バイラテラルフィルタと
効率的な計算

バイラテラルフィルタ

- エッジ保存能力をもつ平滑化
  - 空間フィルタに（ボックスフィルタやガウシアンフィルタ）
    レンジフィルタが加わる.
- レンジフィルタ
  - 領域中央の画素と比べて
    輝度差の大きなハズレ値を除外する役割をもつ

$$x_i = \frac{\sum_{j \in N(i)} k_s(i,j)\ y_j}{\sum_{j \in N(i)} k_s(i,j)}$$

j は i 近傍の画素

$$\Rightarrow$$

$$x_i = \frac{\sum_{j \in N(i)} k_s(i,j)\ \underline{k_r(y_i, y_j)}\ y_j}{\sum_{j \in N(i)} k_s(i,j)\ \underline{k_r(y_i, y_j)}}$$

レンジフィルタ

レンジフィルタ（１／２）

- 外れ値を除外するフィルタ（関数）
- 例
    - −7，+2，−1，+1，0，+2，−9，+3，+4
        - 青は基準値，赤は外れ値

- ５離れていたら，削除

+0，+2，−1，+1，0，+2，+0，+3，+4

レンジフィルタ（２／２）

- 「５離れていたら，削除」を関数として表す
  - 基準値を y_i と定義する
  - 周辺値を y_j と定義する

（今回の例では $y_i = 0$）

$$y_j$$

-7, +2, -1, +1, +0, +2, -9, +3, +4

  - レンジ関数

$$k_r(y_i, y_j) := \begin{cases} 1 & |y_j - y_i| < 5 \\ 0 & \text{otherwise} \end{cases}$$

$$k_r(y_i, y_j)$$

+0, +1, +1, +1, +1, +1, +0, +1, +0

  - 乗算結果

$$\widehat{y}_j = k_r(y_i, y_j)\ y_j$$

$$\widehat{y}_j$$

+0, +2, -1, +1, +0, +2, +0, +3, +4

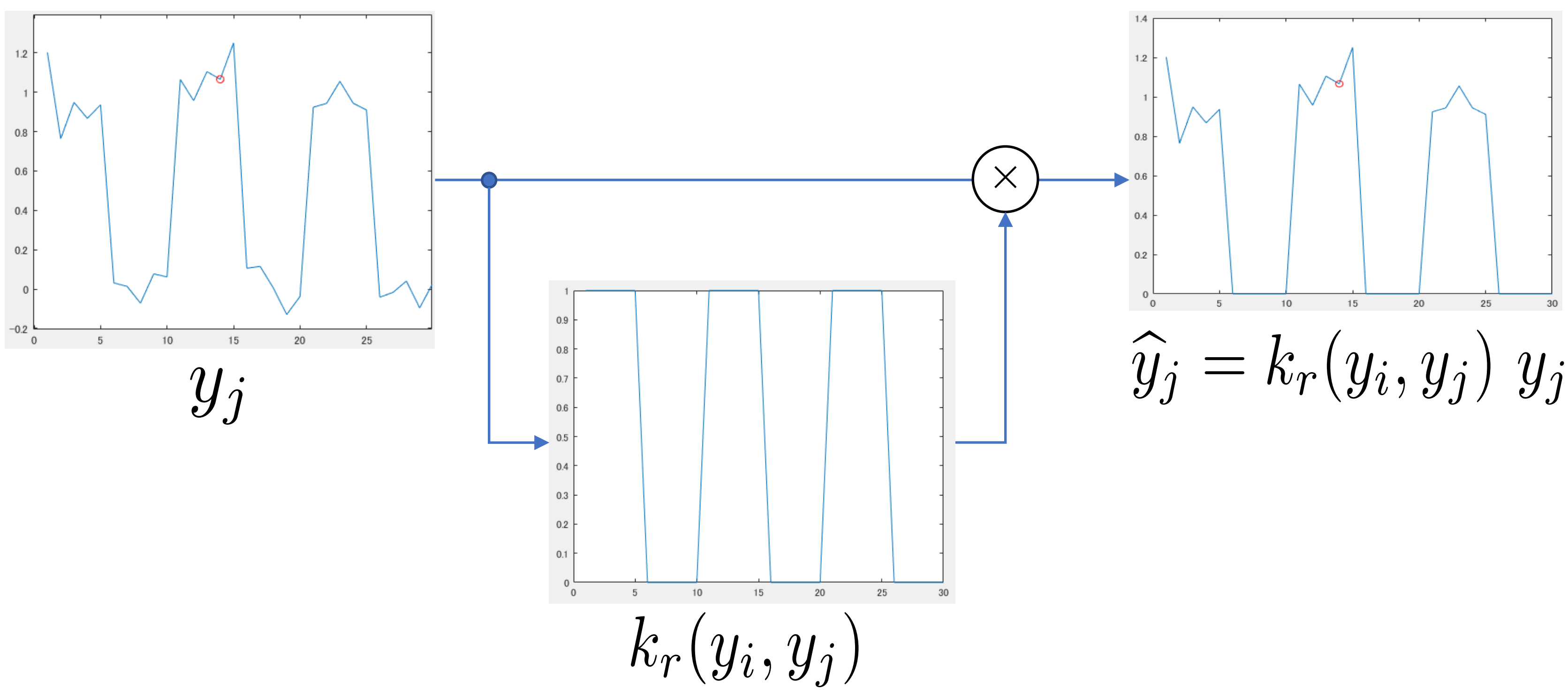
プログラム（1／2）

```
% 入力データの生成（矩形波）
Y = repmat( [ones(1,5), zeros(1,5)], [1,3] );
Y = Y + 0.1 * randn( size(Y) ); % 変動を付加
figure(1), plot( Y );

% レンジ関数の定義
k_r = @(X,xi,tau) abs(X-xi) < tau;


% 基準値を決定
i = 14;  yi = Y(i);
figure(1), hold on; plot(i,yi,'ro'); hold off;

% レンジ関数で処理
Kr = k_r( Y, yi, 0.5 );
figure(2), plot( Kr );
```

プログラム（２／２）

- % レンジフィルタの結果（重み）の掛け合わせ

```
KrY = Kr .* Y;

figure(1), hold on; plot( KrY ); hold off;
```

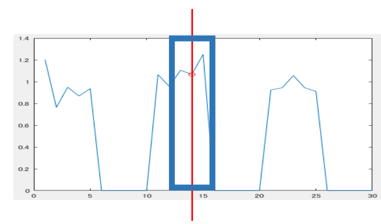


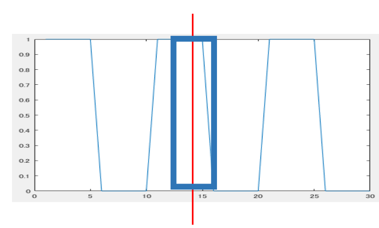
空間フィルタの追加

- ボックスフィルタ（移動平均フィルタ）を追加
  - 画素 i 近傍のみの平均値を求める
  - 例：
    半径 3 画素

$$k_s(i,j) := \begin{cases} 1 & |i-j| < 3 \\ 0 & \text{otherwise} \end{cases}$$

$$x_i = \frac{\displaystyle\sum_{j \in N(i)} \underline{k_s(i,j)}\ k_r(y_i, y_j)\ y_j}{\displaystyle\sum_{j \in N(i)} \underline{k_s(i,j)}\ k_r(y_i, y_j)}$$





