SKRYPT DO LABORATORIUM

Wymiana i składowanie danych multimodalnych

ĆWICZENIE 3: Dyskretna transformata kosinusowa – zasada działania i podstawowe właściwości

autor:
dr inż. Adam Bujnowski

Gdańsk, 2012

1. Wymagania wstępne

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt *„Przygotowanie i realizacja kierunku inżynieria biomedyczna – studia międzywydziałowe”*
współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

1.1.Ustawienia

Wymagania wstępne dotyczące uczestników osób realizujących ćwiczenie: elementarna znajomość podstawowych przekształceń trygonometrycznych

Cele ćwiczenia:

Wykaz przyrządów, materiałów i aparatury niezbędnej do przeprowadzenia ćwiczenia:

Komputer klasy PC z systemem operacyjnym co najmniej MS Windows XP, z zainstalowanym środowiskiem MatLab. Dostęp do internetu.

Spodziewane efekty kształcenia - umiejętności i kompetencje:

Umiejętność implementacji jedno- i dwuwymiarowej transformaty kosinusowej. Znajomość podstawowych własności transformaty. Umiejętność praktycznego zastosowania.

Metody dydaktyczne:

Indywidualna praca studenta z użyciem komputera PC i programu Matlab w oparciu o niniejszą instrukcję

Zasady oceniania/warunek zaliczenia ćwiczenia

Krótki opis

Wykaz literatury podstawowej do ćwiczenia:

1.	Skrypt do wykładu
2.	

2. Przebieg ćwiczenia

L.p.	Zadanie
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	

UWAGI!

3. Wprowadzenie do ćwiczenia

Dyskretna transformata kosinusowa jest przekształceniem dokonującym przekształcenia podobnego do transformaty Fouriera.

DCT przekształca skończony ciąg N liczb rzeczywistych w ciąg liczb rzeczywistych $G(0),\ldots,G(N-1)$ zgodnie z zależnościami:

$$G(0)=\frac{1}{\sqrt{N}}\sum_{m=0}^{N-1} g(m)$$

$$G(k)=\sqrt{\frac{2}{\sqrt{n}}}\sum_{m=0}^{N-1} g(m)\cos\frac{\pi(2\mathrm{m}+1)}{2\mathrm{N}}$$ dla k=1,2,...,N-1

Otrzymane G(k) są nazywane współczynnikami dyskretnej transformaty kosinusowej lub transformatą.

Odwrotna transformata kosinusowa (IDCT) definiowana jest jako:

$$g(m)=\frac{1}{\sqrt{N}}G(0)+\sqrt{\frac{2}{N}}G(k)\cos\frac{\pi k(2\mathrm{m}+1)}{2\mathrm{N}}$$ dla m=0,1,...,N-1

4. Realizacja ćwiczenia

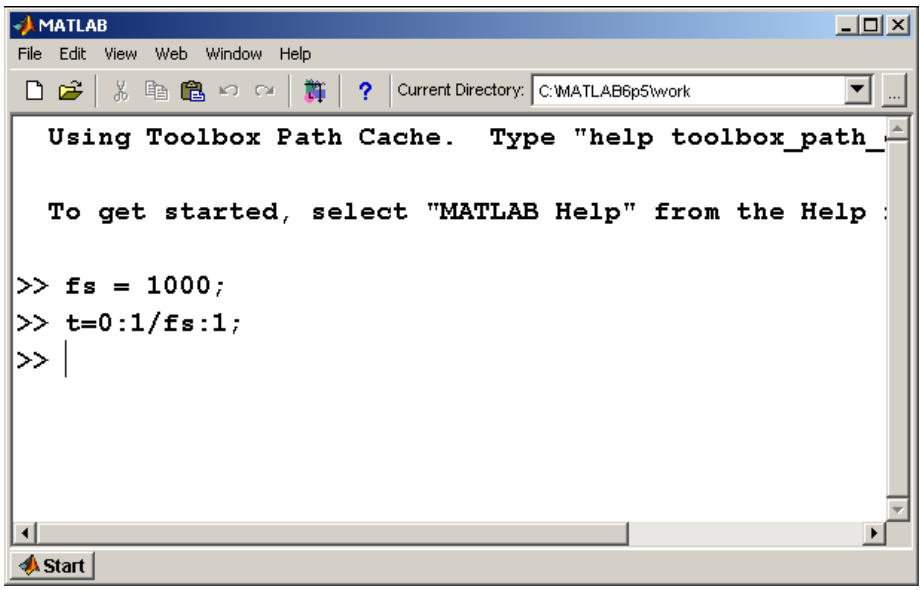
Właściwości dyskretnej transformaty kosinusowej przetestowane zostaną przy pomocy programu Matlab.

