インタラクティブシステム論
第６回

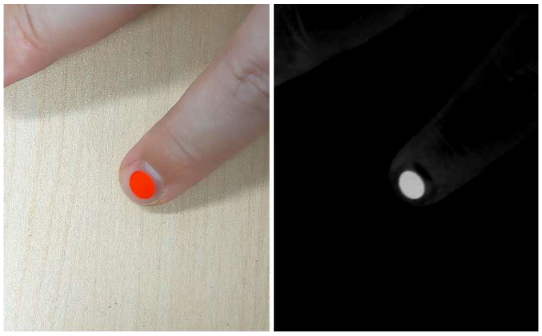
梶本裕之

日程

講義番号	講義日	講義内容	pdf	video	レポート締め切り
1	4/10	イントロダクション	[pdf] 2022年版	video	4/17
		Scilab課題	[pdf](更新なし)		↑
		上記資料のPython版	[pdf](更新なし)		↑
2	4/17	フーリエ変換	[pdf] 2022年版	video	4/24
3	4/24	フーリエ変換と線形システム	[pdf] 2022年版	video	5/1
4	5/1	信号処理の基礎	[pdf] 2022年版	video	5/8
5	5/8	信号処理の応用1(相関)	[pdf] 2022年版	video	5/15
6	5/15	信号処理の応用2(画像処理)	[pdf] 2022年版	video	5/22
-	5/22	中間確認テスト準備（自習）	[pdf] 2022年版		
-	5/29	中間確認テストとその解説	[pdf] 2022年版		
-	6/5	4ターム制試験日のため休み			
7	6/12	ラプラス変換	[pdf] 2022年版	video	6/19
8	6/19	古典制御の基礎	[pdf] 2022年版	video	6/26
9	6/26	行列	[pdf] 2022年版	video	7/3
10	7/3	行列と最小二乗法	[pdf] 2022年版	video	7/10
11	7/10	ロボティクス(出張によりオンライン・オンデマンド)	[pdf] 2022年版	video	7/17
-	7/17	期末テスト準備（自習）	[pdf]2022年版		
-	7/24	期末確認テストとその解説（現在は大学を予定）			

日程およびテストを大学で行うかについては、随時アナウンスします。Google Classroomでもアナウンスの予定。

画像処理とは



元の画像から
- 人間が理解しやすいように加工する
- 何らかの情報を抽出する

信号処理の一種.

特徴
- 2次元データである（動画なら3次元）
- 時間信号のような因果関係がない（動画ならある）

初歩的な画像処理

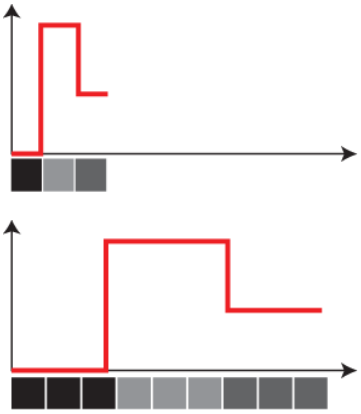
初歩的な画像処理（１）拡大・縮小

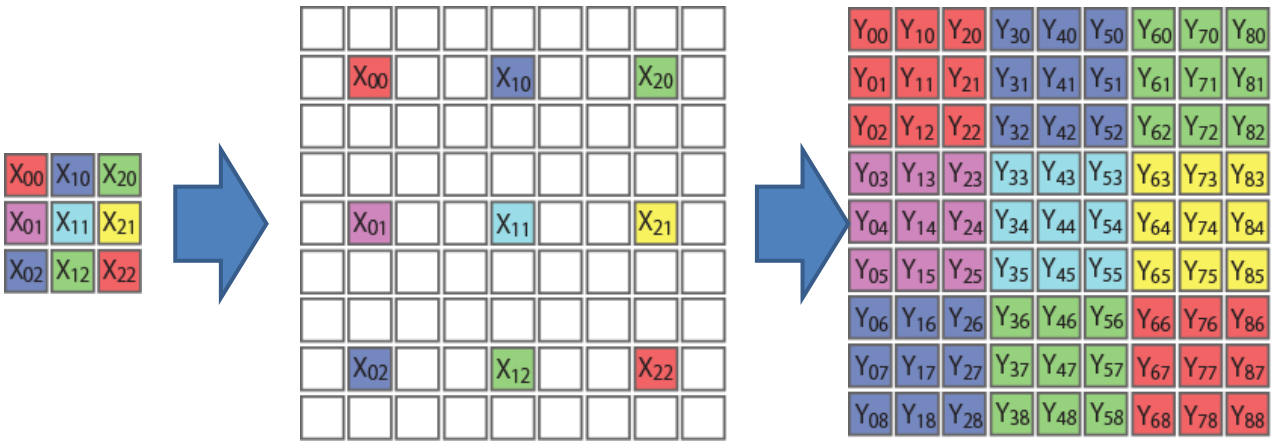
（例）3倍に拡大

一番簡単な方法：Nearest Neighbor (最近傍）法

$$Y_{i,j} = X_{i/3,j/3}$$

(ただしi/3は整数の割り算．1/3=0, 2/3=0, 3/3=1...)

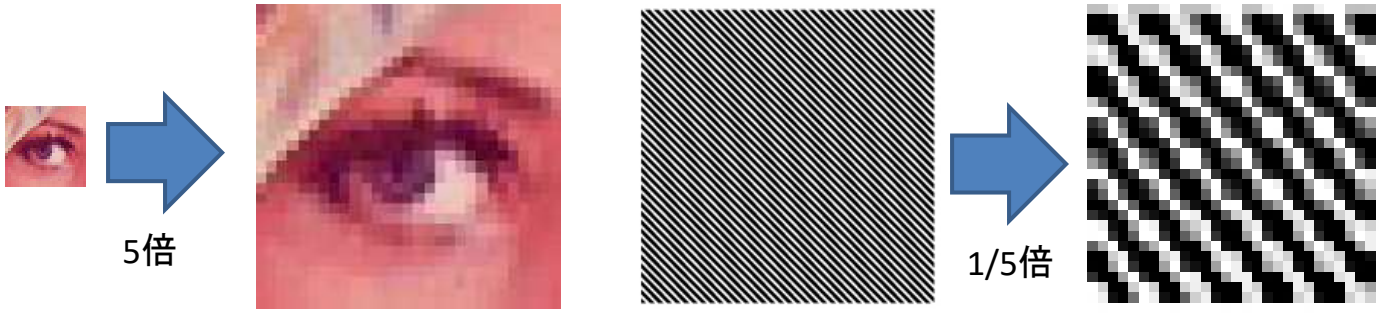




Nearest Neighbor

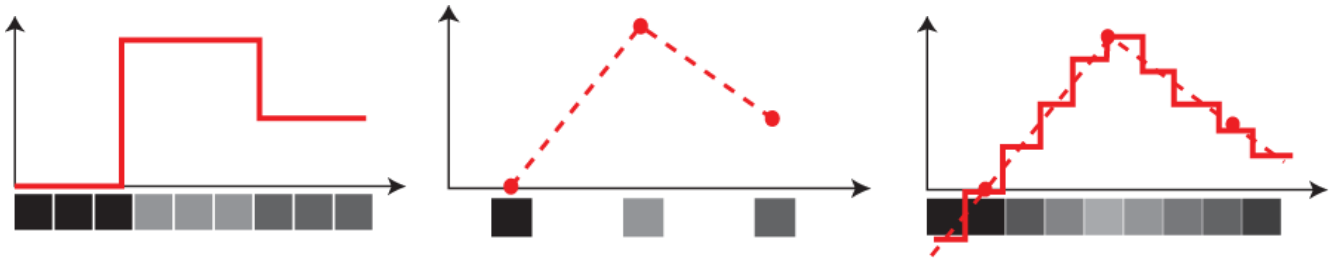
Photoshop使用

Nearest Neighbor法の問題



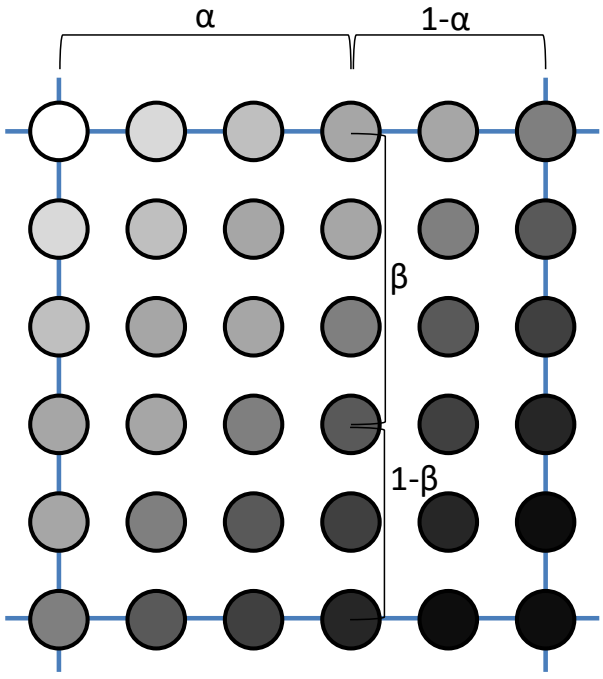
1. 荒さが目立つ
2. 縮小時には偽の周波数(モワレ)を生じる
   (サンプリング間隔の変化によるエリアシング)

もっとなだらかに結べばよい⇒直線補間.

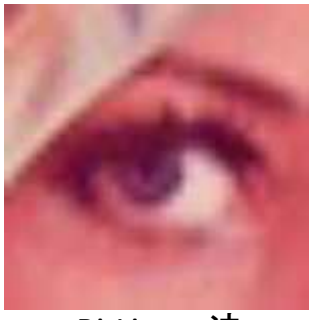


Bi-Linear法

2次元画像なので4点間を線形補間
Bi :線形補間を2回することを表す




Nearest Neighbor法


Bi-Linear法

他にBi-Cubic法など

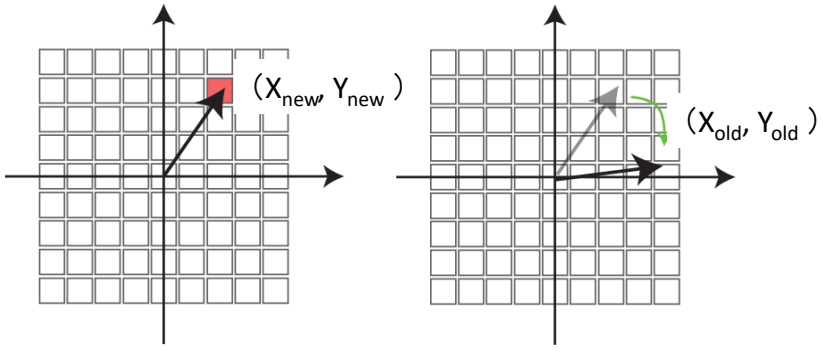
Bi-Linear

Photoshop使用

初歩的な画像処理（２）回転

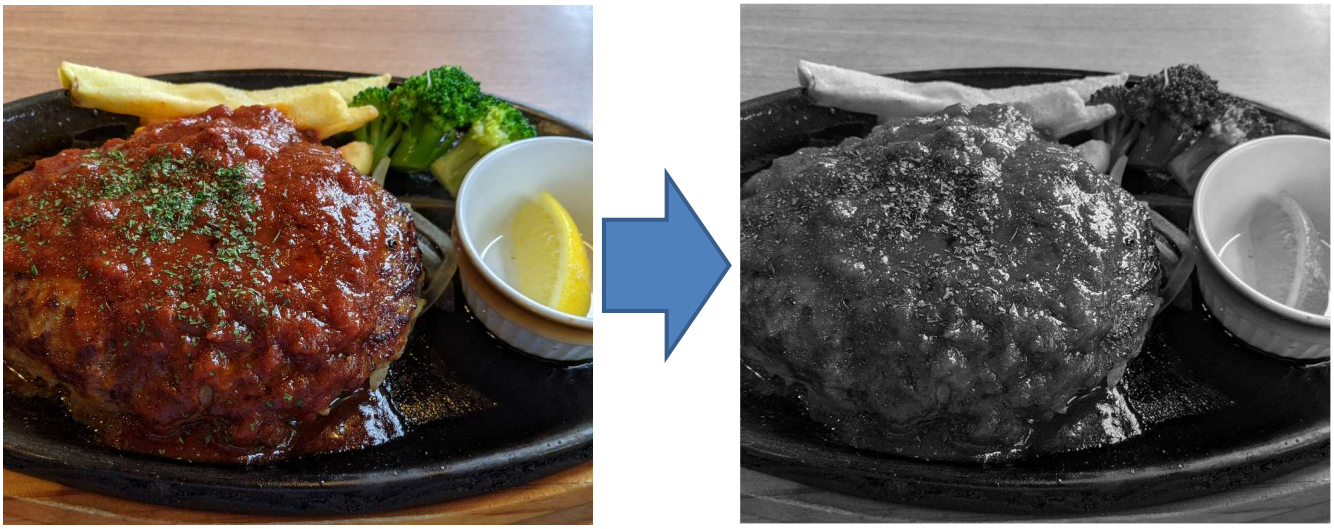
（1）新画像のあるピクセル座標X_{new}, Y_{new}が，
元画像でどこに位置していたか計算．（順番に注意）

$$\begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix} \begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix}$$



（2）X_{old}, Y_{old}は小数
⇒整数にして，そのピクセルの色を使う(Nearest Neighbor法)
⇒周辺の4ピクセルから補間する (Bi-Linear法)

