Ⅳ　Python によるデータ処理

プログラミング言語を用いると，平均値や最大値などの統計計算や特定の値を持つメッシュの抜き出しや，何らかの積算値が閾値を超えた日の計測など，データを自在に処理することができるうえ，分布図や折れ線グラフの作成までもおこなえます．そして，同じ処理を毎日更新されるデータに対して自動的に繰り返しすることも簡単にできます．

この章では，プログラミング言語の一つである Python で作成されたいくつかのプログラムを例に，データ配信サーバーからメッシュ農業気象データを取得する方法や処理の方法を解説します．

経験のない者にとってプログラミングはたいへん敷居が高いものですが，Python は初心者にも比較的わかりやすい言語なので，この機会にぜひチャレンジしてみてください．Ⅶ-1には，Python をインストールして利用できるようにするまでの手順が掲載されています．

1　Python プログラムの編集と実行

1）IPython の起動

Python プログラムは，テキストエディターで編集をして IPython コンソールで実行します．Ⅶ-1に従って作成したフォルダ PythonWorks をファイルエクスプローラーで開き，その中のショートカット「それゆけ Python！」をダブルクリックします．すると，図21のような白いウインドウが表示されます．これを IPython コンソールと呼びます．このとき，黒いウインドウが同時に開きますが，これは気にしなくて結構です．邪魔と感じる場合は，タイトルバーの右にある最小化ボタンを押して目立たなくしてください．

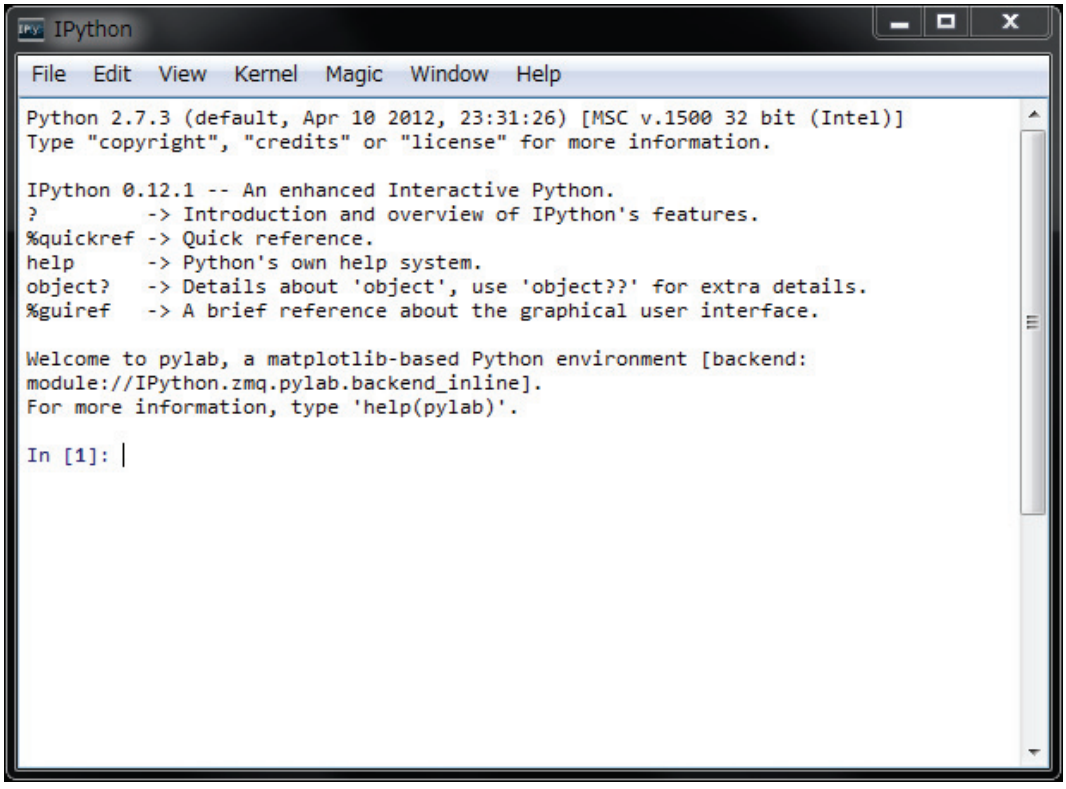

図21．IPython コンソールのスナップショット

Python プログラムをここで実行する．

２）プログラムの実行

IPython コンソールには，「In [1]:」と表示された行があります．この行をプロンプトと呼びます．ここに，「run　プログラムファイル名」と入力してエンターキーを押すと Python プログラムが実行されます（図22）.

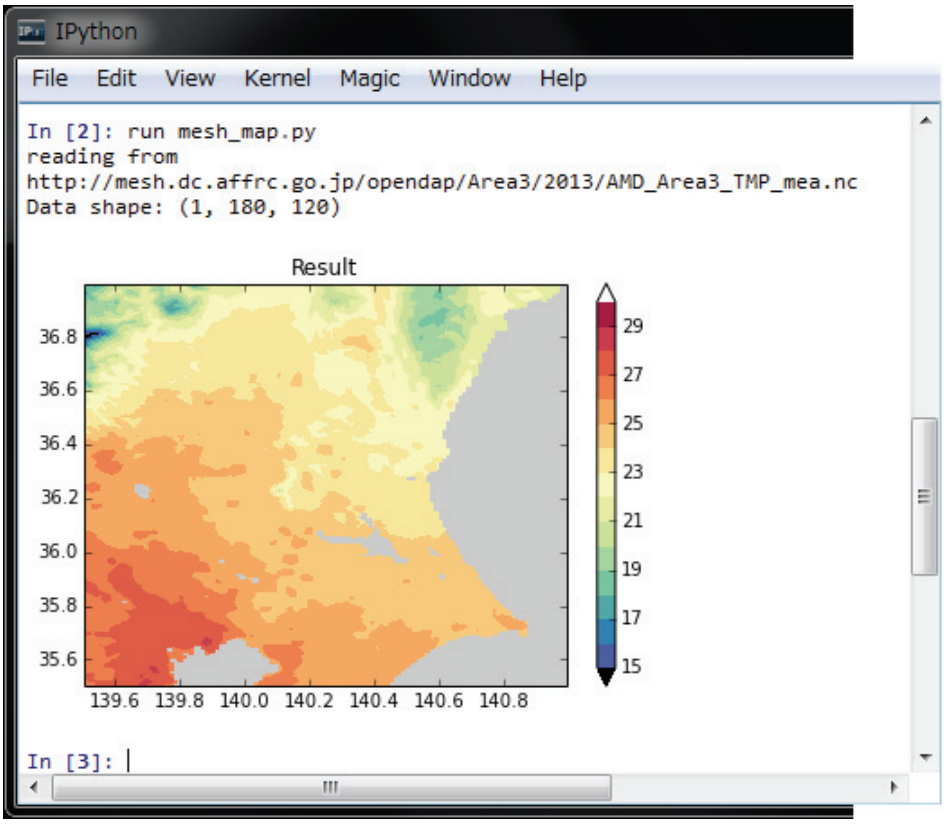

図22.　mesh_map.py の実行結果

BOX 1　メッシュデータは積み上げられた段ボール箱の中

Python プログラムのなかで，メッシュ農業気象データは配列変数に格納されますが，配列変数にデータが格納されている様子は整然と積み上げられた段ボール箱の中の品物に似ています．ここで，品物に相当するのは，特定のメッシュにおける特定の日の特定の気象要素の値　（気温で言えば「23. 0℃」など）です．データを処理する際にはこのような配列変数が複数必要になるので，Ta や Tmax などそれらに名前を付けて識別します．Python は変数名で大文字と小文字を識別します.

メッシュ農業気象データシステムでは，配列変数に個々のデータを格納する順序を決めています．段ボール箱の例えを用いると，南のメッシュのデータが入っている段ボール箱ほど手前に，西のメッシュのデータの箱ほど左に，日付が新しいデータの箱ほど上に積むことに決めています．そして，この山のなかの特定の箱を指し示すときには，必ず，「下から」「手前から」「左から」の順にその箱の場所を数えることにしています．話を配列変数に戻すと，期間初日から i 日目における，南から j 番目，西から k 番目のメッシュのデータは，配列変数の名前を Ta として Ta[i,j,k]と表記されます．i や j のことを添え字「そえじ」と呼びます.

Python では，添え字に整数のほかコロン「：」を使用することができ，「全部」を意味します．したがって，南から j 番目，西から k 番目のメッシュにおける期間全部のデータ（よって普通は複数のデータの集合）を Ta から取り出したいときは Ta[:,i,j]と指定します．なお，Python では配列の順番を０から数える決まりになっています.

