7. februar, 2013

MAT-INF 2360: Obligatorisk oppgave 1

Innleveringsfrist: 28/2-2013, kl. 14:30

Informasjon

Skriftlige besvarelser skal leveres i obligkassa som står i gangen utenfor ekspedisjonen i 7. et. i Niels Henrik Abels hus. Du kan også levere via devilry. Fristen er *kl. 14.30 torsdag 28/2*. Besvarelsen *skal* være skrevet av deg selv, for hånd eller på datamaskin.

Studenter som blir syke eller av andre grunner trenger å søke om utsettelse for denne obligatoriske oppgaven, må ta kontakt med studieadministrasjonen ved Matematisk institutt (7. et. Niels Henrik Abels hus, telefon 22 8558 88, e-post: studieinfo@math.uio.no) i god tid før innleveringsfristen.

Det oppfordres til samarbeid underveis i arbeidet med oppgavene, og gruppelærer og forelesere har anledning til å svare på generelle spørsmål, men kan ikke servere ferdige løsninger. *Den endelige besvarelsen som du leverer skal utarbeides av deg selv, og du må kunne redegjøre for innholdet ved en eventuell muntlig høring (aktuelt ved mistanke om avskrift).*

Husk at de tre obligatoriske oppgavene i MAT-INF 2360 alle må bestås for å kunne gå opp til endelig eksamen i kurset. *For å få bestått på denne første obligatoriske oppgaven må du gjøre seriøse løsningsforsøk på alle oppgavene, og minst 6 av de 8 deloppgavene bør være riktig besvart.*

1. I denne oppgaven skal vi sammenligne effektiviteten av FFT-algoritmen med en mer rett frem algoritme for DFT.

a. Lag en funksjon `y=DFTImpl(x)` og en funksjon `y=FFTImpl(x)`, som genererer en DFT med hhv. den enkle DFT-algoritmen, og FFT-algoritmen fra kapittel 2, og returnerer DFT-en i en vektor `y`. Forklar algoritmene. Du kan anta at du tar inn en søylevektor av lengde $N = 2^n$, hvor n er et positivt heltall. Dersom du skriver koden i Matlab kan du godt bruke koden som er gitt i kompendiet (og som du kan finne på http://folk.uio.no/oyvindry/matinf2360/matlab/), men du må likevel forklare algoritmene.

b. I denne oppgaven skal du teste tidsforbruket til funksjonene i a.. Ta utgangspunkt i fila `castanets.wav` fra kompendiet, som du kan laste ned fra http://folk.uio.no/oyvindry/matinf2360/sounds/castanets.wav, og mål tidsforbruket til hver av funksjonene. Bruk bare den første lydkanalen i stereolyden (det vil si første søyle), og test funksjonene for de 2^n første samplene i fila, der $n = 4,5,6,\ldots,14$. Presenter resultatene i form av et plott der n plottes langs x-aksen. Kommenter resultatene ut fra det som står om numerisk kompleksitet av DFT og FFT i kompendiet. Forsøk også å kjøre koden for høyere verdier av n, og beskriv hva som skjer.

Dersom du skriver i Matlab kan du bruke de innebygde funksjonene `tic` og `toc` på følgende måte for å måle tidsbruk:

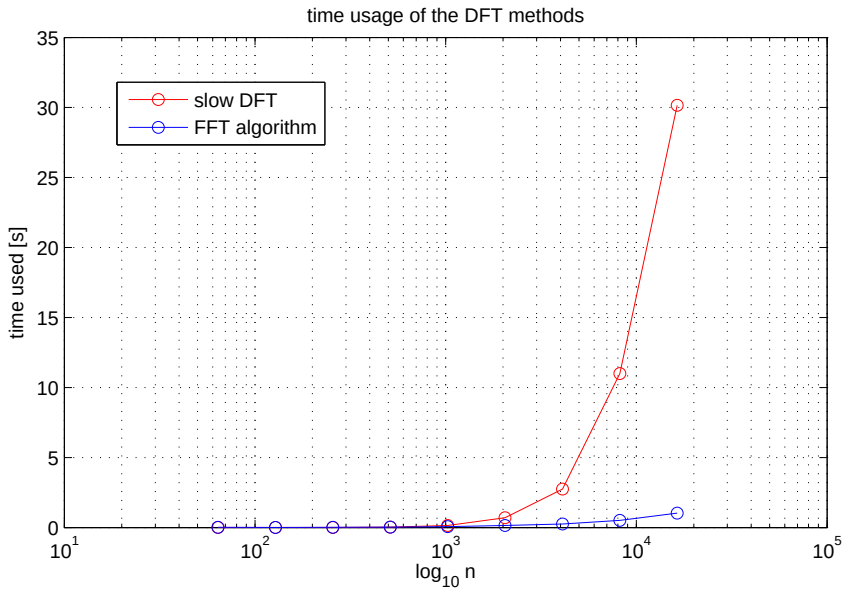
```
clear   all ;
close   all ;
tic ;
start = toc;
% Din kode her
stop = toc;
time_spent = stop-start;
```

Løsningsforslag: En kode som tester de forskjellige verdiene kan i Matlab se slik ut:

```
close all;
clear all;
tic;
totalstart = toc;

[S,Fs] = wavread('castanets');
```

```
mono = S(:,1);

k = 6;
final_k = 14;
t = zeros(2,final_k-k);
n = 2.^(k:final_k);
while k <= final_k
    x = mono(1:2^k);

    start = toc;
    y1 = DFTImpl(x);
    stop = toc;
    time_spent = stop-start;
    t(1,k-5) = time_spent;

    start = toc;
    y2 = FFTImpl(x);
    stop = toc;
    time_spent = stop-start;
    t(2,k-5) = time_spent;

    k = k+1;
end
totalstop = toc;
totaltime = totalstop-totalstart;
slowtime = t(1,:);
fasttime = t(2,:);
semilogx(n, slowtime, 'ro-')
hold on
semilogx(n,fasttime, 'bo-')
grid on
title('time usage of the DFT methods')
legend('slow DFT', 'FFT algorithm')
xlabel('log_{10} n')
ylabel('time used [s]')
```

Resultatet vises i figur 1 og tabell 1. Vi ser at FFT-en definitivt er langt raskere for store verdier av n, og vi kan også se proporsjonaliteten dersom vi lager et log-log plot. Dersom vi prøver $n = 2^{15}$, viser det seg at maskina ikke har nok minne til å konstruere fouriermatrisen (Dette er ikke nød-



Figur 1: Figuren viser tidsforbruken til funksjonene DFTimpl(x) og FFTImpl(x) som funksjoner av lengden på vektoren. n-aksen er logaritmisk.

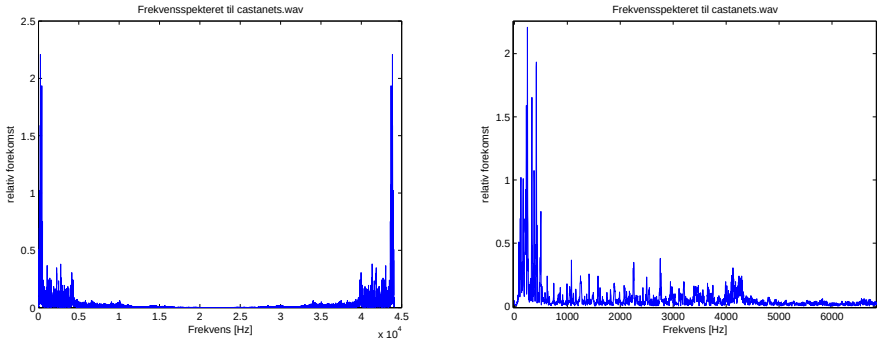
vector length	slow DFT	FFT
2^6	0.0044	0.0119
2^7	0.0020	0.0078
2^8	0.0070	0.0155
2^9	0.0288	0.0314
2^{10}	0.1509	0.0641
2^{11}	0.6969	0.1471
2^{12}	2.7470	0.2578
2^{13}	11.000	0.5122
2^{14}	30.1558	1.0257

Tabell 1: Tabellen viser tidsforbruket til de to DFT-algoritmene for forskjellige verdier av n

vendigvis det tallet som gjelder på datarommet).