青色枠内の
平均値を計算

プログラム（1／2）

- ```
  % 空間フィルタの用意
  r = 2;
  Ks = ones(1,2*r+1);
  Ks = Ks/sum(Ks(:));

  % バイラテラルフィルタリング
  KsKrY = Ks .* Kr(i-r:i+r) .* Y(i-r:i+r);
  KsKr  = Ks .* Kr(i-r:i+r);

  mu = sum( KsKrY ) / sum( KsKr );

  % もしくは，先に i 周辺を切り出しておき
  % Yj = Y(i-r:i+r);
  % Kr = k_r( Yj, yi, 0.5 );
  % KsKrY = Ks .* Kr .* Yj;
  ```
  ```

プログラム（２／２）

- ```
  % 全ての画素に対して処理
  X = zeros( size( Y ) ); % 出力画像

  for i = 1+r:numel(Y)-r

    yi = Y(i);
    Yj = Y( i-r:i+r );      % 周辺画素を切り出し

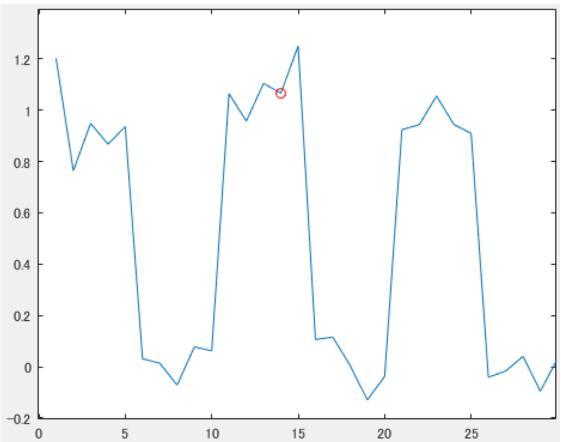
    Kr = k_r( Yj, yi, 0.5 ); % レンジ重みを計算
    % Kr(:) = 1; % レンジ重みを使わない場合

    KsKr  = Ks .* Kr;
    KsKrY = KsKr .* Yj;

    X(i) = sum( KsKrY ) / sum( KsKr );
  end

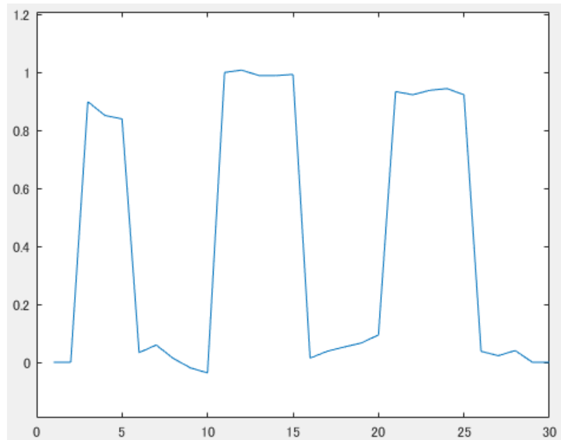
  figure(3), plot( X );
  ```
  ```

結果（1D バイラテラルフィルタ）



$$y_j$$

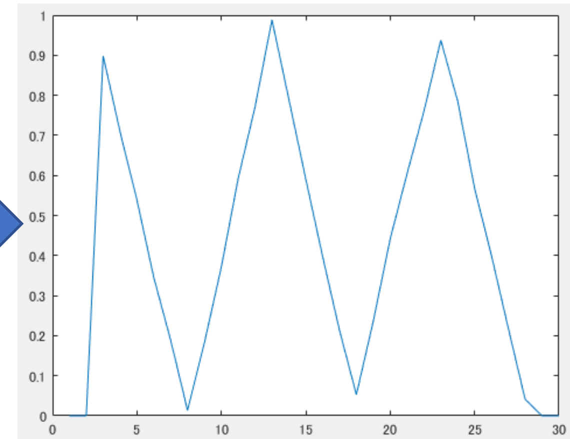
入力データ



$$x_i = \frac{\displaystyle\sum_{j \in N(i)} k_s(i,j)\ k_r(y_i, y_j)\ y_j}{\displaystyle\sum_{j \in N(i)} k_s(i,j)\ k_r(y_i, y_j)}$$

バイラテラルフィルタ

空間フィルタ＋レンジフィルタ



$$x_i = \frac{\displaystyle\sum_{j \in N(i)} k_s(i,j)\ y_j}{\displaystyle\sum_{j \in N(i)} k_s(i,j)}$$