さて，メッシュデータのグラフや分布図を描くときには，i や j や k が，実際何月何日で，北緯何度で，東経何度に相当するのかを知っておく必要があります．これらの情報は，GetData（）関数から戻される３つの配列変数 tim，lat，lon に格納されます．つまり，データ Ta[i,j,k]の日付，緯度，経度はそれぞれ，

tim[i]，lat[j]，lon[k]に格納されています．段ボール箱の例えでは，Ta のほかに，tim，lat，lon という一列に並んだ段ボールの塊が３つあって，それぞれの箱の列の i，j，k 番目の中に Ta[i,j,k]に対する日付，緯度，経度が書かれた紙が納められています．

逆に，日付，緯度，経度を決めてデータを取り出したいときは，箱列 tim，lat，lon の中の日付や緯度経度をそれぞれ順に照合して一致や最も近い値を見つけ，それらが入っていた箱の番号（i や j や k）を明らかにしてから，求める箱を取り出します．段ボール箱の山を相手にこの作業をするのは想像するだけでうんざりしますが，コンピュータはこのような作業が少しも苦でないので心配ありません．

３）プログラムの編集

Ⅶ-1に従って構築された環境では，プログラムファイルをダブルクリックすると，専用のテキストエディターが開きプログラムを編集することができます．それでは，2013年８月１日における茨城県周辺の日平均気温の分布図を作成するプログラム「mesh_map.py」（BOX２）を例に編集をします．このプログラムは，利用者 Wiki から入手できます．

BOX２　Python プログラム「mesh_map.py」

```
1  #!/usr/bin/env python
2  # −*− coding: utf−8 −*−
3  #
4  import numpy as np
5  import matplotlib.pylab as plt
6  import AMD_Tools as AMD
7
8  # 領域，期間，気象要素の指定.
9  element = 'TMP_mea'
10 timedomain = [ '2013−08−01', '2013−08−01' ]
11 lalodomain = [ 35.5, 37.0, 139.5, 141.0]
12 area = 'Area3'
13
14 # メッシュ農業気象データの読み込み.
15 temp, tim, lat, lon = AMD.GetData(element, area, timedomain, lalodomain)
16
17 # データの整形（３次元→２次元への変更）.
18 met = temp[0, :, :]
19
20
21
22
23 # 分布図の画面表示と png ファイル出力.
24 fig = plt.figure(num=None, figsize=(6, 4))
25 sclmin = 15.0          #最小値
26 sclmax = 30.0          #最大値
27 sclint =  1.0          #色の刻み
28 plt.axes(axisbg='0.8')   #背景を灰色に
29 levels = np.arange(sclmin, sclmax+sclint, sclint)
30 cmap = plt.cm.Spectral_r   #カラーマップを愛称で指定.「_r」を末尾に付けて反転.
31 cmap.set_over('w', 1.0)
32 cmap.set_under('k', 1.0)
33 CF = plt.contourf(lon, lat, met, levels, cmap=cmap, extend='both')
34 plt.colorbar(CF)
35 plt.title('Result')
36 plt.savefig('result'+'.png', dpi=600)
37 plt.show()
38 plt.clf()
39
40 # 分布図の保存.
41 AMD.PutCSV_Map(met, lat, lon)
42 AMD.PutNC_Map( met, lat, lon, description='None', symbol=element, unit='degC')
```

プログラムに手を加えて，2013年８月１日ではなく，８月９日の温度分布図を描くようにしてみます．それには，10行目の文；

```
timedomain = [ '2013-08-01', '2013-08-01' ]
```

を次のように書き換えます．

```
timedomain = [ '2013-08-09', '2013-08-09' ]
```

編集が完了したら，［Ctrl］＋s または，メニュー File＞Save で変更を保存します．なお，エディターを閉じる必要はありません．保存が終了したら，プログラムを実行します．先ほどと同様，プロンプトに「run mesh_map.py」と入力します．IPython コンソールには，ヒストリー機能とよばれる機能があり，過去に打ち込んだ文字列を記憶しているので，上矢印キー（↑）でこれを呼び出してエンターキーを押してもかまいません．

このように，Python でのプログラム開発は，エディターでの編集と IPython 上での実行結果の確認を繰り返して進めてゆきます．なお，IPython コンソールは，個々の文を実行することができます．また，実行が終了したプログラムで使用されていた変数を保持しているので，その内容等を表示させることもできます．

BOX３　Python について

Python（パイソン）は，使いやすさとグラフィクスの美しさから近年日本でも人気が出てきた，比較的新しいプログラミング言語です．オープンソースなので，だれでも自由に使用することができます．python（ニシキヘビ）という風変わりな名称は，この言語の開発者が，イギリスのテレビ番組「空飛ぶモンティ・パイソン」のファンだったからと言われています．

プログラミング言語は，実行の方式によりインタープリタ型とコンパイル型に大別することができ，Python は前者に属します．インタープリタ型は，プログラムに書かれた命令の一行一行を逐次解釈して実行してゆきます．このため，実行速度が遅い傾向にあり，特に，同じ計算を繰り返す様な処理が苦手です．反面，一行ずつ実行するので「とりあえずここまでプログラムを書いて実行の様子をみよう」というような場当たり的なプログラミングができ，とにかく結果を出したいというようなプログラムに向いています．

一方，コンパイル型は人間が書いたプログラムの全部をコンパイラ（翻訳機）でコンピュータが理解するコードに変換してそれを実行します．この方式は計算をとても早く実行することができる反面，プログラムがきちんとできるまでは動かすことができないので，大規模でしっかりとしたプログラムを作るのに適しています．

２　期間平均した最高気温を計算するプログラムの作成

プログラム「mesh_map.py」を編集して，８月１日から８月31日の期間について平均した日最高気温の分布を計算し，茨城県における分布図を作成する，より実用的なプログラムを作成してみます．

１）別名保存

もともとのファイルが無くなってしまわないよう，ファイルに別名をつけて保存します．mesh_map.py を専用エディターで開いた状態で，メインメニュー File＞Save As …を選択し，「mesh_

mean_map.py」に改名して保存します.

２）平均期間の設定

平均期間である８月１日から８月31日のデータをメッシュ農業気象データ配信サーバーから取得するために，10行目を次のように書き換えます.

```
timedomain = [ '2013−08−01', '2013−08−31' ]
```

３）気象要素の設定

配信サーバーから取り寄せる気象要素を，日平均気温から，日最高気温に変更します．それには，９行目，

```
element = 'TMP_mea'
```

を次のように書き換えます.

```
element = 'TMP_max'
```

平均値計算処理の追加：１月分＝31枚の日別最高気温分布データを時間方向に串刺しのように平均する処理を追加し，平均化された分布を得ます．それには，17－18行目，

```
#　データの整形（３次元から２次元へ変換）.
met = temp [0, :, :]
```

を，次のように書き換えます.

```
#　データを時間方向に（第０番目の次元について）平均する.
met = np. ma. mean (temp, axis=0)
```

「np.ma.mean( )」は，Python の数値計算モジュール numpy に定義される平均値を計算する関数で，第１引数には，計算対象の配列を，第２引数には，平均をする次元を指定します．メッシュ農業気象データは，日付，緯度，経度の３つの次元を持つデータです．このプログラムでは，時間方向の平均操作なので，「axis＝０」として最初の次元について平均することを指示しています．Python では，次元を０から数える約束なので，最初の次元を０で指示します.

４）茨城県以外の消去

メッシュ農業気象データ配信サーバーは，気象データだけでなく，各都道府県について，その都道府県が一部でも含まれる三次メッシュに１，全く含まれないメッシュには無効値が埋め込まれたメッシュデータが用意されています．そこで，このデータを利用して，最高気温の平均値分布図を茨城県だけに限定することにします．そのために，20－21行に次の文を書き入れます.

```
geo, la, lo  = AMD. GetGeoData ('pref_0800', 'Area3', lalodomain)
```

```
met = met * geo
```

初めの文は，配信サーバーから茨城県の領域が１であるメッシュデータを取り出す文です．GetGeoData( )は地理情報を呼び出す関数で，AMD_Tools に定義されています．茨城県の領域が１のデータであるということは文字列「pref_0800」で指定していて，最後の４桁の数字を変更すると，異なる県の領域データが変数 geo に代入されます．続く文で，このデータを分布図にかけ算しています．これで，変数 met の内容は，茨城県内の領域についてはそのままの値，県外の領域については無効値に変更されます．都道府県と数字の関係は表３を参照してください．

５）計算結果の利用

プログラム mesh_mean_map.py は，画面に表示するほか，計算結果を三種類のファイルで保存します．第１は，画像（.png）ファイルで，これは36行目の文で実行されます．第２は CSV ファイルで，42行目で作られます．第３は NetCDF（.nc）ファイルで，43行目で作成されます．このプログラムでは，何れも result.???の名で作成されています．

NetCDF ファイルは，Ⅴ章で解説するデータ可視化ソフト IDV やⅥ章で解説する地図ソフト GMT が直接読み込める特別な形式のファイルです．Result.nc を IDV で開くには，IDV を起動して Dashboard ウインドウの Data　Choosers のページを開き，左端のペイン（ウインドウ内の区切られた領域のこと）で Files が選択されていることを確認してから，右ペインのファイルブラウザから PythonWorks フォルダーを開いて Result.nc を選択し［Add　Source］ボタンを押します．

BOX４　数値計算モジュール numpy

numpy は，Python の機能を拡張するモジュールの１つで，ベクトルや行列などの計算を行うための関数や変数サポートが納められています．「np.ma.mean()」は，平均計算のための関数で，他に「np.ma.max()」（最大値），「np.ma.min()」（最小値），「np.ma.sum()」（積算値），なども提供されています．numpy を利用するには，プログラム冒頭でこのモジュールを使うことを宣言することが必要です．これが，４行目の文；

```
import numpy as np
```

です．この文により，Python は，冒頭が「np.」で始まる関数を，numpy というパッケージから探すようになります．なお，プログラム mesh_map.py は，numpy の他，グラフィックスモジュール matplotlib と，メッシュ農業気象データを利用するためのモジュール AMD_tools を使用しています．

３　指定地点の気象の経過グラフ

プログラム ts.py（BOX５）は，予めプログラム内に記述した地点における気温や降水量などの気象の日変化を，平年値とともに表示します（図23）．このプログラムを例に，Python プログラムの仕組みを解説します．

