Ⅳ メッシュ農業気象データの処理１(Pythonによる処理)

1　はじめに

Pythonでメッシュ農業気象データを利用するには、データ取得や計算・出力などの関数をまとめた道具箱（AMD_Tools3.py）を用います。ここでは、サンプルプログラムを用いて、以下の内容について実習します。それぞれの項目のカッコ内がAMD_Tools3.pyに入っている関数です。

・データの取得（GetMetData関数）
・メッシュ図の描画（PutGSI_Map関数）
・地理情報データの使用（GetGeoData関数）
・時系列やメッシュデータのCSVファイルへの書き出し（PutCSV_TS関数、PutCSV_MT関数）
・積算気温を例にした、メッシュデータ（配列）のハンドリング
・あらかじめデータをダウンロードしてオフラインで利用する方法

なお、AMD_Tools3.py内の各関数に対しては、変更や書き換えをしないようお願いします。修正が必要な場合は、各自のプログラムの中で新たな関数として記述するようにして下さい。

2　メッシュデータを取ってきて地図上に投影する

1）まずはやってみる

Spyderを起動し、右上のファイル選択画面にPythonWorksを表示したら、その中からIV_sample_1.pyを選択してください。左のエディター画面に読み込まれたことを確認して上のメニューバーの緑色の三角をクリックすると処理が始まり、同じディレクトリに

TMP_mea.html
TMP_mea_l.png
TMP_mea_o.png

という３つのファイルができます。ファイル名のTMP_meaはメッシュ農業気象データシステムで使われている気象要素名で、日平均気温を表します。

TMP_mea.htmlをウェブブラウザで開くと、図１のような表示が現れます。ベースとなっているのは国土地理院の地理院地図で、その上に、プログラムで指定した矩形範囲の気象データの分布が重ねて表示されています。この例では、北緯32.7度～34.37度、東経130.99度～133.05度の領域の2016年1月1日の日平均気温です。また、同時にできたTMP_mea_o.pngは、地図に重ね合わせるメッシュ図の画像、TMP_mea_l.pngは左下に表示される凡例（カラーバー）の画像です。

なお、地理院地図は、出典を記載する等の使用条件に従えば、商用も含めて自由に利用できます。使用条件は、国土地理院コンテンツ利用規約

http://www.gsi.go.jp/kikakuchousei/kikakuchousei40182.html

を参照してください（このリンクはIV_sample_1.pyの105行にあります）。

2) プログラムの概観と宣言部分

次にプログラムの中身を見ていきます。このプログラムは見た目は長いですが、ほとんどは「"""」で挟まれたり「#」で始まるコメント行ですので、必要な行だけを抜き出すと、以下のようになります（行頭の数字は行番号を表します）。なお、52行と81行は便宜上途中で改行していますが、Pythonの文は1行で記述する必要があります。ただし、括弧の中では自由に改行できますので、これを用いて可読性を高めるテクニックもあります（そのようなわけで81行はセーフです）。

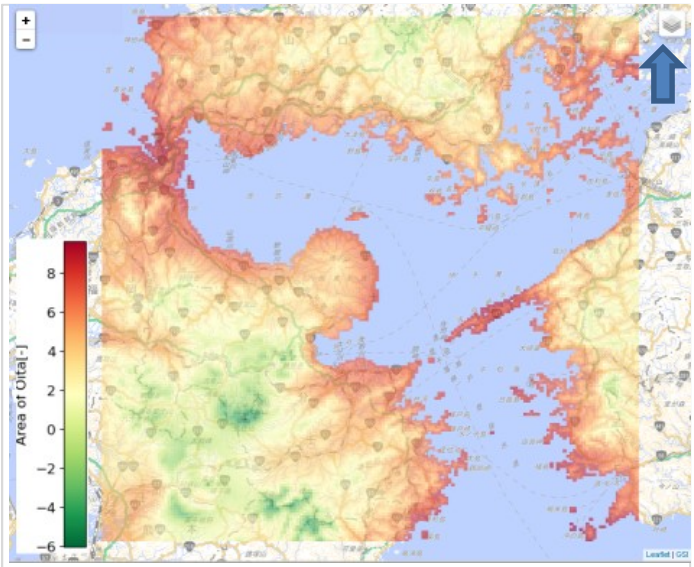

図１． TMP_mea.html をブラウザで表示した例。地図選択アイコン（右上の矢印部分）をクリックすると、標準地図、淡色地図、白地図などの選択や、重ねるメッシュ図の濃度の調整ができる。

```
 1: # -*- coding: utf-8 -*-
49: import AMD_Tools3 as AMD
52: element = “TMP_mea”
53: timedomain = [ "2016-01-01", "2016-01-01" ]
54: lalodomain = [ 32.7, 34.37, 130.99, 133.05]
57: Msh,tim,lat,lon,nam,uni
  =AMD.GetMetData(element,timedomain,lalodomain,cli=False,namuni=True)
66: Msh = Msh[0,:,:]
86: AMD.PutGSI_Map(Msh,lat,lon,label=nam+"["+uni+"]",cmapstr=None,
      minmax=None,filename=element)
```

1行目は「#」で始まってはいますが、このプログラムが「utf-8」というコードで書かれているという宣言です。49行目はAMD_Tools3.pyを使用する宣言です。「as AMD」と書かれているように、この行以降AMD_Tools3.py内の各関数は「AMD.関数名」で呼び出すことができるようになります。ここまでは大抵のプログラムに共通の、いわばおまじないです。

52～54行で、読み込む要素（element）と読み込む日付（timedomain）と読み込む範囲（lalodomain）を指定しています。まず52行目の要素ですが、要素名を「"」（「'」でも良い）で挟んで指定します。現在使用できる要素は、本講習会テキスト第１章の表１に記載されています。53行目のtimedomainという変数は、データ取得期間の始日と終日を表す２つの文字列を格納するリスト変数です。この例では、ある1日だけのデータ（スナップショット）を取得したいの

