subplot(2, 1, 2)
plt.plot(t_fine, y_orig - y_rec, "r")
plt.title("Blad rekonstrukcji")
plt.tight_layout()
plt.show()
```

Listing 4.4: Demonstracja próbkowania i otdtwarzania sygnału prostokątnego

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

f_sig = 2  # Częstotliwość sygnału badanego
fs = 20  # Częstotliwość próbkowania sygnału
T_obs = 1.0  # Okres obserwacji
t_fine = np.linspace(0, T_obs, 10000)
# Poniżej sygnał prostokątny z sinusa przez np.sign()
y_orig = np.sign(np.sin(2 * np.pi * f_sig * t_fine))
t_samp = np.arange(0, T_obs, 1 / fs)
# Poniżej sygnał prostokątny z sinusa przez np.sign()
y_samp = np.sign(np.sin(2 * np.pi * f_sig * t_samp))

y_rec = np.zeros_like(t_fine)
for i in range(len(t_samp)):
    y_rec += y_samp[i] * np.sinc(fs * (t_fine - t_samp[i]))

plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(t_fine, y_orig, alpha=0.5)
plt.plot(t_fine, y_rec, "--")
```

Listing 4.4: Demonstracja próbkowania i otdtwarzania sygnału prostokątnego (c.d.)

```
plt.stem(t_samp, y_samp, "r")
plt.title("Rekonstrukcja")
plt.subplot(2, 1, 2)
plt.plot(t_fine, y_orig - y_rec, "r")
plt.title("Blad rekonstrukcji")
plt.tight_layout()
plt.show()
```

5 Projekt

Zadanie

Na podstawie numeru zadania dla ćwiczenia laboratoryjnego „Pomiary w dziedzinie częstotliwości”, wyciągnij zadaną funkcję z pliku `Dane do ćwiczenia ,,Pomiary sygnałów w dziedzinie częstotliwości''`. Wypróbuj obie metody: analityczną i numeryczną wyznaczenia współczynników ak, bk oraz ck i przesunięcia fazowego. Prawidłowo wyznaczone ck i przesunięcia fazowe pozwolą na wykonanie symulacji w programie `Multisim` oraz wykonanie ćwiczenia laboratoryjnego na przyrządzie „Sumator Widm”.

Współczynniki ak,bk szeregu sinusów i cosinusów można wyznaczyć metodami analitycznymi, chociaż nie zawsze jest to możliwe. Kod do obliczeń przedstawia listing 5.1.

Uwaga!

Obliczenie wszystkich harmonicznych metodą analityczną, zwłaszcza w przypadku funkcji rozłożonych na przedziały, może zająć nawet kilkanaście minut!

Uwaga!

W niektórych przypadkach całkowanie metodą analityczną nie będzie możliwe w ogóle. Pozostaje wtedy zastosowanie całkowania numerycznego, ale należy pamiętać o wykluczeniu z całkowania punktów osobliwych, gdyż numeryczny algorytm całkujący nie będzie w stanie poradzić sobie z nieskończonościami.

Listing 5.1: Wyznaczenie współczynników szeregu Fouriera metodami analitycznymi.

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
import time


def calculate_fourier_series(func_expr, var, period, n_harmonics):
    """
    Oblicza współczynniki szeregu Fouriera dla zadanej funkcji.
    """
    T = period
    omega = 2 * sp.pi / T

    a_coeffs = []
    b_coeffs = []
    c_coeffs = []
    phases = []
```

Listing 5.1: Wyznaczenie współczynników szeregu Fouriera metodami analitycznymi. (c.d.)

```
    # Obliczanie a0
    a0 = (2 / T) * sp.integrate(func_expr, (var, 0, T))
    a_coeffs.append(float(a0.evalf()))
    b_coeffs.append(0.0)
    c_coeffs.append(float(a0.evalf()))
    phases.append(0.0)

    print(f"Obliczanie {n_harmonics} harmonicznych...Cierpliwie
    ↪  poczekaj.")

    for k in range(1, n_harmonics + 1):
        print(f"Obliczam {k} harmoniczną.", end="")
        start = time.perf_counter()
        ak = (2 / T) * sp.integrate(
            func_expr * sp.cos(k * omega * var), (var, 0, T)
        )
        bk = (2 / T) * sp.integrate(
            func_expr * sp.sin(k * omega * var), (var, 0, T)
        )

        ak_val = float(ak.evalf())
        bk_val = float(bk.evalf())
        ck_val = np.sqrt(ak_val**2 + bk_val**2)
        pk_val = np.arctan2(bk_val, ak_val)

        a_coeffs.append(ak_val)
        b_coeffs.append(bk_val)
        c_coeffs.append(ck_val)
        phases.append(pk_val)
        end = time.perf_counter()
        print(f" <- Wykonanie trwało {(end - start):.1f} s.")
    return a_coeffs, b_coeffs, c_coeffs, phases


def main():
    x = sp.Symbol("x")

