دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

عنوان: تکلیف دوم درس مبانی بینایی کامپیوتر

نام و نام خانوادگی: علیرضا ابره فروش

شماره دانشجویی: ۹۸۱۶۶۰۳

نیم سال تحصیلی: بهار ۱۴۰۰/۱۴۰۱

مدرّس: دکتر نادر کریمی

دستیاران آموزشی: بهنام ساعدی - محمدرضا مزروعی

۱

ابتدا متغیرهای زیر را تعریف می‌کنیم.
H و W: ابعاد تصویر
I: تصویر اصلی
J: تصویر مورد بررسی
MAX_I: بزرگ‌ترین سطح روشنایی محتمل در تصویر (مقدار پیش‌فرض ۲۵۵)
MIN_I: کوچک‌ترین سطح روشنایی محتمل در تصویر (مقدار پیش‌فرض ۰)

MAE ۱.۱

حداکثر ۱.۱.۱

حالتی را تصور می‌کنیم که اختلاف سطح روشنایی هر دو پیکسل متناظر در تصویر حداکثر باشد (تصویر تمام سیاه و تمام سفید). بنابراین به ازای هر i و j دامنه، $I(i,j)$ برابر MAX_I و $J(i,j)$ برابر MIN_J یا بالعکس. پس داریم:

$$\max(\frac{1}{H.W}\sum_{i=1}^{H}\sum_{j=1}^{W}|I(i,j)-J(i,j)|) = \frac{1}{H.W}\times H.W\times|MAX_I-MIN_J| = |MAX_I-MIN_J|$$

که در حالت پیش‌فرض این مقدار برابر ۲۵۵ است.

حداقل ۲.۱.۱

حالتی را تصور می‌کنیم که اختلاف سطح روشنایی هر دو پیکسل متناظر در تصویر حداقل باشد (تصاویر برابر باشند). بنابراین به ازای هر i و j دامنه، $I(i,j)$ برابر $J(i,j)$ است. پس داریم:

$$\min(\frac{1}{H.W}\sum_{i=1}^{H}\sum_{j=1}^{W}|I(i,j)-J(i,j)|) = \frac{1}{H.W}\times H.W\times|I(i,j)-I(i,j)| = 0$$

MSE ۲.۱

حداکثر ۱.۲.۱

مشابه قسمت قبل، حالتی را تصور می‌کنیم که اختلاف سطح روشنایی هر دو پیکسل متناظر در تصویر حداکثر باشد (تصویر تمام سیاه و تمام سفید). بنابراین به ازای هر i و j دامنه، $I(i,j)$ برابر MAX_I و $J(i,j)$ برابر MIN_J یا بالعکس. پس داریم:

$$\max(\frac{1}{H.W}\sum_{i=1}^{H}\sum_{j=1}^{W}(I(i,j)-J(i,j))^2) = \frac{1}{H.W}\times H.W\times(MAX_I-MIN_J)^2 = (MAX_I-MIN_J)^2$$

که در حالت پیش‌فرض این مقدار برابر ۶۵۰۲۵ است.

حداقل ۲.۲.۱

مشابه قسمت قبل، حالتی را تصور می‌کنیم که اختلاف سطح روشنایی هر دو پیکسل متناظر در تصویر حداقل باشد (تصاویر برابر باشند). بنابراین به ازای هر i و j دامنه، $I(i,j)$ برابر $J(i,j)$ است. پس داریم:

$$\min(\frac{1}{H.W}\sum_{i=1}^{H}\sum_{j=1}^{W}(I(i,j)-J(i,j))^2) = \frac{1}{H.W}\times H.W\times(I(i,j)-I(i,j))^2 = 0$$

PSNR ۳.۱

برای بیشینه (کمینه) کردن تابعِ $10\log_{10}(\frac{MAX_I^2}{MSE})$ کافیست تابعِ $\frac{MAX_I^2}{MSE}$ بیشینه (کمینه) کنیم.

۱.۳.۱ حداکثر

با فرض ناصفر بودن مقدار MAX_I داریم:

$$\max(10\log_{10}(\frac{MAX_I^2}{MSE})) = \lim_{MSE\to 0}(10\log_{10}(\frac{MAX_I^2}{MSE})) = \infty$$

پس به ازای دو تصویر با MSE نزدیک به صفر (دو تصویر تقریبا برابر)، مقدار PSNR به بی‌نهایت میل می‌کند.