Na początku zbadajmy podstawowe własności przekształceń, które są podstawą dla transformaty Fouriera jak i DCT.

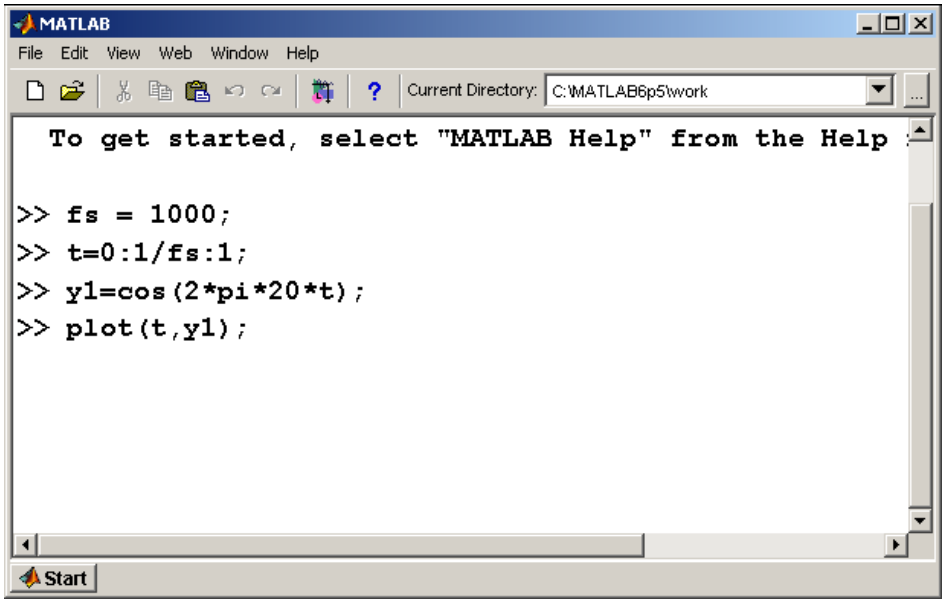
Transformata DCT jest pochodną przekształcenia Fouriera, toteż dla lepszego jej zrozumienia przedstawimy podstawowe własności przekształcenia Fouriera, skupiając się na ich dyskretnej wersji.

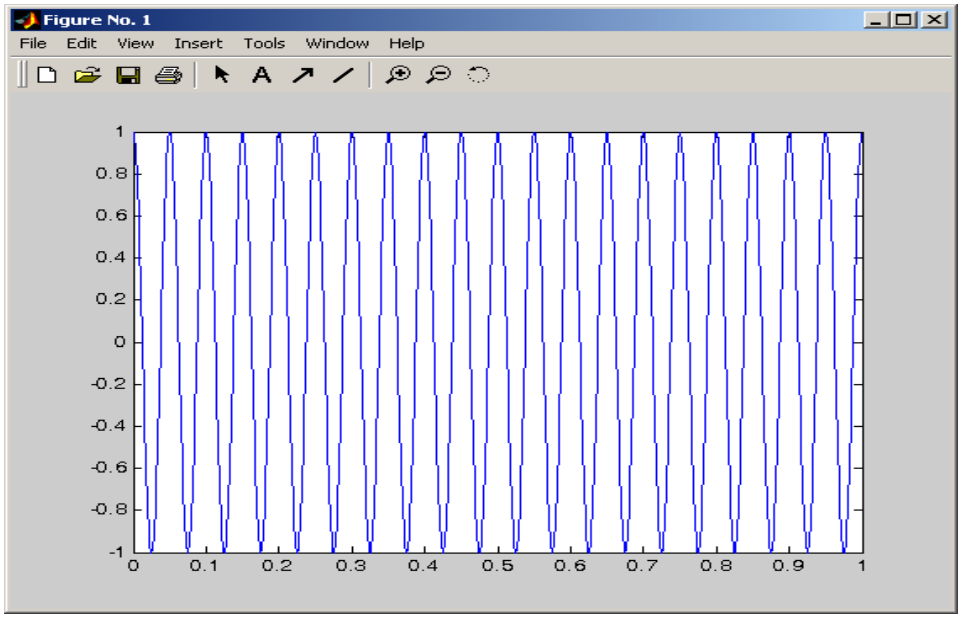
W twierdzeniu Fouriera mówi się, że dowolny sygnał okresowy może zostać przedstawiony jako suma nieskończonej ilości sygnałów sinusoidalnych o różnych częstotliwościach, amplitudach i fazach początkowych. W jaki sposób sprawdza się, czy w danym sygnale wejściowym występuje składowa o danej częstotliwości ?

