コンボリューション関数（conv）と相関関数(xcorr)と離散フーリエ変換関数（fft）の関係

コンボリューション積分

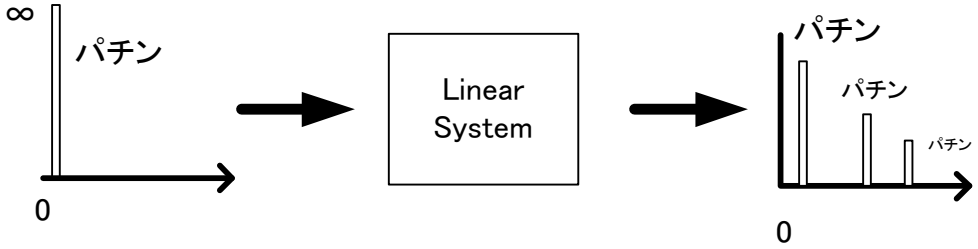
コンボリューション（たたみ込み）積分と言われ、信号処理、画像処理、のフィルタリング処理をするときに欠かすことのできない演算である。一般に、入力信号を u(t),システムのインパルス応答を h(t)、出力信号を x(t)をとして以下のように計算する。

$$x(t) = \int_0^t h(\tau)u(t-\tau)d\tau$$

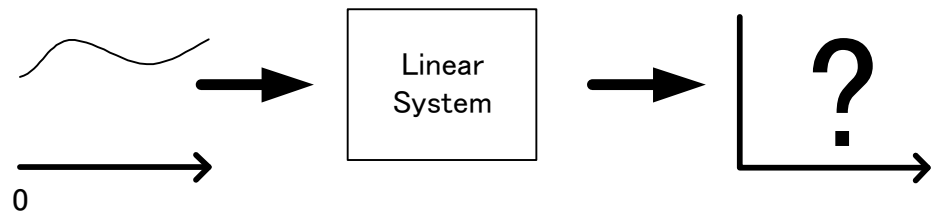
たたみ込み演算の由来

今、反響のある部屋で手をパチンと叩いたとしよう。するとあちこちの壁に反射し反響しあって耳に聞こえてくる。音の発生源から手を叩いた音が伝播し、その音の大きさ、伝播時間の異なる音が再合成され耳に聞こえてくる。もし、部屋の音響が線形ならば、手を叩いた場合でも、声を出した場合でもそれぞれに応じた音の大きさ、同じ伝播時間で音が合成され（たたみ込まれ）耳に聞こえてくるはずである。これを計算で表そうというのがたたみ込み演算である。

コンボリューションでのシステムの基本となる応答は、手をパチンと叩いたことによりできた音に対応する入力信号をインパルス応答と考えると次のようになる。

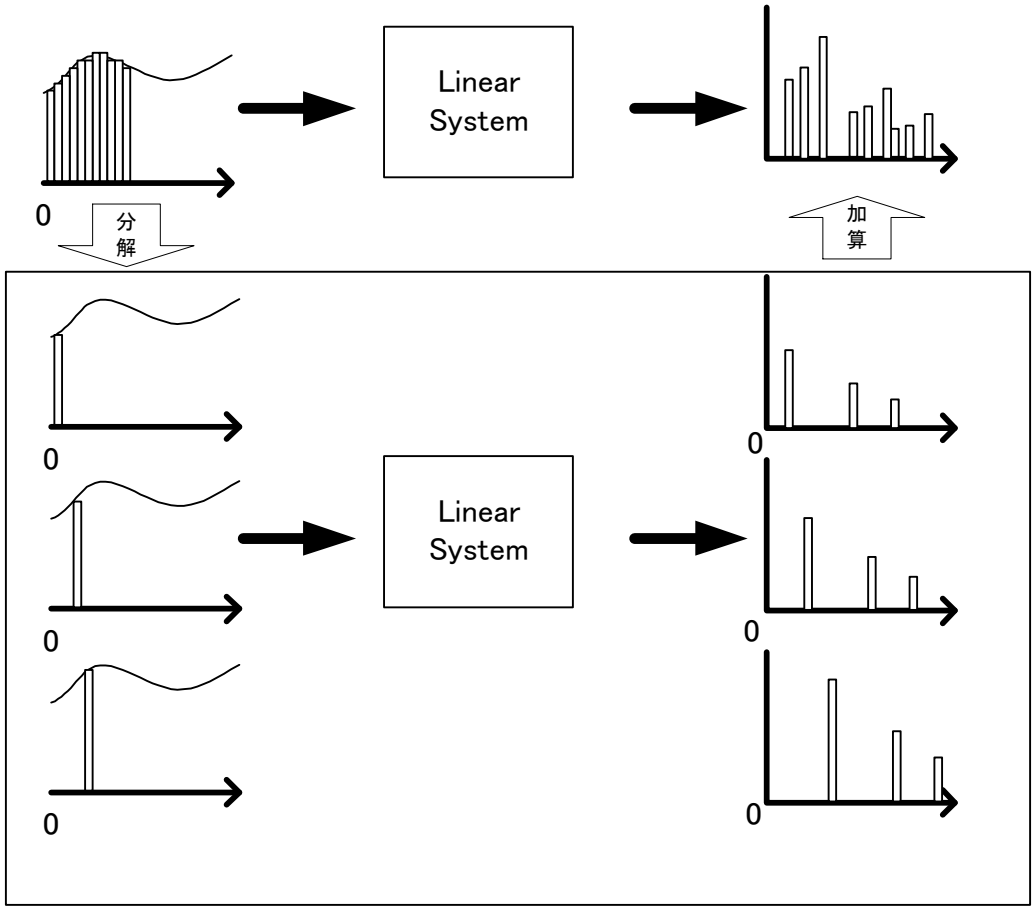


では、入力がインパルスではなく、連続信号であった場合はどのような応答になるのであろうか？



システムが線形の場合、連続信号を短冊状に切って、それぞれが独立した応答であると考えることができ、それらのインパルス応答が連続的に起きたものとして考えることができ

る。つまり



このようなイメージである。インパルス応答がたたみ込まれる形で出力応答が求められ、それら積分した結果が出力波形となるためたたみ込み積分と言われる。

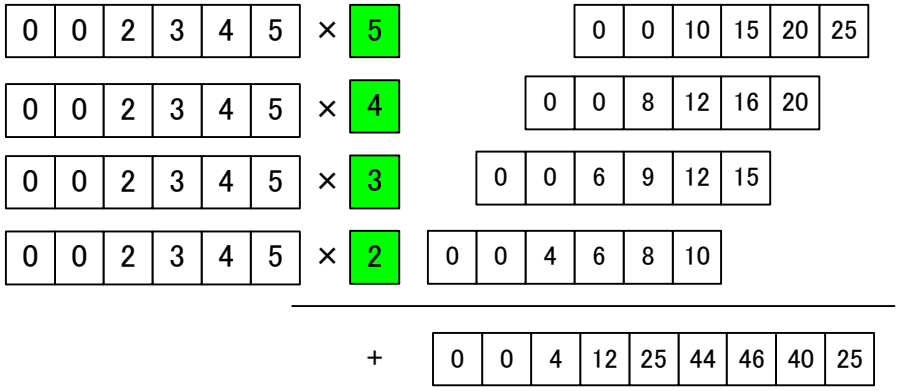
0.4 秒間遅らせた音を合成した場合の例

```
clear all
load train
%load handel
y=[y;y];
sound(y,Fs);
len=length(y);
delay=round(Fs*0.4);
llen = len - delay;
ind=1:llen;
yy=(y(ind)+y((ind)+delay))/2;
sound(yy,Fs)
```

今、MATLAB で使用しやすいように離散値表現で表すと、

$$x[n]=\sum_{m=0}^{\infty}h[m]\cdot u[n-m]=\sum_{m=0}^{\infty}u[m]\cdot h[n-m]$$

で定義できる。今、入力信号 u=[0,0,2,3,4,5]があり、線形システムのインパルス応答が h=[2,3,4,5]であった時のコンボリューション演算は以下のようになる。



まず、入力信号 x を 2 倍、3 倍、4 倍、5 倍の計算をし、それぞれのデータを時間ごと（ステップごと）にずらし最終的に足し合わせる。

MATLAB によるコンボリューションスクリプト例
ちなみに関数を使わないでこの計算をするスクリプト例を示す.

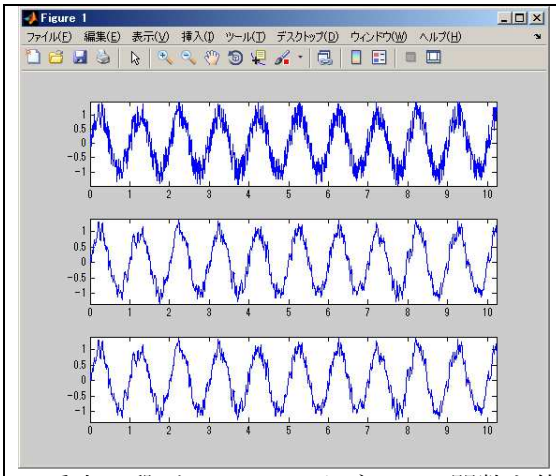
```
u=[0,0,2,3,4,5];h=[2,3,4,5];
res=zeros(1,length(u)+length(h)-1);
for x=1:length(h)
 res((1:length(u))+x-1)=res((1:length(u))+x-1)+u*h(x)
end
```

最初にデータ点数分だけの変数 res を確保して，for 文によりずらしながら足し算をしている.

コンボリューションによる 3 点データの平滑化例

この例については後で詳しく書いてあるが，ここでは，データ処理に利用するデータの平滑化処理にコンボリューションを使う例を示す.

```
close all;clear all
tau=0.01;
t=((1:1024)-1)*tau;
y=sin(2*pi*1*t);
y=y+(rand(size(y))-0.5)*1;
subplot(3,1,1);plot(t,y);axis tight
yy=([y,0,0]+[0,y,0]+[0,0,y])/3;
subplot(3,1,2);plot(t,yy(2:end-1));
axis tight
yyy=conv(y,[1/3,1/3,1/3],'same');
subplot(3,1,3);plot(t,yyy);axis tight
```



一番上の段が 1 Hz のノイズ(rand 関数を使ってノイズを作っている．そのままでは，平均 0 のノイズでないため，平均が 0 になるように 0.5 を差分している．)の乗った波形，2 番めは，前後 3 点のデータを使った平滑化スクリプト例，3 段目は，conv 関数を使った例，2 番目，3 番目の計算は，同じ値である．

MATLAB でコンボリューションを計算するには、conv 関数を使う。conv 関数を使った計算例は、以下のようになる。

```
>> u=[0,0,2,3,4,5];h=[2,3,4,5];
conv(u,h)
ans =    0     0     4    12    25    44    46    40    25
```

ちなみに、conv(u,h)でも conv(h,u)でも結果は変わりない。

コンボリューションでは、システムのインパルス応答 h(t)がわかった時、任意の入力信号 u(t)に対する応答 x(t)を計算する。この計算は、見方を変えてみると、入力信号に対するフィルタリングを行っていることに等しい。

コンボリューションによるフィルタリングは、FIR（Finite Impulse Response）ディジタルフィルタの一種である。DSP などでディジタルフィルタを実現する場合など、コンボリューション積分が重要な役割を果たす。

