CS3220: Digital Audio and Applications

Project 1 Report

Dulcie Jackson

Program Features

The nargin function returns the number of arguments passed into the function. If this is less than the expected 5, I have set the default values required.

The function checks whether the signal x is stereo or mono, by comparing the number of elements in x to its length. If the number of elements is greater than the length, this shows that the array is multi-dimensional, and the boolean stereo is set to true.

The algorithm checks the detectionMode provided. If this value is 'peak', the function performs peak conversion by iterating over x, and converting non-zero values to decibels using the formula:

$$x_{dB} = 20 * \log_{10}(|x|)$$

If the value is 'RMS', the function calculates the sum of the squares values in a window for each item, divided by the window length. These values are converted to decibels using the formula above.

Gain reduction is calculated for all values greater than the threshold T, using the formula:

$$gain\ reduction_{element} = xdB_{element} - (T + \frac{xdB_{element} - T}{R})$$

giving the amount by which the value is reduced by the compression formula:

$$x_c[n] = T + \frac{x_{db}[n] - T}{R}$$

For stereo signals, the maximum gain reduction from the two channels is applied to both channels.

The compression gain is calculated as x_c - x_db. Make-up gain is added to this and stored in g_db. This is converted back from dB using the formula:

$$g = 10^{\frac{g_{db}}{20}}$$

The compressor output, y, is set to x * g.

RMS Mode

Method for Computing RMS Volume

If the detectionMode is 'RMS', the sliding window method is used. The windowLength is set to 4, since this is the default value used by the sliding window methods in MATLAB. A new array, x_squared, is created, containing the squares of the original array x.

The function loops through all values up to the length of x_squared - these being the values for the mono signal, or the first channel of a stereo signal - and calculates the sum of the values in the window, storing this in the x_db array. This is done using the sum() and min() functions, taking the sum from the current x_squared array value, to the minimum of either windowLength values along from the current x_squared value, or the full length of the x_squared array. This overcomes the issue of having less values left in the array than the windowLength, acting in the same way as padding the window with zeros for the nonexistent items.

The function then checks if the signal is stereo using the boolean set previously, and if so repeats this operation with the loop from the length of x_squared + 1 to the total number of elements in x_squared - these are the values in the second channel of a stereo signal. It calculates the sum of the values in the window using the same method as before, although taking the minimum of either windowLength values along from the current x_squared value, or the number of elements in x_squared. All values are stored in the correct positions in the x_db array.

Finally, the values in x_db are each divided by the windowLength, to give an average value of the window elements.

Justification of Algorithm Selected

I chose to use the sliding window method to compute the RMS, as this gives the actual values rather than an approximation. Additionally, the MATLAB documentation suggests

using this method to analyse a finite duration of data[1], and using the exponential weighting method for embedded applications. Since this compressor receives a finite amount of data in the form of the array x, I felt that the sliding window method was preferable in this case. Whilst the exponential weighting method is more efficient, I did not feel that this was a necessity, since the arrays I have been using are in general fairly short, and hence the algorithm runs adequately fast using the sliding window method.