初歩的な画像処理（３）グレースケール化

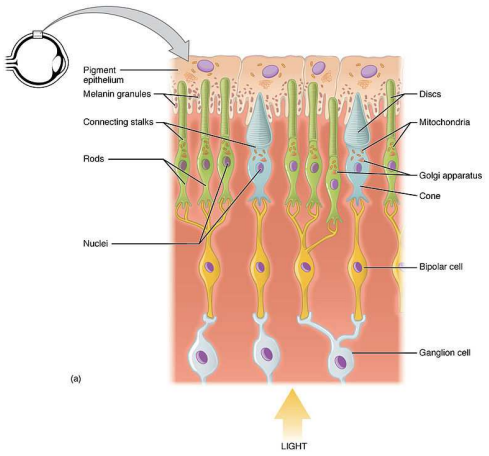


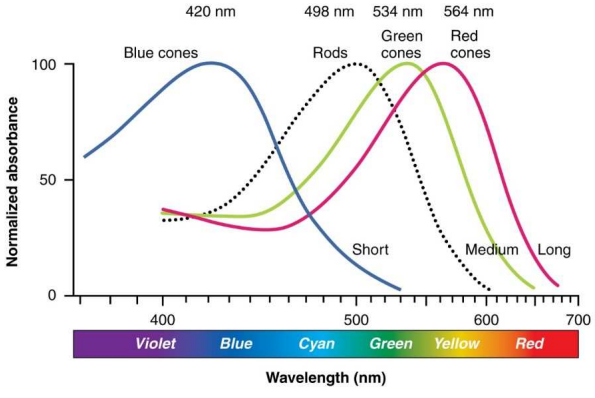
誰でも考える方法：R,G,Bの平均：

$$K_{ij}=(R_{ij}+G_{ij}+B_{ij})/3$$

悪くはないが，最良でもない．

人の光感受性細胞





桿体細胞(Rod)
- 明暗センサ

錐体細胞(Cone)
- 青錐体細胞（S細胞）435nm近辺
- 緑錐体細胞（M細胞）546nm近辺
- 赤錐体細胞（L細胞）600nm近辺

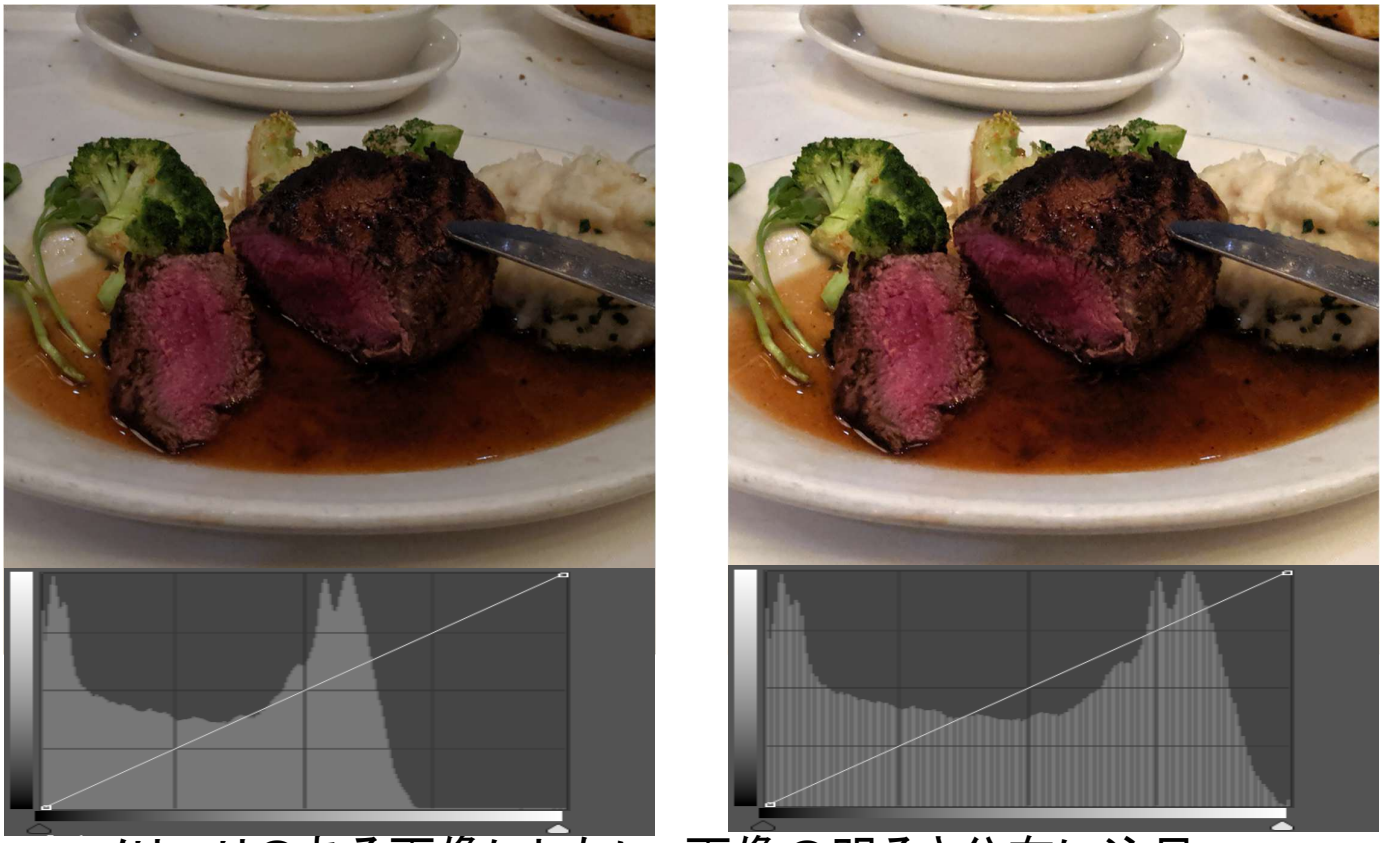
Photoreceptor Cells (Wikipedia) https://en.wikipedia.org/wiki/Photoreceptor_cell

同じ輝度のR, G, Bを，人は同じ明るさに感じない⇒補正

心理的に正しいグレースケール変換：

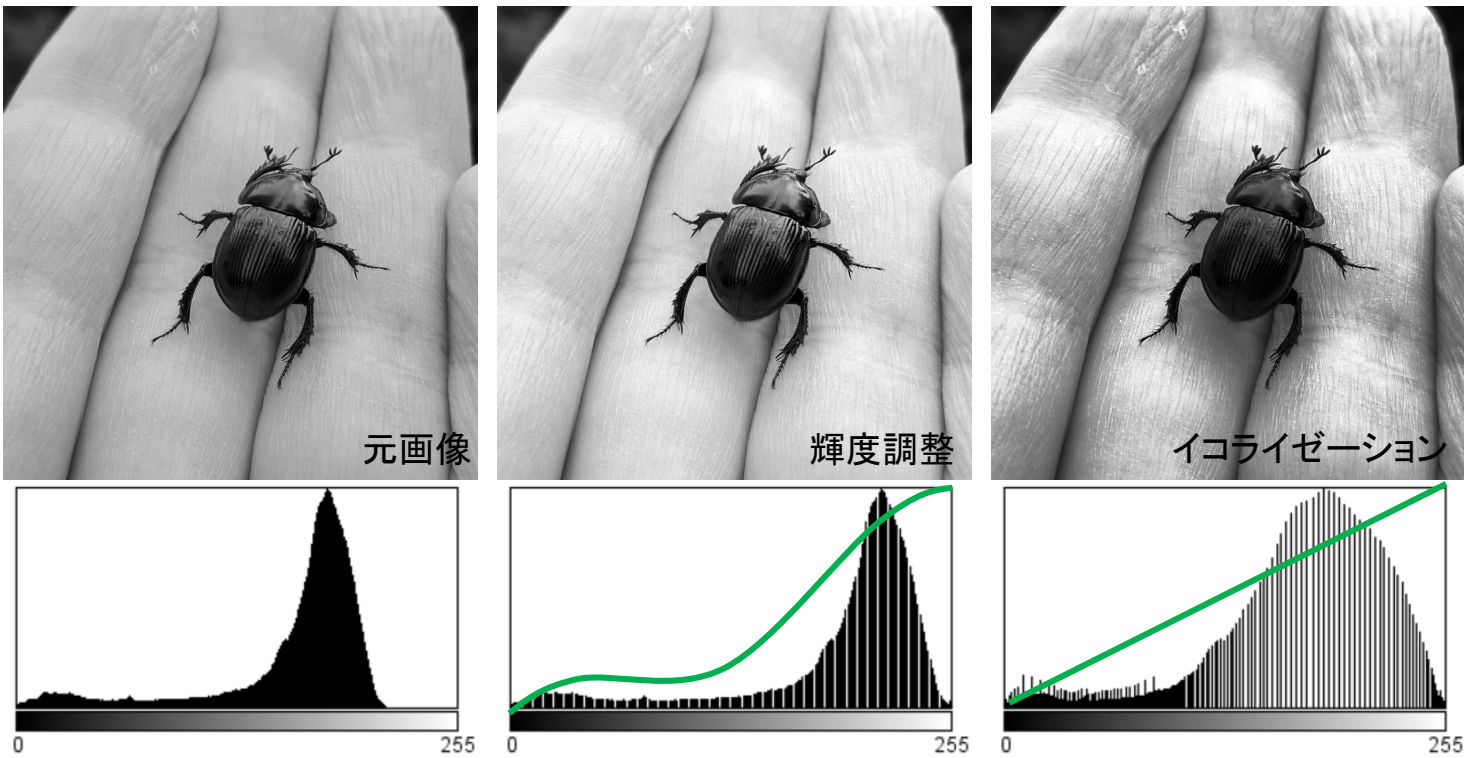
$$K_{ij}= \underline{0.299}R_{ij} + \underline{0.587}G_{ij} + \underline{0.114}B_{ij}$$

初歩的な画像処理（４）濃度調整



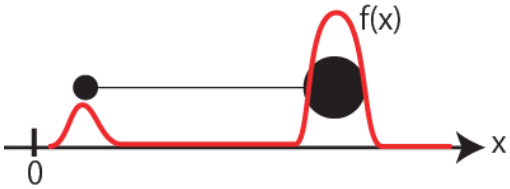
メリハリのある画像にしたい：画像の明るさ分布に注目
ヒストグラムを作成、最大値、最小値を255-0に合わせる。

濃度調整のバリエーション：イコライゼーション

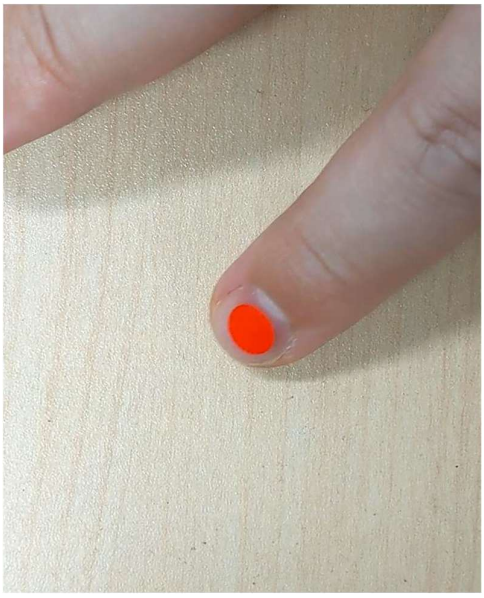


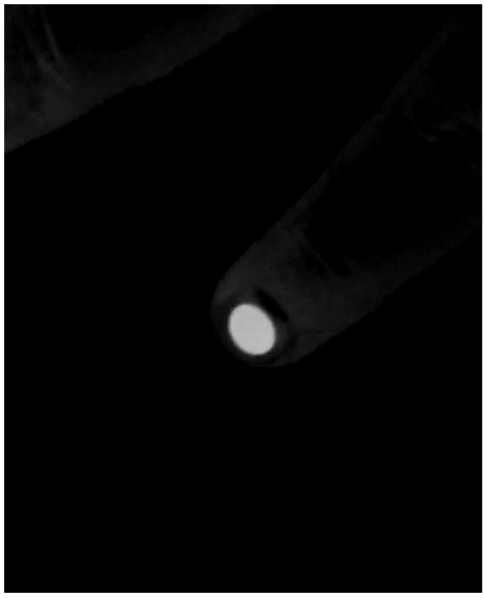
ヒストグラムの累積度数のグラフの傾きを一定化
（＝どの明るさも同じ量だけ使われている）
（この例では手の白領域がより明瞭にコントラストが付いている）

