The Application of Higher-Order Multivariate Markov Chains to Procedurally Generated Music

29 April 2016

Eric Perkey, Bryce Wilson

Section 002

Abstract

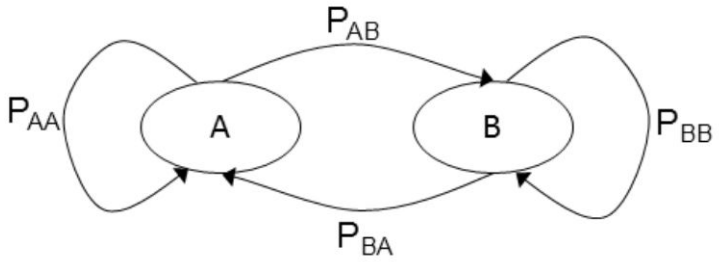
Markov chains are commonly used in modeling many practical systems such as queuing systems, manufacturing systems and inventory systems. If one can model the categorical data sequences accurately, then one can make good predictions and plan in a decision-making process. Markov chains are used most easily with sequences of data that rely on only themselves, such as the letters or words in a sentence, or in our case, notes in a musical piece.

Multivariate chains can be used to model more complicated sets of data. *Higher-order multivariate Markov chains and their applications* by Wai-Ki Ching, Michael K. Ng, and Eric S. Fung (2007) proposes a model for higher-order multivariate Markov chains, which can be used for modeling sequences of data that depend on one another as well as more than one previous state in each sequence. In this paper, we will explore the application of generating music. In a similar fashion to the imitation of Shakespeare using higher-order Markov chains, higher-order multivariate Markov chains could be used to imitate the works of famous composers.

Though difficult to analyze with music theory, the results sound notably improved from a series of random notes. Using a Markov chain as a foundation for procedurally generating music is therefore entirely viable.

Introduction

The simplest form of a Markov chain is one that translates between two states. With two states (defined as A and B) in the state space, there are 4 possible transitions. If it was at 'A' it could transition to 'B' or stay at 'A'. If it was at 'B' it could transition to 'A' or stay at 'B', as shown in Figure 1. In this two state situation, the probability of transitioning from any state to any other state is 0.5.



$$\mathbf{P}=\begin{bmatrix} p_{AA} & p_{AB} \\ p_{BA} & p_{BB} \end{bmatrix}$$

Figure 1: Diagram of two state Markov chain. Matrix P defined as probability matrix based on the states.

We will construct a frequency transition matrix, then by normalizing these matrices, leads us to a "probability matrix" to tally the transition probabilities. Every state in the state space is included once as a row and again as a column, and each cell in the matrix tells us the probability of transitioning from its row's state to its column's state. If the state space adds one state, we add one row and one column, adding one cell to every existing column and row. Higher-order models remember more “history;” additional history can have predictive value.

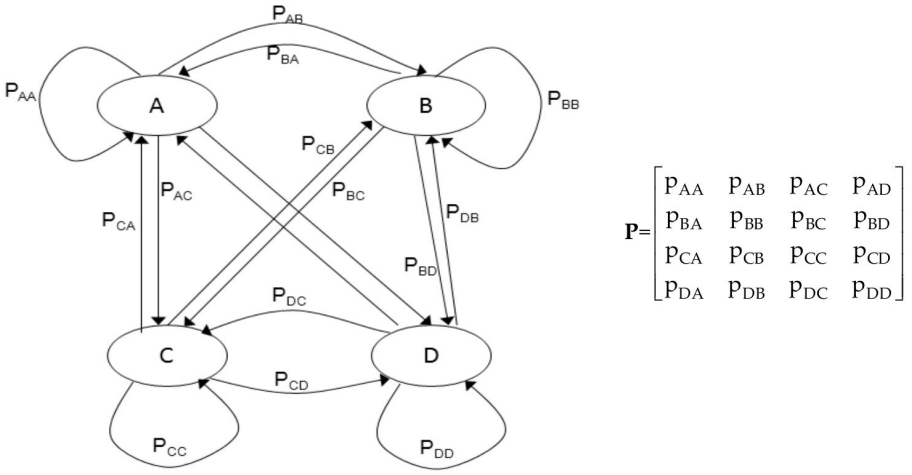


Figure 2: Similar to Figure 2, but with two additional states, creating a more complex Markov chain. Matrix P, again, is defined as the probability matrix for each state switch.

