Digital Signal Processing and Audio Programming

PG Data Science Track (Pairs)

Word Count: 1,496

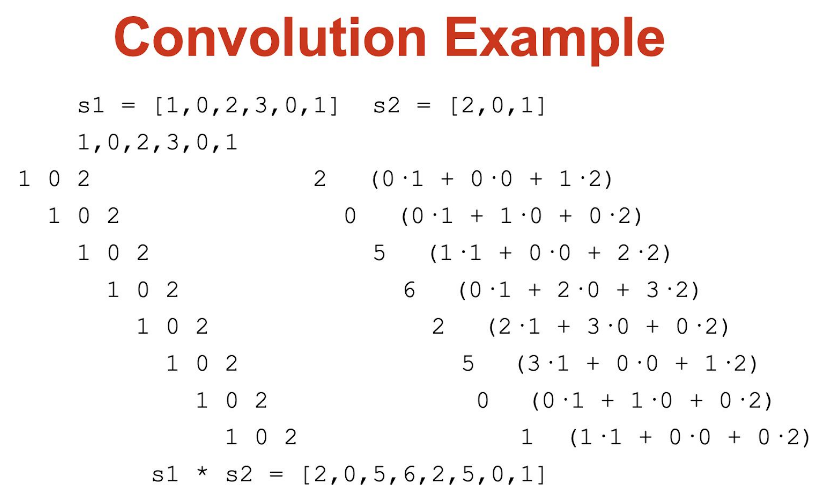
Pablo Quinoa
Yasmin Buzari

Task 1: Common DSP

We started by importing the RockA audio, a stereo signal (2 channels), which was transformed into a mono signal to simplify the processing required. Then, we created a sine control signal of 2 Hz by creating the signal with a number of samples slightly bigger than the original signal, so that the control signal has the same length as the convolution length (original rock signal + filter size - 1, using sample rate).

Because we cannot use the signal.convolve function, we created a dynamic FIR filter by implementing the convolution functionality between two signals ourselves. The reason we couldn't use the built in convolution function is because we need to dynamically change the filter coefficients over time for the RockA signal duration, by using our sine control signal.

The main logic to highlight in this task is how we achieved the convolution. We moved across the length of the convolution resulting signal (original rock signal + filter size - 1), and padded 0's at the start and end of the original signal to match the convolution resulting signal length. Next, we flipped the dynamic filter, and also moved the dynamic filter to the right one position in each iteration to simulate the delay as seen in the example below:



Convolution Example

```
    s1 = [1,0,2,3,0,1]  s2 = [2,0,1]
    1,0,2,3,0,1
1 0 2               2   (0·1 + 0·0 + 1·2)
  1 0 2               0   (0·1 + 1·0 + 0·2)
    1 0 2               5   (1·1 + 0·0 + 2·2)
      1 0 2               6   (0·1 + 2·0 + 3·2)
        1 0 2               2   (2·1 + 3·0 + 0·2)
          1 0 2               5   (3·1 + 0·0 + 1·2)
            1 0 2               0   (0·1 + 1·0 + 0·2)
              1 0 2               1   (1·1 + 0·0 + 0·2)
     s1 * s2 = [2,0,5,6,2,5,0,1]
```

Figure 1: Convolution, DSP-Lecture 4 Slide 27

Task 2B: Digit Recognition

We started by creating a function comparing each test image with the collection of training images, by using the correlation with each training image without offset. Given that each training image has a label, we can use this to classify any test image. We calculated the correlation between each pair of images (matrices) using the correlation coefficient p formula below, which measures the similarity of signals x and y:

$$\rho = \frac{dot(x, y)}{\sqrt{\sum x[n]^2 \cdot \sum y[n]^2}}$$

Figure 2: Lecture 3, Slide 9

We then calculated the accuracy of the above digit recognition system by classifying the test set and comparing its predicted labels against real labels. We achieved an accuracy of 97.66%, which clearly proves the efficiency of using the correlation without offset against training images, to recognise digits given some images.

