4. naloga: Fourierova analiza

Pri numeričnem izračunavanju Fourierove transformacije

$$H(f) = \int_{-\infty}^{\infty} h(t) \exp(2\pi \mathrm{i} f t)\, \mathrm{d}t \tag{1}$$

$$h(t) = \int_{-\infty}^{\infty} H(f) \exp(-2\pi \mathrm{i} f t)\, \mathrm{d}f \tag{2}$$

je funkcija $h(t)$ običajno predstavljena s tablico diskretnih vrednosti

$$h_k = h(t_k), \quad t_k = k\Delta, \quad k = 0, 1, 2, \ldots N-1. \tag{3}$$

Pravimo, da smo funkcijo vzorčili z vzorčno gostoto (frekvenco) $f = 1/\Delta$. Za tako definiran vzorec obstaja naravna meja frekvenčnega spektra, ki se imenuje *Nyquistova frekvenca*, $f_c = 1/(2\Delta)$: harmonični val s to frekvenco ima v vzorčni gostoti ravno dva vzorca v periodi. če ima funkcija $h(t)$ frekvenčni spekter omejen na interval $[-f_c, f_c]$, potem ji z vzorčenjem nismo odvzeli nič informacije, kadar pa se spekter razteza izven intervala, pride do *potujitve* (*aliasing*), ko se zunanji del spektra preslika v interval.

Frekvenčni spekter vzorčene funkcije (3) računamo samo v N točkah, če hočemo, da se ohrani količina informacije. Vpeljemo vsoto

$$H_n = \sum_{k=0}^{N-1} h_k \exp(2\pi \mathrm{i} k n/N), \qquad n = -\tfrac{N}{2}, \ldots, \tfrac{N}{2}, \tag{4}$$

ki jo imenujemo diskretna Fourierova transformacija in je povezana s funkcijo v (1) takole:

$$H(\tfrac{n}{N\Delta}) \approx \Delta \cdot H_n.$$

Zaradi potujitve, po kateri je $H_{-n} = H_{N-n}$, lahko pustimo indeks n v enačbi (4) teči tudi od 0 do N. Spodnja polovica tako definiranega spektra ($1 \le n \le \frac{N}{2} - 1$) ustreza pozitivnim frekvencam $0 < f < f_c$, gornja polovica ($\frac{N}{2} + 1 \le N - 1$) pa negativnim, $-f_c < f < 0$. Posebna vrednost pri $n = 0$ ustreza frekvenci nič ("istosmerna komponenta"), vrednost pri $n = N/2$ pa ustreza tako f_c kot $-f_c$.

Količine h in H so v splošnem kompleksne, simetrija v enih povzroči tudi simetrijo v drugih. Posebej zanimivi so trije primeri:

če je	h_k realna	tedaj je	$H_{N-n} = H_n^*$
	h_k realna in soda		H_n realna in soda
	h_k realna in liha		H_n imaginarna in liha

(ostalih ni težko izpeljati). V tesni zvezi s frekvenčnim spektrom je tudi moč. *Celotna moč* nekega signala je neodvisna od reprezentacije, Parsevalova enačba pove

$$\sum_{k=0}^{N-1} |h_k|^2 = \frac{1}{N} \sum_{n=0}^{N-1} |H_n|^2$$

(lahko preveriš). Pogosto pa nas bolj zanima, koliko moči je vsebovane v frekvenčni komponenti med f in $f + \mathrm{d}f$, zato definiramo enostransko spektralno gostoto moči (*one-sided power spectral density*, PSD)

$$P_n = |H_n|^2 + |H_{N-n}|^2 \,.$$

Pozor: s takšno definicijo v isti koš mečemo negativne in pozitivne frekvence, vendar sta pri realnih signalih h_k prispevka enaka, tako da je $P_n = 2\,|H_n|^2$.

Z obratno transformacijo lahko tudi rekonstruiramo h_k iz H_n

$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n \exp(-2\pi \mathrm{i} k n/N) \tag{5}$$

(razlika glede na enačbo (4) je le predznak v argumentu eksponenta in utež $1/N$).

Naloga:

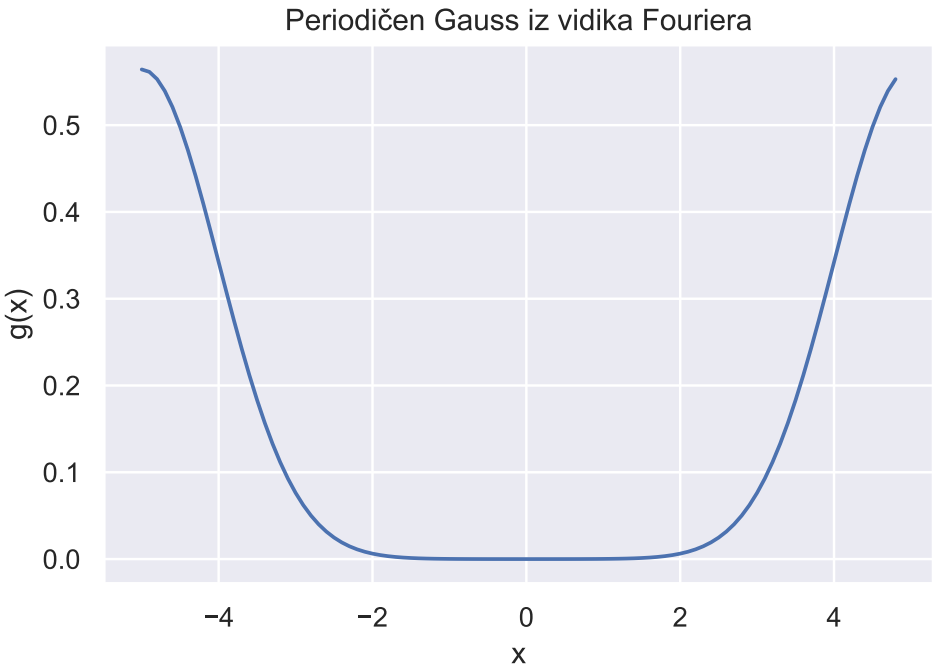
1. Izračunaj Fourierov obrat Gaussove porazdelitve in nekaj enostavnih vzorcev, npr. mešanic izbranih frekvenc. Za slednje primerjaj rezultate, ko je vzorec v intervalu periodičen (izbrane frekvence so mnogokratniki osnovne frekvence), z rezultati, ko vzorec ni periodičen (kako naredimo Gaussovo porazdelitev 'periodično' za FT?). Opazuj pojav potujitve na vzorcu, ki vsebuje frekvence nad Nyquistovo frekvenco. Napravi še obratno transformacijo (5) in preveri natančnost metode. Poglej, kaj se dogaja z časom računanja - kako je odvisen od števila vzorčenj?

