BLG 354E

Signals & Systems for Computer Engineering

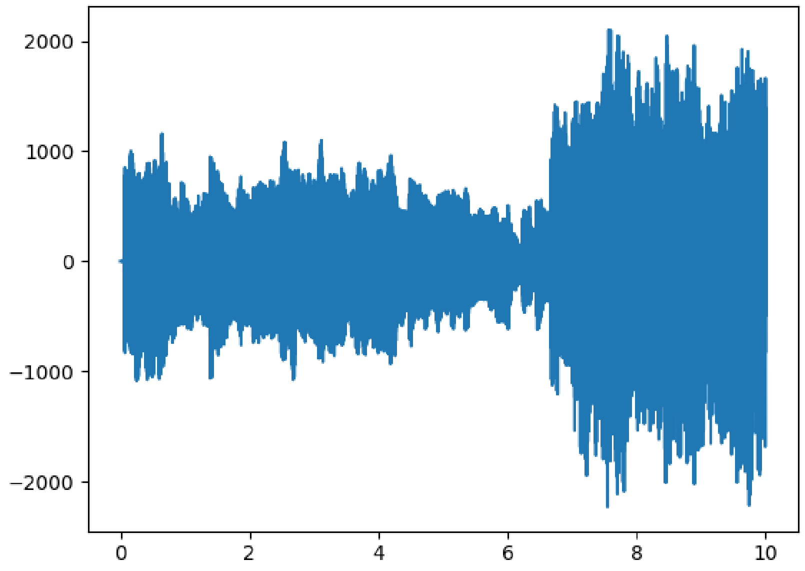
Ayser Ecem Konu
150160711

Output audios can be reached by using link given below

https://ituedutr-my.sharepoint.com/:f:/g/personal/konu16_itu_edu_tr/EowBX5fmDRNKvbHzF6MSawIBbpoHzLFfiCdNWRdPzYSMkQ?e=NOEvqe

Part 1

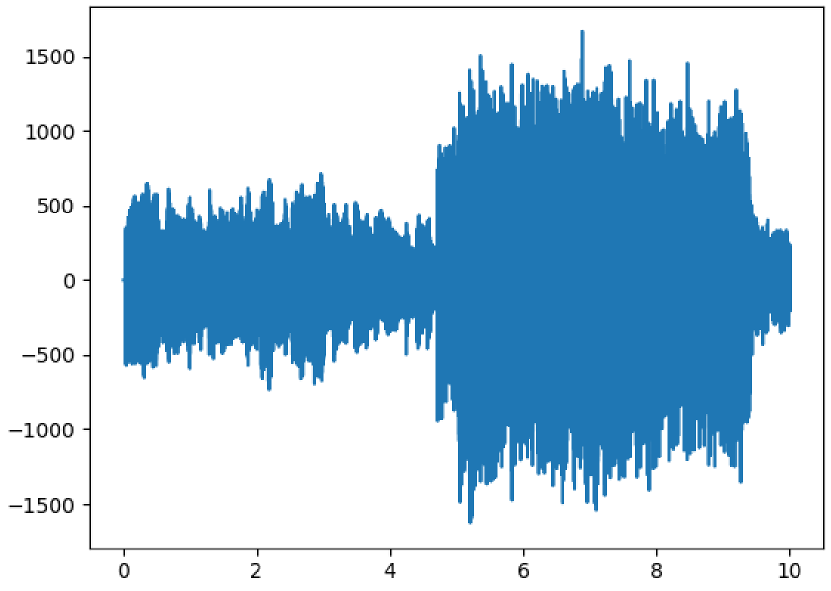
For part 1, I have implemented the code similarly to given code in matlab. Variable names are kept similar to show the purpose. Sample audio is stereo, a left channel and a right channel exists, in order to reduce the size of output file, I have averaged both channels and produced a mono channel signal instead. In this part of the homework, we are expected to increase pitch of signal by one. A pitch includes 2 semitones, therefore a variable named step size, $2^{1/6}$ is used as scaling factor that will lengthen the signal. In order to produce the pitched sound, signal will first be lengthened by scaling factor given as step size, then shortened using same step size in linear interpolation. Value of each step will be extracted from output signal. Hanning window is used in implementing the algorithm because it is a closer approximate for unit impulse than the regular window. Hanning window of 2n is down sampled to match expected frame size in order to meet a slower window signal decay rate. Input signal, named as sample in my implementation, will be divided into frames. Each frame will be multiplied with hanning window; result will be transformed using Fourier Fast Transform(FFT). Phase difference, deltaPhi will be calculated from transformed value. Frequency of the bin, energy of frame that does not change, will be removed from deltaPhi, and also the multiples of phase difference will be extracted by using modular arithmetic, since phase of $2\pi = 0$. Hop value, non-overlapping area ratio is chosen as frame number/4. Step size and hop value is used to calculate overlapping ratio used in synthesis stage. Frames will be fused using overlapping ratio, hopOut in the implementation. Magnitude of Fourier transformed frame is then transformed back, synthesized, using Inverse Fourier Fast Transform(IFFT). Real part of the transformed value will produce output for each index when it is multiplied with window again. Linearly interpolating this output using step size as step values for interpolation indexes, pitched output signal can be produced. Output signal is same length with input, however the notes are increased by 2 semitones. In order to run this code, pydub, scipy, matplotlib, scipy.io.wavfile and numpy packages, additionally mp3Read program that I have written is required. Mp3read uses pydub to convert mp3 file to wav. For pydub to run, ffmpeg should be installed and set in path variables.



