

# グラフ波形数値化プログラム本体 (WaveformDigitizer.py) 説明書

## 目次

1. 機能
2. 動作環境
3. 概要
  - 3.1 構造
  - 3.2 プログラムリストの概要
  - 3.3 入出力
    - 3.3.1 入力
    - 3.3.2 出力
4. 使用方法
5. 関数ライブラリ
6. プログラムリスト

## 1. 機能

このプログラム(WaveForm\_Digitizer.py)は画像ファイルの中から、波形数値化に適した画像を選択して数値化を行い、その結果をCSVファイルに保存する

## 2. 動作環境

### 2.1 モジュール

このプログラムを実行するには、以下のモジュールがあらかじめ準備された環境が必要である  
このプログラムは、各モジュール記載のVersionにおいてWindows10、Linux(Ubuntu 18.04)での動作が検証済である  
各モジュールの記載以外のVersionでの動作は未検証であり注意が必要である

モジュール名	Windows 10	Ubuntu 18.04	comment
Python	3.6.8	3.6.10	
Tensorflow	1.13	2.1.0	
Keras	2.2.4	2.3.1	
matplotlib	3.1.0	3.2.0	
Open CV	4.1.0.25	4.2.0.32	4.0以上であること
Pillow	6.1.0	7.0.0	
tesseract OCR	3.05.02	4.0.0	
pyocr	0.7.2	0.7.2	

### 2.2 学習モデル

このプログラムを実行するには、以下の7本の学習済モデルが、機械学習により生成され、プログラムから読み出せるように設定されていることが必要である

デフォルトでmodelフォルダが用意されており、ここに下記モデルが収納されている

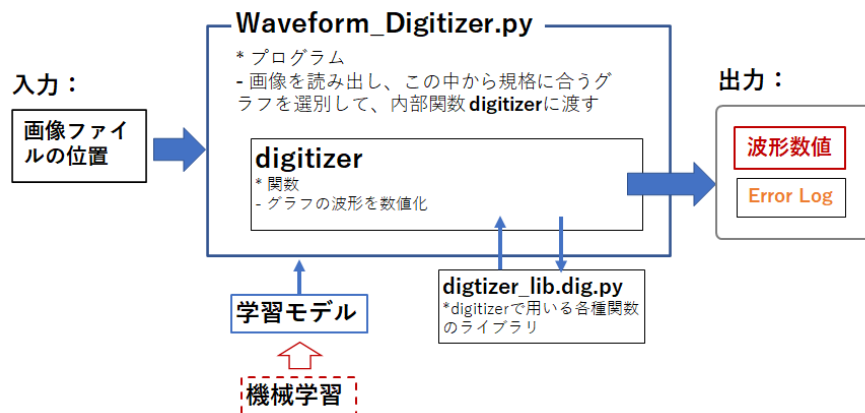
- 画像選択用の学習モデル: FigSelection\_256.h5 <-- (Learning\_FigSelection.pyにより生成)
- 原点位置と軸長読み取り用学習モデル: OrgAxl\_1024.h5<-- (Learning\_OrgAxl.pyにより生成)
- 画像整形用学習モデル: Shaping\_1024.h5<-- (Learning\_FigSelection.pyにより生成)
- 波形本数読み取り用学習モデル: LineNum\_360.h5<-- (Learning\_LineNumber.pyにより生成)
- 1 波形数値化用学習モデル: Digitizer\_1-line\_1024.h5<-- (Learning\_Digitizer\_1-line.pyにより生成)
- 2 波形数値化用学習モデル: Digitizer\_2-lines\_1024.h5<-- (Learning\_FigSelection.pyにより生成)
- 3 波形数値化用学習モデル: Digitizer\_3-lines\_1024.h5<-- (Learning\_FigSelection.pyにより生成)

## 3. 構造

### 3.1 概要

このプログラムは、ユーザーに指定されたディレクトリ下の複数の画像ファイルを検索し、規格に合ったグラフ画像のみを選択する。次に、このプログラムはこの選択されたグラフ画像を自動的に数値化し出力する。すなわち、このプログラムに対してユーザーは画像ファイルを含むディレクトリを指定するのみで、自動的にグラフの数値データを得ることができる

- このプログラムは、1)画像ファイルの読み出しと選別を行うモジュールと、2)1)のモジュールに内包された波形を数値化する関数および、3)波形数値化関数が必要に応じて呼び出す各種関数を収納した関数ライブラリの3つに分けられる
- 画像ファイルの探索は、指定されたディレクトリ下に存在するすべてのサブディレクトリに対して行われる。数値化したファイルは、<画像ファイル名.png.csv>と名付けられ、その画像ファイルの存在した同じディレクトリに収納される
- 画像が数値化できないと判断された場合、および画像が何らかの不具合で読み込めなかった場合、エラーと判断され、logにその旨書き込み、その後の工程は無視し、次の画像ファイルの処理に移行する
- Error logファイルはユーザーに指定されたディレクトリ直下に収められる



### 3.2 プログラムリストの概要

## 1) Waveform\_\_Digitizer.py(プログラム本体)

- 1-20 行: 開始  
3-10 行: 開始コメント  
12-19 行: モジュールのインポート
- 21-306 行: digitizer(グラフを数値化する関数)  
24-28 行: 開始コメント  
30-36 行: 画像ファイルのオープン  
38-52 行: 文字消去、黒領域分離などの前処理  
54-77 行: 原点・軸長の読み取り  
79-159 行: 画像整形処理  
161-204 行: グラフの波形本数読み取り  
206-270 行: 波形の数値化  
272-278 行: 軸数値を読み取りデータ補正  
280-282 行: 数値化の評価  
284-305 行: 数値データの保存
- 307-447 行: 波形数値化の実行  
313-313 行: モジュールのインポート  
318-322 行: 開始処理  
324-341 行: 学習モデルのロード  
343-361 行: 画像の探索と選別  
363-422 行: 画像選別と数値化  
424-445 行: 結果の出力  
447 行: END

## 2) dig.py(関数をまとめたプログラム、ライブラリdigitizer\_libに収納)

- 1-35 行: 開始  
2-19 行: コメント  
21-34 行: モジュールのインポート
- 36-107 行: text\_elim(グラフから不要な文字、数字を消去する関数)
- 109-131 行: gamma\_corrct(画像にガンマ補正を行う関数)
- 133-188 行: erode(画像に浸食・膨張処理を行う関数)
- 190-259 行: rgb\_mxmi(カラーと黒領域を分離する関数)
- 261-508 行: color\_separation(画像の色分解を行う関数)
- 510-619 行: contandline-detect(グラフ内グラフなどの内挿図形を消去する関数)
- 621-728 行: wf\_eval(数値化結果の評価を行う関数)
- 730-921 行: axgain(軸の単位を読み取る関数)
- 923-988 行: linedel(軸などの長い直線を消去する関数)  
990 行: END

## 3.3 入出力

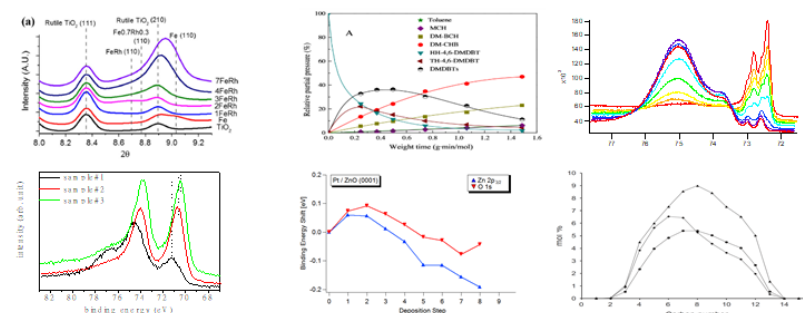
### 3.3.1 入力

#### 画像ファイルの要件

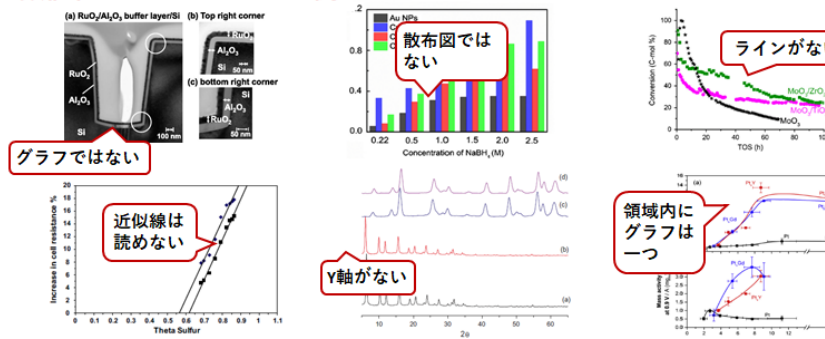
- 画像はpng形式であること（これ以外の形式の画像ファイルは無視される）
- 画像は以下のような要件を満たすグラフ画像であること
  - X,Y軸が明確に書かれ、ラインで結ばれた散布図であること
  - 領域内にはグラフは一つであること

