**Canonizing Melodies**

**Daniel Tompkins**

**Fall 2015**

*Things found in the final project folder:*

* *Python Files (especially CanonMaker and Similarity)*
* *Prolation tests (some melodies for testing)*
* *Bach WTC Subjects (for comparisons)*
* *Scores (scores of various melodies that were successful given certain rules)*
* *Images (histograms of comparisons, similarities, etc.)*
* *CodeWriteup.html (blow-by-blow explanation of the code)*
* *Poweroint (a more general blow-by-blow with more general remarks)*

**Introduction**

For this project, I created two main modules: “CanonMaker” which makes canons with a given melody, and “Similarity” that analyzes correlation among the data from CanonMaker and other parameters. My goal when creating these modules was: to test a melody’s ability to be in counterpoint with itself given certain rules; to compare canon success rate of multiple melodies; and to create scores that contain all possible combinations. These goals were successful, although the value of the comparisons and correlations are still to be determined. I will describe how the program works, problems and possible solutions, and where I hope to go next with these ideas. For a blow-by-blow description of each function, see the HTML file in the folder included with this project. For this paper, I will focus on generalities of the project and discuss the successes and areas of the project that needs more work.

**CanonMaker**

To make a canon, we begin with a single melody as an .xml file. The melody gets altered by some combination of transposition, offset and prolation and is combined with its original self in a music21 score. A function then takes all of the harmonic intervals of that score by chordifying the two parts and getting the absolute difference between each note and returning those numbers in a list. The intervals are then sent through two testers to see if the canon should be rejected or accepted.

The testing phase was the trickiest in that it was a balancing act to figure out how best to keep the “good” canons and reject the “bad.” If the rules are too restrictive, a melody may look less successful than it is because many acceptable canons will be rejected. If the rules are too lax, many bad canons will be accepted.

The first tester is the parallel\_checker function. It checks for parallel fifths and octaves/unisons and optionally tests for too many parallel thirds and sixths. This function was straightforward and was generally successful at taking out bad canons while keeping good canons. And rather than immediately rejecting a canon after the first offence, it keeps a tally of offences so the user can create a threshold if he or she wishes to allow, for example, a few instances of a string of parallel thirds.

The second tester is the resolve\_diss function. It defines dissonances as pc-intervals 1, 2, 5, 6, 10, 11. If two adjacent intervals were dissonant, it checks the dissonance tally. This was very successful at taking out the bad canons while accepting the good, although there are certain contrapuntal instances where adjacent dissonances are acceptable and would be rejected. The next most successful argument is that if an interval is a tritione, it must resolve outward or inward. I included many optional lines of code that micromanages other dissonances, such as making a second go to either a unison or third. After passing through all of the other arguments, these were typically too restrictive. Although, this also has the threshold option if the user wants to allow some offences.

I hope to find a clear way to check to see if a dissonance is a suspension, neighbor, passing, etc. and to know the difference between one voice moving and both voices moving (learning attack vs sustain is not possible with the chordify method). I would also like to find a way to identify a compound melody, so a dissonance can wait to be resolved until the correct voice of the melody resolves it.

Music21 has some modules in process that could potentially do this work too, but it also seems to be stuck in figuring out the balance between being too liberal and too micromanaging with dissonances. This micromanaging can be very useful as a species counterpoint tester, however. It can be easily adapted to fit Fux, Jeppesen, etc.

I made three levels of “testers” to find all acceptable canons. The first tests a single melody given a function and a list of amounts and returns the data to a dictionary. For example, it can take a melody and test offsets [0, 1, 2, 3, 4]. A score is created and tested for each combination (five in this given example). The successful function amounts will be included in the dictionary.

The second “tester” tests multiple functions (alterations to the melody). Given global lists for transposition and prolation amounts, these possibilities are combined with the number of offsets, by quarter note, up to the halfway point of the melody. Each possibility is again tested, and the successful canons are returned. This could be the opposite in that changing only one word can return or accept all of the bad canons and reject the good. This can be good for determining how successful various rules are. This tester also has the option of having all the successful (or unsuccessful) canons combined onto a score.

The third “tester” multiple .xml files containing a single melody in a folder. It separates each melody and runs the second tester function for each one. It collects the success rate (either percentage or success number) into a dictionary and prints a histogram showing the canon success rates of the melodies.

**Similarity**

The “Similarity” module finds correlation in the data of CanonMaker and other various qualities of given melodies. I used the 48 subjects from Bach’s Well-Tempered Clavier books 1 and 2 for comparison. I primarily used Pearson’s R to find correlation, although I included code for finding the cosine similarity. My goal was to see if there were any properties of the fugue subjects that would correlate with canon success. I tested the success with melody length, rhythmic variance, melodic variance, most common interval, first interval, tonic pitch, and first pitch. The only strong correlation was that of melody length; shorter melodies tend to have better canon success. While this seems a bit trivial or at least mundane, there are many other parameters that can be tested.

**Conclusion**

As Cuthbert told our class, it is finding the interesting questions that is the hard part. As a creative tool, I find the CanonMaker to be successful. It can find interesting combinatorial solutions much faster than a human can, and I find many of the results interesting. I also find the problem of rules to be a good thought experiment to see which rules matter most when trying to eliminate most of the bad melodies.

The Similarities module shows the difficulty in navigating the complexity of musical syntax. I would ideally like to have a much more specific set of data to compare with canon success, such looking at intervals that happen on a given beat, and taking it into context. The sample size of the WTC fugues is too small for something so specific, but perhaps a much larger corpus of fugue subjects might yield some interesting data.

Moving forward, I will be using many of these tools in CanonMaker and Similarity to look at early Italian Baroque continuo realization. I will be using much of Kapsperger’s published work to figure out the likely harmonization given a bass movement, and I will analyze the differences between those likelihoods given instrumentation (theorbo, baroque guitar, keyboard). I think there can be some interesting findings and also some practical applications to continuo players who often struggle with harmonizing unfigured basslines. The comparison tools will also allow me to look at the peculiar voice-leading in the Spanish Baroque guitar, which was commonly used as part of a continuo band, and speculate on why its nonconformist voice-leading was allowed but not other instruments.