

Vezba 1 - Python uvod

1 Pregled

Cilj vežbe je da se studenti upoznaju sa elementarnom sintaksom u pajton programskom jeziku i Jupyter razvojnim okruženjem

2 Python jezik

Python je programski jezik visokog nivoa (kao C, C++, MATLAB...), dizajniran za lako pisanje i tumačenje od strane ljudi i lako čitanje i obradu od strane računara. Suštinski, kada pišemo set instrukcija (izvorni kod, *engl. source code*) koristimo set pravila da formiramo rečenice koje formiraju priču koju računar treba da razume. Proces prevođenja sa python jezika na računarski razumljiv (mašinski) jezik je zadatak tzv. interpretera. On iščitava naš kod, tumači ga red po red (princip kao i kod MATLAB-a) i izvršava instrukcije. Neki od osnovnih instrukcija koje se koriste jeste čuvanje vrednosti u obliku promenljivih, izvršavanje instrukcija pod određenim logičkim uslovima, definisanje seta instrukcija pod jednim imenom... Pre nastavka, definisaćemo integrisana razvojna okruženja, tj. aplikacije koje spajaju sve potrebne elemente za pisanje i izvršavanje koda.

2.1 Integrисано развојно окруžење

Integrисано развојно окруžење (*engl. Integrated Development Environment, IDE*) je aplikacija koja obezbeđuje celinu koja objedinjuje različite aspekte u okviru pisanja programa, kao što je pristup editoru, debug-ovanje, pristup interpreteru, kompjleru, fajl browseru, označavanje sintakasnih jedinica...

U okviru kursa predviđeno je korišćenje Jupyter notebook okruženja.

Radi kompletности stoje informacije i o PyCharm okruženju.

2.1.1 Jupyter notebook

Jupyter notebook je web-aplikacija koja omogućuje kreiranje dokumenata koji sadrže kombinaciju koda koji se može pozivati, različitih vizuelizacija, formula i teksta. Kao posebna varijanta jupyter notebook okruženja predstavljaju "sveske" koje su hostovane na Google serverima i čuvaju se na Google disk - Google Colaboratory, u kome je i pisan ovaj dokument. Google takođe obezbeđuje i svoju cloud infrastrukturu u vidu CPU i GPU jedinica za korišćenje pri pokretanju python3 instrukcija.

<https://research.google.com/colaboratory/faq.html>

https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index#scrollTo=5fCEDCU_qrC0

2.1.2 PyCharm

PyCharm predstavlja razvojno okruženje za python jezik koji nudi dodatne pogodnosti u vidu pametne analize koda, bolje organizacije (u vidu skripti i modula), lakšeg debug-ovanja, korišćenja odvojenih python okruženja, itd. Što se uvećava složenost programa ove pogodnosti se dodatno naglašavaju.

<https://www.jetbrains.com/pycharm/features/>

3 Osnove Python-a

U ovom delu ukratko analiziramo osnovne operacije zajedničke za većinu programskih jezika visokog nivoa, kao i neke specifične za python:

- Krijanje varijabli i osnovne operacije nad njima
- Uslovno izvršavanje instrukcija
- Definisanje funkcija
- Pisanje programskih petlji
- Liste
- Tuples (uređene n-torke)
- Paketi i moduli

3.1 Krijanje varijabli i osnovne operacije nad njima

Pravila za imena varijable: * može da sadrži slovne i numeričke karaktere i karakter "_" * ne može započeti brojem * ne može biti jedna od python rezervisanih reči

and, del, from, None, True, as, elif, global, nonlocal, try, assert, else, if, not, while, break, except, import, or, with, class, False, in, pass, yield, continue, finally, is, raise, async, def, for, lambda, return, await

Preporuke za imena varijable: * davati opisna imena * započinjati malim slovom * koristiti velika slova ili "_" za razdvajanje reči u varijabli

Primeri:

```
[ ]: precnik_kruga = 3 # integer
visina = 2.1 # float
ime = 'Marko' # string
```

Napomena: **komentari** predstavljaju delove koda koji se ne izvršavaju. Obično predstavljaju sekvencu karaktera koji imaju informativni sadržaj i navodi se nakon znaka # - može biti u okviru neke instrukcije ili u zasebnoj liniji.

```
[ ]: # primer pogrešnog definisanja promenljive
1broj = 2
```

```
File "<ipython-input-7-c7847b3e87a7>", line 2
 1broj = 2
^
SyntaxError: invalid syntax
```

tip promenljive možemo proveriti pozivom funkcije type

```
[ ]: type(visina)
```

```
[ ]: float
```

ispisivanje promenljive:

```
[ ]: precnik_kruga # moguce za ispis u okviru notebook-a
```

```
[ ]: 3
```

```
[ ]: print(visina) # poziv funkcije za ispis
```

2.1

```
[ ]: print('Njegovo ime je {0}. \nKrug je precnika {1}.'.format(ime, precnik_kruga))
# ispis sa formatiranjem stringa
```

Njegovo ime je Marko.

Krug je precnika 3.

```
[ ]: print(f'Njegovo ime je {ime}. \nKrug je precnika {precnik_kruga}.')
```

Njegovo ime je Marko.

Krug je precnika 3.

Python je jezik slabog tipa, što znači da nema potrebe eksplicitnog navođenja tipa promenljive prilikom definisanja varijabli.

Varijable takođe možemo definisati korišćenjem ranije formiranih vrednosti i dozvoljenih operatorka za dati tip:

```
[ ]: a = 10
b = 5
c = a + b
print(c)
```

15

```
[ ]: a = 'a'
b = 'b'
c = a + b
print(c)
```

ab

Neke druge osnovne operacije nad brojčanim promenljivama

Operacija	Simbol
Sabiranje	+
Oduzimanje	-
Množenje	*
Celobrojno deljenje	//
Deljenje	/
Stepenovanje	**

3.2 Uslovno izvršavanje instrukcija

Kontrola izvršavanja komandi vrši se pomoću sintakse koja je slična i za druge programske jezike

```
if elif else
```

gde se proverava neki logički uslov (boolean tip)

```
if neki_uslov_je_zadovoljen:  
    izvrsi komandu 1  
elif proveri_drugi_uslov:  
    izvrsi komandu 2  
else:  
    izvrsi komandu 3
```

Moguće je napraviti konstrukciju od samo if uslova ili samo od if else

```
if neki_uslov_je_zadovoljen:  
    izvrsi komandu 1  
  
if neki_uslov_je_zadovoljen:  
    izvrsi komandu 1  
else:  
    izvrsi komandu 2
```

Obratiti pažnju da na kraju linije sa pozivom if komande stoji znak ":" a instrukcije koje su u okviru uslova su uvučene za jedan tabulator

Primeri:

```
[ ]: brojac = 5  
  
if brojac == 5:  
    print('Brojac je 5')  
    print("Ova komanda se izvrsava van 'if' uslova")  
  
if brojac+10 == 5:  
    print('Brojac je 5')
```

```

else:
    print('Brojac nije 5')
print("Ova komanda se izvrsava van 'if' uslova")

```

Brojac je 5
Ova komanda se izvrsava van 'if' uslova
Brojac nije 5
Ova komanda se izvrsava van 'if' uslova

Osnovne relacione operacije:

Relacione operacije	Objašnjenje
!=	nije jednako
==	jeste jednako
>	je veće
<	je manje
>=	je veće ili jednako
<=	je manje ili jednako

Kombinovanje uslova je moguće korišćenjem komandi `and` za logičko *i* i `or` za logičko *ili*

```
[ ]: a = 3
b = 5
if a > 0 and b == 5:
    print('Tacno')
```

Tacno

3.3 Definisanje funkcija

Funkcije predstavljaju imenovane sekvene instrukcija koje se izvršavaju po pozivu tog imena.

primeri koje smo imali do sada:

```
type(ime)
print('Hello world')
```

Vrednosti koje su prosleđenje unutar zagrada () predstavljaju argumente funkcije. Funkcija prima ulazni argument i vraća neku povratnu vrednost.

Pored ugrađenih funkcija, možemo definisati naše funkcije koristeći sintaksu:

```
def ime_funkcije():
    neke instrukcije
```

`def` je ključna reč koja ukazuje da se definiše funkcija sa novim imenom. Unutar zagrada mogu se dodatno navoditi potrebni argumenti funkcije, a unutar tela funkcije se može obezbediti i povratna vrednost pomoću ključne reči `return`. Slično kao i ranije, na kraju linije sa `def` pozivom stoji znak ":" a instrukcije koje su u okviru funkcije su uvučene za jedan tabulator

Primeri:

```
[ ]: def saberi(a,b):
    c = a + b
    print('zbir je {}'.format(c))
    return c

rezultat = saberi(3,4)
print(rezultat)
```

```
zbir je 7
7
```

```
[ ]: def saberi_i_oduzmi(a,b):
    c = a + b
    d = a - b
    return c,d

rez_zbir, rez_razlika = saberi_i_oduzmi(3,4)
print(rez_zbir, rez_razlika)
```

```
7 -1
```

3.4 Pisanje programskih petlji

Automatizaciju izvršavanja seta instrukcija možemo ostvariti korišćenjem programskih petlji: `for` i `while`

`for` - petlja koja izvršava naredbe preodređeni broj puta

```
for i in range(a,b,korak):
    izvrsi komandu
```

petlja se izvršava za vrednosti `i` koje su u intervalu $[a, b]$ tj. od `a` do `b` sa korakom `korak`.

```
[ ]: for i in range(1,11,2):
    print(i)
```

```
1
3
5
7
9
```

Funkcija `range` se može pozvati i sa jednim parametrom:

```
for i in range(n):
    izvrsi komandu
```

gde bi se u tom slučaju vrednosti `i` menjale od 0 do $n-1$.

```
[ ]: zbir = 0
      for i in range(3):
          zbir += i

      print(zbir)
```

3

while - petlja koja izvršava naredbe dokle god je definisan logički uslov uz nju.

```
while logicki_uslov:
    izvrsi komandu
```

Dodatna kontrola toka izvršavanja u okviru petlji:

- * break - prekida izvršavanje i izlazi iz unutrašnje petlje
- * continue - preskače trenutnu iteraciju i nastavlja sa sledećom
- * else - set instrukcija se izvršava nakon pravilnog završetka petlje (npr. prolazak kroz sve elemente u for)

```
[ ]: broj = 0

while broj < 10:
    broj += 1

    if broj % 2 == 0:
        print('{} je paran broj'.format(broj))
        continue
    else:
        print('{} je neparan broj'.format(broj))
        continue

    print('naknadni ispis')
else:
    print('Završena petlja')
```

```
1 je neparan broj
2 je paran broj
3 je neparan broj
4 je paran broj
5 je neparan broj
6 je paran broj
7 je neparan broj
8 je paran broj
9 je neparan broj
10 je paran broj
Završena petlja
```

3.5 Liste

Liste predstavljaju nizove nekih vrednosti koje u opštem slučaju mogu biti različitih tipova. Liste se formiraju postavljanjem vrednosti elemenata odvojenih zarezom u uglaste zagrade [].

```
a = [1,2,'3',4]
```

gde je a lista sa elementima celobrojnih vrednosti 1,2,4 i karakterom '3'

Možemo pristupati i menjati vrednosti listi tako što koristimo lokaciju traženih elemenata kao indeks za pristup. U okviru python jezika indeksiranje kreće od 0.

```
print(a[1])
```

bi ispisao 2, jer se on nalazi na poziciji sa indeksom 1, što je druga pozicija

```
[ ]: a = [1,2,'3',4]
      print(a[1])
```

```
a[0] = 4
      print(a)
```

```
2
[4, 2, '3', 4]
```

Indeksiranje takođe može biti i od nazad

```
[ ]: print(a[-1]) # ispisati poslednji element
```

```
4
```

for petlja se često koristi za prolazak po elementima nekih nizova u redosledu po kome se javljaju.

```
for i in ['a','b',3]:
    izvrsi komandu
```

gde su sada vrednosti i elementi ove liste.

```
[ ]: for i in ['a','b',3]:
      print(i)
```

```
a
b
3
```

Stringovi su suštinski uređeni nizovi karaktera kojima možemo baratati slično kao i listama, ali ne možemo menjati vrednosti

```
[ ]: ime = 'Marko'
      print(ime[2])
```

```
r
```

```
[ ]: ime[0] = 'D' # primer da se ne mogu menjati vrednosti
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-2-1b4bdfc9a543> in <module>()
----> 1 ime[0] = 'D' # primer da se ne mogu menjati vrednosti
```

```
TypeError: 'str' object does not support item assignment
```

3.5.1 List slice

Ponekad je potreban pristup više uzastopnih elemenata liste, koji bi isključivali neke elemente. U tom kontekstu pored indeksiranja definiše se tzv. (engl) *slicing*

```
[ ]: a = [1, 2, 3, 4, 5, 6, 7]
```

```
print(a[1:3]) # izvuceni elementi sa indeksima 1,2  
print(a[:2]) # izvuceni elementi sa indeksima od 0 do 2 (ne računajući 2)  
print(a[:-1]) # izvuceni svi elementi osim poslednjeg  
print(a[2:]) # izvuceni svi elementi osim prva dva  
print(a[::-2]) # izvuceni svaki drugi element
```

```
[2, 3]  
[1, 2]  
[1, 2, 3, 4, 5, 6]  
[3, 4, 5, 6, 7]  
[1, 3, 5, 7]
```

3.5.2 List operatori, metode i funkcije

Za liste su definisani operatori + i *

Operator + vrši spajanje listi

```
[ ]: a = [1, 2, 3]  
b = [4, 5, 6]  
c = a + b  
print(c)
```

```
[1, 2, 3, 4, 5, 6]
```

Operator * vrši ponavljanje liste određen broj puta

```
[ ]: d = a * 4  
print(d)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Metode predstavljaju funkcije asocijirane sa pojedinim objektima (kao liste).

Za dodavanje elemenata listama koristimo metodu append (za pojedinačne elemente) i extend (za dodavanje liste na listu)

```
[ ]: a = [1,2,3]  
a.append(4)  
print(a)
```

```
b = [5,6,7]
a.extend(b)
print(a)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5, 6, 7]
```

Za brisanje elemenata spram indeksa koristimo metodu `pop` (ako nam je potrebno sačuvati izbačenu vrednost) i operator `del` (ako nam nije potrebno sačuvati izbačenu vrednost)

```
[ ]: a = [1,2,3,4,5]
x = a.pop(1)
print(x)
print(a)

del a[0]
print(a)
```

```
2
[1, 3, 4, 5]
[3, 4, 5]
```

Ako znamo vrednost a ne njen indeks, može se koristiti metoda `remove`

```
[ ]: a = [1,2,3,4,5]
a.remove(3)
print(a)
```

```
[1, 2, 4, 5]
```

Postoji set ugrađenih funkcija koje se mogu primenjivati nad listama, gde je najčešće korišćena funkcija za računanje broja elemenata liste `len`

```
[ ]: a = [1,20,3,[1,2],4]
print(len(a))
```

```
5
```

3.6 Tuples

Slični listama, ali nakon definisanja tuple vrednosti ne možemo ih menjati. Ove uređene n-torke vrednosti se formiraju u okviru običnih zagrada ()

```
[ ]: t = (1,'a', 2, 'b')
print(t)

print(t[1:3])

t[0] = 2 # primer da se ne mogu menjati vrednosti
```

```
(1, 'a', 2, 'b')
('a', 2)
```

```

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-42-ce4da72c5edb> in <module>()
      4 print(t[1:3])
      5
----> 6 t[0] = 2

TypeError: 'tuple' object does not support item assignment

```

Jedna specifičnost za liste i tuplove je mogućnost dodele njihovih pojedinačnih vrednosti na više promenljivih organizovanih kao tuple

```
[ ]: a,b = ['ime',' prezime']
print(a)
print(b)

# može se napisati i kao
(a,b) = ['ime',' prezime']
print(a)
print(b)
```

```
ime
prezime
ime
prezime
```

3.7 Paketi i moduli

Za navedene osobine i operacije nad listama, postavlja se problem kako bi npr. sabrali elemente 2 liste koje sadrže brojčane vrednosti. Pošto operator + definiše spajanje listi, potrebno je definisati zasebnu funkciju koja bi rešila ovaj problem

```
[ ]: # smatracemo da su liste a i b jednakih dimenzija
def saberi_liste(a,b):
    c = []
    for ai,bi in zip(a,b):
        c.append(ai+bi)

    return c

a = [1,2,3]
b = [4,5,6]
c = saberi_liste(a,b)
print(c)
```

```
[5, 7, 9]
```

`zip` funkcija u ovom slučaju pravi iterator tuplova elemenata parova obe liste po indeksima. Ako

želimo da ih vidimo, potrebno je rezultat konvertovati u listu. U okviru `for` petlje `ai` i `bi` dobijaju njihove odgovarajuće vrednosti za svaku iteraciju.

```
[ ]: print(list(zip(a,b)))
```

```
[(1, 4), (2, 5), (3, 6)]
```

Alternativno, problem se može rešiti i korišćenjem tzv. *list comprehension* koja predstavlja način korišćenja kraće sintakse za kreiranje novih listi.

```
[ ]: c1 = [ai+bi for (ai,bi) in zip(a,b)]
      print(c1)
```

```
[5, 7, 9]
```

Ovaj problem i slični, koji se često postavljaju u ovom kontekstu, razrešeni su u većoj meri u obliku klase i funkcija koji su napisani u okviru modula (fajlova koji sadrže python kod) i organizovani u celine kao paketi (suštinski kao folderi koji sadrže fajlove).

Jedan od najčešće korišćenih paketa za numeričku obradu je **NumPy** paket.

Uključivanje paketa u radno okruženje omogućeno je korišćenjem komande `import` praćeno sa imenom paketa. Dodatno, moguće je promeniti ime paketa pod kojim bi bio poznat u okviru skripte koristeći komandu `as`

```
[ ]: import numpy as np
```

U nastavku mogu se koristiti svi elementi implementirani u okviru paketa. Definisaćemo dva numpy niza koristeći `np.array` i sabratih ih. U ovom slučaju pristupamo `np` celini i u okviru nje sa znakom `.` pristupamo funkciji `array`

```
[ ]: a = np.array([1,2,3])
      b = np.array([4,5,6])
      c = a + b
      print(c)
```

```
[5 7 9]
```

Iz modula se takođe mogu direktno uključiti specifični podmoduli, umesto uključivanja celokupnog paketa.

```
[ ]: print(np.maximum([1, 0, 2], [3, -1, 4]))
```

```
[3 0 4]
```

Funkcija `maximum` vraća maksimalnu vrednost po svakom elementu prosleđenih n-dimenzionih nizova.

```
[ ]: from numpy import maximum
```

```
print(maximum([1, 0, 2], [3, -1, 4]))
```

```
[3 0 4]
```

3.8 Crtanje

U Python-u se za crtanje grafika najčešće koristi biblioteka matplotlib.

Postoji nekoliko tipova grafika koji se mogu nacrtati, a neki od njih su klasičan 2D grafik, scatter, histogram, itd. Pri iscrtavanju se navode 2 ulazna parametra, od kojih je prvi vektor vrednosti na x-osi, a drugi vektor vrednosti na y-osi.

```
[1]: import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0, 10.01, 0.01)
f = 1
s = np.cos(2*np.pi*f*t)

plt.figure()
plt.plot(t, s, linewidth=1)
plt.title('Kosinusni signal frekvencije 1 Hz prikazan na intervalu od 10 s')
plt.show()
```



4 Zadaci za samostalnu vežbu

4.1 Zadatak 1.1

Napisati funkciju koja računa sumu kvadrata prosleđene liste. Napraviti listu sa elemenatima [1,2,3] i testirati funkciju

[]: `#TODO`

4.2 Zadatak 1.2

Ponoviti prethodni zadatak, ali koristeći numpy niz umesto liste. Iskoristiti mogućnosti NumPy paketa.

[]: `#TODO`

4.3 Zadatak 1.3

Napisati funkciju koja proverava da li elementi u listi imaju rastući trend (svaki naredni element je veći od prethodnog). Funkciju proveriti za sledeće liste

[1, 3, 4, 6, 7, 9]
[-1, 2, 4, 1, 6]
[2, 10, 11, 12]
[3, 2, 1, 4, 5]

[]: `# TODO`

4.4 Zadatak 1.4

Napisati funkciju koja proverava da li je uneta reč palindrom. Funkciju isprobati nad rečima

vrata
potop
knjiga
pop
rotor
maam

[]: `#TODO`

5 Dokumentacija novih celina

- [Osnove](#)
- [Kontrola toka](#)
- [Liste](#)
- [List comprehension](#)
- [Paketi i moduli](#)
- [NumPy paket](#)
- [Matplotlib](#)

6 Dodatna literatura

Charles Severance, *Python for Everybody: Exploring Data Using Python 3*

[PY4E online kurs](#) - poglavlja 3-6, 9, 11

Vezba 2 - osnove DOS

1 Pregled

Ova vežba je podeljena u 2 celine. Prva je za upoznavanje sa osnovnom kontrolom python okruženja i integracijom u PyCharm/notebook IDE, dok je druga posvećena upoznavanju sa osnovnim operacijama za digitlanu obradu slike.

2 Preduslov

- Skinuti i instalirati [Anaconda](#) distribuciju za python
- Skinuti i instalirati [PyCharm Community](#) IDE (ili PyCharm Professional, Educational License sa @uns nalogom)

3 Anaconda python distribucija

Anaconda predstavlja distribuciju python jezika sa dodatnom mogućnošću lakog baratanja paketa i organizacije okruženja. Bazno okruženje koje se formira prilikom instalacije pokriva veliki broj paketa i predstavlja spremnu podlogu za početak programiranja. Pokazaćemo kako se može formirati odvojeno okruženje gde se mogu uključiti samo paketi koji nam trebaju (sa dodatnom mogućnosti kontrole verzije koja nam je potrebna).

3.1 Python okruženja

Anaconda podržava mnoštvo verzija python jezika i pratećih paketa. Suštinski jedno okruženje je formirano od jedne verzije python interpretera i paketa. Ovo omogućuje formiranje okruženja spram specifičnih zahteva i potreba za pojedine projekate. Iako je **bazno** okruženje (formirano pri inicijalnoj instalaciji) većinsko dovoljno za naše potrebe, ovde je demonstrirano formiranje novog okruženja.

Novo okruženje se može formirati na osnovu anaconda navigatora (GUI aplikacije koja je uključena u anaconda distribuciju) ili uz pomoć anaconda prompt-a. Ovde ćemo koristiti isključivo prompt jer daje znatno više mogućnosti kontrole pojedinih elemenata (npr. specifičnih verzija pythona i paketa).

Pokretanjem "Anaconda Prompt" aplikacije otvara se terminal sličan klasičnom "Command Prompt" ekranu (za win sisteme) ali koji omogućuje pozivanje specifičnih conda komandi vezane za anakonda distribuciju.

```
(base) C:\Users\Ivan>■
```

Stavka (base) označava da se nalazimo u baznom okruženju koje je formirano prilikom prvobitne instalacije. Kao što smo napomenuli, ovo okruženje je samo po sebi već dovoljno za početak programiranja i python-u bi mogli pristupiti jednostavnim kucanjem komande python. Komanda quit() nas vraća nazad u prompt.

```
(base) C:\Users\Ivan>python
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> quit()
(base) C:\Users\Ivan>■
```

Kreiranje novog okruženja moguće je korišćenjem komande conda create. Moguće je dodatno specificirati i željenu verziju python distribucije koja se instalira (npr. za obezbeđivanje odgovarajuće funkcionalnosti nekih paketa pravljenih u okviru specifične verzije) u formi

python=<#MAJOR version>.<#MINOR version>.<#PATCH version>

sa postavljenim odgovarajućim vrednostima umesto <> polja. Ovde je prikazano kreiranje okruženja sa verzijom 3.10

Novo okruženje možemo nazvati “pyDoslike” sa odgovarajućom python verzijom.

conda create -n pyDoslike python=3.10

```
(base) C:\Users\Ivan>conda create -n pyDoslike python=3.7.10
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: E:\Python\anaconda3\envs\pyDoslike

added / updated specs:
- python=3.7.10

The following packages will be downloaded:

  package          |      build
  certifi-2020.12.5 | py37haa95532_0    141 KB
  pip-21.0.1        | py37haa95532_0    1.8 MB
  python-3.7.10     | h6244533_0       14.5 MB
  setuptools-52.0.0  | py37haa95532_0    711 KB
  wincertstore-0.2   | py37_0            14 KB
  Total:           17.1 MB

The following NEW packages will be INSTALLED:

  ca-certificates   pkgs/main/win-64::ca-certificates-2021.1.19-haa95532_0
  certifi           pkgs/main/win-64::certifi-2020.12.5-py37haa95532_0
  openssl           pkgs/main/win-64::openssl-1.1.1j-h2bbff1b_0
  pip               pkgs/main/win-64::pip-21.0.1-py37haa95532_0
  python             pkgs/main/win-64::python-3.7.10-h6244533_0
  setuptools         pkgs/main/win-64::setuptools-52.0.0-py37haa95532_0
  sqlite             pkgs/main/win-64::sqlite-3.33.0-h2a8f88b_0
  vc                pkgs/main/win-64::vc-14.2-h21ff451_1
  vs2015_runtime    pkgs/main/win-64::vs2015_runtime-14.27.29016-h5e58377_2
  wheel              pkgs/main/noarch::wheel-0.36.2-pyhd3eb1b0_0
  wincertstore      pkgs/main/win-64::wincertstore-0.2-py37_0
  zlib               pkgs/main/win-64::zlib-1.2.11-h62dc97_4

Proceed ([y]/n)? -
```

Ulazak i izlazak iz ovog okruženja je moguć uz pomoć komandi conda activate pyDoslike i conda deactivate

```
(base) C:\Users\Ivan>conda activate pyDoslike
(pyDoslike) C:\Users\Ivan>
```

U okviru ovog okruženja možemo sada dodavati pakete koji su nam potrebni. Za instaliranje paketa potrebno je pozvati komande u formi

conda install ime_paketa

ili

pip install ime_paketa

Obe komande imaju mogućnost postavljanja specifične verzije za pojedine pakete koristeći =#zahtevana verzija po sličnom principu kao što smo specificirali verziju pythona za naš paket.

Za ovu vežbu potrebno je pozvati sledeće komande:

- instalacija paketa za numeričku obradu i analizu podataka

```
pip install numpy  
pip install pandas
```

- instalacija paketa za obradu slike

```
pip install scikit-image
```

- instalacija paketa za prikaz

```
pip install matplotlib
```

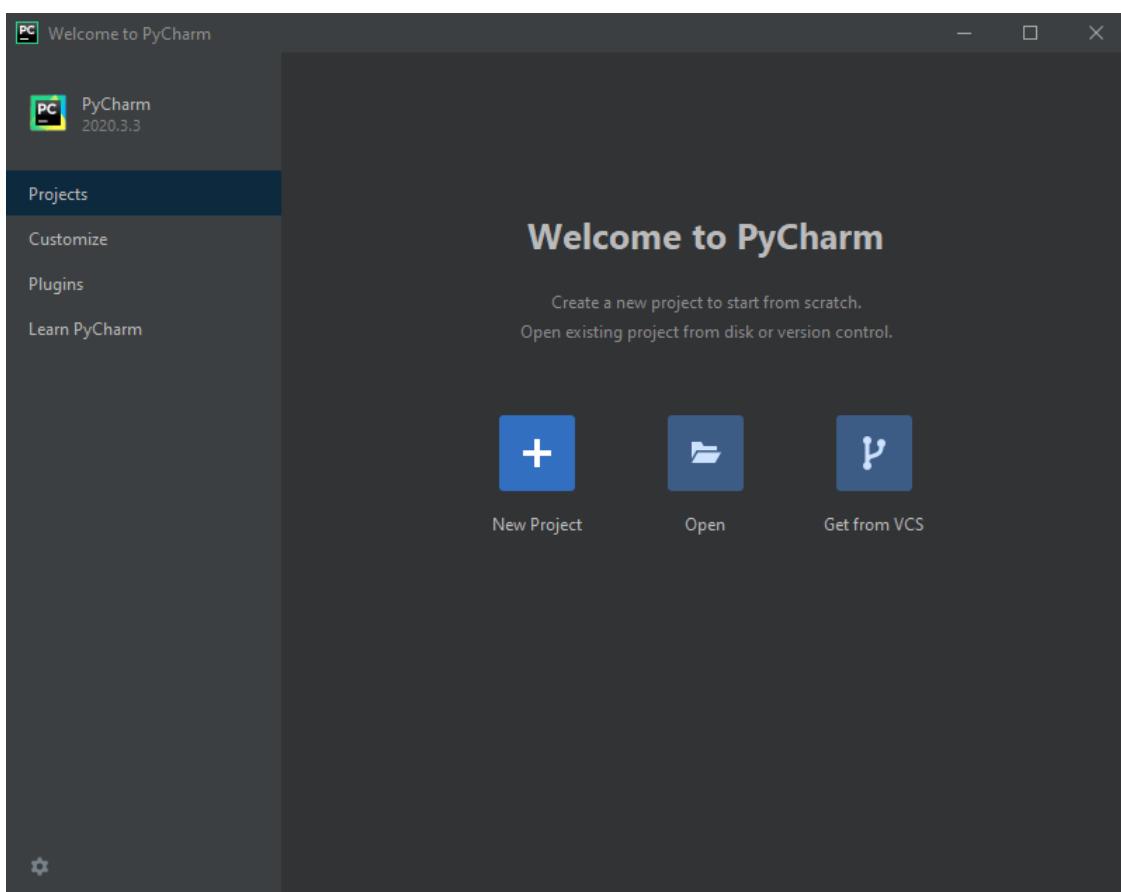
```
pip install plotly
```

```
pip install ipympml
```

Nakon ovih instalacija, potrebno je samo podešiti IDE koji koristimo na putanju odgovarajućeg okruženja.

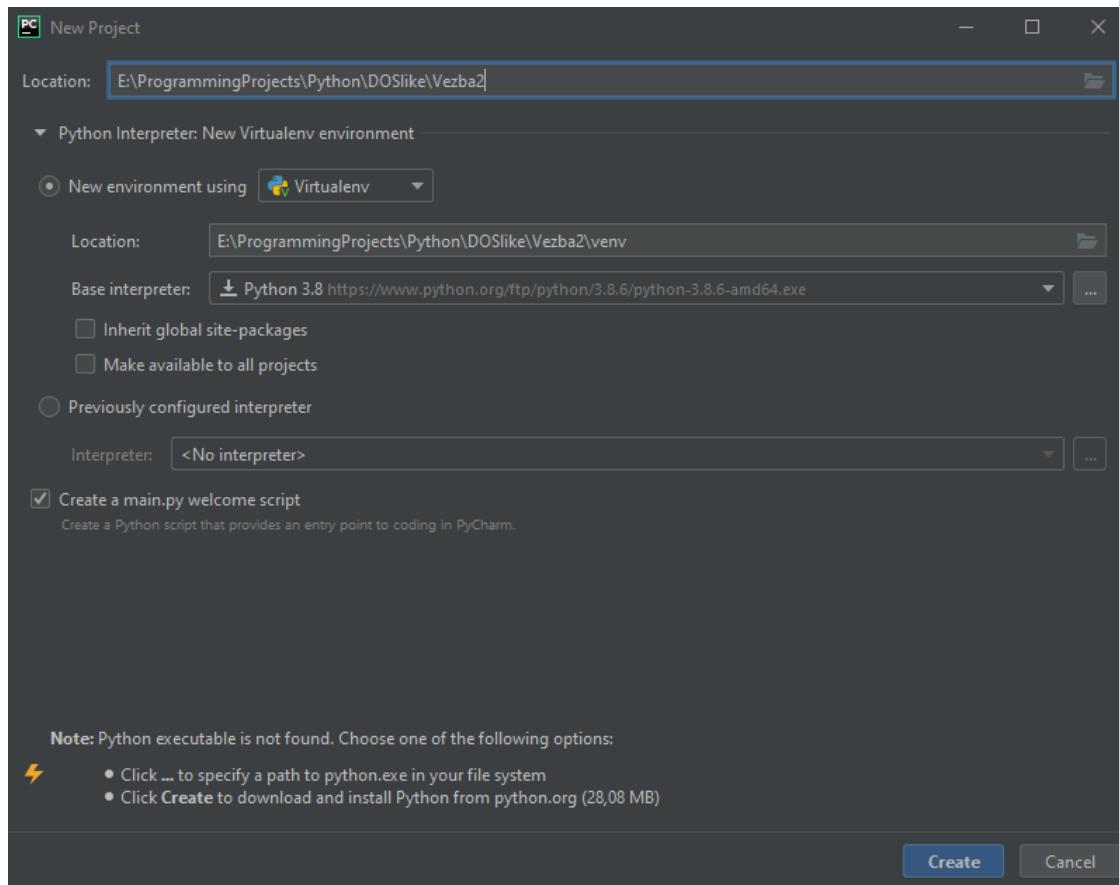
4 PyCharm Community

Prilikom pokretanja aplikacije otvara se prozor koji omogućuje kreiranje novih projekata ili otvaranje ranijih.

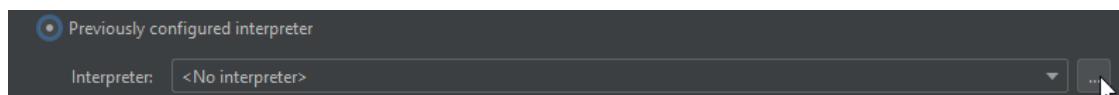


Klikom na "New Project" dugme prikazuje se ekran sa osnovnim podešavanjima u vidu putanje projekta (lokacije na disku koji će čuvati naš radni direktorijum) kao i podešavanje okruženja koje

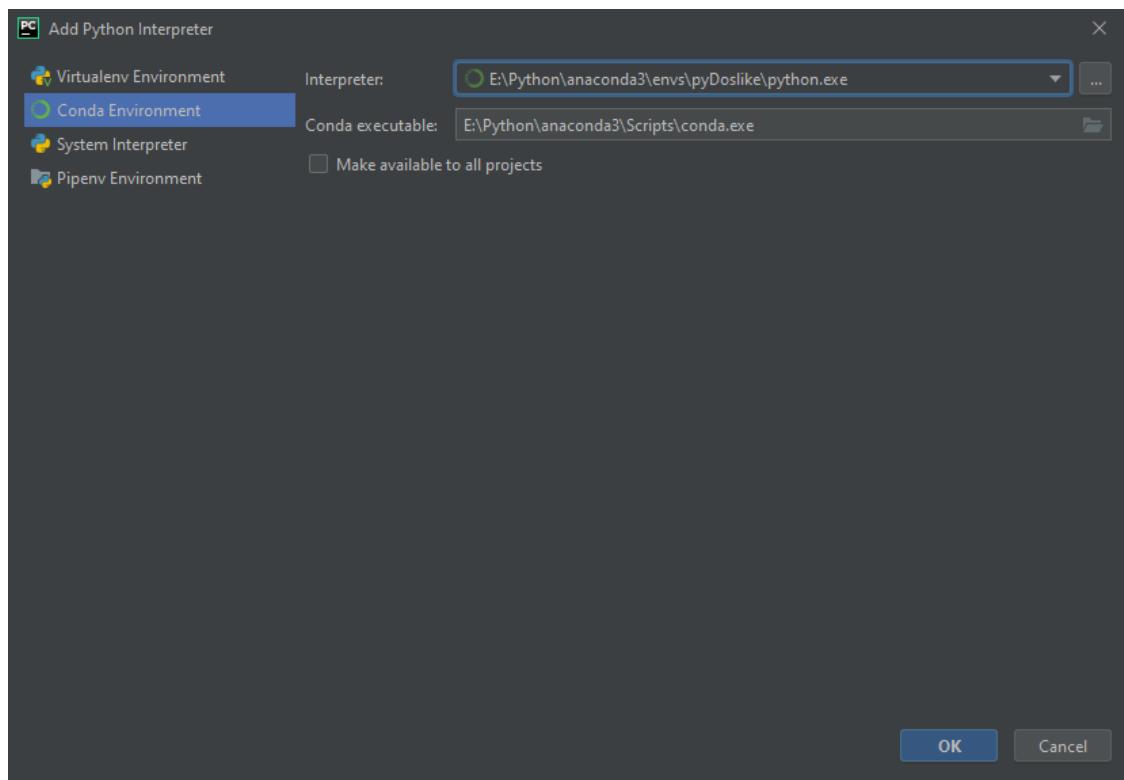
će koristiti. Za polje putanje, savet je da se formira nov folder na disku koji će biti isključivo za laboratorijske vežbe ovog predmeta a u okviru njega formirati projekte kao individualne vežbe.



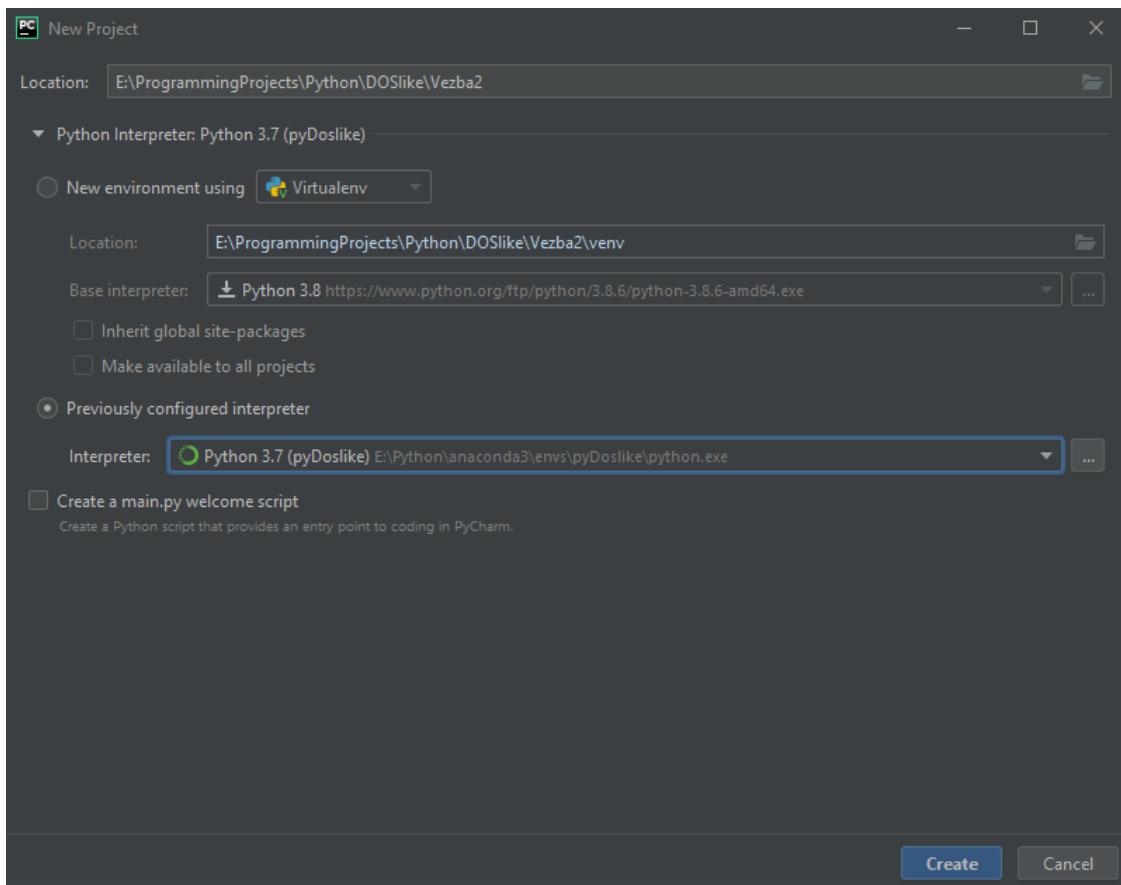
Za podešavanje interpretera potrebno je samo ukazati na njegovu putanju. Pod selekcijom sa "Previously configured interpreter" potrebno je kliknuti na oznaku "..." sa desne strane.



U okviru novog prozora treba se selektovati "Conda Environment" i pod stavkom "Interpreter:" postaviti putanju "python.exe" u okviru definisanog okruženja (lokacija svih novih okruženja je u folderu gde ste instalirali anaconda3 distribuciju pod nazivom "envs").

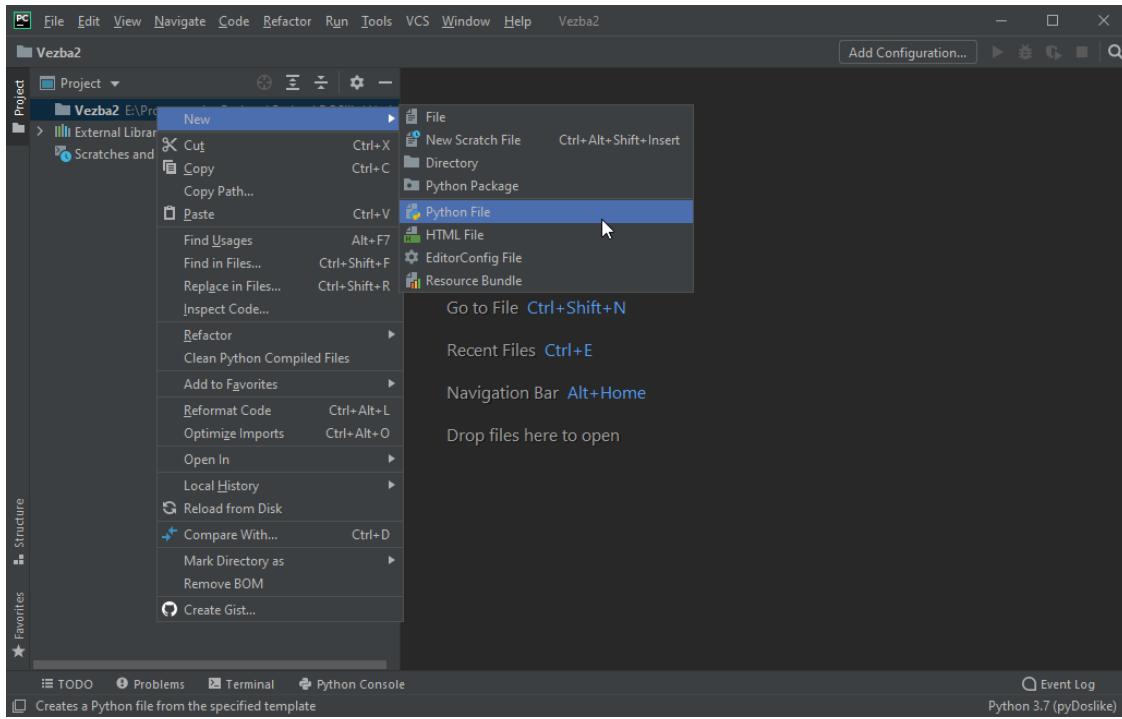


Na kraju možemo isključiti i pravljenje main.py skripte jer pravimo svoju posle.



4.1 Kreiranje skripti

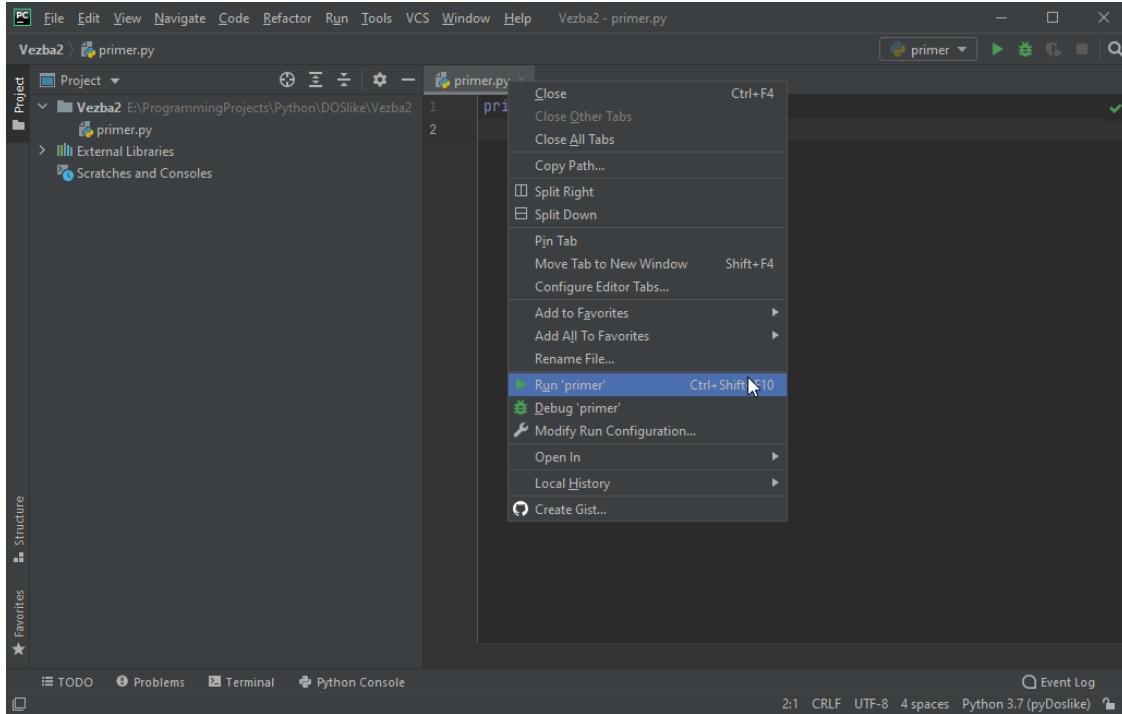
Nove python fajlove možemo ubacivati jednosavnim desnim klikom na folder našeg projekta i odabratи New->Python File. Za ovaj primer stavićemo ime `primer.py`. U opštem slučaju kreirali bi novu skriptu za svaki primer/zadatak.



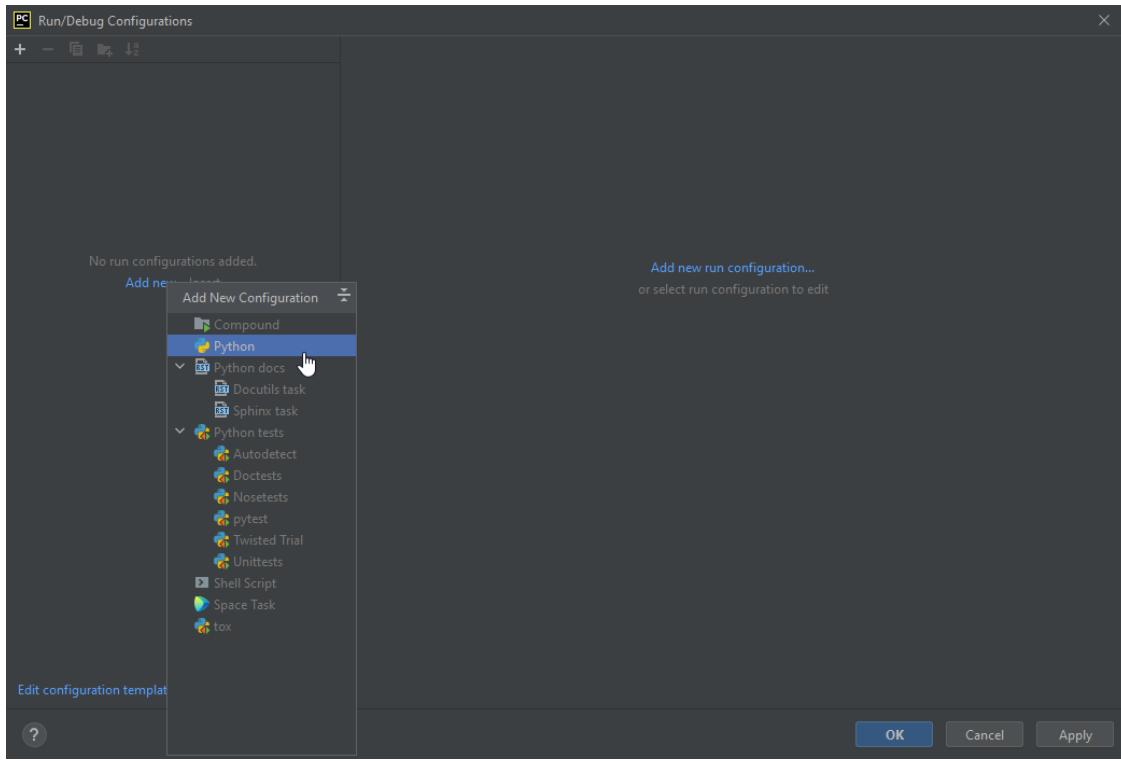
U okviru skripte za proveru da li je sve dobro uvezano možemo ubaciti test kod

```
print('hello world')
```

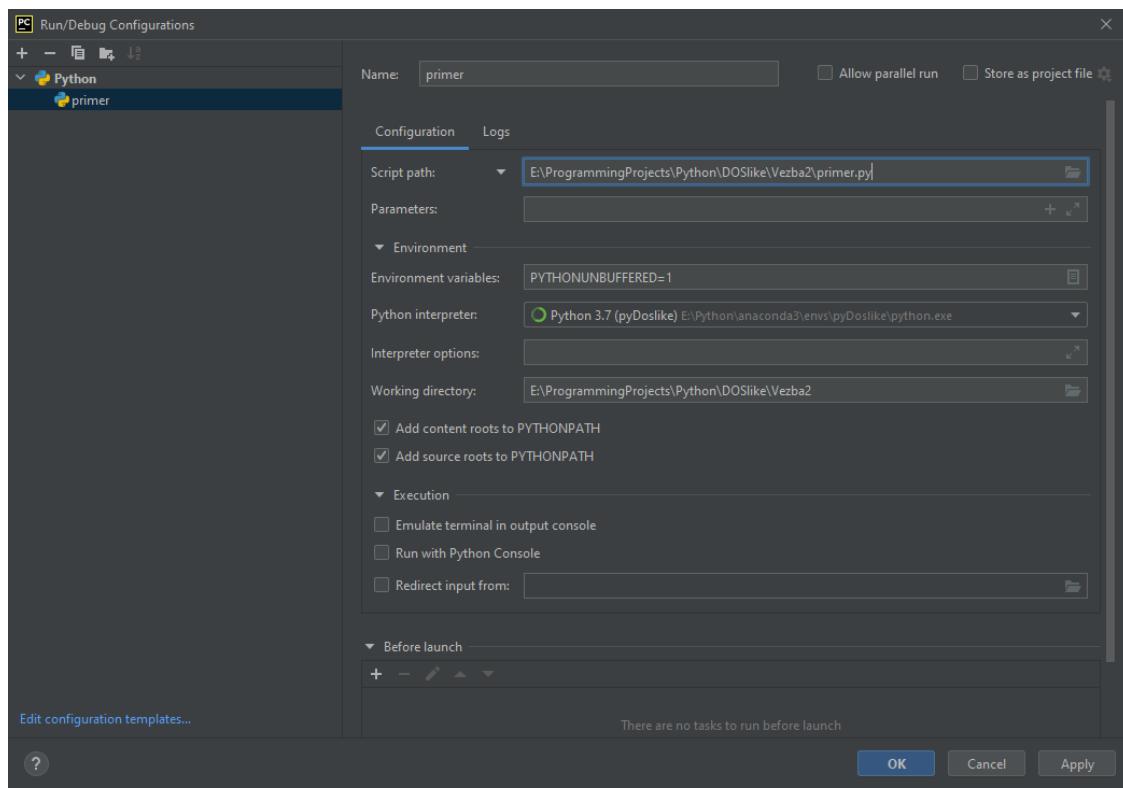
Inicijalno pokretanje ove skripte moguće je desnim klikom na nju i selekcijom "Run 'primer'" opcije .



Podešavanje za pokretanje skripte se može alternativno uraditi klikom na dugme “Add Configuration...” i dodavanjem nove Python konfiguracije

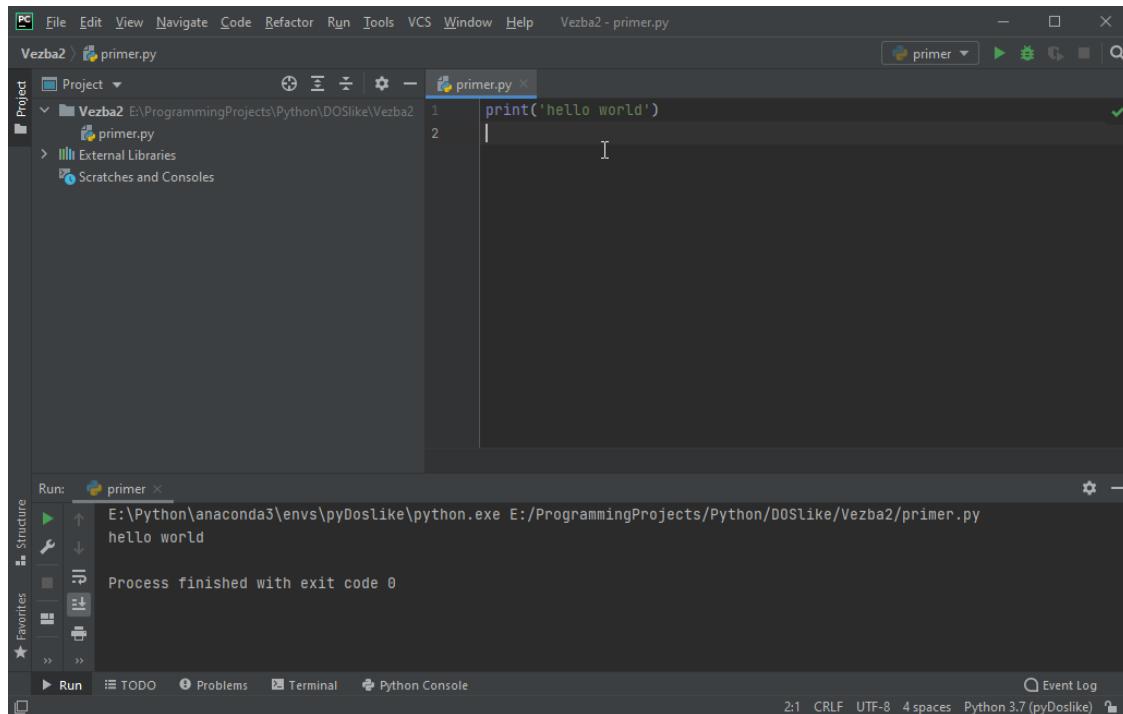


sa poljem “Script path” koje pokazuje na kreiranu skriptu i poljem “Python interpreter” koje pokazuje na kreirano python okruženje.



Nakon toga je moguće selektovati dugme u gornjem desnom delu ekrana.

Rezultat izvršavanja (ispisivanje reči) možemo videti u okviru otvorene sekcije "Run" na dnu ekrana.



4.2 Debugger

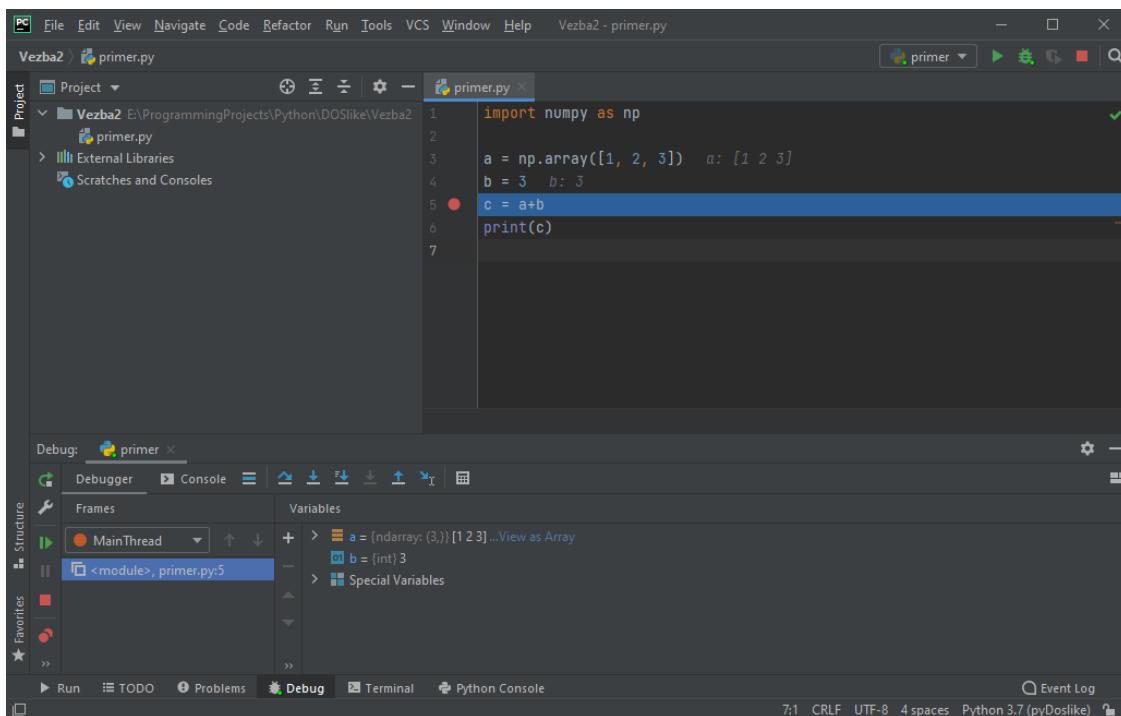
Precizniju kontrolu toka izvršavanja instrukcija i praćenje promena varijabli možemo obezbediti uz pomoć korišćenja debugger-a, koji ima identičnu funkcionalnost kao i na drugim IDE.

Za ovu demonstraciju postavićemo sledeće instrukcije u `primer.py`

```
import numpy as np
```

```
a = np.array([1, 2, 3])
b = 3
c = a+b
print(c)
```

Postavljanje breakpoint-a se vrši levim klikom na prostor desno od brojčane vrednosti za posmatranu liniju koda gde želimo da stanemo sa izvršavanjem. Linija na kojoj je postavljen breakpoint dobija crven krug postavljen ispred . Pokretanje debugger-a može da se uradi ponovo desnim klikom na `primer.py` i selekcijom "Debug 'primer'" ili klikom na  dugme u gornjem desnom delu ekrana.



Kao rezultat dobijamo obeleženu liniju gde je postavljen breakpoint, sa dodatnim ispisima u prethodnim linijama za pojedine vrednosti definisanih promenljivih (klikom na njih imamo dodatni set mogućih interakcija i inspekcija). Kontrola daljeg toka izvršavanja je moguća korišćenjem različitih opcija koje su postavljene u okviru nove sekcije "Debug" na dnu ekrana



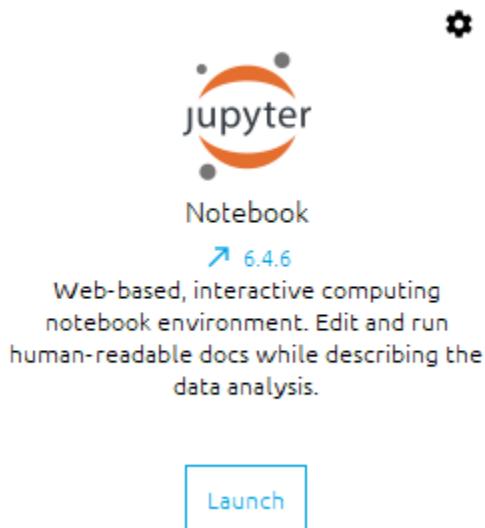
"Console" dugme omogućuje korišćenje interaktivnog konzolnog ekrana za vreme trajanja

debugger-a (što znači da ima mogućnost izvršavanja instrukcija između linija u skripti).

Klikom na "Resume Program" dugme ▶ u donje levom delu ekrana nastavlja se izvršavanje programa do sledećeg breakpoint-a ili kraja. Izlazak iz debugger-a moguć je korišćenjem "Stop" ■ dugmeta u gornjem desnom ili donje levom delu ekrana.

5 Jupyter notebook

Jupyter notebook okruženje se može lako instalirati pozivanjem komande `pip install jupyter` u okviru Anaconda Prompt aplikacije. Pokretanje okruženja je potom moguće pozivanjem komande `jupyter notebook`. Putanja radnog direktorijuma je spram putanje koja je ispisana u okviru konzole pre pokretanja okruženja.



Instalacija i pokretanje ovog okruženja je takođe moguća korišćenjem Anakonda Navigator aplikacije.

Posebna varijanta jupyter notebook okruženja predstavlja [Google Colaboratory](#). Google takođe obezbeđuje i svoju cloud infrastrukturu u vidu CPU i GPU jedinica za korišćenje pri pokretanju python3 instrukcija.

6 Napomena za fajlove i pakete koji se koriste u okviru vežbi

- Za rešenje različitih primera u okviru vežbi koristimo određen set fajlova. U okviru PyCharm/Jupyter notebook okruženja ti fajlovi bi trebali da budu na lokalnom disku, dok za Google Colaboratory potrebno je postaviti te fajlove ili koristiti uvezan google drive.
- Korišćene pakete bi trebalo importovati na početku notebook-a samo jednom, međutim u kontekstu učenja (*repetitio est mater studiorum*) svi potrebni paketi će biti importovani za svaki primer

7 DOSlike, osnovni pojmovi

Kroz primere ćemo se upoznati sa osnovnim operacijama za digitalnu obradu slike. Poseban akcenat je stavljen na uticaj opsega vrednosti slike na njeno prikazivanje i snimanje funkcijama u okviry python-a. Dodatne operacije koje se u vežbi obrađuju spadaju u osnovne manipulacije slikom kao što su: okretanje, isecanje delova i promena dinamičkog opsega.

7.1 Primer 2.1

U promenljivu `img` učitati `lena.png`. Prikazati učitanu sliku tako da se vrednost 0 prikaže kao crna, vrednost 255 prikaže kao bela, a vrednosti između njih prikažu kao nijanse sive.

Napomena: pokretanjem koda za prikaz koristeći `plotly` paket u okviru PyCharm-a otvara se browser sa interaktivnom slikom u prikazu. U slučaju da prikaz ne uspe promenite defaultni browser na vašem sistemu na neki drugi (firefox ili edge).

```
[2]: import plotly.express as px
from skimage import io

img = io.imread('lena.png')
fig = px.imshow(img, zmin=0, zmax=255, color_continuous_scale='gray')

# opcionalno
fig.update_layout(coloraxis_showscale=False)
fig.update_xaxes(showticklabels=False)
fig.update_yaxes(showticklabels=False)

# prikaz
fig.show()
```



```
[3] : img
```

```
[3]: array([[162, 162, 162, ..., 170, 155, 128],  
           [162, 162, 162, ..., 170, 155, 128],  
           [162, 162, 162, ..., 170, 155, 128],  
           ...,  
           [ 43,  43,  50, ..., 104, 100,  98],  
           [ 44,  44,  55, ..., 104, 105, 108],  
           [ 44,  44,  55, ..., 104, 105, 108]], dtype=uint8)
```

Za učitavanje slike koristimo `skimage` paket i funkciju `imread` iz `io` modula (input-output). Ulagani argument funkcije je string koji sadrži putanju do slike. U ovom primeru rezultat poziva funkcije `imread` je numpy ndarray (u ovom slučaju 2D matrica) koja se smešta u promenljivu `img`. Svaki element matrice odgovara jednom pikselu slike i sadrži vrednost intenziteta odgovarajućeg piksela.

Za prikazivanje slike koristimo `plotly` paket i `express` modul koji predstavlja način za brže prikazivanje nekih jednostavnijih formi. Funkcija `imshow` u okviru ovog modula kreira grafički objekat za prikazivanje slika. Prvi ulagani argument funkcije je promenljiva u kojoj se nalaze podaci koji treba da se vizualizuju. Drugi ulagani argument `zmin` predstavlja graničnu vrednost paleta tonova sa donje strane. Sve vrednosti koje su manje od granične vrednosti ili jednake sa graničnom vrednošću će biti prikazane kao minimalne za datu tonsku skalu. Treći ulagani argument `zmax` je granična vrednost paleta sivih tonova sa gornje strane, te će sve vrednosti koje su jednake sa njom ili veće od nje biti prikazane kao maksimalna date skala. Nijanse koje se dodeljuju vrednostima između donje i gornje granice se linearno raspoređuju. Poslednji argument funkcije `color_continuous_scale`

služi da se specificira koja skala boja da se koristi prilikom iscrtavanja. U ovom slučaju želimo da vidimo rezultat kao sivu sliku, te je potrebno staviti 'gray'. Za prikaz grafičkog objekta potrebno je pozvati njegovu metodu `show`.

Dodatne opcije za `fig` grafički objekat obezbeđuju uklanjanje prikaza skale i obeležene x i y ose.

Prelaskom kursora preko slike možemo videti informacije pojedinačnih piksela (elemenata matrice): * x : vrednost kolone pozicije piksela * y : vrednost vrste pozicije piksela * z/color : intenzitet piksela

Slike se obično prikazuju u koordinatnom sistemu gde je početak u gornjem levom čošku. Prilikom indeksiranja matrica za pojedinačne piksele prvo se postavlja indeks vrste pa potom kolone. `img[1,3]` bi nam dalo intenzitet piksela na vrsti sa indeksom 1 i kolonom sa indeksom 3.

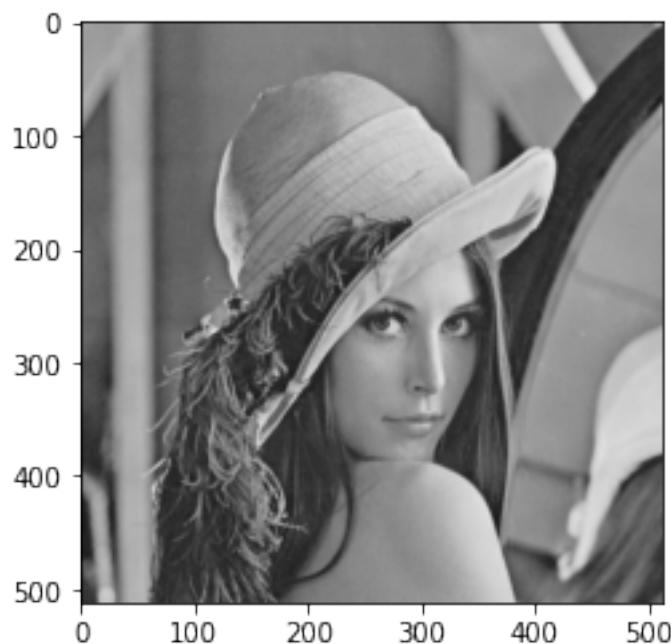
```
[4]: print(img[1,3])
```

161

Korišćeni paketi nisu jedini koji postoje za učitavanje i prikaz slika, stoga je predlog da se samostalno istraže drugi mogući načini. * [matplotlib](#) * [OpenCV](#) * [Pillow](#)

U nastavku je prikazana varijanta korišćenja `matplotlib` paketa.

```
[5]: import matplotlib.pyplot as plt  
  
plt.imshow(img, cmap='gray', vmin=0, vmax=255)  
plt.show()
```



Gde vrednosti parametara cmap, vmin i vmax odgovaraju vrednostima color_continuous_scale, zmin i zmax za plotly.express prikaz.

Obratiti pažnju da ovaj prikaz, iako validan, nema mogućnost korišćenja kurzora za jednostavnu inspekciju pojedinačnih vrednosti piksela. Ovaj nedostatak ne postoji ukoliko se paket koristi u okviru PyCharm okruženja.

Ova funkcionalnost može da se ukljuci, koristeći komandu

```
%matplotlib widget
```

pre prikazivanja slike, međutim treba obratiti pažnju za tumačenje indeksa sa necelobrojnim indeksima prilikom prikazivanja uvećanih segmenata.

```
import matplotlib.pyplot as plt
%matplotlib widget

plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.show()
```

Isključivanje ove funkcionalnosti moguće je pozivanjem komande

```
%matplotlib inline
```

7.2 Primer 2.2

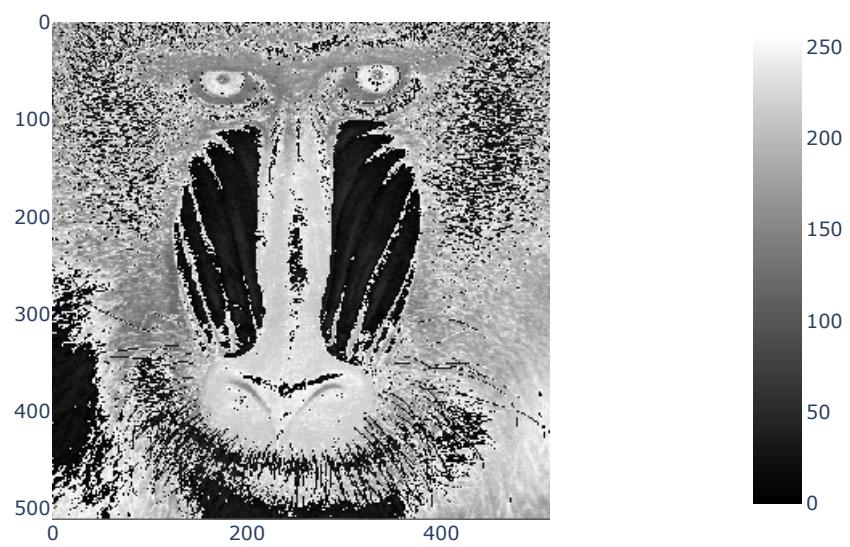
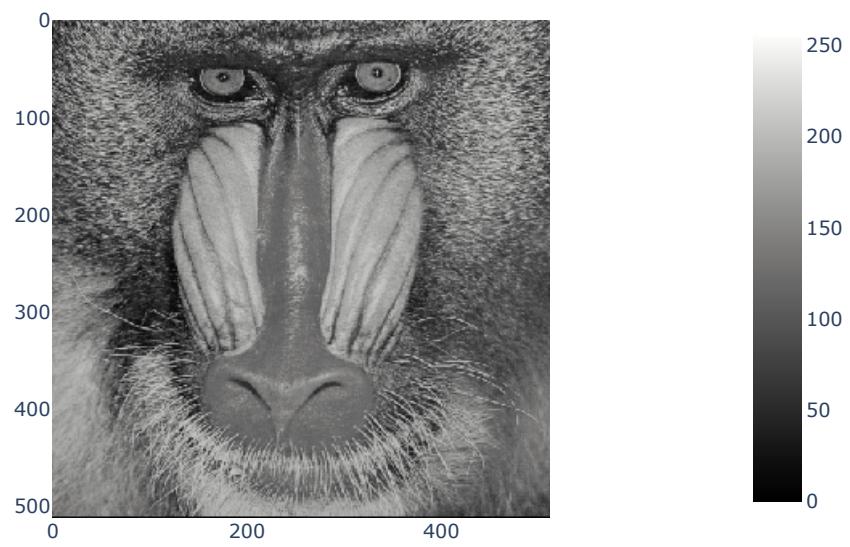
U promenljivu img učitati baboon.png. Odrediti promenljive: 1. img_2 čiji pikseli imaju intenzitete uvećane za 100 u odnosu na img 2. img_3 čiji pikseli imaju intenzitete umanjene za 100 u odnosu na img.

Prikazati slike koje su sadržane u promenljivama.

```
[6]: import plotly.express as px
from skimage import io

img = io.imread('baboon.png')
fig = px.imshow(img, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

img_2 = img + 100
fig = px.imshow(img_2, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



Numpy ndarrays tip omogućuje operacije kao što je sabiranje sa matricama. U slučaju dodava-

nja skalarne veličine ona se dodaje na svaki element matrice. Međutim, za inkrementaciju svih piksela za vrednost 100 očekivali bi znatno svetliju sliku (prema informacijama iz prethodnog primera). Za tumačenje ovog rezultata trebamo pogledati tip numpy niza naše slike gledajući njegov element `dtype`.

```
[7]: print(img.dtype)
```

```
uint8
```

uint8 vrednost označava da se svaki pojedinačni piksel naše slike predstavlja sa 8 bita informacije - tako da imamo mogući opseg nijansi od 0 (svih 8 nula) do 255 (svih 8 jedinica).

Napravićemo jedan manji niz ovog tipa (dodavanjem parametra `dtype`) i uvećati ga za 100 sa ispisivanjem elemenata da uočimo šta se dešava.

```
[8]: import numpy as np
```

```
tmp_array = np.array([0,100,200], dtype='uint8')
print(tmp_array)
print(tmp_array+100)
```

```
[ 0 100 200]
[100 200  44]
```

U slučaju da potencijalni zbir vrednosti elemenata ovog niza sa nekom konstantom prelazi maksimalnu moguću vrednost za dati tip, u tom slučaju vrednosti počinju da se ciklično ponavljaju (nakon 255 ide 0,1...). Ovo možemo primetiti jer dobijena vrednost 44 je u stvari $200+100-256$. Slično važi i za potencijalne rezultate dobijene koji su manji od 0.

```
[9]: 200+100-256
```

```
[9]: 44
```

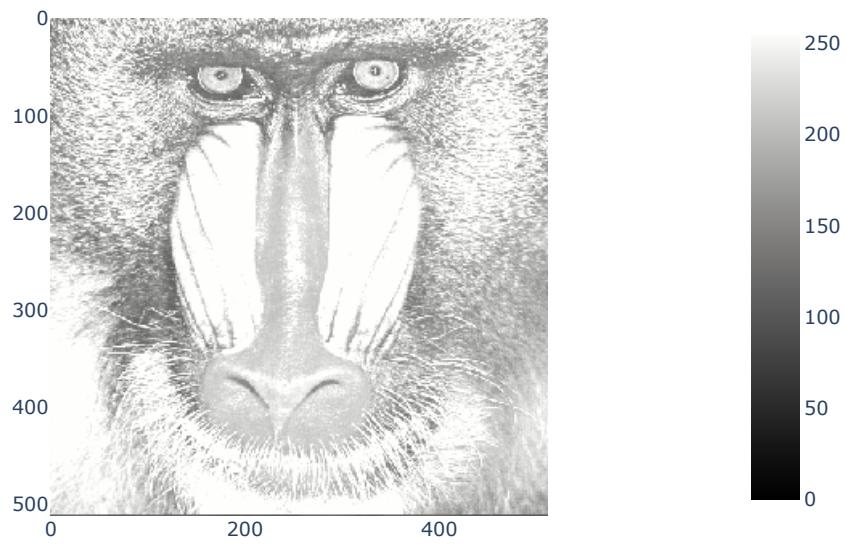
U većini slučajeva bi želeli da ipak imamo rezultat sačuvan u obliku bez ove modifikacije i za to je potrebno obezbediti da rezultat ima tip koji omogućuje predstavljanje ovih vrednosti. Promenu tipa nad numpy nizovima možemo lako uraditi pozivanjem `astype` metode i prosleđivanjem odgovarajućeg tipa.

```
[10]: print(tmp_array)
print(tmp_array.astype('float')+100)
```

```
[ 0 100 200]
[100. 200. 300.]
```

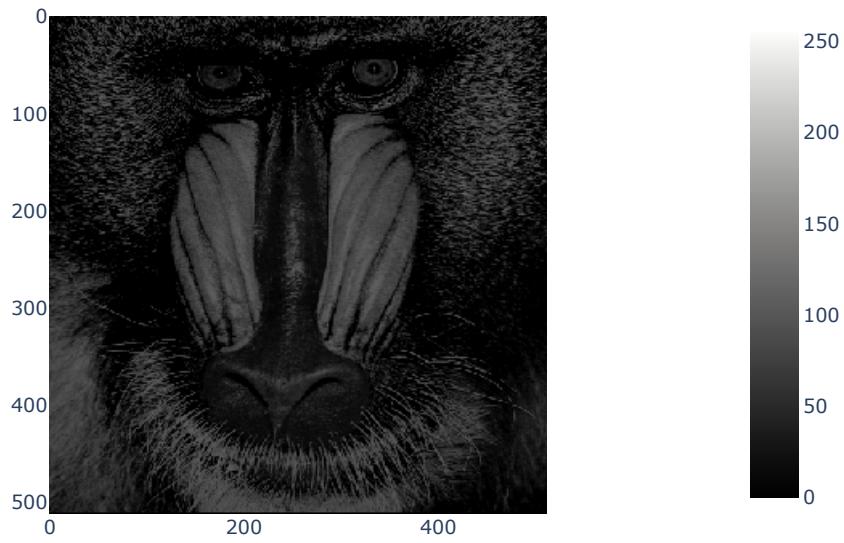
Ponovićemo prvi deo primera sa ovom izmenom.

```
[11]: img_2 = img.astype('float') + 100
fig = px.imshow(img_2, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



i drugi deo

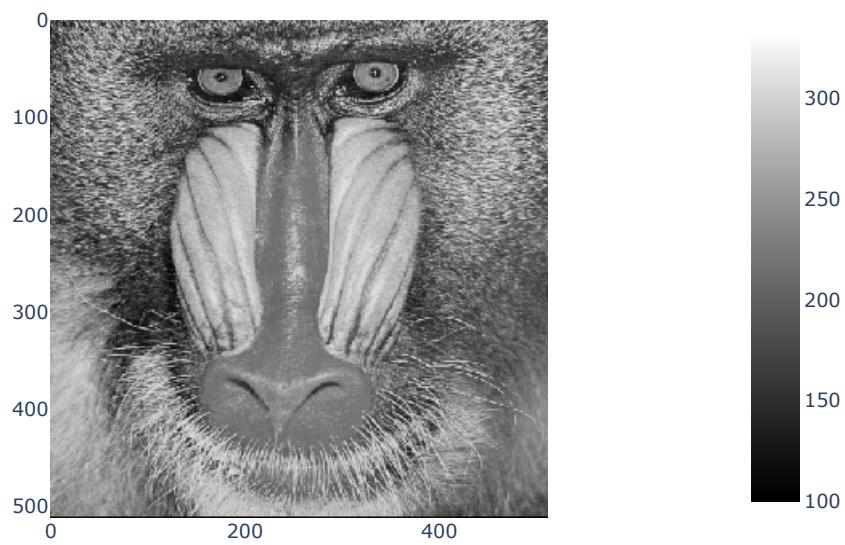
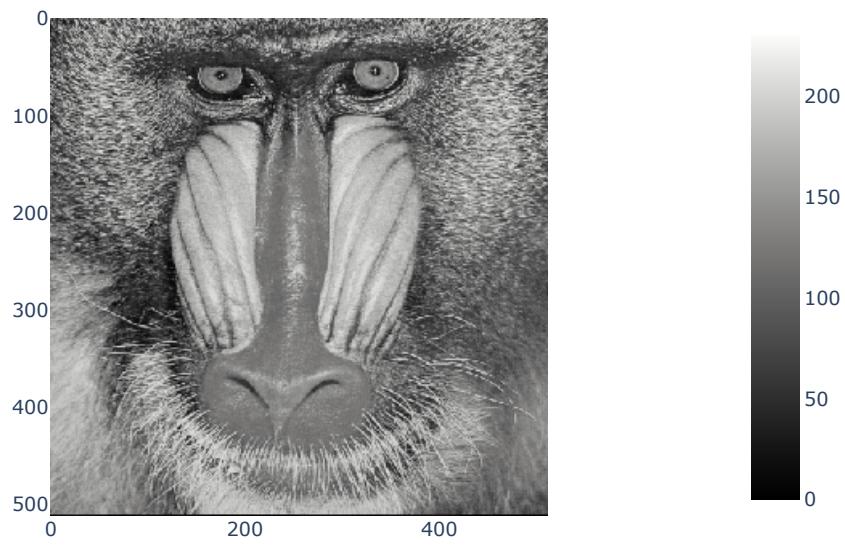
```
[12]: img_3 = img.astype('float') - 100
fig = px.imshow(img_3, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



U opštem slučaju parametri zmin i zmax funkcije imshow se ne moraju navesti. U tom slučaju će oni biti automatski postavljeni na opseg od min i max vrednosti date sive slike.

```
[13]: fig = px.imshow(img, color_continuous_scale='gray')
fig.show()

fig = px.imshow(img_2, color_continuous_scale='gray')
fig.show()
```



Primećujemo da je prikaz identičan u ovom slučaju iako su intenziteti piskela različiti.

