

WXBC 2020年度テクノロジー研修

「メッシュ気象データ分析チャレンジ！」講習テキスト

第3課 モジュール wxbcgrib を用いたGPVデータの処理

Copyright 2021 気象ビジネス推進コンソーシアム

(C) 2021 WXBC

<利用条件>

本書は、本書に記載した要件・技術・方式に関する内容が変更されないこと、および出典を明示いただくことを前提に、無償でその全部または一部を複製、翻案、翻訳、転記、引用、公衆送信等して利用できます。なお、全体を複製、翻案、翻訳された場合は、本書にある著作権表示および利用条件を明示してください。

<免責事項>

本書の著作権者は、本書の記載内容に関して、その正確性、商品性、利用目的への適合性等に関して保証するものではなく、特許権、著作権、その他の権利を侵害していないことを保証するものでもありません。本書の利用により生じた損害について、本書の著作権者は、法律上のいかなる責任も負いません。

この実習をGoogleコラボ上で行なう場合は、最初に以下のコードを実行してください。

In []:

```
# Googleコラボ上で実行する場合
# Google Driveに challenge.zip をアップロードし、このコードを毎回実行する

from google.colab import drive
drive.mount('/content/drive');      # Google Driveをマウントする

import os,sys
if os.path.exists("/content/drive/My Drive/challenge") == False:
    if os.path.exists("/content/drive/My Drive/challenge.zip") == False:
        print ("Google Driveに、challenge.zip をアップロードし、");
        print ("メニューから「ランタイム」→「ランタイムを出荷時設定にリセット」を実行");
        sys.exit();
    !cd "/content/drive/My Drive/" ; unzip challenge.zip          # 解凍

if os.path.exists("/content/drive/My Drive/challenge/wgrib2") == False:
    !wget ftp://ftp.cpc.ncep.noaa.gov/wd51we/wgrib2/wgrib2.tgz
    !tar xvfz wgrib2.tgz
    !export CC=gcc ; export FC=gfortran ; cd wgrib2 ; make          # wgrib2をビルド
    !cp ./grib2/wgrib2/wgrib2 "/content/drive/My Drive/challenge/"

# 必要なモジュールをランタイムに導入
!pip install netCDF4
!apt install proj-bin libproj-dev libgeos-dev
!pip uninstall shapely imgaug -y
!pip install shapely --no-binary shapely
!pip install cartopy

# 使用するファイルやフォルダをGoogle Driveと接続する
!cp "/content/drive/My Drive/challenge/wgrib2" /usr/local/bin/
!chmod +x /usr/local/bin/wgrib2
!mkdir -p c:/wgrib2/ ; ln -s /usr/local/bin/wgrib2 c:/wgrib2/wgrib2
!mkdir -p C:/wgrib2/ ; ln -s /usr/local/bin/wgrib2 C:/wgrib2/wgrib2
!ln -s "/content/drive/My Drive/challenge/jmadata" /content/jmadata
!mkdir "/content/drive/My Drive/challenge/nc" ; ln -s "/content/drive/My Drive/challenge/nc" /content/nc
!ln -s "/content/drive/My Drive/challenge/wxabcgrib.py" /content/wxabcgrib.py
print ("終了");
```

第3課 モジュール wxbcgrib を用いたGPVデータの処理

3-1. モジュール wxbcgribについて

wxbcgribモジュールは、メッシュ形式の気象データを取り扱う際の定型的な処理をひとまとめにして、気象庁のGPVデータを少ない工数で利用できるようにしたモジュールで、気象ビジネス推進コンソ人材育成ワーキンググループの有志が作成したプログラムです。

