

domaci2_18_018

December 4, 2021

1 Drugi domaci zadatak iz predmeta Digitalna obrada slike

1.1 Danica Bandović 2018/0018

1.2 1.Zadatak

Potrebno je izvršiti restauraciju slike u boji koja je dobijena polutoniranjem.

1.2.1 Ucitavanje potrebnih biblioteka

```
[1]: from pylab import *
import skimage
from skimage import util
from skimage import color
from skimage import exposure

from scipy import ndimage
from skimage import restoration
from skimage import io

import numpy as np
```

```
[7]: #ucitavanje slike half_tone.jpg
img = skimage.img_as_float(imread('../sekvence/half_tone.jpg'))

#izdvajanje R,G i B komponente
R = img[:, :, 0]
G = img[:, :, 1]
B = img[:, :, 2]

#originalna slika i njene R.G i B komponenta
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14,10), dpi=120)
ax = axes.ravel()

ax[0].imshow(img); ax[0].set_title('Ulagana slika', fontsize=12); ax[0].axis('off');
ax[1].imshow(R, vmin=0, vmax=1, cmap='jet'); ax[1].set_title('R', fontsize=12);
ax[1].axis('off')
```

```

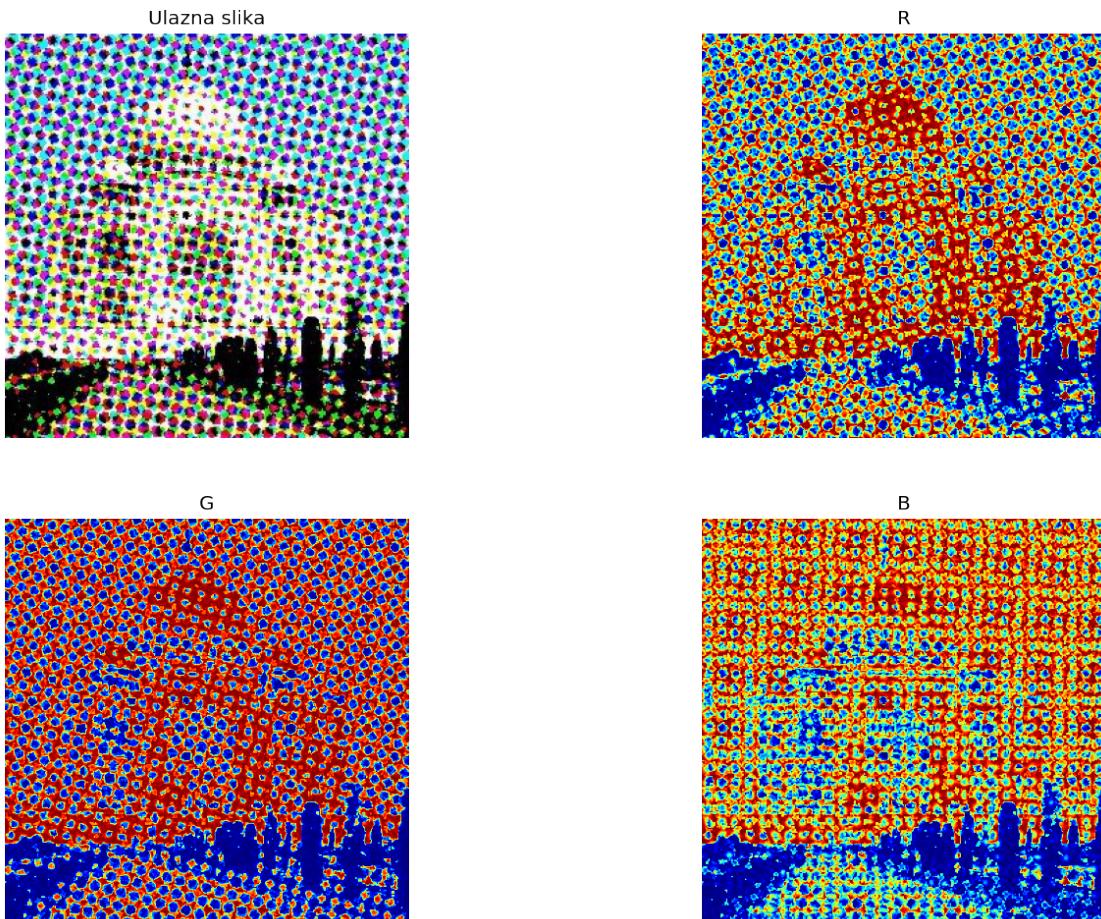
ax[2].imshow(G, vmin=0, vmax=1, cmap = 'jet'); ax[2].set_title('G', fontweight='bold', fontsize=12); ax[2].axis('off');

ax[3].imshow(B, vmin=0, vmax=1, cmap='jet'); ax[3].set_title('B', fontweight='bold', fontsize=12); ax[3].axis('off')

plt.show()

#histogram slice
hist, bin_edge = np.histogram(img.flatten(), bins=256, range=(0,255))
plt.figure(); plt.bar(bin_edge[0:-1], hist); title('Histogram slice');
plt.xlim(0, 50)

```



Preci cemo u frekvencijski domen kako bismo videli koje učestanosti su zastupljene. Izvršićemo filtriranje u frekvencijskom domenu u sve tri komponente, i to istim filtrima, kako bi odnos boja na slici ostao isti.

```
[8]: #prelazak u frekvencijski domen i prikaz spektra, R komponenta slike
img_fft = fftshift(fft2(R))
```

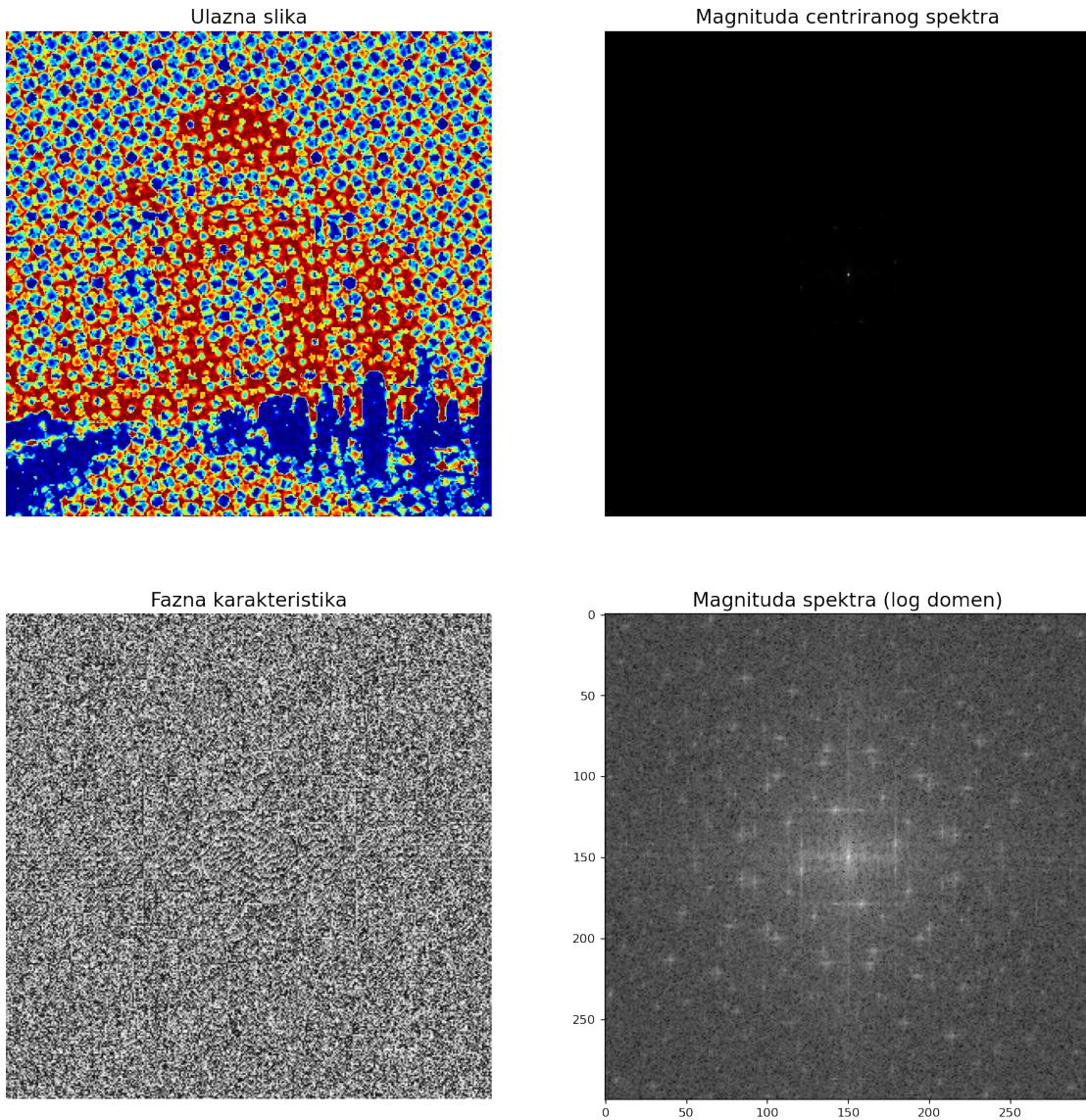
```

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,16), dpi=120)
ax = axes.ravel()

ax[0].imshow(R, cmap='jet'); ax[0].set_axis_off(); ax[0].set_title('Ulazna slika', fontsize=16)
ax[1].imshow(abs(img_fft), cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Magnituda centriranog spektra', fontsize=16)
ax[2].imshow(np.angle(img_fft), cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('Fazna karakteristika', fontsize=16)
ax[3].imshow(log(1+abs(img_fft)), cmap='gray'); ax[3].set_title('Magnituda spektra (log domen)', fontsize=16)

plt.show()