7.3 Primer 2.3

U promenljivu `img` učitati sliku `zelda.png` i prikazati je u opsegu [0,255].

Promenljivoj slike normalizovati opseg vrednosti (preračunati ga na opseg [0, 1]) i prikazati je.

```
[14]: import plotly.express as px
from skimage import io

img = io.imread('zelda.png')
fig = px.imshow(img, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

img_norm = (img - img.min())/(img.max()-img.min())
fig = px.imshow(img_norm, zmin=0, zmax=1, color_continuous_scale='gray')
fig.show()
```





Normalizacija se lako postiže oduzimanjem minimalne vrednosti naše slike (dobijene pozivanjem metode `min` nad numpy nizom) i deljenjem sa razlikom između maksimalne vrednosti slike (dobijene pozivanjem metode `max` nad numpy nizom) i minimalne. Prilikom deljenja rezultat koji se formira automatski dobija tip vrednosti koji je u stanju da sačuva dobijene realne vrednosti (float).

```
[15]: img_norm.dtype
```

```
[15]: dtype('float64')
```

7.4 Primer 2.4

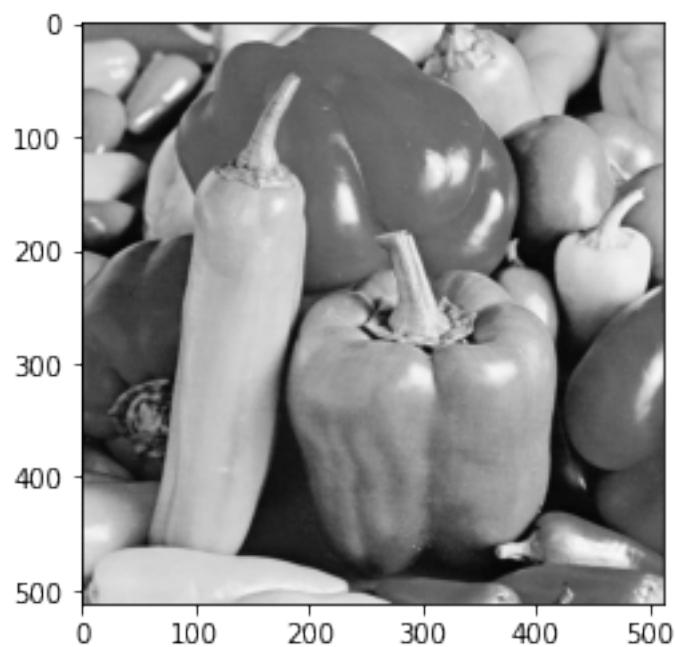
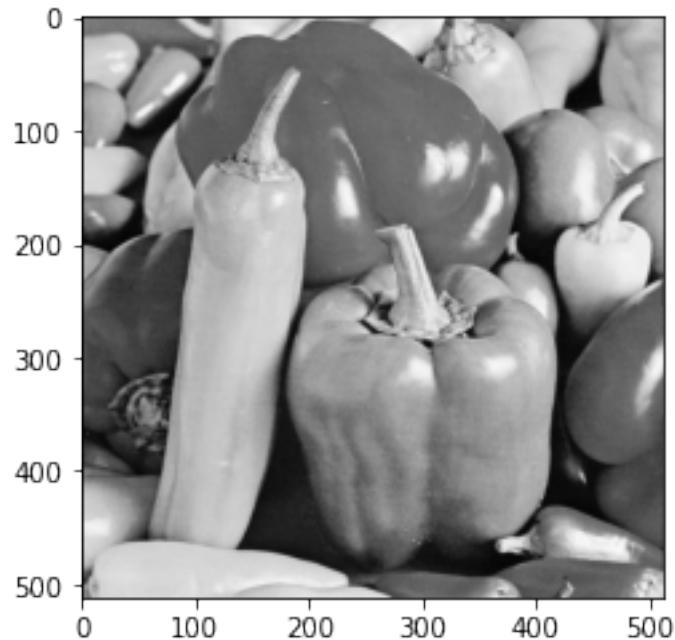
U promenljivu `img` učitati `peppers.png`. Učitanu sliku snimiti pod imenom `peppers2.png`. Snimljenu sliku učitati i prikazati.

```
[16]: import matplotlib.pyplot as plt
from skimage import io

img = io.imread('peppers.png')
plt.imshow(img, cmap='gray')
plt.show()

io.imsave('peppers2.png', img)
```

```
img2 = io.imread('peppers2.png')
plt.imshow(img2, cmap='gray')
plt.show()
```



Za snimanje slika možemo iskoristiti funkciju `imsave` u okviru `skimage` paketa.

7.5 Primer 2.5

U promenljivu `img` učitati `boat.png`. Iseći centralnih 50% slike i prikazati ih.

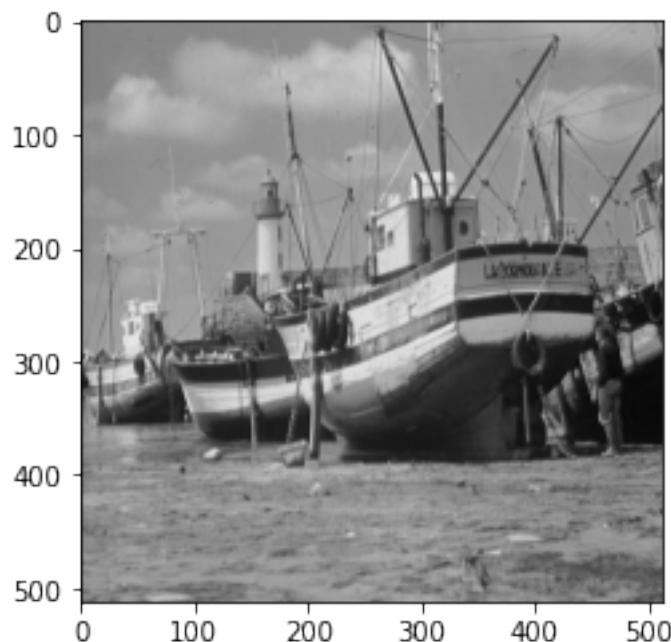
```
[17]: import matplotlib.pyplot as plt
from skimage import io

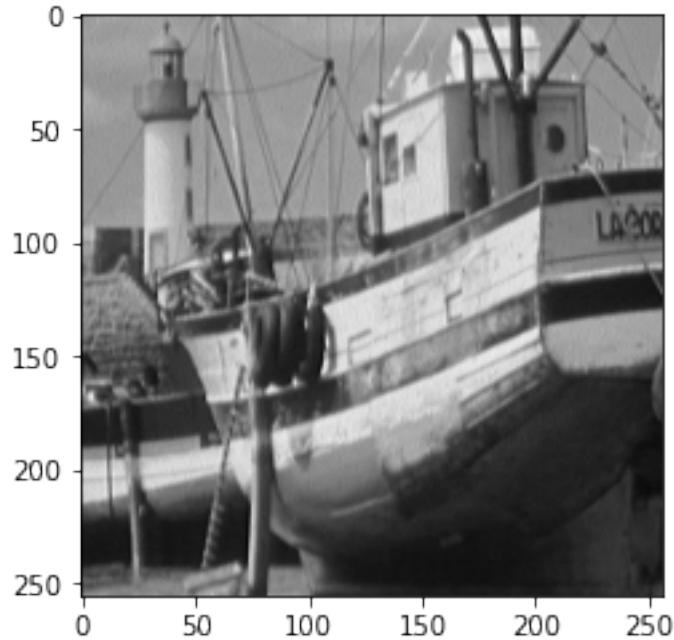
img = io.imread('boat.png')
plt.imshow(img, cmap='gray')
plt.show()

N, M = img.shape

img2 = img[round(N*0.25):round(N*0.75),
           round(M*0.25):round(M*0.75)]

plt.imshow(img2, cmap='gray')
plt.show()
```





Dimenziije slike mogu se odrediti kao i za bilo koju drugu matricu, posmatranjem vrednosti shape numpy niza. Centralni pikseli slike mogu se izdvajati indeksiranjem elemenata matrice img putem slicing-a. Vrednosti pritom trebaju da budu celobrojne, što se obezbeđuje funkcijom round. Ovde trebamo samo napomenuti da **isecanje nije vršilo kopiranje vrednosti originalne slike** za formiranje nove, već samo dalo nov način prikazivanja. Stoga bilo koja promena vrednosti u okviru img2 slike će se odraziti i na img sliku. Da bi se ovo izbeglo, gde je potrebno čuvati vrednosti originalne slike, potrebno je iskoristiti copy metodu prilikom formiranja nove slike.

```
[18]: #1 demonstracija izmene img2 slike kako utice na img_A sliku
img_A = img.copy()
N, M = img_A.shape

# mozemo iskoristiti i celobrojno deljenje u ovom slucaju
img2 = img_A[N//4:N*3//4,
              M//4:M*3//4]
img2[0:50, 0:50] = 0

#2 demonstracija izmene img3 slike kako ne utice na img_B sliku
img_B = img.copy()
N, M = img_B.shape

# mozemo iskoristiti i celobrojno deljenje u ovom slucaju
img3 = img_B[N//4:N*3//4,
              M//4:M*3//4].copy()
img3[0:50, 0:50] = 0
```

Primer subplot funkcionalnosti u okviru `matplotlib` paketa za kreiranje jednog figure prozora sa višestrukim iscrtavanjem grafika. Funkcija `subplot` prima prva dva argumenta kao broj vrsta i kolona u željenom grafiku, sa trećim parametrom kao redni broj grafika koji je u fokusu (za iscrtavanje). Samo iscrtavanje radi se na standardni način.

```
[19]: import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
plt.subplot(1,2,1)
plt.imshow(img_A, cmap='gray')
plt.subplot(1,2,2)
plt.imshow(img_B, cmap='gray')
plt.show()
```



Provera da li je neki numpy niz uvezan sa drugim može posmatranjem elementa `base` tog niza. Ispis `None` označava da samo ta promenljiva poseduje svoje vrednosti.

```
[20]: print('Test prve slike: {}'.format(img2.base))
print('Test druge slike: {}'.format(img3.base))
```

```
Test prve slike: [[127 123 125 ... 165 169 166]
 [128 126 128 ... 169 163 167]
 [128 124 128 ... 178 160 175]
 ...
 [112 112 115 ... 101 97 104]
 [110 112 117 ... 104 93 105]
 [113 115 121 ... 102 95 97]]
```

```
Test druge slike: None
```

7.6 Primer 2.6

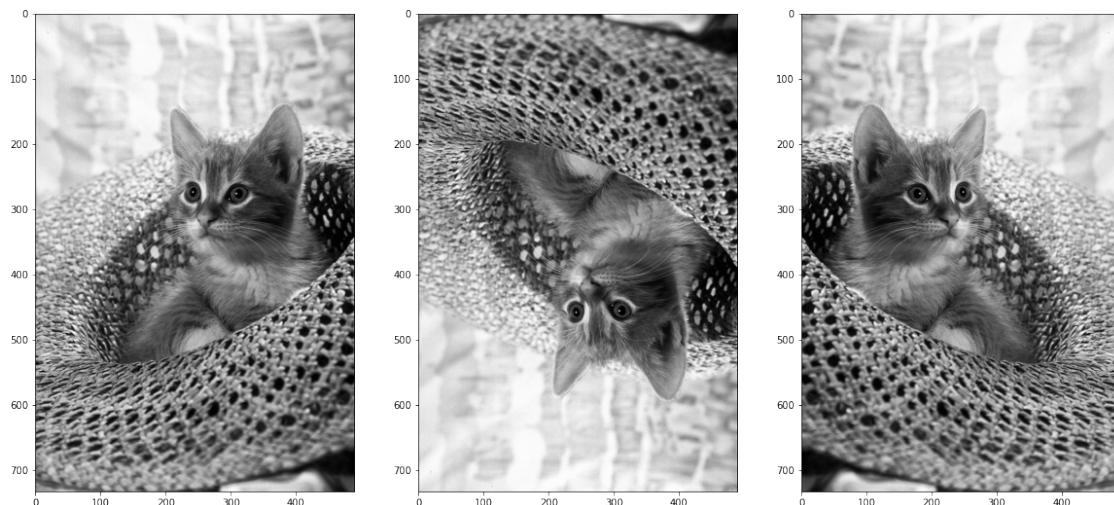
U promenljivu `img` učitati `cat.png`. Sliku prikazati preslikanu (okrenutu, engl. *flipped*) u odnosu na horizontalnu i vertikalnu osu simetrije.

```
[21]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np

img = io.imread('cat.png')

img_flipud = np.flipud(img)
img_fliplr = np.fliplr(img)

plt.figure(figsize=(20,10))
plt.subplot(1,3,1)
plt.imshow(img, cmap='gray')
plt.subplot(1,3,2)
plt.imshow(img_flipud, cmap='gray')
plt.subplot(1,3,3)
plt.imshow(img_fliplr, cmap='gray')
plt.show()
```



Numpy funkcija `flipud` koristi se za okretanje matrice od gore na dole, dok se funkcija `fliplr` koristi za okretanje sa leva na desno. Obe funkcije ne prave kopiju vrednosti.

8 Zadaci za samostalnu vežbu

Zadatak 2.1

Učitati i prikazati CT (engl. *computerized tomography*) snimak `ct_1.png`. Slika je tipa `uint16` (neoznačeni celobrojni tip dužine 16 bita) što znači da je opseg vrednosti piksela od 0 do 65535. Ovaj opseg se preslikava u više nijansi sive boje nego što ljudsko oko može da razlikuje. Međutim, preko podešavanja granica palete moguće je prikazati samo određene delove dinamičkog opsega slike čime se naglašavaju određene vrste tkiva prilikom prikaza. Prikazati sliku sa naglašenim koštanim tkivima podešavanjem granica palete na 3000 i 4000 i sa naglašenim mekim tkivima putem podešavanja granica palete na 2800 i 3150.

[22] : `#TODO`

Zadatak 2.2

Jedan od standardnih načina za interaktivno podešavanje kontrasta kod medicinskih snimaka je podešavanje dva parametra: prozora i nivoa (engl. *window and level*). Parametar nivo određuje vrednost koja će se prikazati kao srednja nijansa sive (na polovini paleta). Parametar prozor određuje širinu opsega koji će biti obuhvaćen paletom sivih tonova. Ako se vrednost parametra nivo obeleži sa L , a vrednost parametra prozor obeleži sa W , intenzitet koji predstavlja donju granicu palete će biti $L-W/2$, a nivo koji određuje gornju granicu će biti $L+W/2$. Napisati funkciju `setWindowLevel` koja prikazuje sliku sa paletom sivih tonova podešenom prema nivou i prozoru. Funkciju pozvati nad MRI (engl. *magnetic resonance imaging*) snimkom `mr_1.png`. Sliku prikazati sa parametrima koji ističu guste strukture, gde je $W = 3000$, a $L = 2500$, zatim sa parametrima koji ističu mekša tkiva, gde je $W = 2100$, a $L = 1450$.

[23] : `#TODO`

Zadatak 2.3

Polje zračenja se prilikom rendgenskih snimanja ograničava samo na anatomiju koju je potrebno snimiti, što za rezultat ima pojavu tamnih regiona u snimku. Ovi regioni predstavljaju suvišan deo snimka koji može da ometa dijagnostiku, te ih je nekada potrebno odstraniti. Učitati i prikazati neobrađene digitalne rendgenske snimke `rtg_1.png`, `rtg_2.png` i `rtg_3.png`. Sa učitanih snimaka ručno odstraniti delove koji ne predstavljaju direktno ozračenu anatomiju. Isečene delove prikazati i sačuvati kao nove slike sa proizvoljnim imenima.

[24] : `#TODO`

Zadatak 2.4

Različiti proizvođači medicinskih uređaja nude snimke koji imaju različite dinamičke opsege. Da bi se omogućila unificirana obrada snimaka, potrebno ih je dovesti na neki unapred poznati opseg.

Napisati funkciju `normalizeImRange` koja opseg ulazne slike preračunava na $[0, 1]$. Napisati funkciju `setRange` koja kao ulazne argumente prima minimalnu i maksimalnu vrednost opsega na koji treba preračunati opseg ulazne slike. Izlaz funkcije treba da bude slika sa podešenim opsegom. U razvoju funkcije `setRange` iskoristiti funkciju `normalizeImRange`. Ispravnost funkcije proveriti na nekoliko proizvoljnih slika (proveriti njihove nove `min` i `max` vrednosti).

[25] : `#TODO`

Zadatak 2.5

Prilikom akvizicije rendgenskih snimaka pluća pacijenti se pozicioniraju leđima prema izvoru zračenja i licem prema detektoru. Na ovako prikupljenim snimcima će desna strana pacijenta biti prikazana na desnoj strani snimka. Pošto je uobičajeno da je na desnoj strani snimka prikazana leva strana pacijenta, snimak je potrebno preslikati oko vertikalne ose simetrije. Napisati program koji ostvaruje navedeno preslikavanje bez upotrebe funkcije `fliplr`. Funkcionalnost programa isprobati na slici `rtg_4.png`.

[26] : `#TODO`

Zadatak 2.6

Učitati sliku `boat.png`. Iz učitane slike potrebno je izdvojiti centralnih 70% piksela kao novu sliku, povećati njihov intenzitet 2 puta i potom vratiti segment na originalnu poziciju. Vrednosti koji prelaze maksimalnu vrednost definisanu tipom originalne slike pravovati na nju.

[27] : `#TODO`

9 Dokumentacija novih celina

- [Anaconda](#)
- [Kreiranje okruženja](#)
- [Instalacija paketa 1](#)
- [Instalacija paketa 2](#)
- [PyCharm interfejs](#)
- [Debug](#)
- [Google Colab](#)
- [scikit-image](#)
- [plotly imshow](#)
- [matplotlib.pyplot imshow](#)
- [numpy indeksiranje](#)
- [numpy.stack](#)
- [plotly subplots](#)
- [matplotlib subplots](#)

- `numpy flipud/lr`
- `numpy copy vs view`

Vezba 3 - DOS osnove 2

1 Pregled

U okviru vežbe radi se upoznavanje sa amplitudskom rezolucijom i načinom njene promene kao i interpolacijom, operacijom koja se koristi da bi se promenio broj odbiraka koji predstavljaju sliku. U vežbi se porede efekti upotrebe različitih metoda interpolacije. Dodatno, prikazuju se mere rastojanja između piksela i način njihove vizuelizacije.

2 Amplitudska rezolucija

Amplitudska rezolucija se odnosi na mogućnost reprezentacije mogućih intenziteta u okviru slike. Obično se definiše sa brojem bita koje koristimo prilikom formiranja vrednosti intenziteta. Najuočljajena reprezentacija je sa 8 bita informacije sa kojima možemo prikazati 256 različitih nivoa, odnosno 2^8 . 16 bita za reprezentaciju nivoa koristi se za slike obično nekog medicinskog kataloga.

2.1 Primer 3.1

U promenljivu `img` učitati `head.png`. Prikazati učitanu sliku sa automatskim podešavanjem tonske skale. Odrediti kvantizovanu sliku koja predstavlja originalnu sliku sa smanjenim brojem mogućih nivoa intenziteta. Koristiti 3 bita informacije za određivanje kvantizovane slike i uniformno određene nivoe.

```
[1]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('head.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()

bit_num = 3
bin_string = '1'*bit_num + '0'*(8-bit_num) # automatsko određivanje stringa sa binarnim zapisom koristeći operacije nad listama
quant_mask = int(bin_string, 2)
print('Decimalni broj za kvantizacionu masku: {}'.format(quant_mask))
```

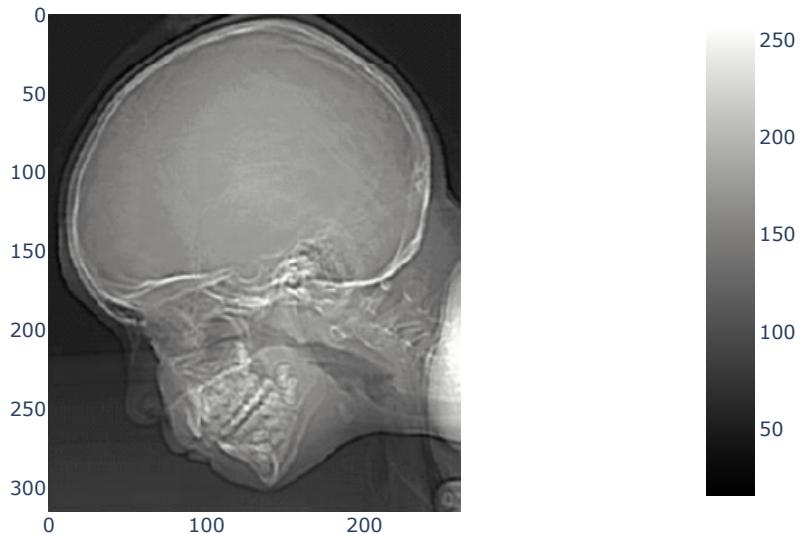
```

Q = img & quant_mask
Q = np.round(Q/quant_mask*255).astype('uint8') #skaliranje na opseg [0,255]

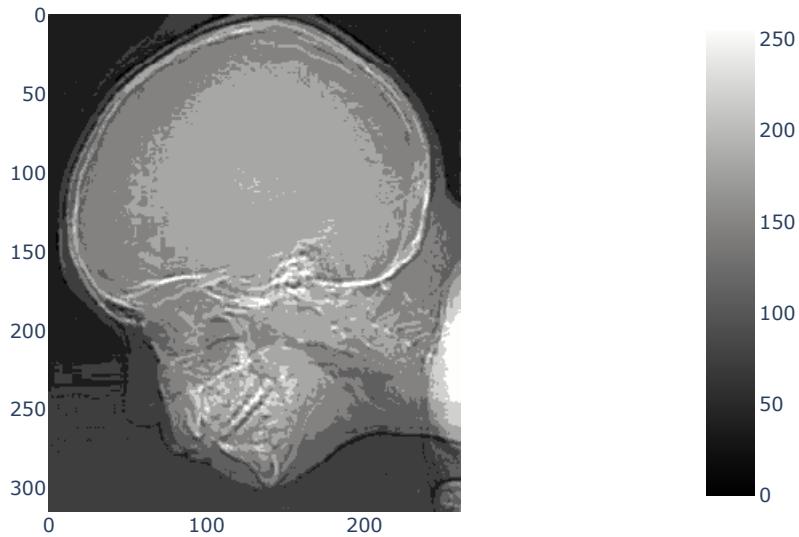
print('Unikatne vrednosti u okviru kvantizovane slike: {}'.format(np.unique(Q)))

fig = px.imshow(Q, color_continuous_scale='gray')
fig.show()

```

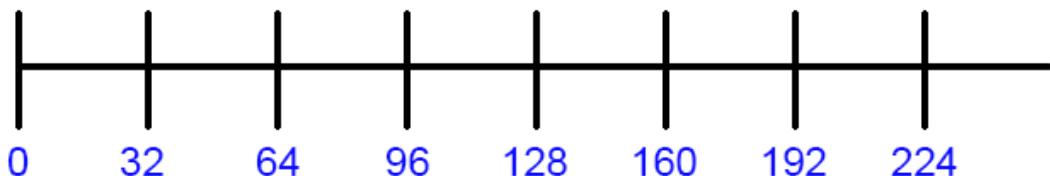


Decimalni broj za kvantizacionu masku: 224
 Unikatne vrednosti u okviru kvantizovane slike: [0 36 73 109 146 182 219 255]



Kvantizacija u ovom slučaju predstavlja način kako da ograničimo ulazni opseg od 256 mogućih nivoa intenziteta sivih tonova na neki oraničeni set vrednosti. Broj mogućih kvantizacionih nivoa se obično definiše sa korišćenim brojem bita kao $2^{\#bita}$.

U ovom primeru imamo 3 bita sa kojim formiramo 8 vrednosti na koje preslikavamo čitav ulazni opseg slike od 0 do 255. Za uniformnu kvantizaciju to znači da opseg [0,255] delimo na 8 podjenačih celina.



Pojedinačni intenziteti se mogu razdvojiti na ove celine tako što se posmatraju vrednosti formirane od njihovih 3 najznačajnija bita. To možemo ostvariti logičkom operacijom & sa maskom koja ima definisane logičke jedinice na prve 3 pozicije i 0 na ostalima, 1110 0000. Možemo iskoristiti python ugrađenu funkciju int za određivanje decimalne vrednosti ove binarno zapisane maske.

```
[2]: print(int('11100000',2))
```

224

Funkciji prosleđujemo string sa binarnim zapisom (bez razmaka) i parametar 2 da označimo bi-

narni sistem.

Logičkom operacijom pojedinačni intenziteti postaju preslikani na moguće realizacije ove operacije, a to su vrednosti

binarni broj	decimalna vrednost
0000 0000	0
0010 0000	32
0100 0000	64
0110 0000	96
1000 0000	128
1010 0000	160
1100 0000	192
1110 0000	224

koje upravo formiraju granice za preslikavanje na 8 podjenakih celina (od po 32 elementa) za opseg [0,255].

Obično se rezultujuće preslikane vrednosti dodatno koriguju množenjem sa konstantom, da bi se najsvetlijia komponenta ponovo prikazala kao vrednost od 255 (bela nijansa za 8bitni prikaz). Ta korekcija se lako postiže deljenjem sa decimalnom vrednosti korišćene maske (ona predstavlja novi dinamički opseg slike prilikom vršenja logičke operacije $\&$ - jer joj je to nova maksimalna vrednost, dok je minimalna 0) za normalizaciju na [0,1] opseg i množenjem sa 255 za povratak na [0,255] interval. U ovom kontekstu, možemo ostati i na normalizovanoj slici kao finalnoj. U svakom slučaju, rezultujuća slika predstavlja kvantizovanu na tačno 8 nivoa koji su maksimalno udaljeni jedan od drugog u cilju prikazivanja slike sa boljim kontrastom.

Ono što možemo uočiti prilikom smanjenja broja krišćenih nivoa je pojavljivanje tzv. **lažnih kontura** (konture koje fizički ne postoje). Ovo se javlja jer u opštem slučaju u okolini nekog piksela najčešće nalazimo njemu srodne (intenzitetski slične) piksele, tako da prilikom kvantizacije oni će kolektivno biti preslikani na zajednički nivo.

3 Interpolacija

Interpolacija prestavlja metodu sa kojom možemo izvršiti promenu broja piksela za reprezentaciju neke slike u cilju npr. uvećanja (engl. *zooming*) i smanjenja (engl. *shrinking*). U osnovi, obe metode predstavljaju neko odabiranje koje vršimo na slici koje se svodi na određivanje vrednosti inteziteta u prostoru *između* piksela.

3.1 Primer 3.2

U promenljivu `img` učitati sliku `baboon.png`. Upotrebom funkcije `rescale` iz `skimage.transform` modula veličinu slike umanjiti 4, 8 i 16 puta. Rezultat snimiti u promenljive `img_2a`, `img_3a` i `img_4a`. Korišćenjem funkcije `resize` iz `skimage.transform` uvećati ove slike na dimenziju originalne slike `img` i rezultate smestiti u `img_2b`, `img_3b` i `img_4b`. Prilikom korišćenja funkcije koristiti **bilinearnu** interpolaciju. Ispisati pojedinačne veličine dobijenih slika i prikazati finalne.

```
[3]: import matplotlib.pyplot as plt
from skimage import io
from skimage.transform import rescale, resize
import numpy as np

img = io.imread('baboon.png')
print('Dimenzija slike img: {}'.format(img.shape))

# 4x

img_2a = rescale(img, 1/4, order=1, preserve_range=True)
print('Dimenzija slike img_2a: {}'.format(img_2a.shape))
img_2b = resize(img_2a, img.shape, order=1, preserve_range=True)
print('Dimenzija slike img_2b: {}'.format(img_2b.shape))

# 8x

img_3a = rescale(img, 1/8, order=1, preserve_range=True)
print('Dimenzija slike img_3a: {}'.format(img_3a.shape))
img_3b = resize(img_3a, img.shape, order=1, preserve_range=True)
print('Dimenzija slike img_3b: {}'.format(img_3b.shape))

# 16x

img_4a = rescale(img, 1/16, order=1, preserve_range=True)
print('Dimenzija slike img_4a: {}'.format(img_4a.shape))
img_4b = resize(img_4a, img.shape, order=1, preserve_range=True)
print('Dimenzija slike img_4b: {}'.format(img_4b.shape))

plt.figure(figsize=(20,20))

plt.subplot(1,4,1)
plt.title('Originalna slika')
plt.imshow(img, cmap='gray')

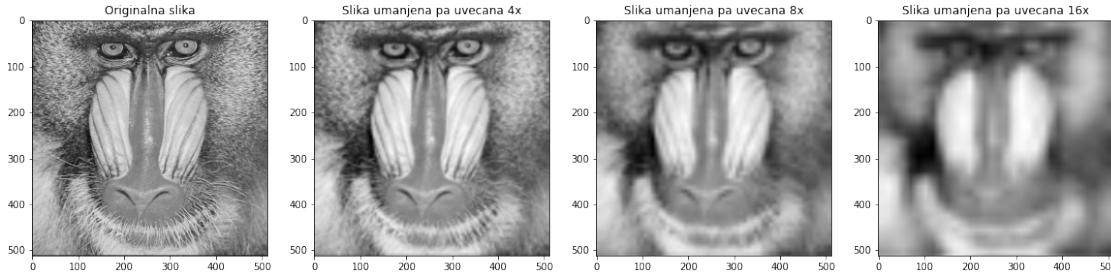
plt.subplot(1,4,2)
plt.title('Slika umanjena pa uvecana 4x')
plt.imshow(img_2b, cmap='gray')

plt.subplot(1,4,3)
plt.title('Slika umanjena pa uvecana 8x')
plt.imshow(img_3b, cmap='gray')

plt.subplot(1,4,4)
plt.title('Slika umanjena pa uvecana 16x')
plt.imshow(img_4b, cmap='gray')
```

```
plt.show()
```

```
Dimenzija slike img: (512, 512)
Dimenzija slike img_2a: (128, 128)
Dimenzija slike img_2b: (512, 512)
Dimenzija slike img_3a: (64, 64)
Dimenzija slike img_3b: (512, 512)
Dimenzija slike img_4a: (32, 32)
Dimenzija slike img_4b: (512, 512)
```



U okviru primjera koristimo dve funkcije za proces interpolacije. Funkcija `rescale` vrši promenu dimenzije slike spram prosleđenog parametra faktora skaliranja. Ukoliko je ta vrednost veća od 1 vrši se uvećanje toliko puta, u suprotnom se vrši smanjivanje. Funkcija `resize` vrši slično promenu dimenzije s tim da se u ovom slučaju kao drugi parametar prosleđuje vrednost željene dimenzije izlazne slike. Parametar `preserve_range` ostavlja originalni opseg vrednosti slike (u suprotnom pravi float sliku koja je u opsegu od 0 do 1) ali rezultat stavlja u tip koji odgovara za predstavljanje rezultata interpolacije. Parametar `order` označava koji tip interpolacije želimo da postignemo. Moguće vrednosti su:

tip interpolacije	vrednost
Nearest-neighbor (default for binary images)	0
Bi-linear (default for nonbinary images)	1
Bi-quadratic	2
Bi-cubic	3
Bi-quartic	4
Bi-quintic	5

Bilinearnom interpolacijom se vrednost piksela na nekoj lokaciji određuje na osnovu vrednosti četiri susedna piksela i rastojanja u odnosu na njih. Posledica bilinearne interpolacije je zamućenje slike.

4 Mere rastojanja piksela

Razlikujemo 3 osnovne mere rastojanja izmedju piksela p i z koji su određeni koordinatama (x, y) i (s, t) :

- Euklidsko
Formula: $D_e(p, z) = \sqrt{(x - s)^2 + (y - t)^2}$
- City-block
Formula: $D_4(p, z) = |x - s| + |y - t|$
- Chessboard
Formula: $D_8(p, z) = \max(|x - s|, |y - t|)$

Napomena: formule su ostavljene kao i na predavanjima, gde (x, y) predstavlja formu koordinata u obliku (*vrsta, kolona*). Podsetimo se samo, u okviru python jezika za analizu slika u većini biblioteka se obično vrste obeležavaju sa y a kolone sa x (što smo i videli prilikom korišćenja kurzora nakon prikazivanja slika). Naredan primer će smatrati takvu notaciju.

4.1 Primer 3.3

Ilustrovati Euklidska rastojanja piksela u odnosu na centralni piksel u slici dimenzije 41x41. Korištiti numpy funkciju `meshgrid` za određivanje pojedinačnih koordinata u matrici.

```
[4]: import plotly.express as px
import numpy as np

# Euklidsko

promene_po_row = np.arange(0, 41, 1)
promene_po_col = np.arange(0, 41, 1)

x, y = np.meshgrid(promene_po_col, promene_po_row)

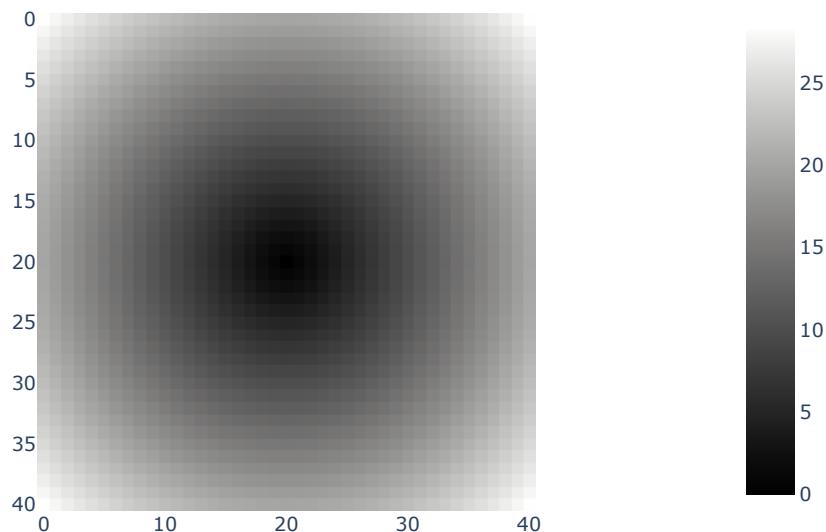
print(x)
print('---')
print(y)
print('---')
print('Koordinata centralnog piksela je: {}'.format((promene_po_row[20], promene_po_col[20])))

De = np.sqrt((x-20)**2 + (y-20)**2) # -20 je jer racunamo udaljenost centralnog piksela na koordinati (20,20) od svih ostalih

fig = px.imshow(De, color_continuous_scale='gray')
fig.show()
```

```
[[ 0  1  2 ... 38 39 40]
 [ 0  1  2 ... 38 39 40]
 [ 0  1  2 ... 38 39 40]
 ...
 [ 0  1  2 ... 38 39 40]
 [ 0  1  2 ... 38 39 40]
 [ 0  1  2 ... 38 39 40]]
```

```
---
[[ 0  0  0 ...  0  0  0]
 [ 1  1  1 ...  1  1  1]
 [ 2  2  2 ...  2  2  2]
 ...
 [38 38 38 ... 38 38 38]
 [39 39 39 ... 39 39 39]
 [40 40 40 ... 40 40 40]]
---
Koordinata centralnog piksela je: (20, 20)
```



Posmatramo udaljenost između centralnog piksela (koordinata (20,20)) i njegove okoline. Funkcija `meshgrid` formira matrice promene pojedinačnih koordinata u dimenziji koja odgovara proslēđenim promenama po kolonama i vrstama. Za dimenziju 41x41 ulazni parametri trebaju biti vektori sa vrednostima od 0 do 40 (ukupno 41 vrednost) što odgovaraju promenama po vrstama i kolonama od pozicije prvog piksela slike u gornje-levom čošku. Takve vektore možemo formirati koristeći numpy funkciju `arange` čiji ulaz prati formu početak,`kraj`,`korak` (vrednost `kraj` ne ulazi u rezultujući vektor). Alternativno isti vektor se može kreirati pozivom numpy funkcije `linspace` čiji ulazi su u formi `početak,kraj,broj` članova niza (u ovom slučaju `kraj` ulazi u okviru rezultata).

```
[5]: # formiranje niza od 2 do 7 sa korakom 1
print(np.arange(2, 8, 1))
```

```
print(np.linspace(2, 7, 6))
```

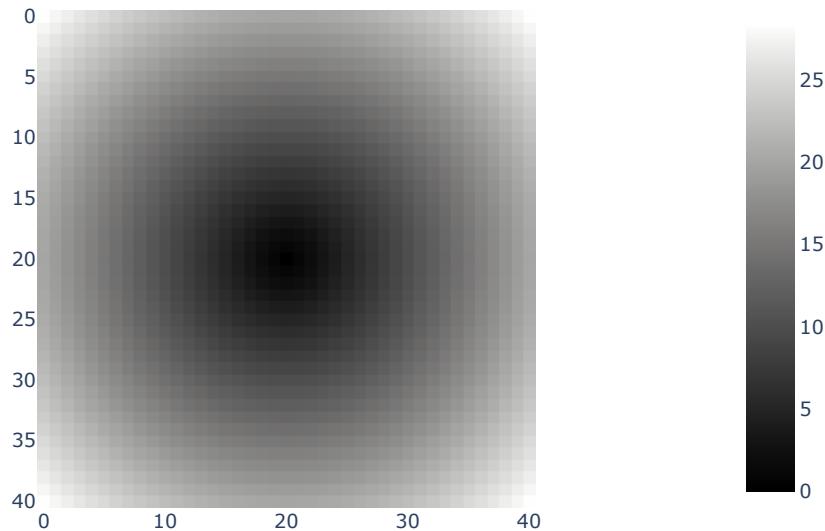
```
[2 3 4 5 6 7]  
[2. 3. 4. 5. 6. 7.]
```

Čitav primer možemo posmatrati i na drugi način. Možemo izmestiti koordinatni početak slike i smatrati da počinje u sredini slike. U tom slučaju moramo modifikovati promene po vrstama i kolonama tako da se vrednost 0 nađe na sredini. Koordinate pre ovog centralnog piksela imaće negativne vrednosti i poređenja će biti između centra (sa koordinatom (0,0)) i ostalih piksela.

```
[6]: import plotly.express as px  
import numpy as np  
  
# Euklidsko  
  
promene_po_row = np.arange(-20, 21, 1)  
promene_po_col = np.arange(-20, 21, 1)  
  
x, y = np.meshgrid(promene_po_col, promene_po_row)  
  
print(x)  
print('---')  
print(y)  
print('---')  
print('Koordinata centralnog piksela je: {}'.format((promene_po_row[20],  
→promene_po_col[20])))  
  
De = np.sqrt((x-0)**2 + (y-0)**2) # -0 je jer racunamo udaljenost centralnog  
→piksela koji je sada u koordinatnom pocetku (0,0)  
  
fig = px.imshow(De, color_continuous_scale='gray')  
fig.show()
```

```
[[[-20 -19 -18 ... 18 19 20]  
 [-20 -19 -18 ... 18 19 20]  
 [-20 -19 -18 ... 18 19 20]  
 ...  
 [-20 -19 -18 ... 18 19 20]  
 [-20 -19 -18 ... 18 19 20]  
 [-20 -19 -18 ... 18 19 20]]  
---  
[[[-20 -20 -20 ... -20 -20 -20]  
 [-19 -19 -19 ... -19 -19 -19]  
 [-18 -18 -18 ... -18 -18 -18]  
 ...  
 [ 18 18 18 ... 18 18 18]  
 [ 19 19 19 ... 19 19 19]  
 [ 20 20 20 ... 20 20 20]]
```

Koordinata centralnog piksela je: (0, 0)



4.1.1 Pogodnosti u numpy

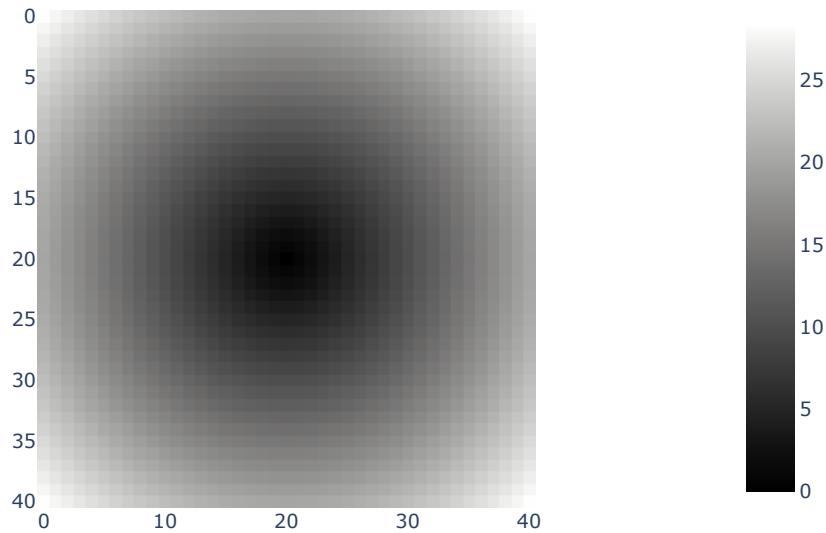
Prethodan način realizacije koordinata (sa `meshgrid` funkcijom) je prikazan radi jednostavnije interpretacije, međutim u okviru numpy biblioteke ove operacije se implicitno odvijaju u pozadini koristeći samo pravilno formirane (u formi vrste ili kolone) vektore za promene indeksa pojedinačnih dimenzija.

```
[7]: import plotly.express as px
import numpy as np

y = np.arange(-20, 21, 1).reshape(-1, 1) # podesavanje da vektor bude u obliku
                                             # kolone, drugi parametar je broj kolona, prvi parametar označava da se vrednost
                                             # prolagodi sama
x = np.arange(-20, 21, 1)

De = np.sqrt((x-0)**2 + (y-0)**2)

fig = px.imshow(De, color_continuous_scale='gray')
fig.show()
```



Metoda `reshape` omogućuje nam da oblikujemo numpy niz tako što prosledimo parametre željeneog `shape-a`. U ovom slučaju nam je potrebna vektor kolona, tako da je drugi parametar sigurno vrednost 1. Da ne moramo da pratimo koji ukupan broj članova (polje `size` za podsetnik) i da računamo preostali broj članova za drugu dimenziju, možemo postaviti -1 kao paramater što će automatski prilagoditi dimenziju nakon postavljenog drugog parametra.

4.2 Primer 3.4

Koristeći Euklidsko rastojanje od centralnog piksela slike `lena.png` kreirati masku gde je to rastojanje manje od polusirine slike. Izvršiti potom maskiranje.

```
[8]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('lena.png')

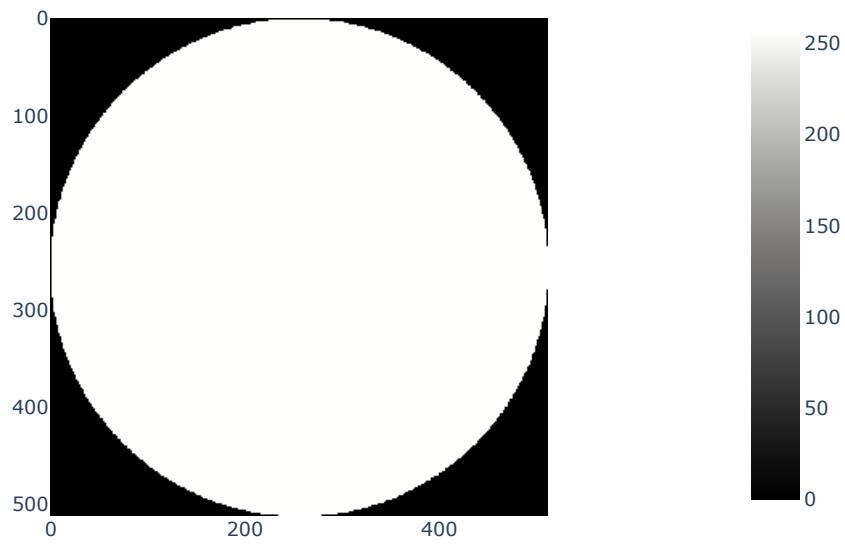
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[9]: y = np.arange(img.shape[0]).reshape(-1, 1) - np.floor(img.shape[0]/2)
x = np.arange(img.shape[1]) - np.floor(img.shape[1]/2)

De = np.sqrt((x-0)**2 + (y-0)**2)
M = De < np.floor(img.shape[0]/2)

fig = px.imshow(M, color_continuous_scale='gray')
fig.show()
```



```
[10]: # maskiranje mnozenjem  
img_masked1 = img * M  
  
fig = px.imshow(img_masked1, color_continuous_scale='gray')  
fig.show()
```



```
[11]: # maskiranje indeksiranjem
img_masked2 = img.copy() # kopiranje originalne slike
img_masked2[~M] = 0

fig = px.imshow(img_masked2, color_continuous_scale='gray')
fig.show()
```



5 Zadaci za samostalnu vežbu

Zadatak 3.1

U promenljivu `img` učitati sliku `baboon.png`. Upotrebom funkcije `rescale` iz `skimage.transform` modula veličinu slike umanjiti 4, 8 i 16 puta. Rezultat snimiti u promenljive `img_2a`, `img_3a` i `img_4a`. Korišćenjem funkcije `resize` iz `skimage.transform` uvećati ove slike na dimenziju originalne slike `img` i rezultate smestiti u `img_2b`, `img_3b` i `img_4b`. Prilikom korišćenja funkcije koristiti interpolaciju **najbližim susedom**. Ispisati pojedinačne veličine dobijenih slika i prikazati finalne.

Interpolacija najbližim susedom stvar **blok efekat** koji se ogleda u grupisanju piksela sa istom vrednošću u pravougaone blokove. Blok efekat nastaje jer se za interpoliranu vrednost uzima vrednost najbližeg suseda te dolazi do ponavljanja iste vrednosti, to jest stvaranja blokova.

[12] : `#TODO`

Zadatak 3.2

Pojedini proizvođači softvera za obradu i prikazivanje medicinskih snimaka nude mogućnost uvećanja samo određenog dela prikazanog snimka. Ovakav softverski alat uvećava region slike koji se nalazi na mestu određenom pokazivačem (kursorom, mišem). Napisati funkciju `magnifier` koja kao ulazne argumente prima originalnu sliku i koordinate pokazivača (proslediti kao tuple vrednost). Funkcija kao izlazni argument treba da vrati 4 puta uvećan region slike veličine 101 x 101

piksela sa centrom određenim koordinatama pokazivača.

Dodatak: razmisliti kako obezbediti da funkcija radi čak i ako oko pozicije pokazivača ne može da se pronađe kvadrat koji će u potpunosti biti ispunjen pikselima, npr. blizu ivica slike.

[13] : #TODO

Zadatak 3.3

Interaktivne transformacije slika mogu biti računarski zahtevne zbog potencijalno velikih dimenzija snimaka. Smanjenjem veličine originalnog snimka pojednostavljuje ovo računanje. Na smanjenom snimku se primenjuje ta transformacija, nakon čega se veličina snimka vraća na originalnu. Uvećanje slike se ostvaruje interpolacijom najbližim susedom koja nije računarski zahtevna operacija. Na ovaj način je transformisano manji broj piksela, čime je ostvarena ušteda na računarskim operacijama a mi opet dobijamo neku vizuelnu predstavu šta da očekujemo. Slika pune veličine se transformiše tek kada smo zadovoljni očekivanim rezultatom transformacije.

U okviru zadatka potrebno napisati funkciju `fastIntensityResPreview` koja kao ulazne argumente prima originalnu sliku i broj bita. Funkcija treba da smanji sliku 5 puta koristeći interpolaciju sa najbližim susedom, nakon čega je potrebno izvršiti kvantizaciju na broj nivoa koji definiše parametar broja bita. Rezultujuća slika se uvećava na originalnu dimenziju i prikazuje se.

Funkciju testirati nad proizvoljnim slikama i brojem bita za kvantizaciju (vrednost od 1 do 8).

[14] : #TODO

Zadatak 3.4

Vodeći se Primerom 3.3, odrediti vizuelan prikaz za City-block i Chessboard rastojanja u slici dimenzije 51x51. Za računanje apsolutne vrednosti koristiti np. `abs` funkciju, a za računanje maksimalne vrednosti koristiti np. `maximum`. Koje figure se mogu uočiti za ove tipove rastojanja?

[15] : #TODO

6 Dokumentacija novih celina

- `numpy.unique`
- `skimage.transform.rescale`
- `skimage.transform.resize`
- `numpy.arange`
- `numpy.linspace`
- `numpy.meshgrid`
- `numpy.ndarray.reshape`
- `numpy.abs`
- `numpy.maximum`

- [numpy.concatenate](#)
- [numpy.zeros](#)
- [numpy.pad](#)

Vezba 4 - Operacije na nivou piksela

1 Pregled

U okviru vežbe se radi upoznavanje sa operacijama na nivou intenziteta pojedinačnih piksela. U okviru vežbe su demonstrirani primeri upotrebe lukap (engl. *lookup*, LUT) tabela kako bi se izbeglo ponavljanje računarski kompleksnih operacija i time ubrzala obrada slika.

2 Operacije na nivou piksela

Ovakve operacije suštinski predstavljaju transformacije nad pojedinačnim intenzitetima slika i jedne su od najjednostavnijih operacija za obradu nad slikama. Mogu se predstaviti kao transformacija T nad nekom slikom f i posmatranim pikselom na (y, x) koordinati, tj. intenzitetom na toj lokaciji $f(y, x)$.

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

gde je $g(y, x)$ neka izlazna vrednost intenziteta.

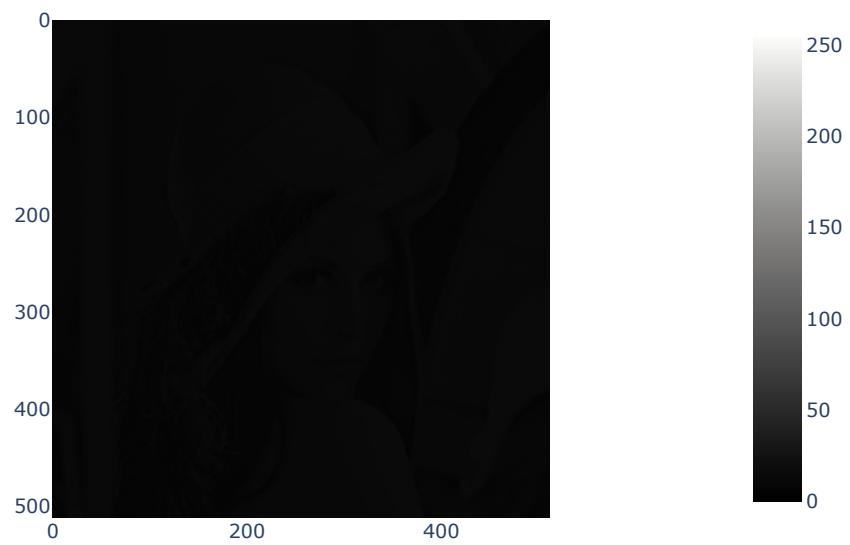
2.1 Primer 4.1

U promenljivu `img` učitati `lena.png`. Odrediti sliku koja nastaje određivanjem kvadratnog korena nad vrednostima piksela učitane slike. Slike prikazati u opsegu tonske skale [0,255].

```
[2]: import plotly.express as px
import matplotlib.pyplot as plt
from skimage import io
import numpy as np

img = io.imread('lena.png')
fig = px.imshow(img, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

img_sqrt = np.sqrt(img.astype('float'))
fig = px.imshow(img_sqrt, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



Rezultujuća slika je jasno tamnija, što je i za očekivati zbog računanja koren vrednosti. Međutim,

ovde bi trebali samo da primetimo način kako je formirana nova slika. Suštinski, idemo piksel po piksel iz originalne slike i računamo koren vrednost i rezultat postavljamo na izlaz. Za sliku koja je u ovom slučaju dimenzije 512x512 imamo ukupno 262144 piksela, što znači da smo isti taj broj računali koren vrednost. Ako se setimo, naša ulazna slika je tipa uint8 i ima opseg mogućih intenziteta od 0 do 255. Jasno je da će u ovom slučaju u okviru ukupnog broja piksela u slici biti ponovljenih računanja. Ovo može biti problematično ukoliko je naša funkcija transformacije veće kompleksnosti i zahteva duže izvršavanje. Što je naša slika veća u tom slučaju, duže će trajati ova obrada.

2.1.1 Lookup tabele (LUT)

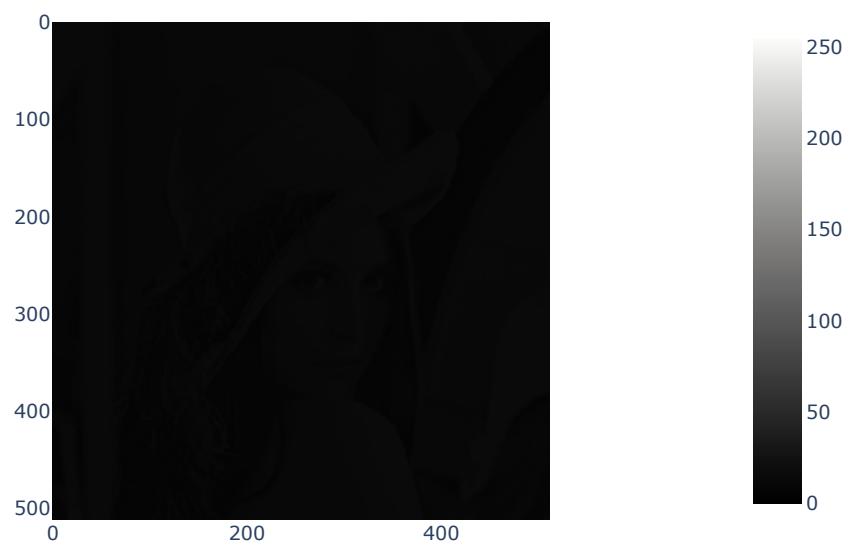
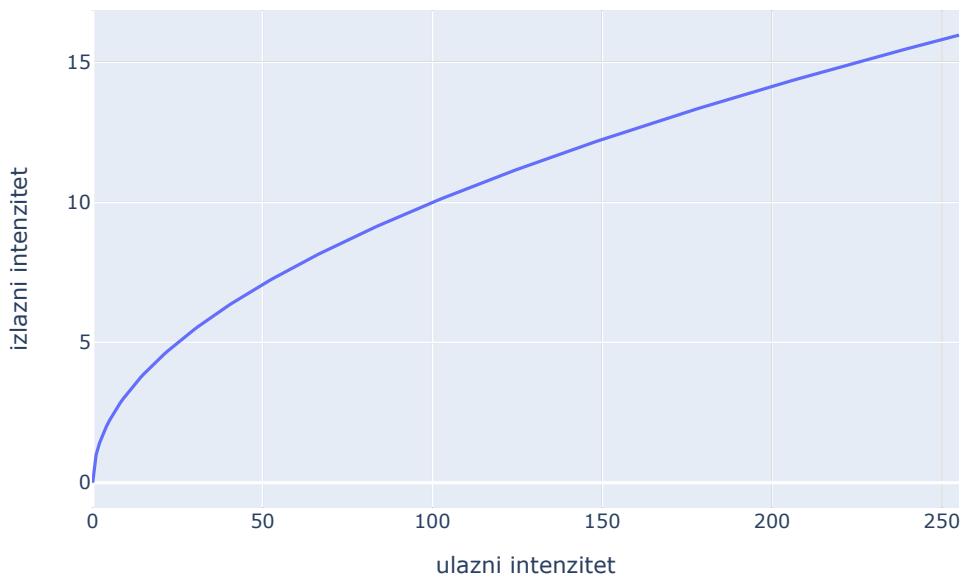
Kao rešenje navedenog problema postavljaju se tzv. lookup tabele. Upravo zbog ograničenja mogućih vrednosti ulaznog opsega slike može se kreirati tabela rezultata pojedinačnih transformacija po svim ulaznim vrednostima. Upotrebom LUT u ovom slučaju, kvadratni koren se određuje samo za vrednosti koje su u intervalu [0,255] tj. ukupno 256 puta. Kada je LUT određena, vrednost kvadratnog korena koji odgovara vrednosti nekog piksela iščitava se iz LUT korišćenjem same vrednosti piksela. Na ovaj način je izbegnuto određivanje kvadratnog korena 262144 puta, odn. za svaki piksel pojedinačno.

Za potrebe transformacija nad slikama koje imaju intenzitete od 0 pa naviše korišćenje LUT putem indeksiranja je moguće samo sa njenim intenzitetima, ali u opštem slučaju tabele se mogu formirati za bilo koji ulazni opsteg vrednosti uz dodatnu korekciju prilikom indeksiranja.

```
[3]: LUT = np.sqrt(np.arange(0,256))

# prikaz tabele, tj. transformacione krive koja izvrsava operaciju korenovanja
# →nad pojedinacnim intenzitetima
fig = px.line(x=np.arange(256), y=LUT)
fig.update_layout(xaxis_title="ulazni intenzitet", yaxis_title="izlazni intenzitet")
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'eraseshape']})

img_sqrt2 = LUT[img]
fig = px.imshow(img_sqrt2, zmin = 0, zmax = 255, color_continuous_scale='gray')
fig.show()
```



Tabelu možemo iscrtati koristeći line funkciju iz plotly.express modula. U ovom pozivu x i y

predstavljaju vrednosti koje želimo da prikažemo po odgovarajućim osama. Sa `update_layout` metodom mogu se dodatni parametri podešavati (kao što su npr. nazivi osa ili dimenzije prikaza).

Sam izlazni oblik rezultata transformacije određen je oblikom ulaza, kao što se može primetiti za prethodni primer za matrice. U slučaju da je skalar, na izlazu je isto skalar.

[4]: `print(LUT[10])`

```
3.1622776601683795
```

U slučaju da je vektor, na izlazu je isto vektor.

[5]: `print(LUT[[10,14,16]])`

```
[3.16227766 3.74165739 4.]
```

Napomena: tip izlazne slike će biti isti kao i tip u okviru elemenata LUT.

2.2 Primer 4.2

Napisati funkciju `gammaCorrection` koja implementira stepenu transformaciju nad slikama sa skaliranjem na opseg [0, 255], određenu parametrom `gamma`. Smatrati da je tip ulazne i izlazne slike `uint8`. U implementaciji koristiti LUT tabelu. Testirati funkciju nad slikom `undercurrent.jpg` i parametrom `gamma` od 0.5.

[6]: `import numpy as np`

```
def gammaCorrection(input, gamma):
    c = 255**(1-gamma)
    LUT = c * np.arange(0,256)**gamma
    LUT = np.round(LUT).astype('uint8')
    return LUT[input]
```

Gama korekcija suštinski predstavlja stepenu transformaciju slike sa dodatnom korekcijom opsega. U slučaju da je parametar za stepen `gamma`, novi opseg koja ima početni [0,255] je sada [0, $255^{1-\gamma}$]. Da bi se on korigovao na originalni opseg, rezultat stepene transformacije je potrebno reskalirati tako što je delimo sa $255^{1-\gamma}$ (na [0, 1] opseg) i množimo sa 255 (da bude [0, 255] opseg). Skraćeno ovo se može zapisati kao množitelj od $255^{1-\gamma}$. Dodatno, tip izlazne slike možemo da definišemo u okviru same LUT, tako što joj postavimo direktno željeni tip.

Izgled rezultata gama korekcije zavisi od postavljanja parametra `gamma`.

[7]: `import plotly.express as px
from skimage import io`

```
img = io.imread('undercurrent.jpg')
fig = px.imshow(img, zmin = 0, zmax = 255, color_continuous_scale='gray')
fig.show()
```

```
img_2 = gammaCorrection(img, 0.5)

fig = px.imshow(img_2, zmin = 0, zmax = 255, color_continuous_scale='gray')
fig.show()
```





Radi ilustracije, ovde su predstavljene i mogućnosti plotly paketa za interaktivan prikaz putem slajdera. Modifikovaćemo funkciju samo da dodatno vraća i samu LUT.

```
[8]: import numpy as np

def gammaCorrection(input, gamma):
    c = 255*(1-gamma)
    LUT = c * np.arange(0,256)**gamma
    LUT = np.round(LUT).astype('uint8')
    return LUT[input], LUT
```

```
[9]: import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Create figure - zelimo da prikazemo rezultat i LUT
fig = make_subplots(rows=1, cols=2)

gamma_range = [0.2, 0.4, 0.67, 1, 1.5, 2.5, 5.0]

# Add traces, one for each slider step
for gamma in gamma_range:
    img_tmp,LUT = gammaCorrection(img, gamma)
    fig.add_trace(go.Image(z=np.stack((img_tmp, img_tmp, img_tmp),axis=2), name="gamma="+str(gamma), visible=False), row=1, col=1)
```

```

fig.add_trace(go.Scatter(x=np.arange(256), y=LUT,
                         mode='lines', line=dict(color='blue'), visible=False), row=1, col=2)

# Make the 4th pair visible
active_idx = 3
fig.data[2*active_idx].visible = True
fig.data[2*active_idx+1].visible = True

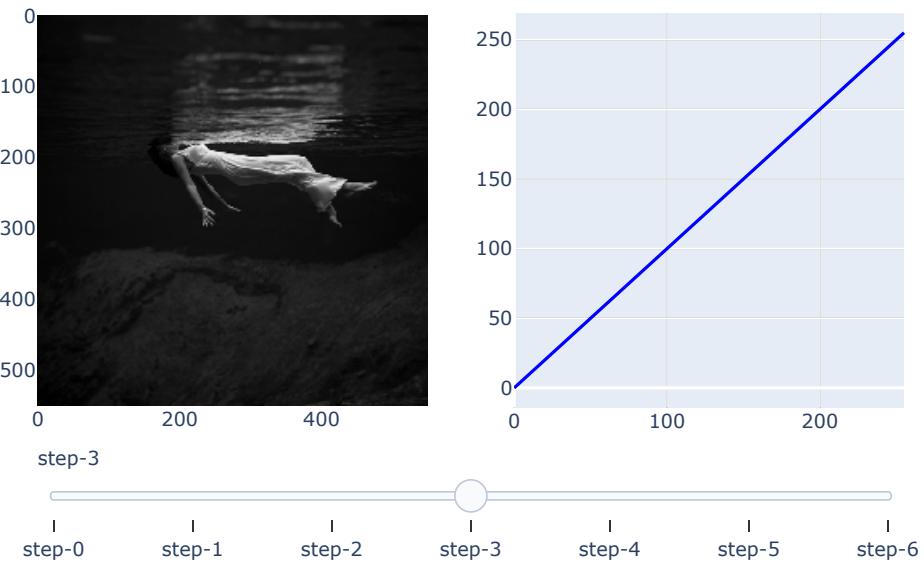
# Create and add slider
steps = []
for i in range(len(fig.data)//2):
    step = dict(method="update",
                args=[{"visible": [False] * len(fig.data)},
                      {"title": "Gamma value: " + str(gamma_range[i])}])

    step["args"][0]["visible"][2*i] = True # Toggle i'th trace to "visible"
    step["args"][0]["visible"][2*i+1] = True
    steps.append(step)

sliders = [dict(active=active_idx, steps=steps)]

fig.update_layout(sliders=sliders)
fig.show()

```



Za obezbeđivanje funkcionalnosti slajdera potrebno je koristiti glafičke objekte plotly paketa. Tabelu u ovom slučaju možemo iscrtati koristeći Scatter objekat. x i y predstavljaju vrednosti koje želimo da prikažemo po odgovarajućim osama a method specificira koji tip prikaza koristimo. Dodatno podešavanje je u okviru line parametra gde možemo u obliku ključ:vrednost rečnika navesti druge osobine, kao što je opcija za boju linije putem color ključa.

2.3 Primer 4.3

Odrediti negativ slike boat.png koristeći LUT.

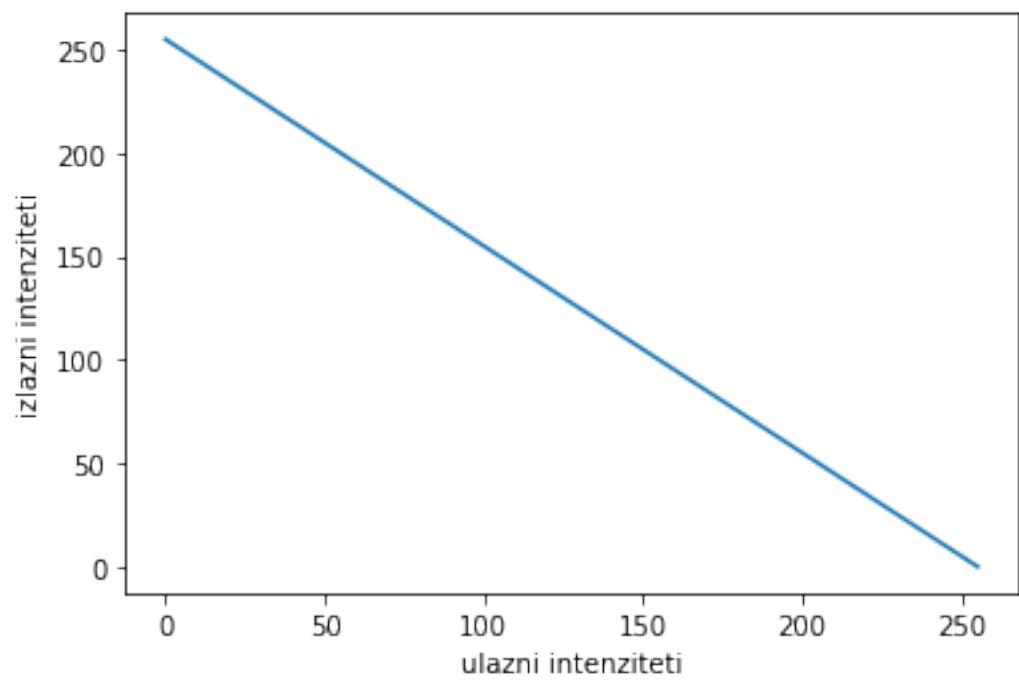
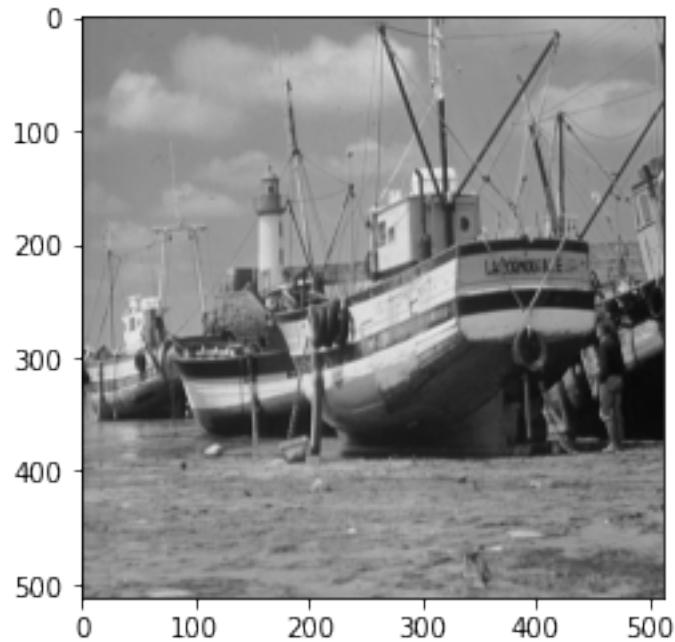
```
[10]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np

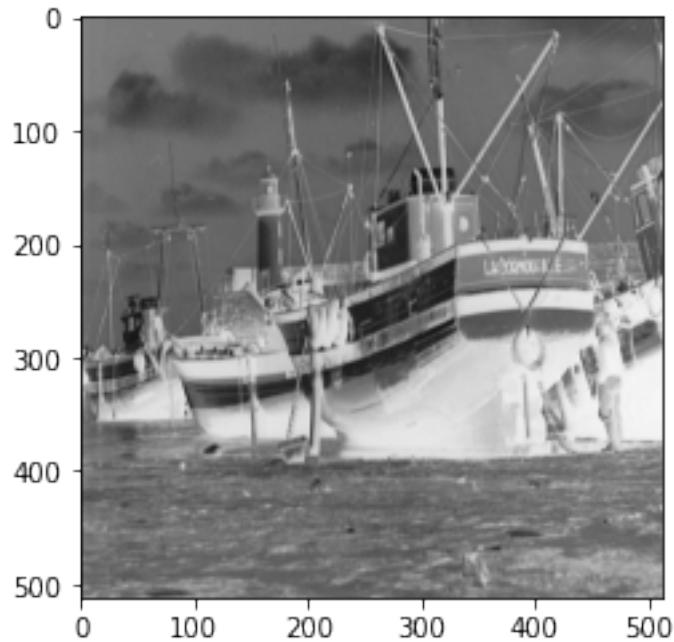
img = io.imread('boat.png')
plt.imshow(img, cmap='gray')
plt.show()

LUT = 255 - np.arange(0,256) # ili jednostavno np.arange(255,-1,-1)

plt.plot(np.arange(256),LUT)
plt.xlabel('ulazni intenziteti')
plt.ylabel('izlazni intenziteti')
plt.show()

img_neg = LUT[img]
plt.imshow(img_neg, cmap='gray')
plt.show()
```





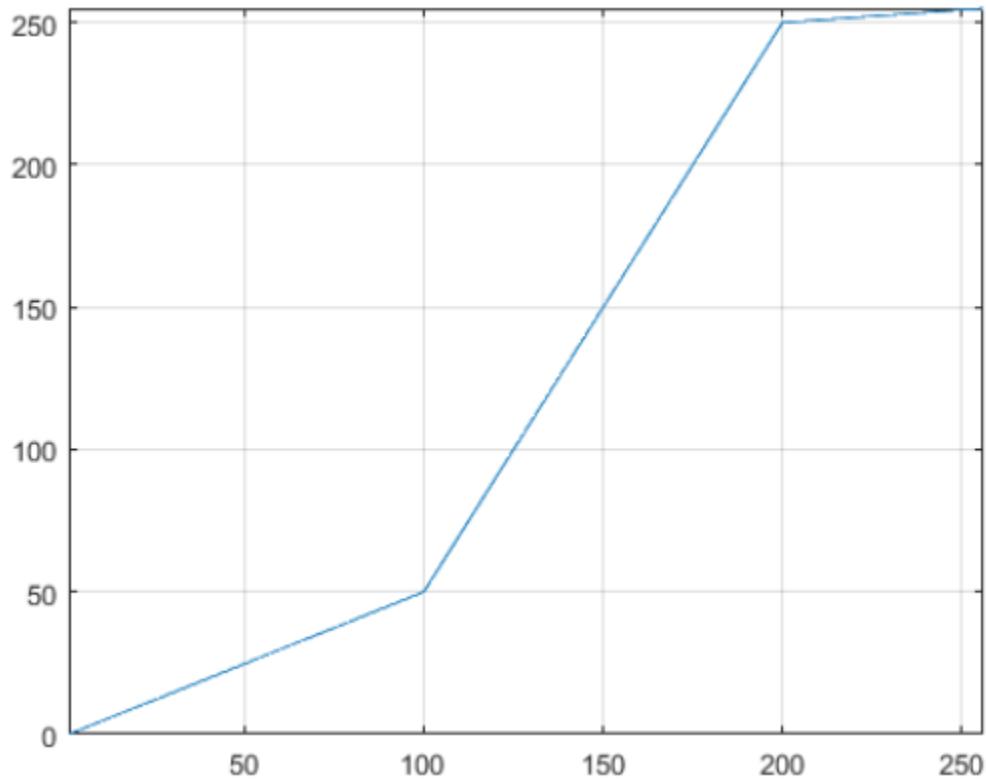
Određivanje negativa podrazumeva „obrtanje“ vrednosti piksela: pikselima sa malim vrednostima se dodeljuju velike, a pikselima sa velikim se dodeljuju male vrednosti. Operacija se jednostavno implementira oduzimanjem vrednosti piksela od najveće moguće vrednosti u slici.

Iscrtavanje LUT je ovde urađenjo korišćenjem `plot` funkcije u okviru matplotlib paketa.

2.4 Primer 4.4

Nad slikom `cat.png` upotrebiti deo-po-deo linearnu transformaciju. Transformacija je određena parovima (`ulaz,izlaz`).

$(0,0), (100,50), (200,250), (255,255)$



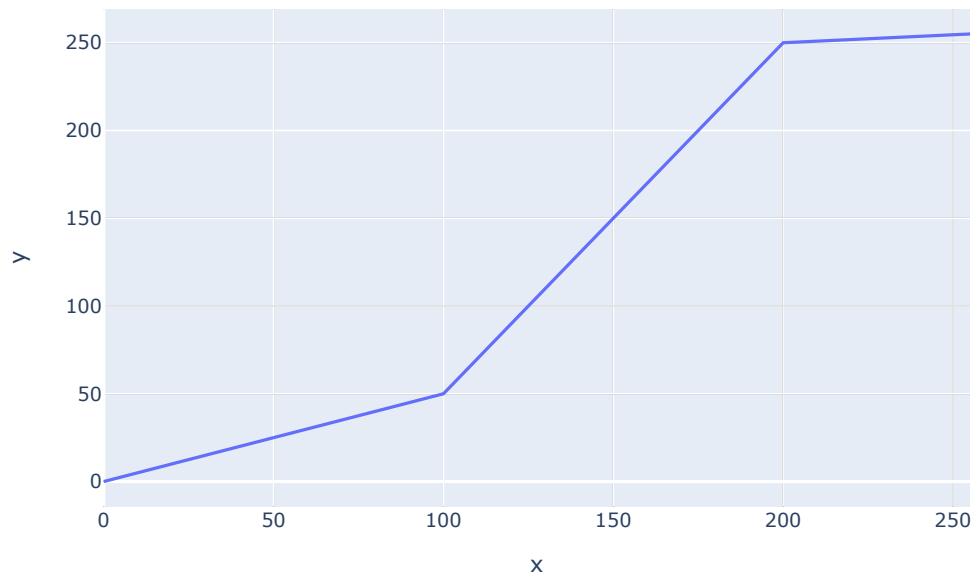
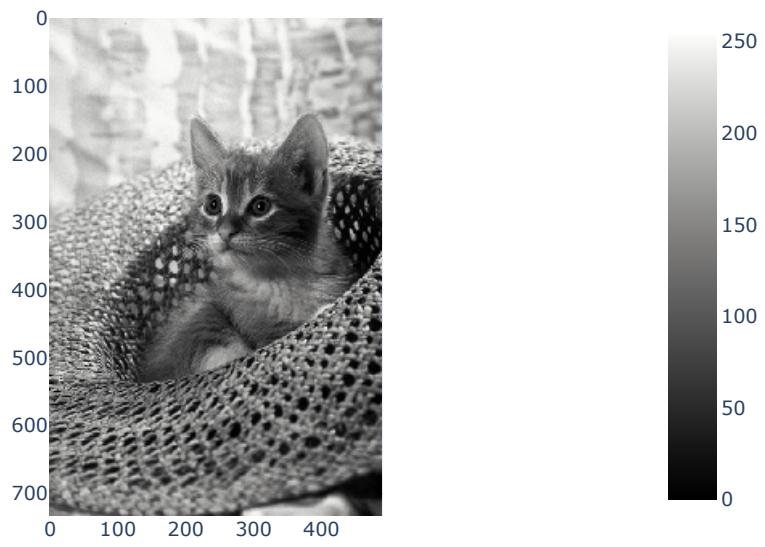
```
[11]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np

img = io.imread('cat.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()

lut1 = np.linspace(0,50,101)
lut2 = np.linspace(50,250,101)
lut3 = np.linspace(250,255,56)
LUT = np.concatenate((lut1[:-1], lut2[:-1], lut3), axis=0)

fig = px.line(x=np.arange(256), y=LUT)
fig.update_layout(width=500,height=500)
fig.show()

img_2 = LUT[img]
fig = px.imshow(img_2, color_continuous_scale='gray')
fig.show()
```





U ovom primeru prikazujemo da transformacija ne mora biti sastavljena od jednoznačne funkcije. Ovde vršimo transformaciju pojedinih podopsega ulaza u pojedine izlazne opsege. U ovom primeru je predstavljena **linearna** transformacija za pojedine delove, ali u opštem slučaju možemo definisati različite oblike. Jedino ograničenje je da je preslikavanje u pitanju "[na](#)".

Jednostavan način realizacije deo-po-deo linearne transformacije je preko LUT tabele određene upotrebom numpy funkcije `linspace`. Pošto funkcija definiše niz elemenata od početnog do krajnje prosleđenog paramatra, treba samo da se obrati pažnja šta se dešava u okviru spojenih celina. Da bi se izbeglo ponavljanje elemenata, vektori koji imaju spojene delove bi trebalo da se gledaju sa jednim manje elementom. Samo spajanje može se uraditi koristeći numpy funkciju `concatenate`, koja spaja numpy nizove po dimenzijama u okviru kojih su definisani.

3 Zadaci za samostalnu vežbu

Zadatak 4.1

U okviru formiranja radiografskih slika vrši se merenje slabljenja X-zraka pri prolasku kroz neki objekat do detektora. Prilikom prolaska zraka kroz neki objekat, dolazi do slabljenja i proces dobijanja izmenere količine zraka I možemo da napišemo kao

$$I = I_0 e^{- \int_{\forall i} \mu_i d_i}$$

gde je I_0 ulazni snop zraka a μ i d parametri slabljenja po jedinici debljine, i debljina celina kroz koje zrak prolazi. Ako prepostavimo da su ove celine homogene imamo

$$I = I_0 e^{-\sum_j \mu_j d_j}$$

Da bi se prikazao koristan deo osvetljaja i da bi se dobila linearna proporcija između I i debljine (količine) pojedinačnih celina, potrebno je izvršiti korekciju koristeći logaritam.

$$\log(I) = \log(I_0) - \sum_j \mu_j d_j$$

Ovakva korekcija za posledicu ima isticanje gustih anatomija, poput kostiju, koje bi bez korekcije bile slabo vidljive. Tradicionalni izgled snimaka (crna pozadina, belo koštano tkivo) se postiže dodatnom modifikacijom za negativ.

Napisati funkciju `correctUnprocessed` koja logaritamski koriguje dinamički opseg rendgenskog snimka i određuje negativ koristeći jednu LUT. Opseg vrednosti izlazne slike treba da bude [0, 255]. Funkciju testirati na slici `rtg_3.png`.

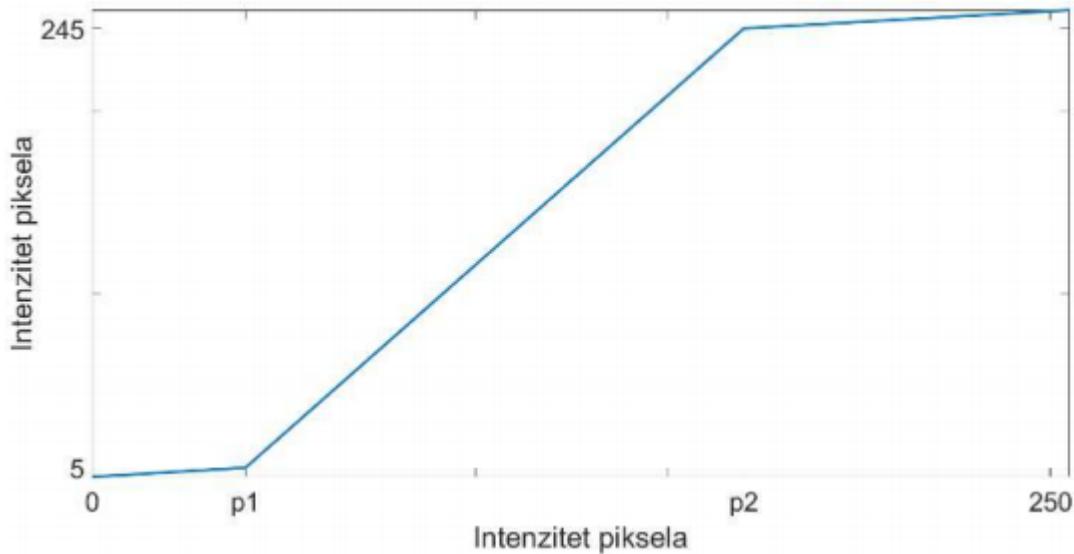
Napomena: zbog veličine slike koristiti `matplotlib` paket za prikaz.

Napomena: pošto pikseli mogu da imaju vrednost nula, pre određivanja logaritma svakom pikselu se vrednost povećava za 1. Na ovaj način je obezbeđeno da su vrednosti logaritma definisane (za vrednost 0 operacija logaritam nije definisana) i nenegativne (za vrednosti veće od 1 operacija logaritam vraća vrednosti koje su veće ili jednake od 0). Koristiti `log` funkciju u okviru `numpy` paketa za računanje prirodnog logaritma.

[12] : `#TODO`

Zadatak 4.2

Linearu deo-po-deo transformaciju moguće je upotrebiti da bi se na mamogramima naglasile strukture određene gustine. Napisati funkciju `tissueEmphasis` koja implementira linearu deo-po-deo transformaciju prikazanu na slici. Funkcija prima parametre $p1$ i $p2$ koji definišu njen izgled.



Funkciju upotrebiti na slici `mam_1.png`. Pronaći parametre p_1 i p_2 koji bi istakli žlezdano tkivo.

Napomena: zbog veličine slike koristiti `matplotlib` paket za prikaz.

[13] : `#TODO`

Zadatak 4.3

Prepraviti funkciju `gammaCorrection` tako da može da se koristi za obradu snimaka `uint16` tipa.

[14] : `#TODO`

4 Dokumentacija novih celina

- `plotly line plot`
- `plotly podešavanje dimenzije prikaza`
- `dictionary`
- `plotly simple slider`
- `numpy.concatenate`
- `numpy.log`

Vezba 5 - Histogram

1 Pregled

U okviru ove vežbe obrađuju se načini za određivanje histograma. Prikazani su primeri histograma koji odgovaraju različitim slikama i analizirana je veza između izgleda slika i oblika njima odgovarajućih histograma. Analizirana je transformacija vrednosti piksela ekvalizacijom histograma i njen uticaj na izgled rezultujućih slika.

2 Histogram slike

Za sliku koja ima L različitih nivoa (obično ograničeno tipom slike), neka r_k predstavlja intenzitet na k -tom nivou, gde je $k = 0, 1, \dots, L - 1$.

Histogram slike se definiše kao:

$$h(r_k) = n_k, \quad k = 0, 1, \dots, L - 1$$

gde je n_k broj piksela u slici sa intenzitetom r_k .

2.1 Primer 5.1

Odrediti koliko piksela slike `zelda.png` ima vrednost 50.

