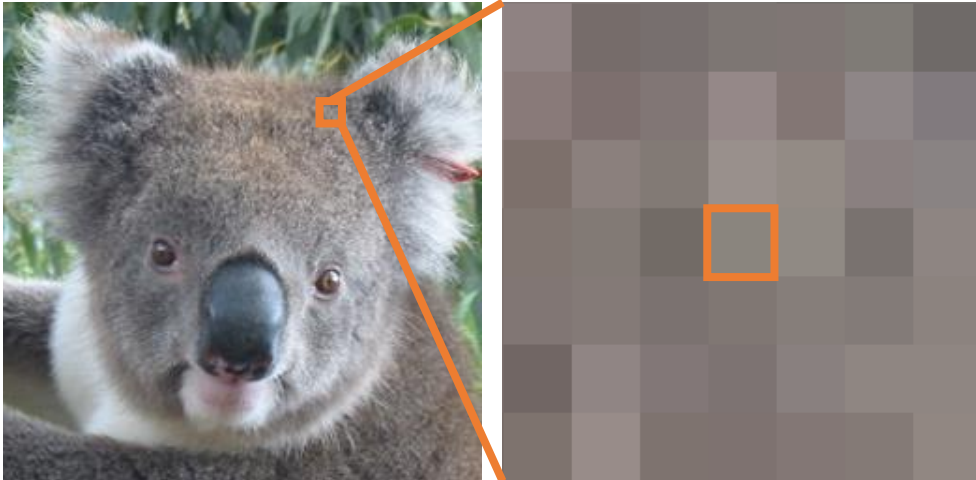


メジアンフィルタと 効率的な計算

メジアンフィルタ

- 平均値の代わりに,
中央値（メジアン）を計算する



出力

中央値

$$y(i, j) = \text{median}(\{x(i + u, j + v)\}_{u,v})$$

中央値（メジアン）

- 値を昇順，または，降順に並べた際，中央に位置する値

- 例

- -9, +4, -4, +1, -3, +2, -3, +2, +4, +9

- -9, -4, -3, -3, +1, +2, +2, +4, +4, +9



$$1.5 = (1 + 2) / 2$$

平均値と中央値の違い

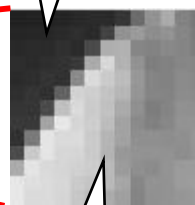
- 外れ値に対して頑健
- 例
 - 外れ値を **含まない** 場合
 - $-9, +4, -4, +1, -3, +2, -3, +2, +4, +9$
 - 平均値 : 0.3 , 中央値 : 1.5
 - 外れ値を **含む** 場合
 - $-900, +4, -4, +1, -3, +2, -3, +2, +4, +9$
 - 平均値 : -88 , 中央値 : 1.5

画像における外れ値

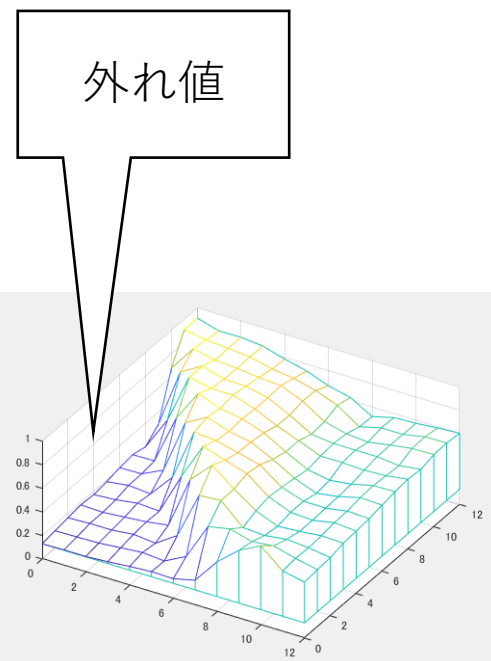
- 被写体と背景などの2つの領域の境界などで、輝度や色が急峻に変わる場合、他方の領域の値は外れ値となる。



背景



被写体



輝度値を垂直軸
に示したもの

プログラム (1 / 3)

```
• I = im2double( imread( '../images/23.png' ) );  
  Iycc = rgb2ycbcr( I );  
  Y = Iycc(:, :, 1);
```

```
iy = 70;    ix = 190;  % 対象画素  
r = 5;      % 窓半径
```

```
Y_local = Y( iy-r:iy+r, ix-r:ix+r );
```

```
% 処理箇所の表示
```

```
figure(1), imshow( Y );
```

```
hold on;
```

```
rectangle( 'Position', [ix-r, iy-r, 2*r+1, 2*r+1], ...  
           'EdgeColor', 'red', 'LineWidth', 3 );
```

```
hold off;
```

```
figure(2), imshow( imresize( Y_local, 4, 'nearest' ) );
```

```
% MATLAB の場合は, nearest の代わりに box
```

画像名と
iy, ix の値以外は
前回の授業の
コードと同じ

プログラム (2 / 3)

