FakeBach

Bhairav Chidambaram

Goal: Build a computer program capable of generating music

Process:

In order to generate music, I needed some sample music to feed into the program. For the samples I chose several inventions composed by JS Bach. These inventions were good candidates because they are melody focused, as there are very few chords. It is difficult to get a program to understand chords, unlike notes which can simply be described by pitch and duration.

In a general sense, the program would "understand" the music as a Markov matrix. In the abstract, a Markov matrix allows us to use a linear sequence to generate a new sequence. The algorithm would first build the Markov matrix by recording the transition probabilities from any element in the sequence to another. Note that the matrix is only useful if the original sequence has repeated elements (this is important for later).

After creating the matrix, the new sequence will be generated using the beginning of the old sequence as a seed. Based on the created Markov matrix, the next element of the sequence will be selected at random based on the transition probability calculated for this initial sequence.

Example:

Consider the input sequence "ABBCABAC"

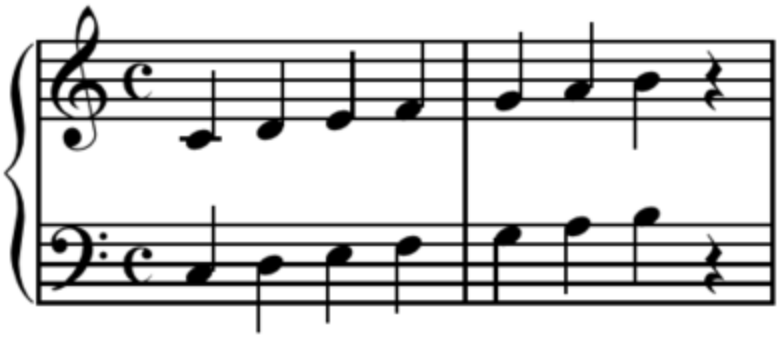
This gives us the matrix:

	A	B	C
A	0	2/3	1/3
B	1/3	1/3	1/3
C	1	0	0

Based on the initial sequence, the seed would be "A". We go to the first row corresponding to "A", and pick the next element based on the probabilities. Suppose we pick "B" as the second element (this would happen 2/3 of the time). Then we go to the second row corresponding to "B", and again pick a new element based on the shown probabilities. This process can be continued indefinitely, so in theory our program could generate music indefinitely (we are limited by program runtime and file size).

I wrote the code in the programming language Python 3, using the module Music21. This module allows Python to read MIDI files (sheet music) as a sequence of notes, rests, chord and miscellaneous symbols.

Example:



This midi file corresponding to this sheet music would be read by Music21 as the following:

song:

part1: [4/4] [C4, 0.25] [D4, 0.25] [E4, 0.25] [F4, 0.25] [G4, 0.25] [A5, 0.25] [B5, 0.25] [R, 0.25]

part2: [4/4] [C3, 0.25] [D3, 0.25] [E3, 0.25] [F3, 0.25] [G3, 0.25] [A4, 0.25] [B5, 0.25] [R, 0.25]

In addition, Music21 provides tools for transposing between keys, and analyzing key. These tools were needed because the Bach's Inventions are all written in different keys. Music21 is also useful because it converts a MIDI file, a complex file-format, into a list which is very easily manipulated using Python. By simply running a single function, Music21 can convert the list back into a MIDI file or into viewable sheet music in Music XML file-format.

Attempt 1:

The first attempt was to simply get the Markov matrix working and capable of generating music. Thus the input sequence in this first attempt was extremely simplified. Note duration was ignored, so this version simply interpreted music as series of pitches, each held for an eighth-note length.

The code for this version is saved in attempt1.py and is 78 lines in length. First the sequence of pitches is extracted from the sheet music and used to create a Markov matrix. Then this matrix is used to generate a new sequence of notes, each of which is assigned the length of an eighth note.

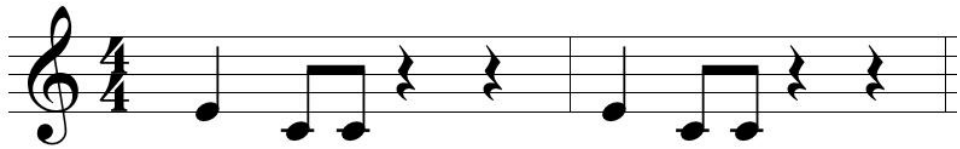
Due to the simplicity of this version, the generated sound file did not sound very musical. In addition, the Markov matrix was generated using only the right hand part of Invention No. 8, so the limited information restricted the *depth of output* (this idea will be clarified in the next attempt).

Attempt 2:

After a working version implementation of the Markov model was created, the next goal was to improve the musicality of the generated sequence. The most obvious fault in the previous version was the lack of rhythm. Thus, this version considers both the pitch and duration of notes, as well as rests. Like the last version, this one also only used the right hand part of Invention No. 8 to build its matrix.

In order to consider both pitch and duration, I needed to create two Markov matrices. The first was simply for pitch, and ignored the length of notes (similar to the matrix in the previous version). The second matrix considered the rhythm of each measure as a single unit.

