# 112: Spektralna analiza in filtriranje

Peter Rupnik

16. januar 2019

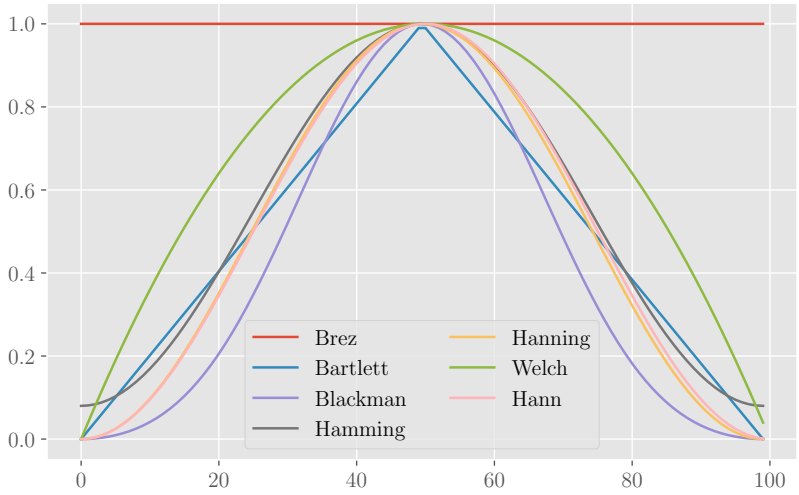
## 1 Prva naloga

Signaloma s 512 točkami na datotekah `val2.dat` in `val3.dat` določi frekvenčni spekter. Preskusi različne okenske funkcije. Kako se spremeni spekter, če analiziramo krajše intervale (64, 128,... točk)?

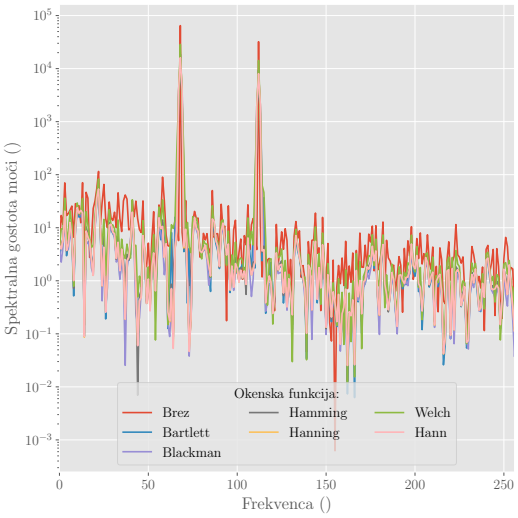
### 1.1 Fantastic window functions and where to find them

V Pythonskem arzenalu za numeriko `numpy` najdemo okenske funkcije `numpy.bartlet`, `numpy.blackman`, `numpy.hamming`, `numpy.hanning` in `numpy.kaiser`. Sam sem vključil še nekaj funkcij, ki smo jih omenili na predavanju (Hann in Welch). Svojim implementacijam ne zaupam preveč, saj pri arrayih z nizko in liho dolžino kažejo v nebo vpijočo asimetrijo. Ker bom delal z arrayi sode dolžine red velikosti ali dva nad kritičnimi, se za te t.i. *Obi–Wan errorje*[1] ne bom kaj dosti menil. Na sliki 1 prikazujem obliko okenskih funkcij na arrayu dolžine 100.

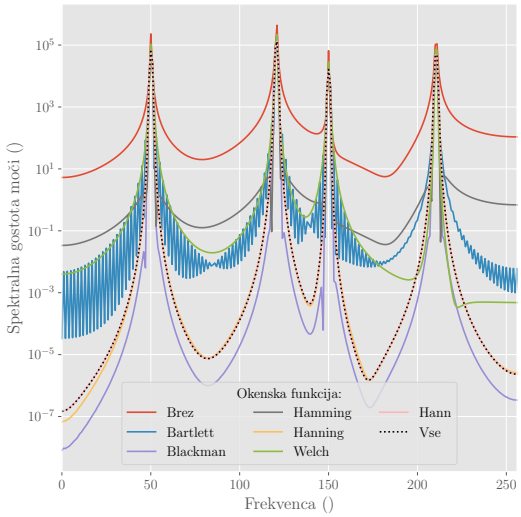
[1] Obi–Wan error je omiljena nomenklatura prof. Bauerja za takozvan *off-by-one error*.




Slika 1: Nekaj študiranih okenskih funkcij, prikazanih na arrayu 100 elementov.