- Y軸はグラフ領域の左端にあること
- マーカー中心とラインは一致していること（近似線ではない）
- マーカーあるいは線種が同一の波形は3本以下であること

#### 数値化できるグラフの例



#### 数値化できないグラフの例



- 画像ファイルは指定されたディレクトリ直下、あるいはそのサブディレクトリのどこかに存在すること  
(例: ディレクトリとして10.1002.polb.24543を指定。下の様ないろいろな種類の混じった文献ファイルでもどこかにpngがあれば検索して数値化できる)

10.1002\_polb.24543+  
 10.1002\_polb.24543\_chemspot.tsv+  
 10.1002\_polb.24543\_figure.tsv+  
 10.1002\_polb.24543\_fulltext.pdf+  
 10.1002\_polb.24543\_fulltext\_pdf.txt+  
 10.1002\_polb.24543\_fulltext\_xml.txt+  
 10.1002\_polb.24543\_jats.xml+  
 10.1002\_polb.24543\_math.xml+  
 10.1002\_polb.24543\_original.xml+  
 10.1002\_polb.24543\_table.xml+  
 1904191705.conversion.done+  
 chem.src.crc+  
 files+  
 polb24543-fig-0001-m.png+  
 polb24543-fig-0001-t.gif+  
 polb24543-fig-0001.png+  
 polb24543-fig-0002-m.png+  
 polb24543-fig-0002-t.gif+  
 polb24543-fig-0002.png+  
 polb24543-fig-0003-m.png+  
 polb24543-fig-0003-t.gif+  
 polb24543-fig-0003.png+

### 3.3.2 出力

以下に実際の出力例をしめす

1) 画像と数値化結果 (画像が収納されているファイルフォルダの例)

f2.png	262.4 kB	PNG 画像
f3.png	446.1 kB	PNG 画像
f4.png	519.5 kB	PNG 画像
f5.png	139.1 kB	PNG 画像
f6.png	219.1 kB	PNG 画像
f7.png	212.7 kB	PNG 画像
f7.png.csv	33.6 kB	CSV ドキュメント
f8.png	172.6 kB	PNG 画像
f8.png.csv	32.6 kB	CSV ドキュメント
f9.png	251.1 kB	PNG 画像
f10.png	224.7 kB	PNG 画像
f10.png.csv	32.5 kB	CSV ドキュメント
f11.png	208.2 kB	PNG 画像
f11.png.csv	28.1 kB	CSV ドキュメント

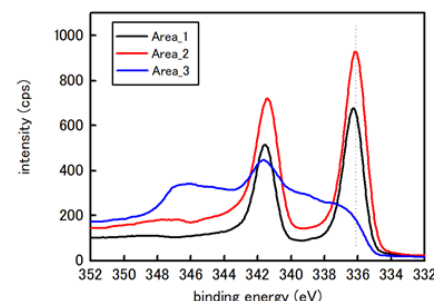
規格に合わない画像として数値化は行われなかった画像

規格に合致して数値化が行われ、画像と同じ名前で数値結果(csvファイル)が収納されている

2)数値化結果の実際例

入力画像名: sample.png --> 出力数値ファイル名: sample.png.scv

入力画像: sample.png



出力数値: sample.png.scv

X軸の値	赤ラインの値	青ラインの値	黒ラインの値
351.93	158.9357	186.4237	120.1383
351.9108	163.803	185.2222	123.093
351.8916	158.4107	184.405	118.9588
351.8724	158.852	185.5357	118.3844
351.8532	158.7503	185.3807	118.3215
351.834	161.4789	184.3323	119.6664
351.8148	161.7395	184.8249	121.6564
351.7956	159.8992	183.4149	119.2461
351.7764	160.2764	186.2757	118.6848
351.7572	158.8313	185.2118	117.2212
351.738	155.0695	183.1426	116.0116
351.7188	158.9155	183.6363	117.5011
.....	.....	.....	.....
332.5396	48.7892	57.783	84.5409
332.5204	49.58	56.9366	83.516
332.5012	49.3698	58.3766	84.8814
332.482	42.9016	58.2561	77.5535
332.4628	43.1881	56.5149	81.3195
332.4436	39.3469	56.9049	77.9555
332.4244	40.0965	55.8472	79.065
332.4052	38.8435	56.2783	79.169
332.386	37.8817	58.124	77.7735
332.3668	35.8225	57.9993	78.7026
332.3476	32.2701	56.6188	76.6425
332.3284	30.8763	56.9382	76.1644
332.3092	26.8191	56.5921	74.0198
332.29	26.7268	57.8797	74.1112

1024点

### 3) Error Log

Error\_log.scvの一部を抜粋して示す

```

2020/06/02/10:33:38 | ===== Digitizer ERROR ===== | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4
2020/06/02/10:33:40 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4998425/temp
2020/06/02/10:33:50 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4998425/temp
2020/06/02/10:33:50 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4998425/temp
2020/06/02/10:33:55 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4998425/temp
2020/06/02/10:33:55 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4998425/temp
2020/06/02/10:34:00 | ===== Digitizer ERROR ===== | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4
2020/06/02/10:34:02 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4998425/temp
2020/06/02/10:34:11 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4982930/temp
2020/06/02/10:34:11 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4982930/temp
2020/06/02/10:34:14 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4982930/temp
2020/06/02/10:34:14 | This file does not fit in my standard | /mnt/external/tkono/Data/nims-articles/test_/0734-2101/2017/10.1116_1.4982930/temp

```

日時

エラー内容

ファイル名 (フルパス)

### エラー内容の説明

This file does not fit in my standard: 画像が規格に合っていない

===== Digitizer ERROR =====: 数値化の途中でエラーが発生して中止した (ファイルが壊れていた場合など)

4. 使用方法

本プログラムの簡単な使用方法を示す

4.1 準備

- プログラム本体（Waveform\_\_Digitizer.py）と7本の学習済モデルを内蔵したmodelフォルダ、および各種関数をまとめたプログラムdig.pyを内蔵したライブラリ(digitizer\_lib)をPythonが実行できる同じ場所に収納する
- 数値化したい画像を含むフォルダ名を確認し、そのフォルダ名を下記の例の様にプログラム本体（Waveform\_\_Digitizer.py）の 347行に入力して保存する

例:

```
# --- Figures Search -----
# type and directory
f_type = '.png'
print('\n Search figure type :',f_type)
directory = 'E:/test'
# directory = '/mnt/exte...les/test'
```

E:/test フォルダの下に画像ファイルがある場合

4.2 実行

- 上記、< 2. 動作環境 > を満たしている環境下のPython上で（Waveform\_\_Digitizer.py）を実行する

例: 仮想環境 tf36 に動作環境が用意されており、C:\Users\XXXXに Waveform\_Digitizer.pyが存在している場合

(tf36) C:\Users\XXXX >python Waveform\_Digitizer.py

以下、プロセスはすべて自動で実行される

5. 関数ライブラリ(dig.py)

ライブラリ(digitizer\_lib)内のdig.pyには9本の関数が収納されている。これらは、グラフ数値化のために定義されたものであるが、関数として独立して使用することもできる。

関数名	機能	パラメータ	返り値
txt_elim	グラフ内の文字を削除する	f: 画像ファイル, alf:削除領域の拡大(pixelで指定), sw: 結果のプロット選択 1:yes, else:no	image: 文字の消去された画像, boxes: 文字を見つけた領域
gamma_corrct	画像のガンマ補正	f: 入力画像, g: ガンマパラメータ	im_gam: 補正後の画像, lookUpTable: 補正用のテーブル

関数名	機能	パラメータ	返り値
erode	画像の浸食/膨張処理	f: 入力画像, sw: 処理の選択 (0:浸食、1:膨張、2:浸食・拡張、3:拡張・浸食), n: 処理繰り返し回数, ngh: 近接領域の定義	out: 出力画像
rgb_mxmi	RBGのMaxとMinの差から、黒とグレイ領域を抽出	f: 入力画像, sl: RGBでのMax-Min のしきい値, v: HSV(RGBでの Max) での value のしきい値	imrgb2: カラー領域の出力画像, imbkgy: 黒灰色領域の出力画像
color_separation	DBSCAN を用いてグラフの色分離	im_: 入力画像, f_na: 入力画像のファイル名, c_dim : リサイズする大きさ, gamma: ガンマ補正の係数, now: 現在時刻, s_sw: データ保存の yes/no 1:yes, else:no	color_line : 色分解された画像データ, 色数
contandline_detect	グラフ内のグラフなどの内挿図形を消去	f: 入力画像, lw: 輪郭を白で上書きする際のラインの幅, dn: 点線を認識する投票数のしきい値, sn: 実線を認識する投票数のしきい値	imrgb: 出力画像
wf_eval	入力画像と予測数値を比較して数値化の信頼性の評価	wo_cont: 入力画像, prd_ol:原点と軸長予測 l_data: 数値データ, ng_bk:NGになった黒ラインの数, col_n: 色数, lin_nu: ライン数, dim: 計算次数	n_zp: 信頼度