```



Sa fazne karakteristike ne možemo zaključiti ništa iz razloga što na slici ima previše šuma, dok u amplitudskom delu spektra vidimo mnogo pikova. Pokušaćemo da promenom amplitudske karakteristike potisnemo učestanosti koje dolaze od degradacije, a sa druge strane, želimo da ne narušimo previše faznu karakteristiku, jer se u njoj nalaze bitne informacije o ivicama. Vidimo da postoje odredjene smetenja na slici u vidu tackica koje se ponavljaju, tako da je ideja primeniti ili noc filter, ili selektivno filtriranje. Pošto previše pikova ima u celom spektru, a najbitniji deo slike je oko DC komponente, odnosno u centru slike, ideja je da primenimo low pass filter kako bismo potisnuli pikove na visokim učestanostima jer oni ne pripadaju originalnoj slici.

```
[9]: def lpfilter(filt_type, Ny, Nx, sigma, n=1):

    if (Ny%2 == 0):
        y = np.arange(0,Ny) - Ny/2 + 0.5
    else:
        y = np.arange(0,Ny) - (Ny-1)/2

    if (Nx%2 == 0):
        x = np.arange(0,Nx) - Nx/2 + 0.5
    else:
        x = np.arange(0,Nx) - (Nx-1)/2

    X, Y = meshgrid(x, y)

    D = np.sqrt(np.square(X) + np.square(Y))

    if filt_type == 'gaussian':
        filter_mask = exp(-np.square(D)/(2*np.square(sigma)))
    elif filt_type == 'btw':
        filter_mask = 1/(1+(D/sigma)**(2*n))
    elif filt_type == 'ideal':
        filter_mask = ones([Ny, Nx])
        filter_mask[D>sigma] = 0
    else:
        print('Greška! Nije podržan tip filtra: ', filt_type)
        return

    return filter_mask
```

```
[10]: #primena low pass filtra
[Ny, Nx] = shape(R)

img_fft = fftshift(fft2(R))
#lp1_filter_freq = lpfilter('ideal', Ny, Nx, 25)
```

```

lp2_filter_freq = lpfilter('gaussian', Ny, Nx, 20)

#lp3_filter_freq = lpfilter('ideal', Ny, Nx, 80)
#lp4_filter_freq = lpfilter('ideal', Ny, Nx, 60)

#lp5_filter_freq = lpfilter('ideal', Ny, Nx, 130)
#lp6_filter_freq = lpfilter('ideal', Ny, Nx, 100)

#br_filter_freq = 1 - lp1_filter_freq + lp2_filter_freq - lp3_filter_freq +_
#→ lp4_filter_freq - lp5_filter_freq + lp6_filter_freq
#br_filter_freq = 1 - lp1_filter_freq + lp2_filter_freq

br_filter_freq = lp2_filter_freq
img_fft_filt = img_fft*br_filter_freq

img_filt = real(ifft2(ifftshift(img_fft_filt)))

img_filt = np.clip(img_filt, 0, 1)

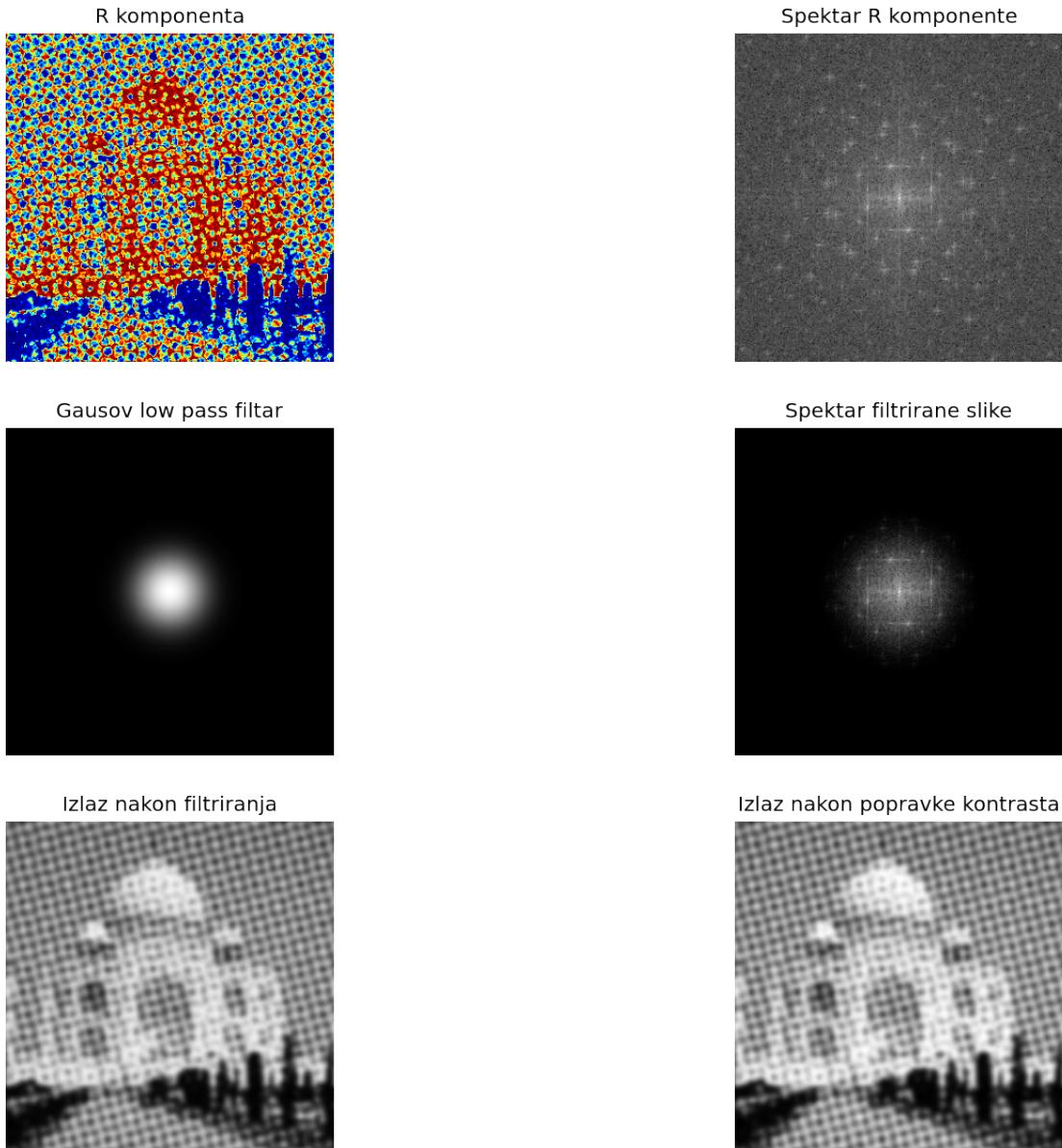
img_filt_enhanced = exposure.rescale_intensity(img_filt)

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(14,12), dpi=120)
ax = axes.ravel()

ax[0].imshow(R, vmin=0, vmax=1, cmap='jet'); ax[0].set_axis_off(); ax[0].
→set_title('R komponenta')
ax[1].imshow(log(1+abs(img_fft)), cmap='gray'); ax[1].set_axis_off(); ax[1].
→set_title('Spektar R komponente')
ax[2].imshow(log(1+abs(br_filter_freq)), cmap='gray'); ax[2].set_axis_off();_
→ax[2].set_title('Gausov low pass filtrar')
ax[3].imshow(log(1+abs(img_fft_filt)), cmap='gray'); ax[3].set_axis_off();_
→ax[3].set_title('Spektar filtrirane slike')
ax[4].imshow(img_filt, cmap='gray', vmin=0, vmax=1); ax[4].set_axis_off();_
→ax[4].set_title('Izlaz nakon filtriranja')
ax[5].imshow(img_filt_enhanced, cmap='gray', vmin=0, vmax=1); ax[5].
→set_axis_off(); ax[5].set_title('Izlaz nakon popravke kontrasta');

imsave('Rspektar.jpg',log(1+abs(img_fft_filt)))

```



Na slikama iznad prikazani su međukoraci i rezultat nakon primene niskofrekventnog filtra. Za uzimanje širine opsega bilo je potrebno naći kompromis između potiskivanja šuma i oštine slike. Ukoliko se uzme preveliki opseg koji se propušta kroz filter, previše pikova će biti propušteno, te nećemo popraviti kvalitet slike mnogo u smislu potiskivanja degradacije. Sa druge strane, ukoliko bi se propustio mali deo spektra slika će se previše zamutiti, odnosno doći će do degradacije ivica i time je oština izgubljena i nećemo moći popraviti oštinu kasnije. Iz tog razloga uzeli smo širinu spektra koja pravi neki kompromis između ova dva. Gausov filter takođe daje bolje rezultate od idealnog jer će se gausovim dobiti bolja oština slike. Takođe, pokušano je filtriranje izvršiti i band reject filrima, ali je low pass filter ipak dao bolje rezultate jer je previše pikova u spektru koji treba da su potisnuti.

Sa dobijenog spektra se vidi da i dalje imamo pikove koje treba da potisnemo, ali pošto ih ima manje, sada možemo da te pikove uklonimo primenom noč filtera. Kako bi se primenio noč filter, prvo je na prikazu amplitudske karakteristike nađeno u kojim tačkama se nalaze pikovi koji treba da se uklone sa karakteristikom, i potom je primenjen noč reject filter oko tih tačaka. Za svaku komponentu slike korišćeni su noč filtri u istim tačkama da se ne bi poremetio odnos boja na slici. Rezultati su prikazani ispod. Takođe, izlazi su prikazani na sivoj slici samo kako bi se videlo bolje koliko šuma je potisnuto i koliko je ivica sačuvano.

```
[11]: def cnotch(filt_type, notch, Ny, Nx, C, r, n=1):
    N_filters = len(C)

    filter_mask = zeros([Ny,Nx])

    if (Ny%2 == 0):
        y = np.arange(0,Ny) - Ny/2 + 0.5
    else:
        y = np.arange(0,Ny) - (Ny-1)/2

    if (Nx%2 == 0):
        x = np.arange(0,Nx) - Nx/2 + 0.5
    else:
        x = np.arange(0,Nx) - (Nx-1)/2

    X, Y = meshgrid(x, y)

    for i in range(0, N_filters):
        C_current = C[i]

        C_complement = zeros([2,1])
        C_complement[0] = -C_current[0]
        C_complement[1] = -C_current[1]

        if (Ny%2 == 0):
            y0 = y - C_current[0] + Ny/2 - 0.5
        else:
            y0 = y - C_current[0] + (Ny-1)/2

        if (Nx%2 == 0):
            x0 = x - C_current[1] + Nx/2 - 0.5
        else:
            x0 = x - C_current[1] + (Nx-1)/2

        X0, Y0 = meshgrid(x0, y0)

        D0 = np.sqrt(np.square(X0) + np.square(Y0))

        if (Ny%2 == 0):
```

```

y0c = y - C_complement[0] - Ny/2 + 0.5
else:
    y0c = y - C_complement[0] - (Ny-1)/2

if (Nx%2 == 0):
    x0c = x - C_complement[1] - Nx/2 + 0.5
else:
    x0c = x - C_complement[1] - (Nx-1)/2

X0c, Y0c = meshgrid(x0c, y0c)

D0c = np.sqrt(np.square(X0c) + np.square(Y0c))

if filt_type == 'gaussian':
    filter_mask = filter_mask + \
        exp(-np.square(D0)/(2*np.square(r))) + \
        exp(-np.square(D0c)/(2*np.square(r)))
elif filt_type == 'btw':
    filter_mask = filter_mask + \
        1/(1+(D0/r)**(2*n)) + \
        1/(1+(D0c/r)**(2*n))
elif filt_type == 'ideal':
    filter_mask[(D0<=r) | (D0c<=r)] = 1
else:
    print('Greška! Nije podržan tip filtra: ', filt_type)
    return

if notch == 'pass':
    return filter_mask
elif notch == 'reject':
    return 1 - filter_mask
else:
    return

```

[13]: [Ny, Nx] = shape(img_filt_enhanced)

```
img_fft = fftshift(fft2(img_filt_enhanced))
```

```
C = [[142, 121], [158, 121], [121, 141], [121, 157], [116, 173], [129, 188], ↳
    ↳ [130, 113], [113, 129], [84, 135], ↳
    ↳ [104, 102], [150, 120], [172, 113], [187, 129], [180, 120], [121, 120],
    ↳
    ↳ [136, 84], [134, 93], [100, 107], [92, 134], [84, 163], [92, 164], [100, 195], [106, 201], [135, 208], [135, 208]
```

```

#C = [[121,142],[130,113],[143,121],[158,121],[172,113],[187,129],[180,143],[113,130]]
#C = [[143, 121], [160, 120], [121, 141], [120, 157], [116,173], [129,188], [130,113], [113,129], [84,135], [104,102], [172,113], [136,84], [134,93], [100,107], [92,134], [84,163], [92,164], [100,195], [106,201], [135,208], [135,208], [90,165], [92,134], [101,106], [138,83], [106,199], [134,91]]]
#C = [[150,120],[121,150]]
nr_filter_freq = cnotch('btw', 'reject', Ny, Nx, C, 2)

img_fft_filt = img_fft*nr_filter_freq

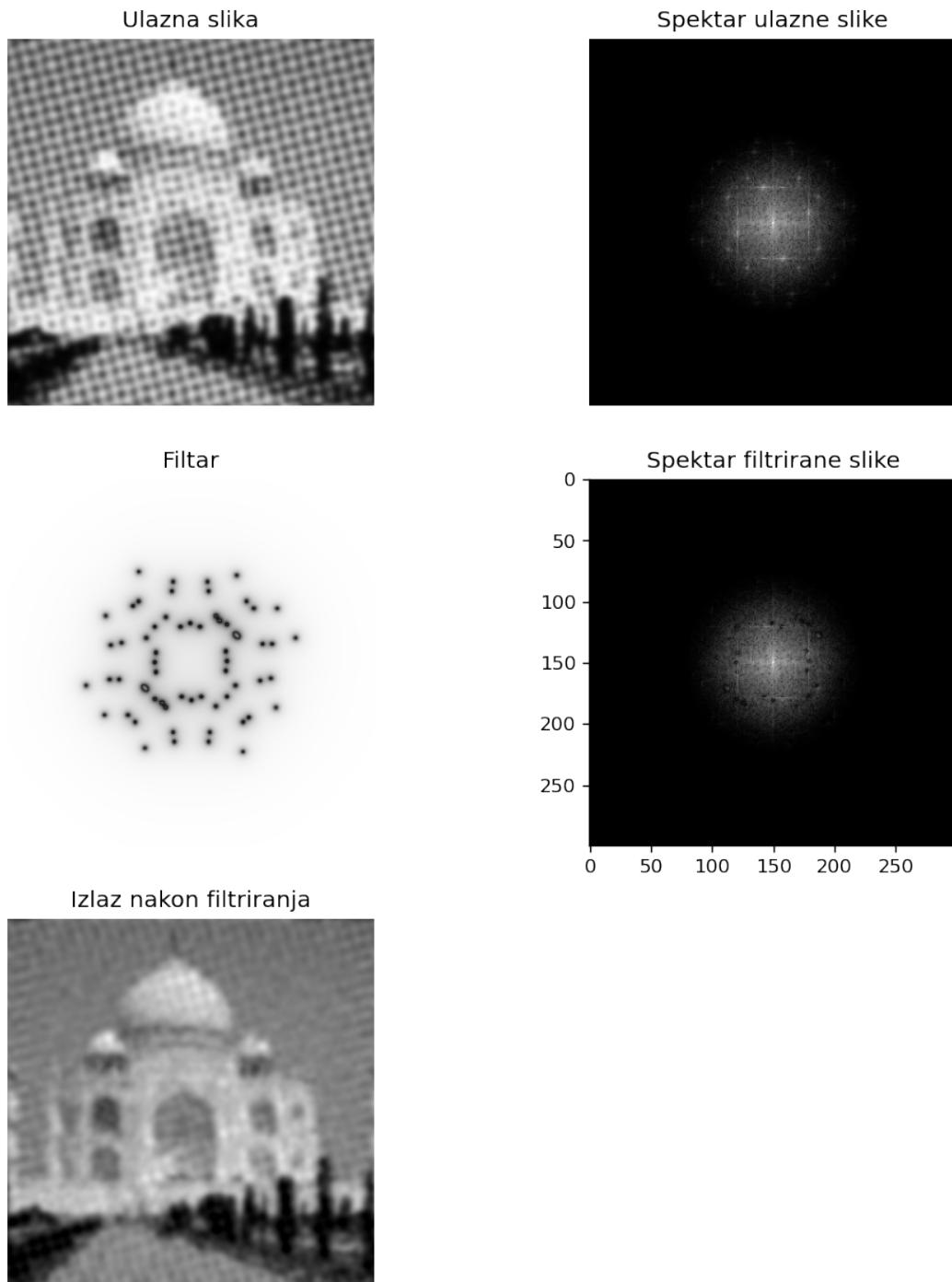
img_filt = real(ifft2(fftshift(img_fft_filt)))
img_filt = np.clip(img_filt, 0, 1)

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(10,12), dpi=120)
ax = axes.ravel()

ax[0].imshow(img_filt_enhanced, cmap='gray'); ax[0].set_axis_off(); ax[0].
    set_title('Ulazna slika')
ax[1].imshow(log(1+abs(img_fft)), cmap='gray'); ax[1].set_axis_off(); ax[1].
    set_title('Spektar ulazne slike')
ax[2].imshow(log(1+abs(nr_filter_freq)), cmap='gray'); ax[2].set_axis_off(); ax[2].
    set_title('Filtar')
ax[3].imshow(log(1+abs(img_fft_filt)), cmap='gray'); ax[3].set_title('Spektar
    filtrirane slike')
ax[4].imshow(img_filt, cmap='gray', vmin=0, vmax=1); ax[4].set_axis_off(); ax[4].
    set_title('Izlaz nakon filtriranja')
ax[5].set_axis_off();

filtR = img_filt