```
[2]: from skimage import io

img = io.imread('zelda.png')

# jedan nacin, iteracije po vrstama
counter1 = 0
for row in img:
    for pixel_intensity in row:
        if pixel_intensity == 50:
            counter1+=1
print(counter1)

# drugi nacin, iteracije po indeksima slike
counter2 = 0
```

```

for y in range(img.shape[0]):
    for x in range(img.shape[1]):
        if img[y,x] == 50:
            counter2+=1
print(counter2)

# treci nacin, iteracije po vektorizovanoj slici
counter3 = 0
for pixel_intensity in img.ravel():
    if pixel_intensity == 50:
        counter3+=1
print(counter3)

# cetvrti nacin, koriscenjem operacija na nivou citave matrice
counter4 = (img==50).sum()
print(counter4)

```

1526
1526
1526
1526

Ovaj problem se može rešiti koristeći brojačku pomoćnu promenljivu i prolazak po pojedinačnim elementima slike. Ukoliko je odgovarajući piksel sa intenzitetom 50, brojač se inkrementira za 1. Ovde su prikazana 3 načina za prolazak po pojedinačnim pikselima slike: 1. Prva petlja koja ide po vrstama i unutrašnja koja ide po elementima tih vrsta. 2. Prva petlja koja ide po indeksima vrsta i unutrašnja koja ide po indeksima kolona. 3. Prolazak kroz sve elemente koristeći `ravel()` metodu koja vraća *view* od slike u obliku 1D vektora.

Drugi način bi bio korišćenjem osobina numpy nizova. Ukoliko postavimo uslov jednakosti na nivou čitave slike, kao rezultat dobijamo binarnu masku dimenzije učitane slike koja ima vrednosti True na pikselima gde je ovaj uslov zadovoljen a False inače. Metoda `sum()` vraća sumu po svim elementima numpy niza, s tim da vrši implicitnu konverziju u slučaju logičkog tipa, tako da sumiranjem ove binarne maske sumiramo suštinski vrednosti jedinica u njoj.

2.2 Primer 5.2

Odrediti koliko piksela slike `undercurrent.jpg` ima vrednost 0, 1, ..., 255. Dobijeni rezultat prikazati grafički.

```
[3]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np

img = io.imread('undercurrent.jpg')

plt.figure(figsize=(10,5))
```

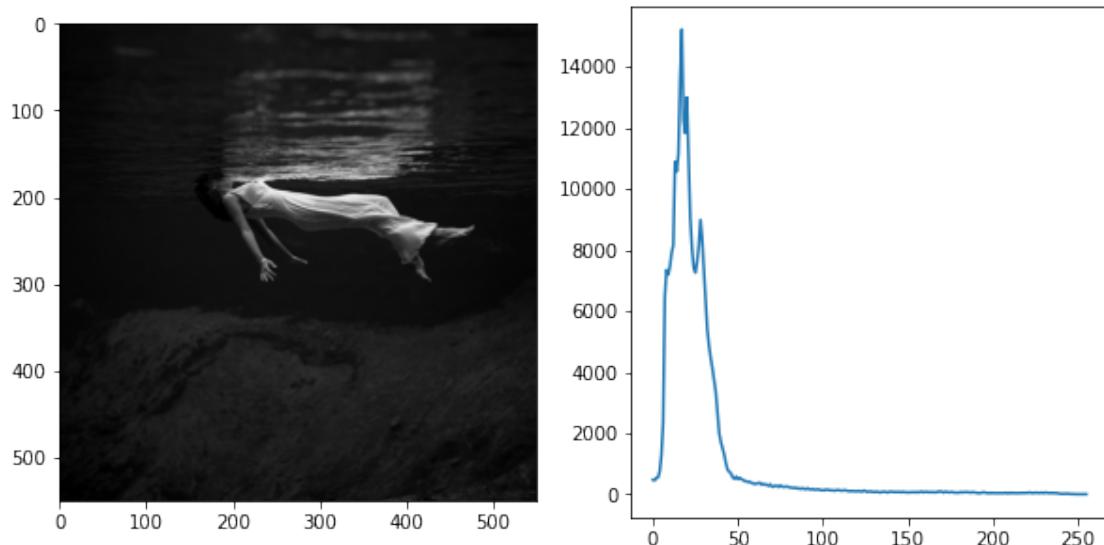
```

plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')

h = np.zeros(256)
for pixel_intensity in img.ravel():
    h[pixel_intensity] += 1

plt.subplot(1,2,2)
plt.plot(np.arange(256), h)
plt.show()

```



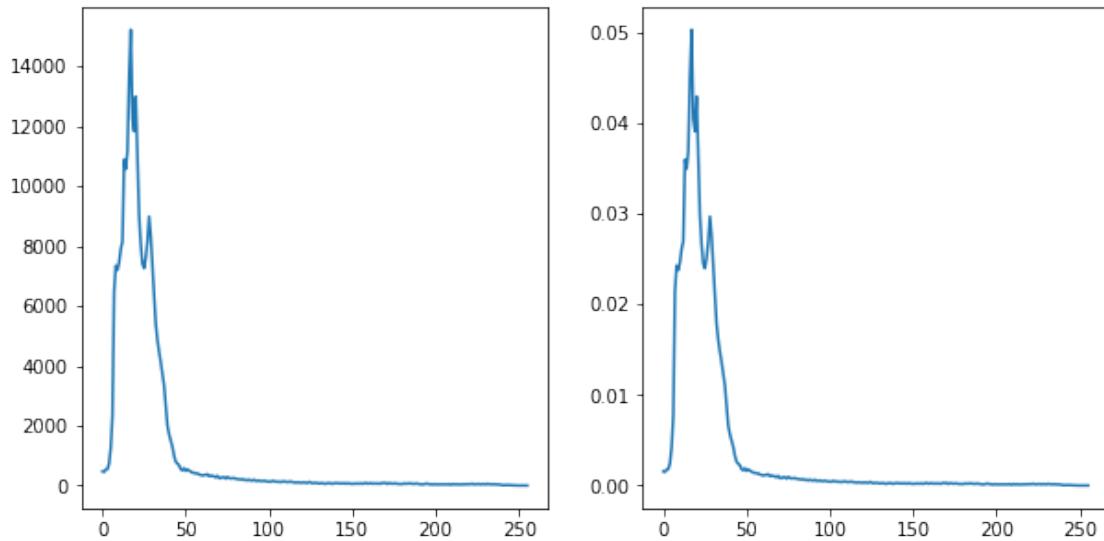
Za čuvanje vrednosti broja piksela pojedinačnih intenziteta koristi se vektor od 256 elemenata čije su vrednosti inicijalno 0. Broj piksela koji imaju intenzitet 0 sačuvaćemo na poziciji indeksa 0 ovog vektora, za intenzitet 1 na indeksu 1, itd. Umesto da vršimo pretragu za svaki intenzitet pojedinačno (kao u prethodnom primeru) ovde koristimo osobinu tipa ulaza učitane slike (uint8) da izvršimo direktno indeksiranje odgovarajuće lokacije gde čuvamo brojnost odgovarajućeg intenziteta i inkrementiramo je za 1.

Ovaj rezultat suštinski predstavlja **histogram**. U ovom primeru broj elemenata u vektoru h je jednak broju mogućih vrednosti definisan tipom ulazne slike, dok u opštem to ne mora da važi. Elementi ovog vektora (obično se nazivaju binovi, engl. *bin*) koriste se za prebrojavanje piksela čija vrednost pripada nekom intervalu. Obično su intervali iste širine i ne preklapaju se međusobno.

Normalizovani histogram se formira tako što se početni histogram podeli sa ukupnim brojem piksela u slici (ili sumom svojih članova). U ovakvoj formi, normalizovan histogram predstavlja prikaz udela (ili verovatnoće) pojedinačnih intenziteta piksela. Suma elemenata normalizovanog histograma je stoga jednaka 1.

```
[4]: norm_hist = h/h.sum() # ili h/img.size

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.plot(np.arange(256), h)
plt.subplot(1,2,2)
plt.plot(np.arange(256), norm_hist)
plt.show()
```



2.3 Primer 5.3

Koristeći python pakete numpy i skimage odrediti i prikazati histograme slike boat.png. Smatratи da histogram određujemo u broju binova definisan opsegom mogućih vrednosti za dati tip (kao u prethodnom primeru).

```
[5]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np
from skimage import exposure

img = io.imread('boat.png')

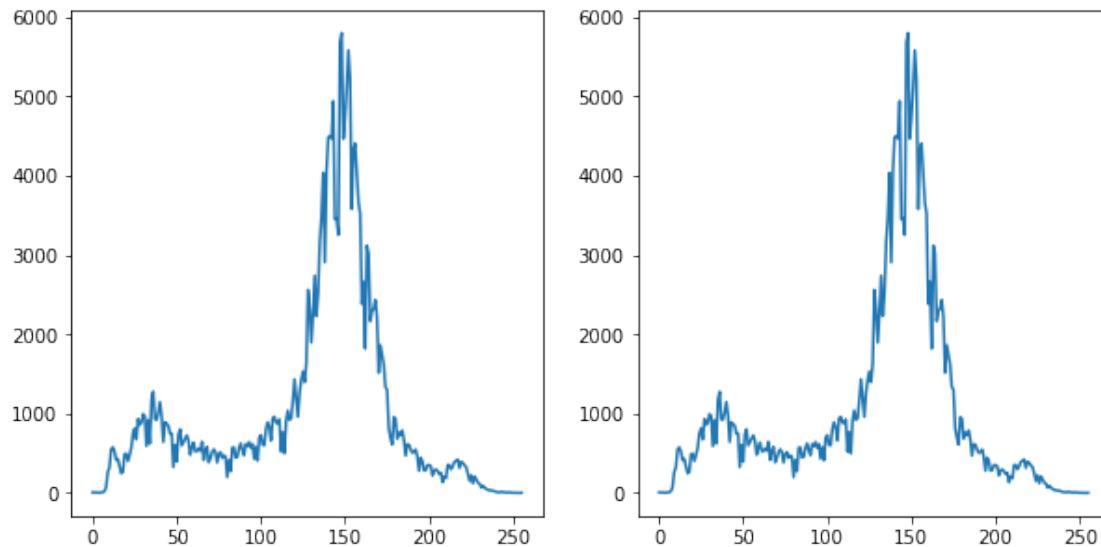
numpy_hist, nbin_edges = np.histogram(img, bins = np.arange(257))
skimage_hist, sbin_centers = exposure.histogram(img, source_range='dtype')

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
```

```

plt.plot(np.arange(256), numpy_hist)
plt.subplot(1,2,2)
plt.plot(sbin_centers, skimage_hist)
plt.show()

```



Histogram slike je moguće odrediti pomoću ugrađenih funkcija pojedinih paketa. Ovde prikazujemo kako to možemo uraditi uz pomoć dosadašnje korišćenih paketa `numpy` i `skimage`, kao i sa paketom `scipy` koji slično služi za matematičku obradu signala. * `numpy.histogram` - Osnovni način pozivanja je sa ulaznim argumentima slike kojoj računamo histogram i lokacija pojedinačnih binova. Za primer binova [1, 2, 3, 4], prvi bin je predstavljen sa poluotvorenim intervalom [1, 2). Slično je i za ostale intervale, osim za poslednji bin gde je interval u ovom primeru [3, 4]. Funkcija ima kao povratnu vrednost histograma i granice binova. * `skimage.exposure.histogram` - Funkcija je u okviru `exposure` modula. Kao i u prethodnoj funkciji dobijamo dve povratne vrednosti, gde je prva histogram, ali druga predstavlja centre pojedinačnih binova. Funkcija radi optimalnije nad celobrojnim vrednostima intenziteta, gde je poželjno navesti parametar `source_range='dtype'` da bi se histogram računao nad vrednostima opsega definisanim tipom slike (u suprotnom računa od minimalne do maksimalne vrednosti u slici).

Obe funkcije imaju mogućnost podešavanja dodatnog parametra za formiranje normalizovanog histograma. Pogledati odgovarajuću dokumentaciju ovih funkcija da bi se upoznale druge mogućnosti.

2.4 Primer 5.4

Za sliku `cat.png` odrediti udeo piksela koji imaju intenzitet 50 ili manji.

```
[6]: import plotly.express as px
      from skimage import io
```

```

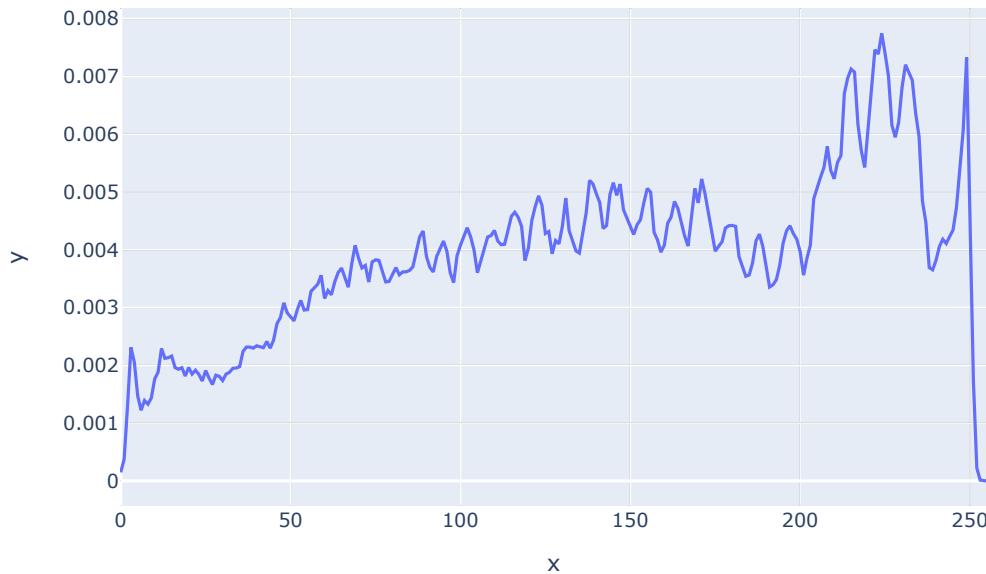
import numpy as np

img = io.imread('cat.png')

hist, bin_edges = np.histogram(img, bins = np.arange(257), density=True)
fig = px.line(x=np.arange(256), y=hist)
fig.show()

udeo = hist[:51].sum()
print('Udeo piksela koji imaju intenzitet 50 ili manji je: {}%'.format(udeo*100))

```



Udeo piksela koji imaju intenzitet 50 ili manji je: 10.0364729793691%

Za rešavanje problema koristimo normalizovan histogram, ovde formiran putem numpy paketa i sa dodatnim parametrom density. Svaki pojedinačni element histograma ukazuje na udeo piksela sa odgovarajućim intenzitetom. Da bi odredili udeo piksela koji imaju intenzitet 50 ili manji, potrebno je stoga sumirati elemente histograma koji odgovaraju ovim intenzitetima.

2.5 Primer 5.5

Za sliku lena.png odrediti udeo piksela koji imaju intenzitet i ili manji za $i = 0, 1, \dots, 255$.

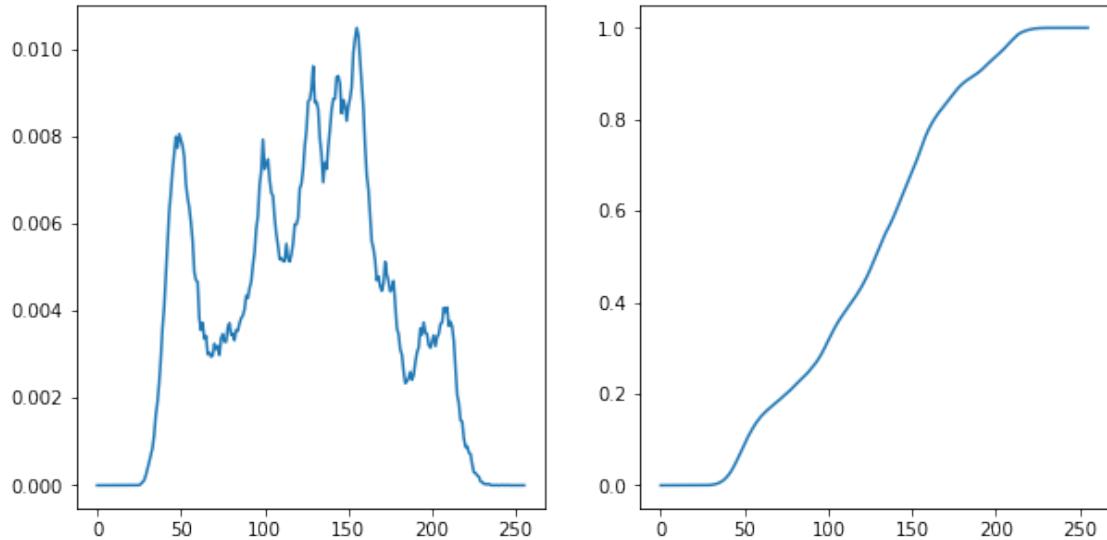
```
[7]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np

img = io.imread('lena.png')

hist, bin_edges = np.histogram(img, bins = np.arange(257), density=True)

udeli = np.zeros(256)
for i in range(256):
    udeli[i] = hist[:i+1].sum()

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.plot(np.arange(256), hist)
plt.subplot(1,2,2)
plt.plot(sbin_centers, udeli)
plt.show()
```



Problem rešavamo prolaskom po svim vrednostima intenziteta i računanjem odgovarajuće sume elemenata od početka histograma. Rezultujuća kriva suštinski predstavlja funkciju raspodele (engl. *cumulative distribution function, cdf*) za datu funkciju gustine verovatnoće (normalizovani histogram).

2.6 Primer 5.6

Napisati funkciju `imageHistEq` koja ekvalizuje histogram ulazne slike. Prepostaviti da je opseg vrednosti ulazne slike $[0, 255]$. Transformacija koja određuje ekvalizaciju histograma je data sledećim izrazom:

$$T(r_k) = (L - 1) \cdot \sum_{i=0}^k \frac{h_i}{n}, \quad k = 0, 1, \dots, L - 1$$

gde k predstavlja nivo intenziteta u opsegu $[0, L - 1]$, gde je L ukupni broj niva intenziteta koji piksel može imati (zavisno od tipa), r_k je intenzitet piksela na nivou k , $T(\cdot)$ je transformacija koja definiše ekvalizaciju histograma, h_i je broj piksela koji imaju intenzitet i , a n je ukupan broj piskela u slici.

Kao rezultat funkcije vratiti sliku sa ekvalizovanim histogramom i transformaciju koja je to omogućila.

Funkciju primeniti nad slikama `dark.png` i `light.png`. Prikazati sliku i njen histogram pre i posle poziva funkcije. Prikazati kako izgleda funkcija T za svaku sliku.

```
[8]: import numpy as np

def imageHistEq(img):
    hist = np.histogram(img, bins = np.arange(257), density=True)[0]
    T = 255 * np.cumsum(hist)
    T = np.round(T).astype('uint8') # ako zelimo rezultat u tipu uint8

    return T[img], T
```

Ekvalizacija histograma je transformacija koja za cilj ima dobijanje slike čiji je histogram približno uniforman. U okviru nje se koristi funkcija raspodele, koju možemo efikasnije napisati preko kumulativne sume računata sa `numpy.cumsum`.

```
[9]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np

# dark
dark = io.imread('dark.png')
hist_dark = np.histogram(dark, bins = np.arange(257), density=True)[0]
dark_eq, T_dark = imageHistEq(dark)
hist_dark_eq = np.histogram(dark_eq, bins = np.arange(257), density=True)[0]

plt.figure(figsize=(10,5))

plt.subplot(2,5,1)
plt.imshow(dark, vmin=0, vmax=255, cmap='gray')
plt.subplot(2,5,2)
plt.plot(np.arange(256), hist_dark)
plt.subplot(2,5,3)
plt.plot(np.arange(256), T_dark)
plt.subplot(2,5,4)
plt.imshow(dark_eq, vmin=0, vmax=255, cmap='gray')
```

```

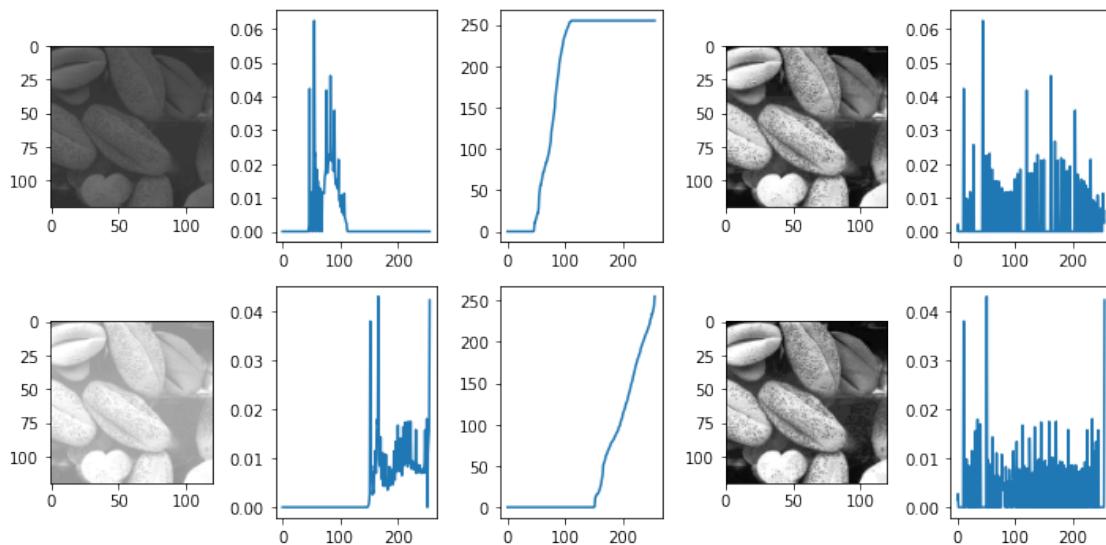
plt.subplot(2,5,5)
plt.plot(np.arange(256), hist_dark_eq)

# light
light = io.imread('light.png')
hist_light = np.histogram(light, bins = np.arange(257), density=True)[0]
light_eq, T_light = imageHistEq(light)
hist_light_eq = np.histogram(light_eq, bins = np.arange(257), density=True)[0]

plt.subplot(2,5,6)
plt.imshow(light, vmin=0, vmax=255, cmap='gray')
plt.subplot(2,5,7)
plt.plot(np.arange(256), hist_light)
plt.subplot(2,5,8)
plt.plot(np.arange(256), T_light)
plt.subplot(2,5,9)
plt.imshow(light_eq, vmin=0, vmax=255, cmap='gray')
plt.subplot(2,5,10)
plt.plot(np.arange(256), hist_light_eq)

plt.tight_layout() # komanda za obezbedjivanje dovoljno prostora za iscrtavanje
                   → svih elemenata
plt.show()

```



Korišćenjem ekvalizacije dolazi do preraspodele originalnih intenziteta po celokupnom opsegu. Ova transformacija omogućuje kreiranje većih rastojanja između originalnih nijansi što prouzrokuje poboljšanje kontrasta čitave slike. Ekvalizovane slike izgledaju slično, dok su im histogrami drugačiji ali približno prate uniforman oblik u poređenju sa originalnim histogramima.

Ekvalizaciju možemo uraditi i korišćenjem funkcije `skimage.exposure.equalize_hist`. U primeru je korišćena funkcija `img_as_ubyte` u okviru `skimage.util` modula za konverziju rezultata u `uint8` tip. Dodatno se koristi računanje histograma iz scikit-image paketa sa dodatnim parametrom `normalize` za određivanje normalizovanog histograma.

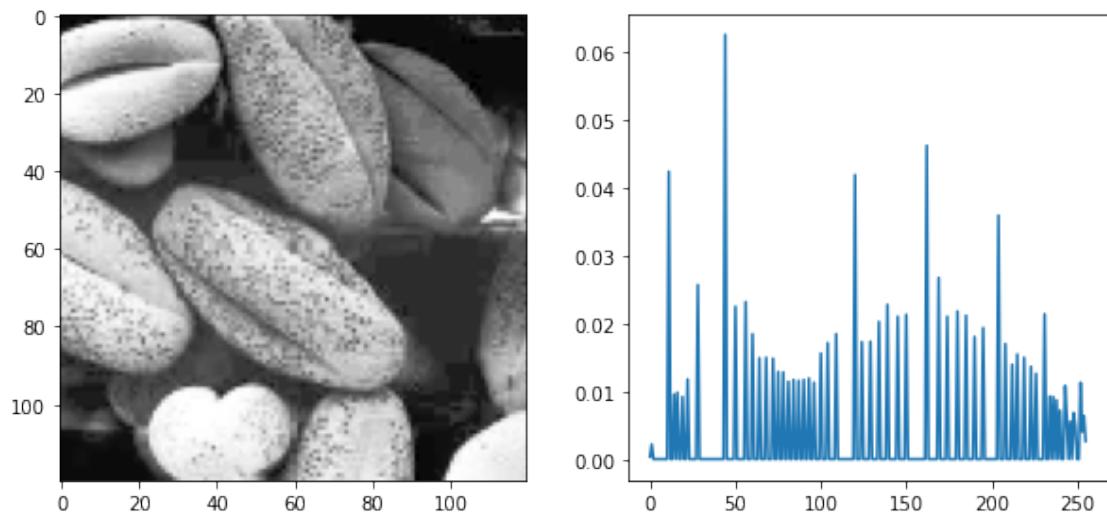
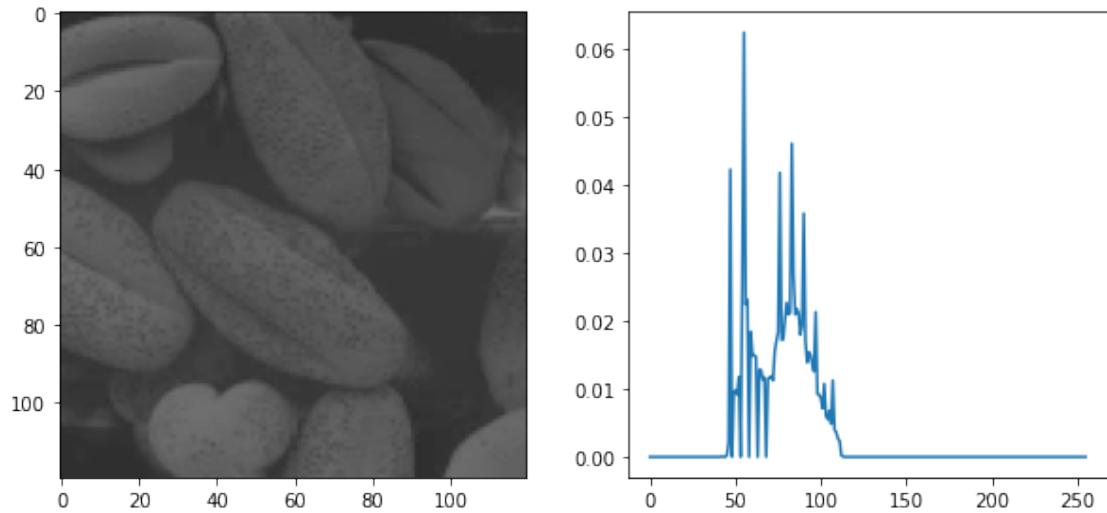
```
[10]: import matplotlib.pyplot as plt
from skimage import io
from skimage import exposure
from skimage.util import img_as_ubyte
import numpy as np

# dark
dark = io.imread('dark.png')
hist, bin_centers = exposure.histogram(dark, source_range='dtype', normalize=True)

dark_eq = img_as_ubyte(exposure.equalize_hist(dark))
hist_eq, bin_centers = exposure.histogram(dark_eq, source_range='dtype', normalize=True)

plt.figure(figsize=(10,10))
plt.subplot(2,2,1)
plt.imshow(dark, vmin=0, vmax=255, cmap='gray')
plt.subplot(2,2,2)
plt.plot(np.arange(256), hist)
plt.subplot(2,2,3)
plt.imshow(dark_eq, vmin=0, vmax=255, cmap='gray')
plt.subplot(2,2,4)
plt.plot(np.arange(256), hist_eq)

plt.show()
```



3 Zadaci za samostalnu vežbu

Zadatak 5.1

Medicinski snimci često imaju i velik opseg vrednosti i velik broj piksela. U određenim primenama je dovoljna samo gruba procena histograma snimka, te prilikom njegovog računanja nije potrebno u obzir uzeti svaki pixel niti svaku moguću vrednost piksela.

Implementirati funkciju `imageHist`, bez korišćenja gotovih funkcija za računanje histograma, koja pored slike prima dodatne ulazne argumente koji određuju način računanja histograma: **broj binova**, **granice opsega** vrednosti piksela i **korak** između susednih piksela koji treba da se uzmu u obzir. Npr. za korak koji ima vrednost 5, svaki peti pixel treba da se uzme u obzir prilikom određivanja histograma. Za postavljene granice opsega, histogram se formira samo od intenziteta koji su u okviru njih. Za postavljen broj binova, histogram ima tačno naznačen broj binova podjene

širine. Funkciju primeniti nad slikom `mam_2.png` za nekoliko različitih parametara.

Napomena: zbog veličine slike koristiti `matplotlib` paket za prikaz.

[11] : `#TODO`

Zadatak 5.2

Algoritmi koji se koriste za poboljšanje vidljivosti detalja i kontrasta na medicinskim snimcima mogu da proizvedu različite artefakte. Jedan od artefakta je pojava „crnog oreola“. Ovaj artefakt je često vidljiv uz granicu anatomije i za posledicu ima smanjenje kontrasta snimka.

Jednostavan način da se kvalitet slike poboljša je podešavanje granica tonske skale tako da crni oreol ne bude vidljiv. Međutim, vrednosti piksela snimka su određene anatomijom koja je snimljena te odgovarajuće granice tonske skale nije moguće unapred odrediti, ali je moguće odrediti procente piksela za koje želimo da se nađu izvan granica tonske skale.

Napisati funkciju `toneScaleLimit` koja određuje granice tonske skale za dati snimak. Granice tonske skale treba da se odrede u zavisnosti od ulaznih parametara koji predstavljaju procenat piksela sa niskim intenzitetom i procenat piksela sa visokim intenzitetom koji treba da se nađu izvan granica tonske skale. Ako su, primera radi, za ulazne argumente funkcije zadate granice 0.02 i 0.003, onda funkcija treba da vrati granice tonske skale tako da se 2% piksela sa najmanjim vrednostima nađu ispod donje granice tonske skale, a da se 0.3% piksela sa najvećim vrednostima nađu iznad gornje granice tonske skale. Funkciju upotrebiti na snimku `rtg_5.png`.

Napomena: zbog veličine slike koristiti `matplotlib` paket za prikaz.

[12] : `#TODO`

4 Dokumentacija novih celina

- [numpy.histogram](#)
- [skimage.exposure.histogram](#)
- [numpy.cumsum](#)
- [skimage.exposure.equalize_hist](#)
- [scikit-image data types](#)

Vezba 6 - Prostorno filtriranje

1 Pregled

U okviru vežbe se radi upoznavanje sa prostornim filtriranjem slike, odnosno operacijama na osnovu okoline piksela. U vežbi se analiziraju funkcije za proširenje slike i njihova primena da bi se ublažili granični efekti filtriranja. Posmatran je uticaj filtara ublaživača na sliku i mogućnost njihove upotrebe za pojačavanje vidljivosti detalja preko ansharp masking (engl. *unsharp masking*) procedure. Izdvajanje detalja slike i njihovo pojačanje je analizirano i preko Laplasijan filtra. U vežbi su obrađeni i elementarni filtri za estimaciju vrednosti gradijenata.

2 Preduslov

Instaliran [SciPy](#) paket - Python-based ecosystem of open-source software for mathematics, science, and engineering.

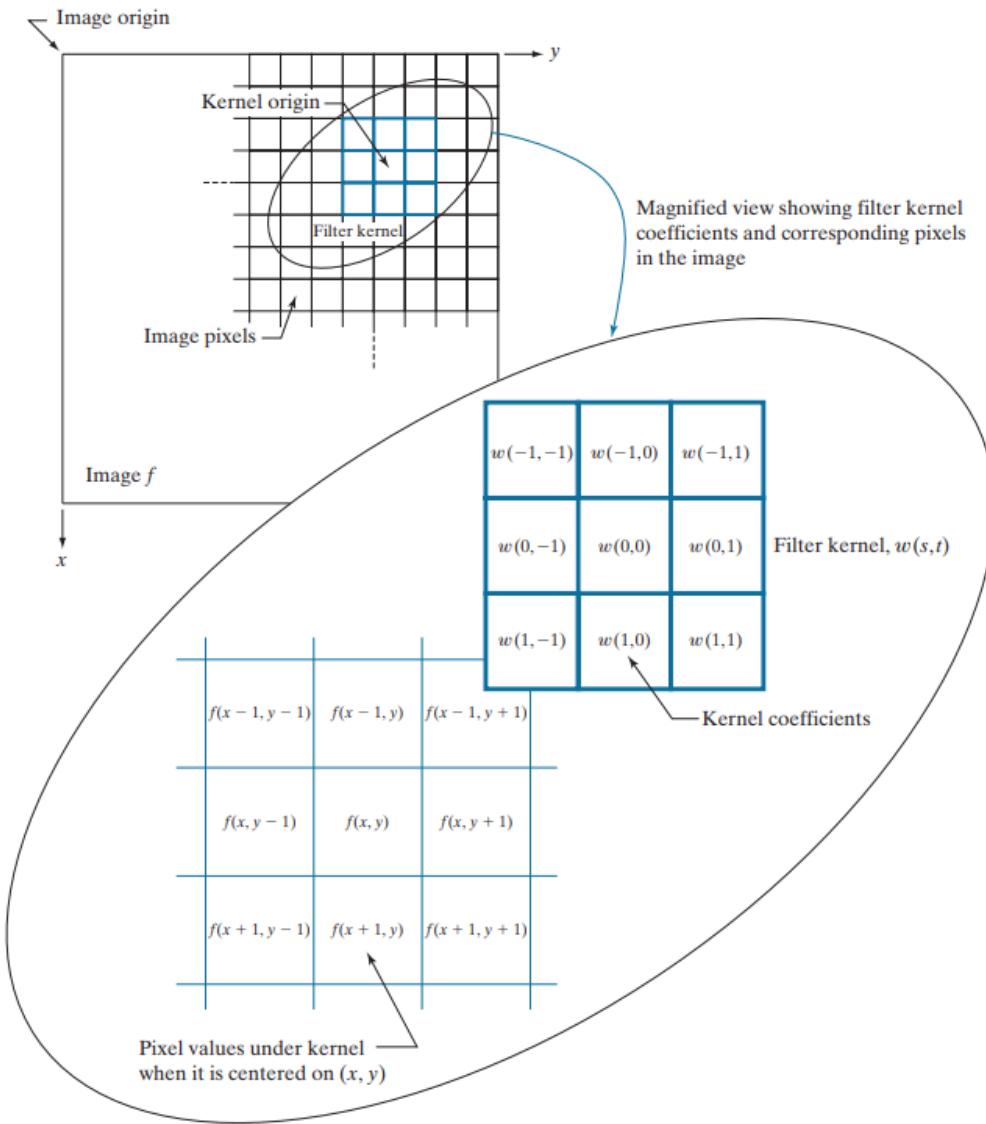
```
pip install scipy
```

3 Prostorno filtriranje

Prostorno filtriranje modifikuje sliku tako što menja vrednost pojedinačnih piksela sa nekim rezultatom funkcije čiji su ulazi intenziteti iz oblasti (ili regionala) oko posmatranih piksela. Ako je funkcija linearne prirode, filtriranje nazivamo prostorno filtriranje sa linearnim filtrom.

Linearni filtri suštinski vrše računanje sume proizvoda između slike i tzv. filter kernelom (drugi naziv samo kernel/maska/filtar/prozor). Kernel predstavlja niz (u okviru slike je 2D niz) čije dimenzije definišu koji okolni region piksela se posmatra a vrednosti određuju prirodu filtriranja.

Slika i formula ispod ilustruje mehanizam linearog filtriranja slike f sa kernelom w dimenzije 3×3 . Za kernele neparnih dimenzija, radi jednostavnosti pisanja izraza za filtriranje, njegov centar (engl. *origin/anchor point*) je definisan pozicijom centralnog elementa.



slika preuzeta iz knjige "Digital Image Processing", Gonzalez, Woods, fig 3.28, strana 155. Napomena da su ose ovde prestavljene sa x za vrste a y za kolone

Rezultat filtriranja na poziciji (x, y) je dat kao:

$$g(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x-s, y-t)$$

što predstavlja konvoluciju ulazne slike i filtra, dok

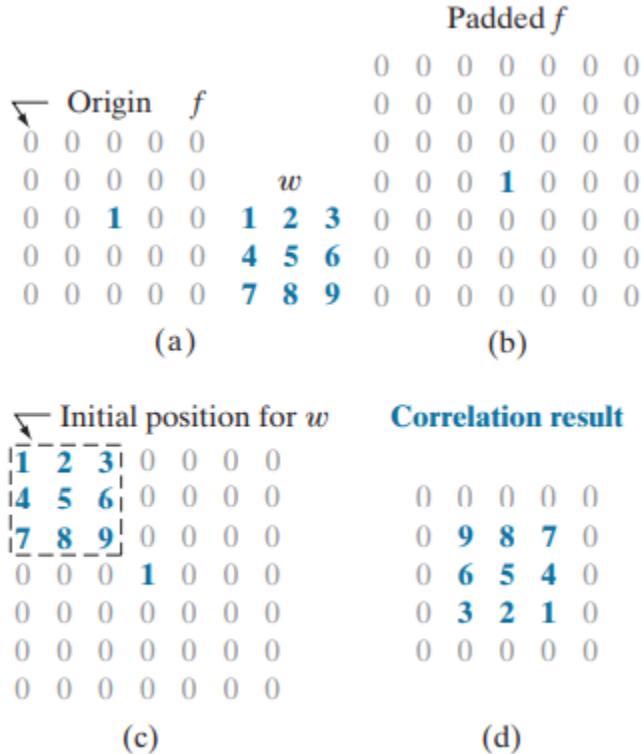
$$g(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x+s, y+t)$$

predstavlja korelaciju ulazne slike i filtra koji je unapred prilagođen okretanjem po horizontalnoj

i vertikalnoj osi.

U slučaju simetričnih kernela, obe operacije daju istovetne rezultate.

Prilikom filtriranja slike potrebno je povesti računa u ivičnim delovima slike gde pozicioniranjem kernela izlazimo iz opsega slike. U tom slučaju je potrebno vršiti proširivanje slike (engl. *padding*) da bi obezbedili vrednosti za operacije konvolucije/korelacije.



slika preuzeta iz knjige "Digital Image Processing", Gonzalez, Woods, fig 3.30, strana 158.

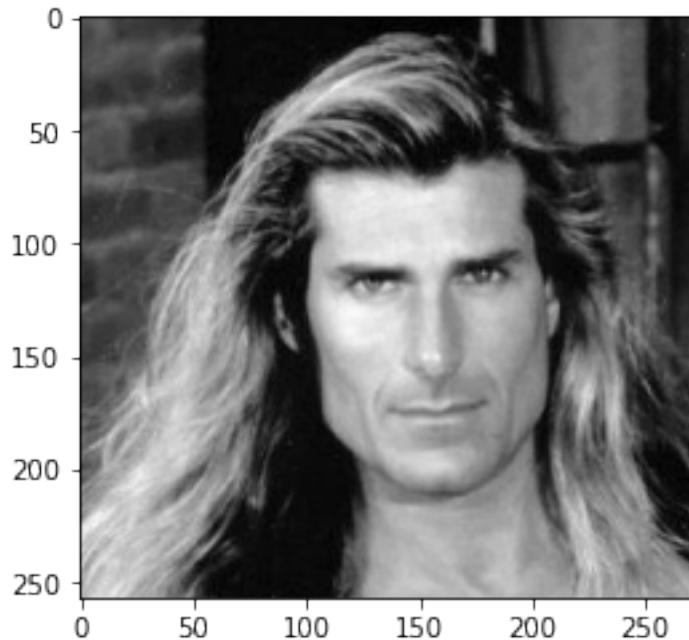
3.1 Primer 6.1

Nad slikom fabio.tif izvršiti proširivanje:

1. sa svih strana za po 200 piksela vrednosti 100
 2. sa gornje i donje strane za po 200 piksela, a sa leve i desne za po 100 piksela vrednosti 0
 3. sa gornje i donje strane za po 150 piksela ponavljanjem vrednosti ivičnih piksela
 4. sa leve i desne strane za po 250 piksela simetričnim ponavljanjem vrednosti
 5. sa donje strane za po 100 piksela cirkularnim ponavljanjem piksela

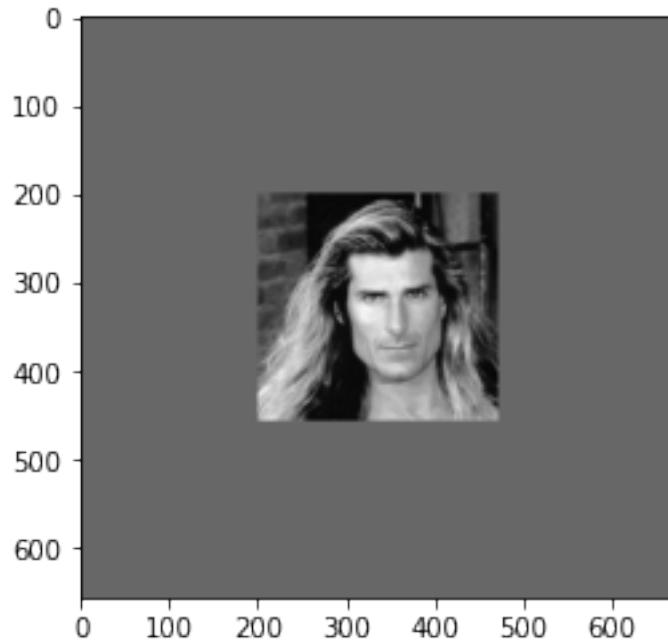
```
[2]: import matplotlib.pyplot as plt  
      from skimage import io  
      import numpy as np
```

```
img = io.imread('fabio.tif')
plt.imshow(img, cmap='gray')
plt.show()
```



[3]: # 1 - "constant" padding

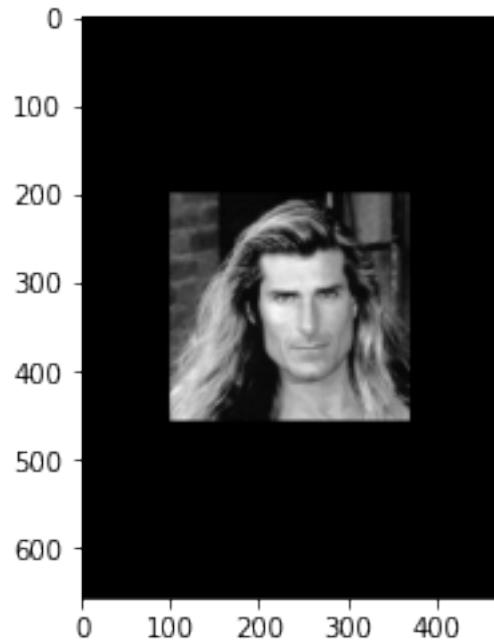
```
img1 = np.pad(img, pad_width=200, mode='constant', constant_values=100)
plt.imshow(img1, cmap='gray')
plt.show()
```



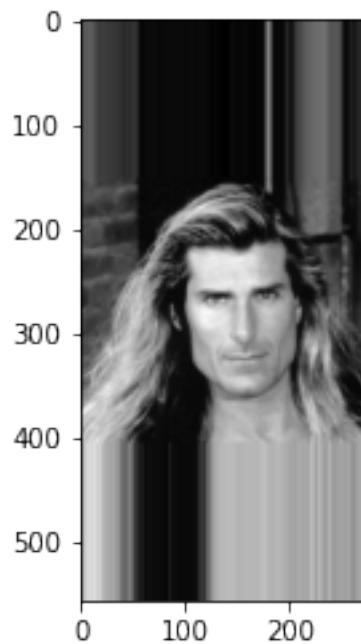
Funkcija `numpy.pad` se može koristi za proširivanje slika. Ulagni argumenti su:

- * Slika koju je potrebno proširiti
- * `pad_width` - parametar proširenja dat u obliku tuple vrednosti
((proširenje iznad, proširenje ispod), (proširenje levo, proširenje desno))
- sa mogućim skraćenim zapisima
- * `mode` - određuje način proširenja
- * `constant_values` - dodatni parametar za konstantni mod proširenja

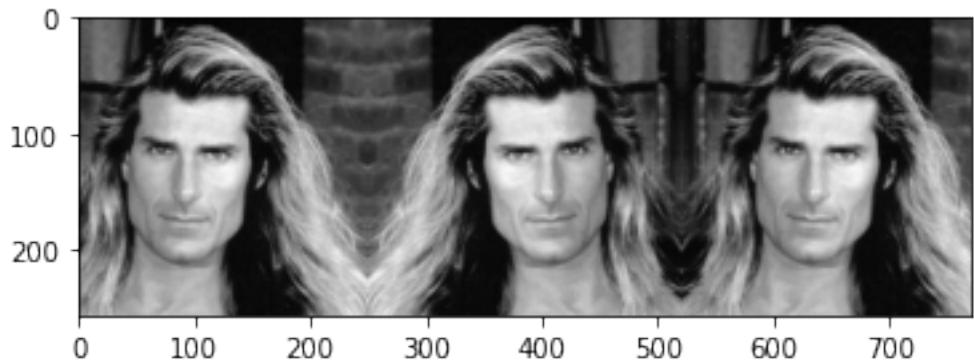
```
[4]: # 2 - "constant" padding sa vrednost 0 -> "zero padding"
img2 = np.pad(img, ((200,), (100,)))
plt.imshow(img2, cmap='gray')
plt.show()
```



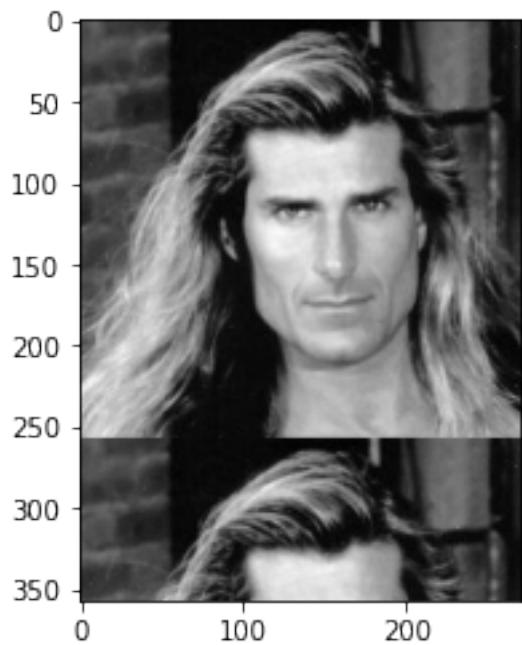
```
[5]: # 3 - "edge" padding
img3 = np.pad(img, ((150,), (0,)), mode='edge')
plt.imshow(img3, cmap='gray')
plt.show()
```



```
[6]: # 4 - "symmetric" padding
img4 = np.pad(img, ((0,), (250,)), mode='symmetric')
plt.imshow(img4, cmap='gray')
plt.show()
```



```
[7]: # 5 - "wrap" padding
img5 = np.pad(img, ((0,100),(0,0)), mode='wrap')
plt.imshow(img5, cmap='gray')
plt.show()
```



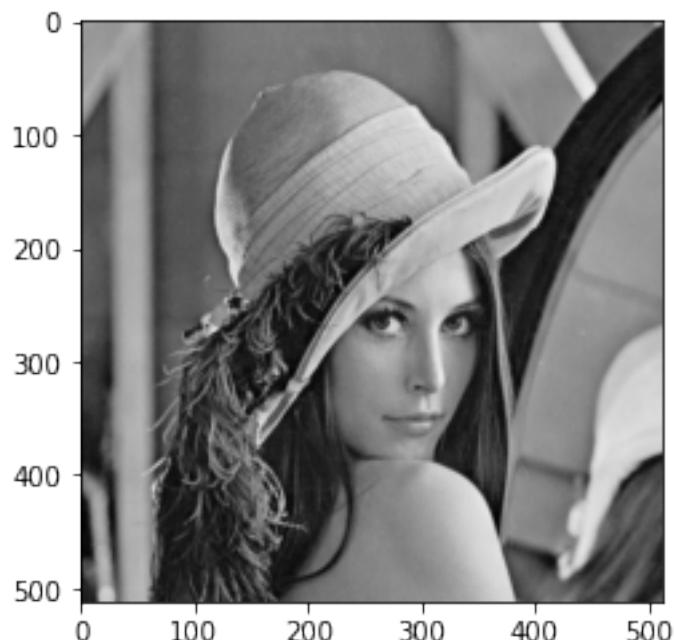
3.2 Primer 6.2

Nad slikom lena.png primeniti filter aritmetički usrednjivač: 1. veličine 101x101, računajući tzv. *full* konvoluciju 2. veličine 11x11, sa zadržavanjem originalne dimenzije slike i korišćenjem zero padding-a 4. veličine 7x7, sa zadržavanjem originalne dimenzije slike i korišćenjem simetričnog proširenja slike 5. veličine 3x3, sa zadržavanjem originalne dimenzije slike i korišćenjem proširenja slike ponavljanjem ivičnih piksela

```
[8]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np

from scipy import signal # za koriscenje scipy.signal.convolve2d funkcije
from scipy import ndimage # za koriscenje scipy.ndimage.convolve funkcije

img = io.imread('lena.png')
plt.imshow(img, cmap='gray')
plt.show()
```

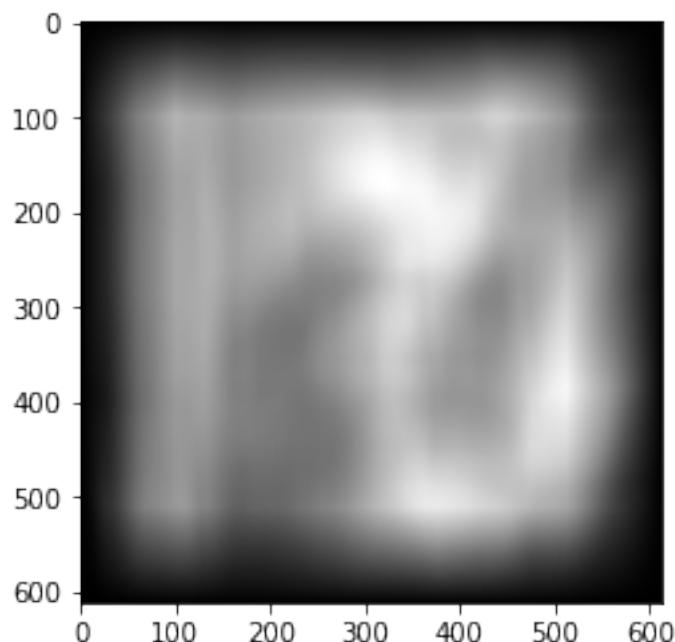


```
[9]: # usrednjivac dimenzije 101x101
w = np.ones((101,101))/101**2

# "full" konvolucija
img1 = signal.convolve2d(img, w, mode='full')
```

```
print(img1.shape)
plt.imshow(img1, cmap='gray')
plt.show()
```

(612, 612)



Aritmetički usrednjivači, kao što ime kaže, računaju rezultat kao aritmetičku sredinu posmatranog regiona. Drugi naziv za njih je **box filter**. Jednostavno se formiraju kao matrica sa vrednostima 1 podeljena sa ukupnim brojem članova.

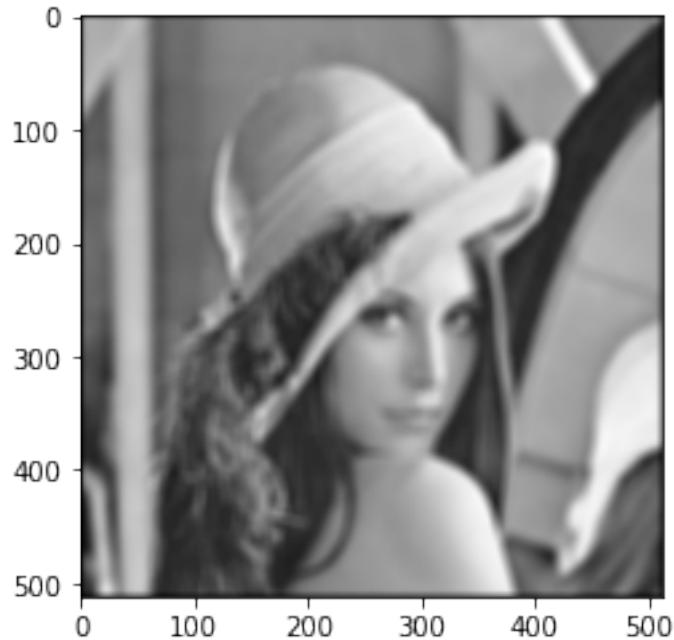
Operacija linearog filtriranja je obavljena korišćenjem `scipy.signal.convolve2d` funkcije sa sledećim parametrima ulazne slike, kernela i parametra `mode` postavljenog na `full` za računanje klasične konvolucije. Slika u ovom slučaju ima uvećanu dimenziju i to za 100 piksela po obe dimenzije kao i tip promenjen u odgovarajući koji može da čuva rezultate. Moguće varijante mode parametra: * '`full`' - klasična konvolucija nad signalima * '`same`' - izlazna vrednost konvolucije biće skraćena na dimenziju prvog parametra * '`valid`' - izlazne vrednosti konvolucije se formiraju samo na osnovu vrednosti slike bez proširenja

Funkcija dodatno može imati postavljen parametar `boundary` za proširenje koje je potrebno da računa. Podrazumevan je zero padding. Moguće varijante: * '`fill`' - proširuje sliku sa vrednostima dodatnog parametra `fillvalue` * '`wrap`' / '`circular`' - proširuje sliku sa vrednostima cirkularno ponovljenih vrednosti * '`symm`' / '`symmetric`' - proširuje sliku sa simetričnim ponavljanjem vrednosti piksela u odnosu na granicu uz ivicu

```
[10]: # usrednjivac dimenzije 11x11
w = np.ones((11,11))/11**2
```

```
# "same" konvolucija
img2 = signal.convolve2d(img, w, mode='same')
print(img2.shape)
plt.imshow(img2, cmap='gray')
plt.show()
```

(512, 512)

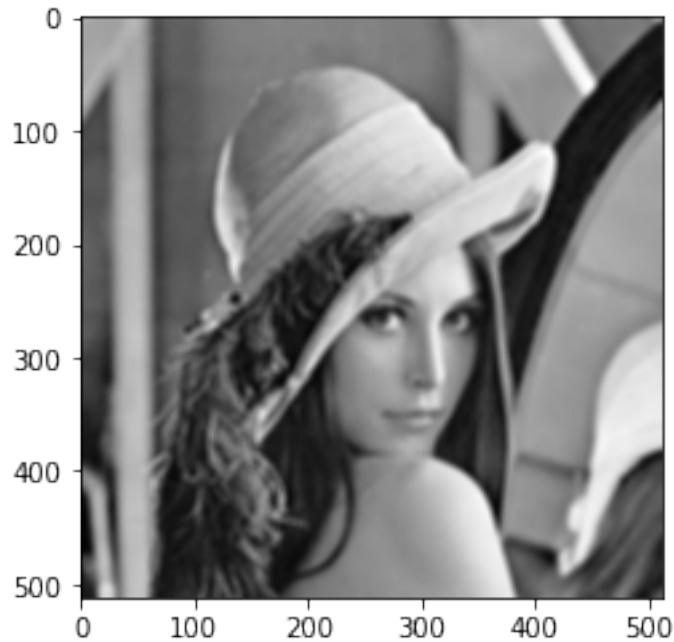


U ovom i prethodnom slučaju uočavamo da su ivice slike znatno tamnije u odnosu na originalnu sliku. Razlog ovog graničnog efekta je jer se koristi ubacivanje nula za pozicije van definisanog regiona slike, koje imaju efekat na računanje srednje vrednosti ove okoline. Efekat se može ublažiti korišćenjem drugih načina proširenja koji nastoje da održe prosečnu energiju (srednju vrednost) prisutnu u ovim delovima slike.

```
[11]: # usrednjivac dimenzije 7x7
w = np.ones((7,7))/49

# "same" symmetric konvolucija
img3 = signal.convolve2d(img, w, mode='same', boundary='symmetric')
print(img3.shape)
plt.imshow(img3, cmap='gray')
plt.show()
```

(512, 512)



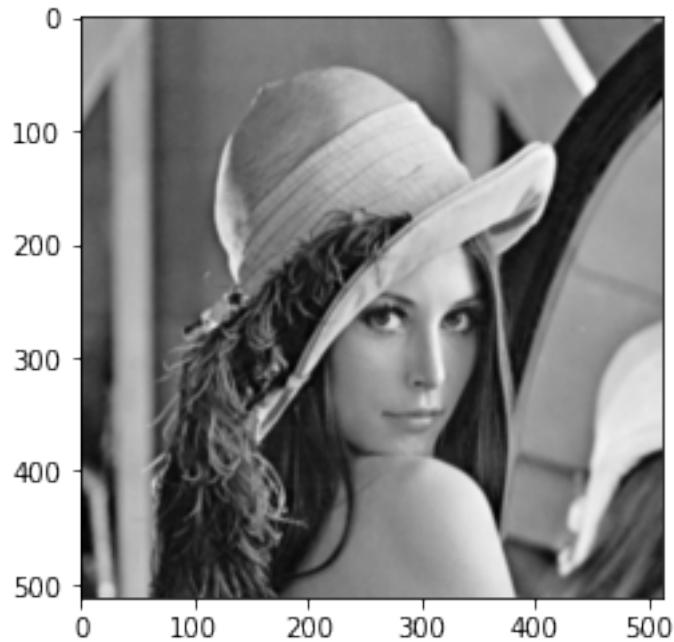
Aritmetički usrednjivači spadaju u filtre za ublažavanje, pa je rezultat filtriranja slika koja ima ublažene ivice u odnosu na originalnu. Što su veće dimenzije maske to je efekat ublažavanja veći.

```
[12]: # usrednjivac dimenzije 3x3
w = np.ones((3,3))/9

# rucno prosirenje sa 'edge' parametrom
img_p = np.pad(img,(1,1),mode='edge')

# "valid" konvolucija
img4 = signal.convolve2d(img_p, w, mode='valid')
print(img4.shape)
plt.imshow(img4, cmap='gray')
plt.show()
```

(512, 512)



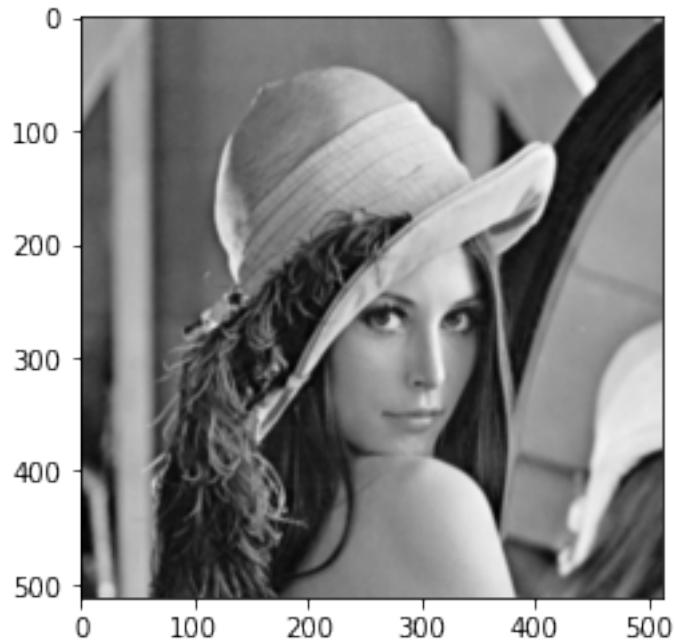
Pošto funkcija `scipy.signal.convolve2d` nema opciju za zahtevani način proširenja (ponavljanje ivičnih vrednosti), potrebno je ručno izvršiti proširenje nakon čega možemo koristiti funkciju sa `valid` parametrom za dobijanje rezultata iste dimenzije kao i polazna slika.

Alternativno, za iste rezultate (i više mogućnosti padding-a) možemo koristiti i funkciju `scipy.ndimage.convolve`. Funkcija održava tip ulazne slike, stoga za dobijanje identičnog rezultata potrebno je izvršiti promenu tipa slike u odgovarajući za izlaz ili korišćenjem dodatnog parametra `output`. Funkcija je specijalizovana za slike i uvek smatra da izlaz treba da bude iste veličine kao i ulazna slika. Poslednji rezultat bi se dobio kao:

```
[13]: # usrednjivac dimenzije 3x3
w = np.ones((3,3))/9

# konvolucija sa 'nearest' prosirenjem - 'nearest' u okviru ndimage.convolve
# odgovara 'edge' u okviru np.pad
img5 = ndimage.convolve(img, w, mode='nearest', output='float')
print(img5.shape)
plt.imshow(img5, cmap='gray')
plt.show()
```

(512, 512)



3.2.1 Tabela naziva proširenja

Radi lakšeg praćenja naziva mogućih proširenja između funkcija, kreirana je tabela ispod za prikaz proširenja sekvence a b c d, kombinovanjem informacija iz dokumentacija odgovarajućih funkcija.

numpy.pad	scipy.signal.convolve	scipy.ndimage.convolve	stracija	objašnjenje
constant	fill	constant	k k k k \ a b c d \ k k k k	proširenje sa konstantnim vrednostima k
edge		nearest	a a a a \ a b c d \ d d d d	proširenje sa ponavljanjem ivičnih elemenata
reflect		mirror	d c b \ a b c d \ c b a	proširenje sa ponavljanjem simetričnih vrednosti u odnosu na ivice
symmetric	symmetric	reflect	d c b a \ a b c d \ d c b a	proširenje sa ponavljanjem simetričnih vrednosti u odnosu na granicu uz ivicu

numpy.pad	scipy.signal.convolve	scipy.ndimage.convolve	histrakcija	objašnjenje
wrap	wrap	wrap	a b c d \ \ a b c d \ a b c d	proširenje sa cirkularnim ponavljanjem elemenata

3.3 Primer 6.3

Nad slikom boat.png upotrebiti Laplasijan filter koji posmatra 4 suseda. Sliku izoštiti upotrebom Laplasijan filtra. Prikazati dobijene rezultate.

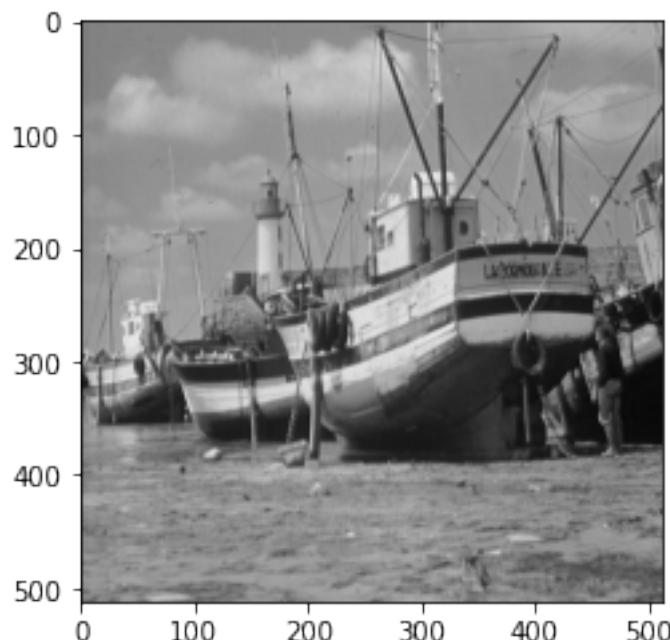
Laplasijan za izdvajanje detalja je dat kao:

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Napomena: ako koristimo `scipy.ndimage.convolve` funkciju potrebno je promeniti tip slike da se mogu sačuvati eventualne negativne vrednosti.

```
[14]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np
from scipy import ndimage

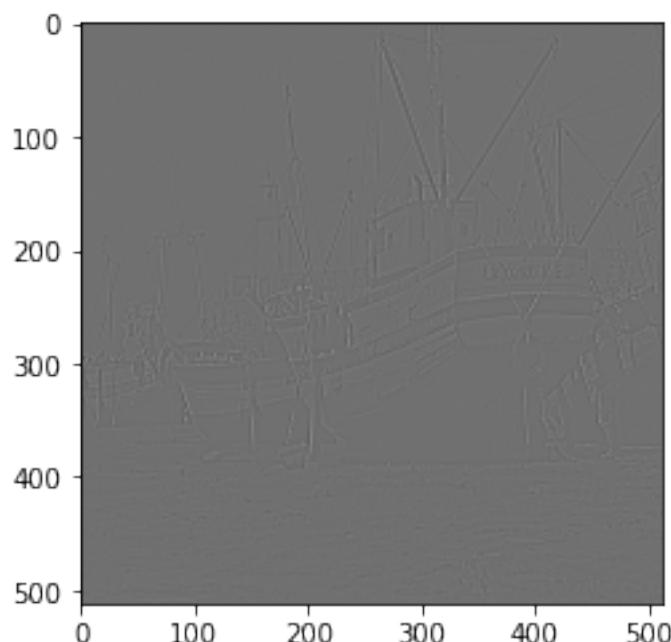
img = io.imread('boat.png')
plt.imshow(img, cmap='gray')
plt.show()
```



```
[15]: L = np.array([[0,-1,0],[-1,4,-1],[0,-1,0]])
print(L)
```

```
[[ 0 -1  0]
 [-1  4 -1]
 [ 0 -1  0]]
```

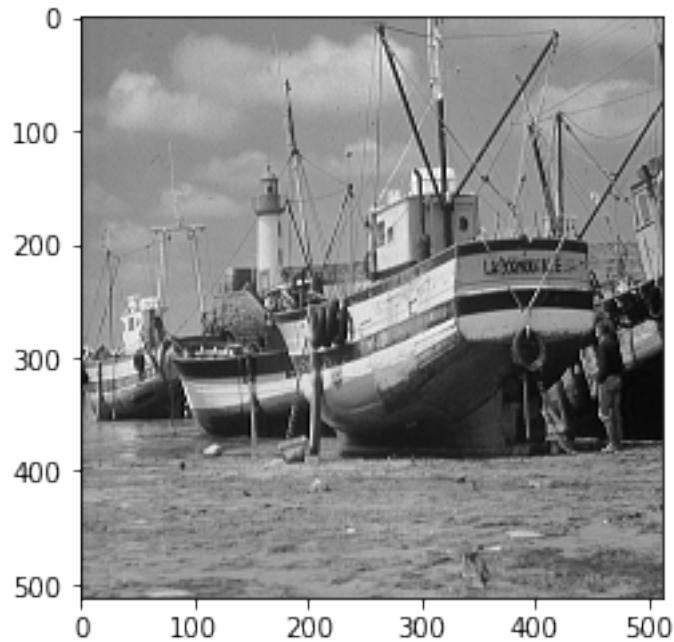
```
[16]: img_lap = ndimage.convolve(img.astype('float'), L, mode='reflect')
plt.imshow(img_lap, cmap='gray')
plt.show()
```



Izdvajanje detalja slike moguće je primenom Lapasijan filtra koji implementira drugi izvod po obe koordinate. Izoštravanje slike (naglašavanje detalja) se postiže dodavanjem rezultata filtriranja Laplasijan filtrom na originalnu sliku.

```
[17]: img_detail_enhanced = img + img_lap

plt.imshow(img_detail_enhanced, vmin=0, vmax=255, cmap='gray')
plt.show()
```

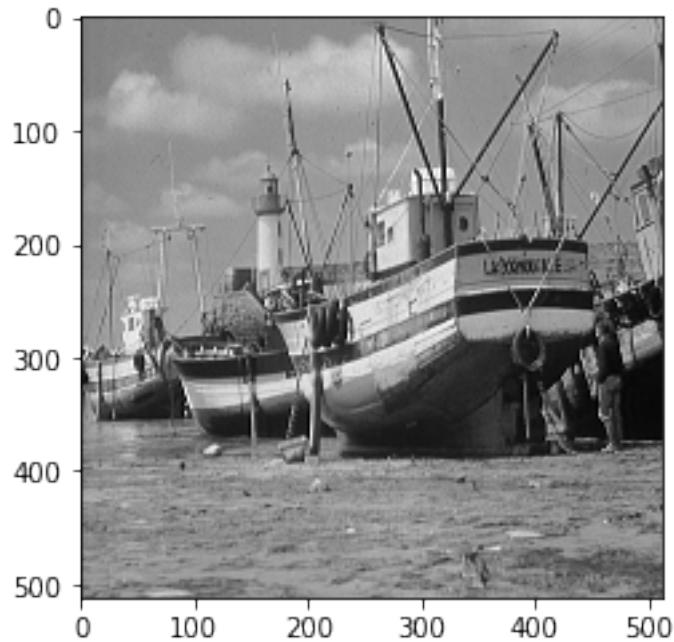


S obzirom na linearnost Lapasijan filtra, operaciju sabiranja rezultata filtriranja i originalne slike je moguće realizovati uvećanjem centralnog koeficijenta maske Laplasijana za 1. Upotrebom ovako modifikovanog filtra dobija se izoštrena slika kao rezultat filtriranja.