Slika 2: Spekter podatkov iz datoteke `val2.dat`. Zaradi zašumljenosti na pogled težko razločimo, katera od okenskih funkcij se najbolje odreže. Z okenskimi funkcijami ali brez njih jasno razločimo dva prominentna vrha, v katerih je spektralna gostota moči skoraj 4 rede velikosti nad ozadjem.




Slika 3: Spekter podatkov iz datoteke `val3.dat`. Na pogled izgleda signal dosti bolj čist, zaradi česar lepše razločimo tudi denimo nepričakovane artefakte Bartlettove okenske funkcije. Ker je signal manj zašumljen, sem poskusil tudi z okensko funkcijo, ki se dobi s komponiranjem vseh predhodnih okenskih funkcij, rezultat prikazujem pod oznako `Vse`. Seveda nisem pričakoval, da bo mineštra vseh funkcij delovala bolje kot posamezne funkcije, a sem bil vseeno presenečen, kako dobro se odreže.

## 1.2 Učinek okenskih funkcij na spekter signala.

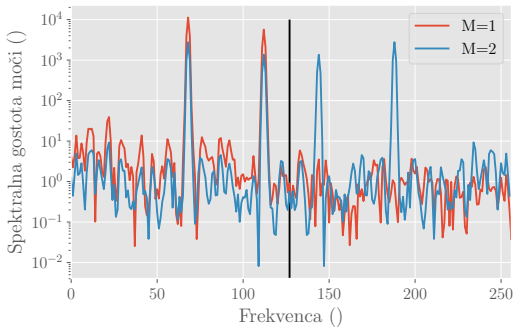
Učinke različnih okenskih funkcij prikazujem na slikah 2 in 3.

## 1.3 Decimacija signala

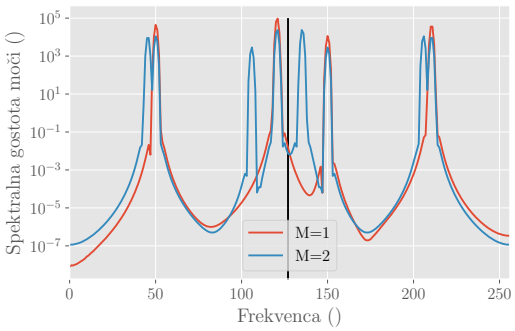
V nadaljevanju bom namesto polne resolucije gledal signal po decimaciji[2], pri čemer bom nekajkrat iterativno signalu porezal vsako drugo točko. Izvoren signal sem pred obdelavo pomnožil z Blackmanovo okensko funkcijo, ki me je v prvem delu naloge najbolj navdušila.

Pri tovrstnem početju lahko slej ko prej pričakujem efekte potujevanja, saj je po decimaciji efektivna frekvenca vzorčenja manjša, zaradi česar bo manjša tudi Nyquistova frekvenca, torej bo po dovolj radikalni decimaciji vsak frekvenčni vrh potujen. Zaradi efektov potujevanja pričakujemo manjše vrhove. Načeloma bi morali pri decimaciji signal vedno prefiltrirati skozi nizkoprepusten filter, ki zagotovi, da v decimiranem signalu ni frekvenc nad Nyquistovo frekvenco. Rezultate mojih poskusov najdemo na sliki 6 in sliki 7.

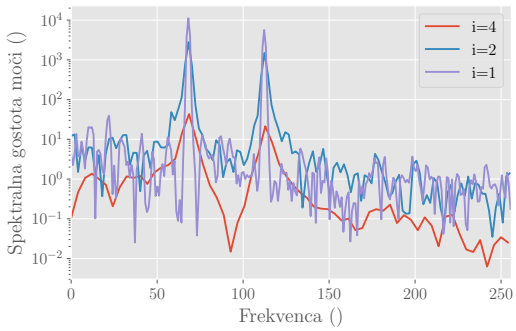
[2]Nekateri striktno ločijo med decimacijo in *downsamplingom*, pri čemer eno pomeni le rezanje vsake $M$–te točke, drugo pa še predhodno filtriranje z low-pass filtrom, vendar konsenza ni čutiti, večkrat sem opazil oba termina za nasprotujoči si uporabi.