```



```
[14]: #filtriranje za G komponentu
img_fft = fftshift(fft2(G))
```

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,16), dpi=120)
ax = axes.ravel()
```

```

ax[0].imshow(G, cmap='jet'); ax[0].set_axis_off(); ax[0].set_title('Ulazna\u
↪slika', fontsize=16)
ax[1].imshow(abs(img_fft), cmap='gray'); ax[1].set_axis_off(); ax[1].
↪set_title('Magnituda centriranog spektra', fontsize=16)
ax[2].imshow(np.angle(img_fft), cmap='gray'); ax[2].set_axis_off(); ax[2].
↪set_title('Fazna karakteristika', fontsize=16)
ax[3].imshow(log(1+abs(img_fft)), cmap='gray'); ax[3].set_title('Magnituda\u
↪spektra (log domen)', fontsize=16)

[Ny, Nx] = shape(G)

img_fft = fftshift(fft2(G))
lp2_filter_freq = lpfilter('gaussian', Ny, Nx, 20)

br_filter_freq = lp2_filter_freq
img_fft_filt = img_fft*br_filter_freq

img_filt = real(ifft2(ifftshift(img_fft_filt)))

img_filt = np.clip(img_filt, 0, 1)

img_filt_enhanced = exposure.rescale_intensity(img_filt)

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(14,12), dpi=120)
ax = axes.ravel()

ax[0].imshow(G, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Ulazna\u
↪slika')
ax[1].imshow(log(1+abs(img_fft)), cmap='gray'); ax[1].set_axis_off(); ax[1].
↪set_title('Spektar ulazne slike')
ax[2].imshow(log(1+abs(br_filter_freq)), cmap='gray'); ax[2].set_axis_off(); u
↪ax[2].set_title('Filtar')
ax[3].imshow(log(1+abs(img_fft_filt)), cmap='gray'); ax[3].set_axis_off(); u
↪ax[3].set_title('Spektar filtrirane slike')
ax[4].imshow(img_filt, cmap='gray', vmin=0, vmax=1); ax[4].set_axis_off(); u
↪ax[4].set_title('Izlaz nakon filtriranja')
ax[5].imshow(img_filt_enhanced, cmap='gray', vmin=0, vmax=1); ax[5].
↪set_axis_off(); ax[5].set_title('Izlaz nakon popravke kontrasta');

imsave('Gspektar.jpg', log(1+abs(img_fft_filt)))

[Ny, Nx] = shape(img_filt_enhanced)

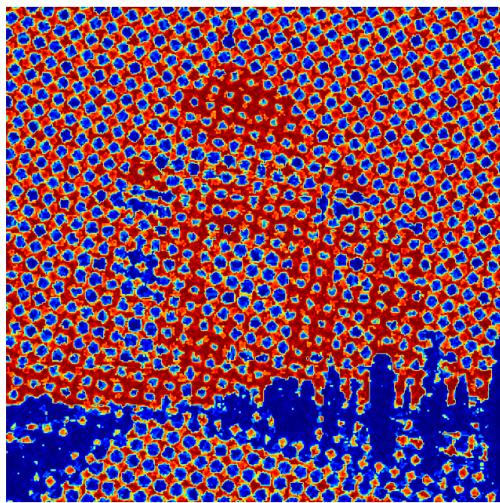
```

```

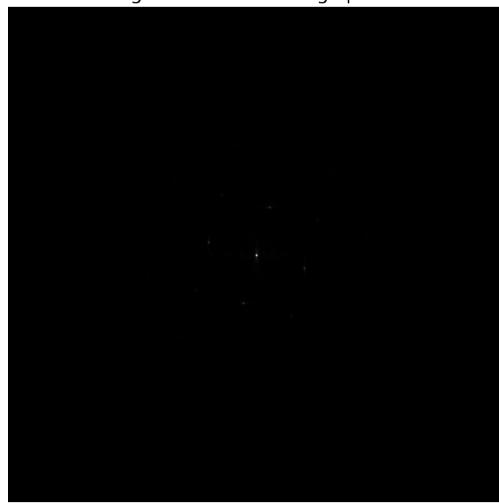


```

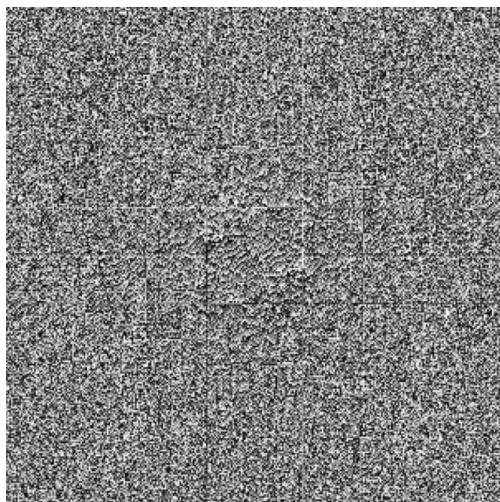
Ulazna slika



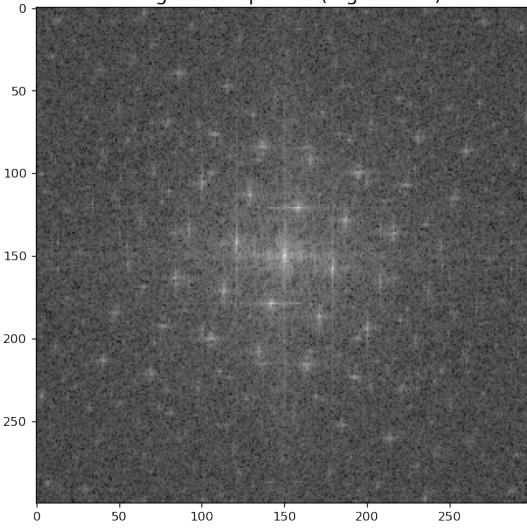
Magnituda centriranog spektra



Fazna karakteristika



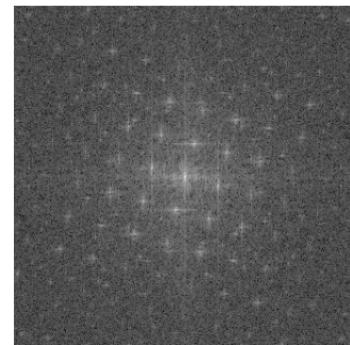
Magnituda spektra (log domen)



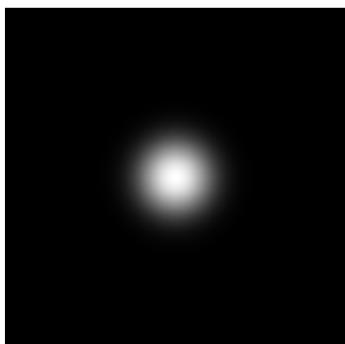
Ulazna slika



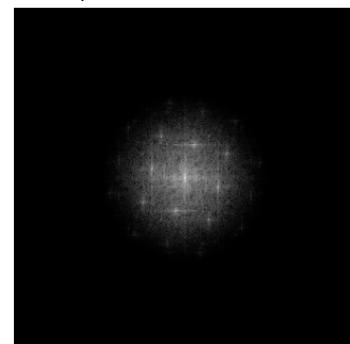
Spektar ulazne slike



Filtar



Spektar filtrirane slike

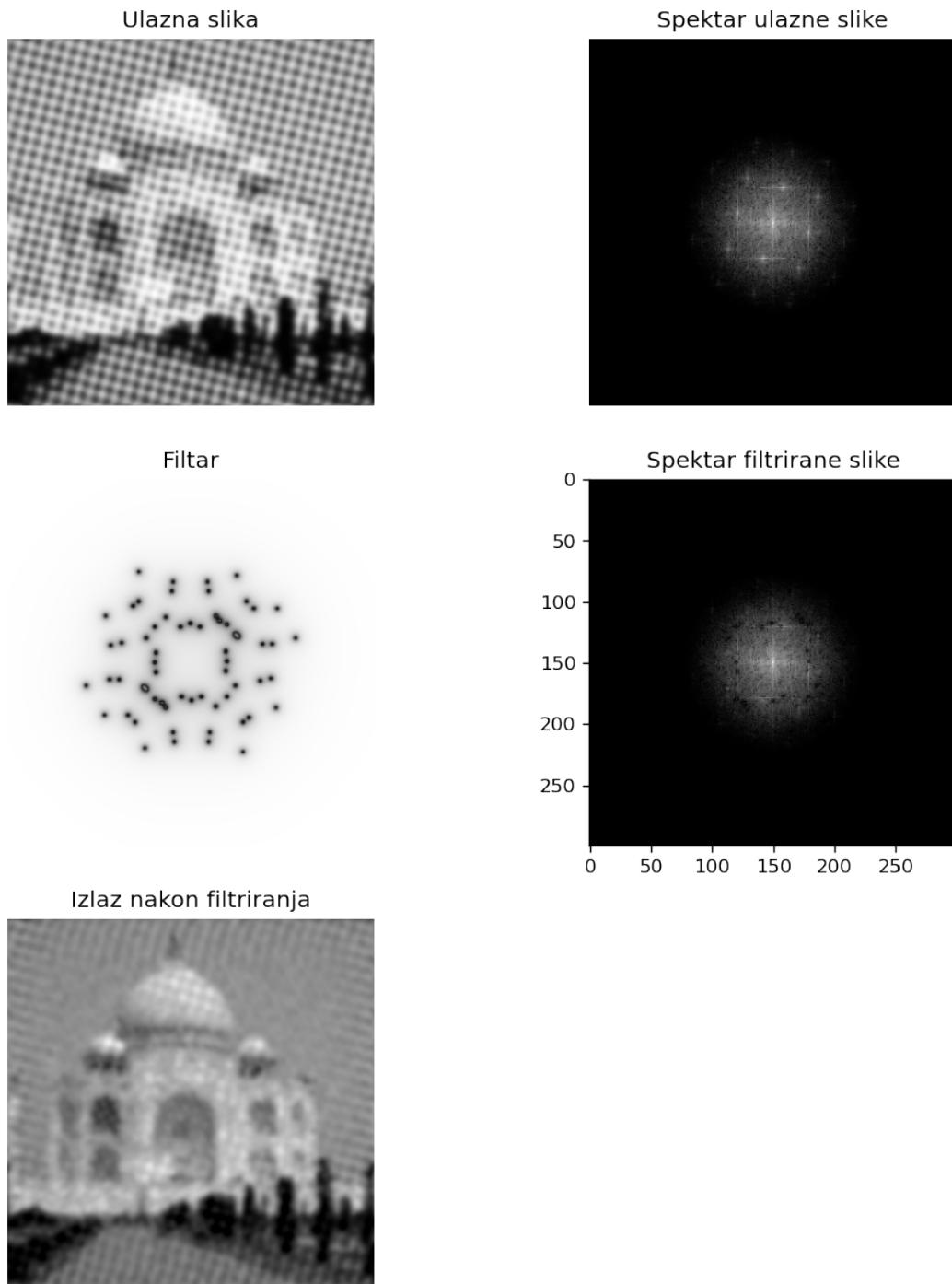


Izlaz nakon filtriranja



Izlaz nakon popravke kontrasta





