Programmering av firkantpuls

Knut Mørken

5. oktober 2004

I en av programmeringsoppgavene i MAT-INF 1100 til uke 41 (oppgave 5.2) skal funksjonen P_0 gitt ved

$$P_0(t) = \begin{cases} 1, & \text{når } 0 \le t < 1/2; \\ -1, & \text{når } 1/2 \le t < 1; \end{cases}$$

(se side 71 i kompendiet) brukes til å generere lyd. Tanken er å erstatte den pene, glatte sinus-funksjonen med noe kantete som også oscillerer. Det mest naturlige er å finne en erstatning for $\sin 2\pi t$ siden denne funksjonen gjennomløper en full periode når t varierer i intervallet $[0, 1]$. Funksjonen P_0 over er nettopp denne erstatningen for $\sin 2\pi t$ når t ligger i $[0, 1]$, men hvordan skal P_0 være for andre verdier av t? Vi vet at $\sin 2\pi t$ er en periodisk funksjon og vi utvider derfor også P_0 til å være periodisk. Dette betyr at hvis $t \in [n, n + 1)$ for et eller annet heltall n så skal $P_0(t)$ være 1 om t er nærmest n og -1 om t er nærmest $n + 1$. For å uttrykke dette mer kompakt kan vi bruke den såkalte gulv-funksjonen ('floor function' på engelsk) som er definert ved

$$\lfloor x \rfloor = \max\{n \in \mathbb{Z} \mid n \le x\}.$$

Denne funksjonen er tilgjengelig i Java ved navnet `Math.floor(t)`. Ved hjelp av denne kan $P_0(t)$ programmeres som (det fins mange andre måter å gjøre dette på)

```
public static double p0(double t)
     {
     if (t-Math.floor(t) < 0.5) return(1.0);
           else return(-1.0);
     }
```

(Merk at det fins en tilsvarende funksjon som i Java har navnet `Math.ceiling`. Den oppfører seg som forventet.)

Når du skal bruke funksjonen P_1 til å generere lyd må denne også først utvides periodisk til hele tallinjen. Deretter kan du programmere den ved hjelp av et par `if`-tester.

Tips om konvertering til `short`. Når jeg først skriver noen linjer om lyd er det en ting til som bør nevnes. I kompendiet er det antatt at verdiene som skal spilles av ligger i intervallet $[-1, 1]$ mens avspillingsprogramvaren i `MatInf1100Sound` tar i mot sampler i en `short`-array som er det vanlige i lyd-sammenheng. Som vist på forelesning hører vi ingenting om vi forsøker å spille av verdier som ligger i intervallet $[-1, 1]$, lydvolumet blir for lavt. For å rette på dette må vi forsterke lyden, noe som rent matematisk betyr å skalere opp alle verdier med en fast konstant. For å holde programmeringen rimelig ryddig vil jeg anbefale å bruke `double`-variable i utregningen av sample-verdiene. Når så alle beregninger er unnagjort kan du multiplisere alle verdier med største `short`-verdi og konvertere til `short`. Hvis du for eksempel har regnet ut sampleverdiene og lagt dem i `double`-arrayen `x` kan du konvertere det hele til passende store `short`-verdier som lagres i arrayen `data` ved å si

```
int max = Short.MAX_VALUE;
short data[];
data = new short[x.length];

for (i=0; i<x.length; i++)
{
data[i] = (short) max*x[i];
}
```