関数名	機能	パラメータ	返り値
axgain	軸目盛を読み取り、軸ゲインを推定	ff_n: 入力画像のファイル名, sz: 画像の大きさ, prd_of: 原点位置と軸長の予測値	x0: x軸の始点の値, xe: x軸の終点の値, y0: y軸の始点の値, ye:y軸の終点の値
linedel	軸などの長い直線を消去	f: 入力画像	imgray: 出力画像

## 6. プログラムリスト

全プログラムリストをしめす

### 6.1 プログラム本体 (Waveform\_\_Digitizer.py)

#### 概要

- 1-20 行: 開始  
3-10 行: 開始コメント  
12-19 行: モジュールのインポート
- 21-306 行: digitizer(グラフを数値化する関数)**  
24-28 行: 開始コメント  
30-36 行: 画像ファイルのオープン  
38-52 行: 文字消去、黒領域分離などの前処理  
54-77 行: 原点・軸長の読み取り  
79-159 行: 画像整形処理  
161-204 行: グラフの波形本数読み取り  
206-270 行: 波形の数値化  
272-278 行: 軸数値を読み取りデータ補正  
280-282 行: 数値化の評価  
284-305 行: 数値データの保存
- 307-447 行: 波形数値化の実行**  
313-313 行: モジュールのインポート  
318-322 行: 開始処理  
324-341 行: 学習モデルのロード  
343-361 行: 画像の探索と選別  
363-422 行: 画像選別と数値化  
424-445 行: 結果の出力  
447 行: END

```

1 # -*- coding: utf-8 -*-
2
3 """

```

```

4 Created on Jan.09,2020 / Fixed on Jun.18,2020
5
6 < グラフ波形を数値化するプログラム >
7 / Final version: Jun.18,2020, to Takada-san
8
9 @author: T.KONO
10 """
11
12 import cv2
13 import numpy as np
14 from PIL import Image
15 from datetime import datetime
16 from keras.models import load_model
17 from keras.preprocessing.image import img_to_array
18
19 import digitizer_lib.dig as dg
20
21 # ----- Digitizer Function -----
22 def digitizer(ff_n,dim):
23
24     """< 指定されたグラフ画像の波形を数値化する関数 >
25
26     ff_n : 画像ファイル名
27     dim : dimension
28     """
29
30     # ----- 画像ファイルのOpen -----
31     im_ = Image.open(ff_n)
32     sz = im_.size
33
34     print('\n-----')
35     print('Name:',ff_n)
36     print('iamge size:',sz[0],'x',sz[1])
37
38     # ----- 画像の前処理 -----
39     # 文字の削除
40     wo_txt_im = dg.txt_elim(im_,0.05,0)
41
42     # 内挿図形の削除
43     wo_cont = dg.contandline_detect(wo_txt_im[0],1)
44
45     # 黒・灰色領域と色領域の分離
46     mxmi = 15 # RGB での Max-Min のしきい値
47     mx = 200 # HSV(RGBでの Max) の value のしきい値
48     col_bk = dg.rgb_mxmi(wo_cont, mxmi, mx)
49
50     # 色分離によるラインの抽出
51     line_data = dg.color_separation(col_bk,f_n,0.3,now,0)
52     col_n = line_data['para'][2]
53
54     # ----- 原点と軸長さの読み取り -----
55     print('\n> Orign-position and Axis-length Read ')
56
57     # ガンマ補正
58     g_fac = 0.5
59     g_corr = dg.gamma_corrct(im_.convert('L'),g_fac)
60
61     # リサイズ
62     re_ = cv2.resize(g_corr[0],(dim,dim),cv2.INTER_LANCZOS4)
63
64     # 整形と正規化及び白黒反転

```

```

65 arr_1024 = 1 - np.reshape(re_(1,dim,dim,1))/255
66
67 # 学習モデルによる読み取り
68 prd_ol = orgaxl_model.predict_on_batch(arr_1024)
69 prd_ol = (np.concatenate([prd_ol[0],prd_ol[1]],1))*(dim-1)
70 prd_ol = prd_ol.astype(np.int16)
71
72 print(' Origin point = ', '{:3.0f}'.format(prd_ol[0,0]),
73       ', ', '{:3.0f}'.format(dim-prd_ol[0,1]),
74       )
75 print(' Axis length = ', '{:3.0f}'.format(prd_ol[0,2]),
76       ', ', '{:3.0f}'.format(prd_ol[0,3])
77       )
78
79 # ----- 画像整形 -----
80 print('\n> Shaping of Waveform ')
81
82 # 浸食処理
83 ''' 色のクラスタ数が画像の総ピクセル数の 0.001 以下の場合
84 elode correction を発動する
85 ...
86 if line_data['para'][3] < sz[0]*sz[1]*0.001:
87     en = 1
88 else:
89     en = 0
90
91 # トリミングとリサイズ
92 color_data = []
93 for i in range(0,col_n+1):
94
95     color_line = np.reshape(line_data['data'][i],
96                             (dim,dim,1)
97                             )
98
99     # トリミング
100    im_tr = color_line[(dim-prd_ol[0,1]-prd_ol[0,3]):
101                      (dim-prd_ol[0,1]),
102                      (prd_ol[0,0]):
103                      (prd_ol[0,0]+prd_ol[0,2])]
104
105    # リサイズ
106    im_re = cv2.resize(im_tr,
107                      (dim,dim),
108                      cv2.INTER_LANCZOS4
109                      )
110
111    # 黒ラインのみ直線削除を導入
112    if i == col_n:
113        im_re = dg.linedel(im_re)
114
115    # 軸領域のトリミング
116    '''黒領域の X,Y軸 および枠線の消え残りを消去する
117    外側から dim/50 の領域をすべて白に塗りつぶす
118    ...
119    ddp=int(dim/50)
120    if i== col_n:
121        im_re[0:ddp,0:dim]=255
122        im_re[0:dim,0:ddp]=255
123        im_re[dim-ddp:dim,0:dim]=255
124        im_re[0:dim,dim-ddp:dim]=255
125

```

```

126 array = np.reshape(im_re,(dim,dim,1))
127
128 # 浸食処理
129 it_n = en # 繰り返し回数
130 ngh = 'n4' # 近接領域の取り方 n4 or n8
131 if it_n == 0 :
132     color_data.append(array)
133 else :
134     er_array = np.reshape(dg.erode(array,0,it_n,ngh),
135                           (dim,dim,1)
136                           )
137     color_data.append(er_array)
138
139 # 正規化と白黒反転
140 color_data = (1 - np.asarray(color_data)/255)
141
142 # 学習モデルを呼び出して波形整形
143 '''学習モデルを適用した整形は黒ラインのみに適用する。色ラインに適用すると
144 必要なラインを消してしまうなど副作用が大きすぎるため
145 color_data = shap_model.predict_on_batch(color_data)
146 ...
147 color_data[col_n] = shap_model.predict_on_batch(np.reshape(color_data[col_n],
148                                                             [1,dim,dim,1]))
149
150 # 各ラインの点数を数える
151 lpn = []
152 for ii in range(0,col_n+1):
153     # line point count
154     '''ラインの点数を数え lpn に収納
155     ...
156     ll = (np.reshape(color_data[ii,:],
157                      (dim,-1))).astype(np.uint8)
158     lp_ = np.count_nonzero(ll[:,50:dim-50])
159     lpn.append(lp_)
160
161 # ----- 波形本数の読み取り -----
162 print('\n> Line Number Read ')
163
164 # 波形本数の読み取りのdimensionは360なので各ラインデータをリサイズする
165 col360=[]
166 for j in range(0,col_n+1):
167     c_data= cv2.resize(color_data[j],(360,360),
168                       cv2.INTER_LANCZOS4)
169     col_ar=img_to_array(c_data).astype(np.uint8)
170     col360.append(col_ar)
171
172 # 学習モデルを呼び出して本数を読み取る
173 col360 = np.asarray(col360)
174 predict_num = np.round(ln_model.predict(col360))
175 lin_nu = np.argmax(predict_num, axis=1)
176
177 # 黒ラインの存在判定
178 ''' 黒ラインの点数が色ライン点数平均の 0.2 以下でかつ点数がdim * 2以下
179 の場合は黒ラインはラインと認めない
180 ...
181 if not col_n == 0:
182     print(' Line point number =',lpn)
183     lp_ave = sum(lpn[0:col_n])/col_n
184     # print(lp_ave)
185     if lpn[col_n] <= lp_ave*0.2 and lpn[j] < dim*2:
186         lin_nu[col_n] = 0

