# ノイズ

## ノイズ彫刻と音響色彩

## さまざまなノイズ

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り、ランダムに振幅が変化する連続した点群でもある。

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、という光のアナロジーから来ている。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一定(横一直線)である。

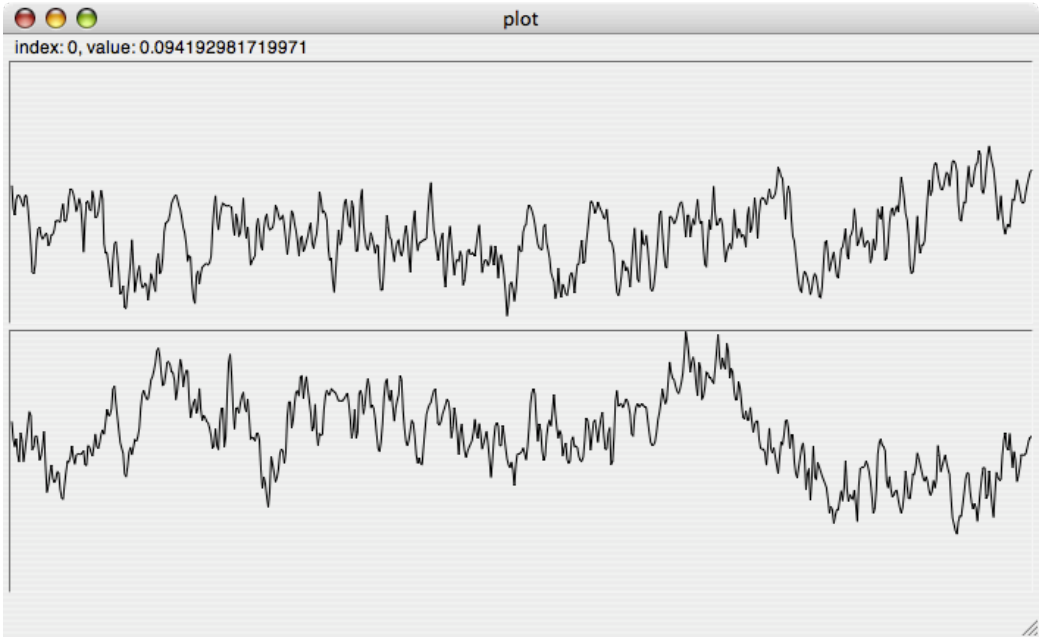
```
// ホワイトノイズ
~out = { WhiteNoise.ar(0.5.dup, 0) }.plot;
```

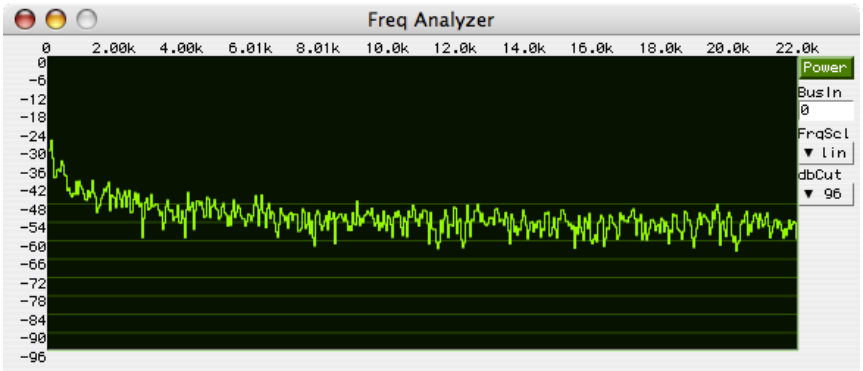
## ピンクノイズ

周波数に反比例して低い周波数成分ほどパワースペクトルが強くなるノイズを ピンクノイズ(桃色雑音または1/fノイズ)と呼ぶ。ホワイトノイズに比べて低音域が膨らんだ、まろやかな音色のノイズである。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たとえからつけられた。

```
// ピンクノイズ
~out = { PinkNoise.ar(0.5.dup, 0) }.plot;
```

