

Remote Sensor Project

S.Matoike

2024 年 5 月 30 日

目次

第 1 章	システム全体の概要	5
1.1	全体の構成	5
第 2 章	ローカルにデータを保持する	7
2.1	リモートマシンの処理	8
2.2	ローカルマシンの処理	11
第 3 章	リモートにデータを保持する	27
3.1	リモートマシンの処理	27
3.2	ローカルマシンの処理	29
第 4 章	クラウド上にデータを保持する	31
4.1	リモートマシンの処理	32
4.2	ローカルマシンの処理	34
4.3	指定日から遡って指定日数分のグラフを作図する	39
第 5 章	補遺	47
5.1	shell を記述する上での注意点	47
5.2	GCS(google-cloud-storage)	50
5.3	新たなシステムへの移行の手順	56
参考文献		61

第 1 章

システム全体の概要

1.1 全体の構成



図 1.1 システム全体の構成

- 5 分毎にリモートマシン上の臭気センサの測定データをローカルマシンに読み取る
- 読み取った 5 分毎のデータは、追記、蓄積していくと同時に、実時間の臭気変化と

してグラフを描画更新する（ローカルマシン上）

- 23時56分に、日毎の処理を起動し、1日分の蓄積データをグラフ化すると共に、蓄積した1日分のデータは別フォルダに退避する（ローカルマシン上）
- 日曜日の0時1分に、週毎の処理を起動し、その日から過去1週間分の蓄積データを読んでグラフ化する（ローカルマシン上）
- クラウドにデータを保持することで、補助記憶装置の負荷を低減させるやり方は第3章にて

第2章

ローカルにデータを保持する

表 2.1 ここで使用する shell とプログラム

リモート	sensor.sh	日付文字列とセンサーの読み取り値を並べて出力する shell
リモート	adrsz0D.py	センサーを読むクラスの定義
リモート	procmain.py	センサーのインスタンス生成とセンサーの読み取り
ローカル	rsensor.sh	5 分間隔で cron に呼び出される。リモートの sensor.sh を起動して、結果をローカルの w.txt に追記蓄積する
ローカル	daily.sh	23 時 56 分に cron により起動される。1 日の蓄積データ w.txt の data/ フォルダへの退避と daily.py の起動
ローカル	daily.py	data/ フォルダ内の指定した日付の 1 日分のデータを読んでグラフを作図する
ローカル	weekly.sh	毎週日曜日の 0 時 1 分に cron により起動される。weekly.py を起動する
ローカル	weekly.py	data/ フォルダ内の指定日より前の 1 週間分のデータを読んでグラフを作図する
ローカル	rtplot.sh	rtplot.py を起動する。デスクトップに置いてマウスクリックで起動させる
ローカル	rtplot.py	w.txt を監視して、更新される度に実時間グラフを更新表示する
ローカル	myplot.py	daily.py、weekly.py から共通に呼び出されるグラフ作図用の関数

2.1 リモートマシンの処理

2.1.1 リモート機器、センサー

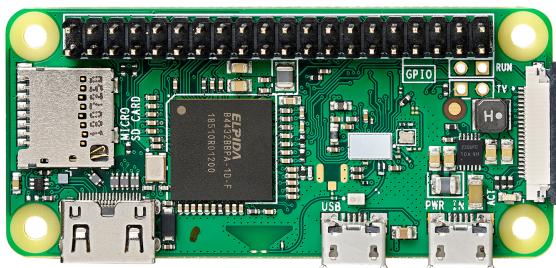


図 2.1 Raspberry Pi Zero2 W / Zero W



図 2.2 ADRSZOD:TP401A

臭気センサ TP-401A

- 低電力で動作可能な高感度の臭気センサ
- コンパクトな Raspberry Pi Zero と重ねて使用可能な pHAT サイズ
- アンモニア、水素、アルコール、一酸化炭素、メタンなど揮発性気体、タバコ、木材燃焼発生した煙など様々な気体に対応
- 連続通電を許容するので、様々な所での継続的なチェックが可能（動作時にはセンサ部が非常に高温となるため、取り扱いには注意）
- 動作状況が見えるステータス LED により、駆動状況の確認が容易

運用方針

- 基本的な運用形態は、モニタ、キーボード等を接続せず電源のみの接続で運用する
- このマシン上では、数行の shell を登録するだけなので、基本的には他の PC のコンソールから ssh で login して利用する（この例では、ユーザは mat）

```
ssh mat@192.168.3.27
```

- VNC を有効にして他の PC からアクセスすれば、デスクトップも利用できる

2.1.2 リモート処理の概要

Raspberri Pi Zero2 W/Zero W 上に用意したこの shell (sensor.sh) は、別の RaspberryPi マシンから ssh によって起動される

I2C で接続された臭気センサの出力（電圧値）は、Python のプログラム (adrszOD.py) によって読み取られる

adrszOD.py は AD 変換器の ch1 から ch4 までの値を出力しているので、その出力を awk にパイプして、1 つ目の ch1 の値だけを取り出している

date コマンドで作成している測定日付は、後ほど Python の matplotlib の都合に合わせて、新たにデータの整形処理が必要にならない様に、書式を「年 - 月 - 日と時 : 分 : 秒の間は、半角カンマ + 半角スペース 1 個」の格好になる様に整形する

日付とセンサの出力する値が 1 行になる様にするため、paste コマンド（テキストファイルを横に並べて結合するコマンド）を使っている^{*1} (sh では動かないで、bash にする)

paste の 2 つのオペランド、<(date +....) と、<(python3 ..procmain.py..) の各々のコマンドは、<(any command) の形に囲むことにより、読み取りのためのファイル記述子を指定したことになり、それぞれを 2 つのファイルの様に扱うことができる^{*1}

```
/home/mat/Documents/sensor.sh —————  
#!/usr/bin/bash  
dir='/home/mat/Documents'  
paste <(date +%Y-%m-%d,\ %H:%M:%S) \  
<(python3 ${dir}/procmain.py | awk '{print $1}')
```

この shell を起動してみると、次の通り

sensor.sh の出力例 —————

```
mat@raspberrypizero:~/Documents $ ./sensor.sh  
2024-04-17, 02:40:55 ch1:0.79827  
mat@raspberrypizero:~/Documents $
```

^{*1} 「Efficient Linux コマンドライン」 オライリー・ジャパン 初版第 1 刷 p.108、p.156（プロセス置換）

以下は、センサーを読むための Python のプログラム^{*1}

```
/home/mat/Documents/adrszOD.py
```

```
import smbus, time

class I2C:
    i2c_bus = None
    def __init__(self, bus_number=1):
        cls = type(self)      # 旧スタイルなら、cls = self.__class__
        cls.i2c_bus = smbus.SMBus(bus_number)

class TP401A(I2C):
    def __init__(self, address=0x68, Vref=2.048) -> None:
        super().__init__(bus_number=1)
        self.address = address
        self.Vref = Vref

    def swap16(self, x):
        return (((x << 8) & 0xFF00) | ((x >> 8) & 0x00FF))

    def sign16(self, x):
        return (-(x & 0b1000000000000000) | (x & 0b0111111111111111))

    def read_val(self, x):
        self.i2c_bus.write_byte(self.address, x) #16bit
        time.sleep(0.2)
        data = self.i2c_bus.read_word_data(self.address, 0x00)
        raw = self.swap16(int(hex(data), 16))
        raw_s = self.sign16(int(hex(raw), 16))
        volts = round((self.Vref * raw_s / 32767), 5)
        return volts
```

クラス TP401A が、クラス I2C を継承する様にしたのは、新たな別の I2C 対応デバイスを、同じバス上に接続する場合を想定したもの

TP401A クラスは I2C クラスを継承しているので、TP401A のコンストラクタの中で I2C クラスのコンストラクタを明示的に実行し、I2C バスの番号を指定している

TP401A クラスの上位クラス I2C クラスのクラス変数 i2c_bus へのアクセスは、「self.

^{*1} <https://github.com/bit-trade-one/RasPi-Zero-One-Series>

変数名」の格好でアクセスすることによって、インスタンス変数->クラス変数->親クラスのクラス変数の順に探索され、I2C クラスのクラス変数にたどり着けることになる

この TP401A クラスを使う主処理は、sensor.sh の中から呼び出されている
/home/mat/Documents/procmain.py

```
#!/usr/bin/env python
from sarzOD import TP401A

if __name__ == "__main__":
    smel = TP401A()
    volts1 = smel.read_val( 0b10011000 )
    volts2 = smel.read_val( 0b10111000 )
    volts3 = smel.read_val( 0b11011000 )
    volts4 = smel.read_val( 0b11111000 )
    out_msg = f"ch1:{volts1} ,ch2:{volts2},ch3:{volts3},ch4:{volts4}"
    print(out_msg)
```

2.2 ローカルマシンの処理

2.2.1 ローカル機器



図 2.3 Raspberry Pi 3/4/5

2.2.2 cron による時刻指定の処理

cron の仕組みを使う（3つの時刻指定の処理を登録している）

- (1) 5分ごとに、リモートセンサの値を読み取る処理、rsensor.sh を起動する
- (2) 毎日 23時56分になったら、その日1日分の測定データを退避する処理、daily.sh を起動する
- (3) 每週日曜日の朝0時1分になったら、先週1週間の（先週の日曜日の0時0分から昨日の土曜日の23時55分まで5分間隔で測定した）測定データをまとめの処理、weekly.sh を起動する

crontab -e

```
MAILTO=""
*/5 * * * * sh /home/mat/Documents/rsensor.sh
56 23 * * * sh /home/mat/Documents/daily.sh
1 0 * * Sun sh /home/mat/Documents/weekly.sh
```

ここで、MAILTO="" としているのは、「指定時刻に cron が処理を起動したよ」というメールを送信するために smtp を起動しようとして失敗し、指定の処理が起動されないという現象に対する対応である²

```
+----- minute (0 - 59)
| +----- hour (0 - 23)
| | +----- day of month (1 - 31)
| | | +----- month (1 - 12)
| | | | +--- day of week (0 - 6) (Sunday=0 or 7)
| | | | |
* * * * * command to be executed
```

² postfix や sendmail など、SMTP の何某かをインストールせよという説もあるが、特にメールの知らせが不要ならこのやり方が最も簡易な解決方法になる

2.2.3 リモートセンサの処理をローカルマシンから起動する

リモート（192.168.3.27）の Raspberry Pi Zero W に接続されている臭気センサの値を読みとる shell（sensor.sh）を、ローカル（192.168.3.21）のマシン Raspberry Pi から ssh を使って起動するための shell（rsensor.sh）であり、cron によって 5 分ごとに起動される

