Wstęp do zastosowań analizy falkowej

Materiały na laboratorium

6.05.2007

1. Odczytywanie, zapisywanie i wyświetlanie obrazków w Matlabie. Obrazki w postaci cyfrowej występują w kilku rodzajach. Są tak zwane formaty indeksowane, gdzie każdemu pikselowi (elementowi obrazu) przypisywana jest liczba (najczęściej 1 bajt) będąca numerem koloru na liście. Lista kolorów, tak zwana paleta, najczęściej składa się z 256 kolorów, które reprezentowane są przez 3 bajty, odpowiadające zawartości składowej czerwonej, zielonej i niebieskiej. Przykładami formatów indeksowanych są formaty `bmp` i `gif`. Format `gif` zawiera obraz oraz paletę w postaci skompresowanej (bezstratnie). W tym formacie w jednym pliku obrazów może być kilka, i każdy może mieć własną paletę. Jeżeli obrazów jest kilka, to całość może być animowana. Jeden z kolorów może być zadeklarowany jako przezroczysty. Plik w formacie `bmp` zawiera obraz i paletę, najczęściej w postaci nieskompresowanej. Sam format dopuszcza prostą kompresję, ale w praktyce nie spotyka się skompresowanych plików w formacie `bmp`. Drugim rodzajem obrazów cyfrowych są obrazy w odcieniach szarości. Obrazy tego typu najczęściej zapisywane są w jednym z formatów indeksowanych, z użyciem standardowej palety. Standardowa paleta zawiera równomiernie rozłożone odcienie szarości (3 bajty kolorów składowych równe sobie), od koloru (0,0,0), czyli czerni (indeks 0) do (255,255,255), czyli bieli (indeks 255). Trzecim rodzajem obrazów cyfrowych są tak zwane obrazy *true color*, w których każdemu pikselowi bezpośredni przypisane są 3 bajty określające zawartość 3 kolorów składowych. Obrazy tego typu zapisywane są w formatach `tiff` (skompresowane bezstratnie) i `jpeg` (skompresowane stratnie).

W tym laboratorium będziemy zajmowali się obrazami drugiego typu, to znaczy obrazami w odcieniach szarości. Kolorowe obrazy indeksowane nie mogą być sensownie przetwarzane przy pomocy stosowanych przez nas metod, ponieważ numery kolorów w palecie nie mają związku z własnościami „kolorystycznymi". Innymi słowy, to, że dwa kolory mają bliskie sobie indeksy nie oznacza, że są do siebie podobne. Oczywiście kolorowy obrazek w formacie indeksowanym można najpierw przekształcić do formatu *true color* lub odcieni szarości, i wtedy obrabiać. Obrazki *true color* można przetwarzać traktując każdy kolor składowy (tak zwany kanał) jako oddzielny obrazek. Lepiej jest najpierw przekształcić kanały z tak zwanej przestrzeni RGB (kanały to składowe czerwone, zielone i niebieskie) do przestrzeni $\mathrm{YC}_b\mathrm{C}_r$ (kanały to mieszanki kolorów składowych, przy czym Y to kanał luminancji, czyli najważniejsza składowa zawierająca odcienie szarości, C_b i C_r to kanały chrominancji, odpowiednio niebieski i czerwony). Przekształcenia pomiędzy przestrzeniami RGB i $\mathrm{C}_b\mathrm{C}_r$ implementuje się przy pomocy konkretnej, odwracalnej macierzy 3×3. Być może starczy nam czasu na zajęcie się obrazkami kolorowymi, wtedy poznamy więcej szczegółów na temat zarządzania barwami.

Do wczytania obrazka służy funkcja `imread`. Jeżeli chcemy wczytać obraz Lena.bmp, który jest w bieżącym katalogu, wydajemy instrukcję

```
A=imread('Lena.bmp','bmp')
```

Powstaje w ten sposób macierz pikseli o wartościach od 0 do 255. Będziemy korzystali z formatów `gif` i `bmp`. Są to tak zwane formaty indeksowane, to znaczy wartość piksela jest numerem koloru w palecie. W obrazach z których będziemy korzystali paleta jest zawsze

taka sama, standardowa. Składa się z 256 odcieni szarości, zmieniających się jednostajnie od czerni (pierwszy kolor palety, odpowiada mu indeks 0) do bieli (256 kolor, o indeksie 255). Jeżeli użyjemy innej palety to nasz obraz będzie miał zmienione kolory. Wczytując obrazek możemy wczytać też jego paletę, zakodowaną wraz z obrazkiem w pliku (dotyczy to, oczywiście formatów indeksowanych, takich jak `gif` lub `bmp`), wywołując funkcję `imread` w następujący sposób:

```
[A,MAP]=imread('goldhill.gif','gif')
```

Paleta obrazka składa się z trójek bajtów, reprezentujących intensywność kolorów czerwonego, zielonego i niebieskiego w skali od 0 do 255. Przy wczytywaniu do tablicy `MAP` (`MAP` jest tablicą 256 na 3, o wartościach typu `double`) te wartości są przeskalowane do zakresu [0,1]. Taki jest format używanej przez Matlab palety (tak zwanej colormapy): dowolna ilość wierszy, 3 kolumny, i wartości typu double z przedziału [0,1].

