

デジタルメディア処理

担当: 井尻 敬

デジタル画像のフィルタリング

フィルタ処理: トーンカーブ, 線形フィルタ

達成目標

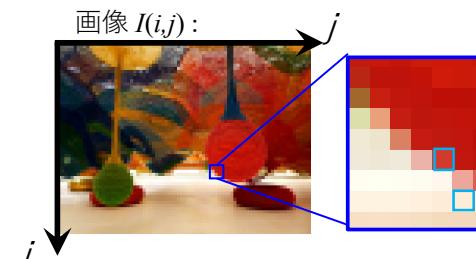
- ・画素ごとの変換である**トーンカーブ**の機能と効果を説明できる
- ・**線形フィルタ**の機能と効果を説明できる

Contents

- ・トーンカーブ
 - ・反転, 二値化, ポスタリゼーション, ソラリゼーション, ガンマ変換, カラー画像
- ・線形フィルタ
 - ・平滑化フィルタ, ソーベルフィルタ, ガウシアンフィルタ, ラプラシアンフィルタ

デジタル画像: カラー画像

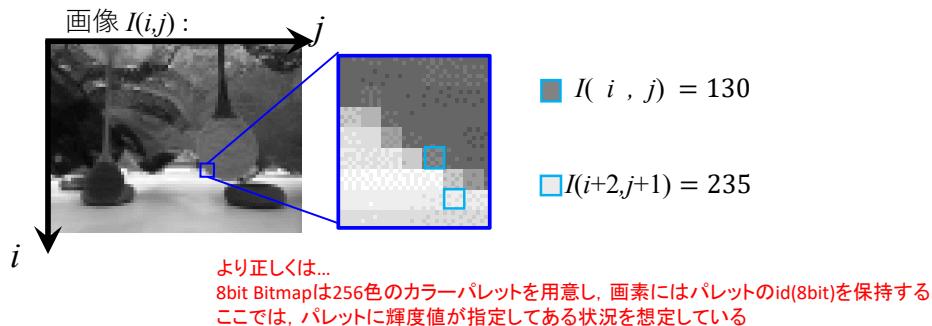
- ・離散値を持つ画素が格子状に並んだデータ
- ・画素: $\text{pixel} = \text{picture} + \text{element}$
- ・例 24bit bitmap : 各pixelが(R,G,B)毎に8bit[0,255]の整数値を持つ



※原点位置は左下のことも

デジタル画像：グレースケール画像

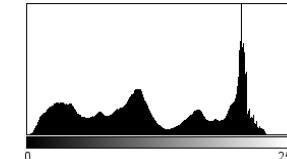
- 離散値を持つ画素が格子状に並んだデータ
- 画素 : pixel = picture + element
- 例 8bit bitmap : 各pixelが8bit [0,255]の整数値を持つ



