Tomasz P. Zieliński

Cyfrowe przetwarzanie sygnałów

Instrukcje laboratoryjne - Lato 2025

March 3, 2025

Instytut Telekomunikacji, AGH Kraków

Organizacja laboratorium

Osoby prowadzące

1. prof. dr hab. inż. Tomasz Zieliński (tzielin@agh.edu.pl)
2. inne (jeśli tak, to wówczas one ustalają szczegółowe zasady organizacji i zaliczenia laboratorium)

Obecność na laboratorium

1. Laboratoria odbywają się co tydzień.
2. Obecność studentów jest obowiązkowa. Dotyczy to także studentów mających ITS.
3. Laboratoriów nie odrabia się.
4. Student może mieć tylko jedną nieusprawiedliwioną nieobecność bez konsekwencji.
5. Za każdą dodatkową nieobecność ma automatycznie obniżaną ocenę o 0.5 stopnia.
6. Łącznie 3 albo 4 nieusprawiedliwione nieobecności powodują, że student nie ma możliwości poprawy końcowej oceny laboratoryjnej podczas sesji.
7. Więcej niż 4 nieusprawiedliwione nieobecności skutkują niezaliczeniem przedmiotu - otrzymaniem oceny 2.0 z laboratorium i egzaminu (jeśli jest w planach).

Wystawianie oceny laboratoryjnej

1. Każde z 12 laboratoriów podstawowych o numerach 1-12 składa się z zadań (problemów do rozwiązania), które są punktowane: 1(*), 2(**) lub 3(***) punkty. Aby zaliczyć pojedyncze laboratorium na ocenę 5.0 student musi uzyskać 3 punkty. Ale może zrobić więcej zadań i zdobyć więcej punktów, maksymalnie 5 z jednego laboratorium.
2. Wszystkie punkty uzyskane przez studenta w trakcie semestru są dodawane.
3. Każdy student jest losowo pytany podczas K różnych laboratoriów, wybranych przez prowadzącego ($6 \leq K \leq 12$). Dlatego może on uzyskać od 0 do $K*3$ punktów (bez punktów dodatkowych, z nimi — więcej).
4. Punkty te są przeliczane na ocenę końcową z laboratorium zgodnie z regulaminem studiów, przyjmując $K*3$ punktów jako MAX=100%. Zakładając, że student był pytany $K = 10$ razy, to mógł on zdobyć MAX=10*3=30 punktów (bez punktów dodatkowych) i wówczas obowiązuje następująca skala ocen:

Procenty [%]	Suma pkt [MAX=30]	Ocena (nazwa)	Ocena (liczba)
≥110%	[33-48]	bardzo dobry EGZ	5.0 EGZAMIN
91-100%	(27-30]	bardzo dobry	5.0
81-90%	(24-27]	plus dobry	4.5
71-80%	(21-24]	dobry	4.0
61-70%	(18-21]	plus dostateczny	3.5
50-60%	[15-18]	dostateczny	3.0
<50%	[0-15)	niedstateczny	2.0

5. UWAGA DODATKOWA: zamiast punktów za wykonanie poszczególnych zadań student może otrzymywać za przygotowanie się do laboratorium jedną, zbiorczą ocenę (w skali 2.0 - 5.0) - w takim przypadku ocena końcowa jest średenią z uzyskanych ocen cząstkowych (czyli "jak w szkole"). Prowadzący i studenci na pierwszych zajęciach razem wybiorą obowiązującą metodę oceny.

Organizacja wykładów i laoratorium

1. Problematyka każdego laboratorium jest przedstawiana na wykładzie tydzień wcześniej wraz z przykładowymi programami w języku Matlab. Programy są pisane w trakcie wykładu i po nim są udostępniane studentom. Studenci zadają pytania.
2. Studenci wstępnie rozwiązują zadania z instrukcji laboratoryjnej w domu, korzystając z programów z wykładu. Wybierają zadania o różnym stopniu trudności (*)(**)(***), kierując się własnymi zainteresowaniami.
3. Dodatkowe programy pomocnicze do nauki, z komentarzami w języku polskim i angielskim, są dostępne w Internecie:

- *http://www.kmet.agh.edu.pl/dydaktyka/wydawnictwa/* - szukaj programów do książek [39] [40],
- *http://teledsp.kt.agh.edu.pl/* - programy do książki [42],
- *http://kt.agh.edu.pl/~tzielin/books/DSPforTelecomm/* - programs for the book [41]

4. W trakcie laboratorium losowo wybrani studenci są pytani o: a) szczegóły napisanych programów, stanowiących rozwiązanie zadań (co? jak? dlaczego?), oraz b) o powiązaną z nim teorię, przedstawioną na wykładzie. Pytania mogą być inne dla każdego studenta.
5. Jeśli laboratorium jest przeprowadzane zdalnie, studenci są informowani o konkretnej godzinie odpowiedzi na platformie MS Teams.
6. Każdy odpowiadający, losowo wybrany student "gra" o 3 punkty. Maksymalnie może uzyskać 5 punktów, jeśli wykonał więcej zadań. Sprawozdawane są tylko zadania z aktualnego ćwiczenia, a nie wcześniejszych.

Poprawa laboratorium

1. W przypadku otrzymania oceny niedostatecznej z laboratorium (poniżej 50% maksymalnej liczby punktów) w zwykłym terminie, tzn. w trakcie semestru, student ma prawo do dwóch terminów dodatkowych podczas sesji, ale tylko wówczas, gdy podczas semestru zdodył więcej niż 25% maksymalnej liczby punktów.
2. Podczas terminów dodatkowych student prezentuje rozwiązania wybranych przez siebie zadań z laboratoriów 1-12, których nie wykonywał w trakcie semestru LUB zadań z dodatkowego laboratorium zaliczeniowego numer 13 (mowa, audio, obrazy). Otrzymane punkty są dodawane do sumy wszystkich punktów. Aby otrzymać ocenę pozytywną, suma zdobytych punktów musi być conajmniej równa 50% maksimum, określonego przez liczbę zdawanych laboratoriów w trakcie semestru.
3. Maksymalna ocena uzyskana w tym trybie poprawkowym to dostatecznie (3.0).

EGZAMIN I OCENA KOŃCOWA

1. Jeśli przedmiot kończy się egzaminem, to egzamin ma jedną z trzech form (o wyborze jednej z nich prowadzący powinien poinformować na poczatku kursu):

   - egzamin pisemny z materiału przedstawionego na wykładzie (2 proste pytania z każdego z 12 wykładów, razem 24 pytania);
   - prezentacja trzech wybranych programów z różnych laboratoriów - innych dla każdego studenta;
   - egzamin może mieć formę pytań bezpośrednio zadawanych studentom podczas laboratorium w przypadka kiedy osoba odpowiedzialna za przedmiot, równocześnie będąca egzaminatorem, sama prowadzi laboratorium.

2. Studenci, którzy zebrali $\geq$ 110% maksymalnej liczby punktów z laboratorium (np. 33 punkty dla MAX=30), automatycznie otrzymują ocenę 5.0 z egzaminu.
3. Studenci, którzy otrzymali oceny 4.5 albo 5.0 z laboratorium oraz byli obecni na 12-stu wykładach z 15-stu, lub więcej, otrzymują automatycznie ocenę 4.5 lub 5.0, odpowiednio, jako ocenę egzaminacyjną. Studenci, którzy chcą skorzystać z takiej możliwości, powinni to zgłosić na pierwszym wykładzie.
4. Po losowo wybranych kilku wykładach, wykładowca zada jedno pytanie z tematyki wykładu. Pierwszy student, który odpowie na nie poprawnie, wygra w ten sposób przepisanie oceny laboratoryjnej jako egzaminacyjnej, niezależnie wysokości tej oceny.
5. Ocena końcowa, jeśli jest wystawiana, to średnia arytmetyczna oceny z laboratorium i egzaminu, jeśli jest egzamin. Kiedy go nie ma, jako ocena końcowa jest przepisywana ocena laboratoryjna.
6. Studenci przepisujący ocenę powinni zgłosić się na początku zajęć w konkretnym semestrze.

Inne

1. Wszystkie problemy dotyczące laboratorium powinny być najpierw zgłoszone osobom prowadzącym laboratorium, a w drugiej kolejności osobie odpowiedzialnej za przedmiot, czyli wykładowcy/egzaminatorowi. Najlepiej pocztą elektroniczną. E-maile wysyłane przez studentów powinny być mailami uczelnianymi.
2. Pytania dotyczące wykładu, egzaminu oraz wystawiania oceny końcowej należy zadawać wykładowcy.
3. Kwestie nieuregulowane powyżej, rozstrzyga Regulamin Studiów.

Kraków, Luty 2025 *Tomasz Zieliński*

Contents

Laboratorium 1
Sygnały: Akwizycja, klasyfikacja, próbkowanie

Streszczenie Na tym laboratorium poznamy czym jest sygnał i jakie typy sygnałów się rozróżnia. Wygenerujemy prosty sygnał zdeterminowany (sinusoidę) oraz losowy (szum). Za pomocą karty dźwiękowej nagramy proste sygnały akustyczne, narysujemy je i zagramy. Napiszemy pierwsze programy w języku Matlab.

TEMAT #1: Sinusoida jako sygnał deterministyczny. Sygnały deterministyczne *proste* mają kształt zdefiniowany za pomocą znanych funkcji *prostych*: sinusoidy, eksponenty, gaussoidy, ... lub sum oraz iloczynów (próbka-przez-próbkę), znanych funkcji, np. sinusioda razy eksponenta. Sygnały deterministyczne *złożone* są sumami sygnałów *prostych*. Najpopularniejszym sygnałem deterministycznym jest sinusoida.
Przykładowo, sygnał sinusoidalny, mający amplitudę (*siłę*) $A_1 = 0.5$, częstotliwość (liczbę powtórzeń na sekundę) $f_1 = 10$ Hz, oraz przesunięcie fazowe (kątowe) $\phi_1 = \frac{\pi}{4}$ radianów, jest opisany równaniem:

$$x_1(t) = 0.5 \cdot \sin\left(2\pi \cdot 10 \cdot t + \frac{\pi}{4}\right), \tag{1.1}$$

Po zastosowaniu podstawień: $dt = \frac{1}{f_{pr}}$ (okres próbkowania, odległość pomiedzy dwiema próbkami sygnału, odwrotność częstotliwości próbkowania f_{pr}, czyli liczby próbek na sekundę), $t = n \cdot dt$ (chwile próbkowania), $n = 0, 1, 2, \ldots$ (numery próbek), otrzymujemy:

$$x_1(n) = 0.5 \cdot \sin\left(2\pi \frac{10}{f_{pr}} n + \frac{\pi}{4}\right), \quad n = 0, 1, 2, \ldots \tag{1.2}$$

Listing 1.1: Generacja sinusoidy w Matlabie

```
% cps_01_sinus.m
  clear all; close all;

  fpr=1000; Nx=1000;            % parametry: czestotliwosc probkowania, liczba probek
  dt = 1/fpr;                   % okres probkowania
  n = 0 : Nx-1;                 % numery probek
  t = dt*n;                     % chwile probkowania
  A1=0.5; f1=10; p1=pi/4;       % sinusoida: amplituda, czestotliwosc, faza
  x1 = A1*sin(2*pi*f1    *t+p1); % pierwszy skladnik sygnalu
% x1 = A1*sin(2*pi*f1/fpr*n+p1); % pierwszy skladnik inaczej zapisany
% x2 = ?;                       % drugi skladnik
% x3 = ?;                       % trzeci skladnik
  x = x1;                       % wybor skladowych: x = x1, x1 + 0.123*x2 + 0.456*x3
  plot(t,x,'o-'); grid; title('Sygnal x(t)'); xlabel('Czas [s]'); ylabel('Amplituda');
```

Problem 1.1 (* Generacja sinusoidy. Parametry sinusoidy. Słuchanie sumy sinusoid.). Uruchom Matlab-a, otwórz edytor, skopiuj kod programu z listingu 1.1 oraz zapisz go na dysk jako zbiór o nazwie `cps_01_sinus.m` (znak "minus" jest zabroniony w nazwie zbioru). Uruchom program, obejrzyj rysunek. Program generuje $N_x = 1000$ próbek sinusoidy, pobranych z sygnału z częstotliwością próbkowania $f_{pr} = 1000$ Hz, czyli 1 sekundę sinusa. Wygeneruj sinusoidy o różnych wartościach amplitudy `A1`, częstotliwości `f1` oraz przesunięcia fazowego `p1`. Zmień warość jednego parametru oraz na jednym rysunku narysuj dwa sygnały, przed i po zmianie wartości parametru. Wygeneruj trzy sinusoidy `x1,x2,x3` mające wszystkie parametry inne, dodaj je `x=x1+x2+x3;` oraz pokaż wynik. Ustaw `fpr=8000; Nx=3*fpr;`. Wygeneruj sumę trzech sinusoid o różnych częstotliwościach, większych niż 100 Hz i mniejszych niż 4000 Hz. Narysuj sygnał, posłuchaj go: `sound(x,fpr);`

TEMAT #2: Twierdzenie o próbkowaniu. Każdy sygnał o wartościach rzeczywistych musi być próbkowany tak, aby występująca w nim sinusoida o największej częstotliwości miała wiecej niż dwie próbki na okres. Tzn. częstotliwość próbkowania musi spełniać warunek:

$$f_{pr} > 2 \cdot f_{max}. \tag{1.3}$$

Uzasadnienie na przykładzie. Zwróćmy uwagę na równość, będącą uogólnieniem równania (1.2) dla sygnału o częstotliwości $f_1 = k \cdot f_{pr} \pm f_x$:

$$x_1(n) = A_1 \sin\left(2\pi \frac{k \cdot f_{pr} \pm f_x}{f_{pr}} n + \phi_1\right) = A_1 \sin\left(2\pi kn \pm 2\pi \frac{f_x}{f_{pr}} n + \phi_1\right) = \pm A_1 \sin\left(2\pi \frac{f_x}{f_{pr}} n + \phi_1\right) \tag{1.4}$$

wskazującą, że po spróbkowaniu pewne sinusoidy są nierozróżnialne, a przecież tego nie chcemy. Przykładowo dla $f_{pr} = 1000$ Hz oraz $f_x = 100$ Hz, po spróbkowaniu tak samo wygladają sinusoidy o częstotliwościach: **100**, **1100**=1000+100, **2100**=2000+100, **3100**=3000+100,... herców. Sinusoidy o częstoliwościach **900**=1000-100, **1900**=2000-100, **2900**=3000-100, ... herców także są identyczne, ale mają znak minus. Dla kosinusa mamy: $\cos\left(2\pi \frac{k \cdot f_{pr} \pm f_x}{f_{pr}} n\right) = \cos\left(2\pi \frac{f_x}{f_{pr}} n\right)$, czyli wszystkie kosinusy o częstotliwościach $100, 1000 \pm 100, 2000 \pm 100, 3000 \pm 100, \ldots$ są po spróbkowaniu takie same. Aby nie mieć dylematu na jakiego sinusa/kosinusa patrzymy, 100 Hz czy 900, 1100, 1900, 2100 Hz, ..., częstotliwość tego kosinusa musi być mniejsza od 500 Hz, czyli od połowy częstotliwosci próbkowania. Cbdo.
WAŻNE! Sygnał o wartościach zespolonych musi być tak próbkowany, aby jego składowa $e^{j2\pi f_{max} t}$ o największej częstotliwości f_{max} miała wiecej niż jedną próbkę na okres.

Problem 1.2 (* Sprawdzanie twierdzenia o próbkowaniu). Zmodyfikuj program 1.1. Wygeneruj trzy sinusoidy `x1,x2,x3` o częstotliwościch `f1=1; f2=fpr+f1; f3=2*fpr+f1` — pozostałe parametry są takie same. Narysuj je na jednym rysunku. Wytłumacz obserwowany efekt, korzystając z równania (1.4). Powtórz eksperyment dla częstotliwości `f1=1; f2=fpr-f1; f3=2*fpr-f1`. Wytłumacz. Ustaw `f1=5;` i powtórz ćwiczenie. Zmień sinusa na kosinusa i powtórz ćwiczenie dla `f1=5;`. Ustaw `fpr=8000; Nx=3*fpr;`. Wygeneruj trzy sinusoidy `x1,x2,x3` mające częstotliwości `f1=200; f2=fpr+f1; f3=2*fpr+f1`. Odsłuchaj je jedna po drugiej, użyj funkcji `sound(x1,fpr)`.
DLA AMBITNYCH. Co prawda nie wiemy jeszcze jak się oblicza *widmo częstotliwościowe* sygnału, czyli informację o tym "jakie częstotliwości zawiera sygnał", ale użyjmy funkcji Matlaba `pspectrum(x,fpr)` do wyznaczenia i narysowania widm sygnałów `x1, x2, x3`. Czy widma te one różne czy identyczne? Wytłumacz dlaczego.

TEMAT #3: Sygnał losowy, np. szum aparatury, kwantowania. Kształtu sygnału losowego, w przeciwieństwie do deterministycznego, nie można przewidzieć i opisać za pomocą konkretnej funkcji. To tak jak podczas rzucania kostką do gry: nie wiemy jaką sekwencję liczb otrzymamy. Sygnały/procesy losowe są charakteryzowane za pomocą funkcji gęstości rozkładu prawdopodobieństwa generowanych liczb, mówiącej nam o tym z jakim pradopodobieństwem sygnał przyjmuje konkretne wartości (równe 1/6 w przypadku rzutu kostką). Najczęstszy jest rozkład równomierny (przypadek kostki) i gaussowski (wynika z centralnego twierdzenia granicznego).

Listing 1.2: Generacja szumu w Matlabie

```
% cps_01_szum.m
clear all; close all;

Nx = 10000;
s1 = rand(1,Nx);    % s1 = 0.1*(2*(s1-0.5)); % rownomierny, skalowanie do [-0.1,0.1]
s2 = randn(1,Nx);   % s2 = 0.1*s2;           % gaussowski, skalowanie do std=0.1

figure;
```

```
subplot(221); plot(s1,'.-'); grid; title('Szum rownomierny [0,1]');
subplot(222); plot(s2,'.-'); grid; title('Szum gaussowski');
subplot(223); hist(s1,20); title('Histogram szumu rownomiernego');
subplot(224); hist(s2,20); title('Histogram szumu gaussowskiego');
```

Problem 1.3 (* Sygnał losowy, np. szum). Wykonaj kod programu z listingu 1.2, który generuje dwa rodzaje szumu: równomierny w przedziale [0,1] oraz gaussowski (wartość średnia = 0, odchylenie standardowe = 1). Obserwuj kształty obu sygnałów (używając *lupy*), znajdź wzrokowo wartości minimalne i maksymalne obu sygnałów oraz porównaj je z zakresami argumentu x, obserwowanymi na wykresach histogramów. Zaobserwuj różnicę kształtu obu histogramów, powiąż ją z wartościami przyjmowanymi przez poszczególne sygnały. Zobacz jak zmieniają się rysunki kiedy wartości szumu są przeskalowane (w tym celu usuń znak % komentarza w odpowiednim miejscu programu). Dodaj szum do sygnałów z problemu 1.1: mały (niewidoczny), średni (sinusoida jest zniekształcona, ale jeszcze ciągle widoczna) oraz bardzo duży (całkowicie *zasłaniający* sinusoidę). Wyznacz i narysuj histogram sygnału sinusoidalnego bez szumu oraz z szumem.

TEMAT #4: Nagrywanie i odtwarzanie sygnałów. Sygnały z czujników (sensorów) są wzmacniane i podawane na przetwornik analog-cyfra (A/C), który je próbkuje (dyskretyzacja argumentu funkcji, np. czasu co 1 milisekunda lub położenia co 1 milimetr) oraz kwantuje (dyskretyzacja wartości funkcji, np. temperatury co 0.1 stopnia Celsjusza). W przypadku sygnałów dźwiękowych czujnikiem jest mikrofon, w naszym przypadku podłączony do karty dźwiękowej komputera/telefonu. N-bitowy przetwornik A/C zwraca numer przedziału kwantowania, do której została przypisana konkretna próbka sygnału, z zakresu $[0, 2^N - 1]$ (liczba całkowita bez znaku, np. uint8 [0,255], uint16) albo $[-2^{\frac{N}{2}}, 2^{\frac{N}{2}} - 1]$ (całkowita ze znakiem, np. int8 [-128,127], int16). Wartości minimalne i maksymalne odpowiadają napięciom zakresowym, np. $[0, 5]$ V albo $[-5, 5]$ V.

Listing 1.3: Nagrywanie i odtwarzanie dźwięku w języku Matlab

```
% cps_01_audio.m
clear all; close all;

% Akwizycja sygnalu audio
fpr = 8000;       % czestotliwosc probkowania (probki na sekunde):
                  % 8000, 11025, 16000, 22050, 32000, 44100, 48000, 96000,
bits = 8;         % liczba bitow na probke: 8, 16, 24, 32
channels = 1;     % liczba kanalow: 1 albo 2 (mono/stereo)

recorder = audiorecorder(fpr, bits, channels);  % tworzenie obiektu
disp('Nacisnij klawisz i nagraj audio'); pause  % pauza przed nagraniem
record(recorder);                               % start nagrania
pause(2);                                       % nagranie 2 sekund
stop(recorder);                                 % stop nagrania
play(recorder);                                 % odsluch
audio = getaudiodata( recorder, 'single' );     % import danych

% Weryfikacja - odsluch, rysunek
sound(audio,fpr);         % odtworz nagrany dzwiek
x = audio; clear audio;   % skopiuj audio, wyzeruj audio
Nx = length(x);           % pobierz liczbe probek
n= 0:Nx-1;                % indeksy probek
dt = 1/fpr;               % oblicz okres probkowania sygnalu
t = dt*n;                 % oblicz chwile probkowania
figure; plot(x,'bo-'); xlabel('numer probki n'); title('x(n)'); grid;
figure; plot(t,x,'b-'); xlabel('t (s)'); title('x(t)'); grid; pause

% Zapisz na dysk i odczytaj z dysku
audiowrite('speech.wav',x,fpr,'BitsPerSample',bits); % zapisz nagranie
[y,fpr] = audioread('speech.wav');                   % odczytaj je z dysku
sound(y,fpr);                                        % odtworz nagranie
```

Problem 1.4 (* Pierwsze nagranie mowy). Zróbmy pierwszy eksperyment. Nagrajmy kilka słów naszej własnej mowy, używając karty dźwiękowej komputera i programu Matlab, albo telefonu. Potem obejrzyjmy nagrany sygnał - jaki jest jego kształt, minimalne i maksymalne wartości próbek? - oraz odsłuchajmy zrobionone nagranie. W tym celu należy uruchomić program 1.3. Powtórzmy nagranie kilka razy, zmieniając wartości parametrów na inne, np. inną liczbę bitów, inną częstotliwość próbkowania. Nagrajmy inny sygnał, np. melodyjne gwizdanie, chrypienie, syczenie.

Problem 1.5 (* Wykorzystanie zakresu dynamicznego przetwornika A/C). Staraj się mówić jakieś krótkie zdanie bardzo głośno podczas nagrywania: zaobserwuj na rysunku `plot(t,x,'b.-')` widoczny efekt "obcięcia/nasycenia", kiedy sygnał przekracza dozwoloną wartość minimalną i maksymalną przetwornika A/C. Następnie mów to samo zdanie bardzo cicho podczas nagrania: zaobserwuj na rysunku `plot(t,x,'bo-')` widoczne w sygnale poziomy kwantyzacji. Korzystniej jest kiedy sygnał zajmuje cały dostępny zakres dynamiczny przetwornika A/C (od wartosci min do max), zwłaszcza kiedy krok kwantyzacji jest duży (mała liczba bitów).

Problem 1.6 (* Rozdzielczość amplitudowa (kwantyzacja wartości) i czasowa (dyskretyzacja w czasie)). Nagraj to samo zadanie z kilkoma słowami zawierającymi fonemy bezdźwięczne ('s','c','sz','cz'). Mów normalnie, niezbyt głośno. Zastosuj dwie opcje nagrywania: 1) niska częstotliwość próbkowania (8 kHz) i mała liczba bitów na próbkę (8b), 2) wysoka częstotrliwość próbkowania (48 kHz) i duża liczba bitów na próbkę (16b). Porównaj wzrokowo wykresy obu sygnałów. Odsłuchaj oba nagrania. Czy widzisz i słyszysz różnicę? Drugie nagranie powinno być bogatsze akustycznie (więcej szczegółów, wyższe częstotliwości).

Problem 1.7 (1/2* Dlaczego należy zapamietać wartość częstotliwości próbkowania?). Nagraj dźwięk (mowę, muzykę) z użyciem karty dźwiękowej z częstotliwością próbkowania `fpr=22050;`. Potem odsłuchaj nagranie z użyciem poprawnej (22050) i niepoprawnej (8000, 11025, 32000, 44100, 48000) częstotliwości próbkowania: `sound(x,fpr_wybrana)`. Wytłumacz dlaczego mowa/muzyka jest szybsza lub wolniejsza?

Problem 1.8 (* Miksowanie sygnałów cyfrowych). Dodaj do siebie nagrania wykonane z tą samą lub różną częstotliwością próbkowania: `x=x1+x2+x3`. Odsłuchaj wynik miksowania dźwięku. Pamiętaj: dodawane wektory muszą mieć tę samą długość i orientację (wszystkie pionową albo poziomą), dlatego na końcu przytnij je lub uzupełnij zerami. Zwróć uwagę, że w przypadku dodania do siebie sygnałów o różnych częstotliwościach próbkowania, potem nie można jednocześnie odtworzyć ich obu poprawnie: jeden jest zawsze grany za wolno lub za szybko. Zapoznaj się z funkcją `resample()`. Przed dodawaniem sygnałów zwiększ mniejszą częstotliwość próbkowania (lepsza jakość wyniku) albo obniż większą wyższą częstotliwość próbkowania (gorsza jakość wyniku).

Problem 1.9 (Montaż dźwięku - całych zdań z pojedynczych słów).** Nagraj długie zdanie. Wyświetl wszystkie próbki sygnału - powiększ wybrane fragmenty sygnału (za pomocą lupy). Podziel długi wektor liczb całego zdania na krótsze wektory liczb, związane z poszczególnymi słowami: `x1,x2,x3,...`. Połącz wektory słów w zmienionej kolejności, np. `x=[x3; x1; x2; x7;...];`. Odłuchaj wynik montażu dźwięku.

Problem 1.10 (* Mowa dźwięczna i bezdźwięczna. Montaż dźwięku - słów z pojedynczych głosek).** Nagraj kilka fonemów dźwięcznych (a, e, i, o, u), podczas wypowiadania których struny głosowe pracują, oraz kilka bezdźwięcznych (s, c, h, k), podczas wypowiadania których struny głosowe nie pracują. Obejrzyj sygnały. Następnie nagraj głoski "wybuchowe" dźwięczne (d, g) (struny pracują) oraz bezdźwięczne (k, p)(struny niepracują). Obejrzyj sygnały. Potem nagraj całe słowa i zdania. Powiększ interesujące cię fragmenty sygnałów. Poszukaj okresowych fragmentów oscylacyjnych oraz nieokresowych fragmentów szumowych. Na koniec **spróbuj połączyć pojedyncze głoski w całe słowa.**

Problem 1.11 (* Częstotliwość podstawowa pracy strun głosowych).** Spróbuj znaleźć (obliczyć) wartość częstotliwości podstawowej pracy twoich strun głosowych podczas normalnego mówienia na podstawie obserwacji nagranego sygnału. Odczytaj z rysunku największą wartość powtarzania się sygnału (okres sygnału w sekundach) oraz oblicz jego odwrotność. Powinieneś otrzymać liczbę z przedziału $[80-250]$ Hz. Sprawdź różne samogłoski (a, e, i, o, u). Nagraj samogłoskę 'a' wypowiadaną przez ciebie: basem, barytonem, tenorem,.., sopranem, altem, czyli podczas zmiany częstotliwości pracy strun głosowych - od wartości małych do dużych (coraz wyższe częstotliwości).

TEMAT #5: Słuchanie różnych intrygujących dźwięków rzeczywistych. Miksowanie ich. Pobierz kilka nagrań dźwiękowych z Internetu, przykładowo ze strony https://www.findsounds.com/. Zbadaj (obejrzyj) je, zmiksuj ze sobą.

Problem 1.12 (* Studio dźwiękowe).** Znajdź w Internecie lub nagraj różne dźwięki. Narysuj sygnały (z wyskalowaniem i opisem osi), odsłuchaj je. Zmiksuj synały, tzn. dodaj je do siebie (np. mowę do muzyki albo śpiew ptaka do szumu miasta). Zanim to zrobisz, najpierw dopasuj do siebie częstotliwości próbkowania sygnałów, najlepiej do najwyższej — zastosuj funkcje Matlaba `rat(), resample()`. Następnie dopasuj do siebie długości sygnałów (dodaj zera do sygnału krótszego) oraz ich orientację (wszystkie wektory powinny być poziome lub pinowe). Możesz także pociąć sygnał na fragmenty (podzielić długi wektor liczb na krótsze wektory) oraz ponownie je połączyć w zmienionej kolejności. Przykładowo, zmontować rozmowę człowieka ze zwierzęciem albo maszyną.

Problem 1.13 (* Fonokardiogram: słuchanie sygnału EKG).** Wczytaj sygnał ECG, dołaczony do pierwszego wykładu/laboratorium: `clear all; load ECG100.mat; whos;`. Narysuj go: `x=y(1,:); plot(x)`. Znajdź częstotliwość próbkowania `fpr`, zakładając 72 uderzenia na minutę, oraz wyskaluj odpowiednio oś czasu (`N=length(x); dt=1/fpr; t=dt*(0:N-1)`). Potem przepróbkuj sygnał do częstotliwości zbliżonej do 8000 Hz: `xup=resample(x,UP,DOWN)`, np. pisząc `xup=resample(x,20,1)` w celu zmiany czestotliwości próbkowania 400 Hz na 8000 Hz (8000/400=20/1). Potem odsłuchaj nadpróbkowany sygnał: `soundsc(xup,8000)`.

TEMAT #6: Generowanie sygnałów w zastosowaniach technicznych i słuchanie ich. Telefon z wybieraniem tonowym DTMF. Transmisja bitów.

Problem 1.14 (* Telefon z wybieraniem tonowym).** Znajdź w Internecie informacje na temat standardu DTMF (Dual Tone Multi Frequency). Przyjmij częstotliwość próbkowania $f_{pr} = 16000$ Hz, długość trwania sygnału jednej cyfry 0.5 sekundy oraz wygeneruj sygnał `s` odpowiadający twojemu numerowi telefonu. Narysuj ten sygnał oraz odsłuchaj go, przyjmując w funkcji `soundsc(s, fpr)` wartości f_{pr} równe 8, 16, 24, 32 i 48 kHz.

Problem 1.15 ((*)(**) Transmisja bitów).** Przyjmij częstotliwość próbkowania $f_{pr} = 16$ kHz oraz $T = 100$ milisekund. Wygeneruj sygnał transmitujący bity kodów znaków ASCII Twojego imienia, np. Janek. Jeśli bit jest równy "0" to czasie T jest transmitowana sinusoida o częstotliwości $f_c = 500$ Hz, natomiast kiedy "1" — to minus sinusoida. Narysuj ten sygnał oraz odsłuchaj go, przyjmując w funkcji `soundsc()` wartości f_{pr} równe 8, 16, 24, 32 i 48 kHz. (*) Czy byłbyś wstanie napisać program odczytujący bity z tego sygnału? (**) Co zrobić, aby szybciej przesłać te same bity? Transmitować krótsze fragmenty sygnału z pojedynczym bitem? A może transmitować kilka bitów jednocześnie? Jak? Np. dodatkowo zmieniać amplitudę sinusoidy i jej przesunięcie kątowe, czyli fazę? Czy umiałbyś to zrobić? Czy umiałbyś zdekodować taki sygnał?

Laboratorium 2
Sygnały: Generacja, modulacja, cechy

Streszczenie Podczas tego laboratorium nauczymy się generować sygnały różnych typów (np. zmodulowane w amplitudzie i częstotliwości). Poznamy także definicje podstawowych cech/deskryptorów sygnałów (takich jak: wartość minimalna, maksymalna, srednia, RMS, wariancja, energia, moc, auto-korelacja) oraz sposoby ich obliczania w języku Matlab.

TEMAT #1: Generacja sygnałów o różnych kształtach. Sygnały deterministyczne są generowane na podstawie *przepisu* matematycznego definiującego ich kształt, tzn. z wykorzystaniem funkcji matematycznych, np. sin(), cos(), exp(), patrz tabela 2.2. Sygnały *proste* mogą się dodawać (składowe addytywne) i mnożyć (składowe multiplikatywne). Dobrym przedstawicielem tej drugiej grupy jest sygnał *tłumiony sinus*, czyli sinusoida wywmnożona z eksponentą (patrz tabela 2.2) — rozwiązanie równania różniczkowego drugiego rzędu, opisującego tłumione oscylatory mechaniczne i elektrycze. Zwróćmy uwagę, że sama sinusoida jest rozwiązaniem równania oscylatorów niettłumionych. Szczególną rolę odgrywają sygnały zmodulowane. Są to sygnały sinusoidalne, mające amplitudę $A_x(t)$ i fazę $\phi_x(t)$ zmienną w czasie (f_{pr} — częstotliwość próbkowania):

$$x(t) = A_x(t) \cdot \sin(2\pi f_x t + \phi_x(t)), \quad x(n) = A_x(n) \sin\left(2\pi \frac{f_x}{f_{pr}} n + \phi_x(n)\right). \tag{2.1}$$

Częstotliwość chwilowa (*intantaneous*) sygnału sinusoidalnego $\sin(\alpha(t))$ jest definiowana jako pochodna jego zmiennego w czasie kąta $\alpha(t)$, podzielona przez 2π. Dlatego dla sygnału (2.1) otrzymujemy:

$$f_{inst}(t) = \frac{1}{2\pi} \frac{\mathrm{d}\alpha(t)}{\mathrm{d}t} = f_x + \frac{1}{2\pi} \frac{\mathrm{d}\phi(t)}{\mathrm{d}t}. \tag{2.2}$$

Z tego powodu, aby otrzymać sygnał mający częstotliwość chwilową równą $f_x + m_F(t)$, czyli zmodulowaną w częstotliwości sygnałem $m_F(t)$, należy w równaniu (2.1) podstawić:

$$\phi_x(t) = \int_0^t m_F(t)dt, \qquad \phi_x(n) = \sum_0^n m_F(n)dt, \qquad \text{w Matlabie: } \texttt{phi\_x(n)=cumsum(m\_F(n))*dt}. \tag{2.3}$$

Dlaczego? Ponieważ pochodna z całki oznaczonej jest równa funkcji podcałkowej.
Patrz tabela 2.2. Przeanalizuj kod Matlaba z listingu 2.1. Uruchom program, zwróć uwagę na wykres sygnału. Wybierz i wykonaj wybrane przez ciebie zadania, które są zaproponowane poniżej.

Listing 2.1: Generacja różnych sygnałów deterministycznych w Matlabie

```
% cps_02_sygnaly.m
clear all; close all;

fpr=100; Nx=1000;              % czestotliwosc probkowania, liczba probek
dt = 1/fpr;                    % okres probkowania
t = dt*(0:Nx-1);               % chwile pobierania probek
x1=sin(2*pi*10*t);             % sinus 10 Hz
x2=sin(2*pi*1*t);              % sinus 1 Hz
x3=exp(-5*t);                  % eksponenta opadajaca w czasie
x4=exp(-25*(t-0.5).^2);        % gaussoida
x5=sin(2*pi*(0*t+0.5*20*t.^2)); % liniowy przyrost czest. (LFM): od 0 Hz, +20Hz/s
x6=sin(2*pi*(10*t-(9/(2*pi*1)*cos(2*pi*1*t)))); % sinus. FM: 9Hz wokol 10Hz 1x na sec
x7=sin(2*pi*(10*t+9*cumsum(x2)*dt));            % to samo co x6; dlaczego?
x = x1;                        % wybor: x1,x2,...,x7, 0.23*x1 + x2, x1.*x3, ...
plot(t,x,'o-'); grid; title('Sygnal x(t)'); xlabel('czas [s]'); ylabel('Amplituda');
```

Problem 2.1 (* Różnorodność sygnałów). Uruchom program. Narysuj i obejrzyj osobno każdy sygnał (`x=x1; x=x2; ...`). Obejrzyj kształ; 1) sumy różnych sygnałów (np. liniowej superpozycji `x=0.25*x1+2*x2;`), oraz 2) iloczynów różnych sygnałów (np. wyniku pomnożenia sygnałów oscylacyjnych `x1, x5, x6, x7` przez inne sygnały, np. `x2, x3, x4` - sinusoidę, eksponentę i gaussoidę), przykładowo:
`x=(1+0.5*x2).*x1; x=x2.*x1; x=x3.*x1; x=x4.*x1;`
`x=(1+0.5*x2).*x5; x=(1+0.5*x2).*x6;`).
Dwie ostatnie sinusoidy są jednocześnie zmodulowane w amplitudzie (AM) i częstotliwości (FM). Prześledź zmianę wartości amplitudy i częstotliwości sygnału w czasie.

Problem 2.2 (* Sprawdzanie twierdzenia o próbkowaniu - po raz drugi). Użyj programu z listingu 2.1. Ustaw `fpr=1000; Nx=10*fpr; df=200;` Wygeneruj sinusoidę, której częstotliwość oscylacji rośnie liniowo `df` herców na sekundę, zaczynając od częstotliwości 0 Hz:
`x=cos(2*pi*(0*t+0.5*df*t.^2));`
Zapoznaj się z kształtem sygnału. Wytłumacz dlaczego powtarza się on okresowo: przecież częstotliwość synału jest inna, coraz większa. Ustaw `fpr=8000; Nx=10*fpr; df=2000;` i wygeneruj nowy sygnał. Zapoznaj się z kształtem sygnału, posłuchaj sygnału: `sound(x,fpr);`.
Wywołaj funkcję Matlaba: `spectrogram(x,256,256-64,512,fpr)`, która pokazuje zmianę częstotliwości sygnału w czasie. Czy zgadzasz się ze zmiennością częstotliwości na rysunku? Przecież sam wygenerowałeś sygnał i wiesz jaka ona jest w każdej chwili!

Problem 2.3 (* Sprawdzanie twierdzenia o próbkowaniu - po raz trzeci - sygnały zespolone w telekomunikacji). Jeśli próbkowanie dalej cię "kręci", to powtórz poprzednie zadanie tylko dla nastepującego sygnału o wartościach zepolonych:
`x=exp(j*2*pi*(0*t+0.5*df*t.^2));`
Co możesz powiedzieć w tym przypadku? Jaki jest dopuszczalny zakres zmienności częstotliwości sygnału? Czy dalej $[0, \frac{f_{pr}}{2}]$?

Problem 2.4 (* Magia sygnałów z łączną modulacją AM-FM. Syntezator dźwięku. Nowy instrument muzyczny). Wygeneruj sygnał z jednoczesną modulacją AM-FM, korzystając z programu 2.1:
`x=(1+0.5*x2).*x6;`
Dla modulacji AM ustaw częstotlowość modulacji $f_A = 0.5$ Hz oraz głębokość modulacji $k_A = 0.25$, natomiast dla modulacji FM: częstotliwość nośną $f_0 = 5$ Hz, częstotliwość modulującą $f_F = 2$ Hz oraz głębokość modulacji $k_F = 5$ Hz. Jaki jest kształt sygnału? Sprawdź wzrokowo czy zmiany amplitudy i częstotliwości sygnału `x` są takie jak zadane. Skorzystaj z tabeli 2.2 - tutaj znajdziesz definicje i kod Matlaba dla modulacji AM i FM. Zwiększ częstotliwość próbkowania ze 100 Hz do 8000 Hz oraz proporcjonalnie wszystkie częstotliwości poprzednio używane, a następnie odsłuchaj wygenerowany sygnał za pomocą funkcji `soundsc(x,fpr)` (ustaw `Nx=5*fpr`).

Problem 2.5 (* Jak brzmi suma tłumionych sinusoid?). Ustaw parametry `fpr=8000; Nx=3*fpr; f1=100;` i wygeneruj sinusoidę o częstotliwości 100 Hz, trwającą 3 sekundy. Wyświetl ją ze skalowaniem osi czasu, posłuchaj jej (`sound(x,fpr);`). Wygeneruj sumę trzech sinusoid o różnych amplitudach i częstotliwościach, mniejszych niż $\frac{f_s}{2}$: `x=x1+x2+x3;`. Przeanalizuj kształt sygnału (z użyciem funkcji lupy), posłuchaj sygnału. Pomnóż każdą składową sygnału 1,2,3 przez inną funkcję modulującą jej amplitudę, eksponente lub gaussoidę: `x=x.*exp(-t)` or `x=x.*exp(-(t-1.5).^2)`. Przeanalizuj kształt, posłuchaj.

Problem 2.6 (* Projektowanie sygnału alarmowego). Ustaw `fpr=8000; Nx=5*fpr;` i wygeneruj sygnał `x` trwający 5 sekund. Skorzystaj z tabeli 2.2 i zaprojektuj sygnał z łączną modulacją AM-FM dla wozu strażackiego, karetki pogotowia oraz samochodu policji. Przeanalizuj kształt, posłuchaj. (`sound(x,fpr);`)

Problem 2.7 (*(*) Komputerowa synteza muzyki). Wygeneruj krótki fragment podkładu muzycznego dla twojej ulubionej piosenki jako sekwencję fragmentów sinusoid o róznej częsttliwości: (`x=[x1,x2,x3,...];`. Skala muzyczna jest zdefiniowana w następujący sposób (patrz https://pages.mtu.edu/ suits/notefreqs.html):

- nuta A_k w k-tej oktawie ma częstotliwość $f_k^A = 2^k \cdot 27.5$ Hz, gdzie $k = 0, 1, 2, 3, ...8$;
- 12 nut $\{A_k, B_k^{flat} = B_k^b, B_k, C_k, C_k^{sharp} = C_k^{\#}, D_k, D_k^{sharp} = D_k^{\#}, E_k, F_k, F_k^{sharp} = F_k^{\#}, G_k, A_k^{flat} = A_k^b\}$ dla k-tej oktawy ma następujące częstotliwości $(m = 0, 1, 2, 3, ..., 11) : f_{k,m} = f_k^A \cdot 2^{m/12}$.

Przykładowy zapis nutowy wybranych piosenek dla instrumentów MIDI jest podany na stronie: https://mypianonotes.com/. Dwa punkty (**) otrzymuje się dla dłuższych piosenek, dobrego brzmienia i rozbudowanego programu.

Problem 2.8 ((**) Akord gitary/pianina. Wirtualny instrument muzyczny.).** W skali muzycznej dźwięki mają dokładnie określone częstotliwości: 8 oktaw, w każdej 12 nut (przeczytaj zadanie poprzednie). Jeśli chcemy czysto i pięknie grać, to struna gitary (lub pianina, skrzypiec, wiolonczeli, kontrabasu) powinna być nastrojona na konkretną, TĄ SAMĄ częstotliwość "muzyczną" f_0. Jak więc rozpoznajemy, że to gra konkretny instrument, np. pianino, jeśli częstotliwości są (powinny być) takie same? Ponieważ generowany dźwięk jest sumą tłumionych K sinusoid $x(n) = \sum_{k=1}^{K} A_k e^{-D_k n} \sin\left(2\pi \frac{f_k}{f_{pr}} n\right), \quad n = 1,2,...,Nx,$ o częstotliwościach harmonicznych (w przybliżeniu): $f_k = kf_0$, $k = 1,2,3,...$, oraz o różnych amlitudach A_k i tłumieniach D_k. To właśnie w wartościach amplitud i tłumień "ukrywa się" konkretny instrument i jego "barwa" (indywidualny charakter). Zastosuj następujące, zmierzone wartości parametrów i wygeneruj jeden akord pojedynczej struny pianina ($f_0 = 220$ Hz oraz $f_{pr} = 44100\ Hz$) i gitary ($f_0 = 110$ Hz oraz $f_{pr} = 22050$ Hz), podane w tabeli 2.1. Zwróć uwagę na różne częstotliwośći próbkowania użyte w obu przypadkach (!). Odsłuchaj. Zastąp zmierzone wartości częstotliwości wielokrotnościami częstotliwości podstawowej kf_0, obowiązującej dla wybranego innego tonu, pozostawiając niezmienione wartości A_k i D_k. Odsłuchaj. Brzmi $\pm$ poprawnie? Czy jesteś w stanie napisać program, który zagra jakąś prostą melodyjkę na wirtualnym pianinie albo gitarze? Jeśli to zrobisz, to otrzymasz dodatkowe 2 punkty, łącznie 4.

Table 2.1: Zmierzone wartości parametrów pojedynczych akordów pianina (P) i gitary (G)

Numer	(P) f_k [Hz]	(P) A_k	(P) D_k	(G) f_k [Hz]	(G) A_k	(G) D_k
1	219.99	0.35578	0.05081e-03	110.08	0.14836	0.03109e-03
2	440.03	0.06525	0.06723e-03	219.96	0.51531	0.29010e-03
3	660.07	0.02664	0.10755e-03	330.19	0.07154	0.02759e-03
4	879.57	0.00870	0.08566e-03	439.90	0.00928	0.02834e-03
5	1100.97	0.04392	0.06843e-03	550.33	0.00101	0.01352e-03
6	1321.62	0.03026	0.06489e-03	660.80	0.00219	0.10835e-03
7	1542.86	0.00420	0.07020e-03	772.02	0.01291	0.22861e-03
8	1763.08	0.00226	0.01801e-03	882.41	0.01744	0.19850e-03
9	1987.69	0.00200	0.04481e-03	992.76	0.00795	0.20998e-03
10	2211.73	0.00473	0.04356e-03	1325.17	0.00572	0.21626e-03

Problem 2.9 (Zagubieni na dworcu kolejowym).** Czy zdarzało ci się nic nie rozumieć z zapowiedzi pociągów czytanych na dworcu kolejowym? Mnie często. Dlaczego tak się dzieje? Ponieważ ściany dworcóch nie pochłaniają tylko odbijają falę dźwiękową i do słuchacza dociera jednocześnie kilka kopii tego samego komunikatu, ale opóźnionych o coraz większe wartości i coraz słabszych. SUMA SYGNAŁÓW! Nagraj jedno, krótkie zdanie `x` z częstotliwością próbkowania $f_{pr} = 8000$ Hz, np. "Pociąg do Warszawy odjedzie z peronu pierwszego" lub dowolne inne. Zbuduj trzy sygnały: 1) `x1` - oryginał: dodaj na końcu `x` 800 zer i pomnóż sygnał przez 1/2, 2) `x2` - opóźniona kopia #1: dodaj na początku i końcu `x` po 400 zer i pomnóż sygnał przez 1/4, 3) `x3` - opóźniona kopia #2: dodaj na początku `x` 800 zer i pomnóż sygnał przez 1/8. Dodaj trzy sygnały `x123=x1+x2+x3;` oraz odsłuchaj otrzymany wynik: `sound(x123,8000)`. Ja wyszło? Pociąg jest do Lublina czy Szczebrzeszyna? Napisz bardziej uniwersalny program, w którym wartości opóźnień (u nas: 0, 400, 800) i współczynników odbić (u nas: $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$) można zmieniać elastyczniej. Pobaw się nim przez chwilę. Podczas zaliczenia pochwal się czymś niesamowitym.

TEMAT #2: Obliczanie parametrów (cech, deskryptorów, właściwości) sygnałów. Obecnie będziemy wyznaczać wartości parametrów, charakteryzujących różne cechy/właściwości sygnałów, takie jak: średnia, wariancja, wartość RMS, energia, pomoc, auto-korelacja. Zapoznaj się z definicjami ww. cech sygnałów oraz ich implementacjami programowymi w języku Matlab, podanymi w tabeli 2.3.
Na szczególną uwagę zasługuje funkcja korelacji, podobieństwa do siebie dwóch sygnałów lub samopodobieństwa jednego sygnału, po wielokrotnym przesuwaniu drugiego z nich (lub kopii sygnału pierwszego). Obliczeniowo dla

sygnałów N-elementowych mamy: 1) przesuń drugi sygnał (lub kopię pierwszego) o k próbek, 2) wymnóż odpowiadające sobie (leżące nad sobą) $N-k$ próbki obu sygnałów, 3) dodaj wszystkie iloczyny, 4) unormuj sumę (patrz uwaga na dole tabeli), 5) powtórz powyższe dla przesunięć $k = 0,1,2,...,N-1$). Wzór jest następujący:

$$Rxy(k) = \sum_{n=1}^{N-k} x(n)y^*(n+k), \; k = 0,1,2,...,N-1. \tag{2.4}$$

Maksima funkcji auto/cross-korelacji $R_{xx/xy}(k)$ wskazują, że po przesunięciu o k próbek drugi sygnał *pokrył się* z pierwszym sygnałem (wszystkie iloczyny są dodatnie i ich suma jest duża). Świadczy to o powtarzaniu się fragmentów sygnału (okresowości sygnału) w przypadku autokorelcji (np. mowy dźwięcznej - z powodu okresowego ruchu strun głosowych) oraz o wystepowaniu podobnego fragmentu w dwóch sygnałach w przypadku funkcji korelacji wzajemnej (np. odbicia od obiektu w przypadku sygnału radarowego).

Problem 2.10 (* Obliczanie wartości deskyptorów sygnałów). Użyj kodu Matlab z tablicy 2.3, ustaw `N=10000` oraz oblicz wartości podstawowych cech sygnałów dla: sinusoidy (`x1=sin(2*pi/1000*(0:N-1))`), równomiernego szumu w przedziale [0,1] (`x2=rand(1,N)`) oraz szumu gaussowskiego (`x3=randn(1,N)`). Podstaw (`y=x;`) jeśli jest to konieczne. Czy obliczone wartości są według ciebie poprawne? Narysuj funkcję auto-korelacji trzech powyższych sygnałów. Oblicz wartości wszystkich parametrów dla sumy sygnałów `x=x1+x3;`. Narysuj jej funkcję auto-korelacji.

Problem 2.11 (Obliczanie funkcji korelacji).** Sam napisz funkcję do obliczania funkcji korelacji - skorzystaj z definicji podanej w tabeli 2.3). Sprawdź poprawność twojej implementacji, porównując otrzymane wyniki z funkcjami `xcorr(x)` oraz `xcorr(x,y)`. Nie zapomnij o normowaniu sumy iloczynów (dzielenie przez 1, N albo $N-k$). Znajdź definicję współczynnika korelacyjnego i też ją zaprogramuj.

Problem 2.12 (Obliczanie częstotliwości podstawowej oscylacji strun głosowych).** Nagraj dowolną głoskę dźwięczną (a, e, i, o, u, b, d, g, m, n,...). Zapoznaj się z kształtem sygnału. Znajdź największy okres powtarzania się kształtu sygnału - wyrażony w liczbie próbek. Użyj funkcji Matlaba `Rxx=xcorr(x,x)` do obliczenia funkcji auto-korelacji sygnału. Wartość `Rxx(N)`, gdzie `N=length(x)`, odpowiada `k=0`-owemu przesunięciu sygnału (po lewej - przesunięcia w lewo (minus), po prawej - przesunięcia w prawo (plus)). Narysuj na jednym rysunku wartości funkcji auto-korelacji, obliczone dla różnych opcji: 'biased', 'unbiased', 'coeff', 'none' (opisz poprawnie oś poziomą). Czym się one różnią? Sprawdź czy indeks maksimum funkcji auto-korelacji jest równy okresowi sygnału, wyznaczonemu bezpośrednio z przebiegu czasowego sygnału (za pomocą kursora).

Problem 2.13 (* Wyznaczenie okresu powtarzania się sygnału EKG). Rozwiąż ostatnie zadanie, ale dla sygnału EKG `ECG100.mat` dołączonego do materiałów, nie dla sygnału mowy. Rozpocznij program tak: (`clear all; load ECG100.mat; whos;`).

Problem 2.14 (* Mini-radar). Nagraj z częstotliwością próbkowania $f_{pr} = 8000$ Hz dwa 5-sekundowe dźwięki upadku tego samego przedmiotu z biurka, ale w różnych chwilach, np. zepchniętego po 1 sekundzie oraz po 3 sekundach. Skorzystaj z funkcji korelacji wzajemnej do obliczenia czasu opóźnienia pomiędzy dwoma “upadkami”. Alternatywnie, nagraj dwa klaśniecia w obie ręce, wykonane w różnych chwilach (albo powiedz dwa razy to samo słowo, np. "raz", "dwa" lub "trzy").

Table 2.2: Funkcje definiujące sygnały deterministyczne oraz ich obliczanie w Matlabie

Typ sygnału	Definicja matematyczna	Kod Matlaba
Sinus, kosinus	$x_1(t) = A \cdot \sin(2\pi f_0 t + \phi)$	`x1=A*sin(2*pi*f0*t+fi);`
Harmoniczny cmplx	$x_2(t) = A \cdot e^{j(2\pi f_0 t + \phi)}$	`x2=A*exp(j*(2*pi*f0*t+fi));`
Exponenta	$x_3(t) = A \cdot e^{-\lambda t}$	`x3=A*exp(-lamb*t);`
Tłumiony sinus	$x_4(t) = A \cdot e^{-\lambda t} \cdot \sin(2\pi f_0 t + \phi)$	`x4=x3.*sin(2*pi*f0*t+fi);`
Gaussoida[†]	$x_5(t) = \frac{1}{\sqrt{2\pi a}} e^{-\frac{(t-t_0)^2}{2a}}$	`x5=exp(-(t-t0).^2/(2*a))/sqrt(2*pi*a);`
Z modulacją AM[‡]	$x_6(t) = A(1 + k_A m_A(t)) \cdot \sin(2\pi f_0 t)$	`x6=A*(1+kA*mA).*sin(2*pi*f0*t);`
Z modulacją FM[‡]	$x_7(t) = A \sin\left(2\pi\left(f_c t + k_F \int_0^t m_F(t) dt\right)\right)$	`x7=A*sin(2*pi*(f0*t+kF*cumsum(mF)*dt));`

[†] $a = \sigma^2$ – wariancja funkcji
[‡] A, F – amplituda i częstotliwość, k_A, k_F – głębokość modulacji amplitudy i częstotliwości, $m_A(t), m_F(t)$ – funkcje modulujące sygnał w amplitudzie i częstotliwości

Table 2.3: Podstawowe właściwości sygnału oraz ich obliczanie w Matlabie

Cecha	Definicja matematyczna	Kod Matlaba
Średnia	$\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x(n)$	`mean(x); sum(x)/length(x);`
Wariancja	$\sigma_x^2 = \frac{1}{N} \sum_{n=1}^{N} (x(n) - \bar{x})^2$	`var(x); sum((x-mean(x)).^2)/N;`
Odch. standard.	$std_x = \sqrt{\frac{1}{N-1} \sum_{n=1}^{N} (x(n) - \bar{x})^2}$	`std(x); sqrt(sum((x-mean(x)).^2)/(N-1);`
Energia	$E_x = dt \cdot \sum_{n=1}^{N} x^2[n]$	`dt*sum(x.^2);`
Moc	$P_x = \frac{E_x}{N \cdot dt} = \frac{1}{N} \sum_{n=1}^{N} x^2[n]$	`sum(x.^2)/length(x);`
RMS	$rms_x = \sqrt{P_x} = \sqrt{\frac{1}{N} \sum_{n=1}^{N} x^2[n]}$	`sqrt(sum(x.^2)/length(x))`
SNR	$10 \cdot \log_{10} \frac{P_x}{P_n}$ (dB)	`10*log10(Px/Pn);`
Korelacja[a]	$R_{xx}[k] = \frac{1}{C} \sum_{n=1}^{N-k} x(n) y^*(n+k)$	`xcorr(x); xcorr(x,y);`

[a] C jest stałą normalizacji równa 1, N albo $N - k$, decydująca o właściwościach estymatora (patrz opis funkcji `xcorr()`

AGH

Laboratorium 3
Transforacje ortogonalne sygnałów

Streszczenie Na tym laboratorium nauczymy się: 1) jak N-elementowy wektor próbek sygnału może być przedstawiony jako ważona suma elementarnych/bazowych wektorów ortogonalnych: $\bar{\mathbf{x}} = \sum_{k=1}^{N} c_k \cdot \bar{\mathbf{v}}_k$, oraz 2) jak obliczyć wagi $c_k, k = 1, 2, ..., N$, występujące w tej sumie, czyli współczynniki ortogonalnej transformacji/reprezentacji sygnału — za pomocą iloczynu skalarnego dwóch wektorów $c_k = \sum_{n=1}^{N} x(n) v_k^*(n),\ k = 1, 2, ..., N$ (znak "*" oznacza sprzężenie zespolone, które jest koniczne, jeśli wektory $\bar{\mathbf{v}}_k$ są zespolone).

TEMAT #1: Pojęcie ortogonalności. Dwa wektory $\bar{\mathbf{v}}_1$, $\bar{\mathbf{v}}_2$ są ortgonalne (prostopadłe) w przestrzeni wektorowej N-wymiarowej, jeśli ich iloczyn skalarny jest równy 0, czyli suma iloczynów odpowiadających sobie elementów:

$$\sum_{n=1}^{N} v_1(n) v_2^*(n) == 0. \tag{3.1}$$

Dwa przykładowe 8-elementowe wektory ortgonalne, należace do 8-wymiarowej przestrzeni wektorowej, są zdefiniowane w prograni poniżej i jest sprawdzona ich ortogonalność. Macierz kwadratowa jest ortogonalna, jeśli wszystkie jej wiersze (lub kolumny) są do siebie ortogonalne. Macierz odwrotna macierzy ortogonalnej jest jest sprzężoną transpozycją (sprzężenie to negacja części urojonej elementów macierzy). Dlatego dla macierzy ortogonalnej $\mathbf{A}$ mamy:

$$\mathbf{A}^{-1} = (\mathbf{A}^*)^T, \qquad \Rightarrow \qquad \mathbf{A}^{-1} \cdot \mathbf{A} = (\mathbf{A}^*)^T \cdot \mathbf{A} = \mathbf{I}. \tag{3.2}$$

Listing 3.1: Ortogonalność wektorów i macierzy

```
% Koncepcja ortogonalnosci wektorow i macierzy
clear all; close all
v1 = [ 1 0 0 0 0 0 0 0 ];     % wektor nr 1
v2 = [ 0 1 0 0 0 0 0 0 ];     % wektor nr 2
prod1 = sum( v1 .* conj(v2) ) % kiedy suma iloczynow odpowiadajacych sobie
prod2 = dot( v1, v2 )         % elementow dwoch wektorow jest rowna 0,
prod3 = v1*v2'                % to wektory te sa ortogonalne w przestrzeni euklidusowej
A = eye(8);                   % diagonalna, ortogonalna macierz identycznosciowa
prod4 = inv(A) * A            % wynik powinien dac macierz identycznosciowa
prod5 = A' * A                % inv(A)->A' dla macierzy ortogonalnych
```

Problem 3.1 (* Krótkie wektory ortonormalne). Kiedy dwa pionowe wektory $\bar{\mathbf{x}}$, $\bar{\mathbf{y}}$ są ortogonalne ($\bar{\mathbf{x}}^{*T} \cdot \bar{\mathbf{y}} = 0$, `sum(x.*y)==0`) oraz równocześnie unormowane ($\bar{\mathbf{x}}^{*T} \cdot \bar{\mathbf{x}} = 1$ i $\bar{\mathbf{y}}^{*T} \cdot \bar{\mathbf{y}} = 1$, `sum(x.*x)==1` i `sum(y.*y)==1`), to są one otronormalne. Znajdź cztery 4-elementowe wektory, które są wzajemnie orthonormalne. Przyjmij, że mają one tylko niezerowe elementy równe 1 oraz -1, pomnożone przez $\frac{1}{\sqrt{4}}$. Zbuduj z nich macierz `A`: włóż je do kolejnych wierszy macierzy. Porównaj `inv(A)` oraz `A'`. Wynik powinien być taki sam.

Problem 3.2 (* Większe macierze orthonormalne). Znajdź osiem 8-elementowych wektorów, będących wzajemnie ortogonalnych i mających tylko elmenty o wartościach 1 oraz -1, pomnożnych przez $\frac{1}{\sqrt{8}}$. Potraktuj je jako wiersze macierzy i zbuduj z tych wektorów macierz ortogonalną `A` o wymiarach 8×8. Porównaj ze sobą macierze `inv(A)` oraz `A'`. Są różne czy identyczne? Znajdź w sieci informacje dotyczące dyskretnej transformacji Haara, Hadamarda, Walsha, ...

Problem 3.3 (Macierz ortogonalna transformacji DCT-IV (kosinusowej)).** Kolejne wiersze typowych macierzy ortogonalnych są zbudowane z coraz szybszych oscylacji sinusoidalnych lub kosinusoidalnych (o coraz większej częstotliwości). Wygeneruj ortogonalną macierz `A` dyskretnej transformacji kosinusowej DCT-IV (Discrete Cosine Transform IV):

$$A(k,n) = v_k(n) = \sqrt{\frac{2}{N}} \cos\left(\frac{\pi}{N}(k+\frac{1}{2})(n+\frac{1}{2})\right), \quad k = 0...N-1, \;\; n = 0...N-1, \tag{3.3}$$

dla $N = 25$, wykorzystując następujący kod Matlaba (`k` - numer wiersza $0...N-1$, `n` - numer kolumny $0...N-1$):

```
for k=0:N-1
A(k+1,1:N)=sqrt(2/N)*cos(pi/N*(k+1/2)*((0:N-1)+1/2);
end
```

albo (alternatywnie)

```
k=(0:N-1); n=(0:N-1); A=sqrt(2/N)*cos(pi/N*(k'+1/2)*(n+1/2));
```

Narysuj w pętli po kolei wartości wszystkich wierszy macierzy. Zobaczysz kolejne funkcje bazowe (elementarne) transformacji DCT-IV - zauważ zwiekszającą się liczbę powtórzeń kosinusa (liczby oscylacji). Sprawdź ortogonalność losowo wybranych 5-ciu par wierszy macierzy - czy pojedyncze wiersze są ortogonalne same do siebie? Oblicz macierz odwrotną `inv(A)` i porównaj ją z macierzą sprzężoną-transponowaną `A'` ($(\mathbf{A}^*)^T$) — powinieneś otrzymać to samo. Porównaj macierze otrzymane w wyniku następujących operacji: 1) `inv(A)*A`, 2) `A'*A`. W obu przypadkach powinieneś otrzymać macierz identycznościową z jedynkami na głównej przekątnej.