で、始日と終日に同じ日付を指定しています。同様に、54行目のlalodomainは、取得範囲を示す緯度経度の 4 つの数字を格納するリスト変数で、

[南端の緯度，北端の緯度，西端の経度，東端の経度]

の順で指定します。こちらも、もし二次元ではなくてある一点だけのデータが欲しい場合は、南端と北端に同じ緯度、西端と東端に同じ経度を指定します。なお、このプログラム内では、緯度経度は度の十進法で表記する必要があります。

3）メッシュ農業気象データを取得する関数GetMetData

57行目で、AMD_Tools3.py内の関数GetMetDataを呼び出しています。これは、メッシュ農業気象データシステムのサーバにリクエストを送って、気象要素を所得する関数です。ようするにコマンドやサブルーチンのようなものですが、PythonなどC言語の流れを汲む言語では「関数」と呼び、表記法もたいてい数学の関数と同様に $y=f(x)$ の形になっています。

まず上式の x にあたる引数（GetMetDataの後ろの括弧の中）を見ていきます。elementは52行目で指定した要素名で“TMP_mea”が入っています。timedomain、lalodomainもそれぞれ53行目、54行目で指定したものです。次のcliは平年値（climatic normal）を取得するかどうかの指定です。cli=Trueを指定すると平年値、この例のようにcli=Falseと指定するか、またはこの項目自体を省略すると、年々の値（観測値や予報値）を取得します。ただし、要素によっては平年値が存在しないものもありますので、マニュアルで確認してください。最後のnamuniは要素の英語正式名称（name）と単位（unit）をサーバから取得するかどうかの指定で、cliと同様にnamuni=Trueなら取得しますし、namuni=Falseか省略なら取得しません。なお、これらを取得する場合の返り値の数（左辺の変数の個数）が6個なのに対し、取得しない場合は4個に減りますので、注意しないとエラーが起こります。ですからプログラムごとに変えずに、必要がなくても常にnamuni=Trueとしておく方が無難でしょう。

次は左辺です。こちらの変数は、GetMetData関数が返してくる値を格納するためにユーザーが準備する入れ物ですから、名前は自由に指定できます。

Mshには気象データが格納されます。この場合は日平均気温です。ここで強調しておきたいのは、GetMetData関数が返す気象データは、常に[時間（日数）×南北方向のメッシュ数×東西方向のメッシュ数]という三次元の形式である点です。Spyderの右下のIPythonコンソールにはプログラムを実行する際に出るメッセージが表示されますが、このプログラムを実行すると、

```
Data shape: (1, 200, 165)
```

という記述が表示されます。これは、1日分の南北方向200メッシュの東西方向165メッシュの配列データであることを示しています。たとえば、ある 1 地点（南北方向 1 メッシュ、東西方向 1 メッシュ）のデータを365日分取得すれば、

```
Data shape: (365, 1, 1)
```

となります。

timには取得したデータの日付が格納されます。これは一次元の配列です。この行の下に print(tim) と書けばIPythonコンソールに配列の中身を表示させることができますが、この例では時間方向の要素数は 1 ですので、[datetime.datetime(2016, 1, 1, 0, 0)]と、timedomainで指定した1日分の日付が表示されます。なお、datetime.datetimeについては、３節で説明します

latには各メッシュの中心点の緯度が格納されます。これは一次元の配列です。lalodomainで指定した南端の座標から順に各メッシュの中心点の緯度が並んでいます。

[32.70416641　32.71250153　32.72083282　32.72916794　32.73749924.....]

lonも同様に、各メッシュの中心点の経度が格納された一次元の配列です。

最後の２つ、numとuniは、上で書いたようにnamuni=Trueとした場合に返される要素名称と単位を格納する文字列変数です。この例では、取得した名称と単位は、図 1 の左下のカラーバーに書かれている “Daily mean air temperature” と “degC” です。

続いて66行目のMsh = Msh[0,:,:]です。これは、三次元の時系列データを二次元化する操作で、1 日 1 枚の絵が重ねられた束の中から 1 枚を抜き出すイメージです。この例では絵は 1 枚（1 日分）しかないので最初のものを選ぶしかありませんが、その指定が括弧の中の最初の「0」です（コンピュータの関係ではたいてい0から数えます）。たとえば、53行目で

```
timedomain = [ "2016-01-01", "2016-01-10" ]
```

というように複数日を指定した場合は、絵もその日数分だけありますから、

```
Msh = Msh[3,:,:]    # 4枚目（1月4日のデータ）を抜き出す
```

などという指定もできます。また、あまり使うことはないでしょうが、

```
Msh = Msh[:,y,x]    # 南からy個目、西からx個目のメッシュの経時変化（一次元）
```

といった使い方もできます。ここで、抜き出す軸以外の「:」というのは、範囲指定の区切り記号で、本来は[0,y1:y2,x1:x2]といった書き方になります。この場合は、南北方向のメッシュのy1番目からy2-1番目、東西方向のメッシュのx1番目からx2-1番目の範囲にトリミングするという意味ですが、このy1,y2,x1,x2を省略した場合は「全範囲」、つまりトリミングしないという意味になります。もちろん時間軸に対してもt1:t2という指定は可能です。ただし、これらはあくまでもトリミングですから、たとえ[0:1,0:1,0:1]と全ての軸の要素数を 1 にしても、次元は三次元のまま変わりません。一方、「0,0,0」と指定すれば「1 日目の南西端のメッシュのデータ」という 1 個の数字（ゼロ次元）になります。いま自分が扱っているメッシュデータがどのような形なのか（何次元なのか）を確認したいときは、プログラム中に

```
print(Msh.shape)     #「配列Mshの形を出力しろ」という意味
```

と書いて実行すれば、IPythonコンソールに(1, 200, 165)などと表示されます。この数字の個数（軸の数）が次元数を表します。

4）メッシュデータを地図上に描き出す関数PutGSI_Map

86行目のPutGSI_Mapは、指定した二次元データを描画する関数です。この関数には返り値はなく、代わりに画像ファイルと国土地理院地図にオーバーレイするためのハイパーテキストを出力します。引数を順に見ていきましょう。

