

domaci2_19_135

December 8, 2022

```
[1]: USE_WIDGETS = False
if USE_WIDGETS:
    %matplotlib widget
else:
    %matplotlib inline

import skimage
from pylab import *
import numpy as np
from skimage import exposure
```

0.1 Zadatak 1

Ulagana slika je nastala postupkom polutoniranja gde se štampanjem crnih i belih polja različite veličine ostvaruje utisak različitih nijansi sive. Potrebno je rekonstruisati sliku, tako da se dobije uniformna slika i da se što više potisnu tragovi tačkica.

Funkcija koja će se koristiti za generisanje maske, **Gausovog**, **Batervortovog** ili **idealnog** filtra.

```
[2]: # Funkcija koja vraca zeljeni filter.
def lpfilter(filt_type, Ny, Nx, sigma, n = 1):

    """Ulagni argumenti:

        filt_type :      Tipa string. Tip zeljenog filtra: gaussian, btw_
        ↪and ideal.

        Ny :           Tipa int, pozitivan broj. Broj vrsta filtra.

        Nx :           Tipa int, pozitivan broj. Broj kolona filtra.

        sigma :         Tipa float, pozitivan broj. Za gaussian filteru_
        ↪predstavlja
                           standardnu devijaciju. Za ideal i btw filteru_
        ↪predstavlja
                           opseg filtra.
```

```

    n:      Tipa int, pozitivan broj. Uzima se u obzir u
→ jedino ako je zeljeni
                                         tip filtra btw. Tada predstavlja nagib filtra. □
→ Sto je vece n to ce i nagib
                                         biti veci, tj. btw filter ce vise liciti na
→ ideal filter.

filter_mask:      Maska filtra specificirane velicine.
"""

# Provera da li je broj vrsta i kolona paran ili neparan.
# Potrebno je napraviti X i Y tako da budu simetricni oko 0.
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

# Simetricni X i Y oko 0.
X, Y = np.meshgrid(x, y)

# Rastojanje piksela od centralnog piksela [Y, X] = [0, 0]
D = (X**2 + Y**2)**0.5

filter_mask = np.zeros((Ny, Nx), dtype = np.double)

# U zavisnosti od zeljenog tipa filtra, pravi se maska.
if filt_type == 'gaussian':
    filter_mask = exp(-D**2 / (2 * sigma**2))
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

```

Funkcija koja će se koristiti za generisanje filtra koji se sastoji iz jednog, ili više, **notch** filtara. Tipovi filtara mogu biti Gausov, Batervortov ili idealni.

```
[3]: def cnotch(filt_type, notch, Ny, Nx, C, r, n=1):
    """Uzazni argumenti:
        filt_type : Tipa string. Tip zeljenog filtra: gaussian, btw
        and ideal.

        notch: Tipa string. Dozvoljene vrednosti su pass i
        reject. Specificira
            da li je filter propusnik ili nepropusnik.

        Ny : Tipa int, pozitivan broj. Broj vrsta filtra.

        Nx : Tipa int, pozitivan broj. Broj kolona filtra.

        C : Tipa list. Vrednosti liste su liste od dva
        pozitivna int
            broja. Predstavlja centre notch filtera.

        r : Tipa pozitivan float. Predstavlja propusni opseg
        filtera.

        n: Tipa int, pozitivan broj. Uzima se u obzir
        jedino ako je zeljeni
            tip filtra btw. Tada predstavlja nagib filtra.
        Sto je vece n to ce i nagib
            biti veci, tj. btw filter ce vise liciti na
        ideal filter.

        filter_mask: Maska filtra specificirane velicine. Sastavljen
        je od duplo veceg
            broja notch filtera od broja elemenata liste C.
        To je zbog toga
            sto su slike realni signali ciji su spektri
        centralno simetricni.

    """
    # Broj filtera, bez komplementarnih filtera.
    N_filters = len(C)

    # Inicijalizacija maske filtra.
    filter_mask = zeros([Ny,Nx])

    # Provera da li je broj vrsta i kolona paran ili neparan.
    # Potrebno je napraviti X i Y tako da budu simetricni oko 0.
    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

# Symetricni X i Y oko 0.
X, Y = meshgrid(x, y)

# Pravljenje N_filters notch filtera i
# toliko komplementarnih notch filtera.
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)

    # Udaljenost piksela od centra notch filtra.
    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)

```

```

# Udaljenost piksela od centra complementarnog notch filtra.
D0c = np.sqrt(np.square(X0c) + np.square(Y0c))

# U zavisnosti od zeljenog tipa filtra, pravi se maska.
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

# Da li zelimo propusnik ili nepropusnik.
if notch == 'pass':
    return filter_mask
elif notch == 'reject':
    return 1 - filter_mask
else:
    return

```

Funkcija koja određuje maksimalne vrednosti u spektru. Prva vrednost koju funkcija prikazuje predstavlja **DC** vrednost, ostali maksimumu pripadaju ili kopijama ili okolini DC vrednosti.

```

[ ]: import scipy.signal as signal

img = skimage.img_as_float(imread('sekvence/girl_ht.tif'))
[Ny, Nx] = shape(img)
img_fft = fftshift(fft2(img))

# svi lokalni maksimumi
peaksPos = signal.argrelextrema(img_fft.flatten(), np.greater)
peaksPos = peaksPos[0]
# indeksi lokalnih maksimuma koji su najveći
# znak "-" stoji da bi se dobili indeksi u opadajućem redosledu - prvi u nizu → odgovaraju maksimalnim vrednostima
maxPeaksPos = np.argsort(-img_fft.flatten())[peaksPos]
# indeksi maksimalnih vrednosti u originalnom nizu u opadajućem redosledu
maxPeakIndex = peaksPos[maxPeaksPos]

maxPeakIndex_matrix = []

```

```

for i in range(50):
    maxPeakIndex_matrix.append(np.unravel_index(maxPeakIndex[i], img.shape))

print(maxPeakIndex_matrix)

```

Ulagana slika je nastala postupkom polutoniranja gde se štampanjem crnih i belih polja različite veličine ostvaruje utisak različitih nijansi sive. To je razlog zašto se na slici vide tačkice. Ukoliko se pokaže spektar ulazne slike, čini se kao da se spektar ponavlja.