Problem 3.4 (* Macierze ortogonalne transformacji DST i Hartleya). Podobnie do poprzedniego problemu: wygeneruj w Matlabie macierze dyskretnej transformaci sinusowej (DST) oraz dysktretnej transformacji Hartleya:

```
k=(0:N-1); n=0:N-1;
A=sqrt(2/(N+1))*sin(pi*(k'+1)*(n+1)/(N+1)); % DST
A=sqrt(1/N)*( cos(2*pi/N*k'*n) + sin(2*pi/N*k'*n) ); % Hartley
```

Wyświetl wartości kolejnych wierszy macierzy. Sprawdź ortogonalność wierszy. Sprawdź ortogonalność macierzy.

TEMAT #2: Wynik ortogonalnej transformacji sygnałów i transformacji odwrotnej. Obecnie zastosujemy macierze ortogonalne do analizy sygnałów. Na początku wygenerujemy ortogonalną macierz $\mathbf{S}$, mającą ortogonalne funkcje bazowe w kolumnach, oraz obliczymy macierz do niej odwrotną $\mathbf{A} = \mathbf{S}^{-1} = \mathbf{S}^{*T}$, czyli transpozycję oraz sprzężenie zespolone macierzy $\mathbf{S}$. Potem wykonamy następujace operacje:

1. po pierwsze, wyznaczymy podobieństwo analizowanego sygnału do wierszy macierzy ortogonalnej $\mathbf{A}$:
$$\bar{\mathbf{c}} = \mathbf{A} \cdot \bar{\mathbf{x}}, \tag{3.4}$$
2. następnie, opcjonalnie, zmodyfikujemy wartości wybranych współczynników transformaty $\bar{\mathbf{c}}$, e.g. $c(3) = 0$; $c(3) = 10 * c(3)$;
3. w końcu, wykorzystamy równoważność $\mathbf{A}^{-1} = \mathbf{A}^{*T} = \mathbf{S}$, prawdziwą dla każdej macierzy ortogonalnej, oraz odtworzymy nasz sygnał na podstawie współczynników transformaty $\bar{\mathbf{c}}$ z użyciem równania:
$$\bar{\mathbf{x}} = \mathbf{A}^{-1} \cdot \bar{\mathbf{c}} = \mathbf{S} \cdot \bar{\mathbf{c}}, \tag{3.5}$$
czyli przedstawimy sygnał jako sumę ważoną kolumn macierzy $\mathbf{S}$.

Wyznaczenie: 1) współczynników ortogonalnej transformacji sygnału (wynikowej transformaty), 2) odtworzenia (syntezy) sygnału na podstawie tych współczynników, w zapisie nie-macierzowym jest opisane następującymi wzorami:

(analiza) $$c_k = \sum_{n=1}^{N} x(n) v_k^*(n), \;\; k = 1,2,...,N, \tag{3.6}$$

(synteza) $$x(n) = \sum_{k=1}^{N} c_k \cdot v_k(n), \;\; n = 1,2,...,N. \tag{3.7}$$

Pierwsze równanie (analiza) to iloczyn skalarny analizowanego sygnału z każdym wektorem bazowym (wyznaczanie miary podobieństwa pomiędzy oboma wektorami).

Drugie równanie (synteza) to obliczanie n-tej próbki sygnału jako sumy n-tych próbek wszystkich wektorów bazowych (wzorców).

Na rysunku 3.1 zilustrowano sekwencję operacji analizy i syntezy sygnałów z użyciem transformacji ortogonalnej, używającej oscylujących sygnałów bazowych/wzorcowych. Po etapie analizy otrzymujemy współczynniki podobieństwa analizowanego sygnału do wzorców ("przepis na zupę"). Natomiast po operacji syntezy – mamy odtworzony sygnał na podstawie ww. współczynników ("zupę ugotowaną według przepisu").

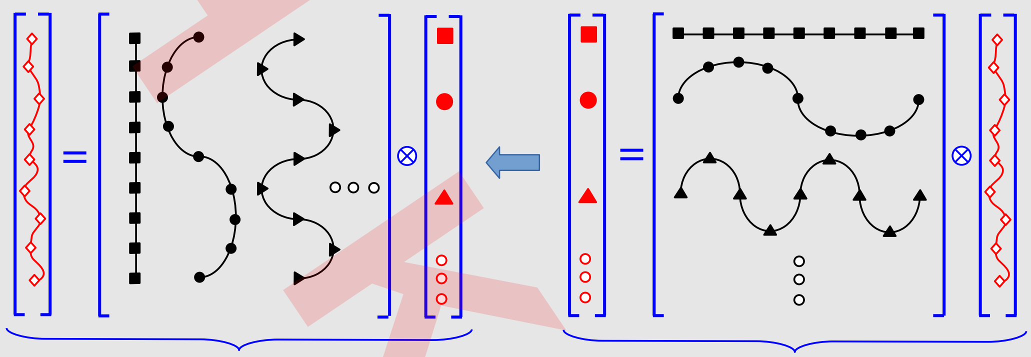

Fig. 3.1: Ilustracja użycia transformcji ortogonalnej do analizy sygnału (po prawej) i syntezy sygnału (po lewej): "znajdź przepis na zupę i ugotuj zupę według tego przepisu".

WAŻNE: Liczba współczynników transformaty różnych od zera jest mała, jeśli sygnał jest dokładnie sumą kilku wektorów bazowych. Kiedy dowolna składowa sygnału jest różna od każdego wektora bazowego (ma nieco inną częstotliwość oscylacji, jest przesunieta w czasie), to wówczas zbiór współczynnków jest rozmyty w otoczeniu najbardziej podobnego wzorca.

Listing 3.2: Ortogonalna transformacja sygnału z użyciem DCT-IV

```
% cps03_trans.m
clear all; close all

% Orthogonal matrix for DCT-IV orthogonal transform
N = 100;                                    % wymiar macierzy kwadratowej; 25, 100
k = (0:N-1); n=(0:N-1);                     % k-kolumny/funkcje, n-wiersze/probki
S = sqrt(2/N)*cos(pi/N*(n'+1/2)*(k+1/2)); % macierz syntezy
A = S';                                     % macierz analizy: transpozycja i sprzezenie S

% S*A, pause  % sprawdzenie ortogonalnosci: macierz z jedynkami na przekatnej?

x1 = 10*S(:,5);                                  % sygnal #1
x2 = 20*S(:,10);                                  % sygnal #2
x3 = 30*sqrt(2/N)*cos(pi/N*(n'     +1/2)*(10.5+1/2) );  % sygnal #3
x4 = 30*sqrt(2/N)*cos(pi/N*(n'+N/4+1/2)*(10  +1/2) );  % sygnal #4
x5 = randn(1,N);                                  % sygnal #5
x = x1; % + x2;                                   % wybor x1, x2, x3, x4, x1+x2, x1+x3, x1+x4
figure; plot(x,'bo-'); title('x(n)'); grid;       % rysunek sygnalu wejsciowego

c = A*x;                 % analiza sygnalu: wyznaczenie wspolczynnikow transformacji
figure; stem(c); grid;   % pokazanie wspolczynnikow transformacji
```

```
%c(5) = 0;                % opcjonalne usuniecie skladowej x3 z sygnalu
y = S*c;                  % synteza sygnalu: suma przeskalowanych funkcji bazowych
figure; plot(y,'bo-'); title('y(n)'); grid;      % rysunek sygnalu wyjsciowego
error = max(abs(x.'-y)), % blad odtworzenia/rekonstrukcji sygnalu
```

Problem 3.5 (* Ortogonalna dekompozycja i rekonstrukcja sygnału z użyciem DCT-IV). Uruchom program z listingu 3.2. Wybierz po kolei różne sygnały wejściowe jako `x`, tak jak zasugerowano w programie. Obserwuj: 1) kształt wejściowego/analizowanego sygnału `x`, 2) kształt wyjściowego/zrekonstuowanego sygnału `y` oraz 3) wartość błędu pomiędzy tymi sygnałami. Pokaż na rysunku i przeanalizuj wartości współczynników transformaty `c`. Spróbuj odczytać z nich informację dotyczącą analizowanego sygnału `x`: jakie funkcje bazowe transformacji DCT-IV zawiera sygnał i w jakiej "ilości"? Jaka jest ich amplituda, czyli współczynnik skalujący (*wsp. podobieństwa*)? Zauważ, że:

1. sygnały `x1` i `x2` są kolumnami macierzy `S` i wynik analizy jest perfekcyjny,
2. sygnał `x3` nie występuje w macierzy `S` - jego "częstotliwość" jest inna: 10.5 zamiast ...,9,10,11,12..., dlatego wynik analizy jest rozmyty,
3. sygnał `x4` nie występuje w macierzy `S` - ma on taką samą "częstotliwość" 10 jak `x2`, ale jest przesunięty w czasie o `N/4` próbek i dlatego wynik analizy jest rozmyty,
4. sygnał `x5` to szum zawierający wszystkie składowe częstotliwościowe.

Problem 3.6 (* Zwartość/kompaktowość (*compactness*) wyniku transformacji ortgonalnej). W programie 3.2 ustaw kolejno `x` równe: `x1, x2, x3, x4`. Zauważ, że pierwsze dwa widma DCT (zbiór współczynników c_k) są bardzo "ostre", ponieważ sygnały są dokładnie równe wektorom bazowym transformacji: 5-temu i 10-temu. Natomiast widmo sygnału `x3` jest "rozmyte", ponieważ jego "częstotliwość" 10.5 nie jest reprezentowana w wierszach macierzy analizy `A` transformaty (i kolumnach macierzy syntezy `S`). Co jednak nie przeszkadza w perfekcyjnym odtworzeniu sygnału `x3`. Z kolei widmo DCT sygnału `x4` także jest rozmyte, ale tym razem powód jest inny: sygnał ma częstotliwość rezprezentowaną w bazie wzorców (10), ale jest przesuniety w czasie o `N/4` próbek.
Wybierz sygnał `x3`. Uruchom program wielokrotnie, zmieniając wartość zmiennej `k`, czyli parametru częstotliwości: 9; 9,5; 10; 10,25; 10.5; 10,75; 11; ... Dla `k=9, 10, 11` powinieneś otrzymać "ostre" widma DCT.
Wybierz sygnał `x4`. Uruchom program wielokrotnie, zmieniając wartość przesunięcia zmiennej `n'`, czyli parametru czasu: 0; 0,5; 1; 10; `N/4`. Dla liczby 0 powinieneś otrzymać "ostre" widmo DCT.
Wniosek: widmo transformaty najczęściej będzie nieostre, gdyż częstotliwości i przesunięcia czasowe składowych analizowanego sygnału są dowolne, co gorsza: NIEZNANE! Ale jak się nie ma tego co się lubi ..., to trzeba się nauczyć żyć z niedoskonałościami transformacji ortogonalnych i odpowiednio je interpretować (tak jak lekarz umie ocenić stan pracy serca na podstawie sygnału EKG).

Problem 3.7 (* Separacja składowych sygnału oraz odszumianie sygnału w dziedzinie współczynników transformacji DCT-IV). Kontynuujemy ostatni problem. Kiedy sygnał `x` jest sumą kilku składowych oraz `x1` jest jednym ze składników tej sumy, ustaw (dwie różne opcje):
1) `c(5)=0`, czyli wyzeruj 5-ty współczynnik `c`, odpowiadający sygnałowi `x1`,
2) `c([1:4,6:N])=zeros(1,N-1)`, czyli wyzeruj wszystkie współczynniki oprócz 5-tego.
Obserwuj kształt zsyntezowanego sygnału `y`. W pierwszym przypadku, składowa `x1` powinna zostać usunięta z sygnału wynikowego, zaś w drugim przypadku - pozostawiona tylko ona. Kiedy w sygnale `x` występuje składowa `x1` i szum `x4`, wynikowy sygnał `y` w opcji 2) jest odszumionym sygnałem `x1`. W tym przypadku staramy się ustawić na wartości zerowe (`c(k)=0;`) wszystkie współczynniki transformaty, związane tylko z szumem. Usunięcie (poprzez wyzerowanie odpowiedniego współczynnika `c(k)=0;`) albo pozostawienie jakiejś składowej w sygnale sumarycznym (poprzez nie zmianianie wartości jej współczynnika) jest prostsze i efektywniejsze, jeśli energia tej składowej jest skoncentrowana w bardzo małej liczbie współczynników transformaty. Czyli jest to proste dla składowych `x1` oraz `x2`, a trudne dla składowych `x3, x4`, gdyż "obraz" tych składowych jest rozmyty w wyniku transformaty.

Problem 3.8 (* Poprawa SNR (stosunku energii sygnału do szumu) dzięki filtracji sygnału). Kontunuujemy ostatni problem. Skorzystaj z definicji SNR, podanej w laboratorium 2. Oblicz SNR dla zaszumionego sygnału analizowanego `x=x1+x5` oraz dla odszumionego sygnału zrekonstruowanego `y`, w przybliżeniu `x1`.

TEMAT #3: Zastosowania. Dzięki zastosowaniu ortogonalnych transformacji sygnału, prostej i odwrotnej, możemy: 1) sprawdzić jakie sygnały elementarne zawiera nasz sygnał (o czym informuje nas wartość obliczonych współczynników podobieństwa analizowanego sygnału do funkcji bazowych), 2) zmodyfikować współczynniki podobieństwa i wykonać transformację odwrotną; dzięki temu możemy odseparować od siebie różne składowe sygnału, w tym odszumić sygnał.

Problem 3.9 (Wyznaczenie częstotliwości oscylacji strun głosowych).** Nagraj samogłoski (a,e,i,o,u) swojej własnej mowy z częstotliwością próbkowania 8000 próbek na sekundę (sps - *samples per second*). Narysuj sygnał. Wzrokowo znajdź okres powtarzania się sygnału `T` (w milisekundach). Potem oblicz DCT stacjonarnego fragmentu sygnału, korzystając z funkcji Matlaba `dct()` oraz wyświetl obliczone współczynniki transformaty `c`:

```
(n1=?; Ndct=2^13; c=dct( x(n1:n1+Ndct-1),'Type',4);
plot(c,'.-');
```

Znajdź wzrokowo pierwsze duże maksimum w wektorze `c` oraz jego indeks (argument). Następnie spróbuj wegerować kosinusoidę o częstotliwości `f0`, mającą maksimum w tym samym miejscu w wektorze `c` (przypomnienie: już generowaliśmy elementarne funkcje bazowe transformacji DCT-IV). Teoretycznie, `f0=1/T`, gdzie wartość `T` - znaleziona wzrokowo - jest równa okresowi oscylacji strun głosowych. Czy widzisz inne maksima w wektorze `c`? Składowe mające częstotliwości `k*f0, k=2,3,4,...`, wielokrotności `f0`, powinny także występować w sygnale i być widoczne w jego widmie.

Problem 3.10 (Dekompozycja dźwięku na składowe częstotliwościowe).** Znajdź w internecie dowolne nagrania "wibrującego" dźwięku (na przykład skorzystaj ze strony *FindSounds* nagrania wycia psa/wilka, śpiewu ptaków, jednego akordu instrumentu muzycznego lub warkotu maszyny/silnika). Sygnał powinien trwać około 3 sekundy. Pobierz je i odsłuchaj (`soundsc(x,fpr)`). Następnie, oblicz współczynniki przedstawienia sygnału jako sumy kosinusów, używając funkcji Matlaba DCT: `c=dct(x)`. Wyświetl wartości współczynników transformaty `stem(c)`. Następnie zsyntezuj dźwięk używając `K` największych współczynników `c(k)`, systematycznie zwiększając wartość `K`=1, 2, 3, ..., 10, ..., 100,..., 1000, ..., `N=length(c)`. Odtwórz sygnał. Jak sam słyszysz, składowe kosinusoidalne mające duże wartości współczynników skalujących są najbardziej istotne.

Problem 3.11 (Filtracja dźwięku z użyciem dekompozycji sygnału na składowe oraz powrotnej syntezy z użyciem transformat DCT/IDCT).** Nagraj 3-4 sekundy fragmentu swojej własnej mowy z częstotliwością próbkowania `fs=8000` Hz, czyli jedno zdanie. Narysuj (`plot(x)`) i odsłuchaj sygnal (`soundsc(x,fpr)`). Wykonaj transformację DCT całego sygnału `c=dct(x)` lub jego części `c=dct(x(n1:n2))`. Wyświetl obliczone współczynniki transformaty `stem(c)`. Następnie dokonaj syntezy mowy z: 1) 25% wszystkich współczynników - wybierz pierwsze wartości, np. `y=idct([c(1:8000); zeros(lenght(c)-8000,1)])`, 2) 75% wszystkich współczynników - wybierz ostatnie wartości. Odsłuchaj wynik przetwarzania. Przykład 1: wszystkie wartości współczynników mniejsze od 50 możesz wyzerowaj w ten sposób: `c(c<50)=0`. Przykład 2: możesz wyzerować tylko współczynniki o indeksach od 100 do 200: `c([100:200])=0`.
Teraz dodaj zakłócenie sinusoidalne o częstotliwości 250 Hz do nagranej mowy:

```
x=x+0.5*sin(2*pi*250/fpr*(0:length(x)-1)')
```

(zwróć uwagę na orientację wektora, poziomy czy pionowy). Narysuj sygnał, odsłuchaj go, wykonaj DCT, zmodyfikuj współczynniki DCT (usuń zakłócenie), wykonaj odwrotne DCT, narysuj sygnał, odsłuchaj wynik przetwarzania.

Laboratorium 4
Dyskretne Transformacje Fouriera: DFT and DtFT

Streszczenie Na tym laboratorium, jednym z najważniejszych w całym module, nauczymy się obliczać Dyskretną Transformację Fouriera (DFT), czyli dyskretną wersję szeregu Fouriera, oraz Dyskretną-w-Czasie Transformację Fouriera (DtFT), czyli zdyskretyzowaną ciągłą transformację Fouriera. Poznamy rolę funkcji tzw. *okien* w analizie częstotliwościowej. Sprawdzimy jakie są widma fourierowskie różnych sygnałów oraz jak różne operacje na sygnałach zmieniają ich widma.

TEMAT #1: Dyskretna Transformacja Fouriera (DFT). DFT jest najbardziej znanym przykładem ortogonalnych transformacji sygnału, którymi zajmowaliśmy się w poprzednim laboratorium. Wykorzystuje ona zespolone funkcje harmoniczne $v_k(t) = e^{j2\pi f_k t} = \cos(2\pi f_k t) + j\sin(2\pi f_k t)$ jako elementarne funkcje bazowe transformacji. Analizowany sygnał o czasie trwania $t = [0, T]$ jest przedstawiany jako suma przeskalowanych funkcji bazowych (pierwszy wzór), o współczynnikach równych wartościom iloczynów skalarnych analizowanego sygnału i poszczególnych funkcji bazowych (drugi wzór):

$$x(t) = \sum_{(k)} c_k v_k(t), \qquad c_k = \frac{1}{T}\int_{t=0}^{T} v_k(t)x(t)dt. \tag{4.1}$$

Funkcje te są zespolonymi oscylacjami o różnych częstotliwościach f_k. Częstotliwości te nie są dowolne: są wielokrotnościami częstotliwości podstawowej $f_0 = \frac{1}{T}$. Jeśli:

1. wykonamy dyskretyzację funkcji bazowych $v_k(t)$ w osi czasu z użyciem okresu próbkowania $\Delta t = \frac{1}{f_{pr}}$ i pobierzemy tylko N pierwszych próbek tych funkcji: $t = t_n = n \cdot \Delta t,\ n = 0, 1, 2, ..., N-1$;
2. uwzględnimy tylko N pierwszych wielokrotności $f_0 = \frac{1}{N \cdot \Delta t}$, czyli: $f_k = k \cdot f_0,\ k = 0, 1, 2, ...N-1$,

to otrzymujemy następującą ortogonalną macierz analizy DFT ($\mathbf{A}$) oraz macierz macierz syntezy DFT ($\mathbf{S}$) (zauważ, że $\mathbf{A} = \mathbf{S}^{*T}$):

$$A(k,n) = v_k^*(t_n) = \frac{1}{\sqrt{N}} \cdot e^{-j2\pi\left(k\frac{1}{N\cdot\Delta t}\right)(n\cdot\Delta t)} = \frac{1}{\sqrt{N}} \cdot e^{-j\frac{2\pi}{N}kn}, \quad k,n = 0,1,2,...,N-1, \tag{4.2}$$

$$S(n,k) = v_k(t_n) = \frac{1}{\sqrt{N}} \cdot e^{j2\pi\left(k\frac{1}{N\cdot\Delta t}\right)(n\cdot\Delta t)} = \frac{1}{\sqrt{N}} \cdot e^{j\frac{2\pi}{N}kn}, \quad k,n = 0,1,2,...,N-1, \tag{4.3}$$

Analiza DFT i synteza IDFT sygnału są wówczas opisane następującymi wzorami macierzowymi:

$$\mathbf{c} = \mathbf{A} \cdot \mathbf{x}, \qquad \mathbf{x} = \mathbf{S} \cdot \mathbf{c}. \tag{4.4}$$

Dla sygnałów o wartościach rzeczywistych równania DFT i IDFT mogą zostać zapisane w bardziej zrozumiałej postaci z wykorzystaniem notacji *trygonometrycznego* szeregu Fouriera (f_{pr} - częstotliwość próbkowania, $k = 0, 1, 2, ..., \frac{N}{2} - 1$):

$$a_k = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\cos\left(2\pi\frac{kf_0}{f_{pr}}n\right), \quad b_k = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\sin\left(2\pi\frac{kf_0}{f_{pr}}n\right) \tag{4.5}$$

$$x(n) = a_0 + 2\sum_{k=1}^{N/2-1}\left[a_k\cos\left(2\pi\frac{kf_0}{f_{pr}}n\right) + b_k\sin\left(2\pi\frac{kf_0}{f_{pr}}n\right)\right] = \tag{4.6}$$

$$= c_0 + 2\sum_{k=1}^{N/2-1}|c_k|\cos\left(2\pi\frac{kf_0}{f_{pr}}n + \sphericalangle c_k\right)$$

$$c_k = a_k - jb_k, \quad |c_k| = \sqrt{a_k^2 + b_k^2}, \quad \sphericalangle c_k = \mathrm{arctg}\left(\frac{-b_k}{a_k}\right)$$

Problem 4.1 (*(plus * dodatkowo) Macierze DFT). Zmodyfikuj program 3.2: dodaj do niego generację macierzy `A` oraz `S=A'` ortogonalnej transformacji DFT:

```
k = (0:N-1); n=(0:N-1); % wiersze=funkcje, kolumny=probki
A = sqrt(1/N)*exp(-j*2*pi/N*k'*n); % macierz analizy
```

Skopiuj fragment programu 3.1 i także dodaj do programu 3.2: sprawdź ortonormalność kilku wierszy wygenerowanej macierzy `A`. Sprawdź czy spełniony jest warunek ortonormalności macierzy: $\mathbf{S}\cdot\mathbf{A} == \mathbf{I}$, (`S*A=A'*A=I`). Wyświetl po kolej na rysunku wartości pierwszych 10-ciu wierszy macierzy `A`: na górze - część rzeczywistą (funkcje kosinus?), na dole — część urojoną (funkcje minus sinus?). Następnie porównaj części rzeczywiste i urojone następujących par wierszy macierzy: 2-iego oraz `N`-tego, 3-ciego oraz `N-1`-szego, 4 oraz `N-2`-iego,... Jaki wniosek można wyciągnąć z tego porównania? Części rzeczywiste wierszy są identyczne, a urojone - zanegowane? Czy mógłbyś udowodnić swoje spostrzeżenie matematycznie? Jeśli tak, to otrzymasz dodatkowy punkt!

Problem 4.2 (* Wprowadzenie do DFT: DFT sygnału sinusa lub kosinusa). Teraz sprawdź wynik transformacji DFT dla różnych sygnałów wejściowych. Użyj zmodyfikowanego programu z poprzedniego problemu. Niech analizowany sygnał będzie liniową kombinacją (sumą ważoną) różnych sygnałów wejściowych, np. `x1,x2`. Rozpocznij od ustawienia: `x=x1` oraz `x=x2;`. Pomnóż wektor `x` przez macierz transformaty DFT `A`. Zaobserwuj jaki kształt ma wektor `stem(c)`, będący wynikiem transformacji. Czy jego wartości informują nas o wartościach amplitud i częstotliwości sygnałów składowych (harmonicznych) analizowanego sygnalu `x`? Potem ustaw: `x=real(x1)`, `x=imag(x1)`. Dlaczego widmo DFT sygnału jest symetryczne w części rzeczywistej dla sygnału `x=real(x1)` oraz asymetryczne w części urojonej dla sygnału `x=imag(x1)`? Dlaczego moduły współczynników transformaty są w obu przypadkach dwa razy mniejsze niż amplituda analizowanych sygnałów o kształcie kosinusa/sinusa? Czy nie ma to związku z równością: $\cos(\alpha) = 0.5e^{j\alpha} + 0.5e^{-j\alpha}$? A jak jest dla funkcji sinus? Teraz wykonaj DFT dla sygnału `x=real(x2)` oraz `x=imag(x2)`. Wnioski z rusunku? Na koniec oblicz DFT dla sumy sygnałów `x=x1+x2;`. Wnioski? Ciąg dalszy nastąpi ...

Problem 4.3 (* Meritum DFT - poprawne skalowanie i interpretacja). Użyj programu z poprzedniego problemu. Wygeneruj następujący sygnał:

```
fs=1000; dt=1/fs; T=N*dt; f0=1/T;
t=dt*(0:N-1); x5=1*cos(2*pi*(10*f0)*t);
```

mający amplitudę `A=1` oraz częstotliwośc `10*f0`. Wyskaluj poprawnie wartości współczynników widmowych `c` (widmo DFT, w pionie) oraz dodaj opis osi częstotliwości (poziomej), zakładając że częstotliwość próbkowania f_{pr} jest znana: $f_0 = \frac{1}{T} = \frac{f_{pr}}{N}$:

```
cs = sqrt(1/N)*c;
fk = f0*(0:N-1);
figure; subplot(211); stem(fk,real(cs)); subplot(212); stem(fk,imag(cs));
figure; subplot(211); stem(fk,abs(cs)); subplot(212); stem(fk,angle(cs));
```

Tak jak się spodziewaliśmy, widmo DFT naszego nowego sygnału jest symetryczne wokół częstotliwości $\frac{f_{pr}}{2}$ w części rzeczywistej (dla kosinusów) oraz asymetryczne (symetryczne zanegewane) w części urojonej (dla sinusów).

Przeanalizuj kształt widma DFT, wyświetl `stem(fk,real(cs))` (na górze) oraz `stem(fk,imag(cs))` (na dole). Jaki wnioski możesz wyciągnąć, dotyczące amplitudy i częstotliwości analizowanego sygnału? Następnie zmień wejściową funkcję `cos()` na `sin()` w definicji sygnału, oraz powtórz analizę sygnału. Twoje wnioski? Teraz zmień `(10*f0)` na `(10.5*f0)` oraz powtórz analizę dla wejściowych sygnałów `cos()` oraz `sin()`. Wnioski? Dlaczego wartości bezwzględne (moduł liczb zespolonych) współczynników widmowych `cs` są dwa razy mniejsze niż spodziewane, biorąc pod uwagę amplitudę analizowanego sygnału?

TOPIC #2: Filtracja sygnałów za pomocą współczynników DFT. Sekwencję operacji: 1) DFT, 2) modyfikacja widma, 3) odwrotne DFT (IDFT) (czyli: znajdź przepis na zupę, zmodyfikuj go i ugotuj nową zupę według zmienionego przepisu), można z powodzeniem wykorzystać do usunięcia z sygnału składowych o "niechcianych" częstotliwościach. Robiliśmy już tak w przypadku transformaty DCT+IDCT w poprzednim laboratorium (transformacje ortogonalne). Idea jest ta sama, jednak teraz należy pamietać o tym, żeby z sygnału usunąć dwie "symetryczne" składowe: o częstotliwości dodatniej i ujemnej. For example, in Matlab: `(c(2)=c(N)=0),(c(3)=c(N-1)=0),....`

Problem 4.4 (* DFT-IDFT #1: Ene due rabe, połknął bocian żabę). Zmodyfikuj program 3.2: dodaj do niego generację macierzy `A` oraz `S=A'` ortogonalnej transformacji DFT, tak jak w problemie/zadaniu 4.1. Wybierz opcję `x=real(x1)+imag(x2)`. Następnie wyzeruj w widmie DFT współczynniki związane z sygnałem: 1) `x1`, albo 2) `x2`. Oblicz IDFT oraz na jednym rysunku wyświetl sygnał `x` oraz otrzymany. Porównaj otrzymany sygnał z oryginałem, np. oryginalny i odzyskany z sumy sygnał `real(x1)`.

Problem 4.5 (IDFT-IDFT #2: un due trois, żaba połknęła bocianà).** Nagraj $N = 2^{15}$ próbek swojej własnej mowy z częstotliwością próbkowania 8000 Hz (użyj programu z laboratorium #1). Wygeneruj sygnał o takiej samej liczbie próbek z sinusoidalną modulacją częstotliwości (SFM): $f_0 = 2500$ Hz, $df = 500$ Hz, $f_m = 0.25$ s (patrz laboratorium #2) i dodaj go do sygnału mowy. Oblicz i wyświetl widmo DFT obu synałów oraz sygnału sumarycznego: zobacz które prążki DFT "okupuje" mowa, a które zakłócenie. Wyzeruj współczynniki DFT związne z zakłóceniem (w widmie sumy sygnałów) i oblicz IDFT. Narysuj i odsłuchaj otrzymany sygnał.

TEMAT #3: Rola funkcji okien podczas analizy DFT sygnałów - wprowadzenie. Najbardziej intuicyjne wytłumaczenie znaczenia funkcji/sygnału *okna* jest następujące. Kiedy nasz sygnał jest sumą różnych przeskalowanych sinusoid/kosinusoid, to jego widmo DFT powinno być prążkowe (pojedyncze niezerowe wartości), ponieważ tylko składowe o określonych częstotliwościach występują w sygnale. Ale tak nie jest, ponieważ nasz sygnał jest pomnożony przez funkcję okna obserwacji, która wycina z niego analizowany fragment. Nawet jeśli wydaje nam się, że okna nie używany, to stosujemy okno prostokątne (czyli same jedynki, które po pomnożeniu przez sygnał nie modyfikują oryginalnych próbek sygnału). Funkcja *okna* jest sygnałem, który też ma widmo DFT, łatwo to sprawdzić. Widmo DFT sygnału sumy przeskalowanych sinusów/kosinusów jest równe sumie analogicznie przeskalowanych widm DFT użytego okna, przesuniętych do częstotliwości sinusów/kosinusów, występujących w sygnale (symetrycznie "scentrowanych" na tych częstotliwościach). Widmo DFT okna ma tzw. listek główny, czyli wysoką "górkę" dla 0 Hz, oraz tzw. listki boczne, czyli niżej leżące oscylacje boczne. Kiedy częstotliwości dwóch składowych sygnału mało się różnią, sąsiednie "górki" okna mogą się zlać w jedną szeroką "górę" i nie będziemy w stanie widzeć/rozróżnić dwóch częstotliwości (zła rozdzielczość częstotliwościowa). Kiedy amplitudy sygnału bardzo się różnią, "górka" mniejszej z nich może "utonąć" w oscylacjach bocznych "górki" większej i być niewidoczna (zła rodzielczość amplitudowa). Przykłady funkcji okien, w kolejności coraz szerszych "górek" oraz coraz niżej leżących oscylacji bocznych: prostokatne (`boxcar(N)`), trójkątne (`bartlett(N)`), Hamminga (`hamming(N)`, Hanninga (`hanning(N`, Blackmana (`blackman(N)`. Okna Kaisera (`kaiser(N,beta)` oraz Czebyszewa (`chebwin(N,Rs)` są "elastyczne": mają zmienne kształty i widma zależne od dodatkowego parametru, dlatego ich użycie jest polecane.
Na rysunku 4.1 graficznie pokazano znaczenie zastosowania funkcji okna, w tym przypadku prostokątnego, podczas analizy częstotliwościowej sygnałów. Więcej szczegółów poznamy podczas realizacji poszczególnych zadań problemu 4.7, należącego do tematu #4.

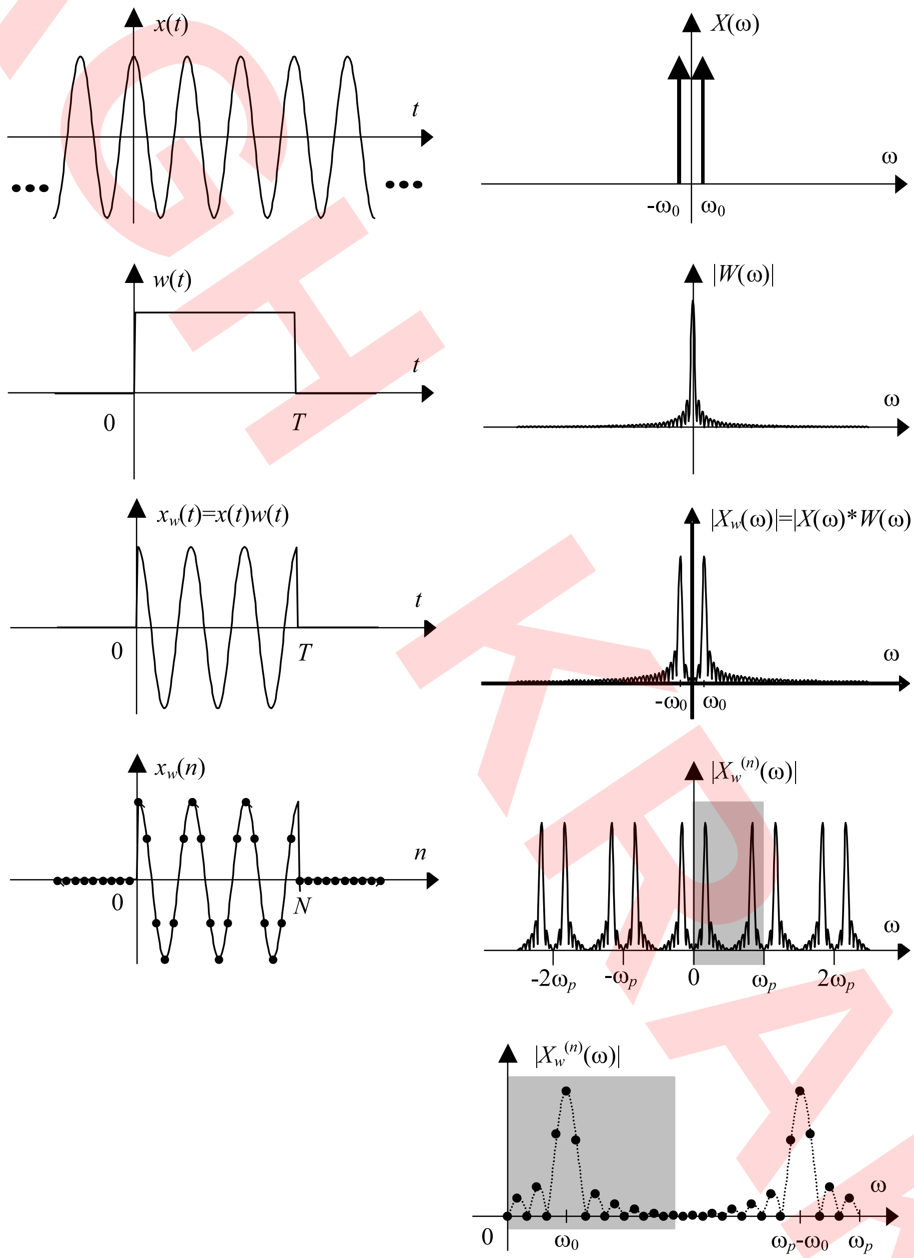

Fig. 4.1: Graficzna ilustracja podstawowych "prawd" analizy częstotliwościowej sygnałów: (po lewej) sygnały, (po prawej) ich widma częstotliwościowe. W kolejnych wierszach: 1) nieskończony-w-czasie kosinus i jego teoretyczne widmo Fouriera: dwa impulsy wynikające z równania $cos(\omega t) = 0.5exp(-j\omega t) + 0.5exp(+j\omega t)$, jeden dla częstotliwości ujemnej, a drugi dla dodatniej, 2) okno prostokątne i jego teoretyczne widmo Fouriera o kształcie typu $\frac{sin(\omega)}{\omega}$ (pokazana wartość |.|), 3) wynik pomnożenia kosinusa i okna prostokątnego oraz jego widmo (splot widm pokazanych powyżej, oznaczony przez "*"), 4) spróbkowany sygnał i jego widmo, powtarzające się okresowo co f_{pr} - konsekwencja twierdzenia o próbkowaniu: wzorce częstotliwości mają za wysoką częstotliwość w stosunku do częstotliwości próbkowania sygnału, 5) spróbkowany jeden okres powtarzającego się widma - patrz rys. 4.16, 4.17, 8.6, 8.7 w [40]

Problem 4.6 (Znaczenie mnożenia sygnału przez funkcję okna).** Użyj programu z listingu 4.1. Na początku, zwróć przede wszystkim uwagę, że widma `X1 = A*x.'` oraz `X2 = fft(x.')` są identyczne, tzn. funckja Matlaba `X=fft(x)` oblicza DFT sygnału. Dlatego w problemach dot. TEMATU 6 (sygnały rzeczywiste!) staraj się wykorzystywać funkcję `fft()`, ponieważ jest ona szybka. Ustaw `w=w1` oraz uruchom program dla `x=x1, x2, x3,` `x1+x3, x2+x3`. Obejrzyj wszystkie rysunki. Jakie wartości błędów są wyświetlane? Następnie ustaw `w=w2` oraz ponownie wykonaj program. Co powoduje, że widma DFT sygnałów `x1` i `x2` tak bardzo się różnią? Dlaczego sygnał `x3` jest niewidoczny w widmie DFT sygnałów `(x1+x3)` oraz `(x2+x3)` w przypadku użycia okna `w1`, natomiast jest widoczny w przypadku użycia okna `w2`? Ustaw `x=x1+x3, w=w1` oraz obejrzyj zrekonstuowany sygnał `y`. Czy jest podobny do sygnału `x`? Ponownie uruchom program lecz tym razem wykonaj także linię programu: `X(1+10)=0; X(N-10+1)=0;` Obejrzyj sygnał `y`. Porównaj go z sygnałem `x3`. Wytłumacz dlaczego `y` jest taki sam jak `x3`.

Listing 4.1: Analiza DFT sygnału w programie Matlab

```
% cps_04_dft.m

clear all; close all
% Macierze transformacji DFT
N = 100;                    % wymiar macierzy
k = (0:N-1); n=(0:N-1);     % wiersze=funkcje/sygnaly bazowe, kolumny=probki
A = exp(-j*2*pi/N*k'*n);    % macierz analizy DFT, z innym skalowaniem niz poprzednio
S = A';                     % macierz syntezy DFT: sprzezenie + transpozycja
% S = exp(j*2*pi/N*n'*k); A = S';  % z lab. o transformatach ortogonalnych
% diag(S*A), pause          % sprawdzenie ortogonalnosci macierzy, N na przekanej

% Signal
fs=1000; dt=1/fs; t=dt*(0:N-1).';  % skalowanie osi czasu, czas pionowo!
T=N*dt; f0=1/T; fk = f0*(0:N-1);   % fskalowanie osi czestotliwosci
x1 = 1*cos(2*pi*(10*f0)*t);        % sygnal 1
x2 = 1*cos(2*pi*(10.5*f0)*t);      % sygnal 2
x3 = 0.001*cos(2*pi*(20*f0)*t);    % sygnal 3
x = x1;                            % wybor: x1, x2, x3, x1+x2, x2+x3

% Funkcja "okna"
w1 = boxcar(N);                    % okno prostokatne, N - dlugosc
w2 = chebwin(N,100);               % okno Czebyszewa, liczba - poziom listkow bocznych
w = w1; scale = 1/sum(w);          % wybor: w1, w2 albo inne, dodane okno

% Windowing
x = x.*w;                          % "okienkowanie" sygnalu

% DFT of the signal
X1 = A*x;                          % nasz kod DFT
X2 = fft(x);                       % funkcja Matlaba DFT (Fast Fourier Transform)
error1 = max(abs(X1-X2)),          % blad wzgledem Matlaba, powinno byc prawie zero
X = X2;                            % wybor: X1 albo X2

% Interpretacja widma DFT, skalowanie
X = scale*X;                       % skalowanie amplitudy
figure;
subplot(211); plot(fk,real(X),'o-'); title('real(X(f))'); grid;
subplot(212); plot(fk,imag(X),'o-'); title('imag(X(f))'); grid;
figure;
subplot(211); plot(fk,20*log10(abs(X)),'o-'); title('abs(X(f)) [dB]'); grid;
subplot(212); plot(fk,angle(X),'o-'); title('angle(X(f)) [rad]'); grid;

% Modyfikacja widma DFT - przyklad
% X(1+10)=0; X(N-10+1)=0;          % usuniecie sygnalu x1 o czestotliwosci 10*f0

% Odwrotne DFT - synteza sygnalu na podstawie jego widma
y = S*X;                           % synteza sygnalu
error2 = max(abs(x-y)),            % blad odtworzenia sygnalu
figure;                            % wynik
subplot(211); plot(real(y),'bo-'); title('real(y(n))'); grid;
subplot(212); plot(imag(y),'bo-'); title('imag(y(n))'); grid;
```

TEMAT #4: Związek pomiedzy DFT a DtFT. Szczegółowo o widmach sygnałów i funkcjach okien. Dyskretna Transformacja Fouriera DFT (analiza sygnału) oraz odwrotna dyskretna transformacja Fouriera (IDFT) (synteza sygnału) są wynikiem dyskretyzacji szeregu Fouriera, stosowanego dla ciągłych sygnałów okresowych. Definicje DFT i IDFT są następujące (czerwonym kolorem dla porównania jest podana definicja szeregu Fouriera dla sygnałów ciągłych):

$$\textbf{Analiza DFT}: \quad c_k = \frac{1}{T}\int_{t=0}^{T} x(t) e^{-j2\pi(kf_0)t} dt, \qquad X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \quad k = 0,\ 1,\ 2, ..., N-1, \tag{4.7}$$

$$\textbf{Synteza DFT}: \quad x(t) = \sum_{k=-\infty}^{+\infty} c_k \cdot e^{+j2\pi(kf_0)t} dt, \qquad x(n) = \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn}, \quad n = 0,\ 1,\ 2, ..., N-1. \tag{4.8}$$

Z kolei Dyskretna-w-Czasie Transformacja Fouriera (DtFT)oraz jej odwrotność (IDtFT) są wynikiem dyskretyzacji ciągłej transformacji Fouriera. Definiowane są one przez następujące równania (czerwonym kolorem dla porównania jest podana definicja ciągłej transformacji Fouriera dla sygnałów ciągłych):

$$\textbf{Analiza DtFT:} \quad X(f) = \int_{t=-\infty}^{+\infty} x(t) e^{-2\pi f t} dt \qquad X\left(\frac{f}{f_{pr}}\right) = \sum_{n=-\infty}^{+\infty} x(n) e^{-j2\pi\frac{f}{f_{pr}}n} = \sum_{n=-\infty}^{+\infty} x(n) e^{-j\Omega n}, \tag{4.9}$$

$$\textbf{Synteza DtFT:} \quad x(t) = \int_{f=-\infty}^{+\infty} X(f) e^{2\pi f t} df \qquad x(n) = \frac{1}{f_{pr}} \int_{-f_{pr}/2}^{+f_{pr}/2} X\left(\frac{f}{f_{pr}}\right) \cdot e^{j2\pi\frac{f}{f_{pr}}n} df \tag{4.10}$$

Kiedy dysponujemy tylko N próbkami sygnału oraz dzielimy wynik transformacj "w przód" przez N, równanie (4.9) przyjmuje następującą postać:

$$\textbf{Analiza DtFT:} \quad X\left(\frac{f}{f_{pr}}\right) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-j2\pi\frac{f}{f_{pr}}n}, \quad -f_{pr}/2 \le f < f_{pr}/2. \tag{4.11}$$

która jest bardzo podobna do równania (4.7). Analiza DtFT umożliwia nam bardziej gęste próbkowanie widma sygnału w wybranym przez nas zakresie częstotliwości.
Poniżej będziemy się zajmowali wyłącznie analizą DFT i DtFT, czyli stosowali równania (4.7)(4.11), w stosunku do sygnału oryginalnego $x(n)$ oraz sygnału $x(n)$ pomnożonego przez funkcję okna $w(n)$: $x_w(n) = x(n)w(n)$.

Listing 4.2: Program Matlaba do porównania DFT i DtFT

```
% cps_04_dtft.m
clear all; close all;

N = 100;                        % liczba probek sygnalu: 100 ---> 1000
fpr = 1000; dt=1/fpr; t=dt*(0:N-1); % fpr to liczba probek sygnalu na sekunde
df = 10;                        % krok w czestotliwosci [Hz] dla DtFT: 10 ---> 1
fmax = 2.5*fpr;                 % max zakres czestotliwosci w DtFT: 2.5 ---> 0.5
fx1 = 100;                      % czestotliwosc skladowej 1 sygnalu
fx2 = 250;  Ax2 =0.001;         % czestotliwosc i amplituda skladowej 2 sygnalu
                                % 250 ---> 110, 0.001 ---> 0.00001
% Sygnal
x1 = cos(2*pi*fx1*t);           % pierwsza skladowa
x2 = Ax2*cos(2*pi*fx2*t);       % druga: 250Hz ---> 110Hz, 0.001 ---> 0.00001
x = x1; % + x2;                 % wybor: x1, x1+x2, 20*log10(0.00001)=-100 dB
stem(x); title('x(n)'); pause   % rysunek analizowanego sygnalu

% Funkcje okien
w1 = boxcar(N)';                % okno prostokatne
w2 = hanning(N)';               % okno Hanninga
```

```
w3 = chebwin(N,140)';              % okno Czebyszewa, 80, 100, 120, 140
w = w1;                            % w1 --> w2, w3 (80, 100, 120, 140)
stem(w); title('w(n)'); pause      % rysunek okna
x = x .* w;                        % wybor: x = x, w, x.*w
stem(x); title('xw(n)'); pause     % "zokienkowany" sygnal

% DFT - czerwone kolka
% k=0:N-1; n=0:N-1; F = exp(-j*2*pi*(k'*n)); X = (1/N)*F*x;
f0 = fpr/N; f1 = f0*(0:N-1);  % krok df w DF = f0 = 1/(N*dt)
for k = 1:N
    X1(k) = sum( x .* exp(-j*2*pi/N* (k-1) *(0:N-1) ) )/ N;
  % X1(k) = sum( x .* exp(-j*2*pi/N* (f1(k)/fpr) *(0:N-1) ) )/ N;
end
%X1 = N*X1/sum(w);                 % poprawne skalowanie dla dowolnego okna

% DtFT - niebieska linia
f2 = -fmax : df : fmax;            % zakres czestotliwosci: od-krok-do, df=10-->1 first this freq. range
for k = 1 : length(f2)
    X2(k) = sum( x .* exp(-j*2*pi* (f2(k)/fpr) *( 0:N-1) ) ) / N;
end
%X2 = N*X2/sum(w);                 % poprawne skalowanie dla dowolnego okna

% Figures
figure; plot(f1,abs(X1),'ro',f2,abs(X2),'b-');
xlabel('f (Hz)'); grid; pause
figure; plot(f1,20*log10(abs(X1)),'ro',f2,20*log10(abs(X2)),'b-');
xlabel('f (Hz)'); grid; pause
```

Problem 4.7 (** DtFT sygnału kosinusoidalnego z dowolnym oknem - wspinaczka krok-po-kroku).** Rzuć okiem na rysunek 4.1, aby szybko zorientować się w "co teraz gramy". Staraj się odnieść wyniki, które uzyskasz poniżej, do tych które są przedstawione na tym rysunku. Użyj programu 4.2 w celu obliczenia widma DtFT sygnału kosinusoidalnego. Ustaw $f_{x1} = 100$ Hz jako częstotliwość sygnału oraz $f_s = 1000$ Hz jako częstotliwość próbkowania. Wygeneruj `N=100` próbek sygnału. Ustaw `x=x1`.

1. **(1/2*)** Przeanalizuj sygnał za pomocą DtFT w zakresie częstotliwości $[-f_{max}, ..., f_{max}]$, $f_{max} = 2.5 f_s$ z krokiem częstotliwościowym $df = 10$ Hz, równym krokowi DFT $f_0 = \frac{f_{pr}}{N}$. Spodziewamy się otrzymać ostre "górki" dla częstotliwości $f = -100$ Hz oraz $f = 100$ Hz, ponieważ po dyskretyzacji nasz sygnał ma postać:
$$\cos\left(2\pi \frac{f_{x1}}{f_s} n\right) = \frac{e^{j2\pi \frac{f_{x1}}{f_s} n} + e^{-j2\pi \frac{f_{x1}}{f_s} n}}{2}. \tag{4.12}$$
Dlaczego widzimy tylko dwie "górki" w widmie DFT oraz o wiele więcej "górek" w widmie DtFT, powtarzających się okresowo co częstotliwość próbkowania f_s? Ponieważ widmo DFT jest obliczane tylko w zakresie $[0, f_s)$ z krokiem f_0: dlatego widzimy tylko dwie składowe harmoniczne (zespolone) kosinusa o częstotliwościach: $f_{x1} = 10 f_0$ and $f_s - f_{x1} = f_s - 10 f_0$. W widmie DtFT sytuacja jest inna. Wygenerowane, zepolone sygnały bazowe o wyższych częstotliwościach $k f_s \pm f_{x1}$ mają dokładnie takie same wartości próbek (i kształty) jak sygnały bazowe o niskich częstotliwościach $\pm f_{x1}$ i z tego powodu też wykazują podobieństwo do analizowanego sygnału. Wniosek: zakres częstotliwości $[-0.5 f_s, ..., 0.5 f_s]$ jest wystarczający do analizy częstotliwościowej sygnałów spróbkowanych - widmo obserwowane w tym zakresie powtarza się okresowo dla pozostałych częstotliwości. Dla sygnałów o wartościach rzeczywistych, mających widma symetryczne względem 0 Hz, nawet zakres $[0, ..., 0.5 f_s]$ jest wystarczający.
2. **(1/2*)** Teraz zwiększymy gestość próbkowania widma DtFT 10-krotnie, czyli ustawimy $df = 1$ Hz. *Ach! Jak to możliwe?! Co za widok!* Proszę, nie tak nerwowo. Teraz dokładnie widzimy powtarzające się okresowo widmo Fouriera okna prostokątnego, czyli moduł funkcji $\frac{sin(2\pi f T)}{2\pi f}$ (T - długość sygnału i okna). *Rozumiem, ale dlaczego wcześniej go nie widzieliśmy?!* Ponieważ widmo okna prostokątego jest oscylacyjne, przechodzi przez zero i my, przypadkowo, wcześniej pobieraliśmy z widma tylko wartości maksymalne i zerowe. *Ale dlaczego widmo okna prostokątnego jest widziane w widmie sygnału kosinusoidalnego? Gdzie funkcja okna prostokatnego występuje w równaniu widma DtFT funkcji kosinus? Nie widzę jej!* A ja widzę! Przecież nie analizujemy całego sygnału tylko

N-próbek jego FRAGMENTU, wyciętego przez okno prostokątne. Z tego powodu dwa, nieskończenie długie sygnały analogowe mnożą się wzajemnie przez siebie, kosinus i okno prostokątne. Właśnie dlatego widmo sygnału wynikowego jest splotem teoretycznego widma kosinusa (dwa "patyki" o wysokości $\frac{1}{2}$ dla częstotliwości dodatniej i ujemnej — równanie (4.12)) oraz teoretycznego widma okna prostokatnego (oscylacyjna funkcja $\frac{sin(2\pi fT)}{2\pi f}$). Właśnie z tego powodu widzimy oscylacyjne widmo okna prostokątnego w miejscach występowania "patyków" widma kosinusa. Ponieważ oba sygnały/funkcje są spróbkowane, obserwowane widmo powtarza się okresowo. Możemy także wytłumaczyć obserwowane zjawisko z wykorzystaniem właściwości "modulacji kosinusowej" ciągłej transformacji Fouriera (patrz odpowiednia tabela 4.1): pomnożenie dowolnego sygnału $w(n)$ przez kosinus o częstotliwości f_{x1}, przesuwa widmo wybranego sygnału do dodatniej i ujemnej częstotliwości kosinusa, czyli do f_{x1} i $-f_{x1}$, oraz skaluje to widmo przez $\frac{1}{2}$. W przypadku sygnałów dysktetnych mamy:

$$W\left(\frac{f}{f_{pr}}\right) = \sum_{n=-\infty}^{+\infty}\left[w(n)\cos\left(2\pi\frac{f_{x1}}{f_s}n\right)\right]e^{-j2\pi\frac{f}{f}n} = \frac{1}{2}\sum_{n=-\infty}^{+\infty}w(n)e^{-j2\pi\frac{(f-f_{x1})}{f_s}n} + \ldots$$
$$\frac{1}{2}\sum_{n=-\infty}^{+\infty}w(n)e^{-j2\pi\frac{(f+f_{x1})}{f_s}n} = \frac{1}{2}W\left(\frac{f-f_{x1}}{f_{pr}}\right) + \frac{1}{2}W\left(\frac{f+f_{x1}}{f_{pr}}\right) \tag{4.13}$$

Ponieważ w naszym programie obliczamy próbki widma DtFT w szerszym zakresie częstotliwościowym niż próbki widma DFT (od minus kilka częstotliwości próbkowania do plus kilka częstotliwości próbkowania, a nie tylko w zakresie $[0, f_{pr})$, obowiązującym dla DFT), dlatego w widmie DtFT mamy kilka kopii widma kosinusa, i konsekwencji widzimy wiele kopii widma okna prostokątnego. Oscylacje tego ostatniego są widoczne w miejscach występowania "patyków 1/2" widma spróbkowanej funkcji kosinus.

3. **(1/2*)** *Hola, hola! Ale ja nie chcę oglądać tego samego "filmu" wiele razy!* "No problem". Zmieńmy więc teraz interesujący nas zakres częstotliwości w widmie DtFT na $[-0.5f_s, ..., 0.5f_s]$, pozostawiając małą wartość kroku $df = 1$ Hz. Proszę, uruchom program. Czy jesteś usatysfakcjonowany? *I owszem. Ale wysoki poziom oscylacji w widmie jest bardzo niepokojący! Jeśli w sygnale występowałaby jeszcze jeden kosinus o innej częstotliwości i bardzo małej amplitudzie, to jego bardzo niska "górka" widmowa utonęła by w bocznych oscylacjach widmowych, pochodzących od "górki" składowej silnej! Ten kosinus w ogóle nie byłaby w widmie widoczny!*
4. **(1/2*)** Tak, masz rację! Dodajmy do sygnału jeszcze jedną, słabą składową kosinusoidalną o innej częstotliwości: `x=x1+x2`, przykładowo o częstotliwości $f_{x2} = 250$ Hz oraz bardzo małej amplitudzie $A_{x2} = 0.001$. Uruchom program: dodana składowa nie jest w widmie widoczna. *A nie mówiłem! Wiedziałem, wiedziałem ...!* Ja także wiem, że nie jest sztuką ciągle narzekać. Sztuką jest umieć zacisnąć zęby i rozwiązywać problemy. Czyli ...
5. **(1/2*)** ... pomnóżmy nasz dwu-składnikowy sygnał przez okno Hanninga, czyli zastąpmy okno prostokątne (`boxcar(), rectwin()` przez okno (`hanning()`). W języku Matlab mamy: `w2=hanning(N); w=w2; x=x.*w;` Ta funkcja ma oscylacje boczne widma leżące niżej niż w przypasku okna prostokatnego. Osiągnięte jest to jednak kosztem poszeszenia srodkowej części widma, tzw. listka główego. Uruchom program. Teraz druga składowa jest widoczna. *No tak, ale jeśli druga składowa sygnału byłaby bardzo, bardzo słaba, np. miała amplitudę równą $A_2 = 0.00001$ (10^{-5}, 10 mikrowoltów), to ...?*
6. **(1/2*)** Nie ma sprawy. Teraz skorzystamy z okna Czebyszewa, które ma listki boczne w widmie DtFT na poziomie 140 dB: $A_{sl} = 10^{-7}$, $20\log_{10}(A_{sl}) = -140$ dB. W Matlabie wybieramy: `w3=chewin(N,140); w=w3`. Uruchom program. Teraz druga składowa sygnału jest widoczna, nieprawdaż? *Rzeczywiście! Ale teraz "górki" w widmie są bardzo szerokie! Jeśli druga składowa miałaby częstotliwość zbliżoną do pierwszej, np. $f_{x2} = 110$ Hz, to jej "górka" połączyłaby się z "górką" składowej silniejszej i w ogóle nie byłaby widoczna!*
7. **(1/2*)** Nie widzę problemu. Skorzystajmy z właściwości "skalowania sygnału w osi czasu" ($x(at)$), posiadanej przez ciągłą transformację Fouriera ($X(\frac{f}{a})$ oraz przez DtFT (znajdź ją w tabeli 4.2). Wydłużanie fragmentu analizowanego sygnału (dla $a < 1$) powoduje odpowiednie zawężenie widma DtFT sygnału. Przykładowo, w wyniku użycia okna 10 razy dłuższego, widmo DtFT sygnału staje się 10-razy węższe (bardziej strome). Z tego powodu, zwiększymy obecnie 10 razy długość wektora próbek analizowanego sygnału, ustawiając: $N = 1000$. Zrób tak. Uruchom program. Widzisz, zadziałało. *Tak, tak, ale teraz ... po użyciu okna ... wartości maksimów widma nie są poprawne! Dla sygnału kosinusoidalnego powinniśmy otrzymać $1/2$!*
8. **(1/2*)** Tak. Podziwiam Twoją dociekliwość! Rzeczywiście, teraz musimy poprawić normalizację widma. Ponieważ użycie funkcji okna zmniejsza amplitudę fragmentu analizowanego sygnału, należy obliczeniowo skompensować ten efekt! Zamiast dzielić widmo przez N (co jest poprawne dla okna prostokatnego), należy je podzielić przez sumę wszystkich próbek użytego okna. W Matlabie: należy odkomentować następującą linię w programe: `X =`

`N*X/sum(w)`. Widzisz! Teraz widmo jest wyskalowane poprawnie. Dla okna prostokątnego mamy `N=sum(w)`, czyli tak jak było poprzednio.

9. *Ale* ... Bang! Czas upłynął! Koniec gry! ... To Ty zwyciężyłeś. Student zadający pytania zawsze zdobywa szacunek nauczyciela.

TEMAT #5: Sygnały i operacje na nich a ich widma. W tabeli 4.1 podano definicje matematyczne różnych sygnałów ciągłych (analogowych). Ponieważ ich kształty są dokładnie określone, dlatego można wyliczyć wzory na funkcje ich widm Fouriera z definicji przekształcenia Fouriera. Wynikowe wzory są także podane w tabeli. Ale to nie wszystko! Jeśli wykonujemy jakąś konkretną operację na sygnałach, to w jej wyniku powstaje inny sygnał, który także ma widmo Fouriera. Jak widmo sygnału wynikowego zależy od widma sygnału oryginalnego i wykonanej operacji? To również można obliczyć podstawiając przekształcony sygnał do wzoru na przekszałcenie Fouriera. Otrzymane wzory zebrano w tabeli 4.2.

Problem 4.8 ((*) Widma różnych sygnałów, czyli góry i doliny).** Zdyskretyzuj czas (`N=2000; fpr=2000; dt=1/fpr; t=dt*(0:N-1);` oraz wygeneruj wszystkie sygnały zdefiniowane w tabeli 4.1. Oblicz widmo DFT każdego z nich za pomocą funkcji Matlaba `fft()`, narysuj je w skali liniowej i decybelowej oraz porównaj jego kształt z wzorem podanym w tabeli. Jeśli “ręcznie" obliczysz widma DtDT, to otrzymasz dodatkowy punkt ().

Problem 4.9 (Widmowe konsekwencje przetwarzania sygnałów).** Zdyskretyzuj czas (`N=2000; fpr=2000; dt=1/fpr; t=dt*(0:N-1);` oraz wygeneruj sinusoidę o częstotliwości 50 Hz (sygnał `x=sin(2*pi*50*t)`) i okno Kaisera z $\beta = 15$ (sygnał `y=kaiser(N,15)`). Oblicz i narysuj widma DFT obu sygnałów (np. `X=fft(x)`). Następnie wykonaj na tych sygnałach dowolnych pięć operacji, zdefiniowanych w tabeli 4.2, oblicz widma DFT sygnałów wynikowych i je narysuj. Sprawdź czy wynik zgadza się z wzorem podanym w ostatniej kolumnie tabeli. *Telefon od przyjaciela*: splot w Matlabie to funkcja `z=conv(x,y)`, korelacja to `Rxy=xcorr(x,y)`, a pochodna to `diff(x)./diff(t)`.

TOPIC #6: Zastosowanie DFT/DtFT do sygnałów nagranych. Algorytmy i programy DFT/DtFT mogą być z powodzeniem zastosowane do analizy częstotliwościowej sygnałów, pochodzących z otaczającego nas świata. Dzięki nim możemy odkyć "co w trawie piszczy?"

Problem 4.10 (* Czy mógłbym śpiewać jak Sławomir lub Mandaryna?).** Użyj algorytmów/programów DFT/DtFT (programy z listing 4.2) do wyznaczenia częstotliwości otwierania się twoich strun głosowych podczas śpiewania samogłosek: a,e,i,o,u. Nagraj oddzielnie pojedyncze dźwięki, narysuj sygnały czasowe (wyskaluj oś czasu w sekundach) oraz ich widma DFT (`X=fft(x)`) i DtFT (wyskaluj oś częstotliwości w hercach). Nałóż na siebie widma DFT i DtFT. Pokaż je w skali liniowej i decybelowej.

Problem 4.11 (* W magicznej krainie dźwięków wszelakich).** Ze strony internetowej *FindSounds* pobierz 2-3 sygnały o różnym pochodzeniu (śpiew ptaków, warkot maszyn, ...) oraz oblicz ich widma DFT (`X=fft(x)`) i DtFT (programy z listing 4.2). Narysuj sygnały i ich widma. Wyskaluj oś czasu sygnału w sekundach oraz oś częstotliwości widma sygnału w hercach. Nałóż na siebie widma DFT i DtFT. Przedstaw je w skali liniowej i decybelowej. Powiedz jakie składowe częstotliwościowe dominują w każdym sygnale.

