第 218 回農林交流センターワークショップ

メッシュ農業気象データ利用講習会
テキスト

令和元年 6 月 27 日(木)～ 6 月 28 日(金)

国立研究開発法人農業・食品産業技術総合研究機構（農研機構）

第 1 研究本館 「大会議室」 （つくば市観音台 3-1-1）

目　次

【講義 1】メッシュ農業気象データの利用について

農研機構 農業環境変動研究センター　佐々木華織

はじめに

農研機構では、水稲における白未熟粒や胴割れ粒、果樹の着色不良、眠り病など、深刻さを増す高温による減収や品質低下に対応する技術や、増加する小規模・分散・多数圃場営農の効率化を支援するために、作物や品種、栽培期間を複雑に組み合わせる技術の開発を進めています。同時に、これらの技術が要求するより高度な気象データへの需要に対応できる気象データサービスの開発にも取り組み、気象予測を含む日別気象データを作成・配信する「メッシュ農業気象データシステム」を構築しました。

メッシュ農業気象データシステムに搭載されるデータ

メッシュ農業気象データシステムは、日別気象データをオンデマンドでサービスするシステムです。標高や土地利用などを考慮しつつ気象庁の気象データを補間して約 1km 四方(基準地域メッシュ)を単位に全国の日別気象データを作成します。どのメッシュについても、観測値、最長 26 日先までの気象予報、平年値がシームレスに接続され、1980 年 1 月 1 日から来年の 12 月 31 日までをカバーするデータが整備されています。図 1 に、2017 年 7 月 8 日にシステムが配信した茨城県内のあるメッシュにおけるこの年の日平均気温データを示します。図中にイラストで示したように、メッシュ農業気象データは作物の全栽培期間をカバーするので、収穫適期などを最新の気象データに基づいて予測することができます。また、1980 年(一部 2008 年)に至る過去データを使用すれば、栽培に適した作物や品種、栽培期間を検討することもできます。

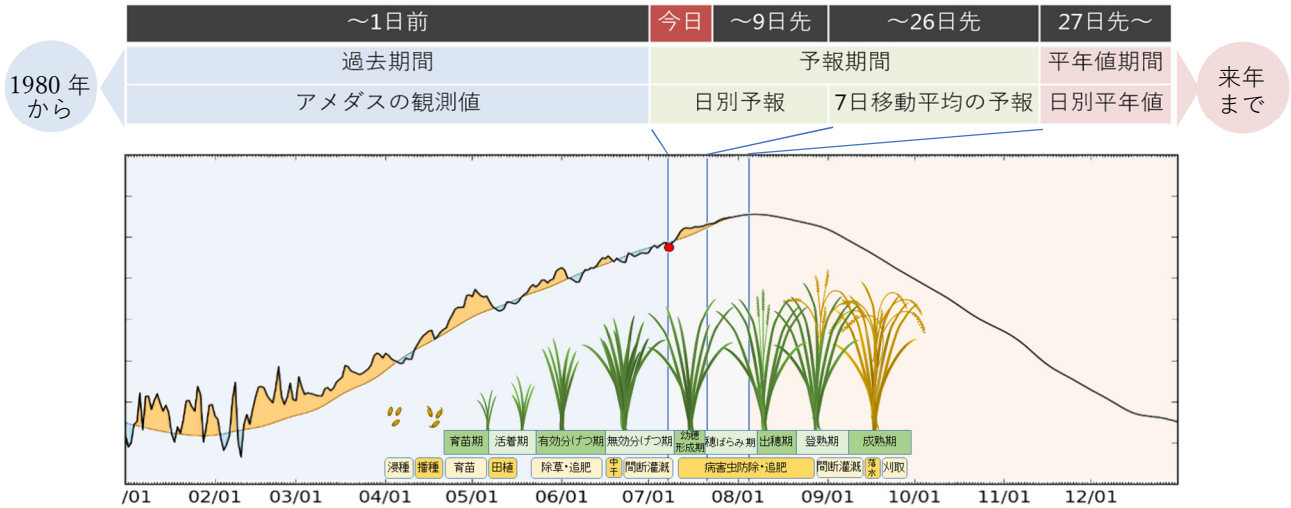


図 1. 2017 年 7 月 8 日にメッシュ農業気象データシステムから取得した、茨城県内のあるメッシュにおけるこの年の日平均気温データのグラフ。この地域で標準的な水稲の生育をイラストで示す。

【講義 1】メッシュ農業気象データの利用について

提供する気象要素は、表 1 に示す 14 種類で、相対湿度や積雪水量など、アメダスでは観測されていない気象要素も研究成果に基づき独自に作成し提供しています。さらに、メッシュ農業気象データシステムには、気象データのほか、メッシュの平均標高や都道府県範囲など、基本的な地理情報も提供されています。地理情報と気象データを組み合わせると、例えば、特定の県だけの分布図の作成や、水田が分布する地域だけの平均気温の計算などが行えます。2018 年度より、気候シナリオデータに基づくデータの配信も開始しました。

表 1.　メッシュ農業気象データシステムに搭載される日別気象値および日別平年値の一覧。

気象要素	記号	単位	日別気象値			日別平年値
			過去期間	予報期間	平年値期間	
日平均気温	TMP_mea	℃	1980年1月～	～26日先	~1年後	2011年～1年後
日最高気温	TMP_max	℃	1980年1月～	～26日先	~1年後	2011年～1年後
日最低気温	TMP_min	℃	1980年1月～	～26日先	~1年後	2011年～1年後
日積算降水量	APCP[1)] APCPRA[2)]	mm/day	1980年1月～ 2008年1月～	～26日先	~1年後	2011年～1年後
1mm以上の降水の有無	OPR	0(無)～ 1(有)	1980年1月～	～9日先	~1年後	2011年～1年後
日照時間	SSD	h/day	1980年1月～	なし	~1年後	2011年～1年後
全天日射量	GSR	$MJ/m^2/day$	1980年1月～	なし	~1年後	2011年～1年後
下向き長波放射量	DLR	$MJ/m^2/day$	2008年1月～	なし	なし	なし
日平均相対湿度	RH	%	2008年1月～	～9日先	なし	なし
日平均風速	WIND	m/s	2008年1月～	～9日先	なし	なし
積雪深	SD	cm	1980年10月～	～9日先	なし	なし
積雪相当水量	SWE	mm	1980年10月～	～9日先	なし	なし
日降雪相当水量	SFW	mm/day	1980年10月～	～9日先	なし	なし
予報気温の確からしさ[3]	PTMP	℃	なし	～26日先	なし	なし

1）アメダスベースの過去値
2）解析雨量ベースの過去値
3）気温予報値の標準偏差近似値

【講義 1】メッシュ農業気象データの利用について

メッシュ農業気象データの詳細

メッシュ農業気象データはメッシュ農業気象データシステムから提供されるデータの総称で、4 種類のデータからなります。いずれのデータも、基準地域メッシュ（3 次メッシュ）に準拠し、海や湖沼を除く全国のメッシュについて整備されています。

1）メッシュ日別気象値

メッシュ日別気象値は、メッシュ毎に整備されている日別気象データで、一部のデータを除き、1980 年 1 月 1 日から来年の 12 月 31 日までの期間が収録されています。この期間は、収録期間の始めから今日の 1 日前までの「過去期間」と、今日から最長 26 日先までの「予報期間」、その翌日から収録期間の終わりまでの「平年値期間」からなります。図 1 に、メッシュ農業気象データシステムが 2017 年 7 月 8 日に配信した、あるメッシュにおける 2017 年 1 年分の日平均気温データのグラフを示します。この例では、7 月 7 日以前が過去期間、7 月 8 日から 8 月 3 日までが予報期間、8 月 4 日以降が平年値期間です。予報期間のうち、9 日先までの予報は、気象庁の数値予報モデル GPV に基づいて行われます。そして、10 日先以降の予報については、1 か月予報ガイダンスと呼ばれる別な気象庁資料に基づいて行われています。前者は日別予報を提供しますが、後者は、7 日を単位とする予報を提供します。この関係で、メッシュ日別気象値も、9 日先までの予報は日別で、10 日先以降については前後 3 日間の期間を持つ移動平均値が、それぞれの日に与えられています。つまり、10 日先から最長 26 日先までについては、データの形式は日別ですが、それぞれの日のデータはその日 1 日の気象値を示しているわけではありません。従って、メッシュ日別気象値では、10 日先以降の日積算降水量が決してゼロと予報されないので、注意してください。病害の発生予察など、降水の有無が重要となる用途に利用する場合は、別途作成されている「1mm 以上の降水の有無」データセットを併用してください。

平年値期間におけるメッシュ日別気象値は、各メッシュに対して推定される日別平年値が与えられています。日積算降水量については、常にゼロでない数値です。平年値が存在しない気象要素に対しては無効値が与えられています。

メッシュ日別気象値における個々の気象要素の整備期間、予報期間の長さ、日別平年値の有無については、表 1 を参照してください。

メッシュ農業気象データは、最新の観測値や予報値に基づいて 1 日 1 回、平日の午前 8 時ごろに更新されています。休日(土・日曜日、休日、年末年始)は更新されません。また、10 日先の予報は火曜日と金曜日、11 日先から 26 日先までの予報は金曜日にのみ行われます。

メッシュ毎の値をどのように計算しているかについては、以下の文献を参照してください。

大野宏之、佐々木華織、大原源二、中園　江（2016）「実況値と予報値、平年値を組み合わせたメッシュ気温・降水量データの作成」、生物と気象、16、71-79。

2）メッシュ日別平年値

日平均気温、日最高気温、日最低気温、日積算降水量、1mm 以上の降水の有無、日照時間、全天日射量については、日別平年値がメッシュ毎に整備されています。2019 年現在のメッシュ日別平年値は、気象庁のメッシュ平年値 2010 他に基づいて作成されていて、2011 年～2020 年の期間において年による違いはありません。また、メッシュ日別気象値の平年値期間のデータは、メッシュ日別平年値のデータと同一です。

3）地理情報

メッシュの面積、平均標高、土地利用割合、所属都道府県が、メッシュ毎に整備されています。気象データとこれらを組み合わせることで、たとえば、標高がメッシュ平均標高とは相当程度異なる特定地点の気温を推定することや、特定県における気温分布図を作成すること、特定領域における降水の総量を推定することなどが行えます。

4）メッシュ気候変化シナリオ

システムには、全球気候モデル MRI-CGCM3 ならびに MIROC5 を用いて、現在気候（1981～2005 年）および温暖化ガス排出シナリオ RCP 8.5、および、RCP 2.6 に基づく将来気候予測（2006～2055 年）を 1km メッシュにダウンスケーリングした気候変化シナリオデータも搭載されています。データ形式を現在気象のデータと揃えてあるので、現在気象向けに開発した解析プログラムを温暖化影響評価に有効活用することができます。図 2 は、メッシュ気候変化シナリオ(MIROC5, RCP8.5)データから、茨城県つくば市の 2050 年における日最高気温(黒太線)を取り出し、現在の平年値(黒細線)と対比して示したものです。なお、ここではユーザーがプログラムの簡単な変更でデータを取得できることを示すため重ねて示してありますが、気候変化シナリオは温暖化予測専用の全球気候モデルが、現在のカレンダーと無関係に計算しているものであり、現実とは異なります。あくまで近い将来の気候予測値として、仮想的に生成されたデータであることに留意してご使用下さい。

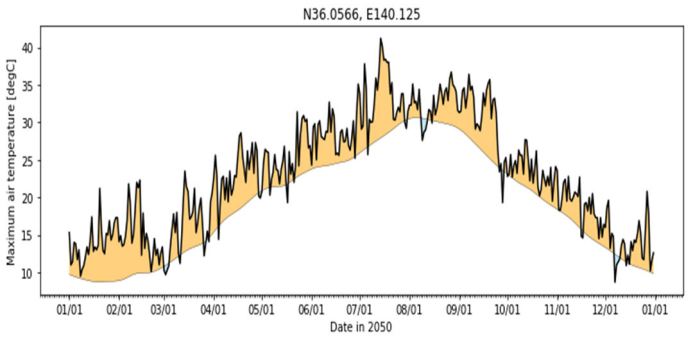


図 2．茨城県つくば市の 2050 年における日最高気温(黒太線)と現在の平年値(黒細線)

【講義 1】メッシュ農業気象データの利用について

メッシュ農業気象データシステムのデータ配信機能

14 気象要素、全国約 40 万メッシュ、40 年約 15000 日の膨大なデータをメッシュ農業気象データシステムは管理しますが（表 1）、利用者はこの中から必要とする気象要素、期間、領域のデータをオンデマンドで取得することができます。

もっとも簡単な方法は、農研機構が提供するデータ取得のためのマイクロソフトエクセルファイルを使う方法です。図 3 のように、気象要素、年次、緯度、経度、をセルに書き込んでボタンをクリックするだけで 1 年分の気象データをシート上に取得することができます。取得したデータを参照する計算式を作れば、最新の気象予測に基づく任意の演算をワンクリックで実行することができます。

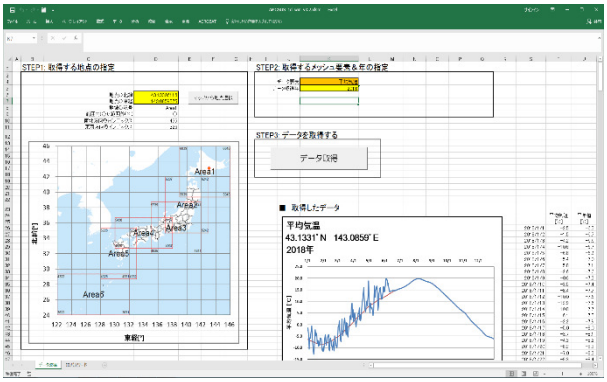
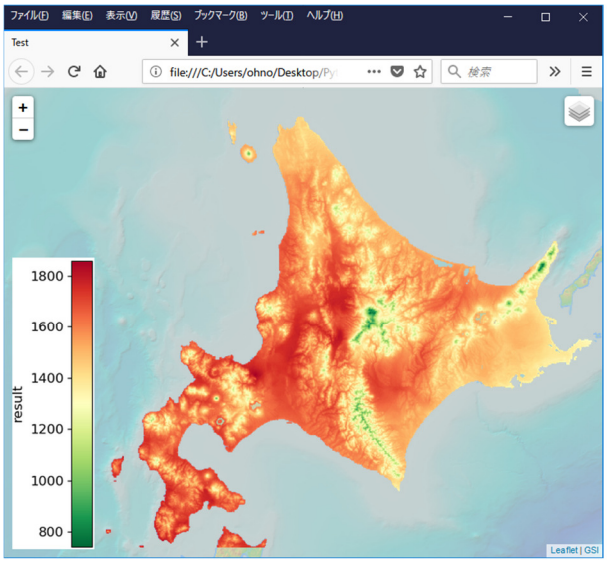


図 3．データ配信サーバーから 1 年分の気象データを取得するマイクロソフトエクセルファイル。ボタンをクリックするだけで最新データが取得できる。

オープンソースのプログラミング言語 Python を利用すれば、気象データをより自在に処理することができます。図 4 (左)は、北海道における 2017 年 6 月～8 月の有効積算気温の分布図で、コメント行も含めてたった 31 行の Python プログラムで作成されました（図 4 (右)）。とはいえ、多くの農業関係者にとって、プログラミングは決して身近ではないので、農研機構ではメッシュ農業気象データの処理に便利な関数やサンプルプログラムを利用者に提供し、利用を支援しています。

メッシュ農業気象データシステムは、最新の観測値や予報値に基づいてデータを毎日更新していますが、2011 年以降、これらをすべてアーカイブとして保存しており、この間に提供したデータを任意の日について再現することができます。メッシュ農業気象データを処理する Python プログラムは、簡単な操作で入力データを再現データに切り替えることができるので、提供される予報データの精度検証や、予報に基づく農業情報の有効性の検証を効率よく行うことができます。




```
# -*- coding: utf-8 -*-
"""
北海道における2017年6～8月の有効積算気温の分布図を計算し、
国土地理院の地図サービス上に表示できるファイルを作成する
プログラム。
積算期間は変数kikan(12行目)に与える、
基準温度は変数To(16行目)に与える。

"""
import numpy as np
import AMD_Tools3 as AMD
# 基本的な設定
youso = 'TMP_mea'              #気象要素の指定。TMP_meaは日平均気温。
kikan = [ "2017-06-01", "2017-08-31" ] #期間の設定。
area = [41.350,45.533, 139.327,146.000]   #領域の設定。北海道がすっぽり収まる範囲。
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)

# 気象データの読み込み
Ta,tim,lat,lon = AMD.GetMetData(youso, kikan, area)

# 有効積算温度の計算
valimesh = ~np.isnan(Ta)
Ta[valimesh] = np.where(Ta[valimesh] > To , Ta[valimesh]-To, 0.0)
Tacc = np.sum(Ta, 0)

# 地理院地図にオーバーレイするファイルの出力
AMD.PutGSI_Map(Tacc,lat,lon)
[EOF]
```

図 4．　左：北海道における有効積算気温分布図。国土地理院地図上に表示できる。右：この分布図を計算する Python プログラム。

【講義 1】メッシュ農業気象データの利用について

メッシュ農業気象データの精度

メッシュ農業気象データの使用により、観測値と平年値を接続する従来の方法(気候値予報)が示す予報誤差が何パーセント低減できるかを誤差低減の効果と定義し、予報日数との関係を 2011 年～2015 年の気象データから計算した結果を図 5 に示します。誤差低減の効果は、日別に評価すると 7 日先程度で消失します。一般に、日別予報の限界は 7 日程度といわれており、メッシュ農業気象データもこれと同様の特性を持ちます（図 5 青線)。しかし、誤差低減の効果を平均値または積算値で評価すると、それはより長い期間まで認められます（図 5 赤線)。これには、1 か月予報ガイダンス等の気象庁の気候予測情報が寄与しています。これまでの研究から、作物の発育は、気温の積算と強い相関を持つことが明らかになっているので、メッシュ農業気象データの使用は、作物の発育予測精度向上に有効と考えられます。メッシュ農業気象データシステムと発育予測モデルとを組み合わせて、国内の任意の地点における作物の発育を最新の気象データに基づいて随時予測するプログラムを実行した結果が図 6 です。これは、北海道十勝地方における小麦の出穂日を、出穂の 2 か月前から毎日予測したものです（図 6 橙線)。図に併せて示したのは、観測値と平年値を接合した従来データを同じプログラムに与えた結果です（図 6 黒破線)。いずれの気象データも最終的には正しい出穂日を導きますが、気象予報が導入されているメッシュ農業気象データでより速やかに正しい出穂日に近づき、観測値と平年値を接続する従来データに比較すると 2 週間程度早くから正しい出穂日を予測することが確かめられました。

より詳しい精度については、「メッシュ農業気象データの詳細」「1）メッシュ日別気象値」にある文献、大野他（2016）を参照してください。

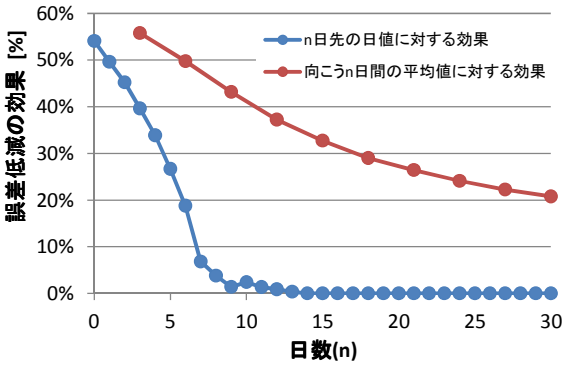


図 5 .平年値をもって予測値とする場合に対する誤差低減の効果((EC-EF)/EC)と予報日数との関係。ただし、EF は予測値誤差（RMSE）、 EC は気候値予測の誤差(2011～2015 年)。

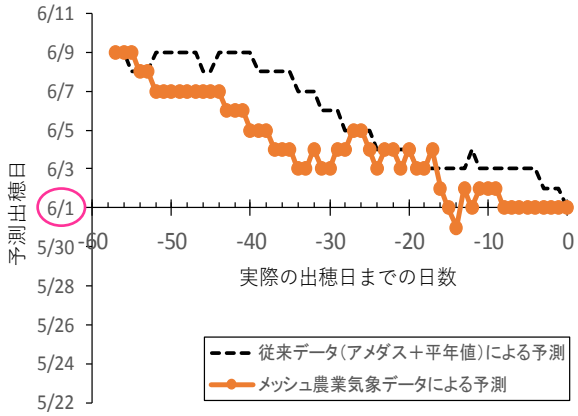


図 6 . 北海道十勝地方における 2016 年産小麦（きたほなみ）の発育予測結果。実際に観察された出穂日は 6 月 1 日。

【講義 1】メッシュ農業気象データの利用について

メッシュ農業気象データシステムの利用状況

メッシュ農業気象データシステムは 2012 年に試験的な配信を開始しました。当初、利用登録数は 10 数件程度でしたが、年を追うごとに登録数が増え、2019 年 3 月末の時点で、633 件です。メッシュ農業気象データの利用登録は、毎年、利用目的毎に受付け、同一目的の範囲で複数人での利用を認めているので、利用人数はこの数倍となります。2018 年度より ID、パスワード認証対応の新システムとなり、登録も紙方式からオンライン方式となりました。全般的にみると、北日本や東日本の方が、南日本や西日本よりも利用登録数が多い傾向で、これは、緯度が高い地方ほど農業生産に気象が与える影響が相対的に大きいことを反映していると考えられます。利用登録毎の所属を見ると、多い順に、公設農業試験場および普及関係機関、農研機構、民間企業、大学となっており、農業法人・生産者の利用も増えてきました。民間企業を業種で分類すると、情報系が最も多く、農業、農業資材、食品と続きます。民間企業の登録数の伸びが大きいことと、業種が幅広くなってきたのが近年の特徴です。特に近年、企業や大学において、農業、気象情報システム開発への利用が次第に多く見られるようになりました。各種研究会や研究開発プロジェクトにおける機会を利用して行った普及活動の効果が考えられますが、2016 年の「気象ビジネスコンソーシアム」発足に見られるように、気象情報と AI や IoT と結び付けて新しいビジネスにつなげようとする我が国の大きな流れを反映した結果とも考えられます。

利用目的をみると、農業生産における解決すべき課題がまずあって、それに気象データの利用を検討する利用者がほとんどです。一方、農業利用とはしているものの作物等が特定されていない利用、また農業外、不特定の利用の場合は、気象データから何らかの利用方法を考えようとする利用者とみられます。

前者の場合、作物の発育や生長、収穫適期、障害等の予測が多くを占め、特に水稲での利用が多くなっています。これらはメッシュ農業気象データに組み込まれている気象予測を利用するものですが、一方でそれと同程度に過去の事例解析にも利用されており、作物の栽培適地や栽培適期の把握にも利用されていることが分かります。メッシュ農業気象データは約 40 年の長い収録期間と全国を網羅するデータ形式から、このような利用にも使いやすいためと考えられます。

気象予測を活用する農業技術を開発するには、技術そのものの実用性と同時に気象予測の妥当性についても検証を重ねることが必要であり、そのためには、過去になされた予報を後に再現することが必要となります。そこで、農研機構では、過去に提供した予報値を再現できるキットの提供を 2016 年度から開始しました。最新の気象予測データや過去のデータに加え、過去の予測データも活用して、高度な技術開発が進むことを期待しています。

おわりに

メッシュ農業気象データシステムは、将来の気候変動、気象変動にも負けない農業を支えるための、栽培技術の基盤となるデータです。小規模・分散・多数圃場営農の増加する今日、気象情報を活用した栽培管理技術の開発を進め、汎用的で経済性の高い技術とし、日本全国において利用されるよう今後も研究開発を進めてゆきます。

【講義 2】メッシュ農業気象データの特性について

農研機構 農業環境変動研究センター　桑形恒男

はじめに

メッシュ農業気象データに限りませんが、各種の気象データを農業などに利用するためには、気象データの精度や特性について、きちんと把握しておく必要があります。実際の農業現場において気象データを活用するためには、気象観測データそのものの精度に関する知見に加え、農耕地における気象環境を正しく理解することが重要です。

本講義ではメッシュ農業気象データにおける気象観測データの位置づけと、農耕地の気象環境について解説した上で、メッシュ農業気象データなどの気象情報を農耕地の環境データとして利用する場合の注意点と、農耕地における気象観測の重要性について簡単に述べます。

メッシュ農業気象データの概要

メッシュ農業気象データ（過去値）は、気象庁が作成した 1km メッシュ気候値に、気象庁の気象観測点の日々の観測データを組み合わせることによって作成されます[1]。国内には約 20km の間隔で気象観測点（地上気象観測所とアメダスで約 900 地点、ただし雨量のみの観測点は除きます）が設置され、ルーチン的に気象データが取得されています。メッシュ農業気象データ（気温データ）の作成において、はじめにこれら気象観測データを空間補完することで、1km メッシュ気候値からの日々の 1km バイアスデータを算定します。次にこれらのバイアスデータを 1km メッシュ気候値に重ね合わせて補正することによって、日々のメッシュ気象データ（過去値）が得られます。あらためて言うまでもないことですが、気象観測点のデータは、メッシュ農業気象データを作成する上で重要な役割を果たしています。

メッシュ農業気象データ（予報値）の作成においては、上記のデータに加え、さらに気象庁の数値予報データ（MSM-GPV：空間解像度 5km の格子点ごとの気象予報データ、GSM-GPV：空間解像度 20km の格子点ごとの気象予報データ、その他）を使用しますが[1]、地点レベルの気象観測データの重要性は変わりません。

農耕地の気象環境（水田における調査結果）

日本国内における気象観測点（地上気象観測所とアメダス）は主に市街地に位置していて、農耕地にある地点は少なくなっています。そのためメッシュ農業気象データの精度を考える上で、気象観測点と農耕地における気象環境の違いが問題となります。ここでは国内屈指の高温地帯（夏季）である関東平野の熊谷市を対象とした、気象台（市街地）と隣接した水田（二毛作で冬季は麦を栽培、気象台との水平距離 3.5km、図 1）との間の気象環境（主として気温環境）の違いとその特徴について、3 年間（2010-2012 年）の現地調査の結果[2]に基づいて紹介します。