A higher-order Markov chain uses multiple graphs and probability matrices. As before, probability matrices are derived from frequency matrices, but now we keep track of states preceding the previous state. That is, instead of only the last state having an effect on the next state, the last n states have an effect on the next state. This leaves us with n probability matrices.

The next type of Markov chain is a multivariate Markov chain, which uses multiple sequences of data. The probability of a certain sequence going to a certain state depends not only on that sequence's last state, but also some number of other sequences' last states. Now our probability matrix must be made up of other probability matrices. In the same way we keep track of going from state A to state B in one sequence, we want to keep track of going from state M in sequence A to state N in sequence B. In order to keep our probabilities from multiplying, there must also be weights between the sequences. The higher the weight, the more of an effect the corresponding sequence has on the next state. These weights must sum to 1 for all effects on each sequence.

The next logical step is to move to higher-order multivariate Markov chains, which are discussed by Wai-Ki Ching, Michael K. Ng, and Eric S. Fung in their paper. Instead of using a matrix of lists of matrices, they propose a single matrix that will achieve the same effect when multiplied by a single column vector of the current state probability vectors. This model can look n steps back in s sequences.

We will use higher-order multivariate Markov chains to imitate the style of Beethoven's piano sonatas.

Mathematical Formulation

In order to keep the basic exploration simple, the most important notes of each eighth note beat in both hands of a piano piece were used. In addition, only pieces in C major in 4/4 time were considered. In that case, we could represent notes themselves as values rather than keeping track of intervals. Ignoring octaves, each named note was assigned a number, as was a rest, giving 13 states (no distinctions were made between enharmonically similar notes). We decided to choose n to be 16, or to let each note depend on the previous two measures in both hands. The sheet music that was available to us was translated to two sequences. Frequencies and probabilities were then calculated in Mathematica. We decided to set all weights to be equal, again for simplicity. This data was then combined in the format discussed by Ching, Ng, and Fung into the matrix Q. For the sake of space, we do not have an image of this matrix, though the code can be found in the appendices.

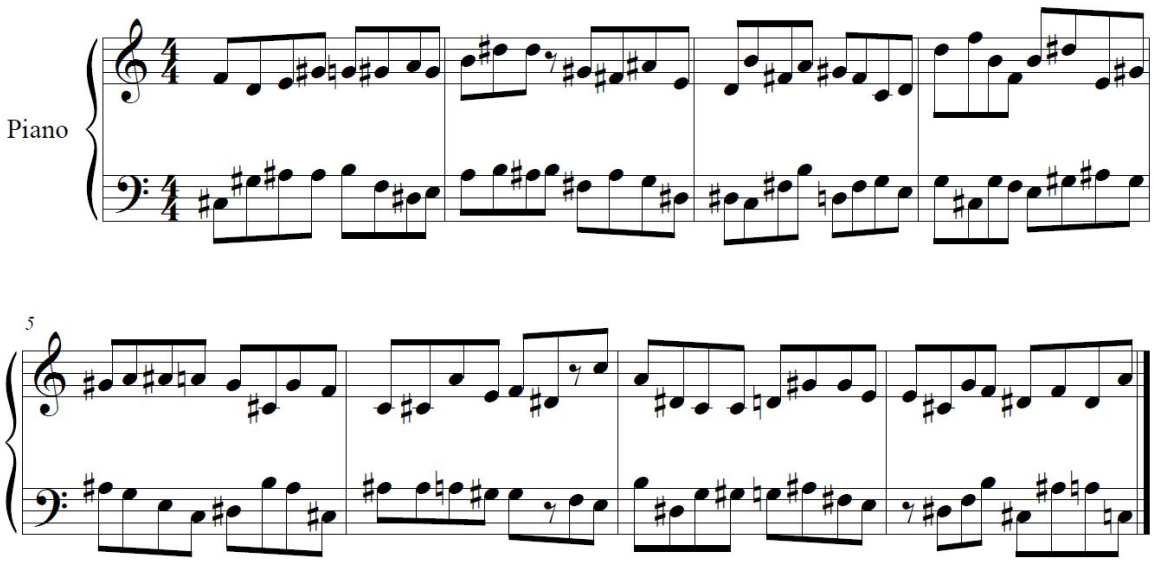
In order to generate a new sequence, we were required to begin with sequences of 16 notes with a probability of occurrence greater than 0. To do this, we simply chose the two measures that already existed in our input data. These states made up our initial probability vector X.

