

Cvičení 9 - Filtrování ve frekvenční doméně

Jak víte z konvolučního teorému, tak konvoluce v prostorové doméně odpovídá násobení matic (prvek po prvku) ve frekvenční doméně.

$$f(x,y)*h(x,y) = H(u,v)F(u,v)$$

(* označuje konvoluci)

Filtrování ve frekvenční doméně tedy probíhá tak, že převedeme obrázek do frekvenční domény, ten vynásobíme maticí filtru prvek po prvku (musí být stejně velká jako je obrázek) a převedeme zpět do prostorové domény. Tento proces je rychlejší, než filtrování pomocí konvoluce a navíc některé druhy šumu nelze jednoduše v prostorové doméně odfiltrovat, ale ve frekvenční ano (například periodický šum).

Hlavním úkolem je najít funkci $H(u, v)$ (filter transfer function), která bude modifikovat $F(u, v)$ požadovaným způsobem.

Příkladem by mohl být filtr, který bude násobit vysoké frekvence a potlačovat nízké. Za předpokladu, že máme F centrovaný (nejvyšší frekvence uprostřed), pak by filtr měl kolem středu vysoké hodnoty a dál od středu má hodnoty nulové (low pass filtr).

Víme, že F je periodická, stejně tak H . Z toho plyne, že i konvoluce v diskrétní frekvenční doméně. Z tohoto důvodu je konvoluce provedena za použití DFT circular convolution. Jediný způsob, jak dosáhnout toho, že prostorová i circular konvoluce dávají stejný výsledek je, že se použije správný okraj obrázku (zero padding - jak jsme si říkali u prostorové konvoluce).

Není těžké si představit, že konvoluce periodické funkce může způsobit rušení mezi sousedními periodami pokud jsou tyto periody blízko sebe. Této inferenci se říká wraparound error a můžeme se jí vyhnout, pokud použijeme zero padding následujícím způsobem:

f velikosti $A \times B$

h velikosti $C \times D$

Vytvoříme rozšířené funkce tak, že přidáme 0 k f i h tak, aby obě měly velikost $P \times Q$

$$P \geq A + C - 1$$

$$Q \geq B + D - 1$$

`paddedsized()`

Tato funkce zvolí nejmenší P a Q , která jsou sudá (kvůli efektivnímu výpočtu FFT) a splňují podmínky výše.

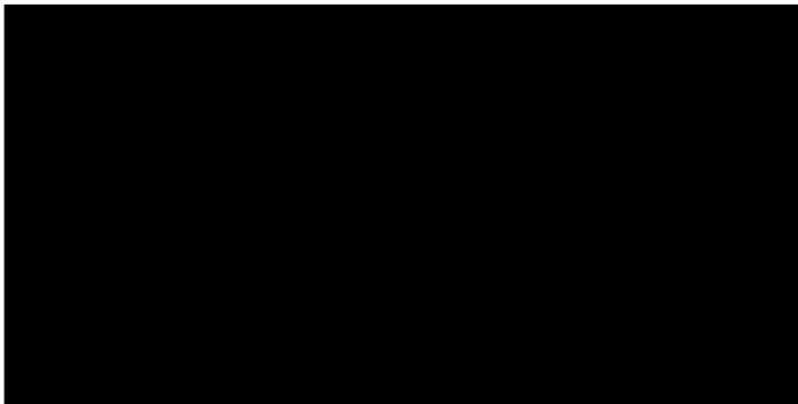
```
A = 100;
B = 200;
C = 21;
D = 20;

PQ = paddedsized([A B],[C D]);
display(PQ);
```

```
PQ = 1x2
    120    220
```

Abychom se vyhli wraparound error musíme v prostorové doméně "obalit" nulami obě funkce f i h

```
% Vytvoříme si obrázek
f = [ones(200,400);zeros(200,400)];
figure, imshow(f);
```



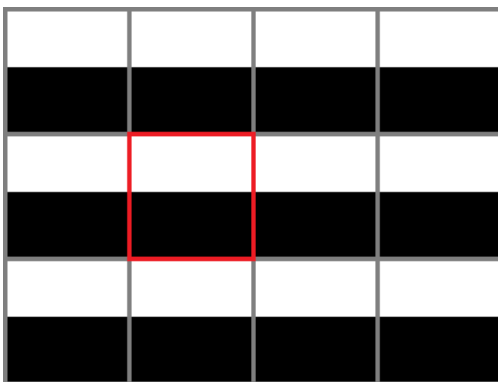
Zjistíme jeho velikost. Pomocí funkce lpfilter vytvoříme Gaussovský low pass filtr

```
[M, N] = size(f);
F = fft2(f);
sig=10; %sigma
H = lpfilter('gaussian' , M, N, sig) ;
G = H.*F;
g= ifft2(G);
figure, imshow(g);
```



V tomto případě jsme obrázku nepřidali dostatečný zero padding - výsledný obrázek je rozmazaný v horizontálním směru, ale vertikální hrany ne.

Je to dáno tím, že obrázek je periodický a tedy tento případ vypadá nějak takto (červený výřez je obrázek):



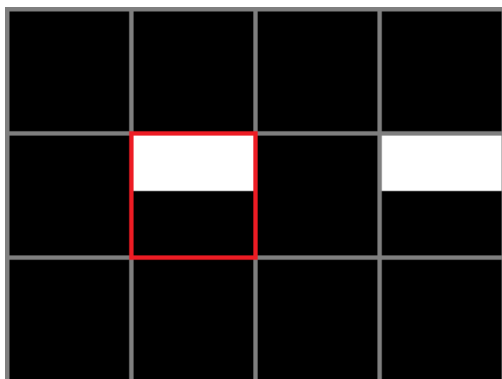
('cviceni10o1.png')

Při přidání zero-padding

```
PQ = paddedsize(size(f));  
Fp = fft2(f, PQ(1), PQ(2)); %přidá padding  
Hp = lpfilter('gaussian', PQ(1), PQ(2), 2*sig);  
Gp = Hp .* Fp ;  
gp = ifft2(Gp);  
gpc = gp(1:size(f,1), 1:size(f,2));  
figure, imshow(gpc);
```



Obrázek je díky zero paddingu 2x tak velký. Zde je vidět i rozmazání ve vertikálním směru. Není to překvapivé, protože tento případ vypadá takto:



('cviceni10o2.png')

Postup filtrování:

1. získání optimální velikosti paddingu

```
PQ = paddedsize(size(f))
```

2. fourierova transformace obrázku s paddingem

```
F = fft2(f, PQ(1), PQ(2))
```

3. vytvoření filtru ve frekvenční doméně o velikosti $P \times Q$. Pokud je filtr centrováný, je nutné cetrovat i F (`fftshift()`), nebo decentrovat H (`ifftshift()`)

4. Vynásobení H a F

```
G = H.*F
```

5. Inverzní furierova transformace

```
g = ifft2(G)
```

6. Ořezání obrázku na původní velikost

```
g = g(1:size(f,1), 1:size(f,2));
```

Vytvořila jsem funkci `dftfilt()`, která jako vstup bere obrázek v prostorové doméně a filter transfer funkci a vrací vyfiltrovaný obrázek v prostorové doméně.

```
f = im2double(imread('a.png'));
```

Jak získat filtry?

Z filtrů v prostorové doméně

V matlabu k tomu je naprogramovaná funkce `freqz2()`, která převádí prostorový filtr do frekvenčního. Pro více informací:

```
help freqz2
```

freqz2 2-D frequency response.