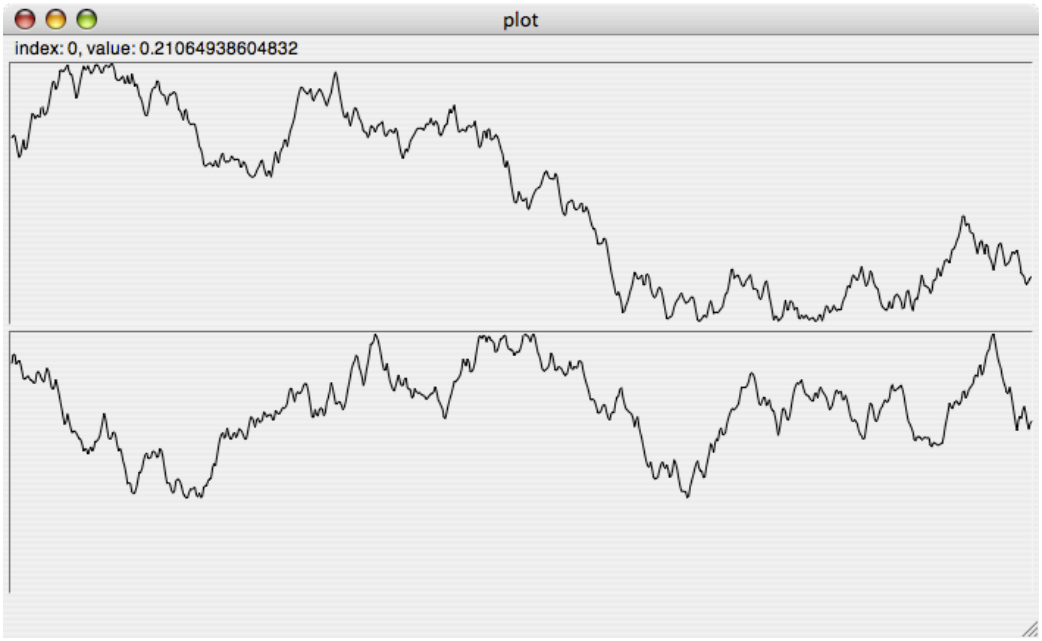
その波形とスペクトルは以下の通りである。

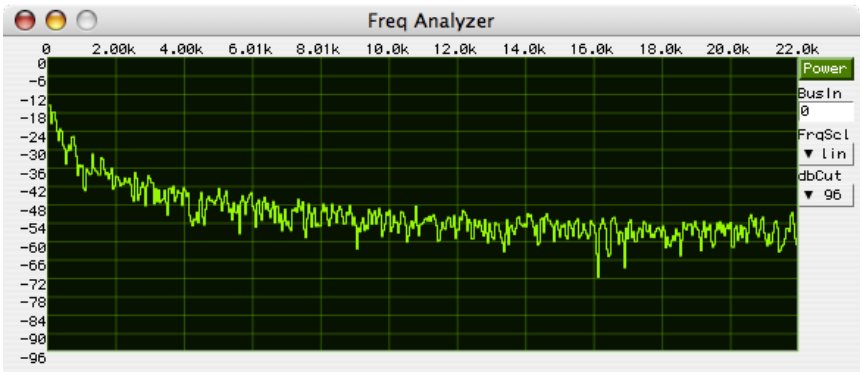


パワースペクトルが周波数の2乗に反比例するノイズをブラウンノイズ(褐色雑音または1/f^2ノイズ)と呼ぶ。ブラウン運動(ランダムウォーク)から波形が得られ、音色はさらに低音が強調されたこもった質感になる。

```
// ブラウンノイズ
~out = { BrownNoise.ar(0.5.dup, 0) }.plot;
```




## フィルター

入出力を持つデジタル・プロセッサーをフィルターと呼ぶ。プロセスの種類によって、入力をさまざまな出力に変形することができる。

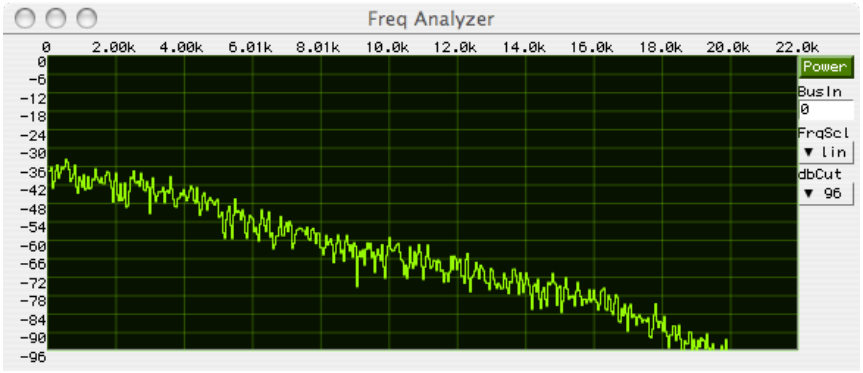
```
// ソース用プロキシの定義
~src.ar(1);

// ソースを面(ホワイトノイズ)とする
~src = { WhiteNoise.ar };
```

## ローパス・ノイズ

まず最初にホワイトノイズをローパス・フィルターに通した音を聴いてみる。ローパス・フィルターはカットオフ周波数よりも高い周波数成分のレベルを下げるフィルターである。ローパス・フィルターを通すことで、ホワイトノイズの音色はピンクノイズやブラウンノイズのように、低音が強調されたこもった音になる。

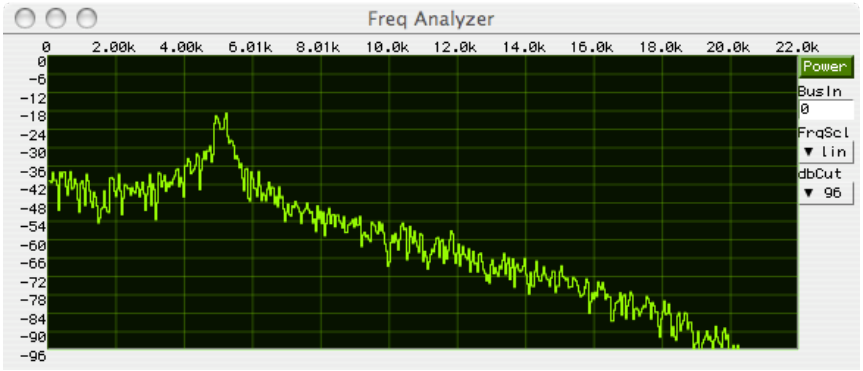
```
// カットオフ周波数用プロキシの定義
~freq.kr(1);
```

```
// ローパス・ノイズ
~out = { LPF.ar(~src.ar, ~freq.kr, 0.2).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する
~freq = { MouseX.kr(20.0, 22050.0, 1) };
```