```



```

187     print(' * Black-line dose not exist ')
188     ng_bk = lpn[col_n]
189 else:
190     ng_bk = 0
191 else :
192     print (' Black line only. ')
193     ng_bk = 0
194
195 # 色ラインの存在判別
196 """色ラインの点数が すべての色ラインの点数の平均の1/3以下で、
197 かつその点数がdim * 1.5以下のものはラインと認めない
198 """
199 for j in range(0,col_n):
200     if lpn[j]< lp_ave/3 and lpn[j] < dim*1.5:
201         lin_nu[j] = 0
202         print(' * Color '+str(j+1)+' has no enough data ')
203
204 print(' Line Number = ',sum(lin_nu), ', ',lin_nu)
205
206 # ----- 波形の数値化 -----
207 print('\n> Digitalizing of Waveform ')
208
209 l_data = []
210 # line read
211 for j in range(0,col_n+1):
212     data = np.reshape(color_data[j,:,:), (1,dim,dim,1))
213     if lin_nu[j] != 0 :
214         """波形の左右に空白が存在する場合、この空白の x の値を見つける
215         x毎に、y方向に値の総和を計算、この総和が0もしくは十分小さい場合
216         この x 位置を空白と見なす
217         """
218         aa = (color_data[j,:,:,0]*255).astype('uint8')/255
219         y_s = np.zeros([dim])
220         for i in range(0,dim):
221             y_sam = sum(aa[:,i]) # Y方向に総和をとる
222             if y_sam <= 1.:
223                 # 小さなゴミは無視する
224                 y_s[i] = 0
225             else :
226                 y_s[i] = y_sam
227
228         # 左側の空白の終わりのインデックス str をを見つける 0-str
229         srt = np.amin(np.where(y_s != 0))
230         # 右側の空白の始まりのインデックス stp をつける s t p -dim
231         stp = np.amax(np.where(y_s != 0))+1
232
233     else:
234         srt = 0
235         stp = dim
236
237 # 学習モデルを読み出して数値化
238 if lin_nu[j] == 1 :
239     # 1本波形の数値化
240     predict_line = model1.predict_on_batch(data)
241
242     # 空白部分のデータを nan に置換
243     predict_line[0,0:srt] = np.nan
244     predict_line[0,stp:dim] = np.nan
245
246     l_data.append(predict_line)
247

```

```

248 elif lin_nu[j] == 2 :
249     # 2本波形の数値化
250     predict_line = model2.predict_on_batch(data)
251
252     # 空白部分のデータを nan に置換
253     for ii in range(0,lin_nu[j]):
254         predict_line[ii][0,0:srt] = np.nan
255         predict_line[ii][0,stp:dim] = np.nan
256
257     l_data.append(predict_line[ii])
258
259 elif lin_nu[j] == 3 :
260     # 3本波形の数値化
261     predict_line = model3.predict_on_batch(data)
262
263     # 空白部分のデータを nan に置換
264     for ii in range(0,lin_nu[j]):
265         predict_line[ii][0,0:srt] = np.nan
266         predict_line[ii][0,stp:dim] = np.nan
267
268     l_data.append(predict_line[ii])
269
270 l_data = (np.asarray(l_data)).T
271
272 # ----- X,Y軸の範囲を読み取りデータを補正する-----
273 # 軸数値の読み取り
274 gain = dg.axgain(ff_n,sz,prd_ol)
275 # データ補正
276 xd = np.linspace(gain[0],gain[1],dim)
277 yg = abs(gain[3]-gain[2])/1
278 y0 = gain[2]
279
280 # ----- 数値化結果の評価-----
281 relblty= dg.wf_eval(wo_cont,prd_ol,l_data,ng_bk,col_n,lin_nu,dim)
282 print(' Reliability = ', '{:3.2f}'.format((relblty)[0]))
283
284 # ----- 結果をcsvファイルとして保存 -----
285 print('\n> Data Save ')
286 val_save = 'y'
287 """if relblty[0] < 0.4 :
288     print(' * Reliability is wrong.')
289     val_save = 'n'
290 # val_s = input(' Save data to csv file? y or n --> ')
291 """
292 if val_save == 'y' :
293     ydata = (1-l_data[0:dim,0,0:sum(lin_nu)])*yg+y0
294     xydata = np.insert(ydata,0,xd,axis=1)
295     np.savetxt(ff_n+'.csv',
296               xydata,
297               delimiter=',',
298               fmt='%0.4f',
299               )
300     print(' The waveform datas were saved ')
301
302 else:
303     print(' The waveform data were NOT saved ')
304
305 return
306
307 # -----
308 #

```

```

309 # Execution of Digitizing
310 #
311 # -----
312
313 import os,time
314
315 '''画像ファイルの選別を行い、規格に合致した場合は digitizer を呼び出し数値化を行う
316 '''
317
318 # ---- Opening -----
319 now = datetime.today().strftime('%Y/%m/%d/%H:%M:%S')
320 s_t = time.time()
321 print("\n< Run: 1024x1024 Waveform Digitizer ver-α_',now,'>')
322 dim = 1024
323
324 # ---- 学習モデルの呼び出し -----
325 # 学習モデル名
326 orgaxl = './model/OrgAxl_1024.h5'
327 shapng = './model/Shaping_1024.h5'
328 linnum = './model/LineNum_360.h5'
329 line_1 = './model/Digitizer_1-lin_1024.h5'
330 line_2 = './model/Digitizer_2-lines_1024.h5'
331 line_3 = './model/Digitizer_3-lines_1024.h5'
332 fig_sel = './model/FigSelection_256.h5'
333
334 # 学習モデルのロード
335 orgaxl_model = load_model(orgaxl)
336 shap_model = load_model(shapng)
337 ln_model = load_model(linnum)
338 model1 = load_model(line_1)
339 model2 = load_model(line_2)
340 model3 = load_model(line_3)
341 fig_sel_model = load_model(fig_sel)
342
343 # ---- 画像検索 -----
344 # 画像の種類とディレクトリの指定
345 f_type = '.png'
346 print('\n Search figure type :',f_type)
347 directory = 'E:/test'
348
349 print('\n<',directory,'> Serch Start ...')
350 loop = 0
351 list=[]
352
353 # 種類の一致する画像の検索
354 for root, dirs, files in os.walk(directory,topdown = False):
355     for name in files:
356         if name.endswith(f_type) :
357             list.append([root,name])
358             loop = loop+1
359
360 # 画像リストの作成
361 out_list = np.asarray(list)
362
363 # ---- 画像を選別して数値化-----
364 t_n = len(out_list)
365 ng_r = 0
366 ng_f = 0
367 error = []
368 ok_list = []
369

```

```

370 for n in range (0,t_n):
371     # 画像ファイル名
372     f_n = out_list[n][1]
373     ff_n = out_list[n][0]+'/' +out_list[n][1]
374     try:
375         '''fileが開けない場合でも、処理を続行
376         '''
377         # 画像ファイルのOpen
378         gray = cv2.imread(ff_n,0) # Gray image
379         sz = gray.size
380
381         # リサイズ
382         re_256 = cv2.resize(gray,(256,256),cv2.INTER_LANCZOS4)
383         # 画像整形と正規化及び白黒反転
384         arr_256 = 1 - np.reshape(re_256,(1,256,256,1))/255
385         # 学習モデルを呼び出し画像を選別
386         p_n_ = fig_sel_model.predict_on_batch(arr_256)
387         p_n = np.argmax(np.round(p_n_,0))
388     except:
389         pass
390         p_n = 9
391
392     now_now = datetime.today().strftime('%Y/%m/%d/%H:%M:%S')
393
394     # 画像数値化
395     '''p_n= 1: 規格に合った画像として数値化を行う
396     p_n= 0: 規格外の画像として数値化を行わない
397     p_n= 9: ファイルが読めなかった, 数値化を行わない
398     '''
399     if p_n == 1:
400         try:
401             '''内部でerrorが発生した場合、error logを発行して処理を中止し、次の画像へ進む
402             '''
403             # 画像数値化
404             digitizer(ff_n, 1024)
405             ok_list.append([ff_n])
406
407         except:
408             pass
409             # Error log
410             log_ = str(now_now)+' | ===== Digitizer ERROR ===== |
'+str(ff_n)
411             error.append([log_])
412             ng_r = ng_r+1
413
414         elif p_n == 9:
415             log_ = str(now_now)+' | ===== File Read ERROR ===== | '+str(ff_n)
416             error.append([log_])
417             ng_r = ng_r+1
418
419         else:
420             log_ = str(now_now)+' | This file does not fit in my standard | '+str(ff_n)
421             error.append([log_])
422             ng_f = ng_f +1
423
424 # ---- 結果のまとめ -----
425 # エラーをエラーログファイルにまとめる
426 import csv
427 with open(directory+'/Error_log.csv','w') as f:
428     writer =csv.writer(f,lineterminator='\n')
429     writer.writerow(error)

