スケール

音響における「スケール(scale)]について考える。スケールといっても、いわゆる音楽用語における音階のことではない。ここではスケールの文字通りの意味である「尺度/縮尺」や「規模/大きさ」について考える。

振動による抽象

音楽/音響現象は、数十ヘルツから数万ヘルツ、すなわち 0.0001秒から0.1秒といった、短い時間レンジのピッチに始まり、数ヘルツ程度のリズムや、その長期的な変動であるテンポ、さらに長い時間レンジの小節や形式、楽曲といったさまざまな周期の振動の集合体であるとみなすことができる。これはいいかえれば、ピッチやリズム、テンポや形式といったさまざまな音楽的諸概念を、広い時間レンジにわたる振動として統一的に抽象することである。

ここではエンヴェロープとモジュレーション(変調)、ディレイ(遅延)とフィルタを対象に、時間スケールと音色や響き(知覚)の関係を探求することで、振動による抽象によるスケールの統合(連続的な取り扱い)を試みる。

モジュレーション(変調)

音のパラメータの時間変化のことをエンヴェロープと呼んだ(音響旋律の項を参照)。その時間変化を次第に速くしていき、変化自体が音として可聴になると、どのように聴こえ方が変化するだろうか。

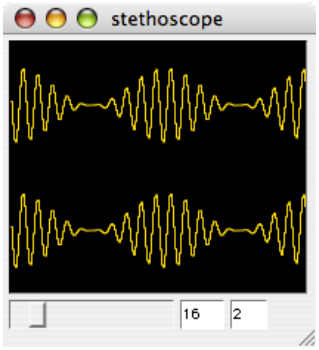
```
// コントロール(時間変化)用プロキシの定義
~ctl.ar(1);

// 4410[Hz]の線の振幅を変化させる
~out = { SinOsc.ar(4410.dup, 0, ~ctl.ar) };
```

まず最初に、線の振幅をエンヴェロープで変化させる。

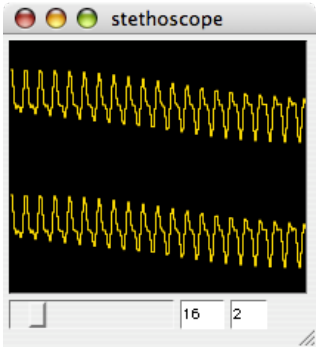
```
// ランダムなエンヴェロープ
~ctl.env(Env.new(Array.rand(8, 0.0, 0.6), Array.fill(7, 0.1)));

// サイン波形状のエンヴェロープ
~ctl.env(Env.sine(1, 0.6));

// エンヴェロープの速度を速くする
~ctl.env(Env.sine(0.1, 0.6));
```

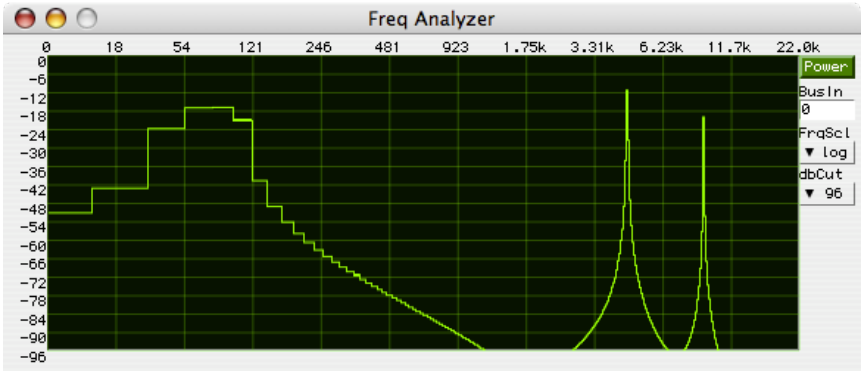
エンヴェロープの代わりに線(サイン波)で振幅を変化させる。

```
// 線の振幅を1Hzの線で変化させる
~ctl = { SinOsc.ar(1.0, 0, 0.3, 0.3) };

// 線の振幅を21Hzの線で変化させる
~ctl = { SinOsc.ar(21, 0, 0.3, 0.3) };

// 線の振幅を441Hzの線で変化させる
~ctl = { SinOsc.ar(441, 0, 0.3, 0.3) };
```

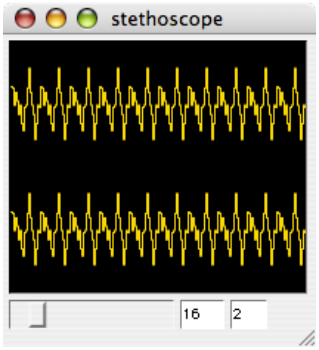



```
// 線の振幅を4410Hzの線で変化させる
~ctl = { SinOsc.ar(4410, 0, 0.3, 0.3) };

// 線の振幅を4510Hzの線で変化させる(4510-4410=100[Hz]の音が聴こえる)
~ctl = { SinOsc.ar(4510, 0, 0.3, 0.3) };
```






```
// 線の振幅を11025Hzの線で変化させる
~ctl = { SinOsc.ar(11025, 0, 0.3, 0.3) };
```



振幅変化の周波数が可聴域に達すると、エンヴェロープの変化そのものが音を一体化していき、元の音とエンヴェロープの区別がつかなくなっていく。こうした高速のエンヴェロープを「モジュレーション(変調、modulation)」と呼ぶ。いいかえれば、エンヴェロープとは、(可聴域以下の)低い周波数によるモジュレーションである。

インタラクティブ・リスニングの手法で、さまざまなエンヴェロープ～モジュレーション・サウンドを探求する。

エンヴェロープからモジュレーションへの遷移

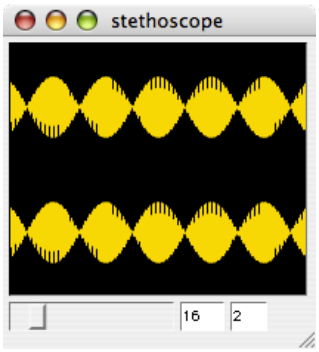
モジュレーション周波数を、非可聴域の低周波から可聴域までインタラクティヴに変化させ、エンヴェロープとモジュレーションの遷移領域を聴く。波形とスペクトルを見ながら音を聴くこと。