レゾナンスはカットオフ周波数付近の成分の音のレベルを調整(強調)するパラメータである。

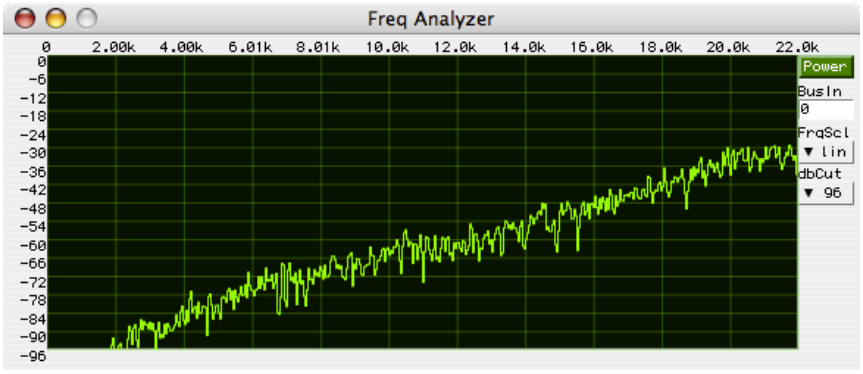
```
// レゾナンス周波数用プロキシの定義
~rq.kr(1);

// レゾナンスつきローパス・ノイズ
~out = { RLPF.ar(~src.ar, ~freq.kr, ~rq.kr, 0.1).dup };

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



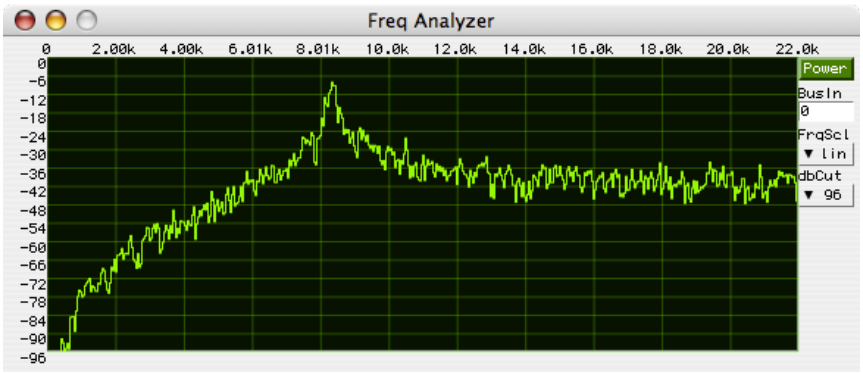

## ハイパス・ノイズ

ハイパス・フィルターは、ローパス・フィルターとは逆に、カットオフ周波数よりも低い周波数成分のレベルを下げるフィルターである。
ハイパス・フィルターを通すことで、ホワイトノイズの音色は高音域が強調されたシャリシャリした音になる。

```
// ハイパス・ノイズ
~out = { HPF.ar(~src.ar, ~freq.kr, 0.5).dup };
```




```
// レゾナンスつきハイパス・ノイズ
~out = { RHPF.ar(~src.ar, ~freq.kr, ~rq.kr, 0.2).dup };
```

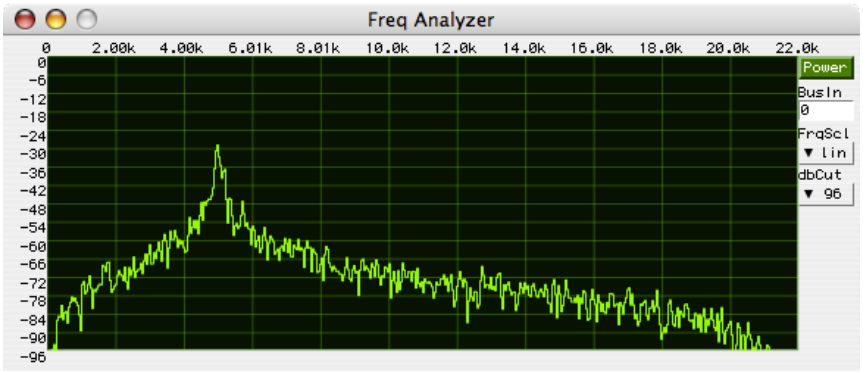



## バンドパス・ノイズ(バンドノイズ)とバンドリジェクト・ノイズ

ハイパス・フィルターは、通過周波数付近の成分以外の音のレベルを下げるフィルターである。レゾナンスで通過周波数の幅(帯域)をコントロールすることができる。

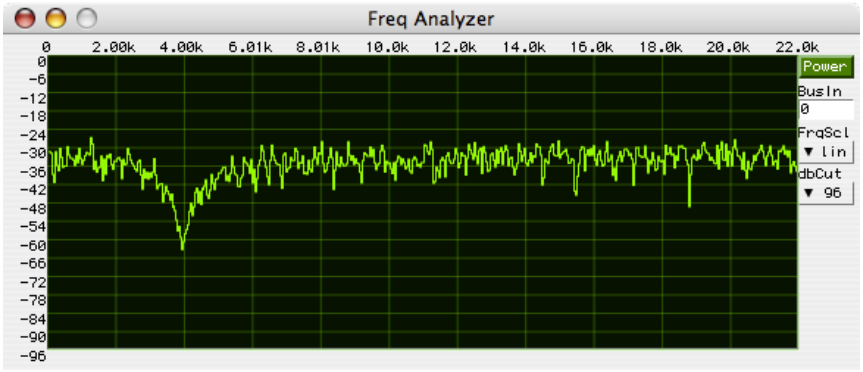
ある特定の周波数の範囲だけの成分を均一の持っているノイズをバンド・ノイズという。ホワイトノイズにバンドパス・フィルターを通すことで、バンドノイズを得ることができる。バンドパス・フィルターの通過周波数のことを、バンドノイズの中心周波数と呼ぶ。

