画像情報処理特論　レポート課題

画像処理への確率統計技術の応用

2006-10-17

M06-A04　井上 謙次

大阪工業大学大学院情報科学研究科

1. はじめに

2006 年 8 月 22 日，確率微分方程式の創始をはじめとする確率解析の業績が評価され，伊藤清博士が第1回ガウス賞を受賞された。ガウス賞は数学の応用に対して新設された賞であり，確率微分方程式を解く伊藤解析は工学，物理学，生物学，経済学など幅広い分野で必須の道具として使われている。

近年進歩が著しい学問に脳科学がある。脳科学では脳活動の画像化を行って脳機能の解明を試みるが，被験者に刺激を提示した際のおのおのの脳活動の変化は非常に小さく信頼に欠けるため，施行を何度も繰り返すことによってデータを集め，脳活動モデルの有意性の検定を行う。その際，SPM (Statistical Parametric Mapping) と呼ばれる手法においては，各 3 次元ボクセルの時系列データに対して GLM (General Linear Model) と呼ばれる手法を用いた統計分析が行われる。

より身近な例に目を向けると，我々が日々使っているメールのフィルタリング，特に迷惑メールの除去にベイズ推定が利用されていることはよく知られている。この技術は従来の決定論的なルールベースの手法に比べ，管理に手間がかからず，かつ精度の高いフィルタリングが可能という利点を併せ持つ。

確率モデル化，有意性検定，統計分析，ベイズ推定といった確率統計手法は，今や様々な領域で使用されている。20 世紀の量子論の研究成果が示唆するところによれば，我々の実存そのものが確率的存在であり，決定論的解釈はその一つの近似手法であるのかもしれない。

ならば，確率統計論を画像処理に応用することは至極自然なことであり，それを妨げる要因は何もない。事実，画像処理への確率モデルの適用は 20 年以上の歴史を持つ。

2. ノイズ除去フィルタの確率モデル化

2.1. 課題設定

確率統計技術を応用できそうな画像処理には大雑把に次の 2 種類の分野が考えられる:

1. 画像の加工・変換
2. 画像の判別・カテゴライズ

前者は個々の画像自体に作用するものであって，伝統的な処理としては 2 値化，幾何学的変形，ノイズ除去，エッジ検出などが挙げられる。後者は本質的に異なる画像間の特徴の差異にかかわるものであって，OCR，顔の識別，画像検索などが挙げられる。

後者に関しては元来確率統計的な手法と相性が良く，例えばニューラルネットワークを用いた顔の識別といった技術が広く研究されている。

そこで，本レポートでは前者を取り上げたい。ここでは，具体例としてベイズ推定を用いて確率モデル化したノイズ除去フィルタを扱うことにする。

2.2. 確率モデル化の意味

伝統的なノイズ除去フィルタである移動平均フィルタやメディアンフィルタを概観し，ノイズ除去フィルタを確率モデルとして表す有効性を考えてみよう。

移動平均フィルタは各画素の輝度値を近傍画素の輝度値の平均値で置き換える操作を施す。ただし，逸脱して非常に大きな値を持つ輝度値（ノイズ）がある場合，その影響が周囲に及んでしまう。それに対して，メディアンフィルタは各画素の輝度値を近傍画素の輝度値の中央値で置き換える操作を施すため，逸脱したノイズは周囲にその影響を波及することなく除去される。

どちらのフィルタについても，「注目画素の輝度値を近傍画素の輝度値から推定する」という原則に基づいており，フィルタを適用した画像はその結果平滑化されることになる。ここで，注目画素の（本来の）輝度値を推定するために使われているのが近傍画素から平均値や中央値を求めるというアルゴリズムである。平均値や中央値が本来の値であると考えるのは，経験的にはそれがノイズ除去フィルタとして効率的に働くことが知られているが，やや恣意的であろう。そこで，そのような平均値や中央値も含めて，注目画素の本来の輝度値が取りうる値を複数考え，その中で最も可能性の高い輝度値を採用するというようにアルゴリズムを拡張することでノイズ除去の精度を上げることが考えられる。これがノイズ除去フィルタの確率モデル化の最も基本的なアイデアである。

実際には，「ノイズがどのように加わったのか」という（確率的な）劣化過程を仮定しなければ，上述の確率モデル化のアイデアは用を成さない（解析的に解けばよいだろう）。本レポートでは，単純な劣化過程（2 元対称通信路）を想定し，最も単純な 2 値画像のケースについて，マルコフ確率場およびベイズの公式を用いた確率モデル（マルコフ確率場モデル）として記述し，その近似解法として基本的なアニーリング（ギブス近似）と平均場近似を用いたアルゴリズムについて概観する。