`[H,Fx,Fy] = freqz2(h,Nx,Ny)` returns `H`, the `Ny`-by-`Nx` frequency response of `h`, and the frequency vectors `Fx` (of length `Nx`) and `Fy` (of length `Ny`). `h` is a two-dimensional FIR filter, in the form of a computational molecule. `Fx` and `Fy` are returned as normalized frequencies in the range `-1.0` to `1.0`, where `1.0` corresponds to half the sampling frequency, or π radians.

`[H,Fx,Fy] = freqz2(h,[Ny Nx])` returns the same result as `[H,Fx,Fy] = freqz2(h,Nx,Ny)`.

`[H,Fx,Fy] = freqz2(h,N)` uses `[Ny Nx] = [N N]`.

`[H,Fx,Fy] = freqz2(h)` uses `[Ny Nx] = [64 64]`.

`[H,Fx,Fy] = freqz2(h,[Ny Nx],[Dx Dy])` or `freqz2(h,Nx,Ny,[Dx Dy])` uses `[Dx Dy]` to specify the intersample spacing in `h`. `Dx` is the spacing in the X (or column) dimension, and `Dy` is the spacing in the Y (or row) dimension. The default value for `Dx` and `Dy` is `0.5`. Changing `Dx` and `Dy` has the effect of changing the spread of values in `Fx` and `Fy`. For example, setting `Dx` to `0.25` instead of `0.5` causes the values in `Fx` to range from `-2.0` to `2.0` instead of `-1.0` to `1.0`. Setting `Dy` to `1.0` instead of `0.5` causes `Fy` to range from `-0.5` to `0.5` instead of `-1.0` to `1.0`.

`freqz2(h,[Ny Nx],D)` and `freqz2(h,Nx,Ny,D)` use `[Dx Dy] = [D D]`.

`H = freqz2(h,Fx,Fy)` returns the frequency response for the FIR filter `h` at frequency values in `Fx` and `Fy`. These frequency values should be in the range `-1.0` to `1.0`, where `1.0` corresponds to half the sampling frequency, or π radians.

When used with no output arguments, `freqz2(...)` produces a mesh plot of the two-dimensional frequency response.

Class Support

The input matrix `h` can be of class `double` or of any numeric class. All other inputs to `freqz2` must be of class `double`. All outputs are of class `double`.

Example

Use the window method to create a 16-by-16 filter, then view its frequency response using `freqz2`.

```
Hd = zeros(16,16);
Hd(5:12,5:12) = 1;
Hd(7:10,7:10) = 0;
h = fwind1(Hd,bartlett(16));
freqz2(h,[32 32]); axis([-1 1 -1 1 0 1]); colormap(jet(64))
```

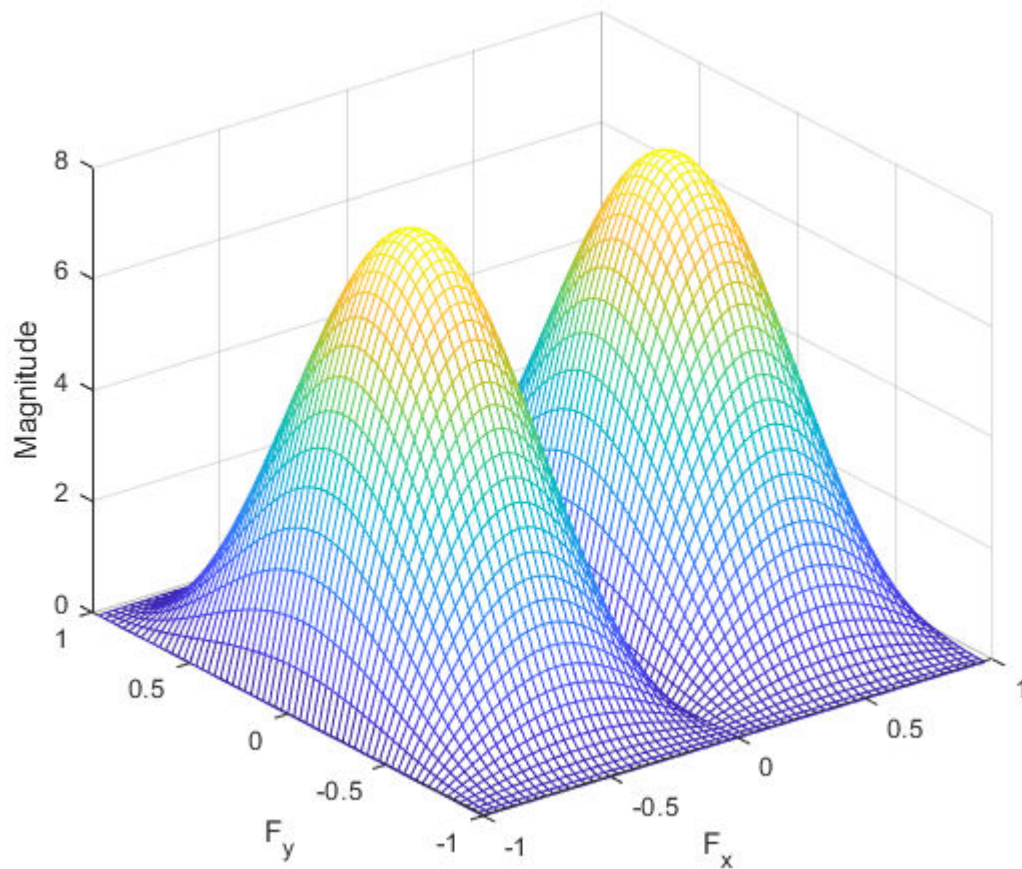
See also `freqz`.

Documentation for `freqz2`

```
h = fspecial('sobel'); % filtr v prostorové doméne
display(h);
```

```
h = 3x3
    1    0   -1
    2    0   -2
```

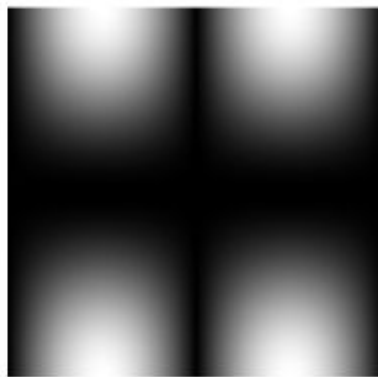
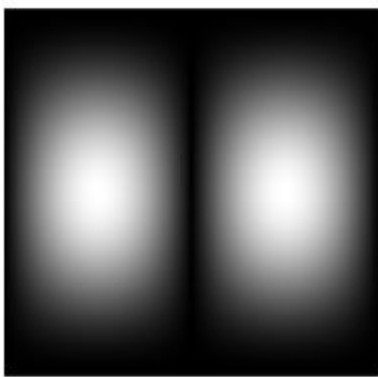
```
freqz2(h); % Převedení do frekvenční domény.
```



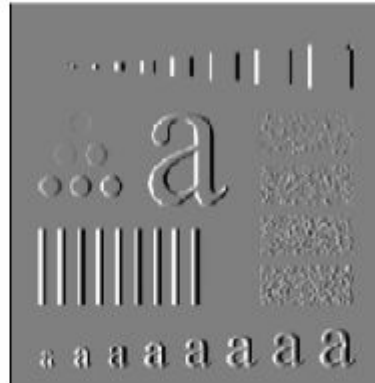
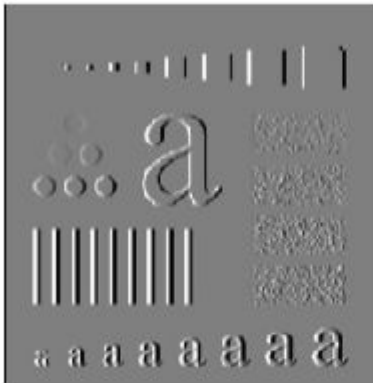
```
PQ = paddedsize(size(f));
H = freqz2(h, PQ(1), PQ(2)); % zde i se specifikací velikosti
H1 = fftshift(H);
```

```
% Zobrazení filtru a centrovaného filtru
```

```
figure,
subplot(1,2,1), imshow(abs(H), [ ]);
subplot(1,2,2), imshow(abs(H1), [ ]);
```