最初の３つ、Msh,lat,lonは、GetMetData関数で取得した気象データ、緯度、経度の配列です。ただしMshは、取得時は三次元だったのが66行目で二次元に変わっていることにご注意下さい。次のlabelは、図１の左下のカラーバーに表示される文字列です。ここではGetMetData関数でnamuni=Trueと指定して取得した要素名namと単位uniを用いて

```
label= nam + "[" + uni + "]"
     = "Daily mean air temperature" + "[" + "degC" + "]"
     = "Daily mean air temperature [degC]"
```

という文字列の演算をやっています。もちろん、namやuniを使わずにlabel= "TMP_mea"などと勝手に指定しても構いません。ただし、現在のバージョンでは日本語の表示はできません。また、label=Noneと指定したり省略した場合は「result」と書き込まれます。

cmapstrは分布図を描くカラーマップの指定です。cmapstr=Noneとしたり省略した場合はデフォルトとなり、自分で指定する場合は、下記URLから選んだ名称を文字列として（" "で挟んで）指定します。なお、デフォルトは"Spectral_r"です。末尾の_rは反転を意味しています。

http://matplotlib.org/examples/color/colormaps_reference.html
（このリンクはIV_sample_1.pyの108行にあります）

minmaxは凡例の色の上限と下限を指定するパラメタです。Noneの場合は描画領域内の最大値と最小値から、適当に区切りのよい範囲にして描画されます。この機能は前もって値の範囲が不明な場合などには役に立つのですが、上限と下限とを手動で決めたい場合には

```
minmax = [0.0, 15.0]
```

などのように、下限値と上限値のリストとして指定します（lalodomainと同じ方式です）。

filenameは出力するファイルのファイル名を指定します。この例では52行目で指定した要素名 element=”TMP_mea”となっていますが、もちろん自由に指定できます。なお、項目を省略した場合には、ファイル名はデフォルトである"result"となります。また、このサンプルでは省略

しましたが、outdirで出力するディレクトリを指定することができます。

```
outdir= "C:/Users/Komimai/Documents/"
```

省略した場合はカレントディレクトリ（プログラムファイルと同じディレクトリ）となります。

5）PutGSI_Map関数の設定項目を変更してみる

まず、カラーマップの変更をやってみましょう。現在は86行目が生きていますが、その行頭に半角の「#」を入力してコメント行にします。するとその下の行と同じように文字色が灰色の斜体になったと思います。次にすぐ下の87行の行頭の「#」を消します。

```
87: AMD.PutGSI_Map(Msh,lat,lon,label=nam+"["+uni+"]",cmapstr="gnuplot",
      minmax=None,filename=element)
```

87行は、ほぼ86行と同じですが、cmapstrの指定がNoneから"gnuplot"になっています。この状態で最初と同じように三角アイコンをクリックして実行し、できたTMP_mea.htmlをブラウザで開いてみてください。図１のデフォルトとは趣の違う図になっていると思います。88行は、87行と同じgnuplotを使っていますが、"gnuplot_r"として反転させています。色の選択は説得力のあるプレゼンテーションには重要ですので、いろいろ試してみてください。

次に凡例の上限と下限とを変えてみましょう（図２）。86行～88行をコメントアウトし（頭に半角の「#」を入力し）、89行の#を取ります。

```
89: AMD.PutGSI_Map(Msh,lat,lon,label=nam+"["+uni+"]",cmapstr=None,
      minmax=[0.0,10.0],filename=element)
```

今度はminmax=[0.0,10.0]としています。これは凡例のカラースケールの範囲を0℃から10にするという意味です。

6）GetGeoData関数で地理情報を取得する

メッシュ農業気象データシステムでは、気象データ以外にも標高や土地利用など、様々な地理情報が利用できます。地理情報の一覧は、2017年版のマニュアル4ページの表2に記載されています。ここでは、「都道府県別範囲」を用いて、特定の県だけのデータを表示させてみましょう。今度は71行、74行、83行の行頭の#を取ります（他の行は現在のままでかまいません）。

```
71: element = "pref_4400"
74: Geo,lat,lon,nam,uni
          = AMD.GetGeoData(element,lalodomain,namuni=True)
```

```
83: Msh = Msh * Geo
```

71行目では文字列変数elementに地理情報の種類を表す文字列を代入します。"pref_4400"というのは、大分県の領域を示す地理情報です。

74行目がGetGeoData関数です。一見してGetMetData関数と似ているのに気づかれたかと思います。ただし、地理情報には時間軸がないので引数にtimedomainが入っていませんし、平年値を指定するcliもありません。左辺を見ると、まずGeoに地理情報のメッシュデータが入ります。時間（日付け）の返り値はないのでtimはなく、あとのlat,lon,nam,uniはGetMetData関数と同様です。

83行目の説明は後でおこないます。

この状態で三角アイコンをクリックして実行すると、TMP_mea.htmlが作成されたのと同じディレクトリに今度はpref_4400.htmlというファイルができているはずです。これをブラウザに読み込むと、図３が表示されます。

