構造

音響を表現するパラメータ間の相互の関係のことを構造と呼ぶ。コンポジションの一つの重要な側面はパラメータ間の関係、すなわち構造をデザインすることでもある。ランダムなパラメータの選択や関係に構造はない。「変奏のデザイン」で試みたように、数や数式を使ってパラメータ間の関係を設定することで、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ることができる。

時間軸上の配置

これまで定義してきたひとつひとつの音響旋律とその変奏を、時間軸上に配置(レイアウト)することを考える。そのために、あるひとつの旋律が鳴り始めるタイミングとしての時間パラメータを考える。

・音高がゆっくりとスライドする3本の線の組み合わせ

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


// Taskの開始と停止
x.play;
x.stop;


// 時間変化する線のアサイン
~out = \line2;


// 8710l[Hz]から8910[Hz]まで高さが変化する10秒の線を1秒ずつずらして3本重ねる
(
x.source = {
    loop {
        ~out.spawn([\amp1, 0.1, \amp2, 0.1, \freq1, 8710, \freq2, 8910, \pan1, -1, \pan2, 1, \sustain,
10]);
        1.wait;
        ~out.spawn([\amp1, 0.1, \amp2, 0.1, \freq1, 8910, \freq2, 8710, \pan1, 1, \pan2, -1, \sustain,
10]);
        1.wait;
        ~out.spawn([\amp1, 0.1, \amp2, 0.1, \freq1, 8710, \freq2, 8910, \pan1, 0, \pan2, 0, \sustain,
10]);
        15.wait;
    };
}
)
```

・異なる周波数の4つの点列の組み合わせ

```
// 点列のアサイン
~out = \points;

// 2[Hz]、20[Hz]、200[Hz]、2000[Hz]の4つの点列を重ねる
(
x.source = {
    loop {
        ~out.spawn([\amp, 0.3, \freq, 2, \pan, 0, \sustain, 10]);
        2.wait;
        ~out.spawn([\amp, 0.2, \freq, 20, \pan, -1, \sustain, 6]);
        1.wait;
        ~out.spawn([\amp, 0.1, \freq, 200, \pan, 1, \sustain, 4]);
        1.wait;
        ~out.spawn([\amp, 0.05, \freq, 2000, \pan, 0, \sustain, 2]);
        10.wait;
    };
}
)
```

ストリームを使ったレイアウト

時間軸上の配置を表現するために「ストリーム」を用いる。ストリームは値を連続的に出力することができるオブジェクトで、nextメッセージを送ることでストリームの要素をひとつづつ出力する。このストリームで時間間隔のシーケンスを記述することで、時間軸上のレイアウトが表現できる。

```
// 出力に線をアサインする
~out = \line;

// Taskの開始と停止
x.play;
x.stop;

// 一定の時間間隔列で短い線を鳴らす
(
t = Pseq([0.2, 0.3, 0.4, 0.5], inf).asStream; // 時間間隔列をストリームにする
x.source = {
    loop {
        ~out.spawn([\amp, 0.2, \freq, 8820, \pan, 0, \sustain, 0.1]);
        t.next.postln.wait; // ストリームを順に読み出す
    };
}
)
```

緊張と弛緩–いかにして構造をつくりだすか?

音響旋律とその変奏の方法を述べた際に、「線の思考」という考え方を導入した。ここではさらに、この線の思考という水平構造のデザインを行うための方法として、「緊張(テンション/活性化)」と「弛緩(リラクゼーション/鎮静化)」という2つの対概念を導入する。緊張とは音響全体を不安定な状態におくこと、弛緩とは逆に安定な状態におくことである。

緊張(テンション：T)と弛緩(リラクゼーション：R)と音響(パラメータ)の基本的関係

・音高(周波数)

緊張：弛緩 ＝ 高音/低音：中音、上昇：下降

・音程

緊張：弛緩 ＝ 遠：近、分散：集中

・音価(音長)

緊張：弛緩 ＝ 短音：長音

・エンヴェロープ

緊張：弛緩 ＝ 減衰音：持続音

・音量(振幅)

緊張：弛緩 ＝ 大/小：中、変化：一定

・テンポ

緊張：弛緩 ＝ 速い：遅い、加速：減速

・リズム

緊張：弛緩 ＝ 不規則：規則

・音色

緊張：弛緩 ＝ 厚：薄、密：疎、多様：統一、乱雑(ノイズ)：秩序(楽音)

・構成

緊張：弛緩 ＝ 変化：オスティナート、発展：固定

緊張だけでも、弛緩だけでも音楽は単調になる。同様に過度の変化が逆に単調さを生み出すこともある。繰り返しや持続を効果的に利用することも必要である。相互に補間しあう緊張と弛緩の推移やバランスが、時間軸上でのプロセスとして聴覚的に認知されることで、音響的な文脈やダイナミズムが生まれる。

緊張Tと弛緩Rの組み合わせ方には以下の2つがある。弛緩から緊張への推移を「不安定化/クライマックス」、逆に緊張から弛緩への推移を「安定化/アンチクライマックス」という。

・二部形式：R→T（クライマックス）、T→R（アンチクライマックス）

・三部形式（囲い込みの構造）：R→T→R、T→R→T

この2つの基本構造を組み合せて拡張していくこともできる。例えば

・R→T→R→T

・T→R→T→T→R

この緊張と弛緩の連鎖の中に、「並列」と「対比」というもうひとつの基本構造を見ることもできる。並列とは同じ要素が反復していく構造(「T→T」あるいは「R→R」)のこと、対比とは異なる要素に変化していく構造(「T→R」あるいは「T→R」)のことである。

・同種エレメントの連鎖＝並列＝「T→T」あるいは「R→R」＝対称

・異種エレメントの連鎖＝対比(コントラスト)＝「T→R」あるいは「R→T」＝非対称

こうした観点から最初の2つの例の構造を分析する。「音高がゆっくりとスライドする3本の線の重ね合わせ」では高音による緊張と集中した音程による弛緩の共時的なバランスが重要である。一方「異なる周波数の4つの点列の重ね合わせ」の場合、音が次第に重なっていくことによる弛緩から緊張へのクライマックスと、逆に音薄くなっていくことによる緊張から弛緩へのアンチクライマックスの経時的な連鎖が重要である。

▼エクササイズ：音響構造の生成と分析

以下のような音響旋律の組み合わせをインプリメントし、それがどのような緊張：弛緩、クライマックス：アンチクライマックスの構造をもっているかを分析せよ。

・高音、低音のそれぞれの線、点、面の4つの要素を使った組み合わせ

・100[Hz]以下の音量が急激に減少する低い線のみを使った組み合わせ

・1秒以下の短い高い線を多数使った組み合わせ

・1秒以下の4つの短い面と10秒以上の長さの線の組み合わせ

・長い時間に渡ってゆっくりとパラメータが変化する600秒の音の組み合わせ

・急激にパラメータが変化する3秒の音の組み合わせ

・ゆっくりとパラメータが変化する音と急激にパラメータが変化する音の組み合わせ

対型の導入

前章で行なった4つの変奏に、原型に対する対比的要素としての「対型（counter）」を導入する。原型Aと対型Bという2つの対比的な要素を組み合わせることで、時間的持続の中で緊張と弛緩が回帰する2元的な構造をデザインする。

A:B

・原型→対型→対型の変奏1→対型の変奏2→対型の変奏3

・原型→原型の変奏1→対型→対型の変奏1→対型の変奏2

・原型→原型の変奏1→原型の変奏2→対型→対型の変奏1

A:B:A

・原型→対型→原型の変奏1→原型の変奏2→原型の変奏3

・原型→対型→対型の変奏1→対型の変奏2→原型の変奏1

・原型→原型の変奏1→対型→対型の変奏1→原型の変奏2

・原型→原型の変奏1→原型の変奏2→対型→原型の変奏3

A:B:A:B

・原型→原型の変奏1→対型→原型の変奏2→対型の変奏1

・原型→対型→対型の変奏1→原型の変奏1→対型の変奏2

・原型→対型→原型の変奏1→原型の変奏2→対型の変奏1

・原型→対型→原型の変奏1→対型の変奏1→対型の変奏2

1つの例として、点列による原型と面による対型を用いた例を示す。

・原型→対型→対型の変奏1→原型の変奏1→原型の変奏2

```
// 原型と対型のプロキシの定義
~out = ~org + ~ctr;

// 点列による原型
(
~org = { arg base=11050, fls=1, flb=0, als=0.2, alb=0, ts=1;
    var farray, aarray, tarray, envf, enva;
    farray = Array.rand(11, 0.9, 1.1); // ランダムな周波数列
    aarray = Array.fill(11, {|i| i % 2}); // ジグザクに振動する振幅列
    tarray = Array.fill(10, 0.3); // 一定の継続時間列
    envf = Env.new(farray, tarray, \step); // 周波数エンヴェロープの生成
    enva = Env.new(aarray, tarray); // 振幅エンヴェロープの生成
    Impulse.ar(base * EnvGen.kr(envf, 1, fls, flb, ts, 2), 0, EnvGen.kr(enva, 1, als, alb, ts, 2)).dup }
)

// 面による対型
(
~ctr = { arg als=1, alb=0, ts=1;
    var aarray, tarray, enva;
    aarray = Array.series(11, 0.0, 0.01.rand); // 増加する振幅列
    tarray = Array.fill(10, 0.01); // 一定の継続時間列
    enva = Env.new(aarray, tarray, \step); // 振幅エンヴェロープの生成
    WhiteNoise.ar(EnvGen.kr(enva, 1, als, alb, ts, 2)).dup }
)



// TaskProxyの定義
x = TaskProxy.basicNew;

// TaskProxyの開始と終了
x.play;
x.stop;


// 原型→対型→(ランダムにパラメータを変化させた){原型，対型} の繰り返しによる変奏
(
var fls, alb, ts;
x.source = {
    loop {
        ~org.rebuild;
        ~org.spawn;
        2.0.rand.wait; // ユニットをオーバーラップさせる
        ~ctr.rebuild;
        ~ctr.spawn;
        2.0.rand.wait;
        if( 0.5.coin, {
            ~org.rebuild;
            fls = rrand(0.5, 1.5);
            ts = rrand(0.5, 1.5);
            ~org.spawn([\fls, fls, \ts, ts]);
        },{
            ~ctr.rebuild;
            alb = rrand(-0.5, 0.0);
            ts = rrand(0.1, 1.0);
            ~ctr.spawn([\alb, alb, \ts, ts]);
        });
        2.0.rand.wait;
    }
}
)
```

▼エクササイズ：原型と対型を用いた変奏

原型と対型の2つの旋律とその変奏を用いた15〜30秒の旋律をデザインせよ。

比による構造–定性的な構造から定量的な構造へ

緊張と弛緩、クライマックスとアンチクライマックス、といった構造はパラメータ間の関係を定性的に表現したものだが、パラメータの値を具体的に決定するためには、関係を定量的に表現する必要がある。

パラメータ間の関係を定量的に表現するのが「比(比例、比率)」である。例えば、音の高さの関係である音程は比によって表現されている。「よく響きあう」オクターブや5度の協和音程は、それぞれ1：2、2：3という周波数の比で表現することができる。

相対的構造と絶対的構造

比によって表現される音程は相対的な構造である。100[Hz]と150[Hz]の音も、80[Hz]と120[Hz]の音も、2:3の周波数比–5度という等しい音程関係を持っている。しかしながら、両者を構成する音の周波数(音の高さ)は、100[Hz]、150[Hz]、80[Hz]、120[Hz]という具体的な値を有している。100[Hz]と150[Hz]の音と、80[Hz]と120[Hz]の音は、両方とも5度の等しい関係を持っているが、100[Hz]の音と80[Hz]の音の響きは異なっているし、その音程差(うなり)の周波数も同様である。こうした値そのもの(パラメータの絶対性)によって構成される構造を、比よる相対的構造に対して「絶対的構造」という。パラメータによって音響の構造をデザインする際には、相対的構造と絶対的構造の双方を考慮する必要がある。

▼エクササイズ：「1:2」の比による旋律と変奏

「1:2」という最もシンプルだが重要な比(相対的構造)を用いて、旋律(原型)とその変奏をデザインせよ。

```
// 「1:2」の比による旋律
(
~out = { arg base=30, fls=1, flb=0, als=0.2, alb=0, ts=0.1;
    var farray, aarray, tarray, envf, enva;
    farray = Array.fill(11, {[1, 2].choose}); // 1と2のランダムな組み合わせによる周波数列
    aarray = Array.fill(11, {[1, 2].choose}); // 1と2のランダムな組み合わせによる振幅列
    tarray = Array.fill(10, {[1, 2].choose}); // 1と2のランダムな組み合わせによる継続時間列
    envf = Env.new(farray, tarray, \step); // 周波数エンヴェロープの生成
    enva = Env.new(aarray, tarray, \step); // 振幅エンヴェロープの生成
    SinOsc.ar(base * [1, 2].choose * EnvGen.kr(envf, 1, fls, flb, ts, 2), 0, EnvGen.kr(enva, 1, als, alb,
ts, 2)).dup }
)

// 変奏の{1秒，2秒}(1:2)間隔での繰り返し
x.source = { loop { ~out.rebuild; ~out.spawn; [1, 2].choose.wait; }}
x.play;
x.stop;
```

「1:3」「1:4」「1:5」などの比を用いて、同様に旋律や変奏をデザインせよ。

▼エクササイズ：数列を使った旋律と変奏

「変奏のデザイン」で用いた数列(漸化式)は構造と密接な関係がある。数と数の関係が相対比で既定される等比数列は相対的構造を、絶対

値で既定される等差数列は絶対的構造をつくりだす。

数列を用いた旋律とその変奏をデザインせよ。

```
// 数列による旋律
(
~out = { arg base=11025, fls=1, flb=0, als=0.1, alb=0, ts=0.1;
    var farray, aarray, tarray, envf, enva;
    farray = Array.geom(11, 1.0, rrand(0.9, 1.1)); // ランダムな公比の等比数列による周波数列
    aarray = Array.geom(11, 1.0, rrand(-0.9, -1.1)); // ランダムな負の公比の等比数列による振幅列
    tarray = Array.series(10, 1.0, rrand(-0.1, 0.1)); // ランダムな公差の等差数列による継続時間列
    envf = Env.new(farray, tarray); // 周波数エンヴェロープの生成
    enva = Env.new(aarray, tarray); // 振幅エンヴェロープの生成
    SinOsc.ar(base * EnvGen.kr(envf, 1, fls, flb, ts, 2), 0, EnvGen.kr(enva, 1, als, alb, ts, 2)).dup }
)

// 変奏の1秒間隔での繰り返し
x.source = { loop { ~out.rebuild; ~out.spawn; 1.wait; }}
x.play;
x.stop;
```

黄金比とフィボナッチ数列

黄金分割とは、ある線分ABを AB:AC = AC:BC となるように分割することである。このとき，AB/AC のことを黄金比と呼ぶ。2次方程式の解の計算から，AB/AC = 1:(1+√5)/2 ≒ 1:1.6180339887499... と求められる。

```
1 + sqrt(5) / 2;

