

Fourierrekker fra bilder

Av Benjamin Dybvik, Tobias Blegeberg og Ole Martin Ruud

Innholdsfortegnelse

1. Introduksjon s. 1
2. Grunnleggende om Fourierrekke s. 2
 - Introduksjon til fourierrekker s. 2
 - Den generaliserte fourierrekken s. 2
 - Odde og like funksjoner s. 3
 - Den komplekse fourierrekken s. 5
 - Et eksempel på en fourierrekke s. 7
3. Noen anvendelser av fourierrekker s. 8
 - Anvendelser av fourierrekker s. 8
 - Eksempel på anvendelse i musikkinstrumenter s. 9
4. Lage en fourierrekke fra et bilde s.10
 - Hvordan fungerer JPEG s. 10
 - Vår omgjøring fra bilde til fourierrekke s. 15
 - Forklaring av teoridelen s.15
 - Anvendelse av teorien s. 18
5. Konklusjon s. 46
6. Referanser s. 47

1. Introduksjon

Hensikten med dette prosjektet er å få en innføring i hva fourierrekker er, og hvorfor fourierrekker brukes i flere løsninger på praktiske anvendelser. For å forstå hvorfor fourierrekker brukes skal vi først se på anvendelsen av fourierrekker i dagligdagse objekter, og litt generelt om anvendelser. Vi skal ikke gå inn i matematikken bak disse anvendelsene, men hovedfokusset kommer til å ligge på hvordan fourierrekker kan løse utfordringene i anvendelsene.

Hovedfokusset er å få en innføring i fourierrekker og hvordan man kan omgjøre et bilde til en fourierrekker. Bakgrunnen for at vi har valgt framstilling av bilder er fordi det er en spennende anvendelse av fourierrekker. Det er en praktisk og visuell bruk av fourierrekker, som er mulig å se grafisk, og er derfor enklere å ta stillig til. Å framstille bilder som fourierrekker er en metode som er mye brukt, da hensikten er å komprimere filstørrelsen på bildefilen betydelig, men uten å miste for mye av oppfattet bildekvalitet. Dette er en praktisk tilnærming til fourierrekker og det er derfor lett å sette seg inn i og forstå hensikten bak fourierrekker.

2. Grunnleggende om fourierrekker

Introduksjon til fourierrekker

En fourierrekke er en sum av harmoniske funksjoner med ulik frekvens og amplitude som er tilnærmet lik en periodisk funksjon. Med uendelig antall ledd i rekken så er fourierrekken til en funksjon lik funksjonen, men med en endelig mengde ledd så er fourierrekken tilnærmet lik funksjonen.

Utdelingen av fourierrekker kalles harmonisk analyse. Fourierrekken er svært nyttig, og kan brukes til å gjøre en tilfeldig periodisk funksjon om til en sum av enkle ledd. Man kan også velge antall ledd man ønsker basert på hvor nøyaktig man ønsker å gjennomføre tilnærmingen.

Den generaliserte fourierrekken

Hvis man bruker den generaliserte fourierrekken som benytter de ortogonale harmoniske funksjonene sinus og cosinus får man utledet fourierrekken til en funksjon ved hjelp av to koeffisienter, et konstant ledd og to summer slik som det er gitt under. For det lar seg interessere så er utledningen av disse generaliserte formlene gjennomført av Eric W. Weisstein i sin artikkel om "Generalized Fourier Series".

Anta at $f(t)$ en tilfeldig periodisk funksjon med periode T og at $c \in \mathbb{R}$. Da får man at fourierrekken til funksjonen $f(t)$ er $\psi(t)$. Under har man også variablen ω som er

vinkelhastigheten. Den er definert som $\omega = \frac{1}{T}$.

$$\begin{aligned} \psi(t) &= a_0 + \text{Sum}(a_n \cdot \cos(n\omega t) + b_n \cdot \sin(n\omega t), n = 1 \dots \infty) \\ \psi(t) &= a_0 + \sum_{n=1}^{\infty} (a_n \cos(n\omega t) + b_n \sin(n\omega t)) \end{aligned} \quad (2.2.1)$$

$$\begin{aligned} a_0 &= \frac{2}{T} \cdot \text{Int}(f(t), t = c \dots (c + T)) \\ a_0 &= \frac{2 \left(\int_c^{c+T} f(t) dt \right)}{T} \end{aligned} \quad (2.2.2)$$

$$\begin{aligned} a_n &= \frac{2}{T} \cdot \text{Int}(f(t) \cdot \cos(n\omega t), t = c \dots (c + T)) \\ a_n &= \frac{2 \left(\int_c^{c+T} f(t) \cos(n\omega t) dt \right)}{T} \end{aligned} \quad (2.2.3)$$

$$b_n = \frac{2}{T} \cdot \text{Int}(f(t) \cdot \sin(n\omega t), t = c \dots (c + T))$$

$$b_n = \frac{2 \left(\int_c^{c+T} f(t) \sin(n\omega t) dt \right)}{T} \quad (2.2.4)$$

Hvis $f(t)$ har diskontinuiteter, er fourierrekken til $f(t)$

$$\psi(t) = \frac{f(t^+) + f(t^-)}{2}$$

$$\psi(t) = \frac{f(t)}{2} + \frac{f(t^-)}{2} \quad (2.2.5)$$

Dette vil også være sant $t \in \mathbb{R}$ når funksjonen er kontinuerlig, men da vil uttrykket kunne bli faktorisert til $\psi(t) = f(t)$. Man sier at $f(t) \sim \psi(t)$.

▼ Odde og like funksjoner

Ved gitte vilkår kan det gjøres noen forenklinger av formelen for fourierrekken. Disse vil vise seg å være nyttige når man skal gjøre et bilde om til en fourierrekke.

Odde funksjoner

Når man finner fourierrekken til en odde funksjon finner man ut at man mister de like leddene i uttrykket. $f(t)$ er en odde funksjon og $\cos(n\omega t)$ er en like funksjon, derfor må produktet $f(t) \cdot \cos(n\omega t)$ være en odde funksjon. Integralet over en periode til en odde funksjon er 0.

Anta at $c \in \mathbb{R}$ og at funksjonen $f(t)$ er en periodisk, odde funksjon med periode T . Ta at $f(t)$ en tilfeldig periodisk funksjon med periode T og at $c \in \mathbb{R}$. Da får vi at fourierrekken til funksjonen $f(t)$ er $\psi(t)$. Under har man også variablen ω som er

vinkelhastigheten. Den er definert som $\omega = \frac{2\pi}{T}$.

$$a_n = \frac{2}{T} \cdot \text{Int}(f(t) \cdot \cos(n\omega t), t = c \dots (c+T));$$

$$a_n = 0$$

$$a_n = \frac{2 \left(\int_c^{c+T} f(t) \cos(n\omega t) dt \right)}{T}$$

$$a_n = 0 \quad (2.3.1)$$

Uttrykket for fourierrekken til en odde funksjon blir derfor:

$$\psi(t) = \text{Sum}(b_n \cdot \sin(n\omega t), n = 1 \dots \infty)$$

$$\psi(t) = \sum_{n=1}^{\infty} b_n \sin(n\omega t) \quad (2.3.2)$$

I tillegg går det an å forenkle uttrykket av b_n slik at det bare trengs å integrere over en halv periode.

$$b_n = \frac{4}{T} \cdot \text{Int}\left(f(t) \cdot \sin(n\omega t), t = c \dots \left(c + \frac{T}{2}\right)\right)$$

$$b_n = \frac{4 \left(\int_c^{c + \frac{T}{2}} f(t) \sin(n\omega t) dt \right)}{T} \quad (2.3.3)$$

Like funksjoner

Når $f(t)$ derimot er like, så skjer det mottatte. Nå er det derimot når man ganger med $\sin(n\omega t)$ at integralet blir 0. Det er fordi nå er $f(t)$ en like funksjon og $\sin(n\omega t)$ er en odd funksjon. Produktet av disse blir derfor en odd funksjon, og integralet over en periode til en odd funksjon er 0.

Anta at $c \in \mathbb{R}$ og at funksjonen $f(t)$ er en periodisk, like funksjon med periode T .

$$b_n = \text{Int}(f(t) \cdot \sin(n\omega t), t = c \dots (c + T));$$

$$b_n = 0$$

$$b_n = \int_c^{c+T} f(t) \sin(n\omega t) dt$$

$$b_n = 0 \quad (2.3.4)$$

Dermed blir uttrykket for fourierrekken til en like funksjon:

$$\psi(t) = \frac{a_0}{2} + \text{Sum}(a_n \cos(n\omega t), n = 1 \dots \infty)$$

$$\psi(t) = \frac{a_0}{2} + \left(\sum_{n=1}^{\infty} a_n \cos(n\omega t) \right) \quad (2.3.5)$$

Her kan man også forenkle uttrykket for a_0 og a_n slik at det bare trengs å integrere over en halv periode.

$$a_0 = \frac{4}{T} \cdot \text{Int}\left(f(t), t = c \dots \left(c + \frac{T}{2}\right)\right);$$

$$a_n = \frac{4}{T} \cdot \text{Int}\left(f(t) \cdot \cos(n\omega t), t = c \dots \left(c + \frac{T}{2}\right)\right)$$

$$a_0 = \frac{4 \left(\int_c^{c + \frac{T}{2}} f(t) dt \right)}{T}$$

$$a_n = \frac{4 \left(\int_c^{c + \frac{T}{2}} f(t) \cos(n\omega t) dt \right)}{T} \quad (2.3.6)$$

Den komplekse fourierrekken

Fourierrekker består av to harmoniske funksjoner. Disse harmoniske funksjonene er sinus og cosinus. Ved hjelp av Eulers formel kan man gjøre disse om til et utrykk av e^{ix} . Eulers formel forteller oss at

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}; \sin(x) = \frac{e^{ix} - e^{-ix}}{2i};$$

$$\cos(x) = \frac{e^{ix}}{2} + \frac{e^{-ix}}{2} \quad (2.4.1)$$

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i} \quad (2.4.1)$$

Når man setter (2.3.1) inn i formelen (2.2.1) for fourierrekken, kan man utlede den komplekse formen av fourierrekken.

$$\psi(t) = \frac{a_0}{2} + \text{Sum}(a_n \cdot \cos(n\omega t) + b_n \cdot \sin(n\omega t), n = 1 \dots \text{infinity})$$

$$\psi(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(n\omega t) + b_n \sin(n\omega t)) \quad (2.4.2)$$

$$\psi(t) = \frac{a_0}{2} + \text{Sum}\left(a_n \cdot \frac{e^{i \cdot n \cdot \omega} + e^{-i \cdot n \cdot \omega}}{2} + b_n \cdot \frac{e^{i \cdot n \cdot \omega} - e^{-i \cdot n \cdot \omega}}{2i}, n = 1 \dots \text{infinity}\right)$$

$$\psi(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \left(\frac{e^{in\omega}}{2} + \frac{e^{-in\omega}}{2} \right) + \frac{b_n (e^{in\omega} - e^{-in\omega})}{2i} \right) \quad (2.4.3)$$

$$\psi(t) = \frac{a_0}{2} + \text{Sum}\left(\frac{a_n - i \cdot b_n}{2} \cdot e^{i \cdot n \cdot \omega \cdot t}, n = 1 \dots \text{infinity}\right) + \text{Sum}\left(\frac{a_n + i \cdot b_n}{2} \cdot e^{-i \cdot n \cdot \omega \cdot t}, n = 1 \dots \text{infinity}\right)$$

$$\psi(t) = \frac{a_0}{2} + \left(\sum_{n=1}^{\infty} \frac{a_n - i b_n e^{in\omega t}}{2} \right) + \left(\sum_{n=1}^{\infty} \left(\frac{i b_n}{2} + \frac{a_n}{2} \right) e^{-in\omega t} \right) \quad (2.4.4)$$

Får å kunne skrive fourierrekken på en enkel måte så lager man noen definisjoner.

$$c_0 = \frac{a_0}{2}$$

$$c_0 = \frac{a_0}{2} \quad (2.4.5)$$

$$c_n = \frac{a_n - i \cdot b_n}{2}$$

$$c_n = -\frac{i b_n}{2} + \frac{a_n}{2} \quad (2.4.6)$$

$$c_{-n} = \frac{a_n + i \cdot b_n}{2}$$

$$c_{-n} = \frac{i b_n}{2} + \frac{a_n}{2} \quad (2.4.7)$$

Deretter setter man disse definisjonene inn i uttrykket (2.3.4) ovenfor så blir det som utledet over.

$$\begin{aligned} \psi(t) &= \frac{a_0}{2} + \text{Sum} \left(\frac{a_n - i b_n}{2} \cdot e^{i \cdot n \cdot \omega \cdot t}, n = 1 \dots \text{infinity} \right) + \text{Sum} \left(\frac{a_n + i b_n}{2} \cdot e^{-i \cdot n \cdot \omega \cdot t}, n = 1 \dots \text{infinity} \right) \\ \psi(t) &= \frac{a_0}{2} + \left(\sum_{n=1}^{\infty} \frac{a_n - i b_n e^{in\omega t}}{2} \right) + \left(\sum_{n=1}^{\infty} \left(\frac{i b_n}{2} + \frac{a_n}{2} \right) e^{-in\omega t} \right) \end{aligned} \quad (2.4.8)$$

$$\begin{aligned} \psi(t) &= c_0 + \text{Sum}(c_n \cdot e^{i \cdot n \cdot \omega \cdot t}, n = 1 \dots \text{infinity}) + \text{Sum}(c_{-n} \cdot e^{-i \cdot n \cdot \omega \cdot t}, n = 1 \dots \text{infinity}) \\ \psi(t) &= c_0 + \left(\sum_{n=1}^{\infty} c_n e^{in\omega t} \right) + \left(\sum_{n=1}^{\infty} c_{-n} e^{-in\omega t} \right) \end{aligned} \quad (2.4.9)$$

$$\begin{aligned} c_n &= \frac{1}{T} \text{Int}(f(t) \cdot e^{-i \cdot n \cdot \omega \cdot t}, t = c \dots (c + T)) \\ c_n &= \frac{\int_c^{c+T} f(t) e^{-in\omega t} dt}{T} \end{aligned} \quad (2.4.10)$$

Resultatet (2.3.9) fra denne utledningen kalles den komplekse formen av fourierrekken. Den komplekse formen for fourierrekker er algebraisk enklere og mer symmetrisk, derfor brukes denne formen av fourierrekker ofte i blant annet fysikk. For å komme rett fra funksjonen til den komplekse formen av fourierrekken kan man bruke (2.3.10).

På den komplekse formen av fourierrekken så er det et teorem som heter Parsevals Teorem som er greit å ha sett. Hvis det antas at $f(t)$ er en periodisk funksjon med en periode på T og at $c \in \mathbb{R}$.

$$\frac{1}{T} \operatorname{Int}\left((|f(t)|^2, t = c \dots (c + T)) \right) = \operatorname{Sum}\left((|C_n|^2, n = -\infty \dots \infty) \right)$$

$$\frac{\int_c^{c+T} |f(t)|^2 dt}{T} = \sum_{n=-\infty}^{\infty} |C_n|^2 \quad (2.4.11)$$

Hvis $f(t)$ er reell, altså at det ikke har en imaginær del, da får man videre at.

$$\operatorname{Sum}\left((|C_n|^2, n = -\infty \dots \infty) \right) = (|C_0|^2) + 2 \cdot \operatorname{Sum}\left((|C_n|^2, n = 1 \dots \infty) \right)$$

$$\sum_{n=-\infty}^{\infty} |C_n|^2 = |C_0|^2 + 2 \left(\sum_{n=1}^{\infty} |C_n|^2 \right) \quad (2.4.12)$$

Et eksempel på en fourierrekke

Under skal det vises fra et eksempel på en periodisk funksjon $f(t)$ som det skal lages en fourierrekke til. T er perioden til funksjonen.

restart;

$f := t \rightarrow \operatorname{piecewise}(0 \leq t \leq \pi, -1, \pi < t \leq 2\pi, 1); T := 2\pi;$

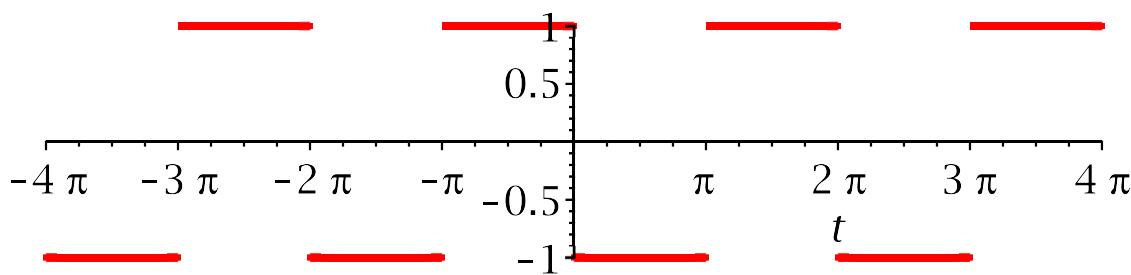
$$f := t \mapsto \begin{cases} -1 & 0 \leq t \leq \pi \\ 1 & \pi < t \leq 2\pi \end{cases}$$

$$T := 2\pi \quad (2.5.1)$$

$P_s := j \rightarrow \operatorname{plot}(f(t - j \cdot T), t = j \cdot T \dots T \cdot (j + 1), \operatorname{discont} = \text{true}, \operatorname{symbol} = \text{"point"}, \operatorname{color} = \text{"red"},$

$\operatorname{thickness} = 3)$:

$P_f := \operatorname{plots:-display}([\operatorname{seq}(P_s(j), j = -2 .. 1)])$



Figur 1

$$c_n := n \rightarrow \text{int} \left(f(t) \cdot \exp \left(-I \cdot n \cdot \left(\frac{2 \cdot \text{Pi}}{T} \right) \cdot t \right), t = 0 \dots T \right)$$

$$c_n := n \mapsto \int_0^T f(t) e^{-\frac{2In\pi t}{T}} dt \quad (2.5.2)$$

$$\psi := (t, l) \rightarrow \sum \left(\frac{1}{T} \cdot c_n(n) \cdot \exp \left(I \cdot n \cdot \left(\frac{2 \cdot \text{Pi}}{T} \right) \cdot t \right), n = -l \dots l \right)$$

$$\psi := (t, l) \mapsto \sum_{n=-l}^l \frac{c_n(n) e^{\frac{2In\pi t}{T}}}{T} \quad (2.5.3)$$

Her regnes ut fourierrekken til $f(t)$ (2.4.1) med l antall ledd.

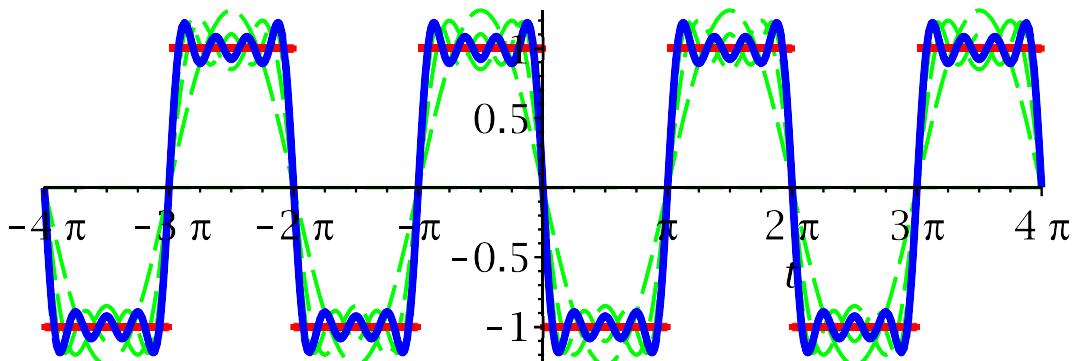
$l := 7$:

$\Psi_1 := \text{evalc}(\psi(t, l))$

$$\Psi_1 := -\frac{4 \sin(7t)}{7\pi} - \frac{4 \sin(5t)}{5\pi} - \frac{4 \sin(3t)}{3\pi} - \frac{4 \sin(t)}{\pi} \quad (2.5.4)$$

Her plottes fourierrekken (2.4.4), vist her i blått, med l antall ledd og orginalfunksjonen $f(t)$ (2.4.1), vist her i rødt. I tillegg er det tegnet opp hver fourierrekke med færre ledd enn rekken (2.4.4) vi regnet ut. Slik kan man se hvordan flere ledd endrer på rekke. Rekkene med færre ledd l er grønne.

```
Pψ := plot(ψ1, t = -2·T ... 2·T, color = "blue", thickness = 3):
Prψ := plot( [seq(ψ(t, j), j = 0 ... (l-1))], t = -2·T ... 2·T, color = "green", linestyle
           = "dash"):
plots:-display([Prψ, Pf, Pψ])
```



Figur 2

3. Noen anvendelser av fourierrekker

Anvendelser av fourierrekker

Det viser seg at man kan bruke fourierrekker til å framstille nesten alle slags bølger som en funksjon. Som vist tidligere kan en fourierrekke kunne tilnærme seg en bølge ved å ha uendelig antall cosinus og sinus ledd. Dette gir oss en enorm mulighet til å framstille mange

ting rundt oss som funksjoner ettersom så mye vi omgår dagligdags enten er eller kan oppfattes som bølger. Eksempler på slike dagligdagse bølger vi omgår kan være i det elektromagnetiske spekteret slik som signaler fra en mobil eller lydspekeret slik som musikk eller en stemme. Mekaniske ting som går i en viss takt kan vi også framstille som cosinus og sinus bølger. Eksempler på dette er klokker, motorer eller kraftverk der maskinene går i en fast syklus. [1] Ingeniører bruker fourierrekker til å illustrere forskjellige bølger som funksjoner. Oppfinneren av fourierrekker Joseph Fourier brukte fourierrekker for å regne på varmetap, varmebølger og vibrasjoner.