2.3. ベイズ推定を用いた 2 値画像のマルコフ確率場モデル

精微な議論に関しては専門書にゆずり，本節では前節で述べた確率モデルを簡単に述べる。

M×N 画素の 2 次元画像 $A=(a_{xy}), x\in[1,M], y\in[1,N]$ を考える。画像は 2 値画像とし，白の階調値を −1，黒を +1 とする。各画素にそれぞれ確率変数を割り当てる。確率変数の（2 次元）集合を確率場と呼ぶ。原画像の確率場を $\boldsymbol{f}=\{f_{x,y}\}$，劣化過程 $P(\boldsymbol{g}|\boldsymbol{f})$ を経て得られた劣化画像の確率場を $\boldsymbol{g}=\{g_{x,y}\}$ とすると，確率場モデルを用いた画像処理の枠組みはベイズの公式を用いて

$$P(\boldsymbol{f}|\boldsymbol{g})=\frac{P(\boldsymbol{g}|\boldsymbol{f})P(\boldsymbol{f})}{\sum_{\boldsymbol{f}} P(\boldsymbol{g}|\boldsymbol{f})P(\boldsymbol{f})}$$

と記述できる。ここで $\sum_{\boldsymbol{f}} = \sum_{f_{1,1}=\pm1}\sum_{f_{1,2}=\pm1}\cdots\sum_{f_{M,N}=\pm1}$ は確率場が取りうる全ての状態に対する重ね合わせであり，2^{MN} 通りの組み合わせが存在する（2 値画像の場合）。事前確率 $P(\boldsymbol{f})$ は原画像 $\boldsymbol{f}$ の出現確率，劣化過程 $P(\boldsymbol{g}|\boldsymbol{f})$ は原画像 $\boldsymbol{f}$ から劣化画像 $\boldsymbol{g}$ が生成される確率であり，それらを計算することで，劣化画像 $\boldsymbol{g}$ が与えられたという条件のもとでの原画像 $\boldsymbol{f}$ に対する確率 $P(\boldsymbol{f}|\boldsymbol{g})$ が求められる。

原画像に対する事前情報 $P(\boldsymbol{f})$ として，近傍画素の輝度値にのみ依存するマルコフ確率場

$$P(\boldsymbol{f})=\frac{\exp\left(-\frac{1}{2}\alpha\sum_{x=1}^{M}\sum_{y=1}^{N}\left((f_{x,y}-f_{x+1,y})^2(f_{x,y}-f_{x,y+1})^2\right)\right)}{\sum_{\boldsymbol{f}}\exp\left(-\frac{1}{2}\alpha\sum_{x=1}^{M}\sum_{y=1}^{N}\left((f_{x,y}-f_{x+1,y})^2(f_{x,y}-f_{x,y+1})^2\right)\right)}\quad,\quad \alpha>0$$

を仮定する。これは「原画像の各画素の輝度値はその近傍画素の輝度値と同じ値をとる確率が高

い」ことを意味する。

また，劣化過程 $P(\boldsymbol{g}|\boldsymbol{f})$ として各画素ごとに独立に確率 p で-1 と+1 が反転する 2 元対称通信路

$$P(\boldsymbol{g}|\boldsymbol{f})=\frac{\exp\left(-\frac{1}{2}\beta\sum_{x=1}^{M}\sum_{y=1}^{N}(f_{x,y}-g_{x,y})^2\right)}{(1+\exp(-2\beta))^{MN}} \quad , \quad \beta=\frac{1}{2}\ln\left(\frac{1-p}{p}\right)$$

を仮定する。この劣化過程は白色雑音であり，多値画像においては容易に加法的白色ガウス雑音 (AWGN) に拡張できる。

上記の事前確率および劣化過程をベイズの公式に代入することで事後確率 $P(\boldsymbol{f}|\boldsymbol{g})$ はマルコフ確率場モデル

$$P(\boldsymbol{f}|\boldsymbol{g})=\frac{\exp(-E(\boldsymbol{f}|\boldsymbol{g}))}{\sum_{\boldsymbol{f}}\exp(-E(\boldsymbol{f}|\boldsymbol{g}))}$$

$$E(\boldsymbol{f}|\boldsymbol{g})=\frac{1}{2}\beta\sum_{x=1}^{M}\sum_{y=1}^{N}(f_{x,y}-g_{x,y})^2+\frac{1}{2}\alpha\sum_{x=1}^{M}\sum_{y=1}^{N}\left((f_{x,y}-g_{x+1,y})^2+(f_{x,y}-g_{x,y+1})^2\right)$$

