

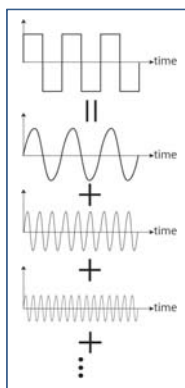
# インタラクティブシステム論 第4回

梶本裕之  
Twitter ID kajimoto  
ハッシュタグ #ninshiki

## 日程

- 4/13 第1回 イントロダクション
- 4/20 第2回 フーリエ変換
- 4/27 第3回 フーリエ変換と線形システム
- 5/4 **みどりの日**
- 5/11 **出張により休講**
- 5/18 第4回 信号処理の基礎
- 5/25 第5回 信号処理応用1(相関)
- 6/1 第6回 信号処理応用2(画像処理)
- 6/08 **インタラクティブシステムの実際(小泉先生)**
- 6/15 第7回 ラプラス変換
- 6/22 第8回 古典制御の基礎
- 6/29 **中間確認テスト(出張予定. 変更の可能性)**
- 7/6 第9回 行列
- 7/13 第10回 行列と最小二乗法
- 7/20 第11回 インタラクティブシステムと機械学習
- 7/27 第12回 ロボティクス
- 8/3 期末テスト? 要確認 **(出張の可能性)**

## (復習): フーリエ級数展開



周期Tの波形 f(t)は次のように分解できる

$$f(t) = a_0 + \sum_{m=1}^{\infty} a_m \cos(2\pi m t / T) + \sum_{m=1}^{\infty} b_m \sin(2\pi m t / T)$$

$$a_0 = \frac{1}{T} \int_0^T f(t) dt \quad \text{平均値 (DC成分)}$$

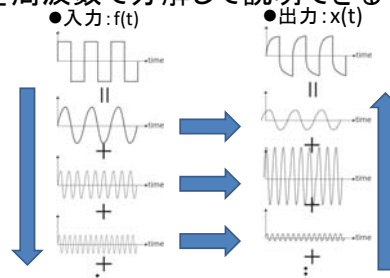
$$a_m = \frac{2}{T} \int_0^T f(t) \cos(2\pi m t / T) dt$$

$$b_m = \frac{2}{T} \int_0^T f(t) \sin(2\pi m t / T) dt$$

※この授業では係数は気にしない。

## (復習: フーリエ級数展開)

歪みを周波数で分解して説明できる



- (1) 入力 f(t) を周波数分解する
  - (2) 周波数ごとの出力の振幅と位相を求める。
  - (3) 合計すると出力が得られる。
- これを連続関数で考えるとどうなるか？

## (復習) フーリエ変換

フーリエ級数展開は周期的な信号を分解するのに使われた。フーリエ変換は周期的ではない信号に対する変換。Tを無限大とした極限から導かれる。逆フーリエ変換によって元に戻すことができる。

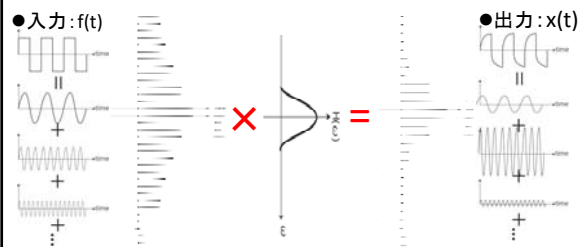
フーリエ変換

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-j\omega t) dt$$

逆フーリエ変換

$$f(t) = \int_{-\infty}^{\infty} F(\omega) \exp(j\omega t) d\omega$$

## (復習) 入出力の関係: 関数同士の掛け算



- (1) 入力 f(t) を周波数分解  $\Rightarrow F(\omega)$
- (2) 周波数ごとにどれだけ振幅と位相が変わるか:  $H(\omega)$
- (3) 出力 (のフーリエ変換):  $X(\omega) = H(\omega) * F(\omega)$
- (4) 逆フーリエ変換すると出力が得られる:  $x(t)$

**(復習) 伝達関数**

●入力:  $f(t)$

●出力:  $x(t)$

フーリエ空間では、入出力は単なる掛け算で表される。  
 $X(\omega) = H(\omega) \times F(\omega)$

この入出力関係を定義するシステムの性質  $H(\omega)$  を伝達関数と呼ぶ。

**今日の話題: 周波数領域ではなく、時間領域のまま議論できないか?**

●入力:  $f(t)$

●出力:  $x(t)$

$X(\omega) = H(\omega) \times F(\omega)$ : 周波数領域で美しいのは分った。時間的な現象として何が起きているのか分からない。