```



```

430 # OK ファイルのリストをファイルにまとめる
431 with open(directory+'\\OK_list.csv','w') as f:
432     writer = csv.writer(f,lineterminator='\\n')
433     writer.writerows(ok_list)
434
435 # 実行時間
436 print('\\n-----')
437 e_t = time.time()
438 print('\\n ... Finish. Exe.Time = ','{:6.2f}'.format(e_t-s_t),'s')
439
440 # 結果の概要出力
441 print('\\n PNG file Number: ',t_n,
442       '\\n OK:',len(ok_list),
443       '\\n Out of standard:',ng_f,
444       '\\n Error:',ng_r
445       )
446
447 # _____ END of Code _____

```

## 6.2 dig.py

### 概要

- 1-35 行: 開始  
2-19 行: コメント  
21-34 行: モジュールのインポート
- 36-107 行: text\_elim(グラフから不要な文字、数字を消去する関数)
- 109-131 行: gamma\_corrct(画像にガンマ補正を行う関数)
- 133-188 行: erode(画像に浸食・膨張処理を行う関数)
- 190-259 行: rgb\_mxmi(カラーと黒領域を分離する関数)
- 261-508 行: color\_separation(画像の色分解を行う関数)
- 510-619 行: contandline-detect(グラフ内グラフなどの内挿図形を消去する関数)
- 621-728 行: wf\_eval(数値化結果の評価を行う関数)
- 730-921 行: axgain(軸の単位を読み取る関数)
- 923-988 行: linedel(軸などの長い直線を消去する関数)  
990 行: END

```

1 # -*- coding: utf-8 -*-
2 """"
3 Created on Jan.15,2020 / Fixed on Jun.18,2020
4
5 < Waveform Digitizerで呼び出される各種関数 >
6 / Final version: Jun.18,2020, to Takada-san
7
8 Text Delete Function
9 Gamma Correction Function
10 Erosion Correction Function
11 Color/Black Separation Function
12 Color Separation Function
13 Inner Figure Delete Function
14 Dizitization Evaluate Function
15 Axis Gain Read Function
16 Line Deleat Function
17
18 @author: T.KONO
19 """"

```

```

20
21 import cv2
22 import numpy as np
23 from PIL import ImageDraw,Image
24 import matplotlib
25 matplotlib.use('Agg')
26 from matplotlib import pyplot as plt
27 import pickle
28 import pyocr, pyocr.builders
29 import sys,re
30
31 from tesseract import PyTessBaseAPI
32 from tesseract import RIL
33
34 from keras.preprocessing.image import array_to_img
35
36 # ----- Text Delete Function -----
37 def txt_elim(f, alf, sw):
38     ""
39     < tesseractを用いて、グラフ内のテキストを消去する関数 >
40
41     入力 f : イメージファイル
42         alf : 領域を拡張する大きさ
43         sw : プロットするかどうかのスイッチ 1:yes, else:no
44
45     出力 image : テキスト消去後のメーじファイル
46         boxes : テキストの位置
47
48     psm option:
49     0 テキストの傾斜角度や言語の種類を検知 (OSD) して出力
50     1 OSDありでOCR (回転した画像にも対応してOCR可)
51     2 OSDなしでテキストの傾斜角度情報を標準出力 (OCRなし)
52     3 OSDなしでOCR (デフォルトの設定はこれ)
53     4 単一列にさまざまなテキストサイズが入り混じったものと想定してOCR
54     5 縦書きのまとまった文章と想定してOCR
55     6 横書きのまとまった文章と想定してOCR
56     7 一行の文章と想定してOCR
57     8 一単語と想定してOCR
58     9 円の中に一単語がある想定でOCR (①、②など)
59     10 一文字と想定してOCR
60     11 順序を気にせずできるだけ画像内に含まれる文章をOCRで取得
61     12 OSDありでできるだけ画像内に含まれる文章をOCRで取得
62     13 Tesseract固有の処理を飛ばして一行の文章としてOCR処理
63     ""
64
65     image = f
66     sz = image.size
67     draw = ImageDraw.Draw(image)
68     imag_or = image.copy()
69
70     with PyTessBaseAPI(lang='eng',psm=11) as api:
71         api.SetImage(image)
72         boxes = api.GetComponentImages(RIL.WORD,True)
73         # Option: RIL.WORD, RIL.TEXTLIN, RIL.SYMBOL
74         for i, (im, box, _ ) in enumerate(boxes):
75             if box['w']*box['h'] < sz[0]*sz[1]*0.01 :
76                 ""検出領域の面積が全pixelの 0.01 より大きな場合は
77                 波形への影響が懸念されるため採用しない
78                 ""
79                 draw.rectangle([box['x']-alf,
80                                box['y']-alf,

```

```

81         box['x']+box['w']+alf,
82         box['y']+box['h']+alf],
83         fill = 'white',
84         outline = 'white'
85     )
86
87 if sw == 1 :
88     fig = plt.figure(figsize=(7,3.5))
89
90     ax = plt.subplot(1,2,1)
91     plt.imshow(imag_or)
92     plt.title('< Original >')
93
94     ax.get_xaxis().set_visible(False)
95     ax.get_yaxis().set_visible(False)
96
97     ax = fig.add_subplot(1,2,2)
98     ax.imshow(image)
99     ax.set_title('< After Elimination >')
100
101     ax.get_xaxis().set_visible(False)
102     ax.get_yaxis().set_visible(False)
103
104     plt.show()
105     plt.close('all')
106
107 return (image, boxes)
108
109 # ----- Gamma Correction Function -----
110 def gamma_corrct(f, g):
111     """
112     < Gamma変換による輝度の修正する関数 >
113
114     入力 f : イメージデータ,
115           g : gamma parameter
116
117     出力 im_gam : アレイデータ,
118           lookUpTable : look up table
119     """
120
121     image = np.asarray(f)
122
123     # lookUpTableの作成
124     lookUpTable = np.zeros((256, 1), dtype = 'uint8')
125     for i in range(256):
126         lookUpTable[i][0] = 255 * pow(float(i) / 255, 1.0 / g)
127
128     # ガンマ補正
129     im_gam = cv2.LUT(image, lookUpTable)
130
131     return (im_gam, lookUpTable)
132
133 # ----- Erosion Correction Function -----
134 def erode(f,sw,n,ngh):
135     """
136     < erosion (浸食/膨張) 処理を行う関数 >
137     Mar.27,2019 : 処理の選択を追加
138     入力
139     f : アレイデータ,
140     sw : 処理の選択 (0:浸食、1:膨張、2:浸食・拡張、3:拡張・浸食)
141     n : 処理繰り返し回数

```

```

142     ngh : 近接領域の定義
143     出力
144     out : アレイデータ
145     """"
146
147     # 4-点近接
148     neighbor4 = np.array([[0, 1, 0],
149                           [1, 1, 1],
150                           [0, 1, 0]],
151                           np.uint8)
152
153     # 8-点近接
154     neighbor8 = np.array([[1, 1, 1],
155                           [1, 1, 1],
156                           [1, 1, 1]],
157                           np.uint8)
158
159     if ngh == 'n4':
160         nghb = neighbor4
161     else:
162         nghb = neighbor8
163
164     # 補正
165     if sw == 0 :
166         out = cv2.erode(f,
167                         nghb,
168                         iterations = n
169                         )
170     elif sw == 1 :
171         out = cv2.dilate(f,
172                         nghb,
173                         iterations = n
174                         )
175
176     elif sw == 2 :
177         out = cv2.morphologyEx(f,
178                               cv2.MORPH_CLOSE,
179                               nghb
180                               )
181
182     elif sw == 3 :
183         out = cv2.morphologyEx(f,
184                               cv2.MORPH_OPEN,
185                               nghb
186                               )
187
188     return(out)
189
190 # ----- Color/Black Separation Function -----
191 def rgb_mxmi(f, sl, v):
192     """
193     < RGBのMaxとMinの差から、黒とグレイ領域を抽出する関数 >
194     Mar.28,2019 : 2nd tex elimination ON/OFF追加
195
196     入力
197     f : RGB イメージデータ
198     sl : RGB での Max-Min のしきい値
199     v : HSV(RGBでの Max) での value のしきい値
200
201     出力
202     imrgb2 :RGB カラー領域のアレイデータ

```

```

203     imbkgy :RGB 黒灰色領域のアレイデータ
204
205 Memo
206     RGBでの各値の最大値をMax、最小値をMinとすると
207     HSVでは S=(Max-Min)/Max, V=Maxとなる
208     ""
209
210     img = np.asarray(f)
211     if len(img.shape) == 2:
212         img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
213
214     # RGB to HSV
215     imhsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
216     bkggy = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
217
218     # graph size
219     sz = imhsv.shape
220
221     for i in range(0, sz[0]):
222         for j in range(0, sz[1]):
223             sa = float(imhsv[i,j][1]) # saturation
224             va = float(imhsv[i,j][2]) # value
225             # mx_mi はRGBでのMax-Min
226             if sa == 0 or va == 0:
227                 mx_mi = 0
228             else:
229                 mx_mi = (sa*va/255)
230
231             if mx_mi <= sl and va < v:
232                 # 白に変換
233                 imhsv[i,j] = [0,0,255]
234                 bkggy[i,j] = [0,0,0]
235
236             else:
237                 # 白に変換
238                 bkggy[i,j] = [0,0,255]
239
240     # HSV to RGB
241     col = cv2.cvtColor(imhsv, cv2.COLOR_HSV2RGB)
242     blk = cv2.cvtColor(bkggy, cv2.COLOR_HSV2RGB)
243
244     # 2nd tex elimination ON/OFF. ON=0,OFF=else
245     sw = 99
246     if sw == 0 :
247         # 2nd text elimination
248         wo_txt_col = txt_elim(array_to_img(col),3,0)
249         wo_txt_blk = txt_elim(array_to_img(blk),3,0)
250
251         # image to array
252         imrgb2 = np.asarray(wo_txt_col[0])
253         imbkgy = np.asarray(wo_txt_blk[0])
254
255     else :
256         imrgb2 = col
257         imbkgy = blk
258
259     return imrgb2, imbkgy
260
261 # ----- Color Separation Function -----
262 def color_separation(im_, f_na,gamma, now, s_sw):
263     ""