> 1.6180339887499
```

古くから黄金比と人間の美や快楽との結びつきが指摘されており、自然界や絵画の構図や建築のなかによくこの比が現れる。音楽作品の中にも、しばしこの黄金比が用いられてきた。

一方、フィボナッチ数列は、

```
a[0] = 1
a[1] = 1
a[n] = a[n-1] + a[n-2] (n>1)
```

で定義される数列である。フィボナッチ数列と黄金比は密接に関係している。フィボナッチ数列の相隣り合う項の比からなる数列 a[n+1]/a[n] は、1/1, 2/1, 3/2, 5/3, 8/5 ... とnが大きくなるにつれて黄金比に収束していく。

```
// フィボナッチ数列の定義
f = { |i| if ( ((i == 1) || (i == 2)), { 1 }, { f.value(i-1) + f.value(i-2) } )} ;

// フィボナッチ数列を第20項目まで出力する
20.do({|i| (i+1).post; " : ".post; f.value(i+1).postln;});

1:1
2:1
3:2
4:3
5:5
6:8
7:13
8:21
9:34
10:55
11:89
12:144
13:233
14:377
15:610
16:987
17:1597
18:2584
19:4181
20:6765
```

```
// フィボナッチ数列の隣りあう項の比を第20項目まで出力する
20.do({|i|  f.value(i+2).post; ":".post; f.value(i+1).post; " = ".post; (f.value(i+2)/f.value(i
+1)).postln;});

1:1 = 1
2:1 = 2
3:2 = 1.5
5:3 = 1.6666666666667
8:5 = 1.6
13:8 = 1.625
21:13 = 1.6153846153846
34:21 = 1.6190476190476
55:34 = 1.6176470588235
89:55 = 1.6181818181818
144:89 = 1.6179775280899
233:144 = 1.6180555555556
377:233 = 1.618025751073
610:377 = 1.6180371352785
987:610 = 1.6180327868852
1597:987 = 1.6180344478217
2584:1597 = 1.6180338134001
4181:2584 = 1.6180340557276
6765:4181 = 1.6180339631667
10946:6765 = 1.6180339985218
```

黄金比とフィボナッチ数列から得られるさまざまな構造を聴く

```
// 周波数用プロキシ
~freq.kr(1);

// 黄金比の高さの関係にある2つの線
~out = { (SinOsc.ar(~freq.kr, 0, 0.2) + SinOsc.ar(~freq.kr * (1 + sqrt(5) / 2), 0, 0.2)).dup };

// 線の高さにマウスのX座標を対数マッピングする
~freq = { MouseX.kr(20.0, 11050, 1) };

// Taskの開始と停止
x.play;
x.stop;

// 線のアサイン
~out = \line;


// フィボナッチ数列による音程の線の重ねあわせ
(
var fibo = Array.fill(16, {|i| f.value(i+1)}).postln;
x.source = {
    16.do ({ |i|
        i.post; ":".post;
        ~out.spawn([\amp, 0.05, \freq, (20 * fibo.at(i)).postln, \pan, 1.0.rand2, \sustain, 20 - i]);
        1.wait;
    });
}
)


// 黄金比の長さと高さ関係にある2つの線の反復
(
var fbase, tbase, gr = 1 + sqrt(5) / 2;
x.source = {
    loop {
        fbase = 4410.0.rand;
        tbase = 3.0.rand;
        ~out.spawn([\amp, 0.2, \freq, fbase, \pan, 0, \sustain, tbase]);
        tbase.wait;
        ~out.spawn([\amp, 0.2, \freq, fbase * gr, \pan, 1.0.rand2, \sustain, tbase * gr]);
        (tbase * gr).wait;
    };
}
)


// 黄金比による旋律の反復
(
~out = { arg base=882, fls=1, flb=0, als=0.1, alb=0, ts=0.1;
    var fibo, farray, aarray, tarray, envf, enva, gr = 1 + sqrt(5) / 2;
```