実は、

```
>> type conv
```

を実行してみるとわかるが、MATLAB での conv 関数の実装は、関数 M ファイルとして記述されており、ちなみに，MATLAB2012a では，関数ファイル内で 2 次元コンボリューション関数である conv2 関数を使ってコンボリューション演算を計算している。

コラム　conv 関数の用途　多項式の乗算　係数の計算

conv 関数は、たたみ込み積分（コンボリューション）を計算する以外にも、多項式乗算にも使える。

たとえば $(x+2)(2x+4)$ という多項式を展開すると $(x+2)(2x+4)=2x^2+8x+8$ となるが、conv 関数を使うと，

```
>> conv([1,2],[2,4])
ans =     2     8     8
```

と、x の乗数に合わせ、係数を計算してくれる。

たとえば、$(x+2)(2x+4)(3x+1)$ の場合には、conv 関数を 2 回使って

```
>> conv(conv([1,2],[2,4]),[3,1])
ans =     6    26    32     8
```

として計算できる。つまり結果は、

$(x+2)(2x+4)(3x+1)=6x^3+26x^2+32x+8$

となる。

フーリエ変換によるコンボリューション演算

コンボリューション演算が非常に重要であるポイントに、”時間領域での h(t),u(t)のコンボリューションは、周波数領域での H(iω)と U(iω)の各周波数成分の積として求められる。”という性質がある。つまり、

$$x(t)=\int_{-\infty}^{\infty}h(\tau)u(t-\tau)d\tau \Leftrightarrow X(i\omega)=H(i\omega)\cdot U(i\omega)$$

の関係がある。この”各周波数成分の積となる”というところに MATLAB 特有の乗算配列演算子を用いることができる。

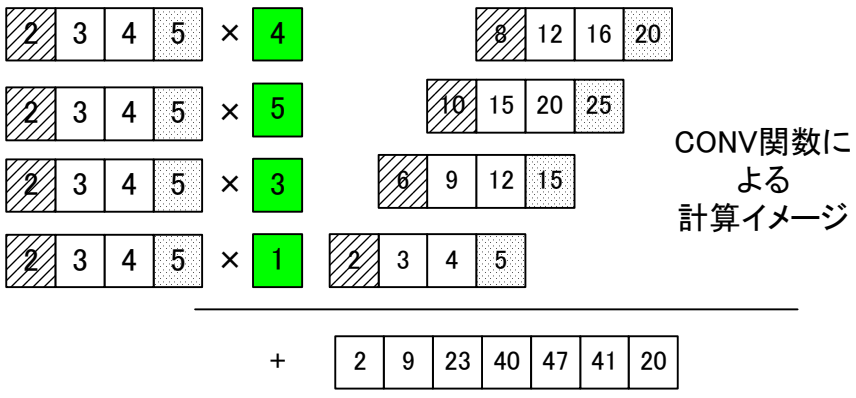
時間領域の離散データを周波数領域に変換する場合、MATLAB では FFT 関数を用いる。一般に FFT（Fast Fourier Transform)というと、2 の n 乗のデータ点数しか扱えないのが普通であるが MATLAB での FFT 関数の実装では、データ点数は任意でも点数に応じた高速な計算アルゴリズムが呼び出されるようになっているため、データの点数はあまり気にする必要はない。

FFT によるコンボリューション演算

conv 関数で行うコンボリューションは、非巡回たたみ込み積分、FFT で行うコンボリューションは、巡回たたみ込み積分と呼ばれ、実は、若干意味合いが異なる。これは、FFT で扱う信号は、周期関数データとして扱われる点にある。周期関数データとは、データの始点と終点は連続しており輪のような状態で繰り返されると仮定し処理している。

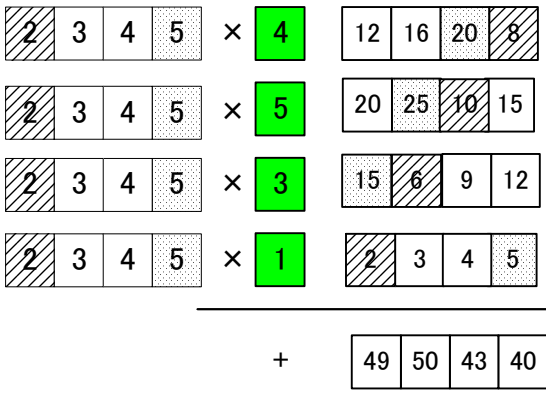
その違いを示すため、h=[1 3 5 4], u=[2 3 4 5]の h,u ともに同じ点数のコンボリューション例を示し、その違いを説明する。

conv 関数による方法では、




```
>> h=[1,3,5,4], u=[2,3,4,5];conv(u,h)
ans =     2     9    23    40    47    41    20
```

となっている。FFT による方法も同様の計算を行うのであるが、この場合、u のデータの最初と最後は巡回していると考えて計算される。つまり、



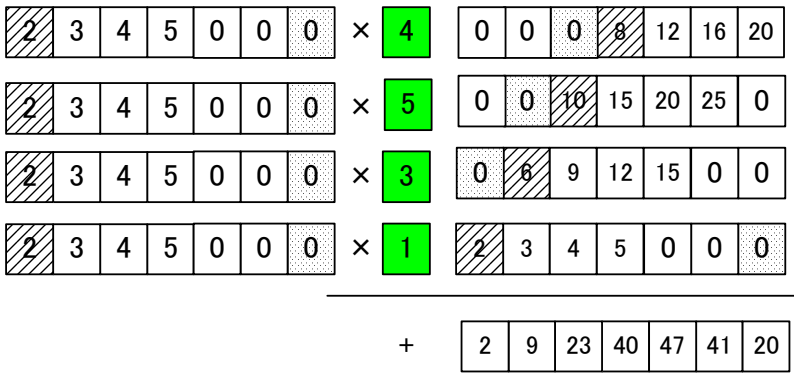
FFT関数による計算イメージ

シフトの代わりに巡回して繰り返すと考える

そして、この場合、データ点数が 4 点なのでデータが全ての演算でオーバーラップした 4 点分の数値の計算と一致する。逆フーリエ変換の結果には、値は小さいが虚数部に数値が生じることがある。これは、数値計算における誤差であり、ここでは、real 関数により実数部のみを抜き出している。

```
>>real(ifft(fft(h).*fft(u)))
ans =    49    50    43    40
```

この FFT による方法でのコンボリューションは、巡回型コンボリューションと言われる。コンボリューション計算の巡回の影響を除去し、conv 関数と同様の計算をするには、ベクトル h,u の長さがそれぞれ（h の長さ＋u の長さ-1）になるようにゼロを挿入し計算する。



FFT関数（巡回対策済み）による計算イメージ

シフトの代わりに巡回して繰り返すと考えるが繰り返される数字が0なので実質的な影響が出ず、直接法と同じ結果になる。

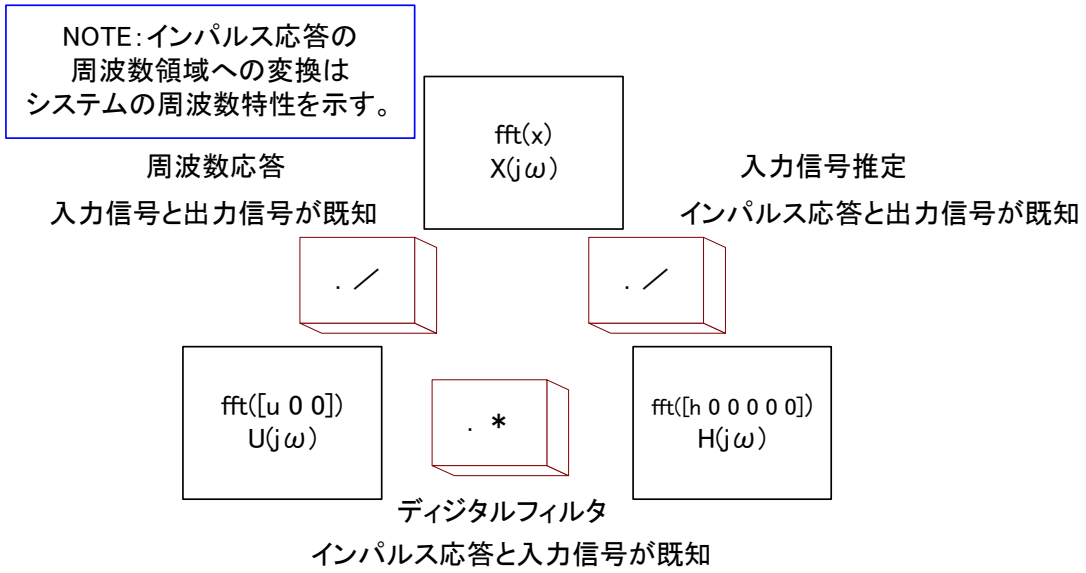
MATLAB での結果は、

```
>> h=[1,3,5,4]; u=[2,3,4,5];
lenh=length(h);lenu=length(u);
hh=zeros(1,lenh+lenu-1);hh(1:lenh)=h;
uu= zeros(1,lenh+lenu-1);uu(1:lenu)=u;
real(ifft(fft(uu).*fft(hh))),    %real(ifft(fft([u 0 0 0]).*fft([h 0 0 0])))
ans =    2.0000    9.0000   23.0000   40.0000   47.0000   41.0000   20.0000
```

となり、conv 関数の結果と一致する。

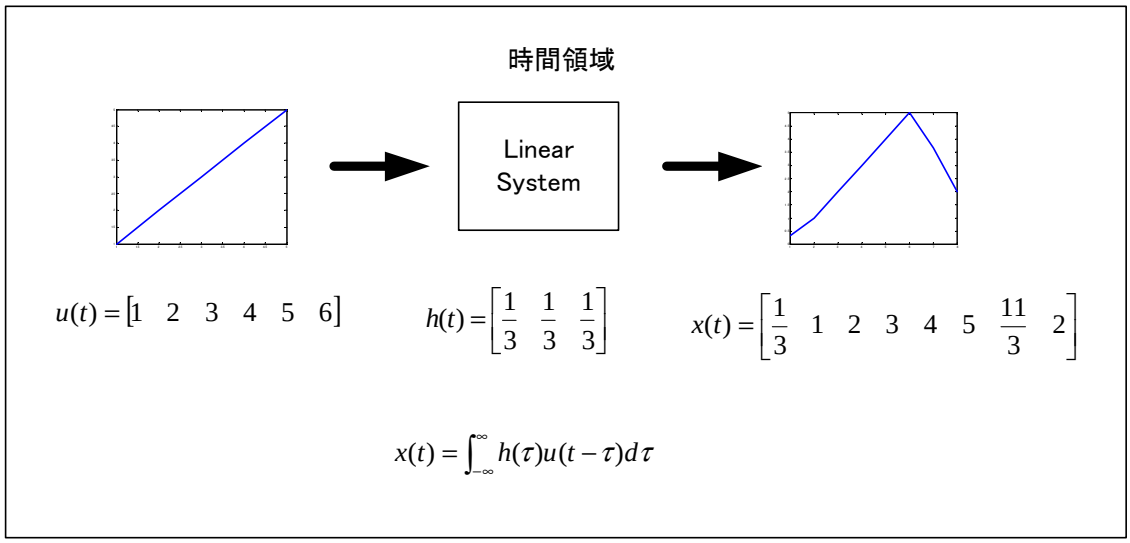
FFT を利用したコンボリューション演算応用

FFT を利用したコンボリューション演算は、データの巡回の問題があるが、関係式が乗算で表せるため入力、出力信号、インパルス応答波形の３つのコンビネーションで、どれか１つの信号がわからない場合、他の２つの信号波形がわかれば、計算で求めることができる。FFT を利用したコンボリューションを利用し、ディジタルフィルタ、周波数応答推定、信号推定などが容易に行うことができるなどその応用例は広い。また、計算速度の面でも時間領域では、繰り返し計算が多かったものが、配列の乗算で表現できることがから有利である。



