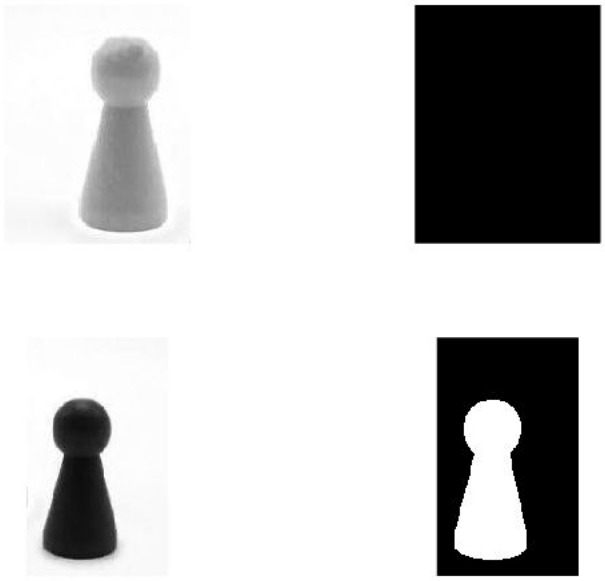
Cvičení 12 - Segmentace

Segmentace je proces, který dělí obraz na jednotlivé objekty, nebo regiony. To, na co ho dělí je záležitostí aplikace. Například segmentace může zastavit, když oddělí objekt, který nás zajímá od zbytku obrazu. Segmentace je jedním z nejsložitějších procesů v digitálním zpracování obrazu. Algoritmy pro monochromatické obrázky jsou založeny na intenzitách pixelů na jejich podobnosti/intenzitě. U barevných obrázků bereme v úvahu barvu. Obrázek můžeme dělit na segmenty na základě prudkých změn (hran v obraze), nebo hledáme oblasti na základě podobnosti hodnot (podle předem definovaného kritéria).

Prahování

viz. cvičení 4.

```
I = imread('figurka1a.jpg');
I2 = imread('figurka1b.jpg');

ROI = I < 110; % 240
ROI2 = I2 < 110;

figure,
subplot(2,2,1), imshow(I);
subplot(2,2,2), imshow(ROI);
subplot(2,2,3), imshow(I2);
subplot(2,2,4), imshow(ROI2);
```



`imbinarize()`

```
ROIa = 1 - imbinarize(I);
ROI2a = 1 - imbinarize(I2);
```

```
figure,
subplot(2,2,1), imshow(I);
subplot(2,2,2), imshow(ROIa);
subplot(2,2,3), imshow(I2);
subplot(2,2,4), imshow(ROI2a);
```




Detekce bodů, linií a hran

Nejčastěji používaná metoda pro hledání nespojitostí (bodů, hran, linií) v obraze je konvoluce maskou, většinou 3x3 pixelů a výpočet tzv. response pro pixel:

`R = w1z1 + ... + w9z9`

`z` je intenzita pixelu

`w` váha

Detekce bodů

K detekci izolovaných bodů se používá maska:

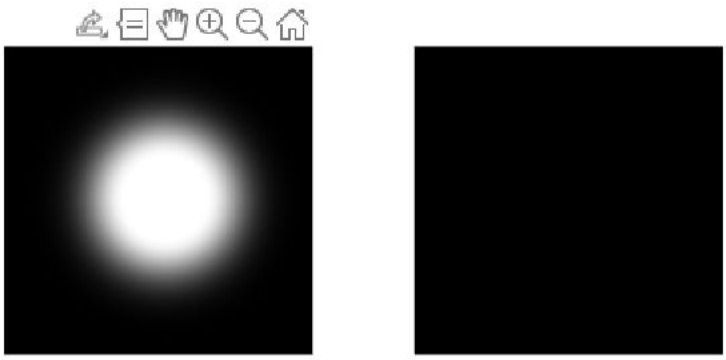
$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Řekneme, že jsme detekovali izolovaný od, pokud odpověď $R \geq T$ je větší než zadaná prahová hodnota `T`.