Problem 4.12 (* Z jaką częstotliwością bije moje serducho?).** Pobierz z internetu kilka sygnałów EKG, np. ze strony https://www. physionet.org/cgi-bin/atm/ATM lub skorzystaj z sygnału EKG dołączonego do laboratorium 1. Wyznacz częstotliwość pracy serca, korzystając z DFT (`X=fft(x)`) i DtFT (programy z listingu 4.2). Pokaż sygnały EKG (wyskaluj oś czasu w sekundach) oraz ich widma DFT i DtFT (wyskaluj oś częstotliwości w hercach). Narysuj widma w skali liniowej i decybelowej.

Problem 4.13 (* Filtracja częstotliwości w dziedzinie współczynników DFT: samochód w lesie).** Pobierz z internetu, np. strony internetowej *FindSounds* dwa dźwieki spróbkowane z tą samą częstotliwością, ale różniące się widmem DFT: jeden powinien zawierać niskie częstotliwości (np. warkot silnika samochodu), a drugi wysokie częstotliwości (np. śpiew ptaka). Osobno oblicz i wyświetl ich widma DFT: zapisz wartości graniczne zakresów częstotliwości. Następnie dodaj do siebie oba sygnały (krótszy dopełnij na końcu zerami). Oblicz i wyświetl DFT sumy. Następnie wyzeruj w widmie współczynniki DFT w większości związane z jednym sygnałem (np. warkotem silnika). Pamiętaj, że musisz usunąć zarówno składową o częstotliwości dodatniej jak i ujemnej. Potem wykonaj transformatę IDFT i wyświetl otrzymany wynik. Jeśli sygnał jest zespolony to znaczy, że źle wyzerowałeś (usunąłeś) częstotliwości i wymagana symetria widma DFT została przez ciebie naruszona - popraw błąd albo weź tylko część rzeczywistą wyniku. Odsłuchaj sygnał. Pewnie ptaszek szaleje ze szczęścia: co za cisza!

Problem 4.14 (Widmowe konsekwencje różnych operacji chirurgicznych wykonanych na sygnałach).** Wczytaj do Matlaba dowolny sygnał dźwiękowy `x` nagrany przez ciebie lub pobrany z Internetu, narysuj go (wyskaluj oś czasu), oblicz i narysuj jego widmo DFT (wyskaluj oś częstotliwości). Wykonaj na sygnale 3 różne operacje z tabeli 4.2 i porównaj na jednym rysunku: 1) oryginał i wynik przetwarzania, 2) DFT oryginału z DFT wyniku. Na koniec odsłuchaj oba sygnały. Sygnał `y` może być np. funkcją okna `y=hanning(N)`. Splot w Matlabie to funkcja `z=conv(x,y)`, dodatkowo użyj np. `y=[1/4,1/2,1/4]` albo `y=fir1(25,0.25)`. Pochodną możesz zrealizować w ten sposób: `z=diff(x)./diff(t)`.

Table 4.1: Sygnały ciągłe i ich ciągłe widma Fouriera

Nr	Nazwa sygnału	Równanie sygnału	Równanie widma
1	Okno prostokątne	$r_T(t) = \begin{cases} 0 & \text{for} \;\; \lvert t\rvert > T \\ 1 & \text{for} \;\; \lvert t\rvert \le T \end{cases}$	$X(\omega) = 2\frac{\sin\omega T}{\omega}$
2	Sygnał znaku	$x(t) = \mathrm{sign}(t)$	$X(\omega) = \frac{2}{j\omega}$
3	Funkcja Gaussa	$x(t) = e^{-at^2}$	$X(\omega) = \sqrt{\frac{\pi}{a}}e^{-\omega^2/(4a)}$
4	Jednostronna eksponenta	$x(t) = \begin{cases} 0 & t < 0 \\ e^{-at} & t \ge 0 \end{cases}, \quad a > 0$	$X(\omega) = \frac{1}{a+j\omega}$
5	Tłumiony sinus	$x(t) = \begin{cases} 0 & t < 0 \\ Ae^{-at}\sin(\omega_0 t) & t \ge 0 \end{cases}$	$X(\omega) = \frac{A\omega_0}{(a+j\omega)^2+\omega_0^2}$
6	Tłumiony kosinus	$x(t) = \begin{cases} 0 & t < 0 \\ Ae^{-at}\cos(\omega_0 t) & t \ge 0 \end{cases}$	$X(\omega) = A\frac{a+j\omega}{(a+j\omega)^2+\omega_0^2}$
7	Fragment kosinusa	$x(t) = \cos(\omega_0 t)\cdot r_T(t)$	$X(\omega) = \frac{\sin((\omega-\omega_0)T)}{\omega-\omega_0} + \frac{\sin((\omega+\omega_0)T)}{\omega+\omega_0}$

Table 4.2: Podstawowe właściwości ciągłej transformacji Fouriera: operacje i ich konsekwencje widmowe

Nr	Właściwość	Operacja na sygnale	Konsekwencje widmowe
1	Liniowość	$ax(t)+by(t)$	$aX(f)+bY(f)$
2	Skalowanie	$x(at), \quad a>0$	$\frac{1}{a}X\left(\frac{f}{a}\right)$
3	Odwrócenie czasu	$x(-t)$	$X(-f)$
4	Sprzężenie	$x^*(t)$	$X^*(-f)$
5	Przesunięcie w czasie	$x(t-t_0)$	$e^{-j2\pi f t_0}X(f)$
6	Przesunięcie w częstotliwości	$e^{\pm j2\pi f_0 t}x(t)$	$X(f \mp f_0)$
7	Mnożenie	$x(t)\cdot y(t)$	$\int_{-\infty}^{\infty} X(v)Y(f-v)dv$
8	Modulacja zespolona	$e^{\pm j2\pi f_0 t}x(t)$	$X(f \mp f_0)$
9	Modulacja kosinusem	$x(t)\cos(2\pi f_0 t)$	$\frac{1}{2}[X(f-f_0)+X(f+f_0)]$
10	Modulacja sinusem	$x(t)\sin(2\pi f_0 t)$	$\frac{-j}{2}[X(f-f_0)-X(f+f_0)]$
11	Splot	$\int_{-\infty}^{\infty} x(\tau)y(t-\tau)d\tau$	$X(f)\cdot Y(f)$
12	Korelacja	$\int_{-\infty}^{\infty} x(t)y^*(t+\tau)dt$	$X(f)\cdot Y^*(f)$
13	Pochodna	$\frac{\mathrm{d}^n x(t)}{\mathrm{d}t^n}$	$(j2\pi f)^n\cdot X(f)$
14	Energia - równ. Parsevala	$\int_{-\infty}^{\infty} x(t)x^*(t)dt$	$\int_{-\infty}^{\infty} X(f)X^*(f)df$

Table 4.3: Basic CFT features: signal processing and its spectral consequence

No	Feature	Signal manipulation	Spectral consequence
1	Linearity	$ax(t)+by(t)$	$aX(f)+bY(f)$
2	Scaling	$x(at), \quad a>0$	$\frac{1}{a}X\left(\frac{f}{a}\right)$
3	Time reverse	$x(-t)$	$X(-f)$
4	Conjugation	$x^*(t)$	$X^*(-f)$
5	Time shift	$x(t-t_0)$	$e^{-j2\pi f t_0}X(f)$
6	Frequency Shift	$e^{\pm j2\pi f_0 t}x(t)$	$X(f \mp f_0)$
7	Multiplication	$x(t)\cdot y(t)$	$\int_{-\infty}^{\infty} X(v)Y(f-v)dv$
8	Complex modulation	$e^{\pm j2\pi f_0 t}x(t)$	$X(f \mp f_0)$
9	Cos() modulation	$x(t)\cos(2\pi f_0 t)$	$\frac{1}{2}[X(f-f_0)+X(f+f_0)]$
10	Sin() modulation	$x(t)\sin(2\pi f_0 t)$	$\frac{-j}{2}[X(f-f_0)-X(f+f_0)]$
11	Convolution	$\int_{-\infty}^{\infty} x(\tau)y(t-\tau)d\tau$	$X(f)\cdot Y(f)$
12	Correlation	$\int_{-\infty}^{\infty} x(t)y^*(t+\tau)dt$	$X(f)\cdot Y^*(f)$
13	Derivative	$\frac{\mathrm{d}^n x(t)}{\mathrm{d}t^n}$	$(j2\pi f)^n\cdot X(f)$
14	Energy - Parseval eq.	$\int_{-\infty}^{\infty} x(t)x^*(t)dt$	$\int_{-\infty}^{\infty} X(f)X^*(f)df$

AGH

Laboratorium 5
Szybka Tranformacja Fouriera - FFT

Streszczenie Podczas tego laboratorium poznamy algorytm szybkiej transformacji Fouriera (FFT - Fast Fourier Transform), a dokładnie algorytm radix-2 (podziału na 2) z decymacją w czasie (DIT - Decimation-In-Time). Nauczymy się także jak obliczyć dwa widma dwóch różnych sygnałów o wartościach rzeczywistych (nie zespolonych!) za pomocą jednej transformacji DFT.

TEMAT #1: Algorytm FFT typu Radix-2 DIT. Równanie DFT jest następujące:

$$\bar{\mathbf{X}}_{N\times 1} = \mathbf{F}_{N\times N}\cdot\bar{\mathbf{x}}_{N\times 1}, \qquad F(k,n) = \frac{1}{\sqrt{N}}\cdot e^{-j\frac{2\pi}{N}kn}, \quad k,n = 0,1,2,...,N-1, \tag{5.1}$$

czyli N-elementowy wektor pionowy $\bar{\mathbf{x}}_{N\times 1}$ z próbkami sygnału jest mnożony przez macierz Fouriera $\mathbf{F}_{N\times N}$ o wymiarach $N\times N$, dając w wyniku N-elementowy wektor $\bar{\mathbf{X}}_{N\times 1}$ ze współczynnikami Fouriera. Idea redukcji liczby obliczeń jest taka: zamiast mnożyć N próbek sygnału przez macierz DFT o wymiarach $N\times N$, dzielimy N-elementowy wektor próbek wejściowych na dwa wektory o długości $\frac{N}{2}$ próbek, biorąc oddzielnie próbki parzyste i nieparzyste, a następnie mnożymy je przez macierze DFT o wymiarach $\frac{N}{2}\times\frac{N}{2}$. Potem wynikowe widmo DFT o długości N jest zrekonstruowane na podstawie dwóch widm DFT o długości $\frac{N}{2}$. W ten sposób liczba mnożeń jest w przybliżeniu zmniejszona o połowę. Podział próbek na parzyste i nieparzyste jest robiony wielokrotnie. Jego wynikiem są tzw. obliczenia *motylkowe*.

Dla przykładu prześledzmy jak szybki algorytm DFT działa w przypadku $N = 4$:

$$\begin{bmatrix} X(0)\\ X(1)\\ X(2)\\ X(3)\end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1\\ 1 & -j & -1 & j\\ 1 & -1 & 1 & -1\\ 1 & j & -1 & -j\end{bmatrix} \begin{bmatrix} x(0)\\ x(1)\\ x(2)\\ x(3)\end{bmatrix} = \tag{5.2}$$

$$= \begin{bmatrix} 1 & 1\\ 1 & -1\\ 1 & 1\\ 1 & -1\end{bmatrix} \begin{bmatrix} x(0)\\ x(2)\end{bmatrix} + \begin{bmatrix} 1 & 1\\ -j & j\\ -1 & -1\\ j & -j\end{bmatrix} \begin{bmatrix} x(1)\\ x(3)\end{bmatrix} = \tag{5.3}$$

$$= \begin{bmatrix} \mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\\ 1 & 1\\ 1 & -1\end{bmatrix} \begin{bmatrix} x(0)\\ x(2)\end{bmatrix} + \begin{bmatrix} 1\\ -j\\ -1\\ j\end{bmatrix} .* \begin{bmatrix} \mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\\ 1 & 1\\ 1 & -1\end{bmatrix} \begin{bmatrix} x(1)\\ x(3)\end{bmatrix} = \tag{5.4}$$

$$= \begin{bmatrix} \mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\\ & \\ & \end{bmatrix} \begin{bmatrix} 1\\ 3\end{bmatrix} + \begin{bmatrix} 1\\ -j\\ -1\\ j\end{bmatrix} .* \begin{bmatrix} \mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\\ & \\ & \end{bmatrix} \begin{bmatrix} 2\\ 4\end{bmatrix} = \tag{5.5}$$

$$= \begin{bmatrix} \mathbf{4}\\ \mathbf{-2}\\ 4\\ -2\end{bmatrix} + \begin{bmatrix} 1\\ -j\\ -1\\ j\end{bmatrix} .* \begin{bmatrix} \mathbf{6}\\ \mathbf{-2}\\ 6\\ -2\end{bmatrix} = \begin{bmatrix} \textbf{10}\\ \textbf{-2+2j}\\ \textbf{-2}\\ \textbf{-2-2j}\end{bmatrix} \tag{5.6}$$

Na początku, równanie (5.3) — próbki sygnału są dzielone na parzyste i nieparzyste oraz macierz transformacji o wymiarach 4×4 jest zastępowana dwoma podmacierzami o wymiarach 4×2. Dodatkowo, równanie (5.4) — mnożenie próbek o indeksach nieparzystych przez drugą podmacierz zostaje zmodyfikowane: najpierw wykonuje się mnożenie przez macierz uproszczoną, a potem koryguje jego wynik - każda wyliczona wartość jest mnożona przez inny współczynnik korekcyjny. $.*$ w równaniu (5.4) oznacza, podobnie jak w Matlabie, mnożenie odpowiadających sobie elementów dwóch wektorów lub macierzy, pierwszy-przez-pierwszy, drugi-przez-drugi, ... i tak dalej. Teraz widzimy, że dolne macierze o wymiarach 2×2 są takie same jak górne macierze (pogrubione cyfry w niebieskim kolorze). Z tego powodu wystarczy tylko pomnożyć próbki sygnałów parzystych i nieparzystyc przez górne macierze o wymiarach 2×2 (pogrubione niebieskie liczby **1** oraz **-1**), a potem skopiować wynik i zapisać go jeszcze raz poniżej. Z tego powodu w równaniu (5.5) brakuje liczb dolnych macierzy oraz w równaniu (5.6) dwie wartości obliczone w części górnej (niebieskie/pogrubione) są skopiowane do części dolnej (jako czerwone)! Ostatecznie otrzymany jst taki sam wynik jak w równaniu (5.2) tylko szybciej (z mniejszą liczbą mnożeń). W oryginalnym algorytmie wykonuje się 4^2=16 mnożeń. W jego szybkiej wersji: $2\cdot 2^2 = 8$ plus 4 mnożenia korekcji, razem 12 mnożeń. Idea FFT jest wytłumaczona graficznie na rysunku 5.1.

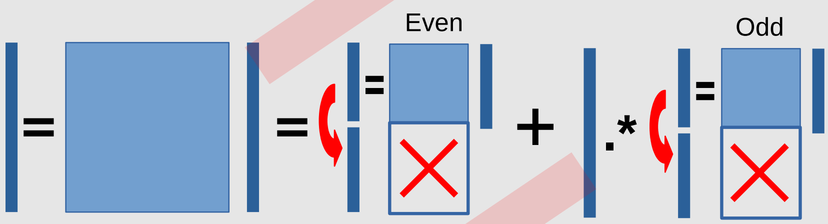

Fig. 5.1: Ilustracja graficzna zasady zmniejszenia liczby mnożeń w algorytmie FFT radix-2 DIT po pierwszym rozdzieleniu próbek na te o indeksach nieparzystychi parzystych. Mnożenie parzystych/nieparzystych próbek przez "białe" pod-macierze z czerwonymi krzyżykami "$\times$" nie jest wykonywane, ponieważ dolne, "białe" pod-macierze są takie same jak górne, "niebieskie" pod-macierze - kopiowany jest wynik górnego iloczynu wektor-macierz (czerwona strzałka). ".*" oznacza mnożenie wektorów element-przez-element tak jak w języku Matlab

A teraz to samo jak powyżej tylko ... OGÓLNIEJ. Pomysł obliczeniowy jest następujacy: zamiast mnożyć N-próbek długiego sygnału przez macierz DFT o wymiarach $N\times N$, transformowany sygnał jest dzielony na dwa o połowę krótsze wektory o długości $\frac{N}{2}$-próbek. Tworzy się je grupując osobno wszystkie próbki o numerach parzystych (0,2,4,...) oraz nieparzystych (1,3,5,...). Potem wektory te powinny być osobno pomnożone przez odpowiednie pod-macierze DFT o wymiarach $N\times\frac{N}{2}$. Jednak, upraszczajac, dolne połówki pod-macierzy są takie same jak górne, dlatego mnoży się próbki parzyste i nieparzyste przez odpowiednie macierze o wymiarach $\frac{N}{2}\times\frac{N}{2}$, obliczone dwa $\frac{N}{2}$-próbkowe widma DFT używa się dwa razy (najpierw oryginał, potem jego kopia) oraz odtwarza się całe N-próbkowe widmo DFT. Dzięki temu liczba mnożeń jest zmniejszona o połowę. Cała operację ilustruje rysunek 5.1. Podział wektorów próbek na próbki o indeksach parzystych i nieparzystych jest dokonywany wielokrotnie aż do otrzymania zbiorów dwuelementowych. Z wielu 2-elementowych widm są odtwarzane widma 4-elementowe, z nich 8-elementowe, potem 16-elementowe, itd. Zamiast $N\times N$ mnożeń wykonuje się $N\log_2(N)$ mnożeń. Obliczenia mają postać tzw. "motylków": dodaj/odejmij dwie liczby. Matematycznie jeden poziom dekompozycji DIT radix-2 można opisać w sposób następujący ($k = 0,1,2,...,N-1$):

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} = \sum_{m=0}^{\frac{N}{2}-1} x(2m)e^{-j\frac{2\pi}{N/2}km} + e^{-j2\pi\frac{k}{N}}\sum_{m=0}^{\frac{N}{2}-1} x(2m+1)e^{-j\frac{2\pi}{N/2}km}, \quad (5.7)$$

$$X^{(N)}(k) = X_e^{(N/2)}(k) + e^{-j\frac{2\pi}{N}k}X_o^{(N/2)}(k), \quad (5.8)$$

gdzie niebieskie (pierwsze) widmo to $\frac{N}{2}$-punktowe DFT próbek parzystych (*even*: $n = 0,2,4,...,\frac{N}{2}-1$), zaś czerwone (drugie) widmo to $\frac{N}{2}$-punktowe DFT próbek nieparzystych (*odd*: $n = 1,3,5,...,\frac{N}{2}-1$). Widma te mają takie same wartości dla $k = 0,1,2,...,\frac{N}{2}-1$ (górna, "niebieska" podmacierz na rysunku) oraz $\frac{N}{2}+k$ (dolna, "biała" podmacierz), czyli powtarzają się z okresem $\frac{N}{2}$, ponieważ dla $\frac{N}{2}+k$ funkcje bazowe są takie same jak dla k:

$$e^{-j\frac{2\pi}{\frac{N}{2}}\left(\frac{N}{2}+k\right)n} = e^{-j\frac{2\pi}{\frac{N}{2}}\left(\frac{N}{2}n\right)}\cdot e^{-j\frac{2\pi}{\frac{N}{2}}(kn)} = e^{-j2\pi n}\cdot e^{-j\frac{2\pi}{\frac{N}{2}}(kn)} = 1\cdot e^{-j\frac{2\pi}{\frac{N}{2}}kn} = e^{-j\frac{2\pi}{\frac{N}{2}}kn} \quad (5.9)$$

Dlatego możemy skopiować wynik mnożenia otrzymany dla podmacierzy górnej (niebieskiej). Co w języku Matlab zapisujemy tak:

```
xe=x(1:2:end); xo=x(2:2:end); Xe=fft(xe); Xo=fft(xo);
X=[Xe, Xe] + exp(-j*2pi/N*(0:N-1)).*[Xo, Xo];
```

Zwróćmy uwagę na równość:

$$e^{-j\frac{2\pi}{N}\left(\frac{N}{2}+k\right)} = \text{-}\, e^{-j\frac{2\pi}{N}(k)}. \tag{5.10}$$

Jak widać wartość korekty DFT jest taka sama dla k oraz $\frac{N}{2}+k$, inny jest tylko znak. Ponieważ wartości $X_e/o^{(N/2)}(k)$ też są identyczne dla tych indeksów:

$$X_{e/o}^{(N/2)}\left(\frac{N}{2}+k\right) = X_{e/o}^{(N/2)}(k) \tag{5.11}$$

dlatego można przeprowadzić korektę (czyli wykonać mnożenie) tylko dla wartości $k < \frac{N}{2}$, a dla $\frac{N}{2}+k$ — jedynie zanegować już posiadany wynik. Prowadzi to do tzw. oblczeń *motylkowych*. Motylkiem FFT nazywa się operację równoczesnego obliczanie dwóch wartości widma $X(k)$ oraz $X\left(\frac{N}{2}+k\right)$ dla $k = 0,1,2,...,\frac{N}{2}-1$:

$$\text{w(k)} = e^{-j\frac{2\pi}{N}k} X_o^{(N/2)}(k), \tag{5.12}$$

$$X^{(N)}(k) = X_e(k) \text{ + w(k)}, \tag{5.13}$$

$$X^{(N)}\left(\frac{N}{2}+k\right) = X_e^{(N/2)}(k) \text{ - w(k)}. \tag{5.14}$$

Problem 5.1 (* Testowanie idei algorytmu radix-2 DIT FFT). Napisz program Matlaba do szybkiego obliczania poniższego równania DFT dla $N = 4$-elementowego wektora liczb $\bar{\mathbf{x}}_{4\times 1} = [x(0), x(1), x(2), x(3)]^T$ o dowolnych warościach:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

Użyj równań od (5.3) do (5.6). Na początku, zdekomponuj macierz DFT o wymiarach 4×4 na dwie podmacierze o wymiarach 4×2. Potem wyciągnij wektor korekty $\exp\left(-j\frac{2\pi}{N}(0:N-1)\right)$, $N = 4$, przed drugą podmacierz i zapiesz go przed nią. Teraz dwie otrzymane podmacierze o wymiarach 4×2 powinny mieć taką samą macierz o wymiarach 2×2 w swoich częściach górnych i dolnych:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{5.15}$$

Dzięki temu mnożenie próbek sygnału przez macierz dolną nie musi być wykonywane: wystarczy pomnożyć próbki sygnału przez macierz górną i skopiować wynik na dół. (*) Teraz spróbuj napisać program szybkiego obliczania DFT dla $N = 8$-elementowego wektora danych wejściowych. Jakie wartości mają elementy macierzy transformacji Fouriera w tym przypadku? Rzuć okiem na równanie (5.1).

Problem 5.2 (* Szybki algorytm DFT z jednokrotnym podziałem próbek sygnału na "parzyste/nieparzyste" (even/odd)). Obecnie dalej zgłębimy problem powtarzania się elementów macierzy DFT i wykorzystania tego faktu. Wstępnie analizowaliśmy go w problemie **??**. Użyj programu 5.1. Ustaw w nim `N=4, 8, 16` oraz zweryfikuj, że wynik DFT sygnału `X` może być zrekonstruowany na podstawie dwóch wektorów `Xe, Xo`, będących wynikiem DFT próbek sygnału o numerach parzystych/even (0,2,4,...) i nieparzystych/odd (1,3,5,...). Ponieważ Matlab indeksuje wektory od 1, to są to następujące wektory próbek: `xe=x(1:2:end), xo=x(2:2:end)`. Utwórz macierze DFT `Ae,` `Ao` związane z wektorami `xe,xo` oraz wykorzystaj je do obliczenia widm DFT obu wektorów, czyli `Xe=Ae*xe.';` `Xo=Ao*xo.';` — w przypadku wątpliwości wróć do laboratorium na temat DFT/DtFT. W programie z listingu 5.1 obliczono widmo DFT całego sygnału z użyciem funkcji Matlaba (`X=fft(x)`) oraz zrekostruowano je na podstawie widm `Xe=fft(xe), Xo=fft(xo)`. Ty oblicz widma `Xe, Xo` korzystając z równań macierzowych, tak jak to wyjaśniono powyżej.

Listing 5.1: Pierwszy krok algorytmu radix-2 DIT FFT

```
% cps_05_fft1.m
clear all; close all;

N = 100; x = rand(1,N);
Xm = fft(x);
Xe = fft(x(1:2:N));
Xo = fft(x(2:2:N));
X = [ Xe, Xe ] + exp(-j*2*pi/N*(0:1:N-1)) .* [Xo, Xo ];
error = max( abs( X - Xm ) ),
```

Problem 5.3 (* Szybki algorytm DFT z dwukrotnym podziałem próbek sygnału na "parzyste/nieparzyste" (even/odd)). Do programu z listingu 5.1 dodaj drugi etap podziału próbek na te o numerach parzystych i nieparzystych, czyli `xee=xe(1:2:end); xeo=xe(2:2:end); xoe=xo(1:2:end); xoo=xo(2:2:end);`. Oblicz widma DFT `Xee, Xeo, Xoe, Xoo` czterech sygnałów `xee, xeo, xoe, xoo`, np. `Xee=fft(xee)`. Zrekonstuuj całe widmo DFT `X=fft(x)` na podstawie obliczonych widm `Xee, Xeo, Xoe, Xoo`.

Problem 5.4 (* Rekursywna implementacja algorytmu radix-2 DIT FFT). W listingu5.2 jest przedstawiona funkcja rekurencyjna, implementująca cały, wielopoziomowy algorytm radix-2 DIT FFT. Przeanalizuj jej kod. Zrozum go. Na najniższym poziomie dla wektorów dwu-elementowych jest wykonywane dwu-punktowe DFT (dodaj i odejmij dwie próbki wejściowe - wyprowadź to na podstawie ogólnego wzoru). Na wyższych etapach - jedynie składanie widm krótszych do dłuższych. Wygeneruj szum gaussowski `p=8; N=2^p; x=randn(N,1)`. Oblicz jego widmo DFT za pomocą funkcji Matlaba `X1=fft(x);` oraz naszej funkcji rekurencyjnej `X2=myRecFFT(x)`. Porównaj oba wyniki `err=max(abs(X1-X2))`. Zmień wartość `p`. Uruchom program. Jaki jest błąd `err`? Zmień transformowany sygnał na inny, obserwuj błąd.

Listing 5.2: Rekurencyjna funkcja implementująca algorytm Radix-2 DIT FFT

```
function X = myRecFFT(x)

% Rekurencyjna funkcja algorytmu radix-2 DIT FFT
N = length(x);
if(N==2)
   X(1) = x(1) + x(2);          % # 2-punktowe DFT
   X(2) = x(1) - x(2);          % # na najnizszym poziomie
else
   X1 = myRecFFT(x(1:2:N));    % samo-wywolanie dla probek parzystych
   X2 = myRecFFT(x(2:2:N));    % samo-wywolanie dla probek nieparzystych
   X = [ X1 X1 ] + exp(-j*2*pi/N*(0:N-1)).* [X2 X2];    % zlozenie widm
end
```

Problem 5.5 (* Szybki algorytm odwrotnego DFT (IDFT) - Radix-2 DIT IFFT). Skopiuj funkcję z listing 5.2 i zapisz ją jako `myRecIFFT.m`. Potem zmodyfikuj zapisaną funkcję: powinna ona teraz obliczać odwrotne DFT w sposób szybki. Jaka jest różnica pomiędzy DFT i IDFT? Podpowiem: bardzo mała. W wyniku wykonania sekwencji wywołań `X=myRecFFT(x); xr=myRecIFFT(X)/N;` powinniśmy otrzymać ten sam sygnał, czyli błąd `err=max(abs(x-xr))` powinien być bardzo, bardzo mały (na poziomie 10^{-14}, tak jak w przypadku użycia sekwencji funkcji Matlaba: `xr=ifft(fft(x))`. Sprawdź to! Napisz funkcję Matlaba, która oblicza FFT albo IFFT otrzymanego wektora liczb, w zależności od wartości wejściowego parametru sterującego, równej 1 albo -1. WAŻNE: zwróć uwagę, że funkcja Matlaba `ifft()`, przeprowadzająca obliczenia podobnie jak `fft()`, dzieli wynik przez liczbę N otrzymanych próbek widma DFT. Takiego dzielenia nie ma w funkcji `fft()`. Dlatego dzielenie to mógłbyś też ukryć wewnątrz Twojej funkcji `myRecIFFT(X)`.

Problem 5.6 (* Algorytm wielopoziomowego sortowania próbek na te o numerach parzystych i nieparzystych). W algorytmie radix-2 DIT FFT, N próbek sygnału wejściowego jest wielokrotnie sortowanych (przestawianych) na oddzielne bloki próbek o indeksach parzystych i nieparzystych. Po zakończeniu przestawiania jest wykonywanych $\frac{N}{2}$ 2-punktowych transformacji DFT. Potem: 1) $\frac{N}{2}$ 2-punktowe widma DFT są składane do $\frac{N}{4}$ 4-punktowych widm DFT, 2) $\frac{N}{4}$ 4-punktowe widma DFT są składane do $\frac{N}{8}$ 8-punktowych widm DFT spectra, itd, aż do momentu otrzymania N-punktowego widma DFT całego sygnału. Nowe pozycje próbek, przed serią 2-punktowych transformat DFT, są znalezione poprzez odwrócenie bitów początkowego numeru każdej próbki sygnału, przykładowo dla $N = 16$-punktowego FFT numer próbki $n = 3 = 0011b$ jest zamieniany na $n = 1100b = 12$. W ten sposób dla $N = 8$-punktowego FFT, indeksy próbek [0,1,2,3,4,5,6,7] są zamieniane na indeksy [0,4,2,6,1,5,3,7] (sprawdź to!). Napisz swój własny program na zmianę kolejności próbek, poprawny dla dowolnej wartości $N = 2^p$ (`N=2^p`). Jest to pierwszy, obowiązkowy etap algorytmu radix-2 DIT FFT. Sprawdź poprawność działania napisanego programu. Jeśli masz wątpliwości, czegoś nie rozumisz, to popatrz na początek programu z listingu 5.3 i w nim poszuka inspiracji.

Problem 5.7 (* Nierekurencyjna wersja algorytmu radix-2 DIT FFT w dowolnym języku programowania).** Przeanalizuj program z listingu 5.3. Znajdź w nim podobieństwo do rekurencyjnej funkcji implementującej algorytm radix-2 DIT FFT, przedstawionej w listingu 5.2. Uruchom program 5.3, sprawdź poprawność otrzymywanych wyników - porównaj je z wynikami funkcji Matlaba `X=fft(x)`. Napisz program implementujący ten algorytm FFT w Twoim ulubionym języku programowania: C, C++, Python, Java, ... Wynik działania Twojego programu powinien być identyczny jak programu 5.3 — dla dowolnej wartości `N=2^p` oraz dowolnego sygnału `x`. Zwróć szczególną uwagę na sposób implementacji obliczeń na liczbach zespolonych, mających część rzeczywistą i urojoną.

Listing 5.3: Algorytm Radix-2 DIT FFT w Matlabie - wersja blokowa

```
% cps_05_fft2.m

% Algorytm radix-2 DIT (decimation in time) FFT
  clear all; close all;

  N=8;                % liczba probek sygnalu (potega dwojki)
  x=0:N-1;            % przykladowy analizowany sygnal, np. inny wybor x=randn(1,N)
  Nbits = log2(N);    % liczba bitow potrzebna na indeksy probek, dla N=8, Nbits=3

% Przestawianie probek sygnalu (odwracanie bitow numeru probki)
  n = 0:N-1;              % indeksy WSZYSTKICH probek
  m = dec2bin(n);         % bity tych indeksow
  m = m(:,Nbits:-1:1);    % odwrocone bity
  m = bin2dec(m);         % nowe indeksy WSZYSTKICH probek
  y(m+1) = x(n+1);        % przestawianie danych wejsciowych
  y,                      % sprawdzenie wyniku

% WSZYSTKIE 2-punktowe DFTs na sasiednich parach probek (po ich przestawieniu)
  y = [ 1 1; 1 -1] * [ y(1:2:N); ...
                       y(2:2:N) ]; y = y(:)';   % 2-punktowe widma DFT

% Rekonstrukcja N-punktowego DFT z 2-punktowych
% Widma DFT: 2-punktowe --> 4-punktowe --> 8-punktowe --> 16-punktowe ...
  Nx = N;                   % liczba probek (zmiana nazwy zmiennej)
  Nlevels = Nbits;          % liczba etapow oliczeniowych rowna log2(Nx)
  N = 2;                    % poczatkowa dlugosc DFT po 2-punktowych DFT
  for lev = 2 : Nlevels                 % nastepny ETAP
      N = 2*N;                          % nowa dlugosc widma DFT po polaczeniu
      Nblocks = Nx/N;                   % nowa liczba widm DFT po polaczeniu
      W = exp( -j*2*pi/N*(0:N-1) );     % korekta stosowana na tym etapie
      for k = 1 : Nblocks                         % nastepny BLOK dwoch DFT
          y1 = y( 1     + (k-1)*N : N/2 + (k-1)*N );    % widmo y1 - NIZEJ
          y2 = y( N/2+1 + (k-1)*N : N   + (k-1)*N );    % widmo y2 - WYZEJ
          y(1 + (k-1)*N : N + (k-1)*N ) = [ y1 y1 ] + W .* [ y2 y2 ];   % LACZENIE
      end
  end
  ERROR = max( abs( fft(x) - y ) ),     % blad?
```

Problem 5.8 (* Nierekurencyjna wersja algorytmu radix-2 DIT FFT z użyciem jednego, wielokrotnie wykonywanego “motylka obliczeniowego” — w różnych językach programowania).** Przeanalizuj program z listing 5.4. Zrozum jego działanie, posiłkując się np. [15] [40]. Znajdź jego podobieństwo do programów z listingu 5.2 oraz listingu 5.3. Uruchom program, sprawdź poprawność otrzymywanego wyniku (czy `ERROR` jest równy zero?). Napisz program w Twoim ulubionym języku programowania (C, C++, Python, Java, ...) implementujący kod Matlaba z listingu 5.4. Powinien on zwracać ten sam wynik dla dowolnej wartości `N=2^p` oraz dowolnego sygnału `x`. Uważaj na sposób implementacji oblczeń na liczbach zespolonych.

Listing 5.4: Algorytm Radix-2 DIT FFT w Matlabie — pojedynczy motylek obliczeniowy wykonywany wielokrotnie

```
% cps_05_fft3.m

% Algorytm radix-2 DIT (decimation in time) FFT
clear all; close all;

N=8;
x=0:N-1;
Nbit=log2(N);  % N=8, Nbit=3

% Przestawienie probek (odwrocenie bitow numeru probki)
for n=0:N-1
    nc = n;                     % stary numer probki - skopiowanie
    m = 0;                      % nowy numer - inicjalizacja wartosci
    for k=1:Nbit                % sprawdzenie wszystkich bitow
        if(rem(n,2)==1)         % sprawdzenie bitu zerowego
        m = m + 2^(Nbit-k);     % akumulowanie wartosci "m" z nowa waga bitu
           n = n - 1;           % liczba nieparzysta --> parzysta
        end
        n=n/2;                  % przesun bity o jeden bit w prawo
    end
    y(m+1) = x(nc+1);           % skopiuj probke na nowa pozycje position
end
y, pause                        % pokaz wynik

% Seria obliczen motylkowych
Nlev=Nbit;
for lev=1:Nlev % LEVELS
    bw=2^(lev-1);               % szerokosc motylka
    nbb=2^(lev-1);              % liczba motylkow w bloku
    sbb=2^lev;                  % przesuniecie pomiedzy blokami
    nbl=N/2^lev;                % liczba blokow
    W=exp(-j*2*pi/2^lev);       % wspolczynnik korekcji
    for bu=1:nbb            % MOTYLKI
        Wb=W^(bu-1);                            % korekcja dla motylka
        for bl=1:nbl        % BLOKI
            up   = 1    + (bu-1) + (bl-1)*sbb;  % numer probki gornej
            down = 1+bw + (bu-1) + (bl-1)*sbb;  % numer probki dolnej
            temp = Wb * y(down);                % wartosc robocza
            y(down) = y(up) - temp;             % nowa wartosc gornej probki
            y(up)   = y(up) + temp;             % nowa wartosc dolnej probki
        end
    end
end
ERROR = max( abs( fft(x) - y ) ), pause
```

TEMAT #2: Różne zadania dla ambitnych studentów. Więcej o odwrotnym FFT. Obliczanie dwóch widm dwóch różnych sygnałów za pomocą jednego FFT. Algorytm FFT z podziałem w częstotliwości (DIF - Decimation In Frequency). Algorytm radix-4. Obliczanie DCT-II z użyciem FFT.

Problem 5.9 (* “Taniec z gwiazdami”: krok do przodu i krok do tyłu - więcej o odwrotnym FFT). Po wykonaniu po sobie (jeden za drugim) algorytmu prostej i odwrotnej transformacji DFT, także FFT i IFFT, powinniśmy wrócić dokładnie do tego samego sygnału (z błędem na poziomie 10^{-14} dla sygnału o amplitudzie równej 1). Napisz program odwrotnej transformacji FFT, modyfikując program z listing 5.3 albo 5.4. Progamy FFT i odwrotnego FFT powinny różnić się tylko znakiem w funkcji `exp(+-j...)`. Sprawdź poprawność Twojego programu, porównując jego wynik z wyjściem funkcji Matlaba `ifft()`. Następnie, wykonaj FFT i odwrotne FFT, wykorzystując napisane funkcje dla dowolnego sygnału, i sprawdź wynik: czy sygnał wyjściowy jest taki sam jak wejściowy? Nie zapomnij o dzieleniu przez N w Twojej funkcji implentującej IFFT.

Problem 5.10 (*(*) Dwukrotnie szybsze obliczanie widma FFT dla sygnałów o wartościach rzeczywistych). W programie z listingu 5.5 widma DFT dwóch sygnałów o wartościach rzeczywistych sa obliczane za pomocą jednego wywołania algorytmu FFT: tworzony jest sygnał zespolony, który ma pierwszy sygnał w swojej części rzeczywistej, a drugi - w części urojonej: $x(n) = x_1(n) + jx_2(n)$. Ponieważ DFT/FFT jest transformacją liniową (widmo sumy kilku sygnałów jest równe sumie widm każdego sygnału z osobna), dlatego widmo DFT nowego sygnału jest dane wzorem: $X(k) = X_1(k) + jX_2(k)$. Ponieważ widma DFT sygnałów rzeczywistych są redundantne i odznaczają się symetrią hermitowską względem punktu $k = \frac{N}{2}$ (np. $X_1(\frac{N}{2}+k) = X_1^*(\frac{N}{2}-k)$, gdzie "*" oznacza sprzężenie zespolone — patrz laboratorium dot. DFT i DtFT; w Matlabie: `X(N)=conj(X(2)), X(N-1)=conj(X(3)),...`), to z widma $X(k)$ można "odzyskać" widma sygnałów $x_1(n)$ i $x_2(n)$:

$$X_1(k) = \frac{\mathrm{Re}\{X(k)+X(N-k)\}}{2} + j\frac{\mathrm{Im}\{X(k)-X(N-k)\}}{2}, \tag{5.16}$$

$$X_2(k) = \frac{\mathrm{Im}\{X(k)+X(N-k)\}}{2} - j\frac{\mathrm{Re}\{X(k)-X(N-k)\}}{2}, \tag{5.17}$$

W poniższym programie widmo DFT pierwszego sygnału jest już odzyskane z widma FFT sumy sygnałów. Dokończ program: odzyskaj z widma FFT widmo DFT drugiego sygnału. Następnie dodaj do programu nową *funkcjonalność*: załóż, że pierwszy synał to próbki o numerach parzystych, a drugi to próbki o numerach nieparzystych tego samego sygnału, dwukrotnie dłuższego. Po odtworzeniu widma próbek parzystych i nieparzystych, zrekonstruuj widmo całego sygnału. Za tą *funkcjonalność* otrzymasz dodatkowy punkt. W tym zadaniu skorzystaj z książki [40] (patrz rozdział 9).

Listing 5.5: Obliczanie dwóch widm DFT za pomocą jednego wywołania FFT

```
% cps_05_fft4.m

clear all; close all;
N=16;

x1 = randn(1,N);      % Dwa sygnaly
x2 = randn(1,N);
x3(1:2:2*N) = x1;    x3(2:2:2*N) = x2;

X1 = fft(x1);         % Ich widma DFT
X2 = fft(x2);
X3 = fft(x3);

% Obliczenie dwoch widm dwoch sygnalow z uzyciem jednego FFT
x12 = x1 + j*x2;     % utworzenie sygnalu zespolonego
X12 = fft(x12);      % obliczenie jego widm DFT
X12r = real(X12);    % czesc rzeczywista wyniku
X12i = imag(X12);    % czesc urojona

% Rekonstrukcja widma X1 na podstawie widma X12
X1r(2:N) = (X12r(2:N)+X12r(N:-1:2))/2;  % uzycie symetrii Real(X1)
X1i(2:N) = (X12i(2:N)-X12i(N:-1:2))/2;  % uzycie asymetrii Imag(X1)
X1r(1) = X12r(1);
X1i(1) = 0;
X1rec = X1r + j*X1i;
error_X1 = max(abs( X1 - X1rec )), pause
```

```
% Rekonstrukcja widma X2 na podstawie widma X12
% ... do zrobienia
% Rekonstrukcja widma X3 na podstawie widm X1 oraz X2
% ... do zrobienia
```

Problem 5.11 (DCT-II za pomocą FFT).** Napisz program Matlaba do obliczania dyskretnej transformacji kosinusowej DCT-II sygnału, zdefiniowanej, np. w [40], jak poniżej ($c(k=0) = \sqrt{1/N}$, $c(k>0) = \sqrt{2/N}$):

$$X^{DCT}(k) = c(k) \cdot \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi(2n+1) \cdot k}{2N}\right), \qquad 0 \le k \le N-1, \tag{5.18}$$

z użyciem algorytmu FFT. Równanie łączące obie transformacje, DCT-II oraz DFT (FFT), jest następujące:

$$X^{DCT}(k) = \mathrm{Re}\left[c(k)\, e^{-j\frac{\pi k}{2N}} \cdot \mathrm{FFT}_N^{(\tilde{x}(n))}(k)\right] \tag{5.19}$$

gdzie $\mathrm{FFT}_N^{(\tilde{x}(n))}(k)$ oznacza k-ty prążek N-punktowego FFT sygnału $\tilde{x}(n)$, który jest równy:

$$\tilde{x}(n) = x(2n), \quad \tilde{x}(N-n-1) = x(2n+1), \quad n = 0,1,2,...,N/2-1. \tag{5.20}$$

Sprawdź poprawność twojej implementacji (w Matlabie jest funkcja implementuaca transformację DCT-II, znajdź ją i użyj ją także do porównania).

Problem 5.12 ((*)(**) Algorytm radix-2 DIF FFT).** Napisz uniwersalny program w języku Matlab, implementujący algorytm DIF FFT (decimation-in-frequency) z podziałem w czestotliwości - patrz rozdz. 9 w [Ziel05]. Powinien on działać dla sygnałów o długości $N = 2^p$. Zaimplementuj: 1) tylko jeden poziom dekompozycj (**), 2) dwa poziomy dekompozycji (***), 3) wszystkie poziomy dekompozycji (*****). Algorytm obliczeniowy dla pierwszych dwóch etapów dekompozycji dla $N = 8$ jest przedstawiony w [40]. W tym algorytmie oddzielnie obliczane są współczynniki DFT o numerach parzystych $X(2k)$ oraz numerach nieparzystych $X(2k+1)$, $k = 0,1,2,..\frac{N}{2}-1$. Porównaj wynik Twojego programu z wynikiem funkcji Matlaba `fft()` — w przypadku pełnego algorytmu DIF pamiętaj o przestawieniu obliczonych próbek widma (odwrócenie bitów numeru każdego obliczonego wspólczynnika Fouriera).

Problem 5.13 (*** Algorytm radix-4 DIT FFT).** Napisz uniwersalny program implementujący w języku Matlab algorytm radix-4 FFT dla dowolnej długości $N = 4^p$ wektora próbek sygnału. Skorzystaj z podejścia DIT (decimation-in-time). Wszystkie niezbędne informacje znajdź samodzielnie.

Problem 5.14 (*** Algorytm split-radix DIT FFT).** Napisz uniwersalny program implementujący w języku Matlab algorytm split-radix FFT, z użyciem podejścia DIT (decimation-in-time). Wszystkie niezbędne informacje znajdź samodzielnie. W algorytmie split-radix (przełączany-radix) sposób dekompozycji sygnału na bloki (na co: drugą, czwartą, ... próbkę) może się zmieniać na kolejnych etapach podziału.

Laboratorium 6
Zastosowania FFT

Streszczenie Podczas tego laboratorium nauczymy się jak poprawnie i efektywnie używać algorytmu szybkiej transformacji Fouriera w kilku zastosowaniach. Przećwiczymy: 1)opisywanie wyniku FFT, 2) interpolację (zagęszczanie punktów) widma FFT, 3) zastosowanie FFT do estymacji gęstości widmowej mocy sygnałów zaszumionych z użyciem metody Welcha, 4) zastosowanie FFT do śledzenia zmian częstotliwości wielu składowych sygnału z użyciem reprezentacji czasowo-częstotliwościowych: krótkoczasowej transformacji Fouriera (STFT, spectrogram) oraz transformacji Wignera. Nauczymy się także jak stosować FFT w szybkich algorytmach splotu (filtracji) i korelacji (szukania podobieństwa) sygnałów.

TEMAT #1: Podstawy wykorzystania FFT do analizy częstotliwościowej sygnałów. W tej części przećwiczymy typowe zastosowanie FFT do analizy widmowej sumy sinusoid. Zajmiemy się poprawnym wyświetlaniem i opisem osi widma FFT.

Listing 6.1: Podstawy zastosowania FFT

```
% cps_06_fftapp1_start.m
clear all; close all;          % "mycie rak"

fpr = 1000;                    % czestotliwosc probkowania (Hz)
N  = 100;                      % liczba probek sygnalu, 100 lub 1000
dt=1/fpr; t=dt*(0:N-1);        % chwile probkowania sygnalu, os czasu

% Signal
f0=50; x = sin(2*pi*f0*t);     % sygnal o czestotliwosciach f0 = 50,100,125,200 Hz
figure; plot(t,x,'bo-'); xlabel('t [s]'); title('x(t)'); grid; pause

% FFT spectrum
X = fft(x);                    % FFT
f = fpr/N *(0:N-1);            % os czestotliwosci
figure; plot(f,1/N*abs(X),'bo-'); xlabel('f [Hz]'); title('|X(k)|'); grid; pause
```

Problem 6.1 (Podstawy zastosowania FFT).** Przeanalizuj kod programu 6.1. Uruchom go. Sprawdź wartości amplitudy i częstotliwości sygnału wygenerowanego w programie i spróbuj je odczytać z rysunku widma FFT sygnału (znaleźć je na nim). Zwróć uwagę na opis osi częstotliwości oraz na "sprzężoną", hermitowską symetrię widma względem częstotliwości $\frac{f_s}{2}$ - oś symetrii ($X(\frac{f_{pr}}{2}+f_0) = X^*(\frac{f_{pr}}{2}-f_0)$, $X(\frac{N}{2}+k) = X^*(\frac{N}{2}-k)$ (w Matlabie centrum symetrii jest w punkcie `x(N/2+1)`, dlatego, np. x(N)=conj(x(2))). Wyjaśnij pochodzenie tej symetrii (przypomnij sobie szczegóły dotyczące powtarzania się funkcji bazowych w macierzy transformacji DFT). Zmień amplitudę sygnału na 10, 100, 1000 oraz obserwuj wartości modułu widma. Dlaczego wartości maksimów widma są dwa razy niższe niż spodziewałeś się? (Przypomnij sobie równość: $\cos(\alpha) = 0.5e^{j\alpha} + 0.5e^{-j\alpha}$. Jaki wzór jest dla sinusa?) Zmień wartość częstotliwości sygnału `f0` na 50, 100, 125, 200 Hz oraz obserwuj wartości argumentów maksimów widma. Czy to są te same wartości? Wytłumacz pochodzenie rozmycia widma dla sygnału o częstotliwości `f0=125` (przypomnij sobie znaczenie funkcji okien: jakiego okna teraz używamy?).

1. pokaż tylko pierwszą połowę (1/2) widma i poprawnie wyskaluj jego moduł (*2/N):
   `k=1:N/2+1; plot(f(k),2/N*abs(X(k)));`
2. zmień wartość `N` z 100 na 1000: obejrzyj widma FFT dla `f0 = 50,100,125,200` Hz;
3. wyskaluj ostatni moduł widma FFT w decybelach `X=20*log10(2/N*abs(X(k)))` oraz obejrzyj go dla `N=100` oraz `N=1000`;
4. zadeklaruj nowy wektor `w=ones(1,N)`, potem zastosuj podstawienie `x=w;` oraz wyświetl widmo FFT/DFT sygnału `x`, czyli samych jedynek w decybelach; czy takiego wyniku się spodziewałeś (tylko wartość średnia)? jak wygląda widmo DtFT (teoretyczne) tych samych danych dla gęstszego próbkowania w osi częstotliwości

niż w DFT (widoczność listka głównego i listków bocznych widma)? czym wytłumaczysz istniejącą różnicę widm DFT i DtFT (pamiętaj: w DFT zakłada się okresowość analizowanego sygnału, a w DtFT - jego zerowe wartości poza przedziałem analizy)? podstaw `w=[ones(1,N/2),zeros(1,N/2)]; x=w;` i wytłumacz obserwowany wynik;

5. ustaw `w=ones(1,N); x=w.*(sin(2*pi*50*t)+sin(2*pi*125*t))` oraz wyświetl widmo FFT sygnału dla `N=100` oraz `N=1000` w skali liniowej i decybelowej; korzystając z wyników poprzedniego punktu objaśnij dlaczego rozmycia widma raz występuje, a raz nie występuje;
6. ustaw `w=chebwin(N,100)` oraz wyświetl widmo FFT okna Czebyszewa w decybelach dla `N=100` oraz `N=1000;` (podstawiając `x=w;`)
7. ustaw `x=w.*(sin(2*pi*50*t)+sin(2*pi*125*t))` oraz wyświetl widmo FFT sygnału dla `N=100` oraz `N=1000` w skali liniowej i decybelowej; wyskaluj poprawnie widmo dla dowolnego okna innego niż prostokątne: `(*2/sum(w)))` zamiast `*2/N`;
8. ustaw `x=sin(2*pi*50*t)+0.001*sin(2*pi*125*t)` oraz wyświetl widma jak poprzednio; wytłumacz dlaczego nie jest widocza druga, słabsza składowa sygnału;
9. ustaw `w=chebwin(N,100); x=w.*(sin(2*pi*50*t)+0.001*sin(2*pi*125*t))` oraz wyświetl widma jak poprzenio; wytłumacz dlaczego druga, słabsza składowa sygnału jest teraz widoczna.

Problem 6.2 (* Interpolacja widma FFT). Przeanalizuj kod programu 6.2, będącego kontunuacją programu 6.1. Zwróć uwagę, że na końcu sygnału są wstawiane dodatkowe wartości zerowe, które sztucznie zwiększają jego długość. W wyniku tego kwadratowa macierz transformacji DFT ma większe wymiary: więcej próbek sygnału (w poziomie) i więcej wzorców częstotliwości (w pionie). W wyniku tego obliczanych jest więcej prążków DFT, pokrywających jednak ten sam zakres $[0, f_{pr})$, czyli próbkujących widmo gęściej ($\Delta f = \frac{f_s}{N}$, teraz N jest większe). Ustaw `N=100; x=sin(2*pi*50*t)+0.001*sin(2*pi*175*t)` oraz wybierz `w=w1`. Uruchom program. Zwróć uwagę na końcowy rysunek, pokazujacy dwa widma FFT: 1) bez dodania zer na końcu sygnału oryginalnego (`X`), 2) z dodaniem zer (`Xz`). Drugie widmo jest spróbkowane `K`-razy gęściej: jest ono interpolowaną wersją widma pierwszego. Sprawdź to: porównaj krok próbkowania częstotliwości (Δf) w obu przypadkach. Przełącz na `w=w2` oraz zapoznaj się z rysunkiem widma. Ustaw `x=ones(1,N)`. Potem najpierw wybierz `w=w1`, a za drugim razem `w=w2`. Powiedz co teraz widzisz w widmie sygnału? Może *listek główny* i *listki boczne* widma DFT funkcji okna? Dodaj kilka innych funkcji okien do programu. Porównaj widma `X` oraz `Xz` dla różnych wartości stopnia nadpróbkowania (interpolacji) widma: `K=10, 8, 6, 4, 2.`

Listing 6.2: Interpolacja widma FFT poprzez dodanie zer na końcu analizowanego sygnału

```
% ... kontynuacja programu - czesc 2
% Interpolacja widma FFT z okienkowaniem sygnalu
K = 10;                  % rzad interpolacji
w1 = rectwin(N);         % okno prostokatne
w2 = chebwin(N,100);     % okno Czebyszewa
w = w1;                  % wybor okna: w1, w2, ...
x = x.*w';               % okienkowanie sygnalu
X  = fft(x,N);           % bez dolaczenia zer na koncu sygnalu
Xz = fft(x,K*N);         % z zerami; Xz = fft([x,zeros(1,(K-1)*N)])/sum(w);
fz = fpr/(K*N)*(0:K*N-1); % os czestotliwosci
figure                   %
plot(f,20*log10(abs(X)/sum(w)),'bo-',fz,20*log10(abs(Xz)/sum(w)),'r.-','MarkerFaceColor','b');
xlabel('f (Hz)'); title('Zoomowanie widma DFT z uzyciem FFT'); grid; pause
```

TEMAT #2: Zastosowanie FFT dla sygnałów zaszumionych: estymacja gęstości widmowej mocy. Widma FFT sygnałów zaszuminych są "zaszumione". W celu redukcji wariancji wartości widma, długi wektor próbek sygnału jest dzielony na wiele krótszych wektorów, częściowo nakładających się lub nie. Następnie jest obliczanych wiele krótkich widm FFT dla każdego krótkiego wektora próbek z osobna, a na ich podstawie jest wyznaczane jedno "średnie" widmo dla całego sygnału. Po odpowiednim wyskalowaniu otrzymujemy estymatę funkcji gęstości widmowej mocy sygnału (PSD - Power Spectral Density).

Problem 6.3 (* Estymata PSD wykorzystująca FFT: metoda Welcha). Przeanalizuj kod programu 6.3 oraz zaobserwuj jak jest obliczana estymata Welcha funkcji widmowej gęstości mocy sygnału, tzn. wektor `X2`: widma amplitudowe `X=fft(bx,Mfft)/sum(w)` są podnoszone do kwadratu, sumowane i na końcu odpowiednio skalowane: `(1/Many)*(1/fpr)`. Ustaw `x=x1+0.5*randn(1,N)`. Uruchom program oraz obejrzyj dwa obliczone widma FFT sygnału: `X1` and `X2`. Dlaczego są one różne? Następnie dodaj rysunek, który wyświetli w `k`-tej iteracji pętli głównej kolejną estymatę widma mocy `X2`: (`plot(f,X2); semilogy(f,X2)`. Zaobserwuj jak ta estymata staje się coraz gładsza (mniej zaszumiona) po dodaniu do niej kwadratu ostatnio obliczonego widma FFT: `X2=X2+abs(X).^2`. Oblicz jedno widmo amplitudowe dla całego sygnału, podnieś go do kwadratu, podziel przez f_{pr} (czyli odpowiednio wyskaluj) oraz porównaj go na jednym rysunku z widmem `X2`, już po wyskalowaniu na końcu programu (użyj funkcji `semilogy()`). Porównaj wynik z funkcją Matlaba `X=pwelch(x,...)`. Napisz swoją funkcję `X=mypwelch(x,...)`.

Listing 6.3: Zastosowanie FFT do analizy częstotliwościowej sygnałów zaszumionych (*periodogram*) oraz mających składowe o częstotliwościach zmiennych w czasie (*spectrogram*)

```
clear all; close all;

fpr = 8000;              % czestotliwosc probkowania (Hz)
T  = 3;                  % czas trwania sygnalu w sekundach
N  = round(T*fpr);       % liczba probek, 100 albo 1000
dt=1/fpr; t=dt*(0:N-1);  % os czasu
n = 1:1000;              % indeksy probkek sygnalu dla rysunkow


% Sygnal
x1 = sin(2*pi*200*t) + sin(2*pi*800*t);                              % 2xSIN
x2 = sin( 2*pi*( 0*t + 0.5*((1/T)*fpr/4)*t.^2 ) );                   % LFM
fm=0.5; x3 = sin(2*pi*((fpr/4)*t - (fpr/8)/(2*pi*fm)*cos(2*pi*fm*t)));  % SFM
x = x1 + 0.5*randn(1,N);                                             % wybor
figure; plot(t(n),x(n),'b-'); xlabel('t [s]'); title('x(t)'); grid; pause  % rysunek

% Widmo FFT
Mwind = 256; Mstep=16; Mfft=2*Mwind;  Many = floor((N-Mwind)/Mstep)+1;
t = (Mwind/2+1/2)*dt + Mstep*dt*(0:Many-1);     % czas
f = fpr/Mfft*(0:Mfft-1);                        % czestotliwosc
w = hamming( Mwind )';                          % wybor okna
X1 = zeros(Mfft,Many); X2 = zeros(1,Mfft);      % inicjalizacja STFT i PSD
for m = 1 : Many                                % petla analizy
    bx = x( 1+(m-1)*Mstep : Mwind+(m-1)*Mstep );  % kolejny fragment sygnalu
    bx = bx .* w;                               % okienkowanie
    X = fft( bx, Mfft )/sum(w);                 % FFT ze skalowaniem
    X1(1:Mfft,m) = X;                           % <--- ! STFT
    X2 = X2 + abs(X).^2;                        % <--- ! Welch PSD
end                                             % konie petli
X1 = 20*log10( abs(X1) );                       % przeliczenie na decybele
X2 = (1/Many)*X2/fpr;                           % normalizacja PSD
% spectrogram(x,Mwind,Mwind-Mstep,Mfft,fpr); pause  % STFT Matlab

figure;
imagesc(t,f,X1);        % macierz widma amplitudowego jako obraz
c=colorbar; c.Label.String = 'V (dB)'; ax = gca; ax.YDir = 'normal';
xlabel('t (s)'); ylabel('f (Hz)'); title('STFT |X(t,f)|'); pause
figure;
semilogy(f,X2); grid; title('PSD Welcha'); xlabel('f [Hz]'); ylabel('V^2 / Hz'); pause
```

Problem 6.4 (Szukanie igły w stogu siana: jedno widmo mocy FFT dla długiego sygnału CZY jedno ŚREDNIE widmo mocy obliczone z wielu widm FFT dla krótkich fragmentów tego samego sygnału).** Wygeneruj sygnał sinusa. Dodaj do niego słaby szum o rozkładzie Gaussa — skorzystaj z funkcji `randn()`. Oblicz i pokaż widmo mocy sygnału (widmo amplitudowe FFT podniesione do kwadratu — identycznie jak podczas obliczania `X2` w programie 6.3). Wyskaluj poprawnie widmo w częstotliwości (Hz) i mocy (V^2). Potem stopniowo zwiększaj odchylenie standardowe szumu, obserwuj widmo i zatrzymaj się, kiedy maksimum widma, związanego z sygnałem sinusa, przestanie być widoczne. Następnie znacznie zwiększ długość sygnału, podziel go na mniejsze fragmenty, oblicz widmo mocy dla każdego fragmentu z osobna, a na koniec – wartość średnią wszystkich widm. Wyznacz ile widm należy uśrednić, aby maksimum sinusa było dobrze widoczne w uśrednionym widmie.

TEMAT #3: Zastosowanie FFT dla sygnałów majacych składowe o zmiennych częstotliwościach: krótkoczasowa-transformacja Fouriers (STFT) i Wignera. Kiedy sygnał posiada składowe o częstotliwościach zmieniajacych się w czasie, to jesteśmy zainteresowani obserwacją tych zmian we współczynnikach widm FFT kolejnych fragmentów sygnału. Dlatego dzielimy sygnał na krótkie fragmenty, obliczmy ich widma FFT i pokazujemy jedno za drugim. Alternatywnie składamy wszystkie widma do jednej macierzy czasowo-częstotliwościowej $|X(t,f)|$ i wyświetlamy jej zawartość (funkcje Matlaba: `mesh()`, `imagesc()`).

Problem 6.5 (* STFT czyli spectrogram FFT sygnałów zmiennych w czasie). Przeanalizuj kod programu 6.3 oraz zaobserwuj jak jest wyznaczana i pokazywana w nim macierz krótko-czasowej transformacji Fouriera (STFT) $|X(t,f)|$ (`X1` w programie). Ustaw `x=x2` (sygnał LFM) oraz `x=x3` (sygnał SFM). Uruchom program. Obejrzyj kolorowy obraz czasowo-częstotliwościowej macierzy STFT. Spróbuj odczytać z niego informację, dotyczącą zmian częstotliwości analizowanego sygnału w funkcji czasu: od-do w hercach dla sygnału LFM `x2` oraz częstotliwość środkowa oraz głębokość modulacji dla sygnału SFM `x3`. Podczas analizy sygnału SFM `x=x3` użyj okien o różnej długości `Mwind`: bardzo krótkich (zła rozdzielczość częstotliwościowa - rozmycie w osi częstotliwości) oraz bardzo długich (zła rozdzielczość czasowa - rozmycie w osi czasu) — zauważ znacząco różne kształty widm w obu przypadkach. Napisz swoją własną funckcję `myspectrogram()`, mającą takie same parametry wywołania jak funkcja Matlaba `spectrogram()`.

