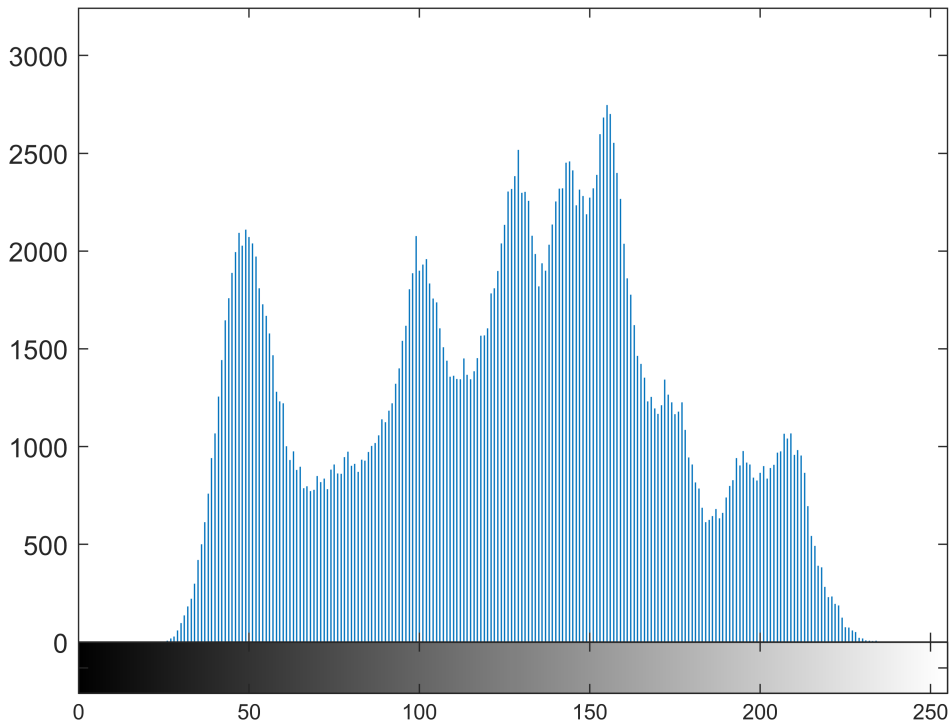
Cvičení 3 - Transformace intenzit

Histogram obrazku

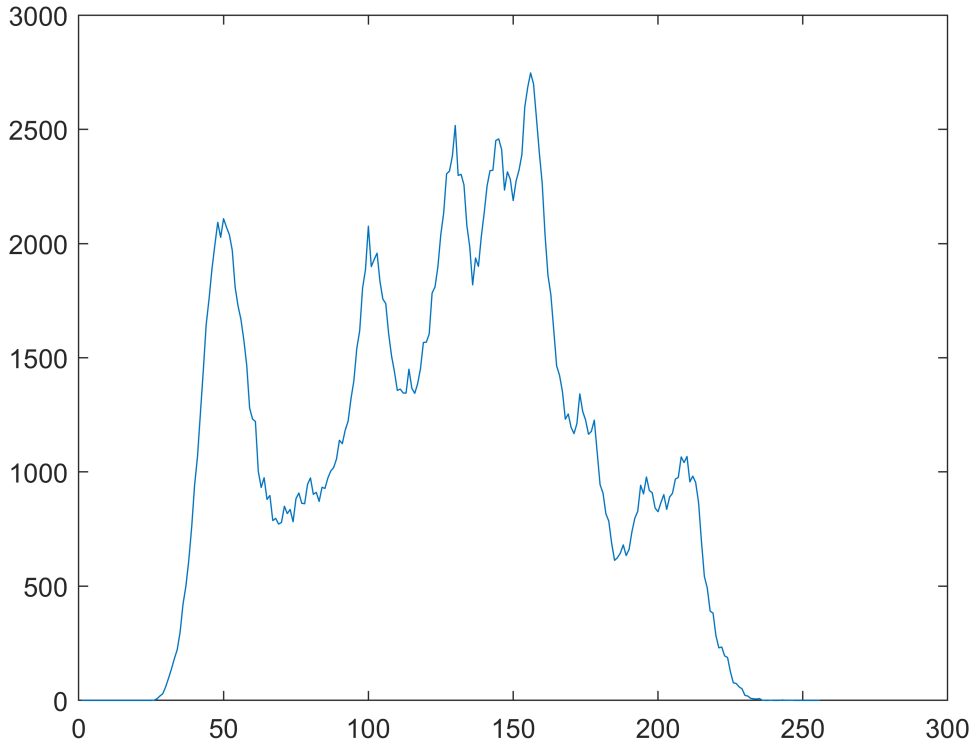
K popisu obrazu můžeme použít histogram. Je to diskrétní funkce, která každé intenzitě přiřadí počet pixelů v obraze s touto intenzitou. V matlabu můžeme histogram vykreslit pomocí funkcí `histogram()` nebo `imhist()`.

```
I = imread('lena_gray.png');
imhist(I)
```



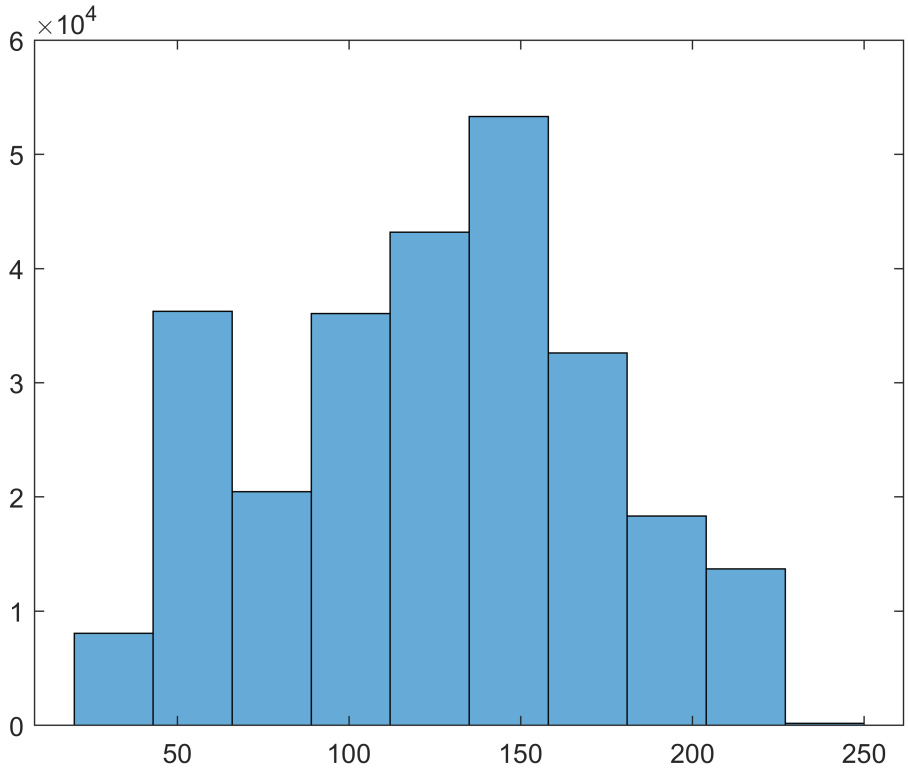
Případně si hodnoty uložit do nějaké proměnné a tu pak vykreslit.

```
histogramI = imhist(I);
figure, plot(histogramI);
```



Funkce histogram() se používá zejména pokud chceme takzvaný binding histogram (viz přednáška).

```
pocet_prihradek = 10;
figure, histogram(I,pocet_prihradek);
```

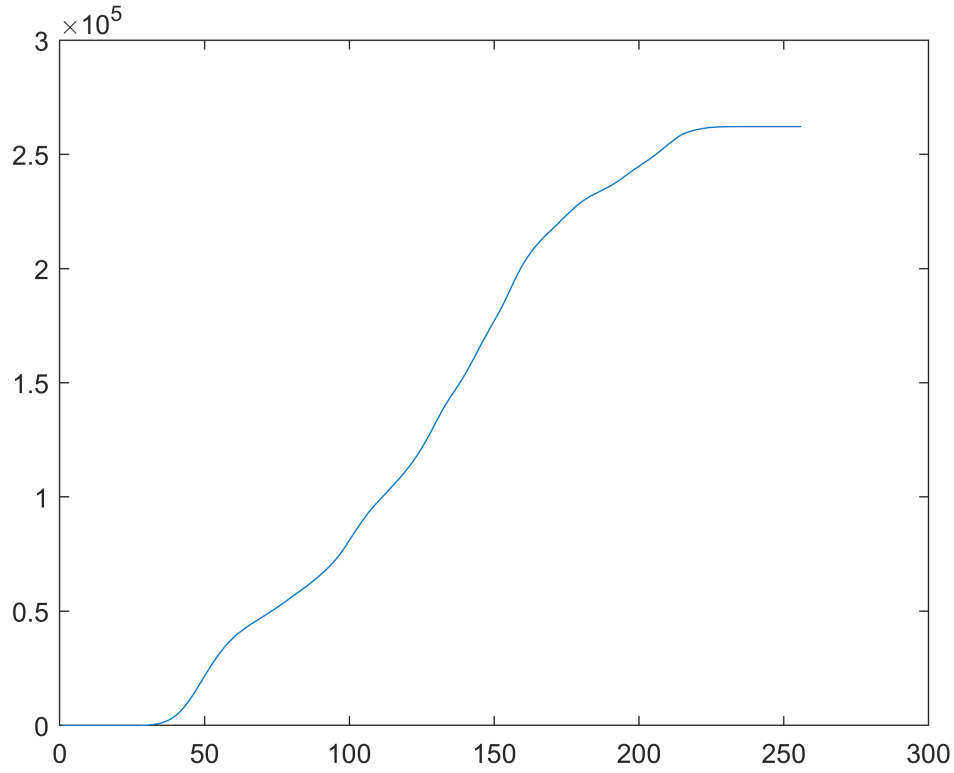


Také funkci imhist() je možné zadat počet přihrádek a tím získat binding histogram.

