answers

February 11, 2016

1 CSC 475

1.1 Andrew Hobden V00788452

2 Question 1

2.1 Q1.1 (*)

A simple fundamental frequency estimation is to compute the magnitude spectrum, select the highest peak, and return the corresponding frequency as the result. Processing the sound in windows will result in a time series of F0 estimates that can be plotted over time. In Marsyas the following `MarSystems` would need to be connected: `SoundFileSource`, `Windowing`, `Spectrum`, `PowerSpectrum`, `Selector`, `MaxArgMax` and `Accumulator`. Check that your method works by using as input a sine. Show the F0 plot for qbh `examples.wav` with this method when using windows of 1024 samples at 22050 sampling rate

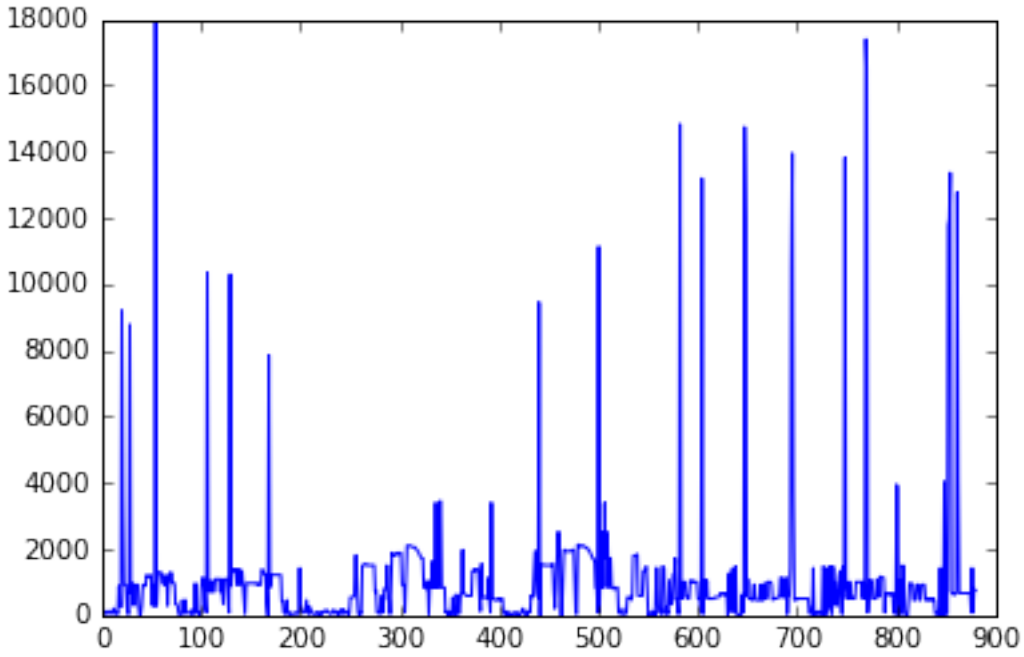
```
Series {
    inSamples = 1024
    -> input: SoundFileSource { filename = "qbh_examples.wav" }
    -> Windowing { size = 1024 }

    -> Spectrum
    -> PowerSpectrum { spectrumType = "magnitude" }
    // It's 513x1 right now and useless.
    -> Transposer
    -> max: MaxArgMax
    // Get rid of the val, only care about index
    -> Transposer
    -> selection: Selector { disable = 0 }

    -> sink: CsvSink { filename = "q1.1.csv" }
    + done = (input/hasData == false)
}
```

```
In [8]: % pylab inline
        import marsyas

        msm = marsyas.MarSystemManager()
        y_data = []

        system = marsyas.system_from_script_file("q1.1.mrs")
        system.getControl("SoundFileSource/input/mrs_bool/hasData").to_bool()
        while (system.getControl("SoundFileSource/input/mrs_bool/hasData").to_bool()):
            system.tick()
            y_data.extend(system.getControl("Selector/selection/mrs_realvec/processedData").to_realvec()
```

```
            y_data[-1] *=  22050 / 513 # Sampling Rate / FFT Size

        plot(range(0, len(y_data)), y_data)
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
Out[8]: [<matplotlib.lines.Line2D at 0x7fe2bd9749d0>]
```



2.2 Q1.2 (*)

Copy your network and modify it to use Autocorrelation for the f0-estimation. Plot the new contour and compare it with the previous one. Is one consistently better than the other?

