

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2課 Python によるGPVデータ処理の基礎

第1課で学習した通り、wgrib2は、コマンドプロンプトにキーボードで文字列を打ち込んで利用するプログラムです。Pythonはこのようなプログラムを実行させることができます。Pythonスクリプトで実行すると、実行により出力された表示やファイルをそのまま次の処理に使うことができ便利です。第2課では、GRIBファイルで提供される気象庁GPVデータをwgrib2でNetCDFファイルに変換し、さらにここからデータを取り出して可視化する処理をPythonで行う方法を学習します。

それでは、第1課で実施したのと同じことをPythonにやらせてみましょう。wgrib2で、気象庁のメソ数値予報モデルGPVのファイルからデータの概要を取り出させます。以下を実行してください。

2-1. モジュールのインポート

Pythonは科学技術計算からAIまで様々な専門分野で利用されますが、それぞれの特殊な機能

はモジュールと呼ばれるプログラムにまとめられていて、利用者が必要に応じ選択して自らのプログラムに組み込んで利用します。Pythonには、**subprocess** と呼ばれるモジュールが用意されていて、これを使うと、第1課で使用したWindowsのコマンドプロンプトやMacのターミナルと同じ実行環境を作り出すことができます。モジュールの組み込みはPythonスクリプトに、以下のように記述して行います。

```
In [ ]: import subprocess
```

2-2. コマンドラインプログラムの実行

コマンドラインプログラムをPythonで実行するには、コマンドラインに打ち込む文字列を *"command string"* として、以下のようにスクリプトを書きます。

```
rc = subprocess.run("command string",
                    shell=True,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    universal_newlines=True)
for line in rc.stdout.splitlines():
    print(line)
```

それでは、Windowsが標準で持つコマンドラインプログラム **dir** を *command string* の部分に代入して実行し、動作を確認してみます。**dir** はディレクトリやファイルの名前を表示させるコマンドラインプログラムです。Macでは、**ls** がこれに相当するのでこれに書き換えてください。

1. 下のセルをクリックして緑色の枠を表示させます(緑の枠はセルが編集可能な状態にあることを示します)。
2. 文字列を書き換えます。
3. ツールバーの **[>Run]** ボタンをクリックするか、**Ctrl**キーを押しながら**Enter**キーを押します。

```
In [ ]: rc = subprocess.run("command string",
                            shell=True,
                            stdout=subprocess.PIPE,
                            stderr=subprocess.PIPE,
                            universal_newlines=True)
for line in rc.stdout.splitlines():
    print(line)
```

ほとんどの人はエラーが出たと思います。Pythonのエラーは下から読むのがコツです。一番下には、「'subprocess'という名は定義されていない。」と表示されてます。これから、**subprocess** モジュールがまだ組み込まれていないことがわかりました。2-1にあるセルをクリックして枠を緑色にしてから[Ctrl]キーを押しながら[Enter]キーを押してインポート文を実行し、その後で改めて上のセルを実行しなおしてください。今度は、フォルダchallengeの中のファイルやサブフォルダの一覧が表示されるはずです。

2-3. GRIBファイル内のメタ情報の表示

それでは、第1課で実施したのと同じことをPythonにやらせてみましょう。wgrib2で、気

象庁のメソ数値予報モデルGPVのファイルからデータの概要を取り出させます。以下を実行してください。

(Macの方は、「**c:/wgrib2/wgrib2**」を「**~/work/grib2/wgrib2/wgrib2**」に書き換えてください)

```
In [ ]: rc = subprocess.run("c:/wgrib2/wgrib2 jmadata/msm/2018/201807/Z__C_RJTD_20180701060000_MSM_GP_V_Rjp_Lsurf_FH00-15_grib2.bin",
                           shell=True,
                           stdout=subprocess.PIPE,
                           stderr=subprocess.PIPE,
                           universal_newlines=True)
for line in rc.stderr.splitlines():
    print(line)
for line in rc.stdout.splitlines():
    print(line)
```

Windowsのコマンドラインに入力する文字列と関数 **subprocess.run()** に入力する文字列とは基本的には同じですが、パスを示す文字だけは異なることに注意してください。日本語版Windowsにおいてパスを示す文字列は円マーク「¥」ですが、pythonではMacと同じ半角スラッシュ[/]を用います。