2. Po Fourieru analiziraj 2.3 s dolge zapise začetka Bachove partite za violino solo, ki jih najdeš na spletni strani Matematičnofizikalnega praktikuma. Signal iz začetnih taktov partite je bil vzorčen pri 44 100 Hz, 11 025 Hz, 5512 Hz, 2756 Hz, 1378 Hz in 882 Hz. S poslušanjem zapisov v formatu `.mp3` ugotovi, kaj se dogaja, ko se znižuje frekvenca vzorčenja, nato pa s Fourierovo analizo zapisov v formatu `.txt` to tudi prikaži.

3. **Dodatno:** Napravi Fourierovo analizo signalov, ki jih dobiš pri vaji *Akustični resonator* pri Fizikalnem praktikumu II. Posnetke treh različnih signalov prav tako najdeš na spletni strani.

1 Reševanje

Nekje je treba začeti. Naprej moramo seveda napisati svoje Fourieove ter Gaussovo funkcijo. Paziti moramo, da preden naredimo poskušamo delovati z Fourierem na katerokoli funkcijo, to funkcijo naredimo periodično. Pri Gaussovki to storimo tako, da nekje obrežemo repe in jo transliramo (na primer s funkcijo `np.roll`) čez sredino. Ko to naredimo dobimo graf 1



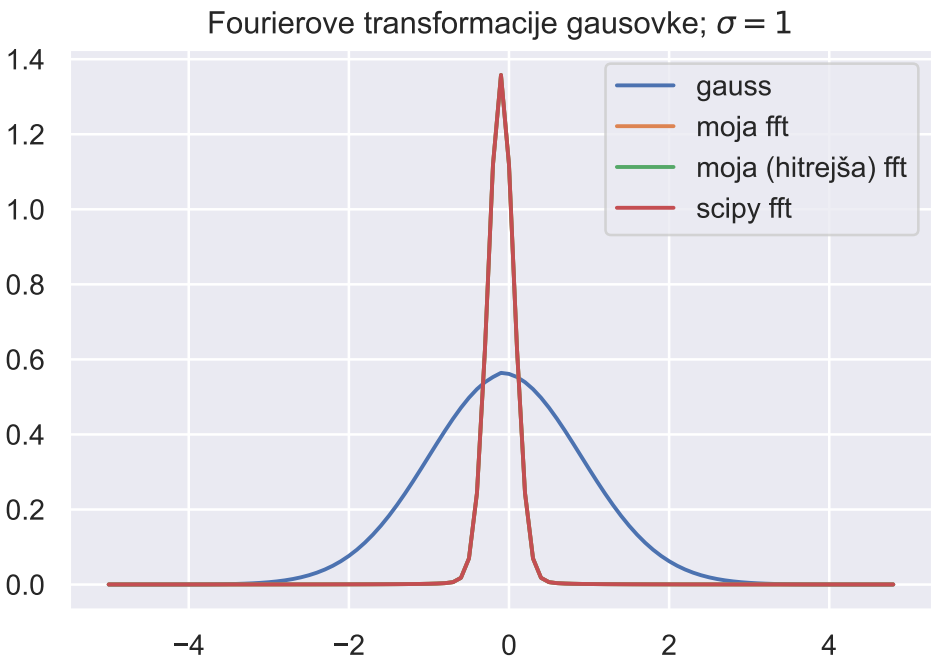
Slika 1: Periodičen Gauss iz vidika Fourierove transformacije

Če sedaj na njem naredimo Fourierovo transformacijo in jo zarotiramo nazaj na prvotno stanje dobimo malce višjo in ožjo Gausovko. To sem naredil s tremi algoritmi, vendar so vsi trije enaki. Najhitrejši pa je seveda `Scipy` implementacija `fft`.

Mogoče nepomemben komentar vendar sem se nekaj ur ukvarjal s tem, da se ugotovil, da je pravilen člen v fourierjovi transformaciji, da vzamem $N-1$ in ne N v koeficientu (odvisno od implementacije, vendar je zares težko opaziti razliko).