として得られる。

2.4. 最適解と近似解法

前節のマルコフ確率場モデルにおける事後確率 $P(\boldsymbol{f}|\boldsymbol{g})$ から，ノイズを除去した修復画像は

$$\hat{\boldsymbol{f}}=arg\max_{\boldsymbol{f}}P(\boldsymbol{f}|\boldsymbol{g})$$

あるいは

$$\hat{\boldsymbol{f}}=arg\min_{\boldsymbol{f}}E(\boldsymbol{f}|\boldsymbol{g})$$

のように求められる。この決定法は最大事後確率推定と呼ばれる。これらの式は最適化問題として捉えることができ，上式の解 $\hat{\boldsymbol{f}}$ を最適解と呼ぶ。

また，事後周辺確率分布 $P_{x,y}(f_{x,y}|\boldsymbol{g})$ を

$$P_{x,y}(\zeta|\boldsymbol{g})=\sum_{\boldsymbol{f}}\delta_{\zeta,f_{x,y}}P(\boldsymbol{f}|\boldsymbol{g}) \quad , \quad \zeta=\pm1$$

$$\delta_{a,b}=\begin{cases}1 & (a=b)\\ 0 & (a\neq b)\end{cases}$$

によって定義すれば，

$$\hat{f}_{x,y}=arg\max_{\boldsymbol{f}}P_{x,y}(f_{x,y}|\boldsymbol{g})$$

により各画素ごとに原画像の推定値を求められる。この決定法は最大事後周辺確率推定と呼ばれる。

さて，具体的に修復画像を求めるアルゴリズムについて考える。まず，確率場 $\boldsymbol{f}$ に対してギ

ブス分布

$$P(\boldsymbol{f}|\boldsymbol{g},T)=\frac{\exp\left(-\frac{1}{T}E(\boldsymbol{f}|\boldsymbol{g})\right)}{\sum_{f}\exp\left(-\frac{1}{T}E(\boldsymbol{f}|\boldsymbol{g})\right)} \quad , \quad T>0$$

を導入する。詳細は省くが，最初はTの値を大きくとり，徐々にTの値を小さくしていくことで，局所最大確率に落ち込みにくくなることが知られている。このような反復法はアニーリングと呼ばれている。

さて，2値画像の場合，修復画像 $\hat{\boldsymbol{f}}=\{\hat{f}_{x,y}\}$ は

$$\hat{f}_{x,y}(T)=sign\left(\sum_{f} f_{x,y}\rho(\boldsymbol{f}|\boldsymbol{g},T)\right)$$

で求めることができる。ここで確率変数 $f_{x,y}$ の期待値 $m_{x,y}(T)$ を

$$m_{x,y}(T)=\sum_{f} f_{x,y}\rho(\boldsymbol{f}|\boldsymbol{g},T)$$

として導入し，確率変数 $f_{x,y}$ と $f_{x+1,y}$ および $f_{x,y}$ と $f_{x,y+1}$ の共分散がほとんど0であり，$(f_{x,y}-m_{x,y}(T))(f_{x+1,y}-m_{x+1,y}(T))\simeq 0$ を満たすことを仮定すると，次の漸化式を導くことができる。

$$m_{x,y}(T)=\tanh\left(\beta g_{x+1,y}+\alpha(m_{x-1,y}(T)+m_{x,y+1}(T)+m_{x,y-1}(T))\right)$$