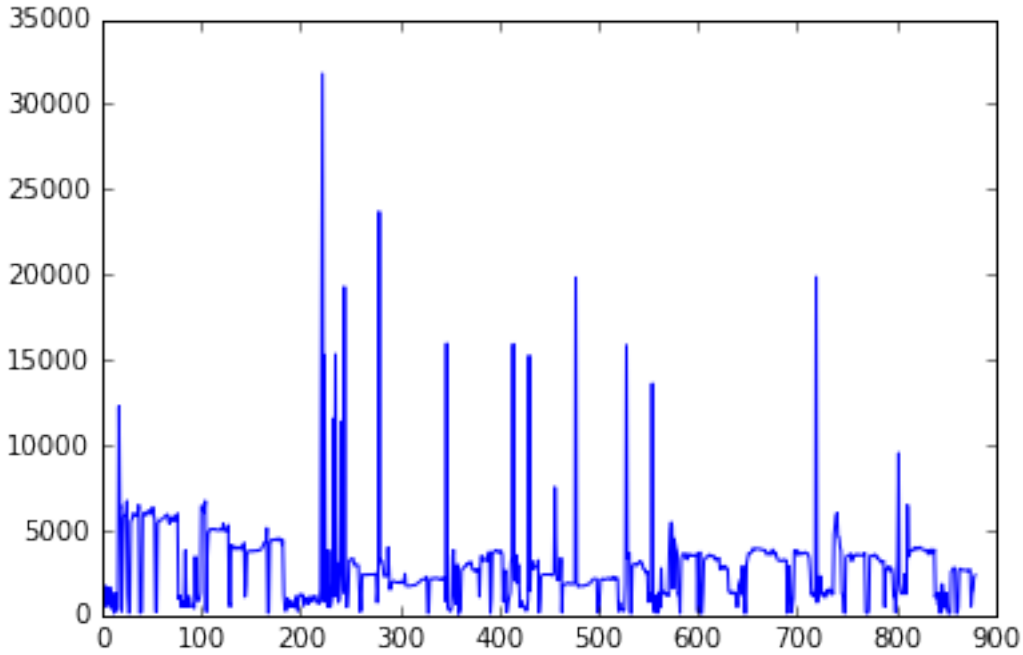
```
Series {
    inSamples = 1024
    -> input: SoundFileSource { filename = "qbh_examples.wav" }
    //-> SineSource
    -> Windowing { size = 1024 }

    //-> Spectrum
    //-> PowerSpectrum { spectrumType = "magnitude" }
    // It's 513x1 right now and useless.
    //-> Transposer
    -> AutoCorrelation
    -> Peaker
    -> MaxArgMax
    // Get rid of the val, only care about index
    -> Transposer
    -> selection: Selector { disable = 0 }
```

```
    -> sink: CsvSink { filename = "q1.2.csv" }
    + done = (input/hasData == false)
}
```

```
In [9]: % pylab inline
        import marsyas

        msm = marsyas.MarSystemManager()
        y_data = []

        system = marsyas.system_from_script_file("q1.2.mrs")
        system.getControl("SoundFileSource/input/mrs_bool/hasData").to_bool()
        while (system.getControl("SoundFileSource/input/mrs_bool/hasData").to_bool()):
            system.tick()
            y_data.extend(system.getControl("Selector/selection/mrs_realvec/processedData").to_realvec()
            y_data[-1] *=  22050 / 513

        plot(range(0, len(y_data)), y_data)
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x7fe2bd8d5290>]
```