Wykonajmy prosty test posługując się programem Matlab. Na początku wygenerujmy oś czasu. Przyjmijmy częstotliwość próbkowania fs=1000Hz, wygenerujmy wektor czasu odpowiadający jednej sekundzie. Jest to podstawa dalszych naszych dociekań:

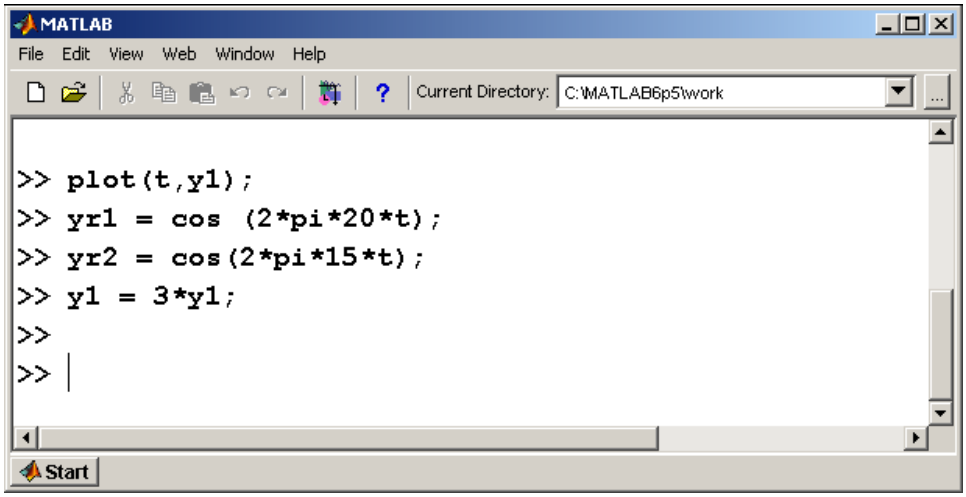


Zauważ, że mając oś czasu jesteś w stanie wygenerować dowolny sygnał. Przygotujmy satem sinusoidę o częstotliwości 20Hz:

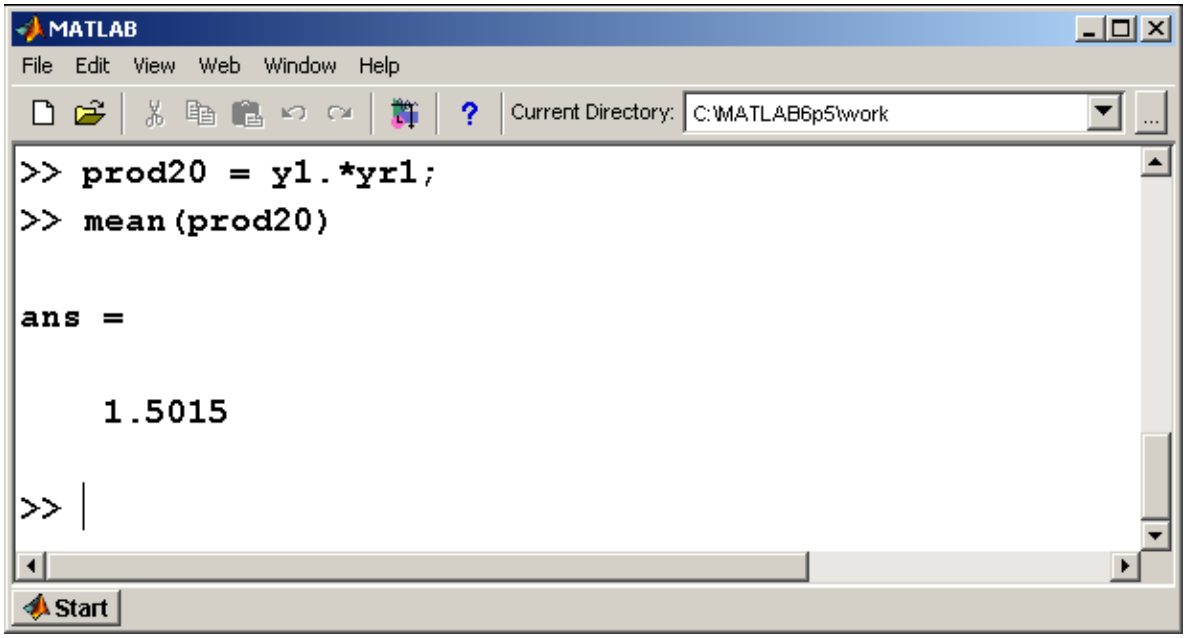




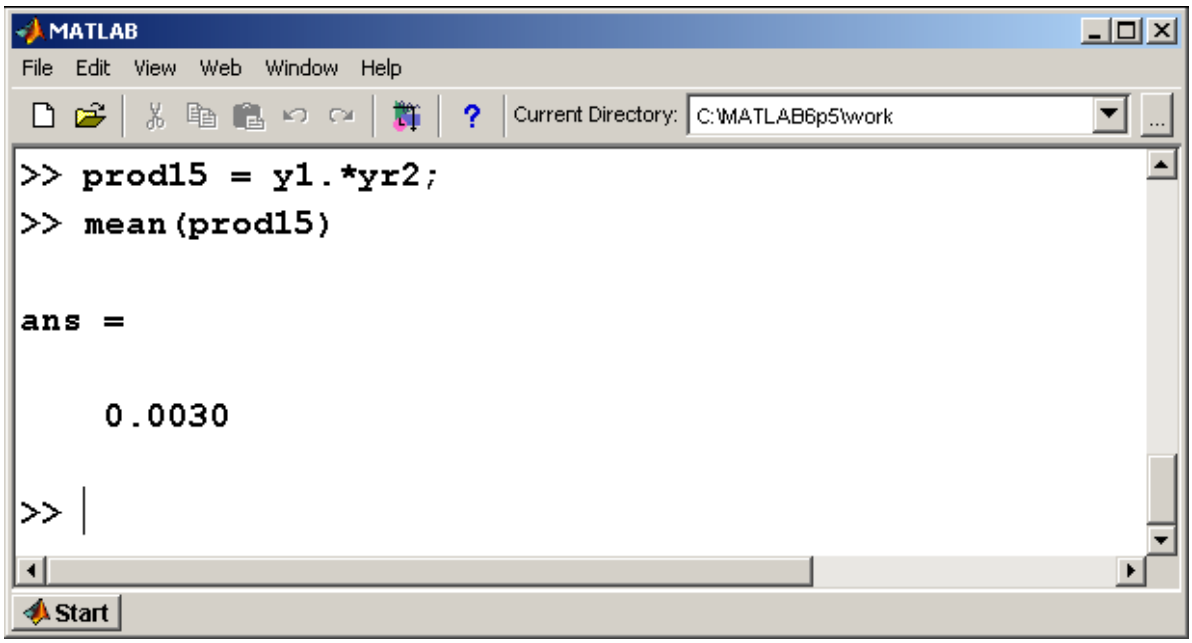
Mając tak przygotowane dane wygenerujmy jeszcze dwa przebiegi sinusoidalne – jeden o częstotliwości 20 Hz (jak poprzedni) , drugi o częstotliwości innej , np 15 Hz, oraz zmieńmy amplitudę dla przebiegu y1:



Sygnały yr1 i yr2 potraktujemy jako wzorcowe, zaś y1 jako sygnał „nieznany”. Sprawdźmy zatem czy w naszym „nieznanym sygnale występuje składowa o częstotliwości 20Hz i 15 Hz odpowiednio. Aby tego dokonać wystarczy wymnożyć przez siebie sygnał nieznany i referencyjny oraz uśrednić wynik mnożenia:



Wykonajmy podobne działanie testując nasz sygnał na obecność składowej 15Hz:



Zauważ, że w pierwszym przypadku wynik operacji dał wynik 1.5015, w drugim przypadku 0,003. Oznacza to , że nasz „detektor” w nieznanym sygnale wykrył składową 20Hz a 15Hz praktycznie nie. Co więcej, zauważ, że składowa 20Hz zwróciła wynik o połowę mniejszy od faktycznej amplitudy sygnału !