この時 ssh から、リモートマシン（Pi Zero）のユーザ（今の環境では mat）のパスワードを要求されるが、人手によるインタラクティブな入力操作を想定していないので、予め sshpass を導入しておいて、それによってパスワードを自動応答させる様にしている

ssh でリモートの shell を実行した結果を受け取り、ローカルの w.txt に追記し蓄積している

```
/home/mat/Documents/rsensor.sh —————  
#!/usr/bin/sh  
### sudo apt install -y sshpass ###  
dir='/home/mat/Documents'  
addr='192.168.3.27'  
usr='mat'  
pass='mypassword'  
sshpass -p ${pass} ssh ${usr}@${addr} \  
"bash ${dir}/sensor.sh" >> ${dir}/w.txt
```

w.txt の様子は、tail -f w.txt などとしてみると、5 分ごとに起動された shell（rsensor.sh）によって、測定結果が蓄積されていく様子がわかる（このファイルは、1 日の終わりの処理で 2024-04-16.txt の様に、日付をつけたファイル名に変えて保存している）

```
w.txt -> 2024-04-16.txt —————  
2024-04-16, 00:35:04 ch1:1.0896  
2024-04-16, 00:40:04 ch1:1.09016  
2024-04-16, 00:45:04 ch1:1.09322  
.... ....  
2024-04-16, 23:45:04 ch1:0.9059  
2024-04-16, 23:50:04 ch1:0.99653  
2024-04-16, 23:55:04 ch1:0.98809
```

2.2.4 日毎の処理

毎日の23時55分に、その日の最後の測定が終わる事を前提として、cronには23時56分になったら次のshell(daily.sh)を起動する様にしている

このshellでは、現在時刻が23時55分を超えていることを確認した上で、次のことを行っている

- 蓄積された1日分の測定データw.txtを、bkup.txtという名前のファイルにコピーしている
- w.txtを、dataフォルダ以下に、「今日の日付.txt」という名前に変更して保存している
- 空のファイルw.txtを次の日の測定データの蓄積場所のために用意している(touch)
- Pythonの仮想環境venv11に入って、daily.pyを起動している(第1引数にdataフォルダ以下に保存した測定データファイルの名前=年月日を指定している)
- プログラムの実行を終えたら、Pythonの仮想環境venv11から抜けている

```
/home/mat/Documents/daily.sh
#!/usr/bin/sh
dir='/home/mat/Documents'
dates=$(date +%Y-%m-%d)
hour=$(date +%H)
minu=$(date +%M)
if [ ${hour} -ge 23 ]; then
    if [ ${minu} -gt 55 ]; then
        cp ${dir}/w.txt ${dir}/bkup.txt
        mv ${dir}/w.txt ${dir}/data/${dates}.txt
        touch ${dir}/w.txt
        . ${dir}/venv11/bin/activate
        python3 ${dir}/daily.py ${dates}
        deactivate
    fi
fi
```

2.2.5 日毎処理のプログラム

myplot.py から、graph_plot() 関数を import している（この関数の詳細は後述）

第 1 引数に指定があったら、そこで指定された日付のデータを元に、グラフを描画できる様にしている

指定のデータファイルを開いて、最初の 20 文字までは、予め matplotlib の dates に対応した様式の日付文字列にしているので、それを Unix の日付けに変換している

25 文字目以降は、測定した電圧の値だが、改行文字が付いていたりするので、.strip() によってホワイトスペースを除去して、実数に変換している

日付と電圧のリストを作って、それを graph_plot() 関数に渡して作図させている

/home/mat/Documents/daily.py

```
import sys
from datetime import datetime as dt
import matplotlib.dates as mdates
from myplot import graph_plot

dirstr = '/home/mat/Documents'
today = dt.now().strftime('%Y-%m-%d')
if len(sys.argv) > 1:
    today = sys.argv[1]

list0=[]
with open(dirstr + "/data/" + today + ".txt") as f:
    for line in f:
        list0.append(line)

title = 'Smel : '
fig = graph_plot(list0, title)
#fig.savefig(dirstr + '/figs/' + today + '.png')
```

この Python のプログラムを実行した結果として、次の様な 1 日分のデータの変化をグラフ化したものが得られる

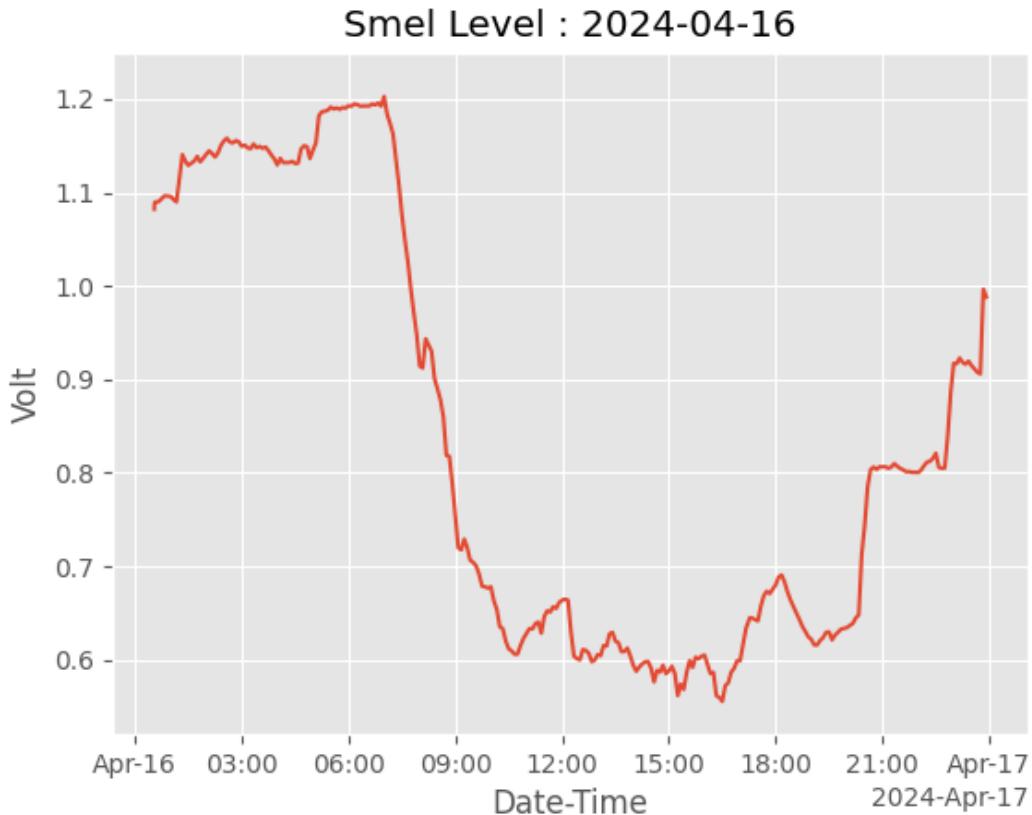


図 2.4 daily.py の出力 (4月 16 日の 0 時から 24 時まで)

2.2.6 グラフ描画のための関数

daily.py と weekly.py で共通に呼び出している `graph_plot()` 関数を、以下の様に定義している

横軸に日付と時刻（データは 5 分毎）を置くため、少し工夫が必要になる

ポイントは、`matplotlib` が理解する日付の様式で書いたものを使うことさえできれば、`matplotlib` の `dates` に、`locator` と `formatter` を自動生成させ、それ以下の全ての面倒な処理を任せきことができる^{*3}点にある

^{*3} https://matplotlib.org/stable/api/dates_api.html の記事が参考になる

```
/home/mat/Documents/myplot.py —————  
import numpy as np, pandas as pd  
from matplotlib import pyplot as plt, style, dates as mdates  
def chop(daystr):  
    x, y = [], []  
    for dstr in daystr.split('\n'):  
        if len(dstr)>0:  
            datev = mdates.datestr2num(dstr[:20])  
            x.append(datev)  
            value = float(dstr[25:].strip())  
            y.append(value)  
    return x, y  
def graph_plot(datalist, tstr):  
    x, y = [], []  
    for daystr in datalist:  
        x0, y0 = chop(daystr)  
        x.extend(x0)  
        y.extend(y0)  
    ymax, ymin, yave = max(y), min(y), np.mean(y)  
    title = tstr +\n        f'Min={ymin:0.3f}, Max={ymax:0.3f}, Mean={yave:0.3f}'  
    matplotlib.style.use('ggplot')  
    fig, ax = plt.subplots()  
    locator = mdates.AutoDateLocator()  
    formatter = mdates.ConciseDateFormatter(locator)  
    ax.xaxis.set_major_locator(locator)  
    ax.xaxis.set_major_formatter(formatter)  
    ax.set_ylim([ymin*0.9, ymax*1.1])  
    ax.set_xlabel('Date-Time')  
    ax.set_ylabel('Volt')  
    ax.set_title(title)  
    ax.plot(x, y)  
    plt.show()  
    return fig
```

2.2.7 週毎の処理

この処理は cron によって、毎週日曜日の朝 0 時 1 分に起動される

この shell (weekly.sh) では、0 時 0 分より後の時刻ならば仮想環境 venv11 に移って、Python のプログラム (weekly.py) を起動し、終わったら仮想環境から抜けている

Python のプログラムは、第 1 引数に日付を受け取ることができる様にしている

```
/home/mat/Documents/weekly.sh
```

```
#!/usr/bin/sh
dir='/home/mat/Documents'
dates=$(date +%Y-%m-%d)
hour=$(date +%H)
minu=$(date +%M)
if [ ${hour} -ge 0 ]; then
    if [ ${minu} -gt 0 ]; then
        . ${dir}/venv11/bin/activate
        python3 ${dir}/weekly.py ${dates}
        deactivate
    fi
fi
```

2.2.8 週毎処理のプログラム

myplot.py から graph_plot() 関数を import している

第 1 引数で処理対象にする最終日付を受け取ることができる様にしている

指定された処理対象の最終日付から、7 日前の日付を求めて、7 日間の日付のリスト datelist を作っている

7 日間の日付リスト datelist から 1 日ずつ取り出しては、そのファイル名のデータを data フォルダから読み出し、日付の変換と電圧の実数への変換を行い、それらのデータのリストを list0 に追記していっている

もし、指定した日付のデータファイルがなかった場合は、例外処理で受けて黙って次の日付のファイルのデータ取得に移る様にしている

graph_plot() 関数に list0 を渡してグラフの作図をさせている

```
/home/mat/Documents/weekly.py —————  
from datetime import datetime as dt, timedelta  
import sys, matplotlib.dates as mdates  
from myplot import graph_plot  
  
dirstr = '/home/mat/Documents'  
s_format = '%Y-%m-%d'  
today = dt.now()  
if len(sys.argv) > 1:  
    today = dt.strptime(sys.argv[1], s_format)  
startstr = (today - timedelta(days=7)).strftime(s_format)  
  
datelist = []  
for i in range(7):  
    day = today - timedelta(days=(7-i))  
    datelist.append(day.strftime(s_format))  
  
list0=[]  
for fname in datelist:  
    try:  
        with open(dirstr + "/data/" + fname + ".txt") as f:  
            for line in f:  
                list0.append(line)  
    except FileNotFoundError:  
        continue  
  
title = 'Smel(W) : '  
fig = graph_plot(list0, title)  
#fig.savefig(dirstr + '/figs/' + startstr + '.png')
```

