Moodleの
出席確認を
提出しておいて
下さい。

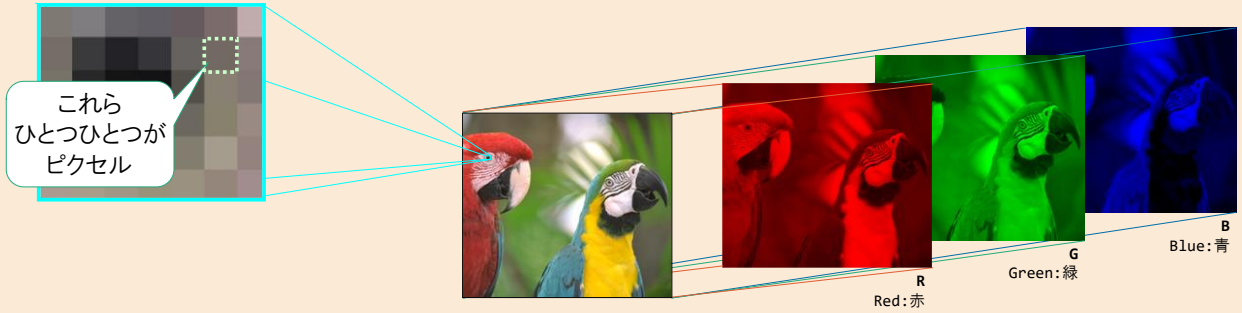
VisualStudio2019(等)で、
C言語+OpenCV のコーディングができる状態に
準備していてください。

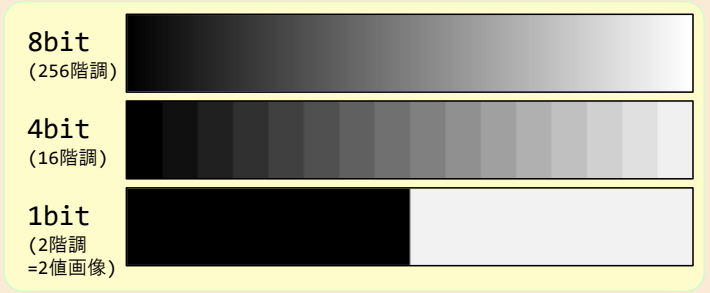
画像処理 (4J)

第11回

第6回のまとめ

- ラスタ画像とベクタ画像 ・・・ この授業では、ピクセル情報の集合であるラスタ画像を扱う
- 解像度 ・・・ 画像の大きさ(細かさ)
- ピクセル(画素) ・・・ ラスタ画像を構成する1つの点
- チャンネル ・・・ 1ピクセルをいくつの値で表現するか (例:RGBの3ch)
- 階調数 ・・・ 濃度を何段階で表現するか (例:8bit(=256段階))



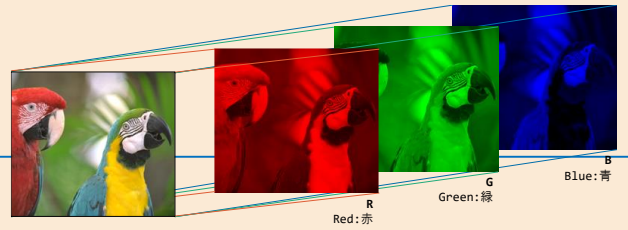


デジタル写真 = 有限の解像度で空間的にサンプリング(標本化)し、
有限の階調値で明るさを表現(量子化) したもの ...と捉えることができる。

※音声信号のデジタル化と対応させると、サンプリング周波数が解像度に、量子化bit数が階調数に、チャンネル数はそのまま対応する

第7回のまとめ

グレイスケール画像



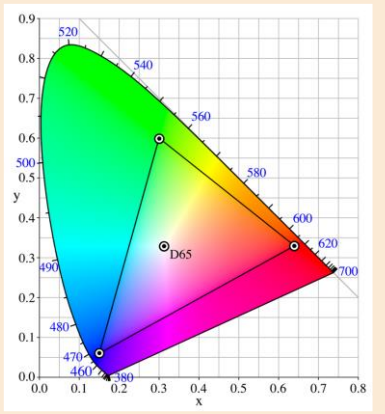
RGBカラー画像

●グレイスケール画像とカラー画像

- ➢グレイスケール画像は1つの(x,y)座標点に1つの濃度値 g(x,y)
- ➢RGBカラー画像は、1つの座標点に、3つの濃度値

●RGBカラー画像

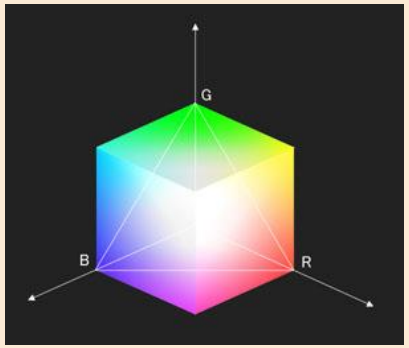
- ➢RGB値が同じでも、同じ色が表示されるとは限らない
- ➢sRGBに準拠させれば、一貫した色表現が可能。
  (ただし表現できる色域が狭い)

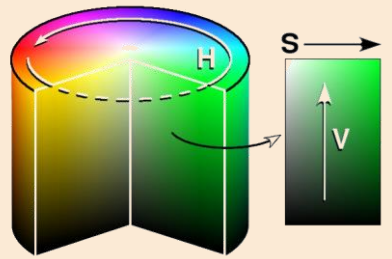


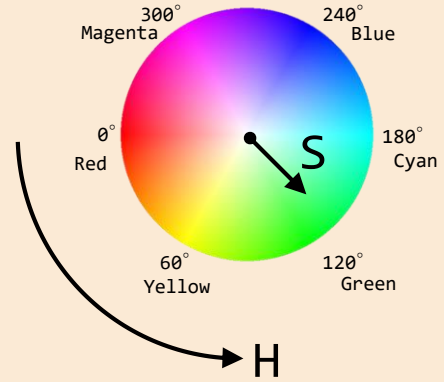
(CIE1931色空間のxy色度図上に示した)
sRGBの色域

●色空間： RGBとHSV

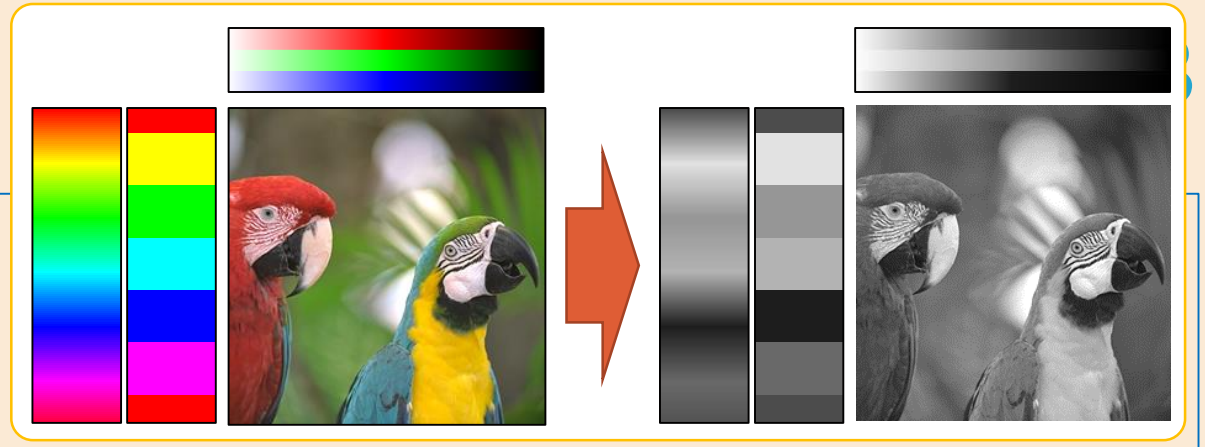
- ➢相互に変換可能
- ➢他にも様々な表色系がある







第8回まとめ



●グレイスケール化

➢**NTSC加重平均法**がよく使われる

$$Y = (0.298912 \times R + 0.586611 \times G + 0.114478 \times B)$$

●二値化

➢**閾値**を堺に、{0,1} の二値の画像に変換

➢閾値は任意に決められるが、画像統計量から閾値を自動決定する方法として**大津の方法**(判別分析法)が有名。

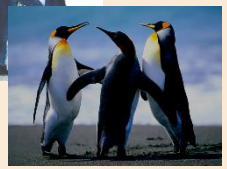
第9回まとめ

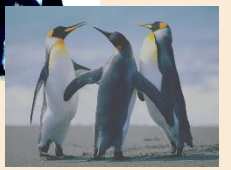
原画像 

階調反転

●濃度変換 ・・・ 濃度値を一定の方法で新しい濃度値に変換

- 線形変換 (Linear Stretch)　　$output = input \times a + b$
- ガンマ変換 (Gmma Stretch)　　$output = 255 \times \left(\frac{input}{255}\right)^{1/\gamma}$


輝度


コントラスト


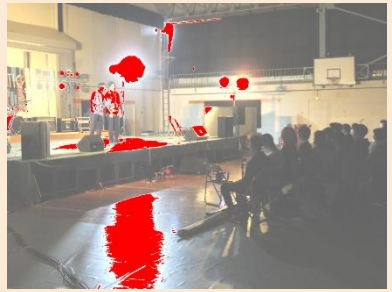
- 輝度調整、コントラスト調整、階調反転などに利用可能

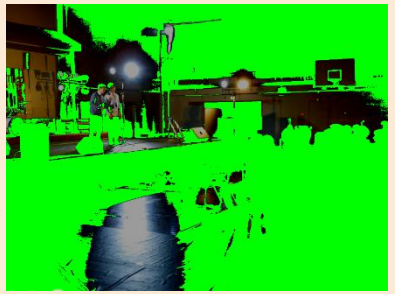


ガンマ

●濃度変換に伴う画像の劣化

- 白飛び　　・・・ 変換後に最大値以上になった場合に、最大値にクリップされる
- 黒つぶれ　　・・・ 変換後に最小値以下になった場合に、最小値にクリップされる
- 階調飛び(トーンジャンプ)　　・・・ 中間値の階調が失われ、濃度値が不連続に変化