**式で考えよう**

フーリエ変換  $F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-j\omega t) dt$

逆フーリエ変換  $f(t) = \int_{-\infty}^{\infty} F(\omega) \exp(j\omega t) d\omega$

$X(\omega) = H(\omega) \times F(\omega)$

両辺を逆フーリエ変換すれば時間領域の信号に戻る。

$x(t) =$

$=$

$=$

$=$

**逆順の計算もしておく(ふつうはこちら)**

フーリエ変換  $F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-j\omega t) dt$

逆フーリエ変換  $f(t) = \int_{-\infty}^{\infty} F(\omega) \exp(j\omega t) d\omega$

$x(t) = \int_{-\infty}^{\infty} f(\tau) h(t - \tau) d\tau$

両辺をフーリエ変換。

$X(\omega) = \int_{-\infty}^{\infty} \exp(-j\omega t) dt \int_{-\infty}^{\infty} f(\tau) h(t - \tau) d\tau$

$= \int_{-\infty}^{\infty} \exp(-j\omega(\tau + (t - \tau))) dt \int_{-\infty}^{\infty} f(\tau) h(t - \tau) d\tau$

$= \int_{-\infty}^{\infty} \exp(-j\omega\tau) \cdot \exp(-j\omega(t - \tau)) dt \int_{-\infty}^{\infty} f(\tau) h(t - \tau) d\tau$

$= \int_{-\infty}^{\infty} d\tau \int_{-\infty}^{\infty} dt$

$= \int_{-\infty}^{\infty} d\tau \int_{-\infty}^{\infty} dt'$

$= F(\omega) H(\omega)$

**コンボリューション定理**

$X(\omega) = F(\omega) H(\omega) = H(\omega) F(\omega)$

フーリエ逆変換 ↓ ↑ フーリエ変換

$x(t) = \int_{-\infty}^{\infty} f(\tau) h(t - \tau) d\tau = \int_{-\infty}^{\infty} h(\tau) f(t - \tau) d\tau$

簡略化のため次のようにも表記

$x(t) = f(t) * h(t) = h(t) * f(t)$

**コンボリューション定理の意味するところ(1)**

●入力:  $f(t)$

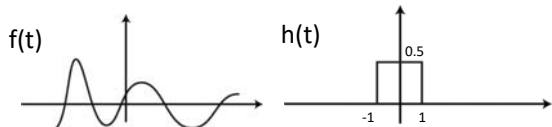
●出力:  $x(t)$

$x(t) = \int_{-\infty}^{\infty} f(\tau) h(t - \tau) d\tau$       $X(\omega) = F(\omega) H(\omega)$

● $h(t)$ のフーリエ変換が $H(\omega)$ であるとする。  
 ●周波数領域でフィルタ $H(\omega)$ をかけることは、時間領域では、入力信号 $x(t)$ に対する関数 $h(t)$ の畳み込み積分(コンボリューション)として表現される。

コンボリューション定理の意味するところ(2)

$$x(t) = \int_{-\infty}^{\infty} h(\tau) f(t-\tau) d\tau$$



例えば,  $h(t)=0.5$  ( $-1 < t < 1$ )なら,

$$x(t) =$$

これは,  $f(t)$ を平均化していくフィルタ

平均化?

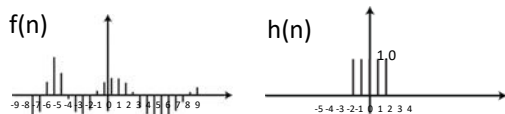


ノイズを「ならし」て大域的な特徴をつかむ

離散化による理解

$$x(t) = \int_{-\infty}^{\infty} h(\tau) f(t-\tau) d\tau \rightarrow x(n) = \sum_{i=-\infty}^{\infty} h(i) f(n-i)$$

$$x(n) = \dots + h(-4)f(n+4) + h(-3)f(n+3) + \dots + h(3)f(n-3) + h(4)f(n-4) + \dots$$