```
[18]: L_izostravanje = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
print(L_izostravanje)
```

```
[[ 0 -1  0]
 [-1  5 -1]
 [ 0 -1  0]]
```

```
[19]: img_izostren = ndimage.convolve(img.astype('float'), L_izostravanje, mode='reflect')
plt.imshow(img_izostren, vmin=0, vmax=255, cmap='gray')
plt.show()
```



3.4 Primer 6.4

Odrediti sliku gradijenata na slici `deda_mraz.png` upotrebom upotreboom Robertsovog kros-gradijentnog operatora (engl. *Roberts cross operator*). Odrediti sliku ivica iz dobijene slike binarizovanjem sa pragom $T=30$.

Robertsov kross operator je dat kao par kernela:

$$r_1 = \begin{bmatrix} -1 & 0 \\ 0 & +1 \end{bmatrix} \quad \text{i} \quad r_2 = \begin{bmatrix} 0 & -1 \\ +1 & 0 \end{bmatrix}$$

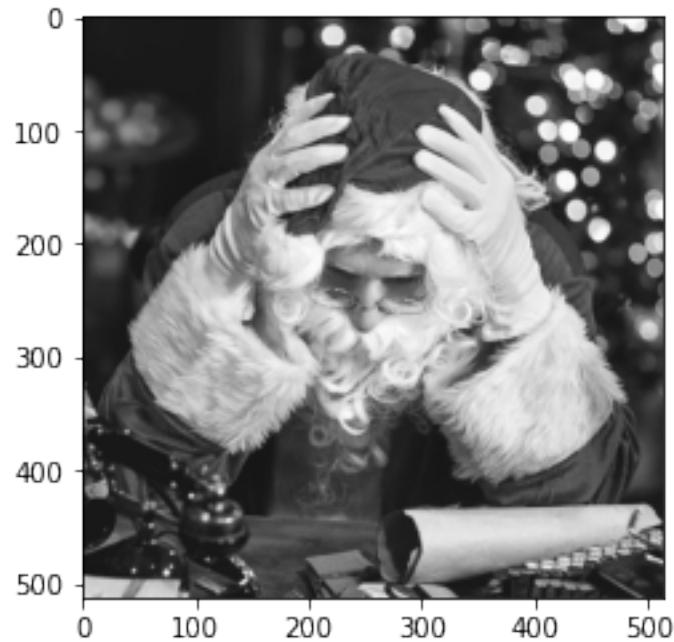
a gradijentna slika se formira kao:

$$G = \sqrt{G_1^2 + G_2^2}$$

gde su G_1 i G_2 slike nastale filtriranjem ulazne slike sa kernelima r_1 i r_2 .

```
[20]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np
from scipy import ndimage

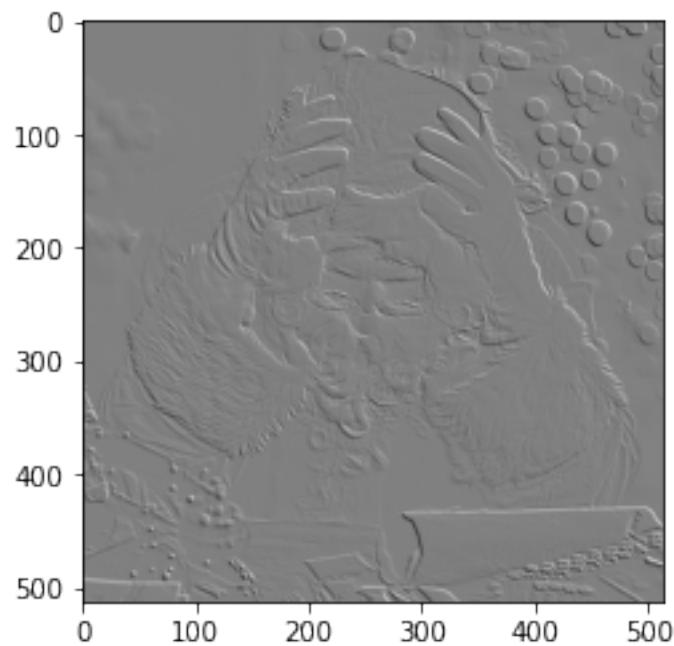
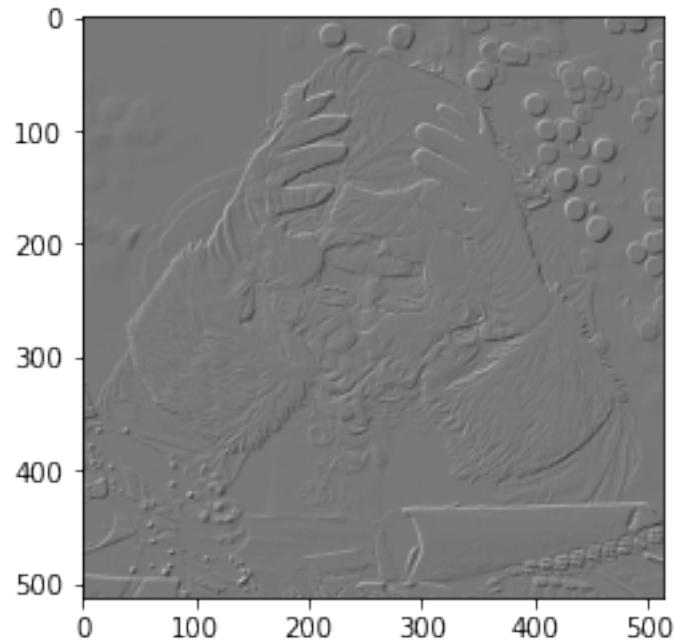
img = io.imread('deda_mraz.png')
plt.imshow(img, cmap='gray')
plt.show()
```



```
[21]: r1 = np.array([[1, 0],[0, -1]])
r2 = np.array([[0, 1],[-1, 0]])

img = img.astype('float')
G1 = ndimage.convolve(img, r1, mode='reflect')
plt.imshow(G1, cmap='gray')
plt.show()

G2 = ndimage.convolve(img, r2, mode='reflect')
plt.imshow(G2, cmap='gray')
plt.show()
```



```
[22]: G = np.sqrt(G1**2+G2**2)
plt.imshow(G, cmap='gray')
plt.show()
```

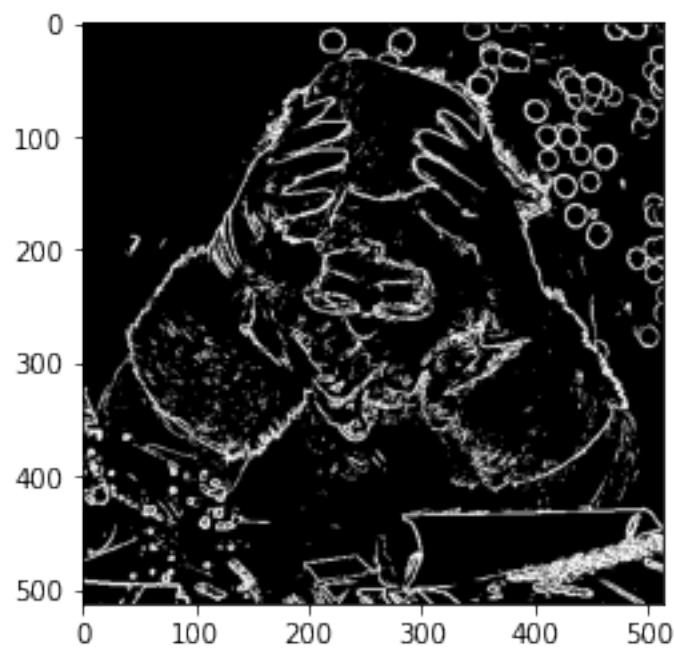
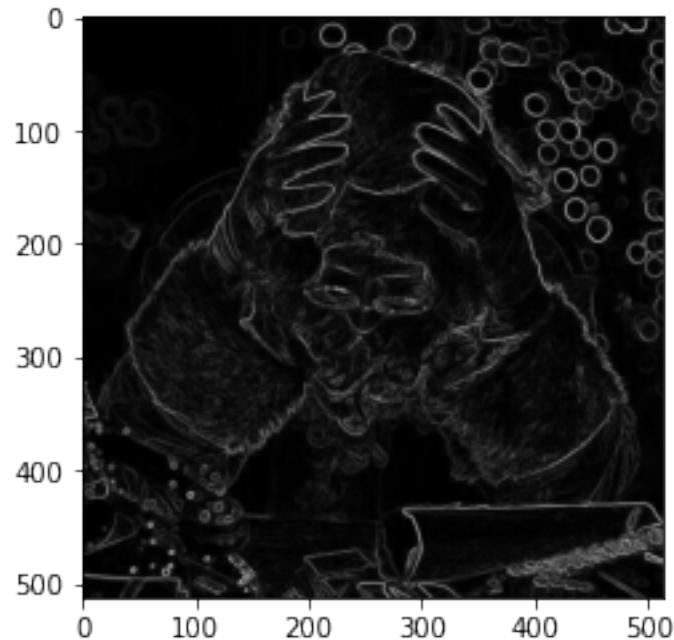
```
# binarizacija pragovanjem
```

```
T = 30
```

```
G_edge = G > T
```

```
plt.imshow(G_edge, cmap='gray')
```

```
plt.show()
```



3.5 Primer 6.5

Napisati funkciju `myImfilter` u kojoj je implementirano linearno filtriranje slike. Ulazni parametri su učitana slika, kernel i željeni način proširenja. Prepostaviti da je filter neparnih dimenzija.

Funkciju isprobati koristeći aritmetički usrednjivač dimenzija nad slikom `peppers.png`: * 21x21 * 11x5 * 3x7

```
[23]: import numpy as np
import math

def myImfilter(img, kernel, padding='symmetric'):
    out = np.zeros(img.shape)

    pad_rows = math.floor(kernel.shape[0]/2)
    pad_cols = math.floor(kernel.shape[1]/2)

    img_p = np.pad(img, ((pad_rows,), (pad_cols,)), mode=padding)

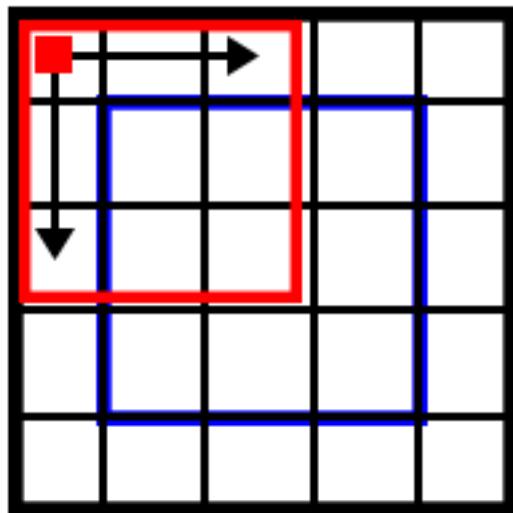
    for row in range(out.shape[0]):
        for col in range(out.shape[1]):
            region = img_p[row:row+kernel.shape[0], col:col+kernel.shape[1]]

            res = (region * kernel).sum()

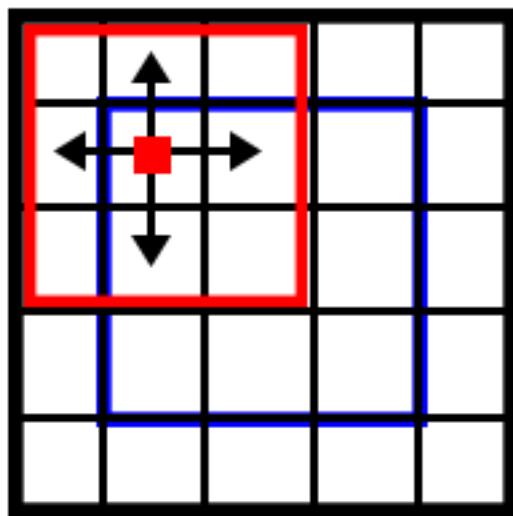
            out[row, col] = res

    return out
```

Ova verzija funkcije posmatra pojedinačne regije kao gornje-levu tačku sa odgovarajućom širinom i visinom u okviru proširene slike, ilustrovano u slici ispod za kernel dimezniye 3x3 (označen crveno) i polaznom slikom (uokvirenu plavom linijom)

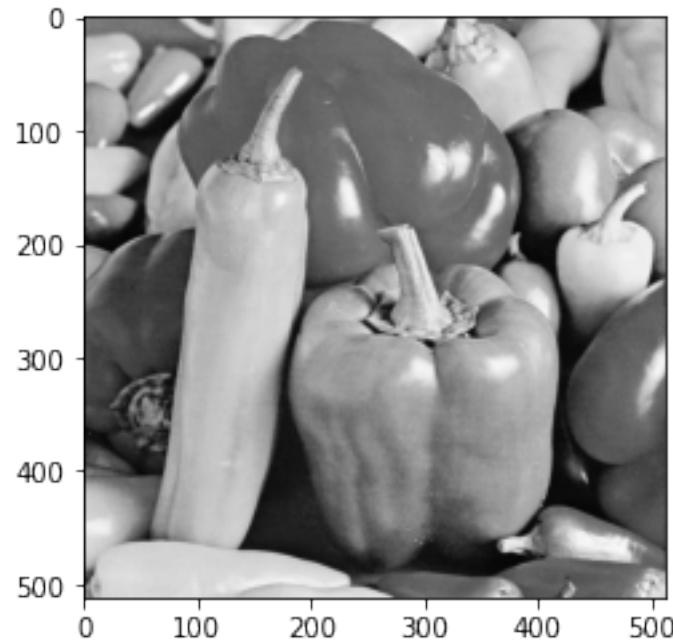


Alternativno, isti region se može posmatrati i kao centralna pozicija koju kernel pokriva sa odgovarajućom polu širinom i visinom, ilustrovano ispod. Oba načina daju identične rezultate.



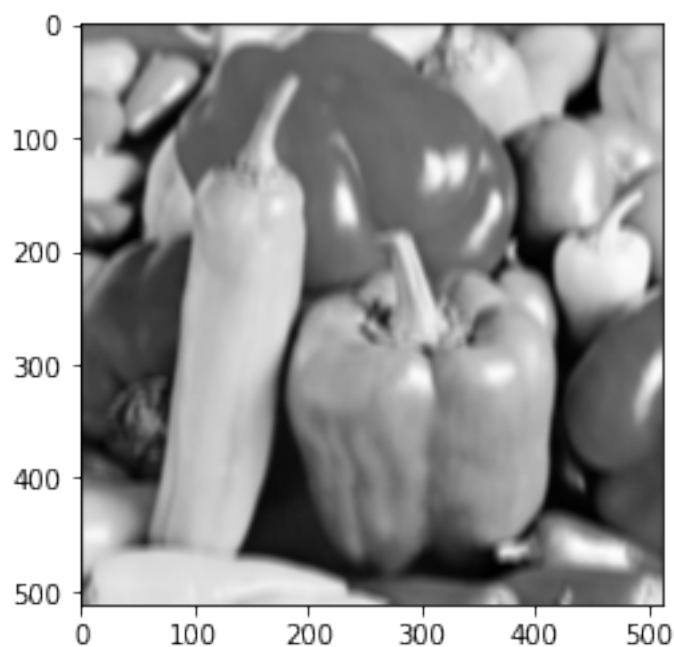
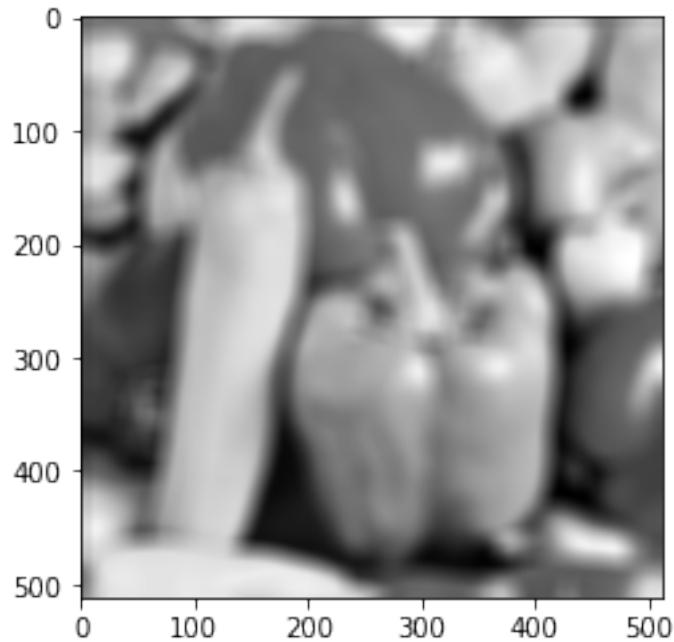
```
[24]: import matplotlib.pyplot as plt
from skimage import io
import numpy as np
```

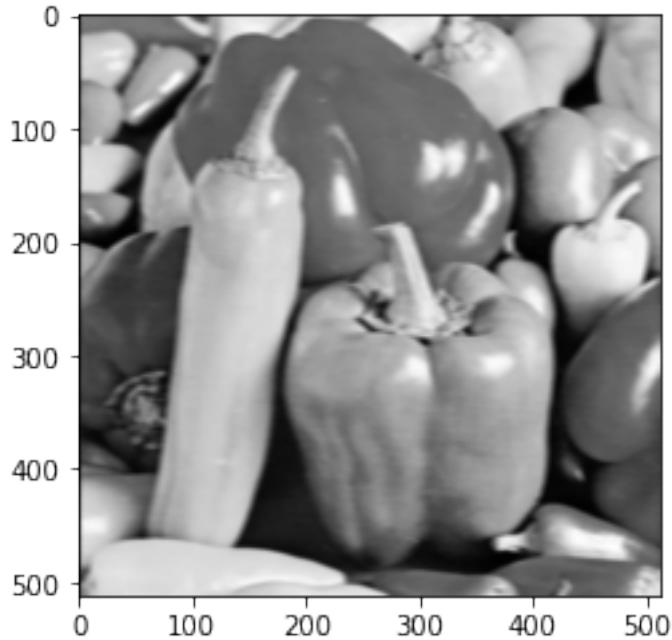
```
img = io.imread('peppers.png')
plt.imshow(img, cmap='gray')
plt.show()
```



```
[25]: kernel_shape = [(21,21), (11,5), (3,7)]
```

```
for shape in kernel_shape:
    w = np.ones(shape)/np.prod(shape)
    out = myImfilter(img, w, 'symmetric')
    plt.imshow(out, cmap='gray')
    plt.show()
```





4 Zadaci za samostalnu vežbu

Zadatak 6.1

Napisati funkciju `lapFilter` koja implementira izoštravanje slike Laplasijan filtrom koji uzima u obzir 8 suseda bez korišćenja ugrađenih funkcija za filtriranje. Funkciju upotrebiti nad proizvoljnom slikom. Zadatak rešiti koristeći alternativni način izdvajanja regiona prikazan u primeru 6.5.

[26] : #TODO

Zadatak 6.2

Izoštravanje medicinskih snimaka doprinosi vidljivosti sitnih anatomske detalja. Ovi detalji su od velikog značaja jer mogu da predstavljaju začetak malignih promena. Jedna od standardnih tehnika za izoštravanje slika je ansharp masking. Prvi korak je određivanje slike detalja koja se dobija kao razlika originalnog snimka i slike nastale njenim ublažavanjem:

$$I_{VF} = I - I_{NF}$$

gde I predstavlja originalni snimak, I_{VF} predstavlja sliku detalja, a I_{NF} označava ublaženu sliku. Izoštravanje slike postiže se na sledeći način:

$$I_{UM} = I + c \cdot I_{VF}$$

gde I_{UM} predstavlja izoštrenu sliku, a c predstavlja nenegativan koeficijent. Za vrednosti c veće od 0 dobija se izoštravanje slike. Napisati funkciju unsharpMasking koja implementira objašnjeno pojačanje detalja. Funkcija treba da ima tri ulazna argumenta. Prvi je slika koja treba da se izoštiri. Drugi ulazni argument određuje širinu filtra koji se koristi za ublažavanje (koristiti filter aritmetički usrednjivač). Treći ulazni argument određuje izoštravanje (koeficijent c). Smatrati da je ulazna slika tipa uint8. Funkciju iskoristiti da bi se izoštrela slika rtg_6.png. Prikazati izlaznu sliku u opsegu tonske skale od 0 do 255.

Napomena: zbog veličine slike koristiti matplotlib paket za prikaz.

[27] : `#TODO`

Zadatak 6.3

Napisati funkciju detectEdge koja pronalazi piksele kojima odgovaraju jaki gradijenti. Funkcija treba da ima dva ulazna argumenta. Prvi ulazni argument je slika, dok drugi predstavlja prag. Izlaz funkcije treba da bude binarna maska koja ima vrednost 1 na mestima gde je intenzitet gradijenta bio veći od praga, a 0 na ostalim mestima. Za procenu gradijenta koristiti Sobelov (engl. *Sobel*) gradijentni operator definisan sledećim maskama:

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad i \quad S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Funkciju isprobati nad slikom rtg_3.png.

Napomena: zbog veličine slike koristiti matplotlib paket za prikaz.

[28] : `#TODO`

5 Dokumentacija novih celina

- [numpy.pad](#)
- [SciPy](#)
- [scipy.signal.convolve2d](#)
- [scipy.ndimage.convolve](#)
- [math](#)
- [numpy.prod](#)

Vezba 7 - Poboljšanje slike u frekvencijskom domenu

1 Pregled

U okviru vežbe naučiće se postupci za spektralnu analizu slike u kontekstu određivanja DFT 2D matrice i načina prikaza amplitudske i fazne karakteristike. Ilustrovaće se određivanje spektralne slike delta impulsa i kosinusnog signala u 2D prostoru kao i konstrukcija najosnovnijih niskofrekventnih (NF) i visokofrekventnih (VF) filtara i prikaz njihovog uticaja na kvalitet slike.

2 Frekvencijska analiza

Spektar slike je određen 2D diskretnom Furijeovom transformacijom (DFT) data formulom:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})}$$

gde (u, v) predstavljaju prostorne koordinate DFT slike, (x, y) prostorne koordinate polazne slike f dimenzije $M \times N$.

Inverzna diskretna Furijeova transformacija je definisana sledećim izrazom:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(\frac{ux}{M} + \frac{vy}{N})}$$

DFT slika je u opštem slučaju kompleksnog tipa i može se predstaviti u obliku:

Amplitudski spektar predstavlja moduo DFT slike

$$|F(u, v)| = \sqrt{R^2(u, v) + I^2(u, v)}$$

dok fazni spektar predstavlja njen argument

$$\phi(u, v) = \operatorname{arctg} \frac{R(u, v)}{I(u, v)}$$

gde su R i I realni i imaginarni deo DFT slike respektivno.

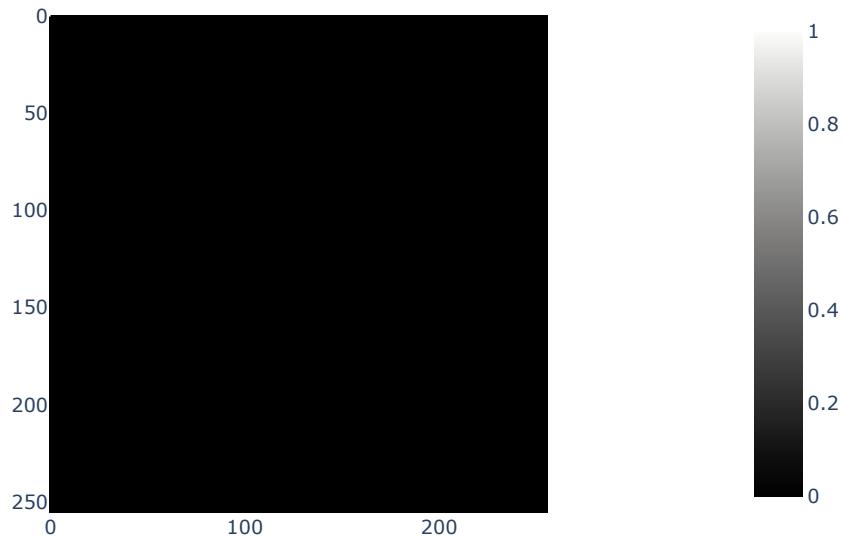
2.1 Primer 7.1

Formirati sliku dimenzije 256x256 koja ima vrednosti nula na svim pikselima osim na prvom, koji ima vrednost 1. Koristeći funkciju `numpy.fft.fft2`, odrediti i prikazati amplitudski i fazni spektar dobijene slike.

```
[2]: import plotly.express as px
import numpy as np

img = np.zeros((256,256))
img[0,0] = 1

fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[3]: IMG = np.fft.fft2(img)
print(IMG)
```

```
[[1.+0.j 1.+0.j 1.+0.j ... 1.+0.j 1.+0.j 1.+0.j]
 [1.+0.j 1.+0.j 1.+0.j ... 1.+0.j 1.+0.j 1.+0.j]
 [1.+0.j 1.+0.j 1.+0.j ... 1.+0.j 1.+0.j 1.+0.j]
 ...
 [1.+0.j 1.+0.j 1.+0.j ... 1.+0.j 1.+0.j 1.+0.j]
 [1.+0.j 1.+0.j 1.+0.j ... 1.+0.j 1.+0.j 1.+0.j]]
```

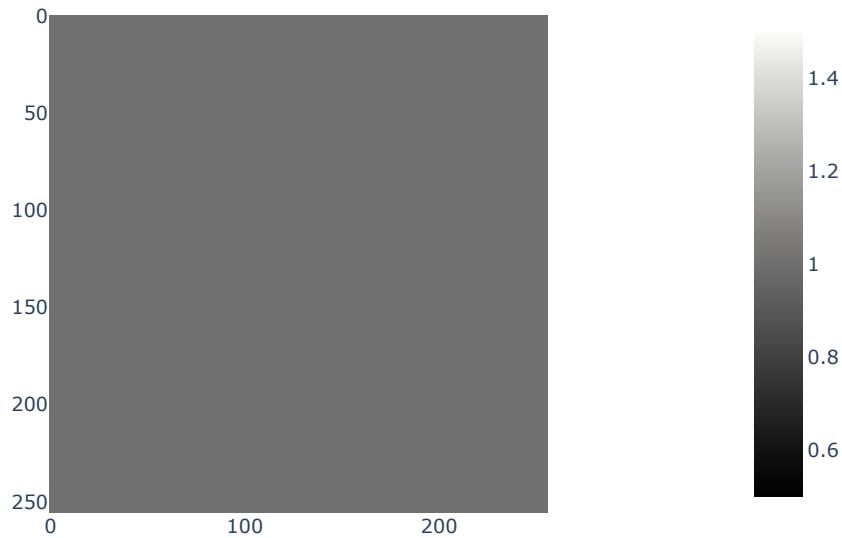
```
[1.+0.j 1.+0.j 1.+0.j ... 1.+0.j 1.+0.j 1.+0.j]]
```

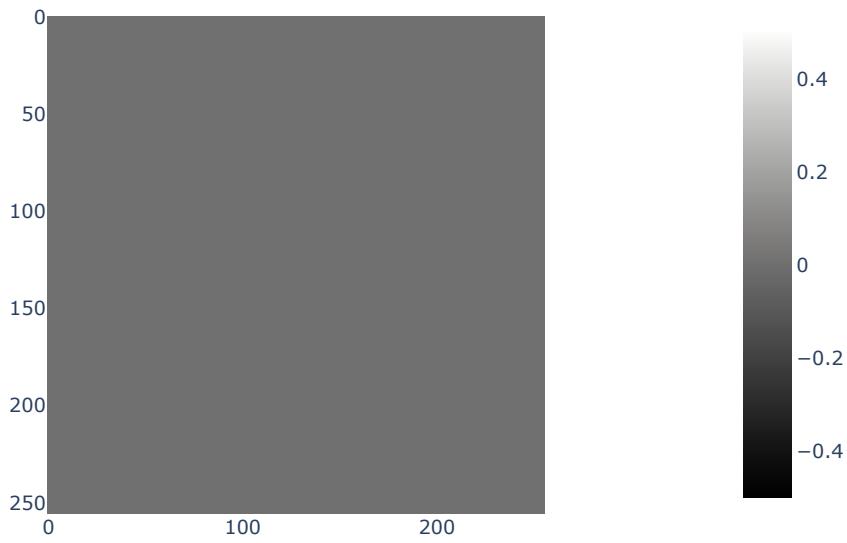
Funkcija `numpy.fft.fft2` računa dvodimenzionalnu brzu diskretnu Furijeovu transformaciju. Dodatni parametri funkcije mogu obezbediti broj tačaka u kome se računa DFT. Za parametar veći od dimenzije slike, slika se proširuje sa zero-paddingom proširenjem vrsta sa donje i kolona sa desne strane. Slika je u tom slučaju finija, ali ne nosi više informacije nego originalni račun (sve nove tačke su suštinski nastale interpolacijom).

Za određivanje amplitudskog i faznog spektra, mogu se koristiti funkcije `numpy.abs` i `numpy.angle`.

```
[4]: AMP = np.abs(IMG)
PHASE = np.angle(IMG)

fig = px.imshow(AMP, color_continuous_scale='gray')
fig.show()
fig = px.imshow(PHASE, color_continuous_scale='gray')
fig.show()
```





Dobijene slike predstavljaju DFT Dirakovog delta impulsa u 2D prostoru. Dobijena amplitudska karakteristika ukazuje da je spektar delta impulsa konstanta i vrednosti 1, što odgovara slučaju koji je rađen u okviru 1D signala. Faza, kao što se vidi sa slike je 0.

Probati prethodni primer sa pomerenom lokacijom delta impulsa i posmatrati rezultat.

2.2 Primer 7.2

2.2.1 $\pi/4$

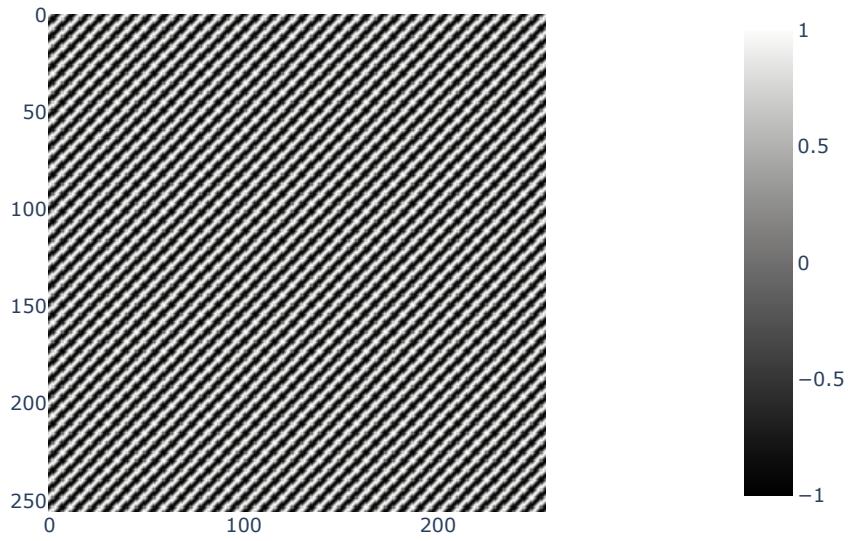
Formirati sliku dimenzije 256x256 kosinusoide učestanosti $\pi/4$ sa promenama po vrstama i kolonama. Odrediti DFT slike i prikazati njen amplitudski spektar.

```
[5]: import plotly.express as px
import numpy as np

# definisanje opsega promena po vrstama (y) i kolonama (x)
y = np.arange(0, 256, 1).reshape(-1, 1)
x = np.arange(0, 256, 1)

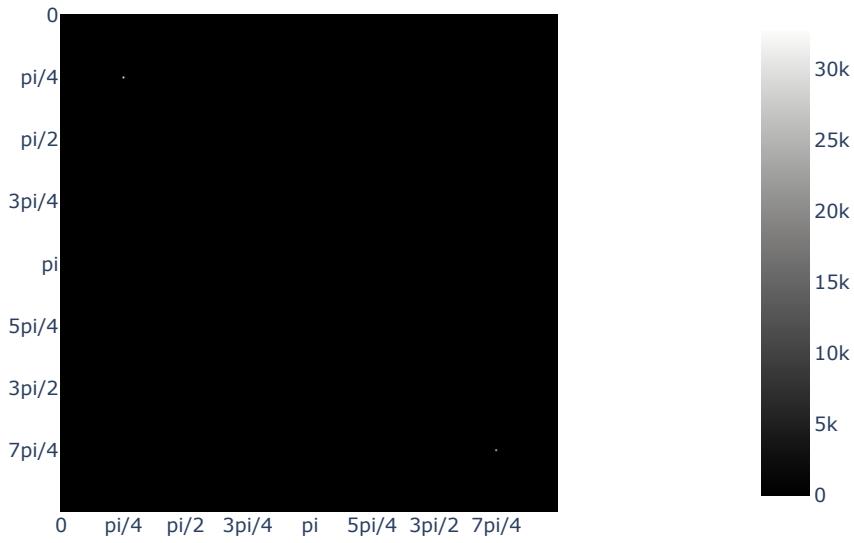
img = np.cos((y+x)*np.pi/4)

fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[6]: IMG = np.fft.fft2(img)
AMP = np.abs(IMG)

fig = px.imshow(AMP, color_continuous_scale='gray')
fig.update_yaxes(tickvals=np.arange(0,256,32),
                  ticktext=['0','pi/4','pi/2','3pi/4','pi','5pi/4','3pi/2','7pi/
→4'])
fig.update_xaxes(tickvals=np.arange(0,256,32),
                  ticktext=['0','pi/4','pi/2','3pi/4','pi','5pi/4','3pi/2','7pi/
→4'])
fig.show()
```



Prostor koordinata (u, v) se može posmatrati u opsegu od 0 do 2π (kružne učestanosti) sa koordinatnim početkom u gornjem levom uglu. Očekivani rezultat DFT-a kosinusnog signala je par delta impulsa pa poziciji $+/-$ učestanost kosinusa. Posmatrajući amplitudski spektar sa obeleženim osama u radijanima može se uočiti da su delta impulsi upravo pozicionirani na odgovarajućim učestanostima i ukazuju da se promene dešavaju podjenako na y i x osi.

2.2.2 π

Formirati sliku dimenzije 256×256 kosinusoide učestanosti π sa promenama po vrstama i kolonama. Odrediti DFT slike i prikazati amplitudsku i faznu karakteristiku.

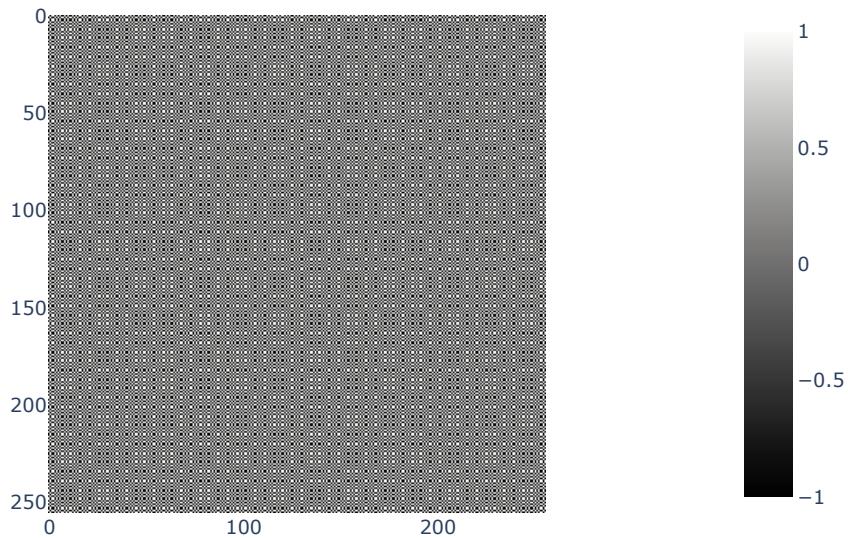
```
[7]: import plotly.express as px
import numpy as np

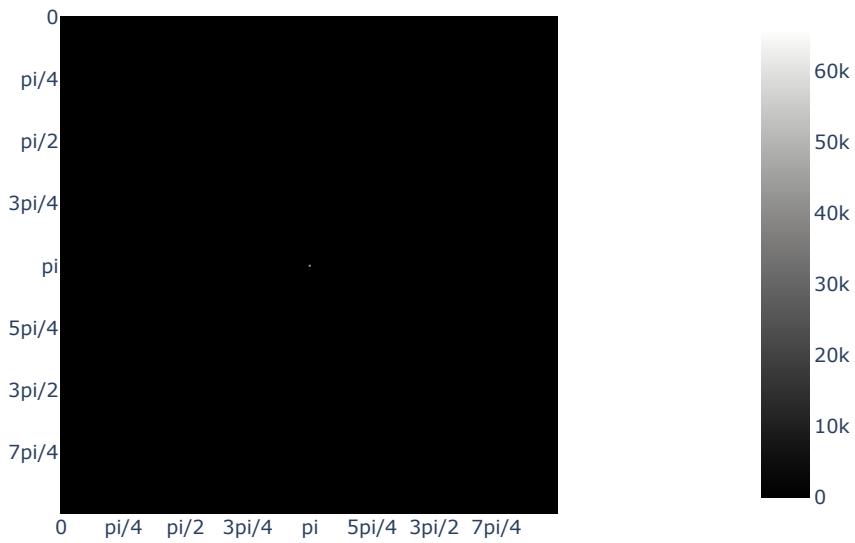
y = np.arange(0, 256, 1).reshape(-1, 1)
x = np.arange(0, 256, 1)
img_pi = np.cos((y+x)*np.pi) # ili = (-1)**(x+y)

fig = px.imshow(img_pi, color_continuous_scale='gray')
fig.show()

IMG_PI = np.fft.fft2(img_pi)
AMP = np.abs(IMG_PI)
```

```
fig = px.imshow(AMP, color_continuous_scale='gray')
fig.update_yaxes(tickvals=np.arange(0,256,32),
                  ticktext=['0','pi/4','pi/2','3pi/4','pi','5pi/4','3pi/2','7pi/
→4'])]
fig.update_xaxes(tickvals=np.arange(0,256,32),
                  ticktext=['0','pi/4','pi/2','3pi/4','pi','5pi/4','3pi/2','7pi/
→4'])]
fig.show()
```





2.3 Primer 7.3

Formirati sliku dimenzije 256x256 koja ima položen beli pravougaonik u njenoj sredini. Odrediti DFT slike i prikazati njen amplitudski spektar.

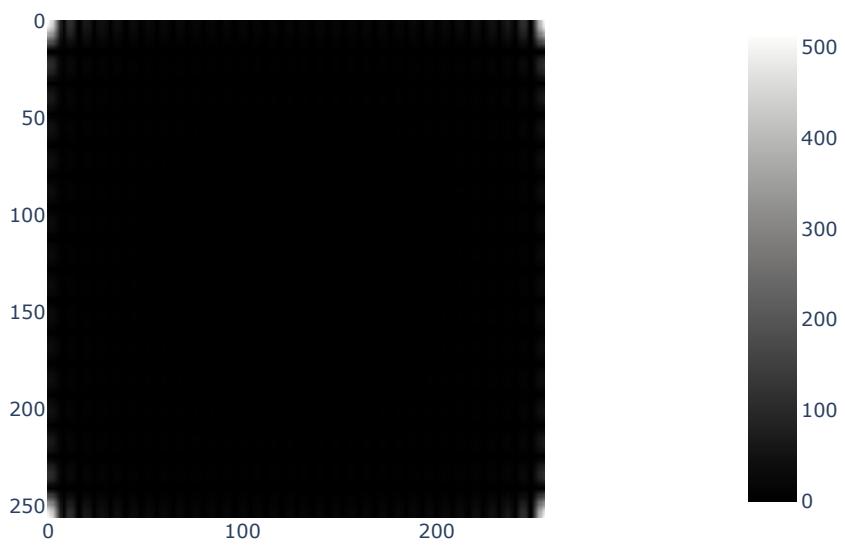
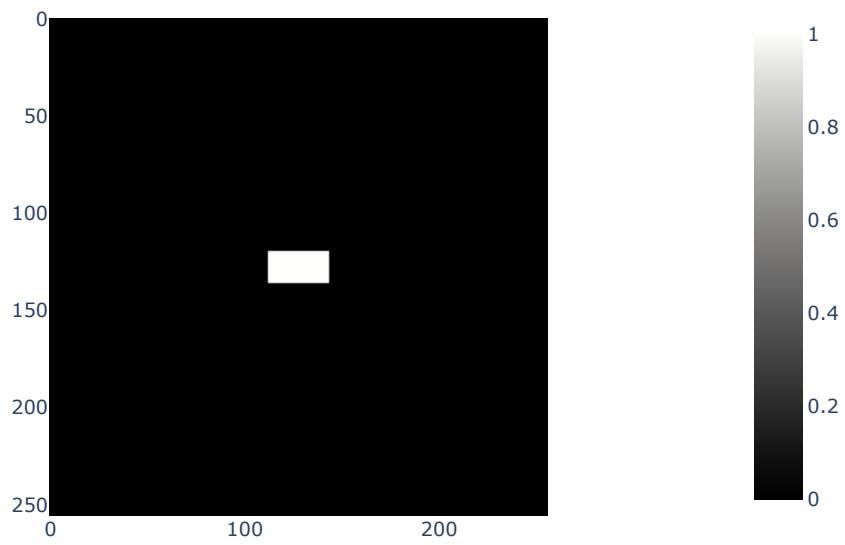
```
[8]: import plotly.express as px
import numpy as np

img = np.zeros((256,256))
img[128-8:128+8,128-16:128+16] = 1

fig = px.imshow(img, color_continuous_scale='gray')
fig.show()

IMG = np.fft.fft2(img)
AMP = np.abs(IMG)

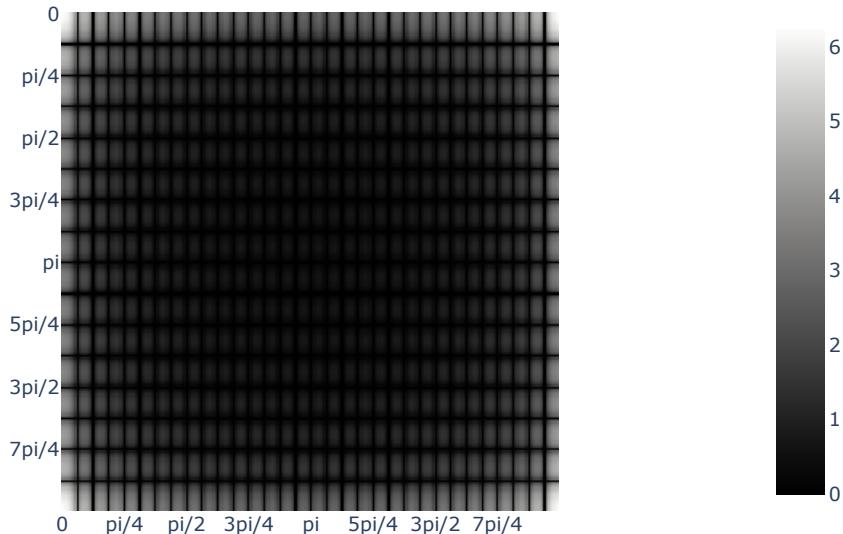
fig = px.imshow(AMP, color_continuous_scale='gray')
fig.show()
```



Prikazana slika je prilično tamna, iz razloga što amplitudski spektar većine slika ima izuzetno

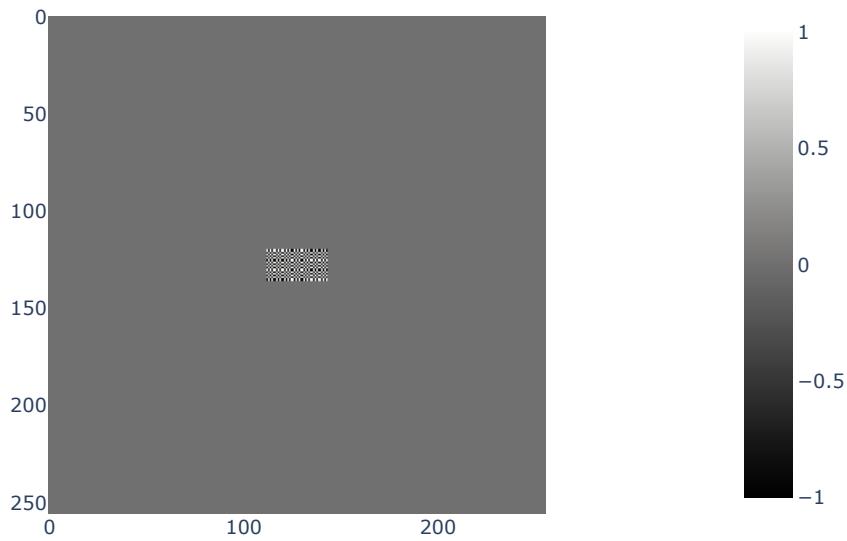
velik opseg vrednosti koji je nemoguće prikazati nijansama sive na standardan način. Ovaj problem se prevazilazi logaritamskom kompresijom opsega. Znak +1 je u okviru funkcije logaritma iz razloga da bi se izbeglo potencijalno računanje logaritma u 0.

```
[9]: fig = px.imshow(np.log(AMP+1), color_continuous_scale='gray')
fig.update_yaxes(tickvals=np.arange(0,256,32),
                  ticktext=['0','pi/4','pi/2','3pi/4','pi','5pi/4','3pi/2','7pi/
→4'])
fig.update_xaxes(tickvals=np.arange(0,256,32),
                  ticktext=['0','pi/4','pi/2','3pi/4','pi','5pi/4','3pi/2','7pi/
→4'])
fig.show()
```



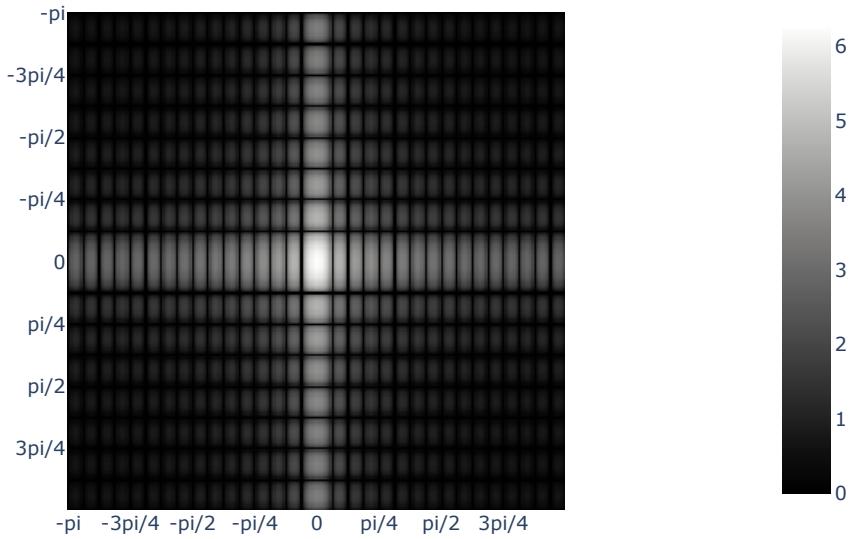
Pomnožiti originalnu sliku sa slikom kosinusoide učestanosti π iz prethodnog primera. Odrediti DFT slike i prikazati njenu amplitudsku i faznu karakteristiku.

```
[10]: img2 = img * img_pi
fig = px.imshow(img2, color_continuous_scale='gray')
fig.show()
```



```
[11]: IMG2 = np.fft.fft2(img2)

fig = px.imshow(np.log(np.abs(IMG2)+1), color_continuous_scale='gray')
fig.update_yaxes(tickvals=np.arange(0,256,32),
                  ticktext=['-pi','-3pi/4','-pi/2','-pi/4', '0','pi/4','pi/2','3pi/4'])
fig.update_xaxes(tickvals=np.arange(0,256,32),
                  ticktext=['-pi','-3pi/4','-pi/2','-pi/4', '0','pi/4','pi/2','3pi/4'])
fig.show()
```



Dobijeni prikaz amplitudske karakteristike ima jednosmernu komponentu postavljenu u sredini slike.

Jednosmerna komponenta može se postaviti u centar slike spektra na dva načina. Prvi je množenjem ulazne slike sa kosinusoidom učestanosti π po y i x osi (ili slikom $(-1)^{x+y}$). Množenjem sa ovakvom slikom u prostornom domenu ima posledicu u spektralnom domenu konvolucije njihovih karakteristika. Pošto smo videli kako izgleda amplitudski spektar ovakve kosinusne slike, jasno je da će konvolucija obezbediti translaciju svih komponenti za faktor π . Ovim načinom komponente niskih učestanosti su pomerene u centar slike, a visoke na krajeve slike. Za neutralizaciju ovog efekta potrebno je sliku ponovo pomnožiti sa $(-1)^{x+y}$.

Drugi način za postavljanje jednosmerne komponente u centar slike spektra je jednostavnim preuređenjem izračunatog spektra: gornja leva i donja desna četvrtina, kao i gornja desna i donja leva četvrtina spektra treba da zamene mesta. Ovo se postiže upotrebom funkcije `numpy.fft.fftshift`. Potrebno je primetiti da je zamena mesta delova spektra računarski efikasnija operacija od množenja slike sa $(-1)^{x+y}$ pošto zahteva samo kopiranje (računarski brza operacija), dok alternativni pristup zahteva množenje piksela nekom vrednošću koja se kreira za svaki piksel (operacija sporija od kopiranja). Ako je za postavljanje jednosmerne komponente u centar slike spektra upotrebljena funkcija `fftshift`, pre određivanja inverzne Furijeove transformacije je spektar potrebno vratiti u početni položaj upotrebom funkcije `numpy.fft.ifftshift`.

2.4 Primer 7.4

Učitati sliku `lena.png` i odrediti njenu DFT. Postaviti niske učestanosti u sredinu slike koristeci `numpy.fft.fftshift` funkciju. Prikazati amplitudski i fazni spektar.

```
[12]: import plotly.express as px
from skimage import io
import numpy as np

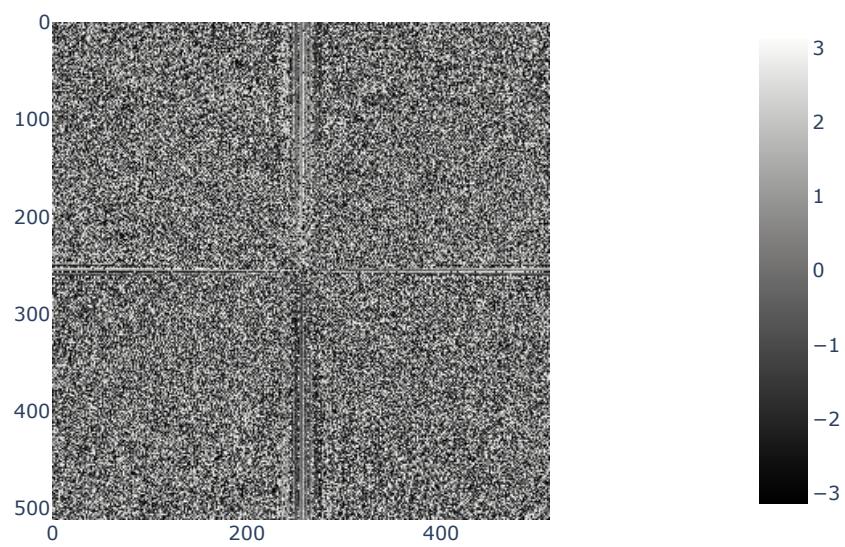
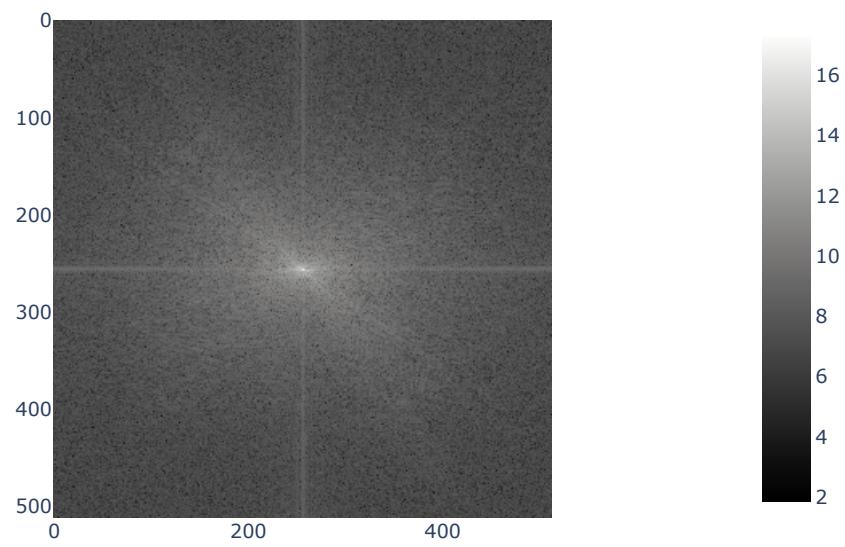
img = io.imread('lena.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[13]: IMG = np.fft.fft2(img)
IMG = np.fft.fftshift(IMG)

fig = px.imshow(np.log(np.abs(IMG)+1), color_continuous_scale='gray')
fig.show()

fig = px.imshow(np.angle(IMG), color_continuous_scale='gray')
fig.show()
```



2.5 Primer 7.5

2.5.1 Idealni

Napisati funkciju `ideal_nf` za racunanje idealnog NF filtra definisan izrazom:

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

gde su (u, v) prostorne koordinate DFT slike, D je euklidsko rastojanje posmatrano od sredine slike a D_0 je poluprečnik.

Funkcija prima argument veličine filtra `H_size` (u formi tuple vrednosti) i parametar `D0`.

Konstruisati filter za parametar veličine (256,256) i poluprečnika 50. Prikazati amplitudski spektar filtra.

```
[14]: import numpy as np

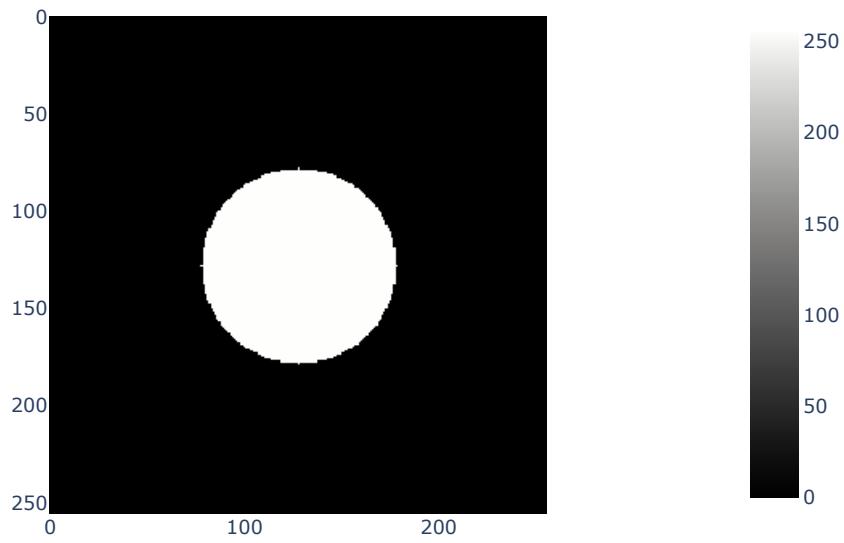
def ideal_nf(H_size,D0):
    u = np.arange(H_size[0]).reshape(-1, 1) - np.floor(H_size[0]/2)
    v = np.arange(H_size[1]) - np.floor(H_size[1]/2)

    D = np.sqrt(u**2 + v**2)
    H = D <= D0
    return H
```

Za formiranje ovog filtra potrebno je definisati Euklidsko rastojanje u okviru frekvencijske ravni. Ovde koristimo identičan postupak rađen u vežbi 3, definisanjem opsega vrednosti za vrste i kolone. Da bi obezbedili da centar izračunatog euklidskog rastojanja bude u sredini slike, vrednosti inicijalnih opsega promena po vrstama i kolonama (koje kreću od 0) potrebno je umanjiti za polovinu odgovarajuće dimenzije. Zaokruživanje na donje celo obezbeđuje korektni račun za parne i neparne dimenzije filtra.

```
[15]: import plotly.express as px

H1 = ideal_nf((256,256), 50)
fig = px.imshow(np.abs(H1), color_continuous_scale='gray')
fig.show()
```



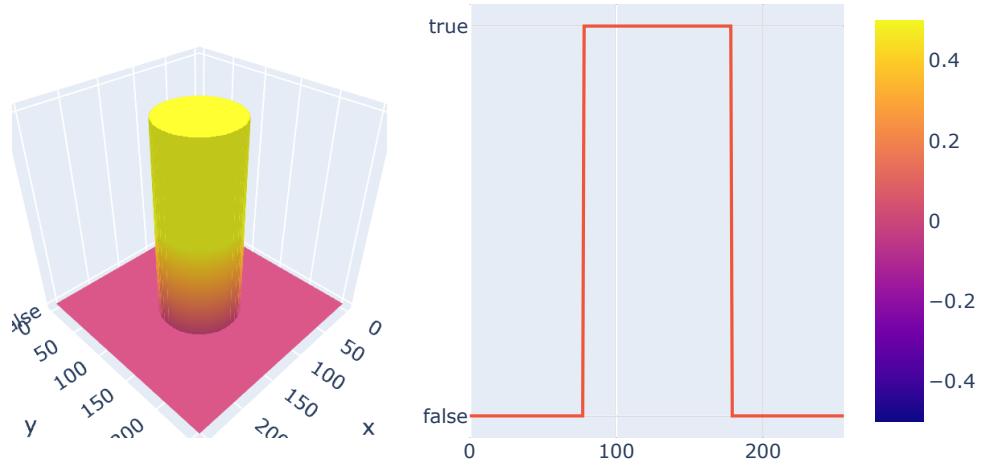
```
[16]: import plotly.graph_objects as go
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'surface'}, {'type': 'scatter'}]]) # mora se naznaciti koja vrsta plota ako se razlikuju

# 3D prikaz
fig.add_trace(go.Surface(z=np.abs(H1)), row=1, col=1)

# poprecni presek
fig.add_trace(go.Scatter(y=np.abs(H1)[128], mode='lines'), row=1, col=2)

fig.show()
```



Amplitudska karakteristika je ovde prikazana na dva dodatna načina. Slika levo prikazuje u okviru 3D prostora korišćenjem [Surface](#) grafičkog objekta, dok slika desno prikazuje kao poprečni presek (koji odgovara predstavi NF filtra za 1D signale). Za potrebe prikazivanja 2 različita tipa objekata u okviru subplot-a, potrebno je definisati dodatan parametar `specs` u okviru `make_subplots` funkcije.

2.5.2 Gausov

Napisati funkciju `gauss_nf` za racunanje Gausovog NF filtra definisan izrazom:

$$H(u, v) = e^{-\frac{D^2(u,v)}{2D_0^2}}$$

gde su (u, v) prostorne koordinate DFT slike, D je euklidsko rastojanje posmatrano od sredine slike a D_0 je poluprečnik.

Funkcija prima argument veličine filtra `H_size` (u formi tuple vrednosti) i parametar `D0`.

Konstruisati filter za parametar veličine (256,256) i poluprečnika 50. Prikazati amplitudsku karakteristiku filtra.

```
[17]: import numpy as np

def gauss_nf(H_size,D0):
    u = np.arange(H_size[0]).reshape(-1, 1) - np.floor(H_size[0]/2)
    v = np.arange(H_size[1]) - np.floor(H_size[1]/2)
```

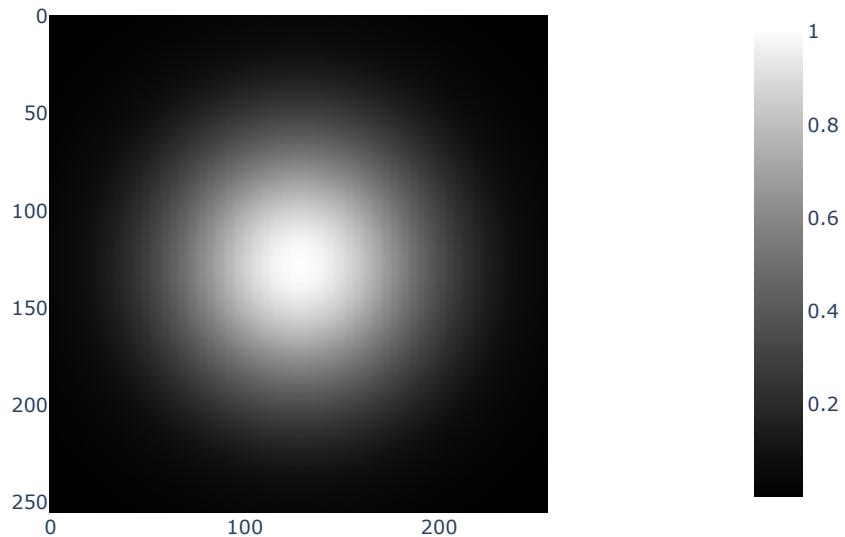
```

D = np.sqrt(u**2 + v**2)
H = np.exp(-D**2/(2*D0**2))
return H

```

```
[18]: import plotly.express as px

H2 = gauss_nf((256,256), 50)
fig = px.imshow(np.abs(H2), color_continuous_scale='gray')
fig.show()
```



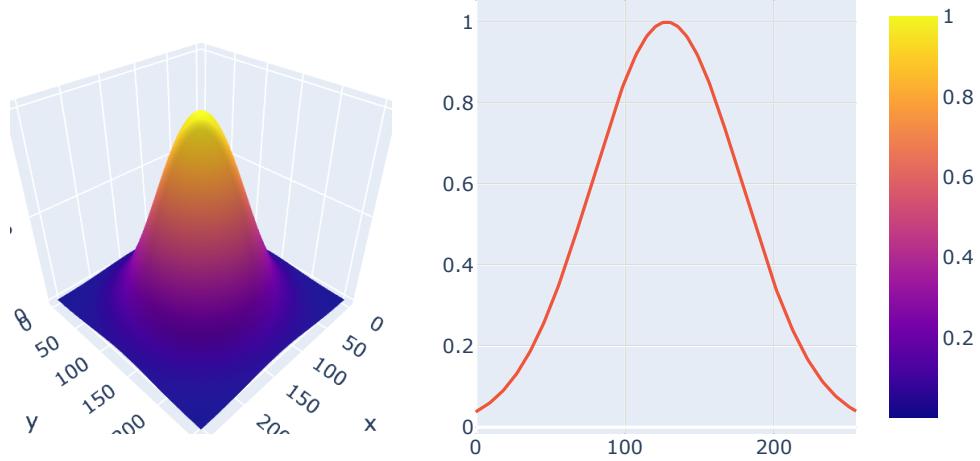
```
[19]: import plotly.graph_objects as go
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'surface'}, {'type': 'scatter'}]]) # mora se naznaciti koja vrsta plota ako se razlikuju

# 3D prikaz
fig.add_trace(go.Surface(z=np.abs(H2)), row=1, col=1)

# poprecni presek
fig.add_trace(go.Scatter(y=np.abs(H2)[128], mode='lines'), row=1, col=2)

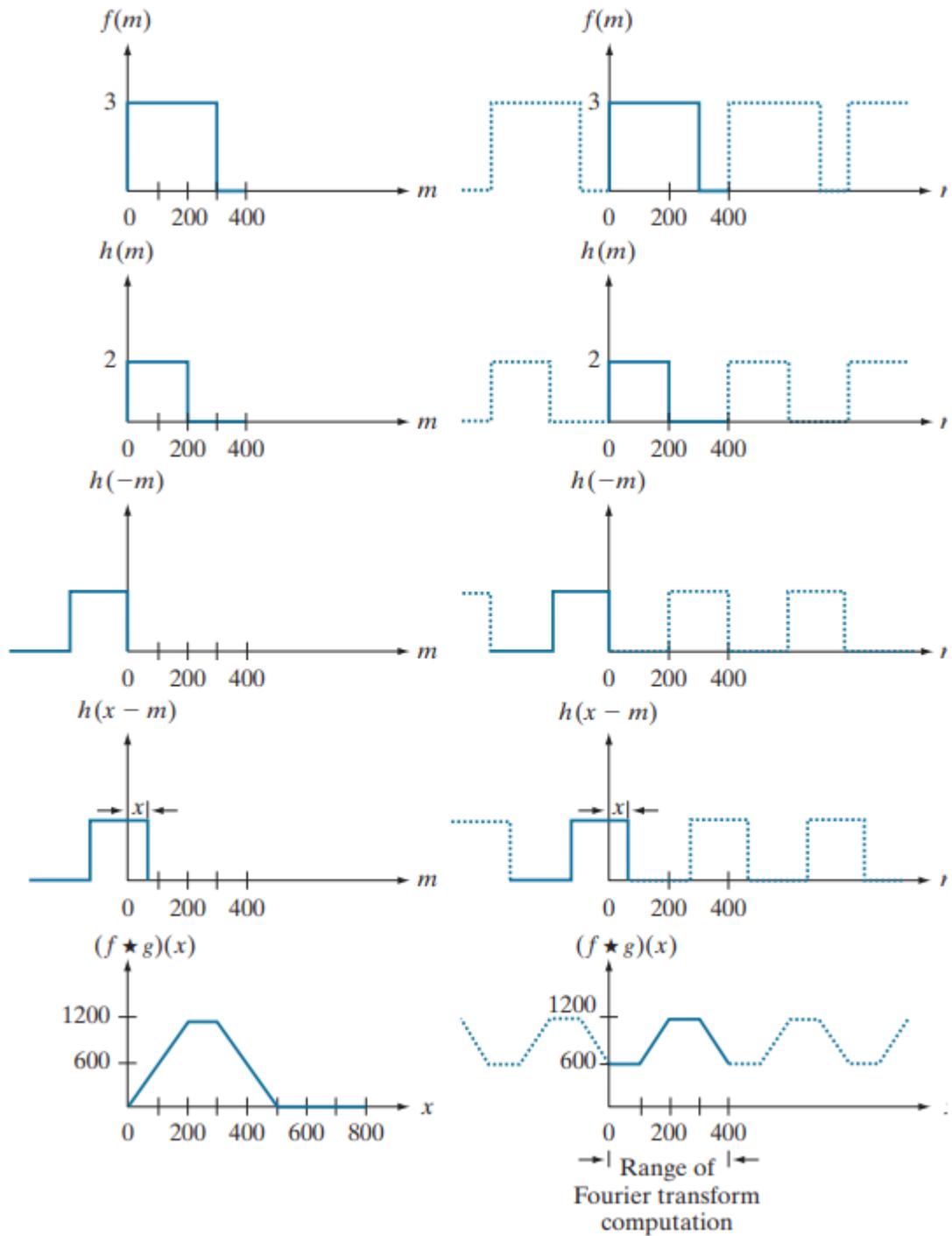
fig.show()
```



2.6 Primer 7.6

Formirati demo sliku dimenzije 256×256 koja ima 5 tačaka raspoređenih kao na kockici za bacanje. Izvršiti filtriranje ove slike sa idealnim NF filtrom sa parametrom $D0=50$ i prikazati rezultat.

Napomena: U okviru filtriranja u frekvencijskom domenu, moramo da pripazimo da množenje u frekvencijskom odgovara u stvari cikličnoj konvoluciji u prostornom domenu. Da bi obezbedili dovoljno prostora da rezultat ciklične konvolucije nema preklapanja sa ponovljenim elementima (tzv. *wraparound error*) potrebno je ulaznu sliku proširiti nekim vrednostima pre računanja DFT-a. Obično se koristi zero-padding koji je ugrađen u okviru `fft2` funkcije.



slika preuzeta iz knjige "Digital Image Processing", Gonzalez, Woods, fig 4.27, strana 255.

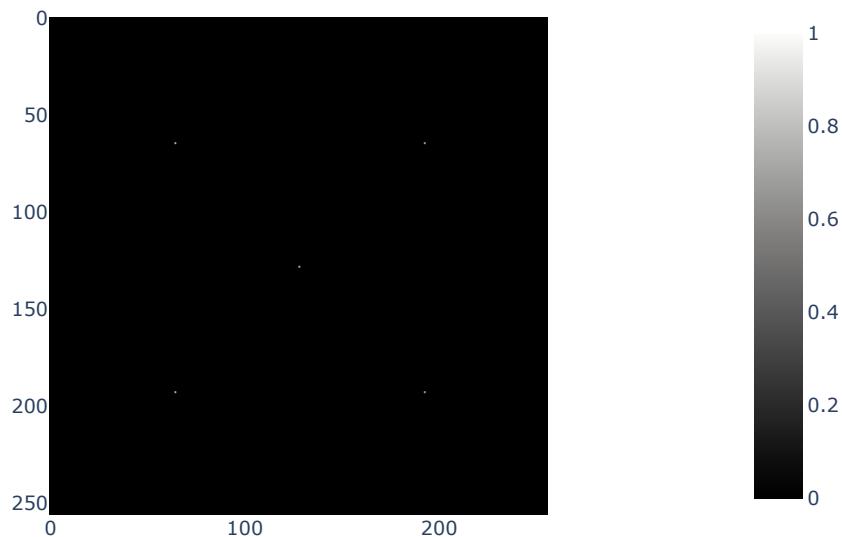
[20]: `import numpy as np`

```
img = np.zeros((256,256))
img[128,128] = 1
```

```



```



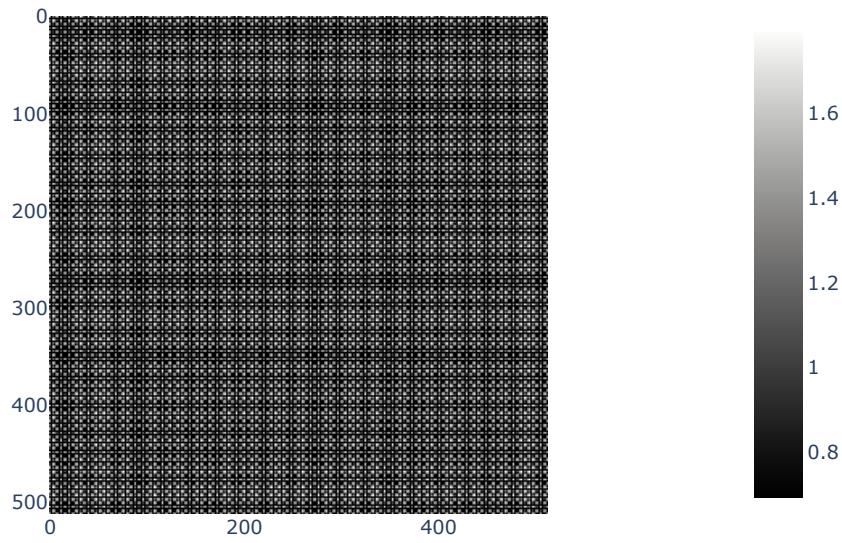
Proširenje sa nulama je obezbeđeno u okviru `fft2` funkcije, sa dodatnim parametrom `s` koji se navodi u obliku tuple vrednosti za željeni broj tačaka po svakoj dimenziji. Proširenje je sa donje i desne strane.

[21]:

```

IMG = np.fft.fftshift(np.fft.fft2(img, s=(2*img.shape[0],2*img.shape[1])))
fig = px.imshow(np.log(np.abs(IMG)+1), color_continuous_scale='gray')
fig.show()

```

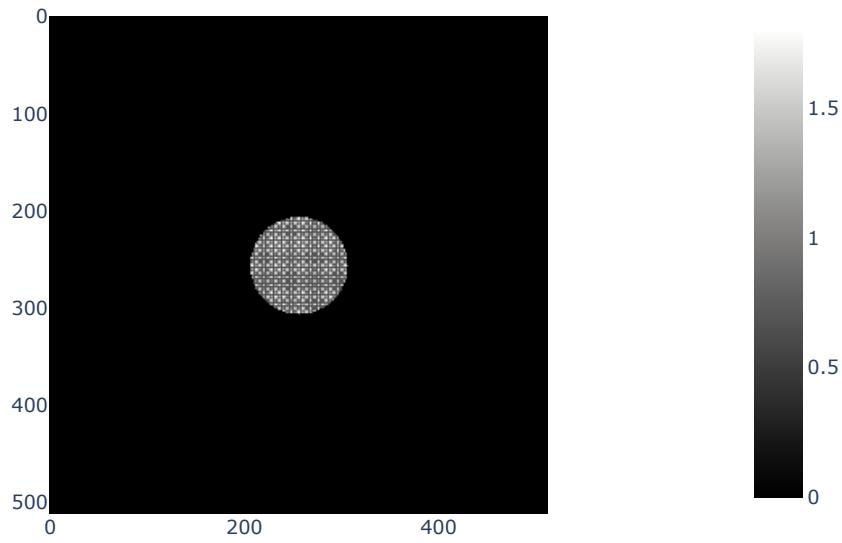


2.6.1 Idealni

```
[22]: H1 = ideal_nf(IMG.shape,50)

# filtriranje
IMG1 = IMG * H1

fig = px.imshow(np.log(np.abs(IMG1)+1), color_continuous_scale='gray')
fig.show()
```



Inverznu DFT možemo računati pomoću `numpy.fft.ifft2` funkcije. Ovde treba napomenuti, pošto počinjemo sa realnom slikom i održavamo simetričnost u spektru, rezultujuća slika nastala preko inverzne transformacije treba biti takođe realna. Funkcija `uvec` vraća kompleksnu sliku, tako da je potrebno samo preuzeti realne vrednosti, što je demonstrirano u primeru. Ponekad se desi da zbog grešaka zaokruživanja ima parazitskih imaginarnih komponenti, koje zanemarujuemo.

```
[23]: img1 = np.fft.ifft2(np.fft.ifftshift(IMG1))
print(img1)
img1 = np.real(img1)
print(img1)

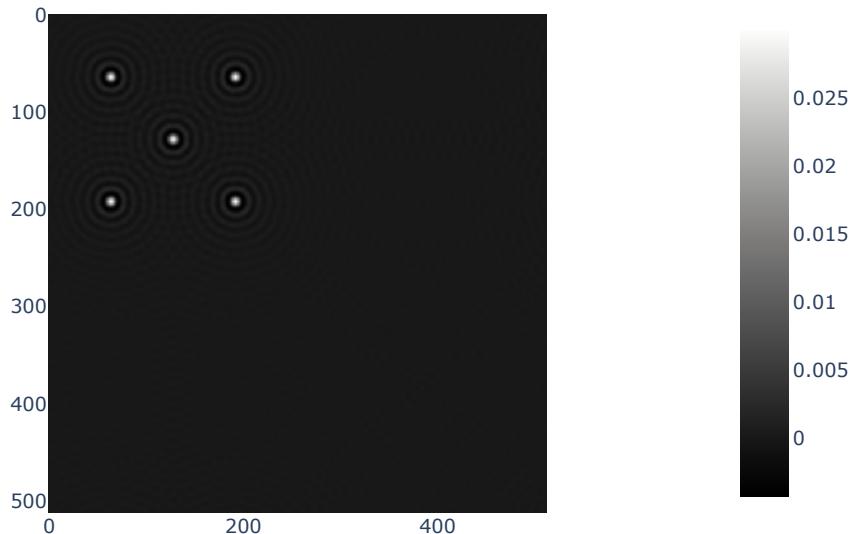
fig = px.imshow(img1, color_continuous_scale='gray')
fig.show()
```

$$\begin{bmatrix} [-1.64031982e-04+0.j \ -9.09478483e-05+0.j \ -2.27036572e-06+0.j \ \dots \\ -1.77038720e-04+0.j \ -2.10207685e-04+0.j \ -2.06370413e-04+0.j] \\ [-9.09478483e-05+0.j \ 1.31679777e-06+0.j \ 9.43745720e-05+0.j \ \dots \\ -2.03024550e-04+0.j \ -2.02859800e-04+0.j \ -1.63607159e-04+0.j] \\ [-2.27036572e-06+0.j \ 9.43745720e-05+0.j \ 1.76825924e-04+0.j \ \dots \\ -1.90527079e-04+0.j \ -1.59644452e-04+0.j \ -9.29485326e-05+0.j] \\ \dots \\ [-1.77038720e-04+0.j \ -2.03024550e-04+0.j \ -1.90527079e-04+0.j \ \dots \\ 7.22530708e-05+0.j \ -2.47040427e-05+0.j \ -1.13967602e-04+0.j] \\ [-2.10207685e-04+0.j \ -2.02859800e-04+0.j \ -1.59644452e-04+0.j \ \dots \\ -2.47040427e-05+0.j \ -1.10988229e-04+0.j \ -1.77681283e-04+0.j] \end{bmatrix}$$

```

[-2.06370413e-04+0.j -1.63607159e-04+0.j -9.29485326e-05+0.j ...
-1.13967602e-04+0.j -1.77681283e-04+0.j -2.11352052e-04+0.j]]
[[-1.64031982e-04 -9.09478483e-05 -2.27036572e-06 ... -1.77038720e-04
-2.10207685e-04 -2.06370413e-04]
[-9.09478483e-05 1.31679777e-06 9.43745720e-05 ... -2.03024550e-04
-2.02859800e-04 -1.63607159e-04]
[-2.27036572e-06 9.43745720e-05 1.76825924e-04 ... -1.90527079e-04
-1.59644452e-04 -9.29485326e-05]
...
[-1.77038720e-04 -2.03024550e-04 -1.90527079e-04 ... 7.22530708e-05
-2.47040427e-05 -1.13967602e-04]
[-2.10207685e-04 -2.02859800e-04 -1.59644452e-04 ... -2.47040427e-05
-1.10988229e-04 -1.77681283e-04]
[-2.06370413e-04 -1.63607159e-04 -9.29485326e-05 ... -1.13967602e-04
-1.77681283e-04 -2.11352052e-04]]

```

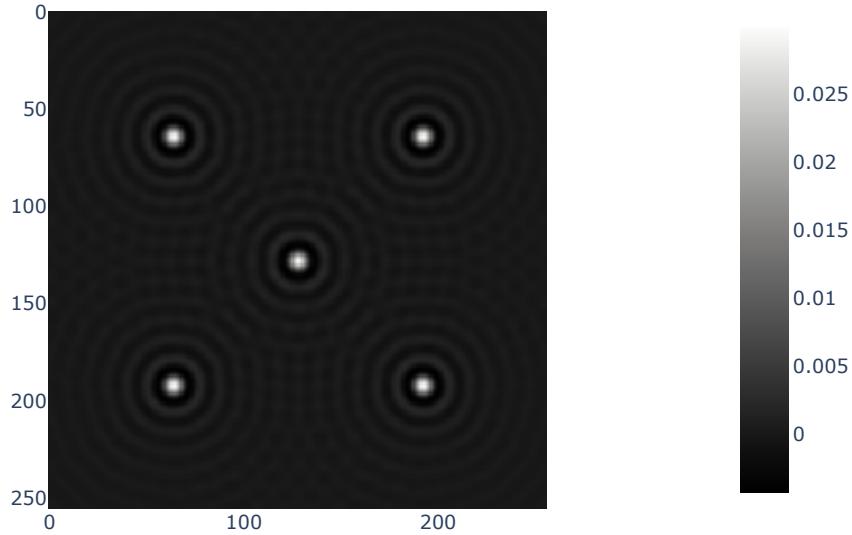


Prikazom izlazne slike vidimo da slika obuhvata dodatne celine koje su ostale od proširenja. Za dobijanje pravog rezultata filtriranja potrebno je samo iseći region slike koji odgovara originalnom pre proširenja (u slučaju proširenja sa donje i desne strane, to je gornje levi region dimenzije polazne slike).

