Virkeligt hurtige Fourier Transformationer

Aksel Mannstaedt

Kode med tekst og grafer - `https://nbviewer.org/github/unic0rn9k/fourier-notebook/blob/master/README.ipynb`
github repo - `https://github.com/unic0nr9k/fourier-notebook`

Indhold

1 Teori

1.1 Rekursion og funktionel programmering

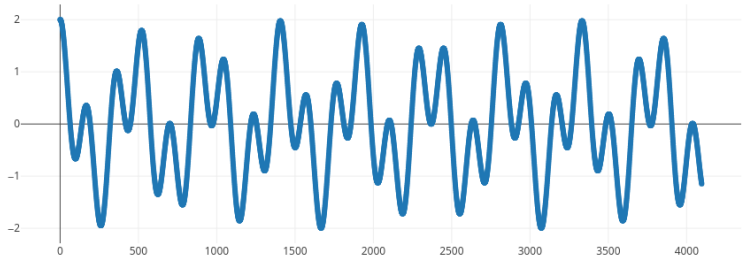
Rekursion er helt simpelt, en funktion der kalder sig selv. Dette bliver brugt meget i funktionel programmering.
Funktionel programmeringssprog er et paradigme kendetegnet ved et fokus på funktioner i stedet for objekter og kontrol strukturer. [1]

Fordele ved funktionelle programmeringssprog er at de er designet til at formindske side effects, og derved uforudventet adfærd. Side effects er defineret som en operation der muterer data udenfor den funktion der ejer data'en. Dette vil altså siges, at hvis man har en funktion der tager en pointer til en værdi og så ændrer på den værdi via pointeren, vil det være en side effect af funktionen.
Dette kan skabe uforudventet adfærd i et program og vil derfor typisk gerne undgås.
Fuldkommen funktionelle programmeringssprog er sprog, hvor man ikke kan mutere data overhovedet. Det betyder altså, at hvis man for eksempel vil lave et for loop, er man nødt til at bruge rekursion.
Fuldkommen funktionelle programmeringssprog bliver ikke rigtigt anvendt i praksis, men de er dog stadig relevante at skrive om, da de er tættere på matematisk notering end traditionelle programmeringssprog.
I min case har jeg anvendt fuldkommen funktionel psudo kode til at lave en model for hvordan jeg ville strukturere min implementering af Cooley-Tukey FFT algoritme.

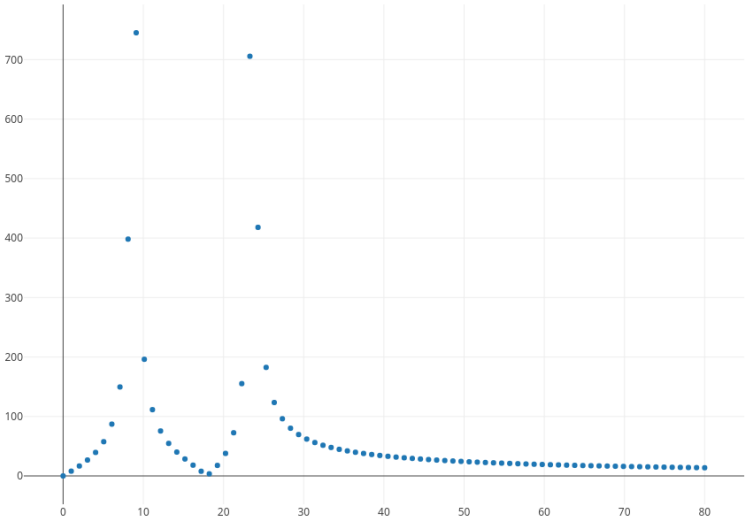
1.2 Hvad er en Fourier transformation?

[1] `https://en.wikipedia.org/wiki/Functional_programming`

En Fourier transformation er en funktion der tager et polynomium, repræsenteret som en koefficient-vektor, og transformerer den til en punkt-vektor repræsentation. Dette bliver også kaldt for en time to frequency -domain transformation, da den også kan repræsenteres som en transformation af bølger repræsenteret som intensitet over tid til intensitet over frekvens. Jeg vil primært fokusere på bølge repræsentationen af fourier transformationen, der jeg synes den er lidt nemmere at forstå.
Her er som eksempel en bølge og dens tilhørende fourier transformation. Bølgen er en kombination af en 10Hz bølge og en 25Hz bølge.



Her kan man altså se der ligger to top punkter lige omkring 10 og 25 på x-aksen.



1.3 DFT

DFT er kort for diskret fourier transformation, og var den originale formel opdaget af Joseph Fourier i 1830. Han fandt ud af, hvordan man kunne lave de her transformationer, før man kunne udregne dem på computere, eller havde nogen praktisk anvendelse til dem[2].
Her er f frekvens, t er tid, g er en sammensat bølge funktion og $\hat{g}$ er den Fourier transformerede funktion af g.

$$\hat{g}(f) = \int_{+}^{-} g(t) \cdot e^{-2\pi i f t} dt$$

Formlen ovenfor viser hvordan en diskret fourier transformation kan udregnes, den kan se lidt skræmmende ud, men er nem nok at forstå når man lige har styr på et par koncepter indenfor matematik.