Kumulativní histogram

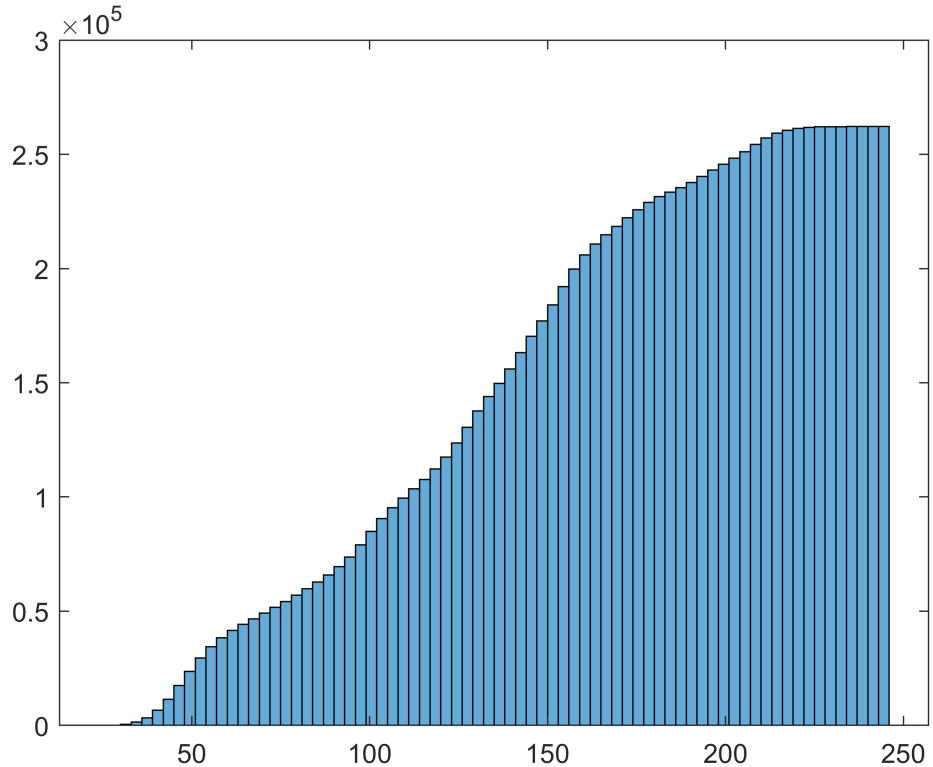
Kumulativní histogram uchovává informaci, kolik pixelů v obrázku má intenzitu menší rovnu každé intenzitě. Z histogramu vytvoříme kumulativní histogram za pomoci funkce `cumsum()`.

```
[pocet,~] = imhist(I);
cumh = cumsum(pocet);
figure, plot(cumh);
```



Připadně můžeme využít funkci histogram() následujícím způsobem.

```
figure, histogram(I,'Normalization','cumcount');
```



Transformace intenzit

$g(x, y) = T[f(x, y)]$

f ... vstupní obrázek

g ... výstupní obrázek

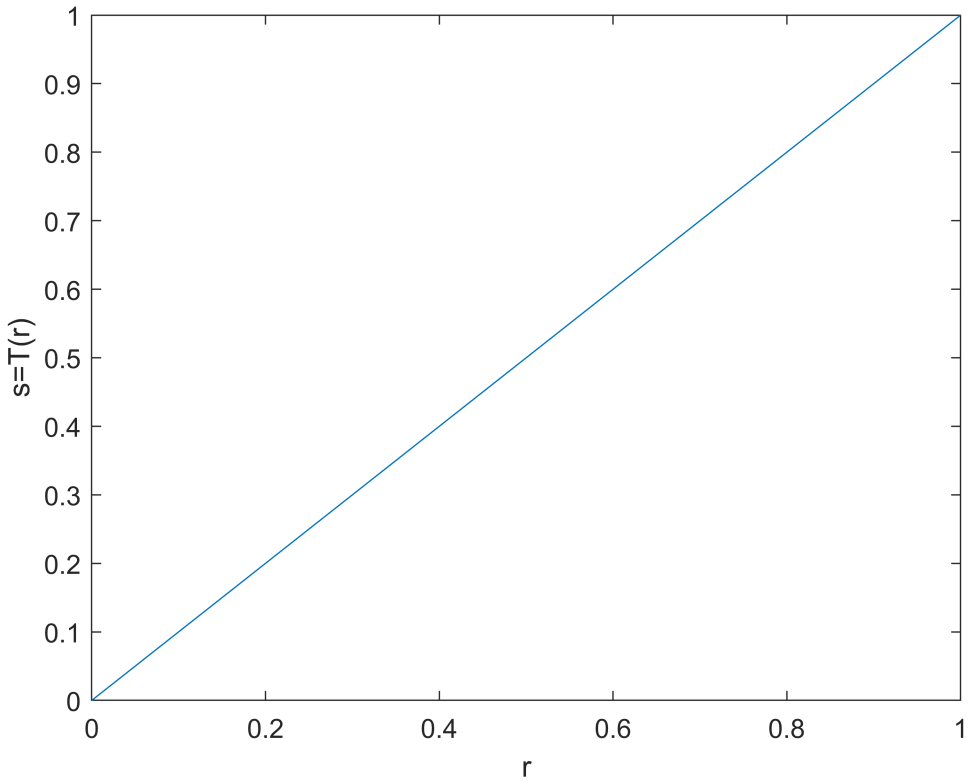
T - transformace barev

Transformaci zadáváme jako funkci (**transformační funkce**), která každé jasové hodnotě *r* přiřadí novou jasovou hodnotu *s*.

$s = T(r)$

Definice a vykreslení trasnformační funkce

```
r = (0:255)/255;
figure, plot(r,r);
xlabel('r');
ylabel('s=T(r)');
xlim([0,1]);
```



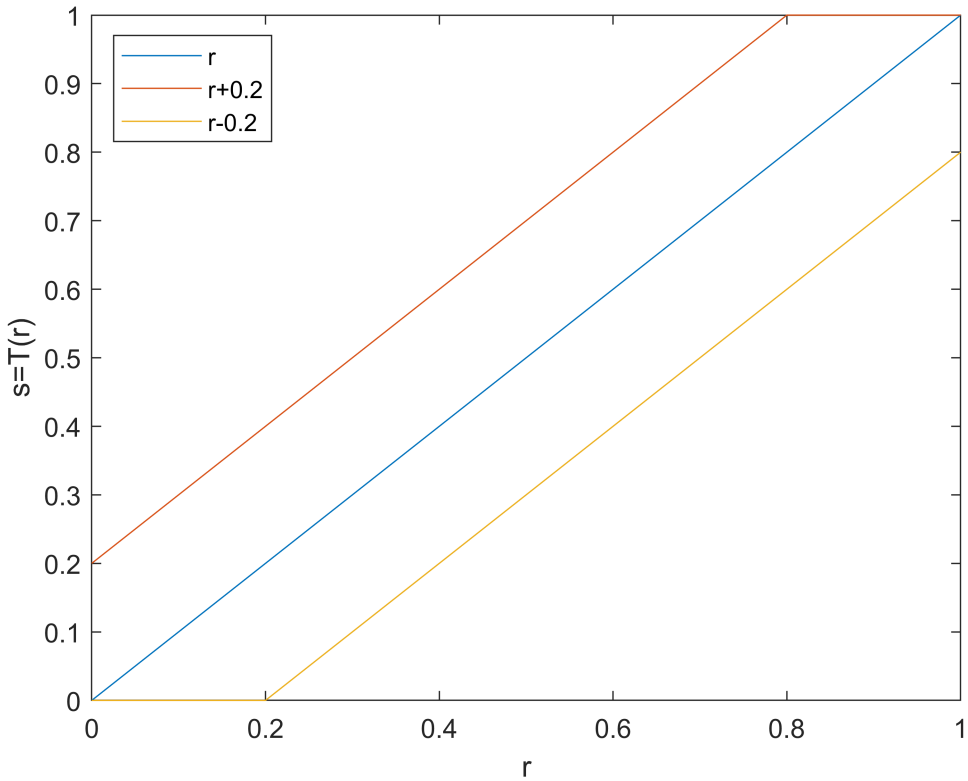
Pro obrázky typu uint8 jsou jasové hodnoty od 0 do 255. My tyto hodnoty převádíme na rozsah od 0 do 1.