Problem 6.6 (Zastosowanie FFT dla sygnałów LFM: czasowo-częstotliwościowa reprezentacja Wignera).** Przeanalizuj kod programu 6.4: jest w nim generowany sygnał o wartościach zespolonych z liniową modulacją częstotliwości (LFM). Następnie sygnał ten jest: 1) odwracany w czasie, 2) sprzęgany (negacja części urojonej) oraz 3) mnożony przez sygnał oryginalny — operacja tzw. jądra Wignera. Na wyniku jest wykonywane FFT. Zauważ, że `xx` jest sygnałem zespolonym o jednej, stałej częstotliwości. Wytłumacz dlaczego tak się dzieje? Udowodnij to matematycznie. Jaką częstotliwość ma sygnał w połowie czasu trwania (środku)? Z środka sygnału? Potnij sygnał na krótsze, nakładające się fragmenty o długości `M=128` próbek. Potem: 1) oblicz jądro Winera dla każdego fragmentu sygnału (odwrócenie w czasie, sprzężenie zespolone i iloczyn z oryginałem), 2) wykonaj FFT na próbkach każdego jądra Wignera, 3) zbierz wszystkie moduły widm FFT w jednej macierzy, 4) wyświetl tę macierz, skalując osie czasu i częstotliwości (podobnie jak w przypadku spektrogramu STFT), 5) znajdź w niej zmianę częstotliwości sygnału. Czy obserwowana zmiana częstotliwości jest poprawna - identyczna z zadaną? Jeśli nie, to popraw skalowanie osi częstotliwości - może podzielenie przez 2, wynikające z pomnożenia sygnału oryginalnego i odwróconego w czasie oraz skutkujące podwojeniem częstotliwości? Wyprowadź. Następnie wygeneruj sygnał zespolony z sinusoidalną modulacją częstotliwości (SFM), w identyczny sposób oblicz jego macierz czasowo-częstotliwościową, wyświetl tę macierz oraz zinterpretuj obserwowane widmo — jak zmienia się chwilowa częstotliwość sygnału? Porównaj wynik z funkcją Matlaba `spectrogram()/pspectrogram()`.

Listing 6.4: Zastosowanie FFT dla sygnałów z liniową modulacją częstotliwości (LFM)

```
% cps_06_fftapp3_wigner.m
clear all; close all;

N  = 1000;                                 % liczba probek sygnalu
fpr = 1000;                                % czestotliwosc probkowania (Hz)
dt=1/fpr; t=dt*(0:N-1);                    % os czasu
x = exp( j*2*pi*(0*t + 0.5*(fpr/2)*t.^2)); % sygnal LFM
spectrogram(x,128,128-16,256,fpr); pause   % STFT sygnalu
xx = x .* conj(x(end:-1:1));               % jadro Wignera dla sygnalu x(n)
X = fft( xx )/N;                           % FFT
stem( abs( X ) );                          % detekcja tylko jednej czestotliwosci
```

TEMAT #4: Szybki splot i szybka korelacja sygnałów z użyciem FFT. Kiedy dwa sygnały są splatane, to ich widma częstotliwiściowe Fouriera mnożą się przez siebie - patrz tabela 4.2. Ta właściwość transformacji Fouriera jest wykorzystywana do szybkiej realizacji operacji splotu dwóch sygnałów, np. `x` oraz `h`: 1) obliczane są widma FFT dwóch sygnałów, 2) następnie są one mnożone, 3) a na wyniku jest wykonywane IFFT, czyli następuje powrót do dziedziny czasu, w skrócie:

```
y = ifft( fft(x) .* fft(h) ); % y=conv(x,h);
```

To samo robi funkcja Matlaba `conv()`. Ponieważ korelacja wzajemna dwóch sygnałów jest zdefiniowana bardzo podobnie jak splot dwóch sygnałów (różnica w funkcji korelacji: drugi sygnał nie jest odwracany w czasie tylko sprzęgany), dlatego szybkie algorytmy splotu można, po drobnej modyfikacji, zastosować do szybkiego obliczania funkcji korelacji.
Jest jednak pewien problem. Aby zastosować szybki splot, to przed użyciem funkcji FFT należy oba sygnały uzupełnić zerami. Jeśli przyjmiemy, że sygnał x ma N próbek, a sygnał h ma M próbek, to do pierwszego z nich należy dodać $M-1$ próbek, a do drugiego $N-1$. Dlaczego? Poszukaj odpowiedzi w podrozdziałach 13.4 i 13.5 książki [40]. Tam też znajdziesz odpowiedź na pytanie co i jak należy zrobić, jeśli jeden z sygnałów jest ba...rdzo długi, albo nawet gorzej: jego próbki ciągle przypływają z przetwornika A/C. Remedium jest tzw. sekcjonowany szybki splot. No proszę: i znowu coś się chowa za horyzontem.

Listing 6.5: Szybki splot sygnałów z użyciem FFT

```
% cps_06_fftapps4_splot.m
clear all; close all;

  sig = 1;    % 1/2, sygnal: 1=krotki, 2=dlugi
  if(sig==1)  N=5;    M=3;   x = ones(1,N);  h = ones(1,M);  % sygnaly
  else        N=256;  M=32;  x = randn(1,N); h = randn(1,M); % splatane
  end
  n = 1:N+M-1; nn = 1:N; % indeksy probek sygnalow
  figure;
  subplot(211); stem(x); title('x(n)');
  subplot(212); stem(h); title('h(n)'); pause

% Splot z uzyciem funkcji Matlaba
  y1 = conv(x,h);
        figure; stem(y1); title('y1(n)'); pause

% Szybki splot - poczatek wyniku jest niepoprawny!
  hz = [ h zeros(1,N-M) ];              % dolacz N-M zer tylko na koncu sygnalu krotkiego
  y2 = ifft( fft(x) .* fft(hz) );       % szybki splot, pierwszych M-1 probek jest zlych
  error2 = max(abs(y1(M:N)-y2(M:N))), pause
  figure; plot(nn,y1(nn),'ro',nn,y2(nn),'bx'); title('y1(n) & y2(n)');

% Szybki splot - wszystkie probki wyniku sa poprawne!
  hzz = [ h zeros(1,N-M) zeros(1,M-1) ];  % uzupelnij zerami do dlugosci N+M-1
  xz = [ x zeros(1,M-1) ];                % uzupelnij zerami do dlugosci N+M-1
  y3 = ifft( fft(xz) .* fft(hzz) );       % szybki splot, wszystkie probki sa dobre
  error3 = max(abs(y1-y3)), pause
  figure; plot(n,y1,'ro',n,y3,'bx'); title('y1(n) & y3(n)');

% Szybki splot z podzialem na czesci - metoda OVERLAP-ADD
  if( sig > 1 )                          % tylko dla dlugiego sygnalu
  L = M;                                 % dlugosc fragmentu sygnalu
  K = N/L;                               % liczba fragmentow sygnalu
  hzz = [ h zeros(1,L-M) zeros(1,M-1) ]; % uzupelnienie zerami odp. impulsowej filtra
  Hzz = fft(hzz);                        % FFT odpowiedzi impulsowej filtra
  y4 = zeros(1,M-1);                     % inicjalizacja sygnalu wynikowego
  for k = 1:K                            % PETLA
      m = 1 + (k-1)*L : L + (k-1)*L;     % indeksy fragmentu sygnalu
```

```
    xz = [ x(m) zeros(1,M-1) ];           % pobranie fragmentu sygnalu, dolozenie zer
    YY = fft(xz) .* Hzz;                  % # szybki splot - iloczyn widm FFT
    yy = ifft( YY );                      % # odwrotne FFT
    y4(end-(M-2):end) = y4(end-(M-2):end) + yy(1:M-1); % wynik: czesc overlap-add
    y4 = [ y4, yy(M:L+M-1) ];                          % wynik: czesc uzupelnij
  end
  error4 = max(abs(y1-y4)), pause
  figure; plot(n,y1,'ro',n,y4,'bx'); title('y1(n) & y4(n)');
  end
```

Problem 6.7 (* Splot liniowy, splot kołowy, szybki splot z użyciem FFT). Przeanalizuj kod programu 6.5. Ustaw `sig=1`: spleciesz 3-szy jedynki z 5-cioma jedynkami. Zwróć uwagę na kształty sygnałów `y1, y2, y3`, czyli na: 1) 7-elementowy wynik splotu liniowego, obliczony metodą: *przesuń, mnóż, dodawaj*, 2) 5-elementowy wyniku splotu kołowego, obliczony z użyciem 5-punktowego FFT (dodajemy tylko dwa zera do krótszego sygnału), oraz 3) 7-elementowy wynik splotu kołowego, obliczony z użyciem 7-punktowego FFT, dający wynik identyczny jak 1) (dodajemy 2 zera do dłuższego sygnału i 4 do krótszego). Sprawdź wartości błędów: `error2` oraz `error3`. Potem ustaw `sig=2` oraz zapoznaj się ponownie z sygnałami wynikowymi oraz wartościami błędów. Następnie dodaj do programu swoje własne dwa sygnały `x` oraz `h`, inne niż poprzednio zdefiniowane, oraz porównaj obliczone sygnały `y1,y2,y3`: 1) wszystkie próbki sygnału `y3` powinny być takie same jak sygnału `y1`, 2) próbki `(M:1:N)` sygnałów `y1` oraz `y2` powinny być identyczne.

Problem 6.8 (Szybki sekcjonowany splot OVERLAP-ADD oraz OVERLAP-SAVE).** Przeanalizuj kod programu 6.5. Obecnie koncentrujemy się na metodzie OVERLAP-ADD, opisanej w książce [40] (rozdz. 13.5). Ustaw `sig=2`, uruchom program oraz zapoznaj się z sygnałem `y4` oraz wartością błędu `error4`. Obecnie szybki splot, wykorzystujący FFT, jest wykonywany nie na całym sygnale, tylko na jego częściach: poszczególne wyniki splotu są razem łączone w jeden sygnał wyjściowy. Dodaj do programu kod implementujący metodę szybkiego splotu OVERLAP-SAVE (z książki lub z ukochanego Internetu): oblicz sygnał `y5` oraz błąd `error5`.

Problem 6.9 (* Szybkie obliczanie DtFT z użyciem FFT: transformacja Chirp-Z). Czasami jesteśmy zainteresowani oliczeniem tylko pewnej wybranej części widma DtFT: od częstotliwości `f1` do częstotliwości `f2` z krokiem `df`. W takim przypadku jest zwykle wykorzystywana transformacja Chirp-Z (CZT), czyli "świergotowa", która wykorzystuje FFT. Znajdź opis transformacji CZT w książce [40] (rozdz. 9.4) oraz program Matlaba, który ją implementuje. Przeanalizuj kod programu. Uruchom program. Jaki jest błąd pomiędzy szybką (z FFT) oraz wolną (bez FFT) implementacją obliczania widma DtFT dla wybranego zakresu częstotliwości? Powtórz eksperyment dla różnych zakresów częstotliwości i różnych sygnałów.

Problem 6.10 (* Szybkie obliczanie funkcji korelacji z użyciem FFT). Definicje splotu dwóch sygnałów oraz korelacji wzajemnej dwóch sygnałów są do siebie bardzo podobne:

$$z(n) = \sum_{n=-\infty}^{\infty} x(n)y(k-n) \tag{6.1}$$

$$Rxy(k) = \sum_{n=-\infty}^{\infty} x(n)y^*(n-k) \tag{6.2}$$

Drugi sygnał jest przesuwany w nich obu i mnożony z sygnałem pierwszym. ALE ... w pierwszej definicji, czyli splotu, sygnał ten jest odwracany w czasie, a w dugiej nie, ale za to jest sprzęgany (dla liczb zespolonych). Z tego powodu szybkie algorytmy i programy opracowane dla splotu, mogą zostać także zastosowane w przypadku obliczania korelacji wzajemnej: `rxy = conv(x,conj(y(end:-1:1)))`. Zweryfikuj tę koncepcję praktycznie z użyciem różnych sygnałów. Uzupełnij program 6.6. Bądź czujny: `error5` sugeruje, że może znajdziej go jako kontynuację programu 6.5.

Listing 6.6: Szybka korelacja dwóch sygnałów z użyciem FFT

```
% ... tutaj generujemy sygnaly x oraz h
```

```
% Szybka korelacja wzajemna dwoch sygnalow z uzyciem FFT
 R1 = xcorr( x, h );
 R2 = conv( x, conj( h(end:-1:1) ) );
 Kmax=max(M,N); Kmin=min(M,N); R2 = [ zeros(1,Kmax-Kmin) R2 ];
 m = -(Kmax-1) : 1 : (Kmax-1);
 figure; plot(m,R1,'ro',m,R2,'bx'); title('R1(n) & R2(n)');
 error5 = max( abs( R1-R2 ) ), pause
```

Problem 6.11 (* Szybka implementacja metody Blackmana-Tukeya, służącej do estymacji gęstości widmowej mocy (PSD) sygnału). Estymacja PSD metodą Blackmana-Tukeya (BT-PSD) jest zdefiniowana jako DFT, czyli u nas FFT, funkcji auto-korelacji sygnału. Dysponując szybkim algorytmem do obliczania funkcji korelacji, możemy go użyć do szybkiego obliczania BT-PSD. W rozdziale 8.7 książki [40], podano szczegóły matematyczne oraz program implementujący metodę BT-PSD. W programie PSD sygnału jest obliczane metodą Welcha oraz Blackmana-Tukeya, zaś otrzymane wyniki są porównane ze sobą. Uruchom program, zapoznaj się z kształtami funkcji PSD. Zwiększaj stopniowo poziom szumu i obserwuj zmianę kształtu obliczonego PSD. Zmień analizowany sygnał na inny. Użyj programu do analizy zaszumionych dźwięków pobranych ze strony *FindSounds*. Sam utwórz takie dźwieki dodając do siebie różne sygnały wzięte z *FindSounds* oraz szum wygenerowany w Matlabie (`randn()`, `y=awgn(x,SNR,'measured')`).

TEMAT #5: Przykłady analizy FFT sygnałów rzeczywistych. W tej części skoncentrujemy się na konkretnych przykładach zastosowań.

Problem 6.12 (* Wyznaczenie parametrów sumy tłumionych sinusoid, czyli IpDFT z użyciem FFT).** Bardzo ważna jest praktyczna umiejętność wyznaczania parametrów sinusoid tłumionych eksponencjalnie:

$$x(n) = A\cos(\Omega n + \phi)e^{-dn}, \qquad n = 0, 1, 2, ..., N-1, \tag{6.3}$$

szczególnie wartości ich częstotliwości radialnej $\Omega = 2\pi\frac{f}{f_{pr}}$ i tłumienia d. Dlaczego? Ponieważ takie sygnały są generowane przez elektryczne (np. układy RLC) i mechaniczne (np. wahadło) oscylatory tłumione, np. także w urządzeniach rezonansu magnetycznego. W tym przypadku jesteśmy mądrzy, gdyż znamy dokładnie równanie analizowanego sygnału, a konsekwencji — równanie jego widma Fouriera. Jeśli weźmiemy kilka prążków w okolicy maksimum modułu widma, to możemy zbudować i rozwiązać układ równań, i w ten sposób wyznaczyć wartości interesujących nas parametrów: częstotliwości i tłumienia. W jednej z metod, Bertocco-Yoshidy, bierze się wartości trzech prążków N-punktowego FFT w okolicy maksymalnej wartości bezwzględnej widma $X(k-1), X(k), X(k+1)$, a następnie oblicza się kolejno wartości R, r, λ, a z nich szukane wartości częstotliwości radialnej Ω, tłumienia d, amplitudy A oraz przesunięcia fazowego ϕ analizowanego sygnału $x(n)$ (6.3) (poniżej $\Omega_k = 2\pi\frac{kf_0}{f_{pr}}$ oznacza częstotliwość k-tego prążka widma):

$$R = \frac{X(k-1) - X(k)}{X(k) - X(k+1)}, \tag{6.4}$$

$$r = \frac{-e^{-j\Omega_k} + e^{-j\Omega_{k-1}}}{-e^{-j\Omega_{k+1}} + e^{-j\Omega_k}}, \tag{6.5}$$

$$\lambda = e^{j\Omega_k}\frac{r - R}{re^{-j2\pi/N} - Re^{j2\pi/N}}, \tag{6.6}$$

$$\Omega = \mathrm{Im}(\ln(\lambda)), \tag{6.7}$$

$$d = -\mathrm{Re}(\ln(\lambda)), \tag{6.8}$$

$$c = \frac{1 - \lambda^N}{1 - \lambda e^{-j\Omega_k}}, \tag{6.9}$$

$$A = |2X(k)/c|, \tag{6.10}$$

$$\phi = \mathrm{angle}\,(2X(k)/c)\,. \tag{6.11}$$

Zauważmy, że opisana powyżej procedura powinna być zastosowana do każdego znalezionego maksimum widma FFT. Zapoznaj się z programem 6.7 — znajdź w nim napisane powyżej wzory. Znajdź także zależności do wyznaczania wartości A oraz ϕ, niepodane powyżej. Zapoznaj się z rysunkami. Czy obliczone wartości amplitudy, częstotliwości, kąta fazowego oraz tłumienia są poprawne? Powinny być. Zmień wartości parametrów, zadane na początku programu, i ponownie sprawdź wynik obliczeń. Pobierz ze strony *FindSounds* nagranie pojedynczego akordu pianina lub gitary albo dźwięku trąbki lub fletu. Wczytaj je do programu, usuń fragmenty ciszy z początkowej i końcowej części nagrania. Oblicz FFT dla pozostałych próbek oraz wybierz trzy współczynniki FFT, leżące wokół jakiegoś lokalnego maksimum widma. Pierwszego? Najwyższego? Najbardziej oddalonego od innych? Jakie wartości parametrów wybranej składowej sygnału zwrócił program? Zsyntezuj sygnał sinusoidy tłumionej eksponencjalnie, mającej takie parametry. Narysuj go na tle sygnału oryginalnego.

Listing 6.7: Implementacja w języku Matlab algorytmu interpolowanego DFT według metody Bertocco-Yoshidy

```
% sps_06_fftapps5_ipdft.m  - interpolowane DFT/FFT
clear all; close all;

% Sygnal testowy
N=256; fs=100;              % liczba probek, czestotliwosc probkowania [Hz]
Ax=6; dx=0.5; fx=4; px=3;   % amplituda, tlumienie, czestotliwosc, faza
dt=1/(fs); t=(0:N-1)*dt;    % chwile probkowania
x = Ax*exp(-dx*t).*cos(2*pi*fx*t+px); figure; plot(t,x); pause % generacja sygnalu

% Interpolowane DFT dla maksimum wartosci bezwzglednej widma
Xw = fft(x); figure; plot(abs(Xw)), pause  % obliczenie DFT/FFT
[Xabs, ind] = max(abs(Xw(1:round(N/2))));  % znalezienie maksimum i jego polozenia
km1 = ind-1; k=ind; kp1 = ind+1;           % trzy probki w otoczeniu maksimum
dw = 2*pi/N;                % krok czestotliwosci w DFT
wkm1= (km1-1)*dw;               % czestotliwosc katowa dla indeksu k-1
wk  = (k  -1)*dw;               % czestotliwosc katowa dla indeksu k
wkp1= (kp1-1)*dw;               % czestotliwosc katowa dla indeksu k+1
r = ( -exp(-j*wk)+exp(-j*wkm1) )/( -exp(-j*wkp1)+exp(-j*wk) ); % eq. ()
R = ( Xw(km1)-Xw(k) )/( Xw(k)-Xw(kp1) );                       % eq. ()
lambda = exp(j*wk)*(r-R)/( r*exp(-j*2*pi/N)-R*exp(j*2*pi/N) ); % eq. ()
we = imag(log(lambda));     % obliczona czestotliwosc katowa
de = -real(log(lambda));    % obliczone tlumienie

fe = we*fs/(2*pi);          % cz. katowa --> czestotliwosc
de = de*fs;                 % znormalizowane tlumienie (de/fs) --> tlumienie (de)

if round(1e6*R)==-1e6       % probkowanie KOHERENTNE, dx=0
   Ae = 2*abs(Xw(k))/N;     % obliczona amplituda
   pe = angle(Xw(k));       % obliczona faza
else                        % probkowanie NIEKOHERENTNE
   c = (1-lambda^N)/(1-lambda*exp(-j*wk));                     % eq. ()
   c = 2*Xw(k)/c;                                              % eq. ()
   Ae = abs(c);             % obliczona amplituda
   pe = angle(c);           % obliczona faza
end

result = [ Ae, de, fe, pe ],                    % wyniki
errors = [ Ae-Ax, de-dx, fe-fx, pe-px ], pause  % bledy
```

Problem 6.13 (Sygnały z huty — raz jeszcze).** Wykorzystaj swoją obecną wiedzę do analizy częstotliwościowej nagranych sygnałów napięcia i prądu pracującego pieca łukowego. Były już one wcześniej wykorzystywane podczas laboratorium DFT/DtFT. Wczytaj sygnały ze zbioru `load('UI.mat'); whos`. Na początku, oblicz widmo amplitudowe FFT całego sygnału (`X=fft(x)`), następnie estymatę PSD sygnału metodą Welcha (`X=pwelch(x,...)`), na końcu czasowo-częstoliwościowy spektrogram sygnału (`spectrogram(x,...)`). Porównaj ze sobą wszystkie widma. Spróbuj odczytać z rysunków wartość częstotliwości podstawowej zasilania 50 Hz oraz jej harmonicznych 100, 150, 200, 250, ... Hz.

Problem 6.14 (Spektrogram dźwięku pianina: w poszukiwaniu źródeł piękna).** W jednym z poprzednich ćwiczeń generowaliśmy dźwięki przypominające pojedyncze akordy pianina. Obecnie zastosuj swój własny program `spectrogram(), pspectrogram()` krótkoczasowej transformacji Fouriera oraz dokonaj analizy nagranego dźwięku rzeczywistego pianina: spróbuj znaleźć jakie częstotliwości są kolejno generowane przez instrument. Aby lepiej to zaobserwować, odpowiednio dobierz wartości: długości okna analizy, przesunięcia okna oraz długości FFT (uwzględniającej ewentualne dodanie zer na końcu fragmentu sygnału). Pobierz nagranie pianina z Internetu, np. ze strony *FindSounds*. Spróbuj zsyntezować podobne nagranie muzyczne na podstawie informacji z przeprowadzonej analizy dźwięku.

Problem 6.15 (Spektrogram mowy: detekcja stanu emocjonalnego osoby mówiącej).** Częstotliwość ruchu strun głosowych, ich otwierania i zamykania, zależy od intonacji i stanu emocjonalnego mówcy, i jest wyższa w stanach ekscytacji. Nagraj kilka razy to samo słowo, systematycznie zwiekszając poziom emocji (zdziwienia, strachu, radości,...). Następnie oblicz i wyświetl spektrogram każdego nagrania (`spectrogram(x,...)`). Porównaj spektrogramy pomiędzy sobą: czym się one różnią? Jeśli są takie same, to "jesteś zimny jak lód". Aby lepiej zaobserwować różnice, odpowiednio dobierz wartości: długości okna analizy, przesunięcia okna oraz długości FFT (uwzględniającej ewentualne dodanie zer na końcu fragmentu sygnału).

Problem 6.16 (Czy mam "złamane" serce?).** Zastosuj widmo FFT oraz szybki algorytm obliczania funkcji korelacji z użuciem FFT do wyznaczenia liczby uderzeń na minutę czyjegoś (Twojego?) serducha. Użyj nagrania EKG wykorzystywanego podczas laboratorium 1 oraz 2.

Problem 6.17 ((***) Między nami, Jaskiniowcami - artystyczny pogłos).** I na koniec coś bardzo prostego, ale efektownego. W sprzęcie muzycznym są używane procesory DSP, które odpowiadają za "efekty" dźwiękowe. W skrócie: polega to na wykonaniu operacji splotu granego utworu z wcześniej zarejestrowaną odpowiedzią impulsową (wagami filtra) jakiegoś pomieszczenia. W wyniku tego mamy iluzję, że koncert odbywa się właśnie w tym pomieszczeniu. Ponieważ pomieszczenia są zazwyczaj duże, dla $f_{pr} = 44100$ Hz filtry mają kilkadziesiąt tysięcy wag i operacja splotu w dziedzinie czasu jest bardzo niefektywna: lepiej to zrobić w sposób szybki za pomocą FFT. Dlatego:

1. wczytaj do Matlaba nagraną odpowiedź impulsową jakieś pomieszczenia, np. `[h,fpr]=audioread('impulse_LargeConcertHall.wav')`: $f_{pr} = 44.1$ kHz, 2 kanały, zastanów się jak rozwiązać ten "problem" — wykorzystać tylko jeden czy dwa kanały?
2. pobierz z Internetu jakiś fragment muzyczny `x`, spróbkowany z taką samą częstotliwością, lub sam coś nagraj, może własną mowę,
3. wykonaj algorytm szybkiego splotu (`y=ifft( fft(hz) .* fft(xz) )`: uzupełnij oba sygnały zerami na końcu, zadbaj o ich taką samą długość (możesz nagranie skrócić jeśli komputer ma mało pamięci) oraz o taką samą orientację (poziomą lub pionową), zapanuj nad liczbą kanałów,
4. odsłuchaj oryginał `x` i wynik końcowy `y` całej operacji, np. `sound(x,fpr)`.

Jeśli napiszesz program, który będzie odtwarzał w czasie rzeczywistym bardzo długie utwory (na bieżąco doczytujący próbki dźwięku z dysku i wykonujący operację sekcjonowanego szybkiego splotu), to otrzymasz dodatkowe trzy punkty.

AGH

AGH

Laboratorium 7
Filtry analogowe

Streszczenie Podczas tego laboratorium nauczymy się jak projektować (dobierać) współczynniki transmitancji $H(s)$ filtrów analogowych - w celu otrzymania charakterystyk częstotliwościowych ($H(f), s = j\omega = j2\pi f$) następujących filtrów analogowych: dolno-przepustowego, górno-przepustowego, pasmowo-przepustowego oraz pasmowo-zaporowego. Będziemy projektować filtry analogowe intuicyjną metodą "Zer i Biegunów" oraz z wykorzystaniem metod: Butterwortha, Czebyszewa i Cauera (filtry eliptyczne).

TEMAT #1: Transmitancja i charakterystyka częstotliwościowa filtrów analogowych. W filtrze analogowym sygnał wejściowy ($x(t)$) oraz wyjściowy ($y(t)$) są powiązane ze sobą następujicym równaniem różniczkowym:

$$b_0 x(t) + b_1 \frac{dx(t)}{dt} + \ldots + b_M \frac{dx^M(t)}{dt^M} = y(t) + a_1 \frac{dy(t)}{dt} + \ldots + a_N \frac{dy^N(t)}{dt^N}, \tag{7.1}$$

w którym współczynniki b_k oraz a_k zależą od wartości użytych elementów pasywnych (R - rezystorów, L - cewek indukcyjnych, C - kondensatorów). Po pierwsze, obliczając transformację Laplace'a zmiennej zespolonej s (dla $s = j\omega = j2\pi f$ otrzymujemy ciągłą transformację Fouriera):

$$L(x(t)) = X(s) = \int_{-\infty}^{+\infty} x(t) e^{-st} dt, \qquad F(x(t)) = X(\omega) = \int_{-\infty}^{+\infty} x(t) e^{-j\omega t} dt \tag{7.2}$$

obu stron równania (7.1) oraz, po drugie, korzystając z następującej własności transformacji Laplace'a:

$$L\left(\frac{dx^m(t)}{dt^m}\right) = s^m X(s), \tag{7.3}$$

otrzymuje się transmitancję filtra analogowego $H(s)$:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_0 + b_1 s^1 + b_2 s^2 + \ldots + b_M s^M}{1 + a_1 s^1 + a_2 s^2 + \ldots + a_N s^N} \tag{7.4}$$

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_M \cdot (s - z_1)(s - z_2) \cdot \ldots \cdot (s - z_M)}{a_N \cdot (s - p_1)(s - p_2) \cdot \ldots \cdot (s - p_N)} \tag{7.5}$$

gdzie rzeczywiste wartości współczynników b_k oraz a_k obu wielomianów (albo zespolne wartości ich miejsc zerowych/pierwiastków z_k oraz p_k) są specjalnie dobierane. Po podstawieniu $s = j\omega = j2\pi f$, $j = \sqrt{-1}$, transmitancja $H(s)$ przekształca się w charakterystykę częstotliwościową filtra $H(f)$, gdzie: $|H(f)|$ (moduł liczby zespolonej) - to odpowiedź amplitudowa filtra (jego wzmocnienie/tłumienie dla składowej o częstotliwości f), a $\angle H(f)$ (kąt liczby zespolonej) - odpowiedź fazowa filtra (przesunięcie kątowe, przeliczane opóźnienie czasowe, składowej o częstotliwości f - patrz poniżej).

Aby współczynniki wielomianów transmitancji przyjmowały wartości rzeczywiste, miejsca zerowe wielomianów muszą występować w parach sprzężonych: z_k, z_k^* i p_k, p_k^* — np. $z_1 = 1 + 2j$ i $z_2 = 1 - 2j$. Warunkiem stabilności filtra jest położenie biegunów transmitancji p_k w lewej półpłaszczyźnie zmiennej zespolonej s, czyli posiadanie przez nie ujemnej części rzeczywistej, np. $p_1 = -5 + j10$. Umieszczenie zera transmitancji z_1 na osi urojonej $j\omega = j2\pi f$ w punkcie $z_1 = j2\pi f_1$ powoduje, że filtr usuwa z sygnału składową o częstotliwości f_1 (ponieważ wówczas $|H(f_1)| = 0$). Umieszczenie bieguna transmitancji p_2 w lewej półpłaszczyźnie, blisko osi urojonej $j\omega = j2\pi f$, np. w punkcie $p_2 = -10 + j2\pi f_2$ powoduje, że filtr wzmacnia w sygnale częstotliwość f_2, tym bardziej im jest bliżej osi urojonej (przykładowo $(-10) \rightarrow (-1)$) powoduje zwiększenie wartości $H|(f_2)|$). Zero transmitancji leżące na osi urojonej powoduje skok charakterystyki fazowo-częstotliwościowej o $-pi$ radianów.

Filtry dzielą się na: dolno-, górno-, pasmowo-przepustowe oraz pasmowo-zaporowe (patrz rysunek 7.1. Oczywiście mówimy o zakresach częstotliwościowych. W paśmie przepustomym powinny mieć wzmocnienie 1, a w zaporowym - 0. Powinny też szybko przechodzić od fazy "przepuszczania" ($|H(f)| = 1$) do fazy "usuwania" ($|H(f)|$=0). Najlepiej jest kiedy charakterystyka fazowa filtra liniowo opada w funkcji częstotliwości w pasmie przepustowym ($\angle H(f) = -\alpha f$): wówczas wszystkie częstotliwości na wyjściu filtra są opóźnione o ten sam czas α i kształt sygnału "przechodzącego" nie zmienia się. Ponieważ kąt liczby zespolonej oblicza się tylko dla zakresu $[-\pi, \pi)$, a kąt $\angle H(f)$ ciągle maleje, obserwuje się skokowe zmiany ch-ki fazowej o 2π, które należy usunąć. W Matlabie jest za to odpowiedzialna funkcja `unwrap()`. Problem ten i jego rozwiązanie są przedstawione na rysunku 7.2.

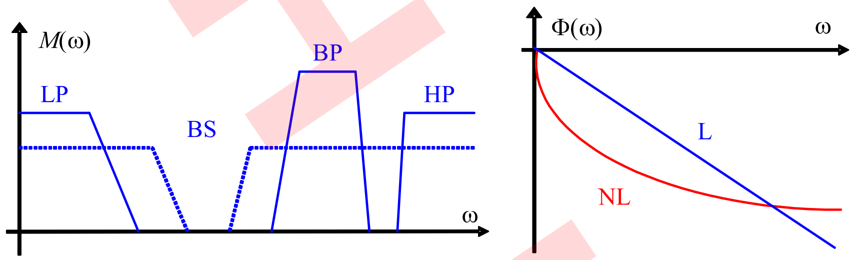

Fig. 7.1: (po lewej) — typy filtrów pod względem przepuszczanych częstotliwości i kształtu odpowiedzi amplitudowej $M(f) = |H(f)|$: Low-Pass (LP), High-Pass (HP), Band-Pass (BP) and Band-Stop (BS), (po prawej) — liniowa (L), wskazana, oraz nie-liniowa (NL), niechciana, odpowiedź fazowa filtra $\Phi(f) = \angle H(f)$. Oznaczenie: $\omega = 2\pi f$. [40]

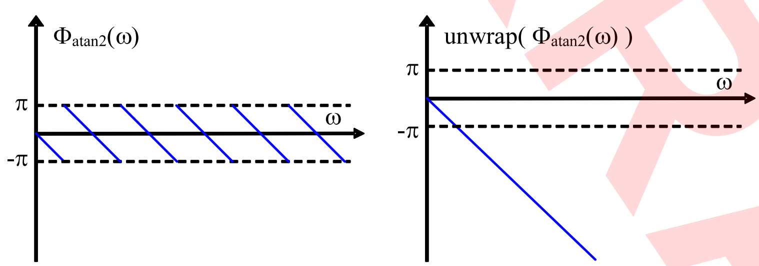

Fig. 7.2: Ilustracja problemu obliczania ch-ki fazowej filtra $\Phi(f) = \angle H(f)$. Faza (kąt liczby zespolonej) może być tylko obliczona dla zakresu $[-\pi, \pi)$, w Matlabie z użyciem funkcji `angle()`, `atan2()` — potem konieczne jest zastosowanie funkcji (`unwrap()`) likwidującej otrzymane skoki fazy o $+2\pi$ radianów. Oznaczenie: $\omega = 2\pi f$. [40]

Listing 7.1: Transmitancja i odpowiedź/charakterystyka częstotliwościowa filtra analogowego w Matlabie

```
% cps_07_analog_intro.m
clear all; close all;

% Zaprojektuj/dobierz wspolczynniki transmitancji filtra analogowego
if(0)  % dobor wartosci wspolczynnikow wielomianow zmiennej 's' transmitancji
   b = [3,2];                        % [ b1, b0 ]
   a = [4,3,2,1];                    % [ a3, a2, a1, a0=1]
   z = roots(b); p = roots(a);       % [b,a] --> [z,p]
else   % dobor wartosci pierwiastkow wielomianow zmiennej 's' transmitancji
   wzm = 0.001;
   z = j*2*pi*[ 600,800 ];           z = [z conj(z)];
   p = [-1,-1] + j*2*pi*[100,200]; p = [p conj(p)];
   b = wzm*poly(z);  a = poly(p);  % [z,p] --> [b,a]
end
figure; plot(real(z),imag(z),'bo', real(p),imag(p),'r*'); grid;
```

```
title('Zera (o) i Bieguny (*)'); xlabel('Real()'); ylabel('Imag()'); pause

% Weryfikacja odpowiedzi (charakterystyki) filtra: amplitudowej fazowej, impulsowej, skokowej
f = 0 : 0.1 : 1000;             % czestotliwosc w hercach
w = 2*pi*f;                     % czestotliwosc katowa, pulsacja
s = j*w;                        % zmienna transformacji Laplace'a
H = polyval(b,s) ./ polyval(a,s);  % h(s) --> H(f) dla s=j*w: iloraz dwoch wielomianow
figure; plot(f,20*log10(abs(H))); xlabel('f [Hz]'); title('|H(f)| [dB]'); grid; pause
figure; plot(f,unwrap(angle(H))); xlabel('f [Hz]'); title('angle(H(f)) [rad]'); grid; pause
figure; impulse(b,a); pause      % odpowiedz filtra na pobudzenie impulsowe (z Control Toolbox)
figure; step(b,a); pause         % odpowiedz filtra na pobudzenie skokowe (z Control Toolbox)
```

Problem 7.1 (* Zaznajomienie się z podstawami fitracji analogowej). Przeanalizuj program 7.1, w którym wybierane są wartości współczynników wielomianów transmitancji $H(s)$ filtra analogowego oraz rysowane są charakterystyki otrzymanego filtra: położenie zer i biegunów transmitancji, odpowiedź amplitudowa $|H(f)|$, odpowiedź fazowa $\angle H(f)$, odpowiedź impulsowa $h(t)$ (na impuls jednostkowy - deltę Diraca - we wejściu) oraz skokowa $u(t)$ (na skok jednostkowy na wejściu). W celu zagwarantowania stabilności filtra (nie wzbudzania się), bieguny jego transmitancji powinny leżej w lewej półpłaszczyźnie zmiennej zespolonej s transformacji Laplace'a, natomiast zera transmitancji - gdziekolwiek. Uruchom program, zapoznaj się z rysunkami otrzymanymi dla aktualnych i zmienionych przez Ciebie wartości parametrów `b,a,z,p`. Wytłumacz pochodzenie wyraźnych maksimów ("górki", "garby wielbłąda") oraz minimów ("dołki","szczeliny"), widocznych w charakterystyce amplitudowo-częstotliwościowej filtra oraz pochodzenie skoków $\pm\pi$ radianów obserwowanych w charakterystyce fazowo-amplitudowej filtra. Zaproponuj inne niż obecnie położenie zer i biegunów transmitancji, zweryfikuj jak zmieniła się czarakterystyka amplitudowo-częstotlliwościowa oraz fazowo-częstotliwościowa filtra po tej zmianie. Sprawdź stabilność filtra: jego odpowiedź impulsowa powinna gasnąć do zera. Zapamietaj: zero transmitancji służy do usuwania wybranych częstotliwości (powoduje "dołek"/"szczelinę" w ch-ce amplitudowej filtra), natomiast biegun transmitancji - służy do wzmacniania wybranych częstotliwości (powoduje "górkę"/"garb"). Zera i bieguny transmitancji występują w parach sprzężonych: $\{z_k, z_k^*\}$, $\{p_k, p_k^*\}$ (dzięki temu współczynniki wielomianów transmitancji mają wartości rzeczywiste, nie zespolone, sprawdź to!). Zaprojektuj filtr analogowy, usuwający częstotliwość 300 Hz oraz wzmacniający częstotliwość 400 Hz.

Problem 7.2 (* Metoda projektowania "Zer & Biegunów" - dobór położenia pierwiastków wielomianów transmitancji). Zmodyfikuj program 7.1 oraz spróbuj zaprojektować transmitancję dobrego filtra analogowego, przepuszczającego ze wzmocnieniem `wzm=1` TYLKO składowe sygnału o częstotliwościach z zakresu (wybierz jedną z czterech opcji):

1. low-pass: [od 0 do 250] Hz;
2. high-pass: [od 250 do nieskończoności] Hz;
3. band-pass: [od 400 do 600] Hz;
4. band-stop: [od 0 do 400] Hz oraz [od 600 do nieskończoności] Hz.

oraz usuwającego wszystkie pozostałe częstotliwości na wyjściu filtra. Spróbuj dobrać odpowiednie wartości: 1) współczynników b_k, a_k wielomianów transmitancji, oraz 2) miejsc zerowych/pierwiastków z_k, p_k tych wielomianów. Która metoda okazała się prostsza i skuteczniejsza? Dlaczego?

Problem 7.3 (* Projekt filtra dolno-przepustowego (low-pass) dla przetwornika A/C). Użyj programu 7.1 oraz zaprojektuj dobry, analogowy filtr dolno-przepustowy dla przetwornika A/C, pracującego z częstotliwością próbkowania $f_{pr} = 48000$ Hz. Filtr ten powinien powinien mieć: 1) maksymalnie płaską charakterystykę amplitudowo-częstotliwościową równą 1 w paśmie przepstowym dla częstotliwości $f < \frac{f_{pr}}{2}$, oraz 2) bardzo duże tłumienie w paśmie zaporowym dla częstotliwości $f \geq \frac{f_{pr}}{2}$. Po 20 minutach prób, proszę, przerwij pracę.

TEMAT #2: Projektowanie filtrów analogowych: Butterwortha, Czebyszewa i Cauera (eliptycznych). Metoda wyboru położenia "Zer & Biegunów Transmitancji" jest metodą "prób i błędów", czyli czasochłonną. Istnieją wzory matematyczne, umożliwiające prostsze, szybsze i skuteczniejsze projektowanie. Poniżej użyjemy metod projektowych, zaproponowanych przez Butterwortha, Czebyszewa (filty typu I oraz II) oraz Cauera (filtry eliptyczne). O czym trzeba pamiętać podczas wyboru konkretnego typu filtra?

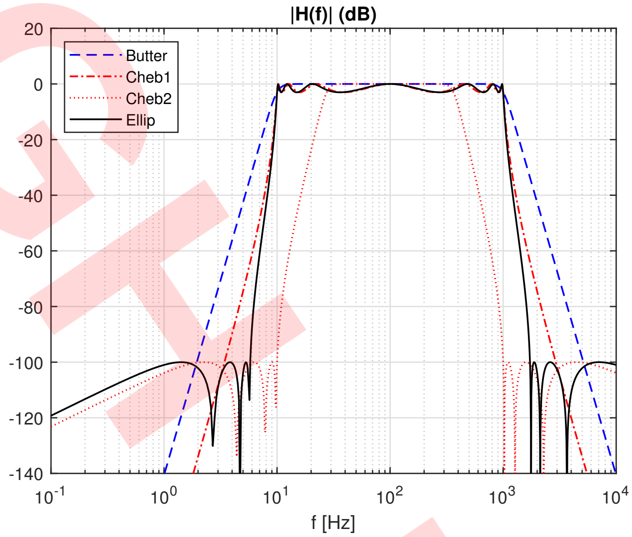

Fig. 7.3: Porównanie różnych rodzajów filtrów analogowych na przykładzie projektu filtra pasmowo-przepustowego w paśmie $[10, 1000]$ Hz. Liczba biegunów $N = 7$. Filtr eliptyczny zapewnia najszybsze przejście od pasma przepustowego do zaporowego.

1. Charakterystyka amplitudowa filtra $|H(f)|$, tak w paśmie przepustowym jak i zaporowym niestety może, ale nie musi, być oscylacyjna. Musimy zdecydować czy i gdzie pozwalamy na oscylacje, dodatkowo na jak silne - patrz tabela 7.1.
2. Im więcej oscylacji dopuszczamy w $|H(f)|$, tym filtr staje się bardziej stromy. Filtr Butterwortha nie ma w ogóle oscylacji (jest najmniej stromy), zaś filtr Cauera - ma najwięcej oscylacji, zarówno w paśmie przepustowym jak i w paśmie zaporowym (jest najbardziej stromy). Filtry Czebyszewa leżą "pomiędzy", różnią się miejscem występowania oscylacji: typ I - w paśmie przepustowym, typ II - w paśmie zaporowym. Patrz rysunek 7.3.
3. Niezależnie od typu: im więcej biegunów transmitancji ma filtr (większy wielomian), tym jest bardziej stromy. W filtrze Butterwortha jeden biegun zapewnia opadanie $|H(f)|$ 20 decybeli na dekadę, czyli 10-krotny wzrost częstotliwości.
4. Dolnoprzepustowy filtr Butterwortha ma bieguny transmitancji leżące na okręgu, a Czebyszewa-I - na elipsie. Oba nie mają zer transmitancji (stała w liczniku).
5. Dolnoprzepustowe filtry Czebyszewa-II i Cauera (elliptyczny) mają zera i bieguny transmitancji (wielomian w liczniku i mianowniku).
6. Należy zwrócić uwagę, że podczas projektowania dla filtra Czebyszewa-II podajemy częstotliwość początku pasma zaporowego, a dla pozostałych filtrów - koniec pasma przepustowego (patrz rysynek 7.3).

Table 7.1: Porównanie filtrów analogowych różnych typów

Nazwa	Typ	Oscylacje w Pass-Band	Oscylacje w Stop-Band	Stromość	Nieliniowość fazy
Butterworth	LP, HP, BP, BS	Nie	Nie	Mała	Bardzo mała
Czebyszew I	LP, HP, BP, BS	Tak	Nie	Duża	Mała
Czebyszew II	LP, HP, BP, BS	Nie	Tak	Duża	Mała
Cauer (elliptyczny)	LP, HP, BP, BS	Tak	Tak	B. duża	Dardzo duża
Bessel	LP	Nie	Nie	B. mała	Brak

Listing 7.2: Dolno-przepustowy filtr Butterwortha w Matlabie

```
% cps_07_analog_butter.m
clear all; close all;

N =5;                                    % liczba biegunow transmitancji
f0 = 100;                                % czestotliwosc 3dB (odciecia) filtra dolnoprzepustowego
alpha = pi/N;                            % kat "kawalka tortu" (okregu)
beta  = pi/2 + alpha/2 + alpha*(0:N-1);  % katy kolejnych biegunow transmitancji
R = 2*pi*f0;                             % promien okregu
p = R*exp(j*beta);                       % bieguny transmitancji lezace na okregu
z = []; wzm = prod(-p);                  % LOW-PASS:  brak zer tramsitancji, wzmocnienie
%z = zeros(1,N); wzm = 1;                % HIGH-PASS: N zer transmitancji, wzmocnienie
b = wzm*poly(z); a=poly(p);              % [z,p] --> [b,a]
b = real(b);      a=real(a);             %

% ... kontynuacja csp_07_analog_intro.m
```

Problem 7.4 (* Badanie dolno- i górno-przepustowych, analogowych filtrów Butterwortha). Możesz zacząć od przekartkowania rozdziału 6.3 w książce [40]. Przeanalizuj program 7.2, w którym jest projektowany analogowy filtr Butterwortha. Uruchom go wybierając opcję położenia zer transmitancji "`z`" dla filtra: 1) dolno-przepustowego, oraz 2) górno-przepustowego. Obejrzyj rysunki wszystkich charakterystyk zaprojektowanego filtra. Zwróć uwagę, że bieguny transmitancji leżą na okręgu o promieniu `R` ($R = \omega_0$ - graniczna częstotliwość radialna filtra). Dodaj okrąg do rysunku, pokazującego położenie zer i biegunów transmitancji: `x=0:pi/1000:2*pi; c=cos(); s=sin(); plot(c,s,'k-',...);` Zauważ występowanie N-krotnego zera transmitancji (pierwiastka wielomianu mianowika) dla górno-przepustowego filtra Butterworth: $(s-0)^N$ `(z = zeros(1,N)`. Dodaj nowy rysunek `semilogx()` charakterystyki amplitudowo-częstotliwościowej filtra: `semilogx(f,20*log10(abs(H)))`. Zmień liczbę biegunów transmitancji: `N=2,3,4,...,10`: każdy biegun zwiększa szybkość opadania ch-ki amplitudowo-częstotliwościowej o 20 decybeli dla jednej dekady częstotliwości (dekada to 10-krotny przyrost częstotliwości). Ustaw 3-decybelową częstotliwość graniczną (odcięcia) filtra na `f0=1,10,100`: zaobserwuj zmianę promienia okręgu oraz zmianę charakterystyki amplitudowej filtra na rysunku `semilogx()`. Porównaj swój projekt z wynikiem użycia funkcji Matlaba:
`[b,a]=butter(N,2*pi*f0,'low','s')` % (low albo high).

Problem 7.5 (Projektowanie analogowego, dolno-przepustowego filtra Czebyszewa typu-I).** Analogowy, dolno-przepustowy filtr Czebyszewa typu-I ma transmitancję bardzo podobną do dolno-przepustowego filtru Butterwortha: w jego przypadku bieguny transmitancji (pierwiastki wielomianu mianownika) leżą nie na okręgu tylko na elipsie. Znajdź odpowiednie równania matematyczne w książce [40] (rozdz. 6.4), a nawet program, w którym projektowany jest filtr tego typu. Oblicz zera i bieguny `[z,p]` transmitancji filtra, pokaż ich położenie na płaszczyźnie zespolonej zmiennej 's' transformacji Laplace'a oraz sprawdź czy leżą one na elipsie. Przekonwertuj `[z,p]` na `[b,a]` z użyciem funkcji `roots()` oraz oblicz i pokaż charakterystykę amplitudowo-częstotliwościową (inaczej: odpowiedź częstotliwościową) tego filtra. Powinnaś (powinieneś) zobaczyć oscylacje w paśmie przepustowym oraz szybsze przejście z pasma przepustowego do pasma zaporowego niż dla filtra Butterwortha o tym samym rzędzie `N` (tej samej liczbie biegunów). Tak jak na rysunku 7.3. Sprawdź to: narysuj charakterystki amplitudowe obu filtrów na jednym rysunku typu `semilogx()`. Powtórz ostatni eksperyment dla różnej liczby biegunów `N=2,3,4,5,6,7,8` oraz dla różnych wartości częstotliwości granicznej `f0=1,10,100`.

Problem 7.6 (Funkcje Matlaba do projektowania filtrów analogowych Butterwortha, Czebyszewa oraz Cauera (elliptycznych)).** Obecnie jesteśmy już ± zaznajomieni z zagadnieniem projektowania filtrów analogowych. Nadszedł czas, aby *zanurkować głebiej*, ale z pomocą funkcji Matlaba. Skopiuj kod zawarty w listingu 7.3 na początek programu 7.1, odpowiednio dopasuj do siebie oba programy oraz przetestuj krok-po-kroku wszystkie typy filtrów, których zaprojektowanie w Matlabie jest możliwe. Obecnie program 7.3 umożliwia tylko zaprojektowanie:

- filtrów LP, HP, BP oraz BS Butterwortha,
- filtrów BP Butterwortha, Czebyszewa typu-I/II oraz elliptycznego (Cauera).

Dodaj brakujące filtry: powinny być wszystkie filtry LP/HP/BP/BS dla B, C-I, C-II oraz E. Potem, przejdź do etapu porównania filtrów. Na początku, przetestuj dolno-przepustowe filtry Butterwortha (`butter()`), Czebyszewa-I (`cheby1()`), Czebyszewa-II (`cheby2()`) oraz elliptyczny (`ellip()`). Na jednym rysunku narysuj ch-ki amplitudowo-częstotliwościowe wszystkich filtrów z użyciem `semilogx()`. Następnie porównaj filtry górno-przepustowe wszystkich typów - ponownie na jednym rysunku. Zakończ porównaniem filtrów pasmowo-przepustowych (rys. 7.3) oraz pasmowo-zaporowych. Zwróć uwagę, że filtr Butterwortha nie ma oscylacji w paśmie przepustowym i zaporowym, Czebyszew-I ma tylko oscylacje w paśmie przepustowym, Czebyszew-II tylko w paśmie zaporowym, żaś filtr eliptyczny — w obu tych pasmach. Zauważ także, że im więcej oscylacji występuje w pasmach przepustowym i zaporowym, tym ch-ka ampltudowa filtra jest bardziej stroma i filtr szybciej przechodzi od fazy *pass* do fazy *stop*. Poza różnymi charakterystykami amplitudowymi (wzmocnieniem), filtry różnią się także charakterystykami fazowymi (opóźnieniem). Zwróć uwagę na nieidealnie liniową (czyli nieliniową!) charakterystykę fazową wszystkich filtrów w paśmie przepustowym (który filtr ma najlepszą?) — będzie to powodować różne opóźnienie poszczególych składowych częstotliwościowych na wyjściu filtrów i, w konsekwencji, zmianę kształtu sygnału przechodzącego. Wreszcze, porównaj jakie jest położenie zer i biegunów wszystkich filtrów dla typu pasmowo-przepustowego oraz pasmowo-zaporowego.

Listing 7.3: Funkcje do projektowania filtrów analogowych w języku Matlab

```
% cps_07_analog_matlab.m
clear all; close all;

N = 8;                   % liczba biegunow transmitancji
f0=10;  f1=10;  f2=100;  % czestotliwosci w Hz [1/s]
Rp = 3; Rs = 100;        % dozwolony poziom oscylacji (dB) w pasmie: pass (p), stop (s)
% [b,a] = butter(N, 2*pi*f0, 'low','s');                % Butt  LP
% [b,a] = butter(N, 2*pi*f0, 'high','s');               % Butt  HP
% [b,a] = butter(N, 2*pi*[f1,f2], 'stop', 's');         % Butt  BS
  [b,a] = butter(N, 2*pi*[f1,f2], 'bandpass', 's');     % Butt  BP
% [b,a] = cheby1(N, Rp, 2*pi*[f1,f2], 'bandpass', 's'); % Cheb1 BP
% [b,a] = cheby2(N, Rs, 2*pi*[f1,f2], 'bandpass', 's'); % Cheb2 BP
% [b,a] =  ellip(N, Rp, Rs,2*pi*[f1,f2], 'bandpass', 's');  % Ellip BP

% ... kontynuuj cps_07_analog_intro.m
```

TEMAT #3: Trudniejszy: transformacja analogowego, dolnoprzepustowego, unormowanego ($\omega_0 = 1$) filtra prototypowego na filtr typu: dolno-przepustowy, górno-przepustowy, pasmowo-przepustowy i pasmowo-zaporowy. Należy wyraźnie podkreślić, że filtry analogowe są projektowane w dwóch krokach: 1) na początku jest projektowany tzw. analogowy znormalizowany ($\omega_0 = 2\pi f_0 = 1$), dolno-przepustowy filtr prototypowy, 2) prototyp ten jest przekształacany za pomocą transformacji częstotliwości na filtr wymaganego typu (dolno-, górno-, pasmowo-przepustowy lub pasmowo-zaporowy) o zadanych częstotliwościach granicznych pasm przepustowych i zaporowych. Transformacja jest dokonywanana drogą zastąpienia ziennej s w $H(s)$ przez wybraną funkcję zmiennej s' oraz wartości jej parametrów: w ten sposób wielomiany transmitancji $H(s)$ zmiennej s są przekształcane na wielomiany transmitancji $H(s')$ zmiennej s', mające inne miejsca zerowe. W tabeli 7.2 podano wzory funkcji transformujacych: $s = \text{funkcja}(s')$. Samą transformację przeprowadza się w ten sposób, że zastępuje się każde wyrażenie $(s - z_k)$ i $(s - p_k)$ w transmitancji $H_{LP}(s)$ prototypu (patrz równanie (7.5)) wyrażeniem podanym w tabeli 7.2, otrzymanym po podstawieniu $s = g_{xx}(s')$ do $(s - z_k)$ oraz $(s - p_k)$. Na rysunkach 7.4 oraz 7.5 wytłumaczono transformację unormowanego, analogowego, dolno-przepustowego filtra prototypowego na filtr o innym paśmie przepustowym.

Table 7.2: Wzory na transformację analogowego filtra prototypowego typu Low-Pass (LP): $H_{LP}^{\omega 0=1}(s) \rightarrow H_{LP,HP,BP,BS}(s')$

Nr	Typ	Funkcja $s = g_{xx}(s')$	Wynik transformacji dla $(s - z_m)$
1	$LP \rightarrow LP$	$s = s'/\omega_0{}^{\ddagger}$	$\frac{s' - z_m \omega_0}{\omega_0}$
2	$LP \rightarrow HP$	$s = \omega_0/s'^{\ddagger}$	$(-z_m)\frac{(s') - \frac{\omega_0}{z_m}}{(s')}$
3	$LP \rightarrow BP$	$s = \frac{s'^2 + \omega_0^2}{\Delta\omega \cdot s'}{}^{\S}$	$\frac{(s')^2 - z_m \Delta\omega \cdot (s') + \omega_0^2}{\Delta\omega \cdot (s')}$
4	$LP \rightarrow BS$	$s = \frac{\Delta\omega \cdot s'}{s'^2 + \omega_0^2}{}^{\S}$	$(-z_m)\frac{(s')^2 - \frac{\Delta\omega}{z_m}(s') + \omega_0^2}{(s')^2 + \omega_0^2}$

‡ — $\omega_0 = \omega_{pass}$
§ — $\omega_0 = \sqrt{\omega_{pass1}\omega_{pass2}}$, $\Delta\omega = \omega_{pass2} - \omega_{pass1}$

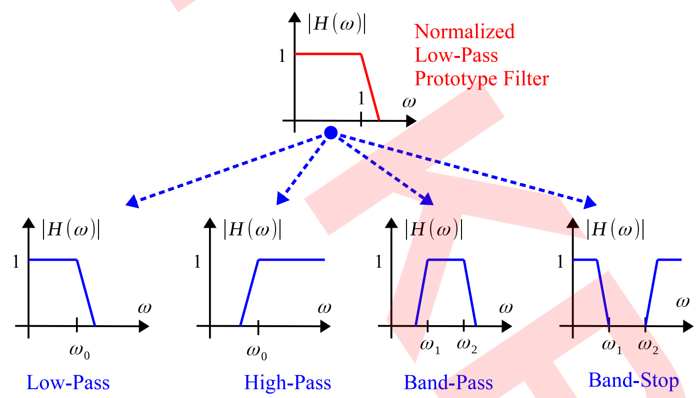

Fig. 7.4: Wytłumaczenie istoty transformacji częstotliwości unormowanego ($\omega_{3dB} = 1$) dolno-przepustowego filtra analogowego na filtr o innym paśmie przepustowym.

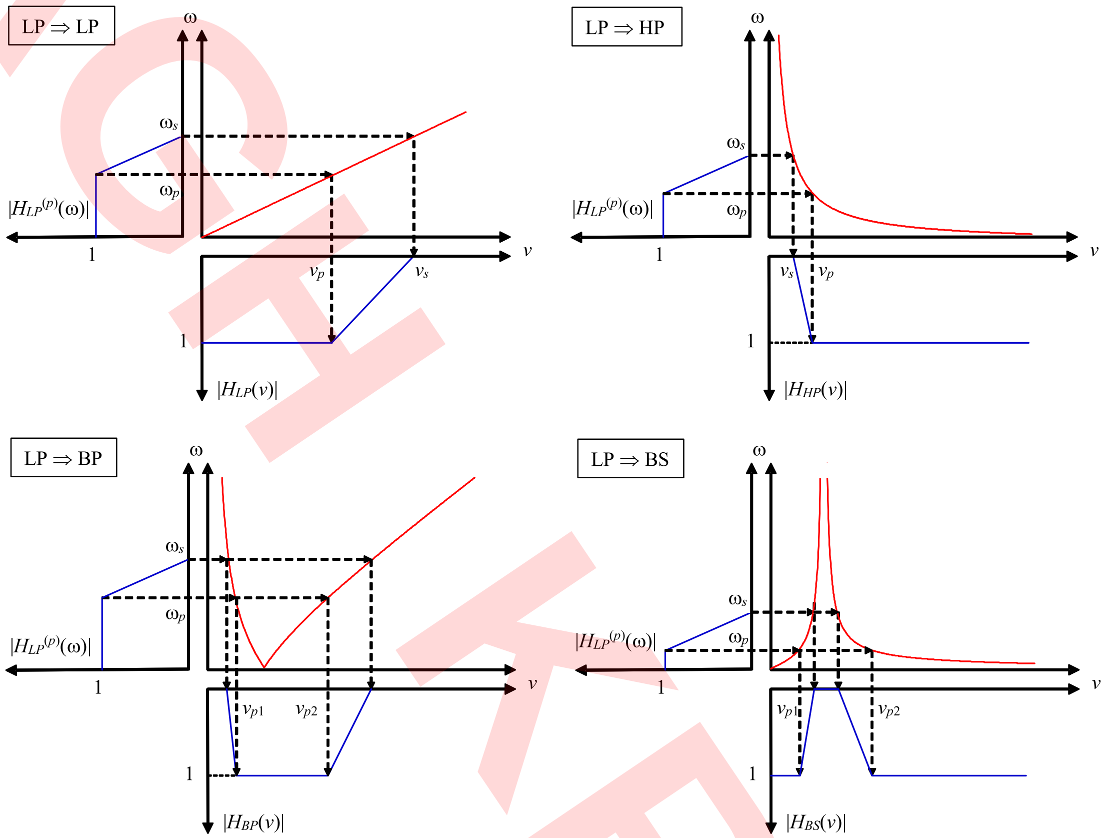

Fig. 7.5: Graficzna ilustracja transformacji dolno-przepustowego prototypu filtra analogowego:, w kolejnych wierszach: $LP \to LP$, $LP \to HP$, $LP \to BP$, $LP \to BS$.

Listing 7.4: Projektowanie znormalizowanego dolno-przepustowego analogowego fitra prototypowego oraz jego transformacja częstotliwościowa na filtr analogowy innego typu

```
% cps_07_analog_tranform.m
clear all; close all;

% Wymagania
N = 8;              % liczba biegunow transmitanci prototypu analogowego
f0 = 100;           % dla filtrow LowPass  oraz HighPass
f1 = 10; f2=100;    % dla filtrow BandPass oraz BandStop
Rp = 3;             % dozwolony poziom oscylacji w pasmie przepustowym (w dB) - ripples in pass
Rs = 100;           % dozwolony poziom oscylacji w pasmie zaporowym (w dB) - ripples in stop

% Projekt analogowego filtra prototypowego - dolnoprzepustowy z w0 = 1
  [z,p,wzm] = buttap(N);          % analogowy prototyp Butterwotha
% [z,p,wzm] = cheblap(N,Rp);      % analogowy prototyp Czebyszewa typu-I
% [z,p,wzm] = cheb2ap(N,Rs);      % analogowy prototyp Czebyszewa typu-II
% [z,p,wzm] = ellipap(N,Rp,Rs);   % analogowy prototyp Cauera (eliptyczny)
b = wzm*poly(z);                  % [z,wzm] --> b
a = poly(p);                      % p --> a
f = 0 : 0.01 : 1000; w=2*pi*f;    % zakres czestotliwosci
H = freqs(b,a,w);                 % odpowiedz czestotliwosciowa filtra (funkcja Matlaba)
fi=0:pi/1000:2*pi; c=cos(fi); s=sin(fi);
figure; semilogx(w,20*log10(abs(H))); grid; xlabel('w [rad*Hz]'); title('Analog Proto |H(f)'); pause
figure; plot(real(z),imag(z),'ro',real(p),imag(p),'b*',c,s,'k-'); grid; title('Analog Proto ZP'); pause

% Transformacja czestotliwosci: znormalizowany (w0=1) LowPass --> xxxPass, xxxStop
% Funkcje xx2yy z Signal Processing Toolbox
```

```
  [b,a] = lp2lp(b,a,2*pi*f0);                        % LowPass na LowPass
% [b,a] = lp2hp(b,a,2*pi*f0);                        % LowPass na HighPass
% [b,a] = lp2bp(b,a,2*pi*sqrt(f1*f2),2*pi*(f2-f1));  % LowPass na BandPass
% [b,a] = lp2bs(b,a,2*pi*sqrt(f1*f2),2*pi*(f2-f1));  % LowPass na BandStop

z=roots(b); p=roots(a);

% ... kontynuuj cps_07_analog_intro.m
```

Problem 7.7 (* Zapoznanie się z prototypami dolno-przepustowych (LP) filtrów analogowych oraz sposobem ich transformacji na filtry innego typu (HP, BP, BS)). Skopiuj kod 7.4 na początek programu 7.1. Użyj po kolei każdy filtr prototypowy (Butterwortha, Czebyszewa-I oraz II, elliptyczny) oraz wybierz inny rodzaj filtru docelowego (LP, HP, BP, BS). Zapoznaj się z kształtem ch-ki ampltudowo-częstotliwościowej filtra, pokazanej na rysunku `semilogx()`, przed i po transformacji częstotliwości. Zwróć uwagę, że wszystkie filtry prototypowe są dolno-przepustowe oraz unormowane (`w0=1`). Mają one pasma przejściowe *pass-2-stop* oraz oscylacje w pasmach przepustowych i zaporowych TYPOWE dla typu użytego filtra prototypowego (B, C1, C2, E). Kiedy wartość `N` rośnie, ch-ka amplitudowa filtra staje się bardziej stroma.