これを平均場近似と呼ぶ。ギブス近似を用いたアニーリングを行って解くアルゴリズムは次のようになる。

1: $for\ x\in[1,M],\ y\in[1,N]$
2: $\quad m_{x,y}\leftarrow g_{x,y}$
3: $T\leftarrow 1.0+R\Delta T\quad (R\in\mathbb{N})$
4: $while\ T>1.0$
5: $\quad for\ x\in[1,M],\ y\in[1,N]$
6: $\quad\quad a_{x,y}(0)\leftarrow m_{x,y}$
7: $\quad T\leftarrow T-\Delta T$
8: $\quad r\leftarrow 0$
9: $\quad while\ 0\le r<r_{max}\ \ or\ \sum_{x=1}^{M}\sum_{y=1}^{N}|a_{x,y}(r)-a_{x,y}(r-1)|>\varepsilon$
10: $\quad\quad for\ x\in[1,M],\ y\in[1,N]$
11: $\quad\quad\quad a_{x,y}(r+1)\leftarrow\tanh\left(\frac{\beta}{T}g_{x,y}+\frac{\alpha}{T}\left(a_{x+1,y}(r)+a_{x-1,y}(r)+a_{x,y+1}(r)+a_{x,y-1}(r)\right)\right)$
12: $\quad\quad r\leftarrow r+1$
13: $\quad\quad for\ x\in[1,M],\ y\in[1,N]$
14: $\quad\quad\quad m_{x,y}\leftarrow a_{x,y}(r)$

15: $\hat{f}_{x,y} \leftarrow sign(m_{x,y})$

本アルゴリズムを適用した結果を図 1, 図 2 に示す。また, プログラムのソースコードを付録 A に添付する。なお, 通常のラップトップコンピュータで実行したところ, 1 枚の画像に対しておよそ十秒から数十秒程度の処理時間がかかったことを参考のため記しておく。

(a) (b)

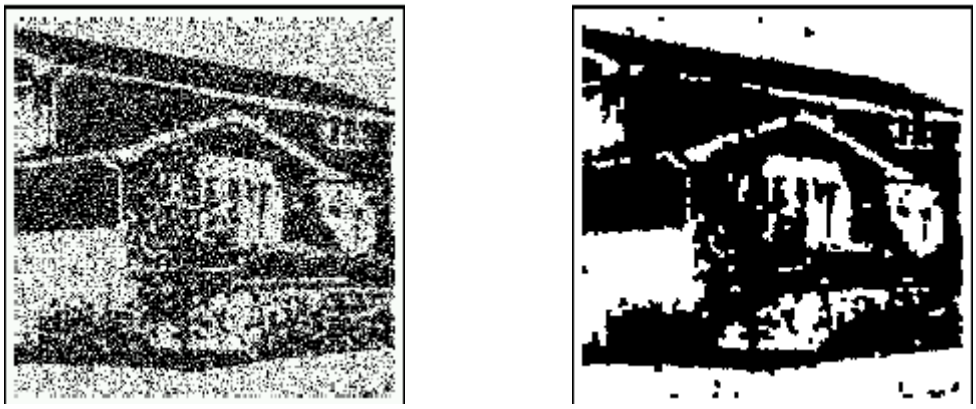

図 1: 2 値画像の修復例 (a) 劣化画像 (p = 0.2) (b) 修復画像 (α = 0.50)

(a) (b)

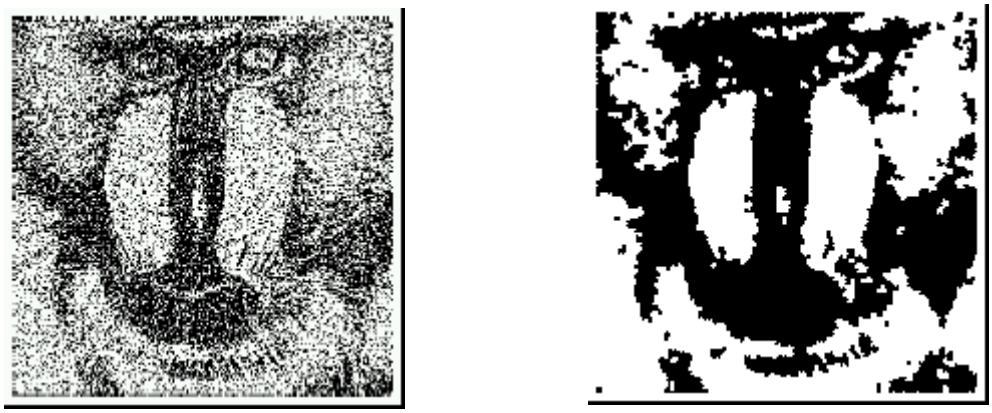

図 2: 2 値画像の修復例 (a) 劣化画像 (p = 0.2) (b) 修復画像 (α = 0.50)

