MelGen: Monophonic Melody Generation

Shauna Morrisey
Department of Music Technology
Georgia Institute of Technology
Atlanta, GA 30318, USA
smorrisey@gatech.edu

1. Introduction

The intent behind this work was to create a late-stage module for an aural skills and music theory training software, called Melodia. Melodia currently has two early modules for self-directed music theory education through aural skills. Users listen to major and minor chords from the diatonic scale and are asked to arrange the chords in specific ways to move on. MelGen would be included as the generation tool powering multiple later components. Some of these components could be dictation, sight singing, or harmonization exercises.

A popular topic in education software explores how users can learn on their own in an engaging and meaningful way. Examples of aural skills and music theory learning softwares are Harmonia (Taube, 2018) and Teoria (Alvira, 2019).

Harmonia is primarily designed to create an easy interface for an online course. Harmonia allows teachers to create assignments, and allows students to submit their assignments. Harmonia also has automatic grading system to expedite the teacher's workload. Harmonia's ability to automatically create assignments based on a set of criteria from the instructor is done well and is meaningful for the implementation of generated material for guided education. Harmonia is not designed for the self-paced independent user. Harmonia does not provide great explanations as to how it is grading and correcting. The corrections require the user to have some basic knowledge. MelGen is designed to provide the generation that Harmonia can provide, while Melodia provides the explanation and basic knowledge that Harmonia is missing.

Teoria is a free aural skills and music theory website. The dictation and aural skills options come from a set of existing files. By having a fixed set of materials, students may eventually reach a point where they receive repeated material. The automatic grading feature implemented into this system is clunky and imperfect. The grading implementation does not recognize multiple possible solutions, like a half note compared to a quarter note and a quarter rest. MelGen can provide new melodies so that students never get repeated material.

Symbolic melody generation is currently a popular topic in music technology. Melody generation is an old problem that attempts to formalize the way humans write music. With recent developments in technology, these music generators have evolved to be much more complex, yet still do not meet objective goals. There are two goals. A piece generated by the generator should be similar to the data it is based on. The style should be similar. If presented two pieces, someone should not be able to tell which piece was generated and which was from the original data. The second goal for a generator is

that it should not plagiarize the dataset. Papadopoulos et al suggest an altered Markov model with a maximum order, or the longest amount of notes that may be quoted at a time from the dataset (Papadopoulos, 2014.)

Current approaches to symbolic music generation typically take one of three forms. One method of music generation is a rule-based approach. Rules for period-appropriate counterpoint and part-writing have been formalized in treatises and textbooks such as *Gradus ad Parnassum* by Johann Fux. By writing a program to follow these rules, the generated output mostly matches the stylistic norms for the time period on which they were trained. Rule based systems unfortunately tend not to mimic pieces from their time period. Since many "rules" in Western tonal writing systems commonly have ment exceptions, the scenarios where a rule can be ignored or modified are frequently ambiguous, undefined, or too prolific. These exceptions create issues that are often too complex for a rule based systems.

Answering questions of how to break rules in a rule based system often rely on a second method of implementing music generation, a probability-based approach. A probability-based approach to symbolic music generation takes datasets and simplifies the data down to the likelihood that any one event would follow any other event. Probability based approaches can overfit (i.e. replicate) their data if not enough data has been used. In Classical music and some folk music, where repeated notes are extremely common, a probabilistic approach requires larger-order probability data in order to avoid getting "stuck" repeating simple two- or three- note patterns. Many probability-based approaches are reliant on Markov models. Conklin overviews several ways to implement a Markov model as a music generation system (Conklin, 2003.)

Some previous work of the author has fit into this field. Mostly relying on a rule based approach, this previous attempt was written in the visual programming language, Max/MSP. It used a set of chord progression probabilities of Bach chorales to inform a rule based system. Using the chord progression chart and melody writing rules found in *Tonal Harmony* (Kostka 2017), the system created a two part melody and accompaniment generator. This generator was limited in that the melody had no rhythmic variety and was difficult to manage successfully.

The most common modern approach to symbolic music generation currently relies on machine learning. Most notably, deep learning, which is a popular machine learning approach to computer simulation of many complex systems. One notable example of symbolic music generation using a deep neural network is MuseGAN (Dong, 2018). While deep learning approaches have provided some improvement in generative music systems, there are still several drawbacks to this approach. For instance, like probabilistic approaches, they rely on large datasets, do not always allow for a lot of programmer manipulation and are computationally intense.