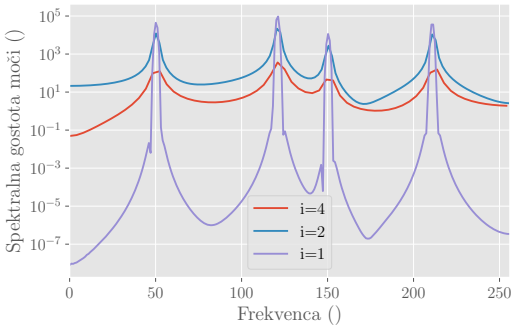

Slika 4: Spekter podatkov iz datoteke `val2.dat` pred in po decimaciji s faktorjem $M$. Opazimo lahko, da smo dobili zrcalno sliko spektra čez Nyquistovo frekvenco, ki je pri decimiranem signalu manjša za faktor $M$.




Slika 5: Spekter podatkov iz datoteke `val3.dat`, decimiran s faktorjem $M$. Spet je očitna simetrija glede na razpolovišče prejšnjega spektra.




Slika 6: Spekter podatkov iz datoteke `val2.dat`, če upoštevamo samo prvih $512/i$ točk.




Slika 7: Spekter podatkov iz datoteke `val3.dat`, če upoštevamo samo prvih $512/i$ točk. Zanimivo je, da je spekter polovice podatkov bolj *zapacan* kot spekter celih podatkov ali spekter četrtine signala.

### 1.4 Spekter krajšega dela signala

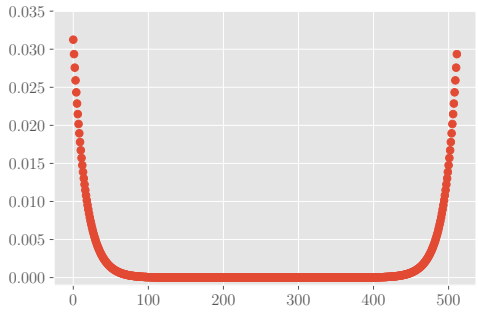
Obdeloval sem tudi manjše dele signala (cel signal, prvo polovico, prvo četrtino) in si ogledal, kakšne spektre dobim. Po pričakovanju je zaradi manjšega časa vzorčenja širina vrhov večja, višina vrhov nad šumom pa manjša.

## 2 Druga naloga

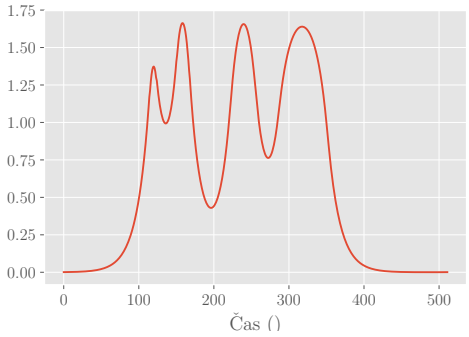
Signal $u(t)$, ki prihaja v merilno napravo s prenosno funkcijo $r(t)$, se ob dodatku šuma $n(t)$ preoblikuje v

$$c(t) = u(t) * r(t) + n(t) = s(t) + n(t).$$

Iz izmerjenega časovnega poteka $c(t)$ bi radi, ob poznavanju odzivne funkcije r(t) in ob nekaterih predpostavkah o šumu $n(t)$, rekonstruirali vpadni signal $u(t)$. N. Wiener




Slika 8: Array prenosne funkcije $r(t)$




Slika 9: Podatki iz datoteke `signal0.dat` v časovnem prostoru. Ti podatki ustrezajo $s(t)$ v navodilu.

je predlagal naslednjo rešitev, ki sledi iz minimizacije napake po metodi najmanjših kvadratov. Pred dekonvolucijo je treba transformiranko $C(f)$ pomnožiti s filtrom,

$$\Phi(f)=\frac{|S(f)|^2}{|S(f)|^2+|N(f)|^2}. \tag{1}$$

S pomočjo Wienerjevega filtra napravi dekonvolucijo signalov na datotekah `signal{0,1,3,4}.dat`. Število točk v posameznem signalu je 512. Na zadnjih treh datotekah je signalu primešan šum. Prenosna funkcija je

$$r(t)=\frac{1}{2\tau}\exp(-|t|/\tau), \qquad \tau=16. \tag{2}$$

Moj prvi korak je bila priprava arraya prenosne funkcije, ki sem ga konstruiral kot:

```
razpon = np.arange(0, 512,1)
r = (np.exp(-np.abs(razpon)/tau)+ np.exp(-np.abs(512-razpon)/tau))/(2*tau).
```

Rezultirajoči array je na ogled na sliki 8.