2.5. 課題と展望

本来ならば本手法の分析を行い, 他の手法との比較を含め, 定性的または定量的に本手法の有効性を示すべきであるが, 本レポートにおいては省略させていただきたい。

本レポートでは 2 値画像の確率モデルを扱ったが, モデルを修正すれば多値画像やカラー画

像にも適用可能である。その他，画像修復の確率論的扱いとして様々なモデル化や近似解法が提案されている。

また，マルコフ確率場モデルは画像修復以外にもエッジ検出や領域分割など多岐にわたる画像処理技術への応用が研究されている。

3. おわりに

ノイズ除去フィルタにおける本手法の適用は実質的には計算コストをかけることによって精度の高い（品質の高い）情報処理を行えるという意味合いであったであろう。その意味合いにおいては，本質的にまったく不可能であったことを可能にしたとは言い難い。

しかしながら，本手法におけるような確率統計的な視点をもって世界を眺め，統計確率的に対象をモデル化することは重要であろう。画像処理の分野に限らず，そのような意識を持っておきたいと思う。

参考文献

[1] 田中和之，“確率モデルによる画像処理技術入門―ベイズ統計と確率的画像処理―”，2002

[2] 田中和之，“確率モデルによる画像処理技術入門―統計力学的プログラム構成法（基礎編）―”，2002

付録

A. ノイズ除去フィルタのソースコード（C++）

```
/**
 * Noise filter for binary png images, using Markov random field model and mean field approximation
 * $Id: noisefilter.cpp 1070 2006-10-16 23:57:56Z SYSTEM $
 **/


#include <math.h>
#include <stdio.h>
#include <stdlib.h>


#include "png.h"


#define TH      128     // threshold


#define REPEAT  30      // number of repeat for annealing loop
#define RMAX    200     // max number of repeat for loop of mean field approximation
#define DT      (0.10)  // delta T
#define EPS     (10e-6)
```

```
#define ALPHA   0.5
#define P       0.2

struct PNG {
  FILE *fp;

  png_structp png_ptr;
  png_infop info_ptr;
  unsigned long width, height;
  int bit_depth, color_type, interlace_type;
  int channels;
  unsigned char **image;
} pngim;

void filter_noise_stat(char **f, unsigned long width, unsigned long height);
int sign(double value);
void read_png(const char *filename);
void close_png();
void write_png(const char *filename, PNG *p);
void malloc_image(PNG *p, bool use_rowbytes);
void free_image(PNG *p);
void **malloc_matrix(unsigned long width, unsigned long height, int size);
void free_matrix(void **ptr, unsigned long items);

int main(int argc, char **argv) {
  if (argc < 3) {
    fprintf(stderr, "Usage: %s input_pngfile output_pngfile\n", argv[0]);
    exit(1);
  }

  // read the png file
  read_png(argv[1]);

  unsigned long width = pngim.width;
  unsigned long height = pngim.height;

  // allocate memory for the image to be processed
  char **im = (char**)malloc_matrix(width, height, sizeof(char));

  // binarization
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      int xoffset = x * pngim.channels; // x offset
```

```
      int value = 0;
      for (int c = 0; c < pngim.channels; c++) {
        if (pngim.image[y][xoffset+c] > value) { value = pngim.image[y][xoffset+c]; }
      }
      im[y][x] = (value > TH ? -1 : 1);
    }
  }

  close_png();

  // noise filtering
  filter_noise_stat(im, width, height);

  // writeback the image to png file
  PNG writeback;
  writeback.width = width;
  writeback.height = height;
  malloc_image(&writeback, false);
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      writeback.image[y][x] = (unsigned char)(im[y][x] > 0 ? 0 : 255);
    }
  }

  write_png(argv[2], &writeback);

  free_matrix((void**)im, height); im = NULL;
  return 0;
}

void filter_noise_stat(char **f, unsigned long width, unsigned long height) {
  int signs = 4;
  int sign_x[4] = {+1, -1,  0,  0};
  int sign_y[4] = { 0,  0, +1, -1};

  double **m = (double**)malloc_matrix(width, height, sizeof(double));
  double **g = (double**)malloc_matrix(width, height, sizeof(double));
  double **current = (double**)malloc_matrix(width, height, sizeof(double));
  double **prev = (double**)malloc_matrix(width, height, sizeof(double));

  double alpha = ALPHA;
  double beta = log((1.0 - P) / P) * 0.50;
  printf("alpha=%f, beta=%f\n", alpha, beta);
```

```
for (int y = 0; y < height; y++) {
  for (int x = 0; x < width; x++) {
    g[y][x] = f[y][x]; // copy it, so f can be used for output
    m[y][x] = g[y][x];
  }
}

double t = 1.0 + REPEAT * DT;
while (t > 1.0 + EPS) {
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      prev[y][x] = m[y][x];