Problem 7.8 (* Przykład transformacji analogowego filtra LP na filtr BS). W tabeli 7.2 podano konkretne równania transformacji częstotliwościowych. Więcej szczegółów możesz znaleźć w [40] (rozdz. 6.2). W tym zadaniu zainteresujemy się równaniem, dotyczącym transformacji częstotliwości LP-2-BS filtra analogowego: $H_{LP}(s) \rightarrow H_{BS}(s')$. W tym przypadku każde wyrażenie $(s - z_m)$ oraz $(s - p_m)$, występujące w transmitancji $H(s)$ filtra prototypowego typu LP, jest zastępowane przez wyrażenie $(s' - z_m)$ oraz $(s' - p_m)$):

$$(s - z_m) \quad \rightarrow \quad \left(\frac{\Delta\omega \cdot (s')}{(s')^2 + \omega_0^2} - z_m \right) \quad \rightarrow \quad (-z_m) \frac{(s')^2 - \frac{\Delta\omega}{z_m}(s') + \omega_0^2}{(s')^2 + \omega_0^2} \tag{7.6}$$

gdzie ω_0 oznacza średnią geometryczną pasma zaporowego filtra: $\omega_0 = 2\pi\sqrt{f_1 \cdot f_2}$ (f_1 - początek pasma zaporowego, f_2 - koniec pasma zaporowego), notomiast $\Delta\omega$ jest szerokością pasma zaporowego ($\Delta\omega = 2\pi(f_2 - f_1)$). Funkcja 7.5 implementuje tę transformację w języku Matlab. Przeanalizuj kod tej funkcji oraz ją zastosuj - sprawdź poprawność jej działania na wybranym filtrze prototypowym. Porównaj wynik działania funkcji oraz funkcji Matlaba `lp2bs()`. Napisz i przetestuj pozostałe funkcje do transformacji częstotliwości analogowego filtra prototypowego LP: `lp2lpMY()`, `lp2hpMY()` and `lp2bpMY()`. Zastosuj je i porównaj wynik z funkcjami Matlaba (te same nazwy tylko bez MY).

Listing 7.5: Transformacja LP2BS prototypu LP filtra analogowego

```
function [zz,pp,wzm] = lp2bsMY(z,p,wzm,w0,dw)
% transformacja LowPass-to-BandStop prototypowego filta analogowego

zz = []; pp = [];
for  k=1:length(z) % transformowanie zer transmitancji
     zz = [ zz roots([ 1 -dw/z(k) w0^2 ])' ];
     wzm = wzm*(-z(k));
end
for  k=1:length(p) % transformowanie biegunoww transmitancji
     pp = [ pp roots([ 1 -dw/p(k) w0^2 ])' ];
     wzm = wzm/(-p(k));
end
for k=1:(length(p)-length(z))
     zz = [ zz roots([ 1 0 w0^2 ])' ];
end
```

TEMAT #4: Sprzętowa (układowa) implementacja filtrów analogowych. Ostatnim tematem jest implementacja sprzętowa układu filtra analogowego i jej testowanie w programie symulacyjnym LTSpice. Po zaprojektowaniu transmitancji filtra analogowego o akceptowalnej charakterystyce częstotliwościowej, amplitudowo-częstotliwościowej i fazowo-częstotliwościowej, współczynniki transmitacji muszą zostać przeliczone na wartości rezystancji R i pojemności C elementów układu sprzętowego, realizującego w praktyce daną transmitancję. Załóżmy, że otrzymaliśmy transitancję postaci:

$$H(s) = \frac{3\,042\,184\,930}{(s^2 + 34088.3s + 3\,042\,184\,930)} \cdot \frac{3\,042\,184\,930}{(s^2 + 89244.3s + 3\,042\,184\,930)} \cdot \frac{55156}{(s + 55156)}, \tag{7.7}$$

Jej realizacja układowa jest przestawiona na rysunku 7.6: jest to kaskada dwóch sekcji bikwadratowych oraz sekcji pierwszego rzędu, wynikająca z przeprowadzonej faktoryzacji wielomianów transmitancji. Na końcu jest dodatkowy dzielnik napięcia, korygujący wzmocnienie. Wartości elementów pasywnych R i C, podane w opisie rysunku, zostały dobrane tak (z dostępnych w typo-szeregu), aby w przybliżeniu otrzymać takie same wartości współczynników transmitancji jak w równaniu (7.7). Więcej szczegółów można znaleźć w [40] (rozdz. 6.6).

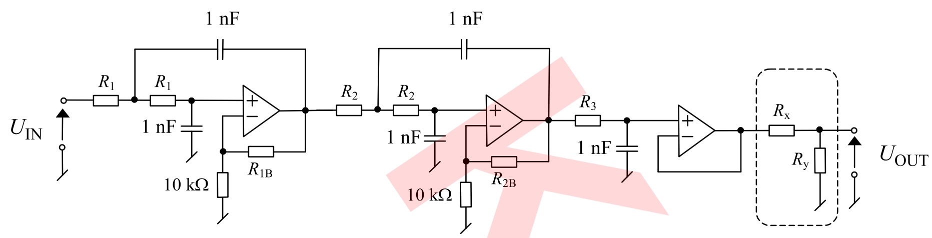

Fig. 7.6: Przykładowa realizacja układowa dolnoprzepustowego filtra Butterwortha $N = 5$-tego rzędu, mającego transmitancję zdefiniowaną równaniem (7.7). Wartości elementów układu są następujące: $R_1 = R_2 = R_3 = 18.13$ kΩ, $R_{1B} = 13.82$ kΩ, $R_{2B} = 3.8197$ kΩ, $K_1 = 2.3820$, $K_2 = 1.3820$, $K_3 = 1$, $K = K_1K_2K_3 = 3.2918$. Dla wymaganego wzmocnienia sumarycznego $G = 1$ oraz wymaganej rezystancji wyjściowej $R_{out} = 10$ kΩ, otrzymujemy: $R_x = \frac{K}{G}R_{out} = 32.92$ kΩ oraz $R_y = \frac{K}{K-G}R_{out} = 14.36$ kΩ [40]

Problem 7.9 (* Weryfikacja projektu filtra analogowego w symulatorze układów analogowych).** Na końcu rozdziału 6 w [40], dotyczącego filtrów analogowych, jest szczegółowo, krok po kroku przedstawiony przykład projektu filtra analogowego i jego implementacji układowej/sprzętowej. Skrótowo zaprezentowano go powyżej. Zaprojektowana transmitancja filtra, iloraz dwóch wielomianów, jest przedstawiona jako kaskadowe połączenie tzw. sekcji bikwadratowych (czyli ilorazów wielomianów rzędu drugiego, zrealizowanych na pojedynczym wzmacniaczu operacyjnym) oraz sekcji pierwszego rzędu. Współczynniki transmitancji są przeliczane na wartości elementów RC sekcji bikwadratowych. Zweryfikuj ten projekt w programie do symulacji układów analogowych. Pobierz i zainstaluj darmowy program LTSpice (http://www.linear.com/designtools/software/). Zaimplementuj w nim filtr analogowy, którego schemat układowy jest przedstawiony na rysunku 7.6. Przeprowadź analizę zmienno-prądową układu i wyznacz w programie cha-kę częstotliwościową układu. Porównaj ją z ch-ką, która została zaprezentowana w podrozdziale 6.6 w [40].
Użyj źródła napięcia AC o częstotliwości 10 kHz oraz amplitudzie 10 V jako sygnału wejściowego (Edit/Components/Voltage). Zastosuj uniwersalny wzmacniacz operacyjny: przejdź do Edit/Components/, potem wybierz UniversalOamp2 z katalogu /Opamps oraz zasil go napięciem stałym DC o wartości ±15 V. Przeprowadź analizę AC układu: Simulation/Edit Simulation Command/AC Analysis.

Laboratorium 8
Filtry cyfrowe IIR (rekursywne)

Streszczenie Podczas tego laboratorium poznamy rekursywne filtry cyfrowe o nieskończonej odpowiedzi impulsowej IIR (*Infinite Impulse Response*): ich strukturę obliczeniową (równanie różnicowe), transmitancję (iloraz dwóch wielomianów zmiennej zespolonej z, pochodzącej z transformacji Z) oraz metody projektowania ich współczynników b_k, a_k (występujących zarówno w równaniu czasowym oraz w transmitancji). Poznamy dwie metody projektowania: 1) metodę "Zer i Biegunów" doboru miejsc zerowych wielomianów transmitacji, 2) metodę transformacji biliniowej, polegającą na przekształcania filtrów analogowych na cyfrowe.

TEMAT #1: Filtry cyfrowe IIR: równanie, transmitancja, charakterystyka (odpowiedź) częstotliwościowa oraz odpowiedź impulsowa. Metoda "zer i biegunów" Rekursywne filtry cyfrowe IIR są zdefiniowane następującym równaniem wejście-wyjście $x(n) \rightarrow y(n)$:

$$y(n) \leftarrow [\text{IIR: } b_k, a_k] \leftarrow x(n), \tag{8.1}$$

$$y(n) = \underbrace{\sum_{k=0}^{M} b_k x(n-k)}_{\text{ostatnie wejścia}} - \underbrace{\sum_{k=1}^{N} a_k y(n-k)}_{\text{poprzednie wyjścia}}, \tag{8.2}$$

$$y(n) = \underbrace{[b_0 x(n-0) + b_1 x(n-1) + \ldots + b_M x(n-M)]}_{\text{ostatnie wejścia}} - \underbrace{[a_1 y(n-1) + \ldots + a_N y(n-N)]}_{\text{poprzednie wyjścia}}. \tag{8.3}$$

gdzie:

- $x(n)$ - liczby/próbki wejściowe,
- $y(n)$ - liczby/próbki wyjściowe,
- b_k, a_k - specjalnie dobrane wartości współczynników/wag filtra, decydujące o tym, które częstotliwości są przez filtr przepuszczane a które nie są.

Aktualna wartość wyjściowa filtra $y(n)$ jest równa:

1. ważonej sumie ostatnich $M+1$ liczb wejściowych $x(n-k)$ (pomnożonych przez odpowiadające wagi filtra b_k), z uwzględnieniem aktualnej liczby wejściowej,
2. od której odejmujemy ważoną sumę ostatnio obliczonych N liczb wyjściowych $y(n-k)$ (pomnożonych przez odpowiadające wagi filtra a_k), oczywiście nie uwzględniająca obliczanej próbki wyjściowej.

Poniżej podano definicję transformacji Z (pierwsza), pełniącą taką samą rolę w świecie cyfrowym jak transformacja Laplace'a w świecie analogowym, oraz pokazano jej związek z dyskretną w czasie transformacją Fouriera DtFT (druga):

$$X(z) = \sum_{n=-\infty}^{+\infty} x(n)(z)^{-n} \qquad X(f) = \sum_{n=-\infty}^{+\infty} x(n)\left(e^{j2\pi \frac{f}{f_{pr}}}\right)^{-n}. \tag{8.4}$$

Transformacja Z posiada następującą właściwość: $Z(x(n-n_0)) = z^{-n_0}X(z)$, czyli transformacja Z sygnału opóźnionego o n_0 próbek, jest równa transformacji Z sygnału nieopóźnionego, czyli $X(z)$, pomnożonej przez z^{-n_0}. Dzięki temu po wykonaniu transformacji Z obu stron równania (8.2), otrzymujemy transmitancję $H(z)$ rekursywnego filtra cyfrowego IIR (zwróć uwagę, że zawsze $a_0 = 1$):

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}} == \frac{b_0}{1} \frac{(1 - z_1 z^{-1})\ldots(1 - z_M z^{-1})}{(1 - p_1 z^{-1})\ldots(1 - p_N z^{-1})}. \tag{8.5}$$

Ponieważ z równania (8.4) wynika, że transformacja Fouriera DtFT jest szczególnym przypadkiem transformacji Z dla $z = e^{j2\pi \frac{f}{f_{pr}}}$, dlatego charakterystykę częstotliwościową $|X(f)|$ filtra otrzymuje się podstawiając do $H(z)$ ww. zależność:

$$H(f) = H(z) \quad \text{dla} \quad z = e^{j2\pi \frac{f}{f_{pr}}}, \tag{8.6}$$

$|H(f)|$ - to ch-ka (odpowiedź) amplitudowo-częstotliwoścowa (wzmocnienie w funkcji częstotliwości), a $\angle H(f)$ - ch-ka (odpowiedź) fazowo-częstotliwościowa (pośrednio opóźnienie w funkcji częstotliwości). Przykładowo, sygnał wejściowy $x(n) = A_0 \sin(2\pi(f_0/f_{pr})n)$ na wyjściu filtra jest równy $y(n) = (A_0 \cdot |H(f_0)|)\sin(2\pi(f_0/f_{pr})n + \angle H(f_0))$, czyli jest przesunięty o $n_0 = \frac{\angle H(f_0)}{2\pi(f_0/f_{pr})}$ próbek.

Dzięki operacjom ważonego uśredniania ostatnich próbek wejściowych i wyjściowych, występującym w równaniu (8.2) w postaci dwóch sum, filtr usuwa z sygnału wejściowego składowe o wybranych częstotliwościach i jest: dolno-przepustowy (LP low-pass), górno-przepustowy (HP high-pass), pasmowo-przepustowy (BP band-pass) lub pasmowo-zaporowy (BS band-stop). Dobór odpowiednich wartości współczynników b_k oraz a_k wielomianów transmitancji (8.5) jest nazywany projektowaniem filtra, natomiast stosowanie równania (8.2) — filtracją cyfrową. W programie 8.1 pokazano implementację kolejnych etapów projektowania i zastosowania przykładowego filtra cyfrowego IIR z użyciem funkcji Matlaba (uważnie przeczytaj komentarze umieszczone po prawej stronie).

Metoda "zer i biegunów transmitancji". W metodzie tej projektujemy filtry cyfrowe IIR, ale nie dobierajmy wartości rzeczywistych współczynników b_k i a_k wielomianów transmitancji, tylko wartości zespolone z_k i p_k miejsc zerowych tych wielomianów. Umieszczamy zera i bieguny (zawsze parami sprzężone, tak jak w przypadku filtrów analogowych) w odpowiedniej pozycji w stosunku do okręgu jednostkowego: $z = e^{j2\pi \frac{f}{f_{pr}}}$, $f = var$. Mówiąc poglądowo: na okrąg jest nałożona oś częstotliwości i cały jego obieg odpowiada zmianie od 0 do f_{pr} herców. Zero umieszczone na okręgu *zeruje* częstotliwość, wynikającą z punktu, w którym się znajduje. Biegun leżący wewnątrz okręgu, ale blisko niego, wzmacnia częstotliwość, związaną z najbliższym punktem na okręgu. Filtr cyfrowy IIR jest stabilny, kiedy jego bieguny leżą wewnątrz okręgu jesdnostkowego (położenie zer jest dowolne). Metoda "zer i biegunów" jest zaimplementowana w programie 8.2.

Zalety i wady filtrów cyfrowych IIR. Zaletą filtrów IIR jest mała liczba współczynników b_k i a_k, która wystarczy do zapewnienia stromości filtra. Wadą filtrów IIR jest trudność projektowania (niebezpieczeństwo niestabilności w implementacjach z ograniczoną precyzją obliczeniową) oraz odstępstwa od liniowości ch-ki fazowej (brak stałego opóźnienia różnych składowych na wyjściu filtra, prowadzący do zmiany kształtu sygnału przepuszczanego).

Listing 8.1: Projekt i użycie przykładowego pasmowo-zaporowego filtra cyfrowego IIR w języku Matlab

```
% cps_08_matlab.m
clear all; close all;

fpr=2000; f1=400; f2=600; N=8; Rp=3; Rs=80;         % 1) wymagania dla filtra
[b,a] = ellip(N, Rp, Rs, [f1,f2]/(fpr/2), 'stop');  % 2) projekt filtra
Npunkt=1000; freqz(b,a,Npunkt,fpr);                 % 3) sprawdzenie H(f) filtra
Nx=1000; dt=1/fpr; t=dt*(0:Nx-1); fx1=10; fx2=500;  % 4) wymagania dla sygnalu
x = sin(2*pi*fx1*t) + sin(2*pi*fx2*t);              % 5) generacja sygnalu
y = filter(b,a,x);                                  % 6) filtracja sygnalu - dwie sumy
figure; plot(t,x,'b-',t,y,'r-'); title('We/Wy');    % 7) wynik filtracji
```

Problem 8.1 (* Przykładowe funkcje Matlaba wspierające filtrację cyfrową IIR). Zapoznaj się z programem 8.1, uruchom go, obejrzyj rysunki. Zmień częstotliwości graniczne `[f1,f2]` pasma zaporowego filtra. Zwiększ tłumienie filtra w paśmie zaporowym. Zmień typ filtra na: 1) pasmowo-przepustowy, 2) górnoprzepustowy, 3) dolnoprzepustowy. Znajdź alternatwy dla funkcji `ellip()`.

Listing 8.2: Transmitancja i ch-ka częstotliwościowa filtra cyfrowego IIR w Matlabie

```
% cps_08_iir_intro.m
clear all; close all;

% Wybor/projekt wspolczynnikow filtra cyfrowego IIR
fpr = 2000;                          % czestotliwosc probkowania
if(1)  % ### dobor wartosci wspolczynnikow wielomianow transmitancji
   b = [2,3];                        % [ b0, b1 ]
   a = [1, 0.2, 0.3, 0.4];           % [ a0=1, a2, a3, a4]
   z = roots(b); p = roots(a);       % [b,a] --> [z,p]
else   % ### dobor miejsc zerowych wielomianow transmitancji
   gain = 0.001;
   z = [ 1,   1    ] .* exp(j*2*pi*[ 600, 800 ]/fpr);   z = [z conj(z)];
   p = [ 0.99,0.99 ] .* exp(j*2*pi*[ 100, 200 ]/fpr);   p = [p conj(p)];
   b = gain*poly(z), a = poly(p), pause       % [z,p] --> [b,a]
end
figure;
alfa = 0 : pi/1000 : 2*pi; c=cos(alfa); s=sin(alfa);
plot(real(z),imag(z),'bo', real(p),imag(p),'r*',c,s,'k-'); grid;
title('Zera i Bieguny'); xlabel('Real()'); ylabel('Imag()'); pause

% Weryfikacja odpowiedzi filtra: amplitudowej, fazowej, impulsowej, skokowej
f = 0 : 0.1 : 1000;                % czestotliwosc w hercach
wn = 2*pi*f/fpr;                   % czestotliwosc katowa
zz = exp(-j*wn);                   % zmienna "z" transformacji Z (u nas z^(-1))
H = polyval(b(end:-1:1),zz) ./ polyval( a(end:-1:1),zz); % stosunek dwoch wielomianow
% H = freqz(b,a,f,fpr)                                   % alternatywna funkcja Matlab
figure; plot(f,20*log10(abs(H))); xlabel('f [Hz]'); title('|H(f)| [dB]'); grid; pause
figure; plot(f,unwrap(angle(H))); xlabel('f [Hz]'); title('angle(H(f)) [rad]'); grid; pause

zz = exp(j*wn);                    % zmienna transformacji Z
H = polyval(b,zz) ./ polyval(a,zz); % stosunek dwoch wielomianow dla zz=exp(j*om)
% H = freqz(b,a,f,fpr)             % alternatywna funkcja Matlaba
figure; plot(f,20*log10(abs(H))); xlabel('f [Hz]'); title('|H(f)| [dB]'); grid; pause
figure; plot(f,unwrap(angle(H))); xlabel('f [Hz]'); title('angle(H(f)) [rad]'); grid; pause

% ... ciag dalszy nastapi
```

Problem 8.2 (* Transmitancja i odpowiedź częstotliwościowa filtra cyfrowego). Przeanalizuj kod programu 8.2, w którym dobierane są wartości współczynników wielomianów transmitancji $H(z)$ (8.5) oraz rysowane są otrzymywane charakteystyki filtra: położenie zer z_k i biegunów p_k transmitancji oraz odpowiedź (charakterystyka) amplitudowa i fazowa. Aby zagwarantować stabilność filtra, bieguny jego transmitancji powinny się znajdować wewnątrz okręgu jednostkowego w przestrzeni zmiennej zespolonej z transformacj Z, natomiast zera transmitancji filtra — mogą leżeć gdziekolwiek. Uruchom program, zapoznaj się z rysunkami dla obecnie wybranych wartości zmiennych `b,a,z,p`. Wytłumacz pochodzenie maksimów ("pików/górek") oraz minimów ("szczelin/dołków"), widocznych na charakterystyce amplitudowej filtra, oraz pochodzenie skoków o wartości $\pm\pi$, widocznych na charakterystyce fazowej filtra. Zapamiętaj: zero transmitancji jest wykorzystywane jako *killer* wybranej częstotlliwości (generuje "szczelinę/dołek" w odpowiedzi amplitudowej filtra), natomiast biegun transmitancji - jest wykorzystywany jako *wzmacniacz* wybranej częstotliwości (generuje "szczyt/górkę" w odpowiedzi amplitudowej). Zera i bieguny transmitancji, jeśli mają wartości zespolone, to występują w parach sprzężonych - liczba zespolona i jej sprzężenie (dzięki temu współczynniki wielomianów transmitancji są rzeczywiste, co jest warunkiem koniecznym). W celu znalezienia odpowiednich wartości współczynników b_k i a_k transmitancji, korzystamy z równania eq. (8.5) w programie 8.2: z jest podnoszone do dodatnich potęg, a współczynniki transmitancji są uporządkowane od najniższych numerów, czyli od b_0 and $a_0 = 1$. Jako sprawdzenie, czy rozumiesz procedurę projektowania, spróbuj zaprojektować filtr cyfrowy, który równoceśnie usuwa z sygnału składową o częstotliwości 300 Hz oraz silnie wzmacnia składową o częstotliwości 400 Hz.

Problem 8.3 (* Projektowanie rekursywnych filtrów cyfrowych IIR metodą "Zer & Biegunów"). Zmodyfikuj program 8.2 i spróbuj zaprojektować transmitancję dobrego filtra cyfrowego IIR, przepuszczającego tylko częstotliwości leżące w podanym przedziale (wybierz tylko jeden z czterech, możliwych projektów):

1. low-pass: [od 0 do 100] Hz;

2. high-pass: [od 100 do $\frac{f_{pr}}{2}$] Hz;
3. band-pass: [od 400 do 600] Hz;
4. band-stop: [od 0 do 400] Hz oraz [od 600 do $\frac{f_{pr}}{2}$] Hz.

Filtr ten powinien usuwać pozostałe częstotliwości na swoim wyjściu. Spróbuj osiągnąć cel dobierając wartości: 1) współczynników $\{b_k, a_k\}$ wielomianów transmitancji, oraz 2) pierwiastki $\{z_k, p_k\}$ tych wielomianów. Która metoda była Twoim zdaniem prostsza?

Listing 8.3: Fitracja cyfrowa w akcji

```
% ... kontynuacja programu cps_08_iir_intro.m

% Sygnal wejsciowy x(n) - dwie sinusoidy: 20 and 500 Hz
Nx = 1000;                                 % liczba probek sygnalu
dt = 1/fpr; t = dt*(0:Nx-1);               % chwile probkowania
%x = zeros(1,Nx); x(1) = 1;                % sygnal delta Kroneckera (impuls jednostkowy)
x = sin(2*pi*20*t+pi/3) + sin(2*pi*500*t+pi/7);  % suma dwoch sinusow

% Rekursywna filtracja cyfrowa IIR: x(n) --> [ b, a ] --> y(n)
% y=filter(b,a,x);                         % funkcja Matlaba
M = length(b);                             % liczba wspolczynnikow {b}
N = length(a); a = a(2:N); N=N-1;          % liczba wspolczynnikow {a}, usun a0=1
bx = zeros(1,M);                           % bufor bx na probki wejsciowe x(n)
by = zeros(1,N);                           % bufor by na probki wyjsciowe y(n)
for n = 1 : Nx                             % PETLA GLOWNA
    bx = [ x(n) bx(1:M-1) ];               % nowa probka x(n) na poczatek bufora bx
    y(n) = sum( bx .* b ) - sum( by .* a ); % filtracja = dwie srednie wazone, y(n)=?
    by = [ y(n) by(1:N-1) ];               % zapamietanie y(n) w buforze by
end                                        %

% RYSUNKI: porownanie wejscia i wyjscia filtra
figure;
subplot(211); plot(t,x); grid;             % sygnal wejsciowy x(n)
subplot(212); plot(t,y); grid; pause       % sygnal wyjsciowy y(n)
figure; % widma FFT drugicj polowek sygnalow x(n) i y(n) (bez stanow przejsciowych)
k=Nx/2+1:Nx;  f0 = fpr/(Nx/2);  f=f0*(0:Nx/2-1);
subplot(211); plot(f,20*log10(abs(2*fft(x(k)))/(Nx/2))); grid;
subplot(212); plot(f,20*log10(abs(2*fft(y(k)))/(Nx/2))); grid; pause
```

Problem 8.4 (* Testowanie filtracji cyfrowej sygnalow). Przeanalizuj kod Matlaba z listing 8.3. Na początku jest generowana suma dwóch sinusoid o częstotliwościach 20 Hz oraz 500 Hz. Następnie sygnał jest poddany filtracji cyfrowej IIR z użyciem dwóch wektorów współczynników wagowych: `b` oraz `a`. Wagi filtra zostały przez nas wcześniej zaprojektowane w programie 8.2.
Dodaj kod programu 8.3, implementującego filtrację cyfrową według równania (8.2), na koniec programu 8.2, w którym wcześniej zaprojektowaliśmy współczynniki b_k i a_k transmitancji filtra cyfrowego, metodą "Zer i Biegunów". Następnie sprawdź skuteczność filtracji sygnału. Zaprojektuj nowy filtr (tzn. nowe wartości jego wag): 1) wzmacniającego pierwszego sinusa i silnie tłumiącego drugiego sinusa, 2) tłumiącego pierwszego i wzmacniającego drugiego sinusa.

Problem 8.5 (* Stabilność filtra — obserwacja odpowiedzi impulsowej filtra). Filtr cyfrowy IIR jest stabilny, kiedy bieguny jego transmitancji (pierwiastki wielomianu mianownika) leżą wewnątrz okręgu jednostkowego. Położenie zer transmitacji (pierwiastków wielomianu licznika) może być dowolne. Sprawdź jaką odpowiedź impulsową posiada filtr zaprojektowany przez Ciebie w ostatnim ćwiczeniu, czyli jaka jest jego odpowiedź na sygnał impulsu jednostkowego: `x=zeros(1,Nx); x(1)=1`? Kiedy filtr jest stabilny, to powinniśmy na wyjściu filtra obserwować oscylacje gasnące do zera po pewnym czasie — dotyczy to sygnału `y(n)`. Umieść jeden lub więcej biegunów Twojego filtra na okręgu jednostkowym. Czy filtr jest stabilny? Czy odpowiedź filtra gaśnie do zera? Umieść jeden biegun na zewnątrz okręgu jednostkowego i sprawdź ponownie.

Problem 8.6 (Usuwanie zakłóceń oscylacyjnych z nagrań sygnałów rzeczywistych).** Nagraj dowolny sygnał, np. mowę, muzykę, radio, TV (patrz instrukcja z pierwszego laboratorium) lub pobierz dowolny sygnał dźwiękowy ze strony *FindSounds*. Dodaj do Twojego sygnału sinusoidę o dużej amplitudzie, mającą częstotliwość 1500 Hz. Odsłuchaj sygnał wyniku dodawania. Zaprojektuj filtr cyfrowy usuwający zakłócenie. Zastosuj go. Sprawdź słuchowo na ile filtr był skuteczny. Zmień zakłócenie na sygnał z sinusoidalną modulcją częstotliwości (SFM): `f0=1500; df=250; fm=0.25;` (częstotliwość nośna, głebokość modulacji, częstotliwość modulująca, wszystko w hercach — patrz laboratorium "Sygnały generacja"). Zaprojektuj filtr do zredukowania poziomu zakłócenia (do jego maksymalnego usunięcia) i go zastosuj. Przeprowadź odsłuch — czy się udało?

TEMAT #2: Rekursywne filtry cyfrowe IIR otrzymane w wyniku transformacji biliniowej filtrów analogowych. Transmitację $H(s)$ zaprojektowanego filtra analogowego można transformować na transmitancję $H(z)$ filtra cyfrowego. Istnieje kilka metod. My poznamy transformację biliniową. $H(s)$ filtra analogowego przechodzi w $H(z)$ filtra cyfrowego w wyniku następującego podstawienia za zmienną s funkcji zmiennej z:

$$s = 2f_{pr}\frac{z-1}{z+1}, \quad \text{gdzie} \quad s = j2\pi f_a, \; z = e^{j2\pi\frac{f_d}{f_{pr}}}. \tag{8.7}$$

W równaniu (8.7) f_d oznacza częstotliwość cyfrową (*digital*), natomiast f_a — odpowiadającą jej częstotliwość analogową (*analog*). W wyniku transformacji wielomiany zmiennej s transmitancji $H(s)$ (filtra analogowego) konwertowne są w wielomiany zmiennej$z$ transmitacji $H(z)$ (filtra cyfrowego). Zgodnie z tym równaniem:

$$(s - z_k) = 2f_{pr}\frac{z-1}{z+1} - z_k = (2f_{pr} - z_k)\,\frac{z - \frac{2f_{pr}+z_k}{2f_{pr}-z_k}}{z+1}, \tag{8.8}$$

każde miejsce zerowe z_k transmitancji $H(s)$ filtra analogowego "generuje" (jest transformowane na): 1) jedno zero transmitancji $H(z)$ filtra cyfrowego o wartości: $\frac{2f_{pr}+z_k}{2f_{pr}-z_k}$, oraz 2) jeden biegun transmitancji $H(z)$ o wartości "-1". Dodatkowo w wyniku transformacji wzmocnienie filtra analogowego jest mnożone przez $(2f_{pr} - z_k)$. Ponieważ transformacja biliniowa jest nieliniową funkcją częstotliwości (deformuje oś częstotliwości), filtr analogowy konwertowany na filtr cyfrowy powinien zostać zaprojektowany an na trochę inne, *zdeformowane* częstotliwości analogowe, określone następującym wzorem $f_a = 2f_{pr} \cdot \tan\left(\pi\frac{f_d}{f_{pr}}\right)/(2\pi)$. Po transformacji, częstotliwości te przechodzą na zadane częstotliwości cyfrowe (ponieważ złożenie funkcji i funkcji odwrotnej $x = f(f^{-1}(x))$ jest powrotem do punktu początkowego).

Problem 8.7 (Transformacja biliniowa filtrów analogowych na cyfrowe w jednym kroku).** Przeanalizuj kod Matlaba NASZEJ transformacji biliniowej, przedstawiony na listingu 8.4. Znajdź w nim wszystkie elementy składowe równania (8.8). Funkcja powinna być wywoływana w identyczny sposób jak funkcja Matlaba `[z,p,gain] = bilinear(z,p,gain,fpr)` — wyniki obu funkcji powinny być identyczne. Sprawdź to w następujący sposób. Zaprojektuj dowolny filtr analogowy za pomocą wybranej funkcji Matlaba: `butter()`, `cheby1()`, `cheby2()`, `ellip()` — pamiętaj o dodaniu na końcu wywołani funkcji opcji `'s'`: wykorzystaj programy z laboratorium dot. filtrów analogowych, narysuj położenie zer&biegunów transmitancji filtra analogowego oraz jego charakterystykę amplitudowo-częstotliwościową (odpowiedź częstotliwościową). Następnie transformuj obliczone wartości parametrów filtra analogowego `z,p,gain` do świata cyfrowego. Narysuj położenie zer &biegunów transmitancji filtra cyfrowego oraz jego charakterystykę amplitudowo-częstotliwościową. Narysuj na jednym rysynku ch-ki czestotliwościowe obu filtrów, analogowego i cyfrowego. Zwróć uwagę na różne wartości częstotliwości granicznych w obu filtrach, wynikających z użycia transformacji: $f_a = 2f_{pr} \cdot \tan\left(\pi\frac{f_d}{f_{pr}}\right)/(2\pi)$.

Transformacja biliniowa filtrów analogowych na filtry cyfrowe jest wytłumaczona na rysunku 8.1. Z kolej rysunek 8.2 pokazuje zależność pomiędzy częstotliwością analogową i cyfrową.

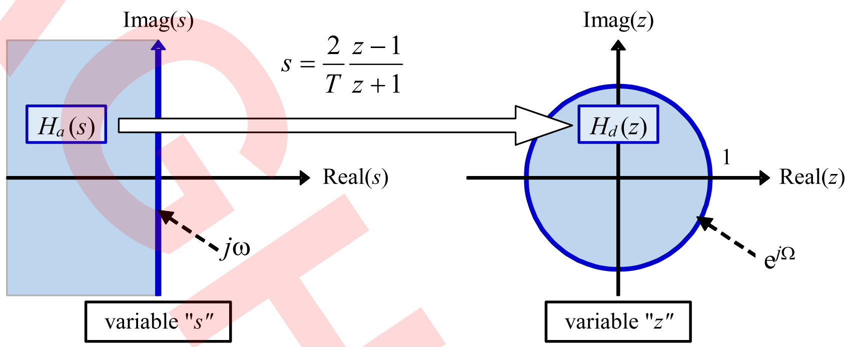

Fig. 8.1: Graficzna ilustracja transformacji biliniowej filtra analogowego, mającego transmitację $H_a(s)$, na filtr cyfrowy, mający transmitancję $H_d(z)$. Linia $s = j\omega$ jest przekształcana na okrąg $z = e^{j\Omega}$ o promieniu 1. Lewa półpłaszczyzna zmiennej s jest przekształacana wnętrze koła jednostkowego: dzięki temu stabilny filtr analogowy (mający zera TF w lwe półpłaszczyźnie) jest transformowany na stabilny filtr cyfrowy (mający zera wewnątrz okręgu jednostkowego).

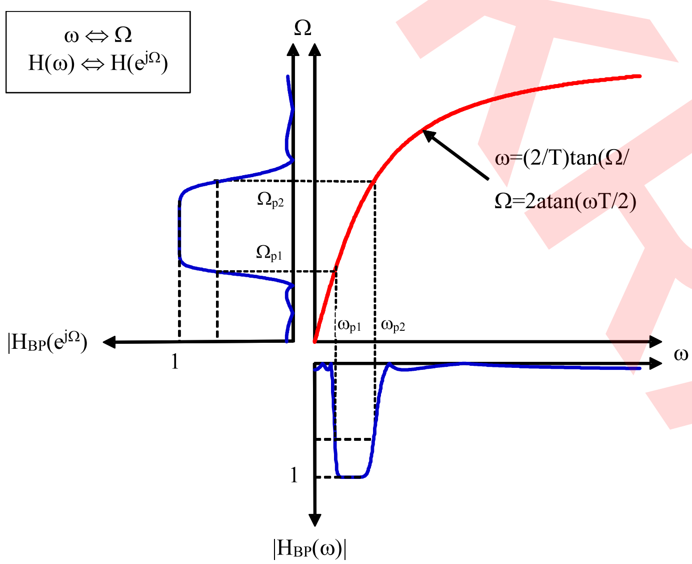

Fig. 8.2: Ilustracja graficzna pokazująca jak charakterystyka częstotliwościowa filtra analogowego (po lewej) jest przekształcana na charakterystykę częstotliwościową filtra cyfrowego (po prawej) poprzez krzywą nieliniowej funkcji atan() tranaformacji biliniowej.

Listing 8.4: Kod Matlaba transformacji biliniowej

```
function [zz,pp,ggain] = bilinearMY(z,p,gain,fpr)
% Transformacja biliniowa: H(s) (filtr analogowy) --> H(z) (filtr cyfrowy)
% zera, bieguny, wzmocnienie (z,p,gain) --> zera, bieguny, wzmocnienie (zz,pp,ggain)

pp = []; zz = []; ggain = gain;
for  k=1:length(z)   % transformacja zer "analogowych"
     zz = [ zz (2*fpr+z(k))/(2*fpr-z(k)) ];
     ggain = ggain*(2*fpr-z(k));
end
for  k=1:length(p)   % transformacja biegunow "analogowych"
```

```
    pp = [ pp (2*fpr+p(k))/(2*fpr-p(k)) ];
    ggain = ggain/(2*fpr-p(k));
end
if (length(p)>length(z)) zz = [ zz -1*ones(1,length(p)-length(z)) ]; end
if (length(p)<length(z)) pp = [ pp -1*ones(1,length(z)-length(p)) ]; end
```

Problem 8.8 (Szczegóły projektowania filtrów cyfrowych IIR za pomocą transformacji biniowej).** Przeanalizuj kod Matlaba z listingu 8.5. Dodaj na jego końcu kod Matlaba z listingu 8.3. Zaprojektuj filtr cyfrowy IIR typu low-pass o częstotliwości granicznej 3 dB (odcięcia) równej 100 Hz, który przepuści tylko składową sygnalu 20 Hz. Następnie filtr cyfrowy typu band-pass o szerokości pasma przepuszczania [400-600] Hz, który przepuści tylko składową 500 Hz. Pokaż gdzie leżą zera&bieguny transmitancji obu filtrów oraz narysuj ich odpowiedzi częstotliwościowe. Przeprowadź filtrację sygnału sumy dwóch składowych, porównaj sygnał wejściowy i wyjściowy oraz ich widma FFT.

Problem 8.9 (* Opóźnienie filtrów cyfrowych typu IIR — obserwacja opóźnienia wprowadzanego przez filtr).** Przeanalizuj kod Matlaba z listingu 8.5. Dodaj na jego końcu kod Matlaba z listingu 8.3. Metodą transformacji biliniowej zaprojektuj dolnoprzepustowy filtry cyfrowe IIR Butterwortha, Czebyszewa 1, Czebyszewa 2 oraz eliptyczny o takiej samej liczbie biegunów transmitancji oraz takiej samej częstotliwości granicznej f_0. Następnie wygeneruj sinusoidę o częstotliwości przepuszczanej przez wszystkie filtry, np. równej połowie częstotliwości granicznej $\frac{f_0}{2}$. Potem dokonaj filtracji sygnału z użyciem wszystkich filtrów oraz narysuj na jednym rysunku wszystkie otrzymane sygnały wyjściowe. Zwróć uwagę na opóźnienie wprowadzane przez każdy filtr. Ile ono wynosi? Od czego ono zależy? Czy umiałbyś je obliczyć na podstawie charakterystyki fazowo-częstotliwościowej każdego filtru $\angle H(f)$? Jeśli tak, to zrób to i porównaj wynik z opóźnieniem obserwowanym. Narysuj na jednym rysunku charakterystyki fazowo-częstotliwościowe czterech zaprojektowanych filtrów. Przypomnijmy: opóźnienie składowej o częstotliwości f_0 na wyjściu filtra jest równe $n_0 = \frac{\angle H(f_0)}{2\pi(f_0/f_{pr})}$ próbek.

Listing 8.5: Projektowanie filtrów cyfrowych IIR metodą transformacji biliniowej fitrów analogowych

```
% cps_08_iir_projekt.m
clear all; close all;

% Wymagania odnosnie filtracji cyfrowej
fpr = 2000;       % czestotliwosc probkowania
f1 = 400;         % czestotliwosc dolna filtra band-pass
f2 = 600;         % czestotliwosc gorna filtra band-pass
N  = 6;           % liczba biegunow prototypu analogowego
Rp = 3; Rs = 60;  % oscylacje (R-ripples) w [dB] w pasmie PASS i STOP

% Wymagania cyfrowe --> analogowe
f1 = 2*fpr*tan(pi*f1/fpr) / (2*pi);
f2 = 2*fpr*tan(pi*f2/fpr) / (2*pi);
w0 = 2*pi*sqrt(f1*f2);
dw = 2*pi*(f2-f1);

% Projekt filtra analogowego
[z,p,gain] = ellipap(N,Rp,Rs);        % prototyp analogowy eliptyczny dolno-przepustowy
[z,p,gain] = lp2bsMY(z,p,gain,w0,dw); % transformacja czestotliwosci NASZA: LP -> BS
   b = gain*poly(z); a = poly(p);      % zera & bieguny analogowe --> wspolczynniki [b,a]
   freqs(b,a), pause                   % odpowiedz czestotliwosciowa filtra analogowego [b,a]
% Dodaj rysunek z polozeniem zer & biegunow filtra analogowego
% Oblicz sam i pokaz odpowiedz czestotliwosciowa filtra analogowego

% Konwersja filtra analogowego na cyfrowy
[z,p,gain] = bilinearMY(z,p,gain,fpr); % funkcja biliearMY() NASZA
b = real( gain*poly(z) ); a = real( poly(p) );  % zera & bieguny cyfrowe --> wsp. [b,a]
fvtool(b,a), pause                              % wyswietlenie zaprojektowanego filtra

% Dodaj rysunek pokazujacy polozenie zer & biegunow filtra cyfrowego
% Oblicz i pokaz odpowiedz czestotliwosciowa filtra, w Matlabie H=freqz(b,a,f,fpr)
% Dokonaj filtracji wybranych sygnalow, w Matlabie y=filter(b,a,x)
```

TEMAT #3: Zastostosowania rekursywnej filtracji IIR do sygnałów rzeczywistych. Cyfrowe filtry IIR (rekursywne), w przeciwieństwie do FIR (nierekursywnych), mają strome charakterystyki amplitudowo-częstotliwościowe dla względnie małej liczby współczynników wagowych, np. 6-8 biegunów. Ale powinny być one projektowane bardzo starannie z powodu istnienia niebezpieczeństwa utraty stabilności (występowanie pętli sprzężenia zwrotnego) - bieguny muszą leżeć wewnątrz okręgu jednostkowego. Ich wadą jest także niemożliwość uzyskania idealnie liniowej charakterystki fazowo-częstotliwościowej, powodującej że kształt sygnału przepuszczanego na wyjściu filtra może być zmieniony, w miejszym lub większym stopniu.

Problem 8.10 (** Filtrowanie dźwięków rzeczywistych).** Znajdź w Internecie różne nagrania dźwiękowe, np. pobierz kilka nagrań ze strony `FindSounds`. Zabaw się w inżyniera dźwięku: dodaj do siebie różne nagrania, np. mowa + wysokoczęstotliwościowy warkot jakiegoś silnika, mowa + wysokoczęstotliwościowy śpiew ptaka, wycie wilka/ryk lwa/ trąbienie słonia + wysokoczęstotliwościowy śpiew ptaka, itp. Oblicz i wyświetl widmo FFT (`fft()`) każdego pojedynczego sygnału oraz jego spektrogram (STFT)(`pspectrogram()`), oraz to samo dla sygnału sumy. Zaprojektuj rekursywny filtr cyfrowy IIR, który spróbuje odseprować pojedyncze źródło dźwięku z sygnału sumy, np. pozostawić tylko mowę a resztę usunąć. Oblicz i wyświetl odpowiedź częstotliwościową filtra w decybelach. Pokaż gdzie leżą na płaszczyźnie zespolonej zera i bieguny transmitancji filtra. Dokonaj filtracji sygnału sumy. Pokaż wynik, odsłuchaj go. Oblicz FFT i STFT (spektrogram) sygnału po filtrze, i wyświetl te widma. Porównaj je z widmami oryginalnego sygnału, jeszcze przed utworzeniem sumy.

Laboratorium 9
Filtry cyfrowe FIR (nierekursywne)

Streszczenie Podczas tego laboratorium nauczymy się projektować współczynniki wagowe cyfrowych filtrów nierekursywnych FIR (*Finite Impulse Response*). W porównaniu z filtrami cyfrowymi IIR, filtry FIR są zawsze stabilne oraz łatwo wymusza się idealną liniowość ich charakterystyki fazowo-częstotliwościowej. Ta ostatnia właściwosc powoduje, że wszystkie składowe częstotliwościowe, które filtr przepuszcza, są opóźniane o taki sam czas i kształt sygnału przechodzącego przez filtr nie zmienia się. Wadą filtrów FIR jest konieczność użycia o wiele większej liczby współczynników niż w filtrach IIR, gdyż tylko to gwarantuje stromość ch-ki amplitudowo-częstotliwościowej filtra. Poznamy trzy metody projektowania filtrów FIR: 1) metodę *okien*, 2) metodę odwrotnej DFT, 3) metodę optymalizacji średniokwadratowej (LS - *Least Squares*).

TEMAT #1: Filtry cyfrowe FIR: równanie filtracji, transmitancja, odpowiedź częstotliwościowa and impulsowa. Filtr cyfrowy o skończonej odpowiedzi impulsowej (FIR - Finite Impulse Response) jest zdefiniowany poprzez następujące równanie wejście $x(n) \rightarrow$ wyjście $y(n)$:

$$y(n) \leftarrow [\text{FIR: } b_k] \leftarrow x(n) \tag{9.1}$$

$$y(n) = \sum_{k=0}^{M} b_k x(n-k) = b_0 x(n-0) + b_1 x(n-1) + b_2 x(n-2) + \ldots + b_M x(n-M). \tag{9.2}$$

Próbka wyjściowa $y(.)$ w chwili n, czyli $y(n)$, jest sumą (*ważoną*) ostatnich $M+1$ próbek wejściowych $x(n-0), x(n-1), x(n-2), \ldots, x(n-M)$, pomnożonych przez współczynniki (wagi) filtra $b_0, b_1, b_2, \ldots, b_M$. Wartości tych wag należy odpowiednio dobrać, ponieważ decydują one o tym, jakie częstotliwości zostaną przez filtr przepuszczone a jakie usunięte. Transmitancja filtra jest zdefiniowana jako (z - zmienna transformacji Z, patrz temat #1 w poprzednim laboratorium):

$$H(z) = \frac{Y(z)}{X(z)} = b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_M z^{-M} \tag{9.3}$$

Dzięki operacji ważonego uśredniania, filtr usuwa składowe sygnału o wybranych częstotliwościach. Rozróżniamy filtry: dolno-przepustowe / nisko-częstotliwościowe (LP low-pass), górno-przepustowe (HP high-pass), pasmowo-przepustowe (BP band-pass) oraz pasmowo-zaporowe (BS band-stop). Wybór wartości współczynników b_k jest nazywany projektowaniem filtra, natomiast ich użycie w równaniu (9.2) — filtracją sygnału. Po podstawieniu $z = e^{j2\pi(f/f_{pr})}$, transmitancja filtra zamienia się w jego odpowiedź częstotliwościową $H(f)$, gdzie: f_{pr} - częstotliwość próbkowania, $|H(f)|$ - odpowiedź amplitudowa filtra (jego ch-ka amplitudowo-częstotliwościowa, czyli wzmocnienie filtra w funkcji częstotliwości), $\angle H(f)$ - odpowiedź fazowa filtra (ch-ka fazowo-częstotliwościowa, zawierająca informację o opóźnieniu sygnałów na wyjściu filtra w funkcji częstotliwości). Przykładowo, sygnał wejściowy $x(n) = A_0 \sin(2\pi(f_0/f_{pr})n)$ na wyjściu filtra jest równy $y(n) = (A_0 \cdot |H(f_0)|) \sin(2\pi(f_0/f_{pr})n + \angle H(f_0))$, czyli jest przesunięty o $n_0 = \frac{\angle H(f_0)}{2\pi(f_0/f_{pr})}$ próbek. Jeśli filtr FIR ma $M = 2P+1$ wag/współczynników b_k, które są symetryczne albo asymetryczne względem wagi środkowej, to odpowiedź fazowa filtra jest zawsze liniowa i filtr wprowadza jednakowe opóźnienie równe P próbek dla wszystkich częstotliwości. Wówczas próbka wejściowa `x(P+1)` odpowiada próbce wyjściowej `y(2*P+1)`, i tak dalej, czyli próbki `x=x(P+1:P+1+Nsamples-1)` odpowiadają próbkom `y=y(2*P+1:2*P+1+Nsamples-1)`. Odpowiedź impulsowa filtra $h(k)$ jest równa wartości jego wag b_k.
Synchronizacja sygnału wejściowego i wyjściowego w filtrze cyfrowym FIR jest wytłumaczona na rysunkach: 9.1 oraz 9.2 (pokazane jest której próbce wejściowej odpowiada konkretna próbka wyjściowa).

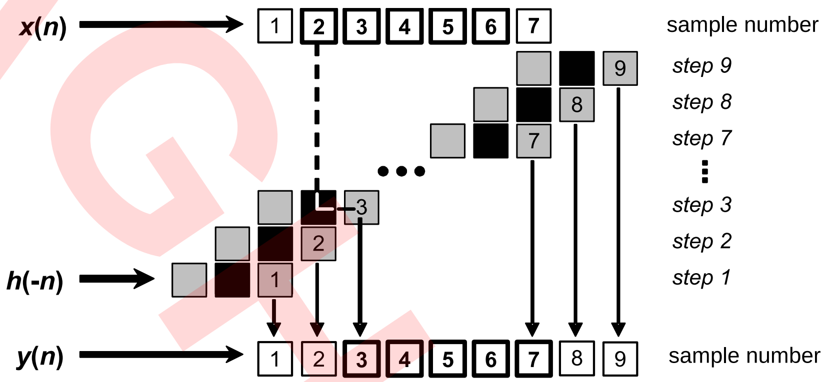

Fig. 9.1: Ilustracja graficzna *synchronizacji* próbek wejściowych i wyjściowych w filtrze cyfrowym FIR o długości nieparzystej. Oznaczenia: $x(n)$ - sygnał wejściowy ($N_x = 7$), $h(-n)$ - wagi filtra ($N = 2M+1 = 3$), $y(n)$ - sygnał wyjściowy ($N_y = N_x + N - 1 = 9$). Odpowiadające sobie próbki wejście-wyjście: $\{x(2), x(3), ..., x(6)\}$ and $\{y(3), y(4), ..., y(7)\}$, or $\{x(M+1), ..., x(N_x - M)\}$ and $\{y(N), ..., y(N_x)\}$

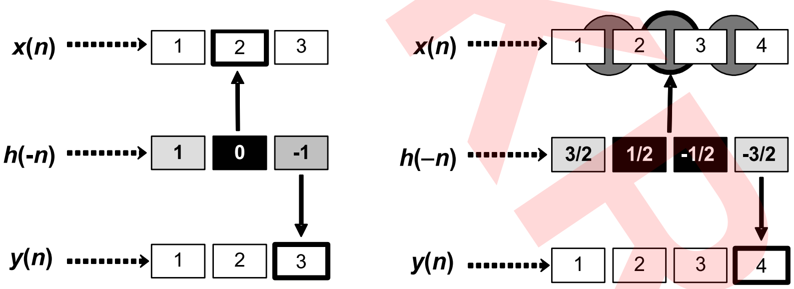

Fig. 9.2: Ilustracja graficzna *synchronizacji* próbek wejściowych i wyjściowych w filtrze cyfrowym FIR dla: (po lewej) nieparzystej liczby wag/współczynników filtra ($N = 3$), (po prawej) parzystej liczby współczynników filtra ($N = 4$)

Problem 9.1 (Podstawy fitracji cyfrowej FIR).** Przeanalizuj kod programu 9.1. W filtracji cyfrowej FIR, kolejność operacji jest następująca: 1) zaprojektowanie wag filtra `b`, 2) obserwacja kształtu otrzymanej odpowiedzi częstotliwościowej filtra $|H(f)|$, 3) filtracja sygnału wejściowego `x` z użyciem równania (9.2), 4) weryfikacja wyniku filtracji: obserwacja sygnału `x` (wejściowego) oraz `y` (wyjściowego) (bez oraz z kompensacją opóźnienia wprowadzanego przez filtr) oraz ich widm Fouriera. Wszystkie wymienione operacje są wykonane w programie, przedstawionym w listingu 9.1.
Uruchom program. Na początku, po kolei wybierz różne metody projektowania wag `b` filtra. Zapoznaj się z: położeniem pierwiastków (miejsc zerowych) wielomianu licznika transmitancji (mianownika nie ma), kształtem odpowiedzi amplitudowo-częstotliwościowej filtra (wzmocnienie w funkcji częstotliwości), kształtem sygnału wejściowego i wyjściowego oraz ich widmami DFT. Następnie, oblicz odpowiedź częstotliwościową filtra za pomocą funkcji Matlaba: `H=freqz(b,1,f,fpr)` oraz porównaj wynik z naszymi obliczeniami. Potem dokonaj filtracji sygnału testowego z użyciem funkcji Matlaba: `y=filter(b,1,x)` oraz `y=conv(x,b)`. Przedstaw na jednym rysunku trzy otrzymane sygnały wyjściowe z filtra (nasz program, `filter()`, `conv()`). Sprawdź jaka jest odpowiedź impulsowa i skokowa filtra (zastosuj odpowiednie sygnały wejściowe: `x=zeros(1,Nx); x(1)=1;` albo `x=ones(1,Nx);`). Zwróć uwagę na możliwość synchronizacji próbek sygnału wejściowego i wyjściowego (z uwzględnieniem opóźnienia wprowadzanego przez filtr o `P` próbek, gdzie `M=2*P+1`): w tym celu wygeneruj sygnał w całości przechodzący przez filtr oraz nałóż na siebie sygnał wejściowy i wyjściowy - bez synchronizacji oraz z jej użyciem.

Listing 9.1: Podstawy filtracji cyfrowej FIR w Matlabie

```
% cps_09_fir_intro.m
clear all; close all;

% Projekt/wybor wartosci wspolczynnikow "b" filtra FIR
fpr = 2000;                    % czestotliwosc probkowania
M  = 101;                      % liczba wspolczynnikow filtra
b = [ 1 2 3 ];                 % wagi filtra [b0, b1, b2, b3,...]
%b =  fir1(M-1, 100/(fpr/2));  % metoda okien: M wspolczynnkow, filtr low-pass, f0=100Hz
%b =  fir2(M-1,[0 75 250 fpr/2]/(fpr/2),[1 1 0 0]);        % odwrotne DFT: czestotliwosc->wzmocnienie
%b = firls(M-1,[0 75 250 fpr/2]/(fpr/2),[1 1 0 0],[1 10]); % optymalizacja LS
%b = firpm(M-1,[0 75 250 fpr/2]/(fpr/2),[1 1 0 0],[1 10]); % minimalny blad maksymalny
        figure; stem(b); title('b(k)'); grid; pause

% Polozenie zer transmitancji
z = roots(b);                   % pierwiastki wielomianu licznika
    figure;
    var = 0 : pi/1000 : 2*pi; c=cos(var); s=sin(var);
    plot(real(z),imag(z),'bo', c,s,'k-'); grid;
    title('TF Zeros'); xlabel('Real()'); ylabel('Imag()'); pause

% Weryfikacja zaprojektowanego filta
% Odpowiedzi: amplitudowa, fazowa, impulsowa, skokowa
f = 0 : 0.1 : 1000;             % czestotliwosc w hercach
wn = 2*pi*f/fpr;                % czestotliwosc katowa unormowana (/fpr)
zz = exp(-j*wn);                % odwrotnosc zmiennej transformacji Z: z^(-1)
H = polyval(b(end:-1:1),zz);    % odpowiedz czestotliwosciowa, transmitancja dla zz=exp(-j*wn)
% H = freqz(b,1,f,fpr);         % to samo za pomoca jednej funkcji Matlaba
    figure; plot(f,20*log10(abs(H))); xlabel('f [Hz]'); title('|H(f)| [dB]'); grid; pause
    figure; plot(f,unwrap(angle(H))); xlabel('f [Hz]'); title('angle(H(f)) [rad]'); grid; pause

% Sygnal wejsciowy x(n) - suma dwoch sinusoid: 20 oraz 500 Hz
Nx = 2000;                      % liczba probek sygnalu
dt = 1/fpr; t = dt*(0:Nx-1);    % chwile pobierania probek (probkowania)
%x = zeros(1,Nx); x(1) = 1;     % impuls jednostkowy (delta Kroneckera)
x = sin(2*pi*20*t+pi/3) + sin(2*pi*500*t+pi/7);  % suma dwoch sinusow

% Cyfrowa filtracja FIR: x(n) --> [ b ] --> y(n)
M = length(b);                  % liczba wspolczynnikow b(k) filtra FIR
bx = zeros(1,M);                % bufor na probki wejsciowe sygnalu x(n)
for n = 1 : Nx                  % PETLA GLOWNA
    bx = [ x(n) bx(1:M-1) ];    % umiesc kolejna probke x(n) w buforze bx
    y(n) = sum( bx .* b );      % wykonaj filtacje (srednia wazona), oblicz y(n)
end                             %
%y = filter(b,1,x);             % to samo z uzyciem funkcji filter()
%y = conv(x,b); y=y(1:NX);      % to samo z uzyciem funkcji conv()

% RYSUNKI: porownanie sygnalu wejsciowego (WE) i wyjsciowego (WY)
figure;
subplot(211); plot(t,x); grid;             % sygnal wejsciowy x(n)
subplot(212); plot(t,y); grid; pause       % sygnal wyjsciowy y(n)
figure; % widma DFT drugich polowek obu sygnalow (usuniecie stanu przejsciowego z WY)
k=Nx/2+1:Nx;  f0 = fpr/(Nx/2);  f=f0*(0:Nx/2-1);
subplot(211); plot(f,20*log10(abs(2*fft(x(k)))/(Nx/2))); grid;
subplot(212); plot(f,20*log10(abs(2*fft(y(k)))/(Nx/2))); grid; pause
figure; % synchronizacja sygnalow WE i WY
subplot(211); plot(t,x,'r-',t,y,'b-'); grid;               % brak synchronizacji WE i WY
P=(M-1)/2; t=t(P+1:end-P); x=x(P+1:end-P); y=y(2*P+1:end); % kompensacja opoznienia
subplot(212); plot(t,x,'r-',t,y,'b-'); grid; pause         % po synchronizacji WE i WY
```

TEMAT #2: Projektowanie wag `b` fitrów cyfrowych FIR. Obecnie poznamy trzy metody projektowania wag fitrów cyfrowych FIR: 1) metodę okien, 2) metodę odwrotnego DFT, oraz 3) metodę optymalizacji średniokwadratowej.

Problem 9.2 (Metoda okien).** Opisana w [40] w podrozdziale 12.5.

Ponieważ w przypadku filtrów FIR wagi b_k filtra są równe jego odpowiedzi impulsowej $h(k)$, to można je obliczyć analitycznie, stosując odwrotną transformację DtFT (patrz laboratorium 4 DFT&DtFT, temat 4) z założonej ch-ki amplitudowo-częstotliwościowej $H_{XX}(f)$ (zakładamy: $M = 2P+1$):

$$h(k) = \frac{1}{f_{pr}} \int_{-f_{pr}/2}^{f_{pr}/2} H_{XX}(f) e^{j2\pi \frac{f}{f_{pr}} k} df, \quad -P \le k \le P. \tag{9.4}$$

Przykładowo, dla filtra LP, przepuszczającego częstotliwości do f_0 mamy: $H(f \le f_0) = 1$, dla pozostałych częstotliwości 0, dlatego otrzymujemy (w zapisie równania (9.4 dla częstotliwości radialnej $\Omega = 2\pi \frac{f}{f_{pr}}$):

$$h_{LP}^{(f_0)}(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_{LP}(e^{j\Omega}) e^{j\Omega k} d\Omega = \frac{1}{2\pi} \int_{-\Omega_0}^{\Omega_0} 1 \cdot e^{j\Omega k} d\Omega = \frac{1}{2\pi} \frac{1}{jk} e^{j\Omega k} \Big|_{-\Omega_0}^{\Omega_0} =$$

$$= \frac{1}{j2\pi k} \left[e^{j\Omega_0 k} - e^{-j\Omega_0 k} \right] = \frac{2j\sin(\Omega_0 k)}{j2\pi k} = \frac{2j\sin(\Omega_0 k)}{j2\pi k} = \frac{\sin\left(2\pi \frac{f_0}{f_{pr}} k\right)}{\pi k} = 2\frac{f_0}{f_{pr}} \frac{\sin(\Omega_0 k)}{\Omega_0 k} \tag{9.5}$$

Z twierdzenia de l'Hospitala (pochodna licznika i mianownika ostatniego wyrażenia względem k) dla $k = 0$ otrzymujemy : $h_{LP}^{(f_0)}(0) = 2\frac{f_0}{f_{pr}}$. Wagi innych rodzajów filtrów otrzymuje się w następujący sposób:

1. HP $[f_0, \frac{f_{pr}}{2}]$: odejmując wagi filtra LP (przepuszczającego do f_0) od wag filtra wszechprzepustowego $[0, \frac{f_s}{2}]$ ($h_{ALL}(0) = 1$, pozostałe równe 0),
2. BP $[f_1, f_2]$: odejmując od siebie wagi dwóch filtrów LP, mających pasma przepustowe o różnych szerokościach ($f_1 < f_2$): wagi węższego filtra (do f_1) od szerszego (do f_2),
3. BS $[0, f_1] + [f_2, \frac{f_s}{2}]$: odejmując wagi filtra BP $[f_1, f_2]$ od wag filtra wszechprzepustowego $[0, \frac{f_s}{2}]$ ($h_{ALL}(0) = 1$, pozostałe równe 0).

W równaniu (9.5) podstawiamy $k = -P, ..., 0, ..., P$ i obliczamy odpowiedź impulsową filtra nieprzyczynowego (filtr przyczynowy powien mieć $h(k < 0) = 0$). Aby uzyskać filtr przyczynowy, przesuwamy obliczone wagi o P próbek w prawo (wprowadzamy P-próbkowe opóźnienie), co powoduje pomnożenie $H(f)$ filtra przez $e^{-j2\pi \frac{f}{f_{pr}} P}$, czyli liniowe malenie wynikowej ch-ki fazowej.
Im większa jest wartość P, tym filtr ma bardziej strome zbocze w dziedzinie częstotliwości. Jednak niezależnie od wartości P zawsze "przycinamy" na końcach $h(k)$, czyli stosujemy "okno" prostokatne. Iloczyn w dziedzinie czasu, odpowiada splotowi w dziedzinie częstotliwości. Dlatego idealna zero-jedynkowa odpowiedź częstotliwościowa filtra "teoretycznego" splata się z oscylacyjnym widmem częstotliwościowym okna prostokątnego $\left(\frac{sin(\omega)}{\omega}\right)$ i zostaje przez to popsuta: staje się oscylacyjna w paśmie przepustowym (wokół wzmocnienia 1, utrata liniowości) oraz w paśmie zamorowym (wokół 0, słabe tłumienie). Z tego powodu obliczne wagi $h(k)$ należy wymnożyć z wybraną funkcją *okna*, np. Kaisera lub Czebyszewa (patrz laboratorium 4 DFT&DtFT).