$h(n)$ が,  $n=-2 \sim 2$ の間だけ1の場合,

$$\begin{aligned} x(1) &= f(3) + f(2) + f(1) + f(0) + f(-1) \\ x(2) &= f(4) + f(3) + f(2) + f(1) + f(0) \\ x(3) &= f(5) + f(4) + f(3) + f(2) + f(1) \\ x(4) &= f(6) + f(5) + f(4) + f(3) + f(2) \\ x(n) &= f(n+2) + f(n+1) + f(n) + f(n-1) + f(n-2) \end{aligned}$$

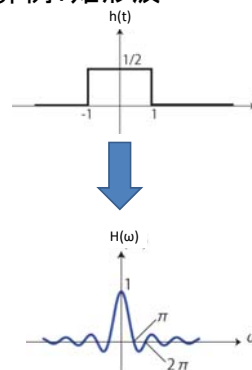
出力 $x$ は, 入力 $f$ の「平均化」になっている。

(復習)フーリエ変換の計算例: 矩形波

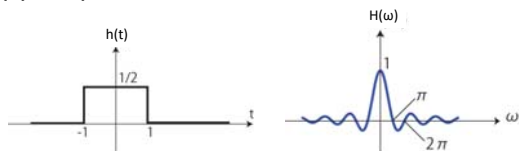
$$h(t) = \begin{cases} 1/2 & -1 \leq t \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$H(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-j\omega t) dt$$

$$\begin{aligned} &= \int_{-1}^1 \frac{1}{2} \exp(-j\omega t) dt \\ &= \left[ \frac{1}{-j2\omega} \exp(-j\omega t) \right]_{-1}^1 \\ &= \frac{1}{-j2\omega} (\exp(-j\omega) - \exp(j\omega)) \\ &= \frac{1}{-j2\omega} (\cos(\omega) - j \sin(\omega) - \cos(\omega) - j \sin(\omega)) \\ &= \frac{-j \sin(\omega)}{-j\omega} \\ &= \frac{\sin(\omega)}{\omega} \end{aligned}$$



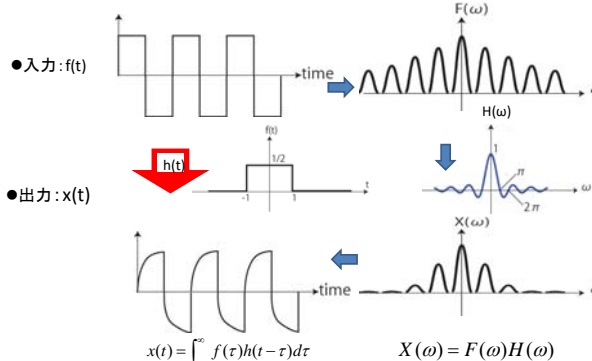
$h(t)$ と $H(\omega)$ の関係: フーリエ変換



つまり, この $h(t)$ のフーリエ変換 $H(\omega)$ は, 大雑把には「低い周波数で大きな値をとり, 高い周波数で小さな値をとる」すなわち, 低域通過フィルタ (LPPF: Low Pass Filter) である。

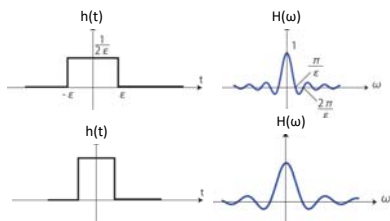
時間領域での「平均化(平滑化)フィルタ」  
≡ 周波数領域での「ローパスフィルタ」

実時間での矩形波による平均化  
= フーリエ空間でのsinc関数による低域通過



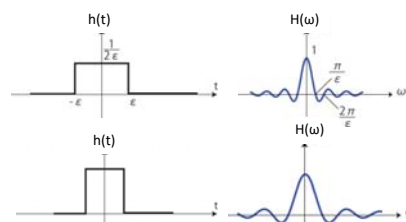
(復習) 矩形波の幅が変わると?

$$h(t) = \begin{cases} \frac{1}{2\varepsilon} & -\varepsilon \leq t \leq \varepsilon \\ 0 & \text{otherwise} \end{cases} \rightarrow H(\omega) =$$