Otrzymana macierz pikseli `A` zawiera dane typu `uint8`. Niektórych obliczeń w Matlabie nie można wykonywać na liczbach typu `uint8` i najpierw trzeba je przekształcić do typu `double`.

```
B=double(A)
```

Podstawowe funkcje pakietu RWT, z którego będziemy korzystać nie mogą operować na danych typu `uint8`, więc nasze dane typu `uint8` zawsze będziemy przed obliczeniami przekształcać na `double`. To nic nie kosztuje. Zmieniany jest tylko typ danych, wartości pozostają bez zmian.

Do zapisu macierzy do pliku graficznego służy funkcja `imwrite`:

```
imwrite(A,'lena.bmp','bmp')
```

Macierz `A` powinna być typu `uint8`. Można ją do tego typu przekształcić z dowolnego innego (na przykład `double`) używając funkcji `uint8`

```
A=uint8(B)
```

Funkcja `uint8` zaokrągla liczby `double` do najbliższej całkowitej, p czym obcina zakres do przedziału [0,255]. Jeżeli funkcję `imwrite` zastosujemy do tablicy o wartościach typu `double` to wartości zostaną najpierw pomnożone przez 255, a nastepnie przekształcone przy pomocy funkcji `uint8`. Wartości poniżej 0 przejdą na 0 a powyżej 1 na 255. Cały używany przez macierz `A` zakres wartości powinien więc się zawierać w przedziale [0,1]. Dlatego przed zapisem do pliku warto ręcznie obrobić wartości macierzy `A` tak, aby zawierały się w przedziale [0,1]. Na przykład, niech

$$M = \max_{i,j} A(i,j), \qquad N = \min_{i,j} A(i,j),$$

i

$$A'(i,j) = \frac{A(i,j) - N}{M - N}.$$

Po takim przekształceniu wartości macierzy `A'` są w przedziale [0,1], a ponieważ przekształcenie wartości jest liniowe więc obraz „graficznie" nie uległ zmianie. Wraz z macierzą do pliku zapisywana jest także standardowa paleta, składająca się z 256 równomiernie rozłożonych odcieni szarości (dotyczy to formatów indeksowanych, takich jak `bmp`). Jeżeli chcemy zapisać inną paletę możemy użyć instrukcji

```
imwrite(A,MAP,'lena.bmp','bmp')
```

gdzie `MAP` jest dowolną colormapą Matlaba. Zostanie ona przeskalowana do formatu palety odpowiedniego pliku graficznego, ucięta do 256 wierszy jeżeli jest dłuższa, i uzupełniona wierszami zer, jeżeli jest krótsza niż 256 kolorów. Matlab nie zapisuje plików w formacie `gif`.

Macierz `A` można wyświetlić jako obraz nie zapisując jej do pliku graficznego. Służy do tego instrukcja `image`

```
image(A)
```

Wartości wyświetlanej macierzy `A` są traktowane jako indeksy do bieżącej colormapy Matlaba. Jeżeli są typu `uint8` to są bezpośrednio traktowane jako indeksy kolorów. Jeżeli są typu `double` to najpierw odejmowana jest od nich 1, następnie są zaokrąglane do najbliższej liczby całkowitej, a następnie obcinane na poziomie 0 i 255. Tak powstałe wartości są traktowane jako indeksy kolorów. Wygodną instrukcją jest też `imagesc`. Wartości tablicy `A` są najpierw przeskalowane liniowo, tak, żeby najmniejsza wartość (może być ujemna) przyjęła wartość 0 a największa 255. Tak otrzymane wartości są zaokrąglane w dół do liczb całkowitych, i traktowane jako indeksy kolorów bieżącej colormapy. Jeżeli chcemy przeskalować inaczej, możemy użyć składni

```
imagesc(A,[a,b])
```

wtedy zakres `[a,b]` zostanie rozciągnięty liniowo do zakresu `[0,255]`.

Do podglądu naszych obrazów będzie nam potrzebna standardowa colormapa, którą musimy sami wygenerować:

```
szara=zeros(256,3);
for i=1:256
  szara(i,1)=(i-1)/255;
  szara(i,2)=szara(i,1);
  szara(i,3)=szara(i,1);
end;
```

Następnie ustalamy bieżącą colormapę instrukcją

```
colormap(szara)
```

Colormapa pozostaje ustalona dla danego okna obrazka aż do jego zamknięcia. Przy następnym otwarciu colormapę trzeba ustawić ponownie. Można najpierw załadować obrazek, a następnie wydać instrukcję `colormap`. Dopóki nie zamkniemy okna obrazka ta sama colormapa będzie stosowana do wszystkich kolejno ładowanych obrazków. Naszą colormapę *szara* możemy zachować w pliku instrukcją `save`, i na następnych zajęciach załadować z pliku instrukcją `load`. Matlab ma pewną ilości predefiniowanych colormap, jedną z nich jest colormapa domyślna. Zawiera ona tylko 64 kolory. W przypadku gdy ustalona colormapa ma mniej kolorów niż zakres wartości wyświetlanej macierzy `A`, to brakujące kolory są zastępowane czarnym. Obrazki w domyślnej colormapie są bardzo niewyraźne. Inne standardowe colormapy to `gray`, `hot`, `cool`, `copper` czy `pink`. Standardowe colormapy są z reguły krótsze niż 256 kolorów, ale każdej z powyższych nazw można użyć do wygenerowania colormapy dowolnej długości. Na przykład zamiast podanej powyżej procedury generowania colormapy „szara” można w Matlabie użyć instrukcji

```
szara=gray(256);
```

albo do ustawienia bieżącej colormapy instrukcji

Fig. 2.1. Lekkie rozjaśnienie

```
colormap(gray(256));
```

2. Podstawowe operacje na obrazkach. Wypróbujemy niektóre typowe przekształcenia na obrazkach. Niektóre z operacji występujących w przykładowych skryptach można w Matlabie wykonać prościej, wykorzystując wbudowane funkcje. Na przykład w Matlabie są specjalne funkcje zwracające największą i najmniejszą wartość współczynników tablicy. W prezentowanych skryptach raczej wszystko wykonywane jest „na piechotę".
a) Rozjaśnij: Zwiększamy wartość pikseli. Wartości powyżej 255 będą ucięte

