بنام خدا

دانشگاه صنعتی اصفهان
دانشکده برق و کامپیوتر

تمرین سری دوم –پردازش تصاویر دیجیتال

استاد: دکتر سعید صدری

رضا سعادتی فرد ۹۴۱۱۳۹۴
پروانه رشوند ۹۴۱۰۱۲۴

تاریخ تحویل ۹۵/۰۷/۱۳

سوال اول:

الف) (کد مربوطه در فایل HW2_Q1_A.m می باشد): در این سوال دنباله ی f(n)=[0,0,1,3,-2,5,5,5,-1,3,3,2,0,0] داده شده است و قرار است این دنباله را توسط روش درونیابی مرتبه ی سوم، چهار و نیم برابر کنیم.

برای این منظور از دو روش استفاده می کنیم. یک روش، استفاده از ماتریس A و دیگری استفاده از روش Cubic-Splin. سپس می خواهیم ببینیم که این دو روش چقدر نسبت به هم خطا دارند:

در روش استفاده از ماتریس A کد برنامه به صورت زیر می باشد:

```
%%%%%%%%%%%%%%%%%%% Matrix A Interpolation

A =[-0.5 1.5 -1.5 0.5; 1 -2.5 2 -0.5;-0.5 0 0.5 0;0 1 0 0];
f = [0 0 1 3 -2 5 5 5 -1 3 3 2 0 0];

step=1/4.5;

p_n = (3:-1:0).';
p=0;
for n=2:step:length(f)-2
    p = p+1;
    f_n(p) = fix(n);                %%% fixed value of n
    d_n = n-f_n(p);                 %%% delta_n
    points = f(f_n(p)-1:f_n(p)+2); %%% extract 4  points in order to interolate
    Coef = A*points.';              %%% calculate coeficients
    f_A(p) = Coef.' * d_n.^ p_n;    %%% calulate interpolated point

end

ind_5=find(f_n==5);                          %%% indices include f(5)
ind_7=find(f_n==7);                          %%% indices include f(7)
f_A_5_7 = f_A(ind_5(1):ind_7(end))          %%% interpolated Data Between 5 and 7
```

پس از اجرای برنامه مقادیر داده های بین f(5) تا f(7) به صورت زیر خواهد بود:

f_A_5_7=

-1.7106 -0.2963 1.7106 3.6845 5.0000 5.4705 5.4801 5.2593 5.0384 5.0329 5.2222

5.4115 5.4033

همچنین ماتریس ضرایب ب صورت زیر خواهد بود که در Cubic_Coef.txt نیز موجود می باشد:

5	1	0	0	0	0	0	0	0	0	0	0
1	4	1	0	0	0	0	0	0	0	0	0
0	1	4	1	0	0	0	0	0	0	0	0
0	0	1	4	1	0	0	0	0	0	0	0
0	0	0	1	4	1	0	0	0	0	0	0
0	0	0	0	1	4	1	0	0	0	0	0
0	0	0	0	0	1	4	1	0	0	0	0
0	0	0	0	0	0	1	4	1	0	0	0
0	0	0	0	0	0	0	1	4	1	0	0
0	0	0	0	0	0	0	0	1	4	1	0
0	0	0	0	0	0	0	0	0	1	4	1
0	0	0	0	0	0	0	0	0	0	1	5

در قسمت بعدی از روش مقاله ی cubic-spline برای ۴,۵ برابر کردن دنباله ی f(n) استفاده می نماییم:

کد برنامه به صورت زیر می باشد:

```
%%%%%%%%%%%%%%%% Cubic Spline Interpolation

Cubic_Coef = load('Cubic_Coef.txt')         %% Coeficient for Cubic Spline Method

%%% Y is Si(xi)
for j=1:12
    Y(j) = f(j) - 2*f(j+1) + f(j+2);
end

M = 6.* inv(Cubic_Coef)*Y.' ;                %% M is the second derivative matrice [Si"(xi)]
M = [M(1);M;M(12)];                          %% Update M and calculate M_1 and M_14

%%%%% calculate a , b , c , d
M_1 = M(2:end);
f_1 = f(2:end);

M_0 = M(1:12); M_1 = M_1(1:12);
f_0 = f(1:12).'; f_1 = f_1(1:12).';

a = (M_1 - M_0)/6;
b = M_0/2;
c = (f_1 - f_0) - ((M_1 + 2*M_0)/6);
d = f_0;


p=0;
for n=2:step:length(f)-2
    p = p+1;
    n_Cubic(p) = fix(n);
    d_n = n-n_Cubic(p);
    points = f(n_Cubic(p)-1:n_Cubic(p)+2);
    Coef = [a(n_Cubic(p)) ;b(n_Cubic(p)) ;c(n_Cubic(p)) ;d(n_Cubic(p))];
    f_Cubic(p) = Coef.' * d_n.^ p_n;
end

ind_5=find(n_Cubic==5);                      %%% indices include f(5)
ind_7=find(n_Cubic==7);                      %%% indices include f(7)
f_Cubic_5_7 = f_Cubic(ind_5(1):ind_7(end)); %%% interpolated Data Between 5 and 7

%%%% Calculate rms error
rms_Error = norm(f_Cubic - f_A)/norm(f_Cubic)
```

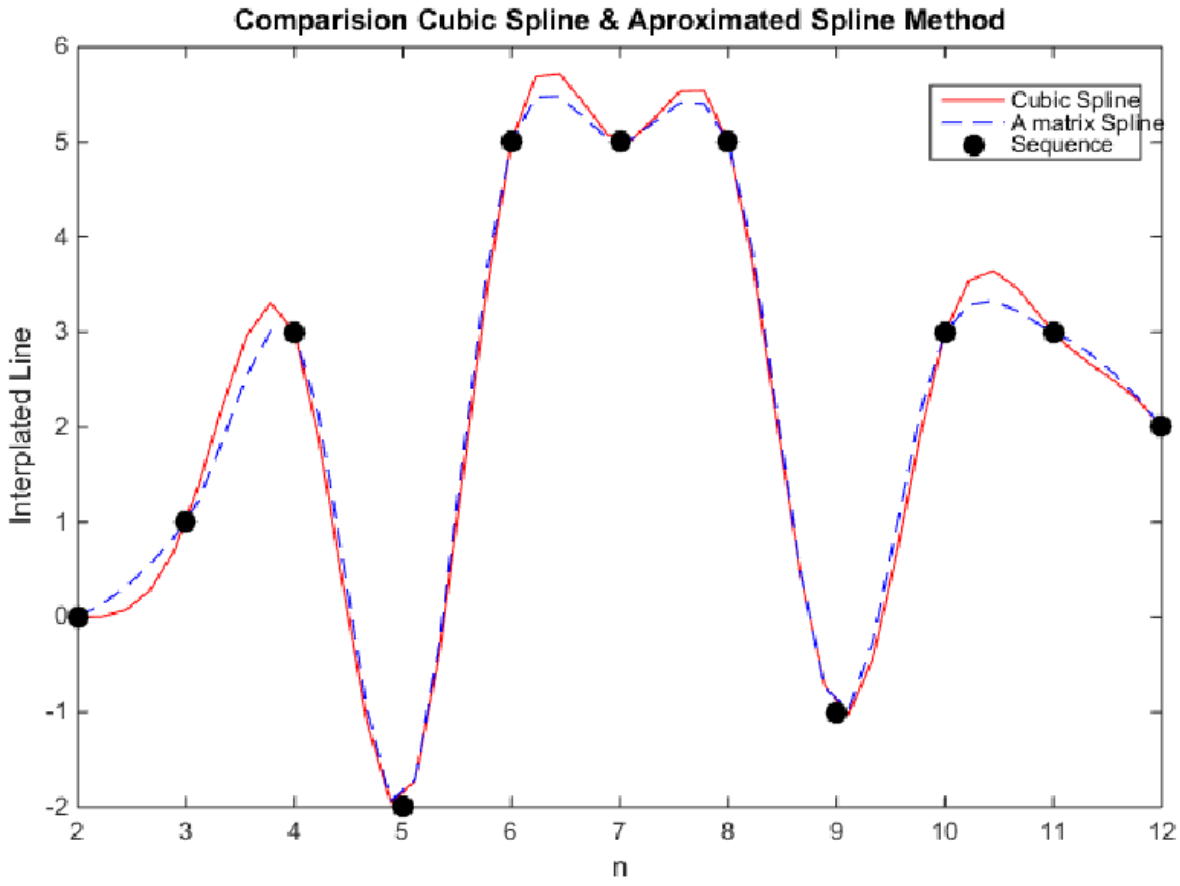
پس از اجرای برنامه، مقادیر بین f(5) تا f(7) به صورت زیر خواهد بود:

```
f_Cubic_5_7 =

   -1.7222   -0.4018    1.5053    3.4792    5.0000    5.6873    5.7181    5.4090    5.0763    5.0240    5.2678
    5.5352    5.5409
```

برای مقایسه ی این دو روش نموداری از مقادیر به دست آمده از هر دو روش را رسم می نماییم تا تفاوت ها بهتر مشخص شوند.

```
%%%% Plotiing Cubic and Aproximated Method
t = (2:step:length(f)-2);
figure;
plot(t,f_Cubic,'r'); hold on;
plot(t,f_A,'b--'); hold on;
plot(2:length(f)-2,f(2:length(f)-2),'k.','MarkerSize',30);
xlabel('n'), ylabel('Interplated Line') , title('Comparision Cubic Spline & Aproximated Spline Method')
legend('Cubic Spline','A matrix Spline','Sequence')
```

نتیجه ی حاصل از رسم هم زمان دو روش به صورت زیر خواهد بود:



هم چنین با محاسبه ی خطای بین دو روش، مشاهده می نماییم که خطا حدود ۵ درصد خواهد بود.

خطا را به صورت `rms_Error = norm(f_Cubic - f_A)/norm(f_Cubic)` محاسبه نموده ایم که در کد ذکر شده نیز موجود می باشد.

هم از روی مقایسه ی داده ها و هم از روی پایین بودن میزان خطا، مشاهده می شود که به علت نزدیک بودن داده ها به هم، تقریبی که در روش ماتریس A از مشتق مرتبه ی اول به صورت $\frac{df}{dx} = \frac{f(m+1)-f(m-1)}{2}$ زده ایم تقریبا دقیق بوده و با روش cubic-spline تقریبا برابر بوده و از دقت خوبی برخوردار می باشد.

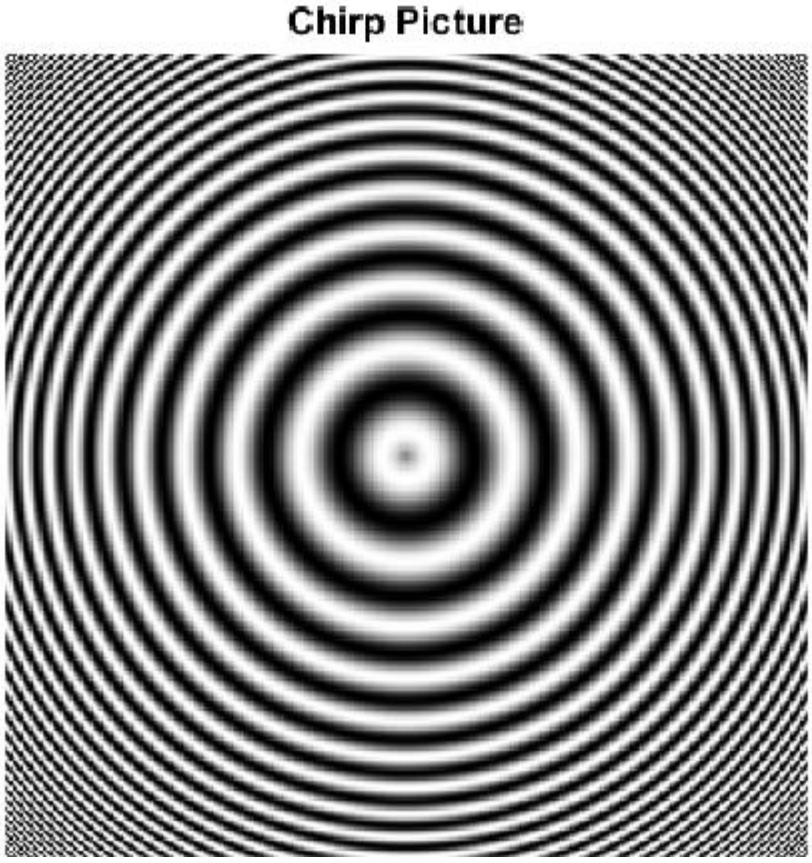
ب) (کد مربوطه در HW2_Q1_B.m قرار دارد) : در این سوال قرار است تصویر دوایر هم مرکز را از سه روش درونیابی نزدیک ترین همسایه، خطی و درجه ۳ با مقیاس ۲،۳ برابر کوچک کرده و سپس نتیجه حاصل را با همان مقیاس بزرگ کرده تا تصویری با همان ابعاد تصویر اولیه به دست آید. در مرحله بعد میزان خطای rms این تصویر را نسبت به تصویر اصلی به دست آورده تا تفاوت میزان عملکرد این ۳ روش را نسبت به یکدیگر به دست آوریم.

