Helsingin yliopisto
Matemaattis-luonnontieteellinen tiedekunta
Matematiikan ja tilastotieteen osasto

Pro gradu -tutkielma

Nopea Fourier-muunnos – teoria ja toteutus modernilla C++:lla

Joel Yliluoma

Tutkielma jätettiin hyväksyttäväksi 8.1.2024, ja hyväksyttiin 14.3.2024 arvosanalla 5/5.
Tämän jälkeen sisältöön on tehty kuitenkin vielä pieniä korjauksia.
Viimeksi päivitetty: 1.6.2024
Alkuperäinen, hyväksyttäväksi jätetty versio löytyy Heldasta
(`http://hdl.handle.net/10138/573505`).

Matematiikan, fysiikan ja kemian opettajan maisteriohjelma
Ohjaaja: Professori Lauri Oksanen 4. tammikuuta 2024

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Matemaattis-luonnontieteellinen	Matematiikan, fysiikan ja kemian aineenopettajan maisteriohjelma

Tekijä — Författare — Author
Joel Yliluoma

Työn nimi — Arbetets titel — Title

Nopea Fourier-muunnos – teoria ja toteutus modernilla C++:lla

Opintosuunta — Studieinrikting — Study track
Matematiikka ja tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Pro gradu -tutkielma	Tammikuu 2024	67 s. + 5 liitettä

Tiivistelmä — Referat — Abstract

Tässä tutkielmassa selvitetään, kuinka voidaan toteuttaa tietokoneohjelmana sellainen nopea yksiulotteinen Fourier-muunnos (FFT), joka toimii nopeasti kaiken kokoisilla syötteillä. Tutkielman alussa käydään läpi matemaattista teoriaa sekä perusmuotoisesta että diskreetistä Fourier-muunnoksesta (DFT), ja selvitetään, mikä niiden päämäärä on. Diskreetistä Fourier-muunnoksesta käsitellään sekä yksiulotteinen että moniulotteinen versio. Myös diskreettiä kosinimuunnosta (DCT) käsitellään ja tutkitaan sen eroja Fourier-muunnokseen.

Tämän jälkeen tutkitaan erilaisia tunnettuja menetelmiä toteuttaa diskreetti Fourier-muunnos niin, että se toimii nopeasti. Erityisesti Cooleyn-Tukeyn menetelmää ja sen eri muotoja tutkitaan sellaisessa tarkkuudessa, että sen toimintaperiaatteen sekä sen syntyyn johtaneen ajatusketjun voi ymmärtää. Lisäksi käsitellään Raderin sekä Bluesteinin FFT-menetelmiä.

Teorian käsittelyn jälkeen DFT sekä esitetyt FFT-algoritmit toteutetaan C++-ohjelmointikielellä. Lisäksi laaditaan ja toteutetaan yhdistetty algoritmi, joka pyrkii tapauskohtaisesti valitsemaan optimaalisen yhdistelmän algoritmeja saavuttaakseen nopeimman mahdollisen muunnoksen. C++-kielestä käytetään kirjoitushetkellä tuoreinta standardiversiota, C++20. Toteutettujen menetelmien nopeutta tutkitaan ja verrataan suhteessa toisiinsa sekä tunnettuun FFT-kirjastoon, FFTW. Lisäksi tutkitaan menetelmien laskentatarkkuutta. Lopuksi analysoidaan vertailumenetelmien tarkkuutta ja pätevyyttä, ja pohditaan toteutuksen puutteita sekä mahdollisia parannuskeinoja. Tutkielmassa esitetään myös FFT-muunnoksesta esimerkkisovellus, joka pyrkii selvittämään puheääninäytteestä sen sisältämät vokaaliäänteet.

Tutkielman päämääränä on tarjota lukijalle selkeä esimerkkitoteutus kaikista esitetyistä muunnoksista sekä käsitellä kattavasti niiden rajoituksia, heikkouksia ja vahvuuksia. Parhaan hyödyn saamiseksi teoriasta lukijan tulisi ymmärtää differentiaali- ja integraalilaskennan perusteet sekä kompleksilukujen perusteet. Erityisesti summamerkinnän, $\sum$, sekä Eulerin lauseen ymmärtäminen on tärkeää matemaattisen teorian kannalta. Tietokonetoteutuksen lähdekoodi on laadittu sellaiseksi, ettei sen ymmärtämiksi tarvitse tuntea C++20-standardin yksityiskohtia, tai edes C++-kieltä kunnolla. Kokemus C-ohjelmoinnista on kuitenkin hyödyksi koodia luettaessa.

Avainsanat — Nyckelord — Keywords
nopea Fourier-muunnos, C++20, Cooleyn-Tukeyn algoritmi, Raderin algoritmi, Bluesteinin algoritmi, C++, optimointi, fftw, fft, dft, dct, pienimmän neliösumman menetelmä, Bluestein, Cooley-Tukey, Rader, fftw3

Säilytyspaikka — Förvaringsställe — Where deposited
Helsingin yliopiston kirjasto / HELDA

Muita tietoja — Övriga uppgifter — Additional information

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Faculty of Science	Master's Programme for Teachers of Mathematics, Physics and Chemistry

Tekijä — Författare — Author
Joel Yliluoma

Työn nimi — Arbetets titel — Title

Fast Fourier Transform – Theory and Implementation in Modern C++

Opintosuunta — Studieinrikting — Study track
Mathematics and computer science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	January 2024	67 pp. + 5 appendices

Tiivistelmä — Referat — Abstract

In this thesis we study how to implement the one-dimensional fast Fourier transform (FFT) as a computer program in such a way that it works fast on inputs of all sizes. First we walk through the mathematical theory behind the continuous and the discrete Fourier transform (DFT), and study the motivation behind them. We explore both the one-dimensional and the multi-dimensional versions of the DFT. We also study the discrete cosine transform (DCT) and explore its differences to the Fourier transform.

After that, we explore various known methods for implementing the fast Fourier transform. We focus especially on the Cooley-Tukey algorithm and its different forms in enough detail, that one can understand both how it works and how it was conceived. We also explore Rader's and Bluestein's FFT algorithms.

After the theory we implement the DFT and the FFTs using the C++ programming language. We also devise and implement a combined algorithm, which seeks to provide the fastest possible transformation by choosing the optimal combination of algorithms in each case. We use C++20, which is the most recent standard version of C++ at the time of writing. We analyze the speed of the implemented methods, comparing them against each others and also compared to a famous FFT library, FFTW. In addition, we analyze the numeric accuracy of the implementations. Then we analyze the accuracy and applicability of the analysis methods and study the shortcomings and possible approaches for improving the implementations. We also present a simple example application of FFT, where the program attempts to identify the vowel sounds present in a voice sample.

The purpose of this thesis is to offer the reader a clear example implementation of all the presented transforms and to thoroughly explore their limitations, weaknesses and strengths. For best value, the reader is expected to understand the basics of calculus and the basics of complex numbers. Especially the sum notation, $\sum$, and Euler's formula are vital for understanding the mathematical theory. The computer source code is designed in such way that the reader does not need to know the details of the C++20 standard, or indeed even the C++ language, but experience in C programming will help.

Avainsanat — Nyckelord — Keywords
fast Fourier transform, C++20, Cooley-Tukey algorithm, Rader's algorithm, Bluestein's algorithm, C++, optimization, fftw, fft, dft, dct, sum of least squares method, bluestein, cooley-tukey, rader, fftw3

Säilytyspaikka — Förvaringsställe — Where deposited
Helsinki university library / HELDA

Muita tietoja — Övriga uppgifter — Additional information

Esipuhe

Tätä tutkielmaa aloitin kirjoittamaan siinä vaiheessa, kun oli kulunut vain vähän yli kaksi vuotta siitä, kun aloitin opintoni Helsingin yliopistossa matematiikan, fysiikan ja kemian aineenopettajan kandi- ja maisteriohjelmassa. 20 kalenterikuukaudessa olin saanut valmiiksi kanditutkintoni aineina matematiikka ja tietojenkäsittelytiede, ja tähtäsin saada maisteritutkinnon valmiiksi myös kolmantena vuotena. Ohessa olin suorittanut myös kosolti opintoja, jotka eivät ollenkaan kuuluneet tutkintooni: fysiikkaa, kemiaa, psykologiaa, filosofiaa, jopa teologiaa; tähtäimessä vastaavuustodistus useimmista näistä aineista. Ja tietysti myös tutkintoon kuuluvat opettajan pedagogiset opinnot opetusharjoitteluineen, joita tein päällekkäin mm. kemian laboratoriotöiden kanssa. Taustalla painoi suuri huoli siitä, miten voin jatkaa tutkinnon ulkopuolisia opintojani, jos valmistun. Menetänkö opinto-oikeuteni? Olin päässyt opintojen makuun: Tätä kirjoittaessani valmiina on 498 opintopistettä; kandin+maisterin tutkintoon tarvitaan yhteensä 300. En haluaisi lopettaa! Toisaalta vaakakupissa painoi jonkinlainen ironian ja ylpeyden sekoitus: Olisihan se hienoa vain kansakoulun käyneiden vanhempien poikana nousta ammattikoulusta, 1990-luvulla ansaitusta tietotekniikan mekaniikon tutkinnosta, maisteriksi yhtäkkiä alle kolmessa vuodessa näin keski-ikäisenä käymättä koskaan lukiota.

Löysin itselleni sopivan aiheen professori Lauri Oksasen laatimasta mielenkiintoisesta listasta, ja otin häneen yhteyttä. Professori oli avulias alkuunpääsemisen kanssa, ja löysin inspiraation heti. Kirjoittamaan aloin. Muutamassa viikossa syntyi materiaalia paljonkin. Ammattitaito ja -kokemus auttoi. Välillä piti pitää taukoa muiden kiireiden takia. Lähetin väliversioita; kommentteja tuli niukalti ja harvoin, enemmän vasta loppuvaiheessa. Ei haitannut, koska kuviot olivat selvät: tiesin, mitä teen; visio oli mielessäni, ja projekti ruokki itse itseään. Vähäkin ohjeistus auttoi korjaamaan huonoja valintoja ja arvioimaan tekemistäni. Yhteistyö oli erilaista kuin kandityöni kanssa, jolloin ohjaajanani toimi yliopistonlehtori Petteri Harjulehto. Kumpikin kuitenkin erittäin asiantuntevia ja osaavia.

Opintoni alkoivat koronapandemian vuosina, syksyllä 2021. Toisille etäopiskelu oli kirous, monille opettajille täysin kohtuuttoman raskasta; itselleni se oli kuitenkin mitä suurin siunaus. Tuolloin ei tunnettu läsnäolopakkoja. Uskon, ettei kuuna päivänä olisi onnistunut tuolloinen opiskelutahtini, mikäli opinnot olisi pitänyt suorittaa lähiopintoina. Ei puhettakaan, ei murto-osakaan siitä. Siksi paluu lähiopintoihin on tuntunut valtavalta regressiolta yliopistokoulutusjärjestelmässä. Onneksi sain ensimmäisenä vuonna hyödynnettyä mahdollisuuden täysimääräisesti, ja opin menetelmät, joilla sain myös myöhemmät opintoni järjesteltyä etäopiskelupainotteisesti. Onni onnettomuudessa, joku sanoisi.

Olen kiitollinen ohjaajalleni professori Lauri Oksaselle tuesta ja opastuksesta tätä tutkielmaa koskien. Olen myös kiitollinen yliopistonlehtori Antti Laaksoselle, joka on mahdollistanut sen, että opintojeni (ja töiden) ohessa olen saanut tehtyä (sivu)töikseni sitä, mistä nautin: oivallusten mahdollistamista muille opiskelijoille. Haluan myös kiittää ystävääni, päätoimista tuntiopettajaa Riikka Rantaa, joka on toiminut minulle ikkunana perusopetuksen maailmaan sekä hyvässä että pahassa, sekä myös vuosikausia rohkaissut opinnoissani ja muutenkin ollut tukenani. Ja Aabrahamin, Iisakin ja Jaakobin Jumalaa, joka on aina ohjannut askeleen eteenpäin, vaikka ei usein yhtään sen enempää; sekä hänen palvelijaansa, edesmennyttä Ulla-Maija Alasvuota זכרונה לברכה, joka alati valmensi minua astumaan rohkeammin esiin.

Mutta opettajaluonteinen ihminen kun olen, siksi tästäkin tutkielmasta olen halunnut tehdä sellaisen, että siitä on lukijalleen hyötyä. Toivon, että kirjoittamani lähdekoodi sekä opettaa että inspiroi, ja että selitykseni avartavat ymmärrystä käsitellyistä matemaattisista menetelmistä. Paljon enemmänkin olisin voinut tehdä, mutta työmäärällisesti raja piti vetää johonkin, jotta valmista tulisi joskus. Jos luet tätä, ehkä tapaamme jossain!

Joel Yliluoma Keravalla 6.12.2023

Sisällys

1. Johdanto Fourier-muunnokseen

Tässä tutkielmassa käsitellään Fourier-muunnosta ja erityisesti nopeaa Fourier-muunnosta (FFT). Ensin käydään hieman matemaattista teoriaa läpi, ja sen jälkeen toteutetaan erilaisia FFT-algoritmeja C++-ohjelmakoodina, ja tutkitaan niiden ominaisuuksia.

Signaalinkäsittelyssä usein mielenkiintoinen tarkastelun aihe on, minkä taajuista tarkasteltava tuntematon signaali on, eli kuinka tiheästi siinä esiintyy jokin piirre. Esitetään ajatuskokeena erikoistapaus, jossa tarkasteltavaa signaalia voidaan kuvata funktiolla $f(t) = \sin(\omega t)$, jossa siniaallon kulmataajuus ω on tuntematon. Funktio $f(t)$ kuvaa signaalin hetkellistä amplitudia ajanhetkellä t.

Esitetään teoria: Tuntemattoman parametrin ω arvo voidaan määrittää etsimällä sellainen x, jolla funktion $f(t)$ sekä koefunktion $\sin(xt)$ tulo keskiarvoistettuna jonkin (suuren) arvoalueen yli saa suurimman arvonsa[1]. Keskiarvoistaminen lasketaan integroimalla. Merkitään koefunktiota $\sin(xt)$, ja keskiarvoistettavaa väliä $[-m, m], m \in \mathbb{R}, m > 0$. Silloin integraalin arvoksi saadaan:

$$\begin{aligned} F(x) &= \int_{-m}^{m} \frac{1}{m} f(t)\sin(xt)\,\mathrm{d}t && \| f(t) = \sin(\omega t) \\ &= \frac{1}{2m} \int\!\!\!\!\!\!\diagup_{-m}^{m} \left(\frac{\sin[t(\omega - x)]}{\omega - x} - \frac{\sin[t(\omega + x)]}{\omega + x} \right) \\ &= \frac{\sin[m(\omega - x)]}{m(\omega - x)} - \frac{\sin[m(\omega + x)]}{m(\omega + x)} \end{aligned} \tag{1.1}$$

Toiston vähentämiseksi tässä luvussa oletetaan jatkossa aina, että keskiarvoistusväli m lähestyy ääretöntä, ellei muuta mainita. Tutkitaan seuraavaksi funktion $F(x)$ raja-arvoa sellaisessa tilanteessa, jossa pätee sekä $\omega - x \neq 0$ että $\omega + x \neq 0$. Saadaan:

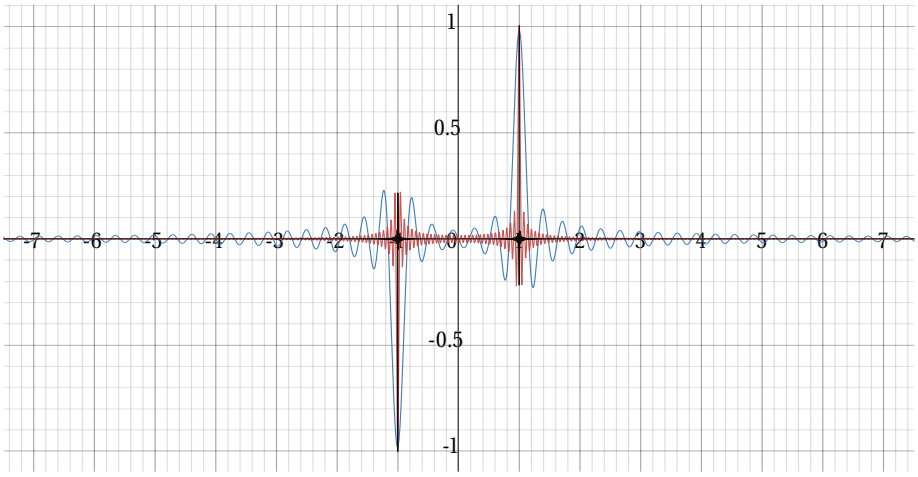
$$\begin{aligned} \lim_{m\to\infty} F(x) &= \lim_{m\to\infty} \left(\frac{\sin[m(\omega - x)]}{m(\omega - x)} - \frac{\sin[m(\omega + x)]}{m(\omega + x)} \right) \\ &= 0. \end{aligned}$$

Tilanne, jossa pätee sekä $\omega - x \neq 0$ että $\omega + x \neq 0$ tarkoittaa, että $|x| \neq |\omega|$. Entä jos $|x| = |\omega|$? Tämä tarkoittaa, että $x = \pm\omega$. Tutkitaan siis seuraavaksi raja-arvoa, jossa $x = \pm\omega$. Saadaan:

$$\begin{aligned} \lim_{m\to\infty} \lim_{x\to\pm\omega} F(x) &= \lim_{m\to\infty} \lim_{x\to\pm\omega} \left(\frac{\sin[m(\omega - x)]}{m(\omega - x)} - \frac{\sin[m(\omega + x)]}{m(\omega + x)} \right) \\ &= \lim_{m\to\infty} \pm \left(1 - \frac{\sin 2\omega m}{2\omega m} \right) \\ &= \pm 1. \end{aligned}$$

[1] Teorian pohja on fysikaalisessa resonanssi-ilmiössä: Fyysisen kappaleen, esimerkiksi ääniraudan, ominainen resonanssitaajuus voidaan määrittää saattamalla se kosketuksiin eritaajuisten värähtelyjen kanssa väliaineen, kuten ilman, kautta. Mitä paremmin värähtelyn taajuus täsmää resonanssitaajuuteen, sitä vahvemmin tutkittava kappalekin alkaa värähdellä.

Mikäli funktiosta $F(x)$ piirretään kuvaaja (kuvaaja 1), voi samat ilmiöt havaita myös graafisesti: Silloin, kun $x = \pm\omega$, lähestyy $F(x)$ arvoja ± 1, mutta muualla funktion arvo lähestyy arvoa 0; sitä jyrkemmin, mitä suurempi m on.



Kuva 1: Kaavassa (1.1) esitetyn funktion $F(x)$ kuvaaja arvolla $\omega = 1$ sekä $m = 20$ (sininen), $m = 100$ (ruskea) ja $m = 2000$ (musta).

Koska silloin, kun $x \neq \pm\omega$, funktion $F(x)$ arvo lähestyy nollaa, mutta pisteissä $x = \pm\omega$ $F(x)$ lähestyy arvoja ± 1, voidaan päätellä, että viimeksi mainitut arvot ovat funktion minimi ja maksimi, sillä muita vaihtoehtoja ei ole.

Funktion $F(x)$ arvo siis lähestyy maksimiaan 1 silloin, kun x lähestyy arvoa ω. On siis alustavasti päätelty, että tietyntaajuisella siniaallolla kertominen ja saadun tulon keskiarvoistaminen suuren arvoalueen yli on hyvä mittari näiden kahden siniaallon taajuuksien korrelaatiolle.

Toisaalta funktion $F(x)$ arvo lähestyy *minimiään* -1 silloin, kun x lähestyy arvoa $-\omega$. Tapaus $x = -\omega$ tarkoittaa, että integroitava lauseke olisi $\sin(\omega t)\sin(-\omega t)$. Koska $\sin(-\omega t) = \sin(\omega t + \pi)$, tämä tarkoittaa tilannetta, jossa tutkittavalla siniaallolla ja koefunktion siniaallolla on π radiaanin vaihe-ero. Vaihe-erolla on siis vaikutus mittarin toimintaan. On syytä tutkia tätä seikkaa tarkemmin.

Kartoitetaan seuraavaksi eri vaihe-erojen vaikutusta integrointiin puolen radiaanin välein, kun $x = \omega = 1$:

$$\begin{aligned}
\int_{-m}^{m} \frac{1}{m} \sin t \sin\left(t + \tfrac{0}{2}\pi\right)\, \mathrm{d}t &= \frac{2m - \sin 2m}{2m} \\
\int_{-m}^{m} \frac{1}{m} \sin t \sin\left(t + \tfrac{1}{2}\pi\right)\, \mathrm{d}t &= 0 \\
\int_{-m}^{m} \frac{1}{m} \sin t \sin\left(t + \tfrac{2}{2}\pi\right)\, \mathrm{d}t &= \frac{\sin 2m - 2m}{2m} \\
\int_{-m}^{m} \frac{1}{m} \sin t \sin\left(t + \tfrac{3}{2}\pi\right)\, \mathrm{d}t &= 0
\end{aligned}$$

Kun tutkittavan siniaallon ja koefunktion siniaallon vaiheet eroavat toisistaan $\frac{\pi}{2}$ radiaanin verran, integraalin arvoksi tuleekin 0! Vaikka taajuus on oikein, laskennan tulos antaa ymmärtää, etteivät taajuudet täsmää ollenkaan. Näyttäisi siltä, että integroimismenetelmä ei sittenkään sovellu taajuuden määrittämiseen.

Kun kuitenkin muistetaan, että $\sin\left(x + \frac{\pi}{2}\right) = \cos x$, sekä lisäksi $\sin^2 x + \cos^2 x = 1$ mille tahansa mielivaltaiselle arvolle x, voidaan päätellä, että vaihe-eron vaikutus saataisiin eliminoitua huomioimalla laskennassa sekä sini että kosini. Tämä onnistuu kätevästi kompleksiluvuilla hyödyntämällä Eulerin lausetta:

$$\cos(x) + i\sin(x) = e^{ix},$$

jossa i viittaa imaginäärilukuyksikköön, jolle pätee $i^2 = -1$. Tällöin saadaan hieman muunnettu versio edellä esitetystä integraalista. Otetaan käyttöön muuttuja φ kuvaamaan jotain mielivaltaista vaihe-eroa:

$$\begin{aligned} F(x) &= \int_{-m}^{m} \frac{1}{m} \sin(t) e^{i(t+\varphi)} \,\mathrm{d}t \\ &= ie^{i\varphi} \left(1 - \frac{\sin m \cos m}{m}\right). \end{aligned}$$

Integraalin raja-arvo, kun m lähestyy ääretöntä, on:

$$\begin{aligned} \lim_{m\to\infty} F(x) &= \lim_{m\to\infty} \left(ie^{i\varphi} \left(1 - \frac{\sin m \cos m}{m}\right)\right) \\ &= ie^{i\varphi} \lim_{m\to\infty} \left(1 - \frac{\sin m \cos m}{m}\right) \\ &= ie^{i\varphi}. \end{aligned}$$

Tuloksessa on edelleen mukana vaihe-ero φ, mutta laskemalla nähdään, ettei vaihe-ero vaikuta lausekkeen itseisarvoon:

$$\begin{aligned} F(x) &= ie^{i\varphi} \\ &= -\sin\varphi + i\cos\varphi \\ |F(x)| &= \sqrt{(-\sin\varphi)^2 + (\cos\varphi)^2} \\ &= \sqrt{\sin^2\varphi + \cos^2\varphi} \\ &= \sqrt{1} \\ &= 1. \end{aligned}$$

On siis löydetty tapa kvantifioida, kuinka hyvin näytesignaali korreloi tietyntaajuisen siniaallon kanssa riippumatta niiden vaihe-erosta.

Fourier-muunnos

Fourier-muunnos on matemaattinen menetelmä, jossa edellä kuvatunlainen signaalin taajuusvertailu tehdään mielivaltaiselle syötesignaalille [20]. Merkitään alkuperäistä signaalia $f(t)$, taajuusarvoa $\xi \in \mathbb{R}$ ja taajuuden vahvuuden ilmaisevaa funktiota $\hat{f}(\xi)$:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(t) \left(\cos(2\pi\xi t) - i \sin(2\pi\xi t)\right) \, \mathrm{d}t \tag{1.2}$$

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(t) e^{-i2\pi\xi t} \, \mathrm{d}t, \tag{1.3}$$

missä kaava (1.2) on esitetty karteesisessa muodossa ja kaava (1.3) on sama yhtälö, mutta esitettynä napakoordinaatistomuodossa. Signaalia kuvaavan funktion $f(t)$ tulee olla jatkuva sekä integroituva koko reaaliakselilla, käytännössä vaimeneva kaukana origosta; vain tällöin integraali on olemassa.

Kaavan yhtäläisyys edellä pääteltyyn on helpompi nähdä, jos parametri esitetään kulmataajuusmuodossa, jossa $\omega = 2\pi\xi$:

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) \left(\cos(\omega t) - i \sin(\omega t)\right) \, \mathrm{d}t$$

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} \, \mathrm{d}t.$$

Funktion $\hat{f}(\xi)$ arvot ovat kompleksisia. Niiden itseisarvo eli magnitudi kuvaa kyseisen taajuuskomponentin amplitudia syötesignaalissa, ja niiden argumentti kuvaa kyseisen taajuuskomponentin vaihekulmaa syötesignaalissa. Signaalifunktion $f(t)$ arvot voivat olla reaalisia tai kompleksisia.

Käänteinen Fourier-muunnos, jossa funktio taajuustasossa muunnetaan funktioksi signaalitasossa, saadaan vaihtamalla eksponentin etumerkkiä [15]:

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\xi) \left(\cos(2\pi\xi t) + i \sin(2\pi\xi t)\right) \, \mathrm{d}\xi \tag{1.4}$$

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{i2\pi\xi t} \, \mathrm{d}\xi. \tag{1.5}$$

Alla on esitetty sama vastaavasti kulmataajuusmuodossa:

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\omega) \left(\cos(\omega t) + i \sin(\omega t)\right) \, \mathrm{d}\omega$$

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega t} \, \mathrm{d}\omega.$$

2. Diskreetti Fourier-muunnos ja käänteismuunnos

Tilanteet, joissa haluttaisiin määrittää signaalin taajuusjakauma, ja tiedossa olisi tarkka matemaattinen lähtöfunktio, ovat harvinaisia. On tyypillisempää, että käsillä on sarja näytearvoja, jotka edustavat signaalin arvoja tietyin intervallein. Tällainen sarja voi esimerkiksi edustaa ääntä, josta on otettu näytteitä tietyin väliajoin [20]. Näytteet voivat edustaa myös kuvaa tai vaikka sähkömagneettisia resonansseja lääketieteellisestä kuvantamiskojeesta.

Näytteiden sarjaa voidaan merkitä $\{x_k\} := x_0, x_1, \ldots, x_{N-1}$, jossa N on näytteiden lukumäärä. Kun tämä sarja muunnetaan signaaliavaruudesta taajuusavaruuteen eli määritetään signaalin sinimuotoisten komponenttien amplitudi ja vaihe, käytetään *diskreettiä*[2] Fourier-muunnosta (DFT), jossa alkuperäisen Fourier-muunnoskaavan (1.3) ääretön integraali on korvattu äärellisellä summalausekkeella:

$$X_n = \sum_{k=0}^{N-1} x_k \cdot e^{-i2\pi nk/N}. \tag{2.1}$$

Diskreetin Fourier-muunnoksen tulos on sarja $\{X_k\} := X_0, X_1, \ldots, X_{N-1}$, missä komponentti X_k kuvaa taajuuskomponentin k amplitudia sekä vaihetta. Esimerkiksi mikäli sarja x sisältää yhden sekunnin mittaisen 48000 näytettä sekunnissa äänitetyn ääninäytteen – tällöin $N = 48000$ – silloin sinikomponentin X_{440} itseisarvo $|X_{440}|$ kertoo, mikä oli 440 Hz:n siniaaltokomponentin amplitudi näytteessä. Sinikomponentti X_0 sisältää kaikkien näytteiden matemaattisen summan. Mikäli näytteet kuvaisivat sähkövirran hetkittäisiä amplitudeja, X_0 ilmaisisi sähkövirran tasavirtakomponentin vahvuuden kerrottuna näytteiden määrällä.

On kuitenkin syytä huomata, että diskreetti taajuusjakauma sisältää arvoja vain kokonaislukutaajuuksille. Mikäli edellä mainittu ääninäyte kuvaakin esimerkiksi 440,5 Hz ääntä, ei taajuusjakaumassa ole erikseen arvoa tällaiselle taajuudelle[3]. Tästä rajoituksesta huolimatta diskreetillä Fourier-muunnoksella on lukemattomia käyttökohteita monilla eri tieteenaloilla.

Käänteinen DFT

Huolimatta siitä, ettei diskreetistä taajuusjakaumasta löydy arvoja kaikille taajuuksille, vaan arvoja löytyy ainoastaan vakiovälein, taajuusjakauma on mahdollista palauttaa täydellisesti näytesarjaksi käänteisellä muunnoksella. Käänteinen DFT eli IDFT (*Inverse Discrete Fourier Transform*) saadaan vaihtamalla eksponentin etumerkki ja jakamalla näytteen pituudella [20]:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi nk/N}. \tag{2.2}$$

[2] Diskreetti (engl. *discrete*) tarkoittaa selkeästi erotettavissa yksiköissä tai askeleissa tapahtuvaa. Ei pidä sekoittaa engl. sanaan *discreet*, joka tarkoittaa arkaluontoista tai hienovaraista.

[3] Käytännössä tällöin löytyy jonkinlainen piikki viereisistä indekseistä X_{440} sekä X_{441}. Jotain voi ehkä päätellä siitä, että harmonisessa monikerrassa 440,5 Hz $\cdot 2 = 881$ Hz saattaa tällöin esiintyä piikki, joka on vahvempi kuin vierekkäisissä kohdissa 880 Hz ja 882 Hz. Tarkan perustaajuuden määrittäminen luotettavasti diskreetistä taajuusjakaumasta on kuitenkin haastavaa, eikä se kuulu tämän tutkielman sisältöön.

Todistus. Alla sarja x on ensin muunnettu DFT:llä sarjaksi X, ja sen jälkeen muunnettu IDFT:llä sarja X sarjaksi y. Apuna on käytetty muuttujaa $W = e^{i2\pi/N}$, jolle pätee mm. lainalaisuus $(W^k)^N = W^{Nk} = e^{i2\pi k} = 1$ kaikilla $k \in \mathbb{Z}$. Lasketaan siis DFT:n ja IDFT:n yhdistelmä:

$$\begin{aligned}
X_n &= \sum_{j=0}^{N-1} x_j \cdot W^{-nj} \\
y_n &= \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot W^{nk} \\
&= \frac{1}{N} \sum_{k=0}^{N-1} \left(\sum_{j=0}^{N-1} x_j \cdot W^{-kj} \right) W^{nk} \\
&= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} x_j \cdot W^{k(n-j)} \\
&= \sum_{j=0}^{N-1} x_j \frac{1}{N} \sum_{k=0}^{N-1} W^{k(n-j)}.
\end{aligned}$$

Nyt sisempi summalauseke jakautuu seuraaviin kahteen tapaukseen:

$$\sum_{k=0}^{N-1} W^{k(n-j)} = \begin{cases} \sum_{k=0}^{N-1} \left(W^{n-j}\right)^k = \dfrac{\left(W^{n-j}\right)^N - 1}{W^{n-j} - 1} = \dfrac{1-1}{W^{n-j}-1} = 0 = N \cdot \underbrace{0 \quad \text{kun } j \neq n}_{\delta_{j,n}}, \\ \sum_{k=0}^{N-1} W^{k \,\cdot\, 0} = \sum_{k=0}^{N-1} W^0 = \sum_{k=0}^{N-1} 1 = N = N \cdot \underbrace{1 \quad \text{kun } j = n}_{\delta_{j,n}}. \end{cases}$$

Hyödyntäen Kroneckerin deltaa[4] voidaan kaksi tapausta yhdistää lausekkeeksi $N \cdot \delta_{j,n}$:

$$\begin{aligned}
y_n &= \sum_{j=0}^{N-1} x_j \frac{1}{N} \ (N \cdot \delta_{j,n}) \\
&= \sum_{j=0}^{N-1} x_j \cdot \delta_{j,n} \\
&= x_n.
\end{aligned}$$

Täten $y = x$; diskreetin Fourier-muunnoksen ja käänteismuunnoksen yhdistelmä säilyttää alkuperäiset arvot[5]. □

[4] Kroneckerin delta $\delta_{x,y} = \begin{cases} 0 & \text{kun } x \neq y, \\ 1 & \text{kun } x = y. \end{cases}$ Vaihtoehtoisesti voidaan esittää $\delta_{x,y} = \left\lfloor \frac{1}{1+|x-y|} \right\rfloor$.

[5] Puhtaan matemaattisesti tarkasteltuna alkuperäiset arvot todella palautuvat. Käytännössä näitä muunnoksia lasketaan vain tietokoneella. Tietokoneella laskettava matematiikka on häviöllistä, sillä laskenta suoritetaan käyttäen n.s. liukulukuja, jotka ovat eritoten irrationaalilukujen tapauksessa aina likiarvoja. Tietokoneella muunnos ja sen käänteismuunnos eivät siis aina palauta tarkalleen alkuperäisiä arvoja. Kadonneen tarkkuuden määrä riippuu käytetystä laskentatarkkuudesta sekä tarkasta laskentamenetelmästä.

2.1. Moniulotteinen DFT

Tavallinen sekä diskreetti Fourier-muunnos voidaan laskea myös useammassa ulottuvuudessa. Merkitään d-ulotteista tietojoukkoa seuraavasti:

$$y = y_{n_1,n_2,\ldots,n_d},$$

missä $n_k = \{0, 1, \ldots, N_k - 1\}$ ja $k = \{1, 2, \ldots, d\}$, ja N_k kuvaa tietojoukon pituutta ulottuvuudessa numero k. Tällöin moniulotteinen diskreetti Fourier-muunnos lasketaan seuraavasti [20]:

$$\begin{aligned} Y_{n_1,n_2,\ldots,n_d} &= \sum_{k_1=0}^{N_1-1} \left(e^{-i2\pi \frac{n_1 k_1}{N_1}} \cdot \sum_{k_2=0}^{N_2-1} \left(e^{-i2\pi \frac{n_2 k_2}{N_2}} \cdot \ldots \sum_{k_d=0}^{N_d-1} \left(e^{-i2\pi \frac{n_d k_d}{N_d}} y_{k_1,k_2,\ldots,k_d} \right) \right) \right) \\ &= \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \cdots \sum_{k_d=0}^{N_d-1} e^{-i2\pi \left(\frac{n_1 k_1}{N_1} + \frac{n_2 k_2}{N_2} + \cdots + \frac{n_d k_d}{N_d} \right)} y_{k_1,k_2,\ldots,k_d}. \end{aligned} \tag{2.3}$$

Kaavasta nähdään, että muunnos on itse asiassa sarja sisäkkäisiä yksiulotteisia DFT-muunnoksia kullekin ulottuvuudelle. Menettelytavan voi hahmottaa visuaalisesti esimerkiksi siten, että kuvitellaan, että tietojoukko on kolmiulotteinen, kuin kuutio. Oletetaan, että kuution koko on $\dot{X} \times \dot{Y} \times \dot{Z}$. Ensin kullekin kuution vaakasuuntaiselle riville (jotka ovat $\dot{X}$-pituisia ja joita on $\dot{Y} \times \dot{Z}$ kpl) lasketaan yksitellen yksiulotteinen DFT, ja rivin alkiot korvataan muunnetuilla alkioilla. Kaikki kuution alkiot korvataan näin muunnetuilla alkioilla. Tämän jälkeen kukin tuloksena syntyneen kuution sarake eli pystysuuntainen rivi (joita on $\dot{X} \times \dot{Z}$ kpl) korvataan vastaavasti niiden yksiulotteisen DFT:n alkioilla. Lopuksi sama tehdään vielä syvyyssuunnassa ($\dot{X} \times \dot{Y}$ kpl $\dot{Z}$-pituisia rivejä). Ulottuvuuksien käsittelyjärjestys voi tietenkin olla mikä tahansa, ja eri ulottuvuuksien pituudet voivat erota toisistaan.

Tutkimalla kaavojen (2.1) ja (2.2) yhteyttä voidaan käänteinen moniulotteinen diskreetti Fourier-muunnos päätellä yhtälöstä (2.3) seuraavasti:

$$y_{n_1,n_2,\ldots,n_d} = \left(\prod_{k=1}^{d} \frac{1}{N_k} \right) \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \cdots \sum_{k_d=0}^{N_d-1} e^{i2\pi \left(\frac{n_1 k_1}{N_1} + \frac{n_2 k_2}{N_2} + \cdots + \frac{n_d k_d}{N_d} \right)} Y_{k_1,k_2,\ldots,k_d}.$$

Tässä tutkielmassa keskitytään jatkossa kuitenkin vain yksiulotteiseen muunnokseen.

2.2. DFT:n ja IDFT:n liittolukusuhde

DFT ja IDFT ovat matemaattisesti hyvin samankaltaisia. Kerrataan vielä nämä kaavat:

$$\mathrm{DFT}: \begin{cases} \mathbb{C}_N & \to \mathbb{C}_N \\ X & \mapsto \mathrm{DFT}(x) \end{cases}, \qquad X_n = \sum_{k=0}^{N-1} x_k \cdot e^{-i2\pi \frac{nk}{N}} \quad \text{ja} \qquad (2.1 \text{ kerrattu})$$

$$\mathrm{IDFT}: \begin{cases} \mathbb{C}_N & \to \mathbb{C}_N \\ x & \mapsto \mathrm{IDFT}(X) \end{cases}, \qquad x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi \frac{nk}{N}}. \qquad (2.2 \text{ kerrattu})$$

Käytetään seuraavaksi liittoluvulle merkintää $\overline{z} = z_{\mathrm{re}} - iz_{\mathrm{im}}$, kun $z = z_{\mathrm{re}} + iz_{\mathrm{im}}$.

Lause 2.4. Käänteinen DFT voidaan kirjoittaa DFT:n funktiona seuraavasti:

$$\mathrm{IDFT}(X) = \frac{1}{N} \overline{\mathrm{DFT}\left(\overline{X}\right)}.$$

Todistus. Mikäli lause on tosi, tällöin pätee:

$$x_n = \underbrace{\frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi \frac{nk}{N}}}_{\mathrm{IDFT}(X)} = \frac{1}{N} \underbrace{\overline{\sum_{k=0}^{N-1} \overline{X_k} \cdot e^{-i2\pi \frac{nk}{N}}}}_{\overline{\mathrm{DFT}(\overline{X})}}$$

$$= \frac{1}{N} \overbrace{\sum_{k=0}^{N-1} \overline{\overline{X_k} \cdot e^{-i2\pi \frac{nk}{N}}}}.$$

Merkitään $\omega = 2\pi \frac{nk}{N}$ sekä $z = X_k$. Osoitetaan siis, että $z \cdot e^{i\omega} = \overline{\overline{z} \cdot e^{-i\omega}}$:

$$\begin{aligned}
\overline{\overline{z}e^{-i\omega}} &= \overline{(z_{\mathrm{re}} - iz_{\mathrm{im}})(\cos(-\omega) + i\sin(-\omega))} \\
&= \overline{(z_{\mathrm{re}} - iz_{\mathrm{im}})\cos(-\omega) + (z_{\mathrm{re}} - iz_{\mathrm{im}})i\sin(-\omega)} \\
&= \overline{(z_{\mathrm{re}} - iz_{\mathrm{im}})\cos(-\omega) + (z_{\mathrm{im}} + iz_{\mathrm{re}})\sin(-\omega)} \\
&= \overline{z_{\mathrm{re}}\cos(-\omega) + z_{\mathrm{im}}\sin(-\omega) + i\left(z_{\mathrm{re}}\sin(-\omega) - z_{\mathrm{im}}\cos(-\omega)\right)} \\
&= z_{\mathrm{re}}\cos(-\omega) + z_{\mathrm{im}}\sin(-\omega) - i\left(z_{\mathrm{re}}\sin(-\omega) - z_{\mathrm{im}}\cos(-\omega)\right) \\
&= z_{\mathrm{re}}\cos(-\omega) + z_{\mathrm{im}}\sin(-\omega) - iz_{\mathrm{re}}\sin(-\omega) + iz_{\mathrm{im}}\cos(-\omega) \\
&= z_{\mathrm{re}}\cos\omega - z_{\mathrm{im}}\sin\omega + iz_{\mathrm{re}}\sin\omega + iz_{\mathrm{im}}\cos\omega \\
&= (z_{\mathrm{re}} + iz_{\mathrm{im}})\cos\omega + (iz_{\mathrm{re}} - z_{\mathrm{im}})\sin\omega \\
&= (z_{\mathrm{re}} + iz_{\mathrm{im}})\cos\omega + (z_{\mathrm{re}} + iz_{\mathrm{im}})i\sin\omega \\
&= (z_{\mathrm{re}} + iz_{\mathrm{im}})\, e^{i\omega} \\
&= ze^{i\omega}.
\end{aligned}$$

□

Tätä yhtäläisyyttä hyödynnetään myöhemmin luvuissa 4.3 ja 4.4 käsiteltäessä Raderin ja Bluesteinin FFT-muunnoksia. Vastaavasti nähdään, että pätee myös:

$$\mathrm{DFT}(x) = N\, \overline{\mathrm{IDFT}\left(\overline{x}\right)}.$$

Toinen tietokoneellisen käsittelyn kannalta hyödyllinen suhde on seuraava:

Lause 2.5. Käänteinen DFT voidaan kirjoittaa DFT:n funktiona seuraavasti:

$$\mathrm{IDFT}(X) = \frac{1}{N}\mathrm{swap}\left(\mathrm{DFT}\left(\mathrm{swap}(X)\right)\right),$$

jossa $\mathrm{swap}(z) = \overline{z}i$.

Päällisin puolin tämä näyttää monimutkaisemmalta kuin pelkkä liittoluku. Kuinka liittoluku + kertolasku voisi olla yksinkertaisempi kuin pelkkä liittoluku? Lausekkeen avaaminen komponentteihinsa paljastaa totuuden:

$$\begin{aligned}\mathrm{swap}(z_{\mathrm{re}} + iz_{\mathrm{im}}) &= \overline{z_{\mathrm{re}} + iz_{\mathrm{im}}} \cdot i \\ &= (z_{\mathrm{re}} - iz_{\mathrm{im}}) \cdot i \\ &= z_{\mathrm{im}} + iz_{\mathrm{re}}.\end{aligned}$$

Kyseessä on siis laskutoimitus, joka nimensä mukaisesti vaihtaa luvun reaali- ja imaginäärikomponentit keskenään. Nykyaikaisten tietokoneiden superskalaarisilla suorittimilla arvojen siirtäminen rekistereistä toiseen on suoritusajan kannalta käytännössä ilmainen operaatio [17]. Siksi tämä voi olla laskennallisesti tehokkaampi menetelmä johtaa IDFT DFT:stä kuin edellä esitetty liittolukumenetelmä[6].