１）プログラム ts.py の実行

IPython コンソールのプロンプトに「run ts.py TMP_mea」と入力してエンターキーを押すと，プログラムが実行され日平均気温の日々変化が表示されます．このプログラムでは，プログラム名の後に気象要素の略号を付け加えて，プログラムに指示を与えています．このように，プログラム実行の際，追加的に付け加える情報をコマンドライン引数といいます．この場合は，文字列

BOX５　Python プログラム「ts.py」

```
#!/usr/bin/env python
# −*− coding: utf−8 −*−
#
# 特定地点における気象要素の時系列変化をグラフ表示します.
# 表示させる期間は変数"timedomain"に開始日と終了日を文字列で与えます.
# 表示させる地点は変数"lalodomain"に緯度，緯度，経度，経度で指定します.
# 特定の一地点を対象とするため，二つの緯度と経度にはそれぞれ同じ値を
# 指定します.
# IPython からの実行は次のようにします.
#
#   In [1]: run browse.py TMP_mea ＜エンター＞
#
# ここで, "TMP_mea"は，日平均気温を示す記号です．他の気象要素に対する記号は,
# 表を参照してください．また，次のように，気象要素記号の後ろに"−a"を付けると,
# 積算グラフを表示します.
#
#   In [1]: run browse.py APCP −a ＜エンター＞
#
#  気象要素略号
#  日平均気温　　TMP_mea
#  日最高気温　　TMP_max
#  日最低気温　　TMP_min
#  日平均相対湿度　RH
#  日照時間(準備中)  SSD
#  日射量　　GSR
#  下向き長波放射量  DLR
#  日積算降水量　　APCP
#  日平均風速(参考)  WIND
#   OHNO, Hiroyuki 2012.08.20
# ====================================================================
import sys # モジュール属性 argv を取得するため
import numpy as np
import datetime
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import AMD_Tools as AMD

argvs = sys.argv  # コマンドライン引数を格納したリストの取得
argc = len(argvs) # 引数の個数
if argc == 1:
    print 'Please specify a handle of the meteorological element.'
    print 'TMP_mea/TMP_max/TMP_min/RH/SSD/GSR/DLR/APCP/WIND'
    print 'なお，引数「−a」 をさらに追加すると，積算値のグラフを作成します.'
    sys.exit()

# 計算の領域と期間の指定
timedomain = [ '2013−01−01', '2013−12−31' ]
lalodomain = [ 36.0566, 36.0566, 140.125, 140.125]#つくば（舘野）
area = 'Area3'
element = argvs[1]
```

```
51
52 # データの取得
53 T1, tim, lat, lon = AMD.GetData(element, area, timedomain, lalodomain)
54 T0, tim, lat, lon, nam, uni = AMD.GetData(element, area, timedomain, lalodomain, cli=1, namuni=1)
55
56 T0 = T0[:,0,0]
57 T1 = T1[:,0,0]
58 if argc == 3 and argvs[2] == '−a':
59    for i in range(len(T0)):
60       T0[i] = T0[i] + T0[i−1]
61       T1[i] = T1[i] + T1[i−1]
62
63
64 # 表示
65 # ・領域の作成
66 fig = plt.figure(num=None, figsize=(12, 4))
67 # ・目盛の作成
68 ax = plt.axes()
69 xmajoPos = mdates.DayLocator(bymonthday=[1])
70 xmajoFmt = mdates.DateFormatter('%m/%d')
71 ax.xaxis.set_major_locator(xmajoPos)
72 ax.xaxis.set_major_formatter(xmajoFmt)
73 xminoPos = mdates.DayLocator()
74 ax.xaxis.set_minor_locator(xminoPos)
75 # ・データのプロット
76 plt.fill_between(tim,T1,T0,where=T0<T1,color='orange',alpha=0.5)    #高温部を橙色
77 plt.fill_between(tim,T0,T1,where=T1<T0,color='skyblue',alpha=0.5)   #低温部を水色
78 plt.plot(tim, T0, 'k', linewidth=0.3)                               #平年値の線
79 plt.plot(tim, T1, 'k')                                             今年の線
80 # ・「今日」印を付ける
81 p = datetime.datetime.today()                                      #「今日」の時刻オブジェクト
82 today = tim == datetime.datetime(p.year,p.month,p.day,0,0,0)       #今日の配列要素番号
83 plt.plot(tim[today], T1[today], "ro")                              #今日に赤点を打つ
84 # ・タイトルの付加
85 plt.xlabel('Date')
86 plt.ylabel(argvs[1] + ': ' + nam + ' (' + uni + ')')
87 plt.title('N'+str(lalodomain[0])+', E'+str(lalodomain[2])+' ('+str(p.year)+'/'+str(p.month)+'/'+str(p.day)+')')
88 # ・図の保存
89 plt.savefig('Fig_'+argvs[1]+'.png', dpi=300)
90 plt.show()
91
92 # 計算結果の保存
93 Table = np.array([T0,T1])
94 AMD.PutCSV_TS(Table, tim, header='Date,Normal,Obs.')               #CSV ファイル出力
```

であって数ではありませんが TMP_mea がコマンドライン引数です.

２）Python によるデータ処理

プログラム ts.py で実行される処理について，順次解説を加えます.

１〜30行：プログラムの名称や使い方を書いた説明です．Python では，「＃」記号の後ろは無視

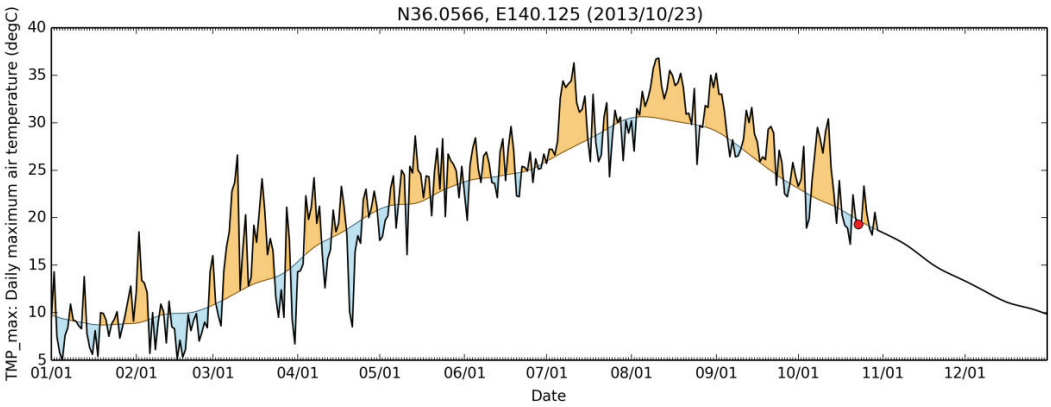

図23. ts.py の実行結果

されるので，＃を書いてからコメントを記述します．プログラムを作成しているときには，動作確認のために特定の文を一時的に無効にしたいときがあります．そのような時にもこれを挿入することがあり，そのような操作を「コメントアウトする」などと言うことがあります．

31〜36行：このプログラムでは，描画や日付計算など，素の Python に組み込まれていない機能を使用するので，import 文を使ってそれぞれの機能を提供するモジュールを組み込みます．このとき，「as」を使用して，モジュール名に別名をつけることができます．

38行：この文によって，引数をプログラム変数 argvs に格納することができます．IPython コンソールは，直前に実行したプログラムの変数の内容を保持しているので，プロンプトに続けて「argvs」と打ち込むとこの変数の中身を見ることができます．

39行：関数 len（）は，括弧の中の変数の個数を求めます．

40〜44行：条件分岐文（if 文）で，if とコロン「：」に挟まれた式が真であればインデントされている44行まで実行し，偽の場合はこれらの文をひとまとめに無視します．なお，44行はプログラムを直ちに終了する命令文です．関数 argv は，最初に入力した文字列 （この場合'ts.py'）を第一番の要素として必ず返すことになっています．そのため，argvs の要素数が１と言うことは，'ts.py'に付随する引数が０個ということです．

47行：日々変化グラフの日付範囲を「yyyy−mm−dd」の形式の文字列で指定します．年を跨る指定もできます．

48行：グラフを作成する地点の緯度と経度を数値で指定します．指定する場所が点の場合は同じ緯度と同じ経度を繰り返し指定します．このプログラムでは，特定の１メッシュにおける気象値を使用しますが，面の場合には緯度と経度の範囲を指定します．

49行：データを取得する地点が属するデータ領域名を文字列で指定します．48行で指定しているのは茨城県にあるアメダスポイント「つくば（館野）」の緯度経度で，この地点は「Area３」に含まれています．

53行：36行でインポートした AMD_Tools モジュールの中の関数 GetData（）を用いてメッシュ農業気象データを配信サーバーから読み込みます．47－50行で指定した期間や緯度経度などの情報がこの関数に与えられています．関数は，気象データ，時間の並び，緯度の並び，経度の並びの４種類のデータを左辺に用意した変数に格納します．

54行：関数 GetData（）は，引数の中に「Cli＝１」というオプション引数が含まれていると，平年値を返します．したがって，変数 T0には平年値が格納されます．また，この関数は，通常は４つのデータを左辺に返しますが，「namuni＝１」というオプション引数が含まれている