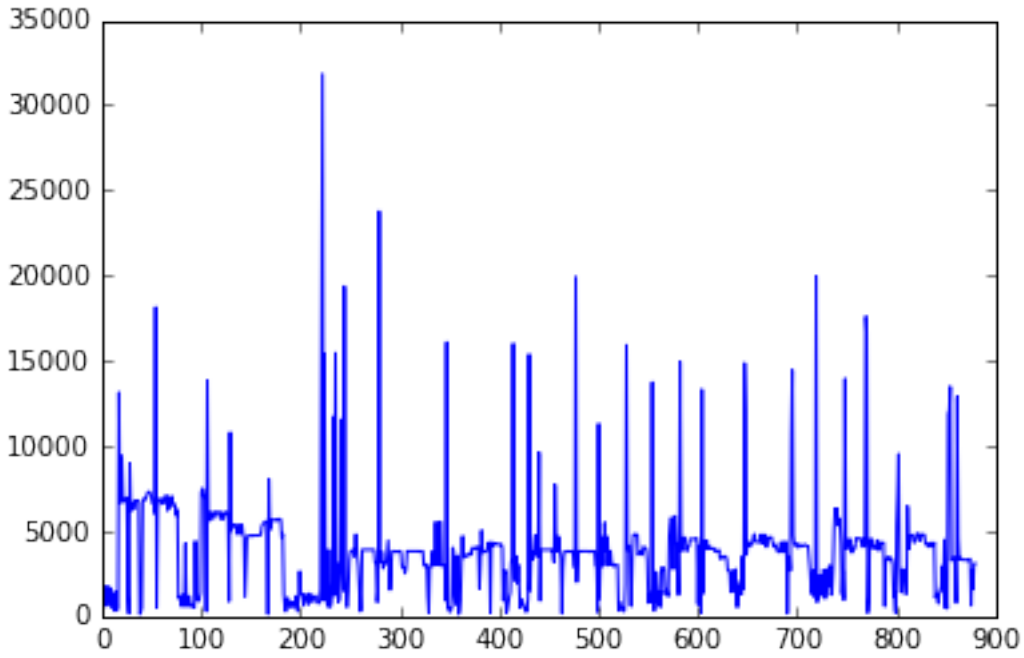
Is it consistently better? Perhaps, it has much less "big peaks" so I presume it is more accurate. I did notice a close sequence of peaks around the 200 mark which I think is an inaccuracy.

2.3 Q1.3 (**)

Change your code so that both the F0 estimates are computed for every window. Plot the sum of the two resulting F0 contours. The computation should be done in one pass through the audio file. In Marsyas use a Fanout after the Windowing followed by a Sum.

```
Series {
    inSamples = 1024
    -> input: SoundFileSource { filename = "qbh_examples.wav" }
    -> Windowing { size = 1024 }
    -> Fanout {
        // Q1a
        -> Series {
            -> Spectrum
            -> PowerSpectrum { spectrumType = "magnitude" }
            // It's 513x1 right now and useless.
            -> Transposer
            -> max: MaxArgMax
            // Get rid of the val, only care about index
            -> Transposer
            -> selection_1: Selector { disable = 0 }
        }
        -> Series {
            //-> Spectrum
            //-> PowerSpectrum { spectrumType = "magnitude" }
            // It's 513x1 right now and useless.
            //-> Transposer
            -> AutoCorrelation
            -> Peaker
            -> MaxArgMax
            // Get rid of the val, only care about index
            -> Transposer
            -> selection_2: Selector { disable = 0 }
        }
    }
    -> summer: Sum
    -> sink: CsvSink { filename = "q1.3.csv" }
    + done = (input/hasData == false)
}
```

```
In [10]: % pylab inline
         import marsyas

         msm = marsyas.MarSystemManager()
         y_data = []

         system = marsyas.system_from_script_file("q1.3.mrs")
         system.getControl("SoundFileSource/input/mrs_bool/hasData").to_bool()
         while (system.getControl("SoundFileSource/input/mrs_bool/hasData").to_bool()):
             system.tick()
             y_data.extend(system.getControl("Sum/summer/mrs_realvec/processedData").to_realvec())
             y_data[-1] *=  22050 / 513

         plot(range(0, len(y_data)), y_data)

Populating the interactive namespace from numpy and matplotlib

Out[10]: [<matplotlib.lines.Line2D at 0x7fe2bd818490>]
```



3 Question 2

The purpose of this question is to experiment with the audio feature called spectral centroid using Marsyas.

3.1 Q2.a (*) Preparation

Download and install Marsyas, and learn the basics with the help of the online tutorial: http://marsyas.info/tutorial/tutorial.html. Don't hesitate to contact your course instructor or TA for help.

In the following steps you will create a Marsyas script that reads an audio file, computes the spectral centroid of each window of the short-term Fourier transform (STFT), and plays back both the original audio file and a sine wave that follows the spectral centroid frequency. The structure of the script will be very much like the example in the "Control links" section of the tutorial: http://marsyas.info/tutorial/tutorial.html#control-links.