- % 中央値を計算

```
Y_sorted = sort( Y_local(:) );
```

```
M = numel( Y_sorted );
```

```
if mod(M,2) == 1 % 奇数なら
```

```
    mu = Y_sorted( (M+1)/2 );
```

```
else
```

```
    mu = 0.5 * sum( Y_sorted( (M/2):(M/2+1) ) );
```

```
end
```

```
% 平均値と中央値の比較
```

```
mean( Y_sorted )
```

```
mu
```

プログラム (3 / 3)

- % 全画素での実行

```
[sy,sx] = size( Y );  
Y_out = zeros( sy,sx );
```

```
for ix = 1+r:sx-r  
    for iy = 1+r:sy-r
```

```
        Y_local = Y( iy-r:iy+r, ix-r:ix+r );  
        mu = median( Y_local(:) );  
        Y_out( iy, ix ) = med;
```

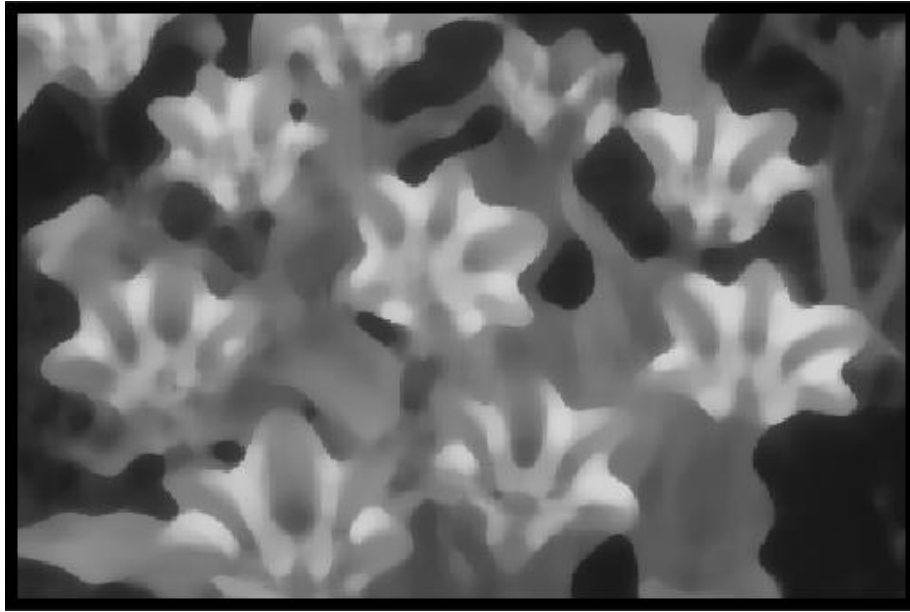
```
    end  
end
```

並べ替え, 及び, 中央値
の抽出は,
median 関数で計算できる

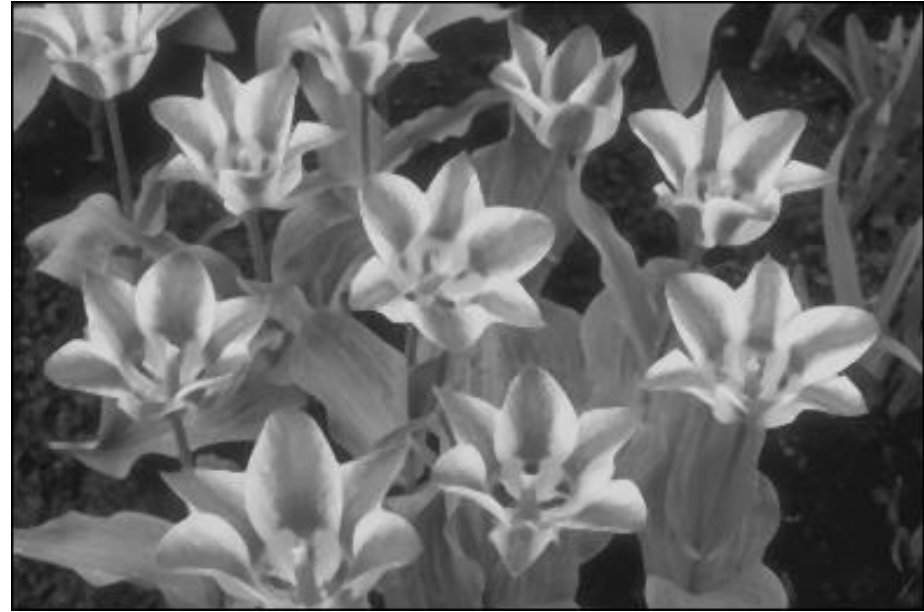
% 出力画像の表示

```
figure(3), imshow( Y_out );
```


実行結果



フィルタ半径
 $r = 5$



フィルタ半径
 $r = 1$

どのような用途に使われるか？

- 平滑化

- エッジを保存しつつ，平滑化する能力を持つ．

- ⇔

- ボックスフィルタやガウシアンフィルタは
エッジ保存能力を持たない．

- ノイズ除去

- ごま塩ノイズに対して，
高いノイズ除去能力を持つ．

ごま塩ノイズの除去

- % ごま塩ノイズの付加
rate = 0.1; % 画像中のどの程度の画素にノイズを加えるか
Y_noise = imnoise(Y, 'salt & pepper', rate);

Y_med = medfilt2(Y_noise, [2*r+1,2*r+1]);

figure(4), imshow([Y_noise, Y_med]);

スイッチングメジアンフィルタ

- % ノイズの生じた画素を検出
% ここでは、簡単に、白飛び、黒飛びした画素値を
% ノイズとして扱う
B_noise = Y_noise < 0.1 | Y_noise > 0.9;
figure(5), imshow(B_noise);

% ノイズの生じた画素のみ,
% メディアンフィルタの結果と置き換える
Y_switch = Y_noise;
Y_switch(B_noise) = Y_med(B_noise);

figure(4), imshow([Y_noise, Y_med, Y_switch]);

スイッチングメディアンフィルタ

補足：画素ごとにノイズ検出とメディアン処理していく場合

- % お手製のメディアンフィルタのコードを基にして、少し修正
Y_out = Y_noise; % 入力画像（劣化画像）を出力画像にコピー

for ix = 1+r:sx-r
 for iy = 1+r:sy-r

 y = Y_noise(iy, ix);
 if y < 0.1 | 0.9 < y % 外れ値なら、メディアン処理

 Y_local = Y_noise(iy-r:iy+r, ix-r:ix+r);
 mu = median(Y_local(:));
 Y_out(iy, ix) = med;
 end

 end

end

% 出力画像の表示
figure(3), imshow(Y_out);

実行結果



ごま塩雑音
1割の画素に
ノイズを付加



メディアンフィルタ
フィルタ半径
 $r = 1$



スイッチング
メディアンフィルタ
フィルタ半径
 $r = 1$

補足 平均値と中央値（1 / 2）

- 平均値と中央値は、それぞれ以下の方程式の答えである。

複数の観測値 y_i からの距離の
二乗値の合計が最小となる x

$$x^* = \arg \min_x \sum_i (x - y_i)^2$$



$$x^* = \frac{\sum_i y_i}{\sum_i 1}$$

複数の観測値 y_i からの距離の
絶対値の合計が最小となる x

$$x^* = \arg \min_x \sum_i |x - y_i|$$

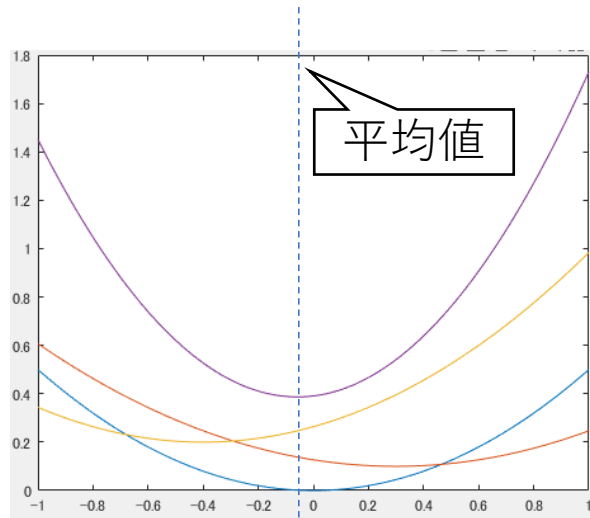


$$x^* = \text{median}(\{y_i\})$$

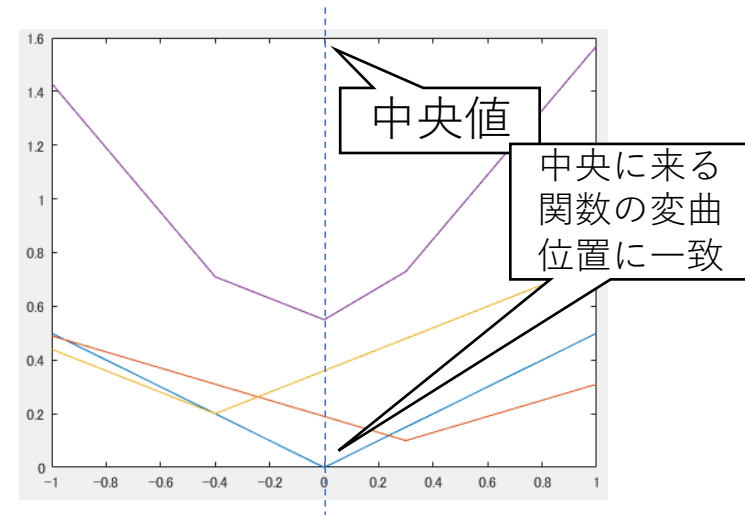
ある領域において、値のわかっている画素値 y_i を基に、その領域の代表値 x を得る際、二乗誤差を最小化する x を与えるのが、ボックスフィルタであり、絶対値誤差を最小化する x を与えるのが、メジアンフィルタである。

補足 平均値と中央値（2 / 2）

- 二次関数
 - 複数個を足し合わせると、同じく、二次関数の形になる。
 - 最小値は微分値が 0 となる位置に存在する。
- 絶対値関数
 - 複数個を足し合わせると、角張った形になる。
 - 最小値を与える位置は、場合によっては複数個並んで存在する。



