

EXECUTIVE SUMMARY

This project performs an exploratory data analysis on Franz Schubert's Impromptu in A-flat major, Op. 90, Number 4. The piece is considered in the score representation, in which the written music is directly translated into data. We find that the piece can be decomposed into three subsections based on the frequencies of the notes being played as well as the number of notes played per measure. There are strong relationships between the six voices being played, regardless of whether missing values in the voices are filled in with values randomly selected from among existing notes, or whether they are treated as rests. Missing values occur when less than 6 notes are played at one time. Spectral analysis quantitatively shows the periodicities that are evident from looking at the score or listening to the piece. An ARIMA(p, d, q) model is fit with mixed success.

1. INTRODUCTION

Since ancient times, music has played an important part in the lives of humans of all cultures. New technologies have made it possible in recent years for researchers to try to quantify what characterizes "music" (as opposed to "noise"), and to try to draw quantitative conclusions about the structure of music or the relationship between pieces. The idea of analyzing music in the context of a time series analysis, especially spectral analysis, is quite a natural one. After all, music is nothing but a series of time-dependent sound waves (cosinusoids). The waves, in turn, are simply combinations of the sound waves of the individual series. Spectral analyses attempt to do the inverse of this – decompose a series into underlying cosines of different frequencies.

This project explores Schubert's Impromptu in A-flat major, Opus 90 Number 4. To listen to the piece, please visit

<http://www.ieor.berkeley.edu/~roeder>

This report is organized as follows. Section 2 describes the data, and the methods used to convert it into the proper format. Section 3 details the analysis performed on the data. Conclusions are given in Section 4. Ideas for future work are presented in Section 5, and acknowledgements and references follow in sections 6 and 7. The code used to process and analyze the data is given in the Appendix, and can also be found at the above website.



Figure 1 The first line of the score for Schubert's Impromptu Op. 90 No. 4

2. THE DATA

The piece analyzed for this project is Franz Schubert's Impromptu in A-flat major, Opus 90 Number 4. I chose this piece for several reasons. The first reason is that it was the last piece I played for my piano lessons, and I enjoyed it very much. The second reason is there are no

ornamentals (like trills or grace notes), and the notes are very regular. I thought this would simplify the analysis. Figure 1 shows the first line of the score.

2.1 Background

Brillinger and Irizarry (1998) describe the two approaches that can be taken to analyzing music: using the signal representation, or the score representation. (The following summary refers to and uses notation from this paper.)

Signal representation is the approach that comes to mind naturally when thinking about analyzing music. Music is nothing more than sound waves, i.e. waves of differing air pressures. Since much of the analysis we have covered in class is harmonic, it is a very natural way to think about analyzing the music.

However, there is a second approach, based on the score. Here, the music's score is transcribed in a way that captures the pitches, rhythms, and volumes. A very common way of doing this is known as the MIDI (Musical Instrument Digital Interface) standard. Here, notes are represented by their pitch (MIDI note number), their volume (MIDI velocity), and their on-off time (time the note is held). The MIDI representation can be used to analyze the score.

In general, pitches can be denoted by their MIDI numbers, or even by their frequencies. For example, concert A has a MIDI number of 69 and a frequency of 440Hz. Adjacent notes (e.g. A# and Ab)¹ add or subtract one from the MIDI number. MIDI numbers can be converted to frequencies using the following equation. X is the MIDI number of the note of interest, and f is its associated frequency:

$$f = 440 \cdot 2^{\frac{X-69}{12}} = 8.175799 \cdot 2^{\frac{X}{12}} \text{ Hz}$$

Note durations are typically denoted using the concept of a *tatum*. A tatum is defined as the smallest subdivision of a beat in the score. The value of a tatum for this project is defined in the next section.

2.2 Initial Data Preparation

This project analyzes the piece using the score representation. I chose this representation because the idea of exploring music in this manner was novel to me. In addition, transcribing the music directly from the score should minimize the noise in the series. A disadvantage of using the score representation is that certain elements of the music are lost – this piece makes heavy use of the “right pedal” on the piano, which causes notes to “run into” each other; the notes are held until the pedal is depressed again, which is often much longer than the duration suggested by the score itself. This is not captured, since it is a tool of expression added by the performer. Similarly, accelerandos, decelerandos, and other stylistic devices do not appear in the score representation as they are not visible in the actual notes themselves, rather are marked separately in the score (if at all). This can be an advantage as the analysis focuses purely on pitches and

¹ A ‘#’ after a note means “sharp,” i.e. one half-note above the note. So A# is “A-sharp.” A ‘b’ after a note means “flat,” i.e. one half-note below the note. So Ab is “A-flat.” Note that $\text{pitch}(G\#) = \text{pitch}(Ab)$.

rhythms. On the other hand, it is a disadvantage since music does *not* consist solely of notes and rhythms.

The specific data used for this project were obtained in a rather roundabout fashion. A MIDI version of the file was processed by *Finale*, a music notation software package. *Finale* condensed the original four MIDI channels to a single channel. The new MIDI file was then processed by *Seek*, a program written by David Wessel at the UC Berkeley Center for New Music and Audio Technologies (CNMAT). This program converts the MIDI file into plain text data. Below are the first few lines from the resulting data file with column headings. (They correspond to the first set of 4 sixteenth notes in the treble clef, and the single quarter note in the bass clef in Figure 1.)

| Index | MIDI Channel | MIDI Note | Velocity (Volume) | On-Off Time (in msec) | On-On Time (in msec) |
|-------|--------------|-----------|-------------------|-----------------------|----------------------|
| 1 | 1 | 83 | 72 | 145 | 0 |
| 2 | 1 | 44 | 57 | 468 | 156 |
| 3 | 1 | 87 | 72 | 148 | 159 |
| 4 | 1 | 83 | 71 | 148 | 156 |
| 5 | 1 | 80 | 68 | 148 | 154 |

“Index” refers to the row’s index in the data file. The “MIDI Channel” is 1 for the entire data set (since the data had previously be condensed to one channel using *Finale*). The “MIDI Note” refers to the MIDI values described above. The “Velocity” is MIDI terminology for the volume at which the note is played. “On-Off Time” denotes the number of milliseconds the note is held, and “On-On Time” indicates the number of milliseconds that elapse before the next note is played.

In this example, notes 1 and 2 are played simultaneously, and are followed by notes 3,4, and 5. Note 2 is held for the duration of notes 1, 3, and 4. From the score, all notes but note 2 are sixteenth notes. This shows a “problem” I had not anticipated with MIDI files – they *do* contain noise, since the music has to be entered (i.e. played) by someone to generate the file. Technically, the lengths of notes 1, 3, 4, and 5 should be identical. In the data, however, there are minor fluctuations (145 versus 148 msec). Similarly, the times between notes (“on-on time”) should be identical for the sixteenth notes, if not identical to the duration of the sixteenth notes. (In the score, the two times should be identical. Practically, this is not possible since a note must be depressed before the same finger can press the next note on the piano.)

Since the goal of this project was to analyze the score, the timing values were converted to tatums to try to minimize errors induced by minute fluctuations in note lengths. Originally, one tatum was set to 150 msec (the equivalent of a sixteenth note). Then it became evident that this was not a good approach, since there are two sections of the piece where Schubert inserts a triplet-theme (around measures 72 and 344). A set of three triplets is of the same length as four sixteenth notes. It is possible to just “gloss over” the differences between the sixteenth notes and triplets – the triplets would all be of length one tatum (which is too short). Unfortunately, this would cause a disconnect, since there would be only three tatums while the accompaniment (playing quarter or half notes) would be held for four tatums, though they should be of the same total length. As a consequence, one tatum is set to 50 msec, greatest common denominator between the sixteenth notes (length 150 msec) and the triplets (length 200 msec). No note will be held for only one tatum; the sixteenth notes (the original “tatum”) will be held for 3, while the triplets will be held for 4.