۲.۳.۱ حداقل

اگر I تصویر تمام سیاه (MAX_I ناچیز باشد)، و J تصویر غیر تمام سیاه باشد، آنگاه چون MAX_I تقریبا صفر است داریم:

$$\min(10\log_{10}(\frac{MAX_I^2}{MSE})) = \lim_{MAX_I\to 0}(10\log_{10}(\frac{MAX_I^2}{MSE})) = -\infty$$

۲

سطح روشنایی هر پیکسل در تصویر خاکستری‌گونه از MIN_I (پیش‌فرض ۰) تا MAX_I (پیش‌فرض ۲۵۵) متغیر است. بدترین MSE که در واقع بزرگ‌ترین MSE است زمانی رخ می‌دهد که اختلاف سطح روشنایی هر دو پیکسل متناظر در تصاویر ماکسیمم باشد. در نتیجه برای ساخت چنین تصویری فاصله‌ی سطح روشنایی هر پیکسل در تصویر اولیه را از مقادیر MIN_I و MAX_I به دست می‌آوریم (قدر مطلق تفاضل). در صورتی که سطح روشنایی به MIN_I (سیاه) نزدیک‌تر بود، مقدار MAX_I (سفید) و در صورتی که به MAX_I (سفید) نزدیک‌تر بود، مقدار MIN_I (سیاه) را در پیکسل متناظر قرار می‌دهیم. در این حالت چون قدر مطلق تفاضل‌ها به ازای همه پیکسل‌های تصویر ماکسیمم هستند پس تضمین می‌شود که ماکسیمم MSE(بدترین MSE) را نسبت به تصویر ورودی داریم.

۳

۱.۳ الف

Function ۱.۱.۳

```
function J = toBlackWhite(I)
%TOBLACKWHITE Summary of this function goes here
% RGB to black and white with maximum psnr
    J = uint8(zeros(size(I)));
    for i = 1: size(I, 1)
        for j = 1: size(I, 2)
```

```
            avg_of_rgb = round(double(I(i, j, 1)) / 3 + double(I(i, j, 2) / 3) + double(I
    (i, j, 3) / 3));
            if(255 - avg_of_rgb <= avg_of_rgb)
                J(i, j, 1) = 255;
                J(i, j, 2) = 255;
                J(i, j, 3) = 255;
            else
                J(i, j, 1) = 0;
                J(i, j, 2) = 0;
                J(i, j, 3) = 0;
            end
        end
    end
end
```

Driver code ۲.۱.۳

```
clc
clear
close all
imtool close all
%%%%%%%%%%%%%%%%
I = imread("images\i3.jpg");
J = uint8(zeros(size(I)));
J = toBlackWhite(I);
psnr(J, I)
figure, imshow(I, []);
figure, imshow(J, []);
```

ب ۲.۳

Function ۱.۲.۳

```
function J = floydSteinberg(I)
%FLOYDSTEINBERG Summary of this function goes here
    I = int16(I);
    J = zeros(size(I));
    for i = 1: size(I, 1)
        for j = 1: size(I, 2)
```

```
            old_value = I(i, j, 1);
            remaining = int16(-1);
            if(255 - I(i, j, 1) < I(i, j, 1))
                I(i, j, :) = 255;
                remaining = -int16(int16(255 - old_value));
            else
                I(i, j, :) = 0;
                remaining = int16(old_value);
            end
            if(j < size(J, 2))
                I(i, j + 1, :) = I(i, j + 1, 1) + floor((7 / 16) * remaining);
                if(i < size(J, 1))
                    I(i + 1, j + 1, :) = I(i + 1, j + 1, 1) + floor((1 / 16) * remaining)
    ;
                end
            end
            if(i < size(I, 1))
                I(i + 1, j, :) = I(i + 1, j, 1) + floor((5 / 16) * remaining);
                if(j > 1)
                    I(i + 1, j - 1, :) = I(i + 1, j - 1, 1) + floor((3 / 16) * remaining)
    ;
                end
            end
        end
    end
    J = uint8(I);
end
```