    # Definicja funkcji przedziałami:
    # 0 dla (0, 1/6)
    # sin(3*pi*(x-1/6)) dla [1/6, 5/6]
    # 0 dla (5/6, 1]
    f_x = sp.Piecewise(
        (0, x < 1 / 6),
        (sp.sin(3 * sp.pi * (x - 1 / 6)), (x >= 1 / 6) & (x <= 5 / 6)),
        (0, x > 5 / 6),
    )
```

Listing 5.1: Wyznaczenie współczynników szeregu Fouriera metodami analitycznymi. (c.d.)

```
T_val = 1.0  # Okres
N = 10  # Liczba harmonicznych

print(f"Funkcja: Piecewise defined")
print(f"Okres T = {T_val}, Liczba harmonicznych N = {N}")

# Obliczenia
a, b, c, phi = calculate_fourier_series(f_x, x, T_val, N)

# Tabela
headers = ["k", "ak", "bk", "ck (Amplituda)", "yk (Faza)"]
print("\nTabela współczynników:")
print(
    f"{headers[0]:<4} | {headers[1]:<8} | {headers[2]:<8} |
    ↪  {headers[3]:<15} | {headers[4]:<15}"
)
print("-" * 75)
for k in range(len(a)):
    phi_pi = phi[k] / np.pi
    phi_str = f"{phi_pi:.4f} * pi" if abs(phi[k]) > 1e-9 else "0.0000"
    print(
        f"{k:<4} | {a[k]:<8.4f} | {b[k]:<8.4f} | {c[k]:<15.4f} |
        ↪  {phi_str:<15}"
    )

# Dane do wykresu
x_vals = np.linspace(0, T_val, 1000)
# Używamy modules='numpy' dla Piecewise (sp.lambdify to obsługuje)
f_num = sp.lambdify(x, f_x, modules=["numpy"])
y_orig = f_num(x_vals)

omega_val = 2 * np.pi / T_val
y_fourier = np.full_like(x_vals, a[0] / 2)

for k in range(1, N + 1):
    y_fourier += c[k] * np.cos(k * omega_val * x_vals - phi[k])

# Wykres
plt.figure(figsize=(12, 8))
plt.plot(
    x_vals, y_orig, "k-", label="Oryginalna f(x)", linewidth=3,
    ↪  zorder=10
)
plt.plot(
    x_vals,
```

Listing 5.1: Wyznaczenie współczynników szeregu Fouriera metodami analitycznymi. (c.d.)

```
        y_fourier,
        "r--",
        label=f"Suma szeregu (N={N})",
        linewidth=2,
        zorder=11,
    )

    for k in range(1, N + 1):
        if c[k] > 0.01:  # tylko istotne dla czytelności
            y_k = c[k] * np.cos(k * omega_val * x_vals - phi[k])
            plt.plot(
                x_vals, y_k, alpha=0.4, linewidth=1, label=f"Harmoniczna
                ↪  k={k}"
            )

    if abs(a[0]) > 1e-4:
        plt.axhline(
            y=a[0] / 2,
            color="gray",
            linestyle=":",
            label="Składowa stała a0/2",
        )

    plt.title("Szereg Fouriera dla funkcji zdefiniowanej przedziałami")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")
    plt.grid(True, linestyle="--", alpha=0.5)
    plt.tight_layout()
    plt.show()


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

Kosztem utraty dokładności, w szczególności w przypadkach ekstremalnych, można wykorzystać całkowanie numeryczne, które wykona się błyskawicznie. Kod przedstawiono na listing'u 5.2.