ボックスフィルタ
（移動平均フィルタ）

空間フィルタのみ

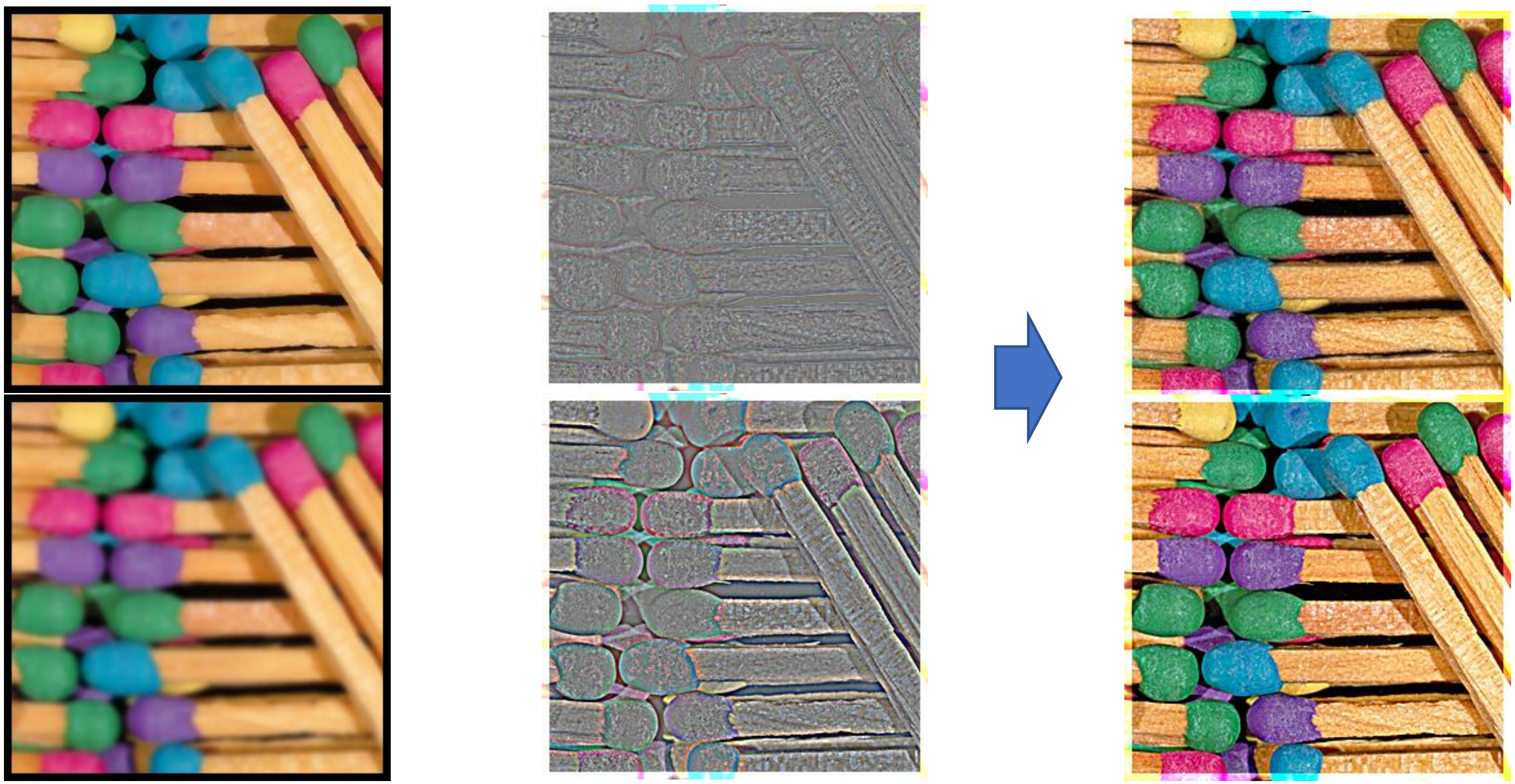
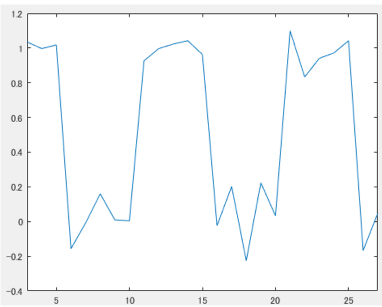
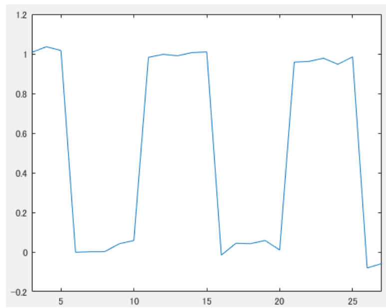
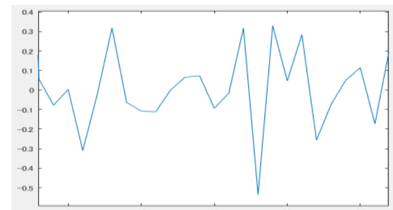
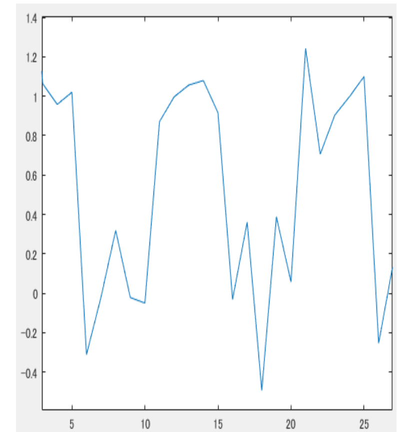
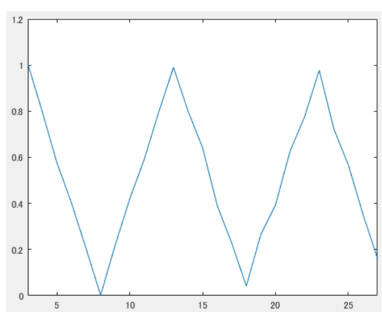
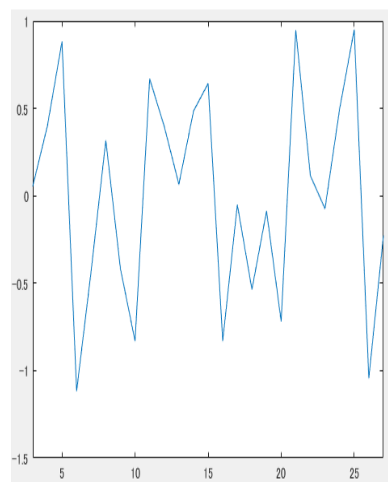
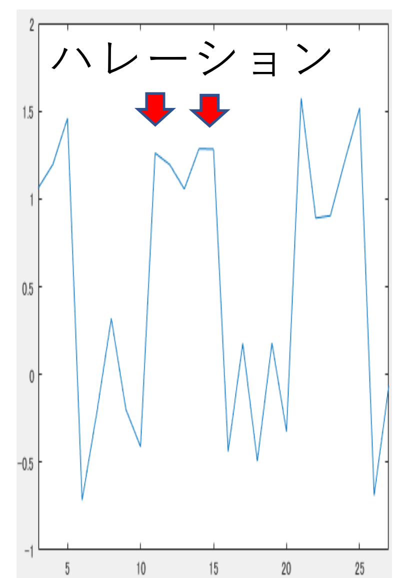
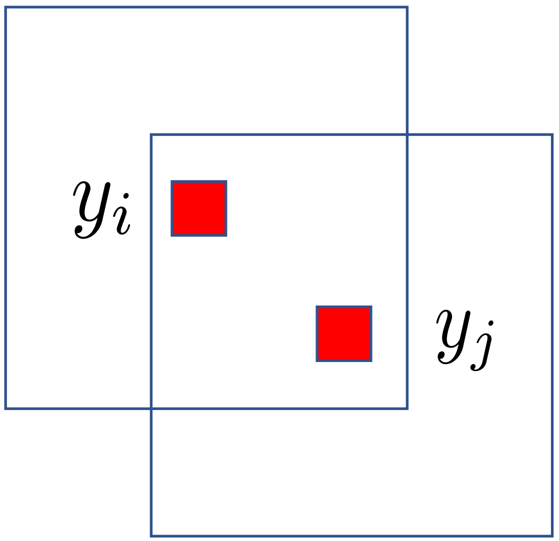
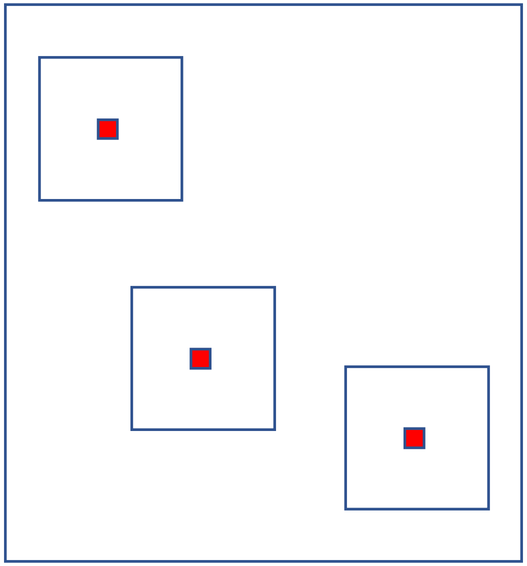
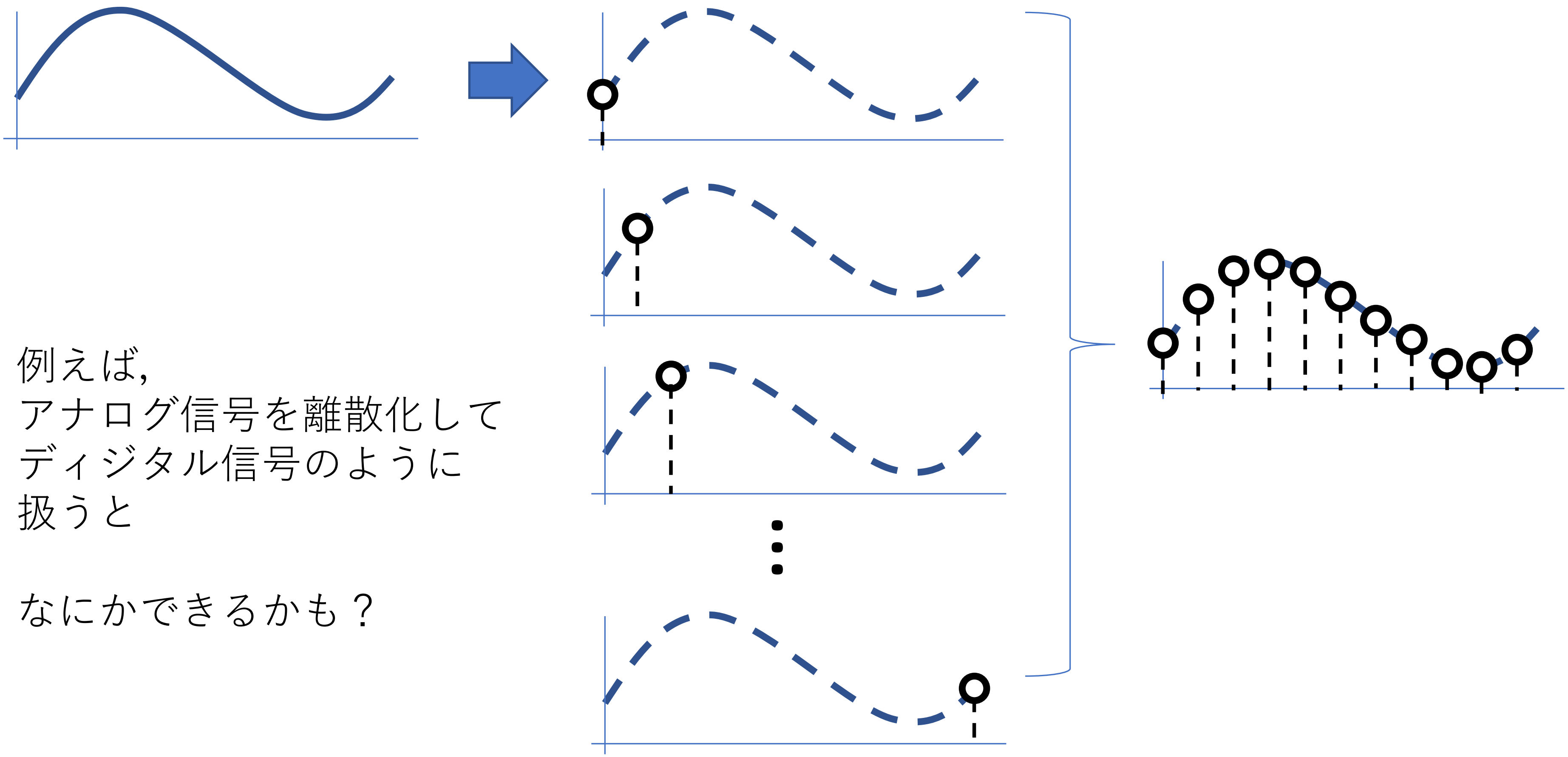