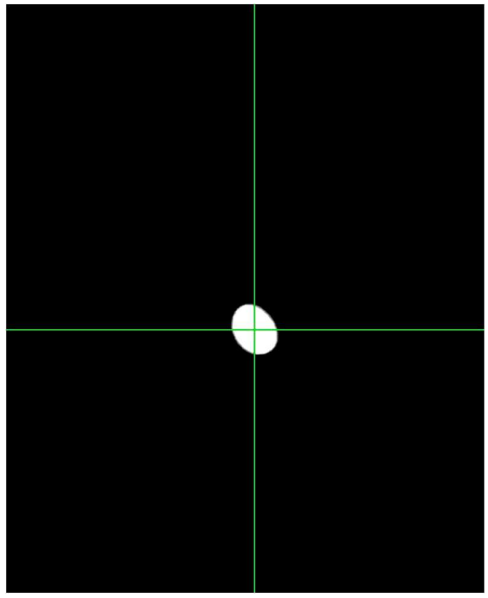
初歩的な画像処理（５）重心計算



$$c = \frac{\int x \times f(x)dx}{\int f(x)dx}$$







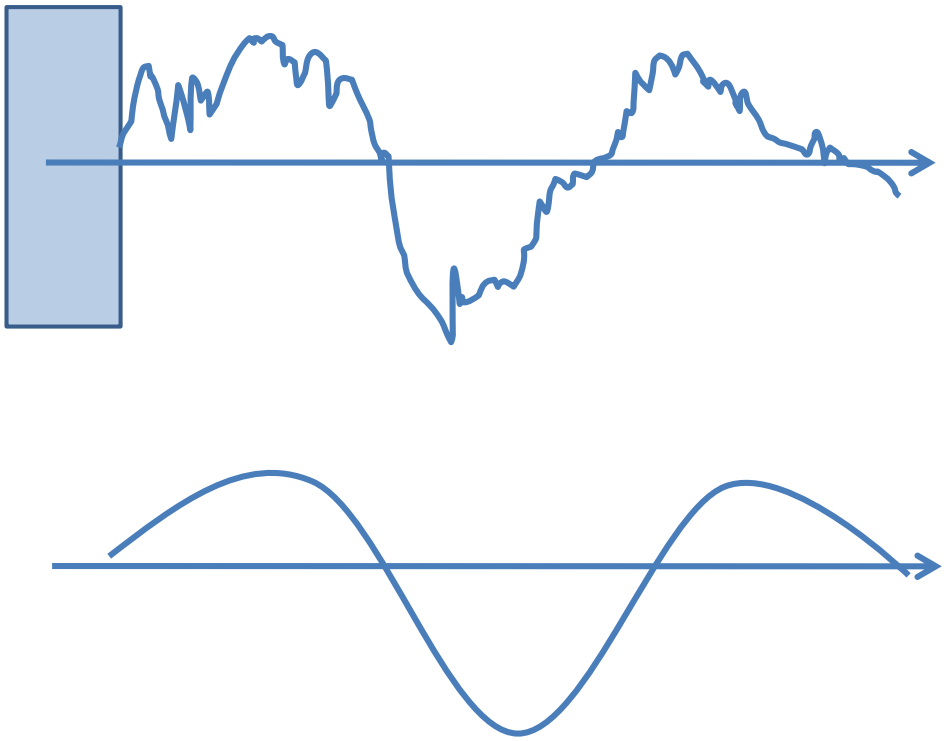
元画像　　Red - Green　　閾値処理後、重心計算

重心計算は事前の閾値処理（による不要部分の0化）が必須

ImageJ使用

画像のフィルタリング

（復習）平均化＝ローパスフィルタ



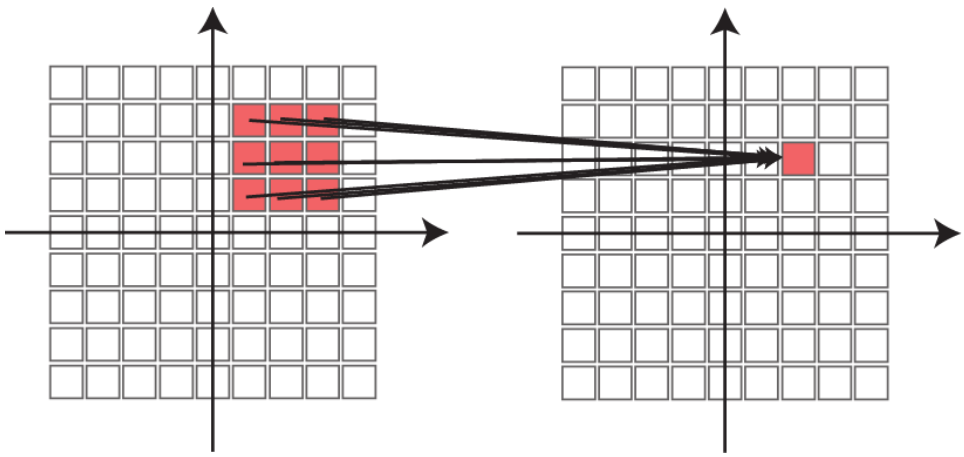
ノイズを「ならし」て大域的な特徴をつかむ

画像の平滑化

1次元信号の平滑化と同様に，2次元的に平均すればよい．

3x3領域を平均化する場合：

$$Y_{i,j} = X_{i-1,j-1} + X_{i-1,j} + X_{i-1,j+1} + X_{i,j-1} + X_{i,j} + X_{i,j+1} + X_{i+1,j-1} + X_{i+1,j} + X_{i+1,j+1}$$



オペレータ

3x3領域を使った演算を一般化：

$$Y_{i,j} = aX_{i-1,j-1} + bX_{i-1,j} + cX_{i-1,j+1} + dX_{i,j-1} + eX_{i,j} + fX_{i,j+1} + gX_{i+1,j-1} + hX_{i+1,j} + iX_{i+1,j+1}$$

この係数行列をオペレータという.
FIRフィルタの係数と同じ役割.

先ほどの平滑化：すべての係数が等しい

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

オペレータの演算例

元画像

$$\begin{bmatrix} 1 & 2 & 3 & 2 \\ 2 & 3 & 3 & 3 \\ 3 & 4 & 2 & 4 \\ 4 & 5 & 1 & 5 \end{bmatrix}$$

オペレータ

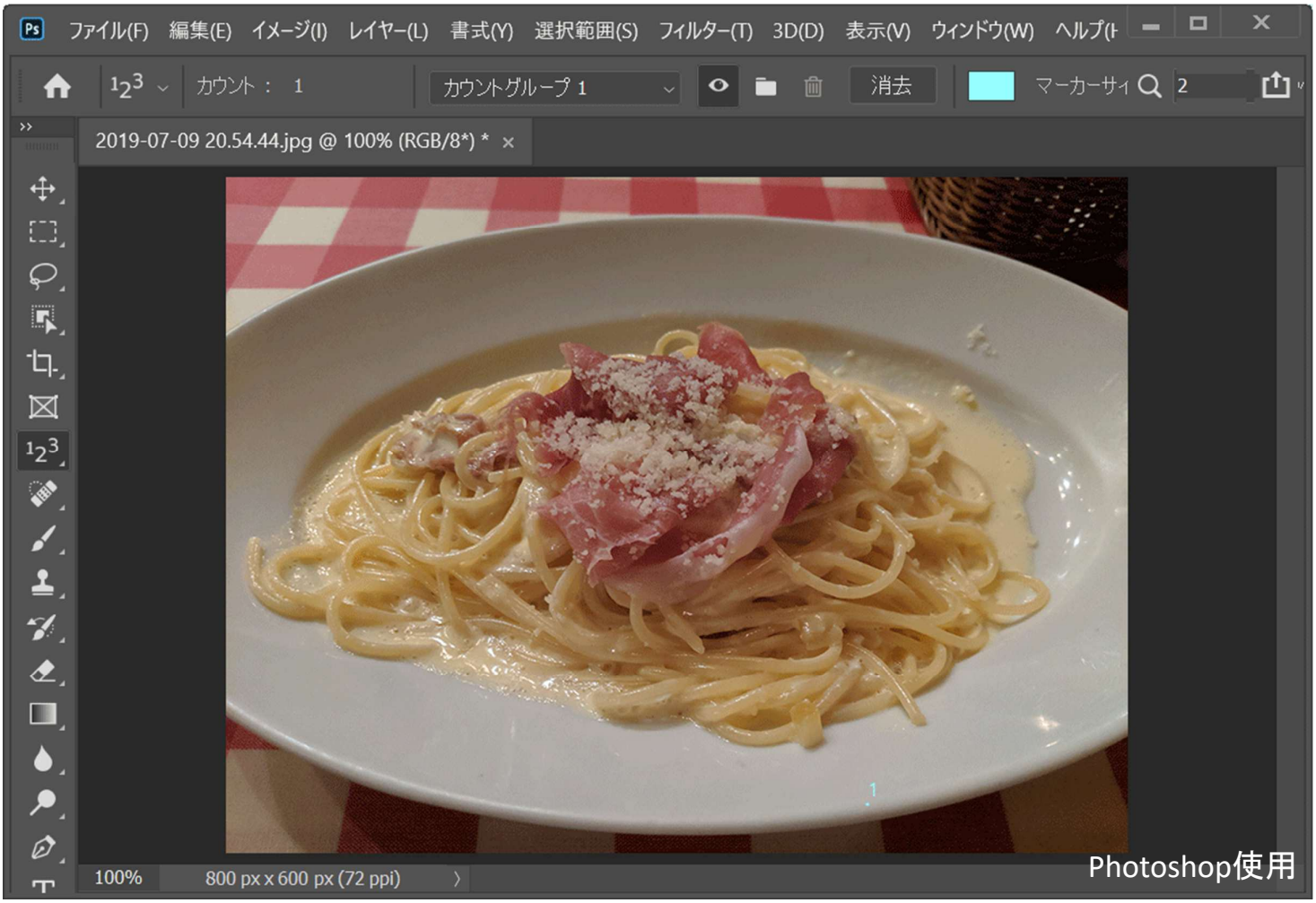
$$\frac{1}{4} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

結果

$$\frac{1}{4} \times \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

端の処理が問題となる場合はとりあえず考えない

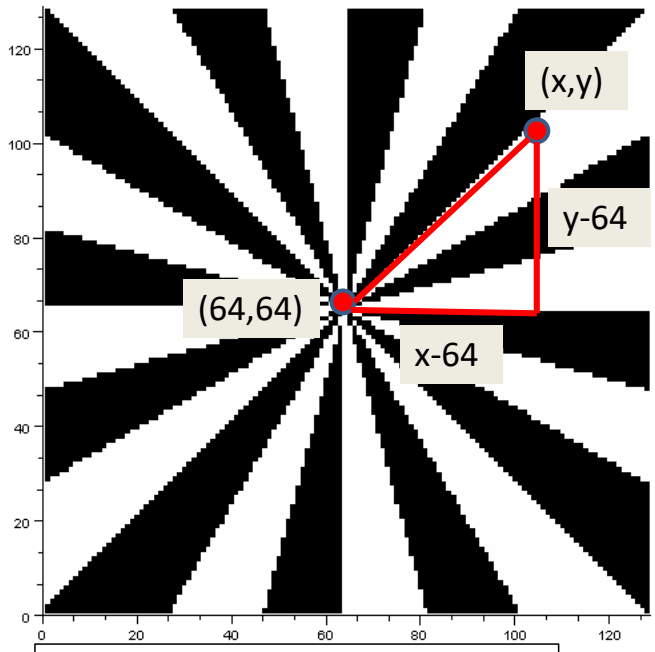
デモ：平滑化



Scilabレポート課題準備：サンプル画像作成

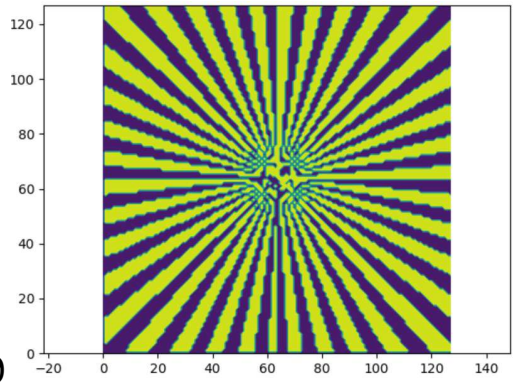
```
for x=1:128,
  for y=1:128,
    deg = atan(y-64,x-64)/%pi*180;
      if(pmodulo(deg,30)<15)
        img(x,y)=255;
      else
        img(x,y)=0;
      end
  end
end

f = scf();
f.color_map = graycolormap(256);
Matplot(img);  //行列を
square(0,0,129,129);
```

128×128の行列を用意
中心から放射状に伸びる縞



256階調グレースケール表示

(参考) Pythonでのコード例




```
import matplotlib.pyplot as plt
import numpy as np
img = np.zeros((128,128))
for x in range(1,128):
    for y in range(1,128):
        deg = np.arctan2(y-64,x-64) * 180
        if(deg%30 < 15):
            img[x,y] = 255
        else:
            img[x,y] = 0
plt.contourf(img)
plt.axes().set_aspect('equal','datalim')
plt.show()
```

Img配列の生成は以下のように書くことも出来る

```
xx, yy = np.meshgrid(np.arange(-64,64),np.arange(-64,64))
deg = np.arctan2(xx,yy) * 180
img = np.floor(np.mod(deg,30) / 15)
```

Scilabによる3x3の平均化

元画像生成部は省略

```
img2=zeros(126,126);
//3x3のオペレータによる平均化
for x=1:126,
 for y=1:126,
  img2(x,y)= ...
  (img(x,y)  +img(x+1,y)  +img(x+2,y)+...
   img(x,y+1)+img(x+1,y+1)+img(x+2,y+1)+...
   img(x,y+2)+img(x+1,y+2)+img(x+2,y+2))/9;
 end
end
```

