Enkel ljudanalys i Matlab

De grundläggande objekten i Matlab är matriser och vektorer. En vektor är en lista med tal, t.ex. [2,3,5,7,11], som man kan spara i en variabel. Skriv in följande i kommandoraden för att konstruera en variabel som heter 'p' och som innehåller de fem talen i listan:

```
p = [2,3,5,7,11];
```

Man kan utföra operationer på hela talvektorn istället för på enskilda tal, som att multiplicera varje element med en konstant:

```
1.5*p
ans =
3.0000 4.5000 7.5000 10.5000 16.5000
```

I det här fallet skrivs svaret ut på skärmen, men vi kunde också ha tilldelat det till en ny variabel. Man kan också addera två vektorer förutsatt att de har lika många element. För att ta reda på hur många element vektorn innehåller, skriv `length(p)`. Vad är summan av elementen i `p`? I matematisk notation skulle man skriva

$$\sum_k p_k$$

och i Matlab kort och gott `sum(p)`. Man kan accessera enskilda element i en vektor med hjälp av parenteser och en address till det elementet. Adressen är ett heltal som börjar på 1.

```
p(1)
ans =
2
```

Hur man gör en sinuston

Först behöver vi en tidsvariabel, som vi kallar t. Låt säga att variabeln t är indelad i hundra steg från 0 till 1. Skriv in följande i kommandoraden:

```
t = [0 :  0.01 :  1]
```

Det här är ett effektivt sätt att ange att listan ska börja på talet 0.0, öka med en hundradel, och fortsätta upp till 1.0. Matlab ska då skriva ut en lista med alla tal 0, 0.01, 0.02, 0.03 osv. För att slippa se den långa listan med tal, lägger man på ett semikolon i slutet av kommandot. En sinuskurva beskrivs av funktionen

$$x(t) = A\sin(2\pi t)$$

där A är amplituden. Vi kan definiera konstanten A som vi vill, t.ex.

```
A = 0.25;
x = A * sin(2*pi*t);
```

När tidsvariabeln t löper från 0 till 1 får vi en hel period av sinustonen. För att plotta signalen x räcker det att skriva:

```
plot(x)
```

Lägg märke till att grafen anpassas automatiskt så att hela sinuskurvan ryms i plotten. Vilken enhet har tidsaxeln i det här fallet, och varför?

För att spela upp en sinuston behöver vi bestämma en samplingsfrekvens och definiera om tidsvariabeln så den rymmer, förslagsvis, en sekund ljud. Vi väljer 48 kHz som samplingsfrekvens och tilldelar den konstanten i en ny variabel (sr):

```
sr = 48000;
t = [1:1:sr];
```

Nu innehåller t istället sampelnummer från 1 till 48000, dvs en sekund. För att skapa en sinuston på 440 Hz definierar vi frekvensvariabeln f och sätter in samplingsfrekvensen i uttrycket för signalen:

```
f = 440;
x = A * sin(2*pi*f*t/sr);
```

För att bara se på ett segment av den här signalen kan vi accessera till exempel de tusen första samplena:

```
plot(x(1:1000))
```

Tidsaxeln har nu alltså enheter av sampel. Det vore praktiskt att ha enheten sekunder istället.

Då behöver vi skalera tidsvariabeln genom att dela den med samplingsfrekvensen. Då kommer tidsaxeln istället att löpa från 0 till 1 och ha enheten sekunder.

```
T = t/sr;
plot(T, x)
```

Lägg märke till att T och t är olika variabler, man skiljer på små och stora bokstäver. Funktionen plot har nu två argument, det första bestämmer punkter på den horisontala axeln och det andra på den vertikala axeln. Båda argumenten måste vara vektorer av *samma* längd.

Vi vill förstås också gärna kunna höra på sinustonen. Spara den i så fall i en wav-fil.

```
wavwrite(x, sr, 'fantastic_sinusoid.wav')
```

För att läsa in ljudfiler (fortfarande bara i wav-formatet) finns kommandot `wavread`. Bruksanvisning finns i hjälpfunktionen:

```
help wavread
```

Man kan också spela av ljudfiler direkt med `sound` (fungerar nog bäst på windows). Ange önskad samplingsfrekvens om du inte vill höra ljudet avspelat med 8192 Hz som samplingsfrekvens.

```
sound(x, sr);
```

Spektralanalys

Det finns en FFT-funktion i Matlab som beräknar spektrumet av en signal. Vi tar samma signal x som ovan, som vi vet är en sinuston, och sparar dess spektrum i variabeln X:

```
X = fft(x);
```

Nu är spektrumet en serie med komplexa tal som representerar amplituden för varje spektralkomponent. Det är framför allt amplitudspektrumet som är intressant i ljudanalys. Det motsvarar magnituden på dessa komplexa tal, och fås genom att ta absolutvärdet av talet.

```
F = abs(X);
plot(F)
```

Amplitudspektrumet är symmetriskt kring mitten, som är nyquistfrekvensen. Därför kan vi nöja oss med att plotta den första halvan.

```
N = length(F)/2
F = abs(F(1:N)); % F innehåller nu amplitudspektrumet
plot(F)
```

Distorsion av det slaget som man får i gitarrförstärkare kan simuleras (på ett ungefär) med en ickelineär funktion som hyperbolisk tangens. Gör en spektralanalys av signalen $y(t) = \tanh(Gx(t))$, där x är en sinusoid och G är en konstant som är större än 1. Vilka deltoner kommer fram i spektrumet?