Pociąg rusza ... Samolot startuje ...
Przeanalizuj kod Matlaba z listingu 9.2. Sprawdź czy równania na wagi filtra, wyprowadzone powyżej, zostały w programie poprawnie zaimplementowane. Dodaj kod programu z listingu 9.2 na początek programu 9.1. Zastosuj wagi filtra LP, HP oraz BP do modyfikcji sygnału wejściowego `x`. Zauważ, że tylko składowa sygnału o częstotliwości 20 Hz (LP), 500 Hz (HP) oraz, ponownie, 500 Hz (BP) jest przepuszczana przez poszczególne filtry na ich wyjście. Oblicz współczynniki wagowe filtra pasmowo-zaporowego BS, których brakuje w programie, oraz zastosuj je do filtracji sygnału: teraz tylko składowa 20 Hz powinna pojawić się na wyjściu filtra. Zmień poziom listków

bocznych widma okna Czebyszewa (Rs=60, 80, 100, 120 dB) oraz narysuj na jednym rysunku odpowiedzi amplitudowo-częstotliwościowe filtrów, zaprojektowanych z użyciem tych okien. Następnie ustaw Rs=100 dB i wybierz różną długość filtra, zmieniając wartość parametru M: M=20,40,100,200. Ponownie narysuj na jednym rysunku wszystkie charakterystyki amplitudowo-częstotliwościowe filtrów, uzyskanym w ten sposób. Tłumienie filtra w paśmie zaporowym zależy od wyboru funkcji okna (sprawdź to stosując dodatkowo okna hanning(), blackman() oraz kaiser()). Natomiast szerokość pasma przejściowego filtra, czyli szybkość przejścia od fazy przepuszczania do tłumienia, zależy od długości filtra (liczby jego wag): dłuższe filtry mają ch-kę amplitudowo-częstotliwościową bardziej stromą.

Listing 9.2: Projektowanie filtrów cyfrowych FIR metodą "okien"

```
% cps_09_fir_okna.m
clear all; close all;

fpr = 2000;      % czestotliwosc probkowania
f0 = 100;        % czestotliwosc graniczna dla filtrow low-pass oraz high-pass
f1 = 400;        % czestotliwosc graniczna dolna dla filtrow band-pass oraz band-stop
f2 = 600;        % czestotliwosc graniczna gorna dla filtrow band-pass oraz band-stop
M = 100;         % polowa dlugosci filtra (N=2M+1)
N = 2*M + 1;     % dlugosc filtra b(n) - u nas zawsze nieparzysta
n = -M :1: M;    % indeksy wag filtra, filtr nieprzyczynowy: b(n) rozne od 0 dla n<0

% Odpowiedzi impulsowe filtrow FIR - ich wagi
hALL = zeros(1,N); hALL(M+1)=1;                        % AllPass
hLP = sin(2*pi*f0/fpr*n)./(pi*n); hLP(M+1) = 2*f0/fpr;    % LowPass f0
hHP = hALL - hLP;                                      % HighPass
hLP1 = sin(2*pi*f1/fpr*n)./(pi*n); hLP1(M+1) = 2*f1/fpr; % LowPass f1
hLP2 = sin(2*pi*f2/fpr*n)./(pi*n); hLP2(M+1) = 2*f2/fpr; % LowPass f2
hBP = hLP2 - hLP1;                                     % BandPass [f1,f2]
%hBS = ?;                                              % BandStop [f1,f2]
b = hLP;                                               % wybierz: hLP, hHP, hBP, hBS
stem( n, b ); title('b(n)'); grid; pause
b = b .* chebwin(N,100)';                              % okno Czebyszewa o tlumieniu -100 dB
stem( n, b ); title('b(n)'); grid; pause
```

Problem 9.3 (Metoda odwrotnego DFT).** Opisana w [40] w podrozdziale 12.2.

Zamiast obliczać wagi filtra $h(k) = b_k$ w sposób analityczny za pomocą równania (9.4), można je wyznaczyć "numerycznie" (komputerowo). W takim przypadku należy:

1. stworzyć $N = M+1 = (2P+2)$-elementowy wektor $\mathbf{H}$, zawierający kolejne, wymagane wartości ch-ki amplitudowo-częstotliwościowej filtra, dbając o zachowanie jej wymaganej symetrii hermitowskiej:

$$H(f_k),\ f_k = (k-1)\cdot\frac{f_{pr}}{N},\ k = 1,\ldots,N, \tag{9.6}$$

$$H\left(\frac{N}{2}+1+k\right) = H^*\left(\frac{N}{2}+1-k\right), \quad k = 1\ldots\frac{N}{2}-1, \tag{9.7}$$

$$\mathrm{Im}(H(1)) = 0, \qquad \mathrm{Im}\left(H\left(\frac{N}{2}+1\right)\right) = 0. \tag{9.8}$$

2. wykonać na nim odwrotną transformację Fouriera: $\mathbf{h} = \mathrm{IDFT}(\mathbf{H})$;
3. uporządkować "czasowo" otrzymane wagi filtra: $\mathbf{h} = [\,h(P+1:-1:2),\ h(1:P+1)\,]$.
4. wywnożyć je z funkcją *okna*: $h(k) = h(k)w(k),\ \ k = 1,2,\ldots,2P+1$.

Przeanalizuj kod Matlaba z listingu 9.3. Wagi filtra są w nim wyliczane za pomocą wykonania odwrotnej dyskretnej transformacji Fouriera DFT (FFT) na zadanej charakterystyce amplitudowo-częstotliwościowej filtra, określonej przez użytkownika. Zadane widmo DFT szukanych wag filtra odznacza się sprzężoną symetrią. Dodaj kod 9.3 na początek programu 9.1 oraz sprawdź jaki kształt mają odpowiedzi częstotliwościowe filtrów LP, HP i BP, zaprojektowanych tą metodą. Zaproponuj projekt filtra pasmowo-zaporowego BS. Użyj wszystkich filtrów do filtracji sygnału testowego. Włącz/wyłącz użycia okna Czebyszewa do modyfikacji kształu sygnału, otrzymanego w wyniku odwrotnego DFT (FFT). Użycie okna powinno umożliwić nam poprawę tłumienia filtra dla częstotliwości leżących w paśmie zaporowym, za cenę pogorszenia stromości filtra, tzn. poszerzenia pasma częstotliwościowego prześcia od fazy przepuszczania do nieprzepuszczania. Stromość filtra można jednak zawsze poprawić zwiększając jego długość. Narysuj na jednym rysunku ch-ki amplitudowo-częstotliwościowe (odpowiedzi amplitudowe) kilku filtrów zaprojektowanych dla: 1) różnych okien (`hanning(), blackman(), kaiser()`, oraz 2) jednego wybranego okna, ale dla filtrów o różnej długosci, np. `M=50, 100, 150, 200`. Spróbuj umieścić więcej niż jeden punkt (obecnie mamy tylko 0.5) w paśmie przejściowym filtra ($1 \rightarrow 0$ lub $0 \rightarrow 1$).

Listing 9.3: Projektowanie filtra cyfrowego FIR metodą odwrotnej DFT

```
% cps_09_fir_idft.m
clear all; close all;

% Parametry
fpr = 2000;                                % czestotliwosc probkowania (Hz)
f0=100; f1=400; f2=600;                    % czestotliwosci graniczne
M = 100;                                   % polowa dlugosci filtra
K = 4;                                     % nadprobkowanie w dziedzinie czestotliwosci
N = 2*M + 1;                               % dlugosc filtra b(n) - zawsze nieparzysta
n = -M :1: M;                              % indeksy wag filtra
NK = N*K; if(rem(NK,2)==1) NK=NK+1; end    % po nadprobkowaniu w czestotliwosci
f = fpr/NK; f = df*(0 : NK/2);             % czestotliwosci
H0 = zeros(1,NK/2+1);                      % inicjalizacja
H1 = ones(1,NK/2+1);                       % inicjalizacja
% Low-Pass - dolnoprzepustowy
ind = find( f<=f0 );
HLP = H0; HLP(ind) = ones(1,length(ind)); HLP(ind(end))=0.5;
% High-Pass - gornoprzepustowy
ind = find( f>=f0 );
HHP = H0; HHP(ind) = ones(1,length(ind)); HLP(ind(1))=0.5;
% Band-Pass oraz Band-Stop - pasmo-woprzepustowy i pasmowo-zaporowy
ind1 = find( f< f1 );
ind2 = find( f<=f2 );
ind  = find( ind2 > ind1(end) );
HBP = H0; HBP(ind) = ones(1,length(ind));  HBP(ind(1))=0.5; HBP(ind(end))=0.5;
% HBS = ?  % to jest Twoje zadanie

% Nasz wybor: LP, HP, BP, BS
H = HBS;                                         % wybor: LP, HP, BP, BS
H(NK:-1:NK/2+2) = H(2:NK/2);                     % odbicie symetryczne

h = ifft( H );                                   % odwrotne DFT
b = [ h(M+1:-1:2) h(1:M+1) ];                    % wybor 2M+1 srodkowych wag
%b = b .* chebwin(N,100)';                       % opcjonalne okienkowanie wag
figure; stem(n,b); title('b(n)'); grid; pause    % rysunek wag
```

Problem 9.4 (Metoda optymalizacji średniokwadratowej LS (Least-Squares)).** Opisana w [40] w podrozdziale 12.3. Przeanalizuj kod Matlaba z listingu 9.4, implementujący projektowanie filtra cyfrowego FIR metodą optymalizacji średniokwadratowej LS (minimalizacja błądu LS pomiędzy zadaną, a otrzymaną odpowiedzią częstotliwościową filtra). Znajdź w programie linie, w których zaimplementowane są wszystkie końcowe równania matematyczne metody, wyprowadzone w [40]: w obecnej formie program umożliwia jedynie projektowanie filtra FIR typu lowpass LP. Dodaj kod 9.4 na początek programu 9.1. Uruchom program. Zapoznaj się z charakterystyką amplitudowo-częstotliwościową zaprojektowanego filtra oraz z wynikiem filtracji sygnału z jego użyciem: tylko składowa 20 Hz powinna zostać przepuszczona przez nasz filtr. Zmień wartości wag `wp,wt,ws` (ważność pasm: pass, transient, stop, im większa waga tym jest to dla nas istotniejsze): jak one wpływają na kształt odpowiedzi częstotliwościowej zaprojektowanego filtra? Narysuj na jednym rysunku odpowiedzi częstotliwościowe zaprojektowanych filtrów LP, różniących się długością: `M=50,100,150,200`: dłuższy filtr powinien oferować bardziej stromą charakterystykę częstotliwościową pass-to-stop. Uzupełnij program: dodaj do niego możliwość projektowania filtrów high-pass, band-pass oraz band-stop (za każdy z nich dostaniesz dodatkowy punkt!). Włącz i wyłącz użycie okna Czebyszewa do wygładzenia "brzegów" zaprojektowanych odpowiedzi impulsowych filtrów - co uzyskujemy dzięki "okienkowaniu" wag filtra?

Listing 9.4: Projektowanie filtra cyfrowego FIR metodą optymalizacji średniokwadratowej jego odpowiedzi częstotliwościowej

```
% cps_09_fir_ls.m
clear all; close all;

% Parametry
fpr = 2000;                   % czestotliwosc probkowania (Hz)
f0 = 100;                     % czestotliwosc graniczna
M = 100;                      % polowa dlugosci filtra, N=2M+1
K = 4;                        % nadprobkowanie w dziedzinie czestotliwosci
% P punktow zadanej ch-ki amplitudowej Ad() dla czestotliwosci katowych 2*pi*p/P, p=0...P-1
P = K*2*M;                    % liczba punktow ch-ki amplitudowej (parzysta; P >= N=2M+1)
L1 = floor(f0/fpr*P),         % liczba pierwszych punktow o wzmocnieniu 1
Ad = [ ones(1,L1) 0.5 zeros(1,P-(2*L1-1)-2) 0.5 ones(1,L1-1)];
Ad = Ad';
% Wybor wspolczynnikow wagowych w(p) optymalizacji, p=0...P-1, dla pasm Pass/Transit/Stop
wp = 1;                      %  wagi dla PassBand
wt = 1;                      %  wagi dla TransientBand
ws = 10000;                  %  wagi dla StopBand
w = [ wp*ones(1,L1) wt ws*ones(1,P-(2*L1-1)-2) wt wp*ones(1,L1-1) ];
W = zeros(P,P);              %
for p=1:P                    %  macierz z wagami optymalizacji na glownej przekatnej
    W(p,p)=w(p);             %
end                          %
% Znajdz macierz F, bedaca rozwiazaniem rownania macierzowego  W*F*h = W*(Ad + err)
F = [];
n = 0 : M-1;
for p = 0 : P-1
    F = [ F; 2*cos(2*pi*(M-n)*p/P) 1 ];
end
% Znajdz wagi h(n), minimalizujace blad LS sum( (W*F*h - W*Ad).^2 )
% h = pinv(W*F)*(W*Ad);      % metoda #1
h = (W*F)\(W*Ad);            % metoda #2
b = [ h; h(M:-1:1) ]';       % odbicie symetryczne
%b = b .* chebwin(N,100)';   % opcjonalne zastosowanie dowolnej funkcji okna
figure; stem(-M:M,b); title('b(n)'); grid; pause   % rysunek
```

Problem 9.5 (Metoda min-max aproksymacji Czebyszewa (algorytm Remeza)).** Opisana w [40] w podrozdziale 12.4. Zapewnia minimalny błąd maksymalny aproksymacji zadanej ch-ki częstotliwościowej, czyli stały poziom oscylacji w paśmie zaporowym, ale także występowanie osylacji w paśmie przepustowym. Do dalszego samodzielnego studiowania. Przeczytaj opis metody w [40]. Ale zastosuj funkcję Matlaba `firpm()`. Do zaliczenia jest wymagane rozumienie metody i działający program.

TEMAT #3: Zastosowania cyfrowych filtrów FIR. Separacja poszczególnych składowych sygnału. Odszumianie sygnału.

Problem 9.6 (* Filtracja sumy różnych dźwięków: separacja poszczególnych składowych).** Znajdź różne sygnały dźwiękowe w Internecie, np. pobierz nagrania ze stronyWWW *FindSounds* - postaraj się, aby miały one tę samą częstotliwość próbkowania. Utwórz różne sygnały sumaryczne: mowa + wysoko-częstotliwościowy warkor silnika, mowa + śpiew ptaka, wycie wilk (ryk lwa, trąbienie słonia) + wysoko-częstotliwościowy śpiew ptaka, etc. Oblicz i wyświetl widmo FFT (`fft()`) oraz spektrogram STFT (`spectrogram()`) dla każdego sygnału z osobna oraz sygnału sumy. Zaprojektuj filtr cyfrowy FIR, próbujący przepuścić tylko jedną składową sumy na wyjście filtra, np. mowę. Oblicz i pokaż ch-kę amplitudową filtra w decybelach. Narysuj jakie jest położenie zer transmitancji filtra na płaszczyźnie zespolonej zmiennej z (ogranicz rysunek do zakresu [-5,5] w osi rzeczywistej i urojonej — funkcja texttaxis(xmin,xmax,ymin,ymax)). Dokonaj filtracji sygnału sumy. Porównaj wynik filtracji z sygnałem sumy oraz z sygnałem oryginalnym (przed zsumowaniem). Odsłuchaj każdy z czterech sygnałów: oryginalny, zakłócenie, ich sumę, wynik odtworzenia oryginału z sumy. Oblicz i pokaż FFT oraz spektrogram (STFT) tych trzech sygnałów.

Problem 9.7 (Usunięcie sygnału zakłócenia (zmodulowanego w częstotliwości) od dźwięków rzeczywistych).** Nagraj lub pobierz z Internetu sygnał Twoich marzeń. Dodaj do niego sygnał z sinusoidalną modulacją częstotliwości (SFM) (przypomnij sobie jak to robiliśmy podczas laboratorium 2). Zaprojektuj pasmowo-zaporowy (BS) filtr cyfrowy FIR do usunięcia zakłócenia (o odpowiedniej charakterystyce amplitudowo-częstotliwościowej). Dokonaj filtracji sygnału. Oblicz widmo częstotliwościowe (`fft()`, `pwelch()`, `pspectrogram()`) sygnału wejściowego, zakłócenia oraz wyniku filtracji. Odsłuchaj każdy sygnał oddzielnie. Oblicz wartość stosunku sygnału do szumu SNR (signal-to-noise ratio), patrz wykład i laboratorium numer 2, dla sygnału przed filtrem i po filtrze. Aby to zrobić poprawnie, musisz uwzględnić (skompensować) opóźnienie wprowadzane przez filtr (o P próbek dla filtra o długości $2P+1$). Wytłumaczenie jak to zrobić jest podane w zadaniu dotyczącym odszumiania sygnału EKG - patrz poniżej. Powtórz eksperyment dla różnych parametrów sygnału zakłócającego (mała i duża amplituda, różna częstotliwość środkowa i głebokość modulacji).

Problem 9.8 (Odszumianie sygnałów rzeczywistych).** Nagraj lub pobierz z Internetu sygnał dźwiękowy Twoich marzeń. Oblicz i wyświetl jego widmo częstotliwościowe (`fft()`, `pwelch()`). Zaobserwuj, w jakim paśmie częstotliwościowym jest skoncentrowana energia sygnału. Dodaj do sygnału szum gaussowski o różnej sile (wariancji, odchyleniu standardowym). Zaprojektuj filtr cyfrowy FIR pasmowo-przepustowy BP, przepuszczający tylko główne składowe częstotliwościowe naszego sygnału (oczywiście z szumem, który się do nich dodał). Porównaj charakterystykę amplitudowo-częstotliwościową zaprojektowanego filtra z widmem oryginalnego sygnału, najlepiej nakładając je na siebie na jednym rysunku. Następnie dokonaj filtracji zaszumionego sygnału. Narysuj na jednym rysunku: 1) oryginalny, niezaszumiony sygnał, 2) sygnał z dodanym szumem, 3) wynik filtracji. Aby miało to sens, musisz usunąć (skompensować) opóźnienie wprowadzane przez filtr. Jak to zrobić - przeczytaj opis w ostatnim zadaniu, dotyczącym odszumiania sgnału EKG). Po kompensacji opóźnienia, oblicz współczynnik sygnału do szumu SNR (zdefiniowany w wykładzie/laboratorium numer 2: stosunek energii sygnału oryginalnego do energii szumu, wyrażony w decybelach) dla sygnału przed i po filtrze. Porównaj widma częstotliwościowe sygnałów: oryginalnego, zaszumionego oraz odszumionego. Odsłuchaj każdy z sygnałów z osobna.

Problem 9.9 (* Odszumianie sygnału EKG).** Wczytaj do Matlaba sygnał EKG (używany podczas laboratorium numer 1) :

```
clear all; load ECG100.mat; whos
```

albo pobierz z Internetu dowolny sygnał elektrokardiograficzny EKG (sygnał elektryczny aktywności serca), np. ze strony https://www. physionet.org/cgi-bin/atm/ATM. W przypadku zapisu `ECG100.mat` przyjmij $f_{pr} = 360\ Hz$. Oblicz i wyświetl widmo częstotliwościowe sygnału (skorzystaj z funkcji `fft()` lub `pwelch()`) : znajdź w jakich częstotliwościach jest skoncentrowana energia sygnału. Zaprojektuj filtr cyfrowy FIR, przepuszczający tylko podstawowe składowe częstotliwościowe sygnału EKG oraz usuwający pozostałe, związane z szumem. Dokonaj filtracji sygnału: na jednym rysunku narysuj sygnał wejściowy `x(n)` oraz wyjściowy `y(n)`. Skompensuj `P`-próbkowe opóźnienie wprowadzane przez filtr. Przypomnij sobie: jeśli filtr ma długość $M = 2P+1$ współczynników/wag, to próbka wejściowa `x(P+1)` odpowiada próbce wyjściowej `y(2*P+1)`, i tak dalej, czyli próbki `x=x(P+1:P+1+Nsamples-1)` odpowiadają próbkom `y=y(2*P+1:2*P+1+Nsamples-1)`. Jeśli poziom szumu w analizowanym sygnale EKG jest mały, dodaj do niego szum gaussowski, sztucznie wygenerowany: `x=x+0.5*randn(1,length(x))` (wektor próbek szumu musi mieć taką samą orientację, poziomą lub pionową, jak wektor próbek sygnału EKG). Dokonaj filtracji zaszumionego sygnału EKG. Ponownie narysuj na jednym rysunku oryginalny sygnał wejściowy ("czyste" EKG) oraz sygnał wyjścowy z filtra (wynik odszumiania "brudnego" sygnał EKG). Oba sygnały powinny się w przybliżeniu pokrywać — mniej lub bardziej, w zależności od poziomu szumu. Zastosuj filtry o różnej szerokości pasma przepustowego, różnym tłumieniu w paśmie zaporowym, różnej długości. Powtórz eksperyment dla szumu o małej i dużej amplitudzie (odchyleniu standardowym - zmień współczynnik skalujący szum równy 0.5 na inny).

Laboratorium 10
Filtry FIR do zmiany częstotliwości próbkowania

Streszczenie Podczas tego laboratorium nauczymy się projektować nierekursywne filtry cyfrowe o skończonej odpowiedzi impulsowej (FIR - *Finite Impulse Response*), przeznaczone do układów zmiany częstotliwosci próbkowania: zwiększenia/nad-prókowania (interpolacji) oraz zmniejszenia/pod-próbkowania (decymacji). Filtry te są stosowane w układach cyfrowych, pracujących z sygnałami o różnej częstotliwości próbkowania, np. w mikserach cyfrowych oraz nadajnikach i odbiornikach telekomunikacyjnych. Poza tradycyjnymi, poznamy także szybkie interpolatory i decymatory sygnałów cyfrowych, zrealizowane w sposób polifazowy.

TEMAT #1: Filtry cyfrowe FIR do K-krotnego zwiększenia częstotliwości próbkowania (interpolacji) sygnału. Patrz rysunek 10.1. Cyfrowy układ interpolatora (*up-samplera*) K-tego rzędu (inaczej $[1:K]$: jedna próbka sygnału jest zastępowana przez K próbek) składa się z:

1. cyfrowego ekspandera K-tego rzędu oznaczanego jako $\uparrow K$: układu wstawiającego $K-1$ wartości zerowych pomiędzy każde dwie próbki sygnału, np. 2 zera dla $K=3$;
2. cyfrowego filtra dolno-przepustowego typu FIR, który przesuwa wstawione zera do krzywej wyznaczonej przez oryginalne próbki sygnału, których wartości nie są zmieniane; filtr ma 3-dB częstotliwość graniczną, równą połowie oryginalnej częstotliwości próbkowania sygnału, równocześnie równej $\frac{1}{K}$-tej połowy nowej częstotliwości próbkowania, która jest K-razy większa (wychodzi na to samo).

Dążąc do uproszczenia złożoności obliczeniowej całej operacji, mnożenie wag filtra przez wstawione wartości zerowe powinno być opuszczone podczas filtracji. Umożliwia to wersja polifazowa zapisu filtracji. Przesuwając wagi filtra nad sygnałem z wstawionymi zerami można zaobserwować, że w średniej ważonej występują za każdym razem, na przemian, kolejne, co K-te wagi filtra i próbki, czyli ich tzw. składowe polifazowe. Np. dla $K=4$, co 4-te, zaczynając:

- od 0 (0,4,8,12,...), od 1 (1,5,9,13,...) , od 2 (2,6,10,14,...), od 3 (3,7,11,15,...);
- od 4 (4,8,12,16,...), od 5 (5,9,13,17,...), od 6 (6,10,14,18,...), od 7 (7,11,15,19,...);
- od 8 (8,12,16,20,...), od 9 (9,13,17,21,...), od 10 (10,14,18,22,...), od 11 (11,15,19,23,...);
- od 12 (12,16,20,24,...), od 13 (13,17,21,25,...), od 14 (14,18,22,26,...), od 15 (15,19,23,27,...);
- ..... itd ......

Z tego powodu można wykonać osobno filtrację składowych polifazowych wag filtra i próbek sygnału, odpowiadających sobie numerami, a potem złożyć na przemian z przeplotem próbki wynikowe, tzn. po kolei wziąć wszystkie próbki pierwsze, potem drugie, potem trzecie, itd., z każdego oddzielnego wyniku filtracji.
Szybka polifazowa interpolacja (nad-próbkowanie) jest wytłumaczona na rysunku 10.2.

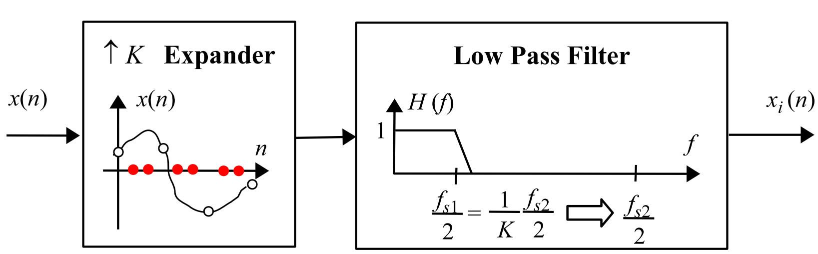

Fig. 10.1: Schemat blokowy interpolatora, który zwiększa K-razy częstotliwość próbkowania (liczbę próbek) sygnału. Na początku, $K-1$ wartości zerowych jest wstawianych przez tzw. ekspander pomiędzy każde dwie próbki sygnału, a potem sygnał z zerami jest wygładzany filtrem dolno-przepustowym, nastrojonym na orginalne pasmo częstotliwościowe sygnału (graniczna częstotliwość odcięcia jest równa połowie początkowej częstotliwości próbkowania sygnału $f_{s1}/2$, czyli $\frac{1}{K}$-tej nowej częstotliwości próbkowania $f_{s2} = \frac{K f_{s1}}{2}$).

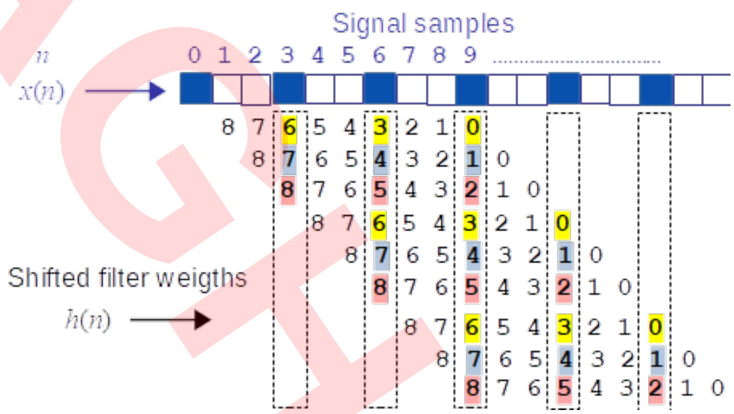

Fig. 10.2: Splotowa interpretacja problemu filtracji, występującego podczas interpolacji sygnału - mnożenie wag filtra przez zera wstawione do sygnału (zaznaczone białymi kwadratami) powinno zostać usunięte. Dzieje się tak w filtracji polifazowej: wykonywane są tylko mnożenia z wagami filtra, zaznaczonymi kolorowym tłem (leżące wewnątrz prostokątów narysowanych liniami przerywanymi). Wagi różnych składowych polifazowych filtra są zaznaczonone różnymi kolorami.

Listing 10.1: Program Matlaba do K-krotnego zwiększenia częstotliwości próbkowania (interpolacji) sygnału

```
% cps10_resample_up.m - nad-probkowanie (interpolacja) sygnalu
clear all; close all;

% Wejscie - parametry i sygnal wejsciowy x
K=5; M=50; N=2*M+1; Nx=1000;            % K - rzad nad-probkowania
x = sin(2*pi*(0:Nx-1)/100);             % sygnal do nad-probkowania
% [x,fs]=audioread('mowa.wav'); x=x(:,1)'; Nx = length(x);  % do dalszych testow
R=rem(Nx,K); x = x(1:end-R); Nx = Nx-R;   % korekta dlugosci dla filtracji polifazowej

% Wolne nad-probkowanie (interpolacja)
% splot wag filtra z sygnalem uzupelnionym zerami
xz = zeros(1,K*Nx);                     % # wstawienie zer pomiedzy
xz(1:K:end) = x;                        % # probki sygnalu
h = K*fir1(N-1, 1/K, kaiser(N,12));     % projekt filtra interpolujacego
yi = filter(h,1,xz);                    % filtracja wygladzajaca (usuwajaca zera)

figure; freqz(x,1,1000,'whole');        % spektrum DFT signalu x(n)
figure; freqz(xz,1,1000,'whole');       % spektrum DFT signalu xz(n)
figure; freqz(h,1,1000,'whole');        % spektrum DFT filtra h(n)
figure; freqz(yi,1,1000,'whole');       % spektrum DFT signal yi(n)

n = M+1:K:K*Nx-M; ni = N:K:K*Nx-(K-1);
figure; plot(n,x(M/K+1:Nx-M/K),'ro-',ni-M,yi(ni),'bx'); title('x(n) and yi(n)');
err1 = max(abs(x(M/K+1:Nx-M/K)-yi(ni(1:K:end)))), pause

% Szybkie nad-probkowanie (interpolacja)
% K splotow sygnalu oryginalnego z K skladowymi polifazowymi wag filtra (co K-ta waga zaczynajac od 1,2,...,K-1)
% sygnal nie jest uzupelniony zerami
for k=1:K
    yipp(k:K:K*Nx) = filter( h(k:K:end), 1, x);
end
err2 = max(abs(yi-yipp)), pause
```

Problem 10.1 (Filtry cyfrowe FIR do zwiększenia częstotliwości próbkowania (nad-próbkowania)).** Przeanalizuj kod programu z listingu 10.1. Zauważ, że filtr interpolujący FIR jest filtrem dolno-przepustowym, K-pasmowym $\left(f_{3dB} = \frac{1}{K}\frac{f_{pr}}{2}\right)$ o wzmocnieniu równym K. Wzmocnienie K kompensuje $\frac{1}{K}$-krotne zmniejszenie energii sygnału, spowodowane wstawieniem $K-1$ wartości zerowych pomiędzy oryginalne próbki sygnału. Zauważ, że oryginalne próbki sygnału nie zmieniają swoich wartości podczas filtracji: obliczane są tylko wartości nowych próbek leżących pomiędzy próbkami oryginalnymi. Efekt ten jest uzyskiwany dzięki temu, że każda co K-ta waga filtra interpolacyjnego, poza środkową, jest równa zero. Narysuj wagi filtra by `stem(h)` i zaobserwuj to "zjawisko". Zauważ, że wartości zerowe nie są wstawiane pomiędzy oryginalne próbki sygnału podczas szybkiej, polifazowej implementacji operacji nad-próbkowania. Zamiast tego sygnał oryginalny jest kilkakrotnie oddzielnie filtrowany przez wszystkie składowe polifazowe (PP) wag filtra (`h(k:K:end),k=1...K-1,`), po czym poszczególne wyniki filtracji są łączone w jeden sygnał w sposób "grzebieniowy". Zwróć uwagę, że szybka implementacja PP operacji nad-próbkowania daje ten sam wyniki co operacja wolna z mnożeniem przez zera (błąd `err2` jest bardzo, bardzo mały ~10^{-16}). Przetestuj operację zwiększania częstotliwości próbkowania dla różnych sygnałów wejściowych. Użyj filtrów o różnej długości oraz użyj różnych funkcji okien podczas operacji projektowania wag filtra metodą okien: oblicz i pokaż odpowiedź częstotliwościową filtra, zwróć uwagę na błąd `err1`. Zmień wartość K na inną. Dodaj do programu funkcje Matlaba `y=interp(x,K)` oraz `y=resample(x,K,1)`, porównaj ich wyjścia z naszym wynikiem nad-próbkowania.

Problem 10.2 (Nad-próbkowanie sygnału mowy).** Nagraj lub znajdź w Internecie sygnał mowy, spróbkowany z częstotliwością 8000 Hz. Zmodyfikuj program 10.1 i użyj go do nad-próbkowania sygnału mowy do 48000 próbek na sekundę (sps - *samples per second*). Posłuchaj sygnału oryginalnego (`soundsc(x,8000)`) oraz nad-próbkowanego (`soundsc(xup,48000)`). Do muzyki spróbkowanej z częstotliwością 48000 Hz, dodaj oddzielnie: 1) oryginalny sygnał mowy ($f_{pr} = 8000$ Hz), 2) nadpróbkowany sygnał mowy ($f_{pr} = 48000$ Hz). Odsłuchaj oba otrzymane sygnały, będące wynikiem cyfrowego miksowania dźwięków. Który z nich został poprawnie utworzony?

TEMAT #2: Filtry cyfrowe FIR do L-krotnego zmniejszenia częstotliwości próbkowania (decymacji). Patrz rysunek 10.3. W układzie cyfrowego decymatora *(down-samplera)* L-tego rzędu, L kolejnych próbek sygnału jest zastąpionych przez jedną próbkę (co zapisujemy $[L:1]$). Układ ten składa się z:

1. dolno-przepustowego filtra cyfrowego, w którym pasmo zaporowe rozpoczyna się od połowy nowej, docelowej częstotliwości próbkowania, która jest L-razy niższa niż oryginalna (co wynika z twierdzenia o próbkowaniu: L-razy niższa częstotliwość próbkowania wymaga L-krotnie węższego pasma sygnału);
2. cyfrowego L-krotnego reduktora, oznaczanego jako $\downarrow L$, pozostawiającego tylko co L-tą próbkę wejściową.

Ponieważ po filtracji jest pozostawiana co L-ta próbka, nie ma sensu obliczać próbek, które i tak zostaną usunięte. W efektywnej obliczeniowo implementacji polifazowej (PP) decymatora, usuwane próbki nie są w ogóle wyznaczane. Można to osiągnąć przesuwając wagi filtra decymatora od razu o L próbek do przodu, a nie o jedną próbkę. Ale wówczas nie można w sposób natychmastowy zastosować szybkiego algorytmu splotu dwóch wektorów, wykorzystującego FFT (`y=ifft( fft(xz) .* fft(hz) )`. Da się to dopiero zrobić kiedy wagi filtra i próbki sygnału zostaną zapisane w sposób polifazowy, np. dla $L = 3$ i pierwszej próbki wyjściowej mamy:

$$y(1) = x_1h_1 + x_2h_2 + x_3h_3 + x_4h_4 + x_5h_5 + x_6h_6 + x_7h_7 + x_8h_8 + x_9h_9 + \ldots$$
$$y(1) = (x_1h_1 + x_4h_4 + x_7h_7 + \ldots) + (x_2h_2 + x_5h_5 + x_8h_8 + \ldots) + (x_3h_3 + x_6h_6 + x_9h_9 + \ldots).$$

Wówczas możemy dokonać po kolei osobnej filtracji L składowych polifazowych próbek synału i wag filtra, otrzymać L oddzielnych wektorów wynikowych oraz dodać ich odpowiadające sobie elementy, tzn. L 1-wszych, L 2-gich, L 3-cich,... Spróbuj to zilustrować na rysunku.
Szybka polifazowa decymacja (pod-próbkowanie) sygnału jest wytłumaczona na rysunku 10.4.

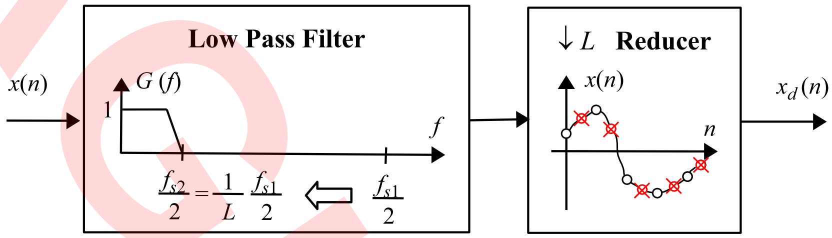

Fig. 10.3: Schemat blokowy decymatora, zmniejszającego L-krotnie częstotliwość próbkowania (liczbę próbek) sygnału: z f_{s1} na $f_{s2} = \frac{f_{s1}}{L}$. Na początku pasmo częstotliwościowe sygnału jest zmniejszane L-krotnie (tzn. dopasowywane do nowej częstotliwości próbkowania, zgodnie z twierdzeniem Nyquist), a potem pozostawiana jest tylko co L-ta próbka. Filtr jest projektowany tak, aby jego pasmo zaporowe rozpoczynało się od częstotliwości f_{s2}.

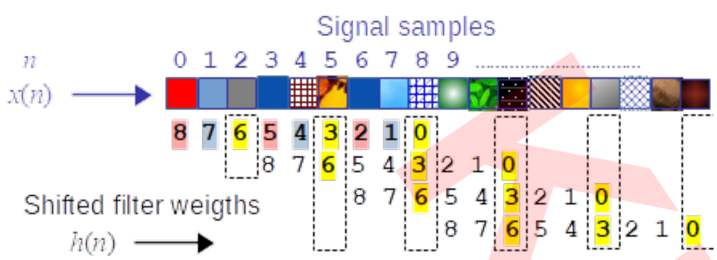

Fig. 10.4: Splotowa interpretacja problemu filtracji, występującego podczas decymacji sygnału - nie są obliczanie wartości próbek, które mają zostać usunięte po filtracji, dzięki przesunięciu wag filtra o $L-1$ próbek, a nie o jedną próbkę. L=3 w naszym przykładzie. Wagi poszczególnych składowych polifazowych filtra są narysowane z użyciem innego koloru tła.

Listing 10.2: Program Matlaba do L-krotnego zmniejszenia częstotliwości próbkowania (decymacji) sygnału

```
% csp_10_resample_down.m - pod-probkowanie (decymacja) sygnalu
clear all; close all;

% Wejscie - parametry oraz sygnal wejsciowy x
L=4; M=50; N=2*M+1; Nx=1000;                   % L - rzad pod-probkowania
x = sin(2*pi*(0:Nx-1)/50); plot(x); pause      % sygnal do pod-probkowania
% [x,fs]=audioread('muzyka.wav'); x=x(:,1)'; Nx = length(x);  % do dalszych testow
R=rem(Nx,L); x = x(1:end-R); Nx = Nx-R;        % korekta dlugosci dla filtracji polifazowej

% Wolne pod-probkowanie (decymacja)
% jeden splot wag filtra z probkami sygnalu, po nim reduktor
g = fir1(N-1, 1/L - 0.1*(1/L),kaiser(N,12));  % projekt filtra decymatora
y = filter(g,1,x);                            % filtracja
yd = y(1:L:end);                              % L-krotny reduktor

n = M+1:Nx-M; nd = (N-1)/L+1:Nx/L;
figure; plot(n,x(n),'ro-',n(1:L:end),yd(nd),'bx'); title('x(n) and yd(n)');
err1 = max(abs(x(n(1:L:end))-yd(nd))), pause

% Szybkie polifazowe pod-probkowanie (decymacja)
% L splotow skladowych PP oryginalnego sygnalu i oryginalnych wag filtra
% probki usuwane nie sa obliczane
x = [ zeros(1,L-1) x(1:end-(L-1)) ];
ydpp = zeros(1,Nx/L);
for k=1:L
    ydpp = ydpp + filter( (g(k:L:end)), 1, x(L-k+1:L:end) );
```

```
end
err2 = max(abs(yd-ydpp)), pause
```

Problem 10.3 (Filtry cyfrowe FIR do zmniejszenia częstotliwości próbkowania (pod-próbkowania)).** Przeanalizuj kod programu 10.2. Zauważ, że cyfrowy filtr FIR stosowany w decymatorze ma wzmocnienie 1 i jest L-pasmowy: ma częstotliwość graniczną równą $f_{3dB} = \frac{1}{L} \cdot \frac{f_{pr}}{2}$, ale **minus pewne przesunięcie**! Dlaczego? Ponieważ dla częstotliwości $\frac{1}{L} \cdot \frac{f_{pr}}{2}$ filtr już się powinen znajdować na początku swojego pasma zaporowego, więc koniec jego pasma przepustowego (f_{3dB}) powinien być "wcześniej", tzn. być czestotliwością niższą. Oblicz i pokaż odpowiedź częstotliwościową filtra, sprawdź czy spełnia on powyższe wymaganie. Zwróć uwagę na rysunku (powiększ go!), że próbki sygnału na wyjściu filtra są nieznacznie różne od próbek oryginalnych: błąd `err1` jest na poziomie $10^{-6} - 10^{-7}$. Zauważ, że w szybkiej implementacji polifazowej nie są obliczane próbki, które zostałyby usunięte przez reduktor $\downarrow L$ na wyjściu decymatora. W tym przypadku odpowiadające sobie składowe polifazowe (PP) sygnału i wag filtra są splatane oddzielnie oraz wyniki wszystkich splotów są do siebie dodawane, tzn. próbki pierwsze, drugie, trzecie, ... Zwróć uwagę, że szybka implementacja PP decymatora (*down-samplera*) daje taki sam wynik jak implementacja wolna (błąd `err2` jest bardzo mały, rzędu ~10^{-15}). Przetestuj procedurę zmniejszenia częstotliwości próbkowania dla różnych sygnałów wejściowych. Użyj filtrów FIR o różnej długości i zastosuj różne okna podczas projektowania filtra FIR metodą okien: oblicz i pokaż ch-kę amplitudowo-częstotliwościową filtra. Jaką wartość ma błąd `err1`? Zmień wartość L. Dodaj do programu funkcje Matlaba `y=decimate(x,L)` oraz `y=resample(x,1,L)`, zastosuj je oraz porównaj ich sygnały wyjściowe z wynikiem naszej decymacji sygnału.

Problem 10.4 (Pod-próbkowanie muzyki).** Znajdź w Internecie sygnał audio spróbkowany z częstotliwością 48000 Hz. Zmodyfikuj program 10.2 oraz zastosuj go do zmniejszenia częstotliwości próbkowania wczytanego nagrania muzycznego do 8000 Hz. Posłuchaj sygnału oryginalnego (`soundsc(x,48000)`) oraz pod-próbkowanego (`soundsc(xdown,8000)`). Do dowolnego sygnału mowy, spróbkowanego z częstotliwością 8000 Hz, dodaj oddzielnie: 1) oryginalny sygnał audio, 2) pod-próbkowany sygnal audio. Odsłuchaj oba sygnały, wynik Twojego miksowania sygnałów cyfrowych: który z nich został poprawnie utworzony?

TEMAT #3: Filtry cyfrowe FIR do zmiany częstotliwości prpbkowania w stosunku $\frac{K}{L}$ (najpierw K-krotne nad-próbkowanie, potem L-krotne pod-próbkowanie). Bardzo często wymagany stosunek zmiany częstotliwości próbkowania sygnału nie jest liczą naturalną (całkowitą). Przykładowo, 1) zmiana częstotliwości z 32 kHz na 48 kHz wymaga 3-krotnego nad-próbkowania, po którym wystepuje 2-krotne pod-próbkowanie, 2) zmiana częstotliwości z 44.1 kHz na 48 kHz wymaga 160-krotnego nad-próbkowania oraz 147-krotnego pod-próbkowania (albo w przybliżeniu {UP=37, DOWN=34} albo {UP=12, DOWN=11}: użyj funkcji Matlaba `[UP,DOWN]=rat(48000/44100,tol);` dla `tol=0.01, 0.001, 0.0001`). Nad-próbkowanie jest zawsze pierwsze, ponieważ podczas pod-próbkowania w sposób nieodwracalny są usuwane z sygnału jego składowe wysoko-częstotliwościowe. Dodatkowo należy pamiętać, że w łańcuchu przetwarzania: interpolator $\rightarrow$ decymator występują po sobie dwa filtry dolno-przepustowe (po ekspandrze $\uparrow K$ w interpolatorze i przed reduktorem $\downarrow L$ w decymatorze): dlatego można zastosować tylko jeden z nich, ten o węższym paśmie częstotliwościowy.

Problem 10.5 (Łączne nad/pod-próbkowanie sygnałów).** Połącz programy 10.1 oraz 10.2 w jeden program, w którym najpierw będzie wykonywane K-krotne nad-próbkowanie, a po nim L-krotne pod-próbkowanie, na przykład 1) (UP) 3-krotne nad-próbkowanie i (DOWN) 2-krotne pod-próbkowanie , oraz 2) (UP) 4-krotne nad-próbkowanie i (DOWN) 5-krotne pod-próbkowanie. Zredukuj liczbę filtrów LP do jednego filtra: tego który ma węższe pasmo częstotliwościowe. Porównaj sygnały, otrzymane po przepróbkowaniu z użyciem tylko jednego oraz dwóch filtrów. Wykonaj operację przepróbkowania z użyciem funkcji Matlaba `y=resample(x,UP,DOWN)` - porównaj jej sygnał wyjściowy z obliczonymi przez ciebie. Zastosuj program do zmiany częstotliwości próbkowania z 48 kHz na 32 kHz. Odsłuchaj sygnał oryginalny (`sound(x,48000)`) oraz pod-próbkowany (`sound(xdown,32000)`.

Problem 10.6 (Zamiana nagrania muzycznego CD (Compact Disc) na nagranie DAB (Digital Audio Broadcasting)).** Spróbuj przekonwertować nagranie muzyki z płyty CD (próbkowanej zawsze z częstotliwością 44100 Hz) na nagranie radia cyfrowego DAB/DAB+ (próbkowanego zawsze z częstotliwością 48000 Hz). Zastosuj funkcję Matlaba `y=resample(x,160,147)` oraz zrób to własno-ręcznie, krok po kroku: najpierw nad-próbkuj sygnał 160 razy (ekspander + filtr LP), potem pod-próbkuj sygnał 147 razy (filtr LP + reduktor). Użyj dwóch filtrów lub tylko jednego, tego o węższym paśmie częstotliwościowym. Odsłuchaj sygnał oryginalny i przepróbkowany.

Problem 10.7 (* Kaskada prostych re-samplerów: jeszcze raz konwersja sygnału CD na sygnał DAB).** Kiedy rząd układu nad- i pod-próbkowania jest bardzo duży, pasma filtrów układów interpolatora i decymatora są bardzo wąskie, odpowiednio: $\frac{1}{K}$ oraz $\frac{1}{L}$ maksymalnej częstotliwości próbkowania. W rozpatrywanym układzie konwersji nagrania z płyty CD (44.1 kHz) do częstotliwości radia DAB (48 kHz) dla $K = 160$, $L = 147$ powinien być użyty jeden filtr o szerokości $\frac{1}{160} = 0.00625$ połowy częstotliwości próbkowania $f_{pr}^{max} = 160 \cdot 44.1$ kHz = 7.056 MHz. Aby zapewnić takie wąskie pasmo i równocześnie dużą stromość odpowiedzi częstotliwościowej filtra, filtr FIR musi być bardzo długi, czyli kosztowny obliczeniowo. W takiej sytuacji o wiele korzystniej jest zrealizować interpolator lub decymator wysokiego rzędu jako kaskadę interpolatorów/decymatorów niższych rzędów, ale połączoych szeregowo, jeden za drugim. Przykładowo: interpolator rzędu 160 może być zaimplementowany jako kaskadowe połączenie interpolatorów rzędu $[2, 2, 2, 2, 2, 5]$ (wynik działania funkcji `factor(160)`) lub jako np. $[4, 4, 10]$, natomiast decymtor rzędu 147 może być zrealizowany jako szeregowe połczeczenie decymatorów rzędu $[3, 7, 7]$ (wynik wywołania funkcji `factor(147)`). Napisz program do przepróbkowania sygnału standardu CD (44.1 kHz) do standardu DAB (48 kHz), realizowanego za pomocą tylko filtra interpolatora (ponieważ jest węższy!), zaimplementowanego w sposób kaskadowy, tzn. nadpróbkowanie, po kolei jedno za drugim: 1) 4-razy, 2) 4-razy, 3) 10-razy, 4) potem pozostawisz co 147-mą próbkę.

TEMAT #4: Inne przykłady przepróbkowywania sygnałów rzeczywistych. Od elektokadiogramu (EKG) to fonokardiogramu. Cyfrowy dekoder radia FM.

Problem 10.8 (Od EKG do fonokardiogramu).** Wczytaj sygnał EKG: `clear all; load ECG100.mat; whos` (dołączony do pierwszego laboratorium) albo pobierz z Internetu jakiś sygnał elektycznej aktywności serca (EKG), np. ze stron *https://physionet.org/about/database/*, *https://github.com/mathworks/physionet_ECG_data*. Załóż, że sygnał jest próbkowany z częstotliwością 360 Hz. Zaprojektuj swój własny układ interpolatora cyfrowego, zwiększający częstotliwość próbkowania do: 1) 8000 Hz, 2) 11025 Hz — nie korzystaj z funkcji Matlaba `fir()`, `interp()`, `resample()`, `conv()`, `filter()`. Odsłuchaj sygnał nadpróbkowany. Powtórz operację dla różnych sygnałów EKG, może dla różnych chorób serca (arytmia, tachykardia, bradykardia). Odsłuchaj wszystkie nadpróbkowane sygnały. Może jesteś w stanie rozpoznać chorobę serca akustycznie, "na ucho"?

Problem 10.9 (** Projektowanie decymatora dla programowego dekodera radia FM mono).** W programie 10.3 jest przetwarzany zespolony sygnał IQ (In-phase, Quadrature), pochodzący z demodulatora kwadraturowego. Zawiera on kilka stacji radia FM wraz z cyfrowym sygnałem tekstowym RDS. Częstotliwość prókowania jest równa $f_s = 3.2$ MHz. W programie są wykonane po kolei następujące operacje:

1. jedna, wybrana stacja radia FM jest przesunięta do częstotliwości 0 Hz (w wyniku pomnożenia sygnału przez `exp(-j*2*pi*f0/fs*(0:Nx-1)` jego widmo DFT ulega cyklicznej rotacji o częstotliwość `-f0`),
2. wynikowy sygnał jest filtrowany wokół składowej stałej (DC - 0 Hz) filtrem dolno-przepustowym o szerokości 200 kHz (1/16 całego pasma 3.2 MHz), a następnie pobierana jest z niego co 16-ta próbka — opisana 16-krotna decymacja sygnału jest realizowana za pomocą pierwszej funkcji `resample()`,
3. potem jest przeprowadzana demodulacja częstotliwościowa sygnału (częstotliwość chwilowa jest wyznaczana jako pochodna kąta liczby zespolonej),
4. teraz sygnał jest filtrowany wokół DC filtrem dolno-przepustowym o częstotliwości granicznej/odcięcia $f_0 = 12.5$ kHz oraz 8-krotnie pod-próbkowywany, czyli jest pozostawiana tylko co ósma próbka — opisana 8-krotna decymacja sygnału jest realizowana za pomocą drugiej funkcji `resample()`,
5. na samym końcu zdekodowany sygnał jest grany - powinniśmy usłyszeć wybraną audycję radia FM.

Zastąp funkcje `resample()` swoim kodem, realizującym operację przepróbkowania sygnału krok po kroku. 1) Zaprojektuj dwa filtry dolno-przepustowe, które są wykorzystywane podczas pierwszej i drugiej operacji `resample()`. 2) Filtrację wykonaj za pomocą funkcji `y=conv(x,h)`. 3) Na koncu zredukuj liczbę próbek, odpowiednio 16 i 8 razy. Zastosuj wolną oraz szybką, polifazową wersję decymatora. Początkowo, uruchamiając/odpluskwiając program, używaj krótkiego fragmentu sygnału `x=x(1:2^(22))`.
Poniższy przykład jest bardzo interesujący. Ponieważ sygnał jest zespolony, to jego widmo DFT (FFT) nie jest symetryczne. Ponieważ częstotliwość próbkowania zarejestrowanego sygnału wynosi 3.2 MHz, to szerokość jego widma jest też równa 3.2 MHz. W tym zakresie jest nadawanych kilka stacji radia FM. Obecnie w programie tylko

jedna z nich jest dekodowana, ta wstępująca w widmie dla częstotliwości `f0=-0.59e+6` MHz. Oblicz FFT dowolnego fragmentu sygnału (`X=fft(x(1:2^16))` lub PSD jakiegoś fragmentu sygnału, za pomocą metody Welcha `X=pwelch(x(1:2^18),fs)` (alternatywnie `pspectrogram(x(x(1:2^18)))`). Wyskaluj oś częstotliwości obu widm w hercach i znajdź w widmie "górki" (piki) odpowiadające innym stacjom radiowym niż ta, która jest obecnie dekodowana. Po pierwsze, znajdź "górkę" widmową stacji granej: powinna być ona widoczna w okolicach częstotliwości `f0=-0.59e+6` MHz. Znajomość jej położenia powinna ci pomóc znaleźć "górki" pozostałych stacji oraz odczytać ich częstotliwości nośne. Następnie, ustaw w programie częstotliwość nośną jednej ze znalezionych stacji jako `f0` oraz spróbuj ją zdekodować. POWODZENIA!

Listing 10.3: Dolnoprzepustowa filtracja FIR w cyfrowym odbiorniku radia FM

```
% cps_10_resample_sdr.m
clear all; close all;

FileName = 'SDRSharp_FMRadio_101600kHz_IQ.wav';  T=3;  demod=1; % sygnal radia FM

inf = audioinfo(FileName), pause                   % co jest wewnatrz zbioru IQ?
fs = inf.SampleRate;                               % czestotliwosc probkowania
[x,fs] = audioread(FileName,[1,T*fs]);             % wczytaj tylko T sekund
whos,                                              % co jest w pamieci?
Nx = length(x),                                    % dlugosc sygnalu

% Odtworzenie sygnalu zespolonego IQ z dwoch kolumn zbioru WAV, jesli konieczne dodaj Q=0
x = x(:,1) - j*x(:,2); else x = x(1:Nx,1) + j*zeros(Nx,1); end

bwSERV=200e+3; bwAUDIO=25e+3;                      % czestotliwosci: serwisu FM, probkowania audio
D1 = round(fs/bwSERV); D2 = round(bwSERV/bwAUDIO); % rzad pod-probkowania
f0 = -0.59e+6; x = x .* exp(-j*2*pi*f0/fs*(0:Nx-1)'); % ktora stacja? przesuniecie czestotliwosci do 0 Hz
x = resample(x,1,D1);                              % pod-probkowanie do czestotliwosci serwisu FM
x = real(x(2:end)).*imag(x(1:end-1))-real(x(1:end-1)).*imag(x(2:end)); % demodulacja FM
x = resample(x,1,D2);                              % pod-probkowanie do czestotliwosci audio
soundsc(x,bwAUDIO);                                % odsluchanie programu radia FM
```

Laboratorium 11
Specjalne filtry FIR - filtr Hilberta i różniczkujący

Streszczenie Podczas tego laboratorium nauczymy się projektować dwa rodzaje specjalnych filtrów cyfrowych typu FIR, czyli filtr Hilberta (przesuwnik fazowy o -90 stopni) oraz filtr różniczkujący. Pierwszy z nich jest wykorzystywany do tworzenia tzw. sygnału analitycznego: jest to sygnał o wartościach zespolonych, związany z konkretnym sygnałem rzeczywistym, który umożliwia jego bardzo prostą demodulację amplitudową, fazową i częstotliwościową. Drugi filtr jest używany w różnych metodach CPS, w których jest wymagane różniczkowanie sygnałów, m.in. także do demodulacji częstotliwościowej.

TEMAT #1: Filtr cyfrowy FIR Hilberta. Filtr Hilberta jest przesuwnikiem fazowym o $-\frac{\pi}{2}$ radianów, który dowolną oscylację opóźnia od o -90 stopni:

$$\cos(\omega_0 t) \quad \rightarrow \quad \cos\left(\omega_0 t - \frac{\pi}{2}\right) = \sin(\omega_0 t). \tag{11.1}$$

Łacząc wejście do filtra Hilberta z jego wyjściem możemy otrzymać zespolony sygnał harmoniczny (tzw. sygnał analityczny):

$$\cos(\omega_0 t) + j \cdot \sin(\omega_0 t) = e^{j\omega_0 t}. \tag{11.2}$$

Odpowiedź częstotliwościowa filtra Hilberta jest równa:

$$H_H(f) = \begin{cases} j = e^{j\pi/2}, & f < 0, \\ 0, & f = 0, \\ -j = e^{-j\pi/2}, & f > 0. \end{cases} \tag{11.3}$$

Obliczając odwrotne DtFT tej charakterystyki, tak jak w opisie problemu 9.2 tematu 2 laboratorium 9 (FIR), otrzymujemy odpowiedź impulsową cyfrowego filtru FIR Hilberta, czyli wzór na jego współczynniki wagowe:

$$h_H(n) = \begin{cases} \frac{1-\cos(\pi n)}{\pi n} = \frac{\sin^2(\pi n/2)}{\pi n/2}, & n \neq 0, \\ 0, & n = 0. \end{cases} \tag{11.4}$$

Wyznaczona teoretycznie odpowiedź impulsowa filtra $h_H(n)$ jest symetrycznie przycinana względem $n = 0$ (jest obliczana tylko dla $n = -M, ...0, ..., M$) i dlatego musi być pomnożona przez wybraną funkcję okna czasowego $w(n)$. Operacja ta zmniejsza oscylacje w charakterystyce amplitudowo-częstotliwościowej wynikowego filtra, spowodowane przez wycięcie tylko fragmentu $h_H(n)$. Transformacja Hilberta $H[x(t)]$ przetwarzanego sygnału $x(t)$ o wartościach rzeczywistych jest wykorzystywana do utworzenia jego zespolonej wersji analitycznej $x_a(t) = x(t) + j \cdot H[x(t)]$: w części rzeczywistej posiadającej oryginał, a w części urojonej — wynik jego filtracji filtrem Hilberta. Dla dowolnego sygnału widmo Fouriera jego wersji analitycznej nie ma składowych o częstotliwościach ujemnych, ma tylko składowe o częstotliwościach dodatnich, podwojone w stosunku do oryginału:

$$X_a(f) = X(f) + j \cdot H[X(f)] = X(f) + j \cdot X(f)H_H(f) = X(f) \cdot (1 + j \cdot H_H(f)) \tag{11.5}$$

$$\text{dla } f < 0: \quad X_a(f) = X(f) \cdot (1 + j \cdot (j)) = X(f) \cdot (1 - 1) = 0, \tag{11.6}$$

$$\text{dla } f > 0: \quad X_a(f) = X(f) \cdot (1 + j \cdot (-j)) = X(f) \cdot (1 + 1) = 2X(f), \tag{11.7}$$

Z tego powodu nie charakteryzuje się ono sprzężoną symetrią względem punktu $f = 0$ Hz. Wytłumaczenie tej cechy na przykładzie sygnału kosinusoidalnego $x(t) = A(t)\cos(\phi(t))$ jest następujące: otrzymujemy $x_a(t) = A(t)e^{j\phi(t)} = A(t)\cos(\phi(t)) + jA(t)\sin(\phi(t))$, w szczególności dla $x(t) = A(t)\cos(2\pi f t)$ mamy $x_a(t) = A(t)e^{j2\pi f t}$, czyli otrzymujemy zespolony sygnał harmoniczny Fouriera o częstotliwości dodatniej f, ujemnej nie ma. Amplituda i kąt sygnału analitycznego mogą być z niego "odzyskane" w bardzo prosty sposób: $A(t) = |x_a(t)|$ oraz $\phi(t) = \angle x_a(t)$. Idąc dalej:

częstotliwość chwilowa sygnału może być obliczona jako pochodna obliczonego kąta: $f_{inst} = \frac{1}{2\pi}\frac{d\phi(t)}{dt}$. I to jest właśnie główne zastosowanie filtru Hilberta: tworzenie sygnału analitycznego i prosta demodulacja sygnału!

Problem 11.1 (* Filtr Hilberta w dziedzinie częstotliwości).
Wygeneruj sygnał kosinusoidalny: `Nx=1000; fs=2000; f0=fs/40; x=cos(2*pi*(f0/fs)*(0:Nx-1));`
Oblicz sygnał analityczny funkcją Matlaba: `xa1=hilbert(x);`
Zrób to sam, korzystając z definicji odpowiedzi częstotliwościowej (11.3) filtru Hilberta:

```
X=fft(x);
n=1:Nx/2; X(n)=-j*X(n); % dodatnie czestotliwosci
X(1)=0; X(Nx/2+1)=0;
n=Nx/2+2:Nx; X(n)= j*X(n); % ujemne czestotliwosci
xH=real(ifft(X));
xa2=x+j*xH; % sygnal analityczny
```

Wyświetl na jednym rysunku sygnały `x` oraz `xH`. Powiększ początkowy fragment rysunku. Kosinus powinien zostać przekształcony na sinus. Czy tak się stało? Narysuj `plot(x,xH,'bo-')`. Czy otrzymałeś okrąg? Dlaczego właśnie okrąg? A może nie otrzymałeś okręgu? Jeśli nie otrzymałeś, to z czego to wynika? Gdzie mogłeś popełnić błąd? Następnie narysuj kilkaset punktów/par $(x(n),x_H(n))$ w pętli, ale z małym opóźnienieniem: `figure; for n=1:100, plot(x(n),xH(n),'bo'); hold on; pause(0.05); end`. Ponownie powinien zostać wykreślony okrąg: punkt po pukcie. Teraz porównaj sygnały `xa1` oraz `xa2`. Sygnał analityczny ma ciekawe widmo DFT. Oblicz widmo FFT sygnału rzeczywistego (`X=fft(x)`) oraz widmo FFT odpowiadającego mu sygnału analitycznego (`Xa=fft(xa)`). Wyświetl pierwsze widmo na rysunku górnym, a drugie na rysunku dolnym. Czym różnią się oba widma? Wyjaśnij dlaczego. Wczytaj sygnał mowy i porównaj oba widma dla tego przypadku.