`imfilter()`

```
f = rgb2gray(imread('segmentace1.png'));
w = [-1 -1 -1;-1 8 -1; -1 -1 -1];
```

```
R = abs(imfilter(f,w));
% jako prahovou hodnotu vezmeme největší hodnotu filtrovaného obrazu
T = max(R(:));

g = R >=T;
figure,
subplot(1,2,1), imshow(f);
subplot(1,2,2), imshow(g);
```



V původním obraze není bod téměř zřetelný.

Detekce linií

Viz. cvičení 6 (ostřící filtry a derivace).

Maska pro detekci horizontálních linií:

$$\begin{pmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{pmatrix}$$

Maska pro detekci linií pod úhlem 45°:

$$\begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}$$

Maska pro detekci vertikálních linií:

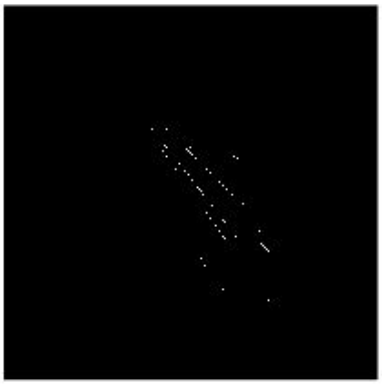
$$\begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix}$$

Maska pro detekci linií pod úhlem -45°:

$$\begin{pmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{pmatrix}$$

Pokud hledáme linie se směrem definovaným maskou, pak uděláme konvoluci nad obrázkem touto maskou a opět porovnáváme odpověď `R` s nějakou prahovou hodnotou.

```
f = rgb2gray(imread('segmentace2.png'));
w = [2 -1 -1; -1 2 -1; -1 -1 2];
R = abs(imfilter(f,w));
T = 120;
g = R >=T;
figure,
subplot(1,2,1), imshow(f);
subplot(1,2,2), imshow(g);
```



Derivace

Konvoluční masky zložené na derivaci, byly více diskutovány ve cvičení 6. Byl zde představen filtr Prewitové, Soblův operátor, Laplaceův operátor. Tyto operátory se dají použít k hledání hran/linií pomocí funkce

`edge()`

funkce bere jako parametr metodu (celkem 6 metod - mimo jiné právě Sobelův operátor, Prewitové, Laplaceův...) a případně parametry.

```
help edge
```

```
f = rgb2gray(imread('segmentace2.png'));
g = edge(f,'prewitt');
figure,
subplot(1,2,1), imshow(f);
subplot(1,2,2), imshow(g);
```

ÚKOL 1

vyzkoušejte i jiné metody a různé parametry.

Detekce hran pomocí Houghovy transformace

V ideálním případě předchozí metody detekují pixely ležící na hranách (liniích). Ve skutečnosti však vybrané pixely nemusí tvořit celou hranu. Nespojitosti hran mohou být zapřičiněny buď nerovnoměrným osvětlením,

šumem a pod. V liniích pak vznikají nespojitosti. Houghova transformace pracuje tak, že v obraze hledá linie, na kterých leží nějaké množství pixelů. Zjednodušeně pracuje tak, že zkouší všechny liniie a počítá, kolik na nich leží pixelů. Běžně se používá nad černobílými obrazy. Je založena na myšlence, že každá přímka je jednoznačně určena 2 parametry (v tomto případě se dvojice parametrů v polárních souřadnicích `r` a θ). Ty představují vzdálenost bodu od středu souřadnicového systému a úhlu svírajícím s osou `x`.

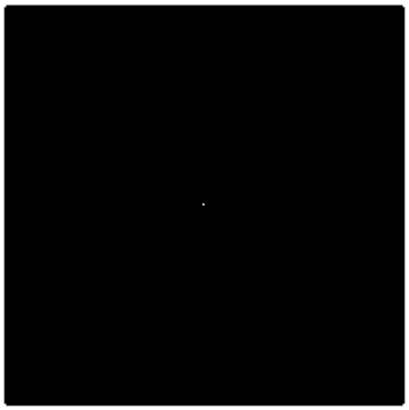
Rovnice přímky je rovna:

$$x\cos(\theta) + y\sin(\theta) = r$$

Pro každý bod `(x,y)` v původním obrázku se dá jednoduše najít množina všech přímek procházejících tímto bodem (všechny dvojice `r`, θ splňující výše uvedenou rovnici po dosazení konkrétních hodnot `x` a `y`).

