A multi-resolution sinusoidal model

Week 10 Assignment

Steve Dwyer

Freesound link to the two sounds chosen

I selected two sounds from freesound, which are polyphonic and have both melodic and percussive components:

- https://freesound.org/people/Setuniman/sounds/141185/
- https://freesound.org/people/frankum/sounds/353773/

In order to process these using *utilFunctions.wavread()*, these sounds needed to be mixed down to mono, and resampled to 44100 Hz. These resampled and remixed sounds have also been included in the zip file, for the reader's convenience.

Explanation and justification of the band edges and the window sizes for each sound

1. Sound `141185_2244250-lq.wav`:

Analysing Sound 1 with Sonic Visualiser revealed that the lowest fundamental in this track is +/- 87.3 Hz (F2) and the highest 1637.48 Hz (G#6 – 25c) Harmonics were visible up to the bin frequency 11046 Hz – 11068 Hz, after which there was no energy visible in the spectrogram.

Based on the frequency of the lowest fundamental, I performed a Sine Model Analysis & Synthesis using the sms-tools GUI, with Window Type blackman, and window size $M = \frac{6\times44100}{87.3} = 3030.92\cdots = 3031$ (rounded up to the nearest whole number), FFT size $N = 4096$ (next power of 2 larger than M) and hop size $H = \frac{3031}{4} = 757.75 = 757$ (rounded down to the nearest whole, odd, number).

From the synthesis produced with these parameters, it was evident that the percussive elements were missing from the reconstruction. This is clearly due to the lack of resolution of the sharp attacks of the percussion instruments. Examining the spectrogram and the plot of the YIN Pitch Detection algorithm in Sonic Visualiser, there appear regular 'beats' at about 350 Hz (F4) and at about 950 Hz (A#5). In the expectation that these 'beats' represent the percussive elements which would have a much sharper attack, I decided to initially work with the following three frequency bands:

$\mathrm{B1}: 0 <= \mathrm{f} < 340\mathrm{Hz}$, $\mathrm{B2}: 340 <= \mathrm{f} < 900$, $\mathrm{B3}: 900 <= \mathrm{f} < 22050$.

The corresponding window sizes, using a blackman window, should be:
$M1 = \frac{6\times44100}{87.3} = 3030.92\cdots = 3031$ (rounded up to the nearest whole number),
$M2 = \frac{6\times44100}{340} = 778.24\cdots = 779$ (rounded up to the nearest whole, odd, number), and
$M3 = \frac{6\times44100}{900} = 294 = 295$ (next higher odd number)

The FFT sizes used for the analysis were chosen to be in each case the corresponding next power of two larger than the window size, i.e. 4096, 1024 & 512 respectively.
These parameters were used for an explorative analysis & synthesis using the Sinusoidal model in the sms-tools GUI, and the lower window size produced a much more sharply defined attack in the percussive components, although, as expected, the lower frequencies were much less well resolved.

The results from the multi-resolution sinusoidal analysis & synthesis can be heard in the output file `A10-b-1.wav`, provided in the zip file submitted with this report.

2. Sound `353773_2305278-lq.wav`:
   The second piece I selected is a longer piece of music (2 min 24 secs), designed as a music background for games, and contains some nice flanging effects, so I'm curious to see how the sinusoidal model handles these. Using a similar approach as above on this track, I found the lowest fundamental to be about 52 Hz (G#1). This 'bass line' appears to progress between 52 Hz and about 98 Hz (G2). Above this pattern, there's a second pattern visible (the 'melody'), which progresses between 117 Hz (A#2) and about 235 Hz (around A#3). Further, more sparsely dispersed fundamentals can be seen above this 'melody' band, which roughly, but not consistently follow the 'melody' track, leading me to conclude that these are the sharp attacks from both harmonic and percussive components.

   I decided to work with the following three frequency bands

   $\mathrm{B1}: 0 <= \mathrm{f} < 100\mathrm{Hz},\ \mathrm{B2}: 100 <= \mathrm{f} < 250,\ \mathrm{B3}: 250 <= \mathrm{f} < 22050.$

   The corresponding window sizes, using a blackman window, should be:
   $M1 = \frac{6\times44100}{52} = 5088.46\cdots = 5089$ (rounded up to the nearest whole, odd, number),
   $M2 = \frac{6\times44100}{100} = 2646 = 2647$ (next higher odd number), and
   $M3 = \frac{6\times44100}{250} = 1058.4 = 1059$ (rounded up to the nearest whole, odd, number)

   The FFT sizes used for the analysis were again chosen to be in each case the corresponding next power of two larger than the window size, i.e. 8192, 4096 & 2048 respectively.

