Wyższa Szkoła Bankowa
Wydział Finansów i Zarządzania
Informatyka Inżynierska
Studia I stopnia, semestr 4
Gdańsk

Podstawy przetwarzania sygnałów Laboratorium

v0.4.2

Autor:
Jakub Głuszek

11 lutego 2022

Spis treści

1 Laboratorium nr 1

1.1 Wprowadzenie do programu Matlab/GNU Octave

Zarówno MATLAB jak i GNU Octave są programami przeznaczonymi do numerycznych obliczeń macierzowych. Posiadają wiele funkcji bibliotecznych i narzędzi wizualizacyjnych, które sprawiają, że owe programy idealnie się sprawdzają w zastosowaniach takich jak: statystyka, przetwarzanie sygnałów, teoria sterowania, czy uczenie maszynowe i big data. Programy te posiada rozbudowane funkcje i wiele toolbox'ów, które sprawiają, że są przydatne w dużej liczbie dziedzin inżynieryjnych, ekonomicznych czy statystycznych. Wszystkie poznane na tych zajęciach zagadnienia powinny być również łatwe do implementacji w innych programach np. w Pythonie.

Podczas pierwszych zajęć laboratoryjnych studenci powinni zaznajomić się, z podstawowymi poleceniami, funkcjami programu MATLAB/Octave. Jest to niezbędne, do późniejszego zgłębiania zagadnień związanych z cyfrowym przetwarzaniem sygnałów.

1.1.1 Podstawowe polecenia i funkcje

Pierwotnie MATLAB, był głównie używany do wykonywania operacji macierzowych. Aby utworzyć nową macierz i przypisać ja do zmiennej należy wpisać w oknie `Command Windows` polecenie:

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

Listing 1: Tworzenie macierzy będącej kwadratem magicznym

Średnikiem oddzielane są poszczególne wiersze macierzy. Po wpisaniu powyższej macierzy dalej mamy możliwość wykorzystywanie tej zmiennej, jest ona zachowana w przestrzeni roboczej programu MATLAB/Octave. Jeśli wyrażeniu nie przypiszemy wartości do konkretnej liczby to program przypisze ją do zmiennej `ans`, aby zachować wynik danego wyrażenia.

Na tak stworzonej macierzy można wykonywać operacje sumowania, transpozycji czy wyznaczenia wartości znajdujących się na przekątnej, odwracania macierzy i obliczania wyznacznika.

```
>> sum(A) % sumowanie wartosci z kolejnych kolumn
>> A' % transpozycja macierzy
>> diag(A) % wartosci znajdujace sie na przekatnej
>> inv(A) % odwracanie macierzy
>> det(A) % oblicza wyznacznik z macierzy
```

Listing 2: Przykładowe funkcje przydatne podczas pracy z macierzami

Często konieczne jest odwołanie się do konkretnego zakresu elementów macierzy, bądź pojedynczej wartości. Pobieranie tych danych zrealizowane jest przez podanie odpowiednich zakresów w nawiasach przy nazwie utworzonej macierzy. **W programach MATLAB/GNU Octave indeksy tablic są numerowane liczby jeden!**

```
>> A(1,2) % wypisuje wartosc elementu znajdujacego sie w pierwszym wierszu
    i drugiej kolumnie
>> A(:,2) % wypisuje wartosci elementow znajdujacych sie w drugiej
   kolumnie
>> A(1:3,2) % wypisuje wartosci elementow znajdujacych sie w drugiej
   kolumnie i pierwszych trzech wierszach
```

Listing 3: Odwoływanie się do elementów macierzy

W wielu sytuacjach konieczne okazuje się tworzenie wektora z użyciem operatora dwukropka, albo przy użyciu funkcji `zeros`, `ones` czy `linspace`.

```
>> B = 1:10 % tworzy wektor B skladajcy sie z wartosci od jeden do
   dziesiec inkrementowanych z krokiem jeden
>> B = 1:0.1:10 % tworzy wektor jak wyzej jednak krok w tym przypadku
   wynosi 0.1
>> B = 2.^[1:4] % tworzy wektor ciagu geometrycznego o ilorazie 2
>> B = zeros(2,4) % tworzy wektor 2x4 skladajacy sie z samych zer
>> B = ones(3,4) % tworzy wektor 3x4 skladajacy sie z samych jedynek
```

```
>> B = linspace(3,42) % tworzy wektor 100 elementow ronomiernie
   rozlozonych pmiedzy wartosciami 3 i 42
>> B = linspace(3,42,50) % tworzy wektor 50 elementow roznomiernie
   rozlozonych pomierzy wartosciami 3 i 42
```

Listing 4: Tworzenie macierzy z wykorzystaniem funkcji oraz operatora :

Na macierzach w programie MATLAB/Octave możliwe jest wykonywanie operacji arytmetycznych zgodnie z zasadami wykonywania działań na macierzach.

```
>> C = [1 2 3]+[1 2 3]
>> C = [1 2 3]*[4; 5; 6]
>> C = [1 2 3]'*[4 5 6]
>> C = [1 2 3]^2 % [1 2 3]*[1 2 3]
>> C = [1 2 3].^2 % operator ^ zapisany z poprzedzajaca go kropka powoduje
    wykonywanie operacji potegowania osobno dla kazdego elemetu
>> C = [1 2 3].*[1 2 3] % wykonuje operacje mnozenia element po elemencie
```

Listing 5: Operacje arytmetyczne na macierzach

W omawianych programach można korzystać z wielu gotowych stałych i funkcji.

```
>> pi % 3.141592...
>> i~% reprezentuje wartosc urojona
>> j % jak wyzej
>> [2 + 2i]*[3 + 3i] % (6 + 6i + 6i + 6i^2) = (0 + 12i)
>> Inf % nieskoczonosc
>> NaN % not a number
>> sqrt(256) % oblicza pierwiastek
>> abs(-12) % obliczna wartosc bezwglednia
>> exp(7) % eksponent z liczby siedem e^7
>> sin(pi/2) % wartosc pi dla argumetu pi/2
>> log10(100) % operacje obliczania logarytmu dziesietnego
>> mean([1 2 3 4 5 6]) % oblicza wartosc oczekiwana z wektora liczb
>> var([4 5 6 5 4 6]) % oblicza wariancje z wektora liczb
>> min([1 2 3 5 -50])
>> max([1 2 3 5 -50])
```

Listing 6: Przykładowe stałe i funkcje

Aby dodać do macierzy albo z niej usunąć konkretną kolumnę bądź wiersz można posłużyć się operatorem `[]`:

```
>> X = [1 2 3; 4 5 6; 7 8 9]
>> X(:,2) = [] % usuwa druga kolumne z macierzy
>> X = [X; [10 11 12]] % dodaje wiersz do macierzy
```

Listing 7: Usuwanie i dodawanie wierszy/kolumn do macierzy

Funkcja `find` pozwala na znalezienie elementów macierzy, które spełniają wyrażenie logiczne. `find` zwraca indeksy pod którymi przechowywane są te elementy.

```
>> X = [1 2 3; 4 NaN 6; 7 8 NaN]
>> find(isnan(X))
>> find(X>3)
```

Listing 8: Szukanie elementów macierzy spełniających warunek logiczny

1.1.2 Tworzenie skryptów i funkcji

Zamiast wpisywać kolejne komendy w wierszu poleceń, MATLAB/Octave umożliwia tworzenie tzw. m-plików, które są skryptami zawierającymi szereg kolejno wykonujących się poleceń czy wywołań funkcji. Jeśli chcemy uruchomić skrypt z poziomu wiersza poleceń wystarczy napisać jego nazwę i zatwierdzić enterem (jeśli

aktualnie nad danym skryptem pracujemy, jest on uruchomiony w oknie programu `Editor`, można również go uruchomić za pomocą przycisku F5).

Analogicznie MATLAB/Octave pozwala na pisanie własnych funkcji, które później mogą być wywoływane z określonymi argumentami i mogą zwracać jedną bądź wiele wartości. Definicja funkcji wygląda w następujący sposób:

```
function [y1,...,yN] = myfun(x1,...,xM)
```

Listing 9: Prototyp funkcji w programie MATLAB/Octave

Ze względu na tzw. duck typing, nie ma konieczności deklarowania typu przekazywanej/zwracanej wartości.

1.1.3 Rysowanie wykresów

MATLAB/Octave posiada bogate możliwości wizualizacji danych, z których podczas zajęć laboratoryjnych będziemy korzystać regularnie. Podstawową funkcją, która odpowiada za rysowanie wykresów jest `plot`. Pierwszym argumentem przekazywanym do tej funkcji jest wektor zawierający liczby znajdujące się osi X, natomiast drugim jest wektor zawierający odpowiadające im wartości (oś Y).

```
x = 0:pi/100:2*pi; % tworzenie wektora czasu
y1 = sin(x); % oblicza wartosci funkcji sin dla wektora czasu x
y2 = sin(x-0.25);
y3 = sin(x-0.5);

figure % tworzy nowy uchwyt na wykres
plot(x,y1,x,y2,'--',x,y3,':') % narysuj na jednym wykresie trzy przebiegi funkcji sinus

figure % utworz nowy uchyt na wykres
plot(x, y1) % narysuj pierwsza funkcje y1
hold on % zachowaj wczesniej narysowany wykres
plot(x, y2) % narysuj wykres y2 obok wczenisjeszgeo y1
plot(x, y3) % dodaj trzeci wykres y3
```

Listing 10: Rysowanie funkcji *sin* na jednym wykresie

Czasami może okazać się konieczne podzielenie jednego rysunku na kilka mniejszych rozmieszczonych w określonej kolejności. Pomocna w takim przypadku okazuje się funkcja `subplot`:

```
subplot(2,2,1) % narysowany ponizsza funkcja plot wykres znajdzie sie na pierwszych miejscu na podzielonym na cztery obszary (2x2) rysunku
x = linspace(0,10);
y1 = sin(x);
plot(x,y1)
title('Subplot 1: sin(x)')

subplot(2,2,2) % narysowany ponizsza funkcja plot wykres znajdzie sie na drugim miejscu na podzielonym na cztery obszary (2x2) rysunku
y2 = sin(2*x);
plot(x,y2)
title('Subplot 2: sin(2x)')

subplot(2,2,3)
y3 = sin(4*x);
plot(x,y3,'ks')
title('Subplot 3: sin(4x)')

subplot(2,2,4)
y4 = sin(8*x);
plot(x,y4, 'r:+)
title('Subplot 4: sin(8x)')
```

Listing 11: Rysowanie funkcji *sin* na współdzielonym wykresie

Na powyższym przykładzie przy rysowaniu niektórych wykresów wykorzystano markery zmieniające styl i kolory rysowanych przebiegów:

- zdefiniowane kolory: `'c','m','y','r','g','b','w','k'` odpowiadają one kolejno kolorom: niebieskozielonym, różowym, żółtym, czerwonym, zielonym, niebieskim, białym i czarnym;
- style linii: `-` ciągła linia, `--` linia przerywana, `:` linia wykropkowana, `-.` linia przerywana myślnikiem i kropką;
- typy markerów: `+`,`o`, `*`, `x`, `s` (kwadrat), `d` (romb), `^`,`>`,`v`,`<` (trójkąty zwrócone w różne strony), `p` (pentagram) i `h` (heksagram).

Jeśli konieczne jest ustalenie zakresów osi XY należy użyć funkcji `axis([xmin xmax ymin ymax])`, jeśli sami nie ustalimy tych zakresów MATLAB/Octave ustali je automatycznie. Włączenie widocznej siatki na wykresie można zrealizować za pomocą wyrażenia: `grid on`. Opisy poszczególnych osi tworzy się za pomocą funkcji: `xlabel` i `ylabel` natomiast legendę do wykresów za pomocą `legend`. Do ustawienia tytułu wykresu korzystamy natomiast z wyrażenia: `title`.

```
t = -2*pi:pi/100:2*pi;
y1 = sin(t);
y2 = cos(t);
plot(t,y1);
hold on
plot(t, y2);
axis([-2*pi 2*pi -1 1]);
xlabel('-2\pi \leg {\itt} \leg 2*\pi')
ylabel('f(x)')
title('My plot');
legend('sin(x)','cos(t)');
```

Listing 12: Nazywanie osi dodawanie tytułu oraz legendy i zmiana zakresów osi układu współrzędnych

1.1.4 Instrukcje warunkowe/pętle

MATLAB/Octave posiada wiele konstrukcji pozwalających na sterownie przepływem programu. Tak jak w większości znanych języków w MATLAB/Octave można korzystać z instrukcji warunkowych `if-else`, pętli `for`, `while` czy `switch-case`.