以下は、週毎処理が output する画像である。4月 21 日（日曜日）の 0 時 1 分に cron によって起動され実行されたものだが、フォルダ data の中には、測定を始めた 4月 16 日（火曜日）から、4月 20 日（土曜日）までの 5 日分のファイルがあるだけである。本来なら 1 週間前の日曜日（4月 14 日）のデータから週毎処理の対象となるところだが、プログラムの中では FileNotFoundError の例外を捕捉して、とにかく continue によって処理を継続させているため、何事もなかったかの様に、データファイルの所在する火曜日以降がグラフに作図されている。

また、日毎処理のプログラム daily.py と全く同じグラフ作図の関数（myplot.py の中の graph_plot() 関数）を使っているのだが、横軸の設定が matplotlib の dates (mdates として

プログラム中では使っている) によって、自動的に調整されているのが分かる(便利)

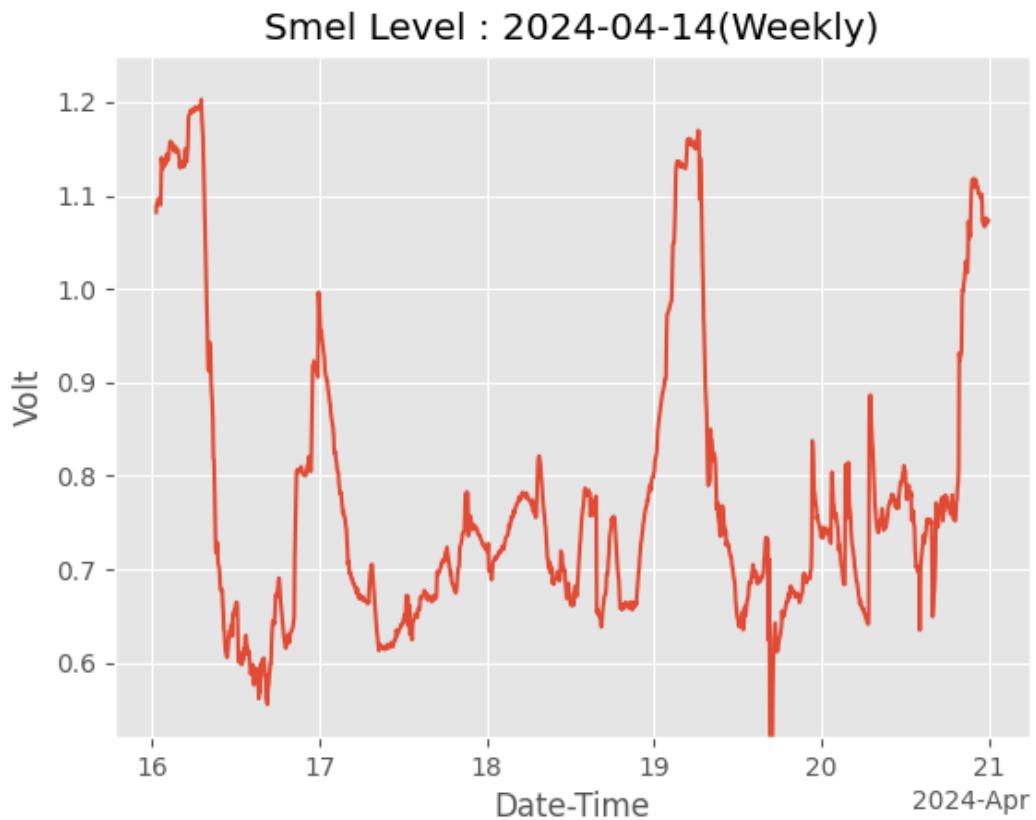


図 2.5 weekly.py の出力

2.2.9 測定データのグラフを実時間で描画する

RTPlot クラスは、実時間によるグラフ描画に必要な関数類をまとめたもの

ポイントとなるのは、plt.show() を使わず、plt.pause(1.0) を使うこと、tail -f をプログラム上で実現することの 2 点

self.x と self.y のリストに順にデータを追記していく、新たなデータが w.txt に追加されたタイミングで plt.pause(1.0) としている（引数の値を 0.1 や 0.01 にすると Raspberry Pi では全て描画しきれないので注意）

tail_f() 関数は、コマンド tail -f w.txt を Python のプログラムで実現したもの（指定したファイルの最終行が追記されるまで continue で待っている）である

```
import sys, time, numpy as np
from datetime import datetime as dt, timedelta
from matplotlib import pyplot as plt, dates as mdates, style

class RTPlot:
    def dt2str(self, dat, sformat):
        return dat.strftime(sformat)

    def str2dt(self, str, sformat):
        return dt.strptime(str, sformat)

    def dateproc(self, datestr):
        #s_format = '%Y-%m-%d, %H:%M:%S'
        s_format0 = '%Y-%m-%d, 00:00:00'
        self.today = dt.now()
        self.todaystr = self.dt2str(self.today, s_format0)
        if datestr != 'w' and datestr != self.todaystr[:10]:
            self.today = self.str2dt(datestr, '%Y-%m-%d')
            self.todaystr = self.dt2str(self.today, s_format0)
        self.tomorrow = self.today + timedelta(days=1)
        self.tomorstr = self.dt2str(self.tomorrow, s_format0)

    def readdata(self, fname):
        try:
            with open(fname, 'r') as f:
                for line in f:
                    self.chop(line)
```

```
except FileNotFoundError:
    print(f'File{fname} not found')

def __init__(self, datestr, fpath):
    self.fname = fpath + datestr + '.txt'
    self.x = []
    self.y = []
    self.ims = []
    self.dateproc(datestr)
    self.readdata(self.fname)
    matplotlib.style.use('ggplot')
    self.fig, self.ax = plt.subplots()
    locator = mdates.AutoDateLocator()
    formatter = mdates.ConciseDateFormatter(locator)
    self.ax.xaxis.set_major_locator(locator)
    self.ax.xaxis.set_major_formatter(formatter)
    xmin = mdates.datestr2num(self.todaystr)
    xmax = mdates.datestr2num(self.tomorstr)
    self.ax.set_xlim([xmin, xmax])
    self.yrange()
    self.ax.set_xlabel('Date-Time')
    self.ax.set_ylabel('Volt')
    self.update()

def chop(self, line):
    datestr = line[:20].strip()
    datev = mdates.datestr2num(datestr)
    value = float(line[25:].strip())
    self.x.append(datev)
    self.y.append(value)

def tail_f(self):
    try:
        with open(self.fname, 'r') as f:
            f.seek(0, 2)
            enter = dt.now()
            while True:
                line = f.readline()
                if not line:
                    time.sleep(1)
```

```
        if dt.now()-enter < timedelta(minutes=6):
            continue
        return
    return line.strip()
except FileNotFoundError:
    print(f'File{self.fname} not found')
return

def statvalue(self):
    selfymax = np.array(self.y).max()
    selfymin = np.array(self.y).min()
    selfmean = np.array(self.y).mean()
    str1 = f"Max={selfymax:.2f}, "
    str2 = f"Min={selfymin:.2f}, "
    str3 = f"Mean={selfmean:.2f}"
    self.ax.set_title("Smel Level : " + str1 + str2 + str3)

def yrange(self):
    self.statvalue()
    step = (selfymax - selfymin)/10.0
    v = 0
    while selfymax > v:
        v += step
    ymax = v + step/2.0
    while selfymin < v:
        v -= step
    ymin = v - step/2.0
    self.ax.set_ylim([ymin, ymax])

def update(self):
    self.yrange()
    if len(self ims) > 0:
        im = self ims.pop()
        im.remove()
        im, = self.ax.plot(self.x, self.y, color='red')
        self ims.append(im)

if __name__ == "__main__":
    dirstr = '/home/mat/Documents'
    datestr = 'w'
```

```

fpath = dirstr + '/'
if len(sys.argv) > 1:
    datestr = sys.argv[1]
    fpath = dirstr + '/data/'
rtp = RTPlot(datestr, fpath)
while dt.now() < rtp.tomorrow:
    plt.pause(1)
    line = rtp.tail_f()
    if line is not None:
        print("[info.log]", line)
        rtp.chop(line)
        rtp.update()
    rtp.fig.savefig(dirstr + '/figs/R_' + rtp.todaystr[:10] + '.png')

```

`tail_f()` 関数の中では、6分を超えて `continue` を繰り返している場合に `None` を返す事にしている（6分待てば、その間には必ず5分間隔のイベントは入ってくると思う）ので、`main` の中の繰り返し処理 `while` の中でその事を（PEP8 が可読性の観点から推奨している書き方）`is not None` によって判定している

実時間処理の画面は次の様になる。

このプログラム `rtplot.py` が終了時に保存する画像は、実は日毎処理のプログラム `daily.py` が出力するものとほぼ同じ傾向を示しているはずだ。ただ `rtplot.py` の方は、新たな点をプロットするたびに y 軸の最大値と最小値を更新しているため、日毎の処理では作図の範囲外に出るデータも示せているし、また具体的な最大値、最小値、平均値の値もその都度更新して表示している

この実時間プロットのプログラムを起動する shell を、実行可能属性を持たせてデスクトップに置いている（マウスのクリックで起動できる）

/home/mat/Desktop/rtplot.sh —

```

#!/usr/bin/sh
dir='/home/mat/Documents'
cd ${dir}
. ${dir}/venv11/bin/activate
python3 ${dir}/rtplot.py
deactivate

```

日付の文字列を第1引数に指定して、コマンドラインから直接実行すると、

```

cd /home/mat/Documents
source venv11/bin/activate

