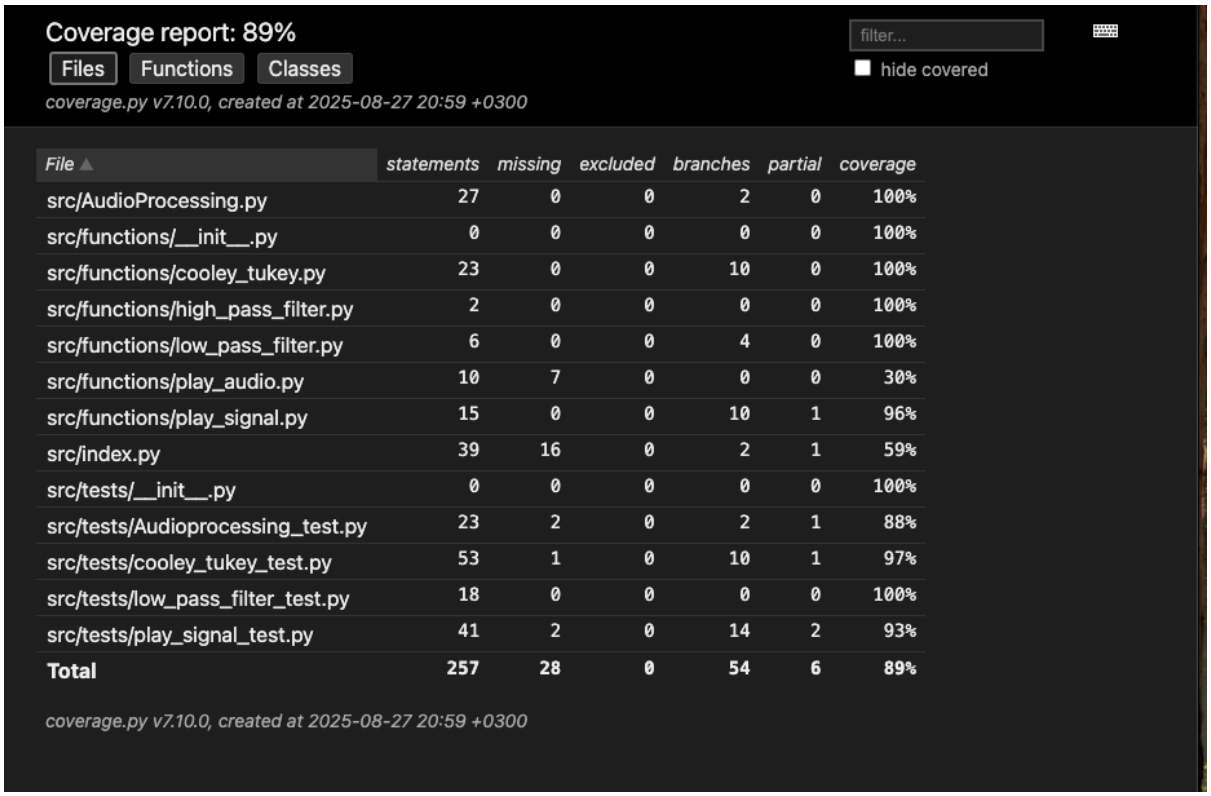
Testausdokumentti

Yksikkötestaus

Automaattiset yksikkötestit on kirjoitettu koodin varsinaista ohjelmalogiikkaa sisältäville tiedostoille ja yksi niiden tavoitteista on mahdollisimman hyvä koodirivien kattavuus. Yksikkötestauksessa käytetään test_song.wav-tiedostoa tai testien nopeuttamiseksi siitä lyhennettyä versiota, ellei muuta ole mainittu. Testien pitäisi olla toistettavissa millä tahansa kappaleella, tosin pidemmällä kappaleella niihin luultavasti menee aikansa.