ابتدا تصویر اصلی را می سازیم. کد مورد نظر به صورت زیر می باشد:

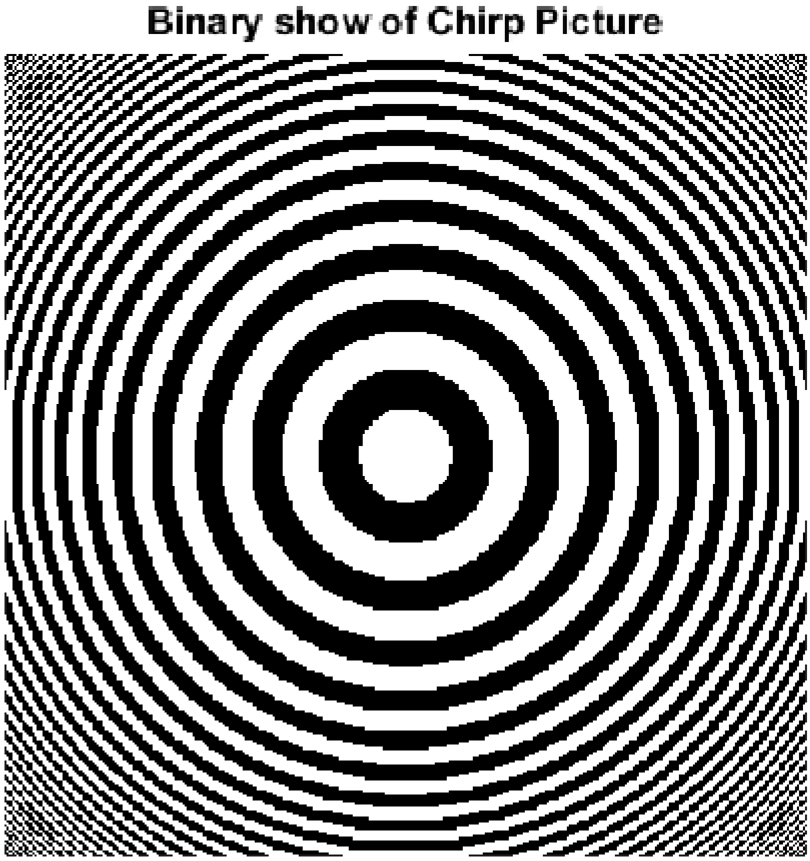
```
%%%% Generate chirp picture
r_max = 200*sqrt(2); r_min = 0;
T_max = 50;          T_min = 10;

p=0;
for m=-200:200
    p=p+1; q=0;
    for n=-200:200
        q=q+1;
        r= sqrt(m^2 + n^2);
        T = ((T_min -T_max)/(r_max - r_min))*r + T_max;
        f(p,q) = sin(2*pi*r/T);
    end
end

f_Binary = f>=0;   %%%% convert photo 2 binary

 figure
 imshow(f,[])
 title('Chirp Picture');
% %%% Binary show
 figure
 imshow(f_Binary,[])
 title('Binary show of Chirp Picture');
```

توجه شود که برای مشاهده ی بهتر تغییرات، تصویر را به صورت باینری نیز تولید کرده ایم.

بنابراین تصویر اصلی به شکل زیر می باشد:



تصویر باینری شده نیز بدین شکل است:



حال در ادامه به بررسی سه روش ذکر شده می پردازیم:

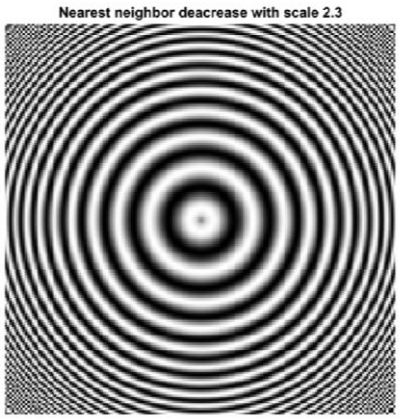
- **روش اول: روش نزدیک ترین همسایه:**

ابتدا تصویر را با مقیاس ۲,۳ کوچک کرده و سپس با همان مقیاس بزرگ می نماییم. کد مورد نظر به صورت زیر می باشد:

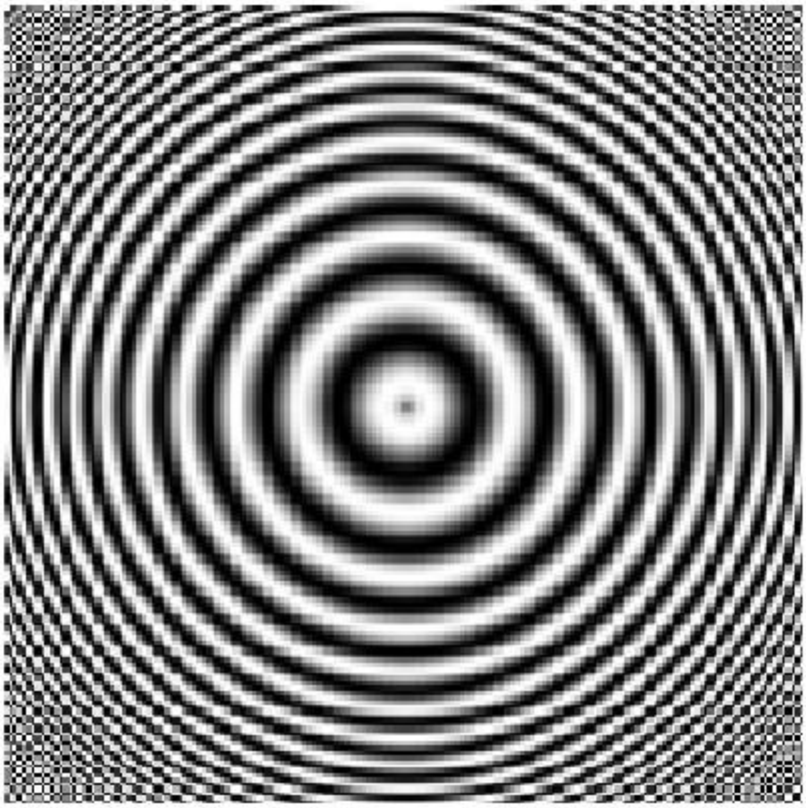
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   downsample photo
S =2.3; %%% downsample rate
[M,N] = size(f);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% nearest neighbohood interpolation
%%%%%%%%%% DOWNSAMPLE
p = 0;
for m=1 :S : M
    p = p+1; q = 0;
    for n= 1 :S :N
        q = q+1;
         l= round(m);
         k=round(n);
        f_Nearest_D(p,q) = f(l,k);
    end
end
f_Nearest_D_B = f_Nearest_D>=0;
 figure
 imshow(f_Nearest_D_B,[])
 str_tit = sprint('nearest neighbor decrease with S=%.1f ',S);
 title(str_tit);
```

```
%%%%%%%% UPSAMPLE
[M,N] = size(f_Nearest_D);
p = 0;
for m=1 : 1/S : M
    p = p+1; q = 0;
    for n= 1 :1/S :N
        q = q+1;
         l= round(m);
         k=round(n);
        f_Nearest_U(p,q) = f_Nearest_D(l,k);
    end
end
f_Nearest_U_Binary=f_Nearest_U>=0;    %%% convert photo to a binary photo

figure
imshow(f_Nearest_U,[])
str_tit = sprintf('Nearest Neighbor increase with S=%.1f ',S);
title(str_tit);
%%%%% Binary show
figure
imshow(f_Nearest_U_Binary,[])
str_tit = sprintf('Binaryb show of Nearest Neighbor interpolation with S=%.1f ',S);
title(str_tit);

[M,N] = size(f_Nearest_U);
error_Neighbor = norm(f(1:M,1:N)-f_Nearest_U)/norm(f);
```

نتیجه ی مشاهده شده از کوچک شدن تصویر به صورت زیر خواهد بود:



پس از بزرگ نمودن تصویر نتیجه زیر را مشاهده می نماییم:



در ادامه مقدار خطای این تصویر نسبت به تصویر اصلی را محاسبه می کنیم که در کد ذکر شده ، شیوه محاسبه ی آن موجود است. نتیجه، خطای ۲۱ درصد را به دست می دهد. همانطور که از مقایسه ی شکل بالا و تصویر اصلی داریم و همچنین مشاهده ی خطای به دست آمده، به این نتیجه میرسیم که روش نزدیک ترین همسایه، نتیجه ی خوبی را ارائه نمی دهد.

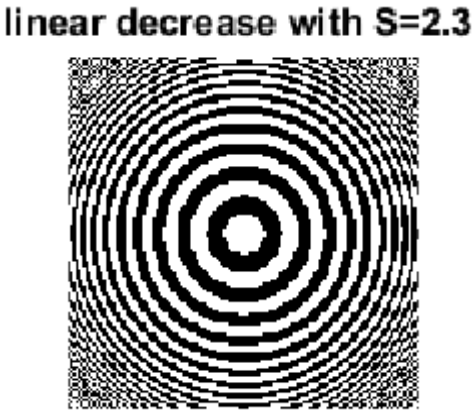
- **روش دوم: روش خطی**

کد مورد نظر به صورت زیر خواهد بود:

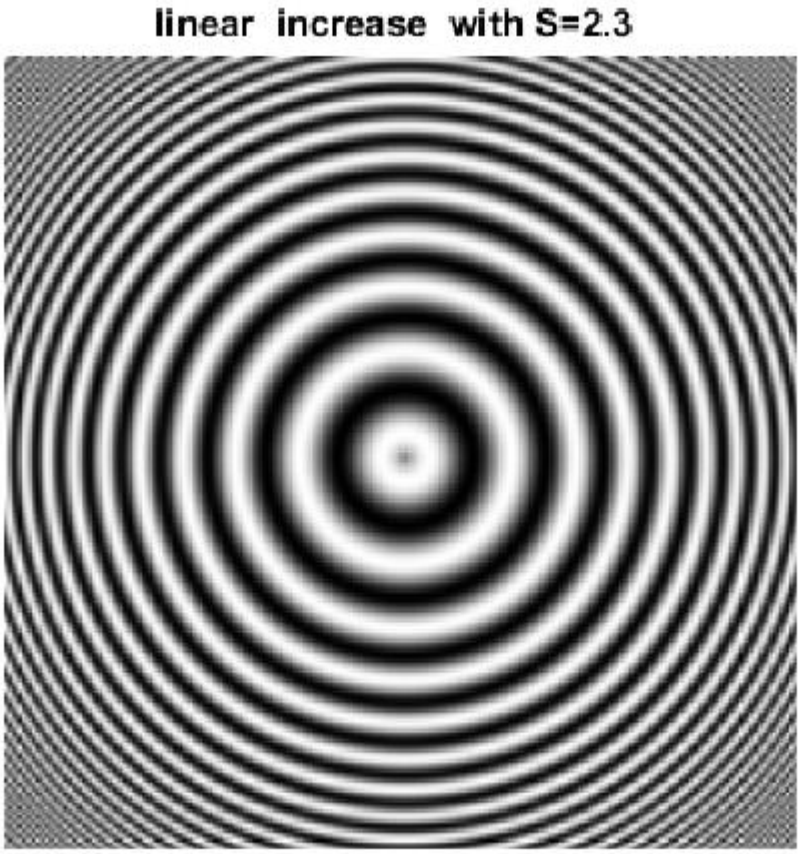
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% linear interpolation
%%%%%%%% DOWNSAMPLE
[M,N] = size(f);
p = 0;
for m=1 :S : M
    p = p+1; q = 0;
    for n= 1 :S :N
        q = q+1;

        step_m = m-floor(m);
        step_n = n-floor(n);


         l= floor(m);
         k= floor(n);

         a = f(l , k) ;
         b = f(l , k+1);
         c = f(l+1 , k);
         d = f(l+1 , k+1);

         A = step_m*c + (1-step_m)*a;
         B = step_m*d + (1-step_m)*b;
```

```
        f_Linear_D(p,q) = step_n*B + (1-step_n)*A;
    end
end
f_Linear_D_B = f_Linear_D>=0;
 figure
 imshow(f_Linear_D_B,[])
 str_tit = sprintf('linear decrease with S=%.1f ',S);
 title(str_tit);

%%%%%%% UPSAMPLE
[M,N] = size(f_Linear_D);
p = 0;
for m=1 : 1/S : M-1
    p = p+1; q = 0;
    for n= 1 : 1/S :N-1
        q=q+1;

        step_m = m-floor(m);
        step_n = n-floor(n);

         l= floor(m);
         k= floor(n);

         a = f_Linear_D(l , k) ;
         b = f_Linear_D(l , k+1);
         c = f_Linear_D(l+1 , k);
         d = f_Linear_D(l+1 , k+1);

         A = step_m*c + (1-step_m)*a;
         B = step_m*d + (1-step_m)*b;

        f_Linear_U(p,q) = step_n*B + (1-step_n)*A;
    end
end
f_Linear_U_Binary=f_Linear_U>=0;    %%% convert photo to a binary photo

figure
imshow(f_Linear_U,[])
str_tit = sprintf('linear  increase  with S=%.1f ',S);
title(str_tit);
%%%%% Binary show
figure;
imshow(f_Linear_U_Binary,[])
str_tit = sprintf('Binary show of linear interpolation with S=%.1f ',S);
title(str_tit);

[M,N] = size(f_Linear_U);
error_Linear = norm(f(1:M,1:N)-f_Linear_U)/norm(f);
```

نتیجه مورد نظر از کوچک شدن تصویر به صورت زیر می باشد :



و نتیجه ی حاصل از بزرگ شدن تصویر به صورت زیر خواهد بود:



میزان خطای این تصویر نسبت به تصویر اصلی در روش درونیابی خطی، ۱۱ درصد محاسبه شد. با مقایسه ی این تصویر نسبت به تصویر اصلی و مشاهده ی خطای به دست آمده، همانطور که انتظار داشتیم، مشاهده می نیم که روش درونیابی خطی به نسبت روش نزدیک ترین همسایه بهتر عمل کرده و خطا تقریبا نصف شده است.