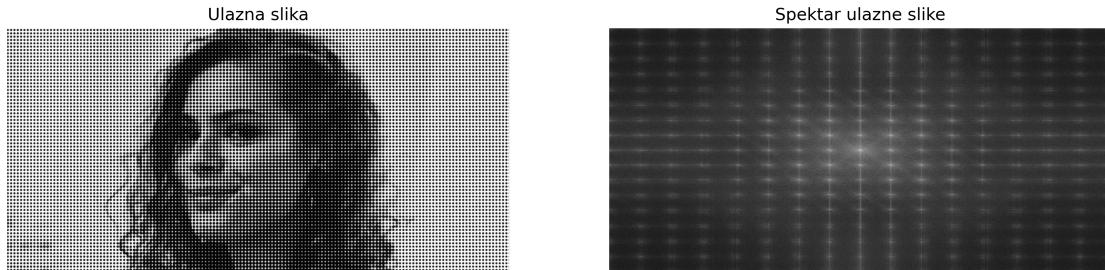
```

[4]: # Ucitavanje slike u float formatu
img = skimage.img_as_float(imread('sekvence/girl_ht.tif'))
# Furijeova transformacija ulazne slike
img_fft = fftshift(fft2(img))

[Ny, Nx] = shape(img)

# IsCRTavanje slika i njihovih frekvencijskih karakteristika.
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14,12), dpi=300)
ax = axes.ravel()
ax[0].imshow(img, 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');

```



Snaga spektralnih komponenti značajno opada sa rastojanjem od DC vrednosti. Najbitnije spektralne komponente se nalaze tamo gde je najveća snaga, tj. oko DC vrednosti. Prirodne slike nemaju veoma visoke frekvencije.

Da bismo uklonili tačke, mogli bismo da primenimo **NF** filter. Prirodno se postavlja da bi najbolji filter bio idealni filter, ali on ne daje najbolje rezultate. Kako je idealni filter *sinc* funkcija u prostornom domenu i kako je maska filtra konačne dužine, on će prouzrokovati **ripple**. Umesto njega bismo mogli da koristimo **Gausov** filter, on rešava taj problem tako što nagib njegove funkcije nije beskonačan, već ima konačnu vrednost. Zbog toga je potrebno povećati propusni opseg, u odnosu na idealni filter. Bolji filter je **Batervortov** filter, pošto nam on daje jedan stepen slobode da biramo koliki nagib želimo. Što je n veće to Batervortov filter više liči na idealni filter. Što je n manje, to je potreban veći propusni opseg. $D(u, v)$ predstavlja udaljenost piksela od centralnog piksela, D_0 predstavlja propusni opseg.

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

Da bismo se rešili tačkica na ovaj način, moramo propustiti samo originalni spektar, sa što manje doprinosa kopija. Na taj način uklanjamo visoke frekvencije sa slike, pa je slika mutna. Na primeru ispod se može videti da se tačkice ne vide, ali je slika mutna.

```
[5]: # Niskopropusni filter
lp_filter = lpfilter('btw', Ny, Nx, 100, 5)

# Filtrirana ulazna slika. Konvolucija
# u prostornom domenu, proizvod u frekvencijskom.
img_fft_filt_lp = img_fft*lp_filter

# IsCRTavanje slika i njihovih frekvencijskih karakteristika.
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(14,12), dpi=300)
ax = axes.ravel()
ax[0].imshow(real(fftshift(img_fft_filt_lp))), cmap='gray'; ax[0].set_axis_off(); ax[0].set_title('Filtrirana slika')
ax[1].imshow(log(1 + abs(img_fft_filt_lp)), cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Spektar filtrirane slike')
ax[2].imshow(log(1 + abs(lp_filter)), cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('Spektar filtra');
```



Problem mutne slike bismo mogli da smanjimo ukoliko pored originalnog spektra propustimo i neke kopije, ali te kopije moramo nekako da potisnemo. Potiskivanje kopija možemo uraditi **notch** filtrom, takođe Batervortovog tipa, jer nam on daje jedan stepen slobode više.

Na primeru ispod, primjenjen je NF filter većeg propusnog opsega, i 4 notch filtra Batervortovog tipa. Dva notch filtra su manjih nepropusnih opsega, jer oni poništavaju delovanje kopija na vertikalnoj osi koje su bliže DC vrednosti. Druga dva notch filtra su većeg nepropusnog opsega i one poništavaju dve kopije na horizontalnoj osi koje su udaljenije od DC vrednosti.

```
[6]: # Niskopropusni filter
lp_filter = lpfilter('btw', Ny, Nx, 70, 5)

# Filtrirana ulazna slika. Konvolucija
# u prostornom domenu, proizvod u frekvencijskom.
img_fft_filt_lp = img_fft*lp_filter
```

```

# Definisanje centara notch filtera.
C1 = [[527, 1240]]
C2 = [[595, 1090]]
# Notch filtri razlicitih velicina.
nr_filter1 = cnotch('btw', 'reject', Ny, Nx, C1, 40, 5)
nr_filter2 = cnotch('btw', 'reject', Ny, Nx, C2, 80, 5)
# Objedinjeni notch filtri u jedan filter.
nr_filter = nr_filter1*nr_filter2

# Filtrirana ulazna slika. Konvolucija
# u prostornom domenu, proizvod u frekvencijskom.
img_fft_filt_nr = img_fft_filt_lp*nr_filter

# Inverzna Furijeova transformacija
img_filt = real(ifft2(ifftshift(img_fft_filt_nr)))

# Isrtavanje slika i njihovih frekvencijskih karakteristika.
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(14,12), dpi=300)
ax = axes.ravel()
ax[0].imshow(real(ifftshift(img_fft_filt_nr)), cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Filtrirana slika')
ax[1].imshow(log(1 + abs(img_fft_filt_nr)), cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Spektar filtrirane slike');
ax[2].imshow(log(1 + abs(nr_filter)), cmap='gray'); ax[2].set_axis_off(); ax[2].set_title('Spektar filtra');

```



Kvalitet izlazne slike možemo još malo popraviti izoštravanjem i poboljšanjem kontrasta. Treba biti vrlo pažljiv prilikom izoštravanja. S obzirom da se u slici nalaze neželjeni detelji, koji su ostali i nakon filtriranja, koji se mogu dodatno naznačiti. Izoštravanje, u takvoj meri da se ne naglase neželjeni detalji, u ovom slučaju ne doprinosi primetnom poboljšanju kvaliteta, tako da se može izostaviti.

```

[7]: from scipy import ndimage
from skimage import exposure

def sharpening_img(img, filtered):
    laplacian_mask = np.array([
        [1, 1, 1],