For the code used to convert the values, please see the Appendix. This code also converts the MIDI numbers to frequencies using the equation above. The resulting data are below:

| Index | MIDI Channel | Frequency (in Hz) | Velocity (Volume) | On-Off Time (in tatums) | On-On Time (in tatums) |
|-------|--------------|-------------------|-------------------|-------------------------|------------------------|
| 1 | 1 | 987.767 | 72 | 3 | 0 |
| 2 | 1 | 103.826 | 57 | 9 | 3 |
| 3 | 1 | 1244.51 | 72 | 3 | 3 |
| 4 | 1 | 987.767 | 71 | 3 | 3 |
| 5 | 1 | 830.609 | 68 | 3 | 3 |

This translation has reduced the millisecond-fluctuations of note durations and between-note times, though it has not made the data completely compatible with the score – note 2 should be held 12 tatums, not 9.

2.3 Final Version

Another problem that appeared when attempting to analyze the data was that there are a varying number of notes being played at once. At any point, there are between zero and six notes. As a result, there is not one single time series, rather, there are six time series with many missing values. (The time average number of notes played at a time is 3.7.) There are two ways of dealing with this problem.

The first is to convert the note(number)s being played to frequencies. The sound wave for note i is $\alpha_i \cos(\omega_i t + \phi_i)$. The sound wave the ear actually hears is the sum of the sound waves for the notes that are being played simultaneously. Unfortunately, no single “number” can characterize these oscillations. The solution is to increase the total number of data points in the study by sampling from the wave at very small time intervals. (The typical frequency used for sampling is 44kHz, based on the Nyquist Sampling Theorem.) If the original piece were of length 600 seconds and the sampling rate was 44kHz (44,000 samples/second), the new number of data points would be 26,400,000.

The second approach one can take is treating the data as six time series with missing values. If, at time t , there are only four notes being played, two of the “series” would have missing values. We can randomly select one of the four notes being played to fill in the gaps (sampling with replacement). One can think of this approach as playing the same piano key with multiple fingers; it will sound the same, but it solves the missing value problem for the time series. An alternative approach is to fill in the remaining voices with zero frequency, corresponding to a rest. This is what actually occurs when playing the piece on the piano – the fingers with no notes to play do not play anything.

When looking at the data, it is unclear which notes belong to which voice. To be systematic, the notes being played were assigned, in decreasing order of pitch, to voice 1, then voice 2, etc. The remaining notes are filled with the randomly selected ones. In the example above, at times 1 and 2, the voices are:²

² For illustrative purposes, “time 2” here refers to the time of the second sixteenth note, not to the time of the actual second tatum. Here, the voices are 988, 104, 104, 988, 988, 104. The tatum number for the second sixteenth note is 4.

| Time | Voice 1 | Voice 2 | Voice 3 | Voice 4 | Voice 5 | Voice 6 |
|------|---------|---------|---------|---------|---------|---------|
| 1 | 988 | 104 | 988 | 104 | 988 | 104 |
| 2 | 1245 | 104 | 104 | 104 | 1245 | 1245 |

Each column represents one of the six time series. The notes are expressed in Hz. The approach of assigning random notes to missing values is used since the existing notes are semi-arbitrarily assigned to the voices, and it may be that the lowest note is *actually* being played by voice 5, not voice 3 (when three notes are being played at once). If the missing values were filled in with rests, all the values for voices 3 through 6 would be zero.

2.4 Derived Time Series: Number of Notes Played Per Measure

When trying to decide how to analyze the primary data set, another way of looking at the piece became evident: analyzing the number of notes played in each measure. More specifically, analyzing the number of notes that are started in each measure. Some notes are started in one measure and held into the next (and perhaps even the following) measure. Here, we consider the notes only when they are started to see whether there is a way to tell by the number of notes being played what section of the piece is being played. (Transitions tend to have either many chords that are held, or have chords that are played as arpeggios³ with little accompaniment.)

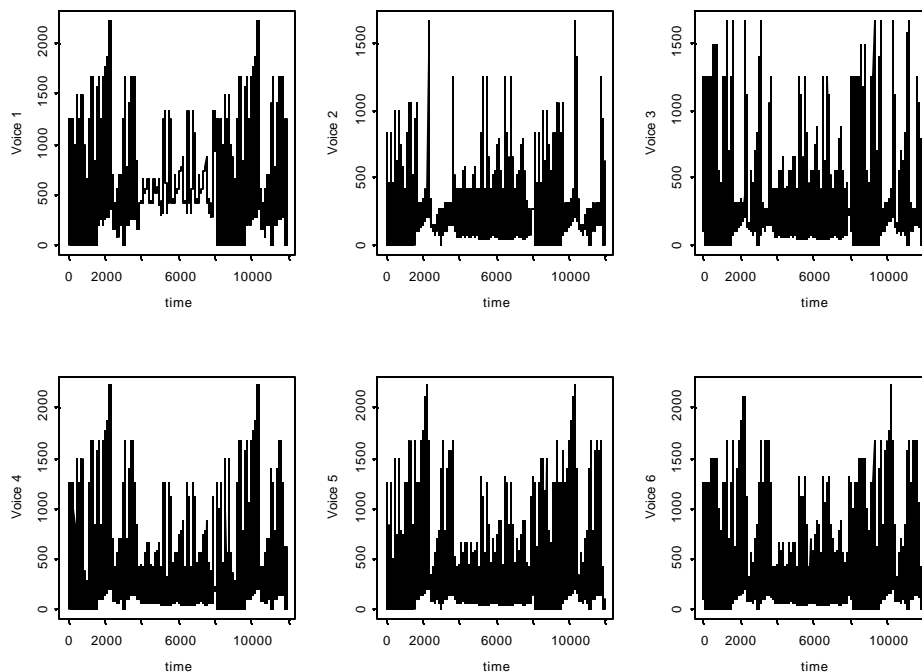


Figure 2 The six voices when missing values are filled in randomly

³ An arpeggio is a chord (for example Ab major, in this case) for which the notes are played successively, not all at once.

3. ANALYSIS

3.1 Some Descriptive Statistics

3.1.1 Voice Data Series

Figures 2 and 3 show graphs of the six voices. Figure 2 shows the data when the “missing values” in the voices are filled in with randomly selected existing values, while Figure 3 shows them when the voices have “rests” when there are no explicit notes assigned to them. In both cases, Voice 1 clearly can be separated into 3 separate sections: In the first and third sections, the frequencies of the notes played are highly variable. These are the sections with the characteristic arpeggio themes, so the differences in the frequencies are quite pronounced. In addition, the step-like nature of the successive notes can be seen the close-up of the first 4000 tatum in Voice 1. (See Figure 4.) The second section, which shows a much more constant frequency level, corresponds to the middle section of the piece, in which there is relatively little variability in the pitch of the notes played.

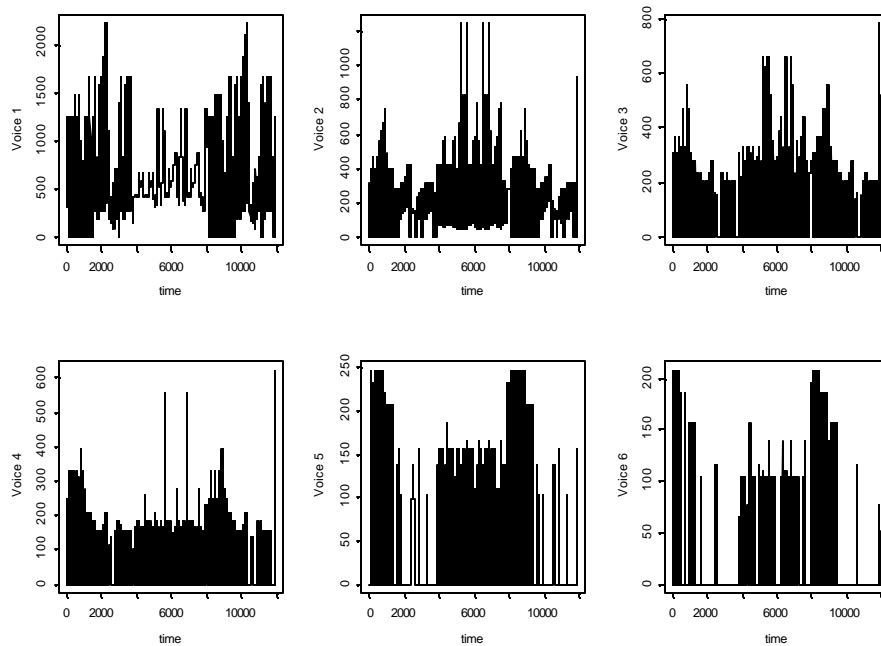


Figure 3 The six voices when missing values are assigned rests

Depending on how missing values were handled, the other five voices show progressively more randomness (when the values were filled in), or progressively sparser data points (when the voices are assigned rests). When there are data in the latter case, the frequencies get lower (200 Hz or less for Voice 6) the higher the voice number. Since all six notes are only played when there is a full chord, this is not surprising, as the lowest note (Voice 6's note) will be fairly low.