矩形波の幅を狭くする ⇒ フーリエ変換結果は幅広に

平均化の時間幅と周波数帯域の関係

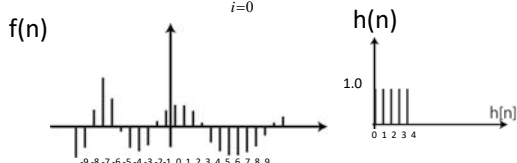


矩形波の幅を狭くする ⇒ フーリエ変換結果は幅広に

時間的な平均化(平滑化)フィルタの幅が広いほど周波数的には低い周波数しか通さなくなる。

時間軸の離散化: FIRフィルタによる実装

$$x(n) = \sum_{i=0}^{\infty} h(i) f(n-i)$$



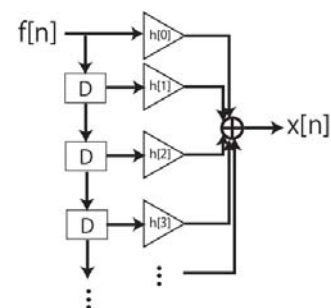
i=0から始める: 未来のデータが使えないことを意味する。この例は、元データf(n)を、4個平均して出力する。

- 未来のデータが使えない例: リアルタイム制御
- 先のデータが使える例: 画像処理

FIRフィルタの図的理解

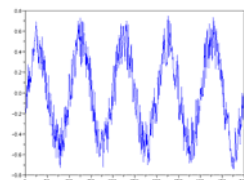
$$x(n) =$$

D: Delay, 遅延器, メモリ.  
h[n]: 増幅器



FIR=Finite Impulse Response  
個々のインパルス応答を有限個足し合わせたもの。

平滑化フィルタの実例(1)



元の信号に高周波ノイズが含まれている。

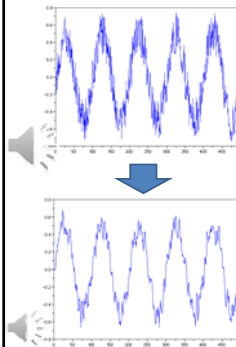


Scilabコード例

```
time = [0:0.01:100];
//振幅0.5の正弦波に最大振幅0.5のノイズが混入
wave=0.5*sin(time*2*pi) + 0.5*(rand(time)-0.5);
playsnd(wave);
savewave('wave.wav', wave);
plot(wave(1:500));
```

平滑化フィルタの実例(2)

メモリを三つ持ったFIRフィルタによって平滑化



Scilabコード例

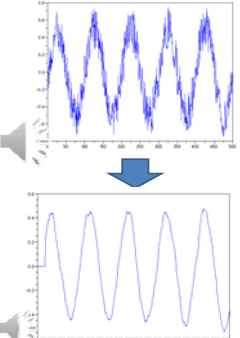
```
time = [0:0.01:100];
//振幅0.5の正弦波に最大振幅0.5のノイズが混入した信号
wave=0.5*sin(time*2*pi) + 0.5*(rand(time)-0.5);

out=zeros(wave);
//3つを平均する。
for n=3:length(wave),
    out(n)=out(n)+wave(n-1)/3;
end
end
playsnd(out);
savewave('wave.wav', out);
plot(out(1:500));
```

### 平滑化フィルタの実例(3)

メモリを20個持ったFIRフィルタによって平滑化

Scilabコード例



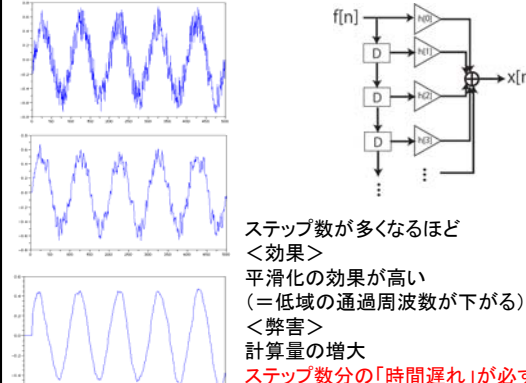
```

time = [0:0.01:100];
//振幅0.5の正弦波に最大振幅0.5のノイズが混入した信号
wave=0.5*sin(time*2*pi) + 0.5*(rand(time)-0.5);
out=zeros(wave);

//20個を平均する.
for n=20:length(wave),
    for i=0:19,
        out(n)=out(n)+wave(n-i)/20;
    end
end

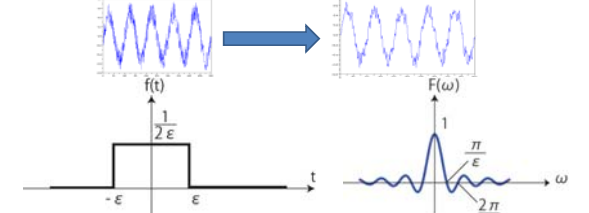
playsnd(out);
savewave('wave.wav',out);
plot(out(1:500));
    
```