Změna jasu

Transformační funkce je ve tvaru $s = r + k$, kde *k* je konstanta určující zda jas snižujeme, nebo zvyšujeme a jak moc. Výsledné hodnoty, které jsou menší než 0 jsou nastaveny na 0 a hodnoty vyšší než maximální intenzita na maximální intenzitu.

`clipMap()` je pomocná funkce, která převadí jasové hodnty menší než 0 na 0 a větší než 1 na 1.

```
k1= 0.2; % zvýšení jasu
k2 = -0.2; % snížení jasu


s1 = clipMap(r + k1,0,1);
s2 = clipMap(r + k2,0,1);

figure,
plot(r,r);
xlabel('r');
ylabel('s=T(r)');
xlim([0,1]);
hold on;
plot(r,s1);
plot(r,s2);
legend('r','r+0.2','r-0.2','Location','northwest');
```



Aplikace změny jasu na každý pixel

Využijeme maticových počtů.

`f` je matice intenzit obrázku. Je typu `uint8`, hodnoty v matici jsou od 0 (černá barva) do 255 (bílá barva).

Pokud bychom měli matici typu `double`, pak by se obrázek reprezentovaný touto maticí interpretoval tak, že 0 je nejmenší hodnota (černá barva) a 1 nejvyšší (bílá barva). Pro převod obrázků z typu `uint8` na `double` (a naopak) používáme funkci `im2double()` (respektive `im2uint8()`). Pouhé přetypování nemění hodnoty v matici. V případě `im2double()` dojde k přeškálování tak, že hodnoty větší rovny jedné se nahradí hodnotou 255, hodnoty mení rovny 0 se nahradí 0 a hodnoty mezi tím se pravidelně namapují na celočíselné hodnoty mezi tím.

```
f = imread('lena_gray.png');

% tic();
g = f + 50;
% toc()

figure,
subplot(1,2,1)
imshow(f)
title('Original')
subplot(1,2,2)
imshow(g)
title('k = 50')
```

Funkce `tic()` a `toc()`, můžeme použít ke změření doby běhu.

Ořezávat hodnoty menší než 0 a větší než 255 není potřeba. Výsledná matice je také typu `uint8` a k ořezání dojde automaticky.

Aplikace transformace na každý pixel pomocí cyklu

V tomto případě nepoužíváme součet matice a hodnoty. V cyklu procházíme jednotlivé pixely (celý obraz) a pro každý pixel počítáme novou jasovou hodnotu.

```
% tic();
h = uint8(zeros(size(f)));
for i = 1 : size(f,1)
    for j = 1 : size(f,2)
        h(i,j) = f(i,j) + 50;
    end
end
% toc()

figure,
subplot(1,2,1)
imshow(f)
title('Original')
subplot(1,2,2)
imshow(h)
title('k = 50')
```

zeros() vrací matici typu double, je nutné ji v našem případě přetypovat na `uint8`.

Odkomentováním funkcí `tic()` a `toc()` si můžeme ověřit, že průchod obrazu pixel po pixelu je pomalejší, než využití maticového počtu.

Využití lookup tabulky (palety)

Při aplikaci jasové transformace, jak bylo výše zmíněno, dochází k mnoha zbytečným výpočtům. Všechny pixely se stejnou vstupní intenzitou budou míte stejnou výstupní intenzitu. V praxi se vypočítají nové hodnoty pro všechny jasové hodnoty a tímto způsobem získáme takzvanou lookup tabulku (paletu). Jak aplikovat paletu na obrázek jsme si ukázali první cvičení.

```
% pomocna funkce pro vytvoreni palety
map1 = createMap(s1);
map2 = createMap(s2);

% Aplikace palety na obrazek
figure
subplot(1,3,1)
imshow(f)
title('Original')
subplot(1,3,2)
imshow(f,map1)
title('k = 0.2')
subplot(1,3,3)
imshow(f,map2)
```

```
title('k = -0.2')
```

Aplikace palety na obrázek v matlabu nezabírá žádný čas, takže pokud chceme srovnat dobu výpočtu, stačí nám zjistit dobu potřebnou pro výpočet palety.

```
tic();
x = createMap(s1);
toc()
```

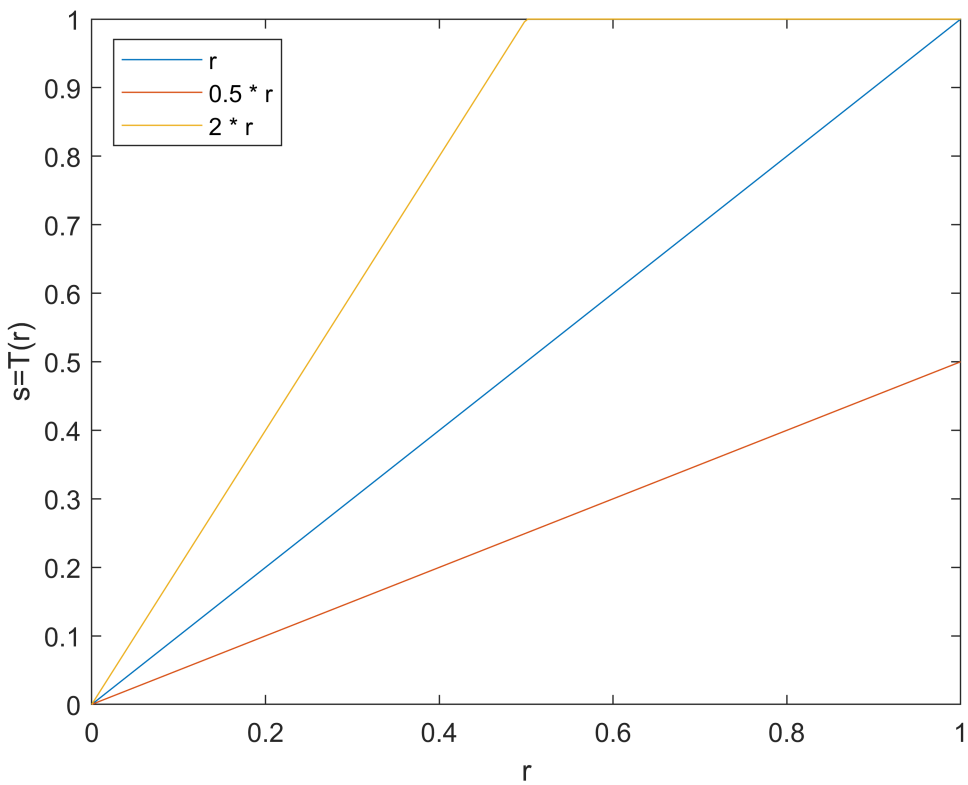
```
Elapsed time is 0.005269 seconds.
```

Změna kontrastu

Transformační funkce je ve tvaru $s = k \cdot r$, kde *k* je konstanta určující zda kontrast snižujeme, nebo zvyšujeme a jak moc.

```
c1 = 0.5;
c2 = 2;

s3 = clipMap(c1*r,0,1);
s4 = clipMap(c2*r,0,1);

figure,
plot(r,r);
xlabel('r');
ylabel('s=T(r)');
xlim([0,1]);
```

```
hold on;
plot(r,s3);
plot(r,s4);
legend('r','0.5 * r','2 * r','Location','northwest');
```



V praxi se změna kontrastu kombinuje spolu se změnou jasu.

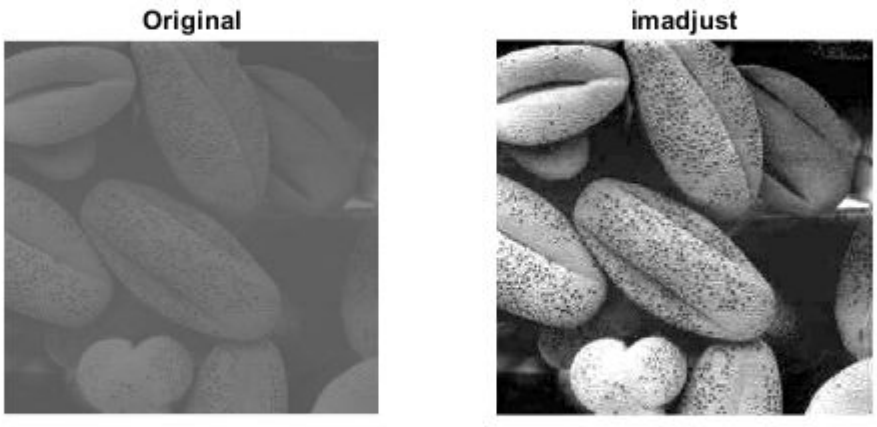
Aplikace změny kontrastu

```
map3 = createMap(s3);
map4 = createMap(s4);

figure
subplot(1,3,1)
imshow(f)
title('Original')
subplot(1,3,2)
imshow(f,map3)
title('c = 0.5')
subplot(1,3,3)
imshow(f,map4)
title('c = 2')
```

Roztažení kontrastu

Transformační funkce zvětšení kontrastu tak, že se hodnoty v obraze roztáhnou na celý možný interval hodnot. V matlabu je tato transformace implementována ve funkci `imadjust()`.

