" بسمه تعالی "

گزارش چهارم آزمایشگاه DSP – زهرا لطیفی – ۹۹۲۳۰۶۹

بخش ۴-۱-الف)

```
pic = imread("lena.bmp");
imshow(pic);
title("Original Image");
```

در این بخش، تصویر lena را با دستور imread خوانده و با دستور imshow نمایش دادیم.

شکل ۱

بخش ۴-۱-ب)

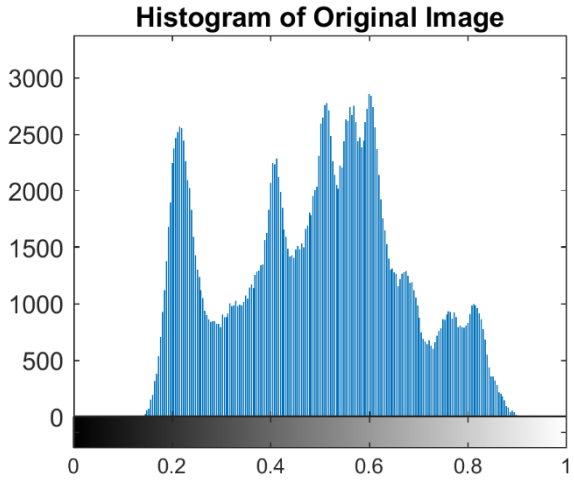
```
pic_d = im2double(pic);
imshow(pic_d);
title("Original Image - Double");
```

سپس با استفاده از دستور double2im تصویر را به نوع double تبدیل کردیم تا محاسبات به درستی انجام شوند.

بخش ۴-۱-ج)

```
imhist(pic_d);
title("Histogram of Original Image");
```

هیستوگرام این تصویر را با دستور `imhist` رسم کردیم:

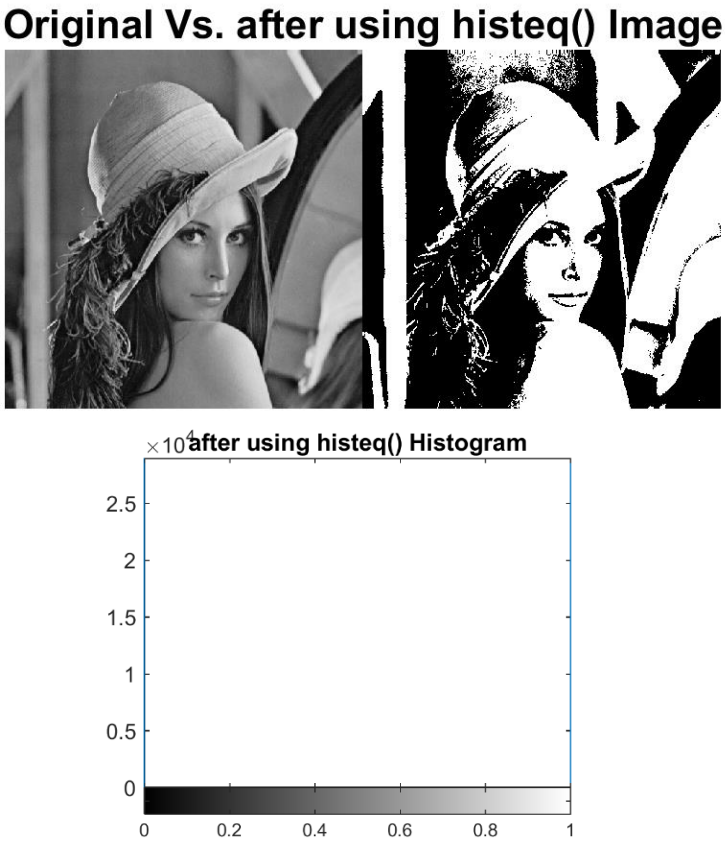


شکل ۲

بخش ۴-۱-د و ه)

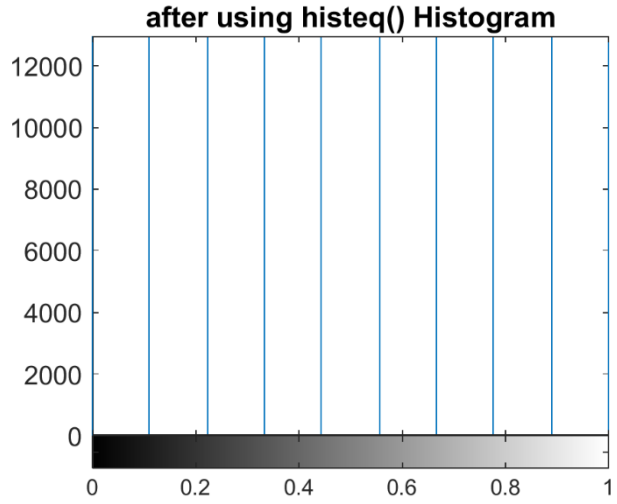
```
J = histeq(pic_d, 2);
imshowpair(pic_d, J,'montage');
title("Original Vs. after using histeq() Image");
```