W rzeczywistości, posługując się prostymi przekształceniami trygonometrycznymi da się zapisać:

$$\cos(\alpha) * \cos(\beta) = \frac{1}{2}\cos(\alpha - \beta) + \frac{1}{2}\cos(\alpha + \beta)$$

Zatem gdy $\alpha = \beta$, powyższe równanie przyjmuje postać:

$$\cos(\alpha) * \cos(\alpha) = \frac{1}{2}\cos(\alpha - \alpha) + \frac{1}{2}\cos(\alpha + \alpha) = \frac{1}{2} + \cos(2\alpha)$$

Kolejnym etapem mnożenia jest uśrednianie. Operacja ta może być zapisana jako:

$$\bar{y} = \int_{-\infty}^{\infty} y(t)dt$$

Jeśli $y(t) = \cos(\omega t)$ i $\omega \neq 0$ to

$$\bar{y} = \int_{-\infty}^{\infty} \cos(\omega t)dt = 0$$

Co więcej, nasz badany sygnał można przetestować na obecność każdej częstotliwości – wystarczy wygenerować sygnał testujący dla każdej dostępnej częstotliwości. Matematyczny zapis tej operacji wygląda następująco:

$$Y(\omega) = \int_{\omega\,\min}^{\omega\,\max} y(t) * \cos(\omega t)dt$$

Jeśli w badanym sygnale nie ma szukanej przez nas częstotliwości – wówczas wynik takiej operacji będzie równy 0, w przeciwnym przypadku otrzymamy połowę amplitudy składowej sygnału

występującego w badanym sygnale. Czy zawsze ?. Otóż nie zawsze. Warunkiem, który należy spełnić jest zgodność faz sygnału referencyjnego i badanej składowej. Zauważ że :

$$\int (\sin(\omega t)\cos(\omega t))\,dt = 0$$

Łatwo jest to sprawdzić w Matlabie.

ZADANIE: Sprawdź takie działanie i zademonstruj wynik prowadzącemu.

Oczywiście – funkcje sinus i kosinus są wzajemnie ortogonalne – zatem w celu wyznaczenia zarówno amplitudy jak i fazy badanego sygnału niezbędne jest wymnożenie sygnału testowanego przez zestaw par kosinusów i sinusów dla całego rozważanego zakresu częstotliwości. Korzystając z liczb zespolonych możliwe jest zapisanie takiej pary jako:

$$\cos(\omega t) + j\sin(\omega t)$$

zatem w wyniku transformaty Fouriera otrzymujemy wielkość zespoloną:

$$\Re(Y(\omega)) = \int_{-\infty}^{\infty} y(t) * \cos(\omega t)\,dt$$

$$\Im(Y(\omega)) = -\int_{-\infty}^{\infty} y(t) * \sin(\omega t)\,dt$$

Co da się zapisać bardziej jednolicie jako:

$$\overline{Y}(\omega) = \int_{-\infty}^{\infty} y(t) e^{-j\omega t}\,dt$$ (pamiętając wzór Eulera $e^{-jx} = \cos(x) - j\sin(x)$

Mając taką zespoloną transformatę możliwe jest wyznaczenie amplitudy i fazy każdej składowej niezależnie:

$$Y(\omega) = \sqrt{\Re(Y(\omega))^2 + \Im(Y(\omega))^2}$$

$$\phi = atan\frac{\Im(Y(\omega))}{\Re(Y(\omega))}$$

Możliwe jest numeryczne wyznaczenie transformaty Fouriera. Wówczas operujemy na dyskretnych i skończonych ciągach próbek, zaś operację całkowania zastępujemy sumą dzieloną przez rozmiar ciągu. Stoi za tym szereg twierdzeń, których nie będziemy tutaj przytaczać. Warto jednak zapamiętać sposób wyznaczania dyskretnej transformaty Fouriera(DFT):

$$Y_k = \sum_{n=0}^{N-1} a_n e^{-j2\frac{PIkn}{N}}$$

Jak widać dyskretna transformata Fouriera wykonana dla ciągu N elementowego zwróci N elementowy ciąg liczb zespolonych. Tak liczona transformata stosowana jest jednak stosunkowo rzadko, ze względu na swoją złożoność

Numerycznie, wykorzystując pewne własności DFT możliwe jest znaczne przyspieszenie działania algorytmu, co prowadzi do FFT, która jednak wyznaczana jest dobrze tylko dla ciągów o długościach będących naturalną potęgą liczby 2.