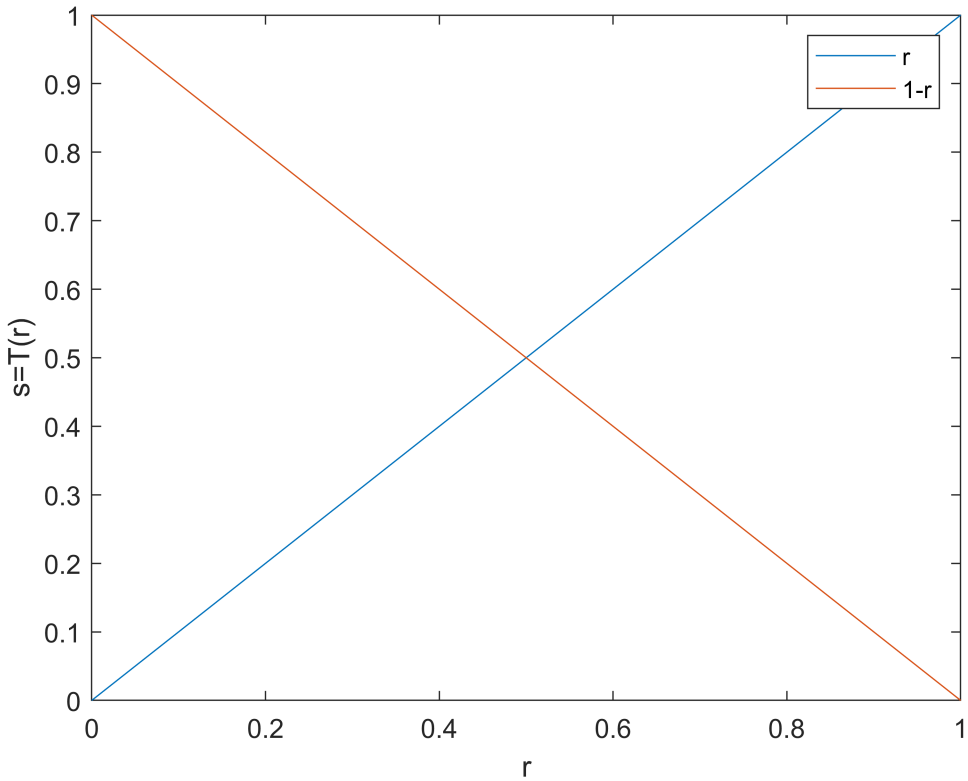
```
I = imread('pic1.png');
J = imadjust(I);

figure
subplot(1,2,1)
imshow(I);
title('Original')
subplot(1,2,2)
imshow(J);
title('imadjust')
```



Negativ obrázku

Jasová transformace má tvar $s = 1 - r$.

```
s5 = 1-r;
figure,
plot(r,r);
xlabel('r');
ylabel('s=T(r)');
xlim([0,1]);
hold on;
plot(r,s5);
legend('r','1-r','Location','northeast');
```




```
map5 = createMap(s5);

figure
subplot(1,2,1)
imshow(f)
title('Original')
subplot(1,2,2)
imshow(f,map5)
title('Negativ')
```

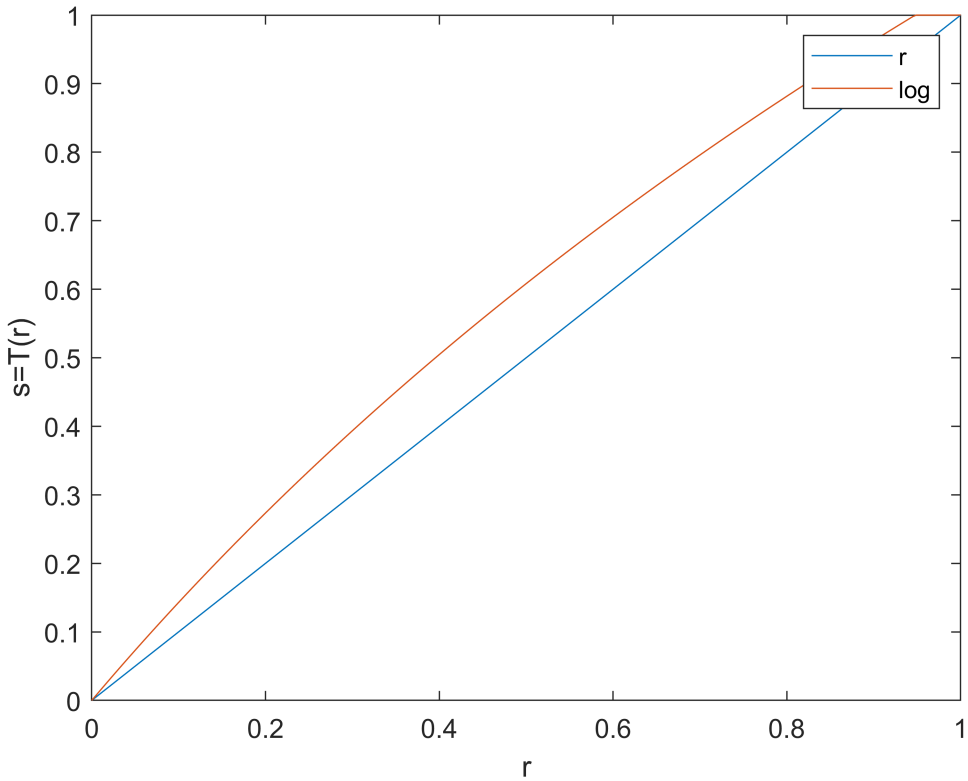
Případně můžeme využít vestavěné funkce `imcomplement()`.

```
neg = imcomplement(f);
```

Logaritmická transformace

$s = c \cdot \log(1 + r)$

```
c = 1.5;
s6 = clipMap(c*log(1+r),0,1);

figure,
plot(r,r);
xlabel('r');
ylabel('s=T(r)');
xlim([0,1]);
hold on;
plot(r,s6);
legend('r','log','Location','northeast');
```



Logaritmická transformace se nejčastěji používá ve spojení s převodem obrázku do frekvenční domény. Příkazům není v tuto chvíli potřeba rozumět, k frekvenční doméně se dostaneme později.

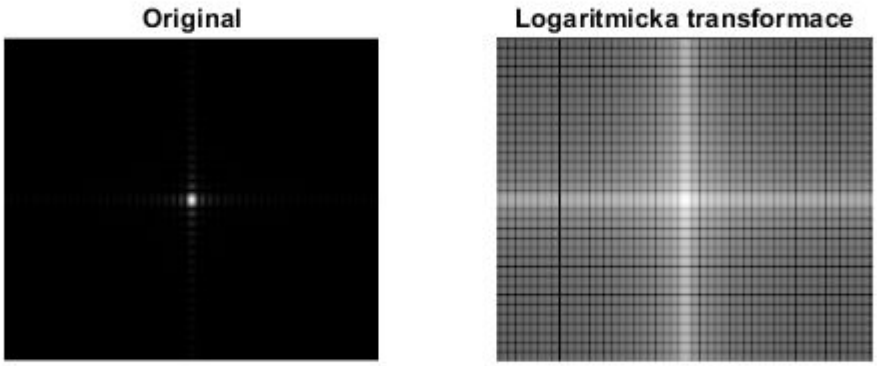
```
obrazek = rgb2gray(imread('fourier.png'));
F = fft2(obrazek);
S = abs(F);
Sc = fftshift(S);
display(min(min(Sc)));
```

```
0
```

```
display(max(max(Sc)));
```

```
442170
```

```
Scl = log(1+Sc);
figure
subplot(1,2,1)
imshow(Sc,[]);
title('Original')
subplot(1,2,2)
imshow(Scl,[]);
title('Logaritmicka transformace')
```