Ingeniører bruker fourierrekker innen mange fagfelt. For å kunne regne på høy og lavvann og hvordan bølger påvirker et skip, til komprimering av data og tolkning av elektromagnetiske bølger slik som i telekommunikasjon. Dette gjelder mange områder innenfor signalbehandling, slik som radar, sonar, x-ray, mobiltelefoni og samband. I for eksempel Wifi brukes fourierrekker for å kode og dekode signalet presist slik at det kan bli tolket digitalt. Det brukes også for å restaurere signaler som er i ferd med å «viskes ut» på grunn av dårlig dekning. [3]

Ved å ta opp lyd kan man konstruere dette om til en fourierrekke funksjon. Denne funksjonen består av mange ledd som går an å dele opp og analysere separat. Ved å analysere de forskjellige frekvensene som den originale lyden består av, kan man danne seg en graf. Et program som Shazam bruker denne teknikken til å finne ut hvilken sang man hører på. Shazam tar grafen og sammenligner den med en database for å se om den matcher. Om Shazam finner en match vil den fortelle deg hvilke sang du hører på. [2] Smarttelefon stemmegjenkenningsprogramvare, slik som Siri, bruker samme teknologi for å forstå hva du ber den om å gjøre. Ved å analysere stemmen din og sjekke det opp med en Apple database skal den være i stand til å gjøre som du ber den om. Så langt kan man få et lite inntrykk av hva man kan framstille med fourierrekker og det er langt flere muligheter. Fourierrekker kan være et veldig kraftig verktøy i forskjellige anvendelser om det blir brukt smart.

[1] <https://math.stackexchange.com/questions/579453/real-world-application-of-fourier-series> 21.09.17

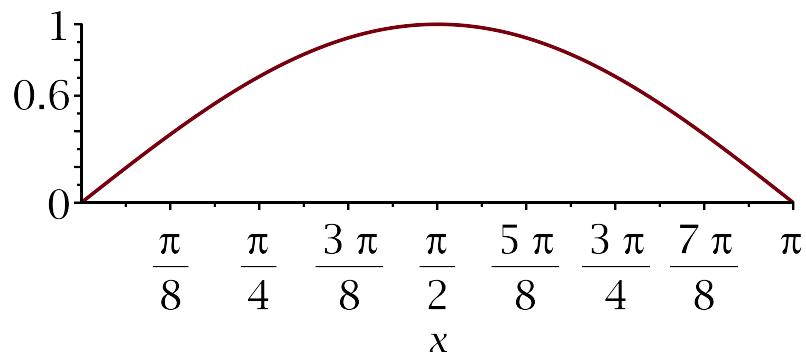
[2] https://www.youtube.com/watch?v=K0_TP5Sf7yc 21.09.17

[3] <https://www.quora.com/Why-are-Fourier-series-important-Are-there-any-real-life-applications-of-Fourier-series> 21.09.17

Eksempel på anvendelse i musikkinstrumenter

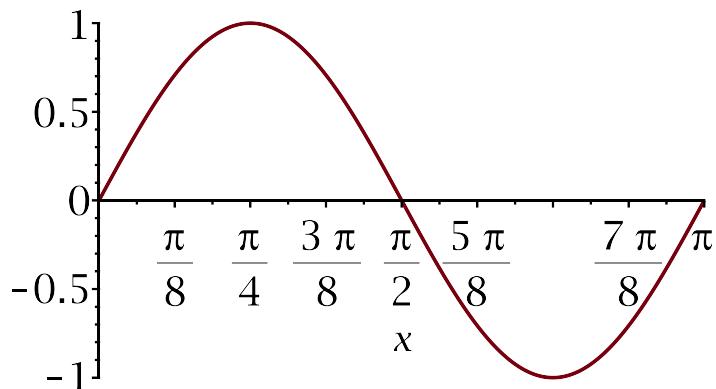
Her skal det ses litt nærmere på hvordan fourierrekker blir anvendt i dagligdagse ting og objekter, som for eksempel lyd og instrumenter. Hva er det som skjer når man drar buen over en cello? Til å begynne med må det bli bestemt hva lyd er for noe. En simpel forklaring er at lyd er fortynninger og fortetninger i luften, eller i et annet medium som disse bølgene kan bevege seg i. Disse bølgene kalles for lydbølger. Det er trommehinnen i øret som greier å registrere disse hurtige endringene i lufttrykket. Dette er fordi mennesker har membraner inne i øret som vil svinge i samme takt som lydbølgene. Grunnen til at celloen kan lage disse harmoniske lydbølgene med frekvenser som kan høres er på grunn av celloens utforming.

Celloen har flere strenger med en bestemt lengde som er tilpasset tonen den skal lage. I dette eksemplet bestemmes det for enkelhetsskyld at lengden på cellostrenget er π . Cellostrenget legges langs med x-aksen. Siden lengden på strengen er π blir også perioden $T = \pi$. Av disse opplysningene kan man lage en funksjon av cellostrenget. Dette er illustrert under som $f(x) = \sin(x)$



Figur 3

Dette er utslaget man får hvis man drar buen over celloen. Dette definerer man i dette eksempelet som en tone på 65 Hz som tilsvarer en C i musikk skalaen. Grafen svinger her opp og ned 65 ganger i sekundet. Så legger man en finger i punktet $\frac{\pi}{2}$, da får vi funksjonen $f(x) = \sin(2x)$.



Figur 4

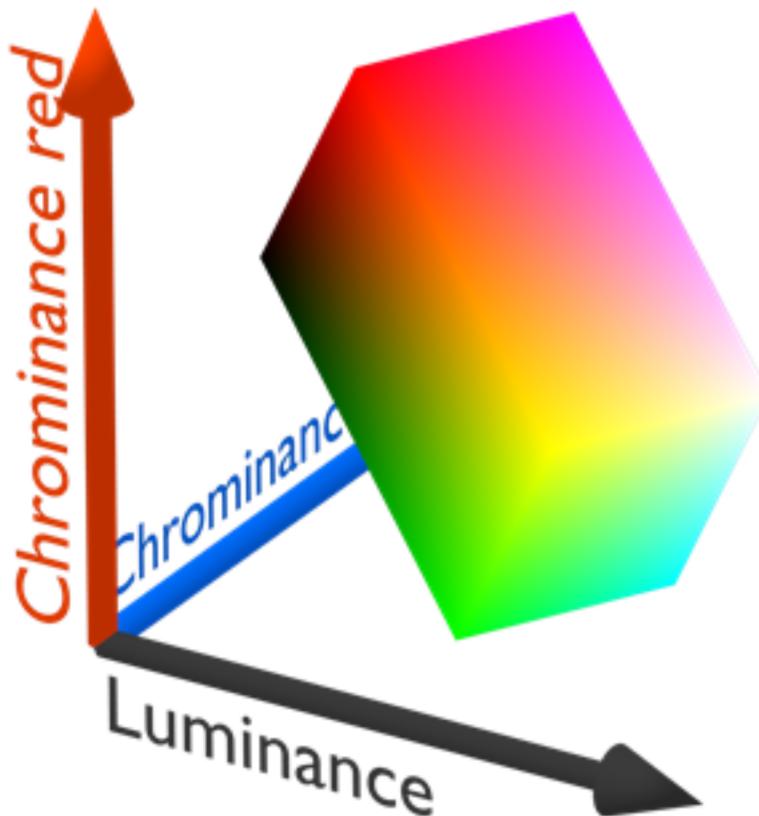
Dette er fouriersinusrekker som kan utledes videre på formen $f(x) = \sum_{i=1}^{\infty} b_n \cdot \sin(n\pi x)$. Dette er en tone på 130 Hz som da også er en C, men denne tonen er høyere i toneskala. Det vil si dobbelt så mange svingninger. Det man skal ta med seg er at for hver gang frekvensen dobles seg så får man den sammen tonen, men i en skala opp. Dette er mye brukt innenfor for eksempel musikkteori.

4. Lage en fourierrekke fra et bilde

Hvordan fungerer JPEG

JPEG (Joint Photographic Experts Group) er et filformat for lagring av bilder. Mens vanlig bildeformat lagrer en verdi for hvert piksel så tar JPEG å komprimerer bildet. Dette resulterer at det tar opp mindre plass enn originalfilen. Det finnes forskjellige versjoner av JPEG. Vi kan få JPEG der bildekvaliteten ikke blir tapt og JPEG der bildekvalitet blir tapt. I utgangspunktet er det ønskelig å bruke JPEG med tap av bildekvalitet. Det er hensiktsmessig fordi litt tap i bildekvalitet kan føre til at man får komprimert bildet betydelig bedre. Den dataen som blir lukket ut av bildet har lite eller ikke noe å si på hvordan vi oppfatter slutt produktet.

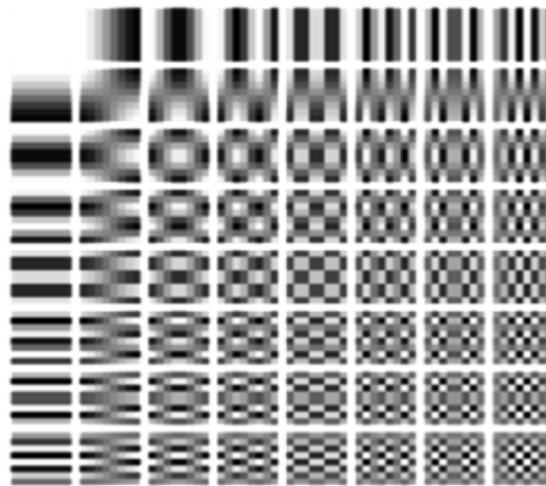
Først blir pikslene gjort om fra RGB til noe som heter YCbCr. YCbCr er en fargemodell slik som RGB. Istedentfor å representere fargen gjennom en fargekombinasjon av rød, grønn og blå så kan YCbCr representere det samme fargespekteret gjennom Y, Cb og Cr. Y er luminans, det er et mål på hvor lyst et område er. Cb og Cr er rødnyanser og blånyanser til bildet. Gjennom disse tre komponentene får vi det samme fargespekteret som RGB gir oss.



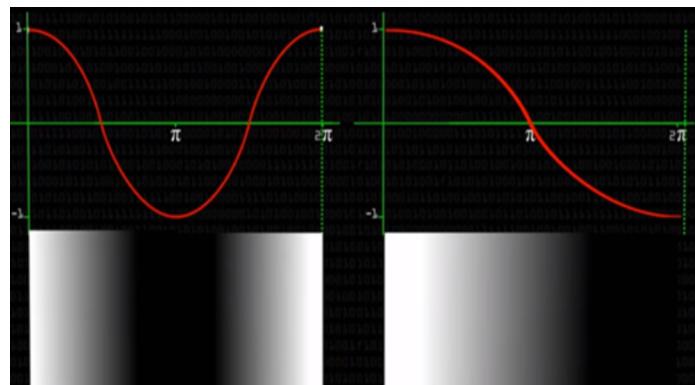
Figur 5

Grunnen til at vi gjør om fra RGB til YCrCb er fordi at vi kan redusere data fra den sistnevnte uten at det endrer hvordan vi oppfatter det ferdige resultatet. Øynene våre er relativt dårlige på å se forskjell mellom fargenyanser. Dette gjør av vi kan kutte ut en del data fra Cb og Cr uten at det reduserer vårt inntrykk av hvordan vi oppfatter bildet. Ved å fjerne data fra Cr og Cb kutter vi litt av kvaliteten på bildet. Dette er noe av bildekvalitetstapet som vi har med JPEG. Luminans oppfattes av oss mennesker som gråtoner. Vi mennesker mye bedre på å se nyanser av gråtoner enn nyanser av farger. Det blir som å se midt på natta når det er mørkt. Du kan skimte hva som er foran deg, men du klarer ikke å se farger. I YCrCb får vi avgrenset luminanskomponenten, slik at vi får redusert de andre komponentene uten å påvirke luminansen i bildet.

Deretter deler JPEG bildet opp i kvadrater på 8x8 piksler der hver fargekomponent er adskilt. Hver av disse komponentene som er delt opp i 8x8 blokker kan bli representert av Discrete Cosine Transformation (DCT), Diskré Cosinus Transformasjon. Diskré Cosinus Transformasjon er basert på Joseph Fourier sin Fourierrekke. I forskjell fra Fourierrekke bruker DCT kun cosinusledd, cosinus dette er for å slippe å ta stilling til både sinus og cosinus ledd og dermed redusere den totale dataen enda mer.



Figur 6



Figur 7

Denne matrisen er bygd opp av cosinusledd med forskjellige frekvenser i x planet (horisontalt) og y planet (vertikalt).

$$a_n \cos(n\omega t)$$

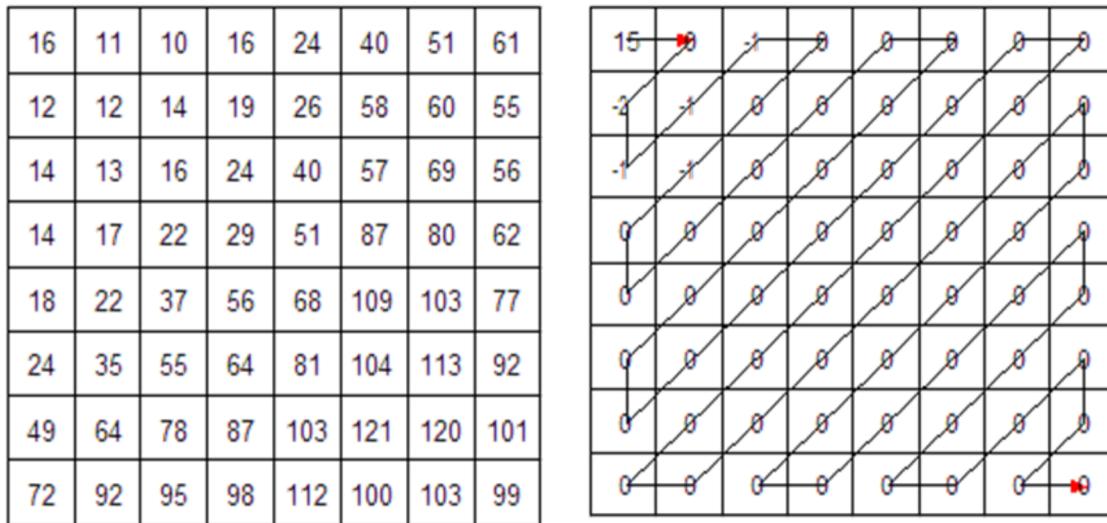
På bildet til høyre ser vi hvordan cosinusleddene representerer matrisen. Amplituden bestemmer nyansen av komplimentene til Y, Cr og Cb. Det er her snakk om komplimentfargene, vi kan se på bildet her at det er luminanskomponenten y. Frekvensen bestemmer hvor ofte det skal byttes mellom komplimentene. Matrisen opp til venstre kombinerer cosinusleddene i y planet og x planet og gis oss 8 ganger 8 antall kombinasjoner. Disse 64 kombinasjonene legges lagvis for å rekonstruere en 8x8 blokk med bilde. De ulike kombinasjonene trengs i ulik grad for å gjenskape bildet presist. For å bestemme hvor mye hver kombinasjon skal bidra regnes bidraget ut som koeffisienten av fourierrekken i DCT. DCT matrisen og vektleggingen av kombinasjonene er kun basert på fourierrekker. Det gjør

ved at fourierrekkeformelen $\psi(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(n\omega t) + b_n \sin(n\omega t))$ er alt vi trenger for å gjenskape 8x8 blokken.

Videre for å redusere data betraktelig brukes en tabell som heter «Quantized table». Hensikten med kvantisering (quantized) er å fjerne de bidragene vi ikke vil ha, slik som de med veldig høy frekvens og de som har liten betydning for bildet. Dette gjøres fordi vi kan redusere filstørrelsen ved å utelukke disse leddene med uvesentlige detaljer. I tillegg er disse mindre detaljene i bildet vanskelig for mennesker å oppfatte, vårt helhetsinntrykk av bildet

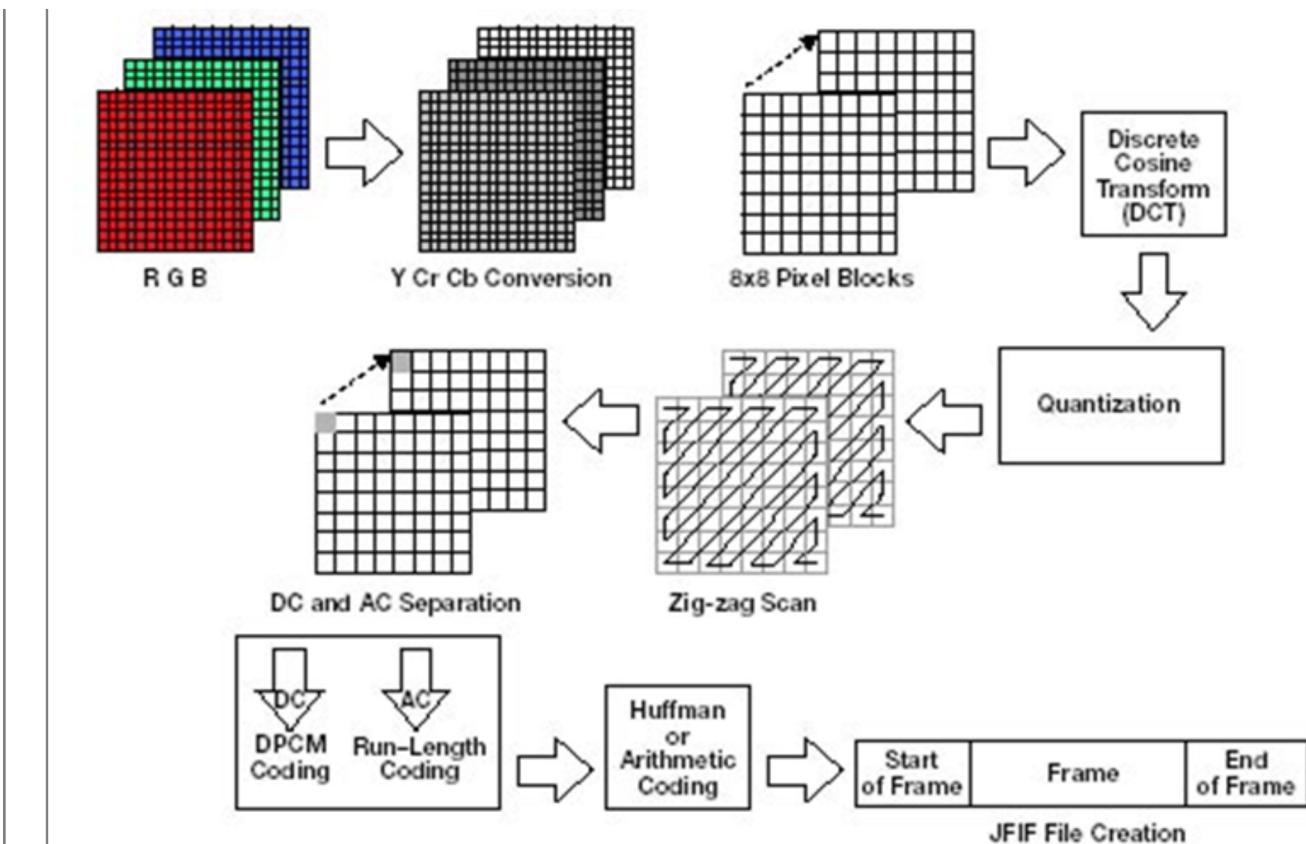
vil forblі det samme. En slik kvantiseringstabell er forhåndsbestemt av JPEG og inngår i et regnestykke sammen med koeffisienten som divisor og dividend for å bestemme hvor mye vi velger å vektlegge bidraget. En slik kvantisert tabell ser vi på bildet under til venstre. Legg merke til at verdiene nede til høyre er større enn de oppo til vestre. Dette er fordi det er ønskelig at verdiene nede til høyre veier mindre enn de verdiene oppo til venstre. Koeffisient / kvantisert = ny data.

Bidragene etter kalkulasjonen vil være mindre enn de var før. I tillegg vil flesteparten av bidragene ha blitt desimaltall. Disse desimaltallene rundes av til nærmeste heltall. Siden en del av disse bidragene har blitt veldig små så vil de rundes av til null. Det vil bli større representasjon av null nede til høyre enn oppo til venstre ettersom de mindre bidragene i matrisen allerede har blitt dividert med de større kvantiseringverdiene. Når vi da leser igjennom tabellen sikk-sakk vil de tellende verdiene komme først og etterfulgt av en lang rekke med null. Det er hensiktsmessig å lese igjennom tabellen sikk-sakk ettersom vi får plassert hele rekka med null bakerst og ved siden av hverandre. En slik lang rekke med null kan lett bli komprimert av Huffman koding. Bildet under til høyre viser hvordan dataen til blir lest om til en rekke.



Figur 8

Prosessen som vi nå har snakket om vil gjenta seg for alle 8x8 blokkene i bildet til bildet har blitt komprimert. Denne dataen vil da bli lagret som en komprimert JPEG fil. Hele prosessen er kort illustrert i følgende bilde



Figur 9

For å gjøre denne JPEG filen om til et forståelig bilde gjøres denne prosessen baklengs. Dataen blir dekomprimert og legges tilbake i en matrise. Deretter blir alle verdiene multiplisert med kvantiseringstabellen. Ettersom bidragene tidligere hadde blitt rundet av etter divisjonen så vil tallene nå bli noe annerledes. Til tross for det er verdiene i helhet ganske like og vil ikke påvirke vår oppfatning. Flesteparten av verdiene hadde blitt rundet ned til null. Disse verdiene vil forblи null etter at bidragene har blitt multiplisert med kvantiseringstabellen. Disse verdiene gjelder i hovedsak for høye frekvenser, slike frekvenser som vi ikke oppfatter mye av, disse annuleringene vil dermed ikke få stor betydning for bildet heller. Når det er sagt så vil sluttproduktet være annerledes. Feil og mangler kan bli sett av et trent øye. Bidragene i forskjellige frekvensområder blir lagt sammen for å gjenskape den originale 8x8 blokken. Blokkene blir så lagt side om side for å gjenskape det fulle bildet.

For å sammenligne JPEG og rådata er det her hentet ut noen sammenligninger fra "Image Compression and the Discrete Cosine Transform" av Ken Cabeen og Peter Gent fra College of the Redwoods.