```
[24]: img1 = img1[0:img.shape[0],0:img.shape[1]]

fig = px.imshow(img1, color_continuous_scale='gray')
```

```
fig.show()
```



U filtriranoj slici uočavamo da je energija sadržana na pojedinim tačkama u polaznoj slici raspodeljenja na neposrednu okolinu. Ovde treba zapaziti efekat filtriranja idealnim NF filtrom, tj. efekat koji on ima u prostornom domenu. Furijeov transformacioni par za idealni 1D NF filter je *sinc* funkcija, za 2D idealni NF biće *sinc* po dve dimenzije. Množenje u frekvencijskom je ovde konvolucija u prostornom, tj. postavljanje 2D *sinc* funkcije na lokaciju svake tačke u slici, gde vidimo jasan efekat curenja spekra u vidu formiranih prstenova u okolini originalnih tačaka (tzv. zvonjava).

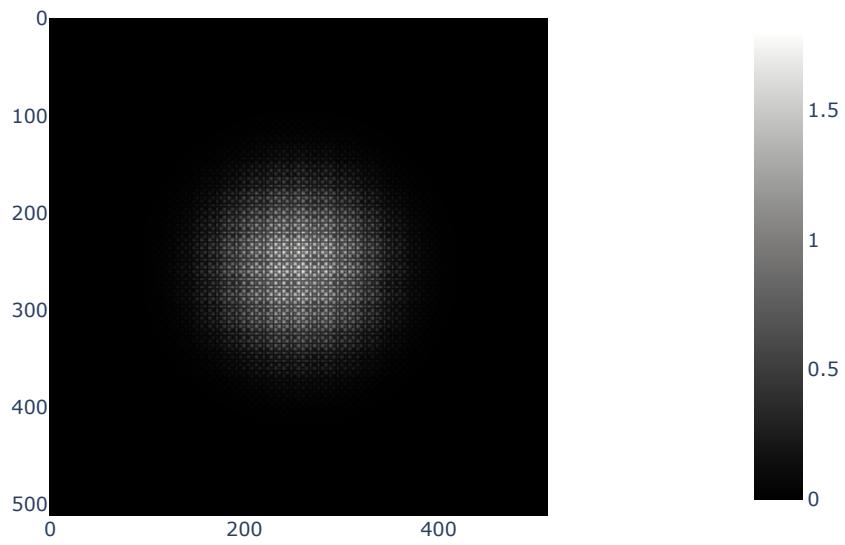
2.6.2 Gausov

Izvršiti filtriranje polazne slike sa Gausovim NF filtrom sa parametrom D0=100 i prikazati rezultat.

```
[25]: H2 = gauss_nf(IMG.shape,50)

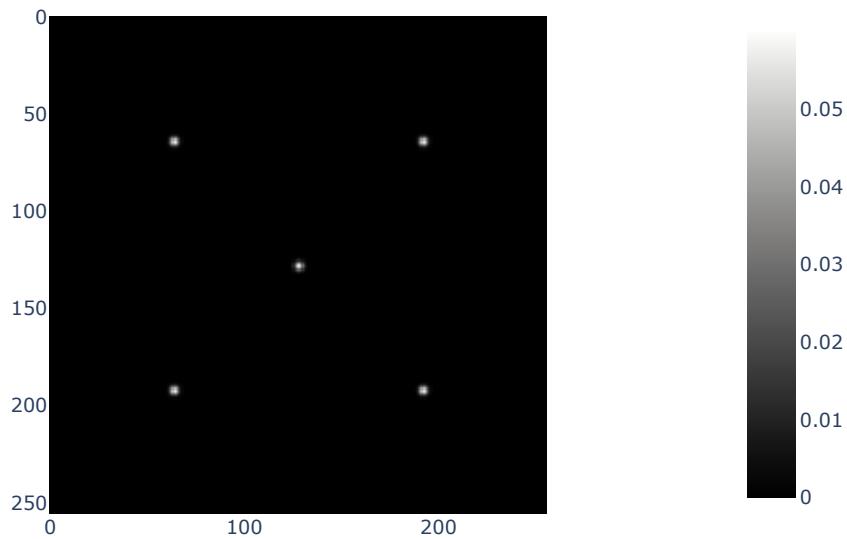
# filtriranje
IMG2 = IMG * H2

fig = px.imshow(np.log(np.abs(IMG2)+1), color_continuous_scale='gray')
fig.show()
```



```
[26]: img2 = np.fft.ifft2(np.fft.ifftshift(IMG2))
img2 = np.real(img2)
img2 = img2[0:img.shape[0],0:img.shape[1]]

fig = px.imshow(img2, color_continuous_scale='gray')
fig.show()
```



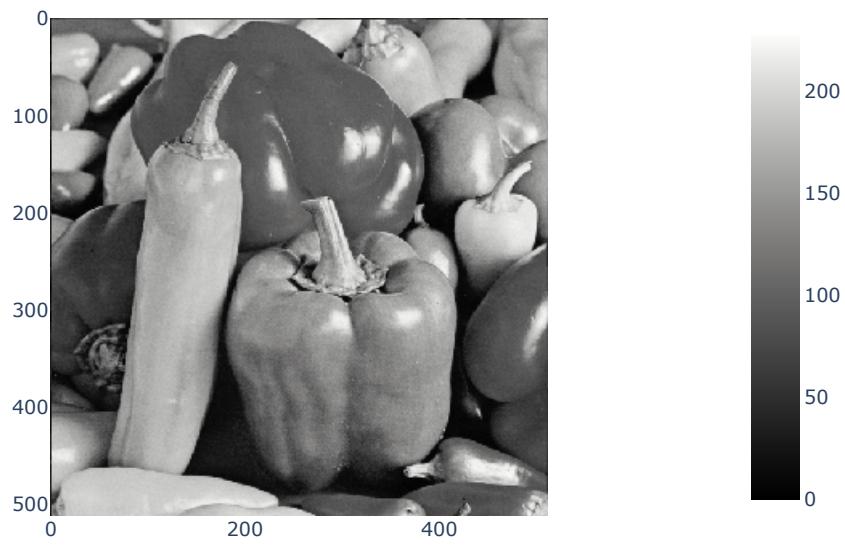
Furijeov transformacioni par Gausovske funkcije je takođe Gausovska funkcija, koja nema dodatne oscilacije za razliku od *sinc* funkcije, što se može primetiti i sa rezultujuće slike.

2.7 Primer 7.7

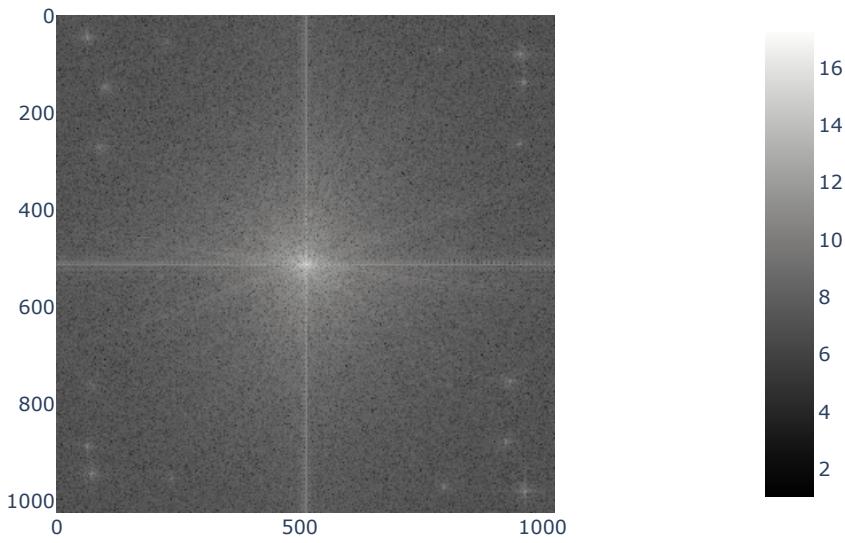
Učitati sliku `peppers.png` i filtrirati je u frekvencijskom domenu sa idealnim VF filtrom nastalim pomoću idealnog NF filtra sa parametrom $D_0=50$. Prikazati usputne slike.

```
[27]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('peppers.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[28]: IMG = np.fft.fftshift(np.fft.fft2(img, s=(2*img.shape[0],2*img.shape[0])))
fig = px.imshow(np.log(np.abs(IMG)+1), color_continuous_scale='gray')
fig.show()
```



Idealni VF filter ima formu:

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

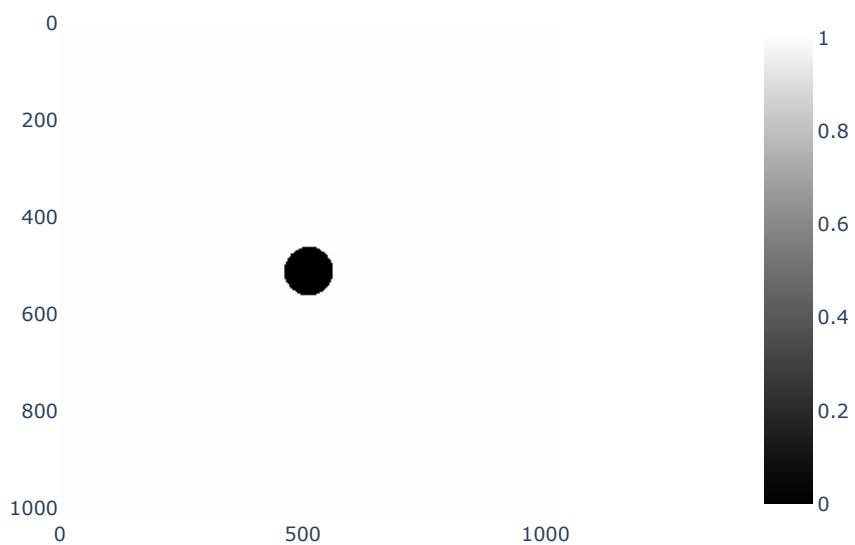
Ovakav filter se može dobiti direktno na osnovu idealnog NF kao:

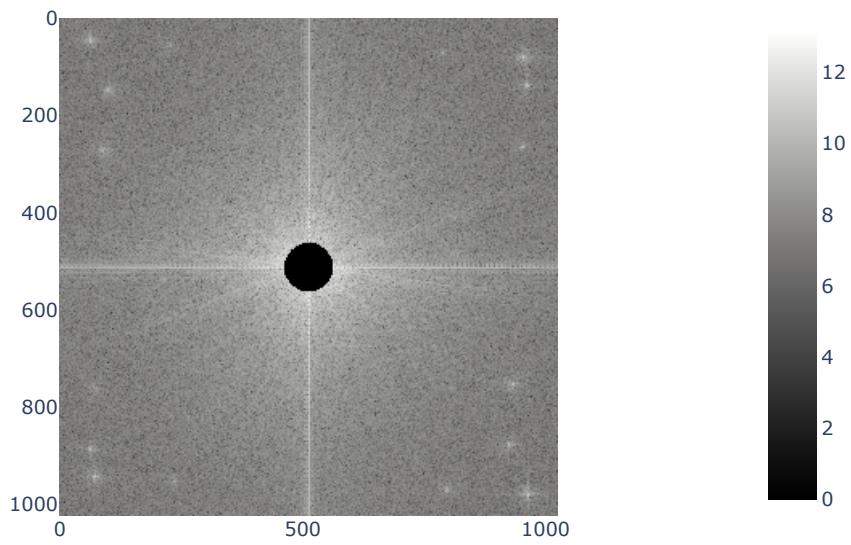
$$H_{VF} = 1 - H_{NF}$$

```
[29]: H1 = 1 - ideal_nf(IMG.shape,50)
fig = px.imshow(np.abs(H1), color_continuous_scale='gray')
fig.show()

# filtriranje
IMG1 = IMG * H1

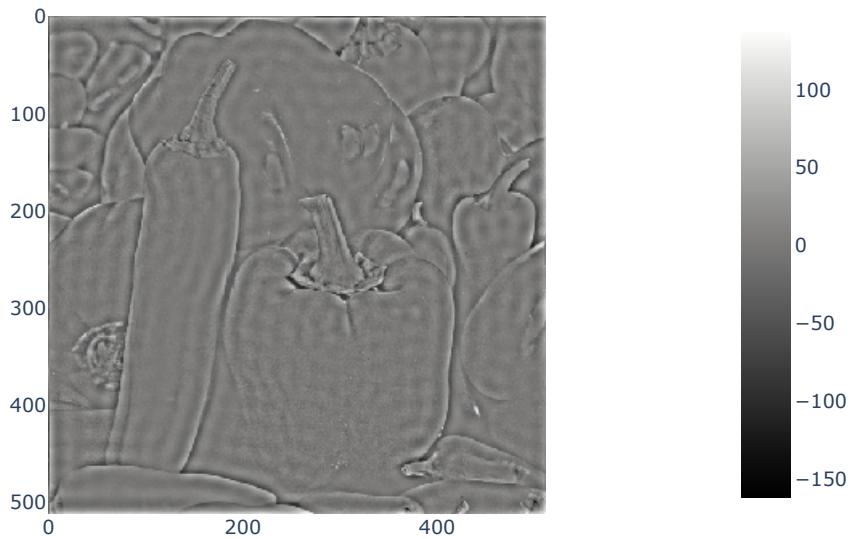
fig = px.imshow(np.log(np.abs(IMG1)+1), color_continuous_scale='gray')
fig.show()
```





```
[30]: img1 = np.real(np.fft.ifft2(np.fft.ifftshift(IMG1)))
img1 = img1[0:img.shape[0],0:img.shape[1]]

fig = px.imshow(img1, color_continuous_scale='gray')
fig.show()
```



2.8 Primer 7.8

Učitati sliku boat.png. Izoštiti sliku u frekvencijskom domenu koristeći Laplasijan kernel za izoštavanje koji posmatra 4 suseda:

$$L_{izostavljane} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

```
[31]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('boat.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```

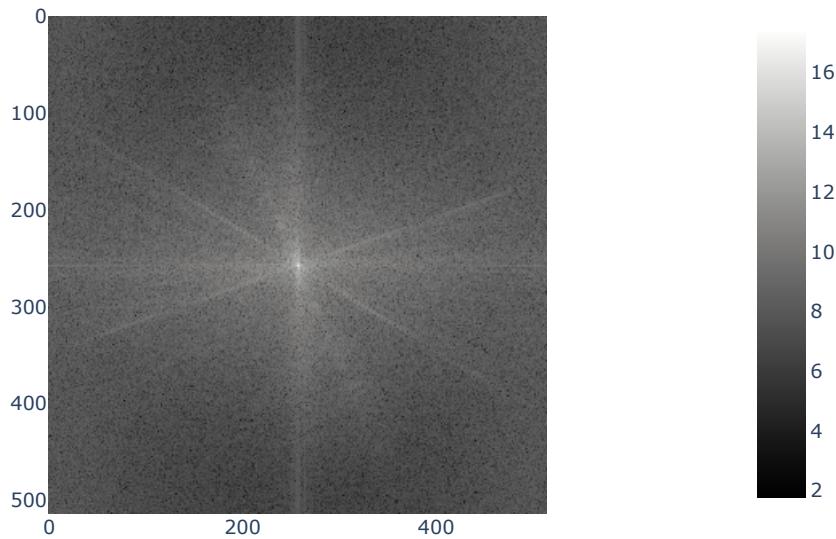


Za potrebe filtriranja u frekvencijskom domenu sa kernelom (definisanim u prostornom) nije potrebno fiksno proširivati sliku na dvostruku dimenziju. U ovom slučaju sliku je potrebno proširiti samo za onaj broj redova i kolona koji bi bili potrebni za regularno prostorno filtriranje. Ovde će biti prikazano sa **simetričnim proširenjem sa svih strana**.

Napomena: Prevođenje kernela u frekvencijski domen će biti rađeno na isti način, tako da ne-ma potrebe za izmeštanjem jednosmerne komponente u centar slike (jer će oboma biti podjenaka rapodela), ali zarad konzistentnosti sa prethodnim primerima ostavljeno je.

```
[32]: # potrebno prosirenje za filter dimenzije 3x3 je 1 po svakoj ivici
pad = 1;
img_p = np.pad(img, pad, 'reflect');

IMG = np.fft.fftshift(np.fft.fft2(img_p))
fig = px.imshow(np.log(np.abs(IMG)+1), color_continuous_scale='gray')
fig.show()
```



Prebacivanje kernela u frekvencijski domen se vrši takođe uz pomoć `fft2` funkcije (na dimenziju proširene polazne slike da bi bio zadovoljen uslov množenja elementa po elementu). Ovde bi samo trebali da pripazimo ako želimo da konstuišemo tzv. *zero-phase shift* filter (da ne utiče na fazni spektar polazne slike) potrebno je prošireni kernel pomeriti tako da se vrednost originalnog centralnog elementa nađe na koordinati $(0,0)$. U ovom primeru to je obezbeđeno funkcijom `numpy.roll`.

```
[33]: L_izostravanje = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])

L_p = np.pad(L_izostravanje,((0,IMG.shape[0]-3),(0,IMG.shape[1]-3)))
print(L_p)
L_p = np.roll(L_p,[-pad,-pad],axis=(0,1))
print(L_p)

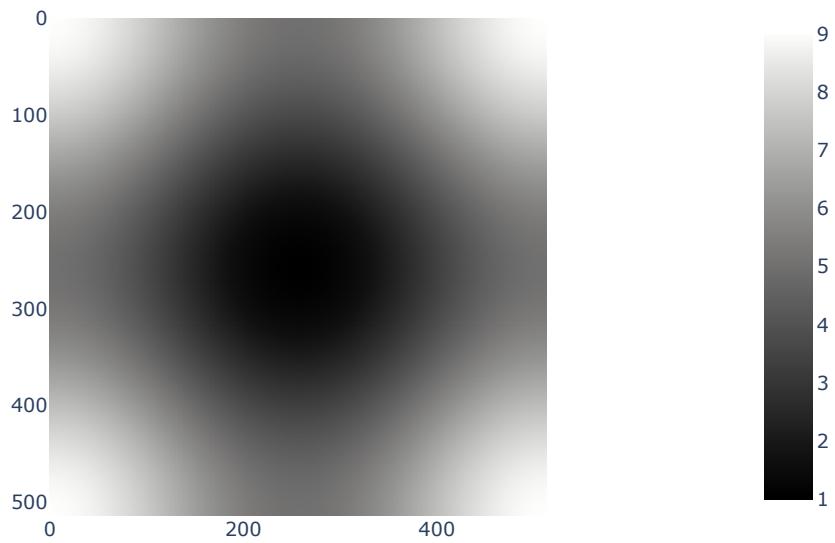
L_f = np.fft.fftshift(np.fft.fft2(L_p))
fig = px.imshow(np.abs(L_f), color_continuous_scale='gray')
fig.show()
fig = px.imshow(np.angle(L_f), color_continuous_scale='gray')
fig.show()
```

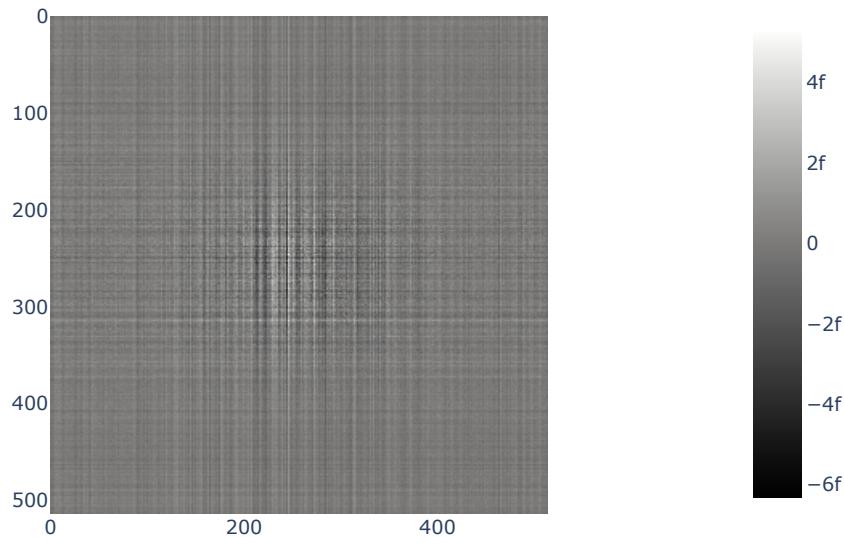
```
[[ 0 -1  0 ...  0  0  0]
 [-1  5 -1 ...  0  0  0]
 [ 0 -1  0 ...  0  0  0]
 ...
 [ 0  0  0 ...  0  0  0]]
```

```

[ 0  0  0 ...  0  0  0]
[ 0  0  0 ...  0  0  0]]
[[ 5 -1  0 ...  0  0 -1]
 [-1  0  0 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]
 ...
 [ 0  0  0 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]
 [-1  0  0 ...  0  0  0]]

```





Inspekcijom vrednosti faznog spektra vidimo da su sve vrednosti zanemarljivo male veličine (tehnički 0), što nam odgovara. (Zakomentarisati liniju koda za pomeranje kernela nakon proširenja i uporediti izgled faznog spektra.)

Nakon filtriranja potrebno je samo ukloniti one vrste i kolone koje su dodate prilikom proširenja (kao i u ranijim primerima). Pošto je proširenje bilo sa svih strana, jednostavno se iseca centralni region definisan parametrom pad upotrebljen za inicijalno proširenje.

```
[34]: IMG1 = IMG * L_f
img1 = np.real(np.fft.ifft2(np.fft.ifftshift(IMG1)))
img1 = img1[pad:-pad,pad:-pad]

fig = px.imshow(img1, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



Rezultujuća slika je jasno izoštrena (za prikaz ograničavamo opseg tonske skale od 0 do 255). Upoređivanjem ovog rezultata sa slikom nastalom korišćenjem konvolucije možemo uočiti da su identične.

3 Zadaci za samostalnu vežbu

Zadatak 7.1

Posmatrana DFT slika se može zapisati i kao:

$$F(u, v) = |F(u, v)|e^{j\phi(u, v)}$$

gde je $|F(u, v)|$ amplitudski spektar a $\phi(u, v)$ fazni spektar.

Učitati slike `lena.png` i `peppers.png` i odrediti njihove amplitudske i fazne spekture. Potom izvršiti konstruisanje novih slika međusobnim mešanjem komponenti:

- napraviti DFT sliku sačinjenu od amplitudskog spektra slike `lena.png` i faznog `peppers.png`
- napraviti DFT sliku sačinjenu od amplitudskog spektra slike `peppers.png` i faznog `lena.png`

Odrediti inverzne DFT i prikazati rezultate. Komentarisati šta se desilo.

Napomena: koristiti `1j` kao imaginarnu jedinicu i funkciju `numpy.exp` za računanje prirodne eksponencijalne funkcije.

[35] : #TODO

Zadatak 7.2

Napisati funkciju `butterworth_nf` koja implementira Batervortov NF filter reda n . Ulazni parametri su tuple vrednost velicine filtra `H_size`, poluprecnik `D0` i red `n`. Funkcionalnost filtra isprobati nad slikom `lena.png`. Filter isprobati sa nekoliko različitih parametara `D0` i `n`. Filter je dat formulom:

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

Prikazati amplitudsku karakteristiku filtra, kao i dodatan 3D prikaz i poprečni presek.

[36] : #TODO

Zadatak 7.3

Napisati funkciju `spectralUnsharpMasking` koja koristi Gausov VF filter za anšarp masking (USM) u frekvencijskom domenu. Gausov VF filter je definisan izrazom:

$$H_{VF}(u, v) = 1 - e^{-\frac{D^2(u, v)}{2D_0^2}}$$

USM u frekvencijskom domenu je definisan preko sledećeg izraza:

$$H_{UM}(u, v) = 1 + C H_{VF}(u, v)$$

gde C predstavlja nenegativan koeficijent. Za vrednosti C veće od 0 se dobija izoštravanje slike. Funkcionalnost filtra isprobati nad slikom `cat.png`. Filter isprobati sa nekoliko različitih parametara. Prikazati izlazne slike u opsegu tonske skale od 0 do 255.

[37] : #TODO

Zadatak 7.4

Odrediti sliku ivica slike `deda_mraz.png` binarizacijom gradijentne slike određenu uz pomoć para Sobel kornela. Filtriranja izvršiti u frekvencijskom domenu. Koristiti prag koji daje vizuelno jasan prikaz jakih ivica u slici. Sobel gradijentni par operatora je definisan sledećim maskama:

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{i} \quad S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

[38] : `#TODO`

4 Dokumentacija novih celina

- [numpy.fft.fft2](#)
- [numpy.fft.fftshift](#)
- [numpy.fft.ifftshift](#)
- [numpy.fft.ifft2](#)
- [plotly.graph_objects.Surface](#)
- [numpy.roll](#)
- [numpy.exp](#)

Vezba 8 - Restauracija slike 1

1 Pregled

U vežbi se analiziraju različita oštećenja slike usled prisustva šuma i načini za njihovo uklanjanje. U okviru ove vežbe razmatra se kompenzacija uticaja dva modela šuma: aditivnog belog Gausovog šuma i impulsnog šuma. Kao objektivna mera kvaliteta je korišćen vršni odnos signal-šum.

2 Restauracija slike

Cilj restauracije je da poboljša kvalitet slike. Za razliku od prethodnih vežbi gde smo imali ekvalizaciju histograma ili izoštravanje kao način subjektivnog poboljšanja slike, restauracija predstavlja modifikaciju slike koristeći apriorne informacije o prisutnim degradacijama i nastoji da ih ukloni u cilju dobijanja originalne slike. U ovoj vežbi analiziramo jednu od čestih situacija u kojoj želimo da vršimo restauraciju a to je u kontekstu šuma prisutnog u slici. Analiziraćemo 2 najosnovnija oblika šuma: * aditivni beli Gausov šum * impulsni so i biber šum

i načine kako možemo da ih simuliramo u okviru slike i kako da ih potisnemo.

3 Aditivni beli Gausov šum

Aditivni beli Gausov šum (engl. *Additive white Gaussian noise (AWGN)*) predstavlja model koji opisuje čestu vrstu šuma koji se može javiti u okviru slike (npr. termički). Opisuje se Gausovom (Normalnom) raspodelom datom formulom:

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z-\bar{z})^2}{\sigma^2}}$$

gde je \bar{z} srednja vrednost a σ standardna devijacija. U većini slučajeva smatraćemo da je srednja vrednost 0, dok standardna devijacija će definisati "jačinu" šuma - što je veća, to će šum biti izraženiji.

U okviru slike ovaj šum se ispoljava kao slučajne vrednosti dobijene iz ove raspodele dodate na vrednosti intenziteta originalne slike (odatle aditivni u nazivu).

3.1 Primer 8.1

Napraviti demo sliku dimenzije 256x256 sa dva kvadrata u sredini (jedan u drugom) koji su sa nijansama 127 i 191. Postaviti pozadinu na 63. Prikazati njen histogram.

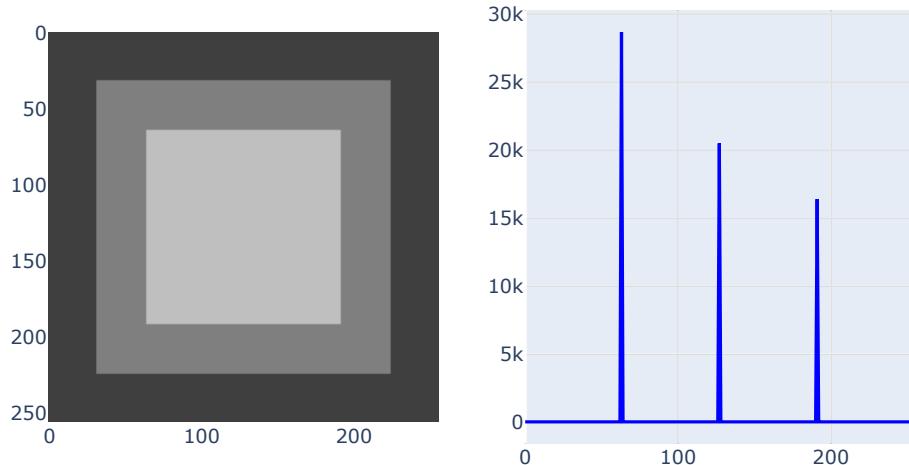
Na ovu sliku dodati AWGN šum varijanse 100 i srednje vrednosti 0. Prikazati rezultat i histogram nove slike. Odrediti vršni odnos signal-šum.

```
[2]: import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np

img = np.full((256,256), fill_value=63) # moze i kao np.zeros() + 63 / np.ones() * 63
img[32:32+192,32:32+192] = 127
img[64:64+128,64:64+128] = 191

hist, bin_edges = np.histogram(img, bins = np.arange(257))

fig = make_subplots(rows=1, cols=2)
fig.add_trace(go.Image(z=np.stack((img, img, img)), axis=2)), row=1, col=1)
fig.add_trace(go.Scatter(x=np.arange(256), y=hist, mode='lines', name='hist', line=dict(color='blue')), row=1, col=2)
fig.show()
```



3.1.1 Dodavanje Gausovog šuma

Za generisanje slučajnih uzoraka koristimo `numpy.random.default_rng` generator koji može da vraća slučajne odbirke iz mnoštva raspodela. Generator može dodatno da primi parametar `seed`

koji omogućuje ponovljivost generisanih vrednosti.

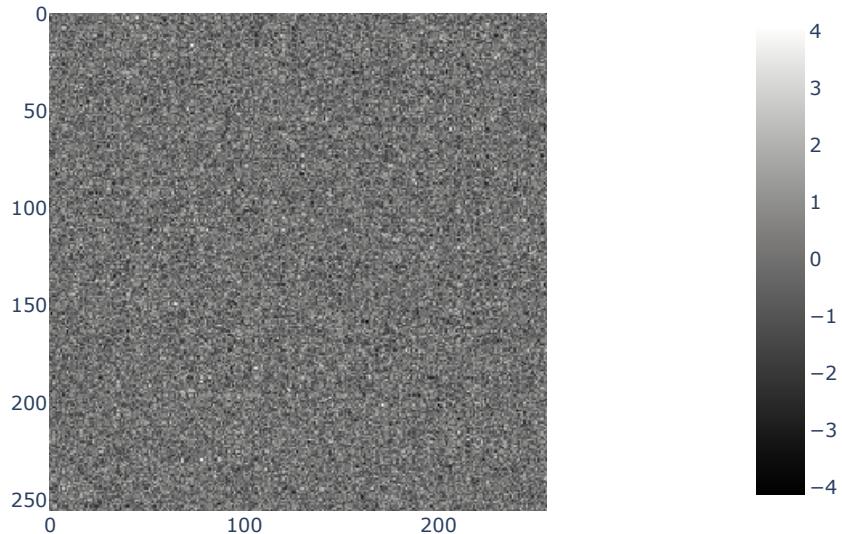
U okviru ovog primera bi želeli da generišemo odbirke iz normalne raspodele, što radimo pomoću `standard_normal` metode. Dobijeni uzorci su iz raspodele koja ima srednju vrednost 0 i varijansu 1. Da bi podešili na željenu varijansu, dobijene uzorke je potrebno pomnožiti sa korenom željene varijanse (što je standardna devijacija), a za postavljanje na željenu srednju vrednost potrebno je sve sabrati sa tom vrednosti.

Napomena: vrednosti piksela mogu potencijalno biti postavljeni na vrednosti van opsega usled uticaja šuma. U ovom slučaju bi trebalo samo postaviti te vrednosti na odgovarajuće granice sa donje i gornje strane.

```
[3]: import plotly.express as px

rng = np.random.default_rng()
noise_im = rng.standard_normal(size=img.shape)

fig = px.imshow(noise_im, color_continuous_scale='gray')
fig.show()
```



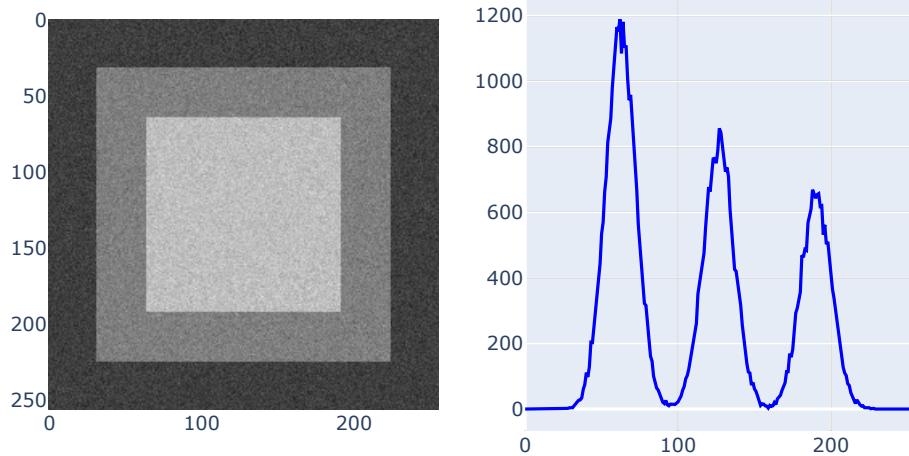
```
[4]: img_noisy = img + noise_im*np.sqrt(100) + 0
img_noisy[img_noisy > 255] = 255
img_noisy[img_noisy < 0] = 0

hist_n, bin_edges = np.histogram(img_noisy, bins = np.arange(257))
```

```

fig = make_subplots(rows=1, cols=2)
fig.add_trace(go.Image(z=np.stack((img_noisy, img_noisy, img_noisy), axis=2)), row=1, col=1)
fig.add_trace(go.Scatter(x=np.arange(256), y=hist_n, mode='lines', name='hist_noisy', line=dict(color='blue')), row=1, col=2)
fig.show()

```



Sa histograma jasno se uočava efekat Gausovog šuma u vidu normalnih raspodela pozicioniranih na mestima intenziteta histograma originalne slike.

3.1.2 PSNR

Vršni odnos signal-šum (engl. *Peak signal-to-noise ratio (PSNR)*) predstavlja objektivnu meru uticaja šuma u okviru slike. Računa se prema formuli:

$$PSNR = 10 \log_{10} \frac{MAX_f^2}{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (f(x, y) - \hat{f}(x, y))^2}$$

gde član MAX_f predstavlja vršnu (maksimalno moguću) vrednost za dati tip slike (za celobrojni tip slike računa se kao $L - 1$, gde je L ukupan broj nivoa slike, a za float sliku to je vrednost 1). Donji član formule predstavlja srednje kvadratnu grešku (engl. *Mean squared error (MSE)*) između originalne f i modifikovane slike \hat{f} .

```
[5]: psnr = 10*np.log10(255**2/((img - img_noisy)**2).mean())
print(psnr)
```

```
28.125151016571156
```

Posmatrajući razliku ispod razlomačke crte, možemo uočiti da što je modifikovana slika bliža originalnim vrednostima piksela originalne slike donji član formule će težiti manjim vrednostima, što znači da će ceo izraz težiti većim - ovo ukazuje da je tada originalan signal dominantniji u odnosu na potencijalne modifikacije (u ovom slučaju zbog šuma).

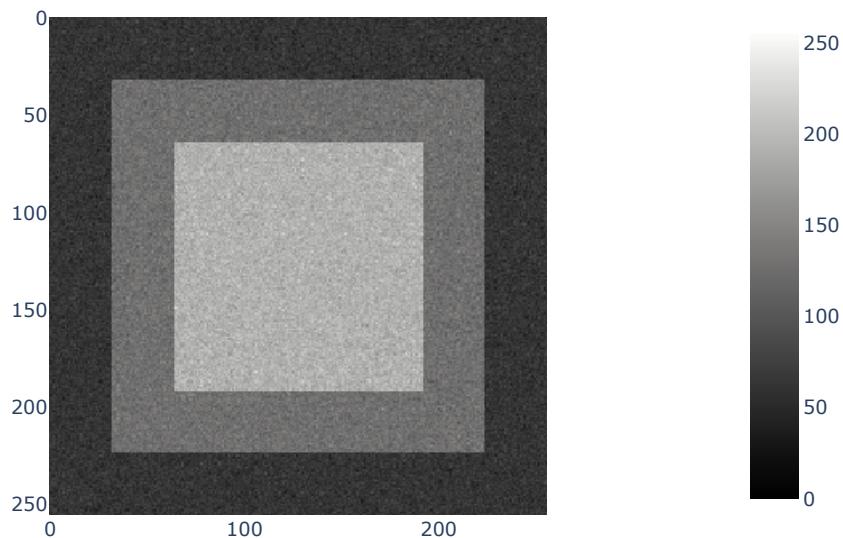
3.2 Primer 8.2

Koristeći zašumljenu sliku iz prethodnog primera, ukloniti šum upotrebom aritmetičkog usrednjivača veličina: * 3x3 * 5x5 * 11x11

Koristeći PSNR odrediti koji filter je dao objektivno najbolje otklanjanje.

```
[6]: import plotly.express as px

fig = px.imshow(img_noisy, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



```
[7]: from scipy import ndimage
import plotly.graph_objects as go
```

```

from plotly.subplots import make_subplots

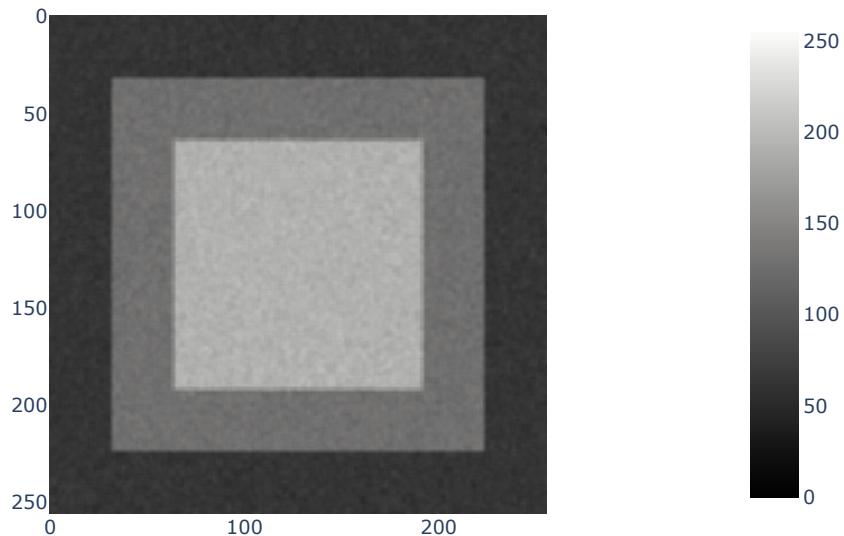
# 3x3 aritmeticki usrednjivac
kernel = np.ones((3,3))/9

img_f = ndimage.convolve(img_noisy.astype('float'), kernel, mode = 'mirror')

fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

psnr = 10*np.log10(255**2/((img - img_f)**2).mean())
print("PSNR za box filter dimenzije {} je {}".format(kernel.shape, psnr))

```



PSNR za box filter dimenzije (3, 3) je 33.50111980847177

```

[8]: # 5x5 aritmeticki usrednjivac
kernel = np.ones((5,5))/25

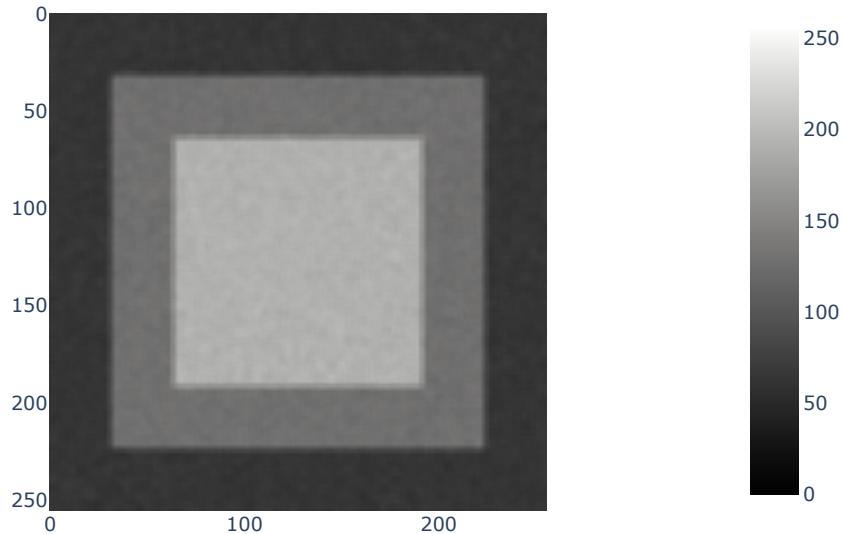
img_f = ndimage.convolve(img_noisy.astype('float'), kernel, mode = 'mirror')

fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

psnr = 10*np.log10(255**2/((img - img_f)**2).mean())

```

```
print("PSNR za box filter dimenzije {} je {}".format(kernel.shape, psnr))
```



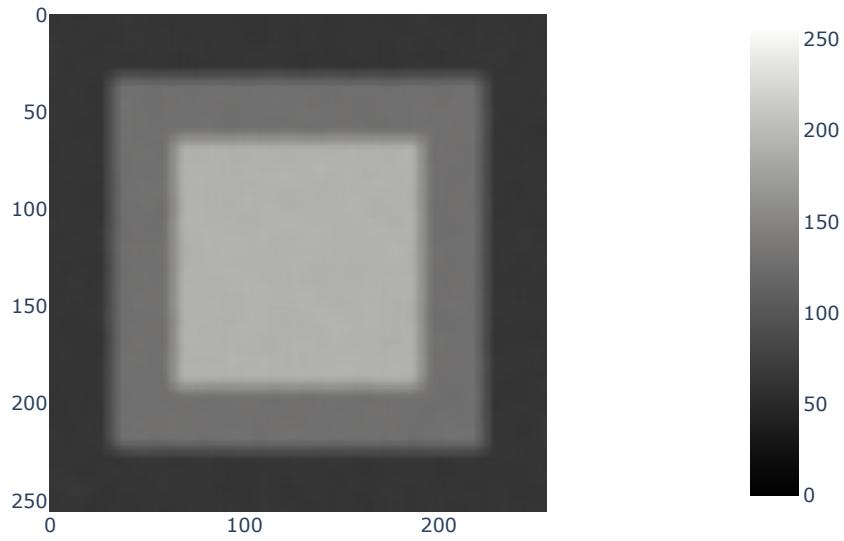
PSNR za box filter dimenzije (5, 5) je 32.52721909134016

```
[9]: # 11x11 aritmeticki usrednjivac
kernel = np.ones((11,11))/121

img_f = ndimage.convolve(img_noisy.astype('float'), kernel, mode = 'mirror')

fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

psnr = 10*np.log10(255**2/((img - img_f)**2).mean())
print("PSNR za box filter dimenzije {} je {}".format(kernel.shape, psnr))
```



PSNR za box filter dimenzije (11, 11) je 29.474137907767272

Na osnovu vrednosti PSNR zaključuje se da filter veličine 3x3 bolje restaurira sliku. Aritmetički usrednjivač ublažava lokalne varijacije intenziteta piksela u slici te je posledično umanjen uticaj šuma, ali je i dobijena slika ublažena. Moguće je uočiti da povećanje maske filtra rezultuje drastičnijim ublažavanjem slike koje nije nužno povećanjem PSNR.

3.3 Primer 8.3

Koristeći zašumljenu sliku iz prethodnog primera, ukloniti šum upotrebnom geometrijskog usrednjivača veličina: * 3x3 * 5x5

Koristeći PSNR odrediti koji filter je dao objektivno najbolje otklanjanje.

Za potrebe rešavanja zadatka napisati funkciju `geometricMeanFilt` koja implementira filter geometrijski usrednjivač dat formulom:

$$\hat{f}(x, y) = \left[\prod_{(s,t) \in S_{x,y}} g(s, t) \right]^{\frac{1}{mn}}$$

gde (s, t) predstavljaju prostorne koordinate regiona $S_{x,y}$ posmatran oko centra pozicije (x, y) , g je ulazna slika a m, n predstavljaju dimenzije regiona koji se posmatra. Podrazumevati simetrično proširenje.

Napomena: za proizvod niza elementa koristimo `numpy.prod`.

```
[10]: import numpy as np
import math

def geometricMeanFilt(g, S_shape):
    f = np.zeros(g.shape)

    pad_rows = math.floor(S_shape[0]/2)
    pad_cols = math.floor(S_shape[1]/2)

    img_p = np.pad(g, ((pad_rows,), (pad_cols,)), mode='symmetric')

    for row in range(f.shape[0]):
        for col in range(f.shape[1]):
            region = img_p[row:row+S_shape[0], col:col+S_shape[1]]

            res = region.prod()**(1/np.prod(S_shape))

            f[row, col] = res

    return f
```

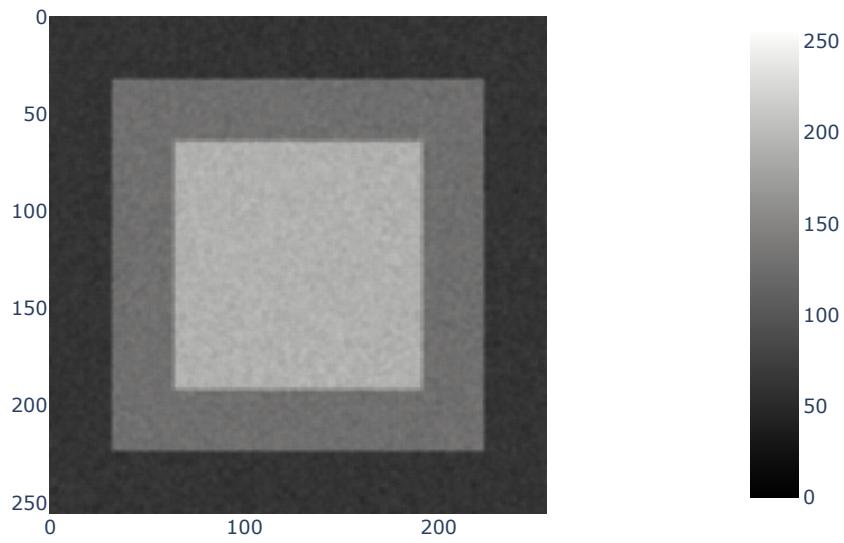
Napomena: pošto se u okviru funkcije poziva množenje na nivou numpy niza, potrebno je obratiti pažnju da je tip niza u mogućnosti da reprezentuje vrednost rezultata. U ovom slučaju slika je već u `float` tipu, tako da nije potrebno.

```
[11]: # 3x3 geometrijski usrednjivac
S_shape = (3,3)

img_f = geometricMeanFilt(img_noisy, S_shape)

fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

psnr = 10*np.log10(255**2/((img - img_f)**2).mean())
print("PSNR za geometrijski usrednjivac dimenzije {} je {}".format(S_shape, psnr))
```



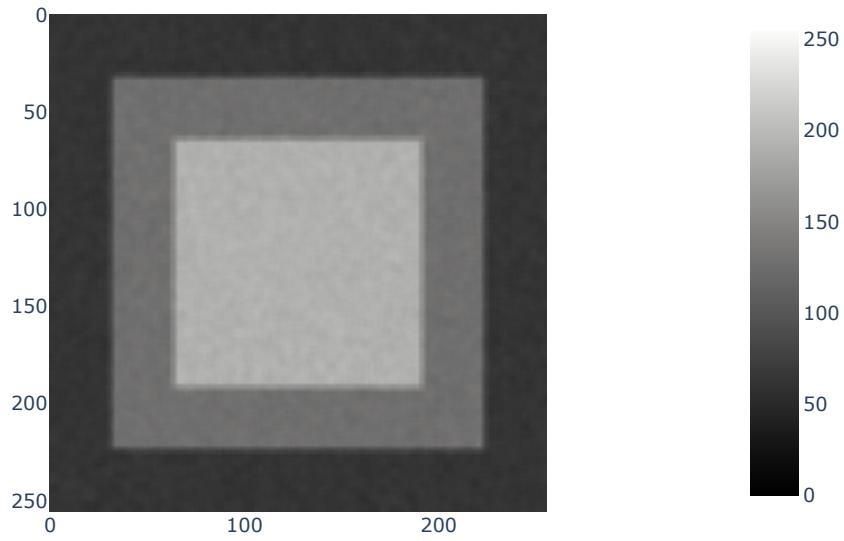
PSNR za geometrijski usrednjivac dimenziije (3, 3) je 33.25299408005567

```
[12]: # 5x5 geometrijski usrednjivac
S_shape = (5,5)

img_f = geometricMeanFilt(img_noisy, S_shape)

fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

psnr = 10*np.log10(255**2/((img - img_f)**2).mean())
print("PSNR za geometrijski usrednjivac dimenziije {} je {}".format(S_shape, psnr))
```



PSNR za geometrijski usrednjivac dimenzije (5, 5) je 32.25186224473841

Uočavamo da filter ima uporedive rezultate sa aritmetičkim usrednjivačem.

Napomena: geometrijski usrednjivač ima inherentan problem ukoliko se zbog uticaja šuma dobije vrednost 0. U tom slučaju svi regioni koji zahvate bar jedan takav piksel daju rezultat 0.

4 Impulsni so i biber šum

Impulsni so i biber šum predstavlja model šuma koji oštećuje izvestan procenat piksela u okviru slike (tako da neki ostaju neoštećeni). Oštećeni pikseli dobijaju vrednosti ili minimalne za dati tip slike (tzv. biber šum, engl. *pepper*) ili maksimalne (tzv. so šum, engl. *salt*). Za definisanje ovog šuma, potrebno je naznačiti verovatnoću pod kojim je piksel pogoden ovom vrstom šuma:

$$P(g(x,y) = n(x,y)) = p_0$$

$$P(g(x,y) = f(x,y)) = 1 - p_0$$

Verovatnoća p_0 suštinski predstavlja gustinu piksela koji su pod oštećenjem.

4.1 Primer 8.4

Napraviti demo sliku dimenzije 256x256 sa dva kvadrata u sredini (jedan u drugom) koji su sa nijansama 127 i 191. Postaviti pozadinu na 63. Prikazati njen histogram.

Na ovu sliku dodati impulsni: * so * biber * so i biber

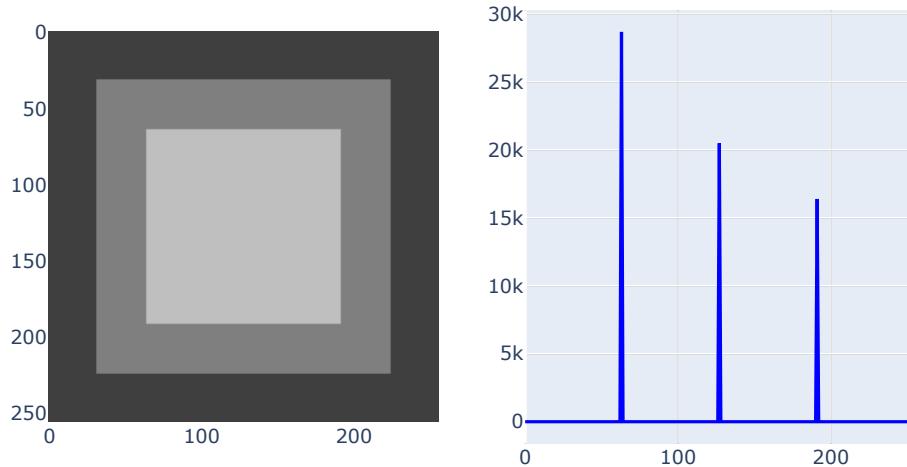
šum gustine 20%. Prikazati zašumljene slike i njihove histograme.

```
[13]: import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np

img = np.full((256,256), fill_value=63) # moze i kao np.zeros() + 63 / np.ones() * 63
img[32:32+192,32:32+192] = 127
img[64:64+128,64:64+128] = 191

hist, bin_edges = np.histogram(img, bins = np.arange(257))

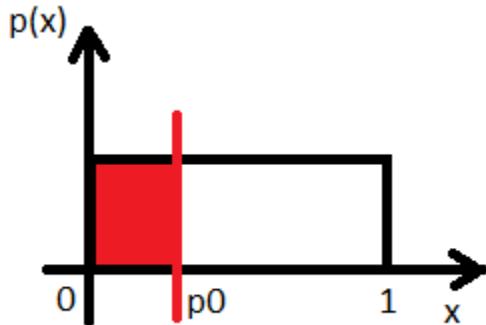
fig = make_subplots(rows=1, cols=2)
fig.add_trace(go.Image(z=np.stack((img, img, img)), axis=2)), row=1, col=1)
fig.add_trace(go.Scatter(x=np.arange(256), y=hist, mode='lines', name='hist',
                         line=dict(color='blue')), row=1, col=2)
fig.show()
```



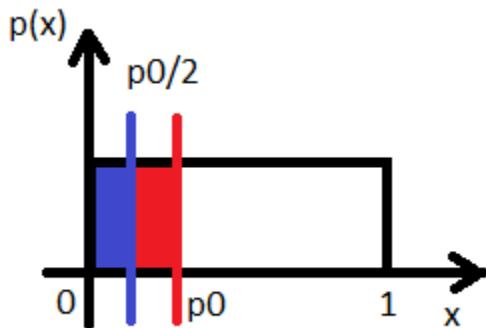
4.1.1 Dodavanje impulsnog šuma

U okviru impulsnog šuma pikseli imaju podjenaku verovatnoću p_0 da budu zahvaćeni ovom vrstom šuma. U tom kontekstu, za svaki piksel potrebno je generisati slučajni uzorak iz uniformne $[0,1]$ raspodele i odabrati jedan interval širine p_0 (najlakše posmatrati opseg $[0, p_0)$). Ukoliko ge-

nerisan slučajni uzorak upadne u ovaj opseg piksel treba da postane oštećen i vrednost mu se postavlja na minimalnu ili maksimalnu za dati tip slike u slučaju biber ili so šuma. Na slici ispod je prikazan princip postavljanja ove granice koja određuje interval.



Ukoliko je potrebno generisati so i biber kombinovani šum, u tom slučaju za datu gustinu p_0 potrebno je formirati 2 opsega koji definiše intervale kada je piksel zahvaćen jendom, odnosno drugom vrstom šuma. Na slici ispod je primer ukoliko smatramo da su so i biber podjenako zastupljeni.



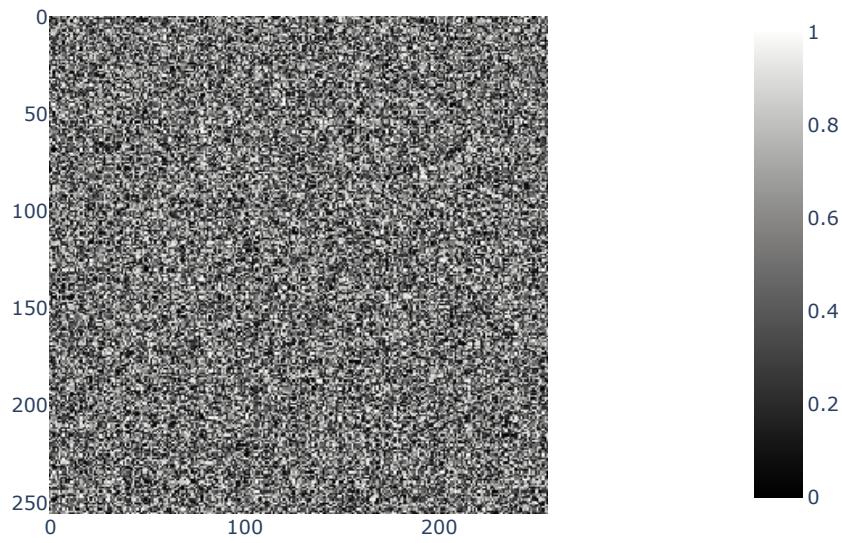
U okviru ovog primera bi želeli da generišemo odbirke iz uniformne raspodele, što radimo pomoću [uniform](#) metode datog generatora.

So

```
[14]: import plotly.express as px

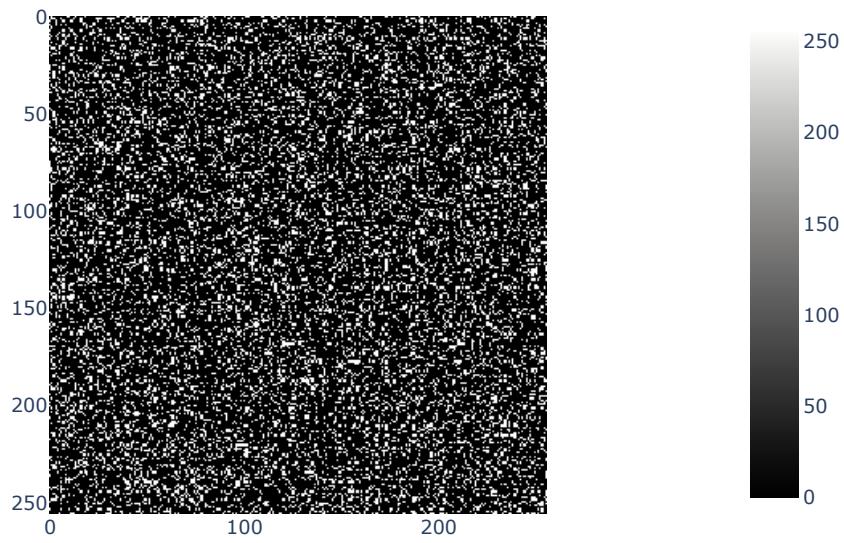
rng = np.random.default_rng()
p = rng.uniform(size=img.shape)

fig = px.imshow(p, color_continuous_scale='gray')
fig.show()
```



Maska oštećenih piksela definisano verovatnoćom p_0 se može prikazati sa

```
[15]: p0 = 0.2  
  
fig = px.imshow(p<p0, color_continuous_scale='gray')  
fig.show()
```



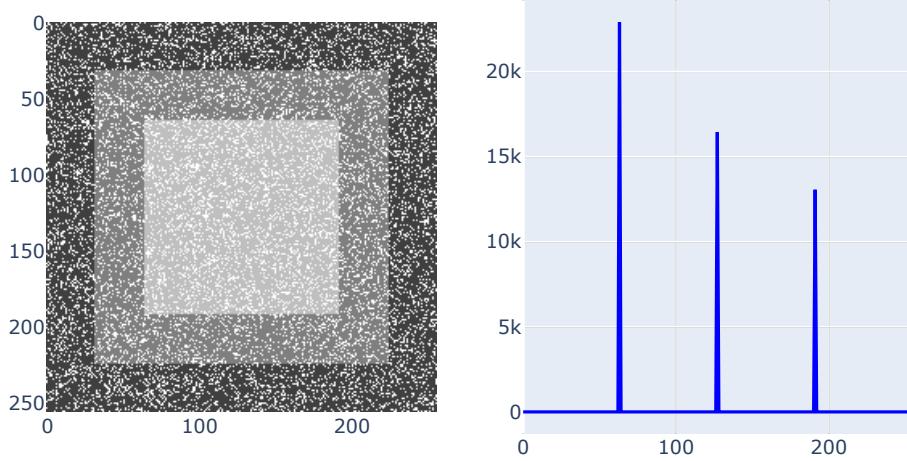
Dodavanje šuma se jednostavno vrši indeksiranjem lokacija definisanih maskom i postavljanjem na odgovarajuće vrednosti

```
[16]: img_so = img.copy()
img_so[p<p0] = 255

hist, bin_edges = np.histogram(img_so, bins = np.arange(257))

fig = make_subplots(rows=1, cols=2)
fig.add_trace(go.Image(z=np.stack((img_so, img_so, img_so), axis=2)), row=1, col=1)
fig.add_trace(go.Scatter(x=np.arange(256), y=hist, mode='lines', name='hist', line=dict(color='blue')), row=1, col=2)
fig.show()

psnr = 10*np.log10(255**2/((img - img_so)**2).mean())
print("PSNR za zasumljenu sliku je {}".format(psnr))
```



PSNR za zasumljenu sliku je 11.639463854620029

Podsetnik: Da nismo kopirali sliku, sve izmene nad `img_so` slikom bi se prevele i na originalnu `img` sliku.

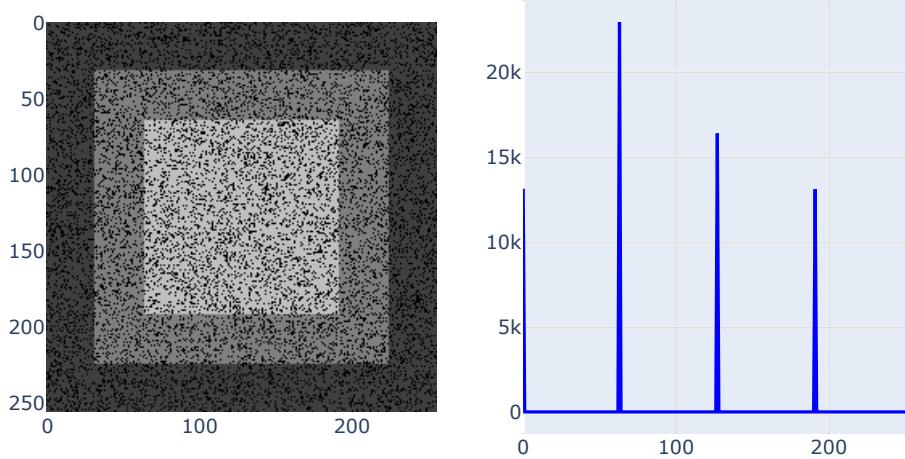
Biber

```
[17]: img_biber = img.copy()
img_biber[rng.uniform(size=img.shape)<p0] = 0

hist, bin_edges = np.histogram(img_biber, bins = np.arange(257))

fig = make_subplots(rows=1, cols=2)
fig.add_trace(go.Image(z=np.stack((img_biber, img_biber, img_biber), axis=2)), row=1, col=1)
fig.add_trace(go.Scatter(x=np.arange(256), y=hist, mode='lines', name='hist', line=dict(color='blue')), row=1, col=2)
fig.show()

psnr = 10*np.log10(255**2/((img - img_biber)**2).mean())
print("PSNR za zasumljenu sliku je {}".format(psnr))
```



PSNR za zasumljenu sliku je 13.098232515941667

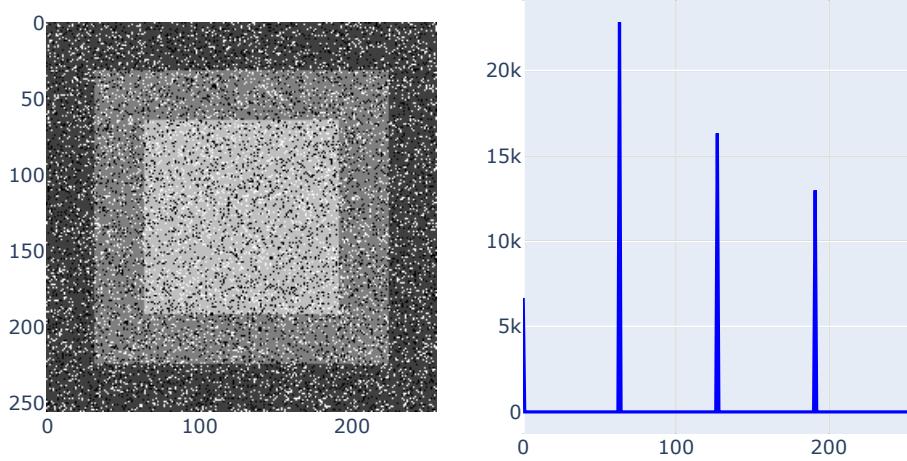
So i biber

```
[18]: img_sb = img.copy()
p = rng.uniform(size=img.shape)
img_sb[p<p0/2] = 0
img_sb[(p0/2<=p) & (p<p0)] = 255

hist, bin_edges = np.histogram(img_sb, bins = np.arange(257))

fig = make_subplots(rows=1, cols=2)
fig.add_trace(go.Image(z=np.stack((img_sb, img_sb, img_sb), axis=2)), row=1, col=1)
fig.add_trace(go.Scatter(x=np.arange(256), y=hist, mode='lines', name='hist', line=dict(color='blue')), row=1, col=2)
fig.show()

psnr = 10*np.log10(255**2/((img - img_sb)**2).mean())
print("PSNR za zasumljenu sliku je {}".format(psnr))
```



PSNR za zasumljenu sliku je 12.208031350600637

4.2 Primer 8.5

Sliku img_so iz **primera 8.4** filtrirati harmonijskim usrednjivačem definisanim izrazom:

$$\hat{f}(x, y) = \frac{mn}{\sum_{(s,t) \in S_{xy}} \frac{1}{g(s,t)}}$$

gde (s, t) predstavljaju prostorne koordinate regiona $S_{x,y}$ posmatran oko centra pozicije (x, y) , g je ulazna slika a m, n predstavljaju dimenzije regiona koji se posmatra.

Funkciju za filtriranje nazvati harmonicMeanFilt i podrazumevati simetrično proširenje. Sliku filtrirati sa regionom 5x5. Odrediti PSNR.

```
[19]: import numpy as np
import math

def harmonicMeanFilt(g, S_shape):
    f = np.zeros(g.shape)

    pad_rows = math.floor(S_shape[0]/2)
    pad_cols = math.floor(S_shape[1]/2)
```

```

img_p = np.pad(g, ((pad_rows,), (pad_cols,)), mode='symmetric')

for row in range(f.shape[0]):
    for col in range(f.shape[1]):
        region = img_p[row:row+S_shape[0], col:col+S_shape[1]]

        res = np.prod(S_shape)/(1/region).sum()

        f[row, col] = res

return f

```

```

[20]: import plotly.express as px

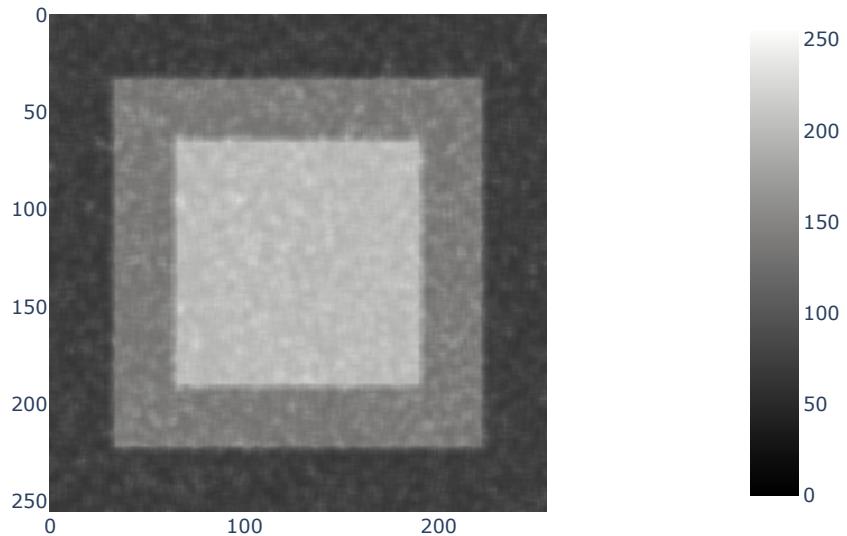
S_shape = (5,5)

img_f = harmonicMeanFilt(img_so, S_shape)

fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

psnr = 10*np.log10(255**2/((img - img_f)**2).mean())
print("PSNR za harmonijski usrednjivac dimenzije {} je {}".format(S_shape, psnr))

```



PSNR za harmonijski usrednjivac dimenzije (5, 5) je 24.98760740406484

Harmonijski usrednjivač je pogodan sa impulsni šum so tipa, međutim može imati loše posledice nad biber ili kombinovanim so i biber šumom.

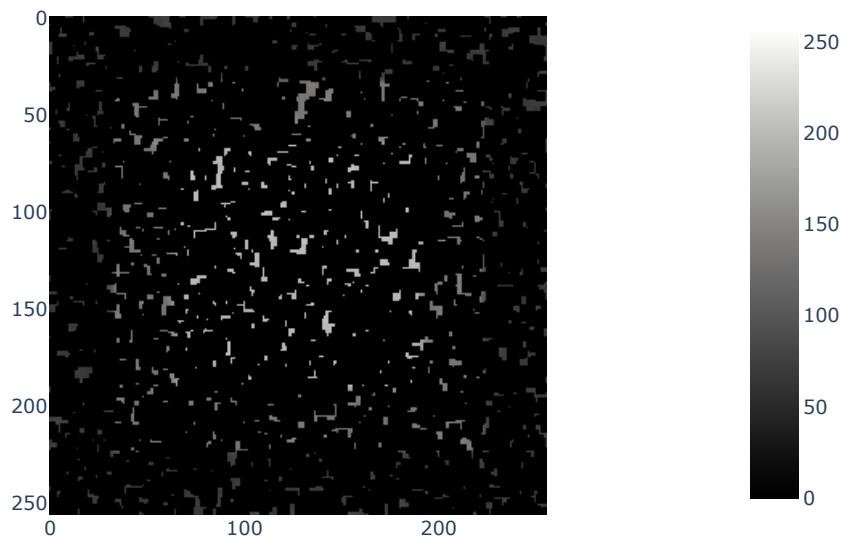
```
[21]: img_f = harmonicMeanFilt(img_sb, S_shape)

fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```

C:\Users\Ivan\AppData\Local\Temp\ipykernel_18244/1454870488.py:16:

RuntimeWarning:

divide by zero encountered in true_divide



4.3 Primer 8.6

Sliku img_sb iz **primera 8.4** filtrirati **median** filtrom statistike poretka dimenzije 3x3.

Filtri statistike poretka (engl. *order-statistics filters*) predstavljaju grupu nelinearnih filtara koji sortiraju elemente iz posmatranog regiona datim nekom maskom u rastućem poretku i vraćaju vrednost određenu zahtevanim indeksom u okviru ovog poretkta.

U zavisnosti od vrednosti indeksa ovim načinom se mogu definisati 3 osnovne vrste:

- Min filter $\hat{f}(x, y) = \min_{(s,t) \in S_{x,y}} g(s, t)$ - indeks je 0, tj. najmanja vrednost u poretku.
- Max filter $\hat{f}(x, y) = \max_{(s,t) \in S_{x,y}} g(s, t)$ - indeks je indeks poslednjeg elementa, tj. najveća vrednost u poretku.
- Median filter $\hat{f}(x, y) = \text{median}_{(s,t) \in S_{x,y}} g(s, t)$ - indeks je indeks centralnog elementa, tj. vrednost koja je veća od 50% preostalih piksela i 50% manja od vrednosti preostalih piksela.

Opšta funkcija za filtre poretna je implementirana u `scipy.signal.order_filter`. Ulagni parametri su N-dimenzioni niz koji se filtrira, domain parametar koji označava masku po kojoj se posmatraju elementi u regionu (sa vrednostima 0 i 1) i rank koji obeležava indeks iz poretna od interesa.

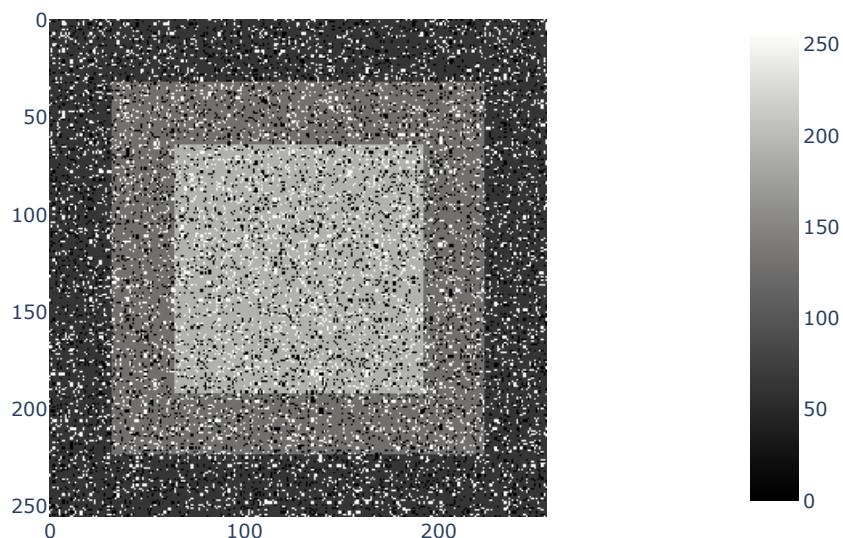
Ukoliko posmatramo sve elemente 3x3 regiona, domain je potrebno definisati kao matricu sa svim jedinicama, a za određivanje median vrednosti, indeks od interesa biće 4 (centralni u opsegu od 0 do 8).

Shodno tome, Min filter je definisan sa indeksom 0 a Max sa indeksom 8.

Funkcija uvek radi zero-padding.

```
[22]: import plotly.express as px
import numpy as np

fig = px.imshow(img_sb, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



```
[23]: from scipy import signal
```

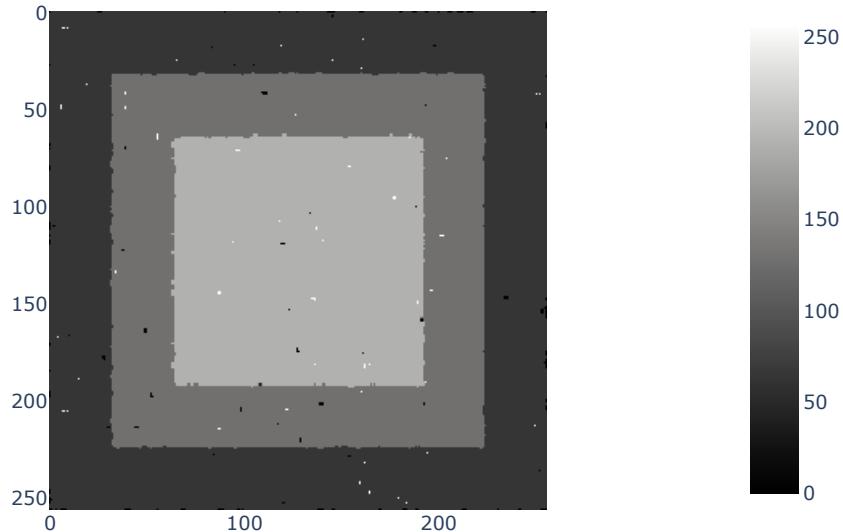
```

region = np.ones((3,3))

img_f = signal.order_filter(img_sb, domain=region, rank=4)

fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()

```



Možemo primetiti da su neki pikseli ostali sa vrednostima oštećenja. Proces filtriranja se može ponavljati do uspešnog filtriranja svih oštećenja ili dostizanja konvergencije (nakon čega primena filtra daje identičnu sliku).

Alternativno, moguće je koristiti grupu filtara iz `scipy.ndimage` modula koji implementiraju osnovne filtre poretkom sa dodatnom mogućnosti podešavanja proširenja.

- `scipy.ndimage.minimum_filter`
- `scipy.ndimage.maximum_filter`
- `scipy.ndimage.median_filter`

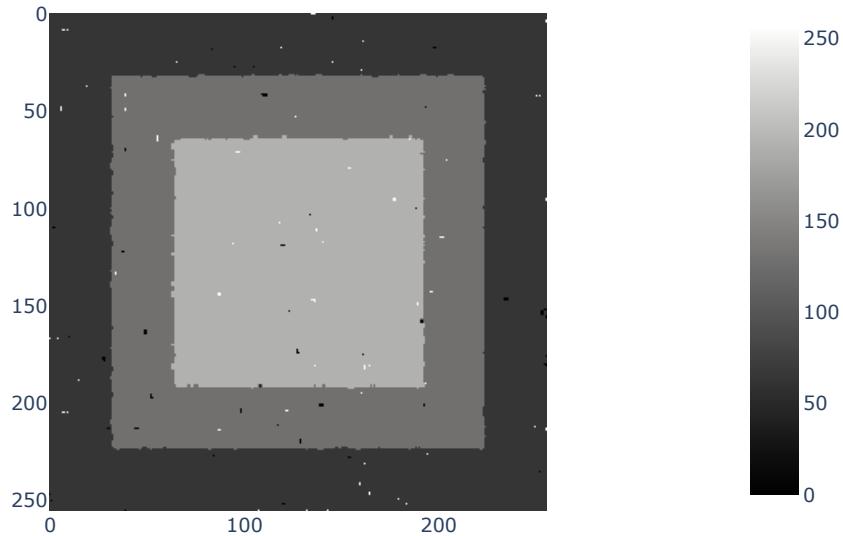
Funkcije primaju parametar slike koja se filtrira, `footprint` parametra koji ima isto ponašanje kao i `domain` u okviru `order_filter()` funkcije i parametar `mode` za vrstu proširenja.

```
[24]: from scipy import ndimage

region = np.ones((3,3))

img_f = ndimage.median_filter(img_sb, footprint=region, mode='reflect')
```

```
fig = px.imshow(img_f, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



4.4 Primer 8.7

Učitati sliku lena.png i dodati joj AWGN varijanse 100 i srednje vrednosti 0. Nad ovom zašumljenom slikom dodati impulsni so i biber šum gustine 5%.

Za potrebe uklanjanja ovog kombinovanog šuma potrebno je napisati funkciju `alphaTrimmedMeanFilt` koja implementira alfa-trimovani usrednjivač (engl. *alpha-trimmed mean*) dat formulom:

$$\hat{f}(x,y) = \frac{1}{mn - \alpha} \sum_{(s,t) \in S_{x,y}} g_r(s,t)$$

gde (s,t) predstavljaju prostorne koordinate regiona $S_{x,y}$ posmatran oko centra pozicije (x,y) , g_r su ulazni intenziteti nakon odbacivanja vrednosti a m, n predstavljaju dimenzijske regiona koji se posmatra. α predstavlja parametar koliko se ukupno najmanjih i najvećih vrednosti odbacuje.

Sliku filtrirati sa posmatranim regionom 3x3 i alfa parametrom od 4. Prikazati rezultat i uporediti PSNR pre i posle filtriranja.

Filtar kombinuje prednosti aritmetičkog usrednjivača i median filtra. Ako posmatramo region postavljen u okolini nekog piksela, vrši se prvo sortiranje vrednosti u rastućem poretku (kao i kod filtara statistike poretku), nakon čega odbacujemo $\alpha/2$ vrednosti sa leve i desne strane ($\alpha/2$ najmanjih i najvećih). Ovin načinom se borimo protiv impulsnog šuma. Od preostalih vrednosti tražimo srednju vrednost, čime pokušavamo da potisnemo Gausov šum.