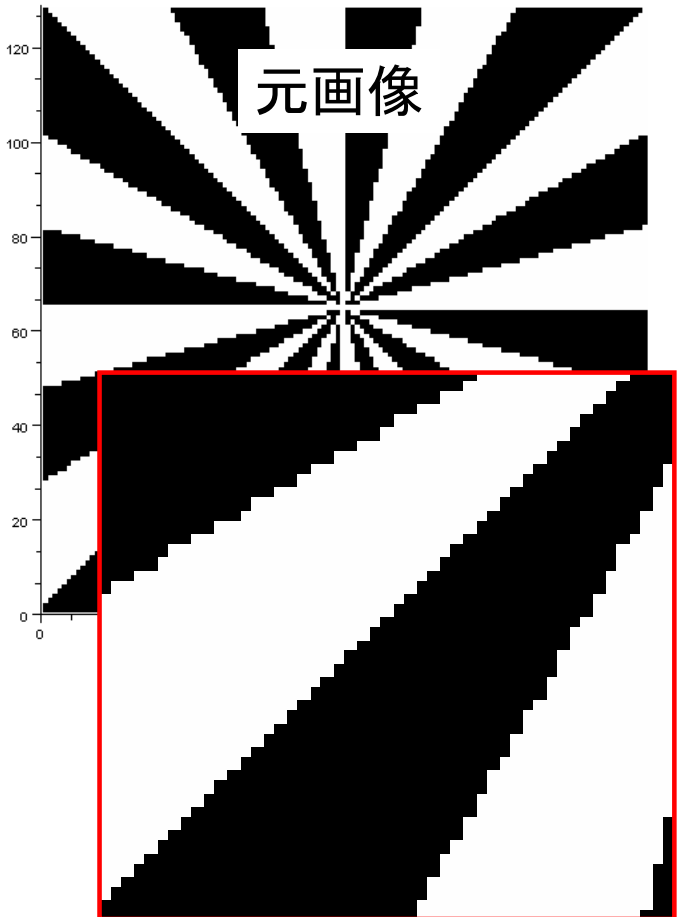
128でないことに注意！

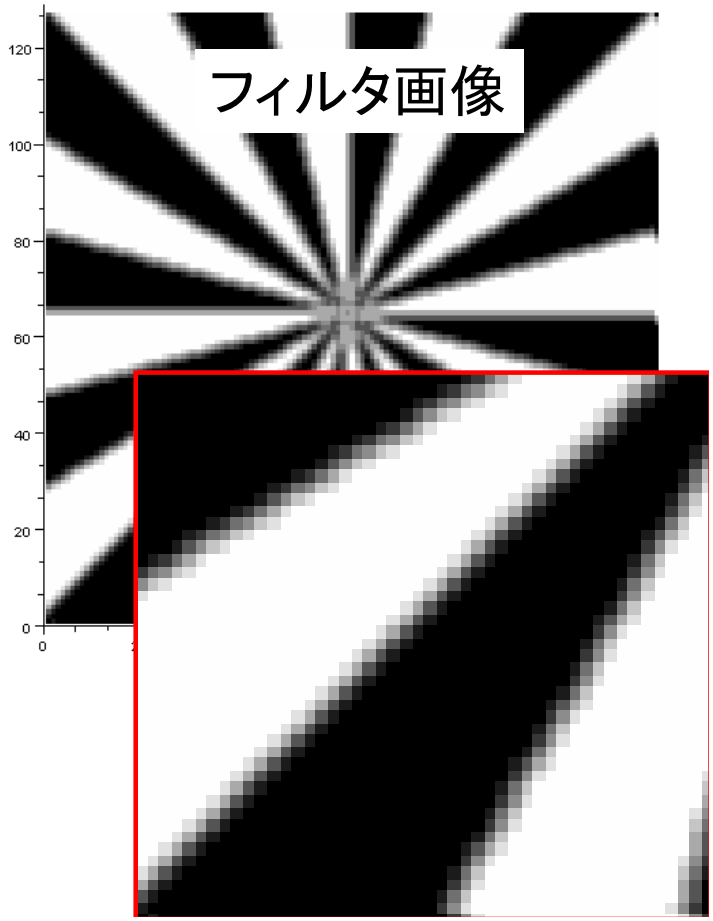
次の行に続く印

平均

画像表示部は省略

3x3の平均化





5x5の平均化

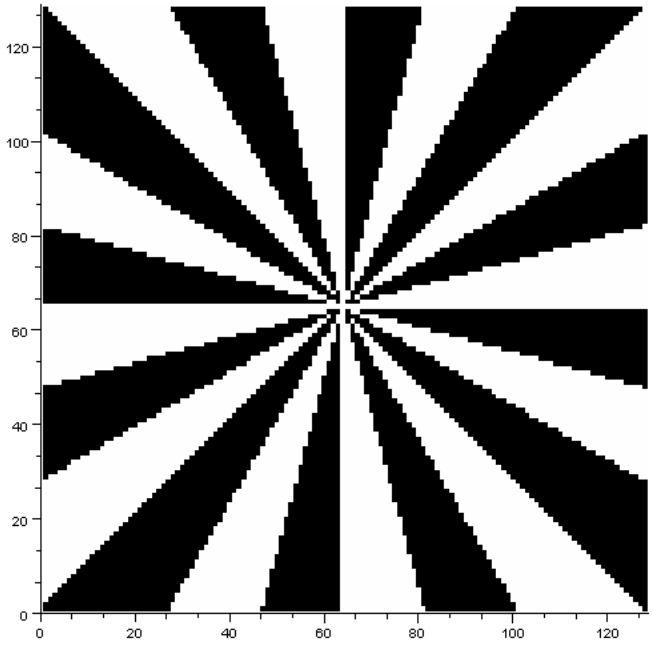
元画像生成部分は省略

```
Img2=zeros(124,124);
for x=1:124,
  for y=1:124,
    img2(x,y)= ...
    (img(x,y)  +img(x+1,y)  +img(x+2,y)  +img(x+3,y)  +img(x+4,y)+...
     img(x,y+1)+img(x+1,y+1)+img(x+2,y+1)+img(x+3,y+1)+img(x+4,y+1)+...
     img(x,y+2)+img(x+1,y+2)+img(x+2,y+2)+img(x+3,y+2)+img(x+4,y+2)+...
     img(x,y+3)+img(x+1,y+3)+img(x+2,y+3)+img(x+3,y+3)+img(x+4,y+3)+...
     img(x,y+4)+img(x+1,y+4)+img(x+2,y+4)+img(x+3,y+4)+img(x+4,y+4))/25;
  end
end
```

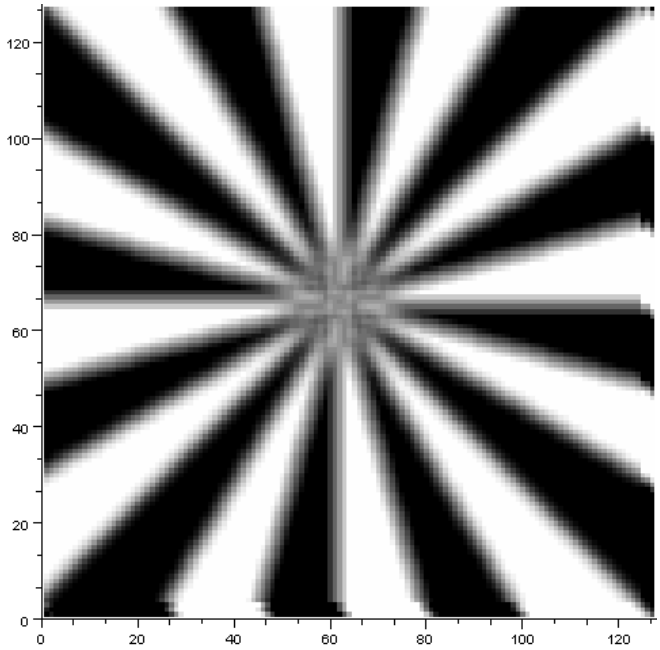
画像表示部分は省略

5x5の平均化

元画像

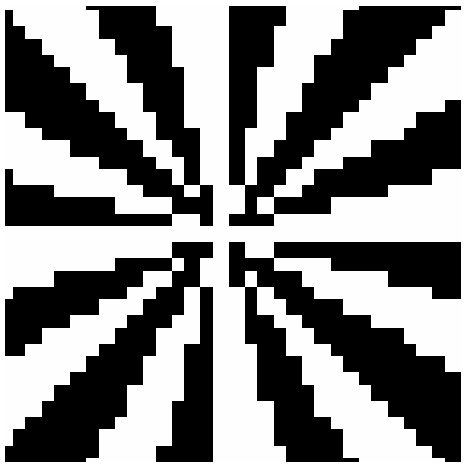


フィルタ画像

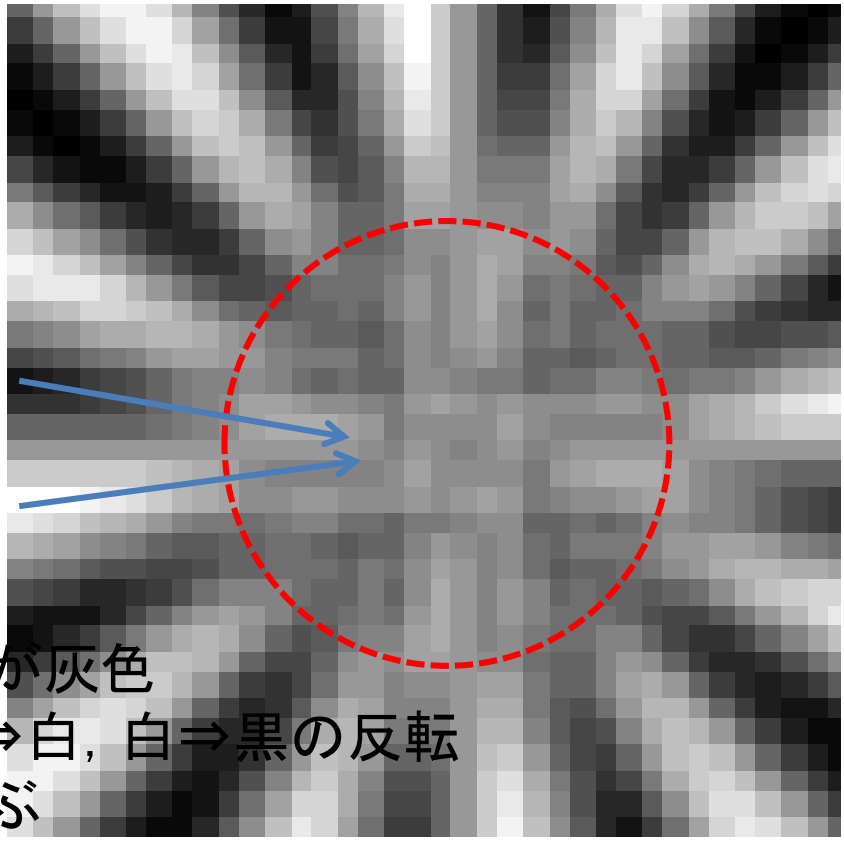


中心付近を拡大してみる

元画像

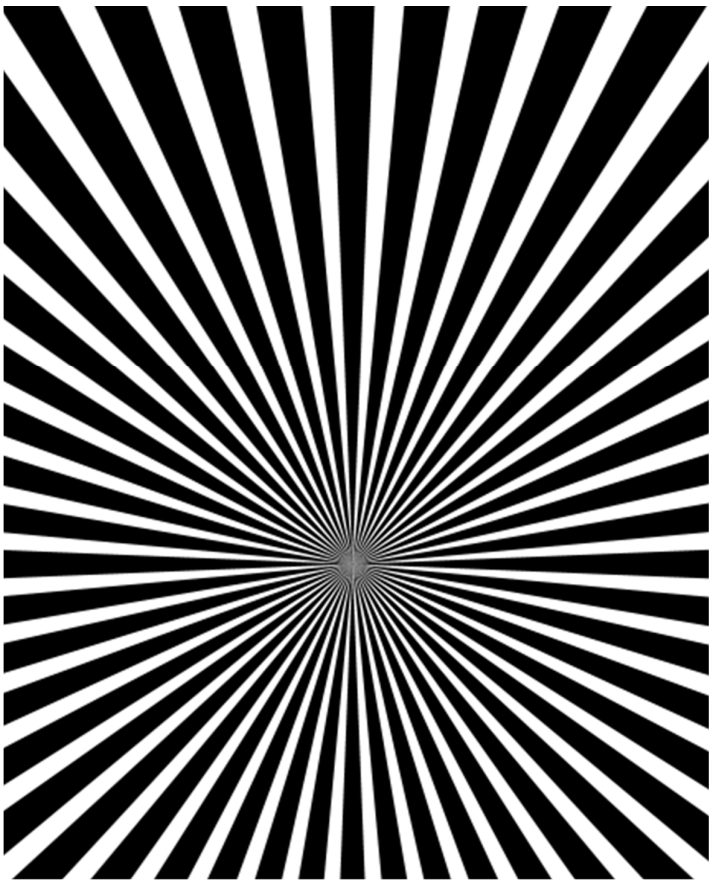


フィルタ画像

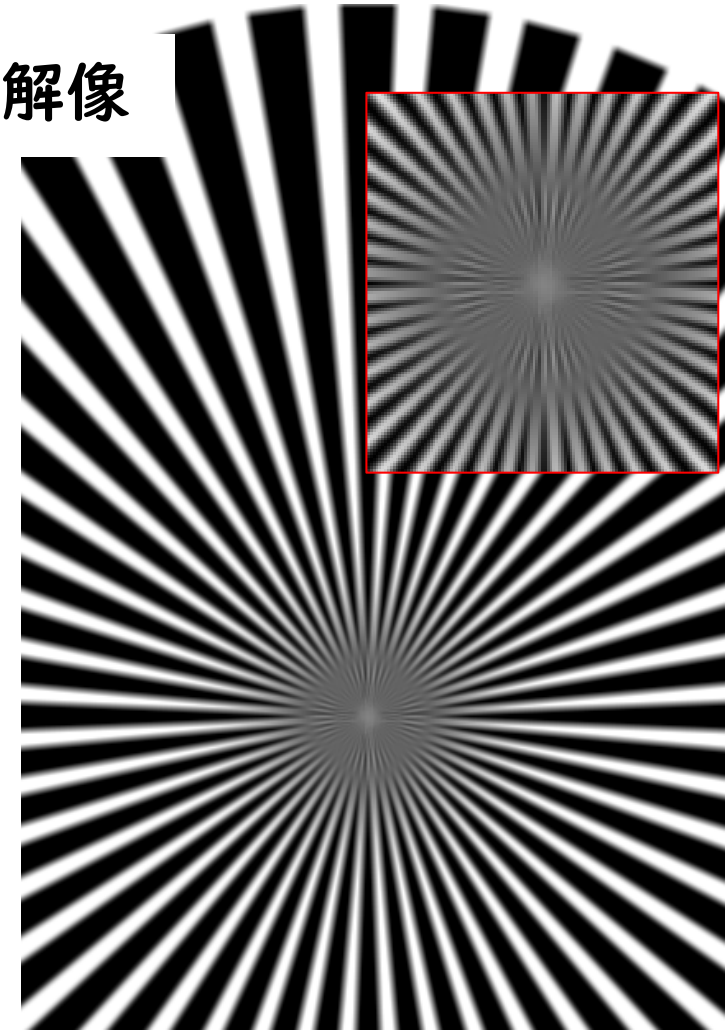


ある直径の円周上が灰色
その内側では，黒⇒白，白⇒黒の反転
「偽解像現象」と呼ぶ

平均化フィルタによる偽解像

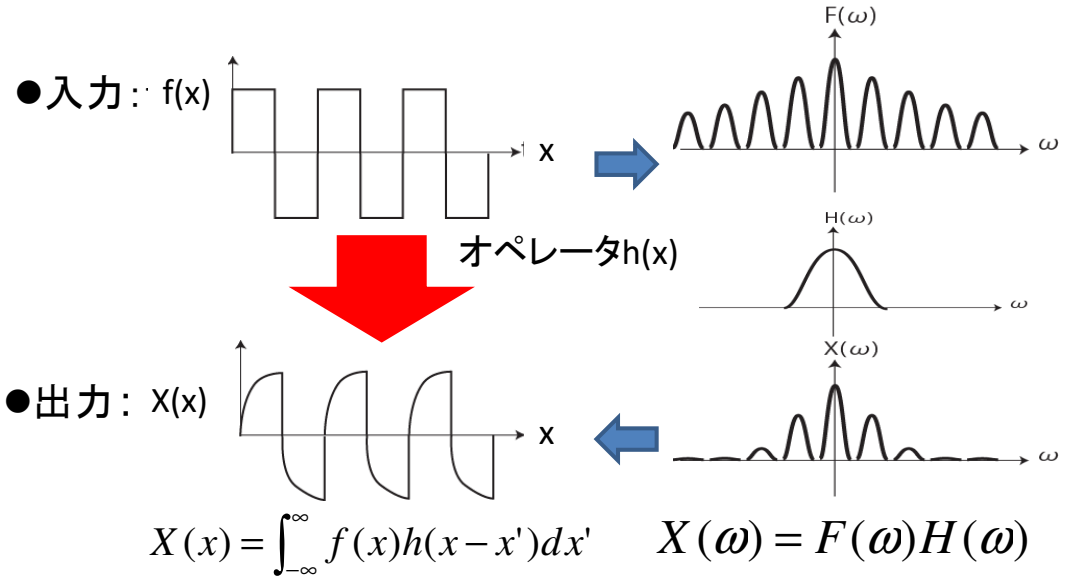


元画像



5x5オペレータによる平均化

（復習）オペレータとフーリエ変換



$$X(x)=\int_{-\infty}^{\infty} f(x)h(x-x')dx' \qquad X(\omega)=F(\omega)H(\omega)$$