We can now find the probabilities of the next states by $QX^{(k)} = X^{(k+1)}$.

Examples and Numerical Results

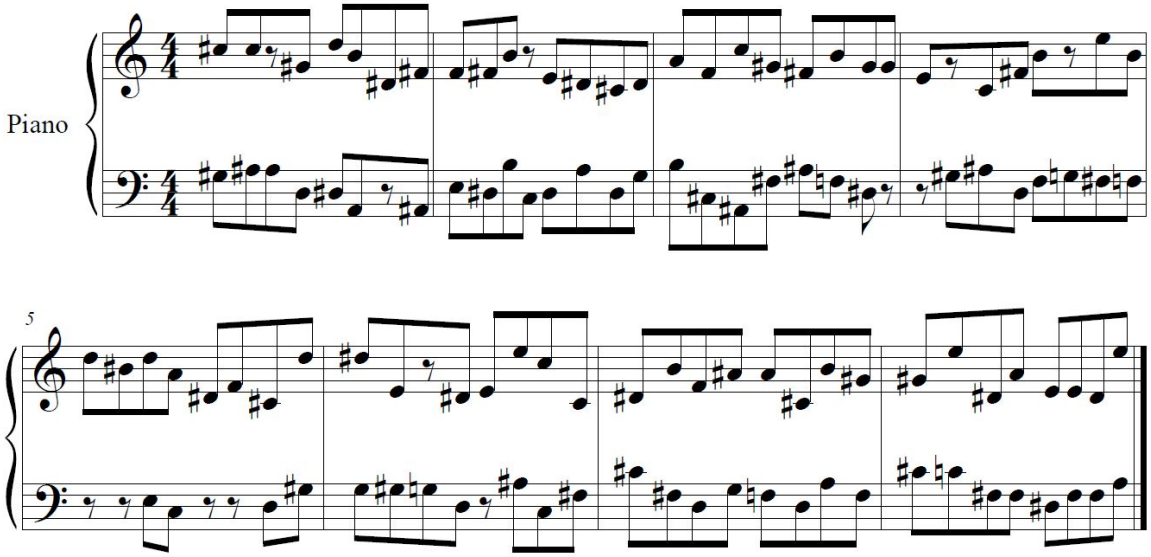
From $QX^{(0)} = X^{(1)}$ we find the probabilities of the next state in both sequences. Comparing a random number to our new vector allows us to choose a new state based on these probabilities. That new state is recorded in our output and replaces the probability vector we just found with an elementary basis vector for the corresponding state, since we are now certain of the state of the most recent note.

This process was repeated 64 times to give us 64 eighth notes, or 8 measures of music. The data was translated back into sheet music and audio files. Because music is subjective, we also created a couple sequences using only random numbers for a control. While listening, we recommend playing a randomized track before each output in order to make distinctions easier. All tests were produced with the same procedure, weights, steps back, input data, etc…