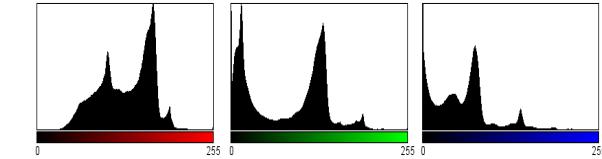
『頻度表（ヒストグラム）』とは

各階調の画素数を数えた表のこと
回転や平行移動に依存しない特徴量 → 画像処理に頻出

グレースケール画像

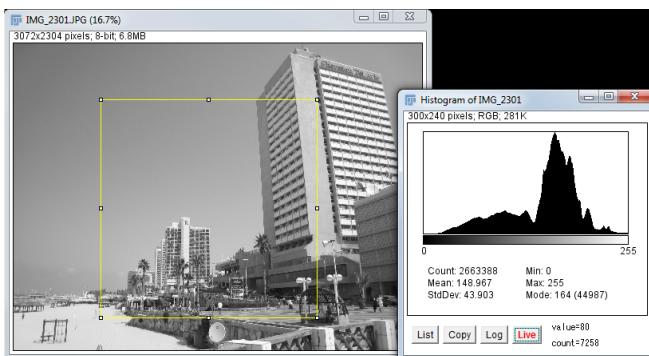


RGBカラー画像



ImageJでヒストグラムを確認してみる

1. ImageJ 起動
2. 画像読み込み
3. Menu > analyze > histogram
4. LiveをOnにすると矩形選択した領域のヒストグラムを確認可能



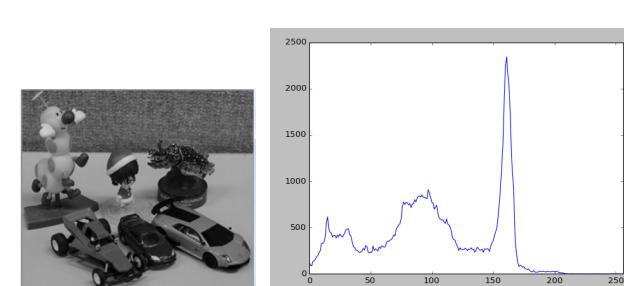
```
import numpy as np
import pylab as plt
import cv2
import itertools
#画像読み込み & グレースケール化
img = cv2.imread("imgs/sample.png")
img_gry = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#histogram生成
hist = np.zeros(256)
for y in range(img_gry.shape[0]):
    for x in range(img_gry.shape[1]):
        hist[img_gry[y,x]] += 1

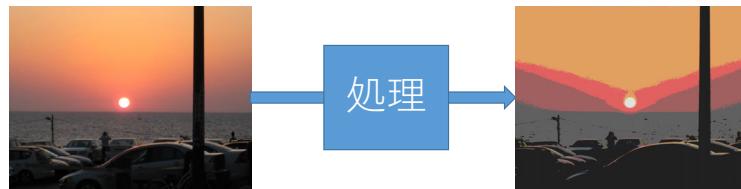
#windowを生成して画像を表示
cv2.imshow("Image", img_gry)

#histをmatplotlibで表示
plt.plot(hist)
plt.xlim([0,256])
plt.show()
```

ヒストグラムの計算： histogram.py



デジタル画像のフィルタリング



入力画像に対し計算処理を施し、所望の画像に変換する

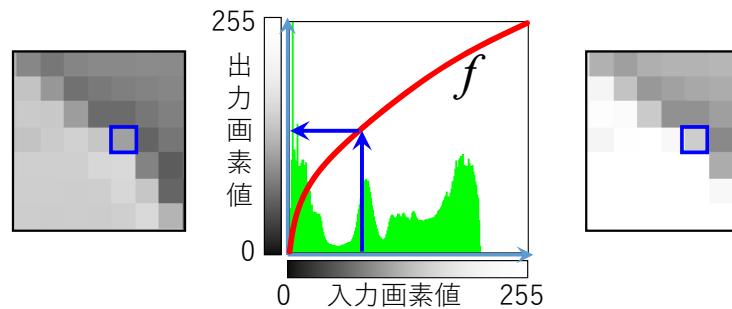
- ・特定の周波数を強調する・捨てる(ノイズ除去)
- ・画像処理(ステレオ視・領域分割・識別器)の前処理として利用する
- ・アーティスティックな効果を得る

トーンカーブ

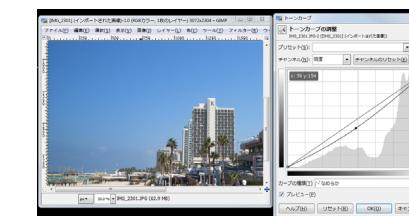
トーンカーブ

ToneCurve.exe (C++)
Image>Adjust>Window/Level (ImageJ)

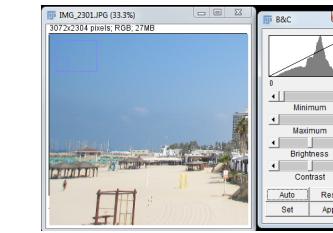
- ・画像は8bit グレースケールとする
- ・各画素の値を異なる値に変換する**階調変換関数**を考える
 - ・新しい画素値 = $f(\text{画素値})$
- ・階調変換関数を曲線で表現したものを**トーンカーブ**と呼ぶ