Todistus. Todistetaan lause 2.5 samalla menetelmällä kuin lause 2.4 edellä:

$$\begin{aligned}\mathrm{swap}(\mathrm{swap}(z)e^{i\omega}) &= \mathrm{swap}\left(\mathrm{swap}(z_{\mathrm{re}} + iz_{\mathrm{im}})e^{i\omega}\right) \\ &= \mathrm{swap}\left((z_{\mathrm{im}} + iz_{\mathrm{re}})e^{i\omega}\right) \\ &= \mathrm{swap}\left((z_{\mathrm{im}} + iz_{\mathrm{re}})(\cos\omega + i\sin\omega)\right) \\ &= \mathrm{swap}\left(z_{\mathrm{im}}\cos\omega + iz_{\mathrm{re}}\cos\omega + z_{\mathrm{im}}i\sin\omega - z_{\mathrm{re}}\sin\omega\right) \\ &= \mathrm{swap}\left(z_{\mathrm{im}}\cos\omega - z_{\mathrm{re}}\sin\omega + i(z_{\mathrm{re}}\cos\omega + z_{\mathrm{im}}\sin\omega)\right) \\ &= i(z_{\mathrm{im}}\cos\omega - z_{\mathrm{re}}\sin\omega) + z_{\mathrm{re}}\cos\omega + z_{\mathrm{im}}\sin\omega \\ &= i(z_{\mathrm{im}}\cos(-\omega) + z_{\mathrm{re}}\sin(-\omega)) + z_{\mathrm{re}}\cos(-\omega) - z_{\mathrm{im}}\sin(-\omega) \\ &= (z_{\mathrm{re}} + iz_{\mathrm{im}})\cos(-\omega) + (iz_{\mathrm{re}} - z_{\mathrm{im}})\sin(-\omega) \\ &= (z_{\mathrm{re}} + iz_{\mathrm{im}})\cos(-\omega) + (z_{\mathrm{re}} + iz_{\mathrm{im}})i\sin(-\omega) \\ &= (z_{\mathrm{re}} + iz_{\mathrm{im}})\ e^{-i\omega} \\ &= z\ e^{-i\omega}.\end{aligned}$$

□

Vastaavasti pätee myös:

$$\mathrm{DFT}(x) = N\mathrm{swap}\left(\mathrm{IDFT}\left(\mathrm{swap}(x)\right)\right).$$

[6]Käytännössä eroa ei välttämättä ole. Mikäli liittolukua käytetään yhteen- tai kertolaskussa, tällöin imaginäärikomponttien yhteenlasku muuttuu kääntäjän suorittamassa optimointivaiheessa vähennyslaskuksi, joka on tietokoneelle aivan yhtä nopeaa. Koska liittolukumenetelmässä lukuarvot säilyvät rekisterin sisällä omilla ”kaistoillaan”, se saattaa jopa olla swap-menetelmää marginaalisesti nopeampi. Kirjoittajan suorittamissa vertailevissa testeissä havaittavia suoritusaikaeroja näiden kahden menetelmän välillä ei ilmennyt. FFTW-kirjasto (versio 3.3.10) käyttää swap-menetelmää Bluesteintotetuksessaan ja liittolukumenetelmää Rader-toteutuksessaan (`dft/bluestein.c`, `dft/rader.c`).

3. Diskreetti kosinimuunnos

Diskreetti Fourier-muunnos (DFT) suorittaa laskennan kompleksiluvuilla, ja tuottaa tuloksena kompleksilukuja. Monissa käytännön sovelluksissa, joissa diskreettiä Fourier-muunnosta voisi käyttää, tavoitteen saavuttamiseksi riittää kuitenkin pelkkä reaalinen taajuustieto. Esimerkiksi mainittakoon JPEG-kuvanpakkaus, joka perustuu siihen psykovisuaaliseen seikkaan, että ihminen ei välttämättä erota kaikkia korkeataajuisia visuaalisia yksityiskohtia, eikä niiden vähäinen puuttuminen heikennä sanottavasti kuvan tunnistettavuutta [10, 21]. Tästä on esimerkki kuvassa 2: Vielä siinäkin vaiheessa, kun kuvan matemaattisista yksityiskohdista on enää 7 % jäljellä, kuvan kaikki konseptuaaliset yksityiskohdat ovat edelleen erotettavissa, ja kauempaa katsottuna kuva näyttää kutakuinkin samalta kuin alkuperäinen. Kuvanpakkauksen kannalta diskreetti kosinimuunnos, DCT, on usein houkuttelevampi menetelmä kuin DFT, sillä se esittää reaalisen signaalin taajuustiedon puolet pienemmässä tilassa kuin DFT käyttäen pelkkiä reaalilukuja, kuitenkin tavalla, joka on käänteismuunnoksella palautettavissa alkuperäiseksi näytesarjaksi [10].

Matemaattisesti DCT muistuttaa paljon DFT-muunnosta, mutta kompleksisen $e^{i2\pi nk/N}$-lausekkeen sijaan, toisin sanoen $\cos(2\pi nk/N) + i\sin(2\pi nk/N)$, siinä esiintyy reaalinen kosinilauseke muotoa $\cos(\pi nk/N)$ tyystin ilman imaginääriosaa. DCT ja DFT eroavat toisistaan myös siten, että DCT:stä esiintyy monia erilaisia muotoja. Yleisesti ottaen ne voidaan tiivistää neljään eri malliin, joita kaikkia voidaan kuvata kaavalla (3.1). Kaavassa esiintyvät parametrit $p_1 \ldots p_6$ riippuvat DCT:n tyypistä, ja ne on listattu taulukossa 1.

$$X_n = (p_1 x_0 + p_2 x_{(N-1)}) + \sum_{k=p_3}^{N-p_4} x_k \cos \frac{\pi(k+p_5)(n+p_6)}{N+1-p_4} \tag{3.1}$$

Tyyppi	p_1	p_2	p_3	p_4	p_5	p_6	Käänteismuunnos
DCT-I	$\frac{1}{2}$	$\frac{1}{2}(-1)^n$	1	2	0	0	$\frac{2}{N-1}$ DCT-I
DCT-II	0	0	0	1	$\frac{1}{2}$	0	$2/N$ DCT-III
DCT-III	$\frac{1}{2}$	0	1	1	0	$\frac{1}{2}$	$2/N$ DCT-II
DCT-IV	0	0	0	1	$\frac{1}{2}$	$\frac{1}{2}$	$2/N$ DCT-IV

Taulukko 1: Eri DCT-tyyppien parametrit kaavaan (3.1) sekä kunkin tyypin käänteismuunnokset [19, 20].

Yleisimmin käytetty näistä tyypeistä on DCT-II, eli tyypin II DCT, jota useimmiten kutsutaankin pelkästään DCT:ksi. Kun tyyppiä II vastaavat parametrit taulukosta 1 sijoitetaan kaavaan (3.1), saadaan seuraava kaava:

$$X_n = \sum_{k=0}^{N-1} x_k \cos \frac{\pi n\,(k+1/2)}{N}. \tag{3.2}$$

Herää kuitenkin kysymys: Miksi pelkkä kosinifunktio riittää nyt signaalisarjan muuntamiseksi taajuussarjaksi, kun luvun 1 tutkinnassa todettiin, ettei pelkkä sini tai kosini riitä? On totta, että diskreetti kosinimuunnos on herkkä syötesignaalin vaihe-eroille.

(a) Tallennusmuoto: PNG (häviötön)

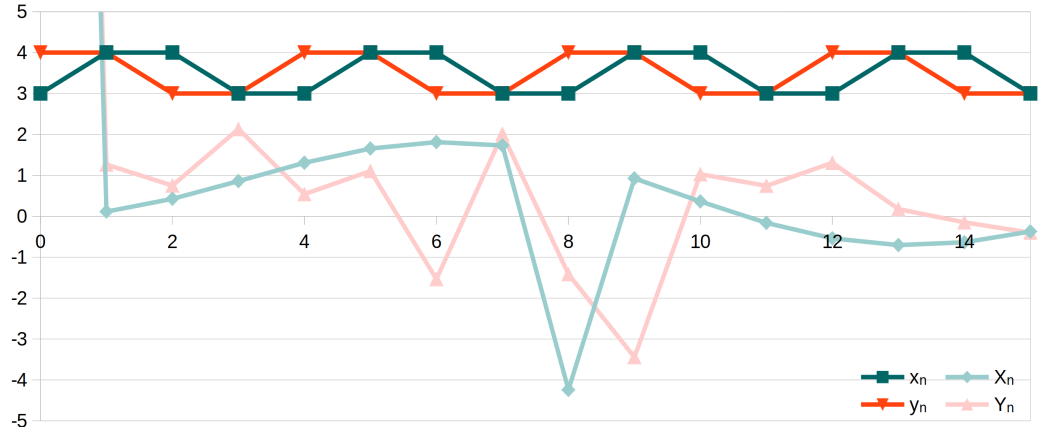
(b) Tallennusmuoto: JPEG (55 %:n laatu)

(c) Tallennusmuoto: JPEG (20 %:n laatu)

(d) Tallennusmuoto: JPEG (7 %:n laatu)

Kuva 2: Eri tavoin pakattu valokuva [28].

Kuvaajassa 3 on esitetty kaksi signaalia x ja y, jotka eroavat toisistaan vain vaiheen osalta. Samasta kuvaajasta nähdään, että näiden DCT:t ovat hyvin erinäköiset. Vaikka DCT ei tuotakaan sellaista muunnosta, josta suoraan näkisi luotettavasti, mikä syötteen jonkin tietyn taajuuskomponentin arvo oli, muunnoksen tuloksena syntyvä taajuussarja korreloi silti syötesignaalin taajuuskomponenttien kanssa riittävän hyvin, että DCT:lle voi keksiä monia käyttökohteita, mm. juuri kuvanpakkauksessa. Lisäksi taajuussarja on muunnettavissa häviöttömästi takaisin alkuperäiseksi signaaliksi aivan kuten diskreetissä Fourier-muunnoksessakin [19]. Käänteismuunnokset on esitetty taulukossa 1.



Kuva 3: Näytesarjan x DCT X ja näytesarjan y DCT Y. Sarja x muodostuu neljästä peräkkäisestä jonon $3, 4, 4, 3$ toistosta ja sarja y neljästä peräkkäisestä jonon $4, 4, 3, 3$ toistosta. Muunnostuloksen ensimmäinen alkio on näytteiden summa, eikä se mahtunut kuvaajaan.

Käyttö JPEG-kuvanpakkauksessa

JPEG-kuvanpakkauksessa käytetään tyypin II DCT-muunnosta. JPEG-pakkauksessa muunnos tehdään kuitenkin kaksiulotteisesti 8×8-soluille, ja kunkin sarakkeen tai rivin ensimmäinen alkio jaetaan vakiolla $\sqrt{2}$ ennen muunnosta [10]. Kun merkitään $N = 8$, alkuperäisen kuvan pikseliarvoja kuvaavaa funktiota $f(x, y)$, missä $f(x, y)$ saa arvoja $[-2^7, 2^7 - 1]$, ja muistetaan kaavan 3.2 lisäksi luvussa 2.1 käsitelty periaate moniulotteiseen muunnokseen, saadaan JPEG:n käyttämän kosinimuunnoksen kaavaksi seuraava[7]:

$$F(u, v) = \left(2^{-\frac{4 + \delta_{u,0} + \delta_{v,0}}{2}}\right) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos \frac{\pi u(x + 1/2)}{N} \cos \frac{\pi v(y + 1/2)}{N},$$

missä merkintä $\delta_{x,y}$ viittaa Kroneckerin deltaan, jonka merkitys selitettiin sivulla 6. Vastaavasti JPEG-pakkauksen käyttämän käänteismuunnoksen (IDCT) kaava on seuraava [10, 24]:

$$f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \left(2^{-\frac{4 + \delta_{u,0} + \delta_{v,0}}{2}}\right) F(u, v) \cos \frac{\pi u(x + 1/2)}{N} \cos \frac{\pi v(y + 1/2)}{N}.$$

[7] Yksinkertaistus. Todellisuudessa laskennassa käytetään nopeampia algoritmeja, jotka tuottavat saman tuloksen vähemmillä laskentaoperaatioilla, verrattavissa seuraavan luvun (4) FFT-algoritmeihin.

4. Nopea Fourier-muunnos

Diskreetin Fourier-muunnoksen kaavasta (2.1) nähdään piirre, joka tekee sen hyödyntämisestä hankalaa. Asian selventämiseksi tutkitaan seuraavaksi laskennassa tarvittavien kertolaskujen määrää. Kun hetkellisiä signaalitasoja kuvaava sarja x muunnetaan taajuusjakaumaa kuvaavaksi sarjaksi X, lasketaan yhteensä N summaa: yksi kullekin arvolle X_k, $k \in \{0, 1, \ldots, N-1\}$. Jokaista summaa varten puolestaan lasketaan yhteensä N kompleksikertolaskua, yksi kullekin syötearvolle x_k. Yhteensä kompleksikertolaskuja pitää tehdä siis $N \cdot N = N^2$ kappaletta[8]. Suhde on havainnollistettu alla:

$$\underbrace{X_n = \underbrace{\sum_{k=0}^{N-1} x_k \cdot e^{-i2\pi nk/N}}_{N \text{ kertolaskua}}}_{N \cdot N \text{ kertolaskua}}. \qquad \text{(2.1 kerrattu)}$$

Algoritmi DFT: Naiivi DFT.

Mikäli näytteen pituus N kasvaa kaksinkertaiseksi, tarvittavien kertolaskujen määrä kasvaa nelinkertaiseksi. Tietojenkäsittelytieteessä algoritmin suorittamien operaatioiden määrää suhteessa syötteen kokoon, eli suoritusaikavaativuutta, merkitään $\mathcal{O}$-merkinnällä, joka tässä neliöllisessä tapauksessa olisi $\mathcal{O}(N^2)$. Algoritmit, joiden suoritusaika on neliöllinen, ovat usein liian hitaita käytännön käyttötarkoituksiin [16]. Esimerkiksi luvussa 2 mainitun yhden sekunnin ääninäytteen tapauksessa kompleksikertolaskuja pitäisi tehdä $48000^2 \approx 2{,}3$ miljardia kappaletta. Useimmille tietokoneille tuo olisi kohtuuttoman raskas operaatio etenkin, mikäli laskenta pitäisi tehdä reaaliajassa.

Nopea Fourier-muunnos (FFT, *Fast Fourier Transform*) tarkoittaa mitä tahansa sellaista diskreettiä Fourier-muunnosta, joka voidaan suorittaa nopeammassa $\mathcal{O}(N \log N)$ -ajassa neliöllisen $\mathcal{O}(N^2)$ -ajan sijaan. Tyypillisesti sillä viitataan nimenomaan James Cooleyn ja John Tukeyn algoritmiin, jonka he julkaisivat vuonna 1965 [1].

4.1. Cooleyn-Tukeyn algoritmi

Cooleyn ja Tukeyn algoritmi perustuu pohdintaan siitä, mitä DFT:ssä voidaan tehdä, jos N on kahden kokonaisluvun tulo. Merkitään $N = r_1 \cdot r_2$. Nyt näytteitä läpi käyvä laskuri k voidaan jakaa kahteen osaan siten, että $k = k_1 r_2 + k_0$, missä $k_0 = 0, 1, \ldots, r_2 - 1$ ja $k_1 = 0, 1, \ldots, r_1 - 1$. Tällöin yhtälö (2.1) saadaan muutettua seuraavasti:

$$\begin{aligned} X_n &= \sum_{k=0}^{N-1} x_k \cdot e^{-i2\pi nk/N} && ||k = k_1 r_2 + k_0 \\ &= \sum_{k_0=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi n k_1 r_2/N} e^{-i2\pi n k_0/N}. \end{aligned}$$

[8] Laskennan kannalta oletetaan, että kerroinarvot $e^{-i2\pi x/N}$ ovat vakioita, sillä niiden arvot eivät riipu syötteen sisällöstä. Yhteenlaskujen lukumäärää ei tässä yhteydessä arvioida.

Vastaavasti jaetaan laskuri n kahteen osaan siten, että $n = n_1 r_1 + n_0$, missä $n_0 = 0, 1, \ldots, r_1 - 1$ ja $n_1 = 0, 1, \ldots, r_2 - 1$. Silloin saadaan seuraavanlainen muutos, joka perustuu havaintoon, että $e^{i2\pi(x+yN)/N} = e^{i2\pi x/N} e^{i2\pi y} = e^{i2\pi x/N}$ kun $y \in \mathbb{Z}$:

$$
\begin{aligned}
X_{(n_1 r_1 + n_0)} &= \sum_{k_0=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi n k_1 r_2/N} e^{-i2\pi n k_0/N} && ||n = n_1 r_1 + n_0 \\
&= \sum_{k_0=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi (n_1 r_1 + n_0) k_1 r_2/N} e^{-i2\pi (n_1 r_1 + n_0) k_0/N} \\
&= \sum_{k_0=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi (n_1 k_1 r_1 r_2 + n_0 k_1 r_2)/N} e^{-i2\pi (n_1 r_1 + n_0) k_0/N} && ||r_1 r_2 = N \\
&= \sum_{k_0=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi (n_1 k_1 N + n_0 k_1 r_2)/N} e^{-i2\pi (n_1 r_1 + n_0) k_0/N} \\
&= \sum_{k_0=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi n_0 k_1 r_2/N} e^{-i2\pi (n_1 r_1 + n_0) k_0/N} \\
&= \sum_{k_0=0}^{r_2-1} \left(\sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi n_0 k_1 r_2/N} \right) e^{-i2\pi (n_1 r_1 + n_0) k_0/N} \\
&= \sum_{k_0=0}^{r_2-1} \left(\sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi n_0 k_1/r_1} \right) e^{-i2\pi (n_1 r_1 + n_0) k_0/N}. && (4.1)
\end{aligned}
$$

Koska yhtälön (4.1) sisempi summa riippuu ulomman summan muuttujasta k_0 ainoastaan indeksin osalta, voidaan summa laskea nyt kahdessa peräkkäisessä vaiheessa kahden sisäkkäisen vaiheen sijaan:

$$
Y_{(n_0 r_2 + k_0)} = \sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi n_0 k_1/r_1} \tag{4.2}
$$

$$
X_{(n_1 r_1 + n_0)} = \sum_{k_0=0}^{r_2-1} Y_{(n_0 r_2 + k_0)} e^{-i2\pi (n_1 r_1 + n_0) k_0/N}. \tag{4.3}
$$

Algoritmi CT0: Nollan rekursion versio Cooleyn-Tukeyn algoritmista.

Algoritmissa CT0 ei suoriteta yhtään rekursiota. Nyt kuitenkin voidaan nähdä, että yhtälö (4.2) on muodoltaan täysin samanlainen kuin alkuperäinenkin yhtälö (2.1). Siihenkin voidaan siis soveltaa FFT-algoritmia rekursiivisesti seuraavasti:

$$
\begin{aligned}
x' k_0, n_0 &= x_{(n_0 r_2 + k_0)} \\
Y_{(n_0 r_2 + k_0)} &= \mathrm{FFT}(x'_{k_0})_{n_0} \\
X_{(n_1 r_1 + n_0)} &= \sum_{k_0=0}^{r_2-1} Y_{(n_0 r_2 + k_0)} e^{-i2\pi (n_1 r_1 + n_0) k_0/N}. && \text{(4.3 kerrattu)}
\end{aligned}
$$

Algoritmi CT1: Yhden rekursion versio Cooleyn-Tukeyn algoritmista.

Algoritmissa CT1 suoritetaan N-kokoisen FFT:n laskemiseksi r_2 kpl r_1-kokoisia FFT-muunnoksia. Myös yhtälön (4.3) summalausekkeelle voidaan tehdä sama käsittely. Tämä edellyttää kyseisessä yhtälössä esiintyvän potenssilausekkeen jakamisen kahteen osaan:

$$e^{-i2\pi(n_1 r_1 + n_0)k_0/N} = e^{-i2\pi n_1 r_1 k_0/N} e^{-i2\pi n_0 k_0/N} = e^{-i2\pi n_1 k_0/r_2} e^{-i2\pi n_0 k_0/N}.$$

Silloin saadaan aikaan kahden rekursion versio Cooleyn-Tukeyn algoritmista:

$$\begin{aligned}
x'_{k_0,n_0} &= x_{(n_0 r_2 + k_0)} \\
Y_{n_0,k_0} &= \mathrm{FFT}(x'_{k_0})_{n_0} && \| r_2 \text{ kpl } r_1\text{-kokoisia FFT-muunnoksia} \\
Y'_{n_0,k_0} &= Y_{n_0,k_0} e^{-i2\pi n_0 k_0/N} \\
X'_{n_1,n_0} &= \mathrm{FFT}(Y'_{n_0})_{n_1} && \| r_1 \text{ kpl } r_2\text{-kokoisia FFT-muunnoksia} \\
X_{(n_1 r_1 + n_0)} &= X'_{n_1,n_0}.
\end{aligned} \tag{4.4}$$

Algoritmi CT2: Kahden rekursion versio Cooleyn-Tukeyn algoritmista.

Analyysi

Määritellään seuraavaksi muutamia merkintöjä sekä oletuksia:

- Tarkoittakoon $h_{\mathcal{A},N}$ niiden kertolaskujen lukumäärää, jonka algoritmi $\mathcal{A}$ suorittaa N-kokoisen osajoukon muuntamiseksi. Esimerkiksi yhtälöstä (2.1) nähdään, että DFT-algoritmille pätee $h_{\mathrm{DFT},N} = N^2$ kaikilla $N \in \mathbb{N}$. Olkoon h_N merkintä, jossa N-kokoisen osajoukon muuntamiseen käytettävää algoritmia ei ole eksplisiittisesti valittu.
- Tarkoittakoon $g_{\mathcal{A},N} = \frac{h_{\mathcal{A},N}}{N}$ sekä $g_N = \frac{h_N}{N}$ niiden kertolaskujen lukumäärää, jonka algoritmi $\mathcal{A}$ suorittaa N-kokoisen osajoukon *yksittäisen* jäsenen arvon laskemiseksi. Esimerkiksi $g_{\mathrm{DFT},N} = N$ kaikilla $N \in \mathbb{N}$. Myös merkintä g_N tarkoittaa, että N-kokoisen osajoukon muuntamiseen käytettävää algoritmia ei ole eksplisiittisesti valittu.
- Oletetaan, että mitään DFT:tä hitaampaa algoritmia ei käytetä. Koska DFT:lle pätee $g_{\mathrm{DFT},N} = N$ kaikilla $N \in \mathbb{N}$, voidaan olettaa, että $g_N \leqslant N$ kaikilla $N \in \mathbb{N}$.
- Tarkoittakoon $\Omega(N)$ luvun N sisältämien alkulukutekijöiden lukumäärää[9].
- Tarkoittakoon $\mathrm{Sopfr}(N)$ luvun N alkulukutekijöiden summaa (*sum of prime factors (with repetition)*). Huomionarvoista on, että $\mathrm{Sopfr}(N) \leqslant N$ kaikilla $N \in \mathbb{N}$. Alkuluvuilla $\mathrm{Sopfr}(N) = N$.

[9] Esim. $\Omega(203) = 2$, koska $203 = 7 \cdot 29$, ja $\Omega(36) = 4$, koska $36 = 2 \cdot 2 \cdot 3 \cdot 3$. Jos $\Omega(N) = 1$, niin N on alkuluku. $\Omega(1) = 0$. Jos pätee $N = p^k$, $\Omega(p) = 1$, $k \in \mathbb{N}$, tällöin $\Omega(N) = k = \log_p N$. Mielenkiintoista on, että tämä funktio noudattaa samoja laskusääntöjä kuin logaritmifunktio: $\Omega(x \cdot y) = \Omega(x) + \Omega(y)$, $\Omega(x^y) = y\Omega(x)$ ja (jos jaollinen niin) $\Omega\left({}^{x}/_{y}\right) = \Omega(x) - \Omega(y)$.

Olkoon $N = r_1 r_2$, missä $r_1, r_2 \in \mathbb{N}$ ja $r_1 > 1, r_2 > 1$. Kertolaskujen lukumäärät tähänastisissa versioissa Cooleyn-Tukeyn algoritmista voidaan määrittää seuraavasti:

- Nollan rekursion versiossa (algoritmi CT0) pätee $g_{\mathrm{CT0},N} = r_1 + r_2$. Tämä nähdään kaavasta (4.2), jossa lasketaan alkiota kohti r_1 kertolaskua ja kaavasta (4.3), jossa lasketaan alkiota kohti r_2 kertolaskua; yhteensä siis $r_1 + r_2$ kertolaskua tulosjoukon alkiota kohti. Tutkitaan seuraavaksi, missä tilanteessa nollan rekursion versio on tehokkaampi kuin naiivi DFT. Muodostetaan siis epäyhtälö $g_{\mathrm{CT0},N} \leqslant g_{\mathrm{DFT},N}$. Sieventämällä tästä saadaan $r_1 + r_2 \leqslant r_1 r_2$, joka on aina tosi[10]. Päätellään siis, että CT0 on aina vähintään yhtä tehokas kuin naiivi DFT. Mikäli $N = 4$, tällöin $r_1 = r_2 = 2$ ja $r_1 + r_2 = r_1 r_2$, mistä seuraa $g_{\mathrm{CT0},N} = g_{\mathrm{DFT},N}$. Kaikissa muissa tapauksissa $r_1 + r_2 < r_1 r_2$ ja $g_{\mathrm{CT0},N} < g_{\mathrm{DFT},N}$ eli CT0 on tehokkaampi kuin naiivi DFT, mikäli algoritmi on ylipäänsä mahdollista suorittaa, eli N on yhdistetty luku.

- Yhden rekursion versiossa (algoritmi CT1) pätee $g_{\mathrm{CT1},N} = g_{r_1} + r_2$. Sieventämällä $g_{\mathrm{CT1},N} \leqslant g_{\mathrm{CT0},N}$ saadaan $g_{r_1} \leqslant r_1$, joka on aina tosi sen oletuksen nojalla, että mitään DFT:tä hitaampaa algoritmia ei käytetä. Päätellään siis, että CT1 on aina vähintään yhtä tehokas CT0.

- Kahden rekursion versiossa (algoritmi CT2) pätee $g_{\mathrm{CT2},N} = g_{r_1} + g_{r_2} + 1$, jossa plus 1 johtuu yhtälöstä (4.4), jota yhden rekursion versiossa ei ole. Sieventämällä $g_{\mathrm{CT2},N} \leqslant g_{\mathrm{CT1},N}$ saadaan $g_{r_2} \leqslant r_2 - 1$, joka voidaan kirjoittaa selkeämmin $g_{r_2} < r_2$, koska g_{r_2} ja r_2 ovat kokonaislukuja. Toisinsanoen mikäli $g_{r_2} < r_2$, silloin CT2 on vähintään yhtä tehokas kuin CT1 olisi samoilla parametreillä r_1 ja r_2.

Edellä päätellyn perusteella Cooleyn-Tukeyn algoritmille pätee seuraava, olettaen, että algoritmi valitaan aina optimaalisesti:

$$\begin{aligned} g_N &= \min\{g_{\mathrm{CT1},N},\, g_{\mathrm{CT2},N}\} \\ &= \min\{g_{r_1} + r_2,\, g_{r_1} + g_{r_2} + 1\} \\ &= g_{r_1} + \min\{r_2,\, g_{r_2} + 1\}, \end{aligned}$$

mistä saadaan myös koko muunnoksen kertolaskujen määräksi yhtä lailla rekursiivinen kaava:

$$h_N = r_2 h_{r_1} + \min\{N r_2,\, N + h_{r_2}\}.$$

[10] Epäyhtälö $r_1 + r_2 \leqslant r_1 r_2$ voidaan muokata muotoon $r_1 r_2 - r_1 - r_2 \geqslant 0$, edelleen $r_1 r_2 - r_1 - r_2 + 1 \geqslant 1$, ja edelleen $(r_1 - 1)(r_2 - 1) \geqslant 1$. Koska $r_1 > 1$ ja $r_2 > 1$, ovat lausekkeet $(r_1 - 1)$ ja $(r_2 - 1)$ molemmat positiivisia, joten epäyhtälö on tosi.

Tekijöiden jakaminen ja suoritusaikavaativuus

Lause 4.5. Olkoon $N = r_1 r_2$ niin, että $r_1, r_2 \in \mathbb{N}$, $r_1 > 1$, $r_2 > 1$. Jos jokaisessa rekursioasteessa Cooleyn-Tukeyn algoritmista valitaan tehokkaampi kahdesta versiosta CT1 ja CT2, ja valitaan r_1, r_2 siten, että $\Omega(r_2) = 1$, tarvittavien kertolaskujen lukumäärän ylärajaksi tulosjoukon alkiota kohti tulee aina $g_N \leqslant \mathrm{Sopfr}(N)$ riippumatta siitä, mikä alkulukutekijöistä valitaan muuttujan r_2 arvoksi.

Todistus. Alkuaskel: Kun $\Omega(N) = 1$ eli N on alkuluku, tällöin lukua N ei voi jakaa kahteen ryhmään alkulukutekijöitä, eikä Cooleyn-Tukeyn algoritmia ole mahdollista hyödyntää. Tällöin käytetään jotain muuta algoritmia. Hitain mahdollinen algoritmi on naiivi DFT, jolle pätee $g_{\mathrm{DFT},N} = N$; todetaan siis $g_N \leqslant g_{\mathrm{DFT},N}$. Koska N sisältää vain yhden alkulukutekijän, pätee $N = \mathrm{Sopfr}(N)$. Tapauksessa $\Omega(N) = 1$ pätee siis $g_N \leqslant \mathrm{Sopfr}(N)$.

Induktioaskel: Oletetaan, että $J \in \mathbb{N}$ ja $g_N \leqslant \mathrm{Sopfr}(N)$ pätee aina, kun $\Omega(N) \leqslant J$. Induktioväite: Tarvittavien kertolaskujen lukumääräksi tulosjoukon alkiota kohti tulee $g_N \leqslant \mathrm{Sopfr}(N)$ silloinkin, kun $\Omega(N) = J + 1$.

Tapauksessa $\Omega(N) = J + 1$ voidaan luvun N tekijät jakaa mielivaltaisesti kahteen ryhmään r_1 ja r_2 siten, että $N = r_1 \cdot r_2$, $r_1 > 1$, $r_2 > 1$ sekä $\Omega(r_2) = 1$. Tämä on mahdollista, koska $\Omega(N) \geqslant 2$.

Koska $\Omega(r_2) \geqslant 1$, seuraa yhtälöstä $\Omega(N) = \Omega(r_1 r_2) = \Omega(r_1) + \Omega(r_2) = J + 1$, että $\Omega(r_1) \leqslant J$. Vastaavasti nähdään myös, että pätee $\Omega(r_2) \leqslant J$. Tällöin induktio-oletuksen perusteella pätee, että $g_{r_1} \leqslant \mathrm{Sopfr}(r_1)$ sekä $g_{r_2} \leqslant \mathrm{Sopfr}(r_2)$. Lisäksi koska $\Omega(r_2) = 1$, pätee myös $r_2 = \mathrm{Sopfr}(r_2)$. Näiden seikkojen perusteella saadaan seuraava tulos riippumatta muuttujien r_1 ja r_2 arvoista:

$$\begin{aligned} g_N &= \min\{g_{\mathrm{CT1},N},\, g_{\mathrm{CT2},N}\} \\ &= \min\{g_{r_1} + r_2,\, g_{r_1} + g_{r_2} + 1\} \\ &\leqslant \min\{\mathrm{Sopfr}(r_1) + \mathrm{Sopfr}(r_2),\, \mathrm{Sopfr}(r_1) + \mathrm{Sopfr}(r_2) + 1\} \\ &\leqslant \mathrm{Sopfr}(r_1) + \mathrm{Sopfr}(r_2) \\ &\leqslant \mathrm{Sopfr}(r_1 r_2) \\ &\leqslant \mathrm{Sopfr}(N). \end{aligned}$$

Johtopäätös: Ainakin siinä tilanteessa, että luvuksi r_2 valitaan jokin luvun N alkulukutekijä, seuraa alkuaskeleesta ja induktioaskeleesta II induktioperiaatteen nojalla, että $g_N \leqslant \mathrm{Sopfr}(N)$ kaikilla $N \in \mathbb{N}$ riippumatta siitä, miten tekijöiden jako lukujen r_1 ja r_2 välille tehdään. □

Vastaavasti $h_N \leqslant N \,\mathrm{Sopfr}(N)$. Lisäksi havaittiin, että näiden oletusten vallitessa ainoastaan yhden rekursion versiota (CT1) oli syytä käyttää.

Mikäli Cooleyn-Tukeyn algoritmi ei hyödynnä sellaisia mahdollisesti olemassaolevia algoritmeja, jotka toimisivat tehokkaasti myös alkulukupituisilla syötteillä, tällöin $g_N = \mathrm{Sopfr}(N)$. Myöhemmin luvussa 5.6 tämä oletus ei enää päde: $g_N < \mathrm{Sopfr}(N)$ on mahdollinen.

Edellä käytettiin ainoastaan yhden rekursion versiota. Seuraavaksi selvitetään kahden rekursion version laskennallinen vaativuus.

Lause 4.6. Olkoon edelleen $N = r_1 r_2$ niin, että $r_1, r_2 \in \mathbb{N}$, $r_1 > 1$, $r_2 > 1$. Jos jokaisessa rekursioasteessa Cooleyn-Tukeyn algoritmista valitaan kahden rekursion versio CT2, tarvittavien kertolaskujen lukumäärän ylärajaksi tulosjoukon alkiota kohti tulee aina $g_{\mathrm{CT2},N} \leqslant \mathrm{Sopfr}(N) + \Omega(N) - 1$ riippumatta siitä, miten tekijät jaetaan muuttujien r_1 ja r_2 kesken.

Todistus. Alkuaskel: Kun $\Omega(N) = 1$ eli N on alkuluku, tällöin lukua N ei voi jakaa kahteen ryhmään alkulukutekijöitä, eikä Cooleyn-Tukeyn algoritmia ole mahdollista hyödyntää. Tällöin käytetään jotain muuta algoritmia. Hitain mahdollinen algoritmi on naiivi DFT, jolle pätee $g_{\mathrm{DFT},N} = N$; todetaan siis $g_{\mathrm{CT2},N} \leqslant g_{\mathrm{DFT},N}$. Koska N sisältää vain yhden tekijän, pätee $N = \mathrm{Sopfr}(N)$. Niinpä tapauksessa $\Omega(N) = 1$ pätee ehto $g_{\mathrm{CT2},N} \leqslant \mathrm{Sopfr}(N)$, ja koska $\Omega(N) - 1 = 0$, pätee myös $g_{\mathrm{CT2},N} \leqslant \mathrm{Sopfr}(N) + \Omega(N) - 1$.

Induktioaskel: Oletetaan, että $J \in \mathbb{N}$ ja $g_{\mathrm{CT2},N} \leqslant \mathrm{Sopfr}(N) + \Omega(N) - 1$ pätee aina, kun $\Omega(N) \leqslant J$. Induktioväite: Tarvittavien kertolaskujen lukumääräksi tulosjoukon alkiota kohti tulee $g_{\mathrm{CT2},N} \leqslant \mathrm{Sopfr}(N) + \Omega(N) - 1$ silloinkin, kun $\Omega(N) = J + 1$.

Tapauksessa $\Omega(N) = J + 1$ voidaan luvun N tekijät jakaa mielivaltaisesti kahteen ryhmään r_1 ja r_2 siten, että $N = r_1 \cdot r_2$, $r_1 > 1$, $r_2 > 1$. Tämä on mahdollista, koska $\Omega(N) \geqslant 2$.

Koska $\Omega(r_2) \geqslant 1$, saadaan yhtälöstä $\Omega(N) = \Omega(r_1 r_2) = \Omega(r_1) + \Omega(r_2) = J + 1$ pääteltyä, että $\Omega(r_1) \leqslant J$. Vastaavasti nähdään myös, että pätee $\Omega(r_2) \leqslant J$. Tällöin induktio-oletuksen perusteella pätee, että $g_{\mathrm{CT2},r_1} \leqslant \mathrm{Sopfr}(r_1) + \Omega(r_1) - 1$ sekä vastaavasti $g_{\mathrm{CT2},r_2} \leqslant \mathrm{Sopfr}(r_2) + \Omega(r_2) - 1$. Näiden seikkojen perusteella saadaan seuraava tulos riippumatta muuttujien r_1 ja r_2 arvoista:

$$\begin{aligned} g_{\mathrm{CT2},N} &= g_{\mathrm{CT2},r_1} + g_{\mathrm{CT2},r_2} + 1 \\ &\leqslant \mathrm{Sopfr}(r_1) + \Omega(r_1) - 1 + \mathrm{Sopfr}(r_2) + \Omega(r_2) - 1 + 1 \\ &\leqslant \mathrm{Sopfr}(r_1 r_2) + \Omega(r_1 r_2) - 1 - 1 + 1 \\ &\leqslant \mathrm{Sopfr}(N) + \Omega(N) - 1. \end{aligned}$$

Johtopäätös: Alkuaskeleesta ja induktioaskeleesta seuraa II induktioperiaatteen nojalla, että algoritmia CT2 käytettäessä pätee $g_{\mathrm{CT2},N} \leqslant \mathrm{Sopfr}(N) + \Omega(N) - 1$ kaikilla $N \in \mathbb{N}$ riippumatta siitä, miten tekijöiden jako lukujen r_1 ja r_2 välille tehdään. □

Vastaavasti $h_{\mathrm{CT2},N} \leqslant (N\,\mathrm{Sopfr}(N) + N\,\Omega(N) - N)$.

Mikäli Cooleyn-Tukeyn algoritmi ei hyödynnä sellaisia mahdollisesti olemassaolevia algoritmeja, jotka toimisivat tehokkaasti myös alkulukupituisilla syötteillä, tällöin pätee myös $g_{\mathrm{CT2},N} = \mathrm{Sopfr}(N) + \Omega(N) - 1$.

Lause 4.7. Silloin, kun N koostuu pienistä alkulukutekijöistä, Cooleyn-Tukeyn FFT-algoritmin suoritusaikavaativuus on $\mathcal{O}(N \log N)$.

Todistus. Olkoon $\alpha \in \mathbb{N}$. Tällöin mille tahansa luvulle p pätee $p^\alpha = \overbrace{p \cdot p \cdot \cdots \cdot p}^{\alpha}$. Jos p on alkuluku, tästä voidaan silloin päätellä, että $\mathrm{Sopfr}(p^\alpha) = \alpha p$, missä $\alpha = \log_p(p^\alpha)$. Lauseiden 4.5 ja 4.6 mukaisesti jokaista FFT:n tulosalkiota kohti tarvitaan enintään $\mathrm{Sopfr}(N) + \Omega(N) - 1$ kertolaskua[11]. Mikäli luku N voidaan esittää muodossa p^α, tällöin $\Omega(N) = \alpha$, ja kertolaskuja tulosjoukon alkiota kohti tarvitaan enintään $\mathrm{Sopfr}(N) + \alpha - 1$, joka tarkoittaa vähemmän kuin:

$$\mathrm{Sopfr}(N) + \alpha = \mathrm{Sopfr}(p^\alpha) + \alpha = \alpha p + \alpha = (p+1)\log_p(N) = \frac{p+1}{\log p} \log N \text{ kappaletta.}$$

Laajennetaan seuraavaksi yllä esitetty päättely tapaukseen, jossa erilaisia alkulukutekijöitä on useampia kuin yksi. Merkitään $\omega(N)$ tarkoittamaan luvun N erilaisten alkulukutekijöiden lukumäärää ja $\Omega(N)$ tarkoittamaan luvun N kaikkien alkulukutekijöiden lukumäärää. Yleisemmässä muodossa jos $N = \prod_k {p_k}^{\alpha_k}$, $p_k > 1$, $\Omega(p_k) = 1$ kaikilla $0 \leqslant k < \omega(N)$, jolloin $\Omega(N) = \sum_k \alpha_k$, ja merkitään suurinta ja pienintä alkulukutekijää $\hat{p} = \max(p)$ sekä $\check{p} = \min(p)$, sekä määritellään g_N ja h_N samoin kuin sivulla 15, saadaan pääteltyä maksimisuoritusaika seuraavasti:

$$\begin{aligned}
g_N &\leqslant \mathrm{Sopfr}(N) + \Omega(N) - 1 \\
g_N &< \mathrm{Sopfr}(N) + \Omega(N) \\
g_N &< \mathrm{Sopfr}(\prod_k {p_k}^{\alpha_k}) + \sum_k \alpha_k \\
g_N &< \sum_k (\mathrm{Sopfr}({p_k}^{\alpha_k}) + \alpha_k) \\
g_N &< \sum_k (\alpha_k p_k + \alpha_k) \\
g_N &< \sum_k \frac{p_k + 1}{\log p_k} \log({p_k}^{\alpha_k}) \\
g_N &< \frac{\hat{p}+1}{\log \check{p}} \sum_k \log ({p_k}^{\alpha_k}) \\
g_N &< \frac{\hat{p}+1}{\log \check{p}} \log N.
\end{aligned}$$

Koska $h_N = N g_N$, saadaan $h_N < N \dfrac{\hat{p}+1}{\log \check{p}} \log N$.

Vakiokertoimia ja -lisäyksiä ei huomioida suoritusaikavaativuuslaskelmassa, joten mikäli suurin ja pienin tekijä $\hat{p}$ ja $\check{p}$ voidaan pienuutensa tähden katsoa vakioksi[12], esimerkiksi jos N on kahden potenssi ja $\hat{p} = \check{p} = 2$, muodostuu Cooleyn-Tukeyn FFT-algoritmin suoritusaikavaativuudeksi $\mathcal{O}(N \log N)$. □

[11] Todistuksen yleiskattavuuden kannalta tähän on valittu kahdesta eri ylärajasta suurempi.

[12] On tulkinnanvarainen kysymys, mitkä tekijät voidaan katsoa vakioiksi. Jos kaikki tekijät voidaan katsoa vakioiksi, silloin voidaan mm. johtaa todistus, että $\mathcal{O}(N^2) = \mathcal{O}(N \log N)$, mikä ei tietenkään pidä paikkaansa. Tässä tutkielmassa vakioiksi luettaviksi tekijöiksi ajatellaan implisiittisesti lähinnä yksinumeroiset tekijät.