```
// バンドパス・ノイズ
~out = { Resonz.ar(~src.ar, ~freq.kr, ~rq.kr, 1.0).dup };
```



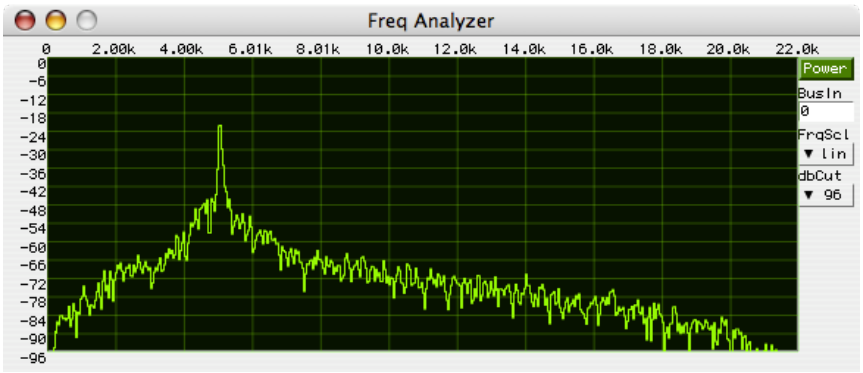

バンドパス・フィルターの逆に、ある周波数(遮断周波数)付近の音だけのレベルを下げるフィルターをバンドリジェクト・フィルターという。

```
// バンドリジェクト・ノイズ
~out = { BRF.ar(~src.ar, ~freq.kr, ~rq.kr, 0.5).dup };
```




バンドパスフィルターの帯域を狭くしていくことで線(サイン波)に近づいていく(フィルターによる面の線化)。ノイズをフィルタリングすることで、ゆらぎをもった線をつくりだすことができる。口笛のような音色である。

```
// バンド幅が非常に狭いバンドパス・ノイズ
~out = { Resonz.ar(~src.ar, ~freq.kr, 0.001, 10.0).dup };
```



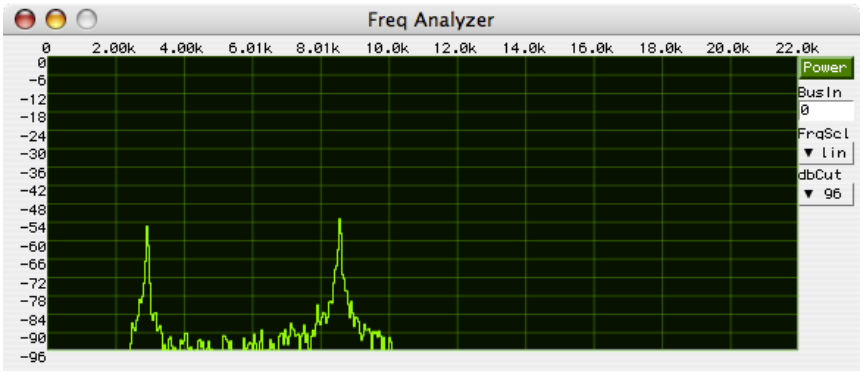


## ノイズ彫刻

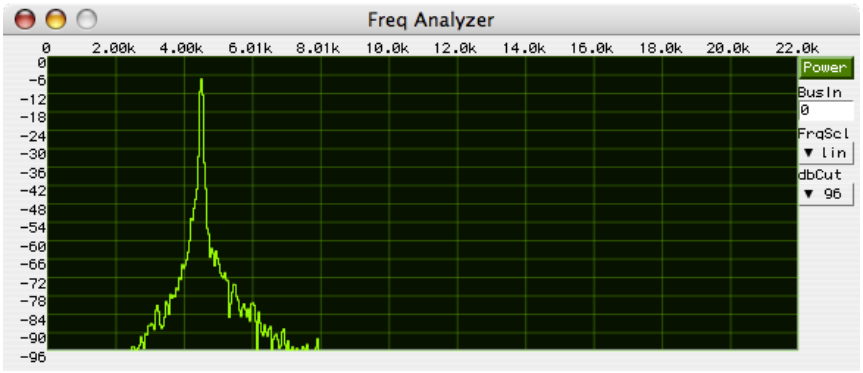
これらのフィルターは、複数重ね合せて用いることができる。あらゆる周波数成分を含むホワイトノイズを、さまざまなフィルターに通すことで、任意のスペクトル分布を有したサウンドを生成することができる。サイン波(線)を重ねあわせて音を組み上げていく加算的手法

を音の塑造(モデリング)だとすれば、ホワイトノイズをフィルターで削っていくことで音をつくりあげていく手法は「ノイズ彫刻」と呼べるだろう。ノイズ彫刻はホワイトノイズをフィルターで減算的に削っていく(一部を切り出す)ことでさまざまなトーン=ハーモニーを生成する「もうひとつの音響和声法」である。

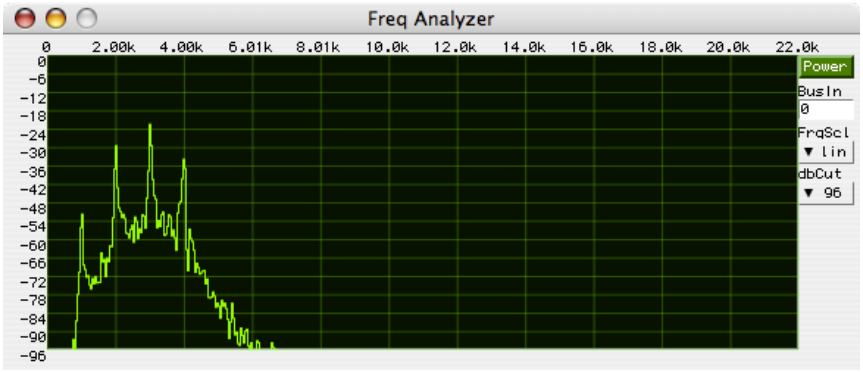
```
// 2つのソースと2つの周波数用のプロキシを定義
~src1.ar(1);
~src2.ar(1);

~freq1.kr(1);
~freq2.kr(1);

// おおもとのソース(src1)は面
~src1 = { WhiteNoise.ar };

// フィルターを2段重ねにする
~out = { Resonz.ar(~src2.ar, ~freq2.kr, 0.001, 20.0).dup };
~src2 = { Resonz.ar(~src1.ar, ~freq1.kr, 0.001, 20.0) };


// マウスのX座標とY座標を2つの通過周波数にそれぞれ対数マッピングする。
~freq1 = { MouseX.kr(20, 22050, 1) };
~freq2 = { MouseY.kr(20, 22050, 1) };
```



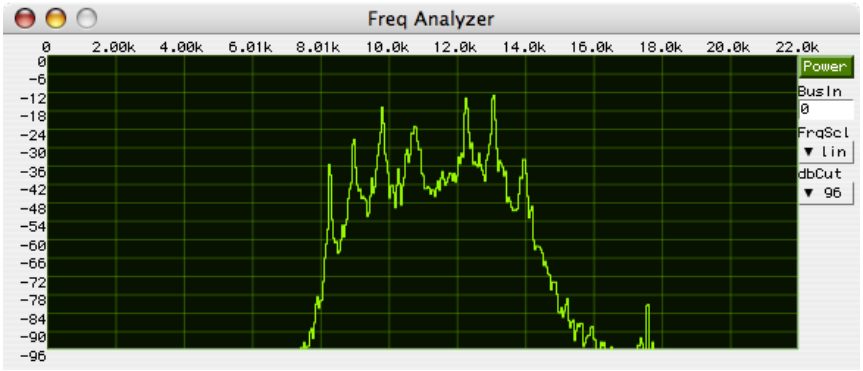

2つのフィルターの通過周波数が重なると強い共鳴を起こす。



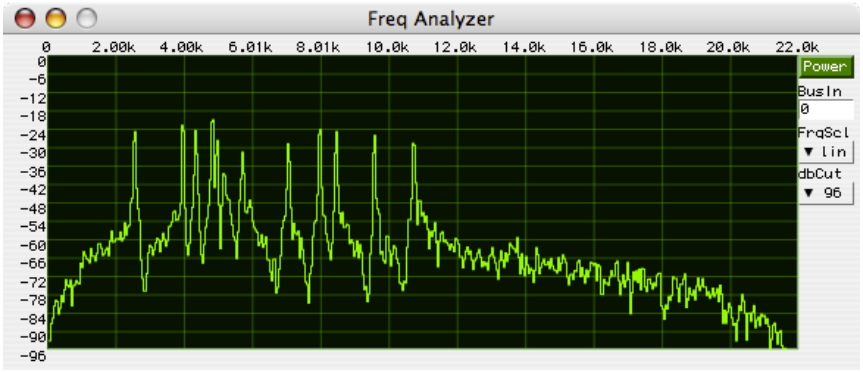

```
// さらに2つのソース用のプロキシを追加
~src3.ar(1);
~src4.ar(1);


// 4つのバンドパス・フィルターによるノイズ彫刻
~out = { Resonz.ar(~src4.ar, 4000, 0.001, 100.0).dup }; // 中心周波数 = 4000[Hz]
~src4 = { Resonz.ar(~src3.ar, 3000, 0.001, 100.0) }; // 中心周波数 = 3000[Hz]
~src3 = { Resonz.ar(~src2.ar, 2000, 0.001, 100.0) }; // 中心周波数 = 2000[Hz]
~src2 = { Resonz.ar(~src1.ar, 1000, 0.001, 100.0) }; // 中心周波数 = 1000[Hz]
~src1 = { WhiteNoise.ar(100) };
```




```
// ランダムな中心周波数による10段のバンドパスフィルター
(
~out = {
    var w = WhiteNoise.ar(100);
    10.do({ w = Resonz.ar(w, 22050.0.rand, 0.001, 100.0) });
    Normalizer.ar(w, 0.8, 0.01).dup; };
)
```




```
// 複数のバンドパスフィルターの組み合せとしてのレゾーネータ・バンクを用いることもできる。
~src = { WhiteNoise.ar(0.01) };
(
~out = { Klank.ar(`[
        [882, 567, 1323, 2646], // 周波数リスト
        [1, 1, 1, 1], // 振幅リスト
        [1, 1, 1, 1]  // 減衰リスト
        ],
        ~src.ar).dup
    };
)

// 12の共鳴周波数を持つランダムなリスト
(
~out = { Klank.ar(`[
        Array.rand(12, 882.0, 11025.0), // 周波数リスト
        Array.rand(12, 0.2, 0.4), // 振幅リスト
        Array.rand(12, 0.1, 2)  // 減衰リスト
        ],
        ~src.ar).dup
    };
)
```




```
// ソースを点列にする
~src = { Impulse.ar(~freq.kr, 0, 0.1) };

// マウスのX座標を点列の周波数に対数マッピングする
~freq = { MouseX.kr(0.1, 4410, 1) };
```

より金属的で精妙な音色になる。

```
// リストを再構築する
~out.rebuild;
```

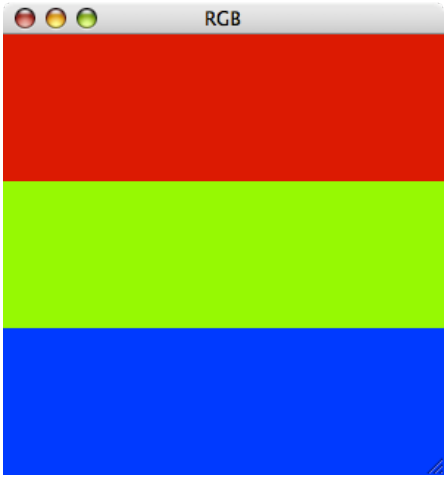
## 音響色彩構成

## 音響的色彩感(音彩)によるノイズ彫刻

## 音の色相

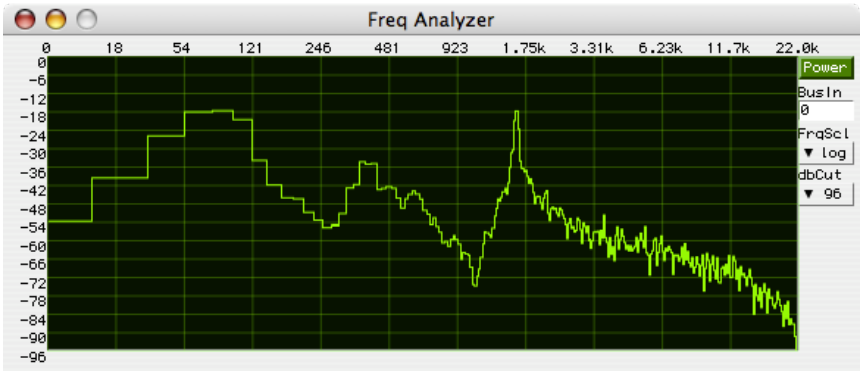
低音=赤(R)、中音=緑(G)、高音=青(B)というアナロジーを用いて、バンドノイズ(ホワイトノイズ+バンドパスフィルター)を重ね合わせてみる。

