第 211 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

平成 30 年 6 月 28 日(木)～6 月 29 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）

食と農の科学館 「オリエンテーションルーム」 （つくば市観音台 3-1-1）

目　次

メッシュ農業気象データとその活用

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。

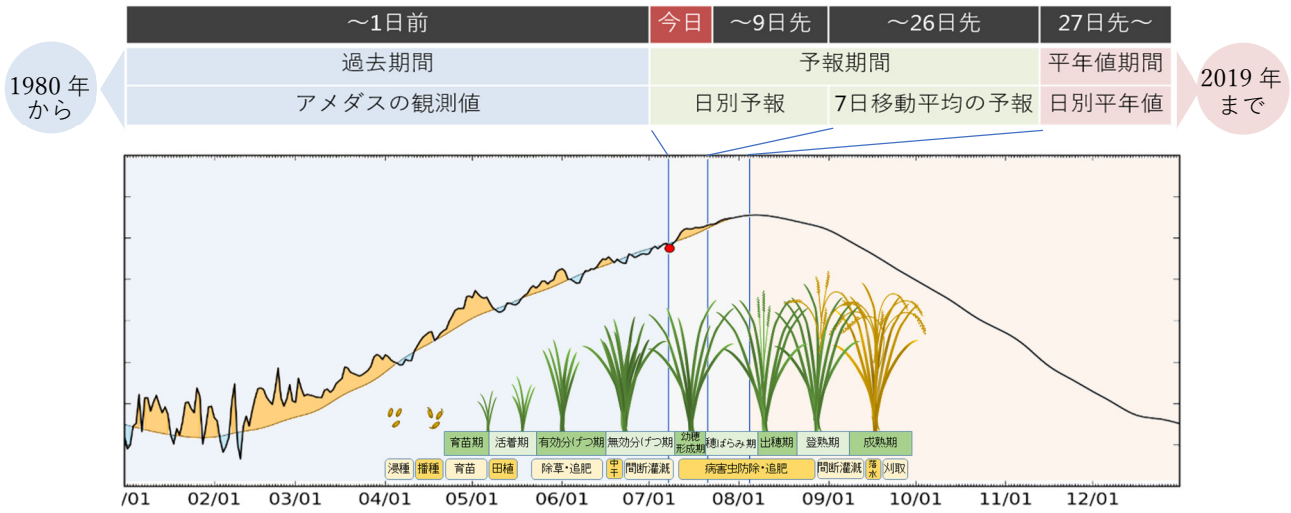


図 1. 2017 年 7 月 8 日にメッシュ農業気象データシステムから取得した、茨城県内のあるメッシュにおけるこの年の日平均気温データのグラフ。この地域で標準的な水稲の生育をイラストで示す。

【講義 1】メッシュ農業気象データとその活用

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

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。2018 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。

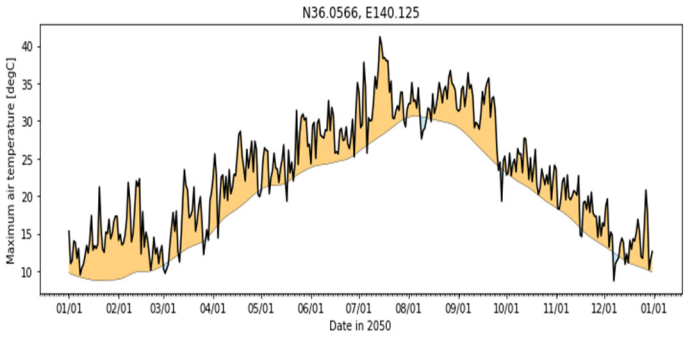


図 2．茨城県つくば市の 2050 年における日最高気温(黒太線)と現在の平年値(黒細線)

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、39 年約 1500 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

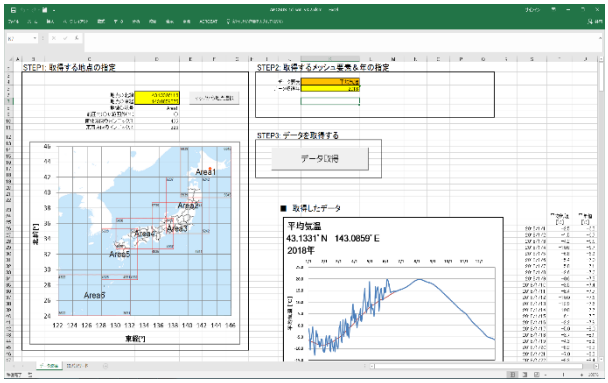


図 3． データ配信サーバーから 1 年分の気象データを取得するマイクロソフトエクセルファイル。ボタンをクリックするだけで最新データが取得できる。

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。

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。

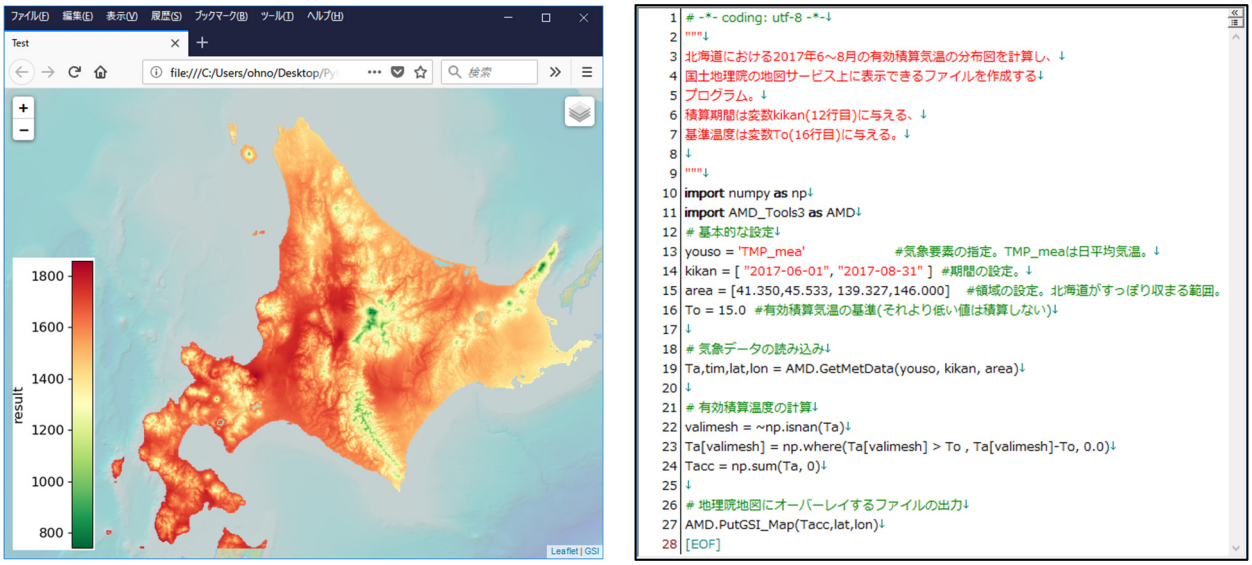


図 4 .　左：北海道における有効積算気温分布図。国土地理院地図上に表示できる。右：この分布図を計算する Python プログラム。

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、

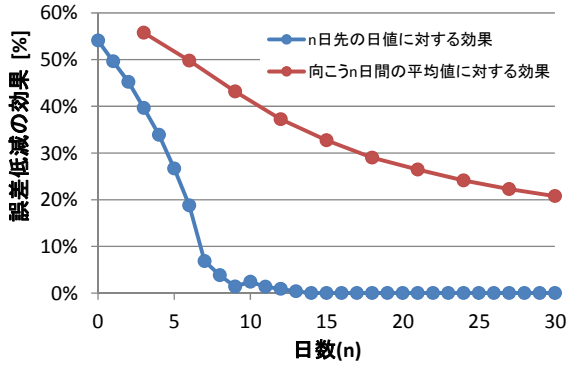


図 5 .　平年値をもって予測値とする場合に対する誤差低減の効果((EC-EF)/EC)と予報日数との関係。ただし、EF は予測値誤差（RMSE)、EC は気候値予測の誤差(2011～2015 年)。

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。

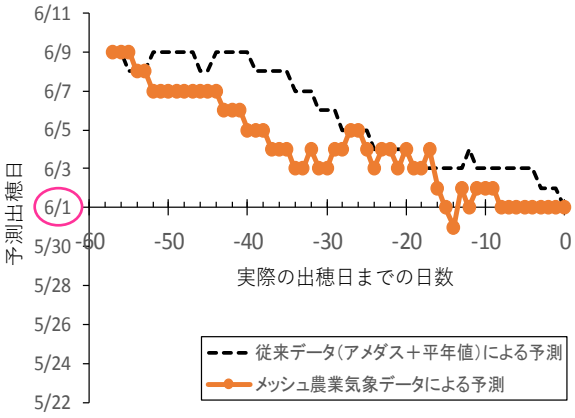


図 6． 北海道十勝地方における 2016 年産小麦（きたほなみ）の発育予測結果。実際に観察された出穂日は 6 月 1 日。

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。

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、2018 年 6 月 1 日現在、232 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。

活用事例１「水稲の発育予測」

鳥取県農業試験場では、メッシュ農業気象データと水稲発育モデルを使って、県下一斉に主要品種を移植した場合の出穂日予測および収穫適期予測を HP 上で提供しています（図７）。データは品種別に標高ごとの表形式で示され、栽培技術情報とともに現場で役立てられています。

図７．水稲収穫適期予測ＨＰ。
鳥取県農業試験場。
http://www.pref.tottori.lg.jp/47787.htm

活用事例２「水稲の生育診断」

栃木県農業試験場では、メッシュ農業気象データと水稲発育モデルを使って、コシヒカリを 5 月 4 日に県下一斉に移植したと仮定した場合の出穂日分布図を作成し、「水稲生育診断情報」の補足情報として各農業振興事務所や県内ＪＡ等に情報提供しています。（図８）