4.2. Cooleyn-Tukeyn radix-2 -erikoistapaus

Palautetaan mieleen Cooleyn-Tukeyn algoritmin (nollan rekursion version) toimintaperiaate:

$$Y_{(n_0 r_2 + k_0)} = \sum_{k_1=0}^{r_1-1} x_{(k_1 r_2 + k_0)} \cdot e^{-i2\pi(n_0 r_2)k_1/N} \qquad \text{(4.2 kerrattu)}$$

$$X_{(n_1 r_1 + n_0)} = \sum_{k_0=0}^{r_2-1} Y_{(n_0 r_2 + k_0)} \cdot e^{-i2\pi(n_1 r_1 + n_0)k_0/N} \qquad \text{(4.3 kerrattu)}$$

Mikäli N on parillinen, voidaan valita $r_2 = 2$, $r_1 = \frac{N}{2}$. Tällöin kaava muuttuu seuraavasti:

$$\begin{aligned}
Y_{(2n_0+k_0)} &= \sum_{k_1=0}^{\frac{N}{2}-1} x_{(2k_1+k_0)} \cdot e^{-i2\pi(2n_0)k_1/N} \\
X_{(n_1 r_1 + n_0)} &= \sum_{k_0=0}^{1} Y_{(2n_0+k_0)} e^{-i2\pi(n_1 r_1 + n_0)k_0/N} \\
&= Y_{(2n_0+0)} e^{-i2\pi(n_1 r_1 + n_0)\cdot 0/N} + Y_{(2n_0+1)} e^{-i2\pi(n_1 r_1 + n_0)\cdot 1/N} \\
&= Y_{(2n_0+0)} + Y_{(2n_0+1)} e^{-i2\pi(n_1 r_1 + n_0)/N},
\end{aligned}$$

missä kaksi erillistä tapausta $k_0 = n_1 = 0$ ja $k_0 = n_1 = 1$ voidaan kirjoittaa auki seuraavasti:

$$\begin{aligned}
Y_{(2n_0+0)} &= \sum_{k_1=0}^{\frac{N}{2}-1} x_{(2k_1+0)} \cdot e^{-i2\pi(2n_0)k_1/N} \\
&= \sum_{k_1=0}^{\frac{N}{2}-1} x_{(2k_1+0)} \cdot e^{-i2\pi n_0 k_1/\left(\frac{N}{2}\right)} \text{ ja} \\
Y_{(2n_0+1)} &= \sum_{k_1=0}^{\frac{N}{2}-1} x_{(2k_1+1)} \cdot e^{-i2\pi(2n_0)k_1/N} \\
&= \sum_{k_1=0}^{\frac{N}{2}-1} x_{(2k_1+1)} \cdot e^{-i2\pi n_0 k_1/\left(\frac{N}{2}\right)}, \text{ sekä} \\
X_{n_0} &= Y_{(2n_0+0)} + Y_{(2n_0+1)} e^{-i2\pi n_0/N} \text{ ja} \\
X_{(\frac{N}{2}+n_0)} &= Y_{(2n_0+0)} + Y_{(2n_0+1)} e^{-i2\pi(\frac{N}{2}+n_0)/N} \\
&= Y_{(2n_0+0)} + Y_{(2n_0+1)} e^{-i(2\pi n_0/N+\pi)} \\
&= Y_{(2n_0+0)} - Y_{(2n_0+1)} e^{-i2\pi n_0/N}.
\end{aligned}$$

Merkitään sitten $E_j = Y_{(2j+0)}$ ilmaisemaan sarjan parillisia alkioita ja $O_j = Y_{(2j+1)}$ vastaavasti sarjan parittomia alkioita:

$$E_{n_0} = \left(\sum_{k_1=0}^{\frac{N}{2}-1} x_{(2k_1+0)} \cdot e^{-i2\pi n_0 k_1/\left(\frac{N}{2}\right)} \right) \tag{4.8}$$

$$O'_{n_0} = \left(\sum_{k_1=0}^{\frac{N}{2}-1} x_{(2k_1+1)} \cdot e^{-i2\pi n_0 k_1/\left(\frac{N}{2}\right)} \right) \tag{4.9}$$

$$\begin{aligned} O_{n_0} &= O'_{n_0} e^{-i2\pi n_0/N} \\ X_{n_0} &= E_{n_0} + O_{n_0} \\ X_{(\frac{N}{2}+n_0)} &= E_{n_0} - O_{n_0}. \end{aligned}$$

Lisäksi siirrettiin kerroin $e^{-i2\pi n_0/N}$ sarjan O laskentaan. Yhtälössä (4.8) suoritetaan normaali FFT tai DFT alkuperäisen sarjan x parillisille alkioille ja yhtälössä (4.9) vastaavasti sen parittomille alkioille. Tämän voi edelleen kirjoittaa muodossa, jossa ratkaisun rekursiivinen luonne on ilmeisempää, kun $k = 0, 1, \ldots, \frac{N}{2} - 1$:

$$\begin{aligned} E'_k &= x_{(2k+0)} \\ O'_k &= x_{(2k+1)} \\ E_k &= \mathrm{FFT}(E')_k \\ O_k &= \mathrm{FFT}(O')_k \cdot e^{-i2\pi k/N} \\ X_k &= E_k + O_k \\ X_{(\frac{N}{2}+k)} &= E_k - O_k. \end{aligned}$$

Algoritmi Radix2: Radix-2 -versio Cooleyn-Tukeyn algoritmista.

Radix2-algoritmissa parittomien alkioiden FFT kierretään viuhkamaisesti $0 \ldots \pi$ radiaania ympäri kompleksitasossa, ja tulos muodostetaan konkatenoimalla parillisten ja parittomien alkioiden FFT:iden summat ja erotukset. Tämän lähestymistavan etu yleiseen Cooleyn-Tukeyn tapaukseen verrattuna on, että se on varsin suoraviivausta toteuttaa.

Samaa menetelmää on myös mahdollista jatkaa isommilla jakajilla kuin kaksi. Esimerkiksi liitteessä IV on esitetty menettely radix-4 -erikoistapaukselle. Liitteessä V asiaa on jalostettu pitemmälle esittämällä yleistetty versio kaikista radix-erikoistapauksista.

Aikavaativuus

Radix2-algoritmissa kompleksikertolaskuja (merkitään tätä $h_{\mathrm{RX2},N}$) tarvitaan seuraavasti:

- tapauksessa $N = 2^m, m = 0$ tarvitaan $h_{\mathrm{RX2},N} = 0$ kertolaskua;
- tapauksessa $N = 2^m, m > 0$ tarvitaan $h_{\mathrm{RX2},N} = 2h_{\mathrm{RX2},N/2} + N/2$ kertolaskua.

Kun $N = 2^m$, pätee $\frac{N}{2} = 2^{m-1}$. Edellisen kahden rivin pohjalta voidaan siis muodostaa seuraava rekursiivinen lukujono:

$$\begin{aligned} a_0 &= 0 \\ a_m &= 2a_{m-1} + 2^{m-1}. \end{aligned}$$

Lukujonon yleinen jäsen on $a_m = 2^{m-1}m$, mistä seuraa $h_{\mathrm{RX2},2^m} = a_m = 2^{m-1}m$. Sijoittamalla $m = \log_2 N$ ja sieventämällä saadaan kertolaskujen lukumäärän lausekkeeksi seuraava:

$$h_{\mathrm{RX2},N} = \frac{1}{2}N\log_2 N = \frac{N\log N}{2\log 2}.$$

Vakiotermiä $\frac{1}{2\log 2}$ ei tietojenkäsittelytieteen periaatteiden mukaisesti huomioida aikavaativuuslaskelmassa [16]. Tällöin algoritmin suoritusaikavaativuus on $\mathcal{O}(N\log N)$, kuten yleisessäkin tapauksessa (lause 4.7).

Huolimatta siitä, että kaikki tähänastiset Cooleyn-Tukeyn algoritmin versiot kuuluvat suoritusaikavaativuuskategoriaan $\mathcal{O}(N\log N)$, on radix-2 -algoritmi neljä kertaa tehokkaampi kuin CT1-algoritmi (yhden rekursion versio Cooleyn-Tukeyn yleisestä algoritmista): Kun N on kahden potenssi, pätee lauseen 4.7 todistuksen ensimmäisen virkkeen sekä lauseen 4.5 nojalla $h_{\mathrm{CT1},N} = N\mathrm{Sopfr}(N) = 2N\log_2(N)$. Suoritusaikavaativuuksien suhteeksi saadaan siis $\frac{h_{\mathrm{RX2},N}}{h_{\mathrm{CT1},N}} = \frac{\frac{1}{2}N\log_2(N)}{2N\log_2(N)} = \frac{1}{4}$. Tästä nähdään, että Radix2-algoritmi selviää muunnoksesta neljä kertaa nopeammin CT1-algoritmiin nähden. Kaksinkertainen nopeutus selittyy sillä, että kertolasku suoritetaan vain parittomien alkioiden joukolle, ei parillisten alkioiden joukolle, hyödyntäen tietoa, että $e^{-i2\pi\cdot 0} = 1$. Toinen kaksinkertaistus selittyy sillä, että puolet kertolaskuista on korvattu vähennyslaskulla hyödyntäen yhtäläisyyttä $e^{-i2\pi(k+1/2)} = -e^{-i2\pi k}$. Yhteensä radix-2 -algoritmi saavuttaa siis nelinkertaisen nopeutuksen yleiseen algoritmiin verrattuna, kun syötteen pituus on kahden potenssi.

4.3. Raderin algoritmi

Yksi merkittävä puute Cooleyn ja Tukeyn algoritmissa on, että se tarjoaa nopeushyötyjä ainoastaan, mikäli syötteen koko voidaan jakaa pieniin alkulukutekijöihin; se ei toimi ollenkaan, mikäli syötteen koko N on alkuluku. Tähän epäkohtaan tarttuu Raderin algoritmi. Raderin algoritmi toimii *vain*, mikäli N on alkuluku [2].

Merkitään ($x \bmod y$) tarkoittamaan jakojäännöstä, kun luku x jaetaan luvulla y. Määritellään osajoukko $K = \{1, 2, \ldots, N-1\}$ ja valitaan kokonaisluku $g \in K$ siten, että funktio $f : K \to K, x \mapsto (g^x \bmod N)$ on bijektio[13].

Varsinainen FFT-laskenta näytesarjasta x taajuussarjaksi X tapahtuu muuttujan g sekä laskurin $k = 0, 1, \ldots, (N-2)$ avulla seuraavasti:

$$\begin{aligned}
\omega'_k &= e^{(-i2\pi g^{(N-1-k)}/N)} \\
Y_k &= x_{(g^k \bmod N)} \\
Z_k &= \mathrm{FFT}(Y)_k \\
W_k &= Z_k \ \mathrm{FFT}(\omega')_k + (N-1)x_0 \ \delta_{k,0} \\
X_0 &= Z_0 + x_0 \\
X_{\left(g^{(N-1-k)} \bmod N\right)} &= \mathrm{IFFT}(W)_k.
\end{aligned}$$

Raderin algoritmin hyöty perustuu siihen, että silloin, kun N on alkuluku, se mahdollistaa FFT-laskennan koolla $N-1$, joka ei ole alkuluku. Tällöin voidaan hyödyntää muiden algoritmien tuottamia optimointeja.

Tietokoneella on hyödyksi, mikäli osa laskennasta voidaan laskea etukäteen ja hyödyntää useissa samanmittaisissa muunnoksissa. Hyödyntäen lauseiden 2.4 ja 2.5 esittämiä suhteita FFT:n ja IFFT:n välillä voidaan muunnos kirjoittaa uudelleen jommalla kummalla alla esitetyistä muodoista, jotka eroavat toisistaan vain viimeisen rivin osalta. Tässä on myös otettu käyttöön apumuuttuja $g' = g^{N-2} \bmod N$, jolla laskenta $g^{(N-1-k)} \bmod N$ yksinkertaistuu muotoon $g'^k \bmod N$:

$$\begin{aligned}
\omega'_k &= e^{(-i2\pi g'^k/N)} \\
\omega_k &= \left(\tfrac{1}{N-1}\right) \mathrm{FFT}(\omega')_k \\
&\vdots \\
Y_k &= x_{(g^k \bmod N)} \\
Z_k &= \mathrm{FFT}(Y)_k \\
W_k &= Z_k \ \omega_k + x_0 \ \delta_{k,0} \\
X_0 &= Z_0 + x_0 \\
X_{\left(g'^k \bmod N\right)} &= \overline{\mathrm{FFT}(\overline{W})_k}
\end{aligned}
\qquad
\begin{aligned}
\omega'_k &= e^{(-i2\pi g'^k/N)} \\
\omega_k &= \left(\tfrac{1}{N-1}\right) \mathrm{FFT}(\omega')_k \\
&\vdots \\
Y_k &= x_{(g^k \bmod N)} \\
Z_k &= \mathrm{FFT}(Y)_k \\
W_k &= Z_k \ \omega_k + x_0 \ \delta_{k,0} \\
X_0 &= Z_0 + x_0 \\
X_{\left(g'^k \bmod N\right)} &= \mathrm{swap}\left(\mathrm{FFT}\left(\mathrm{swap}(W)\right)_k\right).
\end{aligned}$$

Nyt muuttujat g ja g' sekä vektori ω voidaan laskea etukäteen ja hyödyntää useissa eri muunnoksissa samanmittaiselle syötteelle. Lisäksi lukusarjan jakolasku arvolla $(N-1)$ tarvitsee suorittaa vain kerran, eikä erillistä toteutusta IFFT:lle tarvita.

[13] Algebran teoriassa tällaista alkiota g kutsutaan virittäjäksi: Mikäli kaikki osajoukon K jäsenet voidaan johtaa yhdestä alkiosta $g \in K$ toistamalla samaa laskutoimitusta tarpeeksi monta kertaa, tällöin g on kyseisen osajoukosta ja laskutoimituksesta koostuvan aliryhmän virittäjä [18].

4.4. Bluesteinin algoritmi

Bluesteinin algoritmi, toiselta nimeltään Chirp-Z -muunnos (CZT), on toinen FFT-algoritmi, joka toimii myös silloin, kun syötteen pituus N on alkuluku [3]. Algoritmi ei kuitenkaan edellytä sitä, että N olisi alkuluku. Tässä algoritmissa aloitetaan valitsemalla uudeksi kooksi *jokin* luku $m \geqslant 2N-1$. Tehokkaan toiminnan saavuttamiseksi kannattaa pyrkiä valitsemaan mahdollisimman pieni sellainen yhdistetty luku, joka koostuu pääasiassa pienistä alkulukutekijöistä[14]. Varsinainen FFT-laskenta näytesarjasta x taajuussarjaksi X tapahtuu muuttujan m sekä laskurien $k = 0, 1, \ldots, N-1$ sekä $j = 0, 1, \ldots, m-1$ avulla seuraavasti:

$$
\begin{aligned}
w_k &= e^{-i2\pi k^2/(2N)} \\
Y_j &= \begin{cases} x_j w_j & \text{kun } 0 \leqslant j < N, \\ 0 & \text{muuten;} \end{cases} \\
\omega'_j &= \begin{cases} w_j & \text{kun } 0 \leqslant j < N, \\ w_{(m-j)} & \text{kun } (m-N) < j < m, \\ 0 & \text{muuten;} \end{cases} \\
Z_j &= \mathrm{IFFT}(\omega')_j \ \cdot \ \mathrm{FFT}(Y)_j \\
X_k &= w_k \ \mathrm{IFFT}(Z)_k,
\end{aligned}
$$

jossa viimeinen IFFT lasketaan m-pituiselle syötteelle, mutta sen tuloksesta hyödynnetään vain N ensimmäistä alkiota. Hyödyntäen lauseen 2.4 tai 2.5 esittämiä suhdeita FFT:n ja IFFT:n välillä voidaan muunnos kirjoittaa uudelleen jommalla kummalla seuraavista tavoista, jotka eroavat toisistaan vain kahden viimeisen rivin osalta:

$$
\begin{aligned}
w_k &= e^{-i\pi k^2/N} \\
\omega'_j &= \begin{cases} w_j & \text{kun } 0 \leqslant j < N, \\ w_{(m-j)} & \text{kun } (m-N) < j < m, \\ 0 & \text{muuten;} \end{cases} \\
\omega_j &= \left(\tfrac{1}{m}\right) \mathrm{FFT}(\omega')_j \\
&\ \ \vdots \\
Y_j &= \begin{cases} x_j w_j & \text{kun } 0 \leqslant j < N, \\ 0 & \text{muuten;} \end{cases} \\
Z_j &= \omega_j \ \overline{\mathrm{FFT}(Y)_j} \\
X_k &= w_k \ \overline{\mathrm{FFT}(Z)_k}.
\end{aligned}
\qquad
\begin{aligned}
w_k &= e^{-i\pi k^2/N} \\
\omega'_j &= \begin{cases} w_j & \text{kun } 0 \leqslant j < N, \\ w_{(m-j)} & \text{kun } (m-N) < j < m, \\ 0 & \text{muuten;} \end{cases} \\
\omega_j &= \left(\tfrac{1}{m}\right) \mathrm{FFT}(\omega')_j \\
&\ \ \vdots \\
Y_j &= \begin{cases} x_j w_j & \text{kun } 0 \leqslant j < N, \\ 0 & \text{muuten;} \end{cases} \\
Z_j &= \omega_j \ \mathrm{swap}\left(\mathrm{FFT}(Y)_j\right) \\
X_k &= w_k \ \mathrm{swap}\left(\mathrm{FFT}(Z)_k\right).
\end{aligned}
$$

Nyt muuttuja m ja vektorit w ja ω voidaan laskea etukäteen ja hyödyntää useissa eri muunnoksissa samanmittaiselle syötteelle[15]. Lisäksi lukusarjan jakolasku arvolla m tarvitsee suorittaa vain kerran, eikä erillistä toteutusta IFFT:lle tarvita.

[14] Bluestein ehdotti, että muuttujan m arvoksi valitaan pienin kahden potenssi $\geqslant 2N-1$. Muun muassa FFTW-kirjaston toteutus sallii kuitenkin valitun luvun m tekijöiksi alkuluvun 2 lisäksi alkuluvut 3 sekä 5 (`dft/bluestein.c`, `kernel/primes.c`). Useampien tekijöiden salliminen pienentää luvun m keskimääräistä arvoa, mutta edellyttää, että FFT myös näillä ko'oilla on tehokas.

[15] Vektoria ω' ei tarvitse tallentaa, koska se toimii vain välituloksena.

5. Esimerkkitoteutus C++-ohjelmointikielellä

Tässä luvussa esitellään esimerkkitoteutus FFT-muunnoksesta C++-kielellä, tarkemmin sanottuna sen standardiversiolla C++20 [23]. Ohjelmanpätkien kommentit on kirjoitettu englanniksi, koska se on yleensä suositeltava ratkaisu niin julkiseksi tarkoitettujen ohjelmistojen kuin työympäristössäkin laadittavien lähdekoodien tapauksessa [22]. Esimerkkitoteutuksessa käytetään kompleksilukujen käsittelyyn standardikirjaston tietotyyppiä `std::complex<float>`. Tiedostossa `defs.hh` (listaus 1) määritellään tälle tyypille sekä siitä koostetuille taulukkolistoille lyhytnimet `complex` ja `cvector`, joita käytetään läpi tämän esimerkin.

Listaus 1: defs.hh

```hh
#pragma once
#include <vector>
#include <complex>

using complex = std::complex<float>;
using cvector = std::vector<complex>;
```

Listaus 2: exp.cc

```cc
#include <unordered_map>
#include <numbers>
#include <cmath>
#include <mutex>
#include "exp.hh"

// Calculate e^{-2πix/N} for x in 0..(N-1). Uses caching, converts result to exp_vector.
exp_vector ExpCircle(std::size_t N)
{
    static std::unordered_map<std::size_t, cvector> data;
    static std::mutex lock;
    std::unique_lock<std::mutex> lk(lock);

    // Look for N, 2N, 3N, ..., 10N in cache
    for(unsigned n=1; n<=10; ++n)
        if(auto j = data.find(N*n); j != data.end())
            return j->second;

    lk.unlock();
    // Precalculate e^{-2πix/N} for x in 0..(N-1)
    cvector vec(N);
    double mul = -2.0 * std::numbers::pi_v<double> / N;
    for(std::size_t a=0; a<N; ++a) vec[a] = std::polar(1.0, a*mul);
    lk.lock();
    // Insert to cache
    return data.emplace(N, std::move(vec)).first->second;
}
```

Diskreetissä ja nopeassa Fourier-muunnoksessa tarvitaan paljon $e^{-i2\pi k/N}$-muotoisia lausekkeita, joten niille kannattaa laatia lähdekoodissa oma funktionsa. Tämä tehdään

tiedostossa `exp.cc` (listaus 2). Jotta samoja e^x-taulukoita ei tarvitsisi alustaa useaan kertaan, koodi pitää kirjaa aiemmin luoduista taulukoista. Välivaiheet lasketaan `double`-tietotyypillä hyvän laskentatarkkuuden saavuttamiseksi, mutta tulokset tallennetaan `float`-laadulla. Rajapinta määriteltyyn funktioon `ExpCircle` on esitelty tiedostossa `exp.hh` (listaus 3).

Listaus 3: exp.hh

```
#pragma once /* Nonstandard pragma, but supported by practically all compilers. */
#include "defs.hh"
struct exp_vector
{
    const cvector* data = nullptr;
    exp_vector(const cvector& d) : data(&d) {}

    // Returns e^{-2πix/N} — requires only that data.size() is a multiple of N.
    complex operator() (std::size_t i, std::size_t N) const
    {
        return (*data)[(i %  N) * (data->size()/N)];
    }
};
// Creates e^{-2πix/N} for x in 0..(N-1)
exp_vector ExpCircle(std::size_t N);
```

5.1. Perus-DFT

Luvussa 2 esitetty diskreetin Fourier-muunnoksen kaava (2.1) kääntyy C++-koodiksi kahtena sisäkkäisenä `for`-silmukkana. Ulommassa silmukassa alustetaan summamuuttuja, johon sisemmässä silmukassa yhteenlasketaan syötteen ja $e^{-i2\pi ab/N}$ -lausekkeen tuloja. Nämä summat tallennetaan taulukkoon, jonka arvo lopuksi palautetaan. Tiedoston `dft.cc` sisältö on esitetty listauksessa 4.

Listaus 4: dft.cc

```
#include "dft.hh"
#include <numbers>
#include <tuple>
cvector DFT_simple(const cvector& input)
{
    const auto N = input.size();
    auto exps = ExpCircle(N);
    cvector result(N);

    for(std::size_t a = 0; a < N; ++a)
    {
        complex value {};
        for(std::size_t b = 0; b < N; ++b)
            value += input[b] * exps(a*b, N);
        result[a] = value;
    }
    return result;
}
```

Yleiskäyttöisempi versio `DFT`-funktiosta on esitelty listauksessa 6. Tämä versio mahdollistaa sen määrittämisen, millaisin välimatkoin näytteet on tallennettu taulukossa, ja vastaavasti millaisin välimatkoin taajuusarvot tallennetaan. Lisäksi funktiolle voi

antaa parametrinä e^x-arvoja sisältävän taulukon, jolloin sen ei välttämättä tarvitse laskea sitä uusiksi. Listauksessa on myös laskettu ”auki” erikoistapauksina syötekoot $N \leqslant 4$ lisätehokkuuden saavuttamiseksi. Tiedostoa `dft.cc` vastaava otsikkotiedosto `dft.hh` on esitetty listauksessa 5.

Listaus 5: dft.hh

```
#include "exp.hh"
cvector DFT(const cvector& input);
void DFT(const complex* input, complex* output,
         exp_vector exps,
         std::size_t N,
         std::size_t instride, std::size_t outstride);
}
```

Listaus 6: dft.cc (jatkuu)

```
void DFT(const complex* in, complex* out, exp_vector exps,
         std::size_t N, std::size_t is, std::size_t os)
{
    complex a,b,c,d, i{0,1}, p{-.5f, -std::numbers::sqrt3_v<float>/2}, q = std::conj(p);
    switch(N)
    {
        case 0: return;
        case 1: *out = *in; break;
        case 2: std::tie(a,b) = std::tuple(in[0], in[is]);
                std::tie(*out,out[os]) = std::tuple{a+b, a-b};
                break;
        case 3: std::tie(a,b,c) = std::tuple(in[0], in[is], in[is*2]);
                std::tie(*out,out[os],out[os*2])=std::tuple(a+b+c,a+b*p+c*q,a+b*q+c*p);
                break;
        case 4: std::tie(a,b,c,d) = std::tuple(in[0], in[is], in[is*2], in[is*3]);
                std::tie(*out,out[os],out[os*2], out[os*3])
                    = std::tuple(a+b+c+d, a-b*i-c+d*i, a-b+c-d, a+b*i-c-d*i);
                break;
        default:
            for(std::size_t a = 0; a < N; ++a)
            {
                complex value {};
                for(std::size_t b = 0; b < N; ++b)
                    value += input[b*ins] * exps(a*b, N);
                output[a*os] = value;
            }
    }
}
cvector DFT(const cvector& input)
{
    const auto N = input.size();
    auto exps = ExpCircle(N);
    cvector result(N);
    DFT(&input[0], &result[0], exps, N, 1, 1);
    return result;
}
```

5.2. Referenssitoteutus: FFTW

Jotta tässä tutkielmassa esitettyjen FFT- ja DFT-toteutuksien oikellisuutta olisi mahdollista arvioida, tarjotaan mukaan referenssitoteutus FFT:stä, joka tulee kirjastosta nimeltä FFTW. Kirjasto FFTW, *Fastest Fourier Transform in the West* on nopein tunnettu FFT-toteutus ja kansainvälisesti luotettavaksi todettu [11]. Sen lisääminen omaan ohjelmaan on verrattain yksinkertaista, ja esimerkki siitä nähdään listauksissa 7 ja 8. Tässä määritelty funktio `FFT_fftw` tarjoaa saman rajapinnan kuin listauksessa 6 määritelty funktio `DFT`: se ottaa parametrinään kompleksivektorin, joka kuvaa näytteitä, ja antaa paluuarvona syötteen kanssa saman kokoisen kompleksivektorin, joka kuvaa taajuusarvoja. Sana ”vektori” viittaa tässä yhteydessä C++-kielen `std::vector`-nimiseen tietorakenteeseen, joka tarkoittaa taulukkolistaa.

Listaus 7: fft_fftw.hh

```
#include "defs.hh"
cvector FFT_fftw(const cvector& input);
```

Listaus 8: fft_fftw.cc

```
#include "fft_fftw.hh"
#include <fftw3.h>

cvector FFT_fftw(const cvector& input)
{
    const std::size_t N = input.size();
    cvector output(N);
    fftwf_plan plan = fftwf_plan_dft_1d(N,
        (fftwf_complex*)&input[0],
        (fftwf_complex*)&output[0],
        FFTW_FORWARD, FFTW_ESTIMATE);
    fftwf_execute(plan);
    fftwf_destroy_plan(plan);
    return output;
}
```

On tosin syytä huomioida, että FFTW-kirjasto on tarkoitettu käytettäväksi siten, että suunnitelma (plan) laaditaan kerran, ja samalla suunnitelmalla voidaan suorittaa useita FFT-laskentoja peräkkäin. Tässä näin ei tehdä. Suunnitelman luominen on usein moninkertaisesti hitaampaa varsinaiseen muunnokseen nähden. Suunnitelman luontiin ja poistamiseen liittyvät funktiot eivät myöskään ole säieturvallisia [29].

5.3. Cooleyn-Tukeyn algoritmi

5.3.1. Radix-2 -tapaus

Seuraavaksi esitetään yksinkertainen esimerkkitoteutus luvussa 4.2 esitetystä algoritmista, jolla FFT voidaan laskea sellaiselle syötteelle, jonka koko on kahden potenssi.

Listaus 9: fft_radix2.hh

```
#include "defs.hh"
cvector FFT_radix2(const cvector& input);
```

Listauksessa 10, jota vastaava otsikkotiedosto on esitetty listauksessa 9, määritellään funktio `FFT_radix2`. Funktio ottaa syötteenään näytesarjan ja palauttaa taajuussarjan. Toteutus on suoraviivainen C++-käännös sivulla 21 esitetystä Radix2-algoritmista.

Listaus 10: fft_radix2.cc

```
#include "dft.hh"
#include "fft_radix2.hh"

cvector FFT_radix2(const cvector& input)
{
    const auto N = input.size();
    if(N <= 1) return input;

    std::size_t half = N/2;

    exp_vector exps = ExpCircle(N);
    cvector even(half), odd(half);
    for(std::size_t a = 0; a < half; ++a) even[a] = input[a*2];
    for(std::size_t a = 0; a < half; ++a) odd[a]  = input[a*2+1];
    even = FFT_radix2(even);
    odd  = FFT_radix2(odd);
    for(std::size_t a = 0; a < half; ++a) odd[a] *= exps(a,N);

    even.resize(N);
    for(std::size_t a = 0; a < half; ++a)
    {
        even[a+half] = even[a] - odd[a];
        even[a]      = even[a] + odd[a];
    }
    return even;
}
```

Listaus 11: fft_radix2.hh (jatkuu)

```
cvector FFT_radix2_fast(const cvector& input);

void FFT_radix2_fast(
    const complex* input,
    complex*       output,
    exp_vector     exps,
    std::size_t N, std::size_t instride, std::size_t outstride);
```

Listauksessa 10 esitetty `FFT_radix2` tekee lyhyydestään huolimatta paljon turhaa työtä: siinä tapahtuu runsaasti vektorien kopiointia sekä uusia e^x -joukkojen laskemisia. Tietojen kopiointia on kuitenkin mahdollista välttää samalla tavalla kuin jo edellä tehtiin `DFT`-funktion tapauksessa listauksessa 6. Listauksessa 12 on esitetty funktio `FFT_radix2_fast`, joka toteuttaa saman algoritmin kuin funktio `FFT_radix2`, mutta

välttää turhaa tietojen kopiointia. Listauksessa 11 ovat sitä vastaavat otsikkotiedoston rivit.

Listaus 12: fft_radix2.cc (jatkuu)

```
30 cvector FFT_radix2_fast(const cvector& input)
31 {
32     const auto N = input.size();
33     exp_vector exps = ExpCircle(N);
34     cvector output(N);
35     FFT_radix2_fast(&input[0], &output[0], exps, N, 1,1);
36     return output;
37 }
38
39 void FFT_radix2_fast(
40     const complex* input,
41     complex*       output,
42     exp_vector     exps,
43     std::size_t N, std::size_t instride, std::size_t outstride)
44 {
45     if(N <= 1) { output[0] = input[0]; return; }
46
47     // Because of cache efficiency reasons, use a smaller exps table in deeper recursions.
48     if(N>256 && exps.data->size() > N) { exps = ExpCircle(N); }
49
50     std::size_t half = N/2;
51     complex* even = output, *odd = output + half*outstride;
52     FFT_radix2_fast(input,         even, exps, half, instride*2, outstride);
53     FFT_radix2_fast(input+instride, odd, exps, half, instride*2, outstride);
54     for(std::size_t a = 0; a < half; ++a) odd[a*outstride] *= exps(a, N);
55
56     for(std::size_t a = 0; a < half; ++a)
57     {
58         complex part1 = even[a*outstride] + odd[a*outstride];
59         complex part2 = even[a*outstride] - odd[a*outstride];
60         even[a*outstride] = part1;
61         odd[a*outstride] = part2;
62     }
63 }
```

Seuraavaksi luodaan pääohjelma, jolla näitä algoritmeja voi testata sekä verrata toisiinsa. Pääohjelma, tiedosto `main.cc`, on esitetty listauksessa 13. Pääohjelma koostuu testisarjoista, joka sisältää kahden potensseja, kolmen potensseja, viiden potensseja, sekakertoimet 30, 900, 18900 ja 147000, jotka koostuvat alkulukujen 2, 3, 5 ja 7 erilaisista tuloista; sekä muutama sekalainen alkuluku väliltä 3 – 401987.

Kukin testisarja suoritetaan siten, että ensin arvotaan annetun luvun mittainen syöte, joka koostuu kompleksiluvuista muotoa e^{20ix}, missä x on deterministisen satunnaislukugeneraattorin tuottama kokonaisluku. Syöte muunnetaan taajuussarjaksi ensin referenssikirjastoa FFTW käyttäen. Sen jälkeen kukin tarjolla oleva algoritmi testataan vuorollaan siten, että algoritmin tuotosta syötteellä verrataan referenssikirjaston tuotokseen laskemalla keskimääräinen neliövirhe. Mikäli virhe ylittää tietyn kynnyksen, se viittaa siihen, ettei algoritmi toimi oikein, ja tällöin tulostetaan varoitusviesti. Algoritmin suoritusaika mitataan myös ja tulostetaan. Mikäli algoritmia ei voi ajaa annetulla syötteellä esim. siksi, koska algoritmi tukee vain kahden potenssisia syötekokoja, kyseinen algoritmi ohitetaan siinä testissä.

Listaus 13: main.cc

```cpp
#include "fft_fftw.hh"
#include "fft_radix2.hh"
#include <chrono>
#include <cstdio>
#include <cstdlib>
#include <type_traits>

static void RunTest(cvector (*func)(const cvector&),
                    const cvector& input,
                    const cvector& expected,
                    const char* what)
{
    auto begin = std::chrono::steady_clock::now();
    cvector output = func(input);
    auto end   = std::chrono::steady_clock::now();
    double duration = std::chrono::duration<double>(end-begin).count();
    double diff = 0;
    // Measure mean squared difference between reference output and the testcase
    for(std::size_t a=0; a<output.size(); ++a)
    {
        auto absdiff = std::abs(output[a]) - std::abs(expected[a]);
        auto argdiff = std::arg(output[a]) - std::arg(expected[a]);
        diff += absdiff*absdiff + argdiff*argdiff;
    }
    diff /= output.size();
    const char* warning = diff > 1e-3f ? "\33[31m - WARNING\33[m" : "";
    std::printf("    %16.4f ms (%s) - Mean squared difference: %.3e%s\n",
                duration*1e3, what, diff, warning);
    std::fflush(stdout);
}

static void Measure(std::size_t N)
{
    cvector test(N);
    for(std::size_t a = 0; a < N; ++a) { test[a] = std::polar(1.f, 20.f * std::rand()); }
    cvector expected = FFT_fftw(test);

    std::printf("  Measuring N = %zu...\n", N);
    RunTest(FFT_fftw, test, expected, "FFT (Reference version from FFTW3)");
    RunTest(DFT,      test, expected, "DFT");
    if( (N & (N-1)) == 0) // If N is a power of two
    {
        RunTest(FFT_radix2, test, expected, "FFT (Cooley-Tukey radix-2 version)");
    }
}

template<typename... I>
static void RunTests(const char* what, I... n) requires (std::is_integral_v<I> && ...)
{
    std::printf("Testing with %s\n", what);
    (Measure(n), ...);
}
int main()
{
    RunTests("powers of 2", 16, 256, 4096, 16384, 65536, 262144);
    RunTests("powers of 3", 9, 81, 729, 6561, 59049, 177147);
    RunTests("powers of 5", 25, 625, 15625, 78125);
    RunTests("mixed factors", 30, 900, 18900, 147000);
    RunTests("primes", 3,7, 17, 173,971, 2113,5393, 37813,59359, 139901,200183,401987);
}
```

5.3.2. Yleinen tapaus

Luvussa 5.3.1 käsiteltiin Cooleyn-Tukeyn algoritmista erikoistapaus, joka soveltuu käytettäväksi, kun syötteen pituus N on kahdella jaollinen ja on erityisen tehokas, jos N on kahden potenssi. Alkuperäinen Cooleyn-Tukeyn julkaisu esitteli kuitenkin menetelmän, joka soveltuu käytettäväksi, mikäli luku N koostuu ylipäänsä vähintään kahdesta tekijästä. Ennen kuin algoritmi voidaan toteuttaa, tarvitaan funktioita, jotka etsivät luvun tekijöitä. Listauksessa 14 esitellään tekijöihin jakoon liittyviä funktioita. Tiedostossa `factor.cc` (listaus 15) on esitetty yksinkertainen toteutus näille funktioille. Tehokkaampi toteutus on esitetty liitteessä I.

Listaus 14: factor.hh

```
#include <cstddef> /* for std::size_t */

// Finds the smallest factor > 1 of the given integer. If N is prime or ⩽ 1, returns N.
std::size_t smallest_factor(std::size_t N);

// Builds a list of unique factors of N in tgt[]. Returns number of primes written.
std::size_t get_factors(std::size_t N, std::size_t* tgt, std::size_t limit);

// Returns true if N can be written as 2^a 3^b 5^c 7^d for some a,b,c,d ∈ ℕ0.
bool small_factors_only(std::size_t N, std::size_t upto=7);
```

Listaus 15: factor.cc

```
#include <cmath>
#include "factor.hh"
std::size_t smallest_factor(std::size_t N)
{
    for(std::size_t fac=2; fac*fac < N; fac += (fac<3?1:2))
        if(N %  fac == 0)
            return fac;
    return N;
}
std::size_t get_factors(std::size_t N, std::size_t* target, std::size_t limit)
{
    for(std::size_t f,orig=N,count=0,prev=0; ; N/=f)
    {
        if(f=smallest_factor(N); count >= limit || f <= 1 || f == orig)
            return count;
        else if(f != prev)
            target[count++] = prev = f;
    }
}
bool small_factors_only(std::size_t N, std::size_t upto)
{
    for(std::size_t f; ; N/=f)
        if(f = smallest_factor(N); f < 2 || f > upto)
            return f == 1;
}
```

Sen jälkeen määritellään tiedostoon `fft_tukey.hh` funktio Cooleyn-Tukeyn algoritmille (listaus 16), ja sen toteutus tiedostoon `fft_tukey.cc` (listaus 17). Toteutus on suora C++-käännös algoritmeista CT1 ja CT2. Laskennassa hyödynnetään vain yhtä rekursiota. Mikäli annetun syötteen kokoa ei voi jakaa kahteen tekijään, tällöin laskenta suoritetaan `DFT`-funktiolla.

Listaus 16: fft_tukey.hh

```
#include "defs.hh"
#include "exp.hh"

cvector FFT_tukey(const cvector& input);
cvector FFT_tukey_fast(const cvector& input);
void FFT_tukey_fast(
    const complex* input,
    complex*       output,
    exp_vector     exps,
    std::size_t N, std::size_t instride, std::size_t outstride);
```

Listaus 17: fft_tukey.cc

```
#include "dft.hh"
#include "fft_tukey.hh"
#include "factor.hh"

cvector FFT_tukey(const cvector& input, exp_vector exps)
{
    const auto N = input.size(), r2 = smallest_factor(N), r1 = N/r2;
    if(r2 <= 1 || N <= 4) { return DFT(input); } // Factorization failed, revert to DFT.

    cvector result(N), temp(N);
    for(std::size_t k0 = 0; k0 < r2; ++k0)
    {
        cvector slice(r1);
        for(std::size_t k1 = 0; k1 < r1; ++k1) slice[k1] = input[k1 * r2 + k0];
        auto subresult = FFT_tukey(slice);
        for(std::size_t k1 = 0; k1 < r1; ++k1) temp[k0*r1 + k1] = subresult[k1];
    }
#if 1 /* OPTION 1: One recursion */
    for(std::size_t n1 = 0; n1 < r2; ++n1)
        for(std::size_t n0 = 0; n0 < r1; ++n0)
        {
            complex value = 0;
            for(std::size_t k0 = 0; k0 < r2; ++k0)
                value += buf[k0 + n0*r2] * exps((n1*r1+n0)*k0, N);
            result[(n1*r1 + n0)*outstride] = value;
        }
#else /* OPTION 2: Two recursions */
    for(std::size_t n0 = 0; n0 < r1; ++n0)
        for(std::size_t k0 = 0; k0 < r2; ++k0)
            temp[k0 + n0*r2] *= exps(k0*n0, N);
    for(std::size_t n0 = 0; n0 < r1; ++n0)
    {
        cvector slice(r2);
        for(std::size_t k0 = 0; k0 < r2; ++k0) slice[k0] = temp[n0 * r2 + k0];
        auto subresult = FFT_tukey(slice);
        for(std::size_t k0 = 0; k0 < r2; ++k0) result[n0 + k0*r1] = subresult[k0];
    }
#endif
    return result;
}
```

Listauksessa 17 esiintyi `#if`-määritys, jolla valittiin kahdesta koodivariantista, joista yksi suorittaa kaksi rekursiota ja toinen suorittaa yhden. Yleisessä tapauksessa kahden rekursion malli on tehokkaampi, mutta tätä asiaa tutkitaan tarkemmin luvussa 5.6 `FFT_any_fast`-funktion yhteydessä.

Kuten edellä tehtiin funktion `FFT_radix2` kanssa, myös yleisen Cooleyn-Tukeyn algoritmin toteuttavasta funktiosta voidaan laatia nopeampi versio, jossa vältetään turhaa tietojoukkojen kopiointia (listaus 18). Täysin apuvektoreilta ei kuitenkaan onnistuta välttymään.

Listaus 18: fft_tukey.cc (jatkuu)

```
32 #include "fft_radix2.hh"
33
34 void FFT_tukey_fast(
35     const complex* input,
36     complex*       output,
37     exp_vector     exps,
38     std::size_t N,
39     std::size_t instride, std::size_t outstride)
40 {
41     const auto r2 = smallest_factor(N), r1 = N/r2;
42
43     // Factorization failed? Use DFT
44     if(r2 == 1) { DFT(input, output, exps, N, instride, outstride); return; }
45
46     cvector temp(N);
47     for(std::size_t k0 = 0; k0 < r2; ++k0) // Y_{kr2+(0,1,...,r1-1)} = FFT(x_{(k+r2(0,1,...,r1-1))})_k
48         FFT_tukey_fast(input + k0*instride, &temp[k0], exps, r1, r2*instride, r2);
49
50     for(std::size_t n0 = 0; n0 < r1; ++n0)
51         for(std::size_t k0 = 0; k0 < r2; ++k0)
52             temp[k0 + n0*r2] *= exps(k0*n0, N);
53
54     for(std::size_t n0 = 0; n0 < r1; ++n0) // X_n = sum_{k=0}^{r2-1} Y_{(k+r2(n mod r2))} e^{-i2pi nk/N}
55         FFT_tukey_fast(&temp[n0*r2], &output[n0*outstride], exps, r2, 1,r1*outstride);
56 }
57
58 cvector FFT_tukey_fast(const cvector& input)
59 {
60     const auto N = input.size();
61     exp_vector exps = ExpCircle(N);
62     cvector result(N);
63     FFT_tukey_fast(&input[0], &result[0], exps, N, 1, 1);
64     return result;
65 }
```

5.4. Raderin algoritmi

Luvussa 4.3 kuvattu Raderin algoritmi kääntyy C++-ohjelmakoodiksi siten, että aloitetaan laatimalla rutiini, joka laskee vakiot g ja g' sekä vektorin ω jonkin syötekoon N perusteella. Nämä tiedot tallennetaan hajautustauluun, joka toimii välimuistina. Näin samaa laskentaa voidaan hyödyntää myöhemmin, jos lasketaan FFT saman kokoisella syötteellä uudestaan. Rutiini on esitetty listauksessa 19.

Listaus 19: fft_rader.cc

```
#include <mutex>
#include <algorithm>
#include <unordered_map>
#include "fft_rader.hh"
#include "fft_any.hh"
#include "factor.hh"

struct RaderConfiguration
{
    cvector ω;
    std::size_t g;
    std::size_t ginv;
};
static const RaderConfiguration& configure_rader(std::size_t N)
{
    static std::unordered_map<std::size_t, RaderConfiguration> data;
    static std::mutex lock;
    std::unique_lock<std::mutex> lk(lock);
    if(auto i = data.find(N); i != data.end()) return i->second;

    lk.unlock();
    auto [g, ginv] = find_generator(N); // Note: (g · g') mod N = 1

    exp_vector exps = ExpCircle(N);

    cvector omega(N-1); // ω'
    for(std::size_t gpower=1, k=0; k < N-1; ++k, gpower *= ginv, gpower %= N)
        omega[k] = exps(gpower,N); // ω'_k = e^{-i2πg'^k/N}

    RaderConfiguration cfg;
    cfg.g    = g;
    cfg.ginv = ginv;
    cfg.ω    = FFT_any_fast(omega);
    for(auto& f: cfg.ω) f /= float(N-1); // ω = 1/(N-1) FFT(ω')
    lk.lock();
    return data.emplace(N, std::move(cfg)).first->second;
}
```

Laskennassa käytetty funktio `find_generator`, joka määrittää arvon Raderin FFT:ssä tarvittavalla virittäjälle g, on esitetty listauksessa 21. Virittäjä valitaan etsimällä järjestyksessä pienin sellainen $n \in \mathbb{N}, n \geqslant 2$, jolla pätee $n^{(N-1)/p_m} \bmod N \neq 1$ kaikilla $m \in \{ 1, \ldots, \omega(N-1) \}$, missä p_m on luvun $(N-1)$ m's erilainen alkulukutekijä ja $\omega(N)$ on luvun N sisältämien erilaisten alkulukutekijöiden lukumäärä.

Varsinainen osuus algoritmista on esitetty listauksessa 20. Kyseessä on suoraviivainen C++-tulkkaus luvussa 4.3 esitetyistä kaavoista. Toteutuksessa on otettu paljon mallia FFTW-kirjaston vastaavasta C-kielisestä toteutuksesta. Toteutusta vastaava otsikkotiedosto `fft_rader.hh` on esitetty listauksessa 22. Laskenta $(N-1)$ -kokoiselle osajonolle syötteestä suoritetaan `FFT_any_fast`-funktiolla, joka esitetään myöhemmin luvussa 5.6.

