بنام خدا

دانشگاه صنعتی اصفهان
دانشکده برق و کامپیوتر

تمرین سری پنجم –پردازش تصاویر دیجیتال

استاد: دکتر سعید صدری

پروانه رشوند ۹۴۱۰۱۲۴

رضا سعادتی فرد ۹۴۱۱۳۹۴

تاریخ تحویل ۹۵/۰۹/۱۵

سؤال اول) کد این سؤال فایل های HW5_Q1_Global , HW5_Q1_Local , HW5_Q1_Gamma می‌باشد.

در این سؤال قرار است تصویر یک منظره را ارتقاء کانتراست دهیم. برای ارتقاءکانتراست و بهبود دادن تصاویر روش‌های متعددی وجود دارد. دسته‌ای از روش‌ها تحت عنوان روش‌های نقطه‌ای نام‌گذاری شده‌اند. یکی از روش‌های نقطه‌ای برای بهبود دادن تصاویر، روش بهبود γ می‌باشد. در این روش با تغییر پارامتر γ می‌توان در جهت ارتقاء کانتراست تصویر عمل نمود. در این روش طبق معادله‌ی زیر و همچنین نمودار رسم شده، عمل می‌نماییم. اگر S روشنایی پیکسل‌ها بعد از ارتقاء کانتراست و r سطح روشنایی پیکسل‌ها پس از ارتقاء کانتراست باشد، معادله‌ی زیر را می‌توان در این روش بیان نمود:

$$s = cr^{\gamma}$$

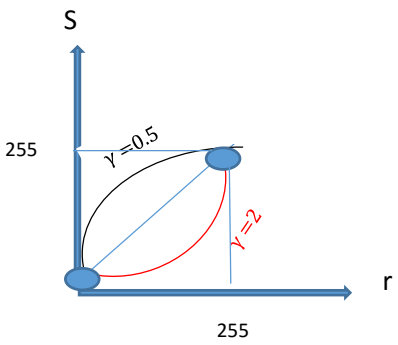
اما سطح روشنایی ۲۵۵ به ۲۵۵ مپ می‌شود لذا می‌توان ثابت c را به راحتی به دست آورد:

$$255 = c255^{\gamma} \rightarrow c = 255^{(1-\gamma)}$$

و بنابراین معادله‌ی ارتقاء کانتراست در روش γ بدین‌صورت درخواهد آمد:

$$s = 255^{(1-\gamma)}r^{\gamma}$$

که منحنی مربوط به این معادله نیز در شکل زیر به ازای $\gamma>1$ و $\gamma<1$ رسم شده است.



همان‌طور که مشاهده می‌شود به ازای $\gamma>1$ پیکسل‌های سیاه و یا تیره، سیاه‌تر میشوند و یا به طور کلی تصویر تاریک تر می‌شود و بنابراین به طور کلی درتصاویر با سطح روشنایی بالا استفاده می‌شود.

همچنین به ازای $\gamma<1$ روشن‌ها روشن تر شده و به طور کلی تصویر روشن تر می‌شود و بنابراین در حالت کلی برای تصاویر تاریک استفاده می‌گردد.

همان‌طور که بیان شد این روش یک روش نقطه ای برای ارتقاء کانتراست در حوزه‌ی مکان می‌باشد و در بسیاری از موارد تنها اعمال همین روش ساده برای بهبود تصاویر کفایت می‌کند. بنابراین در همین ابتدا از این روش استفاده می‌نماییم تا نتیجه را مشاهده نماییم.

تصویر اصلی که قرار است عملیات را روی آن انجام دهیم به صورت زیر می‌باشد:

کد برنامه برای اجرای روش گاما به صورت زیر می‌باشد:

```
clc
clear all
close all
f=imread('lowcontrast3.jpg');
f=rgb2gray(f);
f=im2double(f);
figure,imshow(f),title('original image');
[M,N]=size(f);
h=zeros(1,256);
for m=1:M
    for n=1:N
        r=f(m,n)*255;
        h(r+1)=h(r+1)+1;
    end
end
hist=h/sum(h);
figure,plot(hist),title('histogram');
%%%%%%%%%%%% gma correction %%%%%%%%%%
gama=0.5;
R=zeros(1,256);
S=zeros(1,256);
g=zeros(M,N);
for m=1:M
```

```
    for n=1:N
        r=f(m,n)*255;
        R(r+1)=r;
        s=(255^(1-gama))*(r^gama);
        S(r+1)=s;
        g(m,n)=s;
    end
end

figure,imshow(g,[]),title(['equalized image by gama correction,gama= ',num2str(gama)]);
```

نتایج را به ازای چند گاما تست می‌کنیم تا ببینیم در چه حالتی نتیجه ی بهتری حاصل می‌شود:

ابتدا برای $\gamma>1$: گاما را مقادیر ۲ و ۳ در نظر گرفته و نتایج را مشاهده می‌نماییم:

equalized image by gama correction,gama= 4

و در مرحله بعد نتایج را برای γ<1 مشاهده می‌نماییم:

equalized image by gama correction,gama= 0.2

equalized image by gama correction,gama= 0.05

همان‌طور که مشاهده می‌شود و قبلاً هم بیان شد برای $\gamma>1$ تصویر تیره تری ایجاد شد و برای $\gamma<1$ تصویر روشن تر شد. از آن‌جا که تصویر اصلی بیس روشن داشت در این مورد و در این روش $\gamma>1$ توانست جواب بهتری را ارائه دهد. اما در حالت کلی به نظر می‌رسد که تنها اعمال این روش برای بهبود کانتراست تصویر کافی نبوده چرا که نتیجه به خصوص برای گاماهای کوچکتر از یک اصلا مناسب نیست. اما زمانی که هدف ما شمردن پنجره‌های ساختمان مقابل است گامای ۴ نتیجه ی خیلی بدی را به نسبت سادگی و سرعت اجرای این روش ارائه نکرده است و در مورد این مقصود خاص که در صورت سؤال خواسته شده می‌توان گفت نتیجه برای گامای ۴ در این مقصود خاص خیلی بد نیست.

اما علاوه بر روش‌های نقطه ای روش‌های دیگری نیز برای ارتقاء کانتراست وجود دارد.یکی دیگر از این روش‌ها روش global می‌باشد.

در حالت کلی، از دید انسان تصویری که دارای یک قسمت خیلی روشن و یا یک قسمت خیلی تیره باشد، خوشایند نیست و در حالت کلی می‌توان گفت که از منظر انسان، یک تصویر خوب تصویری است که دارای تابع چگالی احتمال و یا هیستوگرام یکنواخت باشد. یعنی هیستوگرام آن دارای قله و یا دره نباشد. لذا در این روش تمام تلاش بر این است که هیستوگرام تصویر را یکنواخت سازیم تا تصویر بهتری حاصل شود. یکنواخت سازی هیستوگرام درواقع نوعی تبدیل است که بر روی r (سطوح روشنایی قبل از بهبود کانتراست) عمل نموده و s (سطوح روشنایی پس از بهبود کانتراست) را می‌دهد بطوری که تابع چگالی احتمال s حتی‌الامکان یکنواخت شود بنابراین مسئله‌ی ما یافتن این تبدیل است.

برای یافتن این تبدیل می‌دانیم که هر تابع از متغیر تصادفی، خود یک متغیر تصادفی است. اگر x یک متغیر تصادفی باشد، $p_X(x)$ تابع چگالی احتمال آن بوده و $F_X(x)$ تابع توزیع تجمعی می‌باشد که داریم:

$$F_X(x) = \int_{-\infty}^{x} p_X(x)\,dx$$

حال متغیر تصادفی Y را بصورت زیر تعریف می‌نماییم:

$$Y = F_X(x)$$

طبق تعریف تابع توزیع تجمعی، تابع توزیع تجمعی Y به صورت زیر تعریف می‌شود:

$$F_Y(y) = p[Y \ll y] = p[F_X(x) \ll y]$$

می‌دانیم $F_X(x)$ به طور یکنواخت افزایشی است لذا یک به یک بوده و معکوس پذیر است لذا داریم:

$$F_Y(y) = p[F_X(x) \ll y] = p[x \ll F_x^{-1}(y)] = F_x\left(F_x^{-1}(y)\right) = y$$

بنابراین $F_Y(y) = y$ شد یعنی Y دارای تابع توزیع یکنواخت است. بنابراین تبدیلی که به دنبال ان می‌گشتیم یافت شد. یعنی اگر r سطوح روشنایی اولیه باشند، s=$F_r(r)$ دارای توزیع یکنواخت می‌باشد. در روش global از همین روش در جهت بهبود کانتراست استفاده می شود.

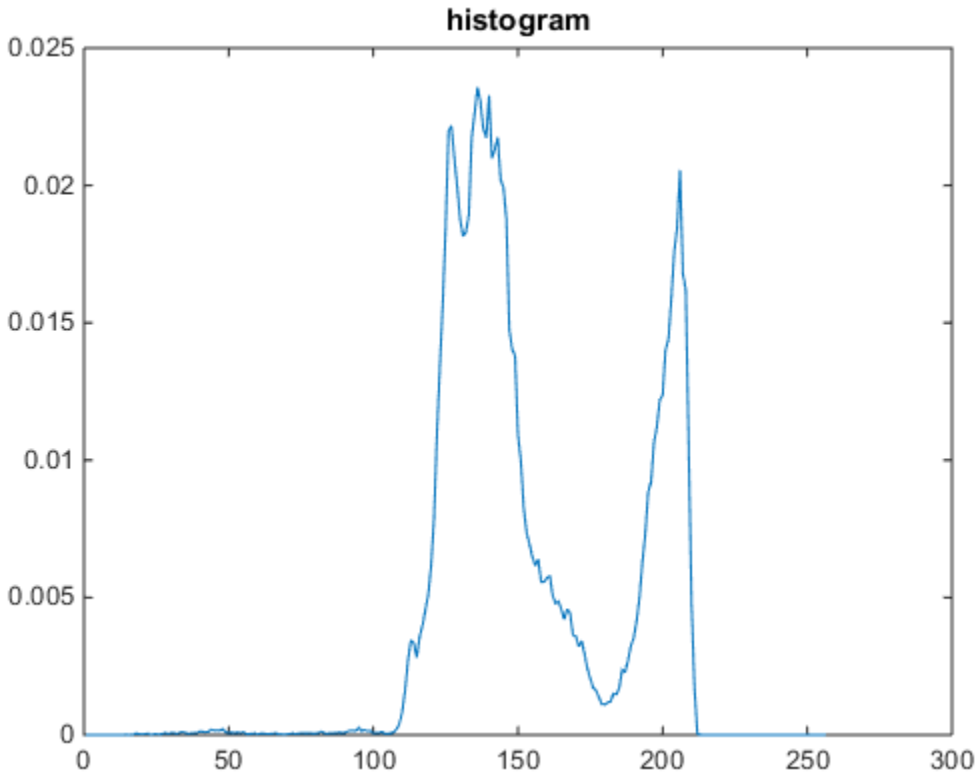
حال این روش را نیز بر روی تصویر مورد نظر اعمال می‌نماییم تا نتایج را مشاهده نماییم.

کد برنامه به صورت زیر می‌باشد:

```
f=imread('lowcontrast3.jpg');
f=rgb2gray(f);
figure,imshow(f),title('original image');
[M,N]=size(f);
%% %%%%%%%%%%structing the histogam function%%%%%%%%%%
h=zeros(1,256);
for m=1:M
for n=1:N
r=f(m,n);
h(r+1)=h(r+1)+1;%h is the histogram function
end
end
hist=h/sum(h);
figure
plot(hist)
title('histogram');
%%%%%%%%%structing the cumulative function%%%%%%%%%5
F=zeros(1,256);
for k=1:256
F(k)=sum(hist(1:k));
end
figure,plot(F),title('cumulative function');
%%%%%%%%%%implementing Histogram equalizeion %%%%%%%%%%55
g=zeros(M,N);
for m=1:M
for n=1:N
s=f(m,n);
```

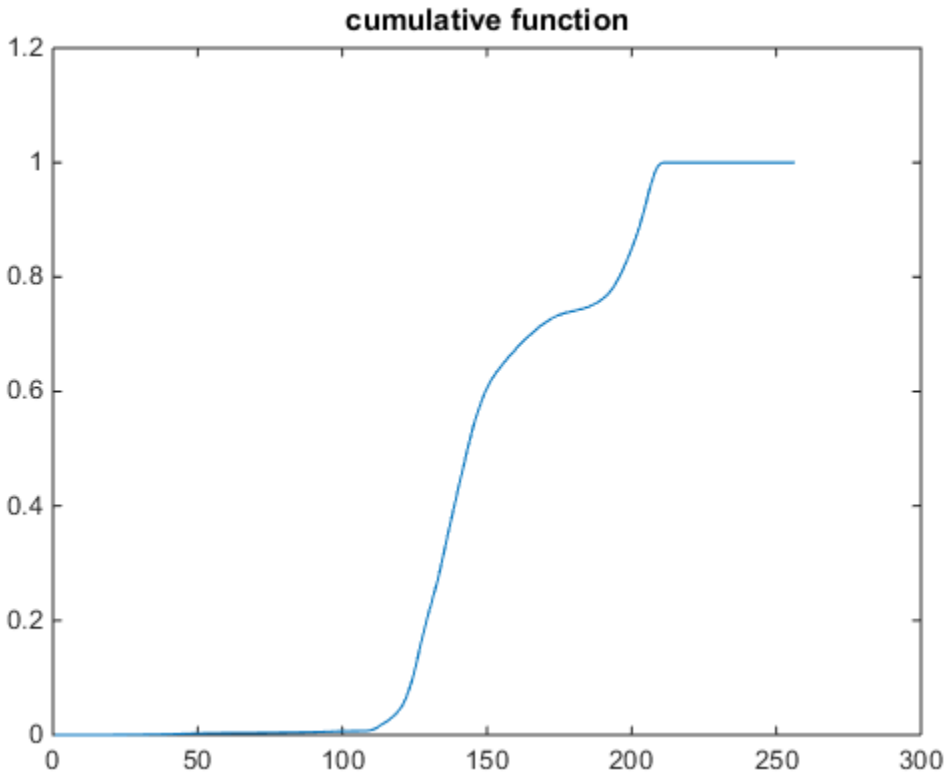
```
g(m,n)=F(s);%g is the resulted image after histogram equalization
end
end
figure,imshow(g),title('equalized image by Global Histogram Equalization');
```

ابتدا هیستوگرام تصویر اصلی را مشاهده می‌نماییم که به صورت زیر می‌باشد:



همان طور که اشاره شد این هیستوگرام دو قله دارد و از نظر چشم انسان مطلوب نیست. لذا به روشی که بیان شد باید هیستوگرام این تصویر را تا حد ممکن یکنواخت نماییم.

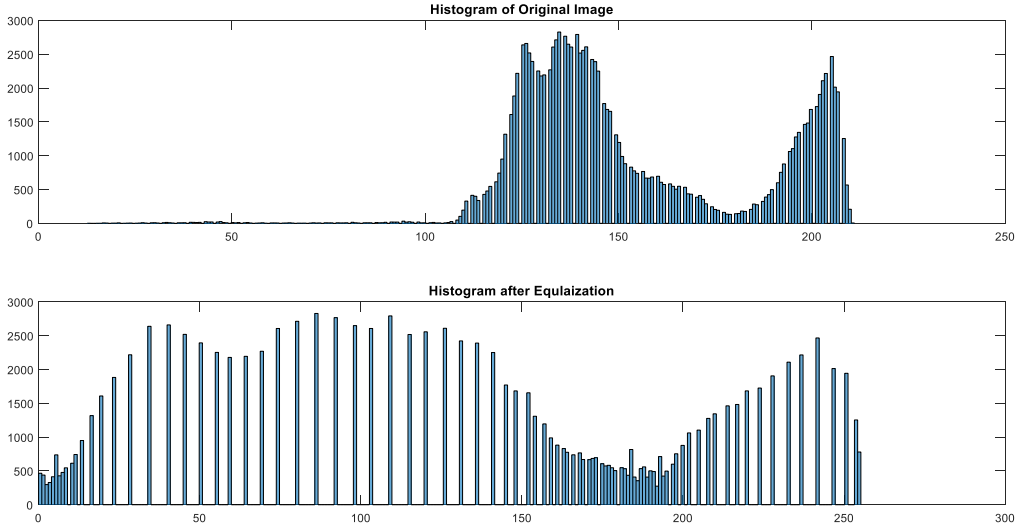
تابع توزیع تجمعی هیستوگرام مورد نظر به صورت زیر می‌باشد:



و تصویر خروجی حاصل از اجرای برنامه به صورت زیر خواهد شد:

مشاهده می‌شود که به نسبت تصویر اصلی نتیجه‌ی ایده‌آل تری را گرفته ایم. و از نظر چشم انسان، این تصویر مطلوب تر می‌باشد. اما همچنان پنجره‌های ساختمان مجاور به‌خوبی قابل‌تشخیص نمی‌باشد.

با مقایسه هیستوگرام بدست آمده از شکل اخیر با شکل اصلی شاهد هستیم این هیستوگرام به نسبت هیستوگرام اصلی، بسیار شباهت بیشتری با یک هیستوگرام یکنواخت دارد و هستوگرام بدست آمده تقریبا مطابق انتظار است. (هر چند کاملا یکنواخت نیست)



در بسیاری از موارد پیش می‌آید که وقتی هیستوگرام یک تصویر را همسان‌سازی می نماییم بازه‌ای از پیکسل‌ها به یک بازه‌ی بسیار کوچک می می‌شوند و این در مواردی که اطلاعات موجود در این بازه برای ما مهم باشد بسیار مشکل‌ساز است به خصوص در تصاویر پزشکی و مثلاً یافتن تومور.

لذا در این موارد روش global مشکل‌ساز بوده چرا که ممکن است موجب از دست رفتن اطلاعات بسیار مهمی همانند تومور در تصاویر پزشکی شود. لذا در این موارد از روش دیگری به نام local استفاده می‌نماییم. در این روش، برای هر پیکسل یک پنجره در نظر گرفته و تابع توزیع تجمعی پنجره را یافته و تکلیف هر پیکسل از خروجی بر مبنای تابع توزیع تجمعی پنجره‌ی گذاشته شده برای هر پیکسل مشخص می‌شود.

یعنی در این حالت مطابق همان توضیحات گفته‌شده در روش global می‌توان گفت:

$$s_i = F_r^{(i)}(r_i)$$

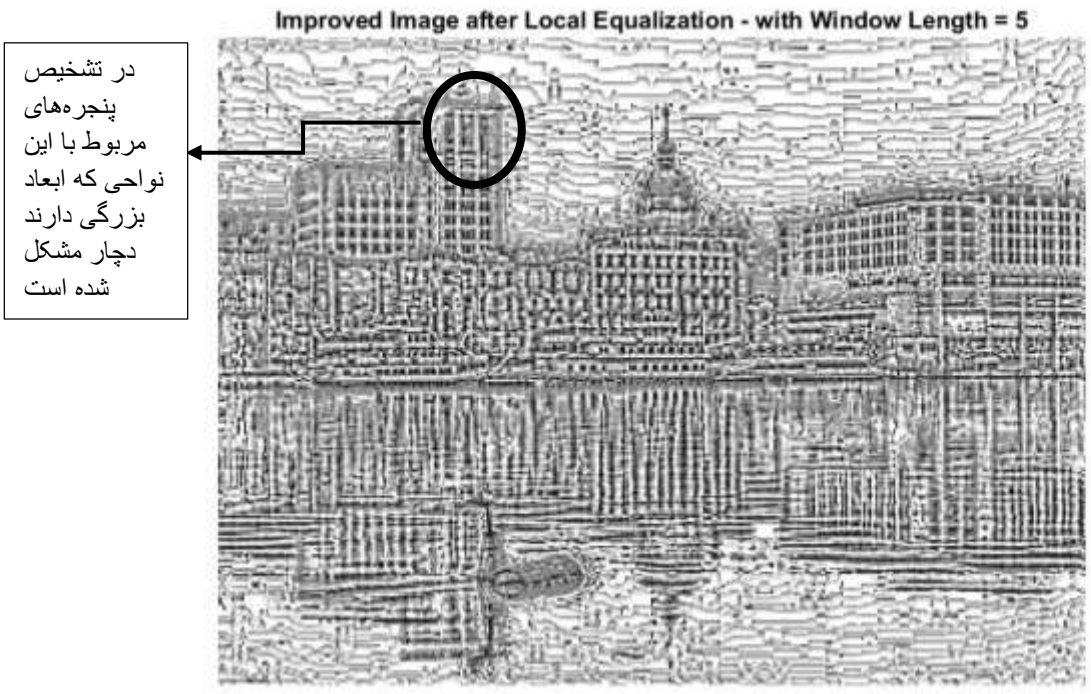
که درواقع s_i برابر با تابع تجمعی پنجره‌ی i ام می‌باشد که روی پیکسل i قرار داده ایم. ابعاد پنجره در این حالت یکی از پارامترهای مهم می‌باشد که به صورت سعی و خطا به دست می آید. حال سعی می‌کنیم این روش را برای تصویر مورد نظر پیاده نماییم تا ببینیم در کدام حالت نتیجه‌ی بهتری حاصل می‌شود.

کد برنامه برای اجرای این روش به صورت زیر می‌باشد:

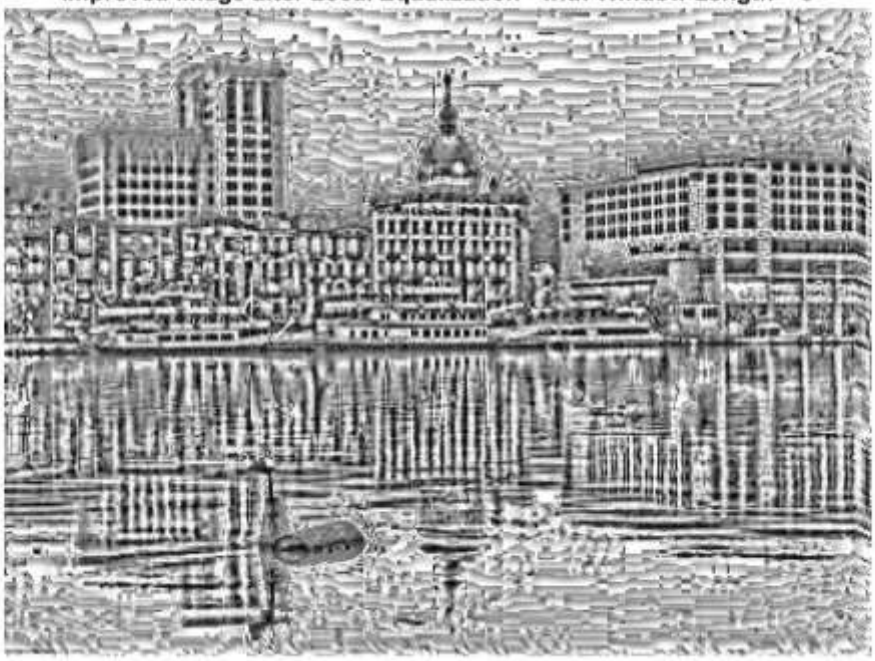
```
%%%%%%local Histogram Equalization%%%%%%%%%%
clc,clear all,close all;
f=imread('lowcontrast3.jpg');
f=rgb2gray(f);
figure,imshow(f),title('original image');
[M,N]=size(f);
w=7;
```

```
fhat=zeros(M+2*fix(w/2),N+2*fix(w/2));
[M1,N1]=size(fhat);
fhat(fix(w/2)+1:M1-fix(w/2),fix(w/2)+1:N1-fix(w/2))=f;
h=zeros(1,256);
%**finding the histogram below the window
for m=fix(w/2)+1:M1-fix(w/2)
    for n=fix(w/2)+1:N1-fix(w/2)

        h=zeros(1,256);
        for i=-fix(w/2):fix(w/2)
            for j=-fix(w/2):fix(w/2)
                r=fhat(m+i,n+j);
                h(r+1)=h(r+1)+1;
            end
        end
        hist=h/sum(h);
%****finding the cumulative function below the window
F=zeros(1,256);
for t=1:256
    F(t)=sum(hist(1:t));
end
%figure,plot(F)
s=fhat(m,n);
g(m,n)=F(s+1);
    end
end
figure,imshow(g,[]),title(['equalized image by local histogram equalization,w= ',num2str(w)]);
```

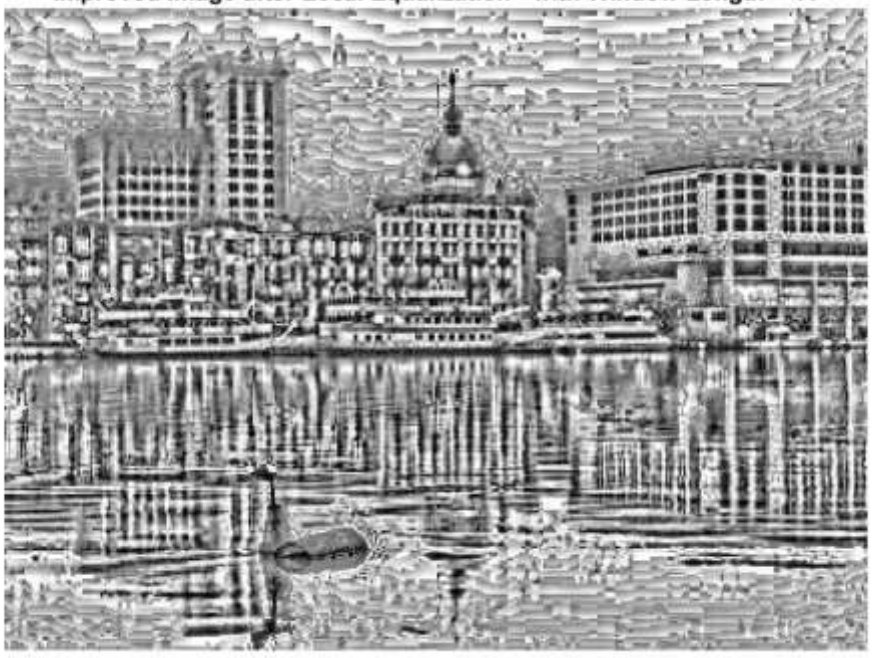
برای پنجره با ابعاد ۷، ۱۱ و ۱۷ نتیجه را تست می‌کنیم:



Improved Image after Local Equalization - with Window Length = 9



Improved Image after Local Equalization - with Window Length = 11



با بررسی شکل‌های فوق مشاهده می‌گردد برای طول پنجره ۹ و ۱۱ جواب ها مناسب می‌باشد و تعدا پنجره‌ها با علت کانتراست محلی ایجاد شده به خوبی قابل شمارش هستند (هر چند که کیفیت عکس تا حد زیادی از بین رفته است) برای طول پنجره کوچک نظیر ۵ در تشخیص پنجره های بزرگ دچار مشکل شده است و برای طول‌های بزرگتر نظیر ۲۵ شاهد بروز مشکل در تشخیص پنجره های کوچک هستیم. (توجه شود با بزرگ‌تر شدن ابعاد پنجره کیفیت تصویر بهبود یافته است که به علت دخالت دادن پیکسل‌های بیشتری در محاسبه توابع احتمال و یکسان سازی تصویر است)

توجه شود که این روش به علت اینکه پیکسل پیکسل باید عملیات Equlaization را انجام دهد بسیار طولانی تر از روش های پیشین است. در مورد نتیجه بدست آمده در روش گاما با گامای۲، شاهد بودیم که در عین سادگی و سرعت بالای اجرا، توانسته بخوبی تعداد پنجره ها را مشخص کند و کیفیت منظره را هم نتنها مانند این روش تخریب نکرده بلکه اندکی شاهد بهبود کیفیت هم هستیم.