با دستور histeq سعی کردیم با انجام equalization بر هیستوگرام این تصویر gray scale، contrast تصویر را بهبود ببخشیم. مثلا با اعمال ضریب ۲ خواهیم داشت:



شکل ۳

و با اعمال ضریب ۱۰ خواهیم داشت:



شکل ۴

با اعمال یک تصویر به صورت gray scale، «histeq» هیستوگرام آن را محاسبه می‌کند. سپس مقادیر پیکسل‌ها را طوری تغییر می‌دهد که هیستوگرام حاصل تقریباً مسطح می‌شود. (توزیع یکنواخت) درواقع هدف این است که سطح شدت را در کل محدوده پخش کرده و نقاط تاریک را روشن‌تر و نقاط روشن را تاریک‌تر کند. "J = histeq(I, n)" تصویر "I" را تغییر می‌دهد به طوری که هیستوگرام تصویر خروجی "J" دارای n bin و تقریباً مسطح است. با استفاده از «n» تعداد binها را مشخص می‌کنیم. حتی می‌توانیم هیستوگرام هدف طراحی کرده و به عنوان ورودی «hgram» به تابع بدهیم.

چرا از equalizer هیستوگرام استفاده می‌کنیم؟

برای افزایش کنتراست؛ با گسترش مقادیر شدت، جزئیات را در یک تصویر بهتر می‌کند و تصاویر با نوردهی ضعیف را بهبود می‌بخشد. در تصویربرداری پزشکی، در اشعه ایکس، ام آر آی یا سی تی اسکن، featureهایی مانند استخوان‌ها یا تومورها را شدت می‌دهد. در Computer Vision، برای تشخیص اشیا، تشخیص لبه و استخراج feature استفاده دارد. در تصاویر ماهواره ای هم featureهای زمین، پوشش گیاهی و آب را شدت می‌دهد.

چرا ممکن است هیستوگرام بدست آمده پس از استفاده از histeq به صورت یکنواخت در نیاید؟

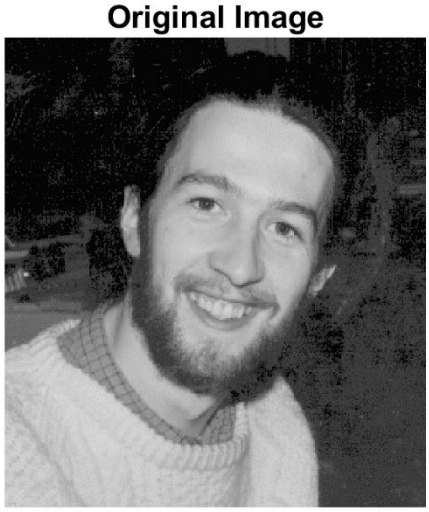
گفتیم که هدف از hesteq تولید یک هیستوگرام کاملاً یکنواخت نیست، بلکه افزایش کنتراست تصویر است. حتی اگر هیستوگرام کاملاً یکنواخت نباشد، کنتراست تصویر همچنان می‌تواند به میزان قابل توجهی بهبود یابد.

هیستوگرام حاصل پس از اعمال «histeq» ممکن است به دلایل مختلفی کاملاً یکنواخت نباشد. مثلا تصاویر دیجیتال دارای مقادیر پیکسل مجزا هستند. (مثلاً در یک تصویر ۸ بیتی، مقادیر پیکسل از ۰ تا ۲۵۵ متغیر است.) این گسستگی می‌تواند منجر به ایجاد binهایی در هیستوگرام شود که ممکن است به طور کامل پر نشوند و باعث شود هیستوگرام غیریکنواخت به نظر برسد. یا اینکه به طور پیش‌فرض، «histeq» سعی می‌کند یک هیستوگرام مسطح را با ۶۴ bin تشکیل دهد. اگر تصویر بیش از ۶۴ مقدار پیکسل منحصربه‌فرد داشته باشد، برخی از binها به ناچار پیکسل‌های بیشتری نسبت به بقیه دارند که منجر به یک هیستوگرام غیریکنواخت می‌شود. از طرفی تابع «histeq» هیستوگرام کاملاً یکنواخت را تضمین نمی‌کند. این الگوریتم با نگاشت تابع توزیع تجمعی (CDF) شدت پیکسل در تصویر ورودی به CDF مورد نظر (معمولاً توزیع یکنواخت برای یکسان سازی هیستوگرام) کار می‌کند. این نگاشت همیشه کامل نیست، به خصوص زمانی که هیستوگرام اصلی دارای قله‌ها یا دره‌های بزرگ باشد.

بخش ۴-۲-الف)

```
pic = im2double(imread("Image02.jpg"));
imshow(pic);
title("Original Image");
```

در این بخش تصویر Image02 را بارگذاری کرده و به double تبدیل کردیم.



