Algorithmic Music Composition

Dr. Rangarajan Krishnamoorthy
Software Consultant and CEO, Man Machine Systems
https://rangakrish.com / ranga@mmsindia.com

Introduction

In the traditional approach, musicians compose music largely by trial and error. They typically use an electronic "Keyboard" or some software such as "Ableton Live" [1], and try out various ideas. Although this is time consuming, it is unavoidable since music composition is a creative process and substantial experimentation is involved before producing music that is considered "acceptable". In the "algorithmic" approach, music is generated by a computer program with very little or no intervention from a human musician. The interesting thing is that this computer program itself could be generated automatically from data (the Machine Learning approach). Over the past few years, the field of algorithmic music composition has gained a lot of traction.

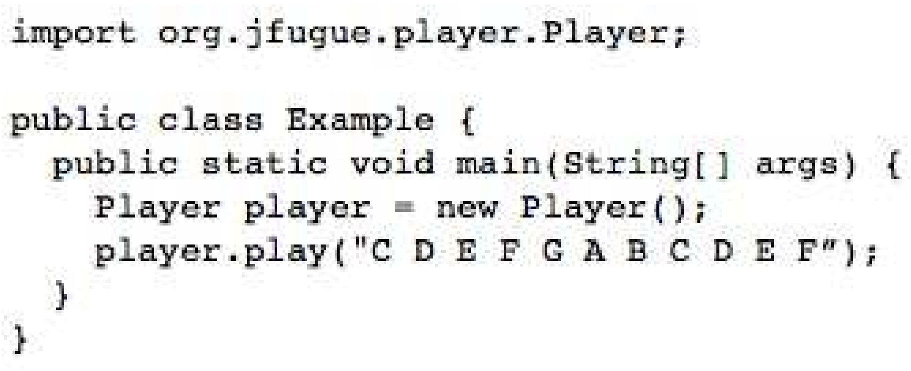
How Is It Done?

Talking of algorithmic composition, there are two ways to generate music. The first is to create a program manually by using one of the many libraries and frameworks available. This program, when executed, will generate music. The second approach is slightly more recent. It applies Machine Learning techniques to first build a model by analyzing vast samples of music. This model is then used to generate a new piece of music.

Approach-1: Writing the Music Composition Program

This involves using a suitable programming language and a framework/library to write a program that generates music. Fortunately, there are several nice tools available for this purpose. I will only mention a few that I have personally found interesting, so it is by no means an exhaustive list.

If you are a Java developer, you can use JFugue[2]. It is a compact Java library that allows you to conveniently express music sequence as a string, manipulate it and finally cause the player to play the music. Here is a toy Java program using JFugue:



```
import org.jfugue.player.Player;

public class Example {
  public static void main(String[] args) {
    Player player = new Player();
    player.play("C D E F G A B C D E F");
  }
}
```

Figure 1. JFugue Example

Max[3] is a popular commercial environment for creating music. It is different from many others in that it supports an intuitive visual programming interface and uses "patches" to assemble a piece of music. If we wish to do some exotic stuff, it is also possible to use Javascript to program the Max engine. Another nice thing about Max is that it can be interfaced to a variety of hardware, including the popular Arduino system.

Here is a simple “Patch” (as it is called in Max) that randomly generates and plays MIDI notes:

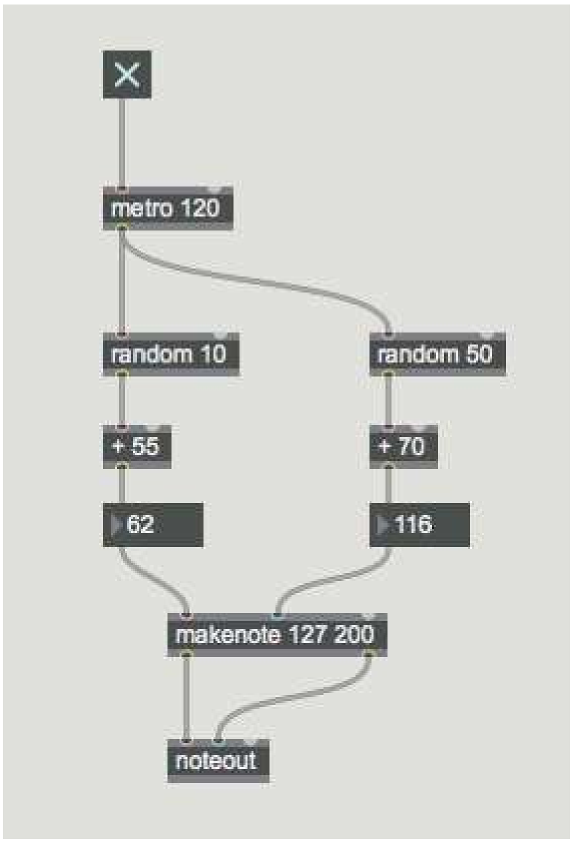
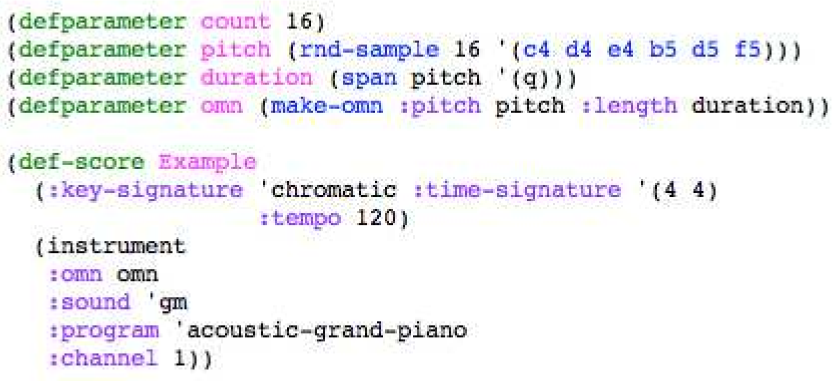