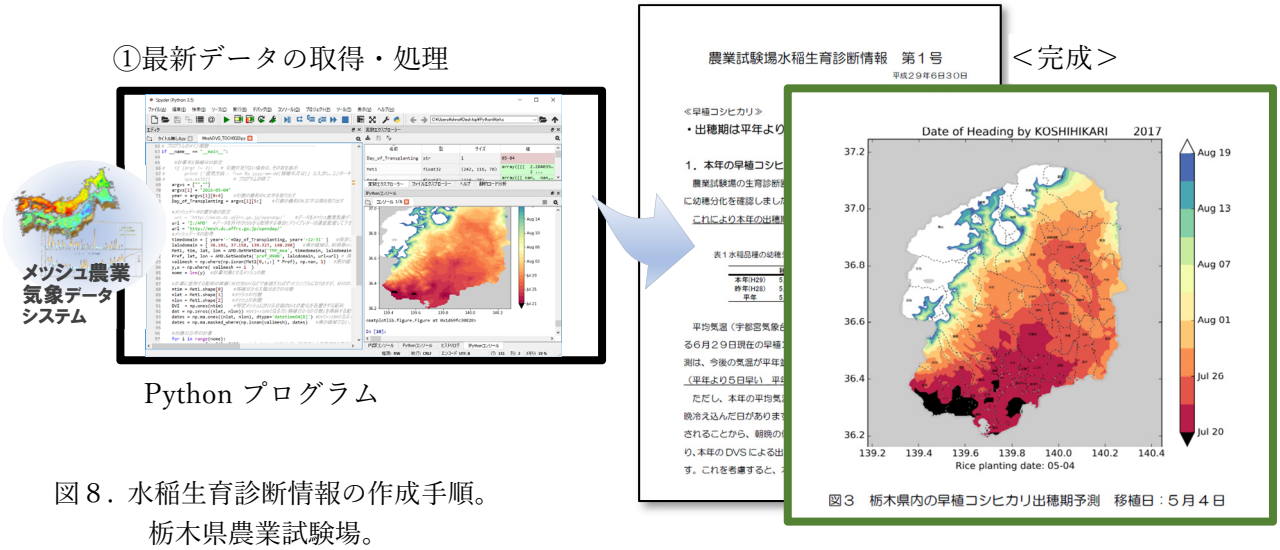


図８．水稲生育診断情報の作成手順。
栃木県農業試験場。

活用事例３「ナシ「幸水」の満開日および収穫始期予測」

茨城県農業総合センター園芸研究所では、ナシ「幸水」の満開日（図９（上））および収穫始期について計算するスクリプトを開発しました。予測値を含むメッシュ農業気象データを利用することにより、平年値を用いる場合に比べて満開日予測の誤差低減効果が認められています（図９（下））。

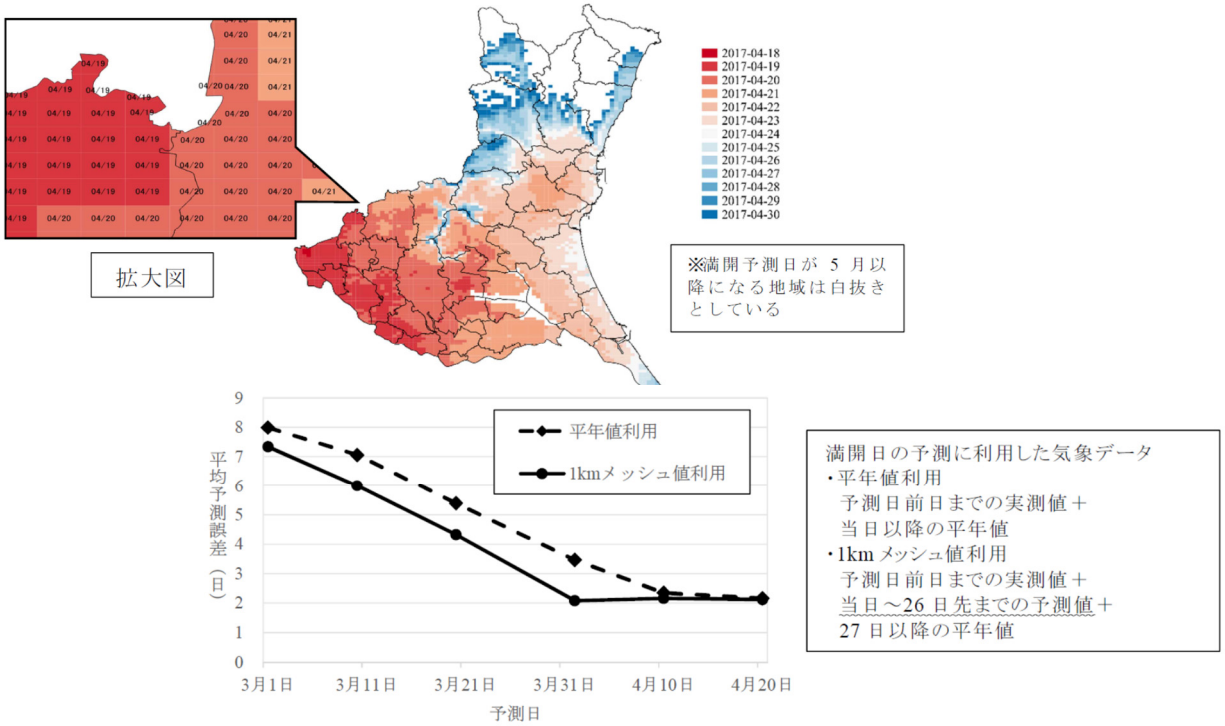


図９．上：GIS ソフトを利用した「幸水」の満開予測日の表示例。下：予測値を含むメッシュ農業気象データ利用による「幸水」満開日予測の誤差低減効果（H24-28）。
茨城県農業総合センター園芸研究所。

活用事例４「マツノマダラカミキリ発生予測」

マツ材線虫は主要な森林病害虫のひとつであり、マツノマダラカミキリがその媒介昆虫であることが知られています。青森県産業技術センター林業研究所では、マツノマダラカミキリの発生予測を行うプログラムを開発し、HP 上で発生予測情報を提供しています（図１０）。メッシュ農業気象データを用いて春から 13℃以上の有効積算気温を計算し、マツノマダラカミキリの脱出時期の予測を分布図および時系列グラフで示しています。

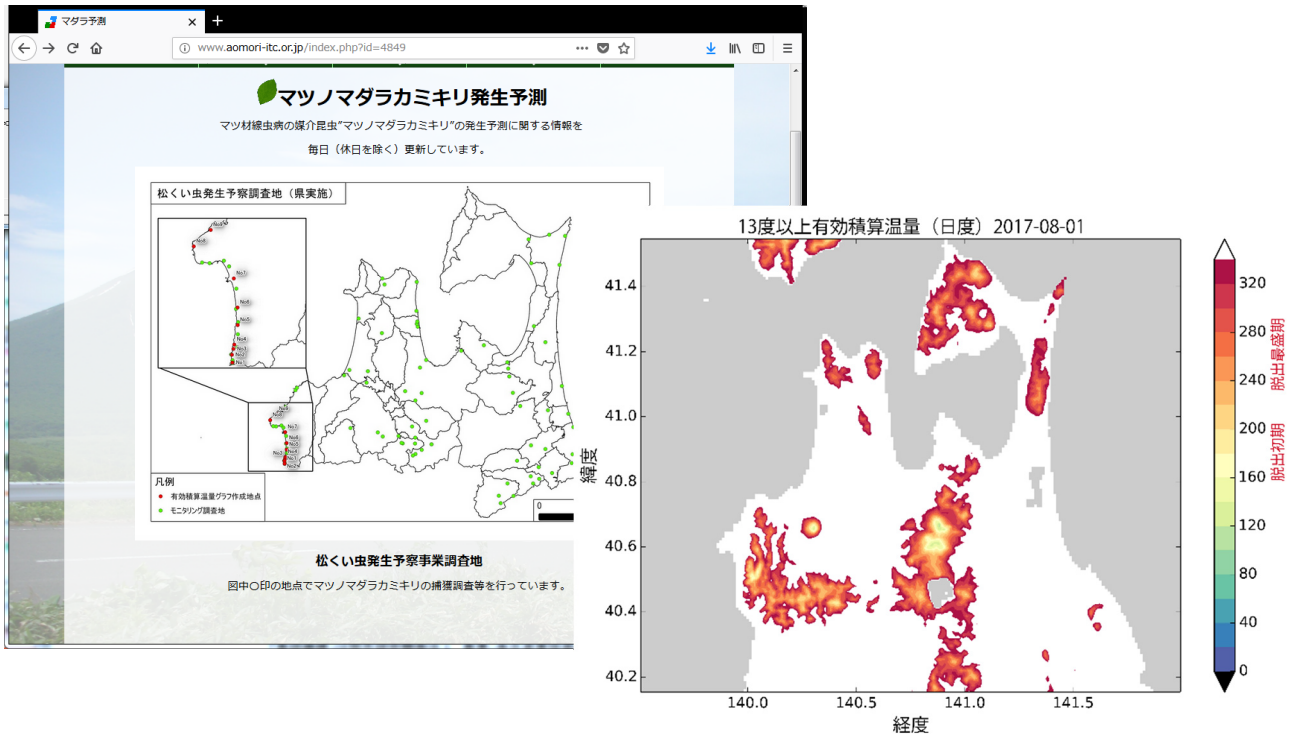


図１０． マツノマダラカミキリ発生予測ＨＰ。
地方独立行政法人青森県産業技術センター林業研究所。
http://www.aomori-itc.or.jp/index.php?id=4849

活用事例５「飼料用トウモロコシ品種選定」

北海道立総合研究機構根釧農業試験場では、飼料用トウモロコシの安定栽培を目指して、主要品種の北海道における適地マップを作製しています（図１１）。過去 20 年のメッシュ農業気象データを使って生育モデルで計算し、利用用途別に適期収穫可能確率マップ（安定栽培マップ）を作製し、DVD 等で配布しています。

活用事例６「樹木の大量枯死の原因究明」

北海道立総合研究機構林業試験場では、メッシュ農業気象データを利用して、2016 年に報告された北海道東部地域における樹木の大量枯死の原因究明が行われました。5 月の最高気温と雨量を GIS で表示したところ、樹木枯損被害報告前の 2014 年および 2015 年の道東地方では、比較的気温が高く雨量が少ない乾燥した期間であったことが確認され（図１２）、このような気象条件が樹木の大量枯死に大きく関係していることが明らかになりました。

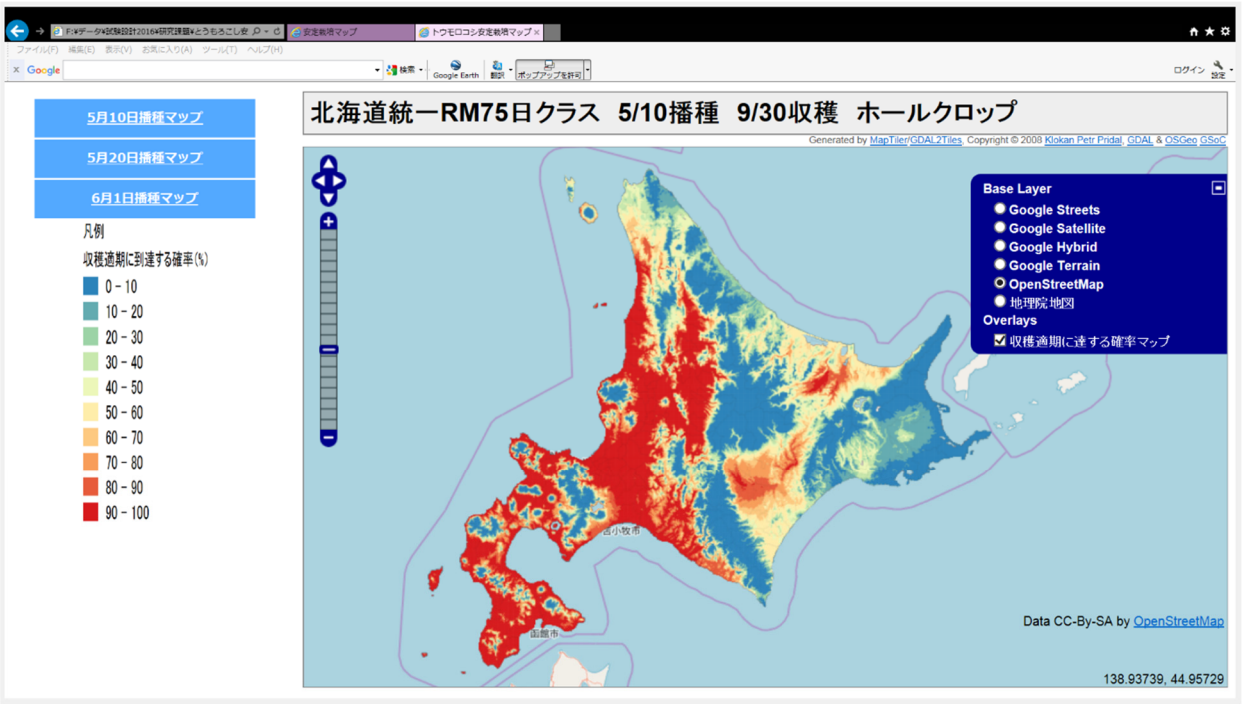


図１１． 飼料用トウモロコシの安定栽培マップ。
ホールクロップ利用において早生の早（北海道統一 RM75 クラス）の品種が収穫適期（総体乾物率 30%）となる確率マップ。設定条件：播種日 5 月 10 日、収穫日 9 月 30 日、気象データ：農研機構メッシュ農業気象データ（The Agro-Meteorological Grid Square Data, NARO）を利用。地方独立行政法人北海道立総合研究機構根釧農業試験場。