```
// RGBの3原色を表示
(
var w;
w = SCWindow("RGB", Rect(100, 100, 300, 300));
w.view.background = Color.white;
w.front;
w.drawHook = {
    Pen.use {
        Color.red.set;
        Pen.fillRect(Rect(0, 0, 300, 100));
        Color.green.set;
        Pen.fillRect(Rect(0, 100, 300, 200));
        Color.blue.set;
        Pen.fillRect(Rect(0, 200, 300, 300));
    };
};
w.refresh;
)
```




```
// RGBの3原色をランダム(順不同)にフリップ
(
var w, rgb, run = true;
w = SCWindow("RGB", Rect(100, 100, 300, 300));
w.view.background = Color.white;
w.onClose = { run = false; };
w.front;
w.drawHook = {
    Pen.use {
        rgb = [0, 1, 2].scramble;
        Color.fromArray([1, 0, 0].rotate(rgb.at(0))).set;
        Pen.fillRect(Rect(0, 0, 300, 100));
        Color.fromArray([1, 0, 0].rotate(rgb.at(1))).set;
        Pen.fillRect(Rect(0, 100, 300, 200));
        Color.fromArray([1, 0, 0].rotate(rgb.at(2))).set;
        Pen.fillRect(Rect(0, 200, 300, 300));
    };
};
{ while { run } { w.refresh; 0.1.wait } }.fork(AppClock)
)


// 3原色のアナロジーによる加算合成
~out = ~red + ~green + ~blue;

// ソースを面(ホワイトノイズ)とする
~src = { WhiteNoise.ar };

// 赤音の定義(中心周波数=100[Hz])
~red = { Resonz.ar(~src.ar, 100.0, 0.01, 20).dup };

// 緑音の定義(中心周波数=400[Hz])
~green = { Resonz.ar(~src.ar, 400.0, 0.01, 5).dup };

// 青音の定義(中心周波数=1600[Hz])
~blue = { Resonz.ar(~src.ar, 1600.0, 0.01, 3).dup };
```

赤音と緑音と青音を均等に混ぜるとグレイ音になるか?



赤音と青音のバランスをマウスでコントロールする。

```
// コントロール用プロキシの定義
~rcnt.kr(1);
~gcnt.kr(1);
~bcnt.kr(1);

// 係数を変化させることができる加算合成
~out = (~red * ~rcnt) + (~green * ~gcnt) + (~blue * ~bcnt);

// マウス座標をそれぞれの係数に対数マッピングする
~rcnt = { MouseX.kr(0.01, 2.0, 1) };
~gcnt = 0.5;
~bcnt = { MouseY.kr(0.01, 2.0, 1) };

// 同様の操作(赤が増加、緑が固定、青が減少していく)を色で行ってみる
(
var w, r = 0, g, b = 256, run = true;
w = SCWindow("RGB", Rect(100, 100, 300, 300));
w.view.background = Color.white;
w.onClose = { run = false; };
w.front;
w.drawHook = {
    Pen.use {
        r = (r+2.rand)%256;
        g = 128;
        b = (b-2.rand)%256;
        Color.new255(r, g, b).set;
        Pen.fillRect(Rect(0, 0, 300, 300));
    };
};
{ while { run } { w.refresh; 0.01.wait } }.fork(AppClock)
)


// フィルターのバンド幅を広くする
~red = { Resonz.ar(~src.ar, 100.0, 0.1, 10).dup };
~green = { Resonz.ar(~src.ar, 400.0, 0.1, 3).dup };
~blue = { Resonz.ar(~src.ar, 1600.0, 0.1, 2).dup };
```


## ▼エクササイズ：色彩旋律

低音域、中音域、高音域の3つの異なる中心周波数を持ったバンドパスノイズを組み合わせて、色彩的に旋律をデザインせよ。

## 音の補色

同一の中心周波数によるバンドパス・ノイズとバンドリジェクト・ノイズは互いに「補色」の関係にあるということができる。

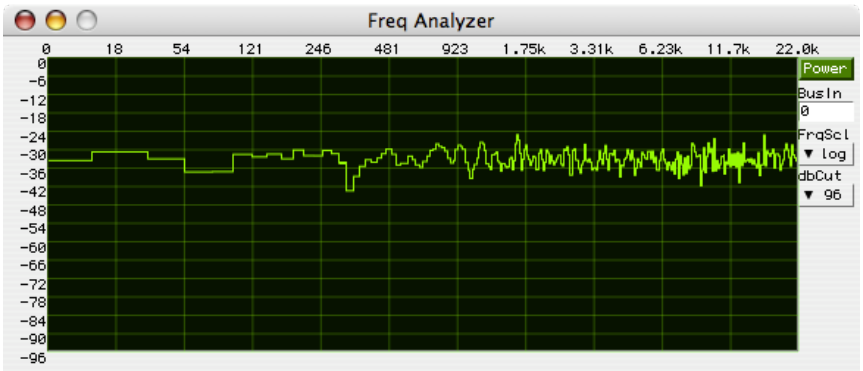
```
// 補色関係にある2色をランダムに表示する
(
var w, r, h, c, run = true;
```