2. Implementation

2.1 Project Goals

MelGen is written in python and depends on music21 and pandas. Music21 is a toolkit for python that is used for the manipulation of symbolic musical data. Music21 includes a subset of the Essen Folksong database, primarily European folk music. The Essen folk music dataset is all monophonic. While music21 can detect keys, it can only detect major and minor modes. In other words, it lacks the ability to recognize modal material. For the goals of MelGen, modal material has

created some noise in the data. It has not caused any large effects.

Since MelGen is intended to eventually be implemented on a website, the probability based approach was the best option. A probability based approach would likely perform better than a rule based approach, but are not as computationally expensive as a machine learning or deep learning approach.

MelGen is to be used as the content for dictation or harmonization exercises. Dictation or sight singing could be implemented with a user friendly interface.

2.2 Implementation Methods

The code for MelGen can be split into three categories. These categories are parsing data, creating probability tables, and generation. Parsing the data includes opening each file, accessing the scores, parts, notes and rests in each file. There are many different probability tables that are useful to music generation. Finally, the main component is music generation. A flowchart describing these components and how they interact is in the appendix.

In MelGen's implementation, only the data parsing and table generation happens ahead of time. Data parsing is the most memory intensive component of MelGen. By saving the tables the Markov model uses as CSV files, the music generation components of MelGen can happen in real time.

To do the data parsing, a function starts by opening each subfolder until it reaches a score. For each score, the key is extracted. Based on the mode of each score, the score is transposed to C major or A minor. The transposed scores are separated into two groups based on the detected mode. Each group of scores is then passed through code that extracts all the notes and rests and adds them into a pandas series. The output is two series of notes and rests, one for major scores and one for minor scores. At this point, a marker is added to the end of each score. The end marker is important because it allows queries of the data for first and last note of a score. Excluding these end markers, there are about 16,000 notes and rests in just one folder of the dataset.

The probability tables are generated from this list of notes and rests. These tables are stored as a pandas Series or Dataframe. Each table is given a number based on the number of conditions it is using, starting at zero. The Zeroth-order probability tables are the simple distribution of musical events according to pitch or duration. The zeroth-order pitch distribution is shown in Figure 1. The first-order probability tables determine the likelihood of any one pitch given a prior pitch. An example of a first-order probability table is shown in Figure 2.

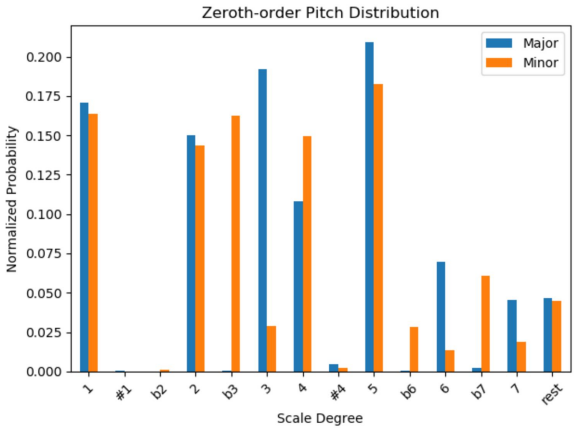

Figure 1: Zeroth-order pitch distributions, normalized by scale degree

The music generation takes weighted random selections based on the probability distributions from the tables, and adds them to the generated score. Several strange cases emerged based on the time signatures of the dataset. A significant amount of the data is in 4/1 (four whole notes per bar). These time signatures have created a higher probability of

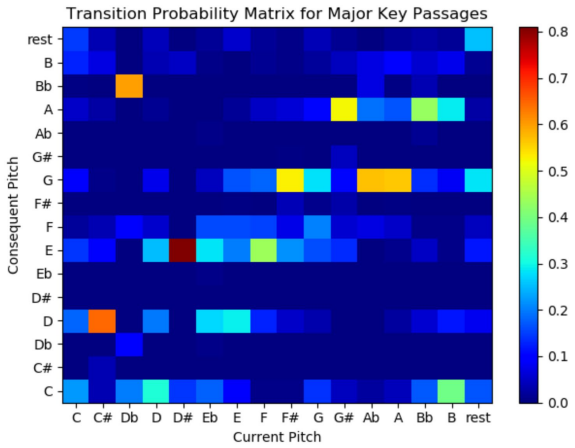

Figure 2: First order pitch probabilities. This figure plots how likely a pitch on the y-axis is to follow a pitch on the x-axis.

whole notes, breves, and longas than expected.