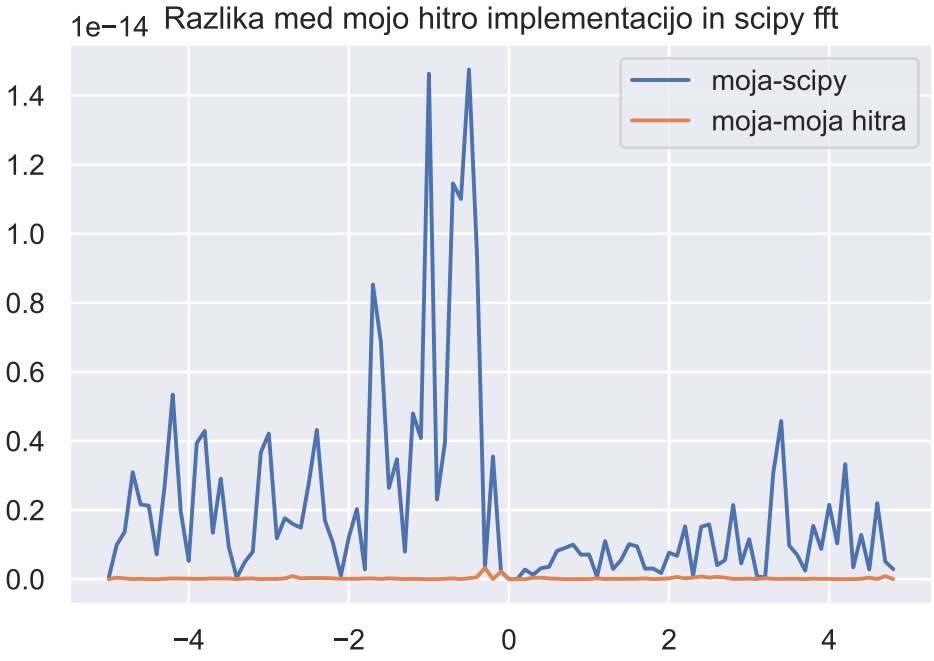
Napisal sem dve različni implementaciji. Najprej sem napisal osnovno verzijo, ki uporablja zanke v pythonu, kar je seveda (relativno) izjemno počasi. Nato sem napisal še verzijo, ki je popolnoma vektorska; generira matriko koeficientov za fourierove transformacije potem pa jo zmnoži z vektorjem vrednosti funkcije. To nam seveda poda vektor H_k fourierove transformacije.

Na grafu 2 lahko opazimo, da se vse metode praktično sovpadajo, moji implementaciji še malenkost bolj ker sta seveda v principu enaki. Vprašanje na mestu je, zakaj bi sploh uporabljal osnovno (ne vektorsko) verzijo FT. Za zelo velike matrike, je precej boljše vzeti horizontalen scaling, ker nam to ne zasede rama (lahko bi napisal še eno verzijo, kjer je tudi množenje vektorjev napisano na roke, vendar bi to imeli druga ozka grla - Python for zanke...). Kot bomo videli v zadnjem poglavju, se je osnovna implementacija izkazala za zelo koristno pri obdelavi Bachove sonate.



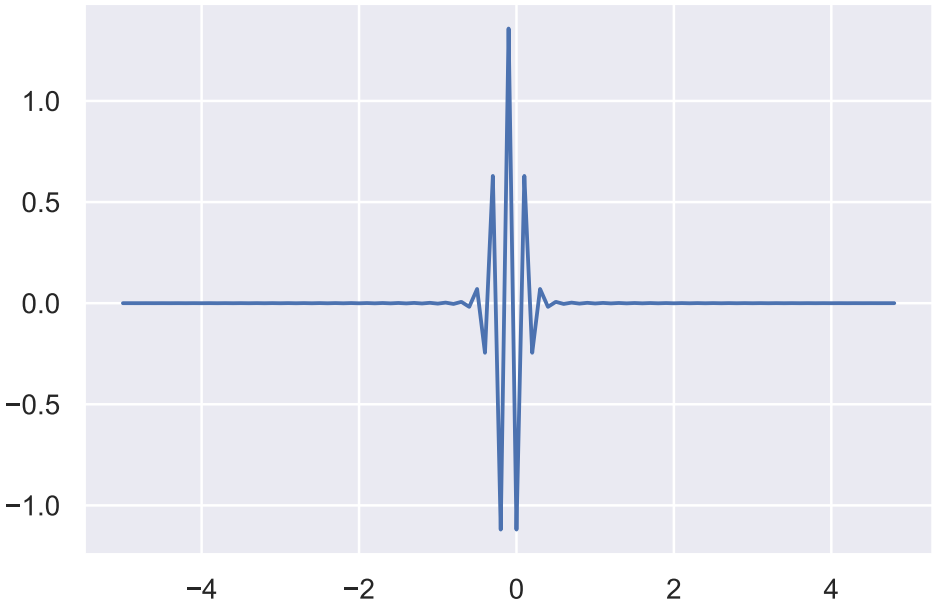
Slika 2: Fourierova transformacija Gausovke s tremi algoritmi.

Bolj pregledno opazimo razliko med algoritmi, če narišemo razliko (ki je zelo majhna).



Slika 3: Razlika Fouriera med različnimi algoritmi.

Kaj se zgodi če ne upošteva pravil in kar naredimo Fouriera na navadni Gaussovki?