と，返されるデータに気象要素名と単位の２つが加わり合計６個になります．このため，それらを受け取るために，左辺には６個の変数が並んでいます．

56，57行：大括弧の中のコロン（：）は「すべての要素」を意味します．メッシュ農業気象データは，時間，緯度，経度からなる３次元「空間」に並べられたデータなので，一般には，何グリッド目の時間，何グリッド目の緯度，何グリッド目の経度という３種類の添え字を指定して気象データを特定します．しかし，このプログラムでは単一メッシュのデータしか扱わないので，緯度と経度についての添え字を識別する必要がありません．これは，建物が１つしかないアパートに「１号棟」という番地情報が不要なのと似ています．これらの文は，３次元の体裁で得たデータを１次元の体裁に変更するために書かれています．

58行：気象データを積算するかしないかを判断する if 文です．引数に「−a」が追加指定されたときは，引数の数が２であり，かつ，２個目（python は０から数える約束）の要素が「−a」であることを条件としています．この条件が成り立つ場合は59行以下を処理します．積算グラフは，降水量を表示する時などに便利です．

59行：気象データを積算する繰り返し計算を指示する文（for 文）で，この文よりもう一段深いインデントの部分を繰り返します．for 文にある「in」の後ろには，数字や文字の列（Python ではリストと呼びます）を置く決まりになっていて，これらのリストが順に変数 i に読み込まれてインデント部分が繰り返されます．range（）は関数で，引数の数の連番のリストを作り出します．注意しなければならないのは，関数 range（）は，引数の一つ手前の数までしか連番を作らないことです．

60行：繰り返し（for 文）の指定により，まず，日付１の気象データに日付０のデータが加えられて日付１に格納し直されます．次の繰り返しでは，日付２のデータに（日付０と日付１の和となっている）日付１のデータが加えられて日付２のデータとして格納されます（つまり日付０，日付１，日付２の和が格納される）．同様にしてその次の繰り返しでは，日付３のデータに日付０～３までの和が格納しなおされます．この繰り返しにより，積算データが作り出されます．

３）Python での作図

プログラム ts.py の64行～90行は，気象データと平年値データをグラフにする部分です．この部分では，図を書く場所を用意し，データ目盛り線や目盛り数値を決め，データの折れ線を描き，プログラム実行日の丸印を打ち，グラフの上にタイトル文字を入れ，図をファイルや画面に出力しています．Python の描画モジュール matplotlib はきわめて表現力に富み，かつ多機能ですが，その分，グラフ各要素を指定する文は複雑難解です．ここでは，大まかな機能を紹介するにとどめます．

66行：横12インチ縦４インチの作図領域を用意します．

69−74行：横軸の目盛りを日付で振ることを指定します．

76行：T0＜T1，すなわち，気象値が平年値よりも大きい部分を薄橙色に着色します．

77行：T1＜T0，すなわち，気象値が平年値よりも小さい部分を薄水色に着色します．

78，79行：T1や T0，すなわち，気象値や平年値の折れ線を黒線で引きます．

80−83行：プログラムを実行した日の気象値の場所に赤丸を打ちます．

85行：横軸の見出しを書き込みます．

86行：気象要素の略号，名称，単位を縦軸の見出に書き込みます．

87行：図のタイトルに，地点の緯度経度，作図年月日を書き込みます．

89行：図を png ファイルで出力します.

93行：365行×１列の配列である T0と T1を結合して365×２列の配列とします.

94行：平年値と気象値を日付とともに CSV ファイルとして出力します.「AMD.PutCSV_TS」は，AMD_Tools.py に書かれている PutCSV_TS（）という関数を呼び出すことを意味しています.

BOX６　Python とインデント

文の行頭を字下げ（インデント）して，プログラムを見やすくすることは，多くのプログラム言語で慣用となっていますが，Python ではループなどひとまとまりで扱うべき文をインデントそのもので表現します．たとえば，５の階乗（１×２×･･×５）を計算するプログラムは，次のように書かれます.

```
k = 1
for i in [1, 2, 3, 4, 5]:
  k = k * i
print k
```

この例では，繰り返し計算の中身は３行目だけで，これが５回繰り返されてから４行目に移ります．どこからどこまでを繰り返すかは同じインデントがどこからどこまで施されているかで決まります.「end for」など，ループの終わりを示す文はありません．従って，次のように，４行目をインデントすると，画面には，計算途中の値が５回表示されるようになります.

```
k = 1
for i in [1, 2, 3, 4, 5]:
  k = k * i
  print k
```

インデントには半角空白とタブが使用できますが，両者は絶対に混用しないようにして下さい．両者が混ざっていると，人の目からは同じインデントでありながら Python には異なるインデントと見なされエラーとなります.

一方で，python は括弧の途中では自由に改行やインデントができます．たとえば，関数にたくさんの引数があり，一行で書くととても長くなるようなとき，見やすくなるよう，括弧のなかの適当な場所で改行し字下げをすることができます.

```
T0, tim, lat, lon, nam, uni = AMD.GetData(element, area,
                timedomain, lalodomain, cli=1, namuni=1)
```

４．茨城県における水稲の発育を推定するプログラム

１）DVI/DVR 法

DVI/DVR 法とは，作物の発育段階を，出芽を０，成熟を２とする発育指数（DVI）で表現する方法で，DVI は日々の気象条件から計算される発育速度（DVR）を出芽日以降積算したものです．この方法は，水稲の出穂日や収穫日を推定するのにとても便利なので，DVR の式を栽培試験から独自に作成したり，DVI の起点を出芽ではなく移植日として定義するなど，地域の実情にあわせた様々なアレンジが加えられて普及や指導の現場で広く使われています.

２）プログラム RiceDevelopment.py の概要

プログラム RiceDevelopment.py（BOX７）は，DVI/DVR 法に基づいて，県下の水稲の発育

を予測し，出穂日の分布図，成熟日の分布図を作成するとともに，日々のDVIの値をNetCDFファイルとして出力します．このプログラムでは，DVIの定義を，出芽で０，出穂で１，成熟で２としています．そして，気象条件からDVRを計算する際，移植～出穂の期間と出穂～成熟の期間とで異なる計算式を用いています．また，実際には，水稲の出芽日や移植日が県下で同一ということはありませんが，「苗は同一日に移植される」，「移植時の苗のDVIは県下で同一である」という二つの仮定をおいて分布図を作成しています．

プログラムの実行は，IPythonのプロンプトに「run　RiceDevelopment.py」と打ち込み，エンターキーを押します．すると，しばらく計算した後に，出穂日の分布図と成熟日の分布図が図24のように表示されます．あわせてこれらの画像ファイルと，DVI分布の日々の変化を記録したNetCDFファイルが出力されます．

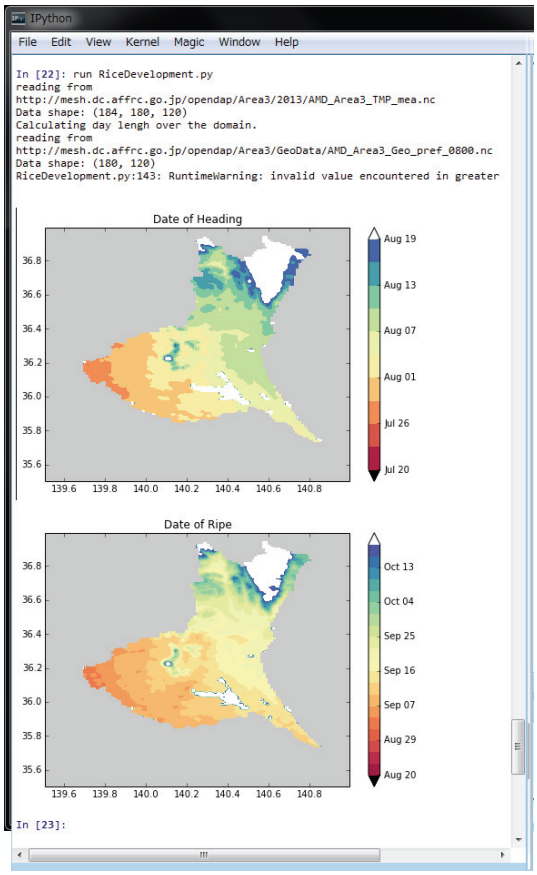

図24.　RiceDevelopment.pyの実行結果

IPythonコンソール上に分布図が描画される．

実際の利用あたっては，移植日と移植時のDVI値，それに，DVRの式を適切に与える必要があります．移植日はプログラム55行で設定します．この文は計算期間を設定する文で，計算開始日を移植日に設定します．計算終了日については，大まかに予想される収穫日の数週間後を目処に設定します．移植時のDVI値は，プログラム52行で指定します．DVRの式の与え方については，次の節で詳しく説明します．

３）独自のDVR関数を使用する方法

DVI/DVR法では，作物の発育の進行を気象条件から計算したDVRという量で表現します．DVRは，地域や品種に応じて様々なものが考案され使用されています．したがって，DVI/DVR法で作物の生長を推定するプログラムでは，DVRを計算する式を柔軟に入れ替えられるようにしておくと便利です．これを実現するプログラミングの方法の一つに，関数の定義という方法があります．これは，特定の計算や処理を行うプログラムを本体とは切り離してプログラミングして名前をつけておき，プログラム本体でそれを呼んで所定の計算や処理をさせるものです．

関数は下のような書式で定義します；

```
def　関数名（引数１，引数２，…）：
  計算文
  計算文
    :
  return　戻り値が入っている変数
```