Next, we calculated the cross correlation between all test and training images to find the best match. We used the builtin function signal.correlate2d, and a subset of 400 images over both the test and training set to decrease the processing time required for this section.

Finally, we implemented variants of the digit recognition system to compare how well it performs. Our first variant rotated the training images by 25 degrees, and then applied our previous calculateCorrelation function (section 1) to see how the accuracy changes when trying to compare the test images against all training images now rotated 25 degrees. This resulted in an accuracy of 53.5%.

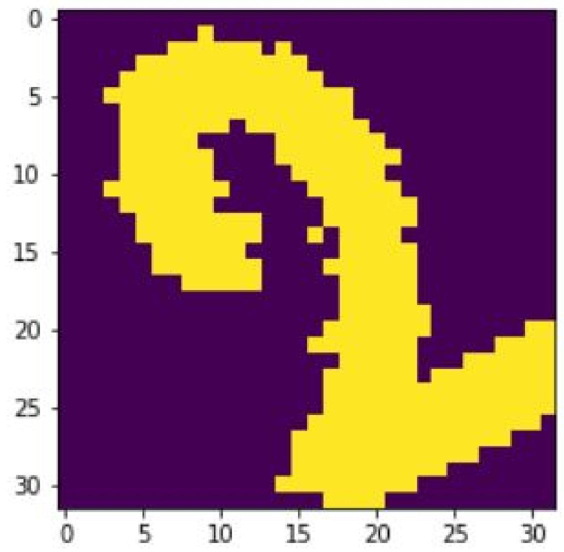


Figure 3: Training image 25 degree rotation

Our second variant included salt and pepper noise to the training images. We used the same calculateCorrelation function, and the accuracy on predicting the digits of the test set images against training with salt and pepper noise was of 96%.

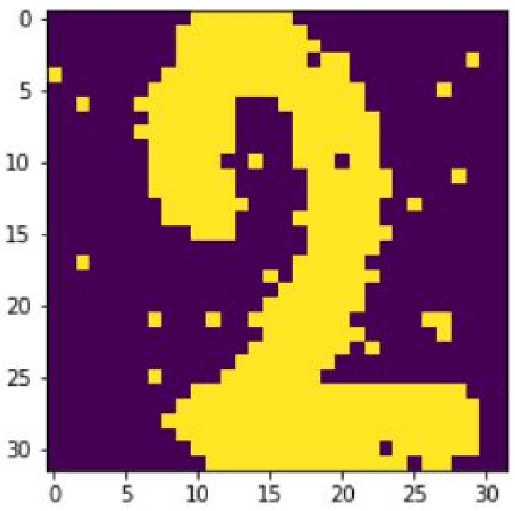
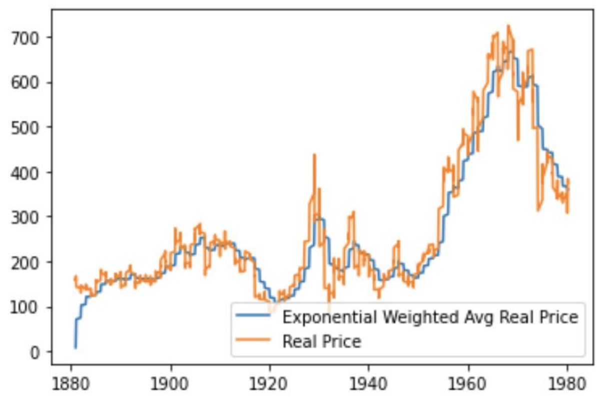


Figure 3: Training image with salt and pepper noise

We concluded that the first digit recognition system using images without noise or rotation (section 1) gives very accurate results (almost 100%). After adding some noise to the training set, the accuracy was still close to 100%, thus some small random noise does not really affect the system. Conversely, if the training images were rotated by 25 degrees the accuracy decreased by almost 50%, which is very considerable. This is because we are using the correlation of a rotated image against a straight one, meaning that too many pixels have moved or are different, returning a significant decreased correlation.