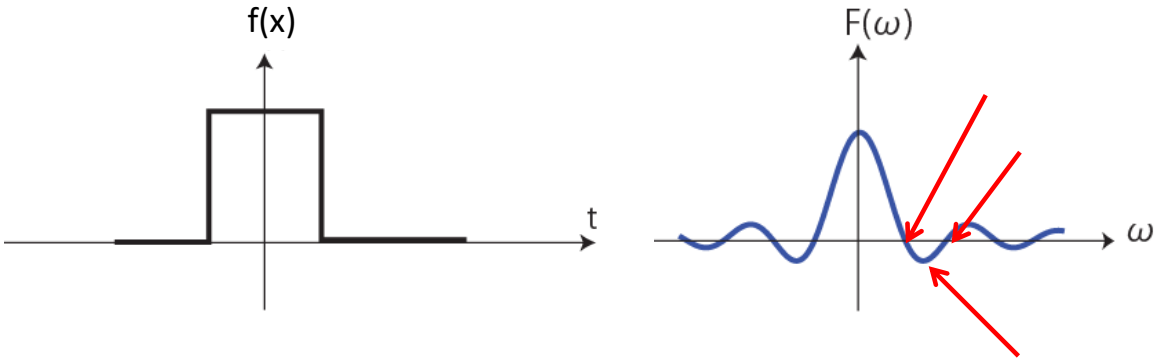
- オペレータh(x)のフーリエ変換がH(ω)であるとする.
- 空間領域でのオペレータの畳込み積分（コンボリューション）は，周波数領域でオペレータをフーリエ変換したフィルタH（ω）をかけることと等価

オペレータ＝フィルタ

オペレータのフーリエ変換例

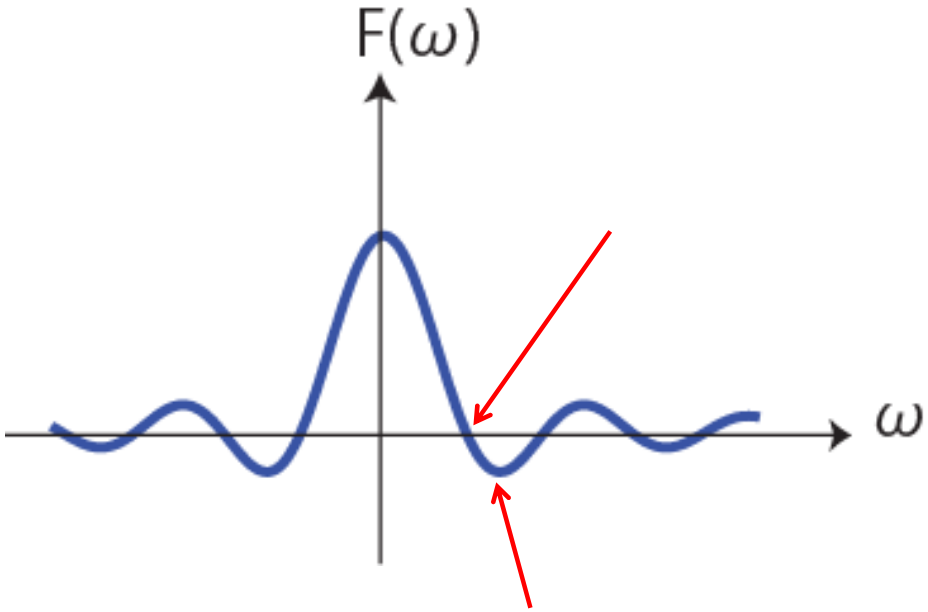
$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

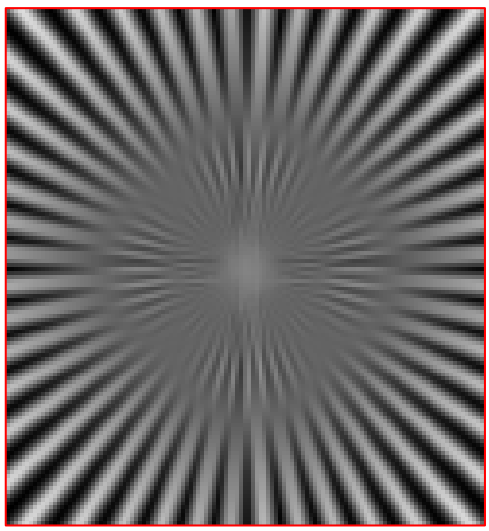
フィルタの形が矩形の場合　⇒　フーリエ変換するとSinc関数



平均化＝Low Pass Filterというのは，近似にすぎない．
単なるLow Pass Filterではない
●特定の周波数のゲインは0（画像では灰色になる）
●周波数によっては位相が反転（画像では白黒反転⇒偽解像）

偽解像現象はなぜ生じるか



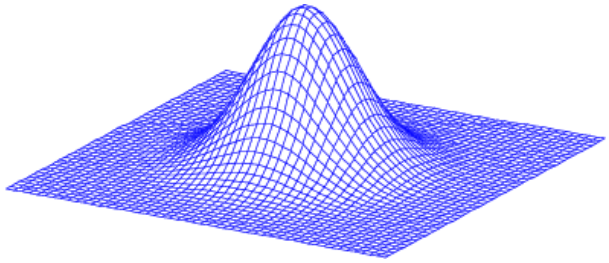


平均化＝Low Pass Filterというのは，近似にすぎない．
単なるLow Pass Filterではない．
●特定の周波数のゲインは0（画像では灰色になる）
●周波数によっては位相が反転（画像では白黒反転⇒偽解像）

画像平滑化の実際：ガウシアンフィルタ

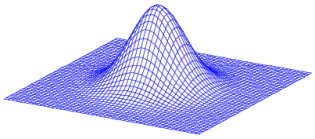
3x3ガウシアンオペレータ

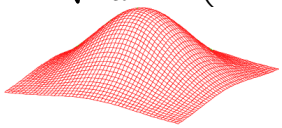
$$\frac{1}{15}\times\begin{bmatrix}1 & 2 & 1\\ 2 & 3 & 2\\ 1 & 2 & 1\end{bmatrix}$$



フィルタの形がガウシアン　⇒　フーリエ変換してもガウシアン

$$h(x)=\exp\left(-ax^2\right) \Rightarrow H(\omega)=\frac{1}{\sqrt{2a}}\exp\left(-\frac{\omega^2}{4a}\right)$$





先ほどの問題点が解決され，素直なLPFとなる．

実用的なオペレータサイズ：3x3，または5x5

5x5ガウシアンオペレータ

$$\frac{1}{331}\times\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 20 & 33 & 20 & 4\\ 7 & 33 & 55 & 33 & 7\\ 4 & 20 & 33 & 20 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix}$$

事前処理としてのガウシアンフィルタ

Photoshop使用

多くの画像処理で，事前にガウシアンをかけて
ノイズを除去する．

レポート課題（１）

元画像に5x5のガウシアンフィルタをかけ，ぼかしてみる
元画像と比較し，ぼけていることを確認せよ

（ヒント）5x5の単純平均化のソースコードを改変

もう一つのノイズ除去：メディアンフィルタ

ノイズが強力かつ小さい時

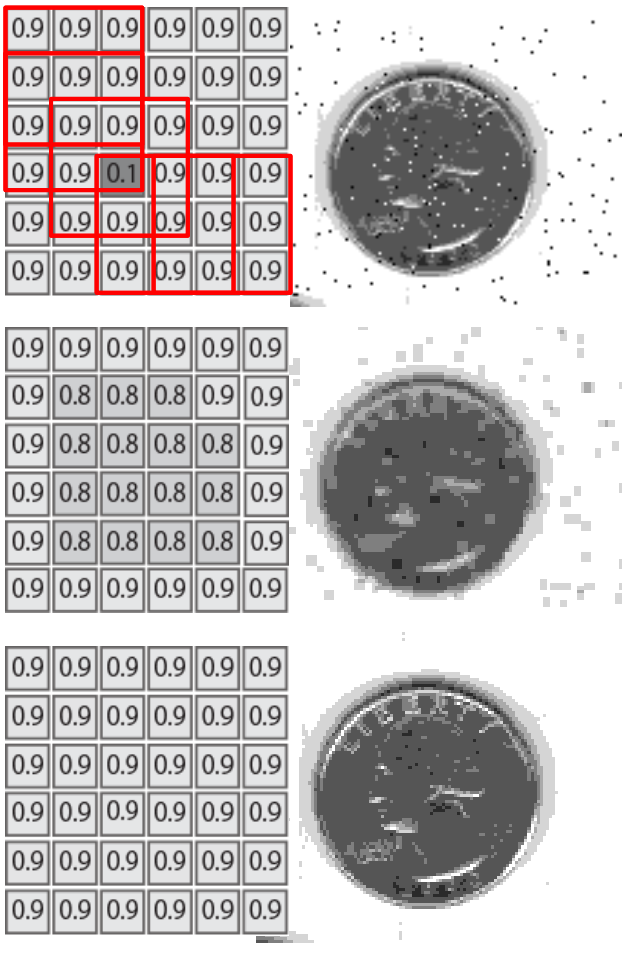
（１）LPFではノイズが「薄く広がる」だけ.

（２）中間値（メディアン）を用いる.

3x3領域を使う場合：

$Y_{i,j}$ = 中間値（$X_{i-1,j-1}$, $X_{i-1,j}$, $X_{i-1,j+1}$,
$X_{i,j-1}$, $X_{i,j}$, $X_{i,j+1}$,
$X_{i+1,j-1}$, $X_{i+1,j}$, $X_{i+1,j+1}$）

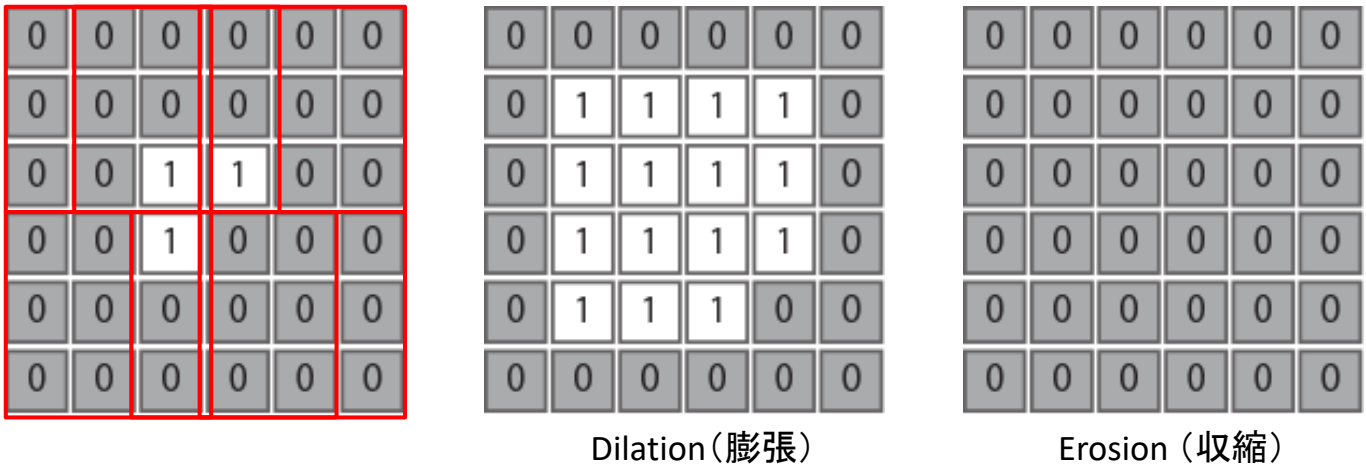
9個の値をソート
⇒5番目を採用



（参考）モルフォロジー（形態）処理

特に2値画像で用いられる.

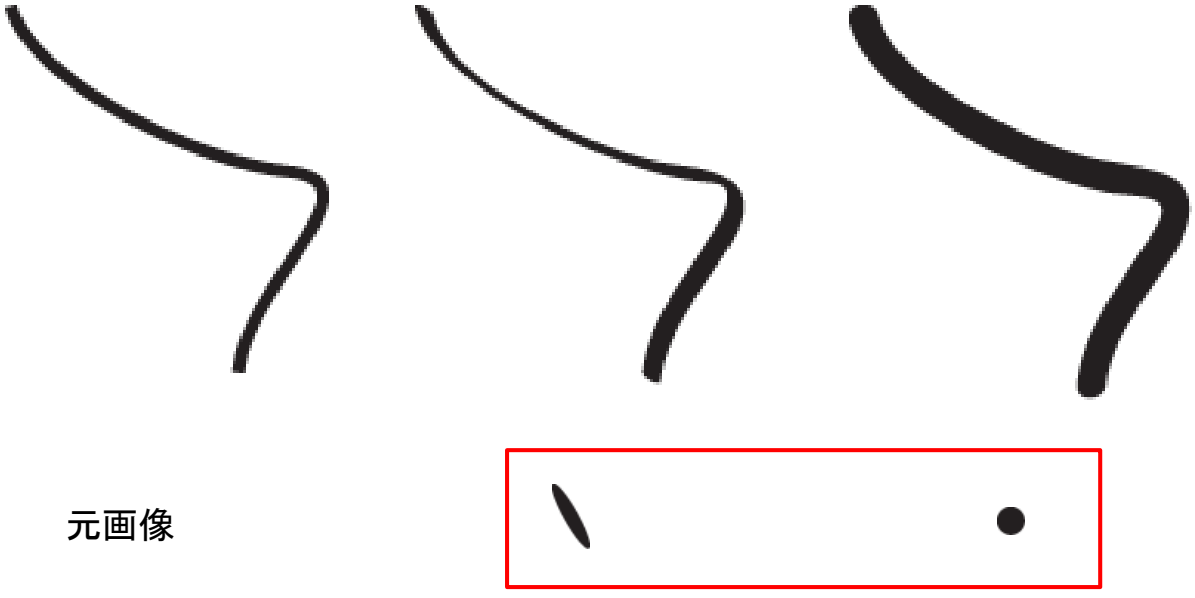
範囲内の最大値を取る：Dilation（膨張）
範囲内の最小値を取る： Erosion（収縮）



Dilation（膨張）　　Erosion（収縮）

（参考）モルフォロジー（形態）処理

「範囲」の形状を定義すれば筆の効果も得られる



元画像

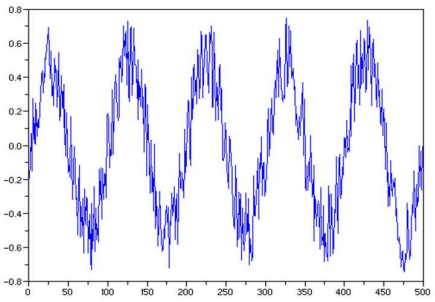
dilation範囲

（復習）
逆に高い周波数成分だけ取り出すには？

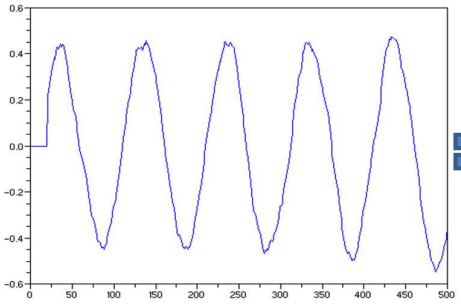
●ローパスフィルタ：低い周波数成分だけを取り出した

●元信号と低周波信号の差をとれば，高周波数成分だけ取り出せる？

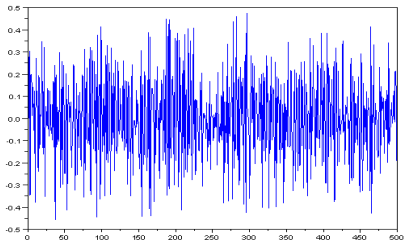
●入力f(t)



