MARKOV CHAINS

& Computer Generated Music

NICE VIBES

Content

Introduction: History of Computer Generated Music

Creating Markov chain

Generating Music/ Understanding Results

History: Music Dice Games

In the 18th, people played Musikalisches Würfelspiel (Musical Dice Games), They would have tables where each square correlated to a prewritten measure of music, and then would roll 2 6-sided dice to determine the first measure, and then again for the second, generating a short piece of music.

Table pour un Menuet avec un Dé.

Premiere Partie.	1	2	3	4	5	6	Seconde Partie.	1	2	3	4	5	6
1 Jet	23	63	79	13	43	32	1 Jet	33	55	4	95	38	44
2 - -	77	54	75	57	7	47	2 - -	60	46	12	78	93	76
3 - -	62	2	42	64	86	84	3 - -	21	88	94	80	15	34
4 - -	70	53	5	74	31	20	4 - -	14	39	9	30	92	19
5 - -	29	41	50	11	18	22	5 - -	45	65	25	1	28	17
6 - -	83	37	69	3	89	49	6 - -	68	6	35	51	61	10
7 - -	59	71	52	67	87	56	7 - -	26	91	66	82	72	27
8 - -	36	90	8	73	58	48	8 - -	40	81	24	16	85	96

Figure 1: Table for using die rolls to construct bars of a minuet and trio. Scan from https://imslp.org/wiki/File:PMLP243537-kirnberger_allezeit_fertiger_usw.pdf.

Pros & Cons of Music Dice Games

PROS

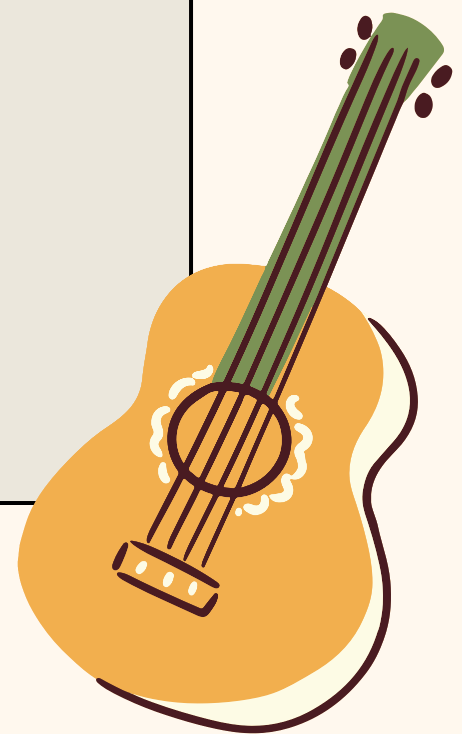
- $6^{16} = 2.82110991 \times 10^{12}$ potential arrangements
- This was a seemingly great way to generate “random” music, especially when we consider that Markov Chains weren’t introduced til 1906

CONS

- Aren’t really generating pieces
  - Music is pre-written, the players are just arranging it
- Leaves us to question if we could really randomly generate music

Early Computer Music Generation

- Unlike Music Dice Games, couldn't let notes be completely independent
  - Would create a cacophony rather than "music"
  - Therefore, use markov chains to stop notes and duration from being independent
- First used Markov Chains & the ILLIAC I computer to compose/generate *Illiac Suite* in 1957
  - Made of four movements, the first more simple, the second created 4 vocal parts, the third was more modern and the fourth was focused on what the system could create



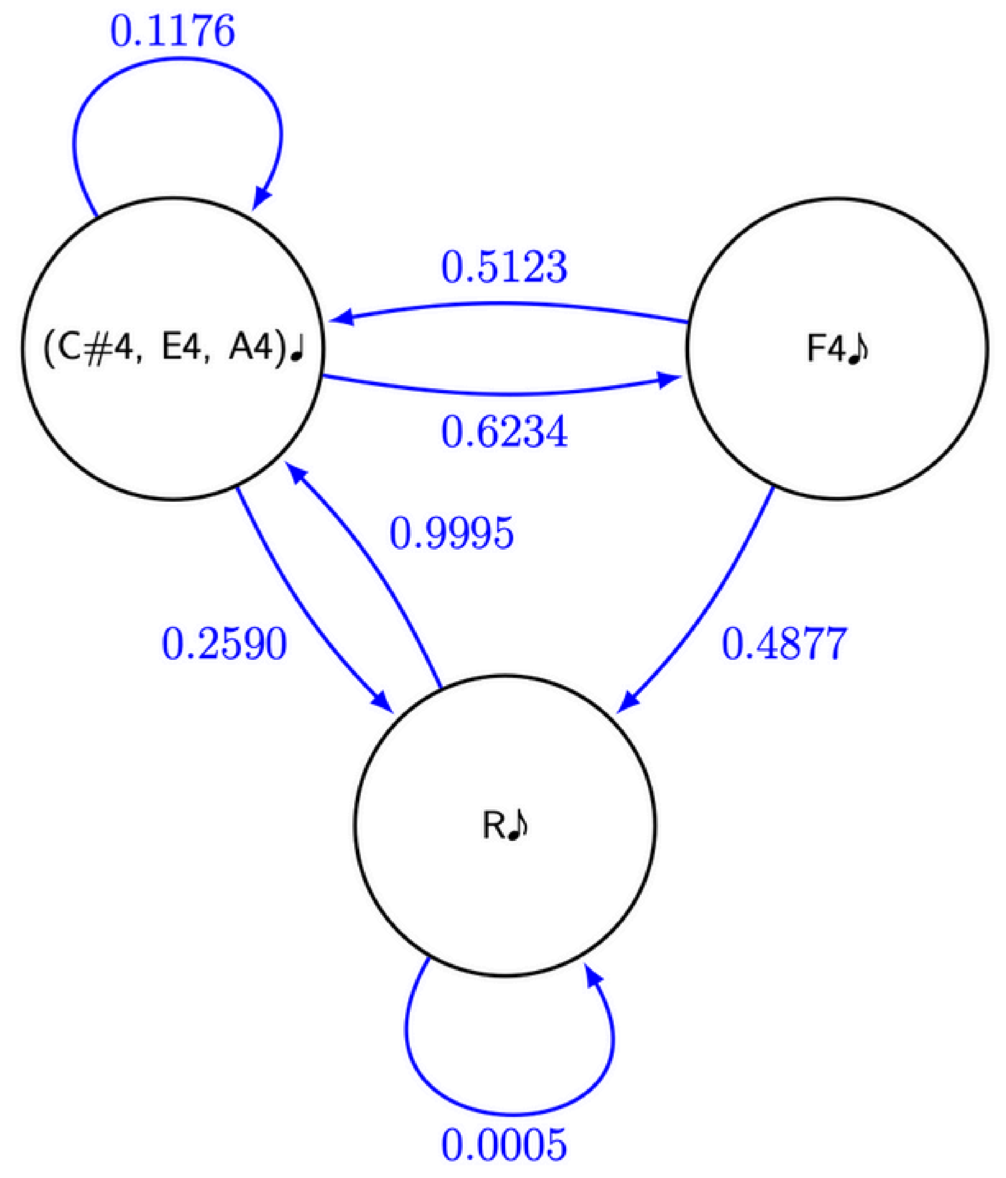


Figure 3: Musical Markov Chain example using graphical representation.

Creating Markov Chain

- Training data: estimates the probabilities for the Markov chain
- Nodes (states): sound objects- contain information about a single note/chord
- Transition matrix
- Initial probability vector: based on number of times that the sound object occurs in training data

Creating Markov chain

- Transition matrix and initial probability distribution vector

State	(C#4, E4, A4)♩	F4♪	R♩
(C#4, E4, A4)♩	0.1176	0.6234	0.2590
F4♪	0.5123	0.0000	0.4877
R♩	0.9995	0.0000	0.0005

Figure 4: Musical Markov Chain example using a transition matrix.

(C#4, E4, A4)♩	F4♪	R♩
0.3333	0.5000	0.1667

Figure 5: Musical Markov Chain Example

Limitations with the research

- Cannot calculate probability with multiple voices/instruments simultaneously
- Do not take the dynamics of the music into account
- Hard to work with less tonal music (such as Jazz)
- Some of the issues can be addressed through other statistical model (Hidden Markov Chain, etc)

Generate music in Python

STEP 1: Parse the training music

For each individual note and chords, four attributes have been extracted from the input music:

Pitch
1) Notes: A, B, C, D, E, F, G,
2) accidentals: including sharp, flat, or natural

Octave
3) Octave of each note: by an integer from 0 to 8

Duration
4) Whole note, half note, quarter note, or shorter value.

These four attributes have been combined to be states. Then two dictionaries have been set to save the transition frequency between states and then generate transition matrix of DTMC.

(1)

$$
\begin{array}{l}
 \\
C\flat3♪ \\
(D3,G3)♪ \\
E\flat2♩ \\
\ldots \\
((B\flat3,B\sharp4)♩ \\
((C6,C7)𝅗𝅥 \\
R𝅝
\end{array}
\begin{array}{c}
\begin{array}{ccccccc}
C\flat3♪ & (D3,G3)♪ & E\flat2♩ & \ldots & ((B\flat3,B\sharp4)♩ & ((C6,C7)𝅗𝅥 & R𝅝
\end{array} \\
\begin{pmatrix}
0.16 & 0.2 & 0.2 & \ldots & 1 & 1 & 1 \\
0.047 & 0.047 & 0.095 & \ldots & 1 & 1 & 1 \\
0.25 & 0.25 & 0.25 & \ldots & 1 & 1 & 1 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1 & 1 & 1 \\
0 & 0 & 0 & \ldots & 0 & 0 & 1 \\
0 & 0 & 0.5 & \ldots & 0.5 & 0.5 & 1
\end{pmatrix}
\end{array}
$$

(2)

$$\alpha = \begin{pmatrix} 0.02 & 0.05 & 0.09 & \ldots & 0.97 & 0.99 & 1 \end{pmatrix}$$

Normalized transitional matrix of notes generated from the input music

Why the sum of each row ≠ 1?

Last note becomes an absorbing state?

$$
\begin{array}{c|ccccccc}
 & C\flat3♪ & (D3,G3)♪ & E\flat2♩ & E3♩ & ((B\flat3,B\sharp4)♩ & ((C6,C7)𝅗𝅥 & R𝅝 \\
C\flat3♪ & (\ 0.16 & 0.04 & 0.04 & 0.5 & 0.2 & 0.06 & 0\ )
\end{array}
$$

$\Downarrow$ **cumsum( ) Let the cumulative sums to 1!**

$$
\begin{array}{c|ccccccc}
 & C\flat3♪ & (D3,G3)♪ & E\flat2♩ & E3♩ & ((B\flat3,B\sharp4)♩ & ((C6,C7)𝅗𝅥 & R𝅝 \\
C\flat3♪ & (\ 0.16 & 0.2 & 0.24 & 0.74 & 0.94 & 1 & 1\ )
\end{array}
$$

Last note

Quarter rest

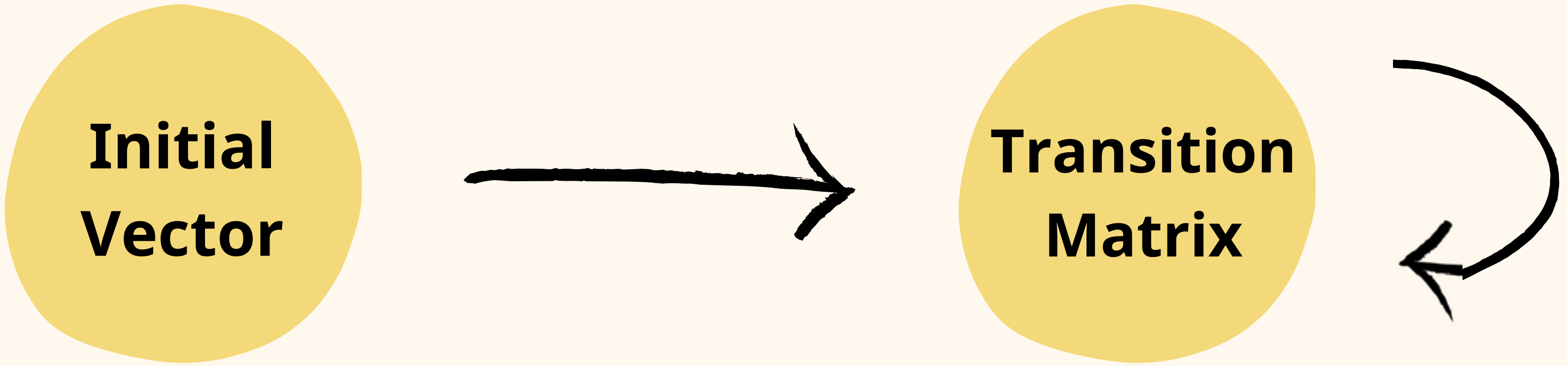
First note

STEP 2: Generate new music

Line segment analogy **Drawed from a uniform distribution U(0,1)**

Figure 1: Example of the Inverse Transform Sampling

Bonus: This method is also used in the default sample() function in R!



We can customize the length of music

Our improvements: Not all durations are defaulted to be floating point numbers, especially source files from different websites. We generalized its applicability to more types of music by conversions.

Enjoy the music!

Thank you