- **روش سوم: روش درونیابی مرتبه سه:**

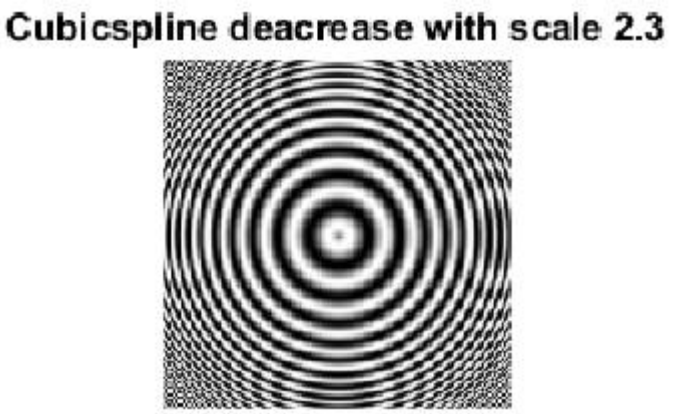
کد مورد نظر به صورت زیر است:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% spline Interpolation
A =[-0.5 1.5 -1.5 0.5; 1 -2.5 2 -0.5;-0.5 0 0.5 0;0 1 0 0];
p_c = (3:-1:0).';                                  %%% power coef
%%%%%%% DOWN SAMPLE
[M,N] = size(f);
f_ZP = zeros(M+3,N+3);                             %%% zero-pad a row and column at the first and 2 row and
coloumn at the end
f_ZP(2:end-2,2:end-2) = f;
[M,N] = size(f_ZP);
p=0;
for m=2:S:M-2
    p=p+1; q=0;
    for n=2:S:N-2
        q=q+1;
        f_m = fix(m);                               %%% fixed value of m
        d_m = m - f_m;                              %%% delta_m
        f_n = fix(n);                               %%% fixed value of n
        d_n = n - f_n;                              %%% delta_n

        IP_Data = f_ZP(f_m-1:f_m+2,f_n-1:f_n+2);    %%% extract 16  points in order to interpolate

        %%%% interpolatin for 1st row
        IP_A = IP_Data(1,:).';                      %%% interpolate with 4 points
        Coef_A = A*IP_A;                            %%% calculate coeficients
        IP_ABCD(1,1) = Coef_A.' * (d_n.^ p_c);      %%% calulate interpolated point
```

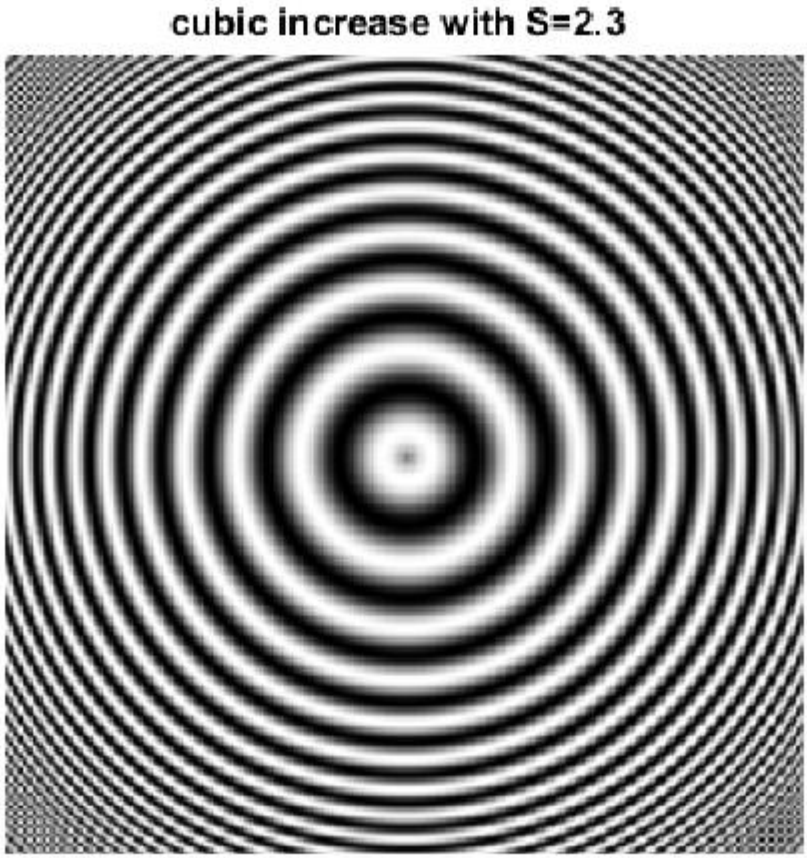
```
        %%%% interpolatin for 2nd row
        IP_B = IP_Data(2,:).';
        Coef_B = A*IP_B;
        IP_ABCD(2,1) = Coef_B.' * (d_n.^ p_c);

        %%%% interpolatin for 3rd row
        IP_C = IP_Data(3,:).';
        Coef_C = A*IP_C;
        IP_ABCD(3,1) = Coef_C.' * (d_n.^ p_c);

        %%%% interpolatin for 4th row
        IP_D = IP_Data(4,:).';
        Coef_D = A*IP_D;
        IP_ABCD(4,1) = Coef_D.' * (d_n.^ p_c);

        %%%% interpolatin of 4 above row
        Coef_Tot = A*IP_ABCD;
        f_A_Spline_D(p,q) = Coef_Tot.' * (d_m.^ p_c);  %%% f down sample with spline approach
    end
end
f_A_Spline_D_B = f_A_Spline_D>=0;
% figure
% imshow(f_A_Spline_D_B,[])
% str_tit = sprintf('cubic interpolation with S=%.1f ',S);
% title(str_tit);

%%%%%% UPSAMPLE
[M,N] = size(f_A_Spline_D);
f_A_Spline_D_ZP = zeros(M+3,N+3);                          %%% zero-pad a row and column at the first and 2
row and coloumn at the end
f_A_Spline_D_ZP(2:end-2,2:end-2) = f_A_Spline_D;
[M,N] = size(f_A_Spline_D_ZP);

p=0;
for m=2:1/S:M-2
    p=p+1; q=0;
    for n=2:1/S:N-2
        q=q+1;
        f_m = fix(m);                           %%% fixed value of m
        d_m = m - f_m;                          %%% delta_m
        f_n = fix(n);                           %%% fixed value of n
        d_n = n - f_n;                          %%% delta_n

        IP_Data = f_A_Spline_D_ZP(f_m-1:f_m+2,f_n-1:f_n+2);   %%% extract 16  points in order to interpolate

        %%%% interpolatin for 1st row
        IP_A = IP_Data(1,:).';                  %%% interpolate with 4 points
        Coef_A = A*IP_A;                        %%% calculate coeficients
        IP_ABCD(1,1) = Coef_A.' * (d_n.^ p_c);  %%% calulate interpolated point

        %%%% interpolatin for 2nd row
        IP_B = IP_Data(2,:).';
        Coef_B = A*IP_B;
        IP_ABCD(2,1) = Coef_B.' * (d_n.^ p_c);

        %%%% interpolatin for 3rd row
        IP_C = IP_Data(3,:).';
        Coef_C = A*IP_C;
        IP_ABCD(3,1) = Coef_C.' * (d_n.^ p_c);

        %%%% interpolatin for 4th row
        IP_D = IP_Data(4,:).';
        Coef_D = A*IP_D;
        IP_ABCD(4,1) = Coef_D.' * (d_n.^ p_c);

        %%%% interpolatin of 4 above row
        Coef_Tot = A*IP_ABCD;
        f_A_Spline_U(p,q) = Coef_Tot.' * (d_m.^ p_c);  %%% f up sample with spline approach
    end
end
f_A_Spline_U_Binary=f_A_Spline_U>=0;    %%% convert photo to a binary photo

figure
imshow(f_A_Spline_U,[])
str_tit = sprintf('cubic interpolation with S=%.1f ',S);
title(str_tit);
%%% Binary show
figure;
imshow(f_A_Spline_U_Binary,[])
str_tit = sprintf('Binary show of cubic interpolation with S=%.1f ',S);
title(str_tit);

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

```
error_Cubic = norm(f(1:M,1:N)-f_A_Spline_U)/norm(f);
```

نتیجه ی حاصل از کوچک کردن تصویر به صورت زیر است:

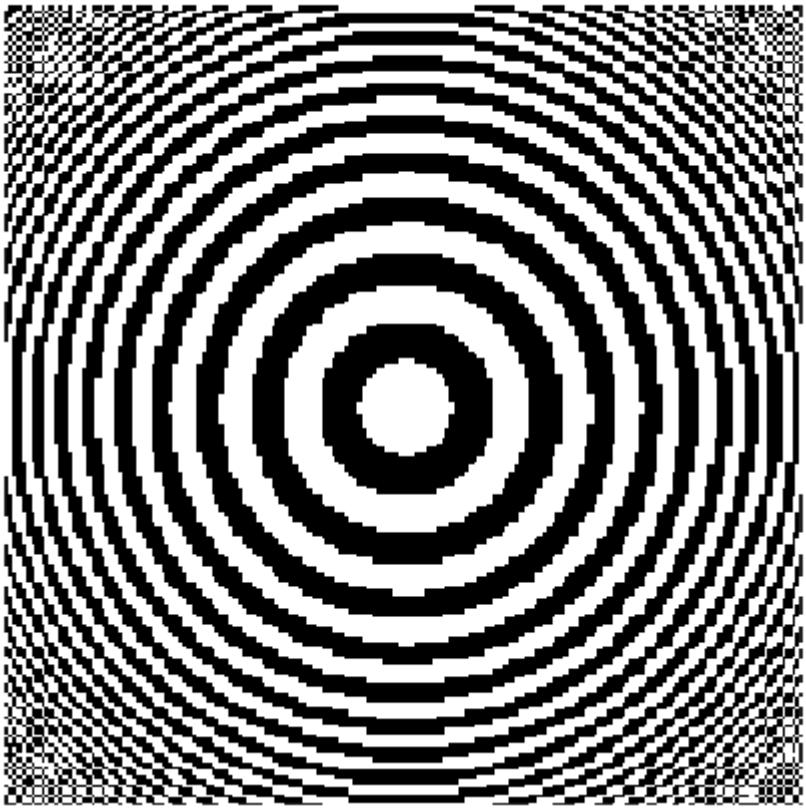


و نتیجه ی بزرگ کردن تصویر بالا به صورت زیر خواهد بود:

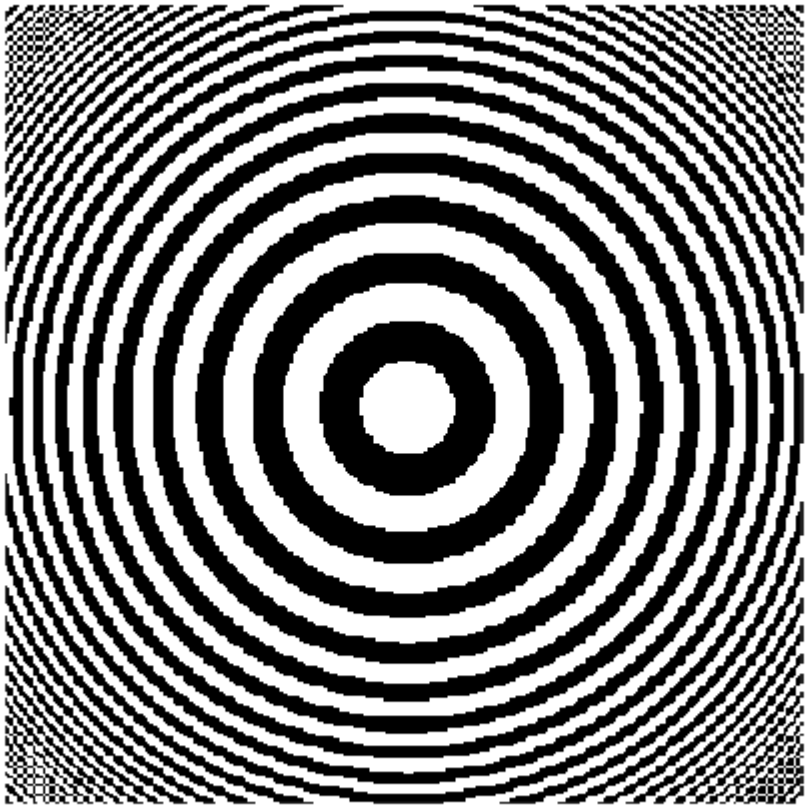


میزان خطای به دست آمده در روش درونیابی مرتبه ی سه، نسبت به تصویر اصلی ۱۲ درصد به دست آمد که بر خلاف انتظار، اندکی از خطای به دست آمده در روش درونیابی خطی بیشتر شد.
در ادامه برای مقایسه ی بهتر این سه روش که در بالا به آن ها پرداخته شد، تصویر اسکیل شده ی باینری از طریق هر سه روش را نشان می دهیم تا تفاوت ها و عملکردشان بهتر آشکار شود:

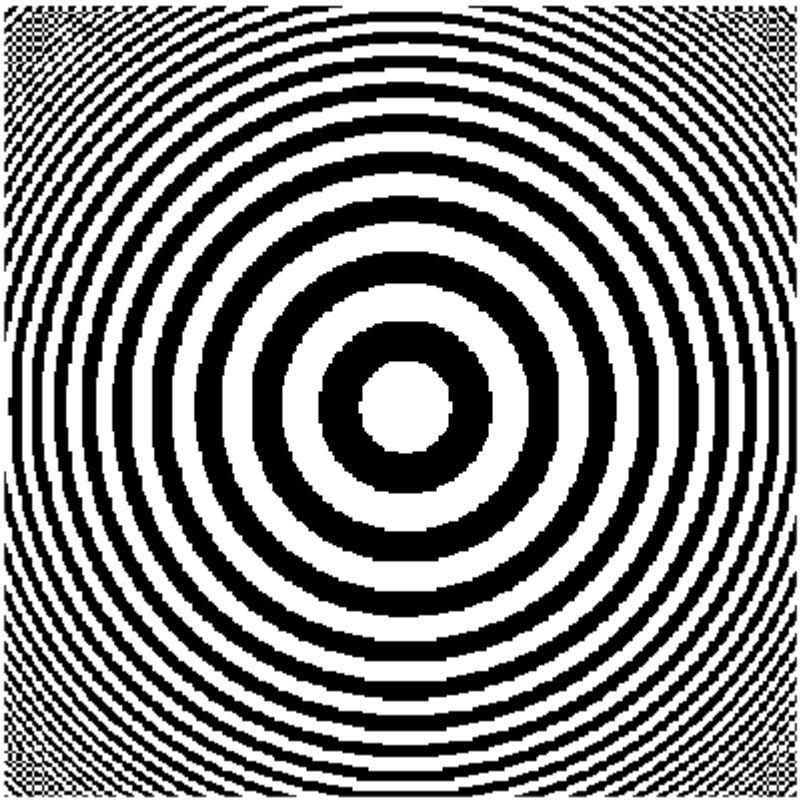
Binaryb show of Nearest Neighbor interpolation with S=2.3