Figure 5 – Original Peppers



Figure 6 – Quality 50 – 84% Zeros



Figure 7 – Quality 20 – 91% Zeros



Figure 8 – Quality 10 – 94% Zeros

Figur 10

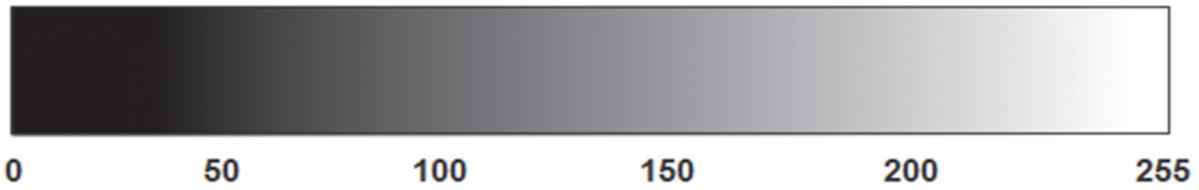
Ovenfor har vi bilde av noen paprikaer og chillier i forskjellige format. Kvaliteten og oppløsningen er angitt under, der alle er JPEG utenom den første som er en ukomprimert rådatafil. Nå som vi er klar over dette kan vi studere og sammenligne bildene i detalj. Vi ser en gradvis endring i bildene når JPEG ellersimerer data og runder av. Bildet blir mer kornete. Om man sammenligner bildet lengst til venstre og lengst til høyre ser vi tydige forskjeller. Bildet lengst til høyre er mye mer kornete og ser ikke like naturlig ut. Når det er sagt har vi ikke noe problem med å se konturen og helheten i bildet.

Vår omgjøring fra bilde til fourierrekke

Forklaring av teoridelen

Tidligere ble det definert en fourierrekke som summen av harmoniske funksjoner med ulike frekvenser og amplituder. I tillegg er et kvav at den er tilnærmet lik en periodisk funksjon. Nå skal et bilde bli representert ved hjelp av en fourierrekke. Ved å bryte ned et

bilde i flere pixler vil hver individuelle pixel få en egen fargeverdi. I dette eksemplet skal det brukes fargeverdier fra og med 0 til og med 255, hvor 0 er sort og 255 er hvit. Fargeverdien definerer altså hvilken fargetone en pixel har. Dette er illustrert i figuren under.



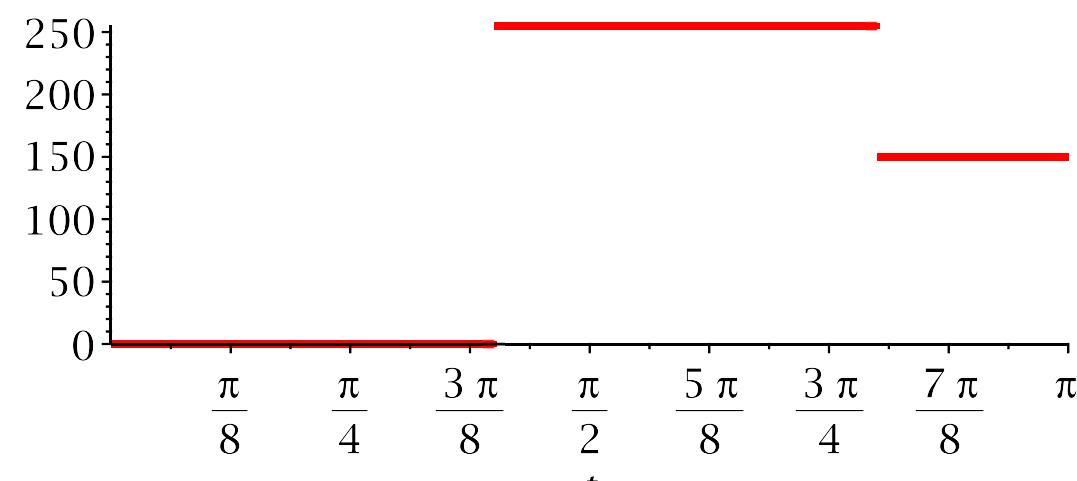
Figur 11

Nå som det er mulig å representere fargetonene som fargeverdier skal også fargeverdiene framstilles grafisk. Langs den horisontale aksen får hver pixel et like stort intervall. Hvor alle pixlene i bildet totalt utgjør perioden T . Langs den vertikale aksen er fargeverdiene, som er representert gjennom amplituden til funksjonen. Når alle verdiene fra pixlene er satt inn i graf $f(t)$ vil funksjonsuttrykket i det gitte intervallet være lik fargeverdien, altså en horisontal rett linje. Som et eksempel tas et bilde som inneholder følgene fargeverdier i kronologisk rekkefølge: 0, 255, 150, 150, 0 (sort, hvit, grå, grå, sort). Under er $f(t)$ presentert med perioden $T = \pi$.

$$f := t \rightarrow \text{piecewise} \left(0 \leq t \leq \frac{2\pi}{5}, 0, \frac{2\pi}{5} \leq t \leq \frac{4\pi}{5}, 255, \frac{4\pi}{5} \leq t \leq \frac{6\pi}{5}, 150, \frac{6\pi}{5} \leq t \leq \frac{8\pi}{5}, 150, \frac{8\pi}{5} \leq t \leq \pi, 0 \right);$$

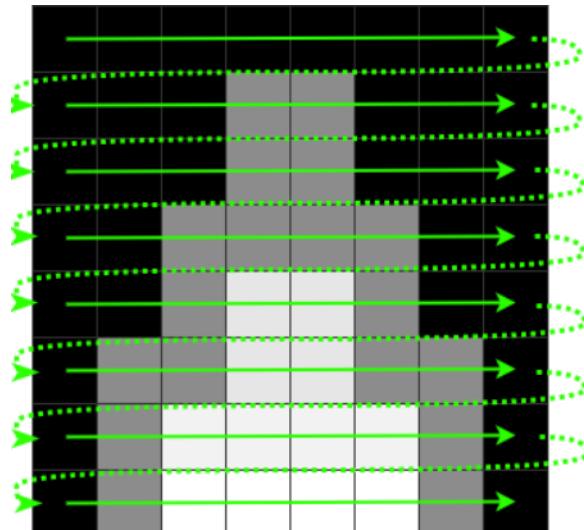
$$f := t \mapsto \begin{cases} 0 & 0 \leq t \leq \frac{2\pi}{5} \\ 255 & \frac{2\pi}{5} \leq t \leq \frac{4\pi}{5} \\ 150 & \frac{4\pi}{5} \leq t \leq \frac{6\pi}{5} \\ 150 & \frac{6\pi}{5} \leq t \leq \frac{8\pi}{5} \\ 0 & \frac{8\pi}{5} \leq t \leq \pi \end{cases} \quad (4.2.1.1)$$

```
plot(f(t), t = 0 .. Pi, discontinuity = true, symbol = "point", color = "red", thickness = 3)
```

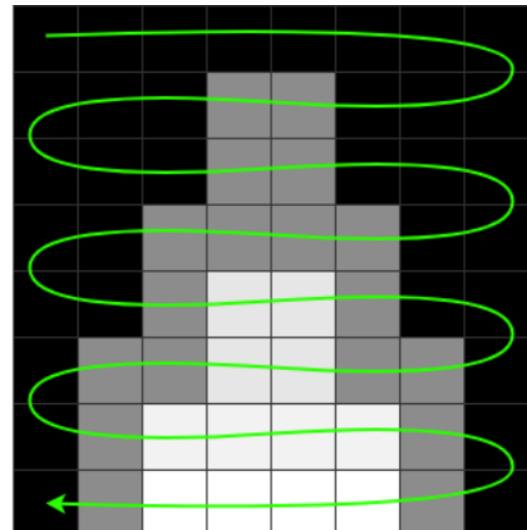


Figur 12

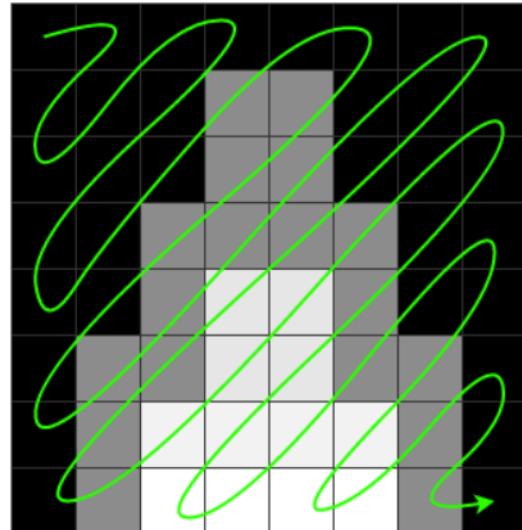
Nå som det er vist hvordan man kan plotte pixlene inn i grafen, må man vite hvordan man systematisk skal legge inn pixlene fra bildet. Her skal man finne den måten som man legger inn data på hvor kurven av fourierrekken blir tilnærmet lik bildet. For eksempel vil en løsning være å legge alle pixlene med lik fargeverdi i samlede grupper. På den måten får kurven færrest utslag, og sannsynligheten for at verdiene som er tilnærmet like bilde, blir lik de faktiske verdiene til bildet. I oppgaven har vi definert tre ulike metoder som vi vil se nærmere på. Metode nr. 1 er å starte øverst i venstre hjørne og lese fra venstre mot høyre for hver linje. Metode nr. 2 starter også i øverste venstre hjørne, men for hver nye linje går man motsatt vei av forrige linje. Metode nr. 3 starter i øverste venstre hjørne og går i et sikksakk mønster diagonalt ned mot det nederste høyre hjørne. (Se Figur 2)



Metode 1



Metode 2



Metode 3

Figur 13

Vi skal i anvendelsesdelen benytte oss av to ulike bilder. Et "kunstig" i den form av at de inneholder bråe overganger fra sort til hvitt. Dette er typisk et bilde av tekst eller figurer. Det andre bildet er "naturlig" i den form av at det inneholder relativt jevne overganger mellom lyse og mørke områder. Dette er typisk bilder man har tatt med fotokamera av landskap eller personer. Med disse to ulike bildene håper vi å illustrere i hvilke tilfeller denne "komprimeringsmetoden" vil fungere best.

Anvendelse av teorien

Fremgangsmåte

Nå som vi har 3 ulike metoder for å gjøre et bilde om til en fourierrekke, så skal vi generere en fourierrekke med hver av metodene og analysere hvilken som gir den beste

tilnærmingen. Under er det en liste i kronologisk rekkefølge som forteller hvordan vi skal gjøre et bilde om til en fourierrekke.

1. Lage to ulike typer bilder, et med skarpe kanter mellom lyst og mørkt og et med naturlige, myke overganger.
2. Dele bildet opp i mindre biter på 8x8 piksler, for å ikke få for mye data per fourierrekke.
3. Generere en funksjon $f(t)$ som minner om (4.2.1) som benytter seg av de 3 ulike metodene.
4. Lage en graf som viser $f(t)$ og $\psi(t)$ med ulike antall ledd.
5. Generere et bilde utifra $\psi(t)$ og sammelikne kvaliteten med orginalen
6. Analysere resultatene for å finne ut:
 - Hvilk metode som er best for ulike bildetyper
 - Hvor nært originalen bildet blir

Metode 1

Naturlig bilde

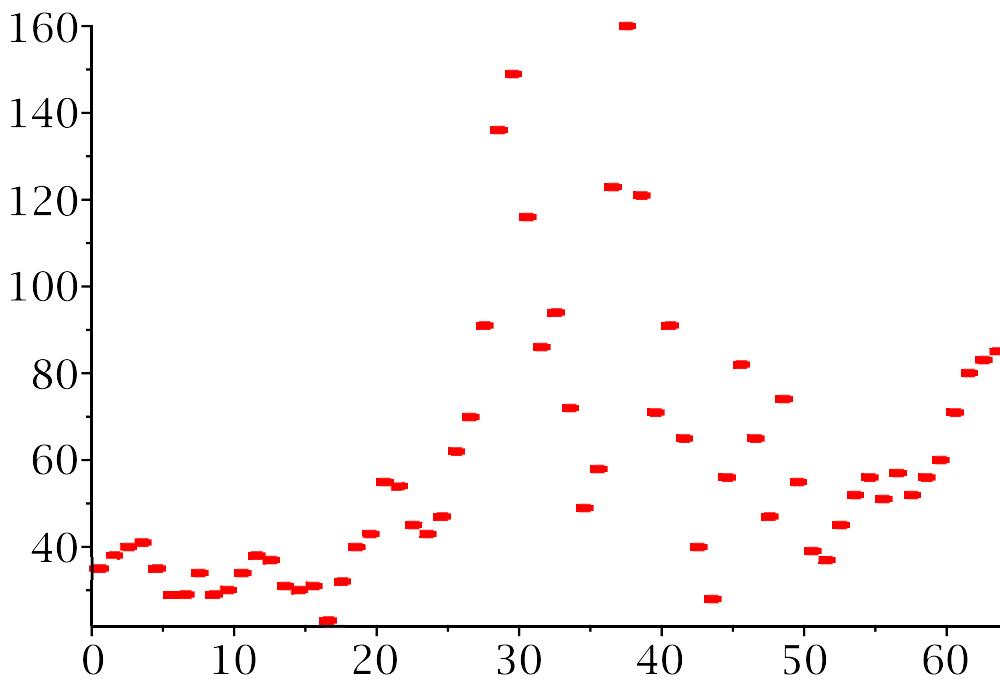
Under er utregningen av metode 1 med det naturlige.png bilde.

restart;

$$f(t) := \text{piecewise}(0 \leq t \leq 1.0, 35, 1.0 \leq t \leq 2.0, 38, 2.0 \leq t \leq 3.0, 40, 3.0 \leq t \leq 4.0, 41, 4.0 \leq t \leq 5.0, 35, 5.0 \leq t \leq 6.0, 29, 6.0 \leq t \leq 7.0, 29, 7.0 \leq t \leq 8.0, 34, 8.0 \leq t \leq 9.0, 29, 9.0 \leq t \leq 10.0, 30, 10.0 \leq t \leq 11.0, 34, 11.0 \leq t \leq 12.0, 38, 12.0 \leq t \leq 13.0, 37, 13.0 \leq t \leq 14.0, 31, 14.0 \leq t \leq 15.0, 30, 15.0 \leq t \leq 16.0, 31, 16.0 \leq t \leq 17.0, 23, 17.0 \leq t \leq 18.0, 32, 18.0 \leq t \leq 19.0, 40, 19.0 \leq t \leq 20.0, 43, 20.0 \leq t \leq 21.0, 55, 21.0 \leq t \leq 22.0, 54, 22.0 \leq t \leq 23.0, 45, 23.0 \leq t \leq 24.0, 43, 24.0 \leq t \leq 25.0, 47, 25.0 \leq t \leq 26.0, 62, 26.0 \leq t \leq 27.0, 70, 27.0 \leq t \leq 28.0, 91, 28.0 \leq t \leq 29.0, 136, 29.0 \leq t \leq 30.0, 149, 30.0 \leq t \leq 31.0, 116, 31.0 \leq t \leq 32.0, 86, 32.0 \leq t \leq 33.0, 94, 33.0 \leq t \leq 34.0, 72, 34.0 \leq t \leq 35.0, 49, 35.0 \leq t \leq 36.0, 58, 36.0 \leq t \leq 37.0, 123, 37.0 \leq t \leq 38.0, 160, 38.0 \leq t \leq 39.0, 121, 39.0 \leq t \leq 40.0, 71, 40.0 \leq t \leq 41.0, 91, 41.0 \leq t \leq 42.0, 65, 42.0 \leq t \leq 43.0, 40, 43.0 \leq t \leq 44.0, 28, 44.0 \leq t \leq 45.0, 56, 45.0 \leq t \leq 46.0, 82, 46.0 \leq t \leq 47.0, 65, 47.0 \leq t \leq 48.0, 47, 48.0 \leq t \leq 49.0, 74, 49.0 \leq t \leq 50.0, 55, 50.0 \leq t \leq 51.0, 39, 51.0 \leq t \leq 52.0, 37, 52.0 \leq t \leq 53.0, 45, 53.0 \leq t \leq 54.0, 52, 54.0 \leq t \leq 55.0, 56, 55.0 \leq t \leq 56.0, 51, 56.0 \leq t \leq 57.0, 57, 57.0 \leq t \leq 58.0, 52, 58.0 \leq t \leq 59.0, 56, 59.0 \leq t \leq 60.0, 60, 60.0 \leq t \leq 61.0, 71, 61.0 \leq t \leq 62.0, 80, 62.0 \leq t \leq 63.0, 83, 63.0 \leq t \leq 64.0, 85) : T := 2·64;$$

$$T := 128 \quad (4.2.2.1.1)$$

$$P_{org} := \text{plot}\left(f(t), t = 0 .. \frac{T}{2}, \text{discont} = \text{true}, \text{symbol} = \text{"point"}, \text{color} = \text{"red"}, \text{thickness} = 3\right)$$



Figur 14

$$a_0 := \frac{4}{T} \cdot \text{int}\left(f(t), t = 0 \dots \frac{T}{2}\right)$$

$$a_0 := 119.0000000 \quad (4.2.2.1.2)$$

$$a_n := n \rightarrow \frac{4}{T} \cdot \text{int}\left(f(t) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), t = 0 \dots \frac{T}{2}\right)$$

$$a_n := n \mapsto \frac{4 \left[\int_0^{\frac{T}{2}} f(t) \cos\left(\frac{2 n \pi t}{T}\right) dt \right]}{T} \quad (4.2.2.1.3)$$

$$\psi := (t, l) \rightarrow \frac{a_0}{2} + \text{sum}\left(a_n(n) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), n = 1 \dots l\right)$$

$$\psi := (t, l) \mapsto \frac{a_0}{2} + \left(\sum_{n=1}^l a_n(n) \cos\left(\frac{2 n \pi t}{T}\right) \right) \quad (4.2.2.1.4)$$

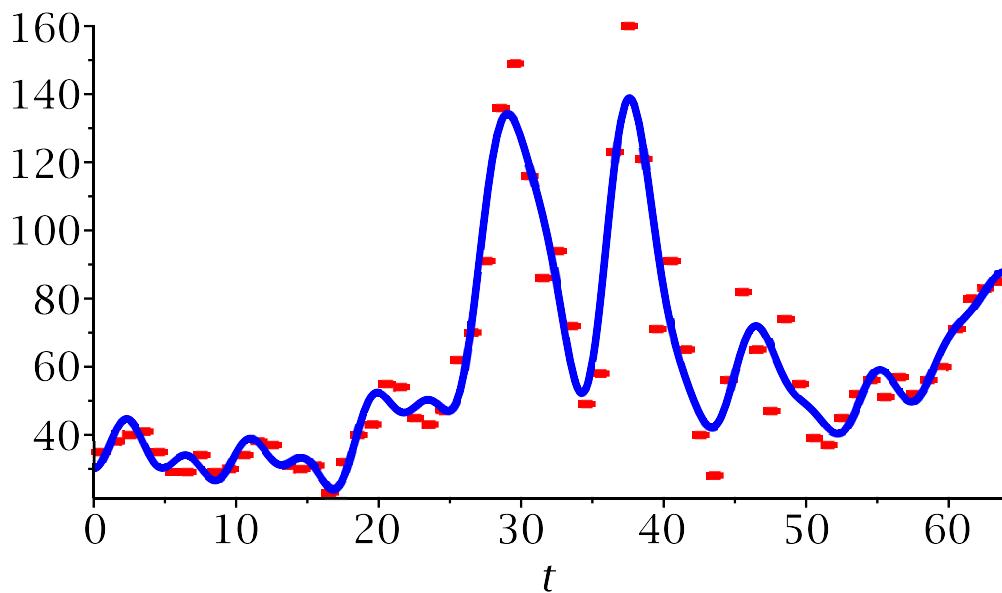
$l := 30 :$

$$\psi_1 := \text{evalf}(\psi(t, l))$$

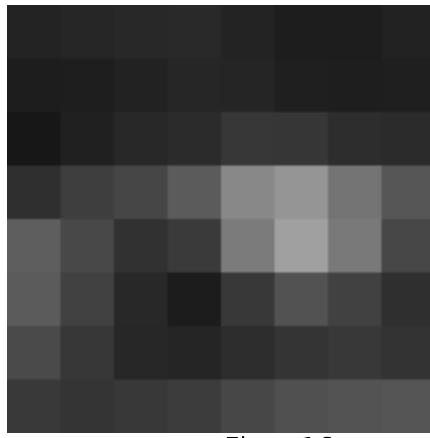
$$\begin{aligned} \psi_1 := & -15.22332499 \cos(0.04908738522 t) \\ & + 1.228146342 \cos(1.325359401 t) \\ & + 6.731108123 \cos(0.6872233931 t) \\ & - 1.896305736 \cos(0.2945243113 t) \end{aligned} \quad (4.2.2.1.5)$$

$$\begin{aligned}
& -0.9876229724 \cos(1.227184630 t) \\
& -3.416310602 \cos(0.3436116965 t) \\
& +6.699547492 \cos(0.8344855487 t) \\
& -4.499224927 \cos(1.423534171 t) \\
& +0.5969812318 \cos(1.178097245 t) \\
& +2.666858801 \cos(0.3926990818 t) \\
& -4.981558929 \cos(0.5890486226 t) \\
& -2.204861449 \cos(0.7853981635 t) \\
& -1.029279685 \cos(0.8835729339 t) \\
& +0.224621038 \cos(0.1472621557 t) \\
& +10.41940379 \cos(0.6381360078 t) \\
& -2.409057142 \cos(0.9326603192 t) \\
& +19.76409236 \cos(0.1963495409 t) \\
& -2.013328299 \cos(1.079922475 t) \\
& -21.51398980 \cos(0.09817477044 t) \\
& +1.555520528 \cos(1.374446786 t) \\
& -0.2065333038 \cos(1.129009860 t) \\
& -5.451235899 \cos(0.5399612374 t) \\
& -14.41919822 \cos(0.7363107783 t) \\
& -1.790991221 \cos(1.472621557 t) \\
& +4.862893996 \cos(0.4908738522 t) \\
& +0.2239924832 \cos(0.9817477044 t) \\
& -3.492448351 \cos(0.2454369261 t) \\
& +1.445977888 \cos(1.030835090 t) \\
& -1.451916587 \cos(1.276272016 t) \\
& +1.158730052 \cos(0.4417864670 t) + 59.50000000
\end{aligned}$$