```
[25]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('lena.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



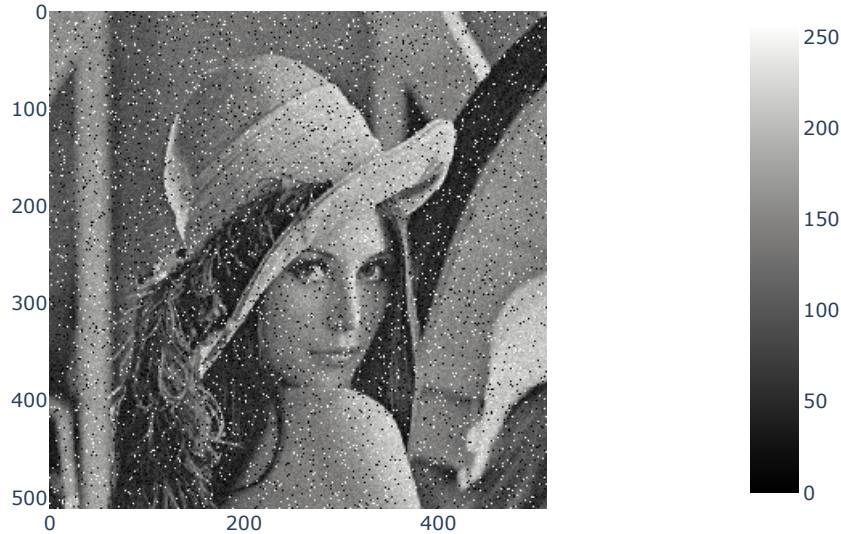
```
[26]: # Gausov sum
rng = np.random.default_rng()
img_noisy = img + rng.standard_normal(size=img.shape)*np.sqrt(100) + 0
img_noisy[img_noisy>255] = 255
img_noisy[img_noisy<0] = 0

# So i biber sum
p = rng.uniform(size=img.shape)
p0 = 0.05
img_noisy[p<p0/2] = 0
```

```



```



Za potrebe sortiranja elemenata u okviru regiona možemo da koristimo `numpy.sort` funkciju. Za parametar `axis=None` region se prevodi u 1D vektor pre sortiranja.

```
[27]: import math

def alphaTrimmedMeanFilt(g, S_shape, alpha):
    f = np.zeros(g.shape)

    pad_rows = math.floor(S_shape[0]/2)
    pad_cols = math.floor(S_shape[1]/2)

    img_p = np.pad(g, ((pad_rows,), (pad_cols,)), mode='symmetric')

    for row in range(f.shape[0]):
        for col in range(f.shape[1]):
            region = img_p[row:row+S_shape[0], col:col+S_shape[1]]

            region = np.sort(region, axis = None)
```

```

res = region[alpha//2:-alpha//2].mean()

f[row, col] = res

return f

```

```
[28]: # 3x3 region
S_shape = (3,3)

img_f = alphaTrimmedMeanFilt(img_noisy, S_shape, alpha=4)

fig = px.imshow(img_f, color_continuous_scale='gray')
fig.show()

psnr1 = 10*np.log10(255**2/((img - img_noisy)**2).mean())
psnr2 = 10*np.log10(255**2/((img - img_f)**2).mean())

print("PSNR za zasumljenu sliku je {}".format(psnr1))
print("PSNR za alfa-trimovani usrednjivac dimenzije {} je {}".format(S_shape, psnr2))
```



PSNR za zasumljenu sliku je 17.961295625624732
 PSNR za alfa-trimovani usrednjivac dimenzije (3, 3) je 31.79103433535889

5 Zadaci za samostalnu vežbu

Zadatak 8.1

Napisati funkciju `midpointFilter` koja implementira midpoint filter dat formulom:

$$\hat{f}(x, y) = \frac{1}{2} \left[\min_{(s,t) \in S_{x,y}} g(s, t) + \max_{(s,t) \in S_{x,y}} g(s, t) \right]$$

gde (s, t) predstavljaju prostorne koordinate regiona $S_{x,y}$ posmatran oko centra pozicije (x, y) i g je ulazna slika.

Funkciju implementirati sa korišćenjem ugrađenih funkcija za filtre poretku.

Funkciju testirati nad proizvoljnom slikom kojoj je dodat Gausov šum varijanse 50 i posmatranjem regiona dimenzije 5x5.

[29] : `#TODO`

Zadatak 8.2

Napisati funkciju `contraHarmonicMeanFilter` koja implementira kontraharmonijski filter usrednjavač dat formulom:

$$\hat{f}(x, y) = \frac{\sum_{(s,t) \in S_{xy}} g(s, t)^{(Q+1)}}{\sum_{(s,t) \in S_{xy}} g(s, t)^Q}$$

gde (s, t) predstavljaju prostorne koordinate regiona $S_{x,y}$ posmatran oko centra pozicije (x, y) , g je ulazna slika a m, n predstavljaju dimenzije regiona koji se posmatra. Parametar Q reguliše način funkcionisanja filtra za otklanjanje ili so ili biber šuma.

Funkciju testirati nad proizvoljnom slikom kojoj je dodat impulsni so šum gustine 10% i nad slikom kojoj je dodat biber šum gustine 10%. Koristiti region dimenzije 5x5 i podešiti parametar Q tako da se mogu ukloniti ove pojedinačne degradacije.

[30] : `#TODO`

Zadatak 8.3

Napisati funkciju `median4Filter` koja implementira median filter koji posmatra samo centralni piksel i njegovih 4 suseda.

Funkciju testirati nad proizvoljnom slikom kojoj je dodat impulsni so i biber šum gustine 5%.

[31] : `#TODO`

6 Dokumentacija novih celina

- [numpy.full](#)
- [numpy.random.default_rng](#)
- [standard_normal](#)
- [numpy.prod](#)
- [uniform](#)
- [scipy.signal.order_filter](#)
- [scipy.ndimage.minimum_filter](#)
- [scipy.ndimage.maximum_filter](#)
- [scipy.ndimage.median_filter](#)
- [numpy.sort](#)

Vezba 9 - Restauracija slike 2

1 Pregled

U okviru vežbe studenti se upoznaju sa procesom uklanjanja AWGN šuma korišćenjem adaptivnog usrednjivača. Dodatno, posmatra se uticaj periodičnog šuma u slici i načine njegovog potiskivanja korišćenjem filtara nepropusnika opsega i noč (engl. *notch*) filtara. Na kraju, analiziraju se načini simuliranja degradacije zamućenja usled pokreta i korišćenjem Vinerovog (engl. *Wiener*) filtra.

2 Napomena za fajlove i pakete koji se koriste u okviru vežbi

- Za rešenje različitih primera u okviru vežbi koristimo određen set fajlova. U okviru PyCharm/Jupyter notebook okruženja ti fajlovi bi trebali da budu na lokalnom disku, dok za Google Colaboratory potrebno je postaviti te fajlove ili koristiti uvezan google drive.
- Korišćene pakete bi trebalo importovati na početku notebook-a samo jednom, međutim u kontekstu učenja (*repetitio est mater studiorum*) svi potrebni paketi će biti importovani za svaki primer

2.1 Primer 9.1

Napisati funkciju `adaptiveMeanFilt` koja implementira adaptivni aritmetički usrednjivač definisan sledećim izrazom:

$$\hat{f}(x, y) = g(x, y) - \alpha[g(x, y) - \bar{z}_{S_{xy}}]$$
$$\alpha = \min\left(\frac{\sigma_n^2}{\sigma_{S_{xy}}^2}, 1\right)$$

gde je g slika koja se filtrira, \bar{z} srednju vrednost okoline S_{xy} a α predstavlja odnos varijanse šuma i lokalne varijanse u okolini S_{xy} .

Funkciju isprobati nad slikom `lena.png` kojoj je dodat AWGN varijanse 100. Koristiti parametre za region dimenzije 3x3. Prikazati dodatno kako izgleda slika α .

Parametar lokalne srednje vrednosti \bar{z} možemo unapred izračunati za svaku poziciju u okviru slike koristeći box filter odgovarajuće veličine definisan okolinom S_{xy} .

Lokalna varijansa je ovde određena korišćenjem `scipy.ndimage.generic_filter` funkcije. Funkcija implementira pokretni region oblika `footprint` parametra nad kojim se računa `function` funkcija i postavlja rezultat na odgovarajuću poziciju izlaza. Kao i ostale funkcije iz `ndimage` modula postoji mogućnost definisanja načina automatskog proširenja slike sa parametrom `mode`. Alternativno, lokalna varijansa se može odrediti i kao razlika između lokalne srednje vrednosti kvadrata slike i kvadrata lokalne srednje vrednosti:

$$\sigma_{S_{xy}}^2 = \bar{z}_{S_{xy}, g^2} - \bar{z}_{S_{xy}}^2$$

Tačna varijansa šuma na slici nam je vrlo retko poznata, ali možemo da prepostavimo da ona predstavlja neki deo globalne varijanse, određen parametrom β u formi:

$$\sigma_n^2 = \frac{\sigma_G^2}{\beta}$$

U okviru zadatka koristiti parametar β od 10.

```
[2]: import numpy as np
import math
from scipy import ndimage

def adaptiveMeanFilt(g, S_shape, beta):

    N = np.prod(S_shape)
    z = ndimage.convolve(g.astype('float'), np.ones(S_shape)/N, mode='mirror')

    sigma_n2 = g.var()/beta

    sigma_S2 = ndimage.generic_filter(g.astype('float'), function=np.var,
    →footprint=np.ones(S_shape), mode='mirror')

    alpha = np.minimum(sigma_n2/sigma_S2, 1)

    f = g - alpha*(g-z)
    #f = (1-alpha)*g + alpha*z
    return f, alpha
```

```
[3]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('lena.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[4]: rng = np.random.default_rng()
img_noisy = img + rng.standard_normal(size=img.shape)*np.sqrt(100) + 0
img_noisy[img_noisy>255]=255
img_noisy[img_noisy<0]=0

fig = px.imshow(img_noisy, color_continuous_scale='gray')
fig.show()

beta = 10
S_shape = (3,3)

img_filt, alpha = adaptiveMeanFilt(img_noisy, S_shape, beta)

fig = px.imshow(img_filt, color_continuous_scale='gray')
fig.show()
fig = px.imshow(alpha, color_continuous_scale='gray')
fig.show()
```





Sa alpha slike možemo primetiti lokacije u kojima je varijansa signala bila veća od varijanse šuma i to su uglavnom bile ivice u okviru slike. Za manju vrednost α filter je uzimao originalne piksele a za veću je uzimao filtrirane vrednosti sa većim udelom. Posledično, primećuje se da je filter očuvao oštре ivice u slici.

3 Periodični šum u slici

Periodični šum se lako može analizirati i ukloniti tehnikama obrade slike u frekvenčnom domenu. Osnovna ideja je da se periodični šum javlja na skoncentrisanim lokacijama definisanim učestanostima smetnji. Uklanjanje se vrši tako što se dizajnira filter koji bi propustio sve učestanosti osim učestanosti na kojima se javila ta smetnja.

Napomena: Naredni primeri i zadaci smatraju da je ciklično proširenje prilikom računanja fft2.

3.1 Primer 9.2

Napisati funkciju gaussBandStopFilter koja implementira Gausov filter nepropusnik opseg dat formulom:

$$H(u, v) = 1 - e^{-\frac{1}{2} \left[\frac{D^2(u, v) - D_0^2}{D(u, v)W} \right]^2}$$

gde su (u, v) prostorne koordinate DFT slike, D je euklidsko rastojanje posmatrano od sredine slike, D_0 je poluprečnik a W širina prstena.

Definisanim filtrom filtrirati periodičnu smetnju prisutnu u slici moon_periodic.png. Dodatno prikazati amplitudsku karakteristiku filtra, 3D prikaz i poprečni presek.

Napomena: Za eventualne probleme deljenja sa 0 koristiti np.finfo('float').eps.

```
[5]: def gaussBandStopFilter(H_size,D0,W):
    u = np.arange(H_size[0]).reshape(-1, 1) - np.floor(H_size[0]/2)
    v = np.arange(H_size[1]) - np.floor(H_size[1]/2)

    D = np.sqrt(u**2 + v**2)
    H = 1-np.exp(-0.5*((D**2-D0**2)/(D*W + np.finfo('float').eps))**2)
    return H
```

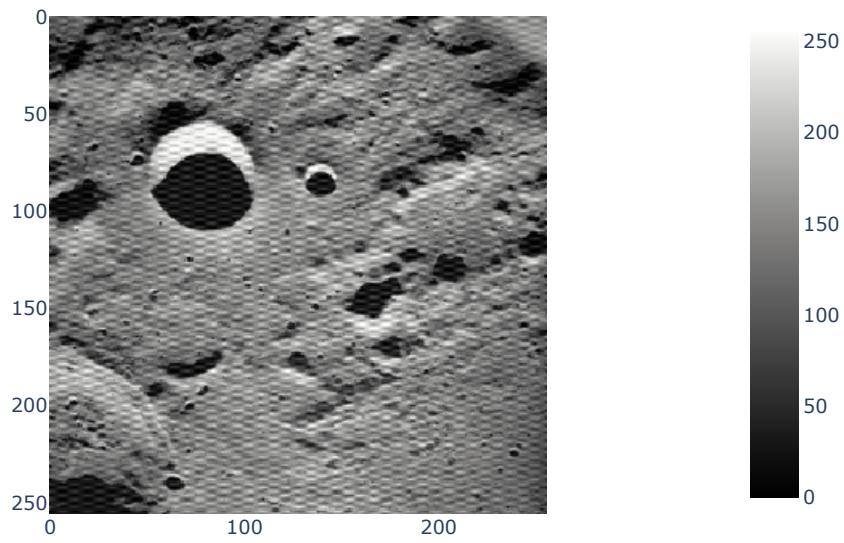
U okviru funkcije imamo deljenje sa matricom D koje u nekom momentu ima vrednost 0 (u sredini slike). Ovo neće praviti neki problem jer u tom slučaju eksponent će težiti $-\infty$ što čini da taj rezultat teži 0 a finalni 1. Da bi se osigurali da nemamo upozorenja na ovo, dodnjem članu dodajemo malu vrednost ϵ definisanu u okviru float tipa. Pristupa se sa komandom np.finfo('float').eps.

```
[6]: import numpy as np
print(np.finfo('float').eps)
```

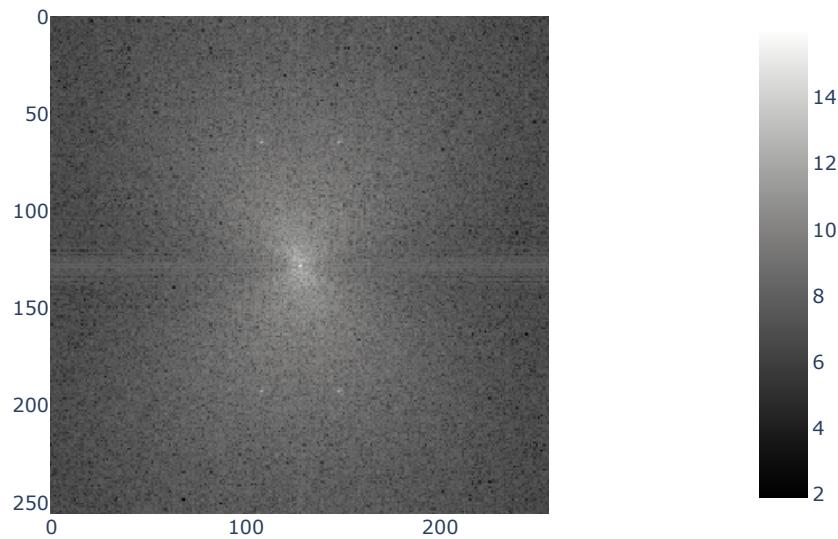
2.220446049250313e-16

```
[7]: import plotly.express as px
from skimage import io

img = io.imread('moon_periodic.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[8]: IMG = np.fft.fftshift(np.fft.fft2(img))
fig = px.imshow(np.log(np.abs(IMG)+1), color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawcircle',
                                         'eraseshape']})
```



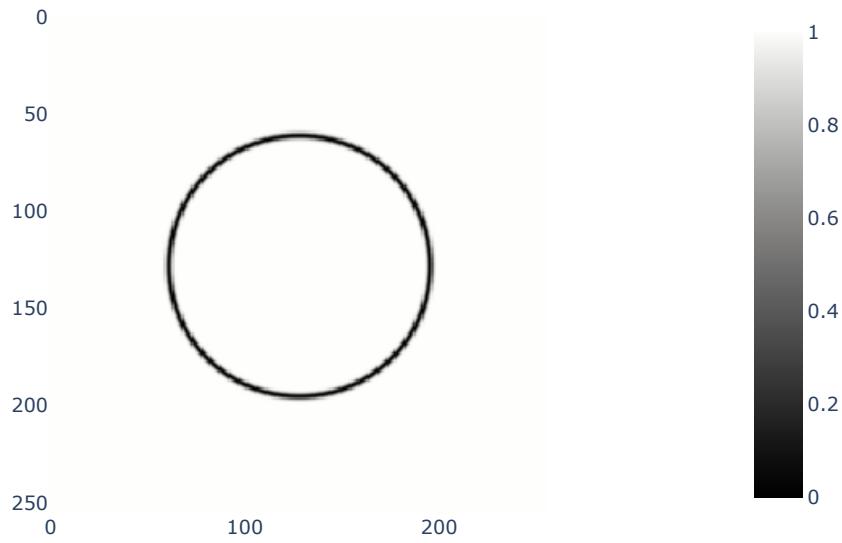
Na osnovu amplitudskog spektra možemo uočiti lokacije pojedinih smetnji i na osnovu njih odrediti potrebne parametre za realizaciju filtra. Prečnik prstena koji bi odgovarao za uklanjanju ove smetnje određuje se na osnovu Euklidskog rastojanja između jednosmerne komponente slike i pozicije smetnje. U okvoru ovog primera jednosmerna komponenta je na koordinati (128, 128) a jedna od smetnji na (64, 108). Rastojanje između njih je:

$$\sqrt{(128 - 64)^2 + (128 - 108)^2} \approx 67$$

```
[9]: print(np.sqrt((128-64)**2+(128-108)**2))
```

67.05221845696084

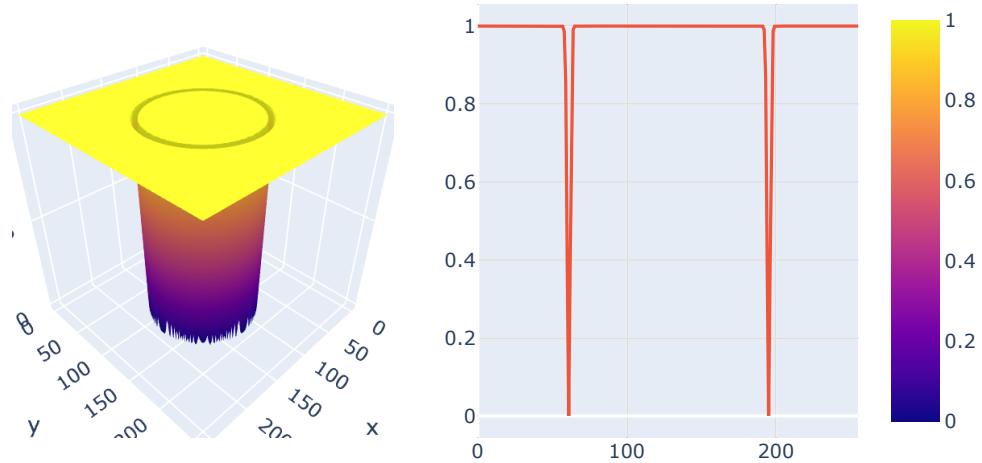
```
[10]: H = gaussBandStopFilter(IMG.shape, 67, 2)
fig = px.imshow(np.abs(H), color_continuous_scale='gray')
fig.show()
```



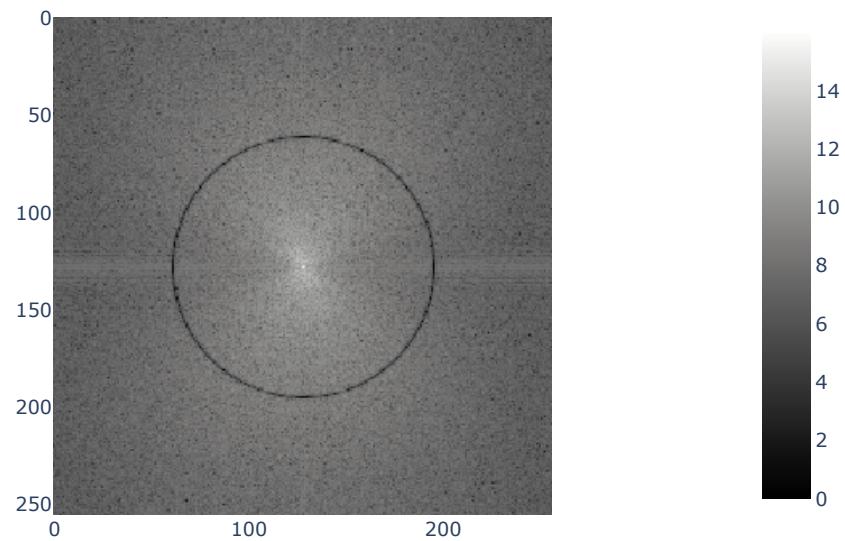
```
[11]: import plotly.graph_objects as go
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'surface'}, {'type': 'scatter'}]])]

fig.add_trace(go.Surface(z=np.abs(H)), row=1, col=1)
fig.add_trace(go.Scatter(y=np.abs(H)[128], mode='lines'), row=1, col=2)
fig.show()
```

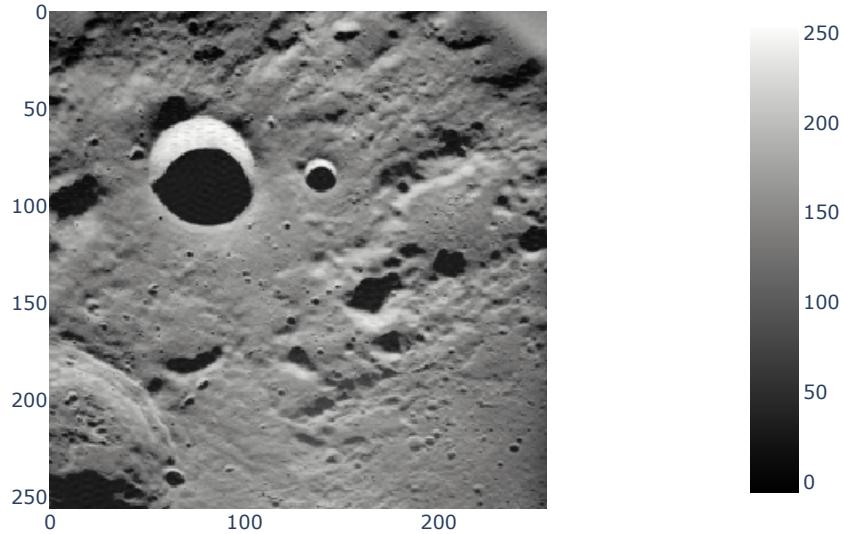


```
[12]: IMG1 = IMG * H
fig = px.imshow(np.log(np.abs(IMG1)+1), color_continuous_scale='gray')
fig.show()
```



```
[13]: img1 = np.real(np.fft.ifft2(np.fft.ifftshift(IMG1)))
img1 = img1[:img.shape[0], :img.shape[1]]

fig = px.imshow(img1, color_continuous_scale='gray')
fig.show()
```



3.2 Primer 9.3

Napisati funkciju `idealNotchFilter` koja implementira idealni noč filter definisan izrazom:

$$H(u, v) = \begin{cases} 0 & D_1(u, v) \leq D_0 \text{ ili } D_2(u, v) \leq D_0 \\ 1 & \text{drugde} \end{cases}$$

gde su (u, v) prostorne koordinate DFT slike, D_1 i D_2 su euklidska rastojanja posmatrana od pozicija para smetnji (u_0, v_0) i $(-u_0, -v_0)$ a D_0 je poluprečnik kružnice.

Prikazati amplitudsku karakteristiku filtra, kao i dodatan 3D prikaz.

Definisanim filtrom filtrirati periodičnu smetnju prisutnu u slici `moon_periodic.png`.

Noč filtri se definišu preko rastojanja $D_1(u, v)$ i $D_2(u, v)$ u odnosu na koordinate (u_0, v_0) i $(-u_0, -v_0)$ frekvencijskih komponenti koje treba potisnuti:

$$D_1(u, v) = \sqrt{(u - \frac{P}{2} - u_0)^2 + (v - \frac{Q}{2} - v_0)^2}$$

$$D_2(u, v) = \sqrt{(u - \frac{P}{2} + u_0)^2 + (v - \frac{Q}{2} + v_0)^2}$$

Član $u - \frac{P}{2}$ i $v - \frac{Q}{2}$ predstavljaju (u, v) koordinate sa koordinatnim početkom u sredini slike.

Svaki notch filter potiskuje jedan par smetnji (jedna smetnja i njena odgovarajuća simetrija u spektru), tako da za potiskivanje više njih potrebno je nadovezati filtre.

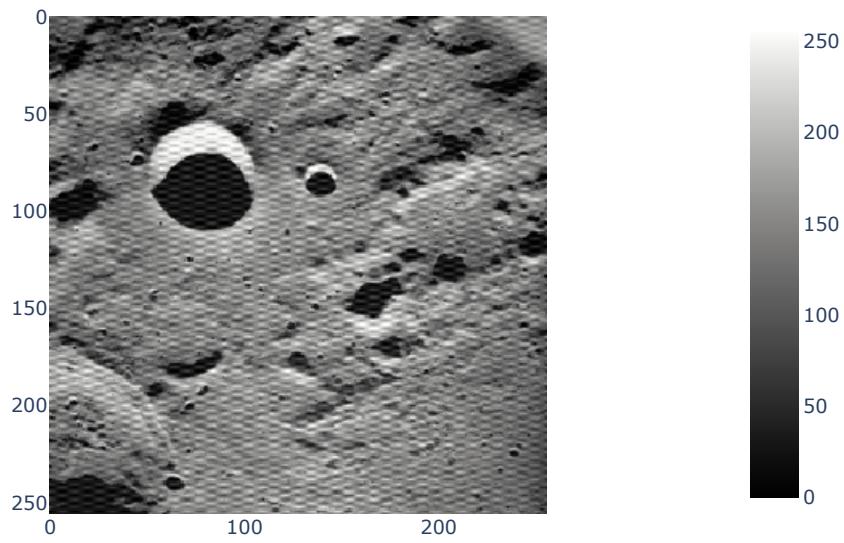
```
[14]: def idealNotchFilter(H_size,D0,u0,v0):
    u = np.arange(H_size[0]).reshape(-1, 1) - np.floor(H_size[0]/2)
    v = np.arange(H_size[1]) - np.floor(H_size[1]/2)

    D1 = np.sqrt((u-u0)**2 + (v-v0)**2)
    D2 = np.sqrt((u+u0)**2 + (v+v0)**2)

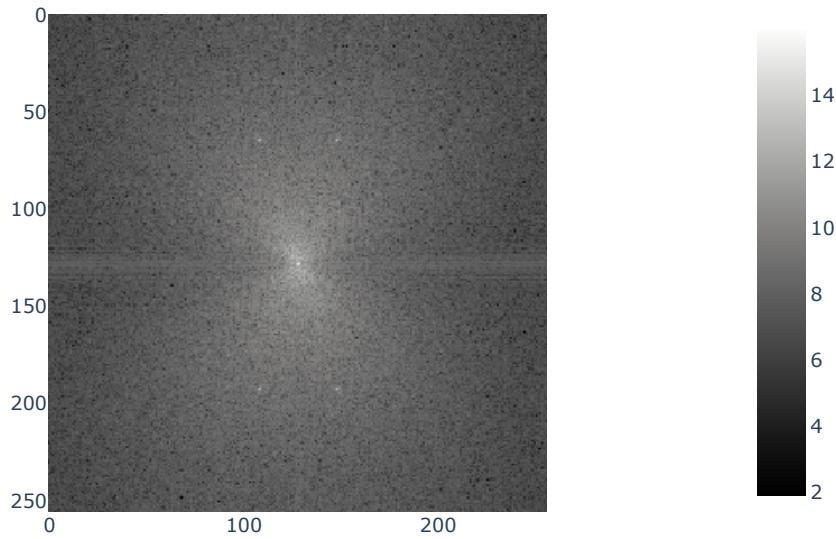
    H = (D1>D0) & (D2>D0)
    return H
```

```
[15]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('moon_periodic.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



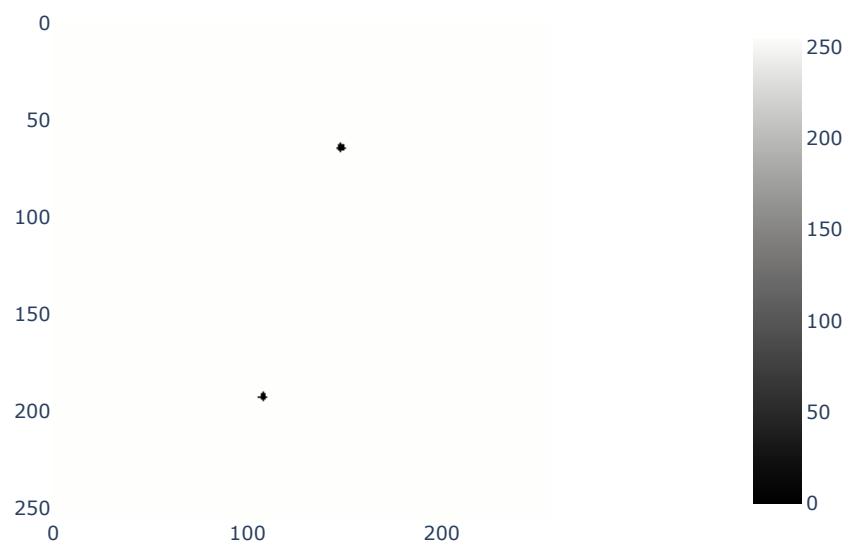
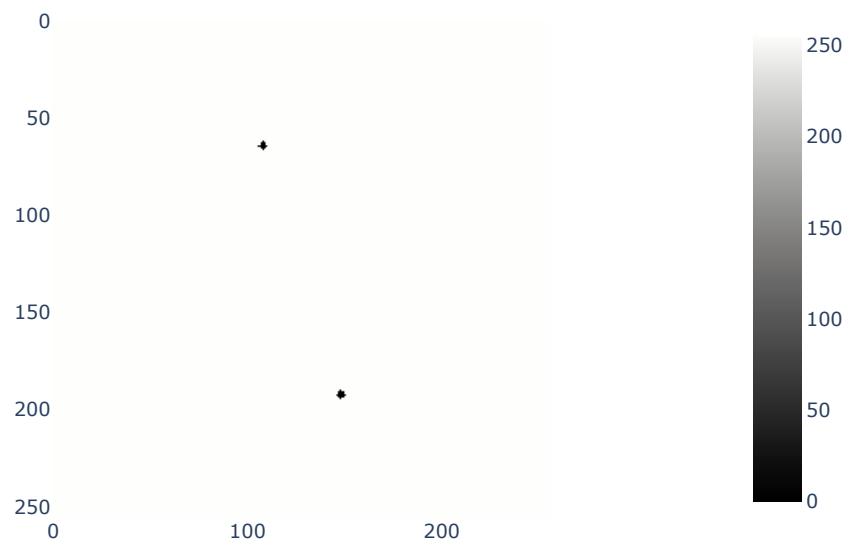
```
[16]: IMG = np.fft.fftshift(np.fft.fft2(img))
fig = px.imshow(np.log(np.abs(IMG)+1), color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawcircle',
                                         'eraseshape']})
```



Sa slike amplitudskog spektra vidimo da je jednosmerna komponenta na koordinati $(128, 128)$ a jedna od smetnji na $(192, 148)$ (donje desna). Ukoliko posmatramo koordinatni sistem (u, v) sa centrom $(0, 0)$ u sredini, pozicija ove smetnje (u_0, v_0) biće razlika ovih koordinata $(192 - 128, 148 - 128) = (64, 20)$, kojoj odgovara simetrična smetnja gore levo $(-u_0, -v_0)$ ili $(-64, -20)$. Ovo odstupanje nam je dovoljno za formiranje jednog noć filtra sa odgovarajućom simetrijom (prosleđivanjem ili jednog ili drugog para koordinata). Drugi par smetnji se lako može odrediti na osnovu prvog pošto su simetrično postavljeni: $(-u_0, v_0)$ i $(u_0, -v_0)$.

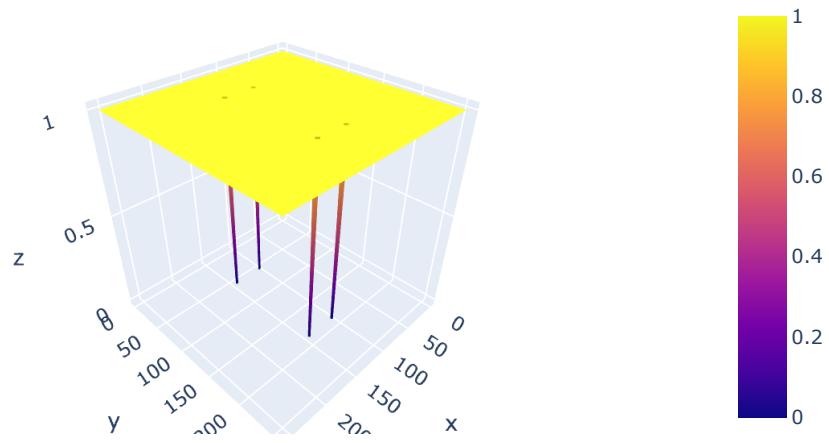
```
[17]: H1 = idealNotchFilter(IMG.shape, 2, 64, 20)
fig = px.imshow(np.abs(H1), color_continuous_scale='gray')
fig.show()

H2 = idealNotchFilter(IMG.shape, 2, -64, 20)
fig = px.imshow(np.abs(H2), color_continuous_scale='gray')
fig.show()
```

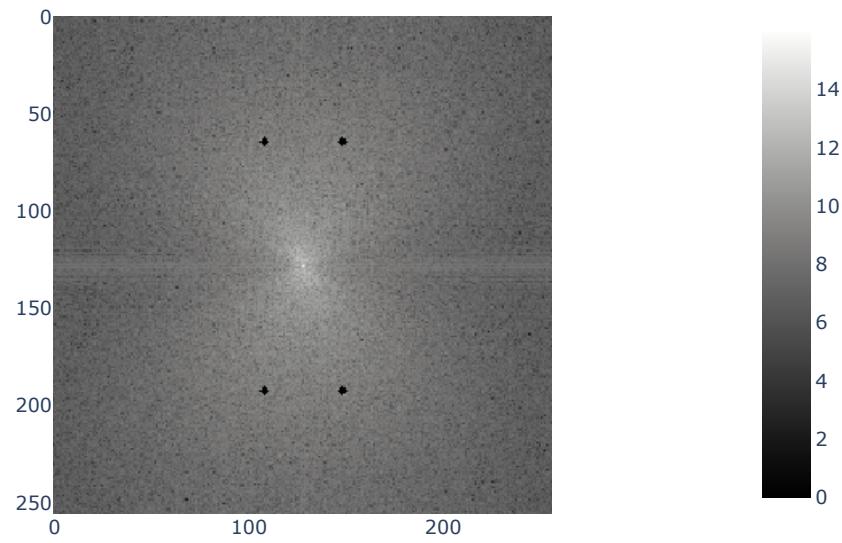


```
[18]: import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Surface(z=np.abs(H1*H2)*1))
fig.show()
```

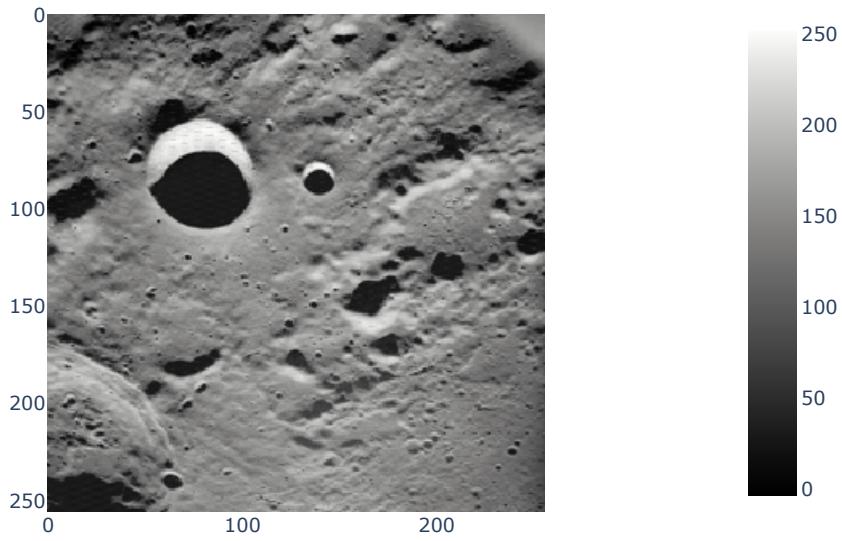


```
[19]: IMG1 = IMG * H1 * H2
fig = px.imshow(np.log(np.abs(IMG1)+1), color_continuous_scale='gray')
fig.show()
```



```
[20]: img1 = np.real(np.fft.ifft2(np.fft.ifftshift(IMG1)))
img1 = img1[:img.shape[0],:img.shape[1]]

fig = px.imshow(img1, color_continuous_scale='gray')
fig.show()
```



3.3 Primer 9.4

Nad slikom `snimak.png` simulirati efekat degradacije atmosferske turbulencije definisane modelom:

$$H(u, v) = e^{-k(u^2+v^2)^{\frac{5}{6}}}$$

gde su (u, v) prostorne koordinate DFT slike sa centrom u sredini slike a k parametar degradacije. Koristiti vrednosti $k = 0.0025$.

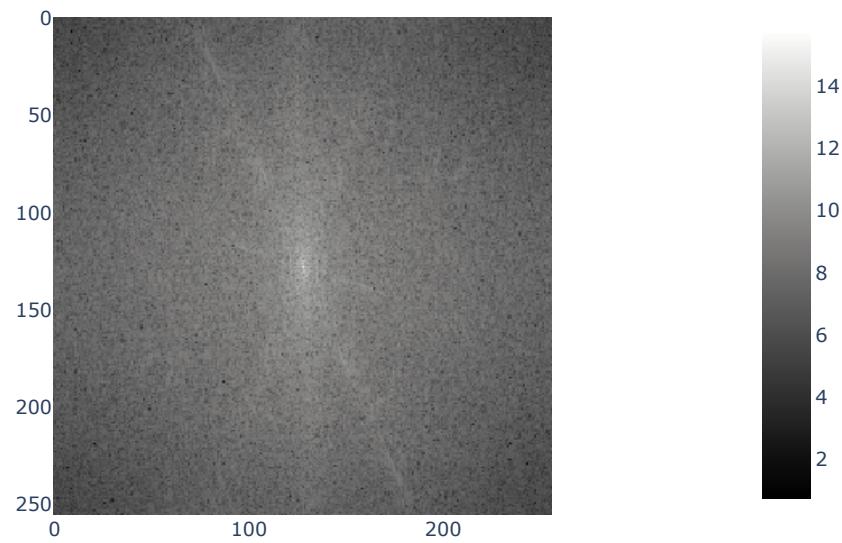
Nad degradiranom slikom dodati AWGN varijanse 1 i pokušati ukloniti degradaciju inverznim filtriranjem a potom koristeći Wiener-ov filter. Prikazati kako izgledaju karakteristike degradacije i Wiener-ovog filtra.

```
[21]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('snimak.png')
fig = px.imshow(img, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



```
[22]: IMG = np.fft.fftshift(np.fft.fft2(img))
fig = px.imshow(np.log(np.abs(IMG)+1), color_continuous_scale='gray')
fig.show()
```

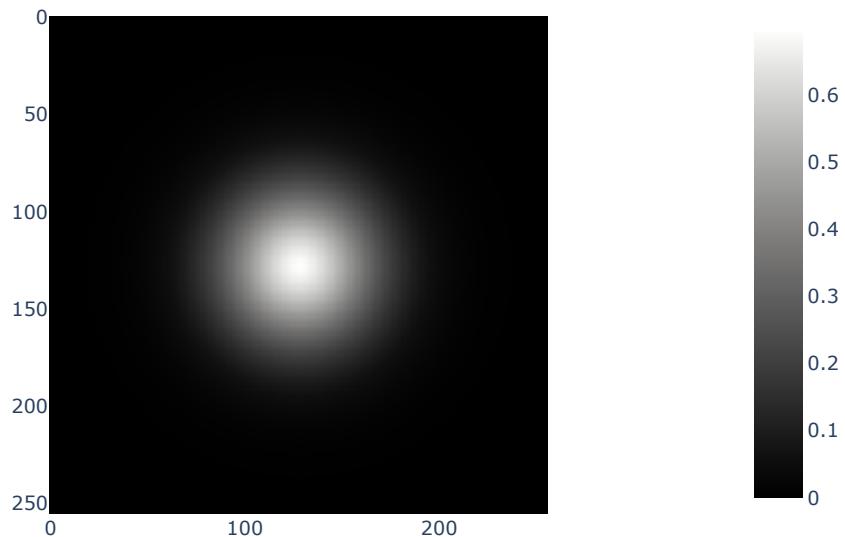


```
[23]: k = 0.0025
```

```
u = np.arange(IMG.shape[0]).reshape(-1, 1) - np.floor(IMG.shape[0]/2)
v = np.arange(IMG.shape[1]) - np.floor(IMG.shape[1]/2)

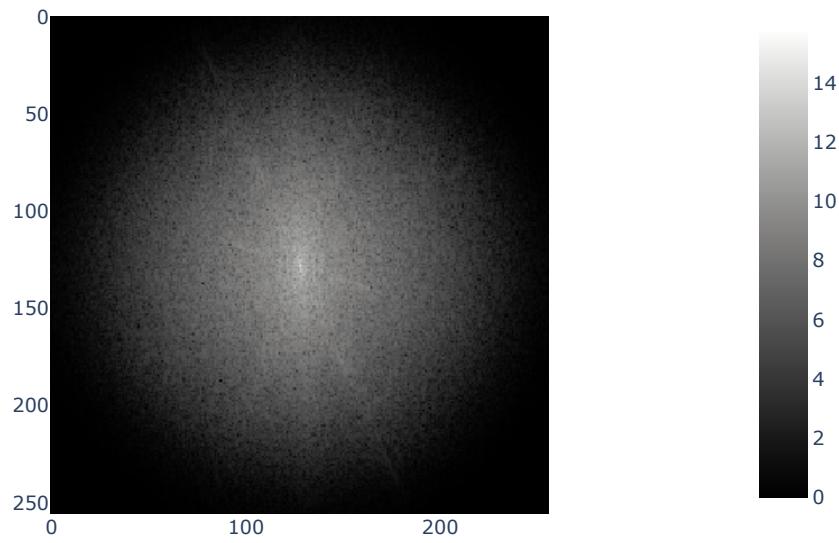
H = np.exp(-k*(u**2+v**2)**(5/6))

fig = px.imshow(np.log(np.abs(H)+1), color_continuous_scale='gray')
fig.show()
```



```
[24]: IMG1 = IMG * H
```

```
fig = px.imshow(np.log(np.abs(IMG1)+1), color_continuous_scale='gray')
fig.show()
```



```
[25]: img1 = np.real(np.fft.ifft2(np.fft.ifftshift(IMG1)))
fig = px.imshow(img1,zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



```
[26]: rng = np.random.default_rng()
noise = rng.standard_normal(size=img1.shape)*np.sqrt(1) + 0

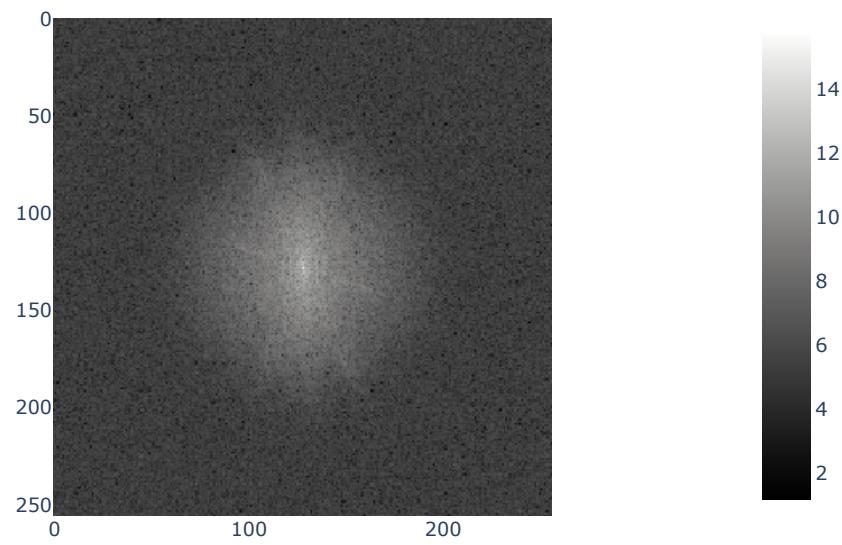
img_noisy = img1 + noise
img_noisy[img_noisy>255]=255
img_noisy[img_noisy<0]=0

fig = px.imshow(img_noisy, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



```
[27]: IMG_NOISY = np.fft.fftshift(np.fft.fft2(img_noisy))

fig = px.imshow(np.log(np.abs(IMG_NOISY)+1), color_continuous_scale='gray')
fig.show()
```

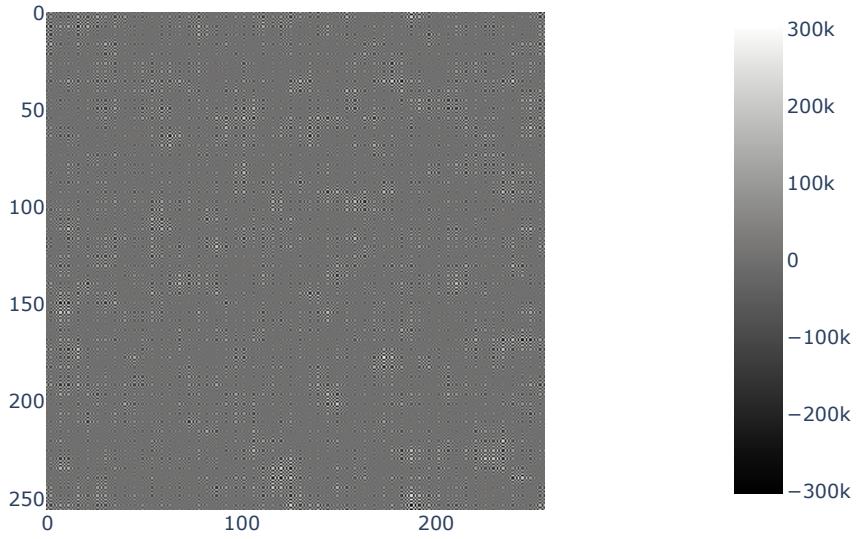


3.3.1 Inverzno filtriranje

```
[28]: IMG_INVF = IMG_NOISY/H

img_invf = np.real(np.fft.ifft2(np.fft.ifftshift(IMG_INVF)))

fig = px.imshow(img_invf, color_continuous_scale='gray')
fig.show()
```



Razlog loše restauracije je prisustvo šuma čija spektralna karakteristika nije nužno poznata. U datom slučaju je dodat AWGN šum na degradiranu sliku tako da imamo:

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

Postupkom inverznog filtriranja dobija se:

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

Član $\frac{N(u, v)}{H(u, v)}$ može da dominira rezultatom, posebno na višim učestanostima gde su vrednosti $H(u, v)$ male, što dovodi do nepoželjnog rezultata.

3.3.2 Wiener filter

Vinerov filter uključuje statističke karakteristike šuma, čime rešava nedostatak direktnog inverznog filtriranja. Postavlja se kao:

$$W = \frac{H^*}{|H|^2 + \frac{|N|^2}{|F|^2}}$$

gde je H frekvencijska karakteristika degradacionog filtra, $|F|^2$ je spektar snage originalne slike a $|N|^2$ spektar snage šuma. Najčešće zbog nepoznavanja spektra snage originalne slike odnos $\frac{|N|^2}{|F|^2}$

se zapisuje kao konstanta K koja to aproksimira.

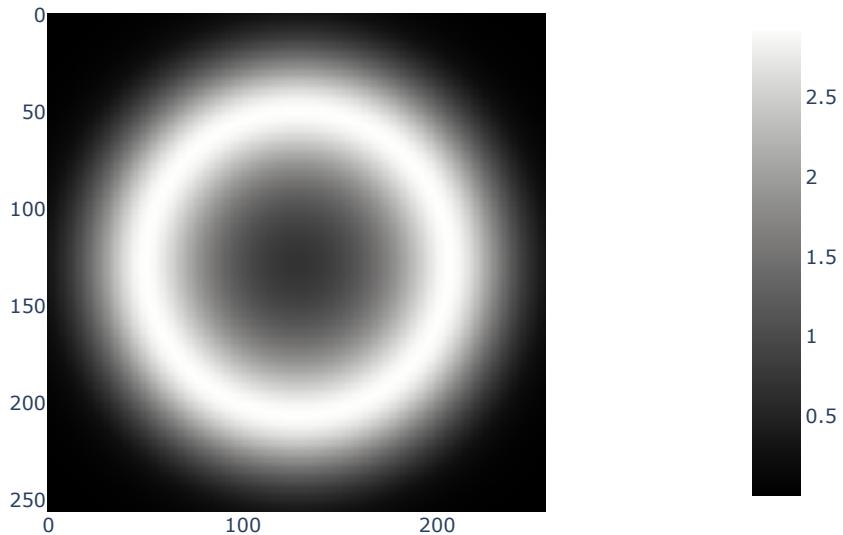
$$W = \frac{H^*}{|H|^2 + K}$$

Parametar K u obliku konstante se može aproksimirati sa odnosom varijanse šuma i varijanse originalne slike ukoliko su poznati.

```
[29]: K = 1/img.var()

W = np.conj(H)/(np.abs(H)**2+K)

fig = px.imshow(np.log(np.abs(W)+1), color_continuous_scale='gray')
fig.show()
```



```
[30]: IMG_INVW = IMG_NOISY * W

img_invw = np.real(np.fft.ifft2(np.fft.ifftshift(IMG_INVW)))

fig = px.imshow(img_invw, zmin=0, zmax=255, color_continuous_scale='gray')
fig.show()
```



4 Zadaci za samostalnu vežbu

4.1 Zadatak 9.1

Napisati funkciju `butterNotchFilter` koja implementira Batervortov notch filter definisan izrazom:

$$H(u, v) = \frac{1}{1 + \left(\frac{D_0^2}{D_1(u,v)D_2(u,v)}\right)^n}$$

gde su (u, v) prostorne koordinate DFT slike, D_1 i D_2 su Euklidska rastojanja posmatrana od pozicija para smetnji (u_0, v_0) i $(-u_0, -v_0)$, D_0 je poluprečnik kružnice a n red filtra.

Prikazati amplitudsku karakteristiku filtra, kao i dodatan 3D prikaz.

Definisanim filtrom filtrirati periodičnu smetnju prisutnu u slici `moon_periodic2.png`.

[31] : #TODO

4.2 Zadatak 9.2

Nad slikom `car.png` simulirati efekat degradacije zamućenja usled pomeranja (engl. *motion blur*) definisan modelom:

$$H(u, v) = \frac{T}{\pi(ua + vb)} \sin(\pi(ua + vb)) e^{-j\pi(ua + vb)}$$

gde su (u, v) prostorne koordinate DFT slike sa centrom u sredini slike, a a, b, T parametri degradacije.

Koristiti vrednosti $a = b = 0.1$ i $T = 1$.

Nad degradiranim slikom dodati AWGN varijanse 1 i pokušati ukloniti degradaciju inverznim filtriranjem a potom koristeći Vinerov filter.

Koristiti np.finfo('float').eps gde je potrebno uzeti u ubzir deljenje sa 0.

[32] : #TODO

4.3 Zadatak 9.3

Napisati funkciju adaptiveMedFilt koja implementira adaptivni median filter. Ovaj filter radi nad promenljivim regionom S_{xy} u zavisnosti od određenih uslova. Kao rezultat filtriranja za pojedini piksel koji je pozicioniran na centar posmatranog regiona S_{xy} dobija se jedna vrednost koja ga zamenuje.

Za potrebe definisanja algoritma koristi se sledeća notacija:

- z_{min} - minimum vrednost u regionu S_{xy}
- z_{max} - maksimum vrednost u regionu S_{xy}
- z_{med} - median vrednost u regionu S_{xy}
- $g(x, y)$ - intenzitet slike sa šumom na poziciji (x, y)
- S_{max} - maksimalna dozvoljena veličina regiona S_{xy}
- $\hat{f}(x, y)$ - intenzitet izlazne slike na poziciji (x, y)

Adaptivni median koristi 2 nivoa obrade (A i B) nad svakom pozicijom (x, y) :

- Prvi nivo A:

if $z_{min} < z_{med} < z_{max}$ idi na nivo B

else povećaj veličinu regiona S_{xy}

if $S_{xy} \leq S_{max}$ idi na nivo A

else $\hat{f}(x, y) = z_{med}$

- Drugi nivo B:

if $z_{min} < g(x, y) < z_{max}$ postavi $\hat{f}(x, y) = g(x, y)$

else $\hat{f}(x, y) = z_{med}$

Smatra se da su mogući regioni neparne širine veće od 1 i kvadratnog oblika.

Algoritam testirati nad slikom lena_60.png. Uzeti da je $S_{max} = 11$.

[33] : #TODO

5 Dokumentacija novih celina

- [scipy.ndimage.generic_filter](#)
- [numpy.finfo\('float'\).eps](#)
- [numpy.conj](#)
- [Annotating image traces](#)

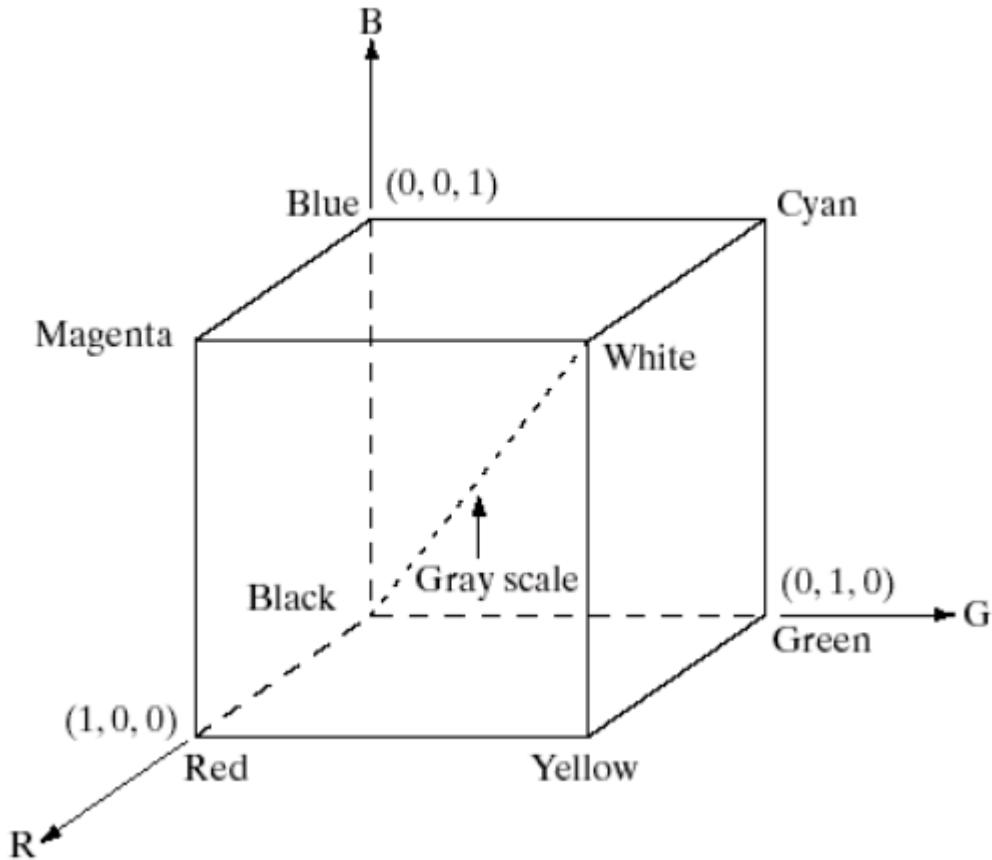
Vezba 10 - Obrada slike u boji

1 Pregled

Cilj vežbe je upoznavanje sa osnovama obrade slike u boji. Analiziraju se različiti sistemi boja, njihove karakteristike i mogućnosti izdvajanje boje u RGB (engl. *Red-Green-Blue*) i HSV (engl. *Hue-Saturation-Value*) prostoru. Analizirano je uklanjanje AWGN šuma u RGB i HSV sistemu boja. Prikazana je mogućnost transformacije monohromatskih slika u slike u boji, odn. pseudo-kolor obrada slike.

2 Kolor slika

Slike u boji se predstavljaju kao trodimenzionalne matrice. Za razliku od piksela sive slike, kojem odgovara skalarna vrednost, pikselu slike u boji odgovaraju minimum tri vrednosti. Pri osnovnom učitavanju slike, elementi vektora predstavljaju intenzitete crvene, zelene i plave komponente boje, respektivno. Ovakva predstava boje odgovara **RGB** sistemu boja. U RGB sistemu svaka boja se predstavlja kao tačka u RGB kocki u čijim su temenima osnovne boje, njihove mešavine, kao i crna i bela boja. Siva skala je u RGB kocki na glavnoj dijagonali, odnosno čini geometrijsko mesto tačaka za koje važi $R = G = B$.



3 RGB domen

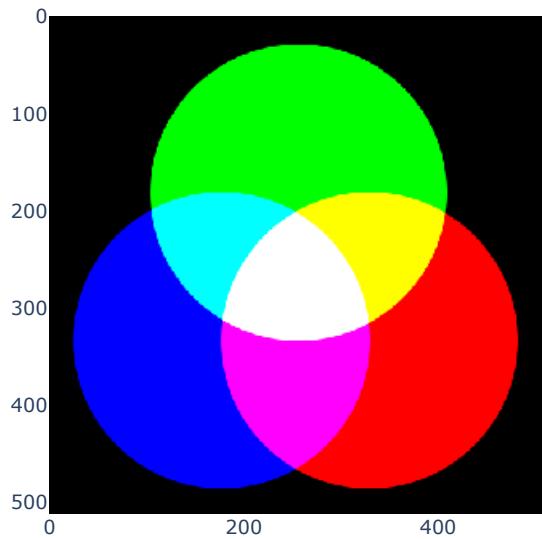
3.1 Primer 10.1

Učitati i prikazati sliku `color_mix.bmp` i intenzitete različitih komponenti boje. Uvećati zeleni kanal za faktor 200 i prikazati modifikovanu sliku.

Učitavanje kolor slike koristeći `io` modul paketa `scikit-image` vrši se na isti način kao kod sive slike. U okviru prikazivanja nije više potrebno koristiti dodatan parametar za kolor skalu.

```
[2]: import plotly.express as px
from skimage import io

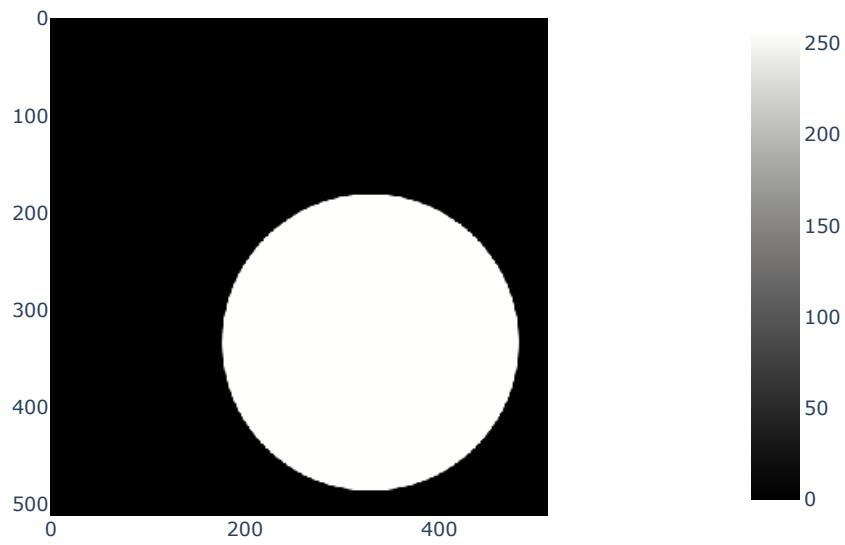
img = io.imread('color_mix.bmp')
fig = px.imshow(img)
fig.show()
```



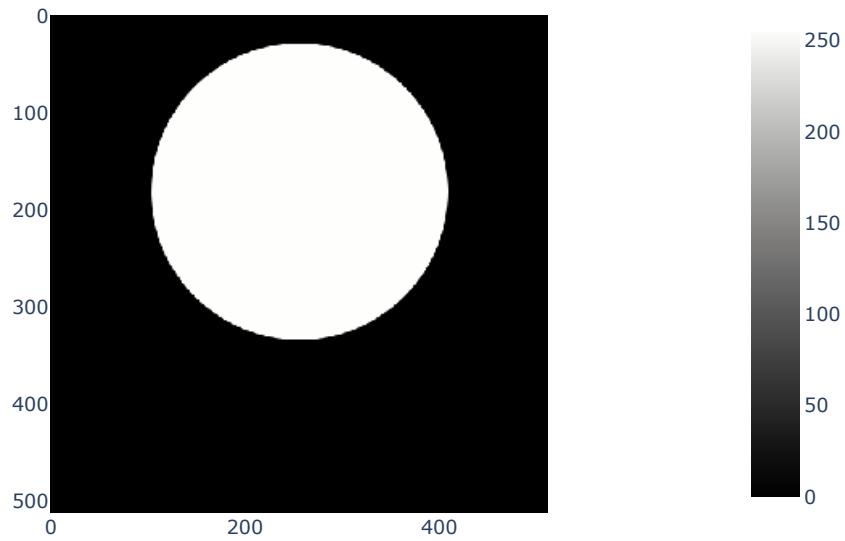
Posmatranjem pojedinačnih vrednosti piksela u okviru slike uočavamo da je svaki piksel sastavljen od trojke vrednosti koje ovde predstavljaju ideo crvene (R), zelene (G) i plave (B) boje.

Pristup pojedinačnim kolor kanalima moguć je korišćenjem indeksiranja gde indeks treće dimenzije označava koji je kanal u pitanju, dok prve dve dimenzije i dalje predstavljaju standardne vrste i kolone.

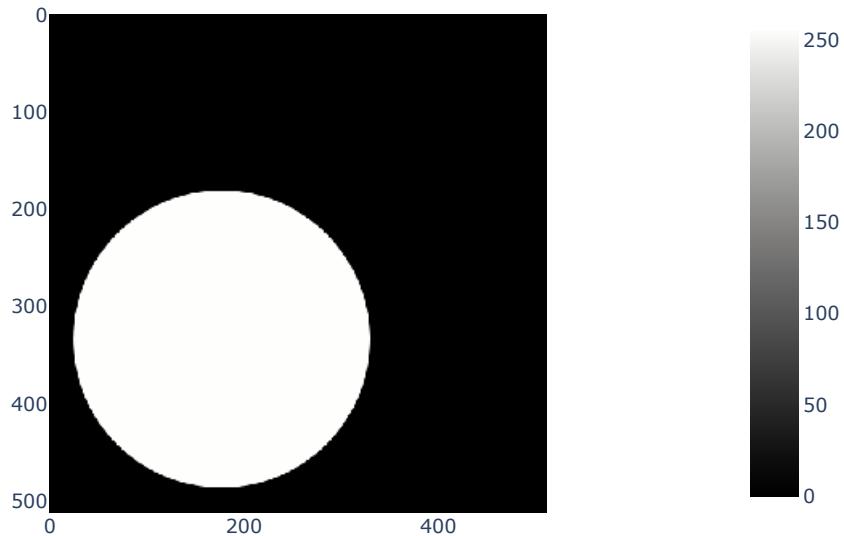
```
[3]: # Red channel
fig = px.imshow(img[:, :, 0], color_continuous_scale='gray')
fig.show()
```



```
[4]: # Green channel  
fig = px.imshow(img[:, :, 1], color_continuous_scale='gray')  
fig.show()
```



```
[5]: # Blue channel  
fig = px.imshow(img[:, :, 2], color_continuous_scale='gray')  
fig.show()
```

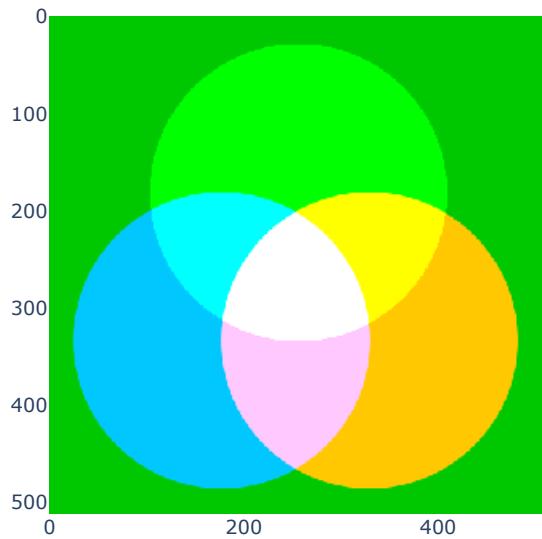


Za korekciju nad zelenim kanalom, potrebno je samo da obratimo pažnju da je potencijalni opseg vrednosti ovako učitane slike ograničen njenim tipom. Za ovu sliku to je opseg definisan uint8 vrednostima.

```
[6]: print(img[:, :, 1].dtype)
```

uint8

```
[7]: Gnew = img[:, :, 1].astype('float') + 200  
# pragovanje  
Gnew[Gnew>255] = 255  
  
img_new = img.copy()  
img_new[:, :, 1] = Gnew  
fig = px.imshow(img_new)  
fig.show()
```

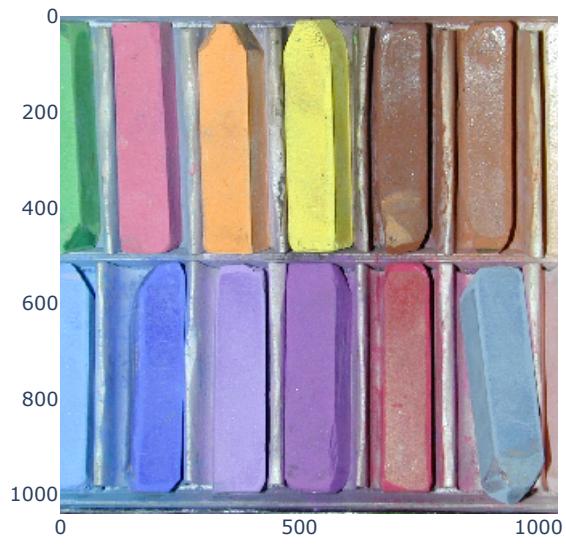


3.2 Primer 10.2

Učitati sliku chalk.tif i korigovati osvetljenje koristeći gama korekciju sa faktorom 1.5.

```
[8]: import plotly.express as px
from skimage import io

img = io.imread('chalk.tif')
fig = px.imshow(img)
fig.show()
```



```
[9]: import numpy as np
```

```
def gammaCorrection(input, gamma):
    c = 255** (1-gamma)
    LUT = c * np.arange(0,256)**gamma
    LUT = np.round(LUT).astype('uint8')
    return LUT[input]
```

```
[10]: img_gamma = gammaCorrection(img,1.5)
fig = px.imshow(img_gamma)
fig.show()
```

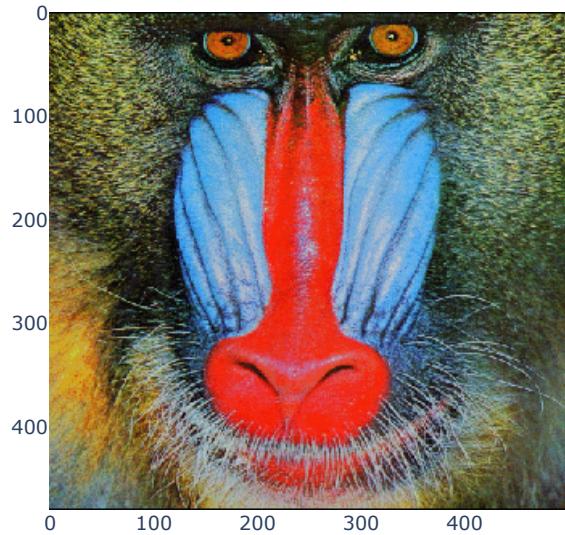


3.3 Primer 10.3

Sa slike baboon.bmp izdvojiti crvene elemente. Smatrati da se boje elemenata od značaja nalaze u kocki čija je stranica dužine 60 u RGB prostoru, a težište u tački (230, 70, 50).

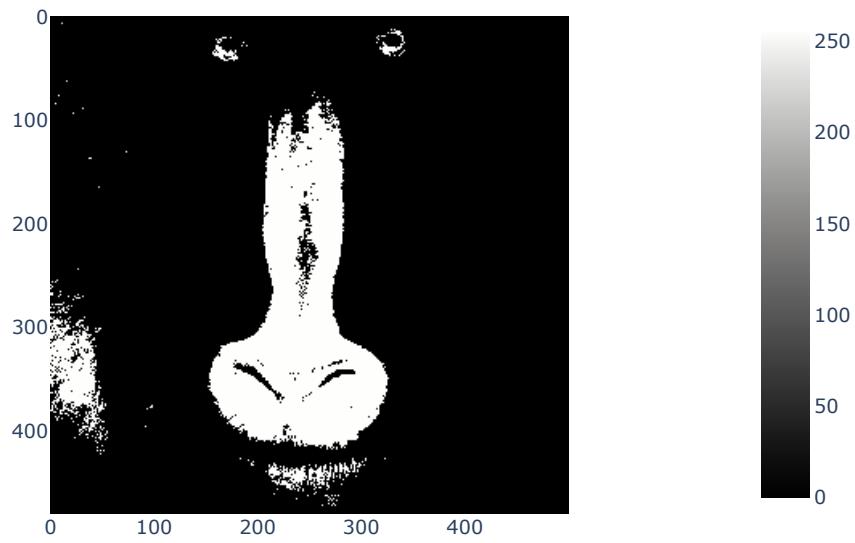
```
[11]: import plotly.express as px
from skimage import io

img = io.imread('baboon.bmp')
fig = px.imshow(img)
fig.show()
```

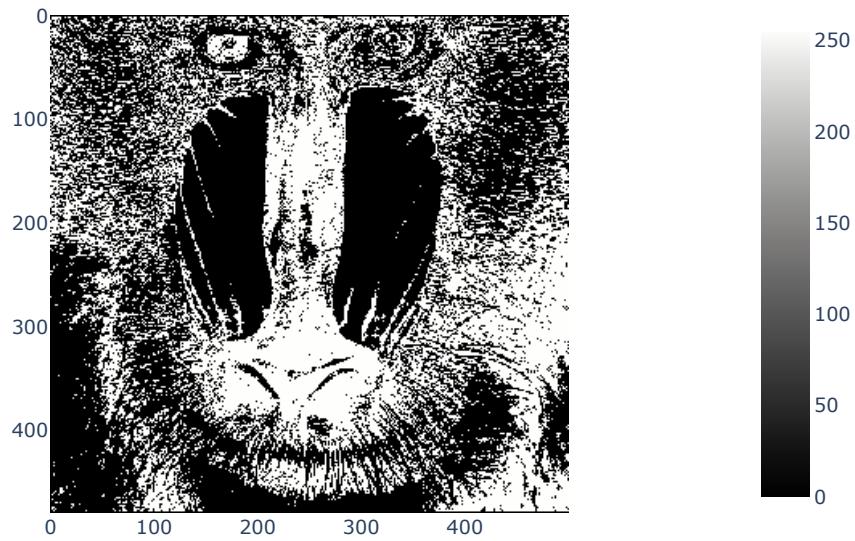


Boja se sa slike izdvaja traženjem piksela čiji vektori boje pripadaju definisanoj kocki. Izdvajanje može da se ostvari po pojedinačnim komponentama boje. Na primer, pikseli čija je R komponenta na apsolutnom rastojanju većem od 30 od R komponente težišta kvadrata se mogu odbaciti. Slično se može uraditi za G i B komponente. Pikseli koji su zadovoljili sva tri uslova se zadržavaju. Odbacivanje piksela u primeru je realizovano preko binarnih slika, r, g i b za svaku od R, G, i B komponenti, respektivno. Na ovim binarnim slikama pikseli koji ne zadovoljavaju uslov označeni su sa 0, a segmentirani pikseli koji pripadaju dатој kocki sa 1.