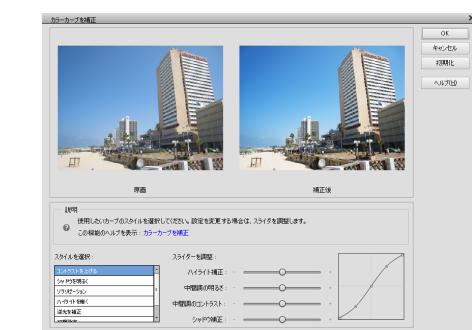
トーンカーブは写真編集の基本ツール



GIMP

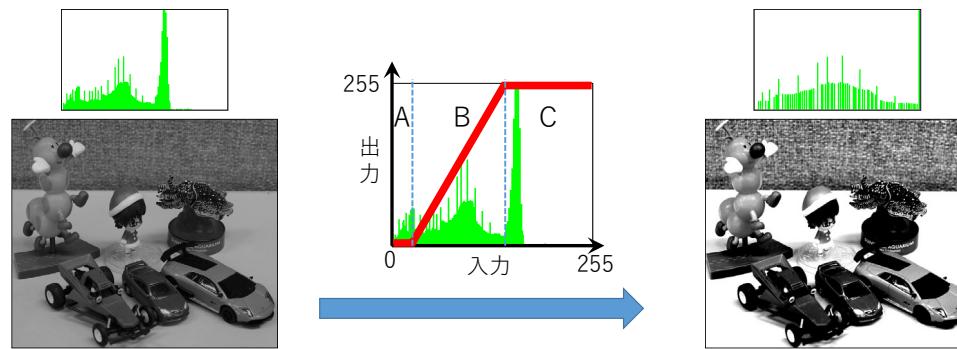


ImageJ: 自由編集でないのでちょっと違うけど



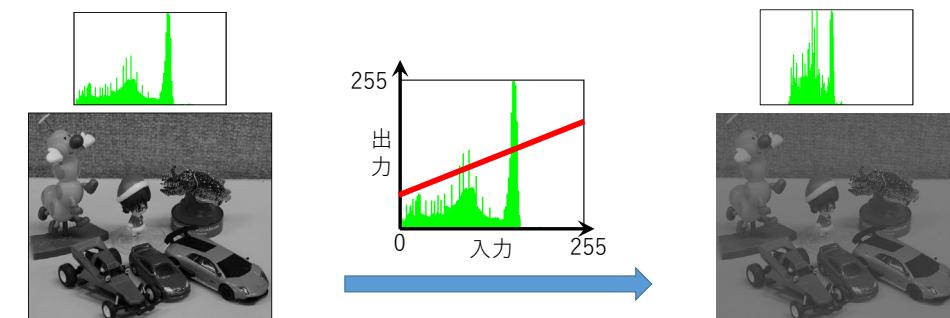
Photoshop Elements カラーカーブ
使いやすいように自由度の限られたトーンカーブのようなもの
Photoshop CSにはトーンカーブがある（あった）

トーンカーブ: コントラストを上げる



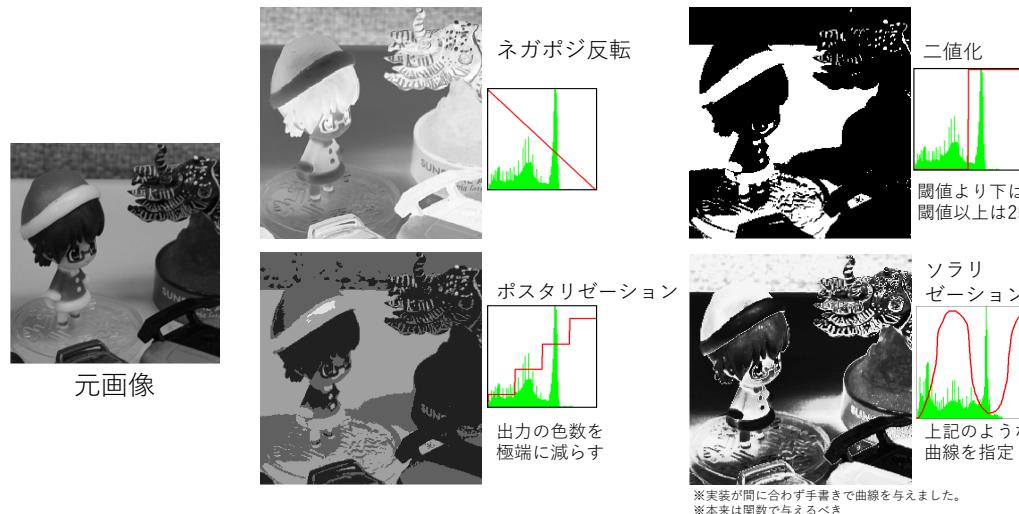
- ・領域A：出力画素値0となり黒つぶれ
- ・領域C：出力画素値255となり白飛び
- ・領域B：傾きが1より大きいため、画素値の取り得る範囲が広がりコントラストが上がる
画素値は離散値であるため出力ヒストグラムは飛び飛びに

トーンカーブ: コントラストをさげる



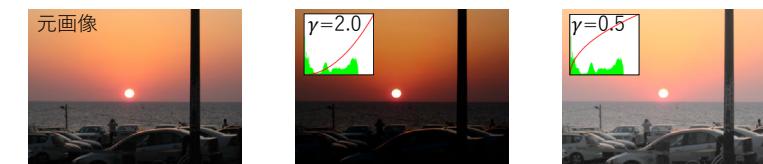
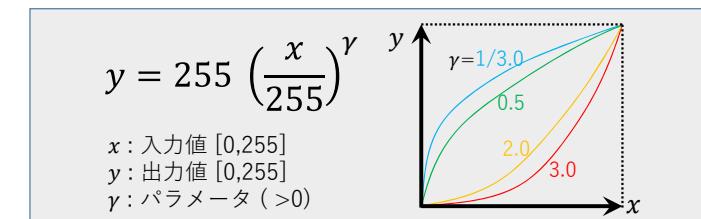
- ・傾きが1より小さいため、出力画素値の取り得る範囲が縮まり、コントラストが下がる

トーンカーブ: 特殊効果



トーンカーブ: ガンマ変換（ガンマ補正）

次のトーンカーブを利用した濃淡変換をガンマ変換と呼ぶ



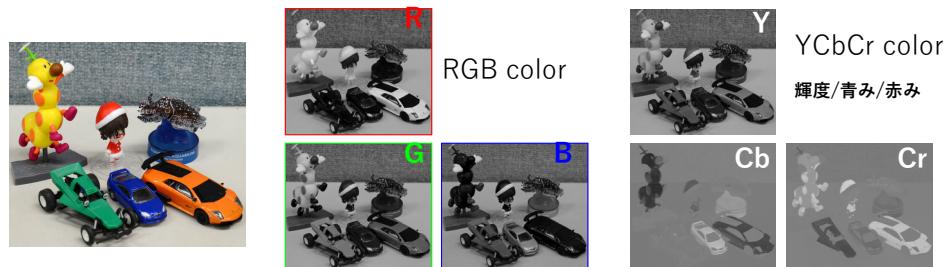
※ RGB各チャンネルに
ガンマ補正を適用

※ 画像出力デバイスには『出力値 = (入力値) $^\gamma$ 』と言う関係があり、この特性を補正する目的で上記の関数が用いられていた。これを画像の補正に利用したのがガンマ変換