Dla zainteresowanych:

Spróbuj wyznaczyć transformatę Fouriera ciągu y1 w programie Matlab:

```
fy = fft(y1);
plot(fftshift(abs(fy)));
```

Zdefiniujmy inny ciąg wejściowy:

```
yy1 = sign(y1);
```

wyznaczmy dla tego ciągu transformatę FFT:

```
fyy = fft(yy1);
plot(fftshift(abs(fyy)));
```

Łatwo można sprawdzić, że w wyniku odwrotnej transformaty Fouriera możliwe jest „odzyskanie” sygnału wyjściowego – w tym przypadku prostokątnego.

Dla sygnału prostokątnego możliwe jest zaobserwowanie jak ilość kolejnych jego składowych wpływa na dokładność odwzorowania funkcji oryginalnej. W tym celu wystarczy wykonać następujący skrypt:

%plik test_prostok.m

```
clear;
close all;
fs=1000;  % czestotliwosc probkowania
t=0:1/fs:1; % os czasu
y = sign(cos(2*pi*10*t));
plot(t,y);
title('Sygnał wejściowy');

fy = fft(y); % transformata
figure;
plot(-fs/2:fs/2, fftshift(abs(fy)));
title('Moduł transformaty');
```

```
figure;
plot((abs(fy(1:500))));
title('dodatnia część');
%pobierzmy amplitudy kolejnych prążkow
tab_pr  =  [real(fy(11))  real(fy(31))  real(fy(51))  real(fy(71))
real(fy(91)) real(fy(111)) real(fy(131)) real(fy(151))]
freqs  = [10 30 50 70 90 110 130 150];
figure;hold on;
iy = zeros(size(t));
for k=1:length(tab_pr),
    iy = iy+ tab_pr(k) * cos(2*pi*t*freqs(k));
    plot(t,iy);
    pause;
end;
```

Do tej pory rozważaliśmy zasadę działania transformaty Fouriera i jej dyskretnych wersji. Dyskretna transformata kosinusowa (DCT) jest transformatą o podobnych własnościach do transformaty Fouriera. Stosowana jest chętnie w przetwarzaniu multimediów – jest wykorzystywana w JPEG, MPEG itp.

To co jest charakterystyczne – wynikiem przekształcenia Fouriera jest ciąg próbek zespolonych. Stwarza to szereg kłopotów oraz ogranicza wydajność algorytmów przetwarzania sygnałów.

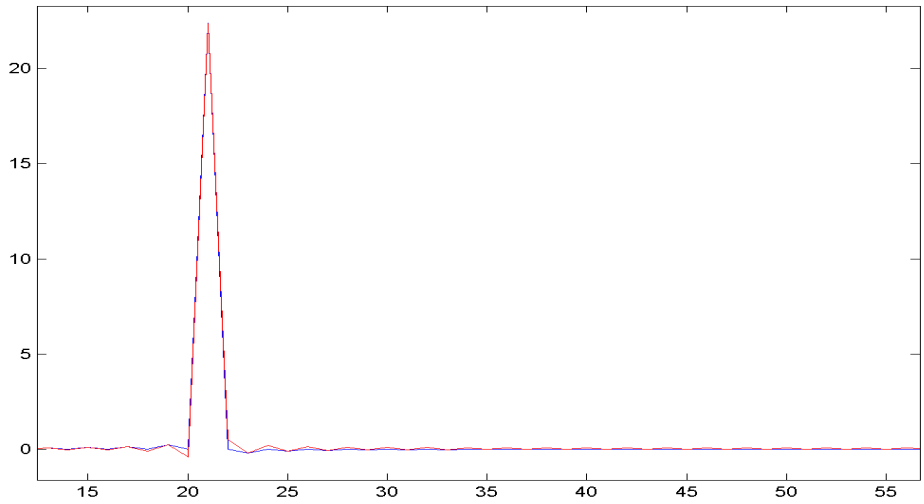
W DCT wynikiem jest ciąg liczb rzeczywistych. Jednowymiarową DCT można zrealizować następująco:

```
function o = dct1(in)
N = length(in);
for k=1:N,
  cc = cos(((k-1)*pi*[0:N-1])  ./   (N) );
  o(k) =  sqrt(2/N)* sum(in .* cc );
end;
o(1) = o(1)/2;
```

