PRACOWNIA MIERNICTWA ELEKTRONICZNEGO / INTERNETU RZECZY

ĆWICZENIE NR 7

Temat:

Programowa symulacja modulacji sygnałów radiowych

AM, FM, ASK, FSK i BPSK — generacja, analiza widmowa i wpływ szumu

Opracowanie wstępne:
Sinusoida • amplituda, częstotliwość i faza • próbkowanie i twierdzenie Nyquista • AM i FM • ASK, FSK i BPSK • FFT • SNR

Cel ćwiczenia

Celem jest prześledzenie całego łańcucha: informacja → sygnał modulujący → fala nośna → sygnał zmodulowany → widmo → wpływ zakłóceń. Wszystkie operacje są wykonywane programowo; nie jest potrzebny generator, oscyloskop ani nadajnik radiowy.

Wymagane oprogramowanie

Element	Wymaganie
Python	wersja 3.10 lub nowsza (może być również Google Colab / Jupyter)
NumPy	generowanie i obliczenia na sygnałach
Matplotlib	wykresy czasowe i widma
Sprzęt	zwykły komputer; ćwiczenie nie wykorzystuje sprzętu pomiarowego

Ważna uwaga: W programie używamy nośnej 20 kHz, a nie np. 100 MHz. Zjawiska matematyczne są identyczne, natomiast znacznie niższa częstotliwość pozwala wygodnie próbkować sygnał i szybko wykonywać obliczenia.

1. Co właściwie będziemy symulować?

Fala sinusoidalna może zostać opisana trzema podstawowymi parametrami: amplitudą A, częstotliwością f oraz fazą φ. W telekomunikacji informację można przenieść przez kontrolowaną zmianę jednego z tych parametrów fali nośnej.

Modulacja	Co niesie informację?	Przykład
AM	amplituda nośnej	radiofonia AM
FM	częstotliwość chwilowa	radiofonia FM
ASK	jedna z kilku amplitud	prosta transmisja cyfrowa
FSK	jedna z kilku częstotliwości	modemy, telemetria
BPSK	faza 0° lub 180°	łączność cyfrowa

1.1. Próbkowanie

Komputer nie zapisuje sygnału w sposób ciągły. Zapisuje kolejne próbki. Częstotliwość próbkowania f_s określa, ile próbek na sekundę pobieramy. W tym ćwiczeniu przyjmujemy f_s = 200 kHz, czyli odstęp między próbkami wynosi 5 µs.

Warunek Nyquista mówi, że aby uniknąć aliasingu, częstotliwość próbkowania powinna być większa niż dwukrotność najwyższej częstotliwości obecnej w sygnale:

$$f_s > 2 f_{max}$$

Co oznacza zły wynik: Jeżeli f_s jest zbyt małe, wykres może sugerować zupełnie inną częstotliwość niż rzeczywista. To nie jest „dziwne zachowanie sinusoidy", lecz aliasing — błąd wynikający z niedostatecznego próbkowania.

1.2. Przygotowanie środowiska

Instalacja bibliotek (wykonujemy tylko raz)

```
python -m pip install numpy matplotlib
```

Kod bazowy — skopiuj na początek programu

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

fs = 200_000        # częstotliwość próbkowania [Hz]
T  = 0.010          # czas obserwacji [s]
t  = np.arange(0, T, 1/fs)

fm = 1_000          # częstotliwość informacji [Hz]
fc = 20_000         # częstotliwość nośnej [Hz]

message = np.sin(2*np.pi*fm*t)
carrier = np.cos(2*np.pi*fc*t)
```

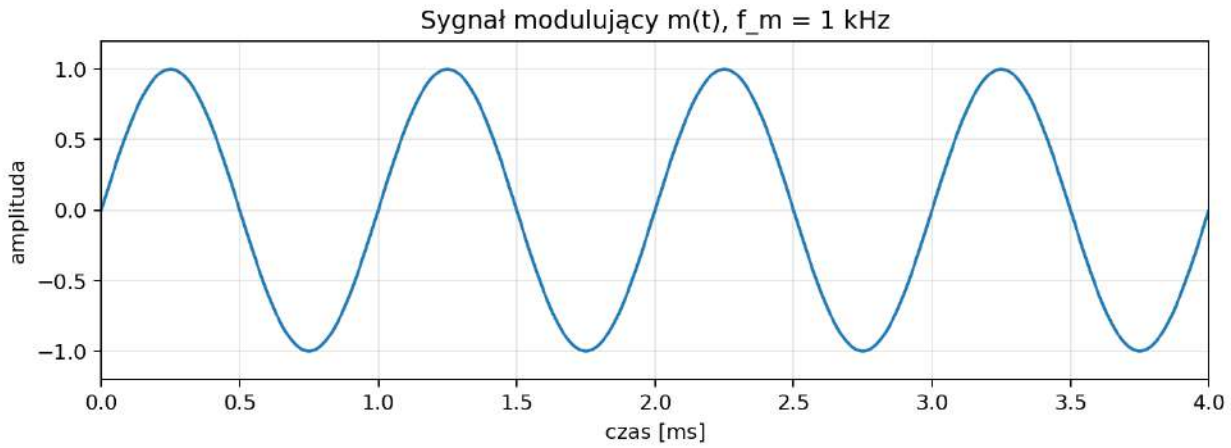
Sprawdzenie: Długość tablicy t powinna wynosić około 2000 próbek: 0,010 s × 200 000 próbek/s = 2000. Jeśli otrzymujesz wartość zupełnie inną, sprawdź jednostki i zapis 200_000.

2. Moduł A — fala informacyjna i fala nośna

Krok A1. Wygeneruj sygnał informacyjny

Sygnałem informacyjnym będzie sinusoida 1 kHz. W rzeczywistej transmisji mogłaby to być mowa, dźwięk, temperatura lub dowolna inna wielkość zmieniona na sygnał elektryczny.