   After performing the multi-resolution sinusoidal analysis & synthesis with these parameters, the results were quite satisfying. I did notice however, that the flange effect in the reconstructed output file had lost some of the dirty, distorted, analogue character that the original had, but was still clearly audible, but as a cleaner flange effect, without distortion.

   The output file from this multi-resolution sinusoidal analysis & synthesis is `A10-b-2.wav`, which is also provided in the zip file submitted with this report.

Observations about the advantages of a multi-resolution analysis

- Time-frequency resolution
  Using a shorter window size at higher frequencies allows us to capture the sharper attacks of the percussion instruments, while the longer window sizes help maintain the resolution of lower frequencies in the harmonic instruments. This can be clearly heard in the rendered output files.

  Obviously, at the very start & end of the track, due to the fact that only the longest window can take samples from the first & last $\frac{\mathrm{M1}-\mathrm{M2}}{2}$ samples, the time resolution will be compromised, leading to sharp attacks not being well resolved in these periods. Given typical window sizes though, we're only talking about periods lasting a few hundredths of a second, which in my opinion can be considered negligible.

- Computational complexity
  Since the complexity of the FFT is $\vartheta\ N \log N$[1], where N is the FFT size (see https://en.wikipedia.org/wiki/Fast_Fourier_transform), and in the original *sineModel* method, this is calculated for each frame, which results in $n = x.size\ /\ H$, where H is the hop size, executions of the FFT, which is linear with respect to $x.size$, and would therefore approximate $\vartheta\ n \times N \log N$.

  Since the IFFT has the same complexity as the FFT, then we can surmise that the complexity of the entire original *sineModel* method would be twice that of performing only the analysis or

[1] The Big ϑ notation, see https://en.wikipedia.org/wiki/Time_complexity.

synthesis part separately. Since $2n$ is still linear to n, we can surmise that the complexity of the entire original *sineModel* method is still *in the order of magnitude of* $\vartheta\ n \times N \log N$.

Since the *sineModelMultiRes* method will need to perform the FFT on the input signal 3 times, with varying analysis window sizes and FFT sizes, and the IFFT once, per hop, one can surmise that the running time would be *approximately* twice that of the non-multi-resolution *sineModel* method. So, although the multi-resolution *sineModel* will clearly take longer to compute, the number 2 is still a constant, so its running time would not be adversely affected by the length of the input signal or window sizes, when its FFT was to be run 3 times across different windows of varying sizes. Effectively, the complexity, or running time of this method is still dependent on the length of the input signal and the complexity of the FFT.

In conclusion, the complexity of the multi-resolution *sineModel* method *sineModelMultiRes* would still be *in the order of magnitude of* $\vartheta\ n \times N \log N$.

In practice, I found that the actual execution time of the multi-resolution sinusoidal model using three windows was just under twice that of performing the *sineModel* with a single window.

- Extensions to HPR and HPS models
  Since the HPR & HPS models call the three core methods which are called by *sineModelMultiRes* as part of the analysis (*dftModel.dftAnal*, *utilFunctions.peakDetection* & *utilFunctions.peakInterp*), plus the additional calls to *utilFunctions.f0Twm* (f0 detection using the two-way mismatch algorithm) and *harmonicModel.harmonicDetection*, extending this multi-resolution functionality to these models would essentially result in the same changes being made to the hprModel's *hprModel* and hpsModel's *hpsModel* methods – changing the parameters to allow for lists of windows, FFT sizes and band widths to be passed and then iterating over these and executing the core analysis for each window, before extracting the relevant frequency band information for the synthesis.

Challenges one might face if one was to extend it to HPR and HPS models

- Sinusoid tracking
  The *sineModel* method adapted for this assignment, *sineModelMultiRes*, has been developed for real-time processing, so is unable to track sinusoids due to the requirements of being able perform post-processing, in order to identify the existence of a sinusoidal frequencies across multiple frames.

- F0 estimation
  With similar changes made to *hprModel.hprModel* and *hpsModel.hpsModel* methods to allow these methods to process multiple window sizes, then extract the relevant frequency bands, in order to build up a multi-resolution sinusoidal model prior to f0 detection and harmonic detection, f0 detection can be achieved. However, as both the *hprModel.hprModel* and *hpsModel.hpsModel* methods, just as *sineModelMultiRes*, have been developed for real-time processing, f0 detection can only be achieved with monophonic signals.