●ローパスx(t)



●高周波成分y(t)



画像の「エッジ抽出」

アイデア：低い周波数成分を取り除く

具体的には？
「変化」だけを取り出せば良い．

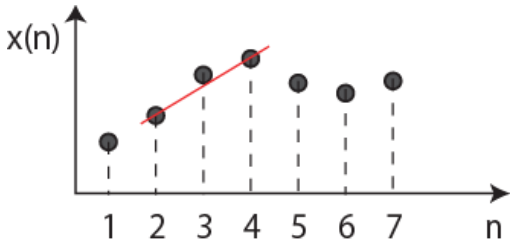
⇒空間的な微分を行っていることに等しい

対応：
- 微分＝エッジ抽出＝ハイパスフィルタ
- 積分＝平滑化＝ローパスフィルタ

微分： Sobelフィルタ

•ディジタルの世界：微分 ⇒ 差分

$$y(t) = \frac{dx(t)}{dt} \quad \Rightarrow \quad y(n) = \frac{x(n+1) - x(n-1)}{2\Delta}$$

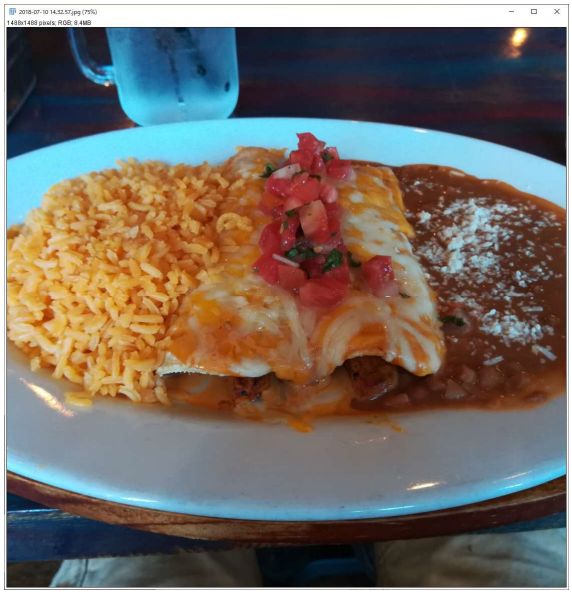


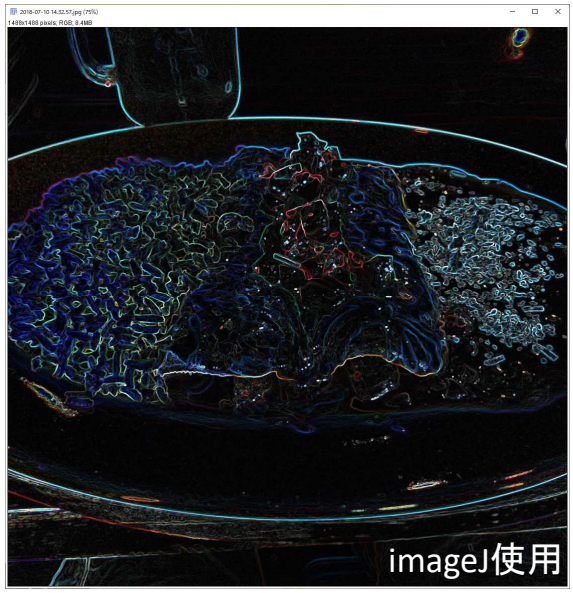
•2次元の微分：x方向，y方向がある．

$$\frac{\partial X}{\partial x} \quad \frac{\partial X}{\partial y}$$

$u_{i,j} = X_{i+1,j} - X_{i-1,j}$ $v_{i,j} = X_{i,j+1} - X_{i,j-1}$

Sobelフィルタ(2)





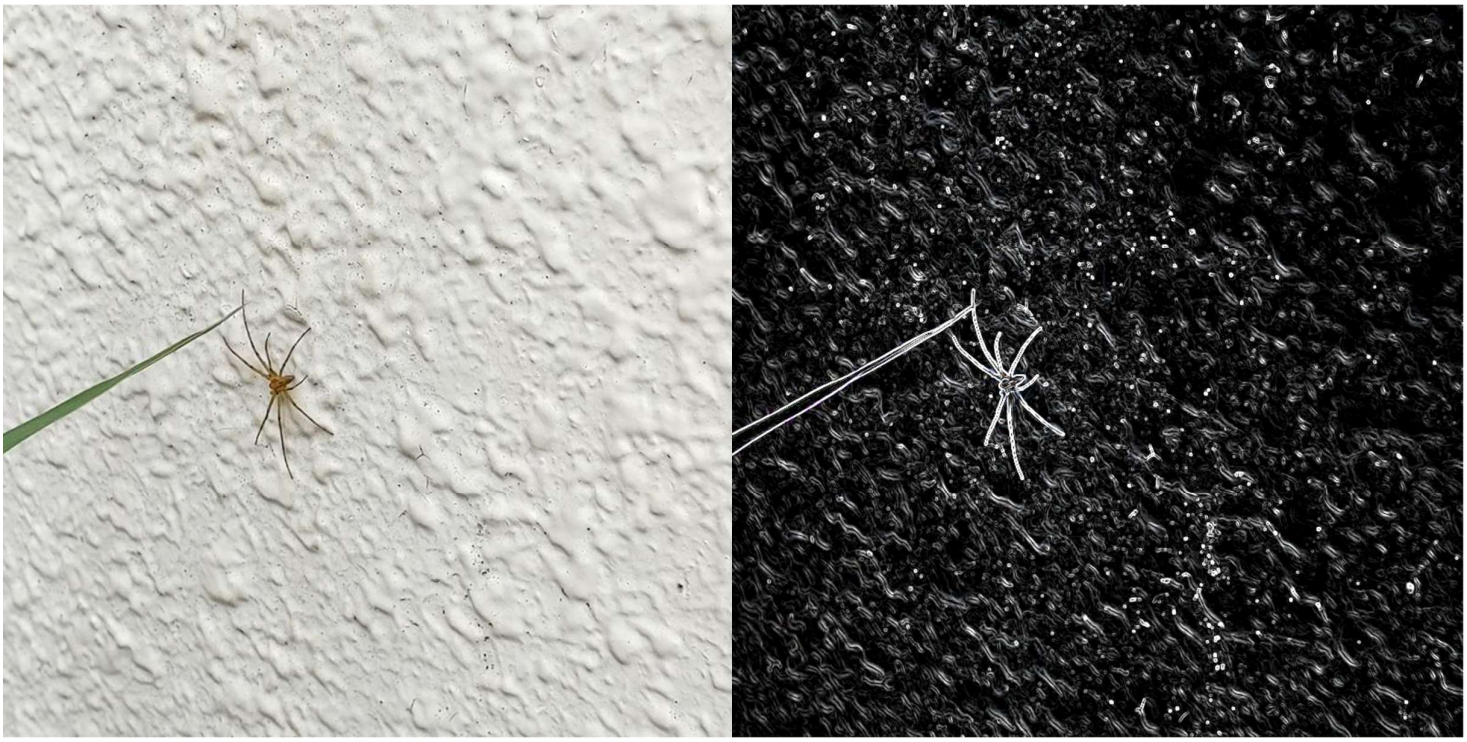
（1）X方向微分と，Y方向平滑化
（2）Y方向微分と，X方向平滑化
（3）(1)(2)の結果をベクトルとみなした時の大きさ＝変化の強さ
（（4）閾値により2値化）

$$\frac{\partial X}{\partial x} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{\partial X}{\partial y} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\sqrt{\left(\frac{\partial X}{\partial x}\right)^2 + \left(\frac{\partial X}{\partial y}\right)^2}$$

Sobelフィルタの動画適用例



ImageJ使用

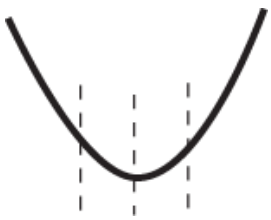
レポート課題（２）

元画像に3x3のSobelフィルタをかけ，エッジを抽出してみよ

ヒント

```
EdgeX=zeros(126,126);
for x=1:126,
  for y=1:126,
    EdgeX (x,y)= 略
end
end

EdgeY=zeros(126,126);
for x=1:126,
  for y=1:126,
    EdgeY(x,y)= 略
end
end

img2 = sqrt(EdgeX.*EdgeX + EdgeY .*EdgeY);
```

2階微分：Laplacianフィルタ

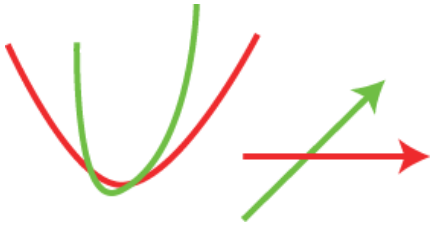


エッジ抽出＝空間的な微分
さらに微分すれば？　二階微分

$$y(t)=\frac{d^2x(t)}{dt^2} \Rightarrow y(n)=\frac{x(n+1)-x(n)}{\Delta}-\frac{x(n)-x(n-1)}{\Delta}$$
$$=\frac{x(n+1)-2x(n)+x(n-1)}{\Delta}$$

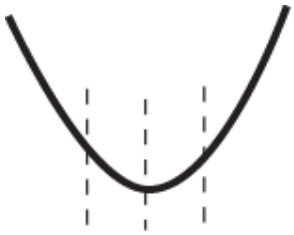
2次元では？

$$\nabla^2 X=\frac{\partial^2}{\partial x^2}X+\frac{\partial^2}{\partial y^2}X$$



2階微分：Laplacianフィルタ（続）

$$\nabla^2 X=\frac{\partial^2}{\partial x^2}X+\frac{\partial^2}{\partial y^2}X$$



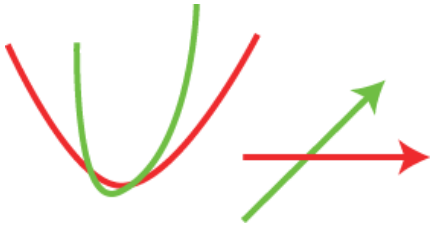
$\frac{\partial^2 X}{\partial x^2}$ □□□

$\frac{\partial^2 X}{\partial y^2}$ □
□
□

$u_{i,j} = X_{i+1,j} - 2X_{i,j} + X_{i-1,j}$

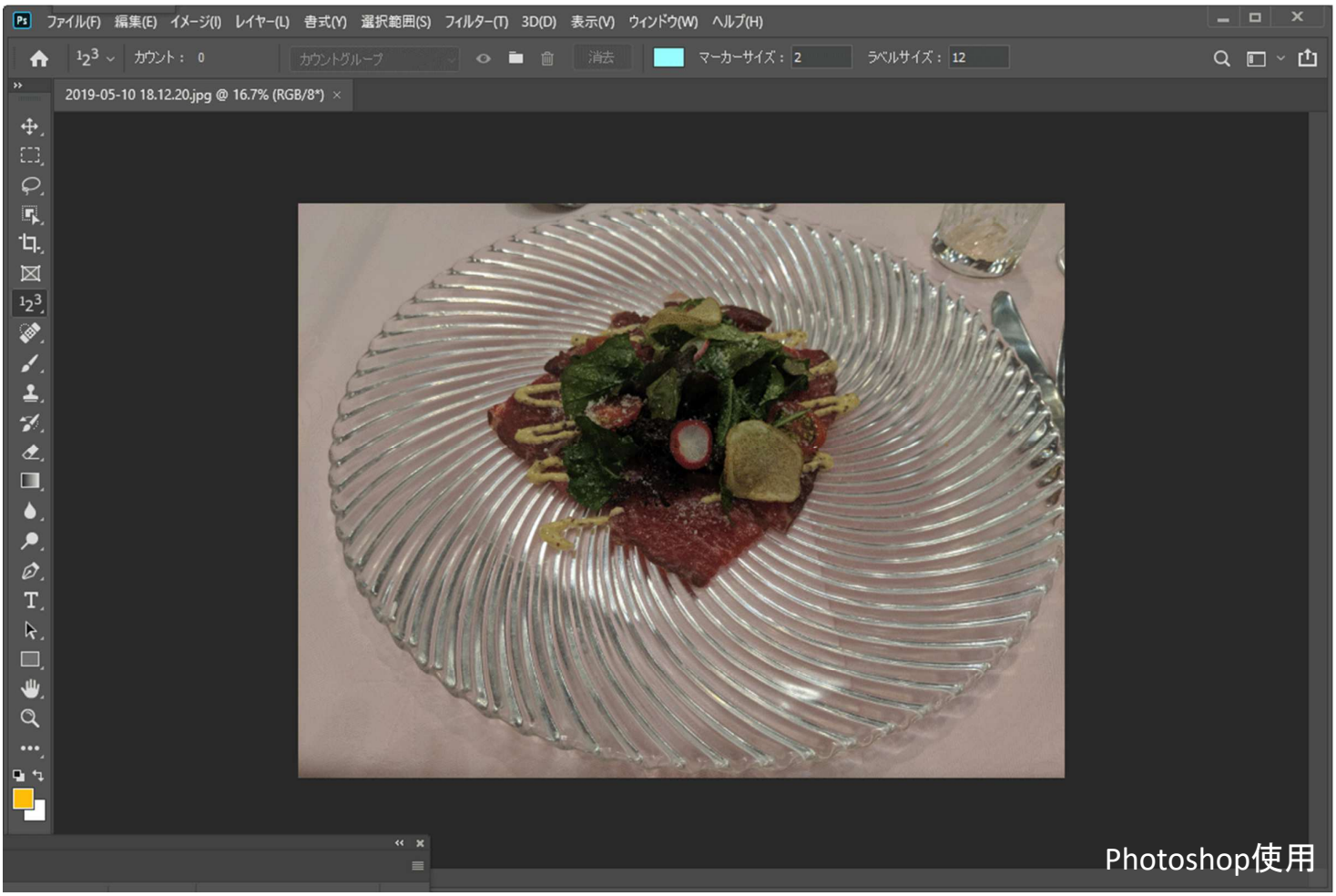
$v_{i,j} = X_{i,j+1} - 2X_{i,j} + X_{i,j-1}$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