(注)一般に、非可聴域の低周波による浅い振幅変調のことを「トレモロ」、非可聴域の低周波による浅い周波数変調のことを「ビブラート」あるいは「ポルタメント」と呼んでいる。

振幅変調(AM)

```
// モジュレータ周波数用プロキシの定義
~mf.kr(1);

// 4410Hzの線の振幅を線でモジュレートする
~out = { SinOsc.ar(4410, 0, ~cnt.ar).dup };
```

```
~cnt = { SinOsc.ar(~mf.kr, 0, 0.5, 0) };

// マウスのX座標を0[Hz](一定)から100[Hz]のモジュレータ周波数に線形マッピングする
~mf = { MouseX.kr(0.0, 100.0, 0) };
```



周波数変調(FM)

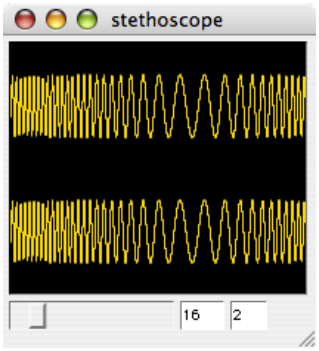
```
// 4410[Hz]の線の周波数をモジュレート(±441[Hz])する
~out = { SinOsc.ar(~cnt.ar, 0, 0.5).dup };
~cnt = { SinOsc.ar(~mf.kr, 0, 441, 4410) };
```

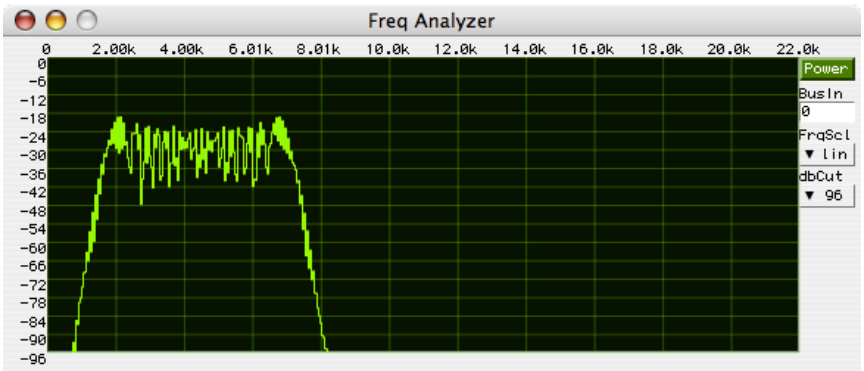
モジュレーションの振幅を大きくする。

```
// 4410[Hz]の線の周波数を線でモジュレート(±4410[Hz])する
~cnt = { SinOsc.ar(~mf.kr, 0, 4410, 4410) };
```

マウスのY座標を周波数のモジュレーション振幅にマッピングする。

```
// モジュレーション振幅用プロキシの定義
~ma.kr(1);

// マウスのY座標をモジュレータ振幅に線形マッピングする
~cnt = { SinOsc.ar(~mf.kr, 0, ~ma.kr, 4410) };
~ma = { MouseY.kr(0.0, 4410.0, 0) };
```



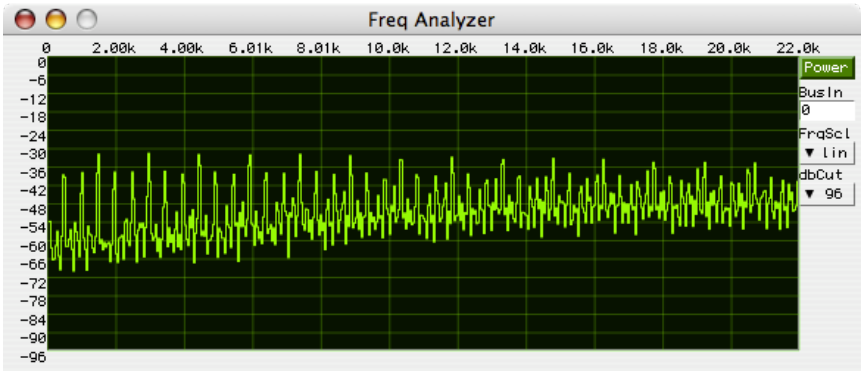


点列(と線)のモジュレーション

振幅変調(AM)

キャリア(変調されるもの)とモジュレータ(変調するもの)の周波数の比を周波数比rと呼ぶ。モジュレータ周波数mは（キャリア周波数）×（周波数比）で求められる。

```
// キャリア周波数用プロキシの定義
~cf.kr(1);

// 点列の振幅を線でモジュレートする
~out = { Impulse.ar(~cf.kr, 0, ~cnt.ar).dup };
~cnt = { SinOsc.ar(~mf.kr, 0, 0.4, 0.4) };

// マウスのX座標をキャリア周波数に対数マッピング
~cf = { MouseX.kr(20, 4410, 1) };

// マウスのY座標をモジュレータとキャリアの周波数比に線形マッピング
~mf = { ~cf.kr * MouseY.kr(0.0, 5.0, 0) };
```



基本的にスペクトルは、ホワイトノイズのように全帯域に渡ってほぼ水平だが、マウスのちょっとした動きで音色やスペクトルは不規則に変化する。

```
// キャリアを線におきかえる(線の振幅を線でモジュレートする = 通常の振幅変調)
~out = { SinOsc.ar(~cf.kr, 0, ~cnt.ar).dup };
```

一般の振幅変調では、r<1(キャリア周波数>モジュレーション周波数)だが、逆にr>1(モジュレーション周波数>>キャリア周波数)になるとどのように音は聴こえるだろうか?