紫が、赤、青、黄の3つ
の関数の合計



紫が、赤、青、黄の3つ
の関数の合計

中央値が解となることの確認

$$\sum_{i=1}^3 |x - y_i| = |x - y_1| + |x - y_2| + |x - y_3|$$

- `X = 0:0.01:3;`

```
y1 = 0.2;
```

```
y2 = 1.3; ← 中央値
```

```
y3 = 2.3;
```

```
Y_ = abs( X - y1 ) + abs( X - y2 ) + abs( X - y3 );
```


```
figure(5), plot( X, Y_ );
```

補足 ガウシアンフィルタと荷重メジアン

- ガウシアンフィルタを含む
正のフィルタ係数からなる線形フィルタも
方程式として表せる.

複数の観測値 y_i からの距離の
重み付き二乗値の合計が最小となる x

$$x^* = \arg \min_x \sum_i w_i (x - y_i)^2$$




$$w_i \geq 0$$

$$x^* = \frac{\sum_i w_i y_i}{\sum_i w_i}$$

複数の観測値 y_i からの距離の
重み付き絶対値の合計が最小となる x

$$x^* = \arg \min_x \sum_i w_i |x - y_i|$$



y_i の値の昇順に, w_i を並び替えた後,
 w_i の累積和を計算していくときに,
 w_i の合計値の半値を超える i に対応する
 y_i が解となる.

荷重メジアン の解

$$\sum_{i=1}^3 w_i |x - y_i| = w_1 |x - y_1| + w_2 |x - y_2| + w_3 |x - y_3|$$

- `X = 0:0.01:3;`

```
y1 = 0.2;    w1 = 0.6;  
y2 = 1.3;    w2 = 0.3;  
y3 = 2.3;    w3 = 0.1;
```



重みの累積和が合計の半値を超える

```
Y_ = w1 * abs(X-y1) + w2 * abs(X-y2) + w3 * abs(X-y3);
```

```
figure(6), plot( X, Y_ );
```

補足 フィルタを方程式として表す利点

- 2つのフィルタの中間の性質をもつ計算を表現できる.
 - ただし、特殊な解法が使えず、計算に時間がかかるようになる.
- 他の制約条件を加えることが可能.

$$\arg \min_x \sum_i (x - y_i)^2 \quad \arg \min_x \sum_i |x - y_i| \quad 0.3 \leq x \leq 0.7$$



$$\arg \min_{x \in [0.3, 0.7]} \sum_i (x - y_i)^2 + \lambda |x - y_i|$$



凸最適化問題や線形計画法の求解アルゴリズムを用いて解く

高速化

メジアン計算に適したソート法

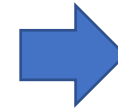
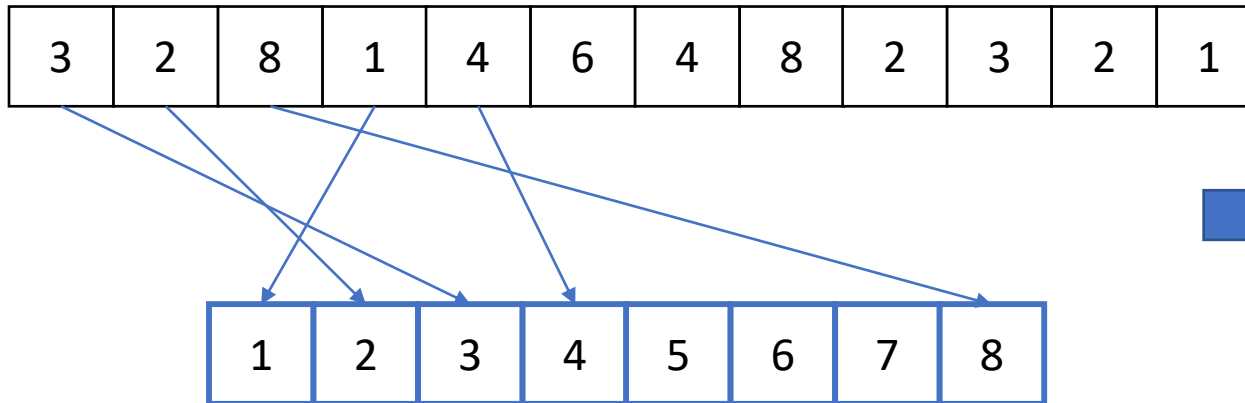
- 高速ソートといえは. . .
 - クイックソート 計算量 $O(n \log(n))$
 - マージソート 計算量 $O(n \log(n))$ 高安定性
 - バイトニックソート 計算量 並列時に $O(\log(n)^2)$
 - バケットソート 計算量 特殊な条件下で $O(n)$
- 画像処理でのメジアンフィルタリングには
バケットソートの考えを利用できる

バケットソート (1 / 2)