Peak vs. RMS Detection

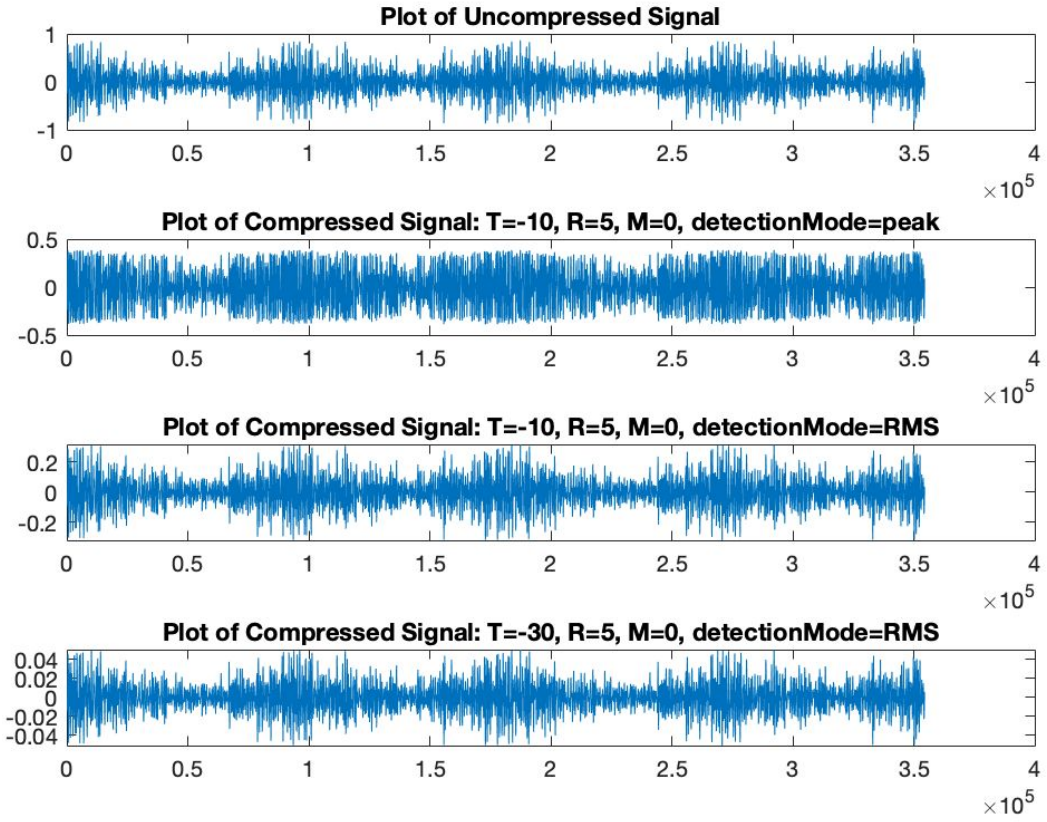
Peak detection uses the individual peaks in the signal, and converts these to decibels one after another. In contrast, RMS detection takes averages of blocks of the data, and applies these averages to each item within the signal. RMS detection can be more 'forgiving', and better when you intend to make tracks louder, whereas peak detection is preferred for volume docking. In RMS detection using the sliding window method, the result of the detection depends on the window length chosen, and with large window sizes, this can cause the overall signal to become flattened as the window averages tend towards the overall average of all peaks in the signal. Peak detection reacts to the individual peaks as they are encountered, and acts on an individual basis to compress these.

These changes in conversion of the signal to dB cause changes in the compression results. Peak detection causes all values in the signal to be individually compared with the threshold, meaning that specific large values will be compressed whilst other, smaller, values nearby will not undergo a change. Comparatively, RMS detection makes it more likely for groups of nearby values to be compressed similarly, since each value's neighbours are taken into account when it is converted to dB. This means that the result of RMS detection may more accurately match the original signal, since values in the signal are most likely compressed in groups, whereas peak detection allows for large variation from the original signal due to the individual basis of compression.

[1]MathWorks, 'Sliding Window Method and Exponential Weighting Method', Available at: https://www.mathworks.com/help/dsp/ug/sliding-window-method-and-exponential-weighting-method.html

Plots

Sound 1



This first set of graphs show a mono signal, compared with compressed variants.

Compression 1

Default values for the compressor were used by calling

```
compressedY = simpleCompressor(y);
```

The defaults for these parameters are set to T=-10, R=5, M=0 and detectionMode='peak'.

The has changed greatly, with the maximum y-axis value (the maximum signal amplitude)

halving. The waveform appears to be ‘flattened’ - there is less difference between the low and high points than in the original.

Compression 2

This version of the compressor used the default values, with ‘RMS’ as the detectionMode.

```
compressedY = simpleCompressor(y,-10,5,0,’RMS’);
```

Similarly to Compression 1, the signal has halved in amplitude, but the final signal appears more similar in shape to the original. This is because using peak detection, any value with a greater absolute value (linear) than

$$10^{\frac{T}{20}} = 10^{\frac{-10}{20}} \approx 0.316$$

will be compressed, whereas with RMS detection the window average is taken. This results in less peaks in the signal being compressed if the average of blocks of the signal tends to lower their value when converted to dB.