ここで，引数（ひきすう）とは，呼ぶ側のプログラムからの数値を受け取る変数のことで，２

通りの表記法があります．一つは，引数となる変数だけが書かれているもの，もう一つは引数となる変数に等号と値が付随するものです．前者のように引数を書くと，それは，関数を呼ぶ際に必ず与えなければならない変数値となります．後者のように書くと，関数を呼ぶ際にこの引数は値を指定してもしなくてもよい引数（キーワード引数）となり，省略されて関数が呼び出された場合には等号の右側の値がセットされます．

プログラム RiceDevelopment.py では，７行～30行で出芽から出穂までを受け持つ DVR 関数を DVR01 () という名前で定義し，32行～45行で出穂から成熟までを受け持つ DVR 関数を DVR 12 () という名前で定義しています．７行目を見てわかるとおり，関数 DVR01 () は DVI，Ta，Ld という３個の通常の引数と Para という１個のオプション引数を持ちます．この関数は，49行目から始まるプログラム本体の中の93行目で呼び出されています．呼び出される際，プログラム本体で使用している DVI，Tmea，Ld という変数の値がこの順で関数 DVR01 () の DVI，Ta，Ld に引き渡されます．キーワード引数 para は指定されていないので，７行に記されているデフォルト値が用いられます．ここで，プログラム本体で使用している変数の名前と，通常の引数の名前は一致している必要がないことに注意してください．関数への値の引き継ぎには，数と順序だけが意味を持ちます．これに対し，キーワード引数は省略ができるものの，使用するときは定義で使われている名前を使用する必要があります．

７行に記されている para のデフォルト値は，この DVR 関数のコシヒカリに対するパラメータです．したがって，この関数を使用して他の水稲品種の発育を推定する場合は，次のようにしてキーワード引数 para にその品種に対するパラメータ値を明示的に与える必要があります．

```
DVR = DVR01(DVI[t−1, i, j], Tmea[t, i, j], Ld[t, i, j], Para=[51.3, 17.8, 0.365, 16.0, 0.566, 0.23])
```

プログラム RiceDevelopment.py において，気象データを取得したり DVR を積算したり，グラフを描いたりするプログラムの本体は，48行以降にあります．プログラムの本体も一塊のプログラムですから，プログラミング言語によっては，main という名で定義することがありますが，Python では，def 文を使用せず，if 文を使います．これは Python の少々風変わりなところですが，このような扱いにすることで，RiceDevelopment.py をそのままモジュールとして DVR 関数を使用する他のプログラムにインポートして使うことができます．

５．モジュール AMD_tools

モジュール AMD_tools は，メッシュ気象データの利用に必要な関数（計算で使う道具）のコレクションで，ファイル AMD_tools.py を作業ディレクトリに置き，プログラムでこれをインポートすることで使用可能となります．モジュールには，メッシュ農業気象データを処理するのに便利な９個の関数が定義されていています．

１）GetData（）

概要：メッシュ農業気象データをデータ配信サーバーまたはローカルファイルから取得する関数．

書式１：ret1, ret2, ret3, ret4 = GetData (element, area, timedomain, lalodomain, cli=0, url='http://mesh.dc.affrc.go.jp/opendap')

書式２：ret1, ret2, ret3, ret4, ret5, ret6 = GetData (element, area, timedomain, lalodomain, namuni=1, url='http://mesh.dc.affrc.go.jp/opendap')

引数：

BOX７　Python プログラム「RiceDevelopment.py」

（誌面の都合上，折り返している行がありますが，実際は１行で記述します.）

```
#! c:/Python27/python.exe
# −*− coding: utf−8 −*−
#
import numpy as np

# 独自定義の DVR 関数−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
def DVR01(DVI, Ta, Ld, Para=[51.3,17.8,0.365,16.0,0.566,0.23] ):
#          Para=[ Gv0, Th ,  A , Lc ,  B ,DVI*]
#   Function name:    'Horie et al. (1995)'
#   Varid phase: emergence to heading
#   Description:
#     Gv0 [days]: the minimum number of days required for heading (GV)
#     Th [C]: the temperature at which DVR is half the maximun rate at the optimum temperature (TH)
#     A [1/C]: empilocal Parameter on air temperature (ALF)
#     Lc [hour]: critical day length (LC)
#     B [1/hour]: empilocal Parameter on day length (BDL)
#     DVI* [ ]: DVI at which the crop becomes photosensitive (DVSAS).
#     For KoshHikari;
#       Gv0 ,   Th ,    A ,    Lc ,     B ,  DVI*
#       51.30, 17.80, 0.365, 16.00, 0.566, 0.230
#
#                      Gv0                 A          Th
    FT = np.max( 1.0 / (Para[0] * (1.0 + np.exp(−Para[2] * (Ta − Para[1])))), 0.0)
#                           B             Lc
    FL = np.max((1.0 − np.exp(Para[4] * (Ld − Para[3]))), 0.0)
    DVR = FT
#           DVIs
    if DVI > Para[5] :
        DVR = FT * FL
        return DVR

def DVR12(Ta, Para=[0.0, 1000.0]):
#           Para=[ T0,  EDDd ]
#    Function name:    'Normalized Effective Degree Days'
#   Varid phase: not specified
#    Description:
#     T0 [degC]: threshold temperature of development
#     EDDd [degC day]: desired effective degree days
#     For the symple cumlative temperature of 1000[degC day];
#       T0,  EDDt
#       0.0, 1000.0
#
#              T0       EDDd
    DVR = (Ta − Para[0]) / Para[1]
    return DVR


# プログラムのメイン関数−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
if __name__ == "__main__":

```

```
51    #移植時 DVI の指定
52    DVI0 = 0.2
53
54    #計算の領域と期間の指定
55    timedomain = [ '2013-05-01', '2013-10-31' ] #計算の開始日を移植日と見なす.
56    lalodomain = [  35.5,  37.0,  139.5, 141.0]
57    area = 'Area3'
58    pref = 'pref_0800'  #茨城県
59
60    #各種データの準備
61    import AMD_Tools as AMD        #メッシュデータ用モジュールをインポート.
62    import DayLength as DL         #日長を計算するモジュールをインポート.
63  #   Tmea, tim, lat, lon = AMD.GetData('TMP_mea', area, timedomain, lalodomain, url='./AMD') #USB
    メモリに保存されている気象データを使用するときはこちらを使い，下の行は消去します.
64    Tmea, tim, lat, lon = AMD.GetData('TMP_mea', area, timedomain, lalodomain)  #配信サーバーの気
    象データを使用するときはこちらを使い，上の行は消去します.
65    Tmea[Tmea.mask == True] = np.nan
66    Ld = DL.daylength(tim, lat, lon)   #モジュール daylengthd で対象時空間全部の日長を計算する.
67    Pref, lat, lon = AMD.GetGeoData(pref, area, lalodomain, url='./AMD')   #USB メモリに保存されて
    いる県域データを使用するときはこちらを使い，下の行は消去します.
68  #   Pref, lat, lon = AMD.GetGeoData(pref, area, lalodomain)  #配信サーバーかの県域データを使用す
    るときはこちらを使い，上の行は消去します.
69
70    #計算結果を保存する配列の定義
71    frst = np.datetime64(timedomain[0])   #期間の初日を数値化したもの
72    last = np.datetime64(timedomain[1])   #期間の最終日を数値化したもの
73    ntim = Tmea.shape[0]   #日数
74    nlat = Tmea.shape[1]   #メッシュの行数
75    nlon = Tmea.shape[2]   #メッシュの列数
76    DVI = np.ma.zeros((ntim, nlat, nlon))   #DVI の時空間分布を記録する配列（気象データと同じ時空
    間サイズ）を確保する.
77    DVI[:,Pref==0.0] = np.ma.masked   #県外にマスクをかける.
78    DVI[:,:,:] = DVI0              #全領域に初期値を代入.
79    DOH = np.ma.zeros((nlat, nlon), dtype='datetime64[D]')   #出穂日（期間の初日からの日数）をしま
    う配列.
80    DOH[Pref==0.0] = np.ma.masked   #県外にマスクをかける.
81    np.ma.harden_mask(DOH)         #マスクを"固く"して以降の計算でも変化しないようにする.
82    DOH[:,:] = last                #全領域に期間の最終日を入れておく.
83    DOR = np.ma.zeros((nlat, nlon), dtype='datetime64[D]')   #収穫適日（期間の初日からの日数）をし
    まう配列.
84    DOR[Pref==0.0] = np.ma.masked   #県外にマスクをかける.
85    np.ma.harden_mask(DOR)
86    DOR[:,:] = last
87
88    #各メッシュにおける DVI の計算
89    for i in range(nlat):   #メッシュ行（緯度方向）
90      for j in range(nlon):   #メッシュ列（経度方向）
91        for t in range(1, ntim):   #日数
92          if DVI[t-1,i,j] < 1.0:   #移植から出穂までは…
93            DVR = DVR01(DVI[t-1,i,j], Tmea[t,i,j], Ld[t,i,j])
94            DVI[t,i,j] = DVI[t-1,i,j] + DVR
95            if DVI[t,i,j] >= 1.0:   #出穂日ならば…
96              DOH[i,j] =  frst + np.timedelta64(t,'D')   #日付を記録する.
```

```
97              else:                   #出穂から成熟までは…
98                DVR = DVR12(Tmea[t−1,i,j])
99                DVI[t,i,j] = DVI[t−1,i,j] + DVR
100               if DVI[t−1,i,j] < 2.0 and DVI[t,i,j] >= 2.0:   #収穫適日ならば…
101                 DOR[i,j] = frst + np.timedelta64(t,'D')      #日付を記録する.
102
103     DVI[np.ma.where(DVI > 2.0)] = 2.0    #全体を見直してみて，DVI が2より大きいメッシュがあればそれを2.0に置きかえてしまう.
104   #   DVI[DVI>2.0] = 2.0 でも同じ.
105     # 計算結果の描画
106     import matplotlib.pyplot as plt
107     from matplotlib.dates import DateFormatter,DayLocator
108     import matplotlib.pylab as plt
109     import matplotlib.colors as clr
110     aspect = (lalodomain[3] − lalodomain[2]) / (lalodomain[1] − lalodomain[0]) + 0.5
111   #   一枚目の図_____
112     fig = plt.figure(num=None, figsize=(5*aspect, 5))  #図のオブジェクト（入れ物）を定義
113     sclint = 3                          #カラーバーの刻み
114     sclmin = np.datetime64('2013−07−21')       #カラーバーの最小値
115     sclmax = np.datetime64('2013−08−20')       #カラーバーの最大値
116     levels = np.arange(sclmin, sclmax+np.timedelta64(sclint,'D'), sclint)
117     plt.axes(axisbg='0.8')          #背景を灰色に
118     cmap = plt.cm.Spectral       #カラーマップを愛称で指定.「_r」を末尾に付けて反転.
119     cmap.set_over('w', 1.0)       #上限を超えたときの色
120     cmap.set_under('k', 1.0)      #下限を超えたときの色
121     CF = plt.contourf(lon, lat, DOH, levels, cmap=cmap, extend='both')   #分布図を描く
122     CB = plt.colorbar(CF, format=DateFormatter('%b %d'))        #カラーバーを描く
123     plt.title('Date of Heading')                    #タイトルを書く
124     plt.savefig('Date_of_Heading.png', dpi=600)        #図をビットマップ画像にする
125     plt.show(fig)                       #図を表示する
126     plt.clf()
127   #   二枚目の図_____
128     fig = plt.figure(num=None, figsize=(5*aspect, 5))
129     sclint = 3
130     sclmin = np.datetime64('2013−08−21')
131     sclmax = np.datetime64('2013−10−20')
132     plt.axes(axisbg='0.8')
133     levels = np.arange(sclmin, sclmax+np.timedelta64(sclint,'D'), sclint)
134     cmap = plt.cm.Spectral
135     cmap.set_over('w', 1.0)
136     cmap.set_under('k', 1.0)
137     CF = plt.contourf(lon, lat, DOR, levels, cmap=cmap, extend='both')
138     CB = plt.colorbar(CF, format=DateFormatter('%b %d'))
139     plt.title('Date of Ripe')
140     plt.savefig('Date_of_Ripen.png', dpi=600)
141     plt.show(fig)
142     plt.clf()
143
144     # 計算結果をファイルに保存します.
145     AMD.PutNC_3D( DVI, tim, lat, lon, description='Rice Development Index',
146                   symbol='DVI', unit='−', filename='DVI.nc')
```