3-1-1. wxbcgrib の主な機能

wxbcgribは、以下の操作を簡単に実行することができます。

1. GRIBファイルからのデータ読み出し(NetCDFファイルを経由)
2. グリッドの切り直し
3. 任意の時刻における気象分布の取り出し
4. 任意の緯度経度における気象時系列の取り出し

wxbcgribは、気象庁が配信するGPVデータのうち以下の気象プロダクトをサポートします。ただし、アスタリスクを付したプロダクトについては気象要素名に「disc」、単位に「unit」が取得される不具合があります。また、高解像度降水ナウキャストについては、ファイル内に複数の領域のデータが混在しているため、処理すべき領域のデータだけからなるGRIBファイルかNetCDFを作成しておく必要があります。

- 全球スペクトルモデルGPV(GSM-GPV)(日本域)
- メソ数値予報モデルGPV(MSM-GPV)
- 局地気象予測モデルGPV(LFM-GPV)
- 推計気象分布
- メソアンサンブル予報システム（M E P S）G P V
- 毎時大気解析
- 波浪モデルGPV
- 1 kmメッシュ解析雨量／降水短時間予報G P V*
- 降水ナウキャスト*
- 土壌雨量指数*
- 大雨警報(浸水害)・洪水警報の危険度分布（統合版）*
- 高解像度降水ナウキャスト*

3-1-2. wxbcgrib の利用に必要な準備

wxbcgribは、以下の外部モジュールを使用します。

- | | |
|---------------|--|
| 1. numpy | 多次元配列計算用モジュール |
| 2. os | ファイルやディレクトリの操作をするモジュール |
| 3. subprocess | コマンドラインプログラムを実行させるモジュール |
| 4. datetime | 時刻や時間間隔を表現するtimedeltaオブジェクトを利用するためのモジュール |
| 5. scipy | 科学技術計算を行うモジュール |
| 6. glob | ファイルやディレクトリの探索を行うモジュール |
| 7. matplotlib | 描画を行うモジュール |
| 8. netCDF4 | NetCDFファイル操作のためのモジュール |
| 9. cartopy | 図中に海岸線等を埋め込むモジュール |

これらのうち、1 から 7 までのモジュールはよく利用されるモジュールなので、Anaconda をインストールする際同時にインストールされますが、下の2つ、**netCDF** と **cartopy**については、それほど一般的でないのでデフォルトではインストールされません。Pythonプログラムから **import** 文で呼び出されたら実行可能になるように、**Anaconda Navigaor** を用いて追加でインストールしておいてください。

GRIBデータ処理プログラム**wgrib2.exe**はPythonのモジュールではありませんが、GRIBファイルの処理に必須のプログラムなので、**wgrib2.exe** と関連ファイルを **c:\¥wgrib2** に置いてください。

Macの方は、wgrib2をコンパイルして**~/work/grib2/wgrib2/**に置いてください。さらに、wxbcgrib.pyの42行目の行頭の「#」を削除しこの行を有効にしてください。

3-1-3. wxbcgrib のライセンス

wxbcgribは、気象ビジネス推進コンソーシアム人材育成ワーキンググループ内勉強会「気象データ×IT勉強会」が著作権を持ち、以下の条件で使用を認めます。なお、この条件は、「MITライセンス」と呼ばれているものと同一です。

Copyright (c) 2021 気象データ×IT勉強会

以下に定める条件に従い、本ソフトウェアおよび関連文書のファイル（以下「ソフトウェア」）の複製を取得するすべての人に対し、ソフトウェアを無制限に扱うことを無償で許可します。これには、ソフトウェアの複製を使用、複写、変更、結合、掲載、頒布、サブライセンス、および/または販売する権利、およびソフトウェアを提供する相手に同じことを許可する権利も無制限に含まれます。

上記の著作権表示および本許諾表示を、ソフトウェアのすべての複製または重要な部分に記載するものとします。

ソフトウェアは「現状のまま」で、明示であるか暗黙であるかを問わず、何らの保証もなく提供されます。ここでいう保証とは、商品性、特定の目的への適合性、および権利非侵害についての保証も含みますが、それに限定されるものではありません。作者または著作権者は、契約行為、不法行為、またはそれ以外であろうと、ソフトウェアに起因または関連し、あるいはソフトウェアの使用またはその他の扱いによって生じる一切の請求、損害、その他の義務について何らの責任も負わないものとします。