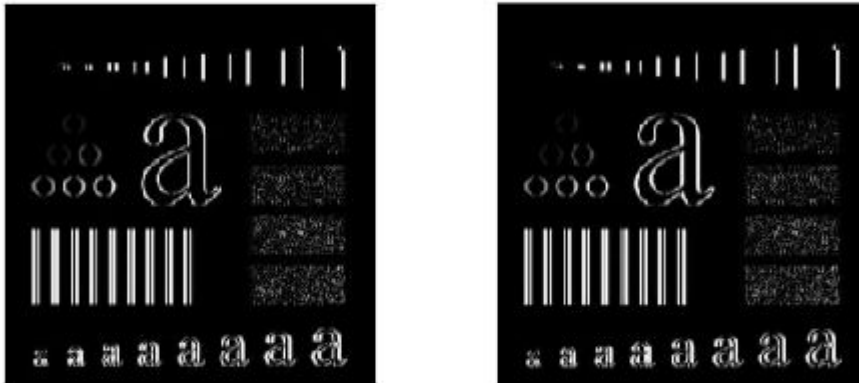


```
gs = imfilter(f, h); % filtrování v prostorové doméně  
gf = dftfilt(f, H1); % filtrování ve frekvenční doméně  
  
% porovnání výsledků.  
figure,  
subplot(1,2,1), imshow(gs, [ ]);  
subplot(1,2,2), imshow(gf, [ ]);
```

gs a gf obsahují i záporné hodnoty. Proto se podíváme na absolutní hodnotu.

```
figure,  
subplot(1,2,1), imshow(abs(gs), []);  
subplot(1,2,2), imshow(abs(gf), []);
```



když porovnáme výsledné obrázky zjistíme, že rozdíl je zanedbatelný

```
d = abs(gs - gf);
max(d(:)) % maximální rozdíl
```

```
ans = 3.1086e-15
```

```
min (d(:))
```

```
ans = 0
```

Generování filtrů ve frekvenční doméně

Většina filtrů je založena na tom, že jsou symetrické okolo středu. Bude se nám hodit funkce pro výpočet vzdálenosti bodu od specifikovaného bodu

funkce `dftuv(U,V)`

vytvoří meshgrid vzdáleností od počátku velikosti $U \times V$ (předpokládá se periodicitu výsledku) podívejte se na kód jakým způsobem je vzdálenost počítána.

```
[U, V] = dftuv(8, 5);
DSQ = U.^2 + V.^2;
DSQ
```

```
DSQ = 8x5
      0      1      4      4      1
```

```

1   2   5   5   2
4   5   8   8   5
9  10  13  13  10
16 17  20  20  17
9  10  13  13  10
4   5   8   8   5
1   2   5   5   2

```

vycentrujeme pomocí fftshift

```
fftshift(DSQ)
```

```

ans = 8x5
    20    17    16    17    20
    13    10     9    10    13
     8     5     4     5     8
     5     2     1     2     5
     4     1     0     1     4
     5     2     1     2     5
     8     5     4     5     8
    13    10     9    10    13

```

Low pass filtry

Slouží k potlacení nízkých frekvencí - rozmazání hran.

ÚKOL 1

Vytvoření ideálního LP filtru doprogramujte do funkce lpfilter.

$$H(u, v) = \begin{cases} 1 & \text{if } D(u, v) \leq D_0 \\ 0 & \text{if } D(u, v) > D_0 \end{cases}$$

```

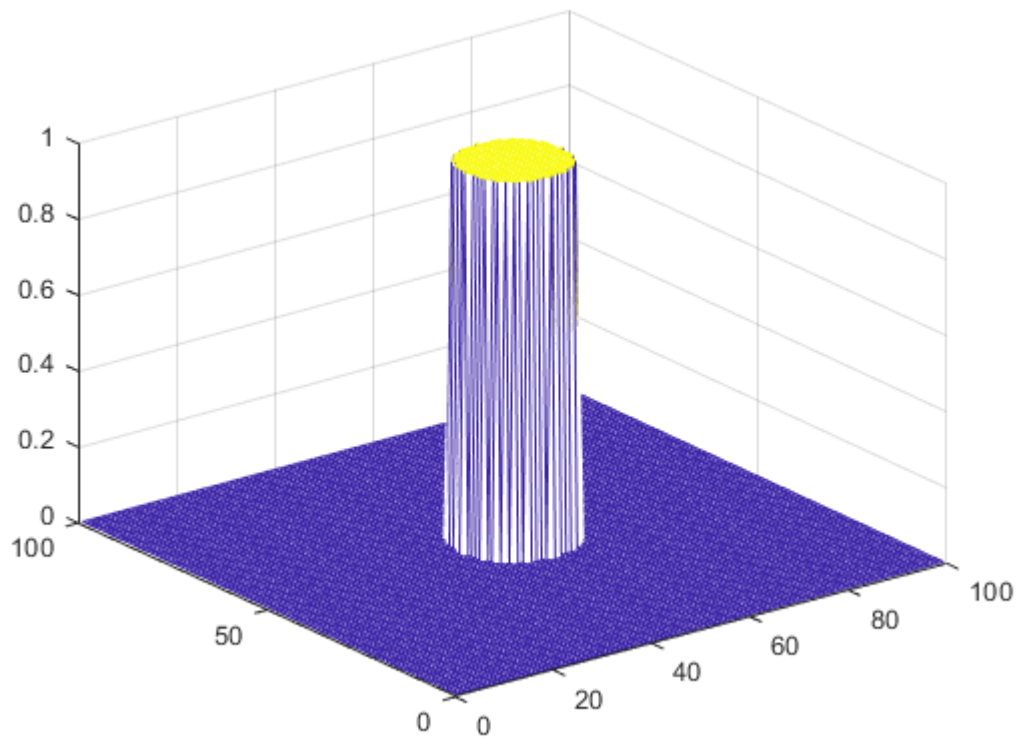
%H = lpfilter('ideal', 100, 100, 10);
H = my_lpfilter('ideal', 100, 100, 10);

figure,
subplot(1,2,1), imshow(H,[]);
subplot(1,2,2), imshow(fftshift(H),[]);