۲.۲.۳ Driver code

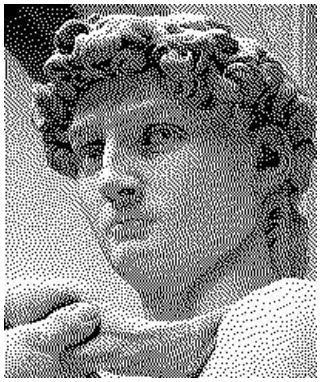
```
clc
clear
close all
imtool close all
%%%%%%%%%%%%%%%%%%
I = imread("images\i3b.png");
J = floydSteinberg(I);
```

```
figure, imshow(I, []);
figure, imshow(J, []);
x = I(:, :, 1);
y = J(:, :, 1);
```

۳.۳ ج

```
clc
clear
close all
imtool close all
%%%%%%%%%%%%%%%
I = imread("images\i3b.png");
J = floydSteinberg(I);
K = toBlackWhite(I);
fs_psnr = psnr(J, I);
bw_psnr = psnr(K, I);
figure, imshow(I, []);
figure, imshow(J, []);
figure, imshow(K, []);
```

با مقایسه‌ی PSNR در دو تصویر تولید شده در قسمت‌های قبل درمی‌یابیم که بهتر بودن PSNR یک تصویر نسبت به تصویر دیگر الزاما به معنی بهتر بودن کیفیت بصری آن تصویر نیست. همانطور که می‌بینیم، تصویر تولید شده توسط الگوریتم Floyd-Steinberg با وجود داشتن PSNR پایین‌تر نسبت به تصویر تولید شده توسط الگوریتم حریصانه (تابع toBlackWhite) جزئیات تصویر را بهتر مشخص می‌کند و کیفیت بصری بهتری دارد. پس نتیجه می‌گیریم که PSNR در همه جا معیار دقیقی برای مقایسه‌ی کیفیت تصاویر نیست.



شکل ۱: خروجی الگوریتم *Floyd-Steinberg*

شکل ۲: خروجی الگوریتم حریصانه

۴

۱.۴ Function

```
function J = My_Imresize_BL(Input_Image, Resizing_Factor)
%MY_IMRESIZE_BL Summary of this function goes here
    old_row = size(Input_Image, 1);
    old_column = size(Input_Image, 2);
    new_row = ceil(old_row * Resizing_Factor);
    new_column = ceil(old_column * Resizing_Factor);
    J = uint8(zeros(new_row, new_column, 3));
    r_ratio = double(old_row - 1) / double(new_row - 1);
    c_ratio = double(old_column - 1) / double(new_column - 1);
    for i = 1: new_row
        i_floor = uint32(floor(r_ratio * i)) + 1;
        i_ceil = uint32(ceil(r_ratio * i));
        if (i_floor > old_row)
            i_floor = old_row;
        end
        if (i_ceil > old_row)
            i_ceil = old_row;
        end
        y = uint32(r_ratio * i - i_floor);

        for j = 1: new_column
            j_floor = uint32(floor(c_ratio * j)) + 1;
```

```
            j_ceil = uint32(ceil(c_ratio * j));
            if (j_floor > old_column)
                j_floor = old_column;
            end
            if (j_ceil > old_column)
                j_ceil = old_column;
            end
            x = uint32(c_ratio * j - j_floor);

            a = uint32(zeros(1, 3));
            b = uint32(zeros(1, 3));
            c = uint32(zeros(1, 3));
            d = uint32(zeros(1, 3));
            e = uint32(zeros(1, 3));
            %if (i_floor <= old_row && i_ceil <= old_row && j_floor <= old_column &&
    j_ceil <= old_column)
                a = uint32(Input_Image(i_floor, j_floor, :));
                b = uint32(Input_Image(i_floor, j_ceil, :));
                c = uint32(Input_Image(i_ceil, j_floor, :));
                d = uint32(Input_Image(i_ceil, j_ceil, :));
                e = uint32(round(a + (b - a) * x + (c - a) * y + (a - b - c + d) * x * y)
    );
                J(i, j, :) = e;
            %end
        end
    end
end
```

Driver code ۲.۴

```
clc
clear
close all
imtool close all
%%%%%%%%%%%%%%%
I = imread("images\lena.gif");
rf = 0.3;
```

```
J = My_Imresize_BL(I, rf);
figure, imshow(I, []);
figure, imshow(J, []);
```