```

```

[1, -8, 1],
[1, 1, 1] ])

if (filtered):
    img_filtered = skimage.filters.gaussian(img, sigma=1, truncate=3)      # Uklanjanje visokih ucestanosti, tj. suma
    ↪Uklanjanje visokih ucestanosti, tj. suma
else:
    img_filtered = img

img_lap = ndimage.correlate(img_filtered, laplacian_mask)
img_sharp = img_filtered - 3*img_lap                                # Mnozenje konstantom
    ↪3, radi veceg naglasavanja detalja

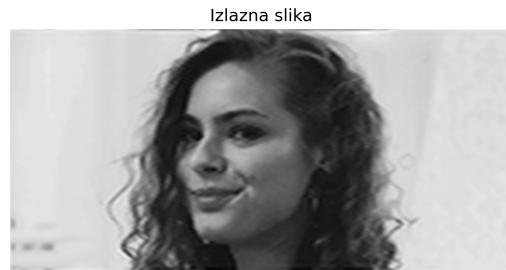
img_sharp = np.clip(img_sharp, 0, 1)

return img_sharp

img_out = exposure.rescale_intensity(sharpening_img(img_filt, True))

# IsCRTavanje slika.
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14,12), dpi=300)
ax = axes.ravel()
ax[0].imshow(img, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Ulazna slika')
    ↪slika')
ax[1].imshow(img_out, cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Izlazna slika');
    ↪set_title('Izlazna slika');

```



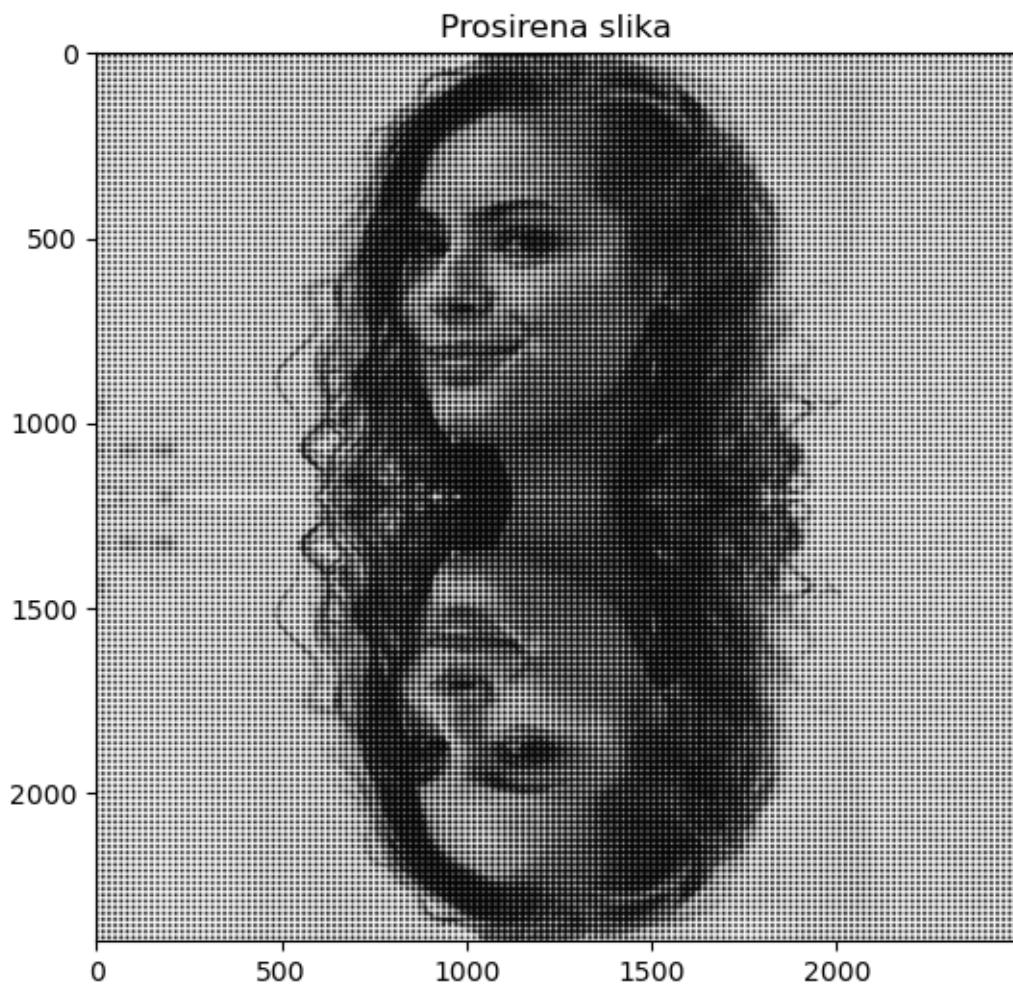
Iznad glave u izlaznoj slici se može videti tamna linija. Ta linija je posledica **DFT-a i cirkularne konvolucije**. Kako je slika konačna, ona ne može biti periodičan signal, prema tome za prelaz i frekvencijski domen, mora se koristiti Furijeova transformacija diskretnih signala. Furijeova transformacija daje kontinualnu funkciju učestanosti. Kako računari ne mogu skladištitи beskonačan broj podataka, Furijeova transformacija se mora diskretizovati. Diskretizacija Furijeove transformacije podrazumeva da je ulazni signal periodičan tako da ulazni signal odgovara jednom periodu.

Da bismo otklonili tu tamnu liniju, trebamo prošitriti ulaznu sliku na određeni način.

```
[8]: # Ucitavanje slike u float formatu
img = skimage.img_as_float(imread('sekvence/girl_ht.tif'))

[Ny, Nx] = img.shape
img_new = np.zeros((2*Ny, Nx))
img_new[:Ny, :] = img
img_new[Ny:, :] = img[:, :-1]

fig, axes = plt.subplots(figsize = [8, 6])
plt.imshow(img_new, cmap='gray'); plt.title('Prosirena slika');
```



Izlazna slika je sa ovom modifikacijom identična kao u prvom slučaju, jedina razlika je na gornjoj ivici. Kako je na bočnim ivicama slika uniformna, dovoljno je sliku proširiti samo po jednoj koordinati, da nije bila uniformna, tamne linije bi se i na tim ivicama pojavile. U tom slučaju bi slika morala drugačije da se proširi.