```



```
figure, mesh(fftshift(H)); % trojrozměrní zobrazení
```

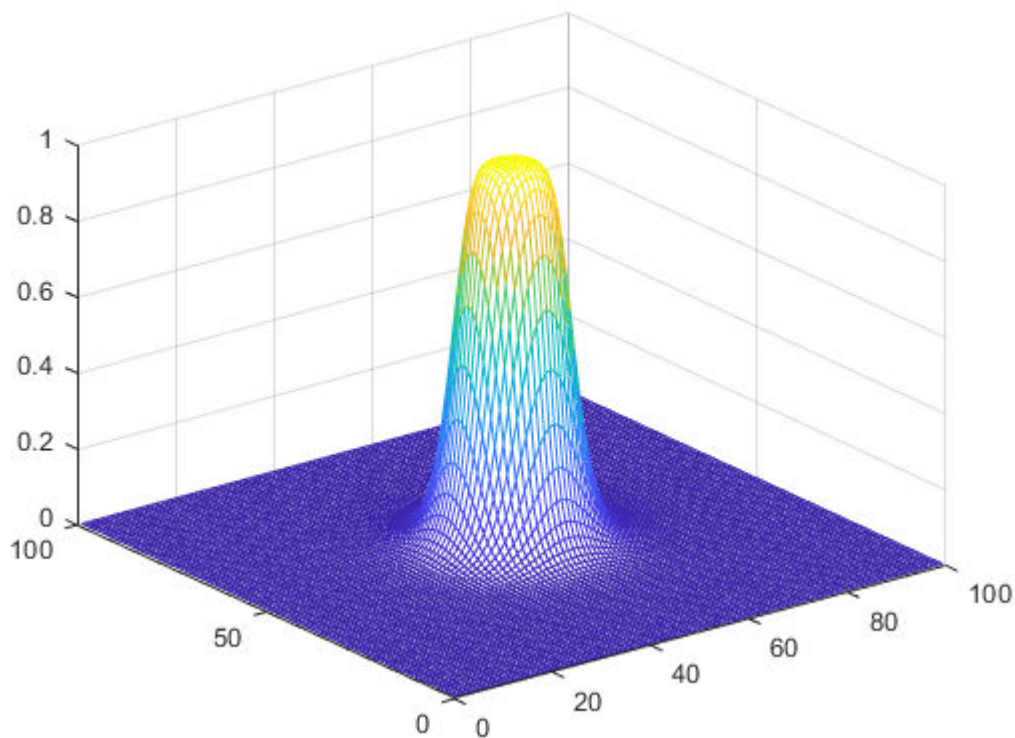


ÚKOL 2

Doprogramujte do `lpfilter()` Butterworth LP filtr

$$H(u, v) = \frac{1}{1 + [D(u, v)/D_0]^{2n}}$$

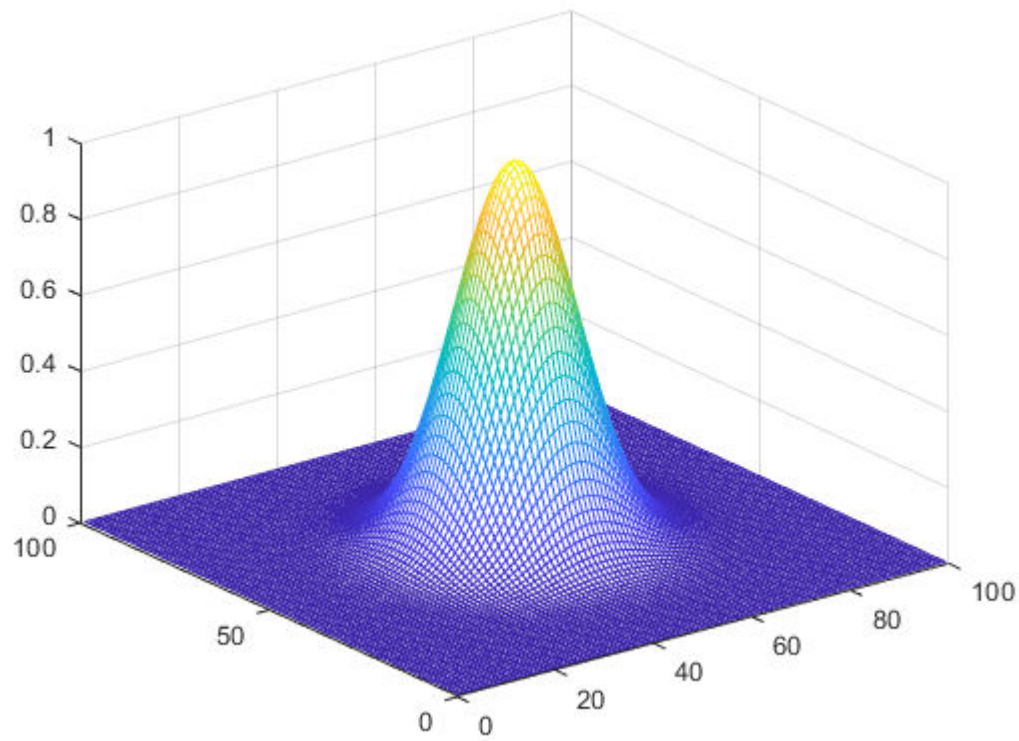
```
%H = lpfilter('btw', 100, 100, 10,3);  
H = my_lpfilter('btw', 100, 100, 10,3);  
%imshow(H,[]);  
%figure, imshow(fftshift(H),[]);  
figure, mesh(fftshift(H));
```



Gaussovský LP filtr je ve funkci `lpfilter` naprogramovaný.

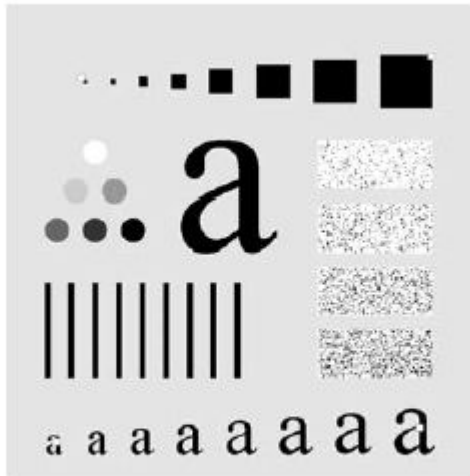
$$H(u, v) = e^{-D^2(u, v)/2\sigma^2}$$

```
H = lpfilter('gaussian', 100, 100, 10);  
%imshow(H,[]);  
%figure, imshow(fftshift(H),[]);  
figure, mesh(fftshift(H));
```



Aplikace lpfilteru

```
f = im2double(imread('a.png'));  
PQ = paddedsize(size(f));  
H = my_lpfilter('ideal', PQ(1), PQ(2), 500,2);  
gf = dftfilt(f, H);  
imshow(gf,[]);
```



ÚKOL 3

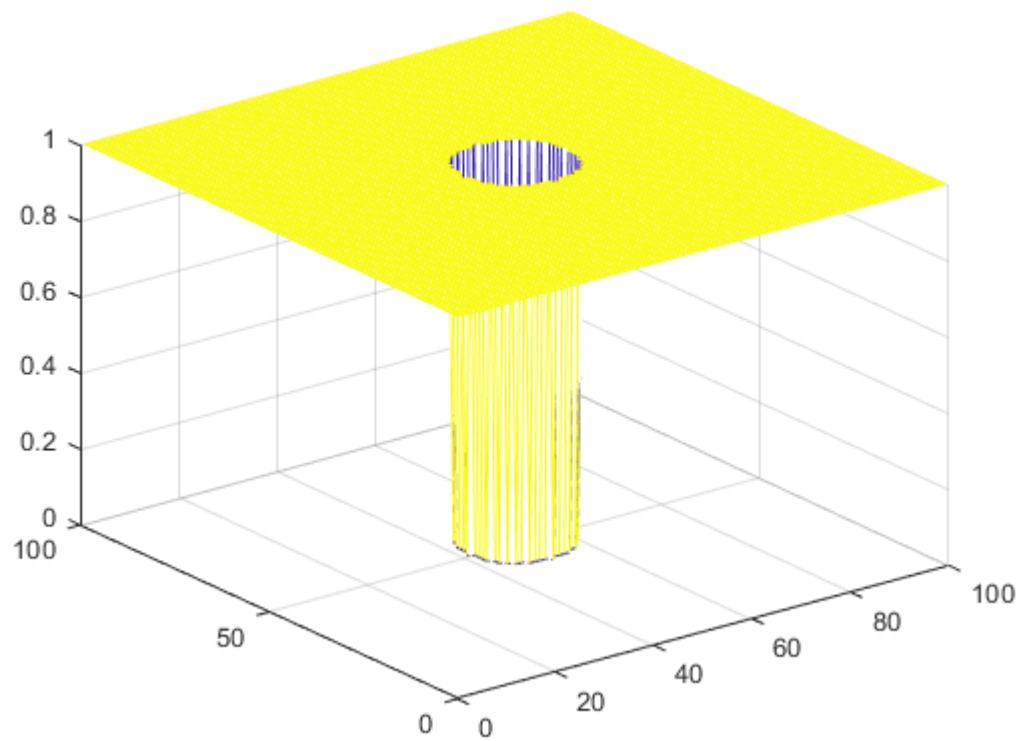
Porovnejte výsledné obrázky po aplikaci jednotlivých LP filtrů.