FFT を用いたコンボリューション演算応用例として conv 関数で計算した応答が既知の前後３点スムージングフィルタを用いて計算してみよう。

前後 3 点の平均のスムージングフィルタのインパルス応答は、`h=[1/3,1/3,1/3];`
とし、そのシステムに入力信号 `u=[1,2,3,4,5,6];`を入れたとき `x=[1/3,1,2,3,4,5,11/3,2]`
となることは、確認済みである。つまり、



がわかっている。もしこれら信号を周波数領域に変換できるとこれらの積分関係は、乗除演算で記述することができる。まずこれらの関係を FFT 関数を使い表現する。FFT で周波数領域に変換するには、巡回の影響をなくすためと、配列の乗算を適用できるようにするためインパルス応答関数 h(t)と u(t)関数の長さを揃える。

```
u=[1,2,3,4,5,6];h=[1/3,1/3,1/3];
```

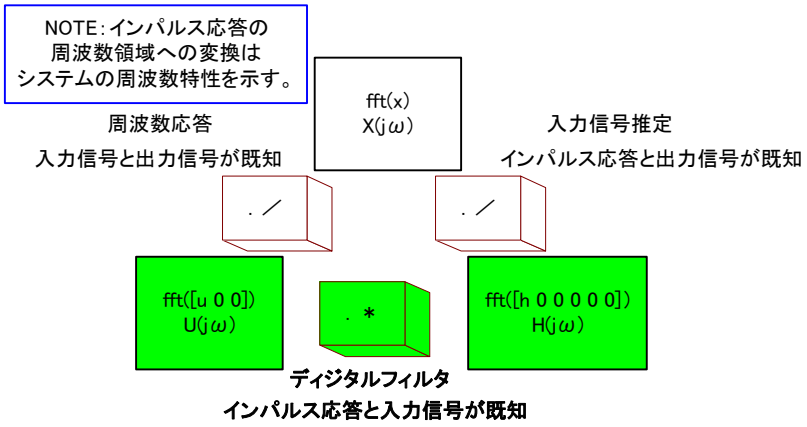
であったとすると入力信号 u,インパルス応答 h の長さは、いずれも length(u)+length(h)-1 の長さになるように 0 を挿入する。そこでここでは、zeros 関数により、すべてゼロの配列を作り、それに必要な値 h,u を代入する。

```
lenu=length(u);lenh=length(h);
uu=zeros(1,lenu+lenh-1);uu(1:lenu)=u;
hh=zeros(1,lenu+lenh-1);hh(1:lenh)=h;
hh
uu
hh =
   0.3333    0.3333    0.3333         0         0         0         0
0
uu =
    1     2     3     4     5     6     0     0
```

となる。

あとは、それぞれ U=fft(uu);H=fft(hh);X=fft(x);を計算すれば、配列の乗算、除算により様々な計算が可能になる。

(1)ディジタルフィルタ　Finite Impulse Response Filter

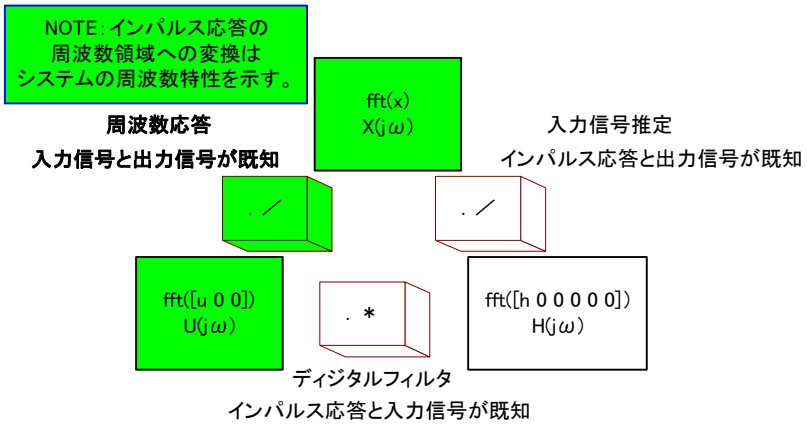


周波数領域での入力信号 U、それにシステムの伝達関数 H がわかれば、x を以下の式で計算することができる。

```
>> u=[1,2,3,4,5,6];lenu=length(u);
h=[1/3,1/3,1/3];lenh=length(h);
hh=zeros(1,lenu+lenh-1);hh(1:lenh)=h;
uu=zeros(1,lenu+lenh-1);uu(1:lenu)=u;
U=fft(uu);H=fft(hh);
real(ifft(U.*H))
x=[1/3,1,2,3,4,5,11/3,2]
ans =
  0.3333     1.0000     2.0000     3.0000     4.0000     5.0000     3.6667
2.0000
x =
  0.3333     1.0000     2.0000     3.0000     4.0000     5.0000     3.6667
2.0000
```

ちなみに、x と比較してみると、まったく等しい。

(2) インパルス応答推定（周波数特性推定、伝達関数推定）



周波数領域での入力信号 U と出力信号 X がわかれば、MATLAB の除算の配列演算子. ／を使うと

```
u=[1,2,3,4,5,6];lenu=length(u);
x=[1/3,1,2,3,4,5,11/3,2];lenx=length(x);
uu=zeros(1,lenx);uu(1:lenu)=u;
U=fft(uu);X=fft(x);
real(ifft(X./U))
```

```
h=[1/3,1/3,1/3]
ans =
    0.3333    0.3333    0.3333    0.0000   -0.0000         0         0
0.0000
h =
    0.3333    0.3333    0.3333
```

となる。h の後に 0 が挿入されているが、前３点の数値は同じである。

H=X./U の関係は、周波数領域での出力／入力であり、特にシステムの周波数特性関数といわれ、フィルタを設計するときなどに重要な関数となる。このことは、インパルス応答の周波数領域への変換は、システムの周波数応答を求めることと同じことを意味する。

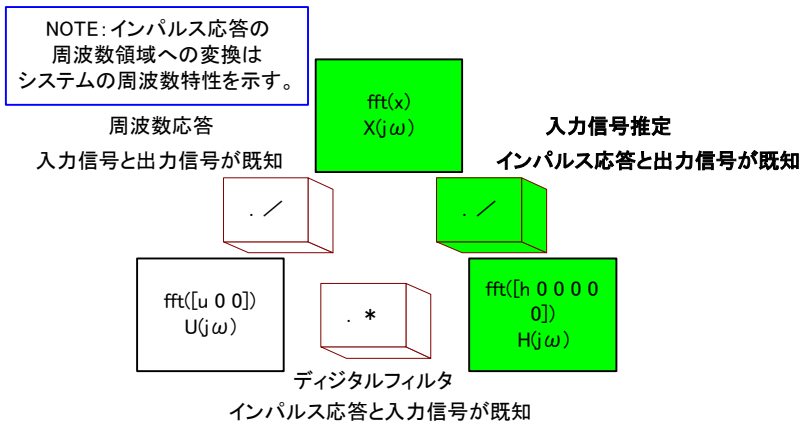
ちなみに、conv 関数の逆関数である、deconv 関数を使って計算すると、

```
>> u=[1,2,3,4,5,6]; x=[1/3,1,2,3,4,5,11/3,2];
>> [h,sa]=deconv(x,u)
h =    0.3333    0.3333    0.3333
sa =1.0e-15 *
         0         0         0    0.2220    0.2220         0    0.2220    0.4441
```

なお、変数 sa は、割り算で計算したときの余りを出している。この場合、10 の－15 乗となっており、ほぼ、0 であり、割り切れていることを示している。以下に手計算の例を示す。

										1/3	1/3	1/3	
1	2	3	4	5	6	1/3	1	2	3	4	5	11/3	2
					−	1/3	2/3	1	4/3	5/3	2		
						0	1/3	1	5/3	7/3	3	11/3	
						−	1/3	2/3	1	4/3	5/3	2	
							0	1/3	2/3	1	4/3	5/3	2
							−	1/3	2/3	1	4/3	5/3	2
								0	0	0	0	0	0

(3)入力信号の推定（デコンボリューション）



システムのインパルス時間応答関数 h(t)と出力時間関数 x(t)が判明している時に入力時間関数 u(t)を推定するのがディコンボリューションである。

これも MATLAB の除算の配列演算子を使って表すと、

```
h=[1/3,1/3,1/3];lenh=length(h);
```

```
x=[1/3,1,2,3,4,5,11/3,2];lenx=length(x);
hh=zeros(1,lenx);hh(1:lenh)=h;
H=fft(hh);X=fft(x);
real(ifft(X./H))
u=[1,2,3,4,5,6]
ans =
    1.0000     2.0000     3.0000     4.0000     5.0000     6.0000    -0.0000
-0.0000
u =
     1     2     3     4     5     6
```

一応、ここでは、しっかりと推定できているが、一般に、システムのインパルス応答関数は、周波数成分が少ないためゼロ割を起こしやすく推定精度が悪くなりがちである点に注意する必要がある。

一応、ここでは、しっかりと推定できているが、一般に、システムのインパルス応答関数は、周波数成分が少ないためゼロ割を起こしやすく推定精度が悪くなりがちである点に注意する必要がある。手計算では、つぎのようになる。

				1	2	3	4	5	6
1/3	1/3	1/3	1/3	1	2	3	4	5	11/3	2
		−	1/3	1/3	1/3				
			0	2/3	5/3	3			
			−	2/3	2/3	2/3			
				0	1	7/3	4		
				−	1	1	1		
					0	4/3	3	5	
					−	4/3	4/3	4/3	
						0	5/3	11/3	11/3
							5/3	5/3	5/3
							0	2	2	2
							−	2	2	2
								0	0	0

同様に deconv 関数を使うと、

```
>> x=[1/3,1,2,3,4,5,11/3,2];h=[1/3,1/3,1/3];
>> [u,~]=deconv(x,h)
u =     1     2     3     4     5     6
```

として計算できる。ちなみに、～は、第二出力変数を出したくないときに使う。

演習問題

5 点平均ディジタルフィルタを conv 関数，for 文による方法，fft 関数による方法で結果が同じになることを確認せよ.

hint:for 文を使った 5 点平均ディジタルフィルタ

```
t=(0:2047)*0.01;y=sin(2*pi*0.5*t);y=y+rand(size(t))-0.5;
subplot(2,1,1);plot(t,y)
for i=1:length(y)-4
  yy(i)=(y(i)+y(i+1)+y(i+2)+y(i+3)+y(i+4))/5;
end
subplot(2,1,2);plot(t(1:end-4),yy);
```