```
plt.plot(t*1000, message)
plt.xlim(0, 4)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('Sygnał modulujący 1 kHz')
plt.grid()
plt.show()
```

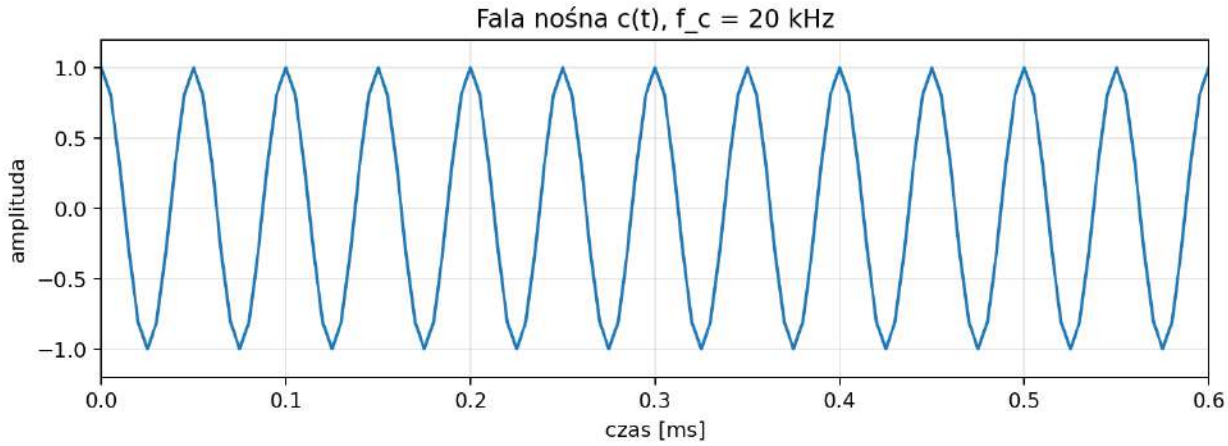


Rys. 1. Sygnał modulujący. Okres 1 kHz wynosi 1 ms.

Jak interpretować wynik: W przedziale 0–4 ms powinny być widoczne dokładnie cztery pełne okresy. Jeżeli jest ich osiem, prawdopodobnie ustawiono 2 kHz; jeżeli dwa — około 500 Hz.

Krok A2. Wygeneruj falę nośną

```
plt.plot(t*1000, carrier)
plt.xlim(0, 0.6)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('Fala nośna 20 kHz')
plt.grid()
plt.show()
```



Rys. 2. Fala nośna 20 kHz. Jej okres wynosi 50 µs.

Sens fizyczny: Nośna sama w sobie nie zawiera jeszcze informacji. Jest „szybkim” sygnałem, którego parametr zmienimy zgodnie z wolniejszym sygnałem 1 kHz.

3. Moduł B — modulacja amplitudy AM

W klasycznej AM informacja steruje amplitudą nośnej. Dla sygnału sinusoidalnego użyjemy zależności:

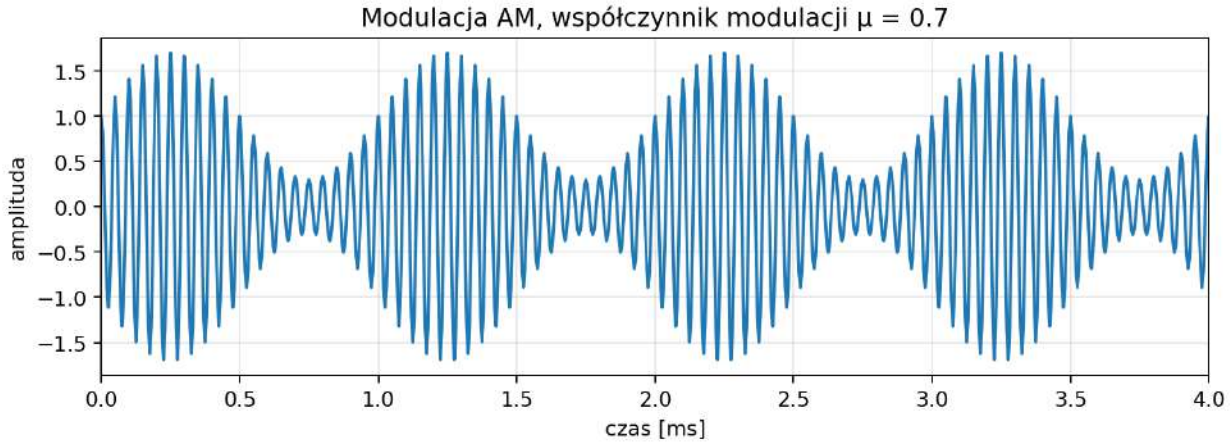
$$s_{AM}(t) = [1 + \mu \cdot m(t)] \cdot \cos(2\pi f_c t)$$

μ jest współczynnikiem głębokości modulacji. Przy μ = 0 nie ma modulacji; przy μ = 1 obwiednia dotyka zera; dla μ > 1 występuje przemodulowanie.

Krok B1. Wygeneruj AM