トーンカーブ：カラー画像への適用

カラー画像をトーンカーブで編集するとき …

- RGBの各チャンネルにトーンカーブの画素値変換を適用
- YCbCr Colorに変換し輝度値成分 (Y) のみに変換を適用
- その他



トーンカーブ：カラー画像への適用



入力画像



RGB各チャンネル



YCbCrの輝度Yのみ

トーンカーブ：カラー画像への適用

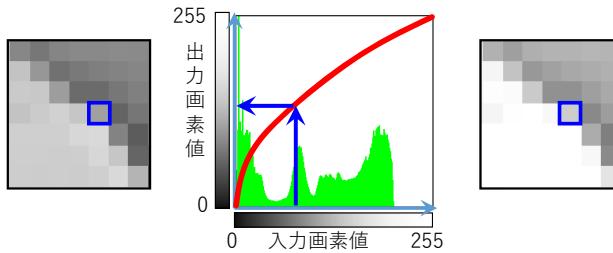


トーンカーブ：カラー画像への適用



トーンカーブ：まとめ

- トーンカーブ：各画素の輝度値・色を変換する階調変換関数
- 画像の見栄えの編集に利用される
- キーワード：コントラスト変換・ネガポジ反転・ポスタリゼーション・ソラリゼーション・2値化・ガンマ補正