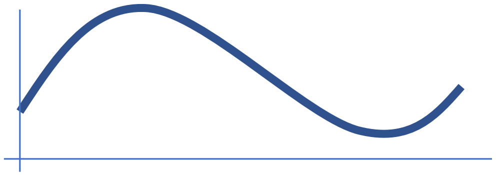
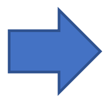
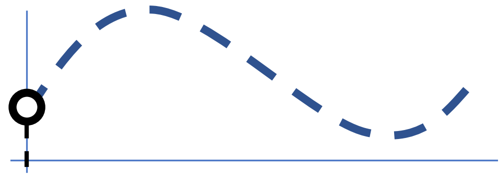
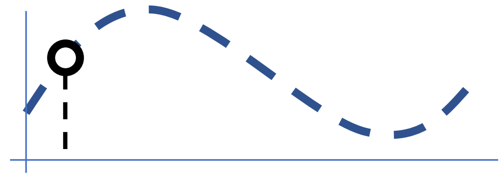
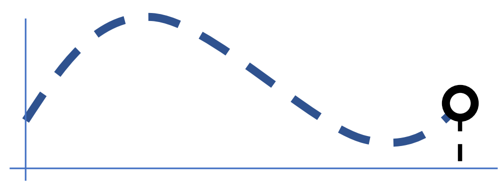
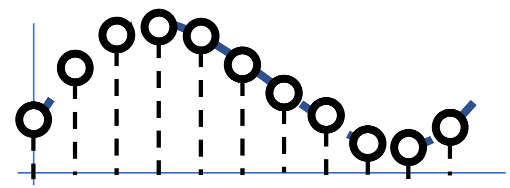
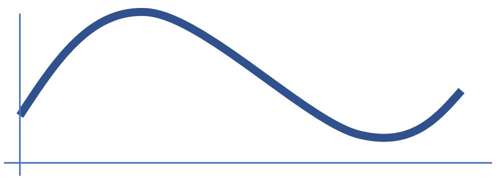
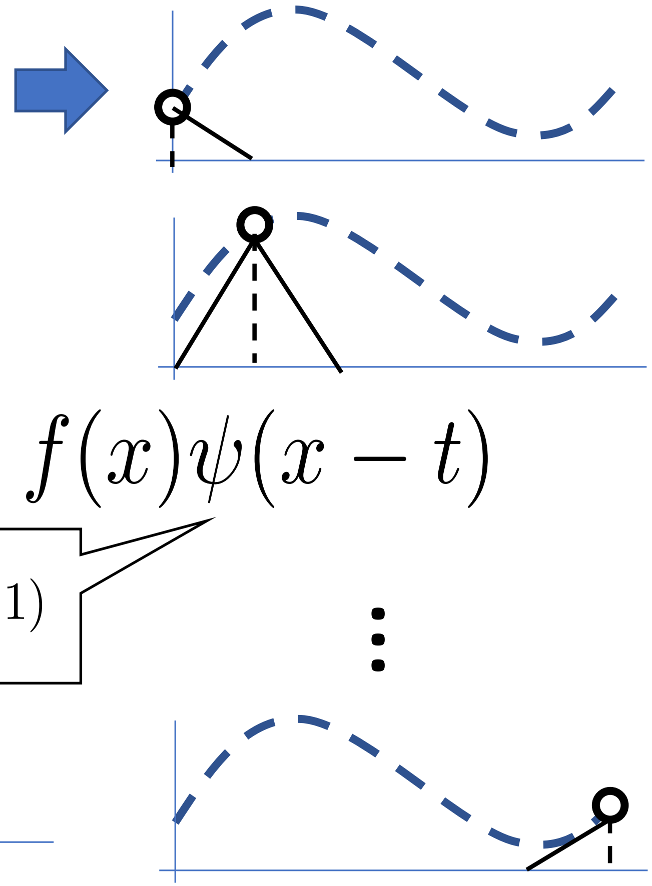
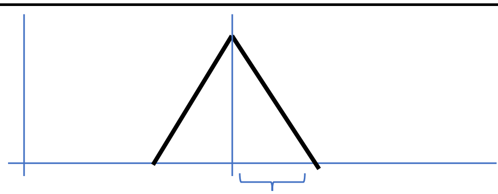
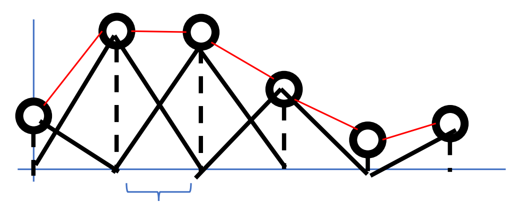
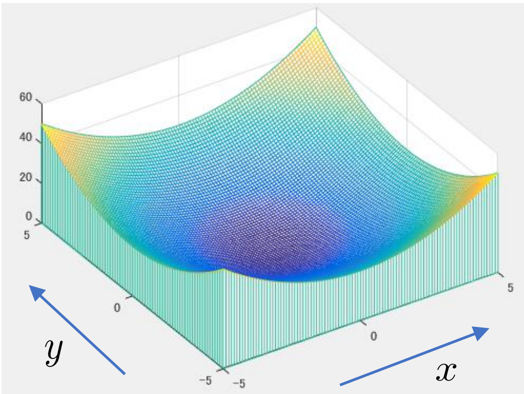