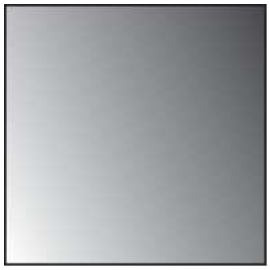
通常は $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ という形を用いることが多い

Laplacianによるエッジ抽出

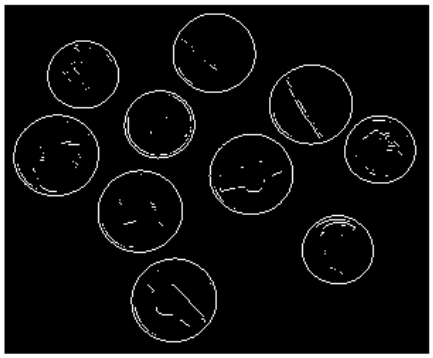


LoG (Laplacian of Gaussian)フィルタ

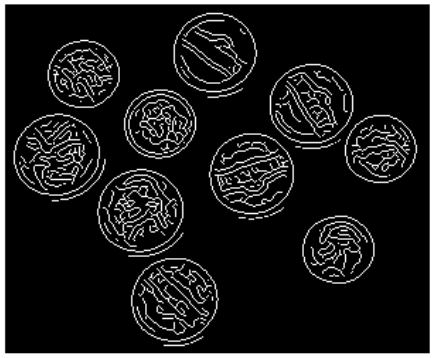
- Gaussianで平滑化後，Laplacianでエッジ抽出
- 人間の網膜上の情報処理そのもの
  - 人間はなだらかな輝度変化に鈍感
    (ex. 大型スクリーンの輝度分布)



（参考）エッジ抽出の実際：Cannyフィルタ



Sobel



Canny

エッジ抽出は通常，最後に2値化して終了，次の処理へ．

Sobelフィルタ：閾値の設定が難しい．

- 必要なエッジが消えてしまう or エッジが出過ぎる

Cannyフィルタ：最も標準的なエッジ抽出手法

- 微分計算自体はSobelの方法を使う
- 戦略：弱いエッジも，長く繋がりそうなら救う（二つの閾値使用）
- 計算量はやや多い．

相関と画像処理

テンプレートマッチング

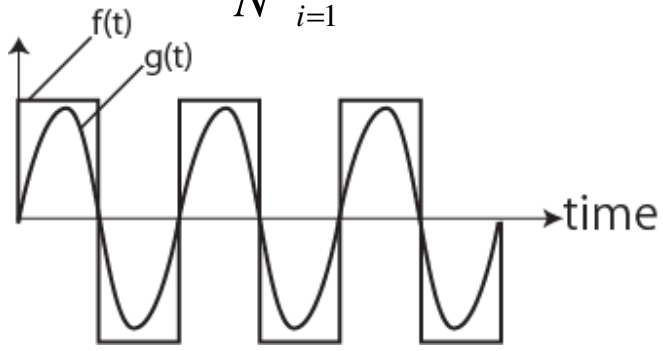
例：画像中から特定の人の顔を認識したい

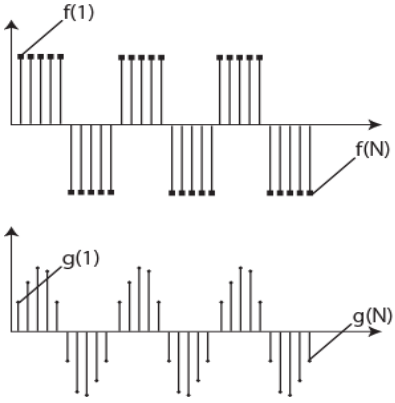
（復習）波形fに波形gはどれだけ含まれるか

波形f中の，波形gの成分

$$= \frac{1}{T}\int_0^T f(t)g(t)dt \quad \text{(連続関数)}$$

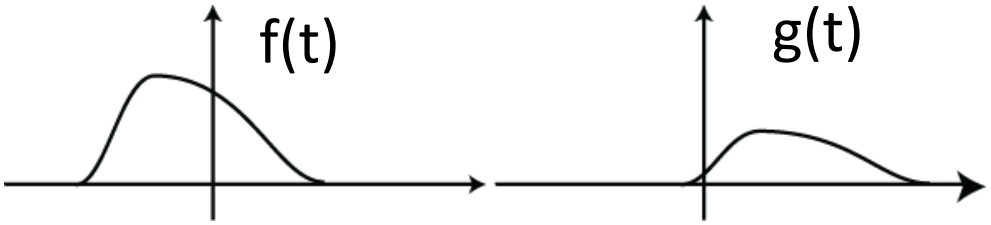
$$= \frac{1}{N}\sum_{i=1}^{N} f(i)g(i) \quad \text{(離散化して考えた場合)}$$





これは二つの波をベクトルと考えた時の内積に他ならない
※内積を連続関数に対して定義

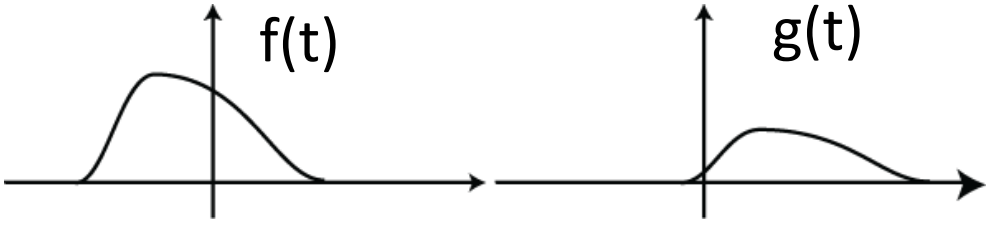
（復習）相互相関



＜問題＞
二つの信号が，
●時間的にどれだけずれているのか
●時間のずれを無視したらどれだけ似ているのか
を測定したい．

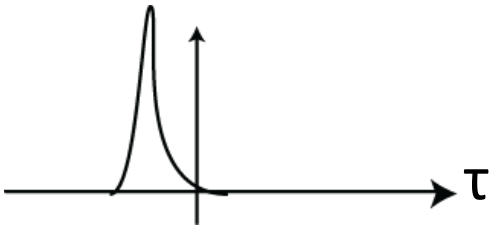
内積を思い出せば，
次の手順で測定すればよいことがわかる
●g(t)をτだけずらしてみる　⇒　$g(t+\tau)$
●f(t)との内積を取ってみる　⇒　$\int_{-\infty}^{\infty} \overline{f(t)} g(t+\tau) dt$
●τを変化させていく．

（復習）相互相関



$R_{fg}(\tau)$：二つの関数f(t)，g(t)の，相互相関関数

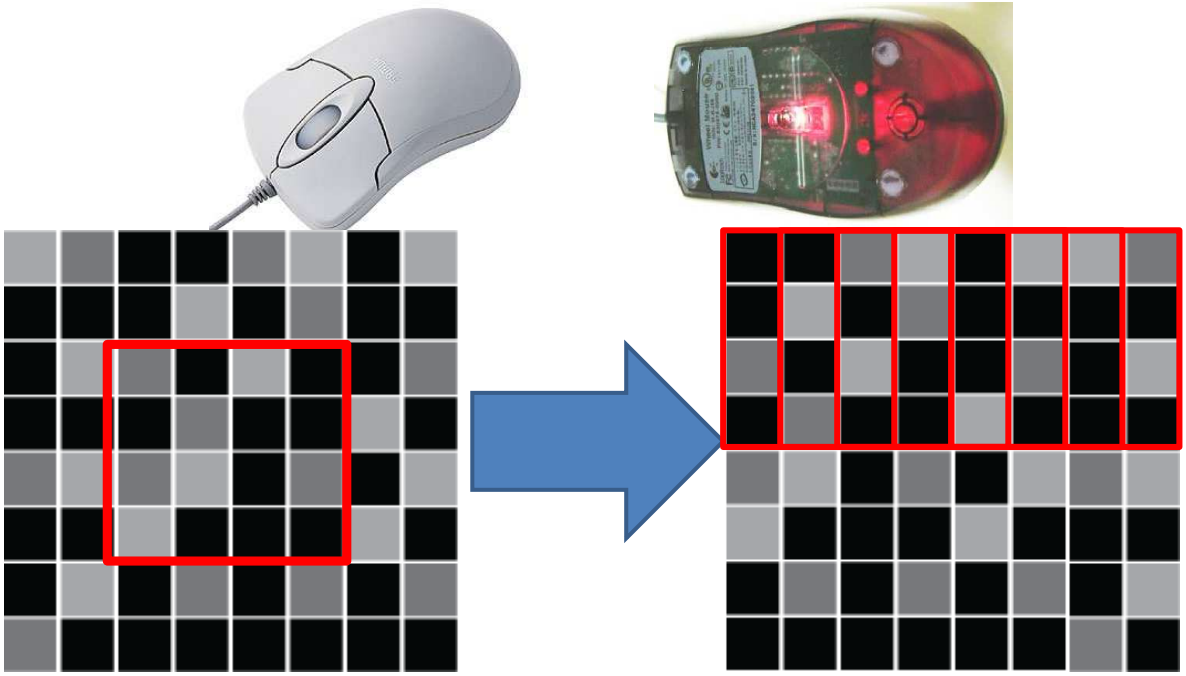
$$R_{fg}(\tau) = \int_{-\infty}^{\infty} \overline{f(t)} g(t+\tau) dt$$



$R_{fg}(\tau)$が最大の値をとるτ＝元の関数f(t)とg(t)のズレ
（ただし直流成分を取り除いた後）

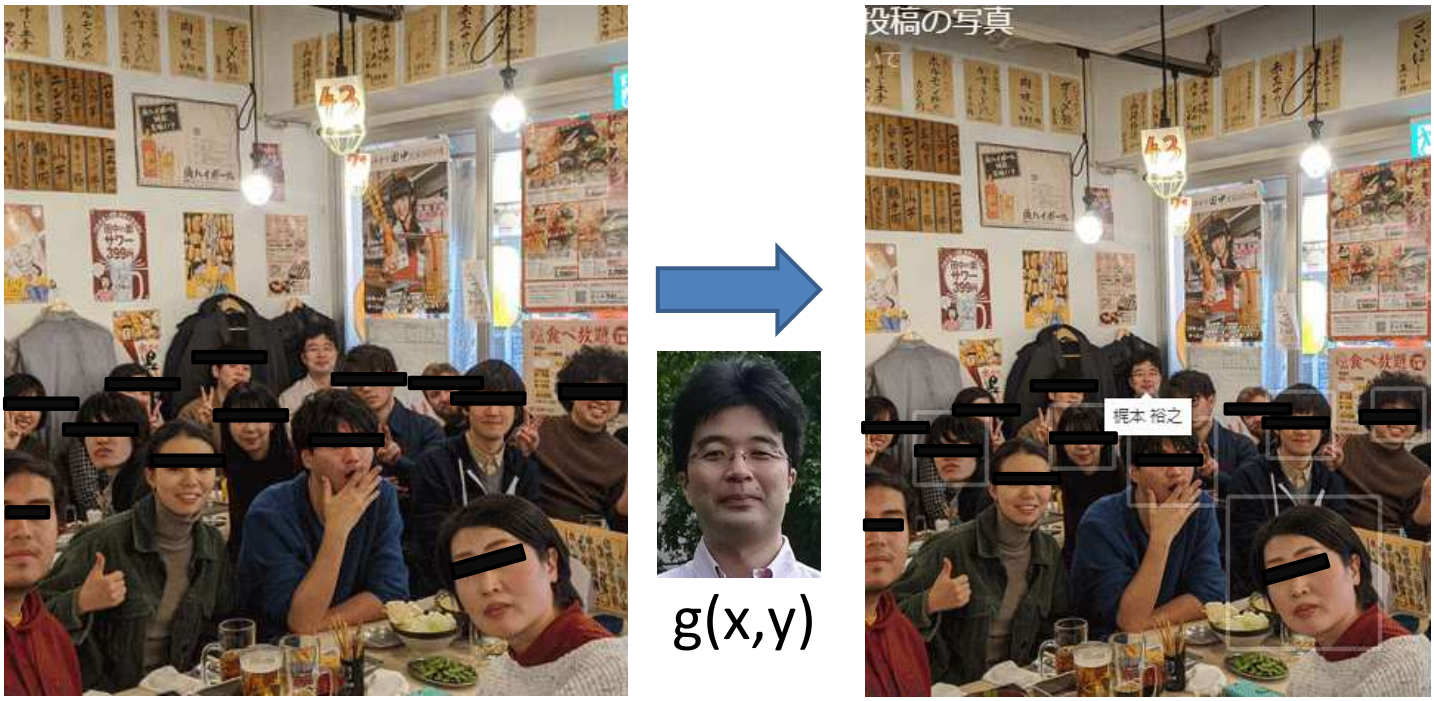
（復習）相互相関の応用：速度計測

光学式マウスの中身＝16x16 pixel のCMOSカメラ



二つの画像（＝2次元関数）同士の相互相関を取ることで
移動量を計測する．自動車の速度計測等にも利用．

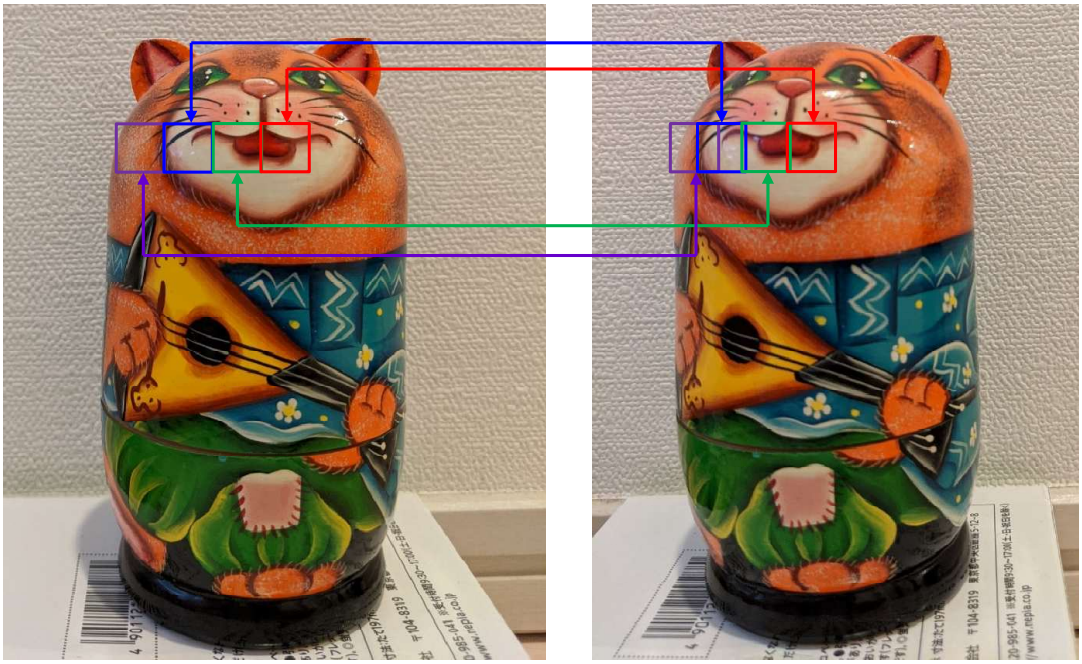
テンプレートマッチング（再）



g(x,y)