```
mu = 0.7
AM = (1 + mu*message) * carrier

plt.plot(t*1000, AM)
plt.xlim(0, 4)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('AM, mu = 0.7')
plt.grid()
plt.show()
```



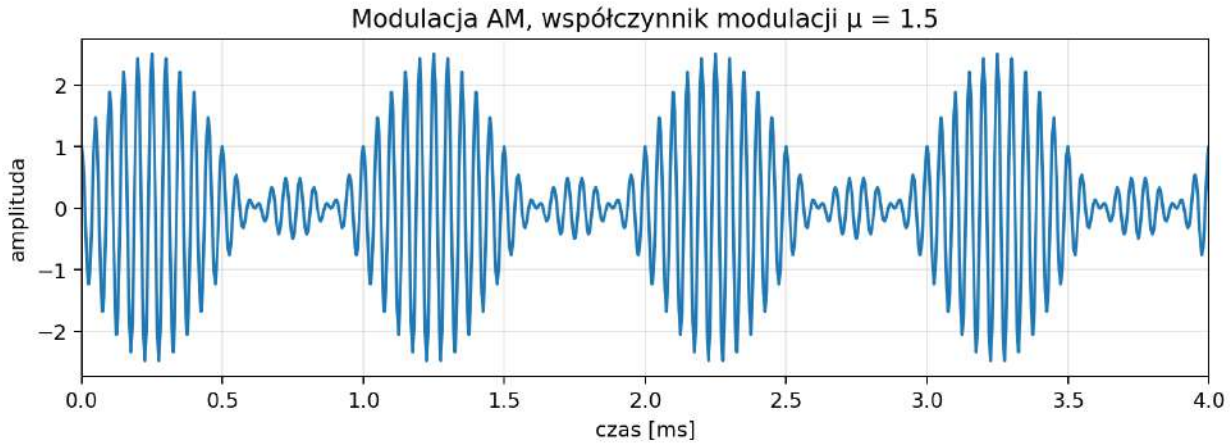
Rys. 3. Prawidłowa modulacja AM dla μ = 0,7.

Co powinno być widoczne: Szybkie oscylacje 20 kHz są „opakowane” wolną obwiednią 1 kHz. Dla μ = 0,7 amplituda obwiedni zmienia się od 1−0,7 = 0,3 do 1+0,7 = 1,7. Obwiednia nie przecina zera.

Krok B2. Zbadaj głębokość modulacji

Powtórz obliczenia dla μ = 0,25; 0,50; 1,00 oraz 1,50. Nie zmieniaj innych parametrów.

μ	Przewidywany zakres obwiedni	Interpretacja
0,25	0,75 … 1,25	modulacja płytka
0,50	0,50 … 1,50	modulacja umiarkowana
1,00	0 … 2,00	100% modulacji; obwiednia dotyka zera
1,50	−0,50 … 2,50	przemodulowanie; detekcja obwiedni byłaby zniekształcona



Rys. 4. Przemodulowanie AM przy μ = 1,5.

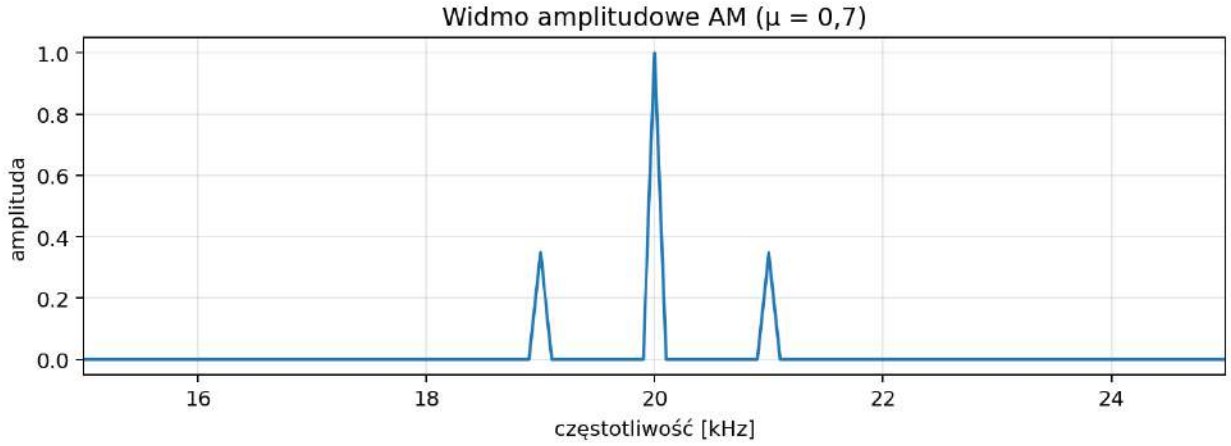
Co oznacza „ujemna obwiednia”: Nie oznacza ujemnej fizycznej amplitudy nadajnika. Matematycznie czynnik [1+μm(t)] zmienia znak, co odpowiada dodatkowej zmianie fazy nośnej o 180°. Prosty detektor obwiedni nie potrafi tego poprawnie odtworzyć.

4. Moduł C — widmo sygnału AM i FFT

Wykres w dziedzinie czasu mówi, jak sygnał zmienia się w czasie. FFT pokazuje, z jakich składowych częstotliwościowych ten sygnał się składa.

```
def widmo(x, fs):
    N = len(x)
    X = np.fft.rfft(x)
    f = np.fft.rfftfreq(N, 1/fs)
    A = 2*np.abs(X)/N
    A[0] = A[0]/2
    return f, A