شکل ۵

بخش ۴-۲-ب)

```
pic_n = imnoise(rgb2gray(pic), 'gaussian', 0, 0.04);
imshow(pic_n);
title("Noisy Image");
```

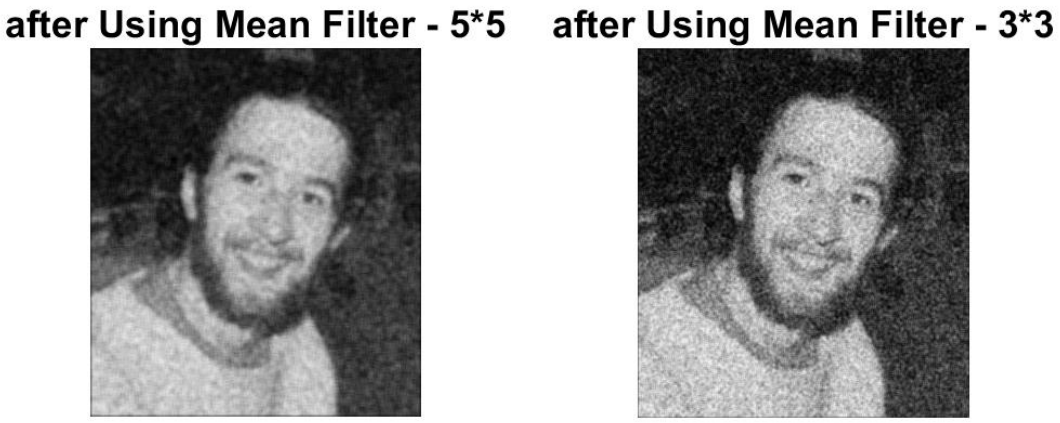
سپس نویز گوسی با میانگین صفر و انحراف معیار ۰.۲ به آن اعمال کردیم.

شکل ۶

بخش ۴-۲-ج و د)

```
mean3 = (1/9)*ones(3,3);
pic_dn3 = imfilter(pic_n , mean3);
imshow(pic_dn3);
title("after Using Mean Filter - 3*3");

mean5 = (1/25)*ones(5,5);
pic_dn5 = imfilter(pic_n , mean5);
imshow(pic_dn5);
title("after Using Mean Filter - 5*5");
```

فیلتر میانگین‌گیر ۳در۳ و سپس ۵در۵ را با دستور ones ایجاد کرده، به تصویر نویزی اعمال کرده و سپس خروجی‌ها را رسم کردیم:



شکل ۷

شاهدیم که از میزان نویز تصویر کاسته می‌شود اما با بزرگتر شدن پنجره، علاوه بر کم شدن نویز، از وضوح حواشی، لبه‌ها و جزئیات تصویر اصلی هم کاسته شده است. که این یعنی بین این دو مورد بده بستان داریم.

بخش ۴-۲-ه)

```
pic_n2 = imnoise(rgb2gray(pic), "salt & pepper", 0.1);
imshow(pic_n2);
title("Noisy Image - S&P");
```

این بار به همان تصویر نویز salt & pepper با p=0.1 اضافه کردیم.

شکل ۸

بخش ۴-۲-و)

```
mean3 = (1/9)*ones(3,3);
pic_dn2 = imfilter(pic_n2 , mean3);
imshow(pic_dn2);
title("after Using Mean Filter - 3*3");
```

فیلتر میانگین‌گیر ۳در۳ را به آن اعمال کردیم و خروجی را نمایش دادیم:

شکل ۹

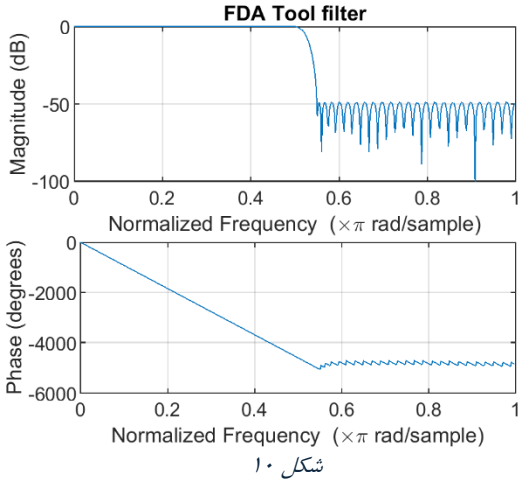
اما این بار فیلتر میانگین‌گیر نتوانست به خوبی در کاهش نویز نمک و فلفل مفید باشد. چرا؟

نویز گاوسی نویز تصادفی است که مقادیر آن به طور نرمال حول میانگین مقدار شدت پیکسل‌های تصویر توزیع شده است. از آنجایی که فیلتر میانگین مقادیر را میانگین‌گیری می‌کند، به طور طبیعی تغییراتی را که در نویز گاوسی معمول است کاهش می‌دهد. اما نویز نمک و فلفل با تغییرات تیز و ناگهانی در تصویر ظاهر می‌شود. به این صورت که به صورت تصادفی پیکسل‌های سفید و سیاه در تصویر ظاهر می‌کند. این نوع نویز توسط فیلتر میانگین به خوبی حذف نمی شود زیرا میانگین مقادیر پیکسل‌ها در حضور چنین نقاط شارپی به طور موثر نویز را کاهش نمی‌دهد و تنها می‌تواند باعث پخش شدن نویز شود چون مقادیر sharp نویز نمک و فلفل به طور قابل توجهی بر محاسبه میانگین تأثیر می‌گذارند.