```
w = SCWindow("complementary color", Rect(100, 100, 300, 300));
w.view.background = Color.white;
w.onClose = { run = false; };
w.front;
w.drawHook = {
    Pen.use {
        h = 0.5.rand;
        Color.hsv(h, 1.0, 1.0).set;
        Pen.fillRect(Rect(0, 0, 300, 150));
        Color.hsv(h+0.5, 1.0, 1.0).set;
        Pen.fillRect(Rect(0, 150, 300, 300));
    };
};
{ while { run } { w.refresh; 0.5.wait } }.fork(AppClock)
)
```


```
// バンドパス・ノイズ
~bpn.ar(2);
~bpn = { Resonz.ar(~src.ar, ~freq.kr, ~rq.kr, 0.5).dup };

~src = { WhiteNoise.ar };
~freq = { MouseX.kr(50.0, 2000.0, 0) };
~rq = { MouseY.kr(0.01, 1, 1) };

// バンドリジェクト・ノイズ(バンドパス・ノイズの補色)
~brn.ar(2);
~brn = { BRF.ar(~src.ar, ~freq.kr, ~rq.kr, 0.5).dup };
```

同じ中心周波数のバンドパス・ノイズとバンドリジェクト・ノイズを加えるとホワイトノイズに戻る。

```
// バンドパス・ノイズのみ
~out = ~bpn;

// バンドリジェクト・ノイズのみ
~out = ~brn;

// バンドパス・ノイズとバンドリジェクト・ノイズを加える
~out = ~bpn + ~brn;
```



## ▼エクササイズ：補色旋律

あるバンドパス・ノイズとそれと補色関係にあるバンドリジェクト・ノイズの対比を用いた旋律をデザインせよ。

## 音の彩度

色の明度が音の大きさ、音の色相が音の高さ(中心周波数)に相当するとすれば、彩度に相当するのはバンドパス・ノイズのバンド幅(レゾナンス)である。彩度とはいいかえれば、音の純度であり、バンド幅が狭くなれば純度は増し、逆に広くなれば純度は減る。

```
// ランダムな色相と明度の色の彩度を少しずつ下げていく
(
var w, r, h, s, v, run = true;
w = SCWindow("RGB", Rect(100, 100, 300, 300));
w.view.background = Color.white;
w.onClose = { run = false; };
w.front;
h = 1.0.rand;
s = 256;
v = 1.0.rand;
w.drawHook = {
    s = (s - 1) % 256;
    if (s==0) {h = 1.0.rand; v = 1.0.rand};
    Pen.use {
        Color.hsv(h, s/256.0, v).set;
        Pen.fillRect(Rect(0, 0, 300, 300));
    };
};
{ while { run } { w.refresh; 0.01.wait } }.fork(AppClock)
)


// ノーマライズされたバンドパス・ノイズ
~out = { Normalizer.ar(Resonz.ar(~src.ar, ~freq.kr, ~rq.kr, 0.5), 0.5, 0.01).dup };
~src = { WhiteNoise.ar };

// 中心周波数をマウスのX座標でコントロールする
~freq = { MouseX.kr(50.0, 2000.0, 0) };

// バンド幅をマウスのY座標でコントロールする
~rq = { MouseY.kr(0.0001, 10.0, 1) };

// 最も彩度の高い音は線(サイン波)である
~out = { SinOsc.ar(~freq.kr, 0, 0.5).dup };