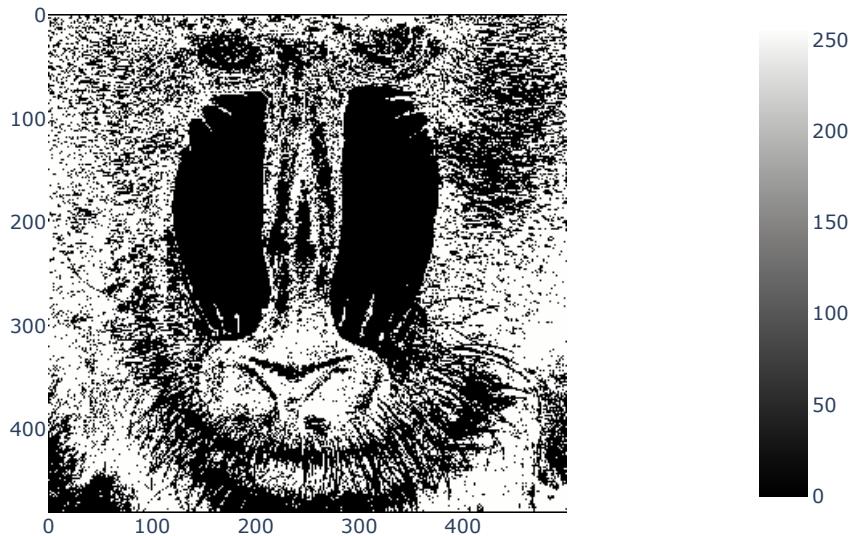
```
[12]: r = np.abs(img[:, :, 0]).astype('float') - 230) <= 30
fig = px.imshow(r, color_continuous_scale='gray')
fig.show()
```



```
[13]: g = np.abs(img[:, :, 1].astype('float') - 70) <= 30
fig = px.imshow(g, color_continuous_scale='gray')
fig.show()
```

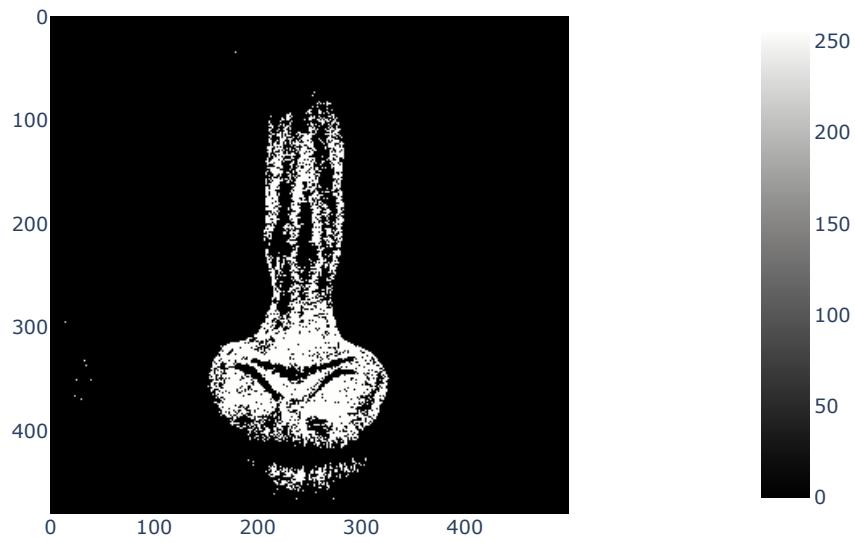


```
[14]: b = np.abs(img[:, :, 2].astype('float') - 50) <= 30
fig = px.imshow(b, color_continuous_scale='gray')
fig.show()
```



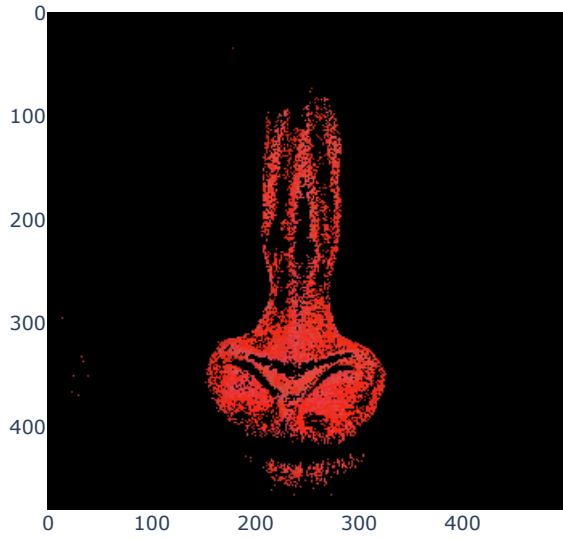
Konačna maska dobija se kombinovanjem binarnih slika r, g i b logičkom operacijom $\&$. Binarne slike se često nazivaju maske.

```
[15]: m = r & g & b
fig = px.imshow(m, color_continuous_scale='gray')
fig.show()
```



Konačno, slika sa izdvojenom bojom od interesa dobija se množenjem binarne maske sa svakom od komponenti boje polazne slike.

```
[16]: img_masked = img * np.stack((m,m,m),axis=2)
fig = px.imshow(img_masked)
fig.show()
```



4 HSI/HSV domen

4.1 Primer 10.4

Odrediti predstavu `color_mix.bmp` slike u HSI (engl. Hue-Saturation-Intensity) domenu koristeći sledeću transformaciju nad RGB slikom:

$$\theta = \cos^{-1} \left\{ \frac{\frac{1}{2}[(R - G) + (R - B)]}{[(R - G)^2 + (R - B)(G - B)]^{\frac{1}{2}}} \right\}$$

$$H = \begin{cases} \theta & \text{ako je } B \leq G \\ 360 - \theta & \text{ako je } B > G \end{cases}$$

$$S = 1 - \frac{3}{R + G + B} [\min(R, G, B)]$$

$$I = \frac{1}{3}(R + G + B)$$

gde su R, G i B skalirane vrednosti pojedinačnih kanala boja na opseg $[0, 1]$. Operaciju transformacije je potrebno implementirati kao funkciju `rgb2hsd`.

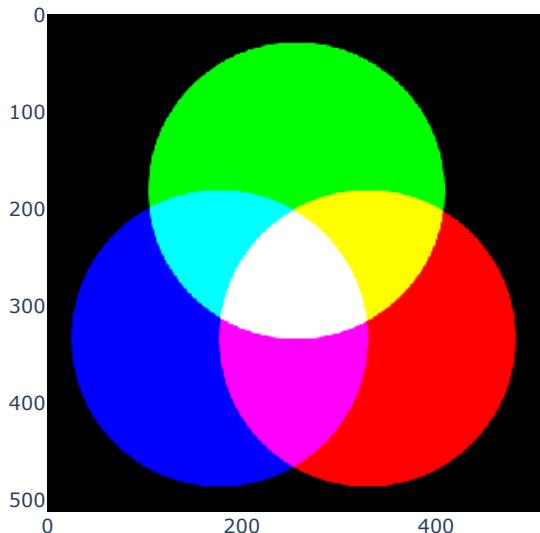
Prikazati pojedinačne komponente HSI slike.

Napomena:

Za piksele kojima odgovaraju sive nijanse ($R = G = B$) smatrati da su H i S vrednosti jednake 0.

```
[17]: import plotly.express as px
from skimage import io

img = io.imread('color_mix.bmp')
fig = px.imshow(img)
fig.show()
```



```
[18]: import numpy as np

def rgb2hsd(img):
    img = img/255

    R = img[:, :, 0]
    G = img[:, :, 1]
    B = img[:, :, 2]

    H = np.rad2deg(np.arccos(0.5*(2*R-G-B)/(np.sqrt((R-G)**2+(R-B)*(G-B))+np.
    -info('float').eps)))
    H[B>G] = 360 - H[B>G]
```

```

Cmin = img.min(axis=2)
GrayMask = (R == G) & (G == B)

H[GrayMask] = 0

S = 1 - 3/(R+G+B+np.finfo('float').eps)*Cmin
S[GrayMask] = 0

I = (R+G+B)/3

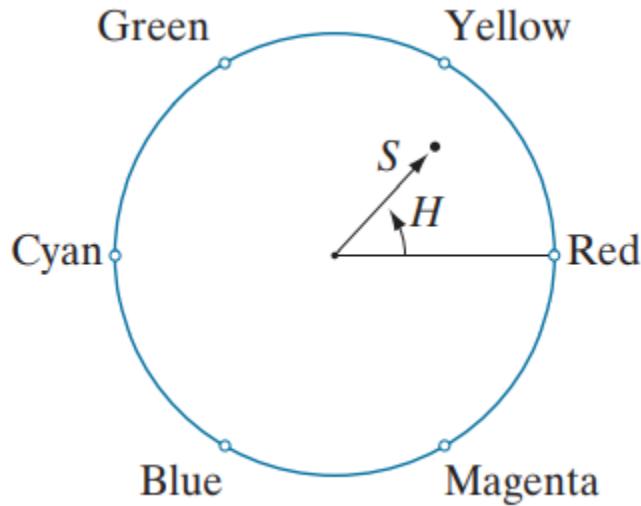
return np.stack((H,S,I),axis=2)

```

[19]: `img_hsi = rgb2hsd(img)`

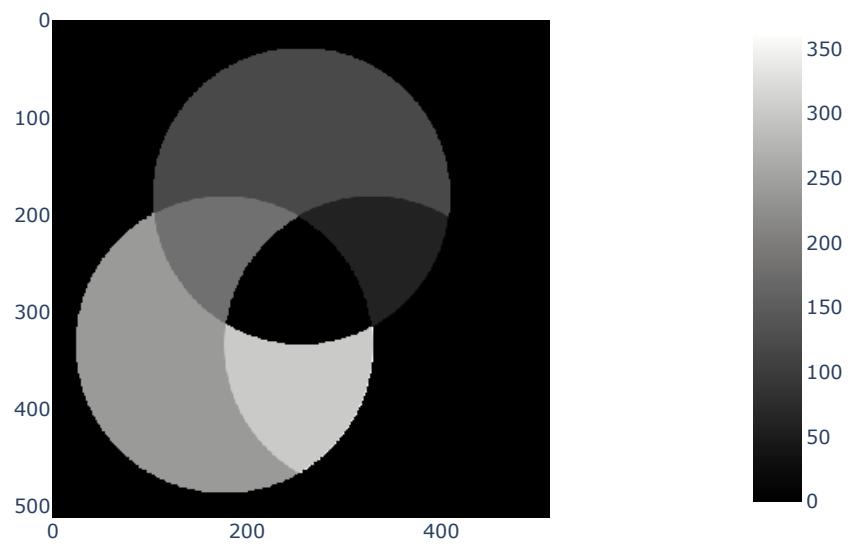
HSI sistem boja nudi intuitivniju predstavu boja u odnosu na RGB sistem:

- H komponenta predstavlja nijansu boje. Predstavlja opseg vrednosti uglova od 0 do 360 gde se meri otklon u odnosu na crvenu boju i svaki predstavlja zasebnu boju.

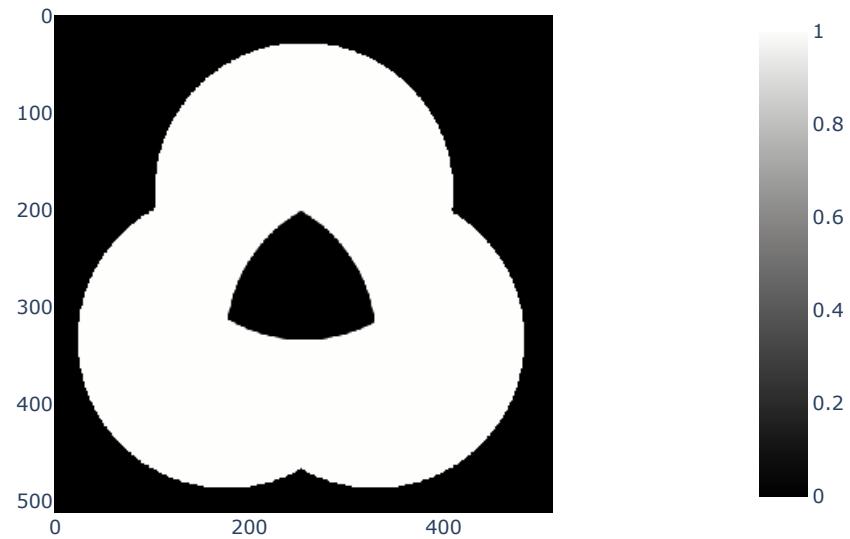


- S komponenta predstavlja zasićenost (čistoću) boje. Opseg vrednosti je od 0 do 1 i što je manja vrednost imamo više primesa sive nijanse.
- I komponenta predstavlja intenzitet boje, odnosno koliko je boja svetla ili tamna.

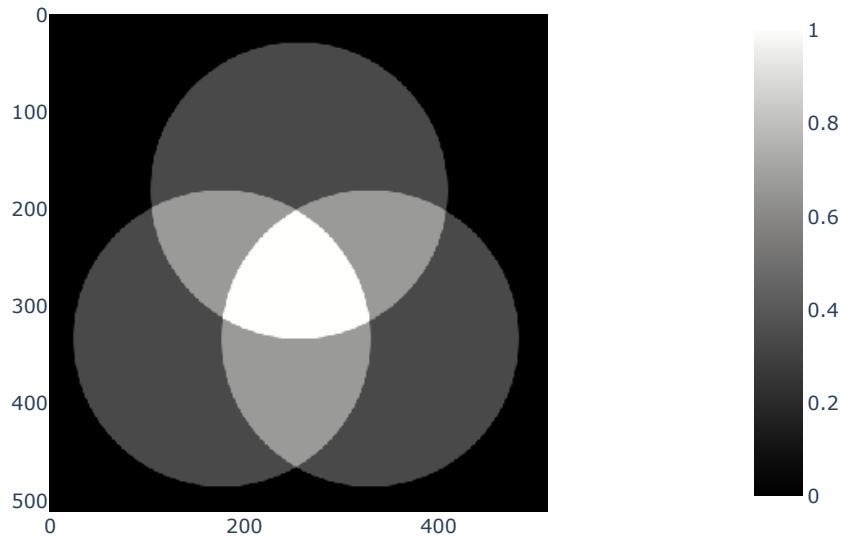
[20]: `fig = px.imshow(img_hsi[:, :, 0], color_continuous_scale='gray')`
`fig.show()`



```
[21]: fig = px.imshow(img_hsi[:, :, 1], color_continuous_scale='gray')
fig.show()
```



```
[22]: fig = px.imshow(img_hsi[:, :, 2], color_continuous_scale='gray')
fig.show()
```



4.2 Primer 10.5

Učitati sliku `lena_color.png` i prevesti je u HSV sistem boja (engl. *Hue-Saturation-Value*).

1. Povećati vrednost H komponente za 0.33, tri puta za redom. Nakon svake promene prikazati sliku.
2. Prikazati slike nastale modifikacijom S komponente za faktor u intervalu od -0.6 do 0.6 sa korakom 0.2.
3. Ekvalizovati histogram V komponente. Prikazati sliku nakon promene.

```
[23]: import plotly.express as px
from skimage import io

img = io.imread('lena_color.png')
fig = px.imshow(img)
fig.show()
```



HSV prostor boja predstavlja domen sličan HSI prostoru ali se znatno češće sreće u obradi kolor slike u okviru ugrađenih funkcija. Razlika je samo u načinu određivanja kanala S i kanala za intenzitet osvetljaja. Mi ćemo ovde koristiti upravo takvu funkciju za jednostavnije prevođenje iz RGB domena koristeći scikit-image paket i funkciju `skimage.color.rgb2hsv`, a isto tako i vraćanje putem `skimage.color.hsv2rgb` funkcije.

```
[24]: from skimage import color

img_hsv = color.rgb2hsv(img)
```

Iako se slika `img_hsv` može prikazati koristeći `imshow` funkciju, uvek prikazujemo pojedinačne kanale jer `imshow` funkcija smatra da je ulaz u RGB domenu. Opseg vrednosti svakog kanala je $[0, 1]$. Inverzna funkcija vraća `float` sliku.

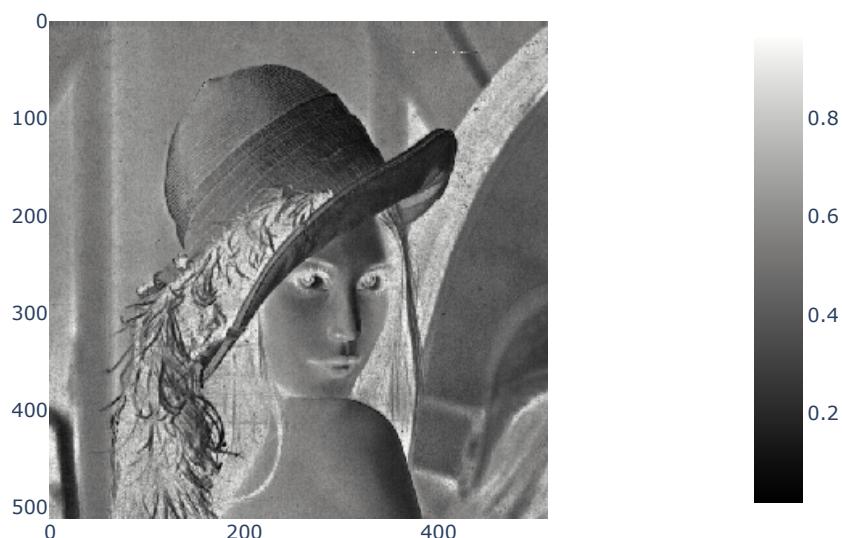
Napomena: funkcija za `float` vrednosti u RGB domenu će vratiti V kanal u intervalu $[0, 255]$. Inverzna funkcija potom vraća u isti opseg.

```
[25]: fig = px.imshow(img_hsv[:, :, 0], color_continuous_scale='gray')
fig.show()

fig = px.imshow(img_hsv[:, :, 1], color_continuous_scale='gray')
fig.show()

fig = px.imshow(img_hsv[:, :, 2], color_continuous_scale='gray')
```

```
fig.show()
```





```
[26]: img_rgb = color.hsv2rgb(img_hsv)

fig = px.imshow(img_rgb)
fig.show()
```



Povećati vrednost H komponente za $\frac{1}{3}$, tri puta za redom. Nakon svake promene prikazati sliku.

Ovde prilikom uvećanja vrednosti trebamo da pazimo da se boje određuju kao ugao u odnosu na poziciju od 0 stepeni koja predstavlja crvenu. U slučaju da vrednost pređe 1 (normalizovanih 360 stepeni) potrebno je ciklično ponavljati vrednosti. Ovde je to obezbeđeno korišćenjem funkcije `numpy.remainder` ili operatora `%` u okviru `numpy` niza.

```
[27]: import plotly.graph_objects as go
import numpy as np

# Create figure
fig = go.Figure()

increment = 1/3

# Add traces, one for each slider step
for i in range(3):
    img_hsv[:, :, 0] = (img_hsv[:, :, 0] + increment) % 1
    out = color.hsv2rgb(img_hsv)

    # izlazna slika je 'float' slika i radi prikaza u colab-u pretvoricemo u uint8
    # like
    out = np.round(255*out).astype('uint8')
```

```

fig.add_trace(go.Image(z=out,name="faktor ="
→"+str((i+1)*increment),visible=False))

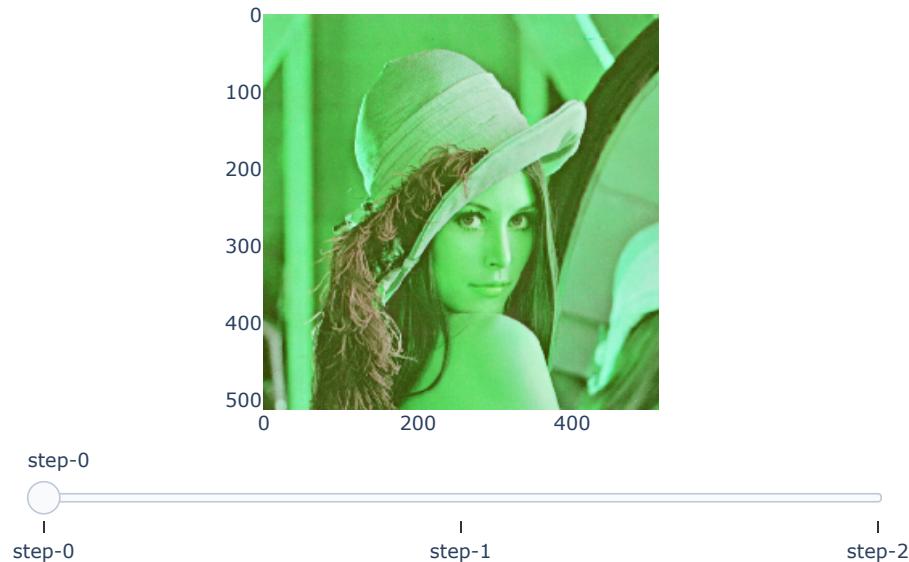
# Make the 1st visible
active_idx = 0
fig.data[active_idx].visible = True

# Create and add slider
steps = []
for i in range(len(fig.data)):
    step = dict(method="update",
                args=[{"visible": [False] * len(fig.data)},
                      {"title": "Faktor : " + str((i+1)*increment)}])
    step["args"][0]["visible"][i] = True # Toggle i'th trace to "visible"
    steps.append(step)

sliders = [dict(active=active_idx,steps=steps)]

fig.update_layout(sliders=sliders)
fig.show()

```



Prikazati slike nastale modifikacijom S komponente za faktor u intervalu od -0.6 do 0.6 sa kora-

kom 0.2.

```
[28]: # Create figure
fig = go.Figure()

s_factor = [-0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6]

# Add traces, one for each slider step
for i in s_factor:
    img_hsv = color.rgb2hsv(img)

    img_hsv[:, :, 1] = np.maximum(np.minimum(img_hsv[:, :, 1] + i, 1), 0)
    out = color.hsv2rgb(img_hsv)

    # izlazna slika je 'float' slika i radi prikaza u colab-u pretvoricemo u uint8
    # slice
    out = np.round(255*out).astype('uint8')

    fig.add_trace(go.Image(z=out, name="faktor = "+str(i), visible=False))

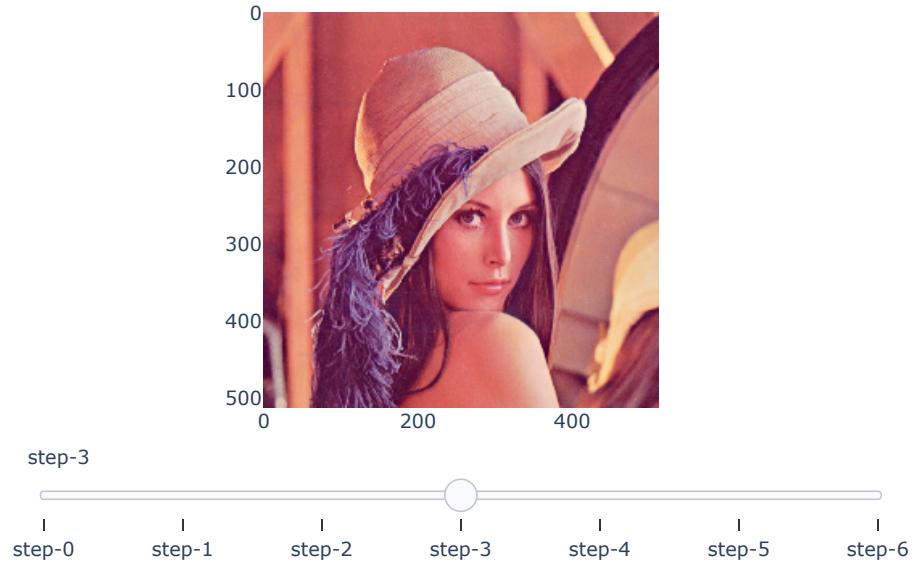
# Make the center visible
active_idx = 3
fig.data[active_idx].visible = True

# Create and add slider
steps = []
for i in range(len(fig.data)):
    step = dict(method="update",
                args=[{"visible": [False] * len(fig.data)},
                      {"title": "Faktor : " + str(s_factor[i])}])

    step["args"][0]["visible"][i] = True # Toggle i'th trace to "visible"
    steps.append(step)

sliders = [dict(active=active_idx, steps=steps)]

fig.update_layout(sliders=sliders)
fig.show()
```



Ekvalizovati histogram V komponente. Prikazati sliku nakon promene.

```
[29]: from skimage import exposure

img_hsv = color.rgb2hsv(img)
img_hsv[:, :, 2] = exposure.equalize_hist(img_hsv[:, :, 2])

out = color.hsv2rgb(img_hsv)
fig = px.imshow(out)
fig.show()
```



4.3 Primer 10.6

Sa slike elementi.bmp izdvojiti plave elemente koristeći HSV domen.

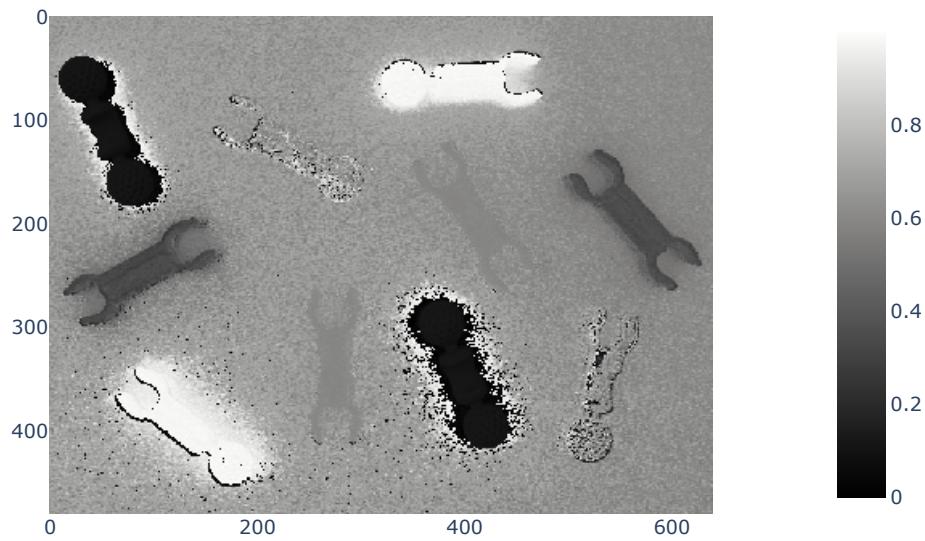
```
[30]: import plotly.express as px
from skimage import io

img = io.imread('elementi.bmp')
fig = px.imshow(img)
fig.show()
```



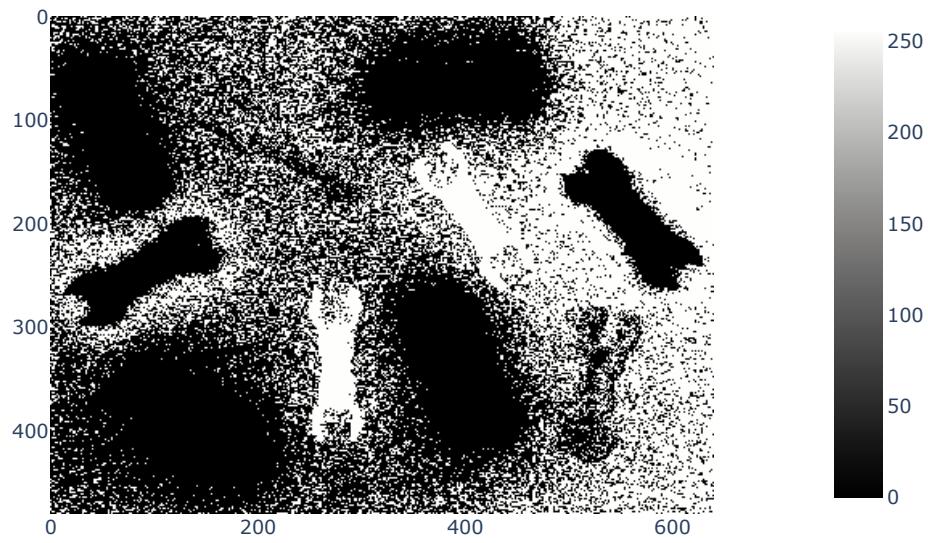
```
[31]: from skimage import color  
  
img_hsv = color.rgb2hsv(img)
```

```
[32]: fig = px.imshow(img_hsv[:, :, 0], color_continuous_scale='gray')  
fig.show()
```



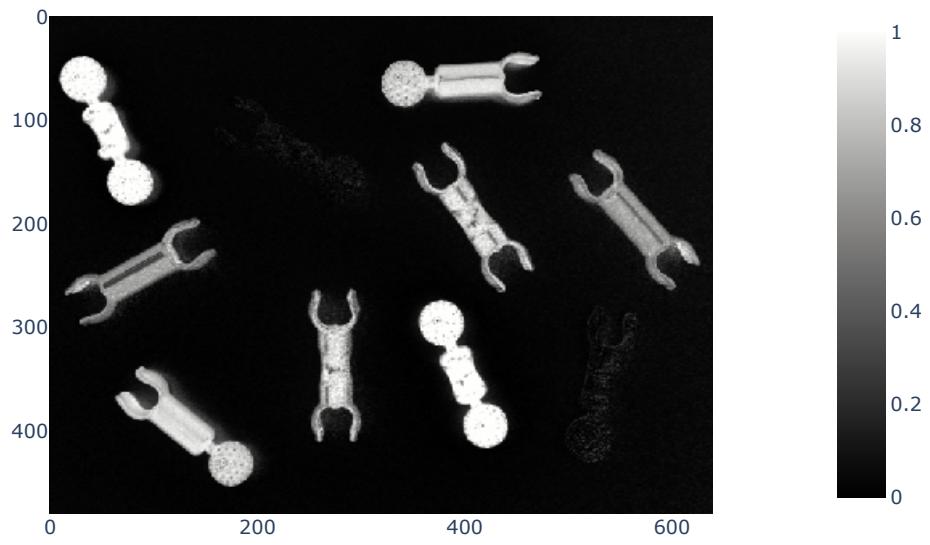
Sa slike H kanala možemo odrediti koji opseg vrednosti bi odgovarao plavim elementima.

```
[33]: mb = (0.55 < img_hsv[:, :, 0]) & (img_hsv[:, :, 0] < 0.65)
fig = px.imshow(mb, color_continuous_scale='gray')
fig.show()
```

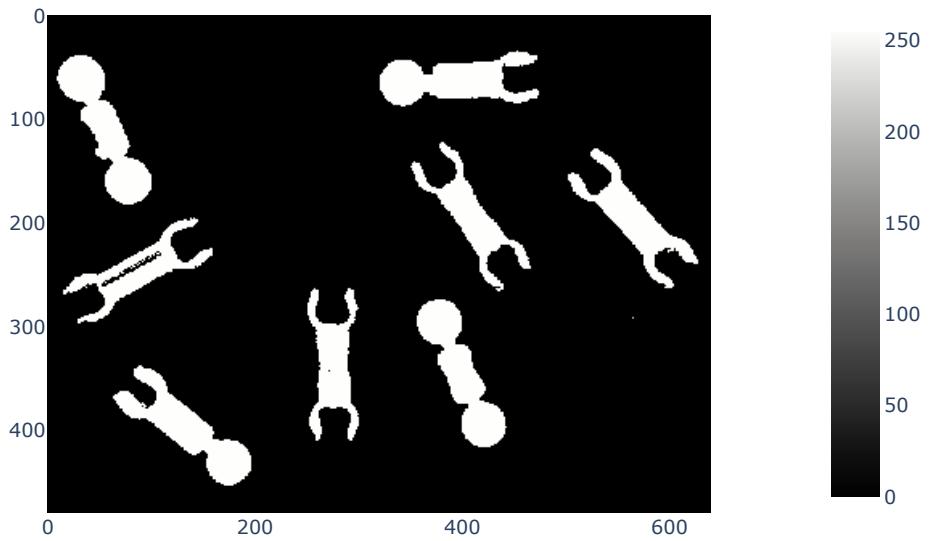


Problem koji se javlja je da ponekad sama informacija o boji nije uvek jedina potrebna za uspešno segmentiranje. U ovom slučaju uključujemo i informacije kanalsa S .

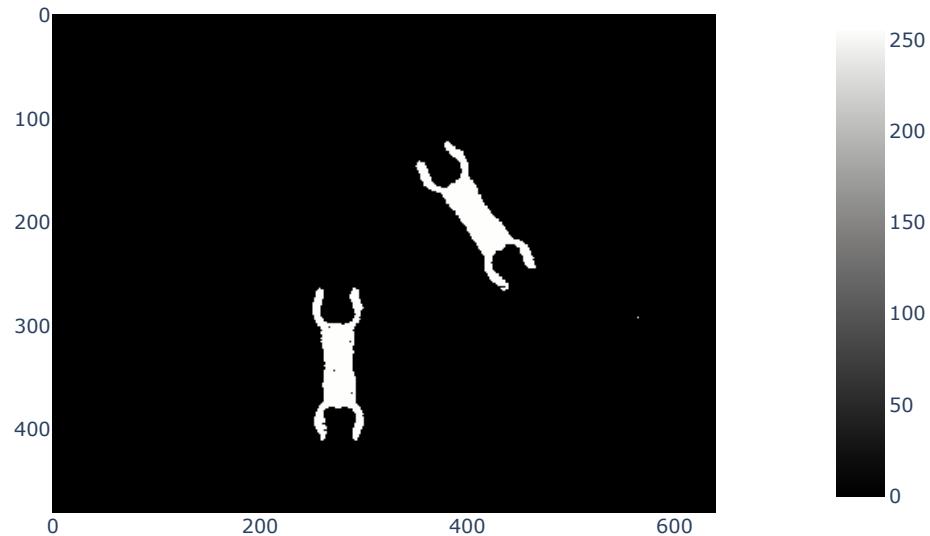
```
[34]: fig = px.imshow(img_hsv[:, :, 1], color_continuous_scale='gray')
fig.show()
```



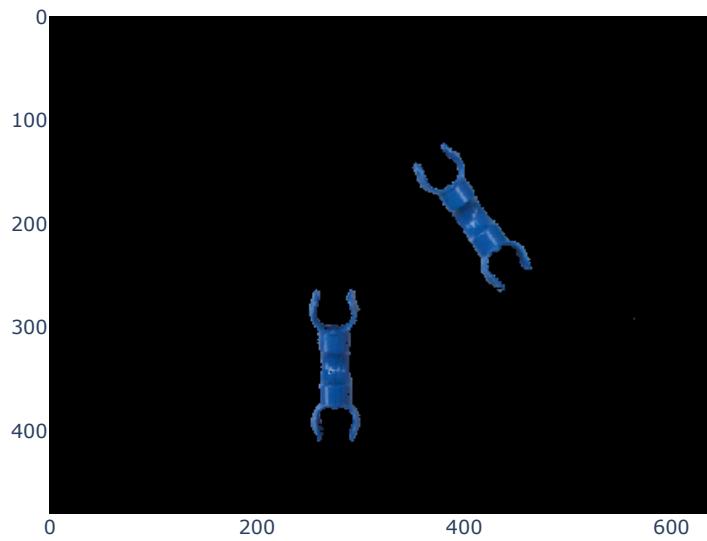
```
[35]: ms = img_hsv[:, :, 1] > 0.35
fig = px.imshow(ms, color_continuous_scale='gray')
fig.show()
```



```
[36]: m = mb & ms
fig = px.imshow(m, color_continuous_scale='gray')
fig.show()
```



```
[37]: img_masked = img * np.stack((m,m,m),axis=2)
fig = px.imshow(img_masked)
fig.show()
```



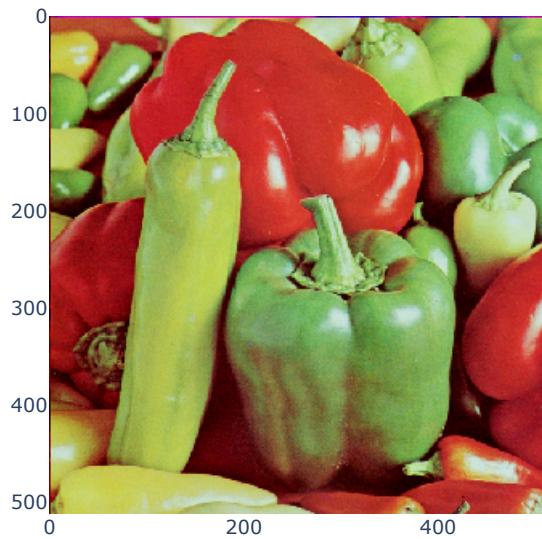
5 Šum u kolor slici

5.1 Primer 10.7

Na svaku od komponenti boja slike `peppers_color.bmp` u RGB sistemu dodati AWGN varijanse 400. Šum ukloniti aritmetičkim usrednjivačem veličine 3×3 sa svake od RGB komponenti. Isti filter upotrebiti na svaku od HSV komponenti i prikazati rezultat.

```
[38]: import plotly.express as px
from skimage import io

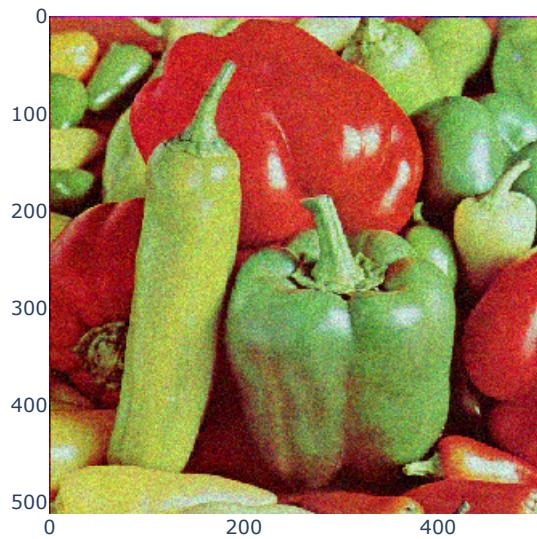
img = io.imread('peppers_color.png')
fig = px.imshow(img)
fig.show()
```



```
[39]: import numpy as np

rng = np.random.default_rng()
img_noisy = img + rng.standard_normal(size=img.shape)*np.sqrt(400) + 0
img_noisy[img_noisy > 255] = 255
img_noisy[img_noisy < 0] = 0

fig = px.imshow(img_noisy)
fig.show()
```



Potiskivanje Gausovog šuma se vrši na klasičan način nad pojedinačnim kanalima slike sa šumom.

```
[40]: from scipy import ndimage

# 3x3 aritmeticki usrednjivac
kernel = np.ones((3,3))/9

img_f = img_noisy.copy()
img_f[:, :, 0] = ndimage.convolve(img_f[:, :, 0], kernel, mode = 'reflect')
img_f[:, :, 1] = ndimage.convolve(img_f[:, :, 1], kernel, mode = 'reflect')
img_f[:, :, 2] = ndimage.convolve(img_f[:, :, 2], kernel, mode = 'reflect')

fig = px.imshow(img_f)
fig.show()

psnr = 10*np.log10(255**2/((img - img_noisy)**2).mean())
print("PSNR pre filtriranja je {}".format(psnr))

psnr = 10*np.log10(255**2/((img - img_f)**2).mean())
print("PSNR za box filter dimenzije {} je {}".format(kernel.shape, psnr))
```



PSNR pre filtriranja je 22.33441242215655

PSNR za box filter dimenzije (3, 3) je 28.13114783904274

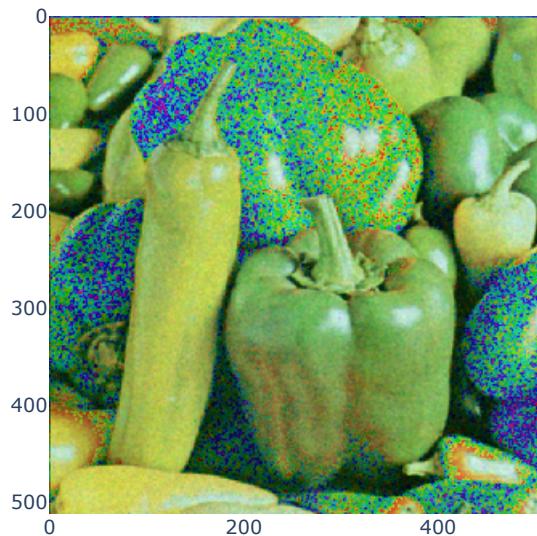
Isti filter upotrebiti pojedinačno na svaku od HSV komponenti i prikazati rezultat.

```
[41]: from skimage import color

img_hsv = color.rgb2hsv(img_noisy)
```

```
[42]: img_f1 = img_hsv.copy()
img_f1[:, :, 0] = ndimage.convolve(img_f1[:, :, 0], kernel, mode = 'reflect')
img_f1[:, :, 1] = ndimage.convolve(img_f1[:, :, 1], kernel, mode = 'reflect')
img_f1[:, :, 2] = ndimage.convolve(img_f1[:, :, 2], kernel, mode = 'reflect')
img_f1 = color.hsv2rgb(img_f1)

fig = px.imshow(img_f1)
fig.show()
```



Filtriranje H komponente u HSV sistemu dovodi do promene boja na slici što je nepoželjan efekat. U dатој конверзији шум се нелинеарно мапирао у H и S компоненту.

Ako se desi da je шум prisutan само u jednoj od RGB komponenti, трансформација у HSV систем uneće шум у сва три канала HSV система и самим тим оtežati njegovo укланjanje.

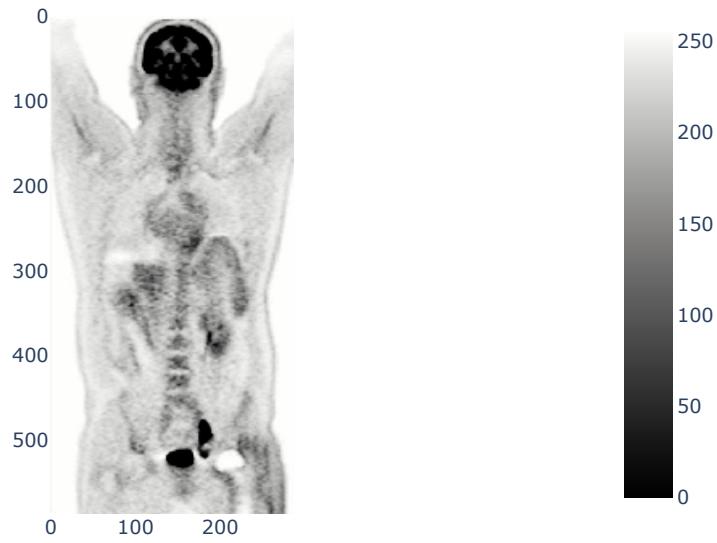
6 Pseudo-kolor obrada

6.1 Primer 10.8

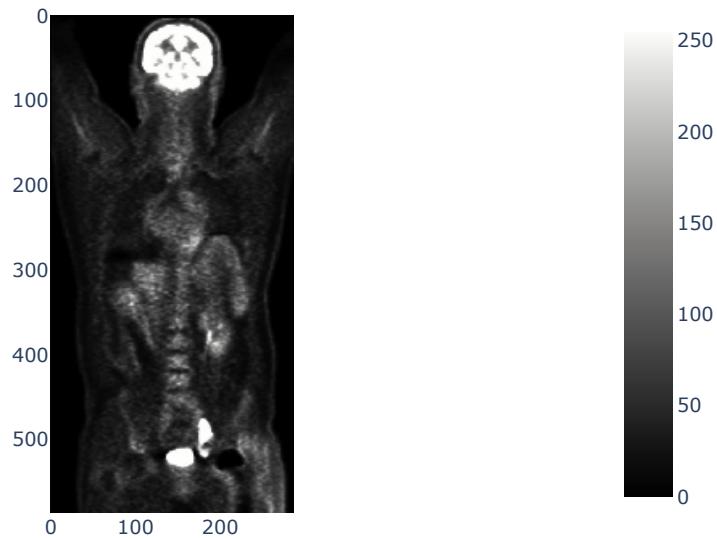
Učitati sliku 2_PET.png i odrediti њен negativ. Пrikazati dobijenu sliku u okviru jet i hsv mape boja.

```
[43]: import plotly.express as px
from skimage import io

img = io.imread('2_PET.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



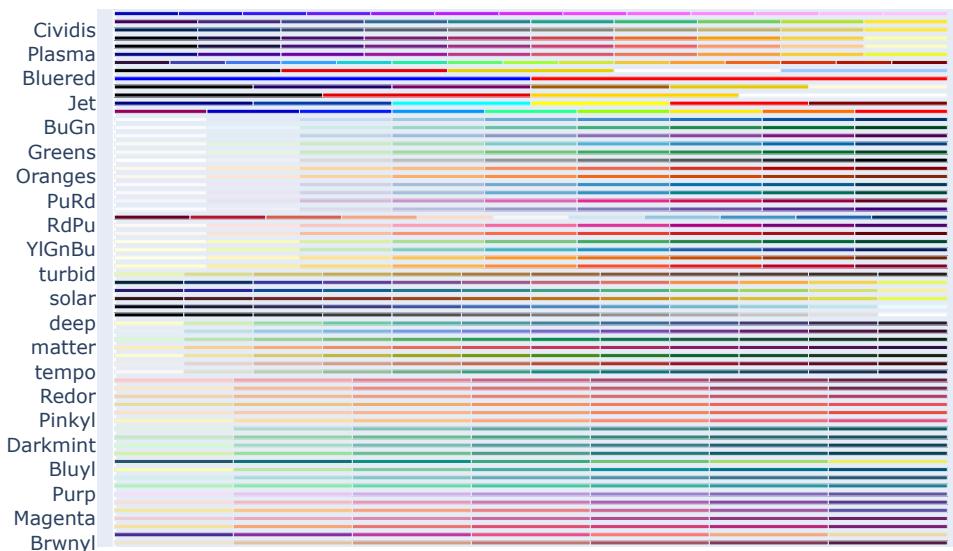
```
[44]: img_neg = 255 - img
fig = px.imshow(img_neg, color_continuous_scale='gray')
fig.show()
```



Mapa boja određuje pridruživanje različitih boja različitim intenzitetima piksela, te se umesto nijanse sive boje, koja bi bila dodeljena pikselu, prikazuje boja određena mapom. U okviry plotly paketa (a i većine drugih paketa za prikaz) ponuđeno je mnoštvo predefinisanih mapa boja. Vizuelni prikaz predefinisanih mapa boja može se obezbediti prikazivanje odgovarajućih uzoraka (tzv. engl. *swatches*) različitih tipova mapa. Ovde je prikazano za sekvencialne i ciklične mape.

```
[45]: fig = px.colors.sequential.swatches()
fig.show()
```

plotly.colors.sequential

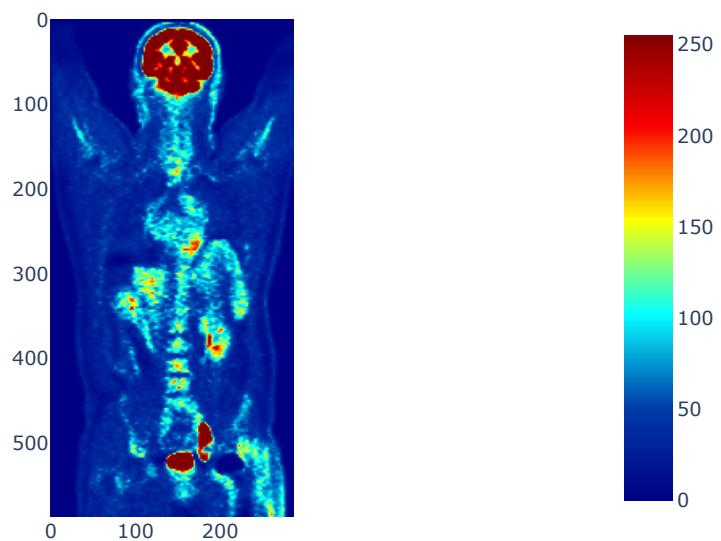


```
[46]: fig = px.colors.cyclical.swatches()
fig.show()
```

`plotly.colors.cyclical`



```
[47]: fig = px.imshow(img_neg, color_continuous_scale='jet')
fig.show()
```

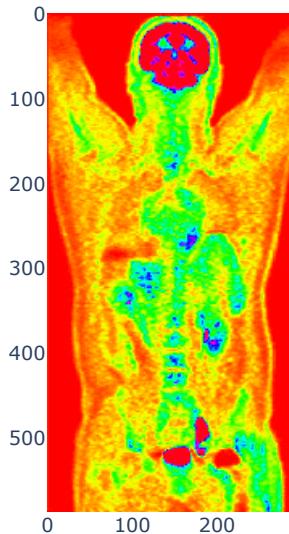


Dobijanje ovog prikaza je obezbeđeno jednostavnim mapiranjem odgovarajućih nijansi sive u definisan set kolor informacija. Jedan ručni pristup za dobijanje ovakve slike je definisanje odgovarajuće mape boja kao LUT. Ovde je to obezbeđeno korišćenjem `matplotlib.cm.get_cmap` funkcije. Naredni primer je za 'hsv' mapu boja.

```
[66]: import matplotlib

cmap = matplotlib.colormaps.get_cmap('hsv')
img_color = cmap(img_neg)[:, :, :3]

fig = px.imshow(img_color)
fig.show()
```



Prikaz pojedinačnih transformacija može se uraditi mapiranjem $[0, 255]$ opsega vrednosti.

```
[67]: import numpy as np

print(cmap(np.arange(256)))
```

[[1.	0.	0.	1.]
[1.	0.02316179	0.	1.]
[1.	0.04632358	0.	1.]
...				

```
[1.          0.          0.14007358 1.          ]
[1.          0.          0.11691179 1.          ]
[1.          0.          0.09375   1.          ]]
```

Napomena: Dobijena mapa boja pretvara u RGBa sliku (dodatni alpha kanal koji reguliše ‘vidljivost’ piksela).

6.2 Primer 10.9

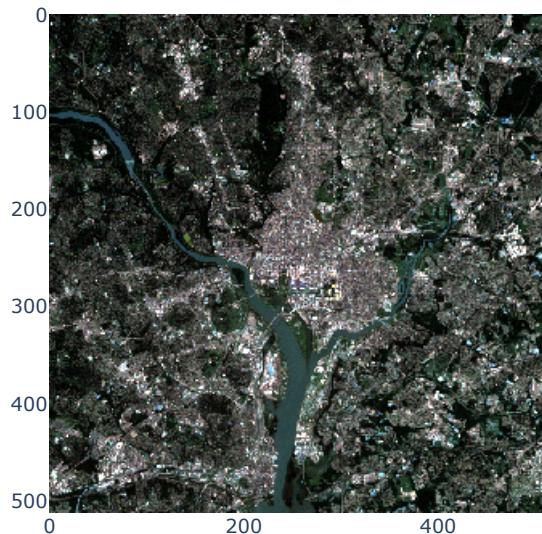
Učitati RGB sliku satellite.tif i infra-crvenu satellite_ir.tif. Formirati sliku u boju nastalu zamenom crvenog kanala sa infra-crvenom.

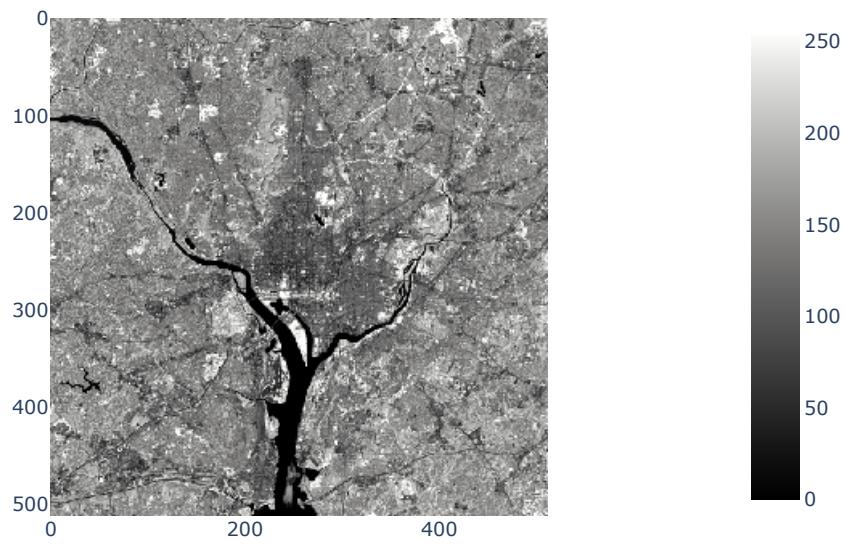
```
[50]: from skimage import io
import plotly.express as px

satellite = io.imread('satellite.tif')
satellite_ir = io.imread('satellite_ir.tif')

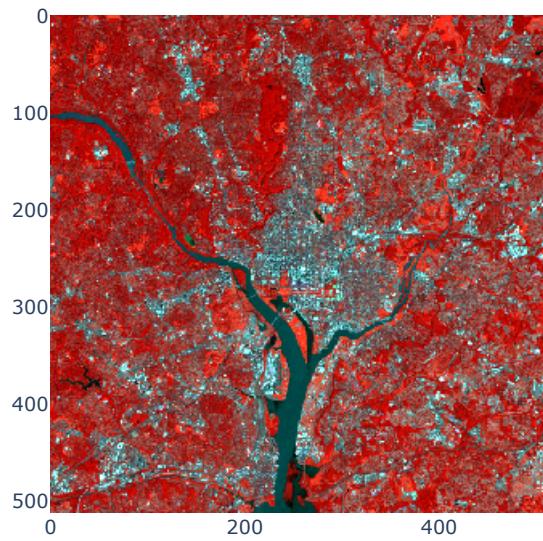
fig = px.imshow(satellite)
fig.show()

fig = px.imshow(satellite_ir, color_continuous_scale='gray')
fig.show()
```





```
[51]: satellite[:, :, 0] = satellite_ir  
fig = px.imshow(satellite)  
fig.show()
```



Na kreiranoj slici imamo jasno razlikovanje vegetacije (crveno) i ljudskih tvorevina od betona i asfalta (plavičasto).

7 Zadaci za samostalnu vežbu

Zadatak 10.1

Napisati funkciju `rgb2gray` koja implementira formu računanja luminanse slike u boji putem formule:

$$Y = 0.2125R + 0.7154G + 0.0721B$$

Funkciju testirati nad slikom `lena_color.png`.

[52] : `#TODO`

Zadatak 10.2

Odrediti predstavu `baboon.bmp` slike u CMYK (engl. *Cyan-Magenta-Yellow-Key*) sistemu boja. Implementirati funkciju `rgb2cmyk` koja vrši konverziju u jednom smeru sa datim instrukcijama polazeći od RGB komponenti koje su skalirane na $[0, 1]$ opseg:

$$K = 1 - \max(R, G, B)$$

$$C = \frac{1 - R - K}{1 - K}$$

$$M = \frac{1 - G - K}{1 - K}$$

$$Y = \frac{1 - B - K}{1 - K}$$

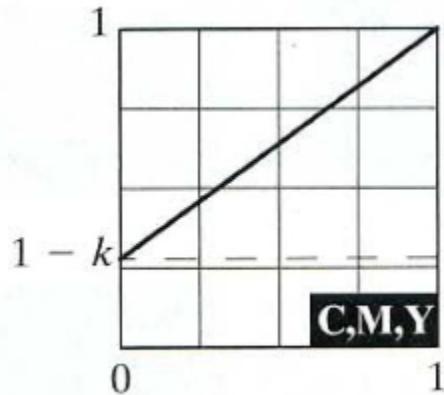
i funkciju `cmyk2rgb` za inverznu transformaciju datu instrukcijama:

$$R = 255(1 - C)(1 - K)$$

$$G = 255(1 - M)(1 - K)$$

$$B = 255(1 - Y)(1 - K)$$

Modifikovati intenzitet slike u CMYK domenu sa faktorom $k=0.7$ na način prikazan na slici ispod:



Napomena: Koristiti np. `f.info('float').eps` gde je potrebno uzeti u obzir deljenje sa 0.

[53] : #TODO

Zadatak 10.3

Sa slike flower.jpg izdvojiti deo koji odgovara laticama cveta. Boje izdvojiti upotrebom sfere poluprečnika 60 sa centrom u (240, 150, 5) u RGB prostoru. Binarna matrica koja ima vrednost 1 na mestima gde su pikseli traženih boja može se odrediti preko Euklidskog rastojanja boje piksela centra zadate sfere:

$$M = \sqrt{(C_R - r_R)^2 + (C_G - r_G)^2 + (C_B - r_B)^2} \leq D$$

gde C predstavlja centar sfere, r predstavlja vektor RGB komponenti nekog piksela, a D obeležava poluprečnik sfere.

[54] : #TODO

Zadatak 10.4

Sa slike baboon.bmp izdvojiti crvene elemente koristeći HSV domen.

[55] : #TODO

Zadatak 10.5

U slici color_mix.bmp postaviti crvenu nijansu na plavu, a plavu na njen komplement. Pozadinu postaviti na belu boju i smanjiti saturaciju zelene na njenu polovinu.

[56] : #TODO

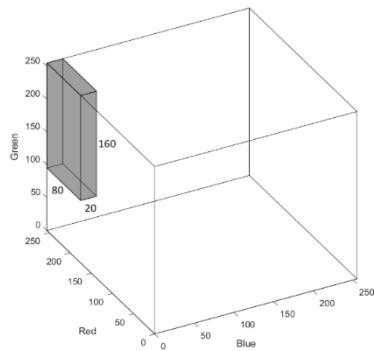
Zadatak 10.6

Na G kanal slike `peppers_matlab.png` u RGB sistemu dodati impulsni so i biber šum gustine 10% i prikazati oštećenu sliku. Šum ukloniti adekvatnim filtrom. Komentarisati kakav je uticaj ovog šuma na HSV sliku.

[57] : #TODO

Zadatak 10.7

Sa slike `flower2.jpg` izdvojiti deo koji odgovara nijansama laticama cveta određenim kvadrom sa slike 1 u RGB prostoru. Centar kvadra u RGB prostoru je $(215, 175, 10)$. Prikazati sliku sa izdvojenim bojama.



[58] : #TODO

8 Dokumentacija novih celina

- `numpy.rad2deg`
- `numpy.arccos`
- `numpy.amin`
- `numpy.amax`
- `skimage.color.rgb2hsv`
- `skimage.color.hsv2rgb`
- `matplotlib.cm.get_cmap`

Vezba 11 - Morfološke operacije

1 Pregled

U ovoj vežbi analiziraju se efekti osnovnih morfoloških operacija dilatacije i erozije nad binarnom slikom. Posmatraju se i rezultati upotrebe morfološkog otvaranja i zatvaranja kao i operacija tophat i bottomhat. Prikazana je primena morfološke analize u izdvajanju ivica objekata i popunjavanja regiona, kao i upotreba osnovnih morfoloških operacija nad slikom u sivoj skali.

2 Morfološke operacije nad slikama

Morfološke operacije u okviru slike se koriste nad 2 tipa skupa piksela, objekti (ili strukture) i strukturni elementi. Objekti se definišu kao setovi piksela u prvom planu (tzv. engl. *foreground pixels*) a strukturni elementi se mogu definisati u kontekstu piksela u prvom planu i pozadinskih piksela (tzv. engl. *background pixels*) sa opcionim elementima gde nam nije bitna vrsta piksela.

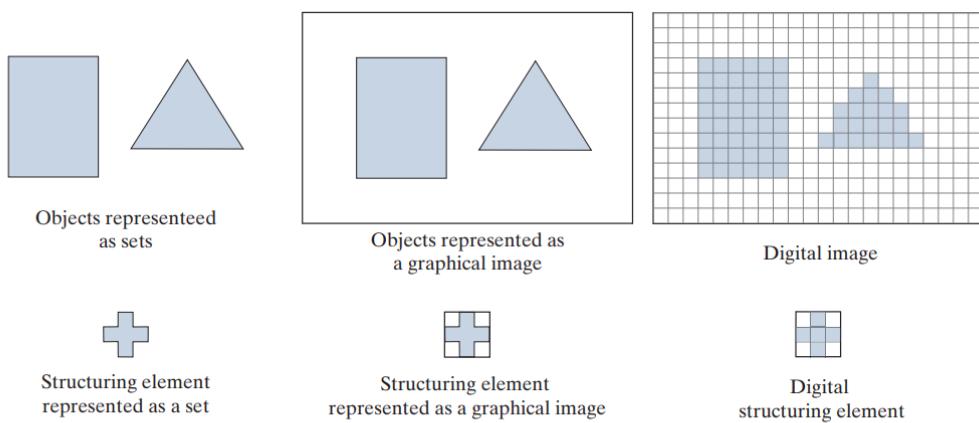


FIGURE 9.1 Top row. *Left:* Objects represented as graphical sets. *Center:* Objects embedded in a background to form a graphical image. *Right:* Object and background are digitized to form a digital image (note the grid). Second row: Example of a structuring element represented as a set, a graphical image, and finally as a digital SE.

Strukturni elementi predstavljaju 2D matrice manjih dimenzija u odnosu na slike nad kojima se primenjuju morfološke operacije.

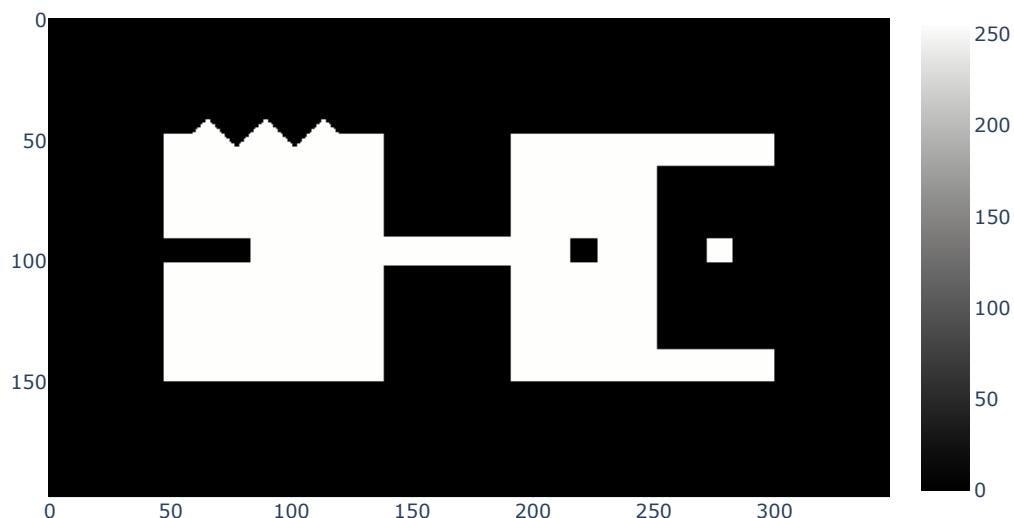
U okviru ove vežbe prikazujemo osnove morfološke obrade slike i koristimo `skimage.morphology` modul za većinu ovih obrada.

2.1 Primer 11.1

Učitati sliku `figura.tif`. Nad slikom primeniti morfološke operacije dilatacije i erozije sa strukturnim elementom kvadratnog oblika veličine 15×15 .

```
[2]: import plotly.express as px
from skimage import io

img = io.imread('figura.tif')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```



Za kreiranje strukturnog elementa se oblika kvadrata koristimo funkciju `skimage.morphology.square` koja prima parametar dužine njegove stranice.

Pored kvadratnog oblika, postoje i drugi predefinisani oblici strukturnih elemenata, ali je moguće kreirati i sopstvene oblike kao numpy 2D niz sa vrednostima 0 i 1.

```
[3]: from skimage import morphology

se = morphology.square(15)
```

```
fig = px.imshow(se, zmin=0, color_continuous_scale='gray')
fig.show()
```

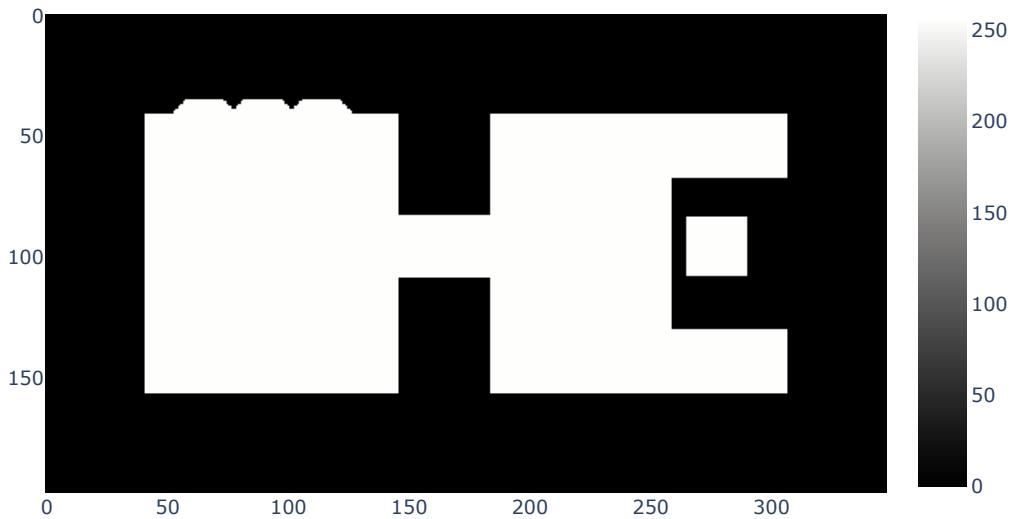


2.1.1 Dilatacija

Dilatacija je morfološka operacija kod koje se struktturni element koristi za proširenje objekata u slici. Za dilataciju se koristi funkcija `skimage.morphology.dilation`. Funkcija prima parametar slike koja se obrađuje i parametar footprint koji definiše struktturni element koji se koristi.

Znak za dilataciju se predstavlja kao \oplus

```
[4]: img_dil = morphology.dilation(img,se)
fig = px.imshow(img_dil, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```

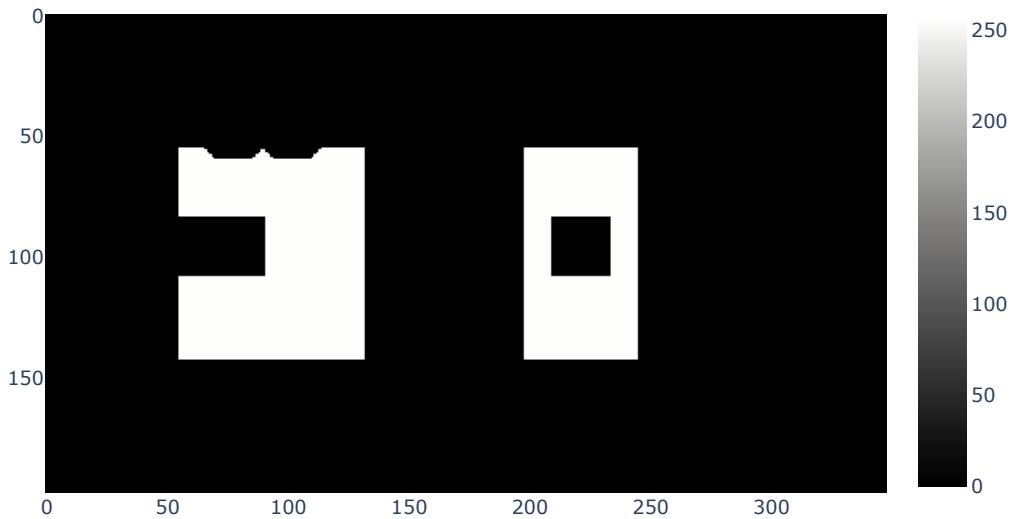


2.1.2 Erozija

Erozija je morfološka operacija koja se upotrebljava za sužavanje objekata. Erozija slike implementirana je funkcijom `skimage.morphology.erosion`.

Znak za eroziju se predstavlja kao \ominus

```
[5]: img_eroode = morphology.erosion(img,se)
fig = px.imshow(img_eroode, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```



2.2 Primer 11.2

Učitati sliku lena.png. Nad slikom primeniti morfološke operacije dilatacije i erozije sa strukturnim elementom oblika dijamanta veličine 31x31.

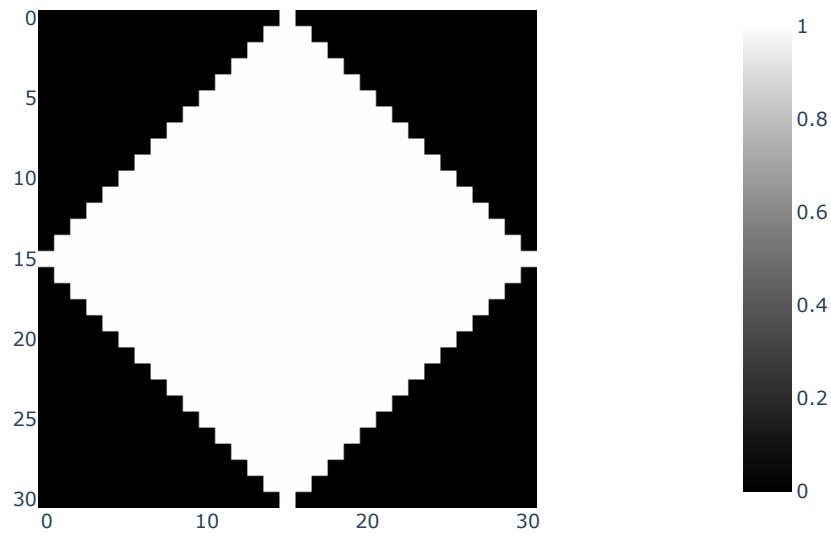
Pored informacije o lokaciji pojedinačnih piksela, morfološke operacije nad sivim slikama dodatno uključuju i informaciju o intenzitetu.

```
[6]: import plotly.express as px
from skimage import io

img = io.imread('lena.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[7]: from skimage import morphology  
  
se = morphology.diamond(15)  
fig = px.imshow(se, zmin=0, color_continuous_scale='gray')  
fig.show()
```



2.2.1 Dilatacija

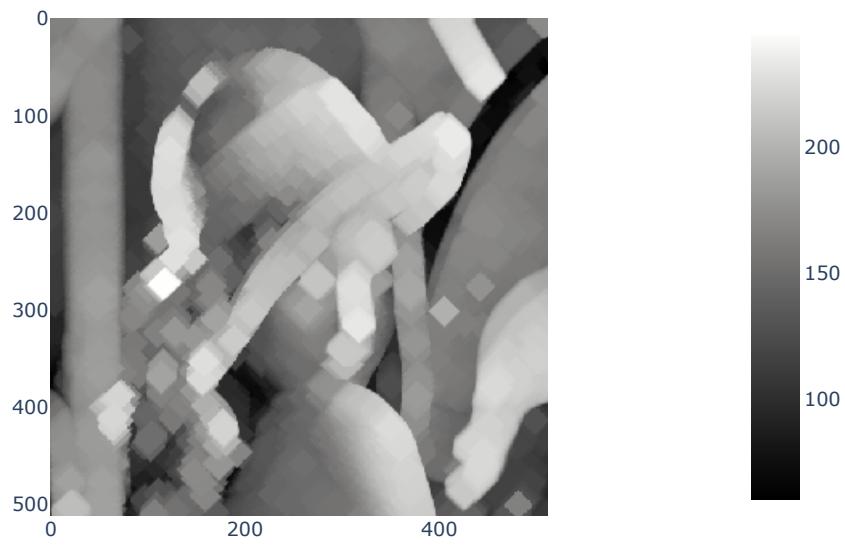
Operacija dilatacije predstavlja operaciju max filtra nad zbirom intenziteta slike i strukturnog elementa.

$$(f \oplus b)(x, y) = \max_{(s,t) \in \hat{b}} \{f(x - s, y - t) + \hat{b}(s, t)\}$$

gde je f slika nad kojom se primjenjuje morfološka operacija a b je strukturni element. \hat{b} predstavlja refleksiju strukturnog elementa b .

Obično se vrednosti strukturnog elementa postavljaju na 0 gde jeste neki piksel od interesa a $-\infty$ inače. Ovim se proces dilatacije svodi na određivanje rezultata max filtra za neki dati region. U okviru korišćene funkcije dilatacije, strukturni element se i dalje jednostavno definiše sa vrednostima 1 kao pikseli od interesa i 0 inače i podrazumeva da se traži max funkcija nad zadatim regionom.

```
[8]: img_dil = morphology.dilation(img,se)
fig = px.imshow(img_dil, color_continuous_scale='gray')
fig.show()
```



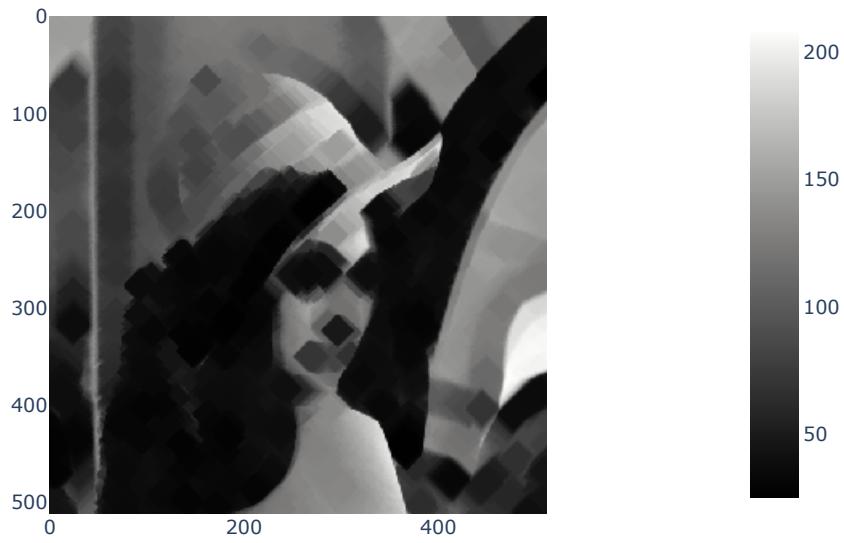
2.2.2 Erozija

Analogno dilataciji u okviru sive slike, operacija erozije predstavlja operaciju min filtra nad razlikom intenziteta slike i strukturnog elementa.

$$(f \ominus b)(x, y) = \min_{(s,t) \in b} \{f(x - s, y - t) - b(s, t)\}$$

gde je f slika nad kojom se primenjuje morfološka operacija a b je strukturni element.

```
[9]: img_erosion = morphology.erosion(img,se)
fig = px.imshow(img_erosion, color_continuous_scale='gray')
fig.show()
```

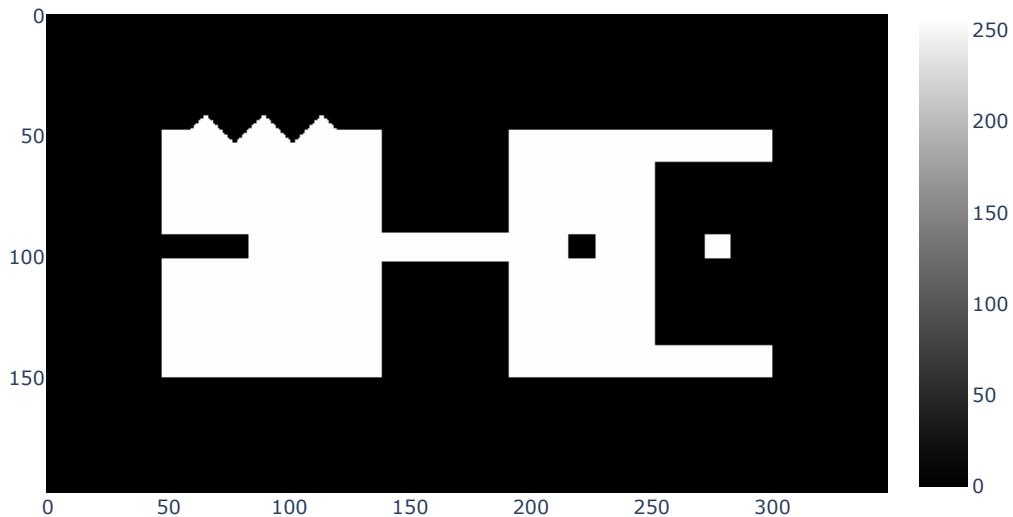


2.3 Primer 11.3

Učitati sliku `figura.tif`. Nad slikom primeniti morfološke operacije otvaranja i zatvaranja sa strukturnim elementom kvadratnog oblika 15×15 .

```
[10]: import plotly.express as px
from skimage import io, morphology

img = io.imread('figura.tif')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```



```
[11]: se = morphology.square(15)
```

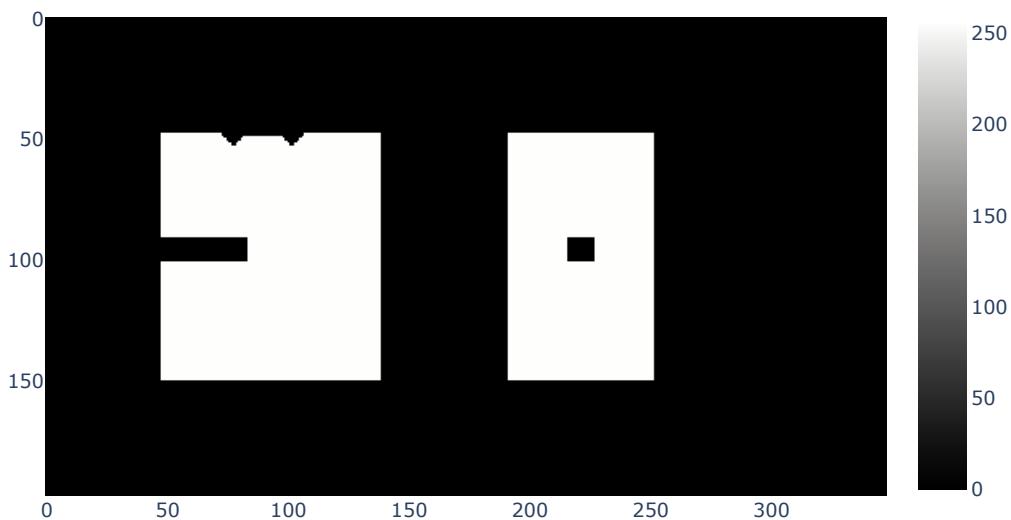
2.3.1 Otvaranje

Morfološko otvaranje se sastoji od erozije i naknadne dilatacije slike upotrebom istog strukturnog elementa.

$$A \circ B = (A \ominus B) \oplus B$$

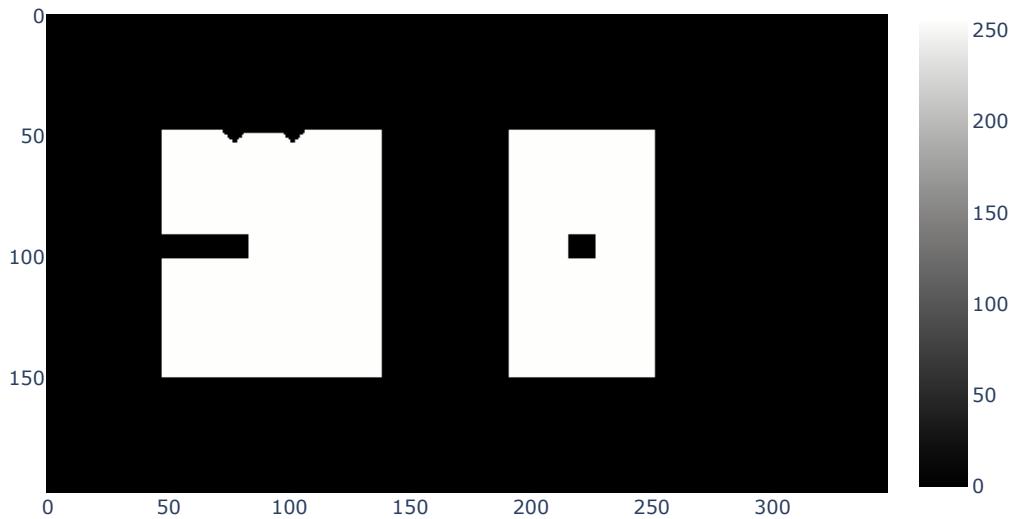
gde je A skup piksela u prvom planu slike I a B strukturni element.