【講義 2】メッシュ農業気象データの特性について

調査の結果、市街地に隣接した水田では、１年を通して気象台（市街地）より低温で、夜間より日中に気象台との気温差が大きいことや、イネ栽培期間（7～9 月）に日中の気温差が拡大する（7～9 月の月平均の日最高気温が水田の方が 1.2～1.6 度も低い）ことなどが分かりました（図 2)。日中の気象台との気温差は、日射量と共に増大します。これらの結果として、水田における猛暑日(日最高気温 35 度以上)の日数は、気象台のわずか 36%にとどまりました。 また熱帯夜（日最低気温 25 度未満）の日数は、気象台の 62%でした。

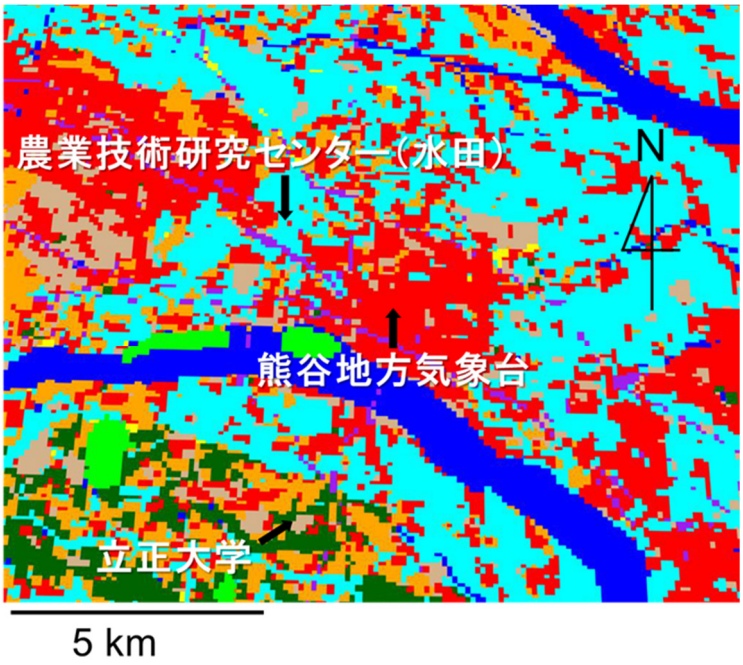


図 1.　熊谷市周辺の土地利用（赤：市街地、水色：水田）文献[２]の Fig.1 からの引用。

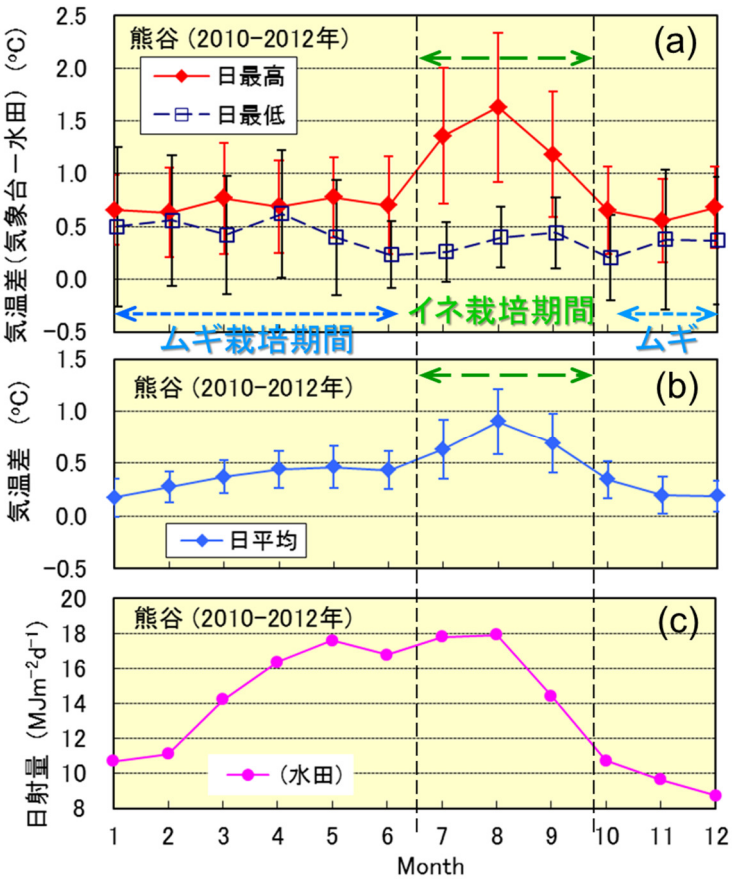


図２.　(a)-(b) 熊谷地方気象台（市街地）と隣接する水田の月平均気温差の季節変化（日最高/日最低/日平均気温）(c) 月平均日射量の季節変化（水田）、3 年平均（2010－2012 年)、文献[２]の Fig.2 からの引用。

気象台と隣接した水田の気温差は、イネ栽培期間（7～9 月）の日中に増加する。４～６月の期間については、日射量は真夏並みに多いものの、気温差は大きくない。

これらに加えて、同じ郊外にある水田と林地（立正大学）では、気象台（市街地）との間の気温差の特徴が全く異なることも明らかとなりました。このように農耕地における気温環境は、そこで栽培されている作物の種類や、ローカルな土地利用形態によって大きく影響を受けます。

メッシュ農業気象データ利用上の注意点

メッシュ農業気象データには、上記で示した水田－気象台間のローカルな気象環境の違いが含まれていないため、その違いがほぼそのまま気温推定の誤差となる点に注意が必要です。実際に、調査対象とした熊谷の水田地点におけるメッシュ農業気象データの気温は、実際に水田で測定された気温よりも、気象台で観測された気温の方に近くなっています。逆に、水田－気象台間のローカルな気温差の特徴を事前に把握しておけば、メッシュ農業気象データに補正を加えることで、調査対象とした水田におけるより高精度な気温推定が可能となります。なお調査対象とした水田における日々の風速、降水量、日射量の各データに関しては、メッシュ農業気象データと実測データとが高精度で一致していました。

おわりに

メッシュ農業気象データはとても便利なツールですが、少なくとも過去値のデータにおいては、現地での高精度な気象観測にはかないません。正確な気象観測にはコストやスキルが必要ですが、事前に現地で気象調査をすることで、メッシュ農業気象データのより高度な利用の可能性が広がることになります。

謝辞

本報告におけるデータ解析を実施するにあたり、農研機構 農業環境変動研究センターの丸山篤志上級研究員にご支援いただきました。

引用文献

[1] 大野宏之・佐々木華織・大原 源二・中園 江（2016）実況値と数値予報，平年値を組み合わせたメッシュ気温・降水量データの作成，生物と気象, 16, 71-79.
[2] Kuwagata, T., Ishigooka, Y., Fukuoka, M., Yoshimoto, M., Hasegawa, T., Usui, Y. and Sekiguchi T (2014) Temperature difference between meteorological station and nearby farmland -Case study for Kumagaya city in Japan-, SOLA, 10, 45-49.

【講義 3】2 週間予報値の紹介と活用

気象庁 地球環境・海洋部　萱場亙起

はじめに

気候変動の影響が顕在化しつつあり、気象・気候災害が頻発する「異常気象時代」が到来している現在、情報インフラ・AI 技術の進歩を背景に、気象データも含めたビックデータを活用した農作物の生産性向上を目的とする農業のスマート化は大きな課題となっている。そのような中、気象庁では、気象情報の作成者と利用者が協力し成功事例を創出することと、そこから得たフィードバックを踏まえて同情報の利便性を向上、改善させるといった取組を進めている。その成果の 1 つとして、2019 年 6 月からは、従来の異常天候早期警戒情報（2 週先の顕著な天候の予測情報）を拡充して、週間天気予報より先の予報として「2 週間気温予報」の提供を開始した。2 週間気温予報は、全国主要地点における週間天気予報の先の 8 日先から 1 2 日先の最高気温・最低気温（5 日平均）の情報であり、毎日午後 2 時 45 分頃に更新する。気象庁ホームページでは、最近 1 週間の実況経過から 2 週間先にかけて、過去の実況から時系列的に一括表示する。2 週間気温予報においてかなりの高温や低温が予想される場合には、「早期天候情報」を原則月曜日と木曜日にあわせて発表し、熱中症予防等の健康管理の徹底や農作業計画における高温や低温による被害リスクを軽減するための早めの対策への呼びかけとして利用される。ここでは、本情報の基となる確率予測資料（以下、2 週間予報値と示す）とその活用について紹介する。

2 週間予報値の紹介

2 週間予報値とは、2 週間気温予報の基礎資料（数値予報から計算される結果で、予報官の判断が含まれない)であり、全国約 150 地点を対象として CSV 形式で午前 9 時 30 分頃までに提供する。従来は異常天候早期警戒情報の基礎資料として平均気温を週 2 回（毎週月、木曜日）提供していたが、今回の拡充によって、最高・最低気温も追加し毎日提供することとした。また、各 7 日間平均から各 5 日間平均とし、気温の変動をより詳細に把握できる。2 週間予測値をボタン一つで取得できるエクセルマクロシートも公開している。

さらに、現在の予測技術を用いて 1981 年まで遡って再予報した結果(以下、再予報データと示す)もあわせて提供する。例えば、極端な天候となった過去の事例を対象に、予測値を用いることの効果を定量的に評価して利用価値を確認できる。2 週間予報値は、気象庁ホームページ内の「気候リスク管理」ポータルサイト（https://www.data.jma.go.jp/gmd/risk/index.html）から取得できる。

2 週間予報値の活用

表に「異常天候早期警戒情報」と「1 か月予報」の基礎資料の活用事例を示した。向こう 2 週間・1 か月の気温予測データは、国立研究開発法人農業・食品産業技術総合研究機構（以下、農研機構）が開発するメッシュ農業気象データシステムで反映されている他、都道府県の農業試験場等における水稲の刈取り適期や果樹の開花期、病害虫の防除適期など定量的な予測事業にも活用されている。再予報データも利用して、予測値の活用メリットを評価した調査事例を紹介する。横山 （2014）では、1985～2012 年を対象に、平年値を用いた従来の水稲刈取適期予測の方法を 4 週間予測値に置き換えてシミュレーションした。その結果、平年値を用いた場合では、観測値による結果との最大の誤差が 4 日以上遅くなった事例がある一方、予測値を用いると 1 事例を除き 3 日程度に改善でき実用上有効であると考察した。農研機構の近畿中国四国農業研究センター（現在は西日本農業研究センター）では、小麦の開花日予測を対象に、1991～2010 年のデータを用いて開花 3 週間前の時点で平年値を用いた方法を 2 週間予測値に置き換えて検証したところ、改善は 20 年中 13 年、改悪は 3 年で、平均して 1 日程度改善でき有効性を確認した。さらに、高温となった 2013 年の事例では、約 2 週間前の時点で 3 日程度改善し的中したことを示し、無人ヘリコプターを用いた赤かび病の防除日の決定をより事前に決定できると考察した（気象庁・農研機構、2016)。萩原（2019）では、2018 年 3 月の極端な高温に対し、従来の平年値を活用した予測よりも、気温予測値を用いることで 3 月の早い段階から開花が早まることを県内に周知することができたと評価した。

表　各機関における向こう 2 週間・1 か月の気温予測データの活用事例
※リンク先は、関連ページを示す。

作物	項目	技術情報例
水稲	冷害･高温障害対策	東北農業研究センター　栽培管理のためのメッシュ情報 https://www.data.jma.go.jp/gmd/risk/taio_suitou.html
	収穫適期予測	山形県　おきたま米づくり情報 https://www.data.jma.go.jp/gmd/risk/taio_kensho.html
		香川県　「おいでまい」通信
		新潟県　稲作技術情報
小麦	開花日予測	西日本農業研究センター　リアルタイムアメダスを用いた麦の発育ステージ予測 https://www.data.jma.go.jp/gmd/risk/taio_komugi.html
果樹	モモの開花日予測	山梨県　モモの開花予想
病害虫	発生予察	沖縄県　技術情報カンシャコバネナガカメムシの防除適期について
その他	メッシュ情報	農研機構 メッシュ農業気象データシステム https://amu.rd.naro.go.jp/

おわりに

地球温暖化等に伴う異常気象によるリスクが増大し、また、予測技術の向上により長期予報の利用可能性も増大している背景で、長期予報の活用可能性の余地が多くある。ただし、予測対象期間が長期になるに従い、不確実性を考慮した活用が必要となる。これには、再予報データも利用し、過去事例を対象とした予測精度の評価をして、活用メリットを把握するのが効果的である。気象データの取り扱いに日ごろから慣れ親しんでいただくことで、極端な天候が予想された際に高温や低温・凍霜害といった農業に関連する障害の発生可能性を早期に検知し、事前に迅速かつより効果的な対策を行うことが可能となる。農業現場における気象予測データの利活用のきっかけにしていただければ幸いである。

引用文献

[1] 気象庁ら：気候予測情報を活用した農業技術情報の高度化に関する研究、共同研究報告書、2016.
[2] 萩原栄揮：気象データを活用した山梨県におけるももの生育予測、グリーンレポート 596. 2-5、2019.
[3] 横山克至：気象確率予測資料を用いた水稲刈取適期の予測、東北の農業気象 58. 1-6、2014.

【講義 4】メッシュ温暖化シナリオデータについて

農研機構 農業環境変動研究センター　西森基貴

はじめに

農研機構農業環境変動研究センター（以下、当センター）では、IPCC 第 5 次報告書のベースである第 5 期（気候予測のための大気海洋）結合モデル相互比較計画（CMIP5）に登録されたもののうち、6 つの全球気候モデル（GCM）出力を日本域で、気象観測統計値と気候モデル出力の年々変動の分散の相違をバイアス補正し、日射量や湿度等を含む 3 次メッシュ（約 1km）のデータセットを作成した（表）。そして 2018 年 5 月に、日本の 2 つの GCM による汎用的要素（気温・降水量）について「メッシュ農業気象データシステム」で早期公開した。

主に文部科学省気候変動適応技術社会実装プログラム（SI-CAT）の支援のもと当センターで新規開発した気候シナリオ（農研機構シナリオ 2017）は、当初は SI-CAT 等、全国スケールでの影響評価・適応策立案プロジェクトの使用を目的としていたが、施行された「気候変動適応法案」に従い、今後は、地域的な適応策策定のための研究プロジェクト（環境研究総合 S15）、環境省・農水省ほかの「地域適応コンソーシアム事業」、および国立環境研究所気候変動適応センターを通じて適応策を策定する地方自治体にも提供されることになっている。地域的な適応策としてはコメ等の農業分野で影響が最も重大で確信度が高く、対応の緊急性が求められる（農林水産省、2015）。そのため、新たな気候シナリオを、これまで農業分野における気候の影響評価、短期的あるいは季節的な予察・予測に多大な利用実績のある「メッシュ農業気象データシステム」に搭載し、地域におけるより長期的な影響評価や適応策立案のために利用いただけるよう提供を続けている。

表　作成した気候シナリオの緒元

ファイルフォーマット	NetCDF4(CF1.6準拠)
使用した全球モデル	MIROC5, MRI-CGCM3, GFDL-CM3, HadGEM2-ES, CSIRO-Mk3-6-0, IPSL-CM5A-LR, GFDL-ESM2M, MIROC-ESM-CHEM bcc-csm-1, MIROC-ESM
温室効果ガス排出シナリオ	historical, RCP2.6, RCP8.5
バイアス補正手法	正規分布型スケーリング法 (Haerter et al., 2011)
計算領域と空間分解能	日本全国3次メッシュ（新座標系[JGD2000]1km）
計算期間と時間分解能	月値、日値 現在（1981-2005）、近未来（2006-2055)、将来（2056-2100）
出力要素	日降水量、日平均気温、日最高気温、日最低気温、 日積算日射量、日平均相対湿度、日平均地上風速

【講義 4】メッシュ温暖化シナリオデータについて

本気候シナリオデータの特性

従来、当センターにおいては、気候変動の農業影響評価のために、主に環境省プロジェクトの中で Ishigooka et al. (2017)による気候シナリオ（農環研シナリオ 2015）を開発した。このデータセットは、日本で初めて、CMIP5 を用いた農業影響評価のための、気温、降水量のほか、日射、湿度及び地上風速を含む高解像度（1km メッシュ）気候シナリオであり、すでに農林水産省「気候変動対策プロジェクト」（A-8）でコメ、コムギ、ダイズの穀物類のほか、野菜や果樹における影響評価に用いられている。この農環研シナリオ 2015 は、月平均値を補正したうえで、日々の変動は確率的に乱数を発生させて日々の値を発生させるウェザージェネレータという手法を用いているものである。３次メッシュごとに乱数を発生させるため、毎日の気象要素の値にはメッシュ間での関係は無いことに注意が必要である。また補正のためのベースラインとして、「農環研アメダスメッシュ化データ」（清野、1993）を用いており、グリッドの座標系が日本測地系（Tokyo Datum）、いわゆる旧座標系に準拠しており、2001 年以降の国土数値情報の座標系と異なるという問題点がある。

これに対し農環研シナリオ 2017 は、この「メッシュ農業気象データシステム」の値を補正のためのベースラインに定め、気候モデルの日々の出力やその変動に準拠したうえで、世界測地系（新座標系）に対応した「メッシュ農業気象データシステム」における観測統計値（予報値ではない）を基準データとして定めた。またバイアス補正法として、気候モデル出力と観測統計値との、長期（20 年）平均値だけでなく、その期間の分散をも補正する正規分布型スケーリング法（Haerter et al., 2011）を採用した。

データの利活用

まず、この気候シナリオの特性と活用例を、Ishigooka et al.(2011)による、コメ品質低下リスクの指標となるヒートドース値を指標にして検証した。比較検証に当たっては、従来の農環研シナリオ 2015 のほか、SI-CAT で同時に開発された気候シナリオ（SI-CAT 防災研シナリオ）、およびバイアス補正の効果を併せて検証するために CMIP5 気候モデルの補正前の出力を併せて示す。また「メッシュ農業気象データシステム」に未搭載のデータも併せて示している。ここでヒートドース値とは、イネの出穂後 20 日間（登熟期前半）の日平均気温が 26℃を超過した分を積算した暑さの指数で、この値が 20（℃･日）を越えると品質低下リスクが高まる、とされる農業気象学的指標である。茨城県南部の例では、いずれのダウンスケーリングシナリオにおいても、基準期間ではヒートドース値が 20 を超えることは稀であるのに対し、近未来期間においてはほとんどのケースで 20 を超え、コメ品質が危険水準に達するとされる 40 を超えるケースも見られた（図 1）。ここで農環研シナリオ 2017 は、他の 2 つのシナリオに比べ、ヒートドース値が過大となっている。これは本気候シナリオが、夏季の気温をやや過大評価する傾向にあり、26℃というヒートドース値の絶対値基準に対して、わずかな差が累積されたためと考えられる。

【講義 4】メッシュ温暖化シナリオデータについて

また降水量の利用を念頭に、気象研究所で極端降水の指標として採用されている 99 パーセンタイル降水量（この場合は、ある地点において日降水量を多い方から順に並べた時の上位 1 パーセントに相当する値）を採用し、図 1 のヒートドース値と同様の 6 気候モデル各 20 年間の出現確率を評価した相互比較を行った。その結果、SI-CAT 農環研シナリオは、他の 2 つの気候シナリオに比べ、基準期間における極端現象の再現性が、基準データ（NARO メッシュ）に近づくように向上しており（図 2）、本気候シナリオが、少なくとも日単位の降水量では、現在課題となっている気候変動下での極端現象の推定に有効であることが示唆された。

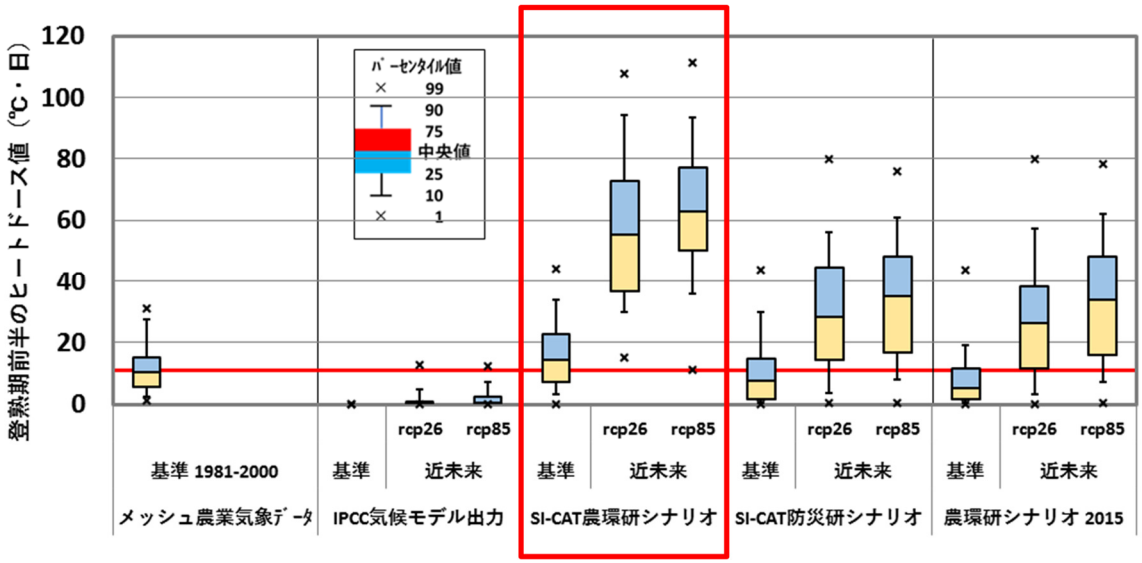


図 1　茨城県南部地帯のヒートドース値の基準期間（1981-2000 年）および近未来期間（2031-2050 年）で RCP 排出シナリオごとの 20 年間での出現確率を示す箱ひげ図。RCP ごとに 6 つの気候モデル全ての各 20 年間、計 120 のサンプルを用いている（文部科学省 SI-CAT 報告書による）。

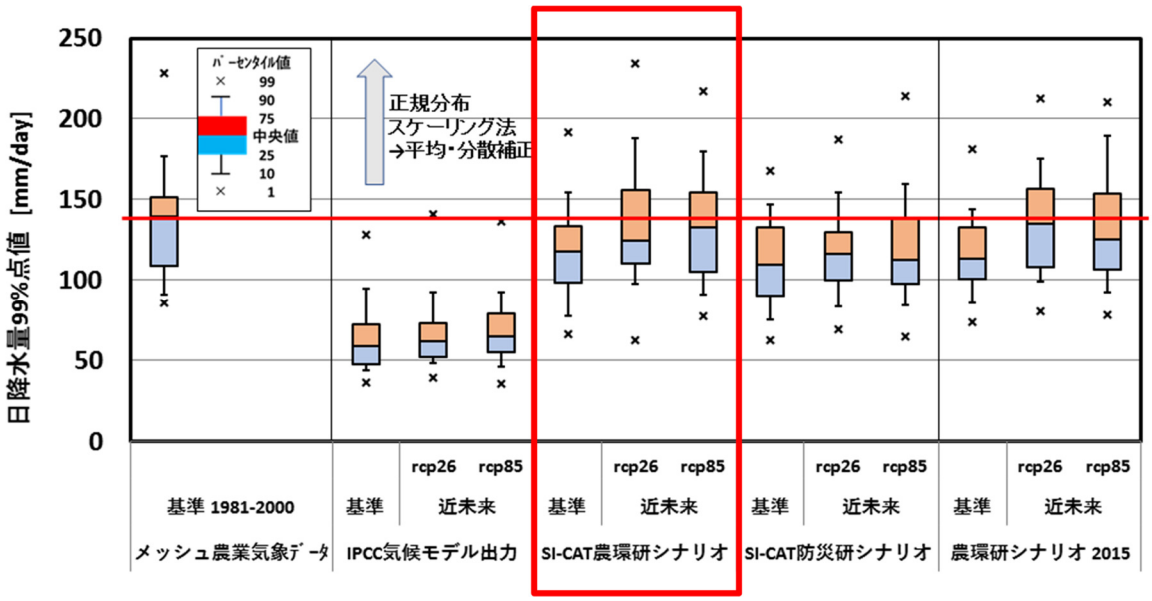


図 2　図 1 に同じ、ただし高知市付近における日降水量年間 99 パーセンタイル値を示す（出典同）。

【講義 4】メッシュ温暖化シナリオデータについて

データ利用の留意点

本気候シナリオは、これまで農業分野で用いられてきた農環研シナリオ 2015 に代わり、気候予測値や変換・複合指標の作成とその空間的図示が可能なシナリオである。簡易なバイアス補正が苦手とする極値的な事象の表現も向上していることが推測されるが、利用に当たっては、いくつかの点に留意する必要がある。まず、気候モデル出力をバイアス補正した気候シナリオの共通点であるが、GCM 出力値の単なる値補正・空間内挿であり、物理的な付加情報は無いことである。また 1km メッシュという現状で最も解像度の高い気候シナリオであるが、開発目的から本来は全国・広域評価用であり、特定地域やポイント抽出には注意が必要である。農環研では従来、農地（平地～中山間地まで）を研究対象にしていること、そもそもベースラインとなる「メッシュ農業気象データシステム」の観測統計値の基となったアメダス観測点が山岳部にほとんどなく、また補正前の気候モデル出力は、空間解像度が 100～200km と粗く、そもそも日本の地形を反映していない。このため、補正後であっても山岳部のデータの信頼性が判断できない。さらに「メッシュ農業気象データシステム」の現在期間の年々変動における分散に併せて補正する本手法は、トレンドを持つ気候モデルの現在値と将来予測値との差をさらに拡大して評価したり、ある日の降水量を極端に過大評価する場合が散見されることが指摘されている。

今後、気温と降水量以外の、日射、湿度および風速についても、本システムへの搭載を行うが、湿度と風速については、既に示されているように観測統計値ではなく予報モデルの出力であり参照年数が短いこと、また正規分布を仮定した補正であり、降水量の過大評価のほか、湿度や風速の過小評価や風速ゼロの値使用には特に注意が必要である。

おわりに

本気候シナリオは、出力値だけでなく、検証を兼ね、さまざまな指標にも変換して利用可能である。例えば、果樹の栽培適地マップの試行作成を行っている。今後は、日射、湿度及び風速データの搭載に併せ、利用プログラムやガイダンスの整備を行っていく。誌面の都合上、気候シナリオそのものと、それにかかわる気候予測モデル、特に地域気候予測における現状や課題など、より一般的な情報については割愛した。影響評価のための気候シナリオの利用例や注意点等も併せ、三村ほか（2015）を参照いただきたい。

謝辞：

本データセットの作成に当たっては、農研機構第 4 期中期計画（気候変動影響評価プロジェクト）、文部科学省気候変動適応技術社会実装プログラム（SI-CAT）および農林水産省「気候変動対策プロジェクト」（A-8）の支援を受けた。

【講義 4】メッシュ温暖化シナリオデータについて

参考文献：

Haerter, J., S. Hagemann, C. Moseley, and C. Piani (2011), Climate model bias correction and the role of timescales, Hydrol. Earth Syst. Sci., 15(3), 1065–1079.