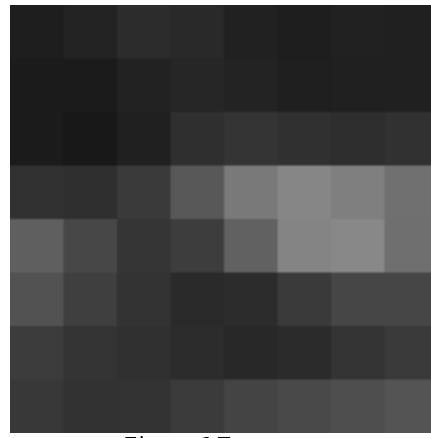
$$\begin{aligned}
P_{\psi} &:= \text{plot}\left(\psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{"blue"}, \text{thickness} = 3\right) : \\
\text{plots:-display}\left(\left[P_{\text{org}}, P_{\psi}\right]\right)
\end{aligned}$$



Figur 15



Figur 16



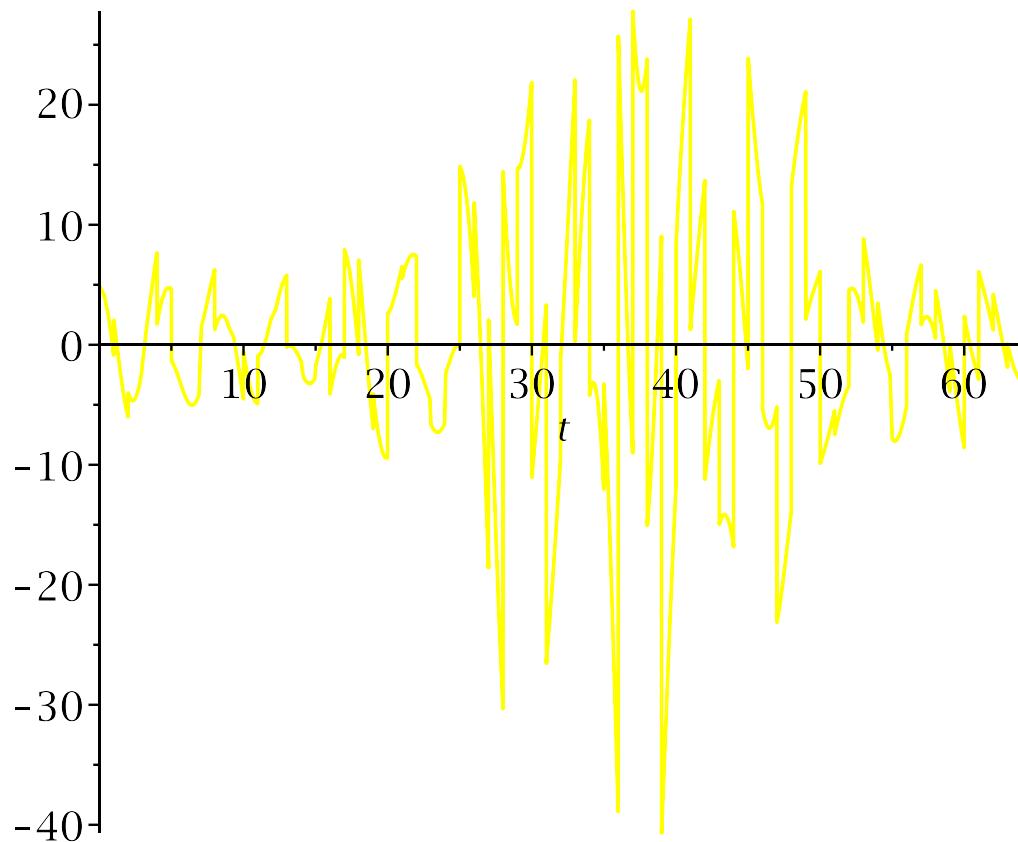
Figur 17

Bilde til venstre er originalbildet, og det til høyre er konstruert av fourierrekken. Bildet som er et resultat av fourierrekkene, har mistet flest fargeverdier i de områdene hvor kontrasten er høyest. Under er endringen i fargeverdier fra det originale bildet til det konstruerte, representert i en matrise og en graf. Dette bekrefter også at med metode 1 er endringene størst i områdene med store kontraster, som for eksempel i det lyse området på høyre siden av bildet.

```
 $\Delta := Matrix( [ [5, 3, -4, 0, 2, -1, -4, 2],$ 
 $[2, 3, 0, 0, 2, 0, -2, -1],$ 
 $[-4, 8, 8, -4, 3, 6, -1, -6],$ 
 $[-2, 15, 12, 3, 15, 15, -11, -26],$ 
 $[-1, 1, -4, -3, 26, 28, -15, -40],$ 
 $[9, 2, -11, -14, 12, 24, -5, -23],$ 
 $[14, 3, -9, -7, 5, 9, 4, -7],$ 
 $[1, 2, 5, 1, 3, 7, 5, 1]] )$ 
```

$$\Delta := \begin{bmatrix} 5 & 3 & -4 & 0 & 2 & -1 & -4 & 2 \\ 2 & 3 & 0 & 0 & 2 & 0 & -2 & -1 \\ -4 & 8 & 8 & -4 & 3 & 6 & -1 & -6 \\ -2 & 15 & 12 & 3 & 15 & 15 & -11 & -26 \\ -1 & 1 & -4 & -3 & 26 & 28 & -15 & -40 \\ 9 & 2 & -11 & -14 & 12 & 24 & -5 & -23 \\ 14 & 3 & -9 & -7 & 5 & 9 & 4 & -7 \\ 1 & 2 & 5 & 1 & 3 & 7 & 5 & 1 \end{bmatrix} \quad (4.2.2.1.6)$$

$$P_{\Delta} := \text{plot}\left(f(t) - \psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{yellow}\right)$$



Figur 18

Kunstig bilde

Under er utregningen av metode 1 med det kunstige bildet. Altså det bildet som har skarpe kanter mellom farger.

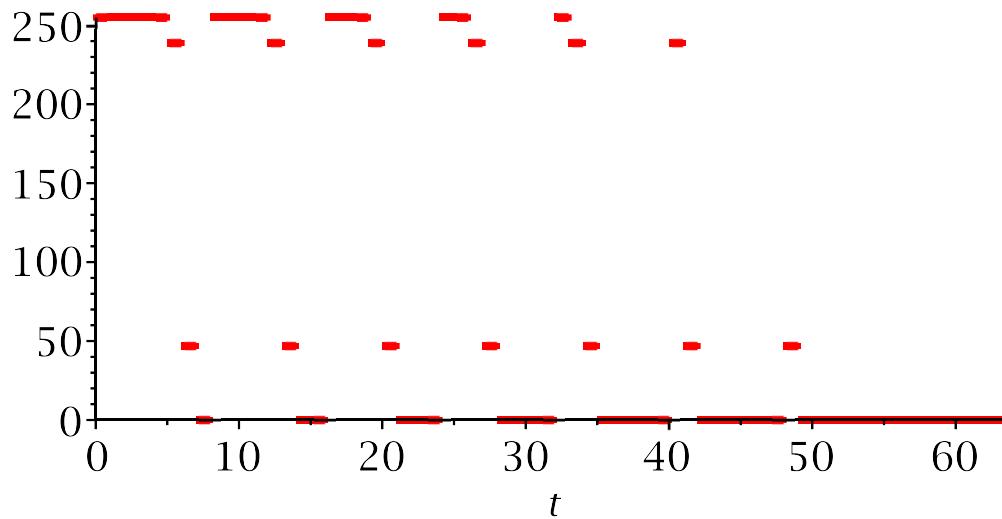
restart,

$f(t) := \text{piecewise}(t, 0 \leq t \leq 1.0, 255, 1.0 \leq t \leq 2.0, 255, 2.0 \leq t \leq 3.0, 255, 3.0 \leq t \leq 4.0, 255, 4.0 \leq t \leq 5.0, 255, 5.0 \leq t \leq 6.0, 239, 6.0 \leq t \leq 7.0, 47, 7.0 \leq t \leq 8.0, 0, 8.0 \leq t$

$\leq 9.0, 255, 9.0 \leq t \leq 10.0, 255, 10.0 \leq t \leq 11.0, 255, 11.0 \leq t \leq 12.0, 255, 12.0 \leq t \leq 13.0, 239, 13.0 \leq t \leq 14.0, 47, 14.0 \leq t \leq 15.0, 0, 15.0 \leq t \leq 16.0, 0, 16.0 \leq t \leq 17.0, 255, 17.0 \leq t \leq 18.0, 255, 18.0 \leq t \leq 19.0, 255, 19.0 \leq t \leq 20.0, 239, 20.0 \leq t \leq 21.0, 47, 21.0 \leq t \leq 22.0, 0, 22.0 \leq t \leq 23.0, 0, 23.0 \leq t \leq 24.0, 0, 24.0 \leq t \leq 25.0, 255, 25.0 \leq t \leq 26.0, 255, 26.0 \leq t \leq 27.0, 239, 27.0 \leq t \leq 28.0, 47, 28.0 \leq t \leq 29.0, 0, 29.0 \leq t \leq 30.0, 0, 30.0 \leq t \leq 31.0, 0, 31.0 \leq t \leq 32.0, 0, 32.0 \leq t \leq 33.0, 255, 33.0 \leq t \leq 34.0, 239, 34.0 \leq t \leq 35.0, 47, 35.0 \leq t \leq 36.0, 0, 36.0 \leq t \leq 37.0, 0, 37.0 \leq t \leq 38.0, 0, 38.0 \leq t \leq 39.0, 0, 39.0 \leq t \leq 40.0, 0, 40.0 \leq t \leq 41.0, 239, 41.0 \leq t \leq 42.0, 47, 42.0 \leq t \leq 43.0, 0, 43.0 \leq t \leq 44.0, 0, 44.0 \leq t \leq 45.0, 0, 45.0 \leq t \leq 46.0, 0, 46.0 \leq t \leq 47.0, 0, 47.0 \leq t \leq 48.0, 0, 48.0 \leq t \leq 49.0, 47, 49.0 \leq t \leq 50.0, 0, 50.0 \leq t \leq 51.0, 0, 51.0 \leq t \leq 52.0, 0, 52.0 \leq t \leq 53.0, 0, 53.0 \leq t \leq 54.0, 0, 54.0 \leq t \leq 55.0, 0, 55.0 \leq t \leq 56.0, 0, 56.0 \leq t \leq 57.0, 0, 57.0 \leq t \leq 58.0, 0, 58.0 \leq t \leq 59.0, 0, 59.0 \leq t \leq 60.0, 0, 60.0 \leq t \leq 61.0, 0, 61.0 \leq t \leq 62.0, 0, 62.0 \leq t \leq 63.0, 0, 63.0 \leq t \leq 64.0, 0) : T := 2\cdot64;$

$$T := 128 \quad (4.2.2.1.7)$$

$P_{org} := \text{plot}\left(f(t), t = 0 .. \frac{T}{2}, \text{discont} = \text{true}, \text{symbol} = \text{"point"}, \text{color} = \text{"red"}, \text{thickness} = 3\right)$



Figur 19

$$a_0 := \frac{4}{T} \cdot \text{int}\left(f(t), t = 0 .. \frac{T}{2}\right)$$

$$a_0 := 174.6250000 \quad (4.2.2.1.8)$$

$$a_n := n \rightarrow \frac{4}{T} \cdot \text{int}\left(f(t) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), t = 0 .. \frac{T}{2}\right)$$

$$a_n := n \mapsto \frac{4 \left(\int_0^{\frac{T}{2}} f(t) \cos\left(\frac{2 n \pi t}{T}\right) dt \right)}{T} \quad (4.2.2.1.9)$$

$$\begin{aligned}\psi := (t, l) \rightarrow & \frac{a_0}{2} + \text{sum}\left(a_n(n) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), n = 1 \dots l\right) \\ \psi := (t, l) \mapsto & \frac{a_0}{2} + \left(\sum_{n=1}^l a_n(n) \cos\left(\frac{2 n \pi t}{T}\right) \right)\end{aligned}\quad (4.2.2.1.10)$$

$l := 30 :$

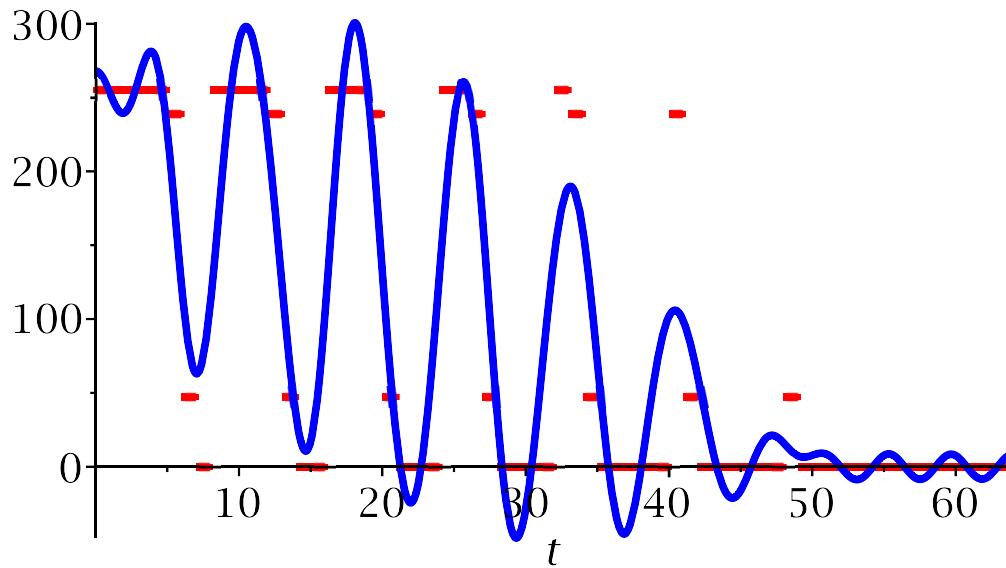
$\psi_1 := \text{evalf}(\psi(t, l))$

$$\psi_1 := 7.889420971 \cos(1.472621557 t) \quad (4.2.2.1.11)$$

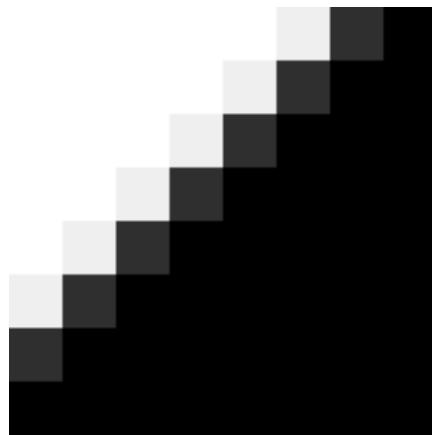
$$\begin{aligned}& + 4.342268974 \cos(1.227184630 t) \\& + 4.103968138 \cos(1.276272016 t) \\& + 3.146958559 \cos(1.325359401 t) \\& - 58.79769526 \cos(0.8835729339 t) \\& + 3.053133579 \cos(0.9326603192 t) \\& + 13.47659130 \cos(0.9817477044 t) \\& + 4.337172612 \cos(1.129009860 t) \\& + 2.326883904 \cos(1.178097245 t) \\& + 6.998963100 \cos(1.374446786 t) \\& + 1.510748248 \cos(1.030835090 t) \\& + 4.367747012 \cos(1.423534171 t) \\& + 3.919654678 \cos(1.079922475 t) \\& + 106.0587638 \cos(0.04908738522 t) \\& + 17.06261344 \cos(0.09817477044 t) \\& + 6.453685153 \cos(0.1472621557 t) \\& + 10.76460122 \cos(0.1963495409 t) \\& + 4.46659445 \cos(0.2454369261 t) \\& + 5.800017107 \cos(0.2945243113 t) \\& + 6.041196494 \cos(0.3436116965 t) \\& + 4.218106831 \cos(0.3926990818 t) \\& + 6.016474272 \cos(0.4417864670 t) \\& + 5.788666909 \cos(0.4908738522 t) \\& + 4.740925918 \cos(0.5399612374 t) \\& + 9.339745330 \cos(0.5890486226 t) \\& + 6.261527688 \cos(0.6381360078 t) \\& + 11.48941866 \cos(0.6872233931 t) \\& + 37.44296117 \cos(0.7363107783 t) \\& + 6.488675927 \cos(0.7853981635 t)\end{aligned}$$

$$-68.47655620 \cos(0.8344855487 t) + 87.31250000$$

$P_\psi := \text{plot}(\psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{"blue"}, \text{thickness} = 3) :$
 $\text{plots:-display}([P_{\text{org}}, P_\psi])$



Figur 20



Figur 21



Figur 22

Bilde til venstre er originalbildet, og det andre er konstruert av fourierrekken. Kontrasten går diagonalt fra nederste venstre hjørne til det øverste høyre hjørne. Med metode 1, som leser fra venstre til høyre på hver linje, er det ikke trivielt at kontraster går på skrått av bildet. Som et av hvordan bildet blir lest, blir det store forskjeller på høyre og venstre kant av bildet. Under er differansen mellom det originale bildet, og bildet som er konstruert med fourierrekker. Differansen er representer i en matrise og en graf.

$$\Delta := \text{Matrix}([[0, 3, 16, 0, 0, 12, -74, -63],$$

```

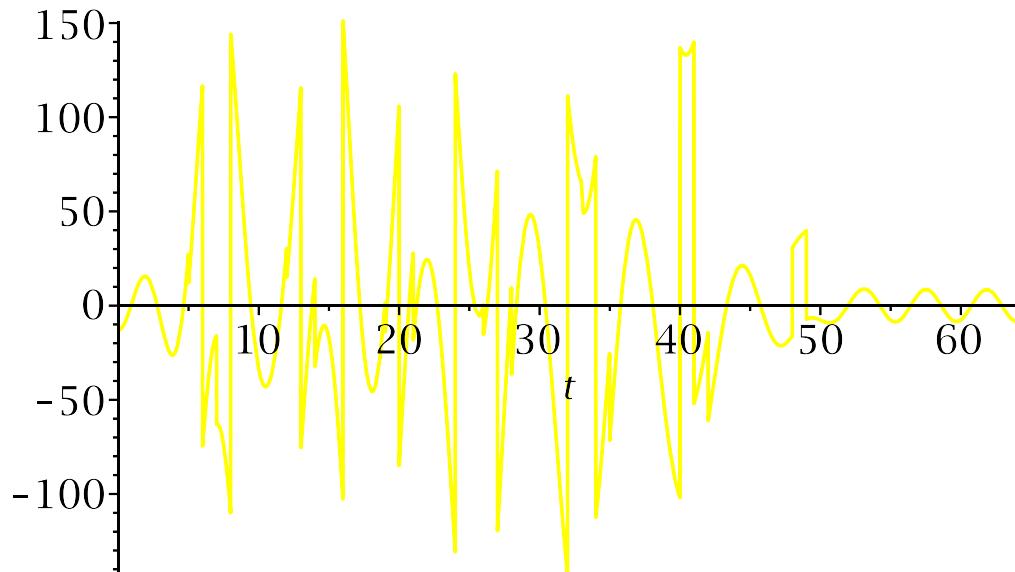
[-111, 40, 0, 0, 15, -76, -32, -16],
[-104, 23, 0, -14, -84, -18, 0, -21],
[124, 20, -15, -119, -37, 0, 0, -52],
[112, 50, -112, -71, 0, 0, 0, -60],
[-118, -51, -61, -13, 0, 0, -6, -20],
[31, -7, -7, -8, 0, 0, 0, -7],
[-5, 0, 0, -4, -7, 0, 0, 0]])

```

$$\Delta := \begin{bmatrix} 0 & 3 & 16 & 0 & 0 & 12 & -74 & -63 \\ -111 & 40 & 0 & 0 & 15 & -76 & -32 & -16 \\ -104 & 23 & 0 & -14 & -84 & -18 & 0 & -21 \\ 124 & 20 & -15 & -119 & -37 & 0 & 0 & -52 \\ 112 & 50 & -112 & -71 & 0 & 0 & 0 & -60 \\ -118 & -51 & -61 & -13 & 0 & 0 & -6 & -20 \\ 31 & -7 & -7 & -8 & 0 & 0 & 0 & -7 \\ -5 & 0 & 0 & -4 & -7 & 0 & 0 & 0 \end{bmatrix}$$

(4.2.2.1.12)

$$P_{\Delta} := \text{plot}\left(f(t) - \Psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{yellow}\right)$$



Figur 23

Metode 2

Naturlig bilde

I metode 2 brukes metoden som leser sikk-sakk. Det vil si at første linje leses fra venstre til høyre og andre linje fra høyre til venstre. Annenhver linje leses motsatt så linje tre er lik som linje én og så videre.