Porównajmy działanie tak stworzonej funkcji z funkcją wbudowaną w Matlaba.

```
fs=1000;
t=0:1/fs:1;
y=cos(2*pi*10*t);
```

```
plot(dct(y));
hold on ;
plot(dct1(y),'r');
```



Analizując oba wykresy – widać nieznaczne różnice. W rzeczywistości mogą one wynikać z różnej wersji DCT implementowanej w naszym skrypcie a tej z Matlaba (jest 8 rodzajów DCT). Tym niemniej, zauważmy , że w obu przypadkach otrzymaliśmy widmo asymetryczne – będące ciągiem liczb rzeczywistych. Zauważmy, że tym razem widmo to jest wyznaczone dla dwa razy większej liczby częstotliwości niż ma to miejsce w DFT. Tutaj – częstotliwości 10Hz odpowiada 21 prążek, w DFT – 11. Za składową stałą odpowiada zerowy element w obu transformatach.

Mając transformatę DFT, możliwe jest zaimplementowanie transformaty odwrotnej. Zauważ, że działa ona tak samo jak DCT – co dodatkowo upraszcza algorytmy przetwarzające sygnały.

Zbadajmy podstawowe własności DCT

1. Filtracja:

Wykonaj następujące polecenia (testfilt.m):

```
close all;
clear;
fs=1000;
t=0:1/fs:1;
y = 10*cos(2*pi*10*t) + 2*cos(2*pi*70*t) + cos(2*pi*200*t);
plot(t,y);
title('Sygnał');
dcty = dct(y);
figure;
plot(dcty);
title('Transformata');
```

```
%filtracja :
dcty(100:1001) = 0; % tu filtracja
pause;
figure;
plot(dcty);
title('Transformata po filtracji');
figure;
fy = idct(dcty);
plot(t,fy);
title('Odfiltrowany sygnał');
```

2. Synteza syhnałów:

```
%synt_test
 close all;
 clear;
 %synteza sygnałów
 fy = zeros(1,1001);
 fy(31) = 10; % wpisz niezerową wartość do tablicy
 fy(100)=20; % w celu wygenerowania sygnału w dziedzinie czasu
 y = idct(fy);
 plot(y);
```

Potestuj zachowanie wpisując różne wartości do tablicy fy – w różnych miejscach.

3. Interpolacja – poprzez uzupełnienie transformaty zerami możliwa jest interpolacja sygnału.

```
% test interpolacji
n = [[0:5] [5 5 5] [5:-1:0]];
plot(n,'o');
title('Sygnał wejściowy');
figure;
fn = dct(n);
plot(fn); title('Transformata');
fn = [fn zeros(1,100)];
plot(fn); title('Transformata uzupełniona zerami');
yn = idct(fn,'o');
```

```
figure;
plot(yn);title('Sygnał po interpolacji');
```

Dwuwymiarowa DCT

W ogólności zarówno transformata Fouriera jak i DCT może być wielowymiarowa. Poznajmy dwuwymiarową DCT, która zdefiniowana jest następująco:

$$DCT2(k_1,k_2)=\sum_{n=0}^{N-1}\sum_{m=0}^{M-1}x(n,m)\cos\left(\frac{\pi}{N}(n+\frac{k_1}{2})\right)\cos\left(\frac{\pi}{N}(m+\frac{k_2}{2})\right)$$

Podobnie jak jednowymiarowa DCT dwuwymiarowa DCT jest również zaimplementowana w Matlabie. Jest to funkcja o nazwie dct2.

Zadanie . Spróbuj samodzielnie zaimplementować funkcję wyznaczającą dwuwymiarową DCT. Porównaj jej zachowanie z funkcją Matlaba.

Zbadajmy jej podstawowe właściwości. W tym celu należy posłużyć się macierzami, jako obrazami wejściowymi.

