

ABSTRACT

I undertook this interdisciplinary project during my undergraduate school years based on my developed skills in computer science and classical music. This work not only gives me the chance to express my combined passions within these subjects, but also applies, synthesizes, and deepens general understanding of baroque music and computer science. In particular, the project requires an understanding of music theory (contrapuntal music composition) along with the science of sound (frequencies, intervals, and harmonics) - and to use a developed ability in programming to write an algorithm that produces and plays musical excerpts that stylistically mimic that of J.S. Bach. The algorithm written in this project can then be served as a tool in research for additional studies as well as an application to provide inspiration for the composition of neo-baroque music. The logic of this algorithm provides a clear formulation of the rules for counterpoint, while the generation of notes in proper intervals illustrates and makes audible the mathematics of frequencies, chords, harmonic and non-harmonic tones. Analysis of the resulting compositions requires the listeners to consider aesthetic elements of composition and other more original compositional styles that they themselves might be able to implement.

1. GOALS OF PROJECT

My project is an interdisciplinary one, synthesizing concepts of music, physical science and computer science through algorithmic musical composition. Thus, it is suitable for an individual like myself who has a background in music theory, digital audio and MIDI, and computer programming. The project could also be undertaken by an interdisciplinary team who work to combine their various backgrounds.

The overall goal of this project is to write a computer program that composes properly constructed, musically interesting, and aesthetically pleasing pieces of contrapuntal music. To achieve this goal, it is necessary to combine a knowledge of music theory, the physical science of sound, and the mathematics of frequencies and harmonics - all brought to bear in the development of an algorithm and its implementation in a particular programming language. Thus, this project not only has the potential of analysis and application, but also has pedagogical value both for the persons undertaking the challenge and for those examining the results, as the project represents an active learning method for applying and synthesizing concepts and thereby clarifying them.

2. APPROACH

In order to illustrate the science of sound in a fundamental way, we chose to implement the algorithm in MATLAB, generating notes at a low level of abstraction by evaluating sine functions at a given frequency.

This algorithm implements the rules for contrapuntal music composition, as described in more detail below. The musical excerpts produced by the algorithm consist of two lines (sometimes referred to as voices) of notes. At each step in the algorithm, the next note of each line is determined, constrained by the rules of contrapuntal style, but randomized within those constraints, and further tuned to obey suggestions of aestheticism towards musical elements. The notes are generated by sine function evaluation and stored in a list of sound sample values that can be played in MATLAB once the piece is fully constructed. Non-harmonic tones are then added to ornament the piece and to create a stylistic effect.

Because of the partially-random nature of note generation, the algorithm composes different pieces from different runs, some more aesthetically pleasing than others. The compositions created by the algorithm are evaluated in two ways. First, we listen to the composition as the program runs. Second, we output the compositions in both digital audio and MIDI format. The MIDI format can be read by a MIDI sequencer (such as MuseScore) and displayed onto a musical score. Using functions in MATLAB's MIDI toolbox, it is possible to view graphs of our composition's melodic contours, interval distribution, pitch distribution, timing, and so forth [1].

As the algorithm was developed, I listened to its compositions, examined them on the score, and evaluated for their correctness and aesthetic properties based on professional knowledge and comparative analysis on previous excerpts. I also used the musical scores from J.S. Bach that I am familiar with as reference to further critique and improve the music produced, on both euphonic and harmonic standards.

The evaluation process helps in the algorithmic development, as it revealed contrapuntal features that had been overlooked or aesthetic elements that could be improved. Listening to the wide variety of pieces composed by the algorithm also has pedagogic value. First, I need evaluate each piece and verify that it has proper contrapuntal structure, solidifying my understanding of this style. Additionally, I am given the opportunity to consider elements that have more aesthetic value in an overall composition, and thus try to articulate precisely what musical properties are more desirable than others.

It is intended that the initial implementation of this project will serve as a platform on which to build more complex and original compositions. Having gained an understanding of algorithmic music composition from this project, the implementer (or reader of this paper) can then go on to develop algorithms in a more personal compositional style, encoding his or her own musical choices and stylist interpretations.

1
3. BACKGROUND

3.1 Basic Concepts

For us humans, sound results from changes in air pressure amplitude (vibrations) that meet our ears and are interpreted by our auditory system. In music, we divide sound into notes – single pitches, as they are called. The pitch of a note – how high or low it is in human hearing – is related to the note's frequency. Frequency is defined as the number of cycle the sound's vibration goes through in a second, measured in Hertz, which is cycles per second (Hertz, abbreviated Hz). A pure-pitch note – for example, the note A4 on an 88-key piano keyboard – can be represented by a single frequency, chosen standardly to be 440 Hz.

Notes on a piano keyboard are lettered from A to G, cycling back to A. The frequency of middle C – the fourth key on an 88-key keyboard and thus called C4 – is approximately 261.63 Hz. In the music of Western culture, instruments are generally tuned relative to the note A4 set to 440 Hz.

3.2 The Mathematical and Algorithmic Nature of Music

From the time of ancient civilizations, in cultures ranging from Mesopotamia to Egypt to China to Greece, the mathematical and algorithmic properties of music have been recognized. Music inherently seems to have a numerical basis, and the process of composing music can readily be conceived as the application of a rule-based procedure.

As early as the 5th century BCE, Greek mathematician Pythagoras analyzed the mathematical relationships between musical intervals by considering the vibrations of strings and the pitches of sound they create. Based on its length, how it is stretched, and how it is plucked, a string vibrates at a fundamental frequency and emits a corresponding pitch of sound. Let's say that we have a string vibrating at 220 cycles per second (220 Hz), which is the note A3. A4, an octave above, sounds "the same" only higher, and there is a simple mathematical relationship between the two notes' frequencies: the frequency of the higher-octave note is two times the frequency of the note an octave lower. Thus, if A3 has a frequency of 220, A4 has a frequency of 440 Hz.

An interval – which is a fundamental building block of musical compositions – is the distance between two notes. Pythagoras observed that intervals can be defined by ratios, and the most consonant intervals – the ones that sound "the best" together – are definable by ratios of integers. This can be illustrated by a vibrating string. The vibrating string has a fundamental frequency that results from its length, its tightness, and how hard it is plucked. Other frequencies can be generated by holding the string down at different positions. If you press down at a point that is 2/3 the length of a string, you generate a pitch called a fifth relative to the string's fundamental frequency. Since the length of the segment is 2/3 of the original string's length, its frequency will be 3/2 of the original. (The shorter the wavelength the higher the frequency – a reciprocal relationship.) If you press a point that is 1/4 of the length of the string, you generate a pitch that is called a fourth relative to the original. Since the length of the segment is 3/4 the original, a fourth has a frequency that is 4/3 the original. Fourths and fifths are expressible as ratios of integers, and they are considered consonant intervals. (These are examples of just intervals, as opposed to equal-tempered ones.)

Intervals are in fact ratios of the harmonic overtones of a fundamental frequency. Given a fundamental frequency, f, the harmonic frequencies are the integer multiples of f: 1f, 2f, 3f, .... The ratio of the second harmonic to the first is and 2f/1f = 2/1, the octave. The ratio of the third harmonic to the second is 3f/2f = 3/2, a fifth. This is illustrated in the figure below.

Many more examples of the mathematical basis of sound and music could be given. Suffice to say that numbers permeate music theory [3] [4].

The potential of representing music composition algorithmically also has a long history [5] [6]. Algorithmic music composition is the creation of a piece of music by means of a set of rules or instructions. The musical canons that evolved from the 9th through the 15th century are some of the best early examples of algorithmic composition. A canon begins with a musical strand, called a voice, to which an additional voice is added. A canon's basic rule is simple, requiring that the second voice be modeled after the first, but typically offset by a certain amount of time. (Think of a round like "Frère Jacque"). A canon uses a basic melodic theme in both voices, with some inversion, augmentation, key change, and melodic variation. Rules of contrapuntal style govern the relationships in the horizontal (melodic) and vertical (harmonic) structure of a canon. The rules of counterpoint will be described in section 3.3.

Similar to canons in their use of contrapuntal form, the fugues of the Baroque period can be described by sets of rules. The fugue is generally longer and more complex than the canon, having more voices (bass, tenor, alto, and soprano) and allowing for more variations of the melodic theme, which appears in the different voices throughout the length of the piece. Bach, Mozart, and Beethoven were masters of this musical form, exemplified in Bach's Prelude and Fugue in A Minor, Mozart's Kyrie Eleison Requiem in D Minor, and Beethoven's "Grosse Fuge." Bach's "The Art of the Fugue" serves as an excellent pedagogical tool for those wishing to master fugal form.

In the modern era, fugal forms have continued in the works of Bartók and Shostakovich, and new algorithmic forms have
appeared. In the 1940s, in the 12-tone works of Arnold Schoenberg and his pupil Anton von Webern, \textit{serialism} was developed using a matrix-based specification of musical parameters – pitch, rhythm, dynamics, and other matters of form. Variations were produced by permuting a tone row in the matrix based on transposition, inversion, retrograde, or retrograde inversion – all algorithmic procedures.

Other examples of modern algorithmic composition can be found in the work of John Cage, who utilized randomness in many of his compositions. In \textit{Reunion}, choices made during a chess game on a photo-receptor equipped chessboard resulted in a musical composition – thus producing a different piece each time a game is played. In Cage's \textit{Atlas Eclipticalis} music is composed by laying score paper on top of astronomical charts and placing notes on the score corresponding to the positions of stars.

Since the advent of computers, the rules and instructions that constitute an \textit{algorithmic music composer} generally take the form of a computer program. Early on in the history of computers – just when Charles Babbage's Analytical Engine was begin designed – one of its first programmers, Ada Lovelace, recognized the computer's potential for algorithm music composition.

[The Analytical Engine] might act upon other things besides number, were objects found [i.e., if it were possible that objects could be found] whose mutual fundamental relations could be expressed by those of the abstract science of operations [i.e., algorithmically], and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine...Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent [7].

One of the first electronic computers built – the ILLIAC at the University of Illinois – was employed by programmers Lejaren Hiller and Leonard Isaacson in 1957 to algorithmically generate "The ILLIAC Suite" scored for string quartet. This was just one of the first in a long history of music compositions produced by means of computers.

With algorithmic music composition implemented in the past, one might say that the program is doing the composing. But on closer consideration, it becomes clear that the real creative part of the process lies in the way the human composer writes the algorithm. Two basic strategies are employed: deterministic and stochastic. In a \textit{deterministic algorithm}, only one composition can result from a given input. In a \textit{stochastic algorithm}, choices are made for notes, harmonies, timing, and so forth either randomly or by a probability distribution. Thus, different compositions are created from different executions of the program. In addition to the two basic methods for algorithmic composition, compositional models have been adapted from computer science theory, physics, biology, statistics, and artificial intelligence – including Markov chains [8], genetic algorithms [9], neural networks and connectionism [10] [11], fractals [12], cellular automata, formal grammars, chaos theory, and Brownian motion, and more [13] [14].

3.3 The Structure of Contrapuntal Music

For this project, our goal is to implement a program that writes music in contrapuntal style, since this form is so fundamentally algorithmic. Contrapuntal music, also called \textit{counterpoint}, is a structure of musical composition that originated in the Renaissance and was developed extensively in Baroque and early Classical music, particularly in canons and fugues written by Bach and Mozart. Many species of counterpoint exist, but we will focus here on the fundamental rules common to all of them [15].

Counterpoint consists of two or more \textit{lines} of musical notes, also called \textit{voices}. Their rhythmic, harmonic, and melodic relationships are what give interest, complexity, and beauty to the piece. Each line provides its own melody, but the melodies line up vertically in intervals that follow a prescribed pattern. That is, the notes that line up vertically are chosen from a particular chord such that the chords follow the diagram shown below. For my algorithm, I will be standardizing its output to generate two lines of musical note, or two voices, in result of the algorithmic complexity and creating manageable guideline to follow. (It is assumed for this section that the reader knows basic music theory. For an introduction to the fundamentals, we refer the reader to Chapter 3 of \textit{Digital Sound and Music} (Burg, Romney, and Schwartz 2016).)

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2a.png}
\caption{Figure 2a}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2b.png}
\caption{Figure 2b}
\end{figure}

Rules for chord progressions in contrapuntal music in both major (Figure 2a) and minor keys (Figure 2b)

Each Roman numeral in the diagram represents a chord. Since major and minor keys in a piece of music have different chord structures, I have attached two figures above to present the details of these differences. The rules for both are very similar, only with slight alterations in pattern, but mainly differing in the harmonic typing of the chord. The arrows show the permitted chord sequences each designated chord can go to. I will now continue to give an example of this process in the major key. For simplification purposes, the piece will always begin on the I chord, the root chord of the chosen key. It is alright to generalize the starting chord as I, as most pieces harmonically do begin like so. The up arrow after chord I indicates that any chord, not including itself, may be used after I. This implies a choice, which can be made randomly in a computer program. Let's say that chord iii is selected next. The fact that there is an arrow between iii and vi as well as an arrow between iii and the box containing IV indicates that either choice could be made. Let's say that the next step is randomly determined to be the box with IV and ii in it. Because both chords are in the box, again the system must make a random choice between the two. If IV is selected, it can go to ii, I, or the box with V and vii°. If ii is selected, it can go to I, or the box with V and vii°. As counterpoint composition progresses, the notes for the two voices are selected from the notes in the chosen chord. For example,
in the key of C major, if the next chord is I, the notes C and E, E and C, C and G, G and C, E and G, or G and E could be chosen (order relating to which line gets which note.) Thus, while there are rules, there is also a great deal of choice within these rules.

Additional constraints in contrapuntal composition emerge from the concepts of consonance, dissonance, and motion. In Western music, the notion of consonance is loosely based on the human perception of pairs of notes that are harmonic and "sound good" together. This sensory phenomenon is related to the intervals between the notes and the way in which the cycles of their respective sound frequencies "match up well" over time. Intervals are classified below.

- **perfect consonances**: unison, octave, fourth, and fifth (A fourth is considered dissonant in some contexts.)
- **imperfect consonances**: major or minor third, major or minor sixth
- **dissonances**: major or minor second, augmented fourth, diminished fifth, and major or minor seventh.

Motion relates to the composition as it proceeds in the horizontal direction. That is, motion results from changes in pitch of the two voices. Each step of the two voices from one note to the next can be identified as one of these types of motion:

- **direct motion**: The voices move in the same direction in pitch (up or down). **Parallel motion** is a type of direct motion in which the voices move in the same direction by the same amount.
- **contrary motion**: One voice moves up while the other moves down or vise versa. This definition is not constrained to any specific intervals.
- **oblique motion**: One voice moves up or down while the other stays the same.

In contrapuntal composition, the type of motion that is permitted from one note to the next is captured in the following rule: If the chord progression goes from perfect to perfect consonances, direct motion is not allowed. For better musicality, contrary motion is preferred over direct or oblique.

4. **RELATED RESEARCH**

Canons, fugues, counterpoint, and the compositions of Bach – whose structures are guided by certain music theoretic rules – have a clear mathematical and algorithmic nature. For this reason, they are among the first types of music that computer programmers have attempted to compose algorithmically. A rule-based expert system has been written for harmonizing Bach four-part chorales [16]. A number of approaches adapted from artificial intelligence have been applied to fugue and contrapuntal composition, including genetic algorithms [17], neural networks and connectionism [11], machine learning [18] [19], probabilistic logic [20], and neighborhood search [21]. Stochastic methods naturally play a part in contrapuntal composition and analysis [22]. Methods have also been devised for evaluating the artistic merit of random choices in algorithmic music composition [23].

My work differs from previous work in three ways. First, the emphasis in my project is on the research and application value of the implementation and its analysis. The project serves as an example of interdisciplinary work that integrates and applies basic knowledge that I have acquired in music theory and computer science. Secondly, my algorithm is implemented in MATLAB with three types of output: digital audio, MIDI, and score sheet. Thirdly, I analyze my results using the MATLAB MIDI toolbox for the purpose of characterizing the shapes and structures of aesthetically pleasing contrapuntal compositions. The figure below is an example of the type of graphical analysis that can be done with the MIDI toolbox.

![Figure 3 Graph from MATLAB MIDI toolbox (from the MIDI toolbox user's manual)](image.png)

5. **IMPLEMENTATION**

6. **The Algorithm**

The foundation of this algorithm is based on the random number generator, probability, and defined rules. Starting the algorithm with the assumption of chord I, the algorithm implements the following chord progressions seen in counterpoint (Figure 2) by using a random number generator to choose from its choices. As there are only two voices, the choice of which two notes is best to be used and where it might land on the range of the piano is made by both probability and rule definition.

This preference is represented in my algorithm by the use of probability. Within each progression of chords, there is a two out of three chance that contrary motion is to be selected over oblique motion. In counterpoint, there is also a suggested addition of musicality when it comes to minimizing range of movement within voices. Though the intervals of the two voices can be large or small, the movement is recommended to be restricted in most general cases. This is an interesting algorithmic effect to pan out through running code as it raises the question of which direction or note is better in being picked. That being said, this is a recommendation and not a rule to be followed, meaning that having greater intervalic jumps in melody could be a positive or negative choice. Some choices that are deemed not to be made are parallel fourths and fifths. Therefore, the computer uses a nested set of probability and restriction to determine possible choices of intervals.

After the algorithm figures out which precise notes to go to, it jumps to figure the need for which nonharmonic tones to use or even a need for them.
As for nonharmonic tones (passing tone, neighbor tone, suspension), there is a general algorithm in place where some are preferred over others. For instance, it is not preferred for there to be 16th notes in both voices playing consecutively. An example of this occurrence is illustrated in the figure above. The baroque style of music was originally played on a harpsichord, so we would want to limit the ‘heaviness’ of its notes. Thus, I have made a rule within my algorithm, where only one hand would play 16th notes at a time. The art of counterpoint supports the featuring of two individual voices and the aesthetics in them coming together yet maintaining individuality. The rule for producing general nonharmonic tones in the algorithm is based on probability. In the primary case, there is a 1 out of 18 chance that no nonharmonic tones will be used in the chord. Otherwise, the algorithm uses a random number generator in the depicted scenario, and chooses one out of the possible nonharmonic tones that could be produced within that interval. For instance, if we were trying to produce nonharmonic tones between the notes C4 and E4, there would be four different possible cases: C4 remains as a quarter note (no nonharmonic tone), D4 is added in to produce the sequence of C4, D4, E4 eighth notes (passing tone), a sequence of C4, D4, E4, F4, E4 sixteenth notes, or a sequence of C4, B3, C4, D4, E4 sixteenth notes. With individually designed probabilities that are estimated based on my musical experience, the algorithm puts together the preferences and creates a analytical musical piece. As much of the composition is based on probability and defined rules, the result of the algorithm is almost always unexpected.

Once all of these details are defined, the algorithm saves the notes, loops, and continues to define these characteristics with newly generated chord sequences. Once the desired number of measures is completed, the program finishes off the excerpt with the final cadence (either perfect authentic cadence, half cadence, or a simple resolution). The problem with a variety of cadence choices is solved with a defined user input prior to the execution of the program. This is a straightforward process that ensures the preferred conclusion of the excerpt, but does not take into account the musical elements in play during the generation of the piece. What may be your chosen cadence type may end up not being your top choice when the general muscality of the piece comes into play.

Lastly, the program finishes off by saving the MIDI file. The user is then able to replay the audio through this file, or if wanted, view what was played on a computer generated music score sheet.

7. Note Generation

In Section 3.2, I noted that sound results from changes in air pressure amplitude (vibrations) that meet our ears and are interpreted by our auditory system. In order to digitize sound so that it can be represented in a way a computer can handle, it is necessary to sample the air pressure amplitude of the sound at evenly-spaced moments in time. This is what a microphone does when we record sound to be stored on a computer. If we record a single-frequency sound and graph the values, we get something in the shape of a sine function.

This leads us to the observation that to generate a single-frequency sound in a computer program, I can use a sine function evaluated over a list of inputs that represent evenly-spaced points in time. For what is called CD-quality music, take 44100 samples per second (also called Hertz), which means the input to the sine function consists of 44100 points in time between 0 and 1 second. Evaluating the sine function for the given frequency results in values that rise and fall regularly between −1 and 1, as graphed below. When used to represent sound, the x-axis represents time and the y-axis represents air pressure amplitude.

In a computer program, I can use sine functions to get lists of numbers that represent sound amplitudes over time (called sound samples), and when I send such a list to the appropriate sound player on a computer, I can hear the note I had asked for. These functions are easily handled in MATLAB. For example,:

\[
t = \text{linspace}(0,1,44100);
\]

generates a list \( t \) of 44100 numbers between 0 and 1 that constitute the inputs evenly divided across one second in the time domain. The statement

\[
a = \sin(2\pi t*440*440);
\]

evaluates a sine function for every point in time in the list \( t \), generating a list \( a \) of audio sample values (air pressure amplitudes) that can be played as sound with MATLAB’s sound function.

\[
\text{sound}(a, 44100);
\]

Since we specified a frequency of 440, the note is A4.

Using single frequency notes in the music composition, however, results in a rather uninteresting piece, with none of the harmonic components that are found in single notes played by violins, pianos, clarinets, and other music instruments. Thus, an addition to my original implementation was the ability to create notes of different timbres, beginning with basic non-sinusoidal waveforms like sawtooth, triangle, and square waves. Non-sinusoidal waveforms are built from a fundamental frequency to which is added some integer multiples of this frequency, called harmonics. I have implemented MATLAB functions for these waveforms and included them as an option in the creation of notes. These functions are included in the Appendix of the paper.
8. WAV and MIDI Output

Creating individual notes by generating lists of sound samples illustrates the nature of digital audio. It is possible to play the piece directly in MATLAB by sending the audio samples to the sound output with the `sound` function. Also, the samples can be written out as an uncompressed WAV file and read into a digital audio processing program like Adobe Audition, Cakewalk Sonar, or Apple Logic. However, for musicians, it's good to be able to see the piece transcribed to a musical score. Saving the piece in a MIDI file format makes this possible.

MIDI (Musical Instrument Digital Interface) is a method of representing each note in a short numeric encoding, with middle C given the number 60. The velocity with which a note is to be struck and the onset time of the note is also encoded in a MIDI file. A MIDI sequencer (available, for example, in Cakewalk Sonar, Apple Logic, or Propellerhead Reason) is a piece of software that can read a properly formatted MIDI file, display the notes on a musical score, and play the notes using whatever instrument is chosen by the user. The sequencer has access to a MIDI synthesizer or sampler which takes the encoded note and turns it into digital audio to be played by the sound card.

Thus, to make the musical compositions are easy to view and analyze with regard to contrapuntal structure, I added MIDI output to my MATLAB program. I did this by means of a MIDI toolbox that includes a `writemidi(inputMatrix, nameOfMIDIFile)` function. The `writemidi` function requires as input an n row by 7 column input matrix where n is the number of notes and the 7 columns give the parameters for each note, as shown in the figure below.

<table>
<thead>
<tr>
<th>onset in beats</th>
<th>duration in beats</th>
<th>MIDI channel</th>
<th>MIDI pitch</th>
<th>velocity</th>
<th>onset</th>
<th>duration</th>
</tr>
</thead>
</table>

The `writemidi` function outputs a properly formatted MIDI file of the given name. I then could input this file into a program like Finale and look at the contrapuntal composition on a musical score. It is not difficult to generate this information in the MATLAB program at the same time that I am generating the digital audio samples. A sketch of how this is done is given in the `createMidiOutput` function in the Appendix.

9. ANALYSIS OF RESULTS

10. FUTURE WORK

11. REFERENCES


12. Appendix

```matlab
function y = squareWave( baseF, numComponents, amp, sampRate, numSecs, numSamps )
%This function produces a square wave, which is composed of the integer multiples of the wave's
%fundamental frequency.
%Arguments:
%squareWave(Base Frequency, # of Components, Amplitude, Sampling Rate, # Seconds,
# Samples);
%Sample function call from command line:
%(440, y = squareWave 20, 0.9, 44100, 2, 88200);
t = linspace(0, numSecs, numSamps);
y = zeros(1, numSamps);
for i = 1:numComponents
    y = y + amp * sin(2*pi*baseF*(2*i-1)*t)/(2*i-1);
end
y = amp * y;
sound(y, sampRate);
pause(0.5);
end

function y = sawtoothWave( baseF, numComponents, amp, sampRate, numSecs, numSamps)
%This function produces a square wave, which is
%composed of the integer multiples of the wave's
%fundamental frequency.
%Arguments:
%sawtoothWave(Base Frequency, # of Components, %Amplitude, Sampling Rate, # of Seconds,
%Number of Samples);
%Sample function call from command line:
%y = sawtoothWave(440, 20, 2, 44100, 2, 88200);
t = linspace(0, numSecs, numSamps);
y = zeros(1, numSamps);
for i = 1:numComponents
    y = y + (2/pi) * sin(2*pi*baseF*i*t)/i;
end
y = y * amp;
sound(y, sampRate);
end
```
function y = triangleWave( baseF, numComponents, amp, sampRate, numSecs, numSamps )
%This function produces a triangle wave, which is
%composed of the odd integer multiples of the
%wave's fundamental frequency,
%with alternating signs.
%Arguments:
%triangleWave(Base Frequency, # of Components, Amplitude, Sampling Rate, # of Seconds,
%# Samples);
%Sample function call:
%y = triangleWave(440, 20, 2, 44100, 2, 88200);
t = linspace(0, numSecs, numSamps);
y = zeros(1, numSamps);
for i = 0:numComponents
    y = y + (8/(pi^2)) * (sin(2*pi*baseF*(4*i+1)*t)/(4*i+1)^2 - sin(2*pi*baseF*(4*i+3)*t)/(4*i+3)^2);
end
y = amp * y;
sound(y, sampRate);
end

function y = singlePitchWave(freq, amp, sampRate, numSecs, numSamps)
%This function creates a sound wave of a single
%pitch
%Arguments:
%singlePitchWave(Frequency, Amplitude, Sampling Rate, # Seconds, # Samples)
%Sample function call:
%y = singlePitchWave(440, .9, 44100, 2, 88200);
t = linspace(0, numSecs, numSamps);
y = amp * sin(2*pi*freq*t);
sound(y, sampRate);
end

function nmat = createMidiOutput(sr, beatsInWholeNote, bpm)
%Create six random notes of random length and
%write the MIDI file. For demonstration
%only
[sampspn, secspn, bpn] = initializeNoteTypes(beatsInWholeNote, sr, bpm);
nmat = zeros(10,7);
prevTime = 0;
waitTime = 0.5;
onsetInBeats = 0;
for i=1:6
    n = round(rand()*12) + 1;
midiNum = 60+n
    freq = 261.63 * 2^((n)/12);
timbre = int8((rand()*3)) + 1;
    noteType = int8(round(rand()*4)) + 1
    note = createNote(noteType, freq, sr, timbre, secspn(noteType), sampspn(noteType));
    nmat(i,1) = onsetInBeats;  %onsetInBeats
    nmat(i,2) = beatsInWholeNote/noteType;  %durationInBeats
    nmat(i,3) = 1;  %MIDI channel;
    nmat(i,4) = 60 + n;  %MIDI pitch
    nmat(i,5) = 127;  %velocity
    nmat(i,6) = prevTime;  %onset
    nmat(i,7) = secspn(noteType);  %duration
    prevTime = prevTime + secspn(noteType);
    onsetInBeats = onsetInBeats + bpn(noteType);
    pause(secspn(noteType) + waitTime);
end
writemidi(nmat, 'test.mid');
end
function note = createNote( noteType, freq, sr, timbre, numSecs, numSamps)
  if (timbre == 1)
    note = singlePitchWave(freq, 0.8, sr, numSecs, numSamps);
  elseif (timbre == 2)
    note = sawtoothWave(freq, 100, 0.8, sr, numSecs, numSamps);
  elseif (timbre == 3)
    note = squareWave(freq, 100, 0.8, sr, numSecs, numSamps);
  elseif (timbre == 4)
    note = triangleWave(freq, 100, 0.8, sr, numSecs, numSamps);
  end
end

function [sampspn, secspn, bpn] = initializeNoteTypes(beatsPerWholeNote, sr, bpm)
  % sr is sampling rate
  % bpm is beats per minute
  % bpn is beats per note
  % sampspn is samples per note
  % secspn is seconds per note
  bpn = zeros(1, 5);
  sampspn = zeros(1, 5);
  secspn = zeros(1, 5);
  bpn(1) = beatsPerWholeNote;
  % Set the number of beats for each type of note, from a full note down to sixteenth
  % Each note has half the number of beats than the previous one
  for i = 2:5
    bpn(i) = bpn(i-1)/2;
  end
  % Set the number of seconds for each type of note.  For example, if there are 120 beats
  % per minute, then there 120/60 = 2 beats per second
  % If there are 2 beats per second and
  % each whole note has four beats, then
  % each whole note is 4/2 = 2 seconds long
  for i = 1:5
    secspn(i) = bpn(i) / (bpm/60);
  end
  % Set the number of samples for each type of note.  For example, if a
  % whole note is 2 seconds long, then it
  % consists of 44100 * 2 samples
  for i = 1:5
    sampspn(i) = int16((secspn(i) * sr));
  end
end