Under har vi dataene lest på denne måten. Her er alle pikslene fra 0 til og med 64 inkludert i en rekke. Tatt ut fra en 8x8 blokk i bildet "natur.png". Videre skal denne rekken brukes i en fourierrekke. Dette gjøres så vi kan erstatte en rekke med pikseldata om til en fourierrekke som bruker mindre data, men viser tilnærmet samme bilde. Først gраfter vi en funksjon av pikselverdiene, og kaller denne for $f(t)$.

restart;

```


$$f(t) := \text{piecewise}(0 \leq t \leq 1.0, 35, 1.0 \leq t \leq 2.0, 38, 2.0 \leq t \leq 3.0, 40, 3.0 \leq t \leq 4.0, 41, 4.0 \leq t \leq 5.0, 35, 5.0 \leq t \leq 6.0, 29, 6.0 \leq t \leq 7.0, 29, 7.0 \leq t \leq 8.0, 34, 8.0 \leq t \leq 9.0, 31, 9.0 \leq t \leq 10.0, 30, 10.0 \leq t \leq 11.0, 31, 11.0 \leq t \leq 12.0, 37, 12.0 \leq t \leq 13.0, 38, 13.0 \leq t \leq 14.0, 34, 14.0 \leq t \leq 15.0, 30, 15.0 \leq t \leq 16.0, 29, 16.0 \leq t \leq 17.0, 23, 17.0 \leq t \leq 18.0, 32, 18.0 \leq t \leq 19.0, 40, 19.0 \leq t \leq 20.0, 43, 20.0 \leq t \leq 21.0, 55, 21.0 \leq t \leq 22.0, 54, 22.0 \leq t \leq 23.0, 45, 23.0 \leq t \leq 24.0, 43, 24.0 \leq t \leq 25.0, 86, 25.0 \leq t \leq 26.0, 116, 26.0 \leq t \leq 27.0, 149, 27.0 \leq t \leq 28.0, 136, 28.0 \leq t \leq 29.0, 91, 29.0 \leq t \leq 30.0, 70, 30.0 \leq t \leq 31.0, 62, 31.0 \leq t \leq 32.0, 47, 32.0 \leq t \leq 33.0, 94, 33.0 \leq t \leq 34.0, 72, 34.0 \leq t \leq 35.0, 49, 35.0 \leq t \leq 36.0, 58, 36.0 \leq t \leq 37.0, 123, 37.0 \leq t \leq 38.0, 160, 38.0 \leq t \leq 39.0, 121, 39.0 \leq t \leq 40.0, 71, 40.0 \leq t \leq 41.0, 47, 41.0 \leq t \leq 42.0, 65, 42.0 \leq t \leq 43.0, 82, 43.0 \leq t \leq 44.0, 56, 44.0 \leq t \leq 45.0, 28, 45.0 \leq t \leq 46.0, 40, 46.0 \leq t \leq 47.0, 65, 47.0 \leq t \leq 48.0, 91, 48.0 \leq t \leq 49.0, 74, 49.0 \leq t \leq 50.0, 55, 50.0 \leq t \leq 51.0, 39, 51.0 \leq t \leq 52.0, 37, 52.0 \leq t \leq 53.0, 45, 53.0 \leq t \leq 54.0, 52, 54.0 \leq t \leq 55.0, 56, 55.0 \leq t \leq 56.0, 51, 56.0 \leq t \leq 57.0, 85, 57.0 \leq t \leq 58.0, 83, 58.0 \leq t \leq 59.0, 80, 59.0 \leq t \leq 60.0, 71, 60.0 \leq t \leq 61.0, 60, 61.0 \leq t \leq 62.0, 56, 62.0 \leq t \leq 63.0, 52, 63.0 \leq t \leq 64.0, 57) :\\ T := 2 \cdot 64;$$

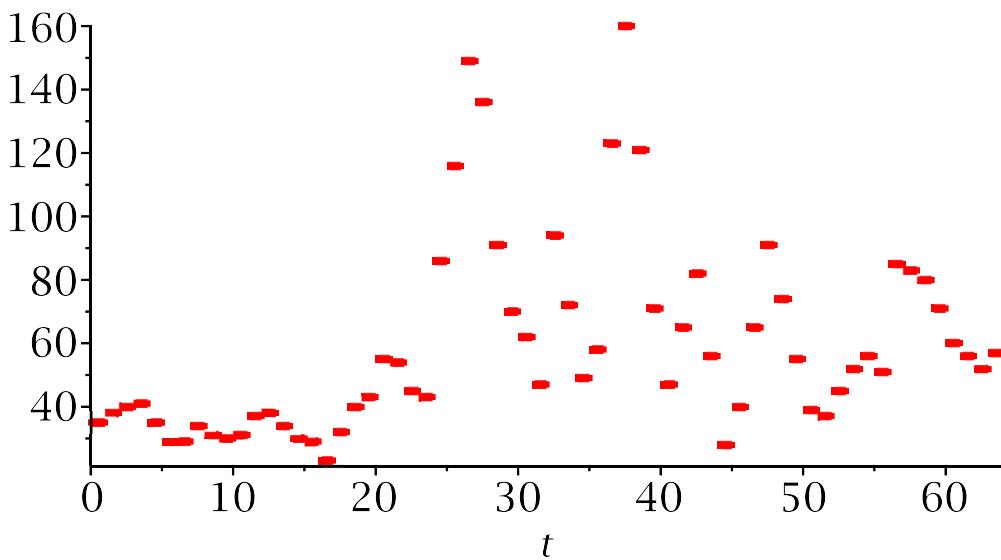

```

$$T := 128 \quad (4.2.2.2.1)$$

```


$$P_{org} := \text{plot}\left(f(t), t = 0 \dots \frac{T}{2}, \text{discont} = \text{true}, \text{symbol} = \text{"point"}, \text{color} = \text{"red"}, \text{thickness} = 3\right)$$


```



Figur 24

Videre kalkuleres a_0 og a_n som skal brukes i fourierrekken ψ .

$$a_0 := \frac{4}{T} \cdot \text{int}\left(f(t), t = 0 \dots \frac{T}{2}\right)$$

$$a_0 := 119.0000000 \quad (4.2.2.2.2)$$

$$a_n := n \rightarrow \frac{4}{T} \cdot \text{int}\left(f(t) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), t = 0 \dots \frac{T}{2}\right)$$

$$a_n := n \mapsto \frac{4 \left(\int_0^{\frac{T}{2}} f(t) \cos\left(\frac{2 n \pi t}{T}\right) dt \right)}{T} \quad (4.2.2.2.3)$$

$$\psi := (t, l) \rightarrow \frac{a_0}{2} + \text{sum}\left(a_n(n) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), n = 1 \dots l\right)$$

$$\psi := (t, l) \mapsto \frac{a_0}{2} + \left(\sum_{n=1}^l a_n(n) \cos\left(\frac{2 n \pi t}{T}\right) \right) \quad (4.2.2.2.4)$$

Når vi regner ut fourierrekken får vi følgende verdier.

$$l := 30 :$$

$$\psi_1 := \text{evalf}(\psi(t, l))$$

$$\psi_1 := 3.231925027 \cos(0.2454369261 t) + 13.99991019 \cos(0.1963495409 t) \quad (4.2.2.2.5)$$

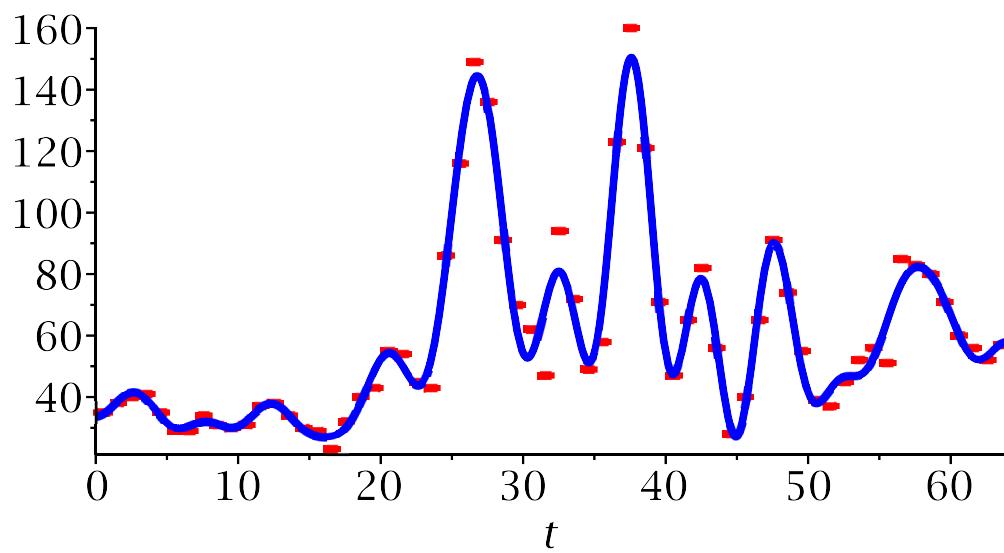
```

- 1.888083892 cos(0.1472621557 t)
- 20.62772475 cos(0.09817477044 t)
- 14.04741668 cos(0.04908738522 t)
+ 2.341256005 cos(1.472621557 t)
- 3.153960483 cos(1.423534171 t)
- 0.8232963193 cos(1.374446786 t)
+ 5.500081529 cos(1.325359401 t)
- 7.026962954 cos(1.276272016 t)
- 3.949392806 cos(1.227184630 t)
+ 11.27713633 cos(1.178097245 t)
- 3.423161035 cos(1.129009860 t)
- 8.161832629 cos(1.079922475 t)
+ 0.5307960018 cos(1.030835090 t)
+ 4.452976207 cos(0.9817477044 t)
+ 2.148735111 cos(0.9326603192 t)
- 0.03472055954 cos(0.8835729339 t)
+ 0.7948032861 cos(0.2945243113 t)
- 1.607026366 cos(0.3436116965 t)
- 9.149249800 cos(0.3926990818 t)
+ 3.416281435 cos(0.4417864670 t)
+ 9.111487361 cos(0.4908738522 t)
+ 4.190673125 cos(0.5399612374 t)
- 15.37498002 cos(0.5890486226 t)
+ 4.698111813 cos(0.6381360078 t)
+ 7.823331422 cos(0.6872233931 t)
- 2.858826811 cos(0.7363107783 t)
- 2.204861484 cos(0.7853981635 t)
- 4.876834624 cos(0.8344855487 t) + 59.50000000

```

$$P_{\psi} := \text{plot}\left(\psi_1(t), t = 0 .. \frac{T}{2}, \text{color} = \text{"blue"}, \text{thickness} = 3\right) :$$

$$\text{plots:-display}([P_{\text{org}}, P_{\psi}])$$

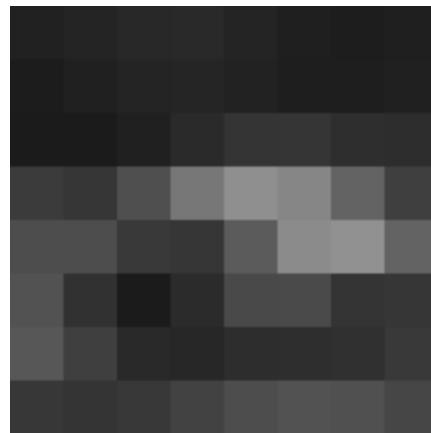


Figur 25

Grafen over har alle pikslene i x-aksen og fargeverdien i y-aksen. De røde strekene representerer presis verdi av pikslene i bildet. Den blå grafen er fourierrekken som er en etterligning av bildet sine faktiske verdier.



Figur 26



Figur 27

Bilde til venstre er det originale bilde, og bilde til høyre er konstruert av fourierrekken. Under ser vi Δ -matrise som forteller oss om forskjellen i pikselverdier.

```

$$\Delta := \text{Matrix}([ [2, 2, 0, 0, -1, -2, 0, 3],$$


$$[1, -2, -2, 1, 3, 1, 0, -1],$$


$$[-4, 5, 8, 1, 3, 1, -1, -2],$$


$$[-12, 8, -9, -28, -7, 15, 17, 23],$$


$$[17, -5, -8, 4, 32, 21, -24, -28],$$


$$[9, 16, 13, -15, -17, 8, 13, -7],$$

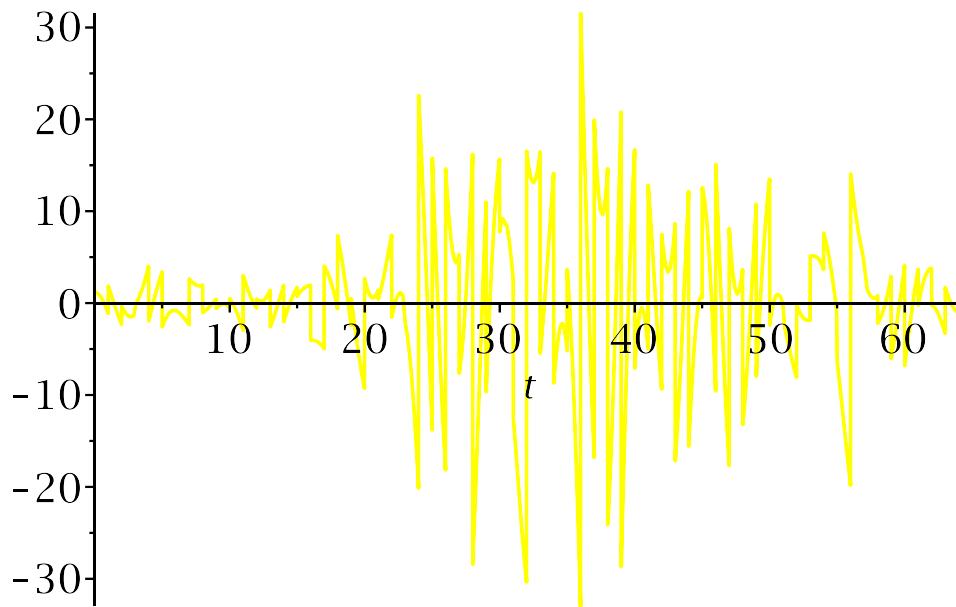

$$[-13, -8, -2, -2, 0, 6, 8, -6],$$


$$[2, 0, 0, -6, -6, -2, 3, 15]])$$

```

$$\Delta := \begin{bmatrix} 2 & 2 & 0 & 0 & -1 & -2 & 0 & 3 \\ 1 & -2 & -2 & 1 & 3 & 1 & 0 & -1 \\ -4 & 5 & 8 & 1 & 3 & 1 & -1 & -2 \\ -12 & 8 & -9 & -28 & -7 & 15 & 17 & 23 \\ 17 & -5 & -8 & 4 & 32 & 21 & -24 & -28 \\ 9 & 16 & 13 & -15 & -17 & 8 & 13 & -7 \\ -13 & -8 & -2 & -2 & 0 & 6 & 8 & -6 \\ 2 & 0 & 0 & -6 & -6 & -2 & 3 & 15 \end{bmatrix} \quad (4.2.2.2.6)$$

$$P_{\Delta} := \text{plot}\left(f(t) - \psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{yellow}\right)$$



Figur 28

Kunstig bilde

Under har vi dataene i $f(x)$ som en rekke. Her er alle pikslene fra 0 til og med 64 inkludert i en rekke. Under ser vi et klassis tilfelle der vi trenger .

restart;

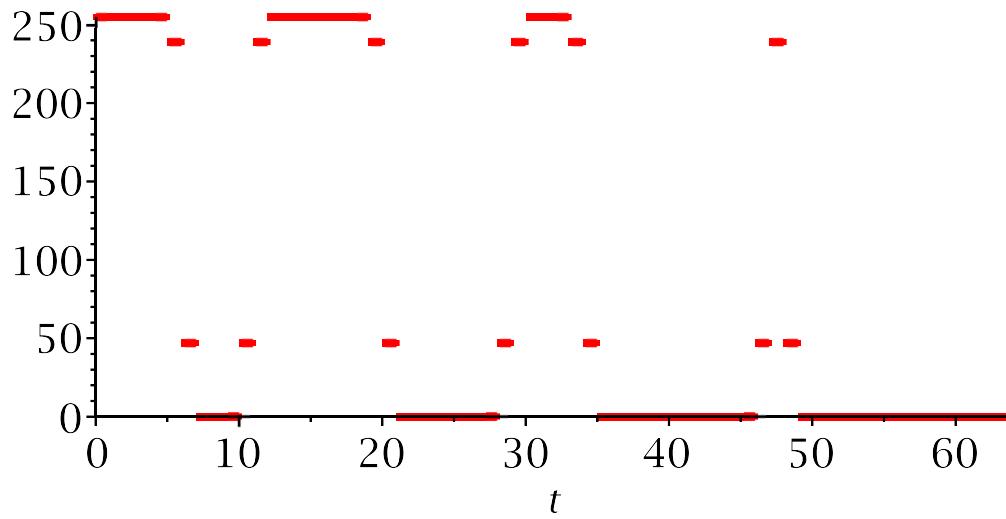
$$\begin{aligned} f(t) := & \text{piecewise}(0 \leq t \leq 1.0, 255, 1.0 \leq t \leq 2.0, 255, 2.0 \leq t \leq 3.0, 255, 3.0 \\ & \leq t \leq 4.0, 255, 4.0 \leq t \leq 5.0, 255, 5.0 \leq t \leq 6.0, 239, 6.0 \leq t \leq 7.0, 47, 7.0 \\ & \leq t \leq 8.0, 0, 8.0 \leq t \leq 9.0, 0, 9.0 \leq t \leq 10.0, 0, 10.0 \leq t \leq 11.0, 47, 11.0 \\ & \leq t \leq 12.0, 239, 12.0 \leq t \leq 13.0, 255, 13.0 \leq t \leq 14.0, 255, 14.0 \leq t \\ & \leq 15.0, 255, 15.0 \leq t \leq 16.0, 255, 16.0 \leq t \leq 17.0, 255, 17.0 \leq t \leq 18.0, \\ & 255, 18.0 \leq t \leq 19.0, 255, 19.0 \leq t \leq 20.0, 239, 20.0 \leq t \leq 21.0, 47, 21.0 \\ & \leq t \leq 22.0, 0, 22.0 \leq t \leq 23.0, 0, 23.0 \leq t \leq 24.0, 0, 24.0 \leq t \leq 25.0, 0, \\ & 25.0 \leq t \leq 26.0, 0, 26.0 \leq t \leq 27.0, 0, 27.0 \leq t \leq 28.0, 0, 28.0 \leq t \leq 29.0, \end{aligned}$$

$$47, 29.0 \leq t \leq 30.0, 239, 30.0 \leq t \leq 31.0, 255, 31.0 \leq t \leq 32.0, 255, 32.0 \\ \leq t \leq 33.0, 255, 33.0 \leq t \leq 34.0, 239, 34.0 \leq t \leq 35.0, 47, 35.0 \leq t \leq 36.0, \\ 0, 36.0 \leq t \leq 37.0, 0, 37.0 \leq t \leq 38.0, 0, 38.0 \leq t \leq 39.0, 0, 39.0 \leq t \\ \leq 40.0, 0, 40.0 \leq t \leq 41.0, 0, 41.0 \leq t \leq 42.0, 0, 42.0 \leq t \leq 43.0, 0, 43.0 \\ \leq t \leq 44.0, 0, 44.0 \leq t \leq 45.0, 0, 45.0 \leq t \leq 46.0, 0, 46.0 \leq t \leq 47.0, 47, \\ 47.0 \leq t \leq 48.0, 239, 48.0 \leq t \leq 49.0, 47, 49.0 \leq t \leq 50.0, 0, 50.0 \leq t \\ \leq 51.0, 0, 51.0 \leq t \leq 52.0, 0, 52.0 \leq t \leq 53.0, 0, 53.0 \leq t \leq 54.0, 0, 54.0 \\ \leq t \leq 55.0, 0, 55.0 \leq t \leq 56.0, 0, 56.0 \leq t \leq 57.0, 0, 57.0 \leq t \leq 58.0, 0, \\ 58.0 \leq t \leq 59.0, 0, 59.0 \leq t \leq 60.0, 0, 60.0 \leq t \leq 61.0, 0, 61.0 \leq t \leq 62.0, \\ 0, 62.0 \leq t \leq 63.0, 0, 63.0 \leq t \leq 64.0, 0) : T := 2 \cdot 64;$$

$$T := 128 \quad (4.2.2.2.7)$$

Videre skal denne rekken brukes i en fourierrekke. Dette gjøres så vi kan erstatte en rekke med pikseldata om til en fourierrekke som bruker mindre data, men viser tilnærmet samme bilde.

$$P_{org} := \text{plot}\left(f(t), t = 0 .. \frac{T}{2}, \text{discont} = \text{true}, \text{symbol} = \text{"point"}, \text{color} = \text{"red"}, \text{thickness} = 3\right)$$



Figur 29

Videre kalkuleres a_0 og a_n , som skal brukes i fourierrekken ψ .

$$a_0 := \frac{4}{T} \cdot \text{int}\left(f(t), t = 0 \dots \frac{T}{2}\right)$$

$$a_0 := 174.6250000 \quad (4.2.2.2.8)$$

$$a_n := n \rightarrow \frac{4}{T} \cdot \text{int}\left(f(t) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), t = 0 \dots \frac{T}{2}\right)$$

$$a_n := n \mapsto \frac{4 \left(\int_0^{\frac{T}{2}} f(t) \cos\left(\frac{2n\pi t}{T}\right) dt \right)}{T} \quad (4.2.2.2.9)$$

$$\begin{aligned} \psi := (t, l) \rightarrow & \frac{a_0}{2} + \text{sum}\left(a_n(n) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), n = 1 \dots l\right) \\ \psi := (t, l) \mapsto & \frac{a_0}{2} + \left(\sum_{n=1}^l a_n(n) \cos\left(\frac{2n\pi t}{T}\right) \right) \end{aligned} \quad (4.2.2.2.10)$$

Når vi regner ut fourierrekken får vi følgende verdier.