Przykładowe użycie instrukcji warunkowej `if-else` zostało pokazane na poniższym listingu 13. Fragment programu wykonuje porównanie dwóch liczb i wypisuje na ekranie konsoli informację o wyniku tego porównania. Dalej zostało pokazane użycie pętli `for` 14. Widać, że pętla zachowuje się jak `foreach`, iteruje ona wektor znajdujący się po prawej stronie znaku `=`.

```
A = 12;
B = 14
if A > B
  disp('Greater')
elseif A < B
  disp('Less')
elseif A == B
  disp('Equal')
else
  disp('Can not be evaluated')
end
```

Listing 13: Porównywanie dwóch liczb z wykorzystaniem instrukcji `if-else`

```
for i = 1:10
  disp(i)
end

for i = 1.0:-0.2:0.0
  disp(i)
```

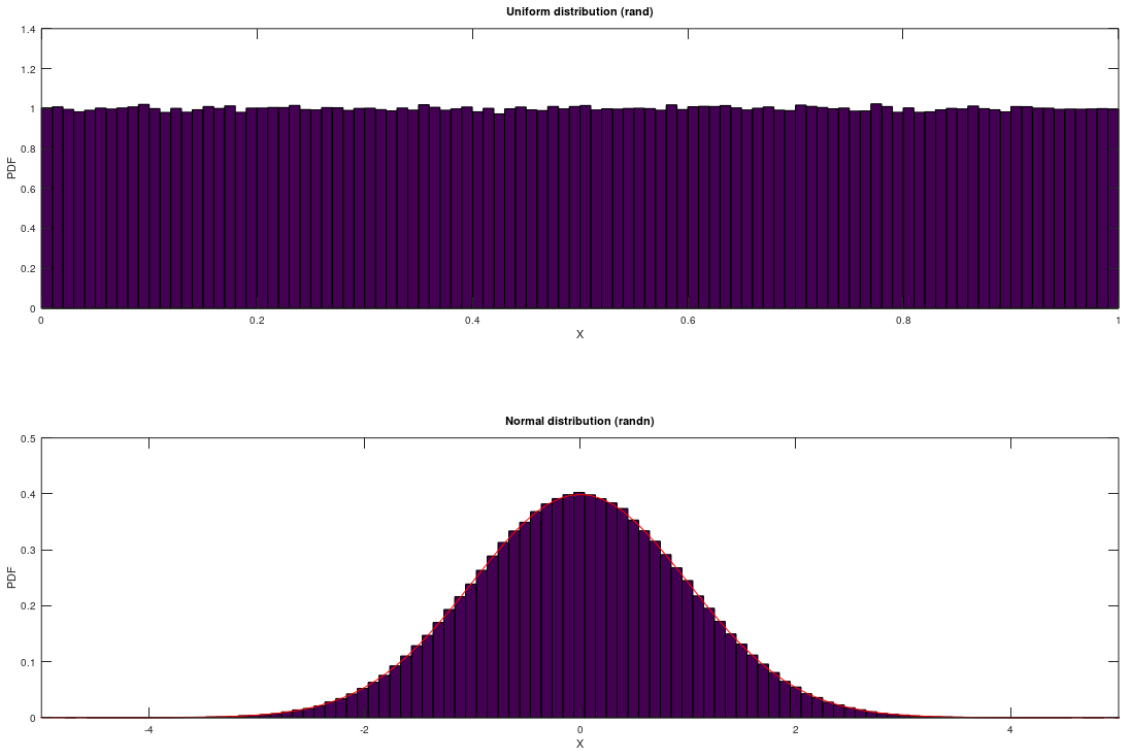
```
end

for i = [1 4 5 10 -8]
  disp(i)
end
```

Listing 14: Trzy pętle `for` wykonujące się na różnych przedziałach wektora `i`

Na tych zajęciach nie będziemy poruszać możliwości programowania obiektowego w języku MATLAB/Octave, tym niemniej należy wspomnieć, że w omawianych językach możliwe jest tworzenie własnych typów (klas). Przy bardziej złożonych problemach, szczególnie przy operowaniu na dużej ilości danych, wykorzystanie obiektów pozwala znaczenie uprościć wykonywaną pracę.

1.2 Szumy

W cyfrowym przetwarzaniu sygnałów często będziemy mieć do czynienia z szumami, które w przeciwieństwie do sygnałów deterministycznych nie są opisane przez żadne znane zależności matematyczne. Szum jest procesem losowym, jednak posiada pewne cechy, które go definiują. Jeśli rozważymy np. rzuty kostką, to rozkład gęstości prawdopodobieństwa takiej zmiennej losowej będzie przyjmował kształt „prostokąta" (rozkład równomierny). Prawdopodobieństwo wylosowania konkretnej liczby oczek wynosi $\frac{1}{6}$. Wiele rzeczywistych procesów losowych posiada jednak gaussowski rozkład gęstości prawdopodobieństwa np. szumy termiczne czy promieniowanie. W przetwarzaniu obrazów szumy gaussowskie mogą być powodowane przez niewystarczające naświetlenie, wysoką temperaturę czy szum elektryczny. Z szumem gaussowskim bardzo często możemy się spotkać w rzeczywistych procesach. Np. w kontekście szumu termicznego, zaburzenia są powodowane ogromną liczbą losowo oscylujących cząsteczek, pobudzanych przez energię cieplną.



Rysunek 1: Wykresy gęstości prawdopodobieństwa dla szumu o rozkładzie równomiernym oraz normalnym.

Oba wymienione powyżej szumy są szumami białymi. Oznacza to, że składają się one z mieszaniny różnych częstotliwości, które powodują, że takie szumy posiadają płaskie widmo [1].

Rozkład równomierny:

$$p(x_0) = \begin{cases} \frac{1}{b-a} & \text{dla } a \leq x_0 \leq b \\ 0 & \text{dla pozosotałych } x_0 \end{cases} \tag{1}$$

[1] Obraz uzyskany w wyniku rozłożenia danego sygnału na szereg składowych o różnych częstotliwościach występujących w tym sygnale

Rozkład normalny:

$$p(x_0) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left[-\frac{(x_0 - \bar{x}^2)}{2\sigma^2}\right] \tag{2}$$

W programie MATLAB/GNU Octavie do utworzenia wektora zawierającego próbki szumu, można wykorzystać funkcję `rand` dla rozkładu równomiernego, `randn` dla rozkładu normalnego 15. Obie funkcje zwracają wektor liczb pseudolosowych. Wywołane z jednym argumentem `n`, tworzą macierz o rozmiarach `nxn`. Jeśli zostaną przekazane dwa parametry to, pierwszy z nich odnosi się do liczby wierszy w tej macierzy, a ten drugi do liczby kolumn.

```
>> rand(5)
ans =

0.064519   0.226600   0.705842   0.346158   0.490804
0.905697   0.095875   0.832029   0.984328   0.834852
0.080022   0.437925   0.978008   0.579209   0.149026
0.801113   0.988285   0.658155   0.395238   0.978390
0.557152   0.358754   0.558685   0.384983   0.508283

>> randn(1,5)
ans =

-1.6292  -1.2919   0.7065  -0.7206  -0.2104
```

Listing 15: Tworzenie macierzy/wektora liczb pseudolosowych z wykorzystaniem funkcji `rand` oraz `randn`

Wykorzystując funkcję `hist(y,n, norm)` można stworzyć histogramy dla dowolnego zbioru danych. Funkcja jako pierwszy argument (`y`) przyjmuje wektor danych, drugi (`n`) liczbę kontenerów („słupków") histogramu. Trzeci argument można podać, jeśli chcemy dokonać normalizacji. Na rysunku 1 zostały pokazane histogramy, wektorów liczb pseudolosowych utworzonych za pomocą dwóch wcześniej wspomnianych metod. Widać, że prawdopodobieństwo wylosowania liczby z przedziału od $\langle 0, 1\rangle$ jest jednakowe, dla każdej wartości. W drugim przypadku prawdopodobieństwo wylosowania wartości 0 jest największe (wartość oczekiwana w tym przypadku wynosi zero).

Zazwyczaj jeśli chcemy wygenerować szum np. w celu dodania go do sygnału, lepiej jest wykorzystać szum o rozkładzie gaussowskim. Szum o rozkładzie normalnym jest bardzo powszechny w rzeczywistych procesach (ten fenomen tłumaczy centralne twierdzenie graniczne `https://en.wikipedia.org/wiki/Central_limit_theorem`).

1.3 Operacja splotu

Splot (ang. convolution) jest operacją matematyczną, podobnie jak operacje dodawania czy mnożenia. Analogicznie do operacji dodawania, gdzie bierzemy dwie liczby i tworzymy nową (trzecią) wartość, tak samo splot „bierze" dwa sygnały i zwraca w wyniku nowy sygnał.

Operacja splotu jest wykorzystywana z bardzo wielu obszarach nauki i inżynierii. W teorii sterowania czy przetwarzaniu sygnałów używa się jej do filtracji danych wejściowych czy wyznaczenia odpowiedzi danego układu na zadane pobudzenie. W przetwarzaniu obrazów splot może być wykorzystany do wykrywania krawędzi czy rozmywania obrazu 2. Wykorzystywany jest również w sieciach neuronowych, teorii prawdopodobieństwa, fizyce i wielu innych dziedzinach.

Splot dwóch ciągłych sygnałów opisany jest wzorem:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau \tag{3}$$

Natomiast dla sygnałów dyskretnych splot definiowany jest następująco:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{4}$$

Obliczenie splotu najprościej mówiąc polega na obróceniu w czasie jednego z dwóch sygnałów, następnie wymnożeniu kolejnych próbek obu sygnałów oraz ich zsumowaniu. W wyniku takiego jednego kroku (wymnożenie i zsumowanie) otrzymujemy pierwszą wynikową próbkę splotu. W kolejnym kroku przesuwamy

Rysunek 2: Przetworzenie oryginalnego obrazu (najbardziej po lewej stronie) za pomocą operacji splotu z użyciem różnych filtrów (kolejno rozmycie, wykrywanie krawędzi, wyostrzanie).

wcześniej obrócony sygnał o jeden krok prawo i ponownie wykonujemy operację mnożenia i sumowania. Operacje te powtarzamy aż kolejne wyliczane próbki będą zerowe. Animację pokazującą splot można zobaczyć na: `https://mathworld.wolfram.com/Convolution.html`.

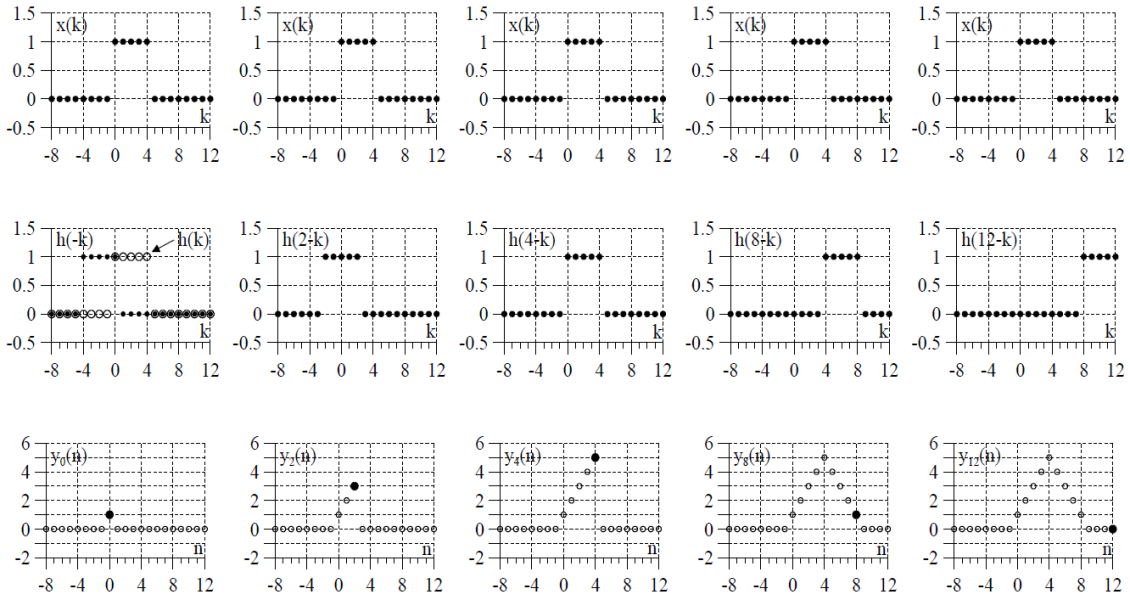
W programie MATLAV/Octave do obliczenia splotu wykorzystuje się funkcję `conv`, która przyjmuje dwa argumenty będące wektorami dwóch splatanych sygnałów. Obliczenie splotu dwóch wektorów A oraz B:

$$A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$
$$B = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

w programie MATLAB/Octave zostało pokazane na poniższym listingu.

```
A = [1 1 1];
B = [1 2 3];
C = conv(A,B);
```

Listing 16: Obliczanie splotu dwóch wektorów



Rysunek 3: Graficzna prezentacja splotu dwóch sygnałów prostokątnych [1].

1.4 Zadania

1.4.1 Tworznie macierzy

Korzystając z operatora : stwórz macierz A:

$$A = \begin{bmatrix} 1 & 1.5 & 2 & 2.5 & 3 & 3.5 & 4 & 4.5 & 5 \\ -5 & -7 & -9 & -11 & -13 & -15 & -17 & -19 & -21 \\ 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 \end{bmatrix}$$

Następnie za pomocą `[]` usuń drugą, czwartą oraz piątą kolumnę ze stworzonej macierzy. Na koniec dodaj do macierzy A wiersz zawierający same jedynki.

1.4.2 Rysowanie przebiegów funkcji

Narysuj na współdzielonym rysunku następujące wykresy (`subplot`):

1. $y(t) = 2sin(2t - \frac{\pi}{2})$,
2. $y(t) = \frac{10}{t-5} + 20$ (konieczne będzie wykorzystanie operatora dzielenia z poprzedzającą go kropką (`./`), pozwoli to na wykonanie dzielenia dla każdego elementu wektora z osobna),
3. szum biały o rozkładzie normalnym (stwórz wektor dowolnej długości za pomocą funkcji `randn`),
4. przebieg piło-kształtny (wykorzystaj funkcję `sawtooth`),
5. przebieg prostokątny (wykorzystaj funkcję `square`).

1.4.3 Histogramy wektorów liczb pseudolosowych

Na podstawie informacji zawartych w rozdziale 1.2 stwórz dwa wektory zawierające po 10000 liczb pseudolosowych za pomocą funkcji `rand` oraz `randn`. Na współdzielonym wykresie narysuj histogramy (funkcja `hist`) tych wektorów. Jako pierwszy argument funkcji `hist` podaj wektor liczb pseudolosowych, jako drugi liczbę słupków histogramu (np. 100) i jako trzeci argument, wartość normalizacyjną (powinna ona wynosić tyle, aby pole pod powierzchnią histogramu wynosiło jeden). Dzięki zastosowaniu normalizacji, histogramy kształtem będą odpowiadać rozkładom gęstości prawdopodobieństwa (rysunek 1).

Korzystając ze wzoru 2 wyznacz wartości funkcji gęstości prawdopodobieństwa rozkładu normalnego i nanieś go na drugi ze stworzonych przed chwilą histogramów. Powinny one się pokryć.

1.4.4 Filtr średniej ruchomej

Filtr średniej ruchomej jest specjalnym przypadkiem filtrów o skończonej odpowiedzi impulsowej, którym poświęcimy jedne z przyszłych laboratoriów. Na ten moment należy jedynie wspomnieć, że filtr średniej ruchomej dodaje do siebie wszystkie wartości filtrowanego sygnału (mieszczące się w oknie tego filtra) i następnie dzieli je przez długość okna tego filtra 5.

$$movAvg = \frac{x[n] + x[n-1] + ... + x[n-N]}{N+1} \tag{5}$$

Stwórz sygnał o przebiegu prostokątnym w tym celu możesz wykorzystać funkcję `square` (w programie GNU Octave może okazać się konieczne wcześniejsze załadowanie pakietu `signal`, aby to zrobić wpisz w wierszu poleceń `pkg load signal`). Dodaj do tej funkcji biały szum gaussowski (o rozkładzie gaussowskim/normalnych) i wariancji 0.25 (`noise = 0.25*rand(size(y,1))`), gdzie `y` jest wygenerowanym sygnałem prostokątnym.