- 並び替える値が整数で値域が狭いときに利用可能
- 手順
 - 準備：値域の個数のバケット（ビン）を用意する

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

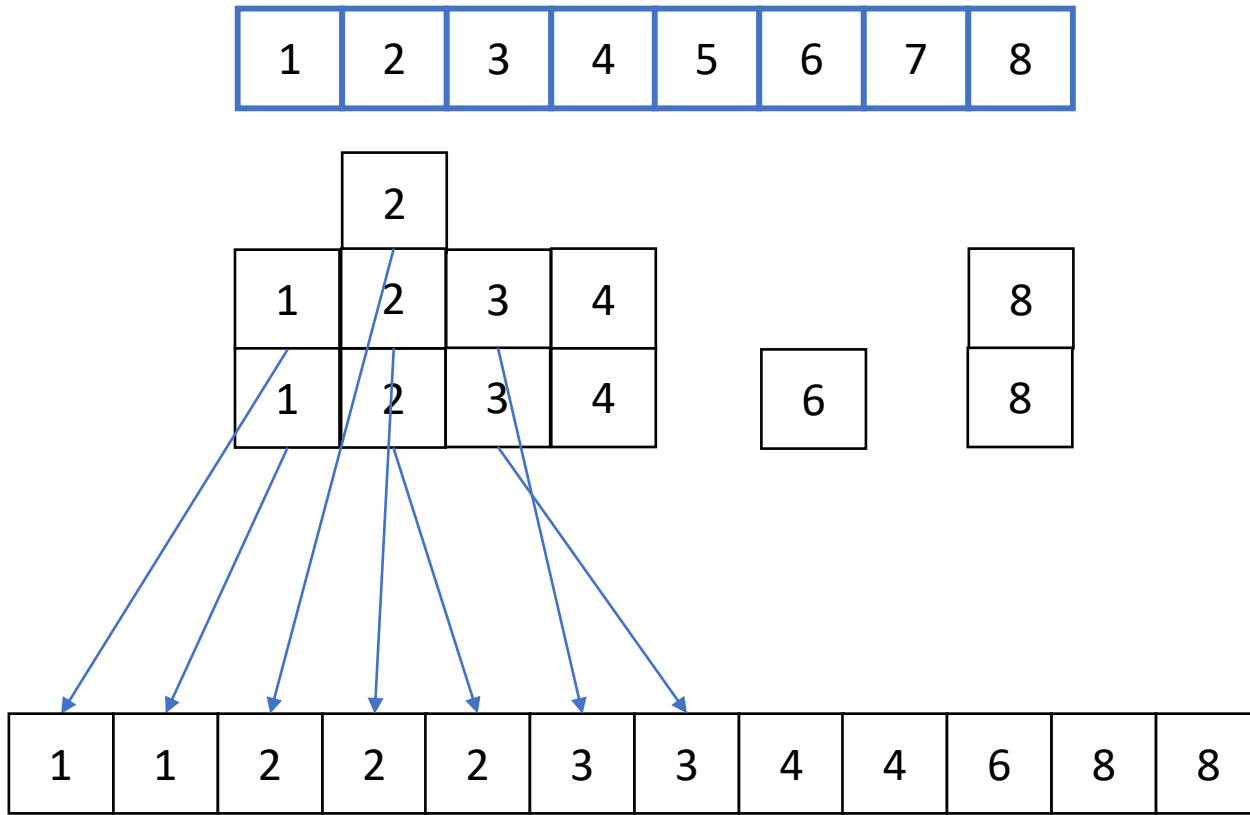
- データを対応するバケットに入れる。



	2						
1	2	3	4				8
1	2	3	4		6		8
1	2	3	4	5	6	7	8

バケットソート (2 / 2)

- バケットの番号が小さい方から順に、データを取り出す。



中央値の求め方 (1 / 2)