Listing 5.2: Wyznaczanie współczynników szeregu Fouriera metodami całkowania numerycznego

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

Listing 5.2: Wyznaczanie współczynników szeregu Fouriera metodami całkowania numerycznego (c.d.)

```
def calculate_fourier_series_num(func, period, n_harmonics):
    """
    Oblicza współczynniki szeregu Fouriera metodą numeryczną (quad).
    """
    T = period
    omega = 2 * np.pi / T

    a_coeffs = []
    b_coeffs = []
    c_coeffs = []
    phases = []

    # Obliczanie a0
    # a0 = (2/T) * integral(f(x) dx) from 0 to T
    res_a0, _ = quad(func, 0, T)
    a0 = (2 / T) * res_a0

    a_coeffs.append(a0)
    b_coeffs.append(0.0)
    c_coeffs.append(a0)
    phases.append(0.0)

    print(f"Obliczanie {n_harmonics} harmonicznych (numerycznie)...")

    for k in range(1, n_harmonics + 1):
        print(f"Obliczam {k} harmoniczną.", end="")
        start = time.perf_counter()

        # Definicje funkcji podcałkowych dla ak i bk
        def integrand_a(x):
            return func(x) * np.cos(k * omega * x)

        def integrand_b(x):
            return func(x) * np.sin(k * omega * x)

        ak_res, _ = quad(integrand_a, 0, T)
        bk_res, _ = quad(integrand_b, 0, T)

        ak = (2 / T) * ak_res
        bk = (2 / T) * bk_res

        ck = np.sqrt(ak**2 + bk**2)
        pk = np.arctan2(bk, ak)
```

Listing 5.2: Wyznaczanie współczynników szeregu Fouriera metodami całkowania numerycznego (c.d.)

```
        a_coeffs.append(ak)
        b_coeffs.append(bk)
        c_coeffs.append(ck)
        phases.append(pk)
        end = time.perf_counter()
        print(f" <- Wykonanie trwało {(end - start):.1f} s.")
    return a_coeffs, b_coeffs, c_coeffs, phases


def f_test(x):
    """
    Funkcja zdefiniowana przedziałami:
    0 dla (0, 1/6)
    sin(3*pi*(x-1/6)) dla [1/6, 5/6]
    0 dla (5/6, 1]
    """
    if 1 / 6 <= x <= 5 / 6:
        return np.sin(3 * np.pi * (x - 1 / 6))
    else:
        return 0.0


def main():
    T_val = 1.0
    N = 10

    print(f"Obliczenia dla funkcji przedziałowej (okres T={T_val})")

    # Obliczenia numeryczne
    a, b, c, phi = calculate_fourier_series_num(f_test, T_val, N)

    # Tabela
    headers = ["k", "ak", "bk", "ck (Amplituda)", "yk (Faza)"]
    print("\nTabela współczynników (numerycznie):")
    print(
        f"{headers[0]:<4} | {headers[1]:<8} | {headers[2]:<8} |
        ↪  {headers[3]:<15} | {headers[4]:<15}"
    )
    print("-" * 75)
    for k in range(len(a)):
        phi_pi = phi[k] / np.pi
        phi_str = f"{phi_pi:.4f} * pi" if abs(phi[k]) > 1e-9 else "0.0000"
        print(
            f"{k:<4} | {a[k]:<8.4f} | {b[k]:<8.4f} | {c[k]:<15.4f} |
            ↪  {phi_str:<15}"
        )
```

Listing 5.2: Wyznaczanie współczynników szeregu Fouriera metodami całkowania numerycznego (c.d.)

```
# Dane do wykresu
x_vals = np.linspace(0, T_val, 1000)
y_orig = np.array([f_test(val) for val in x_vals])

omega_val = 2 * np.pi / T_val
y_fourier = np.full_like(x_vals, a[0] / 2)

for k in range(1, N + 1):
    y_fourier += c[k] * np.cos(k * omega_val * x_vals - phi[k])

# Wykres
plt.figure(figsize=(12, 8))
plt.plot(
    x_vals, y_orig, "k-", label="Oryginalna f(x)", linewidth=3,
    ↪ zorder=10
)
plt.plot(
    x_vals,
    y_fourier,
    "r--",
    label=f"Suma szeregu (N={N})",
    linewidth=2,
    zorder=11,
)

for k in range(1, N + 1):
    if c[k] > 0.01:
        y_k = c[k] * np.cos(k * omega_val * x_vals - phi[k])
        plt.plot(
            x_vals, y_k, alpha=0.4, linewidth=1, label=f"Harmoniczna
            ↪ k={k}"
        )

if abs(a[0]) > 1e-4:
    plt.axhline(
        y=a[0] / 2,
        color="gray",
        linestyle=":",
        label="Składowa stała a0/2",
    )

plt.title("Szereg Fouriera - Całkowanie Numeryczne (scipy.quad)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")
```

Listing 5.2: Wyznaczanie współczynników szeregu Fouriera metodami całkowania numerycznego (c.d.)

