Music Improvisation using Markov Chains

by
Erlijn J. Linskens

Supervisor:
Dr. Gijs Schoenmakers

University College Maastricht
Maastricht University

June 13, 2014

Abstract

This thesis investigates the possibility to create improvisation in music using Markov chains. With improvisation is meant here that within a musical composition one specific part, like one musical line, will be improvised. Using the programming language `Matlab`, an algorithm is created that enables a computer to play out a musical composition, and to define a section within this composition that will be improvised with the help of Markov chains.

Improvisation is meant to be original each time, while it still needs to be recognizable to the original version of the musical line. Markov chains are a suitable tool to model this kind of improvisation in music with, as its dependence on probabilities will ensure that the improvisation will be original each time, while the probabilities can ensure that the improvisation is recognizable to the line it originally was. In this thesis it is found that the best manner in which the improvisation will stay recognizable to the original version, is to use the original version's chords as a basis for the transition matrices. Further, an important part of such improvisation is its transitions from and to the non-improvised lines. These transitions need to be smooth. This too can be incorporated in the algorithm such that the probabilities of a note to be the same note as it would be in the original version are higher towards the end.

Keywords: Markov chains, music generation, Matlab, improvisation

Contents

1 Introduction

One of the reasons why people go to live music concerts is because the way songs or musical pieces you already know are played on CD are different from the way they are played live. They are more original live. Most people believe this to be a completely human trait. Indeed, it might be. However, perhaps it can be faked. What if a computer could improvise a musical piece such that it seems that a person altered it rather than an algorithm? That is exactly the aim of this thesis. Using mathematics, and more specifically Markov chains, it should be possible to generate music improvisation inside an existing musical piece. This improvisation should be suitable to the rest of the piece, have smooth transitions from and to the non-improvised line before and after the improvisation, but also it should not deviate too far from the original version of the musical line.

In previous studies, the concept of music generation using Markov chains has already been grasped at. However, the possibility to create improvisation with smooth transitions into an existing piece has not yet been researched. Thus, the hypothesis of this thesis is that Markov chains can be implemented in a computer program such that it will alter an existing musical piece to make a part of it sound like improvisation. In the rest of this section, the mathematical background and the research that was previously done regarding music generation with Markov chains will be discussed. After that, the section 'Method' will follow. In this section, a short description of what the algorithm should be able to do is given, after which a similar description follows of what the programming language needs to be able to do. Next, a step-by-step explanation is given of how the algorithm that will do the improvisation was created. This creation happened in four steps. First, the algorithm needed to be able to play out the original composition. Second, the variable *pitch* was introduced to the code. Here, the Markov chains made their first appearance in the code. Third, the *duration* was added in a fashion similar to the variable *pitch*. After this the algorithm had the ability to play out the original version of the musical piece, with one musical line that it generated on its own based on transitions of the original version. Lastly, this music generation needed to be transformed into music improvisation. This was done by introducing the variable *chords*. In the section after that, 'Analysis', the final version of the algorithm is implemented with a musical composition to show the final results of this research. These results will be discussed in the discussion, after which of course a conclusion will follow.

1.1 Mathematical background

The topic of this thesis is music improvisation using the mathematical tool Markov chains. Markov chains are a fundamental tool in the field of stochastic processes, which in turn is a more specific subject in the area of probability theory.

Probability theory is a field of study in which is explained how different probabilities interact with each other. When probability theory is used to model random phenomena, this random process is called a stochastic process. A stochastic process is a time-dependent family $(X_t)_{t\in T}$ of a random variable X where t is a time index belonging to a parameter set T (Privault, 2013). A stochastic process $(Z_n)_{n\in N}$ with a discrete state space $\mathbb{S}$ is said to have the Markov property if, for all $n \geq 1$, the probability distribution of Z_{n+1} is determined by the state Z_n of the process at time n, and does not depend on the past values of Z_k for $k \leq n-1$ (Privault, 2013). Thus, a Markov process is a stochastic process whose future state is only dependent on the current state, and not on any previous states. A Markov chain in turn is a discrete-time Markov process with a finite set of states. To visualize this mathematically, for all $n \geq 1$ and all $i_0, i_1, ..., i_n, j \subset \mathbb{S}$ we have

$$\mathbb{P}(Z_{n+1} = j | Z_n = i_n, Z_{n-1} = i_{n-1}, ..., Z_0 = i_0) = \mathbb{P}(Z_{n+1} = j | Z_n = i_n)$$

In this thesis, the probabilities between states will be treated as *time homogeneous*. That is, the probability $\mathbb{P}(Z_{n+1} = j | Z_n = i)$ is independent of time $n \in N$(Privault, 2013). The transition probabilities can be modeled in an $\mathbb{S} \times \mathbb{S}$ matrix. This is called the transition matrix of a Markov chain:

$$[P_{i,j}]_{i,j \in \mathbb{S}} = [\mathbb{P}(Z_1 = j | Z_0 = i)]_{i,j \in \mathbb{S}}$$

This transition matrix models the probability that the chain will go from one state do another, thus attempting to predict the outcome of a random phenomenon as accurately as possible. An example of a transition matrix can be seen in Table 1. In this case, the current states are modeled by the row variables, while the future states are modeled by the column variables.

Table 1: Transition matrix

	A	B
A	0.3	0.7
B	0.4	0.6

For example, the number 0.4 in Table 1 indicates that there is a probability of 40 percent that the chain will move from state B to state A, whereas there is a probability of 60 percent that it will move return to state B. This Markov chain is visualized in Figure 1. Markov chains are very suitable to model music with, as will be explained in the next section.

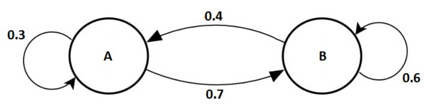


Figure 1: Markov chain

1.2 Previous research

Before continuing this section mentioning the relevant previous research on the topic of music generation and improvisation using Markov chains, it should be explained what precisely music generation is. Music generation is the concept of creating music using a computer. A program should receive certain input, after which it will use this data to output a set of notes. This string of notes would shape a musical line, thus generating music. Music improvisation is different in the way that improvisation is different from regular composition. Improvisation means that the musician creatively alters the existing melody, without previous study. Often musicians would invent faster variations of the melody, or create new melodies in accordance with a set of chords that flows naturally with the existing melody. The improvisation should follow a similar general height of the pitch of the original version to maintain the natural flow of the existing music to the improvised music and vice versa. The final product however, would be a unique improvised melody. Markov chains are perfect to do this improvisation, as they have no memory of earlier states and would assign new notes according to the probabilities of the current note in the transition matrix, thus making the outcome different each time: just like improvisation.