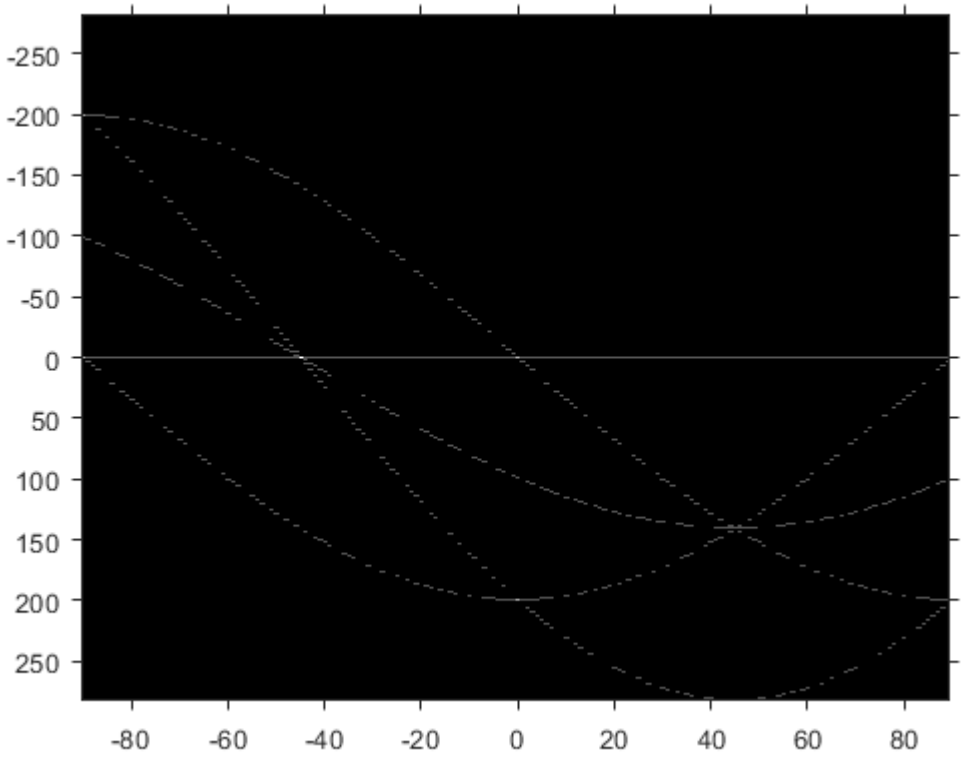
Množina přímek bude mít tvar sinusovky (v prostoru `r`, θ). Pro každý bod (který je potenciálně součástí nějaké přímky. Např je bílý) v obraze zobrazíme množinu všech přímek, které jím procházejí jako sinusovku do (`r`, θ). Průniky takto vytvořených sinusovek pak tvoří přímky v obraze. Čím více sinusovek se protíná, tím více bodů leží na přímce.

```
% Vytvoříme si obrázek obsahující jen několik bílých bodů:
f = zeros(200,200);
f(1,1) = 1;
f(1,end) = 1;
f(end,1) =1;
f(end,end)= 1;
f(100,100)=1;
figure,
imshow(f);
```



```
[H, theta, r] = hough(f);
figure,
imshow(H,[], 'XData', theta, 'YData', r ,'InitialMagnification', 'fit');
```

```
axis on, axis normal   % figure nezavírejte
```