f, A = widmo(AM, fs)
plt.plot(f/1000, A)
plt.xlim(15, 25)
plt.xlabel('częstotliwość [kHz]')
plt.ylabel('amplituda')
plt.title('Widmo AM')
plt.grid()
plt.show()
```



Rys. 5. Widmo AM dla f_c = 20 kHz i f_m = 1 kHz.

Najważniejszy wynik: Powinny pojawić się trzy główne linie: 19 kHz, 20 kHz i 21 kHz. 20 kHz to nośna, a 19 i 21 kHz to dolna i górna wstęga boczna: f_c−f_m oraz f_c+f_m.

Dla μ = 0,7 i amplitudy nośnej 1, amplituda każdej wstęgi bocznej powinna być w przybliżeniu μ/2 = 0,35. Jeżeli częstotliwość modulująca zostanie zwiększona do 2 kHz, wstęgi przesuną się do 18 i 22 kHz.

Jeżeli widmo jest „rozlane”: Może to wynikać z tego, że obserwowany przedział czasu nie zawiera całkowitej liczby okresów sygnału. To zjawisko nazywa się przeciekiem widmowym. W dalszych kursach ogranicza się je m.in. przez okna czasowe, np. Hann.

Zadanie

1. Ustaw f_m = 2 kHz. Przewidź położenie wstęg przed uruchomieniem programu.
2. Ustaw μ = 0,25 i μ = 1,00. Sprawdź, jak zmieniają się amplitudy wstęg, a jak częstotliwości.
3. Zapisz w sprawozdaniu wartości trzech największych składowych widma.

5. Moduł D — modulacja częstotliwości FM

W FM amplituda nośnej pozostaje stała. Informacja jest zapisana w chwilowej częstotliwości. Dla sinusoidalnej modulacji użyjemy:

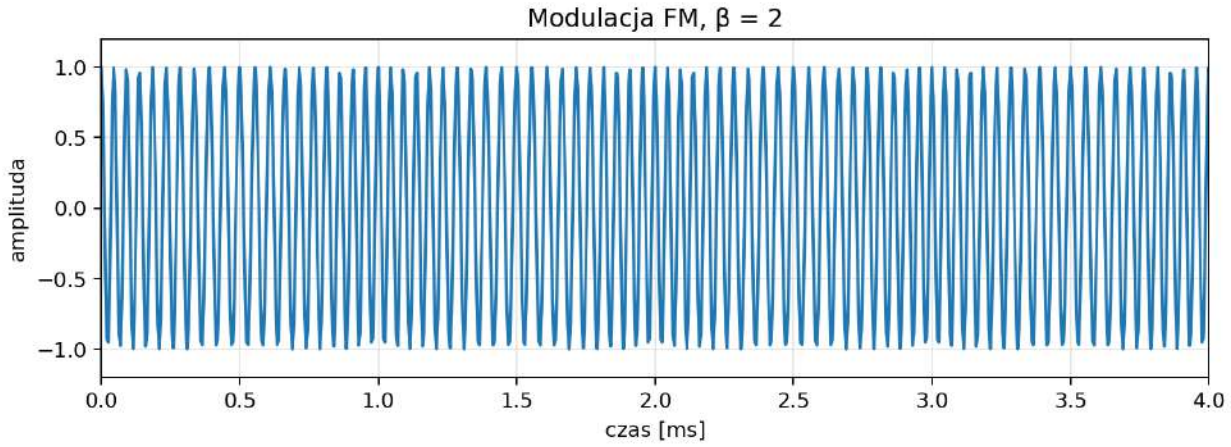
$$s_{FM}(t) = \cos[2\pi f_c t + \beta \sin(2\pi f_m t)]$$

β jest indeksem modulacji FM. Dla tonu sinusoidalnego $\beta = \Delta f / f_m$, gdzie Δf to maksymalna dewiacja częstotliwości.

Krok D1. Wygeneruj FM

```
beta = 2
FM = np.cos(2*np.pi*fc*t + beta*np.sin(2*np.pi*fm*t))

plt.plot(t*1000, FM)
plt.xlim(0, 4)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('FM, beta = 2')
plt.grid()
plt.show()
```



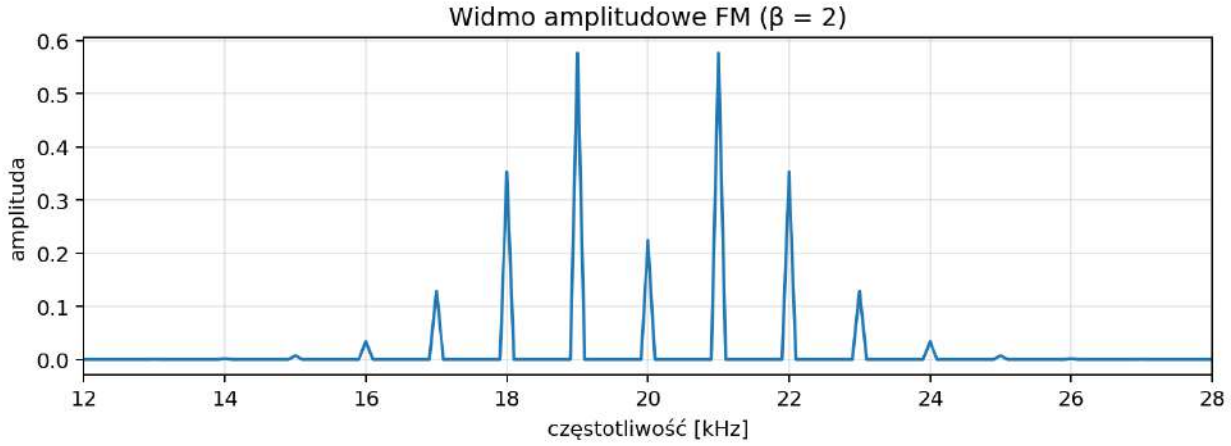
Rys. 6. FM: zmienia się lokalna gęstość okresów, amplituda pozostaje stała.

Jak czytać wykres: Tam, gdzie okresy są „ściśnięte", częstotliwość chwilowa jest większa; tam, gdzie są „rozciągnięte" — mniejsza. Amplituda nadal pozostaje około ±1.

Dla β = 2 i f_m = 1 kHz otrzymujemy Δf = 2 kHz. Częstotliwość chwilowa zmienia się więc w przybliżeniu od 18 do 22 kHz.

Krok D2. Widmo FM

```
f, A = widmo(FM, fs)
plt.plot(f/1000, A)
plt.xlim(12, 28)
plt.xlabel('częstotliwość [kHz]')
plt.ylabel('amplituda')
plt.title('Widmo FM')
plt.grid()
plt.show()
```



Rys. 7. Widmo FM ma wiele wstęg oddalonych od siebie o f_m.

Co oznacza wynik: W FM nie ma tylko dwóch wstęg bocznych. Pojawiają się składowe f_c ± n f_m. Im większe β, tym więcej istotnych wstęg i tym większe pasmo sygnału.

Przybliżoną szerokość pasma można oszacować regułą Carsona: B ≈ 2(Δf + f_m). Dla β = 2 i f_m = 1 kHz: Δf = 2 kHz, więc B ≈ 6 kHz. Jest to oszacowanie, nie ścisła granica widma.

6. Moduł E — przygotowanie informacji cyfrowej

Teraz zamiast sinusoidy będziemy transmitować prostą informację tekstową. Każdy znak zamienimy na 8-bitowy kod ASCII/Unicode zgodny z wartościami ASCII dla liter łacińskich.