白飛び



黒つぶれ

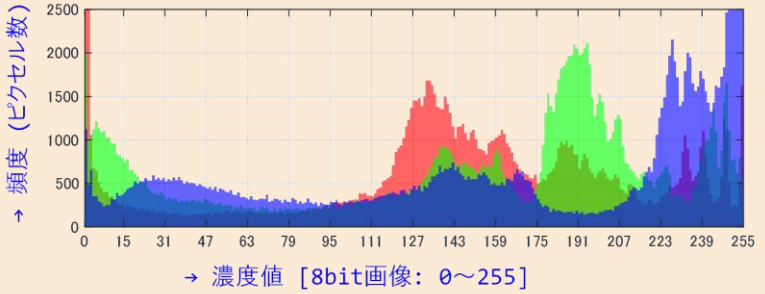


階調飛び

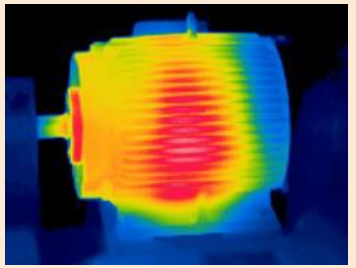
第10回まとめ

●ヒストグラム

- ➢濃度値の頻度(各濃度値が画像中にいくつあるか)を示したもの
- ➢ヒストグラムの形状から、画像の性質がある程度わかる

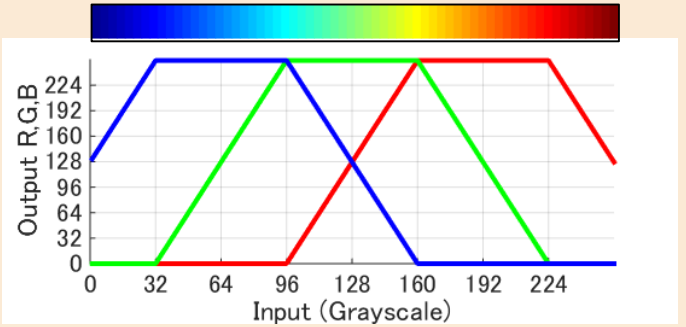


●濃度変換(2)

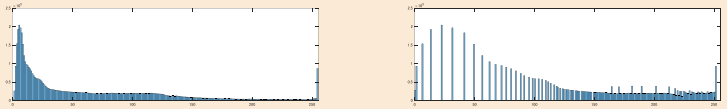


① 疑似カラー

- ➢ グレイスケール値に色(RGB値)を対応付けて表すもの
- ➢ 対応関係を示したもの： カラーマップ

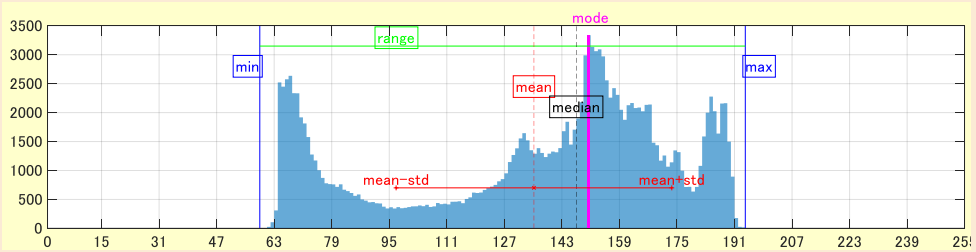


② ヒストグラム平坦化



●画像統計量

- ➢最大/最小/最頻
- ➢平均/中央
- ➢範囲/分散/標準偏差

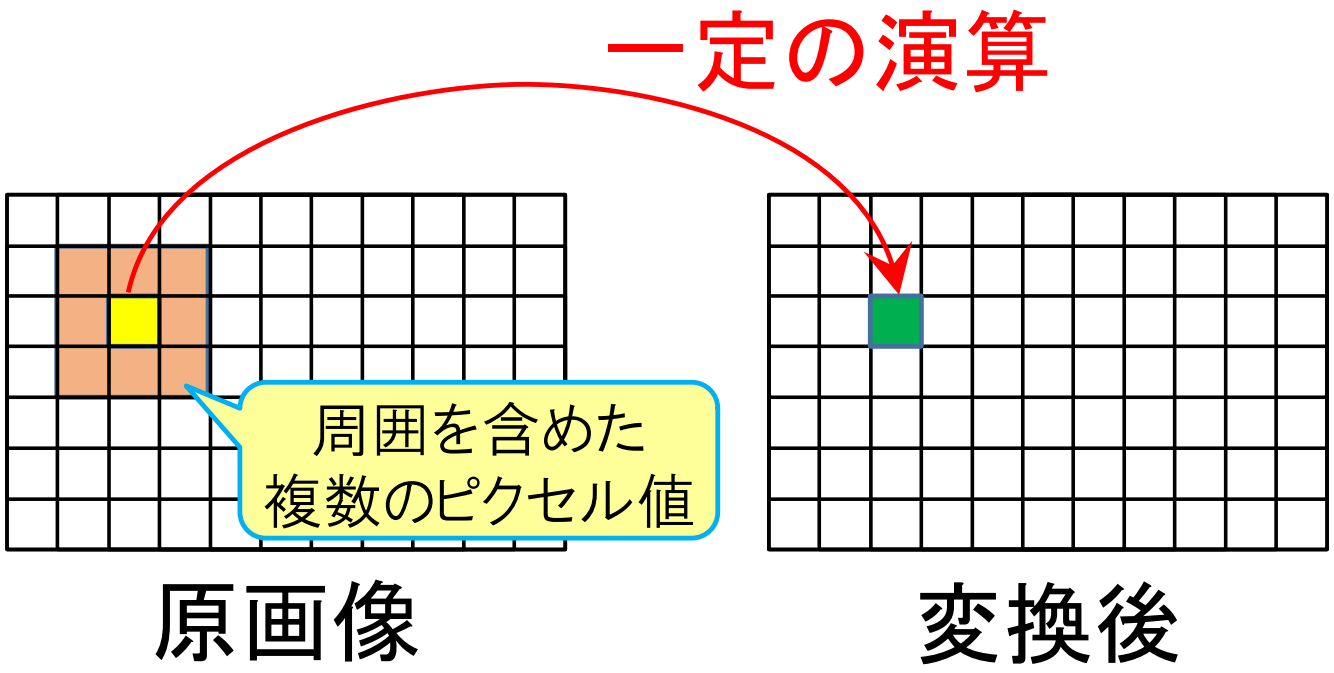


近傍演算(1)
画像の平滑化(ぼかし)

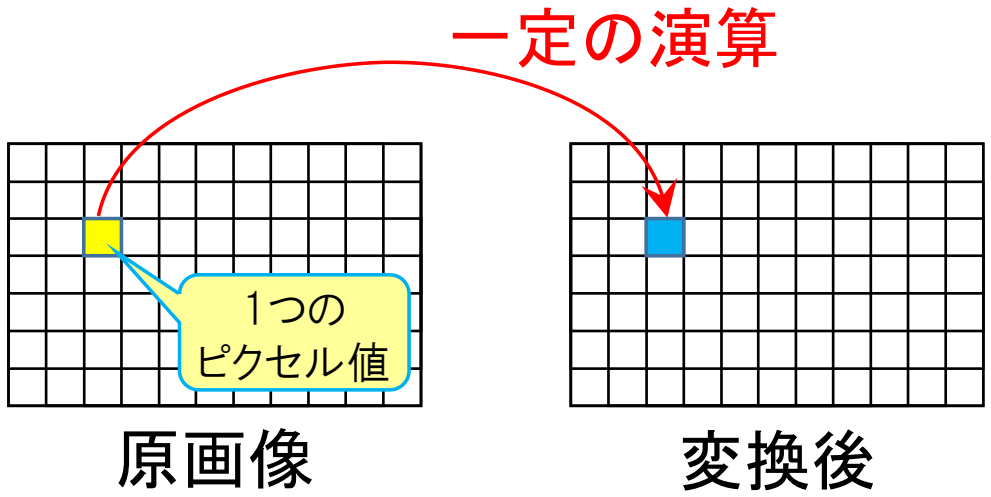
近傍演算とは

近傍演算

【近傍演算】



【階調変換】



階調変換の場合は、1つのピクセル値を一定の方法により変換し、新たなピクセル値に置き換えることで画像を操作する。

注目しているピクセルの近傍(周囲)を含めた
複数のピクセル値を用いて、新たなピクセル値を計算する。

近傍演算によるフィルタリング(畳み込み積分)

●移動平均による平滑化 【まずは 1次元の信号 の場合】

➢注目する点の周囲の点の平均を用いることで、信号の平滑化を行うことができる。

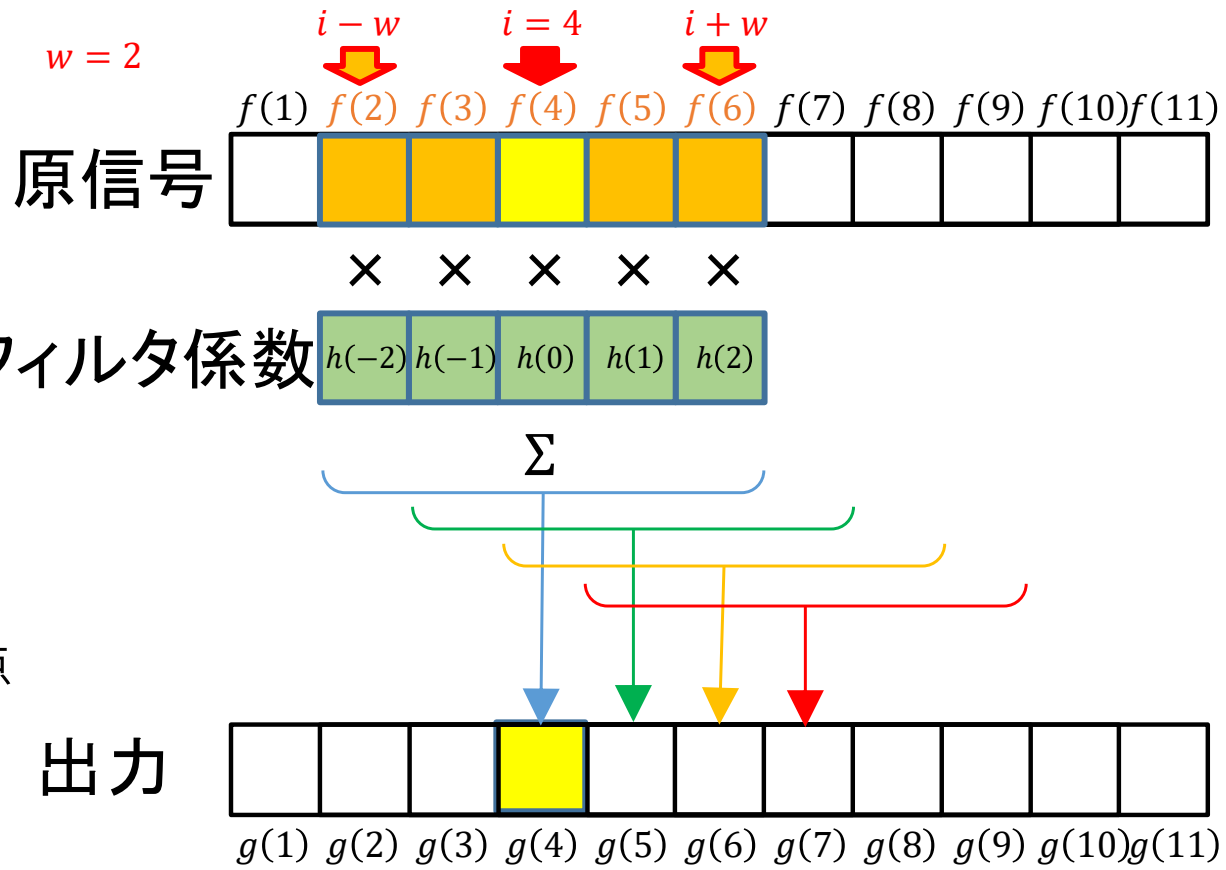
- $f(0), f(1), f(2), \ldots$ ： 原信号
- $g(0), g(1), g(2), \ldots$ ： 平滑化後の信号列