سؤال دوم) الف) کد این سؤال فایل HW5_Q2_A.m می‌باشد.

در این قسمت قرار است که از فیلتر gabor جهت اصلاح اثر انگشت استفاده نماییم. معادله‌ی مکانی این فیلتر به‌صورت زیر می باشد:

$$h(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{\frac{-x^2}{2\sigma_x^2}} e^{\frac{-y^2}{2\sigma_y^2}} \cos\frac{2\pi}{T}x$$

این فیلتر برای هر T ی خاص در ۱۶ جهت به تصویر اثر انگشت ما اعمال می‌شود یعنی برای هر T ی خاص، هرکدام از ۱۶فیلتر، یک جهت خاصی از تصویر را فیلتر کرده و بقیه را مبهم و بلور می‌کند و سپس این ۱۶ تصویر فیلتر شده را با هم fuse می‌نماییم تا نتیجه‌ی نهایی را برای هر T به‌دست آوریم. توجه شود که ما در این سؤال، شش T را با اندازه گیری فاصله‌ی دو نقطه از تصویر اصلی در نظر گرفته‌ایم. این Tها را در نقاط مختلف تصویر اندازه گرفتیم و شش T ی مختلف در این تصویر مشاهده شد و برنامه‌ی مورد نظر را براساس این شش T نوشته‌ایم:

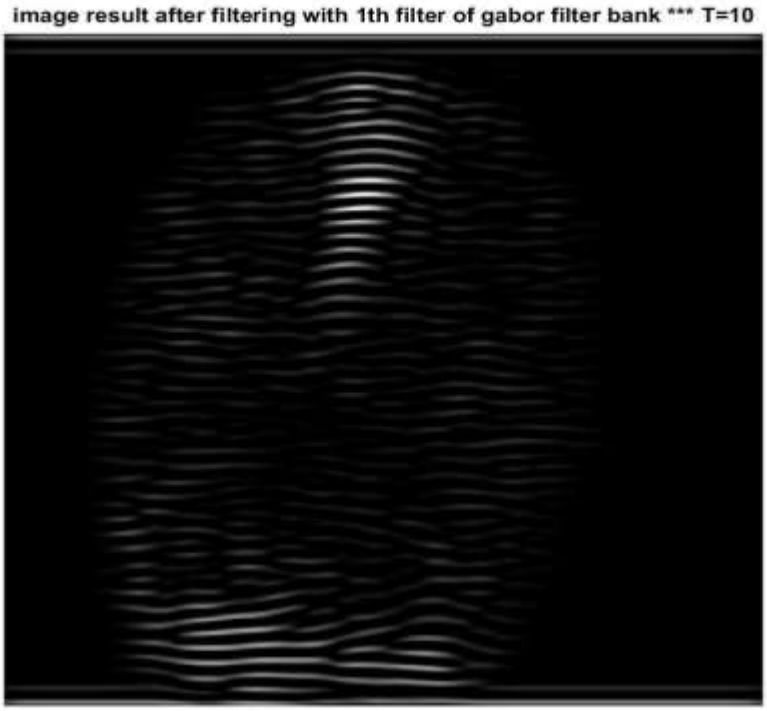
کد برنامه ب صورت زیر می‌باشد:

```
f=imread('finger.tif');
 figure,imshow(f);

%% generate square image
[M , N] = size(f);
temp = [M N];
ind_min = find(temp == min(temp));
switch ind_min
    case 1
        temp1 = N-M;
        ff = f(1:end , 1+temp1/2 : end - temp1/2);
    case 2
        temp1 = M-N;
        ff = f(1+temp1/2 : end - temp1/2 , 1:end);
end
% figure,imshow(ff)


[M , N] = size(ff);

sigma_x=8;
sigma_y=8;
% T=8;

L_X=6*sigma_x+1;
L_Y=6*sigma_y+1;
LX_2 = fix(L_X/2);
LY_2 = fix(L_Y/2);

tt=0;
T_start = 7; T_step=1; T_end=12;
for T=T_start:T_step:T_end
    p=0; tt= tt+1;
    for theta=0:pi/16:pi-0.01
        p = p+1;
        for x = -LX_2 : LX_2
            for y = -LY_2: LY_2
```

```
                x_theta=x*cos(theta)+y*sin(theta);
                y_theta=-x*sin(theta)+y*cos(theta);
                gb_filter(x+LX_2+1 , y+LY_2+1,p) = (2*pi*sigma_x*sigma_y)^-1 .* exp(-
.5*(x_theta.^2/sigma_x^2+y_theta.^2/sigma_y^2)).*cos(2*pi*x_theta/T);
            end
        end
    end
    for k=1:16
        g(:,:,k)=imfilter(ff,gb_filter(:,:,k));
    end

    %%%%% implement pixel method to fuse images

    [M,N] = size(g(:,:,1));
    W = 5;
    w_2 = floor(W/2);
    std_f = zeros(1,16);
    for m= w_2 +1:W : M - w_2 -1
        for n= w_2 +1:W : N - w_2 -1
            for k=1 : 16
                temp = g(m-w_2 : m+w_2 , n-w_2 : n+w_2,k);
                std_f(1,k) = std2(temp);
            end
            ind_max = min(find( std_f == max(std_f)));
            f_fuse(m-w_2 : m+w_2 , n-w_2 : n+w_2,tt) = g(m-w_2 : m+w_2 , n-w_2 : n+w_2,ind_max);
        end
    end

    figure,imshow(f_fuse(:,:,tt),[]), title(['gabor using T =',num2str(T)])

end


%%%%%% Fuse METHOD BLOCK
[M,N] = size(f_fuse(:,:,1));
W = 5;
w_2 = floor(W/2);

for T=T_start:T_step:T_end
    std_f = zeros(1,(T_end-T_start)/T_step);
    for m= w_2 +1:W : M - w_2 -1
        for n= w_2 +1:W : N - w_2 -1
            for tt=1 : T_end-T_start +1
                temp = f_fuse(m-w_2 : m+w_2 , n-w_2 : n+w_2,tt);
                std_f(1,tt) = std2(temp);
            end
            ind_max = min(find( std_f == max(std_f)));
            f_fuse_T_A(m-w_2 : m+w_2 , n-w_2 : n+w_2) = f_fuse(m-w_2 : m+w_2 , n-w_2 :
n+w_2,ind_max);
        end
    end
end

figure, imshow(f_fuse_T_A,[]),title('final Image fuse with BLOCK method')

%%%%%% Fuse MEthod PIXEL

for T=T_start:T_step:T_end
    std_f = zeros(1,(T_end-T_start)/T_step);
    for m= w_2 +1 : M - w_2 -1
        for n= w_2 +1 : N - w_2 -1
            for tt=1 : T_end-T_start +1
                temp = f_fuse(m-w_2 : m+w_2 , n-w_2 : n+w_2,tt);
                std_f(1,tt) = std2(temp);
            end
            ind_max = min(find( std_f == max(std_f)));
            f_fuse_T_B(m , n) = f_fuse(m , n ,ind_max);
        end
    end
end
figure, imshow(f_fuse_T_B,[]),title('final Image fuse with PIXEL method')
```

برای جلوگیری از طولانی شدن گزارش، به ازای یکی از T ها مثلاً T=10، چند جهت از این ۱۶ جهت را رسم می‌نماییم و برای سایر T ها نتیجه ی حاصل پس از fuse شدن این ۱۶ جهت را رسم می‌نماییم. توجه شود که در کد مورد نظر، مقادیر سیگماها در جهت x و y را با سعی و خطا و با انتخاب مقادیر مختلف برای سیگماها و مشاهده‌ی نتایج مختلف، برابر با ۸ در نظر گرفته‌ایم.

برای T=10 و برای نمونه برای ۴ جهت داریم:



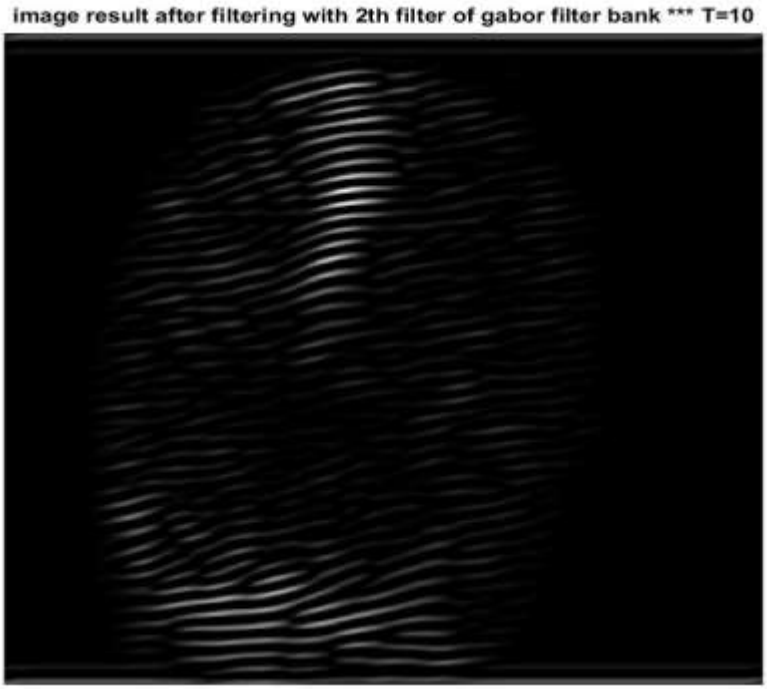


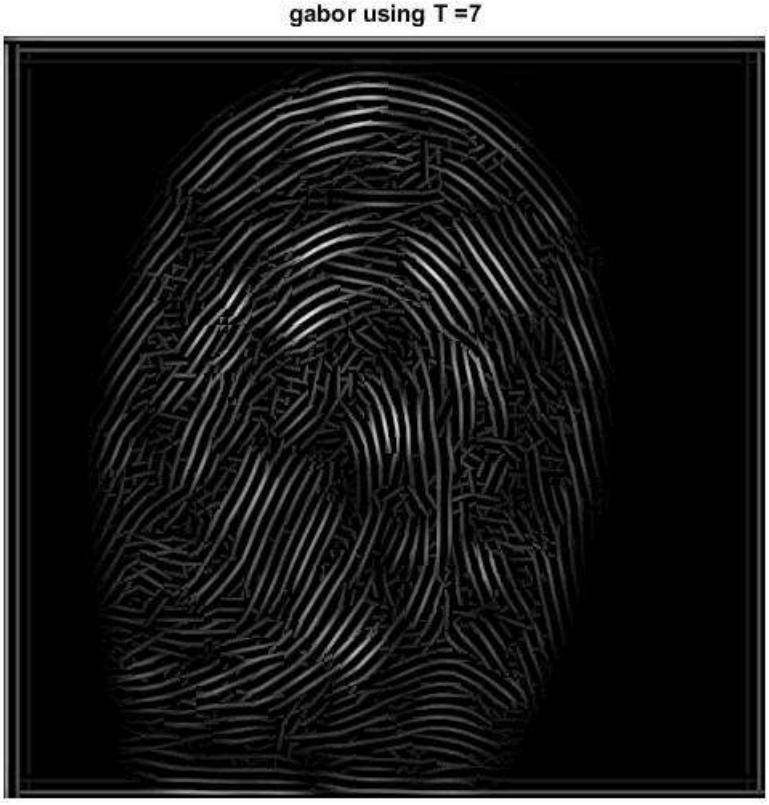
image result after filtering with 3th filter of gabor filter bank *** T=10

image result after filtering with 4th filter of gabor filter bank *** T=10

و برای ۱۲ جهت دیگر نیز نتایجی به همین صورت به دست می‌آیند که از آوردن آن‌ها خودداری می‌نماییم.

حال در ادامه، این ۱۶ تصویر به‌دست آمده برای هر T را به دو روش پیکسلی و بلوکی فیوز می‌نماییم. در روش بلوکی، برای هر تصویر یک پنجره در نظر می‌گیریم و مقدار سیگمای آن پنجره را محاسبه می‌نماییم و سپس کل پیکسل‌های پنجره با سیگمای بزرگ‌تر را به پنجره‌ای با همان ابعاد در خروجی منتقل می‌نماییم و سپس به اندازه‌ی ابعاد پنجره، بلوک را شیفت می‌دهیم. اما در روش پیکسلی، پنجره، پیکسل به پیکسل جابجا می‌شود و سپس سیگمای هر پنجره حساب شده و پیکسل مربوط به پنجره با سیگمای بزرگ‌تر به پیکسل خروجی منتقل می‌شود. انتظار داریم که روش پیکسلی به دلیل دقت بالاتری که دارد، نتیجه‌ی نسبتاً بهتری را ارائه دهد. لازم به ذکر است که برای فیوز کردن این ۱۶ تصویر فیلتر شده و مشاهده‌ی نتایجی که در ادامه رسم می‌نماییم، به دلیل این‌که تعداد پیکسل‌ها و تصاویر بسیار زیاد بود و حجم محاسبات به شدت بالاتر می‌رفت از روش بلوکی استفاده نمودیم.

نتیجه ی حاصل پس از فیوز کردن این ۱۶ جهت برای هرکدام از این T ها بدین شکل می‌باشد:

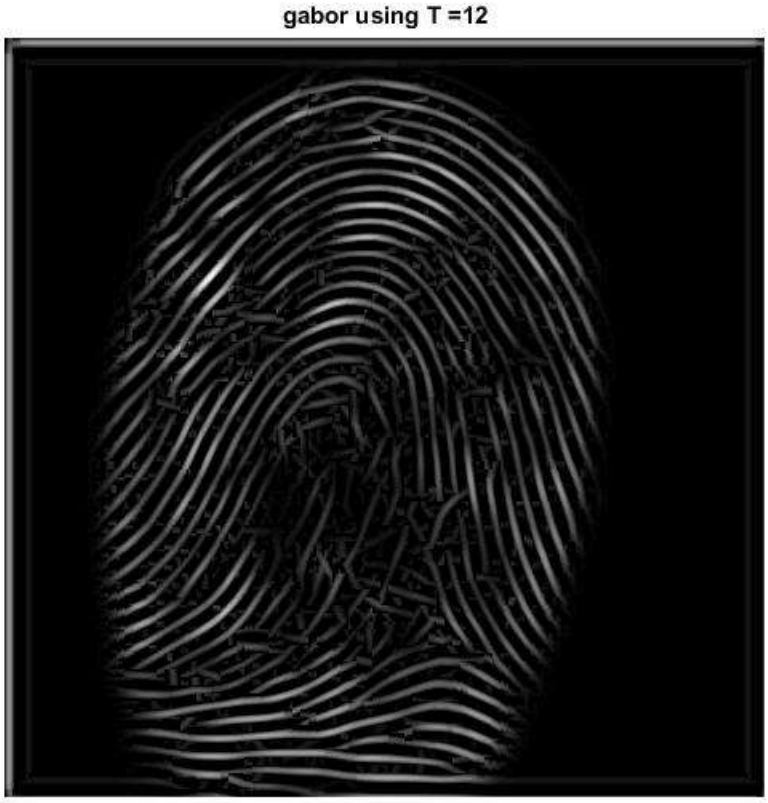


gabor using T =8

gabor using T =9

gabor using T =10

gabor using T =11



حال در ادامه، این ۶ تصویر به‌دست آمده را نیز به هر دو روش پیکسلی و بلوکی که در بالا به آن‌ها اشاره شد فیوز می‌نماییم. نتیجه‌ی حاصل از فیوز کردن این ۶ تصویر با این دو روش در زیر آمده‌است:

final Image fuse with BLOCK method

final Image fuse with PIXEL method

مشاهده می‌شود همان‌طور که انتظار داشتیم، در یک سری از مکان‌ها در روش بلوکی بریدگی‌هایی وجود دارد که در روش پیکسلی در آن مکان‌ها پیوستگی وجود دارد. البته تفاوت‌ها به نسبت محاسبات بیشتر روش پیکسلی خیلی زیاد نیست ولی روش پیکسلی مقدار کمی دقت بالاتری در پیوستگی خطوط اثر انگشت دارد

ب) کد این سؤال فایل HW5_Q2_B.m می‌باشد.

در این قسمت سؤال، قرار است که از فیلتر دیگری به نام quinqunx در جهت اصلاح تصویر اثر انگشت استفاده نماییم. این فیلترها هم فرکانس صفر و هم بالاترین فرکانس تصویر را شامل می‌شوند چرا که شکلی به صورت زیر دارند که همان‌طور که مشاهده می شود شامل تمامی فرکانس‌های تصویر می‌شوند.

حال در این قسمت سؤال قرار است در ابتدا ۸ تا از این فیلتر ها را تولید نماییم. سپس هرکدام از این ۸ فیلتر، تصویر اثر انگشت را در یک جهت فیلتر می‌نماید. لازم به ذکر است که تمامی این فیلترها در حوزه‌ی فرکانس می‌باشند لذا لازم است تصویر را به حوزه فرکانس برده و فیلتر نماییم و در انتها عکس تبدیل فوریه بگیریم تا مجدداً به حوزه مکان بازگردیم.

حال باید در این مرحله این ۸ فیلتر را تولید نماییم:

کد مورد نظر برای تولید این ۸ فیلتر بدین‌صورت می‌باشد:

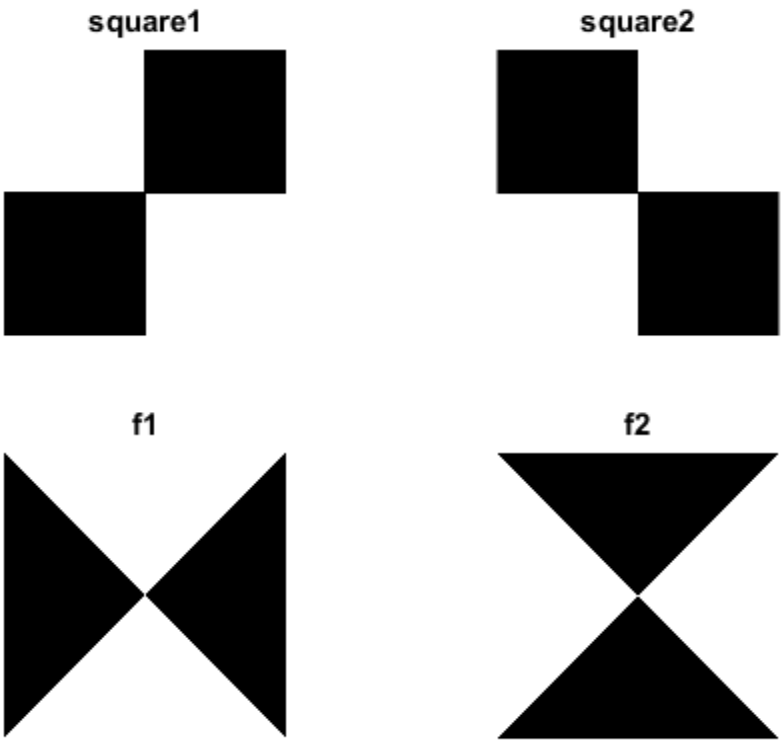
```
g=imread('finger.tif');
%%%% convert image 2 square image
[M , N] = size(g);
temp = [M N];
ind_min = find(temp == min(temp));
switch ind_min
    case 1
        temp1 = N-M;
        g1 = g(1:end , 1+temp1/2 : end - temp1/2);
    case 2
        temp1 = M-N;
        g1 = g(1+temp1/2 : end - temp1/2 , 1:end);
end

g1=im2double(g1);
figure
imshow(g1,[])
title('Orginal Image (square size)')

%% Generate QUINCUNX 8 filter
w=size(g1,1);
s=w/2;
h1=zeros(s);

figure
square1=zeros(w);
```

```
square1(1:s-1,1:s-1)=ones(s-1);
square1(s+2:w,s+2:w)=ones(s-1);subplot(221),imshow(square1,[]),title('square1')
square2=zeros(w);
square2(1:s-1,s+2:w)=ones(s-1);
square2(s+2:w,1:s-1)=ones(s-1);subplot(222),imshow(square2,[]),title('square2')
h2=h1;
for m=1:s
    for n=1:s
        if m<n
            h1(m,n)=1;

        end
        if m>s-n
            h2(m,n)=1;
        end
    end
end

f1=[h1,1-h2;h2,1-h1];  subplot(223),imshow(f1,[]),title('f1')
f2=[1-h1,h2;1-h2,h1];  subplot(224),imshow(f2,[]),title('f2')


h1=zeros(w);
h2=h1;
for m=1:w
    for n=1:w
        if m<n
            h1(m,n)=1;

        end
        if m>w-n
            h2(m,n)=1;
        end
    end
end

rect1=[h1,1-h2;h2,1-h1];
rect2=[1-h1,h2;1-h2,h1];

for m=-s:s-1
    for n=-s:s-1
        y11(m+s+1,n+s+1)=rect1(m+n+w+1,n+w+1);
        y12(m+s+1,n+s+1)=rect1(m-n+w+1,n+w+1);
        y13(m+s+1,n+s+1)=rect1(m+w+1,n+m+w+1);
        y14(m+s+1,n+s+1)=rect1(m+w+1,n-m+w+1);
    end
end
figure,subplot(221),imshow(y11,[]),title('y11')
subplot(222),imshow(y12,[]),title('y12')
subplot(223),imshow(y13,[]),title('y13')
subplot(224),imshow(y14,[]),title('y14')

for m=-s:s-1
    for n=-s:s-1
        y21(m+s+1,n+s+1)=rect2(m+n+w+1,n+  w+1);
        y22(m+s+1,n+s+1)=rect2(m-n+w+1,n+  w+1);
        y23(m+s+1,n+s+1)=rect2(m+  w+1,n+m+w+1);
        y24(m+s+1,n+s+1)=rect2(m+  w+1,n-m+w+1);
    end
end

figure,subplot(221),imshow(y21,[]),title('y21')
subplot(222),imshow(y22,[]),title('y22')
subplot(223),imshow(y23,[]),title('y23')
subplot(224),imshow(y24,[]),title('y24')

figure,
z1=f1.*square1;  subplot(221),imshow(z1,[]),title('z1')
z2=f1.*square2;  subplot(222),imshow(z2,[]),title('z2')
z3=f2.*square1;  subplot(223),imshow(z3,[]),title('z3')
z4=f2.*square2;  subplot(224),imshow(z4,[]),title('z4')
```

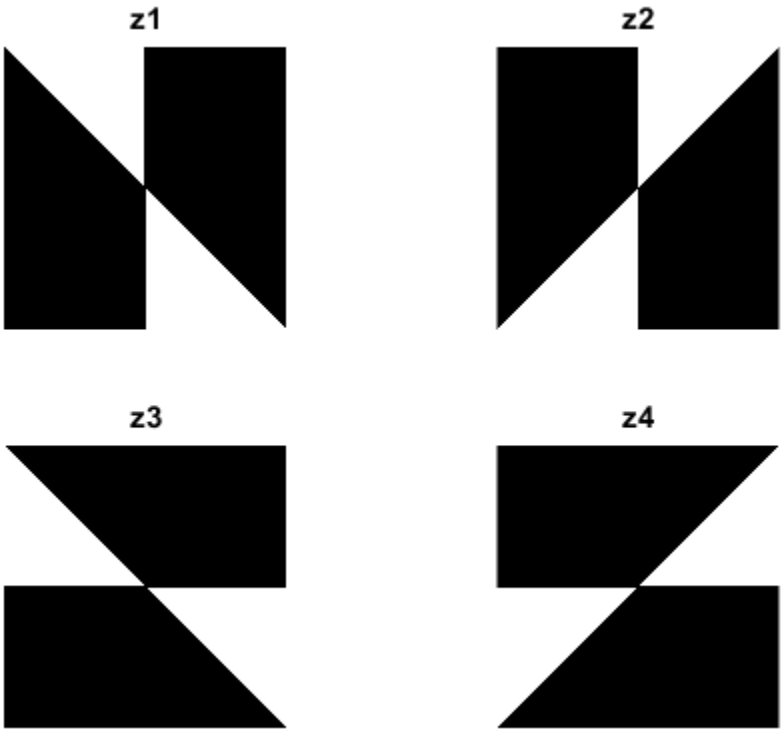
```
figure
Q_8(:,:,1)=y11.*square2;    subplot(421),imshow(Q_8(:,:,1),[]),title('q1')
Q_8(:,:,2)=y12.*square1;    subplot(422),imshow(Q_8(:,:,2),[]),title('q2')
Q_8(:,:,3)=y13.*z4;         subplot(423),imshow(Q_8(:,:,3),[]),title('q3')
Q_8(:,:,4)=y14.*z3;         subplot(424),imshow(Q_8(:,:,4),[]),title('q4')
Q_8(:,:,5)=y21.*z2;         subplot(425),imshow(Q_8(:,:,5),[]),title('q5')
Q_8(:,:,6)=y22.*z1;         subplot(426),imshow(Q_8(:,:,6),[]),title('q6')
Q_8(:,:,7)=y23.*square2;    subplot(427),imshow(Q_8(:,:,7),[]),title('q7')
Q_8(:,:,8)=y24.*square1;    subplot(428),imshow(Q_8(:,:,8),[]),title('q8')
suptitle('8 quincunx filter');


%%% this part causes the frequncy response be symmetric around 0
for m=0:w-1
    for n=0:w-1
        f(m+1,n+1)=(-1)^(m+n)*g1(m+1,n+1);
    end
end
F=fft2(f);
for k=1:8
    Fhat(:,:,k)=Q_8(:,:,k).*F;
end
[M , N] = size(Q_8(:,:,1));
```

