تمرین ۶

هادی تمیمی
۹۶۲۲۷۶۲۴۰۸

اطلاعات گزارش
تاریخ: ۱۴۰۰/۳/۶

واژگان کلیدی:
تبدیل موجک
هرم لاپلاسی
هرم موجک
حذف نویز

چکیده
به معرفی تبدیل موجک و دلایل استفاده از آن به جای تبدیل فوریه میپردازیم.
هرم لاپلاسین و هرم موجک را در مرحله بعد ایجاد کرده و چند روش حذف نویز مبتنی بر موجک بررسی میشود.

۱-مقدمه

در تبدیل فوریه، اطلاعات زمانی و مکانی را متناسب با فرکانس داریم.در نتیجه در فرکانس های پایین، رزولوشن زمانی خوب نداریم ولی رزولوشن فرکانسی خوب داریم. هر چه فرکانس به سمت بالا برود رزولوشن زمانی بهتر و رزولوشن فرکانسی ضعیفتر میشود. میتوان گفت تبدیل فوریه برای سیگنالهای ایستا که با زمان تغییر نمیکنند خوب عمل میکند.
تبدیل موجک (Wavelet Transform) یکی از تبدیلات مهم ریاضی است که در حوزه‌های مختلف علوم کاربرد دارد. ایده اصلی تبدیل موجک این است که بر ضعف‌ها و محدودیت‌های موجود در تبدیل فوریه غلبه کند. این تبدیل را بر خلاف تبدیل فوریه، می‌توان در مورد سیگنال‌های غیر ایستا و سیستم‌های دینامیک نیز مورد استفاده قرار داد.

۲-شرح تکنیکال

سگینال‌های ایستا:حاوی اجزای طیفی هستند که با زمان تغییر نمی‌کنند.

- تمام اجزای طیفی همیشه وجود دارند.
- نیازی به اطلاعات زمانی نیست.
- FT برای سیگنالهای ایستا خوب عمل می‌کند.

سیگنال‌های غیرایستا : محتوای طیفی متغیر با زمان دارند.

- FT تنها مشخص می‌کند که چه اجزایی در طیف وجود دارد و نه زمانی که آن طیف‌ها وجود دارند.
- نیاز به روش‌هایی برای تعیین زمانی اجزای طیفی است.

تبدیل فوریه زمان کوتاه اطلاعات زمانی را به کمک محاسبه تبدیل فوریه های مختلف برای بازه های زمانی متوالی و سپس در کنار هم قرار دادن آنها فراهم میکند. اگر در این تبدیل اندازه پنجره را خیلی بزرگ در نظر بگیریم، رزولوشن فرکانسی خوب و رزولوشن زمانی ضعیف داریم. اگر اندازه پنجره را خیلی باریک در نظر بگیریم، رزولوشن زمانی خوب و رزولوشن فرکانسی ضعیف داریم. در تعیین اندازه پنجره نکته ی قابل توجه، در نظر داشتن اصل عدم قطعیت Heisenberg میباشد:

$$\Delta t \cdot \Delta f \geq \frac{1}{4\pi}$$

طبق این اصل نمی توانیم هر دوی رزلوشن زمانی و فرکانسی را به دلخواه زیاد کنیم. دقیقا نمی دانیم که در چه لحظه ای یک فرکانس خاص اتفاق می افتد . تنها می توانیم بفهمیم که چه محدوده ی فرکانسی در چه فاصله ی زمانی رخ می دهد.

۶ هرم رزولوشن

اطلاعات تصویر در رزلوشن های مختلف قرار دارد.با اعمال عملیات بر روی یک سطح رزولوشن تصویر و downsample کردن به سطح پایین تر میرویم. این سطوح در کنار هم هرم رزولوشن را تشکیل میدهند.