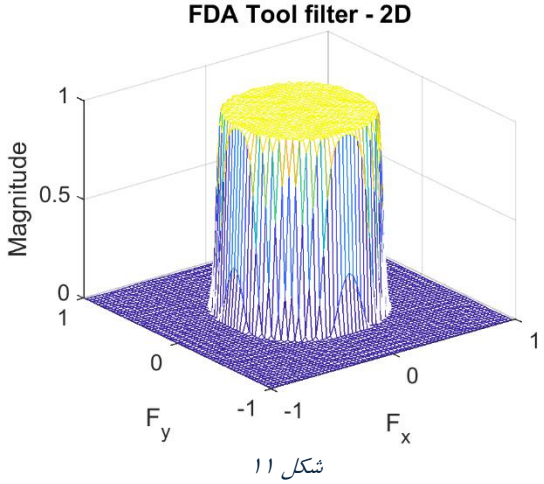
بخش ۴-۲-ز)

```
freqz(Num);
title("FDA Tool filter");

filt2 = ftrans2(Num);
freqz2(filt2);
title("FDA Tool filter - 2D");
```

در این بخش با استفاده از fdatool یک فیلتر پایین گذر FIR طراحی کردیم که فرکانس عبور آن حدود فرکانس نرمالیزه 0.5 بود. سپس با استفاده از دستور ftrans2 فیلتر یک بعدی طراحی شده را به یک فیلتر دو بعدی تبدیل کردیم. سپس با استفاده از دو دستور freqz و freqz2 پاسخ فوریه فیلترهای یک بعدی و دو بعدی طراحی شده را رسم کردیم:



شکل ۱۰

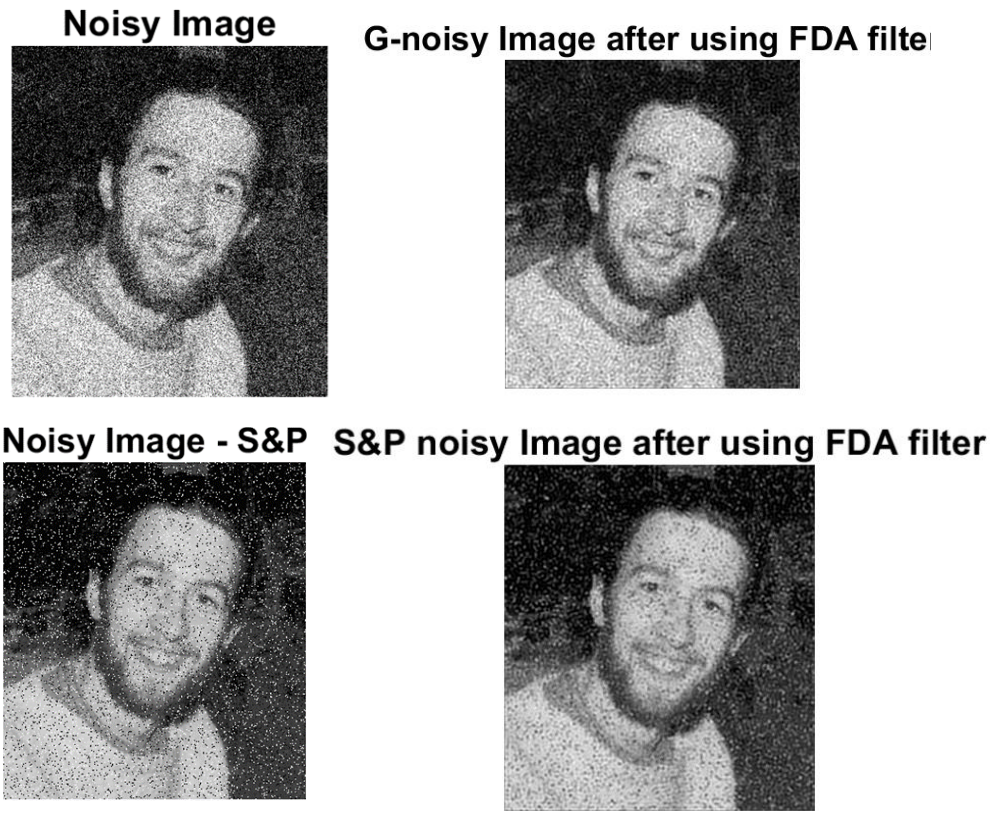


شکل ۱۱

بخش ۴-۲-ح)

این فیلتر را به دو تصویر حاوی نویز اعمال کردیم.

```
pic_dnfG = imfilter(pic_n, filt2);
imshow(pic_dnfG);
title("G-noisy Image after using FDA filter");

pic_dnfSP = imfilter(pic_n2, filt2);
imshow(pic_dnfSP);
title("S&P noisy Image after using FDA filter");
```



شکل ۱۲

هردو کمی بهبود پیدا کردند اما تاثیر این فیلتر روی هیچ کدام خیلی قابل توجه نبود.