```

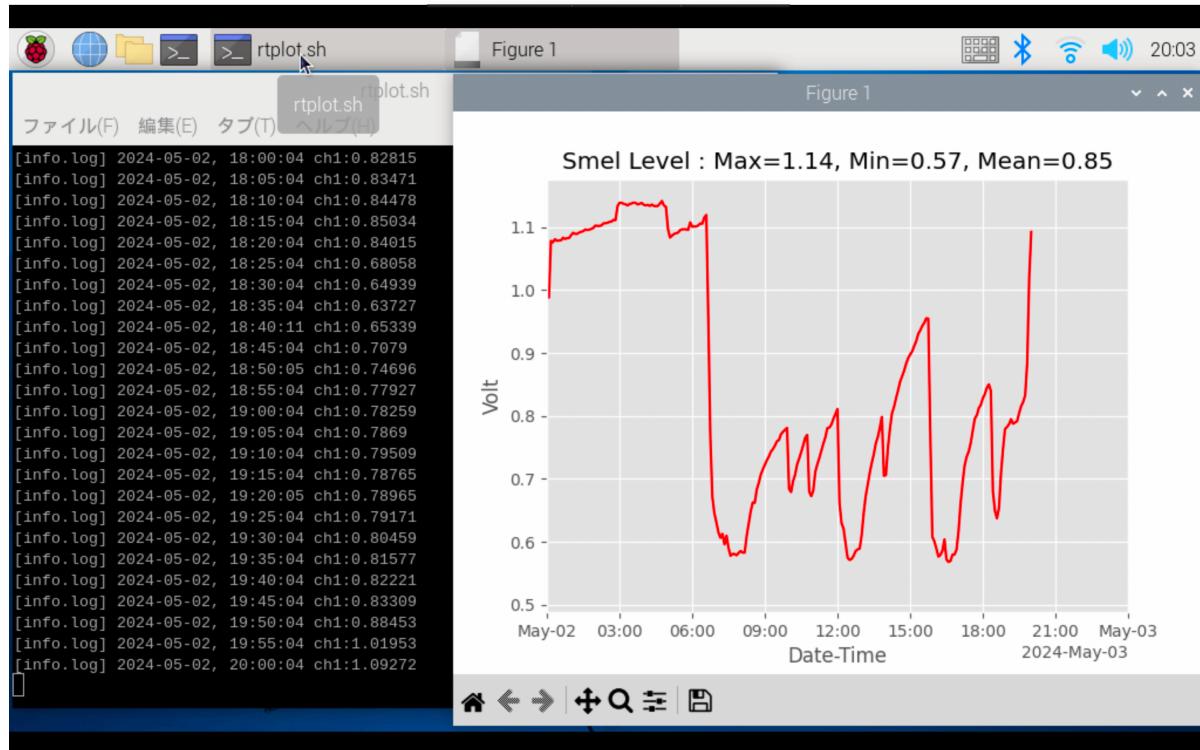


図 2.6 実時間出力（途中経過：5月2日の20時過ぎ）

```
python3 rtplot.py 2024-05-10  
deactivate
```

その日付けのデータに基づく実時間グラフを `figs` フォルダの中に生成する

第3章

リモートにデータを保持する

表 3.1 ここで使用する shell とプログラム

リモート	sensor.sh*	日付文字列とセンサーの読み取り値を並べて出力する shell
リモート	adrsz0D.py*	センサーを読むクラスの定義
リモート	procmain.py*	センサーのインスタンス生成とセンサーの読み取り
リモート	proc.sh	cron により 5 分間隔で起動される。 sensor.sh を起動して w.txt に計測データを追記蓄積する
リモート	procdaily.sh	cron により毎日 23 時 58 分に起動される。1 日分の蓄積データ w.txt を data/ フォルダへ退避させる
ローカル	lackdata.py	data/ フォルダ内に、本日の日付までで保存されていないデータがあったら、その日付の一覧を標準出力へ出力する
ローカル	lack.sh	lackdata.py が output する日付をパイプを介した標準入力で受け、その日付のデータが、ローカルの w.txt、リモートの w.txt、リモートの data/ フォルダ内、ローカルの data/ フォルダ内にないか、すべて探し出してマージし、その日付のファイルとしてローカルの data/ フォルダに保存する

3.1 リモートマシンの処理

リモートとローカルの両方を動かし続けるのが難しい場合に、普段はリモートだけにデータを集めておいて、時々ローカルにリモートのデータを取り込み、それをグラフ化するなどの処理を行う方法をとる運用も考えられる

リモートのコンピュータ内でデータを集める仕組みは、ほぼローカルコンピュータでの処理と同様にして実施する

```
/home/mat/Documents/proc.sh —————
#!/usr/bin/sh
dir='/home/mat/Documents'
bash ${dir}/sensor.sh >> ${dir}/w.txt
```

リモートにも、(ローカルと同様のディレクトリ構成で) /home/mat/Documents/data フォルダを用意しておく

```
/home/mat/Documents/procdaily.sh —————
#!/usr/bin/sh
dir='/home/mat/Documents'
dates=$(date +%Y-%m-%d)
hour=$(date +%H)
minu=$(date +%M)
if [ ${hour} -ge 23 ]; then
  if [ ${minu} -gt 57 ]; then
    cp ${dir}/w.txt ${dir}/bkup.txt
    mv ${dir}/w.txt ${dir}/data/${dates}.txt
    touch ${dir}/w.txt
  fi
fi
```

crontab では、5分ごとの処理を(ローカルマシンの処理時刻とずらすため)、2,7,12,17,... の5分おきに実行される様にした

```
crontab -e —————
MAILTO=""

# m h  dom mon dow   command
2-57/5 * * * * sh /home/mat/Documents/proc.sh
58 23 * * * sh /home/mat/Documents/procdaily.sh
```

3.2 ローカルマシンの処理

Python のプログラムで、ローカルマシン内に欠けている日付のデータファイルを調べる
/home/mat/Documents/lackdata.py

```
from datetime import datetime as dt, timedelta
import glob

s_format = '%Y-%m-%d'
todaystr = (dt.now() + timedelta(days=0)).strftime(s_format)
today = dt.strptime(todaystr, s_format)

dirstr = '/home/mat/Documents'
wlist = glob.glob(dirstr + '/data/*.txt')
dlist = []
for w in wlist:
    dlist.append(dt.strptime(w[len(dirstr)+6:-4], s_format))
dlist.sort(reverse=True) # descending order

ten_days = 10
lacklist = []
for d in range(ten_days):
    day = today - timedelta(days=d)
    if day in dlist:
        continue
    else:
        lacklist.append(day)

for fname in lacklist:
    print(fname.strftime(s_format))
```

このプログラムを実行した段階で、ローカルに無いファイル（名前の日付の部分）が分かったので、ローカルとリモートのマシンからその日付のデータを集めて整理する

```
/home/mat/Documents/lack.sh -----  
#!/usr/bin/sh  
TODAY=$(date +%Y-%m-%d)  
PASS='mypassword'  
USR='mat'  
ADDR='192.168.3.27'  
DIR='/home/mat/Documents'  
cp ${DIR}/w.txt ${DIR}/bkup.txt  
. ${DIR}/venv11/bin/activate  
python3 ${DIR}/lackdata.py | \  
while read LINE; do (  
    grep ^$LINE ${DIR}/bkup.txt > ${DIR}/${LINE}.wrk  
    sshpass -p $PASS ssh -n ${USR}@${ADDR} \  
        "grep ^$LINE ${DIR}/w.txt" >> ${DIR}/${LINE}.wrk  
    CMD=test\ -e\ ${DIR}/data/${LINE}.txt  
    RC=$(sshpass -p $PASS ssh -n ${USR}@${ADDR} ${CMD};echo $?)  
    if [ ${RC} -eq 0 ]; then  
        sshpass -p $PASS ssh -n ${USR}@${ADDR} \  
            "grep ^$LINE ${DIR}/data/${LINE}.txt" >> ${DIR}/${LINE}.wrk  
    fi  
    sort ${DIR}/${LINE}.wrk | uniq - > ${DIR}/${LINE}.wrk2  
    rm ${DIR}/${LINE}.wrk  
    if [ -s ${DIR}/${LINE}.wrk2 ]; then  
        match=$(echo ${LINE} | awk "/^$TODAY/")  
        if [ -n "$match" ]; then  
            cp ${DIR}/${LINE}.wrk2 ${DIR}/w.txt  
        else  
            cp ${DIR}/${LINE}.wrk2 ${DIR}/data/${LINE}.txt  
        fi  
    fi  
    rm ${DIR}/${LINE}.wrk2  
) < /dev/null; done  
deactivate
```

第4章

クラウド上にデータを保持する

表 4.1 ここで使用する shell とプログラム

リモート	sensor.sh*	日付文字列とセンサーの読み取り値を並べて出力する shell
リモート	adrsz0D.py*	センサーを読むクラスの定義
リモート	procmain.py*	センサーのインスタンス生成とセンサーの読み取り
リモート	proc.sh*	cron により 5 分間隔で起動される。 sensor.sh を起動して w.txt に計測データを追記蓄積する
リモート	procdaily.sh	cron により毎日 23 時 58 分に起動される。1 日分の蓄積データ w.txt を data/ フォルダへ退避させる。 upcloud.py を起動して本日分の計測データをクラウドにアップロードし、最後に昨日分の計測データを削除する（更新）
リモート	upcloud.py	クラウドに指定日付のデータをアップロードする。
ローカル	mycloud.py	指定した日付のデータをクラウドから読んで標準出力に出力する。 -w のスイッチを付けて起動した場合、指定日付から 1 週間前までのデータをクラウドから読んで標準出力へ出力する。
ローカル	daily.py	標準入力から受け取った計測データをもとに、1 日の値の変化をグラフにする（更新）
ローカル	weekly.py	標準入力から受け取った計測データをもとに、1 週間の値の変化をグラフにする（更新）
ローカル	dayplot.sh	mycloud.py から daily.py へパイプで繋いでいる
ローカル	weekplot.sh	mycloud.py から weekly.py へパイプで繋いでいる
ローカル	myplot.py	daily.py, weekly.py, cloudplot.py から呼び出される（更新）
両方	gcloud.py	GCSWapper クラス (upload0.py と mycloud.py で継承する)
ローカル	cloudplot.sh	cloudplot.py を呼び出している
ローカル	cloudplot.py	指定日から遡って指定日数分のグラフを作図

4.1 リモートマシンの処理

センサのデータを w.txt に蓄積していく処理は、これまでのものと同じ

```
/home/mat/Documents/proc.sh —————
```

```
#!/usr/bin/sh
dir='/home/mat/Documents'
bash ${dir}/sensor.sh >> ${dir}/w.txt
```

procdaily.sh を次の様にする

- (1) 1 日分のデータ w.txt を bkup.txt に退避して
- (2) それを /home/mat/Documents/data/ フォルダ内に、日付をファイル名として保存し
- (3) そのファイルをクラウドへアップロードする
- (4) その日の終わりに、昨日分のデータを削除する