There were several challenges that needed to be overcome to complete this project successfully. There may be a benefit to using a normalized duration values based on the time signature. Normalizing the duration data would likely remove all chances of longas in the piece.

The documentation for music21 was a limiting factor, which presented in several complications. The hierarchical system of opuses and scores made it difficult to ensure that all the data in each file was being used properly. Issues in using music21's native hierarchical system, and developing a sophisticated understanding of the connection between music21 elements and their attributes and methods requires a large commitment and understanding of python.

3. Results and Discussion

Several variations of MelGen have been implemented with various degrees of success. Examples of each implementation are in Figure 3. The first implementation used only pitch class, without an octave, in a random walk approach. Random walk uses only the zeroth-order distribution in the data. Since all notes were generated within one octave, leaps from C to B are less natural as a major seventh instead of a minor second.

Another implementation selected a first note at random from the frequency in the dataset. Each subsequent pitch was selected from a probability distribution of pitch continuations, as a Markov chain. For example, if an A was selected, then the following note would be selected from conditional a probability of pitches that follow A. The downside to using only pitch information is that duration values are ignored. Therefore, in this implementation all duration values were quarter notes.

The next implementation uses the same pitch probability as the above version as well as rhythm determined from the zeroth-order duration probability distribution.

The current final implementation is based on a second order probability table based on current pitch and duration to a consequent pitch and duration. This implementation currently is adding a lot of syncopation to the generated melodies. To fix the high probability of syncopations, a rule based layer would need to check each beat for completion. In this situation, a quarter note could not be added on an off beat.

4. Future Work

Music21, while powerful, proved confusing to use. There are many intricacies of multi-order probability tables that have not been implemented yet.

As the goal of MelGen is to eventually merge it with Melodia, MelGen will need to be implemented in a browser. The most difficult component of a browser implementation is a good user interface, especially if MelGen is to be used for melodic dictation. A good user interface would most likely mimic a MIDI piano roll or simplified notation software.