Pythonのスクリプトを利用するとwgrib2が扱いやすくなります。文字列変数を用いると、少しだけですがファイルの指定がわかりやすくなります。

```
In [ ]: wgrib2 = "c:/wgrib2/wgrib2"      # Macの場合 wgrib2 = "~/work/grib2/wgrib2/wgrib2"
grdir = "jmadata/msm/2018/201807"
grfile = "Z__C_RJTD_20180701060000_MSM_GP_V_Rjp_Lsurf_FH00-15_grib2.bin"
grpath = grdir + "/" + grfile
rc = subprocess.run(f'{wgrib2} {grpath}',
                    shell=True,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    universal_newlines=True)
for line in rc.stderr.splitlines():
    print(line)
for line in rc.stdout.splitlines():
    print(line)
```

Pythonにもう少し働いてもらい、出力される文字列を整理し、キーワードとしてまとめてみましょう。

```
In [ ]: import subprocess

wgrib2 = "c:/wgrib2/wgrib2" # Macの場合 wgrib2 = "~/work/grib2/wgrib2/wg
grdir = "jmadata/msm/2018/201807"
grfile = "Z__C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-15_grib2.bin"
grpath = grdir + "/" + grfile

rc = subprocess.run(f'{wgrib2} {grpath}',
                    shell=True,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    universal_newlines=True)

msgs = []
for line in rc.stdout.splitlines():
    msg = line.split(":") [1:-1]
    msgs.append(msg)

for i in range(len(msgs[0][:])):
    kw = [msg[i] for msg in msgs]
    print(sorted(set(kw)))
    print("-"*10)
```

GRIBファイルに格納されているメッセージの中から特定のメッセージ(「.183:」 という文字列を持つメッセージ)を取り出してその詳細を表示します。

```
In [ ]: import subprocess

wgrib2 = "c:/wgrib2/wgrib2" # Macの場合 wgrib2 = "~/work/grib2/wgrib2/wg
grdir = "jmadata/msm/2018/201807"
grfile = "Z__C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-15_grib2.bin"
grpath = grdir + "/" + grfile

kwds = '-match "\.183:"' #文字列中に「"」を入れたいときは、文字列を「'」で囲みます

rc = subprocess.run(f'{wgrib2} -V {kwds} {grpath}', # 詳細な情報を表示させる
                    shell=True,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    universal_newlines=True)

for line in rc.stderr.splitlines():
    print(line)
for line in rc.stdout.splitlines():
    print(line)
```

wgrib2には、**-match** や **-V** 以外にもたくさんのコマンドラインオプションがあるので、それらを表示させてみましょう。以下を実行してください。

```
In [ ]: import subprocess

wgrib2 = "c:/wgrib2/wgrib2"      # Macの場合 wgrib2 = "~/work/grib2/wgrib2/wg

rc = subprocess.run(f'{wgrib2} -h',
                    shell=True,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    universal_newlines=True)
for line in rc.stderr.splitlines():
    print(line)
for line in rc.stdout.splitlines():
    print(line)
```

2-4. GRIBファイルのNetCDFファイルへの変換

Pythonには、GRIBファイルを操作する**pygrib** と呼ばれるモジュールが存在しますが、残念ながらWindows版の **pygrib** は提供されていません。さらに、気象庁のGPVプロダクトの一部は **pygrib** で扱うことができません。従って、Pythonですべてを実行するのではなく、wgribを使ってGRIBファイルの内容をPythonが読み書きできる別なフォーマット(NetCDF)のファイルに変換してこれを操作するほうが賢明です。

下のスクリプトで、GRIBファイルの内容をNetCDFファイルに変換することができます。さらに、実用性を考慮し、NetCDFファイルを配置するフォルダがないときには新規に作成する機能も加えてあります。ファイルやディレクトリの操作をするモジュール **os** を使用していることに注意してください。

下のスクリプトを実行して、所定の場所にNetCDFファイルが生成されていることを確認してください。

```
In [ ]: import subprocess
import os

wgrib2 = "c:/wgrib2/wgrib2"      # Macの場合 wgrib2 = "~/work/grib2/wgrib2/wg
grdir = "jmadata/msm/2018/201807"
grfile = "Z__C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-15_grib2.bin"
grpath = os.path.join(grdir, grfile)  # ←このようなやり方もあります

ncdir = "./nc"
if not os.path.isdir(ncdir):          #NetCDFファイルの置き場が無ければ作る
    os.makedirs(ncdir)
ncpath = os.path.join(ncdir, grfile) + ".nc"

rc = subprocess.run(f'{wgrib2} {grpath} -netcdf {ncpath}',
                    shell=True,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    universal_newlines=True)
for line in rc.stderr.splitlines():
    print(line)
for line in rc.stdout.splitlines():
    print(line)
```

2-5. NetCDFファイルからのデータ読み込み

NetCDFファイルは、モジュール **netCDF4** で提供される関数 **Dataset()** でオープンします。

```
In [ ]: import netCDF4
ds = netCDF4.Dataset("./nc/Z__C_RJTD_20180701060000_MSM_GPV_Rjp_Lsurf_FH00-
```