بخش ۴-۲-ط و ک و ل)

```
med_filt = medfilt2(rgb2gray(pic_n2));
imshow(med_filt);
title("after Using Median Filter");
```

در دستورکار خواسته شده بود که تابع میانه را خودمان پیاده‌سازی کنیم اما با استفاده از تابع آماده `medfilt2` این بخش را انجام دادیم و شاهد عماکرد بسیار خوب این فیلتر در حذف نویز نمک و فلفل از تصویر بودیم. چرا؟ یک فیلتر میانه که مقدار هر پیکسل را با مقدار وسط پیکسل‌های مجاور جایگزین می‌کند، برای نویز نمک و فلفل موثرتر است. زیرا می‌تواند تغییرات sharp را بهتر مدیریت کند زیرا میانه نسبت به میانگین حساسیت کمتری نسبت به نقاط پرت دارد. بنابراین، در حالی که یک فیلتر میانگین‌گیر نویز را محو می‌کند، فیلتر میانه تقریبا آن را بدون کاهش وضوح تصویر (blurring) حذف می‌کند.

شکل ۱۳

آیا فیلتر میانه بدی‌هایی نیز دارد؟

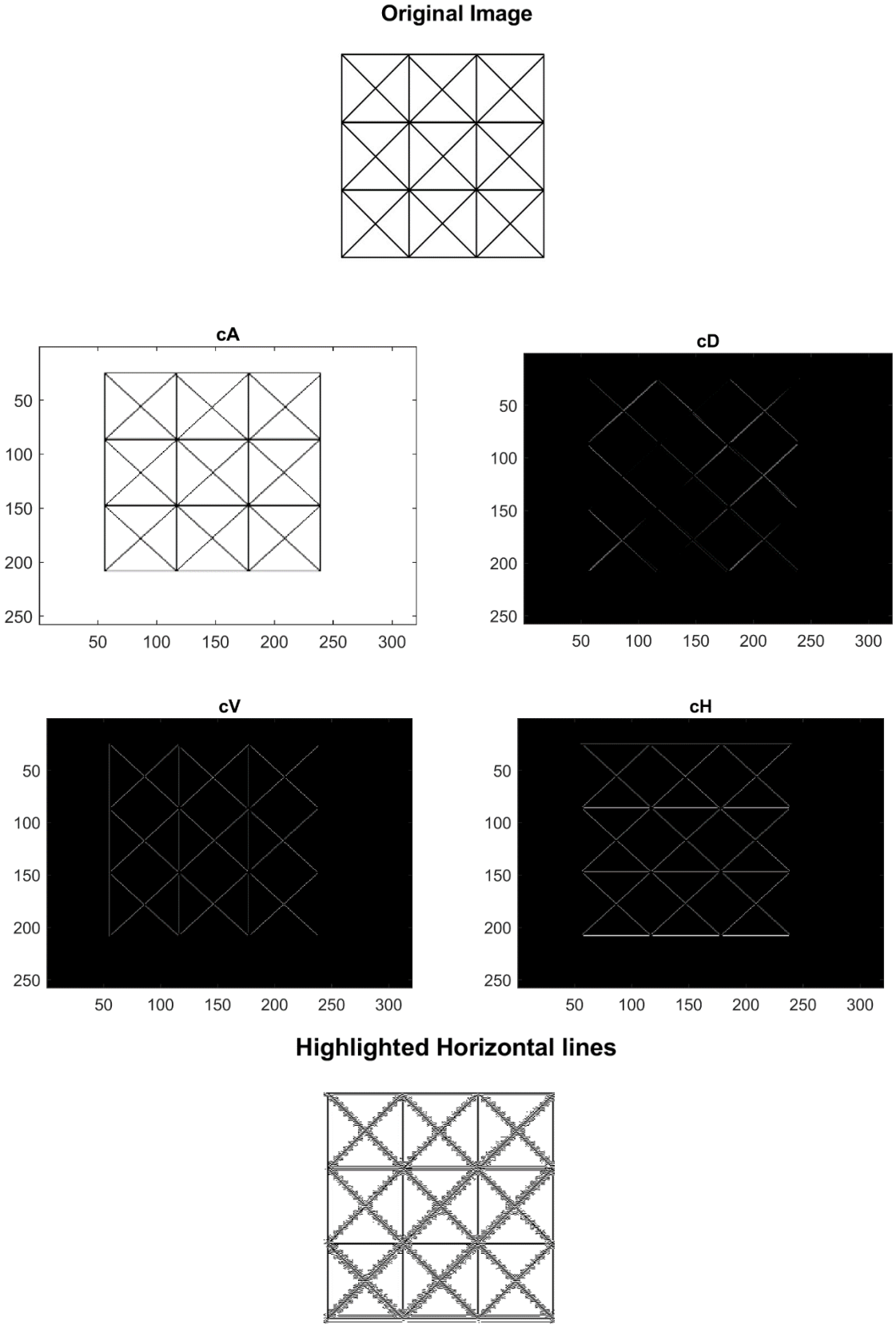
بله برای مثال فیلتر میانه گاهی اوقات می‌تواند جزئیات دقیق تصویر را همراه با نویز حذف کند، به خصوص اگر این جزئیات در تمام پیکسل‌های همسایه یکسان نباشد. فیلتر میانه با همه پیکسل‌ها به طور یکسان برخورد می‌کند، خواه نویز باشند یا محتوای واقعی تصویر. این مسئله می‌تواند بر کیفیت تصویر تأثیر منفی بگذارد.