```

```

264 < DBSCAN を用いてグラフの色分離をおこなう関数 >
265
266 入力
267     im_ : イメージ
268     f_na : 入力イメージのファイル名
269     c_dim : リサイズする大きさ
270     gamma : ガンマ補正の係数
271     now: : 現在時刻
272     s_sw: : データ保存の yes/no 1:yes, else:no
273
274 出力
275     color_line : 色分解されたアレイデータ、本数などの情報を含む
276
277     * DBSCANの計算できる数は10000個に制限がある
278     色分けの行程でのdimは360に固定する
279     ""
280
281     dim = 360
282     c_dim = 1024
283
284     # --- クラスタリングのための前処理 -----
285     # ガンマ補正
286     g_corr = gamma_corrct(im_[0], gamma)
287
288     # Resize and 2HSV for Line Separation
289     image_c_dim = cv2.resize(g_corr[0],
290                             (c_dim, c_dim),
291                             cv2.INTER_LANCZOS4
292                             )
293     # RGB --> HSV
294     hsv_image_c_dim = cv2.cvtColor(image_c_dim,
295                                   cv2.COLOR_RGB2HSV)
296
297     # Resize and 2HSV for Color Clustering
298     image = cv2.resize(g_corr[0],
299                       (dim, dim),
300                       cv2.INTER_LANCZOS4
301                       )
302
303     ""モノクロ画像のエラーを避けるため右下隅に緑の小さな円を描く
304     ""
305     cv2.circle(image,
306                (dim-1, dim-1),
307                10,
308                (0,255,255)
309                )
310
311     # RGB --> HSV
312     hsv_image = cv2.cvtColor(image,
313                              cv2.COLOR_RGB2HSV)
314
315     # S,Vの制限
316     ""S (彩度) の小さな色(あいまいな色) を白に変換する
317     V(明度) の小さな色(黒に近い色) を黒に変換する
318     ""
319     s_lim = 30
320     for i in range(0,dim-1):
321         for j in range(0,dim-1):
322             # S-V 空間で左斜め下を制限
323             lim = 255-hsv_image[i,j,1]
324

```

```

325 # s がs_lim以下とS-V 空間で左斜め下の色をすべて白に置換
326 if hsv_image[i,j,2] <= lim or \
327     0 < hsv_image[i,j,1] <= s_lim:
328     hsv_image[i,j,:] = [0,0,255]
329
330 ''' h が179を 0 に。本来円筒座標でつながっているはずなので便宜的につなげる。
331 h =0の赤の救済'''
332 '''if hsv_image[i,j,0] >= 178:
333     hsv_image[i,j,0] = 0
334     hsv_orig[i,j,0] = 0'''
335
336 # h,s,v color split
337 h_im, s_im, v_im = cv2.split(hsv_image)
338
339 # nolmarization
340 h_nol = np.reshape(h_im,(-1,1))/179
341 s_nol = np.reshape(s_im,(-1,1))/255
342 v_nol = np.reshape(v_im,(-1,1))/255
343
344 # --- 計算点数を削減するための処理 -----
345 '''S=0 を排除する：
346 DBSCAN では計算点が 10000 個以内であることが推奨されている
347 グラフではほとんどのピクセルが白
348 白領域を排除することで計算点数を大幅に減らすことができる
349 S=0 は白、灰色、黒領域を含み色味は持たない。
350 S=0 を排除することで色味を持つピクセルを分離できる
351 灰色・黒領域は別プロセスで分離するからここで排除しても問題ない
352 '''
353
354 # index of nonzero
355 n_zero_ix = np.nonzero(s_nol)
356
357 # h,s,v non zero index
358 h_nol_nz = h_nol[n_zero_ix[0]]
359 s_nol_nz = s_nol[n_zero_ix[0]]
360 v_nol_nz = v_nol[n_zero_ix[0]]
361
362 # --- DBSCANを用いた色分離 -----
363 # clustering data create
364 Z = np.c_[h_nol_nz, v_nol_nz] # h-v space
365
366 if len(Z)>10000:
367     '''DBSCANは10000個以上の計算を実行するとメモリオーバーを起こす場合がある。
368     10000個以上の画像は対象外として中止する
369     '''
370     sys.exit(1)
371
372 # 色空間でのクラスタリングを実行
373 from sklearn.cluster import DBSCAN
374 dbs = DBSCAN(eps = 0.02,
375             min_samples = 70,
376             n_jobs = -1
377             ).fit(Z)
378
379 # clustering of all element
380 y_dbscan = dbs.labels_
381 cls_n = np.max(y_dbscan)+1
382
383 # value of component(element without noise)
384 z_c = dbs.components_
385

```

```

386 # indx of component
387 i_c = dbs.core_sample_indices_
388
389 # cluster number of component
390 c_n = []
391 for i in i_c:
392     c_n.append(y_dbscan[i])
393 c_n = np.array(c_n)
394
395 # 色分離結果の信頼性
396 c_rep = np.zeros([0,6])
397 for j in range(0,cls_n):
398     dammy_ = z_c[np.where(c_n==j),:]
399     nn = len(dammy_[0])
400     # Heu
401     dh_max = np.max(dammy_[0,:;0])
402     dh_min = np.min(dammy_[0,:;0])
403     dh_mean = np.mean(dammy_[0,:;0])
404     # Value
405     dv_max = np.max(dammy_[0,:;1])
406     dv_min = np.min(dammy_[0,:;1])
407     dv_mean = np.mean(dammy_[0,:;1])
408
409     if nn > len(Z)*0.025 :
410         # クラスタに含まれる点の数が点の総数の 0.025 より小さなものは色として採用し
411         # ない
412         c_v = [dh_mean,
413               dh_max,
414               dh_min,
415               dv_mean,
416               dv_max,
417               dv_min
418               ]
419         c_rep = np.append(c_rep,
420                         np.array([c_v[:]]),
421                         axis=0
422                         )
423     else :
424
425         '''print('{:3.0f}'.format(j),'|','{:4.0f}'.format(nn),
426             '*** NG, Because data number are too little'
427             )'''
428
429 cls_n_pass=len(c_rep)
430
431 # --- クラスタリングされた色によりラインを分離する -----
432 line_data = []
433 line_image = []
434 for l in range(0,cls_n_pass):
435     # upper lower range of s, s=50~255 fix
436     # V-range adjustment
437     if 0.2 <= c_rep[l][3] <= 0.8 :
438         dcl = 0.3
439         dcu = 0.2
440
441     else :
442         dcl = 0
443         dcu = 0
444
445     lower_ = np.array([c_rep[l][2]*179,

```

```

446         80,
447         (c_rep[l][5]-dcl)*255]
448     )
449     upper_ = np.array([c_rep[l][1]*179,
450         255,
451         (c_rep[l][4]+dcu)*255]
452     )
453
454     # ラインの分離
455     line_image = 255 - cv2.inRange(hsv_image_c_dim,
456         lower_,
457         upper_
458     )
459
460     # append array
461     line_data.append(line_image)
462
463     # --- 黒ラインの処理 -----
464     bkgy = cv2.cvtColor(im_[1], cv2.COLOR_BGR2GRAY)
465     bklin = cv2.resize(bkgy,
466         (c_dim, c_dim),
467         cv2.INTER_LANCZOS4
468     )
469
470     line_data.append(bklin)
471
472     prt_sw = 0
473     if prt_sw == 1:
474         # --- ラインイメージの確認 -----
475         ax = plt.figure(figsize=(10,3.5*((cls_n_pass)/3+1)))
476         for ii in range(0,cls_n_pass+1):
477
478             ax = plt.subplot((cls_n_pass)/3+1,3,ii+1)
479             plt.imshow(line_data[ii],cmap='gray')
480             # plt.gray()
481             if ii == cls_n_pass:
482                 plt.title("\n Black-Gray')
483             else:
484                 plt.title("\n Color =' +str(ii+1))
485             ax.get_xaxis().set_visible(False)
486             ax.get_yaxis().set_visible(False)
487
488             plt.show()
489             plt.close('all')
490
491     # --- ラインデータの保存 -----
492     color_line = {'data':line_data,
493         'para':(f_na,
494             c_dim,
495             cls_n_pass,
496             len(Z)
497         )
498     }
499     if s_sw == 1:
500         s_name = 'color_line_'+f_na+'_'+str(dim)+'_'+str(now)+'.pik'
501         with open(s_name,'wb') as f1:
502             pickle.dump(color_line, f1, protocol=4)
503         # print("\n Color_data is saved :",s_name)
504
505     '''else :
506         # print("\n > Color_data were not saved. ')'''