Listaus 20: fft_rader.cc (jatkuu)

```
63  void FFT_rader(const complex* input, complex* output, std::size_t N,
64                 std::size_t instride, std::size_t outstride)
65  {
66      const auto& cfg = configure_rader(N);
67      const std::size_t g = cfg.g, ginv = cfg.ginv; // g = g, ginv = g'.
68      cvector buf(N-1); // This buffer is first used as Y and later as W'.
69
70      // Permutation of input
71      const complex in0 = input[0];
72      for(std::size_t gpower=1, k=0; k<N-1; ++k, gpower *= g, gpower %= N)
73          buf[k] = input[gpower*instride]; // Y_k = x_(g^k mod N)
74
75      // FFT
76      exp_vector exps = ExpCircle(N-1);
77      FFT_any_fast(&buf[0], output + outstride, exps, N-1, 1, outstride); // Z_k = FFT(Y)_k
78
79      // Set output DC component:
80      output[0] = in0 + output[outstride]; // X_0 = Z_0 + x_0
81
82      // Multiply by omega.
83      for(std::size_t k=0; k<N-1; ++k) // Note: (k+1) is used because we temporarily use
84      {                        // output[1...] as W and we don't want to clobber output[0].
85          auto f = output[(k+1)*outstride] * cfg.ω[k] + (k ? 0 : in0);
86          output[(k+1)*outstride] = std::conj(f); // W_k = conj(Z_k ω_k + x_0 δ_{0,k})
87      }
88
89      // IFFT (which is calculated by regular FFT)
90      FFT_any_fast(output+outstride, &buf[0], exps, N-1, outstride, 1); // W'_k = FFT(W)_k
91
92      // Inverse permutation
93      for(std::size_t gpower=1, k=0; k<N-1; ++k, gpower *= ginv, gpower %= N)
94          output[gpower*outstride] = std::conj(buf[k]); // X_(g'^k mod N) = conj(W'_k)
95
96      // The two conjugates and the use of FFT instead of IFFT are explained in lemma 2.4.
97  }
98
99  cvector FFT_rader(const cvector& input)
100 {
101     const auto N = input.size();
102     cvector result(N);
103     FFT_rader(&input[0], &result[0], N, 1, 1);
104     return result;
105 }
```

Listaus 21: fft_rader.cc (rivit 8–30)

```
8  // Compute n^m mod p where m ⩾ 0 and p > 0.
9  static std::size_t powermod(std::size_t n, std::size_t m, std::size_t p)
10 {
11     if(m == 0) return 1;
12     if(m %  2 == 0) { n = powermod(n, m/2, p); return (n*n) %  p; }
13     return (n * powermod(n, m-1, p)) %  p;
14 }
15
16 // Find g = smallest n ∈ ℕ, n ⩾ 2 for which n^((N-1)/p_m) ∉ [1]_N for all factors p_m of N-1.
17 static std::pair<std::size_t, std::size_t> find_generator(std::size_t prime)
18 {
19     // log2 of the product of first 16 primes ≈  64.82. Thus, if size_t ⩽  64 bits,
20     const std::size_t maxfac = 16;                    // then size 16 array is enough.
21     if(prime == 2) return { 1, 1 };                   // prime = N, pm1 = N-1.
22     std::size_t factors[maxfac], pm1 = prime-1, g = 2; // Smallest possible g = 2.
23     std::size_t count = get_factors(pm1, factors, maxfac);
24     for(std::size_t m=0; m<count; )
25         if(powermod(g, pm1 / factors[m], prime) == 1)// Is g^((N-1)/p_m) in [1]_N?
26             { ++g; m = 0; }                           // If so, increment g and retry.
27         else
28             { ++m; }
29     return { g, powermod(g, prime-2, prime) };      // Return g and g' = (g^(N-2) mod N).
30 }
```

Listaus 22: fft_rader.hh

```
1 #include "defs.hh"
2
3 void FFT_rader(const complex* input, complex* output, std::size_t N,
4                std::size_t instride, std::size_t outstride);
5
6 cvector FFT_rader(const cvector& input);
```

5.5. Bluesteinin algoritmi

Luvussa 4.4 kuvattu Bluesteinin algoritmi kääntyy C++-ohjelmakoodiksi siten, että aloitetaan laatimalla rutiini, joka määrittää vakion m sekä laskee vektorit w ja ω jonkin syötekoon N perusteella. Nämä tiedot tallennetaan hajautustauluun, joka toimii välimuistina. Näin samaa laskentaa voidaan hyödyntää myöhemmin, jos lasketaan FFT saman kokoisella syötteellä uudestaan. Rutiini on esitetty listauksessa 23. Laskennan alussa määritetään arvo m testaamalla järjestyksessä kokonaislukuja arvosta $(2N-1)$ alkaen ja valitsemalla sellainen, joka koostuu pelkästään tekijöistä $2, 3, \ldots,$ `maxfac`. Tämä on järkevää paitsi tehokkuuden kannalta, myös sen takaamiseksi, ettei FFT-laskenta päätyisi ikuiseen silmukkaan, jossa syötteen koko kasvaa hallitsemattomasti: on esimerkiksi mahdollista, että myös arvo $2N-1$ on alkuluku[16], jolloin myös sisempi FFT pitäisi

[16] Esimerkiksi jos $N = 1531$, niin $2N-1 = 3061$, joka on alkuluku; $2 \cdot 3061 - 1 = 6121$, sekin on alkuluku; $2 \cdot 6121 - 1 = 12241$ on myös alkuluku; samoin $2 \cdot 12241 - 1 = 24481$. Tällainen alkulukuketju on nimeltään kakkostyypin Cunninghamin ketju [9]. Tämä tutkielman kirjoittajan keksimä ketju on viisi lukua pitkä, mutta pisin tunnettu ketju on tällä hetkellä 19 alkulukua pitkä.

laskea Bluesteinin (tai Raderin) algoritmilla, ja tämän kohdalla pätisi mahdollisesti taas sama, jne.

Listaus 23: fft_bluestein.cc

```
#include <mutex>
#include <limits>
#include <algorithm>
#include <unordered_map>
#include "fft_bluestein.hh"
#include "fft_any.hh"
#include "factor.hh"

struct BluesteinConfiguration
{
    cvector w, ω;
    std::size_t m; // convolution size
};
static const BluesteinConfiguration&
    configure_bluestein(std::size_t N, unsigned char maxfac, std::size_t m = 0)
{
    constexpr unsigned B = std::numeric_limits<unsigned char>::digits, W = sizeof(std::size_t)*B;
    std::size_t key = N | ((m?0:maxfac) << (W-B));
    static std::unordered_map<std::size_t, BluesteinConfiguration> data;
    static std::mutex lock;
    std::unique_lock<std::mutex> lk(lock);
    if(auto i = data.find(key); i != data.end()) return i->second;

    lk.unlock();
    std::size_t nb = 2*N - 1;                   // Choose a fast-to-evaluate m ⩾ 2N − 1
    if(m) nb = m;
    else if(maxfac == 2)                        // Allowed factors: 2; choose m = 2^⌈log2(2N−1)⌉
        for(unsigned a=1; a<W; a<<=1) { nb = 1 + ((nb-1) | ((nb-1) >> a)); }
    else                                        // Allowed factors: 2 3 5 7 .. up to maxfac
        while(!small_factors_only(nb, maxfac)) { ++nb; } // Use a loop to find the smallest value

    cvector w(N);
    exp_vector exps = ExpCircle(2*N);
    for(std::size_t k=0; k<N; ++k) { w[k] = exps(k*k, 2*N); } // w_k = e^{−i2π k²/2N}

    cvector ω = w;
    for(auto& f: ω) f /= float(nb);                     // ω'_k       = w_k/m,
    ω.resize(nb);
    for(std::size_t k=1; k<N; ++k) { ω[nb-k] = ω[k]; } // ω'_{(m−k)} = w_k/m  if k > 0,
                                                        // The rest of ω' is zero
    ω = FFT_any_fast(ω);                                // ω = FFT(ω')

    BluesteinConfiguration cfg;
    cfg.m = nb;
    cfg.w = std::move(w);
    cfg.ω = std::move(ω);

    lk.lock();
    return data.emplace(key, std::move(cfg)).first->second;
}
```

Varsinainen osuus algoritmista on esitetty listauksessa 24. Kyseessä on suoraviivainen C++-tulkkaus luvussa 4.4 esitetyistä kaavoista. Toteutuksessa on otettu paljon mallia FFTW-kirjaston vastaavasta C-kielisestä toteutuksesta. Laskenta m -kokoiselle sykliselle konvoluutiolle suoritetaan `FFT_any_fast`-funktiolla, joka esitetään luvussa 5.6.

Listaus 24: fft_bluestein.cc (jatkuu)

```
50 void FFT_bluestein(const complex* input, complex* output, std::size_t N,
51                    std::size_t instride, std::size_t outstride,
52                    unsigned char maxfac, std::size_t m)
53 {
54     const auto& cfg = configure_bluestein(N, maxfac, m);
55     const std::size_t m = cfg.m; // m = m
56     cvector buf(m*2); // The first half of this zero-initialized buffer is first
57     // used as Y and later used as X'. The second half is used as Z' and as Z.
58     complex* buf1 = &buf[0], *buf2 = &buf[m]; // Pointers to both halves
59
60     // Multiply input by Bluestein sequence
61     for(std::size_t k=0; k<N; ++k)
62         buf1[k] = input[k*instride] * cfg.w[k]; // Y_k = x_k w_k  (rest is zero up to m-1)
63
64     // Convolution: FFT
65     exp_vector exps = ExpCircle(m);
66     FFT_any_fast(buf1, buf2, exps, m, 1,1);    // Z'_j = FFT(Y)_j
67
68     // Convolution: pointwise multiplication
69     for(std::size_t j=0; j<m; ++j)
70         buf2[j] = std::conj(buf2[j]) * cfg.ω[j]; // Z_j = ω_j conj(Z'_j)
71
72     // Convolution: IFFT (which is calculated by regular FFT)
73     FFT_any_fast(buf2, buf1, exps, m, 1,1);    // X'_k = FFT(Z)_k
74
75     // Multiply output by Bluestein sequence
76     for(std::size_t k=0; k<N; ++k)
77         output[k*outstride] = std::conj(buf1[k]) * cfg.w[k]; // X_k = w_k conj(X'_k)
78
79     // The two conjugates and the use of FFT instead of IFFT are explained in lemma 2.4.
80 }
81
82 #define BlueFunc(name, maxfac) \
83     cvector name(const cvector& input) \
84     { \
85         const auto N = input.size(); \
86         cvector result(N); \
87         FFT_bluestein(&input[0], &result[0], N, 1, 1, maxfac); \
88         return result; \
89     }
90 BlueFunc(FFT_Bluestein,        7)
91 BlueFunc(FFT_Bluestein_pow2,   2)
92 BlueFunc(FFT_Bluestein_fac2_3, 3)
```

Tässä toteutettiin kolme funktiota, jotka eroavat siinä, mistä tekijöistä ne sallivat apumuuttujan m koostuvan: 2, 3, 5 ja 7, vain 2, vaiko 2 ja 3. Vastaava otsikkotiedosto `fft_bluestein.hh` on esitetty listauksessa 25. Vapaamuotoinen funktio sallii myös sen,

että sille annetaan parametrinä konvoluution m koko suoraan, jolloin funktio `configure_bluestein` ei yritä määrittää sitä.

Listaus 25: fft_bluestein.hh

```
#include "defs.hh"
void FFT_bluestein(const complex* input, complex* output, std::size_t N,
                   std::size_t instride, std::size_t outstride,
                   unsigned char maxfac, std::size_t m = 0);
cvector FFT_bluestein(const cvector& input);
cvector FFT_bluestein_pow2(const cvector& input);
cvector FFT_bluestein_fac2_3(const cvector& input);
```

5.6. Yhdistetty algoritmi

Seuraavaksi luodaan yleinen funktio `FFT_any_fast`, joka pyrkii käyttämään mitä tahansa saatavilla olevaa menetelmää FFT:n laskemiseksi (lukuunottamatta valmiita kirjastoja kuten FFTW), kunhan se vain on nopea.

Tätä varten muodostetaan luettelo kaikista toteutetuista menetelmistä. Menetelmät on listattu taulukossa 2, ja taulukkoa vastaavat C++-määrittelyt on esitetty listauksessa 26. Bluesteinin algoritmista testataan useita eri vaihtoehtoisia arvoja konvoluutiopituusmuuttujalle m. Arvot eroavat sallittujen tekijöiden osalta, kuitenkin niin, että $m \geqslant 2N-1$. Tukeyn yleisestä algoritmista testataan kaikki mahdolliset kombinaatiot jakaa luvun N tekijät muuttujiin r_1 ja r_2 niin, että $N = r_1 r_2$, sekä lisäksi yhden rekursion ja kahden rekursion menetelmät, jotka esiteltiin listauksessa 17.

Nimi	Rekursiot	Kertolask.	Yhteenlask.	Lisätila	Ehdot
Tukey (Radix-2)	$2 \times (N/2)$	N	—	—	$(N/2) \in \mathbb{N}$
Tukey (1 rekursio)	$r_2 \times r_1$	Nr_2	Nr_2	$1 \times N$	$\Omega(N) \neq 1$
Tukey (2 rekursiota)	$r_2 \times r_1$ ja $r_1 \times r_2$	N	—	$1 \times N$	$\Omega(N) \neq 1$
Rader	$2 \times (N-1)$	$N-1$	—	$1 \times (N-1)$	$\Omega(N) = 1$
DFT (optimoitu)	—	$4\delta_{N,3}$	$N^2 - N$	—	$N \leqslant 4$
DFT (yleinen)	—	N^2	N^2	—	$N > 4$
Bluestein	$2 \times m$	$2N + m$	—	$2 \times m$	—

Taulukko 2: Toteutetut FFT-menetelmät ominaisuuksineen.
Merkintä $\delta_{x,y}$ on sivulla 6 selitetty Kroneckerin delta.

Listaus 26: fft_any.cc (osa 1)

```
/* o(name, r1tab,r2val,  rec1val,rec1mul, rec2val,rec2mul, mul,add,temp, condition) */
#define LIST_METHODS(o) \
  o(Radix2,     z,0,      N/2,2, 0,0,   N/2, N,   0, smallest==2) \
  o(Tukey_Rec1, fac,N/r1, r1,r2, 0,0,   N*r2,N*r2,1, smallest!=N) \
  o(Tukey_Rec2, fac,N/r1, r1,r2, r2,r1, N,   0,   1, smallest!=N) \
  o(Rader,      z,0,      N-1,2, 0,0,   N-1, 0,   1, smallest==N) \
  o(DFT,        z,0,      0,0,   0,0,   N<5 ? 4*(N==3) : (N*N), N*(N-(N<5)), 0, true) \
  o(Bluestein,  M,0, r1,2, 0,0, N*2+r1, 0, 3, r1 && N<original*3 && !resolving.contains(r1))

#define o(name,T,R,c,d,E,F,m,a,t,s,condition) name,
enum class AnyMethod: unsigned { LIST_METHODS(o) };
#undef o
```

Taulukossa 2 esitetty merkintätapa rekursioille, esim. $2 \times (N/2)$, tarkoittaa, että algoritmi suorittaa kaksi kappaletta $\frac{N}{2}$ -kokoisia FFT-muunnoksia. Tällöin algoritmin suorittamien kertolaskujen lukumäärä yhteensä on algoritmin omien kertolaskujen lukumäärä, plus kaksi kertaa sen algoritmin kertolaskujen lukumäärä, jota käytetään $\frac{N}{2}$ -kokoiseen FFT-muunnokseen, joka puolestaan saattaa suorittaa pienempiä FFT-muunnoksia. Laskenta on siten luonteeltaan rekursiivinen, aivan kuten itse muunnoskin.

Listaus 27: fft_any.cc (osa 2)

```
14 #include "dft.hh"
15 #include "fft_tukey.hh"
16 #include "fft_rader.hh"
17 #include "fft_radix2.hh"
18 #include "fft_bluestein.hh"
19 #include "factor.hh"
20 #include <unordered_map>
21 #include <unordered_set>
22 #include <mutex>
23
24 static std::unordered_map<std::size_t, AnyConfiguration> data;
25 static std::unordered_map<std::size_t, std::size_t> cache;
26 static std::unordered_set<std::size_t> resolving;
27 static std::size_t original;
28 static std::mutex lock;
29
30 struct AnyConfiguration
31 {
32     cvector buf;
33     std::size_t r1,r2, mult,add,temp;
34     AnyMethod method;
35 };
```

Listauksessa 27 on esitelty määrittelyjä, ja listauksessa 28 on esitetty funktio, joka päättelee, mikä menetelmä soveltuisi parhaiten N-kokoisen syötteen käsittelyyn ja millä parametreillä. Tulokset tallennetaan välimuistiin (hajautustaulukko), jottei selvitystä samanpituiselle syötteelle tehtäisi useaan kertaan (rivit 40 ja 97). Lisäksi funktio pitää kirjaa siitä, minkäpituisia syötteitä se on tutkimassa, jottei rekursio (erityisesti Bluesteinin aiheuttama rekursio) ajautuisi ikuiseen silmukkaan (rivit 42, 43 ja 96).

Riveillä 46–63 funktio laskee erilaisia vaihtoehtoja Bluesteinin algoritmin parametrille m; nämä tallennetaan vektoriin `M`. Riveillä 64–70 määritetään kaikki mahdolliset yhdistelmät luvun N tekijöistä ja tallennetaan niiden tulot vektoriin `fac`. Riveillä 73–91 iteroidaan läpi kaikki edellä listatut menetelmät sekä kyseisen vektorin arvot. Jokaiselle menetelmälle lasketaan niiden edellyttämien kerto- ja yhteenlaskujen määrä sekä tarvittujen apuvektorien lukumäärä, ja näistä valitaan se menetelmä, jolla näiden yhteenlaskettu summa on pienin. Yksittäinen apuvektori katsotaan samanhintaiseksi kuin 64 reaalikertolaskua. Tämä luku on mielivaltaisesti valittu. Lopuksi valitun menetelmän tiedot tallennetaan välimuistiin rivillä 97 ja palautetaan funktiosta.

Luvussa 7 suoritetaan mittauksia, joista on mahdollista päätellä, että Cooleyn-Tukeyn yleisen algoritmin suorituskyky kärsi hieman siitä, että se joutuu joka kerta luomaan väliaikaistaulukon. Siksi tähän moduliin lisättiin sama tekniikka kuin Raderin ja Bluesteinin algoritmeissa, jossa tallennetaan väliaikaistaulukko vain kerran kutakin eri syötekokoa varten. Tämä väliaikaistaulun alustaminen tapahtuu rivillä 93.

Listaus 28: fft_any.cc (osa 3)

```
static AnyConfiguration& configure_any(std::size_t N)
{
    std::unique_lock<std::mutex> lk(lock);
    if(auto i = data.find(N); i != data.end()) return i->second;

    if(resolving.empty()) original = N;
    resolving.insert(N);

    std::vector<std::size_t> M, fac, z(1);
    for(std::size_t nb=2*N-1; ; ++nb)
    {
        std::size_t m = 0;
        for(std::size_t t = nb;;)
        {
            if(auto i = cache.find(t); i != cache.end()) { m |= i->second; break; }
            else if(t % 2 == 0) { m |= 1; t /= 2; }
            else if(t % 3 == 0) { m |= 2; t /= 3; }
            else if(t % 5 == 0) { m |= 4; t /= 5; }
            else if(t % 7 == 0) { m |= 8; t /= 7; }
            else { m = cache.emplace(nb, m+256*t).first->second; break; }
        }
        if(unsigned facmask = m&255, b = std::popcount(facmask); m < 2*256 && b >= 1 && b <= 2)
        {
            M.push_back(nb);
            if(facmask == 1) break; // Up to next power of two
        }
    }
    std::size_t smallest = smallest_factor(N), best = ~z[0], F[16], fs = get_factors(N,F,16);
    for(std::size_t f=0; f< std::size_t(1<<fs); ++f)
        if(std::size_t product = 1; true) //std::popcount(f) <= 8)
        {
            for(std::size_t b=0; b<fs; ++b) { if(f & (1<<b)) { product *= F[b]; } }
            if(product != 1 && product != N) fac.push_back(product);
        }
    lk.unlock();
    AnyConfiguration cfg;
    #define o(name, T,R,c,d,E,F,m,a,t,s, condition) for(std::size_t f=0; f<T.size(); ++f) \
    { \
        std::size_t mult=0, add=0, temp=0, r1 = T[f], r2 = R; \
        if(!(condition)) continue; \
        /* Add our traits, and theirs: */   mult+=(m);        add+=(a);       temp+=(t); \
        if(c) { auto& S = configure_any(c); mult+=(d)*S.mult; add+=(d)*S.add; temp+=(d)*S.temp; } \
        if(E) { auto& S = configure_any(E); mult+=(F)*S.mult; add+=(F)*S.add; temp+=(F)*S.temp; } \
        /* Complex multip. = 4 real multiplications and 2 additions; complex add = 2 real adds */ \
        std::size_t real_mul = mult * 4, real_add = mult * 2 + add * 2; \
        std::size_t slowness = real_mul + real_add + temp*64; \
        if(best == ~z[0] || slowness < best) /* Keep the best algorithm */ \
        { \
            best = slowness; \
            cfg.method = AnyMethod::name; cfg.mult     = mult; \
            cfg.r1     = r1;              cfg.add      = add; \
            cfg.r2     = r2;              cfg.temp     = temp; \
        } \
    }
    LIST_METHODS(o)
    #undef o
    cfg.buf.resize(cfg.r1 * cfg.r2);

    lk.lock();
    resolving.erase(N);
    return data.emplace(N, std::move(cfg)).first->second;
}
```

Listaus 29: fft_any.cc (osa 4)

```
100 void FFT_any_fast(
101     const complex* input, complex* output, exp_vector exps,
102     std::size_t N, std::size_t instride, std::size_t outstride)
103 {
104     auto& cfg = configure_any(N);
105     std::size_t r1 = cfg.r1, r2 = cfg.r2;
106     auto& buf = cfg.buf;
107     switch(cfg.method)
108     {
109         case AnyMethod::DFT:       DFT(input, output, exps, N, instride, outstride); break;
110         case AnyMethod::Rader:     FFT_rader(input, output, N, instride, outstride); break;
111         case AnyMethod::Bluestein: FFT_bluestein(input, output, N, instride, outstride, 0, r1); break;
112         case AnyMethod::Radix2:
113         {
114             // Because of cache efficiency reasons, use a smaller exps table in deeper recursions.
115             if(N >= 256 && exps.data->size() > N) { exps = ExpCircle(N); }
116             std::size_t half = N/2;
117             complex* even = output, *odd = output + half*outstride;
118             FFT_any_fast(input,          even, exps, half, instride*2, outstride);
119             FFT_any_fast(input+instride, odd,  exps, half, instride*2, outstride);
120             for(std::size_t a = 0; a < half; ++a) odd[a*outstride] *= exps(a, N);
121             for(std::size_t a = 0; a < half; ++a)
122             {
123                 complex part1 = even[a*outstride] + odd[a*outstride];
124                 complex part2 = even[a*outstride] - odd[a*outstride];
125                 even[a*outstride] = part1;
126                 odd[a*outstride]  = part2;
127             }
128             break;
129         }
130         case AnyMethod::Tukey_Rec1:
131             if(N >= 256 && exps.data->size() > N) { exps = ExpCircle(N); }
132 
133             for(std::size_t k0 = 0; k0 < r2; ++k0)
134                 FFT_any_fast(input + k0*instride, &buf[k0], exps, r1, r2*instride, r2);
135 
136             for(std::size_t n1 = 0; n1 < r2; ++n1)
137                 for(std::size_t n0 = 0; n0 < r1; ++n0)
138                 {
139                     complex value = 0;
140                     for(std::size_t k0 = 0; k0 < r2; ++k0)
141                         value += buf[k0 + n0*r2] * exps((n1*r1+n0)*k0, N);
142                     output[(n1*r1 + n0)*outstride] = value;
143                 }
144             break;
145         case AnyMethod::Tukey_Rec2:
146             if(N >= 256 && exps.data->size() > N) { exps = ExpCircle(N); }
147 
148             for(std::size_t k0 = 0; k0 < r2; ++k0)
149                 FFT_any_fast(input + k0*instride, &buf[k0], exps, r1, r2*instride, r2);
150 
151             for(std::size_t n0=0; n0<r1; ++n0)
152                 for(std::size_t k0 = 0; k0 < r2; ++k0)
153                     buf[k0 + n0*r2] *= exps(k0*n0, N);
154 
155             for(std::size_t n0 = 0; n0 < r1; ++n0)
156                 FFT_any_fast(&buf[n0*r2], &output[n0*outstride], exps, r2, 1,r1*outstride);
157             break;
158     }
159 }
```

Listauksessa 29 tapahtuu varsinainen FFT-laskenta. Funktio alkaa rivillä 107 `switch`-lausekkeella, jolla valitaan suoritettava menetelmä taulukon 2 listasta. Cooleyn-Tukeyn menetelmät on sisällytetty funktioon suoraan sisälle, jotta ne voivat kutsua `FFT_any_fast`-funktiota rekursiivisesti. Menetelmät sinällään vastaavat täysin listauksissa 12, 17 sekä 18 esitettyjä algoritmeja.

Toteutuksiin on lisätty myös ehtolause, joka hakee sopivamman kokoisen exp-taulukon, mikäli annettu exp-taulukko on liian iso (rivit 115, 131 ja 146). Tämä ei muuta ohjelman toimintaa, mutta muutos osoittautui testeissä nopeuttavan laskentaa marginaalisesti.

Lopuksi listauksessa 30 esitetään yksinkertaistettu rajapintafunktio, ja listauksessa 31 otsikkotiedosto, joka esittelee käytetyt funktiot.

Listaus 30: fft_any.cc (osa 5)

```
161 cvector FFT_any_fast(const cvector& input)
162 {
163     const auto N = input.size();
164     exp_vector exps = ExpCircle(N);
165     cvector result(N);
166     FFT_any_fast(&input[0], &result[0], exps, N, 1, 1);
167     return result;
168 }
```

Listaus 31: fft_any.hh

```
1 #include "defs.hh"
2 #include "exp.hh"
3
4 cvector FFT_any_fast(const cvector& input);
5 void FFT_any_fast(
6     const complex*    input,
7     complex*          output,
8     exp_vector        exps,
9     std::size_t N, std::size_t instride, std::size_t outstride);
```

6. Mittausmenetelmät

Oikeellisuus

Kaikki edellä esitetyt algoritmit testattiin sekä oikeellisuuden että nopeuden kannalta. Oikeellisuus testattiin vertaamalla algoritmin tulosta FFTW-kirjastoon. Testaus tapahtui listaukseen 13 pohjautuvalla mutta monimutkaisemmalla pääohjelmalla, jonka lähdekoodi löytyy koko projektin git-repositoriosta, johon linkki on liitteessä II.

Ohjelmisto- ja laitteistoympäristö

Oikeellisuustestissä hyödynnettiin OpenMP-kääntäjälaajennuksen tarjoamia säikeitä (”`#pragma omp parallel for`” yms.) testauksen nopeuttamiseksi [13]. Nopeus sen sijaan testattiin koodiversioilla, jossa mitään monisäikeisyyttä synnyttävää koodia ei ollut käytössä; koodi siis toimi ainoastaan yhtä suoritusydintä käyttäen seuraavassa ylikellottamattomassa laitteistossa[17]:

- CPU: AMD Ryzen Threadripper 3960X 24-Core Processor (3800 MHz)
- RAM: HyperX 4x16GB 3200MHz DDR4 CL16 (64 GiB)

Käyttöjärjestelmä oli Debian GNU/Linux Trixie. Kääntäjänä toimi GCC 13.2.0 seuraavin optioin: `-Wall -W -std=c++20 -Ofast -march=native -lfftw3f`.

Näytepituuksien valinta

Testeissä, joissa edustettuna on syötteiden pituudet lyhimmästä johonkin suureen lukuun kuten $1{,}6 \cdot 10^6$ saakka, lukualuetta harvennettiin siten, että kutakin näytekokoa n seuraava näytekoko määritettiin seuraavalla kaavalla:

$$n_{\text{next}} = n + \max\left\{1, \operatorname{rand}\left(0, \left\lceil \frac{n^{1{,}1677}}{7658} \right\rceil\right)\right\},$$

missä rand viittaa deterministiseen satunnaislukugeneraattoriin. Tällä saavutettiin se, että näytepituuksien tiheys logaritmisella akselilla pysyi jotakuinkin vakiona alusta loppuun ilman, että syötepituuksissa mikään tietty pituuden tyyppi (esim. alkuluvut tai parilliset luvut) olisi esiintynyt ylikorostuneena. Kaavassa esiintyvät vakiot 1,1677 ja 7658 ovat likiarvoja seuraavan yhtälöparin ratkaisulle:

$$\begin{cases} 3000^x &= 1{,}5\text{y} \\ 1048576^x &= 1400\text{y} \end{cases}.$$

[17] Vaikuttaa ehkä oudolta, miksi testiin on käytetty laitteistoa, jonka valtti on nimenomaan se, että siinä on kymmenittäin suoritusytimiä, mutta sitten testiä ajetaan vain yhdellä ytimellä. Osoittautui, että yksisäikeisesti sai vähiten herkästi ailahtelevia aikamittatuloksia. Mittauksia ei suoritettu sellaisella laitteistolla, joka olisi erikoistunut yksisäikeiseen suoritusnopeuteen, koska sellaista ei ollut kirjoittajalla käytettävissä.

Yhtälöparin tarkoitus on määrittää kaksi kontrollipistettä maksimisiirtymälle edellisestä pituudesta seuraavaan: Kun syötteen pituus on 3000, maksimisiirtymä edellisestä pituudesta seuraavaan on 1,5, ja kun syötteen pituus on 1048576, maksimisiirtymä edellisestä pituudesta seuraavaan on 1400. Nämä maksimisiirtymäarvot on valittu kokeilemalla, mikä on tutkielman kirjoittajaa miellyttävä kompromissi kuvaajien luettavuuden ja laskentaan kuluvan sähköenergian kannalta.

Vertailumenetelmä

Vaikka FFT on kategorialtaan $\mathcal{O}(N \log N)$ -algoritmi, nopeusvertailut eri algoritmien välillä tehtiin pääasiassa käyttäen potenssisovitusta logaritmisen sovituksen sijaan. Tämä osoittautui toimivimmaksi lähtökohdaksi, sillä se mahdollistaa parhaiten myös erilaisten funktioiden vertailun toisiinsa. Lisäksi se on elegantti tapa välttää hämmentävä tilanne, joka syntyy, jos logaritminen sovitus tuottaa yhtälön, joka kuvaa aineistoa heikosti. Näin osoittautui tutkielmaa laatiessa usein käyvän: logaritminen sovitus tuottaa helposti funktion, jonka arvot ovat negatiivisia merkittävällä osalla logaritmisesta x-akselista. Potenssisovituksella tätä ongelmaa ei esiintynyt. Potenssisovituksen ja logaritmisovituksen vertailukelpoisuudesta on tarkempaa pohdintaa kappaleessa 9.1.

6.1. Käytetyt sovitusmenetelmät

Lineaarinen sovitus yhden muuttujan selittävän aineiston x ja selitettävän aineiston y välille lasketaan pienimmän neliösumman menetelmällä (PNS). Jos merkitään aineiston kokoa N, voidaan laskea seuraavasti [12]:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$S_{xx} = \sum_{i=1}^{N} (x_i - \bar{x})^2 \qquad S_{xy} = \sum_{i=1}^{N} (x_i - \bar{x})\,(y_i - \bar{y})$$

$$\beta = \frac{S_{xy}}{S_{xx}} \qquad \alpha = \bar{y} - \beta\bar{x},$$

jolloin saadaan lineaarisen sovituksen kaavaksi $y = \alpha + \beta x$. Merkintä $\bar{x}$ viittaa tässä aineiston x aritmeettiseen keskiarvoon.

Logaritminen sovitus aineistojen x ja y välille saadaan muodostamalla lineaarinen sovitus aineistojen $\ln x$ ja y välille. Tuloksena saadaan yhtälö muotoa $y = \alpha + \beta \ln x$.

Potenssisovitus aineistojen x ja y välille saadaan muodostamalla lineaarinen sovitus aineistojen $\ln x$ ja $\ln y$ välille. Tuloksena saadaan yhtälö muotoa $\ln y = \alpha + \beta \ln x$, joka voidaan sieventää muotoon $y = e^{\alpha}x^{\beta}$. Kun merkitään $\gamma = e^{\alpha}$, saadaan potenssisovituksen kaavaksi $y = \gamma x^{\beta}$.

Sovitukset muotoa $t(n) = \alpha n + \beta n^2$, $t(n) = n(\alpha + \beta \ln n)$ tai $t(n) = \gamma n^{\beta+1}$ saadaan laskettua käyttämällä edellä esitettyjen sovitusten laskennassa arvoja $x = n$, $y = \frac{t(n)}{n}$ ja kertomalla tuloksena saadun yhtälön molemmat puolet muuttujalla n.

7. Mittaustulokset

7.1. Tarkkuus

Taulukossa 3 on esitetty vertailut, joissa kutakin algoritmia verrattiin referenssitoteutukseen (FFTW) eri kokoisilla syötteillä. FFT laskettiin kyseisellä algoritmilla sekä FFTW:llä, ja tuloksista laskettiin neliöityjen absoluuttisten erojen summa jaettuna syötteen pituudella. Yli 10^{-3}:n suuruinen ero katsottiin virheeksi algoritmin toiminnassa. Virhearvoja pohditaan tarkemmin luvussa 9.3. Naiivin DFT:n testaaminen lopetettiin yli 196000 alkion pituisilla syötteillä, sillä sen testaaminen oli kohtuuttoman hidasta. Samassa vaiheessa myös Tukey-algoritmeja muutettiin niin, että ne suorittavat alkulukukokoisen syötteen prosessoinnin `FFT_any_fast`-algoritmilla naiivin DFT:n sijaan.

Kategoria	DFT	Tukey ($2^\square$)	Tukey ($2^\square$ fast)	Tukey (gen)	Tukey (1rec)	Tukey (2rec)	Rader	Bluestein ($2^\square 3^\square 5^\square 7^\square$)	Bluestein ($2^\square$)	Bluestein ($2^\square 3^\square$)	any
1–10 ($n = 10$)											
min	0	0	$9 \cdot 10^{-15}$	0	0	0	0	0	0	0	0
max	$8 \cdot 10^{-14}$	$2 \cdot 10^{-14}$	$3 \cdot 10^{-14}$	$7 \cdot 10^{-14}$	$8 \cdot 10^{-14}$	$4 \cdot 10^{-14}$	$6 \cdot 10^{-14}$	$5 \cdot 10^{-13}$	$2 \cdot 10^{-13}$	$2 \cdot 10^{-13}$	$9 \cdot 10^{-14}$
avg	$3 \cdot 10^{-14}$	$1 \cdot 10^{-14}$	$2 \cdot 10^{-14}$	$4 \cdot 10^{-14}$	$4 \cdot 10^{-14}$	$2 \cdot 10^{-14}$	$3 \cdot 10^{-14}$	$2 \cdot 10^{-13}$	$7 \cdot 10^{-14}$	$7 \cdot 10^{-14}$	$3 \cdot 10^{-14}$
11–100 ($n = 83$)											
min	$9 \cdot 10^{-14}$	$7 \cdot 10^{-14}$	$7 \cdot 10^{-14}$	$6 \cdot 10^{-14}$	$7 \cdot 10^{-14}$	$5 \cdot 10^{-14}$	$2 \cdot 10^{-13}$	$2 \cdot 10^{-13}$	$1 \cdot 10^{-13}$	$2 \cdot 10^{-13}$	$7 \cdot 10^{-14}$
max	$6 \cdot 10^{-12}$	$8 \cdot 10^{-13}$	$9 \cdot 10^{-13}$	$5 \cdot 10^{-12}$	$5 \cdot 10^{-12}$	$5 \cdot 10^{-12}$	$2 \cdot 10^{-11}$	$1 \cdot 10^{-11}$	$6 \cdot 10^{-12}$	$6 \cdot 10^{-12}$	$1 \cdot 10^{-11}$
avg	$2 \cdot 10^{-12}$	$3 \cdot 10^{-13}$	$4 \cdot 10^{-13}$	$1 \cdot 10^{-12}$	$1 \cdot 10^{-12}$	$1 \cdot 10^{-12}$	$5 \cdot 10^{-12}$	$4 \cdot 10^{-12}$	$2 \cdot 10^{-12}$	$3 \cdot 10^{-12}$	$2 \cdot 10^{-12}$
101–1000 ($n = 297$)											
min	$4 \cdot 10^{-12}$	$2 \cdot 10^{-12}$	$2 \cdot 10^{-12}$	$2 \cdot 10^{-12}$	$2 \cdot 10^{-12}$	$2 \cdot 10^{-12}$	$6 \cdot 10^{-12}$	$5 \cdot 10^{-12}$	$4 \cdot 10^{-12}$	$4 \cdot 10^{-12}$	$2 \cdot 10^{-12}$
max	$4 \cdot 10^{-10}$	$1 \cdot 10^{-11}$	$1 \cdot 10^{-11}$	$2 \cdot 10^{-10}$	$2 \cdot 10^{-10}$	$2 \cdot 10^{-10}$	$3 \cdot 10^{-10}$	$2 \cdot 10^{-10}$	$1 \cdot 10^{-10}$	$2 \cdot 10^{-10}$	$2 \cdot 10^{-10}$
avg	$1 \cdot 10^{-10}$	$5 \cdot 10^{-12}$	$6 \cdot 10^{-12}$	$3 \cdot 10^{-11}$	$3 \cdot 10^{-11}$	$3 \cdot 10^{-11}$	$7 \cdot 10^{-11}$	$5 \cdot 10^{-11}$	$3 \cdot 10^{-11}$	$4 \cdot 10^{-11}$	$3 \cdot 10^{-11}$
1001–10 000 ($n = 568$)											
min	$3 \cdot 10^{-10}$	$2 \cdot 10^{-11}$	$3 \cdot 10^{-11}$	$2 \cdot 10^{-11}$	$2 \cdot 10^{-11}$	$3 \cdot 10^{-11}$	$1 \cdot 10^{-10}$	$6 \cdot 10^{-11}$	$5 \cdot 10^{-11}$	$6 \cdot 10^{-11}$	$3 \cdot 10^{-11}$
max	$3 \cdot 10^{-8}$	$3 \cdot 10^{-10}$	$3 \cdot 10^{-10}$	$1 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	$3 \cdot 10^{-9}$	$4 \cdot 10^{-9}$	$1 \cdot 10^{-9}$	$2 \cdot 10^{-9}$	$2 \cdot 10^{-9}$
avg	$8 \cdot 10^{-9}$	$1 \cdot 10^{-10}$	$1 \cdot 10^{-10}$	$7 \cdot 10^{-10}$	$7 \cdot 10^{-10}$	$7 \cdot 10^{-10}$	$1 \cdot 10^{-9}$	$7 \cdot 10^{-10}$	$4 \cdot 10^{-10}$	$6 \cdot 10^{-10}$	$4 \cdot 10^{-10}$
10 001–100 000 ($n = 883$)											
min	$3 \cdot 10^{-8}$	$6 \cdot 10^{-10}$	$6 \cdot 10^{-10}$	$4 \cdot 10^{-10}$	$4 \cdot 10^{-10}$	$4 \cdot 10^{-10}$	$2 \cdot 10^{-9}$	$1 \cdot 10^{-9}$	$7 \cdot 10^{-10}$	$1 \cdot 10^{-9}$	$4 \cdot 10^{-10}$
max	$3 \cdot 10^{-6}$	$3 \cdot 10^{-9}$	$3 \cdot 10^{-9}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$8 \cdot 10^{-8}$	$4 \cdot 10^{-8}$	$2 \cdot 10^{-8}$	$2 \cdot 10^{-8}$	$3 \cdot 10^{-8}$
avg	$7 \cdot 10^{-7}$	$2 \cdot 10^{-9}$	$2 \cdot 10^{-9}$	$3 \cdot 10^{-8}$	$3 \cdot 10^{-8}$	$3 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	$8 \cdot 10^{-9}$	$5 \cdot 10^{-9}$	$7 \cdot 10^{-9}$	$5 \cdot 10^{-9}$
100 001–1 000 000 ($n = 1512$)											
min	$3 \cdot 10^{-6}$	$6 \cdot 10^{-9}$	$6 \cdot 10^{-9}$	$5 \cdot 10^{-9}$	$5 \cdot 10^{-9}$	$5 \cdot 10^{-9}$	$4 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	$9 \cdot 10^{-9}$	$1 \cdot 10^{-8}$	$6 \cdot 10^{-9}$
max	$1 \cdot 10^{-5}$	$3 \cdot 10^{-8}$	$3 \cdot 10^{-8}$	$1 \cdot 10^{-5}$	$9 \cdot 10^{-6}$	$7 \cdot 10^{-6}$	$5 \cdot 10^{-7}$	$5 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	$3 \cdot 10^{-7}$	$5 \cdot 10^{-7}$
avg	$7 \cdot 10^{-6}$	$2 \cdot 10^{-8}$	$2 \cdot 10^{-8}$	$2 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$6 \cdot 10^{-8}$	$8 \cdot 10^{-8}$	$6 \cdot 10^{-8}$
1 000 001–10 000 000 ($n = 2191$)											
min	—	$6 \cdot 10^{-8}$	$6 \cdot 10^{-8}$	$6 \cdot 10^{-8}$	$6 \cdot 10^{-8}$	$6 \cdot 10^{-8}$	$4 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$9 \cdot 10^{-8}$	$1 \cdot 10^{-7}$	$6 \cdot 10^{-8}$
max	—	$6 \cdot 10^{-7}$	$6 \cdot 10^{-7}$	$3 \cdot 10^{-5}$	$3 \cdot 10^{-5}$	$3 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$9 \cdot 10^{-6}$	$3 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$4 \cdot 10^{-6}$
avg	—	$2 \cdot 10^{-7}$	$3 \cdot 10^{-7}$	$8 \cdot 10^{-7}$	$8 \cdot 10^{-7}$	$8 \cdot 10^{-7}$	$3 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$7 \cdot 10^{-7}$	$1 \cdot 10^{-6}$	$7 \cdot 10^{-7}$
10 000 001–100 000 000 ($n = 1237$)											
min	—	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$7 \cdot 10^{-7}$	$7 \cdot 10^{-7}$	$8 \cdot 10^{-7}$	$5 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$7 \cdot 10^{-7}$
max	—	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$9 \cdot 10^{-4}$	$9 \cdot 10^{-4}$	$9 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$4 \cdot 10^{-5}$	$9 \cdot 10^{-6}$	$1 \cdot 10^{-5}$	$9 \cdot 10^{-4}$
avg	—	$3 \cdot 10^{-6}$	$3 \cdot 10^{-6}$	$7 \cdot 10^{-6}$	$7 \cdot 10^{-6}$	$7 \cdot 10^{-6}$	$1 \cdot 10^{-5}$	$5 \cdot 10^{-6}$	$3 \cdot 10^{-6}$	$4 \cdot 10^{-6}$	$6 \cdot 10^{-6}$
Alkuluvut ($n = 375$)											
min	0	—	—	—	—	—	0	0	0	0	0
max	$1 \cdot 10^{-5}$	—	—	—	—	—	$1 \cdot 10^{-4}$	$2 \cdot 10^{-5}$	$9 \cdot 10^{-6}$	$1 \cdot 10^{-5}$	$1 \cdot 10^{-4}$
avg	$7 \cdot 10^{-7}$	—	—	—	—	—	$2 \cdot 10^{-6}$	$9 \cdot 10^{-7}$	$6 \cdot 10^{-7}$	$8 \cdot 10^{-7}$	$1 \cdot 10^{-6}$
Yhdistetyt luvut: Vain pieniä tekijöitä ($n = 2580$)											
min	$9 \cdot 10^{-15}$	0	$9 \cdot 10^{-15}$	0	0	0	—	$5 \cdot 10^{-14}$	$2 \cdot 10^{-14}$	$2 \cdot 10^{-14}$	$9 \cdot 10^{-15}$
max	$1 \cdot 10^{-5}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$6 \cdot 10^{-6}$	$6 \cdot 10^{-6}$	$9 \cdot 10^{-6}$	—	$4 \cdot 10^{-5}$	$4 \cdot 10^{-6}$	$7 \cdot 10^{-6}$	$4 \cdot 10^{-6}$
avg	$1 \cdot 10^{-6}$	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	$5 \cdot 10^{-7}$	—	$1 \cdot 10^{-6}$	$6 \cdot 10^{-7}$	$9 \cdot 10^{-7}$	$5 \cdot 10^{-7}$
Yhdistetyt luvut: Ei pieniä tekijöitä ($n = 596$)											
min	$1 \cdot 10^{-11}$	—	—	$4 \cdot 10^{-12}$	$4 \cdot 10^{-12}$	$4 \cdot 10^{-12}$	—	$1 \cdot 10^{-11}$	$8 \cdot 10^{-12}$	$9 \cdot 10^{-12}$	$8 \cdot 10^{-12}$
max	$1 \cdot 10^{-5}$	—	—	$9 \cdot 10^{-4}$	$9 \cdot 10^{-4}$	$9 \cdot 10^{-4}$	—	$2 \cdot 10^{-5}$	$8 \cdot 10^{-6}$	$1 \cdot 10^{-5}$	$9 \cdot 10^{-4}$
avg	$1 \cdot 10^{-6}$	—	—	$5 \cdot 10^{-6}$	$3 \cdot 10^{-6}$	$3 \cdot 10^{-6}$	—	$2 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$3 \cdot 10^{-6}$
Kahden potenssit (> 2) ($n = 27$)											
min	0	0	$9 \cdot 10^{-15}$	0	0	0	—	0	0	0	0
max	$6 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	—	$1 \cdot 10^{-5}$	$2 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$1 \cdot 10^{-6}$
avg	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	$3 \cdot 10^{-7}$	$3 \cdot 10^{-7}$	$3 \cdot 10^{-7}$	—	$5 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	$9 \cdot 10^{-8}$
Kaikki ($n = 6781$)											
min	0	0	$9 \cdot 10^{-15}$	0	0	0	0	0	0	0	0
max	$1 \cdot 10^{-5}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$9 \cdot 10^{-4}$	$9 \cdot 10^{-4}$	$9 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$4 \cdot 10^{-5}$	$9 \cdot 10^{-6}$	$1 \cdot 10^{-5}$	$9 \cdot 10^{-4}$
avg	$1 \cdot 10^{-6}$	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	$2 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$8 \cdot 10^{-7}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$

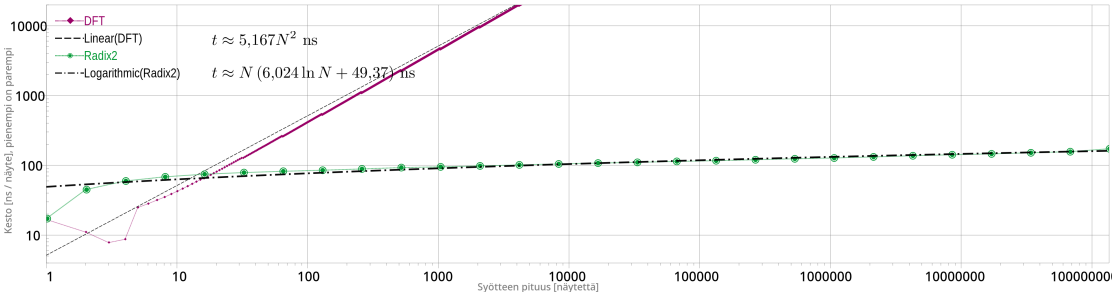
Taulukko 3: Neliölliset virheet (pienin, suurin, keskimääräinen) kunkin algoritmin laskennassa (verrattuna FFTW-kirjastoon) eri kokoisilla syötteillä.