とすると、移動平均による平滑化は以下の
畳み込み積分 の式で示される。

$$g(i) = \sum_{n=-w}^{w} f(i+n)h(n)$$

ここで、

- $(2w+1)$ ： フィルタの大きさ
  例えば注目している i 番目の信号の
  前後 $w = 2$ 個の信号値を用いる場合、
  (前2点 + 後ろ2点 + i 番目の信号) = 合計5点
  の平均値を用いるということ
- $h(n)$ ： フィルタ係数
  原信号から取り出した各値に対する重み。

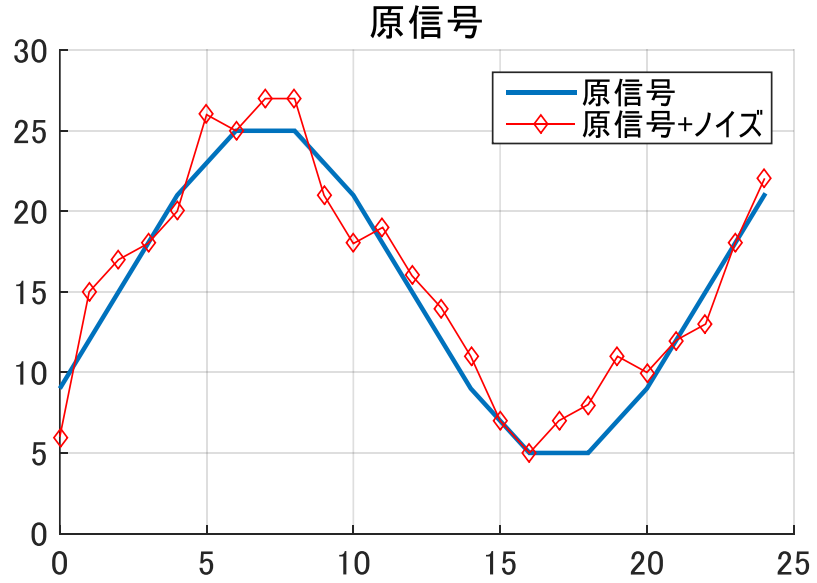


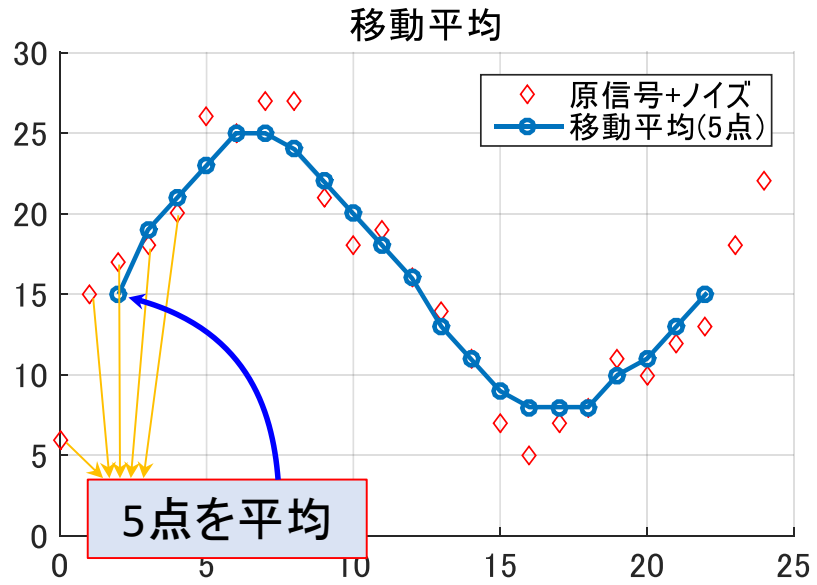
近傍演算による平滑化(Smoothing)=ぼかし(Blur)

①単純移動平均による平滑化 (1次元の例)

単純移動平均の場合はフィルタ係数 $h(n)$ に
全て等しい値 $\frac{1}{2w+1}$ を設定する。

- 上のプロット: 原信号(青線)と、
  原信号に適当なノイズを加えた信号(赤線)
- 下のプロット: 5点(w=2)の単純移動平均により
  平滑化を行った結果
  - 赤線に比べ、移動平均後は細かなギザギザ(=高周波成分)が低減され、滑らかな信号になっている。
    このように、平滑化フィルタは一種のLPF
    (Low Pass Filter: 低域透過フィルタ)として働く。
  - 移動平均後のプロットは、原信号に比べて2w 個分、信号点数が少なくなっている。これは、w番目以前と最後のw個の信号点については、フィルタの大きさ分の信号点が取り出せないからである。
    従って、このような端の方の処理を別に定義しておくか、さもなければデータ点数が減少するということに気をつける必要がある





②加重平均による平滑化(1次元の例)

●単純移動平均･･･近傍すべてが同じ重み　⇒　全体的に波形がなまる

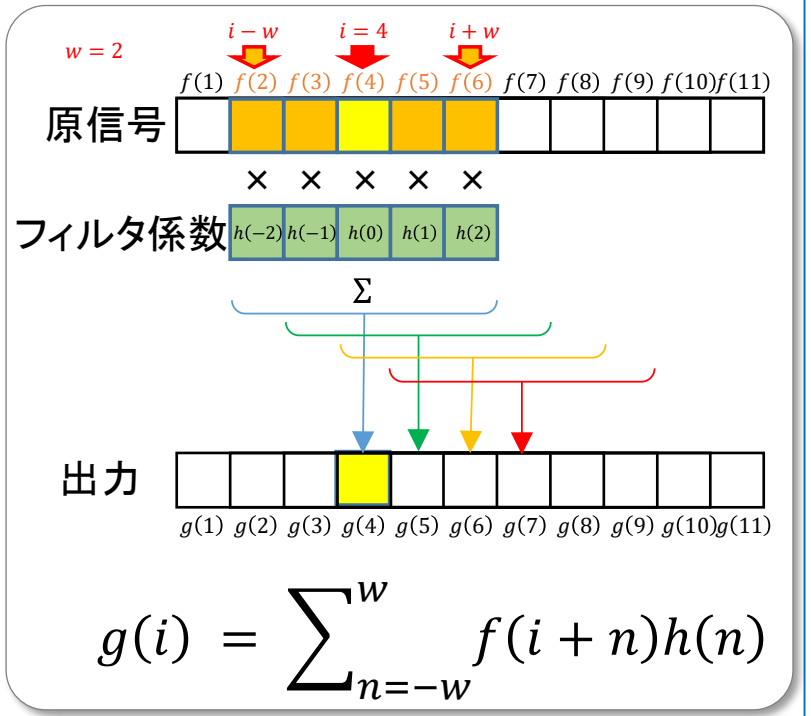
●この影響を抑制するには、
注目している　i番目　の信号に近い点の重みを大きくする　⇒　**加重平均**

➢ 重みは、フィルタ係数　$h(n)\,,(n=-w\sim+w)$ により指定できる。
なお、フィルタ係数の総和は 1.0 となるようにする。

例えば、フィルタのサイズが5点の場合、
$\{h(-2)=0.05,\ h(-1)=0.2,\ h(0)=0.5,\ h(1)=0.2,\ h(2)=0.05\}$
といったイメージ。

※フィルタ係数の総和 $\sum h(n)>1.0$ だと、
変換後の画像が全体的に明るくなってしまう。
（同様に $\sum h(n)<1.0$ だと暗くなってしまう。）

⇒　どのように重みを決定するか？



$$g(i)=\sum_{n=-w}^{w} f(i+n)h(n)$$

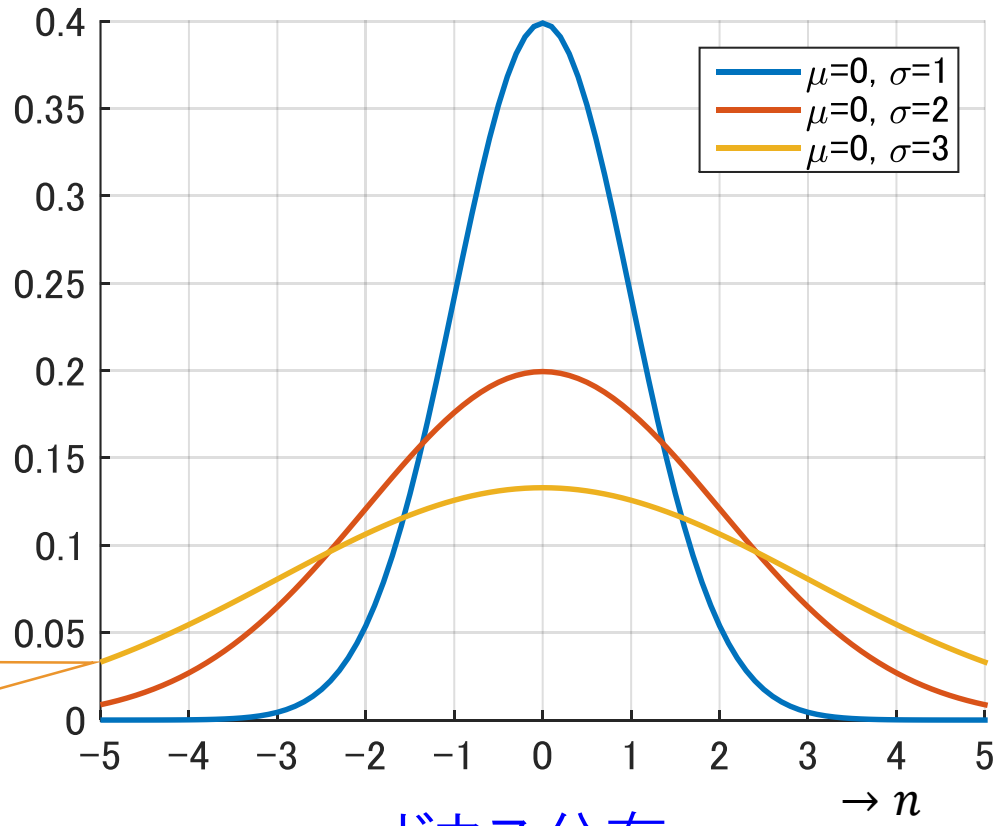
②加重平均による平滑化(1次元の例)