Task 2B: Time Series Prediction

First, we applied a moving average to the data, allowing us to compare future trends based on historical observations and by smoothing the data using an exponential weighted method , we can infer the main trend of the data over time. We did this by using the signal.lfilter function with coefficients w and (1-w). When applied to our 3 major data groups Real Price, Real Earnings, and Real Dividends, we can see the volatile nature (rise and fall) through by comparing the peaks to the smoothed average line. For a holistic view, we applied both a linear and log perspective. However, it is important to consider that analysis based on linear scales tend to overemphasize recent movements in data, and so we de-trended the data across the three main groups using the linear scale before prediction analysis in Figure 4. The difference in the log scale is that logarithmic charts will give more equal weights to the data, highlighting upward trends.



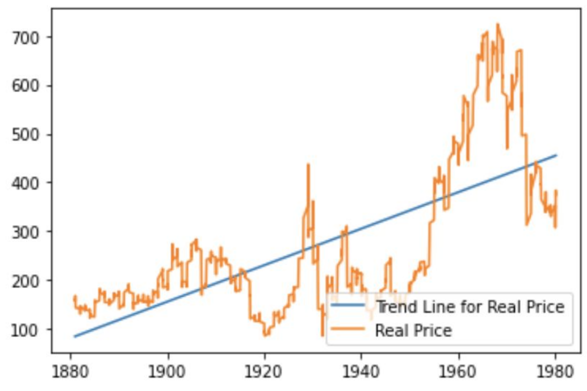
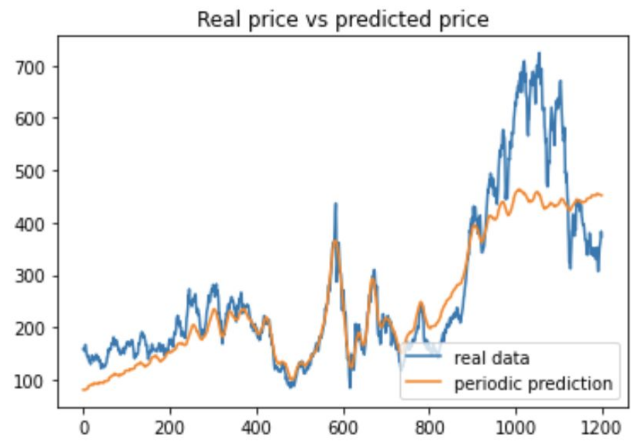


Figure 4: Smoothed Line and Linear Trend for Real Price

Next, we applied the FFT with a hanning window to create a harmonic model for each price group. It is important to apply the linear trend each time when analysing time series data, because there are many possible factors that can affect overall patterns and long term trends. Thus, we readjust each segment of analysis by including the trend line for each window of time we are performing predictive analysis. Applying a hanning window also decreases any spectral leakage.

We used the harmonic model of each group to create a prediction spectrum with specific thresholds applied to eliminate any noise in each model. We refined our analysis further by applying predictions to a test set of the data after observing results of the train set seen in Figure 5. When applied on the training with an excerpt of the entire training set, we expected a very close prediction with the real data because we are using the same data to build the predictions and to test it (figure 5 left graph). We observe the results in the right graph from Figure 5 when applied to the test set and for each, we added the linear trend back.