Nadalje sem si pogledal signal iz datoteke `signal0.dat`, ki ne vsebuje šuma. Zaradi tega ustreza količini $s(t)$. Za poseben primer, ko imamo podatke brez šuma, lahko zapišem:

$$c(t)=s(t)+n(t)=(u*r)(t)+n(t)=(u*r)(t) \tag{3}$$

$$C(f)=U(f)\cdot R(f) \tag{4}$$

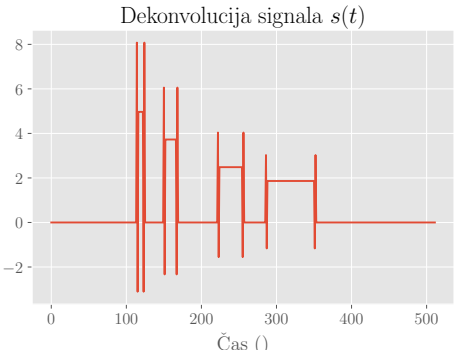
$$\Phi(f)=\frac{|S(f)|^2}{|S(f)|^2+|N(f)|^2}=1 \tag{5}$$

$$\tilde{U}(f)=\frac{C(f)\Phi(f)}{R(f)} \tag{6}$$

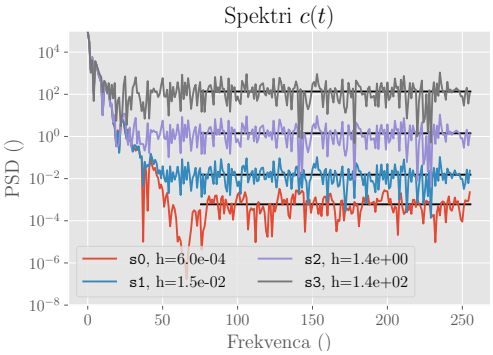
$$\tilde{u}(t)=\mathcal{F}^{-1}\left[\frac{\mathcal{F}(c)\mathcal{F}(\Phi)}{\mathcal{F}(r)}\right]=\mathcal{F}^{-1}\left[\frac{\mathcal{F}(c)}{\mathcal{F}(r)}\right] \tag{7}$$