```
    farray = Array.fill(11, {[1, gr].choose}); // 1と2のランダムな組み合わせによる周波数列
    aarray = Array.fill(11, {[1, gr].choose}); // 1と2のランダムな組み合わせによる振幅列
    tarray = Array.fill(10, {[1, gr].choose}); // 1と2のランダムな組み合わせによる継続時間列
    envf = Env.new(farray, tarray, \step); // 周波数エンヴェロープの生成
    enva = Env.new(aarray, tarray, \step); // 振幅エンヴェロープの生成
    SinOsc.ar(base * [1, gr].choose * EnvGen.kr(envf, 1, fls, flb, ts, 2), 0, EnvGen.kr(enva, 1, als, alb,
ts, 2)).dup }
)
// 変奏の{1秒，2秒}(1:2)間隔での繰り返し
x.source = { loop { ~out.rebuild; ~out.spawn; [1, (1 + sqrt(5) / 2)].choose.wait; }}
x.play;
x.stop;
```

▼エクササイズ：黄金比とフィボナッチ数列を使った旋律と変奏

黄金比を用いて、旋律(原型)とその変奏をデザインせよ。

2と3による構成

フィボナッチ数列は、以下のように2と3を基本単位として分解できる。

```
 5 = 2 + 3 = 2×1 + 3×1
 8 = 2 + 3 + 3 = 2×1 + 3×2
13 = 2 + 2 + 3 + 3 + 3 = 2×2 + 3×3
21 = 2 + 2 + 2 + 3 + 3 + 3 + 3 + 3 = 2×3 + 3×5
34 = 2 + 2 + 2 + 2 + 2 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 = 2×5 + 3×8
```

2と3それぞれの個数がふたたびフィボナッチ数列になる。

項の加算から生成されるフィボナッチ数列は、それ自体がモジュール構造を内包している。2と3を音程や持続時間といった音響のミクロ構造に対応させることも、2と3を緊張と弛緩、2部形式と3部形式、非対称と対称といったマクロ構造と結びつけることも可能である。

n等分とn乗根

ある幅が与えられた時、それを等差数列的に等しく分割するのが「n等分」で、等比数列的に等しく分割するのが「n乗根」である。前者は主に時間構造(拍節構造)に用いられ、後者は主に音程構造に用いられている。

```
// 出力に線をアサインする
~out = \line;

// Taskの開始と停止
x.play;
x.stop;
```

```
// 時間=等差数列、音高=等比数列による線の反復
(
var n = 5;
var time = Pseq(Array.series(n+1, 0, n.reciprocal).postln, inf).asStream;
var freq = Pseq(Array.geom(n+1, 1.0, 2 ** n.reciprocal).postln, inf).asStream;
x.source = {
    loop {
        ~out.spawn([\amp, 0.2, \freq, 441 * freq.next, \pan, 0, \sustain, 0.1]);
        time.next.wait;
    };
}
)

// 時間=等比数列、音高=等差数列による線の反復
(
var n = 5;
var time = Pseq(Array.geom(n+1, 1.0, 2 ** n.reciprocal).postln, inf).asStream;
var freq = Pseq(Array.series(n+1, 1.0, n.reciprocal).postln, inf).asStream;
x.source = {
```

```
    loop {
        ~out.spawn([\amp, 0.2, \freq, 441 * freq.next, \pan, 0, \sustain, 0.1]);
        (time.next - 1).wait;
    };
}
)

// n等分(等差数列)とn乗根(等比数列)による23音音階の混合
(
var n = 23;
var freq1 = Pseq(Array.series(n+1, 1.0, n.reciprocal).postln, inf).asStream;
var freq2 = Pseq(Array.geom(n+1, 1.0, 2 ** n.reciprocal).postln, inf).asStream;
x.source = {
    loop {
        ~out.spawn([\amp, 0.2, \freq, 441 * [freq1, freq2].choose.next, \pan, 0, \sustain, 0.1]);
        0.1.wait;
    };
}
)
```