$l := 30 :$

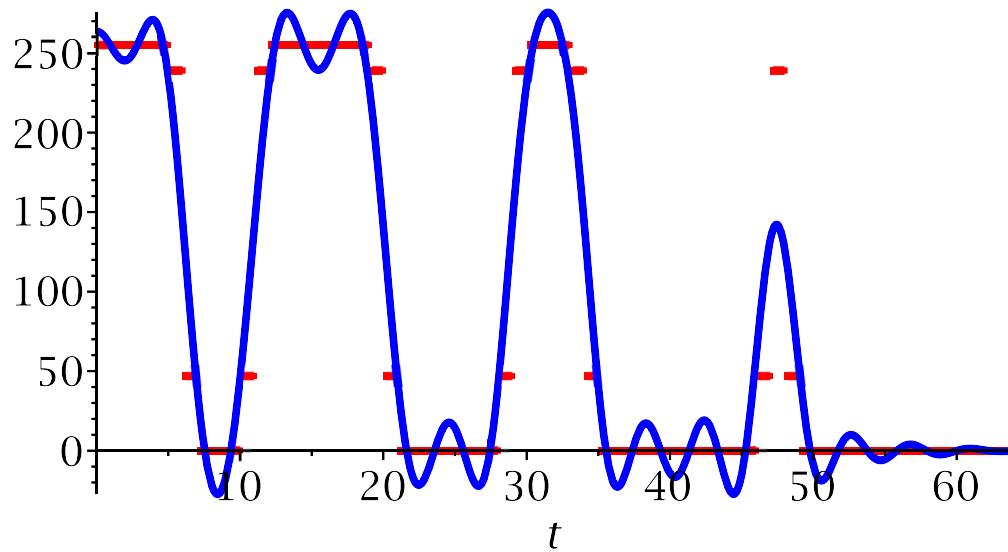
$\psi_1 := \text{evalf}(\psi(t, l))$

$$\psi_1 := -27.53698704 \cos(0.7363107783 t) \quad (4.2.2.2.11)$$

$$\begin{aligned} & + 4.170214808 \cos(0.9326603192 t) \\ & + 9.868053255 \cos(0.9817477044 t) \\ & + 6.488675824 \cos(0.7853981635 t) \\ & - 2.032607729 \cos(1.423534171 t) \\ & + 7.848280087 \cos(1.472621557 t) \\ & - 16.34303865 \cos(0.8344855487 t) \\ & - 26.89808732 \cos(0.8835729339 t) \\ & + 18.53256178 \cos(1.030835090 t) \\ & + 9.288161938 \cos(1.079922475 t) \\ & + 4.725676731 \cos(1.178097245 t) \\ & - 8.128108823 \cos(1.129009860 t) \\ & - 2.928708739 \cos(1.227184630 t) \\ & - 0.2758849377 \cos(1.276272016 t) \\ & + 14.43287479 \cos(1.325359401 t) \\ & - 5.178799326 \cos(1.374446786 t) \\ & + 94.76072589 \cos(0.04908738522 t) \\ & + 7.59868663 \cos(0.09817477044 t) \\ & + 2.820123665 \cos(0.1472621557 t) \\ & + 3.628188084 \cos(0.1963495409 t) \\ & - 2.83351184 \cos(0.2454369261 t) \\ & - 17.08111620 \cos(0.2945243113 t) \\ & + 31.59389714 \cos(0.3436116965 t) \\ & + 94.78152004 \cos(0.3926990818 t) \end{aligned}$$

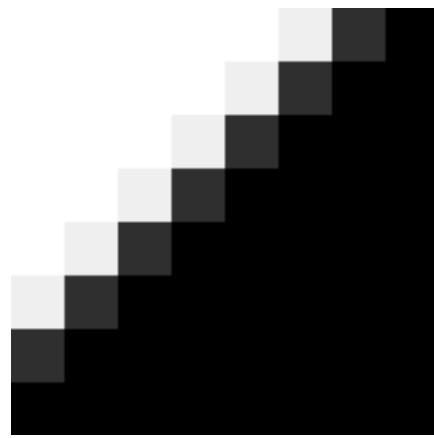
$$\begin{aligned}
 & + 35.73977026 \cos(0.4417864670 t) \\
 & - 23.77173001 \cos(0.4908738522 t) \\
 & - 9.727022590 \cos(0.5399612374 t) \\
 & - 0.4429782361 \cos(0.5890486226 t) \\
 & - 26.28139677 \cos(0.6872233931 t) \\
 & - 0.5581053977 \cos(0.6381360078 t) + 87.31250000
 \end{aligned}$$

$$P_{\psi} := \text{plot}\left(\psi_1(t), t = 0 .. \frac{T}{2}, \text{color} = \text{"blue"}, \text{thickness} = 3\right);$$

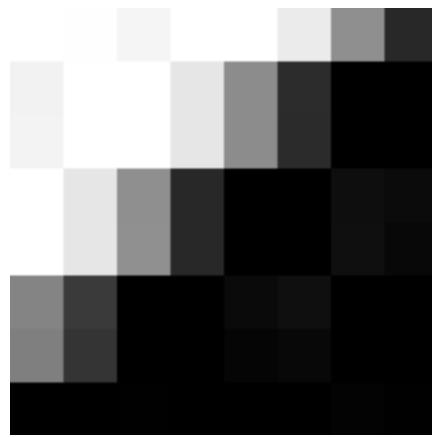
$$\text{plots:-display}([P_{\text{org}}, P_{\psi}])$$


Figur 30

Grafen over har alle pikslene i x-aksen og fargeverdien i y-aksen. De røde strekene representerer presis verdi av pikslene i bildet. Den blå grafen er fourierrekken som er en etterligning av bildet sine faktiske verdier.



Figur 31



Figur 32

$$\Delta := \text{Matrix}([[0, 1, 10, 0, 0, 4, -96, -40],$$

$$\begin{aligned}
 & [13, 0, 0, 25, 99, 3, 0, 0], \\
 & [12, 0, 0, 9, -93, -43, 0, 0], \\
 & [0, 25, 96, 7, 0, 0, -14, -11], \\
 & [0, 9, -97, -40, 0, 0, -15, -8], \\
 & [107, -11, 0, 0, -10, -15, 0, 0], \\
 & [-80, -52, 0, 0, -5, -8, 0, 0], \\
 & [0, 0, -1, 0, 0, 0, -3, -1]] \\
) \\
 \Delta := & \begin{bmatrix} 0 & 1 & 10 & 0 & 0 & 4 & -96 & -40 \\ 13 & 0 & 0 & 25 & 99 & 3 & 0 & 0 \\ 12 & 0 & 0 & 9 & -93 & -43 & 0 & 0 \\ 0 & 25 & 96 & 7 & 0 & 0 & -14 & -11 \\ 0 & 9 & -97 & -40 & 0 & 0 & -15 & -8 \\ 107 & -11 & 0 & 0 & -10 & -15 & 0 & 0 \\ -80 & -52 & 0 & 0 & -5 & -8 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -3 & -1 \end{bmatrix} \quad (4.2.2.2.12)
 \end{aligned}$$

Metode 3

Naturlig bilde

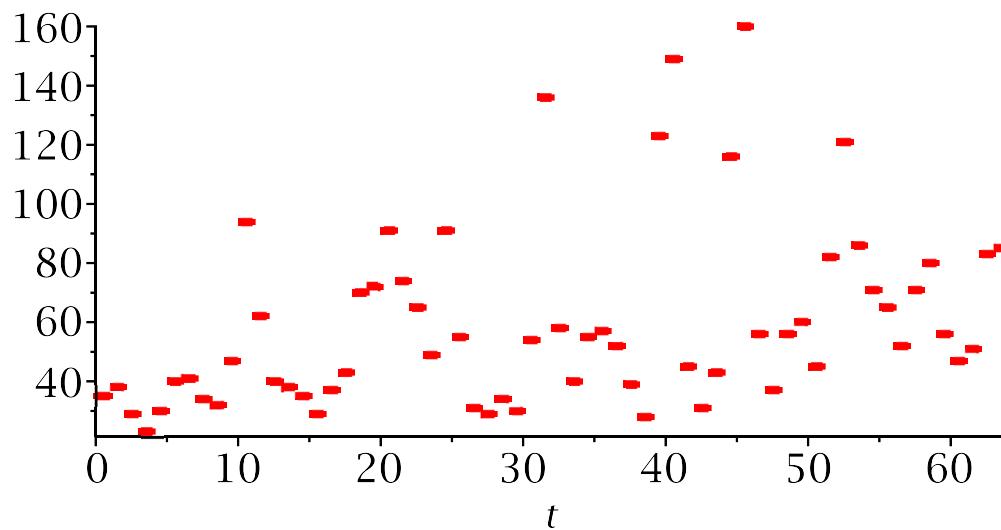
Under er den genererte rekken som ble laget av å bruke metode 3. Som nevnt i fremgangsmåten ble et python-skript brukt for å generere denne funksjonen (Se vedlegg 1). Etter vi har regnet ut $f(t)$, så lager vi en periodisk, like fourierrekke med 30 ledd fra denne funksjonen.

restart;

```
f(t) := piecewise(0 ≤ t ≤ 1.0, 35, 1.0 ≤ t ≤ 2.0, 38, 2.0 ≤ t ≤ 3.0, 29, 3.0 ≤ t ≤ 4.0, 23,
4.0 ≤ t ≤ 5.0, 30, 5.0 ≤ t ≤ 6.0, 40, 6.0 ≤ t ≤ 7.0, 41, 7.0 ≤ t ≤ 8.0, 34, 8.0 ≤ t ≤ 9.0,
32, 9.0 ≤ t ≤ 10.0, 47, 10.0 ≤ t ≤ 11.0, 94, 11.0 ≤ t ≤ 12.0, 62, 12.0 ≤ t ≤ 13.0, 40,
13.0 ≤ t ≤ 14.0, 38, 14.0 ≤ t ≤ 15.0, 35, 15.0 ≤ t ≤ 16.0, 29, 16.0 ≤ t ≤ 17.0, 37,
17.0 ≤ t ≤ 18.0, 43, 18.0 ≤ t ≤ 19.0, 70, 19.0 ≤ t ≤ 20.0, 72, 20.0 ≤ t ≤ 21.0, 91,
21.0 ≤ t ≤ 22.0, 74, 22.0 ≤ t ≤ 23.0, 65, 23.0 ≤ t ≤ 24.0, 49, 24.0 ≤ t ≤ 25.0, 91,
25.0 ≤ t ≤ 26.0, 55, 26.0 ≤ t ≤ 27.0, 31, 27.0 ≤ t ≤ 28.0, 29, 28.0 ≤ t ≤ 29.0, 34,
29.0 ≤ t ≤ 30.0, 30, 30.0 ≤ t ≤ 31.0, 54, 31.0 ≤ t ≤ 32.0, 136, 32.0 ≤ t ≤ 33.0, 58,
33.0 ≤ t ≤ 34.0, 40, 34.0 ≤ t ≤ 35.0, 55, 35.0 ≤ t ≤ 36.0, 57, 36.0 ≤ t ≤ 37.0, 52,
37.0 ≤ t ≤ 38.0, 39, 38.0 ≤ t ≤ 39.0, 28, 39.0 ≤ t ≤ 40.0, 123, 40.0 ≤ t ≤ 41.0, 149,
41.0 ≤ t ≤ 42.0, 45, 42.0 ≤ t ≤ 43.0, 31, 43.0 ≤ t ≤ 44.0, 43, 44.0 ≤ t ≤ 45.0, 116,
45.0 ≤ t ≤ 46.0, 160, 46.0 ≤ t ≤ 47.0, 56, 47.0 ≤ t ≤ 48.0, 37, 48.0 ≤ t ≤ 49.0, 56,
49.0 ≤ t ≤ 50.0, 60, 50.0 ≤ t ≤ 51.0, 45, 51.0 ≤ t ≤ 52.0, 82, 52.0 ≤ t ≤ 53.0, 121,
53.0 ≤ t ≤ 54.0, 86, 54.0 ≤ t ≤ 55.0, 71, 55.0 ≤ t ≤ 56.0, 65, 56.0 ≤ t ≤ 57.0, 52,
57.0 ≤ t ≤ 58.0, 71, 58.0 ≤ t ≤ 59.0, 80, 59.0 ≤ t ≤ 60.0, 56, 60.0 ≤ t ≤ 61.0, 47,
61.0 ≤ t ≤ 62.0, 51, 62.0 ≤ t ≤ 63.0, 83, 63.0 ≤ t ≤ 64.0, 85) : T := 2·64;
```

$$T := 128 \quad (4.2.2.3.1)$$

```
Porg := plot(f(t), t = 0 ..  $\frac{T}{2}$ , discontinuity = true, symbol = "point", color = "red", thickness = 3)
```



Figur 33

$$a_n := n \rightarrow \frac{4}{T} \cdot \text{int}\left(f(t) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \pi}{T}\right) \cdot t\right), t = 0 \dots \frac{T}{2}\right)$$

$$a_n := n \mapsto \frac{4}{T} \left(\int_0^{\frac{T}{2}} f(t) \cos\left(\frac{2n\pi t}{T}\right) dt \right) \quad (4.2.2.3.2)$$

$$a_0 := \frac{4}{T} \cdot \text{int}\left(f(t), t = 0 \dots \frac{T}{2}\right)$$

$$a_0 := 119.0000000 \quad (4.2.2.3.3)$$

$$\psi := (t, l) \rightarrow \frac{a_0}{2} + \text{sum}\left(a_n(n) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), n = 1 \dots l\right)$$

$$\psi := (t, l) \mapsto \frac{a_0}{2} + \left(\sum_{n=1}^l a_n(n) \cos\left(\frac{2n\pi t}{T}\right) \right) \quad (4.2.2.3.4)$$

$$l := 30 \quad l := 30 \quad (4.2.2.3.5)$$

$$\Psi_1 := \text{evalf}(\psi(t, l))$$

$$\Psi_1 := -12.01165081 \cos(0.8835729339 t) \quad (4.2.2.3.6)$$

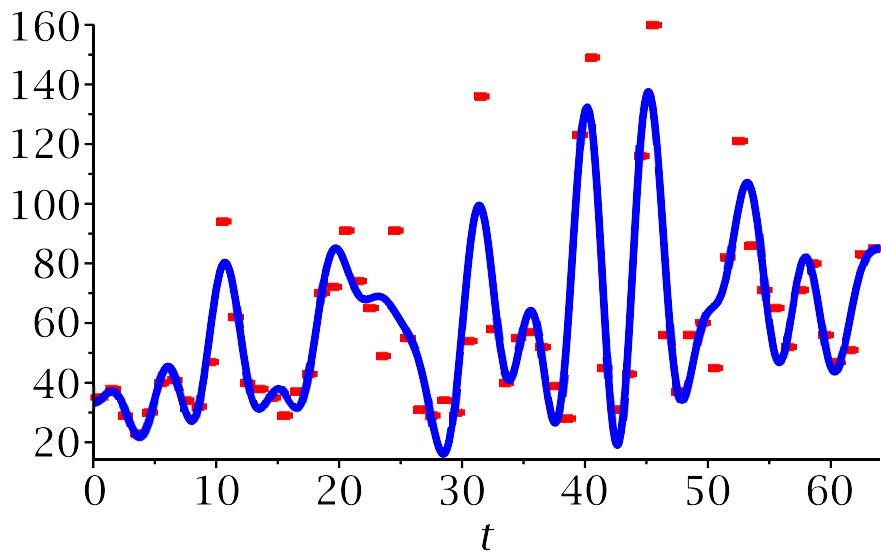
$$\begin{aligned}
&+ 0.893716799 \cos(0.6381360078 t) \\
&- 3.046817470 \cos(0.4908738522 t) \\
&+ 0.0539159494 \cos(0.6872233931 t) \\
&- 3.058664657 \cos(0.4417864670 t) \\
&- 4.103695130 \cos(0.3926990818 t) \\
&+ 3.379156049 \cos(0.2945243113 t) \\
&+ 0.048075103 \cos(0.3436116965 t) \\
&- 15.95154434 \cos(0.04908738522 t) \\
&- 4.920585166 \cos(0.09817477044 t) \\
&- 0.3853117477 \cos(0.1472621557 t) \\
&- 6.875958274 \cos(0.1963495409 t) \\
&- 0.9867164302 \cos(0.2454369261 t) \\
&+ 10.45961345 \cos(0.9817477044 t) \\
&- 7.674725385 \cos(1.030835090 t) \\
&- 4.034061058 \cos(0.7363107783 t) \\
&+ 2.481377689 \cos(1.374446786 t) \\
&+ 5.300766656 \cos(1.423534171 t) \\
&- 12.72286853 \cos(1.472621557 t) \\
&+ 1.769126037 \cos(0.7853981635 t) \\
&+ 7.295606930 \cos(1.079922475 t) \\
&+ 0.2288648915 \cos(0.5399612374 t) \\
&+ 2.491166030 \cos(0.8344855487 t)
\end{aligned}$$

$$\begin{aligned}
 & -1.168868866 \cos(1.129009860 t) \\
 & -2.783162120 \cos(1.178097245 t) \\
 & +8.410422908 \cos(1.227184630 t) \\
 & +6.370079699 \cos(1.276272016 t) \\
 & -10.70114578 \cos(1.325359401 t) \\
 & +14.42273738 \cos(0.5890486226 t) \\
 & +0.8251672269 \cos(0.9326603192 t) + 59.50000000
 \end{aligned}$$

```

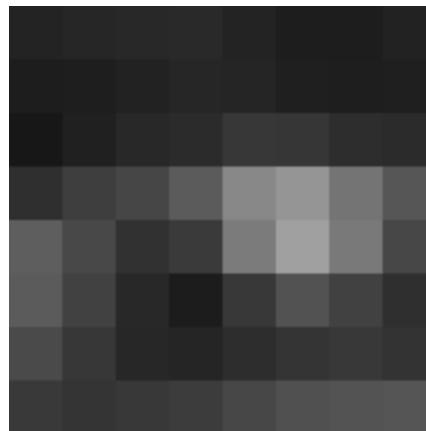
 $P_{\psi} := \text{plot}\left(\psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{"blue"}, \text{thickness} = 3\right);$ 
plots:-display([P_{org}, P_{\psi}])

```



Figur 34

Deretter genererer vi et bilde basert på fourierrekken som vi har regnet ut.



Figur 35



Figur 36

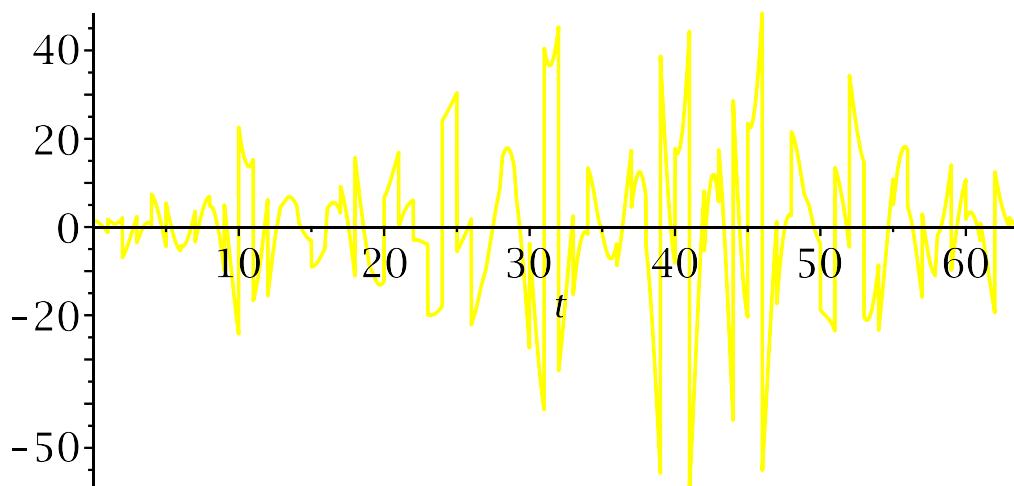
Her ser vi orginalbildet (Figur 35) og det genererte fourierbildet (Figur 36). I fourierbildet har vi noen diagonale hakkete linjer, dette er et resultat av metoden vi

har brukt for å generere $f(t)$. Selvom det er noen diagonale linjer, er dette bare 8x8 piksler. Når vi legger dette inn i bildet igjen vil vi se at det nesten ikke er synlig. I denne fourierrekken er det 30 ledd, og dermed trenger vi bare å lagre 30 (alle a_n -verdier) verdier istedenfor 64 (en verdi for hver piksel). Under er det en analyse av forskjellen Δ mellom pikseldataen til de to bildene.

$$\Delta := \text{Matrix}([[2, 2, 6, -4, 2, -9, -9, 15], [-6, 8, -3, 5, 4, -22, 9, -5], [-3, 5, -15, 10, -5, -3, -59, 18], [6, -16, 16, 25, 41, 18, 29, -20], [23, -6, -19, -32, 39, 24, 35, -23], [7, -2, -15, -4, -55, 14, 6, 2], [0, 14, 5, -17, -18, 5, -9, 1], [-1, -8, 22, 10, 4, -2, 13, 3]])$$

$$\Delta := \begin{bmatrix} 2 & 2 & 6 & -4 & 2 & -9 & -9 & 15 \\ -6 & 8 & -3 & 5 & 4 & -22 & 9 & -5 \\ -3 & 5 & -15 & 10 & -5 & -3 & -59 & 18 \\ 6 & -16 & 16 & 25 & 41 & 18 & 29 & -20 \\ 23 & -6 & -19 & -32 & 39 & 24 & 35 & -23 \\ 7 & -2 & -15 & -4 & -55 & 14 & 6 & 2 \\ 0 & 14 & 5 & -17 & -18 & 5 & -9 & 1 \\ -1 & -8 & 22 & 10 & 4 & -2 & 13 & 3 \end{bmatrix} \quad (4.2.2.3.7)$$

Vi ser her at det er store forskjeller mellom pikselverdier. Dette er det samme fenomenet vi ser i grafen over. Hvis vi grafer forskjellen mellom $f(t)$ og $\psi_1(t)$ får vi en enda bedre visualisering av forskjellene.

$$P_\Delta := \text{plot}\left(f(t) - \psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{"yellow"}\right)$$


Figur 37

Som vi nå også ser fra P_Δ så er forskjellen mellom orginalbildet og fourierrekken ganske stor. Vi ser utifra grafen P_Δ at det er et avvik på rundt 60 på det meste. Dette forteller oss at bildet generert fra fourierrekken mest sannsynlig har steder der den ikke representerer bildet på en god måte, men dette er ikke alltid tilfellet. Som vi ser

utifra bildene over (Se figur 37) så er bildet fortsatt relativt bra selv om det største avviket mellom $f(t)$ og $\psi_1(t)$ er stort

Kunstig bilde

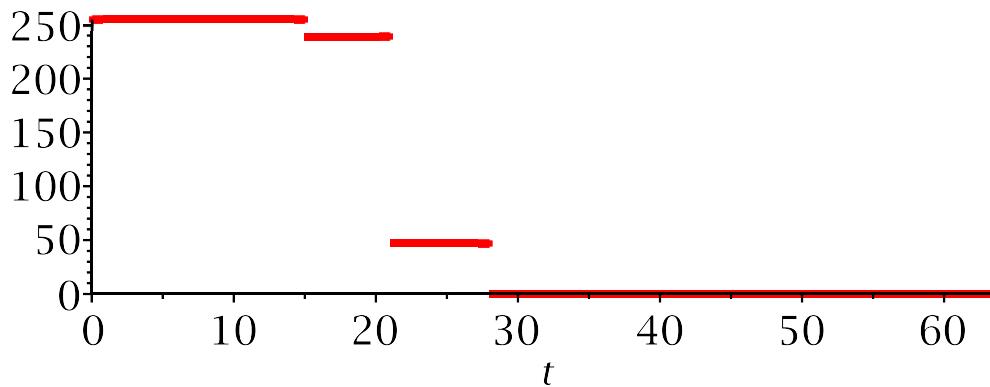
Under er den genererte rekken som ble laget av å bruke metode 3. Som nevnt i fremgangsmåten ble et python-skript brukt for å generere denne funksjonen (Se vedlegg 1). Etter vi har regnet ut $f(t)$, så lager vi en periodisk, like fourierrekke med 30 ledd fra denne funksjonen.