Time vs Output Amplitude plot

Part 2

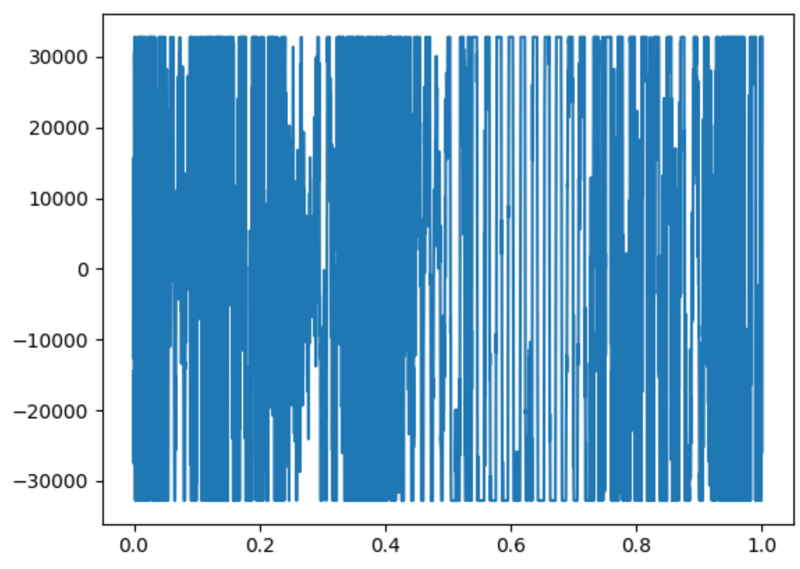
Same code in part 1 is used, however a secondary linear interpolation is used to decrease signal to a length by ~30%. However, in pitch shift I have increased the signal by 6 semitones. In the second linear interpolation, to reduce size by 30%, I have used a step size of $2^{1/2}$.