線形フィルタ



線形フィルタの計算

Convolution1.py



線形フィルタの例

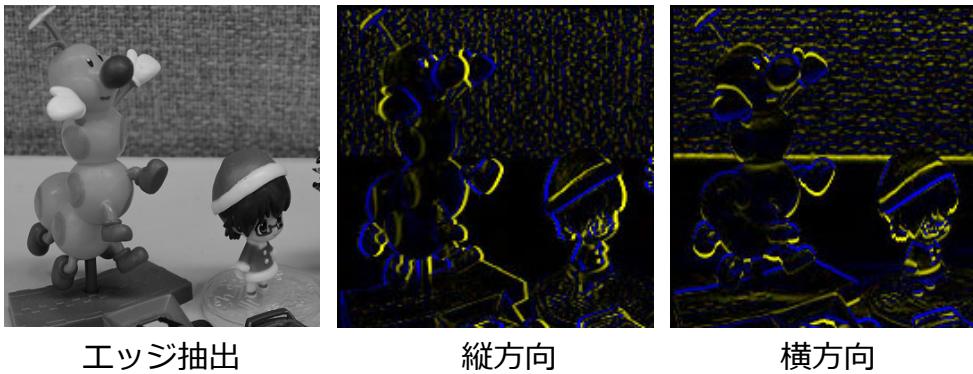


ぼかす



鮮鋭化

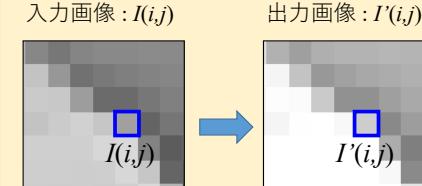
線形フィルタの例



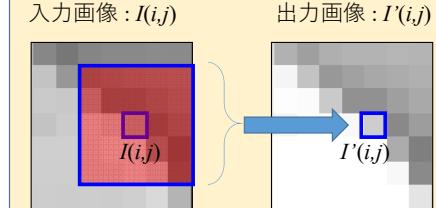
空間フィルタとは

- ・空間フィルタとは周囲の情報を利用して画素値を決めるフィルタ
- ・空間フィルタは、線形フィルタと非線形フィルタに分けられる

トーンカーブ:
出力画素 $I'(i,j)$ を求めるのに
入力画素 $I(i,j)$ のみを利用

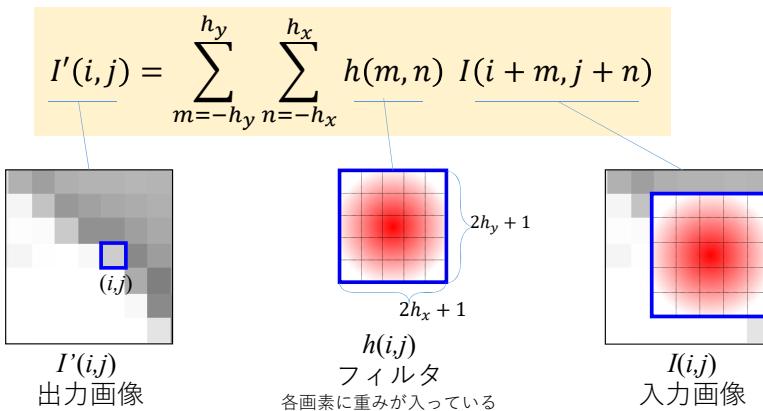


空間フィルタ:
出力画素 $I'(i,j)$ を求めるのに
入力画素 $I(i,j)$ の周囲画素も利用

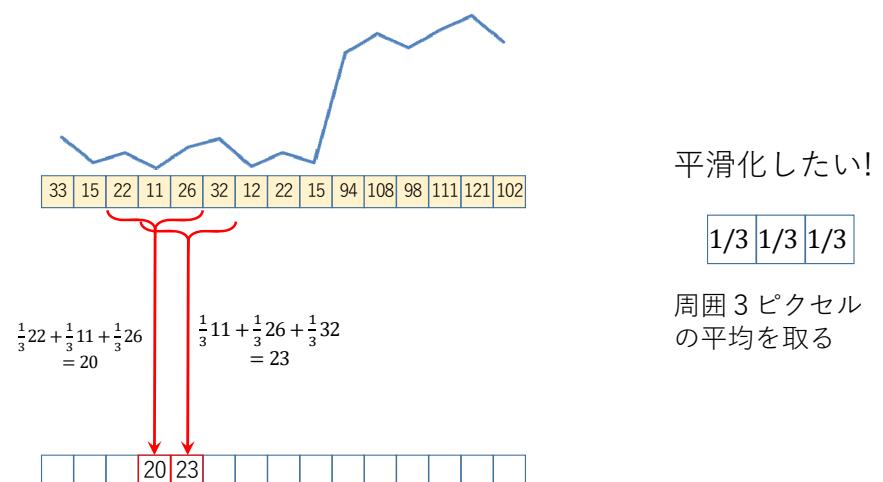


線形フィルタとは

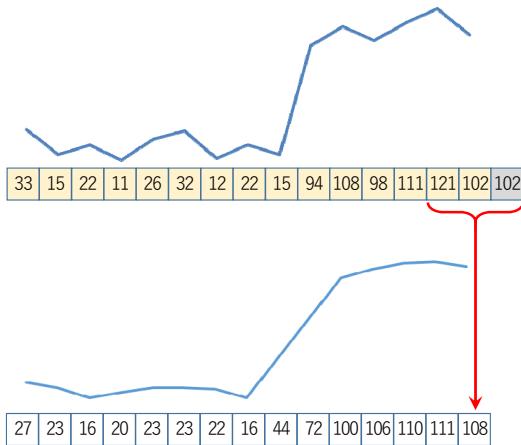
出力画素値を、入力画像の周囲画素の重み付和で計算する



線形フィルタの例 1D



線形フィルタの例 1D



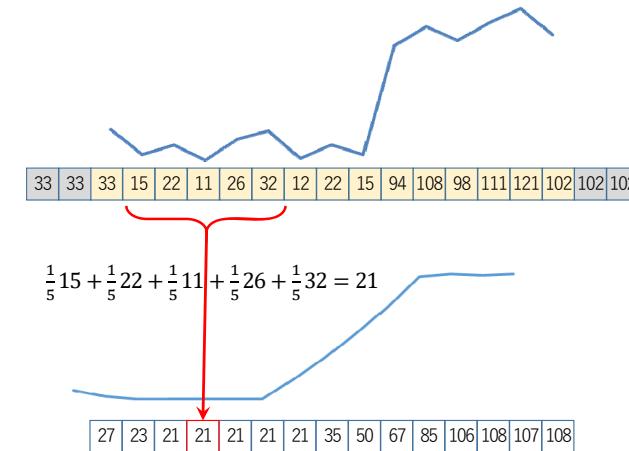
※端ははみ出るので値をコピー（ほかの方法もある）

平滑化したい！

$$\frac{1}{3} \frac{1}{3} \frac{1}{3}$$

周囲 3 ピクセル
の平均を取る

線形フィルタの例 1D

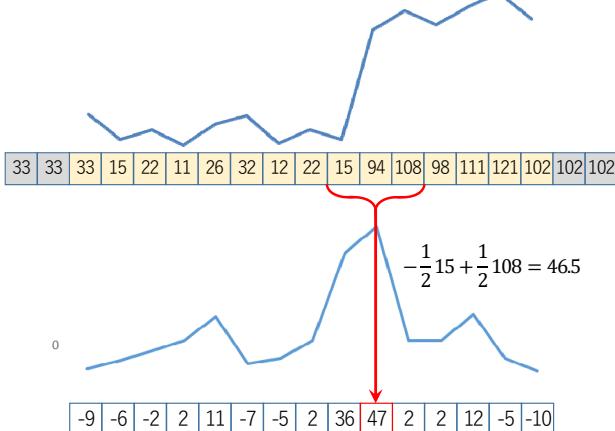


もっと
平滑化したい！

$$\frac{1}{5} \frac{1}{5} \frac{1}{5} \frac{1}{5} \frac{1}{5}$$

周囲 5 ピクセル
の平均を取る

線形フィルタの例 1D



※端ははみ出るので値をコピー（ほかの方法もある）

エッジ
(変化の大きい部分)
を検出したい

$$-0.5 \ 0 \ 0.5$$

右と左のピクセルの
差をとる

線形フィルタ：平滑化



$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$



線形フィルタ：ガウシアンフィルタ

係数をガウス分布に近づけ
中央ほど強い重みに

$$\frac{1}{16} \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

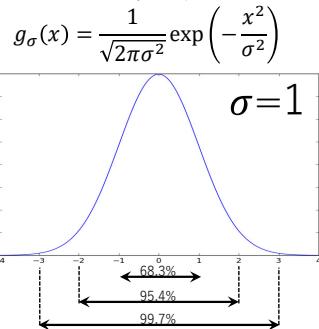
$$\frac{1}{256} \begin{array}{|c|c|c|c|c|} \hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$