```
[15]: #filtriranje B komponente
img_fft = fftshift(fft2(B))

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,16), dpi=120)
ax = axes.ravel()
```

```

ax[0].imshow(B, cmap='jet'); ax[0].set_axis_off(); ax[0].set_title('Ulazna slika', fontsize=16)
ax[1].imshow(abs(img_fft), cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Magnituda centriranog spektra', fontsize=16)
ax[2].imshow(np.angle(img_fft), cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('Fazna karakteristika', fontsize=16)
ax[3].imshow(log(1+abs(img_fft)), cmap='gray'); ax[3].set_title('Magnituda spektra (log domen)', fontsize=16)

[Ny, Nx] = shape(B)

img_fft = fftshift(fft2(B))
lp2_filter_freq = lpfilter('gaussian', Ny, Nx, 20)

br_filter_freq = lp2_filter_freq
img_fft_filt = img_fft*br_filter_freq

img_filt = real(ifft2(ifftshift(img_fft_filt)))

img_filt = np.clip(img_filt, 0, 1)

img_filt_enhanced = exposure.rescale_intensity(img_filt)

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(14,12), dpi=120)
ax = axes.ravel()

ax[0].imshow(B, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Ulazna slika')
ax[1].imshow(log(1+abs(img_fft)), cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Spektar ulazne slike')
ax[2].imshow(log(1+abs(br_filter_freq)), cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('Filtar')
ax[3].imshow(log(1+abs(img_fft_filt)), cmap='gray'); ax[3].set_axis_off(); ax[3].set_title('Spektar filtrirane slike')
ax[4].imshow(img_filt, cmap='gray', vmin=0, vmax=1); ax[4].set_axis_off(); ax[4].set_title('Izlaz nakon filtriranja')
ax[5].imshow(img_filt_enhanced, cmap='gray', vmin=0, vmax=1); ax[5].set_axis_off(); ax[5].set_title('Izlaz nakon popravke kontrasta');

imsave('Bspektar.jpg', log(1+abs(img_fft_filt)))

[Ny, Nx] = shape(img_filt_enhanced)

img_fft = fftshift(fft2(img_filt_enhanced))

```

```

C = [[142, 121], [158, 121], [121, 141], [121, 157], [116,173], [129,188], ↵
    ↵[130,113], [113,129], [84,135], ↵
    ↵[104,102],[150,120],[172,113],[187,129],[180,120],[121,120], ↵
    ↵
    ↵[136,84],[134,93],[100,107],[92,134],[84,163],[92,164],[100,195],[106,201],[135,208],[135,2
nr_filter_freq = cnotch('btw', 'reject', Ny, Nx, C, 2)

img_fft_filt = img_fft*nr_filter_freq

img_filt = real(ifft2(ifftshift(img_fft_filt)))
img_filt = np.clip(img_filt, 0, 1)

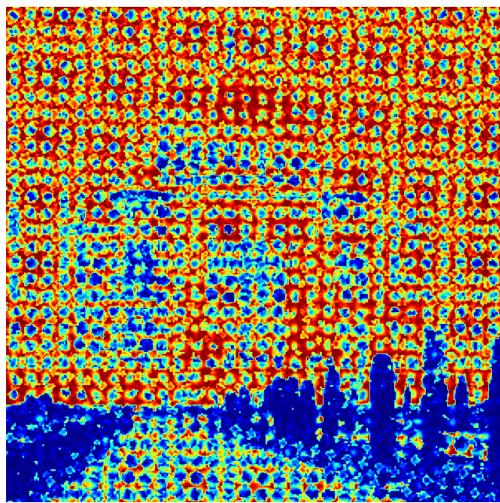
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(10,12), dpi=120)
ax = axes.ravel()

ax[0].imshow(img_filt_enhanced, cmap='gray'); ax[0].set_axis_off(); ax[0].
    ↵set_title('Ulagna slika')
ax[1].imshow(log(1+abs(img_fft)), cmap='gray'); ax[1].set_axis_off(); ax[1].
    ↵set_title('Spektar ulazne slike')
ax[2].imshow(log(1+abs(nr_filter_freq)), cmap='gray'); ax[2].set_axis_off(); ↵
    ↵ax[2].set_title('Filtar')
ax[3].imshow(log(1+abs(img_fft_filt)), cmap='gray'); ax[3].set_title('Spektar
    ↵filtrirane slike')
ax[4].imshow(img_filt, cmap='gray', vmin=0, vmax=1); ax[4].set_axis_off(); ↵
    ↵ax[4].set_title('Izlaz nakon filtriranja')
ax[5].set_axis_off();

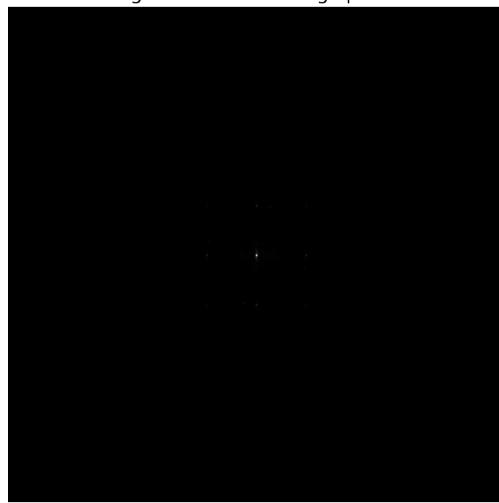
filtB = img_filt

```

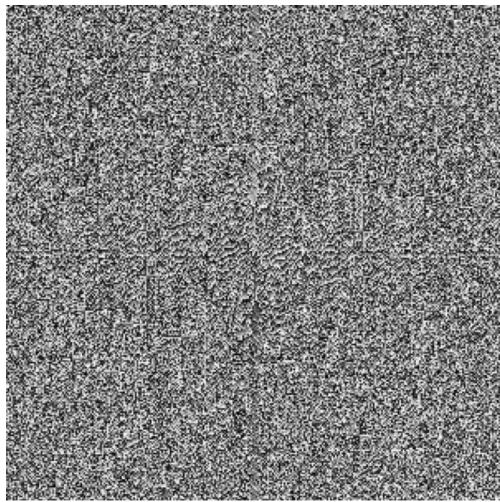
Ulazna slika



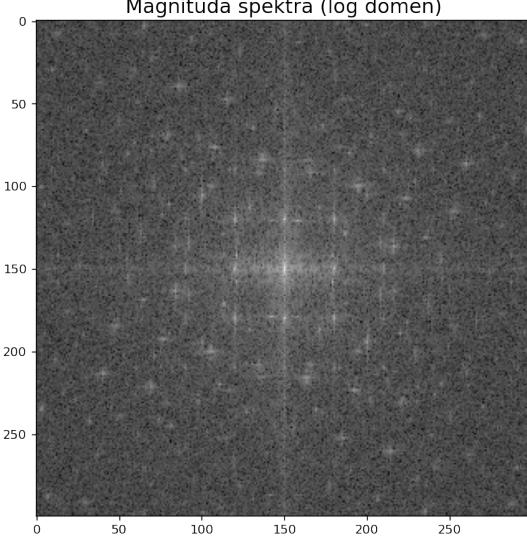
Magnituda centriranog spektra



Fazna karakteristika



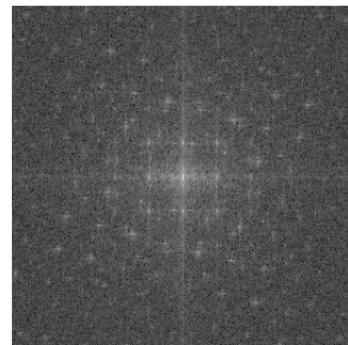
Magnituda spektra (log domen)



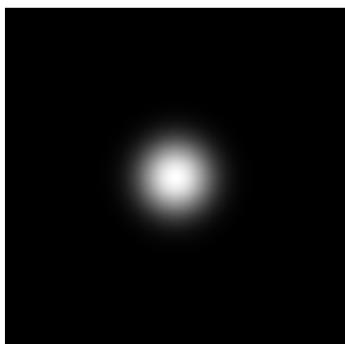
Ulazna slika



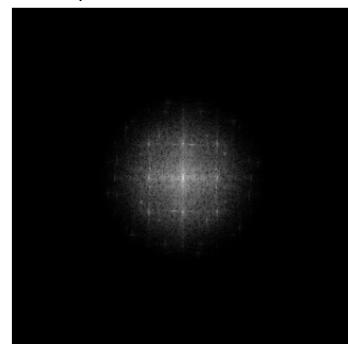
Spektar ulazne slike



Filtar



Spektar filtrirane slike



Izlaz nakon filtriranja



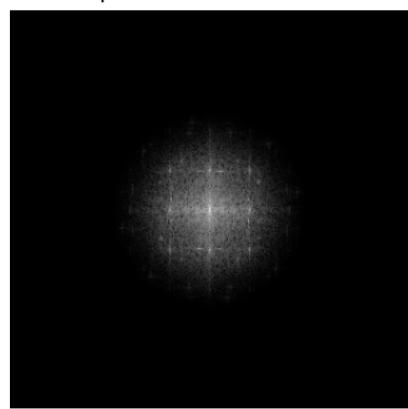
Izlaz nakon popravke kontrasta



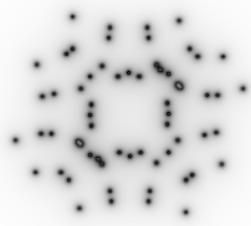
Ulazna slika



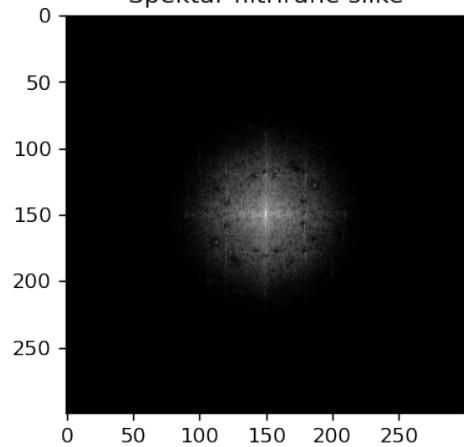
Spektar ulazne slike



Filtar



Spektar filtrirane slike



Izlaz nakon filtriranja



Rezultujuća slika:

```
[16]: img[:, :, 0] = filtR;  
      img[:, :, 1] = filtG;  
      img[:, :, 2] = filtB;
```

```
figure(figsize=(16,8))
imshow(img); plt.axis('off'); plt.title('Izlazna slika')
plt.show()
```

Izlazna slika



Nakon izvršenog filtriranja šum je dosta potisnut, slika postaje zamućena, ali je pokušano da se što više oštchine zadrži. I dalje postoji određen procenat degradacije na slici. Ako pokušamo da izoštrimo ivice to je moguće ali uz to da će i smetnje postati uočljivije.

```
[17]: img_yuv = color.rgb2yuv(img)
      img_y = img_yuv[:, :, 0]
```

```

lowpass_mask = np.ones(shape=(3,3))/9
img_y_blurred = ndimage.correlate(img_y, lowpass_mask)
img_y_details = img_y - img_y_blurred

img_y_sharp = img_y + 2*img_y_details

img_yuv_out = np.zeros(shape(img_yuv))
img_yuv_out[:, :, 1:3] = img_yuv[:, :, 1:3]
img_yuv_out[:, :, 0] = img_y_sharp

img_rgb_out = color.yuv2rgb(img_yuv_out)

img_rgb_out = np.clip(img_rgb_out, 0, 1)