### FIRフィルタによる平滑化の効果と弊害



ステップ数が多くなるほど  
 <効果>  
 平滑化の効果が高い  
 (=低域の通過周波数が下がる)  
 <弊害>  
 計算量の増大  
 ステップ数分の「時間遅れ」が必ず生じる

### どのくらいの周波数まで通過させるか

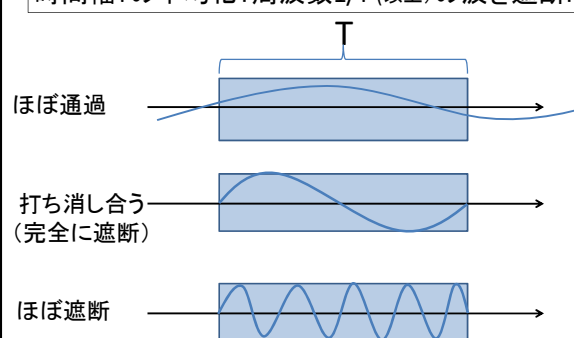


幅 $2\epsilon$ の矩形波のフーリエ変換: 角周波数 $\pi/\epsilon$ で0

時間幅 $T$ で平均化する場合:  
 角周波数 $2\pi/T$ (周波数 $1/T$ )以上の波を遮断.

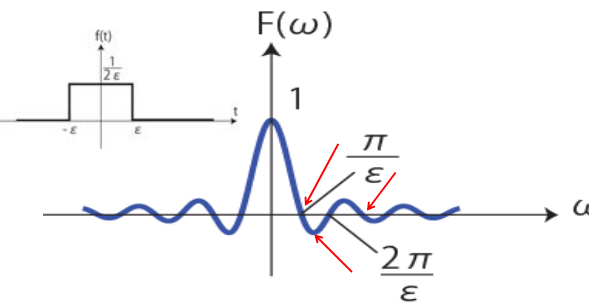
### 平均化による遮断

時間幅 $T$ の平均化: 周波数 $1/T$ (以上)の波を遮断.



ほぼ通過  
 打ち消し合う (完全に遮断)  
 ほぼ遮断

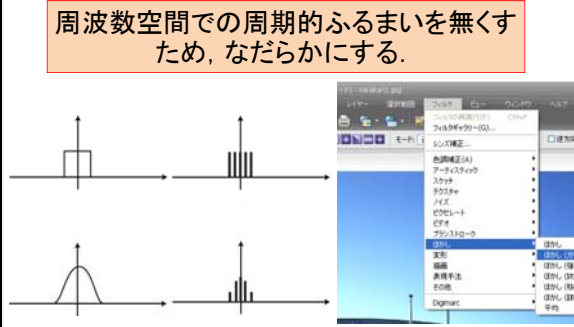
### 単純平均化によるローパスの落とし穴



特定の周波数は全く通さないが、高周波成分の遮断が周期的ふるまいを示す.

### 実際のローパス

周波数空間での周期的ふるまいを無くすため、なだらかにする.



画像の世界では...「ガウスぼかし」

### 平均化によるローパスフィルタ:まとめ

- 入力:  $f(t)$
- 出力:  $x(t)$

$x(t) = \int_{-\infty}^{\infty} f(\tau)h(t-\tau)d\tau$   
 $H(\omega)$ : 周波数的なローパス  
 $X(\omega) = F(\omega)H(\omega)$

### 逆に高い周波数成分だけ取り出すには？

- ローパスフィルタ: 低い周波数成分だけを取り出した
- 元信号と低周波信号の差をとれば, 高周波成分だけ取り出せる？

- 入力  $f(t)$
- ローパス  $x(t)$
- 高周波成分  $y(t)$

### ハイパスフィルタのFIRフィルタによる実装

- 入力  $f(t)$
- ローパス  $x(t)$
- 高周波成分  $y(t)$

$f(n)$        $h(n)$

ローパス例:  $x(n) = (f(n+2) + f(n+1) + f(n) + f(n-1) + f(n-2))/5$   
 ハイパス例:  $y(n) = f(n) - x(n)$   
 $= -1/5 \cdot f(n+2) - 1/5 \cdot f(n+1) + 4/5 f(n) - 1/5 \cdot f(n-1) - 1/5 \cdot f(n-2)$