restart;

```
f(t) := piecewise(0 ≤ t ≤ 1.0, 255, 1.0 ≤ t ≤ 2.0, 255, 2.0 ≤ t ≤ 3.0, 255, 3.0 ≤ t ≤ 4.0,
255, 4.0 ≤ t ≤ 5.0, 255, 5.0 ≤ t ≤ 6.0, 255, 6.0 ≤ t ≤ 7.0, 255, 7.0 ≤ t ≤ 8.0, 255,
8.0 ≤ t ≤ 9.0, 255, 9.0 ≤ t ≤ 10.0, 255, 10.0 ≤ t ≤ 11.0, 255, 11.0 ≤ t ≤ 12.0, 255,
12.0 ≤ t ≤ 13.0, 255, 13.0 ≤ t ≤ 14.0, 255, 14.0 ≤ t ≤ 15.0, 255, 15.0 ≤ t ≤ 16.0,
239, 16.0 ≤ t ≤ 17.0, 239, 17.0 ≤ t ≤ 18.0, 239, 18.0 ≤ t ≤ 19.0, 239, 19.0 ≤ t
≤ 20.0, 239, 20.0 ≤ t ≤ 21.0, 239, 21.0 ≤ t ≤ 22.0, 47, 22.0 ≤ t ≤ 23.0, 47, 23.0 ≤ t
≤ 24.0, 47, 24.0 ≤ t ≤ 25.0, 47, 25.0 ≤ t ≤ 26.0, 47, 26.0 ≤ t ≤ 27.0, 47, 27.0 ≤ t
≤ 28.0, 47, 28.0 ≤ t ≤ 29.0, 0, 29.0 ≤ t ≤ 30.0, 0, 30.0 ≤ t ≤ 31.0, 0, 31.0 ≤ t
≤ 32.0, 0, 32.0 ≤ t ≤ 33.0, 0, 33.0 ≤ t ≤ 34.0, 0, 34.0 ≤ t ≤ 35.0, 0, 35.0 ≤ t ≤ 36.0,
0, 36.0 ≤ t ≤ 37.0, 0, 37.0 ≤ t ≤ 38.0, 0, 38.0 ≤ t ≤ 39.0, 0, 39.0 ≤ t ≤ 40.0, 0, 40.0
≤ t ≤ 41.0, 0, 41.0 ≤ t ≤ 42.0, 0, 42.0 ≤ t ≤ 43.0, 0, 43.0 ≤ t ≤ 44.0, 0, 44.0 ≤ t
≤ 45.0, 0, 45.0 ≤ t ≤ 46.0, 0, 46.0 ≤ t ≤ 47.0, 0, 47.0 ≤ t ≤ 48.0, 0, 48.0 ≤ t ≤ 49.0,
0, 49.0 ≤ t ≤ 50.0, 0, 50.0 ≤ t ≤ 51.0, 0, 51.0 ≤ t ≤ 52.0, 0, 52.0 ≤ t ≤ 53.0, 0, 53.0
≤ t ≤ 54.0, 0, 54.0 ≤ t ≤ 55.0, 0, 55.0 ≤ t ≤ 56.0, 0, 56.0 ≤ t ≤ 57.0, 0, 57.0 ≤ t
≤ 58.0, 0, 58.0 ≤ t ≤ 59.0, 0, 59.0 ≤ t ≤ 60.0, 0, 60.0 ≤ t ≤ 61.0, 0, 61.0 ≤ t ≤ 62.0,
0, 62.0 ≤ t ≤ 63.0, 0, 63.0 ≤ t ≤ 64.0, 0) : T := 2·64;
```

$$T := 128 \quad (4.2.2.3.8)$$

$P_{org} := \text{plot}\left(f(t), t = 0 .. \frac{T}{2}, \text{discont} = \text{true}, \text{symbol} = \text{"point"}, \text{color} = \text{"red"}, \text{thickness} = 3\right)$



Figur 38

$$\begin{aligned} a_n &:= n \rightarrow \frac{4}{T} \cdot \text{int}\left(f(t) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), t = 0 \dots \frac{T}{2}\right) \\ a_n &:= n \mapsto \frac{4 \left(\int_0^{\frac{T}{2}} f(t) \cos\left(\frac{2 n \pi t}{T}\right) dt \right)}{T} \end{aligned} \quad (4.2.2.3.9)$$

$$a_0 := \frac{4}{T} \cdot \text{int}\left(f(t), t = 0 \dots \frac{T}{2}\right)$$

$$a_0 := 174.6250000 \quad (4.2.2.3.10)$$

$$\psi := (t, l) \rightarrow \frac{a_0}{2} + \text{sum}\left(a_n(n) \cdot \cos\left(n \cdot \left(\frac{2 \cdot \text{Pi}}{T}\right) \cdot t\right), n = 1 \dots l\right)$$

$$\psi := (t, l) \mapsto \frac{a_0}{2} + \left(\sum_{n=1}^l a_n(n) \cos\left(\frac{2 n \pi t}{T}\right) \right) \quad (4.2.2.3.11)$$

$$l := 30$$

$$l := 30 \quad (4.2.2.3.12)$$

$$\psi_1 := \text{evalf}(\psi(t, l))$$

$$\psi_1 := -5.852056005 \cos(0.7853981635 t) \quad (4.2.2.3.13)$$

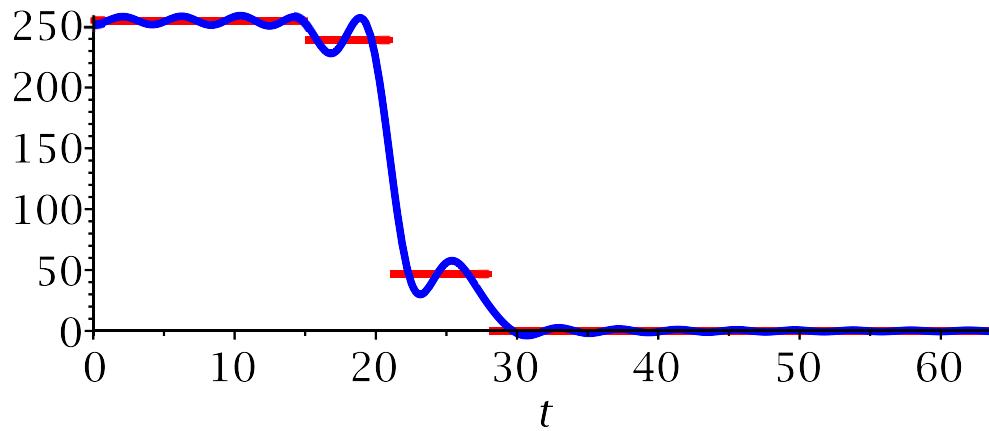
$$\begin{aligned}
&+ 1.725469068 \cos(0.4417864670 t) \\
&- 5.785914227 \cos(0.4908738522 t) \\
&- 8.052902122 \cos(0.5399612374 t) \\
&- 3.278710526 \cos(0.5890486226 t) \\
&- 30.20040468 \cos(0.1963495409 t) \\
&- 19.82176616 \cos(0.2454369261 t) \\
&+ 0.985911939 \cos(0.2945243113 t) \\
&+ 11.87593921 \cos(0.3436116965 t) \\
&+ 9.888450648 \cos(0.3926990818 t) \\
&+ 141.0276691 \cos(0.04908738522 t) \\
&+ 64.69265187 \cos(0.09817477044 t) \\
&- 3.566504710 \cos(0.1472621557 t) \\
&+ 3.258151057 \cos(0.7363107783 t) \\
&+ 7.475453779 \cos(0.9817477044 t) \\
&+ 1.287146206 \cos(1.030835090 t) \\
&+ 4.938007688 \cos(0.6381360078 t) \\
&+ 8.610305090 \cos(0.6872233931 t) \\
&+ 2.971808007 \cos(1.227184630 t) \\
&- 8.730228779 \cos(0.8344855487 t) \\
&- 2.248346107 \cos(0.8835729339 t) \\
&+ 6.159973590 \cos(0.9326603192 t) \\
&- 1.312866830 \cos(1.374446786 t) \\
&- 3.142672782 \cos(1.423534171 t) \\
&- 4.999438596 \cos(1.079922475 t) \\
&- 5.420052033 \cos(1.129009860 t) \\
&- 1.094383682 \cos(1.178097245 t) \\
&- 2.335600308 \cos(1.472621557 t)
\end{aligned}$$

$$\begin{aligned}
 & + 3.729066048 \cos(1.276272016 t) \\
 & + 1.643479624 \cos(1.325359401 t) + 87.31250000
 \end{aligned}$$

```

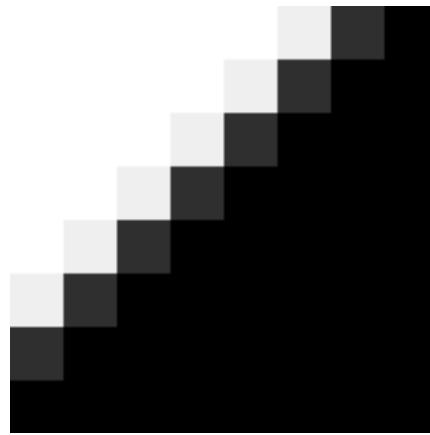
 $P_{\psi} := \text{plot}\left(\psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{"blue"}, \text{thickness} = 3\right);$ 
 $Pr_{\psi} := \text{plot}\left(\left[\text{seq}(\psi(t, j), j = 0 \dots (I - 1))\right], t = 0 \dots T, \text{color} = \text{"green"}, \text{linestyle} = \text{"dash"}\right);$ 
 $\text{plots:-display}\left(\left[P_{\text{org}}, P_{\psi}\right]\right)$ 

```

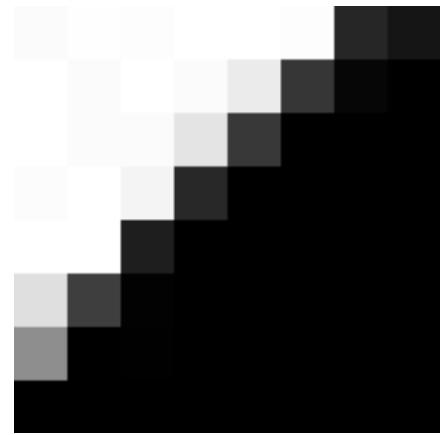


Figur 39

Deretter genererer vi et bilde utifra fourierrekken som vi har regnet ut. Dette gjør vi som nevnt ved å legge funksjonsuttrykket vi fikk av $\psi_1(t)$ inn i skriptet (Se vedlegg 1).



Figur 40



Figur 41

Her ser vi orginalbildet og det genererte fourierbildet. I dette bildet passer fourierrekken utmerket. Dette er fordi den skarpe linjen mellom hvitt og sort er parallel med metode 3. Vi ser også i grafen (Se figur 39) at dette er tilfellet ettersom det bare er en bratt del i figuren. I denne fourierrekken er det 30 ledd, og dermed trenger vi bare å lagre 30 (alle a_n -verdier) verdier istedenfor 64 (en verdi for hver piksel). Under er det en analyse av forskjellen Δ mellom pikseldataen til de to bildene.

 Δ

(4.2.2.3.14)

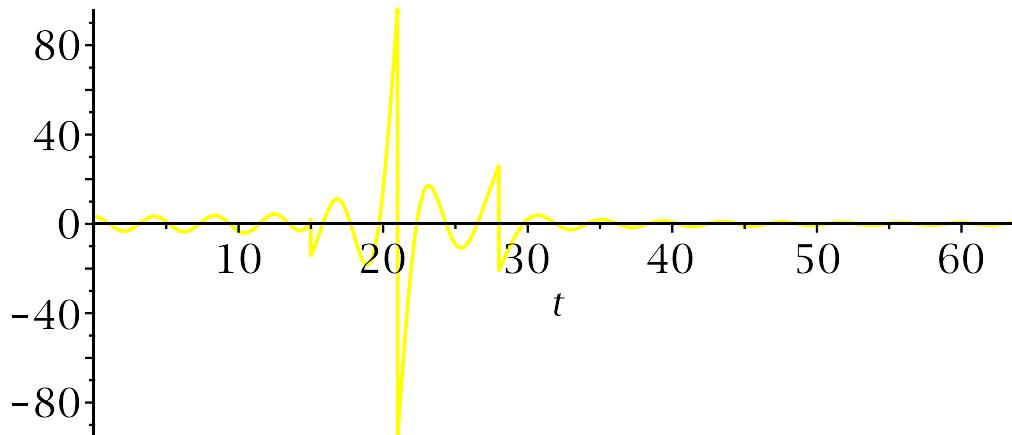
$$\Delta := \text{Matrix}\left(\left[\begin{array}{cccccc} 4, & 1, & 2, & 0, & 0, & -14, & 9, & -21 \end{array}\right], \left[\begin{array}{cccccc} 0, & 4, & 0, & 4, & 4, & -7, & -6, & 0 \end{array}\right]\right)$$

$[0, 4, 4, 11, -9, 0, 0, 0], [3, 0, -5, 7, 0, 0, 0, 0], [0, -16,$
 $17, 0, 0, 0, 0, 0], [16, -15, -2, 0, 0, 0, 0, 0], [-95, 0, -1,$
 $0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]]$

$$\Delta := \begin{bmatrix} 4 & 1 & 2 & 0 & 0 & -14 & 9 & -21 \\ 0 & 4 & 0 & 4 & 4 & -7 & -6 & 0 \\ 0 & 4 & 4 & 11 & -9 & 0 & 0 & 0 \\ 3 & 0 & -5 & 7 & 0 & 0 & 0 & 0 \\ 0 & -16 & 17 & 0 & 0 & 0 & 0 & 0 \\ 16 & -15 & -2 & 0 & 0 & 0 & 0 & 0 \\ -95 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.2.2.3.15)$$

Vi ser her at det er store forskjeller mellom pikselverdier. Dette er det samme fenomenet vi ser i grafen over. Hvis vi grafer forskjellen mellom $f(t)$ og $\psi_1(t)$ får vi en enda bedre visualisering av forskjellene.

$$P_\Delta := \text{plot}\left(f(t) - \psi_1(t), t = 0 \dots \frac{T}{2}, \text{color} = \text{"yellow"}\right)$$



Figur 42

Som vi nå også ser fra P_Δ så er forskjellen mellom orginalbildet og fourierrekken svært liten over et stort område, men det er to høye tagger. Disse kommer når grafen går fra sin høyeste verdi til sin laveste verdi. Til tross for at den høyeste avviket er på 95, så ser bildet svært bra ut. Vi kan nesten ikke se at det er blitt gjort om til en fourierrekke. Det vi kan legge merke til er at i det hvite partiet er det noen mørkere piksler. Disse har bare et avvik på 4, men vi kan se dem svært godt siden de er midt i den hvite delen av bildet. Dette forteller oss at mennesket er svært sensitivt til lyse områder i bilder og til forandringer i luminans.

Analysē

I anvendelsesdelen brukte vi 3 metoder for å generere en fourierrekke fra et bilde. Utifra disse tre metodene har vi sett at i alle tilfeller skapes det forvregninger i bildet. Spesielt ses dette i det kunstige bildet generert fra metode 1 (Se figur 22). I dette bildet blir noen piksler svarte, selv om de skulle vært hvite. Dette er fordi den hurtige

overgangen fra sort til hvit gjør at tilnærmingen ikke klarer å opprettholde detaljene. Derimot så var metode 3 svært god til å skape en tilnærming på det kunstige bildet (Se figur 41). På grafen av tilnærmingen til metode 3 på det kunstige bildet er det bare en forandring i verdi fra høy til lav, og dermed blir det en svært god tilnærming.

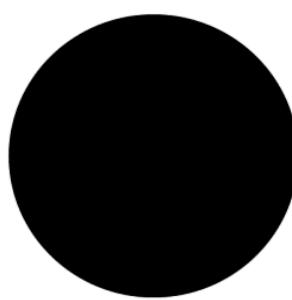
Generelt har vi sett at dette er en svært god måte å komprimere bilder på. I alle gjennomføringene har vi brukt 30 ledd i fourierrekken, og de fleste bildene har blitt godt tilnærmet. Siden vi hadde 64 pikselverdier som vi har forminsket til 30 ledd er dette en komprimering på nesten 50% av filstørrelsen. Dette er en god grunn til at det er mange bilder som er komprimert med JPEG, som tar i bruk konseptene bak fourierrekker for å komprimere bilder.

Et generelt problem som man kan legge merke til ved denne typen omgjøring er ved skarpe overganger mellom lyse og mørke farger. Dette er et kjent problem ved JPEG også. Grunnen til dette er at det er enklere å oppdage feil i nyanser langs kanter og at tilnærmingen mister nøyaktighet ved bråe forandringer. Dermed er ikke denne typen komprimering gunstig ved "kunstige" bilder, altså for eksempel bilder av tekst eller symboler.

Utifra resultatene kan man konkludere med at metode 2 og 3 er de som gir best kvalitet, mens metode 1 gir dårligst kvalitet. Grunnen til dette er at i metode 1 hopper man fra høyre siden av bildet til venstre når man starter på en ny linje. Dette gjør at hvis bildet er mørkere på den ene siden og lysere på den andre, så vil man få skarpe overganger ved hvert eneste linjeskift. Disse skarpe overgangene skaper forregninger i bilde og dermed er metode 1 en dårligere måte å finne en tilnærming til bilder på.

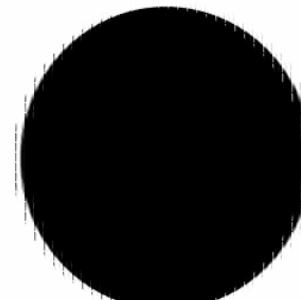
Metode 2 og 3 derimot fungerte bra ved at de hele tiden gikk videre til en "nabo"- piksel. Når det da er jevne overganger i bildet, altså at bildet er "naturlig", så vil det aldri bli bråe hopp mellom mørkt og lyst. Dette gjør at de fungerer godt til å skape en fourierrekke av et bilde.

På bildene under har vi originalbilder og fourierbilder. Dette er bilder som består av mange 8x8 blokker. Disse er generert ved hjelp av vedlegg 1. Her er det kjørt en full analyse av orginalbildet og det er benyttet 30 ledd i fourierrekken og et konstantledd.



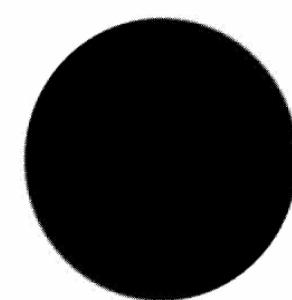
Figur 43

Originalbilde



Figur 44

Metode 1



Figur 45

Metode 3

På bildene over ser vi originalbildet og to bilder som er konstruert av metode 1 og metode 3. Vi kan legge merke til at i metode 1 så kan kantene på høyre og venstre side bli veldig dårlig rekonstruert. Dette er gitt at venstre og høyre kant har forskjellige farger. For mange kan dette virke merkelig, men for de som husker metoden som brukes så kan dette gi mye mening.

Ettersom denne metoden "leser" pikslene i bildet som en bok så vil hver linje starte på venstre side av bildet og slutte på høyre av bildet før neste rad blir analysert. Dette

fører til at pikselinformasjonen på høyre side blir etterfulgt at pikselinformasjonen på venstre side. Når det skal konstrueres en fourierrekke av disse verdiene så vil venstre side og høyre side påvirke hverandre. Dette resulterer da i at store verdiforskjeller fører til en dårlig komprimering av bildet. Det er viktig her å huske på at bildet overdeles inn i mange 8x8 piksel blokker. Når det er snakk om kanter her menes kantene på disse blokkene.

Ved første øyekast framstår metode 3 ganske likt som originalen. Derimot om man studerer fargeendringen fra hvitt til sort mer i detalj så kan man se at det er "grumsete". Det er varierende grad av upresis fargeovergang avhengig av hvor på sirkelomrisset man ser. Overgangen opp til venstre og nede til høyre er mye bedre kopiert enn nede til venstre og opp til høyre. Siden metoden analyserer fargedata diagonalt fra nede venstre til opp høyre (Z-diagonal) så vil diagonale elementer i bildet kompileres i ulik kvalitet. Dette har da å gjøre med hvordan data blir lagret og komprimert. Vi kan se at Z-diagonalen blir mye bedre ivaretatt av metode 3 enn den motsatte diagonalen (S-diagonal).

Om det er hyppig endring av farger eller skarpe overganger fra en farge til en annen generelt i bildet vil det også føre til at bildet framstår som "blurry". Ettersom metodene lagrer lest informasjon som en rekke. Store endringer i pikselverdier, slik som ved en brå overgang fra en farge til en annen vil gjøre det vanskelig å få en optimalisert fourierrekke. Noe som leder til et feilaktig bilde eller grumsete overgang.

Om man ikke ser veldig tydlig de karaktertrekkene som har vært nevnt over er det fordi bildet er veldig bra komprimert og rekonstruert. Om antall fourierledd i vi bruker til rekonstrueringen blir redusert vil disse karaktertrekkene bli mer tydige.

Under er naturbilder med bruk av samme metoder.



Figur 46
Originalbilde



Figur 47
Metode 3

5. Konklusjon

Prosjektet har gitt en grunnleggende innføring i hva fourierrekker er, og sett på typiske anvendelser hvor fourierrekker blir benyttet. Fourierrekker gjør det blant annet mulig å lagre bildedata på mindre plass, samtidig som det ikke fører til at betydelige mengder *oppfattet kvalitet* går tapt i prosessen. Ved å omgjøre et bildet til en fourierrekke ble det vist hvordan en fourierrekke kan inneholde data, som senere ble brukt til å bygge opp et bilde som var tilnærmet likt det opprinnelige bildet. Fourierrekken sparte også brukeren for betydelige mengder data, i forhold til hva man trenger for å lagre det originale bildet.