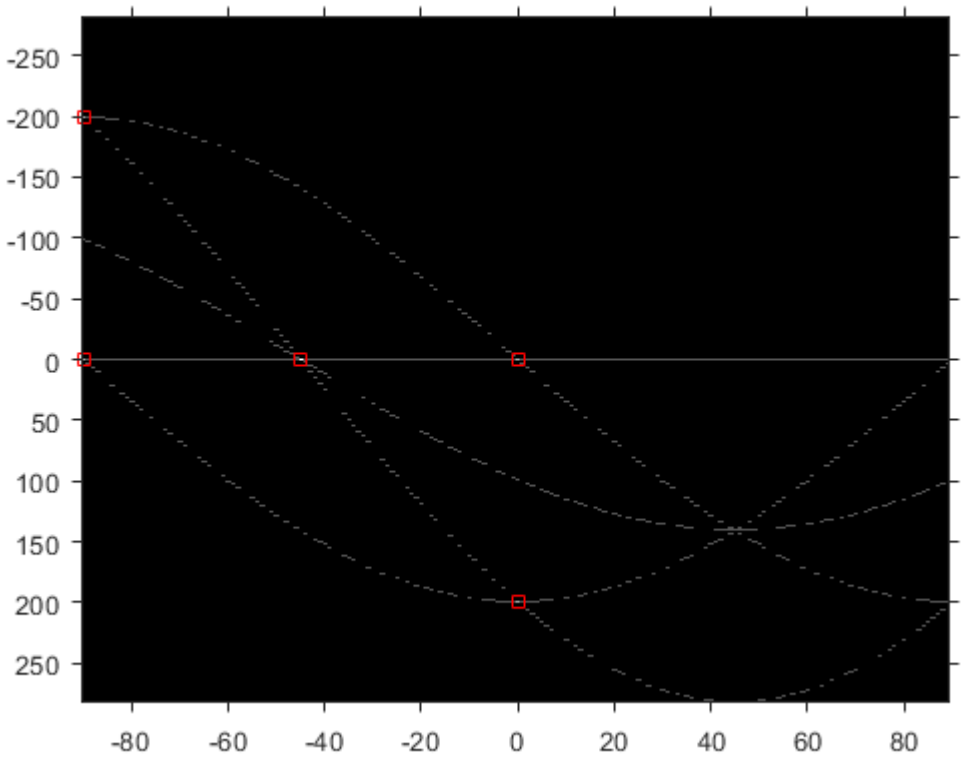


Obraz `f` obsahuje 5 bílých bodů, `H` obsahuje 5 sinusovek (i vodorovná linie je jedna z nich). Jejich průsečíky představují počet bodů na jedné přímce. `X` osa představuje parametr θ, `Y` `r`.

Houghova transformace

Hledání průsečíků se běžně implementuje za pomocí akumulátoru. Zvolí se krok pro vzdálenosti od středu os (běžný krok je 1 pixel) a počet úhlů na. Vytvoří se akumulátor jako dvourozměrné pole / matice, kde jedna souřadnice reprezentuje vzdálenosti a jedna úhly. Pro každý bod a každý úhel se dopočítává vzdálenost a podle vypočítané dvojice se zvýší hodnota akumulátoru. Dvojice s nejvyšší hodnotou akumulátoru pak tvoří přímku (případně se bere prahová hodnota, kolik bodů musí ležet na přímce, aby byla považována za přímku v obraze). funkce houghpeaks detekuje ze sinusovek uložených v matici H numpeaks (druhý parametr) průsečíků. Dá se nastavit kolik bodů musí na přímce ležet.

```
peaks = houghpeaks(H,5);
hold on;
plot(theta(peaks(:,2)), r(peaks(:,1)), 'linestyle', 'none', 'marker', 's',
'color','r');
```



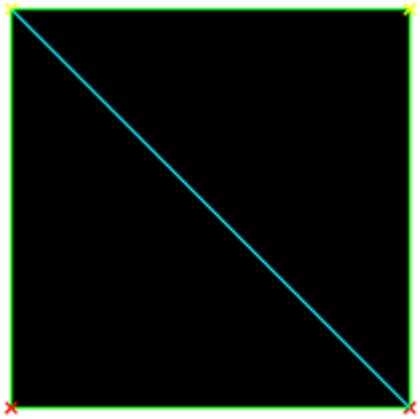

```
% peaks = houghpeaks(H, 1, 'Threshold', 3);
```

Z nalezených peaků je potřeba určit, kde jednotlivé linie začínají a končí. K tomu slouží funkce houghlines