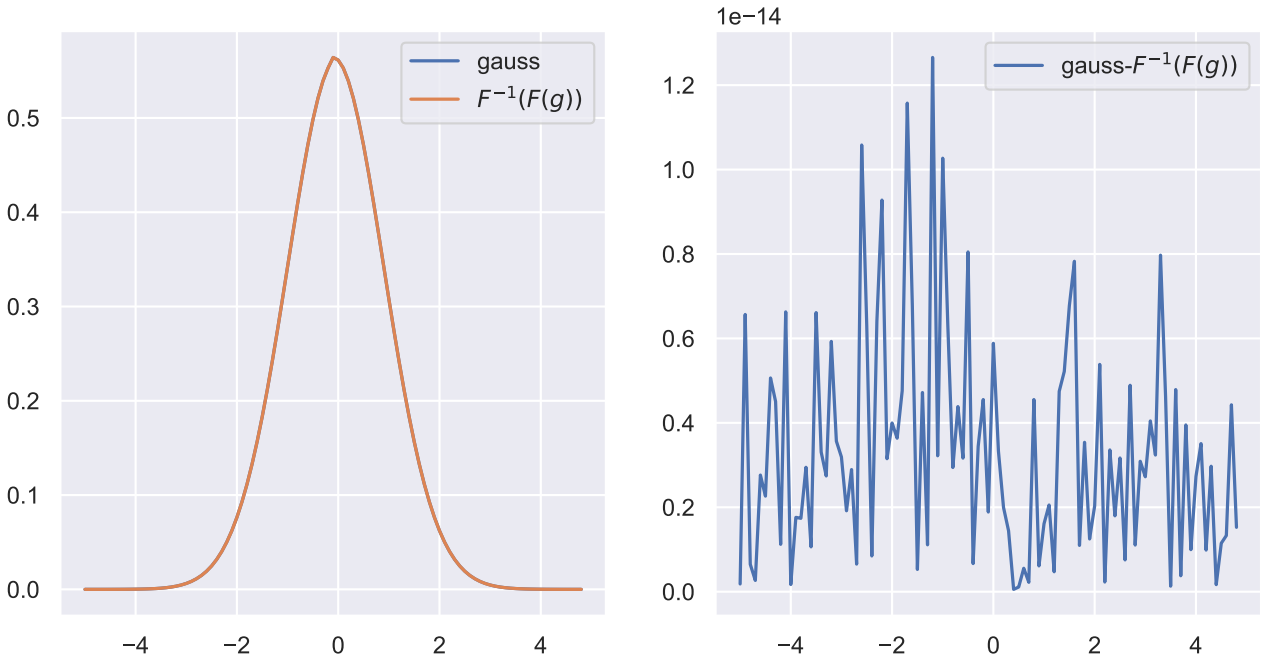


Slika 4: Nepravilna implementacija Fouriera na Gaussovki. Kar dobimo je močna modulacija.

Če funkcije ne naredimo periodične (oziroma jo naredimo napačno periodično), dobimo močno modulacijo kot Fourierovo transformacijo. Zato je potrebna periodičnost ali pa dodamo modulacijski faktor $\exp(2\pi i\nu T(N/2))$.

1.1 Razlika med inverzom Fouriera in Gaussovko

Sedaj lahko preverimo, če so naši algoritmi za Fouriera zares pravilni. To naredimo tako, da delujemo z Inverzno Fourierovo transformacijo na Fourirovo transformacijo. Kar dobimo, je nazaj osnovna funkcija (graf 5).

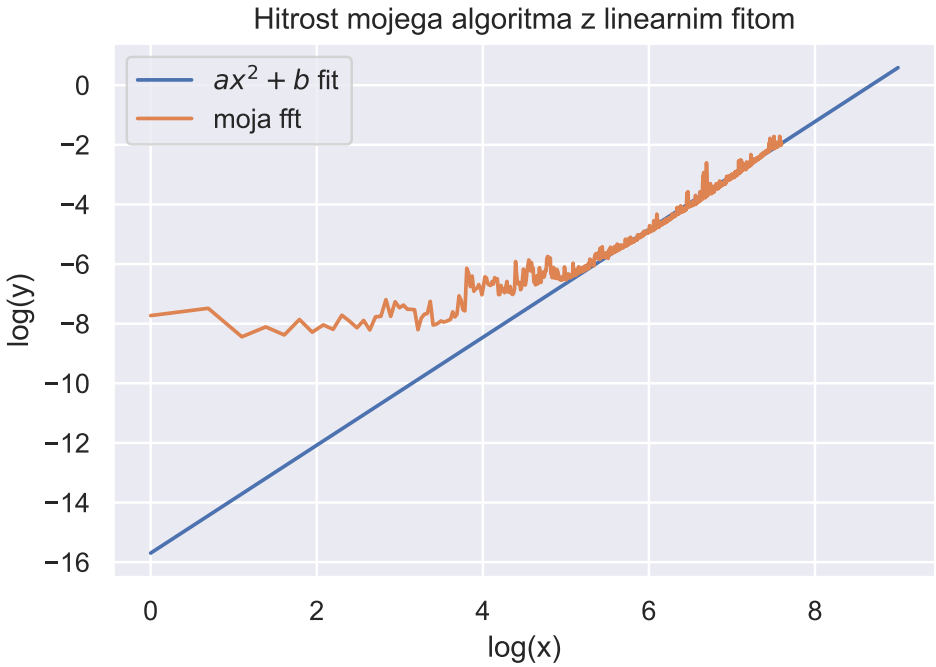


Slika 5: Graf in napaka inverzne Gaussove transformacije.

Opazimo, da je razlika med Gausovko in dvojnim Fourierov zares minimalna, kar potrjuje, da sem privilno napisal algoritme.

1.2 Časovna zahtevnost

Najlažje jo zmerimo (od neke točke naprej), da kar zmerimo, koliko časa je potrebno da se izračuna za nek N. To naredimo za vsak $N < M$, kjer sem izbral $M = 2000$, da se je program izvedel v nekem izvedljivem času.



Slika 6: Časovna zahtevnost z linearnim fitom. Koeficienta regresije sta $a = 1,81 \pm 0,01$ in $b = -15,7 \pm 0,4$.

S regresijo na grafu 6 opazimo, da je koeficient časovne zahtevnosti $x^\alpha = x^{1,81}$, $b = -15,7$. Prvih nekaj meritev ni najbolj relevantnih, ker ne gre za ozko grlo pri številu meritev, vendar je nek konstanten čas za procesiranje števil meritev. Verjetno bi bilo smiselno narisati še nekaj meritev več točk, vendar mislim, da moj računalnik tega žal nebi zmogel.

Opazimo lahko, da ima moj algoritem malce manj kot kvadratno časovno zahtevnost, kar bi lahko razbral že iz same kode. Da je zahtevnost zares skoraj kvadratna nam ponazarja že to, da pred končnim izračunom za največji N dobimo matriko $N \times N$, ki jo pomnožimo z vektorjem.