相関関数

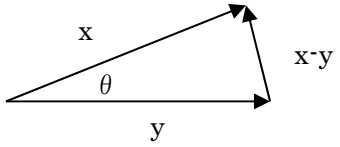
例えば、性質の似た時系列信号 x(t),y(t)があるとしよう。これら時系列データがお互いどれだけ似ているかを調べるには、どのようにしたらよいであろうか？これら信号がどの程度近いかは、２つの信号 x(t),y(t)の差の 2 乗の期待値で表すことができる。その平方根を x と y の距離といい$\|x-y\|$と書く。$\|x-y\|$の 2 乗は、

$$\|x-y\|^2 = E\left[(x(t)-y(t))^2\right] = E\left[x^2(t)\right] + E\left[y^2(t)\right] - 2E\left[x(t)y(t)\right]$$

$$\|x-y\|^2 = \|x\|^2 + \|y\|^2 - \frac{2E\left[x(t)y(t)\right]}{\|x\|\cdot\|y\|}\|x\|\|y\|$$

となる。ここで余弦定理を考えて信号 x,y をある空間のベクトルであると見なし解釈してみよう。余弦定理では、ベクトル x,y が角度 θ をなしていると考えると

$$\|x-y\|^2 = \|x\|^2 + \|y\|^2 - 2\cos\theta\cdot\|x\|\|y\|$$



が成り立つ。この式 2 つの式の対比より

$$\cos\theta \Leftrightarrow \frac{E\left[x(t)y(t)\right]}{\|x\|\cdot\|y\|}$$

の関係になる。つまり、それぞれ x,y ベクトルの示す角度がベクトルの似ている度合いを示すということと等価ということになる。この関係は、空間ベクトルだけでなく、２つの信号の期待値の間でも成り立つ。このような理由で以下の式で定義されたものが、

$$\phi_{XY} = E[x(t)y(t)]$$

相関関数（correlation function）である。相関関数は、一般に、信号の時間遅れ、むだ時間などの検出に使われる。そのため相関関数の引数に τ を用い、時間とともに変化する似ている度合いを表す。

$$\phi_{xx}(\tau) = E[x(t)x(t+\tau)] = \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} x(t)x(t+\tau)dt$$

自己相関関数では、時間 τ だけ遅れた信号ともとの信号との相関をとっているものである

ので、信号がその変動の中に隠れた周期変動成分を持っているならば、τ がその周期と一致しているときに相関が高くなるといった信号解析上有用な性質を持っている。
2 つの信号では、

$$\phi_{XY}(\tau) = E[x(t)y(t+\tau)] = \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} x(t)y(t+\tau)dt$$

を相互相関関数(Cross Correlation function)という。
定義によれば、T は±∞であるが実用上データ点数が N 点であるとし離散値で表現すると相互相関関数は、

$$\phi_{XY}[\tau] = \sum_{k=1}^{N} x[k]y[k+\tau]$$

となる。今、ベクトル変数 x,とベクトル変数 y を用いて、τ＝0 の相互相関関数$\phi_{XY}[0]$を求めてみよう。
定義より素直に for 文により素直にインプリメントしてみよう。例えば、サイン波 x、コサイン波 y の相関は、0(無相関)である。そのことを確かめてみよう。

```
x=sin(0:0.01:4*pi);y=cos(0:0.01:4*pi);
fai_xy=0;
for k=1:length(x)
fai_xy=fai_xy+x(k)*y(k);
end
fai_xy
```

結果は、

```
fai_xy =   -0.0012
```

となり、ほぼゼロである。C など他の言語で記述する場合には、このような形でインプリメントするのが普通である。
しかし MATLAB では、上の記述以外にも MATLAB 特有の配列の乗算演算子（. ＊）それに sum 関数を使い記述すると、for 文を省略し簡潔に表現できる。

```
x=sin(0:0.01:4*pi);y=cos(0:0.01:4*pi);
fai_xy=sum(x.*y);
```

結果は、同様に

```
fai_xy =   -0.0012
```

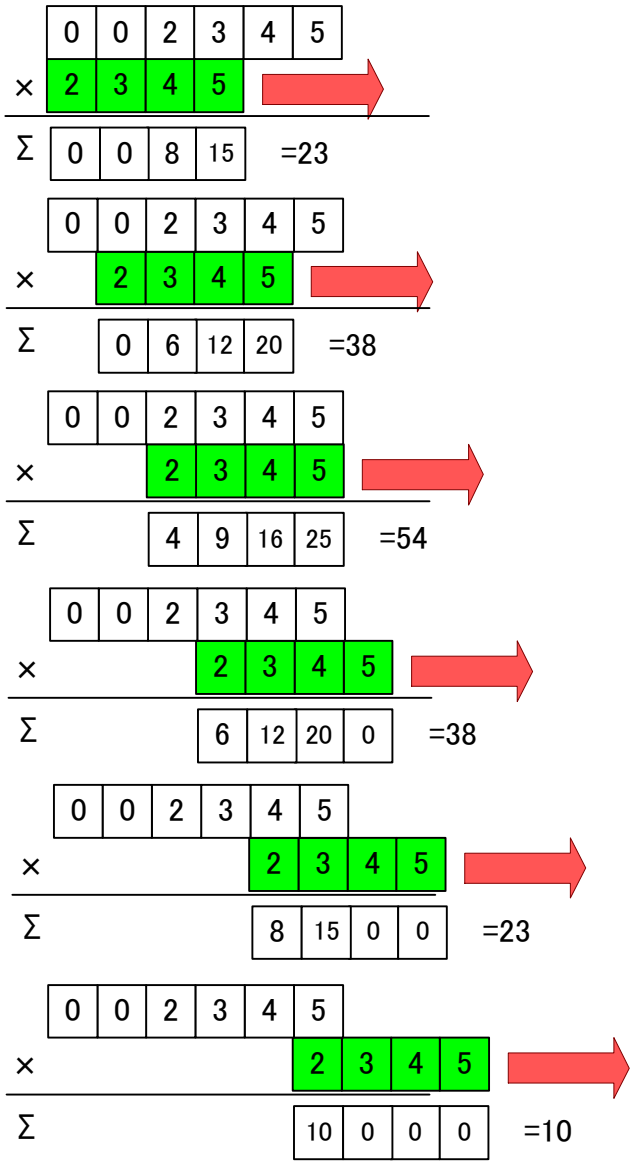
となる。この記述の方が、処理速度も速い上にほとんど定義の数式のまま表現できる。

相互相関

相互相関を計算する場合、通常 τ＝0 の相関値を計算するだけでなく、τ をいろいろ変えて計算し、最も相関が高い箇所を調べるなどのことを行う場合が多い。ここでは、

```
x=[0,0,2,3,4,5];y=[2,3,4,5];
```

このデータ x,y を使って計算をしてみよう。この場合、ずれが、3 の時、数値が一致するため相関値が最も高くなる。以下に計算のイメージを示す。



この流れを配列演算子、それに SUM 関数を活用したスクリプトで表したものを以下に示す。

```
x=[0,0,2,3,4,5];y=[2,3,4,5];
mxy=max(length(x),length(y));
xx=[x,zeros(1,length(y)-1)];
fai_xy=zeros(1, mxy);
scale=1; % sqrt(sum(x.*x))*sqrt(sum(y.*y));
for ii=1:mxy
 hani=(1:length(y))+ii-1;
fai_xy(ii)=sum(xx(hani).*y)/scale;
end;fai_xy

fai_xy =    23    38    54    38    23    10
```

この場合、FOR 文を使う必要が出てくるため、処理速度は、あまり速くはない。

実はこの計算は見方を変えると相関演算とコンボリューション演算とほぼ同様な形をしており、比較するデータの順序をひっくり返しコンボリューションをとったものと等しい。

$\phi_{XY}[\tau]=\sum_{k=1}^{N}x[k]y[k+\tau]$　　相関関数の定義

$x[n]=\sum_{m=-\infty}^{\infty}h[m]\cdot u[n-m]$　　コンボリューションの定義

符号が異なっている点に注目

（この例では、y=[2 3 4 5]の順序を入れ替えコンボリューションを計算している。）

			0	0	2	3	4	5
		×			5	4	3	2
			0	0	4	6	8	10
		0	0	6	9	12	15	
	0	0	8	12	16	20		
0	0	10	15	20	25			
0	0	10	23	38	54	38	23	10

つまり、conv 関数を使って計算すると、

```
x=[0,0,2,3,4,5];y=[2,3,4,5];
xycorr=conv(x, fliplr(y))
```

なおこの時に使う fliplr 関数は、ベクトル又は行列のデータがあった時その左右の順序を逆にする関数である。

```
>> conv([0,0,2,3,4,5],[5,4,3,2])
ans =0     0    10    23    38    54    38    23    10
```

なおこの時４番目のデータから最後までのデータが同じ数値となる。

xcorr 関数

MATLAB の Signal Processing Toolbox には、相互相関値を計算する関数として xcorr 関数がある。この関数は、上述したむだ時間 τ=0 の $R_{XY}[0]$ を計算するだけでなく、データ長に応じたむだ時間ベクトルデータを出力する。

むだ時間が短い場合（例えば、むだ時間０の自己相関などを計算する場合には）には、相互相関で示した、式どおりの直接法の方が効率が良い。しかしむだ時間 τ を連続的に計算したい場合は、FFT による方法の方が処理速度の面で有利である。

相関係数の計算法には、直接法、コンボリューションによる方法、FFT による方法があるが、xcorr 関数での相関係数の計算には、FFT を利用した方法を使用している。FFT による方法では、相関値の計算を直接時間軸で行うのではなく、一旦、周波数領域に分解、それから周波数ごとに計算、一方のデータは複素共役をとり乗算する。これは、時間領域でのコンボリューション演算そのものである。そして周波数ごとのずれ（角度に相当する）を計算、その後、逆 FFT を施し時間軸のデータへ変換している。

$\phi_{XY}=IFFT\left(FFT(x(k)).*conj(FFT(y(k))\right)$

FFT 法では、X,Y のデータ点数を N 点とすると計算される相関関数ベクトルは、むだ時間 -N+1～N+1 までのむだ時間（点数）のずれが計算される。FFT で扱う信号は、周期関数デ

