MATH 2270 Linear Algebra - Project Report

Using Markov Chain to Compose Music Pieces

Prathusha Boppana, Jie Zhang

Introduction

Markov chains contain the probability of transferring from one state to the next possible state in a sequence of events. When Markov chains are used in learning algorithms, it usually is the abstraction of the probabilistic data which can be used to infer how the next steps would be from the previous steps that just went through. Music composition is an interesting subject that can have Markov chains applied relatively easily as a music piece can be easily seen as a sequence of states, with each state as a note, with its specific length played. Since the notes available are not infinite, the length options are not infinite either, even adding in the probability of multiple instruments, the categories of state should also be finite. Markov chains thus can be built from previous musical pieces of different genre and be the basis for learning algorithm to make probabilistic decisions and create new music pieces in the same genre.

Different orders of Markov chains would vary in terms of the level of accuracy in capturing the probability of transferring from one state to the next possible state. A first order chain means only to count the previous one state and example the probability transition from the first to the second. An example of a first-order chain with states of the system such as note or pitch values, and a probability vector for each note constructed, completing a transition probability matrix is shown in Table 1. An algorithm would need to be constructed to produce output note values based on the transition matrix weightings from Table 1.

Table 1. First-order Matrix

1st-order matrix			
Note	A	C♯	E♭
A	0.1	0.6	0.3
C♯	0.25	0.05	0.7
E♭	0.7	0.3	0

(Source: Wikipedia, URL: https://en.wikipedia.org/wiki/Markov_chain#Music)

A second-order Markov chain can be used by creating a table (see Table 2) with the current state and the previous state, similar to Table 1. Higher, *n*th-order Markov chains tend to "group" particular notes together, while creating various patterns and sequences. These higher-order chains offer sequences of notes with a sense of phrasal structure, rather than the random sequences produced by a first-order system.

Table 2. Second-order Matrix

2nd-order matrix			
Notes	A	D	G
AA	0.18	0.6	0.22
AD	0.5	0.5	0
AG	0.15	0.75	0.1
DD	0	0	1
DA	0.25	0	0.75
DG	0.9	0.1	0
GG	0.4	0.4	0.2
GA	0.5	0.25	0.25
GD	1	0	0

(Source: Wikipedia, URL: https://en.wikipedia.org/wiki/Markov_chain#Music)

Experiment Steps

1. Collect digital music pieces (Midi files) which can be processed easily with Java libraries to extract note/pitch info.
2. Using self-developed Java code (below), extract note info and save data in matrix data structure.
3. Examine each probabilistic vector to see if the data reflect reality.
4. Set up starting note for auto composing using probability vectors.
5. Examine the results from the auto-composing program.

