21.oktober, 2005

MAT-INF 1100: Obligatorisk oppgave 2

Innleveringsfrist: 4/11-2005, kl. 14:30

Informasjon

Den skriftlige besvarelsen skal leveres på ekspedisjonskontoret i 7. etg. i Niels Henrik Abels hus senest *kl. 14.30 fredag 4/11.* Javaprogrammer og kjøring av slike leveres som utskrift fra skriver, men for hver oppgave skal du også levere en beskrivelse som kommenterer resultatene. Denne *skal* være skrevet av deg selv (for hånd eller på datamaskin).

Studenter som blir syke eller av andre grunner ønsker å søke om utsettelse eller fritak fra denne obligatoriske oppgaven, må ta kontakt med Elisabeth Seland (rom B726 Niels Henrik Abels hus, telefon 22 85 59 07, e-post: elisabhh@math.uio.no) i god tid før innleveringsfristen.

Det oppfordres til samarbeid underveis i arbeidet med oppgavene, og gruppelærerne 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 to obligatoriske oppgavene i MAT-INF 1100 begge må bestås for å kunne gå opp til endelig eksamen i kurset. *For å få bestått på denne obligatoriske oppgaven bør alle oppgavene være forholdsvis fullstendig besvart (ikke de frivillige oppgavene).*

Oppgave 1. Filtrering av lyd

I denne oppgaven skal du programmere noen enkle filteroperasjoner på lyd. Til hjelp i programmeringen kan du bruke eksempelprogrammet som du finner på fila `Filters.java`. Dette viser hvordan du kan bruke klassen `MatInf1100Sound` (du har fått med deg begge filene om du har utført `wget`-kommandoen på hjemmesida til oblig2.) Programmet `Filters.java` er svært enkelt og bør gi deg nok informasjon til å løse oppgavene under.

Klassen `MatInf1100Sound` inneholder en god del kode for å manipulere lyd, men det er ikke så mye du trenger å vite om den. Det som du faktisk bør vite er at konstruktøren til `MatInf1100Sound` har signaturen

```
MatInf1100Sound(short [] samples, double sampleRate)
```

Den har altså to parametre `samples` (som inneholder sampleverdiene til lyden) og `sampleRate` som inneholder samplingsraten. Disse to størrelsene kan du også få tak i ved dot-notasjon, det vil si at hvis `f` er et objekt av type `MatInf1100Sound` så vil `f.samples` gi deg samplene til `f` og `f.sampleRate` gi deg samplingsraten til `f`. Som du vil se fra `Filters.java` inneholder klassen også metodene `getSoundFromFile(string)` og `getSampleSound()` som du kan bruke for å lese inn en lyd fra fil (både aif- og wav-format er støttet), eventuelt generere strengelyden som er kjent fra en tidligere oppgave.

Legg merke til at lyd-formatet i `MatInf1100Sound`-klassen bruker sampler av type `short` (det er det som for eksempel brukes på CD), og ikke flyttall i intervallet $[-1, 1]$ slik som forutsatt i kapittel 4 i kompendiet. Du må derfor passe på at dataene dine før avspilling blir omgjort til `short` og at støyen du legger til er skalert opp slik at den gir en effekt. (Verdiene til en `short` kan variere i intervallet $[-32768, 32767]$.) Med `wget`-kommandoen følger det noen lyd-filer som du kan teste programmene dine på.

a) Lag et filter som legger støy på en gitt lyd slik som foreslått på side 57 i kompendiet. Du kan eventuelt justere faktoren 0.2 slik at støyen blir godt hørbar. For informasjon om generering av tilfeldige tall kan du ta en titt på programmet `testrandom.java` som også følger med `wget`-kommandoen. Beskriv kort den hørbare effekten dette filteret har på en lyd.

b) Lag et lavpassfilter som demper de høye frekvensene slik som angitt på side 58–59 i kompendiet ved å erstatte et sampel med et veiet gjennomsnitt av samplet selv og de tre naboene på hver side. Som koeffisienter skal du bruke tallene i rad 6 i Pascals trekant dividert med summen av tallene, altså en naturlig generalisering av de to siste filterne på side 59 i kompendiet. Vær forsiktig i endene av signalet. Beskriv kort den hørbare effekten dette filteret har på en lyd.

c) Lag et høypassfilter som demper de lave frekvensene slik som foreslått på side 60 i kompendiet, men erstatt filteret med et filter av samme lengde som filteret i oppgave 1b). Bruk koeffisienter med samme tallverdi som i 1b), men la fortegnet til koeffisientene være positivt og negativt

annenhver gang. Beskriv kort den hørbare effekten dette filteret har på en lyd.

d) (Bare for dem som har lyst!) Skriv et program som spiller av de 8 tonene i A-dur skalaen. Bruk en samplingsrate på 8000 og la hver tone vare i 0.4 s med en pause på 0.1 s mellom hver tone.