Fig. 2.2. Lekkie przyciemnienie

```
A=imread('Lena.bmp','bmp');
B=double(A);
B=1.2*B;
A=uint8(B);
imwrite(A,'Lena2a.bmp','bmp');
```

b) Przyciemnij: Zmniejszamy wartość pikseli:

```
A=imread('Lena.bmp','bmp');
B=double(A);
B=B/1.2;
A=uint8(B);
imwrite(A,'Lena2b.bmp','bmp');
```

Fig. 2.3. Zwiększony kontrast

c) Zwiększ kontrast: Znajdziemy największą i najmniejszą wartość pikseli. Następnie przekształcimy, liniowo, obraz tak, żeby najmniejsza wartość wynosiła 0 a największa 255. W ten sposób maksymalizujemy kontrast. Jeżeli kontrast był już maksymalny to operacja nie daje żadnego efektu.

```
A=imread('Lena.bmp','bmp');
B=double(A);
min=B(1,1);
max=B(1,1);
for i=1:512
  for j=1:512
    if B(i,j)>max
      max=b(i,j);
    end;
    if B(i,j)<min
```

```
      min=B(i,j);
    end;
  end;
end;
c=255/(max-min);
for i=1:512
  for j=1:512
    B(i,j)=(B(i,j)-min)*c;
  end;
end;
A=uint8(B);
imwrite(A,'Lena2c.bmp','bmp');
```

Fig. 2.4. Zmniejszony kontrast

Częstą operacją jest korekta kontrastu tylko w pewnym zakresie jasności. Na przykład przekształcenie $B(i,j) \mapsto B(i,j)^\gamma$ zwiększa kontrast w zakresie czerni jeżeli $\gamma < 1$ i w zakresie jasnym, jeżeli $\gamma > 1$ (obraz musi być wcześniej znormalizowany, tak aby $0 \leqslant B(i,j) \leqslant 1$).

d) Zmniejsz kontrast: Wartości pikseli pomnożymy przez 0.8 i dodamy do nich 25:

```
A=imread('Lena.bmp','bmp');
B=double(A);
for i=1:512
  for j=1:512
    B(i,j)=25+0.8*B(i,j);
  end;
end;
A=uint8(B);
imwrite(A,'Lena2d.bmp','bmp');
```

Fig. 2.5. Binaryzacja na poziomach 50, 100, 150 i 200

e) Binaryzacja: Każdemu pikselowi przyporządkowujemy wartość 0 jeżeli jego wartość jest poniżej progu i 255 jeżeli powyżej.

f) Powiel obraz, używając odbicia: Rozszerzymy obraz w poziomie, uzupełniając prawą połówkę lustrzanym odbiciem lewej:

```
B=zeros(512,1024);
A=imread('Lena.bmp','bmp');
for i=1:512
  for j=1:512
    B(i,j)=A(i,j);
    B(i,1025-j)=A(i,j);
```

```
  end;
end;
A=uint8(B);
imwrite(A,'Lena2e.bmp','bmp');
```

Fig. 2.6. Odbicie poziome

g) Wycięcie fragmentu: Wytniemy kwadratowy fragment obrazka, zastępując resztę białym tłem:

Fig. 2.7. Wycięcie fragmentu

```
A=imread('Lena.bmp','bmp');
for i=1:128
  for j=1:512
    A(i,j)=255;
    A(513-i,j)=255;
    A(j,i)=255;
    A(j,513-i)=255;
  end;
end;
imwrite(A,'Lena2f.bmp','bmp');
```

h) Wycięcie gładkie: Wytniemy kwadratowy kawałek obrazka, ale tym razem przejście do białego tła będzie płynne:

Fig. 2.8. Wycięcie fragmentu z gładkim przejściem do tła

```
A=imread('Lena.bmp','bmp');
B=double(A);
mask=zeros(512,1);
for i=1:256
  if (i>108)&(i<129)
    mask(i)=(i-108)/21;
    mask(513-i)=mask(i);
  end;
  if i>128
    mask(i)=1;
    mask(513-i)=1;
  end;
end;
for i=1:512
```

```
  for j=1:512
    B(i,j)=255-mask(i)*mask(j)*(255-B(i,j));
  end;
end;
A=uint8(B);
imwrite(A,'Lena2g.bmp','bmp');
```

i) Zaszumienie: Dodamy do obrazka tak zwany „biały szum", to znaczy do każdego piksela niezależnie dodamy liczbę losową o rozkładzie normalnym o średniej 0 i jakimś odchyleniu. W Matlabie jest funkcja `randn` która generuje liczbę pseudolosową o rozkładzie normalnym (gaussowskim), o średniej 0 i odchyleniu standardowym 1. Można też użyć składni `randn(512)`, która od razu generuje tablicę 512×512 liczb pseudolosowych, niezależnych o tym samym rozkładzie. W naszym przykładzie tak wygenerowane liczby mnożymy przez 20, w ten sposób rozkład generowanych zmiennych ma odchylenie standardowe 20. Wielkość odchylenia jest związana ze stopniem zaszumienia. Takie zaszumienie symuluje zniekształcenie obrazka często występujące w praktyce, przy przesyłaniu sygnałów, lub przy rejestracji bardzo słabych sygnałów.

```
A=imread('Lena.bmp','bmp');
B=double(A);
for i=1:512
  for j=1:512
    B(i,j)=B(i,j)+20*randn;
  end;
end;
A=uint8(B);
imwrite(A,'Lena2h.bmp','bmp');
```