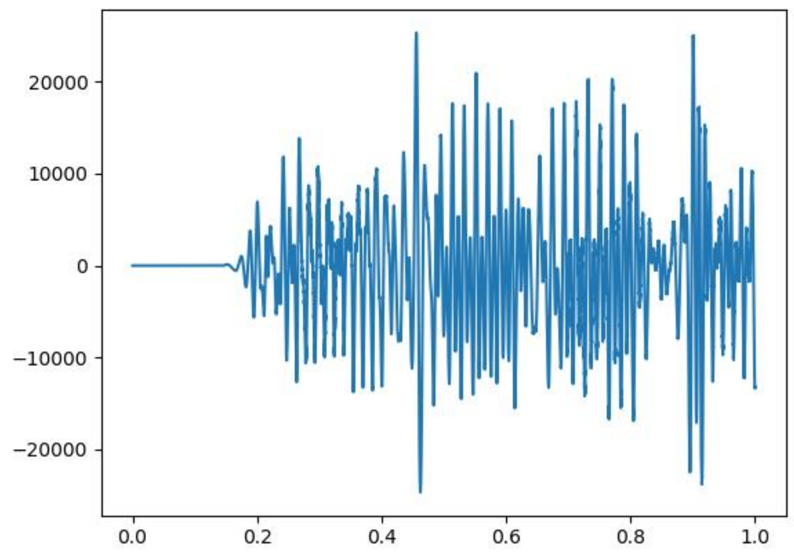
It can be seen that the amplitude of the signal decreased (frequency increased), and the time is shortened.

Part 3

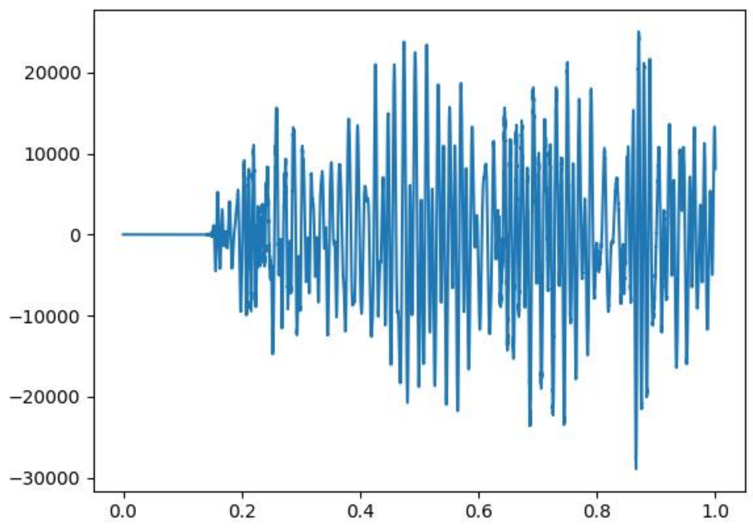
Audio signals are both convoluted using convolution function that I have written and also using fftconvolve function in scipy.signal. Results are similar when they are plotted, however functions do not return same results due to Fourier transform applied in built in function. Scipy signal applies Fourier transform to speed up calculation. Complexity of built in transformation is reduced to O(nlgn). My convolution implementation, using regular form of convolution formula produces result with complexity $O(n^2)$. After convolution, writeImpulseResponse function will be called. In this function, data may vary with high amplitudes, resulting in a distribution that is hard to standardize. Normalization using division by maximum amplitude is done to reduce noise, and normalize data. However, this will produce output between range [-1, 1] , which will not be in audible range. To convert it to audible range, I have extracted the maximum values for both channels from cropped_y2.wav file that was given in ninova as a sample, and multiplied my normalized results with this value to create a signal that can be heard. Channels are stacked together to produce output signal, output signal values will be mapped to 16-bit integer to benefit from Scipy wavfile. In order to run this code, numpy, matplotlib scipy.signal and scipy.io.wavfile packages are required.



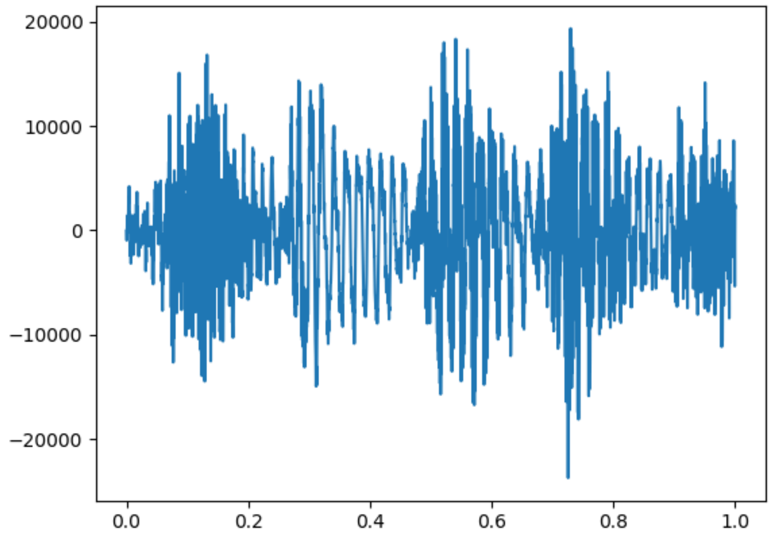
Cropped y2 image where I have extracted maximum values.