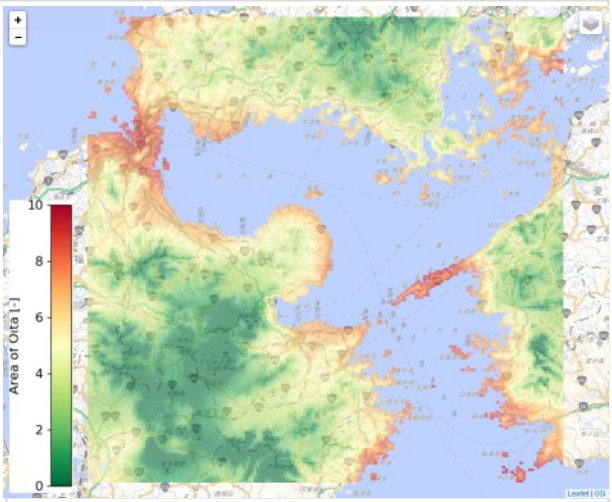

図２．図１と同じデータを、凡例の範囲を変えて作画した例

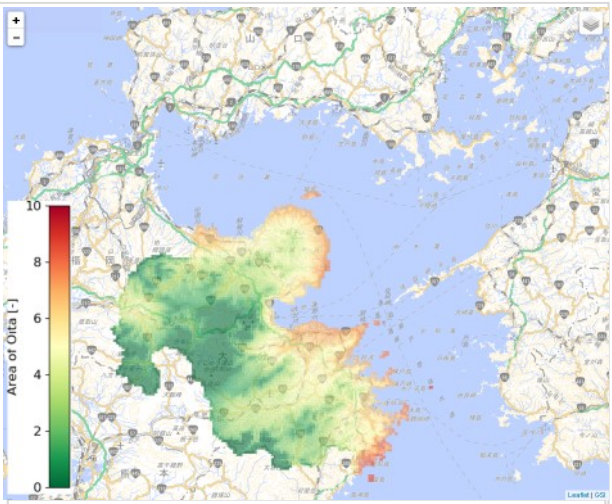

図 3. pref_4400.html を表示した例

今度は大分県の領域のみに色がついています。ただし、左下の凡例に、先ほどは"Daily mean air temperature[degC]"と書かれていたのに、今度は"Area of Oita[-]"になっていますし、ファイル名も先ほどとは変わっています。これは実は、namやuni、elementといった変数名をGetMetData関数と使いまわしたことによる副作用です。GetMetData関数のところで

```
element="TMP_mea"、num="Daily mean air temperature"、uni=”degC”
```

だったものが71行と74行で上書きされて

```
element=" pref_4400"、num="Area of Oita"、uni=”-”
```

となってしまったことが原因でした。これを避けるためには、たとえば地理情報の変数名の頭には「g」をつけるなどの規則を決めて、気象データの変数と区別すれば大丈夫です。

```
71: gelement = "pref_4400"
74: Geo,lat,lon,gnam,guni
          = AMD.GetGeoData(gelement,lalodomain,namuni=True)
```

ここでひとつ注意点ですが、さきほどGetMetData関数が返す気象データ（Mshに格納したもの）は、必ず三次元だと述べました。それを66行で二次元に直したわけですが、今度のGetGeoData関数が返す地理情報データは必ず二次元になっています。さきほど実行した結果、IPythonコンソールには

```
Data shape: (1, 200, 164)
Data shape: (200, 164)
```

という２つの配列の形が表示されていると思いますが、この2つ目がGetGeoData関数で取得した地理情報の形です。最初から二次元であることがわかります。

次に83行の説明です

```
83: Msh = Msh * Geo
```

この式は数学的に考えるとMsh=0かGeo=1しかありえませんが、ここでは、「変数Mshに格納されている配列と、変数Geoに格納されている配列とを掛け合わせ、その結果を変数Mshに代入せよ」という意味です（なお、等しいという意味で等号を使う場合は「A==B」のように２個続けます）。今回の例ではPutGSI_Map関数を書き換えるのが面倒だったのでMshに上書きしましたが、もちろん以下のように新しい変数にしても構いません。

```
Msh_Oita = Msh * Geo
```

Geoに入っているのは、大分県の領域には”1”、それ以外は無効値”nan”で埋められた配列ですから、Mshの大分県の部分は変わらず、その他の県は海と同じ扱いになります。

3　データをcsvファイルに書き出す

1）PutCSV_TS関数で時系列データを書き出す

ここでIV_sample_1.pyを閉じて、IV_sample_2.pyを読み込みます。ここでは１地点の時系列データを取得し、エクセルなどで利用できるcsvファイルに書き出します。まず２．2)で説明した「おまじない」の部分ですが、今回は増えています。

```
47: import AMD_Tools3 as AMD
48: import numpy as np
```

```
49: from datetime import datetime
50: import matplotlib.pylab as plt
51: import matplotlib.dates as md
```

48行目で使用を宣言しているnumpyは、配列の処理を高速に行う数値計算ライブラリNumPyを指しています。実は先ほどのIV_sample_1.pyの中で取り扱った配列もNumPyで取り扱う形式だったのですが、IV_sample_1.pyの中ではNumPyの機能を使用していませんでした。

49行目は、２．3)でも出てきましたが、Pythonで時間を扱うdatetimeモジュールから、datetimeオブジェクトを使用するという宣言です。

50行目、51行目は描画を行うためのライブラリで、このサンプルプログラムは時系列グラフも描くために入っています。

```
55: element = 'TMP_mea'
56: timedomain = [ "2017-04-01", "2017-06-30" ]
57: lalodomain = [ 36.0566, 36.0566, 140.125, 140.125]
```

55～57行目はおなじみのelement,timedomain,lalodomainの指定です。今回は2017年の4月～6月の３ヶ月間で、取得範囲はつくば市館野の高層気象台の一点です（南端北端の緯度、西端東端の経度がそれぞれ同じになっています）。

```
60: Msh,tim,lat,lon,nam,uni
     = AMD.GetMetData(element,timedomain,lalodomain,namuni=True)
61: MshN,tim,lat,lon
    = AMD.GetMetData(element,timedomain,lalodomain,cli=1)
62: Msh = Msh[:,0,0]     # 入れ物の入れ替え(三次元から一次元へ)
63: MshN = MshN[:,0,0]   # 入れ物の入れ替え(三次元から一次元へ)
```

60行目はGetMetData関数でデータを取得してMshに格納、61行目は同じく平年値を取得してMshNに格納しています。さらに62, 63行目では三次元から一次元への変換を行っています。

ここまでで材料となる時系列データはそろいましたが、csvファイルに書き出す前にもう1ステップあります。

```
76: Tateno = np.array([MshN,Msh])
80: AMD.PutCSV_TS(Tateno, tim,
            header="日付,平年値,観測値", filename="result.csv")
```