ータとして扱われるため（つまり、データの始点と終点は連続しており輪のような状態になっていると考える）FFT 法で計算される相関係数は、巡回的相関係数と呼ばれる。
そのため、そのまま計算した場合には、直接法と比べて結果が異なってしまう。
そこで、xcorr 関数では、これら巡回的影響をなくすため、X,Y のデータの大きさに応じ 0 を挿入し、FFT による巡回的影響をなくした相互相関係数を計算している。
以下にそのまとめを示す。

xcorr(x,y)としたときの振る舞い

x ベクトルが長い場合 x=[2,3,4,5];y=[3,4,5];

xcorr 関数の例	直接法による例	FFT 法による例 巡回対策した場合	conv 関数による例 巡回対策はいらない。

fai_xy=…
xcorr(x,y)
	xx=[0,0,0,x,0,0,0];
	mxy=max(length(x),length(y));
	fai_xy=zeros(1,(2*mxy-1));
	for ii=1:(2*mxy-1);
	hani=(1:length(y))+ii-1;
	fai_xy(ii)=sum(xx(hani).*y);
	end;fai_xy
		xx=[0,0,0,x];
		%[0,0,0,2,3,4,5]
		yy=[y,0,0,0,0];
		%[3,4,5,0,0,0,0]
		fai_xy=...
		real(ifft(fft(xx).
		*conj(fft(yy))))
			yy=fliplr([y,0]);
			fai_xy=conv(x,yy)

fai_xy =-0.00 10.00 23.00 38.00 50.00 32.00 15.00			

y ベクトルが長い時 x=[2,3,4];y=[3,4,5,7];

xcorr 関数の例	直接法による例	FFT 法による例巡回対策した場合	conv 関数による例 巡回対策はいらない。

fai_xy=…
xcorr(x,y)
	xx=[0,0,0,x,0,0,0,0];
	mxy=max(length(x),length(y));
	fai_xy=zeros(1,(2*mxy-1));
	for ii=1:(2*mxy-1);
	hani=(1:length(y))+ii-1;
	fai_xy(ii)=sum(xx(hani).*y);
	end;fai_xy
		xx=[0,0,0,x,0];
		yy=[y,0,0,0];
		fai_xy=real(ifft(f
		ft(xx).*conj(fft(y
		y))))
			xx=[x,0];
			yy=fliplr(y);
			fai_xy=conv(xx,yy)

fai_xy = 14.00 31.00 51.00 38.00 25.00 12.00 0.00			

X,Y ベクトルが同じ場合 x=[2,3,4,5];y=[3,4,5,7];

xcorr 関数の例	直接法による例	FFT 法による例 巡回対策した場合

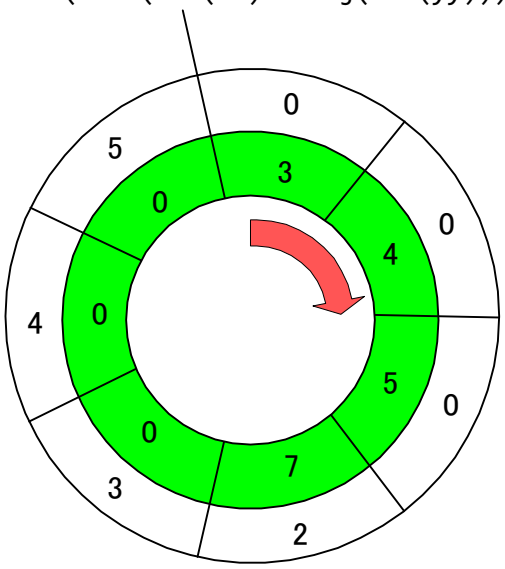
```
x=[2,3,4,5];
y=[3,4,5,7];
fai_xy=...
xcorr(x,y)
```

```
x=[2,3,4,5];y=[3,4,5,7];
xx=[0,0,0,x,0,0,0];
mxy=max(length(x),length(y));
fai_xy=zeros(1,(2*mxy-1));
for ii=1:(2*mxy-1);
 hani=(1:length(y))+ii-1;
fai_xy(ii)=sum(xx(hani).*y);
end;fai_xy
```

0	0	0	2	3	4	5	0	0	0

3	4	5	7

```
x=[2,3,4,5];y=[3,4,5,7];
xx=[0,0,0,x];%[0,0,0,2,3,4,5]
yy=[y,0,0,0];%[3,4,5,7,0,0,0]
fai_xy=...
real(ifft(fft(xx).*conj(fft(yy))))
```




```
fai_xy = 14.00 31.00 51.00 73.00 50.00 32.00 15.00
```

いずれの場合も、x 配列の前にゼロを付加し、y 配列を 1 つづつずらして計算して行く。xy 配列の長い方の長さを N とすると、それぞれの計算されるむだ時間との対応は、

$R_{XY}[-N+1]$	→fai_xy(1)
$R_{XY}[0]$	→fai_xy(N)
$R_{XY}[+N-1]$	→fai_xy(end)

となる。conv 関数による方法では、

```
>> x=[1,2,4,5,6,7,8,8,9];y=[7,8,8];
xcorr_xy=xcorr(x,y)
fai_xy=conv(x,fliplr(y))
xcorr_xy =
   -0.0000    0.0000   -0.0000         0    0.0000    0.0000    8.0000
24.0000   55.0000   86.0000  116.0000  139.0000  162.0000  177.0000
192.0000  128.0000   63.0000
fai_xy =
     8    24    55    86   116   139   162   177   192   128    63
```

で計算することができる。しかしそのままの計算では、conv 関数による場合と xcorr 関数による場合で計算結果の長さが異なってしまう。これは、xcorr 関数では、計算する際、2 つのデータの長さをそろえるため、配列の後にゼロを挿入しているためである。このゼロの挿入は、fft 関数による**コンボリューションの実現のための巡回による影響の対策とは異なり、むだ時間 0 の位置を真ん中にするための処理である点（$R_{XY}[0]$→fai_xy(N)とするため）に注意する。つまり、x と y のずれの差は、N= 9 で 15 番目がピークとなっているので、15-9=6 がむだ時間に相当する。**

つまり、相関の計算は、xcorr 関数の場合

```
>> x=[1,2,4,5,6,7,8,8,9];y=[7,8,8];xcorr(x,y)
ans =
   -0.0000    0.0000   -0.0000         0    0.0000    0.0000    8.0000
24.0000   55.0000   86.0000  116.0000  139.0000  162.0000  177.0000
192.0000  128.0000   63.0000
```

であるが、同様の計算を conv 関数で実現するには、

```
>> x=[1,2,4,5,6,7,8,8,9];y=[7,8,8];
lenx=length(x);leny=length(y);lenxy=max(lenx,leny);
xx=zeros(1,lenxy);xx(1:lenx)=x;
yy=zeros(1,lenxy);yy(1:leny)=y;
conv(xx,fliplr(yy))
ans =
     0     0     0     0     0     0     8    24    55    86   116   139   162
177   192   128    63
```

また、同様に fft 関数で実現するには、巡回対策も考慮し、

```
>>x=[1,2,4,5,6,7,8,8,9];y=[7,8,8];
lenx=length(x);leny=length(y);lenxy=max(lenx,leny);
xx=zeros(1,lenxy);xx(1:lenx)=x;
yy=zeros(1,lenxy);yy(1:leny)=y;
xxx=zeros(1,lenxy*2-1);xxx((1:lenxy)+lenxy-1)=xx;
yyy=zeros(1,lenxy*2-1);yyy(1:lenxy)=yy;
real(ifft(fft(xxx).*conj(fft(yyy))))
ans =
         0         0         0         0    0.0000         0    8.0000
24.0000   55.0000   86.0000  116.0000  139.0000  162.0000  177.0000
192.0000  128.0000   63.0000
```

として計算する。巡回対策の際に 0 を挿入する位置も重要である。

このようにすることで、むだ時間 0 の相関値は、インデックスが lenxy のところを参照すればよい。これらの式より、xcorr 関数、conv 関数による方法、fft 関数による方法、直接法いづれでも同様に似ている度合いがわかるわけであるが、上の式では、対象とする信号の大きさにより相関値も変化するため、似ている度合いの大きさの意味合いがはっきりしない。そこで最も似ている場合を 1、最も似ていない場合を-1 とする、正規化相互相関関数が定義されている。正規化相互相関関数は、

$$R_{XY}(\tau)=\frac{\phi_{XY}(\tau)}{\sqrt{\phi_{XX}(0)}\sqrt{\phi_{YY}(0)}}$$

として計算できる。この場合の計算は、ϕ_{XX} ,ϕ_{YY} ,,ϕ_{XY} に分けて計算する。

$\phi_{XX}(0),\phi_{YY}(0)$ の計算は、xcorr 関数を使った場合には、それぞれ、

```
x=[2 3 4 5];fai_xx=xcorr(x,x)
y=[3 4 5 7];fai_yy=xcorr(y,y)
fai_xx =   10.0000   23.0000   38.0000   54.0000   38.0000   23.0000
10.0000
fai_yy =    21    43    67    99    67    43    21
```

として計算でき、0 に対応するところは、N=4 なので、

```
>> [fai_xx(4),fai_yy(4)]
ans =    54    99
```

なので、正規化相互相関関数は、

```
>> xcorr(x,y)/(sqrt(fai_xx(4))*sqrt(fai_yy(4)))
ans =    0.1915    0.4240    0.6975    0.9984    0.6838    0.4377    0.2052
```

として計算できる。ちなみに自己相関関数をすべて計算する必要はないので、以下のように計算してもよい。

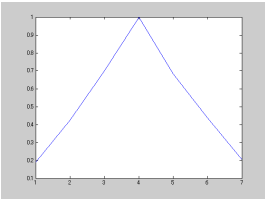
```
>> xcorr(x,y)/sqrt(sum(x.*x))/sqrt(sum(y.*y))
ans =    0.1915    0.4240    0.6975    0.9984    0.6838    0.4377    0.2052
```

MATLAB の xcorr 関数では、coeff オプションがあり、このオプションをつけると正規化相互相関関数が計算できる。（ただし、このオプションは、x,y ベクトルの長さが同じである必要がある。）

```
x=[2,3,4,5];y=[3,4,5,7];
fai_xy=xcorr(x,y,'coeff')
plot(fai_xy);
```

結果は、

```
fai_xy =
    0.1915    0.4240    0.6975    0.9984    0.6838    0.4377    0.2052
```



となり、４番目つまりむだ時間０の時、それぞれの自己相関値が最大の１としてノーマライズして計算される。

また、出力にもうひとつ変数 lag を設け以下のように計算、FIND 関数によりむだ時間 0 の場合の配列番号を調べれば、

```
[fai_xy,lag]=xcorr(x,y,'coeff');
fai_xy(find(lag==0))
ans =    0.9984
```

として計算できる。

MATLAB での xcorr 関数による相関計算では、負の時間の相関も計算していることもあり効率が悪く遅いように思える。しかし、長い時系列データを実際に使用してみると、for 文を使用する直接法に比べると相関法の方が圧倒的に処理速度が速い。

corrcoef 関数

MATLAB では、相関係数を計算する関数として corrcoef 関数もある。ちなみにこの関数で先ほどと同じ計算をしてみると、

```
x=[2,3,4,5];y=[3,4,5,7];
corrcoef(x,y)
ans =
    1.0000    0.9827
    0.9827    1.0000
```

となる。corrcoeff 関数は、相関関数行列を計算するため、2x2 の行列形式で出力される。この場合 corrcoeff 関数では、 (1,1)と(2,2)に自己相関係数が(1,2),(2,1)にむだ時間 0 の相互相関係数が出力される。自己相関係数の部分は 1 になっていることからわかるように、正規化相互相関係数が計算される。xcorr 関数は、オフセット成分も考慮した相関係数を計算しているため、それぞれのデータの平均値の違いも相関係数の計算に影響されるが、corrcoef 関数では、データの平均値の影響つまり、オフセット成分の影響は受けない。そのため、xcorr 関数で同様の計算をするためには、それぞれのオフセット成分（つまり平均値）を差し引いて計算すれば、同じ値となる。

```
fai_xy=xcorr(x-mean(x),y-mean(y), 'coeff');
fai_xy(length(x))
ans =
    0.9827
```

このように計算すると同様の値が計算できる。corrcoef 関数では、相関係数は、むだ時間 0 の相関係数しか計算しないが、相関行列として計算するため複数データの相関係数を計算することもできる。例えば、

```
>> corrcoef([x;y;y]')
ans =
    1.0000    0.9827    0.9827
    0.9827    1.0000    1.0000
    0.9827    1.0000    1.0000
```

となり、それぞれのデータについての相関を計算することができる。

コンボリューション・デコンボリューション・相互相関の計算例とまとめ

conv 関数、fft 関数の応用例と対応関係

conv 関数	u=[2,3,4,5];lenu=length(u);
	h=[1,3,5,4];lenh=length(h);
	% fft による巡回対策
	lenuh=lenu+lenh-1;
	uu=zeros(1,lenuh);uu(1:lenu)=u;hh=zeros(1,lenuh);hh(1:lenh)=h;
	x=[2,9,23,40,47,41,20];	