High pass filtry

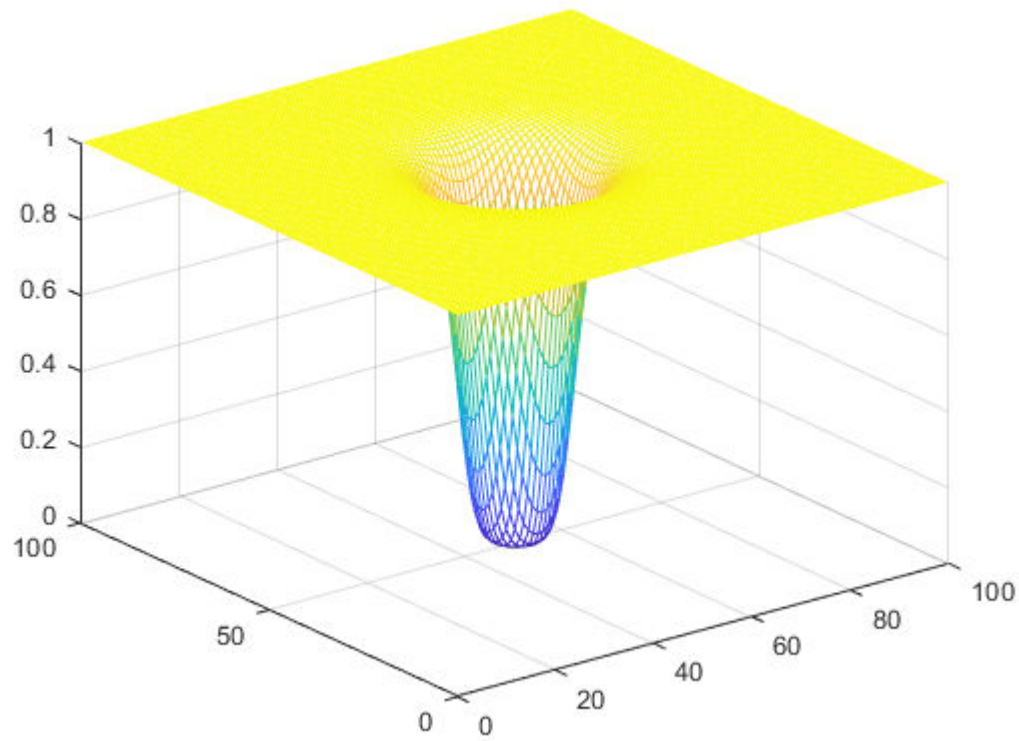
HP filtry zvýrazňují hrany a ostré přechody.

z LP filtrů získáme, tak, že je odečteme od jednotkové matice podívejte se na funkci `hpfiler`

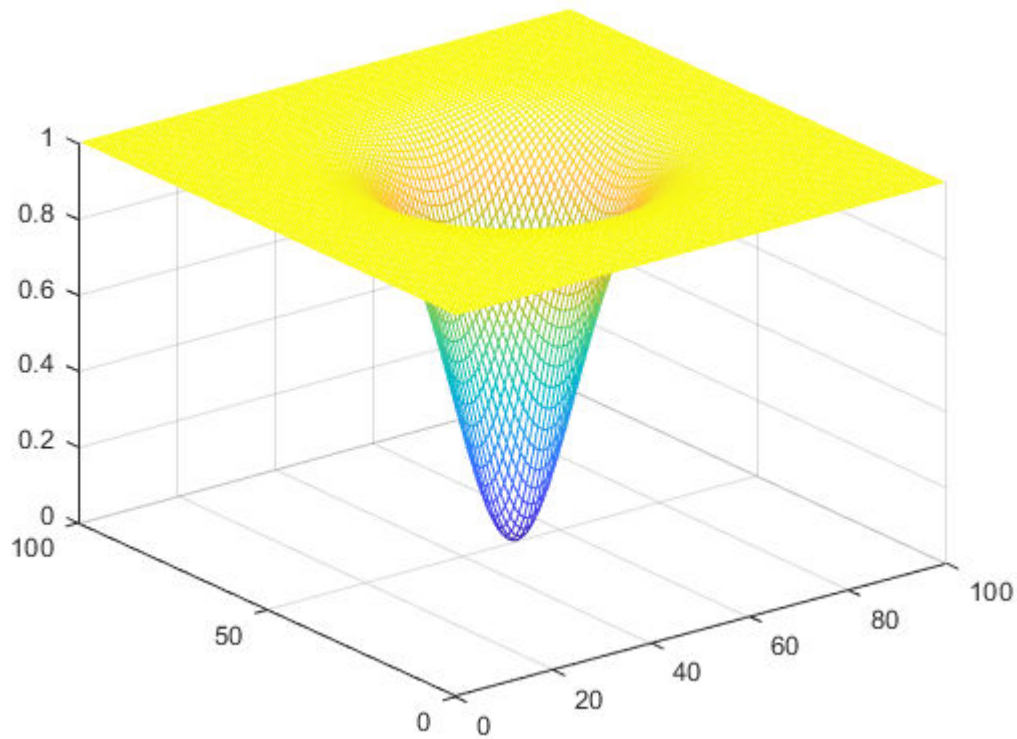
```
H = hpfiler('ideal', 100, 100, 10);  
figure, mesh(fftshift(H));
```



```
H = hpfilter('btw', 100, 100, 10,3);  
figure, mesh(fftshift(H));
```

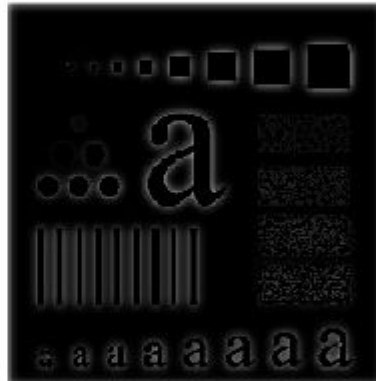
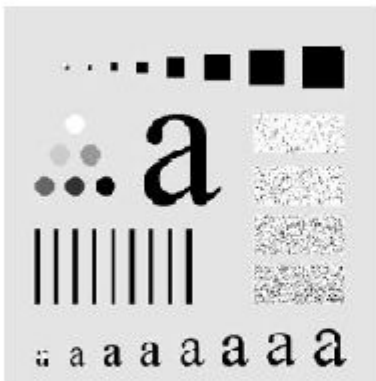
```
H = hpfilter('gaussian', 100, 100, 10);  
figure, mesh(fftshift(H));
```



HP filtry zvýrazňují hrany a ostré přechody. ALE! jak víte, tak informace o průměrné hodnotě je uložena v prvku $F(0,0)$ (dc koeficient), tento prvek jsme ale filtrem potlačili. Obrázek tedy ztratil většinu své barevné informace

```
f = im2double(imread('a.png'));
PQ = paddedsize(size(f));
D0 = 0.05*PQ(1);
H = hpfilter('gaussian',PQ(1),PQ(2),D0);
g = dftfilt(f,H);
```

