Music Composition Using Markov Chain

201321189 김다은
201321173 유수화

목차

1. 연구의 개요

현대수학 세미나 시간에 배운 마르코프체인을 이용하여 작곡을 해보고자 하였다. 알고리즘을 이용하여 알맞은 느낌의 곡을 작곡할 수 있도록 돕는 연구나 프로그램이 존재하는 것을 찾을 수 있었다. 한계는 있겠지만, 인간만의 영역이라고 생각되었던 창작을 컴퓨터를 통해 할 수 있게 된 것이다.

우리는 python으로 기존의 곡을 분석해서 각각 직전의 두 음(pitch), 음의 길이(duration), 음을 누르는 강도(velocity)에 의존하는 마르코프체인을 만들었다. 이 체인으로 만든 transition matrix를 이용하여 기존의 곡과 비슷한 느낌의 새로운 노래를 작곡하였다. 또한 곡 사이의 유사도를 높이기 위하여 order1인 마르코프체인으로 후 order2짜리 마르코프체인으로 작곡해보았다.

하지만 같은 곡을 분석하여 작곡한 곡끼리의 비교는 사실상 어렵다. 따라서 실제로 기존의 곡과 비슷한 느낌으로 작곡 되었는지를 비교해보기 위해 느낌이 다른 두 가수의 곡을 분석하여 두 곡을 작곡하여 비교했다.

2. 확률변수와 마르코프체인

1) 확률변수(random-variable)

(1) 확률변수(random-variable)는 $X : \Omega \to R$ 인 함수이다. (확률공간 : Ω, 실수: R)

(2) Let S be a set. If $X(\Omega) \subset S$, that is, if each outcome of X lie in S, we say that X is a S-valued random variable.

(3) A stochastic process $(Xn)n\in N$ is a sequence of random variables Xn for each $n \in N$. If Xn is S-valued for each $n \in N$, we say that $\{Xn\}n\in N$ is a S-valued stochastic process.

(4) A state is an element of S.

2) 마르코프체인

(1) stochastic process $(Zn)n\in N$일 때, 시간 n에서 확률 분포 Zn+1이 Zn 의 상태에만 의존할 때 Zn을 마르코프체인 혹은 마르코프과정이라고 한다.

$$P(Z_{n+1} = a \mid Z_n = i_n, \dots, Z_0 = i_0) = P(Z_{n+1} = a \mid Z_n = i_n)$$
$$for\ all\ n \geq 1\ and\ all\ i_0, \dots, i_n\ in\ S$$

(2) P(Zn+1 = a|Zn = b) = P(Z1 = a|Z0 = b) for all n ∈ N, a, b ∈ S 일 때, 마르코프체인은 time-homogeneous하다고 한다.

(3) Order k 인 마르코프체인

stochastic process (Zn)n∈N일 때, 시간 n에서 확률 분포 Zn+1이 Zn, … , Zn-k의 상태에만 의존할 때 Zn을 메모리가 k인 마르코프체인 혹은 order k 인 마르코프체인이라고 한다.

$$P(Z_{n+1} = a \mid Z_n = i_n, \dots, Z_0 = i_0) = P(Z_{n+1} = a \mid Z_n = i_n, \dots, Z_{n-k} = i_{n-k})$$
$$for\ all\ n \geq 1\ and\ all\ i_0, \dots, i_n\ in\ S$$

3. 마르코프체인을 이용한 작곡 idea

1) 기존 음악과 비슷한 곡의 작곡 과정을 설명하기 위한 음악적 용어

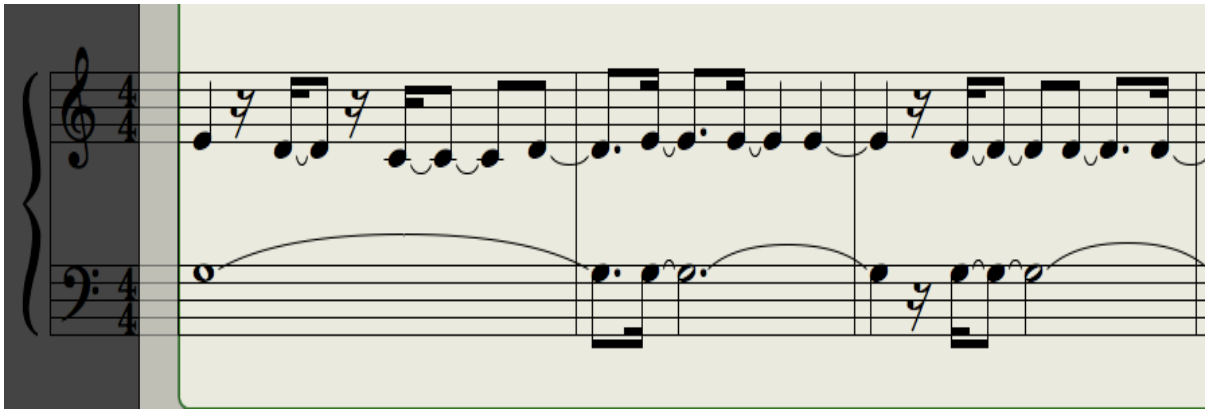
- tick : 음이 연주되는 시간
- tempo : 음악의 박자
- pitch : 음의 높낮이 (C,C#,D,D#,…)
- duration : 음의 길이 (8분음표, 4분음표,…)
- velocity : 음의 세기(쉼표는 velocity = 0 으로 정의한다.)

2) 가정

(1) 음악에는 여러가지 요소가 있는데 그 중 pitch, duration, velocity를 가장 중요한 요소로 고려했다. 어떤 음악과 다른 음악의 pitch, duration, velocity의 진행이 유사하다면 두 음악은 '비슷한 느낌'일 것이다.

(2) 계산 과정을 축소하기 위해 노래에서 사용된 pitch, duration, velocity만 이용하여 마르코프체인을 만들어도 결과적으로 이상이 없다. 이 때 사용된 음은 3옥타브 C# 에서부터 6옥타브 E까지이다.

3) 노래 선정 및 pitch, duration, velocity 추출

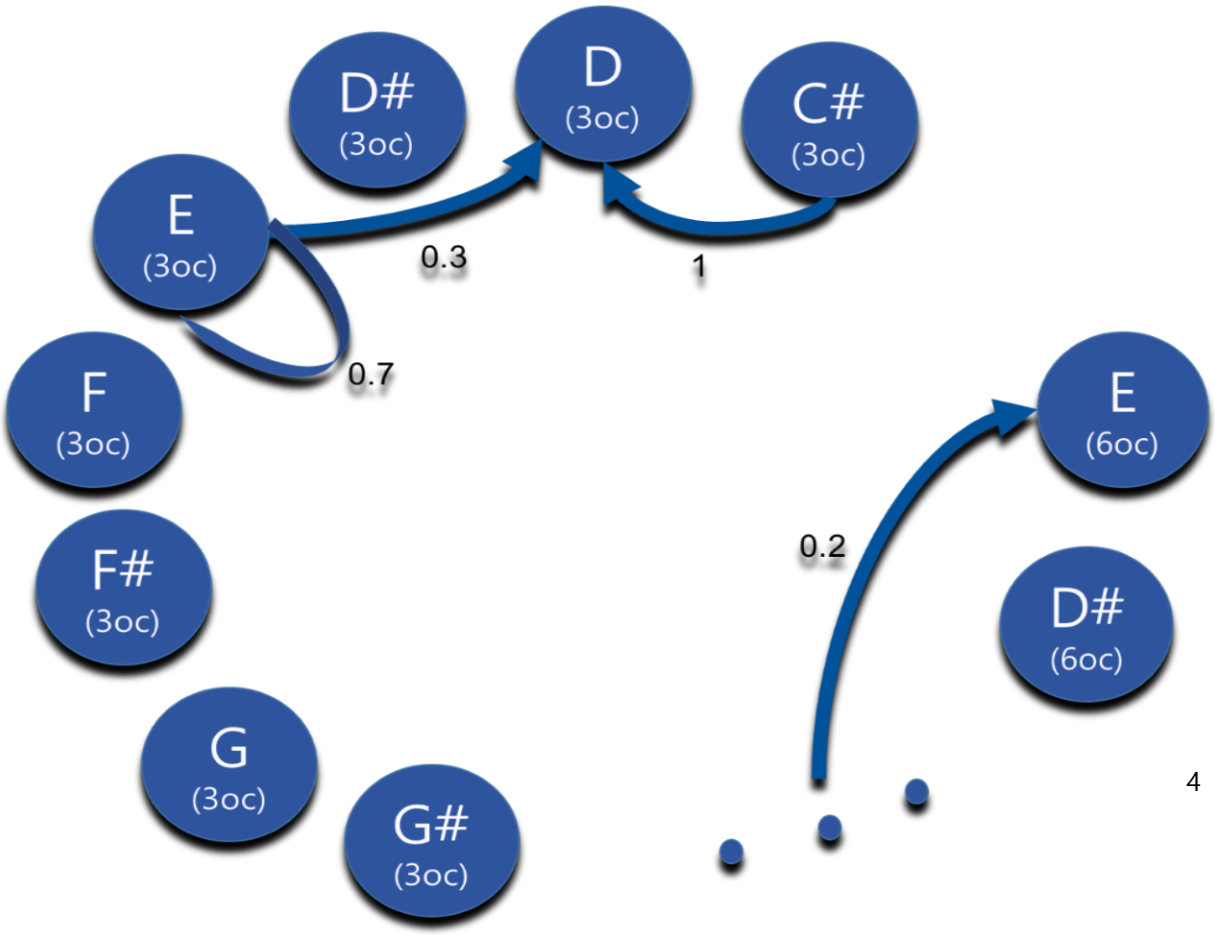


우리는 기존의 곡과 '비슷한'곡을 만들기 위해 악보에서 메인 멜로디 만을 pitch로 취하여 분석하였다. 예시로 위 악보의 노래를 분석하였다고 하면 pitch = [E D C D E E E ...]로 추출한 것이다.

마찬가지로 duration과 velocity도 추출하는데 velocity는 악보에는 정보가 담겨있지 않지만 분석할 때 사용한 midifile에는 정보가 있기 때문에 추출해 낼 수 있다.

4) 추출한 pitch, duration, velocity 로 각각의 마르코프체인 생성

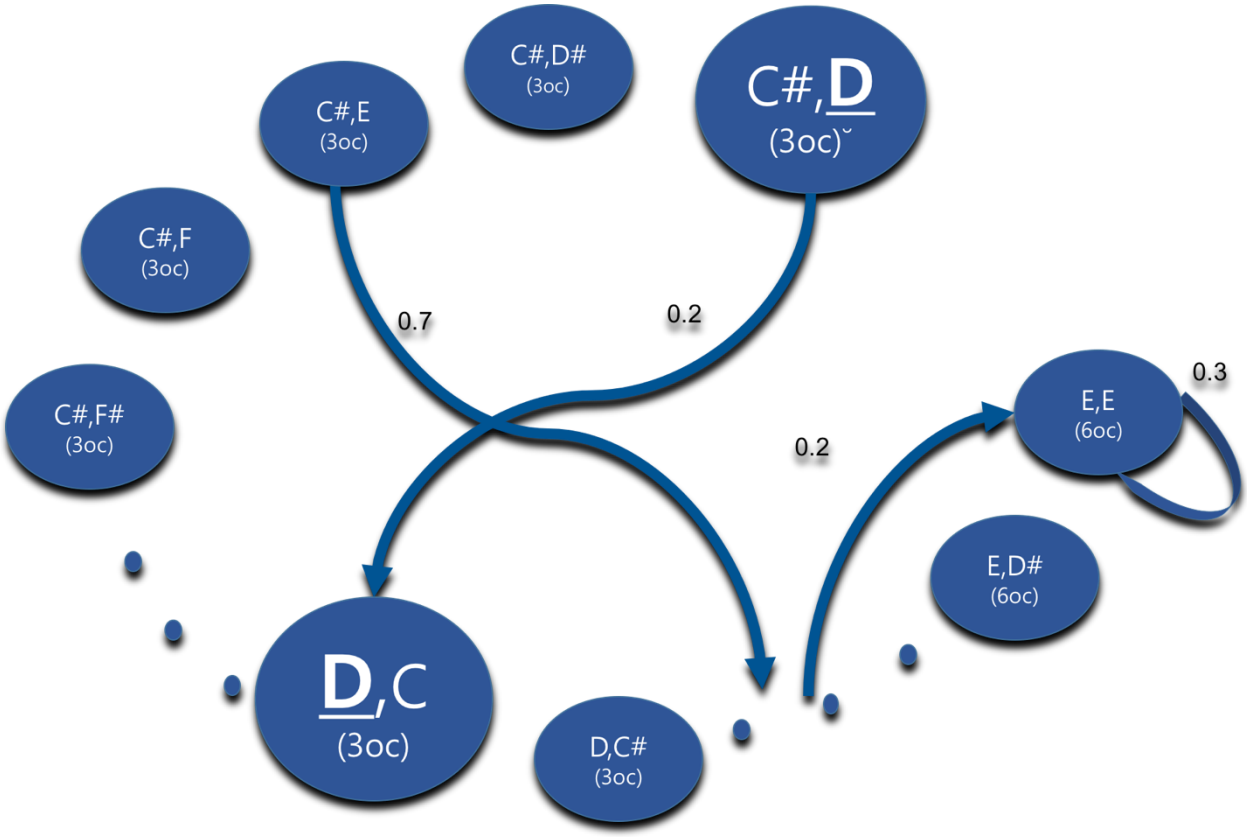
(1) order 1 인 마르코프체인(pitch)



위 그림에서처럼 노래에서 추출한 메인 멜로디에 있는 각각의 음들을 state로 하는 마르코프체인을 만들었다. S = {C#(3),D(3),D#(3), … ,E(6)}이다.

옥타브 뜻하는 oc 생략하고 숫자만 표기하였다.

(2) order 2 인 마르코프체인(pitch)



Order 2인 마르코프체인의 state S = {(C#(3),C#(3)), (C#(3),D(3)), (C#(3),D#(3)), … , (D(3),C(3)), (D(3),C#(3)), … ,(E(6),E(6))이다. Xn = {Zn, Zn-1} (n>=1)이기 때문에 Xn의 두번째 항과 Xn-1의 두번째 항이 일치할 때만 확률이 0보다 큰 값을 가질 수 있다.

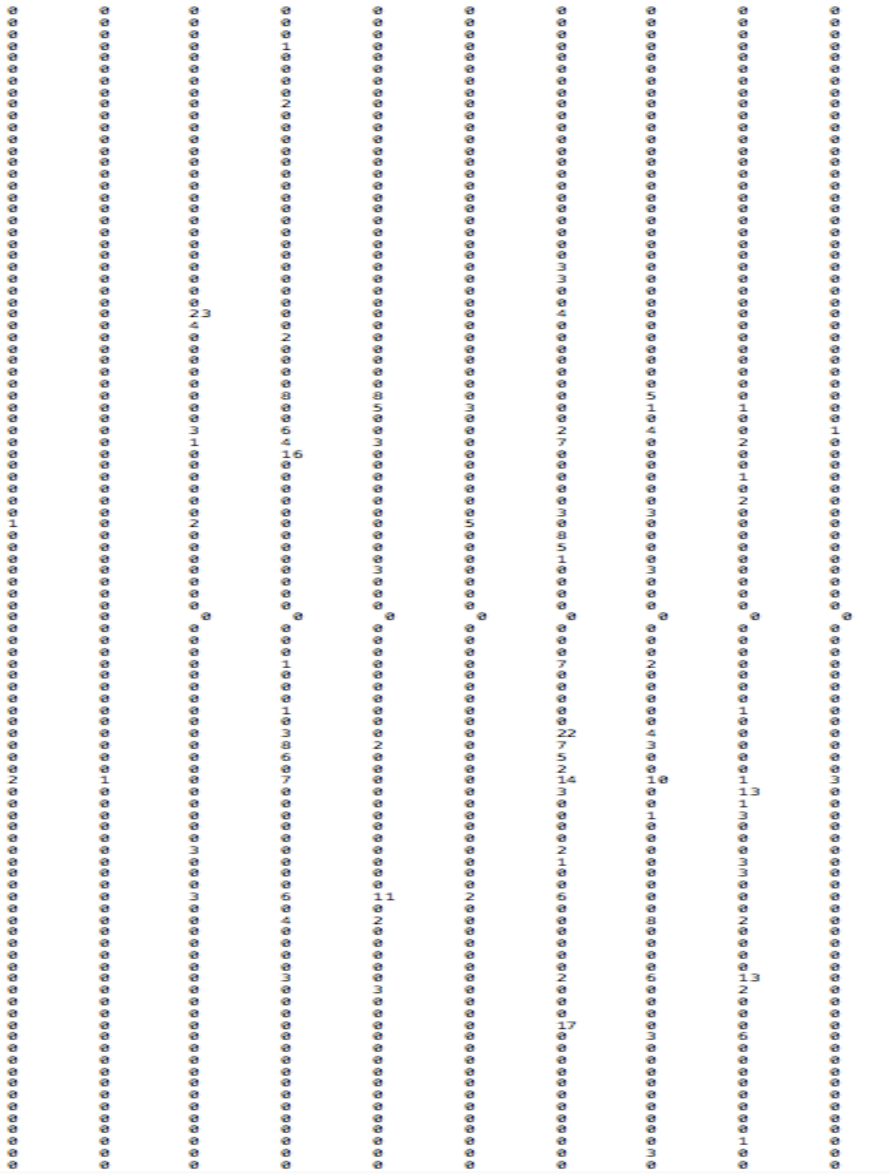
5) 마르코프체인을 토대로 경우의 수 matrix를 생성

(1) order 1인 마르코프체인으로 만든 경우의 수 matrix (velocity)

0	0	0	1	0	0	0	0	2	0
0	0	0	0	0	0	0	0	0	0
0	0	3	3	0	0	27	4	2	0
0	0	0	21	10	0	16	17	16	0
1	0	2	6	8	8	5	1	6	0
0	0	0	0	0	0	10	0	0	0
2	1	29	20	11	2	38	16	1	4
0	0	5	4	3	0	28	0	16	0
0	0	0	24	5	0	0	17	9	0
0	0	0	0	0	0	0	1	3	0

$P_{(i,j)}$ = i번째로 작은 velocity에서 j번째로 작은 velocity로 가는 경우의 수

(2) order 2인 마르코프체인으로 만든 경우의 수 matrix (velocity)



$X_n = \{Z_n, Z_{(n-1)}\}$ $n \geq 1$ 이므로 n^2 X n^2 행렬이 나타난다. 하지만 $P_{a,b,(c)}$가 $P_{(a,b),(b,c)}$의 모든 정보를 나타내므로 n^2 X n 행렬로 matrix를 만들었다.

6) transition matrix 생성

경우의 수 matrix의 원소를 각각 rowSum으로 나누어 주어 transition matrix를 만든다.

7) transition matrix를 이용한 작곡

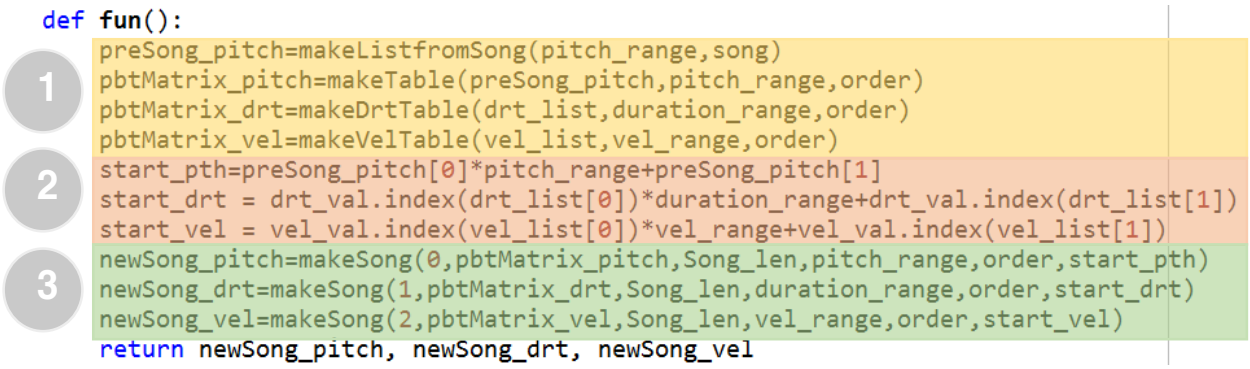
주어진 초기 벡터로 시작하며 transition matrix를 따라 다음에 나올 pitch, duration, velocity를 결정한다. 원하는 곡의 길이를 설정하여 pitch, duration, velocity를 결정한 후 시간에 따라 이들을 조합, 구성하여 새로운 노래를 만든다.

4. python에서의 구현

0) 곡 : Bruno Mars 의 Marry You, Just The Way You are / 누구의 뭐 선택

1) Fun() : main함수로, pitch, duration, velocity list로부터 transition matrix를 만들고, 초기벡터를 설정하고, 이를 이용하여 새로운 곡을 만든다.

아래에서 ①이 나타내는 것이 주어진 order에 따라 각 list를 각자의 range에 맞게 transition matrix를 만드는 것이다. ②는 Order가 2일 때 초기벡터를 정한 것이다. 주어진 list에서 젤 처음 2개의 상태를 참고하도록 정하였다. Order가 1인 경우는, makeSong 함수에서 list의 젤 처음 1개의 상태를 참고하도록 구현하였다. ③은 위에서 구한 transition matrix, 초기벡터로 구성한 Markov chain을 이용하여 새로운 곡을 만든 것이다. 새로만든 pitch, duration, velocity list를 가지고 midiutil package를 이용하여 곡을 쓸 수 있다.




```
def fun():
    preSong_pitch=makeListfromSong(pitch_range,song)
    pbtMatrix_pitch=makeTable(preSong_pitch,pitch_range,order)
    pbtMatrix_drt=makeDrtTable(drt_list,duration_range,order)
    pbtMatrix_vel=makeVelTable(vel_list,vel_range,order)
    start_pth=preSong_pitch[0]*pitch_range+preSong_pitch[1]
    start_drt = drt_val.index(drt_list[0])*duration_range+drt_val.index(drt_list[1])
    start_vel = vel_val.index(vel_list[0])*vel_range+vel_val.index(vel_list[1])
    newSong_pitch=makeSong(0,pbtMatrix_pitch,Song_len,pitch_range,order,start_pth)
    newSong_drt=makeSong(1,pbtMatrix_drt,Song_len,duration_range,order,start_drt)
    newSong_vel=makeSong(2,pbtMatrix_vel,Song_len,vel_range,order,start_vel)
    return newSong_pitch, newSong_drt, newSong_vel
```

2) makeTable(list1, state, order) : 주어진 list로 transition matrix를 만드는 함수이다.

주어진 list를 분석하여, current 상태 다음에 next 상태가 나타난 경우를 모두 count하여 matrix에 저장하였다. State가 n개, Order가 k일 경우, n^k X n matrix로 표현할 수 있다.

Order가 2인 경우를 생각해보면 아래 코드

```
currentIdx=0

for j in range(1,k+1):
```

currentIdx+=(n**(k-j))*list1[i+j-1]

을 간단하게 currentIdx=list1[i]*n + list1[i+1]로 표현할 수 있다.

```
#song list로 probability matrix 만들기
def makeTable(list1,n,k): #n=state 수?
    NewList=makeZeroM(n**k,n)
    for i in range(len(list1)-k):
        currentIdx=0
        for j in range(1,k+1):
            currentIdx+=(n**(k-j))*list1[i+j-1]
        nextIdx=list1[i+k]
        NewList[currentIdx][nextIdx]+=1
    return NewList
```

3) makeSong(mode, transitionMatrix, SongLength, range, order, initialVector) : 새로운 곡의 입력한 mode(0 : pitch, 1 : duration, 2 : velocity)의 list를 만드는 함수이다.

현 상태에 따라 다음 상태를 결정할 수 있는 확률이 주어질 경우, 그 확률에 의해 다음 상태를 정해야한다. 이 기능을 아래 코드와 같이 구현하였다. 먼저 임의의 상태 a일 때(a번째 row) 나타날 수 있는 값들은 경우의 수이다 (row의 합으로 나누어 확률로 나타내지 않았다.). 그 경우의 수를 누적하여 cummulative matrix로 나타낸 것이 makePbtCml()함수(=cmtM)이며, transition matrix의 row 합을 구하여 rowSum에 저장했다. 1)에서 설명했듯이, order가 1인 경우는 원래곡의 index 1이 초기상태이고, order가 2인 경우는 fun()에서 구한 초기상태값이(n*list[0]+list[1])이 초기상태이다. Newsong의 길이가 입력한 길이가 될 때까지, while문을 실행하며 노래를 만든다. 초기상태는 주어진 곡에서 얻은 것이므로 rowSum이 0이 아닌 값이 나오고 따라서 계속해서 rowSum이 0이 아닌 값이 나오게 된다. 따라서 rowSum이 0이되어 else문으로 빠지는 경우는 드물다. 다만 list의 길이가 q일 때, index q-1이 마지막 index이므로 이를 참조하게되면 rowSum이 0이 될 수 있다. 이 때는 다음 상태를 random하게 선택하였다. rowSum이 positive인 동안 cmtM을 탐색하여 해당하는 next 상태를 찾아 newSong에 추가한다. 이때 Next가 current가 되야하는데, order가 2인 경우는, nXn으로 변환을 해주어야한다. Next+(current%rng)*rng과 같은 식을 사용하여 index를 정해주었다.

```
#probabilty matrix에서 주어진 확률에 의해 다음상태 선택하여 새로운 song 만들기
def makeSong(mode,pbtM, lenofSong,rng,Ord,start):
    cmtM=makePbtCml(pbtM)
    rowSum=sumofRows(pbtM)
    Next=0
    if Ord==1:
        if mode==0:
            current=song[1]-pitch_min
        elif mode==1:
            current=drt_val.index(drt_list[1])
        elif mode==2:
            current=vel_val.index(vel_list[1])
    elif Ord==2:
        current=start
    if mode==0:
        Newsong=[song[1]-pitch_min]
    elif mode==1:
        Newsong=[drt_val.index(drt_list[1])]
    elif mode==2:
        Newsong=[vel_val.index(vel_list[1])]
    while len(Newsong)<lenofSong:
        if rowSum[current]!=0:
            ran=random.randrange(0,rowSum[current])
            #print("&",len(cmtM),len(cmtM[0]))
            for Next in range(len(cmtM[0])):
                if cmtM[current][Next]>ran:
                    #print(Newsong)
                    Newsong.append(Next)
                    if Ord == 1:
                        current = Next
                    else:
                        current=Next+(current%rng)*rng
                    break

       else:
            ran=random.randrange(0,rng)
            Newsong.append(ran)
            if mode==0:
                print("0",len(Newsong)-1,Newsong[len(Newsong)-1])
            elif mode==1:
                print("1",len(Newsong)-1,Newsong[len(Newsong)-1])
            else:
                print("2",len(Newsong)-1,Newsong[len(Newsong)-1])
            if Ord == 1:
                current = Next
            else:
                current = ran+((current%rng)*rng)
   return Newsong
```

4) tick_list(tick) : duration과 velocity list를 구성할 때 사용하는 함수이다.

기존 곡에서 얻어낸 duration(velocity) list를 tick이라 한다. 이때 duration(velocity)의 state를 알기 위해 set(tick)을 사용하여 중복을 제거해주고, sort()하여 오름차순으로 dictionary로 표현하였다. 따라서 우리는 곡에 사용된 duration(velocity)만으로 matrix를 만들 수 있다.

```
def tick_list(tick):
    tick=list(set(tick))
    tick.sort()
    newTick={}
    for i in range(len(tick)):
        newTick[i]=tick[i]
    return newTick

drt_dic=tick_list(drt_list)     #duration dictionary
duration_range=len(drt_dic)     #duration range
drt_val = list(drt_dic.values())    #dictionary로 부터 value를 뽑아 velocity list를 구성
vel_dic=tick_list(vel_list)     #velocity dictionary
vel_range=len(vel_dic)          #velocity range
vel_val=list(vel_dic.values())  #dictionary로 부터 value를 뽑아 velocity list를 구성
```

5) readSong(pattern) : 제일 처음 실행되는 함수로 midi file을 읽어서 pitch, duration, velocity를 분석한다. Pitch는 NoteOffEvent일 때만 분석하여 NoteOnEvent와 중복하지 않게 선택하였다. 그리고 duration은 event.tick이라는 함수를 사용하여 구하였고, duration이 0인경우는 고려하지 않았다. Velocity는 NoteOnEvent일때만 분석하여, 너무 잦은 0이 나오는 경우를 방지한다(쉼표의 경우 velocity가 0인데, 따라서 쉼표는 고려하지 않았다.).

read_midifile함수를 사용하여, midi file을 읽고, pattern에 저장하여 readSong(pattern)을 이용해 분석한다. 아래 코드에서는 BrunoMars의 just the way you are과 marry you를 입력한다. Song(pitch list), drt_list(duration list), vel_list(velocity list)는 모두 global로 정의하였으므로 여러 개의 midi file을 읽어도 각 pitch, duration, velocity 리스트에 이어서 저장된다.

```
def readSong(ptn):
    for track in ptn:
        for event in track:
            if isinstance(event, midi.NoteOffEvent):     #
                p=event.get_pitch()
                song.append(p)
                t=event.tick
                if t!=0:
                    drt_list.append(t)
            if isinstance(event, midi.NoteOnEvent):
                vel_list.append(event.get_velocity())

#midi package import시 사용가능
pattern=midi.read_midifile("JustTheWayYouAre_vocal.mid")
readSong(pattern)
pattern=midi.read_midifile("MarryYou_vocal.mid")
readSong(pattern)
```

6) 마지막으로 실행되는 부분으로 tempo, track, channel, pitch, time, duration, velocity의 정보를 가지고 midi file을 생성한다. Track=0, channel=0으로 고정해두고 변화되는 pitch,

duration, velocity에 신경쓰며 MyMIDI를 완성한다. Pitch의 경우 range를 줄이기 위해 pitch_min을 빼주었던 것을 다시 더해주고, time은 duration만큼 계속 더해진다. 최종적으로 길이(=note 수) Song_len의 hi.mid라는 midi file이 만들어졌다.

```
track    = 0
channel  = 0
time     = 0    # In beats
duration = 1    # In beats
tempo    = 60   # In BPM
volume   = 100  # 0-127, as per the MIDI


MyMIDI = MIDIFile(1)  # One track, defaul
                      # automatically)
MyMIDI.addTempo(track, time, tempo)
```

```
for i in range(Song_len):
    MyMIDI.addNote(track, channel, mySong_pth[i]+pitch_min,
                   time, drt_dic[mySong_drt[i]]*0.005, vel_dic[mySong_vel[i]])
    time = time + drt_dic[mySong_drt[i]]*0.005

with open("hi.mid", "wb") as output_file:
    MyMIDI.writeFile(output_file)
```

5. 결과물(작곡한 음악 파일 첨부)

[simulation]

새로운 곡의 pitch-duration-velocity의 list 일부가 아래와 같은 결과가 나왔다.

```
65 0.01 74
65 0.025 76
67 0.025 77
69 0.045 79
72 2.41 77
72 0.01 74
69 0.01 77
72 0.01 77
69 0.01 67
72 0.01 81
65 0.295 72
65 0.16 81
67 0.01 72
65 0.195 79
65 0.01 77
65 0.02 74
67 0.01 72
69 0.02 79
76 0.225 70
74 0.91 70
72 0.01 77
74 0.06 70
70 0.01 77
69 0.035 70
70 0.06 77
```

6. 고찰

마르코프체인을 이용하여 작곡된 곡이 귀로 들었을 때 원곡과 느낌이 비슷하다.

보완이 되면 좋을 3가지가 있다. 먼저 data가 부족하여서 충분히 더 좋은 음악이 나오지 못했다. 연구를 더 하게 된다면 더 많은 data를 이용하여 작곡을 해보고 싶다. 또한 음악의 처음부터 끝까지 비슷한 구성으로 진행이 되어서 재미가 덜 했던 것 같다. 시작-후렴-마무리 구간에 따른 조건이 만들어 진다면 더 좋은 노래가 될 것이다. 마지막으로 메인 멜로디 만을 추출하여서 작곡하였기 때문에 음악이 단조로운 부분이 있다. 여러 악기와 화음으로 작곡을 한다면 풍성한 음악이 될 수 있을 것이다.