3-2. GPVデータの読み込み

モジュールwxbcgribをインポートすると、関数 **DS_from_grnc()** が利用できるようになります。この関数は、GRIBファイルからNetCDFファイルへの変換とそこからの読み出しを一気に行う関数で、以下のように用いることでGPVデータを変数 **var** に読み込ませることができます。

```
var = wxbcgrib.DS_from_grnc(path, label)
```

ここで、**path** はGRIBファイルのパス名(置かれているディレクトリ名とファイル名の両方)、**label**は気象要素を示すラベルを示します。第二課で、wgrib2によりGRIBファイルから取り出される気象要素のラベルとNetCDFに変換されたファイルから取り出される気象要素のラベルとが異なることを学習しましたが、この関数で用いるのは後者のラベルですので注意してください。

GRIBファイル **Z_C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-15_grib2.bin** は、2018年7月1日06UTCを初期値とする気象庁メソ数値予報モデルGPVのファイルです。このファイルから、地上気温データ(ラベルは **TMP_1D5mabovground**)を取り出して変数 **var** に格納するスクリプトは、以下のように記述します。

このスクリプトを実行(Ctrl+Enterを押)してください(見た目何も起きませんが、それで構いません)。

```
In [ ]: import wxbcgrib as wg

# MSM-GPVデータの読み込み
grdir = "./jmadata/msm/2018/201807/"
grfile = "Z_C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-15_grib2.bin"
grpath = grdir + "/" + grfile
var = wg.DS_from_grnc(grpath, "TMP_1D5maboveground")
```

気象要素を特定するためのラベルがわからなくなったときは、少々荒っぽいですが、適当な文字列を与えてみてください。エラーメッセージと共に、そのGPVデータにおいて有効なラベルが表示されます。

```
In [ ]: wg.DS_from_grnc(grpath, "hogehoge")
```

関数 **DS_from_grnc()** は、キーワード変数 **lalomima** を追加することでデータを読み込む地域を限定することができます。この時には、読み込む範囲の緯度と経度を用いて、次のようにリストで与えます。このキーワード変数は省略することができ、その場合はすべてのグリッドが読み込まれます。

var = wxbcgrib.DS_from_grnc(path, label, lalomima=[緯度下限, 緯度上限, 経度下限, 経度上限])

返されるのは特殊な形式の変数で、この変数一つに以下に示す7種類のデータが格納されています。

1. 気象要素の名称 : name
2. 気象要素の単位 : unit
3. 無効を示す数値 : _FillValue)
4. データ本体 : data
5. 時刻 : time
6. 緯度 : lat
7. 経度 : lon

変数varからそれぞれのデータを取り出すには、変数名とコロンの後のキーワードをピリオドで繋げます。例えば、気象要素名を取り出すには、**var.name**とします。

下のスクリプトの属性の部分をいろいろと書き換えて実行し、取り出されるデータを確認してください。

```
In [ ]: var.name
```

コンピュータでデータを処理する場合、このように、目的とする処理に適したデータの形式とその形式ならではのデータ操作をあらかじめ定義しておき、これを前提にプログラミングすると、すっきりとした表現でスクリプトを記述することができます。目的とする処理に適したデータの形式とその形式ならではのデータ操作のコレクションのことをプログラミングの世界では**クラス**と呼んでいます。モジュールwxbcgribは、GPVデータの取り扱いを容易にするために、クラス **DataSet** を提供しています。7種類の情報を一纏めにする上の変数 **var** は、クラス **DataSet** の変数です。一般にクラスの変数のことをオブジェクトと呼びます。

3-3. 読み込まれたデータの確認

クラス **DataSet** の変数に保持されたグリッドデータは、メソッド **ql()** により容易に図化することができます。メソッドとは、オブジェクト(クラスの変数)に保持されている情報に対して何がしかの処理をするプログラムのことです。