Code

```
package midiConverter;
import java.io.File;
import java.util.ArrayList;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.Sequence;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Track;

public class converter {
        public static final int NOTE_ON = 0x90;
        public static final int NOTE_OFF = 0x80;
        public static int[][] notes = new int[12][12];
        public static int[][] Composednotes = new int[12][12];
```

```
            public static int prevNote1 = 0;
            public static int prevNote2 = 0;
            public static int currentNote = 0;
            public static int[] totalforEachRow = new int[12];
            public static int runningtotal=0;
            static ArrayList<Integer> musicpiece = new ArrayList<Integer>();
            static ArrayList<Integer> allsamples = new ArrayList<Integer>();
            public static int[][] notes2 = new int[144][12];
            static ArrayList<Integer> order2musicpieces = new ArrayList<Integer>();

             public static final String[] NOTE_NAMES = {"C", "C#", "D", "D#", "E", "F", "F#", "G",
"G#", "A", "A#", "B"};
            public static void main(String[] args) throws Exception {
                  String[]files    =    {    "bach_846.mid","ai_cho_em_tinh_yeu.mid","ai_biet.mid"
,"999_doa_hong.mid","1-2-3_ngoi_sao.mid",
"238878.mid","bjsbmm01.mid","988-aria.mid","988-v01.mid","988-v02.mid",
"988-v03.mid","988-v04.mid","988-v05.mid","988-v06.mid","988-v07.mid","988-v08.mid","988-
v09.mid"};
                  for (int i=0; i<files.length; i++){
                        Sequence sequence = MidiSystem.getSequence(new File(files[i]));
                        convertMidiToArray(sequence);
                  }
                  buildSecondOrderMarkov(allsamples);

                  for(int i = 0; i<notes2.length; i++){
                  for(int j = 0; j<notes2[i].length; j++){
                        System.out.print(notes2[i][j] +"," + " ");
                  }
                  System.out.println("");
               }
//                composerMethod();
                  composerMethod2();
             }

            public static void convertMidiToArray(Sequence s){
//                int trackNumber = 0;
               for (Track track :  s.getTracks()) {
//                trackNumber++;
//                System.out.println("Track " + trackNumber + ": size = " + track.size());
//                System.out.println();
                  for (int i=0; i < track.size(); i++) {
                     MidiEvent event = track.get(i);
```

```
//                      System.out.print("@" + event.getTick() + " ");
                        MidiMessage message = event.getMessage();
                        if (message instanceof ShortMessage) {
                            ShortMessage sm = (ShortMessage) message;
//                          System.out.print("Channel: " + sm.getChannel() + " ");
                            if (sm.getCommand() == NOTE_ON) {
                                int key = sm.getData1();
//                              int octave = (key / 12)-1;
                                int note = key % 12;

                                allsamples.add(note);
//                              notes[prevNote1][note] += 1;
//                              prevNote1 = note;
//
//                              String noteName = NOTE_NAMES[note];
//                              int velocity = sm.getData2();
//                               System.out.println("Note on, " + noteName + octave + " key=" + key + "
velocity: " + velocity);
                            } else if (sm.getCommand() == NOTE_OFF) {
//                              int key = sm.getData1();
//                              int octave = (key / 12)-1;
//                              int note = key % 12;
//                              String noteName = NOTE_NAMES[note];
//                              int velocity = sm.getData2();
//                               System.out.println("Note off, " + noteName + octave + " key=" + key + "
velocity: " + velocity);
                            }
                        } else {
//                          System.out.println("Other message: " + message.getClass());
                        }
                    }
                }
            }

            public static void buildSecondOrderMarkov(ArrayList<Integer> samples){
                for (int i=2; i<samples.size(); i++){

                                prevNote2 = samples.get(i-2);
                                prevNote1 = samples.get(i-1);
                                currentNote = samples.get(i);
                                notes2[prevNote2*12+prevNote1][currentNote]+= 1;
        }
```

```
	}

	public static void generateProbability(){
		for(int i = 0; i<notes.length; i++){
		for(int j = 0; j<notes[i].length; j++){
			runningtotal+=notes[i][j];
		}
		totalforEachRow[i]=runningtotal;
		runningtotal=0;
		}
	}

	public static void composerMethod(){
		int currentNode =composerMethod(1);
		musicpiece.add(currentNode);
		for (int i = 0; i <100; i++){
			currentNode = composerMethod(currentNode);
			musicpiece.add(currentNode);
		}
		System.out.print(musicpiece.toString());

	}
	public static int composerMethod(int startNote){
		int largestPossbileNote = notes[startNote][0];
		int indexOfLPN = 0;
		for (int i=0; i<notes[startNote].length; i++){
			if (startNote!=i){
			if (notes[startNote][i]>largestPossbileNote){
				largestPossbileNote = notes[startNote][i];
				indexOfLPN = i;
			}}
		}
		return indexOfLPN;
	}

	public static void composerMethod2(){
		order2musicpieces.add(0);
		order2musicpieces.add(1);
		int currentNode=0;
		for (int i = 2; i <100; i++){
			currentNode=composerMethod2(order2musicpieces.get(i-2),
order2musicpieces.get(i-1));
```

```
                        order2musicpieces.add(currentNode);
                }
                System.out.print(order2musicpieces.toString());

        }
        public static int composerMethod2(int firstStartNote, int SecondStartNote){
                int largestPossbileNote = notes2[firstStartNote*12+SecondStartNote][0];
                int indexOfLPN = 0;
                for (int i=0; i<notes2[firstStartNote*12+SecondStartNote].length; i++){
                        if (firstStartNote*12+SecondStartNote!=i){
                        if (notes2[firstStartNote*12+SecondStartNote][i]>largestPossbileNote){
                                largestPossbileNote=
notes2[firstStartNote*12+SecondStartNote][i];
                                indexOfLPN = i;
                        }}
                }
                return indexOfLPN;
        }
}
```