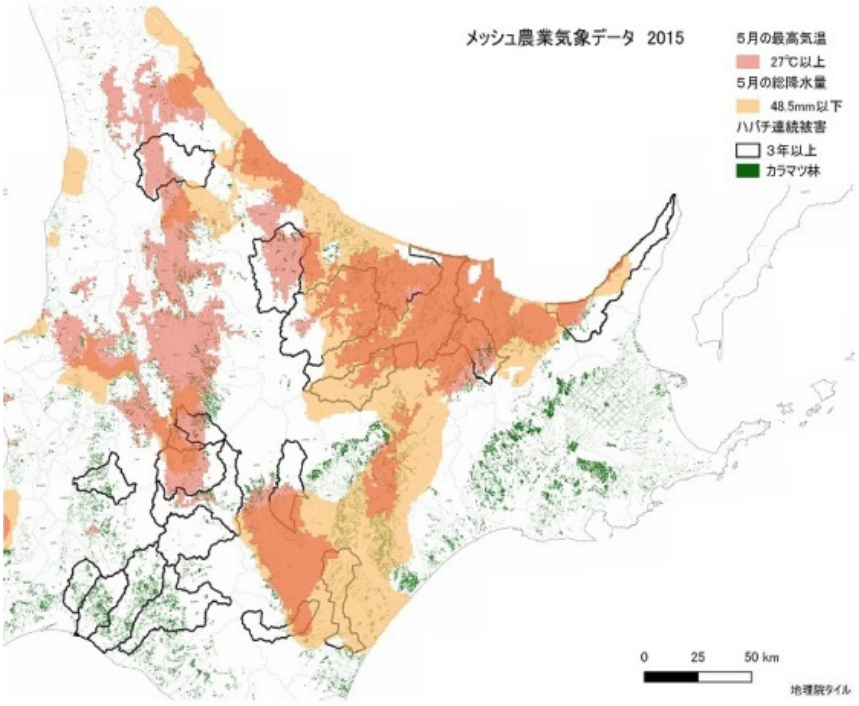


図１２． 2015 年 5 月の最高気温が 27℃以上の地域、および 5 月の総降水量が 48.5mm 以下の地域。
地方独立行政法人北海道立総合研究機構林業試験場。

活用事例７「野菜取引の安定化」

レタス等の葉菜類では、露地野菜での契約取引が増加する中、定時、定量出荷が求められていますが、露地ではダイレクトに気象条件によって生育が左右されるため、正確な出荷時期や出荷量の算出が困難な状況です。そこで、農研機構中央農業総合研究センター（現革新工学研究研究センター）において、レタスの契約取引を行う出荷団体を対象とした、生育シミュレーションに基づく出荷予測アプリケーションが開発されました。

レタスなどの葉菜類は、地上部の成長が気象条件によってよく説明できることが知られており、その性質を利用した生育モデルが作られています。露地野菜出荷予測アプリケーションでは、メッシュ農業気象データを読み出してこの生育モデルに読み込むことによって収穫期予測を行い、収穫可能な在庫収量を把握することで出荷団体間の数量調整および契約交渉・取引が可能となり、不足分の調達や余剰分の販路確保などの対応を行うこともできます（図１３）。

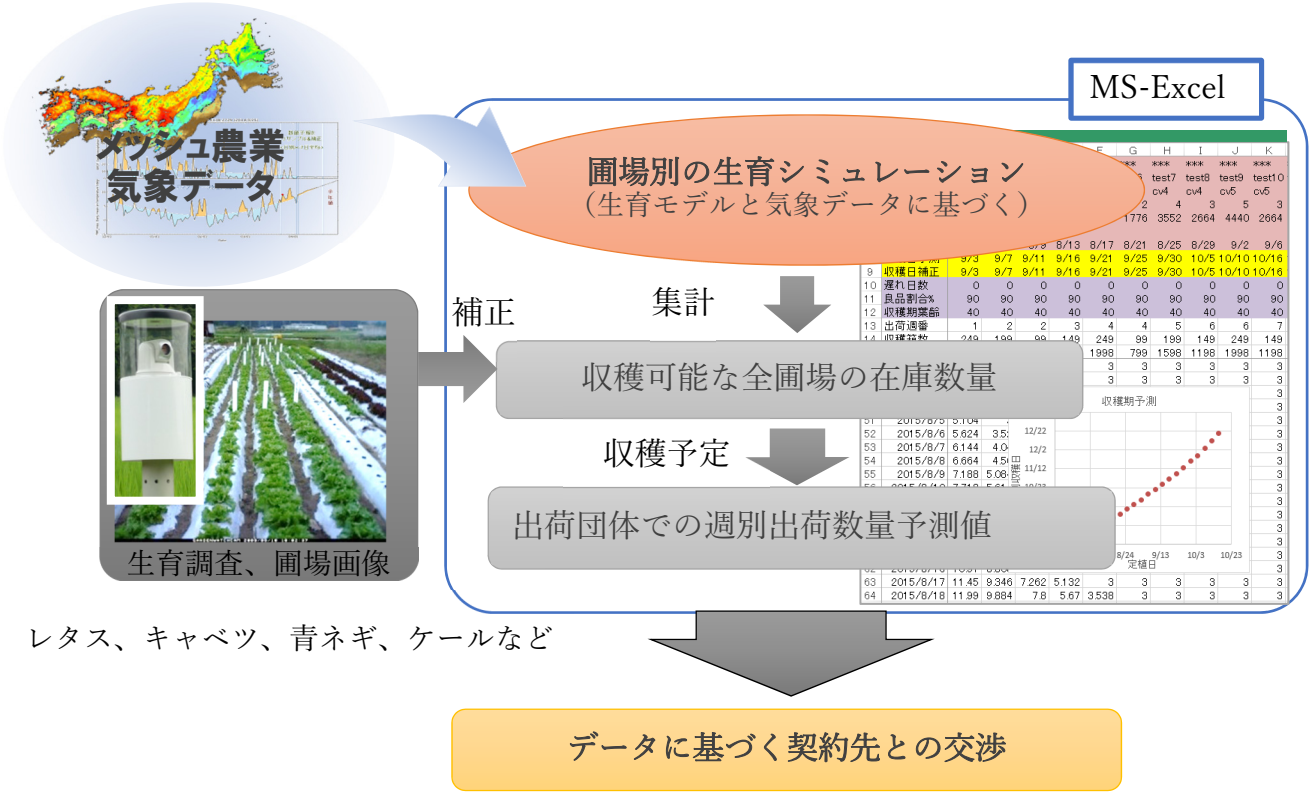


図１３．露地野菜出荷予測アプリケーションの概要図。
農研機構農業技術革新工学研究センター。
http://cse.naro.affrc.go.jp/sugak/yasai/pamphlet.pdf

活用事例８「温室の暖房燃料使用量試算ツール」

農研機構西日本農業研究センターでは、温室の暖房燃料使用量を試算するツールを試作しました。まず、温室位置における暖房期間中の日最高気温と日最低気温をメッシュ農業気象データから取得し、暖房デグリアワーを計算します。これを基に、暖房負荷、燃油使用量が計算され、表示されます（図１４）。

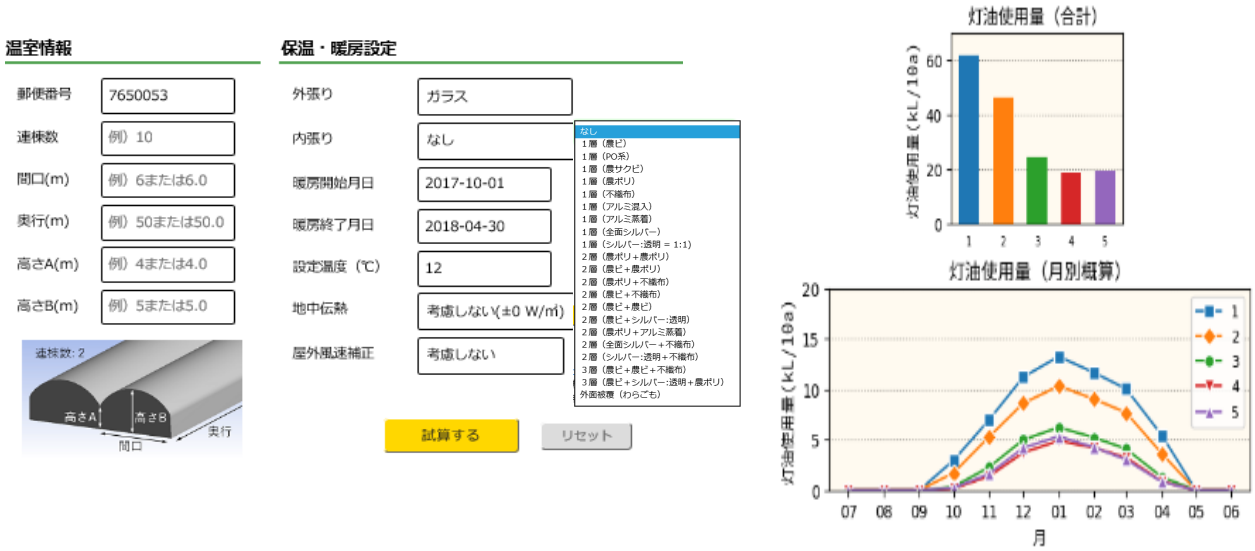


図１４．左：暖房負荷、燃油使用量算出ツールの入力画面。右：燃油使用量試算結果
農研機構西日本農業研究センター。

活用事例９「ドローンマッピング技術を活用した露地野菜出荷予測システムの構築」

アカデミックエクスプレス株式会社では、平成 29 年度いばらきロボット実証試験・実用化支援事業「ドローンマッピング技術を活用した露地野菜出荷予測システムの構築と実証」において、メッシュ農業気象データを生育モデルの入力値とし、キャベツの出荷可能日と収量予測を行いました（図１５（上））。ドローンマッピングによる実際の生育状況の情報で予測値の補正を行い（図１５（下））、高精度の出荷予測を行うシステムを構築しました。