S obzirom da u ovom slučaju, razlika nije velika, a na ovaj način, proširenjem slike, povećavamo

memorijske i računske kapacitete, nema potrebe vršiti ovaku transformaciju.

```
[9]: # Furijeova transformacija ulazne slike
img_fft = fftshift(fft2(img_new))
[Ny, Nx] = img_new.shape

# Niskopropusni filter
lp_filter = lpfilter('btw', Ny, Nx, 70, 5)

# Filtrirana ulazna slika. Konvolucija
# u prostornom domenu, proizvod u frekvencijskom.
img_fft_filt_lp = img_fft*lp_filter

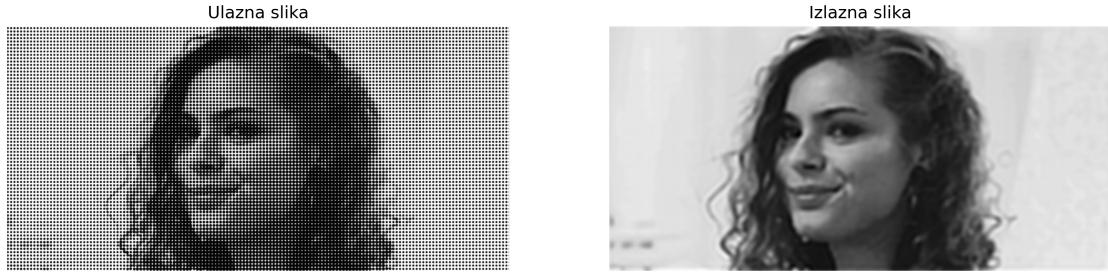
# Definisanje centara notch filtera.
C1 = [[527, 1240]]
C2 = [[595, 1090]]
# Notch filtri razlicitih velicina.
nr_filter1 = cnotch('btw', 'reject', Ny, Nx, C1, 40, 5)
nr_filter2 = cnotch('btw', 'reject', Ny, Nx, C2, 80, 5)
# Objedinjeni notch filtri u jedan filter.
nr_filter = nr_filter1*nr_filter2

# Filtrirana ulazna slika. Konvolucija
# u prostornom domenu, proizvod u frekvencijskom.
img_fft_filt_nr = img_fft_filt_lp*nr_filter

# Inverzna Furijeova transformacija
img_pad_filt = real(ifft2(ifftshift(img_fft_filt_nr)))
img_filt = img_pad_filt[0:img.shape[0], 0:img.shape[1]]

img_out = exposure.rescale_intensity(sharpening_img(img_filt, True))

# IsCRTavanje slika.
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14,12), dpi=300)
ax = axes.ravel()
ax[0].imshow(img, cmap='gray'); ax[0].set_axis_off(); ax[0].set_title('Ulazna slika')
ax[1].imshow(img_out, cmap='gray'); ax[1].set_axis_off(); ax[1].set_title('Izlazna slika');
```



0.2 Zadatak 2

Funkcija koja prima **RGB** sliku i vrši jednako inverzno filtriranje svake komponente. Možemo izabrati dodatno filtriranje **Vinerovim** filtrom ili **LP** filtrom.

```
[10]: def filterRGBImg(img, Ny, Nx, sigma, filtertype = "non"):
    # Izdvajanje R, G i B komponenti.
    img_r = img[:, :, 0]
    img_g = img[:, :, 1]
    img_b = img[:, :, 2]

    # Furijeova transformacija.
    img_fft_r = fftshift(fft2(img_r))
    img_fft_g = fftshift(fft2(img_g))
    img_fft_b = fftshift(fft2(img_b))

    # Degradaciona funkcija.
    H = lpfilter('gaussian', Ny, Nx, sigma)

    # Provera i odabir odgovarajućeg filtra.
    # U slučaju da je korisnik uneo pogresan tip, prijavljuje se greska.
    if filtertype == "wiener":
        k = 10**(-4)
        W = (abs(H)**2) / (abs(H)**2 + k)
    elif filtertype == "lowpass":
        W = lpfilter('ideal', Ny, Nx, 100)
    elif filtertype == "non":
        W = 1
    else:
        raise ValueError("filtertype must be \"wiener\", \"lowpass\" or \"non\"!")
    ↵

    # Množenje u frekvencijskom domenu, konvolucija u prostornom.
    img_fft_est_r = img_fft_r / H * W
    img_fft_est_g = img_fft_g / H * W
    img_fft_est_b = img_fft_b / H * W
```

```

# Inverzna Furijeova transformacija.
# Mogu se pojaviti neki mali imaginarni clanovi zbog zaokruzivanja.
img_est_r = real(fftshift(ifft2(img_fft_est_r)))
img_est_g = real(fftshift(ifft2(img_fft_est_g)))
img_est_b = real(fftshift(ifft2(img_fft_est_b)))

# Moze se probiti opseg.
img_est_r = np.clip(img_est_r, 0, 1)
img_est_g = np.clip(img_est_g, 0, 1)
img_est_b = np.clip(img_est_b, 0, 1)

img_out = np.zeros(img.shape)
img_out[:, :, 0] = img_est_r
img_out[:, :, 1] = img_est_g
img_out[:, :, 2] = img_est_b

return img_out

```

Potrebito je izvrsiti restauraciju ulazne slike. Ulazna slika je nastala tako sto je originalna slika degradirana funkcijom $H(u, v)$. Poznato nam je da je degradaciona funkcija Gausova funkcija, ali nam je nepoznata standardna devijacija te funkcije.

$$G(u, v) = F(u, v)H(u, v)$$

$$F(u, v) = \frac{G(u, v)}{H(u, v)}$$

Ovo daje dobre rezultate pod pretpostavkom da ulazna slika, pored degradacije, nema nimalo šuma. Kako to nije ispunjeno u ovom slučaju, rezultati ovakvog postupka neće biti očekivani.

$$G(u, v) = F(u, v)H(u, v) + N(u, v)$$

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)} = F(u, v) + \frac{N(u, v)}{H(u, v)}$$