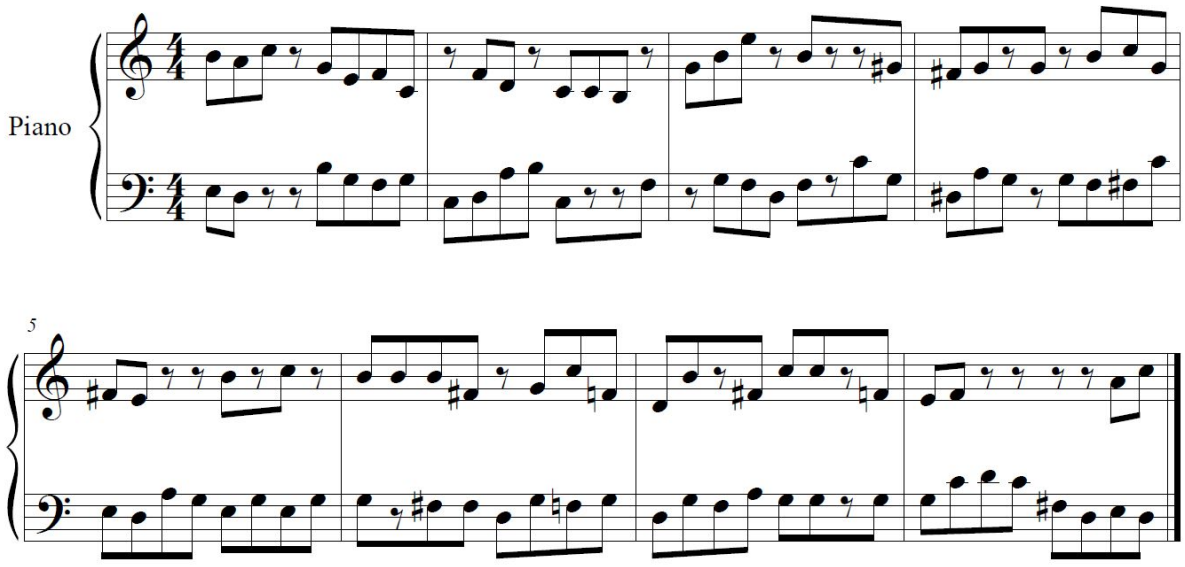
Random 1

http://tinyurl.com/jqrrxus



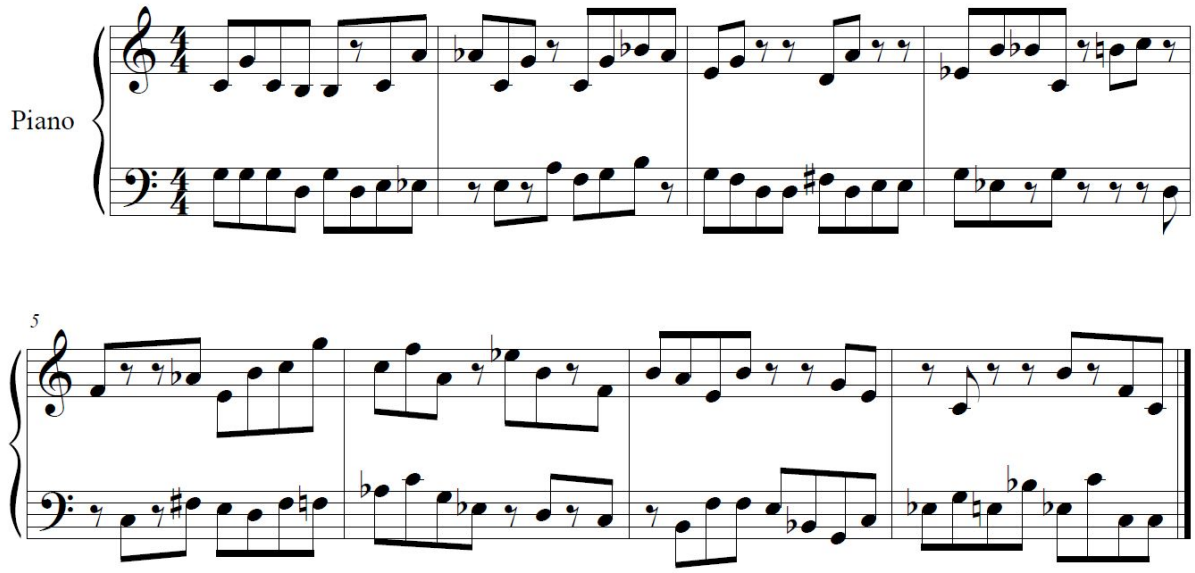
Random 2

http://tinyurl.com/jhph753



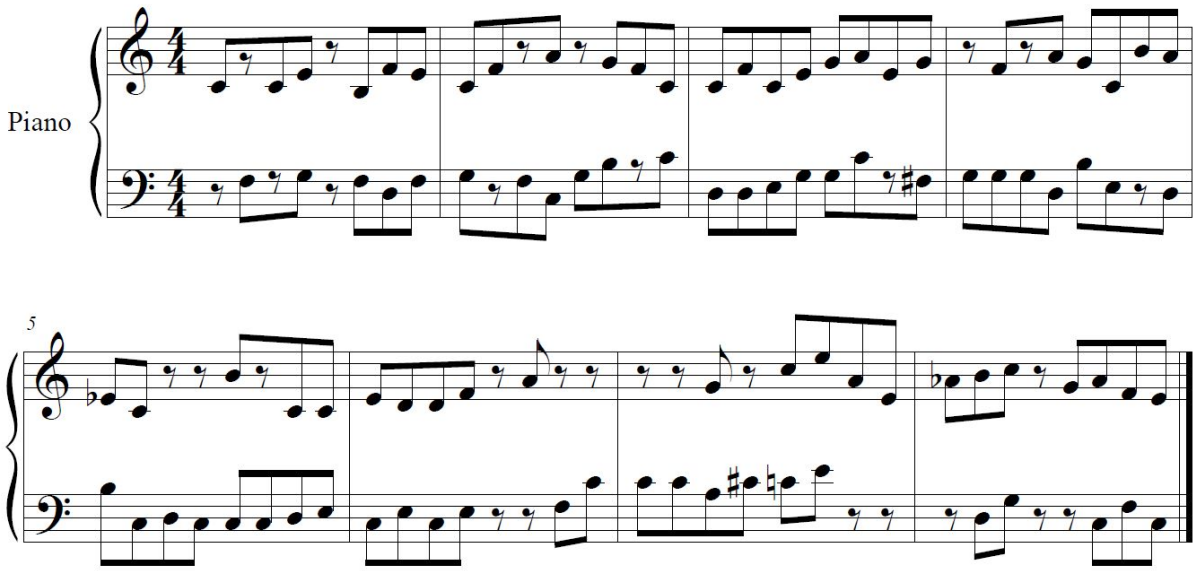
Test 1

http://tinyurl.com/z53sk4f



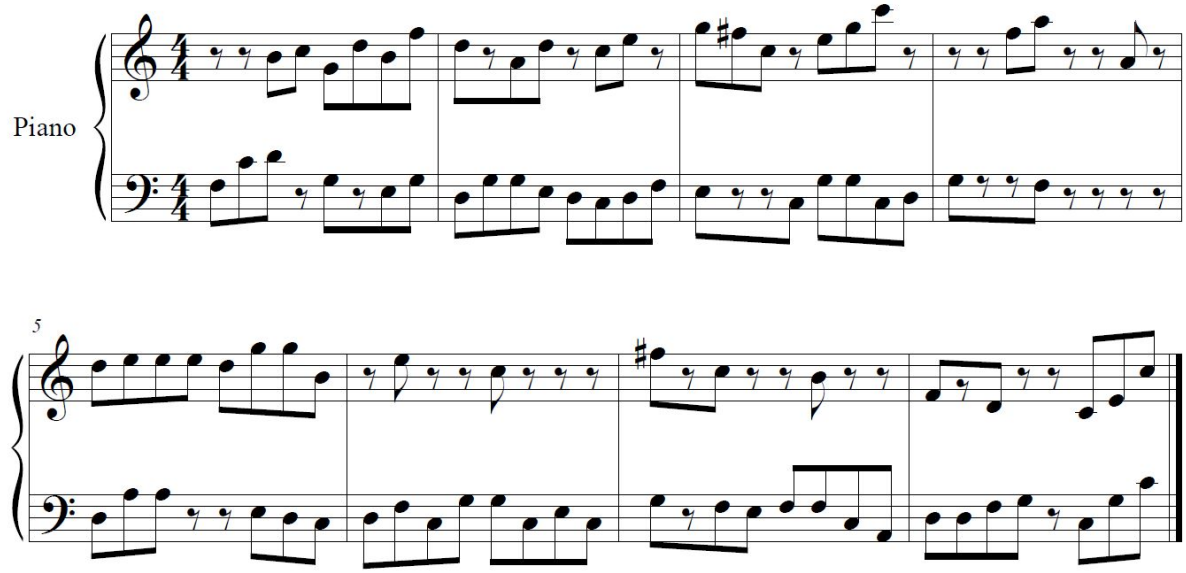
Test 2

http://tinyurl.com/j6mfevz



Test 3

http://tinyurl.com/zv724vw



Test 4

http://tinyurl.com/h8wouth

Figures titled *Random* are simply random notes generated in the same format as the program output by a random number generator. Their purpose is as a control or reference to the outputs of our program. Figures titled *Test* are all outputs of the Markov chain procedure. Each image is accompanied by a link to a midi file that will play the corresponding music. The enharmonic spelling of a note should be considered irrelevant. Because the computer is not attempting to follow the rules of music theory, there is currently no "correct" way to spell the note. Whether a note is written F# or Gb is arbitrary.

Conclusion