element：気象要素記号で，'TMP_mea'などの文字列で与える
area：データの領域で，'Area3'などの文字列で与える
timedomain：取得するデータの時間範囲で，［'2008−05−05'，'2008−05−05'］のような文字列の２要素リストで与える．特定の日のデータを取得するときは，二カ所に同じ日付を与える．
lalodomain：取得するデータの緯度と経度の範囲で，［36.0，40.0，130.0，135.0］のように緯度経度の順で指定する．特定地点のデータを取得するときは，緯度と経度にそれぞれ同じ値を与える．
cli：平年値のデータを取得するとき cli＝１として与える．１以外の数値であったり，このパラメータそのものが省略されている場合は，観測値が返される．
namuni：変数の名前と単位を取得するとき namuni＝１として与える．このとき，関数の戻り値の数は２つ増えて６つになる．１以外の数値であったり，このパラメータそのものが省略されている場合は，戻り値は４つ（変数，時刻，緯度，経度）である．
url：データファイルの場所を指定する．省略した場合はデータ配信サーバーに読みに行く．ローカルにあるファイルを指定するときはディレクトリー構造をデータ配信サーバーと同一（図３）とし，AreaN（N＝１～６）の上までの場所（通常は"…/AMD"）を指定する．
戻り値：
ret１：指定した気象要素のマスク付きの三次元データ．［時刻，緯度，経度］の次元を持つ．なお，マスクとは対象とする／しないを示す配列変数の付随情報のことで，メッシュ農業気象データシステムでは水域などデータのある／なしを表すのに用いています．
ret２：切り出した気象データの時刻の並び．Python の時刻オブジェクトの一次元配列である．時刻オブジェクトとは，Python で時刻を表現するために使用される特別な形式のデータで実数ではない．このため，Excel の時刻連番のように，１を足して翌日を表現するというようなことはできない．
ret３：切り出した気象データの緯度の並び．実数の一次元配列である．
ret４：切り出した気象データの経度の並び．実数の一次元配列である．
ret５：オプション引数「namuni＝１」が指定されたときに限り，気象要素の名称の文字列が返される．
ret６：オプション引数「namuni＝１」が指定されたときに限り，気象要素の単位の文字列が返される．
使用例１：以下により，北緯35度，東経135度の地点の2008年１月１日～2012年12月31日の日最高気温が一次元の配列変数 Tm に格納される．関数 GetData（）は，特定単一メッシュにおける複数日の日別値を取得する場合でも三次元配列を返すので，これを一次元配列に変換するために「Tm = Tm3D [:, 0, 0]」が実行されている．

```
import numpy as np
import AMD_Tools as AMD
timedomain = ['2008−01−01', '2012−12−31']
lalodomain = [35.0, 35.0, 135.0, 135.0]
Tm3D, tim, lat, lon = AMD.GetData('TMP_max', 'Area4', timedomain, lalodomain)
Tm = Tm3D[:,0,0]
```

使用例２：以下により，北緯35～36度，東経135～136度の範囲地点における日最高気温の日別平

年値の分布が三次元配列変数 Tmo に格納される．平年値であるから2011年の10月１日～12月31日と2012年10月１日～12月31のデータについては互いに等しい値が格納される一方，時刻オブジェクト配列 tim は，2011年10月１日から2012年12月31日までのすべて異なる値の日付オブジェクトが格納される．

```
import numpy as np
import AMD_Tools as AMD
timedomain = ['2011-10-01', '2012-12-31']
lalodomain = [35.0, 36.0, 135.0, 136.0]
Tmo, tim, lat, lon = AMD.GetData('TMP_max', 'Area4', timedomain, lalodomain, cli=1)
```

使用例３：以下により，北緯35～36度，東経135～136度の範囲地点における日最高気温の日別分布が，データ配信サーバーではなく，コンピュータの D:￥data￥以下に置かれたデータファイルから取得される．この際，D:￥data￥以下のディレクトリ構造は，データ配信サーバーと同じでなければならない．

```
import numpy as np
import AMD_Tools as AMD
td = ['2011-10-01', '2012-12-31']
lalo = [35.0, 36.0, 135.0, 136.0]
ele = 'TMP_max'
Tmo, tim, lat, lon = AMD.GetData(ele, 'Area4', td, lalo, url='D:￥data￥')
```

２）GetGeoData（）

概要：土地利用や都道府県などの地理情報をデータ配信サーバーまたはローカルファイルから取得する関数.

書式１：ret1, ret2, ret3 = GetGeoData(element, area, lalodomain, url='http://mesh.dc.affrc.go.jp/opendap')

書式３：ret1, ret2, ret3, ret4, ret5 = GetGeoData(element, area, timedomain, lalodomain, namuni=1, url='http://mesh.dc.affrc.go.jp/opendap')

引数：

element：地理情報の記号で, 'landuse_H210100'などの文字列で与える.

area：データの領域で, 'Area３'などの文字列で与える

lalodomain：取得するデータの緯度と経度の範囲で，［36.0，40.0，130.0，135.0］のように緯度経度の順で指定する．特定地点のデータを取得するときは，緯度と経度にそれぞれ同じ値を与える.

namuni：変数の名前と単位を取得するとき namuni＝１として与える．このとき，関数の戻り値の数は２つ増えて６つになる．１以外の数値であったり，このパラメータそのものが省略されている場合は，戻り値は４つ（変数，時刻，緯度，経度）である.

url：データファイルの場所を指定する．省略した場合はデータ配信サーバーに読みに行く．ローカルにあるファイルを指定するときは，AreaN（N＝１～６）の上までの場所を指定する.