Za male vrednosti degradacione funkcije, koliko god da je mala vrednost šuma, izraz će eksplodirati. Izlazna slika neće uopšte ličiti na sliku.

```
[11]: # Ucitavanje slike u float formatu
img = skimage.img_as_float(imread('sekvence/road_blur.png'))
img = img[:, :, :3]
[Ny, Nx] = img.shape[:2]

imgEst = filterRGBImg(img, Ny, Nx, sigma = 60, filtertype = "non")

img_fft_r = fftshift(fft2(img[:, :, 0]))
imgEst_fft_r = fftshift(fft2(imgEst[:, :, 0]))
```

```
# IsCRTavanje slike.
fig, axes = plt.subplots(nrows = 2, ncols = 2, figsize=(14, 12), dpi = 300)
ax = axes.ravel()
ax[0].imshow(img); ax[0].set_axis_off(); ax[0].set_title('Ulazna slika')
ax[1].imshow(imgEst); ax[1].set_axis_off(); ax[1].set_title('Izlazna slika');
ax[2].imshow(log(1 + abs(img_fft_r)), cmap = "gray"); ax[2].set_axis_off(); ↳
    ↳ ax[2].set_title('Spektar R komponente ulazne slike')
ax[3].imshow(log(1 + abs(imgEst_fft_r)), cmap = "gray"); ax[3].set_axis_off(); ↳
    ↳ ax[3].set_title('Spektar R komponente izlazne slike');
```



0.2.1 Ograničenje opsega

S obzirom da slika inverznim filtriranjem dobija značajne komponente na visokim učestanostima, možemo ih ukloniti Low Pass filtrom. Iz par pokušaja se dobija da se zadovoljavajući rezultati dobijaju primenom idealnog LP filtra propisnim opsegom 100.

```
[12]: # Ucitavanje slike u float formatu
img = skimage.img_as_float(imread('sekvence/road_blur.png'))
img = img[:, :, :3]
[Ny, Nx] = img.shape[:2]

imgEst = filterRGBImg(img, Ny, Nx, sigma = 60, filtertype = "non")
```

```


```

imgEst_filt = filterRGBImg(img, Ny, Nx, sigma = 60, filtertype = "lowpass")

imgEst_fft_r = fftshift(fft2(imgEst[:, :, 0]))
imgEst_filt_fft_r = fftshift(fft2(imgEst_filt[:, :, 0]))

IsCRTavanje slike.
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14,12), dpi=300)
ax = axes.ravel()
ax[0].imshow(imgEst); ax[0].set_axis_off(); ax[0].set_title('Nefiltrirana
slika')
ax[1].imshow(imgEst_filt); ax[1].set_axis_off(); ax[1].set_title('Filtrirana
slika');
ax[2].imshow(log(1 + abs(imgEst_fft_r)), cmap = "gray"); ax[2].set_axis_off();
ax[2].set_title('Spektar R komponente nefiltrirane slike')
ax[3].imshow(log(1 + abs(imgEst_filt_fft_r)), cmap = "gray"); ax[3].
set_axis_off(); ax[3].set_title('Spektar R komponente filtrirane slike');

```


```



0.2.2 Vinerov filter

Vinerov filter potiskuje one elemente kod kojih je $H \approx 0$ dok elemente za koje je H dovoljno veće od K ne potiskuju. Faktor potiskivanja određujemo parametrom K .

$$W(u, v) = \frac{|H(u, v)|^2}{|H(u, v)|^2 + K}$$

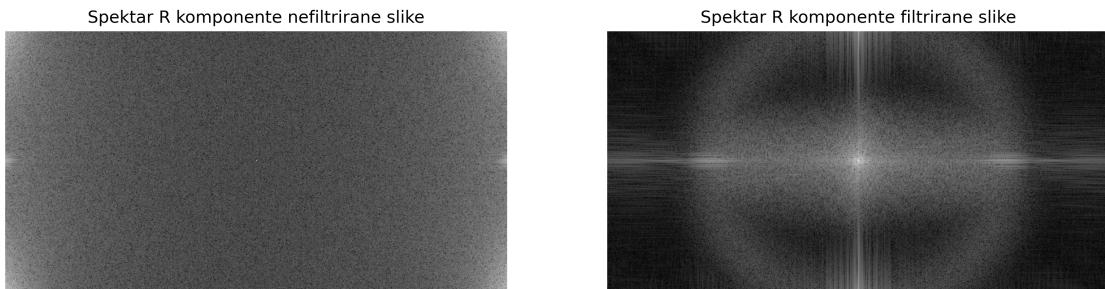
$$F(u, v) = \frac{G(u, v)}{H(u, v)} W(u, v)$$

```
[23]: # Ucitavanje slike u float formatu
img = skimage.img_as_float(imread('sekvence/road.blur.png'))
img = img[:, :, :3]
[Ny, Nx] = img.shape[:2]

imgEst = filterRGBImg(img, Ny, Nx, sigma = 60, filtertype = "non")
imgEst_filt = filterRGBImg(img, Ny, Nx, sigma = 60, filtertype = "wiener")

imgEst_fft_r = fftshift(fft2(imgEst[:, :, 0]))
imgEst_filt_fft_r = fftshift(fft2(imgEst_filt[:, :, 0]))

# IsCRTavanje slika.
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14,12), dpi=300)
ax = axes.ravel()
ax[0].imshow(imgEst); ax[0].set_axis_off(); ax[0].set_title('Nefiltrirana slika')
ax[1].imshow(imgEst_filt); ax[1].set_axis_off(); ax[1].set_title('Filtrirana slika');
ax[2].imshow(log(1 + abs(imgEst_fft_r)), cmap = "gray"); ax[2].set_axis_off();
ax[2].set_title('Spektar R komponente nefiltrirane slike')
ax[3].imshow(log(1 + abs(imgEst_filt_fft_r)), cmap = "gray"); ax[3].set_axis_off();
ax[3].set_title('Spektar R komponente filtrirane slike');
```



Izlaznoj slici možemo dodatno promeniti kontrast, kako bi se tablice su AWCA – 510. Slika je izoštrena, ali nije jednaka originalu zbog prisustva šuma, koliko god mali on bio.