```
text = 'UWM'
bits = ''.join(format(ord(ch), '08b') for ch in text)
print(bits)
```

Oczekiwany wynik: Dla napisu UWM program powinien zwrócić: 010101010101011101001101. Jest to 24-bitowy ciąg: po 8 bitów dla każdej z trzech liter.

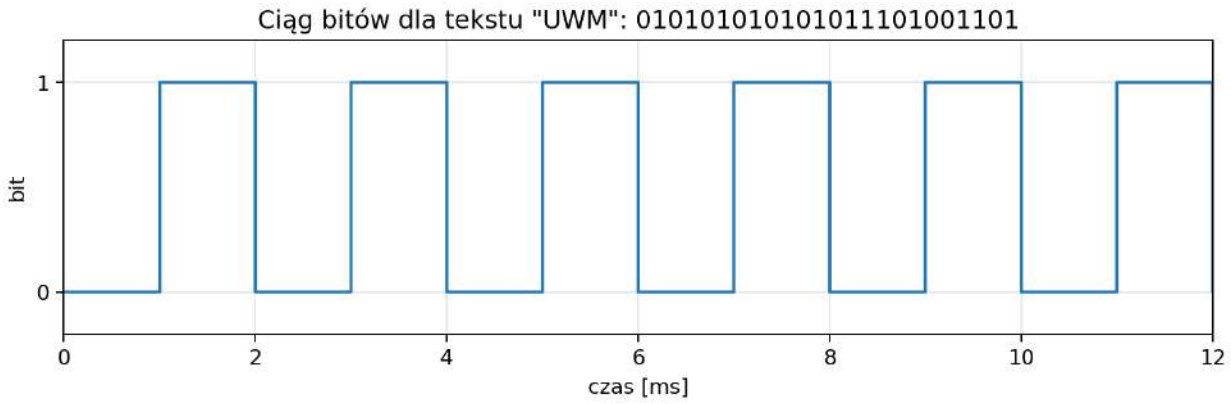
Znak	Kod dziesiętny	Kod 8-bitowy
U	85	01010101
W	87	01010111
M	77	01001101

Krok E1. Zamień tekst na przebieg bitowy

Przyjmij szybkość transmisji R_b = 1000 bit/s. Jeden bit trwa wtedy T_b = 1 ms. Przy f_s = 200 kHz na jeden bit przypada 200 próbek.

```
bit_values = np.array([int(b) for b in bits])
Rb = 1000
samples_per_bit = int(fs/Rb)
bit_wave = np.repeat(bit_values, samples_per_bit)
t_bits = np.arange(len(bit_wave))/fs

plt.step(t_bits*1000, bit_wave, where='post')
plt.xlim(0, 12)
plt.ylim(-0.2, 1.2)
plt.yticks([0, 1])
plt.xlabel('czas [ms]')
plt.ylabel('bit')
plt.grid()
plt.show()
```



Rys. 8. Pierwsze bity wiadomości „UWM”. Każdy bit trwa 1 ms.

Kontrola: 24 bity przy 1000 bit/s powinny trwać 24 ms. Jeżeli przebieg trwa np. 2,4 ms, prawdopodobnie pomylono bity/s z kbit/s albo czas podano w złych jednostkach.

7. Moduł F — ASK: cyfrowa modulacja amplitudy

W ASK wartość bitu wybiera amplitudę nośnej. Przyjmiemy A0 = 0,2 dla bitu 0 i A1 = 1,0 dla bitu 1. W odmianie OOK można przyjąć A0 = 0, czyli całkowicie wyłączać nośną dla zera.

```
A0 = 0.2
A1 = 1.0
amplitude = np.where(bit_wave == 1, A1, A0)
ASK = amplitude * np.cos(2*np.pi*fc*t_bits)

plt.plot(t_bits*1000, ASK)
plt.xlim(0, 8)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('ASK')
plt.grid()
plt.show()
```

Rys. 9. ASK — amplituda wskazuje wartość bitu.

Jak rozpoznać bity: Odcinki o dużej amplitudzie odpowiadają jedynkom, a o małej — zerom. Częstotliwość nośnej pozostaje 20 kHz. Jeżeli amplituda nie zmienia się pomiędzy bitami, sprawdź czy użyto bit_wave, a nie pojedynczej wartości bits.

Eksperyment

4. Zmień A0 na 0.0. Otrzymasz OOK (On-Off Keying).
5. Zmień R_b z 1 kbit/s na 2 kbit/s. Zwróć uwagę, że bity trwają krócej i widmo staje się szersze.

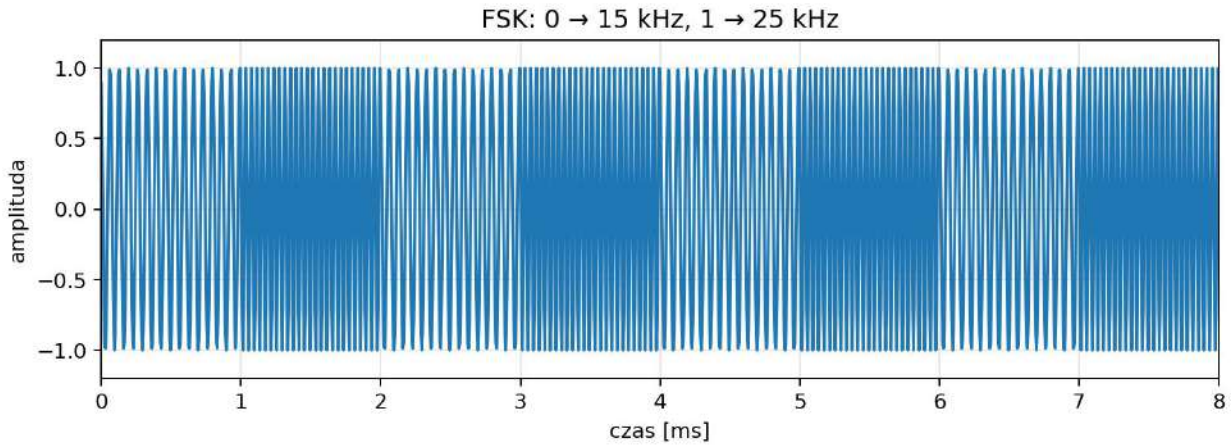
8. Moduł G — FSK: cyfrowa modulacja częstotliwości

W FSK bit 0 i bit 1 są reprezentowane przez dwie różne częstotliwości. Przyjmujemy 15 kHz dla 0 oraz 25 kHz dla 1. Amplituda pozostaje stała.