However, instead of applying the RMS energy to the amplitude of a noise generator, you will apply the spectral centroid to the frequency of a sine wave generator. It is a good idea to start with the example code and modify it to achieve the goal of this question.

Note that the same experimentation can be done with the Marsyas Python bindings or the Marsyas C++ library, and you are welcome to do so if you prefer.

In any case, please provide your entire code for this question.

```
In [6]: # I did those things and I can see that the next questions
        # build off the example, so my code will be included with them.
```

3.2 Q2.2 (**) Compute the spectral centroid

This step replaces the RMS energy computation. First compute the STFT with a window size of 512 samples and 1/2 window overlap. Do this by combining a ShiftInput, a Spectrum, and a PowerSpectrum MarSystem. Then add a Centroid MarSystem. Smooth the centroid stream by applying an "audio feature

texture window": compute the mean of the last 20 values using a Memory and a Mean MarSystem. Finally, convert the result to a control stream using a FlowToControl MarSystem.

```
Series {
    inSamples = 256
    -> input: SoundFileSource { filename = "qbh_examples.wav" onSamples = 256 inSamples = 256 }
    -> MixToMono
    -> ShiftInput { winSize = 512 }
    -> Fanout {
        -> Series {
            -> Spectrum
            -> PowerSpectrum
            -> Centroid
            -> Memory { memSize = 20 }
            -> Mean
            -> centroid: FlowToControl
        }
        -> Series {
            -> SineSource { frequency = (centroid/value * (input/israte / 2)) }
            -> AudioSink
        }
    }
    + done = (input/hasData == false)
}
```

3.3 2.3 (**) Apply the centroid to the frequency of a sine wave

This step replaces the playback of noise that follows the RMS energy. Use a SineSource MarSystem to generate the sine wave. Create a link between the centroid control stream and the frequency control of the SineSource.

However, the Centroid MarSystem produces a number between 0 and the number of power spectrum bins. This needs to be converted to a frequency value in Hz between 0 and the Nyquist frequency. Use a control expression to do the conversion using the current sample rate. The sample rate is available as the "israte" control of any MarSystem (in this case SineSource itself).

3.3.1 Same as above

```
Series {
    inSamples = 256
    -> input: SoundFileSource { filename = "qbh_examples.wav" onSamples = 256 inSamples = 256 }
    -> MixToMono
    -> ShiftInput { winSize = 512 }
    -> Fanout {
        -> Series {
            -> Spectrum
            -> PowerSpectrum
            -> Centroid
            -> Memory { memSize = 20 }
            -> Mean
            -> centroid: FlowToControl
        }
        -> Series {
            -> SineSource { frequency = (centroid/value * (input/israte / 2)) }
            -> AudioSink
        }
```

```
    }
    + done = (input/hasData == false)
}
```

3.4 2.4 (***) Experiment with music genres

Download one or two classical music files and metal music files from the corresponding folder of: http://marsyas.cs.uvic.ca/sound/genres/

Run two instances of your centroid sonification script at the same time (using two command-line windows) to compare the results of pairs of different input audio files:

- classical and classical
- classical and metal
- metal and metal

To some extent this would be what an automatic genre classification system based purely on the Spectral Centroid would "hear" to make a decision of classical or metal. FYI using just the mean of the spectral centroid a trained classifier makes the right decision 75% of the time. This is impressive given that decisions are made every 20 milliseconds so even better results could be obtained by some majority voting/filtering over the entire file.

To see the influence of the texture window comment out the MarSystems Memory and Mean. Provide a very brief commentary (no more than 3-4 sentences) of your observations on this sonification experiment.

3.4.1 Answer:

I found it very confusing to play the sounds at the same time and noted that none of the provided audio files sound particularly like music when used via `sfplay`. Audacity was used instead.

However I did note audible differences between the classical and metal tracks, and noted similarities between the classical & classical, and metal & metal tracks. I'm not really sure how to explain them. The metal tracks sounded similar in rate and style, and different than the similar classical ones.

Removing the Memory and Mean produces more "jitter" in the signal and makes it much less smooth. It almost seems to speed it up.

In [ ]: