音響和音

音色と和音

アコースティックな楽音による伝統的な作曲法において、和音(chord)は複数の音程の音が同時に響きあっている状態のことであり、その和音の連結方法(和声、harmony)を理論化したものが和声法であった。そこでは楽器の音色と音の重ね合わせ方(和音)は区別され、和声法自体が楽音そのものが有している倍音関係をベースに構成されていた。

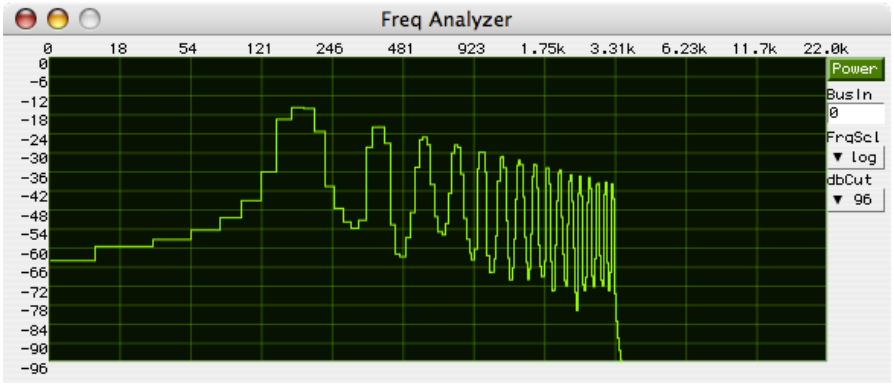
サイン波やクリック、ノイズといった音響素材に、アコースティックな楽音のような倍音列(上音列)としての音色はない。それらはより根源的な音響素材であり、和音や和声を倍音を前提に考える必要はない。そこから生み出される音響に、音色を構成する倍音列と和音を構成する音程の積み重ねの区別はない。

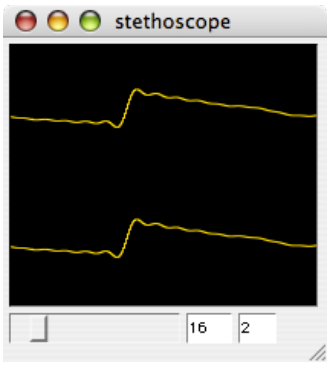
和音と音色が未分化のまま構成される「音響和音」には、古典的な意味における和音の連鎖のみならず、倍音周波数の音量調整による加算合成のような、さまざまな音響合成法による音色の生成も含まれる。音程の重ねあわせによる和音と、倍音の重ね合せによる音色の2つをいかにして1つの統合的な枠組みの中で扱うか–そこに音響和音のポイントがある。

倍音列とその響き

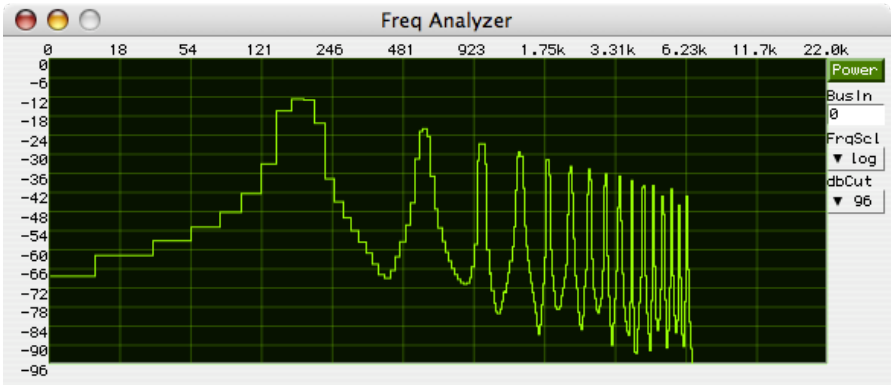
最初に倍音列、すなわち基音とその整数倍の高さ(周波数)の線の重ね合せを聞いてみる。倍音列の周波数は初項と公差を基音の周波数とする等差数列で表現できる。

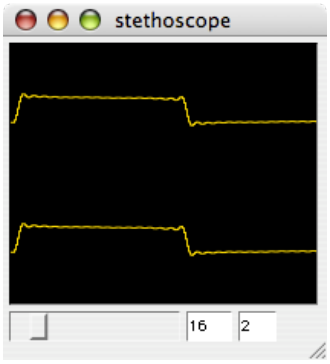
```
// 16本の線を重ねて鳴らす関数を定義する
(
~out = {arg base = 441, fscale = 1, foffset = 0, pan = 0, level = 0.8;
        var farray, aarray;
        farray = Control.names(\farray).kr(Array.fill(16, 0)) * base;
        aarray = Control.names(\aarray).kr(Array.fill(16, 1).normalizeSum);
        Pan2.ar(Klang.ar(`[ farray, aarray, nil ], fscale, foffset), pan, level) };
)

// 200[Hz]とする第16倍音までの重ね合せ(振幅一定)
(
~out.setn(\base, 200,
          \farray, Array.series(16, 1, 1).postln, // 倍音列は等差数列で表現できる
          \aarray, Array.fill(16, 1).normalizeSum.postln).rebuild;
)

// 200[Hz]を基音とする第16倍音までの重ね合せ(第n倍音の振幅は1/n)
(
~out.setn(\base, 200,
          \farray, Array.series(16, 1, 1).postln,
          \aarray, Array.series(16, 1, 1).reciprocal.normalizeSum.postln).rebuild; // ノコギリ波の合成
)
```



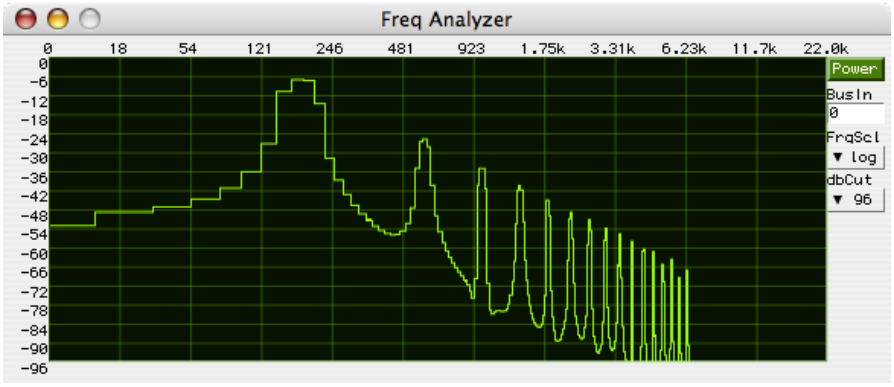


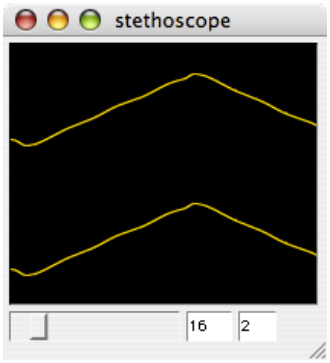

```
// 200[Hz]を基音とする第16奇数次倍音までの重ね合せ(第n倍音の振幅は1/n)
(
~out.setn(\base, 200,
          \farray, Array.series(16, 1, 2).postln,
          \aarray, Array.series(16, 1, 2).reciprocal.normalizeSum.postln).rebuild; // 矩形波の合成
)
```






```
// 200[Hz]を基音とする第16奇数次倍音までの重ね合せ(第n倍音の振幅は1/n**2)
(
~out.setn(\base, 200,
          \farray, Array.series(16, 1, 2).postln,
          \aarray, Array.series(16, 1, 2).squared.reciprocal.normalizeSum.postln).rebuild; // 三角波の合成
)
```





倍音列を表わす等差数列を操作することで音色を変化させる(基音200[Hz])。

```
// 初項を公差の整数倍にする(部分倍音列としての等差数列)
(
~out.setn(\base, 200,
          \farray, Array.series(16, 10, 1).postln,
          \aarray, Array.series(16, 1, 1).reciprocal.normalizeSum.postln).rebuild;
)
```

200[Hz]の基音が聴こえるか?

```
// 初項に比べて公差を小さくする(うなりによるクラスター)
(
~out.setn(\base, 200,
          \farray, Array.series(16, 1, rrand(0.01, 0.1)).postln,
          \aarray, Array.fill(16, 1).reciprocal.normalizeSum.postln).rebuild;
)

// 公差をさらに小さくする
(
~out.setn(\base, 200,
          \farray, Array.series(16, 1, rrand(0.001, 0.01)).postln,
          \aarray, Array.fill(16, 1).reciprocal.normalizeSum.postln).rebuild;
)

// ノコギリ波の基音の変化による音色の違いを聴く(初項を公差の非整数倍にする)
(
~out.setn(\base, 200,
          \farray, Array.series(16, rrand(0.5, 2.0), 1).postln,
          \aarray, Array.series(16, 1, 1).reciprocal.normalizeSum.postln).rebuild;
)

// ノコギリ波の倍音の間隔の変化による音色の違いを聴く(公差を初項の非整数倍にする)
(
~out.setn(\base, 200,
```

```
            \farray, Array.series(16, 1, rrand(0.5, 2.0)).postln,
            \aarray, Array.series(16, 1, 1).reciprocal.normalizeSum.postln).rebuild;
)
```

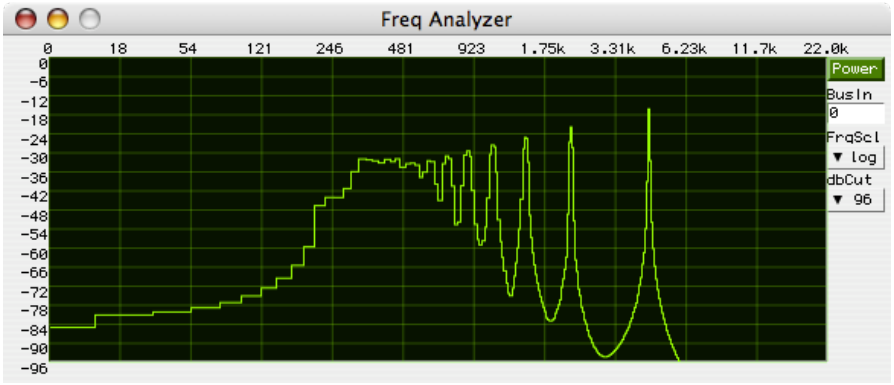
▼エクササイズ：等差数列による音色

等差数列を使ってさまざまな周波数列、振幅列を発生させ、そこからどのような音色が生れるかを探求せよ。

下方倍音列

基音とその整数分の1の高さ(周波数)の線の重ね合せを下方倍音列(下音列)という。下方倍音列の周波数は初項と公差を基音の周波数とする等差数列の逆数で表現できる。現実の物理的な音響現象としては下方倍音列は存在しないが、倍音列(上音列)の逆数としての下方倍音列は、その基音が倍音として含まれる音列であり、倍音列と鏡像関係の音程比を有している。その響きをコード上で確認する。

```
// 4410[Hz]を基音とする第16下方倍音までの重ね合せ(振幅一定)
(
~out.setn(\base, 4410,
            \farray, Array.series(16, 1, 1).reciprocal.postln, // 下方倍音列
            \aarray, Array.fill(16, 1).normalizeSum.postln).rebuild;
)

// 4410[Hz]を基音とする第16倍音までの重ね合せ(第n倍音の振幅は1/n)
(
~out.setn(\base, 4410,
            \farray, Array.series(16, 1, 1).reciprocal.postln, // 下方倍音列
            \aarray, Array.series(16, 1, 1).reciprocal.normalizeSum.postln).rebuild;
)
```



音程とその響き

古典的な和音は、いくつかの音程によって構成される。音程とは音と音との(基本)周波数の比のことである。基本的な音程には、以下のようなものがある。

1 : 1=同音(ユニゾン)
1 : 1.067(15 : 16)=短2度
1 : 1.111(9 : 10)=小全音
1 : 1.125(8 : 9)=長2度(大全音)
1 : 1.2(5 : 6)=短3度
1 : 1.25(4 : 5)=長3度

1 : 1.333(3 : 4)=完全4度
1 : 1.406(32 : 45)=増4度
1 : 1.422(45 : 64)=減5度
1 : 1.5(2 : 3)=完全5度
1 : 1.6(5 : 8)=短6度
1 : 1.667(3 : 5)=長6度
1 : 1.75(4 : 7)=自然7度
1 : 1.778(9 : 16)=短7度
1 : 1.875(8 : 15)=長7度
1 : 2=オクターブ

これらの音程は以下のような特性を持っている。

完全5度、オクターブ：開放的な協和(完全協和音程)
長短3度、長短6度：やわらかい協和(不完全協和音程)
短2度、長7度：鋭い不協和
長2度、短7度：おだやかな不協和
完全4度：協和、あるいは不協和(前後の音程によって変わる)
増4度、減5度(3全音)：あいまい、中立、不安定

なお、平均律的な増4度=減5度の響きは、オクターブを丁度2分する音程、すなわち

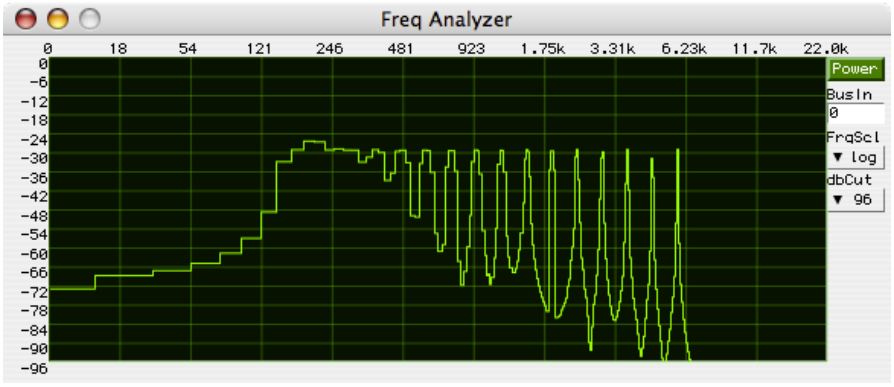
1 : 1.4142(1 : √2)=(平均律的な)増4度/減5度

である。一般に、オクターブを等分割する平均律的なn音音階の音程は、1オクターブのn乗根による比で表わされる。

```
// 1～20音音階における音程比の一覧
20.do({|i| (i+1).post; "-tone scale:".post;  " interval ratio = ".post; (2**(1/(i+1))).postln });

 1-tone scale: interval ratio = 2
 2-tone scale: interval ratio = 1.4142135623731
 3-tone scale: interval ratio = 1.2599210498949
 4-tone scale: interval ratio = 1.1892071150027
 5-tone scale: interval ratio = 1.148698354997
 6-tone scale: interval ratio = 1.1224620483094
 7-tone scale: interval ratio = 1.1040895136738
 8-tone scale: interval ratio = 1.0905077326653
 9-tone scale: interval ratio = 1.0800597388923
10-tone scale: interval ratio = 1.0717734625363
11-tone scale: interval ratio = 1.06504108944
12-tone scale: interval ratio = 1.0594630943593
13-tone scale: interval ratio = 1.0547660764816
14-tone scale: interval ratio = 1.0507566386532
15-tone scale: interval ratio = 1.0472941228206
16-tone scale: interval ratio = 1.0442737824274
17-tone scale: interval ratio = 1.0416160106506
18-tone scale: interval ratio = 1.0392592260318
19-tone scale: interval ratio = 1.0371550444462
20-tone scale: interval ratio = 1.0352649238414
```

倍音列が基音の倍数であることから、それらは一般的に等差数列で表現できたのに対し、音程は周波数間の比であらわされることから、等間隔の音程列は等比数列で表現することができる。

```
// 200[Hz]から長3度音程で重なっていく16本の線(振幅一定)
(
~out.setn(\base, 200,
          \farray, Array.geom(16, 1, 5/4).postln, // 音程列は等比数列で表現できる
          \aarray, Array.fill(16, 1).normalizeSum.postln).rebuild;
)
```




```
// 100[Hz]から短3度音程で重なっていく16本の線(振幅一定)
(
~out.setn(\base, 100,
          \farray, Array.geom(16, 1, 6/5).postln, // 音程列は等比数列で表現できる
          \aarray, Array.fill(16, 1).normalizeSum.postln).rebuild;
)
```

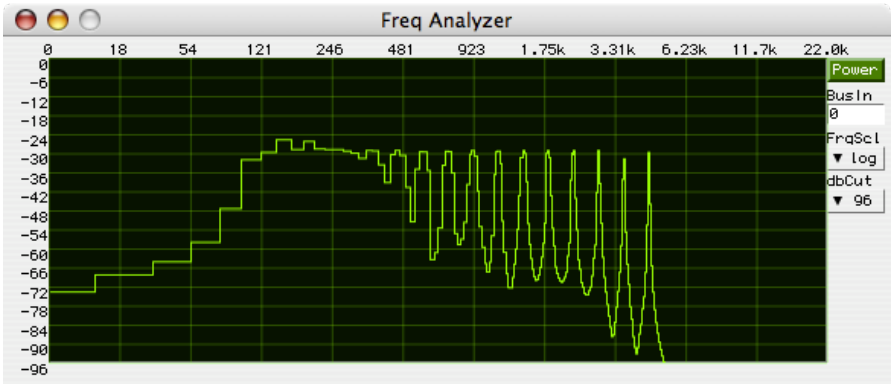
さまざまな音程列の重ね合せによる響きを聴く。

```
// 短2度
~out.setn(\farray, Array.geom(16, 2, 16/15).postln).rebuild;
// 小全音
~out.setn(\farray, Array.geom(16, 2, 10/9).postln).rebuild;
// 長2度
~out.setn(\farray, Array.geom(16, 2, 9/8).postln).rebuild;
// 短3度
~out.setn(\farray, Array.geom(16, 1, 6/5).postln).rebuild;
// 長3度
~out.setn(\farray, Array.geom(16, 1, 5/4).postln).rebuild;
// 完全4度
~out.setn(\farray, Array.geom(16, 0.5, 4/3).postln).rebuild;
// 増4度
~out.setn(\farray, Array.geom(16, 0.5, 45/32).postln).rebuild;
// 平均律増4度
~out.setn(\farray, Array.geom(16, 0.5, 2.sqrt).postln).rebuild;
// 減5度
~out.setn(\farray, Array.geom(16, 0.2, 64/45).postln).rebuild;
// 完全5度
~out.setn(\farray, Array.geom(16, 0.2, 3/2).postln).rebuild;
// 短6度
~out.setn(\farray, Array.geom(16, 0.05, 8/5).postln).rebuild;
// 長6度
~out.setn(\farray, Array.geom(16, 0.05, 5/3).postln).rebuild;
// 自然7度
~out.setn(\farray, Array.geom(16, 0.05, 7/4).postln).rebuild;
// 短7度
~out.setn(\farray, Array.geom(16, 0.02, 16/9).postln).rebuild;
// 長7度
~out.setn(\farray, Array.geom(16, 0.02, 15/8).postln).rebuild;
```

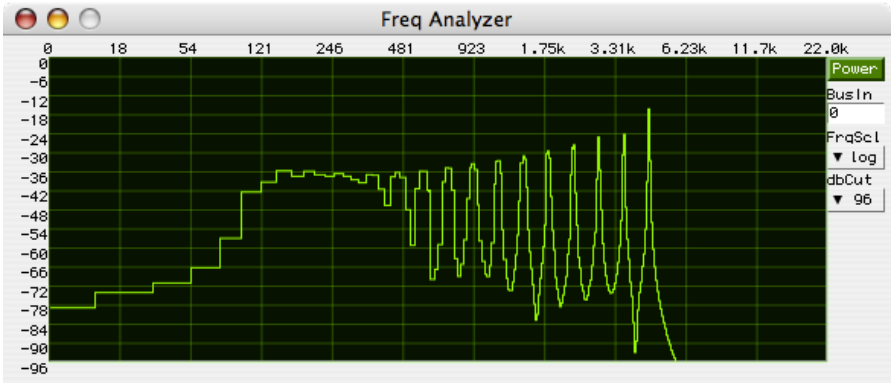
下方音程列

下方倍音列と同様に、基音からある音程で下っていく音程列を考えることできる。

```
// 4410[Hz]を基音とする長3度の下方音程の重ね合せ(振幅一定)
(
~out.setn(\base, 4410,
          \farray, Array.geom(16, 1, 4/5).postln, // 下方音程列
          \aarray, Array.fill(16, 1).normalizeSum.postln).rebuild;
)
```




```
// 4410[Hz]を基音とする長3度の下方音程の重ね合せ(第n音程の振幅は1/n)
(
~out.setn(\base, 4410,
          \farray, Array.geom(16, 1, 4/5).postln, // 下方音程列
          \aarray, Array.series(16, 1, 1).reciprocal.normalizeSum.postln).rebuild;
)
```



▼エクササイズ：等比数列による音色

等比数列を使ってさまざまな周波数列、振幅列を発生させ、そこからどのような音色が生れるかを探求せよ。

倍音列と音程

倍音列をオクターブ移動して正規化(1オクターブ以内の音程に変換)すると、以下のような音程が得られる。最初の4つの倍音から長3和音が得られる。

1：1=同音
1：2→1：1=同音
1：3→2：3=1：1.5=完全5度
1：4→1：1=同音
1：5→4：5=1：1.25=長3度
1：6→2：3=1：1.5=完全5度
1：7→4：7=1：1.75=自然7度

1 : 8→1 : 1=同音
1 : 9→1 : 1.125=長2度
1 : 10→4 : 5=1 : 1.25=長3度
1 : 11→8 : 11=1 : 1.375=完全4度と増4度の間
1 : 12→2 : 3=1 : 1.5=完全5度
1 : 13→8 : 13=1 : 1.625=短6度と長6度の間
1 : 14→4 : 7=1 : 1.75=自然7度
1 : 15→8 : 16=1 : 1.875=長7度
1 : 16→1 : 1=同音

```
// 倍音列から導かれる和音(長3和音)
~out = { Mix.ar( SinOsc.ar([1, 5/4, 3/2]*441, 0, 0.2)).dup };

// 自然7度を加える
~out = { Mix.ar( SinOsc.ar([1, 5/4, 3/2, 7/4]*441, 0, 0.2)).dup };
```

これらを音程の狭い順にならべかえると以下のようになる。倍音列から導かれる音階である。

1 : 1=同音
1 : 1.125=長2度
1 : 1.25=長3度
1 : 1.375=完全4度と増4度の間
1 : 1.5=完全5度
1 : 1.625=短6度と長6度の間
1 : 1.75=自然7度
1 : 1.875=長7度

```
// 倍音列から導かれる音階(倍音列音階)
(
~out = Pbind(
        \instrument, \line,
        \amp, 0.5,
        \freq, Pseq([1, 9/8, 5/4, 11/8, 3/2, 13/8, 7/4, 2] * 441, inf),
        \sustain, 0.5,
        \dur, 0.5,
        \pan, 0.0
    );
)
```

同様に下方倍音列をオクターブ移動して正規化(1オクターブ以内の音程に変換)すると、以下のような音程が得られる。最初の4つの倍音から短3和音が得られる。

1 : 1=同音
2 : 1→1 : 1=同音
3 : 1→3 : 4=1 : 1.333=完全4度
4 : 1→1 : 1=同音
5 : 1→5 : 8=1 : 1.6=短6度
6 : 1→3 : 4=1 : 1.333=完全4度
7 : 1→7 : 8=1 : 1.143=長2度と短3度の間
8 : 1→1 : 1=同音
9 : 1→9 : 16=1 : 1.778=短7度
10 : 1→5 : 8=1 : 1.6=短6度
11 : 1→11 : 16=1 : 1.455=完全4度と増4度の間
12 : 1→3 : 4=1 : 1.333=完全4度
13 : 1→13 : 16=1 : 1.231=短3度と長3度の間
14 : 1→7 : 8=1 : 1.143=長2度と短3度の間
15 : 1→15 : 16=1 : 1.067=短2度
16 : 1→1 : 1=同音

```
// 下方倍音列から導かれる和音(短3和音)
~out = { Mix.ar( SinOsc.ar([2/3, 4/5, 1.0]*441, 0, 0.2)).dup };
```

これらを音程の狭い順にならべかえると以下のようになる。下方倍音列から導かれる音階である。

1 : 1=同音
1 : 1.067=短2度
1 : 1.143=長2度と短3度の間
1 : 1.231=短3度と長3度の間
1 : 1.333=完全4度
1 : 1.455=完全4度と増4度の間
1 : 1.6=短6度
1 : 1.778=短7度

```
// 下方倍音列から導かれる音階(下方倍音列音階)
(
~out = Pbind(
        \instrument, \line,
        \amp, 0.5,
        \freq, Pseq([1, 8/7, 16/13, 4/3, 16/11, 8/5, 16/9, 2] * 441, inf),
        \sustain, 0.5,
        \dur, 0.5,
        \pan, 0.0
    );
)
```

和音のデザイン

変奏の場合と同様に、数式や数列を使って、特定の志向性を持たないランダム(確率的)な状態に構造を与えていくことで「音響和音のコンポジション」へと発展させていく。

```
// 16本の線を重ねて鳴らす関数を再定義する
(
~out = {arg base = 441.0, fscale = 1, foffset = 0, pan = 0, level = 0.8;
        var farray, aarray;
        farray = Control.names(\farray).kr(Array.fill(16, 0)) * base;
        aarray = Control.names(\aarray).kr(Array.fill(16, 1).normalizeSum);
        Pan2.ar(Klang.ar(`[ farray, aarray, nil ], fscale, foffset), pan, level) };
)
```

まず最初に、すべての線の振幅は一定とする。

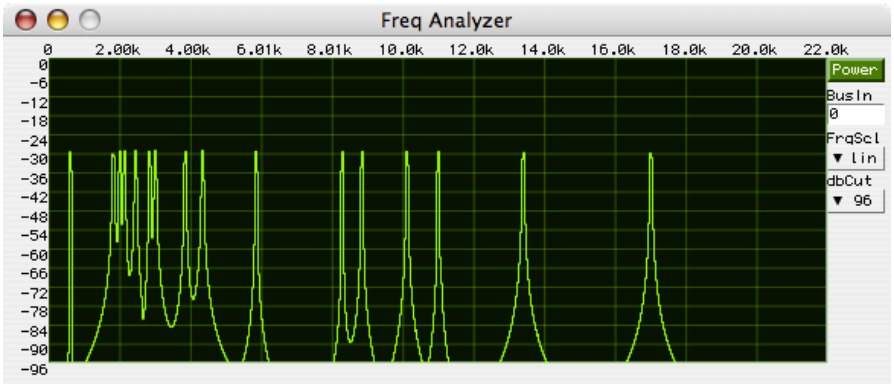
```
~out.setn(\aarray, Array.fill(16, 1).reciprocal.normalizeSum.postln).rebuild;
```

基音は1[Hz](周波数を直接設定する)。

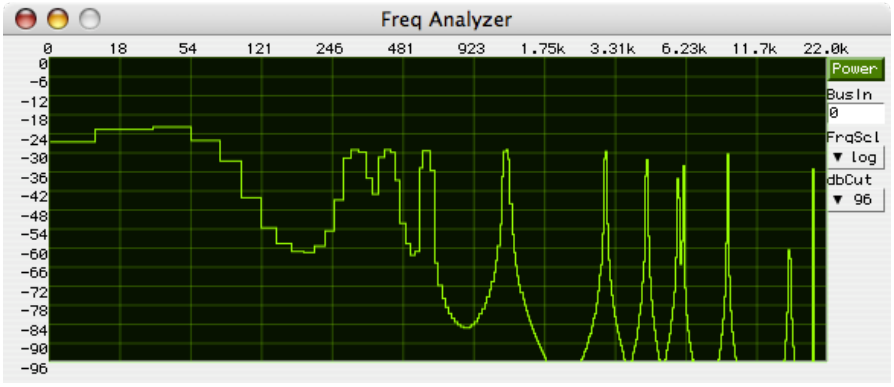
```
~out.setn(\base, 1).rebuild;
```

広い帯域のランダムな周波数分布による音響和音を生成する。

```
// 20[Hz]から20000[Hz]の間のランダムな周波数による16の線(線形分布：通常の一様乱数)
~out.setn(\farray, Array.linrand(16, 20.0, 20000.0).sort.postln).rebuild;
```




```
// 20[Hz]から20000[Hz]の間のランダムな周波数による16の線(指数分布：対数目盛上での一様乱数)
~out.setn(\farray, Array.exprand(16, 20.0, 20000.0).sort.postln).rebuild;
```



線形分布の方が全体的に高音域に音が寄っている。

次に、振幅も同様にランダム(一様分布)にする。

```
~out.setn(\aarray, Array.linrand(16, 0.0, 1.0).normalizeSum.postln).rebuild;
```

振幅分布が変化すると音色も大きく変化する。

周波数分布の帯域を狭くする。

まず、再び振幅分布を一定にする。

```
~out.setn(\aarray, Array.fill(16, 1).reciprocal.normalizeSum.postln).rebuild;
```

```
// 1000[Hz]から1200[Hz]の間のランダムな周波数による16の線(線形分布)
~out.setn(\farray, Array.linrand(16, 1000.0, 1200.0).sort.postln).rebuild;

// 1000[Hz]から1200[Hz]の間のランダムな周波数による16の線(指数分布)
~out.setn(\farray, Array.exprand(16, 1000.0, 1200.0).sort.postln).rebuild;
```

音が密集している場合には、線形分布と対数分布の間にあまり音色の差はない。

ゆらぎやズレとしてのランダムネスの役割を聴く。

```
// 初項(基音)が500[Hz]、公差が400[Hz]の等差数列(均等分布)による音響和音
~out.setn(\farray, (Array.series(16, 500.0, 400.0)).sort.postln).rebuild;

// 周波数列に振幅が±5[Hz]の乱数を重ねる
~out.setn(\farray, (Array.series(16, 500.0, 400.0) + Array.rand2(16, 5.0)).postln).rebuild;
```

周波数の小さなズレによって、コーラスのような効果が出る。

```
// 周波数列に振幅が±50[Hz]の乱数を重ねる
~out.setn(\farray, (Array.series(16, 500.0, 400.0) + Array.rand2(16, 50.0)).postln).rebuild;
```

周波数が大きくズレると響きそのものが変化する。

次に等差数列と線形分布による乱数を合成(連結)する。

初項=500[Hz]、公差=400[Hz]の等差数列による16音の和音から出発する。

```
~out.setn(\farray, (Array.series(16, 400.0, 400.0)).sort.postln).rebuild;
```

16音のうち4音をランダムな周波数に置き換える。

```
// ランダムな周波数成分を全体に分布させる
~out.setn(\farray, (Array.series(12, 400.0, 400.0) ++ Array.linrand(2, 400.0,
4800.0)).sort.postln).rebuild;

// 高域の4音の周波数成分をランダムに分布させる
~out.setn(\farray, (Array.series(12, 400.0, 400.0) ++ Array.linrand(2, 4800.0,
6400.0)).sort.postln).rebuild;
```

16音のうち8音をランダムな周波数に置き換える。

```
// ランダムな周波数成分を全体に分布させる
```

```
~out.setn(\farray, (Array.series(8, 400.0, 400.0) ++ Array.linrand(8, 500.0,
3200.0)).sort.postln).rebuild;
```

音色が金属的になる(ランダムな周波数成分が非整数次の倍音として機能する)。

```
// 高域の8音の周波数成分をランダムに分布させる
~out.setn(\farray, (Array.series(8, 400.0, 400.0) ++ Array.linrand(8, 3200.0,
6400.0)).sort.postln).rebuild;
```

低域と高域で音色が分離して聴こえる。

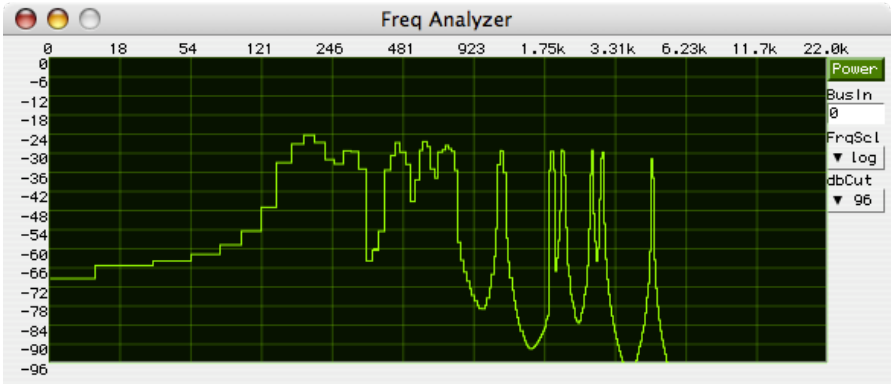
▼エクササイズ：等比数列による和音

同様のプロセスを等比数列(等音程列)による和音で行ってみよ。

```
// 基音が200[Hz]、公比が5/4(長3度)の等比数列による音響和音
(
~out.setn(\base, 200,
          \farray, Array.geom(16, 1, 5/4).postln,
          \aarray, Array.fill(16, 1).normalizeSum.postln).rebuild;
)

// 周波数列に振幅が±10%の乱数を重ねる
~out.setn(\farray, (Array.geom(16, 1, 5/4) + Array.rand2(16, 0.1)).postln).rebuild;

// 周波数列に振幅が±50%の乱数を重ねる
~out.setn(\farray, (Array.geom(16, 1, 5/4) + Array.rand2(16, 0.5)).postln).rebuild;
```

等比数列と指数分布による乱数を合成(連結)する。
16音のうち8音をランダムな周波数に置き換える。

```
// ランダムな周波数成分を全体に分布させる
(
var series = Array.geom(8, 1, (5/4)**2).postln;
~out.setn(\farray, (series ++ Array.exprand(8, series.first, series.last)).sort.postln).rebuild;
)
```



面の思考–倍音列と音程列の統合

倍音列と音程列を、線の集合(重ね合せ)として統合的に取り扱う。旋律という音響の水平構造に対して、基音をベースに音を重ねていく和音は音響の垂直構造と呼ばれているが、和音を構成するひとつひとつの音が有している倍音列を意識することで、和音を単に垂直的な(線的な)音の連なりとしてではなく、倍音列という音色による奥行も含んだひとつの面として捉えることができる。このように、和音と音色を音面(マトリクス)の拡がりとして統合的にデザインすることを、音響和音における「面の思考」と呼ぶ。ここで音面としてのマトリクスは単なるメタファーではなく、和音を構成する音程列と音色を構成する倍音列の外積をとることで得られる実体(パラメータ・マトリクス)としての面である。

```
// 64本(8×8のマトリクス)の線を重ねて鳴らす関数を定義する
(
~out = {arg base = 441, level = 0.8;
        var farray, aarray, parray;
        farray = Control.names(\farray).kr(Array.fill(64, 0));
        aarray = Control.names(\aarray).kr(Array.fill(64, 0));
        parray = Control.names(\parray).kr(Array.fill(64, 0));
        Mix.new(
            Pan2.ar(
                SinOsc.ar(
                    farray * base, 0.0, aarray, 0.0),
                parray, level))};
)
```

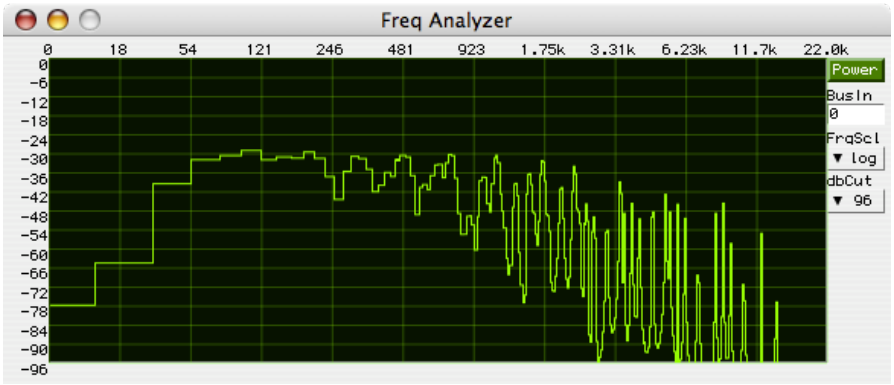
倍音列としての等差数列と音程列としての等比数列の外積から周波数マトリクスを生成する。

```
(
var farray1 = Array.series(8, 1, 1).round(0.1).postln; // 等差数列(倍音列)
var farray2 = Array.geom(8, 1, 1.5).round(0.1).postln; // 等比数列(音程列)
(farray1 * .t farray2).do({ |row| row.round(0.1).postln }); // 倍音列と音程列の周波数マトリックス
)

[ 1, 2, 3, 4, 5, 6, 7, 8 ]

[ 1, 1.5, 2.3, 3.4, 5.1, 7.6, 11.4, 17.1 ]

[ 1, 1.5, 2.3, 3.4, 5.1, 7.6, 11.4, 17.1 ]
[ 2, 3, 4.6, 6.8, 10.2, 15.2, 22.8, 34.2 ]
[ 3, 4.5, 6.9, 10.2, 15.3, 22.8, 34.2, 51.3 ]
[ 4, 6, 9.2, 13.6, 20.4, 30.4, 45.6, 68.4 ]
[ 5, 7.5, 11.5, 17, 25.5, 38, 57, 85.5 ]
[ 6, 9, 13.8, 20.4, 30.6, 45.6, 68.4, 102.6 ]
[ 7, 10.5, 16.1, 23.8, 35.7, 53.2, 79.8, 119.7 ]
[ 8, 12, 18.4, 27.2, 40.8, 60.8, 91.2, 136.8 ]
```

倍音列に対しては振幅反比例、音程列に対しては振幅一定の振幅マトリクスを生成する。

```
(
var farray1 = Array.series(8, 1, 1).reciprocal.round(0.01).postln; // 振幅反比例
var farray2 = Array.fill(8, 1).round(0.01).postln; // 振幅一定
(farray1 * .t farray2).do({ |row| row.round(0.01).postln }); // 振幅反比例と一定の振幅マトリックス
)

[ 1, 0.5, 0.33, 0.25, 0.2, 0.17, 0.14, 0.13 ]

[ 1, 1, 1, 1, 1, 1, 1, 1 ]

[ 1, 1, 1, 1, 1, 1, 1, 1 ]
[ 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 ]
[ 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33 ]
[ 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25 ]
[ 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2 ]
[ 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17 ]
[ 0.14, 0.14, 0.14, 0.14, 0.14, 0.14, 0.14, 0.14 ]
[ 0.13, 0.13, 0.13, 0.13, 0.13, 0.13, 0.13, 0.13 ]
```

```
// 振幅反比例の倍音列と振幅一定の完全5度の音程列(基音100[Hz]、中央パン)
(
~out.setn(\base, 100,
          \farray, (Array.series(8, 1, 1) * .x Array.geom(8, 1, 1.5)).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```

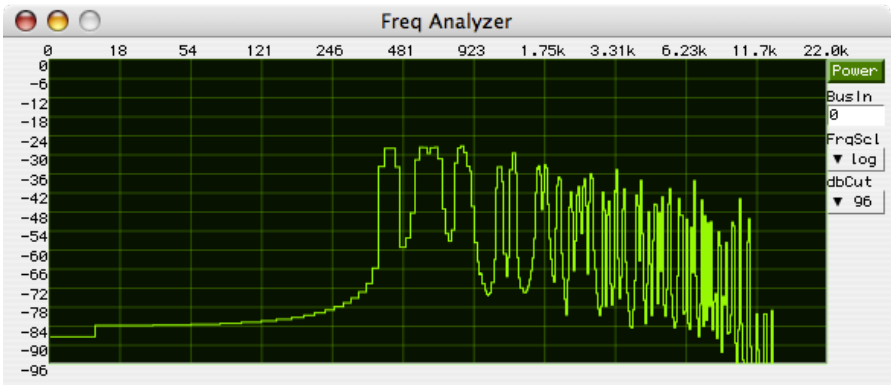


▼エクササイズ：音響和音のリスニング

さまざまな音程や倍音列の組み合わせから周波数マトリックスと振幅マトリクスを生成し、どのような響きがするかを確かめよ。

3度構成による音響和音

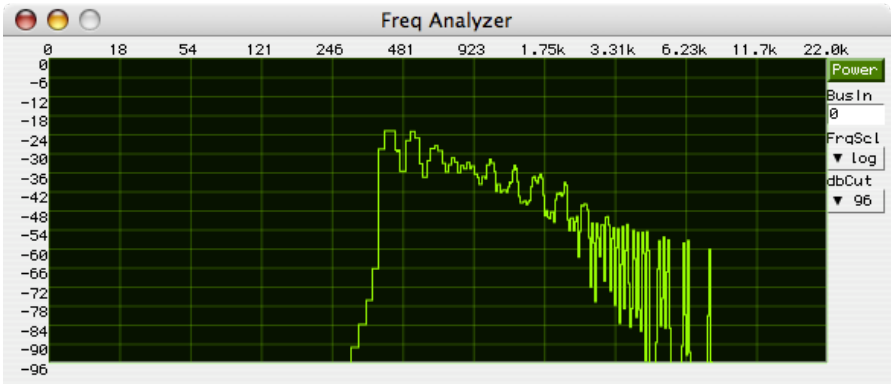
```
// 古典的長7和音(基音441[Hz]、中央パン)
(
~out.setn(\base, 441,
          \farray, (Array.series(16, 1, 1) * .x [1, 4/3, 3/2, 15/8]).postln,
          \aarray, (Array.series(16, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```



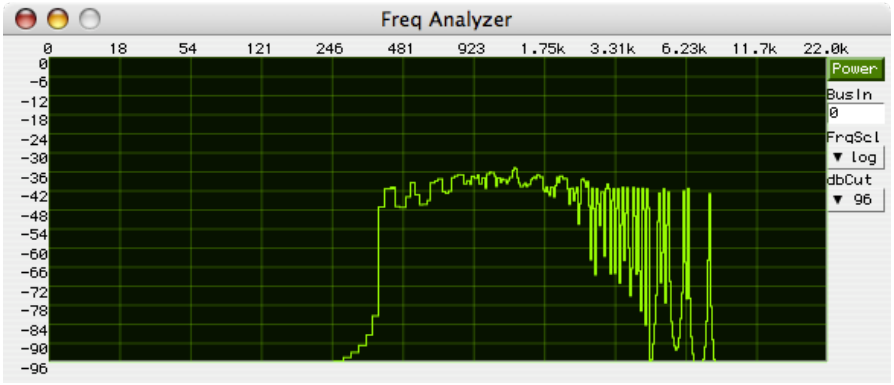

```
// 古典的短7和音(基音441[Hz]、中央パン)
```

```
(
~out.setn(\base, 441,
          \farray, (Array.series(16, 1, 1) * .x [1, 5/4, 3/2, 9/5]).postln,
          \aarray, (Array.series(16, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)


// ランダムな3度構成の4音(7の)和音(基音441[Hz]、ランダムパン)
(
var intervals = [1, 1, 1, 1];
intervals.put(0, 1);
3.do({|i| intervals.put(i+1, intervals.at(i) * [6/5, 5/4].choose)});
intervals.postln;
~out.setn(\base, 441,
          \farray, (Array.series(16, 1, 1) * .x intervals).postln,
          \aarray, (Array.series(16, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray, Array.rand2(64, 1.0).postln).rebuild;
)


// ランダムな3度構成の8音(15の)和音(基音441[Hz]、ランダムパン)
(
var intervals = [1, 1, 1, 1, 1, 1, 1, 1];
intervals.put(0, 1);
7.do({|i| intervals.put(i+1, intervals.at(i) * [6/5, 5/4].choose)});
intervals.postln;
~out.setn(\base, 441,
          \farray, (Array.series(8, 1, 1) * .x intervals).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).normalizeSum.postln,
          \parray, Array.rand2(64, 1.0).postln).rebuild;
)

// 振幅マトリックスを逆順(変奏の一種、高次倍音の強調)にする(基音441[Hz]、中央パン)
(
var intervals = [1, 1, 1, 1, 1, 1, 1, 1];
intervals.put(0, 1);
7.do({|i| intervals.put(i+1, intervals.at(i) * [6/5, 5/4].choose)});
intervals.postln;
~out.setn(\base, 441,
          \farray, (Array.series(8, 1, 1) * .x intervals).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).reverse.normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)


2つの音程列からマトリックスを生成すると、どのような音が聴こえるか?


// ランダムな2つの3度構成音列の組み合わせ(基音441[Hz]、中央パン)
(
var intervals1 = [1, 1, 1, 1, 1, 1, 1, 1], intervals2 = [1, 1, 1, 1, 1, 1, 1, 1];
intervals1.put(0, 1);
intervals2.put(0, 1);
7.do({|i| intervals1.put(i+1, intervals1.at(i) * [6/5, 5/4].choose)});
7.do({|i| intervals2.put(i+1, intervals2.at(i) * [6/5, 5/4].choose)});
intervals1.postln;
intervals2.postln;
~out.setn(\base, 441,
          \farray, (intervals1 * .x intervals2).postln,
          \aarray, (Array.series(8, 1, 1) * .x Array.series(8, 1, 1)).reciprocal.normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)


// 振幅とパンをランダムにする
(
var intervals1 = [1, 1, 1, 1, 1, 1, 1, 1], intervals2 = [1, 1, 1, 1, 1, 1, 1, 1];
intervals1.put(0, 1);
intervals2.put(0, 1);
```

```
7.do({|i| intervals1.put(i+1, intervals1.at(i) * [6/5, 5/4].choose)});
7.do({|i| intervals2.put(i+1, intervals2.at(i) * [6/5, 5/4].choose)});
intervals1.postln;
intervals2.postln;
~out.setn(\base, 441,
          \farray, (intervals1 * .x intervals2).postln,
          \aarray, Array.rand(64, 0.0, 1.0).normalizeSum.postln,
          \parray, Array.rand2(64, 1.0).postln).rebuild;
)

// 長3度音程列(増3和音)と短3度音程列(減3和音)の組み合わせ(基音441[Hz]、中央パン)
(
~out.setn(\base, 441,
          \farray, (Array.geom(8, 1, 5/4) * .x Array.geom(8, 1, 6/5)).postln,
          \aarray, (Array.series(8, 1, 1) * .x Array.series(8, 1, 1)).reciprocal.normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```



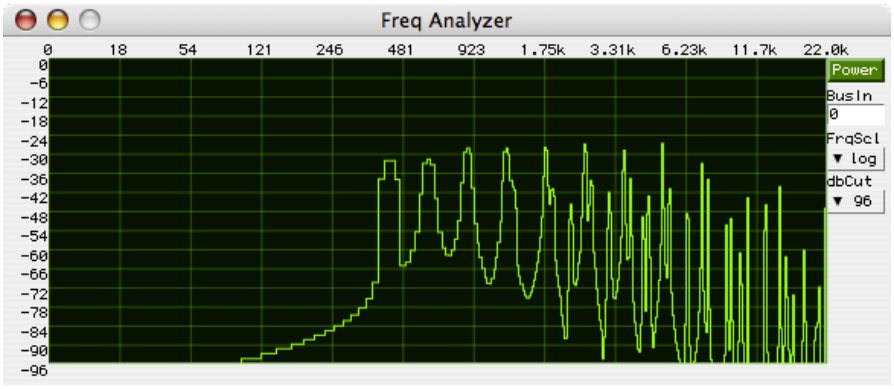

```
// 振幅を一定にする
(
~out.setn(\base, 441,
          \farray, (Array.geom(8, 1, 5/4) * .x Array.geom(8, 1, 6/5)).postln,
          \aarray, Array.fill(64, 1).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```



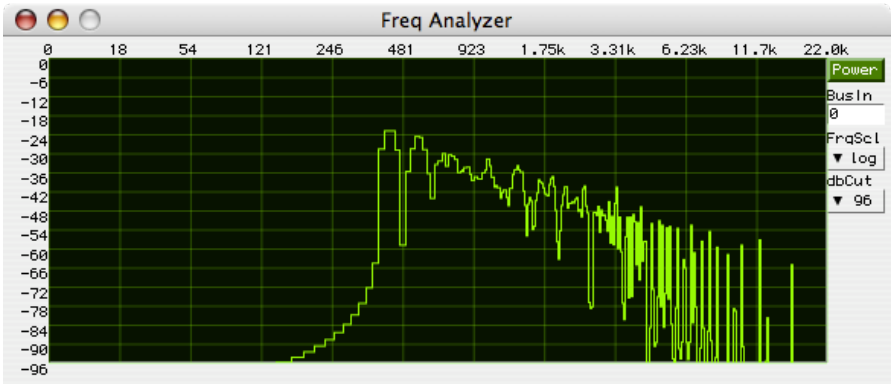
4度構成による音響和音

```
// 完全4度音程の4音和音(基音441[Hz]、中央パン)
(
```

```
~out.setn(\base, 441,
          \farray, (Array.series(16, 1, 1) * .x Array.geom(4, 1, 4/3)).postln,
          \aarray, (Array.series(16, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)

// 完全4度音程の8音和音(基音441[Hz]、中央パン)
(
~out.setn(\base, 441,
          \farray, (Array.series(8, 1, 1) * .x Array.geom(8, 1, 4/3)).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)


// 増4度音程の8音和音(基音441[Hz]、中央パン)
(
~out.setn(\base, 441,
          \farray, (Array.series(8, 1, 1) * .x Array.geom(8, 1, 2.sqrt)).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```

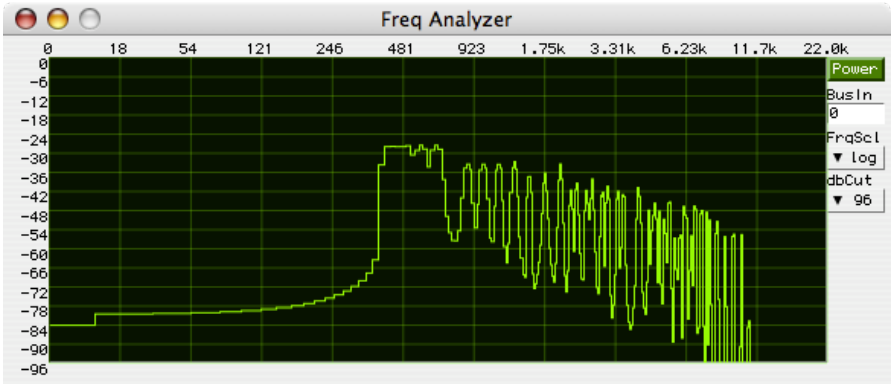



```
// 長3度音程列と完全4度音程列の組み合わせ(基音441[Hz]、中央パン)
(
~out.setn(\base, 441,
          \farray, (Array.geom(8, 1, 5/4) * .x Array.geom(8, 1, 4/3)).postln,
          \aarray, (Array.series(8, 1, 1) * .x Array.series(8, 1, 1)).reciprocal.normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```

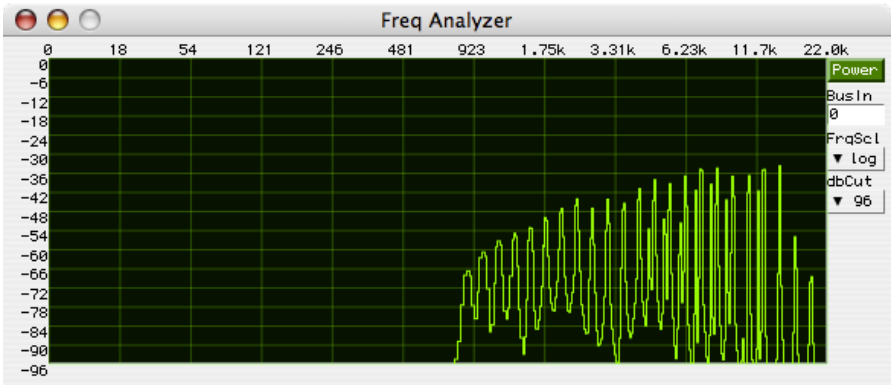


ペンタトニック(5音音階)による音響和音

```
// ペンタトニック音程の4音和音(基音441[Hz]、中央パン)
(
~out.setn(\base, 441,
          \farray, (Array.series(16, 1, 1) * .x Array.geom(4, 1, 2**(1/5))).postln,
          \aarray, (Array.series(16, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```



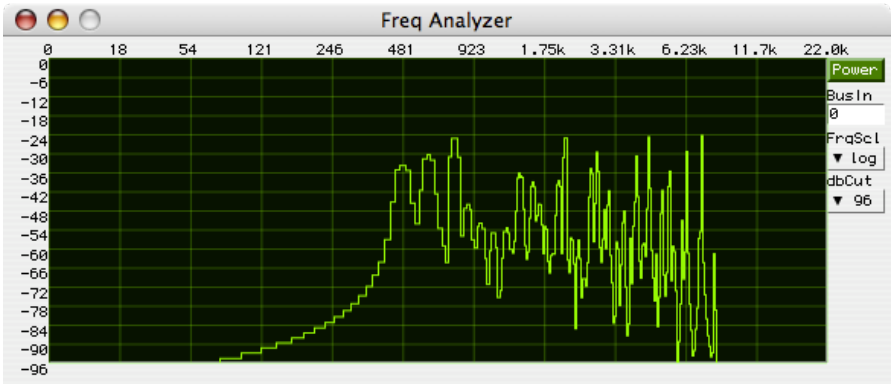

```
// ペンタトニック音程の8音和音(基音441[Hz]、中央パン)
(
~out.setn(\base, 441,
          \farray, (Array.series(8, 1, 1) * .x Array.geom(8, 1, 2**(1/5))).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)

// 高域にいくにつれて振幅を大きくする(基音882[Hz]、中央パン)
(
~out.setn(\base, 882,
          \farray, (Array.series(8, 1, 1) * .x Array.geom(8, 1, 2**(1/5))).postln,
          \aarray, (Array.series(8, 1, 1) * .x Array.series(8, 1, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```



ホールトーン(6音音階)による和音

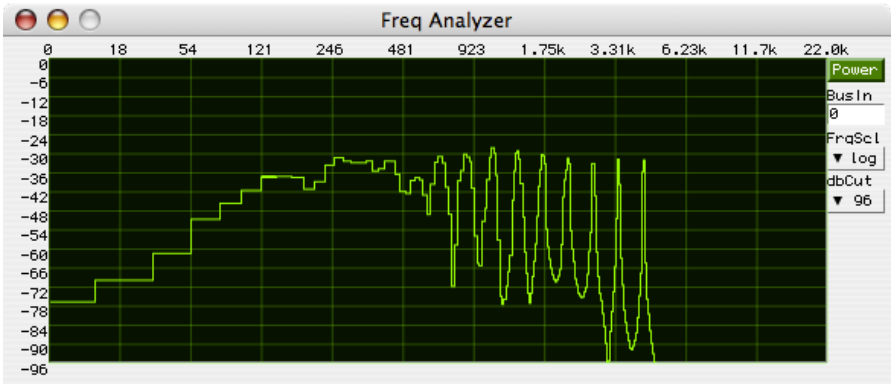
```
// ホールトーン音程の8音和音(基音441[Hz]、中央パン)
(
```

```
~out.setn(\base, 441,
          \farray, (Array.series(8, 1, 1) * .x Array.geom(8, 1, 2**(1/6))).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)

// ランダム振幅、ランダムパン
(
~out.setn(\base, 441,
          \farray, (Array.series(8, 1, 1) * .x Array.geom(8, 1, 2**(1/6))).postln,
          \aarray, Array.exprand(64, 0.001, 1.0).normalizeSum.postln,
          \parray, Array.rand2(64, 1.0).postln).rebuild;
)
```



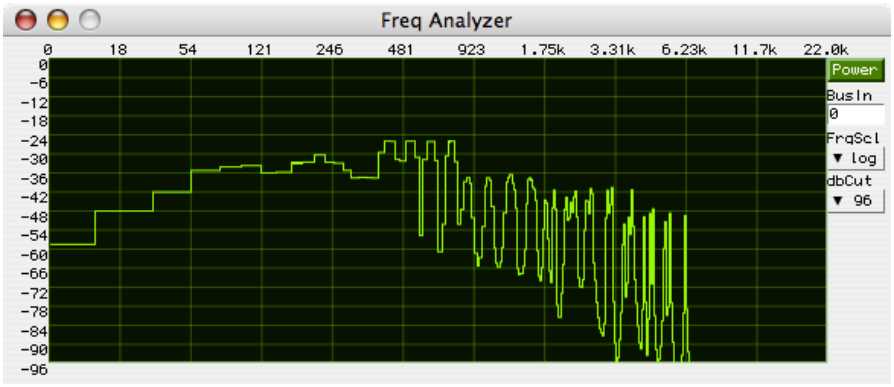
下方倍音列を用いた音響和音

```
// 上行する音程列と下方倍音列の組み合わせ(基音882[Hz]、中央パン)
(
~out.setn(\base, 882,
          \farray, (Array.series(8, 1, 1).reciprocal * .x Array.geom(8, 1, [6/5, 5/4,
3/2].choose)).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```

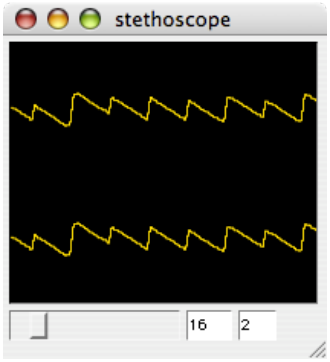


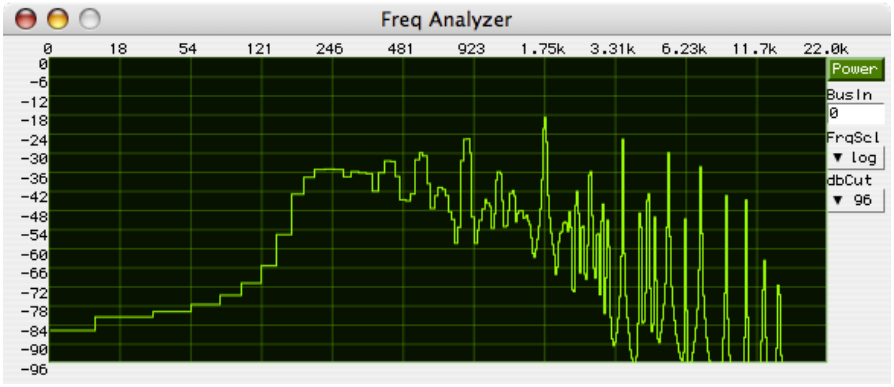

```
// 音程列と対称(上行+下行)倍音列の組み合わせ(基音441[Hz]、中央パン)
(
~out.setn(\base, 441,
          \farray, ((Array.series(8, 1, 1) ++ Array.series(8, 1, 1).reciprocal)
```

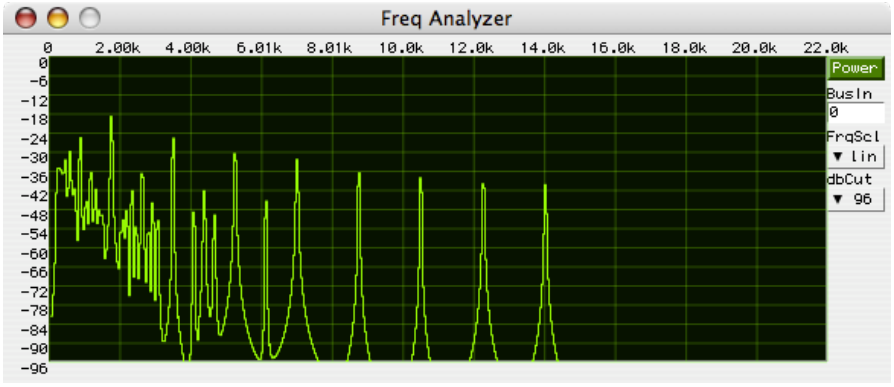
```
            * .x Array.geom(4, 1, [1.2, 1.25, 1.5].choose)).postln,
          \aarray, ((Array.series(8, 1, 1).reciprocal ++ Array.series(8, 1, 1).reciprocal)
            * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```




```
// 倍音列と下方倍音列の組み合わせ(基音1750[Hz]、中央パン)
(
~out.setn(\base, 1750,
          \farray, (Array.series(8, 1, 1) * .x Array.series(8, 1, 1).reciprocal).postln,
          \aarray, (Array.series(8, 1, 1) * .x Array.series(8, 1, 1)).reciprocal.normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```



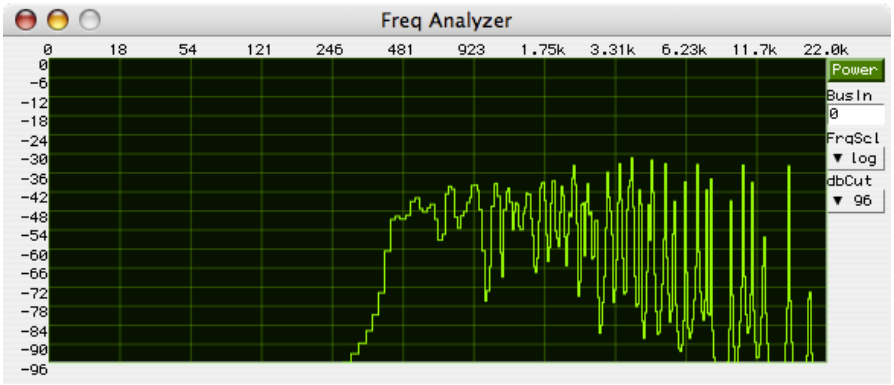




基音から上下に倍音列が伸びていく。

ぶらさがる(積み下がる)和音

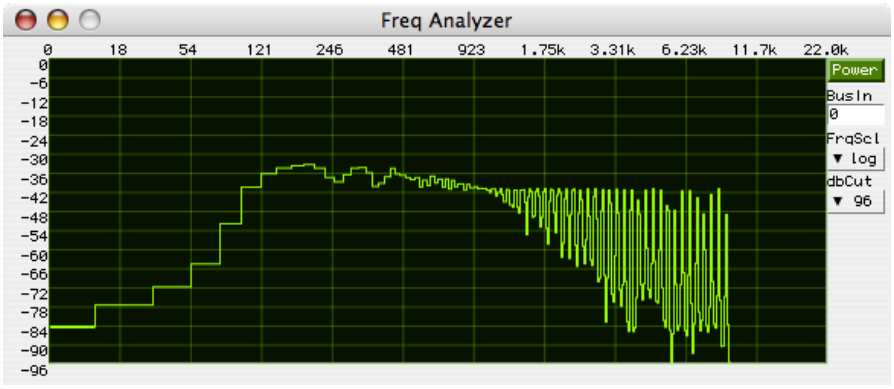
```
// 下方倍音列と下行音程列(3度/5度)の組み合わせ(基音22050[Hz]、中央パン)
(
var note = [1, 1, 1, 1, 1, 1, 1, 1];
note.put(0, 1);
7.do({|i| note.put(i+1, note.at(i) * [5/6, 4/5, 2/3].choose)});
note.postln;
~out.setn(\base, 22050,
          \farray, (Array.series(8, 1, 1).reciprocal * .x note).postln,
          \aarray, (Array.series(8, 1, 1).reciprocal * .x Array.fill(8, 1)).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```



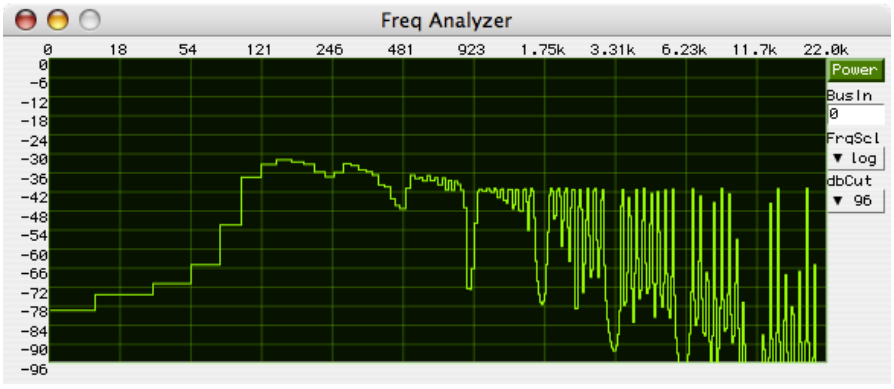

```
// より高域を強調する
(
var note = [1, 1, 1, 1, 1, 1, 1, 1];
note.put(0, 1);
7.do({|i| note.put(i+1, note.at(i) * [5/6, 4/5, 2/3].choose)});
note.postln;
~out.setn(\base, 22050,
          \farray, (Array.series(8, 1, 1).reciprocal * .x note).postln,
          \aarray, (Array.series(8, 1, 1) * .x Array.series(8, 1, 1)).reciprocal.normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```

クラスター（2度/7度構成の和音）

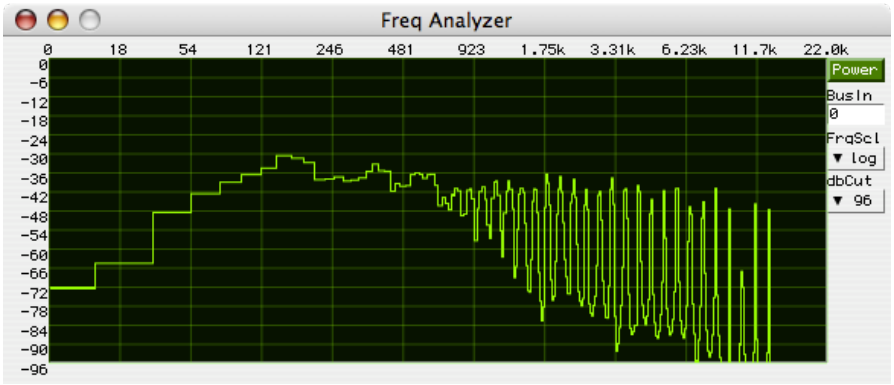
```
// 64個の短2度音程列の組み合わせ（基音150[Hz]、中央パン）
(
~out.setn(\base, 150,
          \farray, Array.geom(64, 1, 16/15).postln,
          \aarray, Array.fill(64, 1.0).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```




```
// 64個の長2度音程列の組み合わせ（基音100[Hz]、中央パン）
(
~out.setn(\base, 100,
          \farray, Array.geom(64, 1, 9/8).postln,
          \aarray, Array.fill(64, 1.0).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)

// ランダム振幅、ランダムパン
(
~out.setn(\base, 100,
          \farray, Array.geom(64, 1, 9/8).postln,
          \aarray, Array.rand(64, 0.0, 1.0).normalizeSum.postln,
          \parray, Array.rand2(64, 1.0).postln).rebuild;
)

// 短2度音程列と長7度音程列の組み合わせ（基音150[Hz]、中央パン）
(
~out.setn(\base, 150,
          \farray, (Array.geom(8, 1, 16/15) * .x Array.geom(8, 1, 15/8)).postln,
          \aarray, Array.fill(64, 1.0).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```



```
// 長2度音程列と短7度音程列の組み合わせ(基音100[Hz]、中央パン)
(
~out.setn(\base, 100,
          \farray, (Array.geom(8, 1, 9/8) * .x Array.geom(8, 1, 16/9)).postln,
          \aarray, Array.fill(64, 1.0).normalizeSum.postln,
          \parray, Array.fill(64, 0.0).postln).rebuild;
)
```




```
// ランダムな2度あるいは7度構成音列によるクラスター(基音150[Hz]、ランダムパン)
(
var intervals1 = [1, 1, 1, 1, 1, 1, 1, 1], intervals2 = [1, 1, 1, 1, 1, 1, 1, 1];
intervals1.put(0, 1);
intervals2.put(0, 1);
7.do({|i| intervals1.put(i+1, intervals1.at(i) * [16/15, 9/8, 16/9, 15/8].choose)});
7.do({|i| intervals2.put(i+1, intervals2.at(i) * [16/15, 9/8, 16/9, 15/8].choose)});
intervals1.postln;
intervals2.postln;
~out.setn(\base, 150,
          \farray, (intervals1 * .x intervals2).postln,
          \aarray, Array.rand(64, 0.0, 1.0).normalizeSum.postln,
          \parray, Array.rand2(64, 1.0).postln).rebuild;
)
```

ポリ・コード

複数のマトリクスの結合(同時的な組み合わせ)による音響和音をポリ・コードという。

```
// 64本(8×8のマトリクス)の線を2組に分けて鳴らす関数を定義する
(
~out = {arg dur = 10, base1 = 441, base2 = 441, level = 0.4;
        var farray1, farray2, aarray1, aarray2, parray1, parray2;
        farray1 = Control.names(\farray1).kr(Array.fill(32, 0));
        farray2 = Control.names(\farray2).kr(Array.fill(32, 0));
        aarray1 = Control.names(\aarray1).kr(Array.fill(32, 0));
        aarray2 = Control.names(\aarray2).kr(Array.fill(32, 0));
        parray1 = Control.names(\parray1).kr(Array.fill(32, 0));
        parray2 = Control.names(\parray2).kr(Array.fill(32, 0));
        Mix.new(
            Pan2.ar(
                SinOsc.ar(
                    (farray1 * base1) ++ (farray2 * base2), 0.0, aarray1 ++ aarray2, 0.0),
                parray1 ++ parray2, level))};
)

// 低域(基音100[Hz])の5度和音と高域(基音1100[Hz])の3度和音の重ね合わせ
```

```
(
~out.setn(\base1, 100,
          \farray1, (Array.series(8, 1, 1) * .x Array.geom(4, 1, 3/2)).postln,
          \aarray1, (Array.series(8, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray1, Array.fill(32, 0.0).postln,
          \base2, 1100,
          \farray2, (Array.series(8, 1, 1) * .x [1, 4/3, 3/2, 15/8]).postln,
          \aarray2, (Array.series(8, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray2, Array.fill(32, 0.0).postln).rebuild;
)


// 短2度音程の2つの基音(441[Hz]と441*16/15[Hz]上の長3和音と短3和音の重ね合わせ
(
~out.setn(\base1, 441,
          \farray1, (Array.series(8, 1, 1) * .x [1, 4/3, 3/2, 15/8]).postln,
          \aarray1, (Array.series(8, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray1, Array.rand2(32, 1.0).postln,
          \base2, 441*16/15,
          \farray2, (Array.series(8, 1, 1) * .x [1, 5/4, 3/2, 9/5]).postln,
          \aarray2, (Array.series(8, 1, 1).reciprocal * .x Array.fill(4, 1)).normalizeSum.postln,
          \parray2, Array.rand2(32, 1.0).postln).rebuild;
)
```

音程列だけでなく、複数の倍音列を組み合せることもできる(ポリ倍音)

```
// 基音441[Hz]の倍音列と基音400[Hz])の1.1倍倍音列の重ね合わせ
(
~out.setn(\base1, 441,
          \farray1, Array.series(32, 1, 1).postln,
          \aarray1, Array.series(32, 1, 1).reciprocal.normalizeSum.postln,
          \parray1, Array.fill(32, 0.0).postln,
          \base2, 400,
          \farray2, Array.series(32, 1, 1.1).postln,
          \aarray2, Array.series(32, 1, 1).reciprocal.normalizeSum.postln,
          \parray2, Array.fill(32, 0.0).postln).rebuild;
)
```

和音と音階

周波数マトリクスをソートしシリアルにスキャンすると、音階(音を高低の順番に並べたもの)が得られる。

```
// 倍音列と長3度音程列から生れる音階(基音100[Hz])
(
var base = 100;
var farray = (Array.series(8, 1, 1) * .x Array.geom(8, 1, 5/4)).sort.postln;
~out = Pbind(
        \instrument, \line,
        \amp, 0.5,
        \freq, Pseq(farray * base, inf),
        \sustain, 0.1,
        \dur, 0.1,
        \pan, 0.0
    );
)



// 倍音列と4度音程列から生れる音階(基音100[Hz])
(
var base = 100;
var farray = (Array.series(8, 1, 1) * .x Array.geom(8, 1, 4/3)).sort.postln;
~out = Pbind(
        \instrument, \line,
        \amp, 0.5,
```

```
        \freq, Pseq(farray * base, inf),
        \sustain, 0.1,
        \dur, 0.1,
        \pan, 0.0
    );
)


// 倍音列と5度音程列から生れる音階(基音50[Hz])
(
var base = 50;
var farray = (Array.series(8, 1, 1) * .x Array.geom(8, 1, 3/2)).sort.postln;
~out = Pbind(
        \instrument, \line,
        \amp, 0.5,
        \freq, Pseq(farray * base, inf),
        \sustain, 0.1,
        \dur, 0.1,
        \pan, 0.0
    );
)


// 下方倍音列と下行長3度音程列から生れる音階(基音11025[Hz])
(
var base = 11025;
var farray = (Array.series(8, 1, 1).reciprocal * .x Array.geom(8, 1, 3/4)).sort.postln;
~out = Pbind(
        \instrument, \line,
        \amp, 0.5,
        \freq, Pseq(farray * base, inf),
        \sustain, 0.1,
        \dur, 0.1,
        \pan, 0.0
    );
)

// 倍音列と四分音程列(24音音階)から生れる音階(基音50[Hz])
(
var base = 50;
var farray = (Array.series(8, 1, 1) * .x Array.geom(24, 1, 2**(1/24)).postln).sort.postln;
~out = Pbind(
        \instrument, \line,
        \amp, 0.5,
        \freq, Pseq(farray * base, inf),
        \sustain, 0.1,
        \dur, 0.1,
        \pan, 0.0
    );
)
```

音階の部分集合としての和音

逆に音階の部分集合としての和音を考えることも可能である。

```
// 64本(8×8のマトリクス)の線を重ねて鳴らす関数を定義する
(
~out = {arg base = 441, level = 0.8;
        var farray, aarray, parray;
        farray = Control.names(\farray).kr(Array.fill(64, 0));
        aarray = Control.names(\aarray).kr(Array.fill(64, 0));
        parray = Control.names(\parray).kr(Array.fill(64, 0));
        Mix.new(
            Pan2.ar(
                SinOsc.ar(
                    farray * base, 0.0, aarray, 0.0),
```

```
                parray, level)};
)


// ホールトーン音程の8音和音マトリックスの部分集合(発音率30[%]、基音441[Hz]、ランダム振幅、ランダムパン)
(
~out.setn(\base, 441,
          \farray, (Array.series(8, 1, 1) * .x Array.geom(8, 1, 2**(1/6))).postln,
          \aarray, (Array.rand(64, 0.0, 1.0) * Array.fill(64, {[0, 1].wchoose([0.7,
0.3])})).normalizeSum.postln,
          \parray, Array.rand2(64, 1.0).postln).rebuild;
)
```

音響和音のランダム・リスニング(音響和声のための準備)

```
// TaskProxyの定義
x = TaskProxy.basicNew;
x.play;


// ランダムな音程による2つの音程列の組み合わせ(基音20[Hz]、ランダム振幅、ランダムパン)
(
var intervals = [16/15, 10/9, 9/8, 6/5, 5/4, 4/3, 2.sqrt, 3/2, 8/5, 5/3, 7/4, 16/9, 15/8];
x.source = {
    loop {
        ~out.setn(
            \base, 20,
            \farray, Array.geom(8, 1, intervals.choose.postln) * .x Array.geom(8, 1,
intervals.choose.postln),
            \aarray, Array.rand(64, 0.0, 1.0).normalizeSum,
            \parray, Array.rand2(64, 1.0)
        ).rebuild;
        10.wait;
    }
}
)
~out.scope

// 狭い音程列と広い音程列のランダムな組み合わせ
(
x.source = {
    loop {
        ~out.setn(
            \base, 20,
            \farray, Array.geom(8, 1, rrand(1.0, 1.1).postln) * .x Array.geom(8, 1, rrand(1.9,
2.0).postln),
            \aarray, Array.rand(64, 0.0, 1.0).normalizeSum,
            \parray, Array.rand2(64, 1.0)
        ).rebuild;
        10.wait;
    }
}
)

// ペンタトニック音程の8音和音マトリックスの部分集合(発音率20[%]、基音441[Hz]、ランダム振幅、ランダムパン)
(
x.source = {
    loop {
        ~out.setn(
            \base, 441,
            \farray, Array.series(8, 1, 1) * .x Array.geom(8, 1, 2**(1/5)),
            \aarray, (Array.rand(64, 0.0, 1.0) * Array.fill(64, {[0, 1].wchoose([0.8,
0.2])})).normalizeSum,
            \parray, Array.rand2(64, 1.0)
        ).rebuild;
```

```
        10.wait;
    }
}
)


x.stop;
```