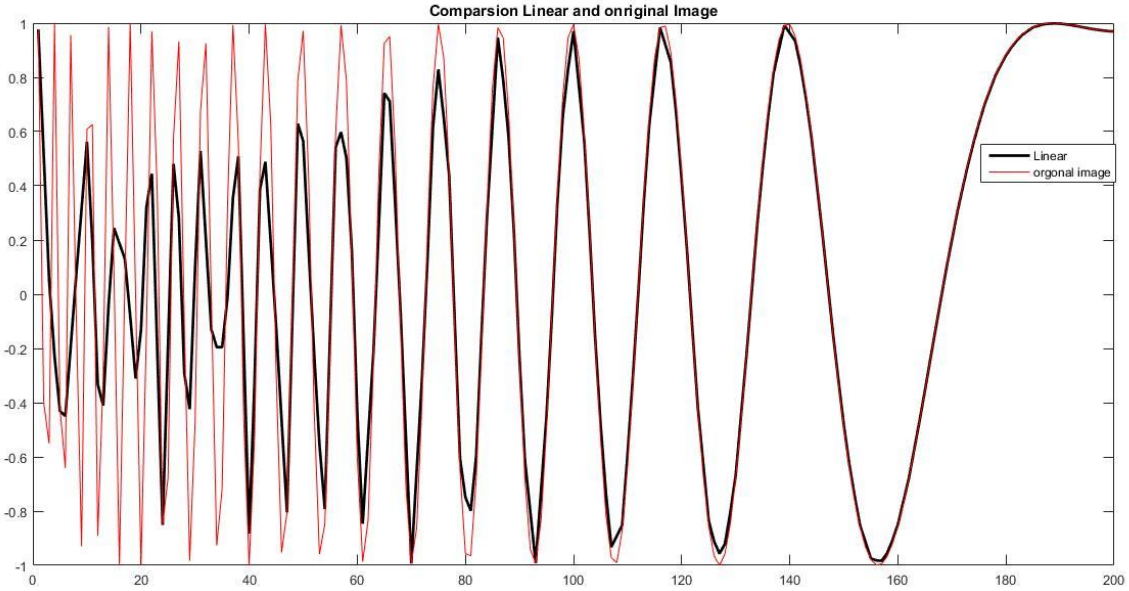
Binary show of cubic interpolation with S=2.3

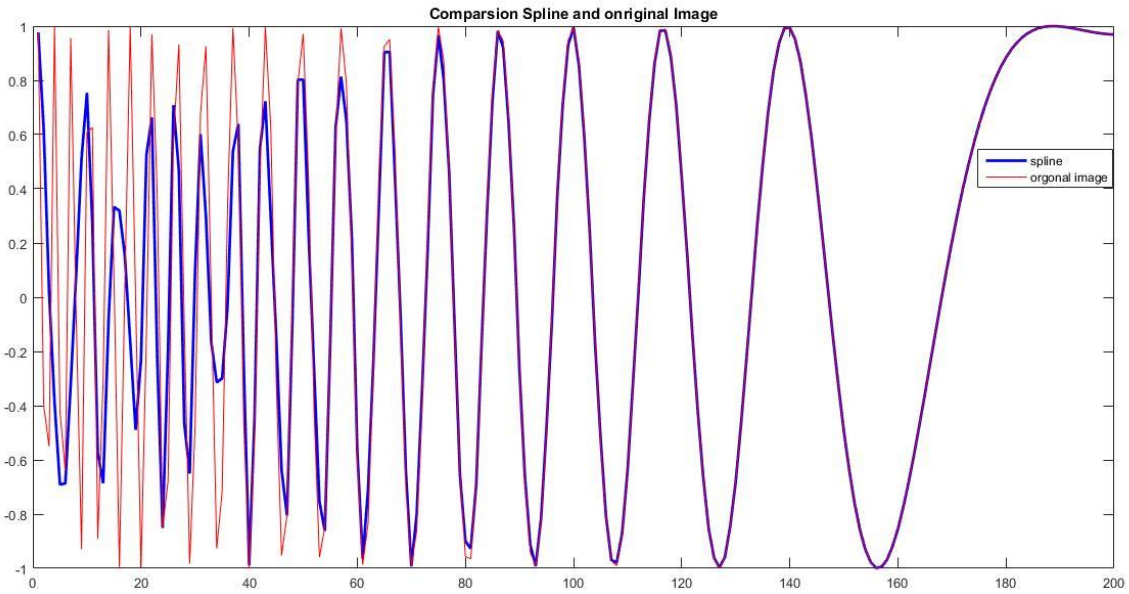


Binary show of linear interpolation with S=2.3



همانطور که مشاهده می کنید، با مقایسه ی نتیجه ی حاصل از سه روش و مقایسه ی آن ها با تصویر اصلی، به این نتیجه میرسیم که مطابق انتظار ، روش نزدیک ترین همسایه نسبت به دو روش دیگر بدتر عمل کرد و خطای ۲۱ درصدی ایجاد شد. روش خطی نتیجه را بسیار بهبود بخشید و به ۱۱ درصد کاهش داد ولی برخلاف انتظار، روش درونیابی مرتبه ی سه خطا را به ۱۲ درصد افزایش داد. در توجیه این پدیده، می توان گفت که از آنجایی که در فرکانس های بالا یعنی در کناره های تصویر، سینوس خیلی تیز شده و در واقع تقریبا رفتار خطی از خود نشان می دهد، لذا به نظر می آید که روش خطی در فرکانس های بالا بهتر توانسته تابع سینوسی را تخمین زده و به همین خاطر خطای کمتری را مشاهده می نماییم. البته معیار فاصله برای محاسبه ی خطا نمی تواند کاملا مناسب هم باشد چرا که روش درونیابی مرتبه ی سوم در فرکانس های پایین تر بهتر و دقیق تر از روش خطی عمل می نماید و تنها در فرکانس های بالا مقداری خطا دارد و ما نمیتوانیم دقت بالای روش مرتبه ی سوم را در فرکانس های پایین نادیده بگیریم و این موضوع نمی تواند دقت روش مرتبه ی سوم را انکار کند. در شکل های زیر، که تخمین دو روش خطی و مرتبه ی سوم از سینوسی های متحد المرکز در فرکانس های بالا و پایین است این موضوع کاملا مشهود است:

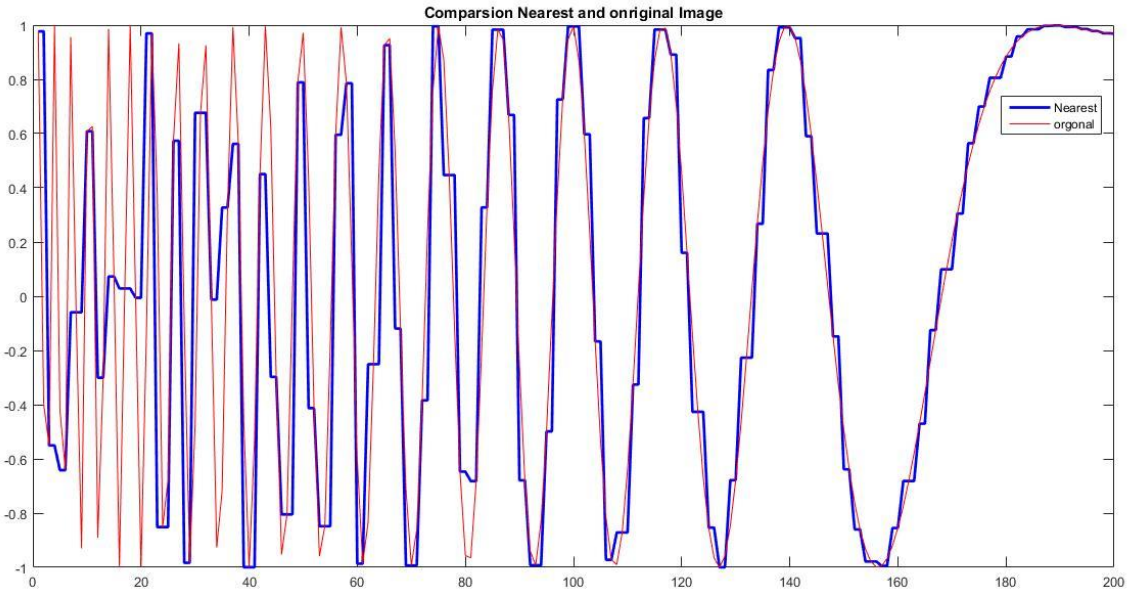




ند

همانطور که در دو تصویر بالا مشاهده می کنید دو روش تفاوت خیلی چندانی با هم ندارند اما به میزان خیلی کمی روش درونیابی مرتبه ی سوم در فرکانس های پایین بهتر عمل کرده است و در فرکانس های بالا، روش خطی به مقدار بسیار اندکی بهتر عمل کرده است.

برای اینکه بدی روش نزدیک ترین همسایه هم معلوم شود، برای یک سطر چگونگی تخمین تابع سینوسی توسط این روش را هم رسم نموده ایم که به صورت زیر می باشد:



مشاهده می شود ک نتیجه، اصلا خوب نمی باشد که در تصویر اسکیل شده هم این موضوع کاملا مشخص بود.

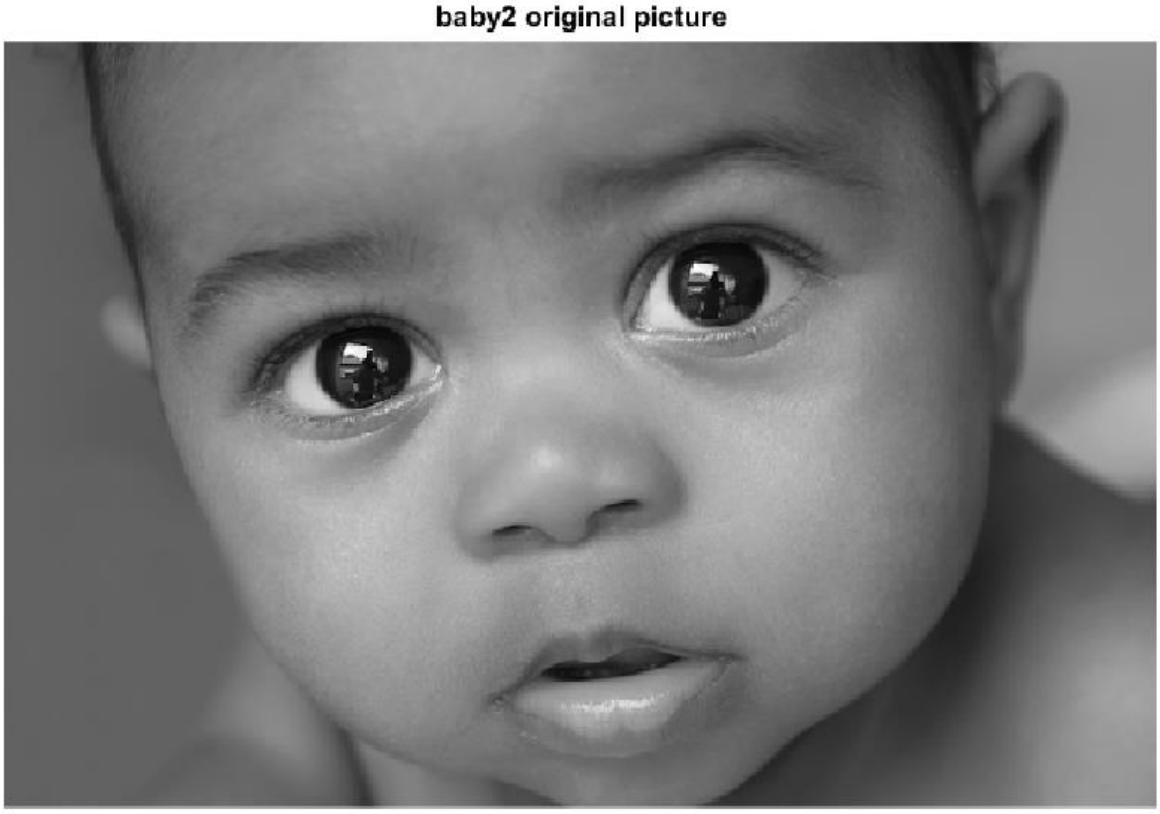
سوال دوم) :(کد مربوطه در فایل HW2_Q2.m قرار دارد):

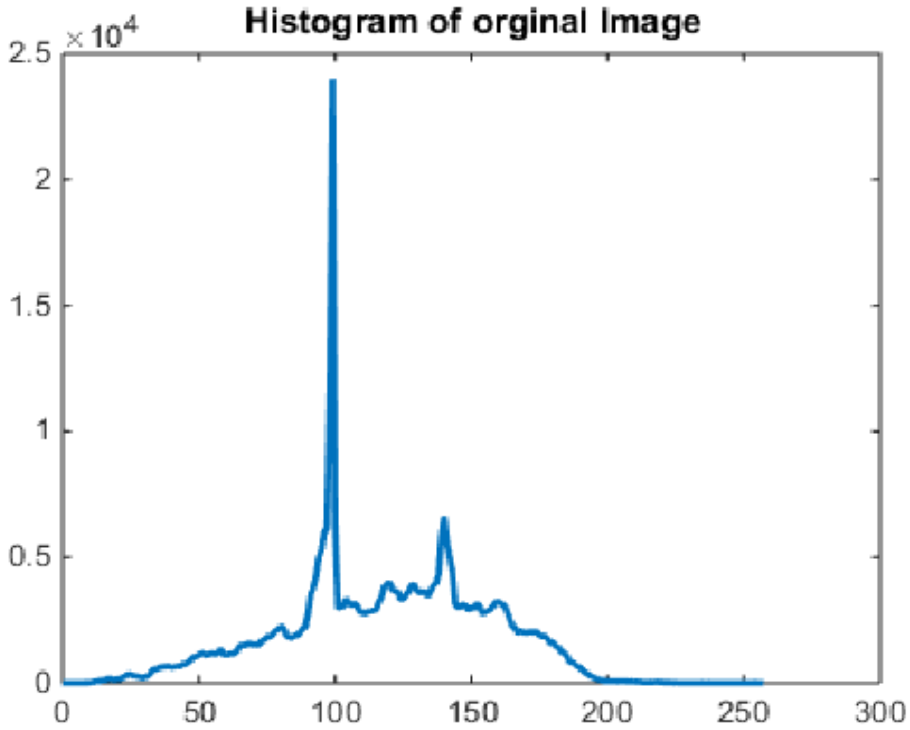
در این سوال، قرار است با استفاده از دو روش uniform و Lloyd-max تعداد سطوح کوانتیزاسیون تصویر مورد نظر را به ۱۶ کاهش دهیم . با این کار به جای اینکه برای هر پیکسل، از ۸ بیت ($256 = 2^8$) استفاده نماییم، از ۴ بیت استفاده می کنیم.