### ハイパスフィルタのFIRフィルタによる実装

ローパス:

- 強  $x(n) = (f(n+2) + f(n+1) + f(n) + f(n-1) + f(n-2))/5$
- ↑  $x(n) = (f(n+1) + f(n) + f(n-1) + f(n-2))/4$
- ↓  $x(n) = (f(n+1) + f(n) + f(n-1))/3$
- 弱  $x(n) = (f(n) + f(n-1))/2$

ハイパス:  $y(n) = f(n) - x(n)$

ローパスが{強い・弱い}ほど, ハイパスは{弱く・強く}なる

最も簡単な場合:  
 $x(n) = (f(n) + f(n-1))/2$   
 ハイパス:  
 $y(n) = f(n) - x(n) =$   
 つまり, 直前との「差分(微分)」.

### ハイパスフィルタ ≡ 微分フィルタ

- 入力  $f(t)$
- 高周波成分  $y(t)$

$y(1) = (f(1) - f(0))/2$   
 $y(2) = (f(2) - f(1))/2$   
 $y(3) = (f(3) - f(2))/2 \dots$

用語整理

(周波数表現)	(時間軸表現)
ローパスフィルタ = 低域通過フィルタ	= 平滑化フィルタ
ハイパスフィルタ = 高域通過フィルタ	= 微分フィルタ

### ハイパスフィルタの例

直前との差分によってハイパス

Scilabコード例

```

time = [0:0.01:100];
//振幅0.5の正弦波に最大振幅0.5のノイズが混入した信号
wave=0.5*sin(time*2*pi) + 0.5*(rand(time)-0.5);

out=zeros(wave);

//差分をとる
for n=2:length(wave),
    out(n)=wave(n)-wave(n-1);
End

playsnd(out);
savewave('wave.wav',out);
plot(out(1:500));
    
```

### フィルタリング... 研究の現場で

筋電計測による笑いの検出→増幅は可能か？

### 研究の現場で

筋電計測:  
 ●心拍による成分: 非常に大きい, 低周波  
 ●笑いによる成分: 小さいが, 高周波

### 研究の現場で

笑っていない時のパワースペクトル  
 笑っている時のパワースペクトル

- (1) ハイパスフィルタで笑い成分を抽出
- (2) 絶対値化フィルタで正の値に変換
- (3) ローパスフィルタで笑い領域を確定

### 参考: エコー

エコー=時間遅れ信号の重畳.  
 これもFIRフィルタで実装できる.

```

Scilabコード例
wave = loadwave('aiueo.wav');
out=zeros(wave);
//エコー(1000ステップ前の信号を重畳)
for n=1000:length(wave),
    out(n)=wave(n)+0.9*wave(n-999);
end
playsnd(out,11000); //11kHzサンプリングで再生
savewave('wave.wav',out,[11000]);
    
```

原音  
 1000ステップ前の信号を重畳  
 1000ステップ前+2000ステップ前の信号を重畳  
 沢山重畳

### レポート課題1

適当なwaveファイルに対して次の三つの操作を行う。  
 (1) FIRフィルタによるローパスフィルタをかけて音をくもらせる。  
 (2) FIRフィルタによる適当なハイパスフィルタをかけて音をとがらせる。  
 (3) エコーを掛けてカラオケのようにする。

Scilabのソースファイルのみ添付すること(原音のwaveファイルは不要です)

※注: Waveファイルの形式によってはScilabで扱えない場合があります。その場合は、例えばWindows標準のWaveサウンドファイル(Windows開始音等)を使うとうまくいきます。C:\Windows\Mediaの下にあります。

### PCで信号を扱う=離散化

●元のアナログ信号⇒サンプリングによって離散的なデータに

●離散化の間隔が十分狭ければ...

元に戻せそう

●離散化の間隔が広いと...

サンプリング間隔2倍  
 サンプリング間隔4倍  
 元に戻せなそう

この違いはなんだろうか? 「元に戻す」とはなんだろうか?

### 元に戻せない(=元が推測できない)場合

元の信号:  $\sin(x)$ , 周期  $2\pi$

離散化の間隔  $3/2\pi$

なめらかに結ぶと...