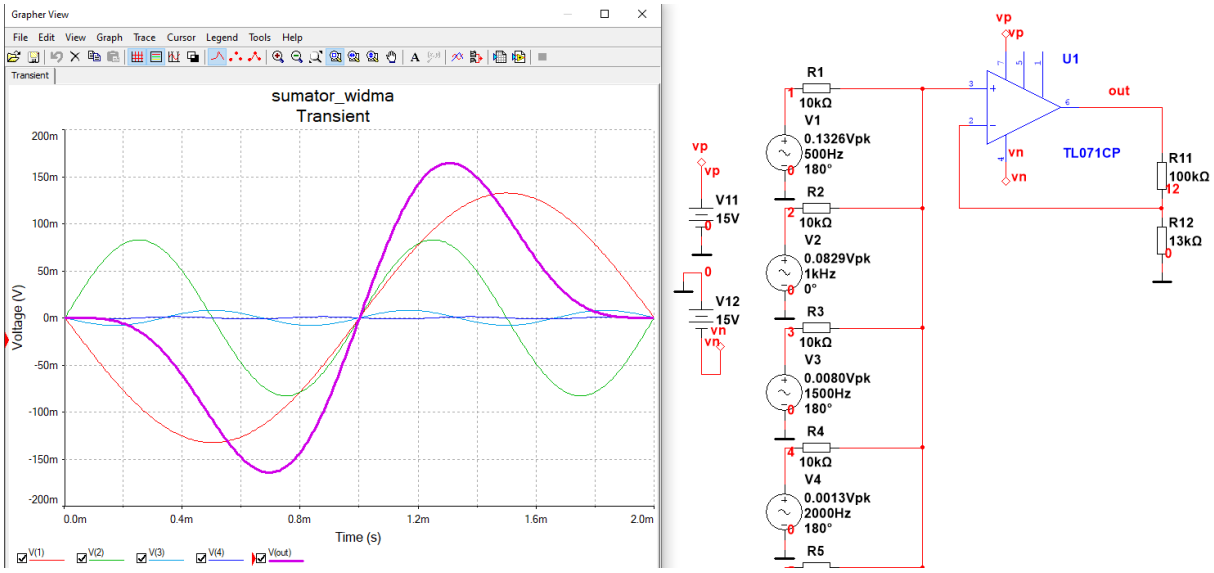
```
    plt.grid(True, linestyle="--", alpha=0.5)
    plt.tight_layout()
    plt.show()


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

Warto wiedzieć

Wyznaczone współczynniki c_k i przesunięcia fazowe ϕ_k dotyczą sygnałów typu $cos()$. Przesunięcia fazowe są we wzorze (2) z minusem! Warto sprawdzić jakie przesunięcie jest wypadkowe po podstawieniu wartości. Zazwyczaj w generatorach sygnału, w symulacji mamy do czynienia z funkcjami $sin()$. Aby przedstawić funkcję $cos()$ wykorzystując funkcję $sin()$ należy dodać przesunięcie fazowe $\frac{\pi}{2}$, czyli dodać 90^o.

Symulację `Transient` w Multisim, złożenia sygnału z sygnałów sinusoidalnych o kolejnych częstotliwościach harmonicznych przedstawia rys. 1



Rysunek 1: Multisim: Symulacja złożenia sygnałów sinusoidalnych o kolejnych 10 częstotliwościach harmonicznych z wykorzystaniem układu sumatora na wzmacniaczu operacyjnym TL071. Na rysunku przedstawiony tylko fragment schematu z pierwszymi czterema generatorami. Częstotliwość podstawową ustawiono na f0=500 Hz.

6 Pytania kontrolne i odpowiedzi

1. **Pytanie: Jakie warunki musi spełniać funkcja, aby można ją było rozłożyć na szereg Fouriera?**
   Odpowiedź: Funkcja $f(t)$ musi spełniać tzw. **warunki Dirichleta**:
   - Musi być okresowa.
   - W obrębie jednego okresu musi być bezwzględnie całkowalna ($\int |f(t)|dt < \infty$).
   - Musi mieć skończoną liczbę ekstremów lokalnych (maksimów i minimów) w jednym okresie.
   - Musi mieć skończoną liczbę punktów nieciągłości pierwszego rodzaju (skokowych).

   Dla sygnałów rzeczywistych występujących w technice warunki te są niemal zawsze spełnione.

2. **Pytanie: Czy przebieg o dyskretnym widmie częstotliwościowym musi być okresowy?**
   Odpowiedź: Tak. Dyskretne (prążkowe) widmo częstotliwościowe jest bezpośrednią cechą sygnałów okresowych. Jeśli widmo składa się z odizolowanych „pików" o częstotliwościach będących wielokrotnościami częstotliwości podstawowej f_0, to sygnał w dziedzinie czasu powtarza się co $T = 1/f_0$. (Wyjątkiem są sygnały quasi-okresowe, gdzie stosunki częstotliwości są niewymierne, ale w klasycznej analizie Fouriera przyjmujemy tę zasadę).

3. **Pytanie: Jakie są ograniczenia syntezy zadanego przebiegu za pomocą wykorzystywanego sumatora harmonicznych?**
   Odpowiedź:
   - **Liczba harmonicznych:** Sumator ma tylko 10 składowych. Sygnały o ostrych krawędziach (np. prostokąt, piła) wymagają teoretycznie nieskończonej liczby harmonicznych dla idealnego odwzorowania.
   - **Pasmo przenoszenia:** Układy sumujące mają skończone pasmo (BW), co może tłumić wyższe harmoniczne.
   - **Precyzja nastaw:** Potencjometry mają ograniczoną rozdzielczość, co utrudnia idealne ustawienie fazy i amplitudy.