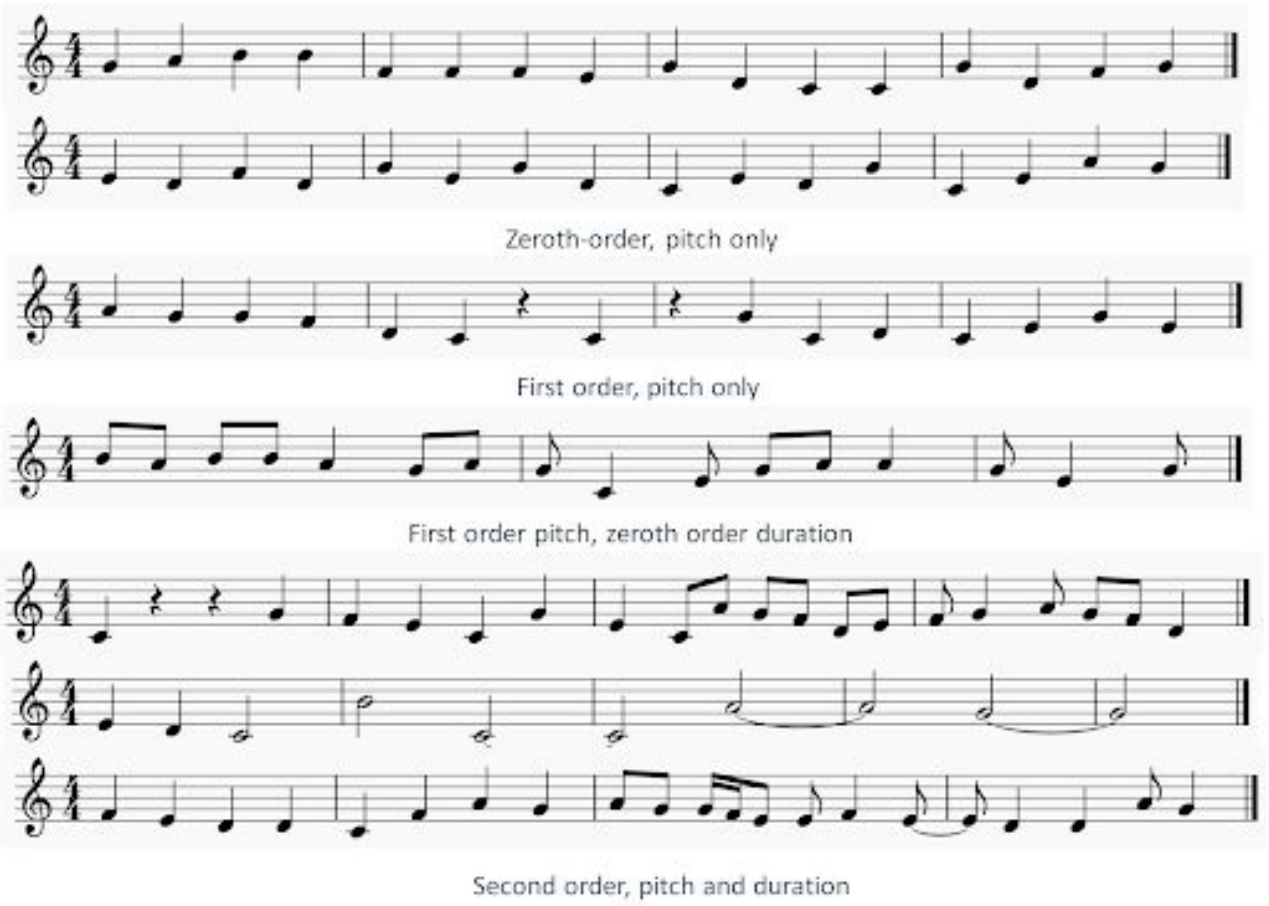

Figure 3: Multiple example outputs from the generator of different implementations

The browser implementation raises several questions. There is a trade off for real time generation and time for the page to load. Time required to generate examples may explain why websites like Teoria have chosen to load pre-generated examples on their website. The ideal case would only require the necessary tables to be loaded in the browser. Any manipulation of the tables and the generation could happen in browser.

References

Alvira, J.R. (2019). Teoria.

Conklin, D. Music Generation from Statistical Models. *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences.* Retrieved from URL.

Dong, H., et all. (2018). MuseGAN: Multi-Track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. *The 32nd AAAI Conference*. Retrieved from URL.

Kostka, S. M., Payne, D., & Almén, B. (2017). *Tonal harmony*. New York: McGraw-Hill.

Papadopoulos, A, Roy, P., and Pachet, F. (2014). Avoiding Plagiarism in Markov Sequence Generation. *Association for the Advancement of Artificial Intelligence.* Retrieved by URL.

Puckette, M. (2019). Max/MSP.

Taube, H. (2018). Harmonia.

Appendix

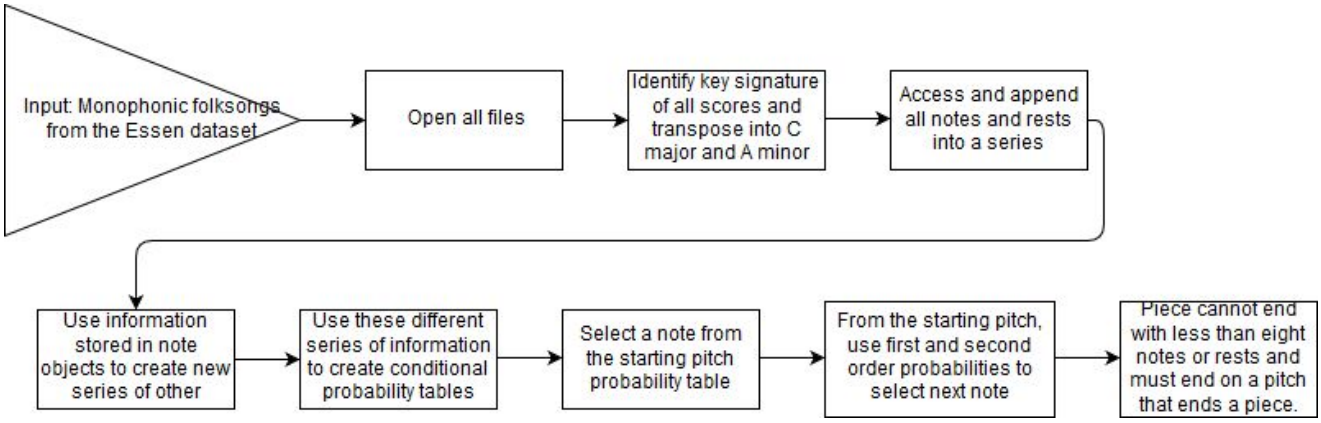

Figure 1: Flowchart of MelGen