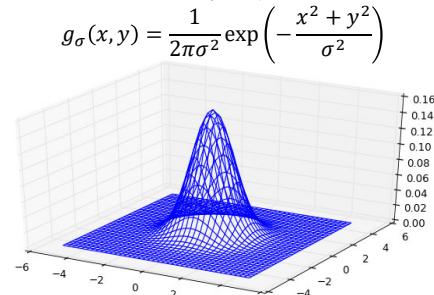
LinaerFilter.exe (C++)
convolution1.py (python)
Process>Filters>Convolve (ImageJ)

線形フィルタ：ガウシアンフィルタ

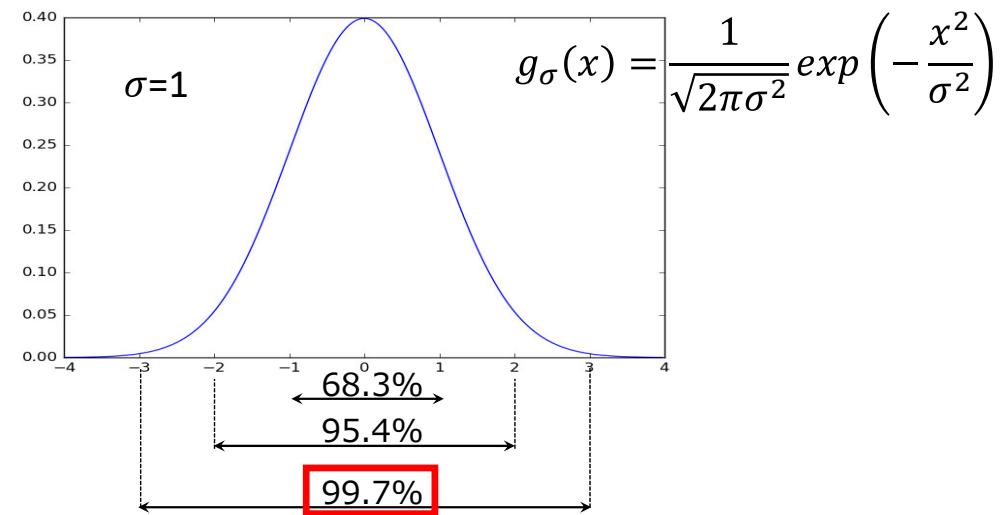
ガウス関数1D



ガウス関数2D



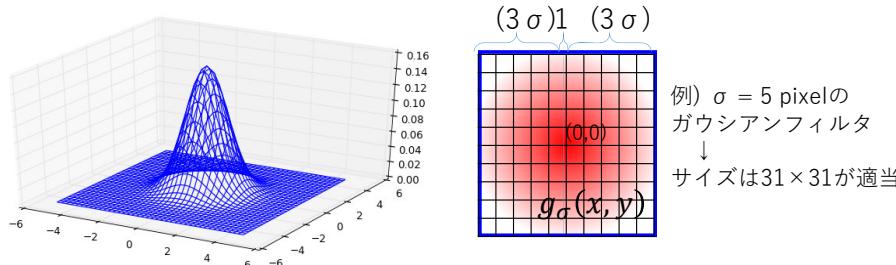
これを重みにして線形フィルタをしたい
さすがに3x3は精度が悪くない？？



線形フィルタ：ガウシアンフィルタ

課題によっては『 3×3 』や『 5×5 』の窓では精度が悪い

→ 精度を出すには窓の半径を 3σ 程度にすべき
(計算時間はかかる)



例) $\sigma = 5$ pixel の
ガウシアンフィルタ
↓
サイズは 31×31 が適当

線形フィルタ：エッジ検出（微分）

関数 $f(x, y)$ の x 軸, y 軸方向の偏微分は以下の通り定義され、

※微分の復習。

$$\frac{\partial f(x, y)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h, y) - f(x-h, y)}{2h}$$

$$\frac{\partial f(x, y)}{\partial y} = \lim_{h \rightarrow 0} \frac{f(x, y+h) - f(x, y-h)}{2h}$$

練習.

$$f(x, y) = -2x^2 - y^2$$

上記の関数の(1,1), (2,3)