The most common note played in Voice 1 is the A_b above middle C (i.e. half a note below concert A) – 1655 of the 11906 data points are this note. The second most common is the A_b one octave above the previous one (932 of 11906). Since the piece is in A_b -major, the result

makes sense. In Voice 6 (with rests), the most common note *played* is the *Ab* two octaves below middle C (1661 of 11906). By far the most time is spent in rests (9528 of 11906 tatum). Another way of looking at this is that six notes are being played only 20% of the time.

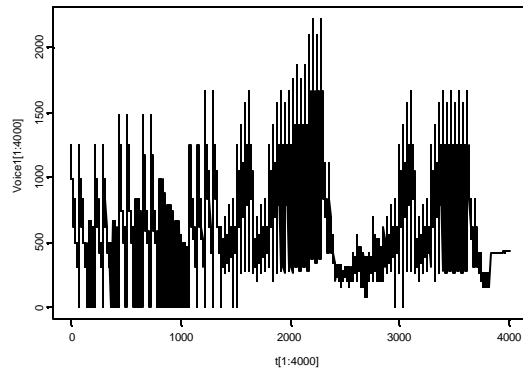


Figure 4 Closeup of the first 4000 values of Voice 1

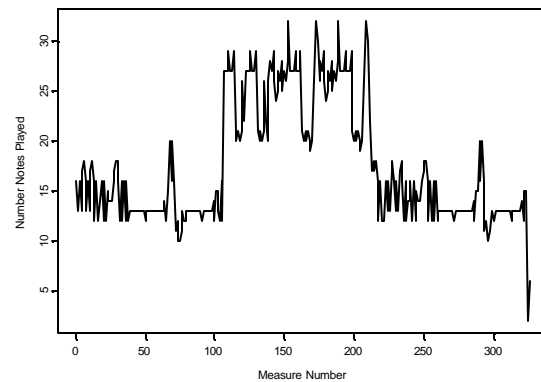


Figure 5 The number of notes played per measure

The three sections of the piece are roughly from tatum 1 to tatum 3700, from 3700 to 7800, and from 7800 to the end. The exact cutoff points could be determined by looking at the score, but we are trying to find all the values from the data, not the score. (The changeover points are not clearly discernible as there are a few measures of transitional music between the different sections, and they cannot be classified as being in either section.)

3.1.2 Number Notes Series

Figure 5 shows the number of notes played in each measure of the piece. (There are a total of 327 measures.) The average number of notes played per measure is 17.5.⁴ This series again delineates the three sections of the piece. For the first section, the average number of notes played is 14.9. The second section (which is, interestingly enough, exactly 100 measures long, including repeated parts, excluding transitions) has an average of 24.9 notes played. The third section averages 13.5 notes played. (The third section is nearly identical to the first, the major difference being the final three measures which each contain only one chord – for totals of 2, 6, and 6 notes respectively.)

The second section is easily identifiable in both the primary and secondary series. In the primary series, the notes played are of more constant frequency than in the other two parts (at least for Voice 1). Similarly, there are 10 more notes, on average, being played per measure. The two series together can illustrate the character of the second section nicely – though more notes are being played, they are held twice as long as the typical note in the first and third sections, and are played as chords, not as arpeggios. Similarly, the chords stay in the same range; in the other two sections, a chord (e.g. *Ab*) may be played as an arpeggio across several octaves. What the two series cannot identify as readily are differences in key. Unfortunately, a *G#* has the same frequency as an *Ab*. The second section is in a minor key for the majority of the time. The only way we may be able to identify this is to consider the six voices all at once.

⁴ To be completely accurate, this should read “the average number of notes that are started being played per measure....” The series counts the notes started, not the actual notes played (which includes those started in previous measures and held through to the current one).

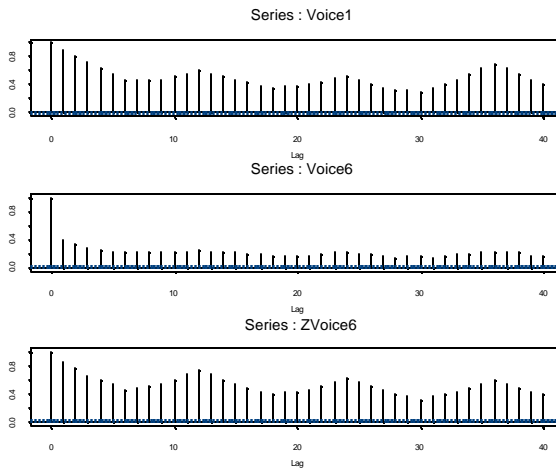


Figure 6 ACFs for Voice 1; Voice 6 with random missing values; Voice 6 with zeroes

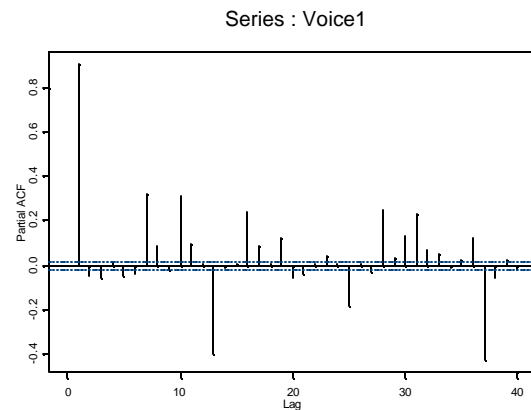


Figure 7 Partial ACF for Voice 1

3.2 Correlation Analysis

3.2.1 Voice Data Series

The correlation plots for the voice data are interesting, though not unexpected. Voice 1 shows nice periodicities in the correlations. As Figure 6 shows, there is a relative peak in the autocorrelations every 12 time periods. 12 time units correspond to one quarter note (or four sixteenth notes). Indeed, the lag 36 correlation is larger than the lag 12 autocorrelation (0.6978 versus 0.5646) – notes that are an entire measure apart have a greater linear relationship than those that are only a quarter note apart. As noted in Venables and Ripley (1999), if a series is $AR(p)$, all correlations will be non-zero. To try to discern meaningful relationships, they suggest using a partial correlation analysis instead. The following discussion will use both the ACF and the PACF. Figure 7 shows the PACF for Voice 1. As noted above, there are sharp peaks at lags 12 and 36, though the partial autocorrelation at 36 is greater than that at 12.

When the missing notes are randomly filled in, the levels of correlation and the periodicities decrease with the voice number: The voices become more randomized, and so there is less structure in them. This is markedly different from the same voices when missing values are assigned rests. Here, the relationships and periodicities become greater. Since there are more and more rests as the voice number increases, this is also not surprising. It is easier to predict another rest if the vast majority of the notes are rests. The second two graphs in Figure 6 illustrate the differences in the correlation plots for Voice 6.