Fig. 2.9. Zaszumienie obrazka, $\sigma^2 = 20$

j) Rozmycie: Obrazek zostaje rozmyty, pozbawiony ostrości. Jest to zabieg stosowany na przykład jako wstępna obróbka przed algorytmem wykrywania krawędzi. Efektem rozmycia jest osłabienie niektórych rodzajów szumu (na przykład szumu białego). Krawędzie też zostają osłabione (rozmyte), ale z reguły w mniejszym stopniu niż szum.

```
A=imread('Lena.bmp','bmp');
B=double(A);
C=zeros(512);
mask=[2 4 5 4 2;4 9 12 9 4;5 12 15 12 5;4 9 12 9 4;2 4 5 4 2];
mask=mask/159;
for i=1:512
  for j=1:512
    suma=0;
    for k=-2:2
      for l=-2:2
        if (i-k<513)&(j-l<513)&(i-k>0)&(j-l>0)
          prod=B(i-k,j-l);
        else
          prod=0;
        end;
        suma=suma+mask(k+3,l+3)*prod;
      end;
    end;
    C(i,j)=suma;
  end;
end;
A=uint8(C);
imwrite(A,'Lena2i.bmp','bmp');
```

Fig. 2.10. Rozmycie filtrem Gaussa 5x5

Dwa często stosowane filtry Gaussa (maski), 3×3 i 5×5:

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, \qquad \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix},$$

filtry te należy podzielić przez sumę współczynników, odpowiednio 16 i 159.

Fig. 2.11. Rozmycie filtrem Gaussa 3x3

k) Filtr medianowy: Czasem chcielibyśmy osłabić szum w obrazku, ale bez ogólnego „zmiękczania". Można wtedy zastosować tak zwany filtr medianowy. Odczytujemy wartość piksela i jego sąsiadów (na przykład najbliższych sąsiadów). Tak otrzymane wartości sortujemy. Pikselowi przypisujemy wartość znajdującą się w środku posortowanej listy (jeżeli mamy parzyście wiele wartości, to pikselowi przyporządkowujemy średnią arytmetyczną wartości w środku posortowanej listy. Filtr medianowy dobrze usuwa niektóre rodzaje szumu, na przykład tak zwany „speckle noise", bez ogólnego „zmiękczenia" obrazka.

```
A=imread('Lena.bmp','bmp');
B=double(A);
C=zeros(512);
lista=zeros(9,1);
for i=2:511
  for j=2:511
    m=1;
    for k=-1:1
      for l=-1:1
        lista(m)=B(i+k,j+l);
```

Fig. 2.12. Filtr medianowy, obejmujący najbliższych sąsiadów

```
        m=m+1;
      end;
    end;
    for k=1:8
      for l=1:9-k
        if lista(l)>lista(l+1)
         t=lista(l);
         lista(l)=lista(l+1);
         lista(l+1)=t;
        end;
      end;
    end;
    C(i,j)=lista(5);
  end;
end;
A=uint8(C);
imwrite(A,'Lena2j.bmp','bmp');
```

Zastosowanie filtru medianowego, podobnie jak filtru Gaussa daje efekt osłabienia szumu białego. Efekt ten najlepiej widać na obrazku kontrastowym. Obrazek `test` składa się z pikseli o wartości 80 i 180 (dosyć ciemne i dosyć jasne), i ma wyraźne krawędzie. Do obrazka dodajemy nieco szumu (odchylenie standardowe 5), a następnie stosujemy filtr Gaussa z maską 5×5, oraz filtr medianowy, też o głębokości 5×5. Na obrazkach możemy porównać efekty.

Oba filtry osłabiają szum, ale filtr Gaussa zmiękcza krawędzie, natomiast filtr medianowy pozostawia krawędzie nienaruszone, jedynie „obgryza” narożniki. Łatwo sobie wytłumaczyć jak to się dzieje.