コンボリューション	conv(u,h) conv(uu,hh)	ans = 2 9 23 40 47 41 20
	real(ifft(fft([u,0,0,0]).*fft([h,0,0,0]))) real(ifft(fft(uu).*fft(hh)))	ans = 2.0000 9.0000 23.0000 40.0000 47.0000 41.0000 20.0000

デコンボリューション	u=[2,3,4,5];h=[1,3,5,4];
	x=[2,9,23,40,47,41,20];
	[uu,~]=deconv(x,h)
		uu =
		2 3 4 5

	real(ifft(fft(x)./fft([h,0,0,0])))	ans = 2.0000 3.0000 4.0000 5.0000 0 0.0000 -0.0000
	[hh,~]=deconv(x,u)	hh = 1 3 5 4
	real(ifft(fft(x)./fft([u,0,0,0])))	ans = 1.0000 3.0000 5.0000 4.0000 -0.0000 0 0.0000

xcorr 関数	u=[2,3,4,5];lenu=length(u);
	h=[1,3,5,4];lenh=length(h);
	% むだ時間 0 の位置をデータの中央に、
	lenuh=max(lenu,lenh);
	uu=zeros(1,lenuh);uu(1:lenu)=u;
	hh=zeros(1,lenuh);hh(1:lenh)=h;
	% fft による巡回対策
	uuu1=zeros(1,lenuh*2-1);uuu1((1:lenuh)+lenuh-1)=uu;
	hhh1=zeros(1,lenuh*2-1);hhh1(1:lenuh)=hh;
	uuu2=zeros(1,lenuh*2-1);uuu2(1:lenuh)=uu;
	hhh2=zeros(1,lenuh*2-1);hhh2((1:lenuh)+lenuh-1)=hh;	

相関	xcorr(u,h)	ans = 8.0000 22.0000 37.0000 51.0000 40.0000 19.0000 5.0000
	conv(u,fliplr(h)) conv(uu,fliplr(hh))	ans = 8 22 37 51 40 19 5
	real(ifft(fft([0,0,0,u]).*conj(fft([h,0,0,0])))) real(ifft(fft(uuu1).*conj(fft(hhh1))))	ans = 8.0000 22.0000 37.0000 51.0000 40.0000 19.0000 5.0000
	real(ifft(fft([u,0,0,0]).*fft(fliplr([0,0,0,h])))) real(ifft(fft(uuu2).*fft(fliplr(hhh2)))) % or real(ifft(fft([u,0,0,0]).*fft(rot90([0,0,0,h],2)))) real(ifft(fft(uuu2).*fft(rot90(hhh2,2))))	ans = 8.0000 22.0000 37.0000 51.0000 40.0000 19.0000 5.0000

演習課題

課題１　ここで述べた以外のコンボリューション積分の具体的応用例について調べ記述せよ。

課題 2　y=sin(0.001*(0:2047)*50)+0.2*rand(1,2048);
のデータがあった時、5 点データによる平滑化フィルタプログラムを、直接法（conv 関数を使わない方法)、conv 関数を使う方法、fft 関数による方法の３パターンを作成し、それぞれ同じ結果になることを確認できるスクリプトを作成せよ。

Hint ４点の例

```
close all
t=0.001*(0:2047);
y=sin(t*50)+0.2*rand(1,2048);
yyy=([y,0,0,0]+[0,y,0,0]+[0,0,y,0]+[0,0,0,y])/4;
yyyy=conv(y,[1,1,1,1]/4,'same');
subplot(3,1,1);plot(t,y,t,yyy(3:end-1));title('direct');
subplot(3,1,2);plot(t,y,t,yyyy);title('conv');
subplot(3,1,3);plot(t,yyyy-yyy(3:end-1));title('error = conv - direct');
```

課題 3　MATLAB では、伝達関数と入力が与えられた場合の応答を求める関数として lsim 関数、また、インパルス応答を求める関数として、impulse 関数[注]がある。これら関数を応用し、以下のスクリプトを作成せよ。
時間を t=(0:2047)*0.01;とし考える。
(1)ディジタルフィルタ
システムのインパルス応答 h=impulse(tf(1,[1,1]),t);h=h/sum(h);と、入力 u=10*(rand(size(t))-0.5);がわかっているとして、コンボリューションにより出力 y 計算し、その結果が y=lsim(tf(1,[1,1]),u,t);にほぼ一致することを確認するスクリプト testhu2y.m を作成せよ。
(2)周波数特性推定（伝達関数推定、インパルス応答推定）
入力 u=10*(rand(size(t))-0.5);と出力 y=lsim(tf(1,[1,1]),u,t);がわかっているとした場合、コンボリューションによりそのシステムのインパルス応答 h を計算し、その結果が、h=impulse(tf(1,[1,1]),t); h=h/sum(h);にほぼ一致することを確認するスクリプト testuy2h.m を作成せよ。
(3)入力信号の推定（デコンボリュージョン）
システムのインパルス応答、h=impulse(tf(1,[1,1]),t); h=h/sum(h);と出力 y=lsim(tf(1,[1,1]),u,t);がわかっているとした場合、コンボリューションにより、入力 u を計算し、その結果が、u=10*(rand(size(t)-0.5);とほぼ一致することを確認するスクリプト testhy2u.m を作成せよ。

(4) t=(0:2047)*0.01;　相関関数　y1=rand(size(t));y2=y1(100:300);
の２つのデータあったときの相互相関を conv 関数を使った方法、fft 関数を使った方法、

xcorr 関数を使った方法それぞれについて計算し結果が一致することを示すスクリプト soukankeisan.m を作成し、そのデータの読み方（どこがむだ時間 0 に相当するか？）を解説せよ。

```
t=(0:2047)*0.01;y1=rand(size(t));y2=y1(100:300);
xycorr=xcorr(y1,y2);
xyconv=conv([y1],[zeros(1,2048-201),fliplr(y2)]);
xyfft=real(ifft(fft([zeros(1,2047),
y1]).*conj(fft([y2,zeros(size(y1)),zeros(1,2047-201)]))));
subplot(2,2,1);plot(xycorr)
subplot(2,2,2);plot(xyconv)
subplot(2,2,3);plot(xyfft)
```

注：
コンボリューション積分でのインパルス応答は、インパルス応答の積分値つまり面積を 1 にノーマライズする必要がある。
MATLAB のインパルス関数で計算したインパルス応答の積分値は、1 にはなっていない。そこで積分値が 1 になるように正規化する必要がある。正規化するには、以下のように、まずインパルス応答を計算し、積分値で割ってあげればよい。

```
h=impulse(tf(1,[1 1]),t);
h=h/sum(h);
```

定義上からもわかるように t はじゅうぶん長い時間をとらないと、上記の計算は成立しない点に注意すること。

testhu2y.m 入力とインパルス応答から出力を計算(フィルタリング)

```
t=(0:2047)*0.01;
h=impulse(tf(1,[1,1]),t);h=h/sum(h);
u=10*(rand(size(t))-0.5);
y=lsim(tf(1,[1,1]),u,t);
y0=conv(u,h);
subplot(3,1,1);plot(t,y);axis tight
subplot(3,1,2);
plot(t,y0(2:2049));axis tight
subplot(3,1,3);
plot(t,y-[0; y0(1:2047)]);axis tight
```

1 点分ずれるので注意

testuy2h.m　入力と出力からインパルス応答を推定

（伝達関数推定、周波数特性推定、インパルス応答推定）

```
t=(0:2047)*0.01;
h=impulse(tf(1,[1 1]),t);h=h/sum(h);
u=10*(rand(size(t))-0.5);
y=lsim(tf(1,[1 1]),u,t);
y2=[y',zeros(1,length(u)-1)];
u2=[u,zeros(1,length(y)-1)];
h0=real(ifft(fft(y2)./fft(u2)));
h0=h0/sum(h0);
plot(t,h,t,h0(1:2048));
```

testhy2u.m

```
t=(0:2047)*0.01;
h=impulse(tf(1,[1 1]),t);h=h/sum(h);
u=10*(rand(size(t))-0.5);
y=lsim(tf(1,[1 1]),u,t);
y2=[y',zeros(1,length(h)-1)];
h2=[h',zeros(1,length(y)-1)];
u0=real(ifft(fft(y2)./fft(h2)));
plot(t,u,t(1:2047),u0(2:2048));
xlim([0 1])
```

soukankeisan.m

```
t = (0:2047)*0.01;
y1 = rand(size(t));y2 = y1(100:300);
yy1=[zeros(1,length(y1)-1) y1];
yy2=[y2 zeros(1,2*length(y1)-length(y2)-1)];
phi_direct=zeros(1,(2*length(y1)-1));
scale = 1;
for ii=1:(2*length(y1)-1-length(y2));
        hani = (1:length(y2))+ii-1;
        phi_direct(ii) = sum(yy1(hani).*y2)/scale;
end
phi_conv=conv(y1,fliplr([y2 zeros(1,length(y1)-length(y2))]));
phi_fft=real(ifft(fft(yy1).*conj(fft(yy2))));
phi_xcorr=xcorr(y1,y2);
[d_val,d_ind]=max(phi_direct(2048:end));
```

```
[c_val,c_ind]=max(phi_conv(2048:end));
[f_val,f_ind]=max(phi_fft(2048:end));
[x_val,x_ind]=max(phi_xcorr(2048:end));
subplot(4,1,1);
plot(t,phi_direct(2048:end),t(d_ind),d_val,'p');
subplot(4,1,2);
plot(t,phi_conv(2048:end),t(c_ind),c_val,'p');
subplot(4,1,3);
plot(t,phi_fft(2048:end),t(f_ind),f_val,'p');
subplot(4,1,4);
plot(t,phi_xcorr(2048:end),t(x_ind),x_val,'p');
[d_ind,c_ind,f_ind,x_ind]
```

コンボリューション関数の２次元への拡張

コンボリューション演算は、時系列データだけでなく、画像データなどでもよく利用される。MATLAB では、conv 関数の２次元への拡張として conv2 関数が用意されている。

```
data1=[1,2,3;4,5,6;7,8,9];
data2=[2,3;5,6];
>> conv2(data1,data2)
ans =
     2     7    12     9
    13    38    54    36
    34    86   102    63
    35    82    93    54
```

conv2 関数では、data2 を 180 度回転させたものつまり rot90(data2,2)と処理した行列との乗算によりコンボリューションを計算している。

$$\begin{bmatrix} 6 & 5 \\ 3 & \mathbf{2} \end{bmatrix}\begin{bmatrix} \mathbf{1} & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}=2$$

$$\begin{bmatrix} 6 & 5 \\ \mathbf{3} & \mathbf{2} \end{bmatrix}\begin{bmatrix} \mathbf{1} & \mathbf{2} & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}=1*3+2*2=7$$

$$\begin{bmatrix} 6 & 5 \\ \mathbf{3} & \mathbf{2} \end{bmatrix}\begin{bmatrix} 1 & \mathbf{2} & \mathbf{3} \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}=2*3+2*3=12$$

・・・

１次元のバージョンの自然な拡張となっている。今、3x3 のデータを使って試してみると、

```
data1=[1,2,3;4,5,6;7,8,9];
data2=[2,3;5,6];
[row1,col1]=size(data1);[row2,col2]=size(data2);
conv2(data1,data2)
ans =
     2     7    12     9
    13    38    54    36
    34    86   102    63
    35    82    93    54
```