周波数変調(FM)

モジュレータの振幅と周波数の比(振幅/周波数)を変調指数(モジュレーション・インデックス)iと呼ぶ。モジュレータ振幅は（モジュレータ周波数)x(変調指数）=（キャリア周波数)x(周波数比)x(変調指数）で求められる。

変調指数が一定であれば、キャリア周波数が変化した場合でも音色を一定に保つことができる。

```
// 変調指数用プロキシの定義
~i.kr(1);

// 点列の周波数を線でモジュレートする
~out = { Impulse.ar(~cnt.ar, 0, 0.5).dup };
~cnt = { SinOsc.ar(~mf.kr, 0, ~mf.kr * ~i.kr, ~cf.kr) };

// 変調指数 = 1
~i = 1;
```

```
// マウスのX座標をキャリア周波数に対数マッピング
~cf = { MouseX.kr(20, 4410, 1) };

// マウスのY座標をモジュレータとキャリアの周波数比に線形マッピング
~mf = { ~cf.kr * MouseY.kr(0.0, 5.0, 0) };

// 周波数比を2.3に固定する
~mf = ~cf * 2.3;

// マウスのY座標を変調指数に線形マッピング
~i = { MouseY.kr(0.0, 5.0, 0) };

// キャリアを線に変更する
~out = { SinOsc.ar(~cnt.ar, 0, 0.5).dup };

// キャリア周波数を可聴域の下限程度に設定する
~cf = { MouseX.kr(1, 50, 1) };

// 変調指数を非常に大きくするとどうなるか?
~i = { MouseY.kr(0.0, 50.0, 0) };

// キャリア周波数を20[Hz]に固定する
~cf = 20;

// マウスのX座標をモジュレータとキャリアの周波数比に線形マッピング
~mf = { ~cf.kr * MouseX.kr(0.0, 5.0, 0) };

// キャリア周波数極端に大きくする
~cf = 11025;

// キャリアを点列に戻す
~out = { Impulse.ar(~cnt.ar, 0, 0.5).dup };
```

振幅変調以上に、マウスの微妙に動きで音色やスペクトルは敏感に変化する。キャリア周波数、周波数比、変調指数の3つのパラメータのさまざまな組み合せを試してみよ。

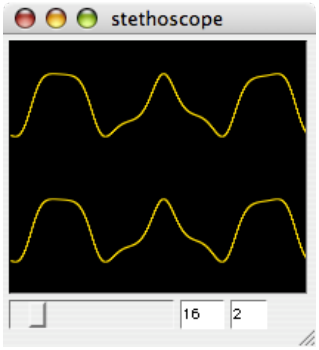
その他の変調(パン変調と位相変調)

振幅や周波数以外にも、すべてのパラメータをモジュレートすることが可能である。

```
// 点列のパンを線でモジュレートする(パン変調)
~out = { Pan2.ar(Impulse.ar(~cf.kr, 0, 0.5), ~cnt.ar) };
~cnt = { SinOsc.ar(~mf.kr, 0, 1) };

// マウスのX座標をキャリア周波数に対数マッピング
~cf = { MouseX.kr(20, 4410, 1) };

// マウスのY座標をモジュレータとキャリアの周波数比に線形マッピング
~mf = { ~cf.kr * MouseY.kr(0.0, 5.0, 0) };

// キャリアを線に変える
~out = { Pan2.ar(SinOsc.ar(~cf.kr, 0, 0.5), ~cnt.ar) };
```

パンのモジュレーションは、左右のチャンネルで位相のずれた振幅変調と同等である。

```
// 線の位相を線でモジュレートする(位相変調)
~out = { SinOsc.ar(~cf.kr, ~cnt.ar, 0.5).dup };
~cnt = { SinOsc.ar(~mf.kr, 0, pi) };
```

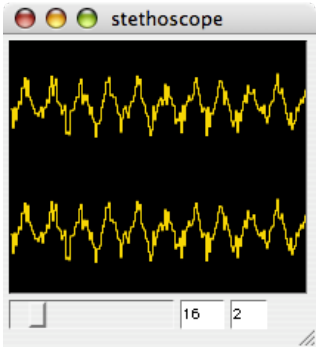


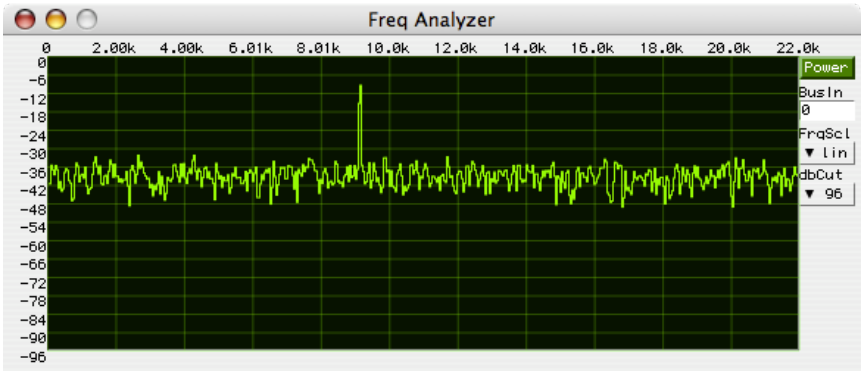
位相変調は波形そのものを歪めることに相当する。

面による変調

線ではなく面(ランダム)でモジュレートするとどのように聴こえるか?

```
// 線の振幅を面でモジュレートする
~out = { SinOsc.ar(~cf.kr, 0, ~cnt.ar).dup };
~cnt = { WhiteNoise.ar(~ma.kr, ~ma.kr) };

// マウスのX座標をキャリア周波数に対数マッピング
~cf = { MouseX.kr(20, 22050, 1) };

// マウスのY座標をモジュレータ振幅に線形マッピング
~ma = { MouseY.kr(0.0, 0.5, 0) };
```