Utwórz teraz wektor współczynników filtra średniej ruchomej, który „spleciony" z zaszumionym sygnałem fali prostokątnej pozwoli na zmniejszenie negatywnego wpływu szumu 17.

```
1 movAvgLength = 5;
2 movAvg = 1/movAvgLength * ones(movAvgLength,1);
```

Listing 17: Przykładowe współczynniki filtra średniej ruchomej

Wykorzystaj poznaną funkcję `conv`, aby obliczyć splot zaszumionego przebiegu z filtrem średniej ruchomej. Narysuj **na jednym wykresie** sygnał fali prostokątnej przed i po filtracji.

2 Laboratorium nr 2

2.1 Funkcja autokorelacji

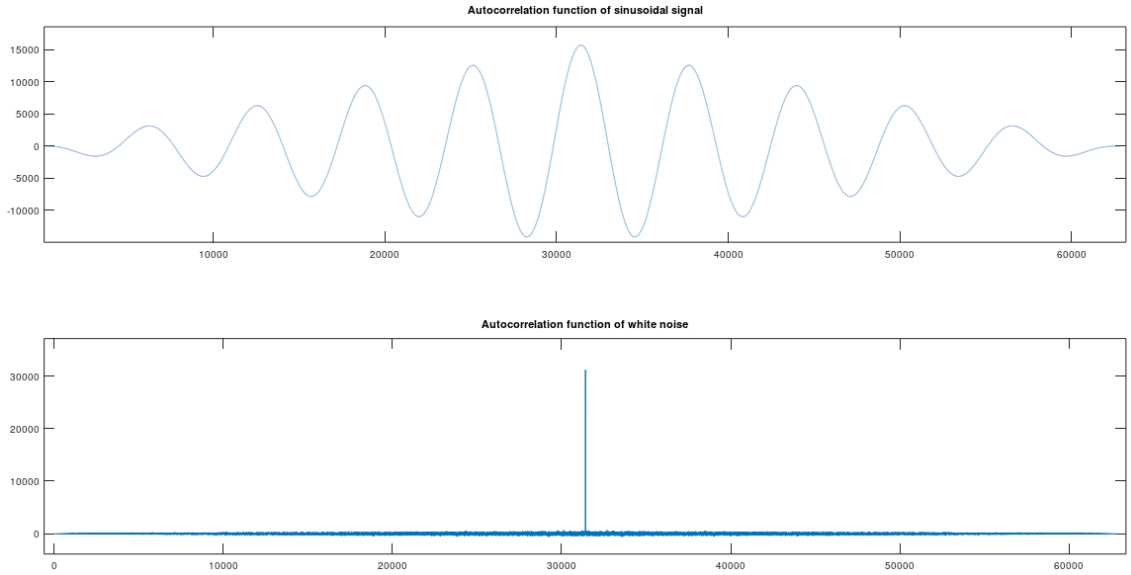
Funkcja (auto)korelacji pozwala uzyskać informacje o podobieństwie dwóch sygnałów w kolejnych chwilach przesunięcia τ. Może być wykorzystywana do badania okresowości sygnału. Pozwala one na sprawdzenie, jak bardzo dwa sygnały są do siebie podobne. Przyjmuje ona największe wartości dla wartości przesunięcia τ wynoszącego wielokrotność okresu sygnału. Funkcja autokorelacji można użyć do wyznaczania okresu głosek dźwięcznych mowy czy detekcji odbić w odebranym sygnale echa (echolokacja).

Dla sygnałów dyskretnych funkcja korelacji własnej może być obliczona z poniższego wzoru:

$$R_{xx}(k) = \sum_{k=\infty}^{\infty} x[n]x[n+k] \tag{6}$$

Widać, że powyższa formuła 6 z wyjątkiem odwrócenia w czasie jednego z sygnałów jest tym samym co splot dwóch wektorów 1.3.

Do obliczenia wartości funkcji (auto)korelacji w programie MATLAB, służy funkcja `xcorr`, która może być wywołana z jednym argumentem albo dwoma argumentami będącymi wektorami próbek analizowanych sygnałów. W pierwszym przypadku zostanie policzona autokorelacja, w drugim natomiast korelacja wzajemna, która pozwala na szukanie podobieństwa pomiędzy dwoma różnymi sygnałami.



Rysunek 4: Przykład funkcji autokorelacji sygnału sinusoidalnego oraz szumu białego.

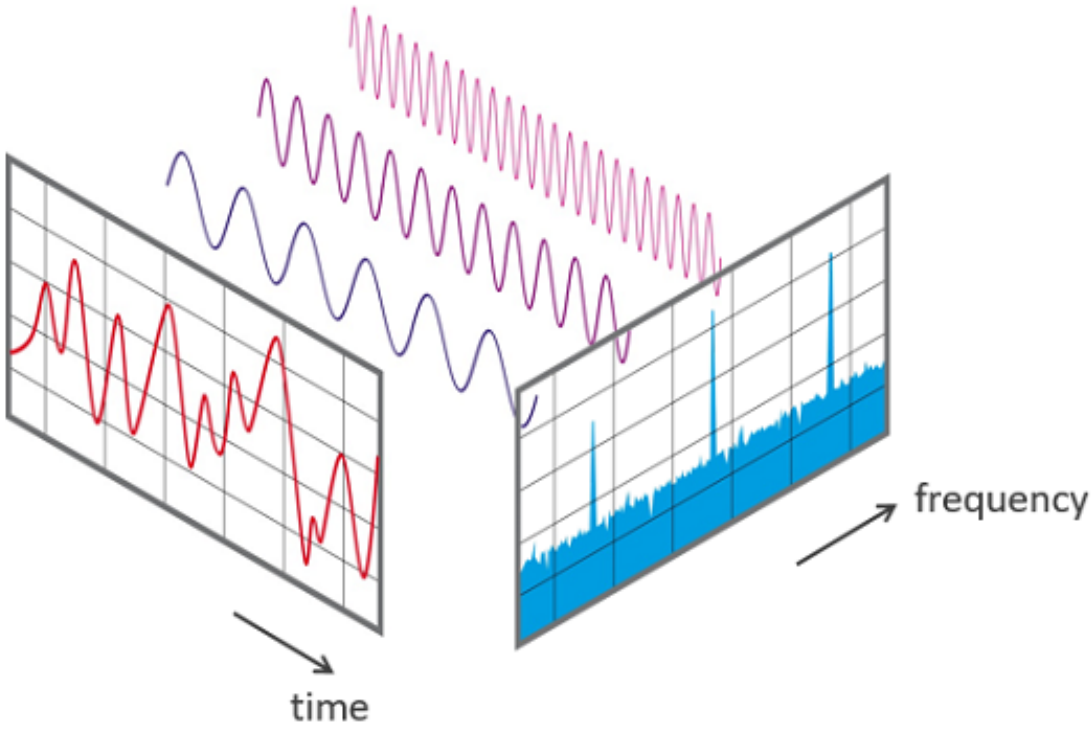
Na rysunku 4 pokazano wykresy dwóch funkcji autokorelacji odpowiednio sygnału sinusoidalnego oraz szumu białego. W pierwszym przypadku funkcja autokorelacji ukazuje okresowość badanego przebiegu, posiada powtarzające się oscylacje. Natomiast ze względu na fakt, że kolejne próbki szumu białego są od siebie niezależne (wymusza to również, fakt że są nieskorelowane), funkcja autokorelacji przyjmuje wartości zerowe (z wyjątkiem próbki zerowej, gdzie przyjmuje wartość maksymalną).

2.2 Dyskretna transformacja Fouriera

Przekształcenie Fouriera umożliwia zamianę reprezentacji sygnału z dziedziny czasu na reprezentację w dziedzinie częstotliwości (inaczej mówiąc jest to transformacja pomiędzy układami współrzędnych).

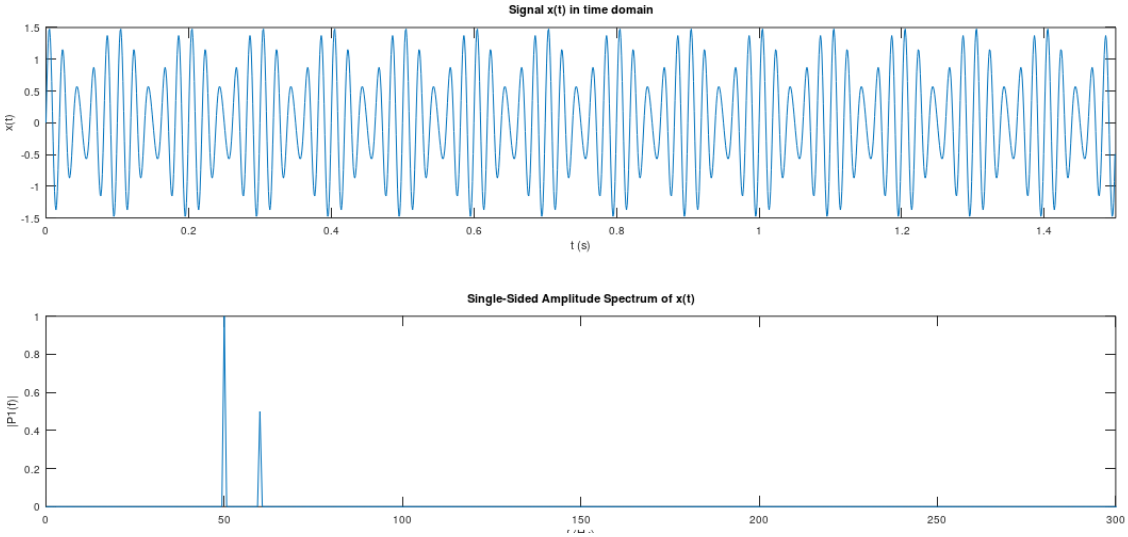
Transformacja ta jest narzędziem, które pozwala dowiedzieć się, jakie częstotliwości składowe zawiera sygnał w dziedzinie czasu poddawany transformacji. Jest ona pewnego rodzaju „skanerem", który bada korelację badanego sygnału z funkcjami sinusoidalnymi i kosinusoidalnymi o różnych częstotliwościach.

Na rysunku 5 widać sygnał w dziedzinie czasu (czerwony wykres). Składa się on z sumy trzech sygnałów sinusoidalnych. W dziedzinie częstotliwości są one reprezentowane jako prążki odpowiedniej częstotliwości poszczególnych składowych. Inaczej mówiąc, jeśli mamy sygnał sinusoidalny o częstotliwości 10 Hz to po transformacji Fouriera otrzymamy pojedynczy prążek dla argumentu 10 Hz o wysokości równej amplitudzie



Rysunek 5: Przykład reprezentacji sygnału w dziedzinie czasu i częstotliwości.

tej składowej. Jeśli sygnał składa się większej liczby składowych będzie to odwzorowane większą liczbą nie zerowych prążków w dziedzinie częstotliwości. Na rysunku 6 przedstawiono sygnał $x(t)$ składający się z sumy dwóch sygnałów sinusoidalnych o częstotliwościach 50 Hz oraz 60 Hz i amplitudach odpowiednio 1 i 0.5. Na górnej części rysunku 6 widać sygnał w dziedzinie czasu, z którego nie jest łatwo określić jakie składowe zawiera $x(t)$. Na dolnym wykresie natomiast, widać sygnał w dziedzinie częstotliwości, gdzie wyraźnie są pokazane prążki reprezentujące jego składowe.



Rysunek 6: Przykład reprezentacji sygnału w dziedzinie czasu i po DFT w dziedzinie częstotliwości.

W przypadku przekształcenia Fouriera konieczne jest realizowanie nieskończonych granic sumowania co jest nierealizowalne w praktycznych zastosowaniach. Dlatego wyznacza się szereg Fouriera dla sygnałów dyskretnych (ang. Discrete Fourier Transform, DFT). Transformacja DFT bierze N próbek przebiegu $x[n]$

i zwraca w wyniku wektor $X[k]$ o długość N. Prosta transformacja DFT przyjmuje postać:

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{kn}{N}} \tag{7}$$

Natomiast odwrotna transformacja DFT może być przedstawiona wzorem:

$$x[n] = \sum_{k=0}^{N-1} X[k] e^{j2\pi \frac{kn}{N}} \tag{8}$$

,gdzie:

$$e^{-j2\pi \frac{kn}{N}} = [cos(2\pi \frac{kn}{N}) - jsin(2\pi \frac{kn}{N})] \tag{9}$$

Z perspektywy tych zajęć laboratoryjnych do wyznaczania transformacji Fouriera, będziemy korzystać z funkcji `fft` programu MATLAB/Octave. Pozwala on obliczenie prostej i odwrotnej dyskretnej transformacji Fouriera za pomocą algorytmu szybkiej transformacji Fouriera (ang. Fast Fourier Transform, FFT).

FFT pozwala na efektywne wyznaczenie szeregu Fouriera. Jest to jeden najważniejszych algorytmów jakie zostały wymyślone. Algorytm ten jest wykorzystywany przy analizie sygnałów audio i obrazów (np. JPEG, MP3, XviD), pozwala na dokonywanie kompresji sygnałów, jest również wykorzystywany przy numerycznym rozwiązywaniu równań różniczkowych. Większość współczesnej komunikacji cyfrowej jest oparta na algorytmie FFT.