توسط کد زیر، تصویر اصلی و هیستوگرام آن را رسم می نماییم:

```
L=16;
b2=imread('baby2.jpg');
b2g=rgb2gray(b2);
figure
imshow(b2g,[])
title('baby2 original picture')

for i=0:255
   hist(i+1) = length(find(b2g == i));
end
figure,plot(hist,'LineWidth',2),title('Histogram of orginal Image')
```

که تصویر اصلی و هیستو گرام آن به شکل زیر می باشد:

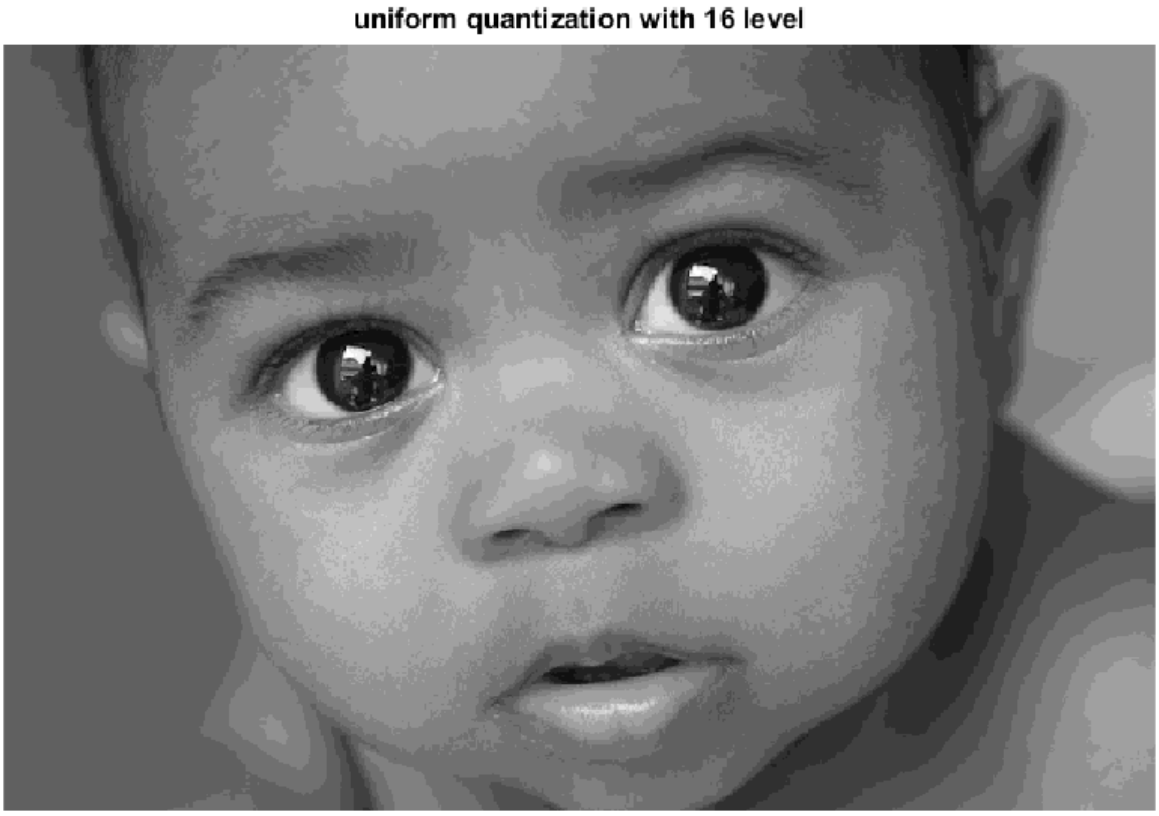




در ابتدا با استفاده از روش uniform، تعداد سطوح روشنایی را به ۱۶ تا کاهش می دهیم. کد مربوطه به صورت زیر می باشد:

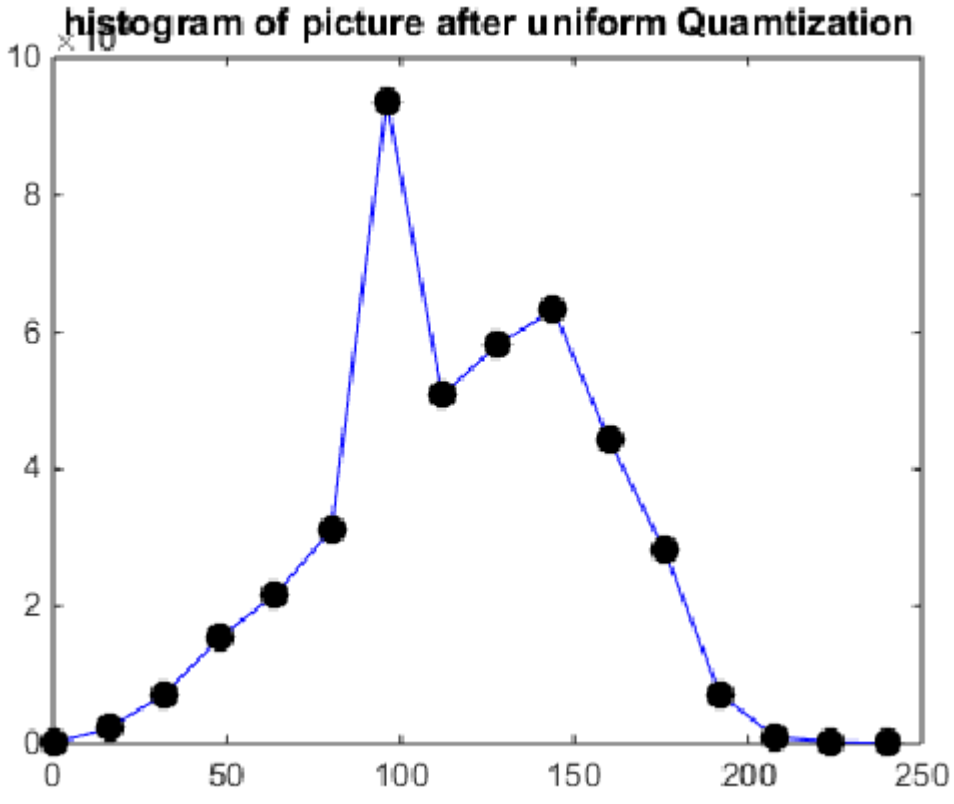
```
%% uniform quantization
f_uniform=fix(b2g/(256/L))*16;
figure,imshow(f_uniform)
title(['uniform quantization with ',num2str(L),' level'])
r_uniform=0:16:240;
for i=1:length(r_uniform)
    hist_uniform(i) = length(find( f_uniform == r_uniform(i) ));
end
figure, plot(r_uniform,hist_uniform,'b'),hold on
plot(r_uniform,hist_uniform,'k.','MarkerSize',30) %histogram of uniform quantization
title('histogram of picture after uniform Quamtization')
ti=fix([0:255/L:255]);
disp('ti of uniform quantization')
disp(ti)
[M,N]=size(f_uniform);
```

پس از اجرای برنامه، نتیجه ی حاصل به صورت زیر خواهد بود:



برای مقایسه ی بهتر تصویر بالا با تصویر اصلی، هیستوگرام تصویر بالا را نیز رسم می نماییم که روش رسم آن در کد بالا ذکر شده است.

هیستوگرام تصویر بالا که از طریق روش uniform به دست آمده است، به صورت زیر خواهد بود:



میزان خطای ایجاد شده از طریق روش uniform را نیز حساب می کنیم به صورت زیر :

```
error_Uniform =  ((norm(f(:)-f_uniform1(:)))/norm(f(:)))*100;
```

که حدود 4.22 درصد خواهد شد.

همچنین مقادیر ti نیز کاملا یونیفرم بوده و به صورت زیر به دست می آید:

ti of uniform quantization

0 15 31 47 63 79 95 111 127 143 159 175 191 207 223 239 255

و مقدار ri ها نیز به صورت زیر خواهد بود:

ri _uniform=

0 16 32 48 64 80 96 112 128 144 160 176 192 208 224 240

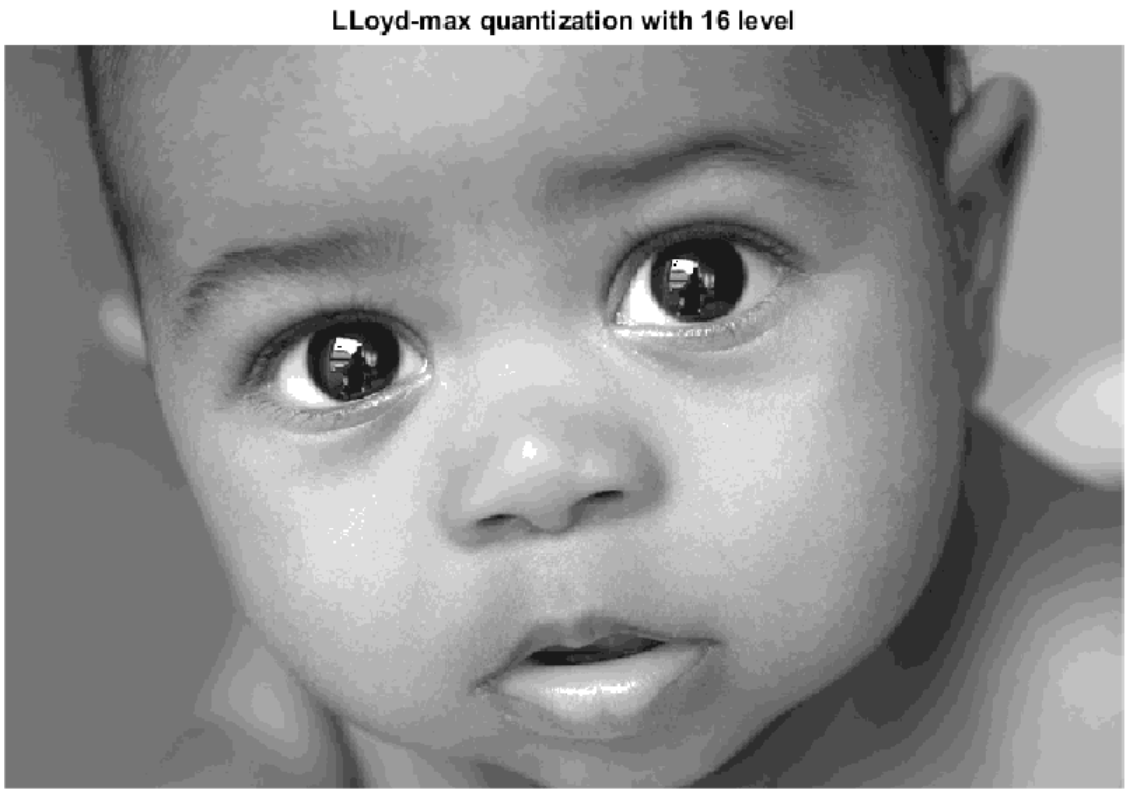
در ادامه برای کاهش سطوح کوانتیزاسیون از روش Lloyd استفاده می نماییم.

کد برنامه برای رسم شکل و هیستوگرام آن به صورت زیر می باشد:

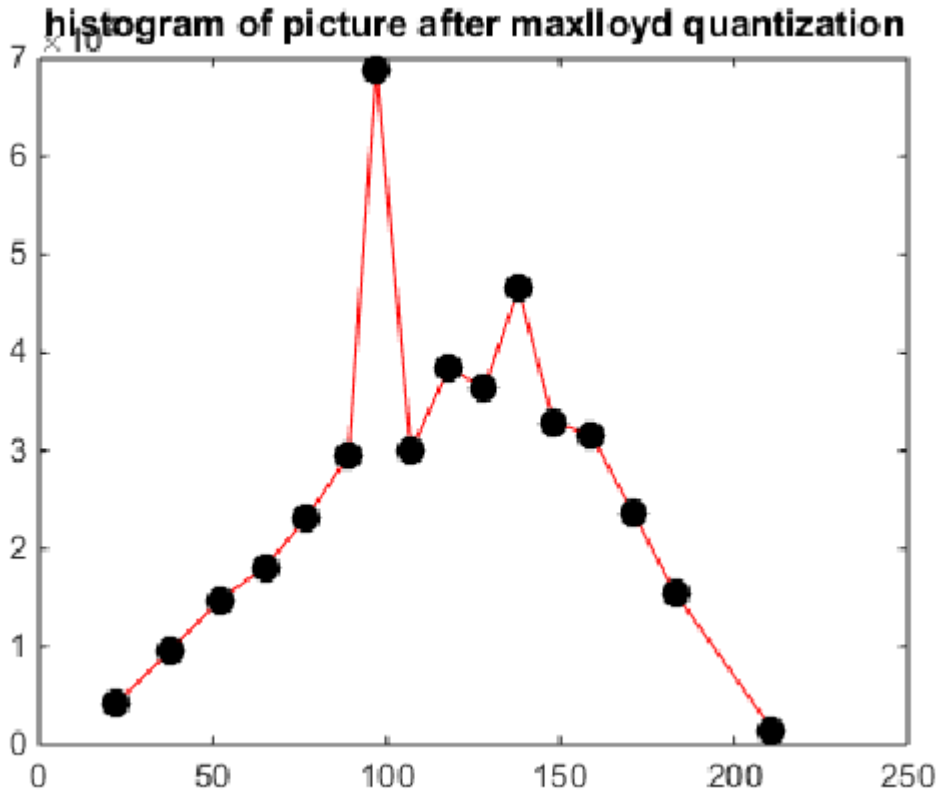
```
%% Max lloyd quantization
p=0;
h=imhist(b2g);
error=100;
while error>.1
    p=p+1;
    for i=1:L
        v=h(fix(ti(i):ti(i+1))+1);
        r(i)=[ti(i):ti(i+1)]*v/sum(v);
    end

    for j=2:L
        tt(j)=.5*(r(j-1)+r(j));
    end

    tt(L+1)=255;
    error=sum((tt-ti).^2);
    ti=tt;
    if error<.1
        break;
    end

end

f_LLoyd = zeros(M,N);
for l=2:L+1
    temp = find( ti(l-1)<b2g & b2g < ti(l));
    f_LLoyd(temp) = fix(r(l-1));
end
figure
imshow(f_LLoyd,[])
title(['LLoyd-max quantization with ',num2str(L),' level'])
fix_r = fix(r);
```

```
for i=1:length(fix_r)
    hist_LLoyd(i) = length(find( f_LLoyd == fix_r(i) ));
end
figure, plot(fix_r,hist_LLoyd,'r'),hold on
plot(fix_r,hist_LLoyd,'k.','MarkerSize',30),title('histogram of picture after maxlloyd
quantization')
disp('ti of max lloyd quantization')
disp(fix(ti));
disp('ri of max lloyd quantization')
disp(r);
[M,N] = size(b2g);

f=zeros(M,N);
for i=0:255
   temp = find(b2g==i);
   f(temp) = i;
end
f_uniform1 =zeros(M,N);
for i=0:16:240
   temp =  find(f_uniform==i);
   f_uniform1(temp) = i;
end
```