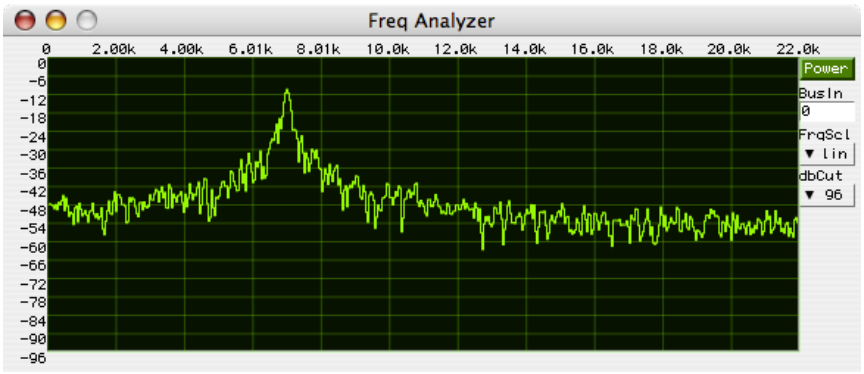




面+線のようなサウンドになる。

```
// 線の周波数を面でモジュレートする
~out = { SinOsc.ar(~cnt.ar, 0, 0.5).dup };
~cnt = { WhiteNoise.ar(~cf.kr * ~ma.kr, ~cf.kr) };

// マウスのY座標をモジュレータ振幅に線形マッピング
```

```
~ma = { MouseY.kr(0.0, 1.0, 0) };
```



振幅変調の場合に比べて、基本周波数のピークがゆるやかになる。

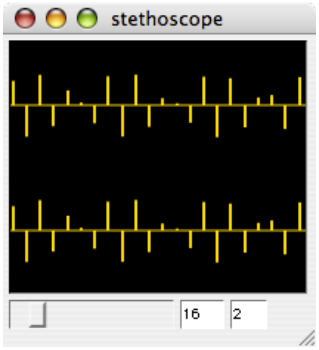
```
// 点列のパンを面でモジュレートする
~out = { Pan2.ar(Impulse.ar(~cf.kr, 0, 0.5), ~cnt.ar) };
~cnt = { WhiteNoise.ar(~ma.kr) };
```

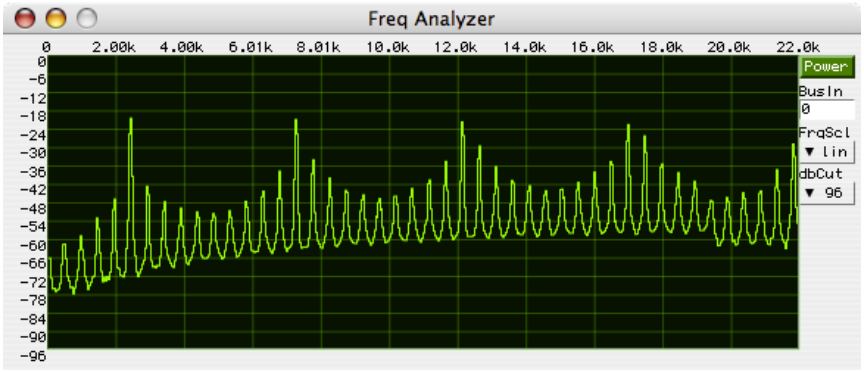
点列による変調

点列でモジュレートするとどのような音が生み出されるか?

```
// 線の振幅を点列でモジュレートする
~out = { SinOsc.ar(~cf.kr, 0, ~cnt.ar).dup };
~cnt = { Impulse.ar(~mf.kr, 0, 0.5, 0) };

// マウスのX座標をキャリア周波数に対数マッピング
~cf = { MouseX.kr(20, 4410, 1) };

// マウスのY座標をモジュレータとキャリアの周波数比に線形マッピング
~mf = { ~cf.kr * MouseY.kr(0.0, 5.0, 0) };
```

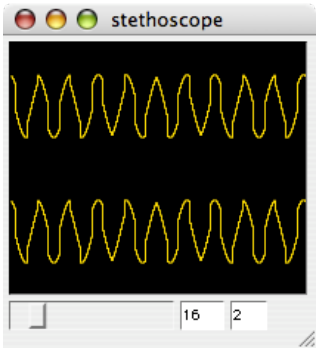
点列の振幅がサイン波上に変化した波形になる。

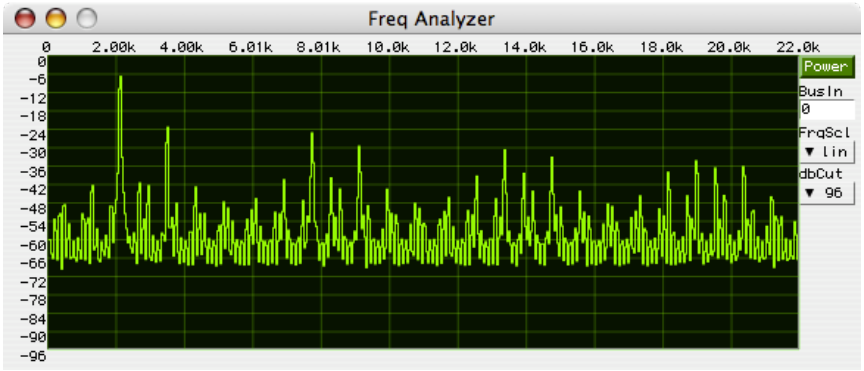





```
// 線の周波数を点列でモジュレートする
~out = { SinOsc.ar(~cnt.ar, 0, 0.5).dup };
~cnt = { Impulse.ar(~mf.kr, 0, ~mf.kr * ~i.kr, ~cf.kr) };

// 変調指数 = 1
~i = 1;

// マウスのX座標をキャリア周波数に対数マッピング
~cf = { MouseX.kr(20, 4410, 1) };

// マウスのY座標をモジュレータとキャリアの周波数比に線形マッピング
~mf = { ~cf.kr * MouseY.kr(0.0, 5.0, 0) };
```

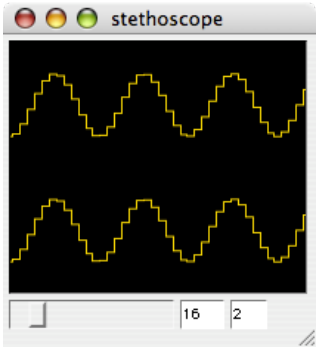
線の波形が歪む。

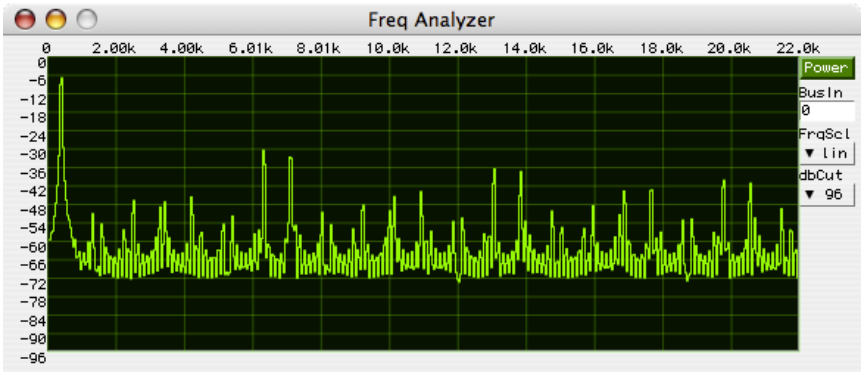





```
// キャリア周波数を0[Hz]にする
~cf = 0;

// マウスのX座標をモジュレータ周波数に対数マッピング
~mf = { MouseX.kr(20, 22050, 1) };

// マウスのY座標を変調指数に線形マッピング
~i = { MouseY.kr(0.0, 5.0, 0) };
```

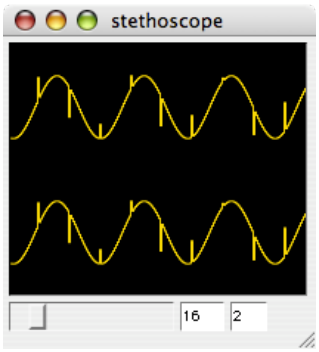
波形は階段状のサイン波になる。

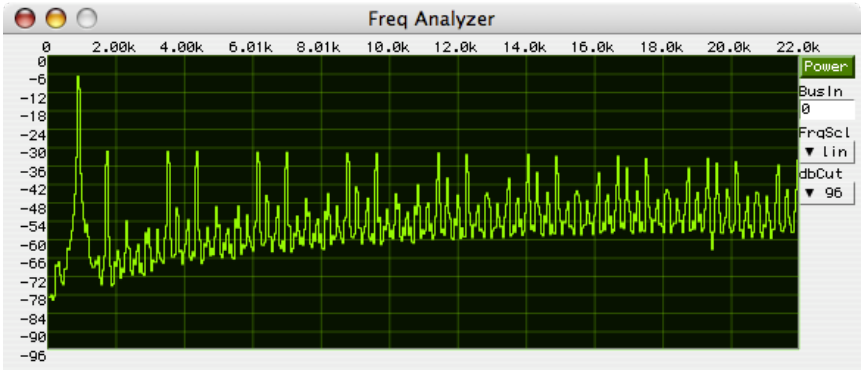





```
// 線の位相を点列でモジュレートする
~out = { SinOsc.ar(~cf.kr, ~cnt.ar, 0.25).dup };
~cnt = { Impulse.ar(~mf.kr, 0, pi) };

// マウスのX座標をキャリア周波数に対数マッピング
~cf = { MouseX.kr(20, 4410, 1) };

// マウスのY座標をモジュレータとキャリアの周波数比に線形マッピング
~mf = { ~cf.kr * MouseY.kr(0.0, 5.0, 0) };
```

スパイクの入ったサイン波になる。





フィードバック周波数変調

モジュレーションした出力を再びモジュレータに入力(フィードバック)することで、より複雑なスペクトル(倍音構造)を生み出す。フィードバック・ゲインをあげていくと、スペクトルはしだいにホワイトノイズ(面)に近づいていく(モジュレーションによる線の面化)。

```
// フィードバック・ゲイン用プロキシの定義
~fb.kr(1);

// 線の周波数を線でフィードバック変調する
~out = { SinOsc.ar(~cnt.ar, 0, 0.3).dup };
~cnt = { ((SinOsc.ar(~mf.kr) + (~out.ar * ~fb.kr)) * ~mf.kr * ~i.kr) + ~cf.kr };


// 変調指数 = 4
~i = 4;

// キャリア周波数 = 4410[Hz]
~cf = 4410;

// マウスのX座標をモジュレータとキャリアの周波数比に対数マッピング
~mf = { ~cf.kr * MouseX.kr(0.01, 1.0, 1) };

// マウスのY座標をフィードバック・ゲインに線形マッピング
~fb = { MouseY.kr(0.0, 1.0, 0) };

// 周波数比をゆっくりと対数増加させる
~mf = { ~cf.kr * XLine.kr(0.01, 1.0, 60, doneAction:2) };
```

パラメータの変化は連続的でも音色の変化は連続的でない。最後はノイズに近くなる。

```
// フィードバック・ゲインを対数スイープする
~fb = { XLine.kr(0.1, 1.0, 60, doneAction:2) };
```

フィードバック・ゲインをゆっくりと大きくしていくとスペクトルにノイズ成分が増えてくる。最後はホワイトノイズになる。

```
// キャリア周波数 = 50[Hz]
~cf = 50;

// 周波数比を大きくする
~mf = { ~cf.kr * MouseX.kr(0.01, 100.0, 1) };

// 線ではなく点列で変調する
~cnt = { ((Impulse.ar(~mf.kr) + (~out.ar * ~fb.kr)) * ~mf.kr * ~i.kr) + ~cf.kr };
```

▼エクササイズ：フィードバック変調の探求

周波数変調以外の変調(振幅変調、パン変調、位相変調)にもフィードバックを適用し、その効果を確かめよ。さまざまなキャリアとモジュレータの組で実験すること。

リング・モジュレーション

2つの波形を直接掛け合わせることによる変調を「リング・モジュレーション」という。2つの音を掛け合わせることで、2音の元の周波数成分が消え、周波数の和と差の周波数が生じる。2つの入力がいくつかの倍音を持っている場合、その結果は非常に複雑なものとなり、鐘が鳴り響くような音色になることが多いため「リング・モジュレーション」という呼ばれるようになった。

リング・モジュレーションは振幅変調と関連している。エンヴェロープの場合同様に、可聴周波数以下の低周波によるリング・モジュレーションから可聴周波数同士のモジュレーションへの遷移を、インタラクティブに聴く。

```
// 周波数設定用プロキシの定義
~f1.kr(1);
~f2.kr(1);

// 波形用プロキシの定義
~w1.ar(1);
~w2.ar(1);

// リング・モジュレーション
~out = { (~w1.ar * ~w2.ar).dup };

// マウスのXY座標を周波数に対数マッピング
~f1 = { MouseX.kr(1, 4410, 1) };
~f2 = { MouseY.kr(1, 4410, 1) };

// 線と線のリング・モジュレーション
~w1 = { SinOsc.ar(~f1.kr, 0, 0.5) };
~w2 = { SinOsc.ar(~f2.kr, 0, 0.5) };

// 線と点列のリング・モジュレーション
~w2 = { Impulse.ar(~f2.kr, 0, 0.5) };

// 線と面のリング・モジュレーション
~w2 = { WhiteNoise.ar(0.5) };

// マウスのY座標を面の振幅に線形マッピング
~w2 = { WhiteNoise.ar(MouseY.kr(0.0, 1.0, 0)) };
```

波形はさらにいくつでも掛け合せることができる。

```
// プロキシの追加
~f3.kr(1);
~w3.ar(1);


// 3本の線によるリング・モジュレーション
~out = { (~w1.ar * ~w2.ar * ~w3.ar).dup };

~f1 = { MouseX.kr(1, 441, 1) };
~f2 = { MouseY.kr(441, 4410, 1) };
~f3 = { SinOsc.kr(0.1, 0, 2205, 2205) };

~w1 = { SinOsc.ar(~f1.kr, 0, 0.5) };
~w2 = { SinOsc.ar(~f2.kr, 0, 0.5) };
~w3 = { SinOsc.ar(~f3.kr, 0, 0.5) };



// 10本のランダムな線(0〜22050[Hz])を掛けあわせる
(
~out = {
    var w = SinOsc.ar(22050.0.rand, 0, 1.0);
    9.do({ w = w * SinOsc.ar(22050.0.rand, 0, 1.0) });
    w.dup; };
)

// 10本のランダムな線(5000〜5100[Hz])を掛けあわせる
(
~out = {
    var w = SinOsc.ar(22050.0.rand, 0, 1.0);
    9.do({ w = w * SinOsc.ar(rrand(5000.0, 5100.0), 0, 1.1) });
    w.dup; };
)

// 20本のランダムな線(5000〜5100[Hz])を掛けあわせる
(
~out = {
```

```
    var w = SinOsc.ar(22050.0.rand, 0, 1.0);
    19.do({ w = w * SinOsc.ar(rrand(5000.0, 5100.0), 0, 1.2) });
    w.dup; };
)

// 100本の線(0~22050[Hz])を掛けあわせるとどうなるか?
(
~out = {
    var w = SinOsc.ar(22050.0.rand, 0, 1.0);
    99.do({ w = w * SinOsc.ar(22050.0.rand, 0, 1.5) });
    w.dup; };
)
```

▼エクササイズ：モジュレーションによる変奏

モジュレーションを用いた旋律をデザインせよ。キャリアとモジュレータの素材と、モジュレーションの方法を選択肢、そのパラメータをエンヴェロープによってコントロールする。旋律の長さは15秒(3秒×5)とする。パラメータの変化をなるべく小さくし、微妙な、しかし時として大きな音色の変化が生じるようなポイントを設定せよ。

▼エクササイズ：モジュレーション＋フィルターによる変奏

ノイズ彫刻のように、モジュレーションによって生成した旋律をフィルターによって加工することで変奏をつくりだす。

```
// 点列の周波数を線でモジュレートする
~src = { Impulse.ar(~cnt.ar, 0, 0.5) };
~cnt = { SinOsc.ar(~mf.kr, 0, ~mf.kr * ~i.kr, ~cf.kr) };

// 変調指数を4とする
~i = 4;

// キャリア周波数を441[Hz]にする
~cf = 441;

// 周波数比を3.7に固定する
~mf = ~cf * 3.7;

// フィルターのカットオフ周波数とレゾナンス用プロキシの定義
~freq.kr(1);
~rq.kr(1);

// レゾナンスありのローパス・フィルターをかける
~out = { RLPF.ar(~src.ar, ~freq.kr, ~rq.kr).dup };

// マウスのX座標をカットオフ周波数に対数マッピング
~freq = { MouseX.kr(20, 20000, 1) };

// マウスのY座標をレゾナンスに対数マッピング
~rq = { MouseY.kr(0.01, 1, 1) };

// レゾナンスありのハイパス・フィルターをかける
~out = { RHPF.ar(~src.ar, ~freq.kr, ~rq.kr).dup };

// バンドパス・フィルターをかける
~out = { Resonz.ar(~src.ar, ~freq.kr, ~rq.kr, 20).dup };

// バンドパス・フィルターをかける
~out = { BRF.ar(~src.ar, ~freq.kr, ~rq.kr, 1).dup };

// 点列の周波数を22050Hzから1Hzまで60秒間かけて対数スイープする
~cnt = { XLine.ar(22050, 1, 60, doneAction:2) };
```

反復からディレイへ

音の時間的操作の中で最も基本的なのが、音の波形をある時間遅らせる「ディレイ」である。

フィードバックをかけることで、ディレイ音が連鎖していく。

一般にディレイは反復構造の時間間隔を短くしたものだといえる。

```
// ソース用プロキシの定義
~src.ar(1);

// 反復時間用プロキシの定義
~t.kr(1);

// ソースを点列とする
~src = { Impulse.ar(~t.kr.reciprocal, 0, 0.8) };

// 反復時間を1秒(1[Hz])とする
~t = 1;

// ソースを聴く
~out = { ~src.ar.dup };

// 反復時間を0.1秒(10[Hz])とする
~t = 0.1;

// 反復時間を1秒から0.001秒まで対数スイープ(しだいに短かく)する
~t = { XLine.kr(1, 0.001, 60, doneAction:2) };

// ソースを1[Hz]の点列に固定する
~src = { Impulse.ar(1, 0, 0.8) };

// ソースにディレイをかける。
~out = { DelayN.ar(~src.ar, 5, ~t.kr, 1, ~src.ar).dup };

// ディレイ時間を0.5秒とする
~t = 0.5;

// ディレイ時間を1秒から0.001秒まで対数スイープ(しだいに短かく)する
~t = { XLine.kr(1, 0.001, 60, doneAction:2) };

// ディレイにフィードバックをかけて反復させる。
~out = { DelayN.ar(~src.ar + ~out.ar, 5, ~t.kr, ~fb.kr, ~src.ar).dup };

// フィードバック・ゲインは0.499
~fb = 0.499;

// ディレイ時間を0.1秒とする
~t = 0.1;

// ディレイ時間を1秒から0.001秒まで対数スイープする
~t = { XLine.kr(1, 0.001, 60, doneAction:2) };
```

反復(パターン)からトーンへの遷移を聴くことができる。

```
// フィードバック・ゲインにモジュレーションをかけてみる
~fb = { SinOsc.kr(10, 0, 0.1, 0.5) };

// モジュレーション周波数を大きくする
~fb = { SinOsc.kr(1000, 0, 0.1, 0.5) };

// ディレイ時間をしだいに長くしていく
~t = 0.1;
~t = 0.5;
~t = 0.9;
~t = 1.1;
~t = 2.sqrt;
~t = 5.sqrt;
```

ディレイ時間を変化させることで、クリックの音色を変化させたり、さまざまなパターンをつくりだすことができる。

▼エクササイズ：ディレイによるノイズ旋律

面(ホワイトノイズ)にはもともと特徴的な周波数はないが、それをディレイによってある時間遅らせることによって、ディレイ時間を周期とする特徴周波数が生まれる。ホワイトノイズをディレイなどによって少しずらして重ねた音を、繰り返しリプルノイズという。

```
// ソースにディレイをかける
~out = { DelayN.ar(~src.ar, 5, ~t.kr, 1, ~src.ar).dup };
```

```
// ホワイトノイズをソースとする
~src = { WhiteNoise.ar(0.2) };

// マウスのX座標をディレイ時間に対数マッピングする(20[Hz]～11025[Hz])
~t = { MouseX.kr(20.0, 11025, 1).reciprocal };
```

ノイズの中からおぼろげな音高が知覚できるようになる。

面(ホワイトノイズ)をソースとして、ディレイを使った旋律をデザインせよ。

▼エクササイズ：ディレイ+モジュレーション

ディレイ時間をモジュレートすることで生れる音色を探求せよ。

```
// フィードバックをかけたディレイ
~out = { DelayN.ar(~src.ar + ~out.ar, 1, ~t.kr, ~fb.kr, ~src.ar).dup };

// フィードバック・ゲインは0.4とする
~fb = 0.4;

// ソースを4410[Hz]の点列とする
~src = { Impulse.ar(4410, 0, 0.1, 0) };

// ディレイ時間をモジュレートする
~t = { SinOsc.kr(~mf.kr, 0, ~ma.kr, ~ma.kr) };

// マウスのX座標をモジュレーション周波数に線形マッピング
~mf = { MouseX.kr(0.0, 20.0, 0) };

// マウスのY座標をモジュレーション振幅に対数マッピング
~ma = { MouseY.kr(0.001, 0.5, 1) };

// ソースを22050[Hz]の点列とする
~src = { Impulse.ar(22050, 0, 0.1, 0) };

// フィードバック・ゲインを0.3とする
~fb = 0.3;

// モジュレーション周波数を高くする
~mf = { MouseX.kr(20.0, 4410.0, 1) };

// ソースを14700[Hz]の線に変える
~src = { SinOsc.ar(14700, 0, 0.1, 0) };

// モジュレーション周波数をもとに戻す
~mf = { MouseX.kr(0.0, 20.0, 0) };

// ソースを面(ホワイトノイズ)に変える
~src = { WhiteNoise.ar(0.1) };
```

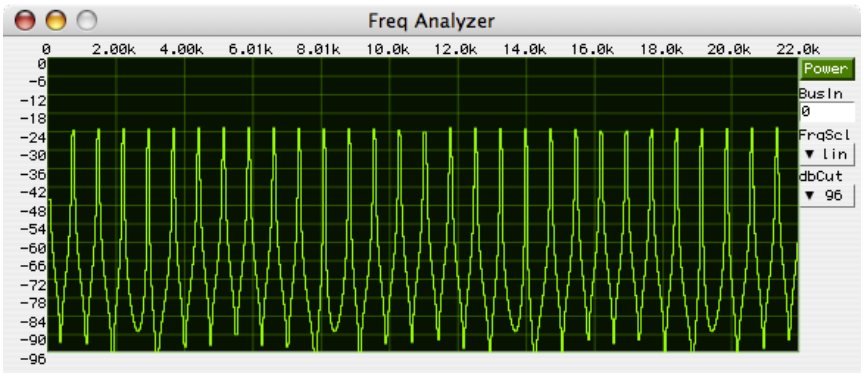
ディレイによるフィルター

くし形フィルター(comb filter)

ディレイを用いて、フィルターをつくることができる。ディレイを用いた最も簡単なフィルターは、入力の2つに分岐させて、その一方にディレイをかけてから、再びミックスする「くし形フィルター」である。くし形フィルターは、ディレイ時間に応じて入力のスペクトルに規則的なピークをつくりだす。それがくしの歯の形に似ていることから「くし形フィルター」という名がつけられた。

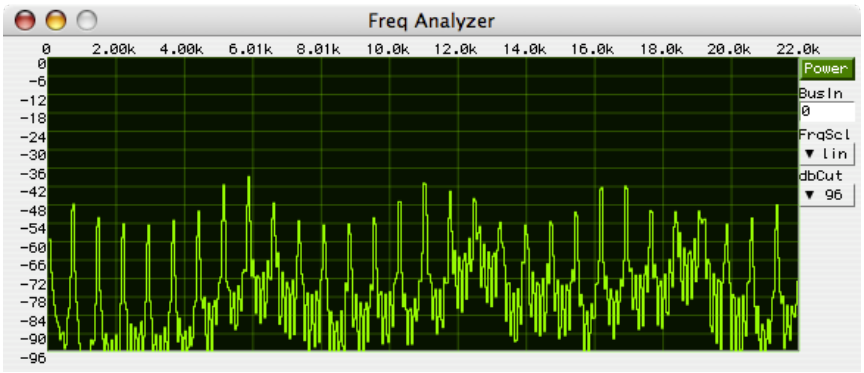
```
// くし形フィルターを定義
~out = { CombN.ar(~src.ar, 1, ~t.kr, 0.2).dup };

// ソースは60サンプル間隔(=44100/60=735Hz)の点列
```

```
~src = { Impulse.ar((44100/60), 0, 0.1, 0) };

// ディレイ時間を1サンプル(1/44100秒)から60サンプルまで、1秒ごとに増加していく
~t = { Line.kr(1, 60, 60, doneAction:2)/44100 };
```

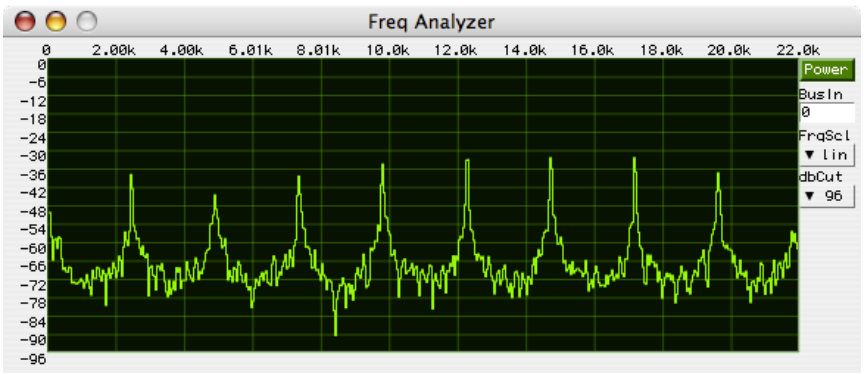


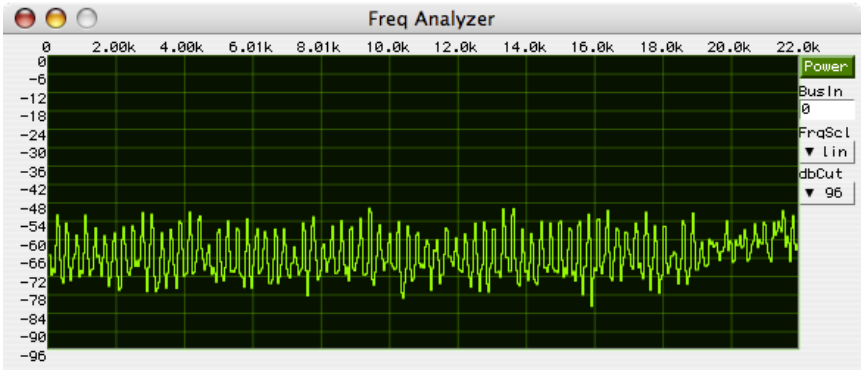
くし形フィルターはディレイ時間の逆数の周波数のレゾネータ(フィルターの一種)として働く。

```
// ディレイ時間をさらに長くする(60サンプルから600サンプルまで)
~t = { Line.kr(60, 600, 60, doneAction:2)/44100 };
```




```
// ソースを面(ホワイトノイズ)に変える
~src = { WhiteNoise.ar(0.01) };

// ディレイ時間を線形スイープする
~t = { Line.kr(1, 60, 60, doneAction:2)/44100 };
~t = { Line.kr(60, 600, 60, doneAction:2)/44100 };
```






```
// ソースを1[Hz]の点列にする
~src = { Impulse.ar(1, 0, 0.5, 0) };

// ディレイ時間を長くする
~t = 0.3;


// ディケイ(減衰)時間を長くする
~out = { CombN.ar(~src.ar, 1, ~t.kr, 10).dup };
```

エコー(フィードバック・ディレイ)のようになる。

```
// ディレイ時間を線でモジュレートする
~out = { CombN.ar(~src.ar, 1, ~t.kr, 0.2).dup };
~t = { SinOsc.kr(~mf.kr, 0, ~ma.kr, ~ma.kr) };

// ソースは60サンプル間隔(=44100/60=735[Hz])の点列
~src = { Impulse.ar((44100/60).dup, 0, 0.1, 0) };

// マウスのX座標をモジュレーション周波数に対数マッピング
~mf = { MouseX.kr(0.1, 22050.0, 1) };

// マウスのY座標をモジュレーション振幅に線形マッピング
~ma = { MouseY.kr(0, 300, 0)/44100 };
```

モジュレーション周波数/振幅によって音色はさまざまに変化する。