結果は、(row1+row2-1)x(col1+col2-1)の行列として出力される。

ちなみに、fft 関数の２次元拡張版である fft2 関数を使い１次元のときと同様のゼロを挿入し計算してみると、

```
data1=[1,2,3;4,5,6;7,8,9];
data2=[2,3;5,6];
[row1,col1]=size(data1);[row2,col2]=size(data2);
data11=zeros(row1+row2-1,col1+col2-1);data11(1:row1,1:col1)=data1;
data22=zeros(row1+row2-1,col1+col2-1);data22(1:row2,1:col2)=data2;
real(ifft2(fft2(data11).*fft2(data22)))
```

```
ans =
     2     7    12     9
    13    38    54    36
    34    86   102    63
    35    82    93    54
```

として計算することができる。ちなみに、fft２関数では、ゼロを挿入する行列を作成しなくても fft２関数の引数として、サイズを指定すると、空いた箇所にゼロを挿入し計算してくれる。

```
data1=[1,2,3;4,5,6;7,8,9];
data2=[2,3;5,6];
[row1,col1]=size(data1);[row2,col2]=size(data2);
real(ifft2(fft2(data1,row1+row2-1,col1+col2-1).*fft2(data2,row1+row2-1,col1+col2-1)))
ans =
     2     7    12     9
    13    38    54    36
    34    86   102    63
    35    82    93    54
```

また、相関関数についても xcorr 関数の２次元版として xcorr2 関数がある。同様に２次元相互相関について計算してみる。ちなみに、**xcorr 関数では、データ点数が異なる場合の実装は、むだ時間０に対応するデータが中央になるようにゼロを挿入していたのに対し、xcorr2 関数では、そのような配慮が無い。（１次元バージョンでは負の時間も考慮していたが、２次元では必要ないため？）** そこでここでも xcorr2 関数に準拠した形で記述してみると、

```
data1=[1,2,3;4,5,6;7,8,9];
data2=data1(2:3,2:3);
xcorr2val=xcorr2(data1,data2)
[row1,col1]=size(data1);[row2,col2]=size(data2);
data11=zeros(row1+row2-1,col1+col2-1);data11(1:row1,1:col1)=data1;
data22=zeros(row1+row2-1,col1+col2-1);data22(1:row2,1:col2)=rot90(data2,2);
%real(ifft2(fft2(data11).*fft2(rot90(data2,2),row1+row2-1,col1+col2-1)))
fftcorr2=real(ifft2(fft2(data11).*fft2(data22)))
conv2(data1,rot90(data2,2))
xcorr2val =
     9    26    43    24
    42    94   122    63
    87   178   206   102
    42    83    94    45
fftcorr2 =
     9    26    43    24
    42    94   122    63
    87   178   206   102
    42    83    94    45
```

として計算できる。この場合、rot90 関数がポイントである。これが１次元の場合の fliplr 関数に相当する。rot90 関数の第二引数の２は、90 度が２回、つまり、180 度の回転を表している。１次元のバージョンとは異なり、むだ時間０を表す数値が、データの中央にあるわけではないため、相関を計算する際その点を考慮する必要がある。なお、上の例題では、２０６となっているところが相関が最大となっている。data1 と対応できるようにするには、行列サイズを同じにすればよいわけであるが、ここでは、conv2 関数の’same’オ

プションに習い、中央のデータを取り出すように次のようにする。

```
>> fftcorr2(floor(row2/2)+(1:row1),floor(col2/2)+(1:col1))
ans =
    94   122    63
   178   206   102
    83    94    45
```

この場合２０６が最大値となっているわけであるが、data2 の左上のデータの位置がその対応する位置となることがわかる。

```
fftcorr2 =
     9    26    43    24
    42    94   122    63
    87   178   206   102
    42    83    94    45
>> fftcorr2(floor(row2/2)+(1:row1),floor(col2/2)+(1:col1))
ans =
    94   122    63
   178   206   102
    83    94    45
>> data1
data1 =
     1     2     3
     4     5     6
     7     8     9
>> data2
data2 =
     5     6
     8     9
```

fft２関数による２次元デコンボリューション

MATLAB では、deconv2 関数は存在しない（deconvreg 関数などがそれに相当。）が、fft2 関数を使うことで計算することができる。今、3x3 のデータを使って試してみると、

```
data1=[1,2,3;4,5,6;7,8,9];
data2=[5,6;5,3];
```

があったとき、data1 と data2 のコンボリューションは、

```
> data3=conv2(data1,data2)
data3 =
     5    16    27    18
    25    62    81    45
    55   119   138    72
    35    61    69    27
```

となる。fft2 関数で同様の計算をすると、

```
[row1,col1]=size(data1);
[row2,col2]=size(data2);
data11=zeros(row1+row2-1,col1+col2-1);
data22=zeros(row1+row2-1,col1+col2-1);
data11(1:row1,1:col1)=data1;
data22(1:row2,1:col2)=data2;
```

```
real(ifft2(fft2(data11).*fft2(data22)))
ans =
     5    16    27    18
    25    62    81    45
    55   119   138    72
    35    61    69    27
```

今、data1,data3 が機知のとき、fft2 関数を使い data2 を計算してみると

```
>> data11=zeros(size(data3));
[row1,col1]=size(data1);
data11(1:row1,1:col1)=data1;
real(ifft2(fft2(data3)./fft2(data11)))
ans =
    5.0000    6.0000         0         0
    5.0000    3.0000         0         0
         0         0         0         0
    0.0000   -0.0000    0.0000   -0.0000
```

また、data2,data3 が機知のとき data1 を計算してみると

```
[row2,col2]=size(data2);
data22=zeros(size(data3));
data22(1:row2,1:col2)=data2;
real(ifft2(fft2(data3)./fft2(data22)))
ans =
    1.0000    2.0000    3.0000         0
    4.0000    5.0000    6.0000         0
    7.0000    8.0000    9.0000         0
    0.0000         0    0.0000    0.0000
```

としてそれぞれ計算できる。

２次元版コンボリューション・デコンボリューション・相互相関の計算例とまとめ

	u=[1,2,3;4,5,6;7,8,9];
	h=[5,6;5,3];
	x=[5,16,27,18;25,62,81,45;55,119,138,72;35,61,69,27];
	[rowu,colu]=size(u);
	[rowh,colh]=size(h);
	rowuh=rowu+rowh-1;coluh=colu+colh-1;	

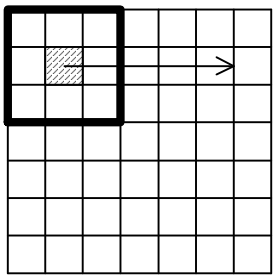
コンボリューション	conv2(u,h)	ans = 5 16 27 18 25 62 81 45 55 119 138 72 35 61 69 27

	real(ifft2(fft2(u,rowuh,coluh).*fft2(h,rowuh,coluh)))
	% or
	hh=zeros(rowuh,coluh);
	hh(1:rowh,1:colh)=h;
	uu=zeros(rowuh,coluh);
	uu(1:rowu,1:colu)=u;
	real(ifft2(fft2(hh).*fft2(uu)))
		ans =
		5 16 27 18
		25 62 81 45
		55 119 138 72
		35 61 69 27

デコンボリューション	real(ifft2(fft2(x)./fft2(u,rowuh,coluh)))	ans = 5.0000 6.0000 0 0 5.0000 3.0000 0 0 0 0 0 0 0.00 -0.00 0.000 -0.00
	real(ifft2(fft2(x)./fft2(h,rowuh,coluh)))	ans = 1.0000 2.0000 3.0000 0 4.0000 5.0000 6.0000 0 7.0000 8.0000 9.0000 0 0.000 0 0.000 0.000
相関	xcorr2(u,h)	ans = 3 11 19 15 18 52 71 45 45 109 128 75 42 83 94 45
	conv2(u,rot90(h,2))	ans = 3 11 19 15 18 52 71 45 45 109 128 75 42 83 94 45
	real(ifft2(fft2(u,rowuh,coluh).*fft2(rot90(h,2),rowuh,coluh)))	ans = 3 11 19 15 18 52 71 45 45 109 128 75 42 83 94 45

2 次元応用 conv2 関数を使ったフィルタリング

画像の微分や平滑化は、最も基本的な処理の 1 つである。一般的なフィルタリング法の一つに局所オペレータによるフィルタリング法がある。この方法を MATLAB でどのように効果的にプログラミングしていくかを考える。局所オペレータとは、あるピクセルとその周囲のピクセルに注目して行う処理のこと。注目するピクセルを順次ずらしながら、画像全体を走査していく。もっとも簡単な局所オペレータの一つに平滑化がある。ここでは、３ｘ３の平均値フィルタを例に解説していく。



$$newgazou(i,j)=\frac{1}{9}\sum_{k=i-1}^{i+1}\sum_{l=j-1}^{j+1}gazou(k,l)$$

一般に、C 言語などのアルゴリズム関連のテキストで画像処理をする例を見ると、for 文を用いているものが多い。C 言語で３×３の走査ウィンドウで平滑化を行う場合の例を示す。C 言語に限らず、他言語でも画像処理の場合、For 文繰り返し文が多用される。（画像は２次元なので最低でも２つの For 文が必要。）

データ gazou(大きさ X,Y)があった場合の C 言語での記述例

```
for(int i = 1;i < X-1;i++) {
  for (int j = 1;j < Y-1;j++) {
    double W=0.0;
    for (int i1=-1;i1<2;i1++) {
      for(int j1=-1;j1<2;j1++) {
        int ti1=i1+i;
        int tj1=j1+j;
        W += (double)gazou[ti1][tj1];
      }
    }
    newgazou(i,j)=W/9;
  }
}
```

この場合、newgazou のサイズが X-1×Y-1 になる点と、３番目、４番目の f or 文は、－１から＋１までの繰り返しになっている点に注意する。

このアルゴリズムを MATLAB に実装した例を示す。

```
load clown
gazou=X;close all
imshow(X,[]);[Y,X]=size(X)
newgazou=zeros(Y-1,X-1);
tic;
for i = 2:X-1
 for j=2:Y-1
  W=0;
```

```
   for i1=-1:1;
    for j1=-1:1;
     ti1=i1+i;tj1=j1+j;
     W=W+gazou(tj1,ti1);
    end
   end
   newgazou(j,i)=W/9;
 end
end
toc
figure(2);imshow(newgazou,[]);
```