Po enačbi (7) sem izračunal dekonvolucijo signala brez šuma (slika 10), nato pa sem si ogledal spektre vseh datotek, s katerimi bom konstruiral filter $\Phi(f)$ za signal vsake izmed datotek.

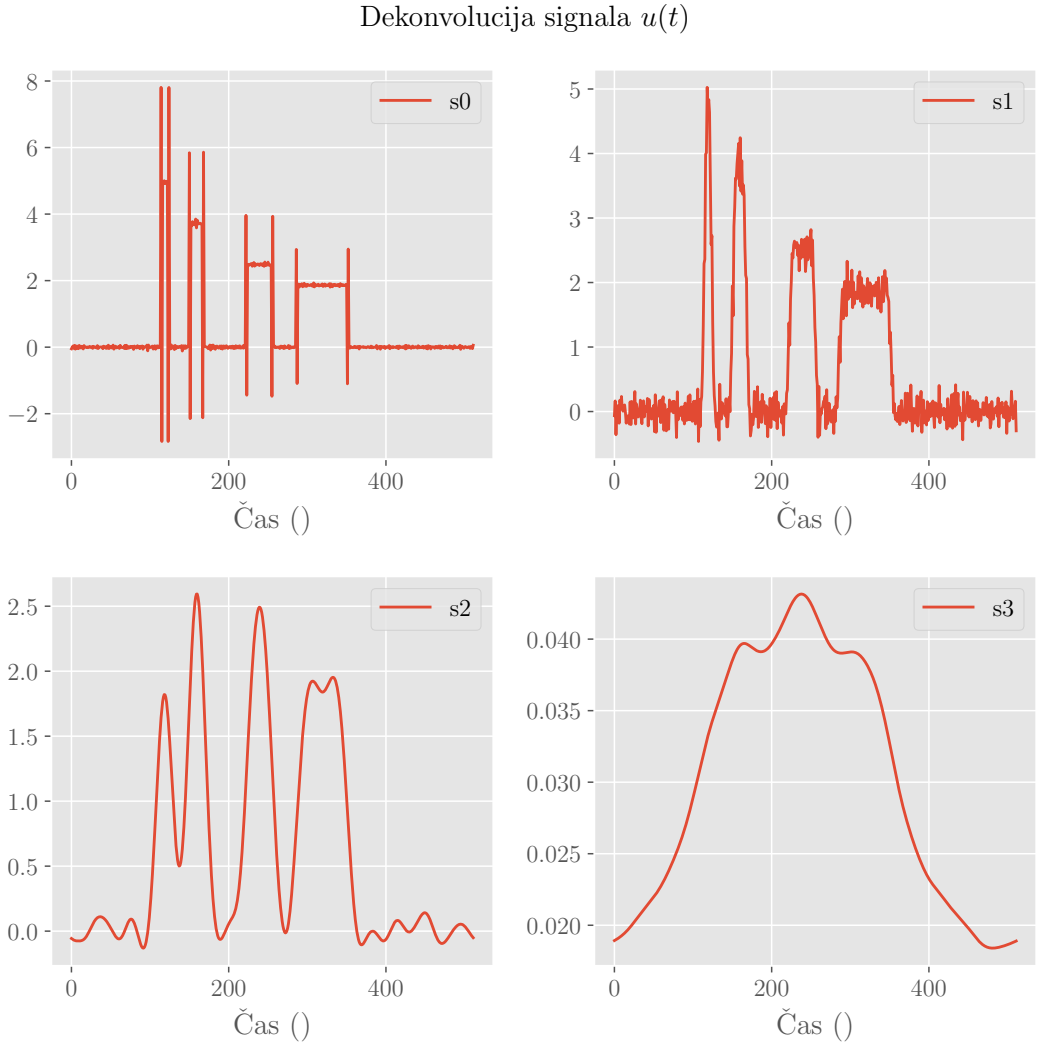
V nadaljni obravnavi sem uporabil signal brez dodanega šuma kot $s(t)$ iz enačb. Za vsako datoteko sem konstruiral filter s šumom, ki sem ga aproksimiral s konstantno vrednostjo v frekvenčnem prostoru, to vrednost pa sem določil v predhodnjem koraku, kot kaže slika 11. Rezultat prikazujem na sliki 12. Ugotovil sem, da lahko to še izboljšam, če ročno prilagodim nivoje šuma, in tako je nastala slika 13.