Przy wywołaniu funkcji `X = fft(x)` z argumentem będącym wektorem próbek w dziedzinie czasu zostanie zwrócony wektor liczb zespolonych o takiej samej długości co długość wektora `x`. Należy jednak mieć na uwadze, że otrzymane widmo jest symetrycznie odbite względem częstotliwości Nyquista[2]. Tak więc, po wykonaniu FFT można wziąć do analizy jedynie połowę otrzymanego wektora. Następnie jeśli chcemy, aby wysokość prążków odpowiadała amplitudzie składowych sygnału konieczne jest wykonanie skalowania tzn. podzielenie próbek wektora przez długość tego wektora oraz pomnożeniu go przez dwa. Następnie do narysowania wykresu (za pomocą funkcji `plot`) należy wziąć moduł otrzymanych liczb zespolonych `plot(abs(X))`. Przykład zastosowania funkcji `fft` programu MATLAB/Octave został pokazany na listeningu 18. Podczas zajęć laboratoryjnych zostanie on omówiony krok po kroku.

```
Fs = 1000; % sampling frequency
T = 1/Fs; % sampling period
L = 1500; % length of signal
t = (0:L-1)*T; % time vector

y = sin(2*pi*50*t+pi/4) + 0.5*sin(2*pi*60*t); % create signal containing
    two components (50 and 60 [Hz])

% FFT
Y = fft(y); % calculate spectum of input signal
Y = abs(Y/L); % take absolute value of Y (we want to plot only amplitude
    of the spectrum)
Y = Y(1:L/2+1); % throw away second half of the spectrum because is
    repeated (all usefull information in first half), reflected via Nyquist
     frequency i.e. Fs/2
Y(2:end-1) = 2*Y(2:end-1);  % multiply by 2 to take into account the fact
    that we threw out second half of FFT above

% Plot the results
subplot(2,1,2)
f = Fs*(0:(L/2))/L; % create frequency vector (fft calculates spectrum
    from 0 to sampling frequency in this example Fs = 1000 Hz), we plot
    only to 500 Hz, bacuse half of the spectrum has been throw out
plot(f,Y)
title('Widmo sygnaly y(t)')
xlabel('f (Hz)')
```

[2]Częstotliwość Nyquista jest maksymalną częstotliwością sygnału, która może być odtworzona po operacji próbkowania. W praktyce jest ona równa połowie częstotliwości próbkowania $f_N = \frac{f_S}{2}$.

```
ylabel('|Y(f)|')
axis([0 300 0 1])

subplot(2,1,1);
plot(t,X)
title('y(t) in time domain')
xlabel('t (s)')
ylabel('y(t)')
axis([0 1.5 -1.5 1.5])
```

Listing 18: Przykład użycia funkcji `fft` programu MATLAB/Octave

Tak jak zostało wcześniej wspomniane w wyniku operacji FFT zostaje zwrócony wektor $X[k]$ o długość równej liczbie próbek wektora sygnału poddawanego transformacji. Jednocześnie kolejne elementy wektora $X[k]$ odpowiadają kolejnym częstotliwością w przedziale $f \in [0; fs]$, gdzie fs jest częstotliwością próbkowania. Chcąc poprawnie odwzorować argumenty na osi X należy stworzyć wektor wartości od zera do połowy częstotliwości próbkowania.

Funkcja `fft` zwraca wektor liczb zespolonych $X_{re} + jX_{im}$. Każda liczba zespolona może być opisana za pomocą modułu i fazy. W przypadku `fft` moduł odpowiada amplitudzie danej składowej przebiegu, natomiast faza przesunięciu w fazie. Dla funkcji $x(t) = 0.5cos(2\pi 5t + \frac{\pi}{3})$ moduł powinien po przeskalowaniu wynieść 0.5 natomiast faza 60^o. Tak więc, amplituda może być wyznaczona za pomocą wzoru 10 natomiast faza ze wzoru 11. Ze względu na fakt, że funkcja tangens jak i jej odwrotność są zdefiniowane na przedziale $[\frac{-\pi}{2}, \frac{\pi}{2}]$ natomiast w wyniku operacji `fft` otrzymujemy liczby zespolone, które mogą posiadać kąt w przedziale $[-\pi, \pi]$, w celu obliczenia przesunięcia fazowego należy korzystać z funkcji `atan2()` zamiast standardowej `atan()`.

$$|X[k]| = \sqrt{X_{re}^2 + X_{im}^2} \tag{10}$$

$$\angle X[k] = tan^{-1} \frac{X_{im}}{X_{re}} \tag{11}$$

Operację odwrotnej dyskretnej transformaty Fouriera w programie MATLAB/Octave można zrealizować przy pomocy funkcji `x = ifft(X)`. Jeśli jako argument tej funkcji przekażemy transformatę Fouriera w wyniku powinniśmy ponownie otrzymać sygnał w dziedzinie czasu.

2.3 Wyciek widma i funkcje okien

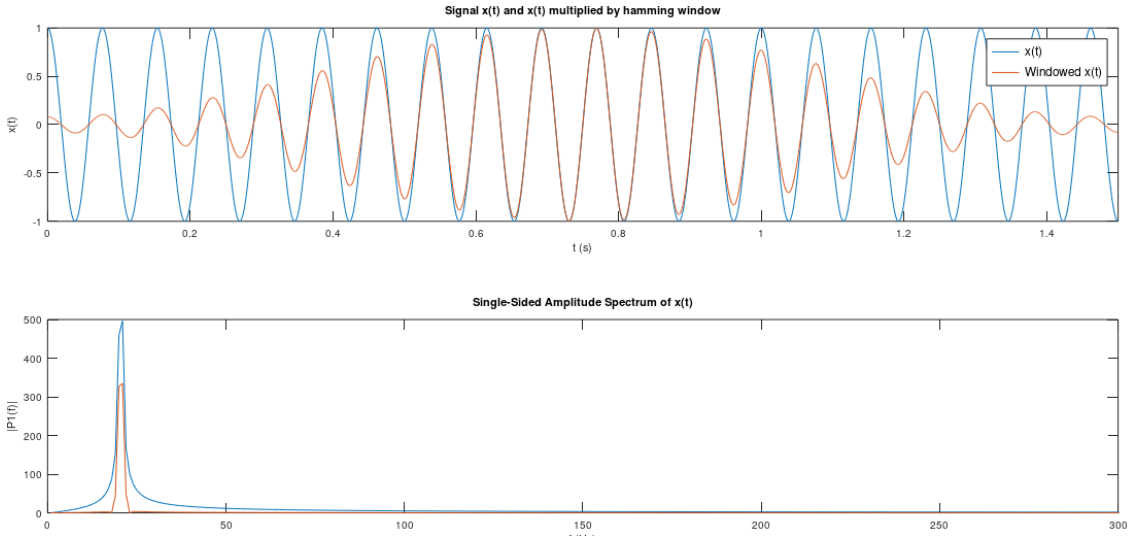
Można powiedzieć, że DFT „przeszukuje" sygnał w celu znalezienia jego składowych. Algorytm wykorzystuje w tym celu pewną **skończoną** liczba pasm o konkretnej szerokości. Zazwyczaj chcemy, aby liczba pasm była jak największa (Δf jak najmniejsza), co powoduje większą rozdzielczość transformaty.

$$\Delta f = \frac{F_s}{N_{FFT}} \tag{12}$$

FFT niejawnie zakłada, że sygnał powtarza się po wystąpieniu ostatniej próbki przekazywanego wektora. Jeśli badamy np. funkcję sinus najlepiej jest poddać operacji DFT wielokrotności jej okresu. Inaczej pojawią się nieciągłości, które powodują tzw. wyciek widma.

Dodatkowo należy wspomnieć, że FFT może „znaleźć" częstotliwości (bez wycieku widma), które idealnie wpasują się w wielokrotności Δf. Np. jeśli chcemy zbadać przebieg o częstotliwości 400 Hz (częstotliwość próbkowania wynosi $f_s = 8000\ Hz$), a długość transformaty wynosi 2048, to nasza składowa powinna znaleźć się w próbce o indeksie $k = \frac{N \cdot f_k}{f_s} = \frac{2048 \cdot 400}{8000} = 102.4$. Oczywiście, jako że k jest indeksem elementu naszej transformaty to indeks ten powinien być dodatnią liczbą całkowitą. W tym przypadku energia z tej składowej zostanie „rozlana" i znajdzie się w elementach wektora o indeksach 102 oraz 103.

Aby ograniczyć wyciek widma, należy pomnożyć sygnał z funkcją okna. Taka operacja powoduje, że na początku i końcu wektora, sygnał ulega wygaszeniu i nie pojawiają się nieciągłości w sygnale. Istnieje wiele typów funkcji okien (Hanna, Hamminga, Blackmana, Kaisera), które charakteryzują się szerokością listka głównego i tłumieniem listków bocznych, a wybór jednego z nich ma wpływ na kształt otrzymywanego widma i oferowaną przez niego rozdzielczość. Poniżej został pokazany sposób utworzenia wektora okna Hamminga w programie MATLAB/Octave i jego wymnożenie przez sygnał sinusoidalny $x(t)$ 19. Na rysunku 7 pokazano natomiast funkcję $x(t)$ oraz funkcję $x(t)$ wymnożoną przez okno Hamminga. Na górnym wykresie w dziedzinie czasu widać funkcja wymnożona przez okno na końcach zbiega do zerach. Natomiast na wykresie w dziedzinie częstotliwości, można zaobserwować, że wykorzystanie okna drastycznie zmniejszyło niepożądany efekt wycieku widma.



Rysunek 7: Przykładowy wyciek widma spowodowany złą długością wektora transformaty Fouriera oraz zmniejszenie niepożądanego efektu za pomocą wymnożenia sygnału przez okno Hamminga.

```
wnd = hamming(length(x));
xWnd = x'.*wnd;
Y = fft(xWnd);
```

Listing 19: Wymnożenie sygnału $x(t)$ przez okno Hamminga

2.4 Zadania

2.4.1 Badanie okresowości sygnału na podstawie funkcji autokorelacji

Dokonaj analizy za pomocą funkcji autokorelacji przebiegu czasowego postaci: $x(t) = sin(2\pi 10t) + 3n(t)$, gdzie n jest szumem białym o rozkładzie normalnym i odchyleniu standardowym równym 3. Narysuj na współdzielonym wykresie:

- przebieg sygnału $x(t)$ w dziedzinie czasu,
- funkcję autokorelacji za pomocą funkcji programu MATLAB/Octave `xcorr(x)`.

Zwróć uwagę, że na wykresie funkcji autokorelacji dużo łatwiej można zaobserwować okresowość sygnału wejściowego.

2.4.2 Wyznaczenie dyskretnej transformacji Fouriera

Wyznacz transformaty Fouriera następujących sygnałów:

- $y(t) = cos(2\pi 10t)$,
- $y(t) = cos(2\pi 10t + \frac{\pi}{6})$,
- $y(t) = cos(2\pi 10t) + cos(2\pi 20t) + cos(2\pi 30t)$,
- $y(t) = cos(2\pi 10t) + n(t)$ (funkcja sinus z dodanym szumem białym o rozkładzie normalnym),
- przebiegu prostokątnego,
- szumu białego o rozkładzie równomiernym i normalnym.

2.4.3 Usunięcie szumów z sygnału z wykorzystaniem FFT

Stwórz sygnał $x(t) = cos(2\pi 10t) + cos(2\pi 20t)$ o takiej długości, aby składowa o niższej częstotliwości zawierała najmniej kilka okresów. Następnie dodaj do niego szum o rozkładzie normalnym (długość wektora zawierającego szumy powinna być taka sama jak długość wektora zawierająca sygnał, możesz posłużyć się funkcją `length()` albo `size()` w celu pobrania długości wektora/tablicy).

Przygotuj współdzielony wykres (`subplot(2,1,2)`). Na górnym wykresie narysuj przebieg sygnału $x(t)$ za pomocą funkcji `plot(t,x)`. Następnie wyznacz dyskretną transformatę Fouriera tego sygnału. Wyzeruj wszystkie elementy wektora (wartości dla których `abs(X) > threshold`), które nie przekraczają zdefiniowanego progu np. 100, nie jest konieczne wykonywanie normalizacji/skalowania transformaty Fouriera na tym etapie. Dalej dla tak zmanipulowanego wektora $X[k]$ oblicz odwrotną transformatę Fouriera (`ifft(X)`). Wynik przedstaw na utworzonym wcześniej współdzielonym wykresie.

2.4.4 Rozdzielczość transformaty Fouriera

Zazwyczaj obliczanych jest podczas dyskretnej transformaty Fouriera tyle prążków widma, ile próbek ma sygnał. Należy jednak pamiętać, że im więcej otrzymamy w wyniku DFT prążków tym lepiej będzie odwzorowane faktyczne widmo sygnału. Stwórz sygnał składający się z dwóch składowych o częstotliwościach znajdujących się blisko siebie np. 50 Hz oraz 60 Hz. Wywołując funkcję `fft()` z drugim dodatkowym parametrem możemy zmieniać długość wektora transformaty (`fft(x,n)` zwróci nam `n` elementową transformatę). Sprawdź jaka jest najmniejsza wartość `n`, która pozwoli na rozróżnienie dwóch prążków stworzonego przed chwilą sygnału $x[n]$

2.4.5 Zmniejszenie efektu wycieku widma przez zastosowanie okna Hamminga

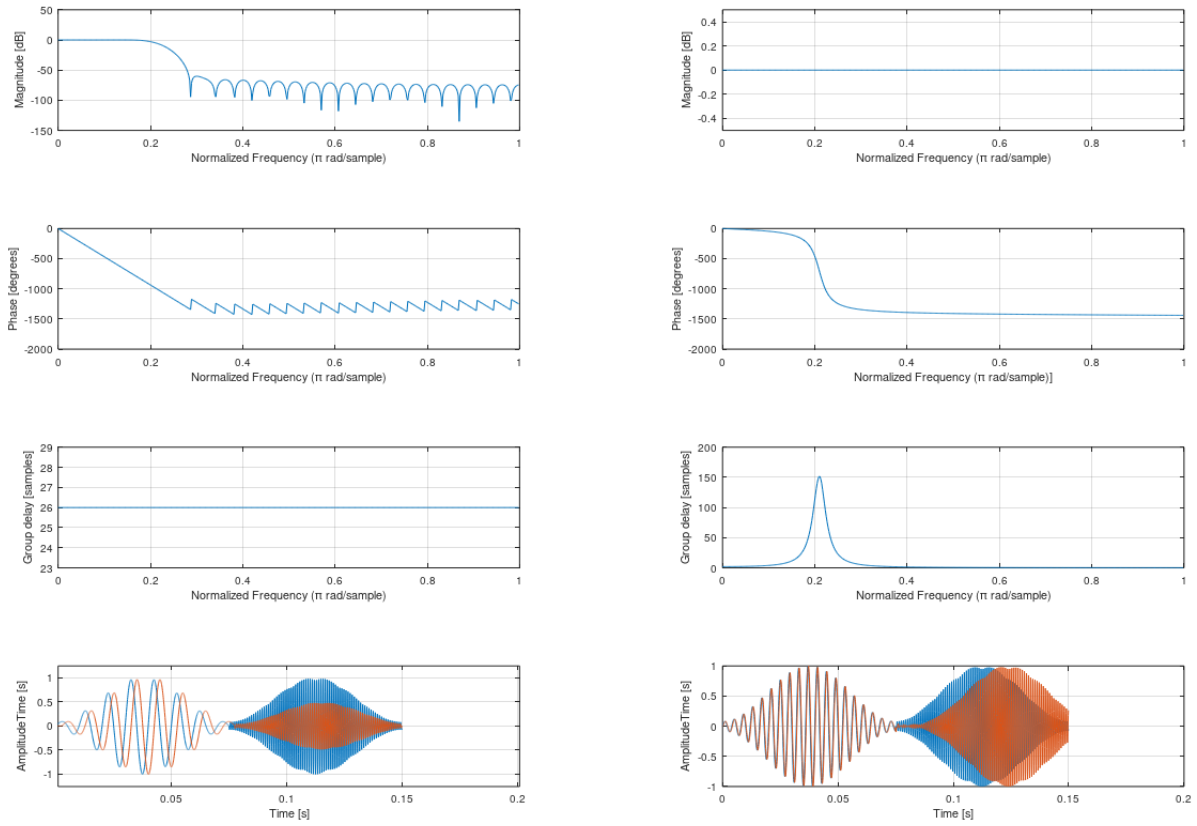
Stwórz sygnał o częstotliwości np. 13 Hz i o długości 1500 próbek. Częstotliwość próbkowania należy ustawić tak, aby spowodować wyciek widma np. $f_s = 1000$. Oblicz widmo utworzonego sygnału i narysuj go na wykresie. Następnie wymnóż ten sygnał przez okno Hamminga (listening 19), o takiej samej długości co sygnał i ponownie wyznacz widmo. Użyj wyrażenie `hold on` i ponownie narysuj wykres widma (tym razem sygnału wymnożonego przez okno). Efekt wycieku widma w drugim przypadku powinien być zniwelowany.