با فرض اینکه یک تصویر n*n داریم و $n= 2_j$:

- ماکسیمم تعدادی که میتوانیم عملیات تجزیه یا ساخت هرم در سطوح مختلف رو داشته باشیم، j+1 است .به همین ترتیب، کل تعداد پیکسلهای هرم برابر است با:

$$N_2+ (12)_2N_2+(12)_4N_2+\cdots+1=43N_2$$

- مقایسه تعداد پیکسل های تصویر اصلی با تعداد کل پیکسل های هرم:

$$N_2+ (12)_2N_2+(12)_4N_2+\cdots N_2=1+(12)_2+(12)_4+\cdots=43=1.33$$

- فواید بیشتر بودن تعداد پیکسل ها در هرم: گاهی به دنبال اطلاعاتی هستیم که در رزولوشن فعلی آنها را نداریم و نیاز داریم در رزولوشن های پایین تر و کمتر آنها را پیدا کنیم.

۶.۱.۳

هرم لاپلاسی با استفاده از تفاضل بین تصویر در رزلوشن خاصی از هرم گوسی و نسخه ی بزرگ شده با رزلوشن پایین تر بدست می آید. در واقع در هر مرحله به این صورت عمل میکنیم:

- تصویر فعلی این مرحله را از یک فیلتر گاوسی (یا فیلتر پایین گذر) عبور میدهیم.
- تصویر که فرکانس های بالای آن حذف شده را downsmaple میکنیم.
- تصویر downsample شده را upsample کرده و آن را از تصویر این مرحله کم میکنیم.
- لاپلاسی این مرحله بدست می آید.
- تصویر downsample شده،تصویر مرحله بعد هرم می باشد.

- در نهایت تمام لاپلاسین ها+تصویر مرحله آخر را ذخیره میکنیم.

همچنین باید دقت شود در هر مرحله سایز تصویر نصف میشود.
اگر تصویرمان $N \times N$ باشد که $N = 2^J$، حداکثر تعداد مرحله ای که میتوان داشت برابر با $\log_2 N = J + 1$ میباشد.

مجموع تعداد پیکسل های تمام مراحل برابر

$$T = 1 \times 1 + 2 \times 2 + 4 \times 4 + 8 \times 8 + \cdots = 2^0 + 2^2 + 2^4 + 2^6 + \cdots$$

می باشد که اگر فرمول دنباله هندسی را در نظر بگیریم:

$$Total\ pixels = 1 \times \frac{4^{J+1}-1}{4-1}$$

حال اگر تعداد پیکسل های تصویر اصلی را حساب کنیم خواهیم داشت:

$$\text{Total pixels original} = 2^J \times 2^J = 2^{J+1}$$

این بدان معناست که تعداد پیکسل های هرم لاپلاسی بیشتر از حالت اورجینال میشود. در صورتی که جزئیات را در نظر بگیریم،اکثر آنها ماتریس های sparse می باشند که حجم کمتری را اشغال میکنند.درست است که تعداد پیکسل ها بیشتر میشود، اما به دلیل استفاده از این ماتریکس ها فشرده سازی خواهیم داشت.

۶.۱.۴

در مرحله قبل نحوه ایجاد هرم لاپلاسین را توضیح دادیم.حال به دنبال نحوه بازسازی تصویر اصلی از هرم هستیم.تصویر مرحله آخر را که جدا کرده ایم upsample کرده (به کمک درون یابی pixel replication) و سایز آن را دو برابر میکنیم.حال این تصویر را با لاپلاسین این مرحله جمع میکنیم. این دو مرحله را انقدر تکرار کرده تا تصویر با لاپلاسین مرحله اول جمع شود و به تصویر اصلی برسیم.

۶.۱.۵

هرم موجک به کمک خاصیت جداپذیری تبدیل موجک و به کمک فیلترهای یک بعدی haar ایجاد میشود.
با هر بار اعمال فیلتر به تصویر،تصویر را به چهار بخش تقسیم میکنیم.
بخش approximation تصویر با رزولوشن کوچک تر می باشد.اگر دوباره به آن فیلتر را اعمال کنیم،به چهار بخش شکسته خواهد شد.بخش های vertical,horizontal,dianogal هر کدام جزئیات عمودی،افقی و قطری تصویر را در آن رزولوشن خاص نشان می دهد.از نظر تعداد پیکسل ها،تفاوتی با تصویر اصلی ندارد ولیبه دلیل sparse بودن ماتریس های جزئیات،میتواند درفشرده سازی استفاده شود.
اگر بخواهیم آنرا با هرم لاپلاسی مقایسه کنیم،از نظرپیچیدگی حافظه ای هرم گاوسی شرایط بهتری دارد.از نظر پیچیدگی زمانی با توجه به اینکه برای هرم لاپلاسی،یک اعمال فیلتر جعبه داریم اما در هرم موجک