●【ガウシアンフィルタ】

… フィルタ係数を正規分布(ガウス分布)に近似させたもの

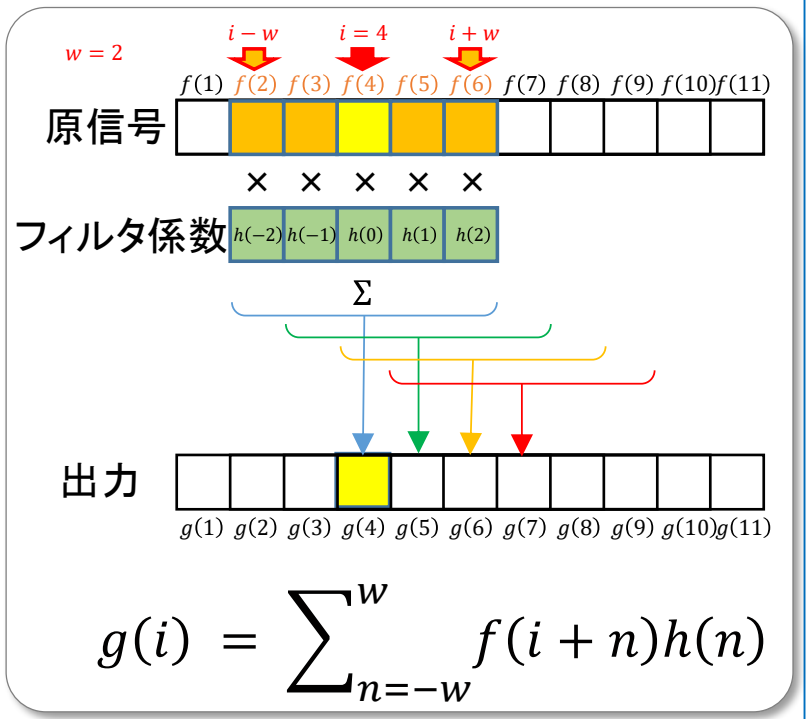
正規分布(平均： μ 、標準偏差： σ)

$$h(n) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{-(n-\mu)^2}{2\sigma^2}}$$



ガウス分布

σ が大きいほど、
平滑化の効果は
大きくなる。



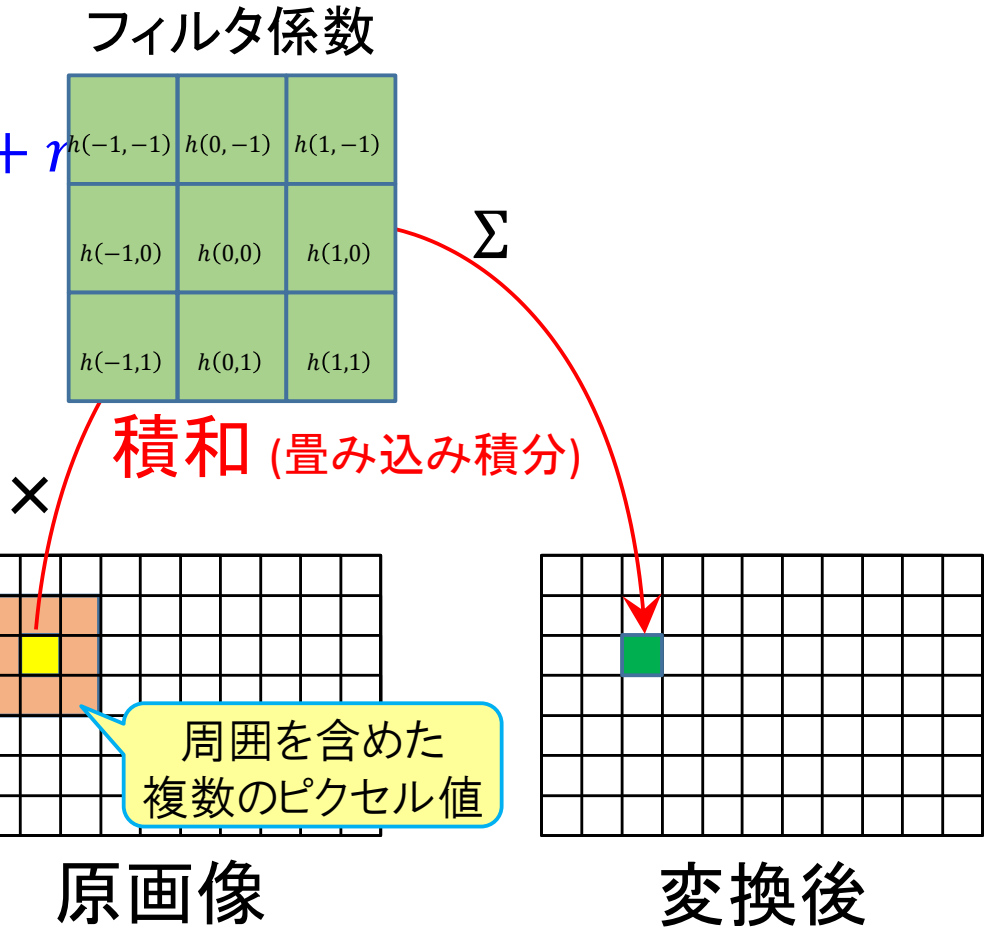
$$g(i) = \sum_{n=-w}^{w} f(i+n)h(n)$$

近傍演算によるフィルタリング(2次元への拡張)

●2次元の空間座標へ拡張　・・・　左右と上下方向の隣接信号(=ピクセル)を考え、注目ピクセル $f(x,y)$ を中心とした、画像上の四角い領域を考える

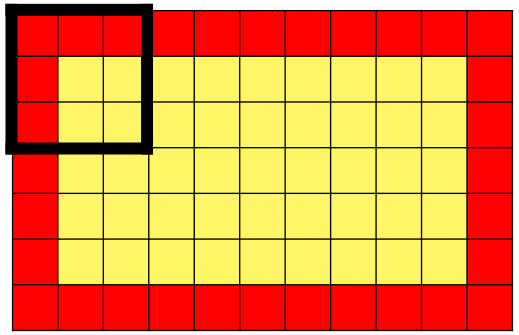
➢畳み込み積分の式は、以下のように拡張される。

$$g(x,y) = \sum_{n=-w}^{w} \sum_{m=-w}^{w} f(x+m, y+n)$$



- フィルタサイズは $(2w+1) \times (2w+1)$

- 1次元の場合と同様に、上下左右の端の、各wピクセル分変換後の画像は小さくなる。
(必要な隣接信号点を
取り出せないため)
あるいは別に端の処理を
定義しておく必要がある。



①単純移動平均による平滑化(画像(2次元)の場合)

●単純移動平均の場合：（1次元の場合と同様に…）

- ➤フィルタ内のどの加重も同一
- ➤フィルタ係数の合計が 1.0

となるようにフィルタ係数 $h(m,n)$ を設定する。

$w=1$ の場合
(3×3)

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

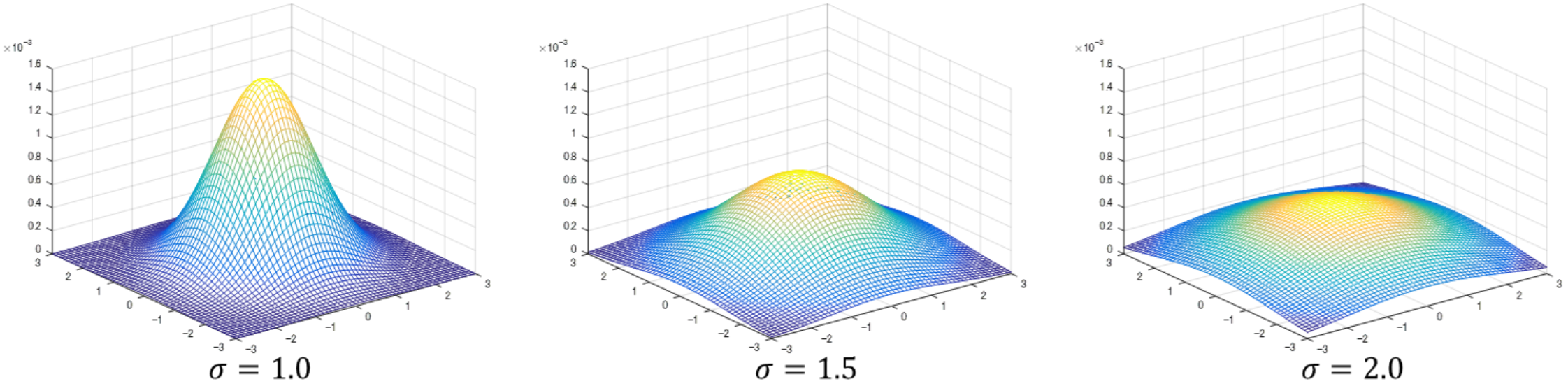
$w=2$ の場合
(5×5)

$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$

②加重平均による平滑化(画像(2次元)の場合)

●ガウシアンフィルタは、以下の2次元正規分布に従ってフィルタ係数を設定する。

➢ $h_1(m,n) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{m^2+n^2}{2\sigma^2}\right)}$



2次元正規分布

➢ ただし、実際には、フィルタ係数の総和が 1.0 になるように正規化する必要がある。
具体的には･･･(次スライド)

②加重平均による平滑化(画像(2次元)の場合)

●ガウシアンフィルタのフィルタ係数の計算(実際)

1．まず、比例定数を無視した仮の係数 $h_2(m,n)$ を求め、求めた係数の総和 s を求める

$$h_2(m,n) = e^{-\left(\frac{m^2+n^2}{2\sigma^2}\right)}$$

$$s = \sum_{n=-w}^{w} \sum_{m=-w}^{w} h_2(m,n)$$

※2次元正規分布 $h_1(m,n) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{m^2+n^2}{2\sigma^2}\right)}$ の比例定数 $\frac{1}{2\pi\sigma^2}$ を無視して計算。

$$\frac{1}{2\pi\sigma^2}$$

2．仮の係数を、係数の総和で除算することで、実際のフィルタ係数 $h(m,n)$ を決定する

$$h(m,n) = \frac{h_2(m,n)}{s}$$

②加重平均による平滑化(画像(2次元)の場合)

ガウシアンフィルタのフィルタ係数の計算例：　$\sigma = 1.0, w = 1$ の場合

1．仮の係数　$h_2(m,n)$ を求めると、

$h_2(-1,-1) = 0.3679$	$h_2(0,-1) = 0.6065$	$h_2(1,-1) = 0.3679$
$h_2(-1,0) = 0.6065$	$h_2(0,0) = 1.0000$	$h_2(1,0) = 0.6065$
$h_2(-1,1) = 0.3679$	$h_2(0,1) = 0.6065$	$h_2(1,1) = 0.3679$