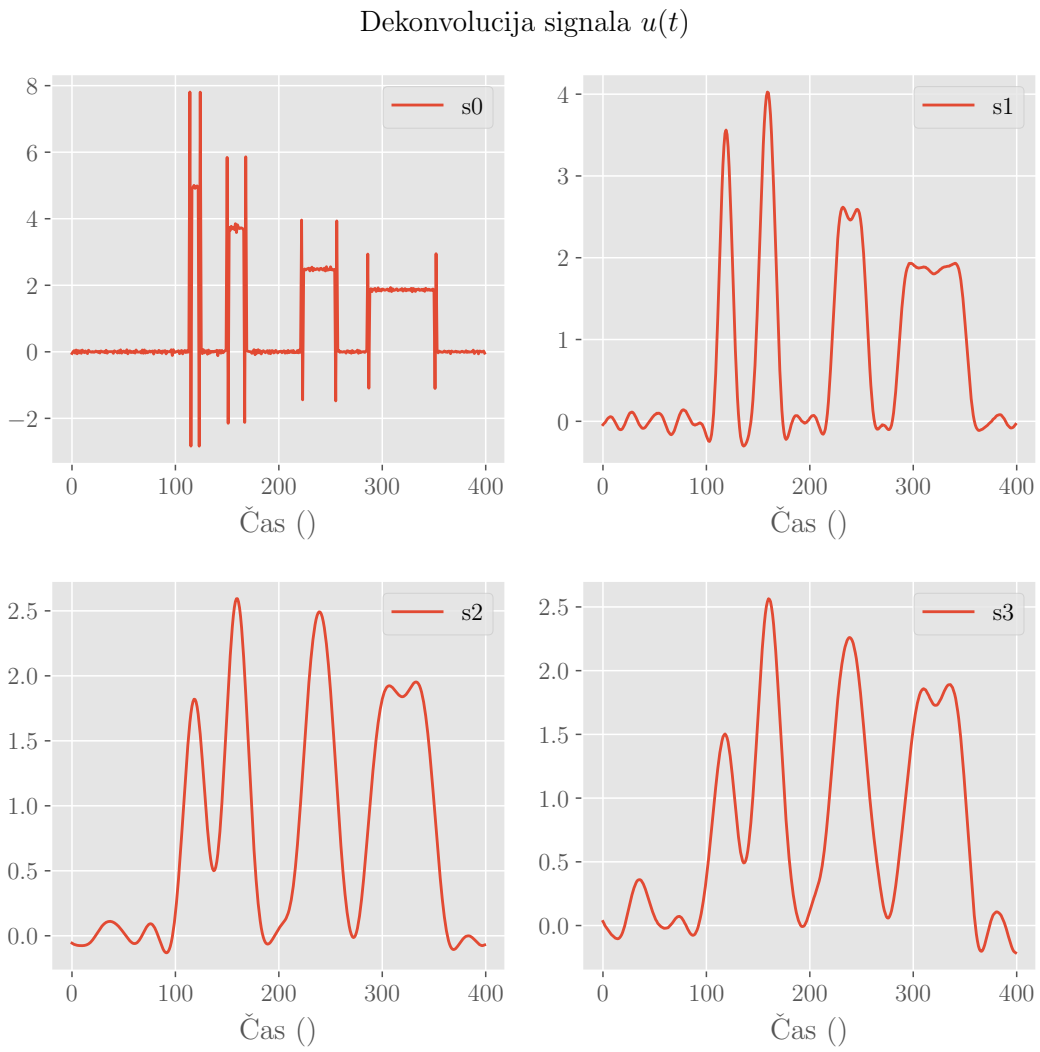

Slika 10: Potek vhodnega signala $u(t)$, kot ga dobimo z dekonvolucijo



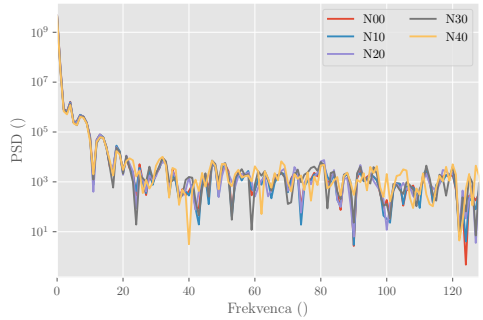

Slika 11: Spektri signalov $c(t)$ iz različnih datotek (označene kot `s{i}`, za `signal{i}.dat`), v legendi najdemo tudi njihove mediane od frekvence pri indeksu `75` dalje. Te nivoje bom uporabil za konstrukcijo filtra.



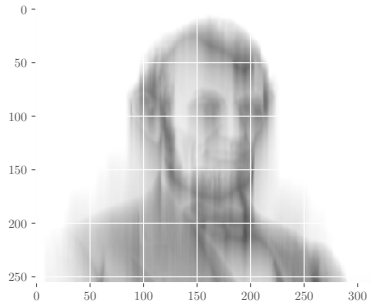

Slika 12: Avtomatska obdelava mi ponudi sledeče rekonstrukcije signala $u(t)$. Predvsem `s1` in `s3` se zdita piškavo rekonstruirana, saj pri prvem opazimo preveč migotanja, pri drugem pa je rekonstruiran signal neuporaben, saj izgubimo bolj ali manj vso informacijo o dogajanju pred konvolucijo.




Slika 13: Z ročnim finim nastavljanjem nivojev šuma, s katerimi sem konstruiral filtre, lahko izboljšam prej problematična signala in izboljšam oceno poteka $u(t)$.




Slika 14: Spektri Lincolnovih podob iz različnih datotek (končnice prikazujem v legendi).



Slika 15: Wienerjev filter najdemo tudi v `scipy.signal.wiener`, tako kot še ogrooomno druge z obdelavo signalov povezane šare z zelo čudnim API. Če takemu filtru ponudimo nezašumljeno datoteko `N00.dat`, nam zna brez drugih informacij ponuditi tako rekonstrukcijo. Naša rekonstrukcija je občutno boljša.