こうして、リモートマシン内には当日集めているデータ w.txt、バックアップの 1 日分のデータ bkup.txt、ファイル名に日付を付けた 1 日分のデータの、3つしか保持していない（日々のデータはクラウド上有る）

```
/home/mat/Documents/procdaily.sh —————
```

```
#!/usr/bin/sh
dir='/home/mat/Documents'
dates=$(date +%Y-%m-%d)
hour=$(date +%H)
minu=$(date +%M)
if [ ${hour} -ge 23 ]; then
  if [ ${minu} -gt 57 ]; then
    cp ${dir}/w.txt ${dir}/bkup.txt
    mv ${dir}/w.txt ${dir}/data/${dates}.txt
    touch ${dir}/w.txt
    python3 ${dir}/upcloud.py ${dates}
    yesterday=$(date --date "${dates} 1 days ago" +%Y-%m-%d)
    rm ${dir}/data/${yesterday}.txt
  fi
fi
```

crontab では、proc.sh を 2,7,12,17,... の 5 分おきに実行する様にし、procdaily.sh は 23 時 58 分に実行する様にしている（proc.sh がその日の最後に実行されるのは、23 時 57 分だから）

```
crontab -e  
MAILTO=""  
# m h dom mon dow command  
2-57/5 * * * * sh /home/mat/Documents/proc.sh  
58 23 * * * sh /home/mat/Documents/procdaily.sh
```

クラウドへ指定のファイルをアップロードする処理 upcloud.py は、ラッパークラス GCSWrapper を継承して、必要なメソッドを呼び出しているだけ（クラウド上にも data/ フォルダを用意して、そこに日付をファイル名とするファイルをアップロードしている）

```
/home/mat/Documents/upcloud.py  
  
from datetime import datetime as dt  
import sys  
from gcloud import GCSWrapper  
  
class cloud(GCSWrapper):  
    def __init__(self, proj_name, bkt_name):  
        super().__init__(proj_name, bkt_name)  
  
    if __name__ == '__main__':  
        project_id = "myprojectid"  
        bucket_name = "mybucketname"  
        CL = cloud(project_id, bucket_name)  
        dirstr = '/home/mat/Documents'  
        path = dirstr + '/data/'  
        datestr = dt.now().strftime('%Y-%m-%d')  
        if len(sys.argv) > 1:  
            datestr = sys.argv[1]  
        CL.upload_file(path+datestr+'.txt', 'data/'+datestr+'.txt')
```

4.2 ローカルマシンの処理

測定データが、日付ごとに名前をつけてクラウド上に保存されているので、ローカルではクラウドからデータを読んできて、日毎のグラフ、あるいは週毎のグラフを適時作図する処理とする

クラウドからファイルをダウンロードしてローカルに保存後、そのファイルを読み込んで処理するのではなく、標準入出力のパイプによる受け渡しの方法をとる方がエレガントだと思う（ローカルに保存されるファイルはない）

4.2.1 日毎のグラフの作図

- (1) mycloud.py でクラウドから指定した日付のデータを読んできて、それを標準出力へ出力する
- (2) mycloud.py の標準出力を、パイプを通して daily.py の標準入力で受け取り、グラフの作図を行う

```
/home/mat/Documents/dayplot.sh —————  
#!/usr/bin/sh  
dir='/home/mat/Documents'  
if [ $# -eq 1 ]; then  
dates=$1  
. ${dir}/venv11/bin/activate  
python3 ${dir}/mycloud.py ${dates} | python3 ${dir}/daily.py  
deactivate  
fi
```

次の様に動作させる（日付を第1引数に指定する）

```
(venv11) mat@raspi:~/Documents $ sh ./dayplot.sh 2024-05-18
```

mycloud.py と daily.py は次の通り（同じ場所に myplot.py と gcloud.py を置いておく必要がある）

```
/home/mat/Documents/mycloud.py —————

from datetime import datetime as dt, timedelta
import sys, pandas as pd
from gcloud import GCSWrapper

class cloud(GCSWrapper):
    def __init__(self, proj_name, bkt_name):
        super().__init__(proj_name, bkt_name)

    def download_as_string(self, gcs_path):
        blob = self._bucket.blob(gcs_path)
        text = blob.download_as_string().decode()
        return text

if __name__ == '__main__':
    project_id = "myprojectid"
    bucket_name = "mybucketname"
    CL = cloud(project_id, bucket_name)
    #CL.show_file_names()
    dirstr = '/home/mat/Documents'
    path = dirstr + '/data/'
    s_format = '%Y-%m-%d'
    datestr = dt.now().strftime(s_format)
    if len(sys.argv) == 3 and sys.argv[1] == '-w':
        today = dt.strptime(sys.argv[2], s_format)
        datestr = (today - timedelta(days=7)).strftime(s_format)
        text = ""
        for i in range(7):
            day = today - timedelta(days=(7-i))
            datestr = day.strftime(s_format)
            try:
                temp = CL.download_as_string('data/' + datestr + '.txt')
            except:
                continue
            text += temp
    else:
        datestr = sys.argv[1]
        text = CL.download_as_string('data/' + datestr + '.txt')
print(text)
```

```
/home/mat/Documents/daily.py —————

from datetime import datetime as dt
import sys, matplotlib.dates as mdates
from myplot import graph_plot

dirstr = '/home/mat/Documents'
today = dt.now().strftime('%Y-%m-%d')
nargv = len(sys.argv)
if nargv==2:
    today = sys.argv[1]

list0=[]
if nargv==2:
    with open(dirstr + "/data/" + today + ".txt") as f:
        for line in f:
            list0.append(line)
else:
    while True:
        try:
            line = input()
            list0.append(line)
        except EOFError:
            break

title = 'Smel : '
fig = graph_plot(list0, title)
#fig.savefig(dirstr + '/figs/' + today + '.png')
```

4.2.2 1週間のグラフの作図

- (1) mycloud.py でクラウドから、指定した日付からさかのぼって 1 週間分のデータを読みてきて、それを標準出力へ出力する
- (2) mycloud.py の標準出力を、パイプを通して weekly.py の標準入力で受け取り、グラフの作図を行う

```
/home/mat/Documents/weekplot.sh -----  
#!/usr/bin/sh  
dir='/home/mat/Documents'  
if [ $# -eq 1 ]; then  
dates=$1  
. ${dir}/venv11/bin/activate  
python3 ${dir}/mycloud.py -w ${dates} | python3 ${dir}/weekly.py  
deactivate  
fi
```

mycloud.py では、-w のスイッチを指定することによって、1 週間分のファイルをクラウドから読み込むことを指示している（ソースプログラムは日毎の処理で使ったものと同じ）

次の様に動作させる（1 週間の最後の日付を第 1 引数に指定する）

```
(venv11) mat@raspi:~/Documents $ sh ./weekplot.sh 2024-05-18
```

weekly.py は次の通り（日毎の処理と同様に、同じ場所に myplot.py と gcloud.py を置いておく必要がある）

```
/home/mat/Documents/weekly.py —————  
import sys, matplotlib.dates as mdates  
from datetime import datetime as dt, timedelta  
from myplot import graph_plot  
  
dirstr = '/home/mat/Documents'  
today = dt.now()  
if len(sys.argv) > 1:  
    today = dt.strptime(sys.argv[1], '%Y-%m-%d')  
startstr = (today - timedelta(days=7)).strftime('%Y-%m-%d')  
datelist, list0 = [], []  
for i in range(7):  
    day = today - timedelta(days=(7-i))  
    datelist.append(day.strftime('%Y-%m-%d'))  
if len(sys.argv)==2:  
    for fname in datelist:  
        try:  
            with open(dirstr + "/data/" + fname + ".txt") as f:  
                for line in f:  
                    list0.append(line)  
            except FileNotFoundError:  
                continue  
        else:  
            while True:  
                try:  
                    line = input()  
                    list0.append(line)  
                except EOFError:  
                    break  
  
    title = 'Smel(W) : '  
    fig = graph_plot(list0, title)  
    #fig.savefig(dirstr + '/figs/' + startstr + '.png')
```

4.3 指定日から遡って指定日数分のグラフを作図する

コマンドラインから動作させる（これで、日毎や週毎のグラフ作成処理は不要になる）

```
例 1 mat@raspi:~/Documents $ sh ./cloudplot.sh
例 2 mat@raspi:~/Documents $ sh ./cloudplot.sh 1
例 3 mat@raspi:~/Documents $ sh ./cloudplot.sh 2
例 4 mat@raspi:~/Documents $ sh ./cloudplot.sh 6 2024-05-10
例 5 mat@raspi:~/Documents $ sh ./cloudplot.sh 7 2024-05-10 2 2
```

例 1 は 実行日の前日の 1 日分のデータをグラフにする

例 2 は 例 1 と同じ

例 3 は 実行日の前日を含めて、その日から遡って 2 日分のデータをグラフ化する

例 4 は 指定日付 2024-05-10 を含めて、その日から遡って 6 日分のデータをグラフ化する

例 5 は 7 日分のデータを 2 行 2 列でグラフ化する（4 日分だけプロットすることになる）

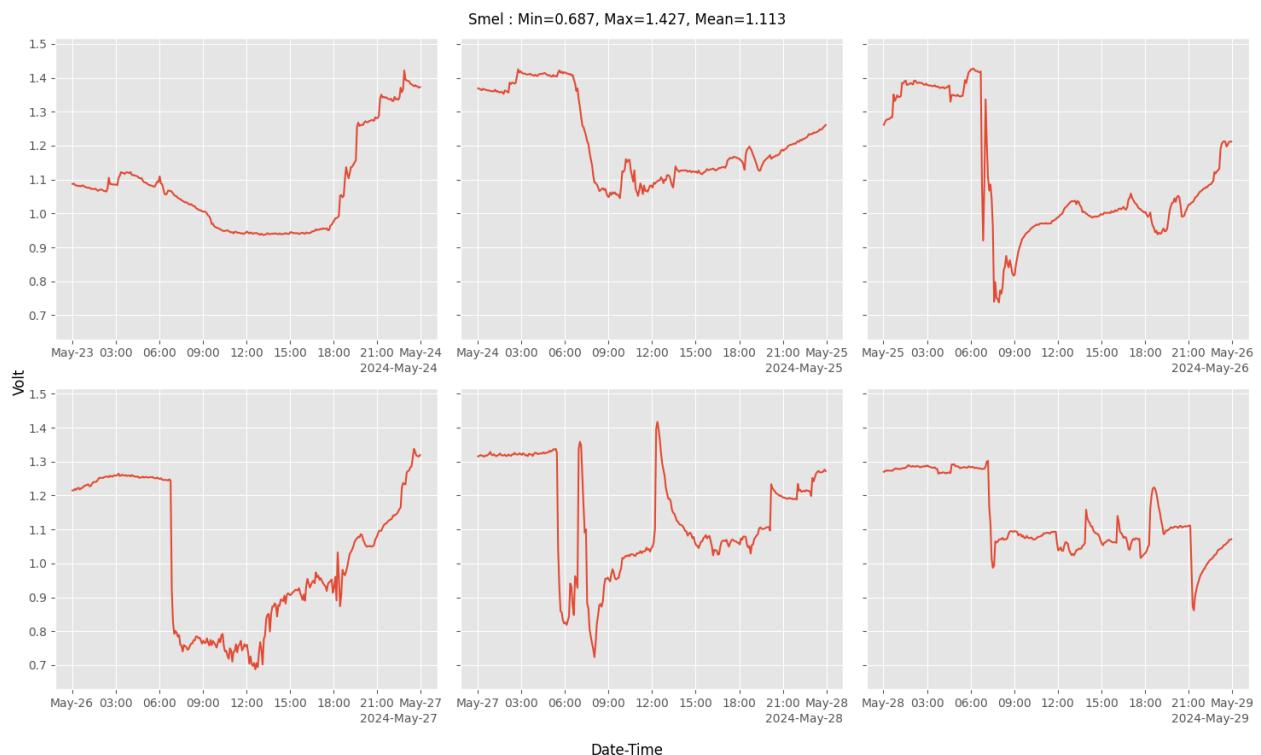


図 4.1 6 日分のグラフ

```
/home/mat/Documents/cloudplot.sh

#!/usr/bin/sh
dir='/home/mat/Documents'
today=$(date +%Y-%m-%d)
dates=$(date --date "${today} 1 days ago" +%Y-%m-%d)
days='1' rows='2' cols='3'
if [ $# -gt 0 ]; then
    days=$1
    #if [ "${days}" -gt 6 ]; then days='6'; fi
    if [ $# -ge 2 ]; then
        if [ $(date -d "$2" +%s) -lt $(date -d "${today}" +%s) ]; then
            dates=$2
        fi
        fi
        if [ $# -ge 3 ]; then
            rows=$3 cols=$4
            #if [ "${rows}" -gt 2 ]; then rows='2'; fi
            #if [ "${cols}" -gt 3 ]; then cols='3'; fi
            fi
        fi
        if [ $# -le 4 ]; then
            . ${dir}/venv11/bin/activate
            python3 ${dir}/cloudplot.py ${dates} ${days} ${rows} ${cols}
            deactivate
        fi
    fi
```

コメントアウトしている3行を生かすと、2行3列の6日分に拘束するようになる

プログラム実行日当日分のデータは、まだクラウド上にない（昨日分までがクラウド上有る）ので、本日以降の日付が指定された場合は、昨日の日付に解釈し直している
 日付は Unix 時間で比較している。date コマンドで一度日付時間を Unix 時間の形に直す（-d あるいは --date オプションによる）Unix 時間として出力するための書式文字には %s を指定する（なお、shell 上での加算減算は、\$((a+b)) などでもできる）

```
[l]/home/mat/Documents/cloudplot.py

from datetime import datetime as dt, timedelta
import sys, matplotlib.dates as mdates
from gcloud import GCSWrapper
from myplot import graph_plot, graph_plots

class cloud(GCSWrapper):
    def __init__(self, prj_id, bkt_name):
        super().__init__(prj_id, bkt_name)

    def download_as_string(self, gcs_path):
        blob = self._bucket.blob(gcs_path)
        text = blob.download_as_string().decode()
        return text

if __name__ == '__main__':
    project_id = "myprojectid"
    bucket_name = "mybucketname"
    CL = cloud(project_id, bucket_name)
    dirstr = '/home/mat/Documents'
    path = dirstr + '/data/'
    s_format = '%Y-%m-%d'
    nargs = len(sys.argv)
    if nargs == 5:
        today = dt.strptime(sys.argv[1], s_format)
        span = int(sys.argv[2])
        if 6 < span:
            span = 6
        if span < 3:
            rows = 1
            cols = span
        else:
            rows = int(sys.argv[3])
            cols = int(sys.argv[4])
            span = rows * cols
    else:
        sys.exit()

    datelist = []
    for i in range(span-1, -1, -1):
        day = today - timedelta(days=i)
        datestr = day.strftime(s_format)
```

```

        datelist.append(datestr)

datalist = []
for datestr in datelist:
    try:
        temp = CL.download_as_string('data/' + datestr + '.txt')
    except:
        continue
    datalist.append(temp)

title = 'Smel : '
if (rows==1 and cols==1):
    fig = graph_plot(datalist, title)
else:
    fig = graph_plots(datalist, title, rows, cols)
fig.savefig(dirstr + '/figs/' + 'savefig' + '.png')

```

myplot.py はこれまでのものを書き換えている

graph_plot() は 1 つのグラフ用紙に描画するが、graph_plots() は nrows=rows 行 ncols=cols 列のグラフ用紙に描画する

rows=cols=1 の場合、graph_plots() でも 1 つのグラフに作図可能だが、ここまで処理との互換性を維持する観点から graph_plot() も残している

```

[1]/home/mat/Documents/myplot.py

import numpy as np, pandas as pd
import matplotlib.pyplot as plt
import matplotlib.style
import matplotlib.dates as mdates

def chop(daystr):
    x, y = [], []
    for dstr in daystr.split('\n'):
        if len(dstr)>0:
            datev = mdates.datestr2num(dstr[:20])
            x.append(datev)
            value = float(dstr[25:].strip())
            y.append(value)
    return x, y

def graph_plot(datalist, tstr):
    x, y = [], []
    for daystr in datalist:

```

```
x0, y0 = chop(daystr)
x.append(x0)
y.append(y0)
ymax, ymin, yave = max(y), min(y), np.mean(y)
title = tstr +\
f'Min={ymin:0.3f}, Max={ymax:0.3f}, Mean={yave:0.3f}'
matplotlib.style.use('ggplot')
fig, ax = plt.subplots()
locator = mdates.AutoDateLocator()
formatter = mdates.ConciseDateFormatter(locator)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
ax.set_xlim([ymin*0.9, ymax*1.1])
ax.set_xlabel('Date-Time')
ax.set_ylabel('Volt')
ax.set_title(title)
ax.plot(x, y)
plt.show()
return fig

def graph_plots(datalist, tstr, rows, cols):
    ymax, ymin, yave = -20.0, 20.0, 0.0
    x, y = [], []
    for daystr in datalist:
        x0, y0 = chop(daystr)
        x.append(x0)
        y.append(y0)
        maxi, mini = max(y0), min(y0)
        ymax = maxi if (maxi>ymax) else ymax
        ymin = mini if (mini<ymin) else ymin
        yave += np.mean(y0)
    yave = yave / len(datalist)
    title = tstr +\
f'Min={ymin:0.3f}, Max={ymax:0.3f}, Mean={yave:0.3f}'

    matplotlib.style.use('ggplot')
    fig, ax = plt.subplots(nrows=rows, ncols=cols,\n                           squeeze=False, tight_layout=True,\n                           sharex=None, sharey='row', figsize=(15,9))

    k = 0
    for i in range(rows):
        for j in range(cols):
```

```
locator = mdates.AutoDateLocator()
formatter = mdates.ConciseDateFormatter(locator)
ax[i,j].xaxis.set_major_locator(locator)
ax[i,j].xaxis.set_major_formatter(formatter)
ax[i,j].set_ylim([ymin*0.9, ymax*1.1])
if k<len(x):
    ax[i,j].plot(x[k], y[k])
    k+=1
else:
    break
else:
    continue
break

fig.supxlabel('Date-Time')
fig.supylabel('Volt')
fig.suptitle(title)
plt.show()
return fig
```

4.3 指定日から遡って指定日数分のグラフを作図する

45

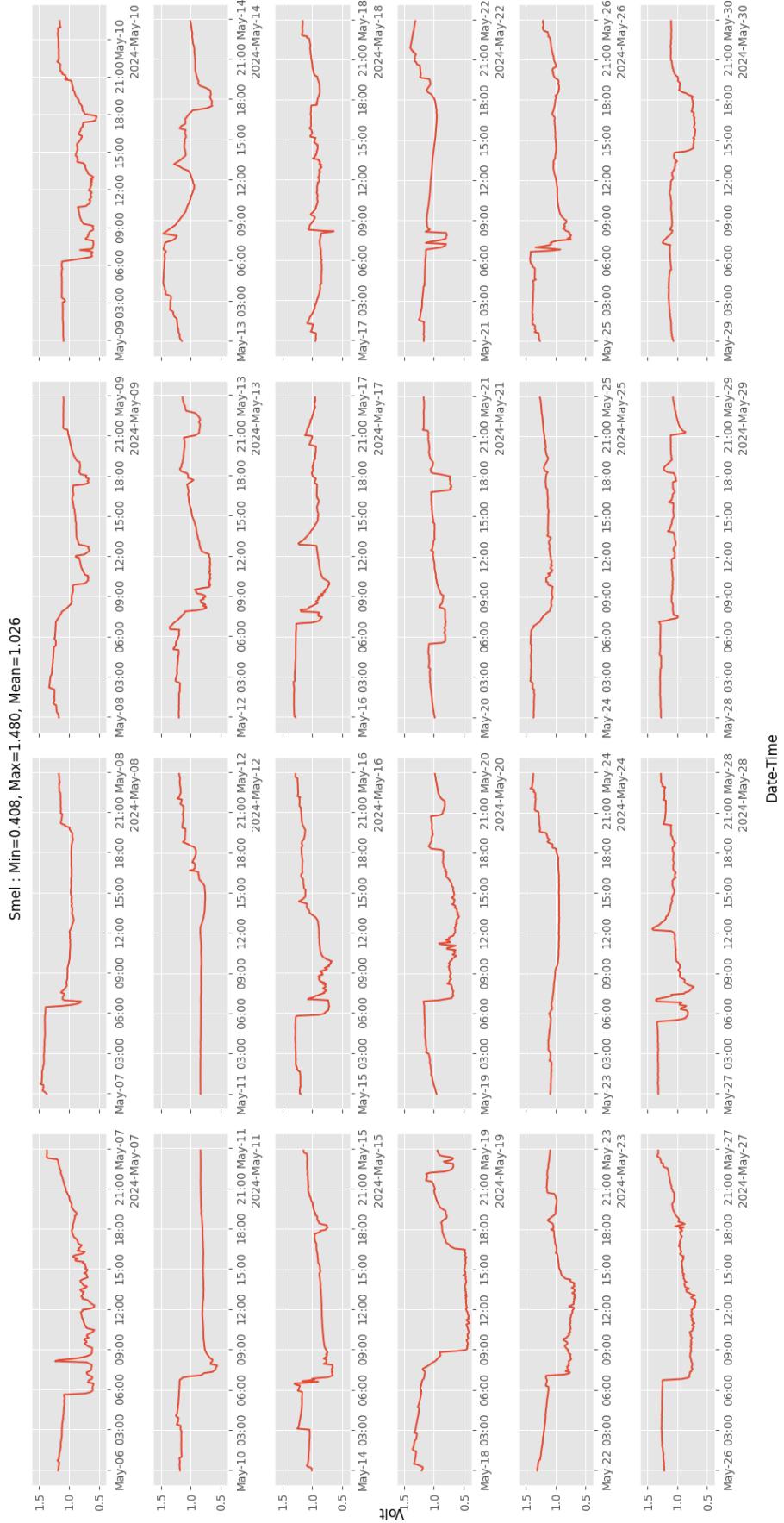


図 4.2 ./cloudplot.sh 24 2024-05-30 6 4

第5章

補遺

5.1 shell を記述する上での注意点

- 次の部分で、\${TODAY} と \${match} はダブルクオートで囲む必要がある

```

TODAY=$(date +%Y-%m-%d)
...
match=$(echo ${LINE} | awk "/^${TODAY}/")
if [ -n "${match}" ]; then