Spektrogram är också lätt tillgängligt. Funktionen `spectrogram` tar signalen som ska visas som första argument, men använd hjälpfunktionen för att ta reda på vilka andra nyttiga parametrar man kan ställa in.

```
S = spectrogram(x);
imagesc(S)
```

Glissando

För oss som spelar slide whistle, såg, hawaii-gitarr och liknande instrument är glissandot ett viktigt uttrycksmedel. Vi kan göra ett uppåtgående glissando genom att öka frekvensen på sinustonen över tid. Först behöver vi tidsvariabeln t för att stega igenom sinusfunktionen, sedan kan vi använda samma variabel en gång till för att öka frekvensen. Därför kan vi helt enkelt multiplicera t med sig själv. Antag att variabeln sr fortfarande är den samplingsfrekvens vi vill använda. Skapa en tidsvariabel som går från 0 till 1 under en sekund med den givna samplingsfrekvensen:

```
t = [0:  1/sr :  1];
```

För att multiplicera två vektorer elementvis i Matlab använder man en operator som består av en punkt följd av tecknet för multiplikation:

```
t2 = t .* t;
```

Försök själv med en kortare vektor och skriv ut resultatet (dvs ta bort semikolonet ovan) för att se om det stämmer. Ett glissando från 0 till 440 Hz genererar vi så här:

```
x = sin(440*2*pi*t2);
```

Undersök signalen x med någon av de metoder vi har gått igenom: 1) skriv ut signalen som en lista

med tal 2) plotta signalen i tidsdomänen 3) plotta amplitudspektrumet 4) plotta sonogrammet, eller 5) skriv den till en ljudfil eller spela upp den direkt och lyssna på den.

Mera om vektorer och matriser

Matriser är det grundläggande dataformatet i Matlab. En matris är en tabell med tal, till exempel

$$\mathsf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

som är en matris med två rader och tre kolonner (*søyler* på norsk). För att ange att en variabel är en matris med flera rader använder man semikolon mellan elementen i varje rad:

```
M = [1,2,3;4,5,6]
```

De vektorer vi har använt hittills har i själva verket varit matriser med bara en rad. En sådan radmatris kan göras om (eller *transponeras*) till en kolonnmatris (en vektor där elementena står i en enda rad) genom att lägga till en apostrof efter variabelnamnet:

```
p = [0,1,2]
q = p'
size(p) % ska vara 1, 3
size(q) % ska vara 3, 1
```

Kommandot `size` anger antal rader och sedan antal kolonner i variabeln. När man multiplicerar två matriser eller vektorer måste deras format stämma överens. För att multiplicera en matris A med en annan matris B, måste antalet kolonner i A vara likt antalet rader i B. Undersök vad som händer om man försöker beräkna följande uttryck där `p` är som ovan. Varför får man ett felmeddelande i ett av dem?

```
p * p
p' * p
p * p'
```

I samband med autokorrelation och olika transformer av signaler är *inreprodukten* viktig (se Sethares bok, kap. 5). Inreprodukten av två vektorer $\boldsymbol{u}$ och $\boldsymbol{v}$ av längd N är

$$u_1 v_1 + u_2 v_2 + \cdots + u_N v_N$$

där u_1 är första elementet av vektorn $\boldsymbol{u}$, osv. I Matlab skulle man kunna räkna ut inreprodukten genom kommandot

```
s = sum(t .* t)
```

men det är lättare att utföra samma sak genom att transponera ena vektorn först:

```
s = t * t'
```

Att den metoden fungerar beror alltså på att vektorer i själva verket är det samma som radmatriser i Matlab.

Alternativ

Matlab är enkelt att använda, men tyvärr inte gratis. Men det finns flera goda alternativ med öppen källkod (FOSS, Free and Open-Source Software) där man kan utföra mycket av det samma som i Matlab – utom att använda MIR Toolbox! Octave har nästan exakt samma syntax som Matlab. Ett annat program är Scilab, vars syntax skiljer sig något mer från Matlab. Programmet R används ofta för statistik, men är utmärkt för plot av data. R skiljer sig ännu mer från de andra programmen. Grundtanken om att utgå från vektorer är emellertid den samma i alla dessa program, och kan man ett av dem är det inte svårt att sätta sig in i ett annat.