Figure 2. Max Patch

We can use pre-defined blocks of functionality (called “objects”), interconnect them in interesting ways to synthesize music. There are blocks for working with arbitrary audio and video data.

OpusModus [4] is a Common Lisp-based environment for generating music. It comes with a rich library of functions for synthesizing arbitrarily complex musical structures. What I like about this system is that it uses a powerful notation called “OMN” that can symbolically model highly complex multi-channel musical structure. Since Lisp is primarily a symbolic language, we can implement a variety of fancy algorithms to manipulate this symbolic structure.

Figure 3. shows a simple program written in OpusModus. It defines a random collection of 16 pitches from a pre-defined set and puts them together as “quarter” notes, with a tempo of 120.



```
(defparameter count 16)
(defparameter pitch (rnd-sample 16 '(c4 d4 e4 b5 d5 f5)))
(defparameter duration (span pitch '(q)))
(defparameter omn (make-omn :pitch pitch :length duration))

(def-score Example
  (:key-signature 'chromatic :time-signature '(4 4)
                  :tempo 120)
  (instrument
   :omn omn
   :sound 'gm
   :program 'acoustic-grand-piano
   :channel 1))
```

Figure 3. OpusModus Example

The music generated by this program, when rendered in the standard musical notation appears thus:

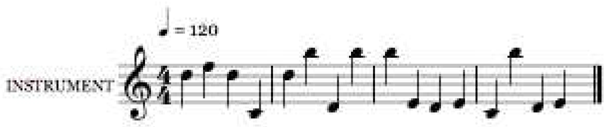


Figure 4. Music in Standard Notaion

For an example of music synthesized using OpusModus, listen to this [5] small piece, which I created.

ChucK [6] is a high-performance music language developed by Dr. Ge Wang at Stanford University. It is possible to run ChucK programs on multiple computers and let them interact seamlessly, thus creating a highly distributed music synthesis system.

Renoise [7] is a Digital Audio Workstation (DAW) that allows recording and composing production quality audio. It supports scripting using Lua language, so it is possible to generate music algorithmically as well.

SuperCollier [8] is an open-source platform for algorithmic music composition. It uses an interpreted scripting language called “sclang” to express composition logic.

You can check out a nice web-based app [9] that allows you to configure parameters such as Mood, Tempo and Accompaniment, and then will generate music algorithmically satisfying the stated constraints. No programming is involved from user side.

Approach-2: Using Machine Learning

In the ML approach, we let the computer analyze a vast amount of existing music, resulting in a “model”. This model can then be used to generate new pieces of music. An important research issue in this case is choosing a “good” representation (encoding) of the music that we are analyzing, so that all information is captured correctly. Recall that the input to any ML processing pipeline is a vector of numbers, not symbols. So if the input representation is not ”right”, the resulting model might not be good enough.

Last year, Google released “NSynth” [10], an innovative approach to audio synthesis using neural networks. A list of deep learning tools for music is available here [11, 12]. A very readable introduction on how to use RNNs to generate lyrics and piano music can be found here [13].

Comparison of the Two Approaches

Most of my research is based on the first approach. It is a very challenging approach and requires one to have a good grasp of the theory of music. Also, to generate good music using this approach, we have to interact with experts in the field and understand how they compose good music (“knowledge acquisition”). We can then try to encode the generation logic in the form of a rule-based system.