// 最も彩度の低い音は面(ホワイトノイズ)である
~out = { WhiteNoise.ar(0.5.dup) };
```

## ▼エクササイズ：彩度旋律

音の彩度の対比を用いた旋律をデザインせよ。

## ▼エクササイズ：和音と旋律による色彩感

線的、動的な音響旋律と面的、静的な音響和音を組みあわせて、そこからどのような音響的色彩感覚が生まれるかを探索せよ。

## 周期音と非周期音

波形が周期的に繰り返す音を「周期音(periodic tone)」、それ以外の音を「非周期音(non-periodic tone)」という。周期音は基本的な繰り返しの単位としての基音(fandamental tone)が存在し、そこにその整数倍の周波数をもつ倍音(harmonic tone)が重なる。基音に非整数次の倍音が重なっていくと、その音は次第に非周期音に近づいていく。

線(サイン波)は周期音である。面(ホワイトノイズ)は非周期音である。しかし周期音と非周期音の境界は曖昧である。複数の線を重ねあわせていくことで、周期音を次第に非周期音に近づけていくことができる。面にフィルターを重ねていくことで、非周期音を周期音に近づけていくことができる。

孤立した点、あるいは非常に短い線は、非周期音である。クリック音はホワイトノイズ同様、すべての周波数成分が含まれている。同様に非常に短い線(トーンバースト)のスペクトルは、クリックに近づいていく。

(注)ホワイトノイズはそれを構成するサイン波の位相がばらばらであるが、クリック音は位相がそろっている。

点が反復し点列になることによって、基音が生れ次第に周期音に近づいていく。

点：非周期音
点列(点の反復)：非周期音→周期音

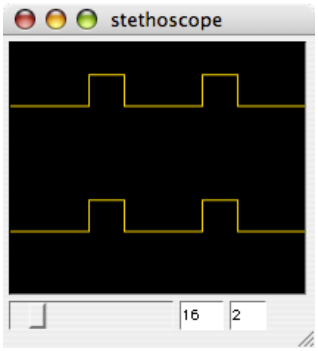
線：周期音
短い線：周期音→非周期音
短い線の反復：非周期音→周期音
線の重ねあわせ：周期音→非周期音

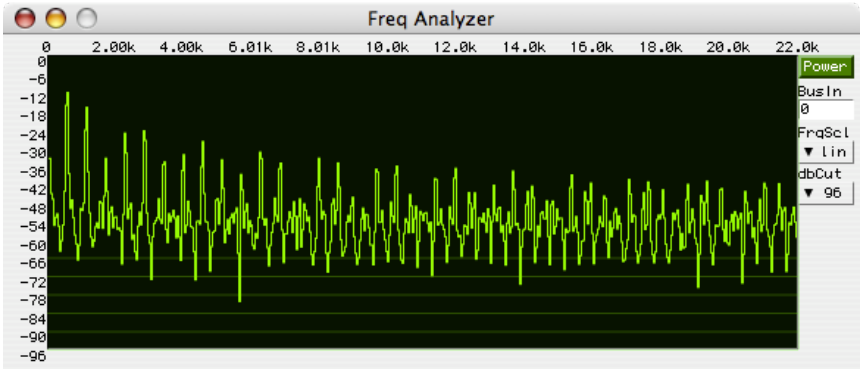
面：非周期音
フィルターされた面：非周期音→周期音

## 周期音と非周期音

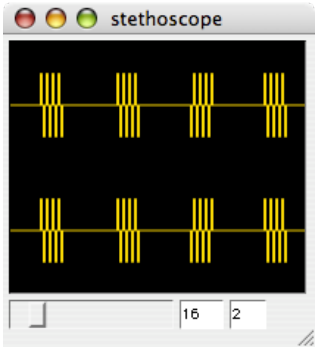
低周波の点(矩形波)

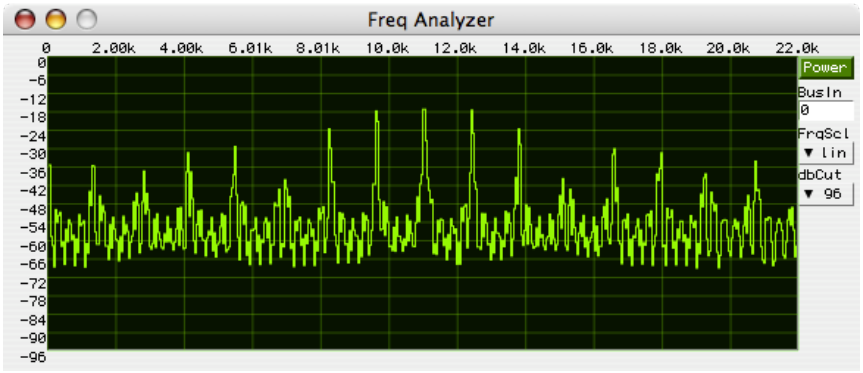
```
// 矩形波は周期音と非周期音の両方のスペクトルの特徴を持っている。
~width.kr(1);
~out = { LFPulse.ar(~freq.kr, 0, ~width.kr, 0.5).dup };

// マウスのX座標を矩形波の周波数に対数マッピング
~freq = { MouseX.kr(1.0, 11025, 1) };

// マウスのY座標をパルス幅(周期に対する割合)に線形マッピング
~width = { MouseY.kr(0.0, 1.0, 0) };
```



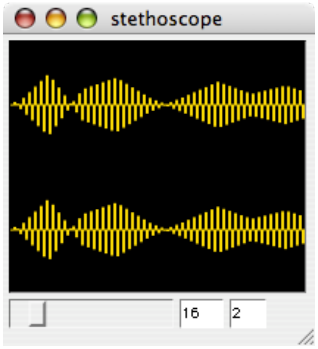



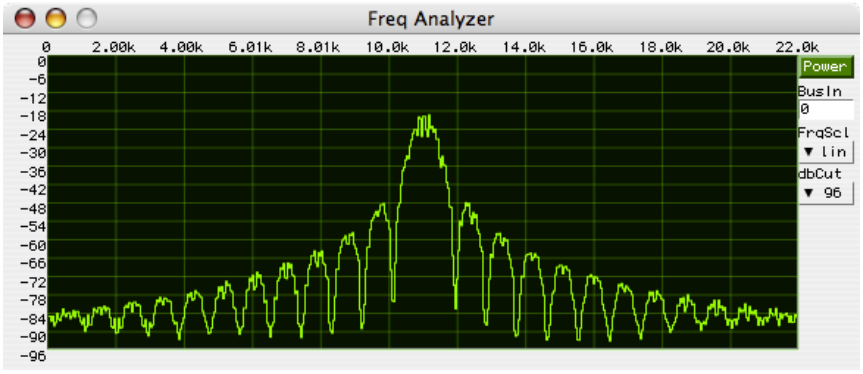

```
// 線に矩形波を掛けあわすこと(リング・モジュレーション)で断続的にオンオフする。
~out = { ~src.ar * LFPulse.ar(~freq.kr, 0, ~width.kr, 0.5).dup };
~src = { SinOsc.ar(11025.0) };
```




```
// 周期音(線)と非周期音(低周波ノイズ)の掛け合わせ
~out = { ~src.ar * LFNoise0.ar(~freq.kr, 0.5).dup };
~out = { ~src.ar * LFNoise1.ar(~freq.kr, 0.5).dup };
```



## ▼エクササイズ：断続によるノイズ旋律

面の断続によるノイズ旋律をデザインせよ。

面にピッチ(周波数)のパラメータはない。面(ホワイトノイズ)をある周波数で断続すること(面と矩形波のリング・モジュレーション)によりピッチを生成できるだろうか?

```
// 面を矩形波で断続的にオンオフする。
~out = { ~src.ar * LFPulse.ar(~freq.kr, 0, ~width.kr, 0.3).dup };
~src = { WhiteNoise.ar };

~freq = { MouseX.kr(20.0, 22050.0, 1) };
~width = { MouseY.kr(0.0, 1.0, 0) };
```