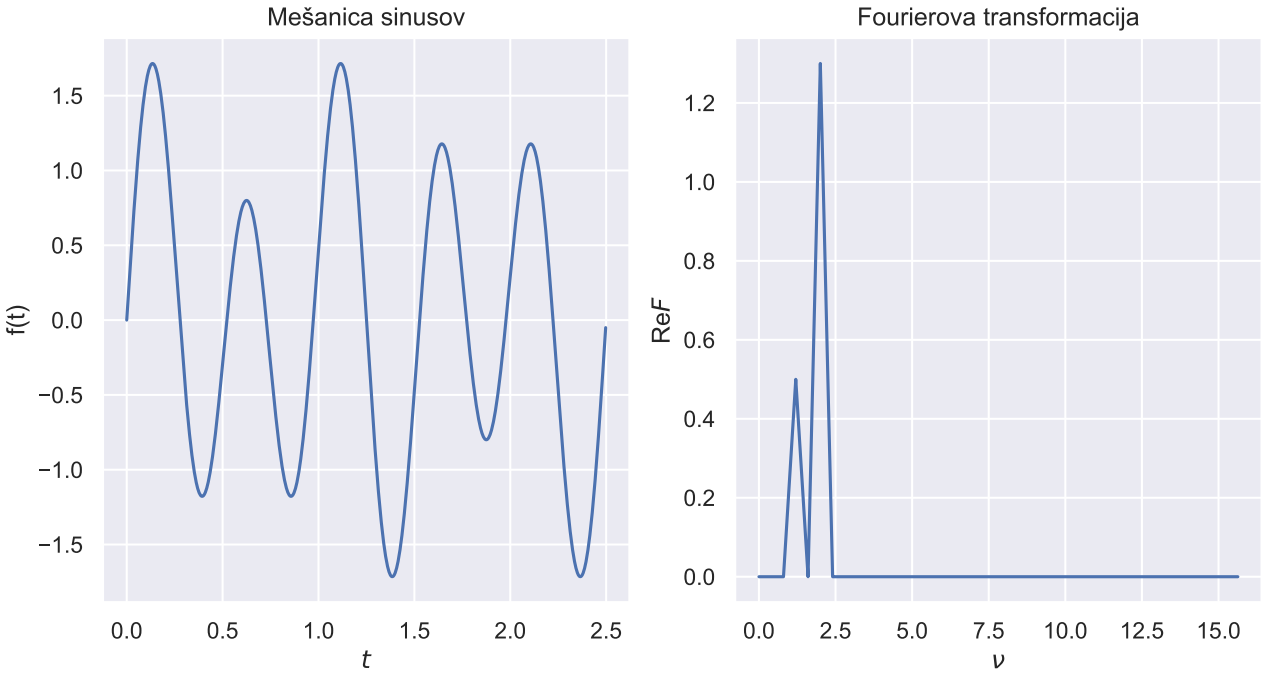
Zanimivo mi je tudi, da je algoritem `Scipy` tako zelo hitrejši, čeprav je moj v celoti zgrajen z `Numpy` vektorji in matrikami. To pomeni, da tudi s tako optimizacijo ni mogoče doseči optimizacijo C-ja.

1.3 Obdelava mešanice sinusnih signalov

Narisal sem funkcijo

$$f(t) = 1.3\sin(2\pi 2t) + 1.5\sin(2\pi 1.2t)$$

ter na njen naril Fourierovo analizo, da dobim pravilne frekvence. Dobimo sledeči graf

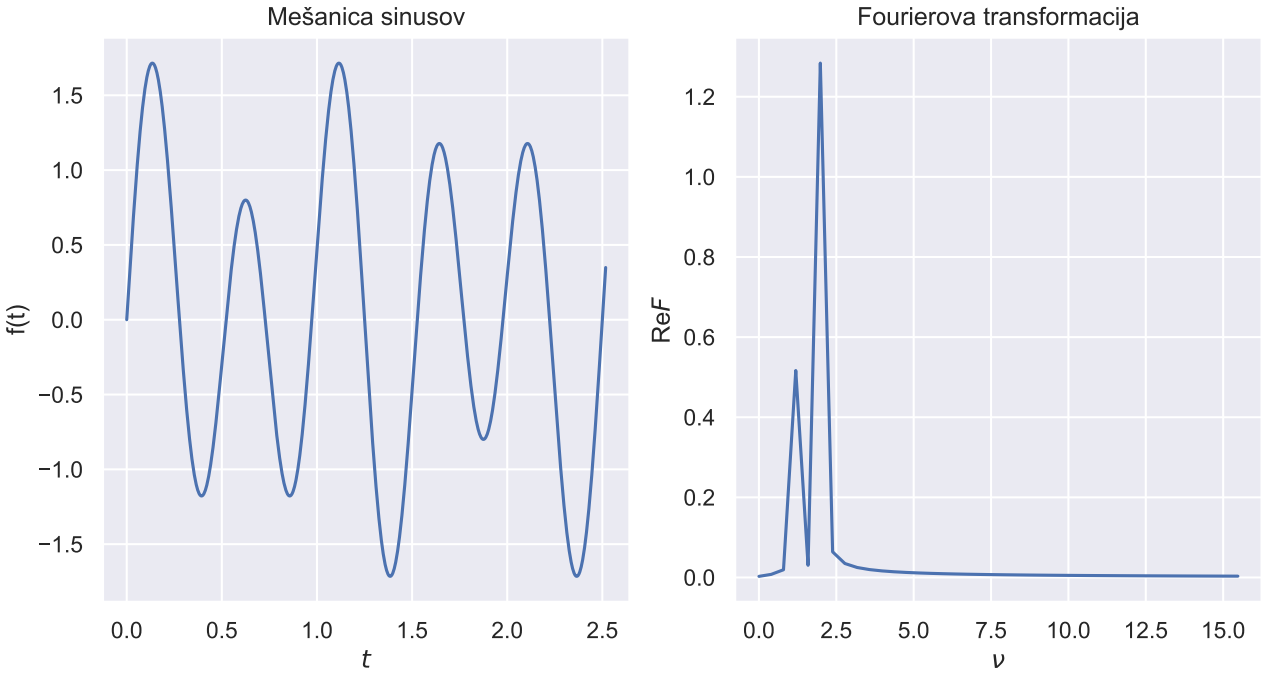


Slika 7: Fourierova analiza mešanice sinusnih signalov. Vidimo, da izračuna pravilni frekvenci.