با اعمال فیلتر haar داریم میتوان گفت هرم لاپلاسی شرایط بهتری دارد.

۶.۱.۶

به کمک چندی سازی هرم بدست آمده در بخش قبل میتوان به فشرده سازی تصویر کمک نمود.به این صورت که در هر مرحله تابع چندی سازی را بر هر ۴ زیر باند اعمال میکنیم.یعنی ۳ بخش جزئیات و بخش تقریب همه این تابع بر روی آنها اعمال می شوند.تابعی که اعمال میکنیم به صورت زیر می باشد.

$$c'(u,v) = \gamma \times sgn[c(u,v)] \times \left\lfloor \frac{|c(u,v)|}{\gamma} \right\rfloor$$

۶.۲

بخش جزئیات در تبدیل موجک یک ماتریس sparse میباشد. معنی آن این است که اکثر مقادیر آن یا صفر بوده و یا بسیار نزدیک به صفر می باشند.در صورت وجود نویز اگر مقادیر خیلی کوچک را به کمک threshold صفر در نظر بگیریم،میتوان به حذف نویز کمک کرد.روش های مختلفی برای تعیین و اعمال threshold وجود دارد.

- soft threshold استفاده میکنیم. فرمول آن برای سیگنال یک بعدی به صورت زیر می باشد که میتوان آنرا به فرم دو بعدی بسط داد:

$$y_{soft}(t) = \begin{cases} \mathrm{sgn}(x(t)) \cdot (|x(t) - \delta|), & |x(t)| > \delta \\ 0, & |x(t)| < \delta \end{cases}$$

- روش آستانه گذاری Garrote روش دیگری است که از دو روش دیگر بهتر عمل میکند و تصویر بازسازی شده با کیفیت بالاتری ارائه میدهد:

$$garrote\ thrshld(x,\lambda) = \{x - \lambda 2x \ |x| > \lambda 0 \ |x| \geq \lambda$$

- روش دیگر حذف باند HH است. از آنجا که نویز اطلاعات ناخواسته ای هست که در بخش جزئیات تصویر تبدیل موجک یافته وجود دارد و اغلب در فرکانس های بالای تصویر موجود است، پس با حذف باند HH از ضرایب موجک انتظار داریم نویز تا حدود زیادی از بین برود.