3 Laboratorium nr 3

Podczas dzisiejszych zajęć omówiony zostanie temat projektowania filtrów zarówno tych o skończonej odpowiedzi impulsowej (ang. Finite Impulse Response, FIR) jak i o nieskończonej odpowiedzi impulsowej (ang. Infinite Impulse Response, IIR). Różnią się one złożonością obliczeniową, filtry nierekursywne są pod tym względem bardziej skomplikowane. Konieczna jest większą liczba współczynników filtra, aby osiągnąć strome zbocze w paśmie przejściowym. Bardzo dużą natomiast zaletą filtrów typu FIR jest fakt, że są one zawsze stabilne. Nie ma takich wartości wagowych filtra, które spowodują że sygnał wyjściowy się wzbudzi i wzrośnie do nieskończoności. Dodatkową zaletą filtrów o skończonej odpowiedzi impulsowej jest liniową charakterystyka fazowo-częstotliwościowa co jest szczególnie pożądane przy niektórych typach sygnałów np. w aparaturze medycznej.

3.1 Odpowiedzi częstotliwościowe filtrów cyfrowych

Filtry (niezależnie od typu) mogą być opisane przez tzw. charakterystyki częstotliwościowe. Opisują one jak dany filtr „działa" na sygnał wejściowy. Filtr może odpowiednio wzmacniań/tłumić dane pasma częstotliwości oraz dodatkowo może dane częstotliwości przesuwać w fazie. Na rysunku nr 8 zostały pokazane odpowiedzi częstotliwościowe dwóch filtrów: dolnoprzepustowego FIR (lewa kolumna) oraz wszystkoprzepustowego filtra IIR (prawa kolumna). Na wykresach w pierwszym wierszu można zaobserwować jak poka-



Rysunek 8: Odpowiedzi częstotliwościowe, opóźnienie grupowe oraz przykład filtracji przykładowego dolnoprzepustowego filtra FIR (lewa kolumna) oraz wszystkoprzepustowego IIR (prawa kolumna).

zane filtry będą zmieniały amplitudy sygnałów wejściowych w zależności od częstotliwości. Pierwszy z nich będzie „przepuszczał" częstotliwości od 0 do 0.2 unormowanej względem jedynki częstotliwości Nyquista. Inaczej mówiąc, przy założeniu, że częstotliwość próbkowania F_s wynosi 10000 Hz (częstotliwość Nyquista $\frac{F_s}{2} = 5000\ Hz$) do filtr przedstawiony w lewej kolumnie będzie przenosił częstotliwości od 0 do 1000 Hz ($5000 \cdot 0.2$). Natomiast filtr przedstawiony w prawej kolumnie, ponieważ wzmocnienie wynosi 0 dB dla całego pasma, nie będzie tłumił/wzmacniał żadnej częstotliwości, kolokwialnie mówiąc „przepuści wszystko".

W drugim wierszu natomiast zostały pokazane charakterystyki fazowe filtrów. Po lewej stronie można zaobserwować liniowy wykres (z wyjątkiem skoków o wartość równą π w miejscach, gdzie charakterystyka amplitudowa zmienia znak). Wszystko przepuszczający filtr natomiast ma nieliniową charakterystykę fazową

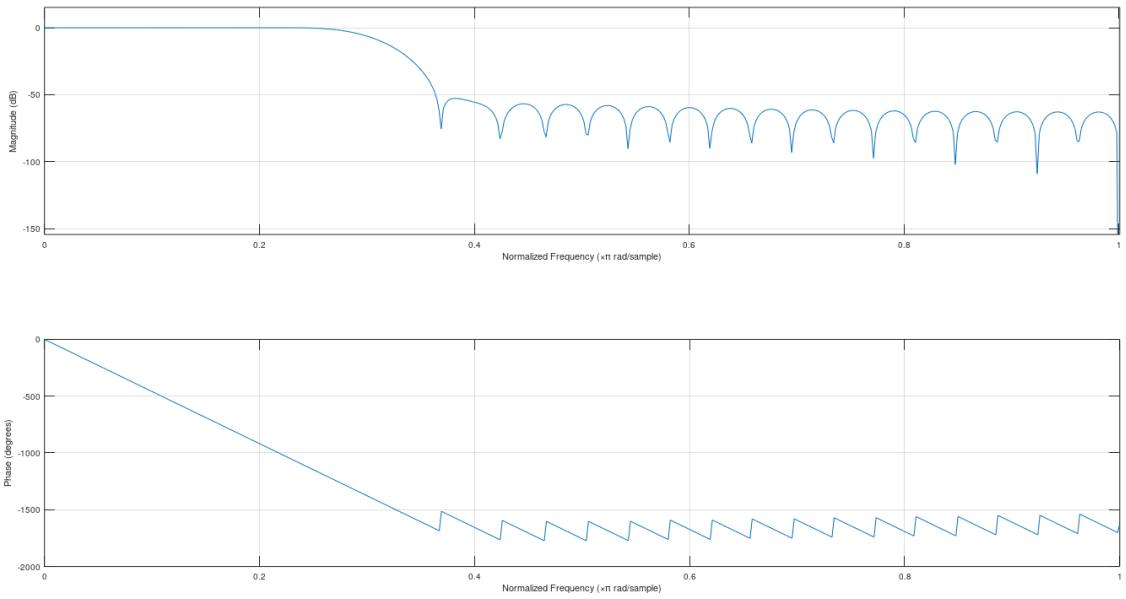
(co jest typowe dla filtrów rekursywnych IIR). Aby dowiedzieć się o jaką wartość dane częstotliwości w sygnale zostaną przesunięte należy policzyć pochodną z charakterystyki fazowej 13. Wykresy tych pochodnych (tzw. przesunięcia grupowe) zostały pokazane w trzecim wierszu.

$$\tau_g = -\frac{d\phi}{d\omega} \tag{13}$$

W ostatnim wierszu zaprezentowano efekt działania filtrów na sygnał wejściowy składający się z dwóch funkcji typu sinus o częstotliwościach odpowiednio 100 i 1100 Hz występujących jeden po drugim. Inaczej mówiąc wektor wejściowy składa się najpierw z jednej funkcji sinus w przedziale od zera do połowy wektora wejściowego, potem następuje druga funkcja sinus o większej częstotliwości. Na wykresie w dolnym lewym rogu można - z łatwościom - zaobserwować, że pierwsza połowa wektora wejściowe (o częstotliwości 100 Hz) nie została stłumiona, w przeciwieństwie do drugiej której amplituda uległa zmniejszeniu. Jednocześnie warto zauważyć, że wszystkie częstotliwości zostały przesunięte w czasie o taką samą liczbę próbek, z wykresu opóźnienia grupowego można odczytać, że tą wartością jest 26 próbek.

Nieco inaczej wygląda sytuacja na wykresie znajdującym się w dolnym prawym rogu (filtr wszystkoprzepuszczający). Żadna z częstotliwości nie uległa stłumieniu, nie powinno to dziwić zważywszy na fakt, że charakterystyka amplitudowa wynosi 0 dB w całym paśmie. Natomiast wartym zauważenia jest fakt, iż filtr ten przesuwa z fazie jedynie częstotliwości około 1100 Hz (0.22 częstotliwości Nyquista). Widać to wyraźnie na wykresie, gdzie część o niższej częstotliwości niemal w ogóle nie uległa przesunięciu, natomiast częstotliwość 1100 Hz jest wyraźnie przesunięta w fazie o około 150 próbek.

Aby narysować charakterystyki częstotliwościowe filtrów (zarówno FIR jak i IIR) w programie MATLAB/Octave można użyć funkcji `freqz(b,a)`. W najprostszym przypadku przyjmuje ona dwa wektory będące współczynnikami filtra, odpowiednio `b` to współczynniki licznika filtra, natomiast `a` to współczynniki mianownika filtra (w przypadku filtrów FIR, gdzie mianownik równa się jedności należy przekazać liczbę 1). Na rysunku 9 pokazano charakterystyki częstotliwościowe filtra typu FIR o unormowanej częstotliwości granicznej 0.3 i rzędzie równym 51. Zakładając, że `b` to wektor współczynników licznika filtra, to pokazany wykres otrzymano wywołując funkcję `freqz(b,1)`.

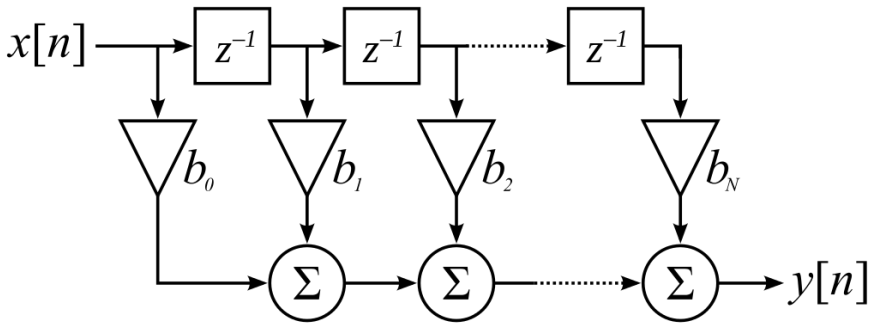


Rysunek 9: Charakterystyki częstotliwościowe dla filtru typu FIR o unormowanej częstotliwości granicznej 0.3 i rzędzie równym 51

3.2 Filtry o skończonej odpowiedzi impulsowej FIR

Filtry nierekursywne FIR nie posiadają sprzężenie zwrotnego. Każda próbka wyjściowa jest sumą ważoną próbek z poprzednich chwil czasu 14. Na rysunku nr 10 pokazano schemat blokowy filtra typu FIR. Zaznaczone na nim kwadraty z z^{-1} oznaczają opóźnienie próbki o wartość jeden w dziedzinie operatora Z.

$$y[n] = b_0 x[n] + b_1 x[n-1] + ... + b_N[x-N] = \sum_{i=0}^{N} b_i \cdot x[n-i] \tag{14}$$



Rysunek 10: Bezpośrednia forma dyskretnoczasowego filtru FIR rzędu $N+1$.

Filtry FIR można projektować na wiele analitycznych sposobów. Projektowanie filtrów sprawdza się do wyznaczenia wartości współczynników wagowych $b_0, b_1, ...b_N$. Jedną z prostszych metod, z której korzysta funkcja `fir1` programu MATLAB/Octave jest metoda okien. Jest ona stosunkowo prosta pod względem zarówno teoretycznym jak i implementacyjnym. Analitycznie wzory na różne rodzaje filtrów FIR wyprowadza się z odpowiedzi impulsowej „idealnego" filtru dolnoprzepustowego. Pozostałe filtry mogą być obliczone jako superpozycja filtrów dolnoprzepustowych o różnych częstotliwościach granicznych oraz filtra „wszechprzepustowego" [3]. Na tych zajęciach nie zgłębiając teoretycznych aspektów projektowania filtrów typu FIR, wykorzystana zostanie wspomniana funkcja `fir1()` programu MATLAB/Octave. Pozwala ona na wyznaczenie współczynników filtra wcześniej wspomnianą metodą okien.

Poniżej został pokazany listing 20 ukazujący filtrację sygnału świergotowego[4]. Ten rodzaj modulowanego sygnału idealnie nadaje się do pokazania idei samej filtracji. Częstotliwość zmienia się w nim liniowo i po jego wymnożeniu z odpowiedzią impulsową filtra cześć sygnału powyżej pewnej częstotliwości zostanie wytłumiona. Do stworzenia takiego sygnału można wykorzystać funkcję `chirp(t, f0, t1,f1)`, gdzie `t` jest wektorem czasu, `f0` częstotliwością początkową w chwili $t = 0$, `t1` chwilą czasu, w której sygnał osiąga częstotliwość `f1`. Do wyznaczenia współczynników filtra typu FIR wykorzystano funkcję `fir1(n, wn)`. Przyjmuje ona minimum dwa argumenty, pierwszym (`n`) jest rząd filtra, drugim natomiast unormowana względem częstotliwości Nyquista częstotliwość odcięcia. W poniższym przykładzie częstotliwość próbkowania F_s została ustawiona na 12000 Hz, częstotliwość Nyquista w tym przypadku wynosi połowę F_s t.j. $F_n = \frac{F_s}{2} = 6000\ Hz$. Chcąc otrzymać współczynniki filtru, który odfiltruje częstotliwości sygnału większe niż 1500 Hz, należy jako argument `wn` przekazać liczbę 0.25, ponieważ $6000 \cdot 0.25 = 1500\ Hz$.