Gamma transformace

$s = r^{\gamma}$

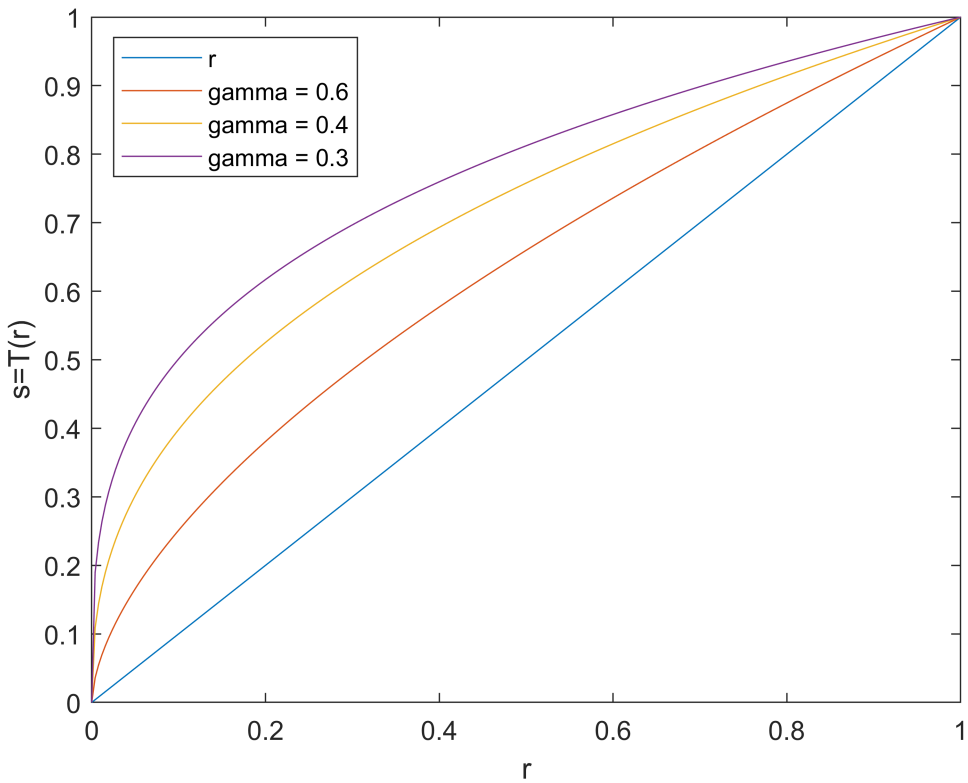
V závislosti na volbě parametru γ potlačujeme tmavou oblast a zvýrazňujeme světlou, nebo naopak.

```
c=1;
Gamma=[0.6 0.4 0.3]
```

```
Gamma = 1×3
    0.6000    0.4000    0.3000
```

```
% Prima aplikace operace na obrazek
% f=rgb2gray(imread('spine.png'));
% x1=double(x);
% y=c*(x1.^Gamma(1));
% y1=c*(x1.^Gamma(2));
% y2=c*(x1.^Gamma(3));

s7 = clipMap(c*(r.^Gamma(1)),0,1);
s8 = clipMap(c*(r.^Gamma(2)),0,1);
s9 = clipMap(c*(r.^Gamma(3)),0,1);

figure,
plot(r,r);
xlabel('r');
ylabel('s=T(r)');
```

```
xlim([0,1]);
hold on;
plot(r,s7);
plot(r,s8);
plot(r,s9);
legend('r','gamma = 0.6','gamma = 0.4','gamma = 0.3','Location','northwest');
```




```
map7 = createMap(s7);
map8 = createMap(s8);
map9 = createMap(s9);

figure
subplot(1,4,1)
imshow(f)
title('Original')
subplot(1,4,2)
imshow(f,map7)
 title('Gamma=0.6')
subplot(1,4,3)
imshow(f,map8)
 title('Gamma=0.4')
 subplot(1,4,4)
imshow(f,map9)
title('Gamma=0.3')
```

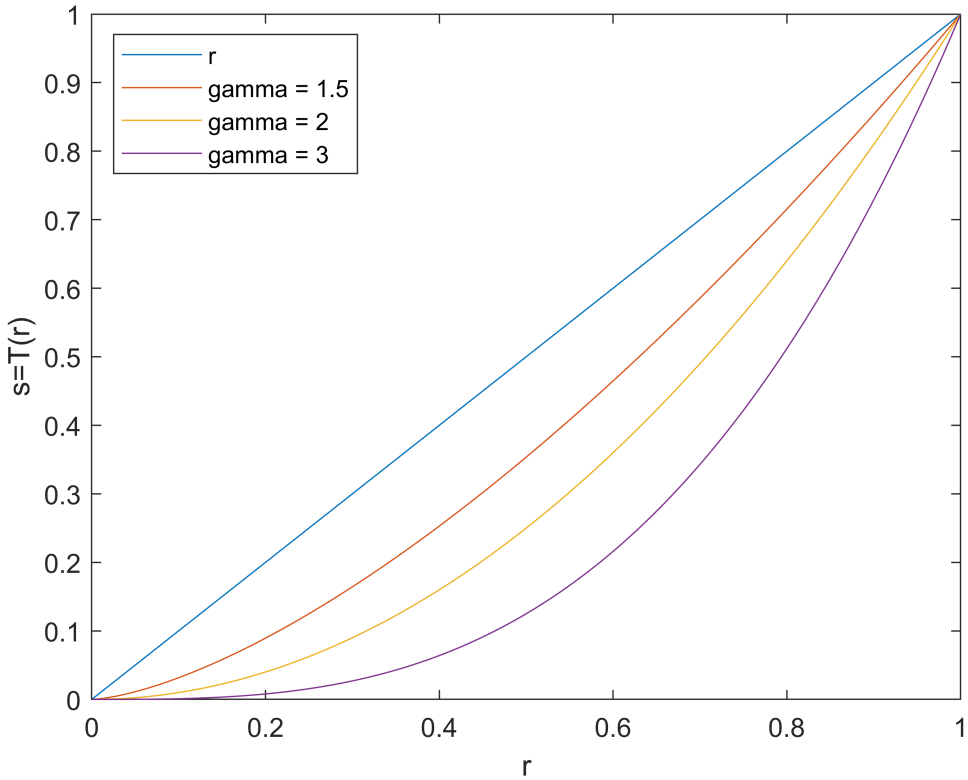
Gamma

```
c=1;
Gamma=[1.5 2 3]
```

```
Gamma = 1×3
    1.5000    2.0000    3.0000
```

```
% f = rgb2gray(imread('vase.jpg'));
% x1=double(x);
% y=c*(x1.^Gamma(1));
% y1=c*(x1.^Gamma(2));
% y2=c*(x1.^Gamma(3));

s10 = clipMap(c*(r.^Gamma(1)),0,1);
s11 = clipMap(c*(r.^Gamma(2)),0,1);
s12 = clipMap(c*(r.^Gamma(3)),0,1);

figure,
plot(r,r);
xlabel('r');
ylabel('s=T(r)');
xlim([0,1]);
hold on;
plot(r,s10);
plot(r,s11);
plot(r,s12);
```

```
legend('r','gamma = 1.5','gamma = 2','gamma = 3','Location','northwest');
```

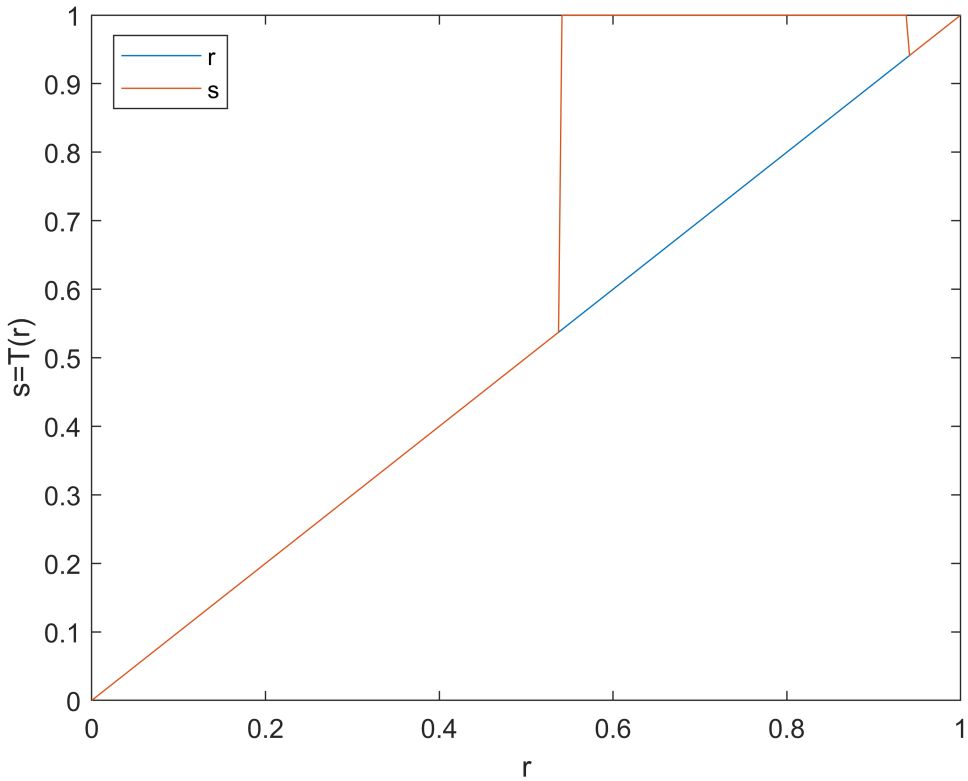



```
map10 = createMap(s10);
map11 = createMap(s11);
map12 = createMap(s12);

figure
subplot(1,4,1)
imshow(f)
title('Original')
subplot(1,4,2)
imshow(f,map10)
 title('Gamma=0.6')
subplot(1,4,3)
imshow(f,map11)
 title('Gamma=0.4')
 subplot(1,4,4)
imshow(f,map12)
title('Gamma=0.3')
```