仮の係数の総和は、　$s = 4.8976$

2．仮の係数を、係数の総和で除算することで、フィルタ係数　$h(m,n)$ を求めると･･･

$h(-1,-1) = 0.0751$	$h(0,-1) = 0.1238$	$h(1,-1) = 0.0751$
$h(-1,0) = 0.1238$	$h(0,0) = 0.2042$	$h(1,0) = 0.1238$
$h(-1,1) = 0.0751$	$h(0,1) = 0.1238$	$h(1,1) = 0.0751$

演習：

①単純移動平均による平滑化
②ガウシアンフィルタによる平滑化

OpenCVを使った画像生成の流れ

【画像生成時の流れ】

```
IplImage* img = cvCreateImage(CvSize size, IPL_DEPTH_8U, int channels);
 ↓                                                  ･･･ 画像を扱うための構造体 img を生成する
cvSetZero(img);                ･･･ 画像データ img を 0(=黒) で初期化
 ↓
 ↓      【画像読み込み時の流れ】
 ↓     IplImage* img = cvLoadImage(const char* filename, CV_LOAD_IMAGE_UNCHANGED);
 ↓     ↓                        ･･･ 画像ファイルを読み取り、画像データ img を生成
 ↓     ↓
<img に対する何らかの処理>
 ↓
 ↓
cvSaveImage(img);              ･･･ 画像データ img を画像ファイルとして保存
 ↓
cvReleaseImage(&img);          ･･･ 画像を扱うための構造体 img に割り当てたメモリの開放
```

各種関数のリファレンス(1)


```
IplImage* img
 = cvCreateImage(CvSize size, int depth, int channels);
```

- size:　　画像のサイズ。
- depth:　　ピクセルのデータ形式。
  ※本授業では常に IPL_DEPTH_8U (符号無し8ビット整数 = unsigned char)とする。
- channels: ピクセル毎のチャンネル数。[グレイスケール = 1 , カラー = 3]

※ロードに失敗した場合は NULL が返る。
内部でmalloc()されているので、cvReleaseImage()で開放する必要がある。

```
typedef struct CvSize {
  int width;    /* 横幅 */
  int height:   /* 高さ */
} CvSize;
```

各種関数のリファレンス(2)

再


```
void cvSetZero(IplImage *img);
```

- img: cvCreateImage() が返した IplImage* のアドレス。
  全ピクセルデータを 0(黒)で初期化する

```
IplImage* img
 = cvLoadImage(const char* filename, int iscolor);
```

- filename:　　ファイル名。対応ファイル形式は(表1)を参照。
- iscolor:　　読み込む画像のカラーの種類。
  ※本授業では常に CV_LOAD_IMAGE_UNCHANGED とする。

指定した画像ファイルを IplImage 形式に読み込む

※内部でmalloc()されているので、cvReleaseImage()で開放する必要がある。

各種関数のリファレンス(3)

再


```
int cvSaveImage(const char* filename, IplImage* image);
```

- filename:　ファイル名。拡張子で保存形式が決まる。→ (表1)を参照。
- image:　保存する画像データ

IplImage を、画像ファイルとして保存する。

※保存に成功した場合は 1 、失敗した場合は 0 が返る(らしい)。

```
void cvReleaseImage(IplImage** img);
```

- img: cvCreateImage() が返した IplImage* のアドレス。

cvCreateImage()やcvLoacImage()で確保された領域を開放する。

（表 1）cvLoacImage()、 cvSaveImage() の対応形式と、指定する拡張子

形式	Windows bitmaps	Jpeg	Portable Network Graphics	Portable image format	Sun rasters	TIFF files	OpenEXR HDR images	JPEG 2000 images
拡張子	BMP,DIB	JPEG, JPG,JPE	PNG	PGM,PGM PPM	SR,RAS	TIFF, TIF	EXR	Jp2

IplImage 構造体（types_c.h 内で定義）再


```
typedef struct _IplImage
{
    int  nSize;             /* sizeof(IplImage) */
    int  ID;                /* version (=0)*/
    int  nChannels;         /* Most of OpenCV functions support 1,2,3 or 4 channels */
    int  alphaChannel;      /* Ignored by OpenCV */
    int  depth;       /* Pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16S,
                         IPL_DEPTH_32S, IPL_DEPTH_32F and IPL_DEPTH_64F are supported.  */
    char colorModel[4];     /* Ignored by OpenCV */
    char channelSeq[4];     /* ditto */
    int  dataOrder;     /* 0 - interleaved color channels, 1 - separate color channels.
                           cvCreateImage can only create interleaved images */
    int  origin;            /* 0 - top-left origin,
                               1 - bottom-left origin (Windows bitmaps style).  */
    int  align;             /* Alignment of image rows (4 or 8).
                               OpenCV ignores it and uses widthStep instead.    */
    int  width;             /* Image width in pixels.                           */
    int  height;            /* Image height in pixels.                          */
    struct _IplROI *roi;    /* Image ROI. If NULL, the whole image is selected. */
    struct _IplImage *maskROI;      /* Must be NULL. */
    void  *imageId;                 /* "             " */
    struct _IplTileInfo *tileInfo;  /* "             " */
    int  imageSize;         /* Image data size in bytes
                               (==image->height*image->widthStep
                               in case of interleaved data)*/
    char *imageData;        /* Pointer to aligned image data.         */
    int  widthStep;         /* Size of aligned image row in bytes.    */
    int  BorderMode[4];     /* Ignored by OpenCV.                     */
    int  BorderConst[4];    /* Ditto.                                 */
    char *imageDataOrigin;  /* Pointer to very origin of image data
                               (not necessarily aligned) -
                               needed for correct deallocation */

}
IplImage;
```

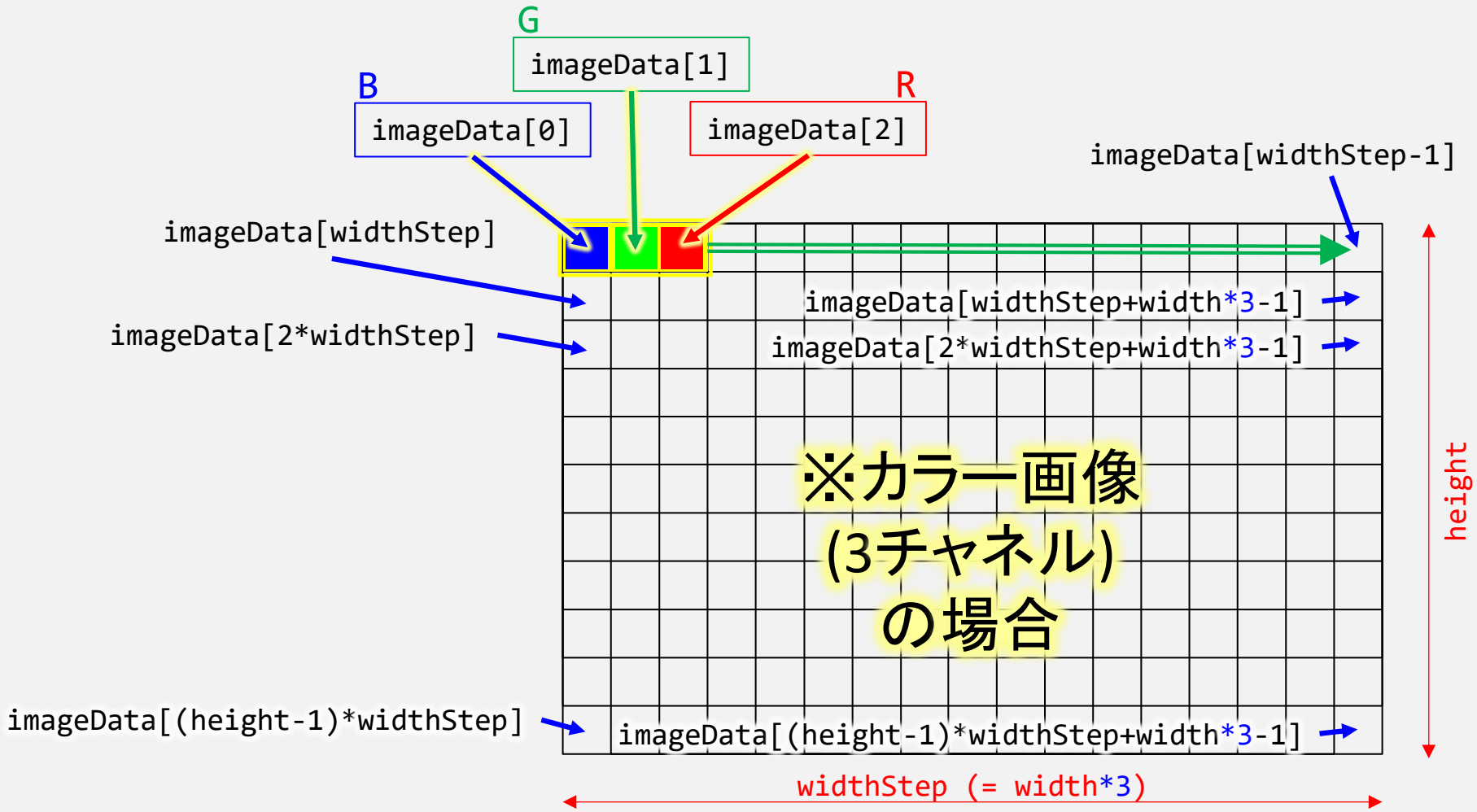
カラー画像／グレイスケール画像の判定 再

- IplImage の メンバ変数の nChannels
  - 3の場合カラー画像
  - 1の場合グレイスケール画像

(※本授業では、nChannelsが1か3の場合のみ、取り扱うものとする)

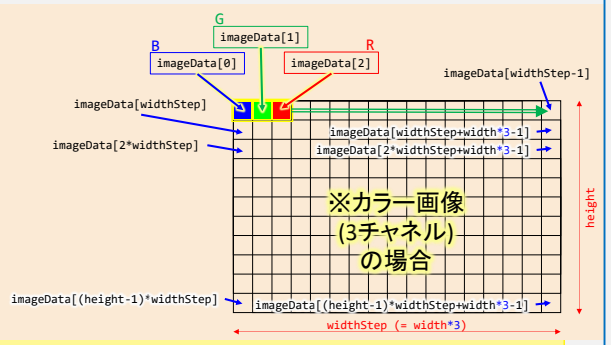
IplImageのRGB値へのアクセス 再

RGBカラー画像の個々のRGB値は、
下図のような順に一次元配列 imageData[] に格納される。



IplImageのRGB値へのアクセス　再（変更後）

- IplImage のメンバ変数を用いて、個々のピクセルへアクセスする。
- imageDataには、
  BGRBGRBGRBGR……の順で格納されていることに注意 （RGBの順ではない！）。
  - char* imageData ･･･ 画像データへのポインタ
  - int　 widthStep ･･･ 画像データ1ライン分のバイト数(= char で数えた数)