```

```

507
508     return color_line
509
510     # ----- Inner Figure Delete Function -----
511     def contandline_detect(f, lw):
512         """
513         < Open CVを用いて、入れ子グラフ を消去するジュール >
514
515     入力
516         f : アレイデータ(BGR)
517         lw : 輪郭を白で上書きする際のラインの幅
518         dn : 点線を認識する投票数のしきい値
519         sn : 実線を認識する投票数のしきい値
520
521     出力
522         imrgb : アレイデータ(BGR)
523         """
524
525         imag = np.asarray(f)
526         # Gray (shape=2) のファイルのShapeをColorの3にそろえる
527         if len(imag.shape) == 2:
528             imag = cv2.cvtColor(imag,cv2.COLOR_GRAY2BGR)
529
530         imgray = cv2.cvtColor(imag,cv2.COLOR_BGR2GRAY)
531         ret,im_ = cv2.threshold(imgray,250,255,cv2.THRESH_BINARY_INV)
532
533         # imag_or = imag.copy() # original keep aside
534
535         dim = im_.shape # 画像の縦・横ピクセル数
536         s_dim = dim[1]*dim[0] # 画像の総ピクセル数
537         # print(s_dim)
538
539         #--- 輪郭検出を行い内挿図形と文字・記号を選別して削除する -----
540
541         # 輪郭検出
542         '''findContours の output が OpenCV の Version-Up (3.4 -> 4.0?)
543         に伴いimage, contours, hierarchy の 3個から
544         contours, hierarchy の 2個に変更された
545         '''
546
547         '''
548         image,contours,hierarchy = cv2.findContours(im_,
549             cv2.RETR_LIST,
550             cv2.CHAIN_APPROX_SIMPLE
551         )
552
553         '''
554         contours, hierarchy = cv2.findContours(im_,
555             cv2.RETR_LIST,
556             cv2.CHAIN_APPROX_SIMPLE
557         )
558
559         # 検出された輪郭の面積、輪郭の点数により選択を行う
560         '''
561         内挿図形のインデックスを big_list,
562         文字や記号のインデックスを sml_list
563         '''
564
565         big_list = [] ; sml_list = []
566         for i, cnt in enumerate(contours):
567             cnt = np.squeeze(cnt, axis=1)

```

```

568     area = cv2.contourArea(cnt)
569
570     if s_dim/4 >= area >= s_dim/30 and len(contours[i]) < 100:
571         ...
572         内挿図形の抽出：
573         外枠、XY軸は除くため全面積の 1/4以下と
574         文字の輪郭などを排除するため1/30以上で選別する。
575         波形を取り込んだ場合を取り除くため要素数が100個以上も除く
576         ...
577         big_list.append(i)
578
579     elif s_dim/1000 <= area <= s_dim/700 :
580         ...
581         文字記号の抽出：
582         文字や記号を検出し、波形の選別はできるだけ避けるための選別
583         ...
584         sml_list.append(i)
585
586 # 内挿図形として抽出された輪郭を白塗りして元の画像に上書きする
587 for i in big_list :
588     cnt = np.squeeze(contours[i], axis=1)
589     # 輪郭の内部を白塗り
590     cv2.fillConvexPoly(imag,
591                        points=cnt,
592                        color=(255,255,255)
593                        )
594     # 輪郭を太くして白塗り
595     cv2.drawContours(imag,
596                    [cnt],
597                    -1,
598                    (255,255,255),
599                    20
600                    )
601
602 # 文字・記号として抽出された輪郭を白塗りして元の画像に上書きする
603 for i in sml_list :
604     cnt = np.squeeze(contours[i], axis=1)
605     # 輪郭内部を白塗り
606     cv2.fillConvexPoly(imag,
607                        points=cnt,
608                        color=(255,255,255)
609                        )
610
611     # 輪郭を白塗り
612     cv2.drawContours(imag,
613                    [cnt],
614                    -1,
615                    (255,255,255),
616                    lw
617                    )
618
619     return imag
620
621 # ----- Dizitization Evaluate Function -----
622 def wf_eval(wo_cont,
623            prd_ol,
624            l_data,
625            ng_bk,col_n,
626            lin_nu,
627            dim
628            ):

```

```

629     print('\n> Evaluation of the digitizing ')
630
631     """
632     < 入力画像と予測数値を比較して数値化の信頼性の評価を行うモジュール >
633
634     入力
635         wo_cont : 背景画像は文字、輪郭を削除後の画像
636         prd_ol :
637             l_1_data,l_2_data,l_3_data : 数値データ
638         ng_bk
639         col_n : 色数
640         lin_nu : ライン数
641         dim : 計算次数
642     出力
643         n_zp : 信頼度
644     """
645
646     # 背景画像は文字、輪郭を削除後の画像を用いる
647     base_im = cv2.resize(wo_cont,(dim,dim),cv2.INTER_LANCZOS4)
648
649     ndpi = 360
650     ww = dim/ndpi ; hh = ww
651     lw = ndpi/72*2
652     fig = plt.figure(figsize=(ww,hh), dpi=ndpi)
653     ax = fig.add_axes([0,0,1,1])
654
655     # 原点の外側の領域を白でマスキング
656     cv2.rectangle(base_im,
657                  (0,0),(prd_ol[0,0],dim-1),
658                  (255,255,255),thickness=-1
659                  ) # left side
660
661     cv2.rectangle(base_im,
662                  (prd_ol[0,0]+prd_ol[0,2],0),(dim-1,dim-1),
663                  (255,255,255),thickness=-1
664                  ) # right side
665
666     cv2.rectangle(base_im,
667                  (0,dim-1),(dim-1,dim-prd_ol[0,1]-1),
668                  (255,255,255),thickness=-1
669                  ) # lower
670
671     cv2.rectangle(base_im,
672                  (0,0),(dim-1,dim-prd_ol[0,1]-prd_ol[0,3]-1),
673                  (255,255,255),thickness=-1
674                  ) # upper
675
676     ax.imshow(base_im)
677
678     ...
679     背景画像の画像の点数を求める。黒にラインがない場合は
680     あらかじめ点数 ng_bk を引いておく
681     ...
682     base_rv = 255-cv2.cvtColor(base_im, cv2.COLOR_BGR2GRAY)
683     base_n_z = len(np.nonzero(base_rv)[0]) - ng_bk
684
685     ...
686     背景画像に数値化データを白いラインで上書。一致する部分は画像上消える
687     lwでラインの幅を指定。
688     ...
689     fx = np.linspace(0,dim-1,dim)

```

```

690 for i in range(0,sum(lin_nu)):
691     ax.plot(prd_ol[0,0]+fx*prd_ol[0,2]/dim ,
692             l_data[0:dim,0,i]*prd_ol[0,3]+
693             dim-prd_ol[0,1]-prd_ol[0,3],
694             'w-',lw=lw
695             )
696
697 # 不要な枠線を描かない
698 ax.spines["right" ].set_color('none')
699 ax.spines["top"   ].set_color('none')
700 ax.spines["left"  ].set_color('none')
701 ax.spines["bottom"].set_color('none')
702 ax.get_xaxis().set_visible(False)
703 ax.get_yaxis().set_visible(False)
704
705 plt.ylim(dim,0);plt.xlim(0,dim)
706
707 '''上書き画像をキャンバス上に描写. バッファに取り込みアレイ化
708 ...
709 fig.canvas.draw()
710 buf = np.frombuffer (fig.canvas.tostring_rgb(),
711                     dtype=np.uint8
712                     )
713 imar = np.reshape(buf,(dim,dim,-1))
714
715 '''上書き画像の画像の点数を求める.
716 ...
717 gray_rv = 255-cv2.cvtColor(imar, cv2.COLOR_BGR2GRAY)
718 # plt.show()
719 plt.close()
720 n_z = len(np.nonzero(gray_rv)[0]) - ng_bk
721
722 '''上書き画像の画像点数と入力（背景）画像の画像点数の比率を求め、
723 1 から引いて信頼度とする.
724 ex. 完全に一致した場合上書き画像の点数は 0 ,信頼度は 1
725 ...
726 n_zp = 1-n_z/base_n_z
727
728 return n_zp, n_z, base_n_z
729
730 # ----- Axis Gain Read Function -----
731 def axgain(ff_n,sz,prd_ol):
732     print( '\n> Axis Gain Read ')
733
734     """"
735     < XY軸の軸目盛をOCR(pyocr)で読み取り、軸ゲインを推定する >
736
737     input
738     ff_n: file name
739     sz: image size
740     prd_ol: Origin position and axis length
741
742     retuen
743     x0:x軸の始点の値
744     xe:x軸の終点の値
745     y0:y軸の始点の値
746     ye:y軸の終点の値
747
748     """"
749
750 #原点位置の読み出し