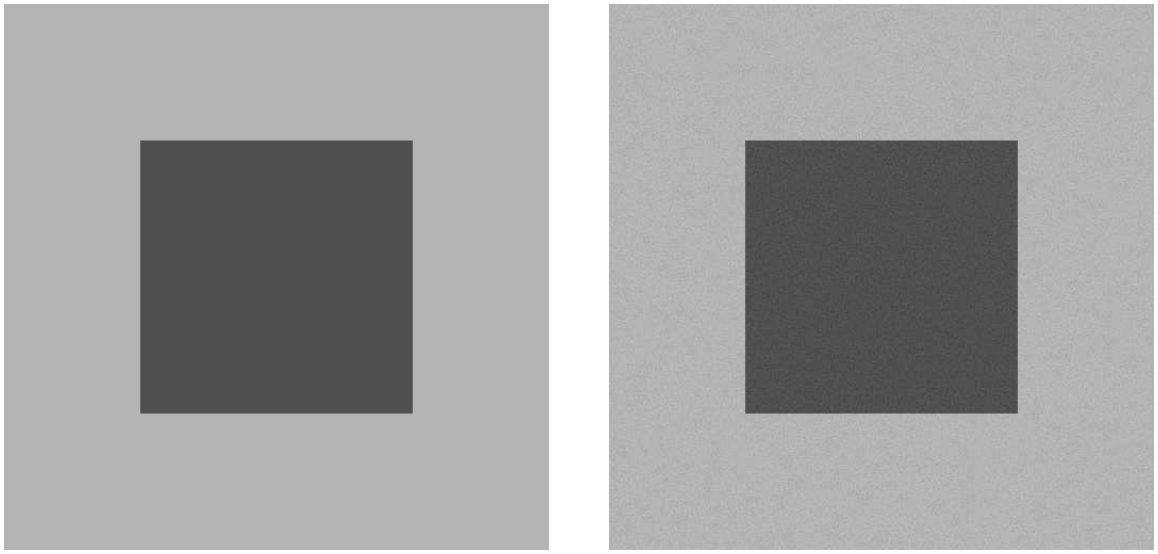

Fig. 2.13. Obraz testowy, bez szumu i z lekkim szumem, $\sigma^2 = 5$

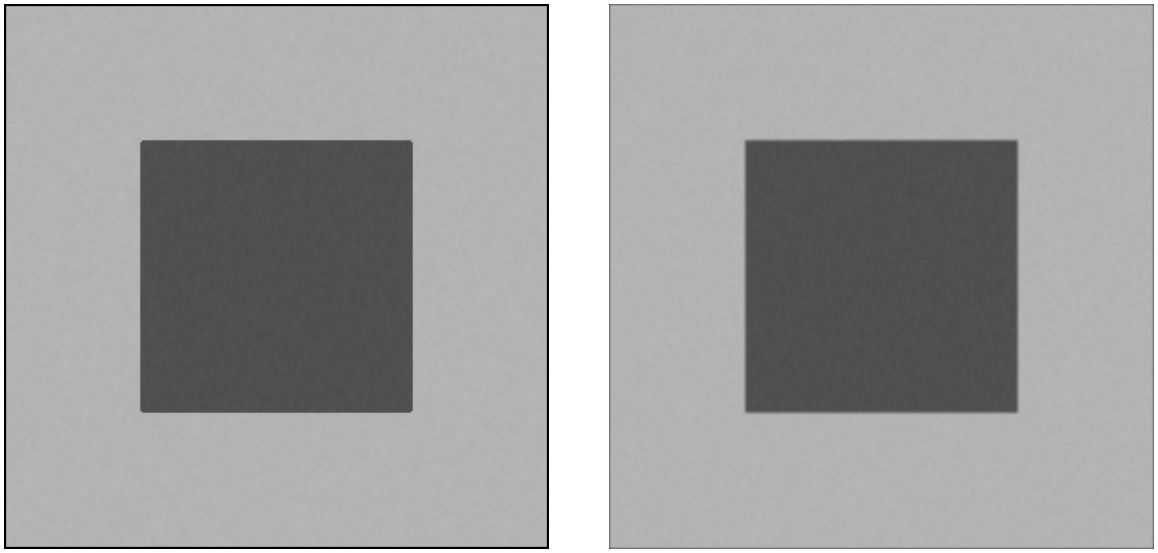

Fig. 2.13. Z lewej zastosowano filtr medianowy, a z prawej filtr Gaussa, oba o wielkości 5×5

3. Transformata Fouriera. Transformata Fouriera to jest nasze główne narzędzie na wykładzie. W praktycznej obróbce obrazków nie będziemy stosowali transformaty Fouriera. W praktyce lepsza jest transformata falkowa. Spróbujmy obliczyć transformatę Fouriera obrazka. W Matlabie są funkcje `fft` i `fft2` obliczające 1– i 2–wymiarową transformatę Fouriera, używające algorytmu szybkiej transformaty. Transformata ma wartości zespolone, nawet dla sygnałów, które mają wartości tylko rzeczywiste. Żeby zwizualizować transformatę osobno wyświetlimy tablicę modułów wartości, i osobno tablicę argumentów (faz) wartości. Do obliczania modułu używamy funkcji `abs(X)`, którą można stosować do zmiennych typu rze-

czywistego i zespolonego, a do obliczania argumentu używamy funkcji `angle`, która zwraca argument jako kąt w zakresie $(-\pi, \pi]$.

```
A=imread('Lena.bmp','bmp');
B=double(A);
C=fft2(B);
A=uint8(abs(C)/20);
imwrite(A,'Lena3a.bmp','bmp');
A=uint8((pi+angle(C))*128/pi);
imwrite(A,'Lena3b.bmp','bmp');
```