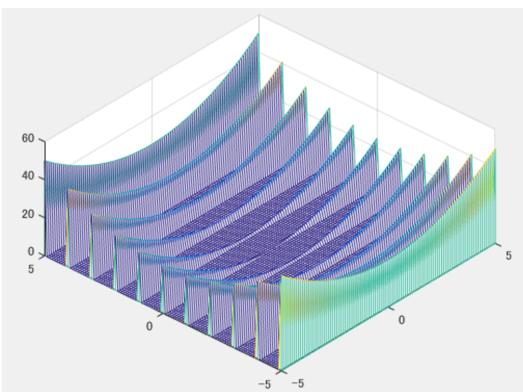
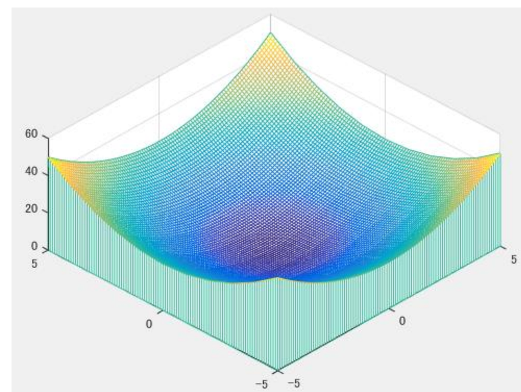
画像バージョン（1／2）

- ```
  I = im2double( imread('../images/match.png') );
  [sy,sx,sc] = size( I );

  % 空間フィルタ（ガウシアンフィルタを使用）
  ss = 2; % 空間フィルタの標準偏差 (standard deviation for spatial)
  r = round(3*ss);
  Ks = fspecial('gaussian', 2*[r,r]+1, ss);

  % レンジフィルタ　関数の定義（ガウス関数を使用）
  sr = 0.1; % レンジフィルタの標準偏差 (standard deviation for range)
  k_r = @(X,xi,sr) exp( (-0.5/sr^2)*(X-xi).^2 );
  ```
  ```

画像バージョン（２／２）