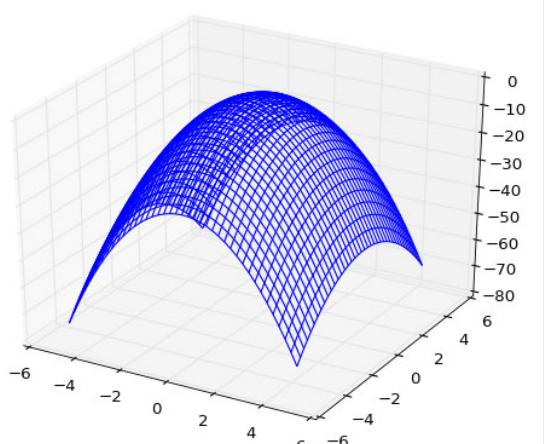
における勾配を計算し、
さらに図示せよ

また, $f(x, y)$ の勾配 $\nabla f(x, y)$ は 2 次元ベクトルであり,

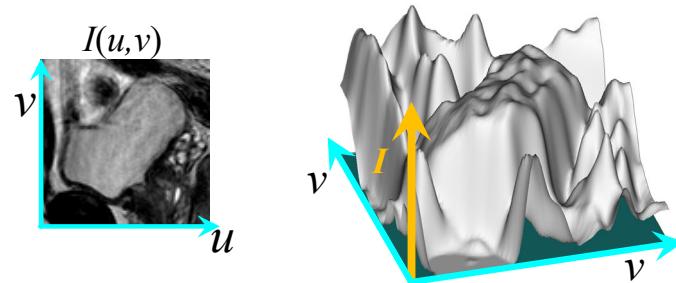
$$\nabla f(x, y) = \begin{pmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{pmatrix}$$

点 (x, y) において $f(x, y)$ の増加が一番大きくなる方向を示す

$$f(x, y) = -2x^2 - y^2$$



線形フィルタ：エッジ検出（微分）



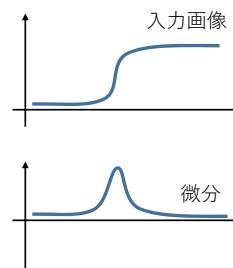
グレースケール画像 $I(u,v)$ は、高さ関数 $z = I(u,v)$ と見なせる
なので関数 $I(u,v)$ の勾配（微分）は計算できそう
 $I(u,v)$ の勾配は、画像の変化の大きい方向を表す

画像の出典 [Ijiri et al 2013, Eurographics]

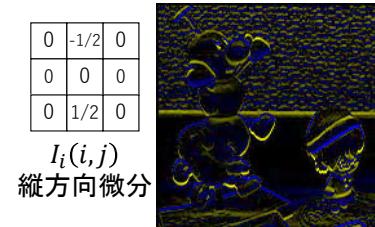
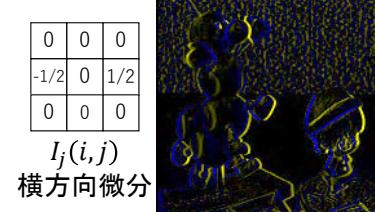
線形フィルタ：エッジ検出(微分)



$I(i,j)$ 入力画像



微分フィルタには画像のエッジ
で強く応答する



線形フィルタ：エッジ検出（微分）

2次元関数 $f(x,y)$ の x 方向偏微分

$$\begin{aligned} f_x &= \frac{\partial f(x,y)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h,y) - f(x,y)}{h} \\ &= \lim_{h \rightarrow 0} \frac{f(x,y) - f(x-h,y)}{h} \\ &= \lim_{h \rightarrow 0} \frac{f(x+h,y) - f(x-h,y)}{2h} \end{aligned}$$

画像 $I(i,j)$ の横方向偏微分（近似）

$$I_j(i,j) \approx I(i,j+1) - I(i,j) \quad \dots(a)$$

$$\approx I(i,j) - I(i,j-1) \quad \dots(b)$$

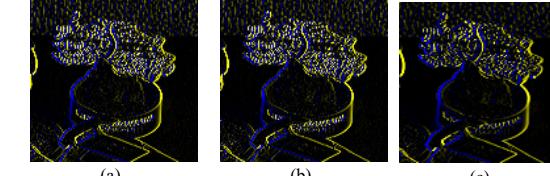
$$\approx \frac{I(i,j+1) - I(i,j-1)}{2} \quad \dots(c)$$

$\ast h = pitch$ (画素サイズ) = 1 と近似



※ 正値: 黄色,
負値: 青 で可視化

$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ -1/2 & 0 & 1/2 \\ 0 & 0 & 0 \end{bmatrix}$
(a)	(b)	(c)



線形フィルタ : エッジ検出（微分）

- 前述の単純なフィルタはノイズにも鋭敏に反応する
 - ノイズを押さえつつエッジを検出するフィルタが必要
- 横方向微分 : 横方向微分し 縦方向平滑化する
縦方向微分 : 縦方向微分し 横方向平滑化する

Prewitt filter

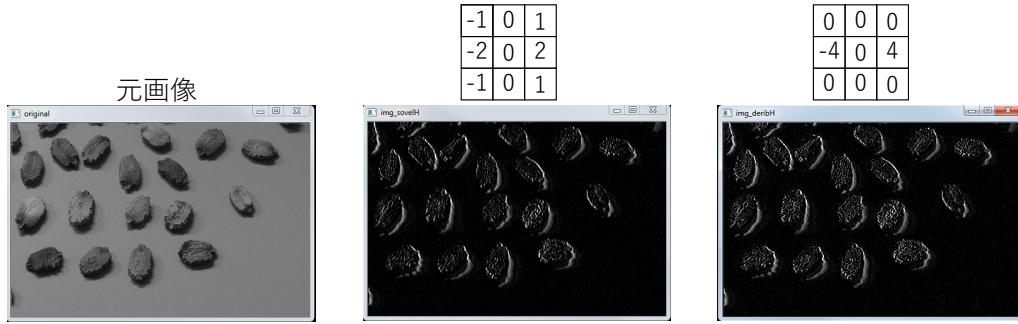
-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Sobel filter