Example:



The program interprets the first measure as the first unit

Unit 0: [Note 0.25] [Note 0.125] [Note 0.125] [Rest 0.25] [Rest 0.25]

Every such unit is assigned an index. Since the second measure has the same rhythm as the first, its corresponding index is also 0. The entire sheet music would have the sequence “00”

The code for this version is saved in attempt2.py, and is 129 lines long. The algorithm works by first building Markov matrices for both pitch and rhythm. Then the program picks the rhythm

sequence using the rhythm Markov matrix. Every time a note occurs, its pitch is decided by the next element in a sequence of pitches generated by the pitch Markov matrix.

Adding rhythm to the music made a noticeable difference in musicality. However, having only one part was not a true analog of Bach's inventions, which have two parts (left hand and right hand).

Both of the next two versions also require the file markov.py to run, which abstracts the Markov model using integer indices corresponding to each pitch, rhythmic unit, etc.

Attempt 3:

The goal of this version was to find a way to generate two parts that mix together well. To do this, my first idea was to use chords: in order to get the left and right hand parts to harmonize, their notes needed to be selected from the same sequence of chords.

Thus I assigned each measure a chord, either based on the first pitch in the measure or all pitches if there weren't too many. Instead of using a sequence of notes, this sequence of chords was used to generate the Markov matrix. In addition, a matrix was created for rhythm as in the last version.

Example:

Part: [F4, 0.25] [R, 0.75] | [D4, 0.25] [E4, 0.25] [F4, 0.25] [D4, 0.25]

The program interprets the first measure as "F4 A5 C5" based on the first note "F" of the first measure (this is an F Major chord in root position). The second chord is interpreted as "D4 E4 F4 " since there are only a few unique pitches in the second measure. Thus the chord sequence of this sheet music is ["F4 A5 C5", "D4 E4 F4"]

The code for this file is saved in attempt3.py, and is 105 lines long (not including markov.py). Right now the program is incomplete. After simply listening to the sequence of chords without any rhythm or multiple parts, I decided that it did not sound good enough to be worth finishing (i.e. adding rhythm and splitting the sequence of chords into parts).

The biggest reason for this was that reducing each measure to a single chord wasted a lot of useful information. While this information was captured by considering each note's pitch, reducing several consecutive pitches in to a single chord compromised the information of the order of these pitches.

Attempt 4 (final version):

The failure of the previous version showed that creating two parts that harmonize well is very difficult for a computer. In this version, I tried a different approach that viewed harmony less strictly.

In order to accommodate several inventions which have different time signatures (e.g. 3/4 time or 4/4 time), I instead used one quarter-note length as the unit of rhythm. Like the second attempt, this version creates Markov matrices for both pitch and rhythm.

However, this one has several rules that also allow it to generate the second hand part based on the first. While rhythm is completely independent between the two parts, the pitches of both parts are selected from the same sequence in a specific way.

Example:

Suppose we have generated two rhythm sequences

Rhythm 1: [Note 0.125] [Rest 0.125] | [Note 0.0625] [Rest 0.0625] [Note 0.125] | [Note 0.25]

Rhythm 2: [Rest 0.0625] [Note 0.0625] [Note 0.125] | [Note 0.25] | [Rest 0.125] [Note 0.125]

As well as a pitch sequence:

[A5] [F4] [C4] [D4] [E4] [G4]

As explained in the attempt 2, the pitches are first assigned normally to rhythm 1. For the second rhythm, each note is one octave below its corresponding note in rhythm 1. More precisely, two notes are corresponding if they are both the ith note in the jth measure for some i and j.

Based on these rules, we would obtain the following, where corresponding notes are color-coded. Note that since different measures have different numbers of notes, some notes are left unpaired. To fill in any extra notes in the left-hand part, we simply use an octave below the next pitch in the generated pitch sequence (this is the case for the green “F3” below).

Song:

part1: [A5, 0.125] [R, 0.125] || [F4, 0.0625] [R, 0.0625] [C4, 0.125] || [D4, 0.25]

part2: [R, 0.0625] [A4, 0.0625] [F3, 0.125] || [F3, 0.25] || [R, 0.125] [D3, 0.125]

The code for this version is saved in attempt4.py, and is about 143 lines long (not including markov.py).

In order to settle on these rules for choosing the second part, I had to go through a lot of trial and error. The conditions under which the two parts fit well together were difficult to determine, but ultimately produced a nice-sounding result.

Conclusion:

The final attempt sounded remarkably like Bach, up to a point. While it was able to generate interesting musical ideas and was able to sound “musical” on a local scale, there was very little long term structure to the music. This issue is inherent to the methodology used to generate the music, as Markov matrices only consider the previous few elements in a sequence in deciding the next.

In addition, it can be argued that the specific conditions I imposed on the last version were the creative steps needed to transform the Markov model into nice-sounding music. Regardless, the fact that a computer can use these conditions to generate music indefinitely suggests that a random note transitions are a good approximation music (at least for Bach).

Perhaps more complex approaches (such as machine learning) may allow a computer to actually understand what makes something music, and use this information to generate something truly creative.