```
J = zeros( size(I) );
for ix = 1+r:sx-r
  for iy = 1+r:sy-r
    for ic = 1:sc

      ii = I(iy,ix,ic);
      Ij = I( iy-r:iy+r, ix-r:ix+r, ic );

      Kr = k_r(Ij, ii, sr);
      % Kr(:)=1; % でオフ

      KsKr  = Ks .* Kr;
      KsKrI = KsKr .* Ij;

      J(iy,ix,ic) = sum(KsKrI(:))/sum(KsKr(:));
    end
  end
end
figure(4), imshow( [I, J] );
```

補足

- MATLAB には imbilatfilt 関数として用意されている．
  - J = imbilatfilt(I, degreeOfSmoothing, spatialSigma);

    spatialSigma が空間フィルタのサイズ
    degreeOfsmoothing がレンジフィルタの平滑化度合い（分散）
  - 前ページで組んだパラメータとの関係

    spatialSigma = ss;
    degreeOfSmoothing = sr^2;

結果（2D バイラテラルフィルタ）

入力データ

バイラテラルフィルタ

空間フィルタ＋レンジフィルタ

ガウシアンフィルタ

空間フィルタのみ

鮮鋭化に用いた場合の比較（１／３）

- アンシャープマスキング
  - 平滑化画像 ＋ （入力画像 − 平滑化画像）の強調

バイラテラル
フィルタ

ガウシアン
フィルタ



平滑化結果

模様＝入力－平滑化
３倍に強調したもの

鮮鋭化

鮮鋭化に用いた場合の比較（２／３）

- ハレーションの発生
  - 輪郭周りが本来よりも明るく／暗くなる現象

バイラテラルフィルタ
を用いたもの

ガウシアンフィルタ
を用いたもの

鮮鋭化に用いた場合の比較（３／３）

- １Ｄの場合


入力信号


バイラテラル
フィルタ


変動
入力－平滑化


鮮鋭化後


ガウシアン
フィルタ


変動
入力－平滑化



鮮鋭化後

効率的な計算

レンジ関数の分解と近似

参考文献
F. Durand and J. Dorsey, “Fast bilateral filtering for the display of high-dynamic-range images,”
ACM Trans. on Graphics, vol. 21, no. 3, pp. 257-266, 2002.

ボトルネック

- レンジフィルタがボトルネック
  - 各画素ごとに計算する必要がある.

$$x_i = \frac{\sum_{j \in N(i)} k_s(i,j)\ \underline{k_r(y_i, y_j)}\ y_j}{\sum_{j \in N(i)} k_s(i,j)\ \underline{k_r(y_i, y_j)}}$$



y_i の周辺画素値 y_j との出力結果と
y_j の周辺画素値 y_i との出力結果は
同一

同一画素値をまとめて計算したい（1／5）

- 画像中には，いくつか同じ値を持つ画素が存在する
  - まとめて計算できないか？
  - ある画素値 t をもつ画素のみに反応するレンジフィルタ

$$k_r(t, y_j)\delta(y_i - t) = \begin{cases} k_r(y_i, y_j) & t = y_i \\ 0 & t \neq y_i \end{cases}$$

t について足し
合わせると，
もとに戻る

$$\sum_t k_r(t, y_j)\delta(y_i - t) = k_r(y_i, y_j)$$



また，y_i と組み合わせ
ると，$t = y_i$ となる
画素値のみを抽出する
機能をもつ
（後ほど使用）

$$\delta(y_i - t)y_i = \begin{cases} y_i & t = y_i \\ 0 & t \neq y_i \end{cases}$$

同一画素値をまとめて計算したい（２／５）

- フィルタ関数に対称性がある場合,
  次のような交換が可能
  - 入力変数を入れ替えても，値が変わらない.

$$k_r(y_i, y_j) = k_r(y_j, y_i)$$

$$k_r(y_i, y_j) = \sum_t k_r(t, y_j)\delta(y_i - t)$$

$$\Updownarrow$$

$$k_r(y_j, y_i) = \sum_t k_r(t, y_i)\delta(y_j - t) \Leftrightarrow \sum_t k_r(y_i, t)\delta(y_j - t)$$

中心画素と周辺画素の
関係を交換できる

同一画素値をまとめて計算したい（３／５）

- フィルタの式への代入　と　順序の置換

$$\sum_{j \in N(i)} k_s(i,j)\, \underline{k_r(y_i, y_j)}\, y_j$$

$$\sum_{j \in N(i)} k_s(i,j) \underline{\sum_t k_r(y_i, t) \delta(y_j - t)}\, y_j$$

y_i と y_j の関数を分けたことで
組み換えが可能となった.

$$\sum_t k_r(y_i, t) \sum_{j \in N(i)} k_s(i,j) \delta(y_j - t)\, y_j$$

補足：　簡単な式での確認

- {a1, a2}, {b1, b2, b3} について以下の計算を行う
  - (a1 + a2)(b1 + b2 + b3)
  - ＝ a1 (b1 + b2 + b3) + a2(b1 + b2 + b3)
  - ＝ (a1 + a2) b1 + (a1 + a2) b2 + (a1 + a2) b3

- どちらも結果は一緒である
  - a1 b1 + a1 b2 + a1 b3 ＋ a2 b1 + a2 b2 + a2 b3
  - a1 b1 + a2 b1 ＋ a1 b2 + a2 b2 ＋ a1 b3 + a2 b3

- 数式で表すと
  - $$\sum_i \left\{ a_i \left(\sum_j bj \right) \right\} = \sum_i a_i \sum_j b_j$$
  - $$\sum_j \left\{ \left(\sum_i a_i \right) bj \right\} = \sum_j \sum_i a_i b_j$$

同一画素値をまとめて計算したい（4／5）

- フィルタの正規化　と　事前計算可能箇所

$$\sum_t k_r(y_i,t) \sum_{j\in N(i)} k_s(i,j)\delta(y_j-t)\; y_j$$

$$\widetilde{k}_s(i,j) = \frac{1}{\sum_j k_s(i,j)} k_s(i,j)$$

なお，ローパスフィルタは通常このように正規化されている

$$\widetilde{y}_j^t = \delta(y_j - t)\; y_j$$

$t = y_j$ の画素値のみを抽出する
フィルタリング前に
画像全体で計算ができる

$$\sum_t k_r(y_i,t) \sum_{j\in N(i)} \widetilde{k}_s(i,j)\; \widetilde{y}_j^t$$

レンジフィルタ

ガウシアン
フィルタ

関数近似を用い，総和関数の位置を交換することで，レンジフィルタを外にくくり出せた

同一画素値をまとめて計算したい（５／５）

- 分子も同様に計算
- 分子と分母を独立に計算した後,
  各画素ごとに割り算を行う.

$$\frac{\sum_{j \in N(i)} k_s(i,j)\ k_r(y_i, y_j)\ y_j}{\sum_{j \in N(i)} k_s(i,j)\ k_r(y_i, y_j)}$$

$\Rightarrow \sum_{t} k_r(y_i, t) \sum_{j \in N(i)} \widetilde{k}_s(i,j)(\delta(y_j - t)\ y_j)$

$\Rightarrow \sum_{t} k_r(y_i, t) \sum_{j \in N(i)} \widetilde{k}_s(i,j)(\delta(y_j - t)\ 1)$

次ページから，δ 関数 以外の関数を使用できるように一般化した後,
プログラムとして実装

一般化に向けて　関数の近似表現（１／５）

- 区間的補間関数の重畳としての表現



例えば,
アナログ信号を離散化して
ディジタル信号のように
扱うと

なにかできるかも？

一般化に向けて　関数の近似表現（２／５）

- 数式表現　（デルタ関数を用いて表せる）



$f(x)$







t

$$f(x)\delta(x-t)$$

⋮



$$\sum_t f(x)\delta(x-t)$$



$$\begin{aligned}&\sum_t f(x)\delta(x-t)\\&= f(x)\sum_t \delta(x-t)\\&= f(x)\times 1\end{aligned}$$

一般化に向けて　関数の近似表現（３／５）

- 重畳したときに,
  一定値となる区分関数であれば

$$\sum_t \psi(x-t) = 1$$

デルタ関数
三角形関数
sinc 関数
など

- 区分的な近似が可能

$$f(x) = \sum_t f(x)\psi(x-t)$$

$$= f(x)\sum_t \psi(x-t) = f(x) \times 1$$

一般化に向けて　関数の近似表現（4／5）

- 三角形関数での表現



$f(x)$



$f(x)\psi(x-t)$

$$\max(0, -\frac{1}{\Delta t}|x-t|+1)$$



t Δt

$$\sum_t f(x)\psi(x-t)$$

合成関数は，点同士を直線で結んだ 赤色 の形状をもつ



Δt

$x \in [0,1]$ として，
T 個の区間に分ける場合は
$\Delta t = 1/T$

一般化に向けて　関数の近似表現（５／５）



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

補間関数を用いて
線形和で近似できる

$f(x,y) = \sum_t f(x,t) \cdot g(t,y)$



$t = -5, -4, \ldots, 4, 5$



$t = -5, -4, \ldots, 4, 5$

$\Delta t = 1$

$e.g.\ \sum_t (x^2 + t^2) \cdot \delta(t,y) \qquad \delta(t,y) = \begin{cases} 1 & \text{if } t = y \\ 0 & \text{otherwise} \end{cases}$

デルタ関数の場合

$e.g.\ \sum_t (x^2 + t^2) \cdot \max(1 - \frac{|t-y|}{\Delta t}, 0)$

三角形関数の場合
Δt は t の間隔

レンジ関数の近似（１／４）

- 補間関数を用いて近似
  - 要点
    ２変数 y_i と y_j を別々の関数に分けること

$$k_r(\underline{y_i}, \underline{y_j})$$

$$\sum_t k_r(\underline{y_i}, t)\psi(\underline{y_j}, t)$$

例えば，デルタ関数であれば

$$\psi(y_j, t) = \delta(y_j - t)$$

$$\sum_t k_r(y_i, t)\delta(y_j - t)$$

$$= k_r\underbrace{(y_i, y_j)\delta(y_j - y_j)}_{t = y_j} + \sum_{t \backslash y_j} \underbrace{k_r(y_i, t)\delta(y_j - t)}_{t \neq y_j}$$

$$= k_r(y_i, y_j) \times 1 + 0$$

レンジ関数の近似（２／４）

- フィルタの式への代入　と　順序の置換

$$\sum_{j \in N(i)} k_s(i,j)\,\underline{k_r(y_i,y_j)}\,y_j$$

⇩

$$\sum_{j \in N(i)} k_s(i,j)\,\underline{\sum_t k_r(y_i,t)\psi(y_j,t)}\,y_j$$

y_i と y_j の関数を分けたことで
組み換えが可能となった.

⇩

$$\sum_t k_r(y_i,t) \sum_{j \in N(i)} k_s(i,j)\psi(y_j,t)\,y_j$$

レンジ関数の近似（３／４）

- フィルタの正規化　と　事前計算可能箇所

$$\sum_t k_r(y_i, t) \sum_{j \in N(i)} k_s(i, j) \psi(y_j, t)\ y_j$$

$$\widetilde{k}_s(i, j) = \frac{1}{\sum_j k_s(i, j)} k_s(i, j)$$

なお，ローパスフィルタは通常このように正規化されている

$$\widetilde{y}_j^t = \psi(y_j, t)\ y_j$$

フィルタリング前に画像全体で計算ができる

$$\sum_t k_r(y_i, t) \sum_{j \in N(i)} \widetilde{k}_s(i, j)\ \widetilde{y}_j^t$$

レンジフィルタ

ガウシアンフィルタ

関数近似を用い，総和関数の位置を交換することで，レンジフィルタを外にくくり出せた

レンジ関数の近似（４／４）

- 分子も同様に計算
- 分子と分母を独立に計算した後, 各画素ごとに割り算を行う.

$$\frac{\displaystyle\sum_{j \in N(i)} k_s(i,j)\ k_r(y_i, y_j)\ y_j}{\displaystyle\sum_{j \in N(i)} k_s(i,j)\ k_r(y_i, y_j)}$$

$$\Rightarrow \sum_t k_r(y_i, t) \sum_{j \in N(i)} \widetilde{k}_s(i,j)(\psi(y_j, t)\ y_j)$$

$$\Rightarrow \sum_t k_r(y_i, t) \sum_{j \in N(i)} \widetilde{k}_s(i,j)(\psi(y_j, t)\ 1)$$

高効率計算（１／２）