Ishigooka, Y., T. Kuwagata, M. Nishimori, T. Hasegawa and H. Ohno (2011): Spatial characterization of recent hot summers in Japan with agro-climatic indices related to rice production. J. Agric. Meteorol., 67, 209-224.

Ishigooka, Y., S. Fukui, T. Hasegawa, T. Kuwagata, M. Nishimori and M. Kondo (2017): Large-scale evaluation of the effects of adaptation to climate change by shifting transplanting date on rice production and quality in Japan. J. Agric. Meteorol., 73, 156-173.

三村信男（監修), 太田 俊二・武若聡・亀井雅敏（編集)(2015)：気候変動適応策のデザイン ~Designing Climate Change Adaptation~. クロスメディア・マーケティング社発行（インプレス社発売）、120p.

農林水産省（2015)：農林水産省気候変動適応計画. http://www.maff.go.jp/j/kanbo/kankyo/seisaku/pdf/ tekiou.html（2018 年 6 月 13 日閲覧）

清野　豁（1993）：アメダスデータのメッシュ化について. 農業気象, 48(4), 379-383.

【実習 1】メッシュ農業気象データの取得 1 Excel の利用

農研機構 北海道農業研究センター　根本 学

はじめに

最初の実習では、メッシュ農業気象データを取得するために専用に作成した、Microsoft 社製のソフトウェア「エクセル」用のファイルを用い、メッシュ農業気象データを取得する方法を実習します。ここでは、二つのエクセルファイルを用いて実習を行います。一つ目は、メッシュ農業気象データの気象要素を一つ選び、指定した緯度・経度を含むメッシュのデータを 1 年単位で取得可能なファイルです。このファイルを「メッシュ取得ファイル」と呼ぶことにします。二つ目に、エクセル上の任意の場所にメッシュ農業気象データを取得するよう設定可能なファイルを扱います。このファイルを「メッシュ組み込みモジュール」と呼ぶことにします。

この二つのエクセルファイルの実習の前に、メッシュ農業気象データがどのように配信されているのか？について、簡単に説明しておきます。

（1）メッシュ農業気象データの配信サーバ（OPeNDAP）について

メッシュ農業気象データの配信は、OPeNDAP ("Open-source Project for a Network Data Access Protocol")という、HTTP ベースのデータ転送プロトコルで行われます。言い換えれば、web ブラウザで HP を閲覧するような方法で、データを取得することができるもの、になります。試しにメッシュ農業気象データの配信サーバーの TOP ページ（https://amd.rd.naro.go.jp/opendap）開いてみると、図 1 のような画面が表示されます（最初に ID とパスワードを求められます）。AMD をクリックするとメッシュ農業気象データを、AMS をクリックするとメッシュ温暖化気象データシナリオデータ（講義 4 を参照）を選択する画面に移動します。

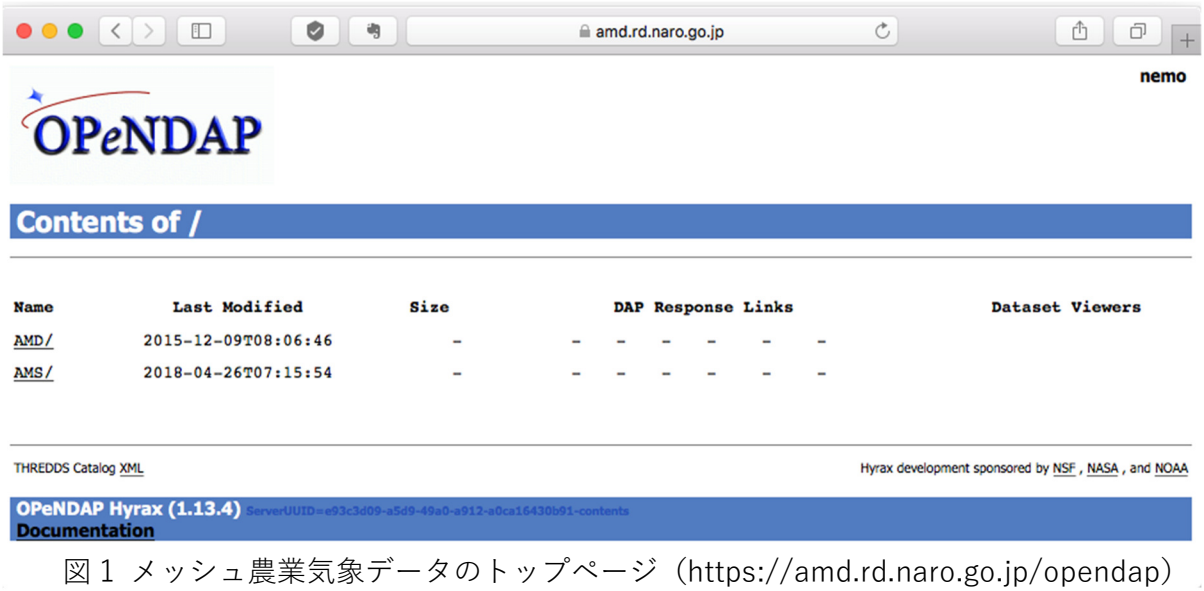


図 1 メッシュ農業気象データのトップページ（https://amd.rd.naro.go.jp/opendap）

【実習 1】メッシュ農業気象データの取得 1 Excel の利用

例として、北海道農業研究センター（北農研）の気象観測露場（北緯 43.0095°、東経 141.4080°）を含むメッシュについて、2017 年の 4 月 1 日から 10 月 31 日までの日平均気温のデータを得たいとします（図 2）。そのためには、メッシュ農業気象データの配信サーバーのトップページ(図 1)から、AMD → Area1 → 2017 → AMD_Area1_TMP_mea.nc と移動します。Variables の TMP_mea のボックスにチェックを入れ、直下の time、lat、lon にそれぞれ、90:303, 441, 192 とパラメータをそれぞれ入れて、Action: の Get ASCII をクリックします（図 3）。 すると、図 4 のようにコンマ区切りのテキストが表示され、右側の数値が欲しいデータとして得られます。しかしながら、余計な文字列も沢山ついていているし、データは日付順に並んでいそうだけど、正直分かりにくいものだと思います。

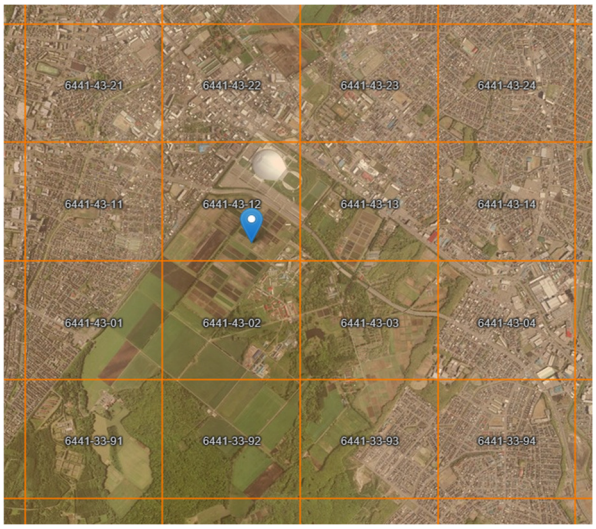


図 2　北農研気象観測露場の位置

マス目は、メッシュ農業気象データの 1 メッシュの範囲で、国土数値情報の 3 次メッシュに対応（約 1km×1km）

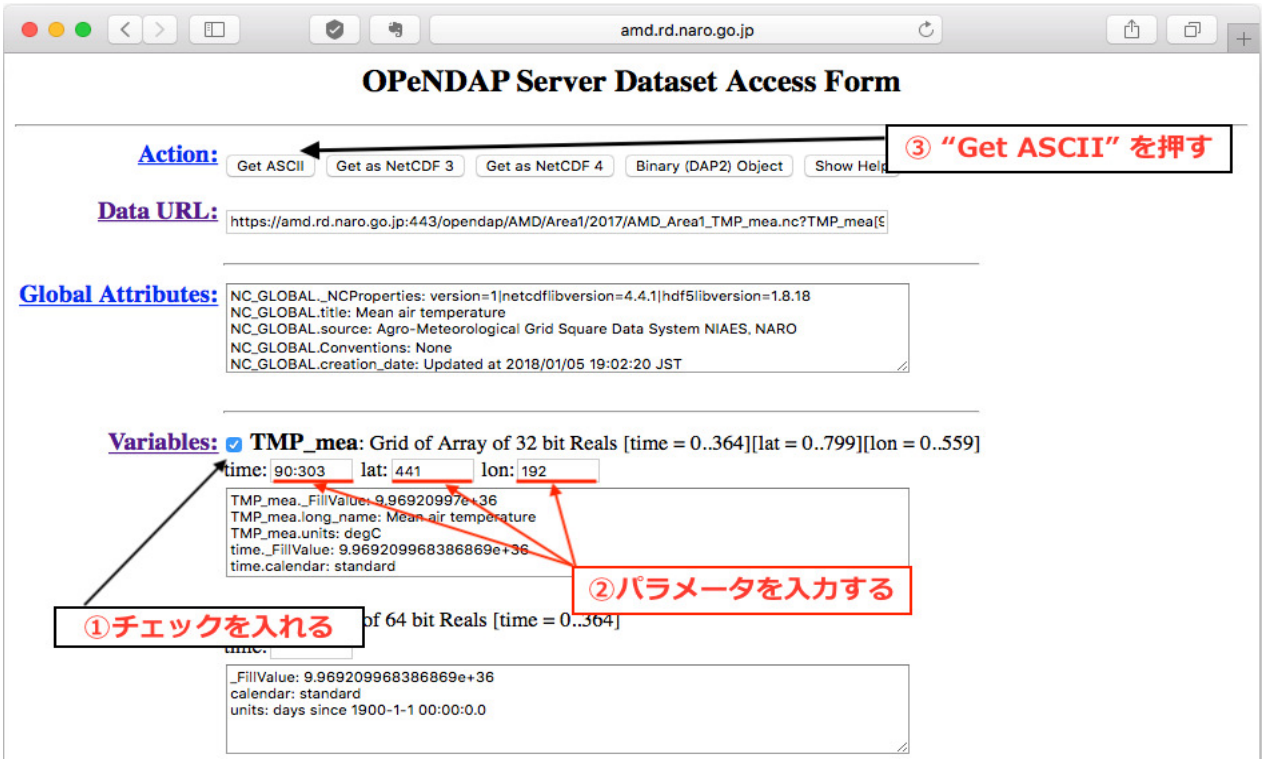


図 3 北農研の 2017 年 4 月 1 日から 10 月 31 までの日平均気温を取得する際の操作

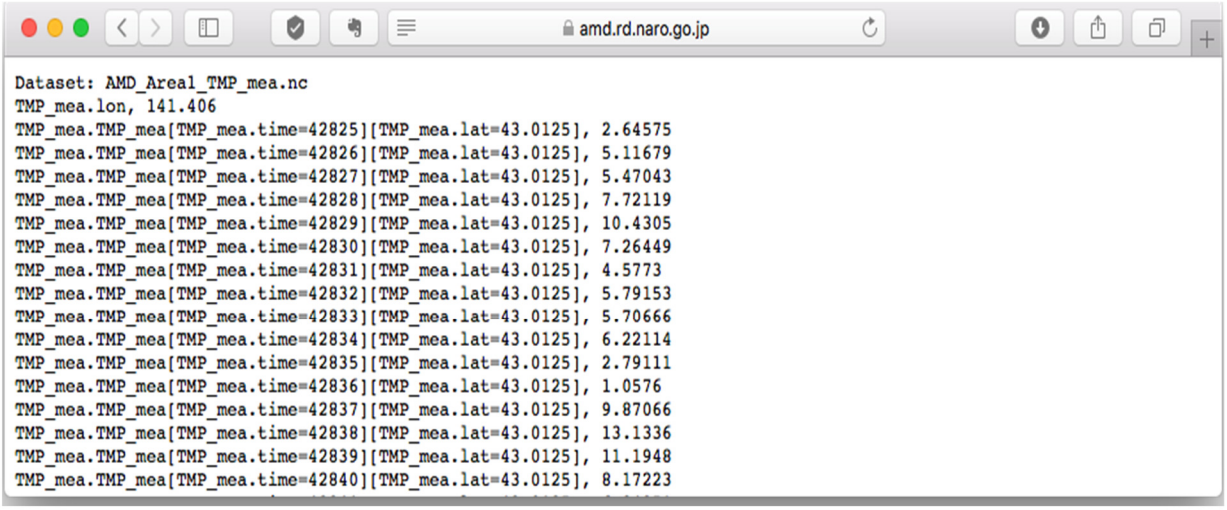


図 4　図 3 の操作で得られる csv データ

※データ中、TMP_mea.time=42825 とあります。42825 は日付を示す整数ですが、エクセルの日付連番とは異なるので注意してください。

このように、メッシュ農業気象データは、web サーバーから欲しいデータを選択し、自在に切り出すことが可能な仕組みを持つ反面、指定する方法（緯度経度や取得期間を独自のインデックスで指定する）が分かりにくく、気象庁の観測データ配信サイト（例えば、http://www.data.jma.go.jp/obd/stats/etrn/index.php: 過去の気象データの検索）のように、web ブラウザ上で指定の期間や気象要素を、ダイレクトに選択して取得することができません。

そこで、データ取得が容易なように開発された「メッシュ取得ファイル」「メッシュ組み込みモジュール」や、Python 用のライブラリ（AMD_Tools3.py）の利用が大変便利です。

（2）エクセルファイルによるデータ取得（メッシュ取得ファイルの利用）

1）利用者 wiki から取得

利用者限定版 wiki （https://amu.rd.naro.go.jp/wiki_user/doku.php）の「メッシュ農業気象データ取得エクセルファイル」から、「AMGSDS_1d_win（or mac）」ファイルをダウンロードして取得しましょう。

※ Mac 版を利用する場合は、wiki の説明に従い、opendap.scpt を所定の場所に配置してください。

2）エクセルファイルを開く

ファイルを開く際に、「マクロを有効」に、「コンテンツを有効化」します。

「ID パスワード」シートに、自分の ID とパスワードを入力する。「接続確認」を押して「メッシュ農業気象データ利用可能です。」というダイアログが表示されれば、準備 OK です。

※ ID とパスワードが正しく入力されていない場合、もしくはサーバーが停止している場合は「認証情報が正しくありません。」「接続に失敗しました。」等の表示が出ます。

【実習 1】メッシュ農業気象データの取得 1 Excel の利用

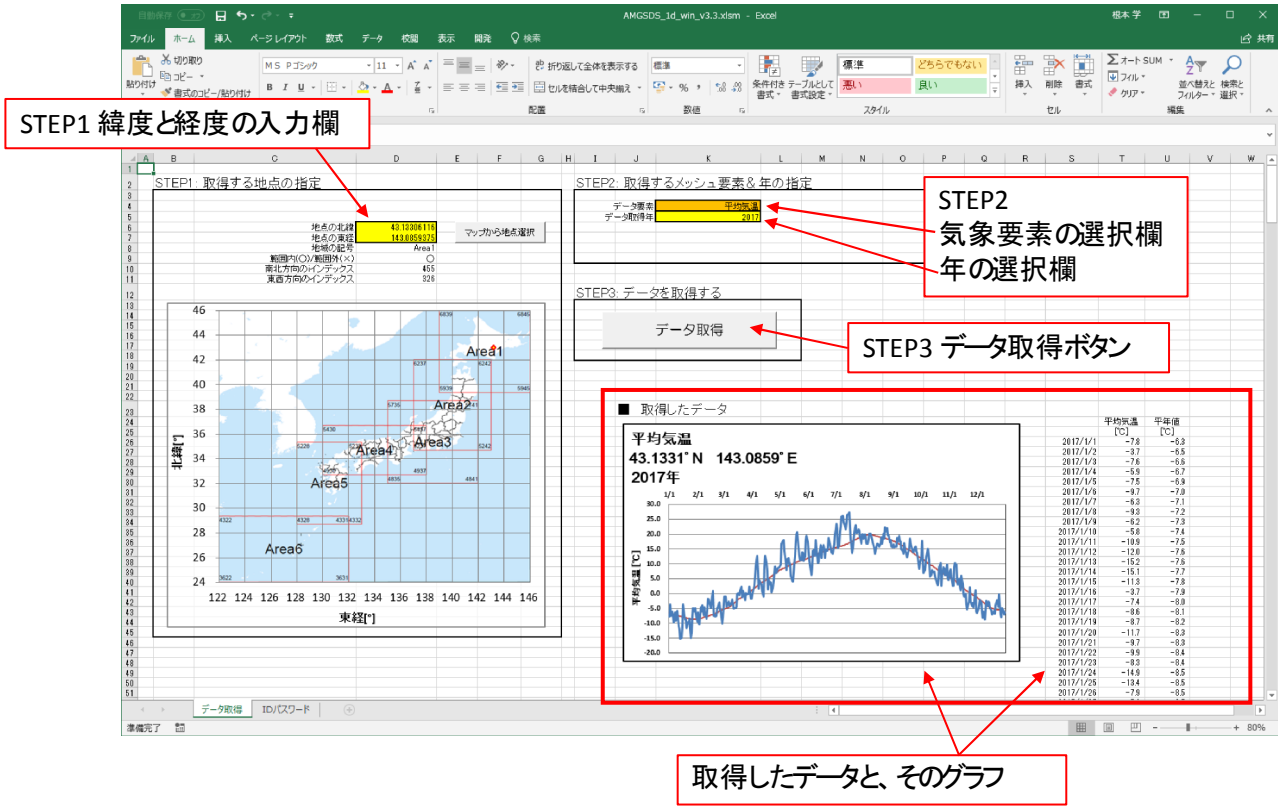


図 5　「データ取得シート」の構成

3) データを取得する

「データ取得シート」は図 5 のようなレイアウトになっています。STEP1 から STEP3 まで、必要な項目を入力／選択していきましょう。例として、北海道農業研究センター（北農研）の気象観測露場（北緯 43.0095°、東経 141.4080°）を含むメッシュについて、2017 年日平均気温のデータを取得します。

STEP1: 緯度、経度の欄に取得したい圃場の緯度経度を入力する。
※ 北農研の圃場の緯度・経度（北緯 43.0095°、東経 141.4080°）
STEP2: データ要素は「平均気温」を選択し、データ取得年は「2017」を入力する。
STEP3: 「データ取得」ボタンを押す。

すると、データ取得シートの右下部分（図 5）に、指定した地点を含むメッシュの 2017 年の平均気温について、図とデータが表示されます。

このエクセルシートは、指定した年の 1 月 1 日から 12 月 31 日までのデータを取得するので、必要な期間（例えば、4 月 1 日から 10 月 31 日まで）のデータをコピーして、他のシートに移すなどして利用することが可能です。

（3）エクセルファイルによる自在なデータ取得（メッシュ組み込みモジュールの利用）

「メッシュ取得ファイル」では、とても簡単にメッシュデータを取得できるものの、一度に 1 要素しかデータを取得することができず、期間も 1 年ずつ、という制約がありました。また、取得したデータを元にして、様々な指標の計算を行いたい場合は、取得したデータを他のエクセルファイルにコピー＆ペーストする必要がありました。このような制約なしに、エクセル上の任意のシート・セルに、任意の複数要素、様々な期間のデータを取得できるように作成されたのが「メッシュ組み込みモジュール」です

この「メッシュ組み込みモジュール」は、本来は、既に気象データを利用するように作られたエクセルファイルに、メッシュ農業気象データ取得機能を追加できるよう意図して作成されたものですが、この「メッシュ組み込みモジュール」のエクセルファイルから、新たにメッシュ農業気象データを利用するファイルを作成していくことも可能です。この実習では、後者の方法について、説明します。

「メッシュ組み込みモジュール」ファイルは、　利用者限定版 wiki（https://amu.rd.naro.go.jp/wiki_user/doku.php）の「メッシュ農業気象データ組み込みモジュール」にあります。このファイル名は「AMGSDS_mesh_module_v06.xlsm」で、こちらは Win/ Mac 両用ですが、Mac の場合は、「メッシュ取得ファイル」と同様に、opendap.scpt の配置（一度行えば OK）が必要です。

実習として、北農研の気象観測露場を含むメッシュについて、2018 年の 6 月 1 日から 6 月 30 日までの、日平均気温、日平均気温の平年値、日射量、降水量を取得する場合を例に説明します。

1）シート構成を確認する

「メッシュ組み込みモジュール」を開きます。（最新バージョンは AMGSDS_mesh_module_v06.xlsm）
ファイルを開く際に、「マクロを有効」に、「コンテンツを有効化」します。

シートは 4 枚あります。

1 枚目：　組み込みモジュールの使い方　　［削除可能］
　モジュールを他のエクセルファイルに組み込む手順が記載されています。
2 枚目：　ID_パスワード　　【削除不可】
　メッシュ農業気象データの利用 ID とパスワードを入力するシートです。
3 枚目：　Mesh 設定　　【削除不可】
　メッシュ農業気象データを取得する各種設定を行うシートです。
　ワークシート「demo」にデータを取得する二つの設定が記載されています。
4 枚目：　demo　　［削除可能］
　Mesh 設定にデモとして設定されている、二つの設定の出力先。

2）エクセルのメニューに、「開発」があることを確認する

「開発」が無い場合は、以下の方法で「開発」を表示させる。

Win の場合：　ファイル　→ オプション　→ リボンのユーザ設定
→ リボン画面の右側最下段にある「開発」にチェックを入れる。

Mac の場合：　Excel → 環境設定　→ リボンとツールバー
→ リボン画面の右側最下段にある「開発」にチェックを入れる。

3）利用者 ID とパスワードを入力する

・ID_パスワードシートを開き、ID とパスワードを入力する。

・接続確認ボタンを押し、データ取得が可能な状態であることを確かめる。「メッシュ農業気象データが利用可能です。」と表示されます。

4）出力先のシートを用意する

1. 新規シートを作成し、シート名を「北農研露場」とする。
2. セル A1 に「緯度」と入力し、セル B1 に　43.0095 を入力する。
3. セル A2 に「経度」と入力し、セル B2 に　141.4080 を入力する。
4. セル A3 に「年」と入力し、セル B3 に 2018 を入力する。
5. セル C5 に 2018/6/1 と入力する。
6. セル C6 に数式「=C5+1」と入力する。（2018/6/2 と表示される。）
7. セル C6 をコピーし、セル C7～C33 を選択してペーストする。
（2018/6/3 ～ 2018/6/30 が表示される）
8. セル D4 に「日平均気温」と入力する。
9. セル E4 に「日平均気温平年値」と入力する。
10. セル F4 に「日射量」と入力する。
11. セル G4 に「降水量」と入力する。

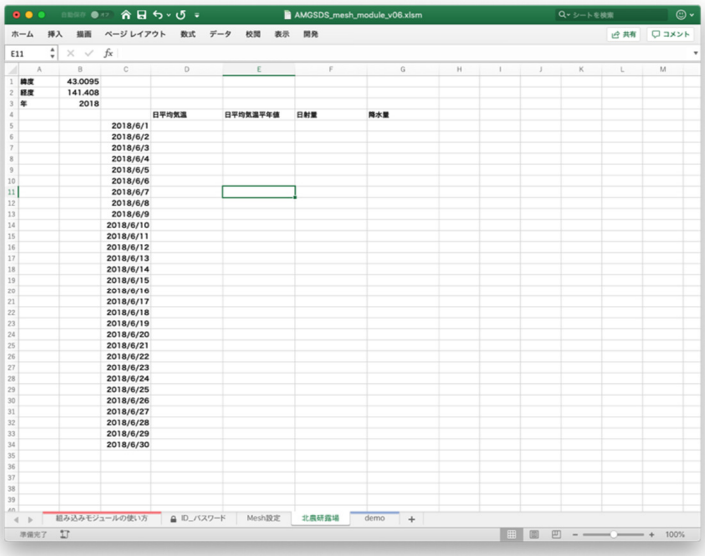


図 6　出力先シートへの入力完了後の画面

５）メッシュ農業気象データの取得設定

１．Mesh 設定シートを開く
12 行目に設定情報の項目名が記載されている（「番号」「2/29 の扱い」まで）
番号 1 の行から順番に、データ取得の設定を入力していく

２．番号 1 （13 行目）の設定項目
ボタン名：　（文字）　データ取得
緯度：　（文字）　43.0095
経度：　（文字）　141.4080
取得開始日：　（文字）　2018/6/1
取得最終日：　（文字）　2018/6/30
要素：　（文字）　平均気温
平年値：　空欄
出力シート：　（文字）　北農研露場
出力セル：　（文字）　D5
縦か横か：　（文字）　縦
2/29 の扱い：　空欄