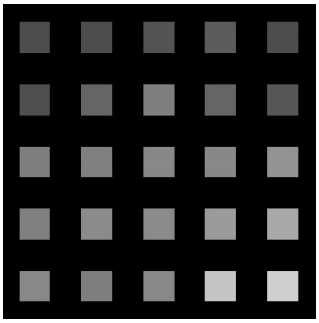
۳.۴ نمونه خروجی

شکل ۳: تصویر اصلی

شکل ۴: تصویر با ابعاد ۳۰ درصد ابعاد تصویر اصلی

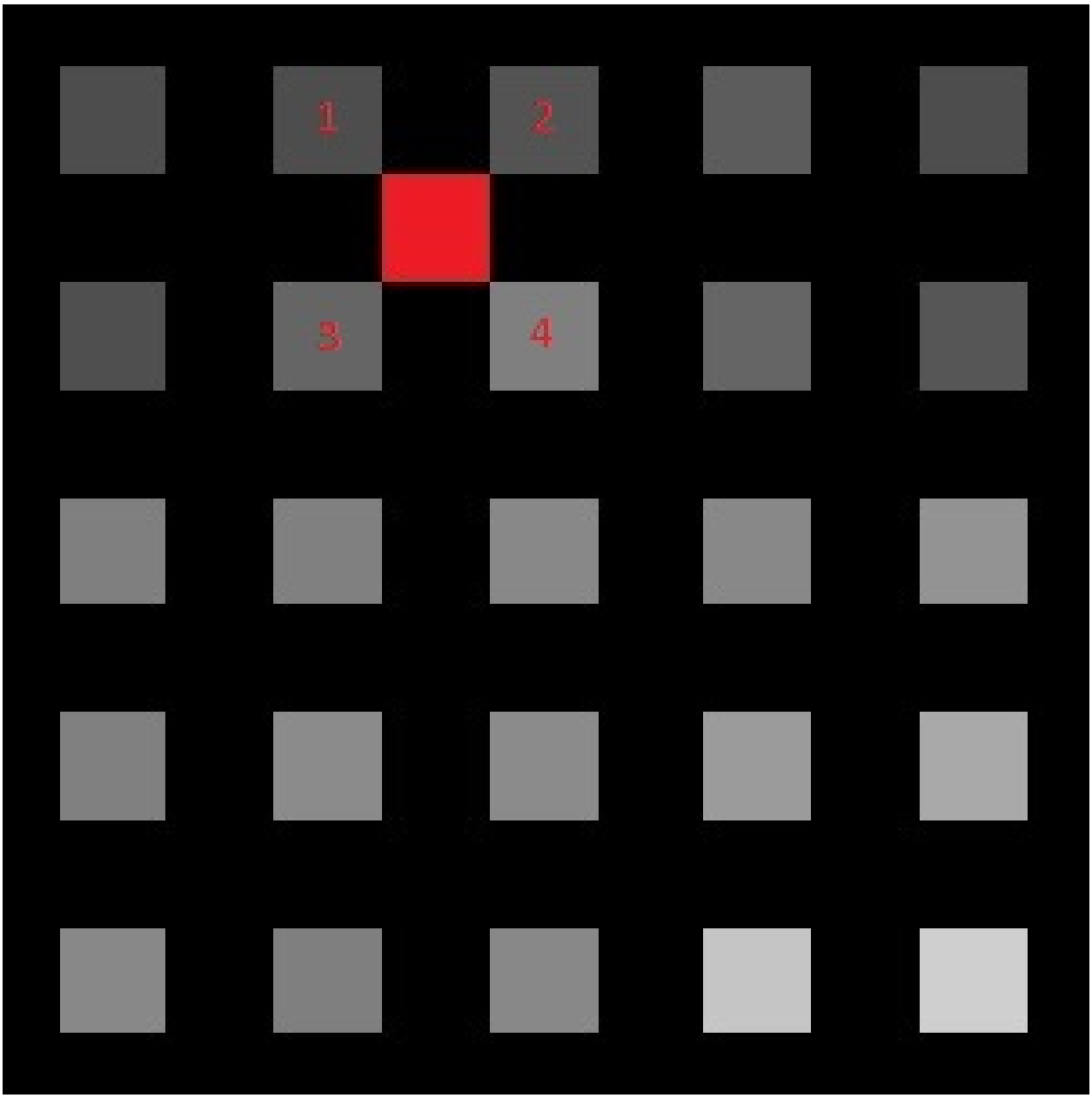
۵

پیکسل‌های تصویر را به طور یک در میان به شکل زیر می‌چینیم:



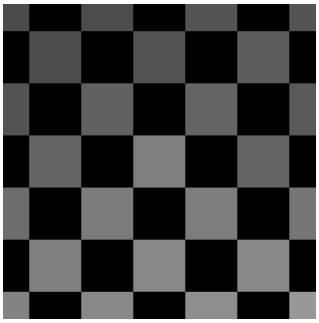
شکل ۵

سطح روشنایی پیکسل‌های قرار گرفته در سطر زوج و ستون زوج را برابر با میانگین پیکسل‌های مجاور (۴ پیکسل مشخص شده در شکل زیر در صورت وجود) قرار می‌دهیم. سطح روشنایی پیکسل‌های مرزی که دارای ۲ همسایه‌ی مورب و ۱ همسایه‌ی مورب هستند را هم به همین صورت مقداردهی می‌کنیم.



شکل ۶

پیکسل‌ها تصویر به شکل زیر در می‌آیند:



شکل ۷

در گام بعد، سطح روشنایی پیکسل‌های باقی‌مانده را برابر میانگین پیکسل‌های مجاور (بالا، پایین، چپ و راست) قرار می‌دهیم. سطح روشنایی پیکسل‌های مرزی هم به طریق مشابه تعیین می‌شوند. این الگوریتم در تابعِ doubleDimensions به شکل زیر پیاده‌سازی شده است.

۱.۵ Function

```
function K = doubleDimensions(J)
    %initializing variables
    height = 2 * size(J, 1);
    width = 2 * size(J, 2);
    K = uint8(zeros(height, width , 3));
    %arranging pixels alternately
    for i = 1: height / 2
        for j = 1: width / 2
            K(2 * i - 1, 2 * j - 1, :) = J(i, j, :);
        end
    end
    %assigning pixels with even row and even column with average of diagonal neighbors
    for i = 2: 2: height - 1
        for j = 2: 2: width - 1
            K(i, j, :) = (1 / 4) * K(i - 1, j - 1, :) + (1 / 4) * K(i - 1, j + 1, :) + (1
     / 4) * K(i + 1, j - 1, :) + (1 / 4) * K(i + 1, j + 1, :);
        end
    end
    %setting border pixels
    for j = 2: 2: width - 1
        K(height, j, :) = (1 / 2) * K(height - 1, j - 1, :) + (1 / 2) * K(height - 1, j +
     1, :);
```

```
    end
    for i = 2: 2: height - 1
        K(i, width, :) = (1 / 2) * K(i - 1, width - 1, :) + (1 / 2) * K(i + 1, width - 1,
 :);
    end
    K(height, width, :) = K(height - 1, width - 1, :);
    %assigning left pixels with average of up, down, right and left neighbors
    for i = 2: height - 1
        for j = 2: width - 1
            if (mod(i, 2) == 0 && mod(j, 2) == 1) || (mod(i, 2) == 1 && mod(j, 2) == 0)
                K(i, j, :) = (1 / 4) * K(i - 1, j, :) + (1 / 4) * K(i + 1, j, :) + (1 /
4) * K(i, j - 1, :) + (1 / 4) * K(i, j + 1, :);
            end
        end
    end
    %setting border pixels
    for j = 2: width - 1
        if (mod(j, 2) == 0)
            K(1, j, :) = (1 / 3) * K(1, j - 1, :) + (1 / 3) * K(1, j + 1, :) + (1 / 3) *
K(2, j, :);
        else
            K(height, j, :) = (1 / 3) * K(height, j - 1, :) + (1 / 3) * K(height, j + 1,
:) + (1 / 3) * K(height - 1, j, :);
        end
    end
    for i = 2: height - 1
        if (mod(i, 2) == 0)
            K(i, 1, :) = (1 / 3) * K(i - 1, 1, :) + (1 / 3) * K(i + 1, 1, :) + (1 / 3) *
K(i, 2, :);
        else
            K(i, width, :) = (1 / 3) * K(i - 1, width, :) + (1 / 3) * K(i + 1, width, :)
+ (1 / 3) * K(i, width - 1, :);
        end
    end
    K(height, 1, :) = (1 / 3) * K(height - 1, 1, :) + (1 / 3) * K(height - 1, 2, :) + (1
/ 3) * K(height, 2, :);
```

```
    K(1, width, :) = (1 / 3) * K(1, width - 1, :) + (1 / 3) * K(2, width - 1, :) + (1 /
    3) * K(2, width, :);
end
```