بخش ۴-۳-الف و ب)

```
pic = im2double(imread("square2.jpg"));
imshow(pic);
title("Original Image");

[cA, cH, cV, cD] = dwt2(pic, 'db1');
imagesc(cA);
title("cA");

imagesc(cD);
title("cD");

imagesc(cH);
title("cH");

imagesc(cV);
title("cV");

imshow(idwt2(cA, 1000*cH, cV, cD, 'db1'));
title("Highlighted Horizontal lines");
```

در این بخش ابتدا تصویر square2 را بارگذاری کرده و نمایش دادیم. سپس تبدیل موجک دو بعدی را با دستور dwt2 و از نوع db1 به آن اعمال کردیم. تبدیل موجک گسسته دو بعدی (DWT2) داده ورودی را با استفاده از موجک db1 محاسبه می‌کند و ماتریس ضرایب تقریبی cA و ضرایب ماتریس‌های جزئیات cH، cV، و cD (به ترتیب افقی، عمودی و مورب) را

برمی‌گرداند. می‌بینیم که cA شامل تمام خطوط تصویر، cD تنها خطوط مورب، cV خطوط عمودی و cH خطوط افقی تصویر است. خواسته شده خطوط افقی را برروی تصویر اصلی هایلایت کنیم. پس ضرایب cH را تقویت کرده و idwt2 می‌گیریم.

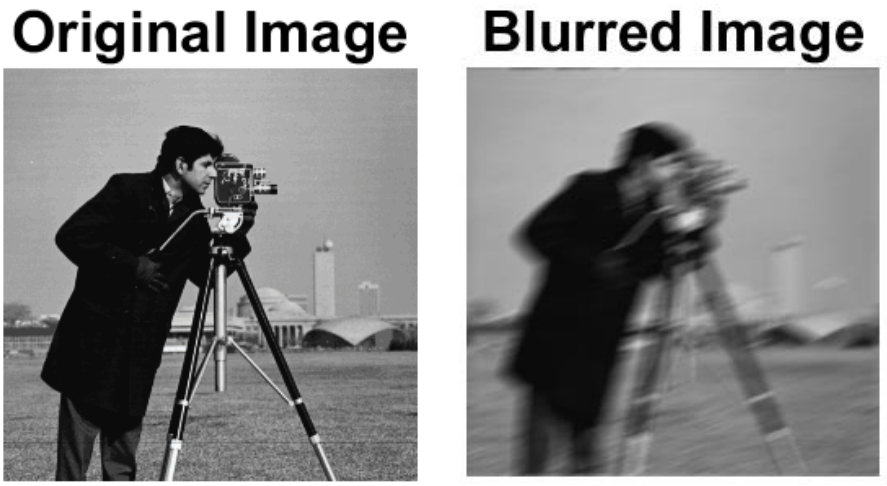


شکل ۱۴

بخش ۴-۴-الف)

```
pic = im2double(imread("Image04.png"));
imshow(pic);
title("Original Image");

motion = fspecial('motion', 15, 20);
blurred = imfilter(pic, motion, 'circular');
imshow(blurred);
title("Blurred Image");
```

در این بخش تصویر Image04 را بارگذاری کرده و یک تاری به اندازه ۱۵ و با زاویه ۲۰ درجه به آن اعمال کردیم:



شکل ۱۵

بخش ۴-۴-ب)

```
wnr = deconvwnr(blurred, motion, 0.001);
imshow(wnr);
title('Restored Blurred Image');
```

تصویر تار شده را با فیلتر wiener اصلاح کردیم. برای این کار از تابع deconvwnr استفاده کردیم. پارامتر NSR را ذره ذره از ۰ تغییر دادیم تا به یک نتیجه منطقی برسیم. نهایتا در مقدار (۰.۰۰۱) به این نتیجه رسیدیم.

شکل ۱۶

بخش ۴-۴-ج)

```
blurred_n = imnoise(rgb2gray(blurred), 'gaussian', 0, 0.01);
imshow(blurred_n);
title("Blurred Noisy Image");
```

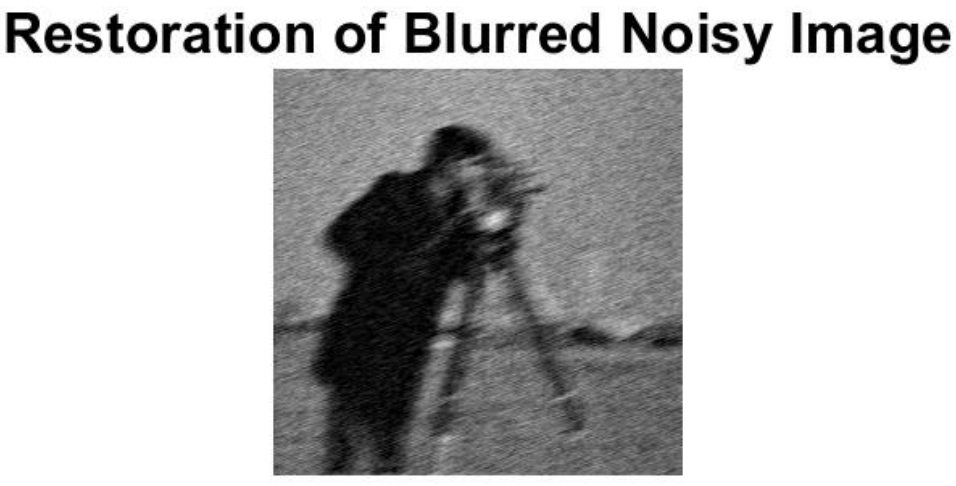
حال یک نویز گوسی با میانگین صفر و واریانس ۰.۰۱ به تصویر تار اضافه کردیم و آن را نمایش دادیم:

شکل ۱۷

بخش ۴-۴-د)

```
estimated_nsr = 0.01 / var(im2double(pic(:)));
wnr2 = deconvwnr(blurred_n, motion, estimated_nsr);
imshow(wnr2);
title('Restoration of Blurred Noisy Image');
```

برای محاسبه NSR در این بخش واریانس نویز گوسی را بر واریانس تصویر اصلی تقسیم کردیم و بخش قبل را تکرار کردیم:

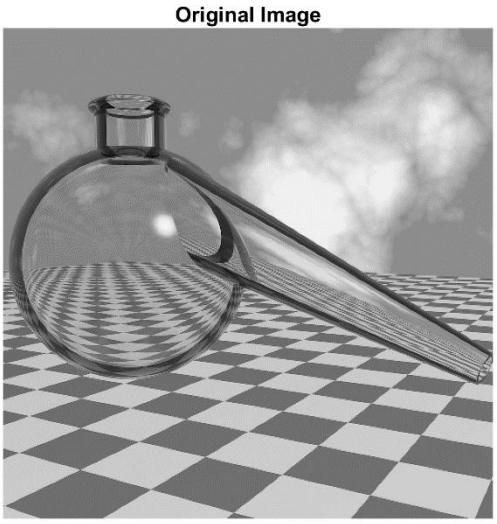


شکل ۱۸

بخش ۴-۵-الف)

```
pic = im2double(imread("glass.tif"));
imshow(pic);
title("Original Image");
```

در این بخش تصویر glass را بارگذاری کردیم و نمایش دادیم.

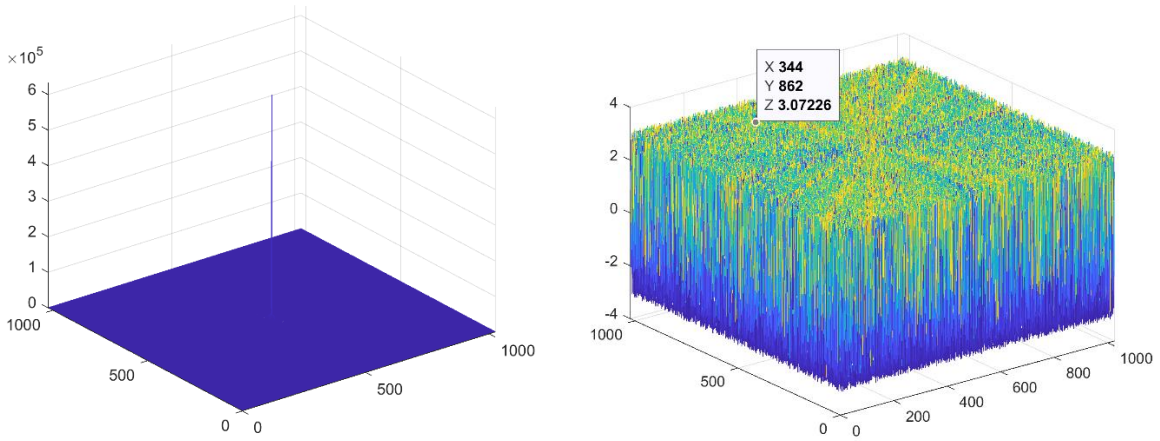


شکل ۱۹

بخش ۴-۵-ب)

```
pic_fft2 = fft2(pic);
mesh(abs(fftshift(pic_fft2)));

figure();
mesh(angle(fftshift(pic_fft2)));
```

DFT دو بعدی تصویر را حساب کرده و دامنه و فازش را رسم کردیم:



شکل ۲۰

fftshift یک تبدیل فوریه را با جابجایی مولفه فرکانس صفر آن به مرکز آرایه به جای اول آن بازآرایی می‌کند. اگر تبدیل فوریه یک بعدی و یک بردار باشد، fftshift نیمه چپ و راست آن بردار را تعویض می‌کند. اگر دو بعدی و یک ماتریس باشد، fftshift ربع (quadrant) اول ورودی را با ربع سوم و ربع دوم را با چهارم تعویض می‌کند. این تابع بیشتر زمانی مفید است که بخواهیم پس از تبدیل فوریه اجزای فرکانس یک سیگنال را به صورت متقارن در اطراف فرکانس صفر ترسیم کنیم.