戻り値：

ret１：指定した地理情報のマスク付きの二次元データ．［緯度，経度］の次元を持つ．
ret２：切り出した気象データの緯度の並び．実数の一次元配列である．
ret３：切り出した気象データの経度の並び．実数の一次元配列である．
ret４：オプション引数「namuni＝１」が指定されたときに限り，地理情報の名称の文字列が返される．
ret５：オプション引数「namuni＝１」が指定されたときに限り，地理情報の単位の文字列が返される．
使用例１：以下により，北緯35～36，東経135～136度の範囲にある各メッシュの水田面積比率の分布が取得される．

```
import numpy as np
import AMD_Tools as AMD
lalodomain = [35.0, 36.0, 135.0, 136.0]
Pad, lat, lon = AMD.GetGeoData(’landuse_H210100’’, ’Area4’, lalodomain)
```

使用例２：以下により，北緯35～36度，東経135～136度の範囲にある各メッシュの水田面積比率の分布が取得される．データは，データ配信サーバーではなく，コンピュータの D:￥data￥以下に置かれたデータファイルから取得される．この際，D:￥data￥以下のディレクトリ構造は，データ配信サーバーのそれ（図３）と同じでなければならない．地理情報は日々更新されないので，よく使うものをローカルストレージに保存しておくとネットワークの負荷を軽減できる．

```
import numpy as np
import AMD_Tools as AMD
lalo = [35.0, 36.0, 135.0, 136.0]
handle = ’’landuse_H210100’
Pad, tim, lat, lon = AMD.GetGeoData(handle, ’Area4’, lalo, url=’D:¥data¥’)
```

３）GetCSV（）

概要：CSV 形式のテキストファイルを配列変数に読み込む関数．配列の列数は取り込み範囲の先頭行で判別され，行数は EOF までの行数から判別する．文字列“nan”は，numpy.nan（数値ではないと理解される特別な数値）として理解し，これ以外の文字列が検出されると警告文を表示する．
書式：ret1 = GetCSV(filename, skiprow=0, fill=9.96921e+36)
引数：
filename：読み込むべき CSV ファイルの名前．
skiprow：余白や見出しなどに使用されていて読み込み対象としない行の数．指定を省略した場合は０に設定される（デフォルト値）．
fill：無効値として使用する数値．この値の配列要素にはマスクがかけられる．指定を省略した場合は9. 96921 e ＋36に設定される．
戻り値：ret１：指定した fill で指定した値のメッシュにマスクがかけられた二次元データ．
使用例１： CSV ファイル data.csv が BOX８のように与えられているとき，下を実行すると，結果は図25のように出力される．大括弧 ’[’ の付き方から，CSV ファイルの内容は，５行４

列の配列 arr に代入されていることがわかる（小数点以下８位以降に変換に伴う誤差が発生している）.

```
import numpy as np
import AMD_Tools as AMD
fn = 'data.csv'
arr = AMD.GetCSV(fn, skiprow=1, fill=−999.9)
print arr
```

BOX８　テキストファイル data.csv

```
lat, lon, dat1, dat2
35.0, 135.0, 1.5, 230.0
35.0, 135.0, 1.7, 253.6
35.0, 135.0, 2.2, nan
35.0, 135.0, 1.4, −999.9
35.0, 135.0, 0.9, 260.3
```

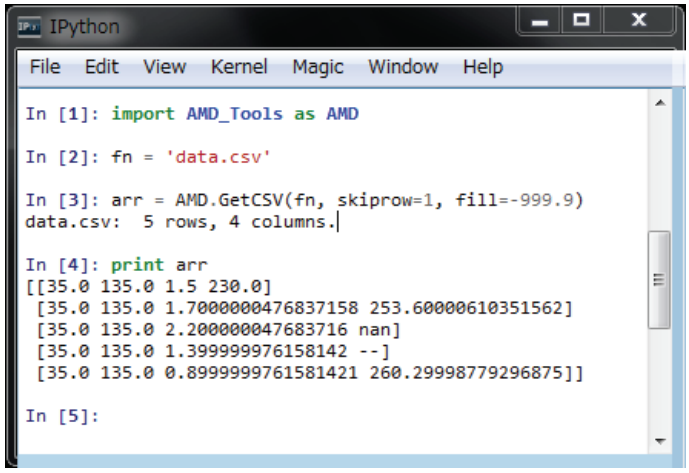

図25.　関数 Get_CSV () の使用例

文字から数値への変換に際し若干の誤差が発生している.

４）PutNC_Map（）

概要：２次元の気象変量（平面上に分布する気象要素）を NetCDF 形式のファイルで出力する関数.

書式：PutNC_Map(Var, lat, lon, description='Variable', symbol='Var', unit='−−', fill=9.96921e+36, filename='result.nc')

引数：

Var：気象変量として書き出す２次元配列変数．Var［緯度の次元，経度の次元］でデータが並んでいること.

lat：気象変量の各要素が並ぶ緯度を示す１次元配列．Var の最初の次元の要素数と一致していなくてはならない.

lon：気象変量の各要素が並ぶ経度を示す１次元配列．Var の２番目の次元の要素数と一致していなくてはならない.

description：気象変量の名前等を description = '名前等' として指定する．指定を省略した場合は 'Variable' という名で出力される.

symbol：気象変量の記号を symbol = '記号' として指定する．指定を省略した場合は 'Var' が使用される.

unit：気象変量の記号を unit = '単位の記号' として指定する．指定を省略した場合は '−−' が使用される.

fill：無効値として使用する数値を fill＝数値で指定する．指定を省略した場合は9. 96921e＋36に設定される．

filename：出力される NetCDF ファイルのファイル名を filename＝'ファイル名'として指定する．指定を省略した場合は 'result.nc' という名で出力される．

戻り値：戻り値はない．

使用例：以下により，データ配信サーバーから北緯35～36度，東経135～136度の範囲における水田占有率データを取得し，PaddyMap.nc という名の NetCDF ファイルとして出力する．

```
import AMD_Tools as AMD
import numpy as np
lalodomain = [35.0, 36.0, 135.0, 136.0]
Pad, lat, lon = AMD.GetGeoData('landuse_H210100'', 'Area4', lalodomain)
AMD.PutNC_Map( Pad, lat, lon, description='Ratio of paddy land', symbol='Rpad',
        unit='%', fill=9.96921e+36, filename='PaddyMap.nc')
```

5）PutNC_3D（）

概要：３次元の気象変量（時空間上に分布する気象要素）を NetCDF 形式のファイルで出力する関数．

書式：PutNC_3D(Var, tim, lat, lon, description='None', symbol='Var',
unit='−−', fill=9.96921e+36, filename='result.nc')

引数：

Var：気象変量として書き出す３次元配列変数．Var［時刻の次元，緯度の次元，経度の次元］でデータが並んでいること．

tim：気象変量の各要素が並ぶ時刻を示す時刻オブジェクトの１次元配列．Var の最初の次元の要素数と一致していなくてはならない．

lat：気象変量の各要素が並ぶ緯度を示す１次元配列．Var の２番目の次元の要素数と一致していなくてはならない．

lon：気象変量の各要素が並ぶ経度を示す１次元配列．Var の３番目の次元の要素数と一致していなくてはならない．

description：気象変量の名前等を description＝'名前等' として指定する．指定を省略した場合は' Variable' という名で出力される．

symbol：気象変量の記号を symbol＝ '記号' として指定する．指定を省略した場合は 'Var' が使用される．

unit：気象変量の記号を unit＝ '単位の記号' として指定する．指定を省略した場合は '−−' が使用される．

fill：無効値として使用する数値を fill＝数値で指定する．指定を省略した場合は9. 96921 e ＋36に設定される．

filename：出力される NetCDF ファイルのファイル名を filename ＝ 'ファイル名' として指定する．指定を省略した場合は 'result.nc' という名で出力される．

戻り値：戻り値はない．

使用例：　以下により，データ配信サーバーから北緯35～36度，東経135～136度の範囲における2008年～2012年の日最高気温データを取得し，MaxTemp.nc という名の NetCDF ファイルと

して出力する．無効値にはデフォルトが用いられる．

```
import AMD_Tools as AMD
import numpy as np
timedomain = ['2008−01−01', '2012−12−31']
lalodomain = [35.0, 36.0, 135.0, 136.0]
Tm, tim, lat, lon = AMD.GetData('TMP_max', 'Area4', timedomain, lalodomain)
AMD.PutNC_3D( Tm, tim, lat, lon, description='Maxmun air temperature', symbol='Tmax',
        unit='degC', filename='MaxTemp.nc')
```

６）PutCSV_TS（）

概要：時刻順に並ぶ配列を，行方向にデータが並ぶ CSV ファイルとして出力する関数.

書式：PutCSV_TS(Var, tim, header=None, filename='result.csv')

引数：

Var：CSV ファイルに書き出す１次元配列変数．Var の第０次元の要素数は，tim の要素数と一致していること.

tim：気象変量の各要素が並ぶ時刻を示す時刻オブジェクトの１次元配列．Var の第０次元の要素数と一致していなくてはならない.

header：CSV ファイルに行見出しを与える時に，header = '列見出し' として指定する．この際，CSV の書式に基づいて，文字列を与える．指定を省略した場合は見出しは付けられない.

filename：出力される CSV ファイルのファイル名を filename＝ 'ファイル名' として指定する．指定を省略した場合は 'result.csv' という名で出力される.

戻り値：戻り値はない.