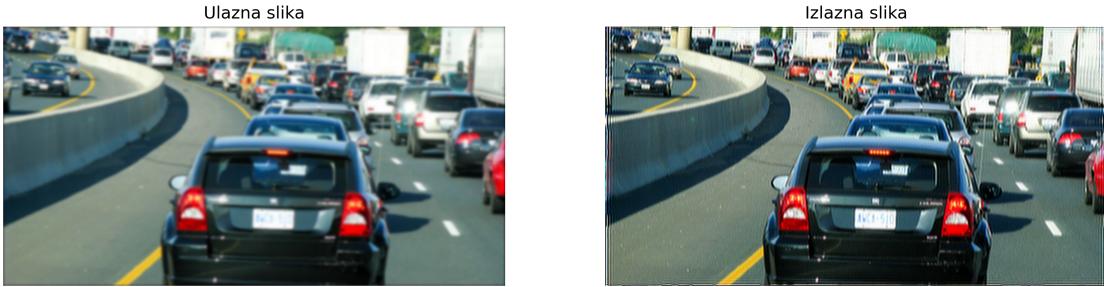
```
[29]: # Stepena funkcija, gde je k iz opsega (0, 1)
def pow_transform(img, k):

    img_out = abs(img)**k
    img_out = skimage.img_as_ubyte(img_out);      # Transformacija podatka iz
    ↵float64 u uint8

    return img_out;

img_out = pow_transform(imgEst_filt, 1.15)

# Isrtavanje slika.
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14,12), dpi=300)
ax = axes.ravel()
ax[0].imshow(img); ax[0].set_axis_off(); ax[0].set_title('Ulagna slika')
ax[1].imshow(img_out); ax[1].set_axis_off(); ax[1].set_title('Izlazna slika');
```



Standardnu devijaciju degradacione funkcije možemo naći iterativnim postupkom korišćenjem widget-a.

```
[30]: # from ipywidgets import interact, interactive, fixed, interact_manual
# import ipywidgets as widgets

# # Ucitavanje slike u float formatu
# img = skimage.img_as_float(imread('sekvence/road.blur.png'))
# img = img[:, :, :3]
# [Ny, Nx] = img.shape[:2]

# img_r = img[:, :, 0]
# img_g = img[:, :, 1]
# img_b = img[:, :, 2]

# img_fft_r = fftshift(fft2(img_r))
# img_fft_g = fftshift(fft2(img_g))
# img_fft_b = fftshift(fft2(img_b))

# def findSigma(k, sigma):
#     H = lpfilter('gaussian', Ny, Nx, sigma)
#     W = (abs(H)**2) / (abs(H)**2 + 10**(-k))

#     img_fft_est_r = img_fft_r / H * W
#     img_fft_est_g = img_fft_g / H * W
#     img_fft_est_b = img_fft_b / H * W

#     img_est_r = real(ifft2(ifftshift(img_fft_est_r)))
#     img_est_g = real(ifft2(ifftshift(img_fft_est_g)))
#     img_est_b = real(ifft2(ifftshift(img_fft_est_b)))

#     img_est_r = np.clip(img_est_r, 0, 1)
#     img_est_g = np.clip(img_est_g, 0, 1)
#     img_est_b = np.clip(img_est_b, 0, 1)

#     img_out = np.zeros(img.shape)
```

```

#     img_out[:, :, 0] = img_est_r
#     img_out[:, :, 1] = img_est_g
#     img_out[:, :, 2] = img_est_b

#     # IsCRTavanje slike.
#     fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14,12), dpi=300)
#     ax = axes.ravel()
#     ax[0].imshow(img); ax[0].set_axis_off(); ax[0].set_title('Ulazna slika')
#     ax[1].imshow(img_out); ax[1].set_axis_off(); ax[1].set_title('Izlazna slika');
#     plt.show();

# interact(findSigma, k = (1, 7, 0.5), sigma = (10, 100, 1));

```

0.3 Zadatak 3

0.3.1 Adaptivni medijan filter

Osnovna suština **medijan** filtra jeste nalaženje medijane. Da bismo našli medijanu moramo sortirati niz. Postoji veliki broj algoritama za sortiranje niza. Neki su vremenski efikasniji, neki prostorno, tj. zauzimaju malo dodatne memorije. Najjednostavniji algoritam za sortiranje je **Straight Insertion Sort**. S obzirom da se sortiranje ne vrši nad mnogo velikim nizovima, ovaj algoritam bi mogao da ima zadovoljavajuće rezultate. Probelem bi mogao da bude broj iteracija. Biće ih onoliko koliko ima piksela u slici, pa bi **Straight Insertion Sort** mogao biti neisplativ. Najgora vremenska kompleksnost ovog algoritma je n^2 .

Sortiranje se vrši tako što se kreće od prvog elementa u nizu, prolazi se kroz sve naredne elemente i ukoliko je neki od njih manji, menjaju mesta. Maksimalni element se nalazi na poslednjem mestu, minimalni na prvom a medijana je središnji element ukoliko je broj elemenata neparan, ukoliko je paran onda je medijana srednja vrednost središnja dva elementa.

```
[38]: def sort(arr):
    n = len(arr)

    for i in range(n - 1):
        for j in range(i, n):
            if arr[i] > arr[j]:
                tmp = arr[i]
                arr[i] = arr[j]
                arr[j] = tmp

    return arr

def maxMinMedian(arr):
    n = len(arr)

    arrSorted = sort(arr)
```

```

if n % 2 == 0:
    median = (arrSorted[n // 2] + arrSorted[n // 2 - 1]) / 2
else:
    median = arrSorted[n // 2]

return arrSorted[-1], arrSorted[0], median

```

Naredni algoritam za sortiranje niza je **Heap Sort**. Algoritam je baziran na **Heap** strukturi podatka. Heap struktura je zapravo binarno stablo. Svaki element ima "roditelja" i "decu" sem prvog i poslednjih $\frac{1}{2}N$ elemenata. Prvi element nema "roditelja", dok poslednji elementi nemaju "decu". Svaki roditelj ima dva "deteta" i svako dete jednog "roditelja".

Max Heap predstavlja takav heap u komje je svaki "roditelj" veći od oba "deteta". Odatle sledi da je prvi element maksimalni element niza. Sortiramo niz tako što zamenimo mesta prvom i poslednjem elementu, izbacimo poslednji element, koji je sada maksimalan u nizu, iz razmatranja, i ponovo napravimo **Max Heap** nad strukturom čija je dužina umanjena za jedan. Postupak ponovimo N puta, dok ne prođemo kroz sve elemente niza. Najgora vremenska kompleksnost je $n \lg(n)$.