Compression 3

Here the compressor used the values

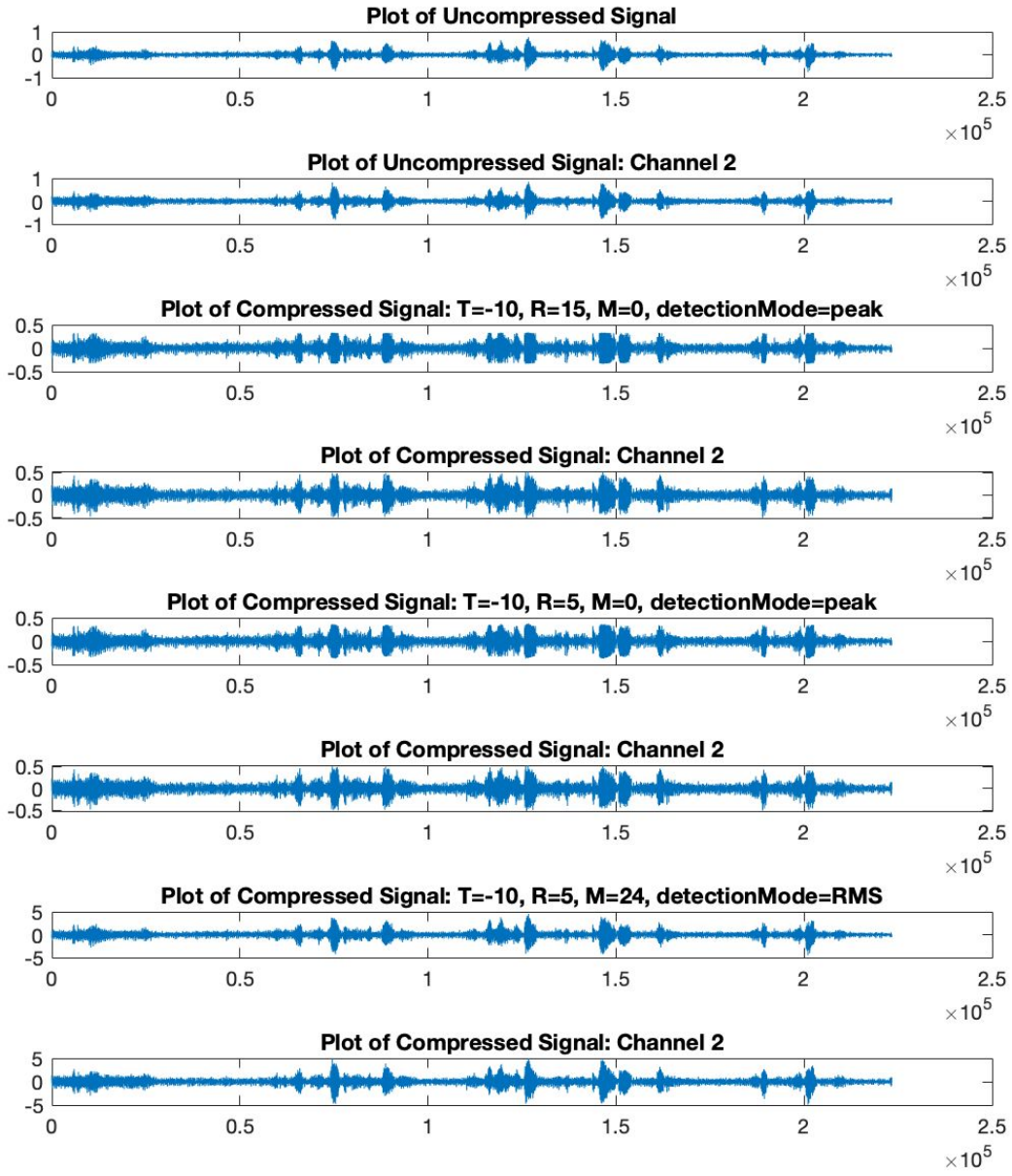
```
compressedY = simpleCompressor(y,-30,5,0,’RMS’);
```

This demonstrates the effect of the T value, as the maximum value on the y-axis is reduced to 0.04 from 1. The compressed signal looks similar to the original, however the significant drop in amplitude can be explained by the formula

$$x_c[n] = T + \frac{x_{db}[n] - T}{R}$$

Since T is a large negative, the result of this calculation will be also, which means that the resulting signal will resolve to a very small linear value.