76行目では、それぞれ一次元の観測値時系列（Msh）と平年値時系列（MshN）とをまとめて、新たな二次元の配列を作っています。エクセルでイメージすると、行方向にデータが並んだ一次

元のデータを横に並べる（列を増やす）操作です。80行目で呼び出している時系列データ用のcsv書き出し関数PutCSV_TSは、もちろん 1 つの要素だけを書き出すことはできるのですが、この例のように平年値と観測値を比較したり、気温と雨量を並べたりできるよう、与えた配列の列数に応じて書き出すようになっています。76行目の括弧の中の[MshN,Msh]は、lalodomainなどと同様のリストで、このように配列を要素にすることも可能です。np.array( )が配列を生成するための命令で、この括弧の中にリスト[MshN,Msh]を入れると、ひとつにまとめて配列となります。72行目、73行目、77行目にMsh,MshN,Tatenoの形を出力する文を入れましたので、実行すると、

```
Msh.shape: (91,)
MshN.shape: (91,)
Tateno.shape: (2, 91)
```

と表示されます。一次元配列であるMshとMshNがくっついて、Tatenoという二次元配列ができたことがわかります。

PutCSV_TSの引数の 1 つ目は、書き出したい配列、2番目のtimはGetMetData関数で取得したもの、3つ目のheaderhaは、書き出すcsvファイルの1行目（凡例）となります。各列ごとにカンマで区切ります。4つ目はファイルネームです。PutGSI_Map関数で見たように、以下のような指定も可能です。

```
filename = element + ".csv"
```

なお、今回の研修では説明しませんが、このサンプルプログラムを実行すると、IPythonコンソールに図 4 のグラフが出力されます。これは84行目以降で描かれているもので、内容を理解すれば様々なアレンジが可能です。なお、このコードを他のプログラムに移植する場合は、50行目、51行目も忘れずにコピーしてください。

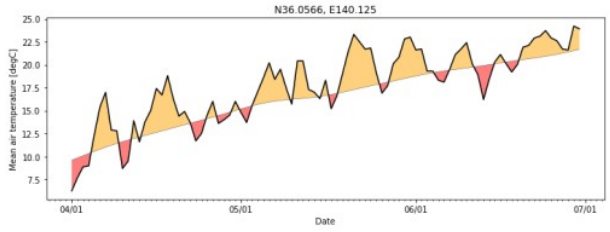

図 4.　サンプルプログラムで描画される時系列グラフ

2）PutCSV_MT関数で国土数値情報３次メッシュコードとともに書き出す

メッシュデータをGISソフトに読み込む場合などは、国土数値情報のメッシュコードをキーとしたテーブル形式への加工が必要な場合があります。その際はPutCSV_MT関数を使います。この使用例はサンプルプログラム集のsample_PutCSV_MT.pyに入っています。

引数などの設定方法は今まで見てきたものとさほど変わらないので、特に戸惑うことはないでしょう。ここでは「このようなこともできる」ということを確認していただければと思います。

なお、AMD_Tools3.pyには緯度経度を国土数値情報メッシュコードに変換するlalo2mesh関

数、逆にメッシュコードから緯度経度を求めるmesh2lalo関数も用意されています。

```
code = AMD.lalo2mesh(lat, lon)
lat, lon = AMD.mesh2lalo(code)
```

ただし、codeは国土数値情報３次メッシュコード（49317789といった8桁の数字）、latとlonはそれぞれ度の十進法で表記された緯度と経度です。

4　期間平均気温や有効積算気温の分布図を作成する

1）基本的なことがら

ここでは、ある期間のメッシュという三次元データを加工するための基本的な事柄を演習します。IV_sample_3.pyを読み込みます。

```
17: element = 'TMP_mea'   # 気象要素の指定（日平均気温）
18: timedomain=[ "2017-06-01", "2017-07-31" ]  # 期間の設定（6月－7月）
19: lalodomain=[41.350,44.0, 139.327,143.000]  # 領域の設定（北海道の西南部）
20: To = 15.0  #有効積算気温の基準(それより低い値は積算しない)
```

このプログラムは18行目で指定した期間の平均気温と有効積算気温を計算して図示するもので、有効積算気温の基準温度は20行目で指定しています。実行するとカレントディレクトリにmean.html（平均気温）とtet.html（有効積算気温）のファイルができ、第二節の例と同様に地理院地図に重ねて表示されます（図５）。

まず平均気温ですが、これにはNumPyの関数であるnp.meanを使います。

```
26: mean = np.mean(Ta, 0)
```

これは、日平均気温の配列Taを0番目の軸方向に平均する、という意味です。括弧の中を省略せずに書くと、(Ta, axis=0)です。IPythonコンソールのログにData shape: (61, 318, 294)とあるように、この配列は318行×294列の絵を61枚（2ヶ月分）束ねた形ですので、それらを時間軸方向に平均して1枚の絵にします（その結果、次元が三次元から二次元に減る）。

次に有効積算気温の計算です。直球勝負で行くなら、たとえば積算気温を入れる配列変数Taccを用意して、i行j列の各メッシュ毎に

```
if Ta[i][j] > To:                          #もしTaが15℃よりも高ければ
   Tacc[i][j] = Tacc[i][j]+(Ta[i][j]-To)   #15℃を超えた分を足しなさい
else:                                      #そうでなければ
```

```
Tacc[i][j] = Tacc[i][j]+0.0                    #0℃を足しなさい
```

という計算を日数分繰り返せば良いのですが、広大な面積のメッシュを 1 つずつ計算していると、とんでもない時間がかかってしまいます。そこでNumPyには、（個々の要素にアクセスせず）多次元のメッシュのまま計算するツールが用意されています。

```
31: valimesh = ~np.isnan(Ta)
33: Ta[valimesh] = np.where(Ta[valimesh] > To , Ta[valimesh]-To, 0.0)
34: Tacc = np.sum(Ta, 0)
```