Ovakvo sortiranje je **sortiranje u mestu** što znači da se ne koristi dodatna memorija. Sve se obavlja nad ulaznim nizom. Nama to ne odgovara, zato što mi ne želimo da traženjem medijane, maksimalnog i minimalnog elementa niza narušimo raspored piksela u lokalnom susedstvu. Zbog toga je potrebno da elemente ulaznog niza, piksele, kopiramo u novi niz i da sortiranje vršimo nad kopiranim nizom.

```
[39]: # Building max or min heap
def heapify(arr, n, i):
    largest = i          # Initialize largest as root
    left = 2 * i + 1     # Left child
    right = 2 * i + 2    # Right child

    # See if left child exist and
    # is greater then root/
    if left < n and arr[left] > arr[largest]:
        largest = left

    # See if right child exist and
    # is greater then root
    if right < n and arr[right] > arr[largest]:
        largest = right

    # If needed, change root and
    # heapify again with largest as a root
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heapSort(arr):
```

```

n = len(arr)    # Size of an array

# Build max heap
for i in range(n // 2 - 1, -1, -1):
    heapify(arr, n, i)

# Sort max heap
for i in range(n - 1, 0, -1):
    arr[i], arr[0] = arr[0], arr[i]    # Swap first and last element of a heap
    heapify(arr, i, 0)

def findMaxMinMedian(arr):
    arrSorted = arr.copy()
    n = len(arrSorted)

    # Sort an array
    heapSort(arrSorted)

    # See if n is even or odd
    if n % 2 == 0:
        median = (arrSorted[n // 2] + arrSorted[n // 2 - 1]) / 2
    else:
        median = arrSorted[n // 2]

    return arrSorted[-1], arrSorted[0], median

```

Funkcija koja proverava validnost unetih parametara. Maksimalni prozor filtra mora biti pozitivan neparan ceo broj veci od 1. Matrica ulazne slike moze biti u formatu *uint8* ili *double*.

```
[43]: def checkProperties(I, Smax):
    #Check if correct digit is entered for maximum size window
    if (Smax <= 0) or (type(Smax) != int) or (Smax % 2 == 0):
        raise ValueError("Maximum window size must be positive odd integer!")

    #Function accepts two type of data format uint8 and double
    if(type(I[0, 0]) != uint8 and type(I[0, 0]) != double):
        raise ValueError("Input image must be in format of uint8 or double!")
```

Ideja **adaptivnog medijan filtra** je da se koristi filter sto manje velicine, kako bi se smanjila izoblicenja ivica na slici. Da ne bismo dosli u situaciju da velicinu prozora povecavamo u nedogled, potrebno ga je ograniciti maksimalnom vrednoscu.

Filtriranje zapocinjemo prozorom minimalne velicine, 3. Ako je medijana lokalnog susedstva jednaka minimumu ili maksimumu tog lokalnog susedstva, i ako je velicina prozora manja od maksimalne velicine treba povecati prozor i krenuti ispochetka. Ako je prozor maksimalan, izlazni piksel je medijana lokalnog susedstva. Ako medijana nije jednaka ekstremumima, potrebno je proveriti da li je centralni piksel jednak ekstremumima. Ako jeste, velika je verovatnoća da je on zašumljen, pa je

izlazni piksel medijana tog lokalnog susedstva. Ako centralni piksel ipak nije jednak ekstremumima lokalnog susedstva, on zapravo nije zašumljen i ne treba ga dirati. Izlazni piksel je u tom slučaju jednak centralnom pikselu.

Sama postavka algoritma nas navodi da je najjednostavnije koristiti rekurzivno pozivanje funkcija. **Astage** prva provera, da li je izabrani prozor dovoljno velik. Ako jeste poziva se funkcija **Bstage**, koja proverava da li je centralni piksel jednak ekstremumima, ako nije prozor se uvećava i ponovo se poziva funkcija **Astage**.

Da bismo uštedeli na memoriji, možemo ne proširivati sliku. Na taj način smo efektivno smanjili prozor na ivičnim pikselima, to se nadomeštava samim algoritmom. Možda smo povećali vreme izvršavanja filtriranja ivičnih piksela, ali smo uštedeli na memoriji.

Radi smanjenja računa i eventualnog ubrzanja izvršavanja koda možemo koristiti stare vrednosti ekstremuma i medijana. Kada nađemo ekstremume i medijan za najmanje lokalno okruženje, ako je potrebno povećati prozor, za računanje novih ekstremuma i medijana možemo iskoristiti stare vrednosti zajedno sa novim elementima lokalnog okruženja. Da bismo ovo realizovali, potrebno je posebno pozivati funkcije za nalaženje maksimuma, minimuma i medijana. Po svakom pozivu morali bismo da prosleđujemo drugačiji niz, u opštem slučaju maksimalna, minimalna vrednost i medijana se razlikuju. To znači da bismo tri puta morali da sortiramo niz, i da pravimo dodatno dve nove kopije koje bi povećale memoriju kompleksnost. Pitanje je da li bi izvršavanje programa trajalo kraće. Filter je realizovan bez ove funkcionalnosti.

```
[56]: # A stage
def Astage(I, rmax, r, i, j):
    # Coordinates of local area of a central pixel.
    x1 = max(0, j - r)
    x2 = min(I.shape[1], j + r)
    y1 = max(0, i - r)
    y2 = min(I.shape[0], i + r)
    localArea = I[y1 : y2 + 1, x1 : x2 + 1]

    # Maximum, minimum and median value of local area.
    zmax, zmin, zmed = findMaxMinMedian(localArea.flatten())

    # Checking if median value is extrema.
    # If not, going to the B stage. If yes, checking if window size is maximum.
    # If window size is not maximum, incrementing window size and going to the
    ↪A stage.
    # If window size is maximum, returning median as a new central pixel value.
    if (zmed - zmin) > 0 and (zmed - zmax) < 0:
        output = Bstage(I[i, j], zmax, zmin, zmed)
        return output
    else:
        r += 1
        if r <= rmax:
            output = Astage(I, rmax, r, i, j)
            return output
        else:
```

```

        return zmed

# B stage
def Bstage(zxy, zmax, zmin, zmed):
    # Checking if central pixel value is extrema of local area.
    # If not, returning central pixel value.
    # If yes, returning mediana of a local area
    if (zxy - zmin) > 0 and (zxy - zmax) < 0:
        return zxy
    else:
        return zmed

def dos_adaptmedian(I, Smax):
    """
    Adaptive median filter implementation. Adaptive median filter removes
    →impulse noise from images,
    without degrading edges.

    This function supports Gray images.

    Input
    I : Input image, uint8 ([0,255]) or double ([0,1]).
    Smax : Maximum size of a filter window. Must be positive odd
    →integer greater than 1.

    Output
    Iout : Output image same size and type as input image.
    """

checkProperties(I, Smax)

[Ny, Nx] = I.shape

# Maximum radius of a moaximum window.
rmax = Smax // 2 - 1

# Initializing output matrix of an image.
Iout = np.zeros((Ny, Nx), dtype = type(I[0, 0]))

# For every pixel in image doing A and B stage.
for i in range(Ny):
    for j in range(Nx):
        r = 1
        Iout[i, j] = Astage(I, rmax, r, i, j)

return Iout