```
f0 = 15_000
f1 = 25_000
instantaneous_f = np.where(bit_wave == 1, f1, f0)

# całkowanie częstotliwości daje ciągłą fazę
phase = 2*np.pi*np.cumsum(instantaneous_f)/fs
FSK = np.cos(phase)

plt.plot(t_bits*1000, FSK)
plt.xlim(0, 8)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('FSK')
plt.grid()
plt.show()
```



Rys. 10. FSK — dla bitów 1 oscylacje są wyraźnie gęstsze niż dla bitów 0.

Jak interpretować wynik: W każdym przedziale 1 ms można policzyć okresy: przy 15 kHz powinno być około 15 okresów, a przy 25 kHz około 25 okresów. To daje prostą kontrolę, czy mapowanie 0/1 jest poprawne.

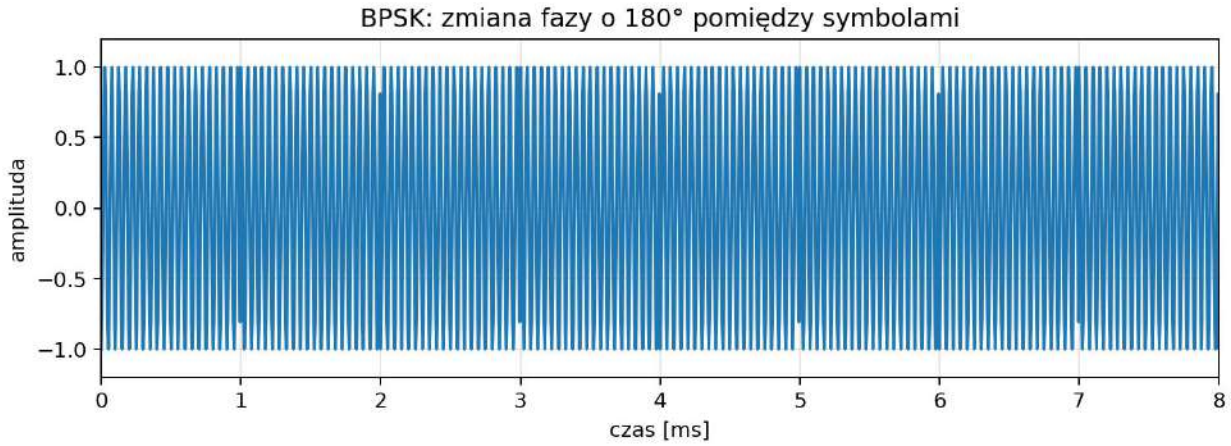
Dlaczego używamy cumsum: Gdyby dla każdego bitu uruchamiać nową sinusoidę od fazy 0, na granicach bitów mogłyby pojawić się sztuczne skoki. Sumowanie częstotliwości tworzy ciągłą fazę i bardziej realistyczny przebieg.

9. Moduł H — BPSK: cyfrowa modulacja fazy

W BPSK amplituda i częstotliwość nośnej pozostają takie same. Bit wybiera jedną z dwóch faz różniących się o 180°. Można to zrealizować przez mnożenie nośnej przez +1 lub −1.

```
symbols = 2*bit_wave - 1      # 0 -> -1, 1 -> +1
BPSK = symbols * np.cos(2*np.pi*fc*t_bits)

plt.plot(t_bits*1000, BPSK)
plt.xlim(0, 8)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('BPSK')
plt.grid()
plt.show()
```



Rys. 11. BPSK — przy zmianie symbolu faza nośnej może odwrócić się o 180°.

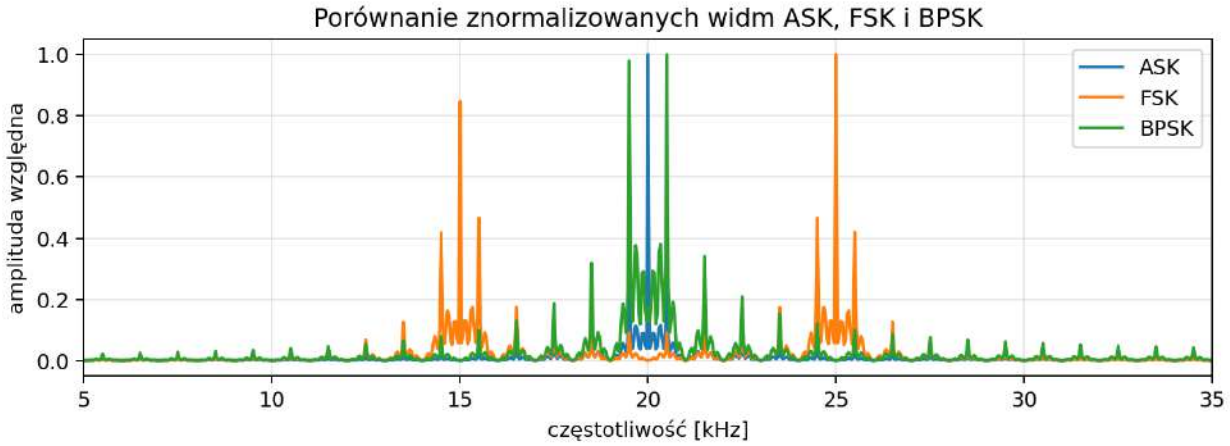
Najczęstsza pułapka: Zmiana fazy nie zawsze jest łatwa do zauważenia na długim wykresie. Powiększenie granicy pomiędzy dwoma różnymi bitami pokaże, że w jednym przypadku maksimum nośnej odpowiada minimum w drugim.

Jeśli dwa kolejne bity są takie same, na granicy nie będzie zmiany fazy. To jest prawidłowy wynik — BPSK zmienia fazę tylko wtedy, gdy zmienia się przypisany symbol.

10. Moduł I — porównanie widm ASK, FSK i BPSK