以下のスクリプトを実行してください。

```
In [ ]: var.ql()
```

先に、クラスの変数 **var** に保持された情報を取り出すのに、変数名と情報名をピリオドでつなぎました。メソッドの場合も変数名とメソッド名をピリオドでつなぎますが、メソッドの場合は、変数が保持する情報を処理して結果を作り出すものなので、メソッド名の後ろに必ず括弧「()」を付けなければなりません。この括弧は、形式的に必要とされることもありますが、多くの場合、その中に何かを書くと、それに応じて一捻りした結果を返します。

メソッド **ql()** は、括弧の中に何も書かないかゼロを指定すると時刻が一番早いデータの分布図を表示します。時間的により後の分布図を表示させたいときには、より大きな整数を括弧の中に与えます。特に、マイナス1を与えると最後の時刻の分布図を表示します。

```
In [ ]: var.ql(-1)
```

3-4. グリッドの切り直し(リグリッド)

GPVデータはそれぞれに異なったグリッドサイズと範囲で作られているので、相互に比較したり演算したりするためにはグリッドを一致させる必要があります。DataSetオブジェクトには、このためのメソッドが二種類用意されています。

1. **lfm()** : 局地気象予測モデルGPV(LFM-GPV)のグリッドに変換します。
2. **tap()** : 推計気象分布のグリッドに変換します。

以下のスクリプトは、MSM-GPVデータから気温を読み込み、推計気象分布のグリッドに変換するものです。実行して結果を比較してください。変換前と変換後では、絵はほぼ同じですが外挿部分が不自然になっているほか、グリッドの数が異なっていることが確認できます。

この処理には少し時間がかかります。下のセル左の「****In []****」の大括弧の中にアスタリスク「*****」が表示されている間は計算中ですので、それが数字にかわるまで少し待ってください。

```
In [ ]: import wxbcgrib as wg

grdir = "./jmadata/msm/2018/201807/"
grfile = "Z_C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-15_grib2.bin"
grpath = grdir + "/" + grfile
var = wg.DS_from_grnc(grpath, "TMP_1D5maboveground")
var.ql()
tatap = var.tap()
tatap.ql()
```

メッシュ？それともグリッド？

ここで、「GPV」と「メッシュデータ」の概念の違いを説明します。GPVは、これまで見てきたように、縦の格子と横の格子の交わった点における値の集合で事物の分布を近似しようとする考え方です。一方メッシュデータとは、連続的に分布する事物を、縦の格子と横の格子で区切られる矩形の範囲で計量して一つの値とし、その集合により事物の分布を表現しようとする考え方です。例えば、人口の分布はメッシュデータとしては表現できますが、GPVでは表現できません。一方、気温などは、矩形範囲で平均した代表的な気温と定義すればメッシュデータとしても表現できるので、格子点データとしてもメッシュデータとしても取り扱うことができます。気象庁の推計気象分布は、あくまでグリッドデータですが、その格子点は、総務省によって定められた「統計に用いる標準地域メッシュ」の第3次地域区画(いわゆる3次メッシュ)の中心点となるように設定されています。

3-5. 任意の時刻における気象分布の取り出し

メソッド **yx()** は、DataSetオブジェクト(**dso**)から、引数で指定した日時の気象値を2次元配列の形式で取り出します。この際、日付はdatetimeオブジェクトで与えてください。オプションとして分布図を画像として出力することが可能です。

書式：

ret = dso.yx(time,fig=False) または、 **dso.yx(time,fig=False)**

引数：

1. time(必須)：取得したい時刻 Python datetimeオブジェクトで指定する。
2. fig(省略可)：Trueを与えると分布図をpngファイルで出力する。デフォルトではFalseが与えられる(すなわち図は作成されない)。
3. prefix(省略可)：fig=Trueのとき有効。出力されるファイル名は、デフォルトで「Tyyyy.mm.dd.HH.MM.png」となるが、prefixに文字列を指定すると、その文字列がデフォルトファイル名のさらに前に付加される。なお、yyyy、mm、dd、HH、MMは、順に、年、月、日、時、分を示す数字である。
4. cmapstr(省略可)：カラーマップを指定（詳細は後述）
5. minmax(省略可)：カラースケールを指定（[最小値, 最大値]のリスト型）

