SAtraceFit

SAtraceによって収集された5分間隔のファイルを集計するPythonコードの説明を行う。

データ定義

データ形式

1000行4列のtxtファイル

ファイル名=タイムスタンプ(測定時刻yyyymmdd_HHMMSS.txt)

日付が最初の8桁、アンダースコアを挟んで、時間を6桁で表している。

1列目 : 行数
2列目 : maxホールド
3列目 : 平均値
4列目 : minimumホールド

スタート周波数が行数をxとしたとき、(x*4+22000)/1000で周波数となる。
ここでは2列目

データの場所

ファイル名confidential.pyに変数rootPathとして格納した。

各モジュールの説明

main.py ver2.3

UPDATE2.3
外部に出したくない情報は別ファイルに格納(confidential.py)

INTRODUCTION
各モジュールを動かすメインファイル

ACTION
引数:
dateFirst, dateLast : コーンソールから入力、テストの際はコード内で書き換える
oldcsv, newcsv : コード内で書き換える

戻り値:なし(CSVファイルに書き込む)

1. datelistにより、フィッティングを行う最初の日付から最後の日付までのリストを抽出する。
   コンソールに出力
2. (oldcsv,newcsv)で、読み込み元CSV, 書き込み先CSVファイルを指定する。
   rawdataPathで、データの位置を指定する(日付)
3. confidentialにより、rootディレクトリとfittingに必要な周波数を指定する。
4. CSV_IOにより、CSVを読み込む。
5. fittingDivにより、fittingを行う。
6. CSV_IOにより、CSVを書き込む。

USAGE
+ コマンドライン上にて `python main.py <最初の日付> <最後の日付>`
+ フォーマットはyymmdd形式(例えば2015年11月1日=151101と打ちこむ)
+ CSV_IO.editCSV内でread, writeメソッドを1つの関数に収めた
+ fitting>read>translate>update>translate>writeの流れは1セット

PLAN
+ プログラムを途中で止めるとこれまでの計算結果が記録されない

> writeメソッドが走るのはfor文の最後だから
> read, writeメソッドが走るタイミングを調整する
> + 二重起動すると強制終了される
> マルチプロセスかができない

datelist.py ver1.0

UPDATE1.0
first commit

INTRODUCTION
main.pyから2つの値が渡される
2つの間の日付のリストを返す

ACTION

yymmdd形式の6桁をdatetime関数で日付にする
ddate1,2に代入する
ddate1がddate2になるまで(while)
ddate1を1日ずつ足して、リストに追加する(append)

USAGE
引数 : 最初の日付yymmdd形式、最後の日付yymmdd形式
戻り値 : 最初の日付から最後の日付を入れたリストyymmdd形式

PLAN
none

confidential.py ver1.0

INTRODUCTION
データの入ったrootディレクトリと注目する周波数を指定するpy

ACTION
関数入れているだけ

USAGE
root():データの入ったrootディレクトリ
引数:なし
戻り値:rootPath(文字列)

freq() : 注目する周波数を指定する
引数 : なし
戻り値 : freqWave(リスト)

UPDATE1.0
first commit

PLAN
none

CSV_IO.py ver1.0

INTRODUCTION
関数readCSV,writeCSV, editCSVで構成される。
組み込み関数csv, os.path, datetime.datetimeを使用して、fittinngの結果を読み込み、書き込み、編集を行う。

各関数の説明は各関数のdocを参照。

readCSV.py ver1.1

UPDATE1.1
クラス'csv.DictReader()'を使ってディクショナリ形式でcsvを取り出す。
キーは見出し(csvの1行目)
行を読むたびリスト'dictList'へ追加する
dictListを返す
INTRODUCTION
csvファイルの中身をディクショナリに入れるpy
引数：ファイル名
戻り値：ディクショナリ'mydict
ACTION
filenameにいれた名前のファイルを開く
リスト'reader'に格納する
リスト'reader'に対して0要素目をキーに、1要素目以降(リスト形式)を値にしたディクショナリ'mydict'を作成する

writeCSV.py ver1.3

UPDATE1.3
周波数の指定は外部ファイルから渡されてくる周波数のリスト'outparam'
計算結果は外部ファイルから渡されてくる計算結果のリスト'cal_result'
INTRODUCTION
辞書の内容ををcsvファイルを書き込む
ACTION
引数として集合'outparam',辞書'cal_result'を渡す
戻り値なし
csvを返す
日付時間のキーは行のラベルにあたる
周波数のキーは列のラベルにあたる
USAGE
列のラベル'paramnames'を定義する
{日付時間,周波数1,周波数2,…}がディクショナリになった'fit_result'を定義する
csv_writer(引数1,引数2)を実行する
改造予定
datetimeでソートしたい

editCSV ver1.0

INTRODUCTION

fitting>read>translate>update>translate>writeの流れを一まとめにした

ACTION

READ FROM CSV

> CSVファイルを読み込む

UPDATE DICTIONARY

> 読み込んだcsvとappenddictを併せる
>
> > dictionary形式は同一キーが存在した場合、後から来たdictionary内のキーの値に更新する

WRITE TO CSV

> 更新したdictionaryをcsvに書き込む

USAGE

引数:

- readcsv:読み込みcsvファイル名
- writecsv:書き込みcsvファイル名
- appendDict:書き込む内容。fittingの結果
- freqWave:csvの1行目(見出し行)

戻り値:None(writecsvに書き込み)