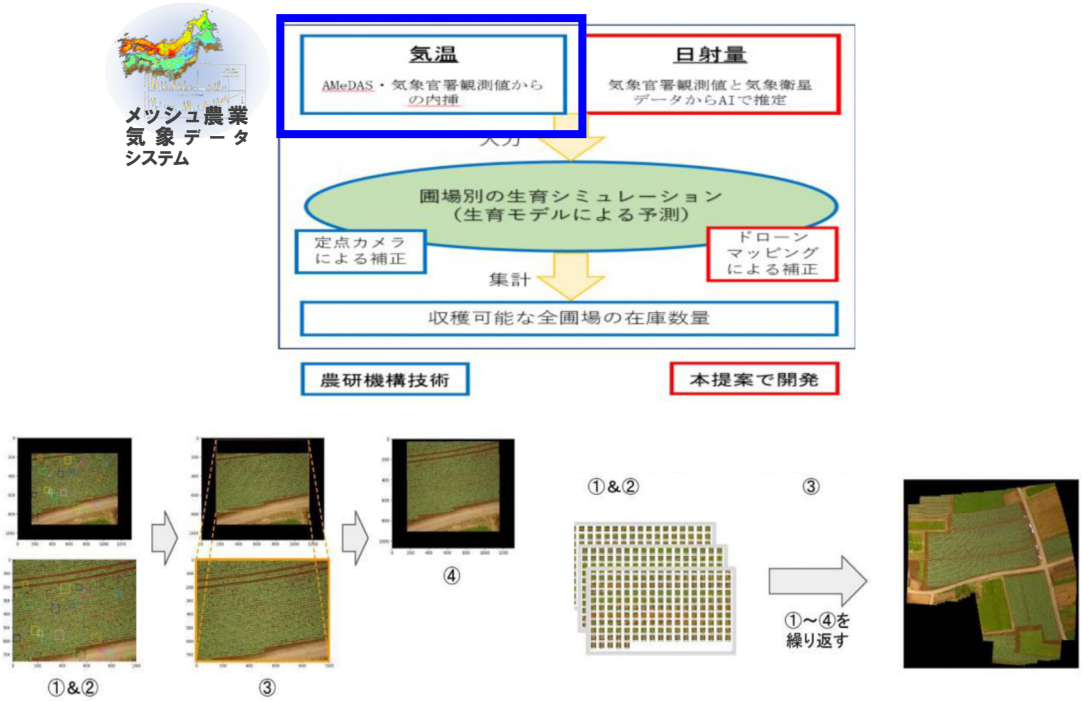


図 １５．上：ドローンマッピング技術を活用した露地野菜出荷予測システム概略図。下：ドローン空撮画像モザイク処理の流れ。
アカデミックエクスプレス株式会社。
平成 29 年度いばらきロボット実証試験・実用化支援事業

活用事例１０「栽培管理支援システム」

農研機構では、これまで蓄積してきた栽培技術、作物モデルと気象予測データを組み合わせた、新しい栽培管理支援情報の創出に取り組んでいます。これまでに、水稲、小麦、大豆を対象とした様々なコンテンツが作成され利用されています（図１６（上））。その中の水稲高温登熟障害回避コンテンツ（図１６
（下））を例にみると、水稲は高温に敏感な期間があり、その期間に気温が高いと米が白く濁る障害が発生します。これを抑えるには、出穂前約 1 週間に追肥を多めに与えることが有効です。しかし一方で、追肥が過剰だと米のタンパク質含量が上昇し食味が損なわれてしまいます。このため、このコンテンツでは、まず 7 月中旬に発育予測モデルで出穂日を予測し追肥日を決定します。そして、追肥日直前に再度発育予測を行って水稲が高温に敏感な期間（出穂後 15 日間）を確定し、気象予報からその期間の平均気温を予測します（図１６（下）黄色部分）。併せて、水稲の葉の色も測定し稲の栄養状態を確認します。そのうえで、予測される気温と現時点の葉の色から最適な追肥量を算出し提示します。

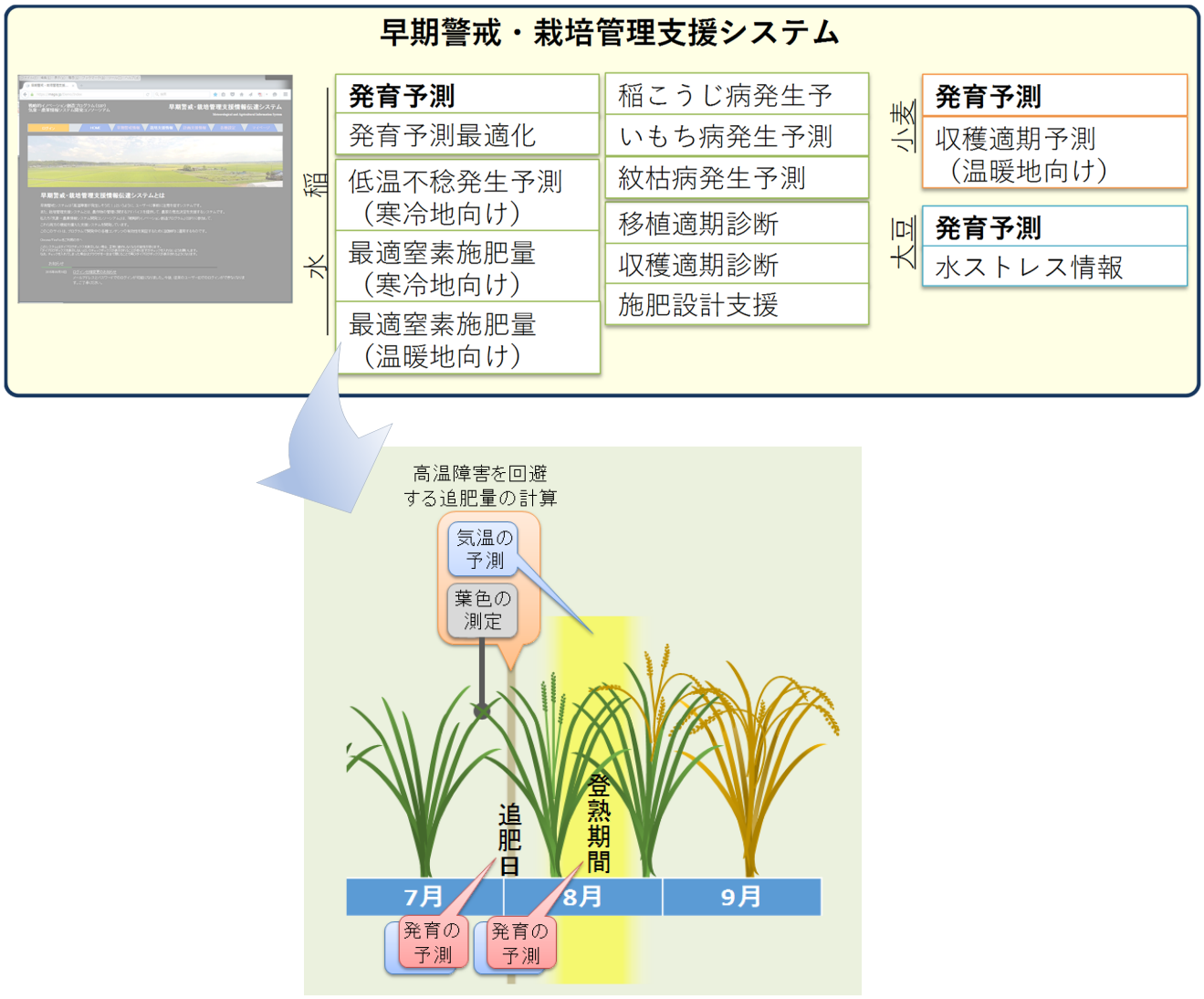


図１６. 上：栽培管理支援システムコンテンツ一覧。下：水稲高温障害回避コンテンツ概略図。農研機構農業環境変動研究センター。

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。

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（農環研シナリオ 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、閣議決定された「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）や環境省・農水省ほかの「地域適応コンソーシアム事業」にも提供されることになっている。ここで、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・予測に多大な利用の実績のある「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開始した。

表　作成した気候シナリオの緒元

使用した全球モデル	GFDL-CM3, HadGEM2-ES, **MIROC5, MRI-CGCM3**, CSIRO-Mk3-6-0, bcc-csm-1, etc.
シナリオ	RCP2.6, **RCP8.5**
DS手法	正規分布型スケーリング法 (Haerter et al., 2011)
計算期間と時間分解能	日値：現在（1981-2005）、近未来（2006-2055） * 一部の気候モデルを2100まで延長中
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）
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

＊太字は現在、早期公開中

【講義 2】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。

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。

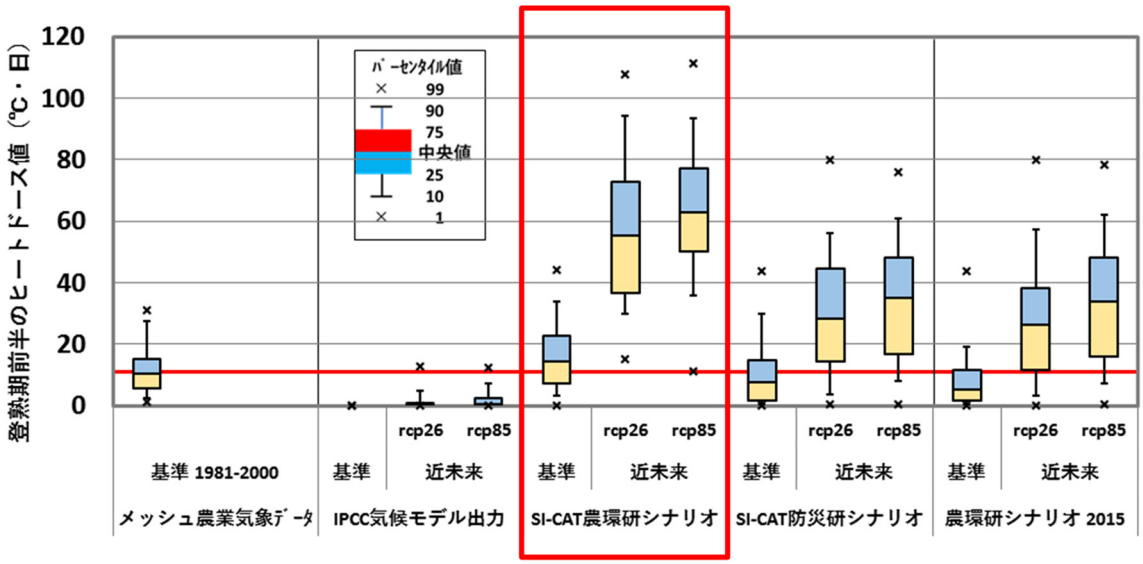


図 1　茨城県南部地帯のヒートドース値の基準期間（1981-2000 年）および近未来期間（2031-2050 年）で RCP 排出シナリオごとの 20 年間での出現確率を示す箱ひげ図。RCP ごとに 6 つの気候モデル全ての各 20 年間、計 120 のサンプルを用いている（文部科学省 SI-CAT 報告書による）。