Sound 2: Stereo



The original signal in this case is a stereo signal, and hence the compressor works in stereo linking mode to produce the desired effect.

Compression 1 and 2

The first compressor uses values

```
compressedY = simpleCompressor(y,-10,15,0,'peak');
```

All the defaults are used, except for R, which takes a value three times the size of its default. This affects the equation

$$x_c[n] = T + \frac{x_{db}[n] - T}{R}$$

which correspondingly increases the compression gain, meaning that the final linear domain signal is slightly larger than with a smaller value of R, as can be seen from the comparison of Compression 1 to Compression 2. Other than this, the waveform is of very similar shape to that in Compression 2.

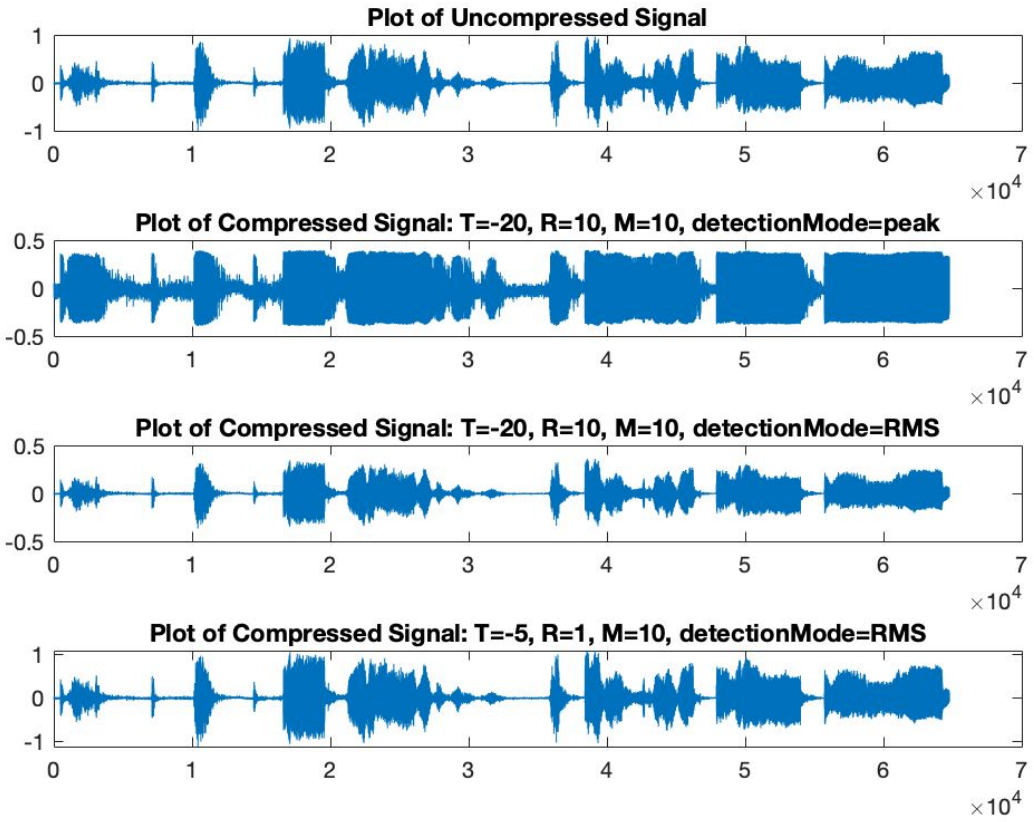
Compression 3

Compression 3 was created with the call

```
compressedY = simpleCompressor(y,-10,5,24,'peak');
```

to show the result of changing M to its maximum of 24 from the default of 0. This causes the signal to have much larger amplitude, with a maximum of 5 rather than 1. Otherwise, the compressor values are of identical shape to those in Compression 2, as all of the values have been increased by a constant value.

Sound 3: Mono



The compressor changes from the default were made to multiple parameters for this mono signal. These examples are more varied than the previous sets of waveforms.