【例】
IplImage *img の画像(RGBカラー画像)に対して、
座標点 (x, y) のカラーチャネルごとのピクセル値(RGB値)へは、

```
b = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 0];
g = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 1];
r = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 2];
```

としてアクセスすることが出来る。

演習 &
課題No.11

●注意(ヒント)

- カラー画像(特にpng画像)を読み込んだのに、 `nChannels == 3` でない場合、インデックスカラーや、透明チャンネルあり(つまり `nChannels == 4`)の場合があります。
- そのような場合は、ペイント等で開いて、BMP形式で保存したものを用いて下さい。

課題に使用するツール


```
// ノイズを付加するツール
// Tool-11 : ホワイトノイズ　または　ごま塩ノイズ　を付加
//

#include <stdio.h>
#include <opencv/highgui.h>

void writeFile(IplImage* img, char* imgFilename, char* addName);
unsigned char clip(int gray);
void addSoltAndPepper(IplImage* img, double noiseLv);
void addWhite(IplImage* img, double noiseLv);
void addNoise(IplImage* img, int noiseType, double noiseLv);

// ========================================================================

// ノイズ付加画像の書き出し（ファイル名も自動生成）
void writeFile(IplImage* img, char* imgFilename, const char* addName) {
    char filename[256];
    char* ext_p;

    strcpy_s(filename, 256, imgFilename); // 読み込んだ画像のファイル名をコピー
    ext_p = strrchr(filename, '.');
    *ext_p = '¥0';  // ファイル名から拡張子を消す
    sprintf_s(filename, 255, "%s+%s.bmp", filename, addName); // ノイズ名を付加したファイル名を生成

    cvSaveImage(filename, img);
}

// 値を 0-255 の範囲にクリップし、unsigned char で返す
unsigned char clip(int gray) {
    return (gray < 0) ? 0 : (gray > 255) ? 255 : gray;
}


// ごま塩ノイズの付加
void addSoltAndPepper(IplImage* img, double noiseLv) {
    for (int y = 0; y < img->height; y++) {
        for (int x = 0; x < img->width; x++) {
            for (int ch = 0; ch < img->nChannels; ch++) {  // グレイスケール画像・カラー画像どちらにも対応
                if (rand() < RAND_MAX * noiseLv) {
                    if (rand() < RAND_MAX / 2) {
                        img->imageData[y * img->widthStep + x * img->nChannels + ch] = 0;
                    }
                    else {
                        img->imageData[y * img->widthStep + x * img->nChannels + ch] = 255;
                    }
                }
            }
        }
    }
}

// ホワイトノイズ（簡易）の付加
void addWhite(IplImage* img, double noiseLv) {
    for (int y = 0; y < img->height; y++) {
        for (int x = 0; x < img->width; x++) {
            for (int ch = 0; ch < img->nChannels; ch++) {  // グレイスケール画像・カラー画像どちらにも対応
                img->imageData[y * img->widthStep + x * img->nChannels + ch]
                    = clip((unsigned char)img->imageData[y * img->widthStep + x * img->nChannels + ch]) + ((float)rand() / RAND_MAX * 511 - 255) * noiseLv);

            }
        }
    }
}

// ノイズの種類により呼び出す関数を振り分け
void addNoise(IplImage* img, int noiseType, double noiseLv) {
    switch (noiseType) {
    case 0:
    default:
        addWhite(img, noiseLv);  // ホワイトノイズを付加
        break;

    case 1:
        addSoltAndPepper(img, noiseLv);  // ごま塩ノイズを付加
        break;
    }
}


void main(int argc, char* argv[])
{
    IplImage* img1;
    IplImage* img2;
    int noiseType = 0;      // ノイズタイプ [0(Default):ホワイトノイズ, 1:ごま塩ノイズ]
    double noiseLv = 0.05;  // ノイズレベル [noiseType=0 : ノイズの強度, noiseType=1 : ノイズの生起割合] (Default=0.05)
    const char* noiseStr[] = { "WhiteNoise", "Salt&PepperNoise" };

    // 起動オプションのチェック
    printf("argc = %d¥n", argc);
    for (int k = 0; k < argc; k++) {
        printf("argv[%d] = %s¥n", k, argv[k]);
    }
    printf("¥n¥n");

    if (argc < 2) {
        printf("ファイル名を指定してください。¥n");
        return;
    }

    if (argc >= 3) {
        noiseType = atoi(argv[2]);  // ノイズレベルの設定
    }
    printf("Noise Type = %d (%s)¥n", noiseType, noiseStr[noiseType]);


    if (argc >= 4) {
        noiseLv = atof(argv[3]);  // ノイズレベルの設定
    }
    printf("Noise Lv = %f¥n", noiseLv);

    // 画像データの読み込み
    if ((img1 = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED)) == NULL) {  // 読み込んだ画像はカラーの場合も、グレイスケール画像の場合もある
        printf("画像ファイルの読み込みに失敗しました。¥n");
        return;
    }
    // 読み込んだ画像の表示
    cvNamedWindow("Original");
    cvShowImage("Original", img1);

    // ------------------------------------------------------
    img2 = cvCreateImage(cvSize(img1->width, img1->height), img1->depth, img1->nChannels);  // 読み込んだ画像と同じ大きさの画像を生成
    cvCopy(img1, img2); // 画像データをコピー

    addNoise(img2, noiseType, noiseLv);


    cvNamedWindow(noiseStr[noiseType]);
    cvShowImage(noiseStr[noiseType], img2);

    writeFile(img2, argv[1], noiseStr[noiseType]);        // ノイズ付加画像をファイルに出力

    cvWaitKey(0);
    cvDestroyAllWindows();
}
```

起動オプション：
(1) 入力ファイル名
(2) ノイズ種類
　[0:ホワイトノイズ,1:ごま塩ノイズ]
(3) ノイズレベル（0.0～1.0）

※入力ファイル名が[Lenna.bmp]の場合、
[Lenna+WhiteNoise.bmp]のようなポスト
フィックスのついたファイル名で出力されます。

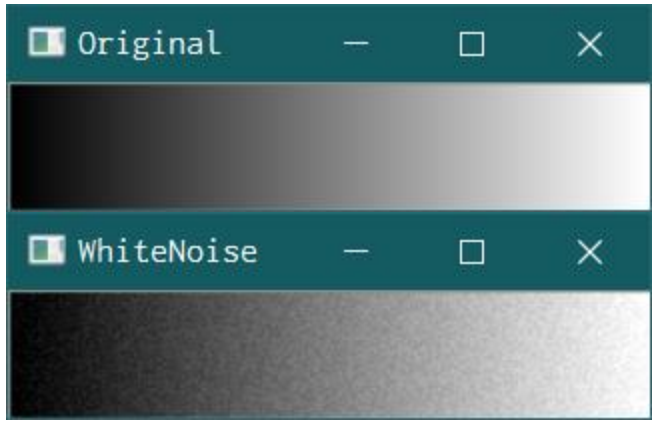
ホワイトノイズの場合①(グレイスケール画像の例):

※ファイル名のみ指定した場合は、ホワイトノイズ(ノイズ種類=0)と解釈する。(ノイズレベルはホワイトノイズのデフォルトの 0.05 となる)

●Tool-11.exe 入力ファイル名 [0] [0.0～1.0(ノイズレベル)]

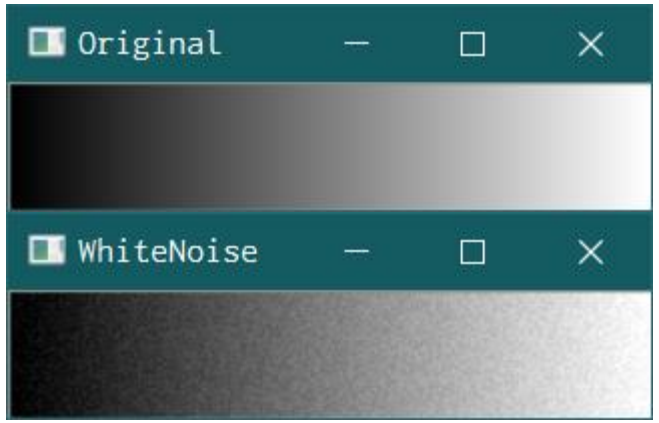
例1: ファイル名のみ指定

```
>Tool-11.exe Gray.bmp
argc = 2
argv[0] = Tool-01.exe
argv[1] = Gray.bmp


Noise Type = 0 (WhiteNoise)
Noise Lv = 0.050000
```



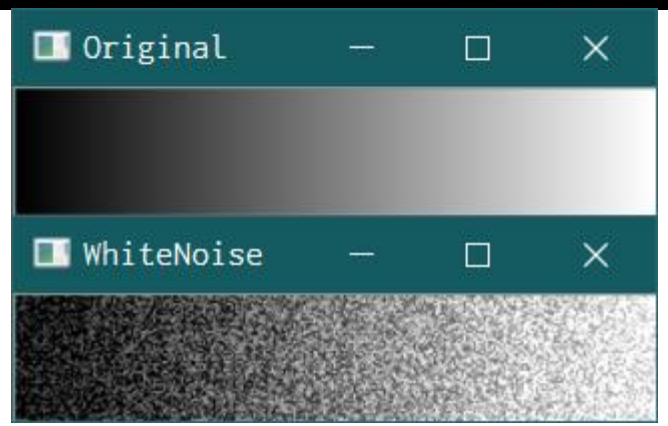
例2: ファイル名とノイズ種類のみ指定

```
>Tool-11.exe Gray.bmp 0
argc = 3
argv[0] = Tool-01.exe
argv[1] = Gray.bmp
argv[2] = 0


Noise Type = 0 (WhiteNoise)
Noise Lv = 0.050000
```



例3: ノイズレベルも指定

```
>Tool-11.exe Gray.bmp 0 0.3
argc = 4
argv[0] = Tool-01.exe
argv[1] = Gray.bmp
argv[2] = 0
argv[3] = 0.3


Noise Type = 0 (WhiteNoise)
Noise Lv = 0.300000
```



ホワイトノイズの場合②(カラー画像の例):

※ファイル名のみ指定した場合は、ホワイトノイズ(ノイズ種類=0)と解釈する。(ノイズレベルはホワイトノイズのデフォルトの 0.05 となる)

●Tool-11.exe 入力ファイル名 [0] [0.0～1.0(ノイズレベル)]