Problem 11.2 (* Cyfrowy filtr FIR Hilberta). Wykorzystaj ten sam sygnał co w zadaniu poprzednim. Oblicz wagi filtra Hilberta, korzystając z poniższego kodu Matlaba:

```
M=50; N=2*M+1; n=-M:M; h=(1-cos(pi*n))./(pi*n); h(M+1)=0;
stem(n,h); title('h(n)'); xlabel('n'); grid; pause
```

Oblicz odpowiedź amplitudowo-częstotliwościową i fazowo-częstotliwościową filtra Hilberta:

```
f=-fs/2 : fs/2000 : fs/2;
H1 = polyval( h(end:-1:1), exp(-j*2*pi*f/fs) );
H2 = freqz(h,1,f,fs);
plot( f, 20*log10(abs(H1)) ); grid; xlabel('f [Hz]') );
plot( f, unwrap(angle(H1)) ); grid; xlabel('f [Hz]') );
```

Pierwszy rysunek jest, mniej więcej, poprawny ponieważ filtr Hilberta powinien przepuszczac wszystkie częstotliwości ($|H(f)| = 1$) za wyjątkiem $f = 0$ Hz ($|H(0)| = 0$). Wyraźną niedoskonałością obserwowanej odpowiedzi amplitudowej filtra są silne oscylacje w niej widoczne. Mogą być one jednak usunięte w prosty sposób metodą okienkowania wyciętego fragmentu teoretycznej odpowiedzi impulsowej filtra (wag filtra), przykładowo: `w=kaiser(N,10)'; h = h.*w` (tak samo jak podczas projektowania fitrów FIR metodą okien w laboratorium 9 - pierwszy problem tematu 2). Sprawdź czy jest to prawda? Zastosuj różne funkcje okien. Użycie funkcji okien, mających bardzo niski poziom listków bocznych w widmie Fouriera, zdecydowanie poprawia płaskość odpowiedzi amplitudowo-częstotliwościowej filtra w paśmie przepustowym, ale pogarsza (rozszerza!) pasmo przejściowe filtra w okolicach częstotliwości $f = 0$ oraz $f = \frac{f_{pr}}{2}$ — czyli zawęża pasmo pracy filtra. Pamiętaj, aby sygnał przetwarzany przez filtr Hilberta zawsze znajdował się w jego pasmie przepustowym.
Jednak wykres odpowiedzi fazowej filtra (ch-ki fazowo-częstotliwościowej), przedstawiony na drugim rysunku, jest inny niż spodziewany: dla filtra Hilberta przesunięcie fazowe powinno być stałe (dla $f < 0: +\frac{\pi}{2}$, dla $f > 0: -\frac{\pi}{2}$), a takie nie jest!? Dlaczego? Ponieważ używamy odpowiedzi impulsowej (wag filtra), która została przez nas przesunięta o `M` próbek w prawo. Przesunięciu sygnału w czasie odpowiada modyfikacja fazowa jego widma Fouriera - patrz odpowiednia właściwość ciągłej transformacji Fouriera, podana w ostatniej tabeli laboratorium 4 DtFT&DFT. Z

tego powodu oryginalna odpowiedź częstotliwościowa filtra jest pomnożona przez $e^{-j2\pi(f/f_s)M}$ i odpowiedź fazowa filtra liniowo maleje wraz ze wzrostem częstotliwości f. Aby skompensować ten efekt na wykresie, możemy pomnożyć obliczoną ch-kę częstotliwościową przez $e^{+j2\pi(f/f_s)M}$ oraz wyświetlić: `plot(f,angle(exp(j*2*pi*f/fs*M).*H1))`. Zrób tak. Mam nadzieję, że zobaczysz teraz kąt przesunięcia $+\frac{\pi}{2}$ dla ujemnych częstotliwości oraz $-\frac{\pi}{2}$ dla dodatnich częstotliwości, co jest poprawne uwzględniając równanie (11.3).

Problem 11.3 (* Zastosowanie filtra FIR Hilberta: obliczanie sygnału analitycznego). Filtr cyfrowy FIR Hilberta jest najczęściej stosowany do obliczania zespolonej wersji analitycznej $x_a(n)$ dla wybranego sygnału rzeczywistego $x(n)$: $x_a(n) = x(n) + j \cdot H[x(n)]$. Kontynuuj program, napisany podczas ostatnego zadania. Zastosuj filtr Hilberta i oblicz `xH = filter(h,1,x)`. Ponieważ cyfrowy filtr FIR Hilberta wprowadadza opóźnienie sygnału o M próbek, w stosunku do sygnału wejściowego, wejście i wyjście z filtru Hilberta powinny zostać zsynchronizowane ze sobą podczas obliczania sygnału analitycznego (synchronizacja ta jest wymagana w przypadku dowolnego filtra FIR, jeśli chcemy nałożyć na siebie sygnał wejściowy i wyjściowy):

```
x = x(M+1 : Nx-M); % synchronizacja wejścia filtra
xH = xH(2*M+1 : Nx); % synchronizacja wyjścia filtra
xa = x + j*xH; Nx = length(xa); % utworzenie sygnału analitycznego
```

Narysuj zsynchronizowane sygnały na jedny wykresie: `plot(1:Nx,x,'ro-',1:Nx,xH,'bo-')` (czy kosinus zmienił się w sinus?) oraz `plot(x,xH,'bo-')` (czy otrzymałeś okrąg?). Jeśli tak, "brawo My!": mamy w pełni funkcjonalny filtr Hilberta, czyli przesuwnik fazowy o określony kąt! Otrzymane rysunki są takie same jak te, które otrzymaliśmy w pierwszym zadaniu/problemie: wówczas zastosowaliśmy funkcję Matlaba `xa=hilbert(x)`, obliczającą sygnał analityczny metodą modyfikacji widma Fouriera sygnału wejściowego. Oblicz widma FFT sygnału oryginalnego i jego wersji analitycznej `(X=fft(x); Xa=fft(xa))`, oraz wyświetl ich moduł na jednym rysynku (poprawnie skalując oś częstotliwości). Przypomnij sobie równania (11.6)(11.7), wyprowadzone we wprowadzeniu do obecnego tematu tego laboratorium, które mówią, że widmo sygnału analitycznego: 1) nie posiada składowych dla ujemnych częstotliwości, 2) ma dwukrotnie większe wartości dla dodatnich częstotliwości niż widmo sygnału rzeczywistego/oryginalnego.

Problem 11.4 (Demodulacja AM oraz FM sygnału).** Mamy "młotek", dlatego czas zacząć "wbijać gwoździe". Wygenerujmy sinusoidę o wartościach rzeczywistch, która jest jednocześnie zmodulowana w amplitudzie i częstotliwości. Spróbujmy ją zdemodulować z użyciem pojecia sygnału analitycznego i transformacji Hilberta. Pierwsze, wstępne rozwiązanie jest zaprezentowane w kodzie z listingu 11.1. Pomysł jest następujący: 1) obliczamy zespolony sygnał analityczny dla sygnału rzeczywistego, stosując transformację/filtr Hilberta (`xa=hilbert(x)`), 2) odtwarzamy amplitudę sygnału ($A(t) = |x_a(t)|$, `A=abs(xa)`), 3) odtwarzamy kąt fazowy sygnału ($\phi(t) = \angle(x_a(t))$, `ph=unwrap(angle(xa))`, 4) odtwarzamy częstotliwość sygnału jako pochodą kąta pomnożoną przez $\frac{1}{2\pi}$ ($f_{inst}(t) = \frac{1}{2\pi} \cdot \frac{d\phi(t)}{dt}$, `finst = (1/(2*pi))*(ph(2:end)-ph(1:end))/dt;`). Pochodna kąta może być obliczona na dwa różne sposoby, tak jak to było opisane powyżej i wykonane w programie 11.1. Uruchom program, sprawdź czy sygnał jest poprawnie zdemodulowany, oblicz chwilowe wartości błędów demodulacji AM i FM, narysuj je. Zaimplementuj przesuwnik fazowy Hilberta w postaci filtra cyfrowego FIR. Narysuj wagi filtra oraz jego ch-kę częstotliwościową: amplitudową i fazową. Zastosuj filtry o różnej długości oraz wykorzystujące różne funkcje okien. Narysuj wartości błędów chwilowych demodulacji AM i FM. Spróbuj znaleźć najkrótszy filtr zapewniający akceptowalny poziom błędów. Stopniowo zwiększaj głebokość modulacji FM, czyli `kF` (aż do $f_c = 500$ Hz) oraz zauważ, że błąd demodulacji FM rośnie, ponieważ sygnał zaczyna wychodzić poza płaski odcinek charakterystyki amplitudowo-częstotliwościowej filtra i jest tłumiony.

Listing 11.1: Przykład demodulacji AM i FM sygnału z użyciem sygnału analitycznego oraz transformacji/filtru Hilberta

```
% cps_11_hilb_AMFM.m - demodulacja AM i FM z uzyciem filtru Hilberta
clear all; close all;

fpr=2000; dt=1/fpr; Nx=fpr; t=dt*(0:Nx-1);
A=10; kA=0.5; fA=2;    xA = A*(1 + kA*sin(2*pi*fA*t));
kF=250; fF=1;          xF = kF*sin(2*pi*fF*t);
fc=500;                x = xA .* sin(2*pi*(fc*t + cumsum(xF)*dt));
```

```
xa = hilbert( x );
xAest = abs( xa );
ang = unwrap(angle( xa ));
xFest = (1/(2*pi)) * (ang(3:end)-ang(1:end-2)) / (2*dt);
figure; plot(t,xA,'r-',t,xAest,'b-'); title('AM'); grid;
figure; plot(t,xF,'r-',t(2:Nx-1),xFest-fc,'b-'); title('FM'); grid;
```

Problem 11.5 ((**) BINGO! Łączna demodulacja AM i FM).** Zmodyfikuj kod programu Matlaba z listingu 11.1. Ustaw częstotliwość próbkowania na $f_{pr1} = 192$ kHz oraz ustaw częstotliwość sygnału nośnego, który jest modulowany, na $f_c = 48$ kHz. Wczytaj do programu dwa sygnały dźwiękowe `x1, x2`, spróbkowane z częstotliwością $f_{pr2} = 8$ kHz, np. `[x1,fpr2]=audioread('sound1.wav',[from,to])`. Nad-próbkuj oba sygnały do częstotliwości $f_{pr1} = 192$ kHz (np. `x1up=interp(x1,fpr1/fpr2),x2up=interp(x2,fpr1/fpr2)`). Zastosuj pierwszy sygnał `x1up` do modulacji amplitudy sygnału "nośnej" o częstotliwości 48 kHz, zaś drugi sygnał `x2up`— do modulacji częstotliwości "nośnej" (np. `x=(1+0.5*x1up).*cos(2*pi*(fc/fs1*n+10000*cumsum(x2up)*(1/fs1)));`) Oblicz sygnał analityczny `xa` dla nośnej, zmodulowanej w amplitudzie i w częstotliwości, czyli dla sygnału `x` (`xa=hilbert(x)`), oraz wykorzystaj program 11.1 do jego demodulacji: powinieneś otrzymać sygnały modulujące `xA, xF`. Pod-próbkuj otrzymane dwa sygnały modulujące (amplitudę i częstotliwość) do częstotliwości próbkowania 8 kHz, np. `xAdown= decimate(xA,fpr2/fpr1)`, oraz porównaj odtworzone sygnały modulujące z sygnałami oryginalnymi. Odsłuchaj je. Na koniec spróbuj zaimplementować filtr Hilberta w dziedzinie czasu za pomocą algorytmu filtracji cyfrowej: użyj wag filtra Hilberta `hH` oraz jednej z dwóch funkcji: `y=conv(x,hH)` albo `y=filter(hH,1,x)`. Jeśli BINGO, to otrzymasz za to dodatkowe 2 punkty.

TEMAT #2: Cyfrowe filtry różniczkujące FIR. Ciągła transformacja Fouriera (FT) pochodnej sygnału jest równa transformacji FT sygnału oryginalnego, pomnożonej przez $j\omega$ (patrz tabela na końcu instrukcji do laboratorium 4 DFT/DtFT). Z tego możemy wyciągnąć wniosek, że odpowiedź czestotliwościowa analogowego filtra różniczkującego jest równa $H(\omega) = j\omega = j2\pi f$. Kiedy obliczymy odwrotne DtFT z tej odpowiedzi (w taki sam sposób jak w metodzie okien w laboratorium 9 FIR), to otrzymamy następującą teoretyczną odpowiedź impulsową (wagi) cyfrowego filtra różniczkującego typu FIR: $h_D(n) = \frac{\cos(\pi n)}{n}$, $h(0) = 0$. Wagi te muszą zostać gładko "przycięte" po bokach przez funkcję okna czasowego (jak zwykle, *przygładzenie włosków na głowie*). W analizie numerycznej są stosowane nastepujace wagi do estymowania pochodnej sygnału: $h_2 = \frac{1}{dt}[-1,1]$, $h_3 = \frac{1}{2\cdot dt}[-1,0,1]$, $h_5 = \frac{1}{12\cdot dt}[1,-8,0,8,-1]$.

Problem 11.6 (* Cyfrowe filtry FIR różniczkujące). Przeanalizuj kod programu z listingu 11.2. Składa się on z następujacych części: 1) obliczenie wag filtra różniczkującego metodą okien (oraz porównanie ich z wagami stosowanymi podczas różniczkowania numerycznego), 2) weryfikacja kształtu odpowiedzi częstotliwościowej filtra: amplitudowej i fazowej w funkcji częstotliwości, 3) filtracja sygnału, w tym przypadku różniczkowanie, 4) porównanie sygnału wejściowego i wyjściowego. Oblicz i narysuj odpowiedzi częstotliwościowe filtrów o różnych długościach (`M=10,20,50,100`) oraz wykorzystujących różne funkcje okien (`rectwin(), hanning(), blackman()` oraz `kaiser()` z różną wartością parametru β). Dokonaj filtracji sygnału o różnej częstotliwości (`f0=10,20,50,100`): czy sinus jest zamieniany na kosinus? czy amplituda kosinusa zależy od częstotliwości sygnału?

Listing 11.2: Zastosowanie filtra FIR do różniczkowania sygnału w języku Matlab

```
% cps_11_diff.m - rozniczkowanie sygnalu z uzyciem filtra FIR
clear all; close all;

% Projekt wag filtra rozniczkujacego FIR
M=50; N=2*M+1; n=-M:M; h=cos(pi*n)./n; h(M+1)=0;
h = h .* kaiser(N,10)';
h = 1/12 * [-1, 8, 0, -8, 1]; M=2; N=2*M+1; n = -M:M; % 1/2*[-1 0 1]
stem(n,h); title('h(n)'); grid; pause
```

```
% Weryfikacja odpowiedzi filtra: amplitudowej i fazowej
fs = 2000; f = 0:1:fs/2;          % czestotliwosc probkowania [Hz], wybrane czestotliwosci
z = exp(-j*2*pi*f/fs);            % zmienna "z" transformacji Z (dokladniej z^(-1))
H = polyval(h(end:-1:1),z);       % odpowiedz czestotliwosciowa
% H = freqz(h,1,f,fs);            % funkcja Matlaba
figure; plot(f,abs(H)); xlabel('f [Hz]'); title('|H(f)|'); grid;
figure; plot(f,unwrap(angle(H))); xlabel('f [Hz]'); title('angle(H(f)) [rad]'); grid;
figure; plot(f,angle(exp(j*2*pi*f/fs*M).*H)); xlabel('f [Hz]'); title('angle(H(f)) [rad]'); grid;

% Filtracja - rozniczkowanie sygnalu
Nx=400; n=0:Nx-1; dt=1/fs; t=dt*n;
fx=50; x=sin(2*pi*fx*t);
y=filter(h,1,x);
nx=M+1:Nx-M; ny=2*M+1:Nx;
figure; plot(nx,x(nx),'ro-',nx,y(ny),'bo-'); title('x(n), y(n)'); grid; pause
```

TEMAT #3: Zastosowania cyfrowego filtra Hilberta i różniczkującego do sygnałów rzeczywistych. Sygnał nośny o wartościach rzeczywistych, zmodulowany w częstotliwości sygnałem $x(t)$, jest zdefiniowany w sposób następujący:

$$y(t) = \cos\left(2\pi\left(f_c t + \Delta f \int_0^t x(t)dt\right)\right) \tag{11.8}$$

Po jego zróżniczkowaniu otrzymujemy:

$$z(t) = \frac{dy(t)}{dt} = -\left[2\pi(f_c + \Delta f \cdot x(t))\right] \cdot \sin\left(2\pi\left(f_c t + \Delta f \int_0^t x(t)dtx\right)\right) \tag{11.9}$$

W równaniu (11.9) wysoko-częstotliwościowy sygnał $\sin(.)$, mający częstotliwość zmieniającą się wokół wartości f_c, ma wolnozmienną amplitudę/obwiednię $a(t) = [2\pi(f_c + \Delta f \cdot x(t))]$, zależącą od sygnału modulującego $x(t)$. Obwiednia $a(t)$, a następnie sygnał $x(t)$, mogą zostać odtworzone z sygnału $z(t)$ (11.9), w wyniku wykonania po sobie sekwencji następujących operacji:

1. na samym początku, zróżniczkowanie sygnału $y(t)$, otrzymanie sygnału $z(t)$ (11.9),
2. podniesienie do kwadratu sygnału $z(t)$:

$$a^2(t) \cdot \sin^2(\alpha) = a^2(t)\frac{1}{2}(1 - \cos(2\alpha)) = \frac{1}{2}a^2(t) - \frac{1}{2}a^2(t)\cos(2\alpha), \tag{11.10}$$

3. filtracja dolno-przepustowa — usunięcie składowej o częstotliwości $2f_c$ (2α) — pozostaje tylko $\frac{1}{2}a^2(t)$,
4. obliczenie pierwiastka kwadratowego:

$$\sqrt{\frac{1}{2}a^2(t)} = \frac{1}{\sqrt{2}}a(t). \tag{11.11}$$

Wszystko razem można zapisać w postaci poniższego wzoru:

$$\sqrt{\text{LPFilter}\left[\left(\frac{dy(t)}{dt}\right)^2(t)\right]} = \frac{1}{\sqrt{2}}\left[2\pi(f_c + \Delta f x(t))\right] \tag{11.12}$$

Schemat blokowy opisanego demodulatora FM jest przedstawiony na rysunku 11.1. Filtr różniczkujący powinien tylko pracować wokół częstotliwości f_c sygnału nośnej (*carrier*). Można to uzyskać na dwa sposoby, łącząc filtr różniczkujący (D) z odpowiednim filtrem pasmowo-przepustowym (BP): 1) dwa filtry pracujące jeden za drugim, najpierw BP, potem D, albo 2) jeden filtr, mający odpowiedź impulsową równą splotowi odpowiedzi impulsowych filtrów D i BP (`hBPD=conv(hBP,hD)`). Filtr dolnoprzepustowy LP usuwa wysoko-częstotliwościowy składnik o częstotliwości $2f_c$.

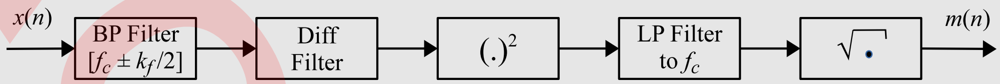

Fig. 11.1: Schemat blokowy demodulatora FM, wykorzystującego filtr różniczkujący

Sygnał (11.8) może być także zdemodulowany w częstotliwości z użyciem cyfrowego filtra Hilberta i koncepcji sygnału analitycznego: 1) najpierw jest obliczony zespolony sygnał analityczny, związany z przetwarzanym sygnałem rzeczywistym (11.8), 2) następnie jest wyznaczany jego kąt fazowy, 3) na końcu jest obliczana i odpowiednio skalowana pochodna tego kąta. Jeśli żadna "lampka nie zapala się", to wróć do ostatniego zadania Tematu #1.

Problem 11.7 (Demodulacja #1 radia FM z użyciem filtra różniczkującego).** Program, przedstawiony w listingu 11.3, implementuje opisaną powyżej metodę demodulacji radia FM. Przeanalizuj ten program, postaraj się zrozumieć każdy etap przetwarzania sygnału. Uruchom program. Zweryfikuj na rysunkach czy po każdym kolejnym kroku otrzymujemy to czego się spodziewałeś. Zaprojektuj i użyj różnych fitrów różniczkujacych (o innej długości, wykorzystujących inne funkcje okien). Wytłumacz znaczenie filtra dolnoprzepustowego. Zmodyfikuj program: do modulacji FM zastosuj sygnały audio (muzyczne), także spróbkowane 44.1 kHz, ale o zdecydowanie szerszym paśmie częstotliwościowym niż sygnał mowy (do 17.5 kHz, a nie do 4 kHz). Nad-próbkuj sygnał muzyki do wyższej częstotliwości oraz użyj nośnej o wyższej częstotliwości, np. 20 kHz lub 25 kHz. Zastosuj filtr dolno-przepustowy o szerszym paśmie. Na rysunkach funkcji `spectrogram()` zwerfikuj poprawność wszystkich wykonanych przez ciebie operacji. Odsłuchaj sygnał zdemodulowany.

Problem 11.8 (Demodulacja #2 radia FM z użyciem filtra różniczkującego).** Na początek przeczytaj opis poprzedniego zadania. Zrób to samo. Jednak ... zmodyfikuj program w inny sposób: 1) utwórz sygnały FM dwóch różnych stacji radiowych, zawierających inne, wąsko-częstotliwościowe audycje słowne (mowa 8 kHz) oraz wykorzystujących inne sygnały nośne (o różnych częstotliwościach), 2) dodaj te dwa sygnału FM do siebie, a następnie: 3) wydobądź każdy z nich z osobna z sygnału sumarycznego, stosując specjalnie zaprojektowane, różne filtry pasmowo-przepustowe, 4) dokonaj demodulacji FM obu audycji oddzielnie. Powinieneś nad-próbkować oba sygnały mowy do wyższej częstotliwości oraz użyć sygnałów nośnych (kosinusów) o różnych, wyższych częstotliwościach.

Problem 11.9 (Demodulacja radia FM za pomocą filtru Hilberta i sygnału analitycznego).** Program przedstawiony w listingu 11.3 implementuje opisaną powyżej metodę demodulacji radia FM z użyciem filtra rózniczkującego. Zastosuj filtr Hilberta, oblicz sygnał analityczny oraz wykorzystaj go do alternatywnej demodulacji radia FM. W tym celu wykorzystaj fragmenty programu 11.1. Zmień wartości parametrów, jeśli jest to wskazane/konieczne.

Listing 11.3: Dekoder radia FM wykorzystujący filtr różniczkujący

```
% csp_11_fmdemod.m - rozniczkowanie sygnalu z uzyciem filtra cyfrowego FIR
% mowa probkowana 44.1 kHz, modulujaca czestotlliwosc kosinusa 7.5 kHz
clear all; close all;

% Parametry
K = 1;                  % opcjonalne K-krotne nad-probkowanie sygnalu mowy 44.1 kHz
fc = 7500;              % czestotliwosc nosnej/kosinus (Hz)
M = 200; N=2*M+1;       % N = 2M+1 = dlugosc projektowanych filtrow FIR

fmax = 4000;            % zalozona maksymalna czestotliwosc mowy (Hz)
DF = 5000;              % 2*DF = wymagane pasmo sygnalu z modulacja FM (Hz)
kf = (DF/fmax-1)*fmax;  % indeks modulacji z reguly Carsona (poszukaj w Internecie)

% Wczytaj sygnal radia FM, modulujacy nosna
[x,fx] = audioread( 'speech44100.wav', [1,1*44100] );  % probki [od,do]
soundsc(x,fx);  x = x';                 % posluchaj
x = resample(x,K,1); fs = K*fx;         % opcjonalnie nad-probkuj
Nx = length(x); dt=1/fs; t=dt*(0:Nx-1);  % uzywane zmienne
figure;
subplot(211); plot(t,x); grid; xlabel('t (s)'); title('x(t)');
subplot(212); spectrogram(x,256,192,512,fs,'yaxis'); title('STFT(1)'); pause

% Modulacja FM
y =  cos( 2*pi*( fc*t + kf*cumsum(x)*dt ) ); % sygnal zmodulowany w czestotliwosci (FM)
figure; spectrogram(y,256,192,512,fs,'yaxis'); title('STFT(2)'); pause

% Filtr rozniczkujacy i rozniczkowanie sygnalu
n=-M:M; hD=cos(pi*n)./n; hD(M+1)=0; w = kaiser(2*M+1,10)'; hD = hD .* w; % projekt filtra
y = filter(hD, 1, y);  y = y(N:end);         % filtracja = rozniczkowanie
figure; spectrogram(y,256,192,512,fs,'yaxis'); title('STFT(3)'); pause

% Podniesienie do potegi
y = y.^2;                                    % potegowanie
figure; spectrogram(y,256,192,512,fs,'yaxis'); title('STFT(4)'); pause

% Filtracja dolno-przepustowa
n=-M:M; hLP=sin(2*pi*4000/fs*n)./(pi*n); hLP(M+1)=2*(4000)/fs; hLP = hLP .* w;  % projekt filtra
y = filter(hLP, 1, y ); y = y(N:end);        % filtracja dolno-przepustowa
figure; spectrogram(y,256,192,512,fs,'yaxis'); title('STFT(5)'); pause

% Pomnozenie przez 2 oraz obliczenie pierwiastka kwadratowego
y = real( sqrt(2*y) );
figure; spectrogram(y,256,192,512,fs,'yaxis'); title('STFT(6)'); pause

% Koncowe skalowanie
y = (y - 2*pi*fc/fs)/(2*pi*kf/fs);
figure; spectrogram(y,256,192,512,fs,'yaxis'); title('STFT(7)'); pause

% Wynik koncowy
t=t(2*N-1:Nx); figure;
subplot(211); plot(t,y); grid; xlabel('t (s)'); title('y(n)');
subplot(212); spectrogram(y,256,192,512,fs,'yaxis'); title('STFT(8)'); pause
x = resample(y,1,K);
soundsc(x,fx); pause
```

AGH

Laboratorium 12
Filtry adaptacyjne FIR

Streszczenie To laboratorium jest poświęcone nierekursywnym filtrom adaptacyjnym typu LMS, NLMS i RLS. Poznamy je i użyjemy w typowych zastosowaniach: 1) adaptacyjnym usuwaniu interferencji (metodą adaptacyjnego usuwania skorelowania ze sobą dwóch sygnałów) oraz 2) adaptacyjnym uzdatnianiu/odszumieniu sygnałów, np. do do adaptacyjnej poprawy jakości linii transmisyjnej. W szczególności poznamy dwa szczególne przypadki zastosowań 1), czyli adaptycje usuwanie echa oraz adaptacyjne usuwanie szumu.

TEMAT #1: Wprowadzenie do filtracji adaptacyjnej - główne typy zastosowań: usuwanie interferencji oraz odszumianie. Filtr adaptacyjny jest bloczkiem obliczeniowym mającym dwa sygnały wejściowe (patrz rysunek 12.1):

1. $d(n)$ - odniesienia,
2. $x(n)$ - adaptacyjnie filtrowany,

oraz dwa sygnały wyjściowe:

1. $y(n) = \text{adapt}[x(n)]$ - wynik filtracji sygnału $x(n)$,
2. $e(n) = d(n) - y(n)$ - sygnał błędu/niedopasowania.

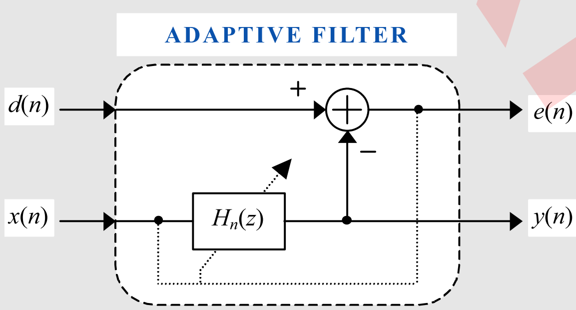

Fig. 12.1: Schemat blokowy filtra adaptacyjnego, mającego dwa wejścia: $d(n)$, $x(n)$, oraz dwa wyjścia: $e(n)$, $y(n)$

Zadaniem filtra jest spowodowanie, aby sygnał $x(n)$ po filtracji upodobnił się do sygnału $d(n)$: $y(n) \to d(n)$. Adaptacyjna zmiana wag filtra, i w konsekwencji jego charakterystyki częstotliwościowej, ma zapewnić minimalizację wybranej funkcji kosztu:

1. kwadratu błędu niedopasowania $y(n)$ do $d(n)$, $J(n) = e^2(n)$: (N)LMS (Normalized Least Mean Squares) - (znormalizowany) filtr najmniejszych średnich kwadratów,
2. ważoną sumę obecnej i poprzednich wartości kwadratu błędu, czyli $J(n) = \sum_{k=0}^{+\infty} \lambda^k \cdot e^2(n-k), \lambda \leq 1$: RLS/WRLS (Weighted Recursive Least Squares) - rekursywny filtr (ważonych) najmniejszych kwadratów.

Dla filtra LMS szukane są takie wartości jego współczynników $h_k(n)$, które w każdej chwili n zapewnią minimalizację *funkcji kosztu*:

$$J(n) = (d(n) - y(n))^2 = \left(d(n) - \sum_{k=0}^{M} h_k(n) x(n-k) \right)^2. \tag{12.1}$$

Aby je znaleźć, oblicza się pochodną funkcji $J(n)$ po każdym współczynniku:

$$\frac{dJ(n)}{dh_k(n)} = 2 \cdot e(n) \cdot \frac{d}{dh_k(n)} \left[d(n) - h_0(n) \cdot x(n-0) - \ldots - h_M(n) \cdot x(n-M) \right], \tag{12.2}$$

$$\frac{dJ(n)}{dh_k(n)} = -2 * e(n) \cdot x(n-k), \tag{12.3}$$

a następnie, podczas adaptacji, zmienia się wartosci współczynników w kierunku przeciwnym do wzrostu funkcji kosztu, czyli jej gradientu (stąd znak minus poniżej):

$$h_k(n+1) = h_k(n) - \frac{dJ(n)}{dh_k}, \tag{12.4}$$

$$h_k(n+1) = h_k(n) + 2 \cdot \mu \cdot e(n) \cdot x(n-k). \tag{12.5}$$

Trudne - dla ambitnych - do opuszczenia przez miłośników górskich dolinek. Optymalna wartość wag filtra dla przypadku stacjonarnego została wyprowadzona przez Wienera: $\mathbf{h}_{opt} = \mathbf{R}_{xx}^{-1} \cdot \mathbf{r}_{dx}$, gdzie $\mathbf{R}_{xx}$ jest macierzą autokoracji sygnału $x(n)$, zaś $\mathbf{r}_{dx}$ - wektorem korelacji wzajemnej pomiędzy sygnałami $d(n)$ oraz $x(n)$. Maksymalna wartość współczynnika szybkości adaptacji μ jest ograniczona od góry równaniem: $0 < \mu < \frac{2}{\lambda_{max}}$, gdzie λ_{max} jest największą wartością własną macierzy $\mathbf{R}_{xx}$ — dla za dużych wartości μ filtr staje się niestabilny. Filtr jest najszybciej zbieżny kiedy $\lambda_{min}/\lambda_{max} \approx 1$, czy kiedy $x(n)$ jest szumem. W filtrze adaptacyjnym typu RLS adaptacyjnie rozwiązuje się równanie Wienera na wagi optymalne: iteracyjnie wyznacza się w nim macierz $\mathbf{R}_{xx}^{-1}$.

W tym temacie skoncentrujemy się wyłącznie na adaptacyjnych filtrach (N)LMS oraz rozpatrzymy tylko dwa scenariusze ich zastosowania:

- adaptacyjne usuwanie korelacji ACC (Adaptive Correlation Canceling) pomiędzy dwoma sygnałami, na przykład pomiędzy mową pilota samolotu z dodanym warkotem silnika oraz tym samym warkotem silnika, ale zarejestrowanym przez inny mikrofon (jest to wzmocniona/stłumiona oraz opóźniona/przyspieszona "kopia" warkotu silnika, który dodał się do mowy pilota); inna nazwa: adaptacyjne usuwanie interferencji AIC (Adaptive Interference Canceling),
- adaptacyjne uzdatnianie pojedynczego sygnału ASE/ALE (Adaptive Signal/Line Enhancement) metodą adaptacyjnej liniowej predykcji.

Listing 12.1 zawiera prosty program demonstrujący filtrację adaptacyjną LMS dla obu przypadków/scenariuszy (wszystkie rysunki są w elektronicznej wersji programu). Z kolei listing 12.2 zawiera ogólniejszą funkcję z filtrami LMS, NLMS i WRLS.

Listing 12.1: Demonstracja dwóch scenariuszy użycia filtra adaptacyjnego LMS

```
% cps_12_adapt_simple.m - proste demo filtracji adaptacyjnej LMS
clear all; close all;

% Parametry sygnalow
fpr = 1000;                  % czestotliwosc probkowania
Nx = fpr;                    % liczba probek, 1 sekunda
dt = 1/fpr; t = 0:dt:(Nx-1)*dt; % czas
f = 0:fpr/1000:fpr/2;        % czestotliwosc

if(0) % Scenariusz #1 - adaptacyjne usuwanie interferencji
   M = 50;                            % liczba wag filtra
   mi = 0.1;                          % szybkosc adaptacji ( 0<mi<1)
   s = sin(2*pi*10*t).*exp(-25*(t-0.5).^2); % sygnal: sinus*gaussoida, EKG lub mowa
   z = sin(2*pi*200*t);               % zaklocenie: harmoniczne, "warkot"
   d = s + 0.5*z;                     % sygnal + przeskalowane zaklocenie
   x = 0.2*[zeros(1,5) z(1:end-5)];   % opozniona i stlumiona kopia zaklocenia
else  % Scenariusz #2 - adaptacyjne odszumianie/uzdatnianie sygnalu
   M = 10;                            % liczba wag filtra
   mi = 0.0025;                       % szybkosc adaptacji ( 0<mi<1)
   s = sin(2*pi*10*t);                % sygnal: sinus, EKG lub mowa
```

```
  z = 0.3*randn(1,Nx);                    % zaklocenie szumowe
  d = s + z;                              % zaklocony sygnal
  x = [ 0, d(1:end-1)];                   % opozniona kopia zakloconego sygnalu
end

% Filtracja adaptacyjna
bx=zeros(M,1);         % inicjalizacja bufora na probki sygnalu wejsciowego x(n)
h  = zeros(M,1);       % inicjalizacja wag filtra
y  = zeros(1,Nx);      % wyjscie puste, jeszcze nic nie zawiera
e  = zeros(1,Nx);      % wyjscie puste, jeszcze nic nie zawiera
for n = 1 : length(x)            % Petla glowna
  % n                            % indeks petli
    bx = [ x(n); bx(1:M-1) ];    % wstawienie nowej probki x(n) do bufora
    y(n) = h' * bx;              % filtracja x(n), czyli estymacja d(n)
    e(n) = d(n) - y(n);          % blad estymacji
    h = h + ( 2*mi * e(n) * bx ); % LMS - adaptacja wag filtra
end
```

Listing 12.2: Funkcja Matlaba implementujaca filtry adaptacyjne (N)LMS oraz RLS

```
function [y, e, h] = adaptTZ(d,x,M,mi,gamma,lambda,delta,ialg)
% M-dlugosc filtra, LMS: mi, NLMS: mi, gamma, RLS: lambda, delta
% Inicjalizacja
h  = zeros(M,1);      % wagi filtra
bx = zeros(M,1);      % bufor wejsciowy na probki x(n)
Rinv = delta*eye(M,M); % odwrotnosc macierzy autokorelacji Rxx^{-1} sygnalu x(n)
for n = 1 : length(x)          % glowna petla filtracji adaptacynej
    bx = [ x(n); bx(1:M-1) ];  % pobranie nowej probki sygnalu x(n) do bufora
    y(n) = h' * bx;            % filtracja sygnalu x(n): y(n) = sum( h .* bx)
    e(n) = d(n) - y(n);        % obliczenie bledu niedopasowania e(n)
    if(ialg==1)       % filtr LMS
        h = h + mi * e(n) * bx;                % korekta wag filtra
    elseif(ialg==2)   % filtr NLMS
        energy = bx' * bx;                     % energia sygnalu x(n) w buforze
        h = h + mi/(gamma+energy) * e(n) * bx; % korekta wag filtra
    else              % filtr RLS
        Rinv = (Rinv - Rinv*bx*bx'*Rinv/(lambda+bx'*Rinv*bx))/lambda; % korekta Rinv
        h = h + Rinv * bx * e(n);                                     % korekta h
    end
end
```

Problem 12.1 (* Adaptacyjne usuwanie interferencji z użyciem filtra NLMS. *Jak mogę pracować w tym zgiełku?!*).** Funkcja Matlaba `adaptTZ()`, zaprezentowana w listingu 12.2, implementuje standardowy oraz znormalizowany adaptacyjny filtr typu LMS, czyli LMS i NLMS (także filtr RLS, ale obecnie nie interesujemy się nim). Napisz program główny, wywołujacy tę funkcję. Sugeruj się programem 12.1, uruchom go dla scenariusza 1. Nagraj 3-4 sekundy swojej mowy, spróbkowanej z częstotliwością $f_{pr} = 8$ kHz, oraz wygeneruj sinusoidę $s(n)$ o częstotliwości 1 kHz, spróbkowanej także z $f_{pr} = 8$ kHz. Dodaj sinusoidę do mowy oraz potraktuj ten sygnał jako $d(n)$ w filtrze adaptacyjnym. Następnie: pomnóż sinusoidę przez 0.25, opóźnij ją o 4 próbki oraz wykorzystaj jako sygnał $x(n)$ w filtrze: `x=[0 0 0 0 0.25*s(1:end-4)]`. Zastosuj filtr adaptacyjny do zmniejszenia poziomu zakłócania sygnału Twojej mowy przez sinusoidę. Dodaj rysunek do pętli adaptacyjnej funkcji `adaptTZ()`, wyświetlający zmieniającą się odpowiedź amplitudowo-częstotliwościową filtra (`f=0:fpr/2000:fpr/2; plot(abs(freqz(h,1,f,fpr)))`. Zastąp sinusoidę sygnałem z sinusoidalną modulacją częstotliwości SFM: $f_c = 1000$ Hz, $\Delta f = 500$ Hz, $f_m = 0.25$ Hz (skorzystaj z laboratorium 2, dotyczącego generacji różnych sygnałów). Zastąp sygnał SFM nagraniem pracy dowolnego, pracującego silnika, np. suszarki do włosów, odkurzacza, lodówki, silnika samochodowego, itp. Możesz zmienić długość filtra `M=5, 10, 20, 50`, wartość współczynnika szybkości adaptacji $\mu < 1, np. \mu = 0.001, 0.01, 0.1$ (im mniej tym wolniej), skorzystać z algorytmu LMS oraz NMLS. Szybkie zmiejszanie się wartości bezwzględnej sygnału `e(n)` potwierdza wybór dobrych wartości parametrów.
Jeśli masz problemy z poprawnym/skutecznym wyborem wartości parametrów filtra (długość `M`, współczynniki adaptacji: `LMS -- mi, NLMS -- mi, gamma`, to proszę, rzuć okiem do listingu 12.3 i tam spróbuj znaleźć

odpowiedzi na "dręczące" Cię pytania (możesz nawet skopiować fragmenty programu 12.3). Możesz także najpierw rozwiązać zadania 12.5 lub 12.6, zanim przystąpisz do realizacji obecnego problemu.

Problem 12.2 (* Adaptacyjna poprawa jakości sygnału/linii transmisyjnej (ASE/ALE) z użyciem filtra NLMS. *Czy ktoś rozumie co ja teraz mówię?!*).** Uruchom program 12.1 dla scenariusza 2, przeanalizuj otrzymane rysunki. Wykorzystuj kod programu podczas realizacji tego zadania. Nagraj swoją własną mowę $s(n)$ oraz dodaj do niej szum, wygenerowany przez funkcję Matlaba `randn()` (albo nagraj i dodaj szum niedostrojonego radia FM lub stacji TV). Potem użyj funkcji `adaptTZ()` z listingu 12.2 oraz zastosuj strategię adaptacyjnej poprawy jakości sygnału metodą liniowej predykcji: `d=[s(2:end)]; x=s(1:end-1);`. Odsłuchaj sygnał przed filtrem i po filtrze. Oblicz stosunek sygnału do szumu (SNR) przed filtrem i po fitrze. Współczynnik SNR był zdefiniowany w laboratorium 2, dotyczącym generacji sygnałów i obliczania ich parametrów. Nie zapomnij: dokonaj pomiaru SNR dopiero po zaadaptowaniu się wag filtra (obserwuj na rysunku proces adaptacji wag i znajdź poprawny punkt startowy pomiaru SNR). Spróbuj znaleźć takie wartości parametrów filtra (N)LMS (długość `M`, szybkość adaptacji `mi`), które zapewniają największą wartość współczynnika SNR po filtracji.
Jeśli masz problemy ze skutecznym doborem wartości parametrów filtra (długość `M`, współczynniki adaptacji: `LMS -- mi, NLMS -- mi, gamma,` to proszę, popatrz do listingu 12.3 i postaraj się znaleźć w nim odpowiedzi na Twoje pytania (nawet skopiuj wybrane fragmenty programu). Możesz także najpierw wykonać zadanie 12.9 lub 12.11, zaniem przystąpisz do rozwiązania obecnego problemu.

TOPIC #2: Wprowadzenie do filtrów adaptacyjnych RLS/WRLS. W tej części laboratorium poznamy filtry adaptacyjne (W)RLS (Weighted Recursive Least Squares) oraz porównany ich szybkość przestrajania się z filtrami adaptacyjnymi (N)LMS.

Problem 12.3 (* Filtr RLS kontra SledgeHammer! Walka wieczoru.). Czy adaptacyjne filtry RLS potrafią rzeczywiście lepiej (szybciej/dokładniej/stabilniej) nadążać za zmianami sygnałów niż filtry NLMS? Tak ponieważ adaptacyjnie rozwiązuje się w nich równanie Wienera na optymalne wagi filtra: $\mathbf{h}_{opt} = \mathbf{R}_{xx}^{-1} \cdot \mathbf{r}_{dx}$. Powtórz eksperymenty wykonane w zadaniach/problemach 12.1 albo 12.2, ale używając adaptacyjnego algorytmu RLS zamiast (N)LMS. Filtr RLS jest także zaimplementowany w funkcji `adaptTZ()`, przedstawionej w listingu 12.2. Przetestuj pracę filtra dla różnych wartości współczynnika zapominania $\lambda \leq 1$ oraz stałej inicjalizacyjnej δ. Jak należy je dobierać? Poszukaj inspiracji w programie 12.3. Filtr mający wartość λ bliską 1 (np. 0.999), zmienia wartości swoich wag bardzo wolno, gdyż λ^k wolno maleje ze wzrostem k i z tego powodu filtr "długo pamięta" wiele poprzednich wartości błędu w swoim kryterium adaptacji: $J(n) = \sum_{k=0}^{+\infty} \lambda^k e^2(n-k)$.
Jeśli uważasz, że to zadanie jest obecnie za trudne dla Ciebie, to porównaj szybkość zbieżności filtrów RLS i (N)LMS rozwiązując prostsze problemy/zadania 12.5 i 12.9, lub trochę trudniejsze problemy/zadania 12.6 and 12.11.

Problem 12.4 (Adaptacyjny "obserwator" RLS. *Nazywam się Holmes. Sherlock Holmes.*).** Zastosuj adaptacyjny algorytm filtra RLS, zaimplementowany w funkcji `adaptTZ()` - patrz listing 12.2. Wygeneru N_x=1000 próbek szumu gaussowskiego, używając funkcji Matlaba: `x=randn(1,Nx)`. Potraktuj je jako sygnał wejściowy $x(n)$ w filtrze adaptacyjnym. Napisz kod Matlaba, implementujący pętlę filtracji FIR, albo użyj gotowej funkcji Matlaba `d=filter(h,1,x)`. Dokonaj filtracji wygenerowanego szumu, czyli sygnału $x(n)$. Dla pierwszych $N_x/2$ próbek zastosuj wagi filtra równe $h = [1, -0.5]$, a dla drugiej połowy sygnału - zanegowane wartości wag, czyli $h = [-1, 0.5]$. Wykorzystaj wynik filtracji jako sygnał wejściowy $d(n)$ filtra adaptacyjnego. Zmodyfikuj funkcję `adaptTZ()` - powinna ona także zwracać kolejne wartości wybranych wag filtra, co umożliwi obserwację ich zmieniania się podczas procesu adaptacji. Alternatywnie, rysuj wartości wybraych wag filtra wewnątrz pętli adaptacyjnej. Wywołaj zmodyfikowaną funkcję `adaptTZ()`. Ustaw `M=2, 5, 10, 20`. Przetestuj różne wartości parametru $\lambda < 1$, np. 0.999, 0.99, 0.95, 0.9. Czy filtr poprawnie śledzi wartości chwilowe wag filtra $h = [\pm 1, \mp 0.5]$, czyli tych których użyto do filtracji sygnału $x(n)$? Rozwiązując to zadanie możesz skopiować fragmenty kodu programu 12.3.

TEMAT #3: Testowanie KROK-PO-KROKU zastosowania filtrów adaptacyjnych w różnych scenariuszach zastosowań. Adaptacyjne usuwanie interferencji (AIC). Adaptacyjne usuwanie echa (AEC). Adaptacyjne poprawianie jakości sygnału (ASE).

Listing 12.3: Program do testowania filtrów adaptacyjnych (N)LMS oraz RLS filters

```
% cps_12_adapt.m
clear all; close all;

% WYBOR: SYGNALOW, ZADANIA (AIC, ASE) oraz ALGORYTMU adaptacji (LMS, NLMS, RLS)
isig  = 1;  % SIG:  1=syntetyczny lub 2=rzeczywisty sygnal
itask = 1;  % TASK: 1=AIC adaptacyjne usuwanie interferencji
            %       2=ASE adaptacyjne odszumianie sygnalu metoda liniowej predykcji
ialg  = 1;  % ALG:  algorytm adaptacji: 1=LMS, 2=NLMS (znormalizowany LMS), 3=RLS

% INICJALIZACJA PARAMETROW FILTROW
% Filtr LMS (Least Mean Squares)
M = 50;               % liczba wag filtra
mi = 0.1;             % wspolczynnik szybkosci adaptacji ( 0<mi<1)
% Filtr NLMS (znormalizowany LMS), szybsza zbieznosc
gamma = 0.001;        % aby nie dzielic przez 0, u nas przez 0.001
% Filtr RLS (rekursywny LS) - bardziej zlozony, o wiele szybciej zbiezny
lambd = 0.999;        % RLS - wspolczynnik zapominania w funkcji kosztu LS
Rinv = 0.5*eye(M,M);  % RLS - inicjalizacja odwrotnosci macierzy Rxx

% SYGNALY TESTOWE - generacja sygnalu LFM/SFM z modulacja czestotliwosci

if(isig == 1) % SYGNALY SYNTETYCZNE ========================================

% PARAMETRY ogolne
fpr = 1000;                 % czestotliwosc probkowania
Nx = 1*fpr;                 % liczba probek
f=0:fpr/500:fpr/2;          % czestotliwosci na rysunkach
dt=1/fpr; t=0:dt:(Nx-1)*dt; % czas na rysunkach

% SYGNAL
A = 1;          % amplituda sygnalu
f0 = 0;         % LFM: czestotliwosc poczatkowa
df = 100;       % LFM: przyrost czestotliwosci [Hz/s], SFM: glebokosc modulacji [Hz]
fc = 200;       % SFM: czestotliwosc sygnalu nosnego
fm = 1;         % SFM: czestotliwosc modulujaca sygnal nosny
s1 = A*cos( 2*pi* (f0*t + 0.5*df*t.^2) );                   % SYGNAL LFM
s2 = A*cos( 2*pi* (fc*t + df/(2*pi*fm)*cos(2*pi*fm*t) ) ); % SYGNAL SFM
s = s1;                                                     % WYBOR

% OBWIEDNIA sygnalu - ksztalt zmiany amplitudy
env  = 1;  % wybor obwiedni sygnalu:
           % 0=prostokatna, 1=alfa*t, 2=exp(-alfa*t), 3=gaussowska
if (env==0)          w = boxcar(Nx)'; end          % 0 = prostokatna
if (env==1) alfa=5;  w=alfa*t;        end          % 1 = alfa*t
if (env==2) alfa=5;  w=exp(-alfa*t);  end          % 2 = exp(-alfa*t)
if (env==3) alfa=10; w=exp(-alfa*pi*(t-0.5).^2); end % 3 = gaussowska

% SYGNAL Z OBWIEDNIA
s = s .* w;

% SYGNALY WEJSCIOWE FILTRA
if (itask==1)                       % TEST 1 - usuwanie interferencji
   P = 0;                           % brak predykcji
   x = 0.1*sin(2*pi*200*t-pi/5);    % interferencja, stlumiona i opozniona
   d = s + 0.5*sin(2*pi*200*t);     % sygnal + interferencja
end
if (itask==2)                       % TEST 2 - odszumianie predykcyjne
   P = 1;                           % rzad liniowej predykcji 1,2,3,...
   x = s + 0.25*randn(1,Nx);        % sygnal + szum
   d = [ x(1+P:length(x)) 0 ];      % d = sygnal x przyspieszony o P probek
end

else % SYGNALY RZECZYWISTE ==================================================
```

```
    [s, fpr] = audioread('speech8000.wav'); s=s';
    [sA,fpr] = audioread('speakerA.wav');   sA=sA';
    [sB,fpr] = audioread('speakerB.wav');   sB=sB';
    P = 1;  % opoznienie w probkach
    if(itask==1)                                  % TEST 1
       s = sA;                                    % zapamietaj do porownania
       x = sB; Nx = length(x);                    % oryginalny sygnal echa
       d = sA + 0.25*[ zeros(1,P) sB(1:end-P) ];  % dodana "kopia" echa:
    end                                           % slabsza (0.25), opozniona (P)
    if(itask==2)                            % TEST 2
       x = s; Nx = length(x);               % oryginalny, zaszumiony sygnal mowy
       d = [ x(1+P:length(x)) zeros(1,P) ]; % d = sygnal x przyspieszony o P probek
    end
    f=0:fpr/500:fpr/2;                      % czestotliwosci dla rysunkow
    dt = 1/fpr; t = dt*(0:Nx-1);            % czas dla rysunkow
end %==================================================================

% Rysunki - sygnaly wejsciowe filtra adaptacyjnego
figure;
subplot(211); plot(t,x); grid; title('WE : sygnal x(n)');
subplot(212); plot(t,d); grid; title('WE : sygnal d(n)'); xlabel('czas (s)'); pause

% TROCHE TEORII
% Obliczenie wag optymalnego filtra Wienera oraz ograniczenia na wspolczynnik mi
for k = 0 : M
  rxx(k+1) = sum( x(1+k:Nx) .* x(1:Nx-k) )/(Nx-M);  % auto-korelacja sygnalu x(n)
  rdx(k+1) = sum( d(1+k:Nx) .* x(1:Nx-k) )/(Nx-M);  % korelacja wzajemna sygnalow d(n) i x(n)
end
Rxx = toeplitz(rxx,rxx);          % symetryczna macierz autokorelacji sygnalu x(n)
h_opt = Rxx\rdx';                 % wagi optymalnego filtra Wienera
lambda = eig( Rxx );              % wartosci wlasne macierzy Rxx
lambda = sort(lambda,'descend');  % sortowanie wartosci wlasnych
disp('Sugerowana wartosc mi:')
mi1_risc = 1/lambda(1),           % odwrotnosc maksymalnej wartosci wlasnej
mi2_risc = 1/sum(lambda), pause   % odwrotnosc sumy wszystkich wartosci wlasnych
figure;
subplot(211); stem( h_opt ); grid; title('Wagi optymalnego filtra Wienera h(n)');
subplot(212); stem( lambda ); grid; title('Wartosci wlasne macierzy Rxx');
% mi = mi2_risc/20;

% FILTRACJA ADAPTACYJNA
bx = zeros(M,1);          % inicjalizacja bufora na probki sygnalu wejsciowego x(n)
h  = zeros(M,1);          % inicjalizacja wag filtra
e = zeros(1,Nx);          % inicjalizacja wyjscia 1
y = zeros(1,Nx);          % inicjalizacja wyjscia 2
for n = 1 : Nx                    % Petla glowna
    bx = [ x(n); bx(1:M-1) ];     % wstawienie nowej probki x(n) do bufora
    y(n) = h' * bx;               % filtracja x(n), czyli estymacja d(n)
    e(n) = d(n) - y(n);           % blad estymacji d(n) przez y(n)
    if (ialg==1)  % LMS  ########
       h = h + ( 2*mi * e(n) * bx );             % LMS  - adaptacja wag filtra
    end
    if (ialg==2)  % NLMS ########
       eng = bx' * bx;                           % energia sygnalu w buforze bx
       h = h + ( (2*mi)/(gamma+eng) * e(n) * bx ); % adaptacja wag filtra
    end
    if (ialg==3)  % RLS #########
       Rinv = (Rinv - Rinv*bx*bx'*Rinv/(lambd+bx'*Rinv*bx))/lambd; % nowe Rinv
       h = h + (e(n) * Rinv * bx );                                % nowe wagi
    end
    if(0) % Obserwacja wag filtra oraz jego odpowiedzi czestotliwosciowej
       subplot(211); stem(h); xlabel('n'); title('h(n)'); grid;
       subplot(212); plot(f,abs(freqz(h,1,f,fpr))); xlabel('f (Hz)');
       title('|H(f)|'); grid; % pause
    end
```

```
end

% RYSUNKI - sygnaly wyjsciowe z filtra adaptacyjnego
figure;
subplot(211); plot(t,y); grid; title('WY : sygnal y(n) = destim');
subplot(212); plot(t,e); grid; title('WY : sygnal e(n) = d(n)-y(n)');
xlabel('czas [s]'); pause
if (itask==1)
   figure; subplot(111); plot(t,s,'r',t,e,'b'); grid; xlabel('czas [s]');
   title('Oryginal (RED), wynik filtracji (BLUE)'); pause
end
if (itask==2)
   n=Nx/2+1:Nx;
   SNR_in_dB =  10*log10( sum( s(n).^2 ) / sum( (d(n)-s(n)).^2 ) ),
   SNR_out_dB = 10*log10( sum( s(n).^2 ) / sum( (s(n)-y(n)).^2 ) ),
   n=1:Nx-P;
   figure; subplot(111); plot(t(n),s(n),'k',t(n),d(n),'r',t(n),y(n),'b');
   grid; xlabel('czas (s)');
   title('Odniesienie (BLACK), zaszumiony (RED), odszumiony (BLUE)'); pause
end
figure; subplot(111); plot(1:M+1,h_opt,'ro-',1:M,h,'bx-'); grid;
title('h(n): Wiener (RED), nasze (BLUE)');   xlabel('n'); pause
```

TEMAT #3A: KROK-PO-KROKU adaptacyjne usuwanie oscylacyjnych zakłóceń ("warkotu"). Adaptacyjne usuwanie interferencji AIC. Adaptacyjne usuwanie echa AEC.

Problem 12.5 (* Wprowadzenie do usuwania interferencji z użyciem sygnałów syntetycznych). Przeanalizuj kod programu 12.3. Uruchom go dla następujących ustawień:

`isig=1; itask=1, env = 0,3, ialg = 1,2,3.`

Obejrzyj wyniki na rysunkach. Spróbuj je zinterpretować. Filtr adaptacyjny pracuje teraz w układzie adaptacyjnego usuwania interferencji:

```
d(n) = s(n) + osc2(n);
x(n) = osc1(n);
```

Chcemy usunąć z `d(n)` oscylacyjne zakłócenie (interferencję) `osc2(n)`, które dodało się do naszego sygnału `s(n)`. Sygnał `osc1(n)` jest używany jako wzorzec zakłócenia: jest on podobny do `osc2(n)`, ale jednak inny, gdyż nagrany oddzielnie przez inny czujnik (np. mikrofon). Zadaniem filtra jest takie przekształcenie sygnału `osc1(n)`, aby maksymalnie upodobnić go do sygnału `osc2(n)`: czyli przesuniecie w czasie oraz przeskalowanie w amplitudzie wszystkich składowych sygnału `osc1(n)`. Po tej operacji, zmodyfikowana wersja sygnału `osc1(n)` jest odjęta od sygnału `d(n)`:

`e(n) = d(n) - y(n) = (s(n) + osc2(n)) - filtracja[osc1(n)];`

Jaka jest dopuszczalna maksymana wartość współczynnika szybkości adaptacji `mi`? Jest ona obliczana z użyciem teorii optymalnego filtra Wienera. Zastosuj różne wartości `mi` w programie, mniejsze i większe. Czy filtr jest zawsze stabilny (nie zbudza aię)? Czy końcowe wagi filtra `h(n)` mają wartości zbliżone do optymalnych wag filtra Wienera `hopt(n)`?

Problem 12.6 (Usuwanie interferencji z nagranych sygnałów rzeczywistych).** W tym zadaniu użyjemy filtra adaptacyjnego do usuwania interferencji z naszego własnego sygnału mowy. Zastosujemy następujące ustawienia w programie 12.3:

`isig=2; itask=1, ialg=1,2,3.`

Nagraj oddzielnie swoją własną mowę `s(n)` oraz zakłócający dźwięk (*warkot* `osc1(n)` dowolnej maszyny/silnika). Przefiltruj nagrany *warkot* jakiś filtrem i dodaj go do sygnału mowy. Następnie spróbuj zmniejszyć poziom zakłócenia/interferncji za pomocą filtra adaptacyjnego. Zmodyfikuj program 12.3. Kod Matlaba do przygotowania sygnałów wejściowych do filtra adaptacyjnego jest podany poniżej:

```
h = [ 0, 0.75, 0, 0.25 ];
x = vibro;
d = speech + filter(h,1,vibro);
```

Czy filtr adaptacyjny dobrze wykonał swoje zadanie? Zmień typ algorytmu adaptacji (LMS/NLMS/RLS) oraz wartości współczynników szybkości adaptacji `mi` oraz `lambd`.

Problem 12.7 (Usuwanie przydźwięku sieci z sygnału EKG).** Użyj programu 12.3. Pobierz z Internetu dowolny sygnał EKG elektrycznej aktywności serca, np. ze stron *https://physionet.org/about/database/*, *https://github.com/mathworks/physionet_ECG_data*, albo skorzystaj z sygnału ECG, analizowanego podczas pierwszego laboratorium. Dodaj do niego sinusoidę o częstotliwości 50 Hz lub 60 Hz, udającą przydźwięk siecowy (interferencję). Spróbuj zmniejszyć/usunąć to zakłócenie poprzez zastosowanie filtra adaptacynego, pracującego w układzie usuwania interferencji.

Problem 12.8 (* Usuwanie echa z sygnałów rzeczywistych).** W tym ćwiczeniu będziemy testować adaptacyjny układ usuwania (kasowania) echa (AEC - adaptive echo canceling), będący szczególnym przypadkiem scenariusza adaptacyjnego usuwania interferencji (AIC - adaptive interference canceling). Filtr adaptacyjny pracuje identycznie jak problemach/zadaniach 12.5 i 12.6, tylko używa następujących sygnałów wejściowych:

```
d(n) = speakerA(n) + echoB(n);
x(n) = speakerB(n);
```

Zadanie jest typowe dla systemów tele-konferencyjnych, systemów telefonicznych "hand-free" - np. w samochodzie, ale także dla transmisji sygnałów - przykładowo sygnał z anteny nadawczej przenika jako echo do anteny odbiorczej. `speakerA=sA` oznacza naszą mowę, `speakerB=sB` jest mową osoby, z którą rozmawiamy, wysłaną do głośnika, natomiast `echoB` jest sygnałem echa `speakerB`, zarejestrowanym przez nasz mikrofon.
Nagraj dwa czyste fragmenty mowy `sA(n)` oraz `sB(n)`, trwające około 5-10 sekund lub pobierz je z Internetu. Utwórz dwa sygnały:

```
d(n) = delayed_1sec( sA(n) ) + filtered( sB(n) );
x(n) = sB(n);
```

Opóźnienie sygnału `sA(n)` jest rekomendowane, aby dać filtrowi czas na zaadaptowanie się do nieznanej odpowiedzi impulsowej pomieszczenia (od głośnika do mikrofonu). Jako wagi filtra, symulującego pomieszczenie, zastosuj np. `h = [ 0, 0, 1, 0, 0, 0.5, 0, 0, 0.25 ]`. Sprawdź czy filtr adaptacyjny dostroił się do zadanych wag testowych `h`?
Znajdź w internecie odpowiedź impulsową jakiegoś pomieszczenia lub skorzystaj z udostępnionego zbioru `impulse_resp_11kHz.w` jako `h`. Czy filtr adaptacyjny pracuje poprawnie w tym przypadku? Jeśli nie, to pomyśl nad zmianą jego typu oraz wartości jego nastaw.

TEMAT #3B: KROK-PO-KROKU odszumianie sygnałów oraz analiza widmowa metodą adaptacyjnej liniowej predykcji. Adaptacyjna poprawa jakości sygnału/linii transmisyjnej (ASE/ALE) metodą liniowej predykcji. Adaptacyjna estymacja widma sygnału za pomocą modelowania autoregresyjnego AR.

Problem 12.9 (* Wprowadzenie do adaptacyjnego odszumiania sygnałów metodą liniowej predykcji). Adaptacyjne *uzdatnianie* sygnałów/linii (ASE/ALE - adaptive signa/linel enhancement) jest od wielu lat stosowane do poprawy jakości sygnałów mowy w systemach telekomunikacyjnych. Jego zadaniem jest odszumienie sygnału: filtr adaptacyjny dopasowuje swoje wagi do składowych sinusoidalnych mowy, gdyż pracuje jako liniowy predyktor a następnej próbki szumu przewidzieć nie może (w przeciwieństwie do mowy jako sumy sinusoid). Maksima ("górki") charakterystyki amplitudowej filtra dopasowują się więc do częstotliwości mowy i wzmacniają je — dzięki temu zostaje poprawiony stosunek sygnału do szumu. Wykorzystaj program 12.3. Zastosuj następujące ustawienia:

```
isig=1; itask=2; env=1; ialg=1,2,3;.
```

Zapoznaj się z wynikami filtracji. Spróbuj zrozumieć je i zinterpretować (są dobre/złe?). W tym przypadku sygnały wejściowe filtra są nastepujące:

```
d(n) = s(n);
x(n) = s(n-1);
```

i chcemy przewidzieć sygnał `s(n)`, dysponując sygnałem opóźnionym o jedną próbkę `s(n-1)`. Ponieważ tylko deterministyczne składowe sygnału `s(n)` mogą zostać przewidziane (a nie szum!), filtr adaptacyjny dopasowuje maksima swojej odpowiedzi częstotliwościowej do częstotliwości składowych "tonalnych" sygnału i poprawia w ten sposób współczynnik SNR (signal-to-noise ratio) stosunku sygnału do szumu.
Usuń znak komentarza "%" przed rysunkami, znajdującymi się w pętli adaptacji :

```
subplot(211); stem(h); xlabel('n'); title('h(n)');
subplot(212); plot(f,abs(freqz(h,1,f,fpr))); title('|H(f)|');
```

Zaobserwuj jak podczas adaptacji zmieniają się wartości wag `h(n)` filtra oraz jak maksima ("piki") odpowiedzi aplitudowo-częstotliwościowej filtra przesuwają się do częstotliwości składowych sygnału. Zwróć uwagę jak bardzo zwiększył się współczynnik SNR w wyniku działania filtra adaptacyjnego.