まず、31行目では、有効な気温の値が存在するメッシュを抽出しています。というのは、メッシュ農業気象データは陸のみに値があるので、海の領域は無効値”nan”が入っています。無効値ですから値がなく、「15℃以上かどうか」といった比較を行うとエラーになってしまいます。そこで、無効値が格納されているメッシュは相手にしない（スキップする）ようにします。np.isnan( )という関数は、括弧内の配列の要素を調べて、

無効値”nan”ならTrue、それ以外ならFalse

という要素を持つ、同じ形の配列を返します。それがvalimeshに格納されるのですが、我々が欲しいのは、その逆の

無効値”nan”ならFalse、それ以外ならTrue

という配列ですから、冒頭に「~」を付けて逆転させています。32行目の#print(valimesh)の#を取ると配列valimeshの内容がコンソールに表示されるので、「~」の有無で真偽が逆転することが確認できます。33行目は、日々のメッシュについて基準温度（To=15℃）よりも気温が高いメッシュは15℃を引いた値を、基準温度よりも気温が低いメッシュには0℃を代入する操作を行っています。ここでTa[valimesh]というのは、[ ]内の条件に合う(値がTrueの)メッシュのみを相手にするという意味です。右辺のnp.where( )というのは、条件によって出力する値を変える関数で、以下の書式になっています。

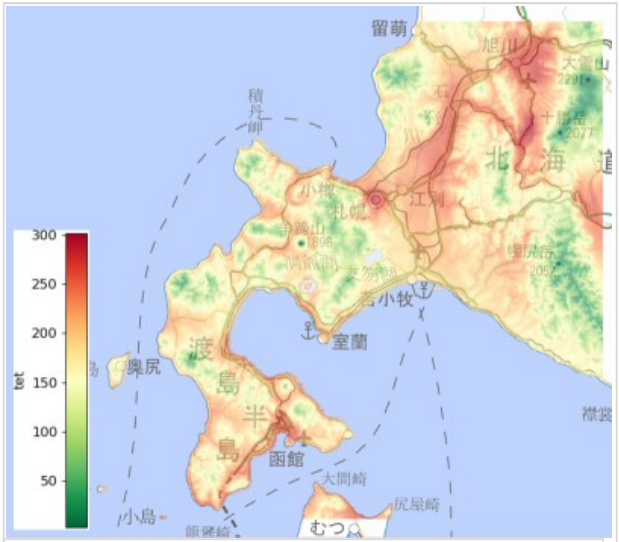

図５． 北海道南西部の 2017 年 6 月 1 日～7 月 31 日の有効積算気温分布

```
np.where(条件式, Trueの場合の返り値, Falseの場合の返り値)
```

条件式は「Ta[valimesh] > To」ですから、これがTrueの場合は、二番目の「Ta[valimesh]-To」が、Falseの場合は3番目の「0.0」が返されます。なお、np.where( )を覚えておけばいろい

ろ応用が利きますが、今回の場合はより簡単に

```
Ta[valimesh] = np.maximum(Ta[valimesh]-To, 0.0)
```

と書いても同じ結果が得られます。np.maximum( )は、括弧内の一番大きな値を出力するもので、選択肢の数が 3 個以上でも大丈夫です。なお、逆のnp.minimum( )もあります。

34行目のnp.sum(Ta, 0)は、Taを時間軸の方向に積算するもので、使い方は26行目のnp.mean(Ta, 0)と同じです。

2) 複数の要素を使った場合

ここでは、TMP_mea（日平均気温）のみを用いた例でしたが、実際には「気温が○℃以上の日に降った雨の量」とか、「日射量が△MJ/m^{2}以上で風速◇m/s以上の日数」とか、複数の要素を組み合わせて使う場合も多いと思います。次はI IV_sample_4.pyを開いて、そのような例を見ておきます。この例では、先ほどの「気温15度以上の積算気温」という条件に、「1mm/d以上の降水があった日」という条件を加えます。

```
34: Ta[valimesh] = np.where(
            np.logical_and(Pr[valimesh]>=Po, Ta[valimesh]>To)
            ,Ta[valimesh]-To, 0.0)
```

34行目に出てくるnp.logical_and( ) というのは、括弧内に列挙された条件式が全て満たされた場合にTrueを返すものです。ここでは降水量がPo（1mm/d）以上で、かつ気温がTo（15℃）を超えた場合にTrueとなります。なお、同じ形の関数にnp.logical_or( )もあります。こちらはどれか1つでも満たされたらTrueを返します。

3) 新しい配列の作り方

上の例の34行目では、日平均気温の配列Taは新しい条件の数値で書き換えられてしまうので、他の計算でも日平均気温を使いたい場合には不都合です。その場合は、Taと同じ形の配列（容器）を新たに作って代入する方が便利です。また、DVIや真夏日の日数など、新しい変数を計算する場合も、格納する配列を新たに作らなければいけません。ここではそのやり方を紹介します。

34行目と35行目の行頭に「#」を入れ、38～41行の「#」を取ります。

```
38: newTa = np.zeros(Ta.shape)     # Taと同じ形で、全要素の値が0の配列
39: newTa[~valimesh] = np.nan      # 海の領域を無効値"nan"に
40: newTa[valimesh] = np.where(
          np.logical_and(Pr[valimesh] >= Po, Ta[valimesh]>To)
          ,Ta[valimesh]-To, 0.0)
```

```
41: Rta = np.sum(newTa, 0)
```

38行目で、Taと同じ形で全要素に”0”が入った新しい配列newTaを作っています。括弧の中（配列の形の指定）はTa.shapeとしていますが、この値を直接入れて、np.zeros((31, 318, 294))と指定しても構いません。また、np.ones( )だと、全要素に”1”が入った配列ができます。

39行目では、海の領域に無効値”nan”を代入しています。31行目で作った真偽の配列valimeshは陸地がTrueなので、「~」を付けて真偽を逆転すれば海のみを相手に操作することができます。もちろん、以下のように普通の数値を代入することもできます。

```
newTa[~valimesh] = 0.0        # 海を0で満たす
newTa[valimesh]  = 1.0        # 陸を1で満たす
```