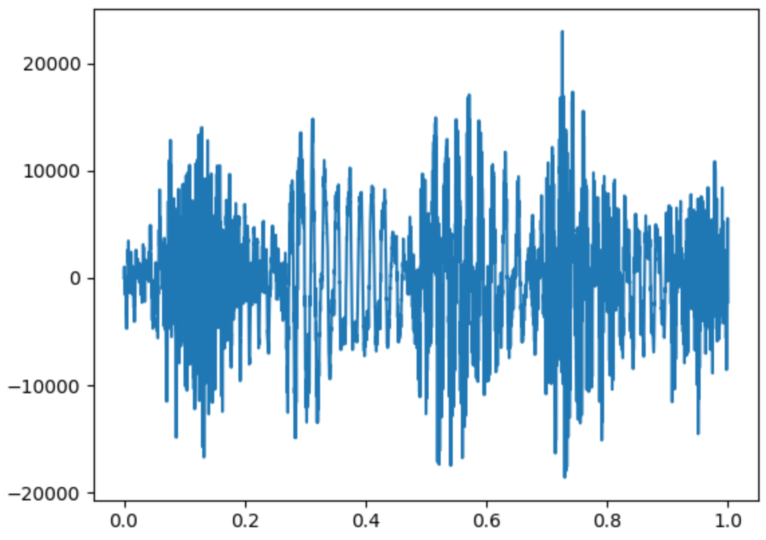
Y1 signal produced by convolution operation written by me



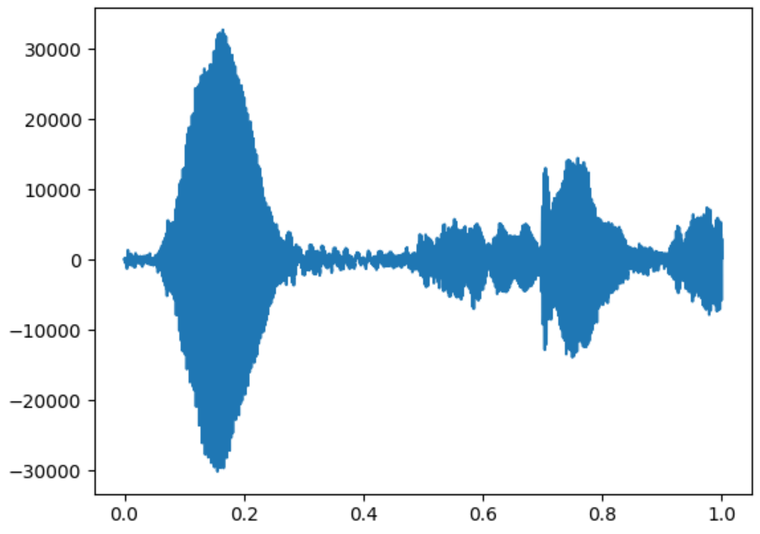
Y1 signal produced from built in library