1.3.1 Komplekse tal og Euler's identitet

Et komplekst tal er et tal bestående af en imaginær del og en reel del (summen af dem). Den imaginære del af det komplekse tal er defineret som produktet af et virkeligt tal og i.
i er defineret som kvadratroden af -1. (altså det findes ikke).
Det giver måske ikke lige umiddelbart så meget mening, men en nem måde at tænke på det, er som en 2d vektor der har nogen specielle regne regler.
For eksempel vil $(0 + 1i)^2$ være -1, da i^2 skal give -1.
Komplekse tal bliver typisk også visualiseret som en 2d-vektor, hvor den reelle del er på x-aksen, og den imaginære del er på y-aksen.
Her vil det svares til en 90° rotation, i det komplekse plan, at gange med i.

$$(1 + 0i) \cdot i = 0 + 1i$$
$$(0 + 1i) \cdot i = -1 + 0i$$
$$(-1 + 0i) \cdot i = 0 - 1i$$

osv...

[2] https://en.wikipedia.org/wiki/Joseph_Fourier

For at kunne forstå fourier transformeringer er det vigtigt at forstå Euler's identitet, som er defineret som

$$e^{\pi i} = -1$$

Dette udtryk kommer af at e^x er sit eget derivativ (e er Euler's konstant [3]), det vil altså siges at $\frac{d}{dx}e^{kx} = k \cdot e^{kx}$, hvor k er en konstant.
Altså man kan sige at e^{kx} bevæger sig i retningen $k \cdot e^{kx}$. Det betyder at hvis vi bytter k ud med i, må e^{ix} bevæge sig mod en 90° rotation med en hastighed af en radian per x.
Her vil en halv rotation derfor svare til $x = \pi$ og vi kan derved konkludere at $e^{\pi i} = -1$.

1.3.2 Uddybning af Fourier transformation

Den innerste del af den diskrete fourier transformering kan ses lidt som et prik produkt

$$f(t) = g(t) \cdot e^{-2\pi i f t}$$

Her vil det virkelige komponent af $f(t)$ være støre når intensiteten af $g(t)$ matcher den der ville findes hvis frekvensen af g var f.
Dette kan intuitivt forstås, som at når t værdier ligger i bølgedale, vil $e^{-2\pi i f t}$ være negativ og derfor vil $f(t)$ være positiv hvis $g(t)$ også er negativ.
-2π sikre at en øgelse af en tidsenhed svarende til en periode frekvensen f også vil resultere i en fuld rotation af $e^{-2\pi i f t}$.

1.4 FFT

1.4.1 Rekursion

2 Tooling (programmering)

Jeg valgte at skrive koden til denne case i rust, da jeg er komfortabel med sproget, og gerne ville eksperimentere med at lave en hurtig implementering af Cooley-Tukey algoritmen.
Rust er et rigtigt hurtigt sprog, dette skyldes blandt andet at det bruger llvm som backend, men også rust's brug af zero-cost-abstractions.
Jeg valgte at skrive koden i en jupyter notebook, da jeg ikke havde nogen egentlig applikation af min kode i tankerne under forløbet. Det viste sig også at være super praktisk til at lave tdd (test-driven-developement), da det

[3] `https://en.wikipedia.org/wiki/Euler%27s_constant`

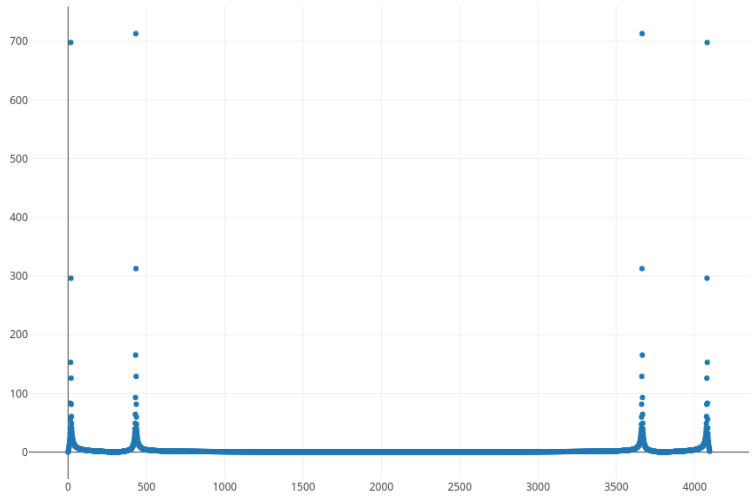
betød jeg kunne smide nogen grafer ind, og have dem opdateret i næsten realtime, mens jeg arbejdede på implementeringen af fft algoritmen.

2.1 Rust sprog paradimer

Rust er et memory-safe programmeringssprog, hvilket betyder at det by-default ikke lader en skrive koder, der kan forudsige undefined-behavior[4]

3 Inplace operationer og statisk allokering

4 Bilag



5 Bibliografi

1: Undefined-behavior - https://doc.rust-lang.org/reference/behavior-considered-undefine
html
2: Funktionel programmering - https://en.wikipedia.org/wiki/Functional_
programming
3: Fourier - https://en.wikipedia.org/wiki/Joseph_Fourier
4: Euler's konstant - https://en.wikipedia.org/wiki/Euler%27s_constant

[4] https://doc.rust-lang.org/reference/behavior-considered-undefined.html