۲-شرح نتایج

۶.۱.۳

تصویر اصلی

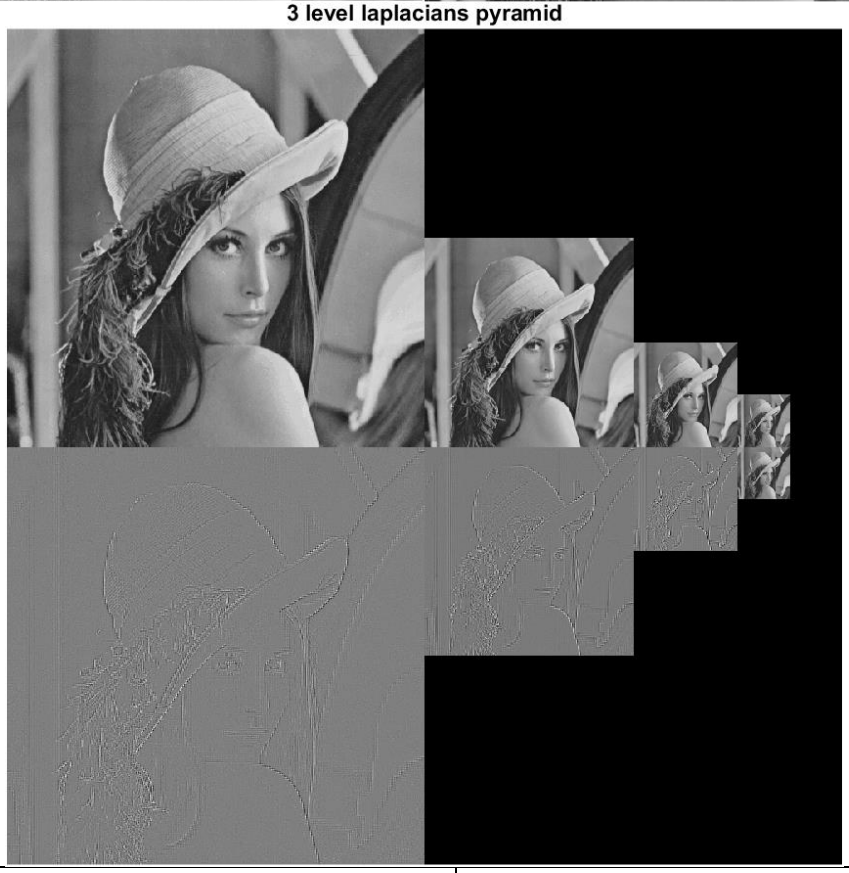
تصویر بازسازی شده

9 level laplacians pyramid

Psnr	Inf
Mse	0

نتایج نشان دهنده آن است که در تبدیل به کمک هرم لاپلاسی،هیچ داده ای از دست نمیرود.حتی اگر تا آخرین سطح تجزیه را انجام دهیم. دلیل آن این است که که جزئیات و داده های مهم را در لاپلاسین ها ذخیره میکنیم.زمانی که فرکانس های بالای تصویر را در لاپلاسین ها ذخیره کردیم و بالاترین فرکانس سیگنال کاهش یافت،قاعده شانون به ما اجازه میدهد نرخ نمونه برداری را بدون از دست دادن داده ای کاهش دهیم.

۶.۱.۴



Psnr	Inf
Mse	0

نتیجه مشابه سوال قبل است

۶.۱.۵

هرم موجک لول ۳ تصویر بازسازی شده

Psnr	Inf
Mse	0

نتایج نشان دهنده آن است که در تبدیل به کمک هرم موجک،هیچ داده ای از دست نمیرود.دلیل آن این است که جزئیات مختلف تصویر برای هر رزولوشن به صورت جداگانه در سه بخش ذخیره می شود و حین بازسازی از آنها برای ساخت تصویر اصلی استفاده میکنیم.

۶.۱.۶

هرم موجک لول ۳ تصویر بازسازی شده

Psnr	46.3947
Mse	1.4589

نتایج نشان دهنده آن است که در تبدیل به کمک هرم موجک و چندی سازی مقدار بسیار کمی از دست دادن داده داریم. با این حال این مقدار خطا قابل چشم پوشی است.

۶.۲

تصویر نویز دار شده ی گوسی　　تصویر رفع نویز شده با آستانه گذاری soft

Psnr	24.036
Mse	311.566

تصویر نویز دار شده ی گوسی　　تصویر رفع نویز شده با حذف زیر باند HH

Psnr	108.518
Mse	29.646

طبق داده های به دست آمده رفع نویز شده با حذف زیر باند HH نتیجه ی بهتری می دهد.