```
for signal, name in [(ASK, 'ASK'), (FSK, 'FSK'), (BPSK, 'BPSK')]:
    f, A = widmo(signal, fs)
    A = A / np.max(A)          # normalizacja tylko do porównania kształtu
    plt.plot(f/1000, A, label=name)

plt.xlim(5, 35)
plt.xlabel('częstotliwość [kHz]')
plt.ylabel('amplituda względna')
plt.legend()
plt.grid()
plt.show()
```



Rys. 12. Przykładowe widma krótkiej wiadomości UWM.

Jak rozumieć ten wykres: FSK tworzy energię wokół dwóch częstotliwości (15 i 25 kHz). ASK i BPSK są skupione wokół nośnej 20 kHz, ale szybkie zmiany bitów tworzą wstęgi boczne. Krótka, nieregularna sekwencja nie daje idealnych „linii", lecz rozciągnięte widmo.

Zwiększenie szybkości bitowej skraca czas pojedynczego symbolu. Krótsze impulsy mają szersze widmo, dlatego większa szybkość transmisji zwykle wymaga większego pasma.

Tabela obserwacji do uzupełnienia

Modulacja	Parametr zmieniany	Co widzisz w czasie?	Co widzisz w widmie?
ASK	amplituda		
FSK	częstotliwość		
BPSK	faza		

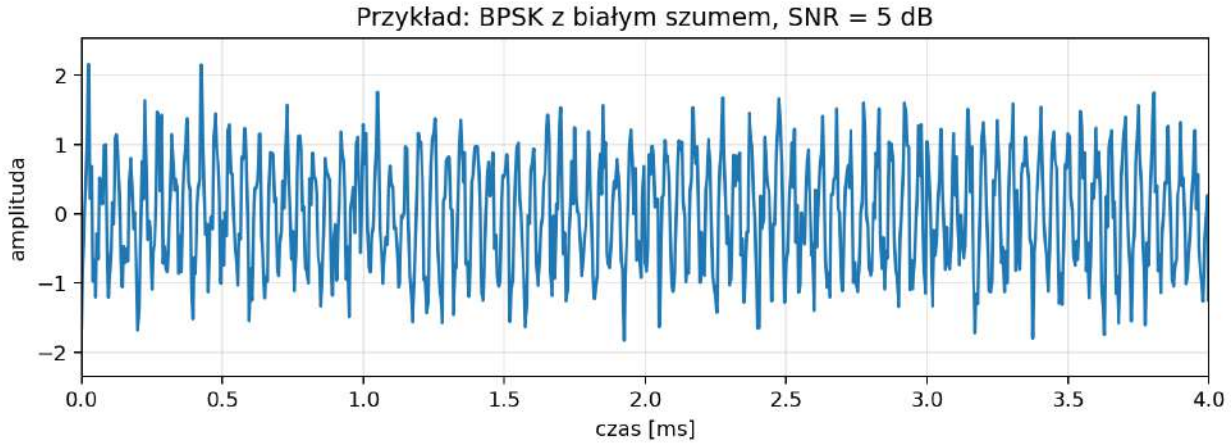
11. Moduł J — wpływ szumu i pojęcie SNR

Sygnały rzeczywiste są zakłócane. Dodamy biały szum gaussowski i ustawimy jego moc tak, aby otrzymać zadany SNR (signal-to-noise ratio).

$$\mathrm{SNR[dB]} = 10\log_{10}(P_s / P_n)$$

```
def add_noise(x, snr_db, seed=1234):
    rng = np.random.default_rng(seed)
    signal_power = np.mean(x**2)
    noise_power = signal_power / (10**(snr_db/10))
    noise = rng.normal(0, np.sqrt(noise_power), len(x))
    return x + noise

BPSK_noisy = add_noise(BPSK, 5)
plt.plot(t_bits*1000, BPSK_noisy)
plt.xlim(0, 4)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('BPSK + szum, SNR = 5 dB')
plt.grid()
plt.show()
```



Rys. 13. Przy SNR = 5 dB przebieg jest wyraźnie zaburzony, ale struktura nośnej nadal bywa rozpoznawalna.

SNR	Stosunek mocy sygnał/szum	Znaczenie jakościowe
20 dB	100 : 1	szum zwykle niewielki
10 dB	10 : 1	szum wyraźny
5 dB	≈ 3,16 : 1	trudniejsze rozpoznawanie
0 dB	1 : 1	moc sygnału i szumu równa
−5 dB	≈ 0,316 : 1	szum ma większą moc niż sygnał

Ważne: SNR dotyczy mocy, a nie bezpośrednio amplitudy. 20 dB oznacza 100-krotnie większą moc sygnału od mocy szumu, nie 20-krotnie większą amplitudę.

Eksperyment

6. Wykonaj wykresy dla SNR = 20 dB, 10 dB, 5 dB, 0 dB i −5 dB.
7. Oceń, przy jakim SNR przestajesz rozpoznawać strukturę ASK, FSK i BPSK na wykresie czasowym.
8. Porównaj widma przed i po dodaniu szumu. Zwróć uwagę na wzrost „tła” widmowego.

12. Zadanie dodatkowe — prosty odbiornik FSK

Aby pokazać, że modulacja faktycznie przenosi informację, można spróbować odzyskać bity z FSK. W każdym przedziale bitowym mierzymy, do której z dwóch częstotliwości sygnał jest bardziej podobny.

```
decoded = []
for k in range(len(bits)):
    start = k*samples_per_bit
    stop  = (k+1)*samples_per_bit
    segment = FSK[start:stop]
    tt = np.arange(samples_per_bit)/fs