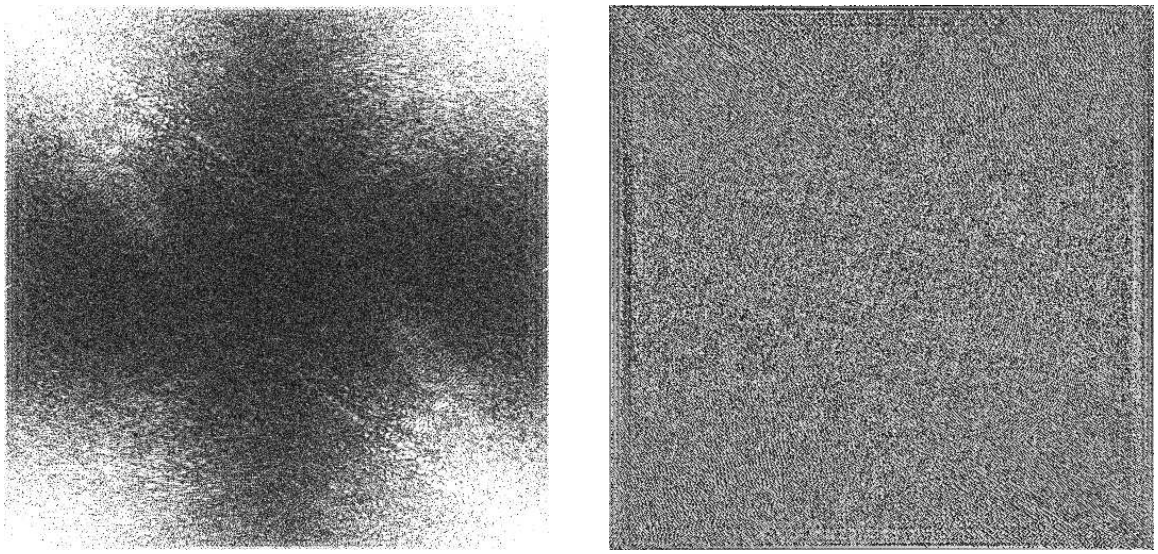

Fig. 3.1. Moduł i argument transformaty Fouriera Leny

Na obrazku 3.1 współczynniki odpowiadające niskim częstotliwościom rozmieszczone są w rogach obrazka. Czasem obraz częstotliwościowy przedstawia się ze współczynnikami niskich częstotliwości umieszczonymi w centrum. W Matlabie jest specjalna funkcja do takiego przesunięcia obrazka `fftshift(C)`. Na obrazku 3.2 przedstawiona jest tak przesunięta transformata Fouriera.

Transformata Fouriera nie jest lokalna. To znaczy, że nawet jeżeli dwa obrazki różnią się tylko na jakimś małym obszarze, to ich transformaty różnią się wszędzie. Na obrazku 3.3 widzimy obraz Leny, z niewielką modyfikacją w okolicach środka. Obok przedstawiony jest obraz modułu różnicy transformaty Fouriera Leny zwykłej i zmodyfikowanej.

Kodowanie pasmowe: Typowym przekształceniem obrazka jest tak zwane kodowanie pasmowe. Zerujemy te współczynniki transformaty Fouriera, które leżą w wybranych obszarach. Tradycyjnie różne obszary geometryczne transformaty Fouriera nazywają się pasmami, i stąd nazwa. Typowym przykładem kodowania pasmowego jest filtr dolnoprzepustowy. Zerujemy wszystkie współczynniki transformaty Fouriera, które leżą poza pewnym, powiedzmy kwadratowym otoczeniem początku układu. Fig. 3.4 pokazuje maskę przykładowego filtru dolnoprzepustowego, oraz przefiltrowany obraz. Obraz jest zupełnie dobrej jakości, jeżeli

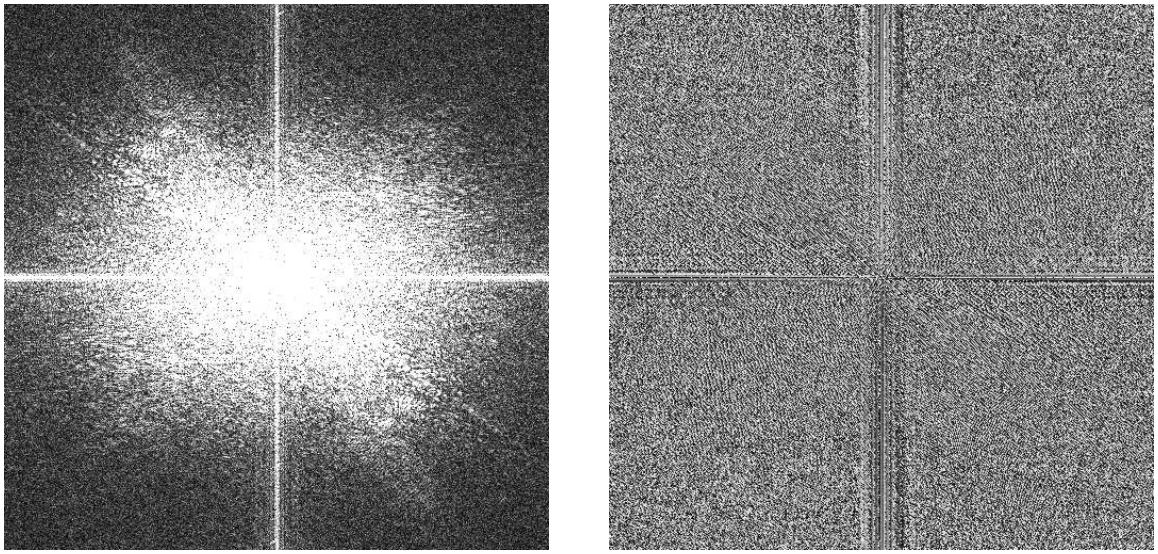

Fig. 3.2. Moduł i argument transformaty Fouriera,
niskie częstotliwości w środku obrazka