```
[12]: img_open1 = morphology.dilation(morphology.erosion(img,se),se)
fig = px.imshow(img_open1, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```



Morfološko otvaranje se upotrebljava za ublažavanje konture objekta, eliminaciju tankih veza između delova objekta i eliminaciju sitnih objekata. Postoji ugrađena funkcija za morfološko otvaranje `skimage.morphology.opening` koja daje identičan rezultat prikazan iznad.

```
[13]: img_open2 = morphology.opening(img,se)
fig = px.imshow(img_open2, color_continuous_scale='gray')
fig.show()
```



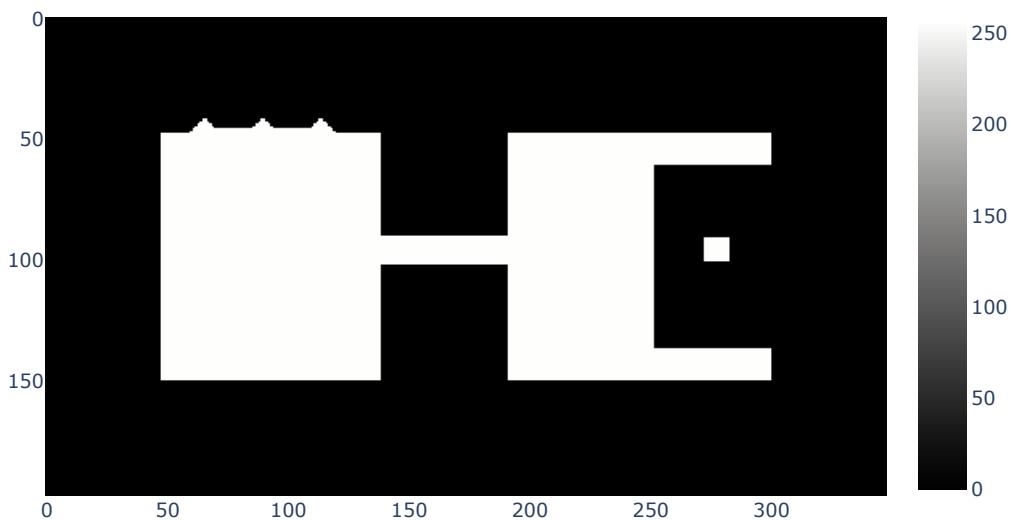
2.3.2 Zatvaranje

Morfološko zatvaranje se sastoji od dilatacije i naknadne erozije binarne slike upotrebom istog strukturnog elementa.

$$A \bullet B = (A \oplus B) \ominus B$$

gde je A skup piksela u prvom planu slike I a B strukturni element.

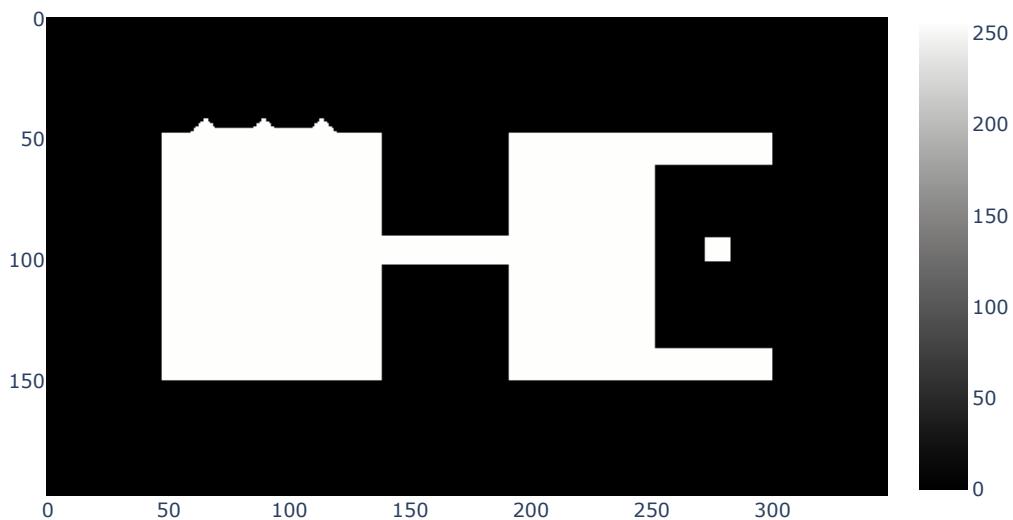
```
[14]: img_close1 = morphology.erosion(morphology.dilation(img,se),se)
fig = px.imshow(img_close1, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```



Morfološko zatvaranje se koristi za uklanjanje procepa u konturi objekata, spajanje uskih prekida između objekata i popunjavanje sitnijih praznina u okviru jednog objekta.

Postoji ugrađena funkcija za morfološko zatvaranje `skimage.morphology.closing` koja daje identičan rezultat prikazan iznad.

```
[15]: img_close2 = morphology.closing(img,se)
fig = px.imshow(img_close2, color_continuous_scale='gray')
fig.show()
```

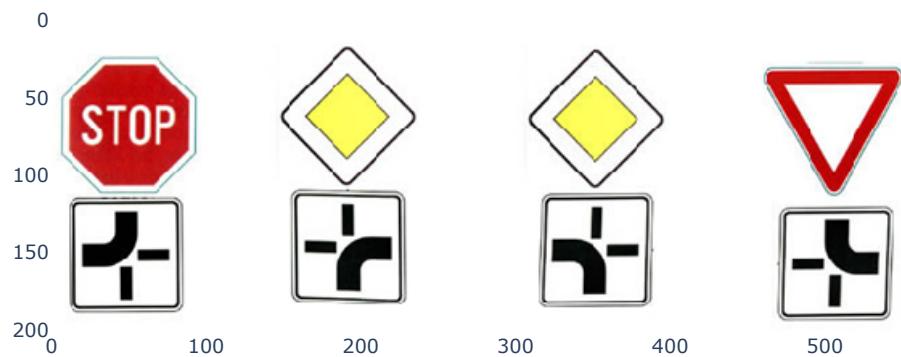


2.4 Primer 11.4

Učitati sliku `znakovi.jpg` i konvertovati je u sliku u sivoj skali koristeći funkciju `skimage.color.rgb2gray`. Pronaći masku za znakove za put sa prvenstvom prolaza (romb sa žutom unutrašnjosti) bez korišćenja kolor informacija.

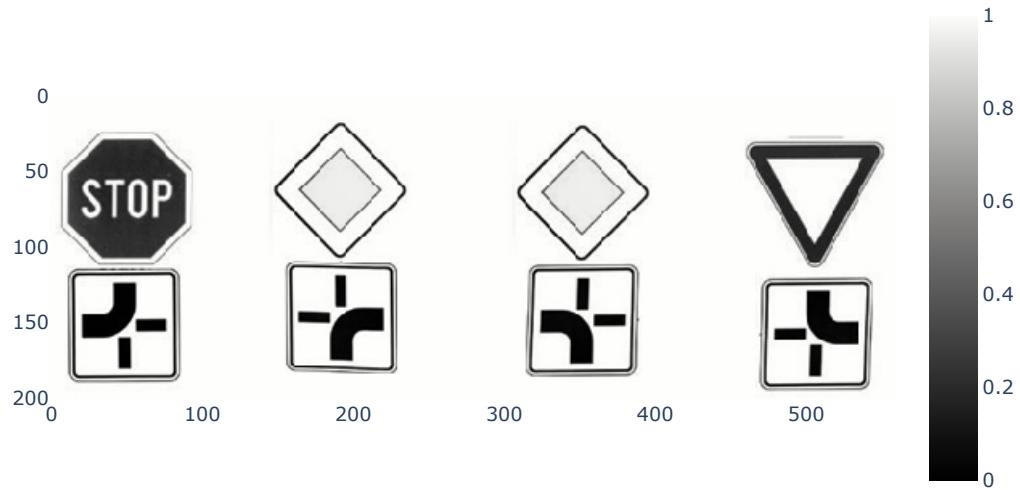
```
[16]: import plotly.express as px
from skimage import io, morphology

img = io.imread('znakovi.jpg')
fig = px.imshow(img)
fig.show()
```



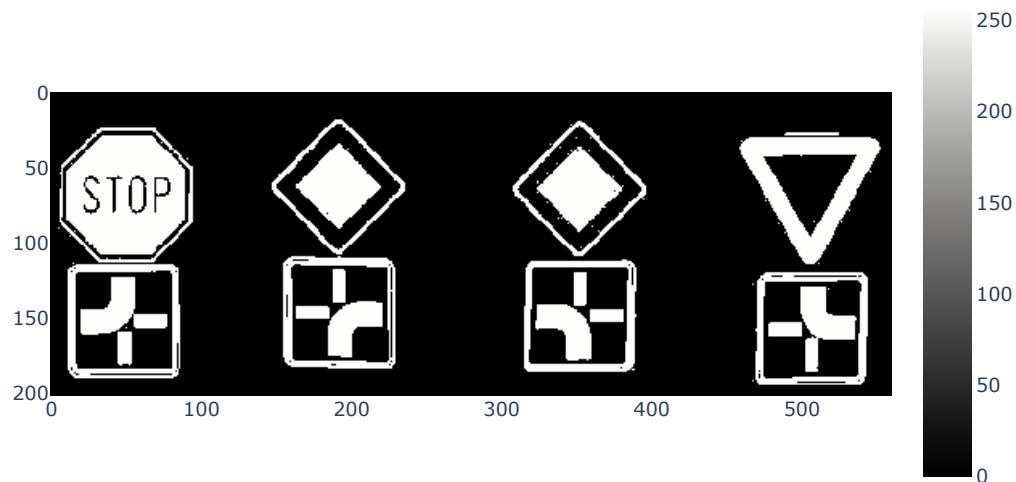
```
[17]: from skimage import color

img_gray = color.rgb2gray(img)
fig = px.imshow(img_gray, color_continuous_scale='gray')
fig.show()
```



Nad sivom slikom je potrebno upotrebiti prag koji će izdvojiti znakove od pozadine.

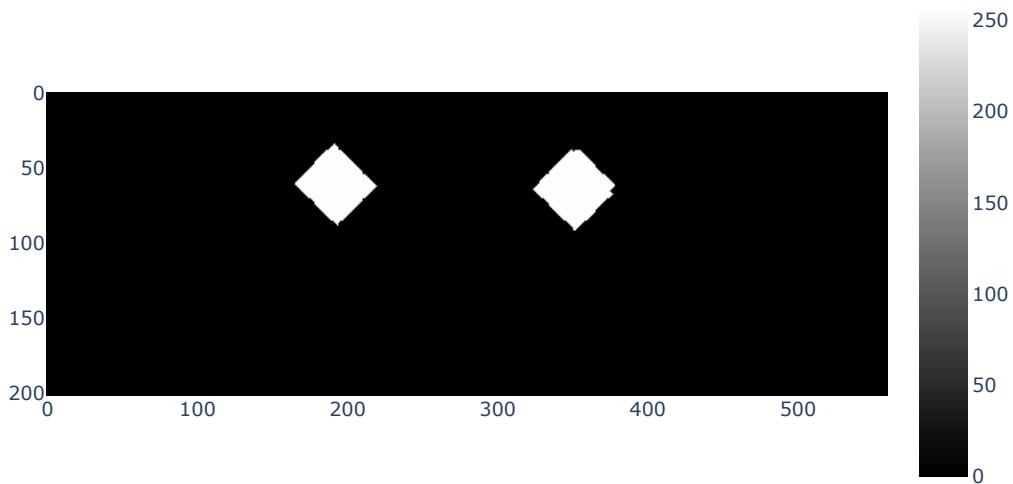
```
[18]: A = img_gray < 0.96
fig = px.imshow(A, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd': ['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```



Izdvajanje struktura od interesa možemo uraditi korišćenjem morfoloških operacija sa strukturalnim elementom sličnim traženom obliku.

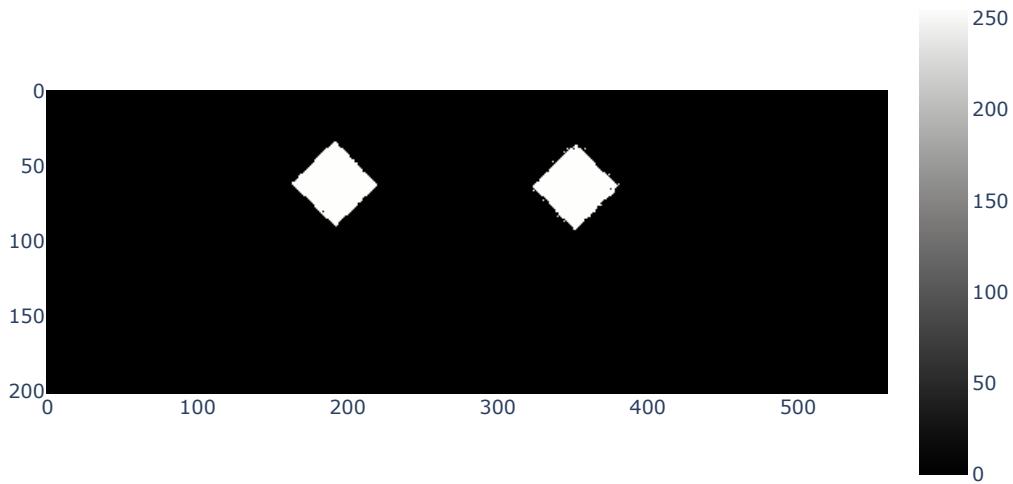
```
[19]: B = morphology.diamond(19)

M1 = morphology.opening(A,B)
fig = px.imshow(M1, color_continuous_scale='gray')
fig.show()
```



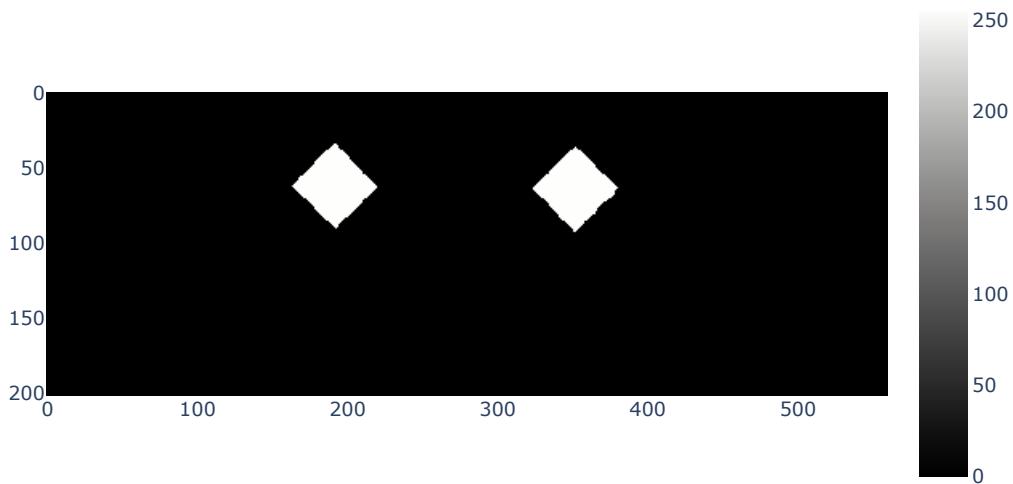
Dodatnu korekciju nad rezultatom možemo postići tako što iskoristimo informaciju o pravom obliku željene strukture. Dilatacijom proširujemo trenutnu masku i presekom ograničavamo na originalni oblik.

```
[20]: M2 = morphology.dilation(M1,morphology.diamond(3)) & A
fig = px.imshow(M2, color_continuous_scale='gray')
fig.show()
```



Finalnu korekciju možemo postići upotrebom morfološkog otvaranja i zatvaranja u cilju uklanjanja sitnih struktura i popunjavanja sitnih praznina.

```
[21]: M3 = morphology.opening(M2,morphology.diamond(1))
M4 = morphology.closing(M3,morphology.diamond(1))
fig = px.imshow(M4, color_continuous_scale='gray')
fig.show()
```



2.5 Primer 11.5

Nad slikom lena.png primeniti morfološku operaciju tophat i bottomhat koristeći strukturni element oblika diska dimenzije 7x7.

```
[22]: import plotly.express as px
from skimage import io, morphology

img = io.imread('lena.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[23]: se = morphology.disk(3)
print(se)
```

```
[[0 0 0 1 0 0 0]
 [0 1 1 1 1 1 0]
 [0 1 1 1 1 1 0]
 [1 1 1 1 1 1 1]
 [0 1 1 1 1 1 0]
 [0 1 1 1 1 1 0]
 [0 0 0 1 0 0 0]]
```

2.5.1 Top-hat

Top-hat transformacija (ili *white tophat*) je definisana kao razlika između polazne slike i slike nastale otvaranjem sa nekim strukturnim elementom.

$$T_{hat}(f) = f - (f \circ b)$$

gde je f polazna slika a b strukturni element. Rezultat ove operacije vraća sve elemente koji bi bili izgubljeni prilikom izvršenja operacije morfološkog otvaranja. Služi za pronalaženje svetlijih objekata u tamnijoj pozadini.

Funkcija koja implementira direktno ovu transformaciju je [skimage.morphology.white_tophat](#).

```
[24]: img_top = img - morphology.opening(img,se)
#img_top = morphology.white_tophat(img,se)

fig = px.imshow(img_top, color_continuous_scale='gray')
fig.show()
```



2.5.2 Bottom-hat

Bottom-hat transformacija (ili *black tophat*) je definisana kao razlika između polazne slike i slike nastale otvaranjem sa nekim strukturnim elementom.

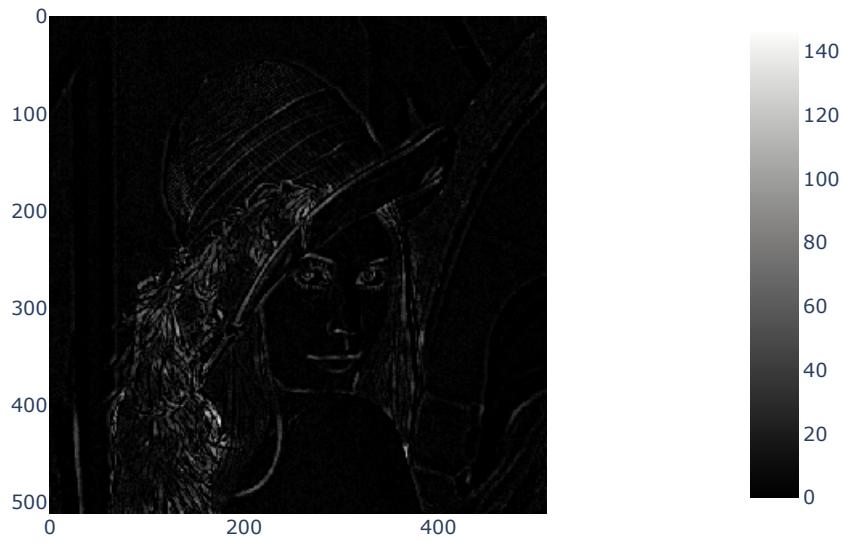
$$B_{hat}(f) = (f \bullet b) - f$$

gde je f polazna slika a b strukturni element. Rezultat ove operacije vraća sve elemente koji bi bili popunjeni prilikom izvršenja operacije morfološkog zatvaranja. Služi za pronalaženje tamnijih objekata u svetlijoj pozadini.

Funkcija koja implementira direktno ovu transformaciju je `skimage.morphology.black_tophat`.

```
[25]: img_bot = morphology.closing(img,se) - img
#img_bot = morphology.black_tophat(img,se)

fig = px.imshow(img_bot, color_continuous_scale='gray')
fig.show()
```



2.5.3 Iстичање детаља

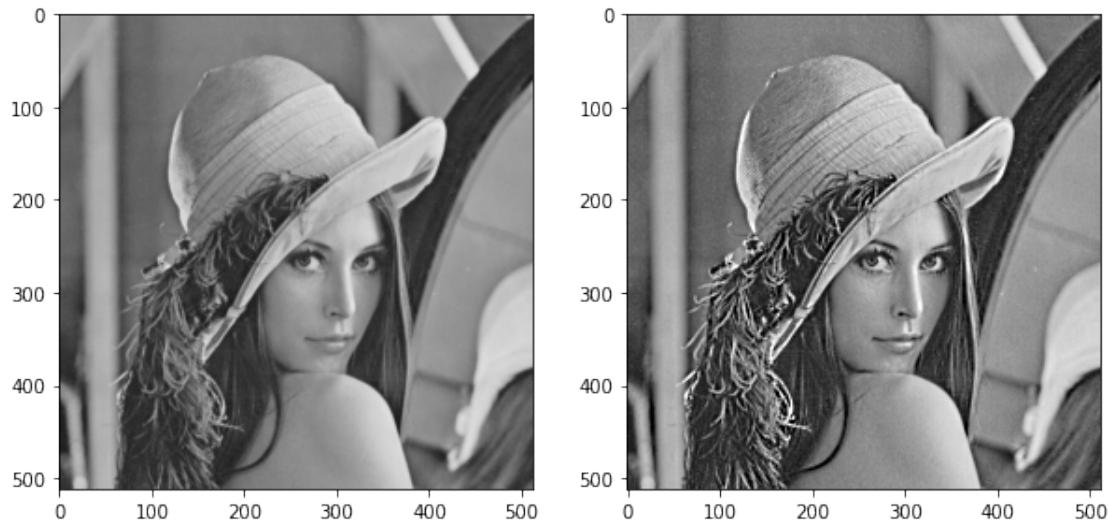
Kombinovanjem ove dve slike možemo izvršiti naglašavanje detalja u okviru slike.

$$f_{izostrena} = f + T_{hat}(f) - B_{hat}(f)$$

```
[26]: import numpy as np
import matplotlib.pyplot as plt

img_detail = img.astype('float') + img_top - img_bot

plt.figure(figsize = (10,10))
plt.subplot(121)
plt.imshow(img, vmin=0, vmax=255, cmap='gray')
plt.subplot(122)
plt.imshow(img_detail, vmin=0, vmax=255, cmap='gray')
plt.show()
```



2.6 Primer 11.6

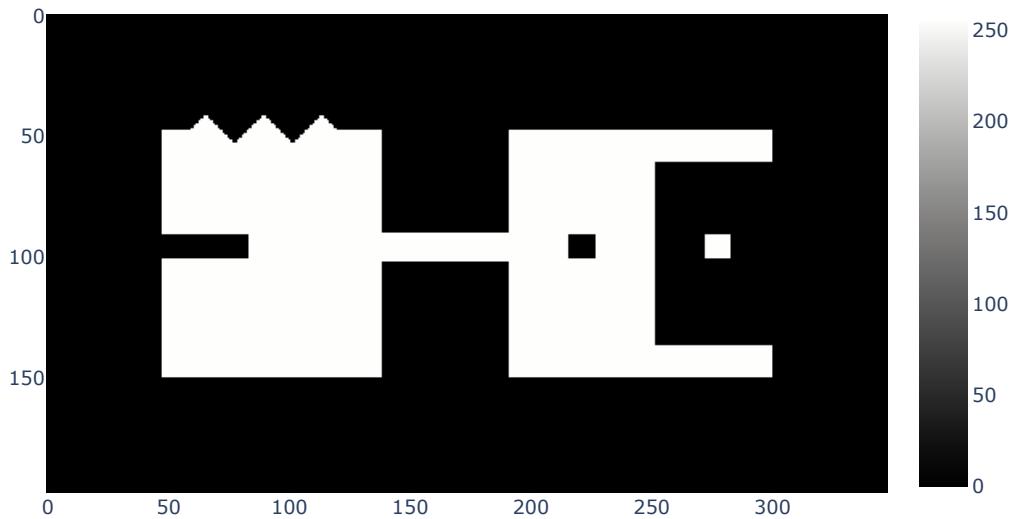
Nad slikom `figura.tif` primeniti morfološki algoritam za izdvajanja ivica definisan kao:

$$\beta(A) = A - (A \ominus B)$$

gde je B strukturni element oblika diska dimenzije 3x3.

```
[27]: import plotly.express as px
from skimage import io, morphology

A = io.imread('figura.tif')
fig = px.imshow(A, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```

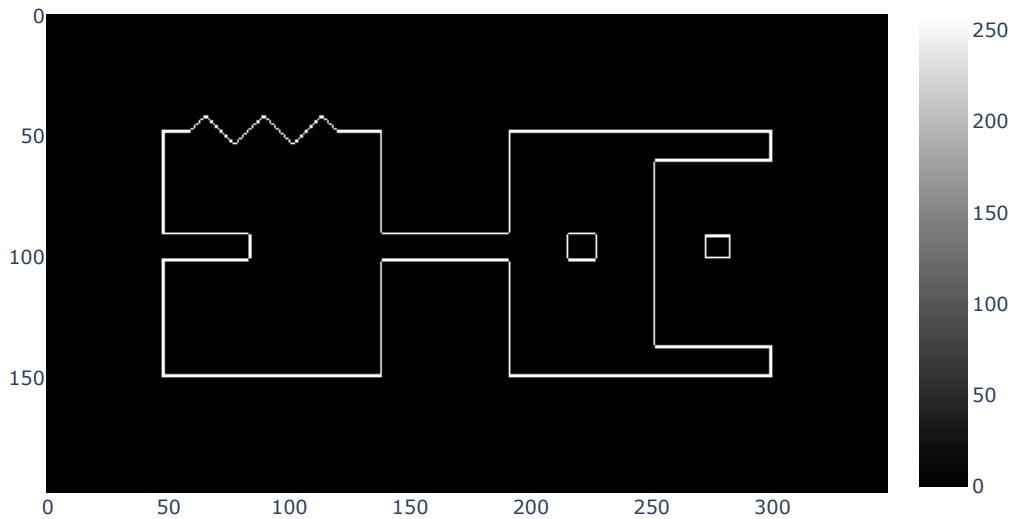


Pošto je slika binarnog tipa, potrebno je ili prevesti u float ili koristiti analognu operaciju oduzimanja za binarne vrednost - xor operaciju koristeći operator ^.

```
[28]: B = morphology.disk(1)
print(B)

img_edge = A ^ morphology.erosion(A,B)
fig = px.imshow(img_edge, color_continuous_scale='gray')
fig.show()
```

```
[[0 1 0]
 [1 1 1]
 [0 1 0]]
```



2.7 Primer 11.7

Nad rezultujućom slikom iz primera 10.6 primeniti morfološki algoritam za popunjavanje regiona (engl. *region filling*) definisan kao:

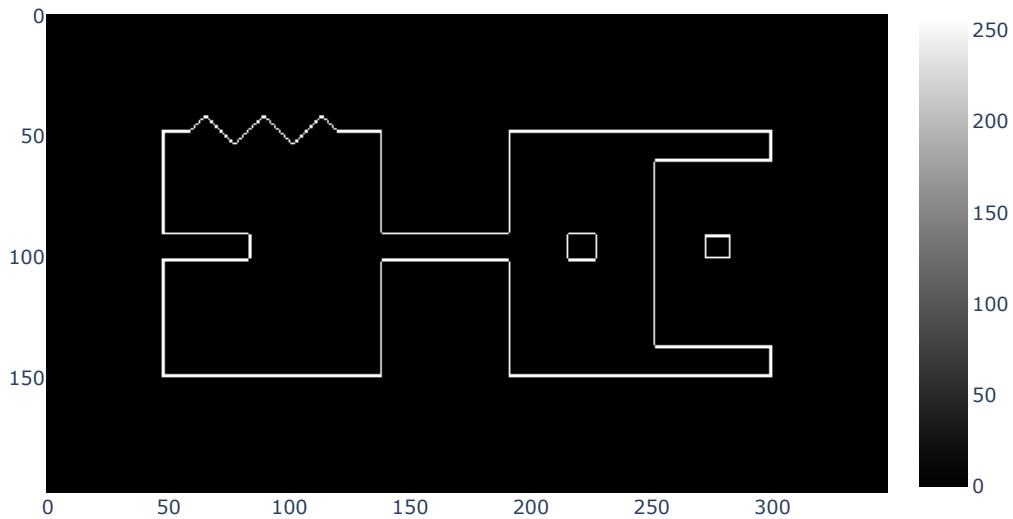
$$X_k = (X_{k-1} \oplus B) \cap A^c, \quad k = 1, 2, 3\dots$$

gde je B strukturni element oblika diska dimenzije 3x3. A^c je komplement od A .

Koristiti početnu tačku (66, 66).

```
[29]: import plotly.express as px
from skimage import io, morphology

fig = px.imshow(img_edge, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd':['drawline',
                                         'drawopenpath',
                                         'drawrect',
                                         'eraseshape']})
```



```
[30]: import numpy as np

y = 66
x = 66

B = morphology.disk(1)

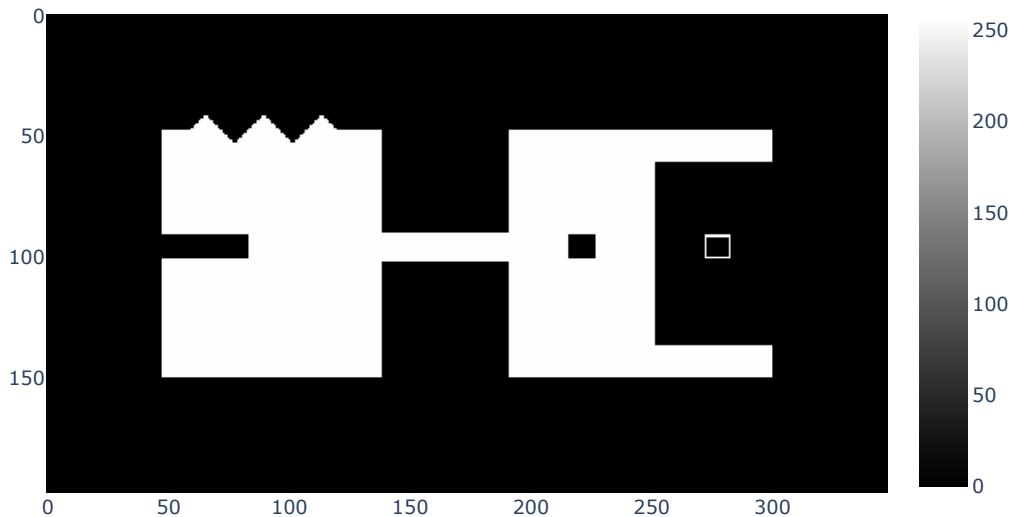
X_k0 = np.full(img_edge.shape, False)
X_k0[y,x] = 1

Ac = ~img_edge

flag = True
while flag:
    X_k = morphology.dilation(X_k0,B) & Ac

    if (X_k == X_k0).all():
        flag = False
    else:
        X_k0 = X_k

fig = px.imshow(img_edge+X_k, color_continuous_scale='gray')
fig.show()
```



3 Zadaci za samostalnu vežbu

Zadatak 11.1

Koristeći morfološke operacije ukloniti galaksiju prisutnu u sredini slike `galaxy.png`.

[31] : #TODO

Zadatak 11.2

Nad slikom `figura.tif` primeniti morfološki algoritam za izdvajanje povezanih komponenti definisan kao:

$$X_k = (X_{k-1} \oplus B) \cap A, \quad k = 1, 2, 3\dots$$

gde je B strukturni element oblika kvadrata dimenzije 3×3 a A polazna binarna slika.

Koristiti početnu tačku $(66, 66)$.

[32] : #TODO

Zadatak 11.3

Sa slike `elementi.bmp` izdvojiti crvene viljuškaste krajeve.

[33] : `#TODO`

Zadatak 11.4

Nad slikom `figura.tif` primeniti morfološke operacije `skeletonize`, `thin`, `convex hull` definisane u okviru `skimage.morphology` modula. Uporediti i komentarisati rezultate.

[34] : `#TODO`

Zadatak 11.5

Učitati sliku `simboli4x4B.png`. U okviru slike potrebno je podesiti boju svih srca na crvenu boju i pozadinu postaviti na cijan. Ostali elementi trebaju biti neizmenjeni.

[35] : `#TODO`

4 Dokumentacija novih celina

- `skimage.morphology`
- `generating structuring elements`
- `skimage.morphology.dilation`
- `skimage.morphology.erosion`
- `skimage.morphology.opening`
- `skimage.morphology.closing`
- `skimage.color.rgb2gray`
- `skimage.morphology.white_tophat`
- `skimage.morphology.black_tophat`

Vezba 12 - Segmentacija

1 Pregled

Segmentacija slike je jedan od najzahtevnijih zadataka u obradi slike, samim tim što njena tačnost utiče na sve naknadne analize slike koje se sprovode nad segmentiranim objektima. U segmentaciji se prevashodno koriste dve osnovne karakteristike intenziteta piksela: diskontinuitet i sličnost. U ovoj vežbi prikazuje se primena različitih metoda za detekciju ivica i linija na slici. Posmatran je izgled odziva gradijentnih operatora i pokazana je upotreba praga da bi se korišćenjem slike intenziteta gradijenata izdvojile ivice. Primena drugog izvoda za detekciju ivica prikazana je preko filtra Laplasijan Gausijana. Objasnjena je Hafova (engl. *Hough*) transformacija i prikazana je njena primena za detekciju pravih linija. Pored ovoga prikazane su i tehnike izdvajanja objekata u slici koristeći lokalna i globalna svojsta.

1.1 Primer 12.1

Prikazati vrednost parcijalnih izvoda slike lena.png određenih upotrebom Sobelovih gradijentnih operatora i konstruisati gradijentnu sliku G . Sobelovi operatori su definisani maskama:

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{i} \quad S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Odrediti binarnu sliku koja odgovara jakim ivicama upotrebom praga T definisanog sledećim izrazom:

$$T = 2 \sqrt{\frac{1}{N} \frac{1}{M} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} G^2(x, y)}$$

gde su M, N dimenzije slike. Nad binarnom slikom upotrebiti morfološko stanjivanje da bi se dobila slika ivica.

```
[2]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('lena.png')
```

```
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



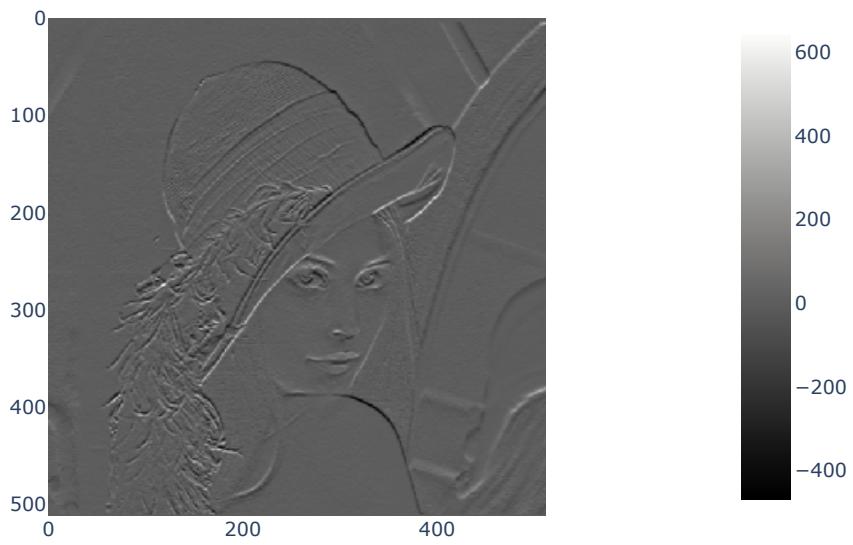
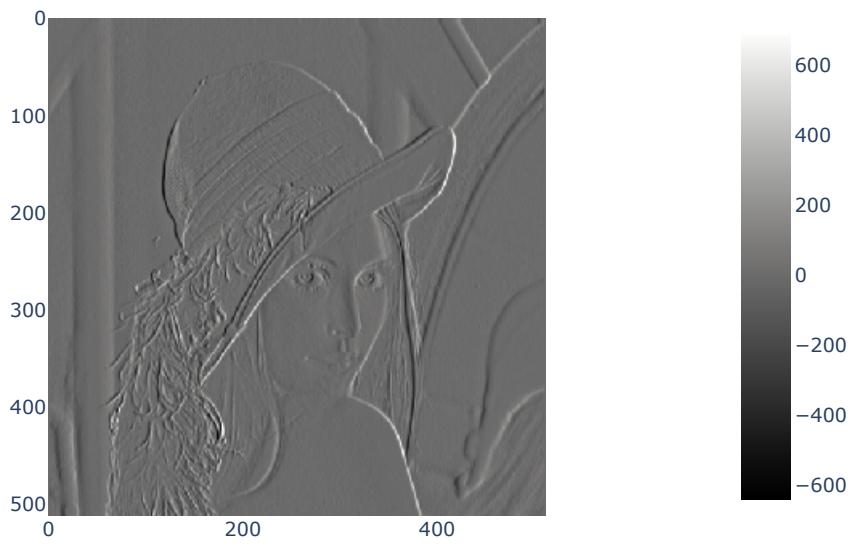
```
[3]: from scipy import ndimage

Sy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
Sx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

img = img.astype('float')

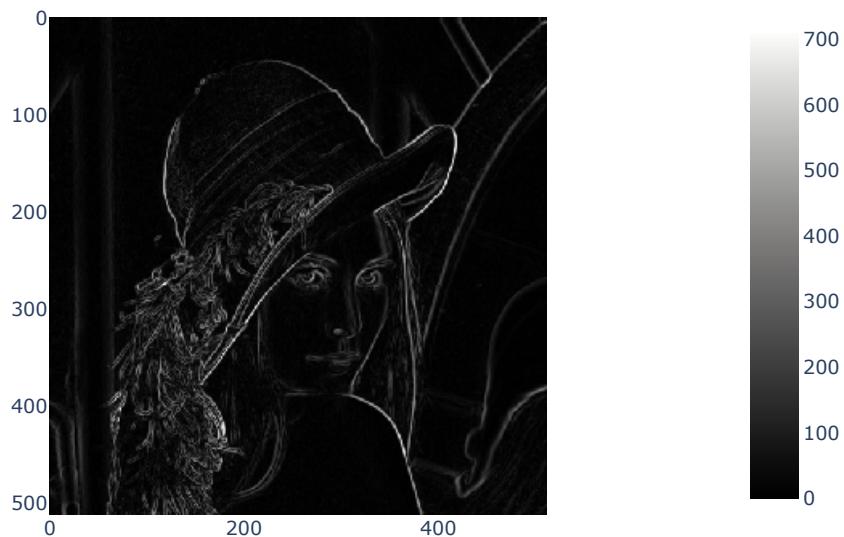
G1 = ndimage.convolve(img, Sx, mode='mirror')
fig = px.imshow(G1, color_continuous_scale='gray')
fig.show()

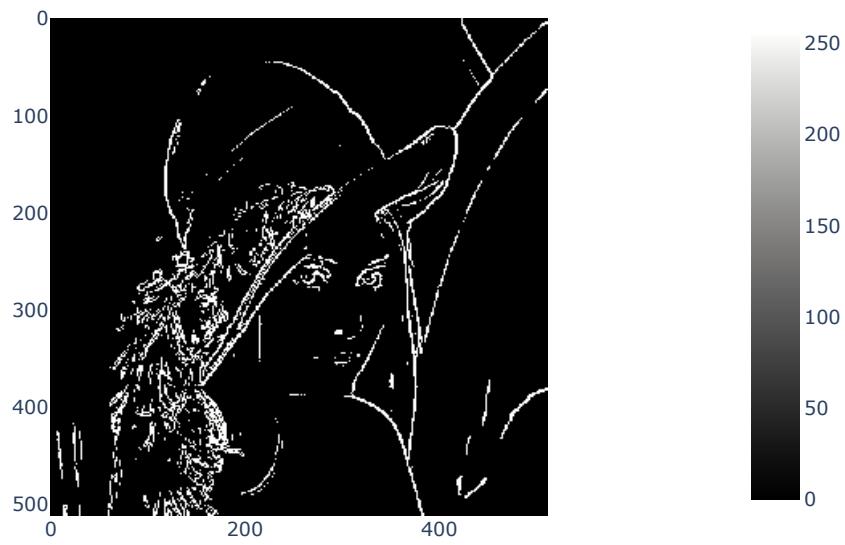
G2 = ndimage.convolve(img, Sy, mode='mirror')
fig = px.imshow(G2, color_continuous_scale='gray')
fig.show()
```



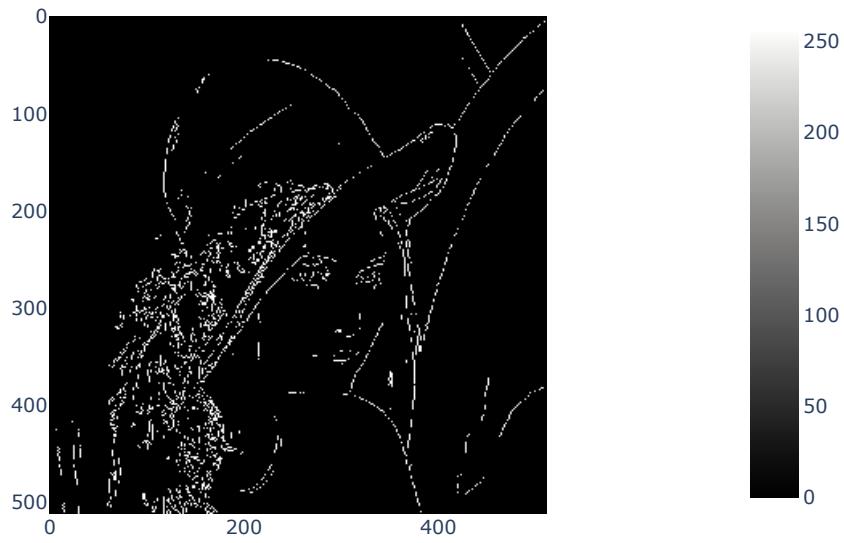
```
[4]: G = np.sqrt(G1**2+G2**2)
fig = px.imshow(G, color_continuous_scale='gray')
fig.show()

# binarizacija pragovanjem
T = 2*np.sqrt((G**2).mean())
G_edge = G > T
fig = px.imshow(G_edge, color_continuous_scale='gray')
fig.show()
```





```
[5]: from skimage import morphology  
  
G_thin = morphology.thin(G_edge)  
fig = px.imshow(G_thin, color_continuous_scale='gray')  
fig.show()
```



1.2 Primer 12.2

Laplasijan kao drugi izvod je izuzetno osetljiv na šum te se obično kombinuje sa Gausovim filtrom za ublažavanje slike, gde efekat ublažavanje raste sa porastom parametra standardne devijacije σ . Potrebno je implementirati funkciju `laplacianOfGaussian` koja kao izlaz vraća filter negativnog Laplasijana Gausijana (LoG) definisan sledećim izrazom:

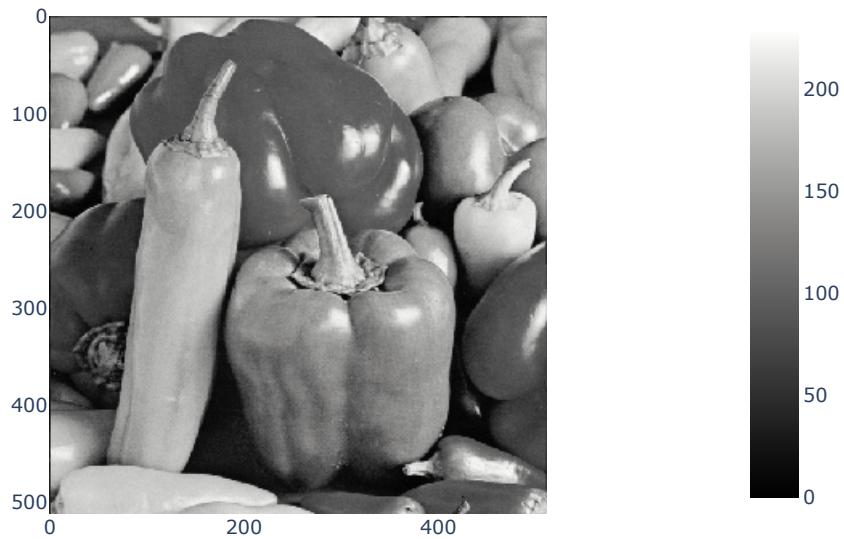
$$LoG(x, y) = - \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

gde je σ parametar filtra. Izlazni filter treba da bude u obliku kvadrata čija je stranica najmanje 6σ .

Primeniti funkciju sa parametrom $\sigma = 4$ nad slikom `peppers.png` i odrediti sliku ivica kao konturu binarne maske gde je odziv filtra veći od 0.

```
[6]: import plotly.express as px
from skimage import io

img = io.imread('peppers.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[7]: import numpy as np

def laplacianOfGaussian(sigma):
    w = np.ceil(6*sigma)

    if w % 2 == 0:
        w += 1

    y = np.arange(w).reshape(-1, 1) - np.floor(w/2)
    x = np.arange(w) - np.floor(w/2)

    LoG = -((x**2+y**2 - 2*sigma**2)/sigma**4)*np.exp(-(x**2+y**2)/(2*sigma**2))
    return LoG
```

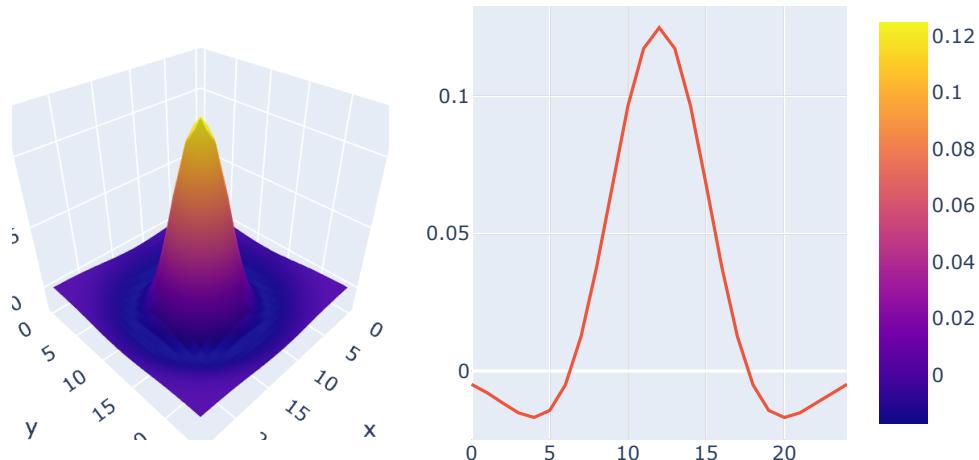
```
[8]: import plotly.graph_objects as go
from plotly.subplots import make_subplots

LoG = laplacianOfGaussian(4)

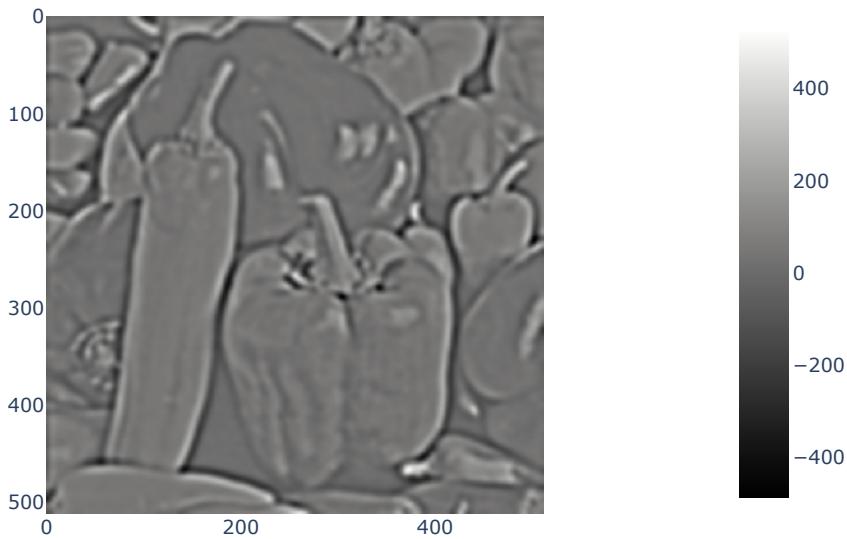
fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'surface'}, {'type': 'scatter'}]]) # mora se naznaciti koja vrsta plota ako se razlikuju

# 3D prikaz
fig.add_trace(go.Surface(z=LoG), row=1, col=1)
```

```
# poprecni presek  
fig.add_trace(go.Scatter(y=LoG[LoG.shape[0]//2], mode='lines'), row=1, col=2)  
  
fig.show()
```



```
[9]: from scipy import ndimage  
  
G = ndimage.convolve(img.astype('float'), LoG, mode='mirror')  
fig = px.imshow(G, color_continuous_scale='gray')  
fig.show()
```



```
[10]: img_bin = G > 0
fig = px.imshow(img_bin, color_continuous_scale='gray')
fig.show()
```



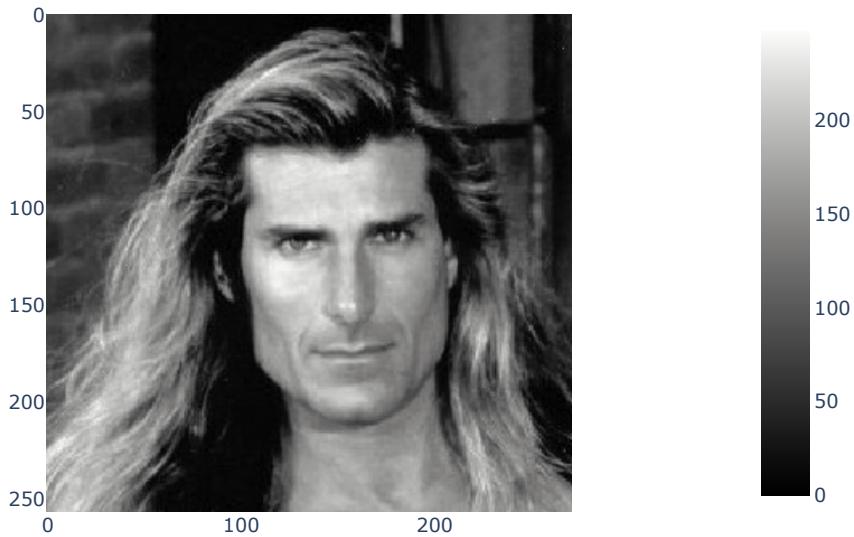
```
[11]: from skimage import morphology  
  
B = morphology.disk(1)  
  
img_edge = img_bin ^ morphology.erosion(img_bin,B)  
fig = px.imshow(img_edge, color_continuous_scale='gray')  
fig.show()
```



1.3 Primer 12.3

Na slici fabio.tif odrediti ivice upotrebom Kenijevog (engl. *Canny*) detektora ivica.

```
[12]: import plotly.express as px  
from skimage import io  
import numpy as np  
  
img = io.imread('fabio.tif')  
fig = px.imshow(img, color_continuous_scale='gray')  
fig.show()
```



Kenijev detektor ivica je jedan od najčešće korišćenih algoritama za detekciju ivica na slikama. Algoritam se sastoji od nekoliko koraka:

1. potiskivanje šuma Gausovim filtrom
2. određivanje gradijenta slike
3. potisnuti vrednosti gradijenta koje nisu maksimalne (engl. *nonmaxima suppression*)
4. primena dvostrukog praga u cilju utvrđivanja jakih i slabih kandidata za ivične tačke
5. odbacivanje slabih ivičnih tačaka koje nisu povezane sa jakim ivičnim tačkama

Funkcija `skimage.feature.canny` implementira Kenijev algoritam i određuje sliku ivica slike sa podesivim parametrima 2 praga i parametrom standardne devijacije σ za Gausovo filtriranje.

```
[13]: from skimage import feature

edges = feature.canny(img)
fig = px.imshow(edges, color_continuous_scale='gray')
fig.show()
```



```
[14]: edges = feature.canny(img, low_threshold = 100, high_threshold=150)
fig = px.imshow(edges, color_continuous_scale='gray')
fig.show()
```



```
[15]: edges = feature.canny(img, sigma=2)
fig = px.imshow(edges, color_continuous_scale='gray')
fig.show()
```



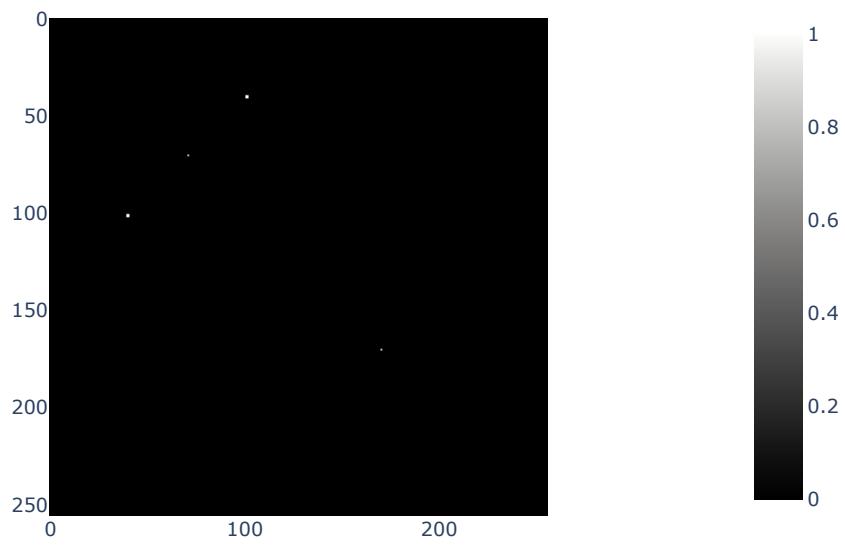
1.4 Primer 12.4

Napisati funkciju houghTransform koja implementira Hafovu (engl. *Hough*) transformaciju. Ulazni argument funkcije je binarna slika ivica.

Funkciju primeniti nad slikom four_dots.tif. Prikazati nad originalnom slikom najdominantniju liniju označenu crvenom bojom.

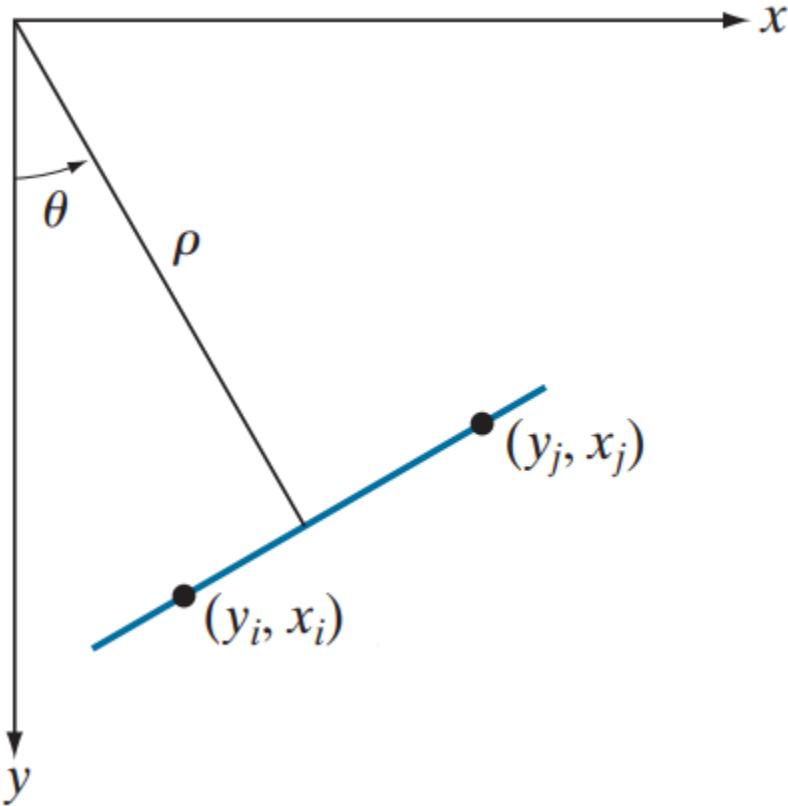
```
[16]: import plotly.express as px
from skimage import io

img_edge = io.imread('four_dots.tif')
fig = px.imshow(img_edge, color_continuous_scale='gray')
fig.show(config={'modeBarButtonsToAdd': ['drawline',
                                         'eraseshape']})
```



Jednačina prave u polarnom obliku za sistem koordiata (y, x) odnosno (*vrsta, kolona*) je:

$$y \cos \theta + x \sin \theta = \rho$$



U okviru Hafove transformacije, za svaki definisani ivični piksel potrebno je generisati sve moguće prave koje prolaze kroz njega. θ vrednost se postavlja na seriju mogućih veličina u opsegu od -90° do $+90^\circ$ i za svaku vrednost se generiše odgovarajući ρ . Za svaku kreiranu pravu sa parametrom (ρ, θ) upisuje se 1 u odgovarajuće polje akumulatorske matrice H .

Napomena: * `numpy.cos` i `numpy.sin` funkcije računaju za vrednosti radijana, stoga je stepene potrebno konvertovati. * Za granice mogućih vrednosti za ρ smatramo da je od $-D$ do D , gde je D najveća moguća Euklidska razdaljina između pozicija piksela.

```
[17]: import numpy as np

def houghTransform(img_edge):
    rho_max = round(np.sqrt(img_edge.shape[0]**2+img_edge.shape[1]**2))
    theta_max = 90

    rho = np.arange(-rho_max, rho_max+1)
    theta = np.arange(-theta_max, theta_max, 0.5)

    H = np.zeros((rho.size,theta.size))

    for y in range(img_edge.shape[0]):
        for x in range(img_edge.shape[1]):
```

```

if img_edge[y,x]:
    for i,t in enumerate(theta):
        r = round(y*np.cos(t*np.pi/180) + x*np.sin(t*np.pi/180))
        H[r+rho_max, i] += 1

return H

```

```

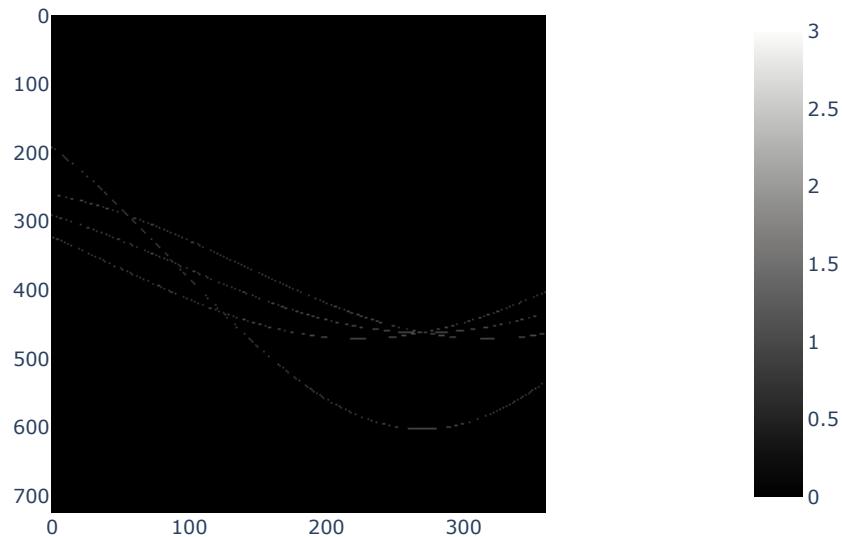
[18]: # img_edge slika je vec u binarnoj formi
H = houghTransform(img_edge)

fig = px.imshow(H, color_continuous_scale='gray')

# prosirivanje x-ose radi boljeg prikaza
fig.update_xaxes(
    scaleanchor = "y",
    scaleratio = 2)

fig.show()

```



Sa H slike je potrebno odrediti N indeksa sa najvećim vrednostima. Ovo možemo postići koristeći `numpy.argsort` funkciju sa parametrom `axis=None` (iako nam je potreban samo jedan član, kada možemo koristiti `numpy.argmax`).

```
[19]: idx = np.argsort(H, axis=None)
idx = idx[-1]

print(idx)
```

166590

Za prebacivanje u matrični indeks koristimo funkciju `numpy.unravel_index`. Potreban parametar je `shape` od H slike.

```
[20]: mat_idx = np.unravel_index(idx, H.shape)
print(mat_idx)
```

(462, 270)

Ove koordinate je potrebno pretvoriti u polazni opseg vrednosti ρ i θ .

```
[21]: rho_max = round(np.sqrt(img_edge.shape[0]**2+img_edge.shape[1]**2))
rho_arr = np.arange(-rho_max, rho_max+1)
theta_max = 90
theta_arr = np.arange(-theta_max, theta_max, 0.5)

rho = rho_arr[mat_idx[0]] # ili, mat_idx[0] - rho_max
print(rho)

theta = theta_arr[mat_idx[1]]
print(theta)
```

100

45.0

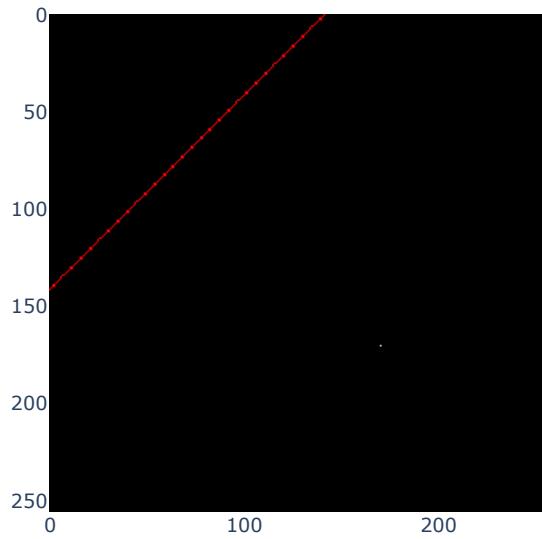
Koristeći ove vrednosti, fiksiramo parametar jedne koordinate x ili y i generišemo drugu radi iscrtavanja crvene linije.

Napomena: ako fiksiramo y , vrednosti x računate putem formule mogu izaći van granice slike, stoga je potrebno postaviti uslov za proveru.

```
[22]: img_rgb = np.stack((img_edge, img_edge, img_edge), axis=2)

for y in range(img_edge.shape[0]):
    x = round((rho - y*np.cos(theta*np.pi/180))/np.sin(theta*np.pi/180))
    if (x>=0) & (x<img_edge.shape[1]):
        img_rgb[y,x,0] = 1
        img_rgb[y,x,1] = 0
        img_rgb[y,x,2] = 0

fig = px.imshow(img_rgb)
fig.show()
```



Dodatak: Prilikom fiksiranja y ose, u slučaju iscrtavanja uglova koji su bliski 0 dolazi do deljenja sa vrednostima bliskim ili jednakim 0, koje ne bi bilo prisutno u slučaju da se vršilo fiksiranje x ose. Da bi se ovo izbeglo, jedno potencijalno rešenje je da na osnovu veličine ugla određujemo ovu odluku.

```
[23]: img_rgb_dodatak = np.zeros(img_rgb.shape)

rho = 100

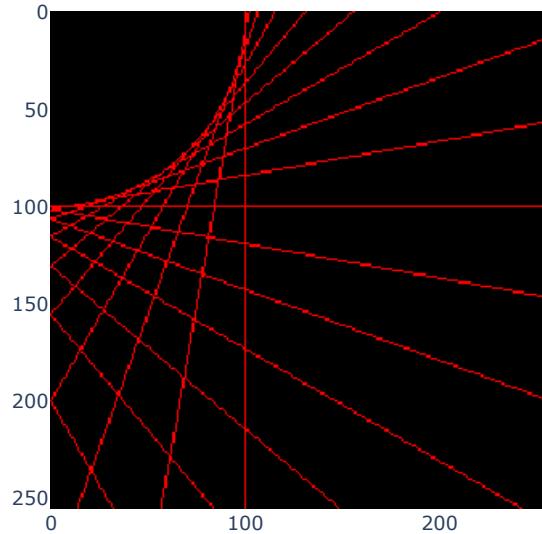
# primer sa razlicitim vrednostima uglova theta
for theta in range(-90,90+1,10):
    if np.abs(theta) < 45:
        for x in range(img_edge.shape[1]):
            y = round((rho - x*np.sin(theta*np.pi/180))/np.cos(theta*np.pi/180))
            if (y>=0) & (y<img_edge.shape[0]):
                img_rgb_dodatak[y,x,0] = 1
                img_rgb_dodatak[y,x,1] = 0
                img_rgb_dodatak[y,x,2] = 0
    else:
        for y in range(img_edge.shape[0]):
            x = round((rho - y*np.cos(theta*np.pi/180))/np.sin(theta*np.pi/180))
            if (x>=0) & (x<img_edge.shape[1]):
                img_rgb_dodatak[y,x,0] = 1
                img_rgb_dodatak[y,x,1] = 0
```

```

img_rgb_dodatak[y,x,2] = 0

fig = px.imshow(img_rgb_dodatak)
fig.show()

```



2 Izdvajanje objekata

Izdvajanje objekata pomoću analize intenziteta piksela i postavljanja adekvatnih pragova (engl. *image thresholding*) zbog jednostavne implementacije zauzima bitno mesto među tehnikama za segmentaciju slike. U opštem slučaju prag koji se primenjuje može da zavisi od položaja piksela, intenziteta piksela i nekih lokalnih svojstava, npr. prosečnog intenziteta. Ukoliko vrednost praga zavisi isključivo od intenziteta piksela reč je o globalnom pragu. Ukoliko zavisi i od lokalnih svojstava, prag nazivamo lokalnim, dok prag koji pored toga zavisi i od pozicije piksela nazivamo dinamičkim ili adaptivnim. U opštem slučaju, globalni prag se može naći vizuelnom analizom histograma i odrediti heuristički.

2.1 Primer 12.5

Implementirati funkciju `globalThresholding` koja na slici primenjuje globalni prag određen sledećim koracima:

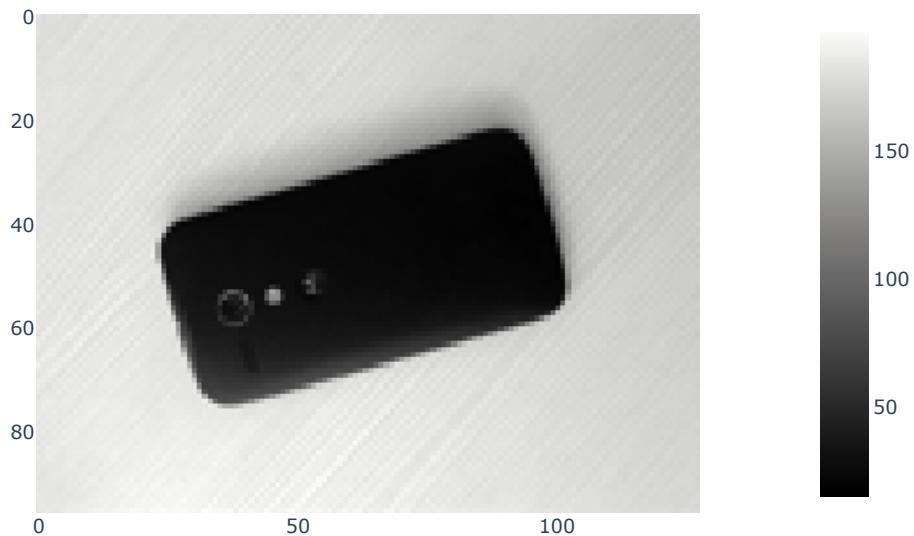
1. postaviti početnu vrednost praga T
2. odrediti srednju vrednost μ_1 svih piksela sa vrednošću manjom od T
3. odrediti srednju vrednost μ_2 svih piksela sa vrednošću većom ili jednakom od T

4. postaviti novu vrednost praga $T = \frac{\mu_1 + \mu_2}{2}$
5. ponavljati korake 2 do 4 sve dok razlika vrednosti pragova određenih u dve uzastopne iteracije ne bude manja od neke unapred zadate vrednosti T_0
6. primeniti dobijeni prag nad slikom

Funkciju primeniti nad slikom `mobile.png`. Prikazati rezultat.

```
[24]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('mobile.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



Za početnu vrednost praga može se uzeti prosečan intenzitet slike u slučajevima kada se očekuje da su objekat i pozadina uporedive veličine. U slučajevima kada je objekat značajno manji prosečnom vrednošću će dominirati pozadina, pa se kao pogodniji početni prag uzima srednja vrednost minimalne i maksimalne vrednosti intenziteta.

```
[25]: def globalThresholding(img, T):
    T0 = 1
    T_old = np.inf
```

```

while np.abs(T-T_old) > T0:
    m1 = img[img < T].mean()
    m2 = img[img >= T].mean()

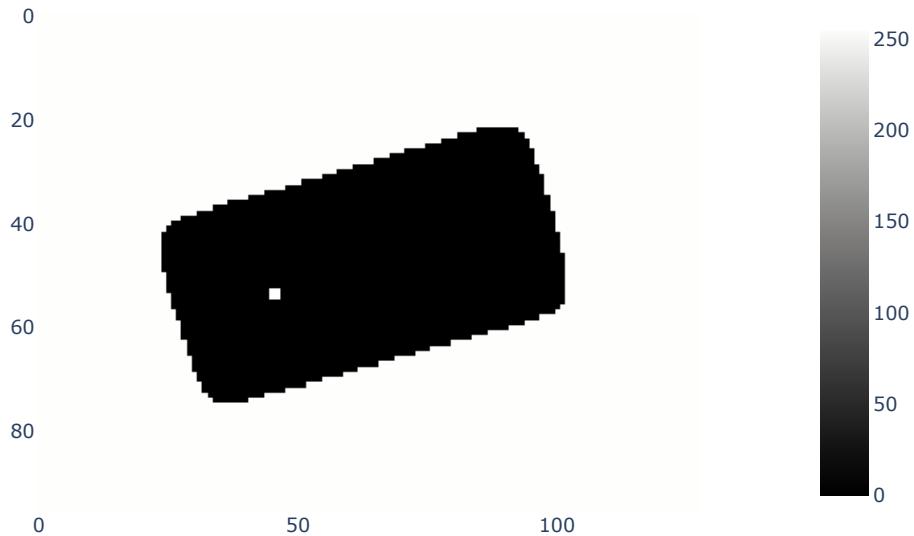
    T_old = T
    T = (m1+m2)/2

return img >= T

```

```
[26]: img_globalseg = globalThresholding(img, T=img.mean())

fig = px.imshow(img_globalseg, color_continuous_scale='gray')
fig.show()
```



2.2 Primer 12.6

Napisati funkciju localThresholding koja na svaki piksel primenjuje lokalni adaptivni prag koji se određuje kao razlika srednje vrednosti u kvadratnoj okolini piksela i parametra konstante C.

Funkciju primeniti nad slikom mobile.png sa parametrom širine kernela 5 i konstante C=5. Prikazati rezultat.

```
[27]: import plotly.express as px
from skimage import io

img = io.imread('mobile.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



```
[28]: from scipy import ndimage

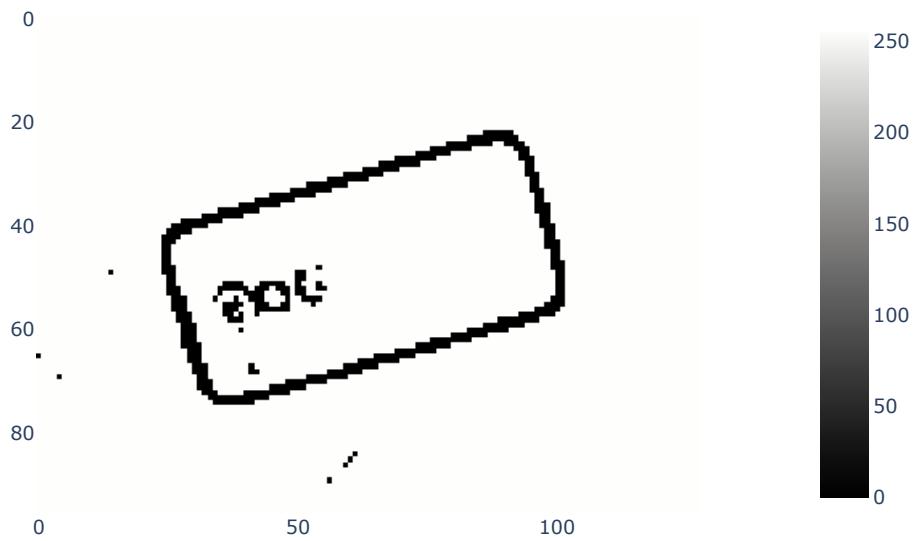
def localThresholding(img, kernel_width, C):
    box = np.ones((kernel_width,kernel_width))/kernel_width**2
    m = ndimage.convolve(img.astype('float'), box, mode='mirror')

    img_localthresh = m - C

    return img >= img_localthresh
```

```
[29]: img_localseg = localThresholding(img, 5, 5)

fig = px.imshow(img_localseg, color_continuous_scale='gray')
fig.show()
```



Parametar C određuje koliko jake promene moraju da budu da bi bile detektovane.

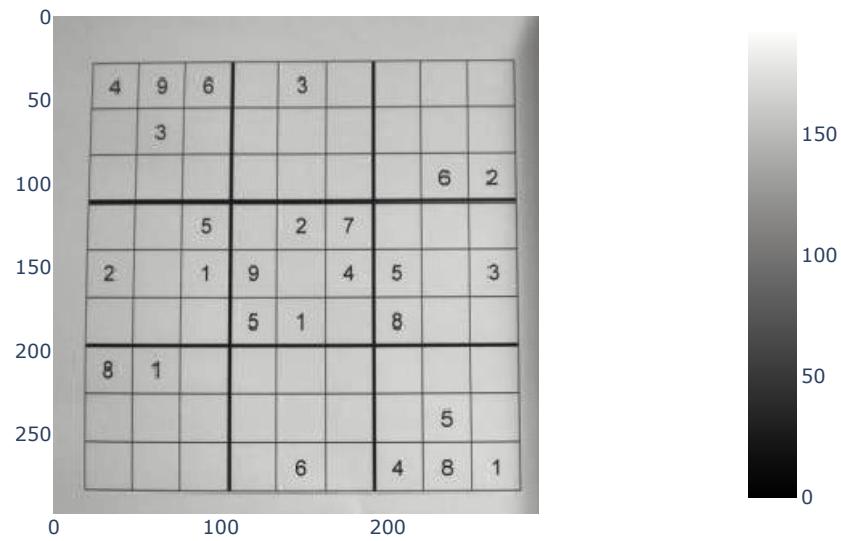
Pored ovog primera lokalni prag može da koristi i druga lokalna svojstva u datom regionu (npr. standardnu devijaciju ili median).

2.3 Primer 12.7

Binarizovati sliku `sudoku.png` upotrebom Ocuove (engl. *Otsu*) metode.

```
[30]: import plotly.express as px
from skimage import io
import numpy as np

img = io.imread('sudoku.png')
fig = px.imshow(img, color_continuous_scale='gray')
fig.show()
```



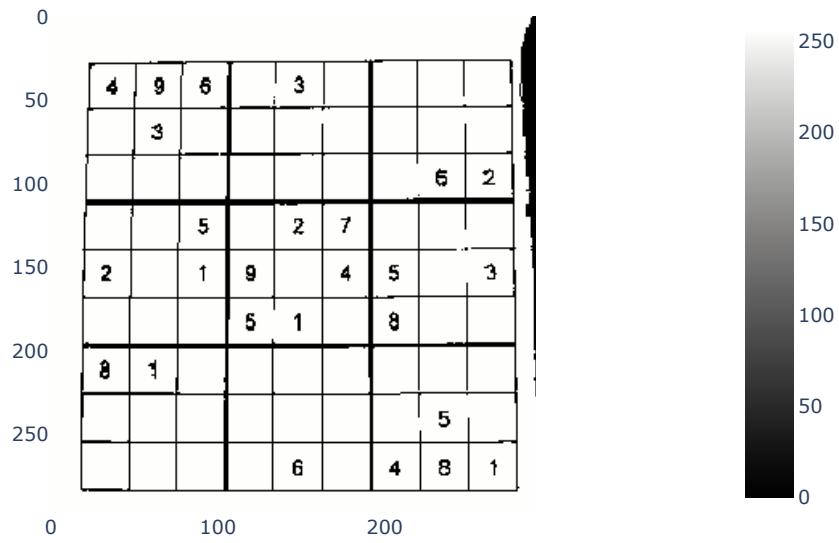
Ocuov metod za cilj ima pronalaženje optimalnog globalnog praga za binarizaciju baziranog na minimizaciji intraklasne varijanse. Funkcija za algoritam je `skimage.filters.threshold_otsu` koja prima parametar ulazne slike (nad kojom se u okviru funkcije računa histogram) ili sam njen histogram.

```
[31]: from skimage import filters

Totsu = filters.threshold_otsu(img)
print(Totsu)

img_otsuseg = img >= Totsu
fig = px.imshow(img_otsuseg, color_continuous_scale='gray')
fig.show()
```

120



3 Zadaci za samostalnu vežbu

Zadatak 12.1

Cilj zadatka je da se automatski odredi ROI koji predstavlja direktno ozračeni deo slike `rtg_2.png`. Da bi se smanjila računarska zahtevnost obrade snimaka često se analiziraju slike dobijene smanjenjem veličine originalnog snimka. Veličinu snimka nakon učitavanja smanjiti 10 puta. Na umanjenom snimku, nakon određivanja slike ivica koristeći Canny algoritam, korišćenjem Hafove transformacije odrediti 4 najznačajnije prave linije, pazeći da svaka odgovara drugoj stranici okvira. Pronaći tačke preseka te 4 linije i iskoristiti ih da bi se automatski izdvojio ROI sa originalnog snimka.

[32] : `#TODO`

Zadatak 12.2

Napisati funkciju `localThresholding` koja vrši lokalno pragovanje koristeći razliku medijana okoline i konstante C . Isprobati ponašanje funkcije na slici `mobile.png` upotrebom različitih vrednosti ulaznih argumenata.

[33] : `#TODO`

4 Dokumentacija novih celina

- [skimage.feature.canny](#)
- [skimage.filters.threshold_otsu](#)
- [numpy.argsort](#)
- [numpy.argmax](#)
- [numpy.unravel_index](#)