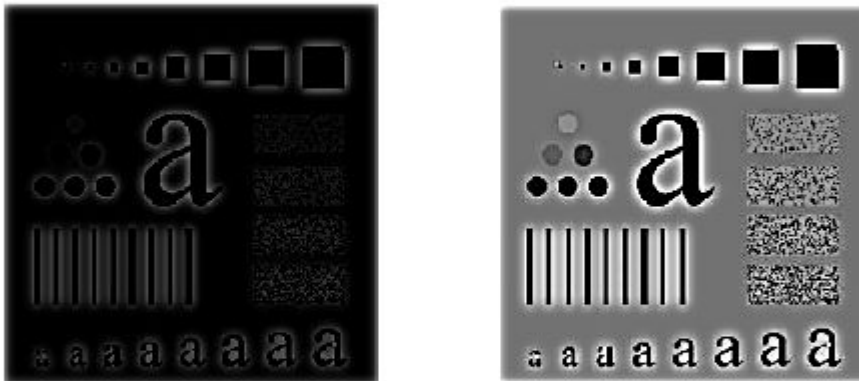
```
figure,
subplot(1,2,1), imshow(f);
subplot(1,2,2), imshow(g);
```



Tím, že je odstraněn dc koeficient, bere se, že průměrná hodnota je 0. Abychom toto kompenzovali, tak se přidává k výslednému obrázku offset. Pokud navíc vynásobíme filtr nějakým koeficientem větším než 1, dostaneme tzv. high-frequency emphasis filter.

```
f = im2double(imread('a.png'));
PQ = paddedsize(size(f));
D0 = 0.05*PQ(1);
H = hpfilt('gaussian',PQ(1),PQ(2),D0);
g = dftfilt(f,H);
H2 = 0.5+2*H;
g2 = dftfilt(f,H2);

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



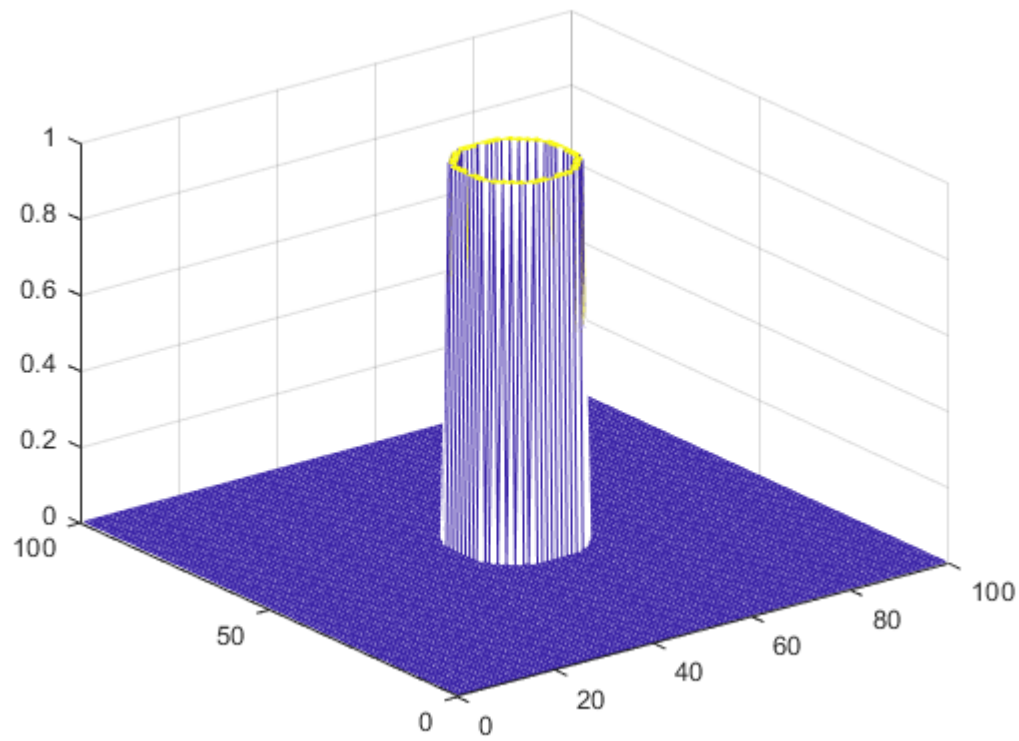
Selektivní filtry

pokud nechceme odstranit všechny nízké/vysoké frekvence, jen nějaké určité, použijeme tzv. selektivní filtry.

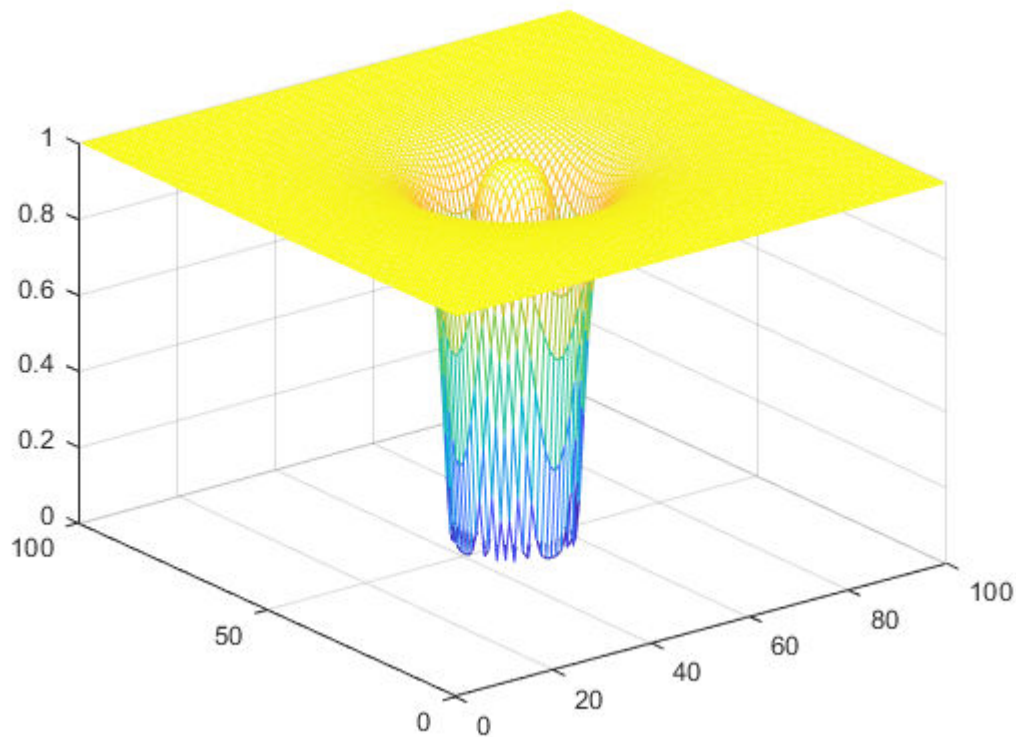
Band reject/pass filtr

prvním příkladem je band reject/pass filtr. Která je snadné zkonstruovat z LPF a HPF.