7.2. Nopeus

DFT vs. Cooley Tukey radix-2 (perustoteutus)

Kuvaajassa 4 on esitetty nopeusvertailu naiivin DFT:n sekä Cooleyn-Tukeyn algoritmin Radix 2 -version perustoteutuksesta. Kuvaaja esittää algoritmien kestot näytettä kohti syötteen pituuden funktiona. Nopeusvertailusta voidaan tehdä seuraavat havainnot tutkittavista algoritmeista:

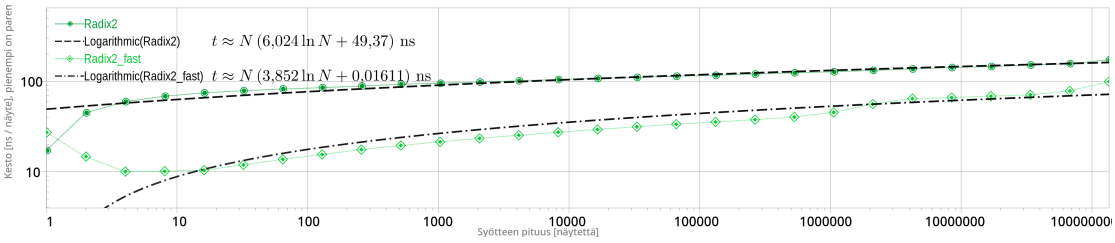
- DFT-laskenta seuraa odotetusti $\mathcal{O}(N^2)$ -suoritusaikavaatimusta: muunnos kesti testilaitteistolla keskimäärin 5,2 N^2 ns näytteen pituuden N funktiona.
- Omatekoinen Radix 2 -FFT noudatti odotetusti $\mathcal{O}(N \log N)$ -suoritusaikavaatimusta: muunnos kesti keskimäärin $6N \ln N + 49N$ ns.



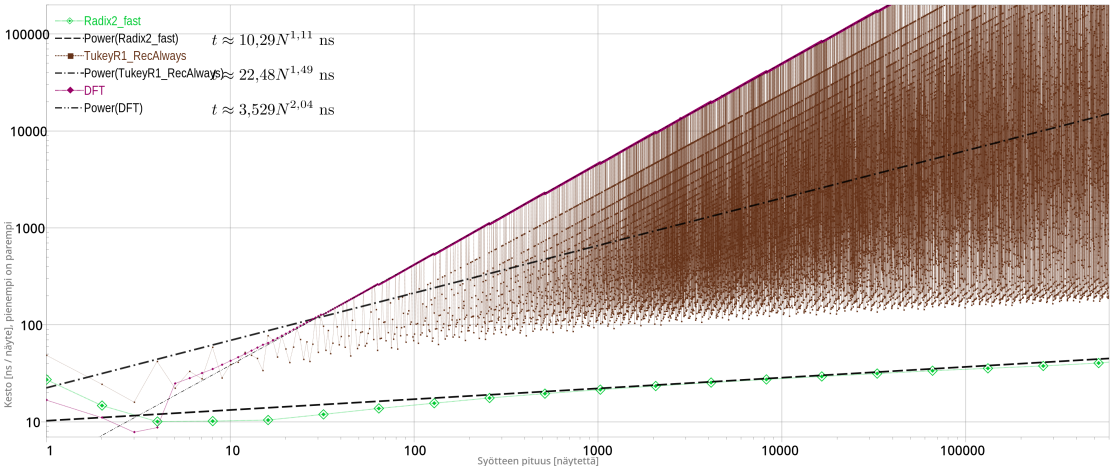
Kuva 4: Nopeusvertailu, jossa naiivi DFT sekä Radix 2 -versio Cooleyn-Tukeyn algoritmista. Akselit ovat logaritmisia. Mustat katkoviivat esittävät mittauspisteisiin laskettua sovitusta (luku 6.1); lineaarinen DFT:n tapauksessa ja logaritminen Radix 2:n tapauksessa.

Cooley Tukey radix-2, perustoteutus vs. optimoitu toteutus

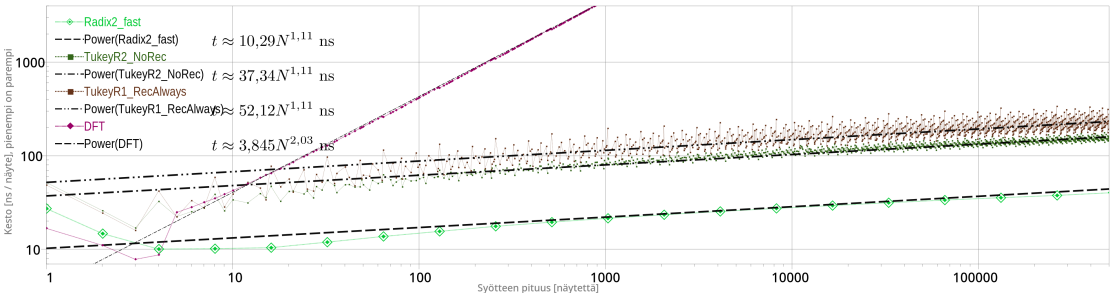
Kuvaajasta 5 nähdään, että listauksessa 11 optimoidun radix-2 -algoritmin suoritusaika oli selvästi parempi kuin ensimmäisen version. Noin kahden miljoonan alkion kohdalla suoritus kuitenkin hidastuu merkittävästi, mikä todennäköisesti johtuu suoritusytimen välimuistin täyttymisestä.



Kuva 5: Nopeusvertailu Cooleyn-Tukeyn algoritmin radix-2 -erikoistapauksen perustoteutuksesta ja optimoidusta toteutuksesta. Mustat katkoviivat esittävät logaritmisia sovituksia.



Kuva 6: Nopeusvertailu, jossa ovat Cooley-Tukeyn yleinen tapaus (kaksi rekursiota), Radix-2 ja DFT. Mustat katkoviivat esittävät mittauspisteisiin laskettuja potenssisovituksia.



Kuva 7: Sama vertailu kuin kuvaajassa 6, mutta nyt mukaan on poimittu ainoastaan ne näytekoot, joissa koko N koostuu pelkästään pienistä alkulukutekijöistä. Kuvaajassa on esitetty erikseen yhden rekursion (”TukeyR2_NoRec”) sekä kahden rekursion (”TukeyR1_RecAlways”) versiot yleisen tapauksen algoritmista.

Radix-2 -versio vs. Cooley-Tukeyn yleinen tapaus

Kuvaajassa 6 vertaillaan Cooleyn-Tukeyn algoritmin Radix-2-versiota sen yleiseen versioon. Toisin kuin aiemmin esitetyissä kuvaajissa, nyt suoritusaika ei muodosta selkeää käyrää, vaan pikemminkin logaritmisella asteikolla se muodostaa kolmion, jonka sisälle yksittäiset aikamäärät sijoittuvat. Kolmion alareuna esittää algoritmin parasta tapausta, jossa näytteen koko koostuu ainoastaan pienistä alkulukutekijöistä, ja yläreuna heikointa tapausta, jossa näytteen koko on alkuluku. Vertailun vuoksi kuvaajassa on esitetty myös DFT, ja onkin helposti nähtävissä, että yleisen tapauksen heikoin tapaus täsmää tarkalleen DFT-algoritmiin[18]. Kuvaajasta on erotettavissa myös lukuisia vino-

[18] Pienillä syöteko’oilla DFT on itse asiassa hieman nopeampi. Tämä johtuu tekijöihin jakamiseen kuluvasta ajasta, jonka suhteellinen merkitys vaikuttaa kuvaajan perusteella muuttuvan mitättömäksi vasta, kun syötteen koko on vähintään n. 20 näytettä.

viivan tapaisesti järjestyneitä pistejonoja. Nämä säännönmukaisesti etenevät pistejonot vastaavat tapauksia, joissa syötteen pituus on jaollinen jollain tietyllä alkuluvulla. Tällöin suoritus on nopeampaa. Kukin alkulukutekijä nopeuttaa suoritusta tietyn laskennallisen kertoimen mukaan (kts. lause 4.7), ja niinpä jokaisesta tekijästä muodostuukin oma pistejononsa.

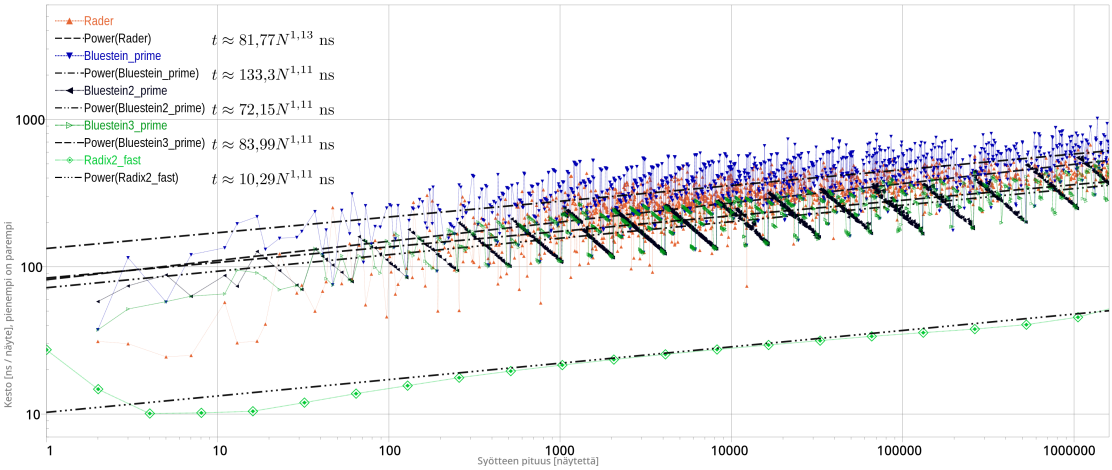
Kuvaajassa 7 on esitetty sama vertailu, mutta tällä kertaa niin, että mukaan on valittu vain ne tapaukset, joissa syötteen koko on pienten alkulukutekijöiden (2, 3, 5 ja 7) tulo. Tällöin yleisen tapauksen suorituskyky osoittautuukin varsin vertailukelpoiseksi optimoidun Radix-2 -version kanssa, vaikkakaan aivan samaan se ei pääse. Ero todennäköisesti johtuu yleisen tapauksen tarvitsemasta aputaulusta välituloksia varten. Aputaululle tarvittavan muistin varaaminen ja vapauttaminen hidastaa suoritusta. Tämä olisi mahdollista välttää varaamalla muisti etukäteen samoin kuin esim. FFTW tekee plan-operaationsa osana.

Kuvaajassa on mukana itse asiassa Tukeyn yleisestä versiosta kaksi eri toteutusta: yhden rekursion ja kahden rekursion versiot. Nämä kaksi eri versiota esiteltiin listauksessa 17. Kuten lauseiden 4.5 ja 4.6 todistusten yhteydessä pääteltiin, silloin, kun alkulukupituisen syötteen muunnos tehdään naiivilla DFT:llä, yhden rekursion versio on selvästi tehokkaampi kuin kahden rekursion versio. Tämä ilmenee myös kuvaajasta: Yhden rekursion malli pärjäsi n. 40 %:a paremmin kuin kahden rekursion malli. Asiasta on lisää pohdintaa luvussa 9.4.

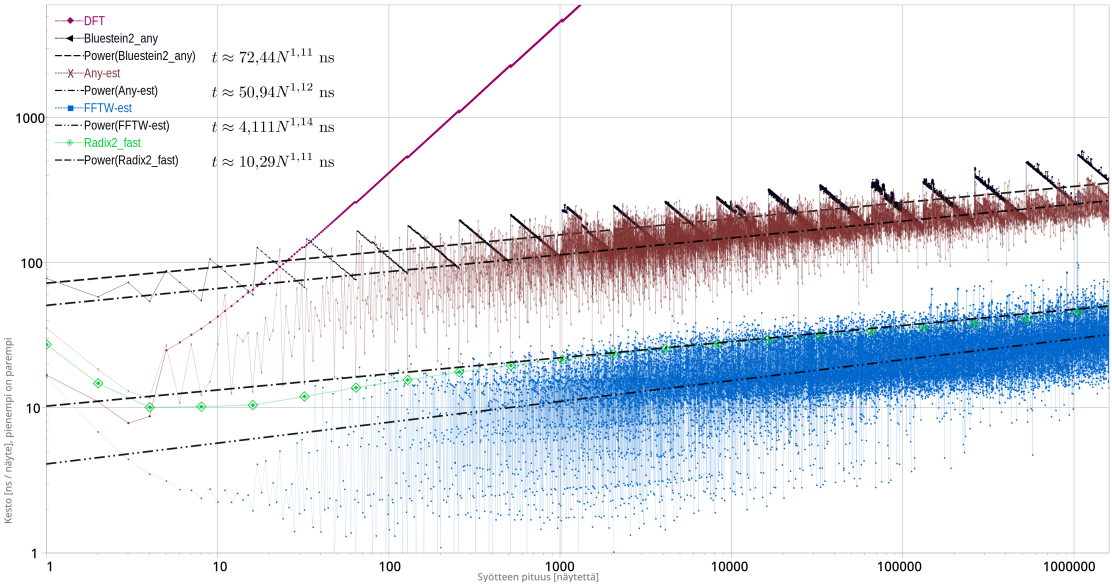
Rader vs. Bluestein

Nopeustestissä tähän asti käsitellyistä algoritmeista vain neliöllinen perus-DFT soveltui sellaisten näytesarjojen muuntamiseen, joiden pituus on alkuluku. Luvuissa 4.3 ja 4.4 esitetyt Raderin ja Bluesteinin algoritmit kuitenkin kykenevät ainakin teoriassa saavuttamaan neliöllistä paremman suorituskyvyn silloinkin, kun näytesarjan pituus on alkuluku. Kuvaajassa 8 on esitetty näiden kahden suoritusnopeus sellaisissa tapauksissa, joissa näytesarjan koko on aina alkuluku. Koska molempien algoritmien tarkoitus on muuntaa syöte sillä tavalla, että voidaan suorittaa FFT jollekin eri kokoiselle syötteelle, molempien algoritmien suorituskyky riippuu dramaattisesti siitä, mitä FFT-algoritmia kyseiset algoritmit käyttävät apunaan muunnetun syötteen käsittelyssä. Tässä testissä on käytetty `FFT_any_fast`-menetelmää, joka saattaa suorittaa lisää Rader- tai Bluestein-kutsuja, riippumatta siitä mitä menetelmää tässä testataan. Yleisesti ottaen näyttää kuitenkin siltä, että lyhyillä syötteillä (alle sata näytettä) Rader on pääsääntöisesti tehokkaampi kuin Bluestein, ja pitemmillä syötteillä tilanne on päinvastoin.

Kuvaajassa on mukana kolme eri versiota Bluesteinin algoritmista. Ne eroavat toisistaan konvoluutiopituuden m valinnan suhteen: ”Bluestein” valitsee pienimmän arvon $m \geqslant 2N-1$, joka koostuu tekijöistä 2, 3, 5 ja 7. ”Bluestein2” valitsee ainoastaan kahden potensseja. ”Bluestein3” valitsee pienimmän vain tekijöistä 2 ja 3 koostuvan arvon. Pääsääntöisesti pelkästään kahden potensseja valitseva algoritmi oli nopein näistä kolmesta, mutta joissain tilanteissa ”Bluestein3” oli kuitenkin nopein.



Kuva 8: Raderin ja Bluesteinin algoritmit vertailussa. Kyseiset algoritmit on laskettu tässä ainoastaan alkulukupituuksille. Vertailun vuoksi on esitetty Cooley-Tukey radix-2, jossa on käytetty ainoastaan kahden potensseja.



Kuva 9: Yhdistetty funktio verrattuna FFTW-kirjastoon. Vertailukohdiksi on esitetty myös naiivi DFT, Cooleyn-Tukeyn Radix-2 -versio pelkillä kahden potensseilla, sekä Bluesteinin kahdenpotenssiversio.

Yhdistetty funktio vs. FFTW

Kuvaajassa 9 on esitetty yhdistetyn funktion `FFT_any_fast` vertailu FFTW-kirjastoon.

Nähdään, että tutkielmassa esitetty yhdistetty funktio tarjoaa suorituskyvyn, joka on parhaimmassa tapauksessaan samaa luokkaa FFTW-kirjaston kanssa ja heikoimmassa tapauksessaan samaa luokkaa aiemmin käsitellyn Bluestein-toteutuksen kanssa. Tällä syötekokovälillä se saavuttaa keskimäärin suoritusaikavaatimuksen $\mathcal{O}(N^{1,12})$ vakiokertoimella 51. Missään tilanteessa sen suoritusaikavaativuus ei käyttäydy neliöllisesti niin kuin naiivilla DFT:llä.

Vertailun vuoksi FFTW-kirjastolla suoritusaikavaativuus on testin perusteella $\mathcal{O}(N^{1,14})$ vakiokertoimella 4,1. FFTW:n kuvaajan kulmakerroin on jyrkempi, sillä pienillä syöteko'oilla se hyötyy suhteessa useammin siitä, että se sisältää äärimmäisen tehokkaita generoituja ja erikoistuneita muunnosfunktioita pituuksille 1–32. Suuremmilla syötteillä tämän edun suhteellinen hyöty vähenee, minkä takia sen keskimääräinen suoritusaika hieman nousee[19].

Joka tapauksessa kuvaajasta on nähtävissä, että FFTW:n suorituskyky on monta kertaluokkaa parempi kuin tässä tutkielmassa laaditulla koodilla, joskin ero on vakiokerroinluonteinen; suoritusaikavaativuuskategoria on molemmilla saman tyyppinen. Tätä parempaa suorituskykyä tuskin saavutetaan ilman FFTW:n hyödyntämää matalan tason ohjelmointia [11].

[19] FFTW-kirjaston suunnitelmia (*plan*) laadittiin ainoastaan `FFTW_ESTIMATE`-moodissa, sillä se vastaa tapaa, jota myös tutkielmassa esitetty funktio `FFT_any_fast` käyttää. Myös `FFTW_EXHAUSTIVE`-moodia testattiin; sitä käyttämällä kirjasto saavutti keskimäärin puolet paremman suoritusajan. Näitä tuloksia ei esitetty tässä tutkielmassa, koska tutkielman tarkoitus ei ole analysoida FFTW-kirjastoa tarkemmin. FFTW:n eri suunnittelumenetelmien vaikutusta sen tarkkuuteen ei myöskään tutkittu.

7.3. Yhteenveto

Taulukossa 4 on esitetty tiivistelmä tämän luvun mittaustuloksista.

Nimi	Keskimääräinen kesto (potenssina)	Luokitus	Rajoitukset	Muuta käytön kannalta huomioitavaa
Naiivi DFT	$3{,}5N^{2{,}0}$ ns	Hitain	—	Soveltuu käytettäväksi vain, jos syöte on hyvin lyhyt.
Rader	$82N^{1{,}1}$ ns		Vain alkulukukoot	Lyhyillä syötteillä yleensä nopeampi kuin Bluestein. Tehokas silloin, jos $N-1$ koostuu pienistä alkulukutekijöistä.
Bluestein ($m = 2^{\square}$)	$72N^{1{,}1}$ ns		—	Hyödyllinen lähinnä silloin, jos syötteen pituus on alkuluku tai suurten alkulukujen tulo. Pitkillä syötteillä yleensä nopeampi kuin Rader.
Yhdistetty	$51N^{1{,}1}$ ns	Nopea	—	Valitsee tapauksesta riippuen parhaan tunnetun tekniikan; ellet tiedä, että jokin muu soveltuu paremmin, käytä tätä.
Cooley-Tukey (yleinen)	$37N^{1{,}1}$ ns		Pituuden on oltava yhdistetty luku	Tekijöiden on hyvä olla pieniä.
Cooley-Tukey (Radix-2)	$10N^{1{,}1}$ ns	Nopea	Kahden potenssikoot	
FFTW	$4{,}1N^{1{,}1}$ ns	Nopea	—	Ylivoimaisesti nopein kaikissa tapauksissa. Vieläkin nopeampi, mikäli annetaan laatia suunnitelma huolellisesti ja ajan kanssa.

Taulukko 4: Toteutetut FFT-menetelmät mittaustuloksineen.

8. Esimerkkisovellus

FFT:llä on lukemattomia sovelluksia, jotka vaihtelevat signaalinkäsittelystä matematiikkaan ja tekoälyyn. Eräs sovellus liittyy puheen tunnistamiseen.

Puheen ja äänen tutkimuksessa käsitteellä *formantti* tarkoitetaan erityisesti äänen taajuusspektrin amplitudihuippuja, eli taajuusalueita, joille akustinen energia on kasaantunut [8]. Ihmisen kyky erottaa puheäänen eri vokaalit toisistaan perustuu kykyyn erotella äänen perustaajuuden lisäksi sen formantit toisistaan [21]. Taulukossa 5 on esitetty lista tyypillisistä miesäänen formanteista eri vokaaleille.

Vokaali (IPA)	Formantti F_1 (Hz)	Formantti F_2 (Hz)	Erotus $F_2 - F_1$ (Hz)
i *	240	2400	2160
y *	235	2100	1865
e *	390	2300	1910
ø *	370	1900	1530
ɛ	610	1900	1290
œ	585	1710	1125
a	850	1610	760
æ	820	1530	710
ɑ *	750	940	190
ɒ	700	760	60
ʌ	600	1170	570
ɔ	500	700	200
ɤ	460	1310	850
o *	360	640	280
ɯ	300	1390	1090
u *	250	595	345

Taulukko 5: Tyypilliset vokaaliformantit [8]. Asteriskilla on merkitty suomen kielessä käytettävät vokaalit, lukuunottamatta vokaalia /æ/, jolle tietoa ei löytynyt.

Formanttianalyysiä varten ladattiin YouTube-videopalvelusta suomenkielistä puhetta sisältävä Helsingin yliopiston video `https://youtu.be/SQtMTGpR4VE`, ”Lauri Oksanen: Miten kuvantamiseen liittyviä ongelmia voi ratkaista kaareutuvissa aikaavaruuksissa?”. Videon ääniraita ladattiin työasemalle `yt-dlp` -työkalulla[20]. Siitä eristettiin 1,3 sekunnin mittainen näyte `ffmpeg` -työkalulla[21], ja tulos muutettiin yksikanavaiseksi 44100 Hz raakatiedostoksi `SoX` -työkalulla[22]. Komentosarja oli seuraava:

```
yt-dlp https://youtu.be/SQtMTGpR4VE -f140
ffmpeg -i *SQtMT*.m4a -ss 1:32.3 -t 1.3 test.wav
sox test.wav -tf32 -c1 -r44100 test.raw.
```

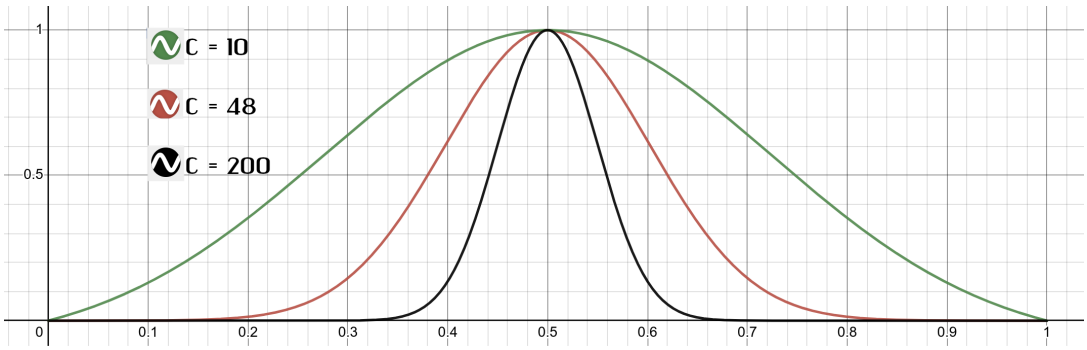
[20] `https://github.com/yt-dlp/yt-dlp`
[21] `https://ffmpeg.org/`
[22] `https://sourceforge.net/projects/sox/`

Kuva 10: Puhetta sisältävää näytesarjaa esittävä kuvaaja.

Kuva 11: Taajuussarjan amplitudispektri, tarkemmin sanottuna sen vasen puolikas. Näytesarjalle on ensin suoritettu diskreetti Fourier-muunnos. Tuloksena syntyneistä kompleksisista arvoista on laskettu amplitudiarvot.

Tuloksena oli `test.raw`-niminen tiedosto, joka sisälsi sanat ”vaikka maapallon rakenne”. Tiedosto sisältää näytesarjan, joka kuvastaa 44100 kertaa sekunnissa näytteistettyjä suhteellisia ilmanpainevaihteluja, jotka ihminen havaitsee äänenä [20]. Tätä esitysmuotoa kutsutaan pulssikoodimodulaatioksi (PCM). Näytesarja on esitetty kuvaajassa 10. Tämä joukko voitaisiin toki muuntaa taajuussarjaksi FFT:llä, ja kuvaajasta 11 nähdään, millainen olisi tulos. Tästä ei kuitenkaan olisi puheanalyysin kannalta erityisesti hyötyä. Kuvaaja ei päällisin puolin kerro oikeastaan muuta kuin että se koostuu pääasiassa matalista taajuuksista, eikä esimerkiksi sisällä korkeataajuisia vihellyksiä.

Jotta äänisignaalin taajuuskomponenttien muutoksia ajan funktiona voitaisiin tutkia



Kuva 12: Gaussin kellokäyrän 0–1 -välille skaalattu muoto $\frac{e^{-C\cdot(x-0{,}5)^2}-e^{-C/4}}{1-e^{-C/4}}$ muutamilla vakion C eri arvoilla.

mielekkäästi, täytyy näytesarjalle suorittaa niin kutsuttu ikkunointi [20]. Se tarkoittaa, että näytesarjalle suoritetaan monta FFT-muunnosta peräkkäin siten, että kuhunkin muunnokseen valitaan näytesarjasta osa, ja valitun osan sijaintia liikutetaan eteenpäin kuin suurennuslasia tekstin päällä. Ikkunan leveys valitaan sen perusteella, kuinka nopeisiin muutoksiin äänen koostumuksessa halutaan reagoida. Olkoon R näytteistystaajuus (esim. 44100 Hz), S FFT:n koko (esim. 44100), W ikkunan leveys (sekuntia) ja A ikkunan alkupositio näytteen alkusta (sekuntia). Tällöin ikkunoitu amplitudisarja X muodostetaan näytesarjasta x seuraavilla kaavoilla, kun $j = 0, 1, \ldots, S-1$. Ikkunafunktiona käytetään 0–1 -välille skaalattua Gaussin kellokäyrää, jonka terävyyden määrittää parametri C (havainnollistettu kuvaajassa 12):

$$C_e = e^{-\frac{C}{4}} \qquad \text{Kellokäyrän marginaalin arvo}$$

$$w_j = \frac{e^{-C\cdot\left(\frac{j}{WR}-\frac{1}{2}\right)^2} - C_e}{1 - C_e} \qquad \text{Kellokäyrä}$$

$$\hat{w} = \sum w_j \qquad \text{Kellokäyrän normalisointiarvo}$$

$$y_j = x_{(j+\lfloor AR \rfloor)} \cdot \frac{w_j}{\hat{w}} \qquad \text{Ikkunoitu näytesarja}$$

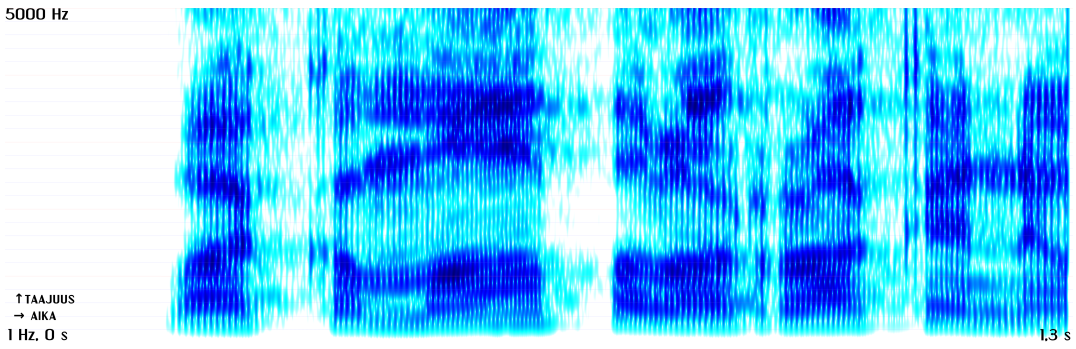
$$Y_j = \mathrm{FFT}(y)_j \qquad \text{Ikkunoitu taajuussarja (kompleksinen)}$$

$$X_j = |Y_j| \qquad \text{Ikkunoitu amplitudisarja (reaalinen).}$$

Amplitudisarjan havainnollistamista varten lineaariset amplitudit (X) muutetaan logaritmisiksi desibeleiksi (L) seuraavasti, missä $\varepsilon = 10^{-30}$ ja $20\,\mu$Pa äänenpainetta kuvaava vertailuteho $p = 4 \cdot 10^{-10}$:

$$O_j = 6 \log_2 \left(\frac{j + \varepsilon}{1000} \right) \qquad \text{Esivahvistus: 6 dB oktaavia kohti}$$

$$L_j = 10 \log_{10} \left(\frac{X_j + \varepsilon}{p} \right) + O_j \qquad \text{Logaritminen amplitudisarja.}$$



Kuva 13: Ikkunoitu amplitudispektri listauksen 32 esittämän ohjelman tuottamana. Tummempi väri tarkoittaa suurempaa amplitudia, vaaleampi pienempää. Akselit ovat lineaarisia.

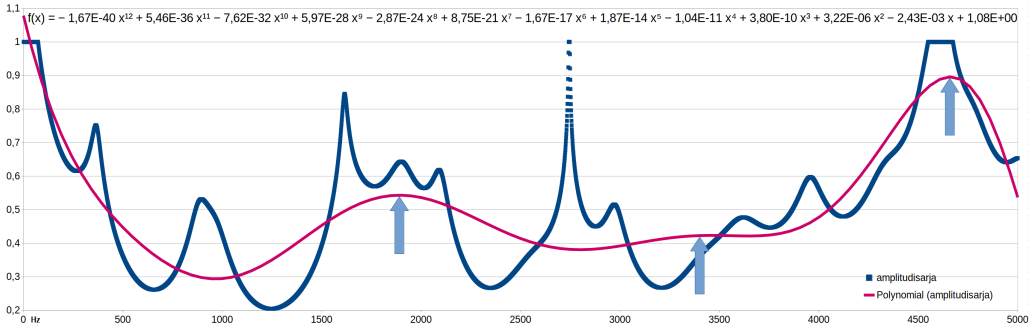
C++-ohjelma, joka suorittaa tämän kaiken laskennan, on esitetty listauksessa 32. Gaussin kellokäyrän skaalausvakiona toimi $C = 48$. Kuvaajaan on piirretty taajuusalue 1–5000 Hz ja amplitudit väliltä 60–90 dB (t.s. alle 60 dB amplitudit katsotaan minimiksi ja yli 90 dB amplitudit katsotaan maksimiksi). Ohjelma käyttää FFT-muunnokseen listauksessa 8 esitettyä `FFT_fftw`-funktiota, mutta lähes mikä tahansa tässä tutkielmissa esitetyistä FFT-/DFT-funktioista käy yhtä hyvin ja antaa saman tuloksen.

Listaus 32: fft-example1.cc

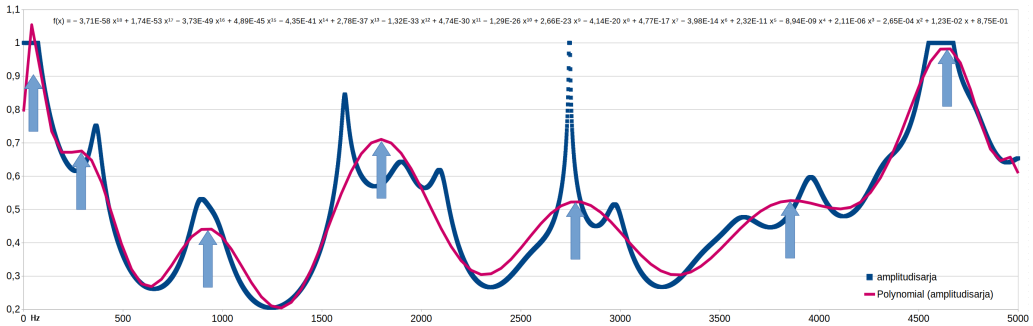
```
#include <algorithm>
#include <fstream>
#include <vector>
#include <cmath>
#include <cstdio>
#include <gd.h>
#include "fft_fftw.hh"
int main()
{
    /* Read input file */
    std::ifstream f("test.raw", std::ios::binary);
    f.seekg(0, std::ios::end);
    std::vector<char> buf(f.tellg());
    f.seekg(0, std::ios::beg);
    f.read(&buf[0], buf.size());
    /* Analyze the data */
    const float* data = reinterpret_cast<float*>(&buf[0]);
    std::size_t size = buf.size() / sizeof(float), rate = 44100;
    double length    = size / double(rate); /* Length in seconds */
    double increment = length/4000, window = 0.013;
    std::size_t maxfreq = 44100, interesting[2] = {0,5000}, x = 0;
    std::size_t columns = std::ceil(length/increment)+1;
    std::vector<double> result(maxfreq * columns);
    for(double start = 0; start < length; start += increment, ++x)
    {
        /* Generate complex sample */
        cvector sample(maxfreq);
        for(std::size_t p=start*rate, a=0; a<maxfreq; ++a,++p) { sample[a] = p < size ? data[p] : 0; }
        /* Apply Gaussian window */
        std::size_t wnd = window*rate;
        double wndsum = 0, C = 48, edge = std::exp(-C/4);
        for(std::size_t a=0; a<maxfreq; ++a)
        {
            double value = (std::exp(-C*std::pow(double(a)/wnd-0.5, 2.)) - edge) / (1-edge);
            sample[a] *= value; wndsum += value;
        }
        // Normalize it
        for(std::size_t a=0; a<maxfreq; ++a) sample[a] /= wndsum;
        /* Transform */
        sample = FFT_fftw(sample);
        /* Convert complex FFT into real power spectrum */
        std::vector<float> real(maxfreq/2);
        for(std::size_t a=0; a<maxfreq/2; ++a) { real[a] = std::abs(sample[a]); }
        /* Convert absolute signal to dB */
        for(std::size_t a=0; a<maxfreq/2; ++a) { real[a] = 10 * std::log10((real[a] + 1e-30) * 25e8) + 6*std::log2((a+!a)/1000.); }
        /* Clamp to 60 ... 90 dB range */
        float min = 60, max = 90;
        for(std::size_t a=0; a<maxfreq/2; ++a) { real[a] = std::clamp((real[a] - min) / (max-min), 0.f, 1.f); }
        /* Render into result */
        for(std::size_t a=0; a<maxfreq/2; ++a) { result[x*maxfreq + a] = real[a]; }
    }
    /* Convert the result into a picture */
    gdImagePtr im = gdImageCreateTrueColor(columns, interesting[1]-interesting[0]);
    for(std::size_t x=0; x<columns; ++x)
        for(std::size_t y=0; y < (interesting[1]-interesting[0]); ++y)
        {
            float f = std::pow(1 - result[x*maxfreq + (interesting[1]-y)], 1.2f);
            int b = f*768, g = b-256, r = g-256;
            unsigned color = (std::clamp(r,0,255)<<16)+(std::clamp(g,0,255)<<8)+(std::clamp(b,0,255)<<0);
            gdImageSetPixel(im, x,y, color);
            if(y % 200 == 0) gdImageSetPixel(im, x,y, 0x5555FF);
            if(y % 1000 == 0) gdImageSetPixel(im, x,y, 0xFF5555);
        }
    /* Write out the picture */
    FILE* fp = std::fopen("test.png", "wb");
    gdImagePng(im, fp);
    std::fclose(fp);
}
```

Tuloksena syntyneestä spektristä, kuvaajassa 13, voi selkeästi havaita useita tummansinisiä aaltoilevia akustisen energian keskittymiä eri taajuuksilla samanaikaisesti. Fourier-muunnoksen luonteesta johtuen niiden tarkan taajuuden määrittäminen on kuitenkin hankalaa. Määrittämiseen on kehitetty useita menetelmiä, mm. Burg-muunnos, joka käsittelee näytesarjaa aivan kuten Fourier-muunnoskin [6]. Koska tutkielman aihe kuitenkin on Fourier-muunnos, Burg-muunnokseen ei tässä yhteydessä keskitytä. Toinen vaihtoehto on muodostaa taajuusarvojen ja amplitudisarjan välille polynomisovitus ja selvittää polynomin huippukohdat[23]. Pienimmän neliösumman menetelmään perustuva polynomisovitus ei kuitenkaan aina onnistu kuvaamaan alkuperäistä taajuusspektriä luotettavasti, eivätkä polynomin huippukohdat täsmää taajuusspektrin kanssa, kuten nähdään kuvaajasta 14. Polynomin astelukua kasvattamalla sovituksesta saadaan parempi, mutta kovin täsmällinen se ei silti ole.

(a) Polynomin asteluku: 12



(b) Polynomin asteluku: 18



Kuva 14: Amplitudispektriin tehty polynomisovitus äänitallenteen kohdassa 0,26 s. Spektri esitetty tummansinisellä, polynomisovitus punaisella. Nuolilla on merkitty sovituksen ne lokaalit maksimit, jotka osuvat kuvatulle välille.

[23] Huippukohdat voi selvittää derivoimalla polynomisovitus, ratkaisemalla sen nollakohdat, ja valitsemalla niistä ne, joissa polynomin toisen derivaatan arvo on negatiivinen. Korkea-asteisen polynomin nollakohtien ratkaisemiseen voi käyttää mm. Newtonin-Raphsonin iteratiivista menetelmää.

Tässä esimerkkitapauksessa riittää kuitenkin paljon yksinkertaisempi menetelmä: jokaisen FFT:n jälkeen käydään läpi taulu 5 ja tutkitaan, millä rivillä kaava $L_{F_1}^2 + L_{F_2}^2$ antaa suurimman arvon, kun L_f tarkoittaa taajuuden f amplitudia amplitudisarjassa. Tämän menetelmän antamat tulokset on esitetty taulukossa 6. Taulukosta voidaan tehdä seuraavat havainnot:

- E-tyyppisten vokaalien (/e/, /ɛ/) tunnistus oli vahvinta ääninäytteen lopussa sekä ”vaikka”-sanan i-kirjaimen kohdalla.
- U-tyyppisten vokaalien (/ɯ/, /u/) tunnistus oli vahvinta ”vaikka”-sanan v-kirjaimen sekä klusiivin kohdalla.
- ”Pallo”-sanan o-kirjaimen äänne ei tunnistunut erityisen vahvasti O-tyyppisenä äänteenä (/o/, /ɔ/, /ɤ/). Muutenkin tunnistus oli vahvasti keskittynyt A-tyyppisiin äänteisiin (erityisesti /ɑ/ ja /ʌ/).

Tästä voidaan päätellä, että videon puhuja ei artikuloinut kovin selkeästi, vaikkakin riittävästi niin, että henkilö, jonka äidinkieli on suomi, ymmärtää puhetta aivan hyvin.