Because of the subjectivity of music, it is difficult to argue that our results convey progress without analyzing many outputs with music theory and statistics, but we believe there is a significant difference between the tests and the random notes. Considering these improvements were generated under heavy restriction, we believe it is very possible to use Markov chains to procedurally generate music. Concerning improvements to the program we already have, changing the weights λ could have a dramatic effect, though it would take research to determine which previous states should have the most weight. The position of a note in a measure has a significant effect on whether a note is a rest. We noticed that our program frequently had strange syncopations that existed because the program wasn't predicting based on the beat the note landed on. This could be done with multiple matrices Q, where each Q corresponds to a different position in a measure. Calculating Q is relatively inexpensive relative to generating sequences, so a list of Q's would not be extreme. Adding more attributes such as chords, accents, or other instruments as sequences could be possible by extension of a working algorithm, though that

would increase computation time greatly. And with a Markov chain procedure that works well, an algorithm could be designed around it that adds key changes, time changes, repeats, tempos, and other broader elements of music.

Our results could be a first step to a very complex program with a realistic running time (albeit large) that can imitate musical works. However, an algorithm that uses Markov chains cannot replicate creativity, and a program like this would likely not be used widely. Such a program could possibly be used as inspiration for a composer with writer's block, but it's difficult to tell so early in the technology.

References

Beethoven, Ludwig van. *International Music Score Library Project.* Last Modified 9 Feb 2016. Web. Accessed 27 April 2016.

Cruise, Brit. "Origin of Markov chains." Online video clip. *Khan Academy.* Khan Academy, 28 Apr. 2014. Web. 27 Apr. 2016.

Fung, Eric S, Ng, Michael K, and Ching, Wai-Kai. *Higher-order multivariate Markov chains and their applications*. Hong Kong: The University of Hong Kong, 2006. Web.

Powell, Victor. "Markov Chains - Explained Visually." *Explained Visually*. Setosa, n.d. Web. 27 Apr. 2016.