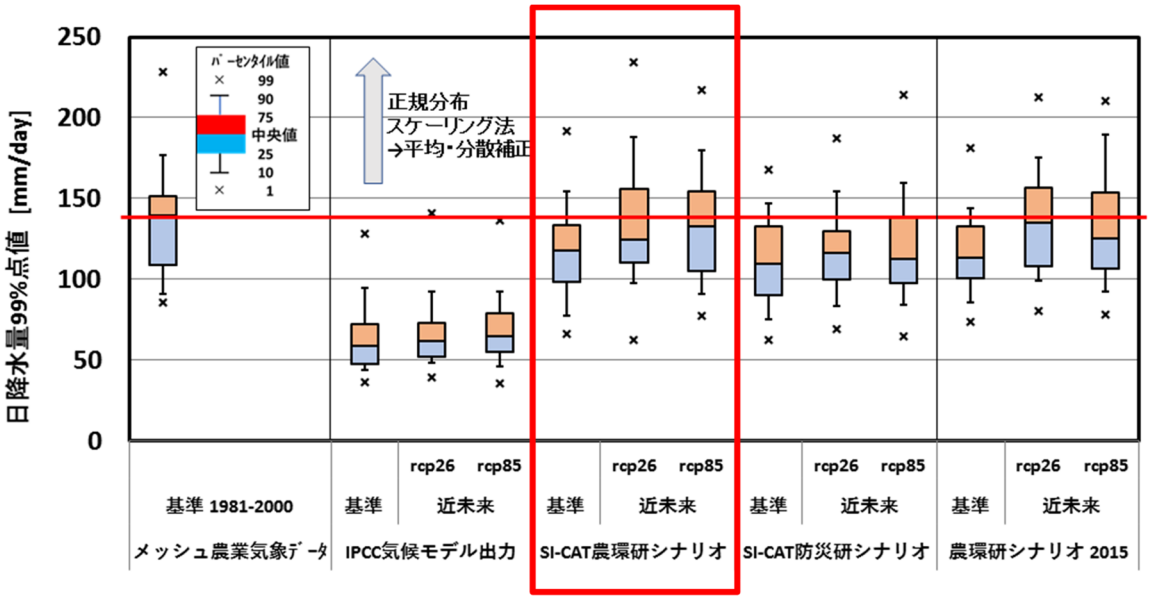


図 2　図 1 に同じ、ただし高知市付近における日降水量年間 99 パーセンタイル値を示す（出典同）。

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。

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。

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

【講義 2】メッシュ温暖化シナリオデータについて

メッシュ農業気象データの取得

農研機構 北海道農業研究センター　根本 学

はじめに

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。

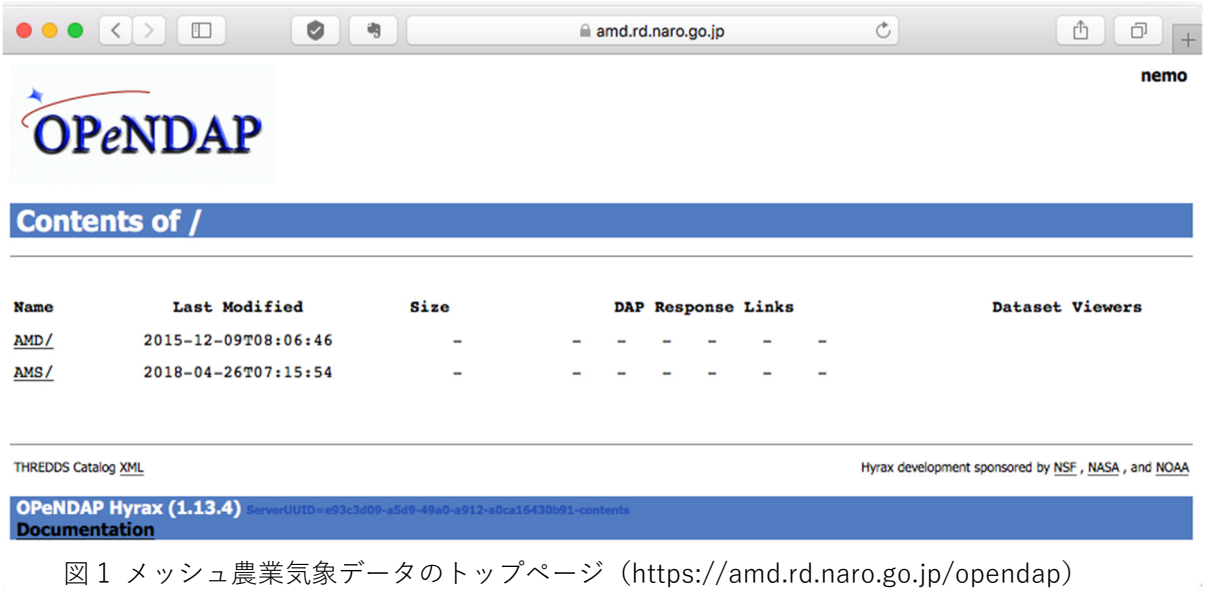


図 1 メッシュ農業気象データのトップページ（https://amd.rd.naro.go.jp/opendap）

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。そのためには、まず、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 2）。　すると、図 3 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。

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。

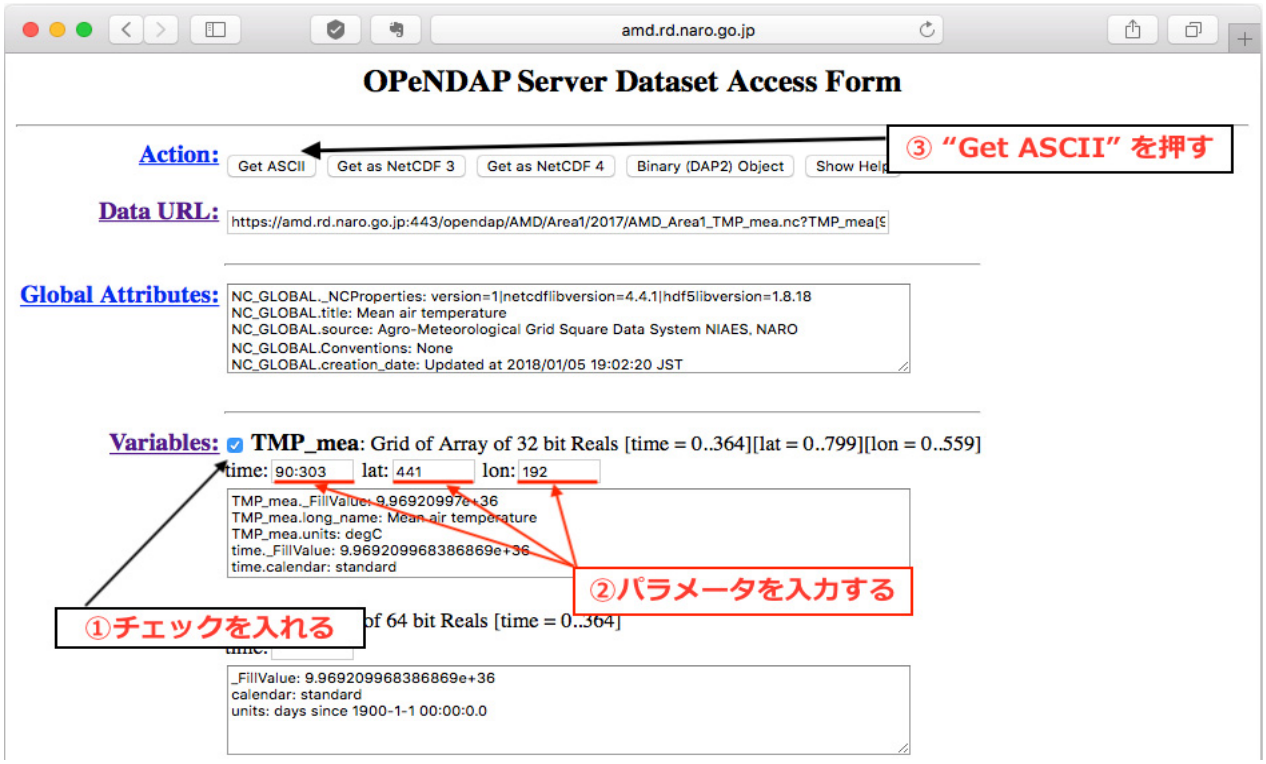


図 2 北農研の 2017 年 4 月 1 日から 10 月 31 までの日平均気温を取得する際の操作

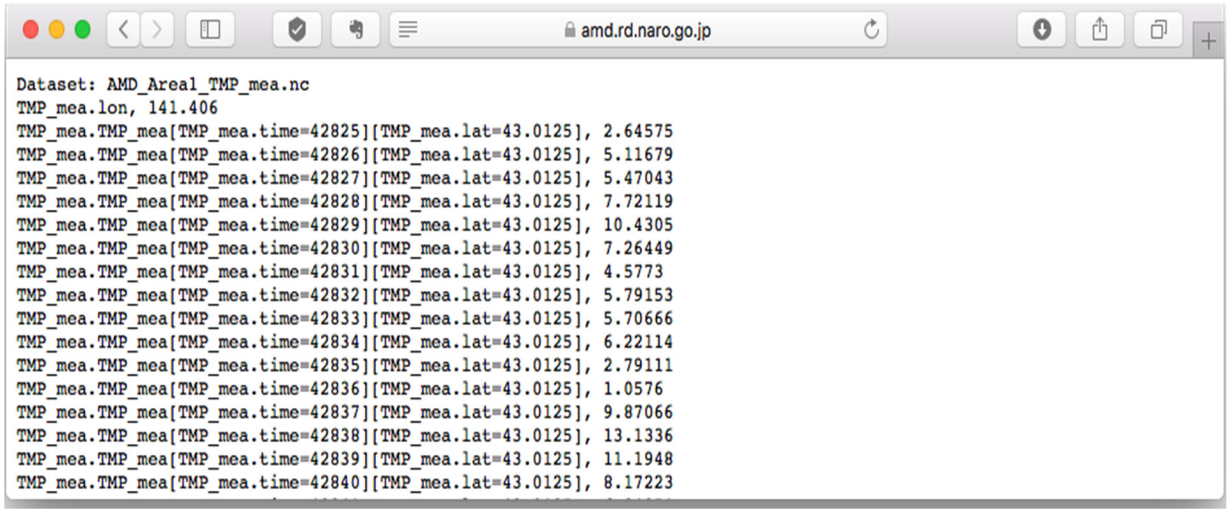


図 3　図 2 の操作で得られる csv データ

※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。

【講義 3】メッシュ農業気象データの取得

本講義では、そんなメッシュ農業気象データを取得する方法として、初級編、中級編、上級編の 3 段階に分けて説明します。初級編は、広く利用されている表計算ソフフトウェアである、Microsoft 社の Excel を用い、専用に作成したエクセルファイルを用いてデータを取得する方法を示します。中級編は、python というプログラミング言語を利用しますが、専用のライブラリが用意されているので、数行の簡単な操作でデータ取得が行えます。上級編では、プログラミング言語に依存しない、データ取得の考え方について簡単に述べます。