例1: ファイル名のみ指定

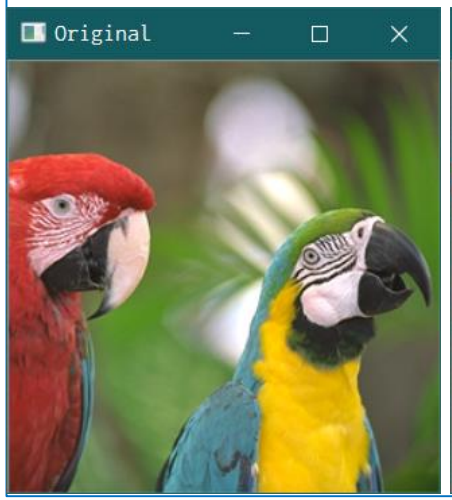
```
>Tool-11.exe
Parrots(Color).bmp
argc = 2
argv[0] = Tool-11.exe
argv[1] = Parrots(Color).bmp


Noise Type = 0 (WhiteNoise)
Noise Lv = 0.050000
```

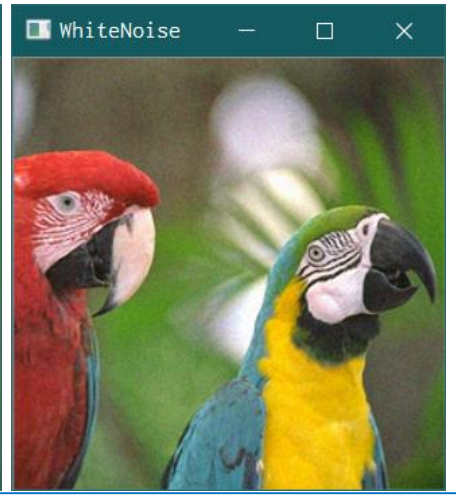
例2: ファイル名とノイズ種類のみ指定

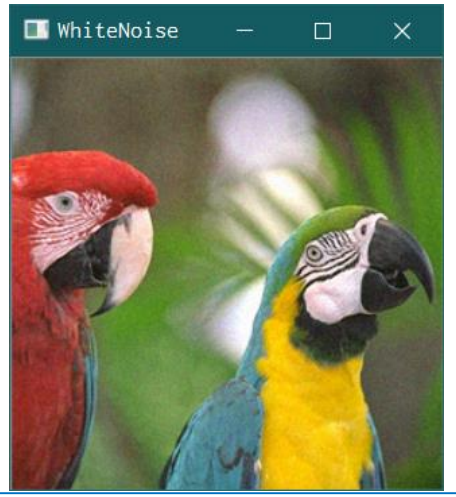
```
>Tool-11.exe
Parrots(Color).bmp 0
argc = 3
argv[0] = Tool-11.exe
argv[1] = Parrots(Color).bmp
argv[2] = 0


Noise Type = 0 (WhiteNoise)
Noise Lv = 0.050000
```

例3: ノイズレベルも指定

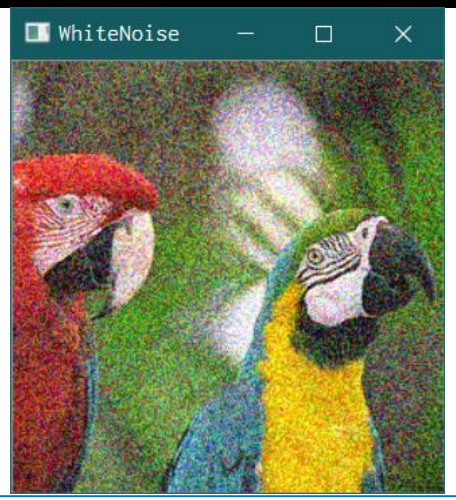
```
>Tool-11.exe Gray.bmp 0 0.3
argc = 4
argv[0] = Tool-01.exe
argv[1] = Gray.bmp
argv[2] = 0
argv[3] = 0.3


Noise Type = 0 (WhiteNoise)
Noise Lv = 0.300000
```







←
※例1と同じ



ごま塩ノイズの場合①(グレイスケール画像の例):

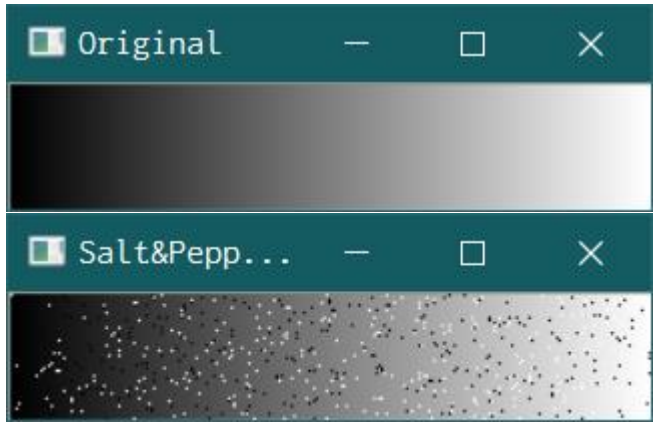
※ (ごま塩ノイズのデフォルトのノイズレベルは 0.05 となる)

●Tool-11.exe 入力ファイル名 1 [0.0～1.0(ノイズレベル)]

※ファイル名のみ指定した場合は、
ホワイトノイズになる。

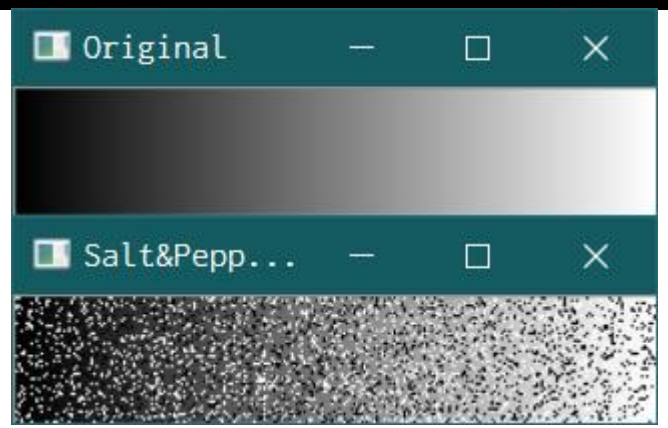
例1: ファイル名とノイズ種類のみ指定

```
>Tool-11.exe Gray.bmp 1
argc = 3
argv[0] = Tool-11.exe
argv[1] = Gray.bmp
argv[2] = 1

Noise Type = 1
(Salt&PepperNoise)
Noise Lv = 0.050000
```



例2: ノイズレベルも指定

```
>Tool-11.exe Gray.bmp 1 0.3
argc = 4
argv[0] = Tool-11.exe
argv[1] = Gray.bmp
argv[2] = 1
argv[3] = 0.3

Noise Type = 1
(Salt&PepperNoise)
Noise Lv = 0.300000
```



ごま塩ノイズの場合②(カラー画像の例):

※ (ごま塩ノイズのデフォルトのノイズレベルは 0.05 となる)

●Tool-11.exe 入力ファイル名 1 [0.0～1.0(ノイズレベル)]

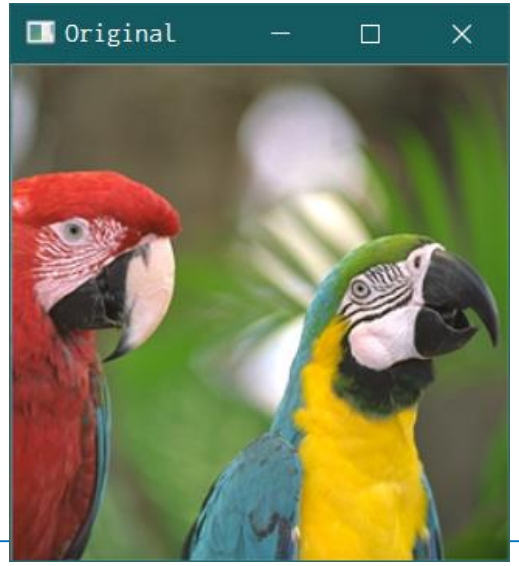
※ファイル名のみ
指定した場合は、
ホワイトノイズになる。

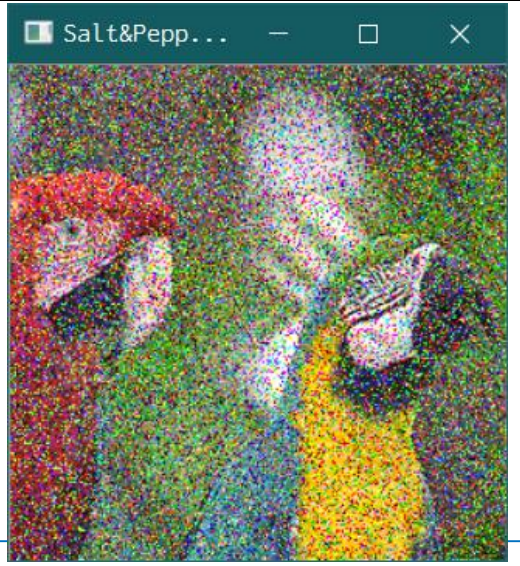
例1: ファイル名とノイズ種類のみ指定

```
>Tool-11.exe Parrots(Color).bmp 1
argc = 3
argv[0] = Tool-11.exe
argv[1] = Parrots(Color).bmp
argv[2] = 1

Noise Type = 1 (Salt&PepperNoise)
Noise Lv = 0.050000
```

例2: ノイズレベルも指定

```
>Tool-11.exe Parrots(Color).bmp 1 0.3
argc = 4
argv[0] = Tool-11.exe
argv[1] = Parrots(Color).bmp
argv[2] = 1
argv[3] = 0.3

Noise Type = 1 (Salt&PepperNoise)
Noise Lv = 0.300000
```





課題 No.11：単純移動平均/ガウシアンフィルタによる平滑化

(1) 以下の2つの関数を完成させる　--- この関数部分のソースコードを載せること

移動平均フィルタの係数を作る

void makeMovingAverageOpe(Operator* ope);

ガウシアンフィルタの係数を作る

void makeGaussianOpe(Operator* ope);

(2) フィルタのサイズ、およびガウシアンフィルタの σ の値を変えて考察

main() 関数内で、

ope.size = 1;

ope.sigma = 1.0;