Markov chains have been considered a suitable tool for generating music for some time. They are widely used in algorithmic music composition, being a standard tool in music mix and production software (Dawin & Volchenkov, 2010). This is the case because music is probabilistic; certain pitch progressions are more likely to occur than others (Bell, 2011). Since music consists of time-sequences of musical events, such as notes; chords; dynamics; and rhythmic patterns, music can be seen as a random process (Liu & Selfridge-Field, 2002). Liu and Selfridge-Field (2002) argue that since other types of time-sequences can be modeled successfully, the same should be the case for music. Markov chains are stochastic processes that develop in time in a manner controlled by probabilistic laws, dealing with the interdependence of events (Marom, 1997). It can be logically derived that Markov chains are an acceptable tool to model music with. Furthermore, due to the assumption that the future is independent of the past given the present, a Markov chain can be characterized by a simple state-transition probability matrix (Liu & Selfridge-Field, 2002), making it easy to use for modeling music.

Previous research on music generation using Markov chains has been conducted among others by (Liu & Selfridge-Field, 2002). Liu and Selfridge-Field believed that composers follow some sort of rules while composing music. Accordingly, it should be possible to create a model that could examine a musical piece it had not been exposed to before, and be able to identify the composer. Liu and Selfridge-Field (2002) only investigated composer identification, and created a relatively simple Markov model to do so. The only variable used in the context of Markov chains was *pitch*.

Similarly, Bell (2011) claimed that all composed music follows a set of rules chosen by either the composer or the musical norms of the time. These rules can be followed by a computer to automate the composition process (Bell, 2011). Bell presented a method for algorithmic music composition using two techniques: Markov chains are utilized in this system to choose the future pitch, rhythm, and chord based on the current pitch, rhythm and chord; while genetic algorithms are used as a search method that finds the set of Markov chains that yield the most pleasant sounding music. In this paper it was noted that there is a disadvantage of using first-order Markov chains, since they generally do not yield 'phrased' passages found in human-composed pieces (Roads, 1996). However, a solution can be found in second-order Markov chains (Bell, 2011). Bell (2011) uses the variables *pitch*, *chord structure* and *rhythm*.

Contrary to Liu and Selfridge-Field (2002) and Bell (2010), Dawin and Volchenkov (2011) did not use an existing musical piece to create the transition matrices in their paper on musical dice games, but tonal harmonies. This means that the probability of two notes following each other is higher if they make a pleasant sound together. Dawin and Volchenkov (2010) mention that the Markov chains determining random walks in music are not ergodic, which has as a consequence that starting from a specific note it might not be possible to reach every other note which is in the score of the musical piece. Further, every pitch in a musical piece is characterized with respect to the entire structure of the Markov chain by its level of accessibility (Dawin & Volchenkov, 2010). This accessibility is estimated by the average length of the shortest random path toward the pitch from any other one randomly chosen in the musical score. The variables Dawin and Volchenkov (2010) used to define a note event are *time* (similar to *rhythm*), *note* (similar to *pitch*), *channel*, and *velocity*. *Channel* accounts for the use of more than one instrument; while *velocity* describes the strength with which the note is played (Dawin & Volchenkov, 2010).

Last but not least, (Marom, 1997) did not just examine music generation, but looked at the concept of improvisation as well, with the aim to create a model that could improvise jazz. He identified two approaches to generate music. The first approach is a self-contained composing procedure that creates the music solely through algorithms that do not depend on any input, and that synthesizes new, truly original, music (Marom, 1997). This is not very relevant to improvisation. The second approach is a procedure that creates music by following some guidelines, or characteristics, that are inherent in an existing piece of music (Marom, 1997). This approach can be seen as trying to reproduce music or style. This is very relevant to improvisation. Marom focused on two important issues in creating his model: First, to capture the style of the composer; second, to allow for spontaneous actions. This means that the algorithm requires both a copying mechanism and a degree of randomness. Naturally, Marom believed that Markov chains are an adequate tool to achieve this. He found two approaches that lie somewhere in between reproducing the piece, and creating something original. The first approach uses a single Markov chain to model each of a number of attributes of jazz improvisation, and then combines the different chains into one model that describes the compositional process as a whole (Marom, 1997). This approach is called Simple Markov Chains. No external event is allowed to influence the development of the model here. The second approach is an extension of the first, where there is an underlying sequence of states controlling the behavior of the Markov chains, which he called Controlled Markov Chains. These states are some sort of external conditions that have an effect on the improvisation, such as other instruments (Marom, 1997). However, since this thesis excludes the possibility of external influences, only the first approach is taken into account here. The jazz attributes used by (Marom, 1997) are *pitch*, *duration* (*rhythm*), *volume* (similar to *velocity*) and *note/rest events*. *Note/rest events* define whether a note or a rest will be played. Further, it is interesting to mention that Marom (1997) found that higher-order Markov chains provide a more 'stable' outcome as opposed to first-order Markov chains, similar to the 'phrased' outcome in Bell (2010). Specifically, the variables *pitch*, *duration* and *volume* seem to benefit from higher-order Markov chains.

Although most of the variables used by different researchers provided overlap, not all of them proved to be useful in the context of this thesis. *Pitch* and *duration* were clearly found useful, as these two variables define notes. However, the musical line that is to be improvised is too short to be able to implement *note/rest events* properly, so this variable is not included. Further, the use of multiple instruments, which requires the variable *channel*, is beyond the scope of this thesis, and *volume* is irrelevant because with the mechanical tone of the computer program, it is difficult to here a difference. *Chord structures* however have been found very useful in the music improvisation, since it will enable the program to not deviate from the original version too far. Thus, this variable has been included as well. Together, these three variables present the most suitable method for creating improvisation using Markov chains, while simultaneously they are not too difficult to be modeled with. Thus, the variables that will be used in this thesis are *pitch*; *duration*; and *chord structures*. For clarity, the variables will be defined once more. *Pitch* is the height of the tone, while *duration* denotes the length of the note, for example whether it is a whole, half, quarter, or an eighth note. The variable *chord structures* define the chords that belong to a specific part of the musical composition. In this thesis, only triads are used, although it is also possible to create chords with four pitches rather than three. A triad consists out of three notes that sound well together. A triad is created by first taking the scale of a base note. The triad is then the combination of the first note of this scale (the base note), the third and the fifth note of the scale. There are differences between types of triads which will be discussed in the subsection 'Improvisation' of the section 'Method'.

Although it should be clear why this thesis is different from the first three papers mentioned in the preceding section, it might be necessary to explain the difference with (Marom, 1997) more clearly. Although both this thesis and the research conducted by Marom focus on improvisation, Marom improvises an entire musical piece, rather than just a part of a composition. Marom focuses on jazz improvisation, in the sense that the algorithm should somehow respond to external events, such as other instruments, in its improvisation. Rather, the algorithm in this thesis could be compared to improvisation in an empty room. It is not influenced by external events, but rather takes an existing piece and tries to recreate it in an original way, taking into account the transitions from the original to the improvised piece and the other way around.

2 Method

2.1 The algorithm

The aim of the algorithm is to enable a computer to improvise music with the mathematical programming language `Matlab`. Since when humans improvise, they often know where they want to improvise, it should be possible to specify in the program the piece of the song that should be improvised. That specified part should be similar to the original version, but still be unique. The rest of the song should be exactly the same as the original version. Further, it should be possible to select several musical lines that will be improvised by the program.

The algorithm is going to be created mainly with mathematical tools. Markov chains and transition matrices will be the most important factors in the creation of the music. A state in the Markov chain will have the three aspects defined in the earlier section: *pitch*; *duration*; and *chord structures*. The first two variables will be incorporated in the algorithm in the form of transition matrices, while the third variable will be incorporated as a vector. The improvisation will start after the algorithm has played out the part of the original musical piece that precedes the improvisation. It will start with a current state. This first current state is based on the chord structure of the part of the musical composition that affects the next state. After the current state is processed, the algorithm will start looking for the next state. To find the next state, first it will process the transition matrix for the *duration*. Once the duration is found, the algorithm will continue to the transition matrix of *pitch*. Combining these two variables, it will then give the output of the new state. After the new state is found the process will be repeated, starting again by assessing the current chord. After most of the notes have been processed, the algorithm should start to find a way to bring the improvised music closer to the original version. This will happen in the last bar that is to be improvised, using altered transition matrices that give higher probabilities to the pitch and duration that are in the original version. These altered transition matrices will ensure that after the last note of the musical improvisation is played, the music will continue with the original version with a fluent transition. The process is visualized in Figure 2.

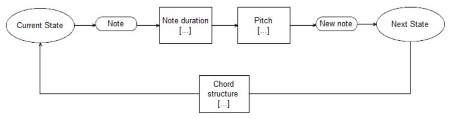


Figure 2: The transition process between notes.

2.2 The programming language

Although the mathematical methods are central in this thesis, computer science is a vital part of the project as well. The mathematical program used should be able to represent the music, but most importantly it should be able to output a state for every transition matrix. There are several criteria which the programming language must fulfill. First of all, the program should be able to play out the original musical piece. Next, it should be able to incorporate transition matrices, which will enable the algorithm to alter the musical piece in such a way that it is similar to the original yet unique every time the algorithm is implemented. Last, the program should be able to somehow select the part of the music that will be improvised. The rest should than stay exactly like the original. The programming language chosen to implement this algorithm is `Matlab`. `Matlab` is a technical computing language by `MathWorks`, known mostly for its ease to work with matrices. Although the musical tones in this program sound quite mechanic, it will do for the purpose of this thesis.

2.3 Creating the algorithm

2.3.1 The original version

In this section a step-by-step explanation will be given of how the algorithm is created. However, first a suitable musical piece had to be chosen to implement the algorithm with. The musical piece could not be too difficult, as otherwise it would take too long to process it in a program. Further, it needed to have an adequate number of notes to make the transition matrices with, and it needed to be played out by only one instrument. The piece chosen was the *Ode to Joy* by Beethoven.

After the musical piece was chosen, the first step that had to be accomplished was to make sure `Matlab` could play out the original piece as a whole. The first step was to define a sample rate Fs. This was defined to be equal to 8000. Then, the notes had to be defined using sine waves. Each pitch has a corresponding frequency, which needed to be defined. In Appendix B, the variables defining these frequencies can be found. For example for the pitch B4, the variable of the frequency is called F_B4. After the frequencies were defined, it was possible to define the actual note. This was done with the equation $B4 = sin(2 * pi * F\_B4 * (0 : 0.000125 : 0.5)$. In this formula, (0:0.000125:0.5) stands for the duration of the note. It creates a matrix from 0 to 0.5 with steps of 0.000125. This matrix contains the samples that capture the amplitude of the sound over time. In this case, the length of the note was defined with 0.5 seconds. Later, this number was replaced when the variable *duration* was incorporated, such that different durations were able to be played. After the pitches were defined, the music composition that was used could be incorporated in the code. Each bar of the musical piece was presented in the code as a $1 \times n$ matrix, where n is the number of notes. To make up the whole musical piece, the matrices of all the bars were concatenated to each other in one long matrix called *original* and the musical composition could be played using the command sound(original,Fs).

2.3.2 Pitch

After the original piece could be satisfactory played out, it was time to continue with a more important part of the algorithm: the music generation. Initially, *pitch* was the only variable used in this implementation. Rather than using a program to extract the data from an existing musical piece, the transition matrices were derived from the lead sheet of the composition by hand. To make the $p \times p$ transition matrix – p being the number of pitches – for *pitch*, the entire melody was taken into account. The more notes are included in making the transition matrix the better, as it gives a more accurate assignment of appropriate notes. The transition matrix for pitch was called *tmpitch*, in which *tm* stands for transition matrix. The most challenging part in this first version of the algorithm was to ensure that the algorithm would choose the new note suitably given what the current note is. Accordingly, given a current note, a $1 \times p$ zeros vector called *multiplier2* was made that would take the current pitch into account and would change the output of the place associated to that pitch to 1 in the vector. This vector would be multiplied with the transition matrix, such that only a $1 \times p$ vector concerning the transition probabilities of the current vector would be left. To choose which pitch comes next to the current one, a random variable was used. In Appendix B it is possible to see how this method went into effect. The note that came as outcome for the new pitch is then recorded in a matrix, after which the new pitch will become the new current pitch. This will continue for the course of one musical line, being four bars. After this was successfully completed, the matrix of this music generation was incorporated in the original piece, replacing bars 9 to 12. The outcome is showed in Appendix B.

2.3.3 Duration

The next variable to include was *duration.* The duration of a note is highly dependent on the time signature of the note, which is placed at the beginning of a musical piece. Here is introduced the concept of 'ticks'. The duration of a note can be defined in ticks. The nominator of the time signature tells us how many ticks a bar is long. The denominator of the time signature defines which note form is worth one tick. For example, with a musical piece with time signature 4/4 the note form 1/4 is worth one tick; and there are four ticks in one bar. A different time signature, for example 2/2, assigns one tick to the note form 1/2 and two ticks to one bar. The duration of one tick is defined by the tempo of a musical piece. Such a tempo is usually defined at the beginning of a musical piece. If it is not, it is usually about a hundred to a hundred-twenty beats per minute (bpm). The beats per minute define how many ticks there are in a minute.

In the algorithm, Fs1 stands for the sampling frequency, or samples per second. If the piece needs to be has a certain number of beats per minute, that would mean there should be $bpm/60$ beats per second. $1/(bpm/60) = 0.6$, which is assigned the variable *temp*; the number that should be multiplied with the sampling frequency to get the desired tempo. In the code, the number of ticks of the note can be placed behind the pitch of the note between brackets, to assign that number of ticks to that pitch.

As has been mentioned before, the formula that creates the notes was changed to include duration. The formula was given a slightly altered structure to include the variables T, defined by $1/Fs$, and *n(M)*, defined by $1 : M * N$ in which M is the number of ticks of a note. The renewed formula is $B4 = @(M)sin(2 * pi * F_B4 * T * n(M))$, as can be seen in Appendix B.

The manner in which the next duration is chosen due to the current duration is similar to the procedure with *pitch* and there is a similar output in the form of an improvisation matrix inserted in the original piece. With the implementation of *duration*, a necessity arose to ensure that the duration of a bar could not exceed the number of ticks assigned in the nominator of the time signature, in this case 4 ticks. Accordingly, this constraint was incorporated in the code in the same place where is defined which duration will be next. This was replaced in the final version by a loop of the chord structures though. The last notable difference with this code and the previous one is the implementation of short rests between different notes. Without them, it is almost impossible to tell where one note stops and the other begins, in the case of the same pitch being assigned to two consecutive notes. The short rest is a very short break between two notes, just long enough to hear that two different notes are being played, rather than one longer one.

2.3.4 The improvisation

The next step to take is to improve the improvisation. So far, it is only a matter of which transition matrices are used how new notes will be output, but that is not how improvisation is supposed to take place. It is, however, difficult to define characteristics of improvisation, because improvisation is an ever-changing phenomenon, indefinable by definition. However, an attempt was made to retrieve some basic characteristics that are independent of the musician, as was necessary before any serious effort to turn music generation into music improvisation could take place. To find such characteristics, the book on improvisation by Bailey (1993) was used. This book contains chapters of improvisation in different musical styles. For this thesis the style of baroque was used, since this style is the one most similar to the classical style of Beethoven's Ode to Joy. As was mentioned by Bailey (1993), it was the practice during the baroque period to construct and organize chords on an actual, rather than a theoretical bass, the construction relating to and deriving from the lowest note, not to a theoretical root (Bailey, 1993). Bailey (1993) mentions this method of using chord structures for improvisation, and it is a suitable method, as chords consist by definition of tones that sound well together. It is from this method of improvisation that the method for improvisation in this thesis is derived. Since the original version should still be recognized slightly, but the improvisation should still have the freedom to move randomly, the chords corresponding to the bars that will be improvised will be used for improvisation. These chords will guide the improvisation in the right direction, while the improvisation will have the freedom to move randomly within this chord structure. For each of these chords, a transition matrix will be created of the notes within the triad for available pitches. A triad consists out of three notes that sound well together. A triad is created by first taking the scale of a base note. The triad is then the combination of the first note of this scale (the base note), the third and the fifth note of the scale. The difference between a minor and a major triad is defined by the way the scale is formed. If the scale is formed by the intervals $basenote + 1 + 1 + 1/2 + 1 + 1 + 1 + 1/2$, the triad is major. If the scale is formed by the intervals $basenote + 1 + 1/2 + 1 + 1 + 1/2 + 1 + 1$, the triad is minor. Thus, a minor triad differs from a major one in the sense that the second note of a major triad is 1/2 tone higher than the second note from the minor triad. The rest is the same. The three notes of a triad are placed in a transition matrix for pitch of the note in the improvisation with the corresponding chord. However, with only three possible pitches the options of the algorithm would be quite limited. That is why the second and fourth notes of the scales were included in the transition matrices with very small probabilities as well. That is, from the intervals mentioned earlier, not just the base note, its third and its fifth tone of its scale were taken, but its second and fourth tone as well. The chords assigned to the bars that need to be improvised are incorporated in the code in a $c \times 1$ vector called *chords*, with c being the number of chords. At the point where the improvisation needs to begin, the algorithm will be in a loop of the matrix *chords.* It will begin with the first value, the first chord of the musical line. This value is assigned to the variable current_chord. Then it will check by means of an if–statement which chord equals that variable. It will then continue choosing the note in the same manner explained before. This continues for the duration of the number of ticks assigned to that chord, after which it will return to the next value of the matrix 'chord' and repeat the process.

Further, the duration transition matrices of the improvisation are changed as well. Improvisation is usually slightly faster than the original version, so that the notes can play around a bit. Rather than ranging from whole to eighth notes, the new duration transition matrices will vary between fourth, eighth, three-sixteenth or sometimes even sixteenth notes. By the end, these transition matrices will be slightly altered such that the durations are a bit longer again, to make for a more smooth transition back to the original piece.

3 Analysis

Since the code is created, it might be useful to show how a musical composition can be incorporated in the algorithm. The musical piece used to create the algorithm is *Ode to Joy*, the final movement in Beethoven's Ninth Symphony, and this composition is also used to show how a composition can be incorporated. The piece was written with the time signature 4/4, in the tempo Maestoso. This means that it is supposed to be played in a stately manner, with a fourth note being one tick and four ticks in one bar. This was incorporated in the code by defining the algorithm to play 120 beats per minute. The notes that make up the composition are shown Appendix A, where the lead sheet of Ode to Joy is displayed. The piece is written in G Major. The chords used in the *Ode to Joy* are mostly major triads, and a few minor triads. The piece consists of sixteen bars. These bars were created in the code with a matrix containing the corresponding notes, as can be seen in Appendix B. These sixteen bars correspond to four musical lines. Of these four musical lines, the third one is to be improvised. In this third line there is only one minor triad, the rest are all major triads. The chords defined for this line are:

```
|D - G | D - G | D - B | Em - A - D - G|
```

This information can be derived from the lead sheet, which is shown in Appendix A. As can be seen, there are two chords assigned to the first three bars. Each of these bars has two ticks. For the last bar however, each bar has only one tick. This was translated to the code by making to different loops. The first one is for the first six chords, assigning notes for the duration of two ticks two each chord before continuing with the next one. After that it continues to the second loop, assigning notes for the duration of one tick to each chord. After that, it outputs the improvisation matrix.

For the chord `D`, the corresponding major triad D consists of the notes `D - F-sharp - A`. The second and fourth notes that are included in its transition matrix as well are `E` and `G`. For the chord `G`, the corresponding notes are `G - B - D`, with extra notes `A` and `C`; the chord `B` has `B - D-sharp - F-sharp`, the notes inbetween being `C-sharp` and `E`; the chord `Em` stands for a minor triad with base note E, and its corresponding notes are `E - G - B` while the notes inbetween are `F-sharp` and `A` while the chord `A` is a major triad again with notes `A - C-sharp - E` with secondary notes `B` and `D`.

The initial transition matrix for pitch between chords is shown in Table 2, with the example taken of the chord `D`. The transition probability to a pitch that is also in the corresponding bar of the original version is for example higher than the transition probability to a pitch that is not. A problem arises here considering the transitions between different chords. Namely, the note with which one part of the improvisation ends will not necessarily be in the transition matrix of the next chord. This can be solved by taking the variable current_pitch to be a note that is similar to both the previous and the coming chord. It happens that every two consecutive chords have one note in common, and this note will be taken to be the current_pitch for the next section.

Table 2: Pitch transition matrix for chord D

	D	E	F#	G	A
D	4/14	1/14	4/14	1/14	4/14
E	4/14	1/14	2/14	1/14	6/14
F#	3/14	1/14	3/14	1/14	6/14
G	4/14	1/14	2/14	1/14	6/14
A	3/14	1/14	3/14	1/14	6/14

4 Discussion

This section discusses the results of the music improvisation. First relevant factors found during the process will be mentioned. Then, the strengths and weaknesses of the model will be assessed. Further, the importance of the findings is mentioned and lastly suggestions for further research are made.

Several factors of a musical piece were found to be relevant for the output of the improvisation. The time signature is one of these factors. The numbers defined in the nominator and denominator of the musical composition define which duration is assigned to the different note forms, and how many of them there are in a bar. If the wrong time signature is used, the speed with which the composition is played in the algorithm will be completely different from what it is supposed to be. A different denominator of the time signature is defined in the code by changing the number of ticks after a note. Another factor that was found to be important is the tempo. The tempo changes how many beats per minute are heard. While the time signature defines how many ticks are heard in one bar, the tempo defines the duration of the tick itself. Further, a factor that needs to be taken into account while using this model is the musical style of the composition that is used. For the improvisation in this thesis, the perspective was taken for baroque-style music. Although this model might work as well for other musical styles, slight changes in the improvisation methods might be in order. Jazz improvisation for example is highly dependent on interaction with external sources, and thus its improvisation is different. Compositions from musical styles with their own specific improvisation methods might be unsuitable to use with this algorithm.

This inability to overarch all improvisation styles is a weakness of the algorithm. However, a major strength of this algorithm is its ability to create improvisation based on harmonic structures. This method is used frequently in improvisation, and might be seen as an overarching aspect of improvisation. Another strength of this model is that it can be applied to other musical compositions (at least within the same style) as well. Although it requires a bit of work to enter the data relevant for the musical piece, it is not at all that difficult to be done. [if ornaments are included, add them as extra weakness as different musical pieces use different ornaments]

This research about music improvisation using Markov chains is important mostly because it has never been done before. It allows for new insights in the field of music generation using Markov chains. Next to that, it could be mentioned as an addition to attempts to re-integrate improvisation and composition in classical composition, which have been continuing since the 1950s (Bailey, 1993). This model can be regarded as an attempt to open a way for classical music in modern times.

In further research, this model could be combined with the model created by Marom (1997), to fully grasp improvisation in jazz. Furthermore, other types of improvisation could be researched and included, such that eventually the model can be used for all musical compositions, rather than just compositions which require this type of improvisation.

5 Conclusion

The aim of this thesis was to investigate the possibility of creating music improvisation using Markov chains. Several steps were taken in order to make this a reality. The computer program `Matlab` needed to play out the original musical piece. Then, music generation within the musical composition was included using Markov chains. First transition matrices of pitch were incorporated, later the transition matrices of duration as well. After it was possible to create music generation using the algorithm, the next step to take was to create music improvisation. It was found that chord structures are a good tool to transform music generation into music improvisation. The algorithm is given the freedom to vary between notes within the chords assigned to the appropriate part of the musical composition, and it even has the possibility to go to a pitch lying between two notes that are defined in the chords. This ensures that the improvisation does not deviate too far from the original composition, and the original version can still be recognized from the improvisation. The Markov chains make sure however that the exact same improvisation is never repeated, the same way improvisation is never repeated in exactly the same manner. The final algorithm was applied to the musical composition Ode to Joy by Beethoven. Here it was discussed what precisely needs to be done to incorporate a musical piece in the algorithm, and it can be used as a manual for incorporating other musical compositions in the algorithm.

References

Bailey, D. (1993). *Improvisation: Its Nature and Practice in Music.* The Perseus Book Group.

Bell, C. (2011). Algorithmic music composition using dynamic markov chains and genetic algorithms. *Journal of Computing Sciences in Colleges*, 27(2):99–107.

Dawin, R. & Volchenkov, D. (2010). Markov chain analysis of musical dice games.

Liu, Y. W. & Selfridge-Field, E. (2002). Modeling music as markov chains: Composer identification.

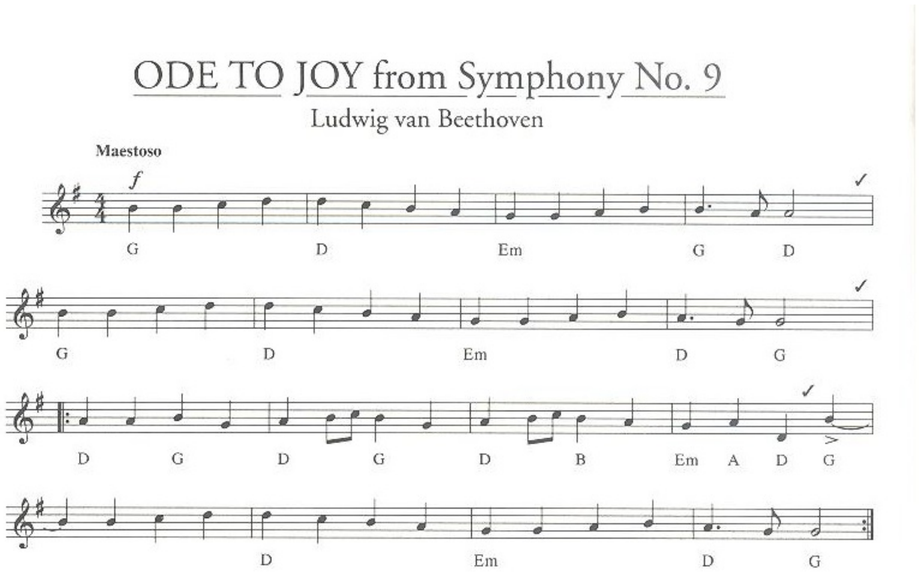
Marom, Y. (1997). Improvising jazz with markov chains.

Privault, N. (2013). *Understanding Markov Chains: Examples and Applications.* Springer Undergraduate Mathematics Series. Springer.

Roads, C. (1996). *The computer music tutorial.* MIT press.

Appendices

A Lead sheet



Source: All the tunes you've ever wanted to play (book 1). (1993).

B Final version

```
clear all

Fs1 = 8000; %Set Sampling Frequency
Fr = Fs1; %Set Reconstruction Frequency
Fs = Fs1; % The multiple of the sampling frequency to be used
T = 1/Fs; %Sampling Period

bpm = 120; %beats per minute
temp = 1/(bpm/60); %tempo

N = temp*Fs1; %Number of samples to achieve desired duration
M = 1; % M is the number of ticks of a note, initial value is 1
n = @(M) 1:M*N; %the array, n, takes an integer multiplier, M, that can reduce or increase the du

%% Define the notes
%Frequencies of notes
 F_C4=261.6;
 F_D4 =293.7;
 F_E4=329.6;
 F_F4=349.2;
 F_Fs4=369.99;
 F_G4=392;
 F_Gs4=415.3;
 F_A4=440; %Frequency of note A is 440 Hz
 F_B4=493.88;
 F_Bb4=466.2;
 F_C5=523.25;
 F_Cs5=554.37;
 F_D5=587.33;
 F_Ds5=622.25;
 F_E5=659.26;
 F_F5=698.46;
 F_Fs5=739.99;

 % Make notes
 C4= @(M) sin(2*pi*F_C4*T*n(M));
 D4= @(M) sin(2*pi*F_D4*T*n(M));
 E4= @(M) sin(2*pi*F_E4*T*n(M));
 F4= @(M) sin(2*pi*F_F4*T*n(M));
 Fs4= @(M) sin(2*pi*F_Fs4*T*n(M));
 G4= @(M) sin(2*pi*F_G4*T*n(M));
 Gs4= @(M) sin(2*pi*F_Gs4*T*n(M));
 A4= @(M) sin(2*pi*F_A4*T*n(M));
 B4= @(M) sin(2*pi*F_B4*T*n(M));
 C5= @(M) sin(2*pi*F_C5*T*n(M));
 Cs5= @(M) sin(2*pi*F_Cs5*T*n(M));
 D5= @(M) sin(2*pi*F_D5*T*n(M));
 Ds5= @(M) sin(2*pi*F_Ds5*T*n(M));
 E5= @(M) sin(2*pi*F_E5*T*n(M));
 F5= @(M) sin(2*pi*F_F5*T*n(M));
 Fs5= @(M) sin(2*pi*F_Fs5*T*n(M));

 %Define Rests
 sr = zeros(1, .05*N); %short rest between any two consequtive notes
 sixtr = zeros(1, .0625*N); %sixteenth rest
 er = zeros(1, .125*N); % eigth rest
 qr = zeros(1, .25*N); % quarter rest
 hr = zeros(1, .5*N); % half rest
 tr = zeros(1, .75*N); % three-quarter rest
 wr = zeros(1, N); % whole rest

 % Define chords
 D = [D4(M) Fs4(M) A4(M)];
 G = [G4(M) B4(M) D5(M)];
 B = [B4(M) Ds5(M) Fs5(M)];
 Em = [E4(M) G4(M) B4(M)];
 A = [A4(M) Cs5(M) E5(M)];

 %% Original piece
 bar1 = [B4(1) sr B4(1) sr C5(1) sr D5(1) sr];
 bar2 = [D5(1) sr C5(1) sr B4(1) sr A4(1) sr];
 bar3 = [G4(1) sr G4(1) sr A4(1) sr B4(1) sr];
 bar4 = [B4(1.5) sr A4(0.5) sr A4(2) sr];
 bar5 = [B4(1) sr B4(1) sr C5(1) sr D5(1) sr];
 bar6 = [D5(1) sr C5(1) sr B4(1) sr A4(1) sr];
 bar7 = [G4(1) sr G4(1) sr A4(1) sr B4(1) sr];
 bar8 = [A4(1.5) sr G4(0.5) sr G4(2) sr];
 bar9 = [A4(1) sr A4(1) sr B4(1) sr G4(1) sr];
 bar10 = [A4(1) sr B4(0.5) sr C5(0.5) sr B4(1) sr G4(1) sr];
 bar11 = [A4(1) sr B4(0.5) sr C5(0.5) sr B4(1) sr A4(1) sr];
 bar12 = [G4(1) sr A4(1) sr D4(1) sr B4(1) sr];
 bar13 = [B4(1) sr B4(1) sr C5(1) sr D5(1) sr];
 bar14 = [D5(1) sr C5(1) sr B4(1) sr A4(1) sr];
 bar15 = [G4(1) sr G4(1) sr A4(1) sr B4(1) sr];
 bar16 = [A4(1.5) sr G4(0.5) sr G4(2)];

 % Make original version
original = [bar1, bar2, bar3, bar4, bar5, bar6, bar7, bar8, bar9, bar10, bar11, bar12, bar13, bar
%sound(original,Fs);   %--- Remove %-sign to hear the original version%
%wavwrite(original,'original.wav'); %--- Remove %-sign to make .wav file of
                                     % of the original version%

%% Improvisation

 chords = ['D '; 'G '; 'D '; 'G '; 'D '; 'B '; 'Em'; 'A '; 'D '; 'D '];
 %% Start improv
improv = [];

firstdur = [0.2 0.3 0.2 0.3]; %Vector defining probabilities to go to a sixteenth note; eigth not
r0 = rand;
if r0 < firstdur(1,1);
    L = 0.25;
elseif r0 < firstdur(1,1) + firstdur(1,2);
    L = 0.5;
elseif r0 < firstdur(1,1) + firstdur(1,2) + firstdur(1,3);
    L = 0.75;
else L = 1;
end

for k = 1:6
    current_chord = chords(k,:);
 ticks = 0;
    while ticks < 2 %There can only by 4 'ticks' in a bar, 1 tick equals a quarternote

% duration
 tmdur = [0 0 1/2 1/2;0 5/20 0 15/20;0 1/3 1/3 1/3;0 8/20 0 12/20];

 % create the vector that will be multiplied with the transition matrix,
 % such that only the row of the current note will be left
 multiplier1 = zeros(1,4);

if L == 0.25;
    multiplier1(1,1) = 1;
 elseif L == 0.5;
    multiplier1(1,2) = 1;
 elseif L == 0.75;
    multiplier1(1,3) = 1;
 elseif L == 1;
    multiplier1(1,4) = 1;
end

 durvector = multiplier1*tmdur;

 % decide which duration will come next, by making sure a random variable r
 % will fall within the probabilities, but also by making sure the number
 % of ticks inside a bar cannot exceed 2 (with 0.5 as a last resort)
 r1 = rand;

 if r1 < durvector(1,1);
           new_dur = 0.25;
 elseif r1 < (durvector(1,1) + durvector(1,2)) && ticks <= 1.5;
           new_dur = 0.5;
 elseif r1 < (durvector(1,1) + durvector(1,2) + durvector(1,3)) && ticks <=1.25;
           new_dur = 0.75;
```

```
elseif r1 < (durvector(1,1) + durvector(1,2) + durvector(1,3) + durvector(1,4)) && ticks <= 1
           new_dur = 1;
else new_dur = 2 - ticks;
end

L = new_dur;


% pitch
multiplier2 = zeros(1,3);

if strcmp(current_chord,'D ');
     current_pitch = D4(L);

     tmpitch = [4/14 1/14 4/14 1/14 4/14;4/14 1/14 2/14 1/14 6/14;3/14 1/14 3/14 1/14 6/14];
    % create the vector that will be multiplied with the transition matrix,
    % such that only the row of the current note will be left

   if current_pitch == D4(L);
           multiplier2(1,1) = 1;
       elseif current_pitch == E4(L);
           multiplier2(1,2) = 1;
       elseif current_pitch == Fs4(L);
           multiplier2(1,3) = 1;
       elseif current_pitch == G4(L);
           multiplier2(1,4) = 1;
       elseif current_pitch == A4(L);
           multiplier2(1,5) = 1;
   end

   notevector = multiplier2 * tmpitch;

   % Decide which variable will be next
   r2 = rand;

   if r2 < notevector(1,1);
           new_pitch = D4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2));
           new_pitch = E4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
           new_pitch = Fs4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
           new_pitch = G4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
           new_pitch = A4(L);
   end


elseif strcmp(current_chord,'G ');
```

```
    current_pitch = G4(L);

      tmpitch = [4/14 1/14 4/14 1/14 4/14;4/14 1/14 2/14 1/14 6/14;3/14 1/14 3/14 1/14 6/14];

    if current_pitch == G4(L);
            multiplier2(1,1) = 1;
        elseif current_pitch == A4(L);
            multiplier2(1,2) = 1;
        elseif current_pitch == B4(L);
            multiplier2(1,3) = 1;
        elseif current_pitch == C5(L);
            multiplier2(1,4) = 1;
        elseif current_pitch == D5(L);
            multiplier2(1,5) = 1;
    end

    notevector = multiplier2 * tmpitch;

    % Decide which variable will be next
    r2 = rand;

    if r2 < notevector(1,1);
            new_pitch = G4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2));
            new_pitch = A4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
            new_pitch = B4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
            new_pitch = C5(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
            new_pitch = D5(L);
    end

 elseif strcmp(current_chord,'B ');
      current_pitch = B4(L);

      tmpitch = [4/14 1/14 4/14 1/14 4/14;4/14 1/14 2/14 1/14 6/14;3/14 1/14 3/14 1/14 6/14];

    if current_pitch == B4(L);
            multiplier2(1,1) = 1;
        elseif current_pitch == Cs5(L);
            multiplier2(1,2) = 1;
        elseif current_pitch == Ds5(L);
            multiplier2(1,3) = 1;
        elseif current_pitch == E5(L);
            multiplier2(1,4) = 1;
        elseif current_pitch == Fs5(L);
            multiplier2(1,5) = 1;
    end
    notevector = multiplier2 * tmpitch;

    % Decide which variable will be next
    r2 = rand;

    if r2 < notevector(1,1);
            new_pitch = B4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2));
            new_pitch = Cs5(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
            new_pitch = Ds5(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
            new_pitch = E5(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
            new_pitch = Fs5(L);
    end

 elseif strcmp(current_chord,'Em');
      current_pitch = E4(L);

      tmpitch = [4/14 1/14 4/14 1/14 4/14;4/14 1/14 2/14 1/14 6/14;3/14 1/14 3/14 1/14 6/14];

    if current_pitch == E4(L);
            multiplier2(1,1) = 1;
        elseif current_pitch == Fs4(L);
            multiplier2(1,2) = 1;
        elseif current_pitch == G4(L);
            multiplier2(1,3) = 1;
        elseif current_pitch == A4(L);
            multiplier2(1,4) = 1;
        elseif current_pitch == B4(L);
            multiplier2(1,5) = 1;
    end

    notevector = multiplier2 * tmpitch;

    % Decide which variable will be next
    r2 = rand;

    if r2 < notevector(1,1);
            new_pitch = E4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2));
            new_pitch = Fs4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
            new_pitch = G4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
            new_pitch = A4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
            new_pitch = B4(L);
    end

 elseif strcmp(current_chord,'A ');
     current_pitch = A4(L);

     tmpitch = [4/14 1/14 4/14 1/14 4/14;4/14 1/14 2/14 1/14 6/14;3/14 1/14 3/14 1/14 6/14];

    if current_pitch == A4(L);
            multiplier2(1,1) = 1;
        elseif current_pitch == B4(L);
            multiplier2(1,2) = 1;
        elseif current_pitch == Cs5(L);
            multiplier2(1,3) = 1;
        elseif current_pitch == D5(L);
            multiplier2(1,4) = 1;
        elseif current_pitch == E5(L);
            multiplier2(1,5) = 1;
    end

    notevector = multiplier2 * tmpitch;

    % Decide which variable will be next
    r2 = rand;

    if r2 < notevector(1,1);
            new_pitch = A4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2));
            new_pitch = B4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
            new_pitch = Cs5(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
            new_pitch = D5(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
            new_pitch = E5(L);
    end
 end

%create the improvisation
improv = [improv,new_pitch,sr]; %the new note will be put in a matrix
current_pitch = new_pitch;    %the new note will become the next current note
ticks = ticks + L;
    end
end

for k = 7:10;
    current_chord = chords(k,:);
 ticks = 0;
    while ticks < 1 %There can only by 4 'ticks' in a bar, 1 tick equals a quarternote
```

```
% duration
 tmdur = [0 0 0.75 0.25;0 2/6 0 4/6;0 1/2 0 1/2;0 1/6 0 5/6];

 % create the vector that will be multiplied with the transition matrix,
 % such that only the row of the current note will be left
 multiplier1 = zeros(1,4);

if L == 0.25;
    multiplier1(1,1) = 1;
 elseif L == 0.5;
    multiplier1(1,2) = 1;
 elseif L == 0.75;
    multiplier1(1,3) = 1;
 elseif L == 1;
    multiplier1(1,4) = 1;
end

 durvector = multiplier1*tmdur;

 % decide which duration will come next, by making sure a random variable r
 % will fall within the probabilities, but also by making sure the number
 % of ticks inside a bar cannot exceed 1
 r1 = rand;

    if r1 < durvector(1,1);
            new_dur = 0.25;
        elseif r1 < (durvector(1,1) + durvector(1,2)) && ticks <= 1.5;
            new_dur = 0.5;
        elseif r1 < (durvector(1,1) + durvector(1,2) + durvector(1,3)) && ticks <=1.25;
            new_dur = 0.75;
        elseif r1 < (durvector(1,1) + durvector(1,2) + durvector(1,3) + durvector(1,4)) && ticks
            new_dur = 1;
        else new_dur = 1 - ticks;
    end

 L = new_dur;

 % pitch
 multiplier2 = zeros(1,3);

if strcmp(current_chord,'D ');
    current_pitch = D4(L);

    tmpitch = [4/10 1/10 2/10 1/10 2/10;4/10 1/10 2/10 1/10 2/10;4/10 1/10 2/10 1/10 2/10];
     % create the vector that will be multiplied with the transition matrix,
     % such that only the row of the current note will be left

    if current_pitch == D4(L);
```

```
            multiplier2(1,1) = 1;
        elseif current_pitch == E4(L);
            multiplier2(1,2) = 1;
        elseif current_pitch == Fs4(L);
            multiplier2(1,3) = 1;
        elseif current_pitch == G4(L);
            multiplier2(1,4) = 1;
        elseif current_pitch == A4(L);
            multiplier2(1,5) = 1;
    end

    notevector = multiplier2 * tmpitch;

    % Decide which variable will be next
    r2 = rand;

    if r2 < notevector(1,1);
            new_pitch = D4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2));
            new_pitch = E4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
            new_pitch = Fs4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
            new_pitch = G4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
            new_pitch = A4(L);
    end

 elseif strcmp(current_chord,'G ');
    current_pitch = G4(L);

    tmpitch = [2/10 1/10 4/10 1/10 2/10;2/10 1/10 4/10 1/10 2/10;2/10 1/10 4/10 1/10 2/10];

    if current_pitch == G4(L);
            multiplier2(1,1) = 1;
        elseif current_pitch == A4(L);
            multiplier2(1,2) = 1;
        elseif current_pitch == B4(L);
            multiplier2(1,3) = 1;
        elseif current_pitch == C5(L);
            multiplier2(1,4) = 1;
        elseif current_pitch == D5(L);
            multiplier2(1,5) = 1;
    end

    notevector = multiplier2 * tmpitch;

    % Decide which variable will be next
    r2 = rand;
```

```
   if r2 < notevector(1,1);
           new_pitch = G4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2));
           new_pitch = A4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
           new_pitch = B4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
           new_pitch = C5(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
           new_pitch = D5(L);
   end

elseif strcmp(current_chord,'B ');
     current_pitch = B4(L);
           tmpitch = [4/10 1/10 2/10 1/10 2/10;4/10 1/10 2/10 1/10 2/10;4/10 1/10 2/10 1/10 2/10

   if current_pitch == B4(L);
           multiplier2(1,1) = 1;
       elseif current_pitch == Cs5(L);
           multiplier2(1,2) = 1;
       elseif current_pitch == Ds5(L);
           multiplier2(1,3) = 1;
       elseif current_pitch == E5(L);
           multiplier2(1,4) = 1;
       elseif current_pitch == Fs5(L);
           multiplier2(1,5) = 1;
   end

   notevector = multiplier2 * tmpitch;

   % Decide which variable will be next
   r2 = rand;

   if r2 < notevector(1,1);
           new_pitch = B4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2));
           new_pitch = Cs5(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
           new_pitch = Ds5(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
           new_pitch = E5(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
           new_pitch = Fs5(L);
   end

elseif strcmp(current_chord,'Em');
     current_pitch = E4(L);
     tmpitch = [2/10 1/10 4/10 1/10 2/10;2/10 1/10 4/10 1/10 2/10;2/10 1/10 4/10 1/10 2/10];

   if current_pitch == E4(L);
           multiplier2(1,1) = 1;
       elseif current_pitch == Fs4(L);
           multiplier2(1,2) = 1;
       elseif current_pitch == G4(L);
           multiplier2(1,3) = 1;
       elseif current_pitch == A4(L);
           multiplier2(1,4) = 1;
       elseif current_pitch == B4(L);
           multiplier2(1,5) = 1;
   end

   notevector = multiplier2 * tmpitch;

   % Decide which variable will be next
   r2 = rand;

   if r2 < notevector(1,1);
           new_pitch = E4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2));
           new_pitch = Fs4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
           new_pitch = G4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
           new_pitch = A4(L);
   elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
           new_pitch = B4(L);
   end

 elseif strcmp(current_chord,'A ');
     current_pitch = A4(L);

     tmpitch = [4/10 1/10 2/10 1/10 2/10;4/10 1/10 2/10 1/10 2/10;4/10 1/10 2/10 1/10 2/10];

   if current_pitch == A4(L);
           multiplier2(1,1) = 1;
       elseif current_pitch == B4(L);
           multiplier2(1,2) = 1;
       elseif current_pitch == Cs5(L);
           multiplier2(1,3) = 1;
       elseif current_pitch == D5(L);
           multiplier2(1,4) = 1;
       elseif current_pitch == E5(L);
           multiplier2(1,5) = 1;
   end

   notevector = multiplier2 * tmpitch;
```

```
    % Decide which variable will be next
    r2 = rand;

    if r2 < notevector(1,1);
            new_pitch = A4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2));
            new_pitch = B4(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3));
            new_pitch = Cs5(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4));
            new_pitch = D5(L);
    elseif r2 < (notevector(1,1)+notevector(1,2)+notevector(1,3) + notevector(1,4) + notevector(1
            new_pitch = E5(L);
    end
end

improv = [improv,new_pitch,sr]; %the new note will be put in a matrix
current_pitch = new_pitch;    %the new note will become the next current note
ticks = ticks + L;
    end
end

originalpart = [bar9 bar10 bar11 bar12];
%sound(originalpart,Fs); % uncomment if you want to hear he original
                         %version and the improvisation at the same time (does not sound very nic
%sound(improv,Fs);
%wavwrite(improv,'improv.wav')

%% Make musical piece with improvisation in one line
sound([bar1 bar2 bar3 bar4 bar5 bar6 bar7 bar8 improv bar13 bar14 bar15 bar16],Fs);
```