کد ها:

۶.۱.۳

```
img = imread('Images/6/Lena.bmp');
img = rgb2gray(img);
figure
imshow(img);

levels = 9;
[pyramid,laplacians] = laplace_pyramid(img,levels);

[n,~] = size(laplacians);

last_img = laplacians{n};
for i=1:levels
    n = n-1;
    last_img = pixel_rep(last_img);
    last_img = last_img + laplacians{n};
end
last_img = uint8(last_img);

psnr = psnr(img,last_img);
psnr
figure
imshow(last_img);
mse = immse(last_img,img);
mse

figure
imshow(pyramid);
title('9 level laplacians pyramid');


function [final,laplacians_last_img] = laplace_pyramid(img,levels)

    avg_filter = [1 1;1 1]/4;
    [R,C]  = size(img);
    final = zeros(2*R,2*C,'uint8');
    old_img = double(img);

    N = levels;
    laplacians_last_img = cell(N,1);
    for i=1:levels

        r1 = R-(0.5)^(i-1)*R+1;
        r2 = R;
        c1 = ((2^(i-1)-1)/(2^(i-2)))*C+1;
        c2 = (2^(i)-1)/(2^(i-1))*C;
        final(r1:r2,c1:c2) = old_img;
        new_img = average_filter(old_img,avg_filter);
        laplace = old_img - pixel_rep(new_img);
        laplacians_last_img{i} = laplace;
        laplace = norm(laplace);
        laplace = uint8(laplace);

        r1 =R+1;
        r2 = R+(0.5)^(i-1)*R;

        final(r1:r2,c1:c2)=laplace ;
        old_img = new_img;
    end
```

```
    i = levels+1;
    r1 = R-(0.5)^(i-1)*R+1;
    r2 = R;
    c1 = ((2^(i-1)-1)/(2^(i-2)))*C+1;
    c2 = (2^(i)-1)/(2^(i-1))*C;
    final(r1:r2,c1:c2) = old_img;
    r1 =R+1;
    r2 = R+(0.5)^(i-1)*R;
    final(r1:r2,c1:c2) = old_img;

    laplacians_last_img{levels+1}=old_img;
end
function output = pixel_rep(img)
    [r,c] = size(img);
    output = zeros(2*r,2*c,class(img));
    for x = 1:r
        for y = 1:c
            j = 2*(x-1) + 1;
            i = 2*(y-1) + 1;
            output(j,i) = img(x,y);
            output(j+1,i) = img(x,y);
            output(j,i+1) = img(x,y);
            output(j+1,i+1) = img(x,y);
        end
    end
end
function output=average_filter(image,filter)
    [R,C] = size(image);
    output = zeros(R/2,C/2,'double');
    for i=1:2:R
        for j=1:2:C
            part = double(image(i:i+1,j:j+1));
            mult = part.*filter;
            out = sum(mult,'all');
            output(ceil(i/2),ceil(j/2)) = out;
        end
    end
end

function output = norm(img)
    output = mat2gray(img);
    Max = max(max(output));
    Min = min(min(output));
    output = (255/(Max-Min))*output;
end
```