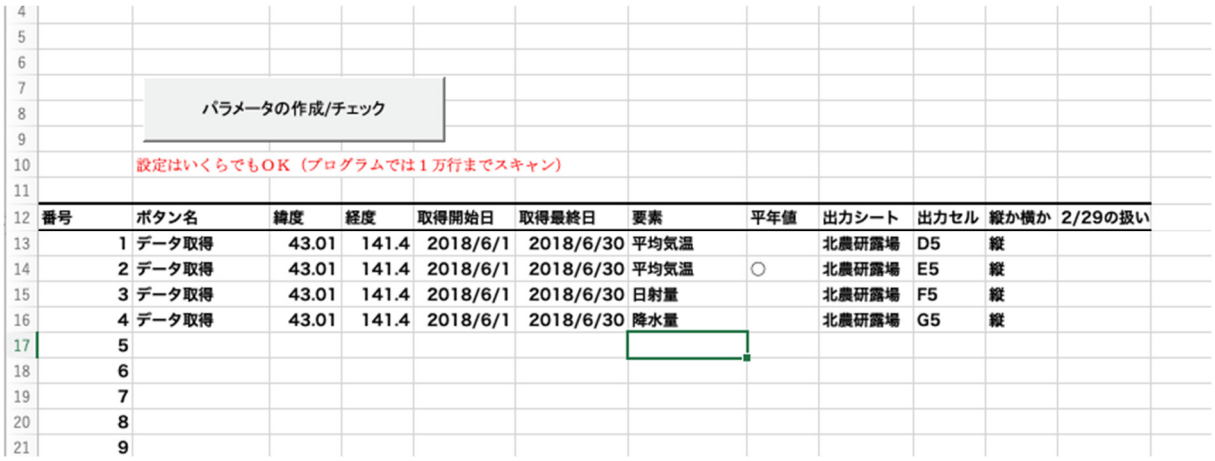
３．番号 2 ～ 4 （14～16 行目）への共通設定項目準備（番号 1 ：13 行目をコピー）
・セル B13～セル L13 を選択して、コピーする
・セル B14～B16 を選択して、ペーストする。

４．番号 2 （14 行目）の設定修正箇所
平年値：　（文字）　○
出力セル：　（文字）　E5

５．番号 3 （15 行目）の設定修正箇所
要素：　（文字）　日射量
出力セル：　（文字）　F5

６．番号 4 （16 行目）の設定修正箇所
平年値：　（文字）　降水量
出力セル：　（文字）　G5

７．パラメータ作成／チェック　ボタンを押す
エラー表示が出なければ、設定完了です。



パラメータの作成/チェック

設定はいくらでもOK（プログラムでは１万行までスキャン）

番号	ボタン名	緯度	経度	取得開始日	取得最終日	要素	平年値	出力シート	出力セル	縦か横か	2/29の扱い
1	データ取得	43.01	141.4	2018/6/1	2018/6/30	平均気温		北農研露場	D5	縦	
2	データ取得	43.01	141.4	2018/6/1	2018/6/30	平均気温	○	北農研露場	E5	縦	
3	データ取得	43.01	141.4	2018/6/1	2018/6/30	日射量		北農研露場	F5	縦	
4	データ取得	43.01	141.4	2018/6/1	2018/6/30	降水量		北農研露場	G5	縦	
5											
6											
7											
8											
9											

図７　Mesh 設定シートへの入力完了後の画面

６）　ボタンの配置／データ取得

１．ボタンの配置

メッシュ農業気象データを取得する機能を与えたボタンを配置します。ここでは、北農研露場シートを開き、適当な場所にボタンを配置します。

Win の場合：　「開発」にある、「挿入」→「フォームコントロール」のボタンを選択　→ シート上の任意の場所でクリックすると、「マクロの登録」画面が表示されます。

Mac の場合：「開発」にある、「ボタン」を押す　→ シート上の任意の場所でクリックすると、「マクロの登録」画面が表示されます。

「マクロの登録画面」で、check_and_get_mesh_data を選択して、OK を押す　（図８）

→ ボタンが配置されます。

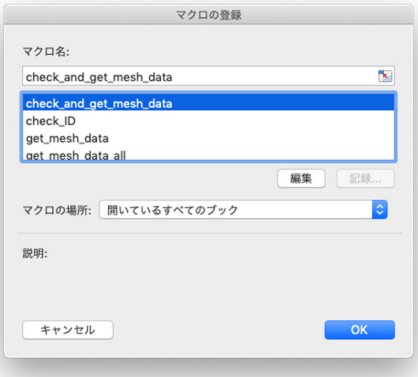


図８　マクロの登録画面（check_and_get_mesh_data を選択）

2．ボタン名の変更

ボタンを右クリックして「テキストを編集」を選択すると、ボタン上の文字列にカーソルが移動するので、「データ取得」と修正する。（Mac の場合は、ボタンを右クリックするなどしてボタンが選択された状態で、ボタン上の文字列を左クリックすると、修正可能となります。）

※ Mesh 設定シートで設定したボタン名のうち、エクセル内に配置したボタンの表示名と同じ行の設定が、ボタンを押した時に動作します。今回の例では、「データ取得」というボタン名について、4 つの設定がされているので、このボタンを押した時に 4 つの設定項目について順次データ取得を行います。ボタンは、ボタン名を変えて複数設置することで、ボタン名ごとに様々なデータ取得機能を割り当てることが可能です。

3．ボタンを押して、データ取得

以上で、全ての設定が終了です。設置した「データ取得」ボタンを押すと、図 9 のように、北農研露場シートに、メッシュ農業気象データを取得することができます。

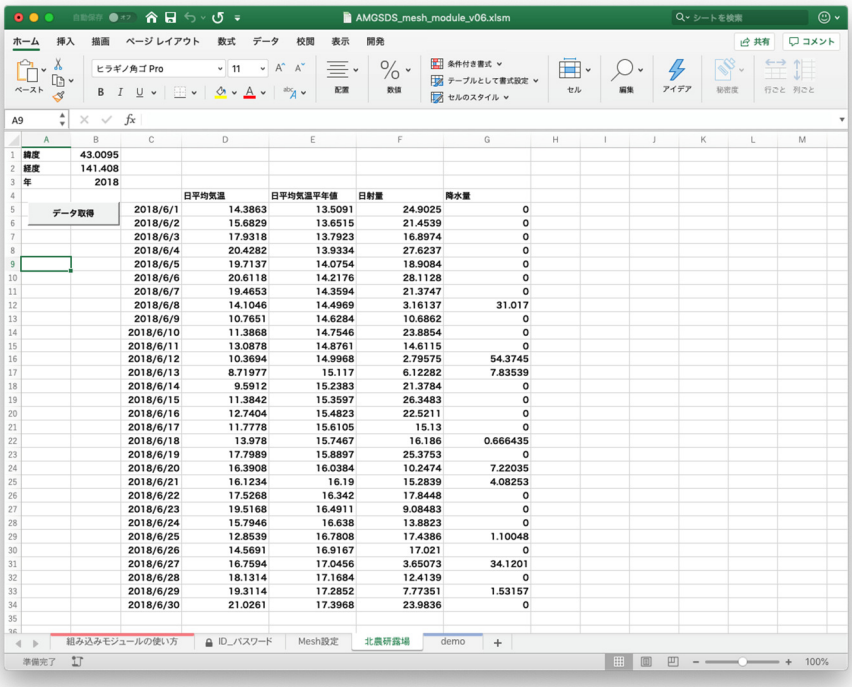


図 9　「データ取得」ボタンを押して、メッシュ農業気象データを取得した後の画面

7）メッシュ農業気象データの取得設定（応用編）

Mesh 設定シートでの設置項目には、エクセルの数式を用いることが可能です。例えば、あるセルにデータを取得する年を入力することにして、このセルを参照したメッシュ農業気象データの取得設定を行えば、このセルの年を変更するだけで、取得するデータの年も変更させることが可能です。

実習の応用編として、5）で設定した「メッシュ設定」シートを、「北農研露場」シートの緯度、経度、年を参照するように修正してみましょう。

1．Mesh 設定シートを開く
5）メッシュ農業気象データの取得設定 の設定が完了した状態であることを確認する。
以後は、これに対する修正箇所のみ示す。

2．番号 1（13 行目）の設定項目
緯度：　（数式）　= 北農研露場!$B$1
経度：　（数式）　= 北農研露場!$B$2
取得開始日：　（数式）　= Date(北農研露場!$B$3,6,1)
取得最終日：　（数式）　= Date(北農研露場!$B$3,6,30)
※$をつけることで、次のコピーで参照がずれないようになります。

3．番号 2～4（14～16 行目）の共通設定項目（番号 1：13 行目をコピー）
・セル B13～セル L13 を選択して、コピーする
・セル B14～B16 を選択して、ペーストする。

4．番号 2（14 行目）の設定修正箇所（3．でコピーしたので再設定）
平年値：　（文字）　○
出力セル：　（文字）　E5

5．番号 3（15 行目）の設定修正箇所（3．でコピーしたので再設定）
要素：　（文字）　日射量
出力セル：　（文字）　F5

6．番号 4（16 行目）の設定修正箇所（3．でコピーしたので再設定）
平年値：　（文字）　降水量
出力セル：　（文字）　G5

7．パラメータ作成／チェック　ボタンを押す
　　エラー表示が出なければ、設定完了です。
8．北農研露場シートを開き、日付セルの修正を行う
　・セル C5：　（数式）　=Date(B3, 6, 1)

以上の設定で、北農研露場シート左上の「緯度」「経度」「年」の右隣のセルの値を変えることで、「データ取得」ボタンを押した時に取得するデータを変更することが可能となります。

【参考資料】

メッシュ農業気象データ Excel 用組み込みモジュール利用マニュアル
https://www.naro.affrc.go.jp/publicity_report/publication/pamphlet/tech-pamph/121104.html
既存のエクセルファイルに「メッシュ組み込みモジュール」を組み込む方法が記載されています。

農地気象環境診断アプリ利用マニュアル
http://www.naro.affrc.go.jp/publicity_report/publication/pamphlet/tech-pamph/121105.html
モバイル端末（iOS、Android）でメッシュに農業気象データを閲覧できるアプリです。10 地点まで地点登録することができます。

【実習 2】Python によるプログラミングの基礎

農研機構 農業環境変動研究センター　片柳薫子

この実習ではサンプルプログラムの仕組みを理解するのに必要な Python の基礎知識を、実習を通して身につけます。実習は、Python の統合開発環境である Spyder を用いて行います。

1) 下準備

(1) メッシュ農業気象データ公開 wiki の「初めて利用される方へ[1]」にある Python 利用環境構築ガイド[2]を参考に Anaconda と追加パッケージをインストールします。

(2) 実習 2 の練習用コードセット practice.py を入手し、作業スペース（ダウンロードしたファイルや作成したファイルを保存するためのディレクトリもしくはフォルダ）に格納します。

(3) AMD_Tools3.py をダウンロードし[3]、作業スペースに格納します。

(4) Anaconda / Spyder を起動して、フォルダマーク（図 1）をクリックしてディレクトリの選択画面を開き、作業スペースを選択して「フォルダーの選択」ボタンをクリックします。

(5) ファイルエクスプローラーのタブ（図 1）を選択し、作業スペース内のファイルを表示します。practice.py と AMD_Tools3.py が格納されていることを確認します。

(6) ファイルエクスプローラに表示された practice.py を開き（ファイルを選択して Enter キーを押す、もしくはファイルを選択してダブルクリック）、図 1 左側のエディタ画面に表示します。

☑Anaconda をインストールすると Python と関連ライブラリ（モジュール）がインストールされます。別途 Python をインストールする必要はありません。

[1] https://amu.rd.naro.go.jp/wiki_open/doku.php?id=python
以下の方法でもアクセスできます:
メッシュ農業気象データ公開 wiki トップページ（https://amu.rd.naro.go.jp/）
→ Sitemap → Python

[2] https://amu.rd.naro.go.jp/wiki_open/lib/exe/fetch.php?media=Python 利用環境構築ガイド.pdf

[3] https://amu.rd.naro.go.jp/wiki_open/lib/exe/fetch.php?media=AMD_Tools3.py

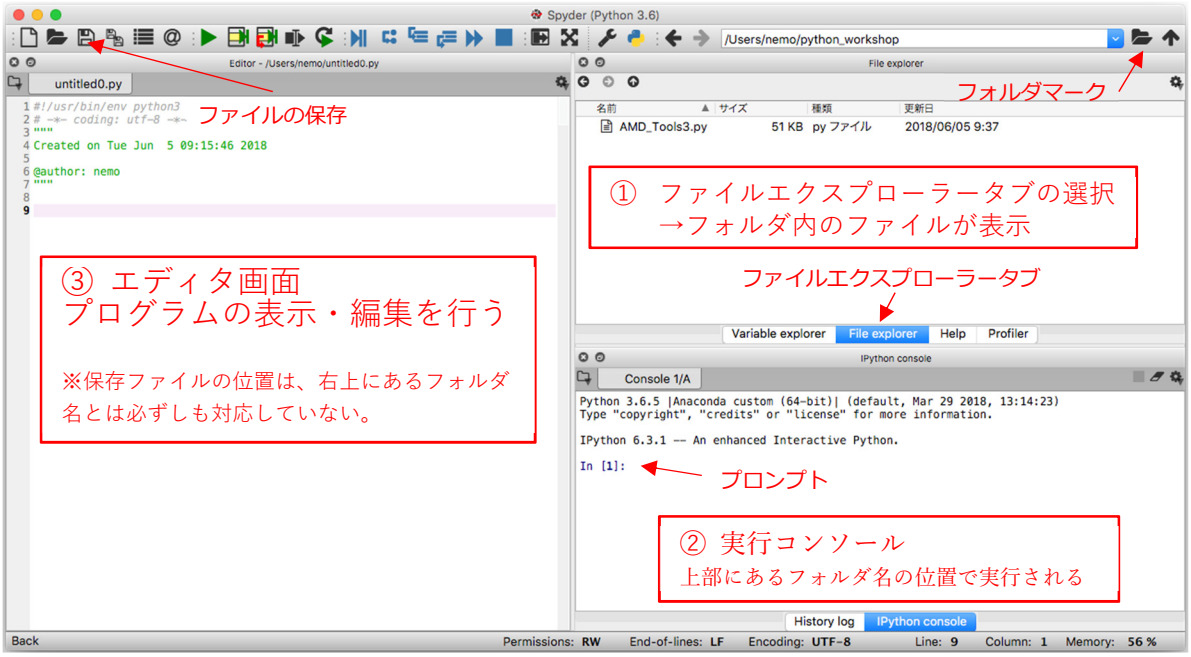


図 1 Spyder を起動して、ファイルエクスプローラーを表示した画面

表 1　Spyder で使われているカラースキーム

色	構成要素	補足
紫	組み込み関数[4] （abs, dict, help, min, max, all など） 組み込み定数[5]（False, True, None など） 組み込み型[6] （int, float, complex など）	Python にはじめから組み込まれている関数・定数・型です。これらの単語や記号を変数名として使用するとトラブルの原因になるので注意が必要です。
青	予約語[7] （and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try）	Python において最初からキーワードとして使われている単語です。通常の変数（識別子）として使うことはできません。
茶	数値	
緑	文字列	
灰	コメント	
黒	上記以外	

☑デフォルトの設定ではコメント行が灰色の斜体になっていますが、本テキストでは斜体を解除して表示しています。
　以下の作業で斜体を外すことができます： ツール/設定/構文強調の配色/選択を編集/「コメント」の右側のイタリックのチェックを外す

☑practice.py のコードに色が付いていることを確認しましょう。

[4] https://docs.python.org/ja/3/library/functions.html
[5] https://docs.python.org/ja/3/library/constants.html
[6] https://docs.python.org/ja/3/library/stdtypes.html
[7] https://docs.python.org/ja/3/reference/lexical_analysis.html#keywords

2) カラースキーム

Spyder のエディタ部分に書き込まれたプログラムは、自動的に文字の種類毎に色がつきます。これは、Spyder がプログラムを解釈し、関数なのか、数値なのか、コメントなのかを解釈しているからです。色分けにより、コードが理解しやすくなり、また、コードの間違いも発見しやすくなります。デフォルトでは表 1 のように色分けがなされています。コードを読んだり書いたりする際、この色分けを意識するとコードが理解しやすくなるかもしれません。

3) 確認の操作

Spyder の実行コンソール（図 1②）の、**In[1]:**と表示されている場所(プロンプトと呼びます)の右に数字や文字、関数などを入力して、Enter(エンターキー)を押すと、Python がその部分をどのように処理/解釈したかが **Out[1]:**に表示されます。そこで、これを利用して Python の基本を学んでいきます。学習の参考に打ち込む数字や文字をあらかじめファイル practice.py にまとめてあります。転記したい部分を選択した状態で Shift + Enter を押すと選択した部分がプロンプトに転記され、実行されます。この方法で簡易にコードの実行結果を得ることができますが、プログラミングが初めてという方は、practice.py のコードを見ながら自分の手で実際にプロンプトに入力しながら学習することをおすすめします。

☑自分で入力して実行する際は、プロンプト部分にコードを書き、最後に Enter キーを押してください。コードが実行されると Out[x]と出力結果（xは整数、場合によっては In[x]: のみ）が出力されます。Out[x]が表示されず「...:」と表示される場合は再度 Enter キーを押してください。

Python の基礎

1) コメント

コメントは、プログラムとして Python が認識しない、コードを読んでいる人に対するメッセージ文です。一般に、プログラムは人と計算機との間の文書と思われていますが、実際はそれ以上に、人と人との文書でもあります。その意味で、コメントはプログラミング言語の構成要素で最も重要なものと言えます。Python ではハッシュ（#）から始まる部分はコメントとみなされ、コードとして解釈されません。行の途中にハッシュをおいた場合はハッシュから後ろがコメントとなります。

```
In [1]: # コメント行です

In [2]:
```

☑コメント行はコードとして認識されないため、Out[1]は表示されません。

☑「123」がコードとして認識されているため、「123」だけがOut[2]に出力されます。

```
In [2]: 123 # 行の途中からでもコメントを書けます
Out[2]: 123
```

2) 文字と数値

(1) 文字列

処理結果を文字で表現したり、与えられた文字に基づいて異なる作業をさせたりするために、Python は文字列を取り扱います。文字列はシングルクオテーション（'）または、ダブルクオテーション（"）で囲んで示します。

```
In [3]: 'メッシュ農業気象データ'
Out[3]: 'メッシュ農業気象データ'
```

```
In [4]: "メッシュ農業気象データ"
Out[4]: 'メッシュ農業気象データ'
```

これらを使い分けると、クオテーションマーク自体を表示することができます。

```
In [5]: '"メッシュ"農業気象データ'
Out[5]: '"メッシュ"農業気象データ'
```

```
In [6]: "That's a good idea."
Out[6]: "That's a good idea."
```

前述の文字列を途中で改行するとエラーとなります。以下のコードを実行するとコード下に示したエラーメッセージが戻ります。

```
In [7]: 'メッシュ
   ...: 農業気象データ'
  File "<ipython-input-9-c8af70be5d49>", line 1
    'メッシュ
        ^
SyntaxError: EOL while scanning string literal
```

☑「`メッシュ」を入力したのち Enter キーを押し、続けて「農業気象データ'」を入力して Enter キーを押します。

※"<ipython-in・の部分の文字は、毎回異なります。

複数行にわたって文字列を書く場合はシングルクオテーション 3 つ（'''）もしくはダブルクオテーション 3 つ（"""）で前後を囲みます。

```
In [8]: '''
   ...: メッシュ
   ...: 農業気象
   ...: データ
   ...: '''
Out[8]: '¥n メッシュ¥n 農業気象¥n データ¥n'
```

☑この記法はコメントを書くときにも使われます。ファイルpractice.pyの2～6行目は、複数行文字列で書かれていることを確認しましょう。

※¥n は改行を示す文字コードです。¥はWindows以外では（場合によりWindowsでも）\（半角バックスラッシュ）で表示されます。

(2) 数値

数の概念に整数と実数があることに対応して、Python には整数型と浮動小数点型が用意されています（ほかにもありますが、ここでは説明しません）。どちらを使っても良い場合もあれば、正しく使い分けなければならない場合もありますが、使い分けについてはその都度説明しますので、まずは 2 種類あるということを覚えておいてください。数字に小数点を用いないと整数型で扱われ、小数点を用いると浮動小数点型で扱われます。

```
In [9]: 1
Out[9]: 1
```

```
In [10]: 1.
Out[10]: 1.0
```

3) 四則演算と代入文

Python では、いわゆる四則演算やべき乗、剰余の演算を表 2 に示す演算子で行うことができます。整数型と浮動小数点型を混ぜて演算してもエラーにはならずそれなりの結果を出しますが、結果の数値型は演算子の種類と演算する二つの数の型に依存します。

加算(+)は文字列にも使えます。

```
In [11]: "メッシュ" + "農業" + "気象"
Out[11]: 'メッシュ農業気象'
```

表 2　Python の二項算術演算子と数値型との関係

演算	演算子	演算結果		
		整数型同士	浮動小数点型同士	混合使用
加算	+	整数型	浮動小数点型	浮動小数点型
減算	-	整数型	浮動小数点型	浮動小数点型
乗算	*	整数型	浮動小数点型	浮動小数点型
除算	/	浮動小数点型	浮動小数点型	浮動小数点型
除算†	//	整数型	浮動小数点型	浮動小数点型
剰余	%	整数型	浮動小数点型	浮動小数点型
べき乗	**	整数型	浮動小数点型	浮動小数点型

†切り捨て

文字と数字の加算はできません。以下を実行するとエラーが戻ります。

```
In [12]: "実習" + 1
Traceback  (most recent call last):

  File "<ipython-input-9-b84a7a772193>", line 1, in <module>
    "実習" + 1

TypeError: must be str, not int
```

文字と数字を組み合わせた文字列を作成したい場合は、数字の前後にクォーテーションを付け、数字を文字列としてから加算をおこないます。

```
In [13]: "実習" + "1"
Out[13]: '実習1'
```

※数値を文字列に変換する関数 str を用いる方法もあります。

【練習問題】

以下のコードを 1 つずつ入力・実行して戻り値を確認しましょう。

```
1 + 2          2 * 2.0          "メッシュ" + "農業" + "気象"
1.0 + 2        6 / 3            "農業" * 3
2 - 1          7 / 3            "農業" + 5
2.0 - 1        7 // 3           "農業" + "5"
2 * 2          7 % 3
```

4）変数と代入文

数学の x や a のように、Python でも特定の役割をする数や文字列等に名前を付けることができます。これを変数と呼びます。変数名にはアルファベットの大文字小文字(区別されます)、数字、アンダースコア(_)が使えます。ただし、最初の 1 文字だけは数字が使えません。

数学で「x =3 とする。」のように、変数に特定の値を与えることをプログラミングでも代入とよび、Python では等号（=）を一つ使います。

```
x = 3
```

いっぽう、すでに与えられた数値がその値かどうかを判定するときには、Python では等号（=）を二つ使います。

```
x == 3
```

「=」と「==」では意味が全く異なるので、注意しましょう。

なお、以下のようにして一つの等号で複数の変数に値を代入することもできます。この例では lat、lon、site_name にそれぞれ 36.0270、140.1104、"Tsukuba" が代入されます。

```
lat, lon, site_name  = 36.0270, 140.1104, "Tsukuba"
```

☑この行を実行したのち、lat, lon, site_name を 1 つずつ入力して実行し、戻り値を確認しましょう。

「$x = x + 3$」という数式はプログラミングが初めての人にとっては違和感がある書き方かもしれませんが、Python では、「今 x に格納されている数に 3 を加えたものを新たに x とする」という文として正しく取り扱われます。

```
In [14]: a = 100
    ...: a = a + 10
    ...: a
Out[14]: 110
```

5）累積代入文

Python では、「a = a + 10」と同じ操作を次の文でも実行できます。このような記法を累積代入文と呼びます。

```
In [15]: a = 100
    ...: a += 10
    ...: a
Out[15]: 110
```

○「-=」や、「*=」なども試してみましょう。

6）比較演算子

Python には、True と False という特別な値(組み込み定数)があります。これらはそれぞれ真(正しい/成り立つ)と偽(誤り/成り立たない)を意味し、条件によって処理を変更するときなどに使用します。比較演算子は、二つの変数を比較して両者の関係に従って、True または False を返します。Python で用いられる比較演算子を表 3 に示します。

表 3　比較演算子

演算子	比較条件
x < y	x は y より小さい
x <= y	x は y より小さい、もしくは x と y は等しい
x > y	x は y より大きい
x >= y	x は y より大きい、もしくは x と y は等しい
x == y	x と y は等しい（等価である）
x != y	x と y は等しくない（等価でない）
x is y	x と y は同じオブジェクトである
x is not y	x と y は同じオブジェクトではない
x in y	x は y に含まれる
x not in y	x は y に含まれない

Python では、変数は数値だけでなく、場合によっては図などを表す場合にも使われます。「x is y」などはそのような特別な場合に使用します。数値か文字列の比較は、等号や不等号の比較演算子を使ってください。なお、Python では比較演算子を以下のように連結して使うこともできます。

x < y < z

連結した場合は、それぞれの比較の論理積が最終的な比較結果になります。つまり、以下のようにしたのと同じです。

```
x < y and y < z
```

【練習問題】

以下のコードを入力・実行して戻り値を確認しましょう。

a, b, c = 100, 200, 50

a < b > c

7) Python で使われる特徴的なデータ型

(1) リスト

リストは Python において複数の要素をまとめて扱う際に最もよく使われる重要なデータ型です。リストは [要素, 要素, ...] のように角括弧の中に要素を半角カンマで区切って格納したものです。リストは様々なものを要素にとることができます。また、異なる種類のものを同居させることもできます。以下はいずれも正しいリストです。

```
[1, 2, 3]                          (整数のリスト)
[]                                 (空のリスト)
["Mesh", "Data"]                   (文字列のリスト)
[[1.0, 2.0, 3.0], [2.1, 3.1, 4.1]]  (リストのリスト)
[10, 20, "Mesh", [100, 200, 300]]   (様々な型の要素のリスト)
```

数値と同様、リストは変数に代入することができます。

```
lis = [1, 2, 3]
```

リスト同士は、演算子＋で連結させることができます。

```
In [16]: lis1 = [1, 2, 3]
    ...: lis2 = [4, 5]
    ...: lis1 + lis2
Out[16]: [1, 2, 3, 4, 5]
```

```
In [17]: lis = [1, 2, 3]
    ...: lis += [4, 5]
    ...: lis
Out[17]: [1, 2, 3, 4, 5]
```