UPDATE

first commit

PLAN

None

csv_dict_transfer.py ver1.0

UPDATE1.0

first commit

INTRODUCTION

関数datatocsv, csvtodataから構成される。

- datetocsv : dataをcsvファイルに読み書きしやすい、ディクショナリ in リ

スト形式に変換する

- csvtodata : csvの内容をデータ整理用の、ディクショナリ in ディクショナリ形式

> ディクショナリはキーが重複した場合、値を更新する機能、キーでソートする機能をもつ

ACTION

フィッティングしたデータをcsvファイルに書き込みたい
このときディクショナリ形式{key:val}の形で表現したとき

- keyはcsvの1列目(見出し)
- valは2列目以降(値)

を意味する。
このディクショナリ形式をn個の要素を持ったリストに収めていくと、n行のcsvが作成される(using CSVweiter.py)

```
(データ整理しやすい形式)
{   '20151201_000011':{'22.2kHz':-87, '23.0kHz:-40', ...},
    '20151201_000512':{'22.2kHz':-80, '23.0kHz:None, ...},
    ...}


            ^
            |   csv_dict_transfer.py
            v


(csvから読み書きしやすい形式)
[   {DateTime:'20151201_000011', '22.2kHz':-87, '23.0kHz:-40',...},
    {DateTime:'20151201_000512', '22.2kHz':-80, '23.0kHz:None,...},
    ...]
```

データ形式について少し整理

```
(データ整理しやすい形式)
外側ディクショナリの形式
    {ディクショナリ1,ディクショナリ2,...}
        つまり
    {'d0':dataInnerDic[0] ,'d1':dataInnerDic[1] ,'d2':dataInnerDic[2],...}
内側ディクショナリの形式
    ディクショナリ1='20151201_000011':{'22.2kHz':-87, '23.0kHz:-40', ...}
        つまり
    dataInnnerDic[0]={'d0Key1':'d0Val1','d0Key2':'d0Val2',...}


(csvから読み書きしやすい形式)
外側ディクショナリの形式
    [ディクショナリ1,ディクショナリ2,...]
        つまり
    [dic1,dic2,...]
内側ディクショナリの形式
    {DateTime:'20151201_000011', '22.2kHz':-87, '23.0kHz:-40',...},
        つまり
    {DateTime:'20151201_000011', '22.2kHz':-87, '23.0kHz:-40',...},
```

要するに、

```
(pythonのデータ整理用 ディクショナリ in ディクショナリ形式)
{
    {v00:{k1:v10,k2:v20,...,k(n-1):v(n-1,0),k(n):v(n,0)},
    {v01:{k1:v11,k2:v21},...,k(n-1):v(n-1,1),k(n):v(n,1)},
    ...,
    {v0m:{k1:v(1,m),k2:v(2,m),...,k(n-1):v(n-1,m),k(n):v(n,m)}
}

        ^
        |    <<ココの相互変換するモジュールを作りたい
        v
(pythonのcsv読み書き用 ディクショナリ in リスト形式)
[
    {k0:v00,k1:v10,k2:v20,...,k(n-1):v(n-1,0),k(n):v(n,0)},
    {k0:v01,k1:v11,k2:v21},...,k(n-1):v(n-1,1),k(n):v(n,1)},
    {k0:v02,k1:v12,k2:v22},...                                  },
    ...,
    {k0:v(0,m-1),k1:v(1,m-1),k2:v(2,m-1),...,k(n-1):v(n-1,m-1)),k(n):v(n,m
    {k0:v(0,m),k1:v(1,m),k2:v(2,m),...,k(n-1):v(n-1,m),k(n):v(n,m)}
]

        ^
        |    <<ココはpython組み込み関数でできる
        v

(CSVファイル)
k0,k1,k2 ,... ,k(n)
v00,v10,v20 ,... ,v(n,0)
v01,v11,v21 ,... ,v(n,1)
v02,v12,v22 ,... ,v(n,2)
.
.
v(0,m-1),v(1,m-1),v(2,m-1),...v(n,m-1)
v(0,m),v(1,m),v(2,m),...v(n,m)
```

USAGE

各関数のdoc参照

Testはコメントアウト外してbuildするのみ

datatocsv ver1.0

UPDATE1.0

first commit

INTRODUCTION

data(ディクショナリ in ディクショナリ形式)をcsvファイルに読み書きしやす

い、ディクショナリ in リスト形式に変換する

ACTION

1. 引数dataは{'日付':{周波数1:パワー1,周波数2:パワー2,…,周波数n:パワーn,}}で渡されてくる
2. 引数dataを2要素のリストにしてdataSortedListに代入する。 `dataSortedList= sorted(list(data.items()))`

   > items()メソッドを用いて日付(値)とデータ(ディクショナリ型)を1セットのタプルにする
   > list()ファンクションを用いてタプル形式をリスト形式にする。(sorted関数を使いたいため。)
   > sorted()ファンクションを用いて文字列(日付)→ディクショナリの順番で並ぶようにしてる

3. 空のリストcsvを作成
4. ここから引数dataの数だけ繰り返す `for i in range(len(data))`
   i. リストcsvをディクショナリ in リスト形式にする `csv.append({})`
   ii. csvというディクショナリ in リスト形式のi番目の要素に対して、「DateTime」をキーに、dataSortedListの0番目の要素=「日付」を値として追加する。 `csv[i]['DateTime']=dataSortedList[i][0]`

   > ソートされたので0番目の要素は文字列

   iii. csvというディクショナリ in リスト形式のi番目の要素に対して、「ディクショナリ(周波数をキーにしたシグナル強度)」を追加する。 `csv[i].update(dataSortedList[i][1])`

   > ソートされたので1番目の要素はディクショナリ(周波数をキーにしたシグナル強度)。

5. リスト in ディクショナリcsvを返して終了

USAGE

引数:pythonのデータ整理用 ディクショナリ in ディクショナリ形式
戻り値:pythonのcsv読み書き用 ディクショナリ in リスト形式

PLAN

none