The ML approach is fast catching on, for a good reason. There is great progress in ML as a whole, and music generation can borrow ideas from the related fields. The important requirement (and a challenge) is to get hold of a vast amount of “good” music out there, encode them properly, and derive usable models from these. Since we can safely assume that the chosen samples represent good music, there is a good chance of generating good music using ML (of course, the model must be good too!)

MIDI and OSC

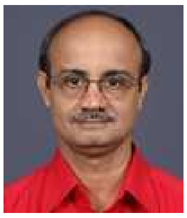
MIDI [14] stands for Musical Instrument Digital Interface. It is a protocol aimed at recording and paying back music on digital synthesizers (hardware or software). When a program generates music, there are two ways to play it. One is using the same computer’s hardware (provided the hardware supports sound and MIDI). The other is connecting the computer to a different piece of hardware such as a high quality digital instrument, or a an electronic keyboard. What happens is that the generated music is converted to MIDI format (this is optional; there are other formats too) and sent across to the target system (either the same computer or the external hardware). In fact, it is possible to save the generated music in MIDI format and play it using appropriate software on any computer (or even mobile phone). Even today, MIDI is widely used by synthesizers. Another protocol that is becoming popular is OSC [15] (Open Sound Control). This is much more expressive than MIDI in what can be represented (the details are not important for now). The good thing is that OSC usually used UDP format and hence it can be sent from one computer to another even wirelessly. It has a highly structured representation in comparison to MIDI and can be used effectively to simulate an orchestra of computers using a master controller. If our chosen music synthesis framework does not support OSC, we can integrate third party libraries available for this purpose. For example, JavaOSC [16] is a Java library for sending and receiving OSC packets.

The Future

With a lot of research going on in this area, the future looks very exciting for algorithmic music composition. One interesting research problem is to generate music based on the predominant emotion/mood to be captured. For example music that is synthesized for a sad occasion has to be different from one that is meant for a joyous occasion. The next level of advancement is to take a context representation as input and generate music for that context. To give an example of this type, consider Indian cinema. For a romantic song in a movie, the music director gets specification from the film director as to the nature of the scene - the age of the lead pair, whether it is a rural, urban or party setting, and so on. This description could be represented in a formal manner and fed to the algorithm for it to generate the matching music. Perhaps the ultimate one can expect (in the context of cinema) is to give the detailed movie script (in natural language) to the computer for it to come up with the background score and also songs! A simplified version of this would be for the computer to generate music after analyzing the lyrics of a song. The way research is progressing, the above may become reality in the next five to ten years! What will then happen to human musicians? We have to wait and see!

References

1) https://www.ableton.com/en/
2) http://www.jfugue.org
3) https://cycling74.com/products/max
4) https://opusmodus.com
5) http://www.rangakrish.com/wp-content/uploads/2015/09/Example1.mid
6) Ge Wang, Perry R. Cook, and Spencer Salazar, “ChucK: A Strongly Timed Computer Music Language”, Computer Music Journal, Winter 2015, pp. 10-29.
7) http://www.renoise.com/products/renoise
8) https://supercollider.github.io
9) http://computoser.com
10) https://magenta.tensorflow.org/nsynth-instrument
11) http://www.asimovinstitute.org/analyzing-deep-learning-tools-music/
12) https://arxiv.org/abs/1709.01620
13) http://warmspringwinds.github.io/pytorch/rnns/2018/01/27/learning-to-generate-lyrics-and-music-with-recurrent-neural-networks/
14) https://en.wikipedia.org/wiki/MIDI
15) http://opensoundcontrol.org/introduction-osc
16) https://github.com/hoijui/JavaOSC



About the Author: Dr. Rangarajan is a software consultant and CEO of Man Machine Systems, Chennai. He has over 30 years of experience in the software industry. His research interests are Programming Languages, Algorithmic Music Composition and Natural Language Processing. He has a BE (ECE) from College of Engineering, Gundry, ME (School of Automation) from IISc and Ph.D from Anna University. Outside of Computer Science, Dr. Rangarajan has a Diploma in Homoeopathy and an MD in Acupuncture. He is also the author of a popular astrology software called KPAstro ™. Dr. Rangarajan is a Member of IEEE and ACM.

Japanese cafe to use robots run by differently abled people: A Japanese cafe is planning to use robotic waiters controlled remotely by people with physical disabilities who wouldn't be able to work otherwise. The robots, which are 1.2-metre tall, transmit video footage and audio via the internet for operations including taking orders and serving coffee. The robots can be controlled even if the operator can only move their eyes.

Amazon unveils microwave that can be used with voice commands: Amazon on Friday unveiled a microwave that can be controlled by users by giving commands to the company's voice-based assistant Alexa. The users only need to give commands like, "Alexa, reheat my coffee" or "Alexa, defrost a potato" and the microwave will begin working on its own. Amazon has priced the microwave at $59.99 (over ₹4,300).