ちなみに、算術演算子＊と整数を使ってリストを整数倍することができます。リストが整数個連結されたものが得られます。

リストから特定の要素を取り出すには、要素の番号を角括弧で括ってリストの後ろに付けます。この際、最初の要素を 0 として数えます。

```
In [18]: lis = ["零", "一", "二", "三", "四"]
    ...: lis[1]
Out[18]: '一'
```

リストから複数の要素をまとめて取り出すこともできます。人間の数え方で 2 番目から 6 番目まで、Python の数え方で 1 番目から 5 番目までを取り出すには、`[1:6]`と指定します。

```
In [19]: lis = ["零", "一", "二", "三", "四", "五", "六", "七"]
    ...: lis[1:6]
Out[19]: ['一', '二', '三', '四', '五']
```

範囲の最後の数として指定した数より 1 つ少ない範囲が切り出されることに注意してください。また、特定の要素を取り出した時は要素が裸で（In[18]の例では文字列として）抜き出され、範囲で取り出すとリストで抜き出されることにも注意しましょう。なお、`lis[1,5,6,3]`などのように要素の番号を指定して要素をランダムに並べなおしたリストを作ることはできません。

【練習問題】

上記のほかにも、リストの要素を指定する様々な記法があります。下記はいずれも有効な記法です。それぞれについて要素がどのように指定されるか確認しましょう。

```
lis[:]  lis[3:]  lis[:4]  lis[-5:-3]  lis[-2:]  lis[:-3]
```

リストの要素にリストが含まれているとき、含まれているリストの特定の要素を取り出すには、角括弧を連ねて指定します。下に示すリストが代入されている変数 mixed から、「200」を取り出す場合考えましょう。

```
mixed = [10, 20, "Mesh", [100, 200, 300]]
```

mixed の要素であるリスト`[100, 200, 300]`の mixed における場所は Python の数え方では 3 番目、200 は要素であるリスト`[100, 200, 300]`の Python の数え方で 1 番目なので、これらを用いて`mixed[3][1]`と表記することにより要素を取り出すことができます。

リストは、要素の値を書き換えることができます。

```
In [20]: lis = [1, 2, 3]
    ...: lis[0] = "メッシュ"
    ...: lis
Out[20]: ['メッシュ', 2, 3]
```

```
In [21]: lis[1:3] = ["農業", "気象"]
    ...: lis
Out[21]: ['メッシュ', '農業', '気象']
```

この例では、整数を文字列に書き換えています。このような文を代入文と呼びます。

リストやこの次に出てくるタプル、文字列など順序がついた要素を持ったものをシーケンスと呼び、前述の方法で要素を抽出できるだけでなく、表 4 に示すような操作（メソッド）を使うことができます。メソッドは、シーケンスの後ろにピリオドで連結して使います。表 4 に、リストに用意されているメソッドのなかから、比較的使う機会が多いものを抜粋して示します。

(2) タプル

リストは極めて融通無碍です。この融通無碍さは、時にヒューマンエラーも引き起こします。そこで、Python には、リストに似ているけど融通が利かないタプルというデータ形式が用意されています。

表 4 リストに用意されているメソッドの例

メソッド	機能	例

append	最後尾に要素を一つ追加	`In [#]: lis = [1, 2, 3]`
		`...: lis.append("後はたくさん")`
		`...: lis`
		`Out[#]: [1, 2, 3, '後はたくさん']`

insert	指定した場所に要素を挿入	`In [#]: lis = [10, 20, 30, 40]`
		`...: lis.insert(2, 100)`
		`...: lis`
		`Out[#]: [10, 20, 100, 30, 40]`

pop	指定した場所の要素を削除	`In [#]: lis = ["a","b","c","d"]`
		`...: lis.pop(2)`
		`...: lis`
		`Out[#]: ["a","b","d"]`

index	指定した値の要素が置かれている場所を知らせる(該当がないときはエラーとなる)	`In [#]: lis = ["a","b","c","d"]`
		`...: lis.index("d")`
		`Out[#]: 3`

sort	配列を昇順に並び替える	`In [#]: lis = [10, 1, 7, 2]`
		`...: lis.sort()`
		`Out[#]:[1, 2, 7, 10]`

reverse	並びを反転する	`In [#]: lis = [1, 2, 3, 4]`
		`...: lis.reverse()`
		`Out[#]:[4, 3, 2, 1]`

※sortメソッドはリストを本当に並べ替えてしまいますが、リスト自体は弄らずに「並べ替えるとこうなりますよ」という結果を返すsortedという関数が別にあります。「〇〇するとこうなりますよ」という結果のことをviewと呼びます。viewをうまく使うと、効率的で速いプログラムが書けます。

タプルは一度定義すると要素の中身や数の変更ができません。例えば、緯度経度のデータセットなどは要素数を変更したり、途中で緯度経度の値を変更したりすべきものではないので、リストではなくタプルで定義する方が望ましいといえます。タプルは、角括弧ではなく普通の丸括弧を用いて要素を括ります。

```
tsukuba = (36.0270, 140.1104)
```

タプルは基本的にリストと同じようなことができますが、更新不能なので、要素の変更・増加・並べ替えなどはできません。例えば、append()メソッドで要素を追加しようとすると、エラーが戻ります。

```
In [22]: tsukuba = (36.0270, 140.1104)
    ...: tsukuba.append(150.3)
Traceback (most recent call last):

  File "<ipython-input-17-0eb660df8b49>", line 2, in <module>
    tsukuba.append(150.3)

AttributeError: 'tuple' object has no attribute 'append'
```

(3) 辞書

観測地点の名称、緯度、経度を管理する場合を考えましょう。リストを用意して、1 番目の要素に名称、2 番目の要素に緯度、3 番目の要素に経度を格納する約束にしてリストを作ります。こうすると、例えば、station[0]とすることで地点名が取り扱え便利です。

```
In [23]: station = ["Tsukuba", 36.0270, 140.1104]
    ...: station[0]
Out[23]: 'Tsukuba'
```

しかし、管理項目が増えてくると、「風速計の型式は何番目に書くことにしてあったっけ？」、などと要素の番号とその要素の内容の紐づけがだんだん面倒になってきます。Python には、そのような場合に便利な、辞書と呼ばれる便利なデータ形式があります。

辞書は、中括弧「{}」とコロン「:」を使って、以下のように定義します。コロンの左側を key（キー）、右側の要素を value（値）と呼びます。

☑Pythonでは、括弧の内側ではどんなに改行しても一つの文が続いているものと認識されます。リスト、タプルも同様です。一文が長くなる場合は適宜改行しましょう。

```
In [24]: station = {"name":"Tsukuba",
    ...:            "lat":36.0270,
    ...:            "lon":140.1104,
    ...:            "asl":-999.9}
    ...: station
Out[24]: {'asl': -999.9, 'lat': 36.027, 'lon': 140.1104, 'name': 'Tsukuba'}
```

辞書内の値を検索するときはキーを指定して検索します。また、辞書の値は代入文で書き換えることができます。

```
In [25]: station["name"]
Out[25]: 'Tsukuba'
```

```
In [26]: station["asl"] = 11.3
    ...: station["asl"]
Out[26]: 11.3
```

辞書に格納されている値だけを知りたい場合は、辞書に用意されているメソッド values を使います。また、キーの一覧を知りたい場合は、メソッド keys を使います。

```
In [27]: station.values()
Out[27]: dict_values(['Tsukuba', 36.027, 140.1104, 11.3])
```

```
In [28]: station.keys()
Out[28]: dict_keys(['name', 'lat', 'lon', 'asl'])
```

得られた値はイテラブルオブジェクト（リストと同様、値が並んだもの。後述します。）として使用できます。

Python の制御構文

プログラムによる計算と電卓による計算の決定的な違いは、プログラムが、計算結果に基づいて次の計算式を変更したり、決まりきった計算を何回も行ったりできる点にあります。Python では、繰り返しや条件分岐などを、制御構文という書式で表現します。

1) for 文による繰り返し（ループ）

繰り返しで最も多く使われるのが for 文を使う構文です。これは、以下のように書きます。

```
for 変数 in イテラブルオブジェクト :
    処理
```

☑行末のコロンを忘れずにつけてください。

ただし、

変数：イテラブルオブジェクトから順次取り出された値を格納する変数の名前

イテラブルオブジェクト：要素を順番に取り出せるもの（リスト、タプル、文字列など）

処理：変数に代入された値を使っておこなう処理

【実習 2】Python によるプログラミングの基礎

0、1、2 という 3 つの数字を順に表示させる処理は、for 文を用いて、以下のように表現します。

```
203
204 for i in [0, 1, 2]:
205     print(i)
206
```

```
In [29]: for i in [0, 1, 2]:
    ...:     print(i)
0
1
2
```

○エディタの204行目～205行目を選択し、続けてShift+Enterを押します。

→左のように表示されます。

※print関数は引数の値を出力するための組み込み関数です。

この例では print(i)の行が他の行よりスペース 4 つ分字下げしてあります。**Python では字下げ自体に意味があります。**字下げによって、print(i)が for の繰り返しをおこなう範囲内であることを示しています。上述のコードでは [0, 1, 2] というリストから値を取得するたびに i に格納した値を出力しています。

次に以下の例を実行してみましょう。

```
206
207 for i in [0, 1, 2]:
208     print(i)
209 print(i,"までで終了")
300
```

```
In [30]: for i in [0, 1, 2]:
    ...:     print(i)
    ...:
    ...: print(i,"までで終了")
0
1
2
2 までで終了
```

※209行目のprint文は字下げされていないことに注意してください。

※print関数はカンマで区切ることにより複数の引数を表示させることができます。

こちらのコードでは 1 つめの print 関数は前述のコードと同じく値を取得するたびに値を出力していますが、2 つめの print 関数は字下げせずに書かれているので繰り返しの範囲として取り扱われず、繰り返しが終わったあとで 1 回だけ実行されます。

見えない文字は、半角空白以外に、全角空白やタブもあります。これらが混ざっていても人間には分かりませんが、Python は識別して混乱します。なので、「インデントは空白 4 つ」と心に決めてプログラムを書いてください。そして、以下のエラーメッセージが出たら空白の数が違っていないか、タブが混ざっていないかなどを確認しましょう。

※SpyderではTabキーを押すことによりスペースが4つ入るようになっています

```
IndentationError: expected an indented block
```

上記の例ではイテラブルオブジェクトとしてリストを用いましたが、このような数列のリストを大きな数まで作るのは大変です。このため、整数列を作る range() という組み込み関数が用意されています。range 関数を使って上と同じ処理をする場合は、以下のように要素の数を指定する引数をひとつ与えます

```
211
212 for i in range(3):
213     print(i)
214 print(i,"までで終了")
215
```

range 関数は引数を 3 つまでとることができます。一般的には以下のように指定します。

range(*start*, stop, *step*)

ただし、
start： 最初の数値。省略時はゼロとなる。
stop: step が正の場合は数列の上限値、負の場合は下限値
step: 数値の増加分または減少分。省略時は 1。

【練習問題】
下の for 文に使われている range 関数の括弧に「1, 5」や「1, 10, 2」などを入れて実行し、range 関数の働きを確認しましょう。

```
218
219 for i in range():
220     print(i)
221 print(i,"までで終了")
222
```

2) リスト内包表記

繰り返し計算が何かのリストを作ることを目的にしている場合、リストの括弧で for 文を包んでしまい一行で終わりにしてしまう方法が Python には用意されています。このような記述の方法を、リスト内包表記と呼びます。リスト内包は以下の形式で書きます。

[*式* for *変数名* in *イテラブルオブジェクト*]

☑リスト内包表記ではコロンを使いません。

ただし、
式：変数をもとにリストの要素を計算する式
変数名：イテラブルオブジェクトから順次取り出された値を格納する変数の名前
イテラブルオブジェクト：要素を順番に取り出せるもの（リスト、タプル、文字列など）

例えば、5 の倍数に 1 を足した数を要素とする[1, 6, 11, 16, 21]というリストを、リスト内包表記で作るには以下のように記述します。

```
[i*5+1 for i in range(5)]
```

このリストを通常の for 文で作ると、以下のようなものになります。

```
235
236 x = []
237 for i in range(5) :
238     x.append(i * 5 + 1)
239 x
240
```

リスト内包表記は、for 文よりも高速で、また、リストを作っているのだということが一見してわかるので、慣れると便利です。

3) if, elif, else による条件分岐

計算結果に基づいて次の処理を切り替えることを条件分岐と言います。Python では、条件分岐は if, elif, else という構文を使って以下のように書きます。

```
if 条件式 1:
    処理 1    # 条件式 1 が真(True)の場合の処理

elif 条件式 2:
    処理 2    # 条件式 1 が偽(False)で条件式 2 が真(True)の場合の処理

else:
    処理 3    # 条件式 1 も条件式 2 も偽(False)の場合の処理
```

☑if や else などの行末にはコロンを忘れずにつけてください。

※必要がなければ、elifやelseは使わなくても構いません。また、elifは複数回を使うことができます。

変数 value の値に応じて、出力される文字を切り替えるには、以下のように記述します。

```
244
245 value = 10
246 if value <= 3:
247     print(value, "じゃすくないな。")
248
249 elif 3 < value <= 5:
250     print(value, "だしまあまあか。")
251
252 else:
253     print(value, "だなんてラッキー。")
254
```

【練習問題】
value の値を様々に変えて上記コードを実行してみましょう。

4) break 文と continue 文について

break 文と continue 文は、ループ処理の中に置かれ、処理を中止する役割を果たします。break 文と continue 文では、中止の度合いが異なります。

break 文は、実行された時点で処理を中止し繰り返しも放棄します。

```
258
259 for i in range(10):
260     if i == 3:
261         break
262
263     print(i)
264
```

```
In [31]: for i in range(10):
    ...:     if i == 3:
    ...:         break
    ...:
    ...:     print(i)
0
1
2
```

※ここで繰り返しは10回用意されていますが、iが3のときbreak文が実行されるので、3以降の数が表示されることはありません。

なお、ループが多重にかかっている場合、break 文は直接のループだけを中止します。

continue 文は実行された時点で処理を中止し次の繰り返しに移行します。ループの繰り返し自体は中断されません。

```
264
265 for i in range(10):
266     if i == 3:
267         continue
268
269     print(i)
270
```

```
In [32]: for i in range(10):
    ...:     if i == 3:
    ...:         continue
    ...:
    ...:     print(i)
0
1
2
4
5
6
7
8
9
```

※繰り返し途中、iが3の時は無条件に次の繰り返しに移ります。3だけが表示されません。

break 文や continue 文は、上記のように、通常 if などの条件分岐と組み合わせて用いられます。

Python の関数

気温変化のグラフを複数の地点について作成する場合、グラフを描くプログラムをひとまとめにして作っておき、必要となったときにそれを呼び出して使えたら便利です。このように、プログラムのいくつかの処理をひとまとめにして繰り返し利用できるようにしたものを関数と呼びます。数学の関数は、数を与えて数を決めるものですが、プログラミングにおける関数はより広い概念であり、「機能」に近いものです。Python では、関数を以下の構文で定義します。

```
def 関数名(引数, キーワード引数 = 値):
    処理
    return 戻り値
```

☑行末のコロンを忘れずにつけましょう。

☑ 引数、キーワード引数、処理、戻り値は複数指定できます。

ただし、

関数名：関数につける名前　（既存の関数の名前と重複しないように注意しましょう）

引数:関数に受け渡す情報を入れる変数　（関数を使うときに指定が必須な変数）

キーワード引数：関数を使うときに必ずしも指定する必要がない引数

処理：目的の処理をさせるための文の集まり

戻り値：処理終了時に出力したい内容　（複数も可、なしも可）

もっとも初歩的に、a と b、2 つの数を受け取ってその和を返す関数をつくり、それに calc_add と名付けてみましょう。それは以下のようになります。

```
273
274 def calc_add(a, b):
275     c = a + b
276     return c
277
```

※ 関数をプログラム中で使うとき、引数（x1やx2）や戻り値（y）に使用する変数名に特別な制約はありません。y でも x でも、定義に際して使ったaや c も使えます。ただし、関数は使う前に（この例では y = calc_add(x1, x2)よりも上の行で）定義しておかなければなりません。

このようにして作った関数は、例えば以下のようにしてプログラムで使うことができます。

```
x1, x2 = 2, 3
y = calc_add(x1, x2)
y
```

関数は複数の処理結果を出すこともできます。二つの数を受け取って、その和と差を計算する関数 calc_addif は下のように定義することができます。

```
277
278 def calc_addif(a, b):
279     c = a + b
280     d = a - b
281     return c, d
282
```

二つの結果（戻り値）を返す関数は、例えば以下のようにして結果を受け取ります。

```
x1, x2 = 2, 3
y, z = calc_addif(x1, x2)
y, z
```

関数 calc_addif は、引数を二つ付けて定義したので、使うときも二つの値を必ず与えなければなりません。これが普通の使い方ですが、Python では、キーワード引数と呼ばれる特別な形式の引数を使うことができます。キーワード引数は、定義するときに、「引数＝値」として、値まで書いておきます。そして、使うときもやはり「引数＝値」として使います。この際、値は同じでも別でもかまいません。さらに、これ自体を省略することもできます。これは、デフォルト値を用意しておくときに便利です。使うときに何も書かなければ定義において指定された値が用いられ、キーワード引数が書かれていたらそれが使用されるからです。

二つの数を受け取って和をとり、それに消費税を加える関数 calc_add_tax は、以下のように定義することができます。

```
282
283 def calc_add_tax(a, b, tax=0.08):
284     c = a + b
285     c += c * tax
286     return c
287
```

キーワード引数を持つ関数は例えば以下の 2 通りの使い方ができます。

```
calc_add_tax(100, 200)  # デフォルト税率の 8%を採用する場合

calc_add_tax(100, 200, tax=0.05)  # 税率 5%を採用する場合
```

※このケースでは、calc_add_tax(100, 200, 0.05)でも動きますが、このような使い方はしないようにしましょう。

モジュール（ライブラリ・パッケージ）とインポート

とても便利な関数ができたとき、それを別なプログラムでも使いたくなります。プログラムそれぞれにその関数定義をコピペしても使いまわしは可能ですが、あまり賢い方法とは言えません。Python では便利な関数を一つのファイルにまとめておき、別なファイルに書かれているプログラムからその中身を取り出すことができます。便利な関数やデータ型を集めたファイルをモジュールとよびます。Python をインストールしたときに一緒にインストールされるモジュールを標準ライブラリと呼びます。一方、Python のインストールとは別にインストールが必要な、第三者が作ったモジュールを外部モジュールまたはサードパーティー製ライブラリ（もしくはサードパーティ製パッケージ）と呼びます。環境構築の際に `conda install` というコマンドでインストールした netcdf4、pyproj などは外部モジュール・ライブラリです。

※機能や関数によっては、一つのファイルには収まらず、ディレクトリ構造を持つ複数のファイルで構成されているようなものもあります。それらはパッケージと呼ばれます。Python ではパッケージやモジュールを総称してライブラリと呼びます。

AMD_Tools3.py も外部モジュールの一つです。メッシュ農業気象データを処理するには、とにもかくにもデータ配信サーバーからデータを取得する必要がありますが、その役割を担う関数は AMD_Tools3 に GetMetData 関数として収められており、みなさんはこれを利用することができます。なお、この GetMetData 関数自体も、一からプログラミングされているわけではなく、Python コミュニティが開発した様々なモジュール・ライブラリを使用しています。

これらの標準ライブラリ・外部モジュールを呼び出し、利用可能とする操作をインポートと呼びます。インポートは簡単で、import 文と呼ばれる文をプログラムの始めに書いておくだけです。作業スペースに格納した AMD_Tools3.py を開いてみましょう。38～58 行目にはたくさんの import 文が記述されています(図 2)。

```
38 from sys import exit
39 from os import unlink #,fdopen
40 from os.path import join,exists,isdir,basename
41 from datetime import datetime as dt
42 from datetime import timedelta as td
43 from math import floor
44 import numpy as np
45 import numpy.ma as ma #PutKMZ_Mapで使ってる
46 import tempfile
47 from random import randint
48 from netCDF4 import date2num, num2date, Dataset
49 import codecs
50 import urllib
51 import urllib.request
52 import ssl
53 # リモートで動かすときに必要
54 # import matplotlib
55 # matplotlib.use("Agg")
56 import matplotlib.pyplot as plt
57 import matplotlib.cm as cm
58 from matplotlib.dates import DateFormatter,DayLocator
```

図 2　AMD_Tools3.py に見られるインポート文

ライブラリに格納されている関数は、import 文でライブラリをインポートしたのち、以下のようにして使うのが基本です。

ライブラリ名.*関数名*(引数, 引数, ・・・)

※ライブラリ名と関数名をピリオドで繋ぎます。

しかし、ライブラリの名称は必ずしも短くないので、この記法だと不便なことが多々あります。また、多種多様なライブラリには名前が同じなのに機能が違う関数がたくさんあり、関数名だけでは混乱が生じます。そこで、ライブラリ名や関数名に別名を付けたり、関数を限定してライブラリ名を付けずに使えるようにしたりする方法が用意されました。図 2 に記されるインポート文が様々な記法なのはそのためです。この実習では、その中から 2 つだけ説明します。一つめは以下の記法です。

```
import ライブラリ  as  略称
```

このようにしてインポートした関数は、以下のようにして使います。

略称.*関数名*(引数, 引数, ・・・)

※略称と関数名をピリオドで繋ぎます。

たとえば AMD_Tools3.py をインポートする場合は以下のように書き、

```
import AMD_Tools3  as  AMD
```

AMD としてインポートした AMD_Tools3.py 内の関数 GetMetData を使う場合は以下のように書きます。

```
AMD.GetMetData(nani, itsu, doko)
```

二つめは以下の記法です。

```
from ライブラリ import 関数, 関数, ・・・
```

このようにしてインポートした場合は、ライブラリ名を付けずに使用できます。

※この方法でインポートした場合、このライブラリに用意されている他の関数は使うことができません。

たとえば図 2 の 43 行目にある

```
from math import floor
```

はこの記法になり、以下のように使うことができます。

```
floor(3.14)
```

※floorは与えた値xの「床」 (x 以下の最大の整数) を返す関数です。この例では3が戻ります。

数値計算ライブラリ NumPy

NumPy は数値計算を効率的に行うためのライブラリです。多数の要素をひとまとまりにしたものとして、Python にはリストがあることを学習しました。このリストで1次元や2次元のデータを取り扱うことも不可能ではありませんが、後述する NumPy の提供するデータ型 ndarray を使えば、圧倒的に強力かつ高速にデータを処理することができます。メッシュ農業気象データのような 3 次元の大きいデータを扱う際には必須のライブラリといってよいでしょう。

NumPy は大変有名なライブラリなので、インポートの方法も半ば慣例化しています。以下のようにしてインポートします。

```
import numpy as np
```

○Spyder で実際に NumPy をインポートしてみてください。プロンプトにこの行を入力し、Enterキーを押し、次のInが表示されたらインポート成功です。

1）多次元数値配列 ndarray

ndarray は型が均一（たとえば浮動小数点数のみ、整数のみなど）な多次元の数値配列（行列など）の操作を行うために用意されたデータ型で、numpy をインポートすると使えるようになります。

np.array 関数を使うと数値のみのリストやタプルから ndarray を生成することができます。

```
In [33]: data = [1, 2, 3]
    ...: npdata = np.array(data)
    ...: npdata

In [33]: array([1, 2, 3])
```

※Numpyをインポートしていないとエラーになります

ndarray には算術演算子を適用できます。ndarray 同士の演算では、相当する要素同士が演算されたものが得られます。

```
In [34]: npdata + npdata
Out[34]: array([2, 4, 6])
```

```
In [35]: npdata - npdata
Out[35]: array([0, 0, 0])
```

```
In [36]: npdata * npdata
Out[36]: array([1, 4, 9])
```

```
In [37]: npdata / npdata
Out[37]: array([1., 1., 1.])
```

ndarray と数値との演算では、全ての要素に対して数値が演算された結果が得られます。

```
In [38]: npdata + 10
Out[38]: array([11, 12, 13])
```

```
In [39]: npdata - 10
Out[39]: array([-9, -8, -7])
```

```
In [40]: npdata * 10
Out[40]: array([10, 20, 30])
```

```
In [41]: npdata / 10
Out[41]: array([0.1, 0.2, 0.3])
```

ndarray とリストを演算すると、リストが ndarray に変換されて、ndarray として演算された結果が戻ります。

```
In [42]: npdata / 10 * data
Out[42]: array([0.1, 0.4, 0.9])
```

【練習問題】

In[34]から[42]について、npdata を data に置き換えて実行し、npdata の演算結果との違いを比較してみましょう。

※ npdata は ndarray 型、dataはリスト型です。

2) 3 次元 ndarray の添え字

メッシュ農業気象データは、日付、緯度、経度の次元からなる 3 次元データとみなすことができるので、3 次元の ndarray データとして扱うのが便利です。3 次元の ndarray を作成する方法はいくつかありますが、所定のサイズの 3 次元配列を手っ取り早く新規作成するには、値がすべて 0.0 の配列を作る numpy.zeros 関数を用いるのが便利です。

この関数を用いて、一番目の次元（Python の数え変え方では 0 次元）の要素が 3 個、二番目の次元(同 1 次元)の要素が 4 個、三番めの次元（同 2 次元）の要素が 5 個の 3 次元のゼロ配列 arr を新規作成するには以下のようにします。

```
arr = np.zeros((3, 4, 5))
```

☑丸括弧が二重になっていることに注意しましょう

このようにして作った配列の中身を Spyder で表示させると、角括弧で多重に括られた下の形で表示されます。

```
In [43]: arr = np.zeros((3, 4, 5))
    ...: arr
Out[43]:
array([[[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]])
```