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1



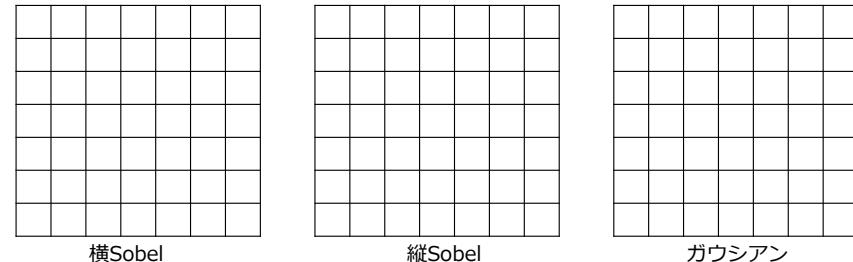
フィルタ処理

右の 7×7 画像に対して…

1. 横方向Sobelフィルタを適用せよ
2. 縦方向Sobelフィルタを適用せよ
3. ガウシアンフィルタを適用せよ

4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3

入力画像



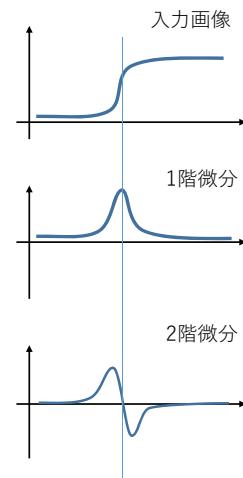
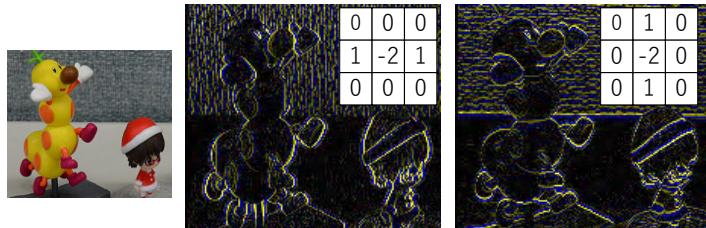
線形フィルタ：2階微分

関数 $f(x, y)$ の2階偏微分は、以下の通り定義される

$$f_{xx} = \frac{\partial^2 f(x, y)}{\partial x^2} = \lim_{h \rightarrow 0} \frac{f(x + h, y) - 2f(x, y) + f(x - h, y)}{h^2}$$

画像 $I(i, j)$ の2階偏微分の近似は…

$$I_{jj} = I(i, j + 1) - 2I(i, j) + I(i, j - 1)$$



線形フィルタ：ラプラシアンフィルタ

関数 $f(x, y)$ のラプラシアン

$$\Delta f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

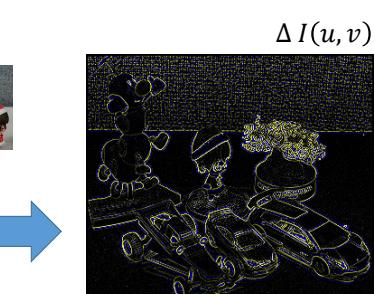
画像 $I(i, j)$ のラプラシアン

$$I(i, j) = I_{ii} + I_{jj}$$

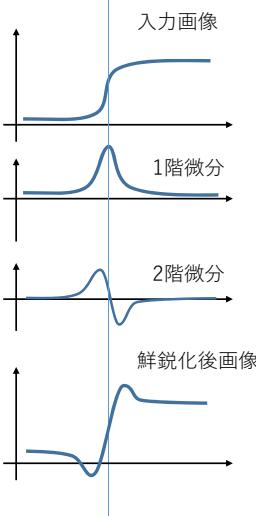
$$\Delta I(i, j) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} \text{Yellow Dog} \\ \text{Santa Claus} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} \text{Yellow Dog} \\ \text{Santa Claus} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} \text{Yellow Dog} \\ \text{Santa Claus} \end{bmatrix}$$

↑
ラプラシアンフィルタ

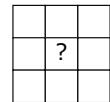


方向に依存しないエッジが一度で得られる
エッジをまたぎ正負の対が現れる
白→黒なら $[0-+0]$ が現れる



線形フィルタ：鮮鋭化フィルタ

2回微分に関するラプラスアンフィルタを改良すると
画像のエッジを強調する鮮鋭化フィルタが設計できる



まとめ: 線形フィルタ

出力画素値を周囲画素の重み付和で計算するフィルタ

$$I'(i,j) = \sum_{m=-h_y}^{h_y} \sum_{n=-h_x}^{h_x} h(m,n) I(i+m, j+n)$$