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16,8), dpi=60)
ax = axes.ravel()

ax[0].imshow(img, vmin=0, vmax=1); ax[0].set_axis_off(); ax[0].
    set_title('Ulazna slika', fontsize=16)
ax[1].imshow(img_rgb_out, vmin=0, vmax=1); ax[1].set_axis_off(); ax[1].
    set_title('Isticanje visokih učestanosti', fontsize=16)

plt.tight_layout()

```



1.3 2.Zadatak

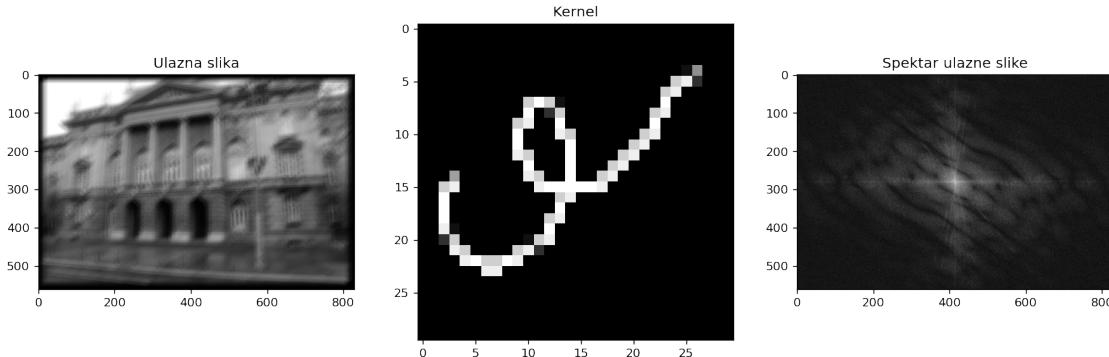
Potrebno je da popravimo kvalitet slike `etf_blur.tif`. Zamućenost je nastala usled pokreta kamere, a snimak pokreta kamere nalazi se na slici `kernel.tif`. Ideja je da pomocu kernela uradimo dekonvoluciju ulazne slike. Imamo degradiranu sliku, tj zamućenu sliku, i imamo kernel, odnosno u kernelu je predstavljena funkcija koja je degradirala sliku. Ako se pretpostavi da je originalna slika, kakvu zelimo da dobijemo, u konvoluciji sa zadatim kernelom, kao rezultat dala ulaznu sliku, onda se originalna slika dobija dekonvolucijom ulazne degradirane slike i zadatog kernela. Dekonvoluciju je najlakše da odradimo u frekvencijskom domenu s obzirom na to da ona u frekvencijskom domenu predstavlja deljenje, a u prostornom domenu bi ovo bio slojeniji proces. Dakle, uradićemo inverzno filtriranje, pri čemu ćemo koristiti Vinerov filter jer je on optimalan filter u smislu srednje kvadratne greške koja nastaje u invreznom filtriranju u prisustvu šuma.

$$\hat{F} = \left(\frac{|H|^2}{|H|^2 + k} \right) \frac{G}{H}$$

```
[6]: #ucitavanje ulazne slike i kernela
img1 = skimage.img_as_float(io.imread('../sekvence/etf_blur.tif'))
kernel = io.imread('../sekvence/kernel.tif')

H_motion = np.copy(kernel)
img_fft2 = fftshift(fft2(img1))

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(16,8), dpi=120)
ax = axes.ravel()
ax[0].imshow(img1, cmap='gray'); ax[0].set_title('Ulazna slika')
ax[1].imshow(abs(H_motion), cmap='gray'); ax[1].set_title('Kernel');
ax[2].imshow(log(1+abs(img_fft2)),cmap='gray'); ax[2].set_title('Spektar ulazne slike')
plt.show()
```



Kernel je veličine 30x30 piksela, tako da je potrebno da ga proširimo kako bismo mogli da ga primenimo na ulaznu sliku. To ćemo uraditi tako što proširimo veličinu kernela pa dodamo nule na polazni kernel. Potrebno je da spektar proširenog kernela ima isto informaciju kao i spektar

polazne slike.

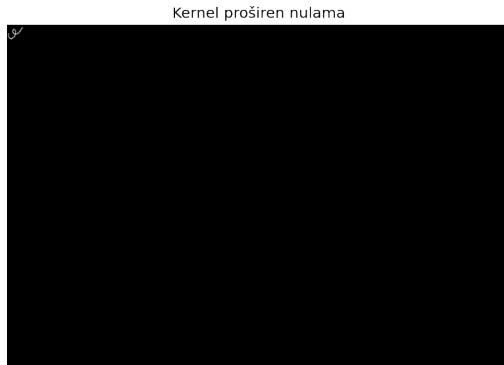
```
[25]: [height, width] = shape(kernel)
[h1,w1] = shape(img1)

img_p = zeros([h1, w1])

img_p[0:height, 0:width] = kernel
img_p_fft = fftshift(fft2(img_p))

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16,8), dpi=120)
ax = axes.ravel()

ax[0].imshow(img_p, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Kernel proširen nulama')
ax[1].imshow(log(1+abs(img_p_fft)), cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Magnituda spektra proširenog kernela');
```



Restauraciju će dalje biti izvršena Wienerovim filtrom.

```
[21]: #restauracija
img_fft2 = fftshift(fft2(img1))

H_motion = img_p_fft
k = 1e-4
W = (abs(H_motion)**2)/(abs(H_motion)**2 + k)
#W = (H_motion**2)/(H_motion**2+k)
img_fft_est = (img_fft2/H_motion)*W

#img_fft_est = img_fft2/H_motion
img_est = real(ifft2(ifftshift(img_fft_est)))
img_est = np.clip(img_est, 0, 1)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,16), dpi=120)
```

```

ax = axes.ravel()

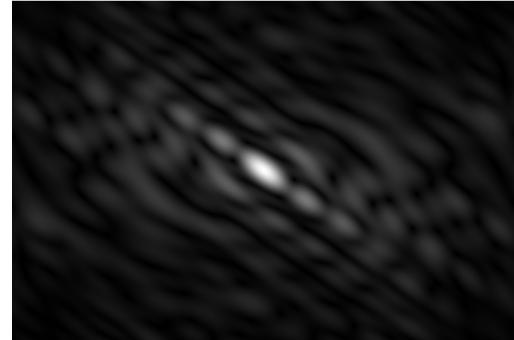
ax[0].imshow(img1, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Ulagana slika')
ax[1].imshow(abs(H_motion), cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Spektar degradacione funkcije')
ax[2].imshow(log(1+abs(img_fft_est)), cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('Spektar nakon Vinerovog filtra')
ax[3].imshow(img_est, cmap='gray'); ax[3].set_axis_off(); ax[3].set_title('Estimirani izlaz');

```

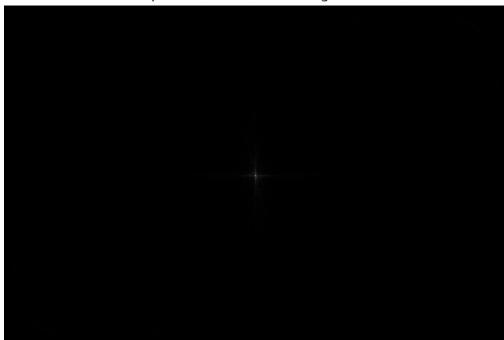
Ulagana slika



Spektar degradacione funkcije



Spektar nakon Vinerovog filtra



Estimirani izlaz



[24]: #ugradjena fja, ali unsupervised
$k = 1e-4$
#deconvolved= restoration.wiener(img1, kernel, k)
#fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 10),
#sharex=True, sharey=True)

```

# plt.gray()

#ax[0].imshow(img1)
#ax[0].axis('off')
#ax[0].set_title('Data')

#ax[1].imshow(deconvolved)
#ax[1].axis('off')
#ax[1].set_title('Self tuned restoration')
#fig.tight_layout()

#plt.show()

```

1.4 3.Zadatak

Potrebno je proceniti varijansu šuma na slici i potisnuti šum primenom adaptivnog filtriranja.

Slika lena_noise.tif sadrži određenu količinu Gausovog šuma nepoznate varijanse. Lokalna varijansa sa lokalnim susedstvom $(2r+1) \times (2r+1)$ predstavlja razliku srednje vrednosti kvadrata svih piksela u okviru susedstva i kvadrata srednje vrednosti ovih piksela. Korišćenjem histograma lokalnih varijansi odrediti varijansu šuma ako se smatra da većina piksela zapravo pripada uniformnim regionima slike.

Bitno je da većina piksela pripada uniformnim regionima slike, iz razloga što je histogram pre zašumljavanja samo jedan pik, i kada je šum dodat histogram se rasuje po Gausovoj raspodeli, pa dobijanjem histograma koji ima Gausovu raspodelu, ako znamo da je histogram pre zašumljavanja bio uniforman, možemo lako odrediti varijansu tog histograma, koja će zapravo biti varijansa šuma.

Prepostavimo da je srednja vrednost šuma jednaka nuli, jer u tom slučaju nije promenjen osvetljaj slike.

```
[26]: img3 = skimage.img_as_float(io.imread('../sekvence/lena_noise.tif'))
figure(figsize=(12,8));
imshow(img3,cmap='gray');
plt.axis('off');
plt.show()
```



Pošto se smatra da većina piksela pripada uniformnim regionima, ti regioni su pre zašumljenja imali neku uniformnu raspodelu. Nakon što je dodat šum nultog očekivanja i varijanse koju je potrebno da estimiramo, svaki region postaje uniformno raspodeljen, sa varijansom koja odgovara varijansi šuma, i očekivanjem koje je bilo očekivanje njegove prethodne raspodele. Posmatrajmo jedan region. Za svaki piksel izračuna se lokalna varijansa. Trebalо bi da je vrednost lokalne varijanse ista odnosno da su slične vrednosti, za sve piksele u tom uniformnom regionu.

```
[27]: r = range(1,21);

#fig, axes = plt.subplots(nrows=10, ncols=2, figsize=(8,4))
#ax = axes.ravel()

plt.figure(figsize=(8,4));
```

```

for r1 in range(1,21,4):
    filt_avg = ones([2*r1+1,2*r1+1])/((2*r1+1)**2)

    img_local_avg = ndimage.correlate(img3, filt_avg) #srednja vrednost nije u
    ↪ promenjena dodavanjem suma, to smo pretpostavili
    img_sqr_local_avg = ndimage.correlate(img3**2, filt_avg)

    img_local_var = img_sqr_local_avg - img_local_avg**2
    hist1, bins1 = np.histogram(img_local_var.flatten(),bins=256)

    #plt.figure(figsize=(8,4));
    #ax[r1-1].plot(bins1[1:], hist1/np.max(hist1), color='b', label = 'ulaz');
    #ax[r1-1].set_title('Histogram lokalnih varijansi, r=' + str(r1));

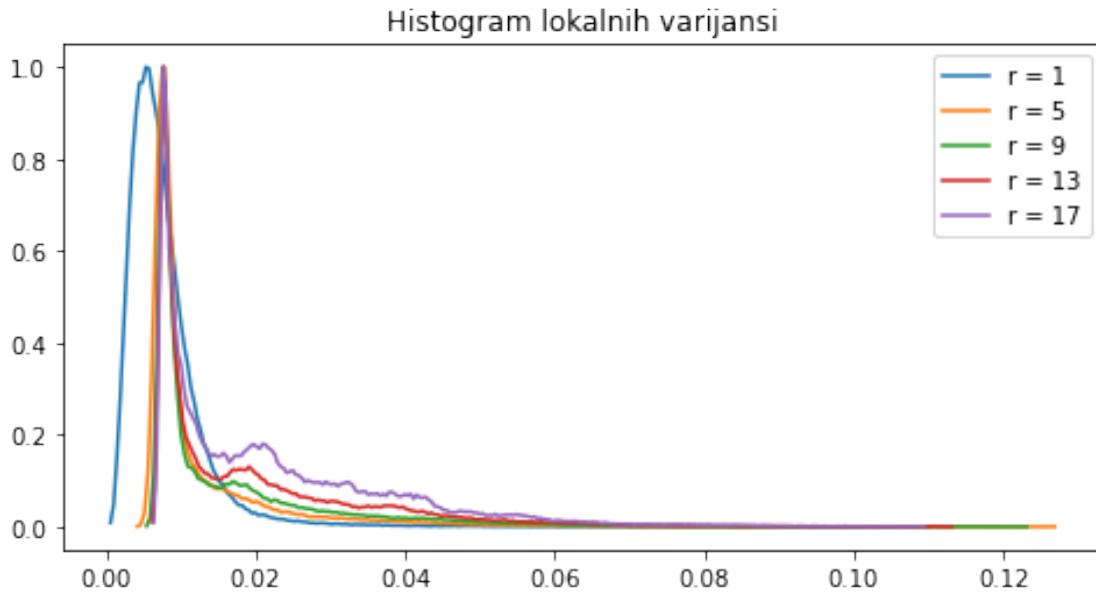
    plt.plot(bins1[1:], hist1/np.max(hist1), label = ('r = ' + str(r1)))

plt.legend(); plt.title('Histogram lokalnih varijansi');

#r1 = r[1];
#print(np.min(img_local_var))
#print(np.mean(hist1/np.max(hist1)))
#bins=bins1[1:];
#print(bins[np.abs((hist1-np.mean(hist1))/np.max(hist1))<0.005])
#v = bins[np.abs((hist1-np.mean(hist1))/np.max(hist1))<0.005]

#x = np.std(hist1/np.max(hist1))*np.sqrt(3.14/2)
#v = bins[np.abs(hist1/np.max(hist1)-x)<0.1]
#print(v)

```



Povećavanjem dimenzije lokalnog susedstva se narušava izgled raspodele lokalnih varijansi. Za $r=1$ raspodela podseća na Rayleigh-ijevu raspodelu, dok kako r raste, raspodela postaje uža i javljaju se visoke vrednosti na repu raspodele. Zato je najlakše za određivanje varijanse šuma posmatrati lokalne varijanse kada je $r=1$.

Zbog prethodno opisanih osobina koje važe za lokalne varijanse, zaključeno je da će se za varijansu šuma smatrati matematičko očekivanje lokalnih varijansi. Sa prikaza histograma lokalnih varijansi, primećuje se da raspodela varijansi ima oblik Rayleigh-jeve raspodele kada je standardna devijacija manja od 1, tako da ćemo zbog toga očekivanje potražiti kao matematičko očekivanje Rayleigh-jeve raspodele.

$$E[x] = \sigma \cdot \sqrt{\frac{\pi}{2}}$$

Takođe, za vrednosti r do 6 se dobija ista varijansa zaokružena na dve decimale. Ako se koristi mala veličina r , šum ne može da se potisne, što je prikazano za slučaj $r=1$. Za $r=3$ šum je dobro potisnut, ali za veće r slika se zamotiću, tačnije ivice se razlivaju.

