コード・コンポジション入門

ver.3.0 α[1]

久保田晃弘[2]＋山路敦司

平成 20 年 1 月 13 日

[1] http://dp.idd.tamabi.ac.jp/dsc/

[2] akihiro.kubota@nifty.com

目 次

第 0 章　はじめに **3**

第 1 章　素材 (material) **4**

1.1　3 つの基本素材 4

1.2　素材の可視化 5

1.3　素材の定義 5

第 2 章　聴取 (listening) **8**

2.1　点 8

2.2　点列 8

2.3　線 8

2.4　面 9

2.5　絶対値と相対値 11

第 3 章　調律 (tuning) **14**

3.1　$44100 = (2 \times 3 \times 5 \times 7)^2$ 14

3.2　デジタル純正律 15

第 4 章　記憶 (memory) **17**

4.1　3 つの時間領域 17

4.2　音響旋律—すべての音は旋律である 19

第 5 章　層 (layer) **21**

5.1　音響対位法 21

5.2　フィードバック 23

第 6 章　塊 (cluster) **26**

6.1　音響和音 26

6.2　和音と音階 33

第 7 章　色彩 (color) **35**

7.1　色と音 35

7.2　ノイズ彫刻 38

第 8 章　式 (formula) **39**

第 9 章　変調 (modulation) **40**

第 10 章 型 (pattern) **41**

第 11 章 流動 (flux) **42**

第0章　はじめに

第1章　素材 (material)

1.1　3つの基本素材

1.1.1　まず初めに、点・線・面という幾何的、構成的なメタファによる、3つの基本的な音響素材を定義する。

点 クリック

線 サイン波

面 ホワイトノイズ

1.1.1　準備

コード 1.1

```
(
// インターナル・サーバーをデフォルトとし変数 s に代入して起動
Server.default = Server.internal;
s = Server.default;
if(not(s.serverRunning), {s.boot});
// プロキシ空間を用意し変数 p に代入
p = ProxySpace.push(s);
// フェイドアウト時間を 1 秒に設定
p.fadeTime = 1;
)
```

コード 1.2

```
(
// 出力用プロキシ (out) を用意
~out.ar(2);
// モニター開始
~out.play;
// 音量を設定
~out.vol = 1.0;
)
```

1.2 素材の可視化

1.2.1 波形とスペクトルによって素材を可視化する[1]。

コード 1.3

```
(
// 波形の表示
Stethoscope.new(s, 2, zoom:0.2);
// スペクトルの表示
FreqScope.new(512, 200, 0, Color.yellow, Color.black);
)
```

1.3 素材の定義

1.3.1 素材をパラメータ (parameter)[2]によって表現する。パラメータによって音を定量化することができる。

点 周波数 (freq)[Hz]・振幅 (amp)・パン (pan)・持続 (sustain)[sec]

線 周波数 (freq)[Hz]・振幅 (amp)・パン (pan)・持続 (sustain)[sec]

面 振幅 (amp)・パン (pan)・持続 (sustain)[sec]

1.3.2 パラメータの数を次元 (dimension) という。点列と線の次元は 4、面の次元は 3 である。

1.3.1 定常パラメータ

コード 1.4

```
(
// 点 (列)：points
SynthDef(\points, {arg freq=1, amp=0.5, pan=0, sustain=1;
    var env = Env.linen(0, sustain, 0, amp, 0);
    Out.ar(0,
        Pan2.ar(
        Impulse.ar(freq) * EnvGen.ar(env, doneAction:2),
        pan))
    }).store;
// 線：line
SynthDef(\line, {arg freq=441, amp=0.5, pan=0, sustain=1;
    var env = Env.linen(0, sustain, 0, amp, 0);
    Out.ar(0,
        Pan2.ar(
        SinOsc.ar(freq) * EnvGen.ar(env, doneAction:2),
        pan))
    }).store;
```

[1] Lance J. Putnam 氏によるクラスライブラリ (http://www.uweb.ucsb.edu/~ljputnam/sc3.html) をインストールする。

[2] SynthDef の引数

```
// 面：plane
SynthDef(\plane, {arg amp=0.5, pan=0, sustain=1;
    var env = Env.linen(0, sustain, 0, amp, 0);
    Out.ar(0,
        Pan2.ar(
        WhiteNoise.ar * EnvGen.ar(env, doneAction:2),
        pan))
    }).store;
)
```

1.3.2 パラメータの時間変化

1.3.2.1 パラメータを初期値から終値へと線形変化させる[3]。

コード 1.5

```
(
// 点 (列)：points2
SynthDef(\points2, {
    arg freq1=2, freq2=0.5, amp1=0.2, amp2=0.7, pan1= -1, pan2=1, sustain=1;
    var env = Env.new([amp1, amp2], [sustain]);
    Out.ar(0,
        Pan2.ar(
        Impulse.ar(Line.kr(freq1, freq2, sustain)) * EnvGen.ar(env, doneAction:2),
        Line.kr(pan1, pan2, sustain)))
    }).store;
// 線：line2
SynthDef(\line2, {
    arg freq1=4410, freq2=441, amp1=0.7, amp2=0.2, pan1= -1, pan2=1, sustain=1;
    var env = Env.new([amp1, amp2], [sustain]);
    Out.ar(0,
        Pan2.ar(
        SinOsc.ar(Line.kr(freq1, freq2, sustain)) * EnvGen.ar(env, doneAction:2),
        Line.kr(pan1, pan2, sustain)))
    }).store;
// 面：plane2
SynthDef(\plane2, {
    arg amp1=0.7, amp2=0.2, pan1= -1, pan2=1, sustain=1;
    var env = Env.new([amp1, amp2], [sustain]);
    Out.ar(0,
        Pan2.ar(
        WhiteNoise.ar * EnvGen.ar(env, doneAction:2),
        Line.kr(pan1, pan2, sustain)))
    }).store;
)
```

[3]パラメータのうち「1」で終わるものが初期値、「2」で終わるものが終値である。

1.3.3 定義した SynthDef を確認する

コード 1.6

```
(
SynthDescLib.global.read;
SynthDescLib.global.browse;
)
```

課題 1.1：環境設定

SuperCollider3 をコンピュータにインストールし、本書のサンプル・コードやエクササイズを実行できる環境を整えよ。定義した音響素材が鳴り、波形やスペクトルを視ることができることを確認せよ。

第2章　聴取 (listening)

2.1 コンポジションは聴くことである。しかしながら、音を音のまま聴くことはできない。物理現象として同じ音であっても、10 人いれば 10 通りの聴取が存在する。音を聴くことは音と聴き手の相互作用である。

2.1　点

2.1.1 デジタル・サウンドは点 (クリック) の集合体である。点はあらゆるデジタル・サウンドの源泉である。点にはあらゆる周波数成分が含まれている。

2.2　点列

2.2.1 点を反復させることで点列が生まれる。点列は反復の源泉である。

2.2.1　点列を聴く

コード 2.1

```
// 1[Hz] の点列
~out = { Impulse.ar(1, 0, 0.5, 0).dup };
// 10[Hz] の点列
~out = { Impulse.ar(10, 0, 0.5, 0).dup };
// 100[Hz] の点列
~out = { Impulse.ar(100, 0, 0.5, 0).dup };
// 1000[Hz] の点列
~out = { Impulse.ar(1000, 0, 0.5, 0).dup };
// 10000[Hz] の点列
~out = { Impulse.ar(10000, 0, 0.5, 0).dup };
// 終了
~out = nil;
```

2.3　線

2.3.1 点の移動と連続が線をつくり、線は持続を生み出す。時間領域における点は周波数領域における線であり、時間領域における線は周波数領域における点である。

2.3.1 線を聴く

コード 2.2

```
// 1[Hz] の線
~out = { SinOsc.ar(1, 0, 0.5, 0).dup };
// 10[Hz] の線
~out = { SinOsc.ar(10, 0, 0.5, 0).dup };
// 100[Hz] の線
~out = { SinOsc.ar(100, 0, 0.5, 0).dup };
// 1000[Hz] の線
~out = { SinOsc.ar(1000, 0, 0.5, 0).dup };
// 10000[Hz] の線
~out = { SinOsc.ar(10000, 0, 0.5, 0).dup };
// 終了
~out = nil;
```

2.4 面

2.4.1 点の運動と跳躍が面をつくる。面にはすべての点とすべての線が含まれている。

2.4.1 面を聴く

コード 2.3

```
// 振幅 0.001 の面
~out = { WhiteNoise.ar(0.001, 0).dup };
// 振幅 0.01 の面
~out = { WhiteNoise.ar(0.01, 0).dup };
// 振幅 0.1 の面
~out = { WhiteNoise.ar(0.1, 0).dup };
// 振幅 1 の面
~out = { WhiteNoise.ar(1, 0).dup };
// 終了
~out = nil;
```

2.4.2 1.3 節で定義した SynthDef を用いて素材を聴く

コード 2.4

```
// パラメータの値をセットする
~out.set(\freq, 100, \amp, 0.5, \pan, 0, \sustain, 1);
// 出力を点にアサインする
~out = \points;
```

```
// 出力を線にアサインする
~out = \line;
// パラメータの値を変更する
~out.set(\freq, 10000);
~out = \line;
// 出力を面にアサインする
~out = \plane;
// パラメータを時間変化させる
~out.set(\freq1, 1, \freq2, 100, \amp1, 0.5, \amp2, 0.5, \pan1, -1, \pan2, 1, \sustain, 10);
// 出力を時間変化する点にアサインする
~out = \points2;
// 出力を時間変化する線にアサインする
~out = \line2;
// 出力を時間変化する面にアサインする
~out = \plane2;
```

課題 2.1：パラメトリック・リスニング

素材パラメータをシステマティックに組み合わせていくことで音のサーベイを行ってみよ。

課題 2.2：ランダム・リスニング

素材パラメータをランダムに選択し、繰り返し聴くことで発見的 (heuristic) に音を探索せよ。

課題 2.3：デジタル・サウンド・ディクテーション (聴音)

ランダムに生成される音のパラメータを聴くことで推測せよ。

コード 2.5

```
// Task の定義
(
var freq, amp, pan, sustain;
Tdef(\x, {
    loop ({
        freq = 22100.0.rand.post; ", ".post; // 周波数
        amp = 1.0.rand.post; ", ".post; // 振幅
        pan = 1.0.rand2.post; ", ".post; // パン
        sustain = 5.0.rand.post; ", ".post; // 継続時間
        ~out.set(\freq, freq, \amp, amp, \pan, pan, \sustain, sustain);
        ~out = [\points, \line, \plane].choose.postln;
        5.wait;
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

2.5　絶対値と相対値

2.5.1　素材を表現するパラメータには絶対値と相対値の 2 つの表現方法がある。絶対値とはパラメータの値そのものをのことであり、相対値とは他の基準となる値との比のことである。

2.5.2　音程や音量など、人間の感覚はおおよそ刺激の対数に比例するため[1]、相対値の場合、パラメータを対数スケールで表現 (マッピング) することが多い。

2.5.3　音程の単位であるセントは対数スケールである。n セントは 1 オクターブの 1200 分の n の音程であり、その周波数比は $2^{\frac{n}{1200}}:1$ である[2]。

2.5.4　音量の単位であるデシベル (dB) も対数スケールである。音の強さが倍になるとデシベル値が約 3[dB]、4 倍になると約 6[dB] 増加し、半分になると約 3[dB] 低下する。

課題 2.4：インタラクティヴ・リスニング

マウス座標をパラメータにマッピングすることで、インタラクティヴに音を探索せよ。マウス座標は 2 次元なので、同時に 2 つのパラメータを操作することができる。

コード 2.6

```
(
// 周波数、振幅、パンのプロキシの定義
~freq.kr(1);
~amp.kr(1);
~pan.kr(1);
)
(
// 初期値の代入
~freq = 4410;
~amp = 0.5;
~pan = 0;
)
// 点列を鳴らす
~out = { Pan2.ar(Impulse.ar(~freq.kr, 0, ~amp.kr), ~pan.kr(1)) };
// マウスの X 座標を周波数に対数マッピングする
~freq = { MouseX.kr(20.0, 22050.0, \exponential) };
// マウスの Y 座標を振幅に対数マッピングする
~amp = { MouseY.kr(1.0, 0.01, \exponential) };
// マウスの X 座標をパンに線形マッピングする
~pan = { MouseX.kr(-1.0, 1.0, \linear) };
// 線を鳴らす
~out = { Pan2.ar(SinOsc.ar(~freq.kr, 0, ~amp.kr), ~pan.kr(1)) };
// 面を鳴らす
~out = { Pan2.ar(WhiteNoise.ar(~amp.kr), ~pan.kr(1)) };
// 終了
~out = nil;
```

2.5.1　マウスモニター

2.5.1.1　マウスの座標をリアルタイム表示する。マウス座標はデスクトップの左上が (0, 0)、右下が (1, 1) である。

[1] フェヒナーの法則 (Fechner's Law)
[2] 12 平均律の全音は 200 セント、半音は 100 セントとなる。

コード 2.7

```
// マウスモニターの SynthDef 定義
(
SynthDef(\mousexToClient, { |trig_id = 0|
    var    mousex, mousey, trig;
    mousex = MouseX.kr(0, 1, \linear, 0);
    mousey = MouseY.kr(0, 1, \linear, 0);
    trig = HPZ1.kr(mousex).abs;
    trig = HPZ1.kr(mousey).abs;
    SendTrig.kr(trig, 0, mousex);
    SendTrig.kr(trig, 1, mousey);
}).send(s);
)
a.free;
a = Synth(\mousexToClient);
// マウス座標をウィンドウに表示する
(
var win, mouse, x, y, text, resp;
win = SCWindow("mouse", Rect(100, 200, 350, 40)).front;
win.view.background_(Color.white);
win.view.decorator = FlowLayout(win.view.bounds);
mouse = SCStaticText(win, 350@20).font_(Font("Monaco", 12)).stringColor_(Color.black);

resp = OSCresponderNode(s.addr, '/tr', { |time, resp, msg|
    if( msg[2] == 0, { x = msg[3] }, { y = msg[3] });
    text = "[ x =" + x.asString + ", y =" + y.asString + "]";
    { mouse.string_(text); }.defer;
}).add;
win.onClose_({resp.remove});
)
```

課題 2.5：音の記憶

音を言葉で表現することは難しい。音の知覚や体験そのものを記憶しておかなければならない。以下のコードから生み出される音を鳴らす前に思い出してみよ。

コード 2.8

```
// サウンド 1(最短の音)
~out.set(\freq, 10, \amp, 1.0, \pan, 0, \sustain, 0.01);
~out = \points;
// サウンド 2(最小の音)
~out.set(\freq, 50, \amp, 1/32768, \pan, 0, \sustain, 3);
~out = \points;
// サウンド 3
~out.set(\freq, 14000, \amp, 1.0, \pan, 0, \sustain, 3);
~out = \points;
// サウンド 4
~out.set(\freq, 35, \amp, 1.0, \pan, 0.0, \sustain, 3);
~out = \line;
// サウンド 5
~out.set(\amp, 0.0005, \pan, 0, \sustain, 30.0);
```

```
~out = \plane;
// サウンド6
~out.set(\amp, 0.5, \pan, 0, \sustain, 0.001);
~out = \plane;
// サウンド7
~out.set(\freq1, 12000, \freq2, 12000, \amp1, 0, \amp2, 0.9, \pan1, 0.0, \pan2, 0.0, \sustain, 30);
~out = \points2;
// サウンド8
~out.set(\freq1, 1000, \freq2, 50, \amp1, 0.5, \amp2, 0.5, \pan1, 0.0, \pan2, 0.0, \sustain, 0.1);
~out = \line2;
```

第3章　調律 (tuning)

3.1　$44100 = (2 \times 3 \times 5 \times 7)^2$

3.1.1　サンプリング周波数はデジタルサウンドの基音である。

3.1.2　コンパクト・ディスク (CD) 等に採用されているサンプリング周波数 44100[Hz] は、$(2 \times 3 \times 5 \times 7)^2$ [Hz] と素因数分解できる。

3.1.1　44100 の約数による音階

3.1.1.1　44100 の約数は [1, 2, 3, 4, 5, 6, 7, 9, 10, 12, 14, 15, 18, 20, 21, 25, 28, 30, 35, 36, 42, 45, 49, 50, 60, 63, 70, 75, 84, 90, 98, 100, 105, 126, 140, 147, 150, 175, 180, 196, 210, 225, 245, 252, 294, 300, 315, 350, 420, 441, 450, 490, 525, 588, 630, 700, 735, 882, 900, 980, 1050, 1225, 1260, 1470, 1575, 1764, 2100, 2205, 2450, 2940, 3150, 3675, 4410, 4900, 6300, 7350, 8820, 11025, 14700, 22050, 44100] の 81 個である。

コード 3.1

```
//44100 の約数を求める
(1..44100).removeAllSuchThat({ |i| (44100 % i) == 0 });
```

コード 3.2

```
// Task の定義
(
var freq = (1..44100).removeAllSuchThat({ |i| (44100 % i) == 0 });
~out.set(\amp, 0.5, \pan, 0, \sustain, 1);
Tdef(\x, {
    (1..freq.size).do({ |i|
        ~out.set(\freq, freq.at(i-1).postln);
        ~out = \points;
        1.wait;
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

3.2 デジタル純正律

3.2.1 サンプリング周波数の単純な整数比の周波数の音による音律を「デジタル純正律 (digital just temperament)」という。

3.2.2 44100 の整数分の 1 の周波数 (整数倍の周期) を高い方から 20 個書き出してみる。

1. 44100[Hz]
2. 22050[Hz]
3. 14700[Hz]
4. 11025[Hz]
5. 8820[Hz]
6. 7350[Hz]
7. 6300[Hz]
8. 5512.5[Hz]
9. 4900[Hz]
10. 4410[Hz]
11. 4009.0909090909[Hz]
12. 3675[Hz]
13. 3392.3076923077[Hz]
14. 3150[Hz]
15. 2940[Hz]
16. 2756.25[Hz]
17. 2594.1176470588[Hz]
18. 2450[Hz]
19. 2321.0526315789[Hz]
20. 2205[Hz]

コード 3.3

```
// 44100 の整数分の 1 を求める
(1..20).collect({|i| 44100 / i});
```

3.2.1 純正律の点列

3.2.1.1 サンプリング周期の 2 倍から 20 倍までの周波数の点列を順に聴く。

コード 3.4

```
// Task の定義
(
~out.set(\amp, 0.5, \pan, 0, \sustain, 5);
Tdef(\x, {
    (2..20).do({ |i|
        ~out.set(\freq, (44100/i).postln); // 純正律周波数
```

```
        ~out = \points;
        5.wait;
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

3.2.1.2 純正律との差異を聴く。差異の周波数の音（うなり）が聴こえる。

コード 3.5

```
// Task の定義
(
~out.set(\amp1, 0.5, \amp2, 0.5, \pan1, 0, \pan2, 0, \sustain, 5);
Tdef(\x, {
    (2..20).do({ |i|
        ~out.set(\freq1, (44100/i).postln, \freq2, (44100/i)+20); // 周波数を 20[Hz] ずらす
        ~out = \points2;
        5.wait;
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

3.2.1.3 サンプリング周波数の i 倍の音から $i+1$ 倍の周波数の音へと連続的に変化させる。途中でどのように聴こえるか。

コード 3.6

```
// Task の定義
(
~out.set(\amp1, 0.5, \amp2, 0.5, \pan1, 0, \pan2, 0, \sustain, 30);
Tdef(\x, {
    (2..20).do({ |i|
        ~out.set(\freq1, (44100/i).postln, \freq2, (44100/(i+1))); // 周波数の初期値と終値
        ~out = \points2;
        35.wait;
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

課題 3.1：線の純正律

点列 (points/points2) を線 (line/line2) に置き換えて純正律の音とそこからの差異や遷移を聴いてみよ。点列と線でどのように聴こえ方が異なるだろうか。

第4章　記憶 (memory)

4.1　3つの時間領域

4.1.1　記憶を基準として、人間の聴取経験は大きく 3 つの時間スケールに分けることができる [1]。

トーン領域 音のピッチや大きさ、音色などの直接的な知覚
フレーズ領域 メロディーやリズムといったグルーピングによる認知
形式領域 カノンやフーガ、ソナタなどのマクロな音楽構造の認知

4.1.2　トーン領域の経験は、人間の感覚記憶[1]に対応している。受け取った刺激をそのままの形で短時間保存するのが感覚記憶である。2 つの音の間隔が可聴周波数の下限である 20 分の 1 秒 (= 2205 サンプル) 以下の場合、2 つの音は融合してピッチを形成し、その順序を聴きとることはできない。

4.1.3　フレーズ領域の経験は、短期記憶の保持時間に対応している。人間の短期記憶の個数は 5〜9 個であり[2]、保持時間の限界は 15〜30 秒程度である。短期記憶の容量が経験のチャンクを形成する。

4.1.4　トーン領域の時間は高低 (ピッチ) のメタファで表現される。フレーズ領域の時間は速度 (テンポ) のメタファで表現される。

4.1.5　形式領域の経験は、半永久的に保持されている長期記憶による、大きな尺度の認知である。

4.1.1　聴覚の境界領域

4.1.1.1　120[bps] のリズムは 2[Hz] の音、20[Hz] の音は 1200[bps] のリズムである。融合領域からフレーズ領域に移行する、聴覚知覚の境界領域を聴く。

コード 4.1

```
// 0〜50[Hz] の間を周期的に変動する点列 (左右で位相を π ずらす)
~out = { Impulse.ar(SinOsc.kr(0.1, 0, 25, 25), 0.8).dup };
// 終了
~out = nil;
```

4.1.2　音の短期記憶

4.1.2.1　0.05〜0.2 秒の任意の間隔で左右の耳に 5〜 9 個 (短期記憶の容量) の短い音を鳴らす。音の順序を思いだすことができるか？

コード 4.2

[1] 視覚情報の感覚記憶をアイコニックメモリー、聴覚情報の感覚記憶を エコイックメモリーという。
[2] 「マジックナンバー 7 ± 2」 と呼ばれている。

```
// Task の定義
(
var pan, sustain, interval;
~out.set(\freq, 441, \amp, 0.8);
Tdef(\x, {
    loop({
        rrand(5, 9).postln.do { // 音の個数
            pan = 1.rand2;
            sustain = rrand(0.005, 0.05); // 音の持続時間
            interval = rrand(0.05, 0.2); // 音の間隔
            ~out.set(\pan, pan, \sustain, sustain);
            ~out = [\points, \line, \plane].choose.postln;
            interval.wait;
        };
        5.wait;
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

4.1.3 線から点への遷移

4.1.3.1 音の長さとピッチ感覚は密接に関連している。1000[Hz] 以上の音の場合、音の長さとピッチ感覚の関係はおおよそ以下の通りである。

約 5ms 以下 クリック (ピッチが知覚できない)

約 5〜10ms クリックピッチ (高低の判別は可能)

約 10ms 以上 トーンピッチ (ピッチの知覚が可能)

4.1.3.2 8820[Hz] の線の長さを、0.1[s] から 20%ずつ短かくしていく。ピッチはいつまで聴きとることができるか?

コード 4.3

```
// Task の定義
(
var sustain = 0.1; // 初期値
~out.set(\freq, 8820, \amp, 0.8, \pan, 0);
Tdef(\x, {
    while({ sustain > 44100.reciprocal }, { // 終了条件
        ~out.set(\sustain, sustain.postln);
        ~out = \line;
        sustain = sustain * 0.8; // 長さを 20%ずつ短くしていく
        1.wait;
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

課題 4.1：面から点への遷移

面の長さを短くしていくことによるピッチ知覚の変化を経験せよ。

4.2　音響旋律—すべての音は旋律である

4.2.1　パラメトリックな操作によって、トーン/フレーズ/形式領域の 3 つの時間スケール (記憶タイプ) を統合的に取り扱うことができる。

4.2.2　フレーズ領域の経験としてのメロディーを他の時間スケールへと拡張することで、旋律の概念を拡張することができる。

4.2.3　音響旋律は、従来の旋律におけるフレーズ領域での音のグルーピングの概念を拡張し、音そのものが内包しているミクロな状態とその関係性や、長い時間にわたるパラメータの微小な変化に着目する。音響旋律は、音内部のより微小で精緻な構造を聴き取るための概念でもある。

課題 4.2：マイクロ・メロディー

0.01[S] の長さのランダムな点/線/面の連鎖から何が聴き取れるか。音の長さやパン、周波数/振幅の範囲を変化させて、どのように旋律が変化するかを確かめよ。

コード 4.4

```
// Task の定義
(
var freq, amp, pan = 0.0, sustain = 0.01; // 初期値
Tdef(\x, {
    loop({
        freq = exprand(20.0, 20000.0); // 周波数レンジ
        amp = exprand(0.1, 1); // 振幅レンジ
        ~out.set(\freq, freq, \amp, amp, \pan, pan, \sustain, sustain);
        ~out = [\points, \line, \plane].choose.postln;
        sustain.wait;
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

課題 4.3：エクステンディド・メロディー

非常にゆっくりとパラメータが遷移する点列や線の長い持続から何が聴き取れるか。

コード 4.5

```
// Task の定義
(
var freq, amp = 0.5, pan = 0.0, sustain = 300.0; // 初期値
~out.set(\amp1, amp, \amp2, amp, \pan1, 0.0, \pan2, 0.0, \sustain, sustain);
```

```
Tdef(\x, {
    loop({
        freq = exprand(20.0, 20000.0); // 周波数レンジ
        amp = exprand(0.1, 1); // 振幅レンジ
        ~out.set(\freq1, freq, \freq2, freq*1.1); // 300[秒] で周波数が 1.1 倍に変化する
        ~out = [\points2, \line2].choose.postln;
        100.wait; // 同時に 3 本の点列/線が鳴る
    });
})
)
// Task の実行と停止
Tdef(\x).play;
Tdef(\x).stop;
```

第5章　層 (layer)

5.1　音響対位法

5.1.1　旋律の重ね合わせ

5.1.1.1　対位法とは「複数の旋律を対等な関係として組み合わせる」作曲技法のことである。ここでは、4.2 節で定義した音響旋律の重ね合わせとしての音響対位法を考える。

5.1.1.2　まず最初に、パラメータ一定の持続する旋律 (定旋律) を鳴らす。

コード 5.1

```
// 出力用プロキシのクリア
~out.clear;
~out.play;
// 11025[Hz] の点列による純正律定旋律
~out[0] = { Impulse.ar(11025, 0, 0.1).dup };
```

5.1.1.3　ここに複数の旋律を重ね合わせていくことで、層状の対位法構造をつくり出す。

5.1.2　差異

コード 5.2

```
// 周波数を少しずらした対旋律を重ねる
~out[1] = { Impulse.ar(11028, 0, 0.1).dup };
```

5.1.2.1　周波数の違いはパルス (うなり) を生み出す。単独で鳴らした場合と 2 つの旋律を重ねた場合とどのように違うか。

コード 5.3

```
// さらに周波数をずらした (3 本目の) 対旋律を重ねる
~out[2] = { Impulse.ar(11050, 0, 0.1).dup };
```

5.1.3 変動

5.1.3.1 点列の周波数をゆっくりと変動させる。

コード 5.4

```
// 2つめの対旋律の周波数をゆっくりと変動させる
~out[1] = { Impulse.ar(11028 + LFNoise1.ar(0.1, 5), 0, 0.1).dup };
// 3つめの対旋律の周波数もゆっくりと変動させる
~out[3] = { Impulse.ar(11050 + LFNoise1.ar(0.1, 5), 0, 0.1).dup };
```

5.1.3.2 周波数の微小な変動 (干渉) によって、響きはどのように変化するか。パラメータの微小な差異が、大きな知覚の変化を生み出すことがある。

5.1.3.3 3つの旋律の周波数の相対的な関係は一定のまま、基音 (全体の音の高さ) をゆっくりと変動 (平行) させる。

コード 5.5

```
// 変動用プロキシ
~fluctuation = { LFNoise1.ar(1.0, 5) };
// 定旋律
~out[0] = { Impulse.ar(11025 + ~fluctuation.ar, 0, 0.1).dup };
// 対旋律 (1)
~out[1] = { Impulse.ar(11028 + ~fluctuation.ar, 0, 0.1).dup };
// 対旋律 (2)
~out[2] = { Impulse.ar(11050 + ~fluctuation.ar, 0, 0.1).dup };
// 対旋律 (2) の変動を反行させる
~out[2] = { Impulse.ar(11050 - ~fluctuation.ar, 0, 0.1).dup };
```

5.1.4 対比

5.1.4.1 パラメータが大きく異なる旋律を重ねる。パラメータの対比によって生み出されるスペースを聴く。

コード 5.6

```
// 出力用プロキシのクリア
~out.clear;
~out.play;
// 60[Hz] の線による定旋律
~out[0] = { SinOsc.ar(60, 0, 0.5).dup };
// 周波数の大きく異なる対旋律を重ねる
~out[1] = { SinOsc.ar(8000, 0, 0.1).dup };
// さらに周波数の大きく異なる対旋律を重ねる
~out[2] = { SinOsc.ar(14000, 0, 0.1).dup };
```

5.1.4.2 3つの旋律の振幅を異なる周期でゆっくりと変動させる。

コード 5.7

```
// 定旋律
~out[0] = { SinOsc.ar(60, 0, 0.5 + LFNoise1.ar(1.0, 0.2)).dup };
// 対旋律 (1)
~out[1] = { SinOsc.ar(8000, 0, 0.1 + LFNoise1.ar(0.5, 0.1)).dup };
// 対旋律 (2)
~out[2] = { SinOsc.ar(14000, 0, 0.1 + LFNoise1.ar(0.2, 0.1)).dup };
```

課題 5.1：さまざまな対位法

- 線あるいは点列による旋律のパンや位相をゆっくりと変動させながら重ねてみよ。
- 点列と線、線と面といった、異なる種類の素材による旋律を重ねてみよ。

コード 5.8

```
// 出力用プロキシのクリア
~out.clear;
~out.play;
// 位相をゆっくりと変動させた線による定旋律
~out[0] = { SinOsc.ar(11025, LFNoise1.ar(1.0, 0.5, 0.5), 0.1).dup };
// 位相をゆっくりと変動させた対旋律 (1)
~out[1] = { SinOsc.ar(11028, LFNoise1.ar(1.0, 0.5, 0.5), 0.1).dup };
// 位相をゆっくりと変動させた対旋律 (2)
~out[3] = { SinOsc.ar(11050, LFNoise1.ar(1.0, 0.5, 0.5), 0.1).dup };
// 11020[Hz] の点列を重ねる
~out[4] = { Impulse.ar(11020, 0, 0.1).dup };
// 点列のパンを左右に振動させる
~out[4] = { Pan2.ar(Impulse.ar(11020, 0, 0.1), SinOsc.ar(1.0, 1.0)) };
// ゆっくりと振幅が変動する小さな面を重ねる
~out[5] = { WhiteNoise.ar(LFNoise1.ar(1.0, 0.001, 0.001)).dup };
```

5.2 フィードバック

5.2.1 フィードバックは自己参照による音響レベルのカノン[1]であり、音響対位法の一種である。

コード 5.9

```
// 出力用プロキシのクリア
~out.clear;
~out.play;
// ゲイン・コントロールおよびフィードバック用のプロキシ
~gain.ar(1);
~feedback = ~out * ~gain;
// 10[Hz] の点列をフィードバックさせる
~gain = 0.3; // ゲインは 0.3
~out = { Impulse.ar(10, 0, 0.1, ~feedback.ar).dup };
```

[1] カノンは、複数の声部が同じ旋律を異なる時点からそれぞれ開始して演奏する様式であり、模倣対位法のひとつである。カノンは一般に輪唱と訳されるが、輪唱が全く同じ旋律を追唱するのに対し、カノンでは、異なる音で始まるものや、リズムが2倍になったり、音高の上下や時間の前後が逆になったものも含まれる。

```
// ゲインを周期的にゆっくりと変動させる
~gain = { SinOsc.ar(0.09, 0, 0.5) };
// ゲイン変動の周波数を大きくする
~gain = { SinOsc.ar(1000, 0, 0.5) };
// 11050[Hz] の点列をフィードバックさせる
~gain = -0.4; // ゲインは-0.4
~out = { Impulse.ar(11050, 0, 0.1, ~feedback.ar).dup };
// 点列の周波数をゆっくりと変動させる
~out = { Impulse.ar(11050 + LFNoise1.ar(0.2, 5), 0, 0.1, ~feedback.ar).dup };
// ゲインをゆっくりと変動させる
~gain = { LFNoise1.ar(0.1, 0.1, -0.4) };
```

5.2.2 フィードバックにディレイ (時間軸の移動) やピッチシフト (周波数の移動) をかけるとさまざまな音色を生成することができる。

5.2.3 パラメータの値を変化させると過渡的な過程を聴くことができる。

コード 5.10

```
// フィードバックに 0.1[s] のディレイをかける
~feedback = { DelayN.ar(~out.ar, 1.0, 0.1, ~gain.ar) };
// ディレイの長さを 0.5[s] にする
~feedback = { DelayN.ar(~out.ar, 1.0, 0.5, ~gain.ar) };
// フィードバックをピッチレシオ 1.5 でピッチシフトする
~feedback = { PitchShift.ar(~out.ar, 0.1, 1.5, 0.0, 0.0, ~gain.ar) };
// ピッチレシオを 0.75 にする
~feedback = { PitchShift.ar(~out.ar, 0.1, 0.75, 0.0, 0.0, ~gain.ar) };
// ピッチレシオを 1 の近くでゆっくりと変動させる
~feedback = { PitchShift.ar(~out.ar, 0.1, LFNoise1.ar(0.2, 0.01, 1.0), 0.0, 0.0, ~gain.ar) };
// ディレイとピッチシフトを両方用いる
~feedback = { DelayN.ar(PitchShift.ar(~out.ar, 0.1, 1.5), 1.0, 0.5, ~gain.ar) };
~feedback = { DelayN.ar(PitchShift.ar(~out.ar, 0.1, 0.75), 1.0, 0.5, ~gain.ar) };
~feedback = { DelayN.ar(PitchShift.ar(~out.ar, 0.1, 0.1), 1.0, 0.5, ~gain.ar) };
~feedback = { DelayN.ar(PitchShift.ar(~out.ar, 0.1, 4.0), 1.0, 0.5, ~gain.ar) };
```

課題 5.2：音響カノン

フィードバック・ループから生まれるさまざまな層的な構造を使って「音響カノン」を作曲する。

平行カノン ピッチ・シフトを介したフィードバック

反行カノン ゲインパラメータが負のフィードバック

拡大/縮小カノン タイム・ストレッチを介したフィードバック

多重カノン 複数の旋律によるクロス・フィードバック

コード 5.11

```
// 二重カノン
~out.clear; // 出力用プロキシのクリア
~out.play;
~out = ~out0 + ~out1; // 2 つの旋律をミックス
// ゲイン・コントロールおよびフィードバック用のプロキシ
```

```
~gain.ar(1);
~feedback0 = { DelayN.ar(~out1.ar, 1.0, 0.17, ~gain.ar) };
~feedback1 = { PitchShift.ar(~out0.ar, 0.1, 1.5, 0.0, 0.0, ~gain.ar) };
// ゆっくりと変動する 11050[Hz] と 10[Hz] の点列によるクロス・フィードバック
~gain = -0.4; // ゲインは-0.4
~out0 = { Impulse.ar(11050 + LFNoise1.ar(0.2, 5), 0, 0.1, ~feedback0.ar).dup };
~out1 = { Impulse.ar(10 + LFNoise1.ar(0.1, 1), 0, 0.1, ~feedback1.ar).dup };
```

第6章　塊 (cluster)

6.1　音響和音

6.1.1　和音 (chord) は複数の音が同時に響きあっている状態のことである。

6.1.2　素材としての点 (クリック)、線 (サイン波)、面 (ホワイトノイズ) に、倍音列の組み合わせとしての音色はない。そのため、和音や和声を倍音を前提に考える必要はない。

6.1.3　線の組み合わせとしての「音響和音」を考える。そこには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。

6.1.4　音響和音は、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を、1 つの統合的な枠組みの中で扱うためのフレームワークである。

6.1.1　倍音と等差数列

6.1.1.1　倍音列は線形スケールであり、等差数列で抽象できる。

6.1.1.2　等差数列による線の重ね合せから生れる響きを聴く。

コード 6.1

```
// 16 本の線を重ね合せる関数の定義
(
~out = { arg base = 441, amp = 0.5, pan = 0;
    var farray, aarray;
    farray = Control.names(\farray).kr(Array.fill(16, 0)) * base; // 周波数列
    aarray = Control.names(\aarray).kr(Array.fill(16, 0)); // 振幅列
    Pan2.ar(Klang.ar(`[farray, aarray, nil], 1, 0), pan, amp) };
)
```

コード 6.2

```
// ノコギリ波の合成
(
~out.setn(
    \base, 441, // 基本周波数
    \farray, Array.series(16, 1, 1).postln, // 整数倍音列
    \aarray, Array.series(16, 1, 1).reciprocal.normalizeSum.postln // 振幅反比例
    ).rebuild;
)
```

```
// 矩形波の合成
(
~out.setn(
    \base, 441, // 基本周波数
    \farray, Array.series(16, 1, 2).postln, // 奇数倍音列
    \aarray, Array.series(16, 1, 1).reciprocal.normalizeSum.postln // 振幅反比例
    ).rebuild;
)

// 三角波の合成
(
~out.setn(
    \base, 441, // 基本周波数
    \farray, Array.series(16, 1, 2).postln, // 奇数倍音列
    \aarray, Array.series(16, 1, 1).squared.reciprocal.normalizeSum.postln // 2 乗に反比例
    ).rebuild;
)
```

課題 6.1：等差数列による響き

上記のコードを出発点に、等差数列の初項や公差を変化させることでさまざまな周波数列と振幅列を生成し、そこからどのような響きや音色が生れるかを探求せよ。

コード 6.3

```
// ランダムな公差による波形の合成
(
~out.setn(
    \base, 441, // 基本周波数
    \farray, Array.series(16, 1, 2.0.rand).postln, // 周波数列
    \aarray, Array.series(16, 1, 2.0.rand).reciprocal.normalizeSum.postln // 振幅列
    ).rebuild;
)
```

6.1.2 下方倍音列

6.1.2.1 基音とその整数分の 1 の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。

6.1.2.2 物理的な音響現象としての下方倍音列は存在しないが、倍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。

6.1.2.3 デジタル純正律はサンプリング周波数の下方倍音列である。

コード 6.4

```
// サンプリング周波数を基本周波数とする下方倍音列の重ね合せ
(
~out.setn(
    \base, 44100, // 基本周波数
    \farray, Array.series(16, 1, 1).reciprocal.postln, // 下方倍音列
```

```
    \aarray, Array.series(16, 1, 0).normalizeSum.postln // 振幅一定
    ).rebuild;
)
```

6.1.3 音程と等比数列

6.1.3.1 音程とは音と音の周波数の比である。

6.1.3.2 1:2 の周波数比をオクターブと呼ぶ。オクターブを等分割する n 音 (平均律) 音階の音程は、1 オクターブの n 乗根の比で表わされる。

コード 6.5

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

6.1.3.3 周波数比で表現される音程は対数スケールであり、等間隔の音程列は等比数列で抽象できる。

6.1.3.4 平均律音程による等音程和音の響きを聴く。

コード 6.6

```
// 5 音平均律音階の 3 音音程
(
~out.setn(
    \base, 20, // 基本周波数
    \farray, Array.geom(16, 1, 2**(3/5)).postln, // 音程列
    \aarray, Array.fill(16, 1).normalizeSum.postln // 振幅一定
    ).rebuild;
)
```

```
// 17 音平均律音階の 2 音音程
(
~out.setn(
    \base, 5000, // 基本周波数
    \farray, Array.geom(16, 1, 2**(2/17)).postln, // 音程列
    \aarray, Array.fill(16, 1).normalizeSum.postln // 振幅一定
    ).rebuild;
)
```

6.1.3.5 倍音列に由来する単純な整数比の音程による等音程和音の響きを聴く。

コード 6.7

```
// 短 2 度音程 (周波数比 15:16)
(
~out.setn(
    \base, 70, // 基本周波数
    \farray, Array.geom(16, 1, 16/15).postln, // 音程列
    \aarray, Array.fill(16, 1).normalizeSum.postln // 振幅一定
    ).rebuild;
)

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

課題 6.2：等比数列による響き

等比数列の初項や公比を変化させることでさまざまな周波数列と振幅列を生成し、そこからどのような響きや

音色が生れるかを探求せよ。

コード 6.8

```
// ランダムな公比による音響和音
(
~out.setn(
    \base, 70, // 基本周波数
    \farray, Array.geom(16, 1, rrand(1.0, 2.0)).postln, // 周波数列
    \aarray, Array.fill(16, 1).normalizeSum.postln // 振幅一定
    ).rebuild;
)
// 公比が小さく密集した音響和音
(
~out.setn(
    \base, 7000, // 基本周波数
    \farray, Array.geom(16, 1, 1.01).postln, // 周波数列
    \aarray, Array.exprand(16, 0.5, 1.0).normalizeSum.postln // 振幅は 0.5〜1.0 の指数分布
    ).rebuild;
)
// 等比数列に乱数を加えた音響和音 (乱数を加えない場合と聴き比べてみよ)
(
~out.setn(
    \base, 5000, // 基本周波数
    \farray, (Array.geom(16, 1.0, 1.1) + Array.rand2(16, 0.2)).postln, // 周波数列
    \aarray, Array.fill(16, 1).normalizeSum.postln // 振幅一定
    ).rebuild;
)
```

6.1.4 音色と和音の統合

6.1.4.1 音色を構成する倍音列と和音を構成する音程列の外積としてのマトリクスを考えることで、音色と和音を 1 つの「音響和音」として統合する。

コード 6.9

```
// 64 本 (8 × 8 のマトリクス) の線を重ねて鳴らす関数の定義
(
~out = {arg base = 441, amp = 0.5;
        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, amp))};
)
```

6.1.4.2 抽象された倍音列としての等差数列と、音程列としての等比数列から周波数マトリクスを生成する。

コード 6.10

```
(
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 }); // 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 ]
```

6.1.4.3 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。

コード 6.11

```
(
var aarray1 = Array.series(8, 1, 1).reciprocal.round(0.01).postln; // 振幅反比例
var aarray2 = Array.fill(8, 1).round(0.01).postln; // 振幅一定
(aarray1 * .t aarray2).do({ |row| row.round(0.01).postln }); // 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 ]
```

課題 6.3：マトリクスによる響き

さまざまな周波数マトリクスおよび振幅マトリクスを生成し、そこからどのような響きや音色が生れるかを探求せよ。

コード 6.12

```
// 古典的長 7 和音
```

```
(
~out.setn(\base, 441, // 基本周波数
        \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; // 中央パン
)
// 長 3 度と短 3 度のランダムな組みあわせによる 8 音和音
(
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, // 基本周波数
        \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; // ランダムパン
)
// 長 3 度音程列と短 3 度音程列の組み合わせ
(
~out.setn(\base, 441, // 基本周波数
        \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; // 中央パン
)
// 長 3 度音程列と完全 4 度音程列の組み合わせ
(
~out.setn(\base, 441, // 基本周波数
        \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; // 中央パン
)
// 高域にいくにつれて振幅が大きくなる 5 音音階 (ペンタトニック) 列
(
~out.setn(\base, 441, // 基本周波数
        \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;
)
// 倍音列と下方倍音列の組み合わせ
(
~out.setn(\base, 2100, // 基本周波数
        \farray, (Array.series(8, 1, 1) * .x Array.series(8, 1, 1).reciprocal).postln,
        \aarray, (Array.fill(8, 1, 1) * .x Array.fill(8, 1, 1)).normalizeSum.postln,
        \parray, Array.fill(64, 0.0).postln).rebuild; // 中央パン
)
// 下方倍音列と下行音程列 (短 3 度/長 3 度/5 度) の組み合わせ
(
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, 44100, // 基本周波数
        \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; // 中央パン
```

```
)
// 64 個の短 2 度音程列の組み合わせ
(
~out.setn(\base, 21, // 基本周波数
        \farray, Array.geom(64, 1, 16/15).postln,
        \aarray, Array.rand(64, 0.0, 1.0).normalizeSum.postln, // ランダム振幅
        \parray, Array.rand2(64, 1.0).postln).rebuild; // ランダムパン
)
// ランダムな長短 2 度あるいは長短 7 度の組み合せによるクラスター
(
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, 21, // 基本周波数
        \farray, (intervals1 * .x intervals2).postln,
        \aarray, Array.rand(64, 0.0, 1.0).normalizeSum.postln, // ランダム振幅
        \parray, Array.rand2(64, 1.0).postln).rebuild; // ランダムパン
)
```

6.2 和音と音階

6.2.1 周波数マトリクスをシーケンシャルにスキャンすることで、縦の和音を横の音階 (スケール) に変換することができる。

6.2.2 マトリクスを高速にスキャンすることで、和音と音階のグレーゾーンが形成される[1]。

コード 6.13

```
// Pattern の実行と停止
Pdef(\y).play;
Pdef(\y).stop;
// 倍音列と 4 度音程列から生れる音階
(
var base = 300; // 基本周波数
var farray = (Array.series(8, 1, 1) * .x Array.geom(8, 1, 4/3)).postln;
Pdef(\y,
    Pbind(
        \instrument, \line,
        \amp, 0.1,
        \freq, Pseq(farray * base, inf),
        \sustain, 0.001,
        \dur, 0.001,
        \pan, 0.0)
    )
)
// 下方倍音列と下行長 3 度音程列から生れる音階
(
```

[1] これはグラニューラ・シンセシスの一種のだとも考えられる。

```
var base = 11025; // 基本周波数
var farray = (Array.series(8, 1, 1).reciprocal * .x Array.geom(8, 1, 3/4)).postln;
Pdef(\y,
    Pbind(
        \instrument, \line,
        \amp, 0.1,
        \freq, Pseq(farray * base, inf),
        \sustain, 0.001,
        \dur, 0.001,
        \pan, 0.0)
    )
)
// 倍音列と四分音程列 (24 音平均律音階) から生れる音階
(
var base = 30; // 基本周波数
var farray = (Array.series(8, 1, 1) * .x Array.geom(24, 1, 2**(1/24)).postln).postln;
Pdef(\y,
    Pbind(
        \instrument, \line,
        \amp, 0.1,
        \freq, Pseq(farray * base, inf),
        \sustain, 0.001,
        \dur, 0.001,
        \pan, 0.0)
    )
)
```

課題 6.4：音階による響き

周波数マトリクスをスキャンする速度や順序 (マトリクスをソートしてみよ) を変えることで、響きや音色がどのように変化するかを探求せよ。

第7章　色彩(color)

7.1　色と音

7.1.1　人間の目に見える光のことを可視光といい、その波長は約 400 ナノメートル（4×10^{-7} m、1 ミリの 1 万分の 4）から 700 ナノメートル（7×10^{-7} m）の範囲である。

7.1.2　400 ナノメートル近くの短い波長の光（単色光）は紫色に見え、 700 ナノメートル近くの長い波長の光は赤く見える。その間を埋めるのが、赤、橙、黄、緑、青、藍、紫という、いわゆる虹の七色である。

7.1.3　人間の可聴域の周波数と可視光の周波数を 1 対 1 に対応させることで、音と色を一意に結びつけることができる。

7.1.4　なお、眼には「三色性」という重要な性質がある。どのような色でも、たった 3 つの色でつくりだすことができる。例えば、RGB(赤緑青) の 3 色の光の強さを調節して加えることで、さまざまな色が表現できる。

7.1.5　光の波長と RGB の比率を結びつけたものが Color Matching Function である[1]。

コード 7.1

```
// 音の周波数と色を結びつけてみる
(
var w;
w = SCWindow("RGB", Rect(100, 100, 300, 300));
w.view.background = Color.white;
w.onClose = { Tdef(\x).stop; };
w.front;
Tdef(\x, {
    var idata, rdata, gdata, bdata, gamma, freq, hasFreq, in, amp, wl, i, r, g, b, f0, l0;
    gamma = 2.2;
    idata = [390, 395, 400, 405, 410, 415, 420, 425, 430, 435, 440, 445, 450, 455, 460, 465, 470,
    475, 480, 485, 490, 495, 500, 505, 510, 515, 520, 525, 530, 535, 540, 545, 550, 555, 560,
    565, 570, 575, 580, 585, 590, 595, 600, 605, 610, 615, 620, 625, 630, 635, 640, 645, 650,
    655, 660, 665, 670, 675, 680, 685, 690, 695, 700, 705, 710, 715, 720, 725, 730, 735, 740,
    745, 750, 755, 760, 765, 770, 775, 780, 785, 790, 795, 800, 805, 810, 815, 820, 825, 830];
    rdata = [1.5000E-03, 3.8000E-03, 8.9000E-03, 1.8800E-02, 3.5000E-02, 5.3100E-02, 7.0200E-02,
    7.6300E-02, 7.4500E-02, 5.6100E-02, 3.2300E-02, 4.4000E-03, 4.7800E-02, 9.7000E-02,
    1.5860E-01, 2.2350E-01, 2.8480E-01, 3.3460E-01, 3.7760E-01, 4.1360E-01, 4.3170E-01,
    4.4520E-01, 4.3500E-01, 4.1400E-01, 3.6730E-01, 2.8450E-01, 1.8550E-01, 4.3500E-02,
    1.2700E-01, 3.1290E-01, 5.3620E-01, 7.7220E-01, 1.0059E+00, 1.2710E+00, 1.5574E+00,
    1.8465E+00, 2.1511E+00, 2.4250E+00, 2.6574E+00, 2.9151E+00, 3.0779E+00, 3.1613E+00,
    3.1673E+00, 3.1048E+00, 2.9462E+00, 2.7194E+00, 2.4526E+00, 2.1700E+00, 1.8358E+00,
    1.5179E+00, 1.2428E+00, 1.0070E+00, 7.8270E-01, 5.9340E-01, 4.4420E-01, 3.2830E-01,
    2.3940E-01, 1.7220E-01, 1.2210E-01, 8.5300E-02, 5.8600E-02, 4.0800E-02, 2.8400E-02,
    1.9700E-02, 1.3500E-02, 9.2400E-03, 6.3800E-03, 4.4100E-03, 3.0700E-03, 2.1400E-03,
    1.4900E-03, 1.0500E-03, 7.3900E-04, 5.2300E-04, 3.7200E-04, 2.6500E-04, 1.9000E-04,
```

[1] http://cvision.ucsd.edu/cmfs.htm

```
    1.3600E-04, 9.8400E-05, 7.1300E-05, 5.1800E-05, 3.7700E-05, 2.7600E-05, 2.0300E-05,
    1.4900E-05, 1.1000E-05, 8.1800E-06, 6.0900E-06, 4.5500E-06];
    gdata = [-4.0000E-04, -1.0000E-03, -2.5000E-03, -5.9000E-03, -1.1900E-02, -2.0100E-02,
    -2.8900E-02, -3.3800E-02, -3.4900E-02, -2.7600E-02, -1.6900E-02,  2.4000E-03,  2.8300E-02,
    6.3600E-02, 1.0820E-01, 1.6170E-01, 2.2010E-01,  2.7960E-01,  3.4280E-01,  4.0860E-01,
    4.7160E-01,  5.4910E-01,  6.2600E-01,  7.0970E-01,  7.9350E-01,  8.7150E-01,  9.4770E-01,
    9.9450E-01, 1.0203E+00, 1.0375E+00, 1.0517E+00, 1.0390E+00, 1.0029E+00, 9.6980E-01,
    9.1620E-01, 8.5710E-01, 7.8230E-01, 6.9530E-01, 5.9660E-01, 5.0630E-01, 4.2030E-01,
    3.3600E-01, 2.5910E-01, 1.9170E-01, 1.3670E-01, 9.3800E-02, 6.1100E-02, 3.7100E-02,
    2.1500E-02, 1.1200E-02, 4.4000E-03, 7.8000E-05, -1.3680E-03, -1.9880E-03, -2.1680E-03,
    -2.0060E-03, -1.6420E-03, -1.2720E-03, -9.4700E-04, -6.8300E-04, -4.7800E-04, -3.3700E-04,
    -2.3500E-04, -1.6300E-04, -1.1100E-04, -7.4800E-05, -5.0800E-05, -3.4400E-05, -2.3400E-05,
    -1.5900E-05, -1.0700E-05, -7.2300E-06, -4.8700E-06, -3.2900E-06, -2.2200E-06, -1.5000E-06,
    -1.0200E-06, -6.8800E-07, -4.6500E-07, -3.1200E-07, -2.0800E-07, -1.3700E-07, -8.8000E-08,
    -5.5300E-08, -3.3600E-08, -1.9600E-08, -1.0900E-08, -5.7000E-09, -2.7700E-09];
    bdata = [6.2000E-03, 1.6100E-02, 4.0000E-02, 9.0600E-02, 1.8020E-01, 3.0880E-01, 4.6700E-01,
    6.1520E-01, 7.6380E-01, 8.7780E-01, 9.7550E-01, 1.0019E+00, 9.9960E-01, 9.1390E-01,
    8.2970E-01, 7.4170E-01, 6.1340E-01, 4.7200E-01, 3.4950E-01, 2.5640E-01, 1.8190E-01,
    1.3070E-01, 9.1000E-02, 5.8000E-02, 3.5700E-02, 2.0000E-02, 9.5000E-03, 7.0000E-04,
    4.3000E-03, 6.4000E-03, 8.2000E-03, 9.4000E-03, 9.7000E-03, 9.7000E-03, 9.3000E-03,
    8.7000E-03, 8.0000E-03, 7.3000E-03, 6.3000E-03, 5.3700E-03, 4.4500E-03, 3.5700E-03,
    2.7700E-03, 2.0800E-03, 1.5000E-03, 1.0300E-03, 6.8000E-04, 4.4200E-04, 2.7200E-04,
    1.4100E-04, 5.4900E-05, 2.2000E-06, 2.3700E-05, 2.8600E-05, 2.6100E-05, 2.2500E-05,
    1.8200E-05, 1.3900E-05, 1.0300E-05, 7.3800E-06, 5.2200E-06, 3.6700E-06, 2.5600E-06,
    1.7600E-06, 1.2000E-06, 8.1700E-07, 5.5500E-07, 3.7500E-07, 2.5400E-07, 1.7100E-07,
    1.1600E-07, 7.8500E-08, 5.3100E-08, 3.6000E-08, 2.4400E-08, 1.6500E-08, 1.1200E-08,
    7.5300E-09, 5.0700E-09, 3.4000E-09, 2.2700E-09, 1.5000E-09, 9.8600E-10, 6.3900E-10,
    4.0700E-10, 2.5300E-10, 1.5200E-10, 8.6400E-11, 4.4200E-11];
    freq = 20;
    w.drawHook = {
        Pen.use {
        Color.fromArray([r, g, b]).set;
        Pen.fillRect(Rect(0, 0, 300, 300));
        freq.asString.drawAtPoint(10@5, Font("Monaco", 60), Color.white);
        };
    };
    loop({
        freq = freq + 10;
        if( freq > 20000, {freq = 20} );
        wl = -156.66 * freq.max(0).log10 + 1033.648;
        i = idata.indexInBetween(wl);
        r = rdata.blendAt(i).max(0)**(1/gamma);
        g = gdata.blendAt(i).max(0)**(1/gamma);
        b = bdata.blendAt(i).max(0)**(1/gamma);
        w.refresh;
        0.01.wait;
        });
})
)
// Task の実行と停止
Tdef(\x).play(AppClock);
Tdef(\x).stop;
```

7.1.1 ノイズと色

7.1.1.1 音響素材としての面、すなわちホワイトノイズは、すべての周波数を均等に含んだ線の集合体である。

7.1.1.2 ホワイトノイズはランダムに振幅が変化する連続した点群でもある。

7.1.1.3 ホワイトノイズ (白色雑音) という名称は、すべての波長の可視光を混ぜると白色になる、という音と色のアナロジーに由来する。あらゆる周波数成分を同等に含むため、パワースペクトルは全周波数帯で (ほぼ) 一定になる。

コード 7.2

```
// ホワイトノイズを鳴らし波形をプロットする
~out = { WhiteNoise.ar(0.5.dup, 0) }.plot;
```

7.1.1.4 ホワイトノイズ (白) とサイレンス (黒) の間に、さまざまな色のノイズが存在する。

ピンクノイズ

7.1.1.5 周波数に反比例して周波数成分が高くなるほどパワー・スペクトルが小さくなっていくノイズをピンクノイズ (または $\frac{1}{f}$ ノイズ) と呼ぶ。

7.1.1.6 ピンクノイズという呼び名は、周波数が低い (波長が長い) 音を波長が長い赤い光になぞらえて、白色より赤みがかっているノイズというアナロジーからつけられた。

コード 7.3

```
// ピンクノイズを鳴らし波形をプロットする
~out = { PinkNoise.ar(0.5.dup, 0) }.plot;
```

ブラウンノイズ

7.1.1.7 パワースペクトルが周波数の 2 乗に反比例するノイズをブラウンノイズ ($\frac{1}{f^2}$ ノイズ) と呼ぶ。

7.1.1.8 ブラウン・ノイズの波形はブラウン運動 (ランダムウォーク) から得られる。

7.1.1.9 ブラウン・ノイズの音色はピンクノイズさらに低音が強調されたこもった質感になる。

コード 7.4

```
// ブラウンノイズを鳴らし波形をプロットする
~out = { BrownNoise.ar(0.5.dup, 0) }.plot;
```

ブルーノイズ

7.1.1.10 ピンクノイズの逆に、周波数に比例して周波数成分が高くなるほどパワー・スペクトルが大きくなるノイズをブルーノイズ (または f ノイズ) と呼ぶ。

パープルノイズ

7.1.1.11 パワー・スペクトルが周波数の 2 乗に比例するノイズをパープルノイズ (または f^2 ノイズ) と呼ぶ。

7.2 ノイズ彫刻

7.2.1 こうした、さまざまな色のノイズは、光の色彩同様、すべての周波数成分を含むホワイトノイズにフィルターをかけることによって得ることができる。

第8章　式(formula)

第9章　変調(modulation)

第10章　型(pattern)

第11章　流動(flux)

参考文献

[1] ボブ・スナイダー『音楽と記憶：認知心理学と情報理論からのアプローチ』音楽之友社，2003.

[2] 大蔵康義『音と音楽の基礎知識』国書刊行会，1999.

[3] カールハインツ・シュトックハウゼン『電子音楽の四つの特徴』ユリイカ 3 月号，青土社，1998.

[4] J.R. ピアース『音楽の科学：クラシックからコンピュータ音楽まで』村上陽一郎訳，日経サイエンス社，1989.

[5] B.C.J. ムーア『聴覚心理学概論』大串健吾監訳，誠信書房，1994.

[6] W. カンディンスキー『点と線から面へ (バウハウス叢書 9)』宮島久雄訳，中央公論美術出版，1995.

ライセンス

このテキストは、クリエイティブ・コモンズの帰属-同一条件許諾 2.1 Japan ライセンスの下でライセンスされています。この使用許諾条件を見るには、http://creativecommons.org/licenses/by-sa/2.1/jp/をチェックするか、クリエイティブコモンズに郵便にてお問い合わせください。住所は：559 Nathan Abbott Way, Stanford, California 94305, USA です。