戻り値：

- 省略可。指定した時刻の気象分布データを数値として取り出したいときに変数で受ける。

カラーマップについて：

- カラーマップには名称があるのでこれを文字列で(「」で囲んで)指定する（例：レインボーカラーは"rainbow"、黄色-オレンジ-赤の順で変化は"YlOrRd"など）。色の順序を反転させたい場合は、rainbow_rのような名称の後ろに「_r」を付加する。詳細は下記URLを参照。

■ http://matplotlib.org/examples/color/colormaps_reference.html

指定した日時に丁度一致するデータが存在しない場合、メソッド **yx()** は、前後の格子点

値から線形補間により指定された日時の値を求めます(補間には非常に時間がかかる場合があります)。

下のスクリプトは、オブジェクト変数 **var** から2018年7月1日 20:00JSTの気象データを取り出して配列mapに渡すとともに、分布図を出力させるものです。ここで、関数datetime()

In []:

```
from datetime import datetime

import wxbcgrib as wg
grdir = "./jmadata/msm/2018/201807/"
grfile = "Z__C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-15_grib2.bin"
grpath = grdir + "/" + grfile
var = wg.DS_from_grnc(grpath, "TMP_1D5maboveground")

time = datetime(2018, 7, 1, 20, 00)
map = var.yx(time, fig=True)
```

3-6. 任意の緯度経度における気象時系列の取り出し

メソッド **ts()** は、引数で指定した緯度経度の気象値を、DataSetオブジェクト(**dso**)から1次元配列の形式で取り出します。オプションとして時系列グラフを画像として出力することが可能です。

書式：**ret = dso.ts(lat, lon, fig=False, csv=False)** または、**dso.ts(lat, lon, fig=False, csv=False)**

引数：

1. lat(必須)：取得したい地点の緯度(十進数)
2. lon(必須)：取得したい地点の経度(十進数)
3. fig(省略可)：Trueを与えると分布図をpngファイルに出力する。デフォルトではFalseが与えられる(すなわち図は作成されない)。
4. csv(省略可)：Trueを与えるとデータをCSVファイルに出力する。デフォルトではFalseが与えられる(すなわちファイルは出力されない)。
5. prefix(省略可)：fig=Trueのとき有効となり、出力される画像ファイル名に追加の文字列を追加する。ファイル名のデフォルトは「Nlat-Elon.png」でありこれの先頭にprefixで指定された文字列が付加される。なお、lat、lonはそれぞれ緯度と経度を示す数字である。

戻り値：

- 省略可。指定した時系列データを数値として取り出したいときは戻り値を変数で受ける。

指定した緯度経度が、格子点にピタリと一致することは普通ありません。メソッド **ts()** は、周囲の格子点値から補間により指定された緯度経度の値を求めます。

以下のスクリプトは、先ほど変数varに読み込んだ気温予測データから、北緯36.0566度、東経140.125度(茨城県つくば市)における値を推定して時系列データとして取り出すものです。これらを実行し、それぞれの書式における動作を確認してください。


```
In [ ]: ta = var.ts(36.0566,140.125)
        print(ta)
```

```
In [ ]: ta = var.ts(36.0566,140.125,fig=True)
```

3-7. 複数のGRIBファイルで構成されるGPVデータの読み込み

GPVデータのいくつかは、一つのデータが複数のGRIBファイルで構成されています。たとえば、例として取り扱っている2018年7月1日 06UTCのMSM-GPV地上気象データは、実際には以下の3ファイルで構成されています。

- Z_C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-15_grib2.bin
- Z_C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH16-33_grib2.bin
- Z_C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH34-39_grib2.bin

関数 **DS_from_grnc(path,label)** は、**path**に、単一のパス名だけでなく、リストやワイルドカードを用いた表現も受け付け、それらを結合したDataSetオブジェクトを生成します。

下のスクリプトは、上に示す3つのファイルを**ワイルドカード**で指定して読み込み、北緯36.0566度、東経140.125の地点の気温データを取り出して時間変化グラフにするものです。実行して動作を確かめてください。

```
In [ ]: import wxbcgrib as wg

grdir = "./jmadata/msm/2018/201807/"
grfile = "Z__C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH*_grib2.bin"
grpath = grdir + "/" + grfile
var = wg.DS_from_grnc(grpath,"TMP_1D5maboveground")
var.ts(36.0566,140.125,fig=True, prefix="msm_", csv=True)
```

今度は、ファイル名を**リスト**で与える場合を実習しましょう。推計気象分布のGPVデータを複数読み込んで、上の例で示したメソ数値予報モデルGPVと同じ期間(2018年7月1日15時JST～3日06時JST)について気温の時系列データと取り出して比較します。

実際に読み出す前に、まずはファイル名のリストの作り方を勉強しておきましょう。実際にファイル名を文字列で書き出すと下のようなスクリプトになります。

```
In [ ]: grfilelist = ["Z__C_RJTD_20180701060000_OBS_GPV_Rjp_Ggis1km_Ptt_A20180701060000",
                    "Z__C_RJTD_20180701070000_OBS_GPV_Rjp_Ggis1km_Ptt_A20180701070000",
                    "Z__C_RJTD_20180701080000_OBS_GPV_Rjp_Ggis1km_Ptt_A20180701080000",
                    "Z__C_RJTD_20180701090000_OBS_GPV_Rjp_Ggis1km_Ptt_A20180701090000",
                    "Z__C_RJTD_20180701100000_OBS_GPV_Rjp_Ggis1km_Ptt_A20180701100000",
                    :
                    :
                    "Z__C_RJTD_20180702200000_OBS_GPV_Rjp_Ggis1km_Ptt_A20180702200000"]
```

けれど、40個ものファイル名をスクリプトに書き込むのは大変です。また応用も効きません。せっかくPhytonを使っているのですから、ファイル名のリストをPythonに生成させてみましょう。

まず、日付の文字列のリストを作ります。これは、最初の日付、2018年07月01日06時00分

の、0時間後、1時間後、・・・39時間後の日付の文字列でできています。なので、以下のようなスクリプトになります。

```
In [ ]: from datetime import datetime as dt
        from datetime import timedelta as td

        yymdddhmm0 = dt(2018,7,1,6)
        [(yyydddhmm0 + td(hours=i)).strftime("%Y%m%d%H%M") for i in range(40)]
```

ファイル名は、これら日付を表す文字列が、「Z_CRJTD」などの決まった文字列の中に埋め込まれています。なので、以下のようなスクリプトになります。

```
In [ ]: [f"Z__C_RJTD_{oo}00_OBS_GPV_Rjp_Ggis1km_Ptt_A{oo}_grib2.bin" for oo in yymdddhmm0]
```

それでは、上記を利用した以下のスクリプトを実行して、推計気象分布GPVデータから地上気象の時系列データを取り出してください。

```
In [ ]: grdir = "./jmadata/obs_gpv/2018/201807/"
        yymdddhmm0 = dt(2018,7,1,6)
        yymdddhmm = [(yyydddhmm0 + td(hours=i)).strftime("%Y%m%d%H%M") for i in range(40)]
        grfilelist = [f"Z__C_RJTD_{oo}00_OBS_GPV_Rjp_Ggis1km_Ptt_A{oo}_grib2.bin" for oo in yymdddhmm]
        grpath = [grdir + "/" + grfile for grfile in grfilelist]

        var = wg.DS_from_grnc(grpath, "TMP_surface", lalomima=[35.0, 37.0, 139.0, 140.0],
                               var.ts(36.0566, 140.125, fig=True, prefix="obs_"))
```