40行目、41行目は34行目、35行目と同じです。

5　オフラインでメッシュデータを使う

ここまで見てきた使いかたはリアルタイムにデータを取得するものでしたが、サーバのデータをいったん自分のハードディスクにダウンロードしておいて、オフラインで（サーバに接続しないで）使う方法もあります。これだとネットにつながっていないパソコンでも作業ができますし、同じデータに頻繁にアクセスするようなプログラムでは、毎回ダウンロードするよりも高速化できます、このやり方については、メッシュ農業気象データ利用者のページ

https://ml-wiki.sys.affrc.go.jp/MeshUser/start

に詳しく載っていますので、これに沿って手順を説明します。

1）データファイルを置く場所を準備する

作業用ディレクトリ（サンプルプログラムのあるディレクトリ）の下に、図６のようにデータを格納するためのディレクトリを作ります。ここで、Area3は今回の実習で取り上げる茨城県の含まれる領域、その下の”2016”,”2017”は年次を表しています。”GeoData”は地理情報の置き場です。このディレクトリ構造はデータサーバと一致していないといけません。ただし、ルートディレクトリ”AMD”の名前と場所はユーザーが自由に設定できます。

次にダウンロードする範囲を決めておきます。サーバから直接ダウンロードする際には、緯度経度ではなく、「領域の端からx1番目～x2番目のメッシュ」という指定をするので、”AMGSDの領域.xls”で調べてメモしておきます。図７は茨城県域のデータをダウンロードする例です。

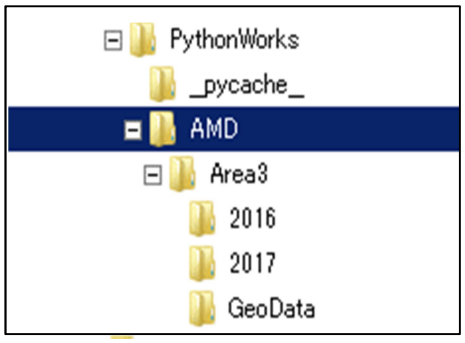

図 6． オフラインデータ格納用のディレクトリ”AMD”。ここでは作業用の”PythonWorks”の直下に作成した。

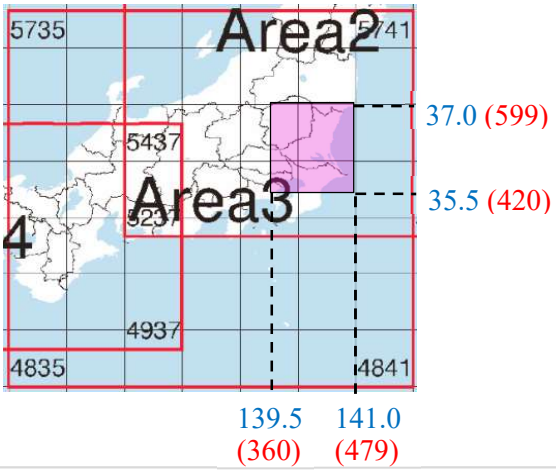

図 7． 茨城県を含む領域（ピンク色の矩形）。青数字は境界の緯度経度、赤数字は領域 Area3 の南端あるいは西端から数えたメッシュ数。

2) データファイルをダウンロードする

データサーバー

http://mesh.dc.affrc.go.jp/opendap/

にアクセスし、目的のエリア・年次のフォルダーに行きます（図 8）。

目当てのファイル（ここではAMD_Area3_TMP_mea.nc）をクリックすると、図 9 の画面が開きます。図内の赤丸部分のチェックボックスをクリックするとボックス内にデフォルト値（全領域）が表示されますので、それぞれを書き換えます。なお、ここではtimeの設定は[0:30]としました。これは 1 月 1 日～1 月31日の意味です。ここで画面最上部にあるAction:の「Get as NetCDF4」をクリックすればファイルを保存するかどうかのダイアログが表れますので、ディレクトリ（./AMD/Area3/2017/）を指定して保存します。ブラウザによっては全てダウンロードフォルダーに保存される場合もありますので、その場合は後で移動します。また、ダウンロードしたファイルのファイル名は、AMD_Area3_TMP_mea.nc.nc4というようによけいなラベルが付いていますので、赤字部分を削除してAMD_Area3_TMP_mea.ncとします。

なお、この例では必要な部分のみを指定しましたが、ネット回線やハードディスクに余裕のある場合は、図 9 の赤丸にチェックだけ入れて、範囲の指定をせずに丸々ダウンロードしても構いません。ただし、たとえばArea3の 1 要素を 1 年分取得した場合のサイズは600MBを超えますので、注意が必要です。また、精度の向上やバグの修正等のために、サーバ上のファイルの数値が書き換えられることも偶にあります。その場合はホームページなどで告知されますので、新しいデータを再度ダウンロードするようにしてください。