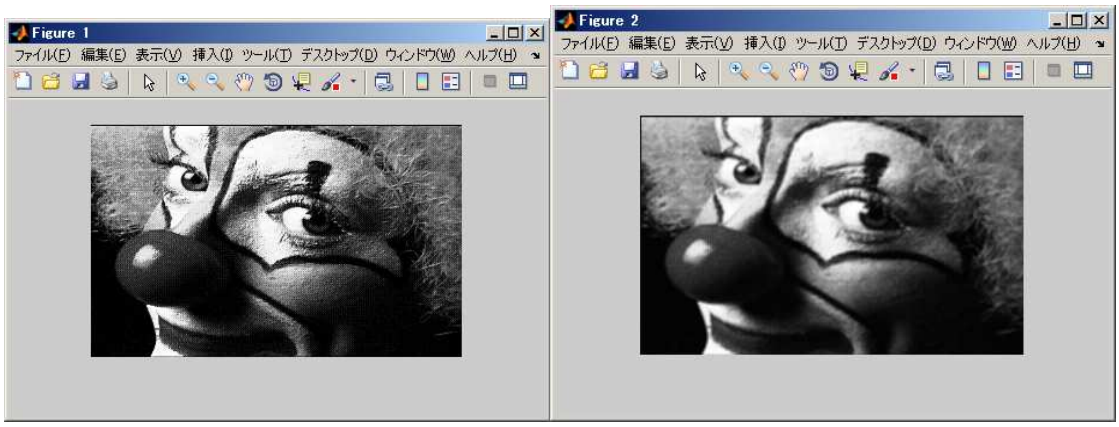


Figure1 がフィルタ前、Figure2 が平滑フィルタ後の画像

この計算と同様のアルゴリズムが conv2 関数を使うと、

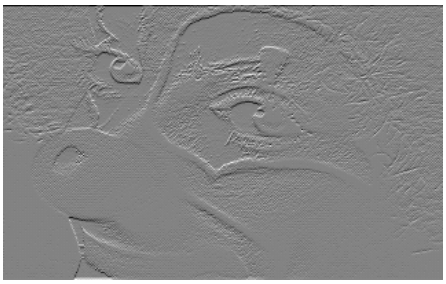
```
load clown
gazou=X;close all
tic
newgazou1=conv2(gazou,ones(3,3)/9,'same');
toc
figure
imshow(newgazou1,[]);
```

として計算できる。

conv2 関数を使うと、さまざまな局所フィルタリングを高速に処理できる。

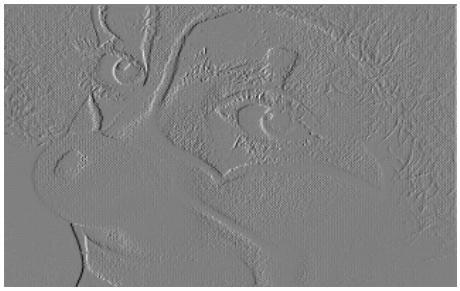
2 次元の相関計算も fft を使った方が圧倒的に速く計算できる。局所フィルタの重み行列の大きさは、３ｘ３に限定されるわけではない。たとえば 2x1 のサイズでも計算できる。

画像の縦差分



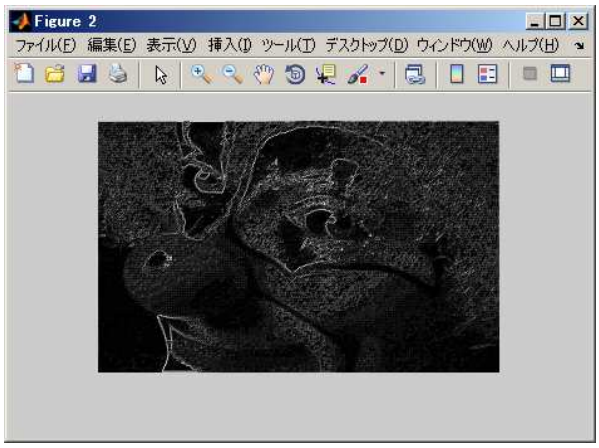
```
close all;load clown
gazou=im2double(X);
newgazou=conv2(gazou,[-1;1],'same');
figure;imshow(newgazou,[]);
```

画像の横差分



```
newgazou=conv2(gazou,[-1 1],'same');
figure;imshow(newgazou,[]);
```

両方の差分




```
newgazou=sqrt(conv2(gazou,[-1,1],
'same').^2+conv2(gazou,[-1;1],'same').^2);
figure;imshow(newgazou,[]);
```

2 次元相互相関計算（テンプレートマッチング）

直接的なパターンマッチング法では、相関による方法がある。MATLAB の場合、xcorr2 関数を使うと 2 次元の相互相関を計算できるが、MATLAB の場合、データサイズにもよるが、直接計算するのではなく、fft2 関数により一旦、フーリエ領域に変換後計算、逆変換をした方が処理速度が速い。ここでは、xcorr2 関数による方法と、fft2 関数によるテンプレートマッチングの例を示す。テンプレートとなる画像は、マウスにより任意の場所を指定する。

```
close all
load clown;gazou=X;
%gazou=double(imread('text.tif'));
[row,col]=size(gazou);
imshow(gazou,[])
[x,y]=ginput(1);
hold on;plot(x,y,'p');hold off
gazou=sign(gazou-median(gazou(:)));
a=gazou((-6:5)+round(y),(-6:5)+round(x));
tic;C1=xcorr2(double(gazou),double(a));xcorrtoc=toc;
tic;C=real(ifft2(fft2(gazou).*fft2(rot90(a,2),row,col)));ffttoc=toc;
figure;subplot(1,2,1);
imshow(gazou,[]);hold on;plot(x,y,'rp');hold off
title('cross correlation by xcorr2');
[y1,x1]=find(C>max(C(:))*0.95);
subplot(1,2,2);imshow(C,[]);
hold on;plot(x1,y1,'ro','markersize',20);hold off
title([num2str(xcorrtoc),'[sec]']);
figure;subplot(1,2,1);
imshow(gazou,[]);hold on;plot(x,y,'rp');hold off
title('cross correlation by fft2');
[y1,x1]=find(C1>max(C1(:))*0.95);
subplot(1,2,2);imshow(C1,[]);
hold on;plot(x1,y1,'ro','markersize',20);hold off
title([num2str(ffttoc),'[sec]']);
```

ここでは、xcorr2 関数と、fft2,ifft2 関数との比較では、ほぼ同じ速度であるが、約 0.15 秒で計算できている。

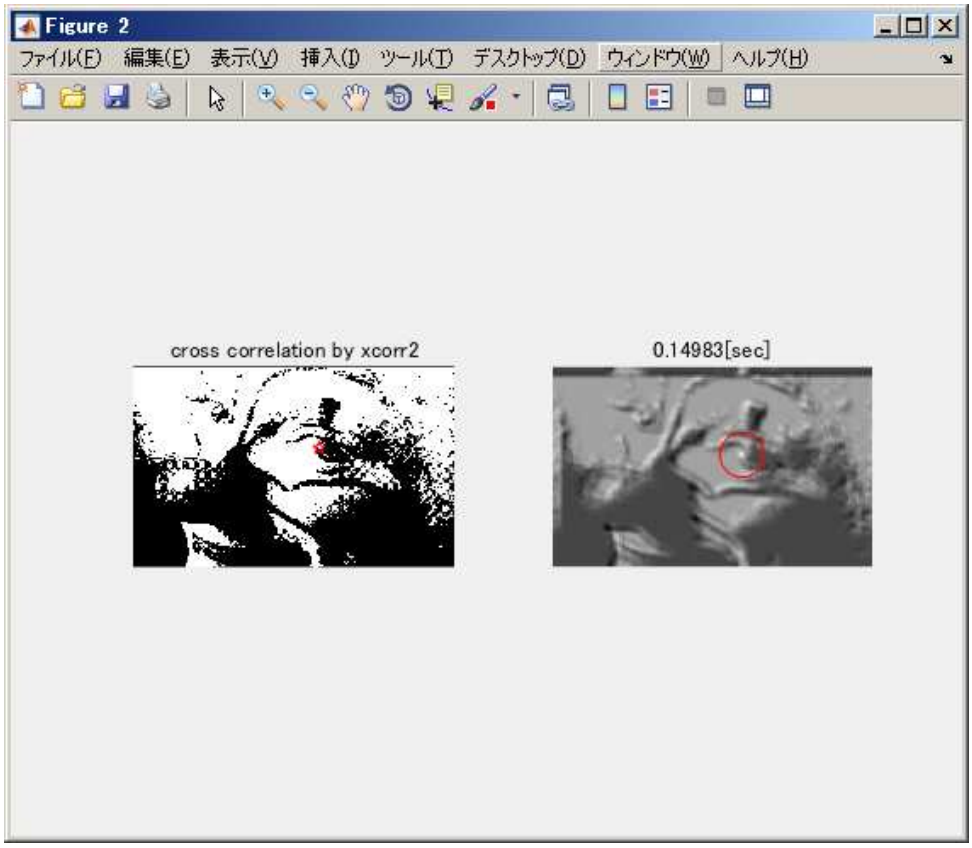


Figure 2
ファイル(F) 編集(E) 表示(V) 挿入(I) ツール(T) デスクトップ(D) ウィンドウ(W) ヘルプ(H)
cross correlation by xcorr2
0.14983[sec]

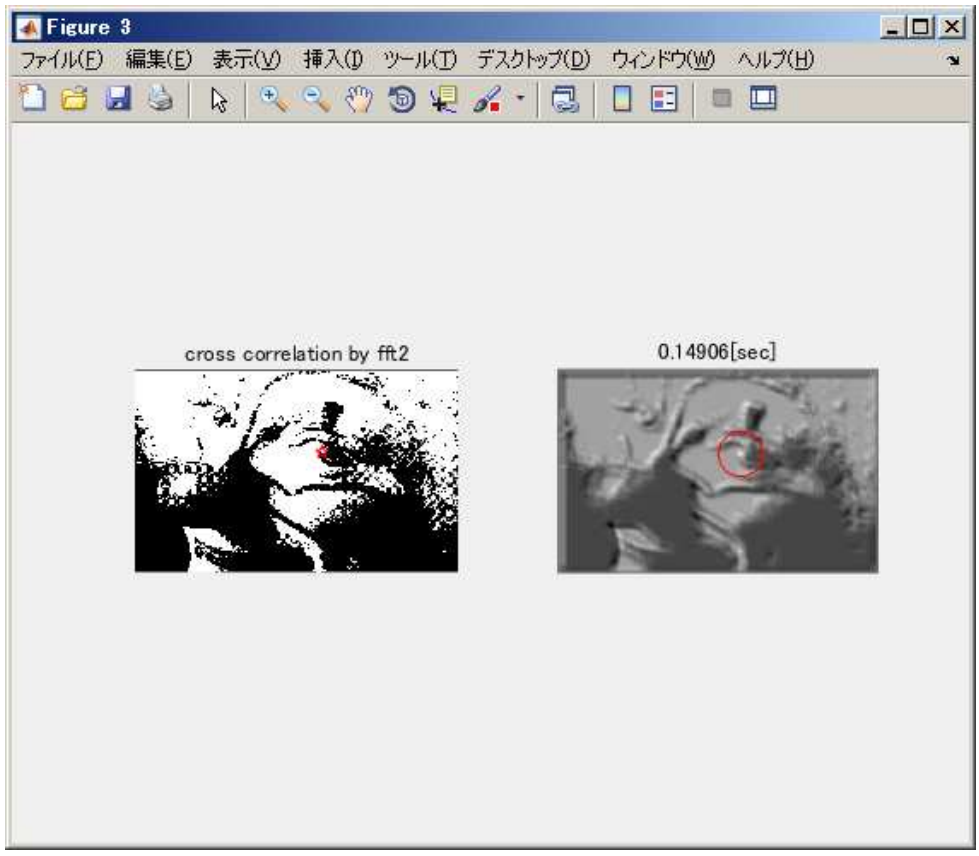


Figure 3
ファイル(F) 編集(E) 表示(V) 挿入(I) ツール(T) デスクトップ(D) ウィンドウ(W) ヘルプ(H)
cross correlation by fft2
0.14906[sec]