元と全く異なる波形となる=エイリアシング

### 離散化に際して: ナイキスト周波数

元の信号:  $\sin(x)$ , 周期  $2\pi$

離散化の間隔  $\pi$

なめらかに結ぶ

うまくいく場合と、うまくいかない場合がありそう

### 離散化に際して: ナイキスト周波数未満

元の信号:  $\sin(x)$ , 周期  $2\pi$

離散化の間隔  $3/4\pi$

正弦波でなめらかに結ぶ

元の波形が再現できる!!

### サンプリング定理(標本化定理)

元信号に含まれる最高周波数の、**倍より高い周波数**でサンプリング(標本化)していれば、元信号はサンプリング点から完全に再生できる。

倍の周波数=ナイキスト周波数

### サンプリング定理(標本化定理)

逆に、エイリアシングを生じないために、**サンプリング周波数の半分以上の周波数は、あらかじめカット**する必要がある。(後でカットしても意味無し!)

カットしないとエイリアシングを生じ、偽の低い周波数が観察される。  
(例) 蛍光灯下の扇風機、テレビ画面のビデオ撮影

カットはアナログ回路によるローパスフィルタなどを用いる事が多い

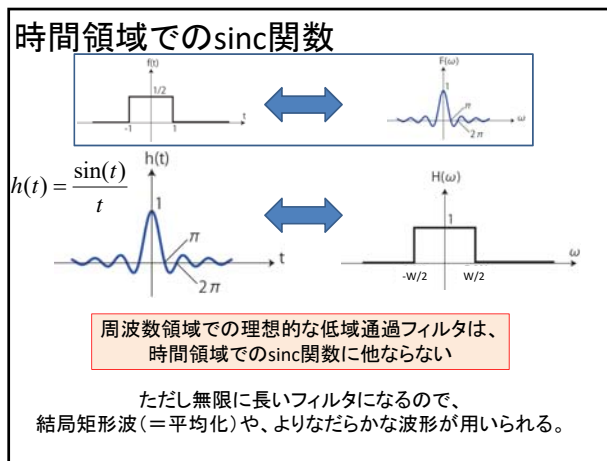
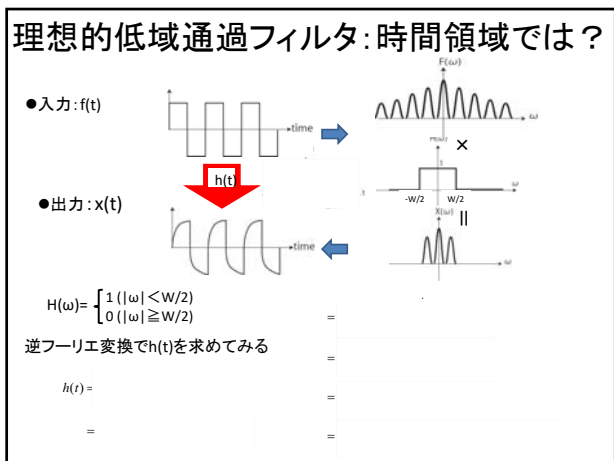
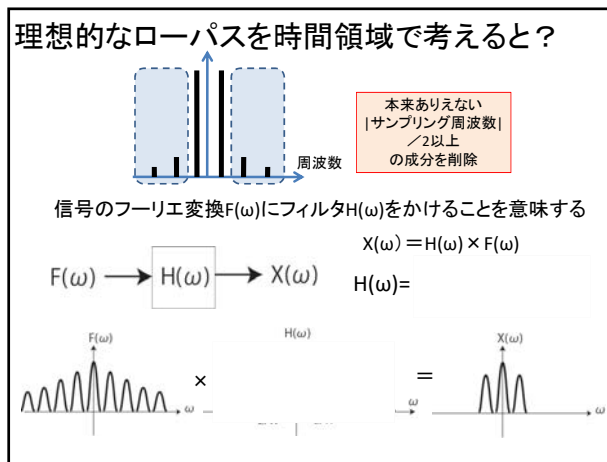
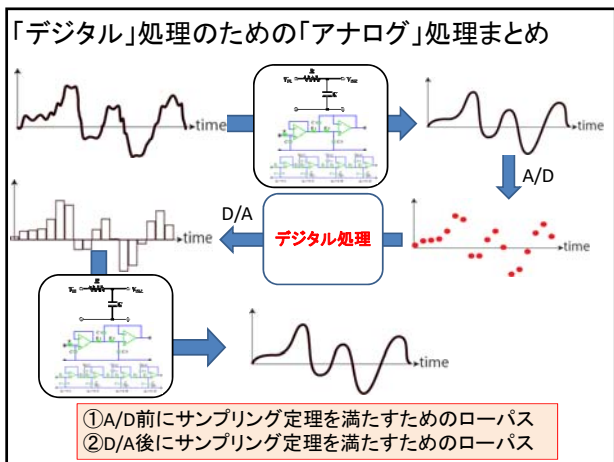
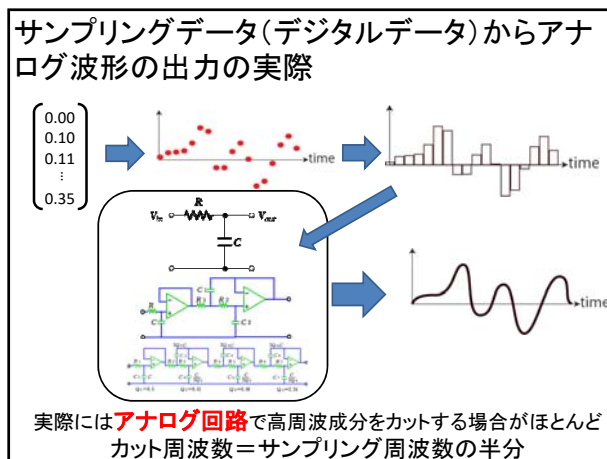
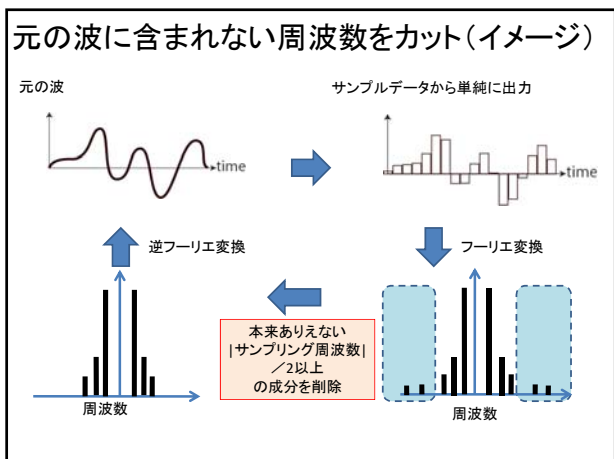
### サンプリングデータを元に戻す(再生)とは?