Aby wymnożyć sygnał filtrowany ze współczynnikami filtra, najlepiej użyć funkcji `filter(b,a,y)`, gdzie `b` to kolejne wagi licznika filtra, `a` to kolejne wagi mianownika filtra, `y` filtrowany wektor próbek. W poniższym przypadku drugi argument wyniósł 1 ponieważ filtr typu FIR nie posiada mianownika, współczynników rekursywnych. W przypadku systemów FIR możliwe jest również wykorzystanie funkcji liczącej splot `conv(a,b)` 1.3 [5].

```
Fs = 12000;              % Sampling frequency
t = 0:1/Fs:1;            % Time vector
y = chirp(t,0,1,3000);   % Generate samples of linear swept-frequency
 cosing signal. 0 frequency at t=0 to 3 kHz at t=1

subplot(2,1,1);
plot(t,y);
title('Input chirp signal with frequency changing from 0 to 3 kHz');
xlabel('Time [s]')
ylabel('Amplitude');

n = 100;
```

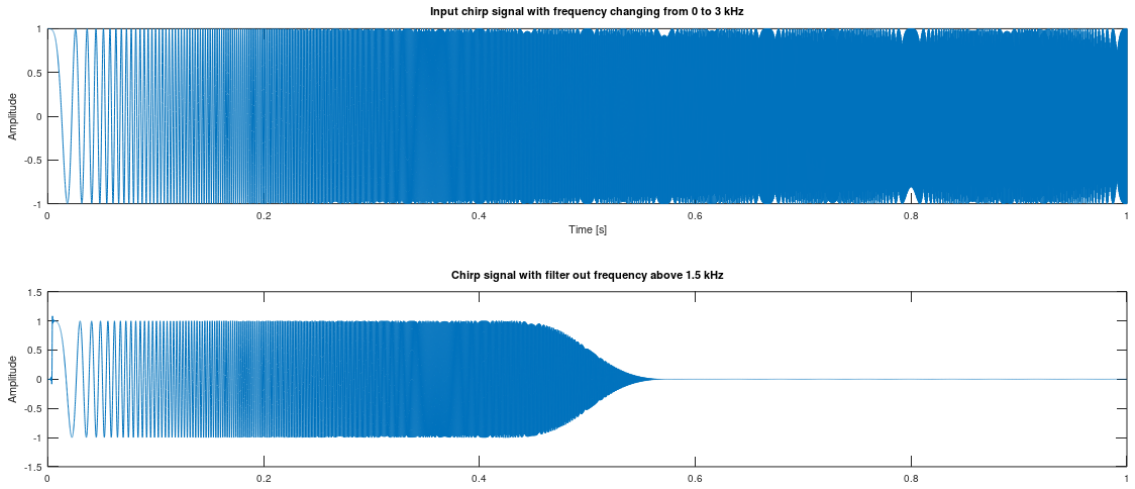
[3] Filtr „wszechprzepustowy" posiada charakterystykę amplitudową, która jest równa jedności w całym paśmie. Może posłużyć do modyfikacji fazy sygnału bez wpływu na jego amplitudę. Jego odpowiedź impulsowa jest deltą Kroneckera $H(e^{j\Omega}) = 1$.

[4] Sygnał świergotowy jest sygnałem, w którym częstotliwość zmienia się (rośnie albo maleje) liniowo w czasie

[5] Funkcja `filter` jest odpowiednią funkcją zarówno dla filtrów FIR jak i IIR. Funkcja licząca splot `conv` pobiera dwa wejścia i zwraca ich splot, dlatego w tym przypadku `conv(x,y)` i `filter(h,1,x)` zwrócą ten sam wynik. Jedynka w filtrze mówi nam o tym, że współczynnik rekursywny filtru wynosi jeden. Jeśli jednak mamy do czynienia z filtrem IIR to nie możemy wykorzystać `conv`.

```
wn = 0.25;                  % Nyquist frequency Fs/2 = 6000. Cut off
 frequency: 6000*0.25 = 1500 Hz
b = fir1(n, wn);            % Create  n order lowpass filter coefficient
yFiltered = filter(b,1,y);%Use filter function to filterout y signal
subplot(2,1,2);
plot(t,yFiltered);
title('Chirp signal with filter out frequency above 1.5 kHz');
xlabel('Time [s]')
ylabel('Amplitude');
```

Listing 20: Filtracja sygnału świergotowego zmieniającego liniowo częstotliwość w przedziale od 0 do 3 kHz



Rysunek 11: Filtracja sygnału świergotowego, zmieniającego liniowo częstotliwość w przedziale od 0 do 3 kHz

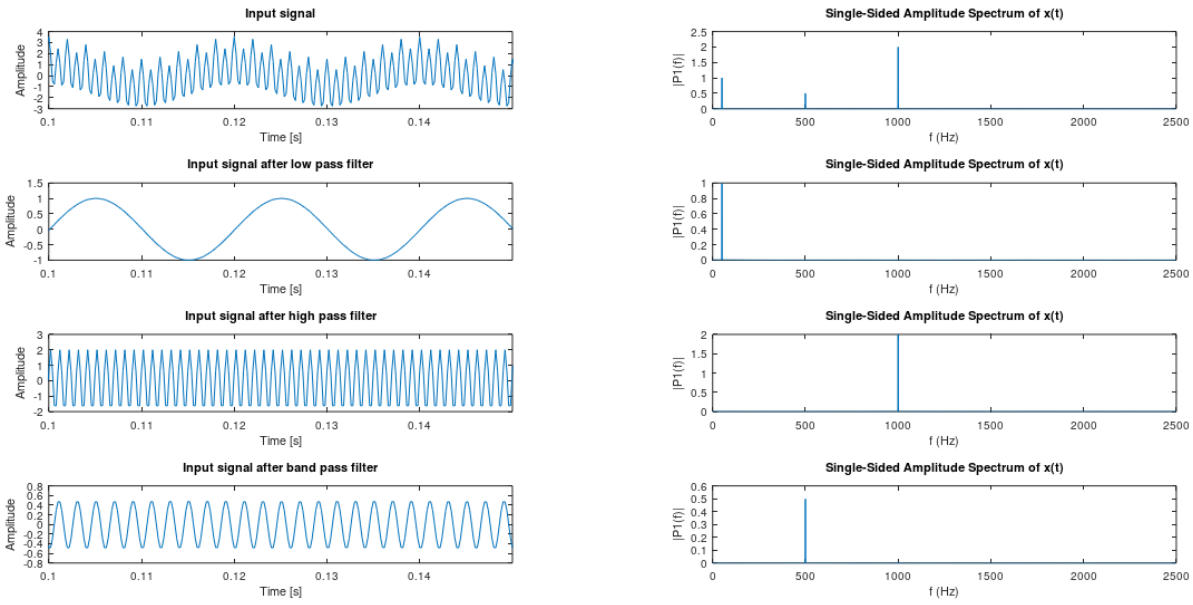
Na listingu 21 pokazano sposób projektowania filtrów nierekursywnych (FIR) odpowiednio dolnoprzepustowego, górnoprzepustowego oraz środkowoprzepustowego. W pierwszej kolejności utworzono sygnał składający się z trzech funkcji kosinus o częstotliwościach odpowiednio 50, 500, 1000 $[Hz]$, częstotliwość próbkowania w tym przykładzie została ustawiona na 5000 Hz. Sygnał poddawany filtracji pokazano na rysunku 12. W pierwszym wierszu po lewej stronie sygnał został pokazany w dziedzinie czasu, po prawej natomiast w dziedzinie częstotliwości po wykonaniu dyskretnego przekształcenia Fouriera za pomocą funkcji `fft` - uwaga funkcja `plotFFT` nie jest domyślną funkcją programu MATLAB/Octave, jest to funkcja napisana podczas drugich laboratoriów 18).

Następnie z wykorzystaniem funkcji `fir1(n, Wn)` obliczono współczynniki filtrów odpowiednio dolnoprzepustowego (tłumione są częstotliwości przekraczające $0.1 \cdot f_N = 250\ Hz$). Dalej obliczono współczynniki filtra górnoprzepustowego (tłumione są częstotliwości poniżej $0.3 \cdot f_N = 750\ Hz$). Warto zwrócić uwagę na przekazanie do funkcji `fir1` dodatkowego trzeciego argumentu `'high'`, który mówi, że chcemy obliczyć współczynniki filtra górnoprzepustowego (domyślnie przyjmowany jest argument `'low'`). Na koniec pokazano filtr środkowoprzepustowy w przypadku którego konieczne jest przekazanie wektora zawierającego przedział częstotliwości, które będą „przepuszczane" przez filtr w tym przypadku `[0.1 0.3]` (stłumiona zostaną wszystkie częstotliwości poza przedziałem $f \in [250,\ 750]\ Hz$).

```
Fs = 5000;              % Sampling frequency
T = 1/Fs;               % Sampling period
L = 5000;               % Signal length
t = (0:L-1)*T;          % Time vector
y = 1.0*cos(2*pi*50*t) + 0.5*cos(2*pi*500*t) + 2*cos(2*pi*1000*t); %
 Prepare signal
subplot(4,2,1); plot(t, y);
title('Input signal');xlabel('Time [s]');
ylabel('Amplitude'); axis([0.1 0.15]);

% Fourier transform of y
subplot(4,2,2); plotFFT(y, Fs);
```

```
12
13
14  % Filter out higher frequency
15  n = 51;
16  Wn = 0.1;
17  b = fir1(n,Wn);
18  yLP = filter(b,1,y);
19  subplot(4,2,3); plot(t, yLP);
20  title('Input signal after low pass filter');xlabel('Time [s]');
21  ylabel('Amplitude'); axis([0.1 0.15]);
22  subplot(4,2,4); plotFFT(yLP, Fs);
23
24  % Filter out lower frequency
25  n = 51;
26  Wn = 0.3;
27  b = fir1(n,Wn, 'high');
28  yHP = filter(b,1,y);
29  subplot(4,2,5); plot(t, yHP);
30  title('Input signal after high pass filter');xlabel('Time [s]');
31  ylabel('Amplitude'); axis([0.1 0.15]);
32  subplot(4,2,6); plotFFT(yHP, Fs);
33
34  % Filter out lower ang higher frequency (band pass filter)
35  n = 51;
36  Wn = [0.1 0.3];
37  b = fir1(n,Wn);
38
39  yBP = filter(b,1,y);
40  subplot(4,2,7); plot(t, yBP);
41  title('Input signal after band pass filter');xlabel('Time [s]');
42  ylabel('Amplitude'); axis([0.1 0.15]);
43  subplot(4,2,8); plotFFT(yBP, Fs);
```

Listing 21: Filtracja sygnału poszczególnych składowych z sygnały $y(t)$



Rysunek 12: Filtracja poszczególnych składowych sygnału $y(t)$

3.3 Filtry o skończonej odpowiedzi impulsowej IIR

Filtry typu IIR są systemami rekursywnymi tzn. do obliczenia wyjścia filtra wykorzystywane są próbki wejściowe (jak w filtrach FIR) oraz poprzednie próbki wyjściowe. Inaczej mówiąc filtry te posiadają sprzężenie zwrotne. Wzór na n-tą próbkę wyjściową filtru IIR został pokazany poniżej 15.

$$y[n] = \frac{1}{a_0}(b_0x[n] + b_1x[n-1] + ... + b_Px[n-P] - a_1y[n-1] - a_2y[n-2] - ... - a_Qy[n-Q]) \tag{15}$$

,gdzie: P jest rzędem wielomianu licznika, Q jest rzędem wielomianu mianownika.

Do najważniejszych zalet filtrów IIR należy zaliczyć fakt, iż specyfikacje wymagane od filtra (np. stromość zbocza) udaje się osiągnąć układem o znacznie niższym rzędzie niż w przypadku układów nierekursywnych. Wśród wad należy nadmienić, że sprzężenie zwrotne przy złym zaprojektowaniu filtru może powodować utratę jego stabilności. Dodatkowo w przeciwieństwie do filtrów FIR nie posiadają one liniowej charakterystyki fazowej. Oznacza to, że takie filtry różnie przesuwają w fazie różne częstotliwości. W wielu układach warunek liniowej charakterystyki fazowej jest wymagany np. przy sygnałach związanych z urządzeniami medycznymi.

Klasyczne filtry IIR są projektowane z wykorzystaniem ich analogowych prototypów. W programie MATLAB/Octave typowe filtry analogowe takie jak filtr Butterwortha, Czebyszewa typu I, Czebyszewa typu II czy filtr eliptyczny za pomocą odpowiednich transformacji mogą być wykorzystane do wyznaczenia cyfrowych odpowiedników. Do obliczenia współczynników wspomnianych filtrów należy skorzystać z funkcji odpowiednio: `butter()`, `cheby1()`, `cheby2()`, `ellip()`. Proces obliczania współczynników filtrów jest analogiczny jak w przypadku filtrów FIR 22.

```
[b,a] = butter(5,0.4);                      % Lowpass Butterworth
[b,a] = cheby1(4,1,[0.4 0.7]);              % Bandpass Chebyshev Type I
[b,a] = cheby2(6,60,0.8,'high');            % Highpass Chebyshev Type II
[b,a] = ellip(3,1,60,[0.4 0.7],'stop');     % Bandstop elliptic
```

Listing 22: Przykłady filtrów rekursywnych

Filtr Butterworth'a charakteryzuje się możliwe najbardziej płaską charakterystyką amplitudową w paśmie przenoszenia kosztem mniejszego niż inne filtry zbocza w paśmie przejściowym. Filtr Czebyszewa typu I posiada zafalowania w paśmie przepustowym oraz płaski przebieg w paśmie zaporowym. Natomiast filtry Czebyszewa typu II posiada płaską charakterystykę w paśmie przepustowym i zafalowania w paśmie zaporowym. W obu przypadkach (typ I oraz II) filtr posiada strome zbocze w paśmie przejściowym. Identyczne zafalowania z paśmie przepustowym i zaporowym ma natomiast filtr eliptyczny, który jednocześnie charakteryzuje się niemal pionowym zboczem w paśmie przejściowym.

3.4 Zadania

3.4.1 Filtracja sygnału świergotowego

Analogicznie jak zostało pokazane na listingu nr 20 utwórz sygnał świergotowy o zmieniającej się liniowo częstotliwości w przedziale $f \in [0, 4000]$ Hz. Częstotliwość próbkowania F_s możne zostać ustawiona na wartość 10000 Hz.

Utwórz **filtr górnoprzepustowy**, który odfiltruje wszystkie częstotliwości w przedziale $f \in [0, 1000]$ Hz. Aby uzyskać współczynniki filtra górnoprzepustowego wywołaj funkcję `fir1`, z dodatkowym trzecim argumentem `'high'`: `b = fir1(n, wn,'high')`. Wyniki przedstaw na współdzielonym wykresie, analogicznie jak zostało to pokazane na rysunku nr 20.

3.4.2 Filtracja poszczególnych składowych sygnału

Podobnie jak na listingu 21 utwórz sygnał składający się z trzech składowych o częstotliwościach odpowiednio 250, 750, 1500 Hz i dowolnych amplitudach. Do tego sygnału dodaj szum biały o rozkładzie normalnym (`y= ... + randn(1,length(t))`).