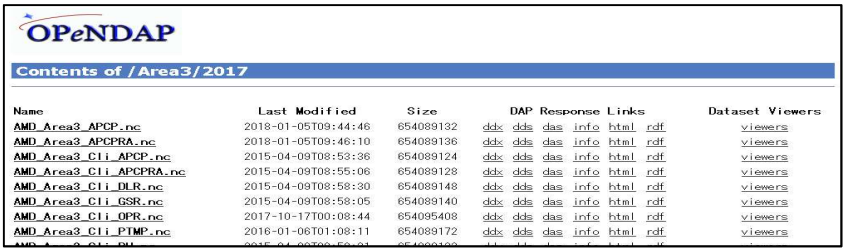

図 8． データサーバ内のデータの並び。ファイル名は ”AMD_エリア名_要素名.nc”。エリア名の後ろに”cli”が入っているのは平年値。

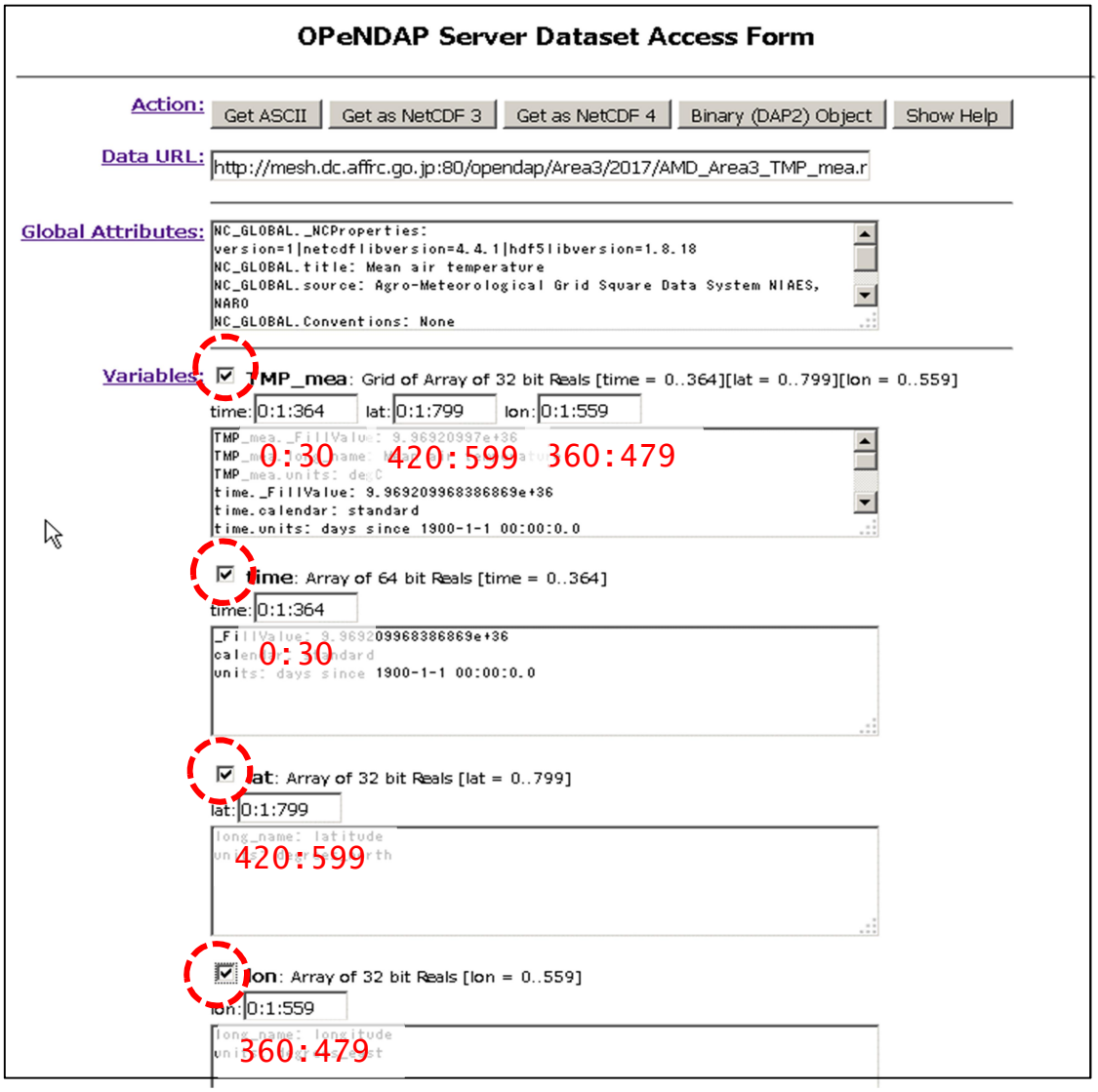

図 9． データファイルのダウンロード設定画面。4 箇所の赤丸部分をチェックして、ボックスに設定値を書き込む。

3) Pythonプログラムの修正

IV_sample_5pyを開きます。このプログラムは冒頭に用いたIV_sample_1.pyと同様に 1 日だけスナップショットを描くものですが、取得範囲は図 7 に合せて[35.5, 37.0, 139.5, 141.0]としています。このプログラムではローカルに保存したデータを読むために、GetMetDataに「area="Area3"」および「url="./AMD"」という赤字部分を加えています。

「url="./AMD"」はデータファイルを保存した場所の指定で、図 6 の”AMD”の場所を指定します（「.」はカレントディレクトリの意味。"D:/MeshData/AMD"などのように絶対指定でも構いません）。

「area="Area3"」は、領域の指定です。データサーバからデータを取得する場合はこの指定は不要ですが、ローカルに保存したファイルを読む場合には指定してください。というのは、図 2 を見るとわかるように、茨城県の領域はArea2にもArea3にも含まれているからです。我々は今回Area3からダウンロードしましたが、GetMetData関数は、エリアの指定がなければArea2のデータを探してエラーを返してしまいます。

```
69: Msh,tim,lat,lon,nam,uni
        =AMD.GetMetData(element, timedomain, lalodomain, area="Area3",
          namuni=True, url="./AMD")
```

4) 切り出し範囲の設定を再利用する

図 9 の「Data URL」ボックスの中身をコピーすると、以下のようになっています。

```
http://mesh.dc.affrc.go.jp:80/
 opendap/Area3/2017/AMD_Area3_TMP_mea.nc?
 TMP_mea[0:30][420:599][360:479],time[0:30],lat[420:599],lon[360:479]
```

これをメモ帳などのエディターに貼り付けて赤字部分の「.nc4」を加えると、データ取得用のURLになります。このURLをブラウザで開くと、そのままデータのダウンロードができます。年次や要素名、取得範囲などの変更がエディター上でできるので、作業効率が上がります。

```
http://mesh.dc.affrc.go.jp:80/
 opendap/Area3/2017/AMD_Area3_TMP_mea.nc.nc4?
 TMP_mea[0:30][420:599][360:479],time[0:30],lat[420:599],lon[360:479]
```