    }
  }
  t -= DT;

  int r = 0; // counter
  while (1) {
    for (int y = 0; y < height; y++) {
      for (int x = 0; x < width; x++) {
        double sum = 0.0;
        for (int s = 0; s < signs; s++) {
          int cx = x + sign_x[s];
          if (cx < 0)       { cx += width; }
          if (cx >= width)  { cx -= width; }
          int cy = y + sign_y[s];
          if (cy < 0)       { cy += height; }
          if (cy >= height) { cy -= height; }

          sum += prev[cy][cx];
        }

        current[y][x] = tanh( ( beta * g[y][x] + alpha * sum ) / t );
      }
    }

    double diff = 0.0;
    for (int y = 0; y < height; y++) {
      for (int x = 0; x < width; x++) {
        m[y][x] = current[y][x];
        f[y][x] = sign(m[y][x]);
```

```
          diff += fabs(current[y][x] - prev[y][x]);
          prev[y][x] = current[y][x];
        }
      }

      r++;
      if (diff < EPS || r > RMAX) { break; }
    }
    printf("t=%f, r=%d\n", t, r);
  }

  free_matrix((void**)m, height);
  free_matrix((void**)g, height);
  free_matrix((void**)current, height);
  free_matrix((void**)prev, height);
}

int sign(double value) {
  if (value > EPS) {
    return +1;
  } else if (value < -EPS) {
    return -1;
  }

  return 0;
}

void read_png(const char *filename) {
  pngim.fp = fopen(filename, "rb");
  if (pngim.fp == NULL) {
    perror("Cannot open file");
    exit(1);
  }

  pngim.png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
  pngim.info_ptr = png_create_info_struct(pngim.png_ptr);
  png_init_io(pngim.png_ptr, pngim.fp);

  png_read_info(pngim.png_ptr, pngim.info_ptr);
  png_get_IHDR(pngim.png_ptr, pngim.info_ptr, &(pngim.width), &(pngim.height),
               &(pngim.bit_depth), &(pngim.color_type), &(pngim.interlace_type),
               NULL, NULL);
```

```
  malloc_image(&pngim, true);

  png_read_image(pngim.png_ptr, pngim.image);

  pngim.channels = png_get_channels(pngim.png_ptr, pngim.info_ptr);
}

void close_png() {
  free_image(&pngim);

  png_destroy_read_struct(&(pngim.png_ptr), &(pngim.info_ptr), (png_infopp)NULL);

  fclose(pngim.fp);
}

void write_png(const char *filename, PNG *p) {
  p->fp = fopen(filename, "wb");
  p->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
  p->info_ptr = png_create_info_struct(p->png_ptr);
  png_init_io(p->png_ptr, p->fp);

  png_set_IHDR(p->png_ptr, p->info_ptr, p->width, p->height,
               8, PNG_COLOR_TYPE_GRAY, PNG_INTERLACE_NONE,
               PNG_COMPRESSION_TYPE_DEFAULT, PNG_FILTER_TYPE_DEFAULT);

  png_write_info(p->png_ptr, p->info_ptr);
  png_write_image(p->png_ptr, p->image);
  png_write_end(p->png_ptr, p->info_ptr);

  png_destroy_write_struct(&(p->png_ptr), &(p->info_ptr));
  fclose(p->fp);
}

/**
 * you first need to set width and height before calling this function
 **/
void malloc_image(PNG *p, bool use_rowbytes) {
  if (p == NULL) { return; }

  // allocate memory
  p->image = (png_bytepp)malloc( p->height * sizeof(png_bytep) );
  for (int i = 0; i < p->height; i++) {
    if (use_rowbytes > 0) {
```

```
      p->image[i] = (png_bytep)malloc( png_get_rowbytes(p->png_ptr, p->info_ptr) );
    } else {
      p->image[i] = (png_bytep)malloc( p->width * sizeof(png_byte) );
    }
  }
}

void free_image(PNG *p) {
  if (p == NULL) { return; }

  for (int i = 0; i < p->height; i++) {
    free(p->image[i]);
  }
  free(p->image);
  p->image = NULL;
}

void **malloc_matrix(unsigned long width, unsigned long height, int size) {
  void **image = (void**)malloc(height * sizeof(void*));
  for (int i = 0; i < height; i++) {
    image[i] = (void*)malloc(width * size);
  }

  return image;
}

void free_matrix(void **ptr, unsigned long items) {
  if (ptr == NULL) { return; }

  for (int i = 0; i < items; i++) {
    free(ptr[i]);
  }
  free(ptr);
}
```