Problem 12.10 (Wprowadzenie do śledzenia widma sygnału (częstotliwości jego składowych) za pomocą adaptacyjnej liniowej predykcji).** Użyj programu 12.3. Ustaw następujące wartości parametrów: `isig=2; itask=2; ialg=1,2,3` , tzn. skorzystaj z filtra adaptacyjnego pracującego w układzie liniowej predykcji. Ustaw długość filtra `M=10`. Dokonaj filtracji nagrania swojej własnej mowy. Współczynniki liniowej predykcji (LP) $\mathbf{a} = [1, \mathbf{h}]$ zapisz do specjalnie stworzonej macierzy. Potem, na zewnątrz pętli filtracji adaptacyjnej, oblicz kolejne widma chwilowe sygnału, wykorzystując obliczone wartości współczynników LP, zapisane w wektorze **a**: : `X=freqz(1,a,f,fpr)`. Wyświetl macierz obliczonych widm sygnału (ich wartości bezwzględne!) oraz porównaj ten rysunek z rysunkiem, otrzymanym z funkcji Matlaba `pspectrogram(x,...), spectrogram(x,...)`. Wyjaśnij czym spowodowane są ich podobieństwa i różnice:

- dla funkcji spektrogram widzimy dokładne widma FFT fragmentów sygnału, pokazujące wartość częstotliwości otwierania strun głosowych oraz jej harmonicznych, oraz maksima częstotliwości rezonansowych filtra traktu głosowego,
- w przypadku naszej metody śledzimy jedynie zmianę odwiedni widma sygnału, czyli zmienność częstotliwości rezonansowych naszego traktu głosowego podczas mówienia — patrz późniejsze laboratorium, dotyczace analizy sygnału mowy.

Możesz także dodać następującą linię kodu Matlaba na końcu pętli adaptacji filtra:

```
plot(f,abs(freqz(1,a,f,fpr))); title('|X(f)|'); pause(1);
```

i wyświetlać dzięki temu chwilowe widma sygnału bezpośrednio w trakcie adaptacji wag filtra. A gdybyś tak zastosował tę metodę do analizy wieloskładnikowych sygnałów z jednoczesną modulacją AM-FM? Spróbujesz?

Problem 12.11 (Poprawa jakości (uzdatnianie) sygnałów rzeczywistych).** Zastosuj program 12.3. Użyj następujących jego ustawień:

```
isig=2; itask=2; ialg=1,2,3;.
```

Przeanalizuj wzrokowo otrzymane wyniki i zinterpretuj je. Następnie zastosuj adaptacyjny, predykcyjny algorytm poprawy jakości sygnału ASE/ALE do nagrania swojej własnej mowy, zaszumionej np. przez szum niedostrojonego radia FM lub telewizora. Alternatywnie, nagraj niezaszumioną mowę, a szum dodaj do niej w Matlabie, korzystając z generatora `randn()`.

Problem 12.12 (* Usuwanie sygnału elektrycznego skórczu mięśni z sygnału EKG).** Skorzystaj z programu 12.3. Pobierz z Internetu zapis EKG elektrycznej aktywności serca, np. ze stron *https://physionet.org/about/database/*, *https://github.com/mathworks/physionet_ECG_data*, albo użyj sygnału EKG udostępnionego podczas naszego pierwszego laboratorium. Dodaj do niego szum gaussowski (`randn()`), uprzednio przefiltrowany jakimś filtrem dolnoprzepustowym, symulujący sygnał elektryczny skórczu mięśni. Znajdź w Internecie informację na temat maksymalnej częstotliwości sygnału skórczu mięśni. Zastosuj filtr adaptacyjny do zmniejszenia zakłócenia sygnału EKG, pracujący w układzie : 1) adaptacyjnego usuwania interferencji, 2) adaptacyjnej liniowej predykcji. Która struktura filtru okazała się lepsza/skuteczniejsza?

Problem 12.13 (Usuwanie zakłóceń z nagrań dźwięków zwierząt).** Powtórz zadanie/problem (12.7) lub (12.12) dla nagrań dźwięków zwierząt, np. śpiewu ptaków, wycia psa/wilka, itp. Dźwięki pobierz z Internetu, np. ze strony *FindSounds*, albo użyj nagrań: kanarka (canary.wav) z MS Windows lub białego walenia (bluewhale.au) z programu Matlab.

Laboratorium 13
Projekt końcowy/zaliczeniowy: mowa, audio, wideo

Streszczenie To laboratorium ma formę **projektu zaliczeniowego**, kończącego cały przedmiot. Powinno ono przekonać studenta, że nauczył się czegoś praktycznego, a osoby prowadzące zajęcia — że cele kursu zostały osiągnięte. Najczęściej to laboratorium będzie wykonywane tylko przez osoby chcące poprawić ocenę końcową z laboratorium, w tym osoby które do tej pory nie uzyskały zaliczenia. Należy wybrać sobie tylko jeden projekt z wielu zaproponowanych, i postarać się wykonać go jak najlepiej/najpełniej. Można posiłkować się rozwiązaniami z literatury. Wszystkie projekty są multimedialne i dotyczą zagadnień, związanych z cyfrowym przetwarzaniem sygnałów: mowy, muzyki i obrazów (patrz rozdziały, odpowiednio, 19+20, 18+21 oraz 22 w książce [40] oraz rozdziały 14, 15 i 16 w książce [41]). Tematyka z nimi związana już została lub wkrótce zostanie omówiona na wykładzie i jest mniej-lub-bardziej poruszona w przykładowych programach, dołączonych do tego laboratorium.

TEMAT #1: Mowa. Model generacji sygnału mowy przez człowieka jest znany (patrz rysunek (13.1): powietrze wydychane z płuc (impulsy powietrza lub ciągły strumień) przechodzi przez filtr traktu głosowego $H(z)$. Kodowanie mowy polega na podziale nagranego sygnału mowy na fragmenty 25-30 milisekundowe (górny rysunek) i na wyznaczeniu dla każdego z nich wartości parametrów dla modelu syntezy mowy:

1. podjecie decyzji czy wypowiadana głoska jest dźwięczna (np. ma okres $T \neq 0$, czyli impuls jednostkowy co T próbek na wejściu traktu głosowego) czy bezdźwięczna ($T = 0$, szum na wejściu traktu głosowego),
2. wyznaczenie częstotliwości otwierania strun głosowych dla głoski dźwięcznej,
3. wyznaczenie wzmocnienia (G) oraz wartości współczynników filtra traktu głosowego $\{a_k\}$ (komory ustno-nosowej z językiem) — czyli znalezienie jego charakterystyki częstotliwościowej/rezonansowej.

Znając wartości tych parametrów, możliwe jest zsyntezowanie mowy podobnej do oryginalnej z wykorzystaniem syntezatora mowy. Zamiast przesyłać blok 180 próbek 8-bitowych przesyła się tylko 12 liczb: $\{T, G, a_1, ..., a_{10}\}$ (kompresja $\frac{180}{12} = 15$-krotna, a przecież można jeszcze te liczby skwantować, w standardzie LPC-10 do 54 bitów - kompresja 27-krotna). Podczas laboratorium spróbujesz przeprowadzić kodowanie mowy aż do poziomu bitowego oraz spróbujesz zsyntezować mowę o identycznej treści, ale wypowiadanej z różnymi emocjami lub przez różne osoby.
Ponieważ treść mowy jest zawarta w obwiedni filtra traktu głosowego, dlatego znając zmienność parametrów, które tę obwiednię opisują, możemy rozpoznać jakie słowo zostało powiedziane. Z tego powodu w tym zadaniu zajmiemy się także podstawami rozpoznawania mowy z użyciem współczynników kepstralnych, czyli kilkunastu pierwszych współczynników transformacji DCT, wykonanej na logarytmie modułu widma DFT fragmentu sygnału mowy.
Użyj programu przedstawionego w listingu 13.1, implementującego bardzo uproszczoną wersję algorytmu LPC-10 kodera i dekodera mowy. Przeanalizuj dokładnie każdą linię kodu: powinieś ją dokładnie rozumieć. Wielkości `T, gain, a(1),...,a(10)` to 12 parametrów, które umożliwiają zsyntezowanie 180 próbek mowy. Obecnie są one zapisywane w Matlabie jako liczby zmienno-przecinkowe podwójnej precyzji (double).

Listing 13.1: Program Matlaba implementujacy w sposób uproszczony algorytm LPC-10 kodera i dekodera mowy

```
% csp_13_mowa.m
% Kompresja mowy z uzyciem liniowej predykcji
clear all; close all;

ifigs = 0;              %  0/1 - czy wyswietlac rysunki w petli analizy-syntezy?

% Parametry
Mlen=240;               % dlugosc jednego, analizowanego fragmentu probek mowy
Mstep=180;              % przesuniecie pomiedzy blokami  (w probkach)
Np=10;                  % rzad predykcji liniowej  (rzad filtra IIR/AR)
where=181;              % pierwsze polozenie pobudzenia dzwiecznego "1"  (reszta to "0")
roffset=20;             % przesuniecie w funkcji autokorelacji podczas szukania jej  maksimum
lpc=[];                 % tablica na wspolczynniki [T, gain, a(1)...a(10)]
s=[];                   % cala mowa zsyntezowana
```

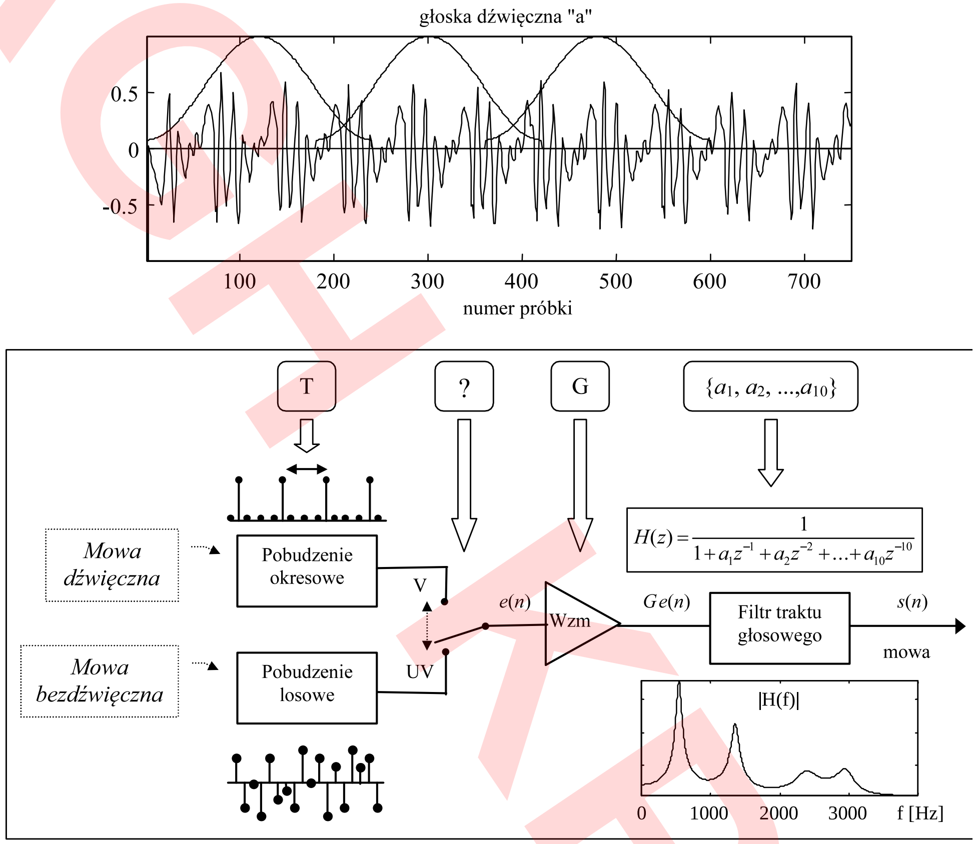

Fig. 13.1: (Góra) sygnał głoski dźwięcznej "a" i jego podział na nakładające się fragmenty 240-elementowe, z przesunięciem 180 elementów ($f_{pr} = 8000\ Hz$), (dół) schemat blokowy programowego syntezatora mowy.

```
ss=[];                    % aktualny fragment mowy zsyntezowanej

% Wczytaj sygnal mowy, ustaw wartosci parametrow kodera LPC-10
[x,fs]=audioread('speech.wav');     % wczytaj sygnal mowy 8000 Hz (audio/wav/read)
figure; plot(x); title('Speech');   % narysuj go
soundsc(x,fs); pause                % odsluchaj go na glosniku (sluchawkach)
N=length(x);                        % dlugosc sygnalu
bs=zeros(1,Np);                     % bufor na fragment zsyntezowanej mowy
Nblocks=floor((N-Mlen)/Mstep+1);    % liczba fragmentow (ramek) sygnalu mowy do analizy

% x = filter([1 -0.9735], 1, x);   % opcjonalna pre-emfaza (podbicie wysokich czestotliwosci)

% PETLA GLOWNA
for  nr = 1 : Nblocks
   % pobierz nowy fragment probek mowy
     n = 1+(nr-1)*Mstep : Mlen + (nr-1)*Mstep;       % indeksy probek
     bx = x(n);                                       % wstawienie do bufora
   % bx = bx .* hamming(Mlen)';                       % opcjonalne okienkowanie
   % ANALIZA - oblicz wartosci parametrow dla modelu syntezy mowy
     bx = bx - mean(bx);                              % usun wartosc srednia
     for k = 0 : Mlen-1
         r(k+1)=sum( bx(1:Mlen-k) .* bx(1+k:Mlen) ); % oblicz autokorelacje
     end                                              % sprobuj: r=xcorr(x,'unbiased')
     if(ifigs==1)
       subplot(411); plot(n,bx); title('ANALIZOWANY fragment mowy x(n)');
       subplot(412); plot(r); title('Jego autokorelacja rxx(k)');
     end
```

```
    [rmax,imax] = max( r(roffset : Mlen)  );        % znajdz maksimum funkcji autokorelacji
    imax = imax+(roffset-1);                        % indeks maximum (z przesunieciem)
    if ( ( r(1) > 0.2 ) & (rmax > 0.35*r(1)) )  T=imax; else T=0; end  % czy mowa jest okresowa/dzwieczna?
    if (T>80) T=round(T/2); end                     % znaleziono druga podharmoniczna
    T, % pause                                      % pokaz wartosc okresu powtarzania mowy T
    rr(1:Np,1)=(r(2:Np+1))';                        % utworz wektor z autokorelacji
    for m=1:Np                                      %
        R(m,1:Np)=[r(m:-1:2) r(1:Np-(m-1))];        % zbuduj macierz z autokorelacji
    end                                             % a=lpc(x,Np), a=levinson(x,Np)
    a=-inv(R)*rr;                                   % znajdz wsp. filtra liniowej predykcji (LP)
    gain=r(1)+r(2:Np+1)*a;                          % znajdz wzmocnienie filtra
    H=freqz(1,[1;a]);                               % oblicz ch-ke czestotliwosciowa filtra
    if(ifigs==1) subplot(413); plot(abs(H)); title('Ch-ka czestotliwosciowa filtra'); end
  % lpc=[lpc; T; gain; a; ];                        % zapisz obliczone wartosci dla syntezatora

  % SYNTEZA - mowy z uzyciem obliczonych parametrow modelu mowienia
  % T = 0; % usun "%" oraz wybierz: T = 80, 50, 30, 0; T = round(1.75*T)
    if (T~=0) where=where-Mstep; end                % pierwsze pobudzenie, pozycja dla "1"
    for n=1:Mstep                                   % START PETLI SYNTEZY
        if( T==0)                                   % pobudzenie szumowe
          % exc=2*(rand(1,1)-0.5); where=271;       % szum losowy rownomierny
            exc=0.5*randn(1,1); where=271;          % szum losowy gaussowski
        else                                        % pobudzenie impulsowe 1/0
           if (n==where) exc=1; where=where+T;      % pobudzenie = 1
           else exc=0; end                          % pobudzenie = 0
        end                                         %
        ss(n) = gain*exc - bs*a;                    % filtracja pobudzenia
        bs = [ss(n) bs(1:Np-1) ];                   % zapisanie zsyntezowanej probki mowy do bufora
    end                                             % KONIEC PETLI SYNTEZY
    s = [s ss];                                     % zapamietaj zsyntezowany fragment mowy
    if(ifigs==1) subplot(414); plot(ss); title('ZSYNTEZOWANY fragment mowy s(n)'); pause, end
end

% Koniec!
%  s = filter(1,[1 -0.9735],s); % opcjonalna de-emfaza (obnizenie wysokich czestotliwosci)
figure; plot(s); title('Zsyntezowana mowa'); pause   % zobacz wynik
soundsc(s,fs)                                        % odsluchaj wynik
```

Problem 13.1 (* Poprawa kodera mowy LPC-10).** Zaprezentowany program nie jest idealny. Ponieważ w trakcie mówienia sygnał mowy ciągle się zmienia, dlatego często 240-próbkowe fragmenty mowy zawierają stany przejściowe pomiędzy głoskami. W takim przypadku mogą być podejmowane błędne decyzje ("ani tak, ani siak"). Przykładowo, obserwowane są duże skoki wartości `T` oraz `gain` z ramki na ramkę. Spróbuj poprawić zaproponowany algorytm i program kodera tak, aby poprawić jakość syntezowanej mowy — usunąć obecnie słyszane świsty/trzaski/stuknięcia. Nagraj specjalne sygnały testowe (na przemian mowa dźwięczna, np. "a", i bezdźwięczna, np. "sz"), sprawdzaj działanie programu oraz ulepszaj algorytm/program. Spróbuj wyznaczać częściej parametry, opisujące stan generatora mowy, np. zmniejsz wartość `Mstep` i zobacz czy nastąpiła poprawa jakości mowy syntezowanej. Zmniejsz i zwiększ rząd filtra: `Np=2,4,6,8,10,12,14` - sprawdź co to powoduje. Włącz i wyłącz filtry pre-emfazy (podbicie wyższych częstotliwości) i de-emfazy (obniżenie wyższych częstotliwości). Dodaj mnożenie wyciętego fragmentu sygnału z oknem Hamminga. Możesz skorzystać z koncepcji LSP (Linear Spectrum Pairs - patrz rozdz. 19.6 w [40]), aby interpolować kolejne stany filtra traktu głosowego ... i przestrajać go częściej.

Problem 13.2 (* Zmiana cech mowy z użyciem kodera LPC-10 (emocje, inna osoba)).** Kodery mowy dzielą sygnał mowy na fragmenty, przeprowadzają ich analize, wyznaczają wartości parametrów dla syntezatora, u nas (`T, gain, a(1),...,a(10)`), kwantują te wartości i zapisują z użyciem mniejszej liczby bitów. W dekoderze są: 1) pobierane paczki bitów ze strumienia bitów, 2) rekonstruowane wartości parametrów oraz 3) syntezowana jest mowa. Spróbuj zmodyfikować obliczone wartości parametrów opisujących fragmenty mowy tak, aby w wyniku syntezy uzyskać jeden, lub równocześnie kilka efektów: 1) emocjonalny charakter wypowiedzi (radość, zdziwienie, strach, złość,...), 2) tą samą treść ... ale wypowiedzianą przez inną osobę (np. nie kobieta tylko mężczyzna, inna kobieta, inny męższyzna, dziecko, ...). Czyli pełna mistifikacja - istny Holywood! W tym celu podczas systezy stosuj np. ciągle taką samą wartość `T=30,50,70,90` próbek, albo proporcjonalnie zmieniaj wartość tego parametru, np. dla `T=1.5*T` wartość `T` zwiększa się (niższa częstotliwość) i głos kobiecy staje się coraz bardziej męski. Intonacja to zmiejszania i zwiększanie wartości `T` w stosunku do oryginału. Sprawdź co usłyszysz dla `T=0`, czyli dla przypadku braku lub paraliżu strun głosowych.

Problem 13.3 (* Kodowanie mowy LPC-10 aż do poziomu bitowego).** Zaproponuj i przeprowadź skuteczną kwantyzacją parametrów obliczonych w koderze mowy LPC-10, czyli `T,gain,a(1),...,a(10)`, dla każdego fragmentu sygnału. Zapisz je do zbioru binarnego, zwróć uwagę na jego długość, odczytaj bity ze zbioru, zdekoduj (odtwórz) z nich sygnał mowy. Nagraj swoją własną mowę z częstotliwością próbkowania 8000 Hz i jej używaj podczas testów. Powinieneś w różny sposób kwantować parametry dla syntezatora, uwzględniając ich znaczenie psychoakustyczne (bardziej kwantować parametry mniej ważne, może np. ostatnie współczynniki a_k?). Możesz konwertować współczynniki filtra "`a`" na współczynniki "`gamma`" oraz podczas syntezy zastosować filtr pracujący w tzw. strukturze kratowej (patrz wykład lub rozdział 20 w książce [40]). Dzięki temu zabiegowi, kwantowanie może być większe/silniejsze, a towarzysząca temu utrata jakości zsyntezowanej mowy — będzie mniejsza.

Problem 13.4 (* Rozpoznawanie mowy z użyciem współczynników cepstralnych).** Rozpoznawanie mowy polega na śledzeniu obwiedni widma sygnału mowy (rezonansów filtra traktu głosowego): to w nim jest ukryta treść tego co mówimy. Kształt obwiedni widma jest opisywany przez pierwszych 12-16 współczynników kepstralnych (CC) — patrz str. 555, 571 w [40] oraz funkcja `cepstrum()` tam zamieszczona. Kepstrum jest definiowane jako wynik transformacji DCT, wykonanej na logarytmie wartości bezwzględnej widma DFT sygnału. Może być ono także obliczone ze współczynników liniowej predykcji sygnału, czyli z `a` (u nas filtra traktu głosowego). Współczynniki kepstralne kolejnych fragmentów sygnału mowy są zapisywane do macierzy. Podczas rozpoznawania poszczególnych słów, macierz współczynników kepstralnych, obliczona dla rozpaznawanego słowa, jest porównywana z macierzami słów znajdujących się w słowniku - wybierane jest słowo mające najbardziej podobną macierz. Zamiast wpółczynników kepstralnych (CC) w praktyce używa się współczynników mel-kepstralnych (MFCC), zaś do porównymania macierzy o różnych wymiarach stosuje się metodę DTW (Dynamic Time Warping) — patrz rozdz. 19.7 w [40].
Spóbuj zbudować prosty system do rozpoznwania komend głosowych. Mów ze stałą prędkością, normalnie. Nagraj kilka słów-komend ($f_{pr} = 8000$ Hz). Manualnie lub automatycznie znajdź początki i końce komend, przytnij nagrania tak, aby tylko pozostały próbki wypowiedzianego słowa. Dodaj zera do końca krótszych wektorów próbek, po tej operacji wszystkie wektory powinny mieć taką samą długość. Podziel wzorce poszczególnych słów na nakładające się fragmenty (`Mlen=240` próbek, `Mstep=180` próbek), oblicz współczynniki CC dla każdego fragmentu oraz zapisz je do macierzy. Powinieneś otrzymać zbiór macierzy o takich samych wymiarach. Następnie wybierz arbitralnie macierz CC jednego słowa ze słownika i porównaj ją z wszystkimi macierzami: w tym celu oblicz współczynniki korelacji pomiędzy wybraną macierzą, a wszystkimi macierzami, po kolei. Który współczynnik ma największą wartość? Nagraj ponownie kilka razy tę samą komendę, oblicz macierze CC dla jej wszystkich wersji testowych oraz przeprowadź rozpoznawanie z użyciem słownika: szukaj najwiekszego współczynnika korelacji. Alternatywnie, zamiast korelacji wzajemnej jako miary podobieństwa, zastosuj metodę DTW. Jej kod znajdziesz w [40] [41], podobnie jak inne proste programy, dotyczące zabawy z ASR (*Automatic Speech Recognition*).
Jeśli jesteś dalej zainteresowany ASR, szukaj informacji na temat metod "deep machine learnig".

Problem 13.5 (* Rozpoznawanie mówcy — I kto to mówi? Ty Brutusie?).** Nagraj 6 razy to samo słowo wypowiadane przez 3 różne osoby. Dla 5-ciu pierwszych zapisów oblicz współczynniki kepstralne CC i wstaw je do bazy wzorców (patrz programy z z rozdz. 19 w [40] oraz rozdz. 14 w [41], zwłaszcza `cepstrum(), dtw()`). Potem oblicz współczynniki CC dla ostatniego, 6-stego słowa, wypowiedzianego przez każdą osobę, i znajdź macierz w bazie wzorców, która jest nabardziej podobna do macierzy otrzymanej. Czy można w ten sposób rozpoznać mówiącą osobę? A może trzeba się skoncentrować na wartości parametru `T`, obliczanej w badanym analizatorze mowy, i na jej zmienności? A może jeszcze na czymś innym? A może łącznie na kilku cechach sygnału na raz? Jesteś Kolumbem. Ruszaj odkryć Amerykę!

TEMAT #2: Muzyka/audio. Podczas kodowania sygnału audio, np. w standardzie AAC (Advanced Audio Coding), sygnał jest dzielony na nakładające się w 50% bloki próbek, wymnażany z sinusoidalnym oknem czasowym (pierwsza połówka sinusoidy), transformowany z użyciem MDCT do dziedziny częstotliwościowej, normalizowany w różny sposób w poszczególnych pod-pasmach częstotliwościowych (znalezienie wartości `max(abs(x))` i podzielenie przez nią wszystkich próbek w danym podpaśmie), kwantowany, kodowany bezstratnie (koder Huffmana lub

arytmetyczny) oraz zapisywany binarnie (skanowanie wszystkich, wynikowych bitów). Podczas dekodowania wszystkie operacje są wykonywane w odwrotnej kolejności (de-skanowanie, de-kodowanie bezstratne, de-kwantyzacja, denormalizacja, odwrotne MDCT (czyli IMDCT), nakładanie okna, sumowanie z 50% przesunięciem) oraz otrzymywany jest zrekonstruowany sygnał, oczywiście nie będący idealną kopią oryginału z powodu kwantyzacji wartości. Modele psycho-akustyczne są stosowane w kodowaniu audio do obliczania siły tzw. efektu maskowania jednych częstotliwości przez drugie: silny sygnał jednej częstotliwości powoduje, że nie jest słyszany słabszy sygnał o zbliżonej częstotliwości — jeśli tak jest, to nie ma sensu przydzielać mu bitów. W kodowaniu audio dzięki dokładnym, psychoakustycznym modelom ludzkiego słuchu (a nie modelom generacji dźwięków, tak jak to było w koderach mowy) bity są przydzialne wyłącznie tym składowym częstotliwościowym, które są istotne percepcyjnie.
Obrazowo możemy powiedzieć, że w koderach audio jeden, duży strumień próbek audio, mających szerokie pasmo częstotliwościowe ("szeroka rzeka", np. 0-22 kHz), jest przekształcany na wiele mniejszych strumieni, mających wąskie pasmo częstotliwościowe (wiele "wąskich strumyczków", np. o szerokości 100 Hz każdy). To tym strumyczkom są przydzielane bity na podstawie decyzji o ich istotności perceptualnej, podejmowanej przez model psychoakustyczny. Potem próbki z tych strumyków są normalizowane, kwantowane, ... itd. W dekoderach audio wracamy z powrotem: próbki we wszystkich podpasmach częstotliwościowych ("strumyczkach") są odtwarzane i łączone w "dużą rzekę", inną jednak od oryginału z powodu kwantowania i ewentualnego usunięcia niektórych "strumyków". Na rysunku przedstawiono schemat blokowy kodera i dekodera standardu MP2, w którym sygnał jest dzielony na 32 podpasma nie za pomocą transformacji DCT, ale z użyciem zespołu współpracujących ze sobą 32 filtrów podpasmowych, o szerokości $\frac{22050}{32} = 690$ Hz każdy. Rysunek ilustruje ogólną filozofię kompresji dźwięku: 1) analiza psychoakustyczna sygnału, 2) jego dekompozycja pod-pasmowa, 3) psycho-akustyczne, bitowe kodowanie sygnałów pod-pasmowych, 4) rekonstrukcja oryginalnego sygnału z pod-pasm.
W poniższych zadaniach użyj programu Matlaba, zaprezentowanego w listingu 13.2, implementującego koder i dekoder typu AAC-like (*Advanced Audio Coding*) sygnału audio. Przeanalizuj go bardzo uważnie, zrozum każdą linię. Następnie zmodyfikuj program, dodając do niego funcjonalności opisane w zadaniach. Rzuć okiem na dołączony w archiwum program `cps_13_audio_aac_kwant.m`, będący kontunuacją programu z listingu 13.2 — znajdziesz w nim sugestie dotyczące sposobu kwantowania sygnałów podpasmowych. Ponieważ jest wymagane przełączanie sinusoidalnych okien analizy/syntezy z krótkich (256 próbek, stosowane dla sygnałów szumowych) na długie (2048 próbek, stosowane dla sygnałów tonowych) i odwrotnie, to zobacz w programie `cps_13_audio_aac_switch.m` (dostępny w materiałach) jak to należy zrobić. Konkretnych szczegółów implementacyjnych można szukać w rozdz. 21 [40] oraz rozdz. 15 [41]

Listing 13.2: Program Matlaba implementujący quasi-AAC algorytm kodera i dekodera sygnału audio

```
% cps_13_aac.m
% Podstawy kodera audio AAC z uzyciem nakladkowej transformacji MDCT
clear all; close all;

Nmany = 100;              % liczba przetwarzanych ramek sygnalu audio
N = 2048;                 % teraz uzywamy tylko jednej dlugosci okna, sa dwie 256 i 2048
M = N/2;                  % przesuniecie okna 50%
Nx = N+M*(Nmany-1);       % liczba przetwarzanych probek sygnalu audio

% Sygnal wejsciowy
[x, fpr ] = audioread('bach44100.wav'); size(x), pause   % rzeczywisty wczytany
% fpr=44100; x=0.3*randn(Nx,1);                           % syntetyczny szum
% fpr=44100; x = 0.5*sin(2*pi*200/fpr*(0:Nx-1)');         % syntetyczny sinus
x = x(1:Nx,1);                                            % wez tylko poczatek
soundsc(x,fpr);                                           % odegraj
figure; plot(x); pause                                    % pokaz

% Macierze transformacji MDCT oraz IMDCT o wymiarach M=N/2 x N
win = sin(pi*((0:N-1)'+0.5)/N);    % pionowe okno do wycinania fragmentu sygnalu
k = 0:N/2-1; n=0:N-1;              % wiersze-czestotliwosci, kolumny-probki
C = sqrt(2/M)*cos(pi/M*(k'+1/2).*(n+1/2+M/2)); % macierz C (N/2)xN analizy MDCT
D = C'; % transpozycja                         % macierz D Nx(N/2)syntezy IMDCT

% Analiza-synteza AAC
```

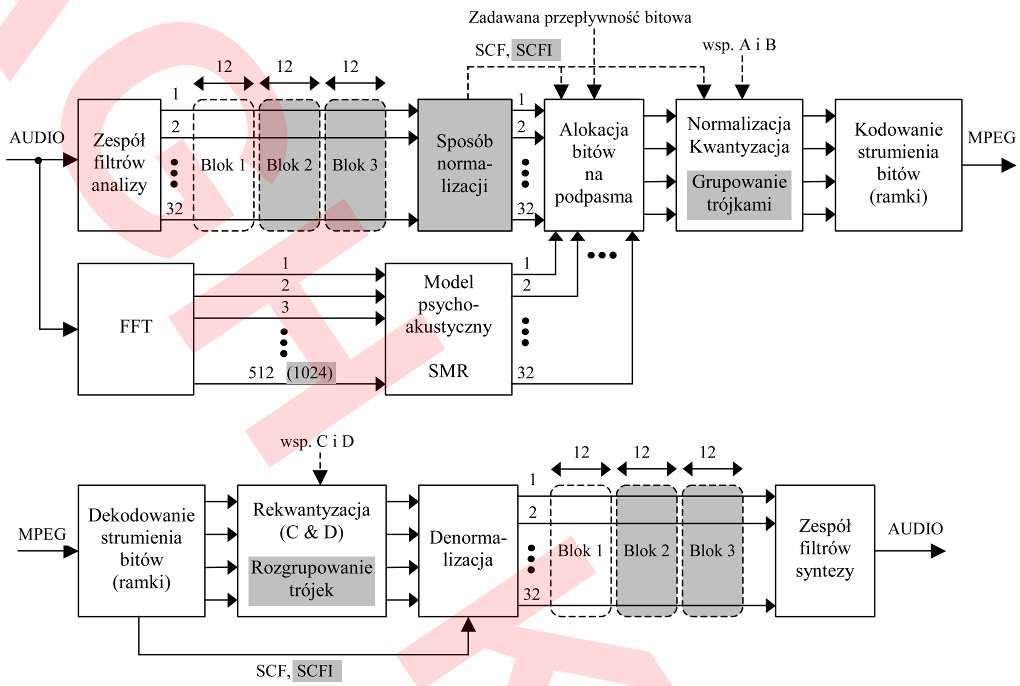

Fig. 13.2: Schemat blokowy kodera standardu MP2 (góra) oraz dekodera (dół).

```
y = zeros(Nx,1); sb = zeros(Nmany,M); figure;     % sygnal wyjsciowy
for k=1:Nmany                                      % PETLA ANALIZY SYGNALU - RAMKI AUDIO
    n = 1+(k-1)*M  : N + (k-1)*M;                  % numery probek fragmentu od-do
    bx = x( n ) .* win;                            % pobranie probek do bufora .* okno
    BX = C*bx;                                     % MDCT
  % plot(BX); title('Samples in bands'); pause     % wydruk wspolczynnikow MDCT (podpasma)
  % BX(N/4+1:N/2,1) = zeros(N/4,1);                % jakies dodatkowe przetwarzanie
    by = D*BX;                                     % IMDCT
    y( n ) = y( n ) + by.*win;                     % .* okno, dodanie probek do poprzedniej ramki
    sb(k,1:M) = BX';                               % ew. zapamietanie do pozniejszej kwantyzacji
end                                                % KONIEC PETLI
n = 1:Nx;
soundsc(y,fpr);                                     % odsluchaj
figure; plot(n,x,'ro',n,y,'bx'); title('Input (o), Output (x)'); pause % porownaj
m=M+1:Nx-M;                                        % indeksy probek
max_abs_error = max(abs(y(m)-x(m))), pause         % blad

%cps_13_audio_kwant                                 % dodatkowy program
```

Problem 13.6 (Analiza-synteza muzyki w koderze AAC - INŻYNIERIA DŹWIĘKU.).** Zakładam zrozumienie kodu programu 13.2. W koderze typu AAC, wejściowy sygnał audio jest dzielony na nakładające się w 50% bloki próbek o długości 256 (fragmenty szumowe) lub 2048 (fragmenty tonalne). Wykorzystywane jest w tym celu przesuwane okno sinusoidalne (połówka sinusoidy) o zmiennej długości. Po tej operacji na każdym bloku jest wykonywana zmodyfikowana transformacja kosinusowa MDCT (256 lub 2048-punktowa), a pod-pasmowe współczynniki MDCT są zapisywane do macierzy roboczej `sb(:,:)`. Następnie współczynniki MDCT są lokalnie normalizowane w różny sposób w poszczególnych pod-pasmach (nie zaimplementowane), localnie kwantowane (nie zaimplementowane) oraz kodowane bitowo. Rekonstrukcja sygnału jest przeprowadzana w odwrotnej kolejności.
Zapoznaj się z programem 13.2. Uruchom go. Sprawdź czy sygnał jest zawsze perfekcyjnie odtworzony, niezależnie od jego rodzaju. Przetesuj różne sygnały syntetyczne (fala prostokątna, fala trójkątna, przełączany szum-sinusoida-szum-sinusoida) oraz trzy różne wymagające nagrania muzyczne). Sprawdź różne wartości `N=2048, 1024, 512, 256, 128, 64, 32`. Sprawdź czy można użyć innego okna zamiast sinusoidalnego. Na koniec dodaj jakieś

przetwarzanie sygnałów w podpasmach: wzmocnij część z nich, część stłum, inną część usuń. Zabaw się w Pana Inżyniera Dźwieku, guru w studiu nagrań Universal Studio Polska.

Problem 13.7 (* Analiza-modyfikacja-synteza dźwięku z użyciem MDCT - SEPARACJA DŹWIĘKÓW.).** Dysponując macierzą współczynników MDCT, obliczonych dla kolejnych fragmentów sygnału i zmieniających się w czasie, możemy wyświetlicz tę macierz (wartości bezwzględne) podobnie jak jest to robione w przypadku spectrogramu sygnału (zobacz rysunek wygenerowany przez funkcję Matlaba `spectrogram()` dla naszego sygnału audio). Widząc czasowo-częstotliwościowy (TF) opis sygnału, możemy w nim zauważyć charakterystyczne wzorce TF (TF *patterns*), np. skoki częstotliwościowe (hip-hop) podczas śpiewu kanarka na tle zgiełku ruchliwej ulicy lub pracującego silnika samochodu. 1) pozostawiając tylko współczynniki MDCT związane z konktetnym źródłem dźwieku, 2) zerując pozostałe i 3) wykonując syntezę sygnału audio, możemy dokonać filtracji sygnału w dziedzinie TF. Dla sygnałów "skaczących po częstotliwościach", jak kanarek, filtracja taka może być skuteczniejsza niż standardowa filtracja FIR/IIR, w której filtry nie zmieniają swoich charakterystyk częstotliwościowych w czasie. Opisana metoda filtracji jest wykonywana w standardowej strukturze analiza-modyfikacja-synteza (sygnał-widmo-modyfikacja-sygnał), ale na szachownicy TF współczynników transformacji MDCT, a nie na pojedynczym widmie całego sygnału (np. FFT(DCT) - modyfikacja widma - IFFT(IDCT)). Dzięki temu możliwe jest lepsze dopasowanie zero-jedynkowej maski widmowej do skoków częstotliwości (hip-hop) sygnałów składowych w osi czasu. Nagraj dźwięk o zmieniających się częstotliwościach lub pobierz z internetu i zmiksuj razem kilka sygnałów o "skaczących" częstotliwościach. Następnie zastosuj program 13.2 kodera-dekodera AAC do separacji poszczególnych dźwięków drogą maskowania 0/1 współczynnków MDCT i syntezy z nich sygnału audio.

Problem 13.8 ((**) KWANTYZACJA sygnałów podpasmowych w koderze AAC).** W koderach dźwięku stosuje się modele psychoakustyczne do obliczenia optymalnej alokacji bitów dla poszczególnych sygnałów podpasmowych. To bardzo poprawia efektywność kompresji sygnałów (zapewnia małą liczbę bitów gwarantujących dobrą jakość odtworzenia oryginalnego dźwięku). Ale dużo daje już samo: 1) wyznaczenie lokalnej wartości maksymalnej w każdym podpaśmie, 2) podzielenie przez nią wszystkich próbek danego podpasma (czyli ich unormowanie), 3) kwantowanie tak przygotowanych sygnałów. Otrzymue się bowiem mniejszy szum kwantowania niż w przypadku podobnie "agresywnego" (liczba przydzielonych bitów) kwantowania oryginalnych próbek sygnału. Zapoznaj się z programem `cps_13_kwant.m`, dołączonym do materiałów, który wykonuje właśnie takie kwantowanie. Powinien on zostać uruchowiony na końcu programu `cps_13_aac.m`, który mu przekazuje wszystkie obliczone próbki podpasmowe. Przetestuj różny podział współczynników MDCT na podpasma częstotliwościowe oraz różny przydział bitów na te podpasma: mniej lub więcej w ogóle, o ile wiecej na niskie częstotliwości, czy w ogóle przydzielać bity dla najwyższych częstotliwości? Sprawdź wypracowaną koncepcję na 2-3 innych utworach muzycznych.
Spróbuj zaproponować jakąś metodę alokacji bitów na pod-pasma częstotliwościowe, zapewniającą ustalony stopień kompresji sygnału wybrany przez użytkownika, np. 64, 128, 256, 384 kbity na sekundę. Możesz także pomyśleć nad zastosowaniem jakiejś metody kompresji bezstratnej w stosunku do skwantowanych współczynników MDCT - np. kodera Huffmana lub arytmetycznego (w obu przypadkach otrzymasz dodatkowe **).

Problem 13.9 (* Kompresja audio AAC - automatyczne przełączanie okien.).** Dodaj do programu 13.2 procedurę przełączania okien, zaimplementowaną w programie `cps_13_audio_aac_switch.m`, dostępnym w materiałach. Następnie, opracuj algorytm automatycznej detekcji charakteru sygnału (składowe szumowe czy tonowe) i automatycznie przełączaj długość okna: 256 dla szumu oraz 2048 dla tonów. W tym celu możesz skorzystać z definici indeksu tonalności, płaskości widmowej, entropii, entropii perceptualnej, lub jakiejś innej miary "szumowości" lub "tonalności" sygnału (patrz rozdz. 19 w [40] albo rozdz. 14 w [41]). Utwórz specjalne, "sklejone" zbiory audio (po kolei: [szum], [tony], [szum], [tony], ... itd.) do testowania opracowanego algorytmu. Dodatkowo, jeśli dalej czujesz *power*, to zaproponuj jakieś pre-definiowane wektory dzielników, wykorzystywanych podczas kwantyzacji, zapewniające wybrane przez użytkownika dwie przepływności bitowe, jedną małą i drugą dużą, np. 64, 96, 128, 192, 256, 384 kbity na sekundę. Możesz także pomyśleć o zastosowaniu na samym końcu łańcucha operacji przetwarzania, do już otrzymanego strumienia bitów, jakiejś metody kodowania bezstratnego. Widzisz, tak wiele możesz!

TEMAT #3: Obrazy. Obrazy to sygnały dwu-wymiarowe (2D). Dla nich funkcje jedno-wymiarowe (1D) przetwarzania sygnałów (takie jak `fft(),dct(),conv()`) są zastępowane przez funkcje 2D (takie jak `fft2(),dct2(),conv2()`):

- dyskretne, ortogonalne transformacje 2D są zdefinowane jako sekwencje transformacji 1D: najpierw względem każdego wiersza obrazu, potem względem każdej kolumny macierzy, otrzymanej w wyniku pierwszej operacji;
- filtracja 2D obrazów w dziedzinie częstotliwości jest wykonywana w następujący sposób: jest obliczane widmo 2D DFT/DCT, następnie jest ono modyfikowane (wymnażane z maską 0/1 lub inną) oraz transformowane 2D z powrotem do dziedziny pikseli obrazu;
- filtracja 2D obrazów w dziedzinie pikseli obrazów polega na obliczeniu lokalnej średniej ważonej każdego piksela na obrazie; zmieniając wagi filtra 2D, otaczające piksel którym się interesujemy, możemy wykonać filtrację: dolno-, górno-, pasmowo-przepustową oraz pasmowo-zaporową, ale także różniczkowanie obrazu w jednym lub dwóch kierunkach (należy teraz rzucić okiem do tabel 22.3 i 22.4 w [40] albo do programu `filterweights.m`, zawierających przykładowe wagi filtrów!)

Uruchom program 13.3. Rzuć okiem na jego kod.

Listing 13.3: Program Matlaba implementujący podstawowe operacje analizy i przetwarzania obrazów

```
% cps_13_obraz.m
% Podstawy przetwarzania obrazow - wymagany Image Processing Toolbox
% Znajdz obrazy demo Matlaba: C:\Program Files\MATLAB\Rxxxxab\toolbox\images\imdata
% Dolacz obrazy Lena.BMP oraz Friends.JPG/Friends.PNG z wykladu
clear all; close all;

% Wczytaj obraz, przeksztalc kolory ######################################
% Inicjalizacja - wczytaj oraz (x) oraz palete kolorow (cmap), jesli kolory sa indeksowane
  [x,cmap] = imread('bike.jpg');       % color (gray-scale), RGB MxNx3, cmap=empty
% [x,cmap] = imread('cameraman.tif'); % BW, MxNx1, cmap=brak
% [x,cmap] = imread('corn.tif');       % color, MxNx1, size(cmap)=256x3=equal=[0,1]?
% [x,cmap] = imread('kids.tif');       % color, MxNx1, size(cmap)=256x3=equal=[0,1]?
% [x,cmap] = imread('trees.tif');      % color, MxNx1, size(cmap)=256x3=equal=[0,1]?
% [x,cmap] = imread('peppers.png');    % color, RGB MxNx3, cmap=brak
  [x,cmap] = imread('hands1.jpg');     % color, RGB MxNx3, cmap=brak
% [x,cmap] = imread('office_4.jpg');   % color, RGB MxNx3, cmap=brak
% [x,cmap] = imread('Lena2.bmp');      % BW, MxNx1, size(cmap)=256x3=equal=[0,1]

disp('Na wejsciu:');
pal = size(cmap), [M, N, K] = size(x), % wymiary palety, M-wierszy, N-kolumn, K=skladowych koloru
xmin = min(min(x)), xmax = max(max(x)),              % wartosci pikseli min, max
x_copy = x;                                          % skopiuj obraz
cmap_copy = cmap;                                    % skopiuj palete kolorow
    figure; imshow(x,cmap), title('Obraz'); pause    % wyswietl obraz uzywajac jego palety

if( ~isempty(cmap) & K==1 ) x=ind2gray(x,cmap); cmap = []; end   % dla indeksow kolorow (np. TIF)
if(  isempty(cmap) & K==3 ) % dla kolorow RGB - brak cmap
    if(1)  x = rgb2gray(x); % 3 skladowe kolory RGB --> indeks koloru, paleta szara
    else   x = x(:,:,2);    % wybierz tylko jedna plaszczyzne koloru: 1=R, 2=G, 3=B
    end
end
disp('Na wyjsciu:'); pal=size(cmap), [M, N, K]=size(x), % M-wierszy, N-kolumn, K-skladowych koloru
    figure; imshow(x,cmap), title('Obraz'); pause      % wyswietl obraz uzywajac jego palety
    disp('Prosze, obroc macierz obrazu!');              % wyswietl obraz jako siatke
    figure; mesh(x); title('Image'); pause              % oraz obroc go recznie

% 2D-DCT OBRAZU ORAZ FILTRACJA WYKORZYSTUJACA 2D DCT ##################################
x = double( x );                               % wartosci uint8 na double
xmin = min(min(x)), xmax = max(max(x)), pause  % min/max = ?
[m,n] = meshgrid(1:N,1:M); MN=max(M,N);        % indeksy maski filtra m,n
H(1:M,1:N) = exp(-(m.^2+n.^2)/(0.075*MN)^2);   % wartosci maski filtra LP w dziedzinie 2D-DCT
%H = ones(M,N) - H;                            % wartosci maski filtra HP w dziedzinie 2D-DCT
X1 = dct2( x ); %<============================ % 2D-DCT Matlaba - analiza obrazu
X2 = dct( dct( x ).' ).';                      % 2D-DCT jako sekwencja 1D-DCT
err_dct = max(max(abs(X1-X2))),                % blad
X = X1; Xmin = min(min(X)), Xmax = max(max(X)), pause   % min/max widma 2D-DCT
```

```
X = X.*H;  % <========================== % modyfikacja widma 2D-DCT obrazu
y = idct2( X ); % <======================= % 2D-IDCT Matlaba - synteza obrazu
    figure;
    subplot(221); imshow(x, cmap);          title('Obraz');
    subplot(222); imshow(scaledB(X), cmap); title('2D DCT');
    subplot(223); imshow(scaledB(H), cmap); title('Maska filtra');
    subplot(224); imshow(y, cmap);          title('Obraz po filtracji'); pause

% FILTRACJA 2D Z UZYCIEM SPLOTU 2D  #####################################
% filry dolno-przepustowe:                                    hLP1,...,hLP4,
% filtry krawedziowe -|\/ (Sobel, Prewitt, Roberts, Gradient): hS1,hS2,hP1,hP2,hR1,hR2,hG1,hG2
% filtry podwojnie rozniczkujace (Laplasjany):                 hL1,...,hL6
filterweights                % dolacz zbior (z wykladu) z wagami roznych filtrow 2D
y = conv2(x, hLP4,'same');   % filtr dolno-przepustowy
    figure;
    subplot(121); imshow(x,cmap); title('PRZED filtracja');
    subplot(122); imshow(y,cmap); title('PO filtracji'); pause
y1 = conv2(y, hS1,'same');   % filtr Sobel1, Prewitt 1 ==, Roberts 1 //
y2 = conv2(y, hS2,'same');   % filtr Sobel2, Prewitt 2 ||, Roberts 2 \\
y12 = sqrt(y1.^2 + y2.^2);   % dodaj dwa obrazy = krawedzie we wszystkich kierunkach
    figure;
    subplot(221); imshow(x,cmap);   title('PRZED filtracja');
    subplot(222); imshow(y1,cmap);  title('PO filtracji == lub //');
    subplot(223); imshow(y12,cmap); title('PO filtracji ==|| lub \\//');
    subplot(224); imshow(y2,cmap);  title('PO filtracji || lub \\'); pause

% BINARYZACJA, PRZETWARZANIE MORFOLOGICZNE #################################
% z = imbinarize( y12 );                           % funkcja Matlaba #1
% level = graythresh(y12); z = im2bw(y12,level);  % funkcja Matlaba #2
[ixy] = find( y12 > 100 );                         % # nasza binaryzacja
z = zeros(M,N); z(ixy) = 255*ones(size(ixy));     % #
    figure; subplot(131); imshow(z,cmap); title('PO binaryzacji');
z = imerode( z,  [ 1 1 1; 1 0 1; 1 1 1] );        % erozja (zawezenie)
    subplot(132); imshow(z,cmap); title('PO erozji');
z = imdilate( z, [ 1 1 1; 1 0 1; 1 1 1] );        % dylatacja (rozszerzenie)
    subplot(133); imshow(z,cmap); title('PO dylatacji'); pause

% Image Studio - miksowanie obrazow ###########################################
% Ten fragment jest tylko poprawny dla obrazow RGB, np. JPG/PNG, np. "handl.jpg"
figure; hist(x(:),50); title('Histogram'); pause
THR = 155; % prog poprawny dla obrazu handl.jpg, ustaw go dla innych obrazow
if( 0 )    % jeden wybrany kolor tla, obecnie zolty R=1, G=1, B=0
   y(:,:,1) = 255*ones(M,N);  % Red
   y(:,:,2) = 255*ones(M,N);  % Green
   y(:,:,3) =   0*ones(M,N);  % Blue
   y = uint8( y );
else       % drugi obraz jest wykorzystywany jako tlo dla fragmentu naszego obrazu
  [y,cmap] = imread('Friends.png'); y = y(end-M+1:end,1:N,:);
end
for m=1:M
    for n=1:N
        if( x(m,n) < THR ) y(m,n,:) =  x_copy(m,n,:); end
    end
end
    figure; imshow(y); title('Wynik miksowania obrazow'); pause

%#######################
function XdB = scaledB(X)
% skalowanie logarytmiczne wartosci pikseli obrazu
XdB = log10(abs(X)+1); maxXdB = max(max(XdB)); minXdB = min(min(XdB));
XdB = (XdB-minXdB)/(maxXdB-minXdB)*255;
end
```

Problem 13.10 (*//***/****/***** Obrazkowy elementarz).** Przeanalizuj kod programu 13.3. Uruchom go. Wybierz do przetwarzania różne obrazy z Matlab Image Toolbox (jest ich bardzo dużo). Zwróć uwagę na wyniki różnych części programu. Wczytaj do programu zdjęcie wykonane przez ciebie. Rozwiń kod programu w dowolnej części. Sam sformułuj cele przetwarzania/detekcji.

Problem 13.11 (* Image Studio: Foto-mikser).** Wykonaj zdjęcie sobie (*selfie*) oraz jakiemuś krajobrazowi. Zastosuj dowolne metody przetwarzania obrazów: (filtrację dono/górno-przepustową, filtrację różniczkującą wyostrzającą krawędzie obiektów, filtrację nieliniową (np. medianową) usuwającą szum impulsowy, binaryzację, filtrację morfologiczną (erozję, dylatację, zamknięcie, otwarcie), oraz wydziel twoją postać/twarz z pierwszego zdjecia. Wykonaj rozmycie (efekt Bokeh) drugiego zdjęcia z krajobrazem za pomocą: 1) filtracji dolno-przepustowej w dziedzinie pikseli obrazu (z użyciem `conv2()` - zastosuj wagi 2D filtra zdefiniowane w`filterweights.m`)), oraz 2) modyfikacji współczynników 2D transformacji DCT, czyli w dziedzinie częstotliwościowej (`dtc2()+modyfikacja+idct2()`). W tym drugim przypadku skorzystaj z gotowych funkcji 2D DCT Matlaba oraz sam wykonaj sekwencję transformacji 1D DCT. Nałóż twoją postać/twarz na rozmyty widoczek. Wykonaj cyfrowe powiększenie tła za pomocą funkcji interpolacji 2D (`interp2()`).

Problem 13.12 (* Znaki wodne w obrazach).** Przeczytaj bardzo krótki podrozdział 22.5.2 w [40], poświęcony dodawaniu znaków wodnych do obrazów. Napisz program w języku Matlab, implementujący dowolny inny algorytm niż ten opisany. Na przykład, dodający znak wodny w dziedzinie transformacji DCT. Sprawdzić działanie algorytmu: czy znak modny jest poprawnie dekodowany? Dokonaj "ataku" na obraz i zmodyfikuj jego wybrany fragment. Czy znak wodny został uszkodzony? Czy mozesz stwierdzić gdzie nastąpił atak? Czy mógłbyś w obrazie "ukryć" imię i nazwisko autora obrazu?

Problem 13.13 (** Kompresja obrazów: quasi-JPEG pikselo-łamacz/zgniatacz).** W podrozdziale 22.5.1 w [40], został opisany krok-po-kroku algorytm kompresji pojedynczych obrazów/zdjęć JPEG (podział obrazu na kwadraty pikseli 8x8, wykonanie transformacji 2D 8x8 DCT, kwantyzacja, skanowanie zig-zag, koder VLI, koder Huffmana) - ale żaden program nie został załączony. Napisz program kodera i dekodera standardu JPEG, implementujący go w dowolnym języku: C, Java, Python lub Matlab. Zaprogramuj łańcuch operacji przetwarzania w przód i w tył tak długi jak chcesz (jak cię to jeszcze bawi). Zrób sobie zdjęcie oraz je zakoduj i rozkoduj. Przetestuj program na zdjęciu czarno-białym i kolorowym.

Problem 13.14 (** Rozpoznawanie drukowanych liter (OCR)).** Zrób zdjęcie kilku słów wydrukowanych na kartce papieru: 1) bez zniekształceń, 2) lekko obróconych kątowo i przesuniętych, 3) pomniejszonych - widzianych z większej odległości, 4) z przekrzywieniem perspektywicznym, 5) z bliska - z "efektem rybiego oka". Wczytaj obraz do Matlaba i spróbuj napisać program, który automatycznie rozpozna wszystkie litery i słowa. Pierwszych sugestii rozwiązania problemu poszuj w podrozdziale 22.5.3 w [40].

Problem 13.15 ((*)(*) Łyżwiarstowo figurowe - program dowolny).** Zrób z obrazami to co Ci w duszy gra! Nie musi to być symfonia, ale, proszę, niech to nie będzie pojedynczy akord. Rozpoznawanie kilku figur geometrycznych, liczby kół pojazdu (rower/motocykl, samochód, ciężarówka/autobus), liczby osób stojących w kolejce, liczba samochodów na ulicy, cyfr na tablicy rejestracyjnej samochodu, numeru autobusu lub tramwaju, liczby wyprostowanych palców u ręki? Cokolwiek interesującego dla Ciebie.

Literatura

1. T.P. Barnwell, K. Nayebi, and C.H. Richardson, *Speech Coding: A Computer Laboratory Textbook*. New York, Wiley, 1996.
2. C.S. Burrus C.S., and T.W. Parks, *DFT/FFT and Convolution Algorithms. Theory and Implementation*. New York, Wiley 1985.
3. C.S. Burrus, J.H. McCellan, A.V. Oppenheim, T.W. Parks, R.W. Schafer, H.W. Schuessler, *Computer-Based Exercises for Signal Processing Using MATLAB*. Englewood Cliffs, Prentice Hall, 1994.
4. T.F. Collins, R. Getz, D. Pu, and A.M. Wyglinski, *Software Defined Radio for Engineers*. Analog Devices - Artech House, 2018.
5. M.E. Frerking, *Digital Signal Processing in Communication Systems*. New York, Springer, 1994.
6. R.C. Gonzales, R.E. Woods, and S.L. Eddins, *Digital Image Processing Using Matlab*. Upper Saddle River, Pearson Prentice Hall, 2004.
7. M.H. Hayes, *Schaum's Outline of Theory and Problems of Digital Signal Processing*. McGraw-Hill, 1999, 2011.
8. E.C. Ifeachor, and B.W. Jervis, *Digital Signal Processing. A Practical Approach*. Wokingham, Addison-Wesley 1993.
9. **V.K. Ingle, J,G. Proakis, *Digital Signal Processing Using Matlab*. Boston, PWS Publishing, 1997, CL Engineering 2011.**
10. **J. Izydorczyk, G. Płonka, i G. Tyma, *Teoria sygnałów. Kompendium wiedzy na temat sygnałów i metod ich przetwarzania*. Gliwice, Helion, 2006.**
11. A.K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, Prentice Hall 1989.
12. C.R. Johnson Jr, W.A. Sethares, and A.G. Klein, *Software Receiver Design. Build Your Own Digital Communication System in Five Easy Steps*. Cambridge, Cambridge University Press, 2011.
13. S.M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Englewood Cliffs, PTR Prentice Hall, 1993.
14. R.G. Lyons, *Understanding Digital Signal Processing*. Boston, Addison-Wesley Longman Publishing, 1996, 2005, 2010.
15. **R.G. Lyons, *Wprowadzenie do cyfrowego przetwarzania sygnałów*. Warszawa, WKŁ, 1999, 2010.**
16. **MathWorks, *Matlab Signal Processing Toolbox Documentation***.
    ON-LINE: *https://www.mathworks.com/help/signal/*
    MathWorks, *Matlab Signal Processing Toolbox Getting Started and User's Guide*.
    PDF ZA DARMO: *https://www.mathworks.com/help/pdf_doc/signal/index.html*
17. D.G. Manolakis, and V.K. Ingle, *Applied Digital Signal Processing*. Cambridge, Cambridge University Press, 2011.
18. **J.H. McClellan, R.W. Schafer, and M.A. Yoder, *DSP FIRST: A Multimedia Approach*. Prentice Hall, 1998, 2015.**
19. S.K. Mitra, *Digital Signal Processing. A Computer-Based Approach*. New York, McGraw-Hill, 1998.
20. S.K. Mitra, *Digital Signal Processing Laboratory Using Matlab*. Boston, McGraw-Hill, 1999.
21. A.V. Oppenheim, R.W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, Prentice Hall, 1989.
22. A.V. Oppenheim, A.S. Willsky A.S., and S.H. Nawab, *Signals & Systems*. Upper Saddle River, Prentice Hall, 1997, 2006.
23. P.E. Papamichalis, *Practical Approaches to Speech Coding*. Englewood Cliffs, Prentice Hall, 1987.
24. T.W. Parks, and C.S. Burrus, *Digital Filter Design*. New York, Wiley, 1987.
25. W.K. Pratt, *Digital Image Processing*. New York, Wiley, 2001.
26. J.G. Proakis, and D.G. Manolakis, *Digital Signal Processing. Principles, Algorithms, and Applications*. New York, Macmillan, 1992.
27. T.F. Quatieri, *Discrete-Time Speech Signal Processing*. Upper Saddle River, Prentice Hall, 2001.
28. L.R. Rabiner, and R.W. Shafer, *Digital Processing of Speech Signals*. Prentice Hall 1978.
29. M. Rice, *Digital Communications: A Discrete-Time Approach*. Upper Saddle River, Pearson Education, 2009.
30. K. Shin, and J.K. Hammond, *Foundamentals of Signal Processing for Sound and Vibration Engineers*. Chichester, John Wiley & Sons, 2007.
31. **S.W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, California Technical Publishing, 1997, 1999.**
    PDF ZA DARMO: *https://www.analog.com/en/education/education-library/scientist_engineers_guide.html*
32. **S.W. Smith, *Cyfrowe przetwarzanie sygnałów. Praktyczny poradnik dla inżynierów i naukowców*. Warszawa, BTC, 2007.**
33. A. Spanias, T. Painter, and V. Atti, *Audio Signal Processing and Coding*. Hoboken, Wiley-Interscience, 2007.
34. **K. Steiglitz, *A Digital Signal Processing Primer: With Applications to Digital Audio and Computer Music*. Pearson, 1996.**
35. **L. Tan, J. Jiang, *Digital Signal Processing. Fundamentals and Applications*. Cambridge MA, Academic Press, 2019 (3e).**
36. C.W. Therrien, *Discrete Random Signals and Statistical Signal Processing*. Englewood Cliffs, Prentice Hall, 1992.
37. S.A. Tretter, *Communication System Design Using DSP Algorithms*. New York, Springer Science+Business Media, 2008.
38. P. Vary, and R. Martin, *Digital Speech Transmission. Enhancement, Coding and Error Concealment*. Chichester, Wiley, 2006.
39. T.P. Zielinski, *Od teorii do cyfrowego przetwarzania sygnałów*. Kraków, AGH, 2003, 2004.
    Programy: *http://www.kmet.agh.edu.pl/dydaktyka/wydawnictwa/*
40. **T.P. Zielinski, *Cyfrowe przetwarzanie sygnałów. Od teorii do zastosowań*. Warszawa, WKŁ, 2005 ... 2014.**
    Programy: *http://www.kmet.agh.edu.pl/dydaktyka/wydawnictwa/*
41. **T.P. Zielinski, *Starting Digital Signal Processing in Telecommunication Engineering. A Laboratory-based Course*. Springer, 2021.**
    Programy: *http://kt.agh.edu.pl/~tzielin/books/DSPforTelecomm/*
42. **T.P. Zielinski, P. Korohoda, R. Rumian (red.), *Cyfrowe przetwarzanie sygnałów w telekomunikacji. Podstawy. Multimedia. Transmisja*. Warszawa,**
    Programy: *http://teledsp.kt.agh.edu.pl/*
43. U. Zölzer (ed), *DAFX Digital Audio Effects*. Chichester, John Wiley & Sons, 2002, 2011.