Compression 1 and 2

These examples show the substantial difference between peak and RMS detection with large values for R and M, produced using the calls

```
compressedY = simpleCompressor(y,-20,10,10,'peak');
```

and

```
compressedY = simpleCompressor(y,-20,10,10,'RMS');
```

The peak version appears to be extremely distorted compared to the original signal, with a higher proportion of high amplitudes (relative to the highest amplitude of the signal). The RMS version, however, maintains a very similar shape to the original, except for the

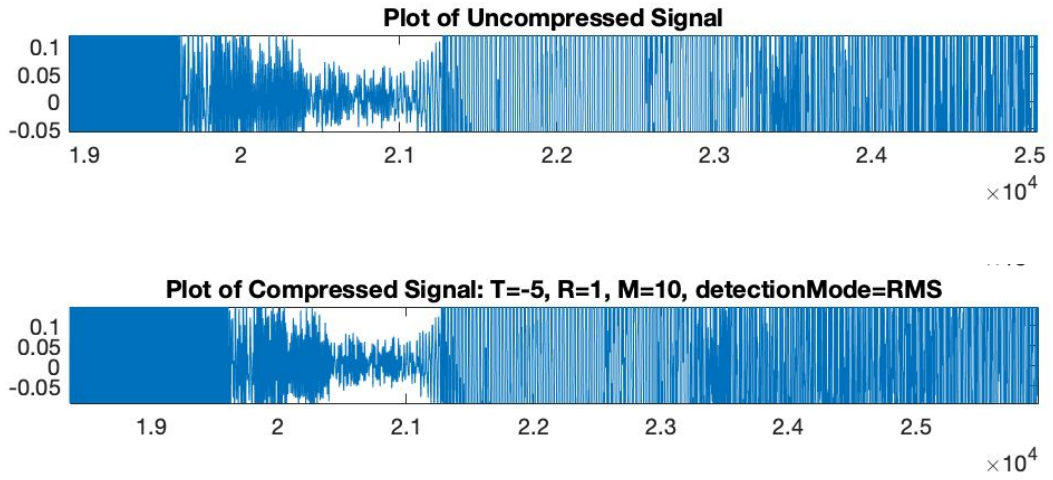
difference between the highest and lowest amplitude values in the signal being lessened. This demonstrates the 'gentler' approach of RMS detection when compared to peak detection.

Compression 3

Here the compressor produced a near-identical signal to the original. This version was called using

```
compressedY = simpleCompressor(y,-5,1,1,'RMS');
```

There are very slight differences to the signal when viewed up close, such as a lack of definition in the compressed signal when compared with the original, as shown below.



This can be seen around the 2 mark on the x-axis, where the original signal appears less dark at the top and bottom (an impression created by lower-amplitude values throughout the signal). The compressed signal does not have this variation, as the window averages cause these lower-amplitude elements to be lost. M=1 gives the signal the same maximum amplitude as the original, and overall the waveform is practically indistinguishable from the original except when zoomed closely.

Conclusion

I feel that this project went well, as I was able to successfully implement a compressor which appears to work as expected, and which can accept both mono and stereo signals. I am particularly impressed with this, as I believe that I have correctly implemented stereo linking within my function. Additionally, I am pleased with my use of MATLAB's capability to handle arrays, as I feel that my function puts this to good use when performing operations on every element.

If I were to work on this project again, I would like to implement both the sliding window and exponential weighting methods for RMS detection, in order to investigate the difference in results and efficiency. I would also like to experiment by incrementally changing the windowLength, causing the averages to tend towards the average of the entire signal, and to see this visually through plots. This would be interesting in better understanding the sliding window method, whilst implementing various stereo linking methods would be fascinating to see if these provide noticeably different results.

Overall, I am happy with my work on this project, and feel that I have successfully implemented all necessary elements.

Bibliography

Creating Tracks (2014), 'Peak Vs. RMS Compression - The Perfect Pumping Effect! - Creating Tracks' [video], Available at: https://www.youtube.com/watch?v=1dVR5ZwKVm8

MathWorks, 'Sliding Window Method and Exponential Weighting Method', Available at: https://www.mathworks.com/help/dsp/ug/sliding-window-method-and-exponential-weighting-method.html

Wikipedia (2005), 'Dynamic range compression', Available at: https://en.wikipedia.org/wiki/Dynamic_range_compression#Stereo_linking