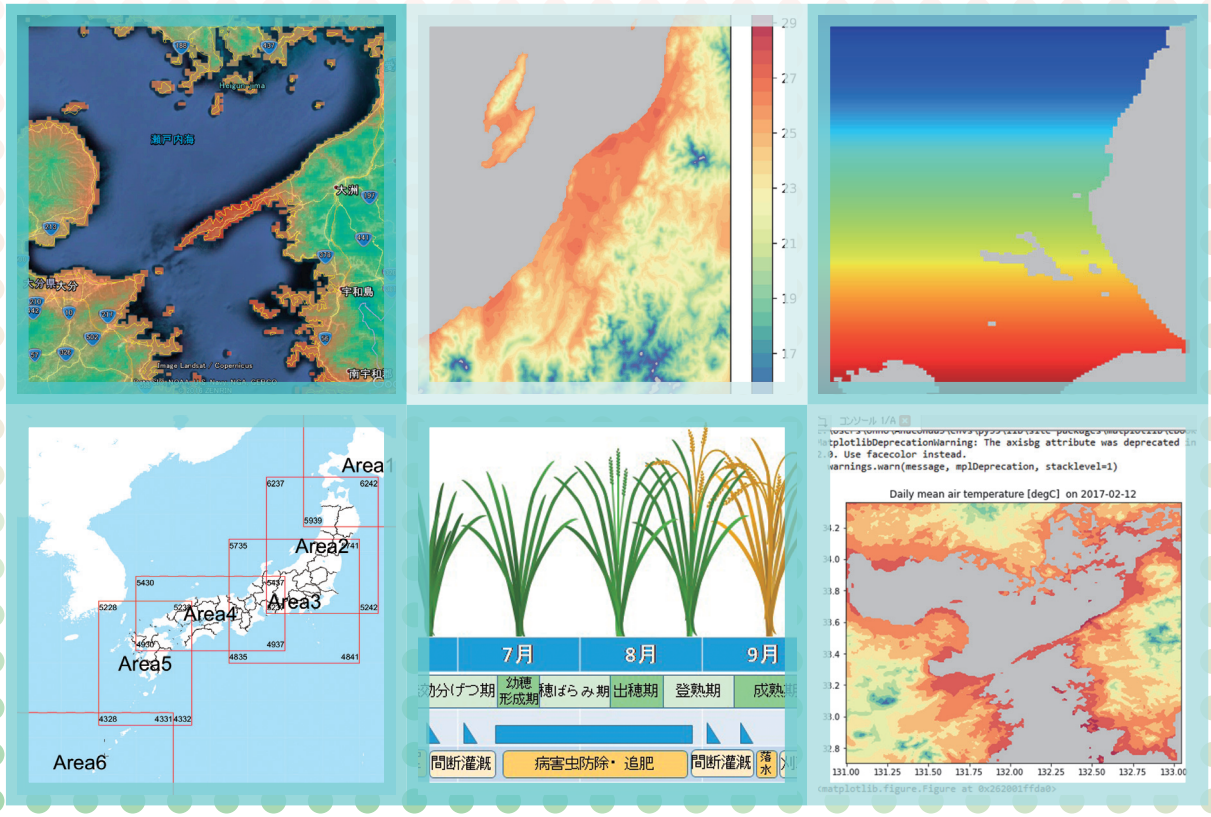
農研機構
NARO 国立研究開発法人 農業・食品産業技術総合研究機構

メッシュ農業気象データ
利用マニュアル

（2017年版）



農業環境変動研究センター

はじめに

農業に対する温暖化の影響を解析して対応策を検討したり，気象被害を評価したりするには，過去や直近の気象観測データが必要です。そして，農業気象災害を事前に把握し対策をとる技術を開発するには予測値も必要となります。このため，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）は，1980年から現在の１年先までの日別農業気象データを全国について１km のメッシュで自由に取り出して利用できる，メッシュ農業気象データシステム（The Agro-Meteorological Grid Square Data System）を運用して研究に役立てています。

近年の温暖化傾向を背景として，農業現場でも気象条件を考慮して作物を管理する必要性が増し，発育予測に基づく栽培管理や病害虫防除適期の提示，作付け適期の設定，冷害や高温障害の予測に基づく対策など，新しい栽培管理技術の開発が各地で盛んに進められるようになり，このために利用できる気象データを求める声が全国的に高まってきました。

そこで，農研機構では，2016 年から気象業務許可のもとメッシュ農業気象データ（The Agro-Meteorological Grid Square Data, NIAES）を特定向け気象予報として一定の条件で農研機構外部にも提供しています。

本マニュアルは，農研機構メッシュ農業気象データを利用するための知識と利用方法を説明するものです。

メッシュ農業気象データ利用マニュアル

(2017 年版)

(研)農業・食品産業技術総合研究機構
農業環境変動研究センター

目　次

Ⅰ　メッシュ農業気象データシステムの概要…… 1
Ⅱ　メッシュ農業気象データシステムを利用するには…… 6
　1　利用条件…… 6
　2　免責事項…… 6
　3　利用の申請・報告…… 6
　4　利用者サポート…… 6
Ⅲ　メッシュ農業気象データシステムへのアクセス…… 8
　1　データアクセスフォームを利用したデータの取得…… 8
　2　専用表計算シートを用いたデータの取得……12
　3　プログラミング言語 Python を用いたデータの取得 ……13
IV プログラミング言語 Python を用いたメッシュ農業気象データの処理 ……15
　1　メッシュ農業気象データの取得……15
　2　気象分布図の作成……19
　3　気象の時系列変化グラフの作成……23
　4　地理情報の利用……26
　5　CSV 形式のメッシュデータの読み込み ……28
　6　時系列データの書き出し……29
　7　CSV 形式の分布図の書き出し ……30
　8　メッシュ番号をキーとする属性テーブルの出力……32
　9　Google Earth 上に表示できる分布図の作成 ……34
　10　発育指数法を用いた水稲の出穂日分布図の作成……36
　11　CSV ファイルに整理した試験圃場における出穂日の予測 ……42
　12　日長の計算……46
Ⅴ　メッシュ農業気象データ利用ツールリファレンス……50
　1　AMD_Tools3 モジュール ……50
　　（1）GetMetData 関数 ……50
　　（2）GetGeoData 関数……51
　　（3）GetCSV_Table 関数 ……52
　　（4）GetCSV_Map 関数 ……53

（5）PutCSV_TS 関数……53
（6）PutCSV_Map 関数……54
（7）PutCSV_MT 関数……54
（8）PutNC_Map 関数……55
（9）PutNC_3D 関数……55
(10) PutKMZ_Map 関数……56
(11) lalo2mesh 関数……57
(12) mesh2lalo 関数……57
(13) timrange 関数……57
(14) latrange 関数……58
(15) lonrange 関数……58
2　DayLength3 モジュール……58
（1）daylength 関数……58

コラム目次

コラム 1　Python と引用符……16
コラム 2　メッシュデータはきれいに並べられたお菓子……19
コラム 3　コメントをたくさん書きましょう……22
コラム 4　リストと numpy.ndarray……24
コラム 5　「順次」「分岐」「反復」……26
コラム 6　インデントについて……34
コラム 7　if 文……37
コラム 8　for 文……39
コラム 9　日付・時刻・時間間隔の取り扱い……44
コラム 10　リスト内包表記……46
コラム 11　日の出，日の入，南中の時刻……48

付録：Python 利用環境の構築……60

Ⅰ　メッシュ農業気象データシステムの概要

メッシュ農業気象データシステムは，日別の農業気象データを提供するシステムです。システムには，日々更新される日別農業気象データ，10 年毎に更新される日別平年値データ，ならびに，若干の地理情報データが基準地域メッシュ第 3 次地域区画（以下，「3 次メッシュ」と呼ぶ。サイズは約 1 km × 1 km）を単位として全国について保持され，利用者は，これらを専用のデータ配信サーバーからオンデマンドで取得することができます。その際，気象要素や領域（または地点），期間を任意に指定することができます。

日平均気温など一般的な気象要素のほか，耕地表面の温度推定に欠かせない下向き長波放射量や農業施設の雪害に深くかかわる積雪相当水量など農業での利用が期待される 13 の気象要素を搭載し，確定値，予報値，および，平年値がシームレスに接続されたデータとして，1980 年（一部 2008 年）1 月 1 日から現在の 1 年後の 12 月 31 日までの期間について利用可能です（図 1）。

確定値データは，全国のアメダスデータを空間補間して作成されます。1 日前のデータが最新の観測値で 1 日 1 回更新されています。予報値データは，当日から 9 日先までと 10 日先まで，11 日先から 26 日先までの 3 種類の異なった方法で作成されています。当日から 9 日先までは，気象庁の数値予報モデル（メソ数値予報モデル：MSM，ならびに，全球数値予報モデル：GSM）GPV を補正して作成され，1 日 1 回最新の数値予報で 9 日先までが更新されます。10 日先の予報値データは，異常天候早期警戒情報ガイダンスに基づいて作成され，火曜日と金曜日に

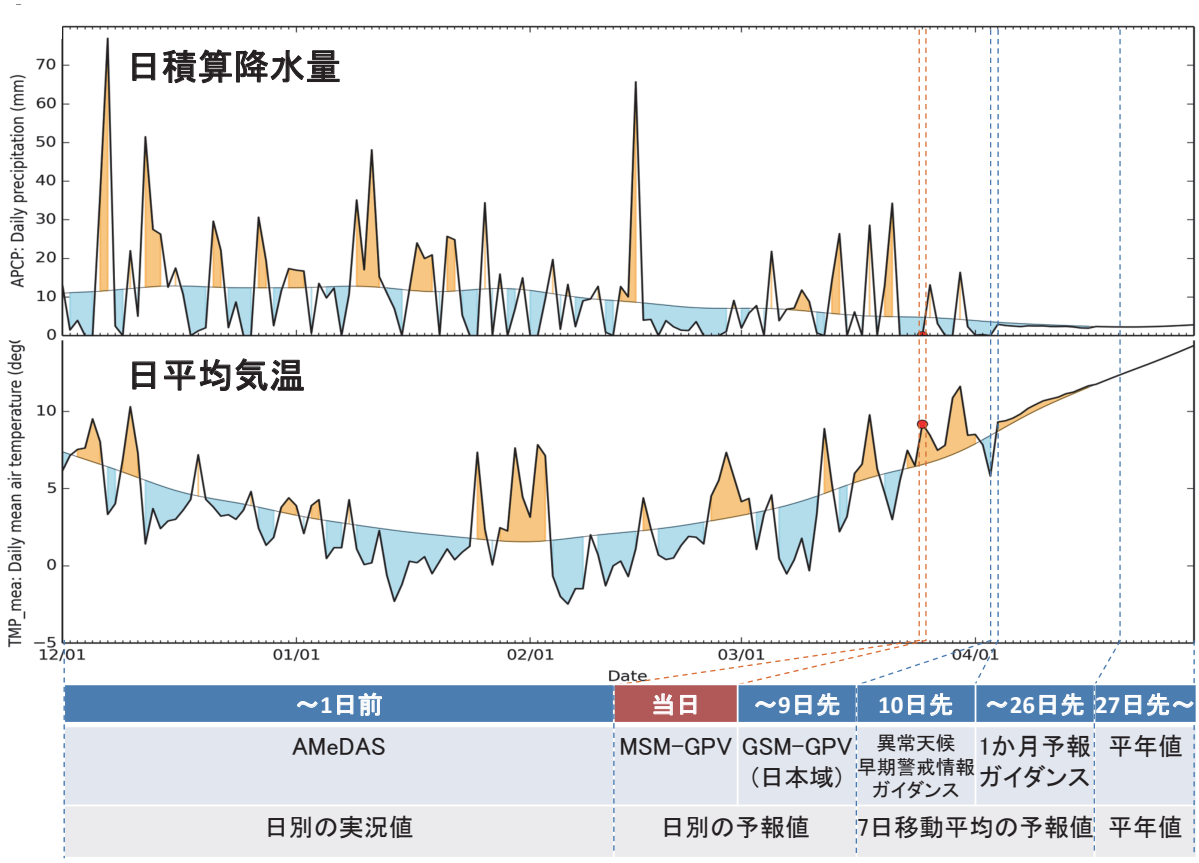


図 1．システムが作成した気象データの例（上段）と作成に使用する気象資料と処理の概要（下段）
北緯 36.06 度，東経 140.13 度における 2014 年 12 月 1 日～ 2015 年 4 月 30 日の日積算降水量と日平均気温を，2015 年 3 月 24 日（図中の「当日」）にシステムから取得して作成したグラフ。データの作成には下段に示す気象資料が使用される。

更新されます。そして，11 日から 26 日先の予報値データは 1 か月予報ガイダンスに基づいて毎週金曜日に更新されます。27 日より先については，日別平年値データがある気象要素については，その値が与えられています。湿度等平年値が得られない気象要素については，無効値が与えられています。

ここで，9 日先までの予報と 10 日先以降予報には質的な違いがあることに注意してください。すなわち，9 日先までについてはそれぞれの日について予報が行われていますが，それより先については，前後 3 日間（期間としては 7 日間）の平均値が予報日の値として与えられています。この違いは，降水量について考えると明確になります。降水量の予報値データは，9 日先までは予報に基づき 0 であったり特定の予報値だったりしますが，10 日先以降については，必ず 0 でない降水量が与えられます（図 1）。

日別平年値データは，日平均気温，日最高気温，日最低気温，降水量，日照時間，全天日射量について整備され，2011 年から 2020 年の期間について利用可能です。それより以前の期間については保持されていません。メッシュ農業気象データシステムが提供する農業気象データと平年値データの整備状況を表 1 に示します。

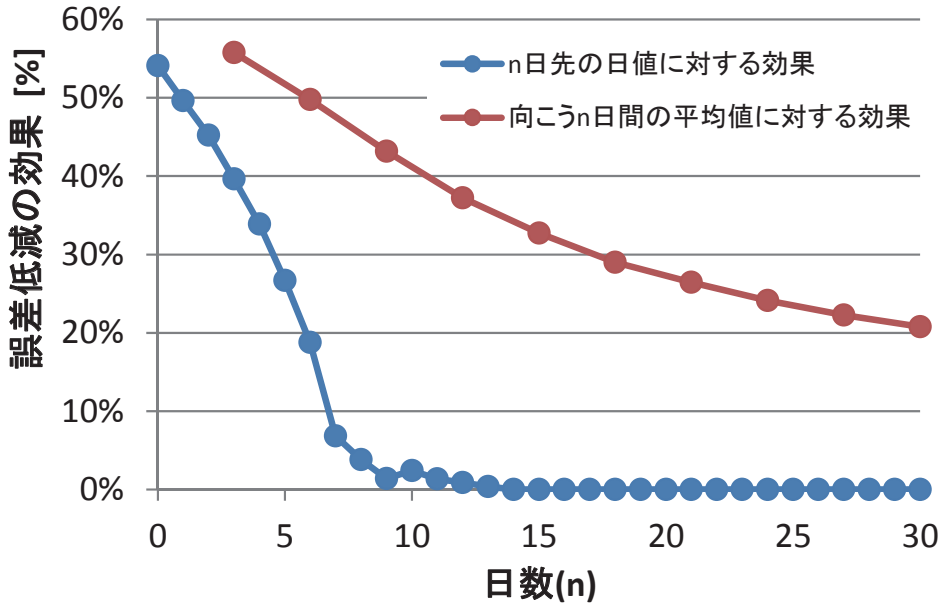
表 1．メッシュ農業気象データシステムが提供する農業気象データの一覧

気象要素	記号	単位	過去値	予報値	平年値
日平均気温	TMP_mea	℃	1980 年 1 月 1 日～前日	当日～ 26 日先	2011 年～ 2020 年
日最高気温	TMP_max	℃	1980 年 1 月 1 日～前日	当日～ 26 日先	2011 年～ 2020 年
日最低気温	TMP_min	℃	1980 年 1 月 1 日～前日	当日～ 26 日先	2011 年～ 2020 年
降水量	APCP	mm/day	1980 年 1 月 1 日～前日	当日～ 26 日先	2011 年～ 2020 年
日照時間	SSD	h/day	1980 年 1 月 1 日～前日	なし	2011 年～ 2020 年
全天日射量	GSR	$MJ/m^2/day$	1980 年 1 月 1 日～前日	なし	2011 年～ 2020 年
下向き長波放射量	DLR	$MJ/m^2/day$	2008 年 1 月 1 日～前日	なし	なし
日平均相対湿度	RH	%	2008 年 1 月 1 日～前日	当日～ 9 日先	なし
日平均風速	WS	m/s	2008 年 1 月 1 日～前日	当日～ 9 日先	なし
積雪深	SD	cm	2008 年 1 月 1 日～前日	なし	なし
積雪相当水量	SWE	mm	2008 年 1 月 1 日～前日	なし	なし
日降雪相当水量	SFW	mm/day	2008 年 1 月 1 日～前日	なし	なし
予報気温の確からしさ*	PTMP	℃	2011 年 1 月 1 日～前日	当日～ 26 日先	なし

*気温予報値の標準偏差近似値

予報は，期間が長くなればなるほど難しいものです。図 2 は，メッシュ農業気象データシステムの予報値の誤差と，「その日の気象値は平年値と同じになる」と予測した時の誤差との比を調べたものです。予報が的中する場合は 100%，平年並みとしたのと差がない場合は 0％となります。これを見ると，予報を日別に考えた場合の効果はせいぜい 7 日程度に留まることが分かります。一方，これから先 n 日間の平均気温という観点で効果を見ると 20 日よりも先まで認められることが分かります。作物の発育は個々の日の寒暖よりも積算した気温に強く影響されるので，この特性は発育を予測する場合に有利です。実際，2015 年に北海道十勝地方で小麦を対象に実施した出穂日の予測についての試験では，メッシュ農業気象データを用いた発育予測が従来の方法に比べて 15 日程度早くから正しい出穂日を予測しました（図 3）。

地理情報データは，日別農業気象データを処理する上で利用頻度が高い 4 種類が利用可能です（表 2）。平均標高は，各メッシュの平均標高で，日別農業気象データや日別平年値データを作成



誤差低減の効果は $(E_C-E_F)/E_C$ で定義
ただし、E_F は予測値誤差（RMSE）、E_C は気候値予測の誤差

図 2. 予報導入による誤差低減効果と予報日数との関係
全アメダス地点を対象に 2011 年～ 2015 年について計算した。

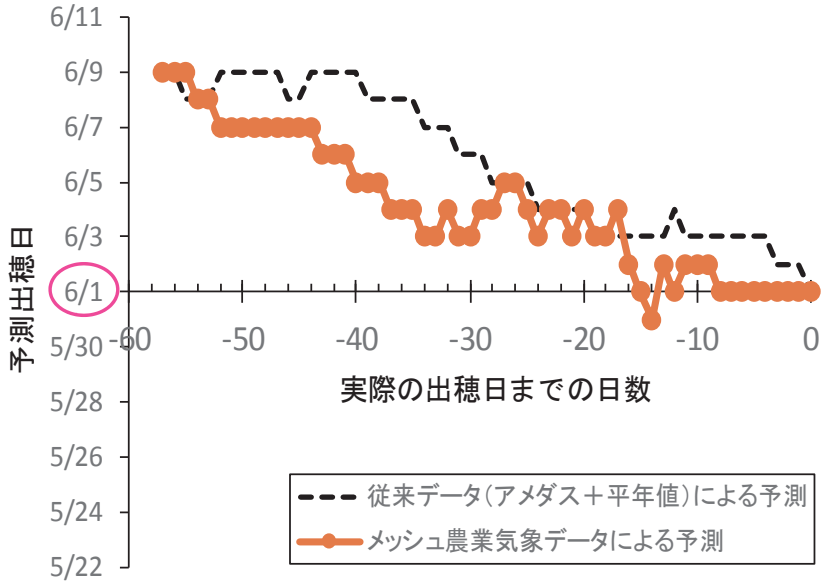


図 3. 北海道十勝地方における 2015 年産小麦の出穂日予測値の経日変化

する際に基準として用いられています（図 4a）。このデータは範囲内で標高に大きな違いがあるメッシュにおいて特定の地点の気温補正をしてより正確に推定する必要があるときなどに利用することができます。

面積は，3 次メッシュ域が持つ正確な面積です。3 次メッシュは約 1 km × 1 km の範囲ですが，緯度・経度を基準として区切られているため北ほど面積は小さくなり一様ではありません。このデータは特定領域における降水の総量を見積もるときなどに利用します（図 4b）。

土地利用比率は，各メッシュにおける土地利用比率データで，「国土交通省国土数値情報土地

利用細分メッシュデータ（平成 21 年度）」から作成されています（図 4c）。

都道府県範囲データは，「国土交通省国土数値情報行政区域データ（平成 24 年度）」をもとに 2 種類作成されています。全国一括都道府県範囲は，都道（振興局）府県に対し一意に割り振った番号を各メッシュに割り付けたもので，都道府県で色分けした日本地図のような形式です（図 4d）。複数の都道府県に所属するメッシュには，メッシュの中心点が所属する都道府県の番号が割り付けられています。都道府県と番号の対応は表 3 のとおりです。都道府県別範囲データは，都道府県毎（北海道については振興局毎）に作成され，その都道府県に含まれるメッシュに値 1，他のメッシュに無効値を与えたものです（図 4e）。一部でも当該都道府県に含まれていれば，そのメッシュには 1 が与えられています。記号は‘pref_####’で，#### には表 3 の 4 桁の数字が入ります。例として，茨城県に含まれるメッシュだけが 1，他のメッシュには無効値が入っている地理データの記号は，‘pref_0800’です。

メッシュ農業気象データシステムに搭載されるすべてのデータは，JGD2000（新測地系）に準拠しています。農研機構農業環境変動研究センターが従来提供している「アメダスデータのメッシュ化データ」が準拠する Tokyo（旧測地系）とは異なっているので混用はできません。

表 2. メッシュ農業気象データシステムが提供する地理情報の一覧

地理情報	記号	単位	備　　考
平均標高	altitude	m a.s.l.	気象庁が「メッシュ平年値 2010」を作成する際に使用したメッシュの平均標高データ。
面積	area	m^2	各メッシュの正確な面積。
－土地利用比率－			
田	landuse_H210100	%	湿田・乾田・沼田・蓮田及び田
その他の農用地	landuse_H210200	%	麦・陸稲・野菜・草地・芝地・りんご・梨・桃・ブドウ・茶・桐・はぜ・こうぞ・しゅろ等を栽培する土地とする。
森林	landuse_H210500	%	多年生植物の密生している地域とする。
荒地	landuse_H210600	%	しの地・荒地・がけ・岩・万年雪・湿地・採鉱地等で旧土地利用データが荒地であるところとする。
建物用地	landuse_H210700	%	住宅地・市街地等で建物が密集しているところとする。
道路	landuse_H210901	%	道路などで，面的に捉えられるものとする。
鉄道	landuse_H210902	%	鉄道・操車場などで，面的にとらえられるものとする。
その他の用地	landuse_H211000	%	運動競技場，空港，競馬場・野球場・学校港湾地区・人工造成地の空地等とする。
河川地及び湖沼	landuse_H211100	%	人工湖・自然湖・池・養魚場等で平水時に常に水を湛えているところ及び河川・河川区域の河川敷とする。
海浜	landuse_H211400	%	海岸に接する砂，れき，岩の区域とする。
海水域	landuse_H211500	%	隠顕岩，干潟，シーパースも海に含める。
ゴルフ場	landuse_H211600	%	ゴルフ場のゴルフコースの集まっている部分のフェアウエイ及びラフの外側と森林の境目を境界とする。
－都道府県範囲－			
全国一括都道府県範囲	pref_all60	なし	都道（振興局）府県に割り振られた番号（表 3 参照）を各メッシュに割り付けたもの。複数の都道府県に所属するメッシュには，メッシュの中心点が所属する都道府県の番号が付与されている。
都道府県別範囲	pref_####	なし	#### には表 3 の数字が入る。数字に対応する都道（振興局）府県に含まれるメッシュに値 1，他のメッシュに無効値が与えられている。一部でも当該都道府県に含まれていればそのメッシュには 1 が与えられる。

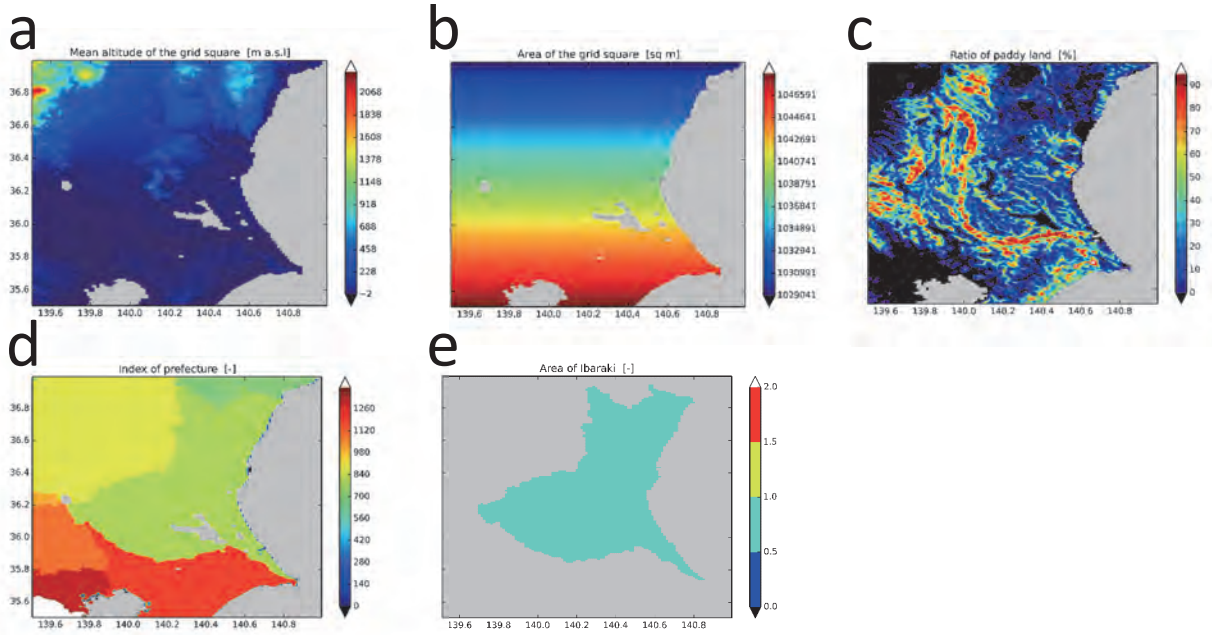


図 4. メッシュ農業気象データシステムに搭載される地理情報の例

a：メッシュ平均標高，b：メッシュ面積，c：土地利用比率，d：全国一括都道府県範囲，e：都道府県別範囲。

表 3. メッシュ農業気象システムにおける都道府県の識別番号

番号	道（振興局）	番号	都府県	番号	都府県	番号	都府県
0100	北海道	0200	青森県	1800	福井県	3400	広島県
0101	石狩振興局	0300	岩手県	1900	山梨県	3500	山口県
0102	渡島総合振興局	0400	宮城県	2000	長野県	3600	徳島県
0103	檜山振興局	0500	秋田県	2100	岐阜県	3700	香川県
0104	後志総合振興局	0600	山形県	2200	静岡県	3800	愛媛県
0105	空知総合振興局	0700	福島県	2300	愛知県	3900	高知県
0106	上川総合振興局	0800	茨城県	2400	三重県	4000	福岡県
0107	留萌振興局	0900	栃木県	2500	滋賀県	4100	佐賀県
0108	宗谷総合振興局	1000	群馬県	2600	京都府	4200	長崎県
0109	オホーツク総合振興局	1100	埼玉県	2700	大阪府	4300	熊本県
0110	胆振総合振興局	1200	千葉県	2800	兵庫県	4400	大分県
0111	日高振興局	1300	東京都	2900	奈良県	4500	宮崎県
0112	十勝総合振興局	1400	神奈川県	3000	和歌山県	4600	鹿児島県
0113	釧路総合振興局	1500	新潟県	3100	鳥取県	4700	沖縄県
0114	根室振興局	1600	富山県	3200	島根県		
		1700	石川県	3300	岡山県		

Ⅱ　メッシュ農業気象データシステムを利用するには

農研機構は，利用を許可したものに，下に示す利用条件でメッシュ農業気象データを無償で提供します。ただし，メッシュ農業気象データは，農研機構自らの研究業務のために作成しているものなので，研究の進展等に伴いデータ作成方法や提供形式が変更されることがあります。また，機器の保守やトラブル，改造に伴ってデータ更新が停滞することがあります。下に示す免責事項を承諾の上利用してください。

1　利用条件

1．農研機構は，農業分野や他の分野における，研究・開発を目的とする者に，審査に基づきメッシュ農業気象データ（以下，「このデータ」と呼ぶ。）の利用を許可します。
2．このデータを無断で他に転載したり第三者に提供したりすることはできません。
3．このデータを利用して作成した情報を販売することはできません。
4．利用者は，利用期間の終了後速やかに，利用結果を報告することとします。
5．このデータを利用して得た成果等を発表する場合は，「農研機構メッシュ農業気象データ（The Agro-Meteorological Grid Square Data, NIAES）」を利用した旨を明記してください。

2　免責事項

農研機構は，利用者がこのデータの利用によって生じた結果，ならびに，このデータが利用できないことによって生じた結果について，一切の責任を負いません。

3　利用の申請・報告

このデータの利用を希望する者は，所定の様式により利用を申請し許可を受けてください。利用できる期間は，許可日からその年度の 3 月 31 日までとします。ただし，くりかえし再申請することができます。利用申請書は郵送で提出してください。メールでは受け付けていません。

利用報告書は年度終了後提出してください。皆さんと私たちとのコミュニケーションツールの一つと考え，積極的に利用してください。特に，継続的な提供を確保するために，利用の成果も積極的に報告してください。利用報告書については，郵送のほかメール添付も受け付けます。

4　利用者サポート

メッシュ農業気象データシステムでは，利用者がメンバーとなるメーリングリストを開設しています。このメーリングリストでは，メッシュ農業気象データに関する様々な話題を利用者やメッシュ農業気象データ開発チームで議論するほか，技術的な質問も受け付けています。メーリングリストのアドレスは次の通りです。なお，HTML メールの使用とファイルの添付はできません。また，メーリングリスト登録者以外の方のメールは受け付けません。

MeshUser@ml.affrc.go.jp

また，メッシュ農業気象データシステムでは，Wiki と呼ばれる掲示板のようなホームページも開設しています。Wiki では，システムの運用状況の連絡や，新しいデータセットの紹介，データ処理のためのサンプルプログラムの提供などを行っています。ホームページの URL は，次の通りです。

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

利用報告書の提出，IP アドレスの変更など利用年度途中での申請事項の変更等，利用手続きにかかわるお問い合わせは以下のメールに連絡してください。このアドレスでは HTML メールの使用はできませんが，ファイル（3MB 以下）の添付は可能です。

MeshAdmin@ml.affrc.go.jp

Ⅲ　メッシュ農業気象データシステムへのアクセス

メッシュ農業気象データシステムは，全国を 6 つの領域に分けて各種データを管理し（図 5），さらに，時間的に継続している農業気象データについては，1 年を単位として管理しています。データ配信サーバーは，この管理単位（1 領域，1 年）から利用者が指定する範囲を取り出して配信します。

データ配信サーバーは，以下に示す 3 通りの方法でデータを配信することができます。

なお，データ配信サーバーにはアクセス制限が設定されており，利用申請の際申告した IP アドレスの PC からの接続だけが許可されます。

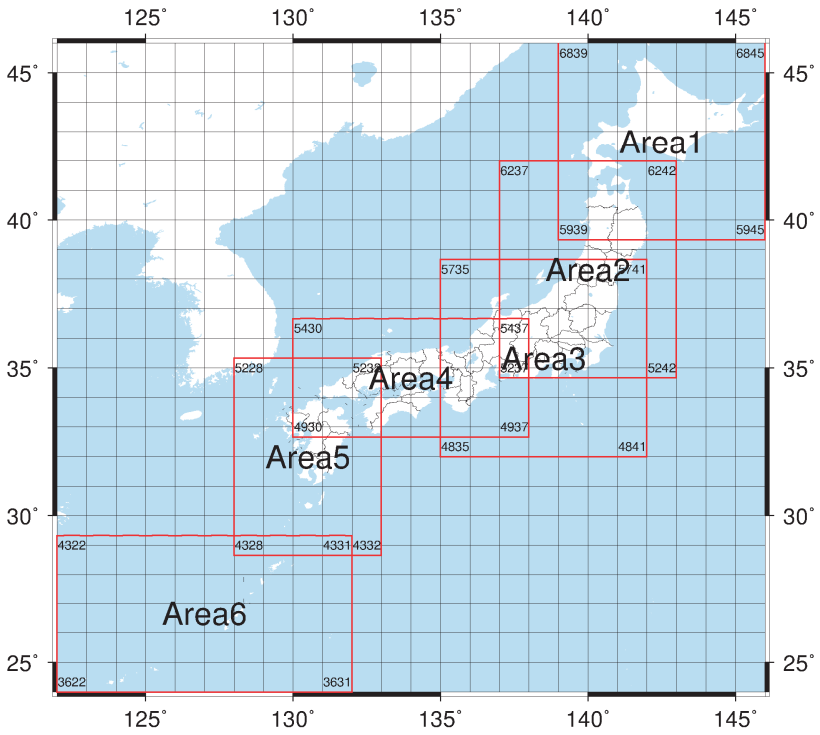


図 5. メッシュ農業気象データ配信サーバーが提供するデータの領域（赤線）
領域四隅の数字は，それぞれに位置する基準国土 1 次メッシュ番号を示す。

1　データアクセスフォームを利用したデータの取得

メッシュ農業気象データ配信サーバーは簡素なホームページを持っており，ここに置かれているデータセットアクセスフォーム（Dataset Access Form）を通してデータを要求して取得することができます。

以下に，北緯 34.5 ～ 36.0 度，東経 139.0 ～ 140.5 度（東京湾の周辺）の日平均気温を 2013 年 1 月 1 ～ 10 日について CSV ファイルで取り出す方法を例として示します。

メッシュ農業気象データ配信サーバーホームページ（**http://mesh.dc.affrc.go.jp/opendap/**）を Web ブラウザでアクセスすると，図 6 のようなトップページが表示されます。東京湾周辺は Area3 に含まれ取得するのは 2013 年のデータなので，トップページのリストから，**Area3**，**2013**，と順にクリックすると，**AMD_Area3_APCP.nc** などのリスト表示がされます。この中から表 1 に掲載される気象要素の記号を含む項目を選んでクリックすると，それに対するデータセットアクセスフォーム（図 7）が開きます。この例で取得するのは日平均気温なので，**AMD_**

Area3_TMP_mea.nc のフォームを開きます。フォーム中程に表示される見出し **Variables** の右側の **TMP_mea** の左脇の小さなチェックボックスをチェックすると，空欄だったテキストボックスに，0：1：364，0：1：799，0：1：559 という数字が表示されます。これらの数字は，Area3 領域の 2013 年の日平均気温データが，日付方向に 365 層，緯度方向に 800 メッシュ行，経度方向に 560 メッシュ列の大きさであることを示しています。中央の数字 1 は気にしないでください。取得するデータの時空間範囲の指定は，これらの数字を書き直すことで行います。1 月 1 ～ 10 日は，最初の 10 層に相当するので「0：9」です。コンピューター特有の作法で，1 番最初は 1 でなく 0 を指定します。緯度 34.5 ～ 36.0 度の緯度範囲に対応するメッシュの行位置は 300：480 です。そして，経度 139.0 ～ 140.5 度に相当するメッシュの列位置は 320:440 です。テキストボックスのなかの数値をこれらに書き換えた後，頁上部の **[GET ASCII]** ボタンをクリックすると，目的のデータが CSV ファイルでダウンロードできます。

日付，緯度，経度とメッシュの層，行，列との対応は，Excel ファイル「**AMGSD の領域 .xls**」のワークシートで調べることができます（図 8）。このワークシートのセル B39：B40 に知りたい緯度と経度を十進小数表記で入力すると対応する緯度方向のメッシュ番号（lat），経度方向のメッシュ番号（lon）が計算されます。そして，B50 に日付を入力すると，time の要素番号が計算されます。このファイルは，利用者 Wiki から入手することができます。

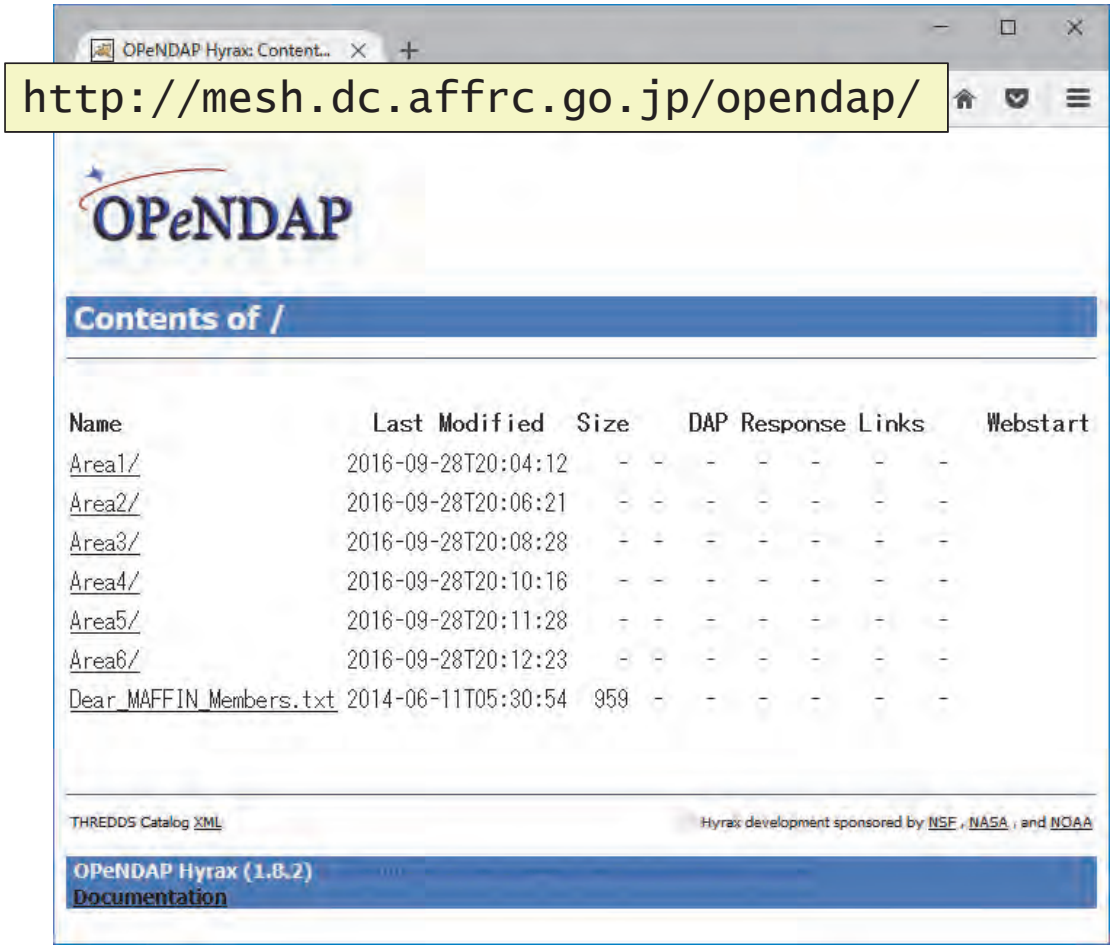


図 6.　メッシュ農業気象データ配信サーバーのトップページ
このホームページへは利用登録した IP アドレスの PC からのみアクセスできる。

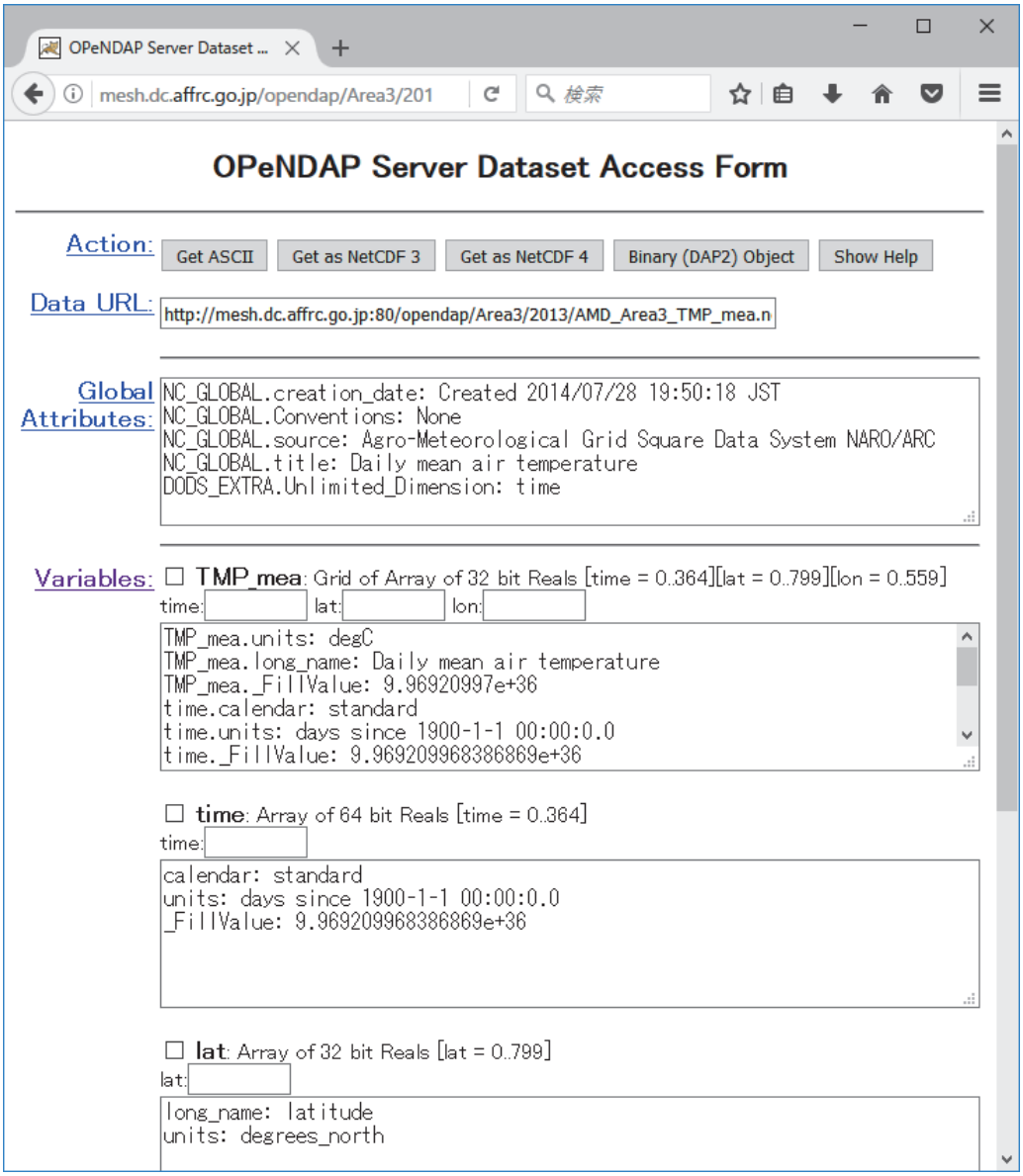


図 7. メッシュ農業気象データ配信サーバーホームページのデータセットアクセスフォーム
テキストボックスに必要とする期間や範囲を入力してデータ配信を要求する。

ブラウザの設定によっては，ファイルがダウンロードできずに，新しいページが開いて，文字が全面に並んだ画面が表示されることがありますが，その場合には，ページ上でマウスを右クリックし，「名前を付けてページを保存」を選びます。メニューから「ファイル」，「名前を付けて保存」とするブラウザもあります。保存の際，ファイル名の拡張子を変更して **AMD_Area3_TMP_mea.nc.csv** として保存してください。このファイルの中身を確認してみましょう。表計算ソフトで開き，ウインドウ右下の表示倍率スライドバーを左いっぱいに動かして縮小表示すると，南北が逆転した房総半島～伊豆大島が縦に 10 枚繋がっている様子を確認することができます（図 9）。

メッシュ農業気象データ配信サーバーは，海や湖沼など，未定義であることを実数で示す場合に，値 9.96921E+36 を使用します。図 9 の「地図」で，海上のメッシュに相当するセルにはこの値が代入されていることを確認してください。

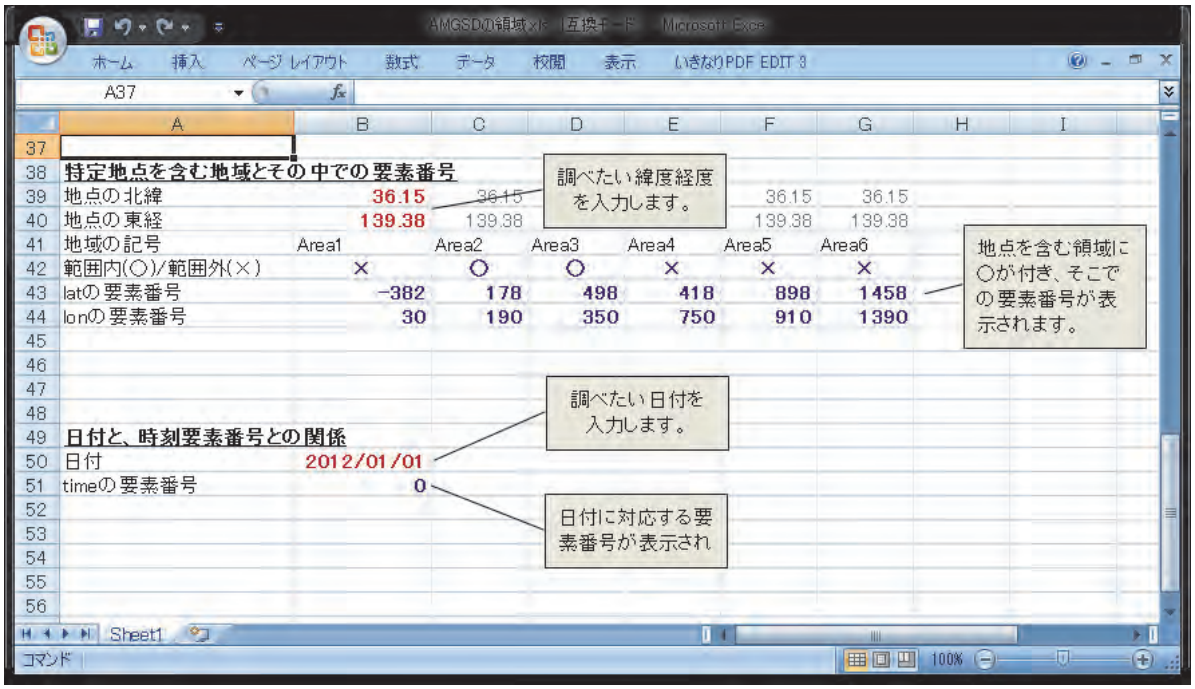


図 8.　日付，緯度，経度から，メッシュの層，行，列を求めるワークシート「AMGSD の領域 .xls」
利用者 Wiki から入手することができる。

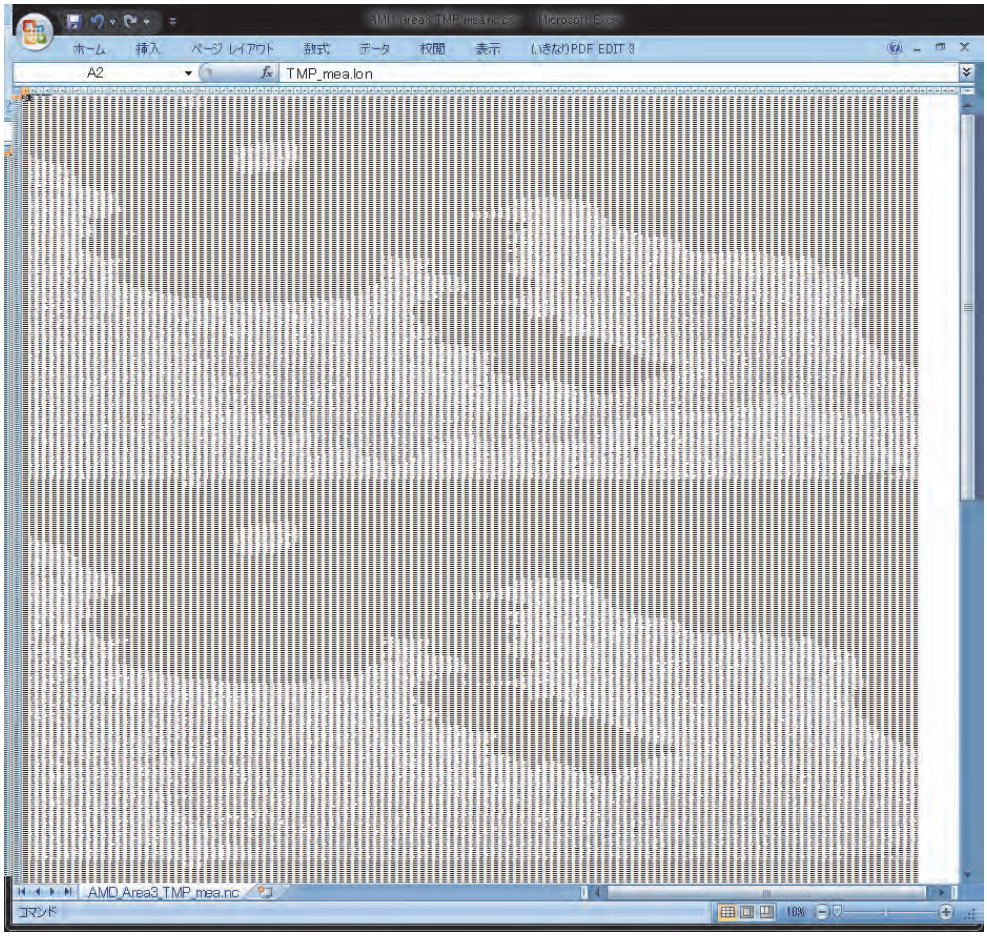


図 9.　取得した CSV ファイルを表計算ソフトで開いたところ
南北が逆転した房総半島～伊豆大島が縦に 10 枚繋がっている様子が確認できる。

このホームページを閉じる前に，見出し **Data URL** の右側の文字列（**http://mesh.dc.affrc.go.jp/opendap/Area3/2013/AMD_Area3_TMP_mea.nc.ascii?TMP_mea[0:9][300:480][320:440]**）をメモ帳等にコピーしておいてください。そして，ホームページを閉じたのちもう一度開いて，URL にこの文字列を入力してみてください。先ほどと全く同じデータを取得することができます。したがって，同じ場所の最新予報を繰り返し取得する場合には，ホームページのデータアクセスフォームを使用する必要はなく，ブックマーク等に記録した URL をブラウザで開くだけでデータが取得できます。

2　専用表計算シートを用いたデータの取得

利用者 Wiki から入手できる「メッシュ農業気象データ _ ポイント単要素抽出 _ver2.0.1.xlsm」（図 10）を使用すると，表計算ソフト Microsoft Excel を用いて特定メッシュにおける 1 年分の気象データをきわめて簡単に取得することができます。このブックには VBA マクロが組み込まれているので，ブックを開いたら，まず，［コンテンツの有効化］ボタンをクリックしてマクロを実行可能とします。

シート上方に着色されたセルがあり，ここで取得するデータの気象要素（「データ要素」），「データ取得年」，「地点の北緯」，「地点の東経」を設定します。「データ要素」は，プルダウンメニューになっているので一覧の中から選択します。また，緯度と経度を入力すると，その位置がシート左側の地図上に菱形で表示され指定メッシュの大まかな位置が確認できるようになっています。

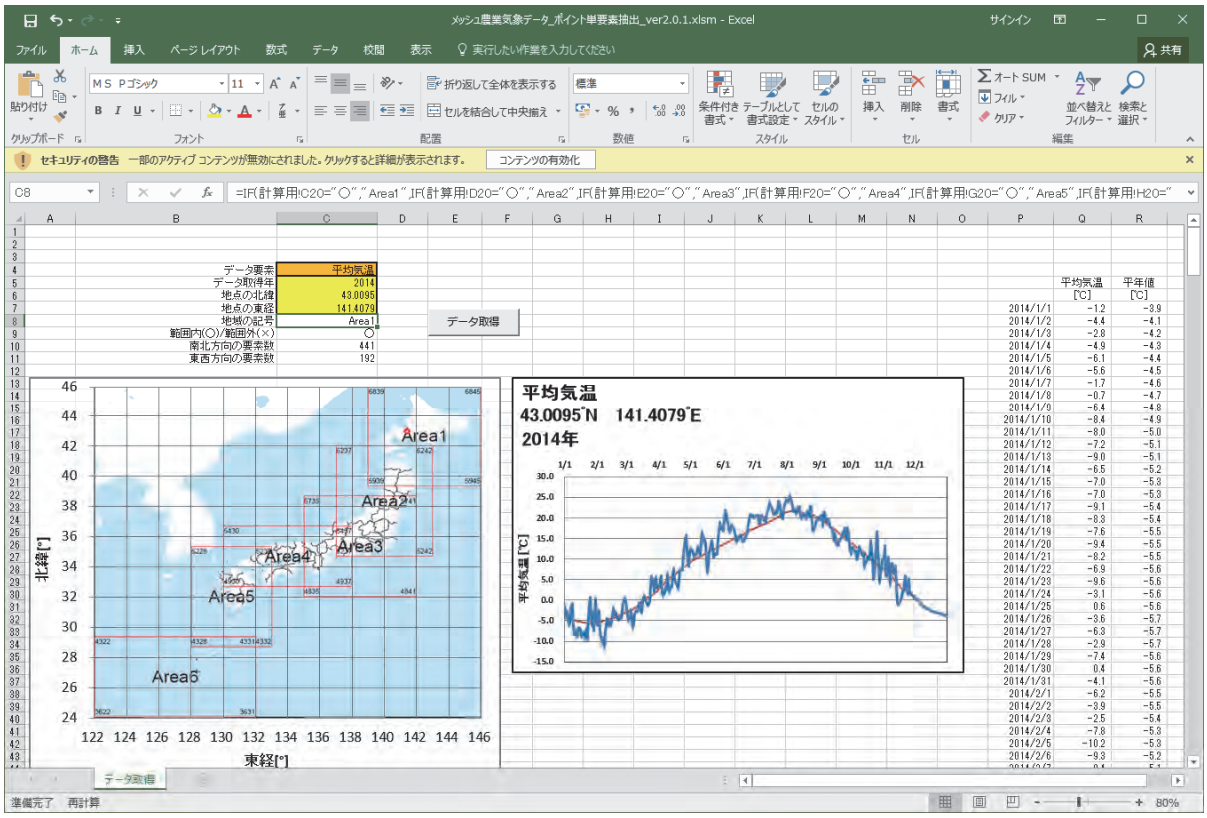


図 10.　データ取得用 Microsoft Excel ブック「メッシュ農業気象データ _ ポイント単要素抽出 .xlsm」の画面

指定が終了したら，［データ取得］ボタンをクリックします。しばらくするとグラフが表示され，その横に日別値と平年値が表示されます。年次や気象要素によっては，平年値が利用できないことがあります。その場合は，平年値のセルは黒色に着色されます。

このほかに，全 13 種類の気象要素を一括して取得する「メッシュ農業気象データ _ ポイント全要素抽出 _ver2.0.1.xlsm」も用意されており，利用者 Wiki から入手することができます。

3　プログラミング言語 Python を用いたデータの取得

気象データには取得後，必ずなにがしかの処理が施されます。農業分野では表計算ソフトを使用して処理や解析を行う例が多いようですが，メッシュ農業気象データは日時，緯度，経度，の変数を持つ三次元データなので，表計算ソフトで行える処理には限界があります。たとえば，ある日にコシヒカリを県下一斉に移植したとした時に予想される出穂日の分布図は農業指導に便利な参考図ですが，このような図を表計算ソフトで描くことはできません。しかし，プログラミング言語を使用すればこのような図を簡単に描くことができます。

ここでは，新潟県周辺における 2016 年 8 月の平均気温分布図（図 11）を作成するプログラム（図 12）を例に説明します。このプログラムでは，読み込むべき気象要素や日付範囲，緯度経度範囲を 6 行目から 8 行目で指定し，それらをもとに 9 行目でデータ配信サーバーからデータを取得し，その結果を 11 行目で日付で平均し月平均値とします。そして，分布図の作成と出力は 13 行目以降で行っています。

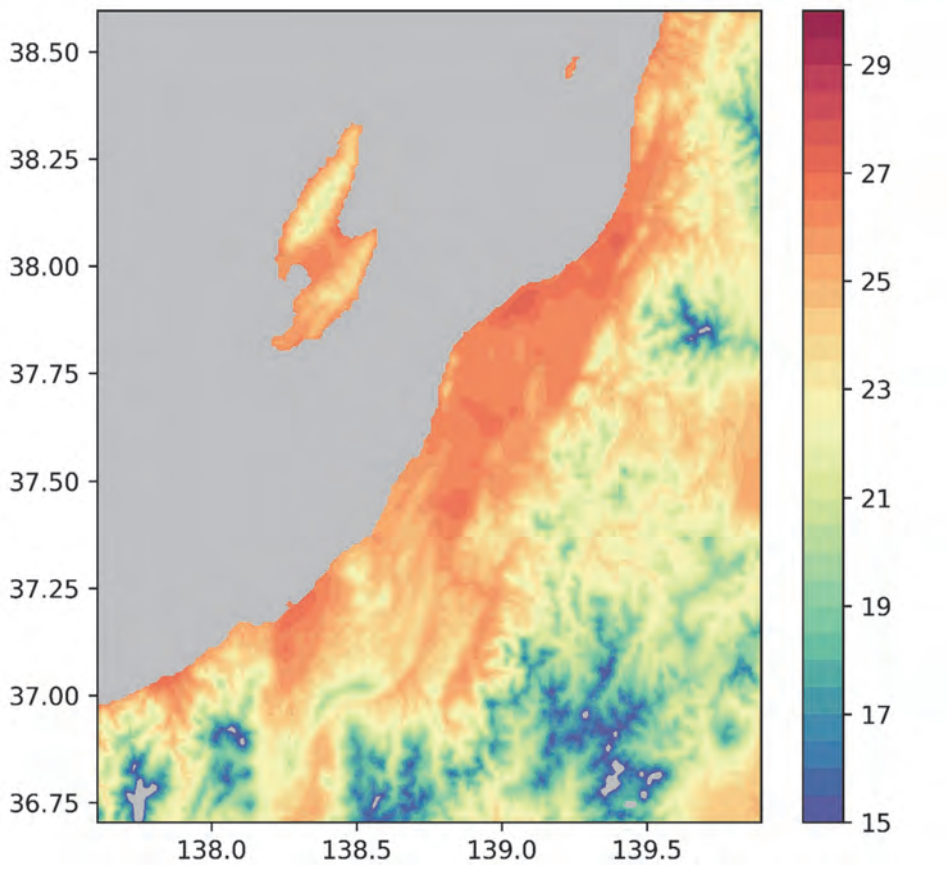


図 11．Python プログラムで作成した新潟県周辺における 2016 年 8 月の平均気温分布図
画面に表示された図は，画像ファイルにも出力される。

このようなプログラムを自力で書くのは大変ですが，プログラムに書かれている文の意味を類推するのはそれほど難しくはないのではないでしょうか。メッシュ農業気象データシステムでは，このような処理ができるサンプルプログラムを Python（パイソン）で作成して利用者に提供しています。初心者にとってプログラミング言語はたいへん敷居が高いものですが，Python は初心者にも比較的わかりやすい言語ですので，これを機会にぜひ挑戦してみてください。

プログラム

```
# -*- coding: utf-8 -*-
import AMD_Tools3 as AMD
import numpy as np
import matplotlib.pylab as plt

element = "TMP_mea"                          #気象要素の指定
days = [ "2016-08-01", "2016-08-31" ]        #期間の指定
area = [ 36.7, 38.6, 137.6, 139.9]           #領域の指定
Msh,tim,lat,lon = AMD.GetMetData(element,days,area) #データ取得

Msh = np.mean(Msh, axis=0) #平均値の計算

sclmax  = 30.0               #色スケール最大値の設定
sclmin  = 15.0               #色スケール最小値の設定
sclint  =  0.5               #色スケール刻みの設定
levels = np.arange(sclmin, sclmax+sclint, sclint)
fig = plt.figure(num=None, figsize=(6, 6))
plt.axes(axisbg='0.8')       #背景を灰色に
cmap = plt.cm.Spectral_r     #色調の指定
CF = plt.contourf(lon, lat, Msh, levels, cmap=cmap) #分布図描画
plt.colorbar(CF)             #色スケール描画
plt.savefig('result.png', dpi=600) #図の出力
plt.show()
```

図 12.　気温分布図を作成する Python プログラムのスクリプト

このプログラムは，分布図を画面に表示するだけでなく，画像ファイルも作成する。

Ⅳ　プログラミング言語 Python を用いたメッシュ農業気象データの処理

本章では，幾つかのサンプルプログラムを実行したり編集したりしながら，Python を用いたメッシュ農業気象データの処理を説明します。説明は，Spyder と呼ばれる Python の統合開発環境での操作を前提に行います。付録に，Spyder を利用可能とするまでの手順を説明していますので，それを参照してあらかじめ環境を整えておいてください。また，この章で使用するサンプルプログラム一式（PythonWorks.zip）が利用者 Wiki の「初めて利用される方へ」から入手できるので，あらかじめダウンロードしてデスクトップに展開しておいてください

なお，サンプルプログラムには別途補足説明が付加されていることがあります。このため，マニュアルの図や本文での説明とは番号が一致しないことなどがあることをご了承ください。

1　メッシュ農業気象データの取得

利用者 Wiki で配布する **sample_GetMetData-a.py** を実行しながら，Python プログラムにおけるメッシュ農業気象データの取得を説明します。このプログラムは，愛媛県西宇和郡伊方町の佐田岬半島の先端付近の日平均気温を，2016 年 1 月 1 日から 1 月 3 日までの 3 日間について取得して配列変数に格納した後，それを表示するものです。

Spyder を起動して右上のフォルダマークをクリックし，デスクトップにダウンロードした **PythonWorks** を選択し，作業フォルダに設定します。次に，右上ペインのタブを「ファイルエクスプローラー」にし，ファイルの一覧を表示させます。そして，リストの中から **sample_GetMetData-a.py** を選択してダブルクリックし，エディタペインに開きます。そのうえで，ツールバーの三角ボタンをクリックして実行します。プログラムが実行されると，右下ペインに実行結果が出力されます。

それでは，プログラムの中身を順に説明します。1 行目に，灰色の文字が記されていますが，これはおまじないと考えて必ず付けてください。2 行目と 42 行目に二重引用符が 3 回繰り返された行があります。Python では，3 回連続した二重引用符で挟まれた範囲はコメントと理解し，プログラム実行の際には無視します。コンピューターにとって意味のある最初の文は 43 行目です（図 13）。この文は，メッシュ農業気象データシステムの利用ツールのコレクションを使用する宣言です。この文もおまじないと思って必ず記述するようにしてください。この利用ツールの実体は，作業フォルダ内に置かれているファイル「**AMD_Tools3.py**」です。

メッシュ農業気象データは，51 行目の文によりデータ配信サーバーから取得されます。ここに使われているのは，メッシュ農業気象データ利用ツールの一つである **GetMetData** 関数です。関数は function の日本語訳です。“関数”よりは“機能”と訳した方が理解しやすいですが，慣例に従って関数と記載します。**GetMetData** 関数は，**AMD_Tools3** のなかに入っているので，最初に「**AMD.**」を付けて存在場所を示しています。この関数を動作させるには，読むべき気象要素，期間，領域を指定する必要があります。これは，46 ～ 48 行目で，気象要素を日平均気温（表 1 参照），期間を 2016 年 1 月 1 日～ 1 月 3 日，領域を北緯 33.32 度 / 東経 131.99 度～北緯 33.37 度 / 東経 132.05 度と指定しています。この領域には，6（緯度方向のメッシュ行数）× 5（経度方向のメッシュ列数）= 30 個の 3 次メッシュが含まれます。**GetMetData** 関数は，この指定に基づいて，データ配信サーバーから 4 種類のデータを取得し，左辺の変数 **Msh**，**tim**，**lat**，**lon** に格納します。

それでは，取得されたデータを順次確認しましょう。51 行目の文の左辺の最初の変数 **Msh** には，気象値本体が格納されます。54 行目の文によりその内容が右下ペインに表示されます。こ

れを見ると，**Msh** という一つの変数に 3（日）× 6（行）× 5（列）＝ 90 個のメッシュデータが保持されていることが分かります。そして，図 9 と同様に南北が反転した地図が 3 日分繰り返されたように格納されていることが分かります。ここで，**nan** は，Python で使われている無効値です。

数字の並びをよく見ると，[] 括弧が 3 重の入れ子になって付随していることが分かります。これは，メッシュ農業気象データが，日付，緯度，経度を変数とする三次元データであることに

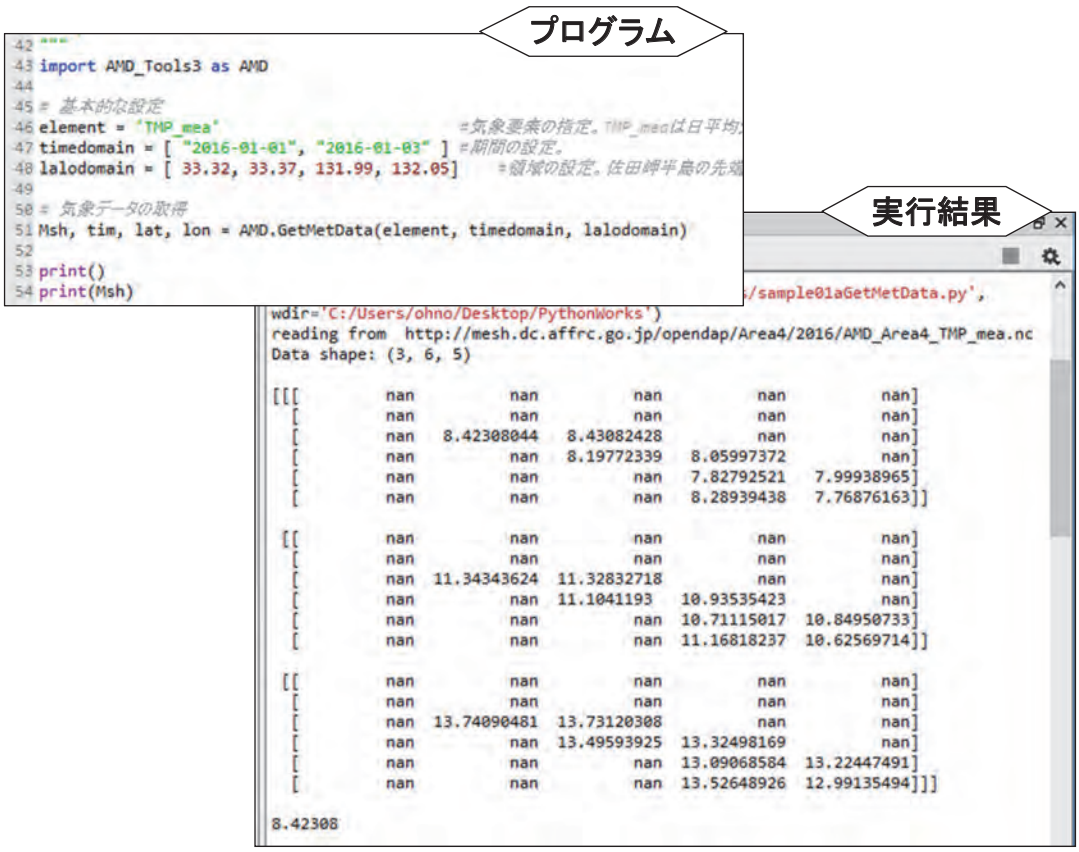


図 13. サンプルプログラム sample_GetMetData-a.py の 42 行～ 54 行目（左上）と対応する実行結果（右下）

コラム 1

Python と引用符

Python では，値が文字列であることを示すときにそれを引用符で括ります。この時，引用符は一重引用符「'」と二重引用符「"」の両方が使用できます。以下のように使うことができます。

print("today") →　today
print('today') →　today
print("That's a good idea!") →　That's a good idea!
print('I say "Hello".') →　I say "Hello".

このほか，Python では，二重引用符を 3 回連続して書くことで，複数行にわたるコメントを記述することができます。これについてはコラム 3 で詳しく説明します。

対応します。

変数 **Msh** のように，複数の数値を保持する変数を配列変数と呼びます。Python では，リストと呼ぶこともあります。配列変数やリストは，細かな仕切りのある菓子箱や重箱を想像すると理解が簡単です。この例の場合は，縦（緯度）6 列 / 横（経度）5 列に仕切られた箱が 3 段（日）重なっていると考えることができます。配列変数の中の特定の場所の値が必要な時には，**Msh[0,2,1]** のように記述します。メッシュ農業気象データシステムでは，最初の数字は日付，2 番目の数字は緯度，3 番目の数字は経度の場所を示すことになっていて，**Msh[0,2,1]** と指定した場合は，1 日目，南端から北に向かって 3 列目，西端から東に数えて 2 列目のメッシュ値が指定されます。日常生活で考えれば **[1,3,2]** と指定したいところですが，Python では順番を指定するときに 1 ではなく 0 から数え始める決まりになっているためこのように指定します。

51 行目の文の左辺 2 番目の変数 **tim** には，日付のリストが格納されます。91 行目の文によりその内容が右下ペインに表示されます（図 14）。メッシュ農業気象データシステムは，日付や時刻を Python の「**datetime** オブジェクト」というもので取り扱っていて，**tim** は 3 つの **datetime** オブジェクトの配列変数です。**datetime** オブジェクトは単純な数値ではないので面倒な一面もありますが，格納されている年月日が一目でわかり，日付や時刻に関する高度な取り扱いが可能です。51 行目の文の左辺 3 番目と 4 番目の変数 **lat** と **lon** には，それぞれ，メッシュの中心緯度と中心経度が格納されます。

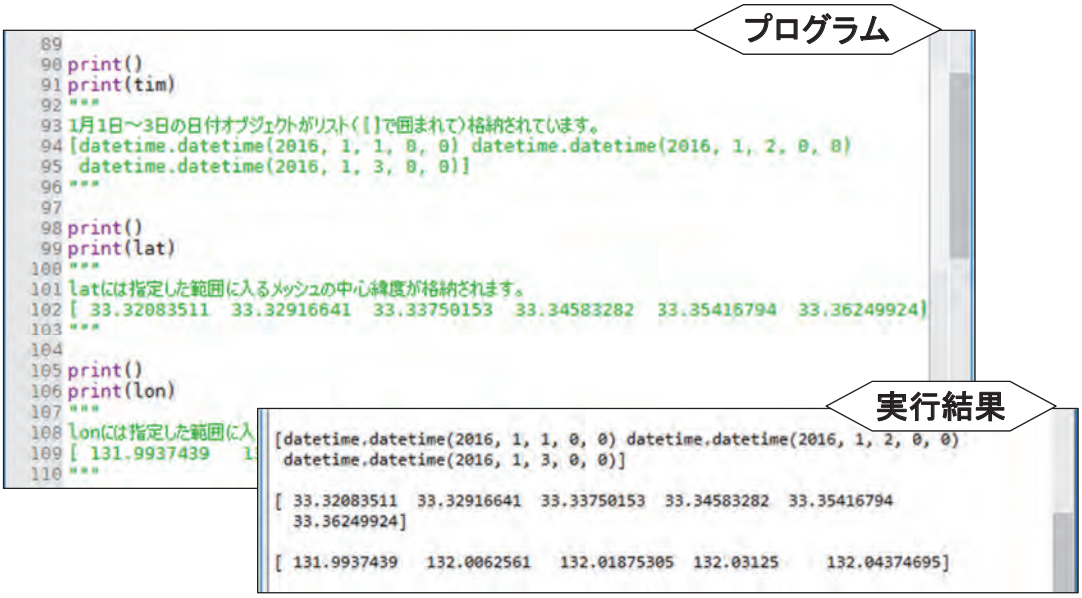


図 14. サンプルプログラム sample_GetMetData-a.py の 89 行～ 110 行目（左上）と対応する実行結果（右下）

特定の日のメッシュのデータを取得するときは，取得期間の開始日と終了日に同じ日付を設定して **GetMetData** 関数に与えます。同様に，特定メッシュの値だけを取得する場合は領域を指定する緯度と経度に同値を指定して与えます。例えば，2016 年 1 月 1 日だけのデータを取得するには，以下のようにします。

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

取得されたデータは特定の日のデータなので，実質的には二次元（緯度×経度）ですが，

GetMetData 関数は，データを常に三次元用の入れ物に入れて返すので，Msh[0,2,1] で参照しなければなりません。不要な次元を落とし Msh[2,1] で参照できるようにするには，以下の文を実行します（図 15）。

```
Msh = Msh[0, : , :]
```

GetMetData は，最新のメッシュ農業気象データをサーバーから取得しますが，引数に「cli=True」を追加すると，最新データではなく，対応する平年値を取得します。また，引数に「namuni=True」を追加すると，気象要素の正式名称と単位を追加で取得します（図 16）。

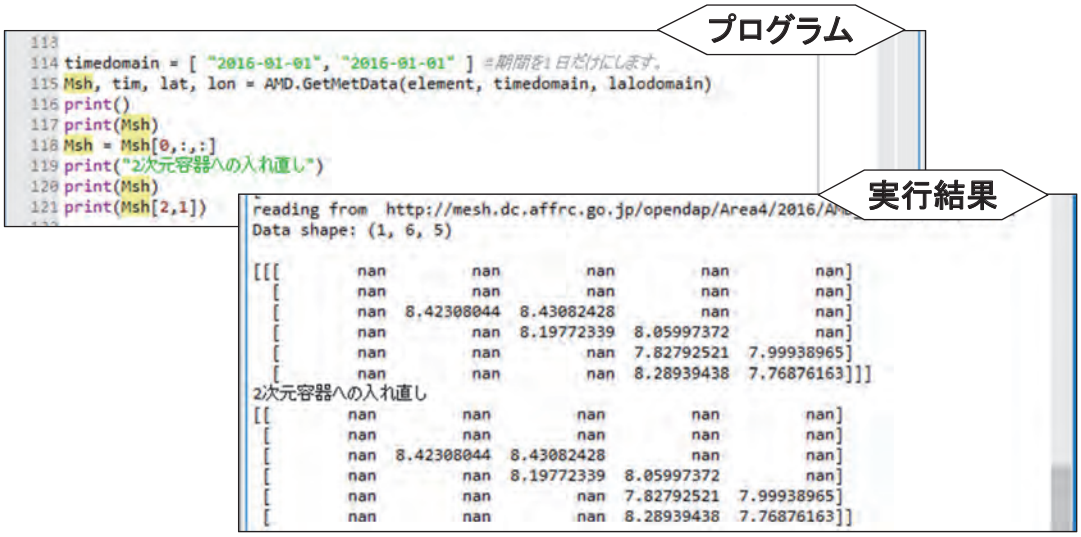


図 15. サンプルプログラム sample_GetMetData-a.py の 113 行～ 121 行目（左上）と対応する実行結果（右下）

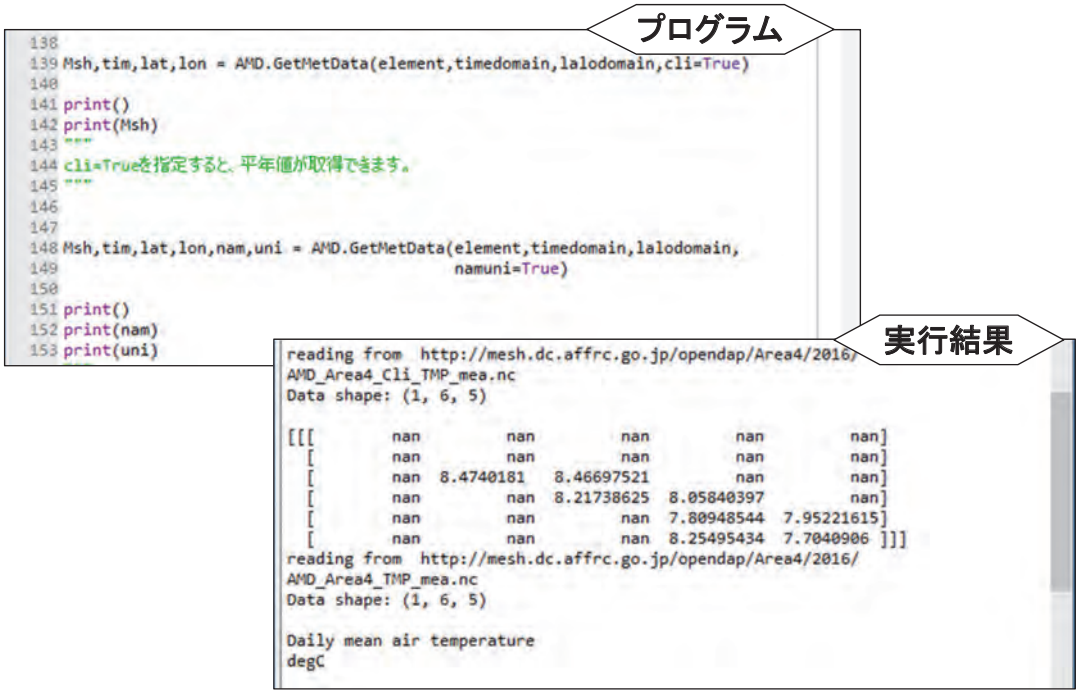


図 16. サンプルプログラム sample_GetMetData-a.py の 138 行～ 153 行目（左上）と対応する実行結果（右下）

コラム 2

メッシュデータはきれいに並べられたお菓子

本文において，配列変数を「細かな仕切りのある菓子箱や重箱」で例えました。この例えでは，特定の日における特定のメッシュの気象要素の値（気温で言えば「23.0℃」など）が重箱に並べられたお菓子に相当します。お菓子の場合は，個々のお菓子が箱のどこに置かれているかはあまり重要ではありませんが，気象データの処理においては，どのデータがどこに格納されているかはきわめて重要です。このため，メッシュ農業気象データシステムでは，「お菓子」を箱に詰める順番と「重箱」における場所を指定する方法とを一つに決めています。すなわち，「お菓子」は，古いほど下の段に，南のほど手前の列に，西のほど左の列に詰めることにし，「重箱」における場所については，下から数えた段数，手前から数えた列数，左から数えた列数をこの順で示して指定することにしています。

話をプログラミングに戻すと，**GetMetData** 関数によってメッシュ農業気象データが格納された配列変数 **Msh** の特定のデータは **Msh[i, j, k]**，と書くことによって指定することができます。配列変数の要素を指定するために使用する **i** や **j** のことを添え字「そえじ」と呼びます。そして，これによって指定されるのは，取得期間の初日から数えて **i** 日目における，南端から **j** 番目，西端から **k** 番目のメッシュのデータです。

メッシュデータのグラフや分布図を描くときには，**i** 番目の日付や **j** 番目の緯度，**k** 番目の経度が，実際のところ何日であり何度であるかを知っておく必要があります。これらの情報は，**GetMetData** 関数から戻される 3 つの配列変数 **tim**，**lat**，**lon** に格納されています。つまり，気象データ **Msh[i, j, k]** は，**tim[i]** 日，北緯 **lat[j]** 度，東経 **lon[k]** 度のデータです。

ある領域の特定の日のデータや特定メッシュの 1 年分のデータなど，メッシュ農業気象データは，取得範囲の設定によっては必ずしも三次元ではありません。しかし，**GetMetData** 関数は，データの実質的な次元とは無関係に三次元の配列で結果を返します。例えば，期間が 1 日しかないメッシュデータが **Msh** に返されたとして，南から **j** 番目，西から **k** 番目のデータをこれから取り出すときは，**Msh[j, k]** ではなく **Msh[0, j, k]** としなければなりません。一段しかない重箱の中のお菓子を毎回「下から一段目の」と言って指定するような感じです。これは時に不都合を生じます。そのような時は，「お菓子」を二次元の配列に入れなおすとよいでしょう。以下のようにすると，配列変数名を変えずに次元を変更することができます。

```
Msh = Msh[0, :, :]
```

1 番目の要素なのに 0 と書かれていて変と思われたかもしれません。Python には，配列の添え字に関していくつか決まりがあります。まず，添え字は 1 からではなく 0 から数え始めます。初日・南端・西端のメッシュデータは，**Msh[1, 1, 1]** ではなく **Msh[0, 0, 0]** です。そして，添え字に使用する数字は整数でなければなりません。ただし，例外としてコロン「：」を使用することができます。これは「その間」を意味します。したがって，南から **j** 番目，西から **k** 番目のメッシュにおける最初の 10 日分のデータを **Ta** から取り出したいときは **Ta[0:10, j, k]** と指定します。0 ～ 9 までを取り出すのに 0：10 と書くのがこれまた気持ち悪いですが，決まりとして覚えてください。コロンを挟む前後の数字はなくても構いません。その場合は，それぞれ「初めから」，「最後まで」と解釈されます。コロンだけの場合は，「全部」となります。このほか，妙な決まりとして負の整数があります。これは後ろから数えて何番目かを示します。例として，**-1** は一番後ろの要素を示します。

2　気象分布図の作成

利用者 Wiki で配布する **PythonWorks** にある **sample_GetMetData-b.py** を例に，気象分布図の作成手法を説明します。このプログラムは，2017 年 2 月 12 日における日平均気温の分布図を佐田岬半島の先端を中心とする地域について描画するものです（図 17）。プログラムの実行方法は，「1 メッシュ農業気象データの取得」を参照してください。

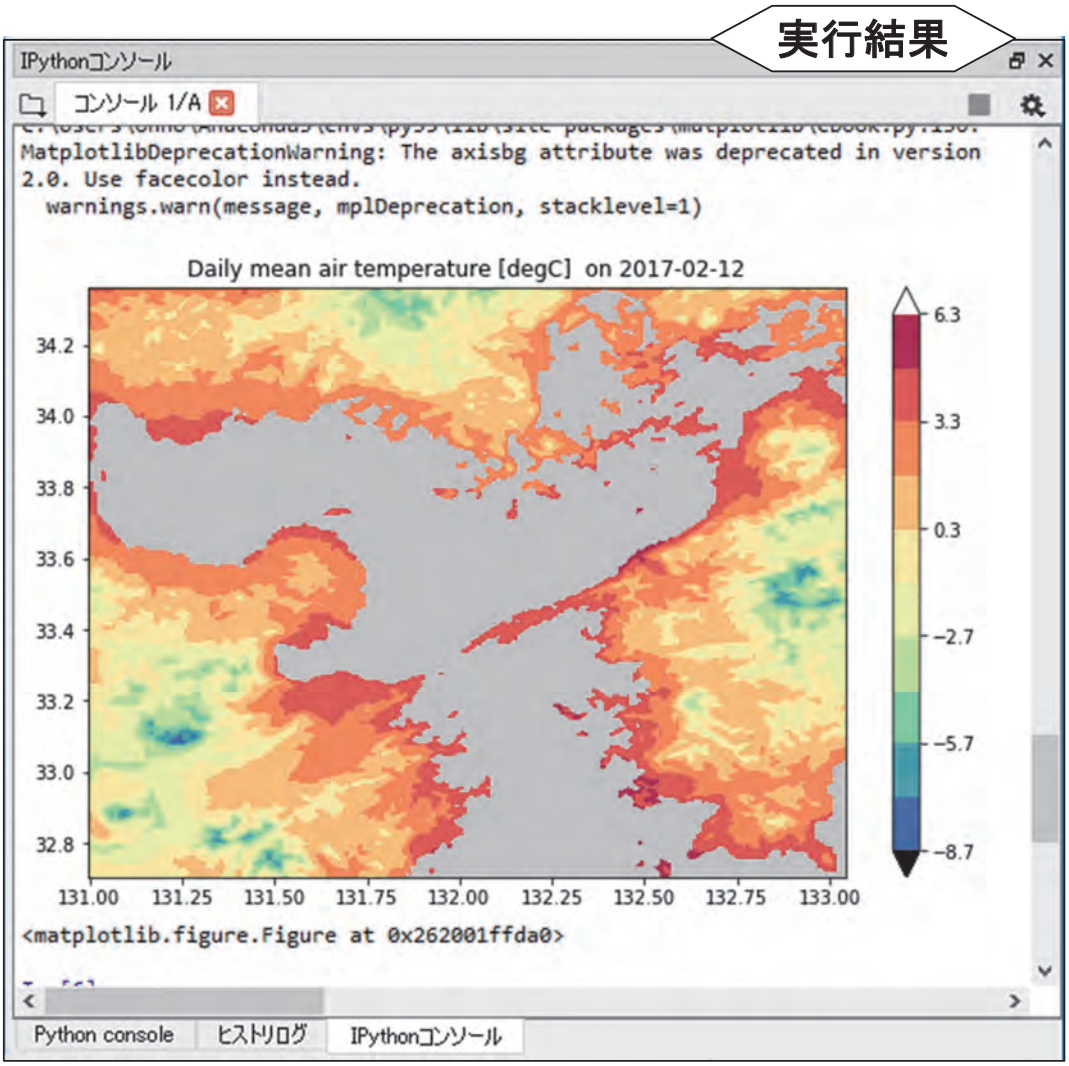


図 17. サンプルプログラム sample_GetMetData-b.py の実行結果

それでは，プログラムの中身を順に説明します。このプログラムでは若干の数値計算をします。このため，数値計算のための外部モジュール **numpy** を使用することを 43 行目で宣言します（図 18）。また，描画をするための外部モジュール **matplotlib** を使用することを 44 行目で宣言します。**matplotlib** は，きわめて高機能な描画モジュールですが，このプログラムではシンプルな分布図描画機能（**pylab**）さえ使えれば良いので，その部分だけを取り込むことをこのように宣言して指示します。なお，**matplotlib** でどのような描画を行うことができるかについて興味のある方は，**http://matplotlib.org/gallery.html** を参照してください。

47 行目にハッシュ記号（＃）で始まる灰色の文があります。この文はコメント文です。Python は，ハッシュ以降の部分をコメントとして無視します。

48 行目から 50 行目で，データ配信サーバーから取得するデータの気象要素や期間，領域を指定し，53 行でデータを取得します。今回は，分布図のタイトルに，気象要素名や単位を書き出すので，**GetMetData** 関数の引数に「**namuni=True**」を記述し，左辺では 6 つの変数で結果を受け取ります。受け取った日平均気温データは 1 日分ですが，三次元の入れ物に入っているので，54 行目の文で二次元の入れ物に入れなおします。

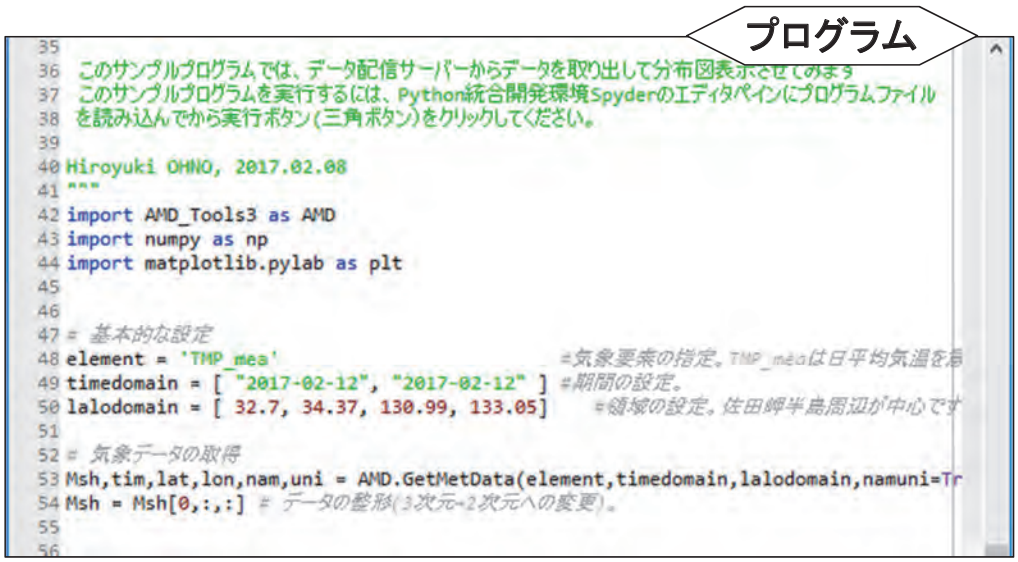


図 18.　サンプルプログラム sample_GetMetData-b.py の 35 行〜 55 行目

図 19 にサンプルプログラム **sample_GetMetData-b.py** の 57 行目以降を示します。この部分は，分布図を作成して右下ペインに表示するとともに PNG 画像（図 17 と全く同じ画像）ファイルを出力させるためのものです。入門段階の利用者は，これらの文を完全に理解する必要はありません。この部分を別なプログラムで再利用するうえで抑えるべき事柄を以下に説明しますのでそれだけ理解してください。

描画の核心部分（80 行目）において，**D2D** という名の二次元配列変数が図化に使用されているので，描画したい **Msh** の内容をそっくりこの配列変数に引き渡す必要があります。58 行目の文でこれを行っています。同様に，分布図の縦軸と横軸には，それぞれ **lat** と **lon** という名の一次元配列変数が使われているので，59 行目と 60 行目の文でこれらの引き渡しをしています。ただし，これらについては，プログラムの前半で同じ変数名を使用しており，引き渡す必要がないのでコメントアウト（文頭にハッシュを付けて無効化すること）しています。61 行目の変数 **tate** は，図の大きさを決めるのに使用します。より大きい値を与えるとより大きい図となります。縦横比は，緯度範囲と経度範囲の比を反映するようにしてあります（73 行目）。

62 行目の変数 **figtitle** には，図の上に表示させる文字列を代入します。この例では，**GetMetData** 関数で取得した気象要素の正式名称（**nam**）と単位（**uni**），データの日付を表示させるようにしています。Python では，引用符で囲んだ文字を文字列として扱います。文字列と文字列は「＋」記号で連結することができます。この文の右辺の最後に「**tim[0].strftime ('%Y-%m-%d')**」という項がありますが，これは，読み込んだデータの日時オブジェクトの内容を「**yyyy-mm-dd**」という形式の文字列で返させるものです。**GetMetData** 関数は，読んだ結果が単一日のデータであっても三次元の入れ物で返すのと同じように，時刻オブジェクトも内容がたった一つしかなくても，配列の入れ物に入れて返すので，**tim** ではなくて **tim[0]** として中身を取り出します。

63 行目の変数 **manualscl** は，色スケールの最大値と最小値を自動で割り振るか，値として直接指定するかを指示するために使われています。おまかせで設定するときには右辺に「**False**」と記入して偽を代入し，指定する場合は「**True**」と記入して真を代入してください。ここで，**False** や **True** は引用符で囲まれていないことに注意してください。これらは文字列ではなく，それぞれ真と偽を示す Python で予約されている記号です。

プログラム

```
57 #以下、分布図を書くのに重宝します。必要に応じてコピペして使用してください。
58 D2D = Msh          #分布図はD2Dという名の変数に入れてください。
59 #lat = lat        緯度はlatという名の変数に入れてください。
60 #lon = lon        経度はlonという名の変数に入れてください。
61 tate = 6        #図を入れるオブジェクト(入れ物)の全体的な大きさを指定します。
62 figtitle = nam+" ["+uni+"]  on "+tim[0].strftime('%Y-%m-%d')
63 manualscl = False     #カラースケールの上限値と下限値を指定したいときに使用します。
64 #----------
65 sclmax  = 12.8              #最大値
66 sclmin  =  5.4              #最小値
67 sclint  =  0.1              #色の刻み
68 if not manualscl :
69     sclmax = round(np.nanmax(np.array(D2D)),1)
70     sclmin = round(np.nanmin(np.array(D2D)),1)
71     sclint = round(((sclmax-sclmin)/10.0),1)
72 levels = np.arange(sclmin, sclmax+sclint, sclint)
73 yoko = tate * (np.max(lon)-np.min(lon))/(np.max(lat)-np.min(lat)) + 2
74 fig = plt.figure(num=None, figsize=(yoko, tate))#図を入れるオブジェクト(入れ物)を定義
75 plt.axes(axisbg='0.8')     #背景を灰色に
76 levels = np.arange(sclmin, sclmax+sclint, sclint)
77 cmap = plt.cm.Spectral_r     #色調(カラーマップ)を要称(「_r」を最後に付けると反転する)で指定
78 cmap.set_over('w', 1.0)
79 cmap.set_under('k', 1.0)
80 CF = plt.contourf(lon, lat, D2D, levels, cmap=cmap, extend='both')
81 plt.colorbar(CF)
82 plt.title(figtitle)
83 plt.savefig('result'+'.png', dpi=600) #この文を有効にすると、図をpng形式で保存します。
84 plt.show()
85 plt.clf()
86 #----------------------------------------
```

図 19.　サンプルプログラム sample_GetMetData-b.py の 57 行～ 86 行目

コラム 3

コメントをたくさん書きましょう

動作は同じであっても，プログラムには良いプログラムと悪いプログラムがあるといわれます。良いプログラムの条件はいくつかあるようですが，コメントをできるだけ入れるということは文句なしにトップの条件でしょう。

Python では，コメントを二通りの方法で書くことができます。一つはハッシュ記号（#）をつける方法です。コンピューターはハッシュ記号以下を無視します。

```
# 基本的な設定
element = 'TMP_mea'  # 気象要素の指定。
timedomain = [ "2016-01-01", "2016-01-03" ] # 期間の設定。
```

もう一つは，コメントしたい部分を 3 回連続する二重引用符で挟む方法です。長い補足説明を記録しておくときに便利です。

```
print（Msh）
"""
1 日分のデータなので，実質的には二次元（緯度，経度）ですが，三次元用の入れ物 [[[ ]]] に入っています。従って，「print(Msh[2,1])」とするとエラーになります。三次元用の入れ物から二次元用の入れ物に入れなおすには，次のようにします。
Msh = Msh[0,:,:]
"""
```

コメントは，プログラムの説明を記録するだけでなく，プログラム中の特定の文や一部分を一時的に無効化するためにもしばしば使われます。これをコメントアウトといいます。例えば以下の例です。

```
# 積算値を得るときは以下を活かす
"""
for i in range(len(tim)):
    Msh[i] = Msh[i] + Msh[i-1]
    MshN[i] = MshN[i] + MshN[i-1]
"""
```

3 連続二重引用符の使用にあたっては一点注意が必要です。3 連続二重引用符で挟まれた内側は自由にインデントできますが，3 連続二重引用符自体は，インデントのレベルを前後と揃えなければなければなりません。

色スケールの指定をする場合は，65 行目～ 67 行目でスケールの上限値と下限値，刻みを指定してください。**manualscl** に **False** を代入した場合は，これらの指定は無視されます。

PNG 画像ファイルへの出力は，83 行目の文で実行されます。違うファイル名で保存したいときは，「**'result'**」の部分を適宜変更してください。

3　気象の時系列変化グラフの作成

利用者 Wiki で配布する **sample_GetMetData-c.py** を例に，気象値の時系列変化グラフの作成手法を説明します。このプログラムは，緯度経度で指定した特定のメッシュにおける日平均気温と対応する平年値の推移を，2016 年 10 月 1 日から 2017 年 10 月 31 日までの期間について折れ線グラフで示すものです。プログラムを実行すると，図 20 のような折れ線グラフが表示されます。

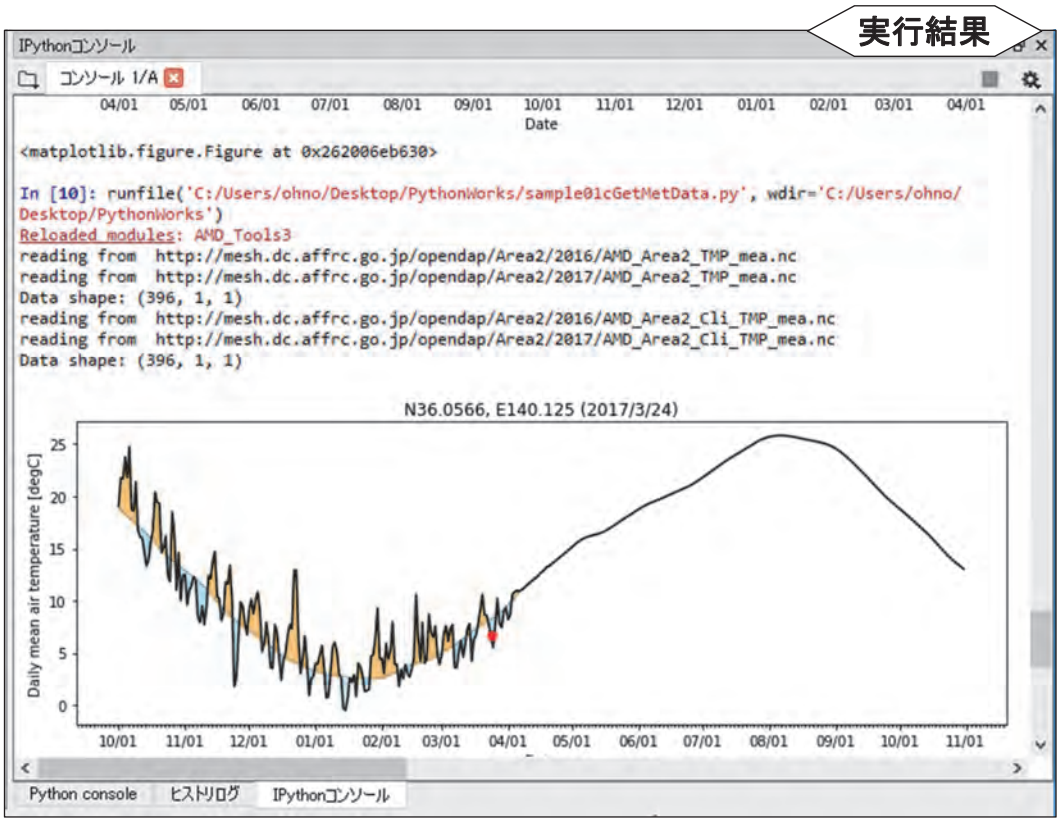


図 20.　サンプルプログラム sample_GetMetData-c.py の実行結果

それでは，プログラムの中身を順に説明します（図 21）。まず，プログラム 51 行目で，対象とする気象要素を指定します。52 行目ではグラフ化の期間を指定します。この例でわかる通り，Python プログラムにおいては期間の指定の際に年を跨ぐことが可能です。53 行目では，データを取り出す地点を指定します。この例でわかる通り，特定の 1 メッシュの気象値を取り出すときは，同じ緯度経度を 2 回指定します。

56 行目の文で，気象データを取得し，変数 **Msh** に格納します。この例では気象要素の正式名と単位を要求するためのオプション引数 **namuni** に **True** を与えています。Python において，0 は論理値 **False** と等値で，0 以外の整数は **True** と等値なので，**True** や **False** の代わりに数値を記述することもできます。57 行目の文は平年値を取得し変数 **MshN** に格納します。観測や予報の値ではなく平年値を取得することをオプション引数「**cli=1**」で指定しています。

メッシュ農業気象データは，日付，緯度，経度からなる三次元のデータなので，これを切り出す **GetMetData** 関数は，切り出し方に関係なく文の左辺にある気象データの入れ物（**Msh** や **MshN**）を三次元用に成形したうえでデータを流し込みます。しかし，今回の例では，緯度も経度も 1 要素なので，変数の入れ物が三次元用だと不便です。そこで，58 行目と 59 行目の文で，**Msh** や **MshN** を一次元の入れ物に作り直しています。

プログラム

```
41 Hiroyuki OHNO, 2017.02.08
42 """
43 import AMD_Tools3 as AMD
44 import numpy as np
45 from datetime import datetime
46 import matplotlib.pylab as plt
47 import matplotlib.dates as md
48
49
50 # 基本的な設定
51 element = 'TMP_mea'                          #気象要素の指定。TMP_meaは日平均気温を意味します。
52 timedomain = [ "2016-10-01", "2017-10-31" ] #期間の設定。
53 lalodomain = [ 36.0566, 36.0566, 140.125, 140.125]  #茨城県つくば市「つくば（館野）」
54
55 # 気象データの取得
56 Msh,tim,lat,lon,nam,uni = AMD.GetMetData(element,timedomain,lalodomain,namuni=True)
57 MshN,tim,lat,lon = AMD.GetMetData(element,timedomain,lalodomain,cli=1)
58 Msh = Msh[:,0,0]      #入れ物の入れ替え(3次元から1次元へ)
59 MshN = MshN[:,0,0]    #入れ物の入れ替え(3次元から1次元へ)
60
61
62 # この部分を有効にすると、積算値のグラフを作成することができます。
63 """
64 for i in range(len(tim)):
65     Msh[i] = Msh[i] + Msh[i-1]
66     MshN[i] = MshN[i] + MshN[i-1]
67 """
68
```

図 21．サンプルプログラム sample_GetMetData-c.py の 41 行～ 68 行目

コラム 4

リストと numpy.ndarray

Python には，「リスト」と呼ばれる配列のようなものが標準で定義されていて，プログラム中で多用されます。リストは，数や文字をカンマで区切って両脇を大括弧で括って作られます。リストは結構節操がなく，数字と文字列を混ぜたり，リストの要素にリストを与えたりすることができます。以下の例はすべて正しいリストです。

[1,2,3]　["a","b","c"]　[" 山川太郎 ", 13, 152.3, [" 美化委員会 "," 剣道部 "]]

この柔軟性を利用して，Python ではリストが様々な用法で使われています。
　さて，以下のプログラムを実行すると，どのような結果が表示されるでしょうか。

```
a = [1,2,3]
b = [3,2,1]
c = a + b
print(c)
```

答えは，[1,2,3,3,2,1] です。リストとリストをプラス記号（+）で繋ぐと，二つのリストが連結されます。要素同士で演算されることはありません。これはこれで便利ですが，気象データを処理するのには向いていません。そこで，メッシュ農業気象データシステムでは，気象データの処理に「ndarray」と呼ばれる型の配列を使用します。**AMD_Tools3** に含まれている **GetMetDat** 関数で気象データを取得すると，データは ndarray 型の配列で返されます。なお，**numpy** には似た名前の「array」という型も用意されていますが，これは用いません。ndarray の配列で上と同様な計算をしてみましょう。

```
import numpy as np
a = np.ndarray(3) # 要素を 3 つ持つ ndarray 型の配列変数を用意する
b = np.ndarray(3) # 要素を 3 つ持つ ndarray 型の配列変数を用意する
a[:] = [1,2,3]　#a に，リスト [1,2,3] の各要素を順に格納する
b[:] = [3,2,1]　#b に，リスト [3,2,1] の各要素を順に格納する
c = a + b
print(c)
```

今度は，[4,4,4] となりました。ndarray と ndarray 同士で演算をすると，それぞれの配列要素間で演算が実行されます。ndarray と数で演算をすると，配列のすべての要素にその数との演算が施されます。
　上の例で，**a[:] = [1,2,3]** ではなく **a = [1,2,3]** としてしまうと，せっかく ndarray として作った a はリストとして再定義されてしまいますので注意してください。ただし，a か b どちらか一方さえ，**a[:] = [1,2,3]** のようにしてあれば，もう片方はリストであっても結果は [4,4,4] となります。ndarray とリストとの演算が指示されると，Python は自動的にリストを ndarray に変換してから処理を実行します。**AMD_Tools3** で提供される **GetMetData** 関数は，取得したメッシュ農業気象データを ndarray の型で返すので，以降の演算に仮にリストを与えたとしても要素同士の演算となります。

気象値の推移をみる際，日々の値ではなくその積算を見たいケースがしばしばあります。その時は，63 行目と 67 行目の「" " "」を消去して 64 行目から 66 行目の文を有効にします。この部分は **for** ループと呼ばれ，同じ処理を一定回数繰り返すのに用いられます。

72 行目から 106 行目にかけての部分（図 22）には，気象データをグラフにする処理が記述されています。折れ線間を着色したり横軸の目盛りを日付にしたりするなどなかなか複雑なので，当座は理解する必要はありません。変数 **D1D1** と変数 **D1D2** にグラフ化する一次元配列を与え（73，74 行目），変数 **tim** に横軸として指定する日時オブジェクトを与え（75 行目），必要に応じて変数 **figtitle** に見出しの文字列（76 行目）を与えることだけ理解すれば使いまわすことができます。

プログラム

```
72 #以下、基準値と比較する折れ線グラフを書くのに重宝します。必要に応じてコピペして使いまわしてください。
73 D1D1 = Msh        #太線で描くデータをD1D1という名の変数に入れてください。
74 D1D2 = MshN       #細線で描くデータをD1D2という名の変数に入れてください。
75 #tim = tim       日付はtimという名の変数に入れてください。
76 figtitle = 'N'+str(lalodomain[0])+', E'+str(lalodomain[2])
77 #------------
78 D1D1 = np.array(D1D1)
79 D1D2 = np.array(D1D2)
80 tim = np.array(tim)
81 fig = plt.figure(num=None, figsize=(12, 4))
82 # ・目盛の作成
83 ax = plt.axes()
84 xmajoPos = md.DayLocator(bymonthday=[1])
85 xmajoFmt = md.DateFormatter('%m/%d')
86 ax.xaxis.set_major_locator(xmajoPos)
87 ax.xaxis.set_major_formatter(xmajoFmt)
88 xminoPos = md.DayLocator()
89 ax.xaxis.set_minor_locator(xminoPos)
90 # ・データのプロット
91 ax.fill_between(tim,D1D1,D1D2,where=D1D1>D1D2,facecolor='orange',alpha=0.5,interpolate=True)
92 ax.fill_between(tim,D1D2,D1D1,where=D1D1<D1D2,facecolor='skyblue',alpha=0.5,interpolate=True)
93 ax.plot(tim, D1D1, 'k')                                    #太線
94 ax.plot(tim, D1D2, 'k', linewidth=0.3)  #細線
95 # ・「今日」印を付ける
96 p = datetime.today()                                                    #今日の時刻オブジェクト
97 today = tim == datetime(p.year,p.month,p.day,0,0,0)          #今日の配列要素番号
98 plt.plot(tim[today], D1D1[today], "ro")                          #今日に赤点を打つ
99 # ・ラベル、タイトルの付加
100 plt.xlabel('Date')
101 plt.ylabel(nam+' [' + uni + ']')
102 plt.title(figtitle)
103 plt.savefig('result'+'.png', dpi=600) #この文をコメントアウトすると、図のpngファイルは作られません。
104 plt.show()
105 plt.clf()
106 #------------------------------------------------------------
```

図 22.　サンプルプログラム sample_GetMetData-c.py の 72 行～ 106 行目

コラム 5

「順次」「分岐」「反復」

コンピューターは，プログラムに書かれた文を上から順に実行するのが基本です。しかし，仕事に柔軟性を持たせるためには，条件により実行する文を変える必要があります。また，同じことを反復して実行する場合には，繰り返す内容を何度も書かずに，繰り返す範囲と繰り返しの回数や条件を指示して楽をします。Python において分岐をさせるには **if** 文を使います。反復をさせるには **for** 文や **while** 文を使用します。

4　地理情報の利用

メッシュ農業気象データシステムは，表 2 に示される地理情報を保持しており，気象データとほぼ同じ方法で取得することができます。この方法を，利用者 Wiki で配布するサンプルプログラム **sample_GetGeoData.py** を用いて説明します。このプログラムは図 23 で，静岡県周辺における水田面積比率分布図，静岡県域の図，静岡県域だけの水田面積比率分布図の 3 つを表示します（図 24）。

メッシュ農業気象データシステムから地理情報を取得する関数は **GetGeoData** です（43 行目ほか）。この関数は，期間を指定する引数がないことを除いて，気象データを取得する関数 **GetMetData** とほとんど同じです。地理情報の名称「**landuse_H210100**」（水田面積比率データ）を変数 **element** に一時的に保管し（39 行目），43 行目の文でそれを使ってデータを取得します。**GetGeoData** 関数により取得された水田面積比率データ，緯度範囲，経度範囲，正式名称，単位をもとに，46 行目から 75 行目までの文により分布図が作成されます（図 24a）。図 23 において

この部分は省略していますが，気温の分布図を作成するときに使用したスクリプト（図 19）と同じです。

メッシュ農業気象データシステムにおいて，静岡県の県番号は 2200 なので（表 3），静岡県の県域データの名称は「**pref_2200**」です（I 章参照）。78 行目でこの文字列を変数 **element** に一時的に保管し，以下，同様にしてデータを取得して静岡県域の分布図が作成されます（図 24b）。

Python では，同じサイズの 2 つの配列変数間で四則演算を行うと，対応する要素同士すべてに対してその演算が行われます（コラム 4 参照）。このため，120 行目の文により，**ShizPadd** には **Msh** と **Msh2** の対応する要素同士の積が格納されます。また，Python では，計算する値の中に無効値がある場合は，結果が常に無効値となります。このため，水田面比積率の分布図 **Msh** に，静岡県が 1，それ以外が無効値である **Msh2** を掛け合わせることにより，静岡県域だけについての水田面積比率分布図を作成することができます（図 24c）。

プログラム

```
38 # 基本的な設定
39 element = 'landuse_H210100'                    #地理情報の指定。landuse_H210100は水田面
40 lalodomain = [ 34.5, 36.0, 137.0, 139.5]      #領域の設定。駿河湾周辺です。
41
42 # 地理データの取得
43 Msh,lat,lon,nam,uni = AMD.GetGeoData(element,lalodomain,namuni=True)
44
45
46 #以下、分布図を書くのに重宝します。必要に応じてコピペして使用してください。----------
47 D2D = Msh          #分布図はD2Dという名の変数に入れてください。
48 #lat = lat         緯度はlatという名の変数に入れてください。
49 #lon = lon         経度はlonという名の変数に入れてください。
50 tate = 6        #図を入れるオブジェクト(入れ物)の全体的な大きさを指定します。
51 figtitle = nam+" ["+uni+"]"
52 manualscl = False     #カラースケールの上限値と下限値を指定したいときに使用します。
53 #----------
73 plt.show()
74 plt.clf()
75 #------------------------------------------------------------
76
77 # 基本的な設定
78 element = 'pref_2200'                    #地理情報の指定。pref_2200は静岡県域を意味します。
79
80 # 地理データの取得
81 Msh2,lat,lon,nam2,uni2 = AMD.GetGeoData(element,lalodomain,namuni=True)
116 pit.show()
117 plt.clf()
118 #------------------------------------------------------------
119
120 ShizPadd = Msh * Msh2
121 """
122   静岡県下の水田面積率布図を作成するには、水田面積率図と静岡県域のデータを掛け算します。
123 このように書くと、2枚の分布図の対応するメッシュすべてに対して掛け算が実行されます。無効値nan
124 には何を掛けてもnanになるので、結果として、静岡県域の以外が無効値となった分布図が
125 得られます。
126 """
127
128 #以下、分布図を書くのに重宝します。必要に応じてコピペして使用してください。----------
129 D2D = ShizPadd          #分布図はD2Dという名の変数に入れてください。
130 #lat = lat         緯度はlatという名の変数に入れてください。
```

図 23.　サンプルプログラム sample_GetGeoData.py の抜粋

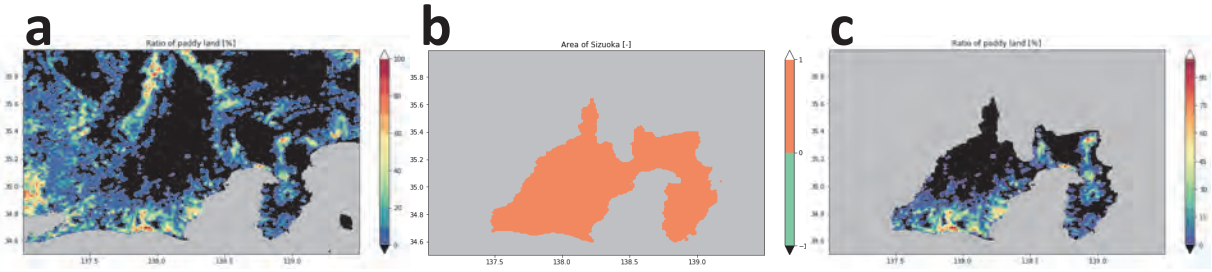


図 24.　サンプルプログラム sample_GetGeoData.py の実行により作成される 3 つの分布図

a：静岡県周辺域について取得された水田面積比率分布図，b：静岡県周辺域について取得された静岡県域分布図，c：計算の結果作成された静岡県域における水田面積比率分布図。

5　CSV 形式のメッシュデータの読み込み

GIS などで作成された 3 次メッシュに準拠するラスターデータをメッシュ農業気象データと組み合わせて Python で処理する場合，ラスターデータを Python の配列変数に読み込ませる必要があります。そのような場合には **AMD_Tools3** の **GetCSV_Map** 関数を使用します。利用者 Wiki で配布するサンプルプログラム **sample_GetCSV_Map.py** とサンプルデータ **sample_GetCSV_Map-data.csv** を用いてこの関数の使い方を説明します。サンプルデータは，テキストエディタにより図 25 右のように表示されるものです。サンプルプログラム（図 25 左）では，このデータのファイル名を一時的に変数 **filename** に格納し（36 行目），**GetCSV_Map** 関数はこのファイルを開いて数値を読み込み，配列変数 **Msh** に格納します（37 行目）。配列変数 **Msh** の内容を表示させる（38 行目）と，結果は図 26 上段のようになります。画面には「Warning: possibly illegal elements」（警告：不正な要素の可能性）という文が表示されています。**GetCSV_Map** 関数は CSV ファイルを浮動小数の配列変数に格納しようと試み，数字にならないデータを読み取ると警告のメッセージを出します。この警告は，見出しの行が数値に変換できないために表示されました。見出の行を読み込み対象から外すには，63 行目の文のようにオプション引数 **skiprow** に無視する行数を与えます。次の警告文は「“3.5”」に対して発せられたものです。**GetCSV_Map** 関数は数値でないデータが取り込まれると，警告を表示してその要素に無効値「**nan**」を埋め込むことに注意してください。

GetCSV_Map 関数はデータを読み込んだ後，配列変数における行の順序を入れ替えることにも注意してください。メッシュ農業気象データは，配列の中で南から北にデータを格納しているのに対し，外部の地理データの多くは北を上，すなわち，北から南にデータを格納しているため

図 25.　サンプルプログラム sample_GetCSV_Map.py の抜粋（左）と，サンプルデータ sample_GetCSV_Map-data.csv の内容（右）

に，**GetCSV_Map** 関数は標準で行の順序を入れ替えるように作られています。南北の転置をさせなくするには，63 行目の文のようにオプション引数 **upsidedown** に偽（**False** または 0）を与えます。データの最初の行を無視し，さらに行転置しないようにオプション引数を指定した場合の結果を図 26 下段に示します。

```
Warning: possibly illegal elements
Row 1: "A列","B列","C列","D列","E列"

Warning: possibly illegal elements
Row 4: 2.0 ,"3.5",nan,0.0 ,0.0

sample03_GetCSV_Map_data.csv:  (5, 5)
[[ 9.89999962  9.89999962  9.89999962  9.89999962  9.89999962]
 [ 2.                nan         nan  0.          0.        ]
 [ 1.          0.          0.          0.          0.        ]
 [        nan  1.          0.5         3.          4.5       ]
 [        nan         nan         nan         nan         nan]]

Warning: possibly illegal elements
Row 4: 2.0 ,"3.5",nan,0.0 ,0.0

sample03_GetCSV_Map_data.csv:  (4, 5)
[[        nan  1.          0.5         3.          4.5       ]
 [ 1.          0.          0.          0.          0.        ]
 [ 2.                nan         nan  0.          0.        ]
 [ 9.89999962  9.89999962  9.89999962  9.89999962  9.89999962]]
```

図 26. サンプルプログラム sample_GetCSV_Map.py をオプション無しで実行した場合の結果（上段）と，オプション引数により 1 行の読み飛ばしと，行転置の禁止を指定した場合の結果（下段）

6　時系列データの書き出し

特定メッシュの日別気象値を取り出して表計算ソフトに読み込ませたいときなどに，**AMD_Tools3** の **PutCSV_TS** 関数を使用することができます。この関数の使い方をサンプルプログラム **sample_PutCSV_TS.py**（図 27）を使って説明します。このプログラムは，茨城県つくば市のアメダス観測所付近のメッシュにおける日平均気温を 2016 年 4 月 1 日から 4 月 14 日までの期間について取得して CSV ファイルとして出力するものです。

気象要素，期間，地点の指定は 36 行目から 38 行目の文で実行されます。41 行目でこのデータが読み込まれて配列変数 **Msh** に格納されます。42 行目では配列変数 **Msh** の形状が三次元用から一次元用に変更されます。CSV ファイルへの書き出しは 44 行目の文で実行されます。この文が実行されるとプログラムファイルと同じディレクトリに **result.csv** というファイルが作成されます。これを表計算ソフトで開くと，図 28 図左のようになります。

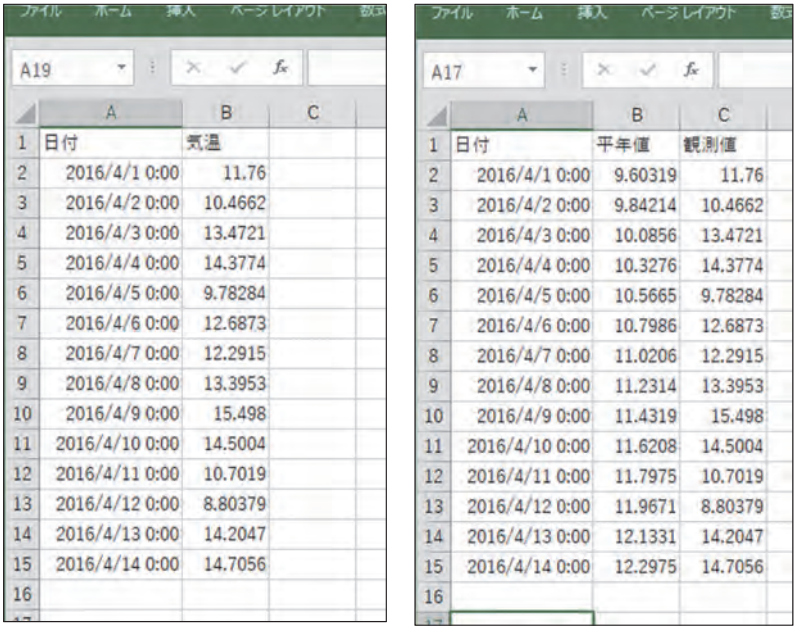
この例では日付とデータの 2 列だけですが，気温と日射量，降水量というように，複数種のデータを並べる場合には，並べるデータに工夫を加えてから **PutCSV_TS** 関数の第一引数に与えます。サンプルプログラムには，平年値と観測値を並列して出力させる例が示されています。66 行目と 67 行目で同地点・同期間の気温の平年値を取得し **MshN** に格納したあと，**PutCSV_TS** 関数に与える前に 75 行の文を実行して，**MshN** と **Msh** の各要素が一対で並んだもの一括りを要素とする配列変数 **Tateno** を作り，これを関数に与えます。ただし，この文をエラーなく実行するには，プログラムの最初にインポート文を書いて **numpy** モジュールをインポートしておかなくてはなりません（32 行目）。結果を図 28 右に示します。

プログラム

```
31 import AMD_Tools3 as AMD
32 import numpy as np
33
34
35 # 取得する気象データの設定
36 element = 'TMP_mea'                                    #気象要素の指定。TMP_meaは日平均気温を意味します。
37 timedomain = [ "2016-04-01", "2016-04-14" ] #期間の設定。
38 lalodomain = [ 36.0566, 36.0566, 140.125, 140.125]  #茨城県つくば市「つくば(諸野)」
39
40 # 気象データの取得
41 Msh,tim,lat,lon,nam,uni = AMD.GetMetData(element,timedomain,lalodomain,namuni=True)
42 Msh = Msh[:,0,0]      #入れ物の入れ替え(3次元から1次元へ)
43
44 AMD.PutCSV_TS(Msh, tim, header="日付,気温")
45 """
65
66 MshN,tim,lat,lon = AMD.GetMetData(element,timedomain,lalodomain,cli=1)
67 MshN = MshN[:,0,0]    #入れ物の入れ替え(3次元から1次元へ)
68 """
75 Tateno = np.array([MshN,Msh])
76
77 AMD.PutCSV_TS(Tateno, tim, header="日付,平年値,観測値", filename="result2.csv")
78 """
```

図 27.　サンプルプログラム sample_PutCSV_TS.py の抜粋

PutCSV_TS 関数には **filename** というオプション引数が用意されていて，ここに文字列を代入することで任意のファイル名で結果を出力させることができます。省略した場合は，**result.csv** の名で出力されます。このサンプルプログラムの実行後にできている **result.csv** は，オプション引数を省略して **PutCSV_TS** 関数を実行した 44 行目の文による出力です。



A19

	A	B	C
1	日付	気温	
2	2016/4/1 0:00	11.76	
3	2016/4/2 0:00	10.4662	
4	2016/4/3 0:00	13.4721	
5	2016/4/4 0:00	14.3774	
6	2016/4/5 0:00	9.78284	
7	2016/4/6 0:00	12.6873	
8	2016/4/7 0:00	12.2915	
9	2016/4/8 0:00	13.3953	
10	2016/4/9 0:00	15.498	
11	2016/4/10 0:00	14.5004	
12	2016/4/11 0:00	10.7019	
13	2016/4/12 0:00	8.80379	
14	2016/4/13 0:00	14.2047	
15	2016/4/14 0:00	14.7056	
16			

A17

	A	B	C
1	日付	平年値	観測値
2	2016/4/1 0:00	9.60319	11.76
3	2016/4/2 0:00	9.84214	10.4662
4	2016/4/3 0:00	10.0856	13.4721
5	2016/4/4 0:00	10.3276	14.3774
6	2016/4/5 0:00	10.5665	9.78284
7	2016/4/6 0:00	10.7986	12.6873
8	2016/4/7 0:00	11.0206	12.2915
9	2016/4/8 0:00	11.2314	13.3953
10	2016/4/9 0:00	11.4319	15.498
11	2016/4/10 0:00	11.6208	14.5004
12	2016/4/11 0:00	11.7975	10.7019
13	2016/4/12 0:00	11.9671	8.80379
14	2016/4/13 0:00	12.1331	14.2047
15	2016/4/14 0:00	12.2975	14.7056
16			

図 28.　サンプルプログラム sample_PutCSV_TS.py の実行により作成されたファイルを表計算ソフトに読み込んだ結果

左：result.csv の読み込み結果。右：result2.csv の読み込み結果。

7　CSV 形式の分布図の書き出し

表計算ソフトのワークシート上に並ぶセルをメッシュに見立て，Python プログラムで作成したメッシュデータをワークシートの各セルに流し込んでしまうと便利なことが時にあります。**AMD_Tools3** の **PutCSV_Map** 関数を用いると，各セルにメッシュの値を与えることができる CSV ファイルを作成することができます。この方法を利用者 Wiki で配布するサンプルプログラム **sample_PutCSV_Map.py** を用いて説明します。

このプログラムは，愛媛県の佐田岬半島周辺における 2016 年 1 月 1 日の日平均気温を CSV ファイルに出力するものです（図 29）。30 行目から 32 行目で取得するデータを指定し，これをもとに 35 行目で変数 **Msh** に取り込み，36 行目で変数の入れ物を二次元に変更します。この分布図

を CSV ファイルで出力します。この処理は 39 行目で行います。分布図データ，緯度値の並び，経度値の並びを関数 **PutCSV_Map** 関数に与えます。オプション引数 **filename** に文字列を指定すると，その文字列がファイル名に使用されます。この例では，文字列変数 **element** に格納される「**TMP_mea**」を名前にするように指定しています。**filename** を指定しない場合は **result.csv** という名前で出力されます。出力されたファイルを表計算ソフトで開き，50% 程度の倍率で表示させると，佐田岬半島周辺の海陸分布が数値で感じ取れると思います。

なお，プログラム実行後に，Spyder の右上ペインに並ぶタブから，「変数エクスプローラー」を選択すると，プログラムで用いた変数の名前や数値型，サイズなどを確認することができます。そして，その中から **Msh** に関するものを探し，その行をダブルリックすると，メッシュ毎の値を数値と色で確認することができます（図 30）。

プログラム

```
26 import AMD_Tools3 as AMD
27
28
29 # 取得するデータの設定
30 element = 'TMP_mea'                          #気象要素の指定。TMP_meaは日平均気温を意味します。
31 timedomain = [ "2016-01-01", "2016-01-01" ]  #期間の設定。
32 lalodomain = [ 32.7, 34.37, 130.99, 133.05]  #領域の設定。佐田岬半島の先端周辺です。
33
34 # 気象データの取得
35 Msh,tim,lat,lon = AMD.GetMetData(element,timedomain,lalodomain)
36 Msh = Msh[0,:,:] # データの整形(3次元→2次元への変更)。
37
38 # 気象データの書き出し
39 AMD.PutCSV_Map(Msh, lat, lon, filename=element+".csv")
```

図 29.　サンプルプログラム sample_PutCSV_Map.py の 26 行～ 39 行目

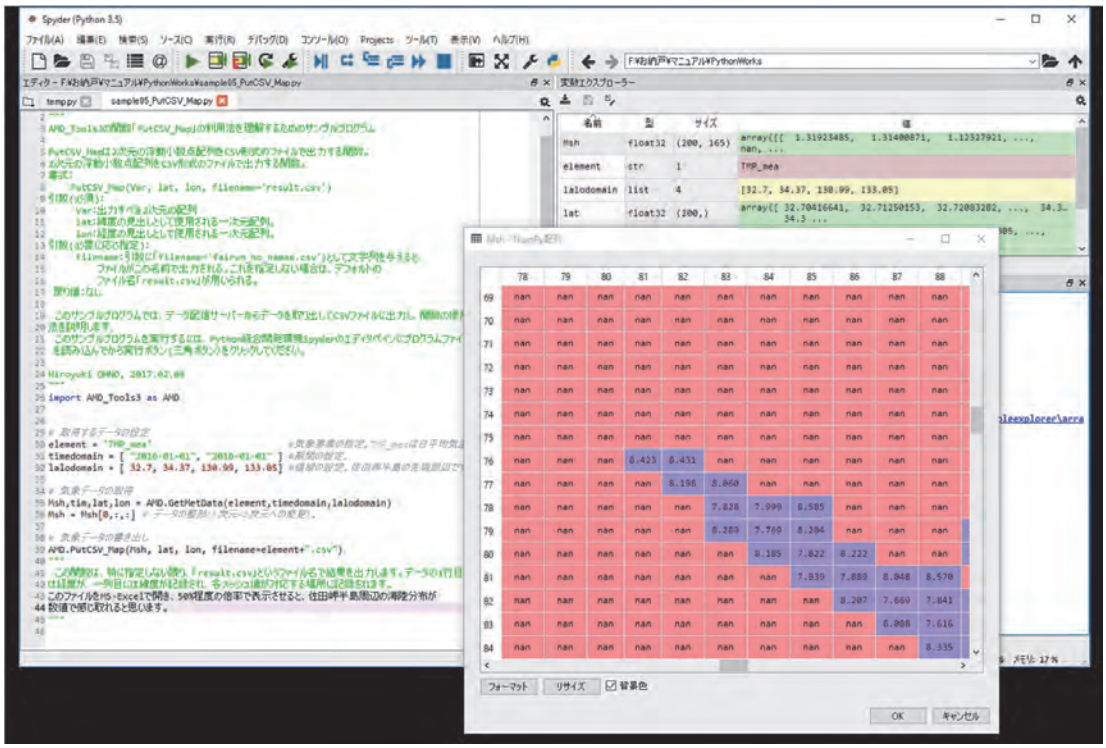


図 30.　Python プログラム統合開発環境 Spyder の変数エクスプローラー機能を利用し，変数 Msh に格納されている数値を確認したところ

8　メッシュ番号をキーとする属性テーブルの出力

Python プログラムで作成した分布図を GIS にインポートする最も実用的な方法は，3 次メッシュコードとそこでの値の対からなる CSV 形式のテーブルを用いることです。その方法の概要は以下のようなものです。まず，3 次メッシュの外周の四角形を地理情報要素とし，3 次メッシュコードを属性データベースとして持つ GIS データを予め用意しておきます。小さな 4 角形が集合した日本地図のように見えます。次に，3 次メッシュコードと数値の対からなる CSV ファイルを GIS にテーブルとしてインポートします。そして最後に，3 次メッシュコードをキーとしてこのテーブルを GIS データの属性データベースに連結します。

AMD_Tools3 の **PutCSV_MT** 関数を用いると，このような CSV ファイルを作成することができます。利用者 Wiki で配布するサンプルプログラム **sample_PutCSV_MT.py** を用いて説明します。図 31 にプログラムの抜粋を示します。

プログラム 39 行目から 41 行目までで取得するデータを指定し，それに基づいて 44 行目でメッシュデータを取得します。取得したメッシュデータ，緯度情報，経度情報を 47 行目で示す通り **PutCSV_MT** 関数の引数に与えると，図 32a のような CSV ファイルが出力されます（**result.csv**）。ファイルの内容は，1 列目が基準 3 次メッシュコード，2 列目が 2016 年 1 月 1 日の日平均気温，3 列目が同年 1 月 2 日の日平均気温，4 列目が同年 1 月 3 日の日平均気温，5 列目が同年 1 月 4 日の日平均気温です。41 行目で指定した領域には 30 個のメッシュが含まれますが，このうち海でないメッシュは 8 個だけなので，8 個のメッシュだけが出力されています。すべてのメッシュをあからさまに出力させるには，引数に「**removenan=False**」を追加します。

GIS に読み込ませるのであればこのような CSV ファイルで十分ですが，メッシュ中心点の緯度経度や見出しも付加しておくと，あとから確認ができて便利です。64 行目のように，オプショ

プログラム

```
35 import numpy as np
36 import AMD_Tools3 as AMD
37
38 # 取得するデータの設定
39 element = 'TMP_mea'                          #気象要素の指定。TMP_meaは日平均気温を意味します。
40 timedomain = [ "2016-01-01", "2016-01-04" ] #期間の設定。
41 lalodomain = [ 33.32, 33.37, 131.99, 132.05]     #領域の設定。佐田岬半島の先端あたりです。
42
43 # 気象データの取得
44 Msh,tim,lat,lon = AMD.GetMetData(element,timedomain,lalodomain)
45
46 # 気象データの書き出し
47 AMD.PutCSV_MT(Msh, lat, lon)
```

```
63
64 AMD.PutCSV_MT(Msh, lat, lon, addlalo=True,
65               header="メッシュコード,緯度,経度,1月1日,1月2日,1月3日,1月4日",
66               filename="result2.csv") #括弧の中では、自由に改行とインデントができます。
```

```
83 timedomain = [ "2016-01-01", "2016-01-01" ] #期間の設定。
84 MshA,tim,lat,lon = AMD.GetMetData('TMP_mea',timedomain,lalodomain)
85 MshA = MshA[0,:,:]
86 MshB,tim,lat,lon = AMD.GetMetData('TMP_max',timedomain,lalodomain)
87 MshB = MshB[0,:,:]
88 MshC,tim,lat,lon = AMD.GetMetData('TMP_min',timedomain,lalodomain)
89 MshC = MshC[0,:,:]
90 Msh2 = np.array([MshA,MshB,MshC])
91 AMD.PutCSV_MT(Msh2, lat, lon,
92               header="メッシュコード,Tmean,Tmax,Tmin", filename="result3.csv")
```

図 31．サンプルプログラム sample_PutCSV_MT.py の抜粋

ン引数 **addlalo** と **header** を追加で指定すると，図 32b のような CSV ファイルを作成することができます（**result2.csv**）。

ばらばらに作成した配列を連結して出力するには，90 行目のように，**numpy** モジュールに含まれる **array** 関数を用いて配列変数を結合してから **PutCSV_MT** 関数に引き渡します（**result3.csv**，図 32c）。

なお，Python では，インデントの深さはプログラムの実行単位を示す意味を持つので，見やすさのために勝手に字下げをすることができませんが，関数などで括弧に括られている内部については，この規則が適用されません。したがって，64 ～ 66 行目の文のように，関数の引数が長くなる場合には，好きなところで改行できます。

a

50320000	8.42308	11.3434	13.7409	13.0435
50320001	8.43082	11.3283	13.7312	13.0511
50320011	8.19772	11.1041	13.4959	12.8265
50320012	8.05997	10.9354	13.325	12.6688
50320022	7.82793	10.7112	13.0907	12.4455
50320023	7.99939	10.8495	13.2245	12.5943
50320032	8.28939	11.1682	13.5265	12.8809
50320033	7.76876	10.6257	12.9914	12.3725

b

メッシュ	緯度	経度	1月1日	1月2日	1月3日	1月4日
50320000	33.3375	132.006	8.42308	11.3434	13.7409	13.0435
50320001	33.3375	132.019	8.43082	11.3283	13.7312	13.0511
50320011	33.3458	132.019	8.19772	11.1041	13.4959	12.8265
50320012	33.3458	132.031	8.05997	10.9354	13.325	12.6688
50320022	33.3542	132.031	7.82793	10.7112	13.0907	12.4455
50320023	33.3542	132.044	7.99939	10.8495	13.2245	12.5943
50320032	33.3625	132.031	8.28939	11.1682	13.5265	12.8809
50320033	33.3625	132.044	7.76876	10.6257	12.9914	12.3725

c

メッシュ	Tmean	Tmax	Tmin
50320000	8.42308	12.9888	5.07289
50320001	8.43082	12.9816	5.06517
50320011	8.19772	12.5856	4.82417
50320012	8.05997	12.4761	4.73078
50320022	7.82793	12.1788	4.44593
50320023	7.99939	12.2195	4.71737
50320032	8.28939	12.6806	5.01522
50320033	7.76876	12.1676	4.4425

図 32. サンプルプログラム sample_PutCSV_MT.py を実行して作成される CSV ファイルの内容

a：オプション引数なしで実行させた結果（result.csv）。b：オプション引数を指定して，メッシュの中心緯度経度，列見出しを表示させた結果（result2.csv）。c：同じ領域の二次元配列変数の結果を連結して関数に与えた結果（result3.csv）。

コラム 6

インデントについて

Python では, ループなどひとまとまりで扱うべき文をインデントそのもので表現します。下の例では, 繰り返し計算の中身は 3 行目だけで，これが 5 回繰り返されてから 4 行目に移ります。どこからどこまでを繰り返すかは同じインデントがどこからどこまで施されているかで決まります。「end for」など, ループの終わりを示す文はありません。

```
sum = 0
for i in [1,3,5,7,9]:
   sum = sum + i
print(sum)
```

従って，次の例のように，4 行目をインデントすると，画面には，計算途中の値が 5 回表示されるようになります。

```
sum = 0
for i in [1,3,5,7,9]:
   sum = sum + i
   print(sum)
```

インデントには半角空白を使うようにしてください。タブが混ざっていると Python はこれを異なるインデントと理解しエラーの原因になります。また，全角空白は，Python にとって未知の記号としてエラーとなります。

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

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

Spyder のプログラムエディターは, 仮にタブを使用しても半角空白に変換されて入力されます。また, 上の例の場合，続けて書いた後に適当なところで改行を入れると，切りのいいところまで続きを自動でインデントしてくれます。

9　Google Earth 上に表示できる分布図の作成

AMD_Tools3 の **PutKMZ_Map** 関数を用いてファイルを作成すると，メッシュ農業気象データから作成した分布図を，Google Earth の地球儀上に張り付けて表示することができます。Google Earth に張り付けられた分布図は，拡大縮小，回転，遠近法表示が簡単にできるほか, 分布図を半透明にもできるので分布図が示す内容を深く理解することができます。このファイルを作成する方法を，利用者 Wiki で配布するサンプルプログラム **sample_PutKMZ_Map.py** を用いて説明します。図 33 にプログラムの抜粋を示します。47 行目から 49 行目で取得するデータを指定し，これをもとに 52 行目で配列変数 **Msh** に取り込み，61 行目で変数の入れ物を二次元に変更します。そして，66 行目で分布データから KMZ ファイルを作成します。**PutKMZ_Map** 関数は，凡例の最大値最小値の指定や，使用する色合い，凡例のラベル，出力するファイルの名前などを指定するための多数のオプションがあります。それらの詳細は次章を参照してください。Google Earth がインストールされた PC で，このファイルをダブルクリックすると，図 34 のように，分布図が張り付けられた形で表示されます。

プログラム

```
44 import AMD_Tools3 as AMD
45
46 # 基本的な設定
47 element = 'TMP_mea'                          #気象要素の指定。TMP_meaは日平均気温を意味します。
48 timedomain = [ "2016-01-01", "2016-01-01" ] #期間の設定。
49 lalodomain = [ 32.7, 34.37, 130.99, 133.05] #領域の設定。佐田岬半島の先端周辺です。
50
51 # 気象データの読み込み
52 Msh,tim,lat,lon,nam,uni = AMD.GetMetData(element, timedomain, lalodomain,namuni=True)
53 # Mshには気象データが格納されます。
54 # timには、timedomainで指定した2016年1月1日の日付オブジェクトが格納されます。
55 # latには、北緯36.0～38.5度の範囲に含まれるメッシュ中心点の緯度値が格納されます。
56 # lonには、東経137.5～141.5度の範囲に含まれるメッシュ中心点の経度値が格納されます。
57 # namには、日平均気温が英語で格納されます。
58 # uniには、気温の単位が格納されます。
59
60 # データの整形(3次元→2次元への変更)。
61 Msh = Msh[0,:,:]
62 # メッシュ農業気象データーは、日付、緯度、経度の3つの次元を持ちますが、
63 # 今回は1日だけのデータなので、上のようにして2次元のデータ変換して後の処理を行います。
64
65 # GoogleEarth表示用ファイル出力
66 AMD.PutKMZ_Map(Msh,lat,lon,label=nam+" ["+uni+"]",filename=element)
```

図 33.　サンプルプログラム sample_PutKMZ_Map.py の 44 行～ 66 行目

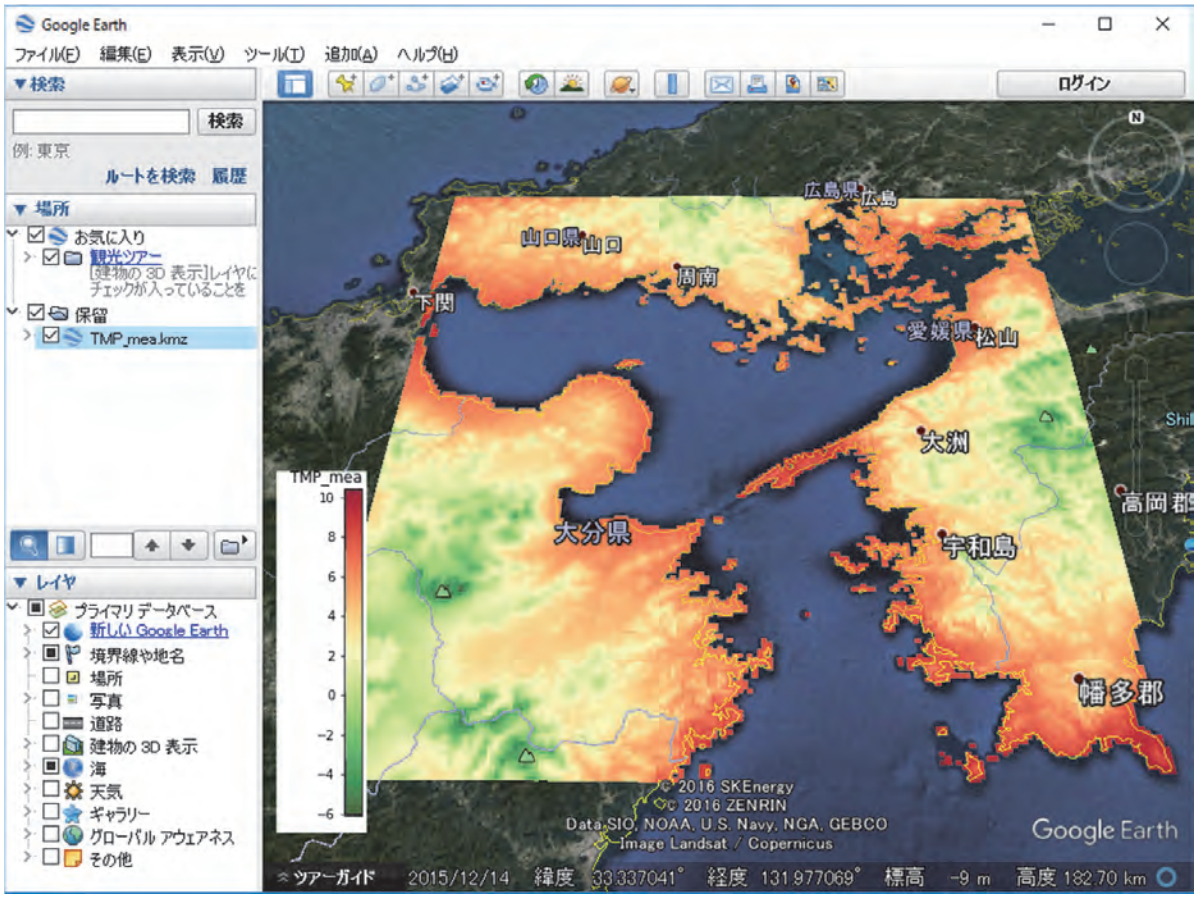


図 34.　PutKMZ_Map 関数を用いて，愛媛県の佐田岬半島を中心とする地域における 2016 年 1 月 1 日の日平均気温の分布図を，Google Earth 上に張り付けて表示した画面

10　発育指数法を用いた水稲の出穂日分布図の作成

メッシュ農業気象データは最長 26 日先までの気象予報値が含まれているので，水稲をはじめとする作物の発育予測で威力を発揮します。また，面的な気象分布が得られるのもこのデータの特徴なので，特定品種の水稲を県下一斉に移植したと仮定した時の出穂日の予測分布図を作成してみましょう。出穂日は発育指数法で推定することとし，発育の速度は有効積算気温モデルを用いることにします。ここで，発育指数法とは，気温等の気象要素から計算される発育の速度を日々積算してそれが 1 となった日を出穂日とする発育の予測法です。そして，有効積算気温モデルとは，ある日の発育速度が有効気温に比例すると考えるモデルです。ここで，有効気温とは，作物が発育を進めるのに必要な最低限の環境温度（基準温度）を日平均気温から差し引いたものです。**sample_RiceDevel.py** は，この方法で出穂日の分布を予測するサンプルプログラムです。このプログラムは利用者 Wiki から入手することができます。

発育指数法では日々の気象データから発育速度を計算して積算するので，出穂の予測に際し，メッシュ当たり 60 回程度同じような計算を繰り返すことになります。そこで，発育速度の計算式を関数としてあらかじめ登録し簡単に使えるようにします。Python では，関数の名前と関数が使用する引数を **def** 文で宣言します（図 35，17 行目）。def 文の最後にコロン（：）を付けるのを忘れないでください。有効積算気温モデルは，二つのパラメータ（基準温度と，出穂に達する積算値）を持ます。これらは，モデルを適用する地域や品種で異なるので，これらも引数として与えます。サンプルプログラムでは，パラメータを 2 要素リストのオプション引数 **Para** として定義しています。オプション引数とは，あらかじめデフォルト値を決めておき，それでよければ，引数の記述が省略できるというものです。

有効積算気温モデルでは，日平均気温が基準温度より高いか低いかで異なる計算を実行する必要があります。Python プログラムでこのような処理を行うには if 文を使用します。if 文は，「**if**」とコロン（：）に挟まれた部分に記述した条件式が成り立つ（真の）場合はその下の文を実行し，成り立たない（偽の）場合はそれを無視します。ただし，「**else**：」という文がある場合には，偽の時だけその下を実行します。真のときや偽の時に実行する文は，if 文や else 文よりも深いインデントにしなければなりません。また，このサンプルプログラムでは，真の場合も偽の場合も実行する文はそれぞれ一つだけですが，複数の文でも構いません。この場合，実行させる文のイ

プログラム

```
11
12 import numpy as np
13 import matplotlib.pylab as plt
14 import AMD_Tools3 as AMD
15
16 #発育速度関数の定義部分
17 def DVR(Ta, Para=[10.0, 1000.0]): #関数はdef文で定義します。
18     """
19     積算温度型の発育速度関数。
20     引数(必須):
21       Ta:日平均気温
22     引数(必要に応じ設定)
23       Para:基準温度(これ以下の場合積算をしない)、ならびに、出穂到達積算温度を束ねたリスト
24             デフォルト値として[10, 1000]が設定されている。
25     戻り値:気温に対する発育速度値
26     """
27     if Para[0] < Ta:
28         DVR = (Ta-Para[0])/Para[1]
29     else:
30         DVR = 0.0
31     return DVR
32
```

図 35.　サンプルプログラム sample_RiceDevel.py の 12 行～ 32 行目

ンデントを一致させます。

このように，Python ではインデントのレベルで文のまとまりを表現します。したがって，17 行目の def 文に対して 18 行目の 3 回連続した二重引用符が一段低いのにも意味があり，一段低いことによって関数を定義する部分であること表現しています。

コラム 7

if 文

条件分けをするときに使われる if 文は，多くのプログラム言語でも使われていますが，Python では独特な書き方をします。たとえば，変数 x が 1 のとき hello と表示するプログラムは，次のように書かれます。

```
if x == 1 :
   print('hello')
```

if とコロン（：）に挟まれた部分に，条件を記します。「＝＝」は，両辺の値が等しいかどうかを調べる記号です。調べた結果が真のとき，その下の文が実行されます。Python では，この際，同じ深さのインデントの範囲が if 条件下の処理部分となり，「end if」などの if 文の終わりを表す文はありません。両辺の値を比べる記号は比較演算子とよび，「等しい」のほかにも幾つかあります。

==	a == b	a と b は等しい
!=	a != b	a と b は等しくない
>	a > b	a は b より大きい
>=	a >= b	a は b 以上
<	a < b	a は b より小さい
<=	a <= b	a は b 以下

ある比較の結果と別な比較の結果を組み合わせるとより複雑な条件を判別することができます。この時に使われるのが論理演算子です。Python における論理演算子は以下のとおりです。

and	a and b	a と b が真の場合に真
or	a or b	a または b の少なくともどちらか一方が真の場合に真
not	not a	a ではない場合に真

比較演算子と論理演算子を組み合わせた例として，たとえば x が 1 で，且つ，y が 2 より小さいとき hello と表示するプログラムは，次のように書かれます。

```
if x == 1 and y < 2:
   print('hello')
```

この例でわかる通り，比較演算と論理演算がずらずらある場合は，比較演算が優先して実行されます。

条件によって実行内容を二つ以上に分岐させたいときがあります。たとえば，x が 1 のとき hello，2 のとき fine，それ以外のとき thank you と表示させるようなときです。このようなときは，次のようにします。elseif は複数書くことができます。

```
if x == 1：
   print('hello')
elseif x == 2:
   print('fine')
else:
   print('thank you')
```

このプログラムを呼び出して気象データから出穂を予測する部分は，34 行目から始まります（図 36）。これは，この文がインデントされていないことでもわかります。プログラムの本体ともいえる部分が if 文で始まるのは奇妙ですが，このように理解してください。すなわち，「あるプログラムファイルがあってその中にいろいろな関数が書かれているときに，もし何の指定もしないでプログラムを実行させたならばその時はこの if 文以下が実行される」。

この if 文の下は，1段インデントが深くなっている以外は他のサンプルプログラムと同じです。まず初めに，県下一律と仮定する移植日（36, 37 行目），移植時の発育指数（38 行目），発育速度のパラメータ（39 行目）など移植にかかわる設定をします。続いて，県域データの略記号（42 行目）と緯度経度範囲（43 行）を指定し，これらに基づいて気象データと県域データを取得して配列変数 **Msh**（45 行目）と配列変数 **Pref**（46 行目）にそれぞれ格納します。このプログラムでは，分布図を作成する対象に新潟県を選んでいますが，42 行目と，43 行目を変更すると他の都道府県の分布図を作成することができます。県域データの略記号に含まれる県番号は，表 3 から知ることができます。また，ある県がすっぽり入る緯度経度範囲は，利用者 Wiki に掲載される表から知ることができます。

このプログラムで使われる発育速度の式は単純ですが，これを繰り返して積算を取る処理をメッシュ毎に繰り返し実行するので計算量自体は多く時間がかかります。そこで，海上など気象データがないメッシュや新潟県外のメッシュについては計算対象から除外して実行時間を短縮することにします。この選別は 49 行目と 50 行目で行われています。メッシュ農業気象データ配信サーバーから取得した日平均気温データ **Msh** から第 1 日目を取り出したもの（**Msh[0, :, :]**）と新潟県の県域データ（**Pref**）とを掛け合わせると，結果は，気象データが無いかまたは新潟県外であるメッシュに対応する配列要素に対し無効値（**np.nan**）が与えられそれ以外には第 1 日目の日平均気温が与えられた配列が計算されるので，まず 49 行目では，気象データが無いかまたは新潟県外であるメッシュに対応する配列要素に対し無効値（**np.nan**），それ以外の要素には 1 であるような配列変数 **valimesh** を作ります。そして 50 行目では，値が 1 である **valimesh** の要素の 1 番目の添え字のリスト **y** と 2 番目の添え字のリスト **x** を作ります。ちょっとわかりにくいですが，**y** と **x** は同じ長さのリストで，これらの対 **[y[0], x[0]]**，**[y[1], x[1]]**，**[y[2], x[2]]**，**[y[3], x[3]]**，・・・・を要素番号とする **valimesh** の要素の値は必ず **1** となり，このリストにない番号の要素の値は必ず無効値となります。

49 行目と 50 行目の両方には，Python の数値計算パッケージ **numpy** に含まれる **where** 関数が使用されています。この関数は，以下に示す 2 通りの使用法があります。

用法 1：　**配列 = numpy.where（配列に対する論理式，真の場合の値，偽の場合の値）**
用法 2：　**配列 $_1$, 配列 $_2$ = numpy.where（配列に対する論理式）**

一つ目の用法では，論理式に書かれた配列と同じサイズの配列を返します。この際，引数の論理式を満たしている要素には真の場合の値，満たしていない要素には偽の場合の値が格納されます。二つ目の用法では，引数の論理式を満たす配列要素の要素番号の配列（並び）が返されます。返される要素番号の配列の数は，論理式に書かれた配列の次元数と一致します。

54 行目から 58 行目で計算に使用する情報や配列を準備します。移植後日数で表した出穂日は，配列変数 **dat** に入れます。57 行目の **zeros** 関数は，引数のサイズの配列を生成しすべての要素に 0 を代入する関数です。58 行目で配列変数 **valimesh** を掛けているのは，処理対象外のメッシュに無効値を埋め込むためです。

発育日数の計算は 61 行目から始まる for 文で実行されます（図 36）。for 文は，反復回数があらかじめ分かっているときに用いる繰り返しの指示文です。この文により 62 行目から 70 行目までが繰り返されます。for 文の中に **range** という関数が用いられていますが，これは，0，1，2，3，・・・と，0 から始まって引数より 1 小さい数までの（引数個の）整数からなるリストを作成する関数です。これで作られたリストの値が順次 **i** に代入され 62 行目以降で使用されます。繰り返し範囲の中，65 行目にも別な for 文が使われています。この文が指示する繰り返し範囲は 66 行目から 69 行目までです。この for 文にも **range** 関数が使われていますが，引数が二つです。この場合は，第 1 引数から始まって第 2 引数より 1 小さい数までのリストが作られます。

日平均気温と品種パラメータから日々の発育速度を求めて積算する処理は，66 行目で実行されています。積算した後，積算値 **DVI** が 1 を超えているかどうかを 67 行の if 文で調べ，まだ達していなければ次の **d** について 66 行以下を繰り返します。**DVI** が 1 を超えていたらその時の **d** を配列変数 **dem** に保存して **d** に関する繰り返しを終了し，**i** に次の値を入れて（次のメッシュに対象を移して）62 行目からの処理を実行します。このようにして，**DVI** が 1 を超えるまでの繰り返し計算を，計算対象メッシュすべてに対し実行して配列変数 **dat** を作ります。

コラム 8

for 文

Python で使われる繰り返しのうち，for 文は，繰り返す回数や対象が事前に決まっている場合に用いられます。例として，事前に決まっている数 1，3，5，7，9 を順次足し上げて最終結果を表示させるスクリプト（文の集合）を下に示します。

```
sum = 0
for i in [1,3,5,7,9]:
   sum = sum + i
print(sum)
```

for 文の末尾にコロンを付けることを忘れないでください。繰り返す部分は同じ深さのインデントにすることで示します。

次に，5 個の要素からなる配列変数 x があって，この中身を全部足す場合を考えましょう。この方法はいくつかあって，最もシンプルに書くと次のようになります。

```
sum = 0
for i in [0,1,2,3,4]:
   sum = sum + x[i]
print(sum)
```

でも，これではまったく融通が利きません。より実用的には，配列を構成する要素の数を調べる関数 **len** と，1 ずつ増える整数のリストを作る関数 range を用いて次のようにします。

```
sum = 0
j = len(x)
for i in range(j):
   sum = sum + x[i]
print(sum)
```

ここで，2 行目と 3 行目はひとまとめにして，「**for i in range(len(x)):**」としても構いません。

このようにすれば，配列 x が持つ要素の個数が何個であっても対応できます。なお，range(n) は，**[0,1,2,・・n-1]** と，0 から始まり引数 n より 1 小さい数までのリストを作ることに注意してください。

図 36 には示していませんが，サンプルプログラムの 72 行目より下には，「2 気象分布図の作成」で説明した分布図を作成するスクリプトが書かれていて，**D2D** に **dat** が渡されています。結果を図 36 右下に示します。きれいな図が描けました。県下一斉に特定品種が移植されることはあり得ませんが，自らの生産圃場における移植日と移植品種が分かっている生産農家にとっては，このような図は参照データとして有益です。

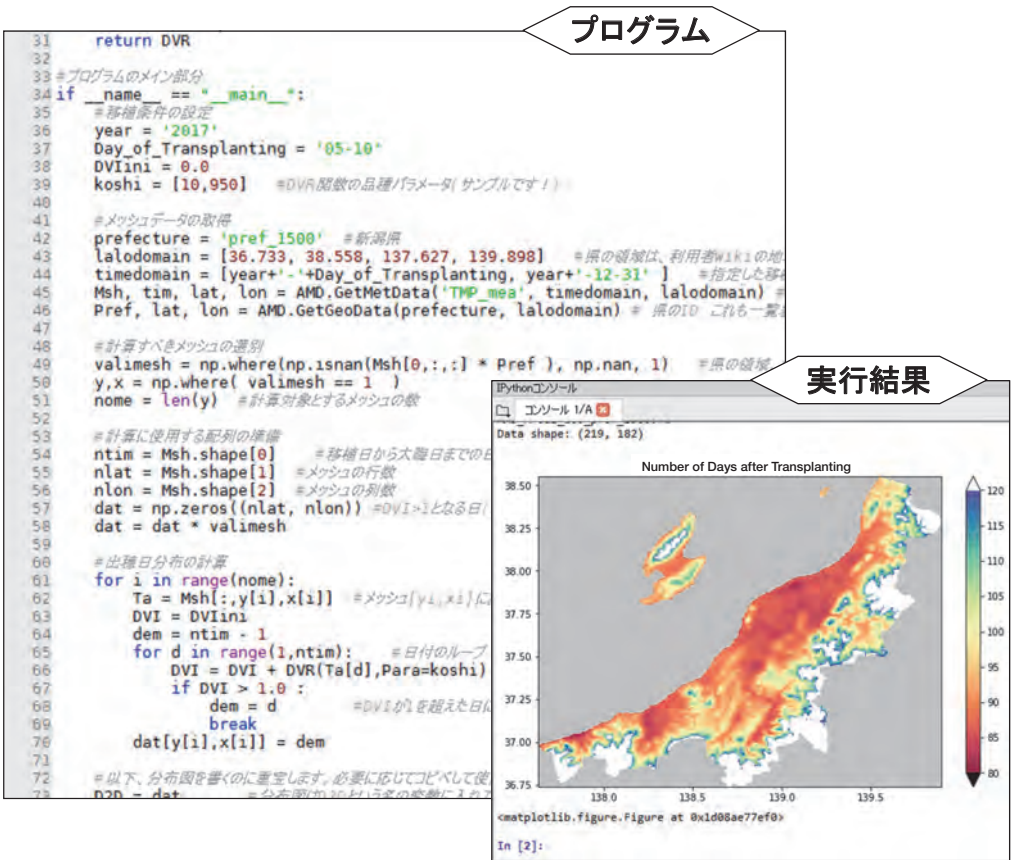


図 36. サンプルプログラム sample_RiceDevel.py の 31 行～ 72 行目，ならびに，このプログラムの実行結果の図

さて，このプログラムでは，出穂日を移植後日数で示していますが，できれば何月何日といった日付で分布図を作りたいところです。そこで，日付の分布図を描かせるようにサンプルプログラムを修正してみました。これは利用者 Wiki に，**sample_RiceDevel-b.py** として掲載されています。このプログラムの抜粋を図 37 に示します。日時オブジェクトを格納する特殊な配列変数 **dates** を導入し（57 行目），これに出穂日の日付を格納します（71 行目）。

残念なことに，「2 気象分布図の作成」で使用した分布図を描画するスクリプトは，今回は使えません。描画させる配列が特殊な配列であることと，カラーバーに付随させるスケールに日付の書式を指定しなければならないからです。**sample_RiceDevel-b.py** には，これらに対応した描画スクリプトが 74 行目から 103 行目にかけて書かれているので，日時オブジェクトの分布図を表示させる時にはこれを使いまわしてください。

また，このプログラムの末尾には，**PutKMZ_Map** 関数が追加されていて，分布図を Google Earth 上に表示することができるファイル（Date_of_Heading.kmz）が出力されるようになっています。Google Earth が利用できる環境の方は，このファイルをダブルクリックしてみて下さい。表示される分布図は，透過度を調整できるので着目する地点を把握するのが極めて容易です（図 38）。

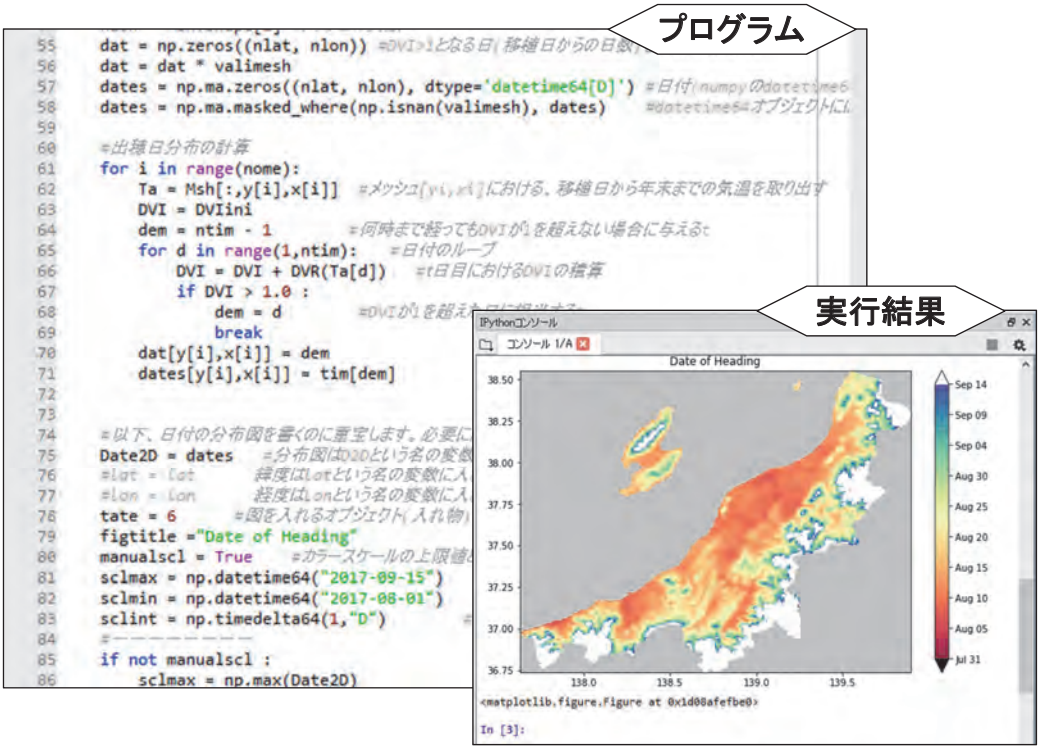


図 37.　サンプルプログラム sample_RiceDevel-b.py の 55 行～ 86 行目，ならびに，このプログラムの実行結果の図

このプログラムでは，分布図の凡例が日付で示される。

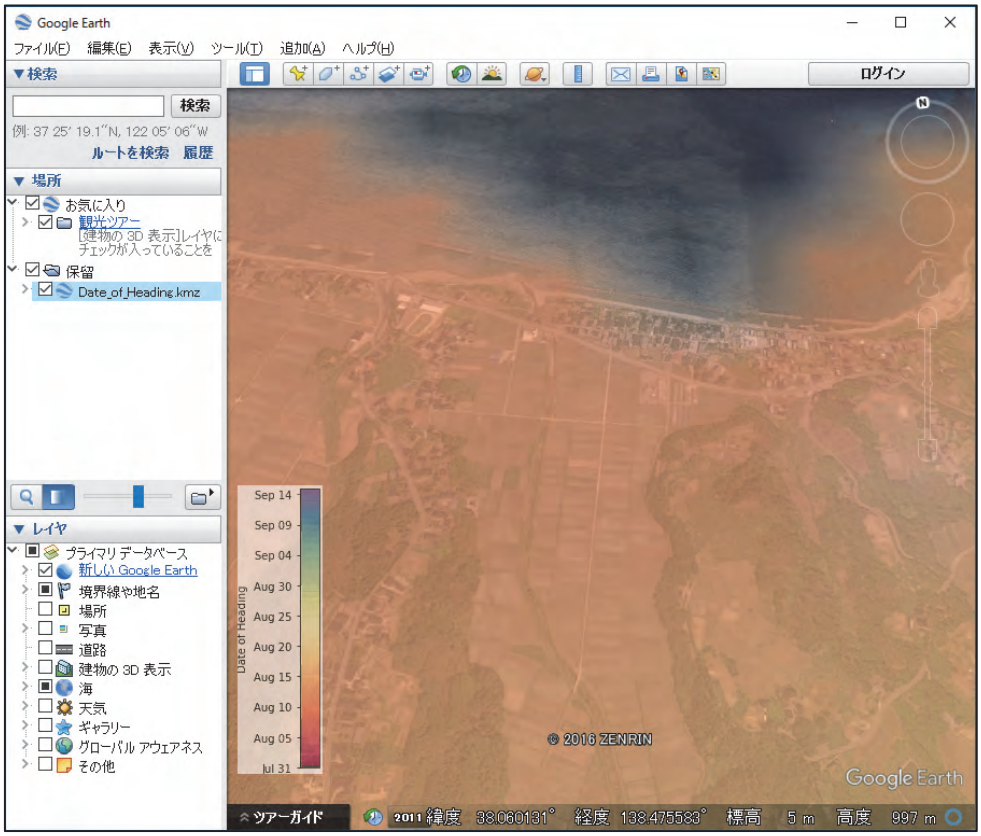


図 38.　サンプルプログラム sample_RiceDevel-b.py で作成されるファイルを用いて，出穂日分布図を Google Earth 上に表示した画面

11　CSV ファイルに整理した試験圃場における出穂日の予測

圃場の位置や移植日，品種などが異なる複数の作付けそれぞれに対する出穂日と，去年と比べたときの早晩を個別に予測してみましょう。圃場の位置など栽培情報は，表計算ソフトで管理されていることがほとんどなので，出穂日の予測に必要な情報を CSV ファイルから読み取って計算し，結果も CSV ファイルで出力させてみます。栽培情報のサンプルとして，図 39 に示す表を CSV ファイルに落としたものを使用します（**sample_RiceDevelPoi-data.csv**）。図 40 に，これを読み取って出穂日を予測するサンプルプログラムラム **sample_RiceDevelPoi.py** の前半部分を示します。

栽培情報の読み取りは，**AMD_Tools3** に含まれる **GetCSV_Table** 関数を使用して 28 行目で実行します。この関数は，CSV ファイルの 1 行目を見出し，それ以外の行をデータ本体とみなしてそれぞれをリストで返すので，式の左辺では二つの変数で受けます。見出しが格納された配列変数 **header** の中身は下のようになっています：**['ID', '地区', '地点名', '緯度', '経度', '移植時 DVI', '品種パラメータ 1', '品種パラメータ 2', '移植日']**。そして，データ本体が格納された，配列変数 **body** には，特定行における各列からなるリストが，各行について集められたリストになっています。例として，**body[0]** の中身は **['1', '谷和原', 'A-3', '35.69', '139.76', '0', '10', '950', '2017/5/5']** です。また，配列やリストの要素数を数える関数 **len** を適用して **len**（**body**）とすると，結果は 9 となります。**body** から特定のアイテム，例えば，栽培 ID5，谷和原地区の A-7 圃

ID	地区	地点名	緯度	経度	移植時DVI	品種パラメータ1	品種パラメータ2	移植日
1	谷和原	A-3	35.69	139.76	0	10	950	2017/5/5
2	谷和原	A-4	36.38	140.4667	0	10	950	2017/6/1
3	谷和原	A-5	36.54833	139.8683	0	10	950	2017/5/23
4	谷和原	A-6	36.405	139.06	0	10	950	2017/5/10
5	谷和原	A-7	36.15	139.38	0	10	950	2017/5/10
6	観音台	KR01	35.73833	140.8567	0	10	1000	2017/5/18
7	観音台	KR02	35.43833	139.6517	0	10	1000	2017/5/18
8	観音台	KR03a	35.66667	138.5533	0	10	1010	2017/5/26
9	観音台	KR03b	36.66167	138.1917	0	10	1010	2017/5/26

図 39.　サンプルデータ sample_RiceDevelPoi-data.csv の内容
サンプルプログラム **sample_RiceDevelPoi.py** はこのファイルを読み込む

プログラム

```
23 import AMD_Tools3 as AMD
24 from datetime import datetime, timedelta
25 from sample_RiceDevel import DVR  #他のプログラムで開発した関数を利用することができます。
26
27 # 地点リストの読み込み。
28 header, body = AMD.GetCSV_Table('sample_RiceDevelPoi-data.csv')  #関数GetCSV_Tableは
29 nore = len(body)                                    #リストに掲載される地点数
30 sid = [body[oo][0] for oo in range(nore)]           #これがリスト内包表記です
31 slat = [float(body[oo][3]) for oo in range(nore)]
32 slon = [float(body[oo][4]) for oo in range(nore)]
33 DVIini = [float(body[oo][5]) for oo in range(nore)]
34 p1 = [float(body[oo][6]) for oo in range(nore)]
35 p2 = [float(body[oo][7]) for oo in range(nore)]
36 dtp = [datetime.strptime(body[oo][8], '%Y/%m/%d') for oo in range(nore)]
37
38 dem = []   #今年の出穂日の日付の要素番号(何個目の日付か)のための空のリスト
39 deml = []   #前年の出穂日の日付の要素番号(何個目の日付か)のための空のリスト
40 for s in range(nore):
41     print(sid[s])
42     para = [p1[s],p2[s]]    #発育速度関数に与える品種パラメータ
43     #前年の出穂日の計算
44     tbeg = datetime(dtp[s].year-1, dtp[s].month, dtp[s].day)      #丁度1年前の同じ日の
```

図 40.　サンプルプログラム sample_RiceDevelPoi.py の 23 行～ 44 行目

場における移植日を取り出すときは，**body[4][8]** などとします。なお，**GetCSV_Table** 関数は，内容にかかわらず，文字列のリストを返すことに注意してください。

さて，**body** に格納された情報を呼び出して使う際，**body[○][△]** の形のままだと大変面倒なうえ，出来上がったプログラムは大変読みにくいものとなってしまいます。そこで，栽培の ID，圃場緯度，圃場経度等，データごとのリストに作り直すことにします。サンプルプログラムの 30 行目から 36 行目の部分でリストの組み換えと数値や日時オブジェクト等への型の変換が行われています。これらの文は，Python 独特の「リスト内包表記」と呼ばれる文法で書かれています。例えば，**body** から圃場緯度のリスト **slat** を作り出す処理は 31 行目の文だけで記述されています。

図 41 に，このサンプルプログラムの後半部分を示します。CSV ファイルから読み込まれた栽培情報から，出穂日を順次予測する反復処理は 40 行目から 70 行目にかけて記述されています。このサンプルプログラムは，今年の出穂日だけでなく，一年前の同月同日を移植日とする計算も行い出穂日の早晩を求めるので，反復の処理の中には，出穂日を計算するよく似たスクリプトが 2 回繰り返されています（44-56 行目と 58-70 行目）。

先ほどはリストの新規作成をリスト内包表記で行いましたが，リストのオーソドックスな新規作成の方法は，まず初めに空のリストを作成しこれに要素を一つずつ追加してゆく方法です。出穂日のリストはこの方法で作ります。38 行目と 39 行目で今年の出穂日と前年の出穂日を格納するのに使用する空のリスト **dem** と **deml** をそれぞれ定義します。そして，計算の結果求められた出穂日がそれぞれ 55 行目と 69 行目でリストに追加されます。

プログラム

```
38 dem = []    #今年の出穂日の日付の要素番号(何個目の日付か)のための リ空
39 deml = []    #前年の出穂日の日付の要素番号(何個目の日付か)のための リ空のリスト
40 for s in range(nore):
41     print(sid[s])
42     para = [p1[s],p2[s]]    #発育速度関数に与える品種パラメータ
43     #前年の出穂日の計算
44     tbeg = datetime(dtp[s].year-1, dtp[s].month, dtp[s].day)    #丁度1年前の同じ日の
45     tend = tbeg + timedelta(days=90)        #その日から90日の時刻オブジェクト
46     timedomain = [tbeg.strftime('%Y-%m-%d'),tend.strftime('%Y-%m-%d')]
47     lalodomain = [slat[s],slat[s],slon[s],slon[s]]
48     Ta,tim,lat,lon = AMD.GetMetData('TMP_mea', timedomain,lalodomain)
49     Ta = Ta[:,0,0]
50     noda = len(tim)
51     DVI = DVIini[s]
52     for d in range(noda):
53         DVI = DVI + DVR(Ta[d],Para=para)    #DVRの積算
54         if DVI > 1.0 :
55             deml.append(d)  #出穂日(要素番号)の記録
56             break
57     #今年の出穂日の計算
58     tbeg = dtp[s]
59     tend = tbeg + timedelta(days=90)        #その日から90日の時刻オブジェクト
60     timedomain = [tbeg.strftime('%Y-%m-%d'),tend.strftime('%Y-%m-%d')]
61     lalodomain = [slat[s],slat[s],slon[s],slon[s]]
62     Ta,tim,lat,lon = AMD.GetMetData('TMP_mea', timedomain,lalodomain)
63     Ta = Ta[:,0,0]
64     noda = len(tim)
65     DVI = DVIini[s]
66     for d in range(noda):
67         DVI = DVI + DVR(Ta[d],Para=para)    #DVRの積算
68         if DVI > 1.0 :
69             dem.append(d)  #出穂日(要素番号)の記録
70             break
71 diff = [dem[oo]-deml[oo] for oo in range(nore)]    #前年に対する推定日との遅速を計算を全地
72
73 with open('result.csv', 'wt') as f:    #結果書き出しのためにファイルをオープン
74     f.write(header[0]+','+'"予測出穂日('+datetime.today().strftime('%m/%d')
75                                 +'現在)",'去年との遅速(日)'\n')    #見出し行の出力
76     for s in range(nore):
77         f.write(sid[s]+','+tim[dem[s]].strftime('%Y/%m/%d')+','+str(diff[s]) +'\n')
```

図 41．サンプルプログラム sample_RiceDevelPoi.py の 38 行～ 77 行目

データ配信サーバーから一年前の同月同日のデータを入手するには，今年の移植日の文字列から一年前の日付の文字列を作り出す必要があります。この方法は何通りか考えることができますが，時系列の気象データを活用する上では日付の計算を避けて通るわけにはゆかないので，Python における日付計算を理解するために，このプログラムでは，敢えて正攻法を採用します。すなわち，CSV ファイルから文字列で取り出した移植日情報を日時オブジェクトに変換（36 行目）し，次に，**year** メソッド，**month** メソッド，**day** メソッドを用いてこれから年，月，日を整数として取り出してこれらを **datetime** 関数に与えて 1 年前の同月同日の日時オブジェクトを生成（44 行目）します。そして，**strftime** メソッドでこれを日付文字列に変換（46 行目）して **GetMetData** 関数に与えます（48 行目）。