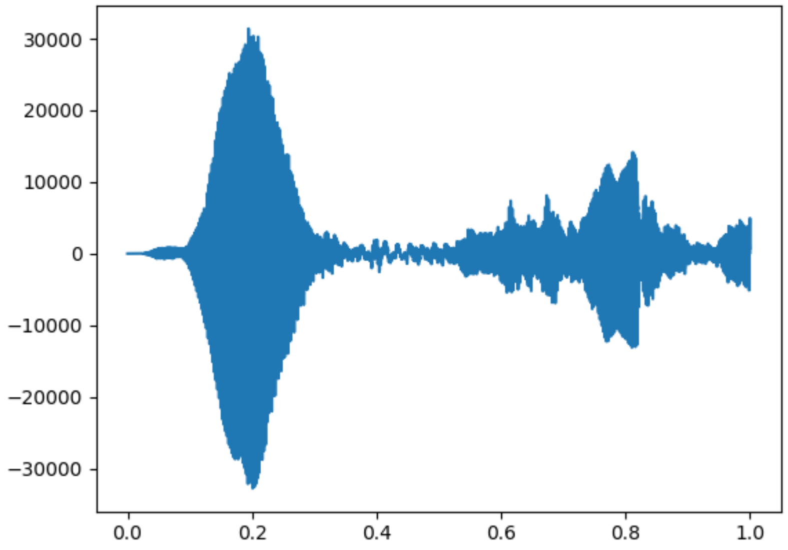
Y2 signal produced by convolution operation written by me



Y2 signal produced from built in library



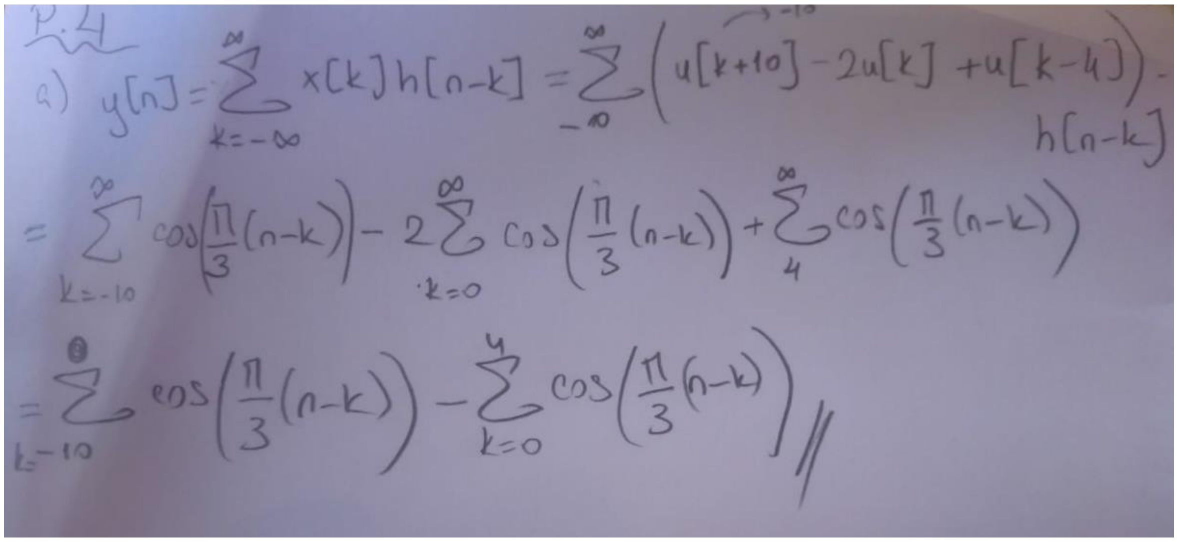
Y3 signal produced by convolution operation written by me



Y3 signal produced from built in library

Part 4

Convolution function previously implemented in part 3 is used to evaluate these functions.