2. I denne oppgaven skal vi se nærmere på noen egenskaper for DFT til et lydklipp. Først trenger vi en rask implementasjon av invers DFT.

a. Algoritmen `FFTImpl` var basert på Teorem 2.34 i kompendiet. Lag på



(a) Hele frekvensspekteret (b) området 0-7000 Hz

Figur 2: Et bilde av frekvensspekteret, med et utsnitt på det kraftigste området. Figuren bruker absoluttverdien til fourierkoeffisientene.

samme måte en implementasjon `IFFTImpl` av IDFT, som er basert på Teorem 2.35. i kompendiet.
Løsningsforslag: Koden kan se slik ut:

```
function x = IFFTImpl(y)
N = length(y);
if N == 1
    x = y(1);
else
    ye = y(1:2:(N-1));
    yo = y(2:2:N);
    xe = IFFTImpl(ye);
    xo = IFFTImpl(yo);
    D=exp(2*pi*1j*(0:N/2-1)'/N);
    x=[ xe + xo.*D; xe - xo.*D]/sqrt(2);
end
```

b. Lag et plott som viser DFT-verdiene til de første 2^{17} samplene av fila `castanets.wav`. Plottet skal ha DFT-indeks langs x-aksen. Kommenter.
Løsningsforslag: Resultatet vises i figur 2. Vi ser at fouriertransformen gir en symmetri om midten, som skyldes symmetrien vi har lært om for DFT-koeffisientene for reelle signaler. Spesifikt ser vi for denne lyden at de mest tydelige frekvensene ligger på 5000 Hz og lavere. Dette stemmer med hva vi hører i lyden.

c. I neste deloppgave skal vi høre på lyden vi får dersom vi fjerner frekvensene som vi typisk ser på som overflødige. Vi skal betrakte frekvenser på 5000 Hz eller høyere som overflødige, og vi må første finne ut hvilke DFT-indekser dette svarer til. Bruk sammenhengen mellom frekvens i Fourierrekker og DFT-indeks i kompendiet til å finne ut hvilke DFT-indekser som svarer til frekvenser under 5000 Hz i lydfila. For å finne disse indeksene trenger du også å vite samplingsraten for lyden. Denne skal du ha fått ut når du leste inn fila med kommandoen `wavread`.
Løsningsforslag: I en Fourierrekke svarer den rene tonen $e^{2\pi int/T}$ til DFT-indeks n, og denne har frekvens $\nu = n/T$. Videre har vi sammenhengen $N = Tf_s$, der T er perioden, f_s er samplingsfrekvens, og N er antall sampler i en periode. Vi ser her kun på de $N = 2^{17}$ første samplene, og `wavread` sier at $f_s = 44100$. Sammenhengene $\nu = n/T$ og $N = Tf_s$ gir dermed at $\nu = 5000$ Hz svarer til DFT indeks

$$n = \nu T = \frac{\nu N}{f_s} = \frac{5000 \times 2^{17}}{44100} \approx 14861.$$

Dette betyr at vi skal nullstille DFT-indekser i $[14861, 2^{17} - 14861] = [29722, 116211]$: De andre indeksene i $[0, 2^{17} - 1]$ svarer til frekvenser under 5000 Hz i lydfila.

Du kan også argumentere med at frekvensindeks $N/2$ svarer til frekvens $\nu = f_s/2$, og regne deg frem til riktig frekvensindeks ut fra dette.

d. Sett alle DFT-indekser som svarer til frekvenser over 5000 Hz lik 0. Ta en invers DFT og spill av det tilbaketransformerte signalet. Beskriv hva du hører. Gjør det samme igjen, men fjern alle frekvenser over 3000 Hz i stedet. Beskriv forskjellen.
Hint: Det kan hende du får noen komplekse komponenter i det tilbaketransformerte signalet som er i størrelsesorden på maskinnøyaktighet. Dette skyldes små avrundingsfeil og kan enklest løses ved å hente ut kun realdelen av signalet ved kommandoen `x=real(x)`.
Løsningsforslag: Bruker vi $\nu = 3000$Hz i stedet over finner vi $n = 3000 \times 2^{17}/44100 \approx 14861$, slik at vi her skal nullstille DFT-indekser i $[8916, 2^{17} - 8916] = [8916, 122156]$. Følgende kode vil avspille lyden først når vi kutter ut DFT-indekser over 5000 Hz, deretter over 3000Hz.

```
[S,fs]=wavread('castanets.wav');
x=S(1:2^17,1);
y=FFTImpl(x);
indices=14861:116211;
```

```
y(indices) = zeros(length(indices),1);
xnew=IFFTImpl(y);
playerobj=audioplayer(real(xnew),fs);
play(playerobj);

indices=8916:122156;
y(indices) = zeros(length(indices),1);
xnew=IFFTImpl(y);
playerobj=audioplayer(real(xnew),fs);
play(playerobj);
```

Når vi fjerner alt over 5000 Hz høres lyden nokså lik ut, men fjerner vi alt over 3000 Hz er den tydelig forringet.

3. Mange standarder for å komprimere lyd (slik som MP3) kjører en FFT på lyden, og gjør deretter en avrunding av DFT-koeffisientene, slik som at tallene rundes av til et visst antall binære bits. Dette gir oss kompresjon, siden DFT-koeffisientene lagres med færre antall bits. Ulempen er en forringelsen i lyden. Moderne lydstandarder minimerer forringelsen i lyden ved å bruker flere bits på frekvenser som hørselen er mer sensitiv på - det vil si at avrundingen er forskjellig for forskjellige frekvenser. I denne oppgaven skal vi for enkelhets skyld anta at alle lydsamplene rundes av etter samme regel. Vi skal igjen fokusere på de første 2^{17} samplene av `castanets.wav`.

a. Skriv en funksjon `newy=roundvals(y,n)`, som returnerer en vektor `newy` av samme lengde som `y`, men der alle verdier er erstattet av nærmeste tall med kun n desimale bits.

Hint: Du kan bruke Matlab-kommandoen `round` til å hjelpe deg her.
Løsningsforslag: Koden kan se slik ut

```
function newy = roundvals(y, n)
  y=y*2^n;
  y=round(y);
  newy=y/2^n;
```

b. Skriv en funksjon `kvantiser` som tar parameter n og som kjører en FFT på `castanets.wav`, kjører funksjonen `roundvals` med parameter n på DFT-koeffisientene, kjører IFFT, og spiller av lyden. Kjør `kvantiser` med $n = 8, 5, 3, 1$, og gi kommentarer til kvaliteten i hvert av tilfellene. Hvor

mange bits sparer du ved lagring av hver DFT-koeffisient for $n = 1$, sammenlignet med for $n = 8$?

Løsningsforslag: Koden kan se slik ut

```
function kvantiser(n)
  [S,fs]=wavread('castanets.wav');
  x=S(1:2^17,1);
  y=FFTImpl(x);
  y=roundvals(y,n);
  xnew=IFFTImpl(y);
  playerobj=audioplayer(real(xnew),fs);
  playblocking(playerobj);
```

For $n = 8$ bruker vi 7 flere bits for å lagre desimaldelen for hver DFT-koeffisient, sammenlignet med for $n = 1$.