Aikaväli (s)		i	y	e	ɛ	ø	œ	a	æ	ɑ	ɒ	ʌ	ɔ	ɤ	o	ɯ	u
0,00–0,18	-	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0,18–0,22	ʊ	45	4	0	0	0	0	0	0	4	7	0	16	0	13	9	3
0,22–0,25	ɑ	0	0	0	0	0	0	0	0	8	33	58	1	0	0	0	0
0,25–0,30	i	0	0	11	6	0	0	4	10	15	15	21	2	16	0	1	0
0,30–0,36	-	4	0	1	0	0	0	0	3	0	43	12	8	17	0	12	0
0,36–0,40	k	0	0	0	0	0	0	0	21	0	10	6	0	54	0	8	0
0,40–0,43	ɑ	4	0	3	0	0	0	0	0	32	1	52	0	2	5	0	0
0,43–0,51	m	9	0	2	0	0	0	1	0	78	10	0	0	0	0	0	0
0,51–0,66	ɑ	0	0	0	0	0	0	0	0	75	9	11	0	1	0	0	2
0,66–0,74	p	5	0	0	0	0	0	5	2	29	26	17	5	2	3	5	1
0,74–0,77	ɑ	0	0	0	0	0	0	0	0	8	16	68	7	1	0	0	0
0,77–0,84	l	0	0	0	0	0	0	0	0	19	4	67	8	2	0	0	0
0,84–0,89	o	0	0	0	12	0	0	0	0	9	9	59	0	9	0	0	0
0,89–0,97	n	0	2	4	6	0	0	1	0	10	7	49	6	14	0	0	0
0,97–1,00	r	0	0	0	0	0	0	0	0	25	15	51	0	9	0	0	0
1,00–1,11	ɑ	1	0	3	1	0	6	5	4	18	25	11	1	23	0	1	0
1,11–1,14	k	2	0	29	21	0	44	3	0	0	0	3	0	0	0	0	0
1,14–1,20	e	13	0	18	1	0	36	10	5	15	0	0	2	0	0	0	0
1,20–1,27	n	19	0	42	2	0	6	15	10	0	5	0	0	0	0	0	0
1,27–1,30	e	18	0	15	0	0	0	0	0	8	29	13	0	16	0	0	0

Taulukko 6: Puheesta tunnistetut vokaalit formanttien perusteella. Jokaiselle riville määritettiin käsityönä tosiasiallisesti kuultava äänne. Numeroarvo kertoo, kuinka usein ohjelman valinta osui sarakkeen edustaman vokaalin kohdalle, prosentteina.

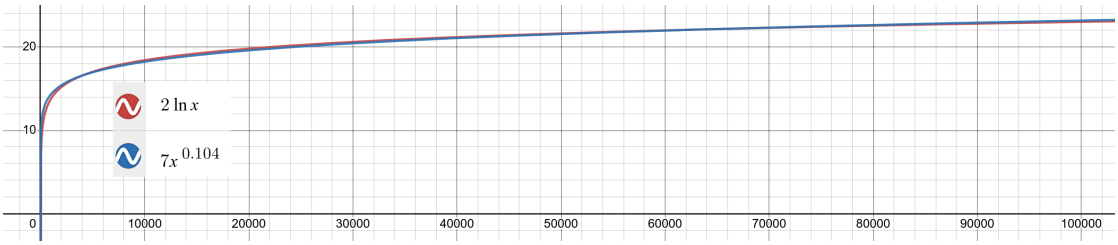
9. Pohdinta

9.1. Logaritminen funktio ja potenssifunktio

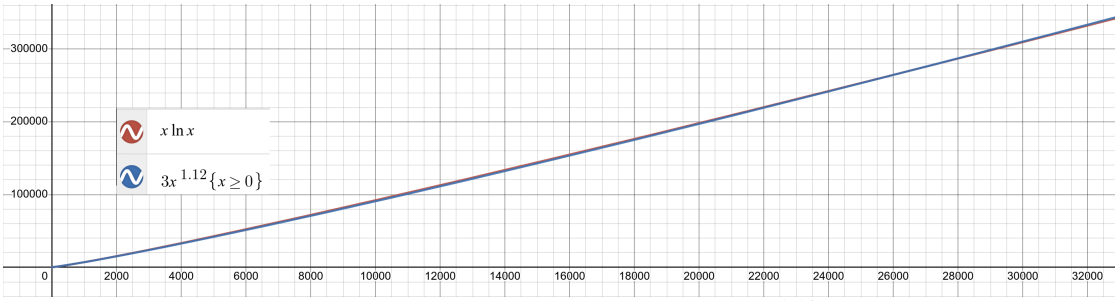
Kun tarkastelu rajataan jollekin tietylle alueelle, logaritmifunktio ja potenssifunktio antavat hyvin samankaltaisia tuloksia. Tämä on hyödyllinen ominaisuus silloin, kun halutaan verrata toisiinsa logaritmisia sekä ei-logaritmisia funktioita. Esimerkki tästä on esitetty kuvaajassa 15, jossa on vertailtu funktioita $f(x) = 2\ln x$ ja $f(x) = 7x^{0,104}$. Tällä valitulla välillä funktiot näyttävät lähes identtisiltä. Kuvaajassa 16 on vertailtu funktioita $f(x) = x\ln x$ ja $f(x) = 3x^{1,12}$, jotka niinikään näyttävät identtisiltä valitulla tarkasteluvälillä. Mikäli kuitenkin tarkasteluväli kasvatettaisiin kymmenkertaiseksi, nähtäisiin selvästi, että funktioiden kuvaajat erkanevat toisistaan. Oletetaan nyt pohdinnan kannalta, että funktio kuvaa jonkin algoritmin suoritusaikakäyttäytymistä.

Mikäli kahden funktion potenssit eroavat toisistaan, saadaan käsitys siitä, kumpi algoritmi toimii tehokkaammin suurilla syötteillä. Pienempi potenssi on tehokkaampi. Esimerkiksi mikäli $f(x) = Ax^2$ ja $g(x) = Bx^3$, missä A ja B ovat mielivaltaisia positiivisia reaalilukuja, on aina mahdollista löytää sellainen arvo y, jossa pätee $f(x) < g(x)$ kaikilla $x > y$. Potenssi, joka on lähellä ykköstä, tarkoittaa, että suoritusaikavaativuus on suurinpiirtein lineaarinen (esim. $f(x) = 3x$), ja potenssi, joka lähellä kakkosta (esim. $f(x) = 3x^2$), tarkoittaa, että suoritusaikavaativuus on suurinpiirtein neliöllinen. Potenssi näiden väliltä tarkoittaa, että algoritmin suoritusaikaan liittyy muitakin vaikuttavia tekijöitä, jotka voivat olla esim. logaritmisia.

Toisaalta mikäli potenssit ovat samaa luokkaa, mutta funktioiden vakiokertoimet eroavat toisistaan, voidaan päätellä, että toinen algoritmi on joka tilanteessa tietyn kertoimen verran toista tehokkaampi. Esimerkiksi jos $f(x) = 3x^{1,5}$ ja $g(x) = 30x^{1,5}$, niin $\frac{g(x)}{f(x)} = 10$ kaikilla muuttujan x positiivisilla arvoilla.



Kuva 15: Funktioiden $f(x) = 2\ln x$ ja $f(x) = 7x^{0,104}$ kuvaajat.



Kuva 16: Funktioiden $f(x) = x\ln x$ ja $f(x) = 3x^{1,12}$ kuvaajat.

DFT ja neliöllinen potenssi

Kuvaajasta 6 nähtiin, että naiivin DFT-algoritmin suoritusajan eksponentti ei ollut tarkalleen teoriaan täsmäävä 2, vaan hieman suurempi, 2,04. Tämä viittaa siihen, että algoritmin suoritusajassa oli neliöllisen komponentin lisäksi pieni lineaarinen komponentti. Tutkittaessa listausta 4 voidaan nähdä, että funktiossa `DFT` tapahtuu ulommassa silmukassa sijoitus kohdetaulukkoon, ja näitä sijoituksia tapahtuu lineaarinen määrä. Tämä ei ehkä ole ainoa syy, mikä saattaisi aiheuttaa pienen poikkeaman eksponenttiin – muita syitä voi ehkä löytää kääntäjän suorittamista SIMD-optimoinneista – mutta käytännössä voidaan silti sanoa, että naiivin DFT:n suoritusaikavaativuus on neliöllinen. Tästä seikasta voidaan myös johtaa se johtopäätös, että pienet erot potenssisovituksen eksponentissa ovat merkityksettömiä tarkastelun kannalta.

9.2. Yhdistetyn algoritmin toiminta

Seuraavia havaintoja tehtiin luvussa 5.6 esitellystä `FFT_any_fast`-funktiosta kerättäessä aineistoa luvun 7 mittauksia varten:

- Vaihtoehtoja vertaileva rutiini ei koskaan päätynyt valitsemaan yhden rekursion Cooleyn-Tukeyn algoritmia. Pelkästään tässä käytettyjen laskentakaavojen perusteella kahden rekursion versio oli aina tehokkaampi, kun mittarina toimivat yksinomaan laskutoimitusten lukumäärät ja lisämuistitilan tarve. Vaikka kuvaajasta 7 saattoikin päätellä, että itse asiassa päinvastainen olisi totta useammissa tapauksissa, tämä tapahtui tilanteessa, jossa alkulukupituisen syötteen muunnos tapahtui naiivilla DFT:llä. Yhdistetyn algoritmin tapauksessa näin ei ollut. Silti on mahdollista, että suoritusaikaan vaikutti jokin sellainen tekijä, jota ei voinut kvantifioida pelkästään laskutoimitusten lukumääriä laskemalla. Erot pitäisi käytännössä testata mittaamalla tapauskohtaisesti. Näin tekeekin FFTW silloin, kun sen annetaan laatia suunnitelma moodissa `FFTW_MEASURE`. Tässä tutkielmassa raja vedettiin kuitenkin pelkkään estimointiin.

- Cooleyn-Tukeyn algoritmin Radix-2-versiota käytettiin aina, mikäli syötteen pituus oli kahden potenssi. Sitä käytettiin usein myös, vaikka pituus sisältäisi vain yhden 2-tekijän.

- Bluesteinin algoritmin tapauksessa valinnat muuttujalle m olivat aina joko muotoa $2^a \cdot 3^b$ tai $2^a \cdot 5$, jossa $a \geqslant 5$ ja $0 \leqslant b \leqslant 2$. Missään tilanteessa tehokkain valinta ei ollut sellainen, jossa m sisältäisi tekijänään alkuluvun 7 tai sitä suuremman, tai ei sisältäisi tekijää 2. Mikäli ohjelma sisältäisi tehokkaan erikoistoteutuksen 7 alkion pituiselle syötteelle, tilanne olisi saattanut olla erilainen.

- Alle 2 miljoonan alkion mittaisissa syötteissä Bluesteinin algoritmia käytettiin n. 0,3 %:ssa tapauksista. Näistä noin puolet oli sellaisia tapauksia, joissa syötteen pituus N ei kuitenkaan ollut alkuluku. Tällöin N kuitenkin joko sisälsi yhden verrattain suuren tekijän suuruusluokkaa $\frac{1}{2}\sqrt{N}$, tai muuten koostui pienestä määrästä tekijöitä. Bluesteiniä ei valittu koskaan silloin, jos N oli parillinen.

- Raderin algoritmia käytettiin huomattavasti useammin pienillä syöteko'oilla kuin suuremmilla, mutta ei ollut havaittavissa mitään sellaista rajaa, jota suuremmilla syötteillä Raderin algoritmia ei käytettäisi ollenkaan.
- Käsin kirjoitetut tehokkaat DFT-algoritmit 1–4 alkion pituisille syötteille olivat myös algoritmin valinta näille ko'oille, ja niiden kirjoittaminen alunperinkin toi `FFT_any_fast`-funktion suoritusaikaa merkittävästi alaspäin. FFTW sisältää perusteellisesti optimoidut erikoistapaukset ko'oille 2–16, 20, 25, 32, 64 ja 128 (tiedostoissa `dft/simd/common/n1fv_*.c`), mikä on todennäköisesti suurin syy siihen, miksi se osoittautui mittauksissa yli kymmenen kertaa tehokkaammaksi kuin tämän tutkielman esimerkkikoodi.
- Kategorisesti päästiin kuitenkin samaan suorituskykyvaativuuteen kuin FFTW: esimerkkikoodi selviytyy muunnoksesta $\mathcal{O}(N \log N)$ -ajassa kaikilla mahdollisilla syötteen ko'oilla.

9.3. Laskentatarkkuus

Taulukosta 3 nähtiin, että syötteen pituuden lähestyessä satoja miljoonia alkoivat keskivirheet myös kasvaa ja lähestyä virheeksi määriteltyä kynnysarvoa 10^{-3}. On selkeää, että erot algoritmien välillä ovat suoraan verrannollisia laskuprosessin pituuteen johtuen liukulukulaskennan kasautuvista epätarkkuuksista[24]. Siksi tuloksia ei pidä tulkita niin, että algoritmi toimisi väärin kymmenien tai satojen miljoonien pituisilla syötteillä, mikäli tulokset kuitenkin ovat selkeästi tarkkoja lyhyemmillä syötteillä. Taulukkoa laadittaessa pohdittiin, olisiko virhearvot syytä jakaa syötteen pituudella epätarkkuuksien kasautumisilmiön kompensoimiseksi. Se todellakin yhdenmukaistaisi arvot – taulukon arvot vaihtelisivat 10^{-11} ja 10^{-15} välillä – mutta toisaalta se ehkä myös kätkisi todelliset laskuvirheet, mikäli sellaisia tapahtuisi.

9.4. Lisäkehitysideoita FFT:n nopeuttamiseksi

Kuten edellä tehdyistä tutkimuksista kävi ilmi, pelkästään se, että sanotaan, että FFT on mikä tahansa diskreetin Fourier-muunnoksen toteuttava algoritmi, jonka suoritusaikavaativuus on $\mathcal{O}(N \log N)$, ei itse asiassa kerro paljoakaan siitä, kuinka tehokas jokin nimenomainen toteutus on. On täysin mahdollista, että yksi algoritmi on tuhat kertaa hitaampi kuin toinen, ja kummankin suoritusaikavaativuus on silti tuo sama $\mathcal{O}(N \log N)$. Seuraavassa on listattuna muutamia ajatuksia, joilla FFT-laskentaa voi nopeuttaa enemmän kuin tässä tutkielmassa saavutettiin, sekä minkälaisia suorituskykyyn vaikuttavia seikkoja tutkielmassa esitettyjä ohjelmakoodeja hyödyntävän ohjelmiston suunnittelijan tulisi huomioida.

[24] Tarkemmin sanottuna erot algoritmien välillä ovat verrannollisia suoritettujen laskutoimitusten lukumäärään. Koska naiivi DFT suorittaa eniten laskutoimituksia, siksi tällä algoritmilla erot referenssituloksiin verrattuna kasvoivat nopeiten.

- FFT perustuu siihen, että suuri ongelma halkaistaan pieniin palasiin niin, että pohjimmiltaan laskenta suoritetaan suurella joukolla todella pieniä palasia. Tutkielman `DFT`-funktiossa on esitetty menetelmä, jolla 1–4 -kokoiset pienet palaset saadaan muunnettua varsin tehokkaasti, mutta vastaavat käsin tai automatiikalla laaditut todella tiiviisti optimoidut rutiinit olisi hyvä toteuttaa sitäkin suuremmille ko'oille. Malli näiden toteuttamiseksi on esitetty liitteessä III. Tämä aiheuttaisi kertautuvia nopeushyötyjä kaikilla syöteko'oilla.
- Tutkielmassa esitetty toteutus, joka käyttää `ExpVector`-luokkaa, ei aivan saavuta sitä tehokasta toimintaa, joka luokkaa suunniteltaessa oli mielessä. Tutkitaan esimerkiksi listauksessa 33 esitettyä koodia, joka on otos `FFT_any_fast`-funktiosta. Rivi 8 on peräisin todellisesta funktiosta, ja riveillä 10 ja 12 on esitetty selkeyden vuoksi, mitä rivin 8 koodi todella tarkoittaa. Kääntäjä pystyisi kyllä tehokkaasti optimoimaan kertolaskun käyttäen SIMD-tekniikkaa[25], mutta kokonaislukujen modulo-operaatiolle tällaista mahdollisuutta ei ole; tämän takia koko silmukka jää tältä osin optimoimatta. Mikäli koodirivi olisi rivin 14 mukainen, tällöin kääntäjä saisi optimoitua sen hyvin tehokkaasti. Ongelma on siinä, että muuttuja z saa arvoja väliltä 0 ja N^2-1. Jotta rivin 14 koodia voisi hyödyntää, exp-taulukon pitäisi olla kooltaan N^2 alkion mittainen. Jo muutaman tuhannen alkion pituisilla muunnoksilla tällaisen taulukon laskeminen ja tallentaminen muistiin olisi kohtuuton vaatimus. Voisi olla ehkä tehokkaampaa luopua `ExpVector`-tekniikasta kokonaan ja käyttää ennemmin jotain tehokasta menetelmää laskea exp-arvot suoraan silmukassa – käytännössä siis sin ja cos parametrillä $\left(\frac{-2\pi}{N}\right) z$ – ilman erillistä muistia, hyödyntäen tehokkaasti SIMD-toimintoja. Tällainen tehokas menetelmä saattaisi esimerkiksi hyödyntää Agner Fogin *Vector Class Library* -kirjastoa [26].
- Tietokoneille on tehokkaampaa käsitellä peräkkäisiä muistiosoitteita kuin yksittäisiä toisistaan etäällä olevia muistiosoitteita[26]. Mikäli muistiosoitteet ovat etäällä toisistaan, esimerkiksi jos silmukassa käsitellään joka kahdeksatta alkiota, tämä aiheuttaa sen, että kun ydin joka tapauksessa käsittelee keskusmuistia väylänleveyden suuruisissa yksiköissä, osa muistiliikenteestä jää hyödyntämättä. Lisäksi koska suoritusytimen välimuistiin ladataan kerrallaan tietyn suuruisia muistilohkoja, välimuisti tulee sitä huonommin hyödynnetyksi, mitä etäämpänä käsiteltävät muistiosoitteet ovat toisistaan [17]. Tämä seikka saattoi olla yksi vaikuttava tekijä, joka selittäisi, miksi yhden rekursion Cooleyn-Tukeyn algoritmi oli käytännössä usein nopeampi kuin kahden rekursion versio (kuvaaja 7), vaikka pelkästään kertolaskujen määrällä mitattuna asian pitäisi olla päinvastoin. Siksi saattaisi sittenkin olla eduksi käyttää laskennassa väliaikaistaulukoita joissakin tilanteissa. On kuitenkin vaikea ennustaa, missä tilanteissa tästä olisi hyötyä tai haittaa suorituskyvyn kannalta.

[25] SIMD = single instruction, multiple data: Tekniikka, jolla suoritusydin käsittelee useita arvoja yhdellä käskyllä, esimerkiksi kun yhdellä AVX-512-arkkitehtuurin `VMULPS`-käskyllä lasketaan samanaikaisesti 16 liukulukukertolaskua. Tehokkaasti hyödynnettynä SIMD-tekniikalla voidaan saavuttaa moninkertainen nopeutus perinteiseen skalaaritekniikkaan nähden.

[26] Ero ei ole yhtä dramaattinen kuin kiekkolevyissä, joissa siirtyminen eri kohtaan levyä vaatii mekaanisen siirtymän odotusaikoineen, mutta ero on silti olemassa ja todellinen.

Listaus 33: Esimerkki `ExpVector`-luokan käytöstä silmukassa.

```
for(std::size_t p = 0, n1 = 0; n1 < r2; ++n1)
    for(std::size_t q = 0, n0 = 0; n0 < r1; ++n0, ++p)          /* p käy 0...(N-1) */
    {                                                           /* Huom. N = r1*r2 */
        complex value = 0;
        for(std::size_t z = 0, k = 0; k < r2; ++k, z += p, ++q) /* q käy 0...(N-1) */
        {
            /* Lasketaan B_q * e^{-2πiz/N}, missä q = n0*r2 + k, p = n0 + n1*r1 ja z = k*n0 */
            value += buf[q] * exps(z, N);
            /* ↑ Todellinen koodirivi, ↓ kyseinen rivi käytännössä toimii näin: */
            value += buf[q] * exps->data[(z % N) * (exps->data.size() / N)];
            /* eli: */
            value += buf[q] * exp_data_storage[(z % N) * (exp_data_size / N)];
            /* Tämä sen sijaan olisi tehokas: */
            value += buf[q] * exp_data_storage[z];
        }
        output[p] = value; /* Laskettiin siis T_{(n0+n1*r1)} = sum_{k=0}^{r2-1} ( B_{(n0*r2+k)} * e^{-2πi*k*n0/N} ) */
    }
```

- Tutkielman FFT-funktioissa käytetään paljon memoize-tekniikkaa: hajautustaulua, johon tallennetaan funktion laskemat tiedot, jotta niitä voidaan hyödyntää myöhemmin, kun funktiota kutsutaan samoilla arvoilla. Tämä muistiin tallennettu tietomassa vie tietenkin muistista tilaa. Kun tutkielmaa varten suoritettiin nopeus- ja tarkkuustestejä tuhansilla erilaisilla syöteko'oilla, ohjelma kulutti kymmeniä gigatavuja virtuaalimuistia. Tämä aiheutti usein kohtuutonta järjestelmän hidastelua, jopa jumiutumisia niin, että tietokone piti käynnistää uudelleen. Muistia ei tulisi käsitellä rajattomana resurssina. Erityisesti silloin, jos ohjelma toimii pitkään, tätä muistia olisi hyvä joskus vapauttaakin.
- On olemassa muitakin FFT-algoritmeja kuin ne, jotka tutkielmassa on esitetty, esim. Rader-Brenner [4], Winograd [5, 7], ym. On mahdollista, että joissakin tilanteissa jokin näistä algoritmeista toimisi tehokkaammin kuin tutkielmassa toteutetut algoritmit.
- Voi olla tehokkaampaa laskea FFT käyttäen rinnakkaislaskentaa. Mikäli aineisto on tarpeeksi suuri, soveltuisi tähän tarkoitukseen jokin laskentakiihdytin, esimerkiksi näytönohjain (GPU). Pienillä aineistoilla tiedonsiirtoon näytönohjaimen ja keskusmuistin välillä kuluva aika todennäköisesti peittoaisi GPU-laskennasta saatavat hyödyt, mutta suurilla aineistoilla saattaisi olla mahdollista saavuttaa hyötyjä.
- On todistettu, että diskreetti Fourier-muunnos on mahdollista laskea neuroverkolla [14]. Neuroverkkoja on aikaisemmin käytetty onnistuneesti uusien tehokkaiden menetelmien löytämiseen matriisilaskentaan sekä tietojoukkojen lajitteluun [25, 27]. Tällaista lähestymistapaa on todennäköisesti mahdollista hyödyntää myös nopeampien FFT-algoritmien löytämiseksi.

Hakemisto

Viitteet

[1] James Cooley ja John Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". *Mathematics of Computation* 19.90 (1965), s. 297–301. DOI: 10.2307/2003354.

[2] Charles M. Rader. "Discrete Fourier transforms when the number of data samples is prime". *Proceedings of the IEEE* 56 (1968), s. 672–675.

[3] Leo Bluestein. "A linear filtering approach to the computation of discrete Fourier transform". *IEEE Transactions on Audio and Electroacoustics* 18.4 (1970), s. 451–455. DOI: 10.1109/TAU.1970.1162132.

[4] Charles M. Rader ja N. M. Brenner. "A new principle for fast Fourier transformation". *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24.3 (1976), s. 264–266. DOI: 10.1109/TASSP.1976.1162805.

[5] Shmuel Winograd. "On Computing the Discrete Fourier Transform". *Proc. Nat. Acad. Sci. USA* 73.4 (1976), s. 1005–1006. DOI: 10.1073/pnas.73.4.1005.

[6] Donald G. Childers. *Modern Spectrum Analysis*. USA: IEEE Press, 1978, s. 252–255. ISBN: 978-0879421083. URL: https://api.semanticscholar.org/CorpusID:118323255.

[7] Shmuel Winograd. "On Computing the Discrete Fourier Transform". *Mathematics of Computation* 32.141 (1978), s. 175–199.

[8] J. C. Catford. *A Practical Introduction to Phonetics*. Yhdistyneet kuningaskunnat: Oxford University Press, 1988, s. 161. ISBN: 978-0198242178. URL: https://archive.org/details/practicalintrodu00catf.

[9] Günter Löh. "Long chains of nearly doubled primes". *Mathematics of Computation* 53.188 (1989), s. 751–759. DOI: 10.1090/S0025-5718-1989-0979939-8.

[10] Gregory K. Wallace. "The JPEG still picture compression standard". *IEEE Trans. on Consumer Electronics* 38.1 (1992), s. xviii–xxxiv.

[11] Matteo Frigo ja Steven G. Johnson. "The Design and Implementation of FFTW3". *Proceedings of the IEEE* 93.2 (2005). Special issue on "Program Generation, Optimization, and Platform Adaptation", s. 216–231.

[12] Ilkka Mellin. *Tilastolliset menetelmät: Lineaarinen regressioanalyysi*. 2006. URL: http://math.aalto.fi/opetus/sovtodb/oppikirja/Regranal.pdf (viitattu 09. 11. 2023).

[13] Joel Yliluoma. *Guide into OpenMP: Easy multithreading programming for C++*. 2007. URL: https://bisqwit.iki.fi/story/howto/openmp/ (viitattu 08. 07. 2023).

[14] Rosemarie Velik. *Discrete Fourier Transform Computation Using Neural Networks*. 2008. URL: https://publik.tuwien.ac.at/files/PubDat_171587.pdf (viitattu 09. 11. 2023).

[15] James J. F. *A Student's Guide to Fourier Transforms : With Applications in Physics and Engineering*. Vol. 3rd ed. USA: Cambridge University Press, 2011. ISBN: 9780521176835. URL: https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=366247&site=ehost-live&scope=site.

[16] Robert Sedgewick ja Kevin Wayne. *Algorithms*. 4. painos. USA: Addison-Wesley Professional, 2011.

[17] Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. 2012. URL: https://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf (viitattu 09. 11. 2023).

[18] Jokke Häsä ja Johanna Rämö. *Johdatus abstraktiin algebraan*. 3. painos. Helsinki: Gaudeamus, 2015. ISBN: 978-952-495-756-4.

[19] Humberto Ochoa-Domínguez ja K. R. Rao. *Discrete Cosine Transform*. 2. painos. New York: CRC Press, 2019.

[20] Øyvind Ryan. *Linear Algebra, Signal Processing, and Wavelets - A Unified Approach: Python version*. Sveitsi: Springer Nature Switzerland AG, 2019. DOI: 10.1007/978-3-030-02940-1.

[21] Jamie Ward. *The Student's Guide to Cognitive Neuroscience*. 4. painos. Lontoo: Taylor & Francis, 2019. ISBN: 978-1351035163. URL: https://books.google.fi/books?id=5kDCDwAAQBAJ.

[22] Free Software Foundation. *GNU Coding Standards*. 2020. URL: https://www.gnu.org/prep/standards/ (viitattu 18. 10. 2023).

[23] ISO Central Secretary. *Programming languages – C++*. E. Standard ISO/IEC 14882:2020. Geneva, CH: International Organization for Standardization, 2020. URL: https://www.iso.org/standard/79358.html.

[24] Jacob John. *Discrete Cosine Transform in JPEG Compression*. 2021. arXiv: 2102.06968v1 [cs.CV].

[25] Fawzi Alhussein et al. "Discovering faster matrix multiplication algorithms with reinforcement learning". *Nature* 610 (2022), s. 47–53. DOI: 10.1038/s41586-022-05172-4.

[26] Agner Fog. *vectorclass: The Vector Class Library is a C++ tool that allows programmers to use Single Instruction Multiple Data (SIMD) instructions to process data in parallel*. 2023. URL: https://github.com/vectorclass (viitattu 09. 11. 2023).

[27] Mankowitz Daniel J. et al. "Faster sorting algorithms discovered using deep reinforcement learning". *Nature* 618 (2023), s. 257–263. DOI: 10.1038/s41586-023-06004-9.

[28] Dustin Podell et al. *SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis*. 2023. arXiv: 2307.01952 [cs.CV].

[29] Massachusetts Institute of Technology. *Thread safety (FFTW)*. URL: http://www.fftw.org/doc/Thread-safety.html (viitattu 18. 10. 2023).

Liitteet

I. Nopeampi tekijöihin jako

Tutkielman sivulla 32 esitetty toteutus tiedostolle `factor.cc` oli tarkoituksella laadittu paperitilan kannalta lyhyeksi, muttei kovin tehokkaaksi. Tässä liitteessä listauksessa 34 on esitetty ajoitusmittauksissa käytetty todellinen koodi, jonka tarkoitus on minimoida tekijöihin jakamiseen kuluvan ajan vaikutus suorituskykymittauksissa. Otsikkotiedosto `factor.hh` on edelleen sama kuin listauksessa 14.

Listaus 34: factor.cc

```
#include <set>
#include <cmath>
#include <mutex>
#include <vector>
#include <type_traits>
#include <unordered_map>

#include "factor.hh"

/* This structure caches prime-related information. */
static class lore
{
    std::unordered_map<std::size_t, std::size_t> smallest_factors{ {0,0}, {1,1}, {2,2} };
    std::unordered_map<std::size_t, std::size_t> half_factors{};
    std::vector<signed char> is_prime{1,1,1}; // 0=not known, -1=no, 1=yes
    std::size_t              lastprime{2};    // extent of contiguous knowledge
    std::set<std::size_t>    primes{2};       // list of known primes
    std::mutex lock{};

public:
    auto get_lock() { return std::unique_lock<std::mutex>(lock); }

    template<typename T>
    std::size_t find(std::size_t N, T&&)
        requires std::is_same_v<std::remove_reference_t<T>,std::unique_lock<std::mutex>>
    {
        if(auto i = smallest_factors.find(N); i != smallest_factors.end()) return i->second;
        std::size_t solution = N;
        for(auto p: primes)
        {
            if(N %  p == 0) { solution = p; break; }
            if(p*p >= N) { break; }
        }
        for(std::size_t p = lastprime|1; p*p <= N; p+=2) if(N %  p == 0) { solution = p; break; }
        if(is_prime.size() <= N) is_prime.resize(N+1);
        for(is_prime[N] = (solution==N) ? 1 : -1;
            lastprime+1 < is_prime.size() && is_prime[lastprime+1] != 0; ) ++lastprime;
        if(solution == N) primes.insert(solution);
        return smallest_factors.emplace(N, solution).first->second;
    }

    std::size_t find_half(std::size_t N, std::unique_lock<std::mutex>&&)
    {
        if(auto i = half_factors.find(N); i != half_factors.end()) return i->second;
        std::size_t result = N, a = std::sqrt(N);
        for(a |= 1; a > 1; a -= (a>3?2:1)) { if(N %  a == 0) { result = a; break; } }
        half_factors.emplace(N, result);
```

```
48          return result;
49      }
50
51      lore()
52      {
53          auto l = get_lock();
54          for(std::size_t a=0; a<1048576; ++a)
55          {
56              find(a, l);
57              if(a<31) find(1<<a, l);
58          }
59      }
60  } prime_lore;
61
62  std::size_t smallest_factor(std::size_t N)
63  {
64      return prime_lore.find(N, prime_lore.get_lock());
65  }
66
67  std::size_t get_factors(std::size_t N, std::size_t* target, std::size_t limit)
68  {
69      auto lock = prime_lore.get_lock();
70      for(std::size_t orig=N,count=0,prev=0,f; ; N/=f)
71          if(f=prime_lore.find(N, lock); count >= limit || f <= 1 || f == orig) { return count; }
72          else if(f != prev) { target[count++] = prev = f; }
73  }
74
75  bool small_factors_only(std::size_t N, std::size_t upto)
76  {
77      auto lock = prime_lore.get_lock();
78      for(std::size_t f; ; N/=f)
79          { f = prime_lore.find(N, lock); if(f < 2 || f > upto) return f == 1; }
80  }
81
82  bool small_factors_mostly(std::size_t N, std::size_t upto)
83  {
84      auto lock = prime_lore.get_lock();
85      for(std::size_t n=0, f; ; N/=f)
86          { f = prime_lore.find(N, lock); if(f < 2 || f > upto) return f == 1 || n >= 2; }
87  }
88
89  std::size_t half_factors(std::size_t N)
90  {
91      return prime_lore.find_half(N, prime_lore.get_lock());
92  }
```

II. Kuvaajien piirtäminen

Aluksi luvussa 7 esitettyjen kuvaajien piirtämiseen käytettiin Libreoffice Calc -ohjelmistoa. Mutta kun aineistojen koko kasvoi suureksi niin, että taulukossa oli kymmeniä tuhansia rivejä ja kymmenen saraketta, osoittautui, ettei Calc selviytynyt enää hommasta. Calcin käyttöliittymä hidastui käyttökelvottomaksi. Tämän takia oli laadittava oma työkalu kuvaajien piirtämiseen. Työkalun lähdekoodi on ladattavissa osoitteessa `https://github.com/bisqwit/fft/blob/master/fft/render.cc`. Työkalu tarvitsee avukseen libcairo-kirjaston. Tutkielman kirjoittajan työasemassa tämä asennettiin komennolla `apt-get install libcairo2-dev`, joka asensi version 1.18.0.

Työkalu lukee työhakemistosta tiedoston `tmp.csv`, jonka se olettaa sisältävän CSV-tiedoston, jossa erotinmerkkinä toimii pilkku ja desimaalierottimena piste; ensimmäinen rivi on otsikkorivi, ensimmäisessä sarakkeessa on jaksopituus, ja seuraavissa sarakkeissa kunkin menetelmän kyseisellä jaksopituudella kuluttama aika millisekunteina.

Koko projektin lähdekoodi löytyy osoitteesta `https://github.com/bisqwit/fft/tree/thesis`.

III. Nopeat FFT-erikoistoteutukset

Nämä silmukattomat erikoistoteutukset nopealle Fourier-muunnokselle on poimittu FFTW-kirjaston (versio 3.3.10) tiedostoista `dft/simd/common/n1fv_*.c` (menetelmät, joissa asetus `PREFERS_FMA` on epätosi) ja käännetty C-kielestä LaTeXiksi. Listattuna on toteutukset syöteko'oille 0–16, 20, 25, 32 ja 64. FFTW sisältää myös erikoistoteutuksen ko'olle 128, mutta se jätettiin pois tästä liitteestä, koska sen listausta ei onnistuttu mahduttamaan yhdelle A4-arkille. Tutkielmassa tiedostossa `dft.cc` toteutettiin erikoistapaukset $N = 0 \ldots 4$ suunnilleen samalla tavalla kuin tässä liitteessä, joskin alla esitetty menetelmä tapaukselle $N = 3$ on aavistuksen tehokkaampi. Muuttujat muotoa x_n viittaavat syötteen alkioihin, muuttujat X_n tuloksen alkioihin, t_n välituloksiin, k_n reaalisiin numerovakioihin ja i imaginääriyksikköön. Huom. nämä listaukset ovat automatiikalla laadittuja tulkkauksia koodista, eikä listausten toimivuutta ole testattu ja varmistettu pistokoeluonteista yksittäisten lausekkeiden tarkastamista lukuunottamatta.

FFT, $N = 3$

$k_0 = 1/2$	$k_1 = \sqrt{3}/2$
$t_0 = x_1 + x_2$	$t_1 = ik_1(x_2 - x_1)$
$X_0 = x_0 + t_0$	$t_2 = x_0 - k_0 t_0$
$X_2 = t_2 - t_1$	$X_1 = t_2 + t_1$

FFT, $N = 9$

$k_0 = \sin(\pi/9)$	$k_1 = \cos(\pi/9)$
$k_2 = \sin(2\pi/9)$	$k_3 = \cos(2\pi/9)$
$k_4 = \sin(4\pi/9)$	$k_5 = \cos(4\pi/9)$
$k_6 = 1/2$	$k_7 = \sqrt{3}/2$
$k_8 = \sin(\pi/3)\sin(4\pi/9)$	
$k_9 = \sin(\pi/3)\cos(4\pi/9)$	
$k_a = \sin(\pi/9)\sin(\pi/3)$	
$k_b = \cos(\pi/9)\sin(\pi/3)$	
$k_c = \sin(2\pi/9)\sin(\pi/3)$	
$k_d = \cos(2\pi/9)\sin(\pi/3)$	
$t_0 = x_3 + x_6$	$t_1 = x_0 + t_0$
$t_2 = k_7(x_6 - x_3)$	$t_3 = x_0 - k_6 t_0$
$t_4 = x_5 + x_8$	$t_5 = x_8 - x_5$
$t_6 = x_2 + t_4$	$t_7 = x_2 - k_6 t_4$
$t_8 = k_8 t_5 + k_5 t_7$	$t_9 = k_9 t_5 - k_4 t_7$
$t_a = x_4 + x_7$	$t_b = x_7 - x_4$
$t_c = x_1 + t_a$	$t_d = x_1 - k_6 t_a$
$t_e = k_c t_b + k_3 t_d$	$t_f = k_d t_b - k_2 t_d$
$t_g = ik_7(t_6 - t_c)$	$t_h = t_c + t_6$
$t_i = t_1 - k_6 t_h$	$X_0 = t_1 + t_h$
$X_3 = t_i + t_g$	$X_6 = t_i - t_g$
$t_j = t_3 - k_8 t_b - k_1 t_7 - k_a t_5 + k_5 t_d$	
$t_k = i(k_b t_5 - k_4 t_d - k_9 t_b - k_0 t_7 - t_2)$	
$X_7 = t_j - t_k$	$X_2 = t_j + t_k$
$t_l = t_e + t_8$	$t_m = t_3 + t_l$
$t_n = t_3 - k_6 t_l + k_7(t_f - t_9)$	
$t_o = t_f + t_9$	$t_p = i(t_2 + t_o)$
$t_q = i(t_2 + k_7(t_8 - t_e) - k_6 t_o)$	
$X_8 = t_m - t_p$	$X_4 = t_n + t_q$
$X_1 = t_p + t_m$	$X_5 = t_n - t_q$

FFT, $N = 4$

$t_0 = x_0 - x_2$	$t_1 = x_0 + x_2$
$t_2 = i(x_1 - x_3)$	$t_3 = x_1 + x_3$
$X_1 = t_0 - t_2$	$X_0 = t_1 + t_3$
$X_3 = t_0 + t_2$	$X_2 = t_1 - t_3$

FFT, $N = 2$

$X_1 = x_0 - x_1$	$X_0 = x_0 + x_1$

FFT, $N = 0$

FFT, $N = 1$

$X_0 = x_0$

FFT, $N = 6$

$k_0 = \sqrt{3}/2$	$k_1 = 1/2$
$t_0 = x_0 - x_3$	$t_1 = x_0 + x_3$
$t_2 = x_2 - x_5$	$t_3 = x_2 + x_5$
$t_4 = x_4 - x_1$	$t_5 = x_4 + x_1$
$t_6 = t_2 + t_4$	$t_7 = t_3 + t_5$
$X_3 = t_0 + t_6$	$X_0 = t_1 + t_7$
$t_8 = t_0 - k_1 t_6$	$t_9 = ik_0(t_4 - t_2)$
$X_5 = t_8 - t_9$	$X_1 = t_8 + t_9$
$t_a = t_1 - k_1 t_7$	$t_b = ik_0(t_5 - t_3)$
$X_2 = t_a - t_b$	$X_4 = t_a + t_b$

FFT, $N = 8$

$k_0 = \sqrt{2}/2$	$t_0 = x_0 - x_4$
$t_1 = x_0 + x_4$	$t_2 = x_2 - x_6$
$t_3 = x_2 + x_6$	$t_4 = x_1 - x_5$
$t_5 = x_7 - x_3$	$t_6 = k_0(t_4 + t_5)$
$t_7 = x_7 + x_3$	$t_8 = k_0(t_5 - t_4)$
$t_9 = x_1 + x_5$	$t_a = t_0 + t_6$
$t_b = i(t_8 - t_2)$	$X_7 = t_a - t_b$
$X_1 = t_a + t_b$	$t_c = t_1 - t_3$
$t_d = i(t_7 - t_9)$	$X_6 = t_c - t_d$
$X_2 = t_c + t_d$	$t_e = t_0 - t_6$
$t_f = i(t_2 + t_8)$	$X_5 = t_e - t_f$
$X_3 = t_e + t_f$	$t_g = t_1 + t_3$
$t_h = t_9 + t_7$	$X_4 = t_g - t_h$
$X_0 = t_g + t_h$	

FFT, $N = 5$

$k_0 = 1/4$ | $k_1 = \sin(\pi/5)$ | $k_2 = \sin(2\pi/5)$ | $k_3 = \sqrt{5}/4$ | $t_0 = x_1 + x_4$
$t_1 = x_2 + x_3$ | $t_2 = k_3(t_0 - t_1)$ | $t_3 = x_2 - x_3$ | $t_4 = t_0 + t_1$ | $t_5 = x_1 - x_4$
$X_0 = x_0 + t_4$ | $t_6 = i(k_1 t_3 + k_2 t_5)$ | | $t_7 = i(k_2 t_3 - k_1 t_5)$ |
$t_8 = x_0 - k_0 t_4$ | $t_9 = t_2 + t_8$ | $t_a = t_8 - t_2$ | $X_1 = t_9 - t_6$ | $X_3 = t_a - t_7$
$X_4 = t_6 + t_9$ | $X_2 = t_7 + t_a$ | | | |