- バケットに含まれるデータの個数（頻度）の累積和を計算
- データの合計数の半値を超える箇所が中央値

バケット番号	1	2	3	4	5	6	7	8
--------	---	---	---	---	---	---	---	---

		2						
1	2	3	4					8
1	2	3	4		6			8



各バケットに
格納されたデータ数
つまり、頻度

2 3 2 2 0 1 0 2

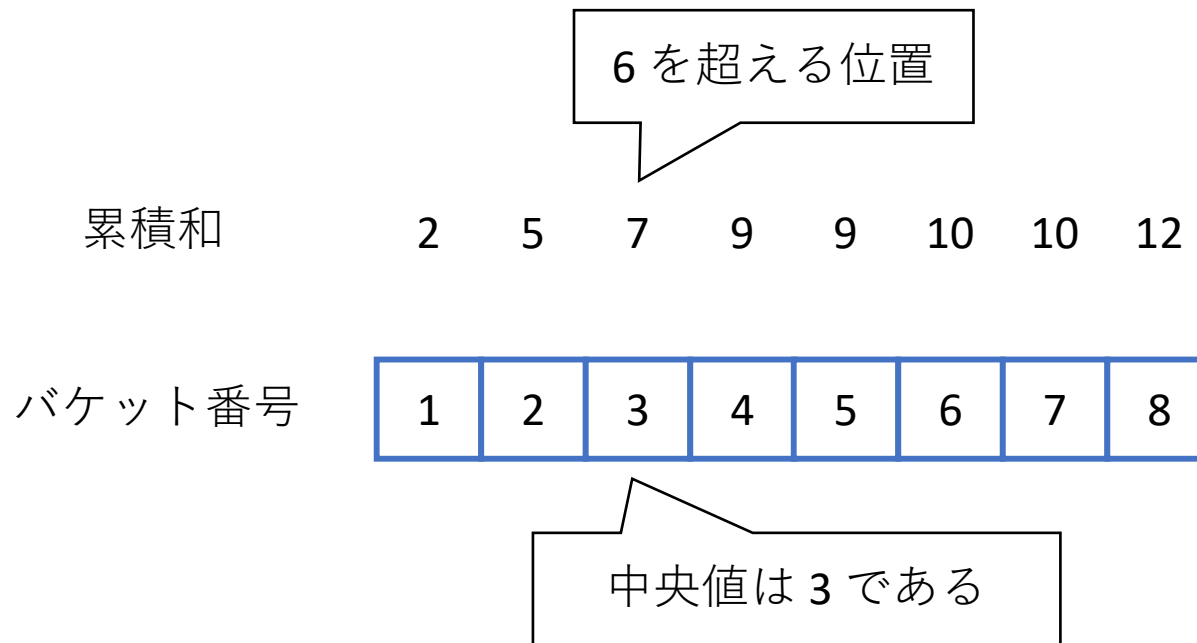


累積和を計算

2 5 7 9 9 10 10 12

中央値の求め方 (2 / 2)

- データ数が12個の場合、半値は6



試しに、並び直した値を確認すると、確かに中央値は3 $(=(3+3)/2)$ である

1	1	2	2	2	3	3	4	4	6	8	8
---	---	---	---	---	---	---	---	---	---	---	---

プログラム (1 / 2)

- `nTone = 256; % 階調数`
`Tone = round(Y * (nTone-1)) + 1;`
`% 画素値を整数値 [1,nTone] に変換`

`Bucket_idx = 1:nTone; % バケット番号`

`iy = 70; ix = 190; % 対象画素`
`r = 5; % 窓半径`

`T_local = Tone(iy-r:iy+r, ix-r:ix+r);`

`% 各バケットに割り当てられる画素数を計算`
`Bucket = hist(T_local(:), Bucket_idx);`
`Bucket_cum = cumsum(Bucket); % 累積和`

`figure(7), bar(Bucket);`
`figure(8), plot(Bucket_cum);`

プログラム (2 / 2)

- % 半値を超える位置を調べる

```
total = numel( T_local );  
half = total / 2;
```

```
if mod(total,2) == 1 % 奇数なら  
    idx = find( Bucket_cum > half, 1 );  
    mu = idx;
```

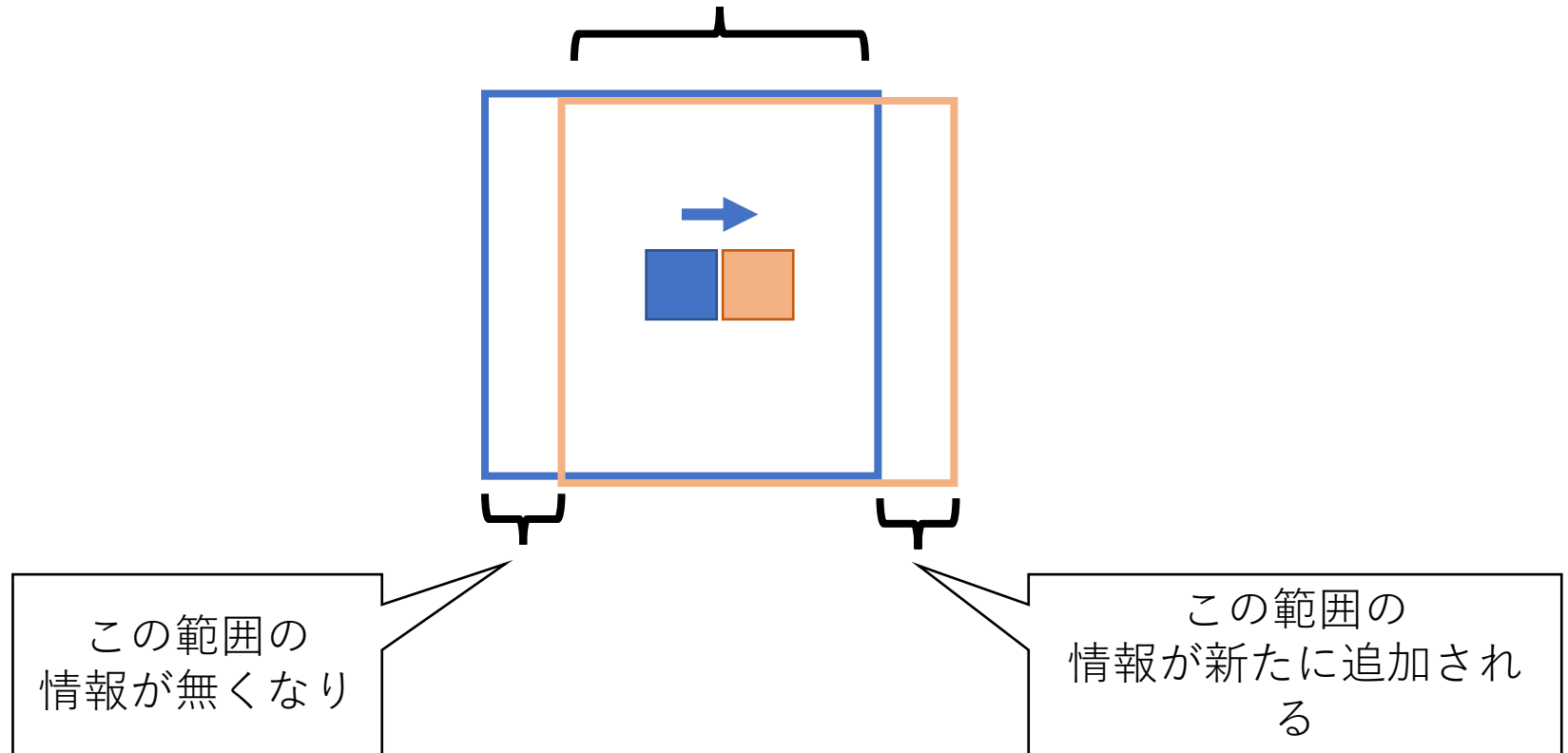
```
else % 偶数の場合, 直前の累積和がちょうど半値である可能性がある  
    idx = find( Bucket_cum > half, 1 );  
    mu = idx;  
    if ( Bucket_cum( idx-1 ) == half )  
        mu = 0.5 * ( mu + (idx-1) );  
    end  
end
```

```
% 通常のソートを用いた方法との比較  
median( T_local(:) )  
mu
```

一つ隣の画素での計算 (1 / 2)

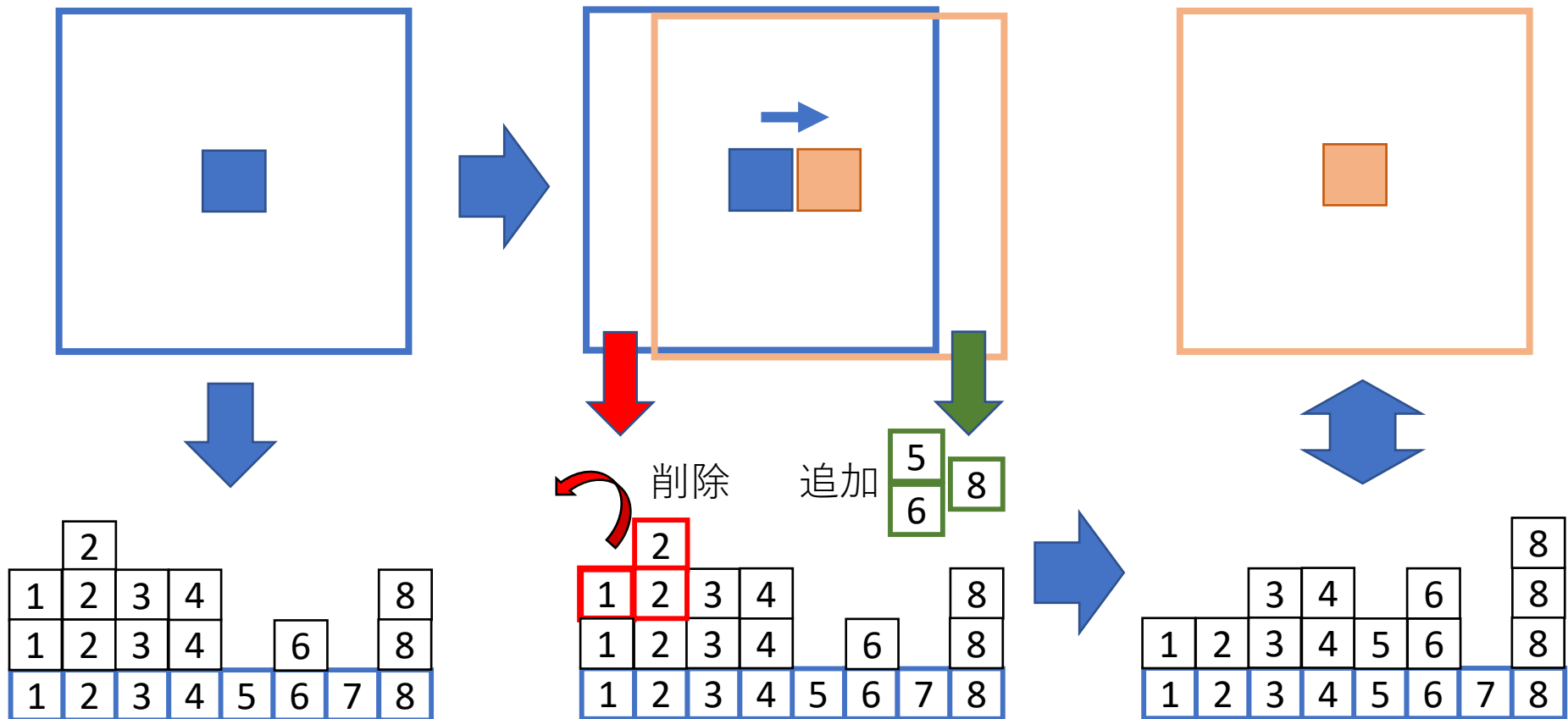
- ある画素の周辺の画素情報と
一つ隣の画素の周辺情報は、ほとんど変わらない。

オーバーラップ



一つ隣の画素での計算 (2 / 2)