```
• Ks = Ks / sum(Ks(:)); % 正規化
```

$$\widetilde{k}_s(i,j) = \frac{1}{\sum_j k_s(i,j)} k_s(i,j)$$

```
% 補間関数
T = 10; % [0,1] を分ける点数
psi = @(X,t,T) max(0,-abs(X-t)/(1/T)+1);

% 分子の計算
Nume = zeros( size(I) );

for t = 0:1/T:1

  PI = psi( I, t, T ) .* I;
  PI_s = imfilter( PI, Ks, 'replicate' );

  Nume = Nume + k_r( I, t, sr ) .* PI_s;
end
```

$$\widetilde{y}_j^t = \psi(y_j, t)\ y_j$$

$$\sum_{j \in N(i)} \widetilde{k}_s(i,j) \widetilde{y}_j^t$$

$$\sum_t k_r(y_i, t) \sum_{j \in N(i)} \widetilde{k}_s(i,j)\ \widetilde{y}_j^t$$

高効率計算（２／２）

```
% 分母の計算
Deno = zeros( size(I) );
U = ones( size(I) );

for t = 0:1/T:1

  PU = psi( I, t, T ) .* U;
  PU_s = imfilter( PU, Ks, 'replicate' );

  Deno = Deno + k_r( I, t, sr ) .* PU_s;
end

J_fast = Nume ./ Deno;


figure(5),  imshow( [I, J, J_fast] );
```

$$\psi(y_j, t)\ 1$$

補足：分子と分母の計算をまとめたもの

- ```
  Ks = Ks / sum(Ks(:)); % 正規化

  % 補間関数
  T = 10; % [0,1] を分ける点数
  psi = @(X,t,T) max(0,-abs(X-t)/(1/T)+1);

  U = ones( size(I) );

  Nume = zeros( size(I) );
  Deno = zeros( size(I) );

  for t = 0:1/T:1
    Pt = psi( I, t, T );
    Krt = k_r( I, t, sr );

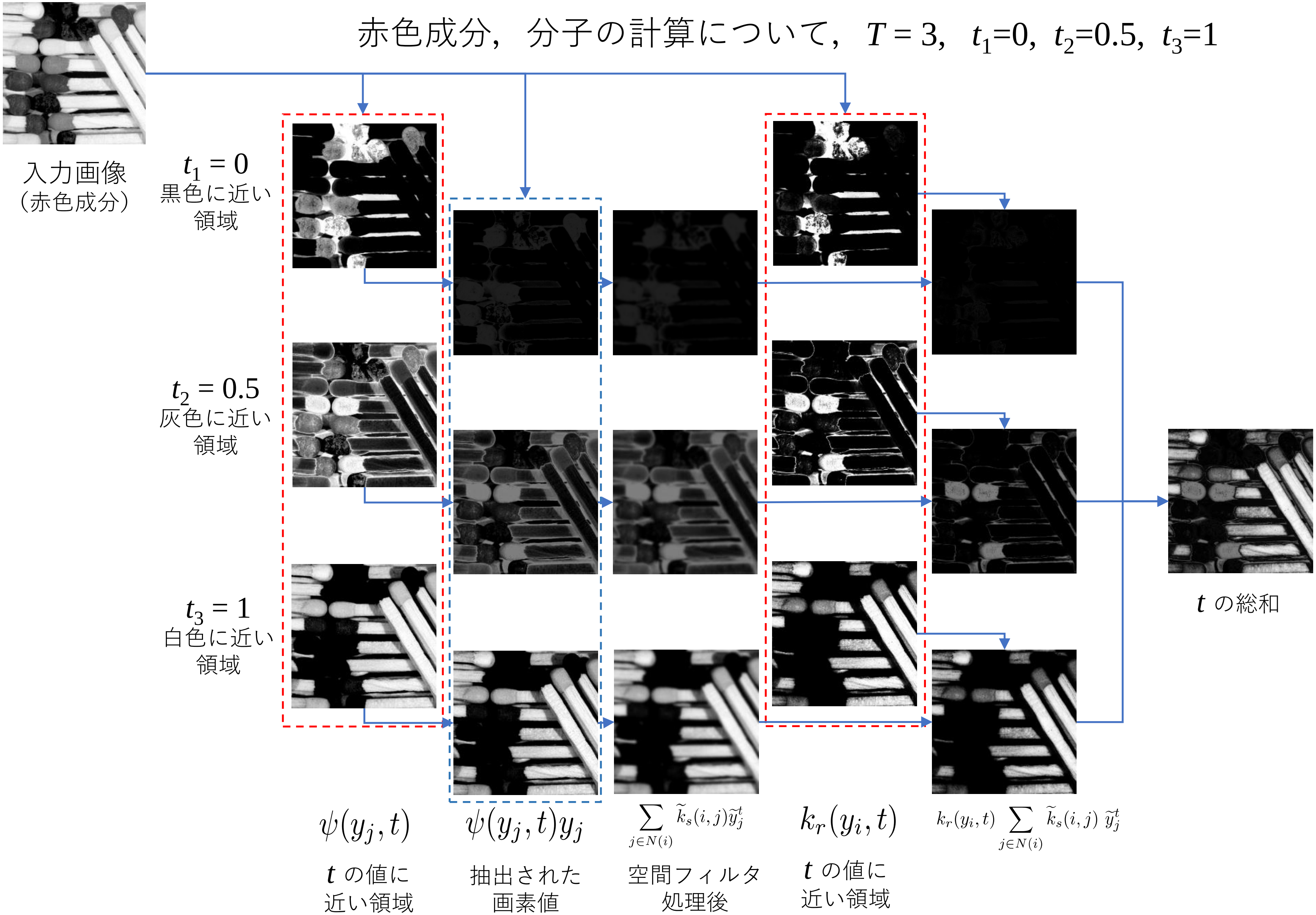
    Nume = Nume + Krt .* imfilter( Pt .* I, Ks, 'replicate' );
    Deno = Deno + Krt .* imfilter( Pt .* U, Ks, 'replicate' );
  end

  J_fast = Nume ./ Deno;