2次元に拡張．
g(t)⇒g(x,y)として，顔の標準的な画像を用意して相互相関をとれば，顔の部分でピークを生じる

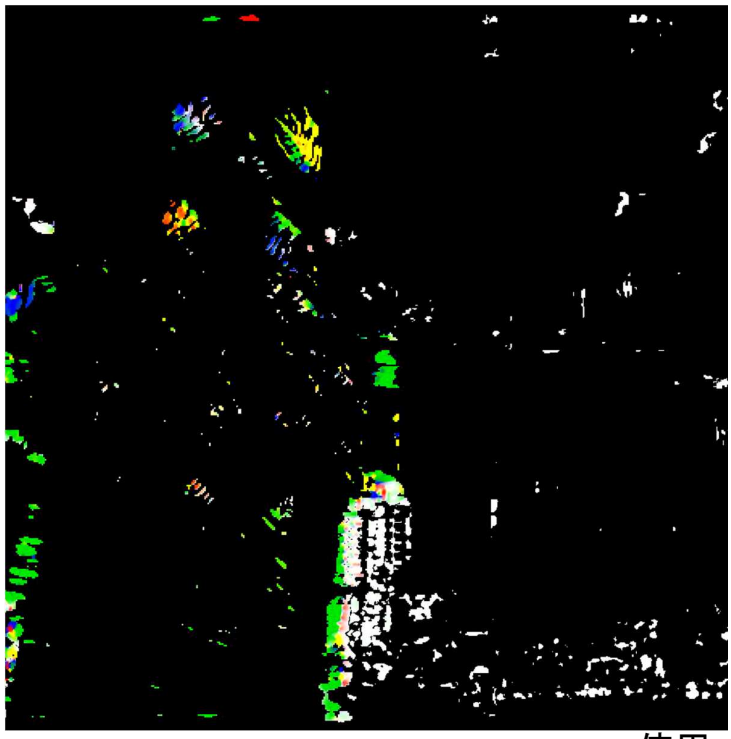
（参考）ステレオビジョンによる立体計測



左目映像　　右目映像

- 二つ以上のカメラを使用
- 画像間の局所的な相互相関から視差計測、立体再構成

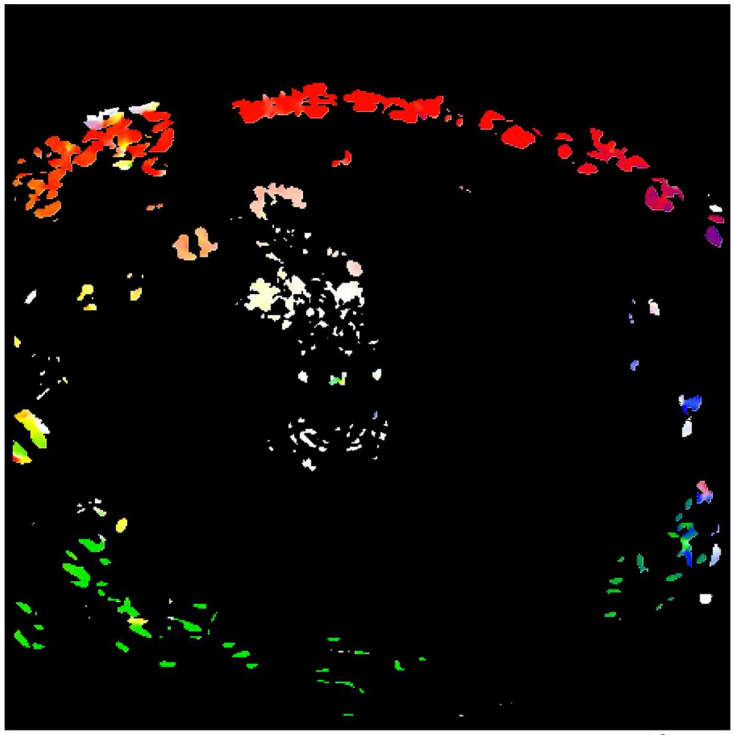
Optical Flow



ImageJ使用

- 時間的に前後の2枚以上の画像で相互相関を計測

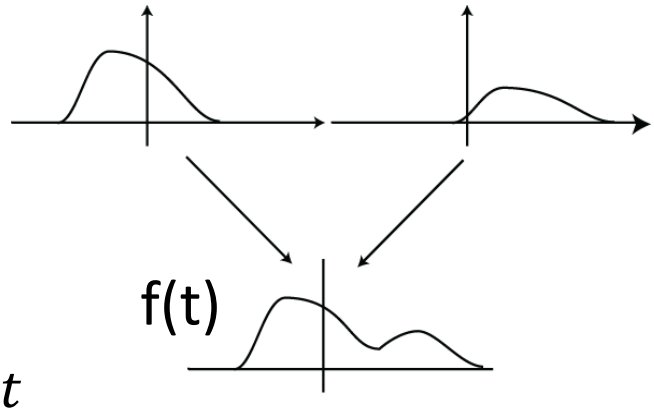
Optical Flow



ImageJ使用

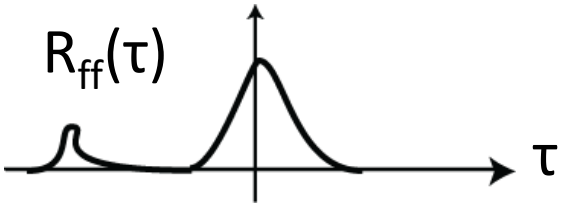
- 時間的に前後の2枚以上の画像で差を計測

（復習）自己相関



二つの関数f(t), g(t)の代わりに,
ひとつの関数f(t)の相関を取る.

$$R_{ff}(\tau) = \int_{-\infty}^{\infty} \overline{f(t)} f(t+\tau) dt$$



自己相関関数は,
「どれだけずれたら自分自身に近い形になるか」
を表す.
すなわち, エコーを発見していることに他ならない.

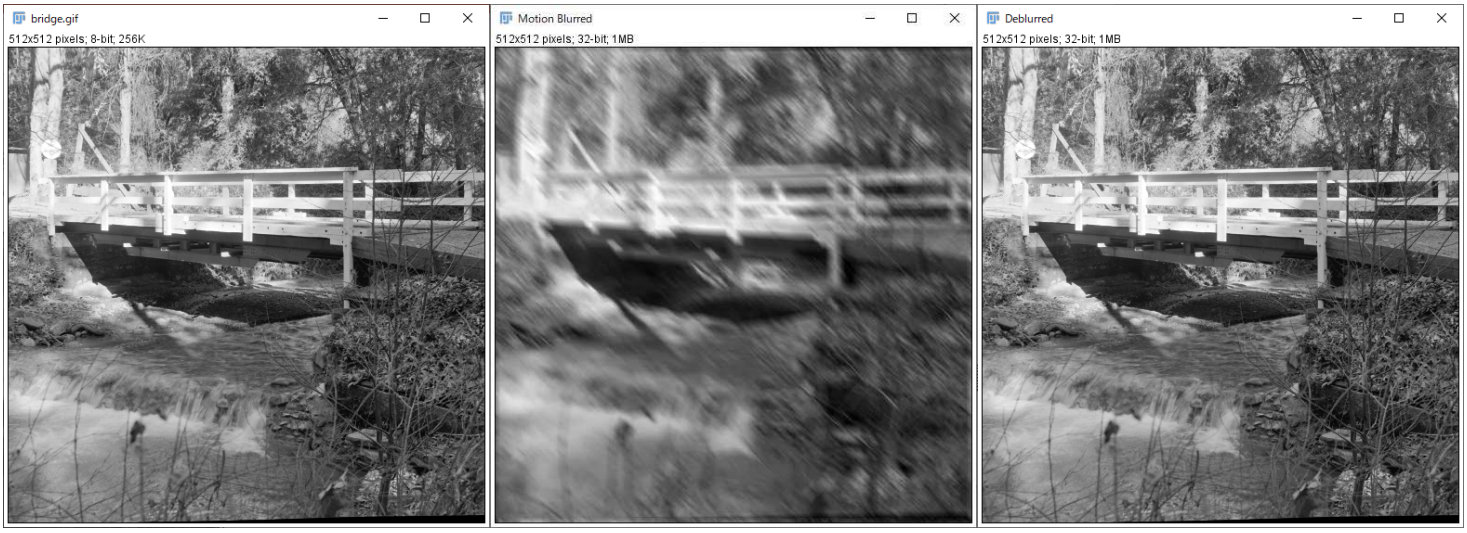
ブレ（Motion Blur)について

Motion Blur (Wikipedia) https://en.wikipedia.org/wiki/Motion_blur

ブレ(Motion Blur)：
カメラを使って、イメージを捕らえる過程中での移動、
または、長い露光時間を使う場合の被写体の移動。

ブレの検出と除去

ImageJ使用



Ground Truth　　Motion Blur　　Reconstructed

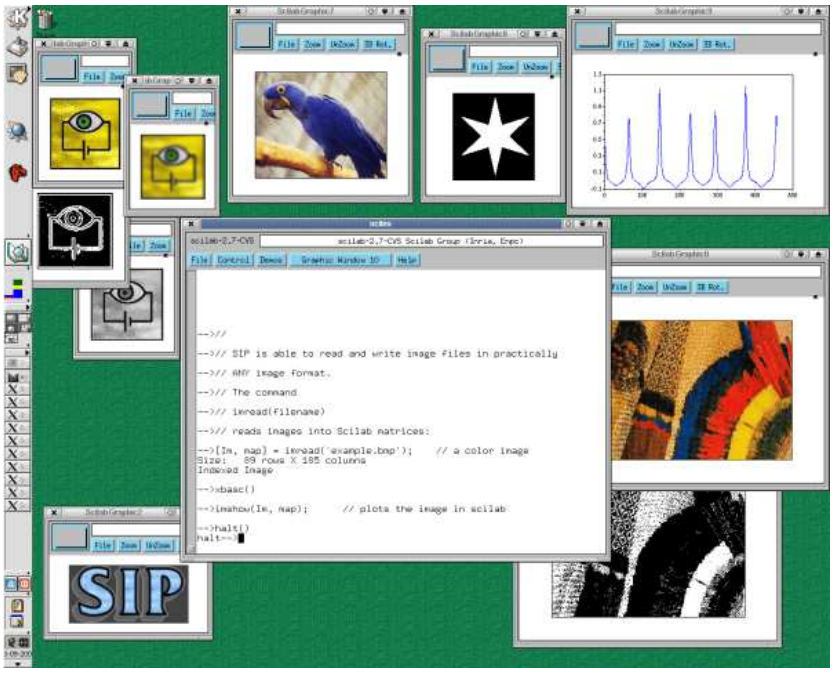
基本原理
（1）2次元の自己相関計算によってブレの方向と量を推定
（2）ブレの方向に微分フィルタを適用する
実際はもうすこし複雑

画像処理を使うために

Scilabでの画像処理

SIP＝Scilab Image Processing toolbox

http://siptoolbox.sourceforge.net/



画像の読み出しと保存が可能.

普通に知られているアルゴリズムは大体ある.

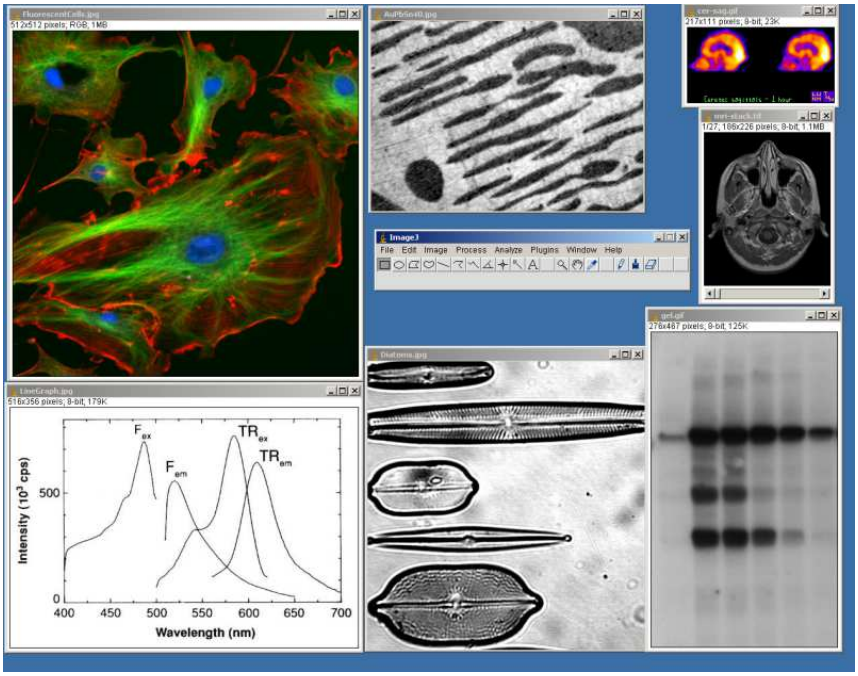
画像処理ライブラリ OpenCV

世界で最も広く使われている画像処理ライブラリ
これにより画像処理研究はソフト開発から解放された.

日本語の情報源
・Webページ（奈良先端大）
http://opencv.jp/
・OpenCVプログラミングブック

画像処理プログラミングは配列を扱うため，自力でプログラミングするとバグに苦しみます．まずはライブラリを使いましょう．（ただし最終的にはライブラリを作る人へ）

研究用画像処理ツール ImageJ



ImageJ (Wikipedia) https://en.wikipedia.org/wiki/ImageJ

1.Rasband, W.S., ImageJ, U. S. National Institutes of Health, Bethesda, Maryland, USA, http://imagej.nih.gov/ij/, 1997-2012.
2.Schneider, C.A., Rasband, W.S., Eliceiri, K.W. "NIH Image to ImageJ: 25 years of image analysis". Nature Methods 9, 671-675, 2012.

画像に対する計測や加工を行うソフトウエア。すべてオープンソース。生物系等幅広い分野で使用されている。

中間確認テスト

中間テスト用の問題集をwebに置きました。

一度式の導出を覚えることを意図しています。

場所は通常通り西5-109で行います。