   - **Szumy i zniekształcenia:** Każdy stopień wzmacniający w sumatorze wprowadza własne szumy i nieliniowości.

4. **Pytanie: Jak na ekranie oscyloskopu odczytać fazy składowych?**
   Odpowiedź: Fazę odczytujemy porównując składową k-tą z sygnałem odniesienia (np. harmoniczną podstawową $k = 1$). Mierzymy czas Δt między odpowiadającymi sobie punktami (np. przejście przez zero w górę) obu przebiegów. Przesunięcie fazowe

w stopniach wynosi:

$$\varphi_k = \frac{\Delta t}{T_k} \cdot 360^\circ$$

gdzie T_k to okres k-tej harmonicznej. Alternatywnie można użyć krzywych Lissajous (tryb X-Y oscyloskopu).

5. **Pytanie: Jak należy regulować fazy składowych fourierowskich?**
   Odpowiedź: Regulacja odbywa się za pomocą dedykowanych potencjometrów fazy w sumatorze. Należy pamiętać, że zmiana fazy k-tej harmonicznej o kąt φ przesuwa ją w czasie o $\Delta t = \varphi/(k\omega_0)$. W praktyce dąży się do takiego ustawienia, aby suma składowych tworzyła pożądany kształt (np. wszystkie sinusy startujące w zerze dla symetrii nieparzystej).

6. **Pytanie: Czym spowodowane jest odchylenie otrzymanego kształtu zrekonstruowanej funkcji od założonego?**
   Odpowiedź:
   - **Błąd obcięcia (truncation error):** Brak harmonicznych powyżej 10-tej (najważniejszy czynnik).
   - **Zjawisko Gibbsa:** Charakterystyczne oscylacje przy nieciągłościach, których nie da się wyeliminować sumując skończoną liczbę fal sinus.
   - **Niedokładność kalibracji:** Błędy w ustawieniu amplitud i faz przez operatora.
   - **Wpływ aparatury:** Niedoskonałości sumatora i oscyloskopu (np. przesunięcia fazowe wprowadzane przez same wzmacniacze operacyjne urządzenia).

7 Podsumowanie i wnioski

Wnioski z ćwiczenia powinny obejmować analizę wpływu liczby harmonicznych na jakość rekonstrukcji sygnału oraz porównanie wyników uzyskanych analitycznie (Python) z wynikami eksperymentalnymi (sumator i oscyloskop).

Informacje dodatkowe

Więcej informacji dostępne jest na naszym Moodle: https://study.engined.eu/mod/lesson/view.php?id=784. W szczególności polecam zajrzeć do filmiku z symulacjami w LTSpice na stronie: https://study.engined.eu/mod/lesson/view.php?id=785. Można go także obejrzeć bezpośrednio na serwisie Youtube: https://www.youtube.com/watch?v=RmC6vepoNwc.

7.1 Wymagane oprogramowanie

Do wykonania ćwiczenia wymagany jest NI Multisim, Jupyter-lab i Python 3 z zainstalowanymi modułami:

- `Sympy` — do obliczeń symbolicznych i transformat,
- `Matplotlib` — do wizualizacji wyników,
- `Numpy` — do podstawowych struktur danych,
- `Scipy` — do obliczeń na sygnałach

Plac Politechniki 1
00-661 Warszawa
www.pw.edu.pl

8 Autorzy i historia opracowania

- dr inż. Dariusz Tefelski - wersja z 2026r.