Intensity-level slicing

Používá se ke zvýraznění některých intenzit. Ostatní intenzity se odstraní (v našem případě nastaví na bílou barvu).

```
% J(J >= 140 & J<=240) = 255;

r1 = 0.54;
r2 = 0.94;

s13 = r;
s13(s13 >= r1 & s13<=r2) = 1;

figure,
plot(r,r);
xlabel('r');
ylabel('s=T(r)');
xlim([0,1]);
hold on;
plot(r,s13);
legend('r','s','Location','northwest');
```



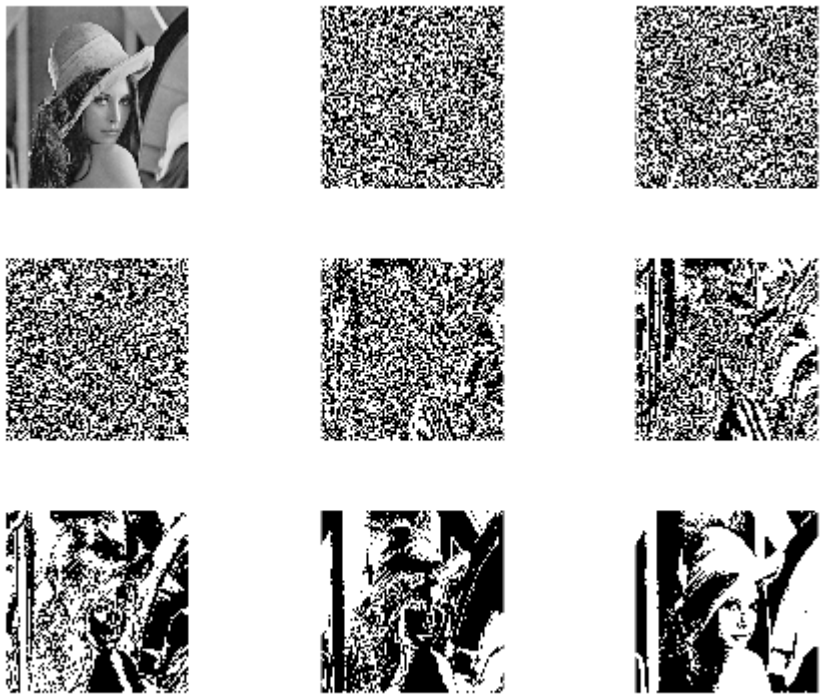

```
map13 = createMap(s13);

figure
subplot(1,2,1)
imshow(f);
title('Original')
subplot(1,2,2)
imshow(f,map13);
title('Intensity-level slicing')
```

Bit-level slicing

Slouží k ořezaní některých bitů. Například z důvodu komprese. Zde si můžeme prohlídnout jednotlivé bitové roviny.

```
I1 = bitand(f,1);
I2 = bitand(f,2);
I3 = bitand(f,4);
I4 = bitand(f,8);
I5 = bitand(f,16);
I6 = bitand(f,32);
I7 = bitand(f,64);
I8 = bitand(f,128);

figure
subplot(3,3,1)
imshow(f,[])
subplot(3,3,2)
imshow(I1,[])
subplot(3,3,3)
imshow(I2,[])
subplot(3,3,4)
imshow(I3,[])
subplot(3,3,5)
imshow(I4,[])
subplot(3,3,6)
imshow(I5,[])
```

```
subplot(3,3,7)
imshow(I6,[])
subplot(3,3,8)
imshow(I7,[])
subplot(3,3,9)
imshow(I8,[])
```



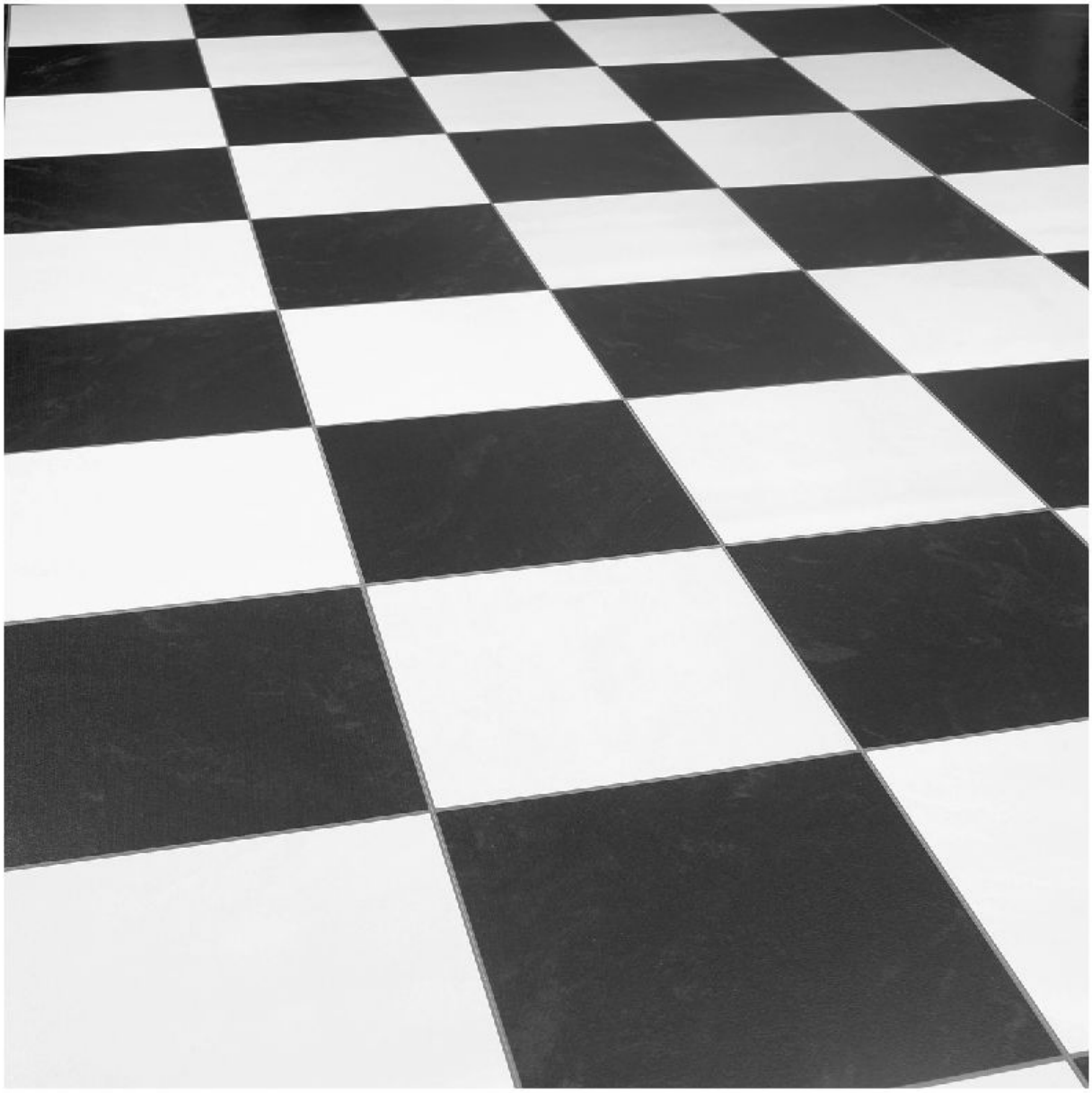
Obrázky se mohou skládat jen z několika nejvýznamnějších bitů.

```
J1 = I8 + I7;
J2 = I8 + I7 +I6;
J3 = I8 + I7 +I6 + I5;

figure,
subplot(1,3,1)
imshow(J1);
title('Dva nejvice vyznamne bity')
subplot(1,3,2)
imshow(J2);
title('Tri nejvice vyznamne bity')
subplot(1,3,3)
imshow(J3);
title('Ctyri nejvice vyznamne bity')
```

Prahování

$$s = \begin{cases} 0, & \text{pokud } r \leq prah \\ 1, & \text{jinak} \end{cases}$$

```
I = rgb2gray(imread('chess.jpg'));
figure, imshow(I);
```