FFT, $N = 64$

$k_0 = \sin(5\pi/32)$ | $k_1 = \cos(5\pi/32)$ | $k_2 = \sin(\pi/32)$ | $k_3 = \cos(\pi/32)$ | $k_4 = \sin(3\pi/32)$ | $k_5 = \cos(3\pi/32)$
$k_6 = \sin(7\pi/32)$ | $k_7 = \cos(7\pi/32)$ | $k_8 = \sin(3\pi/16)$ | $k_9 = \cos(3\pi/16)$ | $k_a = \cos(\pi/16)$ | $k_b = \sin(\pi/16)$
$k_c = \sin(\pi/8)$ | $k_d = \cos(\pi/8)$ | $k_e = \sqrt{2}/2$ | $t_0 = x_0 - x_{32}$ | $t_1 = x_0 + x_{32}$ | $t_2 = x_{16} - x_{48}$
$t_3 = x_{16} + x_{48}$ | $t_4 = x_8 - x_{40}$ | $t_5 = x_8 + x_{40}$ | $t_6 = x_{56} - x_{24}$ | $t_7 = x_{56} + x_{24}$ | $t_8 = t_1 - t_3$
$t_9 = t_7 - t_5$ | $t_a = k_e(t_4 + t_6)$ | $t_b = t_0 + t_a$ | $t_c = t_0 - t_a$ | $t_d = k_e(t_6 - t_4)$ | $t_e = t_d - t_2$
$t_f = t_2 + t_d$ | $t_g = t_1 + t_3$ | $t_h = t_5 + t_7$ | $t_i = t_g + t_h$ | $t_j = t_g - t_h$ | $t_k = x_4 - x_{36}$
$t_l = x_4 + x_{36}$ | $t_m = x_{12} - x_{44}$ | $t_n = x_{12} + x_{44}$ | $t_o = x_{20} - x_{52}$ | $t_p = x_{20} + x_{52}$ | $t_q = x_{60} - x_{28}$
$t_r = x_{60} + x_{28}$ | $t_s = k_d t_k - k_c t_o$ | $t_t = k_c t_m + k_d t_q$ | $t_u = t_s + t_t$ | $t_v = t_t - t_s$ | $t_w = t_l + t_p$
$t_x = t_r + t_n$ | $t_y = t_w + t_x$ | $t_z = t_x - t_w$ | $t_A = k_c t_q - k_d t_m$ | $t_B = k_d t_o + k_c t_k$ | $t_C = t_A - t_B$
$t_D = t_B + t_A$ | $t_E = t_l - t_p$ | $t_F = t_r - t_n$ | $t_G = k_e(t_E + t_F)$ | $t_H = k_e(t_F - t_E)$ | $t_I = x_{62} - x_{30}$
$t_J = x_{62} + x_{30}$ | $t_K = x_{14} - x_{46}$ | $t_L = x_{14} + x_{46}$ | $t_M = x_6 - x_{38}$ | $t_N = x_{54} - x_{22}$ | $t_O = k_e(t_M + t_N)$
$t_P = x_6 + x_{38}$ | $t_Q = k_e(t_N - t_M)$ | $t_R = x_{54} + x_{22}$ | $t_S = t_I + t_O$ | $t_T = t_Q - t_K$ | $t_U = k_a t_T + k_b t_S$
$t_V = k_a t_S - k_b t_T$ | $t_W = t_J + t_L$ | $t_X = t_P + t_R$ | $t_Y = t_W + t_X$ | $t_Z = t_W - t_X$ | $t_{10} = t_I - t_O$
$t_{11} = t_K + t_Q$ | $t_{12} = k_9 t_{10} - k_8 t_{11}$ | | $t_{13} = k_9 t_{11} + k_8 t_{10}$ | | $t_{14} = t_J - t_L$
$t_{15} = t_R - t_P$ | $t_{16} = k_d t_{14} - k_c t_{15}$ | | $t_{17} = k_d t_{15} + k_c t_{14}$ | | $t_{18} = x_{18} - x_{50}$
$t_{19} = x_{18} + x_{50}$ | | $t_{1a} = x_2 - x_{34}$ | $t_{1b} = x_2 + x_{34}$ | $t_{1c} = x_{58} - x_{26}$ | $t_{1d} = x_{10} - x_{42}$
$t_{1e} = k_e(t_{1c} - t_{1d})$ | | $t_{1f} = x_{10} + x_{42}$ | $t_{1g} = k_e(t_{1d} + t_{1c})$ | | $t_{1h} = x_{58} + x_{26}$
$t_{1i} = t_{1e} - t_{18}$ | $t_{1j} = t_{1a} + t_{1g}$ | $t_{1k} = k_a t_{1i} - k_b t_{1j}$ | | $t_{1l} = k_b t_{1i} + k_a t_{1j}$ |
$t_{1m} = t_{1b} + t_{19}$ | $t_{1n} = t_{1f} + t_{1h}$ | $t_{1o} = t_{1m} + t_{1n}$ | $t_{1p} = t_{1m} - t_{1n}$ | $t_{1q} = t_{1a} - t_{1g}$ | $t_{1r} = t_{18} + t_{1e}$
$t_{1s} = k_8 t_{1r} + k_9 t_{1q}$ | | $t_{1t} = k_9 t_{1r} - k_8 t_{1q}$ | | $t_{1u} = t_{1b} - t_{19}$ | $t_{1v} = t_{1h} - t_{1f}$
$t_{1w} = k_c t_{1v} + k_d t_{1u}$ | | $t_{1x} = k_d t_{1v} - k_c t_{1u}$ | | $t_{1y} = x_{63} - x_{31}$ | $t_{1z} = x_{63} + x_{31}$
$t_{1A} = x_{15} - x_{47}$ | | $t_{1B} = x_{15} + x_{47}$ | $t_{1C} = x_{39}$ | $t_{1D} = x_7 - t_{1C}$ | $t_{1E} = x_{55} - x_{23}$
$t_{1F} = k_e(t_{1D} + t_{1E})$ | | $t_{1G} = x_7 + t_{1C}$ | $t_{1H} = k_e(t_{1E} - t_{1D})$ | | $t_{1I} = x_{55} + x_{23}$
$t_{1J} = x_3 - x_{35}$ | | $t_{1K} = x_3 + x_{35}$ | $t_{1L} = x_{19}$ | $t_{1M} = t_{1L} - x_{51}$ | $t_{1N} = t_{1L} + x_{51}$
$t_{1O} = k_d t_{1J} - k_c t_{1M}$ | | $t_{1P} = t_{1K} - t_{1N}$ | $t_{1Q} = k_d t_{1M} + k_c t_{1J}$ | | $t_{1R} = x_{59}$
$t_{1S} = t_{1R} - x_{27}$ | $t_{1T} = t_{1R} + x_{27}$ | $t_{1U} = x_{11} - x_{43}$ | $t_{1V} = x_{11} + x_{43}$ | $t_{1W} = k_c t_{1U} + k_d t_{1S}$ |
$t_{1X} = t_{1T} - t_{1V}$ | $t_{1Y} = k_c t_{1S} - k_d t_{1U}$ | | $t_{1Z} = t_{1y} + t_{1F}$ | $t_{20} = t_{1O} + t_{1W}$ | $t_{21} = t_{1Z} - t_{20}$
$t_{22} = t_{1Z} + t_{20}$ | $t_{23} = t_{1z} + t_{1B}$ | $t_{24} = t_{1G} + t_{1I}$ | $t_{25} = t_{23} + t_{24}$ | $t_{26} = t_{23} - t_{24}$ | $t_{27} = t_{1K} + t_{1N}$
$t_{28} = t_{1T} + t_{1V}$ | $t_{29} = t_{27} + t_{28}$ | $t_{2a} = t_{28} - t_{27}$ | $t_{2b} = t_{1Y} - t_{1Q}$ | $t_{2c} = t_{1H} - t_{1A}$ | $t_{2d} = t_{2b} - t_{2c}$
$t_{2e} = t_{2c} + t_{2b}$ | $t_{2f} = t_{1y} - t_{1F}$ | $t_{2g} = t_{1Q} + t_{1Y}$ | $t_{2h} = t_{2f} + t_{2g}$ | $t_{2i} = t_{2f} - t_{2g}$ | $t_{2j} = t_{1z} - t_{1B}$
$t_{2k} = k_e(t_{1P} + t_{1X})$ | | | $t_{2l} = t_{2j} + t_{2k}$ | $t_{2m} = t_{2j} - t_{2k}$ | $t_{2n} = t_{1I} - t_{1G}$
$t_{2o} = k_e(t_{1X} - t_{1P})$ | | | $t_{2q} = t_{2o} - t_{2n}$ | $t_{2r} = t_{1A} + t_{1H}$ | $t_{2s} = t_{1W} - t_{1O}$
$t_{2t} = t_{2r} + t_{2s}$ | $t_{2u} = t_{2s} - t_{2r}$ | $t_{2v} = x_{49}$ | $t_{2w} = x_{17} - t_{2v}$ | $t_{2x} = x_{17} + t_{2v}$ | $t_{2y} = x_1 - x_{33}$
$t_{2z} = x_1 + x_{33}$ | $t_{2A} = x_{57} - x_{25}$ | $t_{2B} = x_9$ | $t_{2C} = t_{2B} - x_{41}$ | $t_{2D} = k_e(t_{2A} - t_{2C})$ |
$t_{2E} = t_{2B} + x_{41}$ | $t_{2F} = k_e(t_{2C} + t_{2A})$ | | $t_{2G} = x_{57} + x_{25}$ | $t_{2H} = x_{29}$ | $t_{2I} = x_{61} - t_{2H}$
$t_{2J} = x_{61} + t_{2H}$ | $t_{2K} = x_{13} - x_{45}$ | $t_{2L} = x_{13} + x_{45}$ | $t_{2M} = k_c t_{2I} - k_d t_{2K}$ | | $t_{2N} = t_{2J} - t_{2L}$
$t_{2O} = k_c t_{2K} + k_d t_{2I}$ | | $t_{2P} = x_5 - x_{37}$ | $t_{2Q} = x_5 + x_{37}$ | $t_{2R} = x_{21} - x_{53}$ | $t_{2S} = x_{21} + x_{53}$
$t_{2T} = k_d t_{2R} + k_c t_{2P}$ | | $t_{2U} = t_{2Q} - t_{2S}$ | $t_{2V} = k_d t_{2P} - k_c t_{2R}$ | | $t_{2W} = t_{2M} - t_{2T}$
$t_{2X} = t_{2D} - t_{2w}$ | $t_{2Y} = t_{2W} - t_{2X}$ | $t_{2Z} = t_{2X} + t_{2W}$ | $t_{30} = t_{2z} + t_{2x}$ | $t_{31} = t_{2E} + t_{2G}$ | $t_{32} = t_{30} + t_{31}$
$t_{33} = t_{30} - t_{31}$ | $t_{34} = t_{2Q} + t_{2S}$ | $t_{35} = t_{2J} + t_{2L}$ | $t_{36} = t_{34} + t_{35}$ | $t_{37} = t_{35} - t_{34}$ | $t_{38} = t_{2y} + t_{2F}$
$t_{39} = t_{2V} + t_{2O}$ | $t_{3a} = t_{38} - t_{39}$ | $t_{3b} = t_{38} + t_{39}$ | $t_{3c} = t_{2y} - t_{2F}$ | $t_{3d} = t_{2T} + t_{2M}$ | $t_{3e} = t_{3c} + t_{3d}$
$t_{3f} = t_{3c} - t_{3d}$ | $t_{3g} = t_{2z} - t_{2x}$ | $t_{3h} = k_e(t_{2U} + t_{2N})$ | | $t_{3i} = t_{3g} + t_{3h}$ | $t_{3j} = t_{3g} - t_{3h}$
$t_{3k} = t_{2G} - t_{2E}$ | $t_{3l} = k_e(t_{2N} - t_{2U})$ | | $t_{3m} = t_{3k} + t_{3l}$ | $t_{3n} = t_{3l} - t_{3k}$ | $t_{3o} = t_{2w} + t_{2D}$
$t_{3p} = t_{2O} - t_{2V}$ | $t_{3q} = t_{3o} + t_{3p}$ | $t_{3r} = t_{3p} - t_{3o}$ | $t_{3s} = t_i + t_y$ | $t_{3t} = t_{1o} + t_Y$ | $t_{3u} = t_{3s} + t_{3t}$
$t_{3v} = t_{3s} - t_{3t}$ | $t_{3w} = t_{32} + t_{36}$ | $t_{3x} = t_{25} + t_{29}$ | $t_{3y} = t_{3w} + t_{3x}$ | $t_{3z} = i(t_{3x} - t_{3w})$ | $X_{32} = t_{3u} - t_{3y}$
$X_{16} = t_{3v} + t_{3z}$ | $X_0 = t_{3u} + t_{3y}$ | $X_{48} = t_{3v} - t_{3z}$ | $t_{3A} = t_i - t_y$ | $t_{3B} = t_Y - t_{1o}$ | $t_{3C} = t_{32} - t_{36}$
$t_{3D} = t_{25} - t_{29}$ | $t_{3E} = k_e(t_{3C} + t_{3D})$ | | $t_{3F} = k_e(t_{3D} - t_{3C})$ | | $t_{3G} = t_{3A} + t_{3E}$
$t_{3H} = i(t_{3B} + t_{3F})$ | $X_{56} = t_{3G} - t_{3H}$ | $X_8 = t_{3G} + t_{3H}$ | $t_{3I} = t_{3A} - t_{3E}$ | $t_{3J} = i(t_{3F} - t_{3B})$ | $X_{40} = t_{3I} - t_{3J}$
$X_{24} = t_{3I} + t_{3J}$ | $t_{3K} = k_e(t_{1p} + t_Z)$ | $t_{3L} = t_j + t_{3K}$ | $t_{3M} = t_j - t_{3K}$ | $t_{3N} = k_e(t_Z - t_{1p})$ | $t_{3O} = t_z + t_{3N}$
$t_{3P} = t_{3N} - t_z$ | $t_{3Q} = k_c t_{37} + k_d t_{33}$ | | $t_{3R} = k_d t_{26} - k_c t_{2a}$ | | $t_{3S} = t_{3Q} + t_{3R}$
$t_{3T} = t_{3R} - t_{3Q}$ | $t_{3U} = k_d t_{37} - k_c t_{33}$ | | $t_{3V} = k_d t_{2a} + k_c t_{26}$ | | $t_{3W} = t_{3U} + t_{3V}$
$t_{3X} = t_{3V} - t_{3U}$ | $t_{3Y} = t_{3L} + t_{3S}$ | $t_{3Z} = i(t_{3O} + t_{3W})$ | $X_{60} = t_{3Y} - t_{3Z}$ | $X_4 = t_{3Y} + t_{3Z}$ | $t_{40} = i(t_{3P} + t_{3T})$
$t_{41} = t_{3M} + t_{3X}$ | $X_{12} = t_{40} + t_{41}$ | $X_{52} = t_{41} - t_{40}$ | $t_{42} = t_{3L} - t_{3S}$ | $t_{43} = i(t_{3W} - t_{3O})$ | $X_{36} = t_{42} - t_{43}$
$X_{28} = t_{42} + t_{43}$ | $t_{44} = i(t_{3T} - t_{3P})$ | $t_{45} = t_{3M} - t_{3X}$ | $X_{20} = t_{44} + t_{45}$ | $X_{44} = t_{45} - t_{44}$ | $t_{46} = t_8 - t_G$
$t_{47} = t_{17} - t_{1x}$ | $t_{48} = t_{46} + t_{47}$ | $t_{49} = t_{46} - t_{47}$ | $t_{4a} = k_9 t_{3n} - k_8 t_{3j}$ | | $t_{4b} = k_9 t_{2q} + k_8 t_{2m}$
$t_{4c} = t_{4a} + t_{4b}$ | $t_{4d} = t_{4b} - t_{4a}$ | $t_{4e} = k_8 t_{3n} + k_9 t_{3j}$ | | $t_{4f} = k_9 t_{2m} - k_8 t_{2q}$
$t_{4g} = t_{4e} + t_{4f}$ | $t_{4h} = t_{4f} - t_{4e}$ | $t_{4i} = t_H - t_9$ | $t_{4j} = t_{16} - t_{1w}$
$t_{4k} = t_{4i} + t_{4j}$ | $t_{4l} = t_{4j} - t_{4i}$ | $t_{4m} = t_{48} + t_{4g}$ | $t_{4n} = i(t_{4k} + t_{4c})$ | $X_{58} = t_{4m} - t_{4n}$ | $X_6 = t_{4m} + t_{4n}$
$t_{4o} = i(t_{4l} + t_{4h})$ | $t_{4p} = t_{49} + t_{4d}$ | $X_{10} = t_{4o} + t_{4p}$ | $X_{54} = t_{4p} - t_{4o}$ | $t_{4q} = t_{48} - t_{4g}$ | $t_{4r} = i(t_{4c} - t_{4k})$
$X_{38} = t_{4q} - t_{4r}$ | $X_{26} = t_{4q} + t_{4r}$ | $t_{4s} = i(t_{4h} - t_{4l})$ | $t_{4t} = t_{49} - t_{4d}$ | $X_{22} = t_{4s} + t_{4t}$ | $X_{42} = t_{4t} - t_{4s}$
$t_{4u} = t_b - t_u$ | $t_{4v} = t_U - t_{1k}$ | $t_{4w} = t_{4u} + t_{4v}$ | $t_{4x} = t_{4u} - t_{4v}$ | $t_{4y} = k_7 t_{2Y} - k_6 t_{3a}$
$t_{4z} = k_6 t_{21} + k_7 t_{2d}$ | | $t_{4A} = t_{4y} + t_{4z}$ | $t_{4B} = t_{4z} - t_{4y}$ | $t_{4C} = k_7 t_{3a} + k_6 t_{2Y}$
$t_{4D} = k_7 t_{21} - k_6 t_{2d}$ | | $t_{4E} = t_{4C} + t_{4D}$ | $t_{4F} = t_{4D} - t_{4C}$ | $t_{4G} = t_C - t_e$
$t_{4I} = t_{4G} + t_{4H}$ | $t_{4J} = t_{4H} - t_{4G}$ | $t_{4K} = t_{4w} + t_{4E}$ | $t_{4L} = i(t_{4I} + t_{4A})$ | $X_{57} = t_{4K} - t_{4L}$
$t_{4M} = i(t_{4J} + t_{4F})$ | $t_{4N} = t_{4x} + t_{4B}$ | $X_9 = t_{4M} + t_{4N}$ | $X_{55} = t_{4N} - t_{4M}$ | $t_{4O} = t_{4w} - t_{4E}$
$X_{39} = t_{4O} - t_{4P}$ | $X_{25} = t_{4O} + t_{4P}$ | $t_{4Q} = i(t_{4F} - t_{4J})$ | $t_{4R} = t_{4x} - t_{4B}$ | $X_{23} = t_{4Q} + t_{4R}$
$t_{4S} = t_c + t_D$ | $t_{4T} = t_{1s} + t_{12}$ | $t_{4U} = t_{4S} + t_{4T}$ | $t_{4V} = t_{4S} - t_{4T}$ | $t_{4W} = k_5 t_{3q} - k_4 t_{3e}$
$t_{4X} = k_5 t_{2t} + k_4 t_{2h}$ | | $t_{4Y} = t_{4W} + t_{4X}$ | $t_{4Z} = t_{4X} - t_{4W}$ | $t_{50} = k_4 t_{3q} + k_5 t_{3e}$
$t_{51} = k_5 t_{2h} - k_4 t_{2t}$ | | $t_{52} = t_{50} + t_{51}$ | $t_{53} = t_{51} - t_{50}$ | $t_{54} = t_f + t_v$
$t_{56} = t_{54} + t_{55}$ | $t_{57} = t_{55} - t_{54}$ | $t_{58} = t_{4U} + t_{52}$ | $t_{59} = i(t_{56} + t_{4Y})$ | $X_{61} = t_{58} - t_{59}$
$t_{5a} = i(t_{57} + t_{53})$ | $t_{5b} = t_{4V} + t_{4Z}$ | $X_{13} = t_{5a} + t_{5b}$ | $X_{51} = t_{5b} - t_{5a}$ | $t_{5c} = t_{4U} - t_{52}$
$X_{35} = t_{5c} - t_{5d}$ | $X_{29} = t_{5c} + t_{5d}$ | $t_{5e} = i(t_{53} - t_{57})$ | $t_{5f} = t_{4V} - t_{4Z}$ | $X_{19} = t_{5e} + t_{5f}$
$t_{5g} = t_8 + t_G$ | $t_{5h} = t_{1w} + t_{16}$ | $t_{5i} = t_{5g} + t_{5h}$ | $t_{5j} = t_{5g} - t_{5h}$ | $t_{5k} = k_a t_{3m} - k_b t_{3i}$
$t_{5l} = k_a t_{2p} + k_b t_{2l}$ | | $t_{5m} = t_{5k} + t_{5l}$ | $t_{5n} = t_{5l} - t_{5k}$ | $t_{5o} = k_b t_{3m} + k_a t_{3i}$
$t_{5p} = k_a t_{2l} - k_b t_{2p}$ | | $t_{5q} = t_{5o} + t_{5p}$ | $t_{5r} = t_{5p} - t_{5o}$ | $t_{5s} = t_9 + t_H$
$t_{5u} = t_{5s} + t_{5t}$ | $t_{5v} = t_{5t} - t_{5s}$ | $t_{5w} = t_{5i} + t_{5q}$ | $t_{5x} = i(t_{5u} + t_{5m})$ | $X_{62} = t_{5w} - t_{5x}$
$t_{5y} = i(t_{5v} + t_{5r})$ | $t_{5z} = t_{5j} + t_{5n}$ | $X_{14} = t_{5y} + t_{5z}$ | $X_{50} = t_{5z} - t_{5y}$ | $t_{5A} = t_{5i} - t_{5q}$
$X_{34} = t_{5A} - t_{5B}$ | $X_{30} = t_{5A} + t_{5B}$ | $t_{5C} = i(t_{5r} - t_{5v})$ | $t_{5D} = t_{5j} - t_{5n}$ | $X_{18} = t_{5C} + t_{5D}$
$t_{5E} = t_b + t_u$ | $t_{5F} = t_{1l} + t_V$ | $t_{5G} = t_{5E} + t_{5F}$ | $t_{5H} = t_{5E} - t_{5F}$ | $t_{5I} = k_3 t_{2Z} - k_2 t_{3b}$
$t_{5J} = k_2 t_{22} + k_3 t_{2e}$ | | $t_{5K} = t_{5I} + t_{5J}$ | $t_{5L} = t_{5J} - t_{5I}$ | $t_{5M} = k_3 t_{3b} + k_2 t_{2Z}$
$t_{5N} = k_3 t_{22} - k_2 t_{2e}$ | | $t_{5O} = t_{5M} + t_{5N}$ | $t_{5P} = t_{5N} - t_{5M}$ | $t_{5Q} = t_e + t_C$
$t_{5S} = t_{5Q} + t_{5R}$ | $t_{5T} = t_{5R} - t_{5Q}$ | $t_{5U} = t_{5G} + t_{5O}$ | $t_{5V} = i(t_{5S} + t_{5K})$ | $X_{63} = t_{5U} - t_{5V}$
$t_{5W} = i(t_{5T} + t_{5P})$ | $t_{5X} = t_{5H} + t_{5L}$ | $X_{15} = t_{5W} + t_{5X}$ | $X_{49} = t_{5X} - t_{5W}$ | $t_{5Y} = t_{5G} - t_{5O}$
$X_{33} = t_{5Y} - t_{5Z}$ | $X_{31} = t_{5Y} + t_{5Z}$ | $t_{60} = i(t_{5P} - t_{5T})$ | $t_{61} = t_{5H} - t_{5L}$ | $X_{17} = t_{60} + t_{61}$
$t_{62} = t_c - t_D$ | $t_{63} = t_{13} - t_{1t}$ | $t_{64} = t_{62} + t_{63}$ | $t_{65} = t_{62} - t_{63}$ | $t_{66} = k_1 t_{3r} - k_0 t_{3f}$
$t_{67} = k_1 t_{2u} + k_0 t_{2i}$ | | $t_{68} = t_{66} + t_{67}$ | $t_{69} = t_{67} - t_{66}$ | $t_{6a} = k_0 t_{3r} + k_1 t_{3f}$
$t_{6b} = k_1 t_{2i} - k_0 t_{2u}$ | | $t_{6c} = t_{6a} + t_{6b}$ | $t_{6d} = t_{6b} - t_{6a}$ | $t_{6e} = t_v - t_f$
$t_{6g} = t_{6e} + t_{6f}$ | $t_{6h} = t_{6f} - t_{6e}$ | $t_{6i} = t_{64} + t_{6c}$ | $t_{6j} = i(t_{6g} + t_{68})$ | $X_{59} = t_{6i} - t_{6j}$
$t_{6k} = i(t_{6h} + t_{6d})$ | $t_{6l} = t_{65} + t_{69}$ | $X_{11} = t_{6k} + t_{6l}$ | $X_{53} = t_{6l} - t_{6k}$ | $t_{6m} = t_{64} - t_{6c}$
$X_{37} = t_{6m} - t_{6n}$ | $X_{27} = t_{6m} + t_{6n}$ | $t_{6o} = i(t_{6d} - t_{6h})$ | $t_{6p} = t_{65} - t_{69}$ | $X_{21} = t_{6o} + t_{6p}$

$t_{6f} = t_{12} - t_{1s}$
$X_5 = t_{6i} + t_{6j}$
$t_{6n} = i(t_{68} - t_{6g})$
$X_{43} = t_{6p} - t_{6o}$

FFT, $N = 7$

$k_0 = \cos(\pi/7)$ | $k_1 = \cos(3\pi/7)$ | $k_2 = \cos(2\pi/7)$ | $k_3 = \sin(5\pi/7)$ | $k_4 = \sin(3\pi/7)$
$k_5 = \sin(\pi/7)$ | $t_0 = x_3 + x_4$ | $t_1 = x_4 - x_3$ | $t_2 = x_1 + x_6$ | $t_3 = x_6 - x_1$
$t_4 = x_2 + x_5$ | $t_5 = x_5 - x_2$ | $X_0 = x_0 + t_2 + t_4 + t_0$
$t_6 = i(k_4 t_1 - k_3 t_5 + k_5 t_3)$ | $t_7 = x_0 - k_0 t_2 - k_1 t_0 + k_2 t_4$ | $X_4 = t_7 - t_6$
$X_3 = t_7 + t_6$ | $t_8 = i(k_4 t_3 - k_5 t_5 - k_3 t_1)$ | $t_9 = x_0 - k_1 t_2 - k_0 t_4 + k_2 t_0$
$X_5 = t_9 - t_8$ | $X_2 = t_9 + t_8$ | $t_a = i(k_5 t_1 + k_4 t_5 + k_3 t_3)$
$t_b = x_0 - k_1 t_4 - k_0 t_0 + k_2 t_2$ | $X_6 = t_b - t_a$ | $X_1 = t_b + t_a$

FFT, $N = 12$

$k_0 = 1/2$ | $k_1 = \sqrt{3}/2$ | $t_0 = x_4 + x_8$ | $t_1 = x_8 - x_4$
$t_2 = x_{10} + x_2$ | $t_3 = x_2 - x_{10}$ | $t_4 = x_0 + t_0$ | $t_5 = x_6 + t_2$
$t_6 = t_1 + t_3$ | $t_7 = k_1(t_1 - t_3)$ | $t_8 = x_6 - k_0 t_2$ | $t_9 = x_0 - k_0 t_0$
$t_a = x_9$ | $t_b = x_7 + x_{11}$ | $t_c = x_{11} - x_7$ | $t_d = x_1 + x_5$
$t_e = x_5 - x_1$ | $t_f = x_3 + t_b$ | $t_g = t_a + t_d$ | $t_h = t_c + t_e$
$t_i = t_a - k_0 t_d$ | $t_j = x_3 - k_0 t_b$ | $t_k = k_1(t_c - t_e)$ | $t_l = t_4 - t_5$
$t_m = i(t_f - t_g)$ | $X_9 = t_l - t_m$ | $X_3 = t_l + t_m$ | $t_n = t_4 + t_5$
$t_o = t_f + t_g$ | $X_6 = t_n - t_o$ | $X_0 = t_n + t_o$ | $t_p = t_9 - t_8$
$t_q = t_p - t_k$ | $t_r = t_p + t_k$ | $t_s = t_j - t_i$ | $t_t = i(t_7 + t_s)$
$t_u = i(t_7 - t_s)$ | $X_5 = t_q - t_t$ | $X_{11} = t_r - t_u$ | $X_7 = t_t + t_q$
$X_1 = t_u + t_r$ | $t_v = ik_1(t_h - t_6)$ | $t_w = ik_1(t_6 + t_h)$ | $t_x = t_9 + t_8$
$t_y = t_j + t_i$ | $t_z = t_x - t_y$ | $t_A = t_x + t_y$ | $X_{10} = t_z - t_v$
$X_4 = t_A + t_w$ | | $X_2 = t_z + t_v$ | $X_8 = t_A - t_w$

FFT, $N = 15$

$k_0 = \sin(\pi/3)\sin(\pi/5)$ | $k_1 = 1/2$ | $k_2 = \sqrt{3}/2$
$k_3 = \sin(\pi/3)\sin(3\pi/5)$ | $k_4 = 1/4$ | $k_5 = \sqrt{3}/8$
$k_6 = \sin(\pi/5)$ | $k_7 = \sin(2\pi/5)$ | $k_8 = \sqrt{5}/4$ | $k_9 = \sqrt{15}/8$
$t_0 = x_5 + x_{10}$ | $t_1 = x_0 + t_0$ | $t_2 = x_{10} - x_5$ | $t_3 = x_0 - k_1 t_0$
$t_4 = x_8 + x_{13}$ | $t_5 = x_3 - k_1 t_4$ | $t_6 = x_{13} - x_8$ | $t_7 = x_9$
$t_8 = x_{14} + x_4$ | $t_9 = t_7 - k_1 t_8$ | $t_a = x_4 - x_{14}$ | $t_b = x_2 + x_7$
$t_c = x_{12} - k_1 t_b$ | $t_d = x_7 - x_2$ | $t_e = x_{11} + x_1$ | $t_f = x_6 - k_1 t_e$
$t_g = x_1 - x_{11}$ | $t_h = t_g - t_a$ | $t_i = t_f - t_9$ | $t_j = t_5 - t_c$
$t_k = t_6 - t_d$ | $t_l = x_3 + t_4$ | $t_m = x_{12} + t_b$ | $t_n = t_l + t_m$
$t_o = x_6 + t_e$ | $t_p = t_7 + t_8$ | $t_q = t_o + t_p$ | $t_r = t_5 + t_c$
$t_s = t_f + t_9$ | $t_t = t_r + t_s$ | $t_u = t_6 + t_d$ | $t_v = t_g + t_a$
$t_w = k_9(t_u - t_v)$ | $t_x = t_u + t_v$ | $t_y = t_3 + t_t$ | $t_z = ik_2(t_2 + t_x)$
$X_5 = t_y - t_z$ | $X_{10} = t_y + t_z$ | $t_A = k_8(t_n - t_q)$ | $t_B = t_n + t_q$
$t_C = t_1 - k_4 t_B$ | | $t_D = t_o - t_p$ | $t_E = t_l - t_m$
$t_F = i(k_7 t_D - k_6 t_E)$ | | $t_G = i(k_6 t_D + k_7 t_E)$
$X_0 = t_1 + t_B$ | $t_H = t_A + t_C$ | $X_6 = t_H - t_G$ | $X_9 = t_G + t_H$
$t_I = t_C - t_A$ | $X_3 = t_I - t_F$ | $X_{12} = t_F + t_I$ | $t_J = k_3 t_h - k_0 t_k$
$t_K = k_7 t_i - k_6 t_j$ | $t_L = k_6 t_i + k_7 t_j$ | $t_M = k_0 t_h + k_3 t_k$ | $t_N = k_2 t_2 - k_5 t_x$
$t_O = t_w - t_N$ | $t_P = t_w + t_N$ | $t_Q = t_3 - k_4 t_t$ | $t_R = k_8(t_r - t_s)$
$t_S = t_Q - t_R$ | $t_T = t_R + t_Q$ | $t_U = t_S - t_J$ | $t_V = i(t_K - t_O)$
$X_8 = t_U - t_V$ | $X_7 = t_U + t_V$ | $t_W = t_T - t_M$ | $t_X = i(t_L + t_P)$
$X_{11} = t_W - t_X$ | $X_4 = t_W + t_X$ | $t_Y = t_S + t_J$ | $t_Z = i(t_K + t_O)$
$X_{13} = t_Y - t_Z$ | $X_2 = t_Y + t_Z$ | $t_{10} = t_T + t_M$ | $t_{11} = i(t_P - t_L)$
$X_{14} = t_{10} - t_{11}$ | | | $X_1 = t_{10} + t_{11}$

FFT, $N = 10$

$k_0 = 1/4$ | $k_1 = \sqrt{5}/4$ | $k_2 = \sin(\pi/5)$ | $k_3 = \sin(2\pi/5)$ | $t_0 = x_0 - x_5$
$t_1 = x_0 + x_5$ | $t_2 = x_2 - x_7$ | $t_3 = x_2 + x_7$ | $t_4 = x_6 - x_1$ | $t_5 = x_6 + x_1$
$t_6 = x_8 - x_3$ | $t_7 = x_8 + x_3$ | $t_8 = x_9$ | $t_9 = x_4 - t_8$ | $t_a = x_4 + t_8$
$t_b = t_2 - t_6$ | $t_c = t_9 - t_4$ | $t_d = t_3 - t_7$ | $t_e = t_a - t_5$ | $t_f = t_3 + t_7$
$t_g = t_a + t_5$ | $t_h = t_f + t_g$ | $t_i = t_2 + t_6$ | $t_j = t_9 + t_4$ | $t_k = t_i + t_j$
$X_5 = t_0 + t_k$ | $X_0 = t_1 + t_h$ | $t_l = i(k_2 t_c + k_3 t_b)$
$t_m = i(k_3 t_c - k_2 t_b)$ | | $t_n = k_1(t_i - t_j)$ | $t_o = t_0 - k_0 t_k$ | $t_p = t_n + t_o$
$t_q = t_o - t_n$ | $X_1 = t_p - t_l$ | $X_7 = t_m + t_q$ | $X_9 = t_l + t_p$ | $X_3 = t_q - t_m$
$t_r = i(k_3 t_e - k_2 t_d)$ | | $t_s = i(k_2 t_e + k_3 t_d)$ | | $t_t = t_1 - k_0 t_h$
$t_u = k_1(t_f - t_g)$ | $t_v = t_t - t_u$ | $t_w = t_u + t_t$ | $X_2 = t_r + t_v$ | $X_6 = t_w - t_s$
$X_8 = t_v - t_r$ | $X_4 = t_s + t_w$

FFT, $N = 16$

$k_0 = \cos(\pi/8)$ | $k_1 = \sin(\pi/8)$ | $k_2 = \sqrt{2}/2$ | $t_0 = x_4 + x_{12}$ | $t_1 = x_0 + x_8$
$t_2 = x_4 - x_{12}$ | $t_3 = t_1 + t_0$ | $t_4 = x_0 - x_8$ | $t_5 = t_1 - t_0$ | $t_6 = x_{14} - x_6$
$t_7 = x_{14} + x_6$ | $t_8 = x_2 - x_{10}$ | $t_9 = x_2 + x_{10}$ | $t_a = k_2(t_6 - t_8)$ | $t_b = t_9 + t_7$
$t_c = k_2(t_8 + t_6)$ | $t_d = t_7 - t_9$ | $t_e = x_{15} - x_7$ | $t_f = x_{15} + x_7$ | $t_g = x_3 - x_{11}$
$t_h = x_3 + x_{11}$ | $t_i = k_1 t_e - k_0 t_g$ | $t_j = t_f + t_h$ | $t_k = k_1 t_g + k_0 t_e$ | $t_l = t_f - t_h$
$t_m = x_9$ | $t_n = x_1 - t_m$ | $t_o = x_1 + t_m$ | $t_p = x_5 - x_{13}$ | $t_q = x_5 + x_{13}$
$t_r = k_0 t_p + k_1 t_n$ | $t_s = t_o + t_q$ | $t_t = k_0 t_n - k_1 t_p$ | $t_u = t_o - t_q$ | $t_v = t_3 + t_b$
$t_w = t_s + t_j$ | $X_8 = t_v - t_w$ | $X_0 = t_v + t_w$ | $t_x = t_3 - t_b$ | $t_y = i(t_j - t_s)$
$X_{12} = t_x - t_y$ | $X_4 = t_x + t_y$ | $t_z = k_2(t_u + t_l)$ | $t_A = t_5 + t_z$ | $t_B = t_5 - t_z$
$t_C = k_2(t_l - t_u)$ | $t_D = i(t_d + t_C)$ | $t_E = i(t_C - t_d)$ | $X_{14} = t_A - t_D$ | $X_6 = t_B + t_E$
$X_2 = t_A + t_D$ | $X_{10} = t_B - t_E$ | $t_F = t_i - t_r$ | $t_G = t_a - t_2$ | $t_H = i(t_F - t_G)$
$t_I = i(t_G + t_F)$ | $t_J = t_4 + t_c$ | $t_K = t_t + t_k$ | $t_L = t_J - t_K$ | $t_M = t_J + t_K$
$X_7 = t_H + t_L$ | $X_{15} = t_M - t_I$ | $X_9 = t_L - t_H$ | $X_1 = t_I + t_M$ | $t_N = t_4 - t_c$
$t_O = t_r + t_i$ | $t_P = t_N + t_O$ | $t_Q = t_N - t_O$ | $t_R = t_2 + t_a$ | $t_S = t_k - t_t$
$t_T = i(t_R + t_S)$ | $t_U = i(t_S - t_R)$ | $X_{13} = t_P - t_T$ | $X_5 = t_Q + t_U$ | $X_3 = t_P + t_T$
$X_{11} = t_Q - t_U$

FFT, $N = 20$

$k_0 = \sin(\pi/5)$ | $k_1 = \sin(2\pi/5)$ | $k_2 = 1/4$ | $k_3 = \sqrt{5}/4$ | $t_0 = x_0 + x_{10}$
$t_1 = x_5 + x_{15}$ | $t_2 = x_0 - x_{10}$ | $t_3 = t_0 + t_1$ | $t_4 = x_5 - x_{15}$ | $t_5 = t_0 - t_1$
$t_6 = x_4 - x_{14}$ | $t_7 = x_4 + x_{14}$ | $t_8 = x_{13} - x_3$ | $t_9 = x_{13} + x_3$ | $t_a = x_{17} - x_7$
$t_b = x_{17} + x_7$ | $t_c = x_{16} - x_6$ | $t_d = x_{16} + x_6$ | $t_e = x_8 - x_{18}$ | $t_f = x_8 + x_{18}$
$t_g = x_9$ | $t_h = x_{19}$ | $t_i = t_g - t_h$ | $t_j = t_g + t_h$ | $t_k = x_1 - x_{11}$
$t_l = x_1 + x_{11}$ | $t_m = x_{12} - x_2$ | $t_n = x_{12} + x_2$ | $t_o = t_k - t_i$ | $t_p = t_6 - t_c$
$t_q = t_e - t_m$ | $t_r = t_a - t_8$ | $t_s = t_f - t_9$ | $t_t = t_n - t_b$ | $t_u = t_s + t_t$
$t_v = t_f + t_9$ | $t_w = t_n + t_b$ | $t_x = t_v + t_w$ | $t_y = t_7 + t_j$ | $t_z = t_d + t_l$
$t_A = t_y + t_z$ | $t_B = t_7 - t_j$ | $t_C = t_d - t_l$ | $t_D = t_B + t_C$ | $t_E = t_6 + t_c$
$t_F = t_e + t_m$ | $t_G = t_E + t_F$ | $t_H = k_3(t_E - t_F)$ | $t_I = t_i + t_k$ | $t_J = t_8 + t_a$
$t_K = t_I + t_J$ | $t_L = k_3(t_J - t_I)$ | $t_M = t_2 + t_G$ | $t_N = i(t_4 + t_K)$ | $X_5 = t_M - t_N$
$X_{15} = t_M + t_N$ | $t_O = k_3(t_A - t_x)$ | $t_P = t_A + t_x$ | $t_Q = t_3 - k_2 t_P$ | $t_R = t_y - t_z$
$t_S = t_v - t_w$ | $t_T = i(k_0 t_S + k_1 t_R)$ | | $t_U = i(k_1 t_S - k_0 t_R)$
$X_0 = t_3 + t_P$ | $t_V = t_Q - t_O$ | $X_8 = t_V - t_U$ | $X_{12} = t_U + t_V$ | $t_W = t_O + t_Q$
$X_4 = t_T + t_W$ | $X_{16} = t_W - t_T$ | $t_X = k_3(t_D - t_u)$ | $t_Y = t_D + t_u$ | $t_Z = t_5 - k_2 t_Y$
$t_{10} = t_s - t_t$ | $t_{11} = t_B - t_C$ | $t_{12} = i(k_1 t_{10} - k_0 t_{11})$
$t_{13} = i(k_0 t_{10} + k_1 t_{11})$ | | $X_{10} = t_5 + t_Y$ | $t_{14} = t_X + t_Z$ | $X_6 = t_{14} - t_{13}$
$X_{14} = t_{13} + t_{14}$ | $t_{15} = t_Z - t_X$ | $X_2 = t_{12} + t_{15}$ | $X_{18} = t_{15} - t_{12}$ | $t_{16} = k_0 t_r + k_1 t_o$
$t_{17} = k_0 t_q + k_1 t_p$ | $t_{18} = k_1 t_q - k_0 t_p$ | $t_{19} = k_1 t_r - k_0 t_o$ | $t_{1a} = k_2 t_K - t_4$ | $t_{1b} = t_{1a} + t_L$
$t_{1c} = t_L - t_{1a}$ | $t_{1d} = t_2 - k_2 t_G$ | $t_{1e} = t_H + t_{1d}$ | $t_{1f} = t_{1d} - t_H$ | $t_{1g} = t_{1e} + t_{16}$
$t_{1h} = i(t_{1b} - t_{17})$ | $X_{19} = t_{1g} - t_{1h}$ | $X_1 = t_{1g} + t_{1h}$ | $t_{1i} = t_{1f} + t_{19}$ | $t_{1j} = i(t_{18} + t_{1c})$
$X_{13} = t_{1i} - t_{1j}$ | $X_7 = t_{1i} + t_{1j}$ | $t_{1k} = t_{1e} - t_{16}$ | $t_{1l} = i(t_{17} + t_{1b})$ | $X_{11} = t_{1k} - t_{1l}$
$X_9 = t_{1k} + t_{1l}$ | $t_{1m} = t_{1f} - t_{19}$ | $t_{1n} = i(t_{1c} - t_{18})$ | $X_{17} = t_{1m} - t_{1n}$ | $X_3 = t_{1m} + t_{1n}$

FFT, $N = 11$

$$
\begin{array}{llll}
k_0 = \cos(\pi/11) & k_1 = \cos(3\pi/11) & k_2 = \cos(4\pi/11) & k_3 = \cos(5\pi/11) \quad | \quad k_4 = \cos(2\pi/11) \\
k_5 = \sin(\pi/11) & k_6 = \sin(3\pi/11) & k_7 = \sin(4\pi/11) & k_8 = \sin(5\pi/11) \quad | \quad k_9 = \sin(9\pi/11) \\
t_0 = x_1 + x_{10} & t_1 = x_{10} - x_1 & t_2 = x_5 + x_6 & t_3 = x_6 - x_5 \quad | \quad t_4 = x_4 + x_7 \\
t_5 = x_7 - x_4 & t_6 = x_3 + x_8 & t_7 = x_8 - x_3 & t_8 = x_9 \quad | \quad t_9 = x_2 + t_8 \\
t_a = t_8 - x_2 & X_0 = x_0 + t_0 + t_9 + t_6 + t_4 + t_2 & & \\
t_b = i(k_5 t_5 - k_8 t_a - k_7 t_3 + k_9 t_7 + k_6 t_1) & & & \\
t_c = x_0 - k_1 t_0 - k_3 t_9 - k_0 t_4 + k_2 t_2 + k_4 t_6 & & & | \quad X_7 = t_c - t_b \quad | \quad X_4 = t_c + t_b \\
t_d = i(k_8 t_3 - k_9 t_a - k_7 t_5 + k_6 t_7 + k_5 t_1) & & & \\
t_e = x_0 - k_0 t_0 - k_1 t_6 - k_3 t_2 + k_2 t_4 + k_4 t_9 & & & | \quad X_6 = t_e - t_d \quad | \quad X_5 = t_e + t_d \\
t_f = i(k_6 t_3 - k_5 t_a - k_7 t_7 + k_9 t_5 + k_8 t_1) & & & \\
t_g = x_0 - k_3 t_0 - k_0 t_9 - k_1 t_2 + k_4 t_4 + k_2 t_6 & & & | \quad X_8 = t_g - t_f \quad | \quad X_3 = t_g + t_f \\
t_h = i(k_5 t_3 + k_6 t_5 + k_8 t_7 + k_7 t_a + k_9 t_1) & & & \\
t_i = x_0 - k_3 t_6 - k_1 t_4 - k_0 t_2 + k_2 t_9 + k_4 t_0 & & & | \quad X_{10} = t_i - t_h \quad | \quad X_1 = t_i + t_h \\
t_j = i(k_6 t_a - k_5 t_7 - k_8 t_5 - k_9 t_3 + k_7 t_1) & & & \\
t_k = x_0 - k_1 t_9 - k_0 t_6 - k_3 t_4 + k_4 t_2 + k_2 t_0 & & & | \quad X_9 = t_k - t_j \quad | \quad X_2 = t_k + t_j
\end{array}
$$

FFT, $N = 13$

$$
\begin{array}{llll}
k_0 = \frac{4}{3}(\sin(\pi/13)\sin(9\pi/13)\sin(10\pi/13)) & k_1 = 2 & k_2 = 1/2 \\
k_3 = \frac{4}{3}(\sin(2\pi/13)\sin(6\pi/13)\sin(8\pi/13)) & k_4 = \sqrt{3} & k_5 = 1/12 \\
k_6 = \sin(2\pi/13)\sin(3\pi/13)\sin(6\pi/13)\sin(9\pi/13) & & k_7 = \sqrt{3}/2 \\
k_8 = 2\sin(2\pi/13)\sin(3\pi/13)\sin(6\pi/13)\sin(9\pi/13) & & k_9 = \sqrt{13}/12 \\
k_a = \frac{8}{3}(\cos(\pi/13)\sin(4\pi/13)\cos(5\pi/13)\sin(6\pi/13)\cos(8\pi/13)\cos(12\pi/13)) & & \\
k_b = 4\cos(\pi/13)\sin(4\pi/13)\cos(5\pi/13)\sin(6\pi/13)\cos(8\pi/13)\cos(12\pi/13) & & \\
k_c = 0{,}075902986037193864721217551050 & k_d = 0{,}132983124607418651264012510183 & \\
k_e = 0{,}300238635966332656490607178057 & k_f = 0{,}011599105605768289875556753543 & \\
k_g = 0{,}156891391051584616622704970723 & k_h = 0{,}256247671582936598078106271714 & \\
k_i = 0{,}113854479055790797081826326576 & k_j = 0{,}265966249214837302528025020365 &
\end{array}
$$

$$
\begin{array}{llll}
t_0 = x_8 - x_5 & t_1 = x_8 + x_5 & t_2 = x_{10} + x_4 & t_3 = x_{12} + t_2 \\
t_4 = x_{10} - x_4 & t_5 = x_{12} - k_2 t_2 & t_6 = x_9 & t_7 = x_3 + t_6 \\
t_8 = x_1 + t_7 & t_9 = x_3 - t_6 & t_a = x_1 - k_2 t_7 & t_b = x_{11} - x_6 \\
t_c = x_{11} + x_6 & t_d = x_7 - x_2 & t_e = x_7 + x_2 & t_f = t_b + t_d \\
t_g = t_c + t_e & t_h = t_0 + t_f & t_i = t_8 - t_3 & t_j = t_a - t_5 \\
t_k = k_7(t_c - t_e) & t_l = t_j + t_k & t_m = t_j - t_k & t_n = t_8 + t_3 \\
t_o = t_1 + t_g & t_p = k_9(t_n - t_o) & t_q = t_n + t_o & t_r = t_a + t_5 \\
t_s = t_1 - k_2 t_g & t_t = t_r - t_s & t_u = t_r + t_s & t_v = k_7(t_9 - t_4) \\
t_w = t_0 - k_2 t_f & t_x = t_v - t_w & t_y = t_v + t_w & t_z = t_9 + t_4 \\
t_A = t_b - t_d & t_B = t_z - t_A & t_C = t_z + t_A & X_0 = x_0 + t_q \\
t_D = k_j t_t + k_b t_B & t_E = k_i t_C - k_8 t_u & t_F = t_D - t_E & t_G = t_D + t_E \\
t_H = k_0 t_h + k_3 t_i & t_I = k_h t_m - k_g t_y & t_J = k_e t_l + k_f t_x & t_K = t_I - t_J \\
t_L = t_H + t_K & t_M = k_4(t_I + t_J) & t_N = k_3 t_h - k_0 t_i & t_O = k_f t_l - k_e t_x \\
t_P = k_g t_m + k_h t_y & t_Q = t_O - t_P & t_R = k_4(t_P + t_O) & t_S = t_N - t_Q \\
t_T = k_a t_t - k_d t_B & t_U = t_p - t_T & t_V = k_c t_u + k_6 t_C & t_W = x_0 - k_5 t_q \\
t_X = t_W - t_V & t_Y = t_p + k_1 t_T & t_Z = t_U + t_X & t_{10} = t_W + k_1 t_V \\
t_{11} = t_X - t_U & t_{12} = i(t_N + k_1 t_Q) & t_{13} = t_Y + t_{10} & X_1 = t_{12} + t_{13} \\
X_{12} = t_{13} - t_{12} & t_{14} = i(k_1 t_K - t_H) & t_{15} = t_{10} - t_Y & X_5 = t_{14} + t_{15} \\
X_8 = t_{15} - t_{14} & t_{16} = t_G + t_Z & t_{17} = i(t_S + t_M) & X_4 = t_{16} - t_{17} \\
X_9 = t_{17} + t_{16} & t_{18} = i(t_S - t_M) & t_{19} = t_Z - t_G & X_3 = t_{18} + t_{19} \\
X_{10} = t_{19} - t_{18} & t_{1a} = i(t_R - t_L) & t_{1b} = t_{11} - t_F & X_6 = t_{1a} + t_{1b} \\
X_7 = t_{1b} - t_{1a} & t_{1c} = t_F + t_{11} & t_{1d} = i(t_R + t_L) & X_2 = t_{1c} - t_{1d} \\
X_{11} = t_{1d} + t_{1c} & & &
\end{array}
$$

[27]

[27] Kaikissa tämän liitteen listauksissa esiintyvät reaalivakiot yritettiin muuntaa yksinkertaisimmiksi mahdollisiksi laskentakaavoiksi, jotka tuottavat samat tulokset, mutta tässä nimenomaisessa tapauksessa vakioille $k_c \ldots k_j$ ei onnistuttu löytämään yhtään ainoata realistista laskentakaavaa, joka tuottaisi saman tuloksen, huolimatta siitä, että tavoitteeseen kulutettiin satoja tietokonelaskentatunteja. Kaavat olisi ehkä mahdollista määrittää muokkaamalla FFTW-kirjaston skriptiä `gen_notw_c.ml`, joka kyseiset algoritmit on generoinut, sekä sen riippuvuuksia siten, että se tallentaisi kertoimien yhteydessä metadataa kertoimien laskentakaavasta, mutta tähän urakkaan ei ryhdytty.