Następnie wykorzystując funkcję `fir1()` usuń z sygnału odpowiednio:

- składową o częstotliwości 250 Hz,
- składową o częstotliwości 1500 Hz,
- składową o częstotliwości 750 Hz (stwórz filtr środkowozaporowy, analogicznie jak filtr środkowoprzepustowy pokazany na listingu 21, ale z dodanym trzecim argumentem `'stop'` np. `fir1(n, Wn, 'stop')`).

Wykresy umieść na współdzielonym wykresie analogicznie jak pokazano na rysunku 12. Zwróć uwagę, że szum biały został usunięty jedynie w obszarach pasma zaporowego tzn. dla filtru dolnoprzepustowego jedynie od częstotliwości granicznej, leżącej gdzieś powyżej 250 Hz, w zależności od zaprojektowanego filtru. Proszę mieć na uwadze, że szum biały składa się z częstotliwości znajdujących się w całym zakresie widma.

3.4.3 Stromość pasma przejściowego, a długość filtra FIR

Oblicz współczynniki trzech filtrów dolnoprzepustowych o częstotliwości granicznej `Wn=0.5` oraz o różnych rzędach np.:

```
b1 = fir1(5, 0.5);
b2 = fir1(25, 0.5);
b3 = fir1(50, 0.5);
```

Następnie narysuj po kolei charakterystyki częstotliwościowe tych filtrów za pomocą funkcji `freqz(b1,1)`, `freqz(b2,1)` oraz `freqz(b3,1)`. Zwróć uwagę, że im wyższy rząd filtru tych bardziej wąskie pasmo przejściowe (pomiędzy pasmem przepustowym, a zaporowym). Warto również zauważyć, iż im wyższy rząd filtru tym bardziej strome nachylenie liniowej charakterystyki fazowej, co przekłada się na większe przesunięcie fazowe filtru o wyższym rzędzie. Aby narysować te charakterystyki na współdzielonym wykresie (`subplot`) można posłużyć się poniższym fragmentem kodu.

```
[y,x] = freqz(b1,1);
subplot(311)
plot(x./pi, 10*log10(abs(y)));grid on; %x./pi because of normalization,
   10*log10() because of logarithmic scale (in dB)
%...
```

3.4.4 Sprawdzenie charakterystyk amplitudowych filtrów rekursywnych IIR

Za pomocą funkcji programu MATLAB/Octave wyznacz charakterystyki amplitudowe kilku filtrów rekursywnych. W pierwszej kolejności sprawdź odpowiedzi częstotliwościowe filtra Butterwortha dla różnych wartości rzędu np. 5, 10, 15, częstotliwość odcięcia można ustawić na połowę częstotliwości Nyquista. Aby lepiej zaobserwować falowania w paśmie przejściowym/zaporowym oraz stromość zbocza przejściowego, charakterystyki można narysować w skali liniowej, w przeciwieństwie do poprzedniego zadania, gdzie została użyta skala logarytmiczna.

```
[b1,a1] = butter(5,0.5);
[y1,x1] = freqz(b1,a1);
subplot(311)
plot(x1./pi, abs(y1));grid on; %x./pi because of normalization

[b2,a2] = butter(10,0.5);
[y2,x2] = freqz(b2,a2);
subplot(312)
plot(x2./pi, abs(y2));grid on; %x./pi because of normalization

%...
```

Funkcje programu MATLAB/Octave wyznaczające współczynniki filtrów Czebyszewa zarówno typu I jak i II posiadają większą liczbę parametrów niż funkcja `butter`. Pierwszym parametrem jest rząd filtru drugim natomiast dopuszczalne zafalowania w paśmie przepustowym (filtr typu I) albo dopuszczalne zafalowania w paśmie zaporowym (filtr typu II) wyrażone w decybelach. Jako trzeci parametr przekazywana jest częstotliwość/ci odcięcia. Sprawdź jak zmienia się charakterystyka amplitudowa w zależności od przekazywanych parametrów. Zbadaj dla każdego typu filtra dwie wartości rzędu np. 5 i 10 oraz różne wartości dopuszczalnych zafalowań np. dla filtru typu I (0.1 oraz 0.2) i dla filtru typu II (20 oraz 40).

```
% Chebyshev filter type I
[b1,a1] = cheby1(5,0.1,0.5);
[y1,x1] = freqz(b1,a1);
subplot(411)
plot(x1./pi, abs(y1));grid on; %x./pi because of normalization
```

```
6
7  [b2,a2] = cheby1(10,0.2,0.5);
8  [y2,x2] = freqz(b2,a2);
9  subplot(412)
10 plot(x2./pi, abs(y2));grid on; %x./pi because of normalization
11
12 % Chebyshev filter type II
13 %...
```

Jako ostatnią sprawdź charakterystykę amplitudową filtra eliptycznego (funkcja `[b,a] = ellip(n,Rp,Rs,Wp)`), gdzie `n` to rząd filtru, `Rp` zafalowania w paśmie przepustowym, `Rs` zafalowania w paśmie zaporowym, `Wn` to natomiast częstotliwość odcięcia. Sprawdź analogicznie jak w przypadku filtra Czebyszewa dwie wartości rzędu filtra oraz dwie różne wartości dopuszczalnych zafalowań. Zwróć uwagę na bardzo strome zbocze tej charakterystyki.

```
1  % Elliptic filter
2  [b1,1] = ellip(6,5,40,0.6);
3  [y1,x1] = freqz(b1,a1);
4  subplot(411)
5  plot(x1./pi, abs(y1));grid on; %x./pi because of normalization
6  %...
```

4 Laboratorium nr 4

4.1 Interpolacja elementów wektora

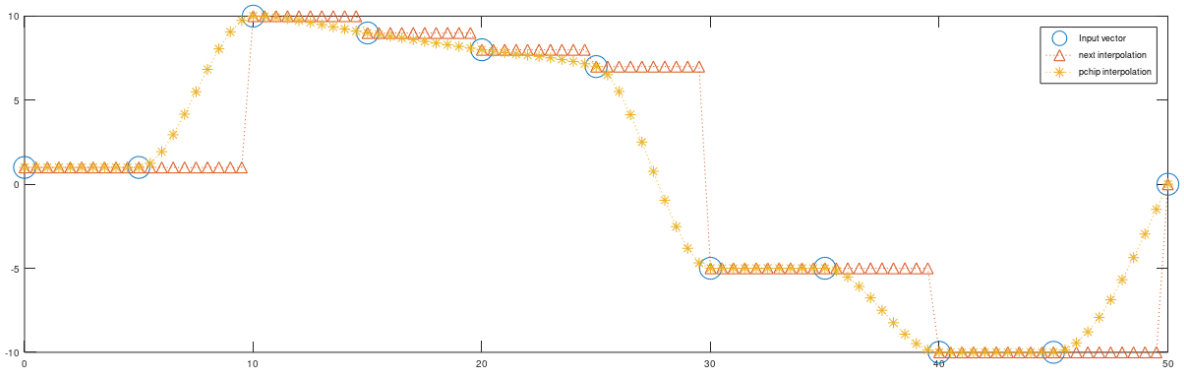
Zdarza się, że posiadamy wektor danych pomiarowych w pewnych chwilach czasu, jednak konieczne jest wyznaczenie wartości dla innej chwili czasu. Np. posiadamy wektor czasu `[0 0.1 0.2 0.4]` i odpowiadające jemu wartości `[2 3 4 6]`, natomiast konieczna jest estymacja wartości dla chwili `0.3`. Może też istnieć potrzeba zagęszczenia tablicy pomiarów albo obliczenia wartości, która z jakiś powodów jest niewiarygodna bądź niedostępna.

Interpolacja jest metodą numeryczną, która pozwala rozwiązać powyższe problemy. Umożliwia ona estymowanie wartości wektora/funkcji dla niezdefiniowanych argumentów. Funkcją programu MATLAB/GNU Octave, która dokonuje interpolacji jednowymiarowego wektora danych jest `vq = interp1(x,v,xq)`, gdzie: `x` jest wektorem czasu (oś X) próbek wejściowych, `v` jest zbiorem wartości tych próbek (oś Y), natomiast `xq` jest nowym wektorem czasu dla którego funkcja wyznacza wartości i zwraca wynik w postaci wektora `vq`. Jako czwarty argument można przekazać rodzaj metody interpolującej. Dostępne są następuje typy interpolacji:

- `'linear'` interpolacja liniowa,
- `'nearest'` interpolacja schodkowa wartością najbliższą,
- `'next'` interpolacja schodkowa wartością następną,
- `'previous'` interpolacja schodkowa wartością poprzednią,
- `'pchip'` interpolacja sklejanymi wielomianami Hermite'a trzeciego rzędu,
- `'spline'` jak wyżej z zaznaczeniem, że interpolujące wielomiany są gładsze.

```
t = 0:5:50; % input time vector
y = [1 1 10 9 8 7 -5 -5 -10 -10 0]; % values for input vector

t1 = 0:.5:50; % query vector
y1 = interp1(t,y,t1, 'previous');  % zero order hold
y2 = interp1(t,y,t1, 'pchip'); % piecewise cubic interpolation

% Plot the results
plot(t,y,'o','MarkerSize',12);
hold on;
plot(t1,y1,':^');
plot(t1,y2,':*');
legend('Input vector','next interpolation', 'pchip interpolation');
```

Listing 23: Interpolacja danych pomiarowych



Rysunek 13: Wykres porównujący dwie metody interpolacji wektora danych wejściowych

Na powyższym listingu 23 pokazano przykładowe użycie funkcji `interp1`. Utworzono wejściowy wektor czasu ze skokiem co 5 próbek oraz wektor wartości odpowiadających temu wektorowi czasu. Następnie

dodano nowy wektor czasu z krokiem `0.5`. Posiada on zatem dziesięć razy więcej elementów. Za pomocą funkcji `interp1` wyznaczono wartości w tych punktach. Nowy wektor posiada zatem dodatkowe próbki z wartościami estymowanymi na podstawie wektora wejściowego.

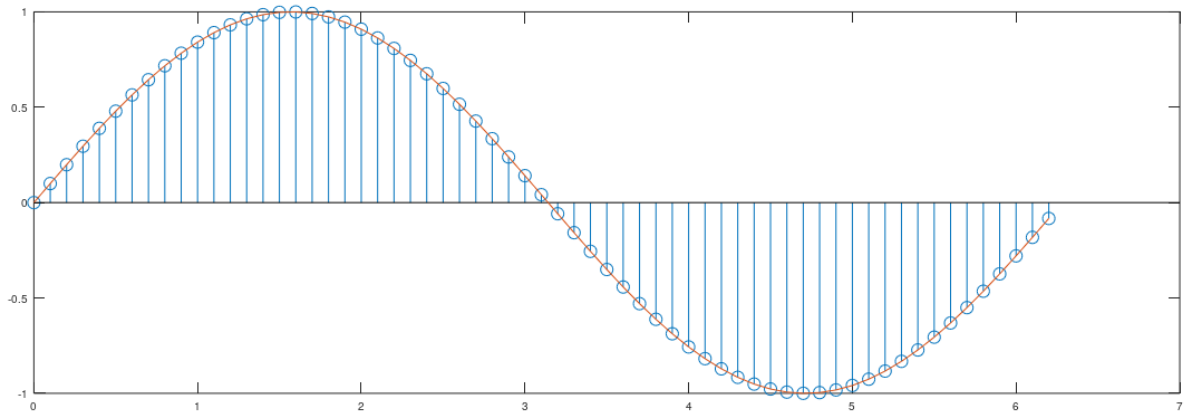
Na rysunku 14 pokazano wyniki tej interpolacji. Wejściowy sygnał zaznaczono niebieskimi okręgami. Widać, że tych pomiarów jest niewielka liczba. Następne czerwonymi trójkątami przedstawiono wyniki dla nowego, zagęszczonego wektora czasu, które zostały interpolowane przez podtrzymanie wcześniejszej wartości do momentu pojawienia się nowej próbki (tzw. interpolator zerowego rzędu). Żółtymi gwiazdkami pokazano drugą metodę estymującą wartości za pomocą wielomianów trzeciego rzędu.

4.2 Próbkowanie i tor przetwarzania sygnałów

Podczas wcześniejszych zajęć laboratoryjnych sygnały, które były poddawane analizie i przetwarzaniu (np. w postaci filtracji) były ciągłymi sygnałami czasu dyskretnego. Na listingu 24 pokazano tworzenie wektora danych, tak jak było to robione na poprzednich laboratoriach. Mimo, że funkcja `plot` standardowo rysuje wykres łącząc kolejne próbki ciągłą linią, dalej są to jedynie wartości funkcji w chwilach wyznaczonych przez wektor czasu 14.

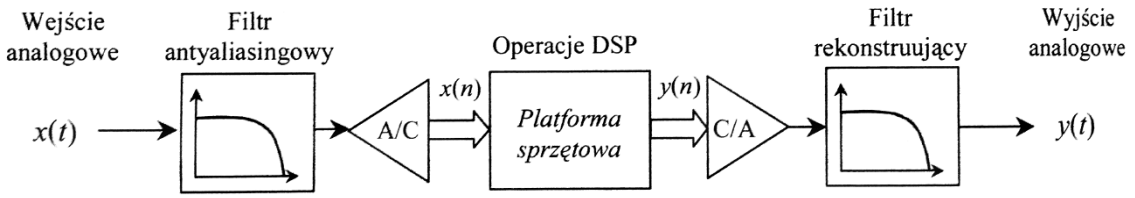
```
t = 0:.1:2*pi; % time vector
y = sin(t); % value vector
stem(t,y); % stem doesn' connect vector samples using straigh lines
hold on;
plot(t,y); % regular plot which connect samples using straigh lines
```

Listing 24: Ciągły sygnał czasu dyskretnego



Rysunek 14: Prezentacja próbek sygnału za pomocą funkcji `stem` oraz `plot`

Na rysunku 15 pokazano standardowy tor przetwarzania sygnału analogowego przez układ cyfrowy. W pierwszej kolejności ciągły, analogowy sygnał (np. przebieg napięcia, sygnału audio) jest przy pomocy przetwornika analogowo-cyfrowego przetwarzany na sygnał ciągły czasu dyskretnego tzn. na ciąg próbek sygnału (proces ten nazywany jest **próbkowaniem**). Takie, już wynikowe, sygnały były tworzone podczas laboratoriów. Dalej sygnał może być analizowany i przetwarzany poznanymi wcześniej metodami. Po tych operacjach możliwe jest ponowne przekształcenie sygnału na postać analogową (ciągłą) przy pomocy przetwornika cyfrowo-analogowego.