ところで，皆さんは，53 行目と 67 行目で使用されている発育速度関数（**DVR**）がこのサンプルプログラムのどこで定義されているかお分かりでしょうか。「10 発育指数法を用いた水稲の出穂日分布図の作成」で示したサンプルプログラムには，**DVR** 関数を定義する部分がプログラム

コラム 9

日付・時刻・時間間隔の取り扱い

気象データを処理する際，日付や時刻の計算や表示を避けて通ることはできません。しかし，日数の計算はなかなか面倒なうえ，同じ日付でも何種類もの表示形式があります。これらに対応するために，Python には日付と時刻を示すものとして datetime オブジェクト，時間の間隔を示すものとして timedelta オブジェクトが用意されています。オブジェクトとは，一つの数値では表現できないような対象をプログラムで効率よく取り扱うために考えられた概念で，表現に必要な複数の数値（これらを属性といいます）をひとまとめにして保持し，さらにそれらを用いる計算プログラム（メソッドと呼びます）が定義されているものです。datetime オブジェクトの場合，year，month，day，hour，minute，second，microsecond をひとかたまりにして保持し，年月日を所定の書式の文字列にするメソッドなどが定義されています。まあ，用語や概念は必要になったときに少しずつ理解してください。以下では，datetime オブジェクトと timedelta オブジェクトの利用方法を説明します。

datetime オブジェクトや timedelta オブジェクトを利用するには，外部モジュール datetime をインポートする必要があります。

from datetime import datetime, timedelta

モジュール名とオブジェクトを作り出すもの（クラス）の名前の両方に同じ「datetime」という名前が使われていますが，両者は別物です。使用例を下に示します。

2017 年 1 月 1 日 0 時 0 分の時刻 t1 を作る。	**t1 = datetime(2017,1,1,0,0,0)** または **t1 = datetime(2017,1,1)** または， **t1 = datetime.strptime('2017/1/1', '%Y/%m/%d')**
現在の時刻 t2 を作る。	**t2 = datetime.today( )**
1 日間の時間間隔 d1 を作る。	**d1 = timedelta(days=1)**
h 時間の時間間隔 d2 を作る。	**d2 = timedelta(hours=h)**
現在の k 日後の時刻 t3 を計算する。	**t3 = datetime.today( ) + d1 * k** または， **t3 = datetime.today( ) + timedelta(days=k)**
2017 年 1 月 1 日 0 時 0 分（t1）から t2 までの時間間隔 d3 を計算する。	**d3 = t2 – t1**

datetime オブジェクトが抱えている年や月などの属性は，オブジェクト名の後に属性名をピリオド（.）で繋げると参照することができます。上で例示した t1 に対し，t1.year とすると，2017 という整数が得られます。なお，timedelta の属性は，days と seconds です。

datetime オブジェクトのメソッドで利用頻度が特に高いのは特定の書式の文字列を判読して datetime オブジェクトを作り出す strptime メソッドと，datetime オブジェクトが示す時刻から，指定する書式の文字列を作る strftime メソッドです。前者については，上に用例を示しています。後者については下

に例を示します。

時刻 t1 を「yyyy/mm/dd」形式で示す文字列 s1 を作る。	**s1 = t1.strftime('%Y/%m/%d')**
時刻 t1 を「yy-mm-dd hh:mm」形式で示す文字列 s2 を作る。	**s2 = t1.strftime('%y-%m-%d %H:%M')**

上の表で，日付の書式に使われている記号について補足します。文字列の中の「%○」のところに，決められた年や月の数字が入ります。それ以外の「/」や「-」，空白などは任意です。ただし，残念ながら日本語は使えません。「%○」が示す内容を，2017 年 2 月 3 日 9 時 5 分の場合の例で示します。

%Y ：2017（4 桁の西暦年）
%y ：17　（西暦年の下二桁）
%m：02　（2 桁の月）
%d ：03　（2 桁の日）
%H：09　（2 桁の時）
%M：05　（2 桁の分）
%h ：Feb（月の英語略名）
%S ：00　（2 桁の秒）

の始めの方にありましたが，このサンプルプログラムに **def** 文は存在しません。

答えは 25 行目です。このサンプルプログラムは，サンプルプログラム **sample_RiceDevel.py** で定義されている **DVR** 関数を借用しているのです。この仕組みは，例えば，データ配信サーバーからメッシュデータを取り込む **GetMetData** 関数を毎回定義せずに，**AMD_Tools3** から呼び出して使用するのとまったく同じ作法です。関数を個々のプログラムで定義するのと特定のファイルに定義して共有するのとはそれぞれに一長一短があるので，利用場面を考えて使い分けてください。

今年と前年の出穂日の計算が終了したらそれらの差を 71 行目で計算します。この計算も，リスト内包表記を使って 1 行で済ませます。これで出力すべき情報が揃ったので，73 行目以降でこれを CSV ファイルとして出力します。

Python では，**open** 関数でファイルを開きますが，このサンプルプログラムでは，with 文を用いて，ディスクが書き込み禁止等何らかの理由でファイルのオープンが失敗した場合にそれ以降をキャンセルすることができるようにしています。73 行目の文を日本語で解説するとつぎのようになります。「テキスト文字列を書き出すので **'result.csv'** という名前のファイルを作りなさい。プログラムからは **f** の名でこのファイルを扱います。ファイルが作れなければ以下は止めなさい」。

f はファイルオブジェクトで，write メソッドを使ってこれに書き出します。74 行目では見出し行が出力されます。77 行目には出穂日とその遅速を出力する文が書かれており，これが for 文により栽培情報の数だけ繰り返されます。

このサンプルプログラムにより作成される CSV ファイルの内容を図 42 に示します。

ID	予測出穂日(05/22現在)	去年との遅速(日)
1	2017/8/11	2
2	2017/8/10	6
3	2017/8/10	3
4	2017/8/11	3
5	2017/8/10	2
6	2017/8/21	7
7	2017/8/11	3
8	2017/8/6	2
9	2017/8/14	4

図 42.　サンプルプログラム sample_RiceDevelPoi.py によって作成されたファイル result.csv の内容

コラム 10

リスト内包表記

Python で何が変かといえば，リスト内包表記ほど変てこなものはありません。以下のスクリプトの 2 行目の右辺を見てください。

```
from datetime import datetime, timedelta
week = [datetime.today()+timedelta(days=i) for i in range(7)]
```

これがリスト内包表記です。この一文で，今日から一週間後までの日付オブジェクトのリスト **week** が作られます。リスト内包表記を用いずに同じものを作るには，for 文を使って以下のようにします。

```
from datetime import datetime, timedelta
week = [ ]
for i in range(7):
  week.append(datetime.today()+timedelta(days=i))
```

ここで，append はリストのメソッドで，要素を一つ追加するという機能です。
Python ではいろいろな場面でリストが使われるので，ものぐさをしてリストを作る方法も用意されているようです。

12　日長の計算

農作物のいくつかは，発育が夜の長さの影響を受けます。このため，これらの作物の発育を予測するには，気温などの気象データに加え栽培期間における日々の日長も必要となります。利用者 Wiki で配布する **DayLength3** モジュール（DayLength3.py）に含まれる **daylength** 関数を使用すると，指定したメッシュ範囲における指定した期間の日長の時空間分布を計算することができます。サンプルプログラム **sample_daylength.py** を例に日長の計算方法を説明します。

サンプルプログラムの 16 行目から 21 行目で，このプログラムで使用するモジュールや関数をまず宣言します。次に，26 行目から 28 行目で，茨城県の県域データを取得します。そして，30 行目で日長の計算をし，最後に，33 行目で県域データと日長データを掛け合わせて茨城県以外を無効としたものを作って，34 行目から 58 行目の文で分布図を描きます。これにより 2017 年 6 月 21 日における茨城県における日長の分布図が作成されます（図 43）。

daylength 関数は，一定の期間における一定のメッシュ範囲の日長を一気に計算することを想定して作られているので，関数に与える日付，緯度，経度の引数には一次元の配列を渡さなければなりません。緯度と経度の配列については，28 行目で **GetGeoData** 関数により返される **lat** と **lon** がそのまま使えるので新たに作る必要はありませんが，日時の配列については，**GetGeoData** 関数では生成されないので，**datetime** 関数を使って 29 行目で作成します。このとき，**datetime** 関数を大括弧で括っていることに注意してください。大括弧で括ることで，要素が 1 つだけの一次元配列の体裁となります。

なお，プログラムで，**GetMetData** 関数を使う場合は，これが返す日付，緯度，経度の配列をそのまま **daylength** 関数に渡せば，気象データと全く同じ時空間範囲について日長が計算されます。

次いで，サンプルプログラムは，61 行目から 91 行目の部分を実行し，北緯 36.0566 度・東経 140.125 度における 2017 年の日長の年変化グラフを作成します（図 44）。今度は，**daylength** 関数に与える引数をすべて用意します。まず，365 個の要素を持つ日時の配列 **tim** は，66 行目においてリスト内包表記により作成されます。緯度と経度の配列は，62 行目と 63 行目で設定した浮

プログラム

```
16 import AMD_Tools3 as AMD
17 import DayLength3 as DL
18 import numpy as np
19 import matplotlib.pylab as plt
20 import matplotlib.dates as md
21 from datetime import datetime, timedelta
22
23
24 # ーー茨城県における2017年6月21日の日長時間の分布図の作成ーー
25
26 element = 'pref_0800' #地理情報の指定、茨城県
27 lalodomain = [ 35.733, 36.950, 139.690, 140.860]  #茨城県が収まる領域
28 Pref,lat,lon = AMD.GetGeoData(element,lalodomain)
29 tim = [datetime(2017,6,21)]
30 ld = DL.daylength(tim,lat,lon)
31 ld = ld[0,:,:]  #3次元の入れ物で返さ
32
33 D2D = ld * Pref  #分布図は2Dという名の
34 tate = 6         #図を入れるオブジェクト(
35 figtitle = "Day length over Ibaraki
36 manualscl = True     #カラースケールの上
37 sclmax  = 14.75                #最大値
38 sclmin  = 14.55                #最小値
39 sclint  = 0.005                #色の刻
```

実行結果

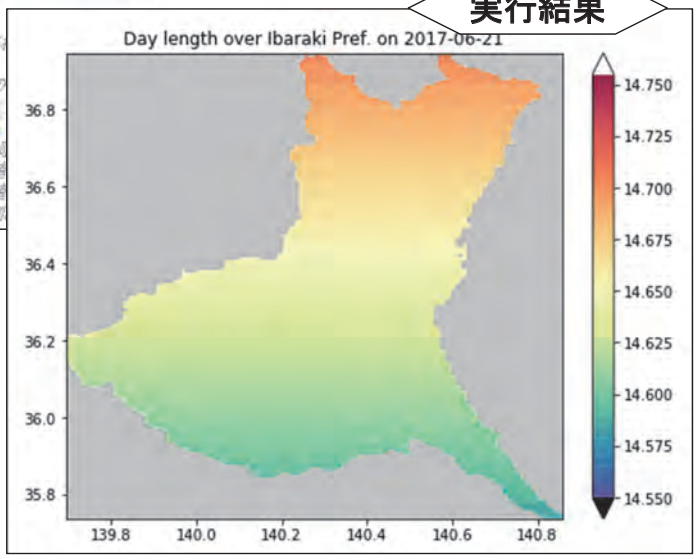


図 43.　サンプルプログラム sample_daylength.py の 16 行～ 39 行目，ならびに実行結果の図

プログラム

```
60
61 # ーー茨城県つくば市館野における2017年の日長時間の年変化グラフーー
62 lat = 36.0566   #茨城県つくば市館野の緯度
63 lon = 140.125   #茨城県つくば市館野の経度
64
65 # 日付のリストの作成
66 tim = [datetime(2017,1,1)+timedelta(days=oo) for oo in range(365)]  #2017年1月1日
67 ld = DL.daylength(tim,[lat],[lon]) #daylength関数には日付、緯度、経度を配列で与えなけれ
68 ld = ld[:,0,0] #daylength関数は結果を3次元の入れ物(配列)で返すので、これを1次元の入れ物に
69
70 #折れ線グラフの作成
71 figtitle = 'N'+str(lat)+', E'+str(lon)
72 fig = plt.figure(num=None, figsize=(12, 4))
73 # 目盛
74 ax = plt.axes()
75 xmajoPos = md.DayLocator(bymonthday=[1])
76 xmajoFmt = md.DateFormatter('%m/%d')
77 ax.xaxis.set_major_locator(xmajoPos)
78 ax.xaxis.set_major_formatter(xmajoFmt)
79 xminoPos = md.DayLocator()
80 ax.xaxis.set_minor_locator(xminoPos)
81 # 折れ線
82 ax.plot(tim, ld, 'r')
83 # ラベル、タイトル
84 plt.xlabel('Date')
85 plt.ylabel('Day Length [hours]')
86 plt.title(figtitle)
87 # 画像ファイル
88 plt.savefig('result'+'.png', dpi=600)
89 # 表示
90 plt.show()
91 plt.clf()
92
```

実行結果

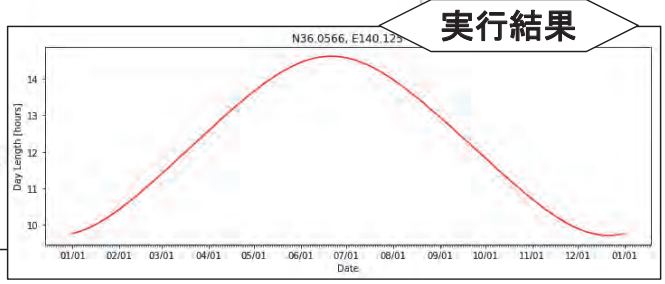


図 44.　サンプルプログラム sample_daylength.py の 60 行～ 92 行目，ならびに実行結果の図

動小数を **daylength** 関数に渡すところで括弧で括り，配列の体裁にしています。

daylength 関数は，計算した日長を常に三次元配列の形式で返します。このため，31 行目や 68 行目で結果を二次元や一次元の配列に入れなおしてから描画処理を行います。

コラム 11

日の出，日の入，南中の時刻

DayLength3 モジュールには，太陽位置の計算をする SUN オブジェクトが格納されていて，これを利用すると，太陽の日の出時刻や南中時刻などを求めることができます。

SUN オブジェクトは，time（日時：日時オブジェクト），latitude（緯度：十進小数表記の浮動小数），longitude（経度：十進小数表記の浮動小数）の 3 つの属性を持ちます。そして，この観測条件における太陽の赤緯・赤経，太陽方位角，太陽高度，太陽の視半径，太陽の大気差などがメソッドとして用意されています。

SUN オブジェクトを利用するには，これを DayLength3 モジュールからあらかじめインポートしておきます。また，多くの場合，日時オブジェクトの操作も必要なのでこれもインポートします。

```
from DayLength3 import SUN
from datetime import datetime
```

北緯 36.0566 度, 東経 140.1250 度の地点における 2017 年 1 月 1 日，正午（日本標準時）の SUN オブジェクト sun1 は，次のように作成します。

```
sun1 = SUN(datetime(2017,1,1,12,0,0), 36.0566, 140.1250)
```

または，

```
sun1 = SUN( )    # 何も指定しないで SUN オブジェクトを作成すると，2000 年 1 月 1 日 0:00JST，北緯 35 度，東経 135 度が設定されます。
sun1.settime(datetime(2017,1,1,12,0,0))
sun1.setlat(36.0566)
sun1.setlon(140.1250)
```

ここで，settime，setlat，setlon はいずれもメソッドで，それぞれ，時刻，緯度，経度を設定します。以下に，主なその他のメソッドを示します。

sun1 における太陽方位角（0 ～ 360）を求める。	**sun1.azimus( )**
sun1 における太陽高度角（0 ～ 90）を求める。	**sun1.elevation( )**
sun1 における太陽の赤緯（-90 ～ 90）を求める。	**sun1.delta( )**
sun1 における太陽の赤経（0 ～ 360）を求める。	**sun1.alfa( )**
sun1 の日における日の出時刻を求め，time をその日時にセットするとともに，24 時間を 1 とする時刻値を返す。	**sun1.sunrise( )**
sun1 の日における日の入り時刻を求め，time をその日時にセットするとともに，24 時間を 1 とする時刻値を返す。	**sun1.sunset( )**
sun1 の日における南中時刻を求め，time をその日時にセットするとともに，24 時間を 1 とする時刻値を返す。	**sun1.meridian( )**
sun1 の日における見かけの太陽高度角が午前中に deg［度］となる時刻を求め，time をその日時にセットするとともに，24 時間を 1 とする時刻値を返す。	**sun1.sunrise(elevangle=deg)**
sun1 の日における見かけの太陽高度角が午後に deg［度］となる時刻を求め，time をその日時にセットするとともに，24 時間を 1 とする時刻値を返す。	**sun1.sunset(elevangle=deg)**

参考として，**SUN** オブジェクトのメソッドを利用して求めた，北緯 36.0566 度・東経 140.1250 度（茨城県つくば市内）における 2017 年の日の出時刻（下部の黒線），日の入時刻（上部の黒線），南中時刻（中央部の赤線）のグラフを示します。

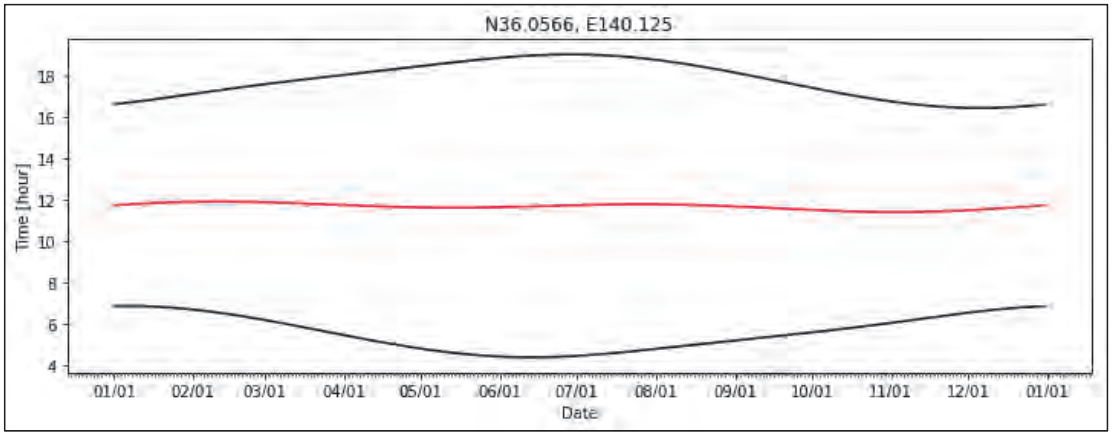


図. DayLength3 モジュールで計算した，北緯 36.0566 度・東経 140.1250 度（茨城県つくば市内）における 2017 年の日の出時刻（下部の黒線），日の入時刻（上部の黒線），南中時刻（中央部の赤線）

Ⅴ　メッシュ農業気象データ利用ツールリファレンス

メッシュ農業気象データシステムでは，Python を利用してメッシュ農業気象データの取得や処理に利用できる便利な関数や機能を二つのモジュールで用意しています。一つ目は **AMD_Tools3** モジュールで，これにはデータの入出力を中心とする関数や機能が収められています。もう一つは **DayLength3** モジュールで，これには日長（昼間の長さ）を全国について正確に求める関数が収められています。これらの実体はプログラムファイルで，それぞれ，**AMD_Tools3.py** と **DayLength3.py** です。これらは利用者 Wiki からダウンロードすることができるので，あらかじめ入手し，プログラムファイルと同じフォルダに配置しておいてください。

プログラムにおいては，関数名とそれが収められているモジュール名を import 文で宣言して利用します。例えば，**AMD_Tools3** モジュールに収められている **GetMetData** を使用する場合は，以下のように宣言します。

```
from AMD_Tools3 import GetMetData
```

利用する関数はコンマで区切って複数宣言することができます。

```
from AMD_Tools3 import GetMetData, PutKMZ_Map, lalo2mesh
```

利用する関数を特定せず，モジュールの利用だけを宣言する方法もあります。この場合は，関数を使用する際，頭にモジュールの略称を付加しなければなりません。

```
import AMD_Tools3 as AMD
Msh,tim,lat,lon = AMD.GetMetData ('TMP_mea',timedomain,lalodomain)
```

以下に，**AMD_Tools3** モジュールおよび **DayLength3** モジュールに収められている関数とその使用法を示します。

1　AMD_Tools3 モジュール

(1) GetMetData 関数

GetMetData 関数は，メッシュ農業気象データを取り込む関数です。

書式：

```
GetMetData (element, timedomain, lalodomain, area=None, cli=False, namuni=False,
    url='http://mesh.dc.affrc.go.jp/opendap')
```

引数（必須）：

element：気象要素記号で，**'TMP_mea'** などの文字列で与える。

timedomain：取得するデータの日付の範囲で，**['2008-05-05', '2008-05-05']** のような文字列の 2 要素リストで与える。特定の日のデータを取得するときは，二カ所に同じ日付を与える。

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

引数（必要に応じ指定）：

cli：平年値を取得するときに **cli=True**（または **1**）として指定する。省略した場合は観測値や予報値が返される。

namuni：気象要素の正式名称と単位をデータと共に取り出すときに **namuni=True**（または **1**）として指定する。この指定により戻り値の数は 6 つ（気象値，日付，緯度，経度，正式名称，単位）になる。省略した場合は，戻り値の数は 4 つ（気象値，日付，緯度，経度）となり名称等は返されない。

area：データを読み出すエリア（Area1 ～ Area6）を明示的に指定するときに **area='AreaX'** と指定する。省略した場合は自動的に選ばれる。

url：メッシュ農業気象データの取得場所を明示的に指定するときに **url=** ＜ファイルへのパス＞として指定する。省略した場合はデータ配信サーバーが指定される。ローカルにあるファイルを指定するときは，**AreaX**（X=1 ～ 6）の直上（通常は '・・・/AMD'）を指定する。

戻り値：

第 1 戻り値：指定した気象要素の並び（浮動小数の三次元（日付×緯度×経度）配列）。

第 2 戻り値：切り出した気象データの日付の並び（日時オブジェクトの一次元配列）。

第 3 戻り値：切り出したメッシュの中心緯度の並び（浮動小数の一次元配列）。

第 4 戻り値：切り出したメッシュの中心経度の並び（浮動小数の一次元配列）。

第 5 戻り値（**namuni=True** のときのみ）：気象要素の正式名称（文字列）。

第 6 戻り値（**namuni=True** のときのみ）：気象要素の単位（文字列）。

使用例：北緯 35 度，東経 135 度の地点の 2008 年 1 月 1 日～ 2012 年 12 月 31 日の日最高気温を取得する場合。

```
import AMD_Tools3 as AMD
timedomain = ['2008-01-01', '2012-12-31']
lalodomain = [35.0,  35.0, 135.0, 135.0]
Tm, tim, lat, lon = AMD.GetMetData ('TMP_max', timedomain, lalodomain)
```

(2) GetGeoData 関数

GetGeoData 関数は，土地利用区分などの地理情報を取り込む関数です。

書式：

```
GetGeoData (element, lalodomain, area=None, namuni=False, url='http://mesh.dc.affrc.
    go.jp/opendap')
```

引数（必須）：

element：地理情報記号で，**'altitude'** などの文字列で与える

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

引数（必要に応じ指定）：

namuni：地理情報の正式名称と単位をデータと共に取り出すときに **namuni=True**（または **1**）として指定する。この指定により戻り値の数は 5 つ（地理情報値，緯度，経度，正式名称，単位）になる。省略した場合は，戻り値の数は 3 つ（地理情報値，緯度，経度）となり名称等は返されない。

area：データを読み出すエリア（Area1 ～ Area6）を明示的に指定するときに **area='AreaX'** と指定する。省略した場合は自動的に選ばれる。

url：地理情報の取得場所を明示的に指定するときに **url=** ＜ファイルへのパス＞として指定する。省略した場合はデータ配信サーバーが指定される。ローカルにあるファイルを指定するときは，**AreaX**（X=1 ～ 6）の直上（通常は '・・・/AMD'）を指定する。

戻り値：

第 1 戻り値：指定した地理情報の並び（浮動小数の二次元（緯度×経度）配列）。

第 2 戻り値：切り出したメッシュの中心緯度の並び（浮動小数の一次元配列）。

第 3 戻り値：切り出したメッシュの中心経度の並び（浮動小数の一次元配列）。

第 4 戻り値（**namuni=True** のときのみ）：地理情報の正式名称（文字列）。

第 5 戻り値（**namuni=True** のときのみ）：地理情報の単位（文字列）。

使用例：北緯 35 ～ 36 度，東経 135 ～ 136 度の範囲にある各メッシュの水田面積比率の分布を取得する場合。

```
import AMD_Tools3 as AMD
lalodomain = [35.0, 36.0, 135.0, 136.0]
Ppad, lat, lon = AMD.GetGeoData ('landuse_H210100', lalodomain)
```

(3) GetCSV_Table 関数

GetCSV_Table 関数は，CSV ファイルのように，カンマ（,）等で区切られた表を内容とするテキストファイルを読み込み，それを文字列のリストとして返す関数です。数表先頭の 1 行をヘッダーとみなし，2 行目以降とは異なるリストとして返します。また，先頭行のフィールド数をこの数表全体のフィールド数とみなします。そして，2 行目以降にこれと異なるフィールド数（区切り文字の数）のレコード（行）があった場合には読み飛ばします。

なお，フィールドの区切として取り扱う文字は変更することができます。

書式：

```
GetCSV_Table (filename, delimiter=',')
```

引数（必須）：

filename：読み込むべき CSV ファイルの名前（文字列）。プログラムファイルとは異なるフォルダにある場合はそのフォルダ名も含める。

引数（必要に応じ指定）：

delimiter：区切りとして取り扱う文字を設定するときに **delimiter='\t'** のように指定する。省略した場合はカンマが区切り文字に設定され，CSV ファイルを読むことができる。タブ区切りの場合はタブ文字（\t）を使用する。

戻り値：

第 1 戻り値：見出しの並び（文字列のリスト）。

第 2 戻り値：表の内容の並び（フィールドを要素に持つリストのリスト）。

使用例：表がカンマ区切りのテキスト形式で保存されているファイル TxtTable.txt の内容を取り込む。

```
import AMD_Tools3 as AMD
header, body = AMD.GetCSV_Table ('TxtTable.txt',delimiter=',')
print ('header')
print (header)
```

```
print (  )
print ('body 全体')
print (body)
print (  )
print ('body の第 5 レコード')
print (body[4])
print (  )
print ('body の第 5 レコードの第 3 番フィールドの値')
print (body[4][2])
```

(4) GetCSV_Map 関数

GetCSV_Map 関数は，CSV 形式のテキストファイルを読み込み，それを浮動小数の Numpy Array Object として返す関数です。配列の列数は先頭行から判断し，行数は EOF（ファイルの終端記号）までの行数から判断します。数値として理解できないデータには，numpy.nan を与えます。

書式：

GetCSV_Map (filename, skiprow=0, upsidedown=True)

引数（必須）：

filename：読み込むべき CSV ファイルの名前（文字列）。プログラムファイルとは異なるフォルダにある場合はそのフォルダ名も含める。

引数（必要に応じ指定）：

skiprow：データの上部に余白や見出しがあり読み飛ばす必要があるときに skiprow=X（X は読み飛ばす行数）と指定する。先頭行から読み取る。

upsidedown：読み込んだ配列に行方向の転置を施さない場合に upsidedown=False を指定する。省略すると配列は行方向に反転される。この機能は，メッシュ農業気象データシステムがデータを南から北の順に配列に格納して取り扱う一方で，北が上になっている地理的データを読み込むと北から南の順にデータが配列に格納される不整合を修正するために用意されている。

戻り値：

numpy の浮動小数点配列。数値として理解できなかった要素には，numpy.nan を与えられる。

(5) PutCSV_TS 関数

PutCSV_TS 関数は，時系列のデータを CSV 形式のファイルで出力する関数です。

書式：

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

引数（必須）：

Var：時系列の一次元配列データ。ただし，Var=np.array（[V1,V2,..,Vn]）とすれば，n 個の一次元配列 V1,V2,…,Vn を一度に出力することができる。

tim：日付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：出力する CSV ファイルに見出しを付加するときに，header='見出し 1, 見出し2' のようにカンマで区切った文字列を与える。省略すると何も付加されない。

filename：出力ファイルの名称を指定するときに **filename='ファイル名.csv'** のように文字列を指定する。省略するとファイル名は **result.csv** となる。

戻り値：なし。

(6) PutCSV_Map 関数

PutCSV_Map 関数は，二次元の浮動小数点配列を CSV 形式のファイルで出力する関数です。データの前に，各列の経度値が記された行が挿入されます。また，データの左端に，各行の緯度値が記された列が挿入されます。