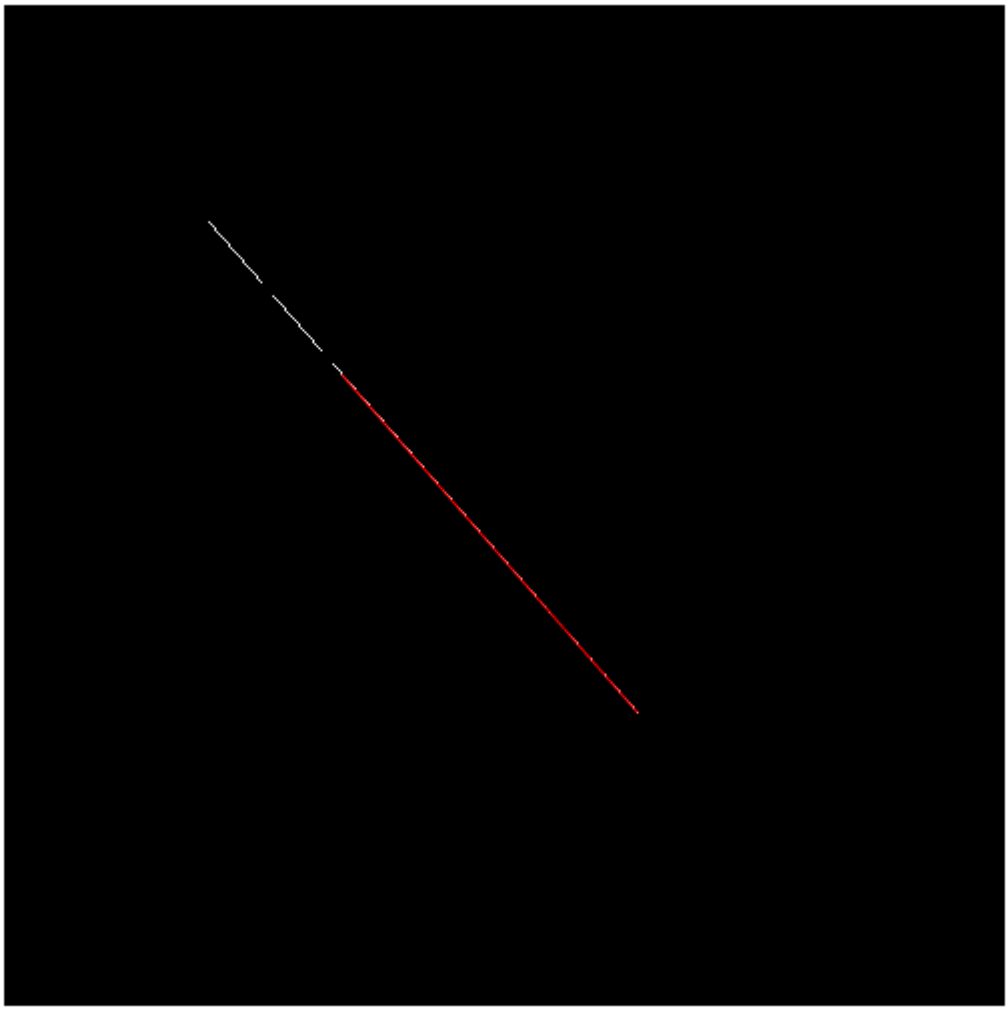
`lines = houghlines(f,theta,r,peaks)`

```
lines = houghlines(f,theta,r,peaks,'FillGap',200);
       figure, imshow(f), hold on
       max_len = 0;
       for k = 1:length(lines)
         xy = [lines(k).point1; lines(k).point2];
         plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','green');

         % Vykreslení začátků a konců
         plot(xy(1,1),xy(1,2),'x','LineWidth',1,'Color','yellow');
         plot(xy(2,1),xy(2,2),'x','LineWidth',1,'Color','red');

         % určení konce nejdelší úsečky
         len = norm(lines(k).point1 - lines(k).point2);
         if ( len > max_len)
           max_len = len;
           xy_long = xy;
         end
       end

       % Zvýraznění nejdelšího úseku
```

```
        plot(xy_long(:,1),xy_long(:,2),'LineWidth',1,'Color','cyan');
```



```
f = imread('segmentace3.png');
[H, theta, r] = hough(f, 'ThetaResolution', 0.2);
peaks = houghpeaks(H,5);
lines = houghlines(f,theta,r,peaks);

figure, imshow(f,[]), hold on
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1),xy(:,2), 'Color', 'r');
end
```



Region-based segmentace

Předpokládáme, že celý obraz můžeme rozdělit do disjunktních regionů. Každý region je spojitý (4- nebo 8- spojitost). V každém regionu jsou pixely, které splňují nějakou podmínku (např. všechny mají stejnou intenzitu...). Sousední regiony jsou disjunktní v řeči podmínky, kterou pixely regionu splňují.