برای تولید این ۸ فیلتر مطابق کد بلا بدین‌صورت عمل نموده‌ایم:

ابتدا تصاویر زیر را تولید می‌نماییم:



با توجه به تصاویر بالا و کد نوشته‌شده، از ترکیب f1 با square1، z1 و با square2، z2 حاصل می‌شود. همچنین از ترکیب f2 با square1، z3 و با square2، z4 حاصل می‌شود که نتیجه‌ی آن به صورت زیر حاصل می‌شود:

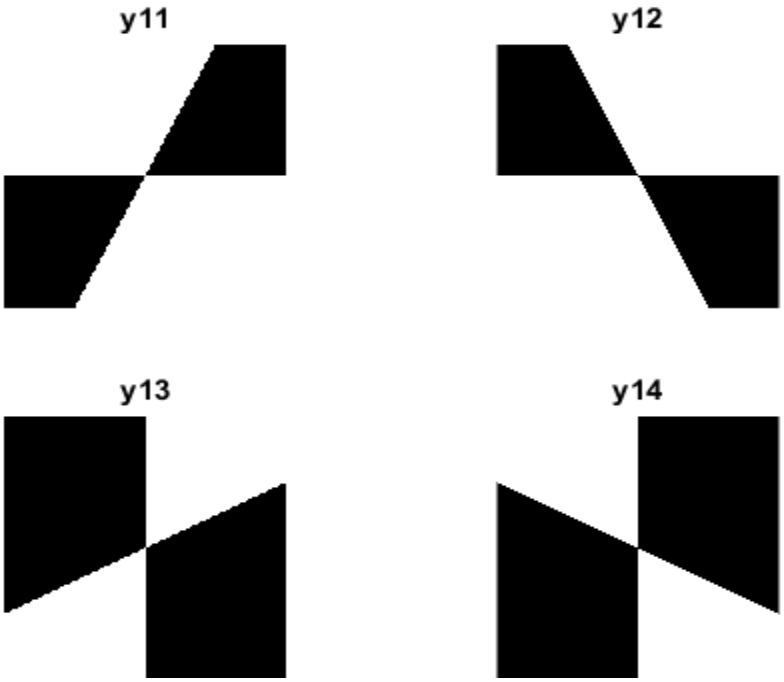


سپس با نمونه‌برداری از f1 با ماتریس‌های چهارگانه‌ی R0 تا R3 که در زیر آمده است، y11,y12,y13,y14 ساخته می‌شوند که با توجه به کد نوشته‌شده در زیر رسم می‌شوند:

$$\boldsymbol{R0} = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \qquad \boldsymbol{R1} = \begin{bmatrix} \mathbf{1} & \mathbf{-1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$$

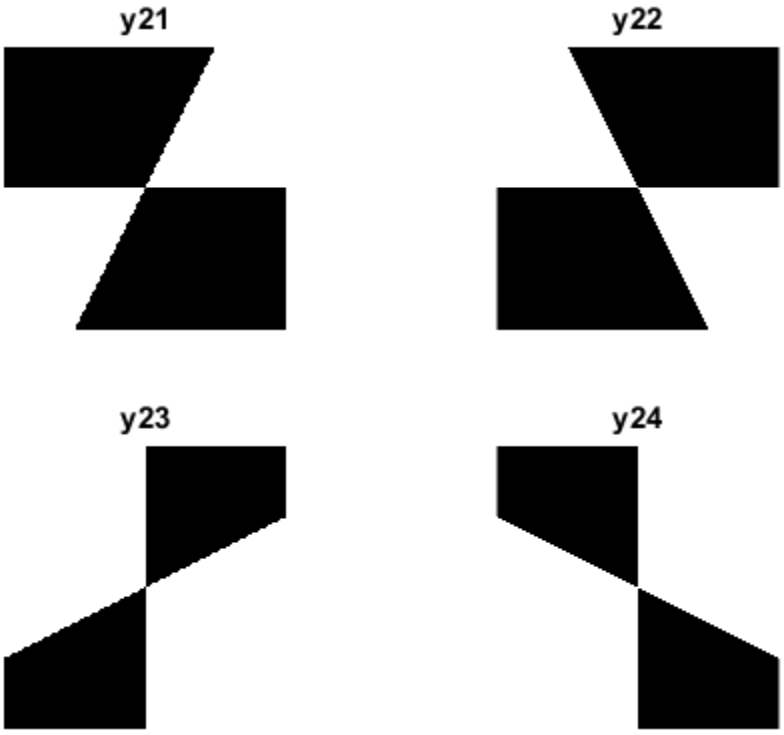
$$\boldsymbol{R2} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} \end{bmatrix} \qquad \boldsymbol{R3} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{-1} & \mathbf{1} \end{bmatrix}$$

که نتایج حاصل از اجرای برنامه بدین‌صورت است:

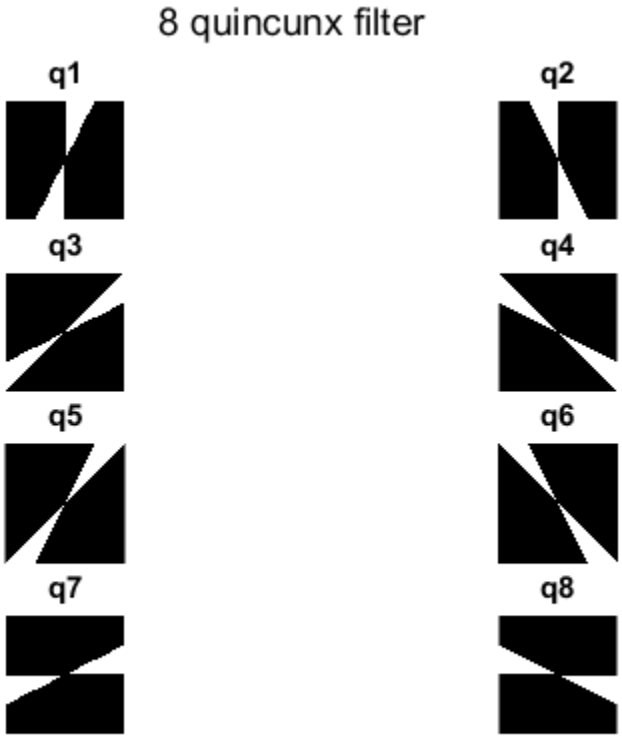


با در دست داشتن این‌ها و square1 وsquare2 وz3 وz4 چهارتا از هشت فیلتر ساخته می شود.

عیناً همین کار با نمونه‌برداری چهارگانه ازf2 انجام می شود تا y21,y22,y23,y24 ساخته شود و سپس این‌ها نیز با square1 وsquare2 وz3 وz4 ترکیب می‌شوند و چهارتای دیگر ساخته می شود:



و در انتها ۸ فیلتر به صورت زیر به دست می‌آید:



حال باید این فیلترها را توسط کد زیر به تصویر اعمال کنیم تا در ۸ جهت تصویر را فیلتر نماید و در انتها آن‌ها را فیوز نماییم:

```
% %% imfiter
for m=0:w-1
    for n=0:w-1
        f(m+1,n+1)=(-1)^(m+n)*g1(m+1,n+1);
    end
end
F=fft2(f);
for k=1:8
    Fhat(:,:,k)=Q(:,:,k).*F;
end
[M , N] = size(q8);

for k=1:8
    fhat=ifft2(Fhat(:,:,k));
    fhat=real(fhat);
    for m=0:w-1
        for n=0:w-1
            final_f(m+1,n+1)=(-1)^(m+n)*fhat(m+1,n+1);
        end
    end
    final_ff(:,:,k) = final_f;
    figure,imshow(final_f,[]);
end
```

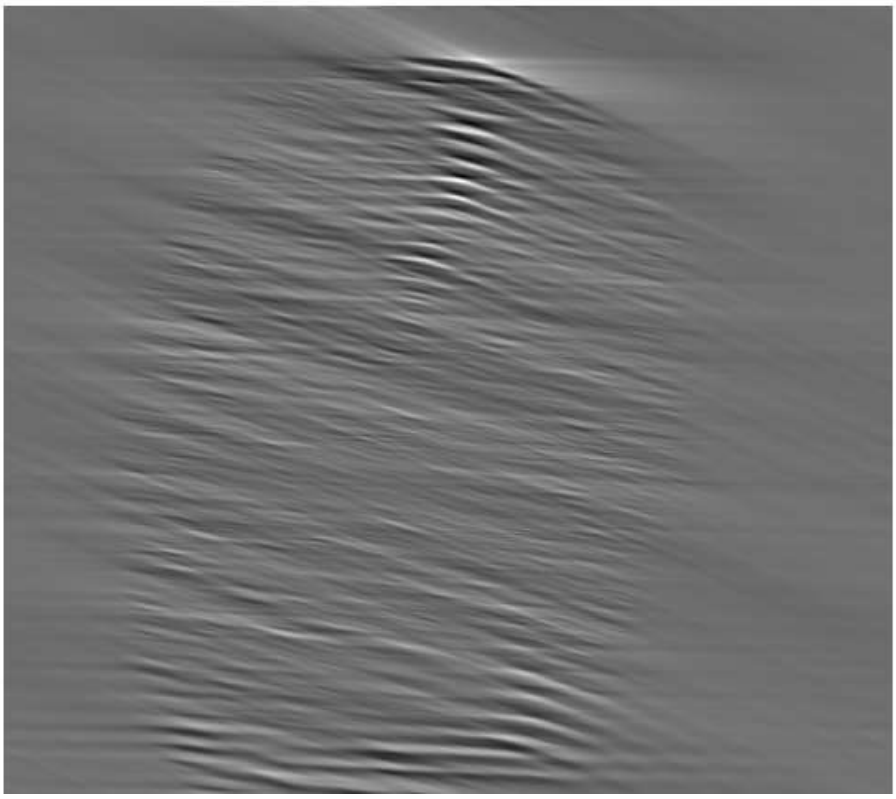
برای جلوگیری از طولانی شدن گزارش تنها دو جهت فیلتر شده را نمایش می‌دهیم:

تصویر اصلی در حوزه مکان به صورت زیر می‌باشد:

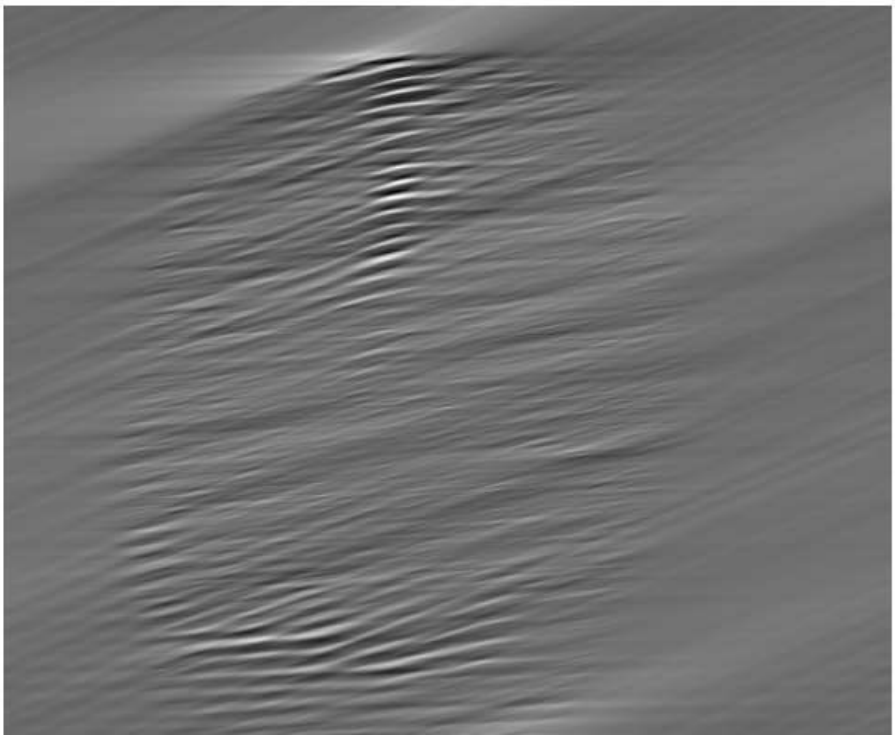
Orginal Image (square size)

و دو جهت از ۸ جهت ذکرشده بدین‌صورت خواهد بود:

1th part of Image after using QUINQUNX (8 Filter)



2th part of Image after using QUINQUNX (8 Filter)



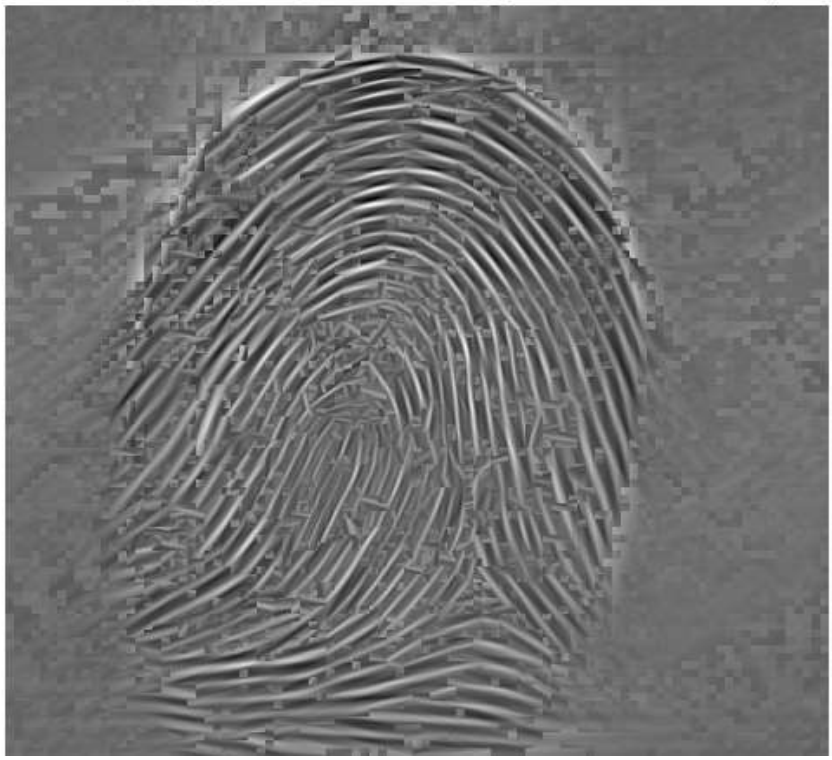
۶ جهت دیگر نیز به طرز مشابه به دست خواهند آمد.

حال باید این ۸ تصویر به‌دست‌آمده را با هم فیوز نماییم برای این منظور از هردو روش بلوکی و پیکسلی که قبلاً به آن‌ها اشاره شد استفاده نموده‌ایم. کد مورد نر بدین‌صورت می‌باشد:

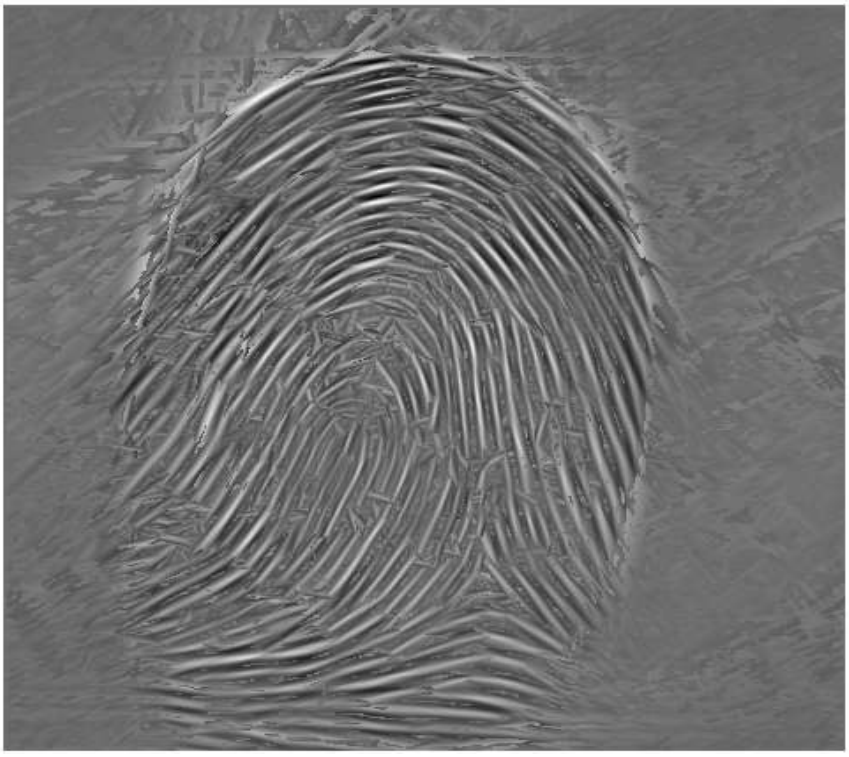
```
%%%%%% Fusing Image with using BLOCK Method
W = 5;
w_2 = floor(W/2);

for k=1 : 8
    std_f = zeros(1, 8);
    for m= w_2 +1 : W : M - w_2 -1
        for n= w_2 +1 : W : N - w_2 -1
            for k=1 : 8
                temp = final_ff(m-w_2 : m+w_2 , n-w_2 : n+w_2,k);
                std_f(1,k) = std2(temp);
            end
            ind_max = min(find( std_f == max(std_f)));
            f_fuse_block_8(m-w_2 : m+w_2 , n-w_2 : n+w_2) = final_ff(m-w_2 : m+w_2 , n-w_2 :
n+w_2 ,ind_max);
        end
    end
end

figure,imshow(f_fuse_block_8,[]),title('Fusing Image Using Block Method-(QUINCUNX 8 FILTER)')


%%%%%% Fusing Image with using PIXEL Method
W = 5;
w_2 = floor(W/2);

for k=1 : 8
    std_f = zeros(1, 8);
    for m= w_2 +1 : M - w_2 -1
        for n= w_2 +1 : N - w_2 -1
            for k=1 : 8
                temp = final_ff(m-w_2 : m+w_2 , n-w_2 : n+w_2,k);
                std_f(1,k) = std2(temp);
            end
            ind_max = min(find( std_f == max(std_f)));
            f_fuse_pix_8(m , n) = final_ff(m , n ,ind_max);
        end
    end
end

figure,imshow(f_fuse_pix_8,[]),title('Fusing Image Using Pixel Method-(QUINCUNX 8 FILTER)')
```

و نتیجه بدین‌صورت می‌باشد:

Fusing Image Using Block Method-(QUINCUNX 8 FILTER)



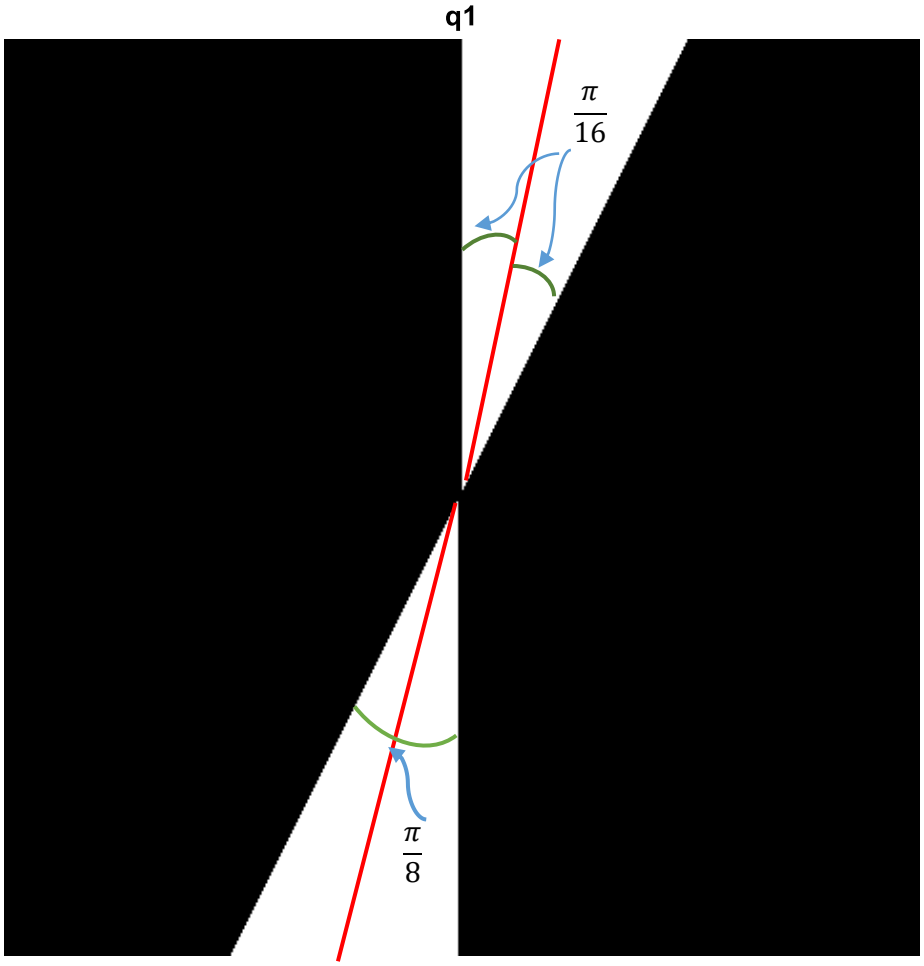
Fusing Image Using Pixel Method-(QUINCUNX 8 FILTER)



سؤال ۲) اختیاری

در این قسمت از ما خواسته شده است که از فیلترهای ۱۶ گانه Quiccunx برای فیلتر کردن پاسخ فرکانسی استفاده کنیم. توجه شود که برای تولید فیلترهای Quincunx روش‌های بسیار متنوعی وجود دارد و نحوه تولید این فیلتر ها اصلا اهمیتی ندارد تنها مسئله حائز اهمیت برای ما این است که فیلترهای تولید شده هرکدام زاویه $\frac{۲\pi}{۱۶}$ را نسبت به مرکز مستطیل را پوشش دهد و همچنین این فیلتر ها دارای ابعاد برابر با ماتریس تبدیل فوریه باشند. ما برای تولید این فیلتر ها از فیلترهای ۸ گانه تولید شده در قسمت قبل استفاده کردیم.

برای مثال به شکل مربوط به فیلتر q1 از فیلترهای ۸ گانه توجه فرمایید :

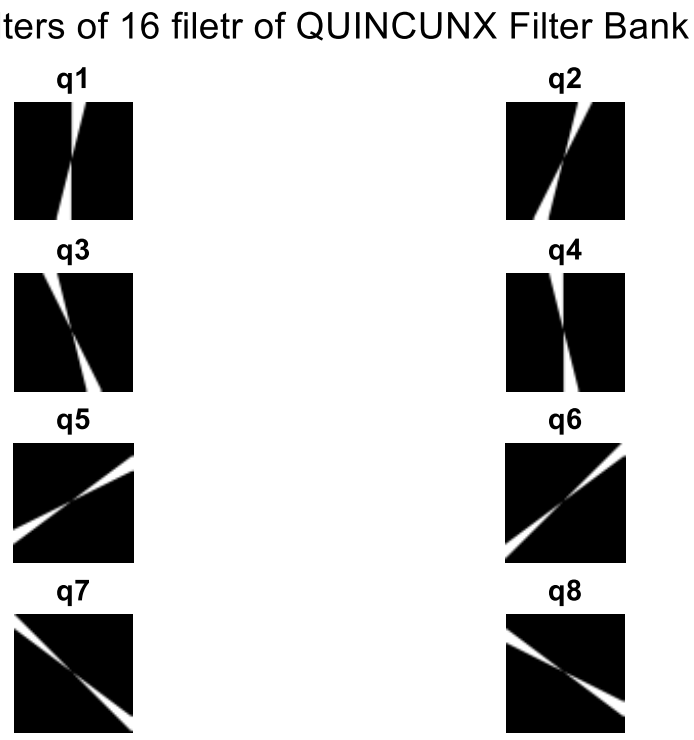


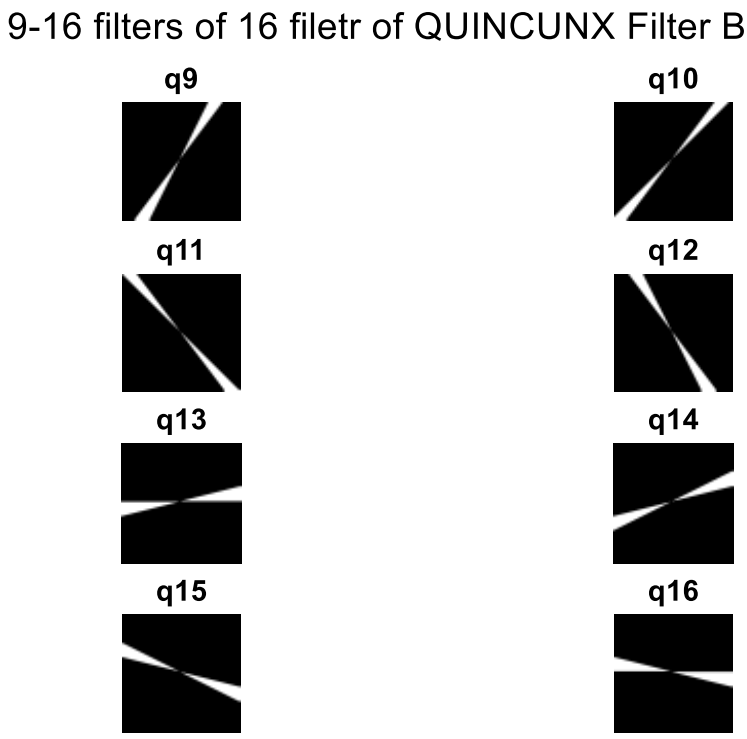
در صورتی که نیمساز این فیلتر را رسم کنیم و سپس این فیلتر را از روی خط نیمساز به دو فیلتر تبدیل کنیم و این کار را برای تمام فیلترهای ۸ گانه انجام دهیم ما قادر خواهیم بود که از این ۸ فیلتر که در دست داریم ۱۶ فیلتر تولید کنیم به طوری که ۱۶ فیلتر تولید شده قادر باشند تمام محتوای فرکانسی ما رو پوشش دهند.

