Question 1

All we needed to do here was apply our FT function from conv-fft to the original and synthesized signals to view their frequency spectrum. This required us to build a time domain for the two signals based on the sampling rate. Given the sampling rate in Hz, we know the time elapsed between two data points would be the reciprocal of the sampling rate. We used this as our time increment for the two time domains, building each to an equal length as its corresponding time signal.

The results are as pictured:

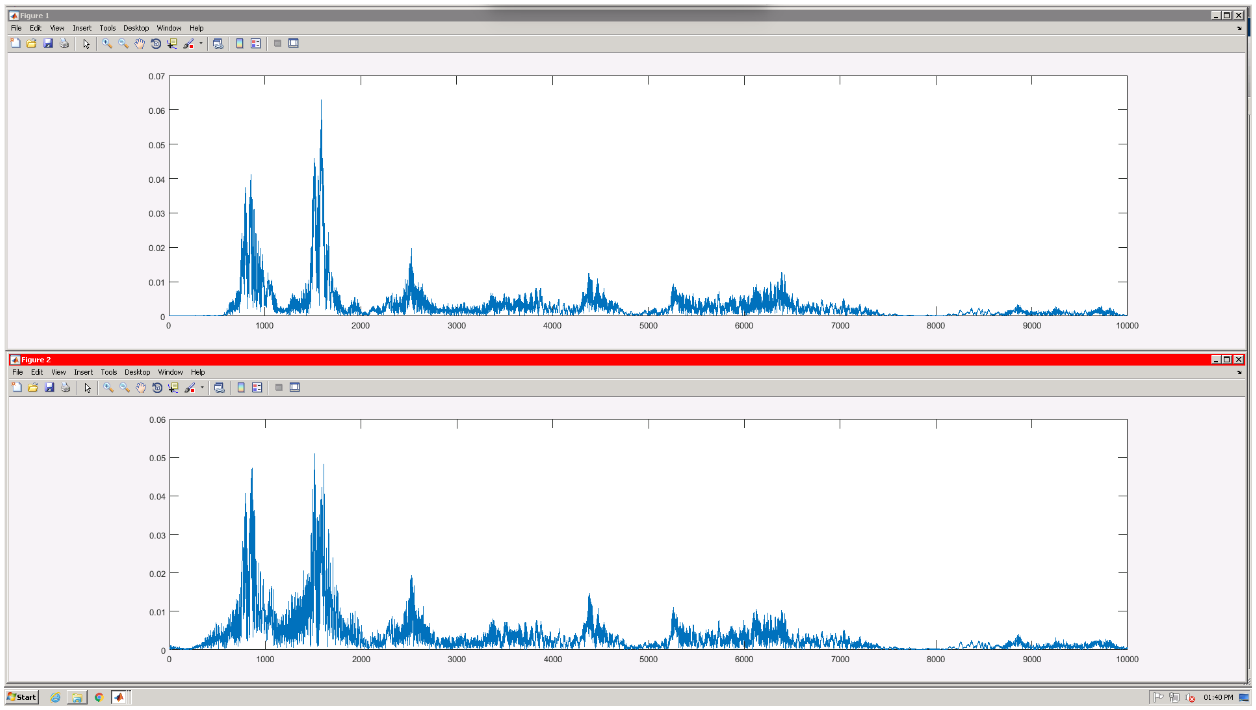

Figure 1 - Frequency spectrum of original signal (top) and synthesized signal (bottom). Domain 0 to 10,000 Hz.

Firstly, it is evident that these signals are very similar – as they should be, given that the second signal is the deconvoluted and re-synthesized version of the first. Both signals have large peaks around 900 and 1500 Hz, composing the primary frequency content of the voice signal. There are smaller peaks from 2500 – 7000 Hz, after which the spectrum dies out.

The primary noticeable difference in the spectrums is the quality of the synthesized signal. Specifically, there is more noise in the synthesized signal, especially visible between the peaks at 900 and 1500 Hz. This is likely due to some loss of information caused by the simplified deconvolution in getModel.m, and perhaps amplified by synthVoice.m.