۲.۵ Driver Code

```
clc
clear
close all
imtool close all
%%%%%%%%%%%%%%%
I = imread("images\Cameraman.png");
J = imread("images\LR_Cameraman.png");
nn = imresize(J, 2, "nearest");
bil = imresize(J, 2, "bilinear");
bic = imresize(J, 2, "bicubic");

tic
K = doubleDimensions(J);
toc

[psnr(nn, I) psnr(bil, I) psnr(bic, I) psnr(K, I)]
imtool(I)
imtool(K)
```

جدول مقادیرِ PSNR به شکل زیر می‌باشد.

نام تصویر	مقدار **PSNR** به ازای **Resizing_Factor** برابر با ۲				
	روش **Nearest Neighbor**	روش **Bilinear**	روش **Bicubic**	روش پیشنهادی شما	زمان اجرا بر حسب ثانیه
Boat	**25.5147**	**27.1040**	**26.9296**	**28.8864**	**0.899658**
Peppers	**28.1116**	**29.9490**	**29.7193**	**32.1106**	**0.941563**
Cameraman	**28.0316**	**30.3392**	**30.4896**	**34.6318**	**0.913748**
House	**27.5452**	**29.3679**	**29.2722**	**31.5815**	**0.228155**
متوسط **PSNR**	**27.30075**	**29.190475**	**29.102675**	**31.802575**	

۶

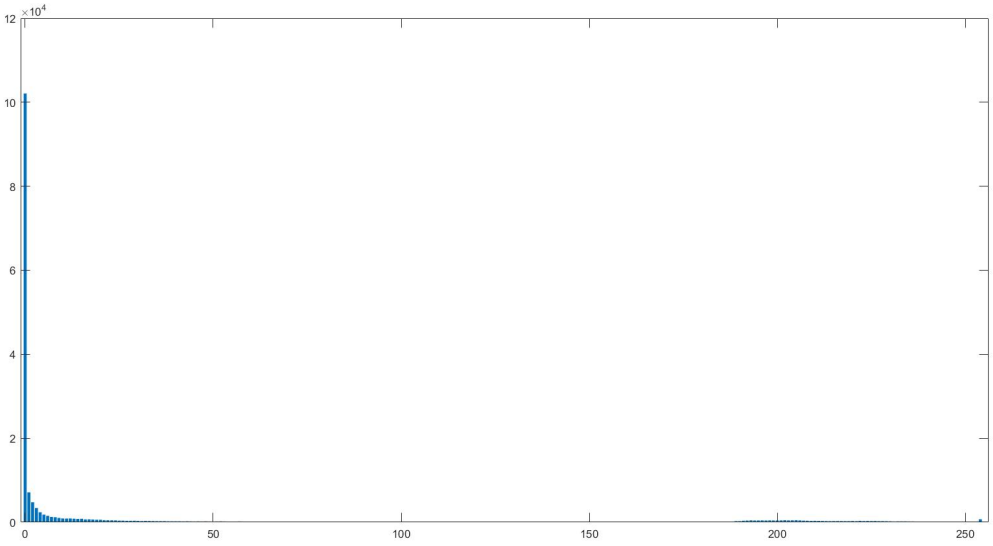
۱.۶ الف

۱.۱.۶ Function

```
function hist_vector = myHist(I)
%MYHIST Summary of this function goes here
    x_axis = 0: 255;
    y_axis = zeros(1, 256);
    for i = 1: size(I, 1)
        for j = 1: size(I, 2)
            y_axis(I(i, j) + 1) = y_axis(I(i, j) + 1) + 1;
        end
    end
    hist_vector = y_axis;
    figure, bar(x_axis, y_axis);
end
```