3-8 DataSetオブジェクト間の演算

モジュールwxbcgribは、DataSetオブジェクトを用いた演算に関するメソッドは提供していません。オブジェクトから必要なデータを取り出し、それらを演算したり書き直したりした後に、新しいDataSetオブジェクトにまとめてください。関数 **DS_like()** を使うと、既存のDataSetオブジェクトを下敷きにして、新しいDataSetオブジェクトを生成することができます。

書式：**new_dso = DS_like(dso,data=None,name=None,unit=None,_FillValue=None)**

引数：

1. dso(必須)：下敷きとしたいDataSetオブジェクト
2. data(省略可)：データ本体の値をnew_dataに変更したいときは、data=new_dataと指定する。
3. name(省略可)：データの名称をnew_nameに変更したいときは、name=new_nameと指定する。
4. unit(省略可)：データの単位をnew_unitに変更したいときは、unit=new_unitと指定する。
5. _FillValue(省略可)：データの無効値をnew_FillValueに変更したいときは、_FillValue=new_FillValueと指定する。

不快指数の導出を例に、DataSetオブジェクト間の演算とそれに基づくDataSetオブジェクトを作成してみましょう。日本で用いられている不快指数(Temperature-Humidity Index：THI)とは以下で定義される量です。

- $THI = 0.81T + 0.01H(0.99T - 14.3) + 46.3$

ここで、**T** は気温(°C)、**H** は相対湿度(%)を示します。

日本人の場合、不快指数が85になると93%の人が暑さによる不快を感じるとされています。

次に示すスクリプトは、気象庁局地気象予測モデルGPV(LFM-GPV)から近畿周辺の気温と湿度データを取りだし、上の式に代入して不快指数を計算し、それをDataSetオブジェクト**THI**にまとめたうえで、データの概要、分布図、大阪市付近の値の時系列グラフを表示するものです。

このスクリプトを実行して動作を確認してください。

```
In [ ]: import numpy as np

grdir = "./jmadata/lfm/2018/201807/"
grfile = "Z__C_RJTD_20180701060000_LFM_GPV_Rjp_Lsurf_FH*_grib2.bin"
grpath = grdir + "/" + grfile

var = wg.DS_from_grnc(grpath, "TMP_1D5maboveground", lalomima=[33.3, 36.0, 13
var.ts(34.69, 135.42, fig=True)
T = var.data - 273.15
T_filval = var._FillValue - 273.15

var = wg.DS_from_grnc(grpath, "RH_1D5maboveground", lalomima=[33.3, 36.0, 13
var.ts(34.69, 135.42, fig=True)
H = var.data
H_filval = var._FillValue

THI = 0.81 * T + 0.01 * H * (0.99 * T - 14.3) + 46.3
THI_filval = 0.81 * T_filval + 0.01 * H_filval * (0.99 * T_filval - 14.3) +

dsTHI = wg.DS_like(var, data=THI, name="Temperature-Humidity Index", unit="-",
dsTHI.ts(34.69, 135.42, fig=True)
dsTHI.yx(dsTHI.time[0], fig=True)
```

おわりに

この課では、気象庁のGPVデータをwxbcgribモジュールを利用して取り扱う方法を学習しました。Pythonを利用すると気象庁のGPVデータも結構簡単に利用できることを感じてもらえたでしょうか。

wxbcgribモジュールの実態は、フォルダ **challenge** 内のテキストファイル **wxbcgrib.py** です。これは、必ずしもPythonのエキスパートではないWXBC人材育成ワーキンググループの有志が作成したいわばバージョン1です。GRIBファイルやPythonプログラミングにより深い知識のある方々の手でさらに便利でさらに効率的なものに改良され、利用がどんどん広がることを願っています。