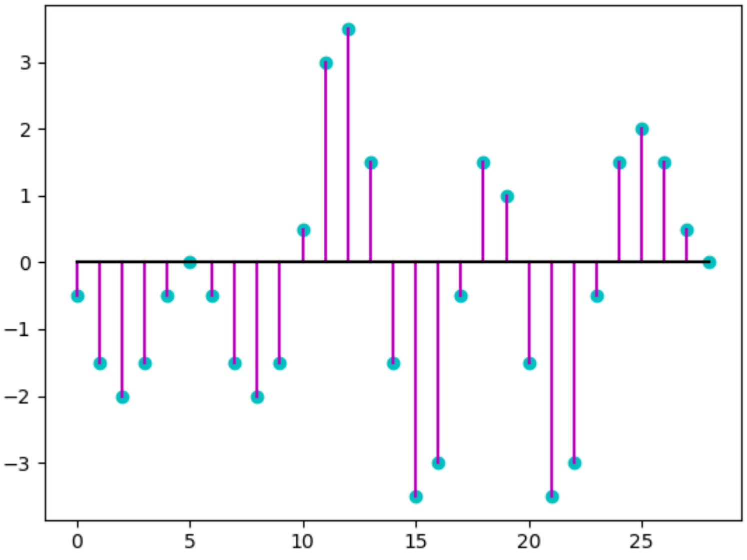
P.4

a) $$y[n]=\sum_{k=-\infty}^{\infty}x[k]h[n-k]=\sum_{-10}^{\infty}\left(\overset{\to -10}{u[k+10]}-2u[k]+u[k-4]\right)\cdot h[n-k]$$

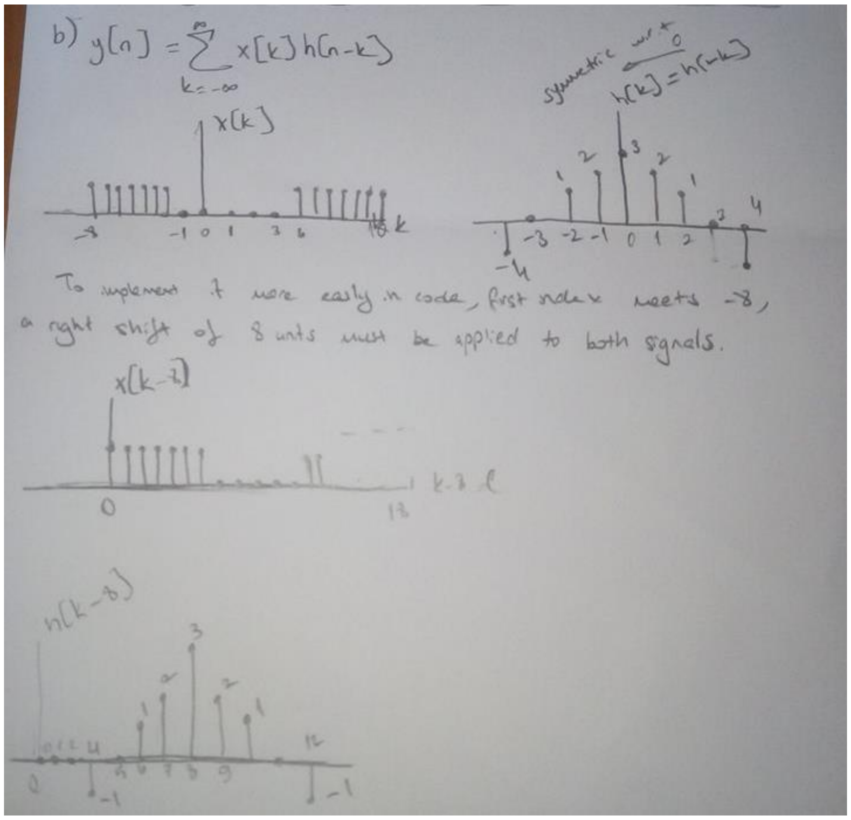
$$=\sum_{k=-10}^{\infty}\cos\left(\frac{\pi}{3}(n-k)\right)-2\sum_{k=0}^{\infty}\cos\left(\frac{\pi}{3}(n-k)\right)+\sum_{4}^{\infty}\cos\left(\frac{\pi}{3}(n-k)\right)$$

$$=\sum_{k=-10}^{0}\cos\left(\frac{\pi}{3}(n-k)\right)-\sum_{k=0}^{4}\cos\left(\frac{\pi}{3}(n-k)\right)\,//$$

For further calculation, I have implemented hw2p4.py, which will sequentially execute each part.



Line spectrum of y[n-10] can be seen in the plot above. However, indices are not accurate, I have implemented the plot using matplotlib.stem, which did not provide me with a tool to rename each axis.