今度は、別な方法で、各要素に格納される値が 1 ずつ増える同じサイズの 3 次元配列を作ってみます。

```
In [44]: arr = np.arange(60.0).reshape(3, 4, 5)
    ...: arr
Out[44]:
array([[[ 0.,  1.,  2.,  3.,  4.],
        [ 5.,  6.,  7.,  8.,  9.],
        [10., 11., 12., 13., 14.],
        [15., 16., 17., 18., 19.]],

       [[20., 21., 22., 23., 24.],
        [25., 26., 27., 28., 29.],
        [30., 31., 32., 33., 34.],
        [35., 36., 37., 38., 39.]],

       [[40., 41., 42., 43., 44.],
        [45., 46., 47., 48., 49.],
        [50., 51., 52., 53., 54.],
        [55., 56., 57., 58., 59.]]])
```

※関数numpy.arange(プログラム中ではnp.arange)は、関数rangeと似た機能の関数ですが、計算が速く、整数以外の増分を指定できるなどより多機能です。

※reshapeは、配列の形状を変えて見せる(Viewを作る)ndarray型のメソッドです。

さて、このような配列から、特定の要素を取り出すには、添え字をどのように指定したらよいでしょうか。3 次元 ndarray は、図 3 のような 3 つの軸に沿った立体と考えると理解が容易です。

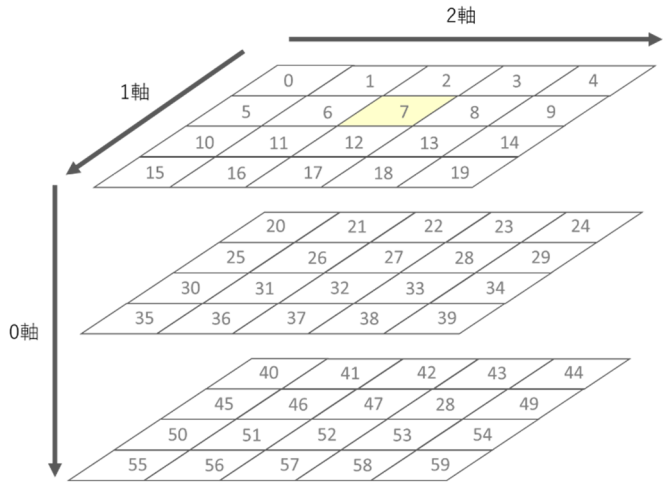


図 3　3 次元 ndarray `arr` の模式図

※図では、小数点以下を省略しています。

図 3 中、7 の値が入っている要素は、0 軸において 0 番目、1 軸において 1 番目、2 軸において 2 番目(いずれも Python の数え方)なので、これらを以下のように一つの角括弧の中に指定することにより抽出できます。

```
In [45]: arr[0, 1, 2]
Out[45]: 7.0
```

今度は、0 軸方向の 0 から数えて 2 番目の、1 軸・2 軸のすべての要素を取り出してみます(図 4)。

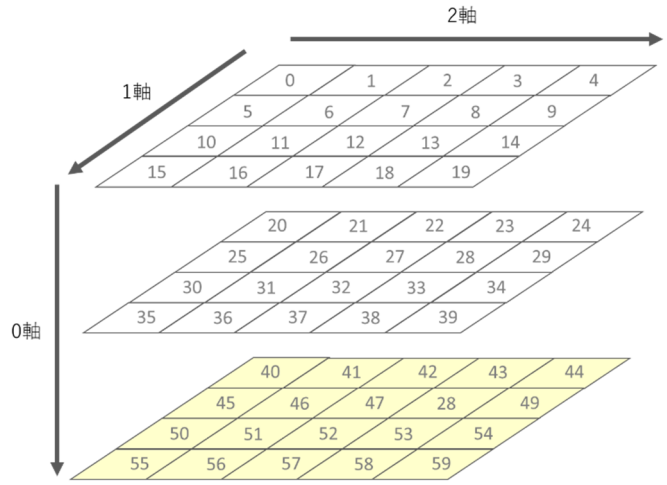


図 4　0 軸方向の 0 から数えて 2 番目のすべての要素(黄色)

この場合は、以下のようにコロンを用いて指定します。

```
In [46]: arr[2, :, :]
Out[46]:
array([[40., 41., 42., 43., 44.],
       [45., 46., 47., 48., 49.],
       [50., 51., 52., 53., 54.],
       [55., 56., 57., 58., 59.]])
```

ここで、コロンはすべての要素を選択することを意味します。同様に 1 軸において 2 番目、2 軸において 3 番目（いずれも Python の数え変え方）のすべての要素（図 5）を抽出する場合は以下のように指定します。

```
In [47]: arr[:, 2, 3]
Out[47]: array([13., 33., 53.])
```

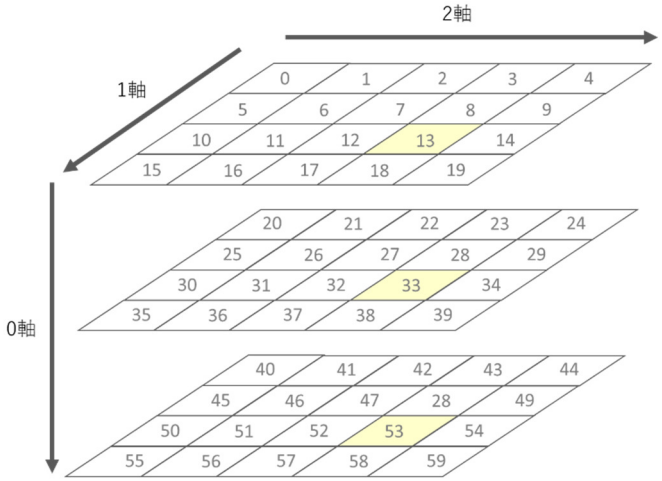


図 5　array[:, 2, 3]の選択範囲

3）無効値 nan

nan は、NumPy をインポートすると利用可能となる特別な浮動小数値で、「数として扱えない数」（not a number）を示します。数として扱えないので、nan と何かを演算すると結果は nan になります。また、nan と何かを比較しようとするとエラーとなります。

メッシュ農業気象データは、日本全国の陸地に対してデータが作られており、海上や湖沼上のデータはありません。また、日平均湿度など平年値がない気象要素では、未来のデータがありません。AMD_Tools3.GetMetData 関数は、このようなメッシュのデータを読み込むと、対応する配列要素に nan を代入します。

関数 numpy.isnan は、引数に浮動小数やその ndarray をとり、nan が格納されている要素を調べ、True（真）または False（偽）を返す関数です。この様子を見るために、図 3 で示した場所に nan を代入してみます。

※無効値の名前は nan ですが、ライブラリ numpy に定義されていて、しかもそれを略称 np でインポートしているので、プログラム中で何かに代入したりするには「np.nan」と書かなければなりません。

※nan を代入する際、元の配列 arr も浮動小数点型でないとエラーが出ます。

```
In [48]: arr[0, 1, 2] = np.nan
Out[48]:
array([[[ 0.,  1.,  2.,  3.,  4.],
        [ 5.,  6., nan,  8.,  9.],
        [10., 11., 12., 13., 14.],
        [15., 16., 17., 18., 19.]],

       [[20., 21., 22., 23., 24.],
        [25., 26., 27., 28., 29.],
        [30., 31., 32., 33., 34.],
        [35., 36., 37., 38., 39.]],

       [[40., 41., 42., 43., 44.],
        [45., 46., 47., 48., 49.],
        [50., 51., 52., 53., 54.],
        [55., 56., 57., 58., 59.]]])
```

この状態で、関数 numpy.isnan に配列 arr を与えると、下のように、データがあれば False、nan であれば True の 3 次元論理値配列が返されます。

```
In [49]: np.isnan(arr)
Out[49]:
array([[[False, False, False, False, False],
        [False, False,  True, False, False],
        [False, False, False, False, False],
        [False, False, False, False, False]],

       [[False, False, False, False, False],
        [False, False, False, False, False],
        [False, False, False, False, False],
        [False, False, False, False, False]],

       [[False, False, False, False, False],
        [False, False, False, False, False],
        [False, False, False, False, False],
        [False, False, False, False, False]]], dtype=bool)
```

関数 numpy.isnan の頭にチルダを付けて~numpy.isnan として用いると、真偽が逆転した結果を得ることができます。

【練習問題】

実行コンソールに今度は以下のように入力し、チルダにより戻り値の真偽が反転することを確認してください。

```
~np.isnan(arr)
```

○ ~numpy.isnan の出力結果を numpy.isnan の出力結果と比較しましょう。

4) クラスの属性

関数 len は組み込み関数で、リストやタプル、ndarray などのイテラブルオブジェクトの要素を数えてくれる関数です。len 関数にリストを渡すと、リストの中の要素の数を戻します。リストの要素にリストが格納されているとき、そのリストをばらすことまではせず、一つの要素として数えます。ndarray の配列を関数 len に渡すと、多次元の場合は次元数が戻され、1 次元の場合は要素数が戻されます。

```
In [50]: len(arr)
Out[50]: 3
```

```
In [51]: len(arr[0])
Out[51]: 4
```

ある ndarray が何次元でそれぞれの次元の要素数はいくつかを知るには、ndarray が持つ、属性 shape を呼び出すと便利です。一般に、特別なデータ型(クラス)にはそれを定義するときに必要となる基本的な数値があり、内部的に保存されています。これを属性(attribute)と呼びます。以下のようにすると、属性を呼び出すことができます。

変数名.属性変数名

※変数名と属性変数名をピリオドで繋ぎます。

ndarray というデータ型は、多次元の数値だけでなく、自らの形を shape という名の属性として保持しているので、これを呼び出せば、配列の形と要素数がすぐわかります。

```
In [52]: arr.shape
Out[52]: (3, 4, 5)
```

※配列の次元数や要素数は変化するべきものではないので、タプルで保存されています。

属性の呼び出し方はメソッドに似ていますが、括弧を付けないことに注意してください。属性変数は変数の仲間なので括弧は不要ですが、メソッドは関数の仲間なので、引数があろうとなかろうと括弧が必要なのです。

参考文献

1. 「Python 文法詳解」(2014) 石本敦夫著（オライリージャパン）
2. 「いちばんやさしい Python の教本 人気講師が教える基礎からサーバサイド開発まで（「いちばんやさしい教本」シリーズ）(2017) 鈴木たかのり、杉谷弥月、株式会社ビープラウド著（インプレス）
3. 「Python によるデータ分析入門 —NumPy、pandas を使ったデータ処理」(2013) Wes McKinney 著、 小林 儀匡・鈴木 宏尚・瀬戸山 雅人・滝口 開資・野上 大介訳（オライリージャパン）

参考 URL

1. Python 3.7.3 ドキュメント
   https://docs.python.jp/3/index.html
2. Python 3.7.3 ライブラリリファレンス
   https://docs.python.jp/3/library/index.html
3. Python Boot Camp テキスト
   http://bootcamp-text.readthedocs.io/textbook/index.html
4. NumPy
   http://www.numpy.org/

【実習 3】メッシュ農業気象データの取得 2 Python の利用

農研機構 北海道農業研究センター　根本 学

python プログラミングによるデータ取得（とにかく、取得してみる）

エクセルシートの機能でやりたいことが実現できない場合には、一歩踏み込んで、自分でプログラミングを行う必要があります。その際に、メッシュ農業気象データを利用するためのプログラミング言語は Python に限りませんが、Python 利用だと、①開発チームによるメッシュ農業気象データ利用のためのライブラリが用意されていること、②実行環境が容易に構築できること、などのメリットがあり、他のプログラミング言語を使用する場合よりもはるかに簡単だといえます。ここでは、詳細な説明は抜きにして、とにかく Python でデータを取得してみましょう。

1）下準備

(1) Anaconda / Spyder を起動し、ファイルエクスプローラーで作業するフォルダを表示

(2) AMD_Tools3.py をエディタ画面に表示し、利用者 ID とパスワードを所定の位置（図 1）に記入して保存（前後のダブルクオテーションを残して文字列を消し、そこに利用者 ID もしくはパスワードを入力）

```
#=====================================
USER="ここにIDを記入する"
PASSWORDS=["ここにパスワードを記入する","二つまで設定できる"]
#=====================================
```

図 1 AMD_Tools3.py 内の利用者 ID およびパスワード記入箇所

2）データの取得

ここでは、実行コンソール（実習 2 図 1 の②の部分）で操作してみます。以下の図に従って、1 行ずつ入力し、リターンキーを押して実行してみましょう。

```
In [1]: import AMD_Tools3 as AMD

In [2]: nani = "TMP_mea"

In [3]: itsu = ["2017-04-01","2017-10-31"]

In [4]: doko = [ 43.0095, 43.0095, 141.4080, 141.4080 ]

In [5]: TMP, tim, lat, lon = AMD.GetMetData(nani, itsu, doko)
TMP_mea (214, 1, 1)

In [6]: TMP = TMP[:,0,0]
```

図 2 Python によるメッシュ農業気象データ取得の例
1 行目: ライブラリ（AMD_Tools3.py）を AMD としてインポート
2 行目: 取得したい要素を指定（TMP_mea は日平均気温）
3 行目: 取得期間の指定
4 行目: 取得範囲の指定
5 行目: データの取得（TMP に日平均値、tim に日付、lat に緯度、lon に経度の情報が記憶される）
6 行目: 緯度経度の次元を無くした 1 次元の配列に変換

【実習 3】メッシュ農業気象データの取得 2 Python の利用

ここまでで、欲しいメッシュ農業気象データが取得できています。
データを画面表示するには、図 3 のように　print 関数を使用します。

```
In [7]: print(TMP)
[  1.34699106   2.64574981   5.11678553   5.47043467   7.72118711
  10.43045902   7.26449251   4.57730198   5.79152679   5.70665932
   6.22114134   2.79110742   1.05759621   9.87066174  13.13363266
  11.19481945   8.17222595   6.8425107    5.56662226   3.80877686
   5.96142197   3.68501115   5.02329922   7.65051603  10.42037868
   9.55707169   8.09294224   8.98794842  10.46194363   9.83893394
   8.60268116   9.03901672  13.82842445  16.21395302  16.18736839
  12.75365067  11.94936371   8.74837494  11.3544445   12.21031857
  12.13356972  13.02930927  10.20945072   8.85198784   9.35840416
```

図 3 取得した気象データの表示例

4 月 1 日から 10 月 31 日に対応した 214 データが順番に表示されている（図に収まっていません。）

データをファイルとして保存したい場合は、用意されたライブラリから PutCSV_TS 関数を利用すると、csv 形式で保存することができます(図 4)。

```
In [8]: AMD.PutCSV_TS(TMP, tim)
```

図 4 PutCSV_TS 関数の使用例
TMP に気温データ、tim に日付の情報が含まれています。

上記のコマンドを実行すると、result.csv というファイルが作成されます。コマンドを実行したフォルダにファイルが作成されるので、Spyder 画面右上のファイルエクスプローラーでも確認できます。これをダブルクリックして開くと、エディタ画面に図 5 のように表示されます。

untitled0.py　result.csv

```
1 2017-04-01 00:00:00,1.34699
2 2017-04-02 00:00:00,2.64575
3 2017-04-03 00:00:00,5.11679
4 2017-04-04 00:00:00,5.47043
5 2017-04-05 00:00:00,7.72119
6 2017-04-06 00:00:00,10.4305
7 2017-04-07 00:00:00,7.26449
8 2017-04-08 00:00:00,4.5773
9 2017-04-09 00:00:00,5.79153
10 2017-04-10 00:00:00,5.70666
11 2017-04-11 00:00:00,6.22114
12 2017-04-12 00:00:00,2.79111
13 2017-04-13 00:00:00,1.0576
14 2017-04-14 00:00:00,9.87066
15 2017-04-15 00:00:00,13.1336
16 2017-04-16 00:00:00,11.1948
17 2017-04-17 00:00:00,8.17223
18 2017-04-18 00:00:00,6.84251
```

図 5 result.csv の中身
4 月 1 日から 10 月 31 日までの 214 データ（ここでは日平均気温）が日付とともにコンマ区切りで表示されている（図に収まっていません。）
数値は、図 7 と同じ値になっています。

ところで、PutCSV_TS 関数ってどんな関数だろう？と思ったときは、help 関数を使うと、関数の説明を表示させて確認することができます（図 6）。

```
In [9]: help(AMD.PutCSV_TS)
Help on function PutCSV_TS in module AMD_Tools3:

PutCSV_TS(Var, tim, header=None, filename='result.csv')
    概要:
        時系列のデータをCSV形式のファイルで出力する関数
    書式:
        PutCSV_TS(Var, tim, header=None, filename='result.csv')
    引数(必須):
        Var:時系列の1次元配列データ。ただし、「Ver=np.array([V1,V2,..,Vn])」とすれ
            ば、n個の1次元配列V1,V2,..,Vnを一度に出力することができる。
        tim:時刻の見出しとして使用される一次元配列。第1列に行方向に出力される。
    引数(必要に応じ指定):
        header:引数に「header='moji,retsu'」として文字列を与えると、出力ファイルの
            第1行目にこの文字列が出力される。
        filename:引数に「filename='fairun_no_namae.csv'」として文字列を与えると、
            ファイルがこの名前で出力される。これを指定しない場合は、デフォルトの
            ファイル名「result.csv」が用いられる。
    戻り値:なし
```

図 6 help 関数の使用例

AMD としてインポートしたライブラリ中の PutCSV_TS 関数を確認しています。

※　表示される説明文は、各関数の”””“””で囲まれたコメント文になります。

ちなみに、ライブラリファイル”AMD_Tools3.py”を開くと、その冒頭に、ライブラリに含まれる関数がコメント文として表示されます（図 7）。

図 7 AMD_Tools3 ライブラリーの中の関数

各関数の詳細な利用方法は、先ほどのように help 関数を利用して確認できるので、活用してください。もちろん、AMD_Tools3 に含まれる以外の関数にも help 関数を使用することができます。(詳細な説明が用意されていない関数もあります。)

※ 公開 wiki（https://amu.rd.naro.go.jp/wiki_open/doku.php?id=python）には、グラフや空間分布を作成するサンプルプログラムもありますので、こちらも参考にしてください。

【実習 3】メッシュ農業気象データの取得 2 Python の利用

おまけ：その他プログラミングによるデータ利用

Python 言語を利用する以外の方法でも、メッシュ農業気象データを取得することは可能です。
実習 1 の図 3 の画面で、指定した時間、緯度、経度のパラメータを入力すると、DataURL の欄に以下の文字列が表示されています。

```
https://amd.rd.naro.go.jp:443/opendap/AMD/Area1/2017/AMD_Area1_TMP_mea.nc?TMP_mea[90:303][441][192]
```

この URL に、欲しい気象要素（記号：TMP_mea）、期間（2017、90:303 のインデックスに対応）、場所（Area1 と 441 と 192 のインデックスに対応）の情報が埋め込まれています。この URL の、"nc"と"?"の間に".ascii"を追加した URL にアクセスすることで、テキストとしてデータを取得することができます。

※サーバーの認証方式は Basic 認証を使用しており、ユーザーID とパスワードを用いてアクセスするための適切な手続きが必要です。

1）気象要素について

気象要素取得ための記号は以下の通りです。

- 日平均気温： TMP_mea ※ 平年値は TMP_mea_cli
- 日最高気温： TMP_max ※ 平年値は TMP_max_cli
- 日最低気温： TMP_min ※ 平年値は TMP_min_cli
- 降水量： APCP ※ 平年値は APCP_cli
- 1mm 以上の降水の有無： OPR ※ 平年値は OPR_cli
- 日照時間： SSD ※ 平年値は SSD_cli
- 全天日射量： GSR ※ 平年値は GSR_cli
- 下向き長波放射量： DLR
- 日平均相対湿度： RH
- 日平均風速： WS
- 積雪深： SD
- 積雪相当水量： SWE
- 日降雪相当水量： SFW

※ 最新情報、および詳細情報については、メッシュ農業気象データの公開ページ「多彩な気象要素が用意されています」にて確認できます。

2) 日付のインデックスについて

日付のインデックスは、1 月 1 日を 0 として、1 月 1 日からの日数に対応しています。表 1 は日付インデックスの早見表です。この表から 4 月 1 日に対応するインデックスは 90、10 月 31 日に対応するインデックスは 303 であることを確認してください。

表 1　メッシュ農業気象データの日付インデックス一覧

うるう年の場合は、1を加える

	1月	2月	3月	4月	5月	6月	7月	8月	9月	10月	11月	12月
1日	0	31	59	90	120	151	181	212	243	273	304	334
2日	1	32	60	91	121	152	182	213	244	274	305	335
3日	2	33	61	92	122	153	183	214	245	275	306	336
4日	3	34	62	93	123	154	184	215	246	276	307	337
5日	4	35	63	94	124	155	185	216	247	277	308	338
6日	5	36	64	95	125	156	186	217	248	278	309	339
7日	6	37	65	96	126	157	187	218	249	279	310	340
8日	7	38	66	97	127	158	188	219	250	280	311	341
9日	8	39	67	98	128	159	189	220	251	281	312	342
10日	9	40	68	99	129	160	190	221	252	282	313	343
11日	10	41	69	100	130	161	191	222	253	283	314	344
12日	11	42	70	101	131	162	192	223	254	284	315	345
13日	12	43	71	102	132	163	193	224	255	285	316	346
14日	13	44	72	103	133	164	194	225	256	286	317	347
15日	14	45	73	104	134	165	195	226	257	287	318	348
16日	15	46	74	105	135	166	196	227	258	288	319	349
17日	16	47	75	106	136	167	197	228	259	289	320	350
18日	17	48	76	107	137	168	198	229	260	290	321	351
19日	18	49	77	108	138	169	199	230	261	291	322	352
20日	19	50	78	109	139	170	200	231	262	292	323	353
21日	20	51	79	110	140	171	201	232	263	293	324	354
22日	21	52	80	111	141	172	202	233	264	294	325	355
23日	22	53	81	112	142	173	203	234	265	295	326	356
24日	23	54	82	113	143	174	204	235	266	296	327	357
25日	24	55	83	114	144	175	205	236	267	297	328	358
26日	25	56	84	115	145	176	206	237	268	298	329	359
27日	26	57	85	116	146	177	207	238	269	299	330	360
28日	27	58	86	117	147	178	208	239	270	300	331	361
29日	28	59	87	118	148	179	209	240	271	301	332	362
30日	29		88	119	149	180	210	241	272	302	333	363
31日	30		89		150		211	242		303		364

3) Area と緯度経度のインデックスについて

メッシュ農業気象データは、図 8 の赤枠で示される 6 つの Area に分かれて、サーバー上に置かれています。そこで、まず自分が得たいメッシュデータの緯度経度が、この 6 つのエリアのどれに入るかを判定します。各 Area の範囲は、表 2 の通りになっています。

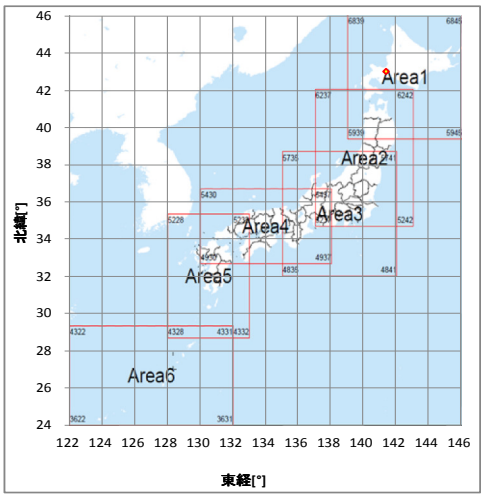


図 8　メッシュ農業気象データの 6 つのエリア

北農研の気象観測露場の場合、北緯 43.0095° 東経 141.4080° ですから、Area1 に含まれます。

ちなみに、6 つの Area はお互いに重なり合っていますので、得たい緯度経度が含まれる Area は一つとは限りません。例えば、青森県の全域は、Area1 にも、Area2 にも含まれています（図 8）。この場合、どちらの Area からもメッシュ農業気象データを得ることが可能ですが、後で説明する緯度と経度のインデックス値が異なってきます。

表 2　6 つのエリアの情報

地域の記号	Area1	Area2	Area3	Area4	Area5	Area6
北端の北緯（°）	46	42	38.66667	36.66667	35.33333	29.33333
南端の北緯（°）	39.33333	34.66667	32	32.66667	28.66667	24
西端の東経（°）	139	137	135	130	128	122
東端の東経（°）	146	143	142	138	133	132
南北方向の要素数	800	880	800	480	800	640
東西方向の要素数	560	480	560	640	400	800

3 次メッシュの大きさは、図 9 のように定義されています。1 次メッシュは、緯度方向に 40 分（2/3 度）経度方向に 1 度の大きさになっています。2 次メッシュは、1 次メッシュを緯度・経度方向にそれぞれ 8 分割にした大きさになっています。3 次メッシュは、2 次メッシュを緯度・経度方向にそれぞれ 10 分割にした大きさです。つまり、3 次メッシュの緯度方向の大きさは　2/3/8/10 ＝ 0.0083333°、経度方向の大きさは　1/8/10 ＝ 0.0125° となっています。

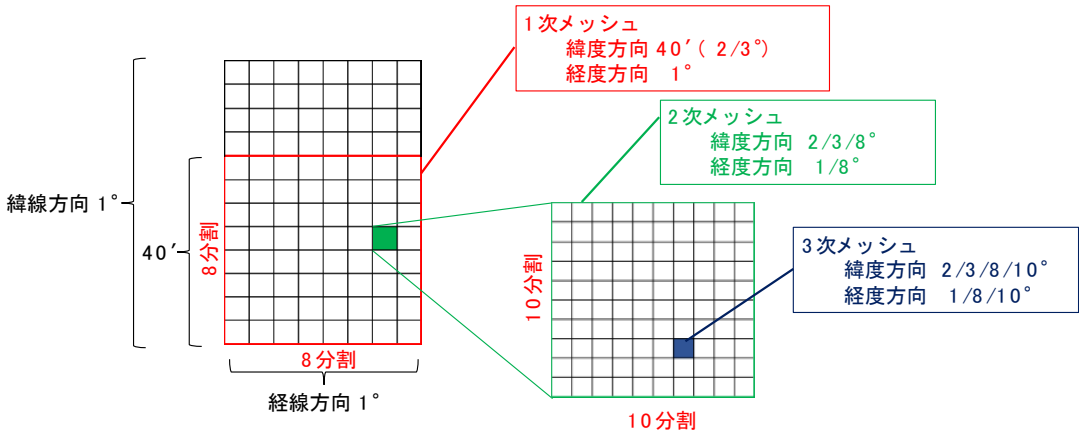


図 9　3 次メッシュの大きさ

メッシュの緯度と経度のインデックスは、緯度方向には南から、経度方向には西から順番に振られています。図 10 は Area1 について説明したものですが、他の Area についても同様です。