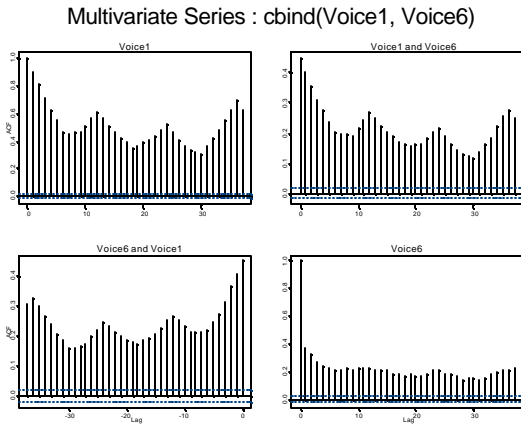


Figure 8 Cross-correlations for Voice 1 and Voice 6 (random values)

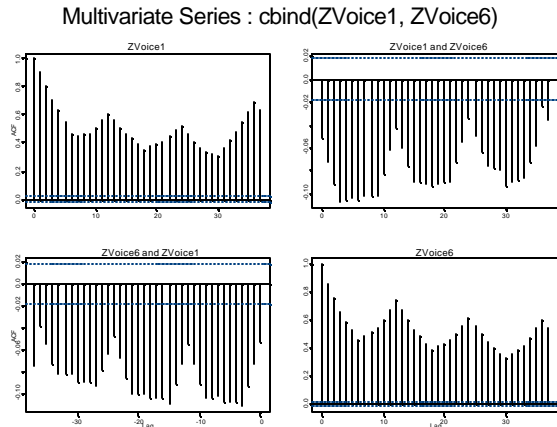


Figure 9 Cross-correlations for Voice 1 and Voice 6 (zeroes)

Looking at the cross-correlations between the different voices did turn up interesting results. When the missing values are filled in, the cross-correlations for the Voice 1 with the other voices are as expected – there are positive relationships between the series, and the relationships become more pronounced as the voice number increases. (With increasing voice number i , it is more likely that the Voice i will be playing the same note as Voice 1, for example.) The cross-correlations between Voices i and j show significant correlations, but fewer periodicities in the actual correlation values. Even with the higher-numbered voices (whose autocorrelations are relatively low and exhibit virtually no periodicities), the relationships are strong. Again, this can be explained by the fact that i and j will be strongly related since they are sampled from the same (very small) set of notes. They also exhibit periodicities. Figure 8 illustrates this phenomenon for Voice 1 and Voice 6.

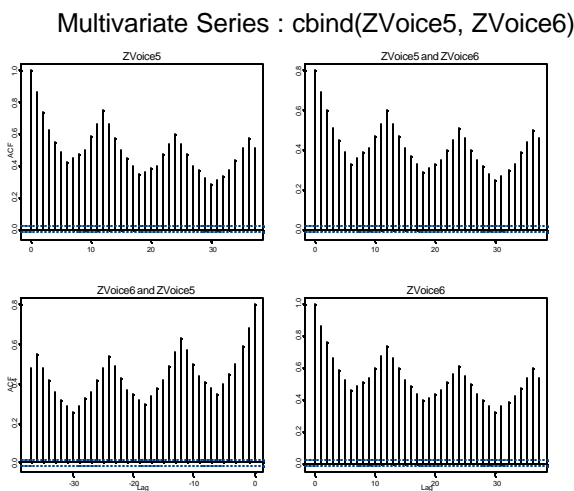


Figure 10 Cross-correlation for Voice 5 and Voice 6 (zeroes)

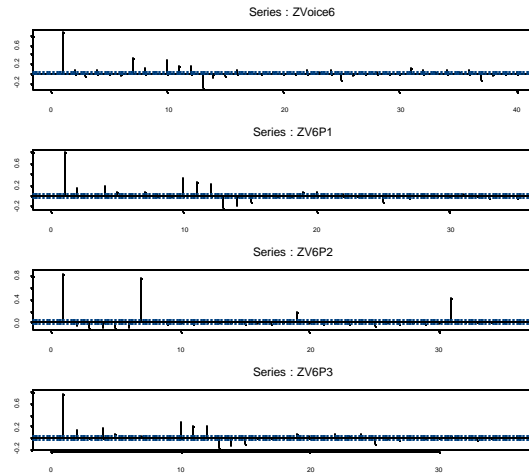


Figure 11 PACFs for Voice 6: Whole series, Sections 1-3

The more interesting results occur when the missing values are filled in with rests. Here, the cross-correlations between Voice 1 and Voices 4-6 have become markedly negative! Figure 9

shows the ACF for Voices 1 and 6. The most negative value is -0.1117 , which occurs at a lag of -3 for Voice 6 versus Voice 1. Another interesting note is the strong periodicity in the correlations. A possible explanation for these findings is that Voice 1 almost always contains a note, whereas Voices 4 through 6 are more likely to contain rests. (Voice 4 rests 5432 of 11906 time units. Voice 5's rests have increased to 8139 time units.) As a consequence, it is likely that Voices 4-6 are resting while Voice 1 is playing a note, resulting in a negative relationship between the voices. The periodicities may be due to the fact that Voices 4-6 are played when there are chords. Chords are typically held longer than arpeggios (common in Voice 1), and often occur at the beginning of a measure, or every quarter note. There are, therefore, very definite points when chords are played, and when the voice is resting.

The negative correlations just described do not appear for any other pairs of voices. Pairwise comparisons of Voices 2 through 6 show more and more positive correlations as the voice numbers increase, and the periodicities become sharper and sharper. Figure 10 shows the cross-correlations between Voice 5 and Voice 6. The average correlation is around 0.4.

Multivariate Series : cbind(ZVoice1, ZVoice3)

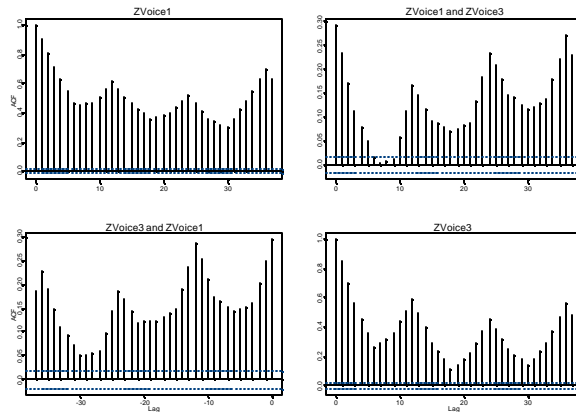


Figure 12 ACF for Voice 1 and Voice 3 (zeroes), whole piece

Multivariate Series : cbind(ZV1P1, ZV3P1)

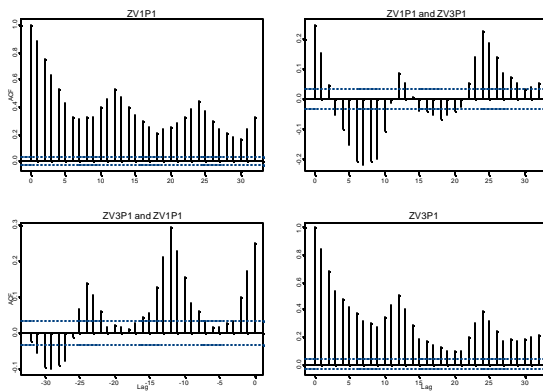


Figure 13 ACF for Voice 1, Voice 3 (zeroes), Section 1

Multivariate Series : cbind(ZV1P2, ZV3P2)

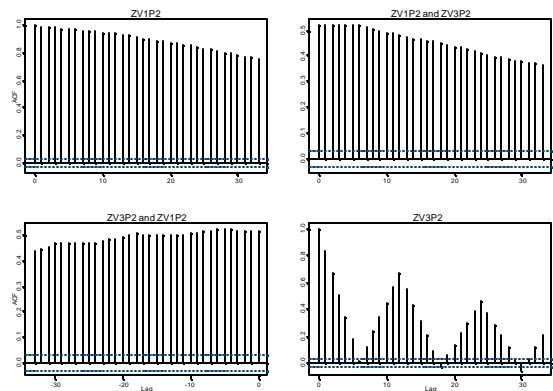


Figure 14 ACF for Voice 1, Voice 3 (zeroes), Section 2

Since the piece is clearly divided into three distinct sections, additional correlation analyses were conducted on the individual sections for each of the voices. (The sections are

defined as in Section 3.1.1 – tatum 1 to 3700, 3701 to 7800, and 7801 to 11906.) There are now 36 series (6 voices \times 2 methods of dealing with missing values \times (1 complete series + 1 Section 1 series + 1 Section 2 series⁵)). The general tendency of the results are the same as they were for the analysis of the voices as a whole, though there are a few differences.