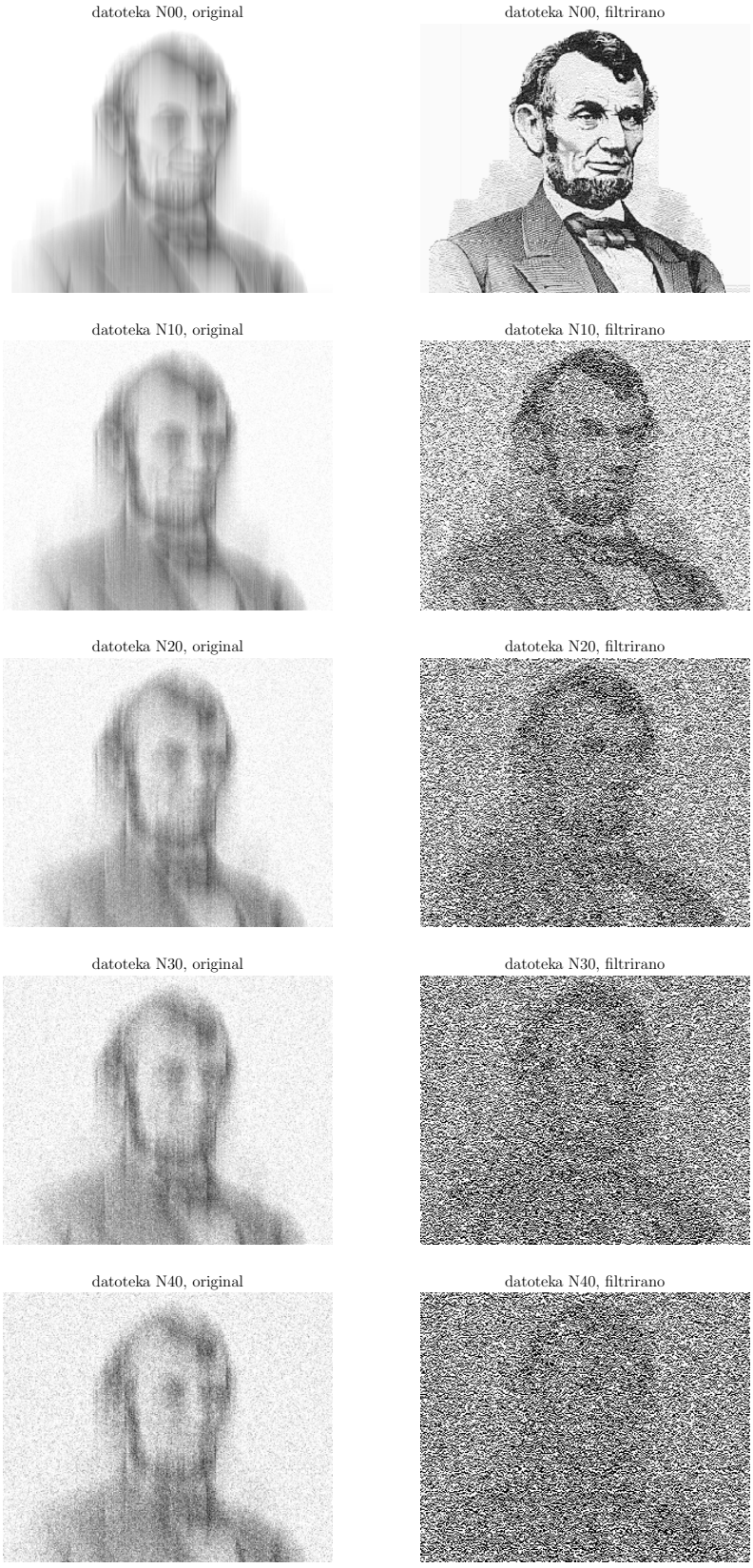
# 3 Tretji del

Poskusi očistiti Lincolnovo podobo, odbrano po stolpcih, ki se v metodi LA-P-MS tradicionalno uporablja za testiranje dekonvolucijskih algoritmov.