Appendix

Mathematica Code:

```
(*data is our variable for inputs. It is the reference on which we will build
our probability matrices. The first list is every "primary" eighth note beat
in the right hand, and the second list is the same in the left. "Primary" just
means the most important note played in that space, like the root of a
chord.*)
(*
A     = 1;
A#/Bb = 2;
B     = 3;
C     = 4;
C#/Db = 5;
D     = 6;
D#/Eb = 7;
E     = 8;
F     = 9;
F#/Gb = 10;
G     = 11;
G#/Ab = 12;
Rest  = 13;
*)
(*Most input data has been omitted for the sake of space*)
data = {{4, 11, 8, 4, 13, 13, 13, 13, 11, 8, 4, 11...},
   {13, 13, 13, 13, 4, 8, 11, 4, 13, 13, 13, 13...}};
(*s is the number of sequences we're working with (two in this case)
n is the number of steps back we want to look to base our next state on
states is the number of possible states either list can be in (12 notes plus a
rest)*)
s = Length[data];
n = 32;
states = 13;

(*F is all our frequency matrices (i.e. the number of times a change occurs).
P is all our probability matrices. F is an sxs matrix of lists of n mxm
matrices where m is the number of possible states. In order to find the matrix
that corresponds to the change from list 1 to list 2, 2 steps back, we want
F[[2,1]][[2]], which is a 3x3 matrix.*)
F = {};
For[i = 1, i <= s, ++i,
 F = Append[F, {}];
 For[j = 1, j <= s, ++j,
  F[[i]] = Append[F[[i]], {}];
```

```
  For[k = 1, k <= n, ++k,
   F[[i, j]] = Append[F[[i, j]], ConstantArray[0, {states, states}]];
   ]
  ]
 ]
For[i = 1, i <= s, ++i,
 For[j = 1, j <= s, ++j,
  For[k = 1, k <= n, ++k,
   For[l = k + 1, l <= Length[data[[i]]], ++l,
    ++F[[i, j]][[k]][[data[[i]][[l]], data[[j]][[l - k]]]];
    ]
   ]
  ]
 ]

(*P just divides each entry in F by its column sum.*)
P = {};
For[i = 1, i <= s, ++i,
 P = Append[P, {}];
 For[j = 1, j <= s, ++j,
  P[[i]] = Append[P[[i]], {}];
  For[k = 1, k <= n, ++k,
   P[[i, j]] = Append[P[[i, j]], ConstantArray[0, {states, states}]];
   ]
  ]
 ]
For[i = 1, i <= s, ++i,
 For[j = 1, j <= s, ++j,
  For[k = 1, k <= Length[F[[i, j]]], ++k,
   For[l = 1, l <= Length[F[[i, j]][[k]]], ++l,
    For[m = 1, m <= Length[F[[i, j]][[k]][[l]]], ++m,
     If[Total[Transpose[F[[i, j]][[k]]][[m]]] != 0,
       P[[i, j]][[k]][[l, m]] =
        F[[i, j]][[k]][[l, m]]/Total[Transpose[F[[i, j]][[k]]][[m]]];
     ]
    ]
   ]
  ]
 ]

(*λ is our weights. There is an individual weight for each combination of
sequence and steps back (s and n), so the sum of the weights across s and n
for each change of state must be 1. To simplify the program for now, all
weights are the same 1/ns*)
λ = {};
```

```
For[i = 1, i <= s, ++i,
 λ = Append[λ, {}];
 For[j = 1, j <= s, ++j,
  λ[[i]] = Append[λ[[i]], {}];
  For[k = 1, k <= n, ++k,
   λ[[i, j]] = Append[λ[[i, j]], 1/n/s];
   ]
  ]
 ]

(*In order to make computation a little more reasonable, we define a new
matrix Q whose values are matrices Subscript[B, ij] that contain P across the
possible states. Now, if we define the probability states similarly, we can
simply multiply the column vector of all state probability vectors by our
matrix Q to get the result.
This is the matrix proposed in the paper we based our project on.*)
Q = {};
For[i = 1, i <= s, ++i,
  Q = Append[Q, {}];
  For[j = 1, j <= s, ++j,
   B = {};
   For[k = 1, k <= n, ++k,
    B = Append[B, {}];
    For[l = 1, l <= n, ++l,
     If[i == j,
       If[k == 1,
        B[[k]] = Append[B[[k]], λ[[i, j]][[l]] P[[i, j]][[l]]],
        If[k == l + 1,
          B[[k]] = Append[B[[k]], IdentityMatrix[states]],
          B[[k]] = Append[B[[k]], ConstantArray[0, {states, states}]];
          ];
        ],
       If[k == 1,
         B[[k]] =
          Append[B[[k]], λ[[i, j]][[l]] P[[i, j]][[l]]],
         B[[k]] = Append[B[[k]], ConstantArray[0, {states, staλλλtes}]]
         ];
       ];
     ];
    ];
   Q[[i]] = Append[Q[[i]], B]
   ];
  ];
```

```
(*This section is a test for a single iteration of our matrix
multiplication.*)
(*In order for the program to function correctly, we need to start with a list
of n previous states that could possibly exist in our original data. In order
to ensure this happens, we just choose 16 sequential states from somewhere in
our original data. The "start" variable stores the location of the first item
in this sequence. We want to start our data at the beginning of a measure, so
we move to the previous multiple of 8 after we pick a random number.*)
start = Floor[RandomInteger[{1, Length[data[[1]]] - (n + 1)}]/8]*8;
(*X is our column vector*)
X = {};
For[i = 1, i <= s, ++i,
  X = Append[X, {}];
  For[j = 1, j <= n, ++j,
   X[[i]] = Append[X[[i]], ConstantArray[0, states]];
   X[[i]][[j]][[data[[i]][[start + j]]]] = 1;
   ];
  ];

(*Y is an empty matrix that we will use to fill the solution as we calculate
it, instead of replacing values in X before we use them*)
Y = {};
For[i = 1, i <= s, ++i,
  Y = Append[Y, {}];
  For[j = 1, j <= n, ++j,
   Y[[i]] = Append[Y[[i]], ConstantArray[0, states]];
   ];
  ];

(*This is the actual product of Q and X. Mathematica can't do a matrix product
with embedded matrices, but rewriting Q and X would be a major formatting
inconvenience.*)
For[i = 1, i <= Length[X], ++i,
  For[j = 1, j <= Length[X], ++j,
    For[k = 1, k <= Length[X[[j]]], ++k,
      For[l = 1, l <= Length[Q[[i, j]][[k]]], ++l,
        Y[[i]][[k]] += Q[[i, j]][[k, l]].X[[j]][[l]];
        ];
      ];
    ];
  ];
X = Y;
```

```
(*Printing X. The first entry of each list is the new probability state
vector. We want the sum of the vector to be equal to 1. If it's not, something
has gone wrong.*)
MatrixForm[N[X]]
Total[X[[1]][[1]]]
Total[X[[2]][[1]]]
```

```
(*In this section, we will do the actual generation of a new list of numbers
that can be translated back to notes. These sequences are stored in "output."
All other variables are the same as before.*)
X = {};
output = {};
start = Floor[RandomInteger[{1, Length[data[[1]]] - (n + 1)}]/8]*8;
For[i = 1, i <= s, ++i,
  X = Append[X, {}];
  output = Append[output, {}];
  For[j = 1, j <= n, ++j,
   X[[i]] = Append[X[[i]], ConstantArray[0, states]];
   X[[i]][[j]][[data[[i]][[start + j - 1]]]] = 1;
   ];
  ];

(*This outer loop determines the number of times we create the next state for
each sequence, or the number of eighth notes we want to make.*)
For[t = 1, t <= 64, ++t,
  Y = {};
  For[i = 1, i <= s, ++i,
   Y = Append[Y, {}];
   For[j = 1, j <= n, ++j,
    Y[[i]] = Append[Y[[i]], ConstantArray[0, states]];
    ];
   ];

(*Here is the product of Q and X again*)
  For[i = 1, i <= Length[X], ++i,
   For[j = 1, j <= Length[X], ++j,
     For[k = 1, k <= Length[X[[j]]], ++k,
       For[l = 1, l <= Length[Q[[i, j]][[k]]], ++l,
         Y[[i]][[k]] += Q[[i, j]][[k, l]].X[[j]][[l]];
         ];
       ];
     ];
   ];
  X = Y;
```

```
(*Here we use the probability state vector to choose the next state. "a" is a
random number between 0 and 1, and as soon as it is smaller than the sum of
the first j elements of the probability state vector, we say that we've found
our new state.
  This state is stored in "output" and replaces the first element of the
sequence in X, and we move to the next sequence.
  Once all states have been determined, we start a new multiplication.*)
  For[i = 1, i <= s, ++i,
   a = RandomReal[];
   For[j = 1, j <= Length[X[[1]][[i]]], ++j,
    If[a <= Sum[X[[i]][[1]][[k]], {k, j}],
      Break[];
      ];
    ];
   output[[i]] = Append[output[[i]], j];
   X[[i]][[1]] = ConstantArray[0, states];
   X[[i]][[1]][[j]] = 1;
   ];
  ];
Column[output]
```

```
(*This section just generates random data for input for the purpose of testing
based on number of sequences, number of states, length of input, etc...
It was also used to generate the random notes for reference.*)
random = {};
For[i = 1, i <= 2, ++i,
 random = Append[random, {}];
 For[j = 1, j <= 64, ++j,
  random[[i]] = Append[random[[i]], RandomInteger[{1, 13}]];
  ]
 ]
Column[random]
```