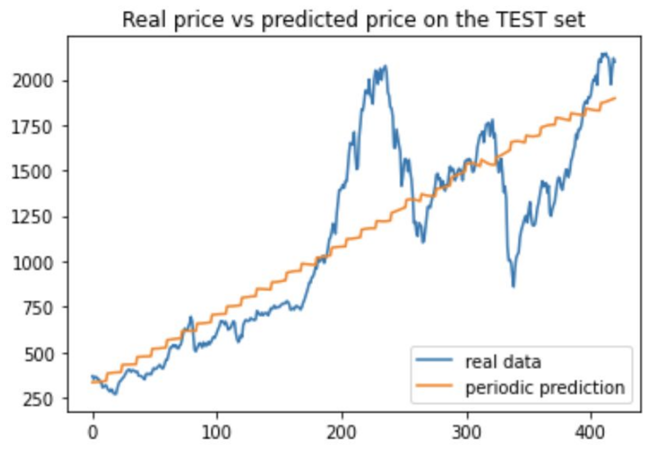


Figure 5: Real Price Predictions

Task 2D: Frequency Domain Filtering

In this case we implemented the FIR filter from task 1, but in the frequency domain (FD) instead of the time domain. Because we know that convolution in the time domain equals multiplication in the FD, we used the same 2 arrays of FIR coefficients from task 1 for interpolation, given the sine control signal. The generation of the dynamic filter was done similar to task 1, but this time we transformed the resulting dynamic filter using FFT because we are now in the frequency domain.

Next, we computed the Short Time Fourier Transform (STFT) of the RockA signal so that we would have an array of smaller time windows, each one with its corresponding frequencies. We then iterated over all these time windows (or segments), by multiplying the segment's frequency bin with the dynamic filter´s frequencies. We applied the STFT on the rock signal using a desired length for each segment of 21 (same size as the dynamic filter), as we needed to have equally sized arrays to multiply them. We finally multiplied the segment's frequencies with the dynamic filter (filter that was recalculated for each time segment) for each segment of the STFT.

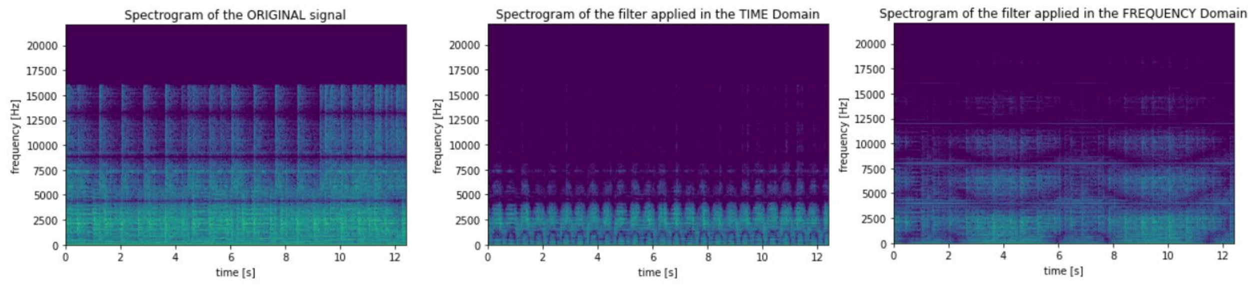


Figure 6: Spectrograms of the original signal (left), signal after filter applied in time domain (centre), and signal after filter applied in frequency domain (right).

We can see how the spectrogram in the centre of the signal which was filtered in the time domain (task 1) shows 4 'bumps' between each line of the time axis (2 seconds in between lines) making 2 cycles per second, so 2 Hz. We can clearly see the periodicity of the control sine in the resulting signal after applying the filter.

In the spectrogram on the right, from the filter applied in the frequency domain, we cannot really appreciate such periodicity. We can see in the spectrogram and by listening to the signal audio, that the signal differs from the resulting signal in task 1, suggesting that something was done wrong during the implementation of our FIR filter in the FD. We know that these two should sound and look very similar when plotted in a spectrogram because of the convolution theorem.

Processing in the frequency and time domain are two sides of the same coin, processing the same signal using inverse dimensions. So there are no advantages or disadvantages of using one or the other, it really depends on what type of processing we want to perform on a given signal, that will tell us which domain works better. If we talk about the specific case studied in this coursework of applying an FIR filter, it is probably computationally more efficient to operate in the frequency domain than in the time domain because its faster. It is less costly (computationally) to multiply the signal frequencies element by element rather than having to convolve a signal with the filter (delay, and flip the filter, then compute the dot product).