Pomemben je tudi komentar, da bi bilo verjetno bolj smiselno v tem primeru narisati točke, ker so meritve zelo skupaj, vendar se potem na opazi dobro puščanja. Graf 7 je vbistvu nezvezna funkcija, ki skoči pri frekvencah v signalu.

Kaj pa se zgodi, če vklučimo še nekaj točk, ki vbistvu pripadajo naslednji frekvenci? (Puščanje)

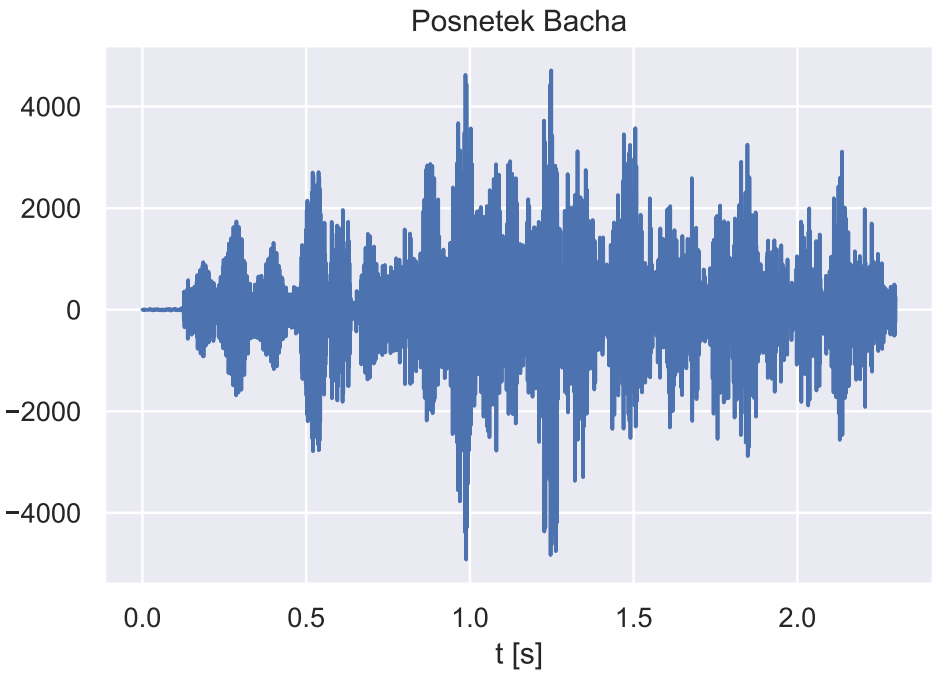
Zdaj lahko še enkrat isto funkcijo narišemo na malce večje definicijsko območje (iz 2,5 na 2,52; fino je malo pretirati, da se zares dobro opazi puščanje - enak pojav je tudi, če vključimo zgolj točko 2.5, vendar ni tako izrazit). Kar vidimo je, da je puščanje zares prisotno.



Slika 8: Prikaz puščanja.

2 Obdelava Bachovih posnetkov

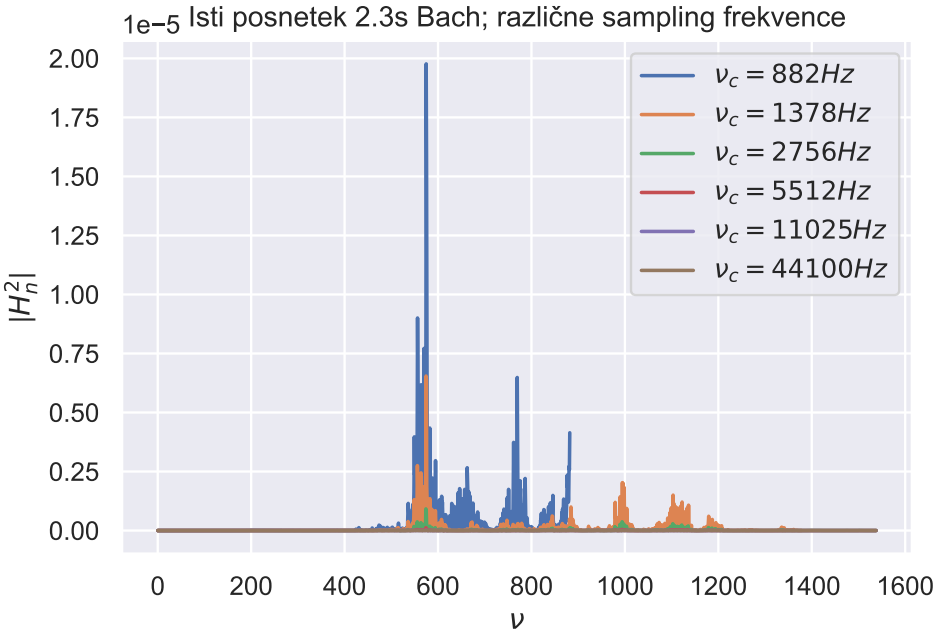
Najprej je priročno narisati kako zares zgleda 2,3s dolg posnetek.



Slika 9: 2,3s sekunde dolg posnetek zvoka Bachove sonate za violino.

Ne pove nam zelo veliko, ker je frekvenc ogromno, še več zelo skupaj so, kar pomeni, da zares vidimo zgolj območje. Vendar je lepa vizualizacija.

Lahko narišem vse Fourierove transformacija na isti graf.



Slika 10: 2,3s sekunde dolg posnetek zvoka Bachove sonate za violino obdelan z Fourierovo transformacijo.

Ugotovil sem, da je moja funkcija preveč počasna, da bi izračunal sampling pri 44kHz. Zato sem jo še enkrat prepisal s paralelno prijazno kodo. Postavil sem, da naenkrat uporablja vsa jedra v mojem računalniku. Tako je 2 minuti in pol Python uporabljal 100% CPU (vseh 8 jeder) in 100% RAMa. Brez te optimizacije se je koda poganjala 15 minut in še ni izračunal, kar je precej slabše. Še vedno pa je Scipy Fourier precej boljši, ker je za isti izračun porabil zgolj nekaj sekund, vendar seveda z C kodo pač ne moremo tekmovati s Pythonom.

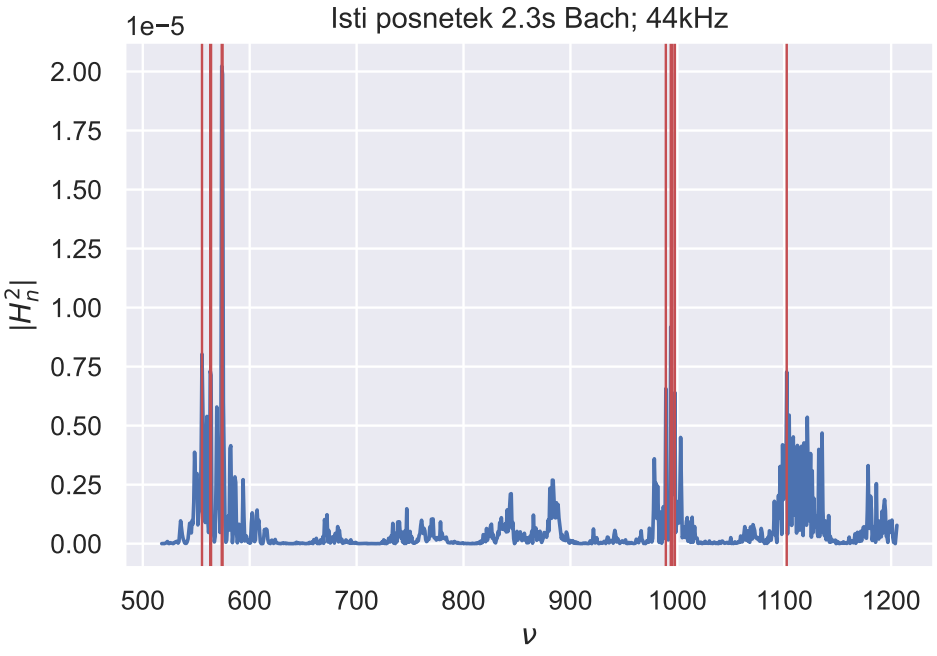
Na grafu je nekaj zanimivih razlik med različnimi sampli. Seveda se frekvenčno območje poveča pri večjih samplih (po definiciji) vendar sem vse narisal na območje zgoraj omejeno na 2000 Hz. Tako je območje precej bolj pregledno (če narišem 44kHz se vidi samo največji sample). Pri večjih ν_c so posamezne bolj ostre in nižje (ker je frekvenčno območje večje). Ta razlika je zelo očitna na mp3 posnetkih. Pri nižjih samplih od 44kHz se praktično ne sliši da gre za violino, ker je zvok popolnoma pridušen.

Zanimivih je tudi 10 najbolj pogostih frekvenc, ki so od najmanjše do največje 555Hz, 563Hz, 563Hz, 573Hz, 574Hz, 989Hz, 993Hz, 995Hz, 997Hz, 1102Hz.

Frekvence so seveda odvisne od tega kako oglasimo violino, vendar se lahko vseeno omejimo na območje C5 (523 Hz) do E6 (1318 Hz).

Najbolj pogosti toni so (po vrsti), ki jih lahko narišemo tudi na graf 11:

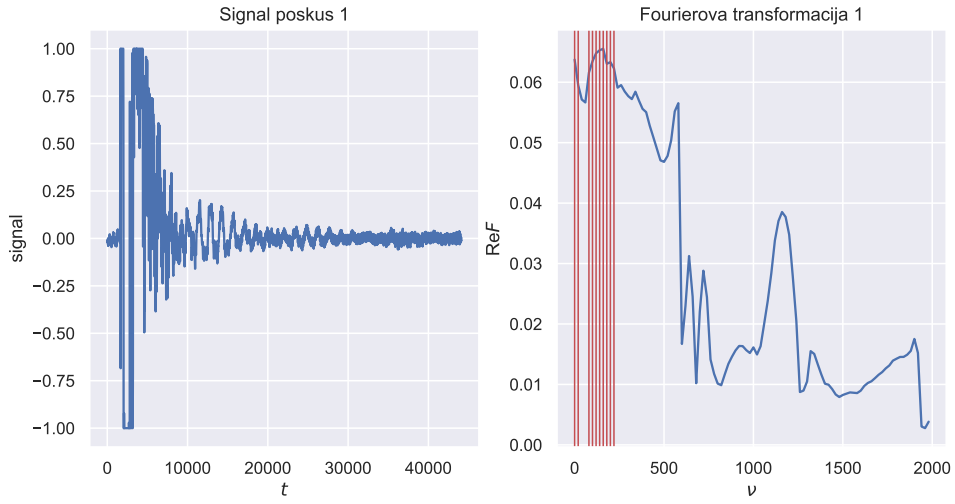
1. C#5
2. D5
3. B5
4. C#6



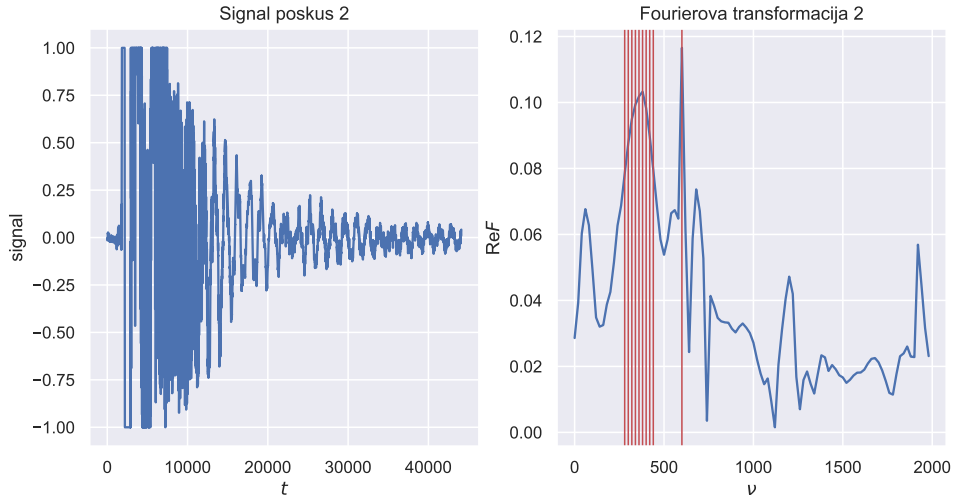
Slika 11: Graf Fourierove transformacije z narisanimi frekvencami pri 555Hz, 563Hz, 563Hz,573Hz, 574Hz, 989Hz, 993Hz, 995Hz, 997Hz, 1102Hz.

3 Dodatna naloga - akusticni resonator analiza

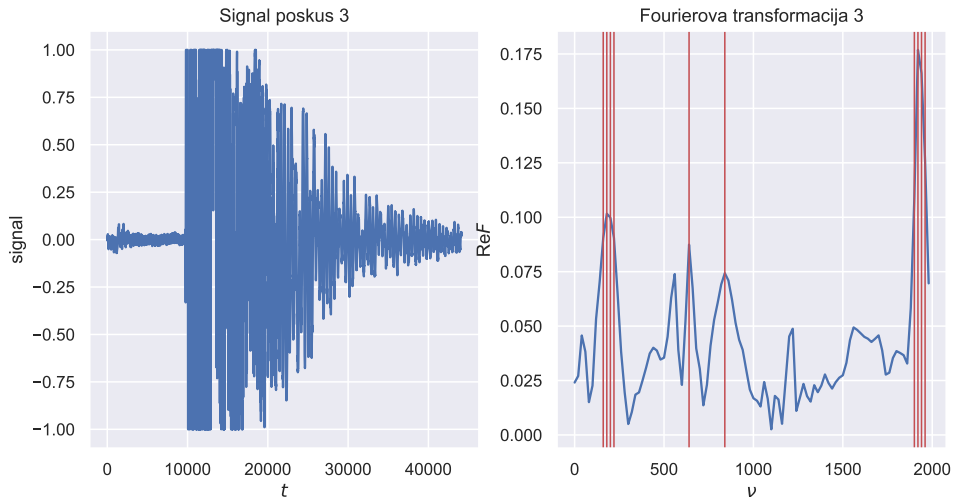
Lahko analiziramo grafe, ki jih dobimo iz akusticnega resonatorja. Narisal bom grafe ter njihove Fourierove transformacije z najbolj pogostimi frekvencami.



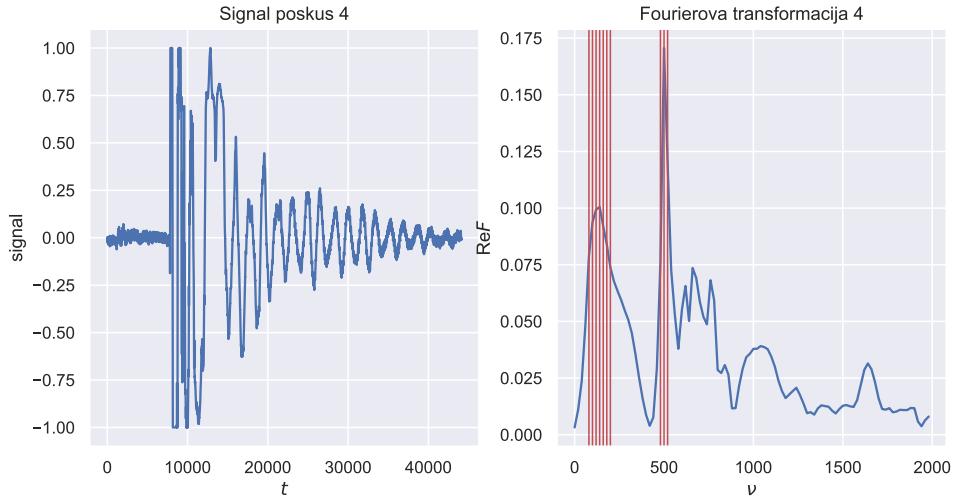
Slika 12: Graf 1. poskusa transformacije z narisanimi frekvencami pri 0Hz, 20Hz, 80Hz, 100Hz, 120Hz, 140Hz, 160Hz, 180Hz, 200Hz, 220Hz.



Slika 13: Graf 2. poskusa transformacije z narisanimi frekvencami pri 280Hz, 300Hz, 320Hz, 340Hz, 360Hz, 380Hz, 400Hz, 420Hz, 440Hz, 600Hz.



Slika 14: Graf 3. poskusa transformacije z narisanimi frekvencami pri 160Hz, 180Hz, 200Hz, 220Hz, 640Hz, 840Hz, 1900Hz, 1920Hz, 1940Hz, 1960Hz.



Slika 15: Graf 4. poskusa transformacije z narisanimi frekvencami pri 80Hz, 100Hz, 120Hz, 140Hz, 160Hz, 180Hz, 200Hz, 480Hz, 500Hz, 520Hz.

4 Komentar

Fourierovo transformacijo sem očitno pravilno napisal, ker popolnoma sovpada z grafom, ki jo poda vgrajena funkcija *Scipy*. Tudi napaka obratne Fourierove transformacije je zelo majhna.