```

Adaptivni medijan filter može da se izbori sa procentima šuma 0.2 i 0.4 a da ne naruši ivice. Već za procenat šuma 0.8, ivice su vidno narušene, ali je slika i dalje filtrirana.

```
[57]: from skimage import util

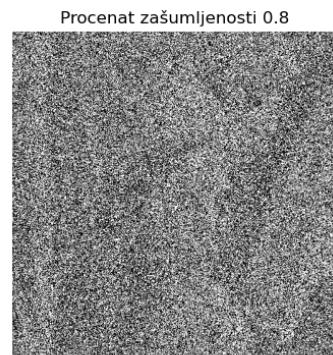
img = skimage.img_as_float(imread('sekvence/lena.tif'))

Smax = [7, 9, 21]
precNoise = [0.2, 0.4, 0.8]

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

for i in range(3):
    noisyImg = util.random_noise(img, mode = 's&p', amount = precNoise[i])
    clearImg = dos_adaptmedian(noisyImg, Smax[i])

    ax[i].imshow(noisyImg, cmap='gray'); ax[i].set_axis_off(); ax[i].
    set_title('Procenat zašumljenosti ' + str(precNoise[i]));
    ax[i + 3].imshow(clearImg, cmap='gray'); ax[i + 3].set_axis_off(); ax[i + 3].set_title('Prozor ' + str(Smax[i]) + 'x' + str(Smax[i]));
```



Prethodna funkcija je realizovana preko rekurzije. Rekurziju bi trebalo izbegavati zato što se pri svakom pozivu funkcije programski brojač postavlja na stek. Ovo usporava izvršavanje programa i okupira memoriju. Bolja realizacija bi bila uz pomoć petlji.

Iz tabele se vidi da postoji razlika u korišćenju algoritama za sortiranje, iako veličine nizova nisu mnogo velike. Vremena izvršavanja funkcija korišćenjem rekurzije i petlji se praktično ne razlikuju, ali i dalje je zauzeće memorije rekurzijom veće.

procenat zašumljenosti (veličina prozora)	0.2 (7x7)	0.4 (9x9)	0.8 (21x21)
Straight Insertion Sort (rekurzija) [s]	10.948	14.505	151.238
Heap Sort (rekurzija) [s]	7.529	8.374	41.944
Heap Sort (petlja) [s]	7.050	8.020	40.050

```
[54]: def dos_adaptmedian(I, Smax):
    """
        Adaptive median filter implementation. Adaptive median filter removes
        →impulse noise from images,
        without degrading edges.

    This function supports Gray images.

    Input
        I : Input image, uint8 ([0,255]) or double ([0,1]).
        Smax : Maximum size of a filter window. Must be positive odd
        →integer greater than 1.

    Output
        Iout : Output image same size and type as input image.
    """

    checkProperties(I, Smax)

    [Ny, Nx] = I.shape

    # Maximum radius of a moaximum window.
    rmax = Smax // 2 - 1

    # Initializing output matrix of an image.
    Iout = np.zeros((Ny, Nx))

    for i in range(Ny):
        for j in range(Nx):
            r = 1
            while r <= rmax:
                # Coordinates of local area of a central pixel.
```

```

        x1 = max(0, j - r)
        x2 = min(Nx, j + r)
        y1 = max(0, i - r)
        y2 = min(Ny, i + r)
        localArea = I[y1 : y2 + 1, x1 : x2 + 1]

        # Maximum, minimum and median value of local area.
        zmax, zmin, zmed = findMaxMinMedian(localArea.flatten())

        # Checking if median value is extrema. If yes, window size
        # is incremented. If not, breaking from loop.
        if (zmed == zmax) or (zmed == zmin):
            r += 1
        else:
            break

        # Checking if widow size is maximum or if central pixel value
        # is extrema. If yes, new central pixel value is median of
        # local area. If not, new pixel value is old value, pixel is unchanged.
        if (r == rmax) or (I[i, j] == zmin) or (I[i, j] == zmax):
            Iout[i, j] = zmed
        else:
            Iout[i, j] = I[i, j]

    return Iout

```

Rezultati su identični prethodnim.

```

[55]: img = skimage.img_as_float(imread('sekvence/lena.tif'))

Smax = [7, 9, 21]
precNoise = [0.2, 0.4, 0.8]

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

for i in range(3):
    noisyImg = util.random_noise(img, mode = 'salt', amount = precNoise[i])
    clearImg = dos_adaptmedian(noisyImg, Smax[i])

    ax[i].imshow(noisyImg, cmap='gray'); ax[i].set_axis_off(); ax[i].set_title('Procenat zašumljenosti ' + str(precNoise[i]));
    ax[i + 3].imshow(clearImg, cmap='gray'); ax[i + 3].set_axis_off(); ax[i + 3].set_title('Prozor ' + str(Smax[i]) + ' x' + str(Smax[i]));

```

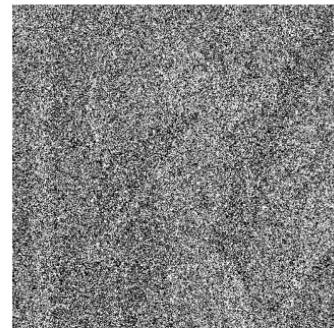
Procenat zašumljenosti 0.2



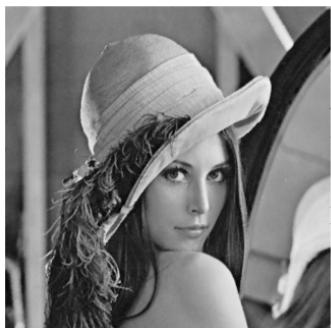
Procenat zašumljenosti 0.4



Procenat zašumljenosti 0.8



Prozor 7x7



Prozor 9x9



Prozor 21x21



[]: