卒業論文

アーティストの特徴を用いた
自動作曲ツールの作成

情報工学科
東海林研究室
出席番号　３番
岡田　大輝

目次

1. はじめに

「音楽鑑賞」は人が最も好む趣味のひとつである. そして衣食住に関わる生活空間のムード作りなどにも重要な要素として取り入れられている. これらは人間の感情に影響を与えるという音楽の持つ特性といえる. この観点から, 誰もが自分だけのオリジナルの楽曲を作曲するということが実現されれば, 自己表現の手段の一つとなる可能性も大いにあると思われる. しかしながら, 音楽的な知識は誰もが持っているものではなく, 作曲をするという行為はそう簡単な事ではない. 一方, コンピュータによる自動作曲の研究は, コンピュータが世の中に現れた当時から課題とされ, 今日までに様々な方法が提案されてきた.

ある一つの楽曲を聴いた複数の人の評価は異なることがある. ある人は「良い曲だ」と思ったとしても, 他の人は「あまり良いとは思わない」と答えるかもしれない. これは楽曲を聴くという行為に人間の感性が大きく影響し, 楽曲に対する一人一人の主観の違いが原因となって起こる事であると考えられる. 例をあげると, ある一つの方法で自動作曲された曲に, 果たして全ての人が納得することができるのか, 必ずしもそうとは言えない結果になると思われる. また, 2 人の人が楽しい曲として作曲した 2 つの作品は, 個人の主観によって大きく違うものである可能性がある. こういった事から, コンピュータによる自動作曲においても, 作り手それぞれが納得できる作曲を行える必要があると考えられる. 人間の感性や主観の違いを考慮し, 作り手が好きなアーティストの曲調に類似性がある曲を生成できればよいと考えた.

本研究では, 初心者の作曲活動の補助として, 好きなアーティストの曲風に似た作曲ができるようなツールを作成する. その足がかりとして, 対象アーティストの曲を数曲読み込み, そこから曲の特徴を抽出し, その特徴を基にデータベースを作成する. そのデータベースと累積分布を用いて自動作曲するツールを作成する. 本研究のオリジナリティを出すために, WAVE ファイルからの抽出をする機能を重点的にツールの作成した.

2. 特徴について

本研究では，3 つの以下の特徴について取り扱った.

- 基本周波数(メロディー部)
- 音価(リズム部)
- コード進行(コード部)

このうち，基本周波数はメロディーとなっていて，一般的にメロディーの音は同じ音程か，隣接する音程へ進行するのがほとんどで，離れた音程への進行ほど少なくなる．このような比率は，ジャンルごとに傾向がある．例をあげるとクラッシックに比べてロックでは離れた音程に進行する割合が高くなる．クラッシックのほうがロックより穏やかな感じがするのはこのためである．さらに，クラッシックと現代音楽を比較してみると，クラッシックでは，音と音の流れがスムーズなのに対して，現代音楽では，あちこちに音がジャンプする．このような進行の比率情報は重要である．よって知識ベースに記録して，ジャンルやアーティストの特徴を取り込むようにしなければならない.

次に，音符の長さとしても 16 分音符，8 分音符，4 分音符などの各音価が，どれくらいの割合で使用されているか，それを推移確率として統計的に調べることによって知識ベースを作成する．音価の並びはリズムパターンとして知覚されるのでこれもアーティストによって特徴がでる.

最後に，コード進行の知識ベースはメロディーの場合と同様に，推移確率を調べることにより作成する．これもアーティストやジャンルによってバリエーションがいくつかあり，推移の比率情報が重要である[1][2][3][4][5].

2.1 メロディー

メロディ(旋律)は音楽を構成する要素の一つ．ある一定時間の音のうち，ヒトが音楽として意味のあるひとまとまりであると認識する（通常単音の）基本周波数の連なりである.

基本周波数とはヒトの聴覚により，音の周波数成分のなかで「基本」になっていると認識される周波数のことで，通常は最も低く，最も音圧の大きい成分である

基本周波数は，自己相関によって抽出できると推定される.

2.1.1　自己相関

　下記のような条件で，sin波があるとする．このグラフのsin波をを用いて自己相関について説明する．

振幅：　A　[m]

周期：　T　[sec]　１回振動する間の時間

周波数：　f　[Hz]　１秒間の振動数

T = 1 / f [sec]

１回転するときの時間が　T　[sec]　なので，

ω×T＝2π　[rad]

となる．

ω＝２πｆ

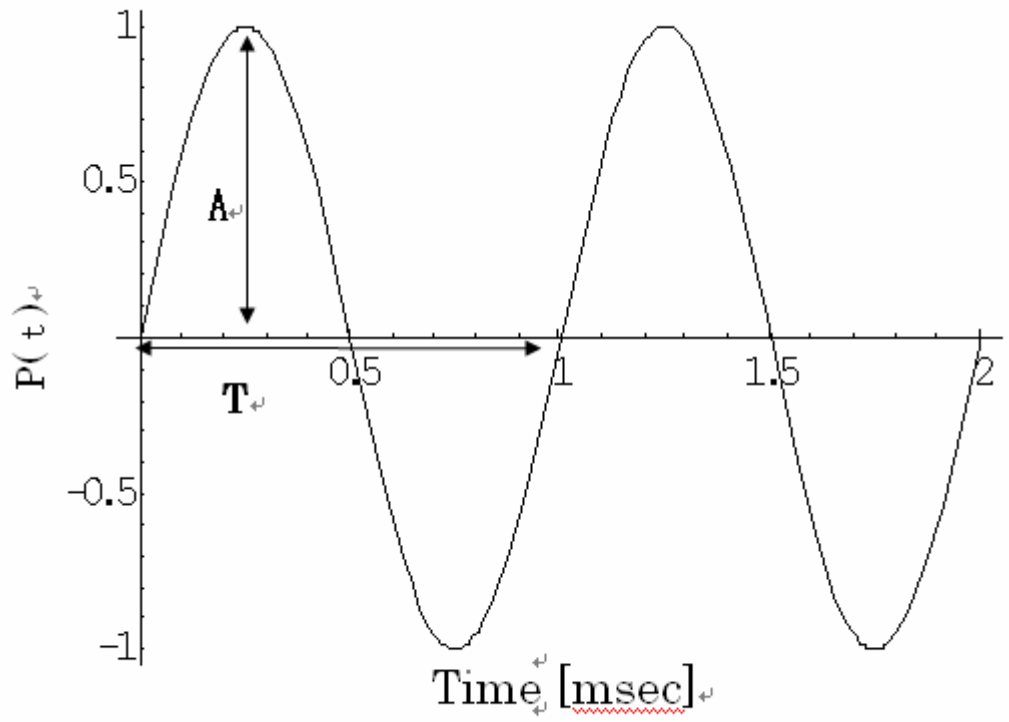


図1

・自己相関関数の定義

時間的または空間的に離れた２点で関係の強さを関数で表します定量的に相関の強さを調べることができたら，波の重ね合わせである音の解析に応用できる．音の時間的特徴を表すために自己相関関数G(τ)を定義する．

$$G(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} p(t)\, p(t+\tau)\, dt$$

$$G(\tau) = \int_{-\infty}^{\infty} p(\omega)\, e^{i\omega t}\, d\omega$$

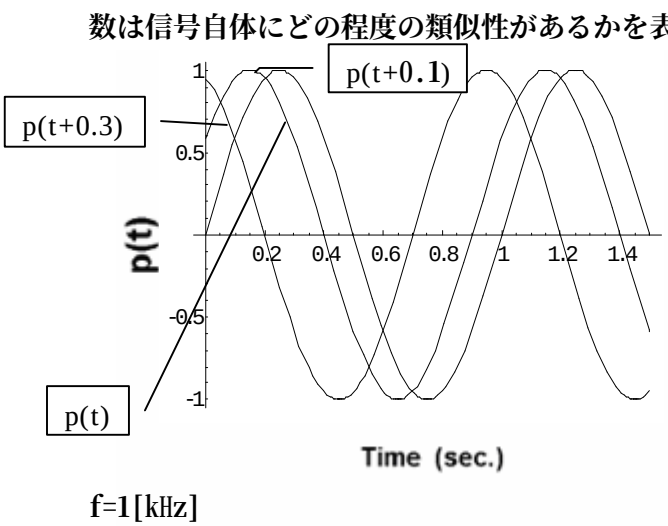
$$P(\omega) = \int_{-\infty}^{\infty} G(\tau)\, e^{i\omega t}\, d\tau$$

音の時間関数をｐ(ｔ)とする．ここで，τ は遅れ時間，Tは積分区間です．信号のパワースペクトルを P(ω) は，自己相関関数 G(τ) のフーリエ変換から求めることができる．式を離散データ p(k) に適用すると，

$$G(L) = \frac{1}{N} \sum_{k=0}^{N-1} p(k)\, p(k+L) \qquad (L=1,\ 2,\ 3,\ \cdots N-1)$$

上式のようになる．

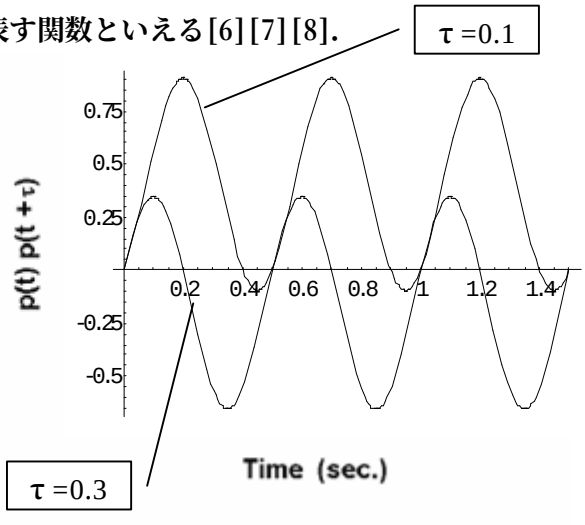
次に，周波数(f = 1Hz)の波を考える．積分区間 T の長さの信号 p(t) を切り出すと，時間 τ 遅れた信号は p(t+τ) となる．もし p(t) と p(t+τ) の振幅が大きく，同様な繰り返し成分があれば，２つの信号の相関値 G(τ) は大きくなる．つまり，自己相関関数は信号自体にどの程度の類似性があるかを表す関数といえる[6][7][8]．



f=1[kHz]

p(t)=sin(wt)のグラフ

図2



p(t)×p(t+τ)のグラフ

図3

・自己相関関数の具体例

sin関数について

p(t) = sin(ωt)の自己相関関数は

$$G(\tau)=\lim_{T\to\infty}\frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}}\sin(\omega t)\sin(\omega t+\omega\tau)\,dt$$

$$=\lim_{T\to\infty}\frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}}\sin(\omega t)\{\sin(\omega t)\cos(\omega\tau)+\cos(\omega t)\sin(\omega\tau)\}\,dt$$

$$=\lim_{T\to\infty}\frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}}\left\{\sin^2(\omega t)\cos(\omega\tau)+\sin(\omega t)\cos(\omega t)\sin(\omega\tau)\right\}dt$$

$$=\lim_{T\to\infty}\frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}}\left\{\sin^2(\omega t)\cos(\omega\tau)+\sin(\omega t)\cos(\omega t)\sin(\omega\tau)\right\}dt$$

$$=\lim_{T\to\infty}\frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}}\left\{\frac{1-\cos(2\omega t)}{2}\cos(\omega\tau)+\frac{\sin(2\omega t)}{2}\sin(\omega\tau)\right\}dt$$

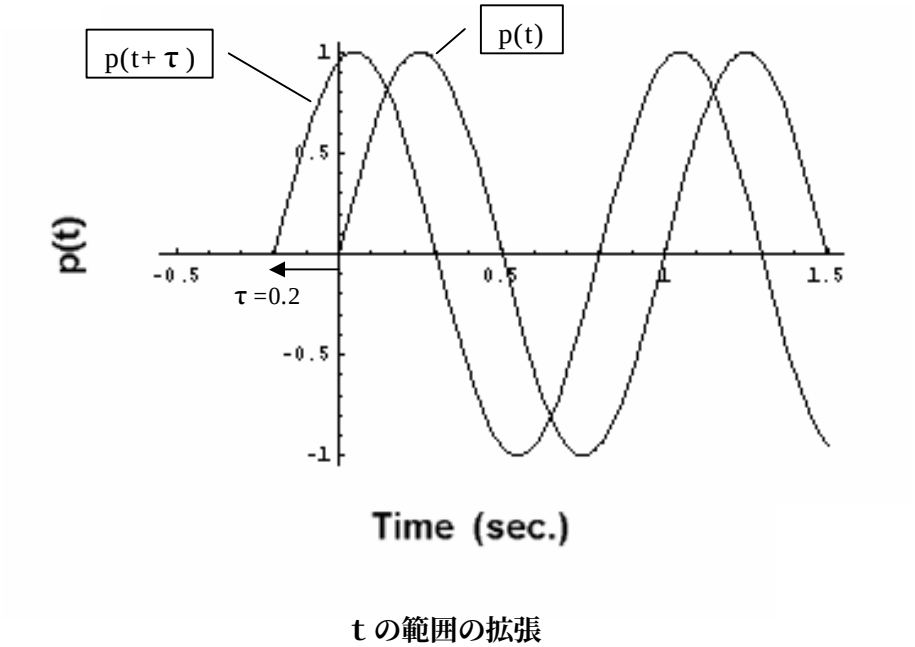
$$=\lim_{T\to\infty}\frac{1}{T}\left\{\cos(\omega\tau)\left[\frac{t}{2}-\frac{\sin(2\omega t)}{4\omega}\right]_{-\frac{T}{2}}^{\frac{T}{2}}+\sin(\omega\tau)\left[-\frac{\cos(2\omega t)}{4\omega}\right]_{-\frac{T}{2}}^{\frac{T}{2}}\right\}$$

$$=\lim_{T\to\infty}\frac{1}{T}\left\{\cos(\omega\tau)\frac{T}{2}\right\}=\frac{\cos(\omega\tau)}{2}$$

となる.

τ範囲が (0 ≦ τ ≦ T)となるので, p(t)は, (0 ≦ t ≦ 2T)の範囲で値を持たなければならない. 但し, p(t+τ) (0 ≦ τ ≦ T) である.

もし, T≦ t で p(t) が0になると, τが大きくなるにつれて p(t) × p(t+τ)が小さくなっていく.



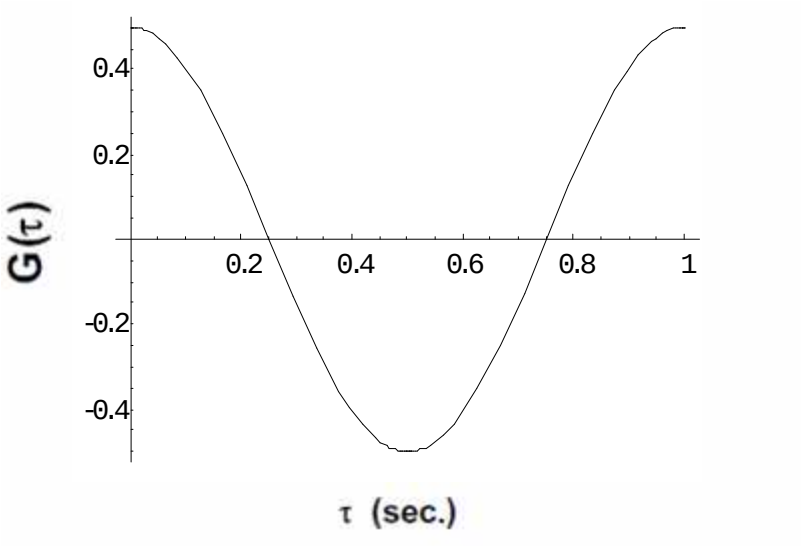
tの範囲の拡張

図4

sin関数の自己相関自己相関を行うと以下の図のような

$$G(\tau)=\frac{\cos(\omega\tau)}{2}$$

の式のグラフが得られる.



sin(ωt) の自己相関関数G(τ)：p(t) (0 ≦ τ ≦ 2T)

図5

・周期→周波数→MIDIナンバーへの変換方法

MIDIでの音名：音名はドレミ･･･のように音に名前をつけたもので，MIDIでは中央のドを60として，上下に0～127までを割り振っている．周波数とMIDIナンバーの関係は，グラフにあらわすと以下のようになる.

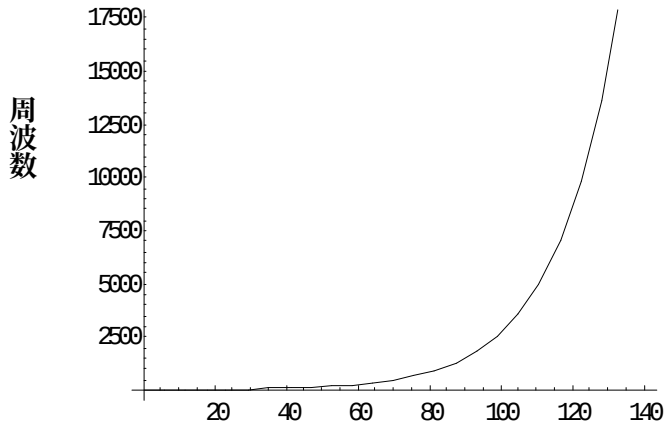


図6

周波数の関係を式で表してみる.

音階	周波数		コード
	測定値	計算値	
0	8.176	8.175798916	C

図7

[変数]

０．変数は　周波数 F(Hz) とノートナンバー N とする.

１．基準音は中央のラ(69)で，440Hz.

２．1オクターブ上がると周波数は2倍になる.

３．1オクターブには12音在る.

指数関数は$Y=M*X^n$であるので，まずは１より

$$F(Hz)=440(Hz)*X^{(69-N)}$$

と仮定する.

２よりX=２が確定し,

$$F(Hz)=440(Hz)*2^{(69-N)}$$

となる.

さらに，３からNが12増えると２倍になる事から,

$$F(Hz)=440(Hz)*2^{\{(69-N)/12\}}$$

この仮定でN=0として計算すると

$$F(Hz)=440(Hz)*2^{\{(69-0)/12\}}$$

$$=23679.64Hz$$

となり測定値とはかなり違う.

そこで，乗数部分の(69-N)を(N-69)にして,

$$F(Hz)=440(Hz)*2^{\{(N-69)/12}$$

として，N=0のときを計算すると

$$F(Hz)=440(Hz)*2^{\{(0-69)/12\}}$$

$$=8.1757Hz$$

となり，測定値(N=0のとき約8.176Hz)と等しい.

よって，下線部の式をN=の形に直してやると

$$N=12\times\log2(F/440)+69$$

自己相関によって抽出された周期Tを周波数Fにするには

$$F=サンプリングレート/T$$

このFをN=の式に代入し，MIDIナンバーを求める.

・WAVEファイルを読み込んで自己相関した結果の具体例

二つのデータについて自己相関を行った.

○ドレミファソラシド(周波数で求めたもの)

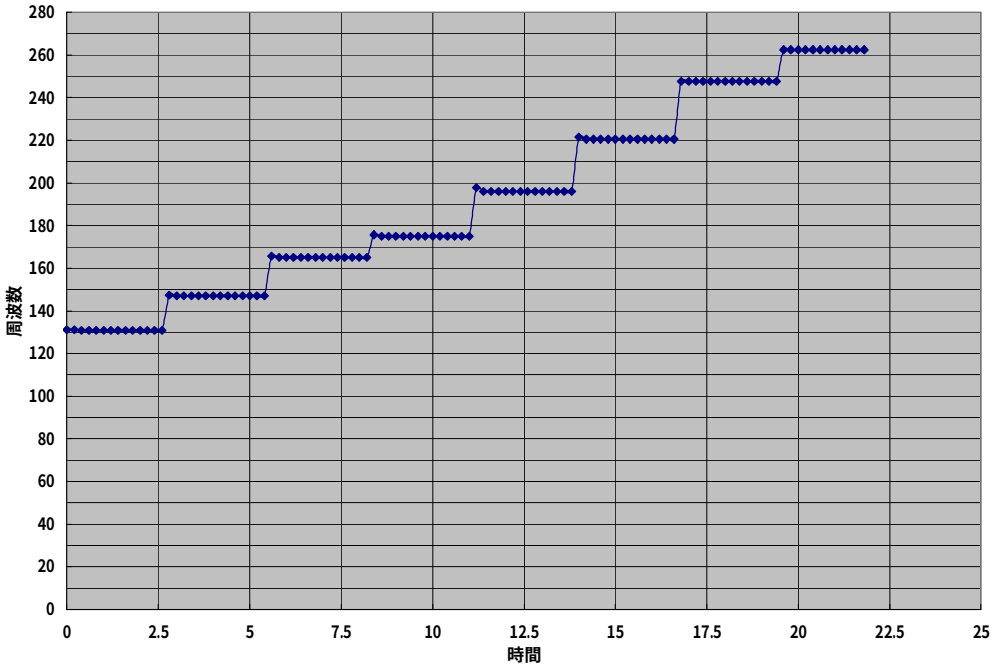


図8

○きらきら星(MIDIナンバーで求めたもの)

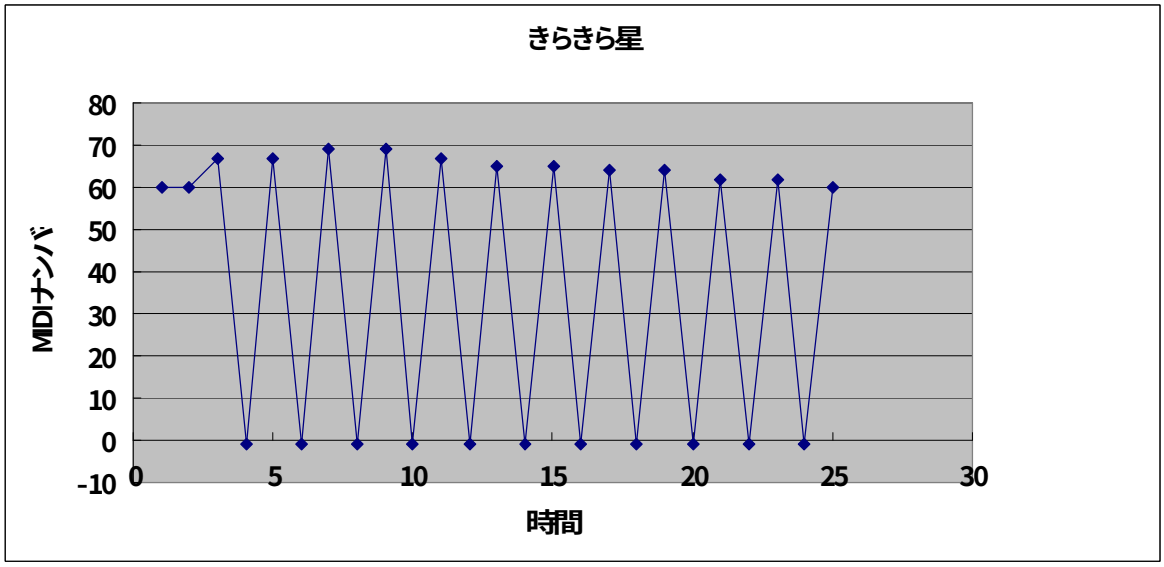


図9

上図のMIDIナンバーが－1の部分は休符である.

2.2　音価

音価（おんか）とは，音楽においてある音（または休止）に与えられた楽譜上の時間の長さをいう．楽譜上における符が拍に対する相対的な長さを示す以上，音価もそうであって拍の長さが（すなわちテンポが）変われば，それに伴って同じ音価も長くなったり短くなったりする[10].

・音符の名称，形，拍数

名称	全音符	2分音符	4分音符	8分音符	16分音符	32分音符
音符						
拍数	4	2	1	1/2	1/4	1/8
休符						
名称	全休符	2分休符	4分休符	8分休符	16分休符	32分休符

図10

音符とそれに対応する休符（4分音符と4分休符）は同じ長さである．

・付点音符

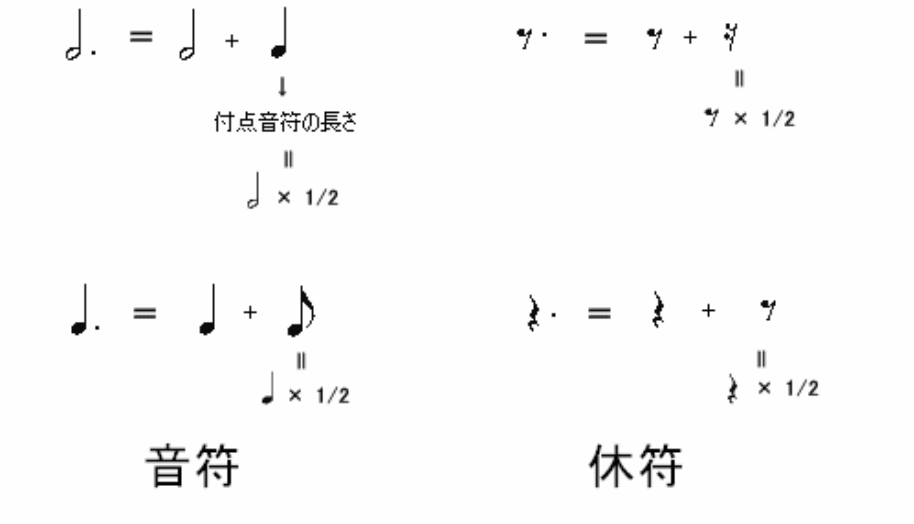


図11

このように付点音符は付点の付く音符の長さの1/2をプラスした長さになる．付点休符も同じである．

2.2.1　音価の抽出手法

先ほどの自己相関で抽出したデータ列を以下に表す.

・きらきら星

秒	MIDI_N	秒	MIDI_N	秒	MIDI_N	秒	MIDI_N	秒	MIDI_N	秒	MIDI_N
0	60	5	-1	10	67	15	67	20	60	25	65
0.1	60	5.1	53	10.1	67	15.1	67	20.1	-1	25.1	-1
0.2	60	5.2	65	10.2	67	15.2	67	20.2	52	25.2	46
0.3	60	5.3	65	10.3	67	15.3	-1	20.3	60	25.3	65
0.4	60	5.4	65	10.4	67	15.4	67	20.4	60	25.4	65
0.5	60	5.5	65	10.5	-1	15.5	67	20.5	60	25.5	65
0.6	60	5.6	65	10.6	67	15.6	67	20.6	60	25.6	65
0.7	60	5.7	-1	10.7	67	15.7	67	20.7	60	25.7	65
0.8	60	5.8	65	10.8	67	15.8	67	20.8	60	25.8	-1
0.9	60	5.9	65	10.9	67	15.9	67	20.9	48	25.9	64
1	60	6	65	11	67	16	65	21	67	26	64
1.1	60	6.1	65	11.1	-1	16.1	65	21.1	67	26.1	64
1.2	60	6.2	65	11.2	65	16.2	65	21.2	67	26.2	64
1.3	48	6.3	-1	11.3	65	16.3	65	21.3	-1	26.3	64
1.4	67	6.4	64	11.4	65	16.4	65	21.4	-1	26.4	-1
1.5	67	6.5	64	11.5	65	16.5	-1	21.5	67	26.5	64
1.6	67	6.6	64	11.6	65	16.6	65	21.6	67	26.6	64
1.7	67	6.7	64	11.7	-1	16.7	65	21.7	67	26.7	64
1.8	-1	6.8	64	11.8	65	16.8	65	21.8	67	26.8	64
1.9	67	6.9	-1	11.9	65	16.9	65	21.9	67	26.9	64
2	67	7	64	12	65	17	65	22	-1	27	64
2.1	67	7.1	64	12.1	65	17.1	65	22.1	69	27.1	62
2.2	67	7.2	64	12.2	65	17.2	64	22.2	69	27.2	62
2.3	67	7.3	64	12.3	-1	17.3	64	22.3	69	27.3	62
2.4	-1	7.4	64	12.4	64	17.4	64	22.4	69	27.4	62
2.5	-1	7.5	-1	12.5	64	17.5	64	22.5	69	27.5	62
2.6	69	7.6	62	12.6	64	17.6	64	22.6	-1	27.6	-1
2.7	69	7.7	62	12.7	64	17.7	-1	22.7	69	27.7	62
2.8	69	7.8	62	12.8	64	17.8	64	22.8	69	27.8	62
2.9	69	7.9	62	12.9	-1	17.9	64	22.9	69	27.9	62
3	-1	8	62	13	64	18	64	23	69	28	62
3.1	-1	8.1	-1	13.1	64	18.1	64	23.1	69	28.1	62
3.2	69	8.2	62	13.2	64	18.2	64	23.2	69	28.2	-1
3.3	69	8.3	62	13.3	64	18.3	64	23.3	-1	28.3	60
3.4	69	8.4	62	13.4	64	18.4	50	23.4	67	28.4	60
3.5	69	8.5	62	13.5	45	18.5	62	23.5	67	28.5	60
3.6	69	8.6	62	13.6	62	18.6	62	23.6	67	28.6	60
3.7	69	8.7	-1	13.7	62	18.7	62	23.7	67	28.7	60
3.8	-1	8.8	60	13.8	62	18.8	62	23.8	67	28.8	60
3.9	67	8.9	60	13.9	62	18.9	62	23.9	67	28.9	60
4	67	9	60	14	62	19	62	24	67		
4.1	67	9.1	60	14.1	62	19.1	62	24.1	67		
4.2	67	9.2	60	14.2	62	19.2	62	24.2	67		
4.3	67	9.3	60	14.3	62	19.3	62	24.3	67		
4.4	67	9.4	60	14.4	62	19.4	62	24.4	67		
4.5	67	9.5	60	14.5	62	19.5	-1	24.5	-1		
4.6	67	9.6	60	14.6	62	19.6	60	24.6	65		
4.7	67	9.7	60	14.7	-1	19.7	60	24.7	65		
4.8	67	9.8	-1	14.8	67	19.8	60	24.8	65		
4.9	-1	9.9	-1	14.9	67	19.9	60	24.9	65		

図12

このようなデータがあったときに以下のように抜粋した.

秒数	MIDI_NUMBER
1.1	60(ド:C4)
1.2	60(ド:C4)
1.3	48(ド:C3)
1.4	67(ソ:G4)
1.5	67(ソ:G4)
1.6	67(ソ:G4)

図13

具体例を示すと

① 図の同じ音が続いている部分の数をカウントする

② カウントした数と0.1秒をかけてやると音符の長さとなる

③ このとき，最初のド4は0.2秒となる

このように，同じ音が何回続いたかで音の長さ(秒)を求めてやる.

・秒から音価に変換する

① 以下の式よりテンポから全音符の長さを求めてやる

全音符の長さ＝240／テンポ

② 音の長さ(秒)を全音符の長さで割ってやる

音価＝音の長さ／全音符の長さ

③ ②で求めたものを拍数と比べてやる(四捨五入)

全音符： 0.875以上

付点2分音符： 0.625以上0.875以下

2分音符： 0.4375以上0.625以下

付点４分音符： 0.3125以上0.4375以下

4分音符： 0.21875以上0.3125以下

付点８分音符： 0.15625以上0.21875以下

8分音符： 0.09375以上0.15625以下

16分音符： 0.046875以上0.09375以下

32分音符： 0.023437以上0.046875以下

音符なし： 0.023437以下

2.3　コード

コードをつなげていくことで，曲の大まかな流れ・雰囲気・曲調を決めることができる．

コード進行には，以下の 2 つの要素がある．

・次の和音の種類（コード進行）

・次の和音に進むまでの長さ

本研究では，次の和音の種類を重点的に行っている．長さについては 1 小節に 1 つの割合でコード作成している[11] [12].

2.3.1　和音(コード)とは

和音とは，高さの違う２つ以上の音同時に響くときの音である．音楽用語であるところの３和音といえば３つ以上の音の響きになるが，どんな音でもいいというわけではない．基準となる音(例としてド)に対し，３度音程の音(ミ)と５度音程の音(ソ)を重ねるというルールがある．このときの音をそれぞれ「根音」「３音」「５音」と呼び，３音と５音を単調や長調に置き換えることで，明るい響きの長３和音や暗い響きの短３和音などを作成することができる．

和音はコードとも呼ばれ，協和音・不協和音が存在する．使われる音の組み合わせをＣやＣｍ７などのように英数字で表したものをコードネームと呼ぶ．ポップス音楽などで譜面上に伴奏を表記するのに便利なものとして広く使われている．また，協・不協和音を組み合わせることによって，さまざまな曲を作成することができる．

2.3.2　ダイアトニック・コード

本研究では，3和音構成のハ長調ダイアトニック･コードを使用した．

ダイアトニック･コードとは，Major Scaleの音をルートとして，その上にひとつ飛ばしで３度音程の３つの音を積み重ねてできたコードのことで，ここで言うMajor Scaleはピアノで言う白鍵盤のことである．ルートというのは，一番下の主になる音のことである．そこに3度と5度を重ねた音となる．下図に示すのが3和音構成Cメジャースケールのコードである[13][14][15].

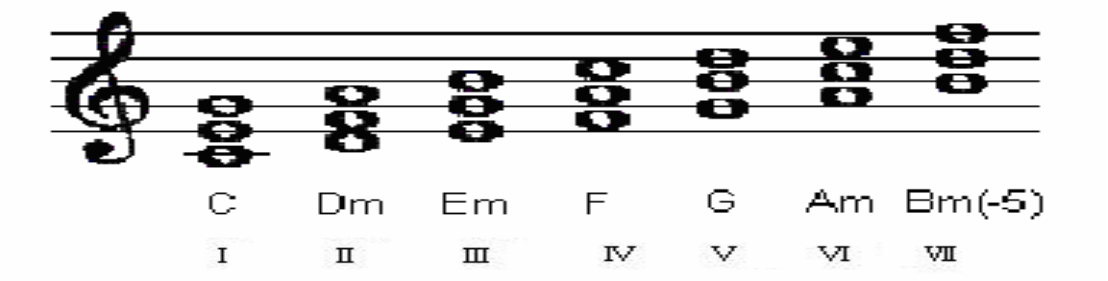


図14

2.3.3　コード抽出手法

本研究では，以下の示す3段階に分けてコードを抽出した．

- FFT
- テンプレートマッチング
- コード進行の基本ルール

まず，FFTによる周波数解析を行い抽出された和音候補列からテンプレートマッチングにより候補を絞る．さらにコード進行の基本ルールを使用して和音を1つに決定する[16] [17]．

2.3.3.1　FFTについて

はじめに，FFTとは高速フーリエ変換のことである．FFTはパワースペクトルを求めることが出来る．このことにより，パワースペクトルに対応した周波数が求まる[18]．

・FFTを行う条件

データ数：	512[点]
間隔：	200[msec]毎
信号周波数：	44.1[kHz]

C2の音程から7オクターブ分の平均律音階に該当する周波数パワーを，1 オクターブの音程群として重ね合わせる．この処理によって，各音程の高次倍音成分が及ぼす悪影響を吸収する効果が得られる．

・実際にFFTを行った結果

コードC（ド・ミ・ソ）を抽出した結果

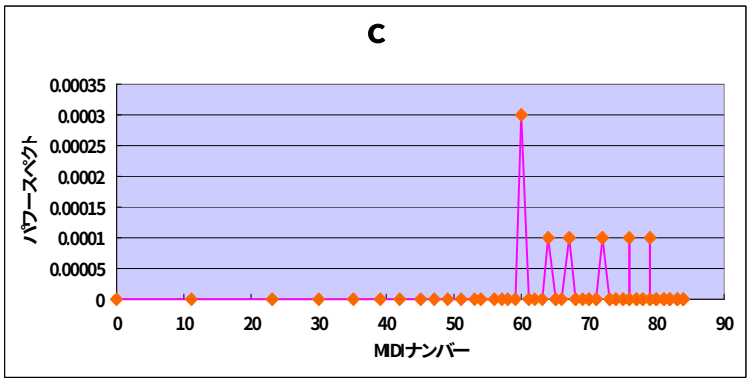


図15

このグラフから，60(ドの4オクターブ)，64(ミの4オクターブ)，67(ソの4オクターブ)の部分にスペクトルがたっていることがわかる．次に，72(ドの5オクターブ)から上は，倍音成分なので一つ下のオクターブ部分に重ね合わせる．

2.3.3.2 テンプレートマッチング

本研究では，ハ長調の3和音構成ダイアトニック・コードを使用しているので，以下の7つのテンプレートを用意した.

・テンプレート

```
//音高      C   C#  D   D#  E   F   F#  G   G#  A   A#  B
C         { 1 , 0 , 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 0 }//Ⅰ
Dm        { 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 1 , 0 , 0 }//Ⅱ
Em        { 0 , 0 , 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 1 }//Ⅲ
F         { 1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 1 , 0 , 0 }//Ⅳ
G         { 0 , 0 , 1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 1 }//Ⅴ
Am        { 1 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 }//Ⅵ
Bdim      { 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 1 }//Ⅶ
```

このテンプレートは，それぞれ和音に含まれる音の部分に1を含まれない部分には0を配置している.

上のテンプレートとFFTにより抽出したデータを以下の式により，相関係数が最大のものを見つけ，テンプレートマッチングを行う.

$$R = \frac{\sum_{i=0}^{n-1}(f[i]-\overline{f})(t[i]-\overline{t})}{\sqrt{\sum_{i=0}^{n-1}(f[i]-\overline{f})^2}\sqrt{\sum_{i=0}^{n-1}(t[i]-\overline{t})^2}}$$

このときのｆ(ｉ)はFFTにより抽出されたデータを関数としたもので，ｔ(ｉ)がテンプレートである． ｆのバーを関数ｆの平均，ｔのバーをテンプレートｔの平均である.

2.3.3.3　コード進行の基本ルールについて

テンプレートマッチングを行う場合に画像と違い1次元なので，正の最大相関係数がいくつか現れることがある．こうした場合に，コード進行の基本ルールを使い1つに絞る．

・コード進行基本ルール

図14に示された　Ⅰ，Ⅱ，Ⅲ，Ⅳ，Ⅴ，Ⅵ，Ⅶの数字でルールを示す．

基本ルール

- ルール①「Ⅰ」の次は，どのコードでもよい
- ルール②「Ⅱ」の次は，必ず「Ⅴ」
- ルール③「Ⅳ」の次に，「Ⅵ」はこない
- ルール④「Ⅴ」の次は，「Ⅰ」または「Ⅵ」
- ルール⑤「Ⅵ」の次に，「Ⅰ」はこない

図16

上図のルールに従って，テンプレートマッチングによって抽出された和音候補列の中から１つ和音を決定する．ここで，コード進行の基本ルールに反するコードしか和音候補列の中にない場合，その抽出コードの中からランダムで選択するシステムとなっている．

・結果の具体例

前の結果：Dm

[FFTの結果]　　[相関係数]

C	1	R[c]=	0	コード→F
C#	0	R[dm]=	0.816497	
D	1	R[em]=	-0.40825	
D#	0	R[f]=	0.816497	
E	0	R[g]=	0	
F	1	R[am]=	0.408248	
F#	0	R[bdim]=	0.408248	
G	0			
G#	0			
A	1			
A#	0			
B	0			

図17

このとき，DmとFの相関値が高い．しかし，ルールよりここではFが選択された．

3. データベースについて

データベースは，それぞれ特徴ごとに分けられて，CSVファイルによって保存される．

- 度数差：　　１次元
- コード：　　２次元
- 音価：　　２次元

のデータベースによって作られる[1][2][3][4].

3.1　メロディー（度数差）

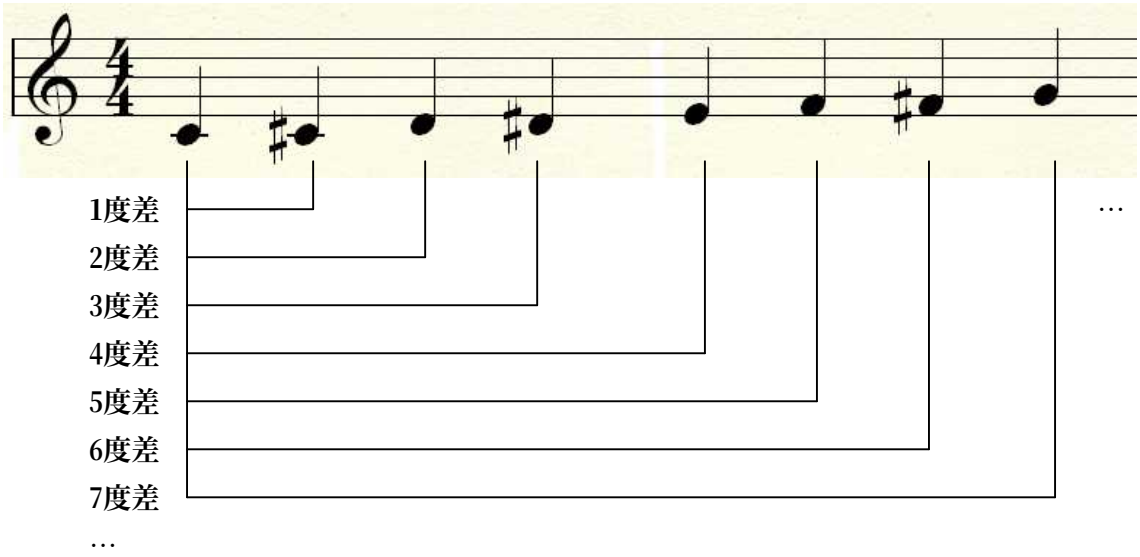


図18

上図について，それぞれ半音ごとに度数が１つ上がるようになっている．
ここで，度数差のグラフ例を表すと左図のようになる．
このグラフを見ると，2度上がる回数が多いことがわかる．以外に12度差もあるところが，このアーティストの特徴である．

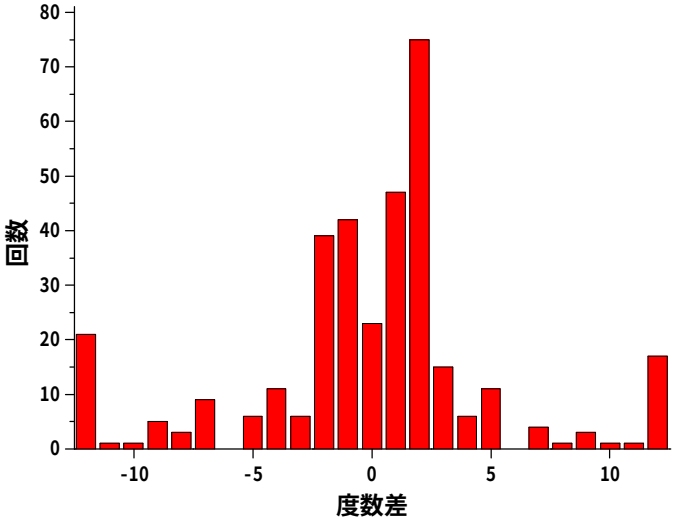


図19

3.2　音価

音価についてもコードと同様に，一つ前の音価とその時点での音価の推移をみて，それをデータベースにする．

音価データベースの具体例を以下に示す．

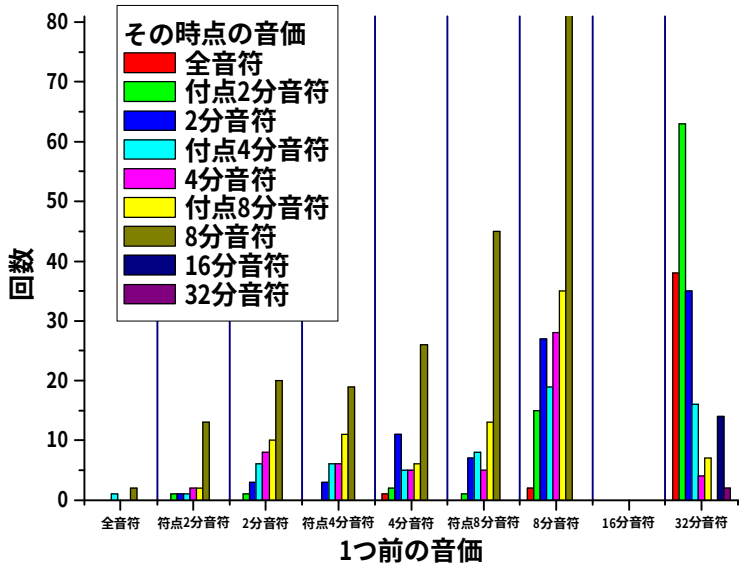


図21

3.3　コード

一つ前のコードとその時点でのコードの推移をみて，それをデータベースとする．

コードデータベースの具体例を以下に示す．

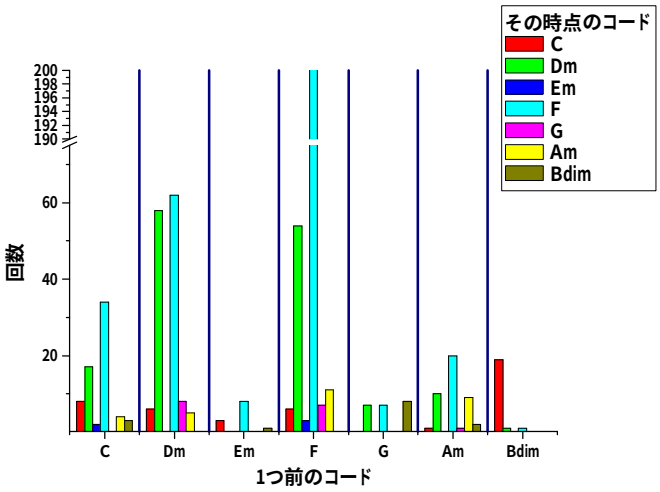


図20

4. 作曲について

4.1 アウトラインの作成

通常の楽曲，特にポップス系の曲ではパターン化されたテーマ（４小節程度のフレーズ）の繰り返しとエンディングなどで曲が構成されている．このステップでは，乱数を用いて曲の大まかな構成を作成する．

自動作曲システムでは，イントロは作成せずに乱数を用いてパターンを決定する．エンディングはどの曲も同じかたちで作る．

本研究では，3パターンのアウトラインをテンプレートとして用意した．そのテンプレートの中から乱数により決定される[19][20]．

・テーマパターンテンプレート

図中のAはAメロ，BはBメロ，CはCメロ，Sはサビのことである．

パターンⅠ	A	B	A	B	S	A	C	S	S
パターンⅡ	A	A	S	A	A	S	B	C	S
パターンⅢ	A	B	S	A	S	A	B	C	S

本研究では，以下のようにテーマの長さを定義した．

Aメロ：　4小節
Bメロ：　3小節
Cメロ：　2小節
サビ：　4小節

4.2　曲の作成

本研究では，累積分布と乱数を用いて作曲を行っている．

・累積分布について

確率変数 x が $-\infty < x < \infty$ の x に対して， x 以下の値を取る確率

$F(x) = Pr(X \leqq x)$

を x の関数としたものを 確率変数 X の（累積）分布関数という．

連続型の確率変数でも，離散型の確率変数でも区別無く定義できる．

値域（関数の値の範囲）

$0 \leqq F(x) \leqq 1$

・具体例(度数差で示す)

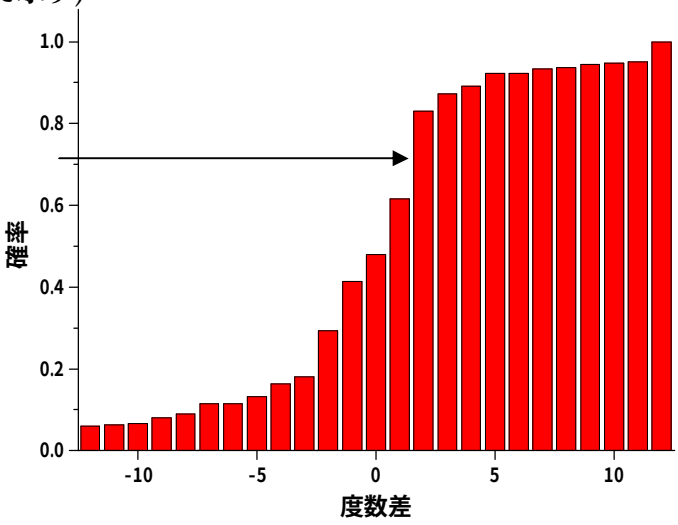


図 22

上図がデータベースより作成した度数差の累積分布である．

小さいほうから，次の事象にその起こる確率を足す．そして，次々と足し合わせていき最終的には，確率は 1 となる．

・データの作成方法

図 22 では，乱数により矢印のような値が得られたとき，矢印と衝突する事象が，成り立つようにデータを作成する．図 22 の場合は 2 度差のデータとなる．

4.2.1 メロディー

テーマごとの音価の長さに合わせて，メロディーを作成する．まず，出だしの音高を決定する．次に，累積分布と乱数により求めた事象(度数差)を用いてその一つ前のデータから求めた度数差だけ離れた音高を選択し，データとする．

このとき，ハ長調なので＃(シャープ)音が出てきた場合にそのデータを上下どちらかに半音ずらすことによってデータを得ている．さらにスムーズにメロディーが聞こえるように 10 度差以上のデータが出来ないような処理をしている．そして，3，4，5 オクターブの間にデータ列が出来るような処理もしている．

4.2.2 音価(リズム)

音価も，最初に出だしのコードを決定する．音価の場合は（A，B，C）メロ，サビの 4 テーマについてすべて 4 分音符か 8 分音符で始まるように設計した．

次に，乱数と累積分布表により，データの作成をする．このとき，本システムでは 4/4 拍子を採用しているので，A メロ，B メロ，C メロ，サビについて，1 小節あたりに全音符 1 個分の長さをその小節分の長さだけのメロディーを作成する．テーマごとの長さも抽出する[21] [22]．

4.2.3 コード進行

本研究では，1 小節に 1 つの割合でコードのデータを作成する．まず，出だしのコードを決定する．今回はハ長調のため出だしのコードは C となる．

次に，乱数と累積分布表により，データの作成をする．このとき，A メロ，サビについてはこの方法をとるが，B メロ，C メロについてはランダムで決定される．これは，まったく同じ曲にならないようにするためである．

5. 実験及び評価

19 というアーティストの曲をピアノで弾いたものを 3 曲～10 曲の WAVE フォーマット形式のデータとして取り込んだ．取り込んだ曲を MIDI により再構築することによって，抽出できているかを確認する．次に自動作曲によりできた曲(10 曲)についてのアンケートをとって自動作曲システムの評価を行った[23].

5.1 入出力ファイルについて

入力：　　WAVE ファイル，44100[Hz]，モノラル

出力：　　MIDI ファイル[24]

5.2 評価方法

・アンケート内容

1．曲として成り立っているか？

2．好きなフレーズはあったか？

3．この曲は人よりの作曲かそれとも PC よりか？

4．19 のようなフレーズはいくつあったか？

1 から 3 の質問については 5 段階評価（1：悪い，5：良い）で 4 の質問についてはフレーズ数を数える形式のアンケートを行った.

被験者については 5 人の方に協力をいただいた.

5.2 評価の結果

・曲に対する評価の結果

曲一つ一つの質問にたいする平均と標準偏差を以下の表に示す.

1 曲目	質問	1	2	3	4
	平均	3	2.2	2.6	3.6
	標準偏差	1	1.095445	0.894427	2.50998

2 曲目	質問	1	2	3	4
	平均	2.6	2	2.6	3.4
	標準偏差	0.547723	0.707107	0.894427	2.302173

3 曲目	質問	1	2	3	4
	平均	3	2.4	2.8	3
	標準偏差	1	0.547723	0.83666	2.236068

4 曲目	質問	1	2	3	4
	平均	3.8	3.2	3.2	4.6
	標準偏差	0.83666	0.83666	1.095445	3.646917

5 曲目	質問	1	2	3	4
	平均	2.2	2.2	1.8	2.4
	標準偏差	0.447214	0.83666	0.447214	1.516575

6 曲目	質問	1	2	3	4
	平均	2.6	2.2	2	2
	標準偏差	0.894427	1.095445	0	1.870829

7 曲目	質問	1	2	3	4
	平均	2.2	2.2	1.6	1.6
	標準偏差	0.447214	0.83666	0.547723	0.894427

8 曲目	質問	1	2	3	4
	平均	1.8	2	1.6	1.2
	標準偏差	0.83666	0.707107	0.547723	1.30384

9 曲目	質問	1	2	3	4
	平均	1.4	1.4	1.6	0.4
	標準偏差	0.547723	0.547723	0.547723	0.547723

10 曲目	質問	1	2	3	4
	平均	2	1.8	2.2	1
	標準偏差	0.707107	0.83666	0.447214	1.414214

・質問に対する評価の結果

このデータは，10 曲の平均のデータをまとめて，平均の平均を出したものである．

質問	1	2	3	4
被験者 1	2	1.9	2.5	2.2
被験者 2	2.4	2	2	2
被験者 3	3.3	1.4	2.1	0
被験者 4	2.5	2.7	2.5	2.9
被験者 5	2.1	2.8	1.9	4.5
平均	2.46	2.16	2.2	2.32
標準偏差	0.512835	0.585662	0.282843	1.62696

5.3 実験及び評価の結果・考察

まず，曲の再構築によりメロディーラインと音かについてはある程度綺麗に抽出できていることが確認できた．そして評価により，平均して 2 フレーズくらい類似性があるような曲を生成することが出来た．評価が高かったものには，メロディーの流れがスムーズであり，コードとメロディの奏でるハーモニーが含まれていたからであると考えられる．それに対して，あまり評価が高くなかったものに対しては，メロディーの流れがとびとびで，メロディーとして成り立っていなかったためと考えられる．成立しなった理由としては，累積分布には従っているがランダムのみでデータを決定したことによるものだと推定できる．したがって，学習機能の導入が必要だと考えられる．

6. まとめ

本研究では WAVE ファイルを読み込み，そこからメロディー，リズム，コード進行の特徴について抽出した．そして抽出した特徴をデータベースに変換して，データベースより類似性のある曲の自動作曲ができるツールを作成した．作成した自動作曲システムを用いて実際に曲を取り込み自動作曲した結果，出力された音楽データでアンケートをとって評価を行った．

得られた評価は，５段階中 2 から 3 の間であった．評価があまり高くなかったのは学習機能がなかったためと考えられる．したがって今後の課題としては，このツールには遺伝的アルゴリズムやニューラルネットワークなどを用いた学習機能の導入などが考えられる．

7. 参考文献

[1] DTMMAGAZINE2 2000 vol.68

[2] DTMMAGAZINE2 2000 vol.71

[3] DTMMAGAZINE2 2000 vol.76

[4] DTMMAGAZINE2 2000 vol.77

[5] パターンマッチング法による選曲システム
東京工科大学 工学部 情報工学科 千種研究室
堀越 葉子　羽山 房枝
http://www.teu.ac.jp/chiit/hayama/Text/half_01.html

[6] コンピュータ音楽　歴史・テクノロジーアート　[著]　Curtis Roads

[7] 音の性質と解析：自己相関関数[著]　阿部　洋
http://www.nda.ac.jp/cc/mse/_development/Abe/ACF.pdf

[8] 自己相関関数入門　http://www.ymec.com/hp/signal/acf.htm

[9] 音階と周波数の関係 http://tissoil.orz.hm/~soil/?co=audio_ex3

[10] 音価について http://www001.upp.so-net.ne.jp/howtobass/basslesson/4-03.htm

[11] 絶対わかる！コード理論　[著]　北川　祐

[12] 作曲講座第３回　http://www3.plala.or.jp/mitt/compose/compose03.htm

[13] 和声学　http://www.minehara.com/ssh/harmony9.htm

[14] コードの連結　http://www3.ocn.ne.jp/~b-winds/gakuten/jimiti/jimiti21.htm

[15] コード進行について　http://www.guitarholic.com/fre/fre_12.html

[16] 楽音信号からの和音進行抽出手法と類似楽曲検索への応用　[著] 我山真一

[17] 純正律音階と平均律音階　http://gabacho.reto.jp/whims/whim0010.html

[18] 高専学生のためのディジタル信号処理

[19] 自動作曲システム　http://mu-tech.co.jp/music/mus111j.html

[20] Binary-ID Music Studio　http://www.binary-id.net/

[21] 確率モデルによる MIDI 演奏のテンポ変化時点の検出
[著]　◎武田晴登　西本卓也　嵯峨山茂樹（東大・情報理工）

[22] リズム語彙を用いた HMM による MIDI 演奏のリズムとテンポ推定
[著]　武田晴登　西本卓也　嵯峨山茂樹（東大・情報理工）

[23] 歌 BON 月号別冊付録　19 全曲集

[24] MIDI 楽器名一覧表　http://www2u.biglobe.ne.jp/~rachi/midinst.htm

8. 付録

・プログラムリスト

[自己相関関数(メロディー)]

```
int acf(short *buf_in,double *box) //buf_in→原関数, box→出力先
{
 int L,k;//L＝τ
 double m=0;int T=0;
        for(L=0;L<TAU;L++) //TAU=440
        {
          for(k=0;k<N;k++) //N=4400
           {
              box[L]+= (((double)buf_in[k]*(double)buf_in[k+L]))/(SHRT_MAX/2);
           }
           box[L]=box[L]/N;
           if((box[L]>m)&&(L>70)) {m=box[L];T=L;}
        }
        return T;
}
```

[音価抽出部分]

```
for(interval_acf=0.0;interval_acf<=time;interval_acf+=0.1)
{
        judge=-1;
        for(k=0;k<(int)(wavefmt->rate*0.1);k++){
                if( fabs(*(buf_in_d+((int)(interval_acf*wavefmt->rate))+k)) >= EPSILON*500 ){
                        judge=1;
                        break;
                }
        }
        if(judge==-1){
                justice_max_M[i]=-1;
                midi_acf[j].lng_opt++;
                j++;
        }
        else{
                justice_max_T[i]=acf(buf_in+((int)(interval_acf*wavefmt->rate)),box);
                justice_max_F[i]=1.0/((1.0/wavefmt->rate)*justice_max_T[i]);
                //MIDIナンバー変換始め
                MIDI_N_d=(double)(12*(log((justice_max_F[i]/440.0))/log(2.0))+69);
                justice_max_M[i]=(int)(MIDI_N_d+0.5);
                if( w < 0 ) w = justice_max_M[i];
                //MIDIナンバー変換終わり
                if(w!=justice_max_M[i]){
                        j++;
                }
                midi_acf[j].lng_opt++;
                w=justice_max_M[i];
                i++;
        }
        int data_number=j;
```

```
        for(i=0;i<data_number;i++){
                if(i==0){
                        midi_acf[i].MIDI_Merody=justice_max_M[i];
                }
                else{
                        midi_acf[i].MIDI_Merody=justice_max_M[count_same];
                }
                count_same+=(int)(midi_acf[i].lng_opt);//同じ音符の個数をカウント
                midi_acf[i].lng_opt=midi_acf[i].lng_opt*0.1;//0.1秒間隔だから×
        }
}
```

[FFT後のコード抽出]

```
void templ(int *opt,double *R,char *code,const char *code_front,int kk)
{
//テンプレート
//                                                      C   C#  D   D#  E   F   F#  G   G#  A
A#  B
int C_templ[12]             = { 1 , 0 , 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 0 };//Ⅰ
int Dm_templ[12]   = { 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 1 , 0 , 0 };//Ⅱ
int Em_templ[12]   = { 0 , 0 , 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 1 };//Ⅲ
int F_templ[12]             = { 1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 1 , 0 , 0 };//Ⅳ
int G_templ[12]             = { 0 , 0 , 1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 1 };//Ⅴ
int Am_templ[12]   = { 1 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 };//Ⅵ
int Bdim_templ[12] = { 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 1 };//Ⅶ
double   Uc=0,Lc_1=0,Lc_2=0,
                Udm=0,Ldm_1=0,Ldm_2=0,
                Uem=0,Lem_1=0,Lem_2=0,
                Uf=0,Lf_1=0,Lf_2=0,
                Ug=0,Lg_1=0,Lg_2=0,
                Uam=0,Lam_1=0,Lam_2=0,
                Ubdim=0,Lbdim_1=0,Lbdim_2=0;
double f_bar=0,t_bar_C=0,t_bar_Dm=0,t_bar_Em=0,t_bar_F=0,t_bar_G=0,t_bar_Am=0,t_bar_Bdim=0;
int i,j,k;
int MAX;
int input[12];
        for(k=0;k<12;k++){
                input[k]=opt[k];
        }
//平均の計算
        for(i=0;i<12;i++){
                f_bar+=input[i];
                t_bar_C+=C_templ[i];
                t_bar_Dm+=Dm_templ[i];
                t_bar_Em+=Em_templ[i];
                t_bar_F+=F_templ[i];
                t_bar_G+=G_templ[i];
                t_bar_Am+=Am_templ[i];
                t_bar_Bdim+=Bdim_templ[i];
        }
        MAX=12;
        f_bar=f_bar/MAX;
        t_bar_C=t_bar_C/MAX;
        t_bar_Dm=t_bar_Dm/MAX;
        t_bar_Em=t_bar_Em/MAX;
        t_bar_F=t_bar_F/MAX;
```

```
        t_bar_G=t_bar_G/MAX;
        t_bar_Am=t_bar_Am/MAX;
        t_bar_Bdim=t_bar_Bdim/MAX;
//0 の例外処理
        if(f_bar<=0)                f_bar=0;
        if(t_bar_C<=0)              t_bar_C=0;
        if(t_bar_Dm<=0)             t_bar_Dm=0;
        if(t_bar_Em<=0)             t_bar_Em=0;
        if(t_bar_F<=0)              t_bar_F=0;
        if(t_bar_G<=0)              t_bar_G=0;
        if(t_bar_Am<=0)             t_bar_Am=0;
        if(t_bar_Bdim<=0)  t_bar_Bdim=0;
//Σ計算
        for(j=0;j<12;j++){
//C
                Uc+=(input[j]-f_bar)*(C_templ[j]-t_bar_C);
                Lc_1+=pow((input[j]-f_bar ),2);
                Lc_2+=pow((C_templ[j]-t_bar_C ),2);
//Dm
                Udm+=(input[j]-f_bar)*(Dm_templ[j]-t_bar_Dm);
                Ldm_1+=pow((input[j]-f_bar ),2);
                Ldm_2+=pow((Dm_templ[j]-t_bar_Dm ),2);
//Em
                Uem+=(input[j]-f_bar)*(Em_templ[j]-t_bar_Em);
                Lem_1+=pow((input[j]-f_bar ),2);
                Lem_2+=pow((Em_templ[j]-t_bar_Em ),2);
//F
                Uf+=(input[j]-f_bar)*(F_templ[j]-t_bar_F);
                Lf_1+=pow((input[j]-f_bar ),2);
                Lf_2+=pow((F_templ[j]-t_bar_F ),2);
//G
                Ug+=(input[j]-f_bar)*(G_templ[j]-t_bar_G);
                Lg_1+=pow((input[j]-f_bar ),2);
                Lg_2+=pow((G_templ[j]-t_bar_G ),2);
//Am
                Uam+=(input[j]-f_bar)*(Am_templ[j]-t_bar_Am);
                Lam_1+=pow((input[j]-f_bar ),2);
                Lam_2+=pow((Am_templ[j]-t_bar_Am ),2);
//Bdim
                Ubdim+=(input[j]-f_bar)*(Bdim_templ[j]-t_bar_Bdim);
                Lbdim_1+=pow((input[j]-f_bar ),2);
                Lbdim_2+=pow((Bdim_templ[j]-t_bar_Bdim ),2);
        }
//相関係数を求める
        R[0]=Uc/((sqrt(Lc_1))*(sqrt(Lc_2)));
        R[1]=Udm/((sqrt(Ldm_1))*(sqrt(Ldm_2)));
        R[2]=Uem/((sqrt(Lem_1))*(sqrt(Lem_2)));
        R[3]=Uf/((sqrt(Lf_1))*(sqrt(Lf_2)));
        R[4]=Ug/((sqrt(Lg_1))*(sqrt(Lg_2)));
        R[5]=Uam/((sqrt(Lam_1))*(sqrt(Lam_2)));
        R[6]=Ubdim/((sqrt(Lbdim_1))*(sqrt(Lbdim_2)));
//休符の例外処理をして最大相関係数を求める
        for(k=0;k<7;k++){
                if(R[k]<=-1)
                        R[k]=999;
        }
```

```
double max=R[0];
int sel=0,sel_hokan[7];
memset(sel_hokan, -1,sizeof(int)*7);
        for(j=1;j<7;j++){
                if(R[j]==999){
                        sel= -1;
                        break;
                }
                else{
                        if(R[j]>=max){
                                max=R[j];
                        }
                }
        }
        j=0;
        if(sel!= -1){
                for(i=0;i<7;i++){
                        if(R[i]==max){
                                sel_hokan[j]=i;
                                j++;
                        }
                }
        }
int w_j=j;
//コード進行の基本ルールの判定
        if(sel!= -1){
                if(strcmp( code_front , "C" )==0)
                        sel=C_hantei(sel_hokan,w_j);
                if(strcmp( code_front , "Dm" )==0)
                        sel=Dm_hantei(sel_hokan,w_j);
                if(strcmp( code_front , "Em" )==0)
                        sel=Em_hantei(sel_hokan,w_j);
                if(strcmp( code_front , "F" )==0)
                        sel=F_hantei(sel_hokan,w_j);
                if(strcmp( code_front , "G" )==0)
                        sel=G_hantei(sel_hokan,w_j);
                if(strcmp( code_front , "Am" )==0)
                        sel=Am_hantei(sel_hokan,w_j);
                if(strcmp( code_front , "Bdim" )==0)
                        sel=Bdim_hantei(sel_hokan,w_j);
                if(strcmp( code_front , "No" )==0)
                        sel=C_hantei(sel_hokan,w_j);
        }
//セレクト
        switch(sel){
                case 0:  strcpy(code,"C");break;
                case 1:  strcpy(code,"Dm");break;
                case 2:  strcpy(code,"Em");break;
                case 3:  strcpy(code,"F");break;
                case 4:  strcpy(code,"G");break;
                case 5:  strcpy(code,"Am");break;
                case 6:  strcpy(code,"Bdim");break;
                default : strcpy(code,"No");break;
        }
}
```

[データベースの作成]

```
void make_db(DATA *data,int total_number,int ii,int **db_code,int *db_mero,int data_number,int
**db_lng)
{
int l=0,m=0,i,j;
char w_code[5],w_code_front[5];
//w_~→その時点のデータ,wf_~→一つ前のデータ
//コードデータベース
        strcpy(w_code_front,data[ii].code[0]);
        for(i=1;i<total_number;i++){
                strcpy(w_code,data[ii].code[i]);
                if(strcmp(w_code_front,"No")!=0){
                        if(strcmp( w_code_front , "C" )==0)
                                l=0;
                        if(strcmp( w_code_front , "Dm" )==0)
                                l=1;
                        if(strcmp( w_code_front , "Em" )==0)
                                l=2;
                        if(strcmp( w_code_front , "F" )==0)
                                l=3;
                        if(strcmp( w_code_front , "G" )==0)
                                l=4;
                        if(strcmp( w_code_front , "Am" )==0)
                                l=5;
                        if(strcmp( w_code_front , "Bdim" )==0)
                                l=6;
                        if(strcmp( w_code , "C" )==0)
                                m=0;
                        if(strcmp( w_code , "Dm" )==0)
                                m=1;
                        if(strcmp( w_code , "Em" )==0)
                                m=2;
                        if(strcmp( w_code , "F" )==0)
                                m=3;
                        if(strcmp( w_code , "G" )==0)
                                m=4;
                        if(strcmp( w_code , "Am" )==0)
                                m=5;
                        if(strcmp( w_code , "Bdim" )==0)
                                m=6;
                        db_code[l][m]++;
                }//if
                strcpy(w_code_front,data[ii].code[i]);
        }//for
int *w_data_midi_lng,w_i;
w_data_midi_lng=(int* )malloc(sizeof(int )*data_number);
        i=0;
        for(j=0;j<data_number;j++){
                if(data[ii].MIDI_LNG[j]!=0){
                        w_data_midi_lng[i]=data[ii].MIDI_LNG[j];
                        i++;
                }
        }
        w_i=i;
int a,b;
int w_MIDI_N,wf_MIDI_N;
```

```
int w_LNG,wf_LNG;
int seni;
//メロディーデータベース
        wf_LNG=w_data_midi_lng[0];
        for(j=1;j<data_number;j++){
                wf_MIDI_N=data[ii].MIDI_N[j-1];
                w_MIDI_N=data[ii].MIDI_N[j];
                seni=wf_MIDI_N-w_MIDI_N;
                if(w_MIDI_N==-1){
                        seni=25;
                }
                else{
                        if(wf_MIDI_N<55||wf_MIDI_N>76){
                                continue;
                        }
                        if(w_MIDI_N<55||w_MIDI_N>71){
                                continue;
                        }
                        if(seni<0){
                                seni=abs(seni)+12;
                        }
                        db_mero[seni]++;
                }
        }
//音価データベース
        for(j=1;j<w_i;j++){
                w_LNG=w_data_midi_lng[j];
                switch(wf_LNG){
                        case 1 :a=0;break;
                        case 3 :a=1;break;
                        case 2 :a=2;break;
                        case 5 :a=3;break;
                        case 4 :a=4;break;
                        case 9 :a=5;break;
                        case 8 :a=6;break;
                        case 16:a=7;break;
                        case 32:a=8;break;
                        default:break;
                }
                switch(w_LNG){
                        case 1 :b=0;break;
                        case 3 :b=1;break;
                        case 2 :b=2;break;
                        case 5 :b=3;break;
                        case 4 :b=4;break;
                        case 9 :b=5;break;
                        case 8 :b=6;break;
                        case 16:b=7;break;
                        case 32:b=8;break;
                        default:break;
                }
                wf_LNG=w_data_midi_lng[j];
                db_lng[a][b]++;
        }
}
```

[アウトラインの作成]

```
void outline(char pattern[])
{
char pattern1[10]={'A','B','A','B','S','A','C','S','S','\0'};//パターンⅠ
char pattern2[10]={'A','A','S','A','A','S','B','C','S','\0'};//パターンⅡ
char pattern3[10]={'A','B','S','A','S','A','B','C','S','\0'};//パターンⅢ
//セレクト
int select;
        select=rand()%3+1;
        switch(select){
        case 1:strcpy(pattern,pattern1);break;
        case 2:strcpy(pattern,pattern2);break;
        case 3:strcpy(pattern,pattern3);break;
        }
}
```

[作曲]

```
//それぞれはじめのデータが決まっている.
//新たなコードの作成
void new_code(int x_hokan,double **bunpu_code,double *bunpu_code0,char **new_code,int nodule)
{

        double rand_pro;
        int y_hokan;
        int i,y;

        for(i=1;i<nodule;i++){
                rand_pro=(double)rand()/(double)RAND_MAX;
                for(y=0;y<7;y++){
                        if((rand_pro<=bunpu_code[x_hokan][y])||(bunpu_code0[x_hokan]==0)){
                                y_hokan=y;
                                if(bunpu_code0[y]==0){
                                        y_hokan=rand()%7;
                                }
                                switch(y){
                                        case 0:  strcpy(new_code[i],"C");break;
                                        case 1:  strcpy(new_code[i],"Dm");break;
                                        case 2:  strcpy(new_code[i],"Em");break;
                                        case 3:  strcpy(new_code[i],"F");break;
                                        case 4:  strcpy(new_code[i],"G");break;
                                        case 5:  strcpy(new_code[i],"Am");break;
                                        case 6:  strcpy(new_code[i],"Bdim");break;
                                }
                                x_hokan=y;
                                break;
                        }
                }
        }


}

//新たな音価の作成
int new_lng(int first_lng,double **bunpu_lng,double *bunpu_lng0,int *new_lng,int nodule)
{
        double rand_pro;
```

```
	int i,y,y_hokan;
	int x_hokan=first_lng;
	double haku=0.0;
	new_lng[0]=first_lng;
	i=1;
	while(haku<(double)(nodule*4.0)){
		rand_pro=(double)rand()/(double)RAND_MAX;
		for(y=0;y<9;y++){
			if((rand_pro<=bunpu_lng[x_hokan][y]) ||(bunpu_lng0[x_hokan]==0)){
				y_hokan=y;
				if(bunpu_lng0[y]==0){
					y_hokan=rand()%9;
				}
				switch(y_hokan){
					case 0:haku+=4;break;
					case 1:haku+=3;break;
					case 2:haku+=2;break;
					case 3:haku+=1.5;break;
					case 4:haku+=1;break;
					case 5:haku+=0.75;break;
					case 6:haku+=0.5;break;
					case 7:haku+=0.25;break;
					case 8:haku+=0.125;break;
				}
				//printf("%4.3lf\n",haku);
				new_lng[i]=y_hokan;
				x_hokan=y_hokan;
				break;
			}
		}
		i++;
	}

	return i;
}

//新たなメロディの作成
void new_mero(int first_mero,double *bunpu_mero,int *new_mero,int num)
{
	double rand_pro;
	int i,j;
	new_mero[0]=first_mero;
	for(i=1;i<num;i++){
		rand_pro=(double)rand()/(double)RAND_MAX;
		for(j=0;j<=25;j++){
			if(new_mero[i-1]==-1){
				new_mero[i]=rand()%12+60;
				break;
			}
			if(rand_pro<=bunpu_mero[j]){
				if(j<13){
					new_mero[i]=new_mero[i-1]+j;
				}
				else{
					if(j==25){
						new_mero[i]=-1;
```

```
                                        break;
                                }
                                else{
                                        new_mero[i]=new_mero[i-1]-j+12;
                                }
                        }
//音とびをなくすための処理
                        if(new_mero[i]<55){
                                new_mero[i]+=12;
                        }
                        if(new_mero[i]>76){
                                new_mero[i]-=12;
                        }
                        if(abs(new_mero[i-1]-new_mero[i])>10){
                                continue;
                        }
//#が入らないようにする処理
if(new_mero[i]==56||new_mero[i]==58||new_mero[i]==61||new_mero[i]==63||new_mero[i]==66||
                new_mero[i]==68||new_mero[i]==70||new_mero[i]==73||new_mero[i]==75){
                                switch(rand()%2){
                                case 0:new_mero[i]-=1;break;
                                case 1:new_mero[i]+=1;break;
                                }
                        }
                        break;
                }
        }
    }
}

[エンディング]
int han;
han=rand()%2;
maeda.set_code( CODE_F );
if(han==0){
        maeda.set_note( 0 , new_S_mero[s_num-1]/12 , LNG_4 , slur );
        maeda.set_note( 2 , new_S_mero[s_num-1]/12 , LNG_8 , slur );
        maeda.set_note( 4 , new_S_mero[s_num-1]/12 , LNG_8 , true );
        maeda.set_note( 5 , new_S_mero[s_num-1]/12 , LNG_4 , slur );
        maeda.set_code( CODE_G );
        maeda.set_note( 7 , new_S_mero[s_num-1]/12 , LNG_4 , slur );
        maeda.set_note( 4 , new_S_mero[s_num-1]/12 , LNG_4 , slur );
        maeda.set_note( 2 , new_S_mero[s_num-1]/12 , LNG_4 , true );
        maeda.set_code( CODE_C);
        maeda.set_note( 0 , new_S_mero[s_num-1]/12 , LNG_2d , slur );
}
else{
        maeda.set_note( 4 , new_S_mero[s_num-1]/12 , LNG_4 , slur );
        maeda.set_note( 2 , new_S_mero[s_num-1]/12 , LNG_4 , slur );
        maeda.set_note( 0 , new_S_mero[s_num-1]/12 , LNG_4 , slur );
        maeda.set_note( 11 , new_S_mero[s_num-1]/12-1 , LNG_4 , true );
        maeda.set_code( CODE_C);
        maeda.set_note( 0 , new_S_mero[s_num-1]/12 , LNG_1 , slur );
}
```