Yksikkötestien kattavuusraportti



Coverage report: 89%

Files | Functions | Classes

filter... | hide covered

coverage.py v7.10.0, created at 2025-08-27 20:59 +0300

File	statements	missing	excluded	branches	partial	coverage
src/AudioProcessing.py	27	0	0	2	0	100%
src/functions/__init__.py	0	0	0	0	0	100%
src/functions/cooley_tukey.py	23	0	0	10	0	100%
src/functions/high_pass_filter.py	2	0	0	0	0	100%
src/functions/low_pass_filter.py	6	0	0	4	0	100%
src/functions/play_audio.py	10	7	0	0	0	30%
src/functions/play_signal.py	15	0	0	10	1	96%
src/index.py	39	16	0	2	1	59%
src/tests/__init__.py	0	0	0	0	0	100%
src/tests/Audioprocessing_test.py	23	2	0	2	1	88%
src/tests/cooley_tukey_test.py	53	1	0	10	1	97%
src/tests/low_pass_filter_test.py	18	0	0	0	0	100%
src/tests/play_signal_test.py	41	2	0	14	2	93%
Total	**257**	**28**	**0**	**54**	**6**	**89%**

coverage.py v7.10.0, created at 2025-08-27 20:59 +0300

Mitä on testattu?

Audioprocessing_test.py

Tässä tiedostossa testataan, että Audioprocessing luokka luo olion ja käyttää siihen audiotiedostoa ‘testsong.wav’. Testit testaavat myös, että äänitiedosto saadaa muutettua mono-tiedostoksi, jotta se noudattaa yksiulotteista taulukko-rakennetta (array). Ohjelmani on rakennettu siten, että taulukot ovat aina yksiulotteisia.

Cooley_tukey_test.py

Tässä tiedostossa testataan koko ohjelman pääalgoritmia, nopeaa Fourier muunnosta “Cooley Tukey”. Ensimmäisessä testissä (def test_that_if_recursion_is_done_and_only_one_sample_is_left) testaan, että rekursion saavuttaessa päätöspisteensä, jäljellä on enää vain yksi näyte. Tällöin voidaan palauttaa lopputulos.

Algoritmi perustuu siihen, että näytteiden lukumäärä on aina jokin luku potenssiin kaksi. Tämä on tärkeää siksi, että tällöin kaikki näytteet voidaan jakaa tehokkaasti kahteen “butterfly” metodilla ja algoritmin tehokkuus paranee. Niinpä testaan, että kun syötteen pituus on jokin toisen potenssi, pituus pysyy muuttumattomana. Testaan myös, että kun se ei ole, syötteeseen lisätään tarpeeksi nollia. Tämä testataan seitsemällä numerolla, jotta testien suorittamiseen ei menisi liikaa aikaa.

Tiedoston toiseksi viimeinen testi (test_if_can_reconstruct_original_signal) testaa sitä, että mikäli Fourier muunnos toimii oikein, normaali cooley tukey muuttaa ensin signaalin taajustasoon ja sitten käänteinen versio siitä muuttaa sen takaisin aikatasoon eli lopputulos pitäisi olla hyvin lähellä alkuperäistä taulukkoa. Taulukolle on annettu desimaalilukuja, jotta ne olisivat edustavampia.