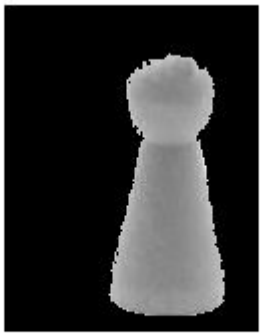
Region growing

Regiony vytváříme tak, že k seedům přdáváme další pixely (sousední) na základě toho, zda splňují podmínku, nebo ne.

Pro obraz určíme počáteční pixel regionu (seed). Definujeme podmínku, na základě které budeme pixely přidávat do regionu (například prahovou hodnotu, které určuje maximální možný rozdíl mezi hodnotami v regionu. Algoritmus funguje tak, že začínáme s prázným regionem (nulová matice) a od seedu přidáváme do fronty sousední pixely. Postupně odebíráme pixely z fronty a počítáme zda na základě podmínky patří, nebo

nepatří do oblasti. Pokud ano, nastavíme v matici regionu tomuto pixelu 1 a do fronty přidáme sousední pixely. Takto pokračujeme dokud není fronta prázdná.

```
I = imread('figurka1a.jpg');
seed = [80,100];
T = 43;

[region, segmented] = regiongrowing(I, seed, T);

figure,
imshow(segmented);
```



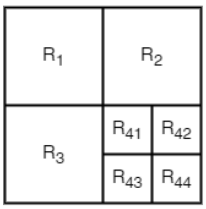
ÚKOL 2

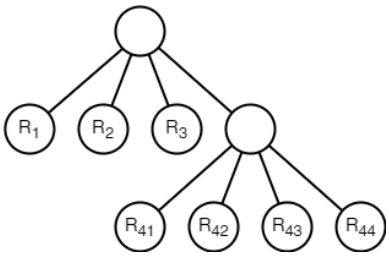
Naprogramujte region growing dle předchozího popisu.

Region splitting and merging

Začínáme s celým obrazem a postupně ho dělíme na kvadranty (každý kvadrat pak na další a pod). Dělíme je do té doby, než celý region splňuje zvolenou podmínku.

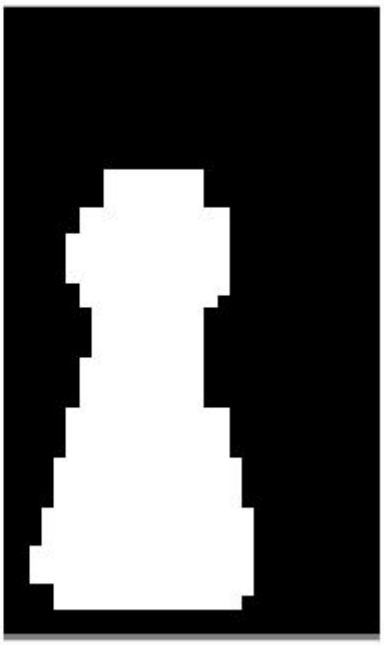
Začneme s celým obrazem. Rozdělíme ho na kvadranty. Pro každý kvadrant otestujeme, zda je splněna podmínka. Pokud ne, dělíme tento kvadrant na kvadranty a opět testujeme. Tímto dělením nám vzniká takzvaný quadtree. (funkce `qtdecomp()`)





Pokud bychom pouze dělili regiony na čtrtiny, pak by se v obraze mohly tvořit sousední regiony, které mají stejné vlastnosti. Proto se spojují sousední regiony, pro které platí, že po sloučení je splněna podmínka.

```
I = imread('figurka1b.jpg');
J = splitmerge(I,5, @funkce);

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



Segmentace v Matlabu

Součásti Image processing toolboxu je `imageSegmenter`, který implementuje celou řadu algoritmů určených k segmentaci.

```
imageSegmenter
```

ÚKOL 3

Podívejte se na `imageSegmenter`. Pomocí něho z obrázku figurka2.jpg vytvořte 3 regiony (světlejší figurka, tmavší figurka a pozadí).