```

shell の変数名をダブルクオートで囲まないと期待した評価がなされない事がある
(理由は以下で) なおシングルクオートで囲むのは、単なる文字列扱いの場合

- コマンドとして実行した結果の文字列が必要なら、アクサングラーブ (バッククオート) でコマンドを囲むか、あるいは\$(command) の形にする。例えば次の \${match} では、TODAY は date コマンドをダブルクオートで囲んでいるため、失敗する (シングルクオートで囲んでも当然の様に失敗する。自明のこと)

```

TODAY="date +%Y-%m-%d"
match=$(echo ${LINE} | awk "/^${TODAY}/")

```

date コマンドを、アクサングラーブで囲むか又は\$(command) の形で記述すると期待通り評価され解決するが、そもそもアクサングラーブ (‘) はシングルクオート (') と見分けがつきにくいので、\$(command) の記述の方がよい (好みの問題かも)

```
TODAY='date +%Y-%m-%d'
```

```

    . . .
match=$(echo ${LINE} | awk "/^${TODAY}/")

#!/usr/bin/sh
dates=$(date +%Y-%m-%d)
datex='date +%Y-%m-%d'
datey="date +%Y-%m-%d"
datez='date +%Y-%m-%d'
echo "${dates}"           2024-05-18
echo "${datex}"           2024-05-18
echo "${datey}"           date +%Y-%m-%d
echo "${datez}"           date +%Y-%m-%d
echo ${dates}              2024-05-18
echo ${datex}              2024-05-18
echo ${datey}              date +%Y-%m-%d
echo ${datez}              date +%Y-%m-%d
echo $dates               2024-05-18
echo $datex               2024-05-18
echo $datey               date +%Y-%m-%d
echo $datez               date +%Y-%m-%d
echo '${dates}'          ${dates}
echo '${datex}'          ${datex}
echo '${datey}'          ${datey}
echo '${datez}'          ${datez}
echo '${dates}'          test.sh: 1: 2024-05-18: not found
echo '${datex}'          test.sh: 1: 2024-05-18: not found
echo '${datey}'          2024-05-18
echo '${datez}'          2024-05-18

```

図 5.1 実験：test.sh

図 5.2 test.sh の実行結果

- if 文の -n は、"\${match}"が「空でないならば」の意味（「空ならば」は、-z）
マッチした場合は、マッチした文字列が"\${match}"に入ってくるので空ではない
- ssh で送る前に、予め CMD 変数に \${LINE} と \${DIR} の評価を済ませている

```
CMD=test\ -e\ ${DIR}/data/${LINE}.txt  
RC=$(sshpass -p ${PASS} ssh ${USR}@${ADDR} ${CMD};echo $?)
```

- 「test -e <file_path>」は、「test -f <file_path>」でも動くかもしれない
- if 文の [-s <file_path>] は、ファイルが存在し、なおかつ中身がある場合に True 、ファイルの中身が空 又は ファイルが存在しない場合には False が返る
- Python プログラムの標準出力を、while read LINE にパイプで渡しているもし、ファイル (FILE_NAME) を介して渡すのであれば、
`cat FILE_NAME | while read LINE; do (...) < /dev/null; done`
- while ループの中で ssh (や rsh) を実行すると、読み込むファイルが複数行あっても、1 行目しか処理されないという現象に遭遇する。
ssh を実行すると標準入力が ssh に振り向けられるため、read で読んだ 1 行のみならずファイル全体が ssh に渡されてしまう。結果として ssh を実行した後にはもう読める行がない事になり、while ループは 1 回で終了してしまうことが起こる。
これを防ぐには、ssh に -n オプションを付けて、標準入力をリダイレクトするのではなく、/dev/null をリダイレクトする様に指示する必要がある。(なお、この辺りの事情は rsh コマンドでも同じ)
- リモートの sensor.sh だけはプロセス置換を実施できる bash によって動作させる
- 「source」コマンド、あるいは「.」コマンド^{*1}は、続いて記述するファイル (コマンド) を、現在の shell 環境下で実行する時に使用する。(例えば、.bashrc を編集して、その編集内容を再 login せずに有効にしたい場合に、「source .bashrc」あるいは「. .bashrc」などとして即座に反映させることができる。)
一方、bash によって shell を起動する場合 (あるいは、shell スクリプト名の指定で直接起動する場合) は、現在の shell 環境とは異なる環境下 (異なる shell 変数や環境変数) の別プロセスとして実行することになる。
(shell 変数は現在実行中の shell でのみ有効な変数、環境変数は実行されたプログラム (子プロセス) に引き継がれる変数。どちらも shell で使用可能。export コマンドを使うと、shell 変数を環境変数に変換できる)
- bash では、「source ファイル名」と全く同じ処理を「. ファイル名」によって実行できるが、bash の基になっている sh では、「.」コマンドしか利用できず「source」コマンドは使用できない。

^{*1} <https://atmarkit.itmedia.co.jp/ait/articles/1712/21/news015.html>

5.2 GCS(google-cloud-storage)

ローカルマシンの data フォルダ以下のファイルを、クラウドへ退避させることを考える

1. 予め Google Cloud の Cloud Console (WebUI) で新規に、プロジェクトとバケットを用意しておく必要がある



2. プログラムからアクセスを試みると、アクセス権が無いという類のエラーに見舞われる所以、Google Cloud CLI をインストール^{*2}した後（ここ^{*3}で議論されていることに従って）コマンドコンソールから次を行う（Yes を応答していく）

```
gcloud auth login
gcloud auth application-default login
```

プログラムで必要になるプロジェクト名は、WebUI 上で作成した時の名前ではなく、application-default login の際、最後にコンソール上に表示されるものを使う必要がある

3. ユーザ認証情報とサービスアカウント認証情報を上の手続きで提供できたなら、以下の json ファイルに必要な情報が保存されているので、他のマシンで google-cloud-storage の API を使用したプログラムを動かしたい場合、この json ファイルを同じ場所にコピーして運用することができる

^{*2} <https://cloud.google.com/sdk/docs/install?hl=ja>

^{*3} <https://stackoverflow.com/questions/49302859/gsutil-serviceexception-401-anonymous-caller-does-not-have-storage-objects-list>

```
~/.config/gcloud/application_default_credentials.json
```

もし、勝手な場所に保存したいなら、次の方法^{*7}もある様だ

```
from google.cloud import storage as gcs
from google.oauth2 import service_account
key_path = '{クレデンシャルを格納している Path}'
credential = service_account.Credentials.from_service_account_file(key_path)
project_id = "{ProjectName}"
client = gcs.Client(project_id, credentials=credential)
```

4. 生成された json ファイルを読んでみると次の様になっている

```
{
  "account": "",
  "client_id": "764086051850-6h 途中略 di341hur.apps.googleusercontent.com",
  "client_secret": "d-FL95Q19q7MQmFpd7hHD0Ty",
  "quota_project_id": "superx-yrunner-123456-z2",
  "refresh_token": "1//0eF3EqrARA-9 途中略 3VL1aJ4ki3HIWdT0H6YM-NKQ_rSw",
  "type": "authorized_user",
  "universe_domain": "googleapis.com"
}
```

プロジェクト名として指定するのに使うのは、"quota_project_id"：の後に示されている文字列になる

json ファイルを読み書きする Python のプログラムは次の通り

^{*7} <https://qiita.com/Hyperion13fleet/items/594c15ac24f149ab73c9>

```
myjson.py —————  
import json  
  
#JSON ファイルの読み込み  
with open('application_default_credentials.json', 'r') as f:  
    json_dict = json.load(f)  
    print('json_dict:{}' .format(type(json_dict)))  
    print(f'json_dict:\n{json_dict}')  
  
#JSON データの変換  
print('----辞書型から JSON 形式の文字列へ変換----')  
    json_str = json.dumps(json_dict)  
    print('json_str:{}' .format(type(json_str)))  
    print(f'json_str:\n{json_str}')  
  
print('----JSON 形式の文字列から辞書型へ変換----')  
    json_dict2 = json.loads(json_str)  
    print('json_dict2:{}' .format(type(json_dict2)))  
    print(f'json_dict2:\n{json_dict2}')  
  
#JSON データの書き込み  
with open('test2.json', 'w') as f2:  
    json.dump(json_dict2, f2)  
with open('test2.json', 'r') as f3:  
    f3_dict = json.load(f3)  
    print(f'f3_dict:\n{f3_dict}' )
```

5. Python の仮想環境に次を導入しておく

```
pip install google-cloud-storage  
pip install gcsfs  
(以下は必要に応じて)  
pip install pillow  
pip install openpyxl
```

6. GCS(google-cloud-storage) の PythonAPI は、以下のラッパークラス⁷を継承して利用させて頂くことにする（ありがたや）

5.1 GCSWrapper (gcloud.py)

```
1 import pandas as pd
2 from google.cloud import storage as gcs
3 from io import BytesIO
4
5 class GCSWrapper:
6     def __init__(self, project_id, bucket_id):
7         """GCSのラッパークラス
8         Arguments:
9             project_id {str} -- GoogleCloudPlatform Project ID
10            bucket_id {str} -- GoogleCloudStorage Bucket ID
11        """
12        self._project_id = project_id
13        self._bucket_id = bucket_id
14        self._client = gcs.Client(project_id)
15        self._bucket = self._client.get_bucket(self._bucket_id)
16
17    def show_bucket_names(self):
18        """バケット名の一覧を表示
19        """
20        [print(bucket.name) for bucket in self._client.list_buckets()]
21
22    def show_file_names(self):
23        """バケット内のファイル一覧を表示
24        """
25        [print(file.name) for file in self._client.list_blobs(self._bucket)]
26
27    def upload_file(self, local_path, gcs_path):
28        """GCSにローカルファイルをアップロード
29
30        Arguments:
31            local_path {str} -- local file path
32            gcs_path {str} -- gcs file path
33        """
34        blob = self._bucket.blob(gcs_path)
35        blob.upload_from_filename(local_path)
36
37    def upload_file_as_dataframe(self, df, gcs_path, flg_index=False, flg_header=True):
38        """GCSにpd.DataFrameをCSVとしてアップロード
39
40        Arguments:
41            df {pd.DataFrame} -- DataFrame for upload
42            gcs_path {str} -- gcs file path
43
44        Keyword Arguments:
45            flg_index {bool} -- DataFrame index flg (default: {False})
46            flg_header {bool} -- DataFrame header flg (default: {True})
47        """
48        blob = self._bucket.blob(gcs_path)
49        blob.upload_from_string(df.to_csv(
50            index=flg_index, header=flg_header, sep=","))
51
52    def download_file(self, local_path, gcs_path):
53        """GCSのファイルをファイルとしてダウンロード
54
55        Arguments:
56            local_path {str} -- local file path
57            gcs_path {str} -- gcs file path
58        """
59        blob = self._bucket.blob(gcs_path)
60        blob.download_to_filename(local_path)
61
62    def download_file_as_dataframe(self, gcs_csv_path):
63        """GCSのファイルをpd.DataFrameとしてダウンロード
64
65        Arguments:
66            gcs_csv_path {str} -- gcs file path (only csv file)
```

```
67
68     Returns:
69         [pd.DataFrame] -- csv data as pd.DataFrame
70     """
71     blob = self._bucket.blob(gcs_csv_path)
72     content = blob.download_as_string()
73     df = pd.read_csv(BytesIO(content))
74     return df
```