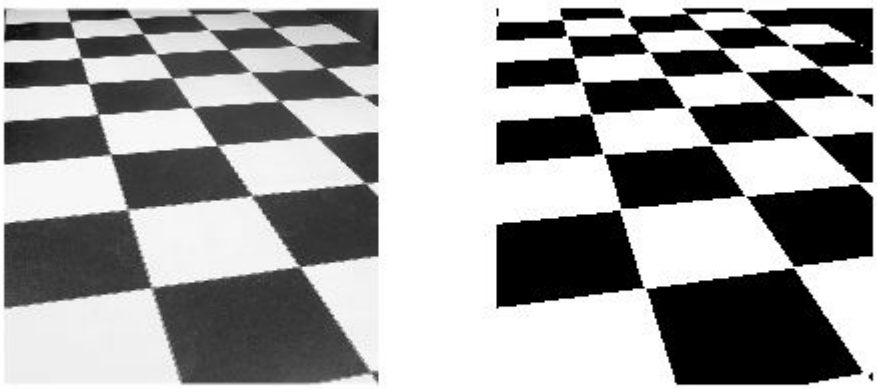


Hledání prahu - experimentálně

Pokud známe prahovou hodnotu, případně odhadujeme tuto hodnotu. K prahování použijeme funkci `imbinarize(obraz,prah)`. Práh se zadává v rozsahu od 0 do 1.

```
prah = 100;

J = imbinarize(I,prah/255);
% J = I > prah;

figure,
subplot(1,2,1)
imshow(I);
```

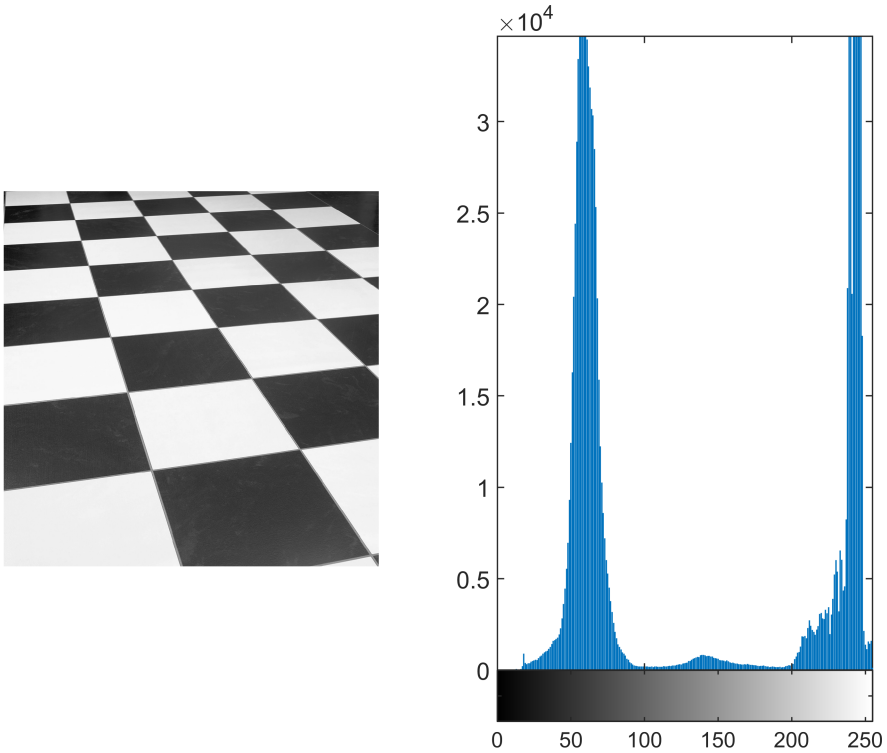
```
subplot(1,2,2)
imshow(J);
```



Hledání prahu - experimentálně z histogramu

Pokud známe prahovou hodnotu, případně odhadujeme tuto hodnotu. K prahování použijeme funkci `imbinarize(obraz,prah)`. Práh se zadává v rozsahu od 0 do 1.

```
figure,
subplot(1,2,1)
imshow(I);
subplot(1,2,2)
imhist(I);
```



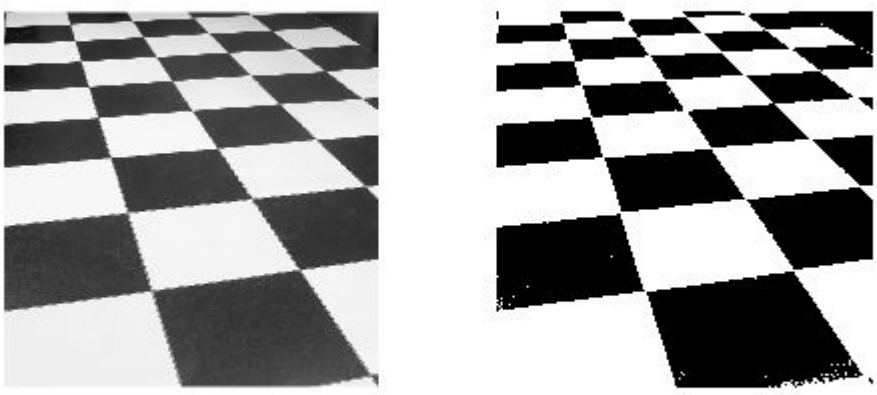
Automatické hledaní prahu

Většinou se práh hledá automaticky. K výpočtu se nejčastěji používají nějaké statisktiky. Například víme, že polovina pixelů je bílá, polovina černá a jiné. K tomu využijeme funkci histogram.

```
[pocet,~] = imhist(I);
cumh = cumsum(pocet);
velikost = size(I,1) * size(I,2);
prah = round(velikost/2);
prah_index = find(cumh>=prah,1,'first');
display(prah_index);
```

```
prah_index = 83
```

```
%J = I>prah_index;
J = imbinarize(I,prah_index/255);

figure,
subplot(1,2,1)
imshow(I);
subplot(1,2,2)
imshow(J);
```



Otsu metoda

Obecně je prahování problém rozdělení pixelů do dvou (a více) tříd tak, aby se minimalizovala globální chyba. Jednou z nejznámějších metod pro automatické hledání prahu je Otsu metoda. V ní se maximalizuje variance mezi těmito třídami. Tato metoda je implementovaná v matlabu. Pokud nezadáme funkci `imbinarize()` prahovou hodnotu, spočítá se pomocí této metody. Prahovou hodnotu můžeme získat pomocí funkce `graythresh()`.

```
J = imbinarize(I);
prah_otsu = graythresh(I)
```

```
prah_otsu = 0.5882
```

```
figure,
subplot(1,2,1)
imshow(I);
subplot(1,2,2)
imshow(J);
```

Otsu metoda - algoritmus

Varianci mezi třídami spočítáme z histogramu obrazu.

Předpokládejme, že obraz obsahuje L intenzit $0, \dots, L-1$

n_i ... počet pixelů s intenzitou i

p_i ... pravděpodobnost výskytu intenzity i v obraze (získáme z normalizovaného histogramu)

Vybereme práh k tak, že nám rozdělí intenzity do dvou tříd.

$c_1 = \{0, \dots, k\}$ a $c_2 = \{k+1, \dots, L-1\}$

Pravděpodobnost výskytu pixelu z první třídy $P_1(k) = \sum_{i=0}^{k} p_i$ a pravděpodobnost výskytu pixelu z druhé třídy $P_2(k) = \sum_{i=k+1}^{L-1} p_i$.

Průměrná intenzita v jednotlivých třídách: $m_1(k) = \frac{1}{P_1(k)} \sum_{i=0}^{k} i \cdot p_i$ a $m_2(k) = \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} i \cdot p_i$.

Globální průměrná hodnota je $m_G = \sum_{i=0}^{L-1} i \cdot p_i$

Pro výpočet efektivity volby prahu použijeme následující vzorec

$$\eta = \frac{\sigma_B^2}{\sigma_G^2}$$

kde σ_G^2 představuje globální varianci a σ_B^2 varianci mezi třídami.

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2$$

$$\sigma_G^2 = \sum_{i=0}^{l-1} (i - m_G)^2 \cdot p_i$$

Optimální k je takové, které maximalizuje σ_B^2.

ÚKOL 1

Implementujte Otsu metodu pro výběr oprimálního prahu. Můžete použít libovolný programovací jazyk.