```

```

751 org_axl=prd_ol[0]/1024
752
753 # --- 前処理 -----
754 # Image to array
755 ar_im = cv2.imread(ff_n)
756 # Origin of array
757 x_or_p = int(sz[0]*org_axl[0])
758 y_or_p = int((1-org_axl[1])*sz[1])
759 # Trimming
760 ar = ar_im.copy()
761 ar[0:y_or_p+10,x_or_p-1:(sz[0]-1),:] = 255
762 x_trm = ar[y_or_p:sz[1]-1,0:sz[0]-1]
763 y_trm = ar[0:y_or_p+10,0:x_or_p-5]
764
765 # --- Pyocr open -----
766 tools = pyocr.get_available_tools()
767
768 if len(tools) == 0:
769     print("No OCR tool found")
770     sys.exit(1)
771
772 # --- X軸数値の読み取り -----
773 # Gamma Correction
774 g = 0.5
775 lookUpTable = np.zeros((256, 1), dtype = 'uint8')
776 for i in range(256):
777     lookUpTable[i][0] = 255 * pow(float(i) / 255, 1.0 / g)
778
779 x_trm = cv2.LUT(x_trm, lookUpTable)
780 # Array to image
781 image = Image.fromarray(x_trm)
782 tool = tools[0]
783 # OCR
784 x_res = tool.image_to_string(image,
785                             lang="eng",
786                             builder=pyocr.builders.\
787                                 WordBoxBuilder\
788                                 (tesseract_layout=3)
789                             )
790 x_nl=[]
791 x_pl=[]
792 i=0
793 for d in x_res:
794     # print(i,d.content,d.position)
795     num = re.sub("\D", "",d.content)
796     jud = str.isdigit(num)
797     if jud == True:
798         er=0
799         # floatでのエラーで計算が止まることの回避
800         try:
801             x_nl.append(float(d.content))
802         except ValueError:
803             er = 1
804     if er == 0:
805         x_pl.append((d.position[1][0]+d.position[0][0])/2)
806
807 cv2.rectangle(x_trm,
808               d.position[0],
809               d.position[1],
810               (255,0,0),
811               1

```



```

812         )
813         i=i+1
814
815 # 小さい順に左から並んでいるとして
816 xn = len(x_nl)
817 if xn >=2 :
818     x_max =x_nl[xn-1]
819     x_min =x_nl[0]
820     x_max_p =x_pl[xn-1]
821     x_min_p =x_pl[0]
822     # pixel当たりの増分を計算
823     x_d = x_max - x_min
824     x_ld =x_max_p-x_min_p
825     x_r = x_d/x_ld
826     # 原点の値を計算
827     x0 = round(x_min-(x_min_p-x_or_p)*x_r,2)
828     xe = round(x0+x_r*(sz[0]*org_axl[2]),2)
829     print (' X-axis Gain:',x0,'-',xe)
830 else:
831     x0 = 0
832     xe = 1
833     print(' *I can not find X-axis Gain. Defalt value set:',x0,'-',xe)
834
835 # --- Y軸の数値読み取り -----
836 # Gamma Correction
837 g = 0.3
838 lookUpTable = np.zeros((256, 1), dtype = 'uint8')
839 for i in range(256):
840     lookUpTable[i][0] = 255 * pow(float(i) / 255, 1.0 / g)
841
842 y_trm = cv2.LUT(y_trm, lookUpTable)
843
844 image = Image.fromarray(y_trm)
845 tool = tools[0]
846 y_res = tool.image_to_string(image,
847                               lang="eng",
848                               builder=pyocr.builders.\
849                                   WordBoxBuilder\
850                                   (tesseract_layout=3)
851                               )
852
853 y_nl=[]
854 y_pl=[]
855 i=0
856 for d in y_res:
857     # print(i,d.content,d.position)
858     num = re.sub("\D", "", d.content)
859     jud = str.isdigit(num)
860     if jud == True:
861         er=0
862         # floatでのエラーで計算が止まることの回避
863         try:
864             y_nl.append(float(d.content))
865         except ValueError:
866             er = 1
867         if er == 0 :
868             y_pl.append((d.position[1][1]+d.position[0][1])/2)
869
870 cv2.rectangle(y_trm,
871               d.position[0],
872               d.position[1],

```

```

873               (255,0,0),
874               1
875               )
876         i=i+1
877
878 # 大きい順に上から並んでいるとして
879 yn = len(y_nl)
880 if yn >=2 :
881     ""OCRの結果は大きさ順ではない場合がある
882     ""
883     y_max = y_nl[y_pl.index(min(y_pl))]
884     y_min = y_nl[y_pl.index(max(y_pl))]
885     y_max_p = y_pl[y_pl.index(min(y_pl))]
886     y_min_p = y_pl[y_pl.index(max(y_pl))]
887
888     # pixel当たりの増分を計算
889     y_d = y_max - y_min
890     y_ld = abs(y_max_p-y_min_p)
891     y_r = y_d/y_ld
892     # 始点・終点の値を計算
893     y0 = round(y_min-(y_or_p-y_min_p)*y_r,2)
894     ye = round(y0 + y_r*(sz[1]*org_axl[3]),2)
895     if y0 == ye :
896         y0 = 0.0
897         ye = 1.0
898
899     print (' Y-axis Gain:',y0,'-',ye)
900
901 elif yn == 1 :
902     y_max = y_nl[0]
903     y_min = 0
904     y_max_p = y_pl[0]
905     y_min_p = y_or_p
906     # pixel当たりの増分を計算
907     y_d = y_max - y_min
908     y_ld = abs(y_max_p-y_min_p)
909     y_r = y_d/y_ld
910     # 始点・終点の値を計算
911     y0 = round(y_min-(y_or_p-y_min_p)*y_r,2)
912     ye = round(y0 + y_r*(sz[1]*org_axl[3]),2)
913     print (' Y-axis Gain:',y0,'-',ye)
914
915 else:
916     y0 = 0
917     ye = 1
918     print(' *I can not find Y-axis Gain. Defalt value set:',y0,'-',ye)
919
920 return x0,xe,y0,ye
921
922 # ----- Line Deleat Function -----
923 def linedel(f):
924     ""
925     < 軸などの長い直線を消去する関数 >
926
927     f: gray_array_data
928
929     ""
930
931     imggray = f
932     dim = 1024
933

```

```

934 ret,im_mono = cv2.threshold(imgray,254,255,0)
935 # --- 直線の検出 -----
936 minLineLength = int(dim/2)
937 maxLineGap    = 1
938 lines         = cv2.HoughLinesP(255-im_mono,
939                                1,
940                                np.pi/180,
941                                int(dim/1.3),
942                                minLineLength,
943                                maxLineGap
944                                )
945 '''第1引数は入力画像であり、2値画像
946 第2,3引数にはそれぞれrouとshitaの精度を指定します。
947 第4引数は、直線とみなされるのに必要な最低限の投票数を意味するしきい値です。
948 投票数は直線上の点の数に依存する。この引数は検出可能な線の長さの最小値。
949 '''
950
951
952 v_l = np.zeros([0, 4])
953 h_l = np.zeros([0, 4])
954 # s_l = np.zeros([0, 4])
955 if lines is not None:
956     '''linesが何も検出できなかったときは処理を行わない
957     '''
958     # 垂直線と水平線および斜線に分離する
959     for i in range(len(lines)):
960         for x1,y1,x2,y2 in lines[i]:
961             out = [x1,y1,x2,y2]
962             if abs(x1-x2) <= 0.5 and dim/10 < y1-y2 :
963                 v_l = np.append(v_l, np.array([out[:]]),axis=0)
964             elif y1 == y2 and dim/10 < x2-x1:
965                 h_l = np.append(h_l, np.array([out[:]]),axis=0)
966             ...
967         else:
968             s_l = np.append(s_l, np.array([out[:]]),axis=0)
969         ...
970
971     # 出力画像
972     for j in range(0,len(v_l)):
973         x1,y1,x2,y2 = v_l[j]
974         # plt.plot([x1,x2],[y1,y2],'r',lw = 5)
975         cv2.line(imgray,
976                 (int(x1),int(y1)),(int(x2),int(y2)),
977                 (255),
978                 int(dim/100)
979                 )
980     for k in range(0,len(h_l)):
981         x1,y1,x2,y2 = h_l[k]
982         cv2.line(imgray,
983                 (int(x1),int(y1)),(int(x2),int(y2)),
984                 (255),
985                 int(dim/100)
986                 )
987
988 return imgray
989
990 # _____ END of Code _____

```

END