```
[35]: r1 = 1;
filt_avg = ones([2*r1+1,2*r1+1])/((2*r1+1)**2)

img_local_avg = ndimage.correlate(img3, filt_avg) #srednja vrednost nije u
#→ promenjena dodavanjem suma, to smo pretpostavili
img_sqr_local_avg = ndimage.correlate(img3**2, filt_avg)
img_local_var = img_sqr_local_avg - img_local_avg**2
hist1, bins1 = np.histogram(img_local_var.flatten(), bins=256)
figure(figsize=(8,4))
plt.plot(bins1[1:], hist1/np.max(hist1), label = 'r = 1')
```

```

plt.legend(); plt.title('Histogram lokalnih varijansi');
x = np.std(hist1/np.max(hist1))*np.sqrt(3.14/2)

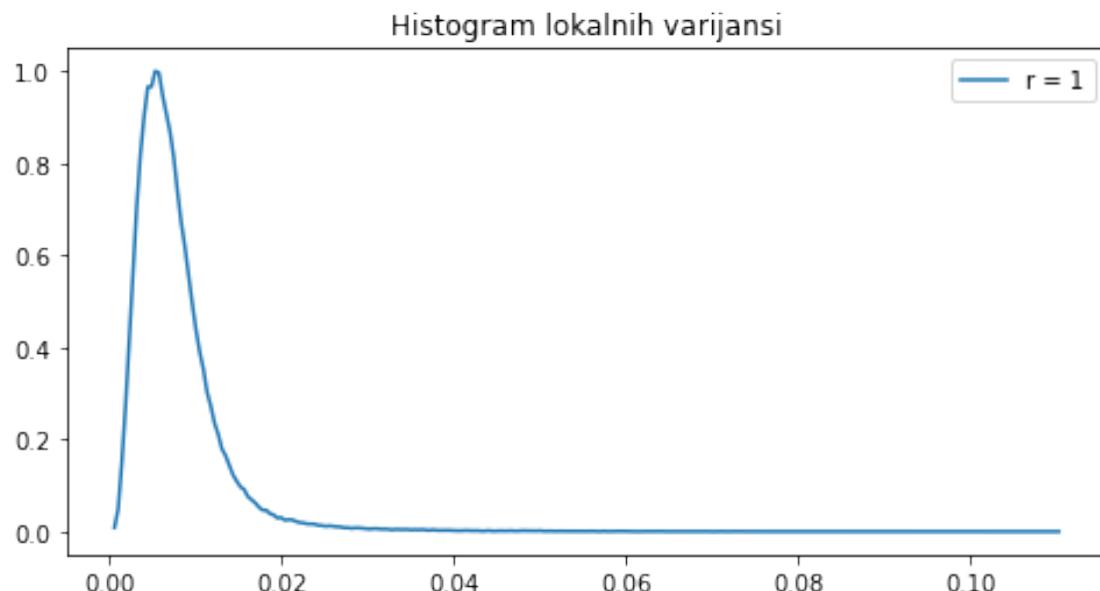
bins=bins1[1:];
v = bins[np.abs((hist1-np.mean(hist1))/np.max(hist1))<0.005]

x = np.std(hist1/np.max(hist1))*np.sqrt(3.14/2)
v = bins[np.abs(hist1/np.max(hist1)-x)<0.1]
print('Estimirana varijansa suma:')
print(np.mean(v))

```

Estimirana varijansa suma:

0.010881921305195937



```

[44]: r1 =1;
filt_avg = ones([2*r1+1,2*r1+1])/((2*r1+1)**2)

img_local_avg = ndimage.correlate(img3, filt_avg) #srednja vrednost nije u
#→ promenjena dodavanjem suma, to smo pretpostavili
img_sqr_local_avg = ndimage.correlate(img3**2, filt_avg)
img_local_var = img_sqr_local_avg - img_local_avg**2

est_var_noise = np.mean(v)
print(est_var_noise)
#est_var_noise = np.median(img_local_var.flatten());
#est_var_noise=np.mean(img_local_var.flatten());

```

```

weight = est_var_noise/img_local_var
weight[weight>1]=1

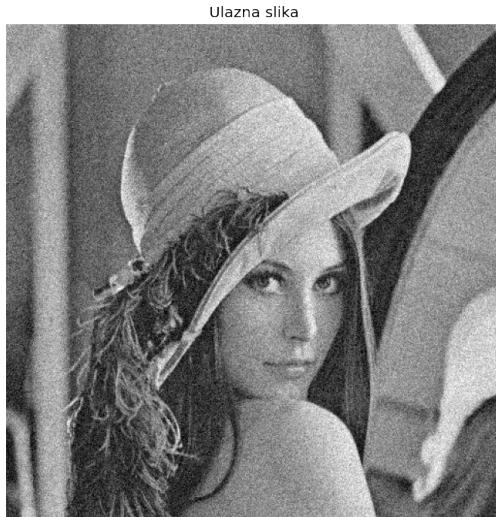
img_est = img3 + weight*(img_local_avg-img3)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,16), dpi=120)
ax = axes.ravel()

ax[0].imshow(img3, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Ulagana slika')
ax[1].imshow(img_local_avg, cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Usrednjena slika')
ax[2].imshow(weight, cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('Mapa težina')
ax[3].imshow(img_est, cmap='gray'); ax[3].set_axis_off(); ax[3].set_title('Izlaz nakon adaptivnog usrednjavanja');

```

0.010881921305195937



```
[45]: r1 =3;
filt_avg = ones([2*r1+1,2*r1+1])/((2*r1+1)**2)

img_local_avg = ndimage.correlate(img3, filt_avg) #srednja vrednost nije u
#promenjena dodavanjem suma, to smo prepostavili
img_sqr_local_avg = ndimage.correlate(img3**2, filt_avg)
img_local_var = img_sqr_local_avg - img_local_avg**2

est_var_noise = np.mean(v)
print(est_var_noise)
#est_var_noise = np.median(img_local_var.flatten());
```

```

#est_var_noise=np.mean(img_local_var.flatten());
weight = est_var_noise/img_local_var
weight[weight>1]=1

img_est = img3 + weight*(img_local_avg-img3)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,16), dpi=120)
ax = axes.ravel()

ax[0].imshow(img3, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Ulagana slike')
ax[1].imshow(img_local_avg, cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Usrednjena slika')
ax[2].imshow(weight, cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('Mapa težina')
ax[3].imshow(img_est, cmap='gray'); ax[3].set_axis_off(); ax[3].set_title('Izlaz nakon adaptivnog usrednjavanja');

```

0.010881921305195937



1.5 4.Zadatak

U ovom zadatku napisana je funkcija `dos_median` u kojoj je realizovana implementacija medijan filtra, kao i adaptivnog medijan fitlra.

1.5.1 Medijan filter

Medijan filter je tehniku nelinearnog digitalnog filtriranja i služi za uklanjanje impulsnog šuma. Ideja algoritma se zasniva na tome da su pikseli koji su pogodjeni impulsnim šumom zakucani na visoku ili nisku vrednost. Zato se u filtriranju, za svaki piksel posmatraju vrednosti lokalnog susedstva tog piksela. Pronađe se medijana tog lokalnog susedstva i potom vrednost trenutnog piksela postavi na vrednost medijane. Adaptivni medijan filter realizuje se na malo drugačiji način. On ima dve faze: prvu, u kojoj se za piksel nalazi minimálna veličina prozora potrebna

da potisne šum, a potom ukoliko vrednost piksela odgovara minimalnoj ili maksimalnoj vrednosti njegovog lokalnog susedstva, smatra se da je taj piksel pogoden šumom i tada se njegova vrednost postavlja na vrednost medijane. Performanse algoritma zavise od procenta slike koji je pogoden šumom, kao i od veličine prozora koja se koristi za lokalno susdestvo. Medijan filter ne može da potisne šum ukoliko je preveliki procenat slike pogoden šumom. Veličina lokalnog susedstva utiče na performanse tako da što se veće lokalno susedstvo posmatra bolje će šum biti uklonjen, ali će se ivice degradirati više. Zato je dobro pronaći minimalnu veličinu lokalnog susedstva koja potiskuje šum, ali koja neće degradirati ivice, što radi adaptivni medijan filter.

Pomoćne funkcije za sortiranje niza Napisane su pomoćne funkcije koje se pozivaju u funkciji dos_median, funkcija koja vraća medijanu niza, i funkcija sortiranja liste. Funkcija particija deli niz na particije i nalazi pivot i potrebna je zbog funkcije sortiranja. Za sortiranje je implementiran quicksort algoritam iz razloga što ima dobre prosečne performanse u odnosu na ostale algoritme sortiranja. Ispitivanjem je uočeno da insertion sort radi brže za male dimenzije lokalnog susedstva, ali dosta lošije od quicksorta za bilo koju veličinu prozora veću od 5.

```
[46]: #funkcija koja vraca medijanu niza
def medijana(arr):
    return arr[int(len(arr)/2)]

#funkcija koja nalazi pivot i deli listu na particije
def particija(arr,l,h):
    i = (l - 1)
    x = arr[h]

    for j in range(l , h):
        if arr[j] <= x:
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[h] = arr[h],arr[i+1]
    return (i+1)

#iterativni quicksort algoritam
def sortiranje(arr,l,h):
    #kreiranje steka
    size = h - l + 1
    stack = [0] * (size)

    #inicijalizacija
    top = -1
    top = top + 1
    stack[top] = l
    top = top + 1
    stack[top] = h

    #brisanje sa steka dok nije prazan
```

```

while top >= 0:
    #uzimanje low i high vrednosti
    h = stack[top]
    top = top - 1
    l = stack[top]
    top = top - 1
    #nalazenje pivota i njegove pozicije u sortiranom nizu
    p = particija( arr, l, h )
    # ako ima element sa leve strane pivota stavlja se na stek
    if p-1 > l:
        top = top + 1
        stack[top] = l
        top = top + 1
        stack[top] = p - 1

    # ako ima element sa desne strane pivota, stavlja se na stek
    if p+1 < h:
        top = top + 1
        stack[top] = p + 1
        top = top + 1
        stack[top] = h

```

dos_median funkcija

[47]: *#funkcije za nalazenje granica prozora lokalnog suedstva jer ako je piksel u blyu ivice slike prozor se drugacije posmatra*

```

prvaVel = lambda i,s,n: np.arange(0,i+int(s/2)+1) if i<int(s/2) else (np.
    ↪arange(i-int(s/2),n) if i>=(n-int(s/2)) else np.arange(i-int(s/2),i+int(s/
    ↪2)+1))
drugaVel = lambda j,s,m: np.arange(0,j+int(s/2)+1) if j<int(s/2) else (np.
    ↪arange(j-int(s/2),m) if j>=(m-int(s/2)) else np.arange(j-int(s/2),j+int(s/
    ↪2)+1))

```

[48]: **def dos_median(img_in,s_max,adaptive=False):**

`"""`

Median filter slike.

Parametri:

img_in(ndarray): ulazna slika zasumljena impulsnim sumom

s_max(int): maksimalna velicina prozora

adaptive(boolean): ako je adaptive=True radi se adaptivni median filter

Izlaz:

img_out(ndarray):filtrirana slika, tj. slika na kojoj je potisnut impulsni sum

`"""`

```

#provera validnosti prosledjenih argumenata
if img_in.dtype!='uint8' and img_in.dtype!='float64':
    print("Slika nije u dobrom formatu")
    return -1
if s_max%2==0:
    print("Velicina prozora mora biti neparna")
    return -1

n,m = np.shape(img_in)[0], np.shape(img_in)[1]
if adaptive==False:
    #medijan filter
    s = s_max
    #u slucaju obicnog medijan filtra, u
    ↵velicina prozora je s_max
    slika = np.copy(img_in)
    #pravi se kopija slike jer ne
    ↵treba da se radi nad ulaznom slikom
    img_out = np.zeros(np.shape(img_in))

    #uklanjanje impulsa piksel po piksel
    for i in range(0,n):
        for j in range(0,m):

            #lista listi koja sadrzi sve elemente lokalnog susedstva
            susedi = [x for x in slika[prvaVel(i,s,n),:][:,drugaVel(j,s,m)]]

            #pravljenje liste ciji je svaki element jedan piksel
            #analogno vektorizacija matrice
            sf = [item for sublist in susedi for item in sublist]

            #sortiranje
            #sorted(sf)
            sortiranje(sf,0,size(sf)-1)