Fig. 3.3. Lena z niewielką modyfikacją (prawe oko). Obok moduł różnicy
transformaty Fouriera Leny zwykłej i zmodyfikowanej.

wziąć pod uwagę, że został odtworzony z 12544 współczynników (pozostałe zostały wyzerowane), co stanowi ok. 4,78% całości. Jedyne co optycznie przeszkadza, to okresowo powielone, „drgające" krawędzie.

```
clear;
mask=zeros(512);
for i=200:311
```

Fig. 3.3. Maska filtru dolnoprzepustowego, i przekształcona Lena. Stopień kompresji wynosi ok. 21 (obraz po prawej ma w rezultacie ok. 0,38 bita na piksel (bpp))

```
  for j=200:311
    mask(i,j)=1;
  end
end;
A=imread('Lena.bmp','bmp');
B=double(A);
C=fft2(B);
C=fftshift(C);
C=mask.*C;
C=ifftshift(C);
B=ifft2(C);
A=uint8(abs(B));
imwrite(A,'low.bmp','bmp');
mask=255*mask;
A=uint8(mask);
imwrite(A,'mask.bmp','bmp');
```

4. Rice Wavelet Toolbox. Naszym narzędziem do analizy falkowej obrazków jest darmowy toolbox falkowy (Rice wavelet toolbox) napisany przez grupę ludzi na uniwersytecie Rice. Można go znaleźć w sieci używając słowa kluczowego „rwt". Będziemy używali następujących funkcji z tego pakietu: `mdwt` - transformata falkowa, `midwt` - odwrotna transformata falkowa oraz `daubcqf` - program generujący filtry falkowe Daubechies. Funkcji używamy następująco

```
h0=daubcqf(N);
```

lub

```
[h0,h1]=daubcqf(N);
```

N jest długością filtru, musi być liczbą parzystą. Dłuższe filtry powinny dawać lepsze rezultaty, ale działają wolniej, i generują większe błędy zaokrągleń. Typowe długości to 2,6,10. Będziemy porównywać nasze algorytmy dla różnych długości filtrów. Uzyskane przy pomocy funkcji `daubcqf` filtry stanowią parametr transformaty falkowej i transformaty odwrotnej. Podajemy tylko współczynniki filtru $h0$ (dolnoprzepustowego), Matlab sam wyliczy współczynniki pasującgo filtru $h1$.

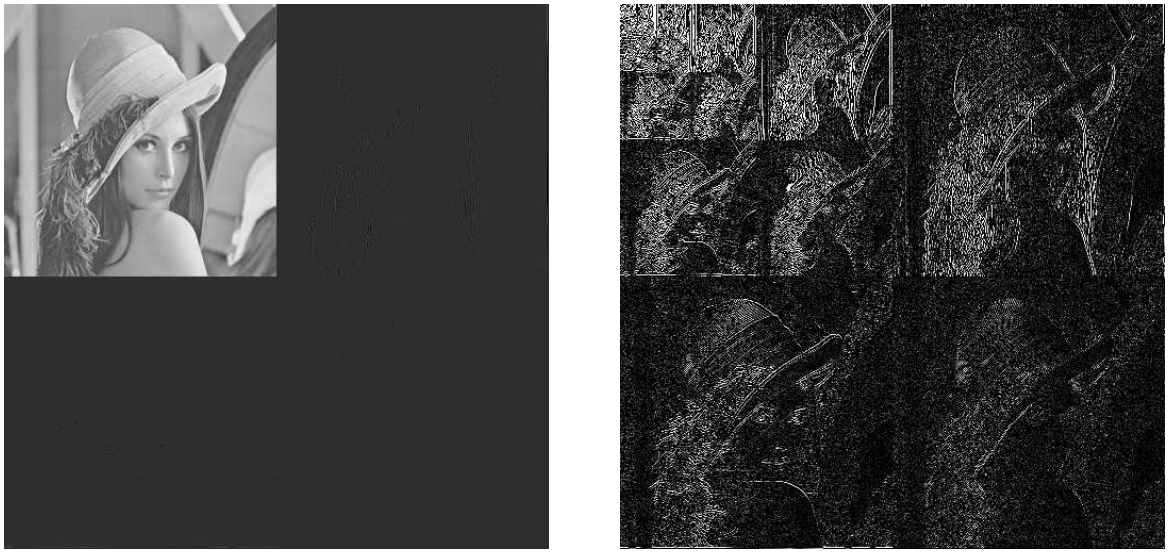

Fig. 4.1. Transformata falkowa głębokości 1 i głębokości 9 (silnie rozjaśniona)

```
[B,L]=mdwt(A,h0,L);      [A,L]=midwt(B,h0,L);
```

A jest obrazkiem, który powinien być kwadratowy o boku będącym potęgą 2. L jest parametrem określającym głębokość transformaty. Wartość parametru L powinna być taka sama dla transformaty i transformaty odwrotnej — w przeciwnym razie obraz nie będzie prawidłowo zrekonstruowany. Wartość L może się zawierać w przedziale od 1 do $\log_2 N$, gdzie N jest długością (w pikslach) boku obrazu. Transformata `mdwt` i transformata odwrotna `midwt` wymagają danych typu `double`. Jeżeli zastosujemy ją do danych typu `uint8` (takich, jakie zwraca `imread`), to transformaty albo będą źle policzone, albo program się wysypie. Przetransformowany obraz możemy obejrzeć instrukcją `image`, pamiętając o ustawieniu odpowiedniej colormapy, i o ewentualnym przekształceniu zakresu wartości transformaty do przedziału [0,255].