使用例：以下により，データ配信サーバーから，北緯35度，東経135度の地点における2008年の日平均気温と降水量を取得し，日付，気温，降水量，の順に３列に並ぶ CSV ファイルとしてデフォルトのファイル名で出力する.

```
import AMD_Tools as AMD
import numpy as np
timedomain = ['2008−01−01', '2008−12−31']
lalodomain = [35.0, 35.0, 135.0, 135.0]
Ta3D, tim, lat, lon = AMD.GetData('TMP_mea', 'Area4', timedomain, lalodomain)
Ta = Ta3D[:,0,0]
Pr3D, tim, lat, lon = AMD.GetData('APCP', 'Area4', timedomain, lalodomain)
Pr = Pr3D[:,0,0]
tapr = np.array([Ta, Pr])
AMD.PutCSV_TS(tapr, tim, header='Date, Ta, Pr')
```

７）PutCSV_MT（）

概要：３次元の配列を，３次メッシュコードをキーとするテーブルの形式の CSV ファイルで出力する関数．第１次元（緯度），第２次元（経度）が同じ第０次元の内容を添え字の順に記号で区切って出力する．３次メッシュコードを属性に持つメッシュのポリゴンデータを GIS に

整備しておくと，GIS 上でこのファイルとポリゴンをリンクするにより，データの分布図を GIS 上で簡単に表示することができる．

書式：PutCSV_MT(Dat, lat, lon, addlalo=False, header=None, filename='result.csv', removenan=True, delimiter=',')

戻り値：なし

引数：

Dat：書き出すべき３次元配列変数.

lat：配列 Dat の各行が位置する緯度値が格納されている配列.Dat の第１次元の要素数と一致していなくてはならない.

lon：配列 Dat の各列が位置する経度値が格納されている配列.Dat の第２次元の要素数と一致していなくてはならない.

addlalo：これを True にすると，３次メッシュ中心点の緯度と経度が出力ファイルの第２フィールドと第３フィールド追加挿入される．デフォルトは False であり挿入されない.

header：一行目に見出しやタイトルなど何か書き出すときはここに「header＝ '文字列'」として指定する.

filename：出力されるファイルの名前．デフォルト値は 'result.csv' .

delimiter：フィールドの区切り文字．デフォルト値は ',' ．すなわち，CSV ファイルとなる.

removenan：無効値だけのレコードを削除するかを指定するキーワード．値は 'True'.しないときは 'removenan=False'とする.

使用例：以下により，データ配信サーバーから，新潟県における2013年８月１日～10の日平均気温を取得し，メッシュ別に「メッシュコード，緯度，経度，８月１日の気温，８月２日の気温，…，８月10日の気温」の順で並ぶテキストファイルを作成することができる.

```
import AMD_Tools as AMD
import numpy as np
timedomain = [ '2013-08-01', '2013-08-10' ]
lalodomain = [36.7, 38.6, 137.6, 140.0]
dat, tim, lat, lon = AMD.GetData('TMP_mea','Area2',timedomain, lalodomain)
pref,    lat, lon = AMD.GetGeoData('pref_1500', 'Area2', lalodomain)
dat = dat * pref
hd = 'MeshID, latitude, longitude'
for t in range(len(tim)):
  hd = hd + ',' + str(tim[t])
AMD.PutCSV_MT(dat, lat, lon, addlalo=True, header=hd )
```

８）PutCSV_Map（）

概要：２次元の配列変数を，緯度を行方向に，経度を列方向に配置する CSV ファイルとして出力する関数．第１行には経度の数値，第１列には緯度の数値が見出しとして出力される．この際，緯度は北が上になるように出力する．また，無効値には文字列 nan が代入される.

書式：PutCSV_Map(Var, lat, lon, filename='result.csv')

引数：

Var：CSV ファイルに書き出す１次元配列変数．Var の最初の次元の要素数は lat の要素数と一

致していること.

lat：配列の各要素が並ぶ緯度を示す１次元配列．Var の最初の次元の要素数と一致していなくてはならない.

lon：気象変量の各要素が並ぶ経度を示す１次元配列．Var の２番目の次元の要素数と一致していなくてはならない.

filename：出力される CSV ファイルのファイル名を filename ＝ 'ファイル名' として指定する．指定を省略した場合は 'result.csv' という名で出力される.

戻り値：戻り値はない.

使用例：以下により，データ配信サーバーから，2008年１月１日の佐渡島周辺おける日平均気温の分布を取得し，CSV ファイルとしてデフォルトのファイル名で出力する（図26）.

```
import AMD_Tools as AMD
import numpy as np
timedomain = ['2008−01−01', '2008−01−01']
lalodomain = [37.7, 38.4, 138.2, 138.6]
Ta3D, tim, lat, lon = AMD.GetData('TMP_mea', 'Area2', timedomain, lalodomain)
Ta2D = Ta3D[0,:,:]
AMD.PutCSV_Map(Ta2D, lat, lon)
```

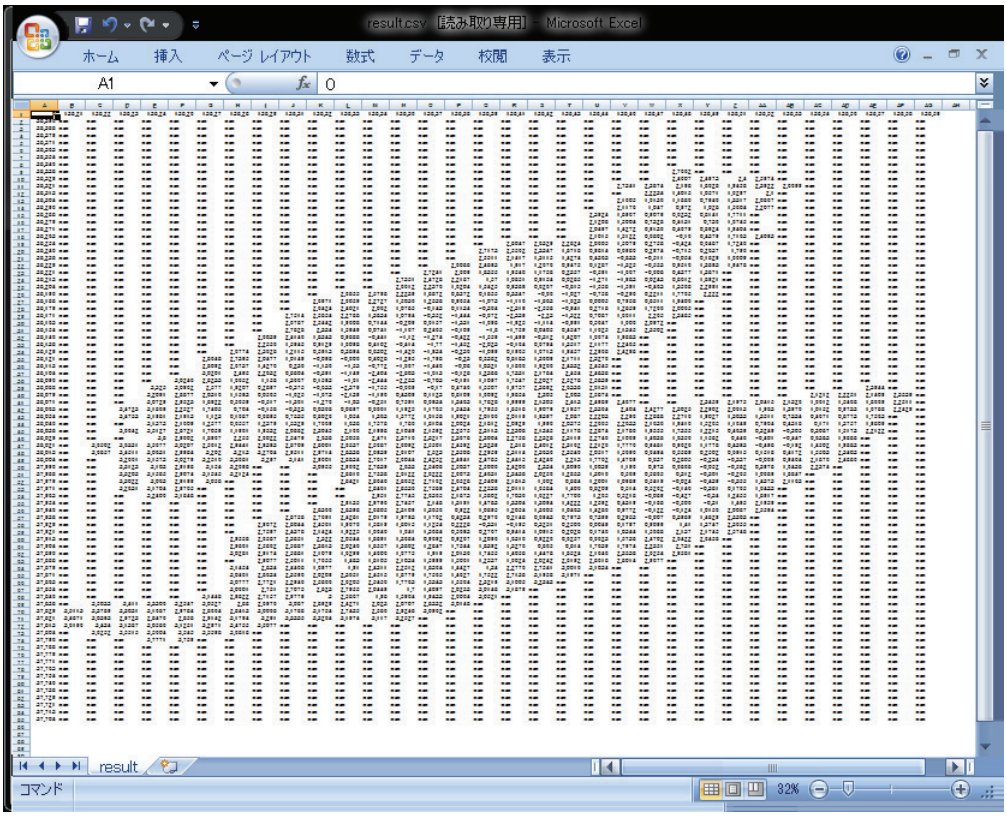

図26.　関数 PutCSV_Map () で出力した CSV ファイルを Excel で表示した画面

佐渡島周辺の平均気温分布が取得されている.

BOX９　NetCDF ファイルについて

NetCDF ファイルは，気象データなどの地球科学的なデータを納めることを目的として Unidata というプロジェクトが策定したファイル形式です．とても複雑なのでこのファイルの読み書きをするには，言語毎に作られているツールを利用するのが普通です．データの各要素が持つ時刻，緯度，経度，高度をはじめとして，データの名前，単位，無効値，数値型，作者などまでもが規約に基づいて整然と格納されているので，これを前提にプログラムを組むと，データのサイズや無効値を予め調べてプログラムの定義文に書き込んだり，月の大小や閏年によるややこしい条件分岐をプログラミングしたりする必要がなくなり，プログラムをとてもシンプルにすることができます．以上の理由から，この手引きでは，農業気象データの処理結果を NetCDF ファイルの形で保存しています．

９）accumulation_of_effective_temperature（）

概要：引数に気温の三次元配列を取り，これをもとに気温有効積算温度を計算する関数．引数として入力する配列と，戻り値として出力される配列のサイズは同一である．戻り値の配列における時間方向に n 番目の（２次元）要素には，１日目から n 日目までの積算値が格納されている．例えば，ret[6, :, :] は，期間の７日目における有効積算温度分布を意味する．

書式：ret = accumulation_of_effective_temperature(Var, To=0.0)

引数：

Var：有効積算温度を計算するもととなる気温の時空間分布データの配列．

To：基準温度を To= ’温度（℃）’ として指定する．指定を省略した場合は０℃が与えられる．

戻り値：

ret：有効積算温度を計算するもととなる気温と同じサイズの３次元配列．戻り値の配列における時間方向に n 番目の（２次元）要素には，１日目から n 日目までの有効積算温度が格納されている．

使用例：以下により，北緯35～36度，東経135～136度の範囲における2013年８月１日を起日とし，基準温度を５℃とする有効積算気温の一ヶ月間の推移を計算し，NetCDF 形式のファイルとして出力する．この結果は，IDV で可視化することができる．

```
import AMD_Tools as AMD
import numpy as np
timedomain = [’2013−08−01’, ’2013−08−31’]
lalodomain = [35.0, 36.0, 135.0, 136.0]
Ta, tim, lat, lon = AMD.GetData(’TMP_mea’, ’Area4’, timedomain, lalodomain)
Tacc = AMD.accumulation_of_effective_temperature( Ta, To=5.0 )
AMD.PutNC_3D( Tacc, tim, lat, lon, description=’Effective Degree Day Temperature’,
symbol=’DDT’,  unit=’degC day’, filename=’DDT_Aug.nc’)
```

Ⅴ　IDV を用いたデータの可視化

第Ⅳ章で見たとおり，プログラミング言語 Python は品質の高いグラフィクスを作成することができますが，そのためには難解な書式設定の文をたくさん書かなければなりません．定番の図として繰り返し使用するものはそれでもいいのですが，数枚しか作成しない図の作成にプログラ