  figure(5),  imshow( [I, J, J_fast] );
  ```

この程度の量の
プログラムで書ける
  ```

アルゴリズムフローと途中画像

赤色成分，分子の計算について，$T = 3$，t_1=0，t_2=0.5，t_3=1

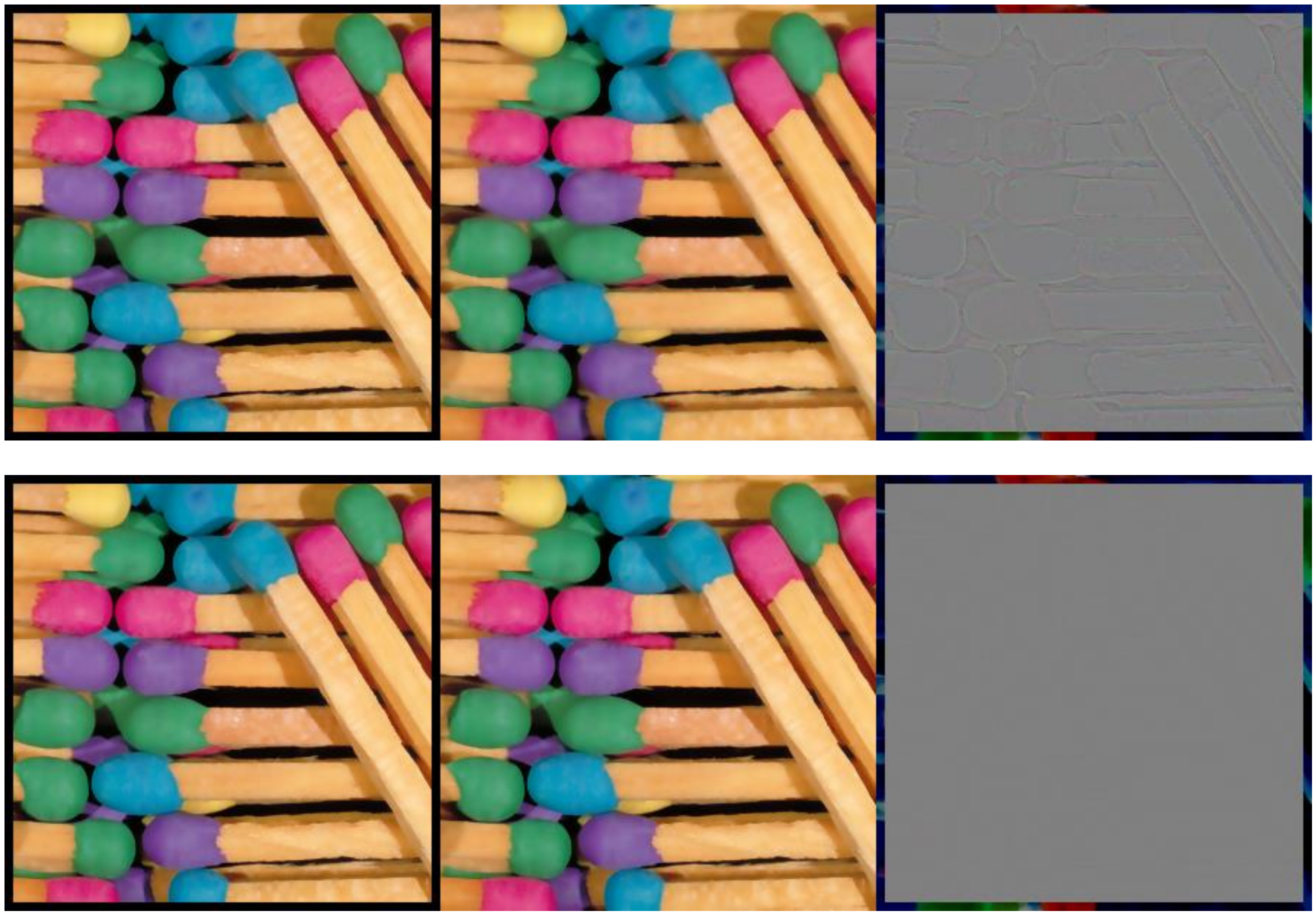


Naïve 版との比較

- 分割は **10** 個程度で十分

３個

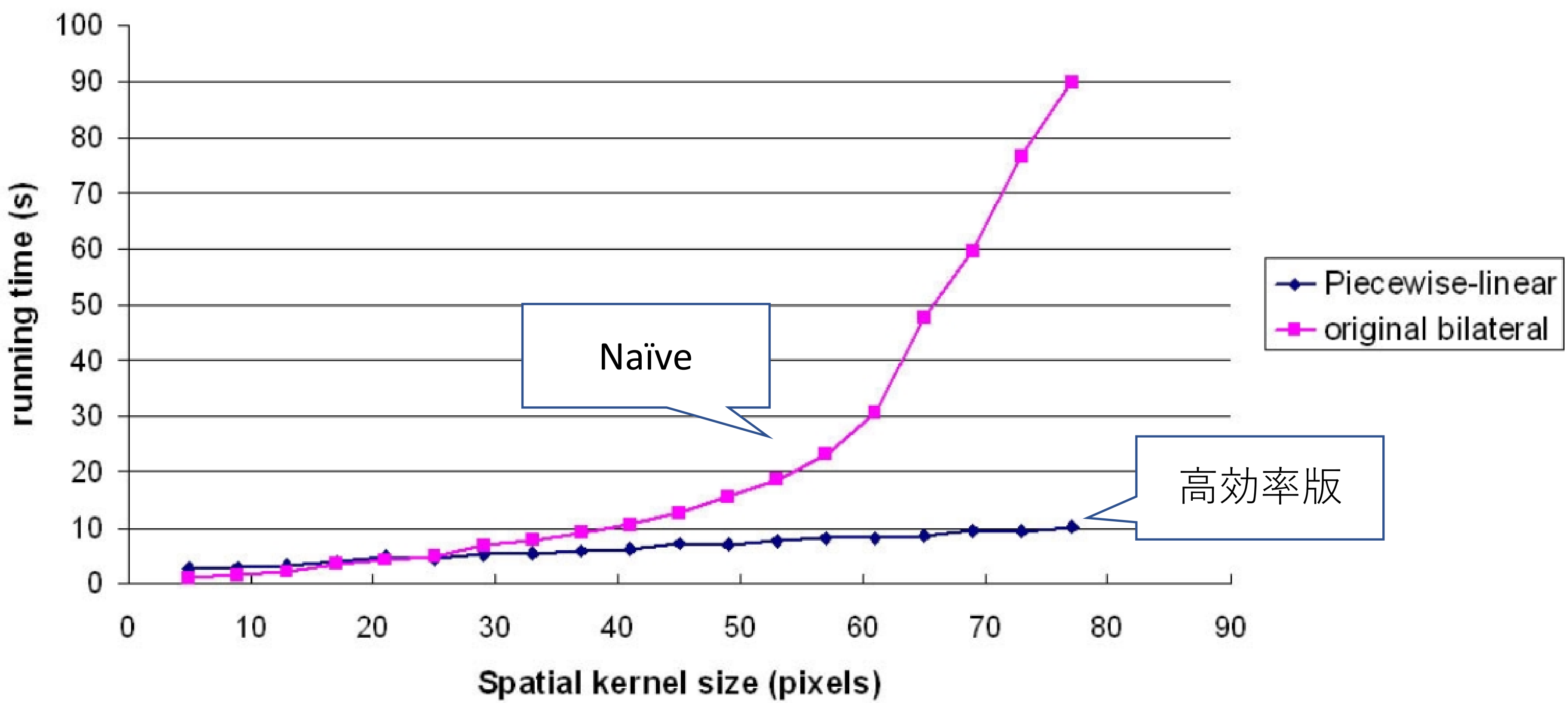
１０個



Naïve　高効率計算　差分

Naïve 版との処理速度の比較

- 論文より引用
  F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," ACM Trans. on Graphics, vol. 21, no. 3, pp. 257-266, 2002.



更なる高効率化

- 関数によっては,
効率の良い近似方法が存在する
  - ガウス関数であれば,
sin と cos 関数の組で近似でき,
積和の公式で分解可能

こんなところ
で使えた
加法定理

$$k_r(y_i, y_j) = \exp\left(-\frac{1}{2\sigma_r^2}(y_i - y_j)^2\right) \Rightarrow \sum_t \beta_t \cos(\alpha_t(y_i - y_j))$$

$$\Downarrow$$

$$\sum_t \beta_t \{\cos(\alpha_t y_i)\cos(\alpha_t y_j) + \sin(\alpha_t y_i)\sin(\alpha_t y_j)\}$$

3 個定度の t で近似できる

参考文献
K.N. Chaudhury, “Acceleration of the shiftable O(1) algorithm for bilateral filtering and nonlocal means,” IEEE Trans. Image Process., vol 22, no. 4, pp. 1291-1300, 2013.
K. Sugimoto and S. Kamata, “Compressive bilateral filtering,” IEEE Trans. on Image Process., vol. 24, no. 11, pp. 3357-3369, 2015.