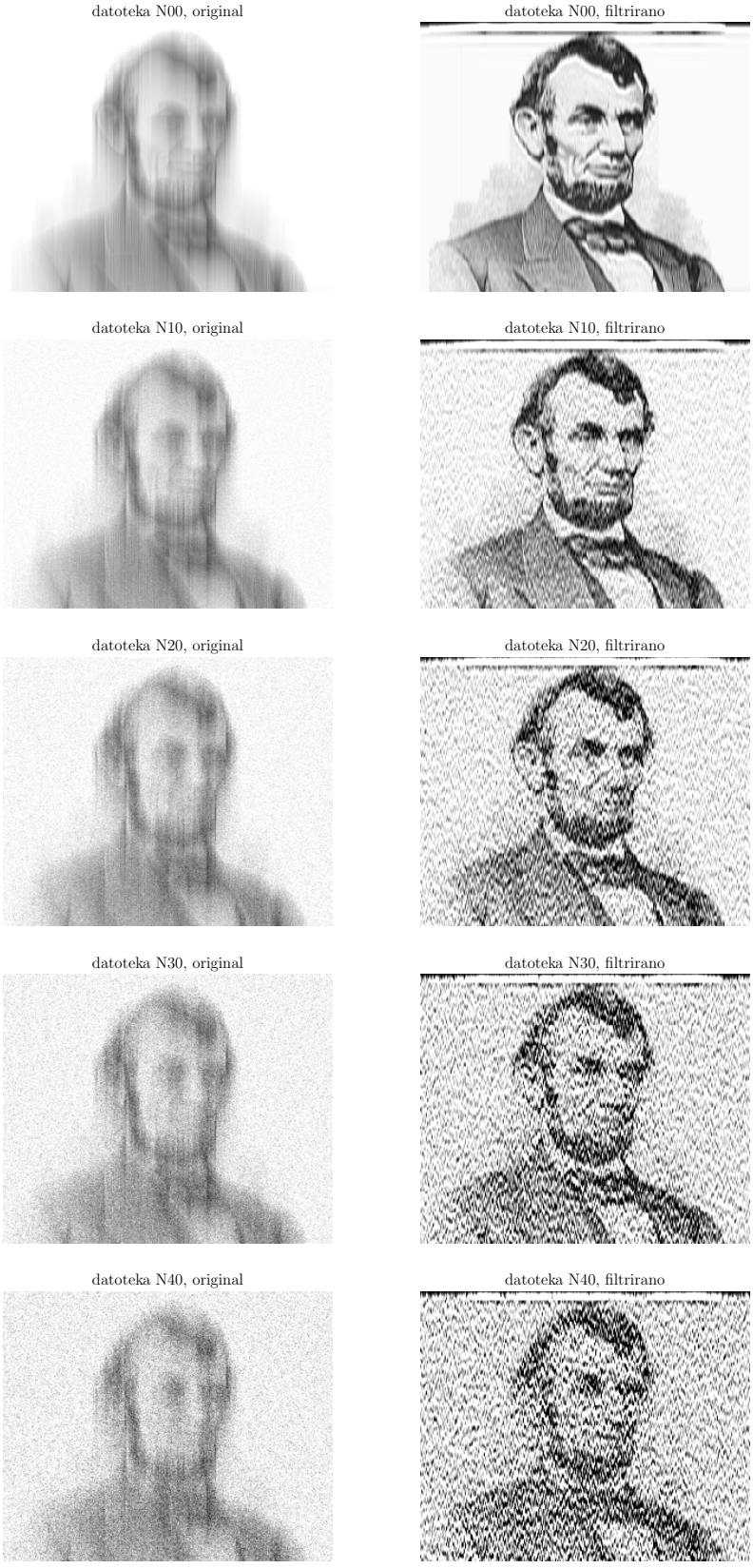
Slike `lincoln_L30_N{00,10,30}.pgm` ($313 \times 256$) najdeš na spletni strani predmeta. V zadnjih dveh datotekah je dodan šum. Prenosna funkcija je

$$r(t) = \frac{1}{\tau}\exp(-t/\tau), \qquad \tau = 30, t > 0. \tag{8}$$

Najprej sem si pripravil funkcijo, ki iz obskurnega formata `.pmg` napravi `.csv`, ki je bolj prebavljiv. Ogledal sem si Lincolne. Vidimo, da bomo filtrirali arraye dolžine 256, kar je odlično za FFT, ki najhitreje deluje na dolžinah $2^n$. Ogledal sem si spektre vseh datotek. Ker dobimo za vsako datoteko 313 spektrov, bi bil prikaz vseh spektrov nepregleden, zato sem spektre posameznih datotek povprečil v enega in ga prikazal na sliki 14. Ta pristop sem izbral, ker sem pričakoval, da bom filter generiral samo enkrat za posamezno datoteko, vendar pa zaradi definicije Wienerjevega filtra vidim, da bom moral filter konstruirati za vsak stolpec posebej, zato sem svoj pristop popravil, sliko 14 pa bom recikliral za določevanje dolžine, od katere naprej bom povprečil spekter in tako pridobil oceno šuma $N(f)$. Za vsako datoteko posebej sem tako dobljeno oceno še pomnožil s predfaktorjem, kar mi je omogočilo, da za vsako datoteko šum v vsakem stolpcu ojačim z istim predfaktorjem. Izkaže se, da nisem profitiral kaj dosti, saj dobim najlepše rezultate takrat, ko so predfaktorji vseh datotek približno enaki.




Slika 16: Vhodne slike (levo) in rekonstruirane slike po dekonvoluciji. Prvo sliko rekonstruiramo perfektno, razločimo celo črte na suknji. Ostale rekonstrukcije so bolj zašumljene.




Slika 17: Če na rezultatih s slike 16 uporabim še nizkoprepusten FIR filter, s katerim se znebim hitrofrekvenčnega migotanja, dobim sledeče rezultate. Za zašumljene vhodne datoteke se kvaliteta malce izboljša, za nezašumljeno rekonstruirano sliko pa poslabša.