- For Voice 1, the large peak for lag 36 no longer appears for any of the 3 sections. The PACFs for all sections show a strong negative peaks at lag 13, and a slightly smaller one at lag 25. There are also several positive peaks for sections 1 and 3, and nothing else of significance for section 2.
- When the missing values are filled in with other notes, Voice 6 shows the same type of behavior as it does for the overall analysis. Interestingly, however, when the missing values are filled in with zeroes, the negative partial correlations are almost exclusively in sections 1 and 3. (Section 2 has some negative values, but they are barely outside the confidence bands.) Conversely, Section 2 shows extremely large positive spikes that did not exist in the PACF for Voice 6 in its entirety. The peaks that appear in the PACFs for the sections are at different lags than those in the full PACF. Figure 11 shows the 4 PACFs for Voice 6. Notice that the PACFs for Sections 1 and 3 are nearly identical.
- The cross-correlations between Voice 1 and the other voices still exhibit negative behaviors. Now, however, there are many, many different patterns, depending on the voices as well as the sections. Figures 12 through 14 show the very different behavior for the ACFs of Voices 1 and 3 (with zeroes for missing values) for the whole piece; for section 1 only; and for section 2 only. Notice the large differences between the plots. In Figure 12, the cross-correlation is positive and strongly periodic. Figure 13 shows the cross-correlation for the two voices on the first (and third) section of the piece. The correlations are of the same magnitude, but there are both positive and negative values. The correlations for section 2 (shown in Figure 14) are very strong and positive, with no clearly discernible pattern, and slowly tapering off. Other voice/section combinations show different patterns, e.g. monotonically increasing or decreasing.

3.2.2 Number Notes Series

Comparatively, the autocorrelation on the number of notes played is fairly straightforward. Figure 15 shows the ACF plot. There is a strong positive relationship at all lags – the correlation never goes below 0.5. Once again, this is not very surprising, as the number of notes played displays little variability.

A very different story appears if this series, also, is split into the three sections. In this case, very few values are outside the bounds, and there may be a sinusoidal shape to the correlations for section 2. The PACF, on the other hand, has alternating positive/negative values for section 1, and only three values outside the bounds. Section 2 shows no pattern, and, again, only 3 significant values. The PACF is given in Figure 16.

In Section 3.4, the correlation analyses will be used together with the spectral analysis of Section 3.3 to fit autoregressive processes to the data.

⁵ We make the assumption that the Section 1 and Section 3 series are nearly identical, thus reducing the total number of possible series from 48 to 36. The assumption appears valid based on the cases where the analyses were done for both Sections 1 and 3; there was virtually no difference between the results.

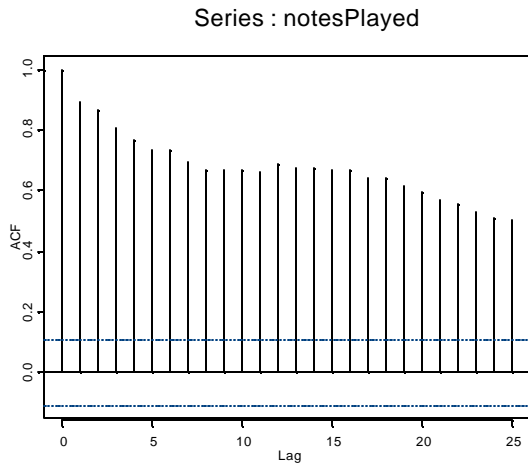


Figure 15 ACF of number of notes played

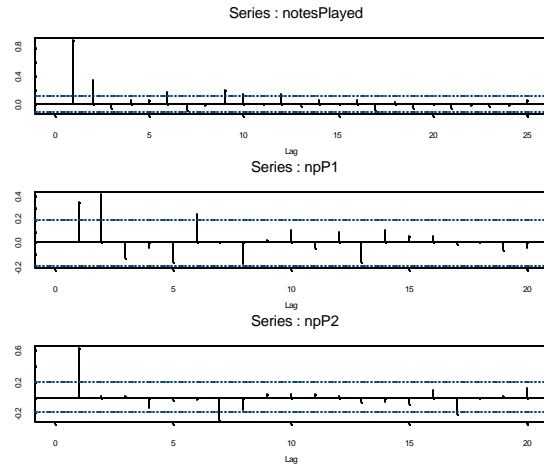


Figure 16 PACF for notes played: whole series, Section 1, Section 2

3.3 Spectral Analysis

3.3.1 Voice Data Series

The plots of the raw voice data in Figures 2 and 3 show definite periodicities in the data. In addition, the ACFs and PACFs show definite relationships between the data and periodicities within the relationships. In this section, we investigate these relationships and periods further using spectral analysis. The spectral analysis is carried out on the whole voices, not on the different sections, since the differences caused by the sections could be accounted for by a wave of very low frequency.

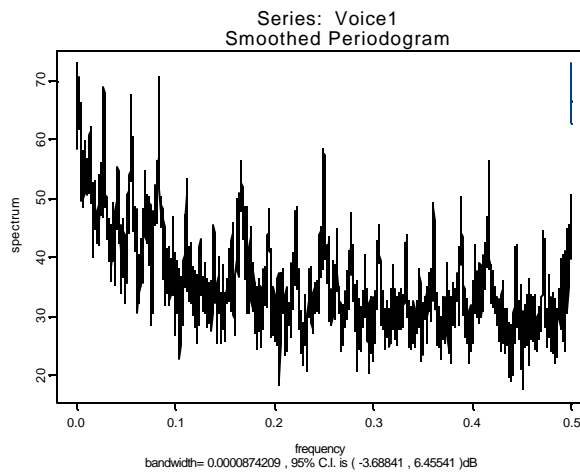


Figure 17 Periodogram for Voice 1

Figure 17 shows the smoothed periodogram for Voice 1. There are many peaks, and it is difficult to say which are more significant than the others.

The cross-spectra for the different voices underscore the results from the previous section. When the missing values are filled in randomly, the spectra for voices i and j show less and less structure with increasing voice number, and the spectral values increase. The closer i and j are, the closer their spectra. Figure 18 shows the cross-spectrum for Voice 1 and Voice 2; and the cross-spectrum for Voice 1 and Voice 6. The former shows the two spectra nearly indistinguishable. For the smoothing parameters in this example (3 and 3), the coherency appears fairly large and constant for all frequencies. When the values of the smoothing parameters are increased, the coherency decreases for larger frequencies. This indicates that the squared correlation between the components at the higher frequencies tend to get smaller when the periodogram is smoothed more.⁶

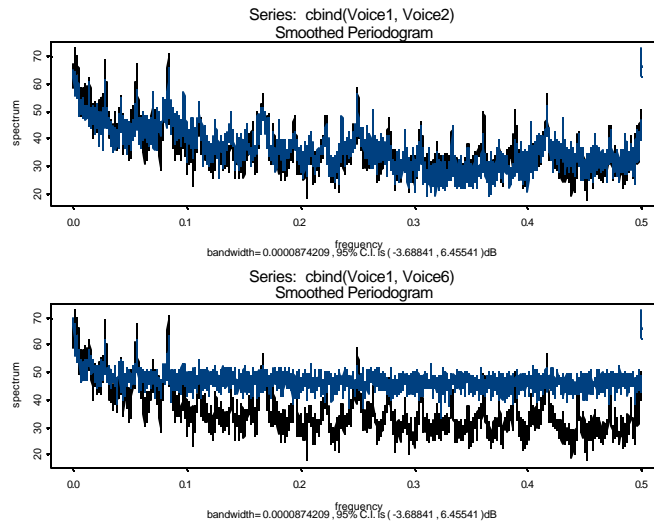


Figure 18 Cross-spectra for Voice 1/Voice 2, Voice 1/Voice 6 (random)