7. Referanser

Weisstein, Eric W. "Fourier Series." From MathWorld--A Wolfram Web Resource.
<http://mathworld.wolfram.com/FourierSeries.html> [Lesedato: 21.09.2017]
Weisstein, Eric W. "Generalized Fourier Series." From MathWorld--A Wolfram Web Resource.
<http://mathworld.wolfram.com/GeneralizedFourierSeries.html> [Lesedato: 21.09.2017]
"Complex form of fourier series", <https://www.math24.net/complex-form-fourier-series/>
[Lesedato: 21.09.2017]

Under avsnitt "Eksempel på anvendelse i musikkinstrumenter", http://www.uio.no/studier/emner/matnat/fys/FYS2150L/v09/undervisningsmateriale/sma_ovelser.html/Digitalisering%20av%20lyd.pdf [Lesedato: 29.09.17]
"Eksempel på anvendelse i musikkinstrumenter", <https://www.youtube.com/watch?v=paVleUlpKQ8>

<http://www.techradar.com/news/computing/all-you-need-to-know-about-jpeg-compression-586268/2> [Lesedato 01.10.2017]
<https://www.youtube.com/watch?v=Q2aEzeMDHMA&t=568s> [Lesedato 26.09.2017]
https://en.wikipedia.org/wiki/Discrete_cosine_transform [Lesedato 01.10.2017]

Bilderefanser

Bildet hentet fra: <https://en.wikipedia.org/wiki/YCbCr> hentet: 02.10.17
Bilde hentet fra: <https://youtu.be/Q2aEzeMDHMA?t=4m35s> Hentet: 02.10.17
Bilde hentet fra: <http://www.ece.ucdavis.edu/cerl/reliablejpeg/compression/> Hetet: 02.10.17
Bildet er hentet fra: <https://www.math.cuhk.edu.hk/~lmlui/dct.pdf> Hentet: 04.10.17
Bildet hentet fra: http://www.eetimes.com/document.asp?doc_id=1225736 Hentet: 1.10.17

```

#!/bin/env python3

# Man må installere python3.6, Pillow og numpy ved hjelp av pip install
# Altså etter man har installert python3.6 så må man skrive disse i et kommandovindu
#
# pip install Pillow
# pip install numpy
#
# Da skal det funke, hvis ikke spør meg
from PIL import Image
import numpy as np
import os
import json
import math
import re
import subprocess

#####
# Velg om du vil kjøre en full analyse av et helt bilde eller valgt blokk
FULL_ANALYZE = True

# Endre tallet for å velge metode!
METHODE = 1

# Skriv inn navnet på bildefilen. NB! Filen må ligge i mappen "fourier_bilder"
IMAGE_NAME = "natur.png"

# Antall ledd i cosinusrekke
AMOUNT_OF_TERMS = 30

#####
# HVIS IKKE FULL ANALYZE, bruk settinger under!
# Velg om du vil generere piecewise for maple eller Lage et bilde fra psi(t) funksjonen.
GENERATE_PIECEWISE_BOOL = True

# Velg perioden til piecewise-kommandoen. Det vi har brukt før er 64
PERIOD = 64

# Velg indeksene for 8x8 blokken dere vil analysere! 0, 0 er den blokken øverst til venstre.
BLOCK_INDEXES = ( 22, 26 ) # VELG HVILKEN BLOKK AV 8x8 DERE VIL UNDERSØKE. (x, y)
# natur (22,26)
# kunstig (11,11)

# Skriv inn en faktor som bildet blir skalert med når det genereres slik at det er enklere å analysere
SCALE_FACTOR = 20

# Her skriver man inn cosinusuttrykket fra maple! Sørg for at det ser riktig ut og at verdien fra
# cosinusuttrykket blir returnert fra funksjonen. DETTE ER BARE HVIS ANALYSE AV BLOKKER.
# Bytt ut "255*cos(.4*t)" med det som kommer ut fra maple!
def psi(t):
    return 87.31250000-3.566504710*cos(.1472621557*t)+141.0276691*cos(0.4908738522e-1*t)+0.985911939*cos(.2945243113*t)-30.20040468*cos(.1963495409*t)-19.82176616*cos(.2454369261*t)+64.69265187*cos(0.9817477044e-1*t)-5.785914227*cos(.4908738522*t)+11.87593921*cos(.3436116965*t)+9.888450648*cos(.3926990818*t)+1.725469068*cos(.4417864670*t)-8.730228779*cos(.8344855487*t)-8.052902122*cos(.5399612374*t)-3.278710526*cos(.5890486226*t)+4.938007688*cos(.6381360078*t)-5.852056005*cos(.7853981635*t)-2.248346107*cos(.8835729339*t)+6.159973590*cos(.9326603192*t)+7.475453779*cos(.9817477044*t)+8.610305090*cos(.6872233931*t)+3.258151057*cos(.7363107783*t)+1.287146206*cos(1.030835090*t)-4.999438596*cos(1.079922475*t)-5.420052033*cos(1.129009860*t)-3.142672782*cos(1.423534171*t)-2.335600308*cos(1.472621557*t)+3.729066048*cos(1.276272016*t)-1.094383682*cos(1.178097245*t)+2.971808007*cos(1.227184630*t)+1.643479624*cos(1.325359401*t)-1.312866830*cos(1.374446786*t)

#####
IMAGE_DIR = os.path.realpath(os.path.join(os.path.dirname(__file__), '../bilder/fourier_bilder/'))
ANALYSERTE_BLOKKER_DIR = os.path.join(IMAGE_DIR, "analyserte_blokker/")

class pf:
    PURPLE = '\u033[95m'
    CYAN = '\u033[96m'
    DARKCYAN = '\u033[36m'
    BLUE = '\u033[94m'
    GREEN = '\u033[92m'
    YELLOW = '\u033[93m'
    RED = '\u033[91m'
    BOLD = '\u033[1m'
    UNDERLINE = '\u033[4m'
    END = '\u033[0m'

    def format(text, l_pf):
        return l_pf + text + pf.END

    def clamp(n, smallest, largest):
        return max(smallest, min(n, largest))

```

```

# Returnerer en 8x8 som er et resultat av psi(t)
def change_to_fourierseriesvalues(eight_by_eight, methode):
    try:
        value_array = ImageArrayToValues.convert(eight_by_eight, methode)
        T = len(value_array)
    except:
        print("Error: Vennligst velg en \"method\" funksjon.")
        return eight_by_eight

    for i in range(T):
        value_array[i] = clamp(int(psi(i)), 0, 255)

    return ValuesToImageArray.convert(value_array, methode)

# Returnerer en 8x8 som er et resultat av psi_string
def change_to_fourierseriesvalues_psi_string(eight_by_eight, methode, psi_string):
    try:
        value_array = ImageArrayToValues.convert(eight_by_eight, methode)
        T = len(value_array)
    except:
        print("Error: Vennligst velg en \"method\" funksjon.")
        return eight_by_eight

    for t in range(T):
        value_array[t] = clamp(int(eval(psi_string)), 0, 255)

    return ValuesToImageArray.convert(value_array, methode)

# Bredde og høyde på arrayen må være delelig på 8
def array_into_eight_by_eight(np_array):
    width = np_array.shape[0]
    height = np_array.shape[1]
    if(width % 8 > 0 or height % 8 > 0):
        return None

    # Les som en bok
    array_eight_by_eights = [ [] for i in range(int(height/8)) ]
    array_eight_by_eight_col_offset = 0

    for y_matrix in range(int(height/8)):
        n1 = np.arange(8*y_matrix, 8*y_matrix+8)
        for x_matrix in range(int(width/8)):
            n2 = np.arange(8*x_matrix, 8*x_matrix+8)
            array_eight_by_eights[array_eight_by_eight_col_offset].append(np_array[n2[None,:,:],n1[:,None]])

        array_eight_by_eight_col_offset += 1

    return array_eight_by_eights

# Gjør om en 2d array med 8x8 blokker til en stor 2d array med pikselverdier
def assembly_array_of_eight_by_eights(array_eight_by_eights):
    rows_eight_by_eight = []
    for row_eight_by_eight in array_eight_by_eights:
        if len(row_eight_by_eight) > 0:
            rows_eight_by_eight.append(np.concatenate(row_eight_by_eight, axis=1))
    return np.concatenate(rows_eight_by_eight, axis=0)

# Gjør om en 2d array med pikselverdier til en png-bildefil
def image_array_to_image(np_array, filepath):
    new_image = Image.fromarray(np_array, 'L')
    with open(filepath, 'wb') as image_file:
        new_image.save(image_file, 'PNG')

def image_array_to_image_scale(np_array, filepath, scalefactor):
    first_row = True
    scaled_array = None
    for row in np_array:
        combined_scaled_row = None
        first_value = True
        for value in row:
            scaled_value_array = np.full((scalefactor, scalefactor), value, dtype=np.uint8)
            combined_scaled_row =
                np.concatenate([combined_scaled_row, scaled_value_array], axis=1)
            if not first_value else scaled_value_array
            first_value = False

        scaled_array = np.concatenate([scaled_array, combined_scaled_row], axis=0) if not first_row else combined_scaled_row
        first_row = False

    new_image = Image.fromarray(scaled_array, 'L')
    with open(filepath, 'wb') as image_file:
        new_image.save(image_file, 'PNG')

# Gjør om en bildefil til en 2d array med pikselverdier (bare sort/hvitt)
def image_to_image_array_loaded_image():
    converted_image_file = loaded_image.convert('L')
    return np.reshape(np.array(converted_image_file.getdata(), np.dtype(np.uint8)), (-1, converted_image_file.width))

# Skriver en array til en json fil
def write_array_to_datafile(filepath, np_array):
    with open(filepath, "w") as json_file:
        json_str = json.dumps(np_array.tolist())

```

```

json_str = re.sub(r'([0-9]+\]',)', '\g<1>\n', json_str)
json_str = re.sub(r'([^\-]\b[0-9]{1}\b)', ' \g<1>', json_str)
json_str = re.sub(r'([^\-]\b[0-9]{1}\b)|([^\-]\b[0-9]{2}\b)', ' \g<1>\g<2>', json_str)
json_str = re.sub(r'([^\-]\b[0-9]{2}\b)|([^\-]\b[0-9]{3}\b)', ' \g<1>\g<2>', json_str)
json_file.write(json_str)

# Leser en json fil med en 2d array
def read_array_from_datafile(filepath):
    res_json = ""
    with open(filepath, 'r') as json_file:
        res_json = json.load(json_file)
    return res_json

# Ser om en mappe eksisterer og lager den hvis ikke den eksisterer. Returnerer dirpath
def check_directory(dirpath):
    if not os.path.exists(dirpath):
        os.makedirs(dirpath)
    return dirpath

# Gjør om 1d array med verdier til en piecewise kommando
def array_to_piecewise(array, periode):
    res = "f(t):=piecewise("
    roffset = 0
    array_len = len(array)
    increment = periode/array_len
    for value in array:
        res += str(roffset) + "<=t<=" + str(roffset+increment) + "," + str(value) + ","
        roffset += increment
    res = res.strip().strip(',')
    return res + ")"

# Gjør om 1d array med verdier til en piecewise kommando
def array_to_piecewise_cli(array, periode):
    res = "f(t):=piecewise("
    roffset = 0
    array_len = len(array)
    increment = periode/array_len
    for value in array:
        res += str(roffset) + "<=t and t<=" + str(roffset+increment) + "," + str(value) + ","
        roffset += increment
    res = res.strip().strip(',')
    return res + ")"

class ImageArrayToValues():
    def __init__(self):
        print("Class not supposed to be instantiated!")
        return None

    # Gjør om en array til en 1d array med pikselverdier. 3 ulike metoder (se "Prosjektoppgaven.mw")
    @staticmethod
    def __image_array_to_values_metode1(np_array):
        res = []
        for row in np_array:
            for value in row:
                res.append(value)
        return res

    @staticmethod
    def __image_array_to_values_metode2(np_array):
        res = []
        reverse_row = False
        for row in np_array:
            altered_row = reversed(row) if reverse_row else row
            reverse_row = not reverse_row
            for value in altered_row:
                res.append(value)
        return res

    @staticmethod
    def __image_array_to_values_metode3(np_array):
        res = []
        increment_offset = 1
        x = 0
        y = 0

        res.append(np_array[y][x])

        for index in range(int(np_array.shape[0]/2)):
            for i in range(increment_offset, 0, -1):
                if(y > 0):
                    y -= 1
                x += 1
                res.append(np_array[y][x])
                #print("X-Loop: " + str(y) + str(x))

            increment_offset += (1 if increment_offset < np_array.shape[0]-1 else 0)

            for i in range(increment_offset, 0, -1):
                if(x > 0):
                    x -= 1
                y += 1

```

```

        res.append(np_array[y][x])
        #print("Y-Loop: " + str(y) + str(x))

    increment_offset += (1 if increment_offset < np_array.shape[0]-1 else 0)

    for index in range(int(np_array.shape[0]/2)):
        moved_over = False
        for i in range(increment_offset, 0, -1):
            if(y > 0 and moved_over):
                y -= 1
            moved_over = True
            x += 1
        res.append(np_array[y][x])
        #print("X-Loop: " + str(y) + str(x))

    increment_offset -= 1

    moved_over = False
    for i in range(increment_offset, 0, -1):
        if(x > 0 and moved_over):
            x -= 1
        moved_over = True
        y += 1
    res.append(np_array[y][x])
    #print("Y-Loop: " + str(y) + str(x))

increment_offset -= 1

return res

image_array_to_values_methods = {
    '1':__image_array_to_values_metode1.__func__,
    '2':__image_array_to_values_metode2.__func__,
    '3':__image_array_to_values_metode3.__func__
}

@classmethod
def convert(cls, np_array, method):
    return cls.image_array_to_values_methods[str(method)](np_array)

class ValuesToImageArray():

    def __init__(self):
        print("Class not supposed to be instantiated!")
        return None

    # Gjør om en 1d array til en 2d array med pikselverdier. 3 ulike metoder.
    @staticmethod
    def __values_to_image_array_metode1(value_array):
        np_array = [ [] for i in range(8) ]
        for i in range(8):
            for j in range(8):
                np_array[i].append(value_array[j + (i * 8)])

        return np.array(np_array, dtype=np.uint8)

    @staticmethod
    def __values_to_image_array_metode2(value_array):
        np_array = [ [] for i in range(8) ]
        reverse_row = False
        for i in range(8):
            for j in range(8):
                row_offset = j if not reverse_row else 7-j
                np_array[i].append(value_array[(i*8) + row_offset])
            reverse_row = not reverse_row
        return np.array(np_array, dtype=np.uint8)

    @staticmethod
    def __values_to_image_array_metode3(value_array):
        np_array = np.indices((8,8))[0]
        increment_offset = 1
        x = 0
        y = 0

        reversed_value_array = list(reversed(value_array))

        np_array[y][x] = reversed_value_array.pop()

        for index in range(int(np_array.shape[0]/2)):
            for i in range(increment_offset, 0, -1):
                if(y > 0):
                    y -= 1
                x += 1
            np_array[y][x] = reversed_value_array.pop()
            #print("X-Loop: " + str(y) + str(x))

        increment_offset += (1 if increment_offset < np_array.shape[0]-1 else 0)

        for i in range(increment_offset, 0, -1):
            if(x > 0):
                x -= 1
            y += 1

```

```

np_array[y][x] = reversed_value_array.pop()
#print("Y-Loop: " + str(y) + str(x))

increment_offset += (1 if increment_offset < np_array.shape[0]-1 else 0)

for index in range(int(np_array.shape[0]/2)):
    moved_over = False
    for i in range(increment_offset, 0, -1):
        if(y > 0 and moved_over):
            y -= 1
            moved_over = True
        x += 1
        np_array[y][x] = reversed_value_array.pop()
        #print("X-Loop: " + str(y) + str(x))

    increment_offset -= 1

    moved_over = False
    for i in range(increment_offset, 0, -1):
        if(x > 0 and moved_over):
            x -= 1
            moved_over = True
        y += 1
        np_array[y][x] = reversed_value_array.pop()
        #print("Y-Loop: " + str(y) + str(x))

    increment_offset -= 1

return np.array(np_array, dtype=np.uint8)

values_to_image_array_methods = {
    '1':__values_to_image_array_metode1.__func__,
    '2':__values_to_image_array_metode2.__func__,
    '3':__values_to_image_array_metode3.__func__
}

@classmethod
def convert(cls, valuearray, method):
    return cls.values_to_image_array_methods[str(method)](valuearray)

# En funksjon som brukes testing av et Lite bilde
def test_main():
    image_name = IMAGE_NAME.split('.')[0]

    np_array = None
    with Image.open(os.path.join(IMAGE_DIR, IMAGE_NAME)) as big_image:
        np_array = image_to_image_array(big_image)

    array_eight_by_eights = array_into_eight_by_eight(np_array)
    print(assembly_array_of_eight_by_eights(array_eight_by_eights))

def generate_piecewise():
    image_name = IMAGE_NAME.split('.')[0]

    np_array = None
    with Image.open(os.path.join(IMAGE_DIR, IMAGE_NAME)) as big_image:
        np_array = image_to_image_array(big_image)

    array_eight_by_eights = array_into_eight_by_eight(np_array)

    selected_eight_by_eight = array_eight_by_eights[BLOCK_INDEXES[0]][BLOCK_INDEXES[1]]
    print("Kjører metode " + str(METHODE) + " paa bilde " + pf.format(IMAGE_NAME, pf.BOLD) +
    " blokk " + pf.format(str(BLOCK_INDEXES),pf.BOLD) + " med " + pf.format(str(PERIOD), pf.BOLD) + " som periode")
    print(array_to_piecewise(ImageArrayToValues.convert(selected_eight_by_eight, METHODE), PERIOD))

def generate_image_from_psi():
    image_name = IMAGE_NAME.split('.')[0]

    print("Laster bilde: " + pf.format(IMAGE_NAME, pf.BOLD))
    np_array = None
    with Image.open(os.path.join(IMAGE_DIR, IMAGE_NAME)) as big_image:
        np_array = image_to_image_array(big_image)

    print("Skriver bildedata av hele bildet til json-filer.")
    write_array_to_datafile(os.path.join(IMAGE_DIR, image_name + ".json"), np_array)

    print("Genererer 8x8 blokker av det lastede bildet.")
    array_eight_by_eights = array_into_eight_by_eight(np_array)

    methode_dir = os.path.join(ANALYSERTE_BLOKKER_DIR, image_name, "metode" + str(METHODE))
    image_org_dir = check_directory(os.path.join(methode_dir, "bilder", "org/"))
    image_fourier_dir = check_directory(os.path.join(methode_dir, "bilder", "fourier/"))
    data_org_dir = check_directory(os.path.join(methode_dir, "data", "org/"))
    data_fourier_dir = check_directory(os.path.join(methode_dir, "data", "fourier/"))
    data_delta_dir = check_directory(os.path.join(methode_dir, "data", "delta/"))

    print("Velger 8x8-blokken som skal analyseres: " + pf.format(str(BLOCK_INDEXES), pf.BOLD))
    print("Lager bilde av 8x8-blokken som skal analyseres før den endres.")
    org_eight_by_eight = array_eight_by_eights[BLOCK_INDEXES[0]][BLOCK_INDEXES[1]]
    image_array_to_image_scale(org_eight_by_eight, os.path.join(image_org_dir, str(BLOCK_INDEXES) + ".png"), SCALE_FACTOR)

    print("Skriver data av 8x8-blokken som skal analyseres før den endres.")

```

```

write_array_to_datafile(os.path.join(data_org_dir, str(BLOCK_INDEXES) + ".json"), org_eight_by_eight)

print(pf.format("Lager en ny 8x8 fra psi(t) funksjonen.", pf.BOLD))
fourier_eight_by_eight = change_to_fourierseriesvalues(org_eight_by_eight, METHODE).astype(np.dtype(np.uint8))

print("Lager bilde av 8x8-blokken som er generert fra psi(t).")
image_array_to_image(fourier_eight_by_eight, os.path.join(image_fourier_dir, str(BLOCK_INDEXES) + ".png"))

print("Skriver data av 8x8-blokken som er generert fra psi(t).")
write_array_to_datafile(os.path.join(data_fourier_dir, str(BLOCK_INDEXES) + ".json"), fourier_eight_by_eight)

print("Skriver data av 8x8-blokken som er forskjellen mellom pikselverdiene fra orginal og fourier")
write_array_to_datafile(os.path.join(data_delta_dir, str(BLOCK_INDEXES) + ".json"),
org_eight_by_eight.astype(np.dtype(np.int8)) - fourier_eight_by_eight.astype(np.dtype(np.int8)))

print(pf.format("Fullført! Sjekk \"fourier_bilder\" mappen for oppdateringer!", pf.BOLD))

def transform_full_image():
    image_name = IMAGE_NAME.split('.')[0]

    np_array = None
    with Image.open(os.path.join(IMAGE_DIR, IMAGE_NAME)) as big_image:
        np_array = image_to_image_array(big_image)

    temp_dir = check_directory(os.path.join(IMAGE_DIR, "tmp/"))

    array_eight_by_eights = array_into_eight_by_eight(np_array)
    max_progress = (len(array_eight_by_eights) - 1) * (len(array_eight_by_eights[0]) - 1)

    string_of_piecewise = ""

    piecewise_to_psi_script_path = os.path.join(temp_dir, "piecewise_to_psi.mpl")
    psi_output_path = os.path.join(temp_dir, "psi_output.txt")
    with open(piecewise_to_psi_script_path, 'r') as script_file:
        piecewise_script_lines = script_file.readlines()

    for y_index, row in enumerate(array_eight_by_eights):
        for x_index, eight_by_eight in enumerate(row):
            string_of_piecewise = array_to_piecewise_cli(ImageArrayToValues.convert(eight_by_eight, METHODE), PERIOD)

            with open(piecewise_to_psi_script_path, "w") as piecewise_to_psi_file:
                piecewise_script_lines[0] = string_of_piecewise + ": T:=" + str(2*PERIOD) + ": N:=" + str(AMOUNT_OF_TERMS) + ":\r"
                piecewise_to_psi_file.writelines(piecewise_script_lines)

            #with open(psi_output_path, 'w') as psi_output_file:
            maple_results = subprocess.Popen(['maple', '-q', piecewise_to_psi_script_path], stdout=subprocess.PIPE, shell=False)

            psi_string = maple_results.stdout.readline().decode('utf-8')

            #with open(psi_output_path, 'r') as psi_output_file:
            #    psi_string = psi_output_file.readlines()[0]

            psi_string = psi_string.replace("cg = ", "")

            array_eight_by_eights[y_index][x_index] = change_to_fourierseriesvalues_psi_string(eight_by_eight, METHODE, psi_string)

            current_progress = (y_index*len(array_eight_by_eights) + x_index)
            print(str(current_progress) + " av " + str(max_progress))

    generated_image_dir = check_directory(os.path.join(IMAGE_DIR, image_name, "metode" + str(METHODE)))
    image_array_to_image(assembly_array_of_eight_by_eights(array_eight_by_eights), os.path.join(generated_image_dir, "fourier.png"))

def run():
    if(FULL_ANALYZE):
        transform_full_image()
    else:
        if(GENERATE_PIECEWISE_BOOL):
            generate_piecewise()
        else:
            generate_image_from_psi()

def main():
    run()

if __name__ == '__main__':
    main()

```