در زیر دو فیلتر بدست آمده از فیلتر q1 شاهد هستیم:

2 filters that obtained from q1(8 filter)

با انجام این کار تمام ۱۶ فیلتر را بدست می‌آوریم که تصاویر مربوط به ۱۶ فیلتر بدست آمده در زیر مشخص هستند





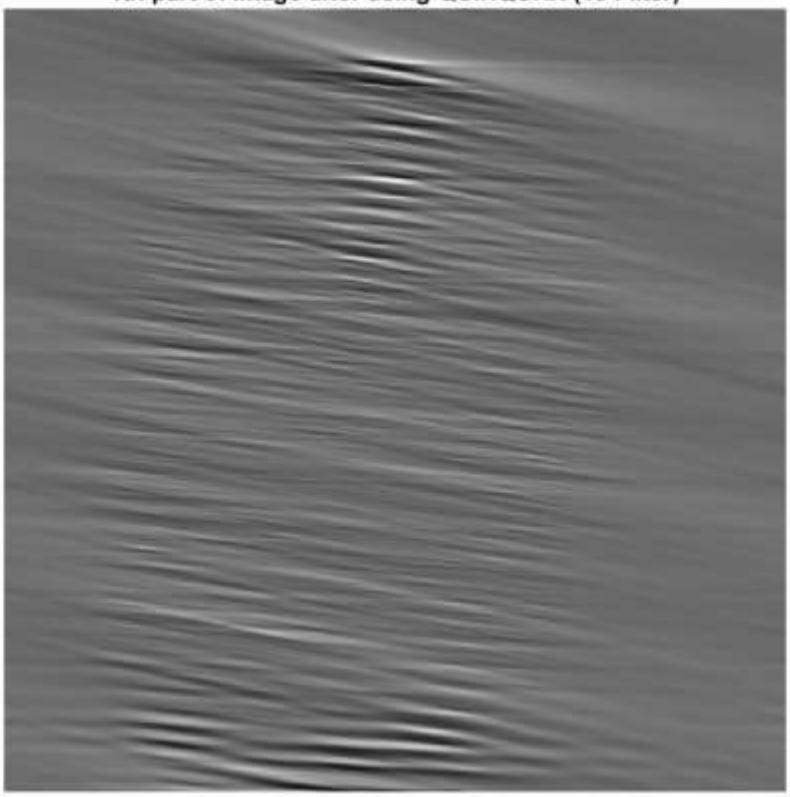
کد مربوط به تولید این فیلترها با استفاده از فیلترهای ۸ گانه را در زیر شاهد هستیم:

```
%% %%%%%%%%%%%%%%%%%%%%%%%% generate QUINCUNX 8 filter
[M,N] = size(Q_8(:,:,1));
p=0;
for i=1:2:16
    p=p+1;
    temp_m = Q_8(1,:,p);
    ind_first_1_m = min(find(temp_m==1));
    temp_n = Q_8(:,1,p);
    ind_first_1_n = min(find(temp_n==1));

    if  isempty(ind_first_1_m)==1
        ind_first_1_m = 480;
    elseif isempty(ind_first_1_n)==1
        ind_first_1_n = 480;
    end

    if ind_first_1_m < ind_first_1_n
        type_m_n = 'm';
    else
        type_m_n = 'n';
    end

    switch type_m_n

        case 'm'

            for m=1:M/2
                temp = Q_8(m,:,p);
                ind_first_1 = min(find(temp==1));
                ind_last_1 = max(find(temp==1));
                ind_med = floor(.5*(ind_first_1+ind_last_1));
                Q_16(m,ind_first_1:ind_med,i) = 1;
                Q_16(m,ind_med+1:ind_last_1,i+1) = 1;
```

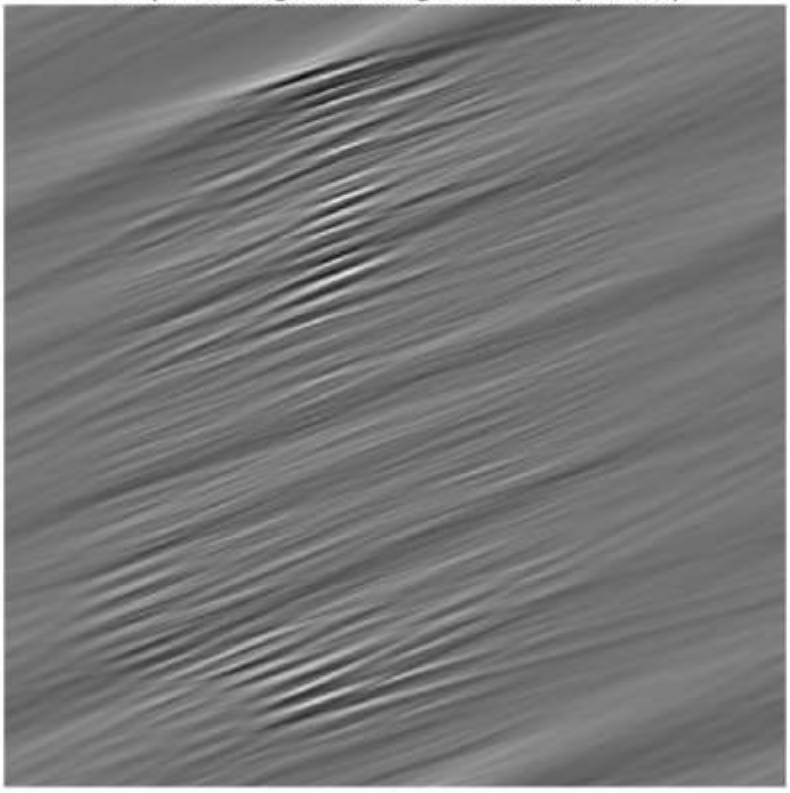
```
            end

            for m=M/2+1:M
                temp = Q_8(m,:,p);
                ind_first_1 = min(find(temp==1));
                ind_last_1 = max(find(temp==1));
                ind_med = floor(.5*(ind_first_1+ind_last_1));
                Q_16(m,ind_first_1:ind_med,i+1) = 1;
                Q_16(m,ind_med+1:ind_last_1,i) = 1;
            end

        case 'n'
            for n=1:N/2
                temp = Q_8(:,n,p);
                ind_first_1 = min(find(temp==1));
                ind_last_1 = max(find(temp==1));
                ind_med = floor(.5*(ind_first_1+ind_last_1));
                Q_16(ind_first_1:ind_med,n,i) = 1;
                Q_16(ind_med+1:ind_last_1,n,i+1) = 1;
            end

            for n=N/2+1:N
                temp = Q_8(:,n,p);
                ind_first_1 = min(find(temp==1));
                ind_last_1 = max(find(temp==1));
                ind_med = floor(.5*(ind_first_1+ind_last_1));
                Q_16(ind_first_1:ind_med,n,i+1) = 1;
                Q_16(ind_med+1:ind_last_1,n,i) = 1;
            end
    end
end

%%%%%% Plot 16 QUINCUNX Filter
figure;
for i=1:2:8
    subplot(4,2,i),imshow(Q_16(:,:,i)),title(['q',num2str(i)]);
    subplot(4,2,i+1),imshow(Q_16(:,:,i+1)),title(['q',num2str(i+1)]);
end
suptitle('1-8 filters of 16 filetr of QUINCUNX Filter Bank');
figure;
for i=9:2:16
    subplot(4,2,i-8),imshow(Q_16(:,:,i)),title(['q',num2str(i)]);
    subplot(4,2,i-8+1),imshow(Q_16(:,:,i+1)),title(['q',num2str(i+1)]);
end
suptitle('9-16 filters of 16 filetr of QUINCUNX Filter Bank');
```

حال باید تبدیل فوریه تصویر را (تصویری که تبدیل فوریه آن حول ۰ متقارن شده است) در هر یک از این ۱۶ فیلتر ضرب کنیم هر یک از این ۱۶ فیلتر در یک جهت خاص محتوای فرکانسی را استخراج می‌کنند. درزیر تصاویر بدست آمده برای ۶ فیلتر از ۱۶ فیلتر را شاهد هستیم (با توجه با تشابه جواب‌ها با یکدیگر از آوردن ۱۶ حالت خودداری می‌کنیم در کد می‌توان تمام ۱۶ حالت را مشاهده کرد)

1th part of Image after using QUINQUNX (16 Filter)

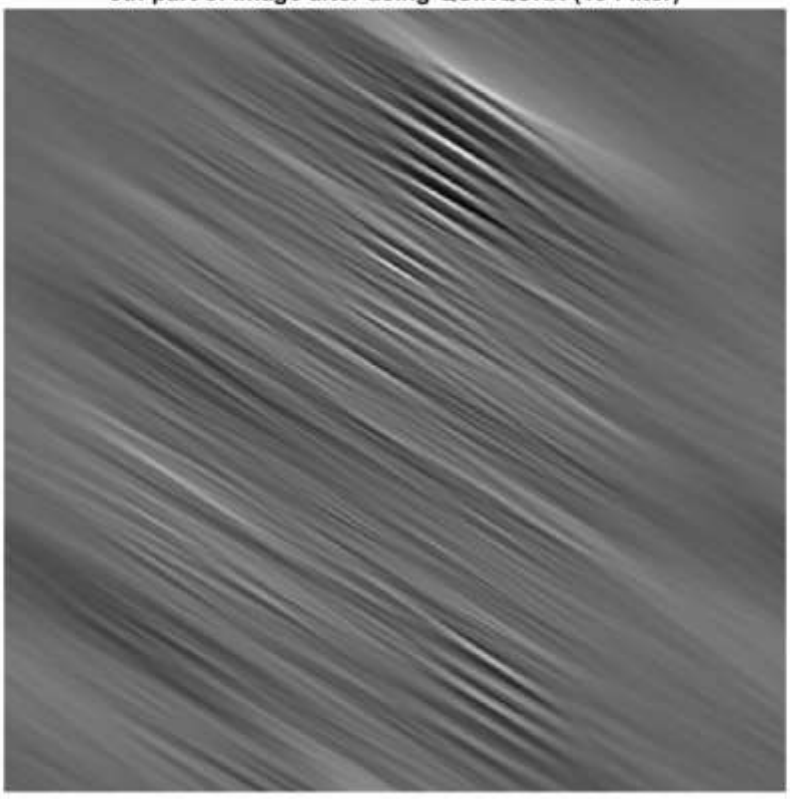


3th part of Image after using QUINQUNX (16 Filter)

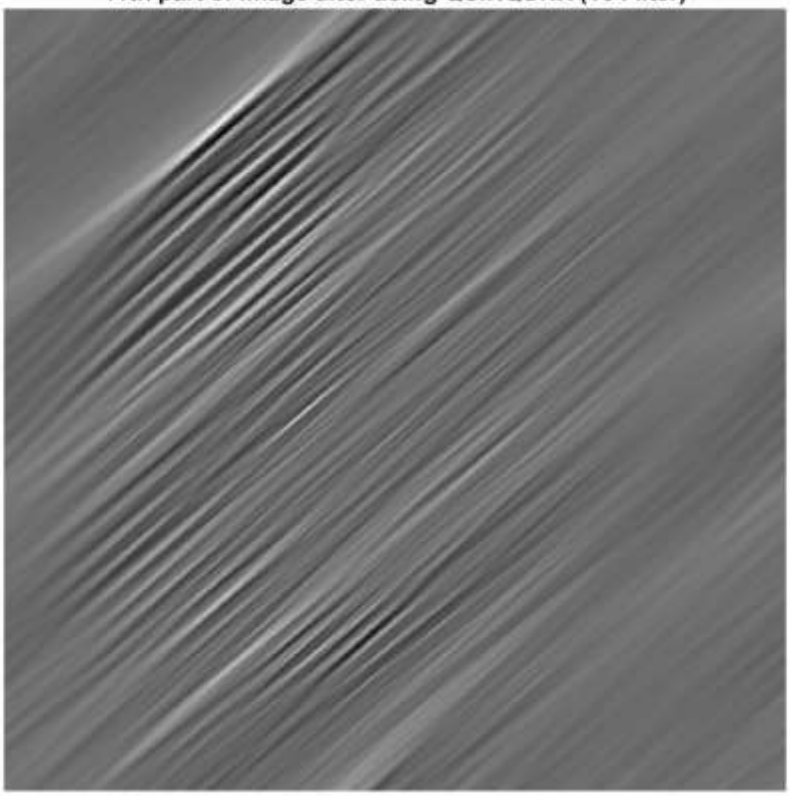


5th part of Image after using QUINQUNX (16 Filter)

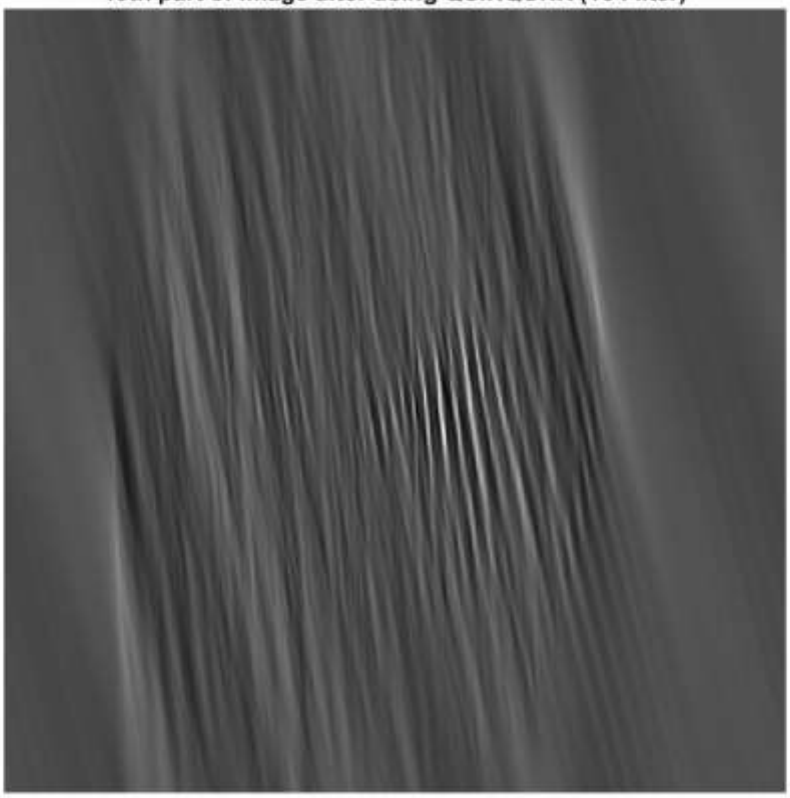
9th part of Image after using QUINQUNX (16 Filter)



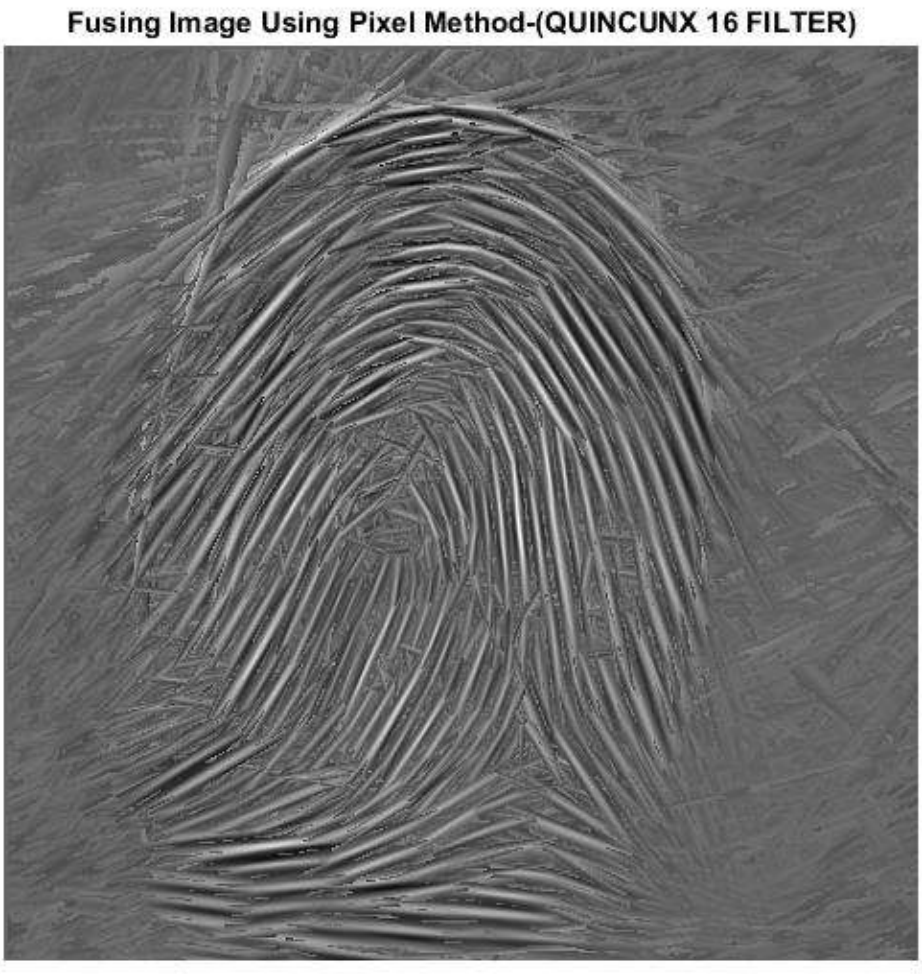
11th part of Image after using QUINQUNX (16 Filter)



13th part of Image after using QUINQUNX (16 Filter)



حال برای بدست آوردن تصویر باید این تصاویر بدست آمده را با یکدیگر ترکیب کرد با توجه به توضیحات مطرح شده برای نحوه مختلف ترکیب تصاویر در اینجا هم از دو روش Pixel fusing و Block fusing استفاده کرده‌ایم که در زیر نتایج آن را شاهد هستیم (توجه شود که در روش ترکیب پیکسلی با توجه بااینکه برای ۱۶ تصویر مختلف باید این کار را پیکسل به پیکسل انجام دهد ران شدن این روش بسیار طولانی است)



ادامه کد مربوطه به این قسمت از سؤال را در زیر شاهد هستیم:

```
%%%%% imfilter
for k=1:16
    Fhat(:,:,k)=Q_16(:,:,k).*F;
end
[M , N] = size(Q_16(:,:,1));

%%%% plot each part of 16 part of Image after filtering
for k=1:16
    fhat=ifft2(Fhat(:,:,k));
    fhat=real(fhat);
    for m=0:w-1
        for n=0:w-1
            final_f(m+1,n+1)=(-1)^(m+n)*fhat(m+1,n+1);
        end
    end
    final_ff(:,:,k) = final_f;
    gcf,figure,imshow(final_f,[]);figure,imshow(final_f,[]),title([num2str(k),'th part of Image
after using QUINQUNX (16 Filter)'])
end


%%%%%% Fusing Image with using BLOCK Method
W = 5;
```

```
w_2 = floor(W/2);

for k=1 : 16
    std_f = zeros(1, 16);
    for m= w_2 +1 : W : M - w_2 -1
        for n= w_2 +1 : W : N - w_2 -1
            for k=1 : 16
                temp = final_ff(m-w_2 : m+w_2 , n-w_2 : n+w_2,k);
                std_f(1,k) = std2(temp);
            end
            ind_max = min(find( std_f == max(std_f)));
            f_fuse_block_16(m-w_2 : m+w_2 , n-w_2 : n+w_2) = final_ff(m-w_2 : m+w_2 , n-w_2 :
n+w_2 ,ind_max);
        end
    end
end

figure,imshow(f_fuse_block_16,[]),title('Fusing Image Using Block Method-(QUINCUNX 16 FILTER)')


%%%%%% Fusing Image with using PIXEL Method
W = 5;
w_2 = floor(W/2);

for k=1 : 16
    std_f = zeros(1, 16);
    for m= w_2 +1 : M - w_2 -1
        for n= w_2 +1 : N - w_2 -1
            for k=1 : 16
                temp = final_ff(m-w_2 : m+w_2 , n-w_2 : n+w_2,k);
                std_f(1,k) = std2(temp);
            end
            ind_max = min(find( std_f == max(std_f)));
            f_fuse_pix_16(m , n) = final_ff(m , n ,ind_max);
        end
    end
end

figure,imshow(f_fuse_pix_16,[]),title('Fusing Image Using Pixel Method-(QUINCUNX 16 FILTER)')
```

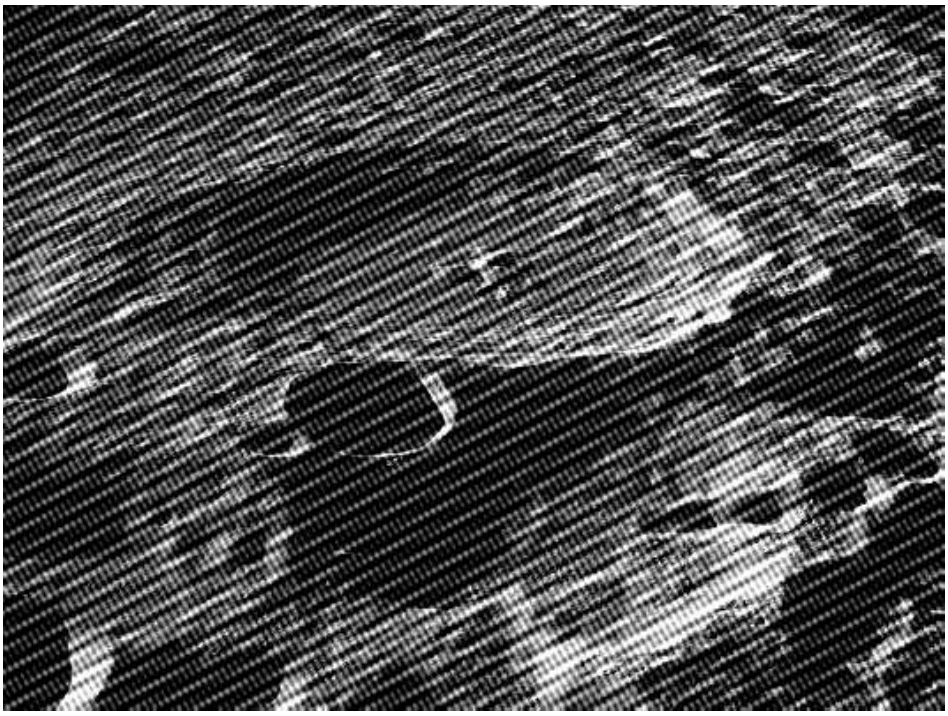
نتیجه‌گیری:

هنگامی‌که از ۱۶ فیلتر استفاده کردیم همان‌طور که انتظار می‌رفت پاسخ نسبت به هنگامی‌که از ۸ فیلتر استفاده می‌کردیم به میزان بسیار اندکی بهبود یافت اما همچنان شاهد هستیم نتایج بدست از فیلترهای گابور به نسبت جواب‌های بدست آمده با استفاده از فیلترهای Quincunx بهتر است که علت آن را می‌توان در این نکته یافت که فیلترهای گابور میزان فاصله بین خطوط اثر انگشت را با محاسبه T های مختلف دخالت می‌دهند درحالی‌که فیلترهای Quincunx فیلترهای جهتی هستند و در یک جهت مشخص فرکانسی، از فرکانس پایین تا فرکانس‌های بالا را در محاسبات دخالت می‌دهند درحالی‌که فیلترهای گابور فیلترهای میان‌گذر با فرکانس مشخص هستند و برای کارکرد اثر انگشت فیلترهای گابور بسیار مناسب هستند (مخصوصاً هنگامی‌که از فرکانس‌های مختلف (T های مختلف) در فیلتر گابور استفاده شود و سپس تصاویر بدست آمده را با هم ترکیب کرد)

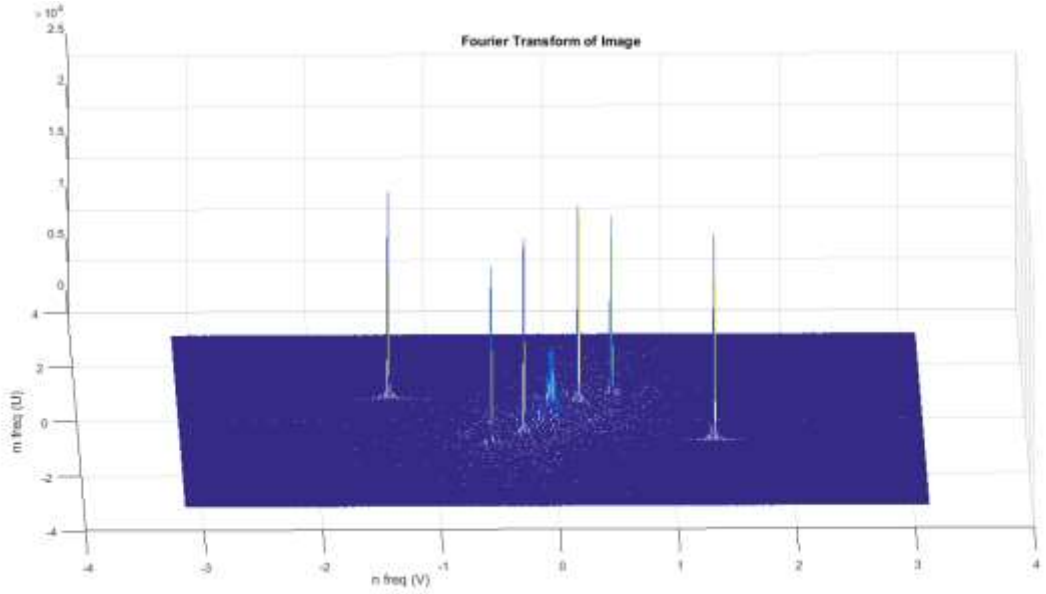
سؤال ۳) کد این سؤال فایل HW5_Q3.m می‌باشد.