- 無くなる画素情報と追加される画素情報のみをバケットに加える



プログラム (1 / 2)

- `ix = 191;` % 隣の画素

```
T_rem = Tone( iy-r:iy+r, ix-r-1 ); % 削除分  
T_add = Tone( iy-r:iy+r, ix+r ); % 追加分
```

```
for i = 1:numel( T_rem )  
    Bucket( T_rem(i) ) = Bucket( T_rem(i) ) - 1;  
end  
for i = 1:numel( T_add )  
    Bucket( T_add(i) ) = Bucket( T_add(i) ) + 1;  
end
```

```
% 以降同様
```

```
Bucket_cum = cumsum( Bucket );
```

プログラム (2 / 2)

- `if mod(total,2) == 1 % 奇数なら`
 `idx = find(Bucket_cum > half, 1);`
 `mu = idx;`

 `else % 偶数の場合, 直前の累積和がちょうど半値である可能性もある`
 `idx = find(Bucket_cum > half, 1);`
 `mu = idx;`
 `if (Bucket_cum(idx-1) == half)`
 `mu = 0.5 * (mu + (idx-1));`
 `end`
 `end`

 `% 通常のソートを用いた方法との比較`
 `T_local = Tone(iy-r:iy+r, ix-r:ix+r);`
 `median(T_local(:))`
 `mu`

実行時間 と 参考文献

- 定数時間メジアンフィルタ

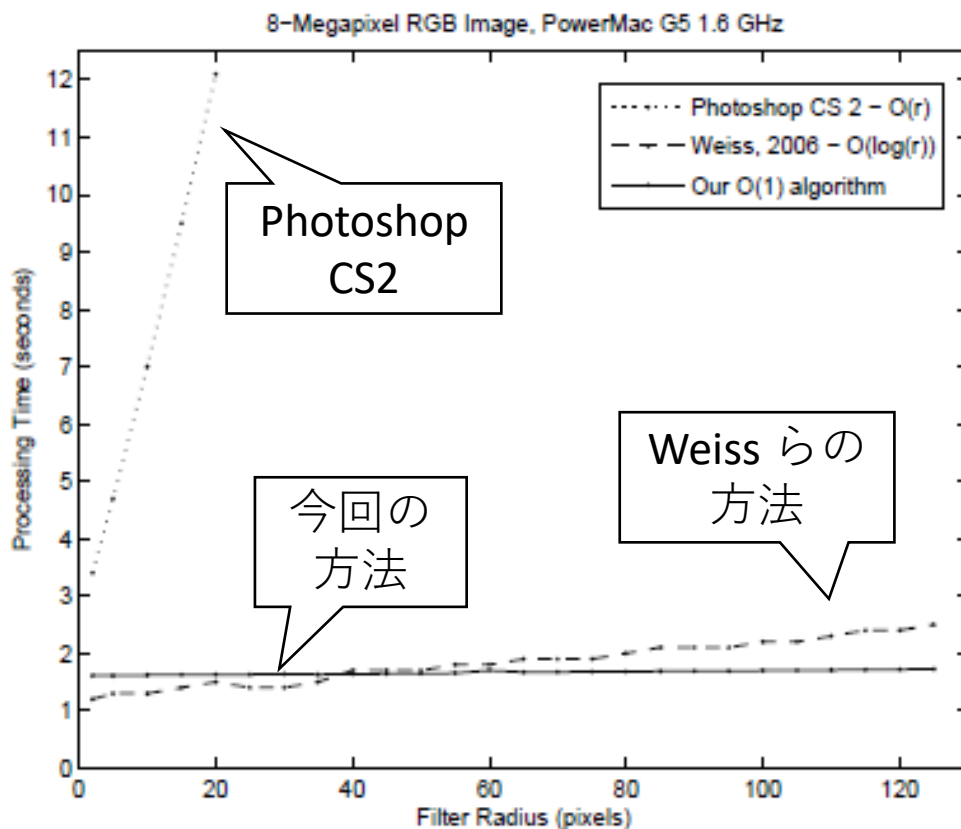
- S. Perreault and P. Hebert, “Median filtering in constant time,”
IEEE Trans. on Image Process. (TIP), vol. 16, no. 9, pp. 2389-2394, 2007.

800万画素のRGB画像
の処理時間
(スマホ写真サイズ)

横軸：フィルタ半径

縦軸：処理時間（秒）

図は Perreault らの論文から引用



Weiss らの方法の詳細

<https://www.youtube.com/watch?reload=9&v=RpuYTPpUgEA>