なお、この講習では、上記の例と同じ、北農研の圃場（北緯 43.0095°、東経 141.4080°）を含むメッシュについて、2017 年 4 月 1 日から 10 月 31 日までの日平均気温データを取得する方法について扱います。

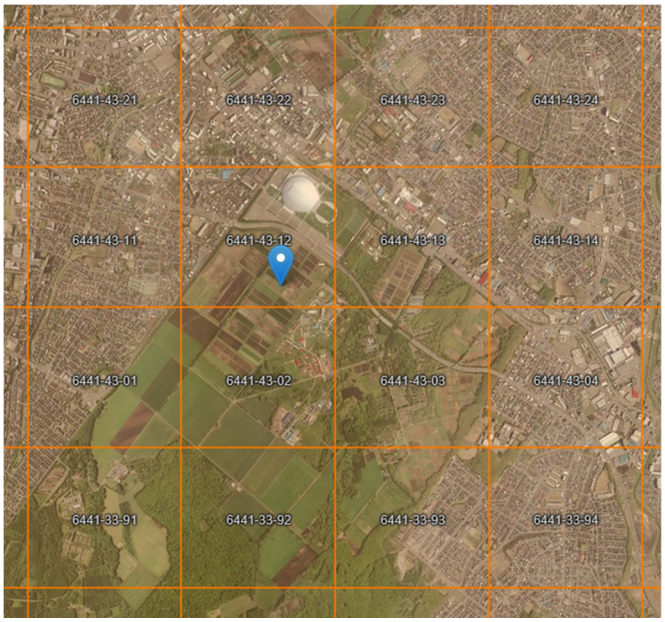


図 4　北農研気象観測露場の位置

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）

初級編：エクセルファイルによるデータ取得

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、「地点抽出用」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

- その際に、「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。

3）データを取得する

「データ取得シート」を開き、必要な項目を入力／選択していきましょう。

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、図 5 のように、欲しい地点の平均気温が、エクセルシート上に、図と共に表示されます。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してくるので、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。

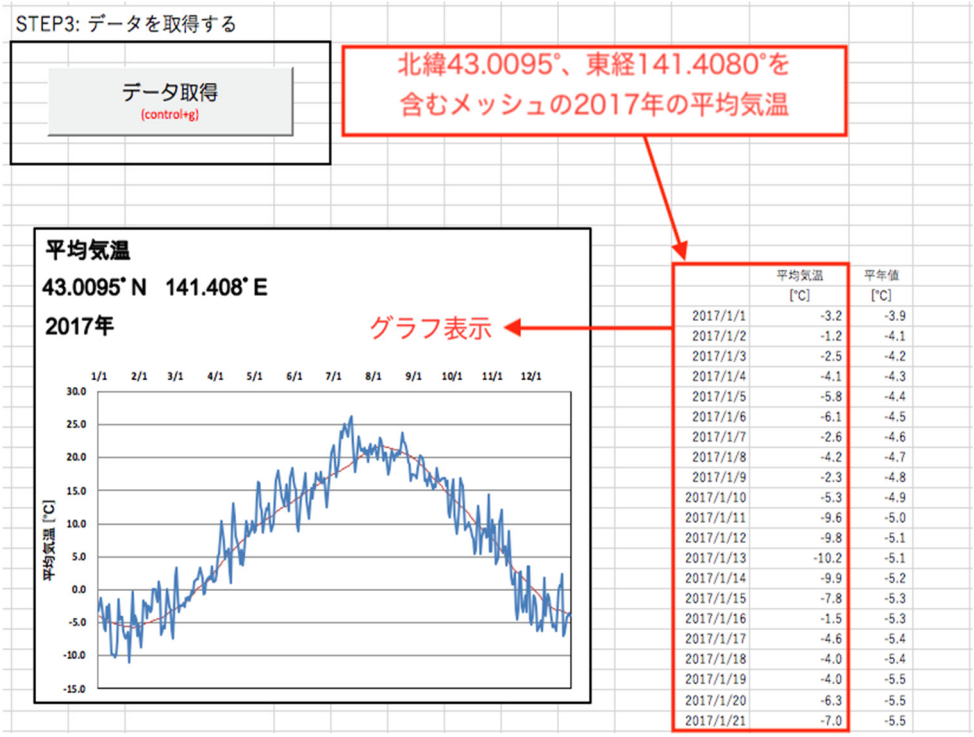


図 5　エクセルシートでデータを取得した画面

中級編：python プログラミングによるデータ取得

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、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（例: c 言語やシェルスクリプト）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（後の講義で説明があります。）、とにかく python でデータを取得してみましょう。

1) 下準備

(1) python を利用する環境として、Anaconda と追加パッケージをインストール

公開 wiki (https://amu.rd.naro.go.jp/wiki_open/doku.php?id=python)

→ 「初めて利用される方へ」にある、Python 利用環境構築ガイドが参考になります。

(2) 開発チームによるライブラリ（AMD_Tools3.py）を用意

公開 wiki (https://amu.rd.naro.go.jp/wiki_open/doku.php?id=python)

→ 「利用ツール・サンプルプログラム」→「利用ツール」にある、AMD_Tools3.py をダウンロードし、作業スペース（どこでも構いません）に配置しておきます。

※ AMD_Tools3.py をテキストエディタ等で開き、利用者 ID とパスワードを所定の位置に記入しておいてください。

(3) 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

Spyder 右上のフォルダマークをクリックすると現れる画面上で、フォルダの移動が簡単です。

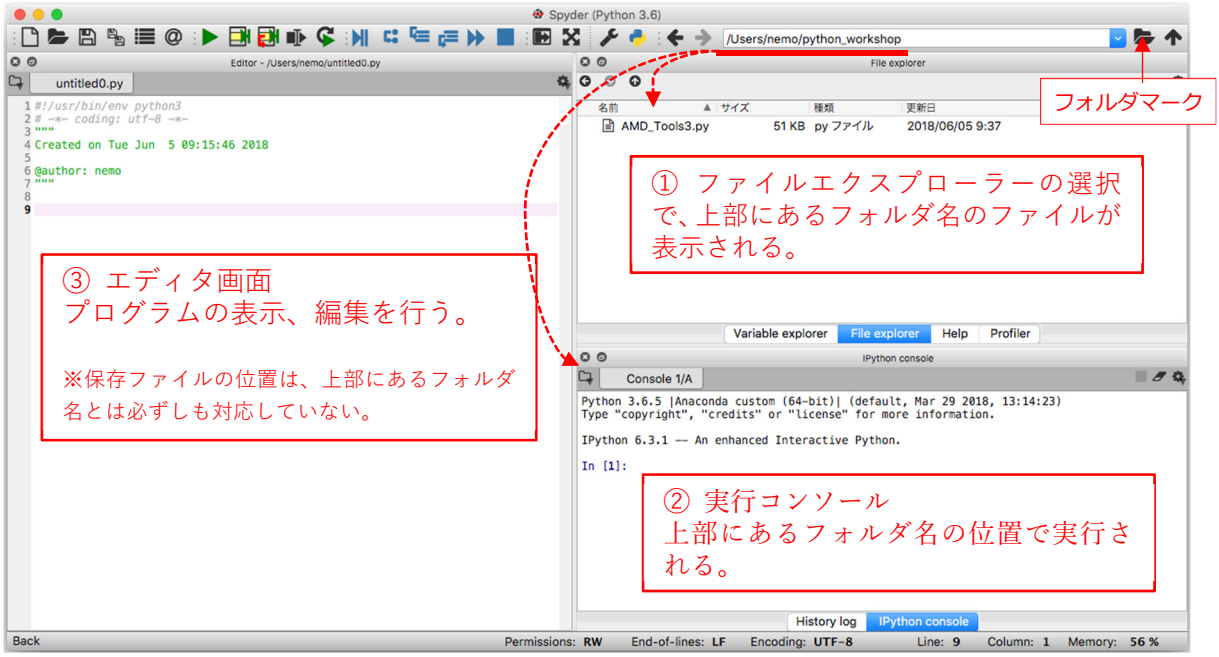


図 6 Spyder を起動して、ファイルエクスプローラーを表示した画面

2) データの取得

ここでは、実行コンソール（図 6 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 7 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

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 8 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 8 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 9)。