Data

For First-order Markov chain:

1st Order	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
C	1643	77	420	70	514	116	606	232	110	554	299	649
C#	86	1750	408	156	377	152	608	157	298	277	117	700
D	589	545	2739	26	607	156	866	581	125	425	419	665
D#	80	141	21	588	293	74	263	78	127	158	58	185
E	429	372	933	308	3203	139	1065	741	346	591	300	850
F	120	138	137	58	207	547	263	96	113	188	88	151
F#	554	386	813	340	1275	266	4199	931	396	1058	823	1474
G	304	185	374	82	703	97	1200	2732	130	633	406	820

G#	124	394	130	94	333	153	352	131	865	316	165	345
A	603	288	563	131	449	166	976	846	386	2954	366	1084
A#	253	101	446	45	322	96	831	394	154	398	992	544
B	504	709	759	168	994	144	1286	748	352	1260	543	4083

[11, 6, 11]

For Second-order Markov chain:

2nd Order	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
CC	157	32	235	22	190	33	140	66	18	210	51	489
CC#	5	31	0	1	5	3	5	1	8	10	1	7
CD	27	1	237	3	17	15	42	13	4	24	15	22
CD#	13	1	6	17	4	0	8	4	5	2	8	2
CE	60	9	36	1	189	8	56	31	12	67	25	20
CF	15	2	11	2	8	32	12	0	3	28	1	2
CF#	99	5	58	16	55	10	143	32	21	62	79	26
CG	28	2	7	8	30	2	27	72	1	14	21	19
CG#	13	15	3	6	13	4	11	3	16	6	9	11
CA	108	7	33	3	48	27	80	20	13	181	17	17
CA#	40	2	24	6	22	5	68	21	5	28	57	21
CB	26	4	25	8	22	5	20	19	9	12	15	484
C#C	25	5	0	5	9	4	5	2	14	7	3	7
C#C#	32	239	355	60	122	55	277	49	64	94	47	356

C#D	0	7	352	1	3	1	10	0	3	4	1	26
C#D#	1	17	0	62	8	12	18	4	18	9	1	6
C#E	6	43	14	17	126	10	35	32	23	28	4	39
C#F	2	17	7	10	11	37	11	7	22	5	2	21
C#F#	5	40	16	8	44	13	334	10	25	38	10	65
C#G	1	13	2	0	23	1	21	54	11	3	4	24
C#G#	10	36	1	14	34	30	25	17	66	13	8	44
C#A	7	42	4	9	31	7	26	3	11	111	1	25
C#A#	1	12	0	0	15	0	9	6	5	1	52	16
C#B	13	38	21	6	37	17	63	13	34	27	13	418
DC	412	0	28	2	26	8	34	14	1	26	19	19
DC#	2	444	8	0	10	4	23	6	2	26	7	13
DD	408	432	311	2	403	62	252	369	50	134	75	241
DD#	1	0	1	7	0	0	2	5	3	1	3	3
DE	16	10	31	0	402	6	44	21	13	13	28	23
DF	7	0	15	1	8	53	26	3	1	21	10	11
DF#	42	12	124	2	59	25	267	50	11	70	90	114
DG	10	0	38	1	17	3	38	375	3	15	36	45
DG#	3	2	11	1	11	1	7	1	66	7	11	4
DA	29	3	42	0	22	22	56	21	3	150	43	34
DA#	15	4	81	5	28	7	85	27	8	41	83	35
DB	25	15	71	1	36	6	94	43	12	30	44	288
D#C	21	3	2	4	3	4	18	4	6	2	9	4
D#C#	1	44	1	15	14	12	13	7	14	12	2	6
D#D	8	0	4	1	0	0	4	1	0	0	3	0

D#D#	21	41	7	60	215	20	64	16	29	54	8	53
D#E	0	9	0	12	212	3	11	16	12	5	3	10
D#F	2	13	0	1	6	20	13	0	9	3	2	5
D#F#	10	11	2	40	16	5	82	9	9	36	11	32
D#G	6	5	2	7	10	4	8	10	10	2	8	6
D#G#	2	21	0	7	20	10	17	16	22	3	0	9
D#A	4	9	3	18	6	2	27	2	14	52	1	20
D#A#	3	0	4	8	3	4	13	10	1	1	9	2
D#B	3	17	3	18	15	4	36	5	8	15	2	59
EC	142	3	27	2	40	11	66	24	13	58	29	14
EC#	10	101	11	12	54	17	25	28	41	27	7	39
ED	26	23	672	3	30	6	67	18	4	22	25	37
ED#	2	18	1	187	15	5	20	16	18	2	4	20
EE	138	122	679	192	421	61	581	314	106	257	69	263
EF	11	13	4	0	18	47	10	5	6	11	4	10
EF#	51	28	47	18	76	6	621	46	13	55	37	67
EG	31	30	15	9	117	7	58	267	46	21	25	115
EG#	16	31	19	13	70	4	24	27	64	8	2	68
EA	55	29	21	4	47	17	68	18	11	272	25	24
EA#	26	6	21	1	33	4	48	19	9	24	74	35
EB	27	40	28	13	180	13	61	125	59	42	31	231
FC	18	5	7	1	7	20	16	1	6	22	13	4
FC#	3	51	1	12	13	4	12	3	19	9	0	11
FD	12	0	47	1	3	15	16	3	2	18	16	4

FD#	1	7	0	20	2	1	5	4	10	2	3	3
FE	9	14	6	7	113	14	4	7	4	12	4	13
FF	10	33	45	15	100	39	115	54	34	41	13	48
FF#	23	12	6	14	6	20	118	6	6	20	22	10
FG	1	7	1	4	8	2	4	56	3	5	1	4
FG#	5	21	0	11	7	7	5	4	31	11	3	8
FA	30	7	10	8	15	27	23	3	2	36	12	15
FA#	7	0	10	0	6	8	23	0	9	9	12	4
FB	2	14	2	6	9	13	17	5	14	11	14	44
F#C	111	6	41	10	52	10	108	33	12	84	55	32
F#C#	6	104	5	9	21	5	87	15	19	31	7	77
F#D	31	34	239	1	36	20	159	33	7	78	73	102
F#D#	19	19	3	114	13	8	55	10	11	36	13	39
F#E	42	33	44	21	757	6	124	53	28	63	34	70
F#F	18	20	20	9	9	119	12	4	9	19	15	12
F#F#	115	145	271	123	769	120	833	564	186	378	113	582
F#G	24	13	33	16	39	2	76	553	11	26	77	61
F#G#	21	17	18	13	21	24	29	12	173	15	19	34
F#A	86	31	77	32	61	10	175	44	34	329	71	108
F#A#	53	8	103	7	50	23	113	67	23	79	198	99
F#B	24	63	136	29	86	6	301	56	32	106	79	556
GC	137	4	6	6	32	2	24	33	5	14	25	16
GC#	1	84	0	9	19	5	7	9	23	2	7	19
GD	12	4	166	5	11	2	25	40	3	21	39	46
GD#	5	4	0	19	10	1	9	7	2	8	9	8

GE	28	26	32	15	300	4	40	68	31	12	33	114
GF	3	5	0	2	5	63	4	5	0	4	2	4
GF#	37	14	43	15	42	4	810	51	7	31	77	69
GG	137	70	176	12	242	60	813	342	13	461	63	342
GG#	2	19	3	8	30	1	11	15	8	5	9	19
GA	17	5	21	2	12	4	28	25	5	461	27	26
GA#	20	4	39	3	36	5	87	48	6	32	79	47
GB	20	16	54	4	145	4	66	82	15	35	39	340
G#C	25	8	3	6	24	3	23	3	13	6	2	8
G#C#	9	148	2	23	32	19	33	11	39	14	6	58
G#D	0	9	41	0	9	4	6	1	31	7	11	11
G#D#	2	21	0	14	7	11	9	3	10	7	0	10
G#E	13	23	11	18	77	5	20	34	39	10	10	73
G#F	8	19	1	11	10	48	17	5	17	4	4	9
G#F#	18	14	10	9	16	9	165	8	18	24	14	47
G#G	5	9	7	6	37	5	5	12	5	8	8	24
G#G#	29	142	50	7	48	46	141	8	84	206	42	62
G#A	5	13	5	6	9	5	27	7	8	209	12	10
G#A#	10	5	9	0	5	5	17	19	8	18	54	15
G#B	6	42	11	8	70	12	43	14	21	24	24	70
AC	231	8	36	5	71	12	88	19	7	72	33	21
AC#	6	89	14	8	41	9	34	8	19	29	1	30
AD	19	5	293	2	14	12	63	32	4	48	37	34

AD#	5	8	0	36	6	2	29	3	9	8	1	24
AE	67	36	19	6	118	10	41	9	13	63	27	40
AF	34	5	13	1	16	28	21	2	5	18	13	10
AF#	72	31	66	46	68	17	312	26	36	123	88	91
AG	19	8	23	4	26	5	39	614	5	32	29	42
AG#	9	30	10	7	19	6	23	2	225	13	20	22
AA	217	112	284	31	148	23	289	642	247	239	57	665
AA#	26	3	33	0	20	18	64	30	18	59	72	23
AB	19	23	44	14	20	12	94	33	24	76	35	690
A#C	46	0	20	5	31	7	43	16	5	26	43	11
A#C#	3	24	7	0	4	3	12	7	9	2	18	12
A#D	22	4	83	5	29	12	94	26	11	36	81	43
A#D#	6	0	3	8	2	2	11	2	2	4	5	0
A#E	30	5	20	6	65	5	44	28	7	30	47	35
A#F	5	1	13	2	6	13	8	6	1	15	20	6
A#F#	54	6	78	13	43	17	155	76	20	86	193	90
A#G	25	8	32	8	28	3	62	67	9	32	75	45
A#G#	7	9	10	1	7	3	20	12	30	14	19	22
A#A	31	2	35	4	19	15	68	38	11	74	65	36
A#A#	42	34	80	14	74	12	216	109	44	66	112	189
A#B	12	8	45	7	27	0	86	43	23	41	73	179
BC	318	3	15	2	29	2	41	16	10	27	17	24
BC#	8	391	4	7	42	16	80	13	41	21	14	72
BD	24	26	294	2	52	7	128	45	6	33	43	99

BD#	4	5	0	44	11	12	33	4	10	25	3	17
BE	20	42	41	13	423	7	65	128	58	31	16	150
BF	5	10	8	4	10	48	14	5	6	19	2	13
BF#	28	68	92	36	81	20	359	53	44	135	89	281
BG	17	20	38	7	126	3	49	309	13	14	59	93
BG#	7	51	5	6	53	17	39	14	80	15	23	42
BA	14	28	28	14	31	7	109	23	27	840	35	104
BA#	10	23	42	1	30	5	88	38	18	40	190	58
BB	327	429	319	54	347	52	405	310	101	841	174	724

[0, 1, 9, 9, 11, 11, 9, 9]

Conclusions

Using Midi files of music pieces by Bach and other composers, we compiled a matrix of all of the occurrences of one note after another. We set a starting note for the auto-composing program to take to index into matrix, and let it run from there. After analyzing the data from our first order calculations, we realized that it was most probable for a note to come after itself. We then forced the program to ignore the probability of a note coming after itself and find the next highest probability, but that just resulted in a pattern of two notes alternating.

When we experimented with the second order Markov chain, we set the first two notes to initialize the parameters for the auto-composing program, allowing it to run. We found that there was a tendency of notes alternating with each note repeating twice.

From these findings, we concluded that Markov chain can realistically record the probabilities of note sequences. At the same time, higher order Markov chains and a better decision-making algorithm are needed to compose a music piece with a more precise and realistic variation of notes.