一番簡単な方法: サンプリングされたデータを、単純に電圧出力する(サンプル&ホールド)

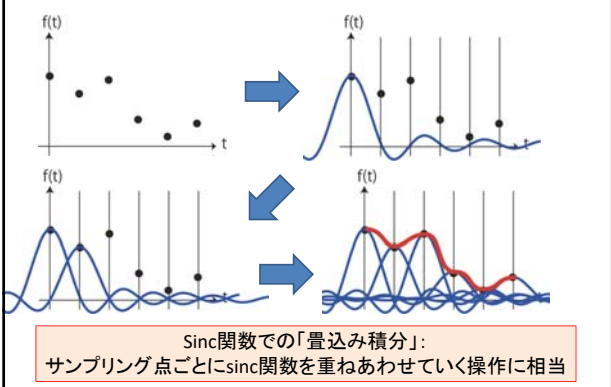
大体同じ。でも微妙な違い

元の波が、ナイキスト周波数未満の成分しかないとなると、単純に、「**高い周波数をカット**すれば元に戻る」はず

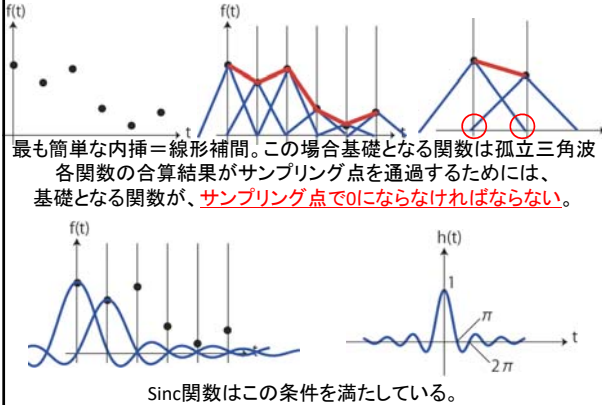




(参考) サンプルング点を「なだらかに内挿する」関数としてのsinc関数



(参考) サンプルング点を内挿する関数の条件



(参考) サンプルング定理ギリギリの波形も