بخش ۴-۵-ج)

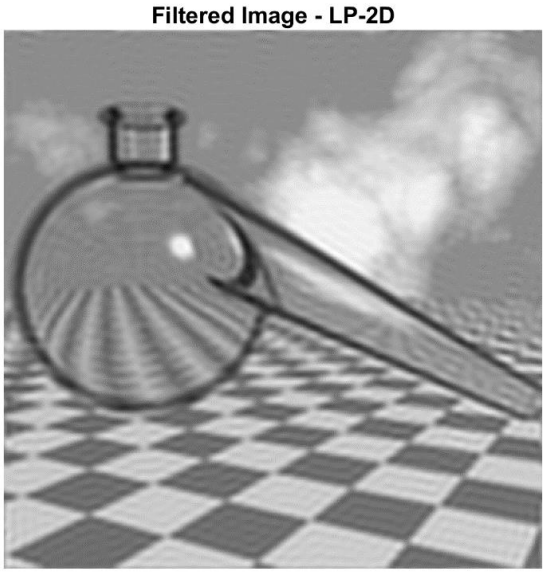
```
function Output_image = FFT_LP_2D(input_image, cutoff_frequency)

    fft_image = fft2(input_image);
    fft_shifted = fftshift(fft_image);

    [rows, cols] = size(input_image);
    center_row = rows / 2;
    center_col = cols / 2;

    % low-pass filter
    filter_mask = zeros(rows, cols);
    for i = 1:rows
        for j = 1:cols
            distance = sqrt((i - center_row)^2 + (j - center_col)^2);
            if ((distance/center_row)*pi) <= cutoff_frequency
                filter_mask(i, j) = 1;
            end
        end
    end


    filtered_fft = fft_shifted .* filter_mask;
    ifft_shifted = ifftshift(filtered_fft);

    % 2D IFFT
    Output_image = ifft2(ifft_shifted, 'symmetric');
end
```

تابع فیلتر پایین گذر دو بعدی را در این بخش پیاده‌سازی کردیم. این تابع تصویر و فرکانس قطع را به عنوان ورودی می‌گیرد. fft دو بعدی تصویر را گرفته و با fftshift آن را شیفت می‌دهد. سپس مختصات مرکز تصویر را پیدا می‌کند. ماسکی به اندازه ابعاد تصویر ورودی ساخته و با پیمایش تصویر فاصله هر پیکسل تا مرکز تصویر را محاسبه می‌کنیم. اگر فاصله نرمالایز شده از فرکانس قطع کمتر بود، مقدار ماسک در آن نقاط برابر یک خواهد بود و در غیر این صورت برابر صفر. نهایتا ماسک را در fft شیفت یافته تصویر ورودی ضرب کردیم تا فیلتر دایروی ما اعمال شود. نهایتا برای ساختن خروجی، ifft2 متقارن گرفتیم.

بخش ۴-۵-د)

```
result = FFT_LP_2D(pic, 0.1*pi);
imshow(result);
title("Filtered Image - LP-2D");
```

در این بخش تصویر را با فرکانس قطع 0.1π به تابع تعریف شده در بخش قبل دادیم و نتیجه را نمایش دادیم:

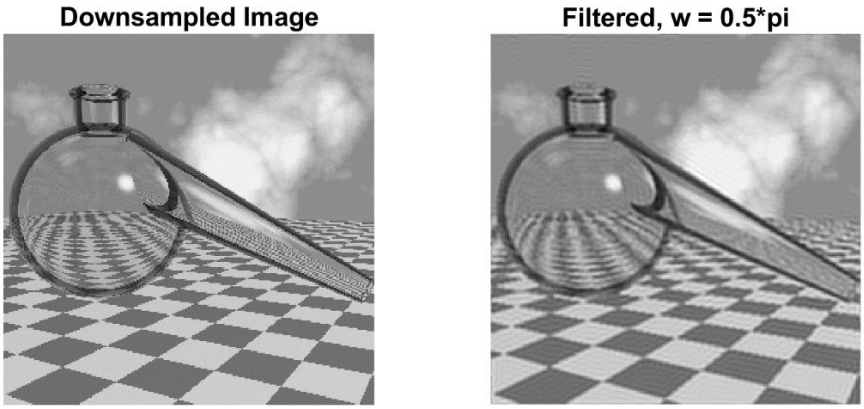


شکل ۲۱

بخش ۴-۵-ه)

```
dwn_pic = downsample(downsample(pic, 4)', 4)';
figure();
subplot(1,2,1);
imshow(dwn_pic);
title("Downsampled Image");


dwn_pic_fil = FFT_LP_2D(dwn_pic, 0.5*pi);
subplot(1,2,2);
imshow(dwn_pic_fil);
title("Filtered, w = 0.5*pi");
```

از تابع downsample استفاده کردیم و ابعاد این تصویر را به ۰.۲۵ تصویر اصلی کاهش دادیم. خروجی را نمایش دادیم و تاثیر aliasing را در تصویر downsampled مشاهده کردیم. aliasing گوشه‌های تصویر یا قسمت‌های راه راه را خراب کرده. از فیلتری که تعریف کردیم استفاده کرده و تصویر را به اندازه کافی تغییر دادیم تا اثرات aliasing را از تصویر downsampled، حذف کردیم.



شکل ۲۲