Area1 における、北農研の気象観測露場（北緯 43.0095° 東経 141.4080° ）の緯度経度のインデックスは以下のように計算できます。

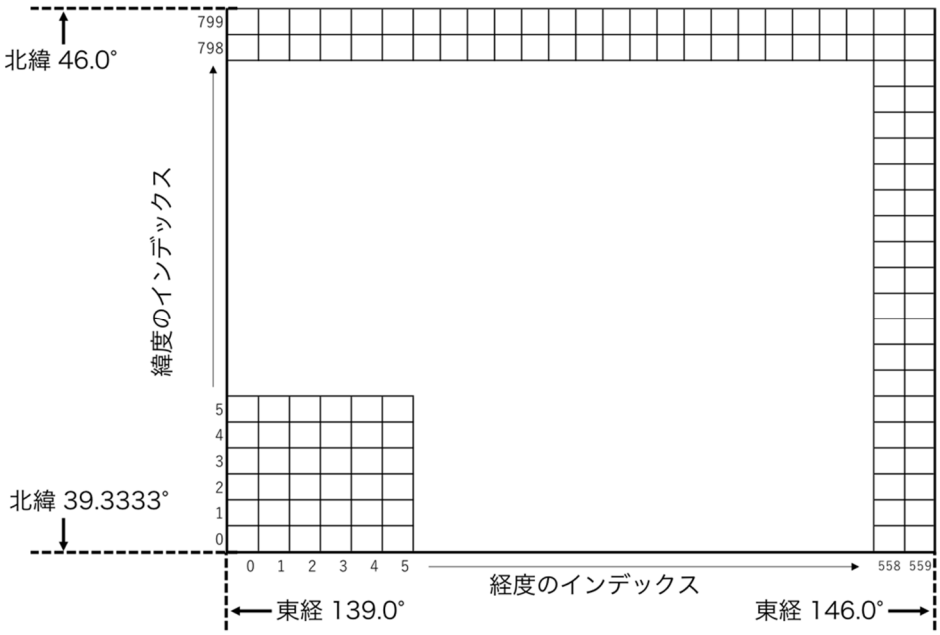


図 10　Area 内のインデックスの数え方（例：Area1）

北緯のインデックス ＝ （43.0095 – 39.3333[Area1 の南端の緯度]）
÷ 0.0083333°［緯度方向の 3 次メッシュの大きさ］
＝ 441.14 （Area1 の南端から 441 番目のメッシュの
南端緯度と、
442 番目の南端緯度の間の緯度の間）
東経のインデックス ＝ （141.4080 – 139.0 [Area1 の西端の緯度]）
÷ 0.0125°［経度方向の 3 次メッシュの大きさ］
＝ 192.64 （Area1 の西端から 192 番目のメッシュの
西端緯度と、
193 番目の西端緯度の間の緯度の間）

以上より、北農研の気象観測露場が含まれる Area1 の緯度経度のインデックスは、441 と 192 となります。

おわりに

以上のように、メッシュ農業気象データ利用のためのライブラリ AMD_Tools3.py では、上記の煩雑な Area の選択とインデックスの計算を、計算していることを意識せずに使用することが可能ですので、メッシュ農業気象データの利用については、Python の利用をお薦めします。必要に応じて、Python 以外のプログラミング言語を使用するのが良いでしょう。

以上、実習 3 では、メッシュ農業気象データを取得する方法について、Python を利用する方法と、おまけとして、その他プログラミング言語で扱う場合のヒントについて、扱いました。

【実習 4】Python によるメッシュ農業気象データの処理 1

農研機構 東北農業研究センター 生産環境研究領域 川方俊和

はじめに

この実習では、一つの Python プログラムを実行しながらプログラミングの方法や注意点を学習します。以下がそのスクリプトです。なにはともあれ、このプログラムを実行してみましょう。

```python
# -*- coding: utf-8 -*-
"""
第218回農林交流センターWS「メッシュ農業気象データ利用講習会」
実習: Pythonによるメッシュ農業気象データの処理1

"""
#Pythonに機能を追加するために外部モジュールをインポートします。
import numpy as np                  #配列計算を高速に実行するためのモジュール
import AMD_Tools3 as AMD            #メッシュ農業気象データを利用するためのモジュール



#ちょっと奇妙ですが、プログラムの本文は次の文で始めます。
if __name__ == "__main__":

    # 取得する気象要素の指定
    #  気象要素は、文字列で指定します。
    nani = "TMP_mea"

    # データを取得する期間の指定
    #  2018年5月1日から9月30日までのデータを取得するには以下のように期間を指定します。
    itsu = ["2018-05-01", "2018-09-30" ]

    # 対象とする地点の指定
    #  北緯36.0566度,東経140.125度の地点を含むメッシュのデータを取得するには場所を以下のよ
    doko = [ 36.0566, 36.0566, 140.125, 140.125]     #つくば(舘野)

    # 気象データの取得
    #  気象データを取得するGetMetData関数の括弧の中に上記の変数を指定し、右辺に置きます。
    #  左辺には関数の戻り値を置きます。順に、気象データ、日付、緯度、経度が格納されます。
    met, tim, lat, lon = AMD.GetMetData(nani, itsu, doko, url='./AMD')  #気象デ
    T = met[:,0,0]  #気象データは常に三次元(日付、緯度、経度)の入れ物に入れられて返されま
    ntim = len(tim)  #何日分の気象データかを数えます。

    #有効積算気温の計算
    Ts = 10.0               #基準温度

    Tcc = np.zeros(ntim)    #有効積算気温の入れ物

    # 第1日目(0番目の配列要素)は特別扱いが必要です。
    Te = T[0] -Ts
    if 0.0 < Te :
        Tcc[0] = Te
    else :
        Tcc[0] = 0.0
    print("---", 0, tim[0], T[0], Tcc[0] )

    # 第2日目(1番目の配列要素)以降は繰り返し計算が可能です。
    for t in range(1, ntim):
        Te = T[t] - Ts
        if 0.0 < Te :
            Tcc[t] = Tcc[t-1] + Te
        else :
            Tcc[t] = Tcc[t-1]
        print("---", t, tim[t], T[t], Tcc[t] )
```

【実習 4】Python によるメッシュ農業気象データの処理1

初めに，Spyder の右上の黒いファルダマークをクリックして，プログラムが格納されている作業フォルダを選択します。ファイルエクスプローラー（右上）からプログラム aet0.py をダブルクリックして，エディタ画面（左）にプログラムが表示されます。または，メニュー，開くから，プログラム aet0.py をクリック，開くから，同画面を表示することもできます。

○実行ボタンを押します。

→気温と有効積算気温が日別に表示されます。

実行コンソール（右下）に，run aet0 と入力，リターンキーを押して下さい。.py は省くことができます。すると，実行コンソールに下のような出力が表示されます。上矢印キー，下矢印キーで過去に入力したコマンドが表示されますので，この履歴機能を使って実行することもできます。エディタ画面にプログラムを表示，選択した状態で，ツールバーの三角印を押して実行することもできます。

```
In [6]: run aet0
TMP_mea (153, 1, 1)
--- 0 2018-05-01 00:00:00 20.807 10.8070354462
--- 1 2018-05-02 00:00:00 19.3069 20.1139450073
--- 2 2018-05-03 00:00:00 21.1068 31.2207431793
--- 3 2018-05-04 00:00:00 15.607 36.8277463913
--- 4 2018-05-05 00:00:00 16.6073 43.4350099564
--- 5 2018-05-06 00:00:00 19.6071 53.0421075821
--- 6 2018-05-07 00:00:00 17.307 60.3490591049
--- 7 2018-05-08 00:00:00 12.807 63.15604496
--- 8 2018-05-09 00:00:00 11.807 64.9630441666
--- 9 2018-05-10 00:00:00 11.1073 66.0703859329
--- 10 2018-05-11 00:00:00 15.0072 71.0776329041
--- 11 2018-05-12 00:00:00 18.0072 79.0848426819
--- 12 2018-05-13 00:00:00 17.0071 86.0919513702
--- 13 2018-05-14 00:00:00 19.1072 95.1991996765
--- 14 2018-05-15 00:00:00 20.9072 106.106435776
```

特定メッシュにおける指定した期間の有効積算気温を計算して表示する

このプログラムは、茨城県つくば市における有効積算気温を、2018 年 5 月 1 日を起点として 9 月 30 日まで計算するものです。それでは、このプログラムがどのようにしてメッシュ農業気象データを取り込み、有効積算気温を計算しているかを、順に解説します。

初めに，メッシュ農業気象データシステムの配信サーバーから気象データを取得するには，GetMetData()を使用します。（）内に，取得する気象要素，取得する期間，取得する緯度，経度の範囲を指定します。

18 行目で，変数 nani に、日平均気温の略記号である文字列”TMP_mea”を代入し，気象要素を指定します。

22 行目で，変数 itsu に、初日(”2018-05-01”)，最終日(”2018-09-30”)を文字列のリストとして代入し，取得するデータの期間を指定します。

26 行目で，変数 doko に、緯度と経度を数値のリストで代入し、取得範囲を指定します。南端の緯度，北端の緯度，西端の経度，東端の経度の順序で指定しま

す。このプログラムは、特定の一つのメッシュにおけるデータを処理するので，南端の緯度と北端の緯度は同じ値，西端の経度と東端の経度は同じ値，[36.0566, 36.0566, 140.125, 140.125]を指定します。

このように値を代入した nani、istu、doko を、31 行目で，GetMetData()の引数として渡します。この実習では，データ配信サーバーではなくて，ローカルの気象データを読み込みます。url='./AMD'でローカルのデータを指定します。ディレクトリ構造は配信サーバーと同一にします。省略した場合は配信サーバーを読みにいきます。これらを受け取った GetMetData 関数は、気象データ(3 次元)、日付(1 次元)、緯度(1 次元)、経度(1 次元)からなる 4 つの ndarray を返します。31 行目では、それらをそれぞれ，met, tim, lat, lon の変数で受け取っています。

今回、GetMetData()は、単一メッシュの気温を 153 日分取得するように命じられているので、3 次元配列 met には、実質 1 次元分のデータしか入っていません。

32 行目は, met の配列要素を取り出して 1 次元配列の形式で T に代入します。全期間のデータを取り出すので，日付の添え字（一番最初）については全要素を意味するコロン ： を指定します。特定メッシュのデータなので，緯度ならびに，経度については幅がありません。このため，緯度の添え字（二番目），経度の添え字（三番目）については，Python の数え方で最初を示す 0 を指定します。この記述のみで，日平均気温（以下，気温）の配列を取り出すことができます。

33 行目では，関数 len()で，tim の要素数を取得し ntim に代入します。。ntim は、後ほど繰り返し計算で使います。なお，特定地点の時系列データの場合，tim の要素数と T の要素数とは一致し，ともに ntim です。

これで、気象データが準備できました。それではいよいよ有効積算気温を計算しましょう。有効積算気温とは，ある基準温度以上の気温を積算した値です。

```
35    #有効積算気温の計算
36    Ts = 10.0              #基準温度
37
38    Tcc = np.zeros(ntim)   #有効積算気温の入れ物
39
40    # 第1日目(0番目の配列要素)は特別扱いが必要です。
41    Te = T[0] - Ts
42    if 0.0 < Te :
43        Tcc[0] = Te
44    else :
45        Tcc[0] = 0.0
46    print("---", 0, tim[0], T[0], Tcc[0] )
47
48    # 第2日目(1番目の配列要素)以降は繰り返し計算が可能です。
49    for t in range(1,ntim):
50        Te = T[t] - Ts
51        if 0.0 < Te :
52            Tcc[t] = Tcc[t-1] + Te
53        else :
54            Tcc[t] = Tcc[t-1]
55        print("---", t, tim[t], T[t], Tcc[t] )
```

36 行目，ここでは，基準温度，Ts = 10.0 に設定します。

38 行目の np.zeros()は，Numpy の配列を生成する機能の一つです。新規配列の要素数を ntim に指定し，全要素を 0 に初期化して，有効積算気温を格納する配列，Tcc を生成します。

41 行目から 45 行目までは，初日（0 番目の配列要素）の有効積算気温を設定します。

41 行目で，0 番目の気温，T[0]を取り出して，基準温度，Ts を減じて，有効温度，Te に代入します。

42 行目から 45 行目では，if:文で条件判断を行い，処理を実施します。条件判断が真（Te が 0 より大）であれば次の処理（Tcc[0]に Te を代入）し，条件判断が偽（Te が 0 以下）であれば，else:以下の処理（Tcc[0]に 0 を代入）を行います。なお，Tcc[0]は，初日の有効気温を意味します。

46 行目では，print()文で，初日の要素番号，日時，気温，有効積算気温を画面表示します。それぞれ，[0]を付けることで，0 番目の配列要素を取り出します。

49 行目から 55 行目までは，for:文で，1 番目の要素番号から，最後の要素番号まで，繰り返します。最後の要素番号は，要素数 ntim から 1 を引いた値になることに注意して下さい。

t 番目の気温，T[t]から基準温度を減じて有効気温，Te を求めます。if:文で，もし，有効気温が 0 より大きければ，t 番目の有効積算気温は，(t-1)番目の有効積算気温に有効気温を加えた値とする，そうでなければ，有効気温は 0 とみなし，t 番目の有効積算気温は，(t-1)番目の有効積算気温と同じとします。

55 行目では，t 番目の，要素番号，日時，気温，有効積算気温を画面表示します。

【練習問題 1】

Python の組み込み max は、複数の引数を取り、それらの中で最も大きいものを戻り値として返します。関数 max を使うと、有効積算気温の計算をもう少し簡単に書くことができます。どのようにしたらよいでしょう。

【解答例 1】

関数 max を，下の図の 41 行目、46 行目のように使用すると有効積算気温が 1 行で計算できます。

```
40    # 第1日目(0番目の配列要素)は特別扱いが必要です。
41    Tcc[0] = max(T[0]-Ts, 0.0)                    #     <--------
42    print("---", 0, tim[0], T[0], Tcc[0] )
43
44    # 第2日目(1番目の配列要素)以降は繰り返し計算が可能です。
45    for t in range(1,ntim):
46        Tcc[t] = Tcc[t-1] + max(T[t]-Ts, 0.0)     #     <--------
47        print("---", t, tim[t], T[t], Tcc[t] )
```

○max()を使った文を記入して，保存し，実行ボタンをクリックします。

→計算が実行されます。

☑同じ計算結果になることを確認しましょう。

【実習 4】Python によるメッシュ農業気象データの処理1

この修正を加えたサンプルプログラムが aet1.py に収めてあります。
指定した積算値に達する日の計算

【練習問題２】

有効積算気温の上限値（950.0℃日）を設定し、それに達する日を計算するにはどのようにすればよいでしょう。

○Spyderのエディタにaet1.pyを開きます。講師の指示に従ってプログラムを修正してください。

【解答例 2】

for:構文のループの中で，有効積算気温を計算するたびにそれが上限値を越えたかどうかを判断し、もし，有効積算値が上限値を越えていた場合には，繰り返し計算を中止するとともに、その繰り返しに相当する日を表示すれば、目的を達することができます。それは以下のようになります。

```
35    #有効積算気温の計算
36    Ts = 10.0              #基準温度
37    Tccx = 950.0           #有効積算気温の上限値                    <--------
38    Tcc = np.zeros(ntim)   #有効積算気温の入れ物
39
40    # 第1日目（0番目の配列要素）は特別扱いが必要です。
41    Tcc[0] = max(T[0]-Ts, 0.0)
42    print("---", 0, tim[0], T[0], Tcc[0] )
43
44    # 第2日目（1番目の配列要素）以降は繰り返し計算が可能です。
45    for t in range(1,ntim):
46        Tcc[t] = Tcc[t-1] + max(T[t]-Ts, 0.0)
47        print("---", t, tim[t], T[t], Tcc[t] )
48        if Tccx <= Tcc[t] :                  #設定値に達したかの判定    <--------
49            break                                #ループ脱出    <--------
50
51    print()                                          #     <--------
52    print(tim[t])                                    #     <--------
```

37 行目で，有効積算気温の上限値，Tccx=950.0 を設定します。

48 行目の，if:構文の条件式に，有効積算気温が上限値を超えたかどうかの判断を記入します。

49 行目に，break 文を追加します。break 文は， for 文のブロック内で，break 文が実行された場合は，for 文を終了し，ループから脱出します

配列 tim には、データ期間の毎日の日付が格納されているので、tim[t]で t 日目の日付を指定することができます。print 文で，この日時を表示します。

この修正を加えたサンプルプログラムが aet2.py に収めてあります。

○修正が終わったら実行ボタンを押します。

→日ごとの気温と有効積算気温が表示され，最後に、950℃日を超えた日が表示されます。

【実習 4】Python によるメッシュ農業気象データの処理1

有効積算気温を利用した発育の予測

植物の発育の進行は、気温や日長、土壌状態等に影響を受けますが、圃場や作付け時期などがあまり変化しなければ、有効積算気温で精度よく推定できることが知られています。有効積算気温で作物などの発育を予測するには、播種や出芽をした日を起点とし、その翌日からの有効温度を積算します。

プログラム aet2.py は、ほんの少し手を加えるだけで、発育予測のプログラムにすることができます。

【練習問題 3 】

出芽翌日からの有効積算気温が 950℃日で出穂を迎える水稲品種があったとします。2018 年 5 月 1 日に出芽したこの水稲品種は，何月何日に出穂すると予測できるでしょう。プログラム aet2.py を修正して予測出穂日を表示するプログラムを作成して下さい。なお，有効気温の基準温度は 10℃とします。

○Spyderのエディタにaet2.pyを開きます。講師の指示に従ってプログラムを修正してください。

【解答例 3】

```
41    Tcc[0] = 0.0                                    #     <--------
```

計算初日の有効積算気温を 0 とおけばそれで完成です。

41 行目の，第 1 日目（0 番目の配列要素）の有効積算気温を 0 に変更します。初日の気温を計算に含めないことで，発育モデルの構造で計算することができます。

○修正が終わったら実行ボタンを押します。

→初日の有効積算気温が0で表示され，その後気温と有効積算気温の日別表示，950℃日を超えた日が表示されます。

この修正を加えたサンプルプログラムが aet3.py に収めてあります。

【実習 4】Python によるメッシュ農業気象データの処理1

有効積算気温を利用した定植日の逆算

有効積算気温を計算するプログラムを工夫すると、指定した日に有効積算気温が指定した値に達するような起算日を求めることもできます。このような計算は、たとえば、目標とした日に作物を収穫するためには，その作物をいつ定植すればよいかといった課題や，ある害虫の成虫を観察したときに，それはいつ頃孵化していたのかを推定するといった課題などに用いることができます。

【練習問題４】

定植翌日からの有効積算気温が 950℃で収穫適期を迎える農作物があるとします。ある圃場において，この作物が 2018 年 7 月 31 日に収穫適期を迎えたとすると，その作物はいつ定植されたと推定できるでしょう。プログラム aet3.py を修正して，定植日を表示するプログラムを作成してください。なお，有効気温の基準温度は 10℃とします。

○aet3.py を Spyder のエディタに開いてください。講師の解説と指示に従って、修正を加えてください。

```
20  # データを取得する期間の指定
21  #  2018年5月1日から7月31日までのデータを取得するには以下のように期間を指定します。
22  itsu = ["2018-03-01", "2018-07-31" ]                    #     <--------
23
24  # 対象とする地点の指定
25  #  北緯36.0566度,東経140.125度の地点を含むメッシュのデータを取得するには場所を以下のよう
26  doko = [ 36.0566, 36.0566, 140.125, 140.125]    #つくば(舘野)
27
28  # 気象データの取得
29  #   気象データを取得するGetMetData関数の括弧の中に上記の変数を指定し、右辺に置きます。
30  #   左辺には関数の戻り値を置きます。順に、気象データ、日付、緯度、経度が格納されます。
31  met, tim, lat, lon = AMD.GetMetData(nani, itsu, doko, url='./AMD')  #気象デ
32  T = met[:,0,0]  #気象データは常に三次元(日付、緯度、経度)の入れ物に入れられて返されま
33  ntim = len(tim)  #何日分の気象データかを数えます。
34
35  #有効積算気温の計算
36  Ts = 10.0               #基準温度
37  Tccx = 950.0            #有効積算気温の上限値
38  Tcc = np.zeros(ntim)    #有効積算気温の入れ物
39
40  # 最終日((ntim-1)番目の配列要素)は特別扱いが必要です。
41  Tcc[ntim-1]=Tccx                                        #     <--------
42  print("---", ntim-1, tim[ntim-1], T[ntim-1], Tcc[ntim-1] ) #     <--------
43
44  # (最終日-1日)((ntim-2)番目の配列要素)以降は繰り返し計算が可能です。
45  for t in range(ntim-1, 0, -1):          #ループを91,90,89、・・・と回す。 <--------
46      Tcc[t-1] = Tcc[t] - max(T[t]-Ts, 0.0)               #     <--------
47      print("---", t-1, tim[t-1], T[t-1], Tcc[t-1] )      #     <--------
48      if Tcc[t-1] < 0.0 :                                 #     <--------
49          break
50
51  print()
52  print("定植日は"+tim[t-1].strftime("%Y/%m/%d")+"と推定されます。")  # <--------
```

【実習 4】Python によるメッシュ農業気象データの処理1

【解答例 4】

22 行目で，取得する期間の指定を変更します。最終日は”2018-07-31”，初日は，余裕をもって”2018-03-01“と設定します。

41 行目で，最終日の有効積算気温を初期値として設定します。要素番号は，0 から始まり，要素数は ntim，従って，最終日の要素番号は，(ntim-1)になります。Tcc[ntim-1]=Tccx と設定します。

42 行目で，最終日の要素番号，日時，気温，有効積算気温を画面表示します。

45 行目から，for:文で，最終日の要素番号から，計算を繰り返します。

46 行目で，当日の有効積算気温から当日の有効気温を減じて,前日の有効積算気温を求めます。

47 行目で，前日の要素番号，日時，気温，有効積算気温を画面表示します。

48 行目，49 行目で，前日の有効積算気温が 0 よりも小さければ，ループを脱出して，計算を終了します。

52 行目で，この有効積算気温が 0 より小さくなった日時を表示します。ここでは、多少手を加えて、「定植日は 2018/05/01 と推定されます。」のように表示させています。

```
52    print("定植日は"+tim[t-1].strftime("%Y/%m/%d")+"と推定されます。")  # <--------
```

○修正が終わったら実行ボタンを押して実行し結果を確認してください。

GetMetData 関数が tim に返す配列には、日時オブジェクト(datetime オブジェクトと呼ばれる特別なデータ型で日付が格納されています。これを文字列型に変換するには，datetime オブジェクトのために用意されている strftime()メソッドを使います。引数の文字列は，日付の表示書式を指定するもので，%Y，%m，%d の場所にそれぞれ，年（西暦四桁），月，日の数字が埋め込まれます，その後に，「/」や「年」，「-」，「.」などの適宜書きます。

この修正を加えたサンプルプログラムが aet4.py に収めてあります。

【実習 4】Python によるメッシュ農業気象データの処理1

Numpy を活用した有効積算気温の計算例

これまで，for ループの中で，１変数の足し算を繰り返して，有効積算気温を求めました。Numpy は，多次元配列の処理に特化したライブラリです。この機能を活用すれば，多次元配列のプログラムを簡潔に記述できます。

【ステップアップ１】

Numpy の機能を活用して，有効積算気温を利用した発育の予測のプログラム，aet3.py を修正してみましょう。

【解答例 5】

```
35    #有効積算気温の計算
36    ts = 10.0              #基準温度
37    tccx = 950.0           #有効積算気温の上限値                    -
38
39    Tc = T - ts                                                #     <--------
40    Tc0 = np.maximum(Tc, 0)                                    #     <--------
41    Tc0[0] = 0                                                 #     <--------
42    Tcc = np.cumsum(Tc0, axis=0)                               #     <--------
43
44    for t in range(ntim):
45        print("---", t, tim[t], T[t],Tcc[t])
46        if tccx <= Tcc[t] :
47            break
48    print()
49    print(tim[t])
50
```

39 行目，T は，既に，ndarray 型の 1 次元配列です。ts は，10.0 の値です。T から ts を引くと，ts は，T と同じ ndarray 型とみなして（ブロードキャスティング），一度に，配列同士の引き算ができます。

40 行目，np.maximum(Tc, 0)の括弧の中は，39 行目と同じく，0 の値を，Tc と同じ ndarray 型とみなします。np.maximum()は，最大値を計算する関数で，配列の全要素に対して，要素毎に，この関数が作用します（ユニバーサル関数）。一度に，０℃以上の有効気温の配列を計算できます。

41 行目，初日の有効温度を 0℃に設定することで，42 行目の初日の有効積算気温を 0℃にします。

42 行目，np.cumsum()は，多次元配列の特定の軸方向の累積和を計算する関数です。axis=0 で，Python の最初の数である第 0 軸方向を指定し，その方向の累積和を計算します。一度で，有効積算気温の配列を計算できます。

for ループを使用せずに，気温の配列から有効積算気温の配列が計算できます。

44 行目から 49 行目，結果を表示します。

この修正を加えたサンプルプログラムが aet5.py に収めてあります。

【実習 4】Python によるメッシュ農業気象データの処理1

有効積算気温を関数にまとめる

関数にまとめておくことで，何度も同じ処理を記述しなくて済みます。

【ステップアップ２】

有効積算気温の計算例（aet5.py）を関数にまとめてみましょう。

【解答例６】

```
12 def Aet(T,ts,tcc0=False):                                    #      <--------
13     """
14     aet:有効積算気温を計算する関数
15     説明文、引数、戻り値、使用例などの記入する。
16     """
17     Tc = T - ts
18     Tc0 = np.maximum(Tc, 0)
19     if tcc0: Tc0[0] = 0                                      #      <--------
20     Tcc = np.cumsum(Tc0, axis=0)
21     return Tcc                                               #      <--------
```

関数 def()を定義します。def の後に，関数名と引数を指定します。関数名は Aet とします。引数は，気温の配列 T，基準温度 ts，初日の有効温度を処理するパラメータ（デフォルトは False）とします。戻り値は，有効積算気温の配列 Tcc として，return Tcc で与えます。

aet5.py の 39 行目から，42 行目までを def()の中に入れます。

19 行目，初日の有効温度を処理するパラメータが，True の場合，初日の有効温度 Tc[0]を 0 とし，False の場合，初日から有効積算気温を計算します。False 以外にも，None，0，空の文字列の場合は，False の判断になります。False の判断以外は，True になります。

```
49     #有効積算気温の計算
50     #初日から有効積算気温を計算する場合は、第3引数を省略する、または、False、None、0、空の文
51     #初日の有効積算気温を0にする場合は、第3引数に、True、0以外の数値、文字列を与える。
52     Tcc = Aet(T, ts, 1)                                      #      <--------
53
54     #有効積算気温が950℃日を超える日を表示
55     for t in range(0, ntim):
56         print("---", t, tim[t], T[t], Tcc[t] )
57         if tccx <= Tcc[t]:
58             break                                  #ループ脱出
59     print()
60     print(tim[t])
```

52 行目，関数 Aet()の引数に，気温の配列 T，基準温度 ts，初日の有効温度を処理するパラメータを与えて，有効積算気温の配列 Tcc を計算します。

54 行目から 60 行目，結果を表示します。aet5.py と同じことを確認して下さい。

この修正を加えたサンプルプログラムが aet6.py に収めてあります。

【実習 4】Python によるメッシュ農業気象データの処理1

別ファイルから、有効積算気温の関数をインポートする

Python のモジュールとは，関数などが含まれるファイルを指します。ファイル aet6.py は，有効積算気温の関数 Aet()を含むので，りっぱなモジュールです。プログラムの規模が大きくなるにつれて，モジュールを作成して，別のファイルからインポートして使うことが多くなりますので，この方法を学びましょう。

【ステップアップ３】

別ファイルから，有効積算気温の関数を含む aet6.py をモジュールとしてインポートして動かしましょう。

【解答例７】

```
11 import aet6                                          #      <--------
```

11 行目，モジュール aet6 をインポートします。

```
42     Tcc = aet6.Aet(T, ts, 1)                         #      <--------
43
44     #有効積算気温が950℃日を超える日を表示
45     for t in range(0, ntim):
46         print("---", t, tim[t], T[t], Tcc[t] )
47         if tccx <= Tcc[t]:
48             break                                    #ループ脱出
49     print()
50     print(tim[t])
```

42 行目，モジュール aet6 の関数 Aet()を参照します。引数に，気温の配列 T，基準温度 ts，初日の有効温度を処理するパラメータを与えて，有効積算気温の配列 Tcc を計算します。

45 行目から 60 行目，結果を表示します。aet6.py と同じことを確認して下さい。

なお，aet6.py の 24 行目，if__name__==”__main__”: の部分は，「このファイルがメイン実行ファイルとして呼び出された場合に実行せよ」という意味の条件分岐文です。今回は，別ファイルから呼び出されるので，この 24 行目以下は，無視されて，24 行目未満のモジュールの役割を果たします。

この修正を加えたサンプルプログラムが aet7.py に収めてあります。

参考文献

科学技術計算のための Python 入門　開発基礎，必須ライブラリ，高速化，
中久喜健司，技術評論社，2016.
入門 Python3, Bill Lubanovic （著），斎藤康毅（監訳），長尾貴弘（訳），
オライリー・ジャパン，2015.