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

図 9 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 10 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 10 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 11）。

```
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 11 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 12）。

図 12 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。

上級編：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。
図 2 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.n
    c?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 12 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。

【講義 3】メッシュ農業気象データの取得

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 13）。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。

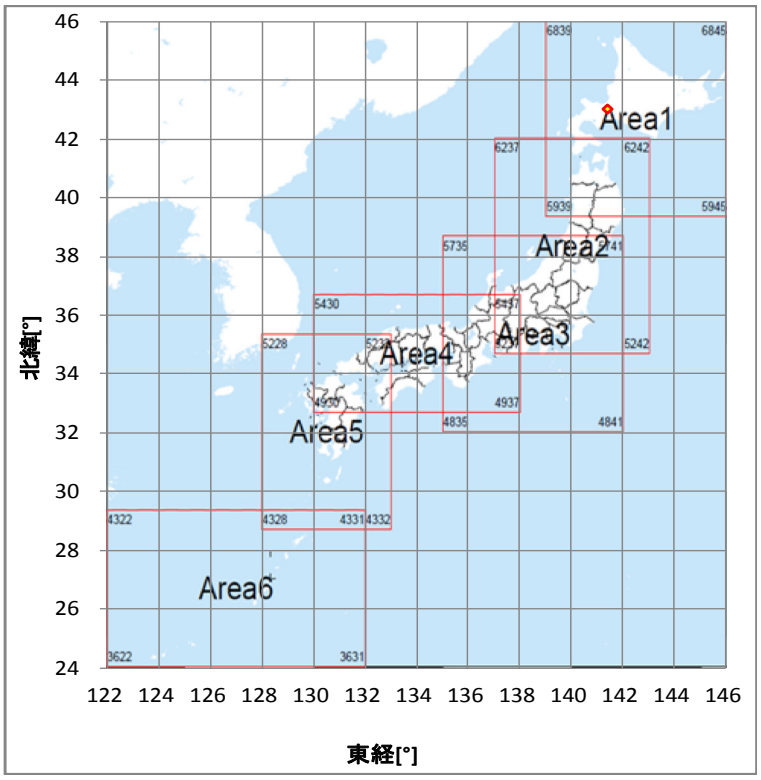


図 13 メッシュ農業気象データの 6 つのエリア

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

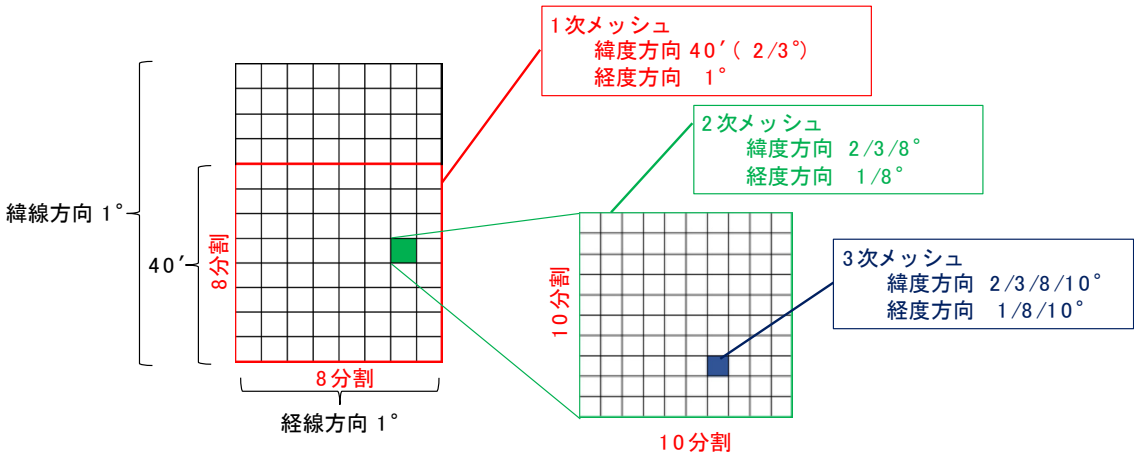


図 14　3 次メッシュの大きさ

【講義 3】メッシュ農業気象データの取得

3 次メッシュの大きさは、図 14 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。

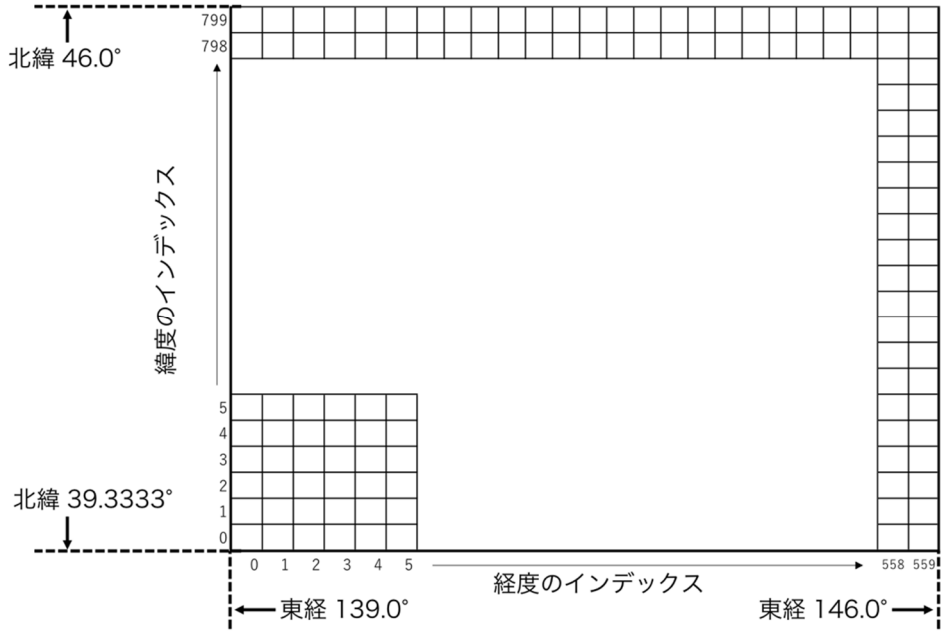


図 15　Area 内のインデックスの数え方（例：Area1）

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 15 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。

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、講義 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、エクセルによる方法（初級編）、python を利用する方法（中級編）、その他プログラミング言語で扱う場合のヒント（上級編）について、扱いました。

補足

ここでは、単地点のデータ取得についてのみ説明しましたが、2 次元（空間分布）データも可能です。

エクセルファイルについては、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取得エクセルファイル」に「2 次元抽出用」ファイルが用意されています。

python での空間分布データの扱い方については、実習編で扱います。また、公開 wiki（https://amu.rd.naro.go.jp/wiki_open/doku.php?id=python）にサンプルプログラムが用意されています。

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・大野宏之

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。

Spyder について

1) カラースキーム

Spyder のエディタ部分にプログラムを書き込むと、自動的にプログラムの文字に色がつきます。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。デフォルトでは以下のように色分けがなされています。

○Spyderのエディタにpractice.pyを開きます。

→コードが表示されます。

☑コードに色が付いていることを確認しましょう。

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[1] （abs, dict, help, min, max, all など） 組み込み定数[2]（False, True, None など） 組み込み型[3] （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[4] （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

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

2）確認の操作

Spyder の実行コンソールの、**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 の基本を学んでゆきます。この際、テキストを見ながら数字や文字を打ち込むのは大変なので、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。これを転記することにします。

エディタペインに表示されているものをプロンプトに転記するにはいくつか方法があります。一つめの方法は、まず転記したい部分を選択し、次に Ctrl + C (Ctrl キーを押しそれを離さずにさらに C キーを押す)を押し、その後マウスカーソルを実行コンソールに動かして一度クリックしてから今度は Ctrl + V を押します。もう一つの方法は、転記したいものを選択しそのままの状態で Shift + Enter を押します。

Python の基礎

1）コメント

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]:
```

```
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
```

※"<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'
```

☑文字列はコメントを書くときにも使われます。ファイル practice.py の 33～35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（Windowsでは場合によって¥nと表示）は改行を示す文字コードです。

(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。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。

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

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'
```

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
```

なお、数学における「x = 3 のとき、・・」のように、変数に値を入れるのではなく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 (=) を二つ使うので、違いに注意してください。

```
x == 3
```

Python では、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ができます。

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

数学で、「$x = x + 3$」と書くと先生から×(バツ)をもらいますが、Python プログラムでこのように書くと、「今 x に格納されている数に 3 を加えなさい。」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、上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。Python 使いは存外ものぐさかもしれません。

```
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

(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

なデータ形式です。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。リストは結構節操がなく、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]]          (混ぜこぜのリスト)
```

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。

リストには、要素があり普通の数値と違います。そして、違うがために、要素を並べ替えるとか、指定した値と一致する要素の番号を調べるなどのような、単なる数値には必要のない操作が必要となります。このように単純な数値にはない性質を持ったデータ形式をクラスといい、そのクラスに特有の、痒い所に手が届く操作をメソッドと呼びます。メソッドは、クラス（正確には、クラスのインスタンス）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。

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。「○○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。

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'
```

リストの一部分を取り出すこともできます。上の例で示したリストから、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てください。

このほかにも、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ておいてください。

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

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。

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。変数 mixed に下のようなリストが代入されていて、これから、「200」を取り出す場合考えましょう。

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

リスト mixed の要素であるリスト`[100, 200, 300]`のリスト mixed における場所は、人間の数え方で 4 なので Python の数え方では 3。200 は要素であるリスト`[100, 200, 300]`の 2 番目なので Python の数え方で 1。これらを用いて`mixed[3][1]`とします。

リストは、代入文で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。このように、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、リストに似ているがきわめて融通が利かないデータ形式が用意されています。

(2) タプル

融通の利かないリストがタプルです。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 番目の要素に名称の文字列、2 番目の要素に地点の緯度、3 番目の要素に経度を格納する約束にすれば、例えば、station[0]とすることで地点名が取り扱え便利です。けれど、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、station["name"]で地点名が取り出せる、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。

```
station = {"name":"Tsukuba", "lat":36.0270, "lon":140.1104,
"asl":-999.9}
```

```
In [23]: station = {"name":"Tsukuba",
    ...:            "lat":36.0270,
    ...:            "lon":140.1104,
    ...:            "asl":-999.9}
    ...: station
Out[23]: {'asl': -999.9, 'lat': 36.027, 'lon': 140.1104, 'name': 'Tsukuba'}
```

☑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。

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 [24]: station["name"]
Out[24]: 'Tsukuba'
```

```
In [25]: station["asl"] = 11.3
    ...: station["asl"]
Out[25]: 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 [26]: station.values()
Out[26]: dict_values(['Tsukuba', 36.027, 140.1104, 11.3])
```

```
In [27]: station.keys()
Out[27]: dict_keys(['name', 'lat', 'lon', 'asl'])
```

得られた値はイテラブルオブジェクト（リストのように値が並んだもの。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
```

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、タプルなど）

処理：変数に代入された値を使っておこなう処理

☑行末のコロンを忘れずにつけてください。

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 [28]: 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
```

※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。

```
In [29]: for i in [0, 1, 2]:
    ...:     print(i)
    ...:
    ...: print(i,"までで終了")
0
1
2
2 までで終了
```

※print関数は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 関数は行頭から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(値の列)としてリストを用いましたが、このような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 関数は、初めて使う人を必ず驚かせます。それは、引数までの数列ではなくその一つ手前までしか作らないからです。range 関数を使って上と同じ処理をする場合は、以下のようにします。

```
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(*最初の数字，最後の数字より 1 大きい数，とびとびの間隔*)

【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 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荒業が 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。

☑リスト内包表記ではコロンを使いません。

[*式* for *変数名* in *イテラブルオブジェクト*]

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、最初は面食らいますが、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 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。

```
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
```

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

※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、ループを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 [31]: 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。

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 b2 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
```

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は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(使う文よりも上の行で)定義しておかなければなりません。

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
```

※このケースでは、calc_add_tax(s1, s2, 0.05)でも実は動きます。が、このような使い方はしないようにしましょう。

オプション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
```

ライブラリとインポート

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。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せたら便利です。

Python ではそれが可能です。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、別ファイルのプログラムからそれを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（たとえば、図 1　46 行目）。

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を皆さんが作る必要はありません。この関数、GetMetData 関数は、すでに私たちが開発し、モジュール AMD_Tools3.py に収めているので、皆さんは、AMD_Tools3 をインポートすれば、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ライブラリをインポートして使用しています。その証拠に、AMD_Tools3.py を開いてみると、38～58 行目にはたくさんの import 文が記述されています(図 1)。

※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。

```
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 1　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。一方で、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 1 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。

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。ただし、このライブラリに用意されている他の関数はインポートされないので使えません。

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。メッシュ農業気象データを扱う際には必須のライブラリと考えてよいでしょう。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てください。

```
import numpy as np
```