۶.۱.۴

```
img = imread('Images/6/Lena.bmp');
img = rgb2gray(img);
figure
imshow(img);

levels = 3;
[pyramid,laplacians] = laplace_pyramid(img,levels);

[n,~] = size(laplacians);

last_img = laplacians{n};
for i=1:levels
    n = n-1;
```

```
    last_img = pixel_rep(last_img);
    last_img = last_img + laplacians{n};
end
last_img = uint8(last_img);

psnr = psnr(img,last_img);
psnr
figure
imshow(last_img);
mse = immse(last_img,img);
mse

figure
imshow(pyramid);
title('3 level laplacians pyramid');


function [final,laplacians_last_img] = laplace_pyramid(img,levels)

    avg_filter = [1 1;1 1]/4;
    [R,C]  = size(img);
    final = zeros(2*R,2*C,'uint8');
    old_img = double(img);

    N = levels;
    laplacians_last_img = cell(N,1);
    for i=1:levels

        r1 = R-(0.5)^(i-1)*R+1;
        r2 = R;
        c1 = ((2^(i-1)-1)/(2^(i-2)))*C+1;
        c2 = (2^(i)-1)/(2^(i-1))*C;
        final(r1:r2,c1:c2) = old_img;
        new_img = average_filter(old_img,avg_filter);
        laplace = old_img - pixel_rep(new_img);
        laplacians_last_img{i} = laplace;
        laplace = norm(laplace);
        laplace = uint8(laplace);

        r1 =R+1;
        r2 = R+(0.5)^(i-1)*R;

        final(r1:r2,c1:c2)=laplace ;
        old_img = new_img;
    end

    i = levels+1;
    r1 = R-(0.5)^(i-1)*R+1;
    r2 = R;
    c1 = ((2^(i-1)-1)/(2^(i-2)))*C+1;
    c2 = (2^(i)-1)/(2^(i-1))*C;
    final(r1:r2,c1:c2) = old_img;
    r1 =R+1;
    r2 = R+(0.5)^(i-1)*R;
    final(r1:r2,c1:c2) = old_img;

    laplacians_last_img{levels+1}=old_img;
end
function output = pixel_rep(img)
    [r,c] = size(img);
    output = zeros(2*r,2*c,class(img));
    for x = 1:r
```

```
        for y = 1:c
            j = 2*(x-1) + 1;
            i = 2*(y-1) + 1;
            output(j,i) = img(x,y);
            output(j+1,i) = img(x,y);
            output(j,i+1) = img(x,y);
            output(j+1,i+1) = img(x,y);
        end
    end
end
function output=average_filter(image,filter)
    [R,C] = size(image);
    output = zeros(R/2,C/2,'double');
    for i=1:2:R
        for j=1:2:C
            part = double(image(i:i+1,j:j+1));
            mult = part.*filter;
            out = sum(mult,'all');
            output(ceil(i/2),ceil(j/2)) = out;
        end
    end
end

function output = norm(img)
    output = mat2gray(img);
    Max = max(max(output));
    Min = min(min(output));
    output = (255/(Max-Min))*output;
end
```

۶.۱.۵

```
img = imread('Images/6/Lena.bmp');
img = rgb2gray(img);

level = 3;
output = wt(img,level,0);

x = iwt(output,level);
x = norm (x);
x = uint8(x);
imshow(output);
figure
imshow(x);
m = immse(img,x);
m
p = psnr(img,x);
p


imwrite(x,'x.png');
function output = iwt(wt_pyramid,level)
    if(level==0)
        output = wt_pyramid;
        return
    end
    output = wt_pyramid;
    [R,C] = size(output);
    dec_part = output(1:R/(2^(level-1)),1:C/(2^(level-1)));


    [r,c] = size(dec_part);
    cA = dec_part(1:r/2,1:c/2);
```

```
    cV = dec_part(r/2+1:r,1:c/2);
    cH = dec_part(1:r/2,c/2+1:c);
    cD = dec_part(r/2+1:r,c/2+1:c);

     output(1:R/(2^(level-1)),1:C/(2^(level-1))) =
idwt2(cA,cH,cV,cD,'haar');

    output = iwt(output,level-1);

end

function output = wt(img,level,present)

    if(level==0)
        output = img;
        return
    end
    [R,C] = size(img);
    output= zeros(R,C,'double');

    [cA,cH,cV,cD] = dwt2(img,'haar');

    if(present)
        cA = norm(cA);
        cV = norm(cV);
        cH = norm(cH);
        cD = norm(cD);
    end
    output(1:R/2,1:C/2) = wt(cA,level-1,present);
    output(R/2+1:R,1:C/2)=cV;
    output(1:R/2,C/2+1:C)=cH;
    output(R/2+1:R,C/2+1:C)=cD;

    if (present)
        output = uint8(output);
    end

end

function output = norm(img)
    output = mat2gray(img);
    Max = max(max(output));
    Min = min(min(output));
    output = (255/(Max-Min))*output;

end
```