```
H = bandfilter('ideal', 'pass', 100, 100, 10, 2, 0);  
figure, mesh(fftshift(H));
```



```
H = bandfilter('btw', 'reject', 100, 100, 10, 2, 3);  
figure, mesh(fftshift(H));
```



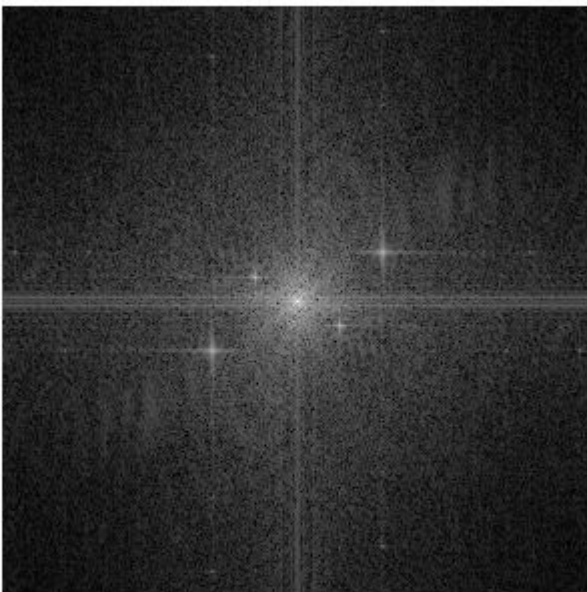
NOTCH

Nejužitečnější filtry. Potlačují, nebo zvýrazňují hodnoty ve speciálním okolí (kolem středu frekvenčního obdélníku). Musí být symetrické kolem středu. Pokud je střed notchu v (u_0, v_0) pak musí být notch i v $(-u_0, -v_0)$. Konstruují se stejně jako high pass (low pass) filtry, ale posunuté do středu notchu. Typickým použitím je použití na filtraci periodického šumu. Viz následující příklad. Když si zobrazíme spektrum vidíme vysoké frekvence i mimo střed. Tyto vysoké frekvence budeme chtít odfiltrovat.

```
f = im2double(rgb2gray(imread('ClownOrig.jpg')));
[M,N] = size(f);
figure, imshow(f);
```

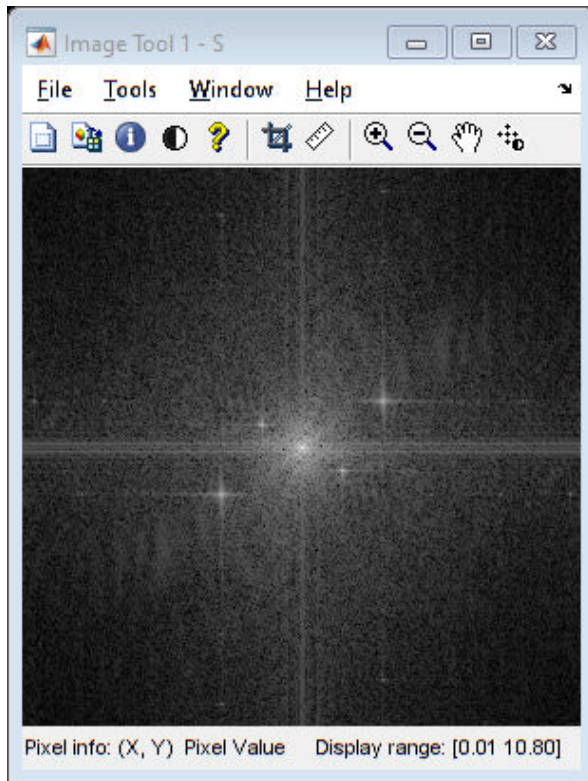


```
F = fft2(f);  
S = fftshift(log(1+abs(F)));  
figure, imshow(S,[]);
```



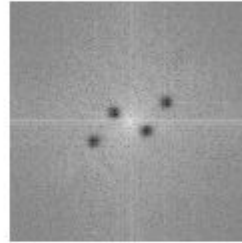
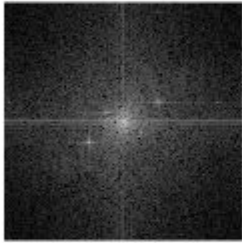
Pomocí imtool nástroje můžeme zjistit středy notchů

```
imtool(S,[]);
```



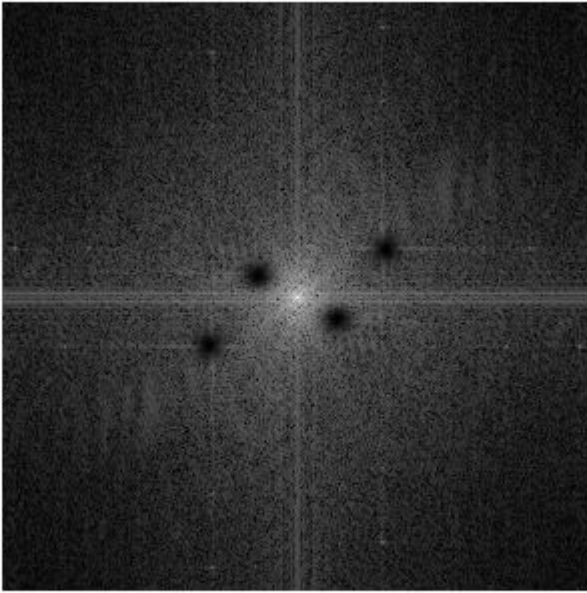
Např. notch zde:

```
C1 = [136 127;  
      123 191];  
H1 = cnotch('gaussian','reject', M,N,C1,5);  
  
figure,  
subplot(1,3,1), imshow(S,[]);  
subplot(1,3,2), imshow(fftshift(H1),[]);  
subplot(1,3,3), imshow(imfuse(fftshift(H1), S, 'blend'));
```

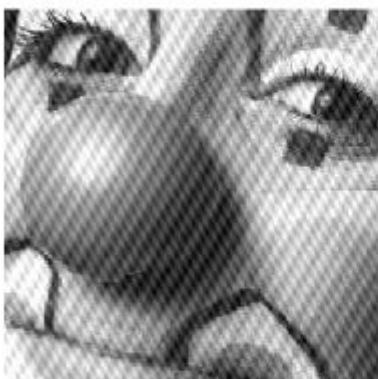



aplikace