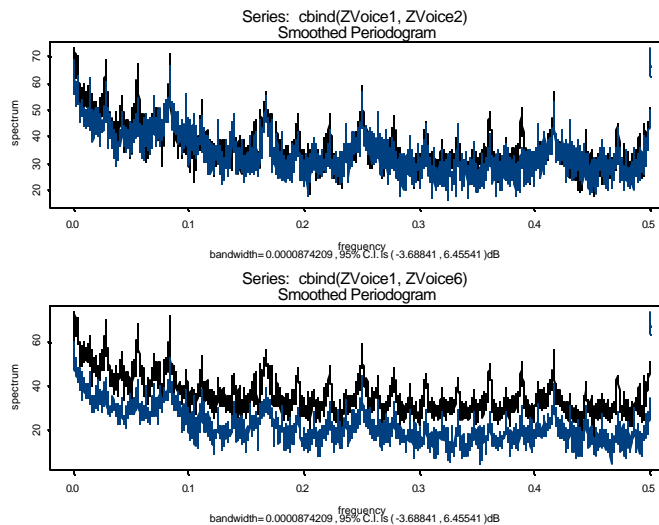


Figure 19 Cross-spectra for Voice 1/Voice 2, Voice 1/Voice 6 (zeroes)

The cross-spectrum for Voice 1 and Voice 2 shows several large peaks. The 10 largest peaks (and their associated frequencies/periods) are as follows:

⁶ No graphs for coherence are presented here, as they were fairly noisy. When adding confidence bands, it became impossible to discern anything from them.

| Frequency | Period | Spectral Estimate |
|--------------|--------|-------------------|
| 0.001427852 | 700 | 70.34 |
| 0.001511843 | 660 | 70.43 |
| 0.001007895 | 992 | 70.60 |
| 0.08315136 | 12 | 70.72 |
| 0.0005039476 | 1984 | 71.64 |
| 0.000923904 | 1082 | 71.83 |
| 0.0008399127 | 1189 | 72.52 |
| 0.0007559214 | 1322 | 72.59 |
| 0.0006719301 | 1488 | 72.67 |
| 0.0005879389 | 1700 | 72.84 |

None of the frequencies are Fourier frequencies. Some of the periods are easily explainable. For example, period 12 is the length of a quarter note (or four sixteenth notes). In Sections 1 and 3, patterns are very often repeated (at different pitches) every quarter note. For others, it is far less obvious – what occurs every 47 measures (period 1700)?

When considering the voices where missing values are filled in with zeroes, we again observe much stronger correlations between the voices, especially higher-numbered voices, than we do when the values are filled in with random values. Again, this is not surprising, since the voices will tend to have far more “notes” in common than they do in the previous case (there are many rests). Figure 19 shows the cross-spectra for Voices 1 and 2; and Voices 1 and 6. Voices 1 and 2 display the identical behavior they do above – they are virtually indistinguishable. The more interesting plot is that for Voice 1 and Voice 6. Here, Voice 6’s periodogram looks the same as Voice 1’s, but is shifted down. This echoes the sharp periodicities in the ACF for Voice 6 discussed above. These graphs are characteristic of all other cross-spectra: The closer the voice indices, the closer the spectra. If the indices are far apart, the spectra will be shifted vertically, but retain the same shape.

Experimenting with different smoothing factors did have an impact on the noisiness of the periodogram and coherence plots. The general conclusions, however, remain the same as the shapes of the plots are retained.

3.3.2 Number Notes Series

The smoothed periodogram for the number of notes played per measure can be seen in Figure 20. There are clearly periodicities in the number of notes played. The local max at roughly frequency 0.05 corresponds to a period of about 16 measures. Perhaps this corresponds to the overall set of arpeggios in Section 1 and Section 3.

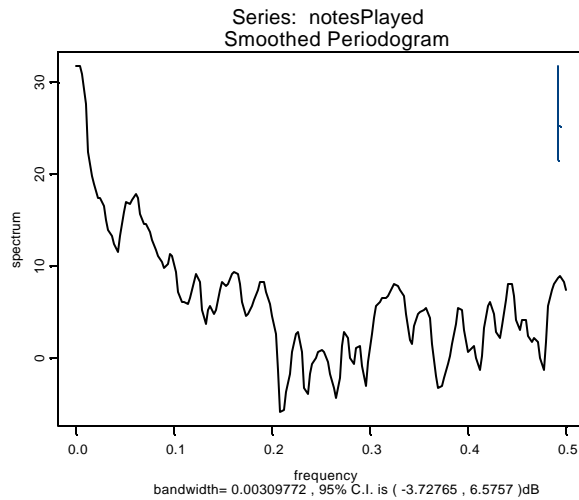


Figure 20 Periodogram of the number of notes played

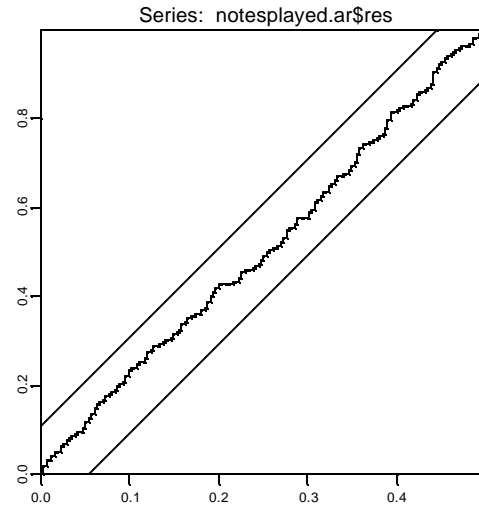


Figure 21 Cumulative periodogram of the residuals of the AR(12) model

3.4 Autoregressive Analysis

3.4.1 Voice Data Series

The correlation and spectral analyses of the previous two sections are now used to try to fit an $ARIMA(p, d, q)$ process to the data. Before using the additional information gained from the other analyses, S-Plus is allowed to select the “best” order for the models.

The first attempt at fitting an $AR(p)$ model to the series was done using the `ar()` function in S-Plus, and letting it automatically determine the order. This was done for the voice for the entire piece, and for the voice series for each of the sections. Of the 36 series, only one had $p < 20$ (it was 15). All but two of the AR models for the complete series were of order $40 = \text{order.max}$. Most others were around order 35. In all “section” cases, the cumulative periodograms of the residuals appeared acceptable, with no points falling outside the bounds and no obvious patterns appearing. The $AR(p)$ models for the full series do not perform well (residuals are not random) – which is expected, since the series are not variance-stationary, nor are they correlation-stationary (see above).

These models are not good from two points of view. The first is that the order of the models is fairly large, which requires much parameter estimation. The second is that the residuals are very ill-behaved. For example, the arpeggio structure is clearly visible in the residuals from the $AR(33)$ model for Voice 1, Section 1.

This obvious pattern in the residuals highlighted a significant problem with the data, a problem that would also cause trouble when trying to fit an $ARIMA(p, d, q)$ model: There is a definite “seasonal” component to the data. The S-Plus function `stl()` was used to try to remove the seasonality, and the seasonal component that resulted was sizeable. Unfortunately, so was the remaining pattern in the residuals – they still appeared nearly identical to the original series. Differencing was used to try to remove the patterns, again to no avail. (Two differencing

procedures had the largest impact – diff(1); or diff(72) followed by diff(6). Neither produced adequate results, however.)⁷

Despite the lack of success in trying to deseasonalize the data, several attempts were made to fit the $ARIMA(p,d,q)$ model. Taking a queue from the period of length 12 that appeared in the cross-spectrum for Voice 1 and Voice 2, p was set to 12. The residuals were anything but random. Indeed, they looked like the graphs one sometimes sees of speech. The same shape reappeared in the residuals regardless of what values were chosen for p , d , and q .

It is unfortunate that we were unable to fit a good model, as it could have been used to try to do a rudimentary prediction of what the next 2 minutes of the piece *would* have sounded like, had Schubert continued composing.