このラッパークラスは、リモートマシンの日毎のデータを GCS にアップロードする際にも使っていた

以下は、GCS の操作^{*8}を試行したもの。試しているのは、

- テキストファイルのアップロードとダウンロード
- pandas のデータフレームのアップロードとダウンロード
- CSV ファイルのアップロードとダウンロード
- GCS 上のファイルの削除とファイル一覧の取得

なお、例えば data/ という名前の blob をアップロードするとフォルダができる

^{*8} <https://cloud.google.com/storage/docs/samples/?hl=ja>

5.2 check_cloud.py

```
1 import pandas as pd
2 from gcloud import GCSWrapper
3
4 class cloud(GCSWrapper):
5     def __init__(self, prj_id, bkt_name):
6         super().__init__(prj_id, bkt_name)
7         self.x = []
8         self.y = []
9
10    def delete_blob(self, blob_name):
11        """Deletes a blob from the bucket."""
12        # bucket_name = "your-bucket-name"
13        # blob_name = "your-object-name-with-gcs-path"
14        blob = self._bucket.blob(blob_name)
15        generation_match_precondition = None
16        # Optional: set a generation-match precondition to avoid potential race
17        # conditions
18        # and data corruptions. The request to delete is aborted if the object's
19        # generation number does not match your precondition.
20        blob.reload()
21        # Fetch blob metadata to use in generation_match_precondition.
22        generation_match_precondition = blob.generation
23        blob.delete(if_generation_match=generation_match_precondition)
24        print(f"Blob_{blob_name}_deleted.")
25
26    def readdata(self, fname):
27        try:
28            with open(fname, 'r') as f:
29                for line in f:
30                    self.chop(line)
31        except FileNotFoundError:
32            print(f'File_{fname}_not_found')
33
34    def chop(self, line):
35        datestr = line[:20].strip()
36        #dateval = mdates.datestr2num(datestr)
37        value = float(line[25:].strip())
38        self.x.append(datestr)
39        self.y.append(value)
40
41    def txt2df(self, fname):
42        self.readdata(fname)
43        self.df = pd.DataFrame({"mdate": self.x, "value": self.y})
44        return self.df
45
46 if __name__ == '__main__':
47     project_id = "myprojectid"
48     bucket_name = "mybucketname"
49     CL = cloud(project_id, bucket_name)
50     CL.show_file_names()
51     dirstr = '/home/mat/Documents'
52     path = dirstr + '/data/'
53     datestr = '2024-05-10'
54     CL.upload_file(path+datestr+'.txt', 'data/'+datestr+'.txt')
55     CL.download_file(dirstr+'/'+datestr+'.txt', 'data//'+datestr+'.txt')
56     df = CL.txt2df(dirstr+'/'+datestr+'.txt')
57     df.to_csv('gs://'+bucket_name+'/data//'+datestr+'.csv', index=False)
58     CL.show_file_names()
59     data_df = pd.read_csv('gs://'+bucket_name+'/data//'+datestr+'.csv')
60     print(data_df)
61     CL.delete_blob('data//'+datestr+'.csv')
62     CL.show_file_names()
```

5.3 新たなシステムへの移行の手順

5.3.1 ローカルマシンの移行

ここまで作成してきたものを、新たなマシン（Raspberry Pi）へ移行する際、必要になる手続きについてまとめておく（新しいマシンのOSが導入され、ネットワークの設定を終えた後の手続きになる）

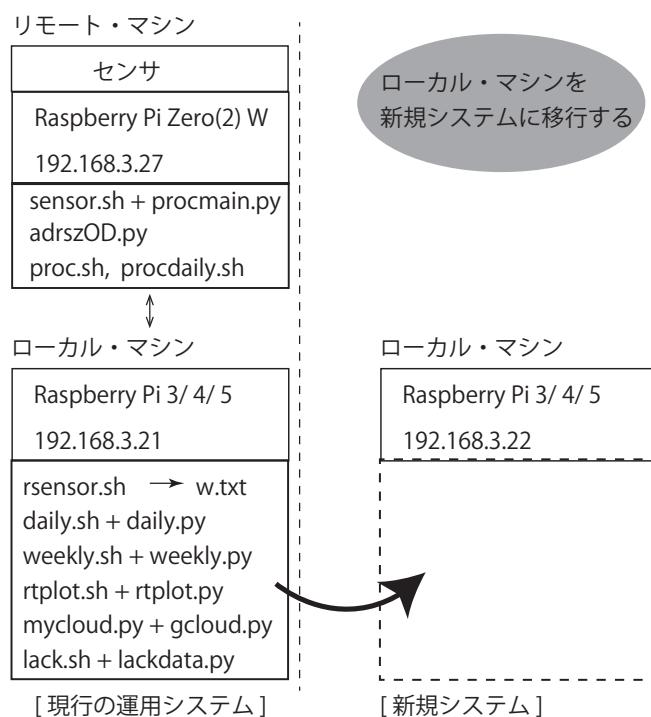


図 5.3 ローカルマシンの移行

以下の手続きはその都度全て、/home/mat/Documents の直下で実施するものとする

1. ssh を有効にする（Raspberry Pi の configuration で）
2. システムを最新の状態にする

```
sudo apt update
sudo apt dist-upgrade
```

3. sshpass を導入する

```
sudo apt install -y sshpass
```

4. Python の仮想環境を /home/mat/Documents/venv11 につくる⁹

```
python3 -m venv venv11
source venv11/bin/activate
python3 -m pip install -U pip
pip install numpy==1.22.4
pip install scipy
pip install pandas==1.4.2
pip install matplotlib
pip install scikit-learn
pip list -o
deactivate
```

5. data フォルダと figs フォルダを移行する

```
scp -r mat@192.168.3.21:/home/mat/Documents/data ./
scp -r mat@102.168.3.21:/home/mat/Documents/figs ./
```

6. shell と Python のプログラムを移行する

```
scp mat@192.168.3.21:/home/mat/Documents/*.sh ./
scp mat@192.168.3.21:/home/mat/Documents/*.py ./
```

ここで、各 shell には実行可能属性がついている事を確認する ls -l *.sh

もし付いていなかったら付ける chmod +x *.sh

また、リモートマシンの IP アドレス、ユーザ名とそのパスワードが変わった場合は、

ここで rsensor.sh の当該箇所（露に記述している）を編集する

Python のプログラムに実行可能属性は不要である

7. cron に時刻指定手続きを登録する (crontab -e)

```
MAILTO=""
*/5 * * * * sh /home/mat/Documents/rsensor.sh
56 23 * * * sh /home/mat/Documents/daily.sh
```

⁹ 寺田学 他 3 名、翔泳社「Python によるあたらしいデータ分析の教科書」第 2 版、p.50

```
1 0 * * Sun sh /home/mat/Documents/weekly.sh
```

8. ssh でリモートマシンに接続を試みて、key fingerprint を登録させる

```
ssh mat@192.168.3.27
```

```
The authenticity of host '192.168.3.27' can't be established.  
ED25519 key fingerprint is SHA256:.....  
This key is not known by any other names.  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
ssh mat@192.168.3.27 /home/mat/Documents/sensor.sh
```

9. txt ファイルを移行する

```
scp mat@192.168.3.21:/home/mat/Documents/*.txt ./
```

10. rsensor.sh が 5 分ごとに起動され、w.txt を更新している事を確認する

```
tail -f w.txt
```

11. 実時間モニタ用のプログラムを起動する

```
source venv11/bin/activate  
python3 ./rtplot.py
```

5.3.2 リモートマシンの移行

新しいリモートマシンへの移行は次の通り

リモートマシンへの OS の導入とネットワークの設定を終えていること

1. configuration で、ssh と i2c を有効にする
2. 最新の状態に更新する

```
sudo apt update  
sudo apt dist-upgrade
```

3. adrszOD.py と procmain.py、及び proc.sh、procdaily.sh を導入する
4. sensor.sh を導入して、実行可能属性をつける (chmod +x sensor.sh)
5. /home/mat/Documents/sensor.sh を実行して、現在日時とセンサの電圧、それぞれ適切な値が 1 行で出力される事を確認する

参考文献

- [1] ビット・トレード・ワン社 zeroone シリーズ拡張基板のサンプルプログラム
(<https://github.com/bit-trade-one/RasPi-Zero-One-Series>)
- [2] BME280 センサーモジュール
(<https://algorithm.joho.info/programming/python/raspberrypi3-bme280-kion-sitsudo-kiatsu/>)
- [3] matplotlib が認識する日付の書式
(https://matplotlib.org/stable/api/dates_api.html)
- [4] テキストの結合、プロセス置換
Daniel J.Barrett 著、オライリー・ジャパン「Efficient Linux コマンドライン」初版第1刷、p.106（テキストの結合）, p.154, p.157 のコラム記事
- [5] Python の仮想環境
寺田学 他著、翔泳社「Python によるあたらしいデータ分析の教科書」第 2 版、p.50
- [6] 「source」コマンドと「.」コマンド
(<https://atmarkit.itmedia.co.jp/ait/articles/1712/21/news015.html>)
- [7] Google Cloud Console -> Cloud Storage
(<https://console.cloud.google.com/welcome/new?hl=ja>)
- [8] Wrapper class for Python API of Google Cloud Storage
(<https://qiita.com/Hyperion13fleet/items/594c15ac24f149ab73c9>)
- [9] Google Cloud CLI
(<https://cloud.google.com/sdk/docs/install?hl=ja>)
(<https://stackoverflow.com/questions/49302859/gsutil-serviceexception-401-anonymous-caller-does-not-have-storage-objects-list>)
- [10] Code samples with GCS API
(<https://cloud.google.com/storage/docs/samples/?hl=ja>)
- [11] gcsfs (<https://dodotechno.com/python-gcs/>)