【実習 5】Python によるメッシュ農業気象データの処理 2

農研機構 農業情報研究センター 大久保さゆり

はじめに

この講義では、前項の内容を応用し、グラフ表示、地図表示、テキストへの出力方法を紹介します。

実習で使用するファイル

Aet8.py

・単一メッシュの時系列データ（有効積算気温）を折れ線グラフとして描画する。
・単一メッシュの時系列データ（　　〃　　）をテキストデータとして出力する。

Aet9.py

単一地点でなく一定の範囲について計算する。
・計算結果を 2 次元のメッシュ図として出力し、地理院地図に重ねて表示する。
・計算結果を 3 次元（2 次元＋時系列）でテキストデータとして出力する。

単一地点のデータ出力

1）単一地点の時系列を折れ線グラフとして描画

Spyder 上で aet8.py を開きます。つくば市舘野の有効積算気温を計算させ、その折れ線グラフとテキストデータを出力するプログラムです。開いたら、Spyder の実行ボタンをクリックして実行してみましょう。

○aet8.py を Spyder のエディタに開き、実行ボタンをクリックして実行します。

→　有効積算気温の推移を表すグラフ（pngファイル）、およびそのテキストデータのファイル（csvファイル）が出力されます。

```
# -*- coding: utf-8 -*-
"""
第211回農林交流センターWS「メッシュ農業気象データ利用講習会」
実習：Pythonによるメッシュ農業気象データの処理2

1)「単一地点の」有効積算気温を計算し（前パートと同一）
2)有効積算気温の推移をグラフで描画し
3)テキストをcsvファイルとして出力する。

"""
#Pythonに機能を追加するために外部モジュールをインポートします。
import numpy as np                #配列計算を高速に実行するためのモジュール
import AMD_Tools3 as AMD          #メッシュ農業気象データを利用するためのモジュール
import matplotlib.pyplot as plt   # python付属の、グラフ等を描画するためのモジュール
```

図 1　aet8.py（その 1、外部モジュール呼び出し設定など）

【実習 5】Python によるメッシュ農業気象データの処理2

aet8.py の冒頭には import 文が 3 つあります（図 1 ）。numpy, AMD_Tools3 に加えて、グラフを描画するために「matplotlib.pyplot」というモジュールを呼び出しています。

```
20    # 取得する気象要素の指定
21    #   気象要素は、文字列で指定します。
22    nani = "TMP_mea"
23
24    # データを取得する期間の指定
25    #   2018年3月1日から7月31日までのデータを取得するには以下のように期間を指定します。
26    itsu = ["2018-03-01", "2018-07-31" ]
27
28    # 対象とする地点の指定
29    doko = [ 36.0566, 36.0566, 140.125, 140.125]   #つくば(舘野)
30
31    # 気象データの取得
32    #     気象データを取得するGetMetData関数の括弧の中に上記の変数を指定し、右辺に置きます。
33    #     左辺には関数の戻り値を置きます。順に、気象データ、日付、緯度、経度が格納されます。
34    met, tim, lat, lon = AMD.GetMetData(nani, itsu, doko)   #気象データを取得
35    #気象データは常に三次元(日付、緯度、経度)の入れ物に入れられて返されます。それを1次元(日付)の入れ物に
36    T=met
37
38    ntim = len(tim)  #何日分の気象データかを数えます。
39    nlat = len(lat)  #南北方向(緯度方向)のメッシュ数を数えます。
40    nlon = len(lon)  #東西方向(経度方向)のメッシュ数を数えます。
```

図 2　aet8.py（その 2 、期間、領域、要素の指定パート）

気象要素の指定（変数 nani）、期間の指定（変数 itsu）、経緯度の指定（変数 doko）は前項と同様です（図 2 ）。今回も舘野を含む 1 メッシュを指定するので、「doko」には「緯度，緯度，経度，経度」と、 2 回ずつ入力します。

```
42    #------1)有効積算気温の計算 ---------------#
43    Ts = 10.0        #基準温度
44    Tccx = 950.0     #有効積算気温の上限値
45    Tcc = np.zeros(ntim)    #有効積算気温の入れ物
46
47    #-  t が 0(1番目)から ntim(選択された日数)になるまで、基準温度以上を積算していきます --#
48
49    for t in range(0,ntim):
50        # まず最初の日(t=0)を計算する
51        if t == 0:
52            Tcc[t] = max(T[t]-Ts, 0.0)
53        # 2日目以降は、「t-1日目までの積算温度」に「t日目の有効温度」を足していく
54        else:
55            Tcc[t] = Tcc[t-1] + max(T[t]-Ts, 0.0)
56        # 上限値に達した場合は、そこで計算をストップ
57        if Tccx <= Tcc[t] :
58         break
```

図 3　aet8.py（その 3 、有効積算気温の計算パート）

【実習 5】Python によるメッシュ農業気象データの処理2

有効積算気温の計算は、上限値（変数 Tccx）として指定した 950℃（44 行目を参照）に達した日でストップするように指定してあります（図 3、56-58 行目）。この計算が終わると、変数「Tcc」には、指定された期間中の有効積算気温（選択したメッシュの時系列データ）が入ります。

続いて、計算された有効積算気温（Tcc）を折れ線グラフに描画します（図 4）。

```
60    #-------2)計算結果のグラフ表示 -----#
61    # 描画領域の作成
62    fig = plt.figure(num=None, figsize=(10,4))
63
64    # x軸、y軸ラベルの指定
65    plt.xlabel('Date')  # x軸
66    plt.ylabel('Effective accumulated temp. [degC]') # y軸
67    plt.title('N'+str(lat)+', E'+str(lon)+
68              ' Effective accumulated temperature') # タイトル
69
70    # 折れ線グラフを描画。
71    # x軸に日付(tim)、y軸に有効積算気温(Tcc)をとり、線の色は赤、線幅は1.5
72    plt.plot(tim,Tcc,'red',linewidth=1.5)
73
74    # 図として保存
75    plt.savefig('chart.png', dpi=150)
```

図 4　aet8.py（その 4、グラフの描画パート）

→「plt」は、matplotlib.pyplot という外部モジュールを表します。aet8.py の冒頭（図 1）の指定を確認しましょう。

62 行目　fig = plt.figure(num=None, figsize=(10,4))

は、図の描画サイズを指定しています。モジュール「matplotlib.pyplot」の figure 関数を使用し、figsize=(10,4) の部分で「図の x 軸（横軸）は長さ 10, y 軸（縦軸）は長さ 4」を指定しています。単位はインチです。figsize を何も指定しない場合は（8,6）で横 8 インチ×縦 6 インチになります。

続く 65-68 行目は、図の x 軸ラベル、y 軸ラベル、および図のタイトルを指定しています。

```
plt.xlabel('Date')
plt.ylabel('Effective accumulated temp. [degC]')
plt.title('N'+str(lat)+', E'+str(lon)+
          ' Effective accumulated temperature')
```

→aet8.py でのさまざまな指定と、chart.png での表示の対応を見てみましょう。

で、x 軸のラベルには「Date」、y 軸のラベルには「Effective accumulated temp. [degC]」、図のタイトルには「N○°、E○°　Effective accumulated temperature」と表示されるように指定しています。

【実習 5】Python によるメッシュ農業気象データの処理2

67-68 行目 plt.title の「str(lat)」、「str(lon)」の箇所は、変数 lat（緯度）、変数 lon（経度）に入っている数値を文字列として表示する、という指定の方法です。

72 行目　plt.plot(tim,Tcc,'red',linewidth=1.5)

で、グラフを描画します。「x 軸に日付（tim）、y 軸に有効積算気温（Tcc）をとり、線の色は赤、線幅は 1.5」と指定しています。

75 行目　plt.savefig('chart.png', dpi=150)

描画したグラフを図として保存します。ファイル名と解像度（150dpi）を指定しています。

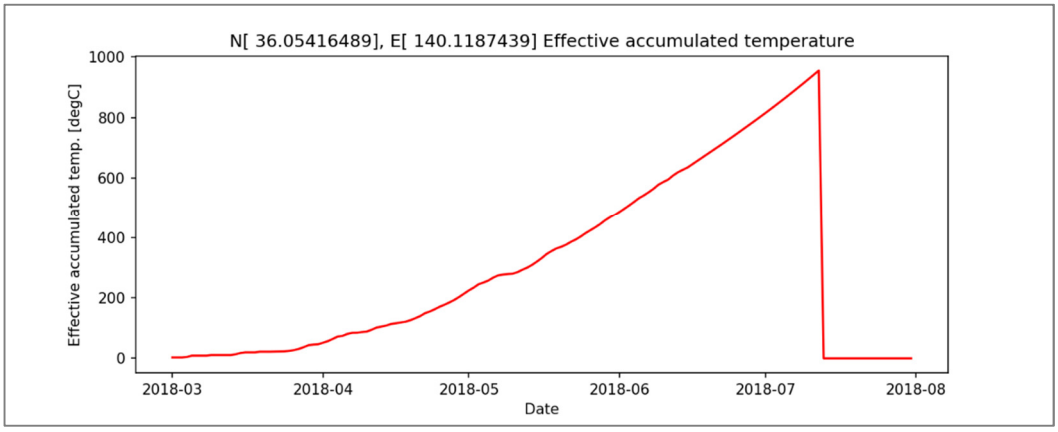


図 5　chart.png

aet8.py 上の有効積算気温 Tcc は「950 度に達したら計算を打ち切る」として計算されているため、「950℃に達した日」以降の値は「0」が入っています。

【応用】

950℃に達しても、選択した日付まで積算し続けるように編集し、グラフに表示してみましょう。

※ヒント：aet9.py を覗いてみましょう。

なお matplotlib.pyplot を使った図の描画方法は、メッシュ農業気象に限らず広く使われており、web 上にも多くの情報が出ています。作図にこだわりたい方はいろいろ調べてカスタマイズしてみましょう。

2）単一地点の時系列をテキストデータとして出力

今度は、描画に用いた Tcc を、テキストに書き出します。

```
77    #------3) 結果をcsvで書き出す-----------#
78    # AMD.tools3 の PutCSV_TS関数を使う
79    AMD.PutCSV_TS(Tcc,tim, header='Date,EAT', filename='EAT-Tateno.csv')
80
```

図 6　aet8.py（その 5）

単一地点の時系列データをテキストに出力するには、AMD_Tools3.py の PutCSV_TS 関数を使用します。Aet8.py 内では 79 行目です。この関数の 1 つめの引数には出力したい値（ここでは Tcc）を、2 つ目の引数には日付の変数（ここでは tim）を指定します。ほかに、header として csv ファイルのヘッダー行を、filename=''では出力するファイル名を指定できます。

細かなオプションは「AMD_Tools3」内の該当箇所に解説があります。aet8.py 内にもコメントとして記載しました。

グラフを描くだけでいい、テキストデータは不要（あるいは、テキストデータだけ必要でグラフは不要）という場合は、 aet8.py 内の必要な部分をコメントアウト（または削除）して実行すると、不要なファイルができません。

2 次元（面）・3 次元（面＋時系列）でのメッシュ農業気象データの処理

1） 2 次元・時系列でのデータ処理

この章では、メッシュ農業気象データの空間データとしての取り扱いを解説します。基本的には 1 地点の時系列の場合とそれほど変わりません。aet8.py を 1 地点から面に拡張した aet9.py を例に紹介します。

aet9.py を Spyder 上で開いて実行すると、html ファイル、複数の png ファイル、 2 つの csv ファイルが出力されます。出力された html ファイルを開くと、一定範囲のメッシュが、国土地理院の地図上に表示されます。

aet9.py の冒頭部分の抜粋を図 7 に示します。aet8 とは

29 行目　doko = [35.8, 36.3, 139.7, 141]

の部分が変わりました。変数 doko には、緯度・経度それぞれ異なる値が指定されています。メッシュ農業気象データで東西南北の 2 次元の範囲を指定するには、[南端, 北端, 西端, 東端]とします。aet9.py では、先ほどまで使っていた舘野を含む周辺の領域を指定しています。

【実習 5】Python によるメッシュ農業気象データの処理2

```
14 #---- 0)気象データを取得----------------
15
16 #ちょっと奇妙ですが、プログラムの本文は次の文で始めます。
17 if __name__ == "__main__":
18
19     # 取得する気象要素の指定
20     #   気象要素は、文字列で指定します。
21     nani = "TMP_mea"
22
23     # データを取得する期間の指定
24     #   2018年3月1日から7月31日までのデータを取得するには以下のように期間を指定します。
25     itsu = ["2018-03-01", "2018-07-31" ]
26
27     # 対象とする地点の指定
28     # 北緯36.0566度,東経140.125度の地点を含むメッシュのデータを取得するには場所を以下のように指定しま
29     doko = [ 35.8, 36.3, 139.7, 141] #つくば周辺の矩形領域
30
31     # 気象データの取得
32     #     気象データを取得するGetMetData関数の括弧の中に上記の変数を指定し、右辺に置きます。
33     #     左辺には関数の戻り値を置きます。順に、気象データ、日付、緯度、経度が格納されます。
34     met, tim, lat, lon = AMD.GetMetData(nani, itsu, doko)   #気象データを取得
35     #この場合のmetは本当に三次元なので、入れ物は替えません。
36     T=met
37
38     ntim = len(tim)   #何日分の気象データかを数えます。
39     nlat = len(lat)   #南北方向(緯度方向)のメッシュ数を数えます。
40     nlon = len(lon)   #東西方向(経度方向)のメッシュ数を数えます。
41
```

図 7　aet9.py（その 1、取得する範囲の指定、データ数の定義）

続いて、指定した範囲の全てのメッシュについて、有効積算気温の計算を行います（図 8）。単一メッシュに対して計算を行った aet8.py よりも少し複雑なループになっています。

48 行目から 51 行目は、「変数 y が nlat（緯度方向のメッシュ数）になるまで以下を行う」「変数 x が nlon（経度方向のメッシュ数）になるまで以下を行う」「変数 t が ntim（対象にした日数）になるまで以下を行う」という繰り返しの指定です。

52-54 行目は、「もしそのメッシュが無効値 nan（水域などを含む場合）であれば、そこは除外して次のメッシュに進む」という指定です。

55 行目から 58 行目は、それぞれのメッシュに対して t=0 から t=ntim になるまでの有効積算気温の計算です。単一メッシュの場合（aet8.py）に y, x の添字が加わった配列での表記になっていますが、計算過程は同一です。

なお、aet9.py では、上限値に達したら計算終了ではなく、全てのメッシュについて、期間の最後まで積算を続けるようになっています。

→aet5.pyの「上限値に達したら計算終了」　の指定と、aet6.pyでの「期間の最後まで積算」部分の記載を比べてみましょう。

【実習 5】Python によるメッシュ農業気象データの処理2

```
31    # 気象データの取得
32    #    気象データを取得するGetMetData関数の括弧の中に上記の変数を指定し、右辺に置きます。
33    #    左辺には関数の戻り値を置きます。順に、気象データ、日付、緯度、経度が格納されます。
34    met, tim, lat, lon = AMD.GetMetData(nani, itsu, doko)    #気象データを取得
35    #この場合のmetは本当に三次元なので、一次元への変換は行いません。
36    T=met
37
38    ntim = len(tim)  #何日分の気象データかを数えます。
39    nlat = len(lat)  #南北方向(緯度方向)のメッシュ数を数えます。
40    nlon = len(lon)  #東西方向(経度方向)のメッシュ数を数えます。
41
42 #------1)有効積算気温の計算 ---------------#
43    Ts = 10.0        #基準温度
44    Tccx = 950.0     #有効積算気温の上限値
45    Tcc = np.zeros((ntim,nlat,nlon))    #有効積算気温の入れ物(三次元)
46
47    #ここから各メッシュごとに計算
48    for y in range(nlat):
49        for x in range(nlon):
50            Tyx = T[:,y,x]                #メッシュ[y,x]における気温の時系列データを取り出す
51            for t in range(ntim):
52                if np.isnan(Tyx[t]):      #メッシュの値が"nan"(無効値)の場合は海や湖。
53                    Tcc[:,y,x] = np.nan   #すべての日の積算気温の入れ物に無効値を入れて、
54                    break  #このメッシュについては日ごとの計算は放棄する。
55                if t == 0:          # まず最初の日(t=0)を計算する。
56                    Tcc[t,y,x] = max(Tyx[t]-Ts, 0.0)
57                else:               #2日目以降は、「t-1日目までの積算温度」に「t日目の有効温度」を足す。
58                    Tcc[t,y,x] = Tcc[t-1,y,x] + max(Tyx[t]-Ts, 0.0)
59
60    #ここまでで全メッシュの計算が終了
61    Tccend = Tcc[ntim-1,:,:]    #最終日の有効積算気温分布(二次元) ☆今度は上限値でのストップはありません
```

図 8　aet9.py（その 2、データ数の定義と有効積算気温の計算）

ここまでで、変数 Tccend に、指定した領域の最終日の有効積算気温が収まりました。

2) 「地理院地図」に重ねて表示させる ー　PutGSI_map 関数

64 行目は、少し長いです。この 1 行で、result.html と、2 つの png ファイル（result_o.png と result_l.png）が出力されます。

→変数 lat, lon はどこで指定されたでしょうか。34行目を参照してみましょう。

```
AMD.PutGSI_Map(Tccend,lat,lon,
                label="Effective accumulated temperature[degC]",
                cmapstr=None,minmax=None)
```

ここで使われている「AMD.PutGSI_Map」は、計算された値を国土地理院の地図に重ねてブラウザで表示するための、AMD_Tools3 に収録されている関数です。

※ PutGSI_Map の「GSI」は、地図の参照元である国土地理院の略称です。「GIS」でないのでスペルミスに注意しましょ

関数 PutGSI_Map の必須の引数は、「2 次元分布配列」、「配列要素を配置すべき緯度の配列」、「配列要素を配置すべき経度の配列」です。64 行目におい

ては、2 次元分布配列に、積算期間最終日における有効積算気温(Tccend)、各メッシュの緯度値の配列(lat)、各メッシュの経度値の配列(lon)を指定しています。lat と lon は、GetMetData 関数の戻り値として得られたものです。

なお、関数 PutGSI_Map には、以下のオプションをキーワード引数で指定することができます。

label: 凡例のカラーバーに表示されるラベルです。

cmapstr: カラーバーの種類を指定します。以下の matplotlib のページの例に応じて名称で指定します。大小を反転して使いたいときは名称に _r をつけます。none にすると、緑から赤に変化するパターンが指定されます。

minmax: minmax=[最小値,最大値]として、カラーバーの範囲を指定します。単に none とすると、データに応じて決まります。

カラーバーの配色とその名称については下記を参照してください。
http://matplotlib.org/examples/color/colormaps_reference.html

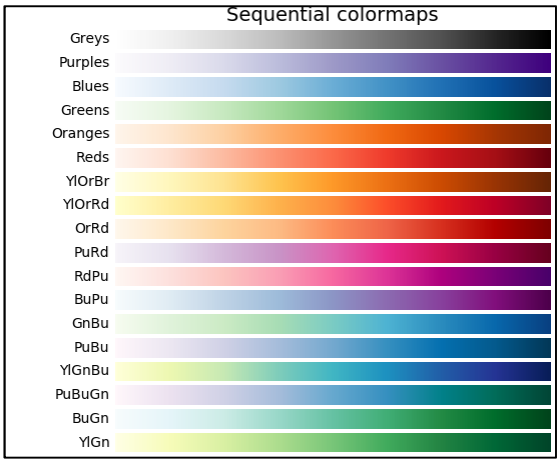


図 9 matplotlib の配色見本と名称（上記 URL より参照）

3）2 次元、2 次元×時系列データのテキスト出力 － PutCSV_MT 関数

最後に、（空間方向の）2 次元データ、あるいはそれに時間も加えた 3 次元のデータをテキストデータとして出力する関数です。PutCSV_MT という関数を使います。この関数も AMD_Tools3 に含まれているものです。

```
86    # AMD_Tools3 の中の PutCSV_MT 関数を使用する。
87    # 3-1)「最終日の有効積算気温」を書き出す---値が「2次元・非時系列」の場合
88    AMD.PutCSV_MT(Tccend,lat,lon,
89                  addlalo=True,header=None,filename='EAT-map.csv')
90    # 3-2)「期間中の日平均気温」を書き出す---値が「2次元・時系列」の場合
91    AMD.PutCSV_MT(met,lat,lon,
92                  addlalo=True,header=None,filename='Temp-map_TS.csv')
93
```

図 10 aet9.py（その 3）

【実習 5】Python によるメッシュ農業気象データの処理2

関数 PutCSV_MT は、２次元または３次元のデータを、３次メッシュコードを行頭として表形式で出力します（図 10）。この関数に必須の指定は、先頭にある３つの「表示する変数」、「緯度の表示範囲」「経度の表示範囲」です。88 行目では変数 Tccend を指定しています。Tccend には「指定した領域の最終日の有効積算気温」、つまり緯度×経度に、時間は１時点のみの、２次元配列のデータが入っています。一方で、91 行目で指定したのは、変数 met で、「指定した領域の指定した期間分の気温」が入っています。こちらは緯度×経度×日数の３次元配列のデータです。

→それぞれのファイルで指定した変数 Tccend, あるいは met が、どんな次元をもつデータであるか、aet9.py 上で確認してみましょう。

その他のオプションには以下があります。

addlalo: True にすると、３次メッシュに加えて中心点の緯度経度も出力します。

header: 先頭行のヘッダーを指定します。aet9.py では None（ヘッダなし）を指定しています。

filename: 出力するファイル名を指定します。

ここで、出力された EAT-map.csv、Temp-map_TS.csv を開いてみましょう。EAT-map.csv では「３次メッシュコード／緯度／経度／値（最終日の有効積算気温）」が並びます。Temp-map_TS.csv は、「３次メッシュコード／緯度／経度」に加えて、「指定した期間分の値（日平均気温）」のカラム（列）が続きます。このように、PutCSV_MT を使うと、指定した領域の一時点のデータ、あるいは指定した期間の時系列データを、テーブル形式で書き出すことができます。aet8.py で扱った一地点用の関数「PutCSV_TS」と合わせて、変数の次元に応じたテキスト出力を行えます。

→ EAT-map.csv, Temp-map_TS.csv をそれぞれ開いてみましょう。

メッシュ農業気象データは、Spyder 環境と Python のプログラミングで様々な処理が可能ですが、例えば他のソフトで統計解析や作図に使用したい場合などは、これらの関数でテキストデータに出力することで、任意のソフトに読み込んで使うことも可能です。