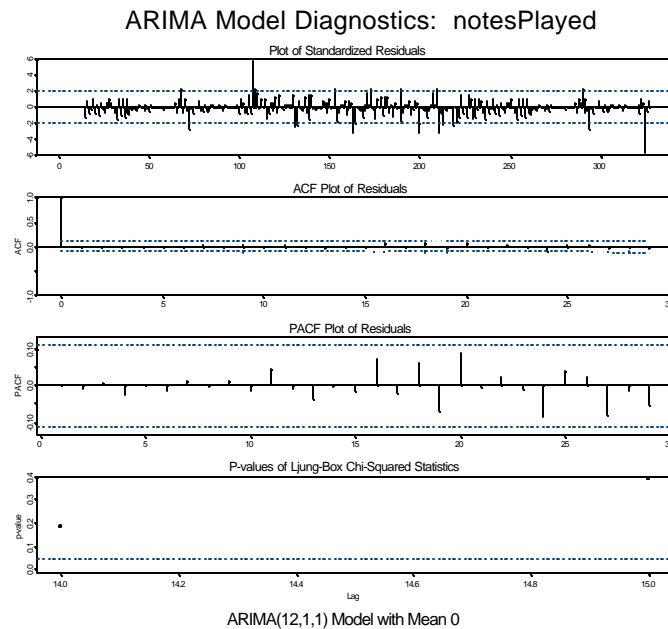


Figure 22 Diagnostics for the $ARIMA(12,1,1)$ model of notes played

3.4.2 Number Notes Series

Trying to fit a model to the number of notes played per measure is slightly more successful. For Voice 1/Section 1, $ar()$ fits a model of size 12, which is far below the `order.max` of 25. Even the residuals are relatively well-behaved. Figure 21 shows the cumulative periodogram for the residuals – they fall well within the bounds, and seem to follow a straight line. It is unclear what significance the value of 12 has here – the series are per measure, not per tatum; apparently, there is a relationship between 12 nearby measures.

Since we were moderately successful in fitting a simple $AR(p)$ model, an attempt was made to fit an $ARIMA(p,d,q)$ model. The value of p was left at 12, while d and q were varied. The diagnostics for one of the models ($p = 12$, $d = 1$, $q = 1$) can be seen in Figure 22. There is still an obvious pattern in the residuals that has not been accounted for by the model. The pattern remained virtually unchanged for all values of d and q tried.

⁷ I was/am at a loss of what else could be done to try to combat the patterns in the data. The TA was also unable to help.

4. DISCUSSION AND CONCLUSIONS

This project does a basic data analysis of a piece of piano music by Franz Schubert. Score representation was chosen because the author felt it was more interesting than the “conventional” approach, and that there would be less noise in the data. These benefits came at a cost – instead of having a single data series, there are six with many “missing values.” The time spent trying to transform the data into a useful format was time that was no longer available for more detailed analyses.

More frustrating aspects of the study were those discussed in the last section – trying to remove obvious patterns from the data. No doubt it is possible to do so, I was just unable to find a way to do it.

On a positive note, it was gratifying to see that some of the things that are clearly discernible from looking at the score or listening to the piece are also discernible from the time series. The most obvious example is the division of the piece into three parts. Less obvious were periodicities within quarter notes etc. It would have been interesting to see what things were discernible from the time series that are *not* immediately obvious to the casual observer of the score or the piece.

5. FUTURE WORK

An extremely interesting analysis not done for this project were the nonlinear methods studied in class. In particular, it would be illuminating to see whether the methods would be able to detect repeated patterns in the piece. A very obvious repeated pattern would be Section 1 in its entirety – apart from transitional (and ending) sections, Section 3 is a carbon copy of Section 1. It would be interesting to see whether the nonlinear methods are able to detect this. Similarly, Section 2 consists of two parts, both of which are repeated. This is another aspect that should be detectable by the pattern-detecting methods.

In trying to deseasonalize/detrend the data made the advantages of wavelet analysis very obvious. There are too many subsections of the piece to be able to nicely fit one model to it. Even separating it into its three main sections did not help attain stationarity very much. Wavelet analysis would, at the very least, allow the piece to be considered as a whole and be able to fit a reasonably accurate model. Unfortunately, there was not enough time to code it all.

Finally, more advanced analyses would be fascinating – fitting HMMs to the series, for example, or trying to do principal components or other analyses. Again, however, much time was lost trying to obtain data and then convert it to an analyzable format, so such formal analyses were not possible.

6. ACKNOWLEDGEMENTS

The author wishes to thank Rafael Irizarry and David Huron for their efforts on her behalf. Lee Schruben provided helpful insights. Especial thanks go to David Wessel of the UC Berkeley CNMAT, without whom this project would not have been possible.

7. REFERENCES

Brillinger, D. and R. Irizarry. 1998. "An investigation of the second- and higher-order spectra of music." *Signal Processing* vol. 65: p.161-79.

Brillinger, D. 2001. "Time series: data analysis and theories." *Society for Industrial and Applied Mathematics*.

Kantz, H. and T. Schreiber. 1999. "Nonlinear time series analysis." *Cambridge University Press*, Cambridge, MA.

Venables, W. and B. Ripley. 1999. "Modern applied statistics with S-PLUS." *Springer-Verlag*, New York.

<http://www.statsoft.com/textbook/stathome.html>

APPENDIX: CODE

C++ code to convert raw data to required format

main.cpp

```

/* Program to convert raw SEEK-format data to tab-delimited columns.
   Converts times in milliseconds to integer-valued number of tatums.
   One tatum = one 16th note = around 146ms. There are several outputs:
   1. raw data, tab delimited
   2. two columns of values, tatums for On-Off and tatums for On-On
   3. single column of values, frequencies
   4. MIDI notes, length held, starting time (in tatums)
   5. 6-columned output containing the 6 voices */
#include <stdlib.h>
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#ifdef __UTILITIES_H
#include "utilities.h"
#endif

main()
{
    ifstream inputFile;
    ofstream rawTabDelimitedFile;
    int i = 0,
        j = 0,
        lastStartTime = 0, // basically the effective # rows for voices
        index = 0, // dummy variable to read in index # of raw data
        MIDICannel = 0, // " " for channel # in raw data
        MIDINotes[NUMBERDATAPOINTS], // original MIDI values
        volume[NUMBERDATAPOINTS], // original "velocities"
        onOffTimes[NUMBERDATAPOINTS], // note durations in msec
        onOnTimes[NUMBERDATAPOINTS], // times between successive notes in msec
        onOffTatums[NUMBERDATAPOINTS], // note durations in tatums
        onOnTatums[NUMBERDATAPOINTS], // times between successive notes in tatums
        startTimes[NUMBERDATAPOINTS]; // start times of the notes, in tatums
    float frequencies[NUMBERDATAPOINTS],
        voices[TOTALTATUMS][NUMBERVOICES]; // contains the notes for the voices
    char ch = 'a';

    inputFile.open( "schubert-all.txt", ios::in );
    rawTabDelimitedFile.open( "schubertRaw.txt", ios::out );
    if ( ( !inputFile.is_open() ) || ( !rawTabDelimitedFile.is_open() ) )
    {
        cout << "Error opening file, exiting.";
        exit( -1 );
    }
}

```

```

// initialize arrays
for ( i = 0; i < NUMBERDATAPOINTS; i++ )
    frequencies[i] = 0.0;
for ( i = 0; i < TOTALTATUMS; i++ )
{
    for ( j = 0; j < NUMBERTATUMS; j++ )
        voices[i][j] = 0.0;
}
// read in raw data, output tab-delimited file
for ( i = 0; i < NUMBERDATAPOINTS; i++ )
{
    inputFile >> j >> ch >> MIDIChannel >> MIDINotes[i] >> volume[i];
    inputFile >> onOffTimes[i] >> onOnTimes[i];
    inputFile >> ch;
    rawTabDelimitedFile << j << "\t" << MIDIChannel << "\t" << MIDINotes[i] << "\t";
    rawTabDelimitedFile << volume[i] << "\t" << onOffTimes[i] << "\t" << onOnTimes[i];
    rawTabDelimitedFile << "\n";
}
computeTatumLengths( onOffTimes, onOnTimes, onOffTatumLengths, onOnTatumLengths );
computeFrequencies( MIDINotes, frequencies );
computeStartTimes( onOnTatumLengths, startTimes, lastStartTime );
createVoices( frequencies, startTimes, onOffTatumLengths, voices, lastStartTime );
inputFile.close();
rawTabDelimitedFile.close();

return(0);
}