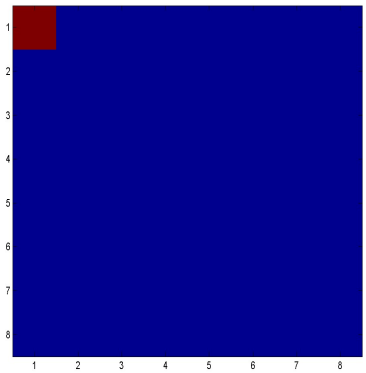
Wygeneruj obraz wejściowy 8x8 px:

y = ones(8,8);

Zbadaj jego transformatę:

imagesc(dct2(y));

Transformata jednorodnej płaszczyzny ma tylko jedną niezerową wartość:



Wygenerujmy obraz, który zmienia się w jednym z wymiarów. W tym celu stwórzmy funkcję :

function y = genpict(n,f1,f2)

y = zeros(n,n)

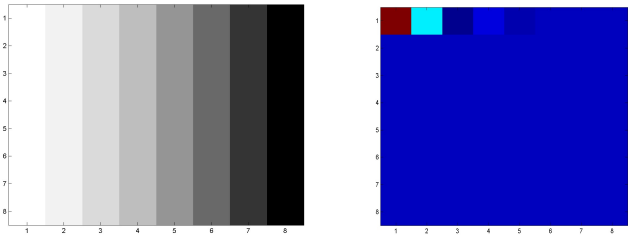
for k=1:n,

for m=1:n,

```
      y(k,m) = cos(2*pi*k*f1/n)*cos(2*pi*f2*m/n);
   end;
end;
```

Wygenerujmy z jej użyciem obraz:

```
o = genpict(8,0,0.2);
imagesc(dct2(o));
```



Sprawdź transformaty dla innych obrazów:

o= genpict(8,0,0.4)

o=genpict(8,0.1,0)

o=genpict(8,0.4,0)

o=genpict(8,0.3,0.3) itd.

Spróbuj zmienić parametry f1 i f2 oraz n dla funkcji genpict.

Zauważ, że tak wygenerowane transformaty mają niewiele elementów niezerowych. Elementy te skupione są wokół elementu (1,1). Fakt ten powszechnie jest wykorzystywany w kompresji obrazów. Obraz jest dzielony na bloki, z których każdy jest poddawany DCT. Następnie każdy blok jest zapisywany i kompresowany z użyciem algorytmu kompresji bezstratnej (np. algorytm Huffmana). Stratność kompresji polega na ograniczeniu rozmiaru zapisywanego bloku.

W tej części ćwiczenia postaramy się to zaprezentować.

Pozłużymy się obrazem testowym.

Wgrajmy ten obraz :

```
load lena512
imagesc(lena512); colormap gray;
```

Do badania właściwości posłużymy się funkcją:

```
function [ow,o1] = decode(oin, blocksize,std)
[a,b] = size(oin);
ow = zeros(a,b);
o1 = ow;
la = a/blocksize;
lb = b/blocksize;
for k=1:la,
  for b = 1:lb,
    pocx = (k-1)*blocksize + 1;
    konx = pocx + blocksize-1;
    pocy = (b-1)*blocksize + 1;
    kony = pocy + blocksize-1;
    od = std;
    % transformata bloku
    cos1 = dct2(oin(pocx:konx,pocy:kony));
     %degradacja
    cos1(od:blocksize,:) = 0;
    cos1(:,od:blocksize) = 0;
    ow(pocx:konx,pocy:kony) = cos1;
    cos2 = idct2(cos1,blocksize,blocksize);
    o1(pocx:konx,pocy:kony) = cos2;
    subplot(3,1,3); image(cos2); colormap gray;
 end;
end;
```

Wywołajmy tą funkcję z paramerami

```
[a,b] =decode(lena512,8,8),
```

Po wykonaniu funkcji obraz a będzie zawierał transformatę bloków, b – obraz po zdekodowaniu.

Przetestuj działanie tej funkcji dla różnych parametrów blocksize (8,16,32) oraz std – stopnia degradacji, który będzie się zmieniał w zakresie od blocksize – brak degradacji do 1 – maksymalna degradacja obrazu.

Możesz sprawdzić stopień degradacji np. mierząc podobieństwo obrazów przy pomocy MSE.

$$MSE(o1, o2) = \frac{1}{MN} \sum_{n=1}^{N} \sum_{m=1}^{M} |u(m,n) - v(m,n)|^2$$

implementacja

```
function o=mse(im1,im2)
% zakładam, ze wymiary obrazów są identyczne.
o = (1./(size(im1,1)*size(im1,2)))*sum(sum((abs(im1-im2).^2)));
```

Zadanie:

Przygotuj wykres podobieństwa między obrazami w funkcji parametru blocksize oraz std (dwa wykresy, lub jeden 3d)