Oppgave 2. Manuell flerskala-analyse

I denne oppgaven skal vi manuelt gjøre en flerskala-analyse av et datasett. Dette vil være nyttig bakgrunnsinformasjon for arbeidet med oppgave 3. For å gjøre denne oppgaven bør du først lese kapittel 10 i kompendiet.

a) Vi har datasettet $(x_i, c_i)_{i=0}^8$ som er gitt ved

$$x_i = i - 4 \quad \text{for } i = 0, \ldots, 8$$

og

$$c_i = x_i^2 \quad \text{for } i = 0, \ldots, 8.$$

Dette er altså 9 punkter som er hentet fra grafen til parabelen $y = x^2$ på intervallet $[-4, 4]$. Vi følger seksjon 10.2.1 i kompendiet og lar $f(x) = L_1(c_0, \ldots, c_8)$ betegne den stykkevis lineære funksjonen som tilfredstiller betingelsene $f(x_i) = c_i$ for $i = 0, \ldots, 8$.

Bruk Lemma 10.2 og dekomponer f i en tilnærming g og en feilfunksjon e i henhold til formelene (10.1)–(10.4). Skisser de tre funksjonene f, g og e.

b) Sett $g_1 = g$ og $e_1 = e$ og dekomponer g_1 på samme måte som $g_1 = g_2 + e_2$. Dekomponer deretter g_2 som $g_2 = g_3 + e_3$. Vi har dermed $f = g_3 + e_3 + e_2 + e_1$. Skisser funksjonene g_3, e_3 og e_2.

c) Ved kompresjon setter vi alle koeffisienter i feilfunksjoner som i tallverdi er mindre enn en gitt toleranse ϵ lik null, slik som beskrevet i seksjonene 10.4 og 10.5 i kompendiet. Forsøk å gjøre kompresjon først med toleransen $\epsilon = 3$ og deretter med toleranse $\epsilon = 16$. I hvert tilfelle får du da feilfunksjoner som er litt endret og som vi kaller $\tilde{e}_i$ for $i = 1, 2, 3$. Regn ut de to tilsvarende tilnærmingene til f gitt ved $\tilde{f} = g_3 + \tilde{e}_3 + \tilde{e}_2 + \tilde{e}_1$, skisser disse og sammenlign med f. Hvilken grad av kompresjon fikk du i de to tilfellene?

Oppgave 3. Kompresjon av lyd

I denne oppgaven skal du programmere en versjon av toleransekompresjon av et lydsignal slik som skissert i kapittel 10 i kompendiet og demonstrert i oppgave 2.

Oppgaven består i å programmere dekomposisjon slik det er angitt i formlene (10.3) og (10.4) i kapittel 10 og sette de av w'ene som i tallverdi er mindre enn en gitt toleranse (som du selv velger) lik null. Du skal også programmere rekonstruksjon ved hjelp av formlene (10.5) og (10.6).

Programmene dine skal først testes på det enkle datasettet fra oppgave 2. Deretter skal du teste på en testlyd ved at du dekomponerer testlyden og lagrer resultatet på fil, bruker programmet `gzip` for å kode fila effektivt og sammenligner størrelsen på den kodede fila med størrelsen på fila med den opprinnelige testlyden. Du skal også rekonstruere den endrede lyden, spille den av og gi en subjektiv vurdering av kvaliteten. Du skal ha minst et slikt testeksempel der størrelsen på den ferdigkomprimerte fila er betydelig mindre enn størrelsen på fila med testlyden, samtidig som det er uproblematisk å gjenkjenne testlyden fra den komprimerte lyden.

a) Programmer en forenklet dekomposisjonsalgoritme der du bare dekomponerer en gang slik at f blir spaltet til $f = g_1 + e_1$. Programmer også rekonstruksjonsalgoritmen. Programmer først tilfellet der antall sampler er et oddetall, men koden din bør også fungere om antallet er et partall. Toleransen kan du enten lese inn fra terminalen eller legge inn som en konstant i programmet.

Test dekomposisjons- og rekonstruksjonsmetodene dine på parabeldataene ved hjelp av programmet `test_parabola.java` med toleranse lik 0, 3 og 16. Sjekk at resultatene stemmer med det du regnet ut for hånd i oppgave 2 og rapporter resultatene. Test deretter med en lyd ved hjelp av `CompressionTester.java` og gi en vurdering av kvaliteten.

b) (For dem som har lyst.) Programmer en fullstendig dekomposisjonsalgoritme som dekomponerer signalet ditt k ganger (du skal selv angi eller regne ut k) og sette alle w_i'er som i tallverdi er mindre enn toleransen til null. Programmer også tilsvarende rekonstruksjonsalgoritme. Husk at det ikke er sikkert at signalet ditt har en lengde som gjør at du kan gjøre mange dekomponeringer slik som foreslått i kapittel 10. Seksjon 10.3.5 diskuterer hva du kan gjøre med det.

Tips og gjennomføring

Ved å bruke kode andre allerede har skrevet blir jobben din med denne oppgaven redusert til en brøkdel av hva det ellers ville vært. Hvis du har utført `wget`-kommandoen som er vist på hjemmesida som beskriver oblig2 har du fått lastet over en del Java-programmer. For å løse denne oppgaven vil du ha glede av filene

```
AbstractCompressedSound.java
Compressed1100Sound.java
CompressionTester.java
MatInf1100Sound.java
forslaga.java
forslagb.java
test_parabola.java
```

Fila `Compressed1100Sound.java` inneholder en svært primitiv dekomposisjonsmetode, men fungerer sammen med `CompressionTester.java`. Filene `forslaga.java` og `forslagb.java` inneholder skjeletter som kan være nyttig for å løse heholdsvis oppgave 3a og 3b.

For oppgave 3a) bør klassen inneholde en konstruktør som dekomponerer lydsignalet en gang, en metode som rekonstruerer og fire variable

```
short d[]; //Samplene til g
short w[]; // Samplene til feilfunksjonene
double sampleRate;
short eps; //Toleranse
```

Konstruktøren tar som parameter en lyd av typen `MatInf1100Sound` og skal dekomponere lyden i henhold til algoritmen gitt ved formlene (10.3) og (10.4) i kompendiet. Fila `forslaga.java` inneholder altså mer informasjon.

For oppgave 3b) bør klassen inneholde en konstruktør som dekomponerer lydsignalet flere ganger, en metode som rekonstruerer et signal og de fire variablene

```
short d[]; //Samplene til
short w[][]; // Feilfunksjonen(e)
double sampleRate;
short eps; //Toleranse
```

Eneste forskjell fra 3a) er altså at vi nå får en array av feilfunksjonssampler. Fila `forslagb.java` inneholder mer informasjon.

Det er enklest om du bare har en versjon av `Compressed1100Sound.java` liggende på filområdet der du arbeider. Det kan derfor lønne seg å ha et filområde for hver av oppgavene.

For å teste programmet ditt kan du bruke `CompressionTester.java`. Dette programmet vil generere en testlyd (eventuelt lese en lyd fra fil) på `MatInf1100Sound`-format som det skriver ut på fila `original.obj`, kalle konstruktøren til `Compressed1100Sound` og derved utføre dekomposisjonsalgoritmen du har skrevet, utføre rekonstruksjonen og spille av det rekonstruerte signalet og til slutt skrive ut det dekomponerte signalet på fil . Du skal så bruke `gzip` for å gjøre eksakt kompresjon av fila som ble generert, slik som beskrevet i instruksjonene på `CompressionTester.java`.

Merk spesielt at hvis du forsøker å bruke en testlyd som varer mer enn noen sekunder må du la Java få mer hukommelse under kjøring. Oppskrift for hvordan det gjøres finner på begynnelsen av `CompressionTester.java`.

Takk Pål!

Helt til slutt: Takk til Pål Hermunn Johansen (dr. gradsstudent i matematikk) som har skrevet programkoden dere bruker her og som dere forhåpentligvis har stor glede av!

Lykke til!!