            #postavljanje vrednosti piksela na medijanu lokalnog susedstva
            img_out[i][j] = medijana(sf)

else:
    #adaptivni medijan filter
    slika=np.copy(img_in)
    img_out = np.copy(img_in)

    #prolazak kroz sliku
    for i in range(0,n):
        for j in range(0,m):

            s=3
            while (s<=s_max):

```

```

#nalazenje minimalne velicine prozora za potiskivanje suma
susedi = [x for x in slika[prvaVel(i,s,n),:] [:,
→,drugaVel(j,s,m)]]]
sf = [item for sublist in susedi for item in sublist]
sortiranje(sf,0,size(sf)-1)

if sf[0]==medijana(sf) or sf[size(sf)-1]==medijana(sf):
    s=s+2
else:

    #otkljanjanje suma ako se smatra da je piksel zasumljen
    if slika[i][j]==sf[0] or slika[i][j]==sf[size(sf)-1]:
        img_out[i][j] = medijana(sf)
    s=s_max+1

return img_out

```

Ulazna i zašumljena slika

```

[49]: import time
from skimage import filters
img4 = skimage.img_as_ubyte(imread('lena.tif'))
proc = 0.2
img_noise4 = skimage.img_as_ubyte(util.random_noise(img4, mode='salt',
→amount=proc))
img_noise4 = skimage.img_as_ubyte(util.random_noise(img_noise4, mode='pepper',
→amount=proc))

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16,16), dpi=120)
ax = axes.ravel()

ax[0].imshow(img4, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Ulazna
→slika')
ax[1].imshow(img_noise4, cmap='gray'); ax[1].set_axis_off(); ax[1].
→set_title('Zasumljenost 0.2 procenta')
plt.show()

```



Primena dos_median funkcije na sliku

```
[20]: pr = 0.2
#zasumljavanje slike lena.tif
img_noise4 = skimage.img_as_ubyte(util.random_noise(img4, mode='salt',
    ↪amount=pr))
img_noise4 = skimage.img_as_ubyte(util.random_noise(img_noise4, mode='pepper',
    ↪amount=pr))

#medijan filter za prozor 3x3
izlaz1 = dos_median(img_noise4,3,adaptive=False)
izlaz11 = filters.median(img_noise4,selem=np.ones((3,3)))

#medijan fitlar za 5x5
izlaz2 = dos_median(img_noise4,5,adaptive=False)
izlaz21 = filters.median(img_noise4,selem=np.ones((5,5)))

#medijan filter za 7x7
izlaz3 = dos_median(img_noise4,7,adaptive=False)
izlaz31 = filters.median(img_noise4,selem=np.ones((7,7)))

#medijan filter za 11x11
izlaz4 = dos_median(img_noise4,11,adaptive=False)
izlaz41 = filters.median(img_noise4,selem=np.ones((11,11)))
```

```
[21]: pr = 0.4
#zasumljavanje slike lena.tif
img_noise4 = skimage.img_as_ubyte(util.random_noise(img4, mode='salt',
    ↪amount=pr))
```

```


    ↪amount=pr))

#medijan filter za prozor 3x3
izlaz04 = dos_median(img_noise4,3,adaptive=False)
izlaz041 = filters.median(img_noise4,selem=np.ones((3,3)))

#medijan fitlar za 5x5
izlaz042 = dos_median(img_noise4,5,adaptive=False)
izlaz043 = filters.median(img_noise4,selem=np.ones((5,5)))

#medijan filter za 7x7
izlaz044 = dos_median(img_noise4,7,adaptive=False)
izlaz045 = filters.median(img_noise4,selem=np.ones((7,7)))

#medijan filter za 11x11
izlaz046 = dos_median(img_noise4,11,adaptive=False)
izlaz047 = filters.median(img_noise4,selem=np.ones((11,11)))

```

[22]:

```

pr = 0.6
#zasumljavanje slike lena.tif
img_noise4 = skimage.img_as_ubyte(util.random_noise(img4, mode='salt',  

    ↪amount=pr))
img_noise4 = skimage.img_as_ubyte(util.random_noise(img_noise4, mode='pepper',  

    ↪amount=pr))

#medijan filter za prozor 3x3
izlaz061 = dos_median(img_noise4,3,adaptive=False)
izlaz062 = filters.median(img_noise4,selem=np.ones((3,3)))

#medijan fitlar za 5x5
izlaz063 = dos_median(img_noise4,5,adaptive=False)
izlaz064 = filters.median(img_noise4,selem=np.ones((5,5)))

#medijan filter za 7x7
izlaz065 = dos_median(img_noise4,7,adaptive=False)
izlaz066 = filters.median(img_noise4,selem=np.ones((7,7)))

#medijan filter za 11x11
izlaz067 = dos_median(img_noise4,11,adaptive=False)
izlaz068 = filters.median(img_noise4,selem=np.ones((11,11)))

```

[23]:

```

pr = 0.8
#zasumljavanje slike lena.tif
img_noise4 = skimage.img_as_ubyte(util.random_noise(img4, mode='salt',  

    ↪amount=pr))

```

```


    ↪amount=pr))

#medijan filter za prozor 3x3
izlaz08 = dos_median(img_noise4,3,adaptive=False)
izlaz081 = filters.median(img_noise4,selem=np.ones((3,3)))

#medijan fitlar za 5x5
izlaz082 = dos_median(img_noise4,5,adaptive=False)
izlaz083 = filters.median(img_noise4,selem=np.ones((5,5)))

#medijan filter za 7x7
izlaz084 = dos_median(img_noise4,7,adaptive=False)
izlaz085 = filters.median(img_noise4,selem=np.ones((7,7)))

#medijan filter za 11x11
izlaz086 = dos_median(img_noise4,11,adaptive=False)
izlaz087 = filters.median(img_noise4,selem=np.ones((11,11)))

```

[33]: #pričaz rezultata za 0.2

```

fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16,10))
ax = axes.ravel()

ax[0].imshow(izlaz1, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('3x3,  

    ↪proc=0.2')
ax[1].imshow(izlaz2, cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('5x5,  

    ↪proc=0.2')
ax[2].imshow(izlaz3, cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('7x7,  

    ↪proc=0.2')
ax[3].imshow(izlaz4, cmap='gray'); ax[3].set_axis_off(); ax[3].  

    ↪set_title('11x11, proc=0.2');
ax[4].imshow(izlaz11, cmap='gray'); ax[4].set_axis_off(); ax[4].  

    ↪set_title('ugradjena 3x3')
ax[5].imshow(izlaz21, cmap='gray'); ax[5].set_axis_off(); ax[5].  

    ↪set_title('ugradjena 5x5')
ax[6].imshow(izlaz31, cmap='gray'); ax[6].set_axis_off(); ax[6].  

    ↪set_title('ugradjena 7x7')
ax[7].imshow(izlaz41, cmap='gray'); ax[7].set_axis_off(); ax[7].  

    ↪set_title('ugradjena 11x11');

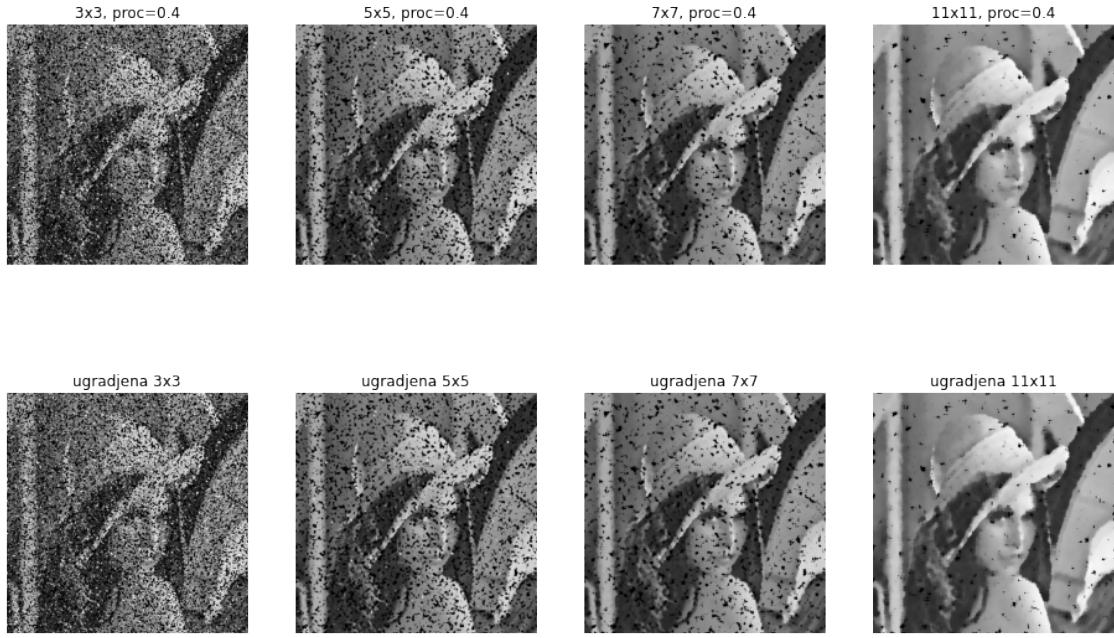
```



```
[34]: #Prikaz rezultata za 0.4
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16,10))
ax = axes.ravel()

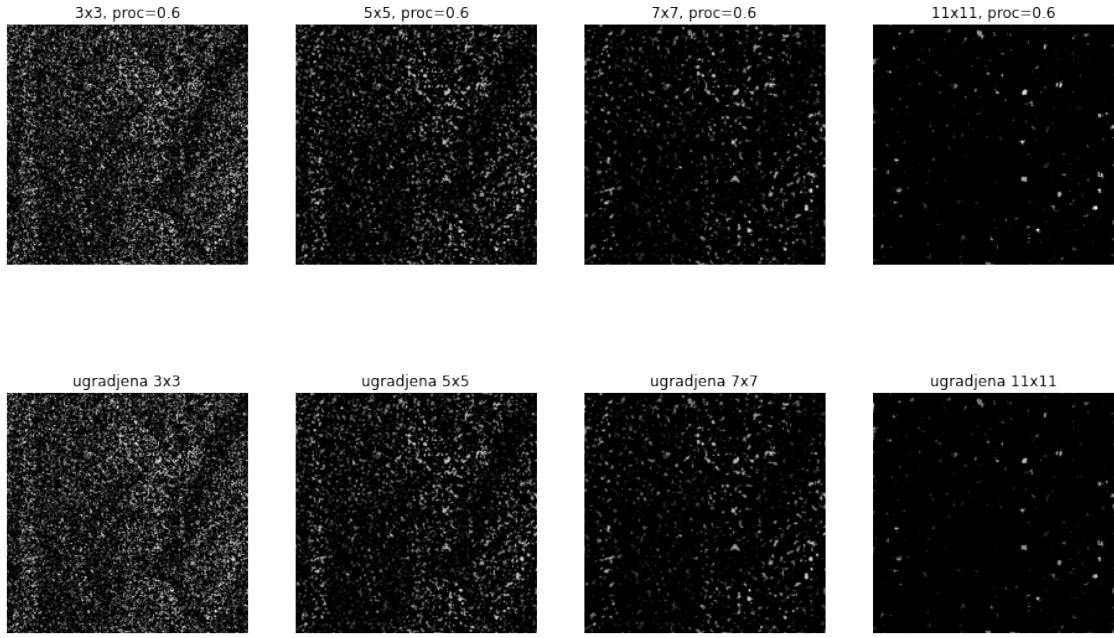
ax[0].imshow(izlaz04, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('3x3, proc=0.2')
ax[1].imshow(izlaz042, cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('5x5, proc=0.2')
ax[2].imshow(izlaz044, cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('7x7, proc=0.2')
ax[3].imshow(izlaz046, cmap='gray'); ax[3].set_axis_off(); ax[3].set_title('11x11, proc=0.2')