```
P1 = fftshift(H1).*S;  
figure, imshow(P1,[]);
```



```
g1 = dftfilt(f, H1);  
figure  
subplot(1,2,1),imshow(f);  
subplot(1,2,2), imshow(g1);
```



ÚKOL 4

zkuste z obrázku odfiltrvat i periodický šum, který je stále přítomný na okrajích obrázku.

Rekonstrukce obrazu

Operace s obrázkem můžeme rozdělit na 2 skupiny. Jednak jsou to operace, které SUBJEKTIVNĚ vylepšují vzhled obrázku. Uživatel přímo sám podle svého subjektivního vnímání říká, zda se mu tento vzhled líbí, hodí ke konkrétní aplikaci a pod. Mezi takové operace patří jasové transformace, geometrické operace. Druhou skupinu pak tvoří operace, které OBJEKTIVNĚ vylepšují vzhled obrázku. Jsou jimi metody rekonstrukce obrazu. Jsou založené na nějakém matematickém modelu degradace obrazu.

Matematický model degradace:

Máme vstupní obraz f , který je degradován funkcí H plus k tomuto obrazu může být přidána funkce šumu. Degradovaný obraz tedy získáme:

$$g = H(f) + n$$

Pokud máme nějaké informace o degradační funkci a šumu, můžeme odhadnout originální obraz f (což je úkolem rekonstrukce obrazu).

Degradační funkci můžeme modelovat pomocí konvoluce (*) v prostorové doméně:

$$g = h * f + n$$

Případně z konvolučního teorému víme, že to odpovídá násobení ve frekvenční doméně.

$$G = HF + N$$

Šum

Pokud je obraz degradovaný pouze šumem pak

$$G = F + N$$

Pokud chceme odstranit pouze šum, pak potřebujeme znát model šumu. Zajímají nás parametry, které definují šum a zda je šum korelovaný s obrazem (zda je hodnota šumu závislá na hodnotě pixelu). Pokud budeme předpokládat, že šum není korelovaný s obrazem, pak se budeme zajímat o statistické chování intenzit v šumu, to můžeme chápat jako náhodné proměnné charakterizované probablistickou funkcí (probability density function PDF)

Šum můžeme do obrazu přidat pomocí funkce `imnoise()`

$$g = \text{imnoise}(f, \text{type}, \text{parameters})$$

F je vstupní obraz, type a parameters jsou argumenty, které se vážou na to, jaký chceme do obrazu přidat.

help `imnoise`

imnoise Add noise to image.

`J = imnoise(I,TYPE,...)` Add noise of a given TYPE to the intensity image I. TYPE is a string or char vector that can have one of these values:

'gaussian'	Gaussian white noise with constant mean and variance
'localvar'	Zero-mean Gaussian white noise with an intensity-dependent variance
'poisson'	Poisson noise
'salt & pepper'	"On and Off" pixels
'speckle'	Multiplicative noise

Depending on TYPE, you can specify additional parameters to **imnoise**. All numerical parameters are normalized; they correspond to operations with images with intensities ranging from 0 to 1.

`J = imnoise(I,'gaussian',M,V)` adds Gaussian white noise of mean M and variance V to the image I. When unspecified, M and V default to 0 and 0.01 respectively.

`J = imnoise(I,'localvar',V)` adds zero-mean, Gaussian white noise of local variance, V, to the image I. V is an array of the same size as I.

`J = imnoise(I,'localvar',IMAGE_INTENSITY,VAR)` adds zero-mean, Gaussian noise to an image, I, where the local variance of the noise is a function of the image intensity values in I. IMAGE_INTENSITY and VAR are vectors of the same size, and `PLOT(IMAGE_INTENSITY,VAR)` plots the

functional relationship between noise variance and image intensity. IMAGE_INTENSITY must contain normalized intensity values ranging from 0 to 1.

`J = imnoise(I,'poisson')` generates Poisson noise from the data instead of adding artificial noise to the data. If `I` is double precision, then input pixel values are interpreted as means of Poisson distributions scaled up by `1e12`. For example, if an input pixel has the value `5.5e-12`, then the corresponding output pixel will be generated from a Poisson distribution with mean of 5.5 and then scaled back down by `1e12`. If `I` is single precision, the scale factor used is `1e6`. If `I` is `uint8` or `uint16`, then input pixel values are used directly without scaling. For example, if a pixel in a `uint8` input has the value `10`, then the corresponding output pixel will be generated from a Poisson distribution with mean `10`.

`J = imnoise(I,'salt & pepper',D)` adds "salt and pepper" noise to the image `I`, where `D` is the noise density. This affects approximately `D*numel(I)` pixels. The default for `D` is `0.05`.

`J = imnoise(I,'speckle',V)` adds multiplicative noise to the image `I`, using the equation $J = I + n \cdot I$, where `n` is uniformly distributed random noise with mean 0 and variance `V`. The default for `V` is `0.05`.

Note

The mean and variance parameters for 'gaussian', 'localvar', and 'speckle' noise types are always specified as if for a double image in the range `[0, 1]`. If the input image is of class `uint8` or `uint16`, the `imnoise` function converts the image to double, adds noise according to the specified type and parameters, and then converts the noisy image back to the same class as the input.

Class Support

For most noise types, `I` can be `uint8`, `uint16`, `double`, `int16`, or `single`. For Poisson noise, `int16` is not allowed. The output image `J` has the same class as `I`. If `I` has more than two dimensions it is treated as a multidimensional intensity image and not as an RGB image.

Example

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper', 0.02);
figure, imshow(I), figure, imshow(J)
```

See also `rand`, `randn`.

Documentation for `imnoise`
Other functions named `imnoise`

Použití

```
f = imread('obr2.png');
%g = imnoise(f,'salt & pepper',0.02);
g = imnoise(f,'gaussian',0, 0.01);

subplot(1,2,1), imshow(f,[]);
subplot(1,2,2), imshow(g,[]);
```



ÚKOL 5

Vyzkoušejte si přidat do obrazu různé typy šumu a pohrajte si i s velikostí ostatních parametrů.

Periodický šum

První šum, který uvažujeme a je závislý na hodnotě pixelů. Tento šum je vhodné filtrovat ve frekvenční doméně.

```
f = im2double(imread('obr2.png'));

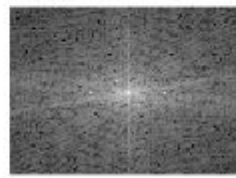
x = linspace(0,1 ,size(f,2));
y = linspace(0,1 ,size(f,1));
[X, Y] = meshgrid(x, y);

sum1 = f + 0.25*cos(32*pi*X);
sum2 = sum1 + 0.25*sin(10*pi*(X+Y));

S1 = fftshift(log(1+abs(fft2(sum1))));
S2 = fftshift(log(1+abs(fft2(sum2))));

figure,
subplot(2,3,1), imshow(sum1);
subplot(2,3,2), imshow(S1, []);
subplot(2,3,3), imshow(S1(348:420,462:562), []);
subplot(2,3,4), imshow(sum2);
```

```
subplot(2,3,5), imshow(S2, []);  
subplot(2,3,6), imshow(S2(348:420,462:562), []);
```



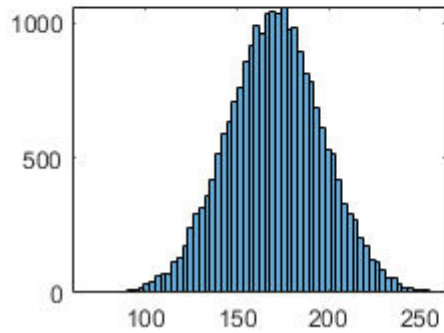
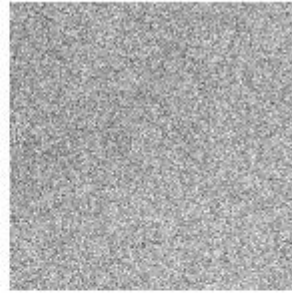
Odstranění šumu

Odhad parametrů šumu

Periodický šum se od ostatních snadno rozezná. vypadá, jako by byl do obrazu přidán nějaký pravidelný vzor. Ostatní typy šumu se odhadují podle PDF. A to konkrétně tak, že se vezme část obrazu s relativně konstantním pozadím a vypočítá se histogram a z jeho tvaru se odvodí typ a jiné parametry šumu (průměr, variance).

```
f = imread('obr2.png');  
%g = imnoise(f,'salt & pepper',0.02);  
g = imnoise(f,'gaussian',0, 0.01);  
subf =g(1:150,1:150); %výběr části obrázku  
%imshow(subf);
```

```
figure,  
subplot(2,2,1), imshow(f);  
subplot(2,2,2), imshow(subf);  
subplot(2,2,3), histogram(subf)
```



U periodického šumu se parametry odhadují analýzou fourierova spektra. Výkyvy jsou znatelné pouhým okem. Je možná i automatická analýza, pokud jsou výkyvy hodně znatelné, nebo známe nějakou vlastnost šumu.

Pokud je obraz degradován pouze šumem, převážně se používají filtry v prostorové doméně (viz cvičení 5 = lineární filtry `imfilter`, nelineární `medfilt2`).

K odstranění periodického šumu se používají Notch reject filtry.