これにより、netCDF4で定義されるDatasetオブジェクト **ds** が生成されます。「Datasetオブジェクト」とは、長々表現すると、モジュールnetCDF4のインポートにより利用可能となる関数Datasetにより作られたモノで、ファイルへのアクセスを取り次ぐ窓口のような役割を果たします。データではありません。 窓口なので問い合わせに作法があります。ファイルに格納されているデータの一覧を見せてもらうには以下のようにします。

```
ds.variables
```

窓口 **ds** は、ファイルに格納されているデータの概要を辞書形式(Pythonのデータ形式の一種)で返します。辞書はキーワードとそれに対する内容で構成されていて、キーワードを指定することで特定のデータの概要を表示させることができます。

まずはどのようなキーワードがあるかを調べましょう。以下を実行してください。

```
In [ ]: ds.variables.keys()
```

示されたキー中から'TMP_1D5maboveground'というキーに紐づけられているデータの情報を取り出してみましょう。以下を実行してください。

```
In [ ]: ds.variables['TMP_1D5maboveground']
```

ここで、NetCDFに変換されたファイルから取り出されるデータ概要は、GRIBファイルから直接取り出したデータの概要とは一致していないことに注意してください。気象要素の名称が異なっていることには特に注意が必要です。

このデータの、本体や名称、単位などを取り出すには、持ち帰る袋 **data** を用意して窓口 **ds** に以下のように依頼します。

```
data = ds['TMP_1D5maboveground']
```

袋 **data** からそれぞれを取り出すには以下のようにします。実行結果を上の情報と見比べてみてください。

```
In [ ]: data = ds['TMP_1D5maboveground']
print("データ名", data.long_name)
print("単位", data.units)
print("サイズ", data[:].shape)
print("データ本体\n", data[:])
```

ここで、ファイルに格納されているデータの次元にも違いがあることに注意してください。GRIBファイルには、データが予報時間毎に格納されていますが、変換されたNntCDFファイルには、これらが結合されて格納されています。

2-6. データの可視化

GPVデータの取り出し方が分かったので、気象庁メソ数値予報モデルGPVデータから気温の予測データを取り出し分布図を描いてみます。ここで、モジュール **matplotlib** はデータを様々に可視化することができるモジュールです。また、モジュール **numpy** は多次元のデータを効率的に取り扱えようとするモジュールです。

In []:

```
# モジュールのインポート
import numpy as np
import matplotlib.pyplot as plt

# データの受け取り
data = ds['TMP_1D5mabovEGround']
dat = data[0,:,:] - 273.15
lat = ds['latitude'][: ]
lon = ds['longitude'][: ]

# 描画
tate = 6 #図の台紙の全体的な大きさを指定します。
figtitle = data.long_name + " [degC]"
manualscl = True #カラースケールの上限值と下限値を指定したいときに使用します。
sclmax = 35.0 #最大値
sclmin = 5.0 #最小値
sclint = 0.5 #色の刻み
if not manualscl :
    sclmax = round(np.nanmax(np.array(dat)),1)
    sclmin = round(np.nanmin(np.array(dat)),1)
    sclint = round(((sclmax-sclmin)/10.0),1)
levels = np.arange(sclmin, sclmax+sclint, sclint)
yoko = tate * (np.max(lon)-np.min(lon))/(np.max(lat)-np.min(lat)) + 2

plt.figure(figsize=(yoko,tate)) # プロット領域の作成 (matplotlib)
ax = plt.subplot(1,1,1, facecolor='0.8')

cmap = plt.cm.Spectral_r #色調(カラーマップ)を愛称(「_r」を最後に付けると反転す
cf = ax.imshow(dat, cmap=cmap, origin='lower', aspect='equal',
               vmin=sclmin, vmax=sclmax)

plt.colorbar(cf)
plt.title(figtitle)
plt.show()
plt.close()
```

おわりに

この課では、コマンドラインプログラムwgrib2を操作してGRIBファイルをNetCDFファイルに変換し、そこから気温予報データを取り出して分布図を作成するまでを、Pythonで実施する方法を学びました。実習では、個々の作業を理解するために、寄り道をしつつ細切れのスクリプトを実行しましたが、ひとつながりのスクリプトにしてしまえば、これら一連の処理を連続して実行することができます。これがPythonを利用する利点の一つです。また分布図の描画で見た通り、Pythonを利用すると高度なデータ処理を簡単に行うことができます。近年急速に一般化して来たAIに投げ込むことも可能です。これがPythonを利用するもう一つの利点です。これを機会に、皆さんも是非Pythonに挑戦してみてください。