شکل حاصل به صورت زیر می باشد:



پس از رسم هیستوگرام نیز نتیجه ی زیر را مشاهده می نماییم:



همچنین مقادیر ti و ri حاصل از کوانتیزیشن Lloyd به صورت زیر می باشد:

ti of max lloyd quantization

0 30 45 58 71 83 93 102 112 123 133 143 154 165 177 197 25

ri of max lloyd quantization

22.0127 38.7784 52.4845 65.3867 77.7824 89.9570 97.9741 107.5269 118.0774
128.2836 138.8868 148.9195 159.3776 171.0031 183.4583 211.5275

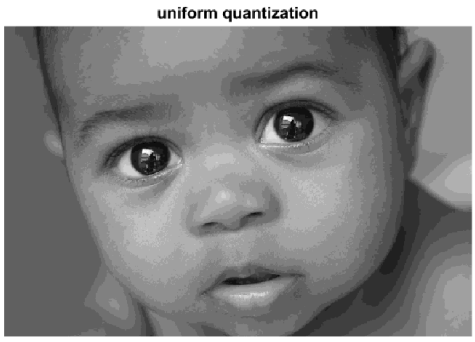
با محاسبه ی خطای این روش به صورت زیر:

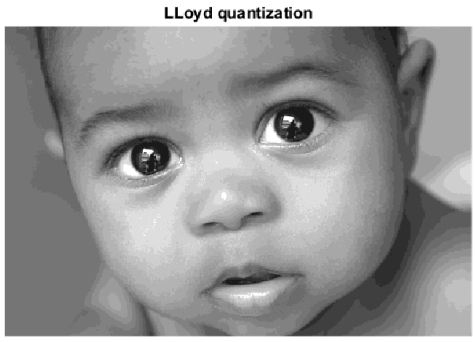
```
error_LLoyd = ((norm(f(:)-f_LLoyd(:)))/norm(f(:)))*100
```

مقدار خطا حدود ۲,۷ درصد خواهد بود.

بنابراین از روی خطا مشاهده می شود که روش Lloyd که در واقع سطوح روشنایی را به صورت غیر یکنواخت تفکیک می کند، بهتر عمل می نماید.

مقایسه ی تصاویر و هیستو گرام آنها نیز نشان می دهد که روش Lloyd بهتر عمل نموده است که در زیر کاملا مشهود است :





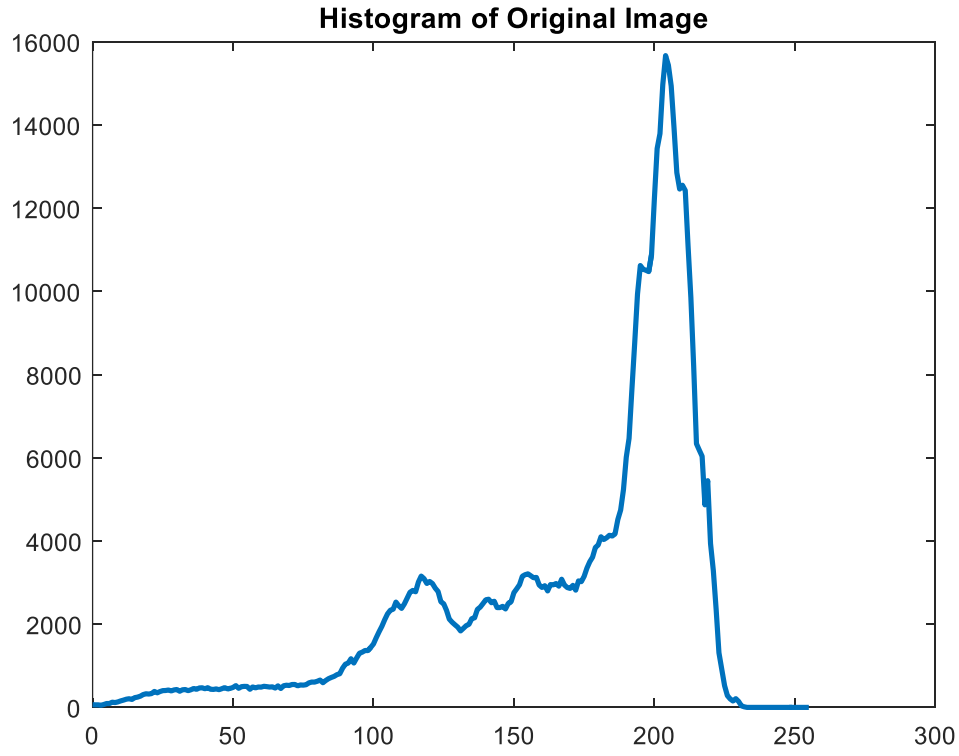
سوال ۳) کد این سوال در فایل HW2_Q3.m می باشد.

در قسمت اول سوال از ما خواسته شده است که تصویر را به دو روش بیان شده در درس کوانتیزه و آستانه بهینه را گزارش دهیم

در روش اول برای کوانتیزه کردن از روش تکرار و با در نظر گرفتن رابطه زیر برای T استفاده می کنیم (در ابتدا T را ۱۲۷ می گیریم و سپس از روش تکرار هر بار مقدار m1 , m2 را بروزرسانی و سپس مقدار T را دوباره محاسبه می کنیم و این کار را تا جایی ادامه می دهیم که میزان تغییرات T از یه حد ε برای مثال ۰,۰۱ کمتر گردد.)

$$T = \frac{m1+m2}{2}$$

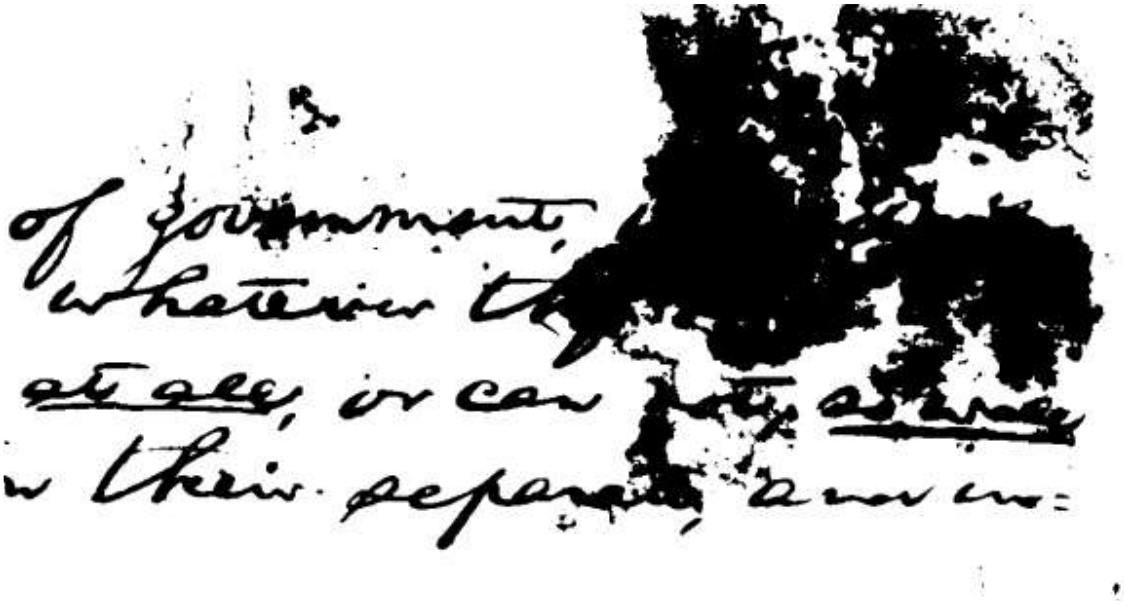
در زیر هستوگرام مربوط به شکل اصلی را شاهد هستیم:



با توجه به نمودار هیستوگرام فوق انتظار داریم مقدار T از مقدار ۱۲۷ اولیه بیشتر شود که مقدار به دست آمده از روش اول مطابق انتظار است و این روش مقدار T=150.898 بدست آورد.

در زیر شاهد تصویر بدست آمده از این روش هستیم

Binary Image using T mehtod



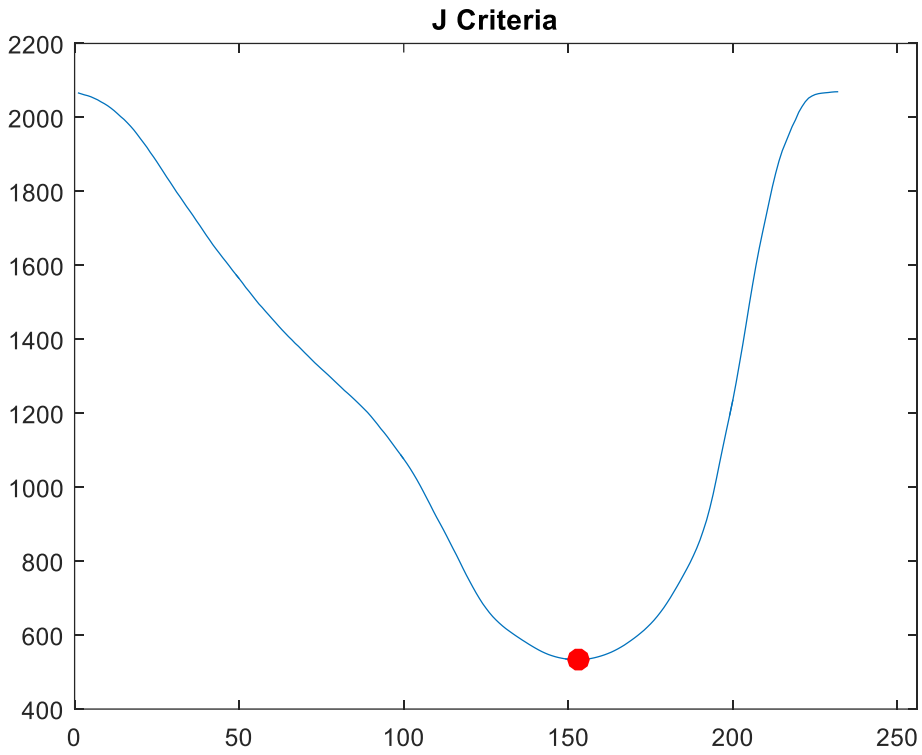
کد مربوط به این قسمت :

```
clc;
clear;
close all;

I = imread('text.jpg');
I = rgb2gray(I);
figure, imshow(I), title('Original Image');

hist = zeros(1,256);
for i=0:255
    hist(i+1) = length(find(I(:) == i));
end
figure, plot(0:255,hist, 'LineWidth',2);
title('Histogram of Original Image')

%%%%%%%%%%%%%%%% Binary, using T method
error = 1;
p = 0;
T=127;
while error > .01
    p= p +1;

    m_1(p) = (0:fix(T)-1)*(hist(1:fix(T))/sum(hist(1:fix(T)))).';
    m_2(p) = (fix(T):255)*(hist(fix(T)+1:256)/sum(hist(fix(T)+1:256))).';
    TT = .5*(m_1(p)+m_2(p));
    error(p) = norm(T-TT);
    T =TT;
end
T_T =T;
a= find(I>=0 & I<=T_T);  %%% indices that lower than T
b= find(I>T_T);          %%% indices taht greater than T
I_Binary_T  = I;
I_Binary_T(a) =0;
I_Binary_T(b) =255;
figure, imshow(I_Binary_T);
title('Binary Image using T  mehtod ')
```

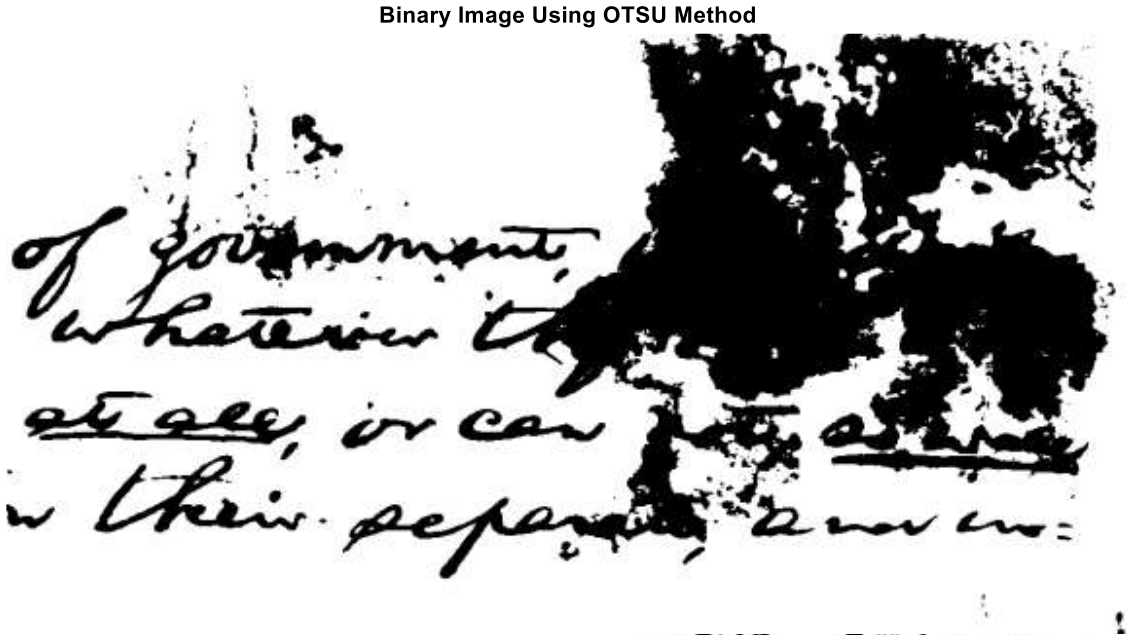
حال در قسمت دوم از روش OTSU برای باینری کردن تصویر استفاده کرده ایم در این روش برای محاسبه T به نام J تعریف کرده ایم که J را برای تمام مقادیر سطوح روشنایی (۰-۲۵۵) محاسبه میکنیم و در نهایت سطح روشنایی را به عنوان معیار انتخاب می کنیم که این معیار J را حداقل کند.