Rysunek 15: Schemat cyfrowego przetwarzania sygnałów [1]

4.3 Twierdzenie o próbkowaniu

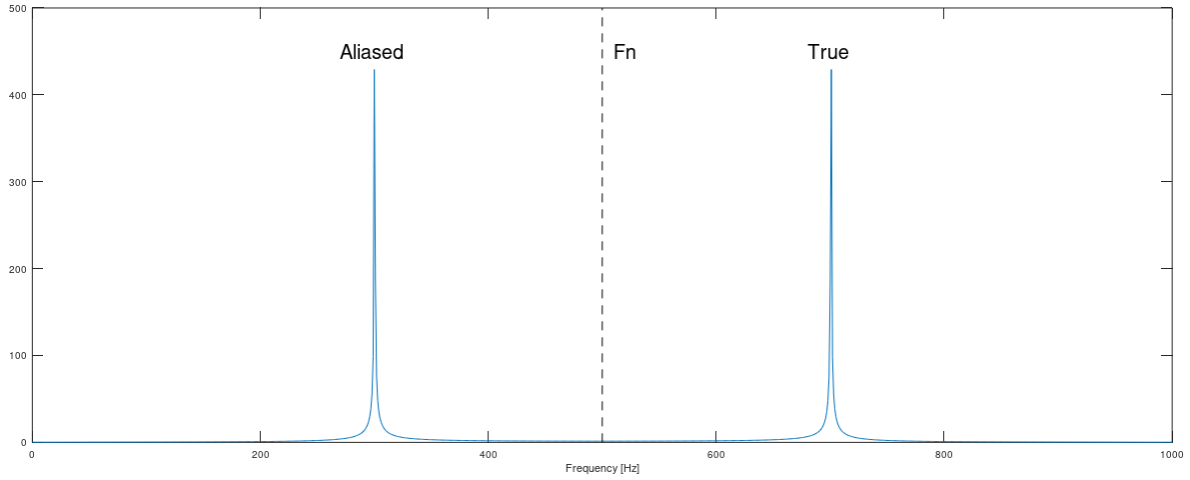
Twierdzenie o próbkowaniu mówi jaka musi być częstotliwość próbkowania, aby możliwe było późniejsze wierne odtworzenie/zrekonstruowanie próbkowanego sygnału bez zniekształceń. Jest to bardzo istotne twierdzenie zarówno w kontekście przetwarzania sygnałów jak i teorii sterowania. Jak zostało wcześniej wspomniane próbkowanie jest realizowane przez przetwornik analogowo-cyfrowy, który na diagramie został zaznaczony jako A/C 15.

Najprościej mówiąc twierdzenie to mówi, że funkcja/sygnał nie zawierająca częstotliwościowi większych niż B może być wiernie odtworzona jeżeli częstotliwości próbkowania wynosi $f_s > 2 \cdot B$ 16. Inaczej mówiąc, jeśli chcemy odtworzyć wszystkie częstotliwości w sygnale, musimy go próbkować z dwa razy większą częstotliwością niż częstotliwość najwyższej składowej. Jest to powód dlaczego sygnały audio są kodowane z częstotliwością 48 kHz (jakość DVD). Ludzki słuch posiada zakres słyszalnych częstotliwości w zakresie od 16 do 20000 Hz. Aby móc odtworzyć maksymalną częstotliwość jaką człowiek jest w stanie usłyszeć należy próbkować z częstotliwością dwa razy większą niż najwyższa składowa, w tym przypadku 20 kHz.

$$B < \frac{f_s}{2} \tag{16}$$

Załóżmy, że nasz sygnał zawiera jedną składową o danej częstotliwości. Potrzeba minimum dwóch pomiarów pomiędzy pojedynczym okresem, aby odtworzyć tę składową. Jeśli będziemy próbkować rzadziej to inna funkcja o niższej częstotliwości będzie mogła interpolować te punkty. Takie niepożądane zjawisko nazywane jest aliasingiem. Aliasing sprawia, że tracimy w sygnale informacje o składowych, które mogą nas interesować.

Na rysunku 16 pokazano widmo sygnału sinusoidalnego o częstotliwości 700 Hz, który został spróbkowany z częstotliwością równą 1000 Hz, warunki twierdzenia o próbkowaniu w tym przypadku nie zostały zachowane. Składowa sygnału jest wyższa niż połowa częstotliwości próbkowania, która nazywana jest częstotliwością Nyquista ($f_N = \frac{f_s}{2} = 500\ Hz$). W wyniku zjawiska aliasingu została ona odbita/„zawinięta" względem f_N. Przy próbie rekonstrukcji tego sygnału np. przez przetwornik cyfrowo-analogowy zaobserwowana częstotliwość będzie wynosić 300 Hz.



Rysunek 16: Widmo sygnału sinusoidalnego o częstotliwości 700 Hz próbkowanego z częstotliwością 1000 Hz.

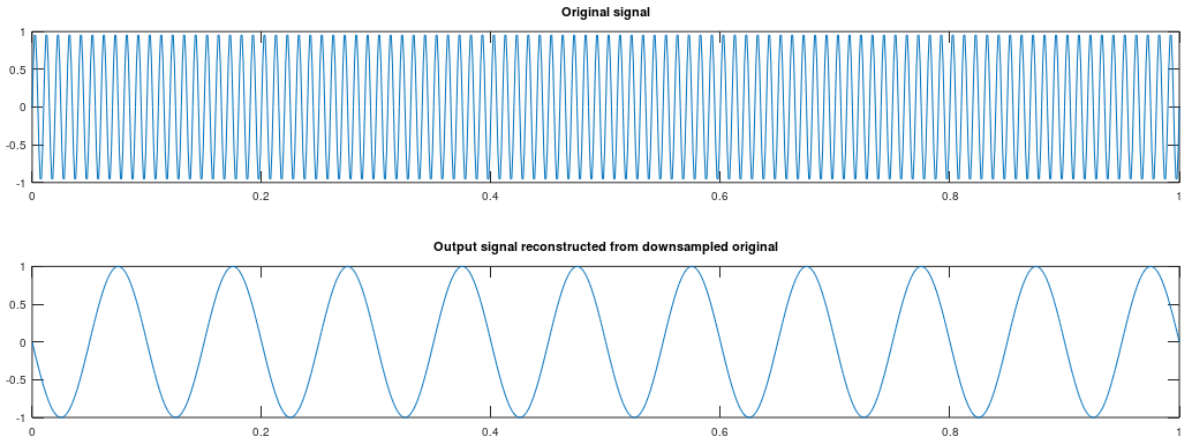
Na listingu 26 zademonstrowano w programie MATLAB/GNU Octave zmianę częstotliwości próbkowania. Najpierw stworzono wektor danych sygnału sinusoidalnego o częstotliwości 100 Hz próbkowanego z częstotliwością 1000 Hz, zgodnie z twierdzeniem o próbkowaniu możliwe jest jego wierne odtworzenie w postać analogową. Następnie dodano nowy wektor czasu tym razem z odstępami pomiędzy próbkami $\frac{1}{110}$, czyli w tym przypadku częstotliwość próbkowania wyniosła 110 Hz. Dalej wyznaczono wektor wartości funkcji sinusoidalnej o częstotliwości 100 Hz tak jak poprzednio z tą różnicą, że tym razem dla nowego wektora czasu ze znacznie mniejszą liczbą argumentów. Na koniec podjęto próbę interpolacji (rekonstrukcji) sygnału tak, aby ponownie uzyskać próbki pomiędzy węzłami (próbkami) drugiego z sygnałów. Inaczej mówiąc chcieliśmy na podstawie wektora z mniejszą liczbą próbek wyznaczyć wartości sygnału w chwilach zdefiniowanych przez pierwszy wektor czasu (z próbkami rozmieszczonymi co $\frac{1}{1000}$).

```
1 Fs = 1000;
2 t1=0:1/Fs:1;
```

```
y1=sin(2*pi*100*t1); % sinusoidal signal with 100 Hz frequency

Fs2 = 110;
t2=0:1/Fs2:1; % simulation of sampling with frequency of 100 Hz
y2 = sin(2*pi*100*t2);
vq2 = interp1(t2,y2,t1,'spline');

subplot(211);
plot(t1,y1); title('Original signal'); % original signal
subplot(212)
plot(t1,vq2); title('Output signal reconstructed from downsampled original
    '); % output signal reconstructed from downsampled original
```

Listing 25: Symulacja niedostatecznie wysokiej częstotliwości próbkowania

Poniżej 17 pokazano przebiegi wynikowe omawianego wyżej kodu źródłowego. Widać, że przez niedostateczne próbkowanie po rekonstrukcji sygnał wejściowy ma zdecydowanie niższą częstotliwość, został zniekształcony przez proces konwersji i zbyt niską częstotliwość próbkowania. Analizując sygnał w dziedzinie częstotliwości należałoby oczekiwać prążka dla argumentu 100 Hz. W związku, że częstotliwość próbkowania została obniżona do 110 Hz, „prążek" na widmie sygnału znalazł się po prawej stronie częstotliwości Nyquista i został odbity symetrycznie względem niej. W wyniku czego „prążek" pojawił się w częstotliwości 10 Hz, której pierwotnie nie było w sygnale.



Rysunek 17: Oryginalny sygnał o częstotliwości 100 Hz po próbkowaniu z niedostateczną częstotliwością po rekonstrukcji został zniekształcony.

Efekt aliasingu można zaobserwować również na nagraniach z kamery wideo, jeśli nagrywana osoba posiada jakiś wzór na obraniach (szachownicę, siatkę, drobne prążki) to kamera może powodować efekt aliasingu podczas poruszania się tej osoby (efekt błyszczenia). To zjawisko można również zauważyć na poniższym zdjęciu 18. Przy zmniejszeniu rozmiaru strony w pliku PDF np. przez użycie kółka myszy z wciśniętym przyciskiem CTRL, widać pojawiające się artefakty.

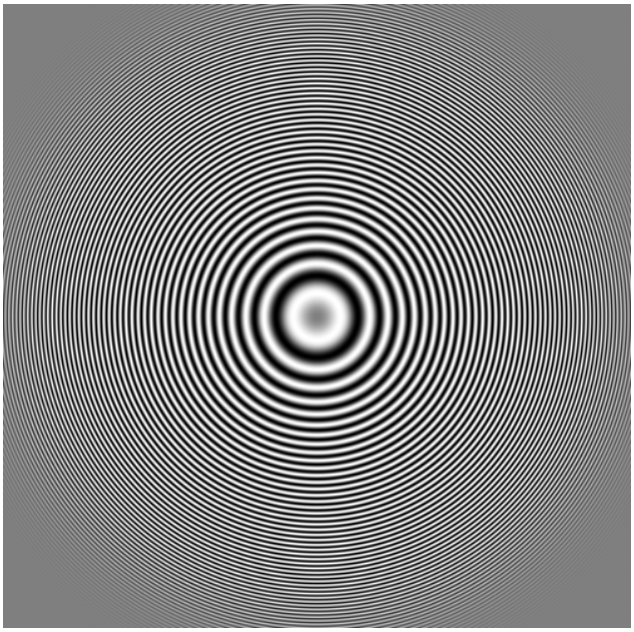
4.4 Zadania

4.4.1 Interpolacja wektora danych pomiarowych

Na listingu 23 pokazano przykład interpolacji danych pomiarowych za pomocą dwóch metod: interpolatorem zerowego rzędu (ang. Zero Order Hold) oraz sklejanymi wielomianami Hermite'a trzeciego rzędu. Przetestuj wszystkie wymienione w podpunkcie 4.1 metody wywołując funkcję `interp1` ze zmienionym czwartym argumentem, w którym przekazywana jest metoda interpolacji.

4.4.2 Widmo sygnału dla różnych częstotliwości próbkowania

Utwórz sygnał sinusoidalny o częstotliwości 500 Hz. Wyznacz wartości tego przebiegu dla co najmniej jeden sekundy, przyjmując częstotliwość próbkowania równą 2000 Hz.



Rysunek 18: Przykładowe zdjęcie na którym można zaobserwować zjawisko aliasingu.

```
Fs = 2000;
t1=0:1/Fs:1;
y1=sin(2*pi*500*t1); % sinusoidal signal with 500 Hz frequency
```

Listing 26: Tworzenie sygnału sinusoidalnego o częstotliwości 500 Hz i częstotliwości próbkowania 2000 Hz

Wyznacz widmo tego sygnału w zakresie częstotliwości $f \in [0; \frac{f_s}{2}]$ tak jak robiliśmy to na drugich laboratoriach 18. Powinieneś zaobserwować pojedynczy prążek dla częstotliwości 500 Hz.

Teraz powtórz ten sam proces zmieniając częstotliwość próbkowania odpowiednio $f_s \in \{1100; 900; 700; 600\}$. Zaobserwuj moment przekraczania granicy Nyquista i zawinięcia się widma, wyższe częstotliwości przechodzącą na stronę pasma poniżej połowy częstotliwości próbkowania.

4.4.3 Rekonstrukcja sygnału ze zbyt niskim okresem pomiędzy próbkami

Podobnie jak w przykładzie 26 spróbuj interpolować sygnał utworzony w poprzednim zadaniu z częstotliwością próbkowania równą 600 Hz. Utwórz nowy wektor czasu tak, aby interpolowany sygnał miał częstotliwość próbkowania ponownie 2000 Hz np. w następujący sposób: `t = 0:1/2000:1`. Wykorzystując funkcję `interp1` wyznacz nowy wektor wartości dla nowego wektora czasu. Narysuj otrzymany sygnał. Zwróć uwagę, że tak jak pokazywała analiza częstotliwościowa nie udało się zrekonstruować częstotliwości 500 Hz. W procesie rekonstrukcji została ona „zawinięta" i stała się składową o częstotliwości 100 Hz.

Bibliografia

[1] Zieliński Tomasz P. Cyfrowe przetwarzanie sygnałów. Wydawnictwa Komunikacji i Łączności WKŁ, Warszawa, 2, 2016.

[2] FIR Filter Design. URL: https://www.mathworks.com/help/signal/ug/fir-filter-design.html.

[3] IIR Filter Design. URL: https://www.mathworks.com/help/signal/ug/iir-filter-design.html.

[4] Fourier Analysis. URL: https://www.youtube.com/playlist?list=PLMrJAkhIeNNT_Xh3Oy0Y4LTj0Oxo8GqsC.