۶.۱.۶

```
img = imread('Images/6/Lena.bmp');
img = rgb2gray(img);


level = 4;
output = wt(img,level,0);

x = iwt(output,level);
x = norm (x);
x = uint8(x);
imshow(output);
figure
imshow(x);
p = psnr(x,img);
p
```

```
m = immse(x,img);
m

function output= quantizer (img,gamma)
    [M,N] = size(img);
    output = zeros(M,N,'double');

    for i=1:M
        for j=1:N
            output(i,j) = gamma* sign(img(i,j))*floor(
abs(img(i,j))/gamma);
        end
    end
end
function output = iwt(wt_pyramid,level)
    if(level==0)
        output = wt_pyramid;
        return
    end
    output = wt_pyramid;
    [R,C] = size(output);
    dec_part = output(1:R/(2^(level-1)),1:C/(2^(level-1)));


    [r,c] = size(dec_part);
    cA = dec_part(1:r/2,1:c/2);
    cV = dec_part(r/2+1:r,1:c/2);
    cH = dec_part(1:r/2,c/2+1:c);
    cD = dec_part(r/2+1:r,c/2+1:c);

     output(1:R/(2^(level-1)),1:C/(2^(level-1))) =
idwt2(cA,cH,cV,cD,'haar');

    output = iwt(output,level-1);

end

function output = wt(img,level,present)

    if(level==0)
        output = img;
        return
    end
    [R,C] = size(img);
    output= zeros(R,C,'double');

    %[LoD,HiD] = wfilters('haar','d');
    [cA,cH,cV,cD] = dwt2(img,'haar');

    cA = quantizer(cA,2);
    cV = quantizer(cV,2);
    cH = quantizer(cH,2);
    cD = quantizer(cD,2);

    if(present)
        cA = norm(cA);
        cV = norm(cV);
        cH = norm(cH);
        cD = norm(cD);
    end
    output(1:R/2,1:C/2) = wt(cA,level-1,present);
    output(R/2+1:R,1:C/2)=cV;
```

```
    output(1:R/2,C/2+1:C)=cH;
    output(R/2+1:R,C/2+1:C)=cD;

    if (present)
        output = uint8(output);
    end

end
function output = norm(img)
    output = mat2gray(img);
    Max = max(max(output));
    Min = min(min(output));
    output = (255/(Max-Min))*output;

end
```

۶.۲

```
img = imread('Images/6/Lena.bmp');
img = rgb2gray(img);
noisy = imnoise(img,'gaussian',0.25);

level =3;
output = wt(noisy,level);
x = iwt(output,level);

x = uint8(x);
figure
imshow(noisy);
figure
imshow(x);

p1 = psnr(noisy,img);
m = immse(img,x);
p2 = psnr(img,x);

function output= soft_tresh(img,beta,sigma)
    [R,C] = size(img);
    output = zeros(R,C);

    local_sigma = std(double(img));
    T = (beta*sigma)/local_sigma;
    for i=1:R
        for j=1:C
            x = img(i,j);

            if(abs(x)<T)
                x =0;
            else
                x = sign(x)*abs(x-T);
            end

            output(i,j)=x;

        end
    end

end

function output = wt(img,level)

    if(level==0)
```

```
        output = img;
        return
    end
    [R,C] = size(img);
    output= zeros(R,C,'double');
    [cA,cH,cV,cD] = dwt2(img,'haar');


    beta = sqrt(log2(R/3));
    sigma = median(abs(cD)./0.6745).^2;

    cH = soft_tresh(cH,beta,sigma);
    cD = soft_tresh(cD,beta,sigma);
    cV = soft_tresh(cV,beta,sigma);

    output(1:R/2,1:C/2) = wt(cA,level-1);
    output(R/2+1:R,1:C/2)=cV;
    output(1:R/2,C/2+1:C)=cH;
    output(R/2+1:R,C/2+1:C)=cD;

end

function output = iwt(wt_pyramid,level)
    if(level==0)
        output = wt_pyramid;
        return
    end
    output = wt_pyramid;
    [R,C] = size(output);
    dec_part = output(1:R/(2^(level-1)),1:C/(2^(level-1)));


    [r,c] = size(dec_part);
    cA = dec_part(1:r/2,1:c/2);
    cV = dec_part(r/2+1:r,1:c/2);
    cH = dec_part(1:r/2,c/2+1:c);
    cD = dec_part(r/2+1:r,c/2+1:c);


    output(1:R/(2^(level-1)),1:C/(2^(level-1))) =
idwt2(cA,cH,cV,cD,'haar');

    output = iwt(output,level-1);

end
```

پایان