    # zespolona korelacja - niewrażliwa na fazę początkową
    score0 = abs(np.sum(segment * np.exp(-1j*2*np.pi*f0*tt)))
    score1 = abs(np.sum(segment * np.exp(-1j*2*np.pi*f1*tt)))
    decoded.append('1' if score1 > score0 else '0')

decoded_bits = ''.join(decoded)
print(decoded_bits)

chars = [chr(int(decoded_bits[i:i+8], 2))
        for i in range(0, len(decoded_bits), 8)]
print(''.join(chars))
```

Oczekiwany wynik: Dla sygnału bez szumu powinieneś odzyskać ten sam ciąg bitów, a następnie napis UWM. Jeśli tak się nie dzieje, sprawdź R_b, samples_per_bit oraz wartości f0 i f1.

Następnie spróbuj zastosować odbiornik do sygnału FSK z szumem. W ten sposób można wyznaczyć, przy jakim SNR pojawiają się pierwsze błędy bitowe.

Co oznacza błąd dekodowania?

Nie musi oznaczać błędu programu. Przy silnym szumie energia w paśmie niewłaściwej częstotliwości może stać się chwilowo większa niż energia właściwego tonu. Jest to programowy odpowiednik rzeczywistego błędu transmisji.

13. Opracowanie wyników i sprawozdanie

Sprawozdanie powinno prowadzić od parametrów wejściowych do interpretacji. Nie wystarczy wkleić wykresów — przy każdym wykresie należy napisać, co z niego wynika.

Minimalna zawartość sprawozdania

- Parametry symulacji: f_s, f_m, f_c, czas obserwacji oraz R_b.
- Wykres sygnału modulującego i nośnej z poprawnie opisanymi osiami.
- AM dla co najmniej trzech wartości μ, w tym jednej > 1, oraz komentarz dotyczący przemodulowania.
- Widmo AM ze wskazaniem f_c, $f_c - f_m$ i $f_c + f_m$.
- FM dla co najmniej dwóch wartości β oraz komentarz o dewiacji częstotliwości i szerokości widma.
- Zakodowana informacja tekstowa, przebieg bitowy i łączny czas transmisji.
- Wykresy ASK, FSK i BPSK oraz krótka interpretacja sposobu kodowania bitu 0 i 1.
- Porównanie widm ASK, FSK i BPSK.
- Przykład sygnału z szumem dla co najmniej dwóch wartości SNR.
- Wnioski: 5–8 zdań wynikających z własnych obserwacji.

Pytania kontrolne

1. Dlaczego do symulacji nie użyliśmy nośnej 100 MHz? Czy z punktu widzenia równań modulacji zmieniłoby to zasadę działania?
2. Co dzieje się z AM przy μ > 1?
3. Dlaczego w widmie AM pojawiają się składowe $f_c \pm f_m$?
4. Jaka jest różnica pomiędzy częstotliwością nośnej f_c a dewiacją Δf w FM?
5. Dlaczego widmo FM rozszerza się wraz ze wzrostem β?
6. Który parametr nośnej reprezentuje bit w ASK, FSK i BPSK?
7. Jak zmiana R_b wpływa na czas trwania bitu i szerokość widma?
8. Co oznacza SNR = 0 dB?
9. Dlaczego krótka sekwencja danych cyfrowych daje widmo ciągłe/lobowe zamiast kilku idealnych linii?
10. Która z badanych modulacji jest najłatwiejsza do rozpoznania na wykresie czasowym i dlaczego? Odpowiedź uzasadnij na podstawie wyników.

Wnioski, które student powinien umieć sformułować

Po wykonaniu ćwiczenia student powinien rozumieć, że modulacja nie tworzy „nowego rodzaju fali”, lecz steruje parametrem nośnej zgodnie z informacją. Powinien też umieć połączyć obraz w dziedzinie czasu z widmem częstotliwościowym i rozumieć, że szybsze zmiany informacji oraz silniejsza modulacja zazwyczaj zwiększają zajmowane pasmo. W części cyfrowej powinien potrafić wskazać, jak bit 0 i 1 są reprezentowane w ASK, FSK i BPSK. Dodanie szumu ma pokazać, dlaczego odbiornik nie odtwarza danych idealnie i dlaczego jakość łącza opisuje się m.in. przez SNR i błędy bitowe.

Załącznik — kompletny program bazowy

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

# --- parametry wspólne ---
fs = 200_000
fm = 1_000
fc = 20_000
T = 0.010
t = np.arange(0, T, 1/fs)
message = np.sin(2*np.pi*fm*t)
carrier = np.cos(2*np.pi*fc*t)

# --- AM ---
mu = 0.7
AM = (1 + mu*message) * carrier

# --- FM ---
beta = 2
FM = np.cos(2*np.pi*fc*t + beta*np.sin(2*np.pi*fm*t))

# --- funkcja FFT ---
def widmo(x, fs):
    N = len(x)
    X = np.fft.rfft(x)
    f = np.fft.rfftfreq(N, 1/fs)
    A = 2*np.abs(X)/N
    A[0] = A[0]/2
    return f, A

# --- wiadomość cyfrowa ---
text = 'UWM'
bits = ''.join(format(ord(ch), '08b') for ch in text)
bit_values = np.array([int(b) for b in bits])
Rb = 1000
samples_per_bit = int(fs/Rb)
bit_wave = np.repeat(bit_values, samples_per_bit)
t_bits = np.arange(len(bit_wave))/fs

# --- ASK ---
A0, A1 = 0.2, 1.0
ASK = np.where(bit_wave == 1, A1, A0) * np.cos(2*np.pi*fc*t_bits)

# --- FSK ---
f0, f1 = 15_000, 25_000
inst_f = np.where(bit_wave == 1, f1, f0)
phase = 2*np.pi*np.cumsum(inst_f)/fs
FSK = np.cos(phase)

# --- BPSK ---
symbols = 2*bit_wave - 1
BPSK = symbols * np.cos(2*np.pi*fc*t_bits)

# przykład wykresu
plt.plot(t_bits*1000, FSK)
plt.xlim(0, 8)
plt.xlabel('czas [ms]')
plt.ylabel('amplituda')
plt.title('FSK')
```

```
plt.grid()
plt.show()

# przykład widma
f, A = widmo(AM, fs)
plt.plot(f/1000, A)
plt.xlim(15, 25)
plt.xlabel('częstotliwość [kHz]')
plt.ylabel('amplituda')
plt.title('Widmo AM')
plt.grid()
plt.show()
```

Sposób pracy: Nie uruchamiaj od razu całego załącznika i nie traktuj go jako „gotowego rozwiązania”. Ćwiczenie jest skonstruowane tak, aby po każdym kroku obejrzeć wykres, przewidzieć wynik kolejnej zmiany i dopiero potem przejść dalej.