Matlab Project 4, Part 2: Signal Compression

EE-125

1 FFT-based compression

1. Load the built-in example file train.mat (using 'load train') which contains a train whistle. Plot and listen to the sound using `soundsc`; make sure to use the sampling rate you read in.
2. Take an FFT of the sound, using the next power of two up from the signal length
   a) Plot the magnitude of the spectrum vs. frequency (freq labeled in Hz, power in dB)
   b) Comment on nature of the signal.
3. Write a Matlab file `FFTcompression.m`, with first line

```
function Yout = FFTcompression(signal,percentRetained)
```

   where `signal` is the signal, and `Yout` is the FFT of the signal but with only the strongest components retained, where 'strongest' means FFT bin amplitudes with the largest absolute value. If `percentRetained` is 90, than the strongest 90% of the original FFT bins should be kept, and others zeroed out (set to zero). If `percentRetained` is 50, than the strongest 50% of the original FFT bins should be kept, and others zeroed out., etc. The reconstructed signal can be made using, for example, `yRecon = ifft(Yout)`.
4. Write a function `SNRoverall` that computes the overall SNR, with the first line being

```
function snr = SNRoverall(signal,reconSignal)
```

   where the SNR is defined as

$$SNR = 10log10\frac{\sum_{n=1}^{L} s(n)^2}{\sum_{n=1}^{L} \left(s(n) - r(n)\right)^2} \tag{1}$$

   where s is `signal` and r is `reconSignal`, both of length L.
5. Call the above functions to get the compressed spectra for 100, 50 ,20, and 10. For each, save the output spectra in the workspace (for example Y100, Y50, etc). For each, reconstruct the signal using ifft, and save the results in other variables called for example recon100, recon50, etc.

   a) Listen to the sounds. Does the compression hurt the sound quality much? If not, how far can you push things and still have a recognizable signal (can you discard >90% of the components?)

   b) Make a plot showing the original magnitude spectrum for trains in blue (basically, the plot you made in part 2a), but overlay on top of it the power spectrum Y10 (the spectrum with the 10% retained) in red. Can you relate this plot to the quality (or lack thereof) in the reconstructed sound?

   c) Compute and tabulate the SNR for each reconstructed signal.
6. Now, clear all variables from the workspace, and load onscreen.wav and repeat step 5. How do the characteristics of the various signals affect the amount of compression possible?

2 Storage savings due to compression

Look at the variables you have in your workspace using the whos command. This command lists both the size of each variable (how many numbers are stored) and the number of bytes (how much computer memory is used).

1. Compare the original signal to any of the frequency-domain version (Y100, Y50, etc). You should see that the FFT appears to have doubled your storage requirements. Is that really true? If you were going to save these variables to disk, how can you use your knowledge of Fourier transform properties to save storage? Would the amount of memory to store the frequency-domain signal be bigger, smaller, or the same as storing the time-series data, and why?

2. Now, compare the no-compression version of the spectrum (Y100) to the 90% compressed (Y10). You should see that the storage used for both is the same, even though 90% of the values in Y10 are zero! The reason for this is that so far, we are not being very clever we are storing the number 0 in each of the discarded FFT bins.

   A better approach is to instead store a) only the non-zero values, and b) a list of which frequency bin corresponds to each non-zero value. Because this is useful fairly often, some computer languages have built-in functionality to handle this. In Matlab, this are arrays are known as sparse arrays (i.e., mostly zero).

   Look at the file `playWithSparse.m` which is provided with the assignment. It should give you an idea how sparse arrays work in Matlab. Investigate this using the following:

   Define a sparse array Ysp which is the same size as Y10. Assign only the non-zero values of Y10 to Ysp. How do the sizes of Ysp and Y10 compare? You should see a substantial savings. Since Ysp only contains 10% of the entries of Y10, you might thing the size would be exactly 10x smaller. Is that true? If not, can you explain why?

3 Discrete Cosine Transform

The discrete cosine transform (DCT) is a close cousin of the DFT. Instead of representing signals as sums of complex exponentials, it represents them as sums of cosines (assuming signals are all real-valued). The link between the DFT and DCT is described in Section 7.5 of Proakis and Manolakis.

The DCT is implemented in Matlab using commands dct and idct which are very analogous to fft and ifft. However, Matlab uses different normalization than in our text book; read the help found under doc dct and doc idct.

Carry out the following steps (step 1 should get you better acquainted with the DCT)

1. Set up a vector D which is all zeros, and is 1024 samples long. Then do the following

   a) Set the first element equal to 1 (D(1) = 1), and take the idct,and plot the result.

   b) Reset D(1) to 0, then set D(2) = 1 and repeat the last step.

   c) Reset D(2) to zero, then set D(3) = 1 and repeat the last step.

   d) In your report, include the 3 plots and discuss how you can explain the results in terms of the `idct` documentation.

2. Now, write an m file `DCTcompression.m`, with first line

```
function Dout = DCTcompression(signal,percentRetained)
```

   This just a modified copy of your FFTcompression.m file, but using the DCT functions instead.

3. Call the above functions to get the compressed spectra for 100, 50 , and 20, and 10. For each, save the output spectra in the workspace (for example D100, D50, etc). For each, reconstruct the signal using idct, and save the results in other variables called for example dctRecon100, dctRecon90, etc.

   a) Compute and tabulate the SNR for each reconstructed signal. You should find that DCT compression gets slightly better SNR than FFT.

   b) Listen to the DCT vs. FFT-based compression with 20% of coefficients retained. You should be able to hear a slight difference, though probably not dramatic.

4 FYI: Real-world audio compression algorithms

Actual audio compression algorithms like MP3 use the basic ideas above, but with important modifications:

1. The signal is split into short frames, typically on the order of 50 msec. Each 50 msec data chunk is then compressed. Because the short section of signal is more likely to be sinusoidal than for the overall signal, the compression is better. The use of short frames is also important because audio files are often many minutes long, not just a few seconds.

2. The DCT is in fact used in MP3, but its part of a more complicated processing chain it is used to compress some intermediate results.

3. Psychoacoustic models are used to determine what parts of the sound you are likely to notice or not notice; the ones that are less noticeable are discarded. Thus, the discarding isnt just purely done based on the amplitude of the coefficient, as done here.