2 SSP2012 / Assignment 2:

2.1 Scripting a melody

In Western music we are using the octave with its twelve notes "a-a#-b-c-c#-d-d#-e-f-f#-g-g#": the next note would be an "a" again. To label all notes we also have to know from which octave they were taken. Each note corresponds to a frequency. In our notation the "a" has a frequency of 440 Hz. The frequencies of the notes in an octave have equal distances on a *logarithmic frequency scale* and can be calculated as $f_k = f_0 2^{k/12}$ if we start counting from $k = 0$ to $k = 11$ and f_0 is the base frequency. Check: $k = 12$ gives $f_{12} = 2f_0$ and $k = -12$ gives $f_{-12} = f_0/2$.

A very simple song goes like this: $cdec - cdec - efg - efg - ga_1gfec - ga_1gfec - cg_0c - cg_0c$, where $a_1 = 2a$ and $g0 = g/2$, i.e. they fall in the higher and lower octave.

1. Make a script that plays this song, don't make all notes of the same duration.

2. Generalize the script to play a score. The musical score consists of an array with frequency duration pairs like for example:

   ```
   score$ = "'c' 0.4 'd' 0.3 ..."
   ```

   You can use the function `extractWord$(text$, precursor$)` to extract frequencies and durations from a text variable. If you use this function make sure that for the next extraction you either change the text$ or the precursor$ otherwise you will get the same number extracted over and over again.

Steps:

- Calculate the frequencies of the notes, e.g. a = 440, b = a*2^(2/12), etc...
  Watch out: don't assign values to a variable named "e" it is a constant just like pi.
- Compose the score: score$ = "'c' 0.4 'd' 0.3 'eh' 0.3 ...."
- Create a large enough sound of say 5 s duration.

Process the score to extract the (frequency, duration) pairs and do a Formula to synthesise the frequency

```
startTime = 0
repeat
  extract a f,d pair
  endTime = startTime + d
  put the tone f in the sound between startTime and endTime
  startTime = endTime
until last in score
```

2.2 DFT

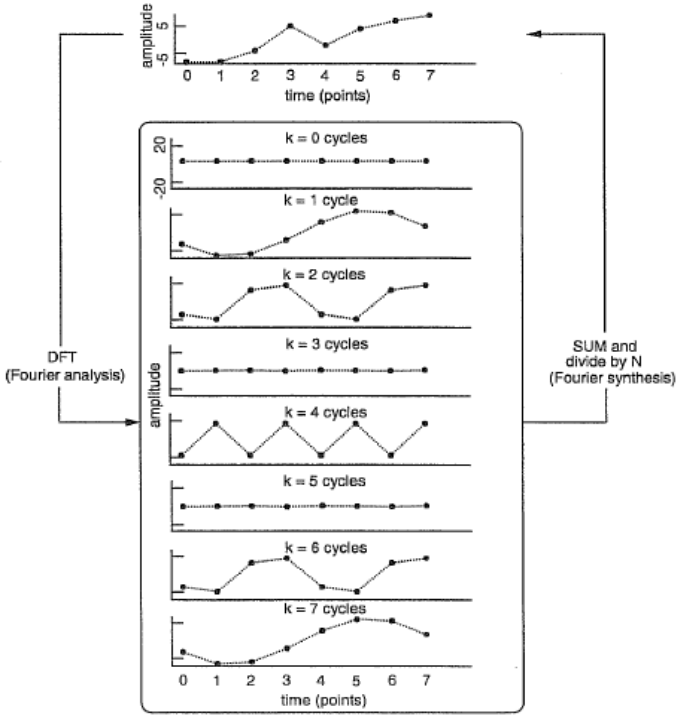

Figure 6.5: An 8-point time-signal (*top panel*) and its decomposition into sinusoids (inside the rectangle). This decomposition is known as *Fourier analysis* and is accomplished by the discrete Fourier transform. When this process is reversed (*Fourier synthesis*), the sinusoids are summed point by point (and the result is also divided by N, which is 8 in this case). Summation results in an exact reconstruction of the original digital time signal.

Reproduce the decomposition of the 8-point signal $s = (-8, -8, -4, 5, -2, 4, 7, 9)$ from the figure above (book of Harrington & Cassidy (1999)). This means a plot with 8 sinoids below each other. These 8 sinoids, when added together, should reproduce the signal s exactly.

- In Praat each sound sample represents the (average) amplitude in an interval of duration T seconds, where T is the sampling time. The first sample is at time $T/2$ and sample s_i therefore at time $T/2 + (i-1)T$. In this assignment we only care for sample numbers and we don't care about times, so sampling frequency is totally irrelevant and can be chosen as you wish. To associate time in *seconds* with sample *numbers* you might choose "Start time" as -0.5 and "End time" as 7.5 (for the 8 samples in the example), with a "Sampling frequency" of 1.

- You can set the value at a sample number with the "Modify>Set value at sample number..." command.

- The DFT is give by $H_n = \sum\{a_k \cos() + b_k \sin()\}$. The a_k and b_k can be obtained by multiplying the signal s with the corresponding cos() or sin() and taking the average value (mean).
  The following code fragment summarizes how to obtain the a_i for the i^{th} component:

```
select Sound s
Copy... cos'i'
Formula... cos(2*pi*i*x/n)*self
ai = Get mean... All 0 0
```

- For the sine part you do something analogous to find the b_i.

- The signal to display can then be calculated with the formula:

```
Formula... ai*cos(2*pi*i*x/n) + bi*sin(2*pi*i*x/n)
```

- The "printline" command can be handy to display intermediate results in the Info window. For example, to print out the values for a_i and b_i with four digit precision use something like

```
printline a'i', b'i': 'ai:4', 'bi:4'
```

- Because we don't have many points it is nice to also draw dots at the sample points:

```
Draw... 0 0 -1 1 no Speckles
Draw... 0 0 -1 1 no Curve
```