```

Utilities.cpp

```

#ifndef __UTILITIES_H
#include "utilities.h"
#endif
#include <math.h>
#include <assert.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>

// computes the tatum lengths, outputs to tatum length file
void computeTatumLengths( int _OnOffArray[], int _OnOnArray[], int _OnOffTatumLengths[], int _OnOnTatumLengths[] )
{
    ofstream tatumFile;
    int i = 0,
        j = 0,
        tatumValues[MAXLENGTH],
        minOnOffTatum = 0,
        minOnOnTatum = 0,
        minIndex1 = 0,
        minIndex2 = 0,
        difference = 0;
    tatumFile.open( "schubertTatum.txt", ios::out );
    if ( !tatumFile.is_open() )
    {
        cout << "Error opening tatum file, exiting.";
        exit( -1 );
    }
    for ( i = 0; i < MAXLENGTH; i++ )
        tatumValues[i] = i*TATUMLENGTH;
    for ( i = 0; i < NUMBERDATAPOINTS; i++ )
    {
        minOnOffTatum = 2000;
        minOnOnTatum = 2000;
        for ( j = 0; j < MAXLENGTH; j++ )
        {
            difference = abs( tatumValues[j] - _OnOffArray[i] );
            if ( difference < minOnOffTatum )
            {
                minOnOffTatum = difference;
                minIndex1 = j;
            }
            difference = abs( tatumValues[j] - _OnOnArray[i] );
            if ( difference < minOnOnTatum )
            {
                minOnOnTatum = difference;
                minIndex2 = j;
            }
        }
        _OnOffTatumLengths[i] = minIndex1;
        _OnOnTatumLengths[i] = minIndex2;
        tatumFile << minIndex1 << "\t" << minIndex2 << "\n";
    }
}

```

```

    }
    tatumFile.close();
    return;
}
/* ----- */
// computes the note frequencies, outputs them to file
void computeFrequencies( int _MIDINoteArray[], float _frequencyArray[] )
{
    int i = 0;
    ofstream frequencyFile;
    frequencyFile.open( "schubertFreq.txt", ios::out );
    if ( !frequencyFile.is_open() )
    {
        cout << "Error opening frequency file, exiting";
        exit( -1 );
    }
    for ( i = 0; i < NUMBERDATAPOINTS; i++ )
    {
        _frequencyArray[i] = (float)8.175799*(float)pow( 2,(float)_MIDINoteArray[i]/12.0);
        frequencyFile << _frequencyArray[i] << "\n";
    }
    frequencyFile.close();
    return;
}
/* ----- */
// finds start times (in tatums) of each note, as well as the last time a note is started
void computeStartTimes( int _OnOnArray[], int _startTimeArray[], int &_lastTime )
{
    ofstream startTimeFile;
    int i = 0,
        currentStartTime = 0;
    startTimeFile.open( "schubertStartTimes.txt", ios::out );
    if ( !startTimeFile.is_open() )
    {
        cout << "Error opening start time file, exiting.";
        exit( -1 );
    }
    _startTimeArray[0] = 0;
    startTimeFile << _startTimeArray[0] << "\n";
    for ( i = 1; i < NUMBERDATAPOINTS; i++ )
    {
        currentStartTime = currentStartTime + _OnOnArray[i-1];
        _startTimeArray[i] = currentStartTime;
        startTimeFile << _startTimeArray[i] << "\n";
    }
    _lastTime = currentStartTime;
    startTimeFile.close();
    return;
}
/* ----- */
// resets all the values in the array to 0
void resetArray( int _sortArray[], int _arraySize )
{
    int i = 0;
    for ( i = 0; i < _arraySize; i++ )
        _sortArray[i] = 0;
    return;
}
/* ----- */
// sorts the _numberNotes in the _sortArray[] in decreasing order
void sortValues( float _sortArray[], int _numberNotes, int _arraySize )
{
    int i = 0,
        j = 0,
        k = 0;
    float *tempArray = NULL,
        tempValue1 = 0,
        tempValue2 = 0;
    bool foundSpot = false;
    tempArray = new float[_numberNotes];
    assert( NULL != tempArray );
    for ( i = 0; i < _numberNotes; i++ )
        tempArray[i] = 0.0;
    tempArray[0] = _sortArray[0];
    for ( i = 1; i < _numberNotes; i++ )
    {
        j = 0;
        foundSpot = false;
        while ( !foundSpot )
        {
            if ( _sortArray[i] > tempArray[j] )

```

```

        {
            foundSpot = true;
            k = j + 1;
            tempValue2 = tempArray[j];
            tempArray[j] = _sortArray[i];
            while ( k <= i )
            {
                tempValue1 = tempArray[k];
                tempArray[k] = tempValue2;
                tempValue2 = tempValue1;
                k++;
            }
        }
        else
            j++;
    }
}
for ( i = 0; i < _numberNotes; i++ )
    _sortArray[i] = tempArray[i];
delete []tempArray;
return;
}
/* ----- */
// randomly selects one of _numberNotes numbers
int selectNote( int _numberNotes )
{
    int note = 0,
        i = 0;
    float randomValue = 0.0;
    bool foundNote = false;
    randomValue = (float)rand()/(float)RAND_MAX;
    while ( !foundNote )
    {
        if ( ( i + 1 ) >= _numberNotes*randomValue )
        {
            foundNote = true;
            note = i;
        }
        else
            i++;
    }

    return( note );
}
/* ----- */
int createVoices( float _frequencyArray[], int _startTimeArray[], int _OnOffTatums[],
                 float _voices[TOTALTATUMS][NUMBERVOICES], int _lastStartTime )
{
    ofstream allVoicesFile,
              allVoicesWithZeroesFile,
              numberNotesFile;
    int i = 0,
        j = 0,
        currentTime = 0,
        totalTime = 0,
        counter = 0,
        tempIndex = 0,
        endingTime = 0,
        noteIndex = 0,
        numberVoices[TOTALTATUMS]; // the "real" # voices at each time
    bool isRest[TOTALTATUMS];
    allVoicesFile.open( "schubertAllVoices.txt", ios::out );
    allVoicesWithZeroesFile.open( "schubertAllVoicesWithZeroes.txt", ios::out );
    numberNotesFile.open( "schubertNumberNotes.txt", ios::out );
    if ( ( !allVoicesFile.is_open() ) || ( !allVoicesWithZeroesFile.is_open() ) ||
        ( !numberNotesFile.is_open() ) )
    {
        cout << "Error opening all voices file(s), exiting.";
        exit( -1 );
    }
    for ( i = 0; i < TOTALTATUMS; i++ )
        isRest[i] = true;
    resetArray( numberVoices, TOTALTATUMS );
    for ( i = 0; i < NUMBERDATAPOINTS; i++ )
    {
        // read through data, place in numberVoices[i]'th spot in array
        counter = 0;
        currentTime = _startTimeArray[i];
        totalTime = _OnOffTatums[i];
        while ( counter < totalTime )
        {

```

```

tempIndex = currentTime + counter;
if ( tempIndex > endingTime )
    endingTime = tempIndex;
_voices[tempIndex][numberVoices[tempIndex]] = _frequencyArray[i];
numberVoices[tempIndex]++;
if ( numberVoices[tempIndex] > 0 )
    isRest[tempIndex] = false;
counter++;
}
}
// fill in "missing values" and write to files
for ( i = 0; i < endingTime; i++ )
{
    if ( !isRest[i] )
    {
        sortValues( _voices[i], numberVoices[i], NUMBERTONES );
        for ( j = 0; j < numberVoices[i]; j++ )
        {
            allVoicesFile << _voices[i][j] << "\t";
            allVoicesWithZeroesFile << _voices[i][j] << "\t";
        }
        for ( j = numberVoices[i]; j < NUMBERTONES; j++ )
        {
            noteIndex = selectNote( numberVoices[i] );
            _voices[i][j] = _voices[i][noteIndex];
            allVoicesFile << _voices[i][j] << "\t";
            allVoicesWithZeroesFile << "0 \t";
        }
    }
    else
    {
        for ( j = 0; j < NUMBERTONES; j++ )
        {
            allVoicesFile << "0 \t";
            allVoicesWithZeroesFile << "0 \t";
        }
    }
    allVoicesFile << "\n";
    allVoicesWithZeroesFile << "\n";
    numberNotesFile << numberVoices[i] << "\n";
}
allVoicesFile.close();
allVoicesWithZeroesFile.close();
numberNotesFile.close();
return( endingTime );
}

```