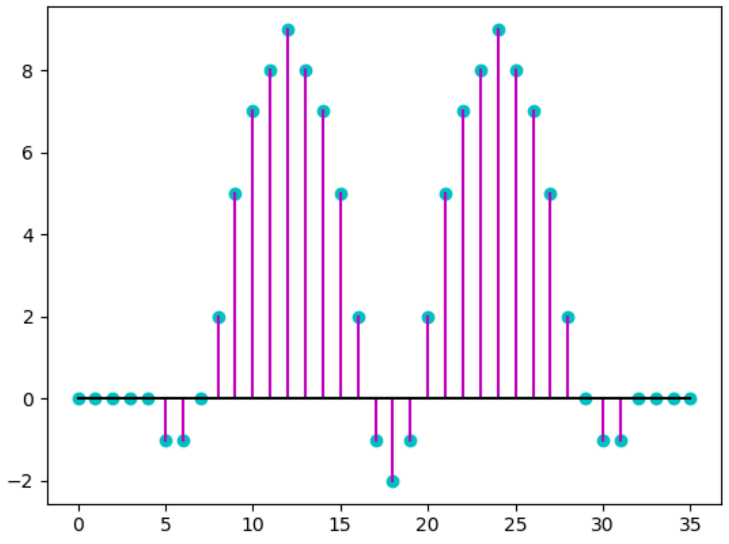


b) $y[n] = \sum_{k=-\infty}^{\infty} x[k]\,h[n-k]$

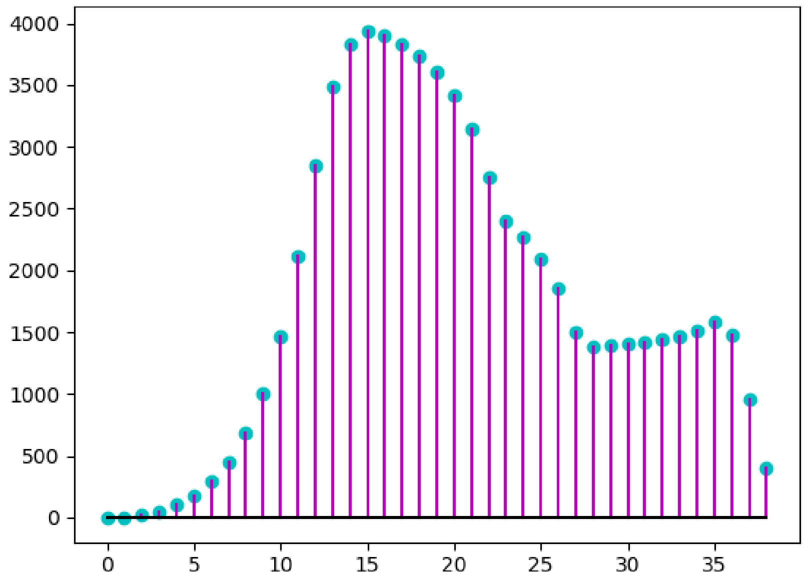
symmetric w.r.t. 0 $h[k] = h[-k]$

To implement it more easily in code, first index meets -8, a right shift of 8 units must be applied to both signals.

Indexes are shifted by 8 units to the right to implement to code more easily. X[k-8]h[n-(k-8)] will be used to produce output convolution. Same as part a, output graph will be shifted by 8 units to the right.



Third example of part 4 is a piecewise function created from many pieces, therefore I only implemented it in python code. I have chosen alpha as 1.42, used np.arange functions to create wanted indexes for x and h values. Np.arange does not include stop point, for stop points I used specified index+1. Different parts are calculated as different ranges, such as h_3_1st_range and then each range is concatenated to produce x_3 and h_3. Concatenated functions are then convoluted to produce f. A right shift of 1 is applied in order to make h[n] start from zero, not -1. Zero is appended to beginning of x.



4000
3500
3000
2500
2000
1500
1000
500
0
0
5
10
15
20
25
30
35