۲.۱.۶ Driver code

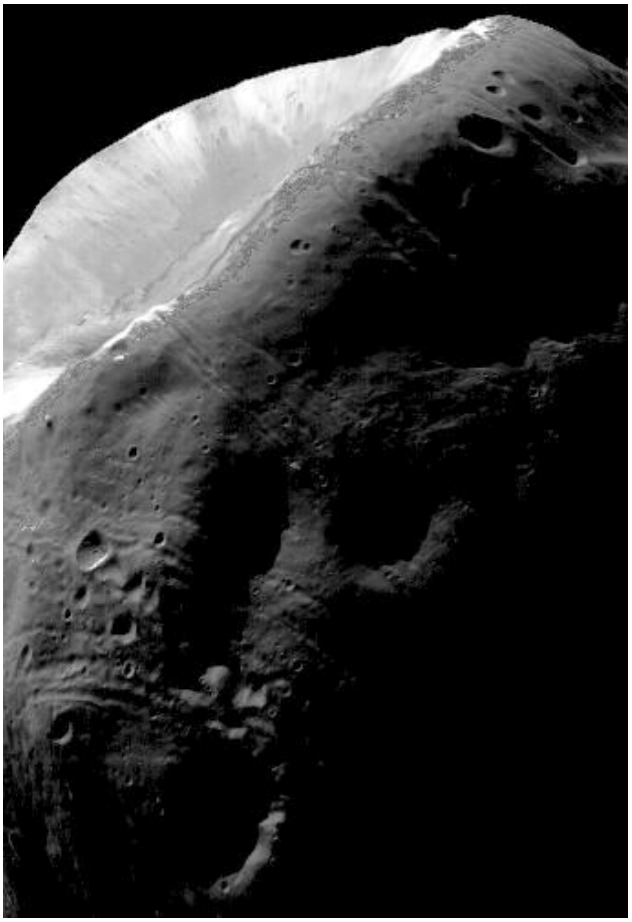
```
clc
clear
close all
imtool close all
%%%%%%%%%%%%%%%%
I = imread("images\Image.tif");
myHist(I);
```



شکل ۸: هیستوگرام تصویر *Image.tif*

۲.۶ ب

همانطور که در هیستوگرام دیدیم، سطح روشنایی‌های صفر و نزدیک به صفر بیشتر پیکسل‌های این تصویر را تشکیل داده‌اند. اما سطح روشنایی‌های بزرگ‌تر و نزدیک به ۲۵۵ هم در این تصویر وجود دارد. برای بهبود کیفیت تصویر برای سطح روشنایی‌های پایین (سطح روشنایی کمتر مساوی k)، تابع تبدیلی می‌تواند مناسب باشد که بتواند سطوح روشنایی پایین را به محدوده وسیع‌تری نگاشت کند و از سوی دیگر قابلیت حفظ تقریبی سطوح روشنایی بالای قبلی را هم داشته باشد. بنابراین از تابع تبدیل لگاریتمی می‌توانیم استفاده کنیم. همچنین برای بهبود کیفیت تصویر برای سطح روشنایی‌های بالا (سطح روشنایی بزرگ‌تر از k)، بخش عمده‌ای از پیکسل‌ها دارای سطح روشنایی بالا هستند، ولی مناطقی شامل پیکسل‌هایی با سطح روشنایی پایین نیز در تصویر وجود دارد. این تبدیل، سطوح روشنایی بالا (نواحی روشن تصویر) را به محدوده گسترده‌تری نگاشت می‌کند. بنابراین تابع تبدیل معکوس لگاریتمی می‌تواند مناسب باشد (برای وضوح بیشتر قسمت سفید مقدار گاما را برابر با 1.2 قرار داده‌ایم و k می‌تواند ۱۲۷ باشد). توابع استفاده شده به ترتیب $J = 31.875 \times \log(I+1)$ و $J = \frac{1}{255} I^2$ هستند. خروجی تصویر به شکل زیر می‌باشد.



شکل ۹: خروجی تابع تبدیل

Code ۳.۶

```
clc
clear
close all
imtool close all
%%%%%%%%%%%%%%%
I = imread("images\Image.tif");
gamma = 1.2;
alpha = double(255) ^ (1 - gamma);
for i = 1: size(I, 1)
    for j = 1: size(I, 2)
        if(I(i, j) <= 127)
            J(i, j) = uint8(round(31.875 * log(double(I(i, j)) + 1)));
        else
```

```
14              J(i, j) = uint8(round(alpha * double(I(i, j)) ^ gamma));
15          end
16      end
17  end
18  imtool(imadjust(J));
```

منابع