5. Kompresja obrazów. Nasze podejście do kompresji będzie bardzo proste. Pierwsza obserwacja jest taka, że transformata falkowa obrazu jest prawie cała czarna. Większość współczynników (dla realistycznego obrazu, takiego jak fotografia) jest bardzo mała. Ustalimy sobie próg ϵ, i wszystkie współczynniki o wartości bezwzględnej poniżej progu zmienimy na 0. W ten sposób w obrazku pozostanie niewiele niezerowych współczynników. Format obrazu z którym pracujemy (`.bmp`) nie kompresuje danych, więc efektów kompresji nie zauważymy w rozmiarze kompresowanego pliku. Dlatego będziemy obliczali stopień kompresji w sposób uproszczony. Każdy wyzerowany współczynnik policzymy, i na koniec podzielimy przez ilość wszystkich pikseli.

Zerowanie współczynników o wartościach poniżej ustalonego progu będziemy nazywać progowaniem (po angielsku „thresholding"). Będziemy rozważać dwa rodzaje progowania, tak zwane progowanie twarde i progowanie miękkie. Różnicę objaśniają wykresy funkcji progujących. Programowanie twarde jest koncepcyjnie prostsze, ale wprowadza do obrazka sztuczne nieciągłości. Z kolei progowanie miękkie obniża kontrast transformaty, i w przypadku wysokich progów należy kontrast wyrównać przed zastosowaniem transformaty odwrotnej.

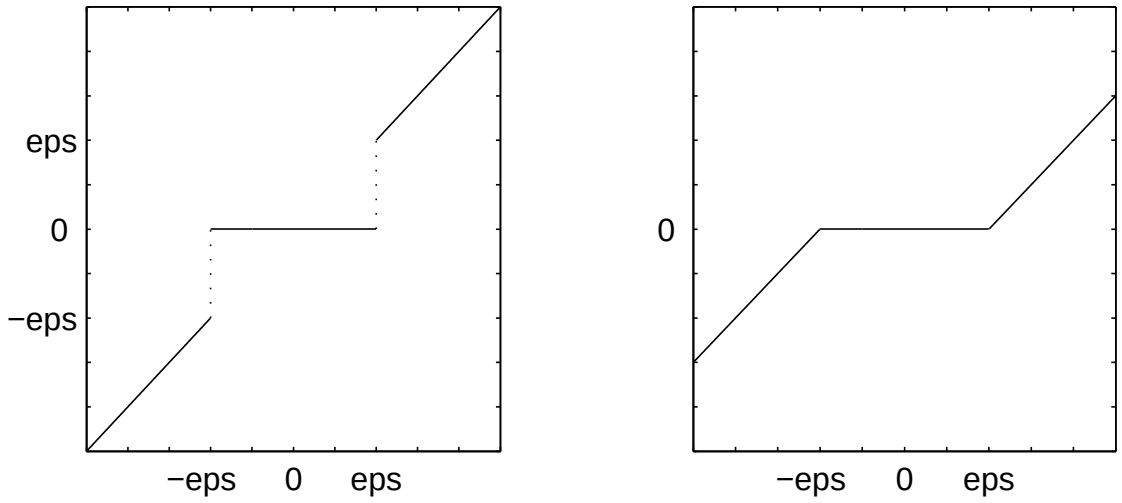

Fig. 5.1. Progowanie twarde i miękkie

Będziemy porównywać subiektywną, optyczną jakość obrazów skompresowanych filtrami Daubechies różnej długości, o różnym stopniu kompresji,i kompresowanych z użyciem obu metod progowania. Przykładowe procedury mogą więc być następujące.

```
A=imread('Lena.bmp','bmp');
A=double(A);
N=6;% 2,10 itp
h0=daubcqf(N);
L=9;
[B,L]=mdwt(A,h0,L);
eps=50;% 30, 100 itp
il=0;
for i=1:512
  for j=1:512
    if abs(B(i,j))<eps
      B(i,j)=0;
      il=il+1;
    end;
  end;
end;
A=midwt(B,h0,L);
```

```
image(A);
100*il/(512*512)
```

Progowanie miękkie możemy zaimplementować następująco

```
if B(i,j)>0
  B(i,j)=B(i,j)-eps;
  if B(i,j)<0
    B(i,j)=0;
    il=il+1;
  end;
else;
  B(i,j)=B(i,j)+eps;
  if B(i,j)>0
    B(i,j)=0;
    il=il+1;
  end;
end;
```

Jeżeli będziemy używać dużej wartości progu ϵ, to należy wyrównać poziomy. To znaczy, należy przed progowaniem znaleźć

$$M = \max_{i,j} |B(i,j)|,$$

a następnie, po progowaniu pomnożyć

$$B(i,j) = B(i,j) * M/(M - \epsilon).$$

Powinniśmy eksperymentować z ϵ tak, aby uzyskać typowe wartości stopnia kompresji: 90%, 95%, 98%. W przypadku trafienia we właściwy stopień kompresji obraz skompresowany należy zapisać, nadając mu nazwę umożliwiającą identyfikację stopnia kompresji, długości filtru i rodzaju progowania. postarajmy się wyciągnąć wnioski na temat roli długości filtru oraz stopnia progowania. Porównajmy wyniki dla kilku różnych obrazków. Wypróbujmy ten algorytm również na obrazkach typu grafika komputerowa.