ax[4].imshow(izlaz041, cmap='gray'); ax[4].set_axis_off(); ax[4].set_title('ugradjena 3x3')
ax[5].imshow(izlaz043, cmap='gray'); ax[5].set_axis_off(); ax[5].set_title('ugradjena 5x5')
ax[6].imshow(izlaz045, cmap='gray'); ax[6].set_axis_off(); ax[6].set_title('ugradjena 7x7')
ax[7].imshow(izlaz047, cmap='gray'); ax[7].set_axis_off(); ax[7].set_title('ugradjena 11x11');
```



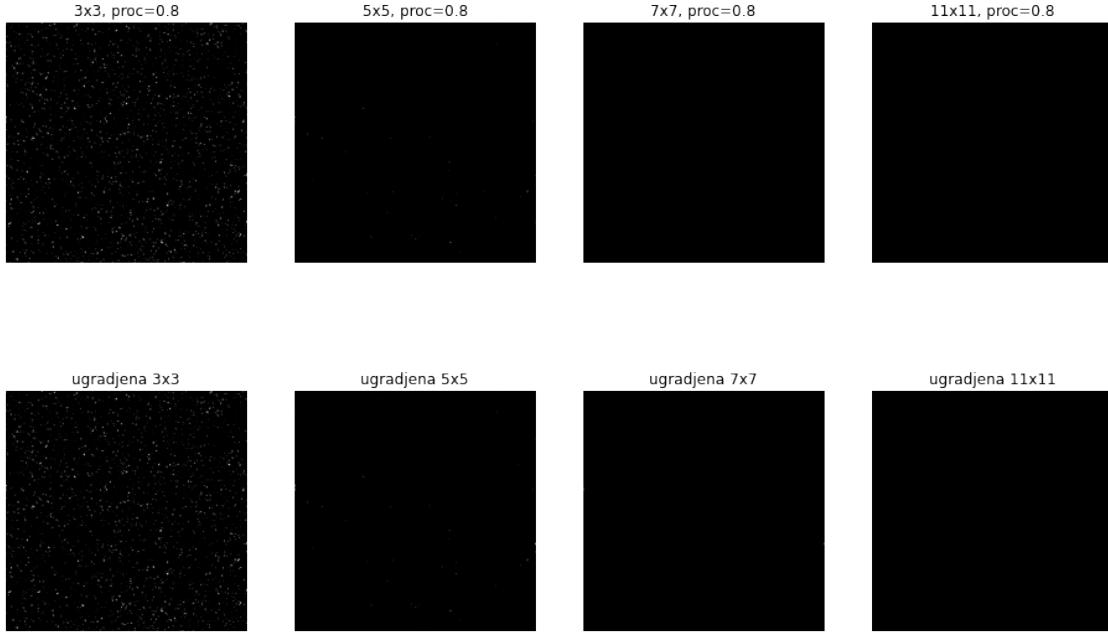
```
[35]: #Prikaz rezultata za 0.6
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16,10))
ax = axes.ravel()

ax[0].imshow(izlaz061, cmap='gray'); ax[0].set_axis_off(); ax[0].
    set_title('3x3, proc=0.6')
ax[1].imshow(izlaz063, cmap='gray'); ax[1].set_axis_off(); ax[1].
    set_title('5x5, proc=0.6')
ax[2].imshow(izlaz065, cmap='gray'); ax[2].set_axis_off(); ax[2].
    set_title('7x7, proc=0.6')
ax[3].imshow(izlaz067, cmap='gray'); ax[3].set_axis_off(); ax[3].
    set_title('11x11, proc=0.6');
ax[4].imshow(izlaz062, cmap='gray'); ax[4].set_axis_off(); ax[4].
    set_title('ugradjena 3x3')
ax[5].imshow(izlaz064, cmap='gray'); ax[5].set_axis_off(); ax[5].
    set_title('ugradjena 5x5')
ax[6].imshow(izlaz066, cmap='gray'); ax[6].set_axis_off(); ax[6].
    set_title('ugradjena 7x7')
ax[7].imshow(izlaz068, cmap='gray'); ax[7].set_axis_off(); ax[7].
    set_title('ugradjena 11x11');
```



```
[36]: #Prikaz rezultata za 0.8
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16,10))
ax = axes.ravel()

ax[0].imshow(izlaz08, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('3x3, proc=0.8')
ax[1].imshow(izlaz082, cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('5x5, proc=0.8')
ax[2].imshow(izlaz084, cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('7x7, proc=0.8')
ax[3].imshow(izlaz086, cmap='gray'); ax[3].set_axis_off(); ax[3].set_title('11x11, proc=0.8');
ax[4].imshow(izlaz081, cmap='gray'); ax[4].set_axis_off(); ax[4].set_title('ugradjena 3x3')
ax[5].imshow(izlaz083, cmap='gray'); ax[5].set_axis_off(); ax[5].set_title('ugradjena 5x5')
ax[6].imshow(izlaz085, cmap='gray'); ax[6].set_axis_off(); ax[6].set_title('ugradjena 7x7')
ax[7].imshow(izlaz087, cmap='gray'); ax[7].set_axis_off(); ax[7].set_title('ugradjena 11x11');
```



Diskusija rezultata medijan filtra Performanse algoritma zavise od procenta zašumljenosti slike i od veličine lokalnog susedstva koja se posmatra. Male veličine prozora neće uspeti dobro da potisnu šum sa slike, i što je veći procenat slike obuhvaćen šumom to će se loše performanse ispoljiti bolje. Ukoliko povećamo veličinu prozora, performanse u vidu potiskivanja šuma biće bolje. Međutim, loša strana medijan filtra je u tome što kako povećavamo širinu lokalnog susedstva koje se posmatra, to će se ivice zamalučivati, što se iz dobijenih rezultata uočava. Razlog za to je što se kod medijan filtra svaka vrednost piksela postavlja na vrednost medijane. Ovo je opravdano raditi ukoliko je region koji se posmatra uniforman, ali ukoliko je region neuinfozman, doći će do "razmazivanja" ivica što se veći region posmatra. Zbog toga nije dobro uvek menjati piksel medijanom lokalnog susedstva. Rezultati dos_median funkcije i ugrađenje funkcije za medijan filter su isti. Razlika je jedino u vremenu izvršavanja. Ugrađena funkcija je dosta brža, i približno je slično vreme izvršavanja ove funkcije za bilo koji procenat šuma i veličinu prozora, reda je veličine 10 sekundi. Vreme izvršavanja funkcije dos_median raste sa povećanjem veličine lokalnog susedstva, te se kreće od 10 sekundi pa do nekoliko minuta. Razlog za ovako velike promene u vremenu izvršavanja leži u funkciji sortiranja. Ukoliko se funkcija koju smo implementirali za sortiranje zameni ugrađenom funkcijom, vreme izvršavanja je dosta kraće. Za male prozore kao što su prozori širine 3x3 i 5x5, vreme izvršavanja je slično kao i sa implementiranim sortiranjem, dok je već za prozor širine 7x7 vreme izvršavanja sa ugrađenim sortom oko 18 sekundi, a vreme izvršavanja implementiranog sorta oko 70 sekundi. Takođe, posmatranjem pojedinačnih delova algoritma, utvrđeno je da korak sortiranja predstavlja "najskuplju" operaciju u smislu vremena.

Odrađeni su medijan filtri za prozore do 11x11. Iz rezultata se vidi da ako je prevelik procenat slike zahvaćen šumom da medijan filter ne može da potisne šum, već se dobija samo crna slika za 0.8 procenat šuma. Razlog za to je što je veliki broj piksela obuhvaćen šumom te ako se posmatra susedstvo medijana će biti jednaka vrednosti šuma odnosno ekstremnoj vrednosti. Možda bi ovo moglo da se posmatra logikom da sada medijan umesto da potisne šum, on potisne ispravne

vrednosti piksela i zato se dobije samo crna slika.

Zato ćemo posmatrati dalje rezultate adaptivnog medijan filtra. Zadavaćemo mu i veličinu prozora veću od 11 pošto običan medijan se dosta dugo izvršava za ovakve veličine, a adaptivni medijan će biti i efikasniji i brži.

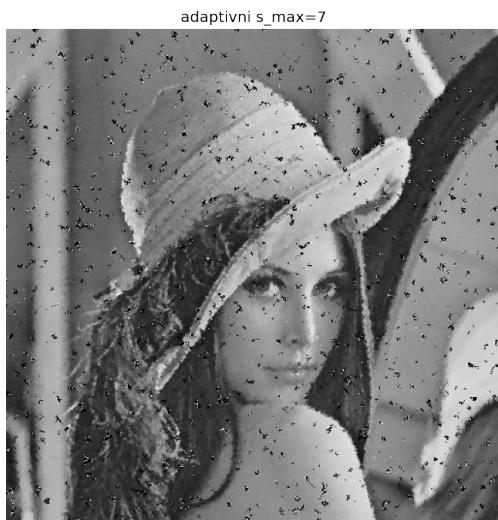
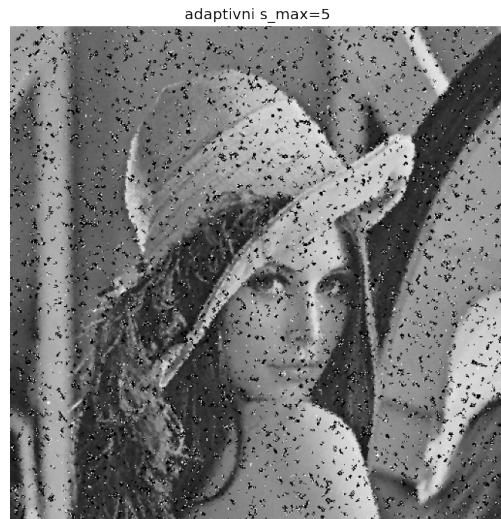
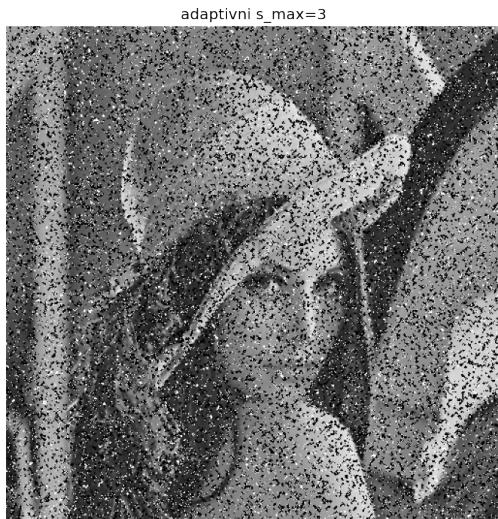
Adaptivni medijan filter

```
[37]: img_noise4 = skimage.img_as_ubyte(util.random_noise(img4, mode='salt', amount=0.4))
img_noise4 = skimage.img_as_ubyte(util.random_noise(img_noise4, mode='pepper', amount=0.4))

izl1 = dos_median(img_noise4,3,adaptive=True)
izl2 = dos_median(img_noise4,5,adaptive=True)
izl3 = dos_median(img_noise4,7,adaptive=True)
izl4 = dos_median(img_noise4,17,adaptive=True)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,16), dpi=120)
ax = axes.ravel()

ax[0].imshow(izl1, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('adaptivni s_max=3')
ax[1].imshow(izl2, cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('adaptivni s_max=5')
ax[2].imshow(izl3, cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('adaptivni s_max=7')
ax[3].imshow(izl4, cmap='gray'); ax[3].set_axis_off(); ax[3].set_title('adaptivni s_max=17');
```



Vreme izvršavanja Kada su diskutovani rezultati govoreno je malo i o vremenu izvršavanja implementiranog dos_median algoritma. U nastavku je još jednom uradjen median filter za velicine prozora 3,5,7 i 11, kako bi se prikazala zavisnost brzine izvršavanja od veličine lokalnog susedstva. Procenat zašumljenosti u ovom slučaju je 0.2.

```
[47]: x = np.shape(img4)[0]*np.shape(img4)[1]
execution_time_norm = np.zeros(5)
img_noise4 = skimage.img_as_ubyte(util.random_noise(img4, mode='salt', amount=0.2))
img_noise4 = skimage.img_as_ubyte(util.random_noise(img_noise4, mode='pepper', amount=0.2))
```

```

start=time.time()
izlaz1 = dos_median(img_noise4,3,adaptive=False)
end = time.time()
execution_time = (end - start)
execution_time_norm[0] = execution_time/x
print("Vreme izvrsavanja 3x3: " + str(round(execution_time,3))+ "s \n")

start=time.time()
izlaz2 = dos_median(img_noise4,5,adaptive=False)
end = time.time()
execution_time = (end - start)
execution_time_norm[1] = execution_time/x
print("Vreme izvrsavanja 5x5: " + str(round(execution_time,3))+ "s \n")

start=time.time()
izlaz3 = dos_median(img_noise4,7,adaptive=False)
end = time.time()
execution_time = (end - start)
execution_time_norm[2] = execution_time/x
print("Vreme izvrsavanja 7x7: " + str(round(execution_time,3))+ "s \n")

start=time.time()
izlaz4 = dos_median(img_noise4,9,adaptive=False)
end = time.time()
execution_time = (end - start)
execution_time_norm[3] = execution_time/x
print("Vreme izvrsavanja 9x9: " + str(round(execution_time,3))+ "s \n")

start=time.time()
izlaz4 = dos_median(img_noise4,11,adaptive=False)
end = time.time()
execution_time = (end - start)
execution_time_norm[4] = execution_time/x
print("Vreme izvrsavanja 11x11: " + str(round(execution_time,3))+ "s \n")

```

Vreme izvrsavanja 3x3: 12.188s

Vreme izvrsavanja 5x5: 34.064s

Vreme izvrsavanja 7x7: 53.803s

Vreme izvrsavanja 9x9: 97.393s

Vreme izvrsavanja 11x11: 169.976s

```
[50]: figure()
plt.plot([3,5,7,9,11],execution_time_norm)
plt.xlabel('veličina prozora')
plt.ylabel('normalizovano vreme')
```

```
[50]: Text(0, 0.5, 'normalizovano vreme')
```