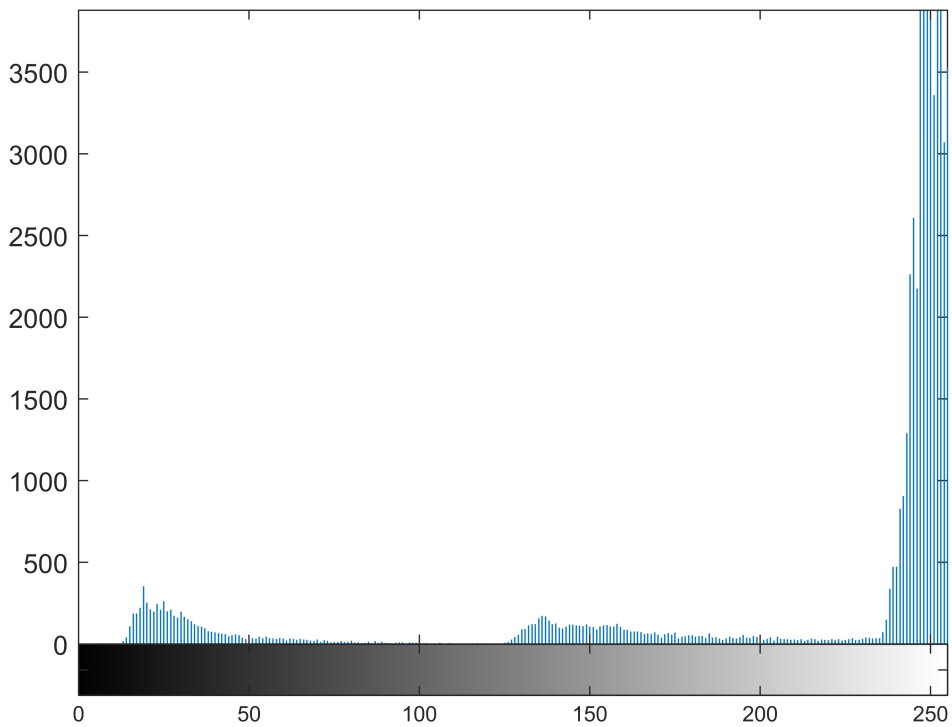
```

```

Vícenásobné prahování

Obraz můžeme chtít prahovat i více prahy. Obecně transformační funkce vypadá následovně,

$$s = \begin{cases} s_1, & \text{pokud } r \leq prah1 \\ s_2, & \text{pokud } prah1 < r \leq prah2 \\ \vdots & \\ s_n & jinak \end{cases}$$

```
f2 = rgb2gray(imread('figurka2.jpg'));
figure, imhist(f2);
```

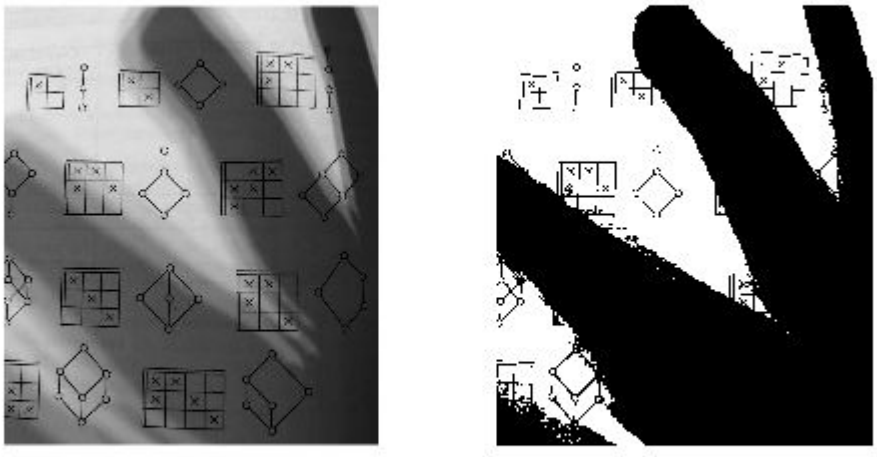



```
T1 = 220;
T2 = 100;
%g2 = 0.5*(f2 >T1) + 0.5*(f2>T2);
g2 = 0.5*imbinarize(f2,T1/255) + 0.5*imbinarize(f2,T2/255);

figure,
subplot(1,2,1)
imshow(f2);
subplot(1,2,2)
imshow(g2);
```

Lokální prahování

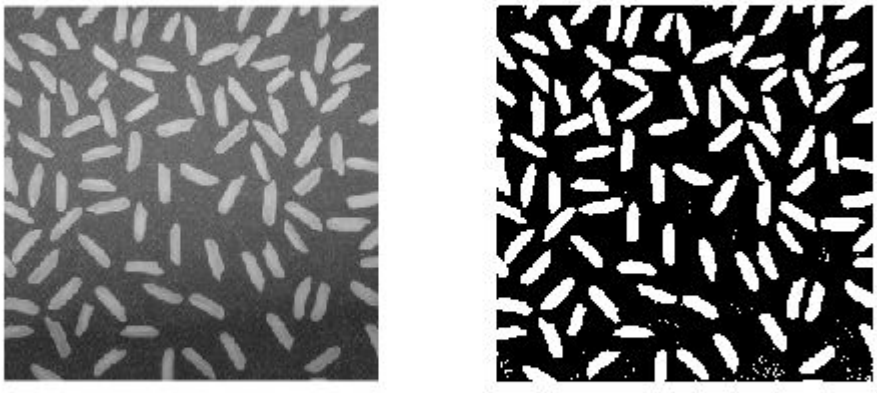
Pro některé obrázky není možné najít optimální prahovou hodnotu.

```
f3 = imread('lokalni.jpg');
g3 = imbinarize(f3);

figure,
subplot(1,2,1), imshow(f3);
subplot(1,2,2), imshow(g3);
```



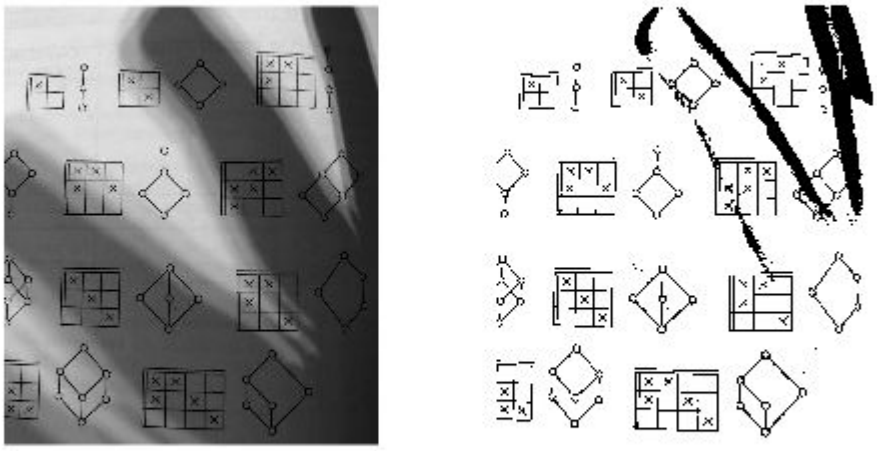
V takovém případě se používá takzvané lokání, nebo také adaptivní prahování. Ideální prahová hodnota se hledá pro každý pixel zvlášť. Pro každý pixel vezmeme nějaké jeho okolí a pro něj spočítáme ideální prahovou hodnotu, například pomocí metody Otsu. V matlabu můžeme pro adaptivní prahování použít `imbinarize()` následujícím způsobem.

```
f3 = imread('rice.png');
g3 = imbinarize(f3,'adaptive');

figure,
subplot(1,2,1), imshow(f3);
subplot(1,2,2), imshow(g3);
```



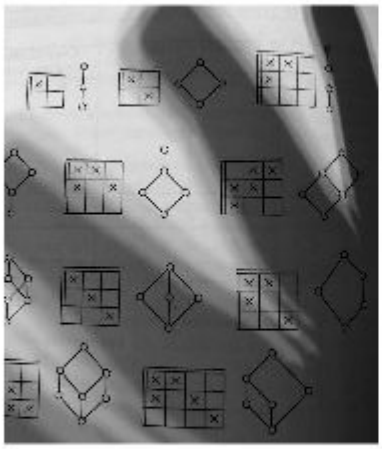
Pokud je pozadí obrázku tmavší, je potřeba nastavit další parametry. Například následovně.

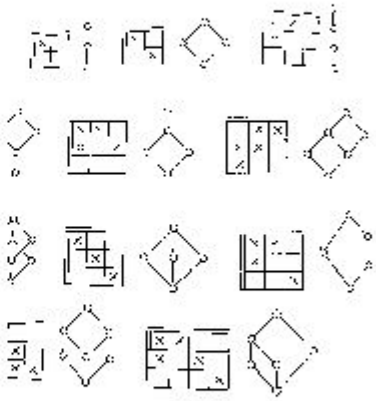
```
f3 = imread('lokalni.jpg');
g3 = imbinarize(f3,'adaptive','ForegroundPolarity','dark','Sensitivity',0.4);

figure,
subplot(1,2,1), imshow(f3);
subplot(1,2,2), imshow(g3);
```



Jednou z nejpoužívanějších metod pro lokální prahování je metoda založená na standardní odchylce počítané pro každý bod v obraze. Tato hodnota se počítá oproti bodům v okolí (nejčastěji velikosti 3x3). Tento způsob má však více společného s filtrováním (operacemi s okolím bodu), kterým se budeme věnovat později. Zde bez dalšíhovysvětlení jen přikládám kód.

```
f3 = imread('lokalni.jpg');
f4=im2double(f3);
nhood = ones(3)/9;
g_std = stdfilt(f4);
g_mean = imfilter(f4,nhood);

a=0.3;
b=1-a;
T = a*g_std + b*g_mean;

g4 = imbinarize(f4,T);

figure,
subplot(1,2,1), imshow(f4);
subplot(1,2,2), imshow(g4);
```





Úkol 2

Popište, jakým způsobem se změní histogram pokud na obrázek aplikujeme následující operace. Svou doměnku ověřte pomocí matlabu.

- *změna jasu (snížení/zvýšení)*
- *změna kontrastu (snížení/zvýšení)*
- *negativ obrázku*
- *gamma korekce (pro různé gamma)*