Question 2

The property that be used to explain why tone is lowered when the audio is played at half speed is the time-scaling property.

According to this property, when a function is expanded in time by a factor of “a”, its Fourier Transform is compressed in frequency by “a”. In other words, ‘stretching’ the signal in time domain leads to a lower frequency, and ‘compressing’ in time leads to a higher frequency.

$x(t/a) \longleftrightarrow aX(\omega a)$

Lower frequencies have a lower pitch therefore an expansion in time will make the output signal sound lower and vice versa.

Question 3

The objective here was to make copies of x[n] compressed to half its original period (‘speeding up’) and expanded it to twice its initial period (‘slowing down’), using Matlab’s interp1 function. We were given the data points for x[n], but no explicit domain (ie time points) – however, we did not need an explicitly defined domain as it is arbitrary in the task of compressing and expanding the signal. For this reason, we used the signature of interp1 that used only two arguments, the signal and the query points (interp1(v, xq)); this uses a domain of points [1:n] where n=length(v).

To speed up the signal (compress it), we chose query points that would sample every other data point of v, by defining xq to be integer points spaced by two (ie 1, 3, 5, 7….). In this way, we compress the same shape of the data into half the buckets. The evident downside is that this is lossy compression, and noticeably makes the audio signal sound more ‘robotic’ (less smooth).

To slow down the signal (expand it), we did the opposite; we chose query points that would sample every 0.5 points (ie 1, 1.5, 2, 2.5, …). This put our original data into twice the buckets, so we had to perform the operation twice, once on the first half of the signal and again on the second half.

The resultant plots of the signals x, x_f and x_s are pictured to show the effect of these operations (Fig 2 through 4);

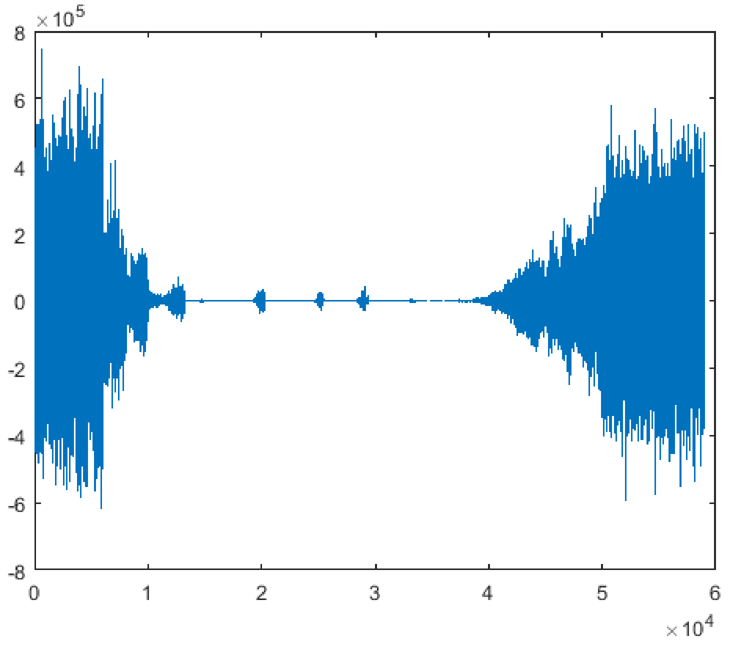

Figure 2 - x (original - note period of 6e4)