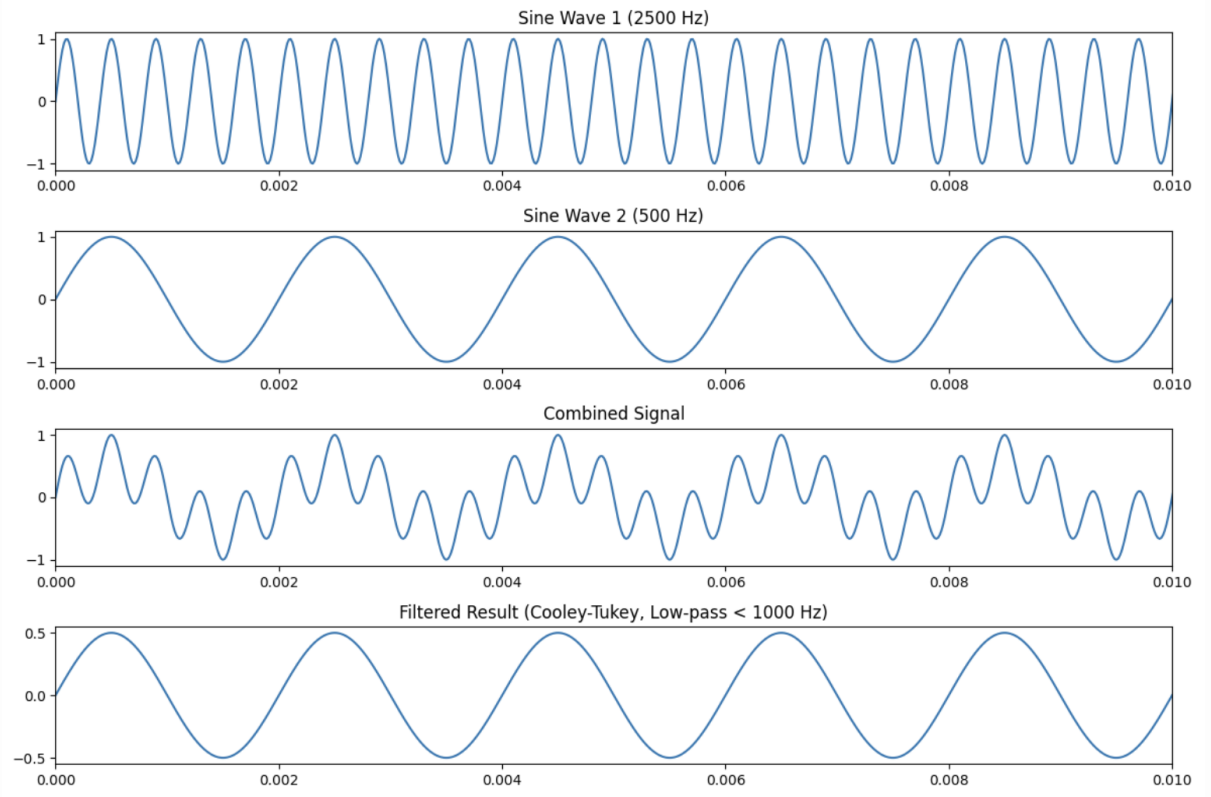
Low_pass_filter_test.py

Tämä tiedosto testaa, että low_pass_filter_ filtteröi signaalia. Tiedostossa testataan myös, että matalat signaalitaajuudet pysyvät muuttumattomina suodattimen jäljiltä. Se poimii taajuuksista ne, jotka ovat pienempiä kuin asetettu taajuuksien raja. Sitten se tutkii, löytyvätkö nämä taajuudet yhä signaalista.

End-to-End testi

Cooley_tukey_test.py-tiedoston viimeinen testi “test_by_removing_one_sine_wave” lähestyy end-to-end testimäisyyttä, sillä testaa tilannetta, jossa algoritmi suodattaa kahdesta signaalista toisen kokonaan pois. Testissä luodaan kaksi sini-muotoista signaalia, joista toinen on matala ja toinen korkea. Sitten testi lisää signaalit toisiinsa ja varmistaa, että ne on skaalattu oikein.

Tämän jälkeen testataan ohjelman toimintaa luomalla Audiprocessing luokan olio. On tärkeää poistaa lopuksi mahdolliset testin loppuun lisätyt nollat, sillä muutoin jäljelle jäävä signaali ei ole pituudeltaan sama, kuin testissä käytetty matalampi signaali.
Kuva “Figure_1_cooley_tukey_test.png”



Esittää, miltä näyttävät siniaalto 1 ja 2 erikseen ja yhdessä. Neljännellä rivillä voidaan nähdä, että jäljelle jäävä siniaalto on identtinen matalamman siniaallon kanssa.

Neljännessä kohdassa aallon amplitudi on pienempi, sillä siniaallot skaalataan, jotta amplitudi ei kasvaisi yhteenlaskun seurauksena: Ensimmäisessä ja toisessa kohdassa amplitudit ovat 1. Kun nämä summataan yhteen, jää amplitudiksi 2, joista puolikas on siniaalto 1. Skaalauksen seurauksena siniaaltojen yhteinen amplitudi on 1. Kun toinen aalto poistetaan, jää jäljelle (0,5 – 0,5).