$J = p_1\sigma_1^2 + p_2\sigma_2^2$

در زیر شکل مربوط به معیار J را مشاهده می کنیم :



همانطور که در شکل فوق با دایره سرخ رنگ نشان داده شده است مقدار T در ۱۵۳ کمینه می‌شود در زیر هم شکل مربوط به باینری شده تصویر با استفاده از این معیار را شاهد هستیم :



در زیر کد مربوط به روش OTSU را شاهد هستیم :

```
%%%%%%%%%%%%%%%% Binary, using OTSU method
for i=1:256
    T=i;
    q_1 = sum(p(1:T));
    q_2 = sum(p(T+1:256));

    m_1 = (0:T-1)*p(1:T).';
    m_1 = m_1/q_1;
    m_2 = (T:255)*p(T+1:256).';
    m_2 = m_2/q_2;

    sigma_1 =0;
    sigma_1 = ((((0:T-1)-m_1).^2)*p(1:T).')/q_1;
    sigma_2 = 0;
    sigma_2 = (((T:255)-m_2).^2 * p(T+1:256).')/q_2;

    J(i)=q_1* sigma_1 + q_2*sigma_2;

end
min_index = find(J == min(J));
T_J= min_index(1);

figure, plot(J), xlim([0 256]),hold on;
plot(T_J,J(T_J),'r.','MarkerSize',30);
title('J Criteria')

a= find(I>=0 & I<=T_J);  %%% indices that lower than T
b= find(I>T_J);          %%% indices taht greater than T
I_Binary_OTSU  = I;
I_Binary_OTSU(a) =0;
I_Binary_OTSU(b) =255;

figure, imshow(I_Binary_OTSU);
title('Binary Image Using OTSU Method ')
```

در این قسمت از سوال از ما خواسته شده است که تصویر را به چهار قسمت تقسیم کرده و هر دو روش فوق را به این چهار قسمت اعمال کنیم و برای هر یک از این چهار تصویر یک آستانه بدست آوریم و سپس عملیات باینری کردن را برروی تصاویر انجام دهیم.

در ابتدا روش اول را (T) را به تصاویر اعمال میکنیم و ۴ آستانه را بدست آورده و ۴ تصویر باینری شده را در نهایت در کنار هم قرار می دهیم و یک تصویر واحد ایجاد میکنیم.

در پایین چهار زیر تصویر و هیستوگرام مربوط به این چهار شکل را شاهد هستیم :

Image - Top Left

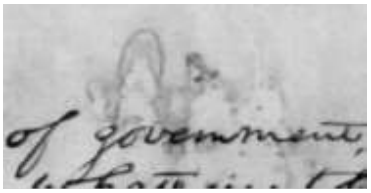


Image - Top Right

Image - Bottom Left

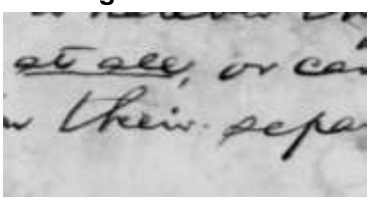
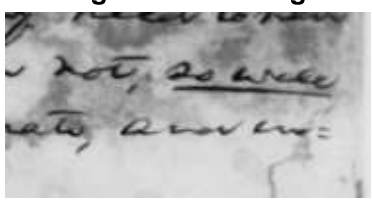
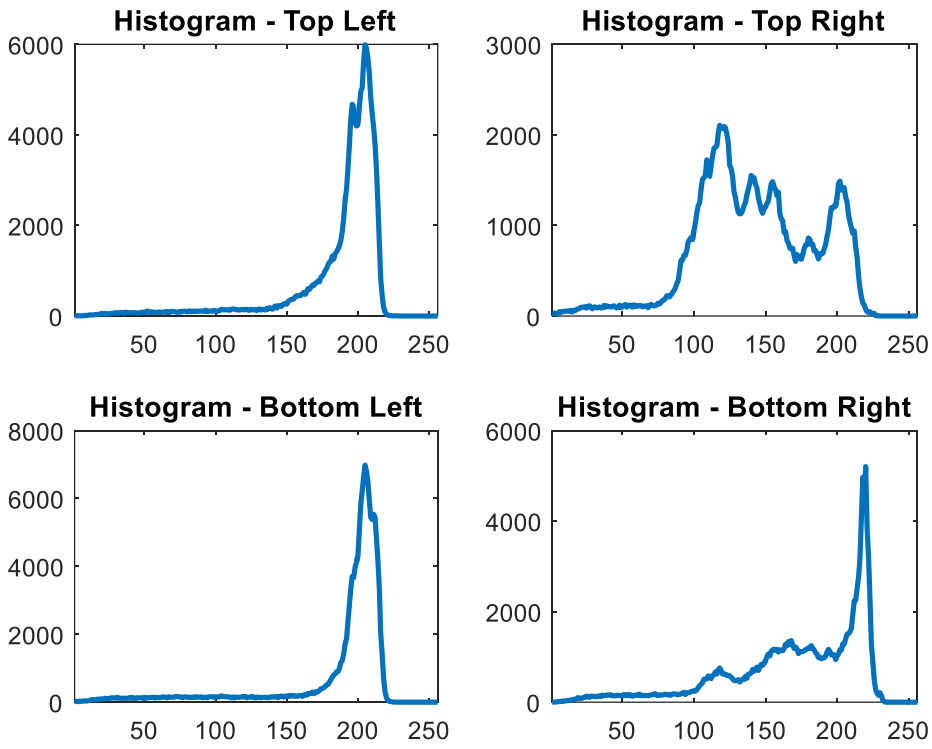


Image - Bottom Right





کد تقسیم عکس به ۴ زیر بخش به صورت زیر است:

```
%%%%%%%%%%%%%%%%%%%%%%%%%    Devide Image to 4 part
[M,N] = size(I);
I_4{1} = I(1:fix(M/2),1:fix(N/2));            %%%% Top left Image
I_4{2} = I(1:fix(M/2),fix(N/2)+1:N);          %%%% Top right Image
I_4{3} = I(fix(M/2)+1:end,1:fix(N/2));        %%%% Bottom left Image
I_4{4} = I( fix(M/2)+1:end,fix(N/2)+1:N);     %%%% Bottom right Image

for i=1:4
    I_temp = I_4{i};
    for j=0:255
        temp(j+1) = length(find(I_temp(:) == j));
    end
    hist_4{i} = temp;
end
%%%%%%%%%%%%%%%%%%%%%%

%%%%% plotting Image after deviding to 4 part
figure,subplot(221),imshow(I_4{1}),title('Image - Top Left')
subplot(222),imshow(I_4{2}),title('Image - Top Right')
subplot(223),imshow(I_4{3}),title('Image - Bottom Left')
subplot(224),imshow(I_4{4}),title('Image - Bottom Right')

%%%%% Histogram of Image after deviding to 4 part
figure,subplot(221),plot(hist_4{1},'LineWidth',2),title('Histogram - Top Left'),xlim([1 256])
subplot(222),plot(hist_4{2},'LineWidth',2),title('Histogram - Top Right'),xlim([1 256])
subplot(223),plot(hist_4{3},'LineWidth',2),title('Histogram - Bottom Left'),xlim([1 256])
subplot(224),plot(hist_4{4},'LineWidth',2),title('Histogram - Bottom Right'),xlim([1 256])
```

حال باید آستانه را برای چهار هیستوگرام فوق بدست آوریم:

Top-Left	Top-Right
۱۴۰,۹۹۱۶	۱۴۴,۷۶۶
Bottom-Left	Bottom-Right
۱۳۷,۹۶۵	۱۴۹,۸۴۲۵

در زیر شاهد تصاویر این ۴ قسمت در ابتدا به صورت جداگانه و همچنین تصویر واحد ایجاد شده پس از قرار دادن این ۴ تصویر در کنار هم هستیم:

Image Binary - Top Left

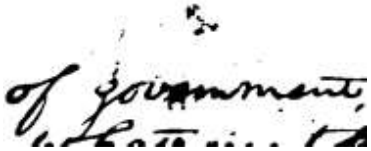


Image Binary - Top Right

Image Binary - Bottom Left

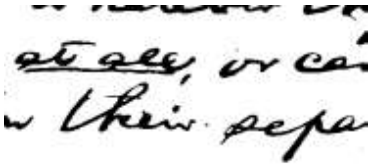
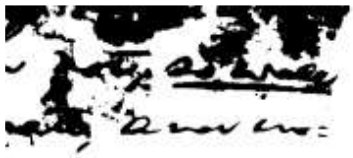
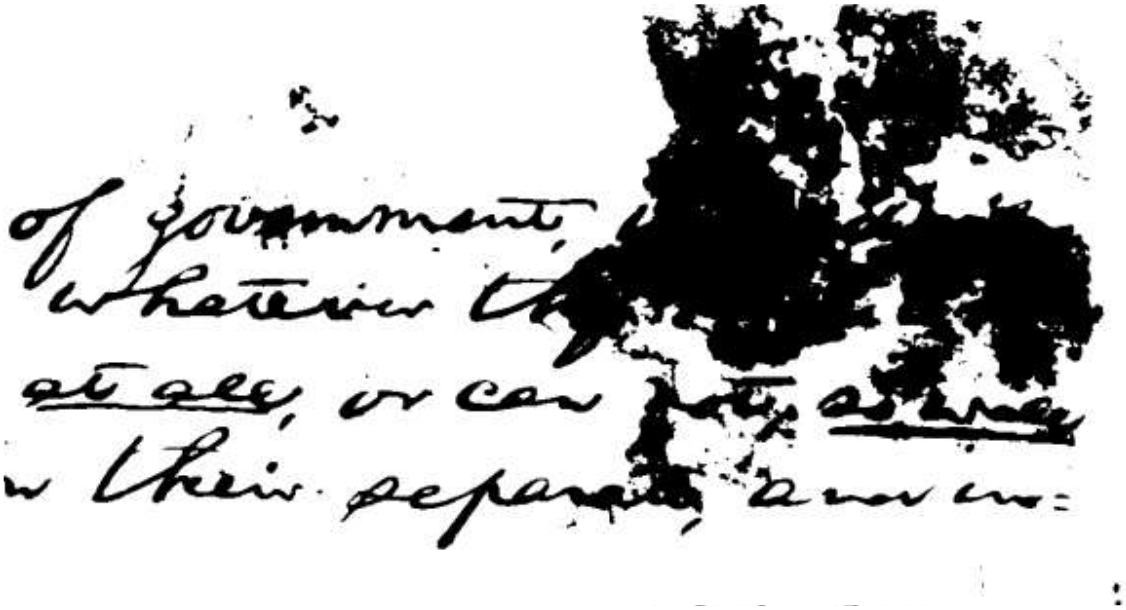


Image Binary - Bottom Right



Merge 4 Image together anfter using Binary Method(T)