FFT, $N = 14$

$$
\begin{array}{llllllllll}
k_0 &= \cos(\pi/7) & k_1 &= \cos(3\pi/7) & k_2 &= \cos(2\pi/7) & k_3 &= \sin(5\pi/7) & k_4 &= \sin(3\pi/7) \\
k_5 &= \sin(\pi/7) & t_0 &= x_0 - x_7 & t_1 &= x_0 + x_7 & t_2 &= x_6 - x_{13} & t_3 &= x_6 + x_{13} \\
t_4 &= x_8 - x_1 & t_5 &= x_8 + x_1 & t_6 &= t_2 + t_4 & t_7 &= t_3 - t_5 & t_8 &= t_4 - t_2 \\
t_9 &= t_3 + t_5 & t_a &= x_9 & t_b &= x_2 - t_a & t_c &= x_2 + t_a & t_d &= x_{12} - x_5 \\
t_e &= x_{12} + x_5 & t_f &= t_b + t_d & t_g &= t_e - t_c & t_h &= t_d - t_b & t_i &= t_c + t_e \\
t_j &= x_4 - x_{11} & t_k &= x_4 + x_{11} & t_l &= x_{10} - x_3 & t_m &= x_{10} + x_3 & t_n &= t_j + t_l \\
t_o &= t_k - t_m & t_p &= t_l - t_j & t_q &= t_k + t_m & X_7 &= t_0 + t_f + t_n + t_6 & & \\
X_0 &= t_1 + t_i + t_q + t_9 & & & & & t_r &= i(k_4 t_h - k_5 t_p - k_3 t_8) & & \\
t_s &= t_0 - k_1 t_f - k_0 t_n + k_2 t_6 & & & & & X_5 &= t_s - t_r & X_9 &= t_s + t_r \\
t_t &= i(k_3 t_7 + k_5 t_o + k_4 t_g) & t_u &= t_1 - k_1 t_i - k_0 t_q + k_2 t_9 & & & & & X_2 &= t_t + t_u \\
X_{12} &= t_u - t_t & t_v &= i(k_5 t_8 + k_4 t_p + k_3 t_h) & t_w &= t_0 - k_1 t_n - k_0 t_6 + k_2 t_f & & & & \\
X_{13} &= t_w - t_v & X_1 &= t_w + t_v & t_x &= i(k_3 t_g - k_4 t_o - k_5 t_7) & & & & \\
t_y &= t_1 - k_1 t_q - k_0 t_9 + k_2 t_i & & & & & X_6 &= t_y - t_x & X_8 &= t_x + t_y \\
t_z &= i(k_3 t_o - k_4 t_7 + k_5 t_g) & t_A &= t_1 - k_0 t_i - k_1 t_9 + k_2 t_q & & & & & X_4 &= t_A - t_z \\
X_{10} &= t_z + t_A & t_B &= i(k_4 t_8 - k_3 t_p + k_5 t_h) & t_C &= t_0 - k_0 t_f - k_1 t_6 + k_2 t_n & & & & \\
X_{11} &= t_C - t_B & X_3 &= t_C + t_B & & & & & &
\end{array}
$$

FFT, $N = 25$

$$
\begin{array}{llllllllll}
k_0 &= \cos(\pi/5) & k_1 &= \cos(2\pi/5) & k_2 &= \cos(\pi/25) & k_3 &= \cos(2\pi/25) & k_4 &= \cos(4\pi/25) \\
k_5 &= \sin(\pi/5) & k_6 &= \sin(2\pi/5) & k_7 &= \sin(\pi/25) & k_8 &= \sin(2\pi/25) & k_9 &= \sin(4\pi/25) \\
k_a &= \cos(6\pi/25) & k_b &= \cos(7\pi/25) & k_c &= \cos(8\pi/25) & k_d &= \cos(9\pi/25) & k_e &= \cos(12\pi/25) \\
k_f &= \sin(6\pi/25) & k_g &= \sin(7\pi/25) & k_h &= \sin(8\pi/25) & k_i &= \sin(9\pi/25) & k_j &= \sin(12\pi/25) \\
k_k &= 2\cos(\pi/25) & k_l &= 2\cos(2\pi/25) & k_m &= 2\cos(4\pi/25) & k_n &= 2\cos(6\pi/25) & k_o &= 2\cos(7\pi/25) \\
k_p &= 2\sin(\pi/25) & k_q &= 2\sin(2\pi/25) & k_r &= 2\sin(4\pi/25) & k_s &= 2\sin(6\pi/25) & k_t &= 2\sin(7\pi/25) \\
k_u &= 2\cos(8\pi/25) & k_v &= 2\cos(9\pi/25) & k_w &= 2\cos(12\pi/25) & k_x &= \cos(3\pi/5)\sin(7\pi/5) & & \\
k_y &= 2\sin(8\pi/25) & k_z &= 2\sin(9\pi/25) & k_A &= 2\sin(12\pi/25) & k_B &= \sin(\pi/5)\cos(9\pi/5) & & \\
k_C &= 1/4 & k_D &= \sqrt{5}/4 & t_0 &= x_{10} + x_{15} & t_1 &= x_{10} - x_{15} & t_2 &= x_5 + x_{20} \\
t_3 &= x_5 - x_{20} & t_4 &= k_D(t_2 - t_0) & t_5 &= k_5 t_1 + k_6 t_3 & t_6 &= k_6 t_1 & t_7 &= t_6 - k_5 t_3 \\
t_8 &= t_2 + t_0 & t_9 &= x_0 - k_C t_8 & t_a &= x_0 + t_8 & t_b &= x_7 + x_{22} & t_c &= x_{12} + x_{17} \\
t_d &= k_D(t_b - t_c) & t_e &= x_{12} - x_{17} & t_f &= t_b + t_c & t_g &= x_7 - x_{22} & t_h &= x_8 + x_{23} \\
t_i &= x_{13} + x_{18} & t_j &= k_D(t_h - t_i) & t_k &= x_{13} - x_{18} & t_l &= t_h + t_i & t_m &= x_8 - x_{23} \\
t_n &= x_2 + t_f & t_o &= x_3 + t_l & t_p &= t_n + t_o & t_q &= k_x t_k + k_B t_m & t_r &= k_B t_k - k_x t_m \\
t_s &= x_3 - k_C t_l & t_t &= t_j + t_s & t_u &= t_s - t_j & t_v &= k_a t_t - k_s t_q & t_w &= k_e t_u + k_A t_r \\
t_x &= k_f t_t + k_n t_q & t_y &= k_w t_r - k_j t_u & t_z &= k_x t_e + k_B t_g & t_A &= k_B t_e - k_x t_g & t_B &= x_2 - k_C t_f \\
t_C &= t_d + t_B & t_D &= t_B - t_d & t_E &= k_4 t_C - k_r t_z & t_F &= k_c t_D + k_y t_A & t_G &= k_9 t_C + k_m t_z \\
t_H &= k_u t_A - k_h t_D & t_I &= x_6 + x_{21} & t_J &= x_{11} + x_{16} & t_K &= k_D(t_I - t_J) & t_L &= x_{11} - x_{16} \\
t_M &= t_I + t_J & t_N &= x_6 - x_{21} & t_O &= x_9 & t_P &= t_O + x_{24} & t_Q &= x_{19} \\
t_R &= x_{14} + t_Q & t_S &= k_D(t_P - t_R) & t_T &= x_{14} - t_Q & t_U &= t_P + t_R & t_V &= t_O - x_{24} \\
t_W &= x_1 + t_M & t_X &= x_4 + t_U & t_Y &= t_W + t_X & t_Z &= k_x t_T + k_B t_V & t_{10} &= k_B t_T - k_x t_V \\
t_{11} &= x_4 - k_C t_U & & & t_{12} &= t_S + t_{11} & t_{13} &= t_{11} - t_S & t_{14} &= k_c t_{12} - k_y t_Z \\
t_{15} &= k_z t_{10} - k_d t_{13} & & & t_{16} &= k_h t_{12} + k_u t_Z & t_{17} &= k_i t_{13} + k_v t_{10} & & \\
t_{18} &= k_x t_L + k_B t_N & t_{19} &= k_B t_L - k_x t_N & t_{1a} &= x_1 - k_C t_M & t_{1b} &= t_K + t_{1a} & t_{1c} &= t_{1a} - t_K \\
t_{1d} &= k_3 t_{1b} - k_q t_{18} & & & & & t_{1e} &= k_4 t_{1c} + k_r t_{19} & & \\
t_{1f} &= k_8 t_{1b} + k_l t_{18} & & & t_{1g} &= k_m t_{19} - k_9 t_{1c} & & & t_{1h} &= k_D(t_Y - t_p) \\
t_{1i} &= t_Y + t_p & & & t_{1j} &= t_a - k_C t_{1i} & t_{1k} &= t_W - t_X & t_{1l} &= t_n - t_o \\
t_{1m} &= i(k_5 t_{1l} + k_6 t_{1k}) & & & t_{1n} &= i(k_6 t_{1l} - k_5 t_{1k}) & & & X_0 &= t_a + t_{1i} \\
t_{1o} &= t_{1j} - t_{1h} & X_{10} &= t_{1n} + t_{1o} & X_{15} &= t_{1o} - t_{1n} & t_{1p} &= t_{1h} + t_{1j} & X_5 &= t_{1p} - t_{1m} \\
X_{20} &= t_{1m} + t_{1p} & t_{1q} &= t_{15} - t_{1e} & t_{1r} &= t_{1g} + t_{17} & t_{1s} &= t_H - t_y & t_{1t} &= t_F - t_w \\
t_{1u} &= k_k t_{10} - k_7 t_{13} & & & t_{1v} &= k_f t_{1c} + k_n t_{19} & & & t_{1w} &= t_{1u} - t_{1v} \\
t_{1x} &= k_e t_D - k_A t_A & t_{1y} &= k_b t_u + k_t t_r & t_{1z} &= t_{1x} - t_{1y} & t_{1A} &= k_a t_{1c} - k_s t_{19} & & \\
t_{1B} &= k_2 t_{13} + k_p t_{10} & & & t_{1C} &= t_{1A} - t_{1B} & t_{1D} &= k_o t_r - k_g t_u & t_{1E} &= k_j t_D + k_w t_A \\
t_{1F} &= t_{1D} - t_{1E} & & & t_{1G} &= t_{1g} - t_{17} & t_{1H} &= t_H + t_y & t_{1I} &= t_{1G} + t_{1H} \\
t_{1J} &= k_D(t_{1G} - t_{1H}) & & & t_{1K} &= t_7 - k_C t_{1I} & t_{1L} &= t_9 - t_4 & t_{1M} &= t_{1e} + t_{15} \\
t_{1N} &= t_F + t_w & t_{1O} &= t_{1M} + t_{1N} & t_{1P} &= k_D(t_{1M} - t_{1N}) & & & t_{1Q} &= t_{1L} - k_C t_{1O} \\
t_{1R} &= i(t_7 + t_{1I}) & & & t_{1S} &= t_{1L} + t_{1O} & X_2 &= t_{1R} + t_{1S} & X_{23} &= t_{1S} - t_{1R} \\
t_{1T} &= i(t_{1w} + t_{1F} - t_7) & & & t_{1U} &= t_{1L} + t_{1C} + t_{1z} & & & X_3 &= t_{1T} + t_{1U} \\
X_{22} &= t_{1U} - t_{1T} & t_{1V} &= i(t_{1J} + t_{1K} + k_6 t_{1q} - k_5 t_{1t}) & t_{1W} &= t_{1P} + t_{1Q} + k_5 t_{1s} + k_6 t_{1r} & & & & \\
X_7 &= t_{1V} + t_{1W} & X_{18} &= t_{1W} - t_{1V} & t_{1X} &= i(t_{1K} + k_6 t_{1t} + k_5 t_{1q} - t_{1J}) & & & & \\
t_{1Y} &= t_{1Q} - k_6 t_{1s} - t_{1P} + k_5 t_{1r} & & & & & X_{12} &= t_{1X} + t_{1Y} & X_{13} &= t_{1Y} - t_{1X} \\
t_{1Z} &= t_{1L} - k_6(t_{1v} + t_{1u}) - k_5(t_{1E} + t_{1D}) - k_0 t_{1z} + k_1 t_{1C} & & & & & & & & \\
t_{20} &= i(k_1 t_{1w} - k_6(t_{1A} + t_{1B}) - k_0 t_{1F} - k_5(t_{1x} + t_{1y}) - t_7) & & & & & & & X_{17} &= t_{1Z} - t_{20} \\
X_8 &= t_{1Z} + t_{20} & t_{21} &= t_{14} - t_{1d} & t_{22} &= t_{16} - t_{1f} & t_{23} &= t_x - t_G & t_{24} &= t_E - t_v \\
t_{25} &= k_c t_{1b} + k_y t_{18} & & & t_{26} &= k_b t_{12} + k_t t_Z & t_{27} &= t_{25} - t_{26} & t_{28} &= k_i t_C + k_v t_z \\
t_{29} &= k_7 t_t + k_k t_q & t_{2a} &= t_{28} + t_{29} & t_{2b} &= k_u t_{18} - k_h t_{1b} & & & t_{2c} &= k_g t_{12} - k_o t_Z \\
t_{2d} &= t_{2b} + t_{2c} & t_{2e} &= k_z t_z - k_d t_C & t_{2f} &= k_p t_q - k_2 t_t & t_{2g} &= t_{2e} + t_{2f} & t_{2h} &= t_{1f} + t_{16} \\
t_{2i} &= t_G + t_x & t_{2j} &= t_{2h} + t_{2i} & t_{2k} &= k_C t_{2j} - t_5 & t_{2l} &= k_D(t_{2i} - t_{2h}) & & \\
t_{2m} &= t_4 + t_9 & t_{2n} &= t_{1d} + t_{14} & t_{2o} &= t_E + t_v & t_{2p} &= t_{2n} + t_{2o} & t_{2q} &= t_{2m} - k_C t_{2p} \\
t_{2r} &= k_D(t_{2n} - t_{2o}) & & & t_{2s} &= t_{2m} + t_{2p} & t_{2t} &= i(t_5 + t_{2j}) & X_1 &= t_{2s} - t_{2t} \\
X_{24} &= t_{2s} + t_{2t} & t_{2u} &= t_{2m} + t_{27} + t_{2g} & & & t_{2v} &= i(t_5 + t_{2d} - t_{2a}) & & \\
X_{21} &= t_{2u} - t_{2v} & X_4 &= t_{2u} + t_{2v} & t_{2w} &= i(k_6 t_{24} + k_5 t_{21} + t_{2k} - t_{2l}) & & & & \\
t_{2x} &= t_{2q} - t_{2r} - k_6 t_{23} + k_5 t_{22} & & & & & X_{11} &= t_{2w} + t_{2x} & X_{14} &= t_{2x} - t_{2w} \\
t_{2y} &= i(k_6 t_{21} - k_5 t_{24} + t_{2k} + t_{2l}) & t_{2z} &= t_{2r} + t_{2q} + k_5 t_{23} + k_6 t_{22} & & & & & X_6 &= t_{2y} + t_{2z} \\
X_{19} &= t_{2z} - t_{2y} & t_{2A} &= i(t_5 + k_0 t_{2a} - k_6(t_{25} + t_{26}) + k_5(t_{2f} - t_{2e}) + k_1 t_{2d}) & & & & & & \\
t_{2B} &= t_{2m} - k_0 t_{2g} + k_5(t_{29} - t_{28}) + k_6(t_{2b} - t_{2c}) + k_1 t_{27} & & & & & X_9 &= t_{2A} + t_{2B} & X_{16} &= t_{2B} - t_{2A}
\end{array}
$$

FFT, $N = 32$

$k_0 = \cos(\pi/8)$ | $k_1 = \cos(\pi/16)$ | $k_2 = \cos(3\pi/16)$ | $k_3 = \sqrt{2}/2$

$k_4 = \sin(\pi/8)$ | $k_5 = \sin(\pi/16)$ | $k_6 = \sin(3\pi/16)$ | $t_0 = x_0 - x_{16}$

$t_1 = x_0 + x_{16}$ | $t_2 = x_8 - x_{24}$ | $t_3 = x_8 + x_{24}$ | $t_4 = x_4 - x_{20}$

$t_5 = x_4 + x_{20}$ | $t_6 = x_{28} - x_{12}$ | $t_7 = x_{28} + x_{12}$ | $t_8 = t_1 + t_3$

$t_9 = t_5 + t_7$ | $t_a = t_7 - t_5$ | $t_b = t_1 - t_3$ | $t_c = k_3(t_6 - t_4)$

$t_d = t_c - t_2$ | $t_e = t_2 + t_c$ | $t_f = k_3(t_4 + t_6)$ | $t_g = t_0 + t_f$

$t_h = t_0 - t_f$ | $t_i = x_{31} - x_{15}$ | $t_j = x_{31} + x_{15}$ | $t_k = x_7 - x_{23}$

$t_l = x_7 + x_{23}$ | $t_m = x_{19}$ | $t_n = x_3 - t_m$ | $t_o = x_3 + t_m$

$t_p = x_{27} - x_{11}$ | $t_q = x_{27} + x_{11}$ | $t_r = k_3(t_n + t_p)$ | $t_s = t_i + t_r$

$t_t = t_i - t_r$ | $t_u = k_3(t_p - t_n)$ | $t_v = t_u - t_k$ | $t_w = t_k + t_u$

$t_x = t_j + t_l$ | $t_y = t_o + t_q$ | $t_z = t_x - t_y$ | $t_A = t_j - t_l$

$t_B = t_q - t_o$ | $t_C = k_0 t_A - k_4 t_B$ | $t_D = k_0 t_B + k_4 t_A$ | $t_E = x_1 - x_{17}$

$t_F = x_1 + x_{17}$ | $t_G = x_9$ | $t_H = t_G - x_{25}$ | $t_I = t_G + x_{25}$

$t_J = x_5 - x_{21}$ | $t_K = x_5 + x_{21}$ | $t_L = x_{29}$ | $t_M = t_L - x_{13}$

$t_N = t_L + x_{13}$ | $t_O = k_3(t_J + t_M)$ | $t_P = t_E + t_O$ | $t_Q = t_E - t_O$

$t_R = k_3(t_M - t_J)$ | $t_S = t_R - t_H$ | $t_T = t_H + t_R$ | $t_U = t_F + t_I$

$t_V = t_K + t_N$ | $t_W = t_U - t_V$ | $t_X = t_F - t_I$ | $t_Y = t_N - t_K$

$t_Z = k_4 t_Y + k_0 t_X$ | $t_{10} = k_0 t_Y - k_4 t_X$ | $t_{11} = x_2 - x_{18}$ | $t_{12} = x_2 + x_{18}$

$t_{13} = x_6 - x_{22}$ | $t_{14} = x_6 + x_{22}$ | $t_{15} = x_{10} - x_{26}$ | $t_{16} = x_{10} + x_{26}$

$t_{17} = x_{30} - x_{14}$ | $t_{18} = x_{30} + x_{14}$ | $t_{19} = t_{18} + t_{14}$ | $t_{1a} = t_{12} + t_{16}$

$t_{1b} = k_0 t_{11} - k_4 t_{15}$ | | $t_{1c} = k_4 t_{13} + k_0 t_{17}$ |

$t_{1d} = t_{1b} + t_{1c}$ | $t_{1e} = t_{1c} - t_{1b}$ | $t_{1f} = k_4 t_{17} - k_0 t_{13}$ |

$t_{1g} = k_0 t_{15} + k_4 t_{11}$ | | $t_{1h} = t_{1f} - t_{1g}$ | $t_{1i} = t_{1g} + t_{1f}$

$t_{1j} = t_{12} - t_{16}$ | $t_{1k} = t_{18} - t_{14}$ | $t_{1l} = k_3(t_{1j} + t_{1k})$ |

$t_{1m} = k_3(t_{1k} - t_{1j})$ | | $t_{1n} = t_8 - t_9$ | $t_{1o} = k_3(t_W + t_z)$

$t_{1p} = t_{1n} + t_{1o}$ | $t_{1q} = t_{1n} - t_{1o}$ | $t_{1r} = t_{19} - t_{1a}$ | $t_{1s} = k_3(t_z - t_W)$

$t_{1t} = i(t_{1r} + t_{1s})$ | $t_{1u} = i(t_{1s} - t_{1r})$ | $X_{28} = t_{1p} - t_{1t}$ | $X_{12} = t_{1q} + t_{1u}$

$X_4 = t_{1p} + t_{1t}$ | $X_{20} = t_{1q} - t_{1u}$ | $t_{1v} = t_8 + t_9$ | $t_{1w} = t_{1a} + t_{19}$

$t_{1x} = t_{1v} + t_{1w}$ | $t_{1y} = t_{1v} - t_{1w}$ | $t_{1z} = t_U + t_V$ | $t_{1A} = t_x + t_y$

$t_{1B} = t_{1z} + t_{1A}$ | $t_{1C} = i(t_{1A} - t_{1z})$ | $X_{16} = t_{1x} - t_{1B}$ | $X_8 = t_{1y} + t_{1C}$

$X_0 = t_{1x} + t_{1B}$ | $X_{24} = t_{1y} - t_{1C}$ | $t_{1D} = t_C - t_Z$ | $t_{1E} = t_{1m} - t_a$

$t_{1F} = i(t_{1D} - t_{1E})$ | $t_{1G} = i(t_{1E} + t_{1D})$ | $t_{1H} = t_b - t_{1l}$ | $t_{1I} = t_D - t_{10}$

$t_{1J} = t_{1H} - t_{1I}$ | $t_{1K} = t_{1H} + t_{1I}$ | $X_{10} = t_{1F} + t_{1J}$ | $X_{26} = t_{1K} - t_{1G}$

$X_{22} = t_{1J} - t_{1F}$ | $X_6 = t_{1G} + t_{1K}$ | $t_{1L} = t_b + t_{1l}$ | $t_{1M} = t_Z + t_C$

$t_{1N} = t_{1L} + t_{1M}$ | $t_{1O} = t_{1L} - t_{1M}$ | $t_{1P} = t_a + t_{1m}$ | $t_{1Q} = t_{10} + t_D$

$t_{1R} = i(t_{1P} + t_{1Q})$ | $t_{1S} = i(t_{1Q} - t_{1P})$ | $X_{30} = t_{1N} - t_{1R}$ | $X_{14} = t_{1O} + t_{1S}$

$X_2 = t_{1N} + t_{1R}$ | $X_{18} = t_{1O} - t_{1S}$ | $t_{1T} = t_h + t_{1i}$ | $t_{1U} = t_e + t_{1e}$

$t_{1V} = t_h - t_{1i}$ | $t_{1W} = t_{1e} - t_e$ | $t_{1X} = k_2 t_T - k_6 t_Q$ | $t_{1Y} = k_2 t_w + k_6 t_t$

$t_{1Z} = t_{1X} + t_{1Y}$ | $t_{20} = t_{1Y} - t_{1X}$ | $t_{21} = k_6 t_T + k_2 t_Q$ | $t_{22} = k_2 t_t - k_6 t_w$

$t_{23} = t_{21} + t_{22}$ | $t_{24} = t_{22} - t_{21}$ | $t_{25} = t_{1T} + t_{23}$ | $t_{26} = i(t_{1U} + t_{1Z})$

$X_{29} = t_{25} - t_{26}$ | $X_3 = t_{25} + t_{26}$ | $t_{27} = i(t_{1W} + t_{24})$ | $t_{28} = t_{1V} + t_{20}$

$X_5 = t_{27} + t_{28}$ | $X_{27} = t_{28} - t_{27}$ | $t_{29} = t_{1T} - t_{23}$ | $t_{2a} = i(t_{1Z} - t_{1U})$

$X_{19} = t_{29} - t_{2a}$ | $X_{13} = t_{29} + t_{2a}$ | $t_{2b} = i(t_{24} - t_{1W})$ | $t_{2c} = t_{1V} - t_{20}$

$X_{11} = t_{2b} + t_{2c}$ | $X_{21} = t_{2c} - t_{2b}$ | $t_{2d} = t_g + t_{1d}$ | $t_{2e} = t_d + t_{1h}$

$t_{2f} = t_g - t_{1d}$ | $t_{2g} = t_{1h} - t_d$ | $t_{2h} = k_1 t_S - k_5 t_P$ | $t_{2i} = k_1 t_v + k_5 t_s$

$t_{2j} = t_{2h} + t_{2i}$ | $t_{2k} = t_{2i} - t_{2h}$ | $t_{2l} = k_5 t_S + k_1 t_P$ | $t_{2m} = k_1 t_s - k_5 t_v$

$t_{2n} = t_{2l} + t_{2m}$ | $t_{2o} = t_{2m} - t_{2l}$ | $t_{2p} = t_{2d} + t_{2n}$ | $t_{2q} = i(t_{2e} + t_{2j})$

$X_{31} = t_{2p} - t_{2q}$ | $X_1 = t_{2p} + t_{2q}$ | $t_{2r} = i(t_{2g} + t_{2o})$ | $t_{2s} = t_{2f} + t_{2k}$

$X_7 = t_{2r} + t_{2s}$ | $X_{25} = t_{2s} - t_{2r}$ | $t_{2t} = t_{2d} - t_{2n}$ | $t_{2u} = i(t_{2j} - t_{2e})$

$X_{17} = t_{2t} - t_{2u}$ | $X_{15} = t_{2t} + t_{2u}$ | $t_{2v} = i(t_{2o} - t_{2g})$ | $t_{2w} = t_{2f} - t_{2k}$

$X_9 = t_{2v} + t_{2w}$ | | | $X_{23} = t_{2w} - t_{2v}$

IV. Cooleyn-Tukeyn radix-4 -erikoistapaus

Luvun 4.2 lopussa esitettiin lyhyt huomio siitä, että radix-2 -menetelmän lähestymistapaa on mahdollista soveltaa suuremmillakin jakajilla kuin 2. Seuraavaksi esitetään, kuinka sama prosessi suoritetaan jakajalla 4, ja todetaan, kuinka sillä saavutetaan vieläkin nopeampi muunnos.

Mikäli N on neljällä jaollinen, voidaan valita $N = r_1 r_2$ siten, että $r_2 = 4$, $r_1 = \frac{N}{4}$. Tällöin kaavat 4.2 ja 4.3 muuttuvat seuraavasti:

$$
\begin{aligned}
Y_{(4n_0+k_0)} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+k_0)} \cdot e^{-i2\pi(4n_0)k_1/N} \\
X_{(n_1\frac{N}{4}+n_0)} &= \sum_{k_0=0}^{3} Y_{(4n_0+k_0)} e^{-i2\pi(n_1r_1+n_0)k_0/N} \\
&= Y_{(4n_0+0)} e^{-i2\pi(n_1r_1+n_0)\cdot 0/N} + Y_{(4n_0+1)} e^{-i2\pi(n_1r_1+n_0)\cdot 1/N} \\
&+ Y_{(4n_0+2)} e^{-i2\pi(n_1r_1+n_0)\cdot 2/N} + Y_{(4n_0+3)} e^{-i2\pi(n_1r_1+n_0)\cdot 3/N} \\
&= Y_{(4n_0+0)} + Y_{(4n_0+1)} e^{-i2\pi(n_1r_1+n_0)/N} \\
&+ Y_{(4n_0+2)} e^{-i4\pi(n_1r_1+n_0)/N} + Y_{(4n_0+3)} e^{-i6\pi(n_1r_1+n_0)/N}.
\end{aligned}
$$

Nyt neljä erillistä tapausta, $k_0 = n_1 = 0 \ldots 3$, voidaan kirjoittaa auki seuraavasti:

$$
\begin{aligned}
Y_{(4n_0+0)} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+0)} \cdot e^{-i2\pi n_0 k_1/\left(\frac{N}{4}\right)} \\
Y_{(4n_0+1)} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+1)} \cdot e^{-i2\pi n_0 k_1/\left(\frac{N}{4}\right)} \\
Y_{(4n_0+2)} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+2)} \cdot e^{-i2\pi n_0 k_1/\left(\frac{N}{4}\right)} \\
Y_{(4n_0+3)} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+3)} \cdot e^{-i2\pi n_0 k_1/\left(\frac{N}{4}\right)} \\
X_{n_0} &= Y_{(4n_0+0)} + Y_{(4n_0+1)} e^{-i2\pi n_0/N} + Y_{(4n_0+2)} e^{-i4\pi n_0/N} + Y_{(4n_0+3)} e^{-i6\pi n_0/N} \\
X_{(\frac{N}{4}+n_0)} &= Y_{(4n_0+0)} - iY_{(4n_0+1)} e^{-i2\pi(\frac{N}{4}+n_0)/N} - Y_{(4n_0+2)} e^{-i4\pi(\frac{N}{4}+n_0)/N} + iY_{(4n_0+3)} e^{-i6\pi(\frac{N}{4}+n_0)/N} \\
X_{(\frac{N}{2}+n_0)} &= Y_{(4n_0+0)} - Y_{(4n_0+1)} e^{-i2\pi(\frac{N}{2}+n_0)/N} + Y_{(4n_0+2)} e^{-i4\pi(\frac{N}{2}+n_0)/N} - Y_{(4n_0+3)} e^{-i6\pi(\frac{N}{2}+n_0)/N} \\
X_{(\frac{3N}{4}+n_0)} &= Y_{(4n_0+0)} + iY_{(4n_0+1)} e^{-i2\pi(\frac{3N}{4}+n_0)/N} - Y_{(4n_0+2)} e^{-i4\pi(\frac{3N}{4}+n_0)/N} - iY_{(4n_0+3)} e^{-i6\pi(\frac{3N}{4}+n_0)/N}.
\end{aligned}
$$

Merkitään sitten neljää sarjaa A, B, C ja D, jolloin saadaan:

$$
\begin{aligned}
A_{n_0} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+0)} \cdot e^{-i2\pi n_0 k_1 / \left(\frac{N}{4}\right)} \\
B'_{n_0} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+1)} \cdot e^{-i2\pi n_0 k_1 / \left(\frac{N}{4}\right)} \\
C'_{n_0} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+2)} \cdot e^{-i2\pi n_0 k_1 / \left(\frac{N}{4}\right)} \\
D'_{n_0} &= \sum_{k_1=0}^{\frac{N}{4}-1} x_{(4k_1+3)} \cdot e^{-i2\pi n_0 k_1 / \left(\frac{N}{4}\right)} \\
B_{n_0} &= B'_{n_0} e^{-i2\pi n_0/N} \\
C_{n_0} &= C'_{n_0} e^{-i4\pi n_0/N} \\
D_{n_0} &= D'_{n_0} e^{-i6\pi n_0/N} \\
S_{n_0} &= (D_{n_0} - B_{n_0}) \cdot i \\
T_{n_0} &= (A_{n_0} - C_{n_0}) \\
U_{n_0} &= (A_{n_0} + C_{n_0}) \\
V_{n_0} &= (B_{n_0} + D_{n_0}) \\
X_{n_0} &= A_{n_0} + B_{n_0} + C_{n_0} + D_{n_0} \quad = U_{n_0} + V_{n_0} \\
X_{\left(\frac{N}{4}+n_0\right)} &= A_{n_0} - iB_{n_0} - C_{n_0} + iD_{n_0} = T_{n_0} + S_{n_0} \\
X_{\left(\frac{N}{2}+n_0\right)} &= A_{n_0} - B_{n_0} + C_{n_0} - D_{n_0} \quad = U_{n_0} - V_{n_0} \\
X_{\left(\frac{3N}{4}+n_0\right)} &= A_{n_0} + iB_{n_0} - C_{n_0} - iD_{n_0} = T_{n_0} - S_{n_0}.
\end{aligned}
$$

Kirjoitetaan tämä edelleen rekursiiviseen muotoon:

$$
\begin{aligned}
A'_k &= x_{(4k+0)} & S_k &= (D_k - B_k) \cdot i \\
B'_k &= x_{(4k+1)} & T_k &= (A_k - C_k) \\
C'_k &= x_{(4k+2)} & U_k &= (A_k + C_k) \\
D'_k &= x_{(4k+3)} & V_k &= (D_k + B_k) \\
A_k &= \mathrm{FFT}(A')_k & X_k &= U_k + V_k \\
B_k &= \mathrm{FFT}(B')_k \cdot e^{-i2\pi k/N} & X_{\left(\frac{N}{4}+k\right)} &= T_k + S_k \\
C_k &= \mathrm{FFT}(C')_k \cdot e^{-i4\pi k/N} & X_{\left(\frac{N}{2}+k\right)} &= U_k - V_k \\
D_k &= \mathrm{FFT}(D')_k \cdot e^{-i6\pi k/N} & X_{\left(\frac{3N}{4}+k\right)} &= T_k - S_k.
\end{aligned}
$$

Algoritmi Radix4: Radix-4 -versio Cooleyn-Tukeyn algoritmista.

Aikavaativuus

Radix4-algoritmissa kompleksikertolaskuja (merkitään tätä $h_{\mathrm{RX4},N}$) tarvitaan seuraavasti:

- tapauksessa $N = 4^m, m = 0$ tarvitaan $h_{\mathrm{RX4},N} = 0$ kertolaskua;
- tapauksessa $N = 4^m, m > 0$ tarvitaan $h_{\mathrm{RX4},N} = 4h_{\mathrm{RX4},N/4} + \frac{3}{4}N$ kertolaskua.

Kun $N = 4^m$, pätee $\frac{N}{4} = 4^{m-1}$. Edellisen kahden rivin pohjalta voidaan siis muodostaa seuraava rekursiivinen lukujono:

$$
\begin{aligned}
a_0 &= 0 \\
a_m &= 4a_{m-1} + 3 \cdot 4^{m-1}.
\end{aligned}
$$

Lukujonon yleinen jäsen on $a_m = 3 \cdot 4^{m-1}m$, mistä seuraa $h_{\mathrm{RX4},4^m} = a_m = \frac{3}{4} \cdot 4^m m$. Sijoittamalla $m = \log_4 N$ ja sieventämällä saadaan kertolaskujen lukumääräksi:

$$
h_{\mathrm{RX4},N} = \frac{3}{4} N \log_4 N = \left(\frac{3}{4 \log 4}\right) N \log N.
$$

Koska vakiotermiä $\frac{3}{4 \log 4}$ ei huomioida aikavaativuuslaskelmassa, algoritmin suoritusaikavaativuus on $\mathcal{O}(N \log N)$. Huomionarvoista on kuitenkin, että vakiotermi $\frac{3}{4 \log 4} \approx 0{,}54$ on pienempi kuin radix2-algoritmin vakiotermi $\frac{1}{2 \log 2} \approx 0{,}72$, mikä tarkoittaa, että mikäli syötteen koko on neljän potenssi, esimerkiksi 16384, on radix4-algoritmi keskimäärin n. 33 % nopeampi kuin radix2. Tutkielman kirjoittajan tekemät empiiriset kokeet vahvistavat teorian.

V. Pohdintaa ja yleistys Cooleyn-Tukeyn erikoistapauksista

Tutkimalla luvun 4.2 radix-2 -erikoistapausalgoritmia, liitteen IV radix-4 -erikoistapausalgoritmia sekä liitteen III listaa nopeista DFT-erikoistapauksista voidaan nähdä, että nopeat erikoistapaukset perustuvat itse asiassa kolmeen optimointiin yleisestä Cooleyn-Tukeyn kahden rekursion algoritmista CT2 (sivu 15), ja ne voidaan yleistää seuraavasti siten, että R on radix-koko ja $Q = \frac{N}{R}$, ja laskurit $r = \{0, 1, \ldots, R-1\}$ sekä $q = \{0, 1, \ldots, Q-1\}$:

$$
\begin{aligned}
&X_{(q+Qr)} := \mathrm{FFT}(x_{\{r,\, r+R,\, r+2R,\, \ldots,\, r+(Q-1)R\}})_q && \|R \text{ kpl } Q\text{-kokoisia FFT-muunnoksia} \\
&X_{(q+Qr)} := X_{(q+Qr)} e^{-i2\pi qr/N} \text{ jos } qr > 0 && \|\text{HUOM: Suoritetaan vain, kun } qr > 0 \\
&X_{(q+Qr)} := \mathrm{FFT}(X_{\{q,\, q+Q,\, q+2Q,\, \ldots,\, q+(R-1)Q\}})_r. && \|Q \text{ kpl } R\text{-kokoisia FFT-muunnoksia}
\end{aligned}
$$

Algoritmi CTgen: Geneerinen radix R -versio Cooleyn-Tukeyn algoritmista. Operaattori := kuvastaa muuttujan arvon korvaamista uudella.

Mainitut kolme optimointia ovat seuraavat:

- Sopivasti järjestelemällä algoritmia voidaan muuntaa niin, ettei lisätilaa ylimääräiselle taulukolle tarvita. Tämä ilmenee yllä esitetyssä algoritmissa siten, että samaan kohdetaulukkoon X kirjoitetaan kolmesti.
- Kun Q-kokoisten muunnosten jälkeen muunnettu taulukko kerrotaan $e^{i2\pi qr/N}$-arvoilla, voidaan nähdä, ettei ensimmäiselle taulukolle, eli indeksille 0, tarvitse suorittaa kertolaskuoperaatiota ollenkaan, sillä kaikki kertoimet ovat 1. Toisin kuin algoritmissa CT2, kertolasku on mahdollista jättää kokonaan pois ilman, että kohdetaulukkoon edes kirjoitetaan, sillä tässä algoritmiversiossa kertolaskun kohde on sama kuin lähde.
- Valitaan sellainen R (jolle pätee $N \equiv 0 \pmod{R}$), että R-kokoinen FFT-toteutus on toteutettu tehokkaasti siten, ettei siinä tarvitse laskea trigonometrisiä arvoja ollenkaan. Tämä pätee silloin, kun käytetään liitteen III algoritmeja erityisesti arvoilla $N \in \{2, 3, 4, 6, 8, 12\}$, ja aivan erityisesti arvoilla $N \in \{2, 4\}$, joilla FFT-laskentaan tarvitaan ainoastaan yhteenlaskuja sekä kompleksikonjugaatteja.

Algoritmin oikea toiminta edellyttää sitä, että kolmannella rivillä käytettävä R-kokoinen FFT kykenee suorittamaan muunnoksen niin, että lähde ja kohde ovat sama taulukko. Tämä algoritmi *itse* ei täytä kyseistä ehtoa, koska se tarvitsee alkuperäisiä lähdearvoja vielä senkin jälkeen, kun se on jo kirjoittanut kohteeseen muutettuja arvoja. Sen sijaan kaikki liitteen III algoritmit täyttävät kyseisen ehdon. Samoin ehdon täyttävät Raderin ja Bluesteinin algoritmit, koska molemmat niistä käsittelevät koko syötteen ennen ensimmäistä kirjoitusta kohdetaulukkoon. Ensimmäisellä rivillä käytettävä FFT-kutsu ei sisällä tällaista rajoitusta; siinä voi käyttää mitä tahansa algoritmia.