○ Spyder で実際に NumPy をインポートしてみてください。

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。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。1 次元や 2 次元のデータをリストで取り扱うことも不可能ではありませんが、ndarray は圧倒的に強力で、早く、簡単です。

np.array 関数を使うと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 として演算されます。

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

【練習問題】

npdata どうしの演算と data どうしの演算を実行し、演算結果の違いを比較してみましょう。

※npdata は ndarray 型です。一方、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.]]])
```

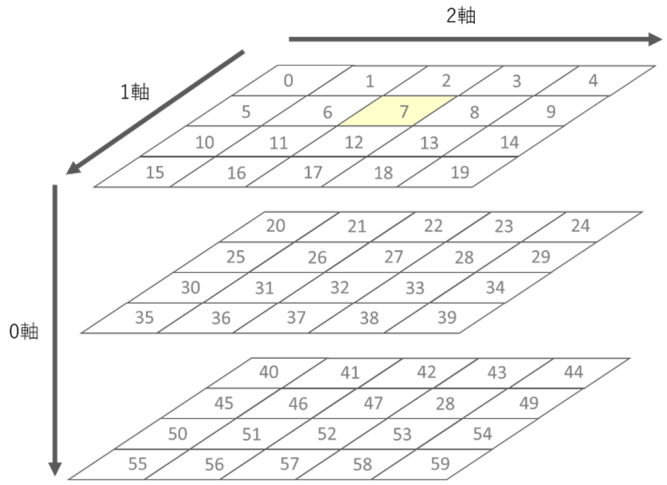
今度は、別な方法で、各要素に格納される値が 1 ずつ増える同じサイズの 3 次元配列を作ってみます。

```
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 2 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。



※図では、小数点を省略しています。

図 2　3 次元 ndarray `arr` の模式図

図 2 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します。

```
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 3)。

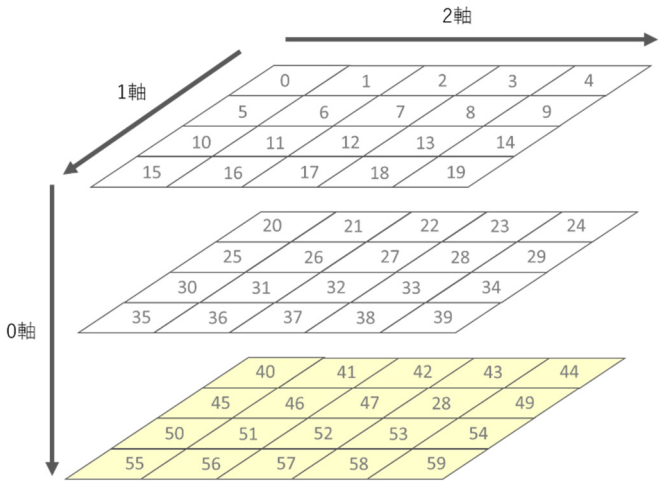


図３　0 軸方向の 0 から数えて 2 番目のすべての要素(黄色)

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。リストにおける記法と同じです。

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 4)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.])
```

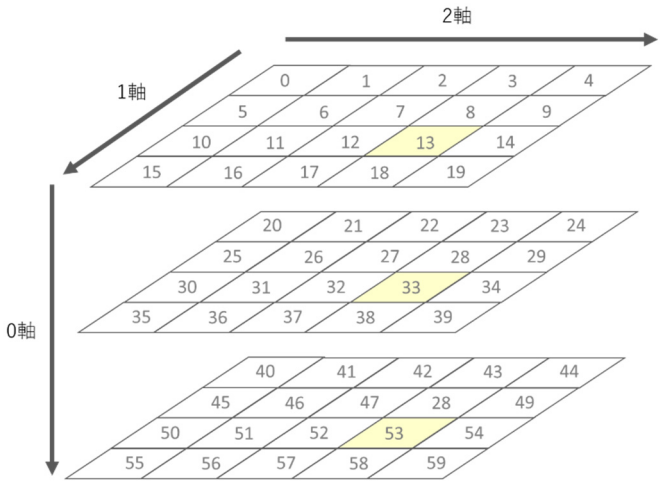


図 4　array[:, 2, 3]の選択範囲

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 2 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。

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

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

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 3.6.3 ドキュメント
   https://docs.python.jp/3/index.html
2. Python 3.6.3 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

NumPy
http://www.numpy.org/。

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)    #気象データを取得
    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] )
```

○Spyderのエディタペインにaet0.pyを開きます。

→エディタ画面に右と同じスクリプトが表示されます。

プログラムが Spyder のエディタ画面に開き表示されている状態で、ツールバーの三角印をクリックします。すると、プログラムが実行され、実行コンソールに下のような表示が出力されます。

```
In [1]: run aet0.py
TMP_mea (153, 1, 1)
--- 0 2018-05-01 00:00:00 20.807035 10.807035446166992
--- 1 2018-05-02 00:00:00 19.30691 20.11394500732422
--- 2 2018-05-03 00:00:00 21.106798 31.22074317932129
--- 3 2018-05-04 00:00:00 15.607003 36.82774639129639
--- 4 2018-05-05 00:00:00 16.607264 43.43500995635986
--- 5 2018-05-06 00:00:00 19.607098 53.042107582092285
--- 6 2018-05-07 00:00:00 17.306952 60.349059104919434
--- 7 2018-05-08 00:00:00 12.806986 63.15604496002197
--- 8 2018-05-09 00:00:00 11.806999 64.96304416656494
--- 9 2018-05-10 00:00:00 11.107342 66.07038593292236
--- 10 2018-05-11 00:00:00 15.007247 71.07763290405273
--- 11 2018-05-12 00:00:00 18.00721 79.08484268188477
--- 12 2018-05-13 00:00:00 17.007109 86.09195137023926
--- 13 2018-05-14 00:00:00 19.107248 95.19919967651367
--- 14 2018-05-15 00:00:00 20.907236 106.10643577575684
--- 15 2018-05-16 00:00:00 21.807234 117.91366958618164
--- 16 2018-05-17 00:00:00 23.407084 131.32075309753418
```

○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。

特定メッシュにおける指定した期間の有効積算気温の計算

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るかを、順に解説します。

まず初めに，メッシュ農業気象データから日平均気温を取得します。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 に代入しています。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，時刻の並びに対応した日平均気温（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、45 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。先の図の 41 行目から 45 行目までが下の図の 41 行目の max()で、先の図の 50 行目から 54 行目までが下の図の 46 行目の max()で置き換えることができます。

```
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。

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。

【実習 2】Python によるメッシュ農業気象データの処理 1

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

【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 2】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、有効積算温度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℃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ください。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。

【回答例 3】

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

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。

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。

○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°C 日を超えた日が表示されます。

【実習 2】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 のエディタに開き、講師の解説と指示に従って、修正を加えてくだ

【回答例 4】

以下のよう修正します。

```
    # データを取得する期間の指定
    #  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。
    itsu = ["2018-03-01", "2018-07-31"]                          #     <--------

    # 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) #気象データを取得
    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
    Tccx = 950.0            #有効積算気温の上限値
    Tcc = np.zeros(ntim)    #有効積算気温の入れ物

    # 最終日((ntim-1)番目の配列要素)は特別扱いが必要です。
    Tcc[ntim-1]=Tccx                                             #     <--------
    print("---", ntim-1, tim[ntim-1], T[ntim-1], Tcc[ntim-1] ) #     <--------

    #(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。
    for t in range(ntim-1,0,-1):          #ループを91,90,89、・・・と回す。    <--------
        Tcc[t-1] = Tcc[t] - max(T[t]-Ts, 0.0)                    #     <--------
        print("---", t-1, tim[t-1], T[t-1], Tcc[t-1] )           #     <--------
        if Tcc[t-1] < 0.0 :                                      #     <--------
            break

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

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。

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。その前後に「/」や「年」、「-」、「.」などを適宜書きます。

○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。

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。

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 Python 3，Bill Lubanovic （著），斎藤康毅（監訳），長尾高弘（訳），オライリー・ジャパン，2015.

Python によるメッシュ農業気象データの処理 2

農研機構 東北農業研究センター 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。出力したテキストを QGIS 上で表示させる方法もオプションとして紹介します。

実習で使用するファイル

aet5.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。

aet6.py

aet5.py を拡張し、単一地点でなく一定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。

GISworks.zip（オプション）

aet6.py の結果を使い、GIS ソフトウェア（QGIS）上でテキストデータを表示する方法を紹介する。

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 aet5.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。

○aet5.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。

```
1 # -*- coding: utf-8 -*-
2 """
3 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」
4 実習:Pythonによるメッシュ農業気象データの処理2
5
6 1)「単一地点の」有効積算気温を計算し(前パートと同一)
7 2)有効積算気温の推移をグラフで描画し
8 3)テキストをcsvファイルとして出力する。
9
10 """
11 #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。
12 import numpy as np                  #配列計算を高速に実行するためのモジュール
13 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
14 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　aet5.py（その 1）

aet5.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　aet5.py（その 2）

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　aet5（その 3）

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)
76
```

図４　aet5.py（その 4）

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')
```

→aet5.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。 なお、67 行目 plt.title の「str(lat)」、「str(lon)」の箇所は、変数 lat（緯度）、変数 lon（経度）の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。

【実習 3】Python によるメッシュ農業気象データの処理 2

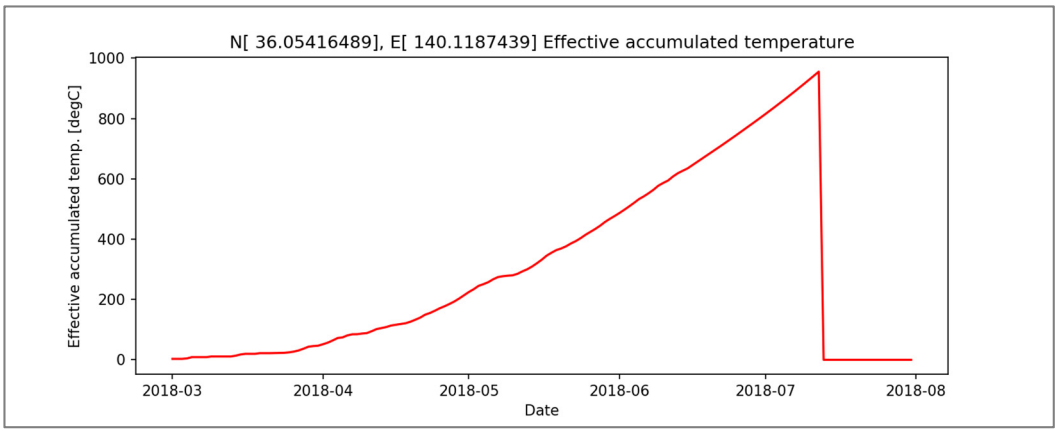


図 5　chart.png

aet5.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：aet6.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、Anaconda 環境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。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　aet5.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。aet5.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。aet5.py 内にもコメントとして記載しました。

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。aet5.py を 1 地点から面に拡張した aet6.py を例に紹介します。

aet6.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。

aet6.py の冒頭部分の抜粋を図 7 に示します。aet5 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。aet6.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。

```
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　aet6.py（その 1）

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 aet5.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（aet5.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。

→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。

なお、aet6.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。

```
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
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
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
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
```

図 8　aet6.py（その 2）

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 ― 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う。

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

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 9 aet6.py（その 3）

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９）。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、aet6.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。aet6.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。aet5.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して使うことも可能です。

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）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。

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。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。

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 4】メッシュ農業気象データの特性について