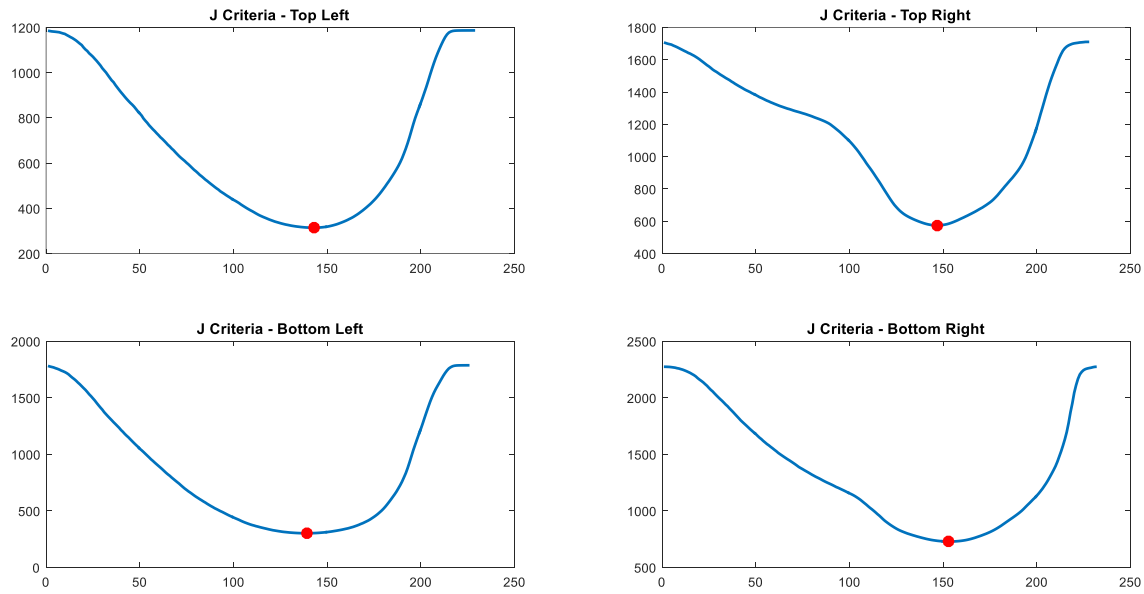


کد مربوط به این قسمت را در زیر شاهد هستیم :

```
%%%%%%%%%%%%%%%%% Binary metod 4 Image
error = 1;

for i=1:4
    hist=[];
    hist = hist_4{i};
    p = 1;
    T=127;
```

```
    e=[]; e=10;
    while e > .01

        m_1 = (0:fix(T)-1)*(hist(1:fix(T))/sum(hist(1:fix(T)))).';
        m_2 = (fix(T):255)*(hist(fix(T)+1:256)/sum(hist(fix(T)+1:256))).';
        TT = .5*(m_1 + m_2);
        e(p) = norm(T-TT);
        T =TT;
        p= p +1;
    end
    T_T_4(i) = T;
    error_4{i} = e;
end

for i=1:4
    temp = I_4{i};
    a= find(temp>=0 & temp<=T_T_4(i));  %%% indices that lower than T
    b= find(temp > T_T_4(i));           %%% indices taht greater than T
    temp(a) =0;
    temp(b) =255;
    I_Binary_T_4{i}  = temp;
end

figure, subplot(221),imshow(I_Binary_T_4{1});title('Image Binary - Top Left')
subplot(222),imshow(I_Binary_T_4{2});title('Image Binary - Top Right')
subplot(223),imshow(I_Binary_T_4{3});title('Image Binary - Bottom Left')
subplot(224),imshow(I_Binary_T_4{4});title('Image Binary - Bottom Right')

I_B_4_T = [I_Binary_T_4{1} I_Binary_T_4{2}; I_Binary_T_4{3} I_Binary_T_4{4}]; %%% merge 4 Image together
figure , imshow(I_B_4_T),title('Merge 4 Image together anfter using Binary Method(T)')
```

در این قسمت روش OTSU را به چهار زیر شکل اعمال می کنیم، باز مطابق قبل ابتدا باید معیار J را بدست آوریم و سپس اندیسی را که این معیار را حداقل می کند را به عنوان T انتخاب کنیم. در زیر نمودار های معیار J مرتبط با ۴ زیر شکل را مشاهده می کنیم:



همانطور که با رنگ قرمز در شکل فوق مشخص شده است مقدار اندیسی که مربوط به هر قسمت را حداقل می‌کند به صورت زیر می باشد:

Top-Left	Top-Right
143	147
Bottom-Left	Bottom-Right
139	153

حال اگر تصاویر را با استفاده از مقدار به دست آمده در فوق باینری کنیم، تصاویر زیر بدست می آید:

Image Binary(OTSU method) - Top Left

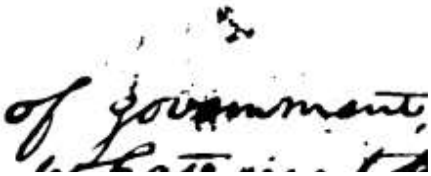


Image Binary(OTSU method) - Top Right

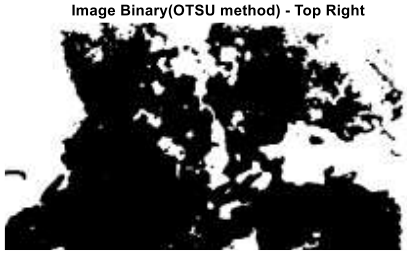


Image Binary(OTSU method) - Bottom Left

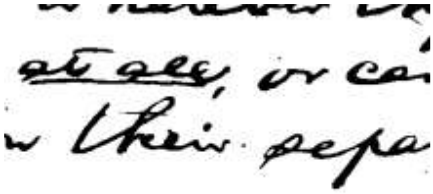
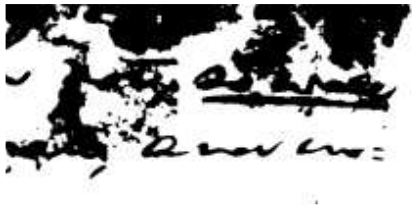
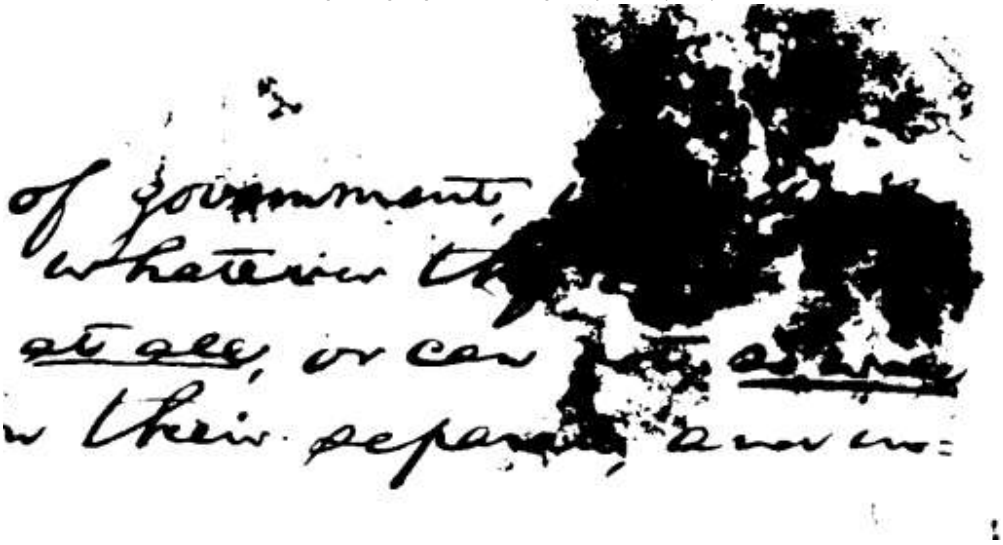


Image Binary(OTSU method) - Bottom Right



و تصویر حاصل از ترکیب ۴ عکس فوق به صورت زیر خواهد بود:

Merge 4 Image together after using Binary Method(OTSU)

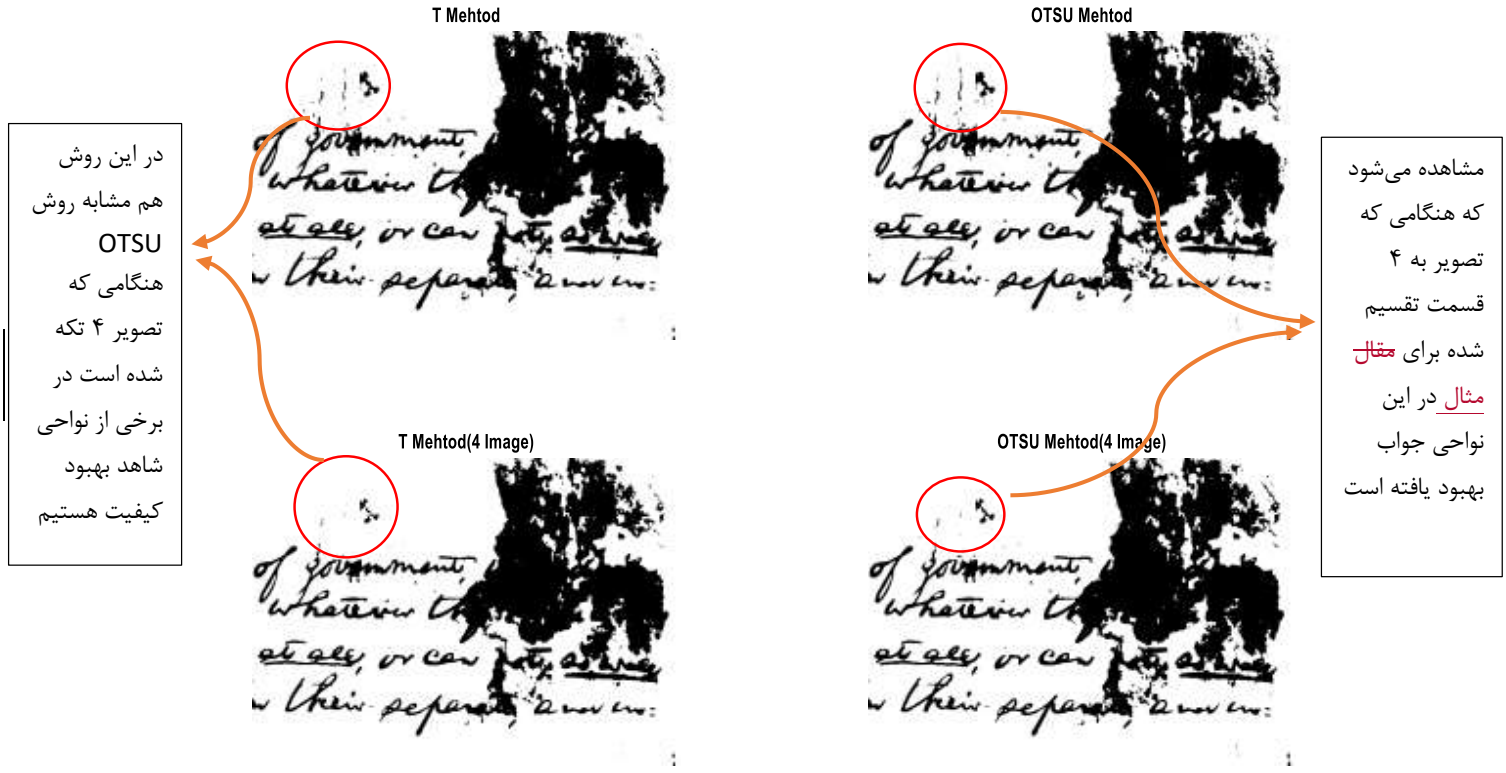


در زیر نیز کد مربوط به این قسمت را شاهد هستیم:

```
%%%%%%%%%%%%%%%% OTSU  4 Image
for j=1:4
    hist = hist_4{j};
    p=hist/sum(hist);
    for i=1:256
    T=i;
    q_1 = sum(p(1:T));
    q_2 = sum(p(T+1:256));

    m_1 = (0:T-1)*p(1:T).';
    m_1 = m_1/q_1;

    m_2 = (T:255)*p(T+1:256).';
    m_2 = m_2/q_2;

    sigma_1 =0;
    sigma_1 = ((((0:T-1)-m_1).^2)*p(1:T).')/q_1;
    sigma_2 = 0;
    sigma_2 = (((T:255)-m_2).^2 * p(T+1:256).')/q_2;
```

```
    J_4(j,i)=q_1* sigma_1 + q_2*sigma_2;

    end
end

for i=1:4
    min_index = find(J_4(i,:) == min(J_4(i,:)));
    T_J_4(i)= min_index(1);
end

figure,subplot(221),plot(J_4(1,:), 'LineWidth',2),title('J Criteria - Top Left'),hold on;
plot(T_J_4(1),J_4(1,T_J_4(1)),'r.','MarkerSize',30);
subplot(222),plot(J_4(2,:), 'LineWidth',2),title('J Criteria - Top Right'),hold on;
plot(T_J_4(2),J_4(2,T_J_4(2)),'r.','MarkerSize',30);
subplot(223),plot(J_4(3,:), 'LineWidth',2),title('J Criteria - Bottom Left'), hold on;
plot(T_J_4(3),J_4(3,T_J_4(3)),'r.','MarkerSize',30);
subplot(224),plot(J_4(4,:), 'LineWidth',2),title('J Criteria - Bottom Right'),hold on
plot(T_J_4(4),J_4(4,T_J_4(4)),'r.','MarkerSize',30);


for i=1:4
    a=[];b=[];
    a= find(I_4{i} >= 0 & I_4{i} <= T_J_4(i));    %%% indices that lower than T
    b= find(I_4{i} > T_J_4(i));                   %%% indices taht greater than T
    temp = I_4{i};
    temp(a) =0;
    temp(b) =255;
    I_Binary_J_4{i} = temp;
end

figure,subplot(221),imshow(I_Binary_J_4{1}),title('Image Binary(OTSU method) - Top Left')
subplot(222),imshow(I_Binary_J_4{2}),title('Image Binary(OTSU method) - Top Right')
subplot(223),imshow(I_Binary_J_4{3}),title('Image Binary(OTSU method) - Bottom Left')
subplot(224),imshow(I_Binary_J_4{4}),title('Image Binary(OTSU method) - Bottom Right')

I_B_J_4 = [I_Binary_J_4{1} I_Binary_J_4{2}; I_Binary_J_4{3} I_Binary_J_4{4}];
figure,imshow(I_B_J_4),title('Merge 4 Image together after using Binary Method(OTSU)')
```

برای بررسی بهتر تغییرات روش‌های مختلف، تصاویر مربوط به این ۴ روش را در تصویر هم در کنار یکدیگر قرار داده ایم:



به طور کلی می‌توان نتیجه گرفت هنگامی که تصویر را به ۴ قسمت تقسیم می کنیم در هر دو روش با توجه به اینکه از هیستوگرام همان قسمت از تصویر استفاده میکنیم، شاهد بهبود ~~کمی~~ در نصاویر هستیم.

و لی با توجه به اینکه تصویر به نواحی زیادی تقسیم نشده است و از ~~یک~~ T گلوبال برای باینری کردن استفاده میکنیم، انتظار تغییر اساسی در کیفیت تصاویر نبود.

اما در کل با توجه به این که در هر دو روش از T گلوبال استفاده شده است شاهد هستیم هر دو روش که در برخی نواحی به خوبی نتوانسته اند پس زمینه را از متن تفکیک دهند و جواب هر دو ~~روش~~ تقریبا مشابه یکدیگر است و در برخی نقاط ممکن است یک روش بر دیگری مزیت نسبی داشته باشد اما در کل تفاوت قابل محسوسی وجود ندارد.