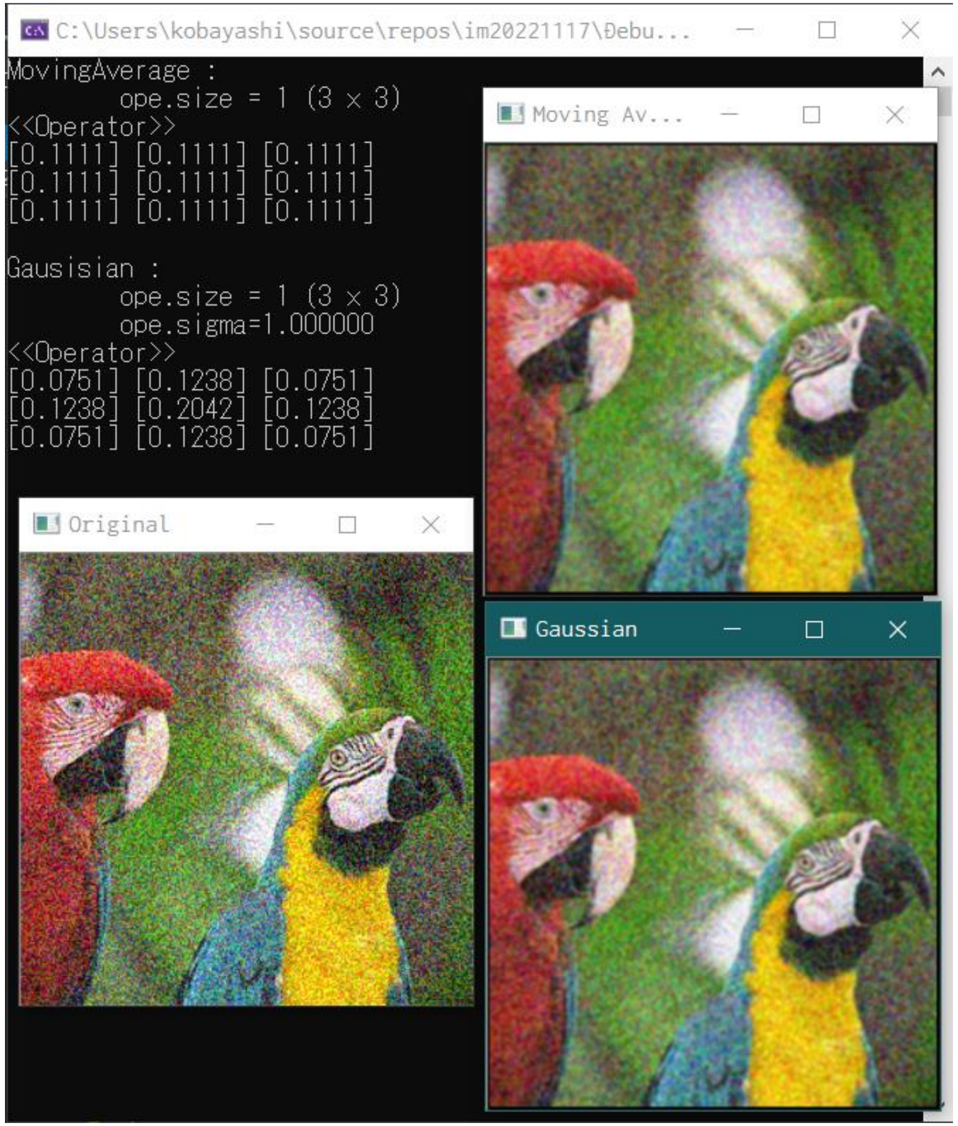
としているパラメータを変化させて、結果を比較する。

--- 設定したパラメータと合わせて、

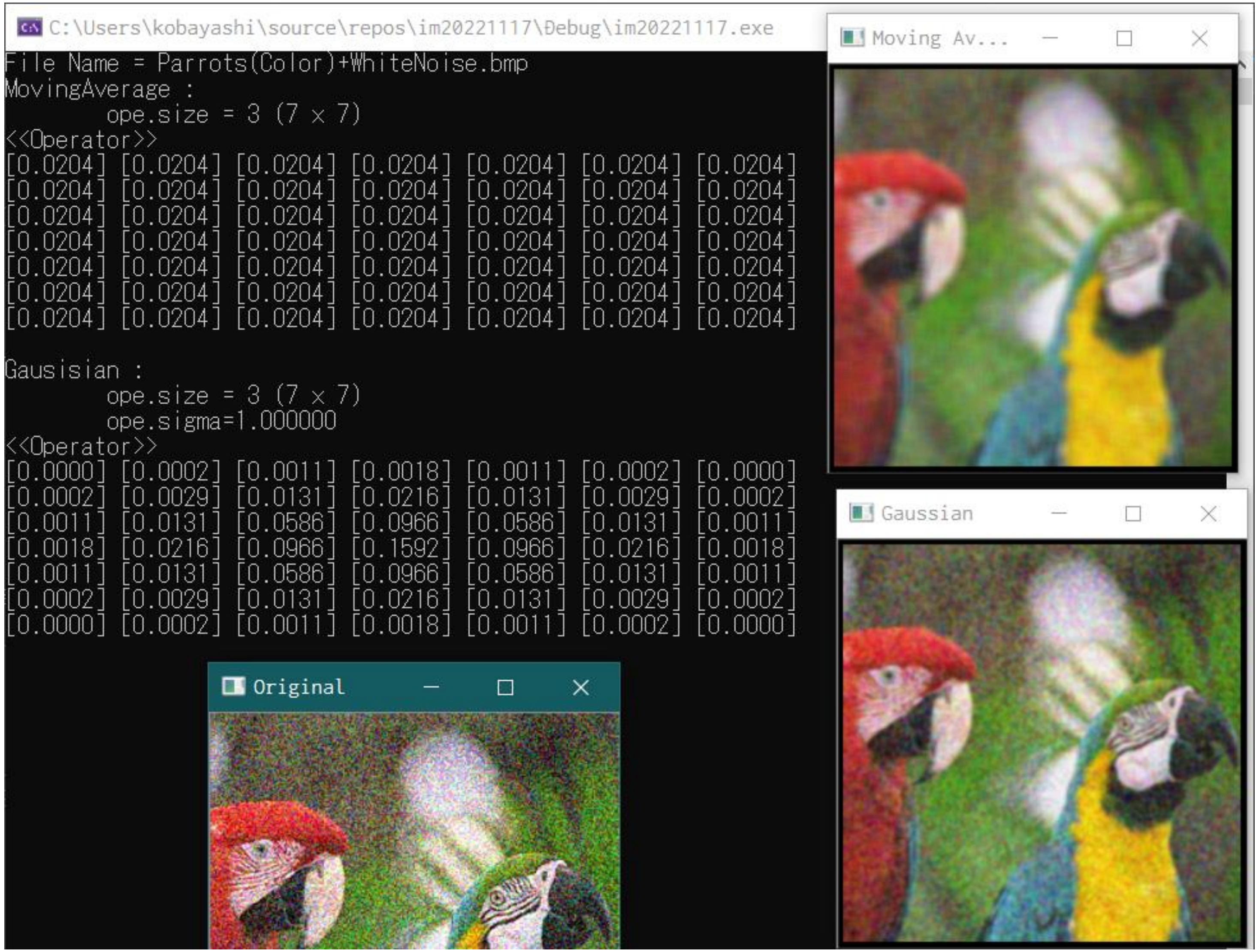
元画像、処理結果の画像も掲載して**考察**すること。

実行例

例1: ope->size = 1, ope->sigma = 1.0

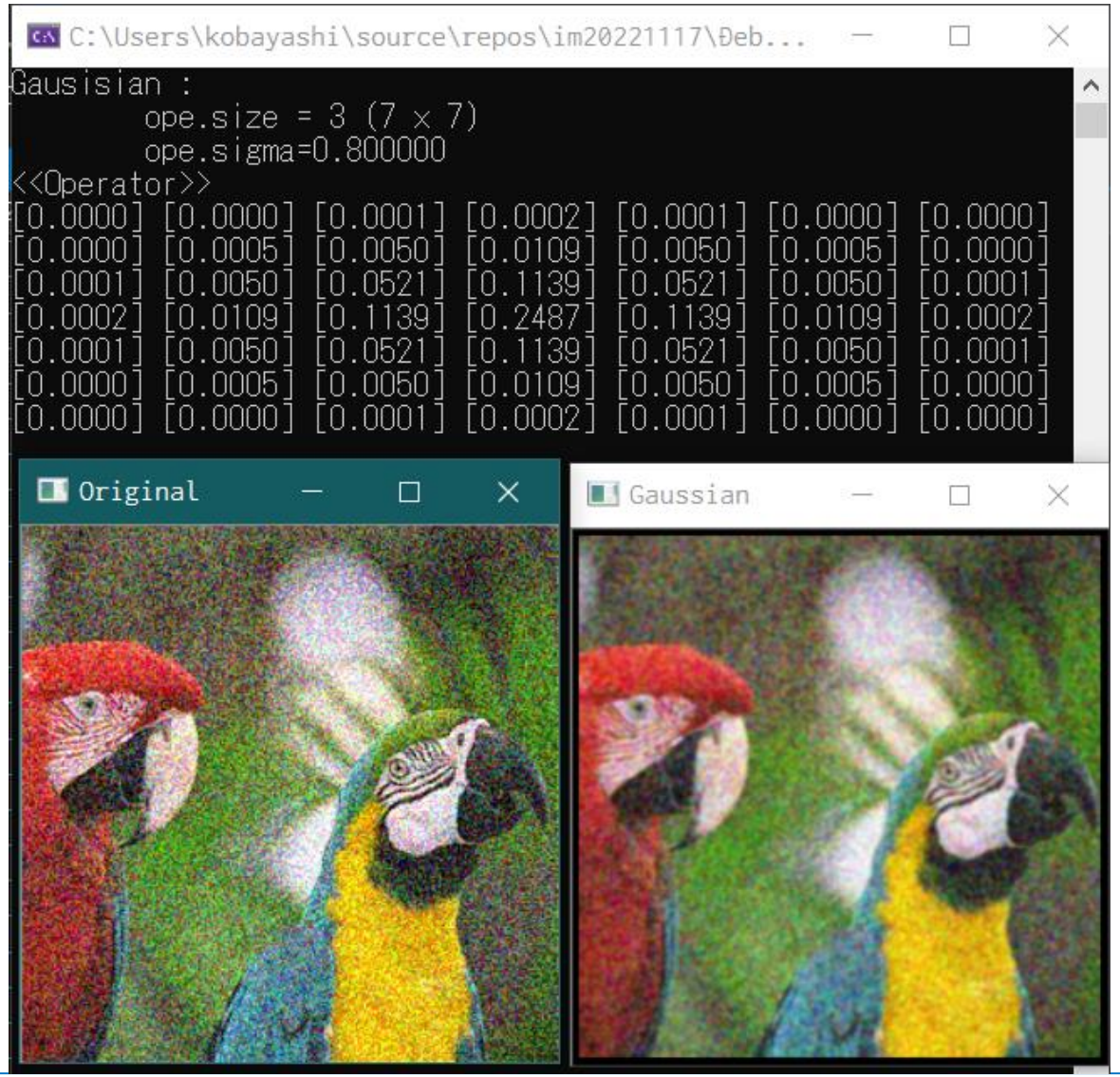


例2: ope->size = 3, ope->sigma = 1.0



実行例 (※Gaussianの結果のみ示す)

例3: ope->size = 3, ope->sigma = 0.8



例4: ope->size = 3, ope->sigma = 3.0