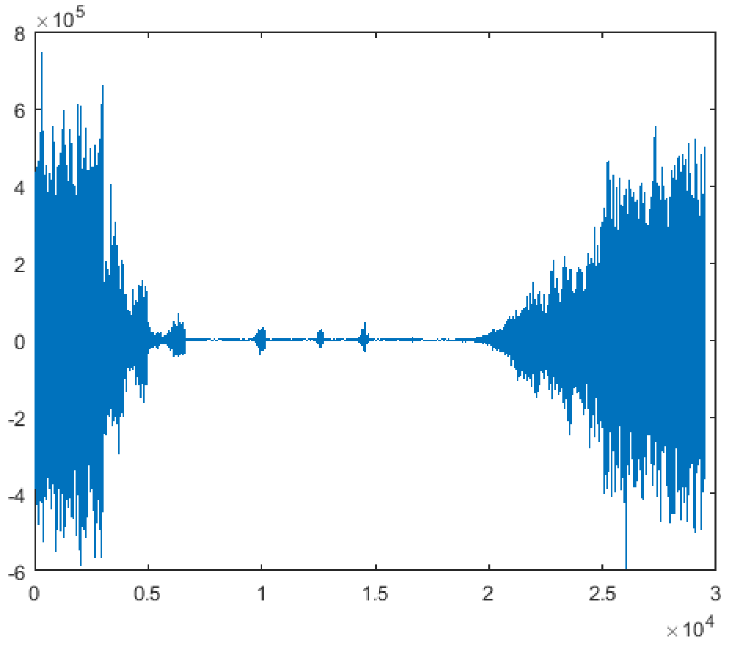

Figure 3 - x_f (sped up - note period of 3e4, half of x)

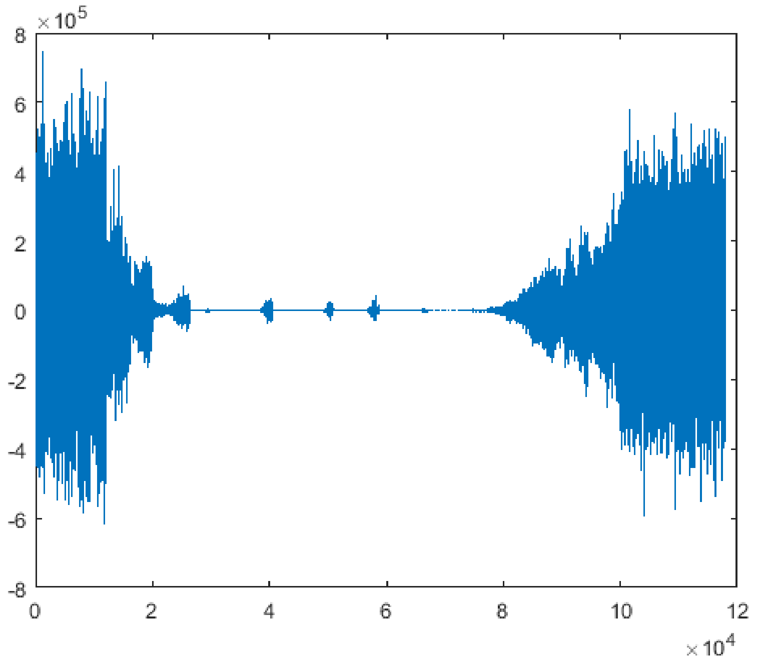

Figure 4 - x_s (slowed down - note period of 12e4, twice that of x)

Question 4:

In order to create a modified signal that has a modified tone without changing the speed, one needs to shift the angle of the poles for the input signal. When one shifts the poles towards higher frequencies without changing their relative positions, the output will have a higher tone. Inversely, when one shifts the poles towards lower frequencies, the output will have a lower tone.

Since we only need to do a rotation, the only relevant things to change are the pole's angles and not the magnitude of the poles. Given a list of the angles of the poles, one loops through them and checks if they are complex. If an angle is complex, one shifts it by a ratio of itself. If the angle is real, one does not shift the angle of change the magnitude of the pole. One does not shift a real pole so to keep the real conjugate pair. If one was to shift a real pole, one would have to create its conjugate pair which is not done due to this exercise being simplified by not wanting to increase the array size of the poles array.

After the shift takes place, one will synthesis the new poles by using the original magnitude and the new angle.