تصویر اصلی که شامل تعدادی نویز سینوسی می‌باشد به صورت زیر می‌باشد :

Original Image

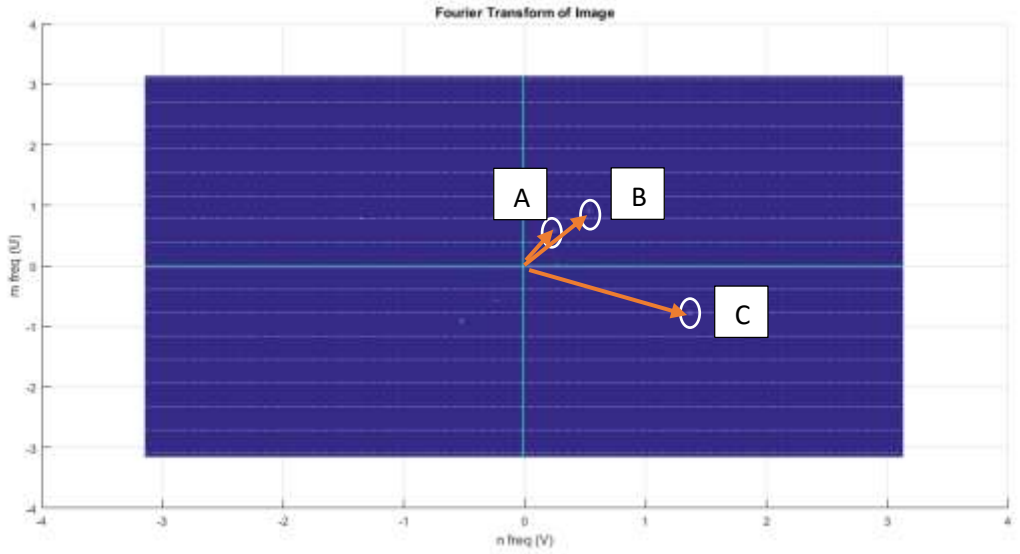


همان‌طور که مشاهده می‌شود خطوط موربی در تصویر وجود دارد که احتمالاً تعدادی موج سینوسی هستند برای بررسی دیدن نواحی رخ داده این سینوسی می‌توان تبدیل فوریه آن را مورد بررسی قرار داد (تبدیل فوریه را پس از قرار دادن آن در بازه [π -π] در زیر مشاهده می‌کنیم):



همان طور که در شکل فوق مشاهده می‌گردد شاهد تعدادی پیک در شکل می‌باشیم که بیان‌گر وجود تعدادی سینوسی در فرکانس‌های خاص است. محتوای فرکانسی که در حوالی فرکانس ۰ می باشد مربوط به شکل اصلی می‌باشد.

در قسمت اول سؤال، خواسته شده است که جهت و فرکانس مربوط به هر یک از سینوسی‌ها را بدست آوریم. در شکل زیر تبدیل فوریه را از نمای بالا پس از حذف محتوای فرکانسی حوالی ۰ (که مربوط به شکل اصلی بود) را شاهد هستیم:



برای مشخص شدن بهتر این نقاط آن‌ها را با دایره مشخص کرده‌ایم، البته می‌دانیم که تبدیل فوریه سیگنال‌های حقیقی حول ۰ متقارن می‌باشد که ما شاهد این نکته در شکل فوق هستیم اما برای بررسی جهت و فرکانس، از نقاطی که در ربع اول و ۴ قرار می‌گیرند، استفاده می‌کنیم.

برای بدست آوردن جهت کافی است اندیس که این نقاط در آن‌ها قرار دارند را بدست آوریم و از اندیس مرکزی کم کنیم $(\frac{M}{2}, \frac{N}{2})$.

اندیس این نقاط که در بالا مشاهده‌شده است را پس از کم کردن از اندیس نقطه مرکزی را در زیر شاهد هستیم

	نقطه A	نقطه B	نقطه C
m	۳۸	۵۹	-۴۹
n	۲۱,۵	۴۵,۵	۱۱۶,۵

برای محاسبه جهت کافی است که $\text{atan}(\frac{m}{n})$ را محاسبه کنیم . در قسمت دیگر که از ما خواسته شده است که فرکانس رو محاسبه کنیم فرکانس را باید در دو جهت m , n محاسبه کنیم میدانیم از ۰-M/2 در بازه ۰-π قرار می‌گیرد درنتیجه برای محاسبه زاویه در جهت m کافی است از رابطه زیر استفاده کنیم (به طریق مشابه زاویه در جهت n نیز قابل محاسبه است):

$$\pi \longrightarrow \frac{N}{2} \quad \Longrightarrow \omega_n = \frac{\pi \times n}{\frac{N}{2}}$$
$$\omega_n \rightarrow n$$

$$\pi \longrightarrow \frac{M}{2} \quad \Longrightarrow \omega_m = \frac{\pi \times m}{\frac{M}{2}}$$
$$\omega_m \rightarrow m$$

در جدول زیر شاهد نتایج مربوط به جهت و فرکانس هستیم:

	نقطه A	نقطه B	نقطه C
جهت	۶۰,۴۹۹۳	۵۲,۳۶۱۱	-۲۲,۸۱۱۷
ω_m	۰,۵۹۶۹	۰,۹۲۶۸	۰,۷۶۹۷
ω_n	۰,۲۵۳۴	۰,۵۳۶۴	۱,۳۷۳۳

در زیر کد مربوط به قسمت اول را شاهد هستیم:

```
clc
clear
close all;

load('p5_1.mat');
figure, imshow(f),title('Original Image')

[M,N]=size(f);

ff =  zeros(M,N);
for m=1:M
    for n=1:N
        ff(m,n) = ((-1)^(m+n)) * f(m,n);
    end
end


F = fft2(ff);
step_u = 2*pi/M; step_v = 2*pi/N;
[M_F,N_F]=meshgrid( -pi:step_v:pi-step_v,  -pi:step_u:pi-step_u);
figure;
mesh(M_F , N_F ,abs(real(F)));
xlabel('n freq (V)'); ylabel('m freq (U)'),title('Fourier Transform of Image')

G= F;
f_m = floor(M/2); f_n = floor(N/2);
G(f_m-20 : f_m+20 , f_n -20:f_n+20) =0;
GG = sort(abs(real(G(:))),'descend');
G_max_ind= length(find(GG> .5*max(GG)));
GG = GG(1:G_max_ind);
for i=1: 2 :length(GG)
    [temp_1 , temp_2] = find(abs(real(F)) == GG(i));
    ind_m(i)= temp_1(1);       ind_n(i)= temp_2(1);
```

```
    ind_m(i+1)= temp_1(2);    ind_n(i+1)= temp_2(2);
end

ind_1_4 = find(ind_n > N/2);
ind_m_2 = ind_m(ind_1_4) - M/2;
ind_n_2 = ind_n(ind_1_4) - N/2;

for i=1 : length(ind_1_4)
    thet(i) = atand(ind_m_2(i) / ind_n_2(i));
    omega_m(i) = abs(ind_m_2(i)*pi/(M/2));
    omega_n(i) = abs(ind_n_2(i)*pi/(N/2));
end
```

در این قسمت از ما خواسته شده است که یک الگوریتم ارائه کنیم که به صورت اتوماتیک سینوسی ها را تشخیص دهد سپس اقدام به حذف سینوسی ها کند. برای پیدا کردن اتوماتیک مقادیر سینوسی از تبدیل فوریه مربوط به آن استفاده می‌کنیم (تبدیل فوریه که به مرکز انتقال داده‌شده است)، پیکسل های حوالی فرکانس ۰ ($\frac{M}{2}$, $\frac{N}{2}$) را در نظر نمی‌گیریم (یک مربع ۴۰ در ۴۰ به مرکز ($\frac{M}{2}$, $\frac{N}{2}$) را در الگوریتم پیدا کردن پیک های تبدیل فوریه دخالت نمی‌دهیم) و سایر عناصر را به صورت صعودی به نزولی مرتب می‌کنیم . اندیس‌های پیک های فرکانسی را که مقدار آن‌ها بیش از نصف بزرگترین پیک باشد را بدست می‌آوریم. مشاهده می‌شود که ۶ نقطه دارای این شرایط هستند که البته می‌دانیم که تبدیل فوریه متقارن است و دو جفت ۳ تایی هستند در زیر این مقادیر که بیشترین مقدار را دارا هستند (جواب بدست آمده دقیقا مشابه آن چیزی است که انتظار داشتیم و در شکل تبدیل فوریه نیز شاهد آن بودیم)و همچنین شماره اندیس‌های مربوطه به آن‌ها را شاهد هستیم:

مقادیر بیشینه	۲۰۲۹۲	۲۰۲۹۲	۱۹۴۰۳	۱۹۴۰۳	۱۷۴۴۷	۱۷۴۴۷
شماره اندیس‌ها	(۲۵۱و۱۵۲)	(۱۵۱و۳۸۳)	(۱۶۴و۲۴۷)	(۲۳۸و۲۸۸)	(۱۴۳و۲۲۳)	(۲۵۹و۳۱۲)

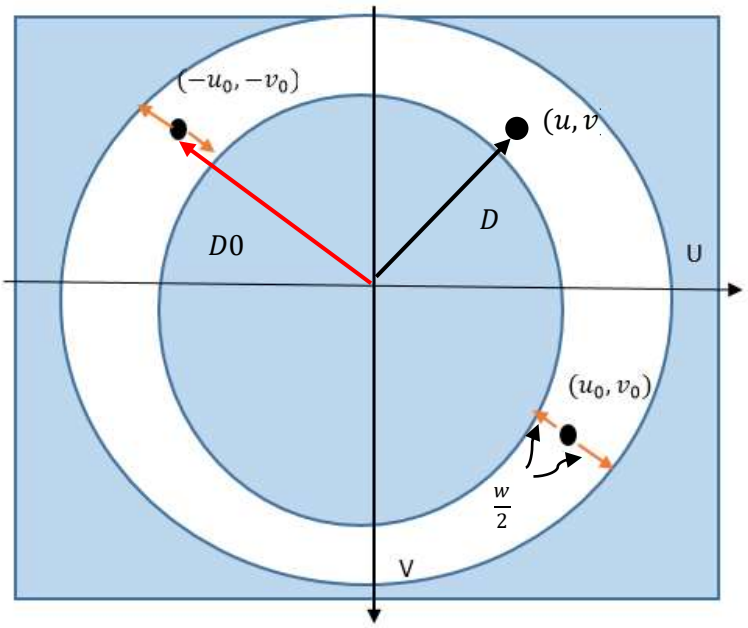
با توجه به اینکه از مقادیر ماکسیموم در قسمت قبل هم استفاده شد، کد مربوطه به این قسمت را در کد قسمت قبل قرار دارد.

حال باید با استفاده از روش‌هایی این مقادیر فرکانسی را حذف کرد و سپس تصویر را بازسازی کرد برای این کار می‌توان از band reject در حوزه فرکانس استفاده کرد.

روش اول) Ideal Band Reject

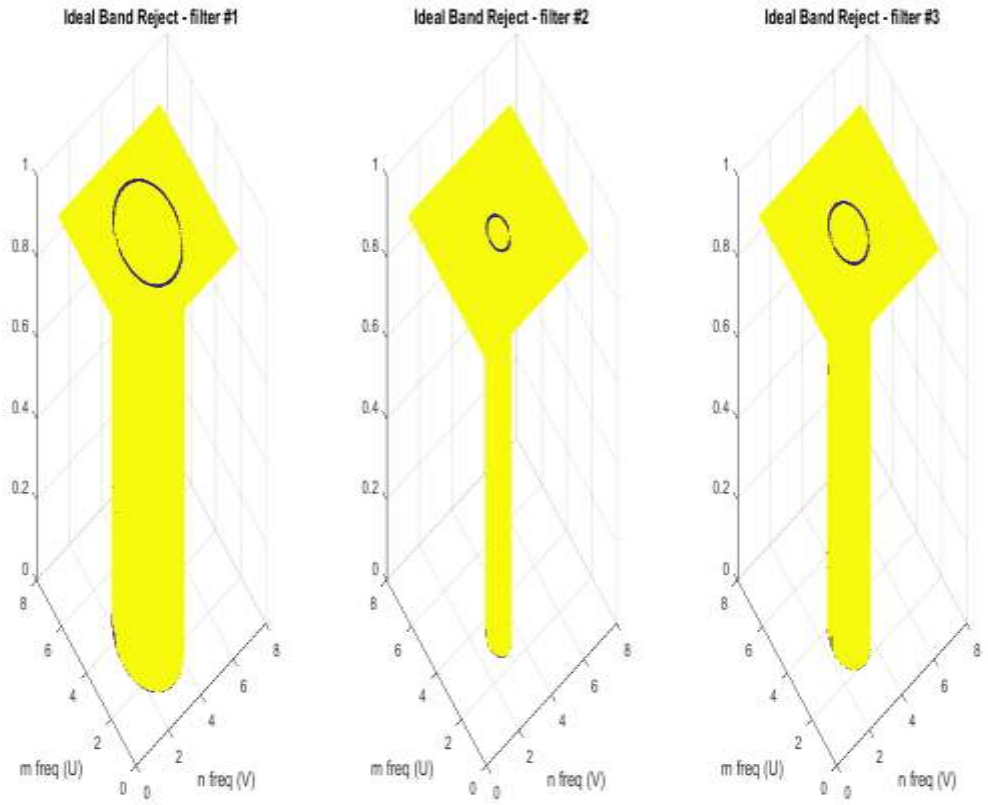
نحوه کار به این‌گونه است که در حوالی فرکانسی که می‌خواهیم آن را حذف کنیم پاسخ فرکانسی را ۰ می‌کنیم و در سایر نقاط پاسخ فرکانسی را ۱ قرار می‌دهیم در شکل صفحه بعد یک نمونه از این فیلتر ها در حوزه فرکانس را شاهد هستیم:

در شکل زیر که پاسخ فرکانسی Band Reject را نشان می‌دهد با استفاده از w ناحیه‌ای که می‌خواهیم پاسخ فرکانسی را در آن ۰ کنیم را مشخص می‌کنیم در شکل زیر ناحیه آبی رنگ دارای پاسخ فرکانسی ۱ و ناحیه سفید رنگ دارای پاسخ فرکانسی ۰ است



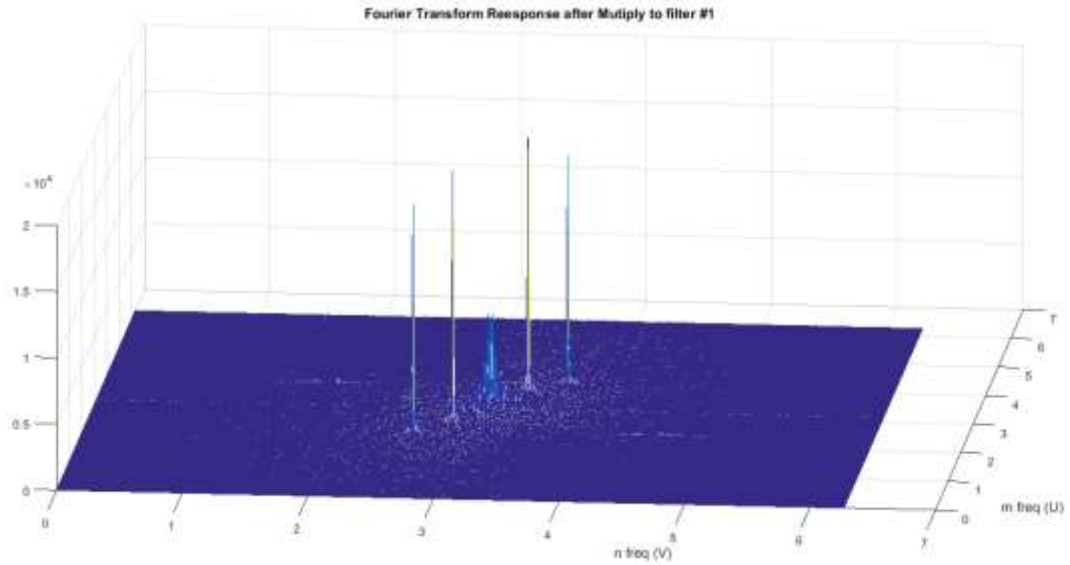
$$H(u,v) = \begin{cases} 1 & D < D0 - \frac{w}{2} \\ 1 & D > D0 + \frac{w}{2} \\ 0 & \text{other} \end{cases}$$

$$D = \sqrt{(u)^2 + (v)^2}$$

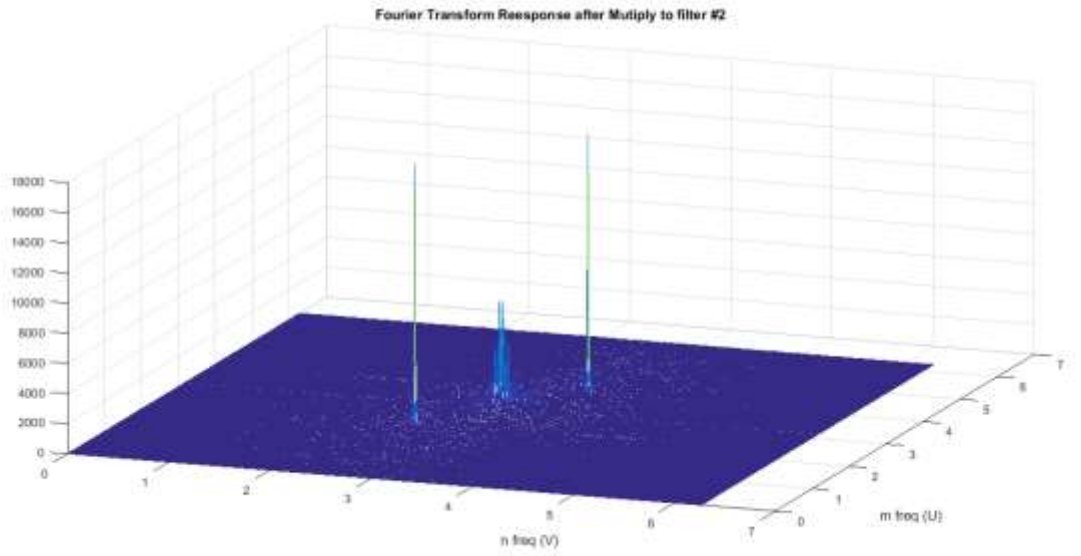


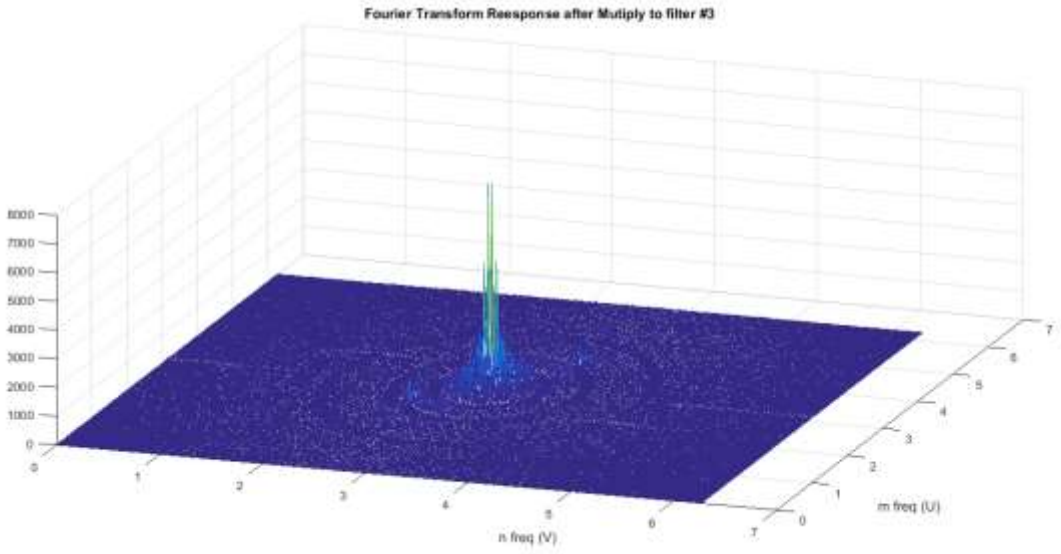
حال هر یک از این فیلتر ها را در پاسخ فرکانسی سیگنالی که دارای موج سینوسی است ضرب می‌کنیم این کار منجر به حذف، سینوسی که در فرکانس متناظر با فیلتر ایده آل می‌گردد.

در زیر نتیجه تبدیل فوریه سیگنال را پس از ضرب هر یک از فیلترهای ایده‌ال مشاهده می‌کنیم:



مشاهده می‌شود در گام اول پیک مربوط به دورترین سینوسی از تبدیل فوریه حذف شده است. انتظار داریم پس از ضرب فیلتر دوم و سوم تنها محتوای فرکانسی که در مرکز قرار دارد باقی بماند در زیر شکل های مربوط به ضرب با فیلتر ۲ و ۳ را شاهد هستیم:





در تمام مراحل تعداد ۵ (w=10) پیکسل را در دو طرف نقطه پیک ۰ کردیم.

حال باید از ماتریس فوریه، اخذ تبدیل فوریه معکوس کنیم و سپس چون که در ابتدا کار، برای این که تبدیل فوریه آن شکل متقارن داشته باشد هر پیکسل را در -1^{m+n} ضرب کرده بودیم دوباره باید این مرحله را برای تمامی پیکسل‌ها انجام دهیم. تصویر نهایی را در زیر شاهد هستیم:

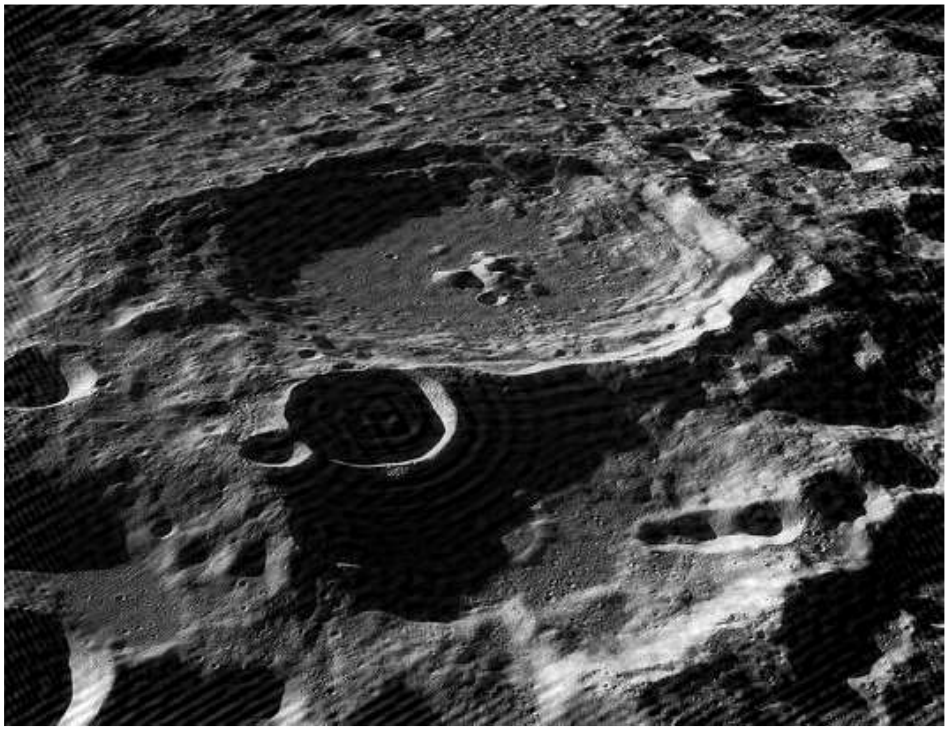
مسئله ای که در اینجا در جواب ما اهمیت دارد این است که میزان W را چگونه انتخاب کنیم که هم سینوسی ها حذف شوند و هم اینکه محتوای فرکانسی شکل اصلی از بین نرود. اگر اندازه W بزرگ انتخاب شود منجربه حذف محتوای فرکانس بالا شکل می‌شود که این منجربه بلور شدن تصویر اصلی میگردد و اگر میزان W کوچک باشد اثر سینوسی ها به طور کامل حذف نمی‌شود و مقداری نویز در تصویر باقی می‌ماند. در زیر پاسخ شکل‌ها را برای ۳ حالت مختلف W=5 , 20 , 30 شاهد هستیم:

در شکل اول که برای W=5 است شاهد هستیم که نتواستیم به طور کامل اثر نویز سینوسی حذف گردد و این نویز ها به وضوح در کناره‌های تصاویر مشهود هستند

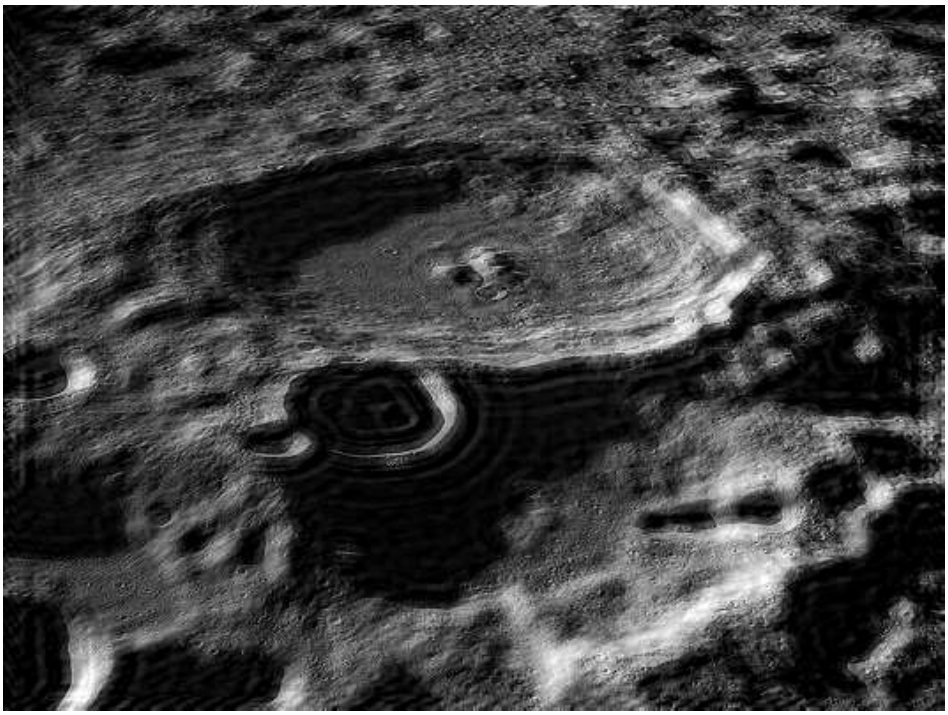
در شکل دوم که برای W=20 است نویز ها بسیار بهتر از شکل اول حذف شده‌اند اما شاهد این نکته هستیم که محتوای فرکانس بالا شکل اصلی تخریب شده است و تصویر حاصله بلور شده است.

در شکل سوم که W=10 انتخاب شده است توانسته ایم یک trade off بین حذف نویز و بلور شدن تصویر بدست آوریم و نتایج بدست آمده به نسبت حالات قبل بهتر می‌باشد.

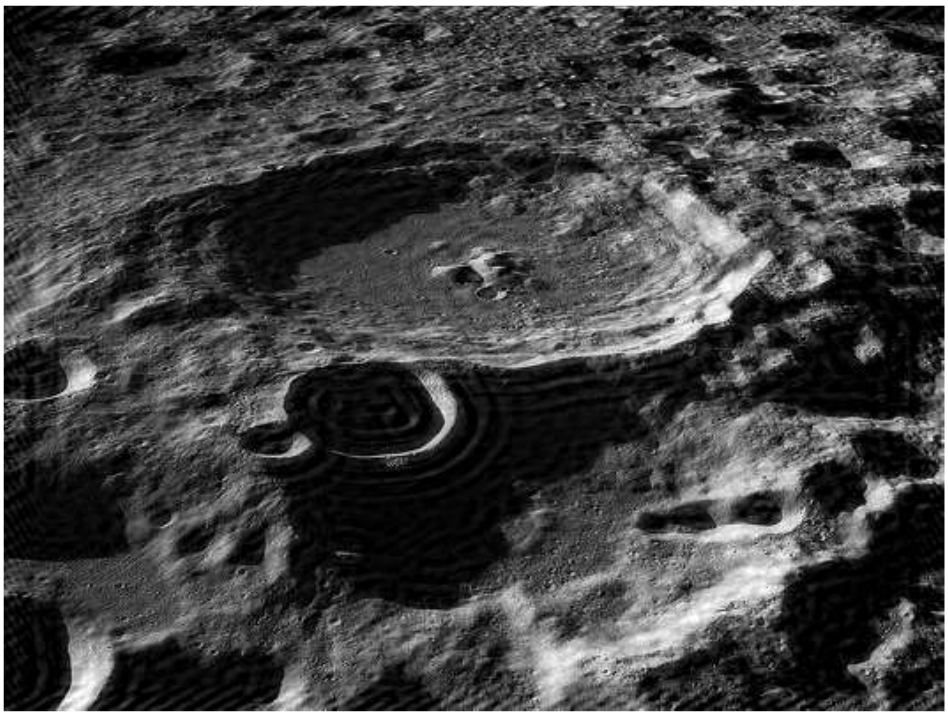
denoising Image with Ideal Band Reject - with W = 5



denoising Image with Ideal Band Reject - with W = 20



denoising Image with Ideal Band Reject - with W = 10



مشاهده می‌شود در تصویر بدست آمده تمامی نویزهای سینوسی به‌خوبی حذف شده‌اند اما مشاهده می‌شود که اندکی لبه ها بلور شده است که علت این امر حذف مقداری از محتوای فرکانس بالا تصویر به هنگام حذف نویز های سینوسی است که منجربه تخریب اندکی در نواحی فرکانس بالا تصویر شده است.

```
%%%% Ideal Band Reject Filter
D0=[]; p=0;
W=10; %%% 10 pixel
for i=1:length(ind_m)
    dist = sqrt((ind_m(i)-M/2)^2 + (ind_n(i)-N/2)^2);
    count = 0;
    %%% check this distanse existed before or not
    for j=1: length(D0)
        dif = D0(j)-dist;
        if abs(dif) <= W
            D0(j) = (D0(j) + dist)/2;
            count = count +1;
        end
    end

    if count ==0
        p=p+1;
        D0(p) = dist;
    end

end

for i=1:length(D0)
    H = zeros(M , N);
    for u=1:M
        for v=1:N
```

```
            D = sqrt((u-M/2)^2 + (v-N/2)^2);
            if D <= D0(i)-W/2 || D >= D0(i)+W/2
                H(u,v) = 1;
            end
        end
    end
    H_ideal(:,:,i) = H;
end

figure,
for i=1:length(D0)
    temp = H_ideal(:,:,i);
    step_u = 2*pi/M; step_v = 2*pi/N;
    [M_F,N_F]=meshgrid( 0:step_v:2*pi-step_v,  0:step_u:2*pi-step_u);
    subplot(1,3,i);
    mesh(M_F , N_F ,temp);
    xlabel('n freq (V)'); ylabel('m freq (U)'),title(['Ideal Band Reject - filter #',
num2str(i)])
end
%%%%%
F_hat= F;
for i=1:length(D0)
    temp = H_ideal(:,:,i);
    F_hat = F_hat.* temp;
    step_u = 2*pi/M; step_v = 2*pi/N;
    [M_F,N_F]=meshgrid( 0:step_v:2*pi-step_v,  0:step_u:2*pi-step_u);
    figure;
    mesh(M_F , N_F ,abs(real(F_hat)));
    xlabel('n freq (V)'); ylabel('m freq (U)')
end

g = real(ifft2(F_hat));
for m=1:M
    for n=1:N
        gg(m,n) = ((-1)^(m+n)) * g(m,n);
    end
end
figure, imshow(gg)
```

روش دوم) Butterworth Notch Filter

در این روش تنها محتوای فرکانسی در نقاطی که سینوسی دارای پیک فرکانسی است حذف می‌گردد و مانند روش band reject نیست که به صورت دایره ای فرکانس ها را حذف کند از این رو انتظار می‌رود این روش پاسخ بهتری نسبت به روش Band Reject داشته باشد چون محتوای فرکانس بالا شکل اصلی را کمتر تخریب می کند.

در این روش نیز پاسخ فرکانس متقارن شده تصویر در دست را باید در یک فیلتر ضرب کرد به گونه ای که این فیلتر در تمام نقاط مقدار ۱ داشته باشد به جز در حوالی نقاطی که ما دارای پیک سینوسی هستیم. برای بدست آوردن این پیک های سینوسی از روش که در قسمت قبل توضیح داده شد، استفاده کرده‌ایم. در اینجا نیز می‌توان از یک رابطه ایده ال استفاده کرد بگونه ای که در حوالی پیک سینوسی تمام پاسخ فرکانسی فیلتر ۰ باشد و در سایر نقاط پاسخ فرکانسی ۱ باشد (تعیین فاصله نقاطی که می‌خواهیم پاسخ فرکانسی ۰ باشد با استفاده از یک سطح آستانه به مانند W در سؤال قبل ، معین می‌گردد) اما استفاده از یک فیلتر Butterworth به جای ایده آل می‌تواند منجربه پاسخ بهتر گردد چون که باتورث در نقطه ای که دقیقا پیک رخ داده با شدت بیشتر پیک را حذف می‌کند (پاسخ فرکانسی فیلتر در این نقطه ۰ است) و در سایر نقاط اطراف با شدت کمتری پاسخ فرکانسی رو تضعیف می‌کند که این کار باعث می‌شود محتوای فرکانس بالا شکل اصلی به طور معقول تری نگه داشته شود و پیک سینوسی به‌خوبی حذف گردد.

رابطه باتورث را در زیر شاهد هستیم:

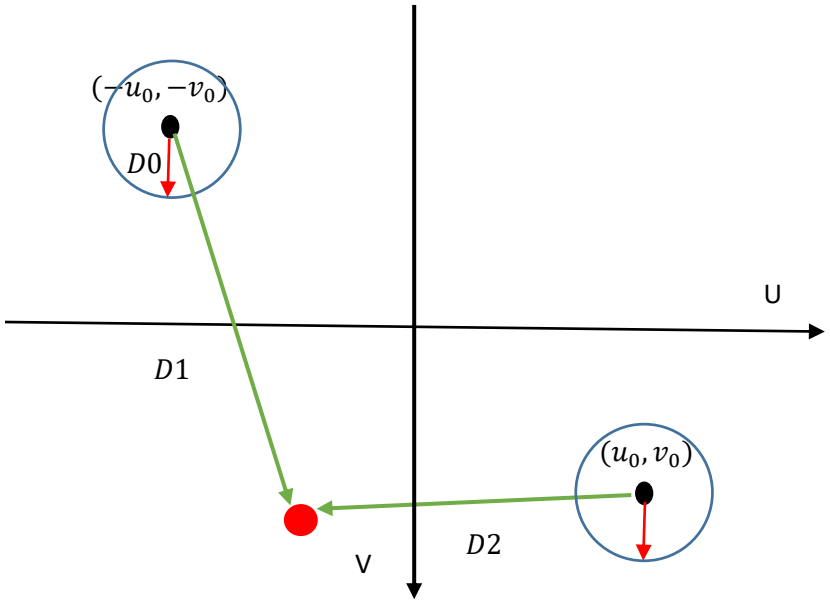
$$H(u,v)=\frac{1}{1+\left(\frac{D0^2}{D1(u,v)\times D2(u,v)}\right)^n}$$

که در این رابطه n و $D0$ ثابت‌های رابطه باتورث هستند که تعیین کننده میزان شدت ۰ شدن در حوالی نقطه مرکزی (u_0, v_0) توسط n و همچنین شعاعی که می‌خواهیم شاهد تضعیف باشیم توسط $D0$ تعیین میکردد.

نحوه بدست آوردن $D1$ و $D2$ نیز به صورت زیر میباشد

$$D1 = \sqrt{(u - u_0)^2 + (v - v_0)^2}$$

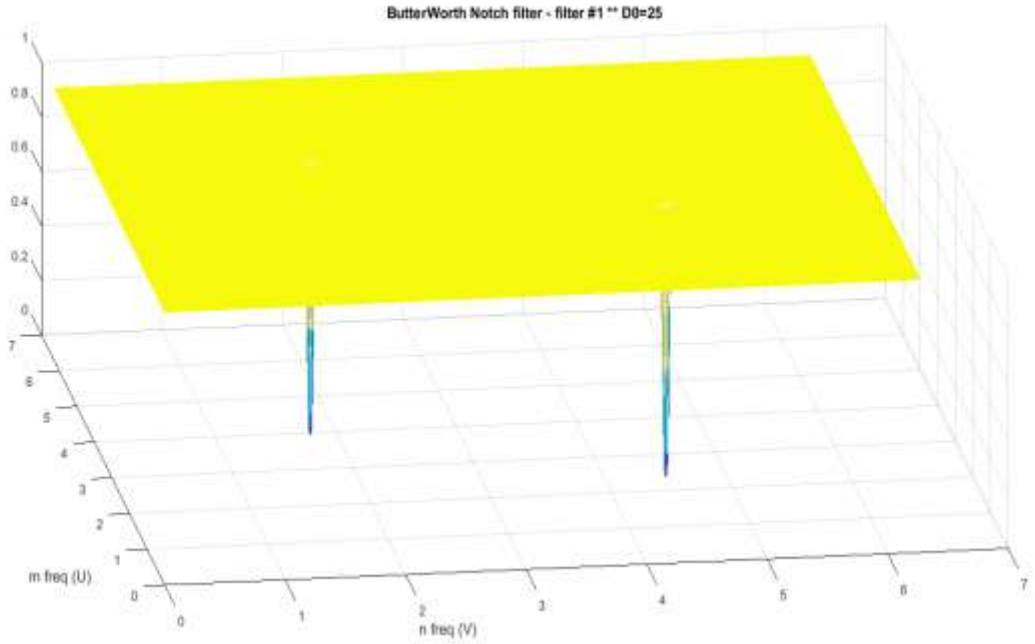
$$D2 = \sqrt{(u + u_0)^2 + (v + v_0)^2}$$

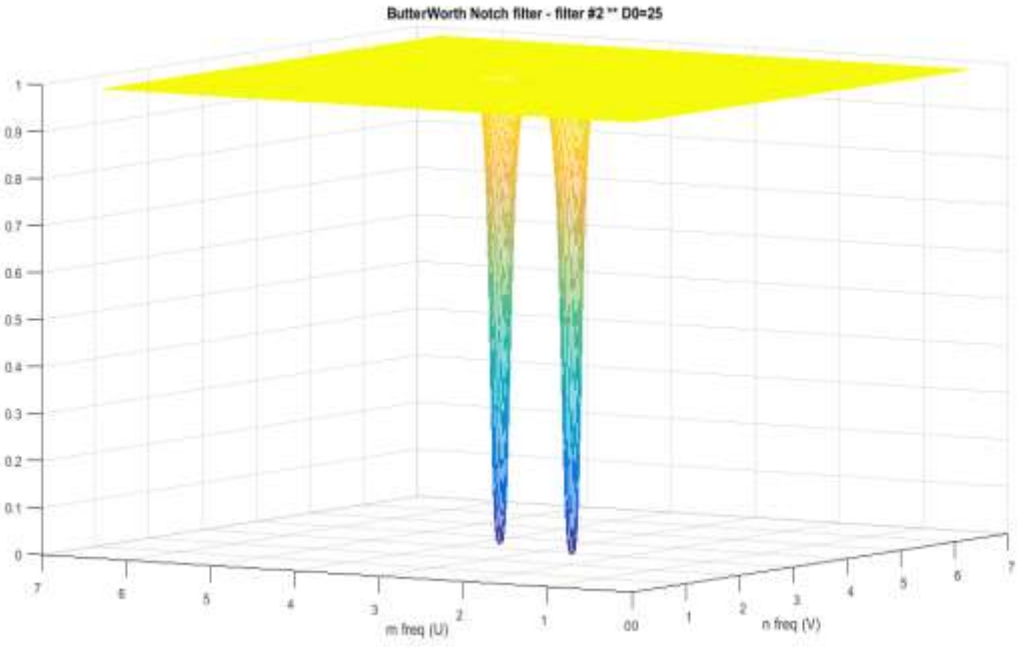


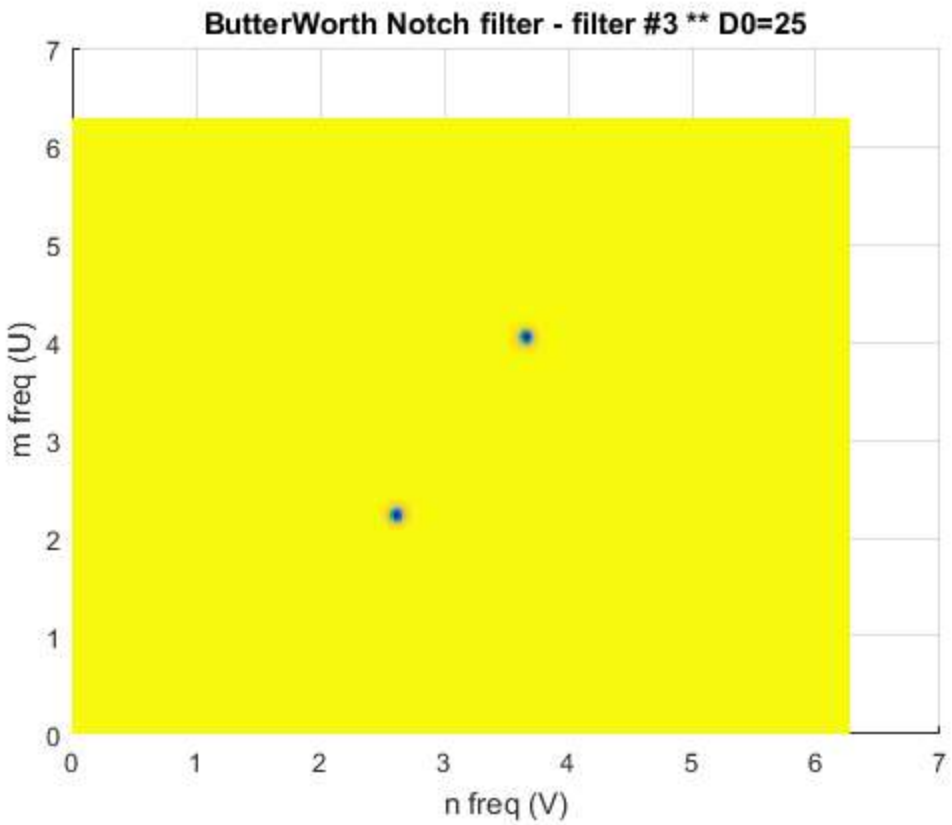
در شکل فوق خارج از دایره های آبی رنگ (مخصوصاً در نقاطی که اندکی از شعاع دورتر باشد) پاسخ فرکانسی برابر ۱ است و در مرکز دایره پاسخ فرکانسی فیلتر برابر ۰ است هر چه از مرکز دایره به سمت شعاع دایره حرکت می‌کنیم پاسخ فرکانسی افزایش بیشتری پیدا می‌کند این

پاسخ فرکانسی فیلتر باعث می‌شود پس از ضرب شدن در پاسخ فرکانسی تصویر، اثر سینوسی ها حذف شده و محتوای فرکانس بالا هم جندان آسیب نبیند.

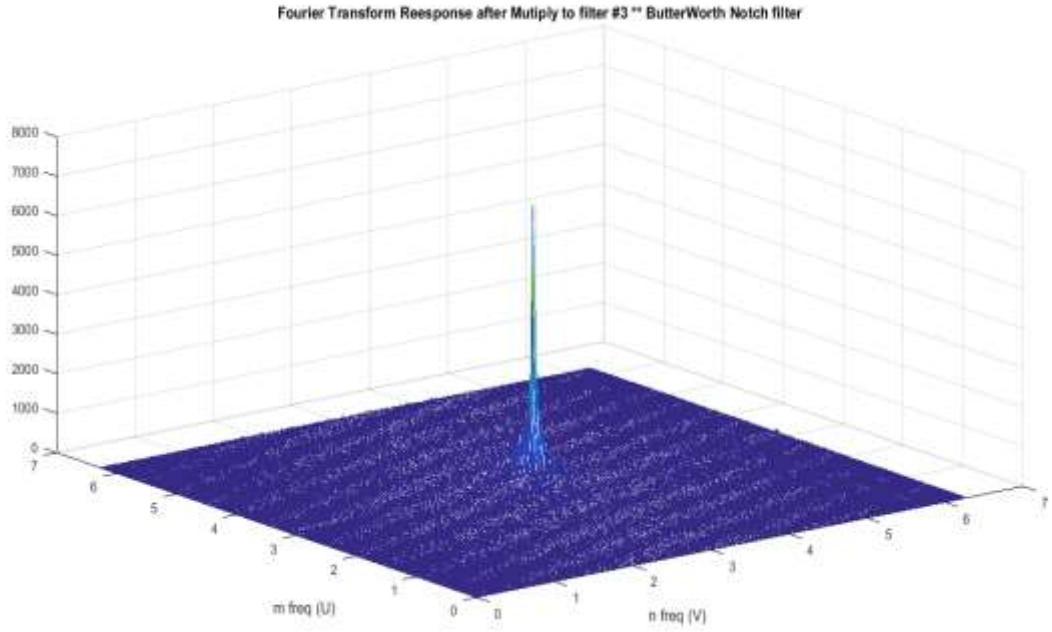
پاسخ فرکانسی سه فیلتر را برای D0=25 , n=4 از زوایای مختلف مشاهده می‌کنیم:







پاسخ فرکانسی شکل اصلی را پس از اعمال هر ۳ فیلتر در پایین مشاهده می‌کنیم:

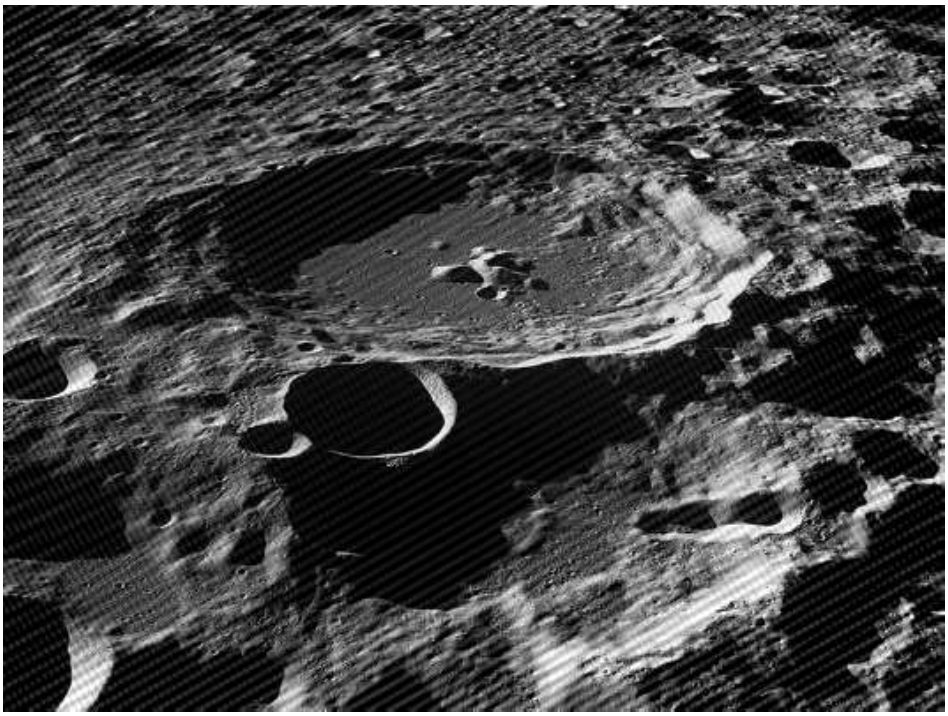


همان‌طور که مشاهده می‌کنیم در این روش محتوای فرکانس بالا بسیار بهتر از روش قبل حفظ شده است (اگر به تبدیل فوریه شکل قبل دقت شود ملاحظه می‌کنیم دوایری در تبدیل فوریه مشاهده می‌شود که ۰ شده‌اند و محتوای فرکانسی شکل را بین برده آند اما در این روش این دوایر بزرگ وجود ندارند و محتوای فرکانسی بسیار بهتر حفظ می‌شوند).

در اینجا نیز انتخاب D0 از اهمیت زیادی برخوردار است. اگر D0 کوچک باشد به‌خوبی نمی‌تواند سینوسی‌ها را حذف کند و اگر خیلی بزرگ باشد محتوای فرکانسی شکل اصلی را از بین می‌برد در زیر نتایج را برای D0 = 10 , 40 , 25 شاهد هستیم:

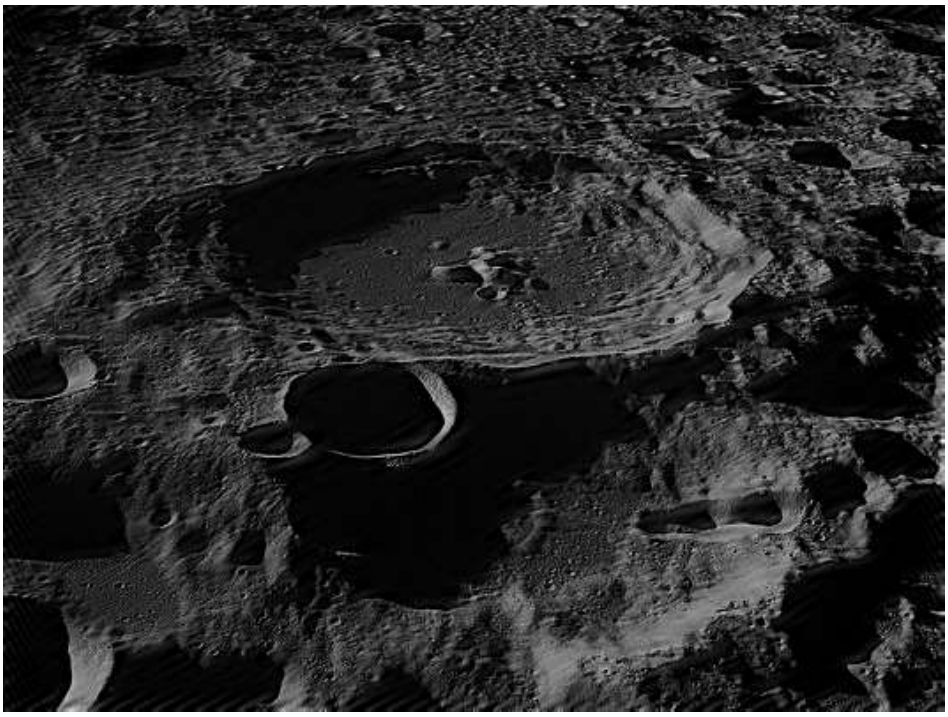
همان‌طور که برای D0=10 مشاهده می‌کنیم نویزهای سینوسی به طور کامل حذف نشده اند و شکل بدست آمده چندان مطلوب نیست.

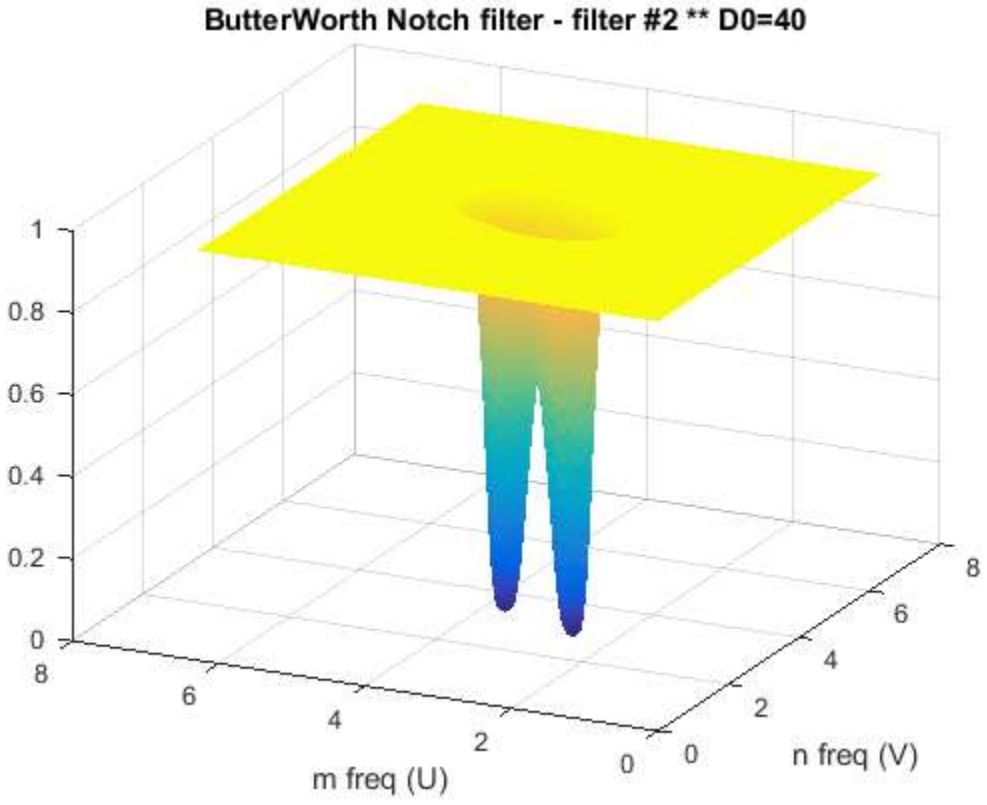
denoising Image with ButterWorth Notch filter - D0 = 10



در شکل صفحه بعد نتیجه بدست آمده را برای D0=40 شاهد هستیم در این حالت اگرچه تمام نویزهای سینوسی به‌خوبی حذف شده‌اند اما مشاهده می‌کنیم که علاوه بر اینکه شکل بلور شده است (که به علت از بین رفتن مقداری از محتوای فرکانس بالا به علت بزرگ شدن چاهک های فرکانسی است) حتی برخی از ویژگی های فرکانس پایین نظیر شکل کلی و رنگ ها نیز در برخی مناطق تغییر یافته است (این ویژگی ها اغلب در فرکانس‌های پایین است) که انتظار رخ دادن این امر را نداشتیم برای بررسی علت رخ داد این موضع به پاسخ فرکانسی فیلترهای مربوطه توجه کردیم مشاهده شد که فیلتر دوم به علت بزرگ شدن بیش از حد چاهک های فرکانسی، و همچنین به علت این‌که در نزدیکی مبدأ قرار دارد محتوای فرکانسی ، فرکانس‌های حوالی ۰ را نیز تحت تاثیر قرار داده و تا حد زیادی آن‌ها را تضعیف کرده است که شکل مربوط به این فیلتر نیز نشان داده‌شده است.

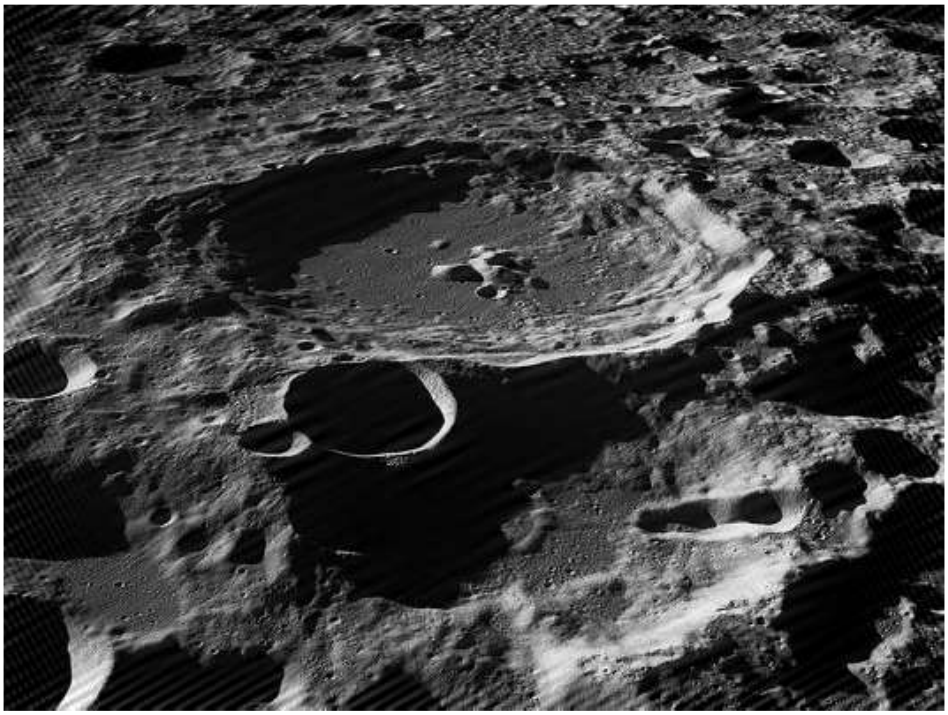
denoising Image with ButterWorth Notch filter - D0 = 40





در نتیجه باید یک trade off بین حذف نویز و از بین رفتن محتوای فرکانسی برقرار کرد. برای D0=25 مشاهده شد تا حد قابل قبولی نویز حذف شده است و همچنین توانسته به‌خوبی کلیت شکل را حفظ کند و جواب کاملا مطلوبی دارد.

denoising Image with ButterWorth Notch filter - D0 = 25



در ادامه کد مربوط به این قسمت را شاهد هستیم:

```
%%%%%%%%%% Butterworth Notch filter
d0=25; n=4;
[r,c,d] = size(ind_pa);
for i=1 : d
    temp_1 = ind_pa(1,:,i);
    temp_2 = ind_pa(2,:,i);
    for u=1:M
        for v=1:N
    D_1 = sqrt( (u-temp_1(1))^2 + (v-temp_1(2))^2);
    D_2 = sqrt( (u-temp_2(1))^2 + (v-temp_2(2))^2);
    H(u,v) = inv(1+ (d0^2/(D_1*D_2))^n);
        end
    end
    H_Butter(:,:,i) = H;
end


for i=1:d
    temp = H_Butter(:,:,i);
    step_u = 2*pi/M; step_v = 2*pi/N;
    [M_F,N_F]=meshgrid( 0:step_v:2*pi-step_v,  0:step_u:2*pi-step_u);
%     figure,
%     mesh(M_F , N_F ,temp);
%     xlabel('n freq (V)'); ylabel('m freq (U)'),title(['ButterWorth Notch filter - filter #',
num2str(i),' ** D0=',num2str(d0)])
end
```

```
%%%%%
F_hat= F;
for i=1:d
    temp = H_Butter(:,:,i);
    F_hat = F_hat.* temp;
    step_u = 2*pi/M; step_v = 2*pi/N;
    [M_F,N_F]=meshgrid( 0:step_v:2*pi-step_v,  0:step_u:2*pi-step_u);
    figure;
    mesh(M_F , N_F ,abs(real(F_hat)));
    xlabel('n freq (V)'); ylabel('m freq (U)'),title(['Fourier Transform Reesponse after Mutiply
to filter #', num2str(i),' ** ButterWorth Notch filter'])
end


g = real(ifft2(F_hat));
for m=1:M
    for n=1:N
        gg(m,n) = ((-1)^(m+n)) * g(m,n);
    end
end
figure, imshow(gg),title(['denoising Image with ButterWorth Notch filter - D0 = ',num2str(d0)])
```

نتیجه‌گیری :

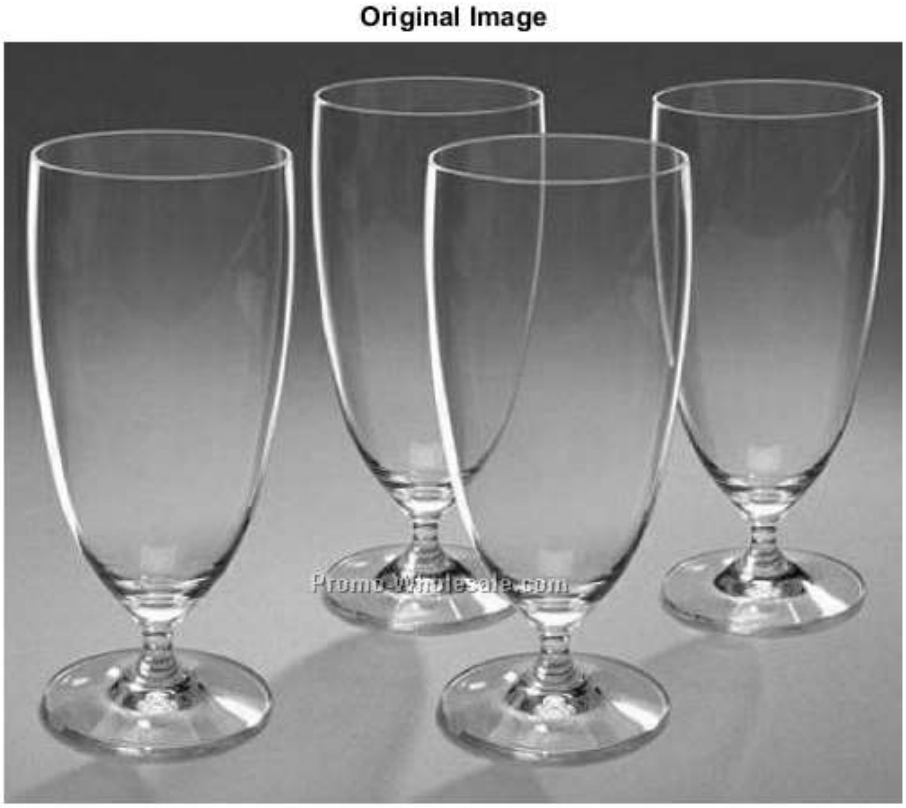
با بررسی نتایج بدست آمده از روش Band Reject و Notch Filter مشاهده می‌شود روش Notch Filter بسیار بهتر از روش Band Reject عمل کرده است که این امر دقیقا مطابق انتظار ما بود. علت آن‌هم همان‌گونه که در طول گزارش بیان شد این نکته است که در روش دوم محتوای فرکانسی بسیار بهتر از روش اول حفظ می‌گردد و لبه ها در روش دوم بسیار واضح‌تر و میزان بلور شدگی بسیار کمتر است.

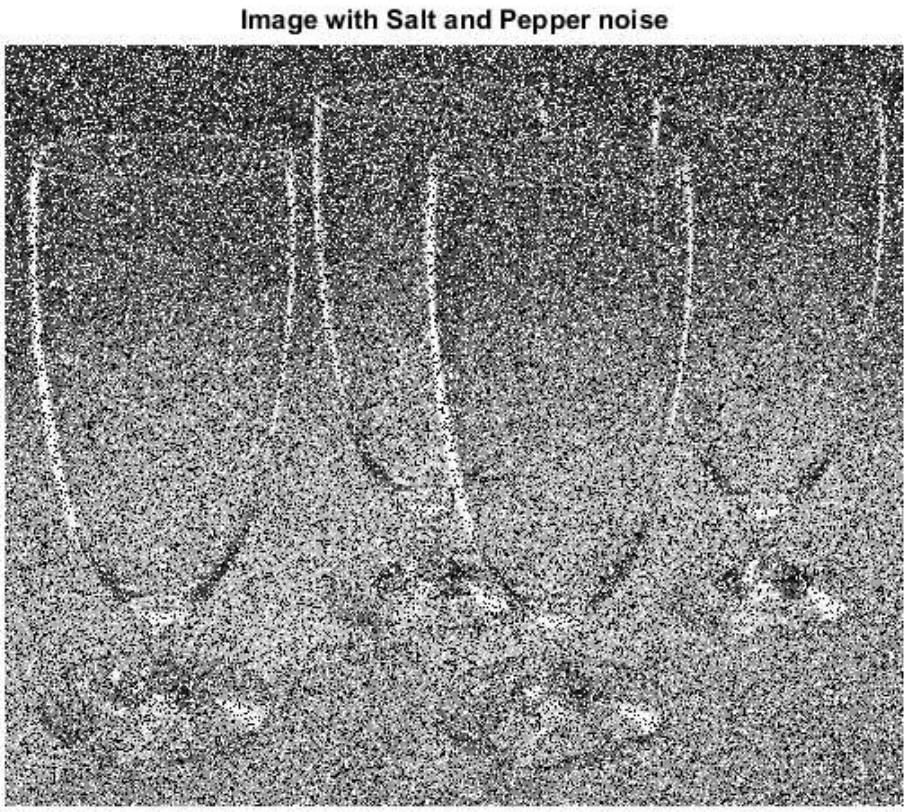
سؤال چهارم) الف) (کد این سؤال فایل HW5_Q4_A.m می‌باشد.)

در این سؤال ابتدا یک تصویر را توسط نویز نمک-فلفل نویزی می‌نماییم و سپس تلاش می‌کنیم که توسط قیلتر median آن را دی‌نویز نماییم. فیلترینگ مدین به این صورت می‌باشد که برای هر پیکسل، یک پنجره با یک بعد خاص که مد نظر ما می‌باشد را در نظر گرفته و سپس سطوح روشنایی پیکسل‌های موجود در این پنجره را sort نموده و پیکسل میانی، median را می‌دهد. سپس median به‌دست آمده را جایگزین پیکسل اولیه می‌نماییم و برای تمامی پیکسل‌ها این کار را تکرار می‌نماییم. توجه شود که در این روش، ابعاد پنجره برای کل تصویر ثابت می‌ماند. همچنین از آن‌جا که در دنباله‌ی sort شده تمامی فلفل‌ها و یا صفرها در لبه‌ی پایین دنباله و تمامی نمک‌ها در لبه‌ی بالای دنباله جمع می‌شوند امید زیادی داریم که median که عنصر وسطی دنباله‌ی sort شده است نه نمک باشد و نه فلفل و بدین‌صورت حتی‌الامکان بتوانیم نویزهای نمک و فلفل را حذف نماییم.

در ابتدا تصویر اصلی را توسط کد زیر با ۴۰ درصد نویز نمک و فلفل آغشته می‌نماییم که به‌صورت زیر درخواهد آمد:

```
I = imread('glass2.jpg');
I = rgb2gray(I);

% figure,imshow(I), title('Original Image');

%%%% Add salt and pepper noise
f = imnoise(I,'salt & pepper',0.4);
figure, imshow(f),title('Image with Salt and Pepper noise');
```





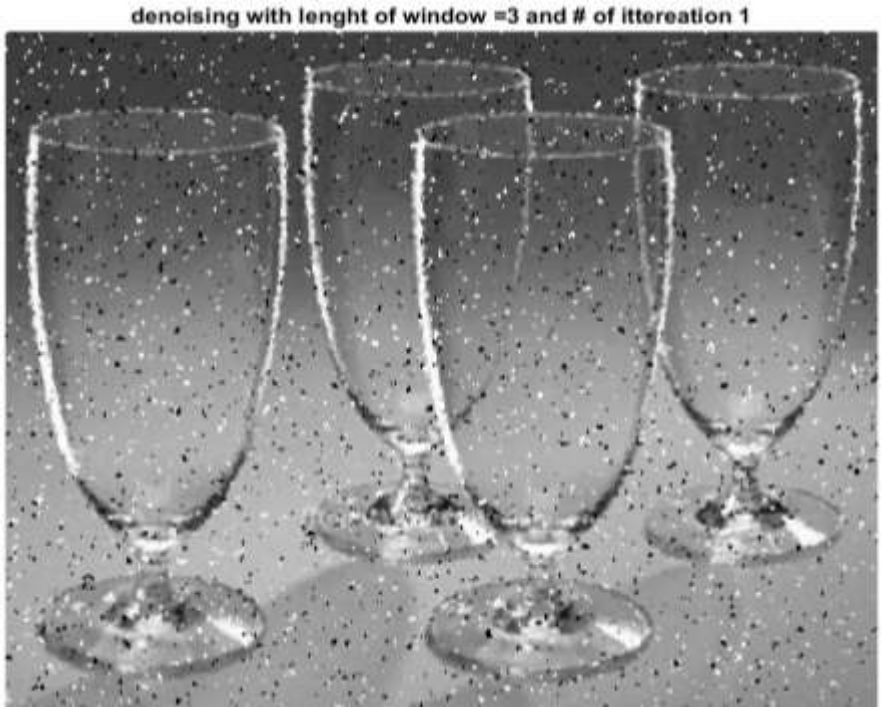
برای حذف نویزهای نمک و فلفل دو مسئله را بررسی می‌نماییم. در ابتدا برای یک بار تکرار و ابعاد مختلف پنجره مسئله‌ی کاهش نویز را بررسی می‌نماییم و در ادامه برای پنجره‌ی ثابت ولی با تکرارهای مختلف، این مسئله مورد بررسی قرار می‌گیرد. می‌خواهیم مقایسه نماییم که در کدام حالت جواب قابل قبول‌تری به‌دست می‌آید.

کد برنامه به صورت زیر است:

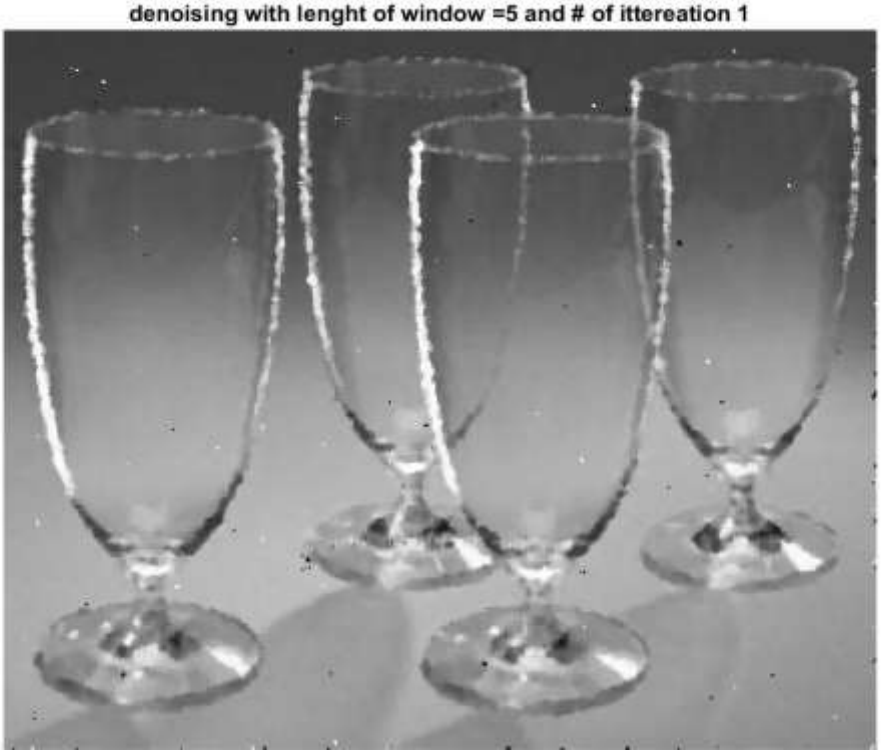
```
clc;
clear;
close all;

I = imread('glass2.jpg');
I = rgb2gray(I);

% figure,imshow(I), title('Original Image');

%%%% Add salt and pepper noise
f = imnoise(I,'salt & pepper',0.4);
figure, imshow(f),title('Image with Salt and Pepper noise');

[M,N] = size(f);
```

```
for W=3:-2:3                    %% window length
    temp_I = f;
    l_w = floor(W/2);
    for itt=1:8                 %% # of itteration
        p=0; g=[];
        for m=l_w+1 : M - l_w
            p= p+1; q=0;
            for n=l_w+1 : N - l_w
                q=q+1;
                temp = temp_I(m-l_w : m+l_w , n-l_w : n+l_w);
                sort_t = sort(temp(:));
                med = median(sort_t);

                g(p,q) = med;
            end
        end
        temp_I(l_w+1 : M - l_w,l_w+1 : N - l_w) = g;
        gcf = figure;
        imshow(g,[]),title(['denoising with lenght of window =',num2str(W),' and # of
ittereation ',num2str(itt)])
        tit_pic = sprintf('denoise_W_%d___ITT_%d.jpg',W,itt);
        saveas(gcf,tit_pic)
    end
end

% figure, imshow(g),title(['denoising with lenght of window =',num2str(W),' and # of
ittereation
```

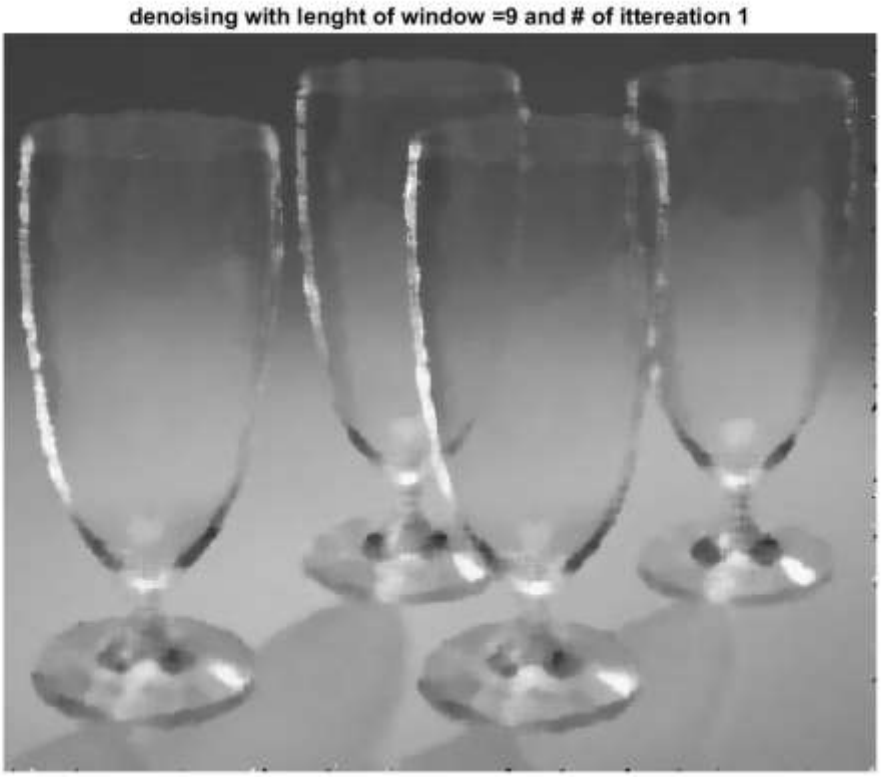
در ابتدا اثر تغییر ابعاد پنجره را مشاهده می‌نماییم. برای این منظور ما برنامه را برای یک بار تکرار و به ازای پنجره‌هایی با ابعاد ۳، ۵، ۷و ۹ اجرا نموده تا نتایج را مقایسه نماییم. نتایج به صورت زیر خواهد بود:



denoising with lenght of window =5 and # of ittereation 1



denoising with lenght of window =7 and # of ittereation 1



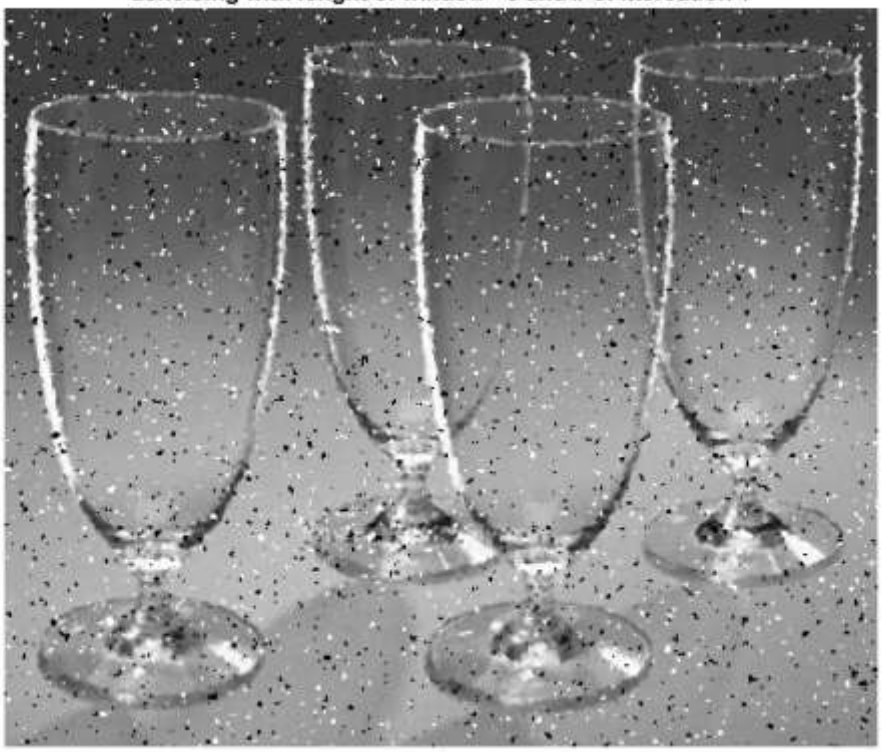
با مقایسه‌ی نتایج حاصل از اضافه کردن ابعاد پنجره مشاهده می‌شود که در پنجره به طول ۳ همچنان مقادیر زیادی از نویزها حذف نشده‌اند. در پنجره به طول ۵ نویزها به‌خوبی حذف شده‌اند و مقادیر بسیار کمی از نویز باقی مانده‌است. با بیشتر شدن ابعاد پنجره درست است که نویزها به طور کامل حذف می‌شوند اما لبه‌ها خورده می‌شود و تصویرها به شدت با اضافه شدن ابعاد پنجره بلور می‌شوند همچنین یکی دیگر از اشکالات اضافه شدن ابعاد پنجره این است که مقدار هر پیکسل با پیکسلی جایگزین می‌شود که با آن correlation کم‌تری دارد. بنابراین ما برای این حالت و در یک بار تکرار، ابعاد پنجره‌ی ۵ را مناسب‌تر می‌بینیم که هم توانسته است نویز را به طور قابل قبولی حذف نماید و هم تصویر خیلی بلور نشده ست.

اما از آن‌جا که فیلتر مدین لبه ها را حفظ می‌نماید (مخصوصاً برای فیلترهای با طول کوچک) ؛ می‌توان آن را به صورت iterative یعنی به ازای چندبار تکرار برای تصویر به کار برد.

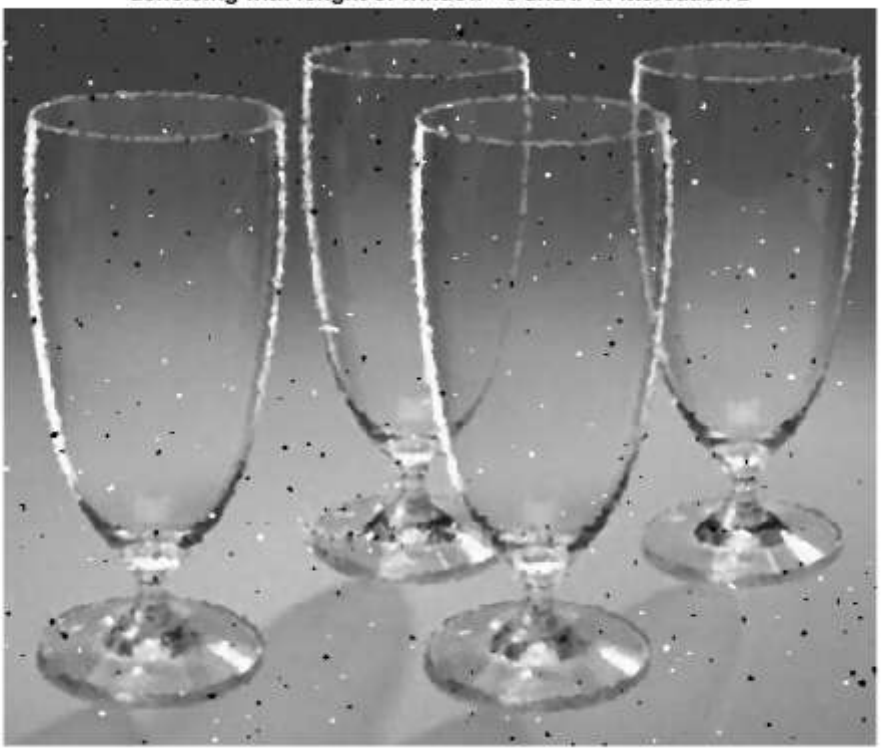
برای مثال در ابتدا پنجره‌ی ۳ را که در بالا نتایج آن را برای یک بار تکرار مشاهده نمودیم را به ازای چند تکرار بررسی می‌نماییم.

نتایج به صورت زیر می‌باشند:

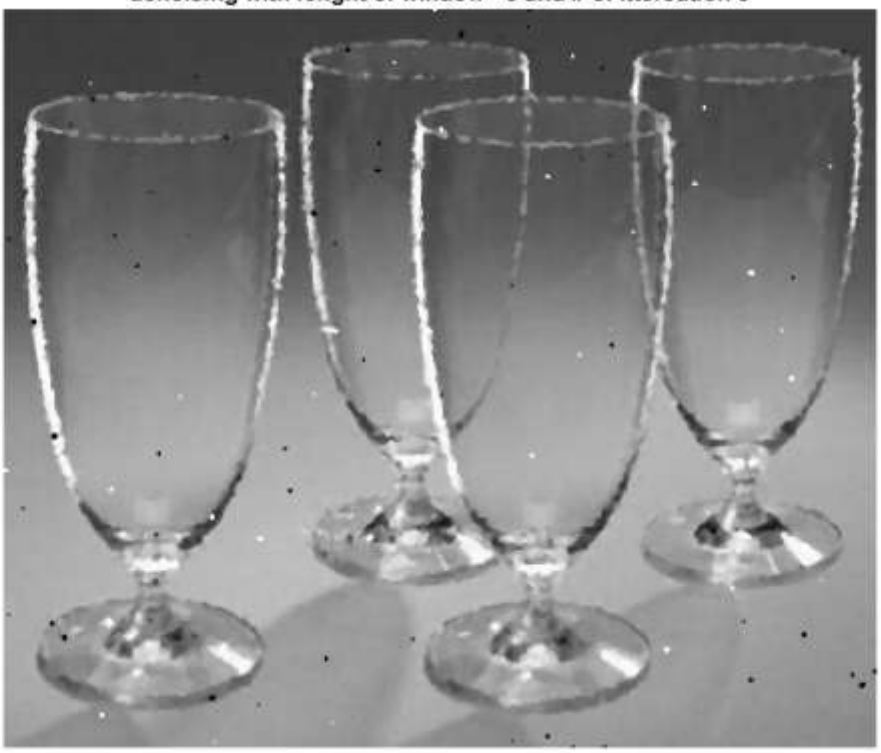
denoising with lenght of window =3 and # of ittereation 1



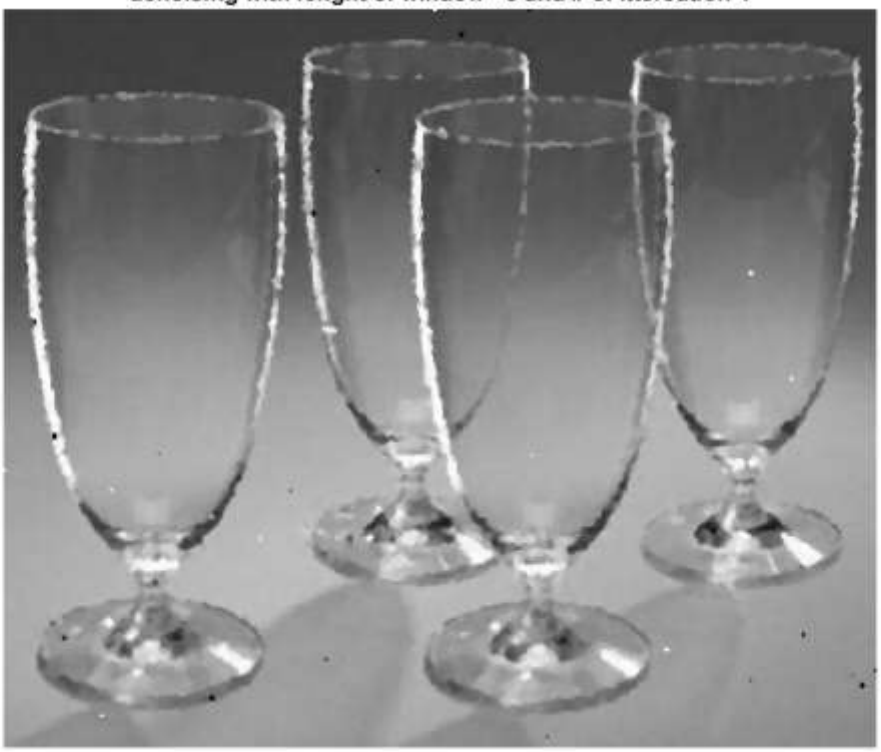
denoising with lenght of window =3 and # of ittereation 2



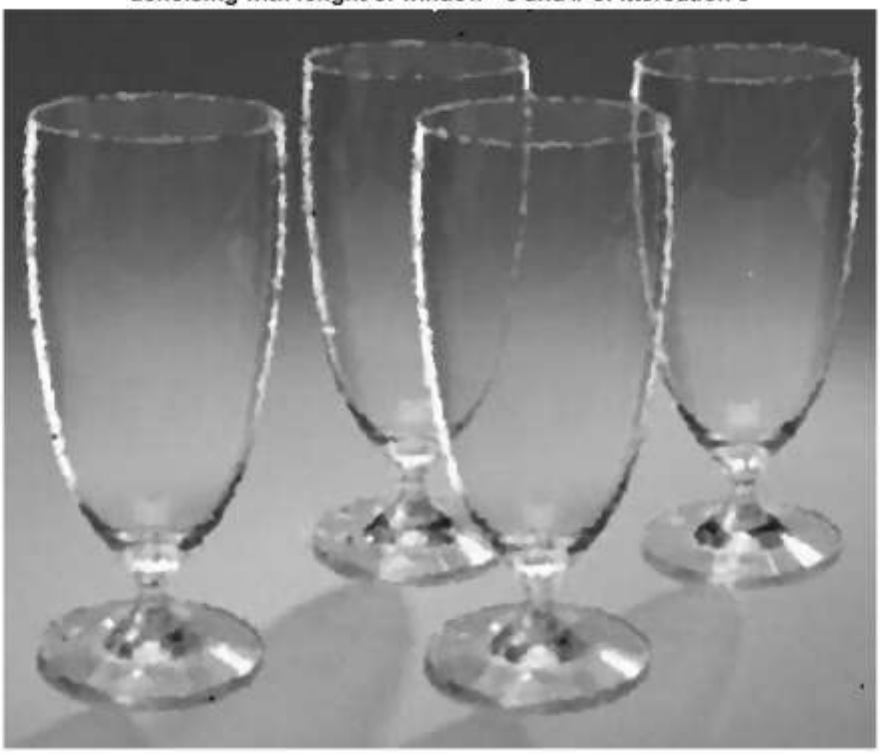
denoising with lenght of window =3 and # of ittereation 3



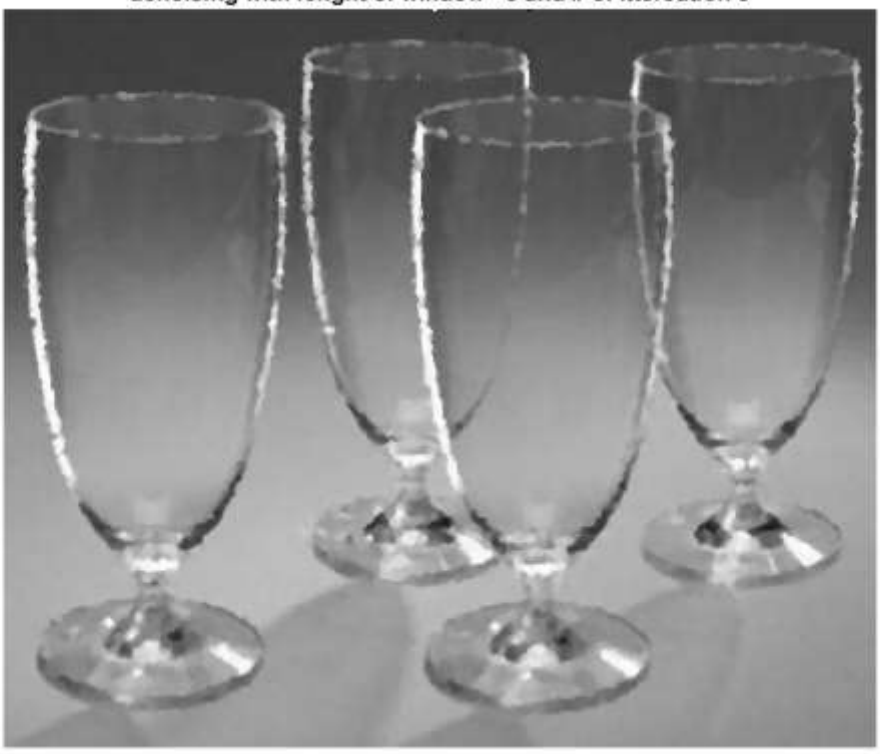
denoising with lenght of window =3 and # of ittereation 4



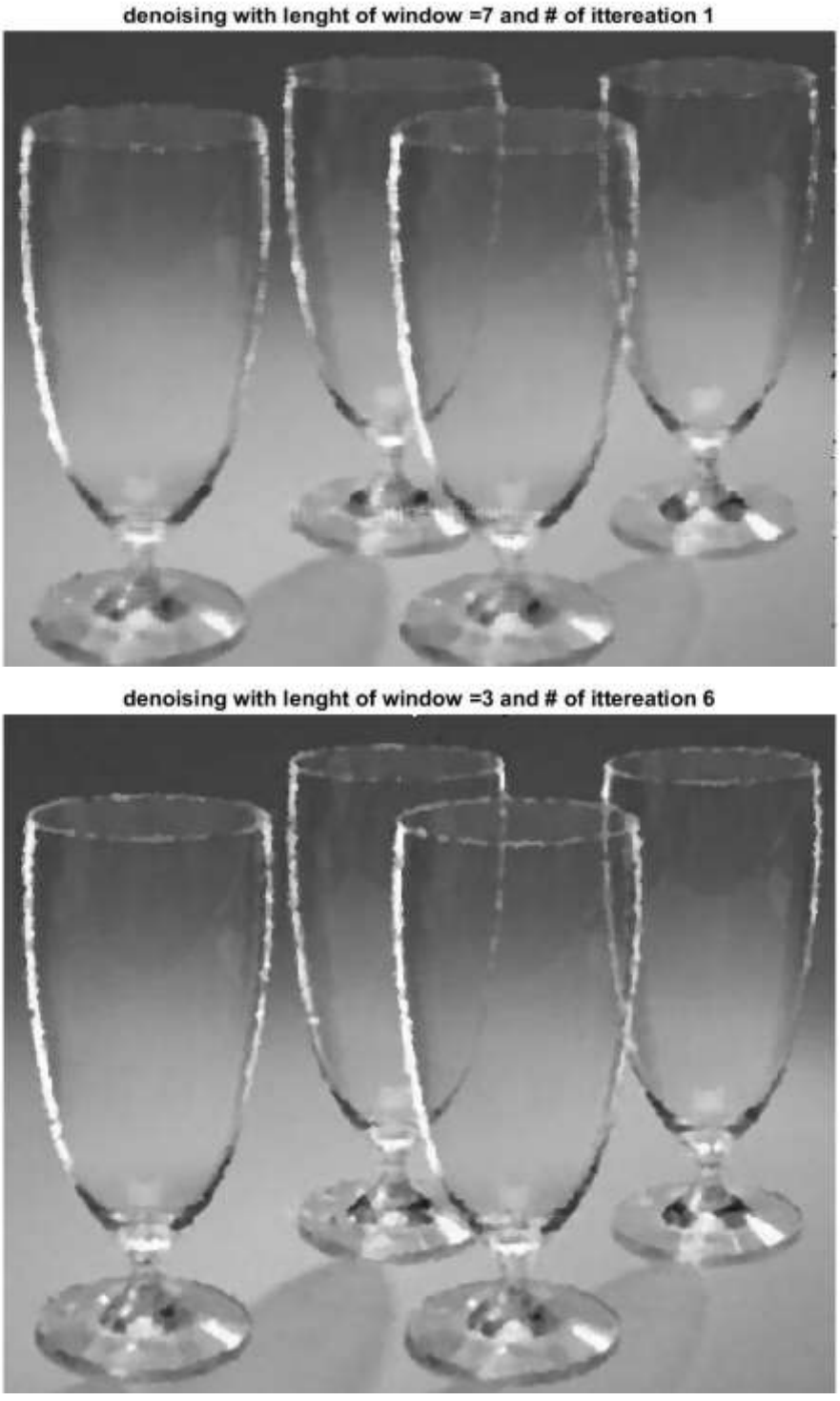
denoising with lenght of window =3 and # of ittereation 5



denoising with lenght of window =3 and # of ittereation 6



مشاهده می‌شود که با پنجره‌ی ۳ و با ۶ بار تکرار توانستیم نویزها را حذف نماییم. حال مسئله این‌جاست که آیا استفاده از پنجره با ابعاد کوچک ولی در عوض تکرارهای چندگانه بهتر است و یا بزرگ کردن ابعاد پنجره ولی با یک بار تکرار. برای پاسخ به این سؤال دو تصویر که نویز آن‌ها حذف شده را با هم مقایسه می‌نماییم. یکی پنجره با طول ۳ و ۶ بار تکرار و یکی پنجره با طول ۷ و ۱ بار تکرار:



کاملا واضح است که پنجره با طول ۳ و استفاده از۶ itteration نتیجه‌ی بهتری از پنجره با طول ۷ و ۱ بار iteration دارد. علت این امر آن است که با بزرگ کردن ابعاد پنجره، به جای هر پیکسل، پیکسلی می‌نشیند که correlation کم‌تری با آن دارد و از طرفی لبه‌ها نیز بلور می‌شوند. اما در استفاده از ittertion ، تنها لبه‌های باریک بلور می‌شوند که در این‌جا که ما ابعاد پنجره را ۳ در نظر گرفته‌ایم احتمال آن‌که پیکسل‌های درون پنجره مشابه باشند زیاد بوده و ممکن است لبه‌های باریک نیز بلور نشوند. لذا می‌توان به طور قطع گفت که استفاده از iteration و در عین حال پنجره با ابعاد کوچک‌تر نتیجه‌ی بهتری را ارائه می‌دهد.

ب) Adaptive Median Filter (کد این سؤال فایل HW5_Q4_B.m می‌باشد.)

همان‌طور که اشاره شد، در استفاده از فیلتر مدین برای حذف نویز نمک و فلفل، برای کل تصویر ابعاد پنجره از ابتدا تا انتها ثابت می‌باشد. اما بسیار اتفاق می‌افتد که در یک قسمت خاصی از تصویر، نویز به شدت است. مثلاً فرض کنید که ابعاد پنجره را ۳ در نظر بگیریم و در یک قسمت خاصی از تصویر در این پنجره ۵ پیکسل با روشنایی ۲۵۶ داشته باشیم. در این حالت اگر همانند روش قبل، پیکسل‌ها را sort نموده و مدین آن‌ها را بگیریم نتوانسته‌ایم نویز را حذف نماییم. چرا که مدین همان ۲۵۶ می‌شود. در این حالت باید در این قسمت خاص تصویر ابعاد پنجره بزرگ‌تر شود. بنابراین لازم است از روشی استفاده نماییم که ابعاد پنجره را در سراسر تصویر ثابت فرض ننموده و بسته به شرایط، ابعاد پنجره تغییر یابد. در روش adaptive median از این الگوریتم استفاده می‌نماییم. انتظار داریم نتیجه نسبت به حالت مدین فیلتر که ابعاد پنجره از ابتدا تا انتهای تصویر ثابت می‌باشد، بسیار بهبود یابد. برای این منظور از کد زیر استفاده می‌نماییم:

```
I = imread('glass2.jpg');
I = rgb2gray(I);

figure,imshow(I), title('Original Image');

%%%% Add salt and pepper noise
f = imnoise(I,'salt & pepper',0.4);
figure, imshow(f),title('Image with Salt and Pepper noise');

[M,N] = size(f);

temp_I = f;
W=9; %%% max length of Window
w=3; %%% initial value of w
l_w = floor(w/2);
l_W = floor(W/2);

p=0;
for m=l_W+1 : M - l_W
    p= p+1; q=0;
    for n=l_W+1 : N - l_W
        l_w = floor(w/2);
        q=q+1;
        ch =1;
        while ch == 1;
            ff = temp_I(m,n);

            temp = temp_I(m-l_w : m+l_w , n-l_w : n+l_w);
            sort_t = sort(temp(:));

            med_t = median(sort_t);
            min_t = min(sort_t);
            max_t = max(sort_t);

            if min_t< med_t && max_t> med_t
                if min_t< ff && max_t> ff
                    g(p,q) = temp_I(m,n);
```

```
                    ch=0;
                else
                    g(p,q) = med_t;
                    ch=0;
                end
            elseif w<W
                w= w+2;
                l_w = floor(w/2);
                ch=1;
            else
                g(p,q) = ff;
                ch=0;
            end
        end

    end
end

gcf = figure;
imshow(g),title('denoising with Adaptive Median Filter method')
```

توجه شود که نقطه‌ی شروع پنجره را ۳ در نظر گرفته و حد بالای پنجره را ۹ در نظر گرفته‌ایم. این موضوع را از نتایج قسمت اول سؤال دریافته‌ایم. چرا که در قسمت اول سؤال در یک بار تکرار، از ابعاد ۹ به بعد نتایج به شدت بد بود و بلور شد و لذا ماکزیمم مقدار پنجره را نیز در این قسمت ۹ فرض کردیم. همچنین مینییم حالت پنجره هم ک مشاهده کردیم ۳ بود که در اینجا آن را لحاظ نمودیم.

نتایج حاصل از شبیه‌سازی به‌صورت زیر خواهد بود:

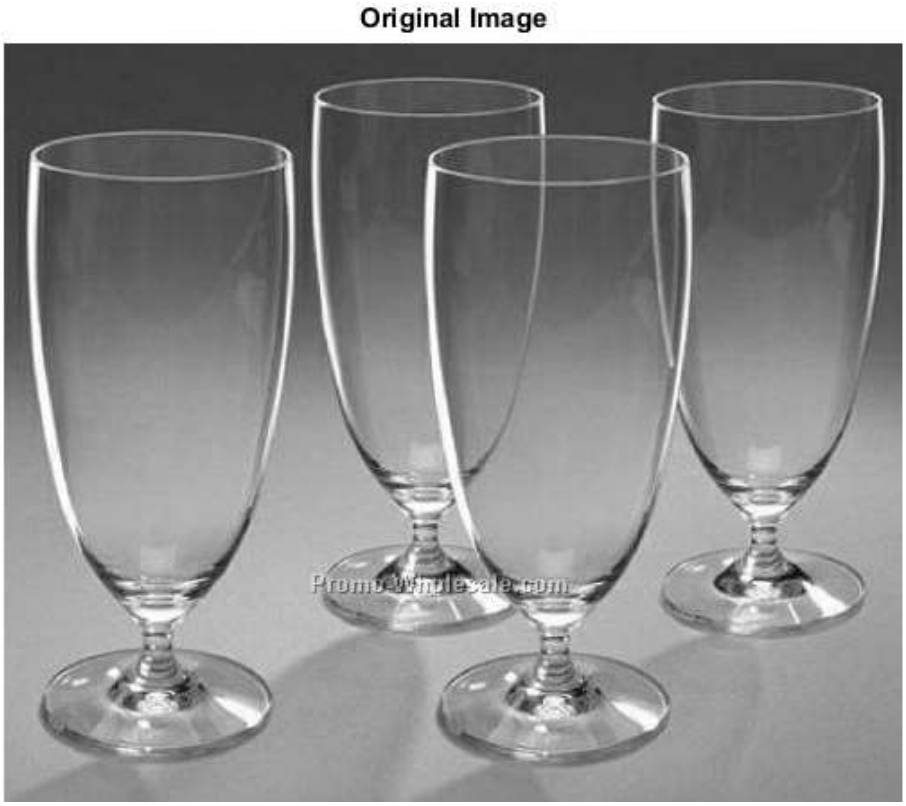
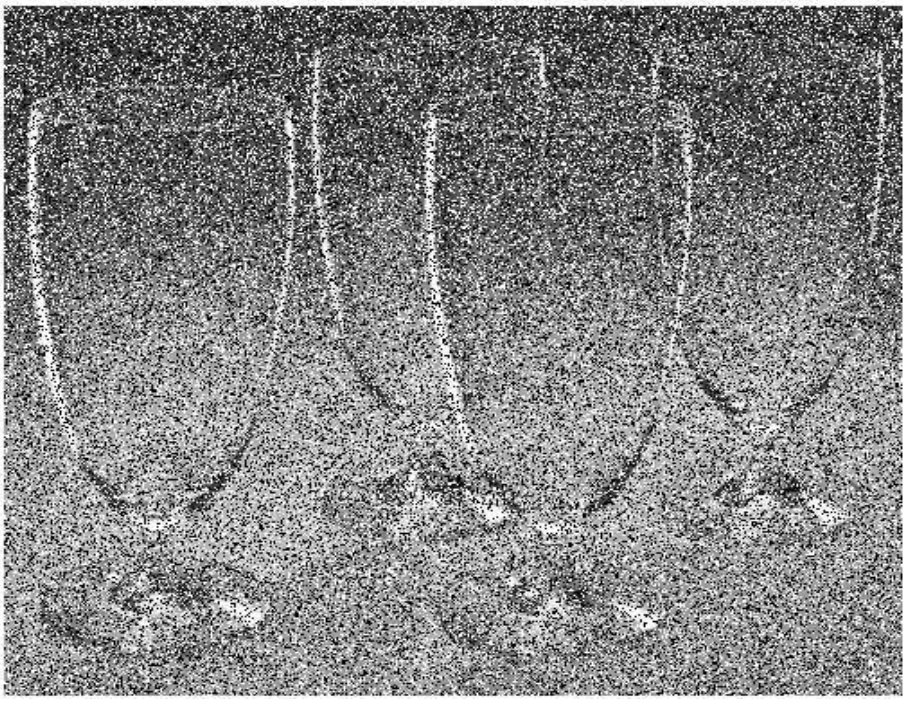


Image with Salt and Pepper noise



denoising with Adaptive Median Filter method

همان‌طور که مشاهده می‌نماییم، نتیجه‌ی حاصل از adaptive median filter بسیار مناسب بوده و هم توانسته نویز را حذف نماید و هم لبه‌ها و گوشه‌ها را به طرز مطلوبی حفظ نماید. علت این امر هم همان مسئله‌ی تغییر ابعاد پنجره در مکان‌هایی از تصویر است که نویز شدید می‌باشد.