書式：

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

引数（必須）：

Var：出力すべきデータ（浮動小数の二次元（緯度×経度）配列）。

lat：メッシュの中心緯度の並び（浮動小数の一次元配列）。

lon：メッシュの中心経度の並び（浮動小数の一次元配列）。

引数（必要に応じ指定）：

filename：出力ファイルの名称を指定するときに **filename='ファイル名.csv'** のように文字列を指定する。省略するとファイル名は **result.csv** となる。

戻り値：なし。

(7) PutCSV_MT 関数

PutCSV_MT 関数は，三次元（任意×緯度×経度）の配列を，3 次メッシュコードをキーとするテーブルとして CSV ファイルに出力する関数です。3 次メッシュは，緯線・経線に沿って整然と並ぶ二次元の並びですが，各メッシュには 8 桁のメッシュコードが付されているので，メッシュコードとメッシュ値の対を作ることで，二次元の分布を 1 列の並びで表現することができます。このことを利用して，メッシュコードを縦にならべて見出しとし，各メッシュにおけるデータの並びを横に並べれば日別気象データのような三次元のデータを表の形式で書き出すことができます。ほとんどの GIS はこのような形式の CSV ファイルを読み込むことができるので，プログラムの実行結果を GIS で表示したり，GIS でさらなる処理を施したりできます。

書式：

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

引数（必須）：

Var：書き出すべきデータ（浮動小数の三次元（任意×緯度×経度）配列）。

lat：メッシュの中心緯度の並び（浮動小数の一次元配列）。**Var** の 2 番目（Python 的には 1 番目）の次元の要素数と一致しなければならない。

lon：メッシュの中心経度の並び（浮動小数の一次元配列）。**Var** の 3 番目（Python 的には 2 番目）の次元の要素数と一致しなければならない。

引数（必要に応じ指定）：

addlalo：出力される表の左端に置かれるメッシュコードとデータとの間に，メッシュの中心緯度と経度を挿入したいときに **addlalo=True** として指定する。省略すると挿入されない。

header：一行目に見出を出力したいときに **header='見出し1, 見出し2,・・・'** のように指定する。省略すると何も出力されない。

filename：出力ファイルの名称を指定するときに **filename='ファイル名.csv'** のように文字列を指定する。省略するとファイル名は **result.csv** となる。

removenan：海上や湖沼上など，**Var** の値が無効値（**numpy.nan**）であるようなメッシュについてもレコードとして出力する場合に **removenan=False** として指定する。省略すると，無効値しか格納されていないメッシュについては行が作られない。

delimiter：フィールドの区切り文字をカンマ（,）以外に設定するときには **delimiter='\t'** のようにその文字を指定する。省略するとカンマとなり CSV ファイルとなる。

戻り値：なし。

(8) PutNC_Map 関数

PutNC_Map 関数は，二次元（緯度×経度）のデータを NetCDF 形式のファイルで出力する関数です。GMT など，この形式を要求するソフトウエアに処理結果を読み込ませるときに使用します。

書式：

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

引数（必須）：

Var：出力するデータ（浮動小数の二次元（緯度×経度）配列）。

lat：メッシュの中心緯度の並び（浮動小数の一次元配列）。**Var** の 2 番目（Python 的には 1 番目）の次元の要素数と一致しなければならない。

lon：メッシュの中心経度の並び（浮動小数の一次元配列）。**Var** の 3 番目（Python 的には 2 番目）の次元の要素数と一致しなければならない。

引数（必要に応じ指定）：

description：データの正式名称などデータを説明する文。

symbol：データに対して用いる記号。

unit：データの数値が従う単位。

fill：無効値として取り扱う数値を設定するときに **fill=** 数値として設定する。省略した場合は 9.96921e+36 が無効値として取り扱われる。

filename：出力ファイルの名称を指定するときに **filename='ファイル名.csv'** のように文字列を指定する。省略するとファイル名は **result.csv** となる。

戻り値：なし。

(9) PutNC_3D 関数

PutNC_3D 関数は，三次元（日付×緯度×経度）のデータを NetCDF 形式のファイルで出力する関数です。GMT など，この形式を要求するソフトウエアに処理結果を読み込ませるときに使用します。

書式：

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

引数（必須）：

Var：出力するデータ（浮動小数の三次元（日付×緯度×経度）配列）。

tim：**Var** がカバーする期間の日付の並び（日時オブジェクトの一次元配列）。

lat：メッシュの中心緯度の並び（浮動小数の一次元配列）。**Var** の 2 番目（Python 的には 1 番目）の次元の要素数と一致しなければならない。

lon：メッシュの中心経度の並び（浮動小数の一次元配列）。**Var** の 3 番目（Python 的には 2 番目）の次元の要素数と一致しなければならない。

引数（必要に応じ指定）：

description：データの正式名称などデータを説明する文。

symbol：データに対して用いる記号。

unit：データの数値が従う単位。

fill：無効値として取り扱う数値を設定するときに **fill=** 数値として設定する。省略した場合は 9.96921e+36 が無効値として取り扱われる。

filename：出力ファイルの名称を指定するときに **filename='ファイル名.csv'** のように文字列を指定する。省略するとファイル名は **result.csv** となる。

戻り値：なし。

(10) **PutKMZ_Map** 関数

PutKMZ_Map 関数は，二次元（空間分布）の配列を KMZ ファイルで出力する関数です。作成されたファイルをダブルクリックすると，分布図を Google Earth 上に表示することができます。

書式：

PutKMZ_Map (data, lat, lon, label=None, cmapstr=None, minmax=None, filename='result')

引数（必須）：

data：表示させるデータ（浮動小数または numpy.datetime64 の二次元配列）。

lat：メッシュ中心点の緯度値の並び（浮動小数の一次元配列）。

lon：メッシュ中心点の経度値の並び（浮動小数の一次元配列）。

引数（必要に応じ指定）：

label：図の凡例にタイトルを付加するときに **label='文字列'** で指定する。省略すると何も表示されない。

cmapstr：カラーバーの配色を指定するときに **cmapstr='カラーマップの名称'** で指定する。省略すると **'RdYlGn_r'**（緑～黄～赤）が指定される。（後述を参照）

minmax：表示の最大値と最小値を指定するときに **minmax=[-10.0,20.0]** のように 2 要素のリストで指定する。省略すると表示データの最大値と最小値が設定される。

filename：出力ファイルの名称を指定するときに **filename='ファイル名'** のように文字列を指定する。この際，拡張子「.kmz」はこの関数が自動的に付加するので含めない。省略するとファイル名は **result.kmz** となる。

戻り値：なし。

備考

カラーマップの名称に「_r」を付加すると色の並びが反転する。カラーマップにつけられた名称は下記 URL を参照のこと。

http://matplotlib.org/examples/color/colormaps_reference.html

使用例：
北緯 36.0 度～ 38.5 度，東経 137.5 度～ 141.5 度の範囲における 2016 年 1 月 1 日の日平均気温分布図の KMZ ファイルを作成する。

```
import AMD_Tools3 as AMD
element = 'TMP_mea'
timedomain = [ '2016-01-01', '2016-01-01' ]
lalodomain = [ 36.0, 38.5, 137.5, 141.5]
Msh,tim,lat,lon,nam,uni = AMD.GetMetData (element, timedomain, lalodomain,
    namuni=True)
dat = Msh[0,:,:]
AMD.PutKMZ_Map (dat,lat,lon,label=nam+' ['+uni+']', cmapstr='rainbow', minmax=None,
    filename=element)
```

(11) lalo2mesh 関数

lalo2mesh 関数は，緯度と経度から，その地点が属する 3 次メッシュのコードを求める関数です。

書式：

```
lalo2mesh (lat, lon)
```

引数（必須）：

lat：十進小数表記の緯度（浮動小数）。

lon：十進小数表記の経度（浮動小数）。

戻り値：3 次メッシュコード（文字列）。

(12) mesh2lalo 関数

mesh2lalo 関数は，3 次メッシュコードから，そのメッシュの中心点の緯度と経度を求める関数です。

書式：

```
mesh2lalo (code)
```

引数（必須）：

code：3 次メッシュコード（文字列）。

戻り値：

第 1 戻り値：十進小数表記の緯度（浮動小数）。

第 2 戻り値：十進小数表記の経度（浮動小数）。

(13) timrange 関数

timrange 関数は，引数に与えた二つの日付を始日と末日とする連続した日付オブジェクトを作成する関数です。

書式：

```
timrange (str1, str2)
```

引数（必須）：

str1：yyyy-mm-dd 形式で表現した期間の最初の日（文字列）。

str2：yyyy-mm-dd 形式で表現した期間の最後の日（文字列）。

戻り値：

最初の日から始まり最後の日で終わる連続する日付の並び（**datetime** オブジェクトの一次元配列）。

（14）**latrange** 関数

latrange 関数は，引数に与えた二つの緯度範囲を包含する 3 次メッシュ範囲を構成する各メッシュの中心緯度の列を作成する関数です。

書式：

latrange (lat1, lat2)

引数（必須）：

lat1：十進小数表記の緯度範囲の南端（浮動小数）。

lat2：十進小数表記の緯度範囲の北端（浮動小数）。

戻り値：

範囲の南端が含まれるメッシュから北端が含まれるメッシュまでの各メッシュの中心緯度の並び（浮動小数の一次元配列）。

（15）**lonrange** 関数

lonrange 関数は，引数に与えた二つの経度範囲を包含する 3 次メッシュ範囲を構成する各メッシュの中心経度の列を作成する関数です。

書式：

lonrange (lon1, lon2)

引数（必須）：

lon1：十進小数表記の経度範囲の西端（浮動小数）。

lon2：十進小数表記の経度範囲の東端（浮動小数）。

戻り値：

範囲の西端が含まれるメッシュから東端が含まれるメッシュまでの各メッシュの中心経度の並び（浮動小数の一次元配列）。

2　DayLength3 モジュール

（1）**daylength** 関数

daylength 関数は，配列変数 **tim**，**lat**，**lon** で指定される時空間範囲における日長 [時間] を計算し配列で返す関数です。長澤（1999）の式により太陽の位置が高い精度で計算され，その結果に基づいて日長が求められています。**daylength** 関数は，太陽の上端の出没を明暗の境とする通常の日長のほか，「夜明け」や「日暮れ」のように，太陽の高度角が特定の角度となる時刻に基づく日長を計算することもできます（コラム 11 参照）。

文献：　長澤　工（1999）日の出日の入りの計算．160p，地人書館（東京）．

書式：

daylength (tim, lat, lon, elevangle=np.nan)

引数（必須）：

tim：時間の格子点を定義する日時オブジェクトの配列。

lat：緯度の格子点を定義する十進小数表記（浮動小数）の配列。

lon：経度の格子点を定義する十進小数表記（浮動小数）の配列。

引数（必要に応じ）：

elevangle：太陽が特定の高度角となるときを昼夜の境界と考える日長を計算するときに elevangle=-6.0 のように十進角度で指定する。正の数値は仰角（日長が短い），負の数値は俯角（日長が長い）を示す。省略すると，太陽上端の出没を昼夜の境界と考える日長（通常の日長）を計算する。

戻り値：

引数で指定した時空間範囲における日長［時間］の三次元（日付×緯度×経度）配列。

使用例 1：

北緯 36.0566 度，東経 140.125 度の地点における 2017 年 7 月 7 日の日長［時間］を計算して表示する場合。

```
from datetime import datetime
from DayLength3 import daylength
tim = [datetime (2017,7,7)]
lat = [36.0566]
lon = [140.125]
ld = daylength (tim,lat,lon)
print (ld[0,0,0])
```

使用例 2：

太陽の高度角が -6.0 度より高い時間帯を昼間とみなし，この長さを北緯 36.0 ～ 36.5 度，東経 140.125 ～ 140.725 度の範囲，2017 年 5 月 1 日～ 10 月 31 日の期間について求めるプログラムは次の通り。

```
import AMD_Tools3 as AMD
from DayLength3 import daylength
tim = AMD.timrange ('2017-05-01', '2017-10-31')
lat = AMD.latrange (36.0, 36.5)
lon = AMD.lonrange (140.125, 140.725)
# GetMetData 関数により別途気象データを取得している場合は，tim，lat，lon を生成する
    必要はない。
ld = daylength (tim, lat, lon, elevangle=-6.0)
```

付録：Python 利用環境の構築

パソコンで Python を利用する方法は多数ありますが，本マニュアルでは，Anaconda（アナコンダ）を利用して Python の利用環境を構築する方法を説明します。Anaconda は，Continuum Analytics 社がフリーミアム（無料で提供し，特殊な機能等を追加する際に課金する商品・サービス）で提供するデータ解析のためのソフトウエアパッケージで，データ解析をする部分に Python が採用されているので Anaconda をインストールすると Python が利用できるようになります。インストーラが親切に作られているほか，Windows 版と Mac 版の両方が用意されているので，初心者が無料で Python の利用環境を構築するよい選択肢です。

1　Anaconda のインストール

まず，Anaconda のダウンロードサイト（**https://repo.continuum.io**）にアクセスします（図 A1）。ページには，クリスマスリースのようなイラストが示されています。このページの下方に「Anaconda installers」と書かれているリンクがあるので，クリックします。すると，様々なバージョンのインストーラがリストされているページに移ります。ファイル名がたくさん並んでいる中から，Anaconda3-4.2.0 というバージョンを選びます。このバージョンは，Python バージョン 3.5 がインストールされる最後の版です。これ以降の版では，Python バージョン 3.6 がインストールされます。最新版をインストールしないのは，メッシュ農業気象データの処理結果を Google Earth 上に表示させるファイルを作成するのに使用する外部モジュールが，Python3.6 にまだ対応していないためです。

Anaconda3-4.2.0 のインストーラもたくさんの種類があるので，お使いの PC の種類（Windows か Mac か，64-bit か 32-bit か）に適合したものを注意して正しく選んでダウンロードしてください。「x86」は 32-bit 版，「x86_64」は 64-bit 版を示しています。ダウンロード先はどこでも構いません。

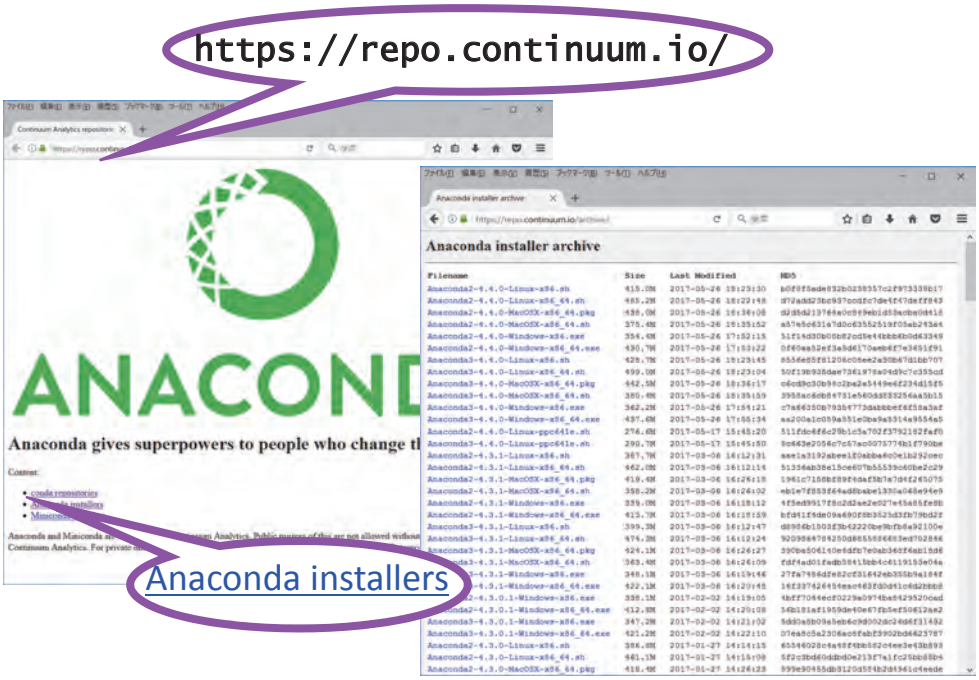


図 A1．Anaconda インストーラの保存サイトのトップ画面（左）と，ダウンロードサイトの画面（https://repo.continuum.io/archive/）（右）

ダウンロードの際，連絡先メールアドレスの入力を促されますが，入力する必要はありません（図 A2）。

図 A2.　連絡先メールアドレスの入力を促すウインドウ
入力は必ずしも必要ではない。

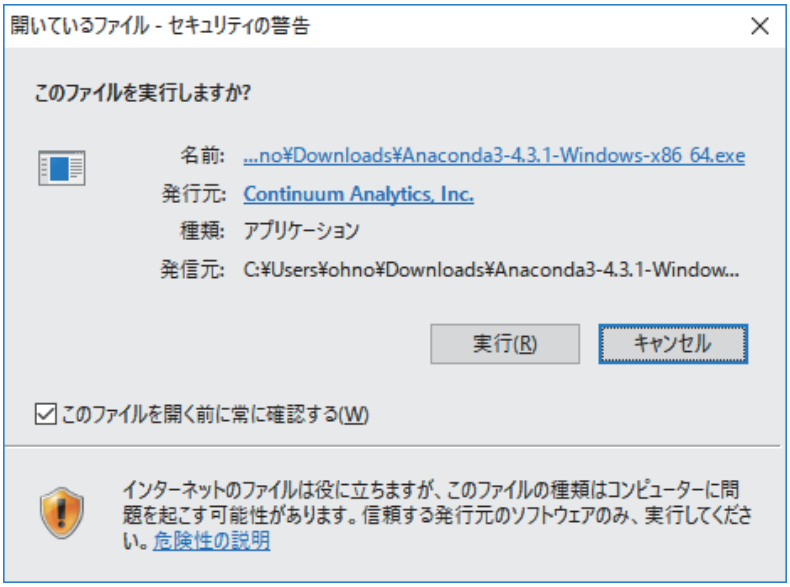


図 A3.　インストール開始時の確認画面
［実行］をクリックして進む。

Windows 版も Mac 版も入手したインストーラをダブルクリックして実行します。確認の画面が表示されるので，［実行］をクリックして進みます（図 A3）。これ以降は，インストーラの案内に従って進みます（図 A4，図 A5）。

Mac 版でも実行内容はほとんど同じです。入手した **Anaconda-3.x.x-MacOSX……dmg** インストーラをダブルクリックして実行します（図 A6）。

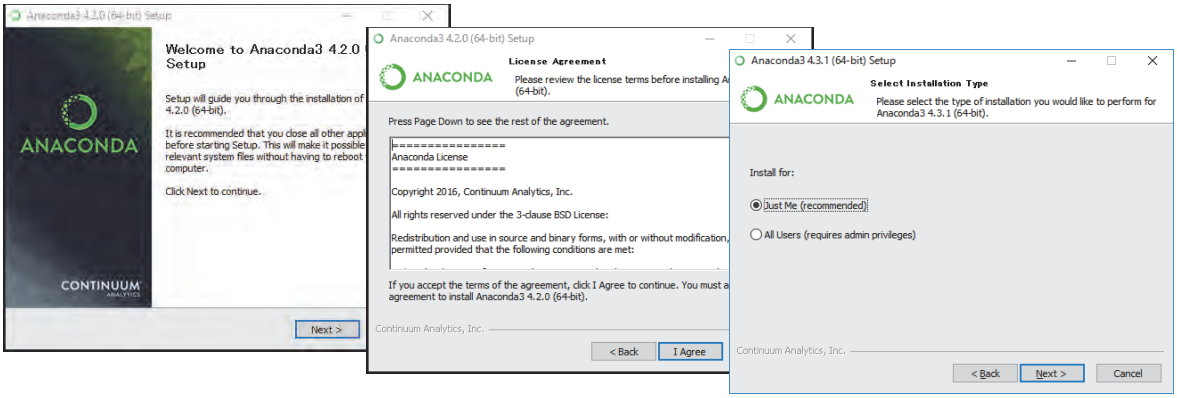


図 A4.　Windows 版 Anaconda インストーラの指示画面（1 ～ 3）

左：インストーラの最初の画面。［Next］をクリックして進む。中：ライセンスの確認。問題なければ［I Agree］をクリックして進む。右：PC の他のユーザーも使えるようにするかの選択。自分だけ（デフォルト）を選択して進む。

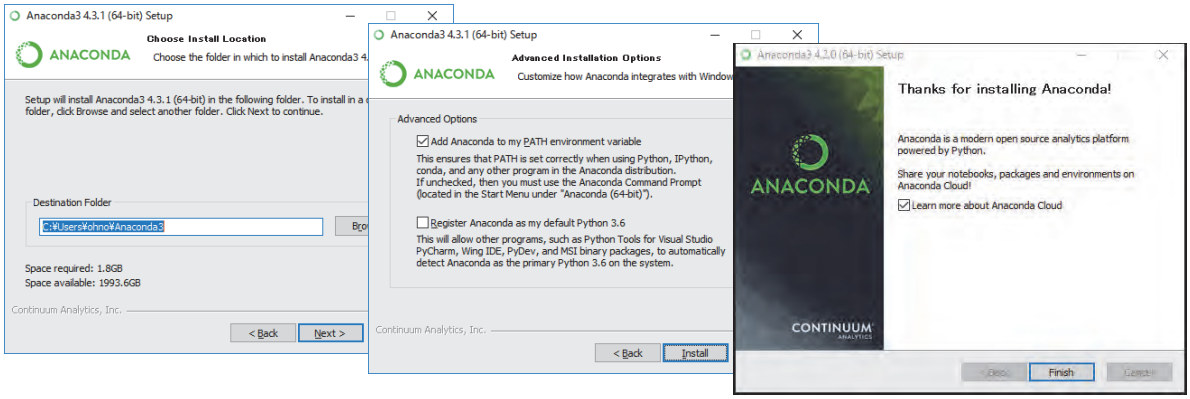


図 A5.　Windows 版 Anaconda インストーラの指示画面（4 ～ 6）

左：インストール場所の指定。ログインしているユーザーのホームディレクトリ（デフォルト）を選択して進む。中：この PC に他の方法で Python が別途インストールされていても Anaconda を優先するかどうかの設定。チェックボックスについてはデフォルトのままにする。[Install]をクリックすると，設定に従ってインストール作業が開始される。右：インストール終了のメッセージ。デフォルトで付いているチェックを外してから，［Finish］をクリックする。

図 A6.　Mac 版 Anaconda インストーラの指示画面

2　Anaconda のアップデート

Anaconda は，きわめて活発に開発が進められているオープンソースプログラムの集合体です。このため，最新のインストーラで新規に Anaconda をインストールしたとしても，それに同梱されているプログラムのどれかは，もはや最新ではないと考えなくてはなりません。このため，次の項で実施する Anaconda に同梱されていない Python モジュールの追加インストールに先立って，Anaconda に同梱されるプログラムを真に最新の状態にアップアップデートしておく必要があります。先ほど，敢えて最新でない Anaconda をインストールしていますが，それでもこの作業は必要です。

Anaconda のアップデートはアプリケーションではなく，コマンドプロンプト（Windows），または，ターミナル（Mac）と呼ばれるウインドウを開き，そこに命令文を打ち込んで実施します。Windows のコマンドプロンプトは，スタートメニューから **Anaconda3 ＞ Anaconda Prompt** と選択して開きます。Mac のターミナルは，アプリケーションから ＞ ユーティリティ ＞ ターミナル .app と選択して開きます。黒いウインドウが開いたら，キーボードからまず「**conda update conda**」と入力しエンターキーを押します。すると，アップデート可能なバージョンが表示され「Proceed？」と確認が求められるので「**y**」を入力してエンターキーを押します（図 A7）。いろいろな表示がされた最後に「［ **COMPLETE** ］」と表示されていることを確認してください。続いて Anaconda をアップデートします。今度は，キーボードから「**conda update anaconda**」と入力しエンターキーを押します。前回と同様，確認に回答してアップデートします。

conda や anaconda のアップデート，並びに，以下に示す Python モジュールの追加は，PC がインターネットに接続されている状態で実行してください。

この作業は，時々実行して Anaconda を新しい状態に保つようにしてください。

3　Python モジュールの追加

Anaconda にはデータを処理するための様々なモジュールが最初から組み込まれていますが，メッシュ農業気象データを効率的に処理するには，同梱されていない 3 つのモジュールを個別に追加しなければなりません。第一はメッシュ農業気象データの読み込みに必要な netcdf4 モジュールで，第二は，作成した図を Google Earth の地図に重ねて表示させるための simplekml モジュールで，第三は，プログラムで作成する分布図の配色のコレクション palettable モジュールです。追加の方法は，conda や anaconda のアップデートとほぼ同じで，PC がインターネット接続された状態で黒いウインドウにキーボードから命令を打ち込みます。それぞれの命令を下記に示します。いずれの場合も，命令を入力したらエンターキーを押しください。

netcdf4：「**conda install netcdf4**」

simplekml：「**conda install --channel https://conda.anaconda.org/conda-forge simplekml**」

palettable：「**conda install --channel https://conda.anaconda.org/conda-forge palettable**」

黒い窓での作業の場合，エラーメッセージは英語で地味に表示されますので，表示されている文字には注意してください。

この作業は，Anaconda のインストール時に一回だけ実施すれば大丈夫です。Anaconda は追加されたモジュールを記憶していて，Anaconda のアップデートの際にこれらも更新します。

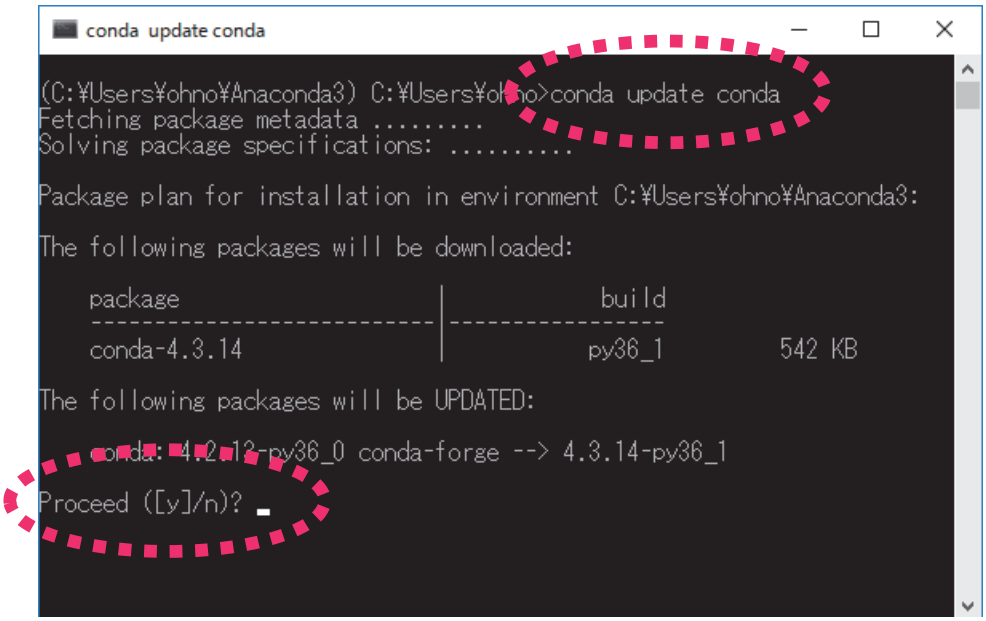


図 A7. conda のアップデートを命令した時の画面

4　作業フォルダの設定

Python のプログラムや，計算結果等を置く場所を設定します。どんな名前でもどこにおいてもよいのですが，簡単のため，ここでは，PythonWorks という名前のフォルダをデスクトップに作ってこれを作業フォルダとします。いくつかのサンプルプログラムを収めた作業フォルダを利用者 Wiki の「初めて利用される方へ」の項に用意しているので，これをダウンロードしてデスクトップに配置してください。

5　Spyder の起動

Spyder は，Python プログラムを作成したり実行したりすることができる統合的なアプリケーションで，Anaconda に同梱されています。Windows PC の場合は，スタートメニューに Anaconda3 という項目が追加されているので，それをクリックして展開し，「**Spyder**」と選択して起動します。Mac の場合は，アプリケーションに追加された **Anaconda-Navicator.app** をまず起動し，そこに表示される Spyder の項目の［Launch］をクリックします（この際，コミュニティへの勧誘画面が出ることがあります）。

Spyder が起動すると図 A8 のようなウインドウが表示されます。上部にメインメニューがありその下にツールバーアイコンが並んでいます。ウインドウの左半分の領域（ペイン）はプログラムエディタで，ここで Python のプログラムを書いたり修正したりします。ウインドウ右上部には，タブが付いていて，これを選択して表示される内容を切り替えることができます。「変数エクスプローラー」では，プログラムの変数にどのような数値が格納されているかを見ることができます。「ファイルエクスプローラー」では，指定したフォルダの中身を表示させたり，プログラムエディタに呼び出したりすることができます。右下のペインは，プログラムの実行結果を表示させる部分です。Anaconda のアップデートの際使用した，コマンドプロンプト（Windows）またはターミナル（Mac）の黒い窓と似た操作性を持ちます。このペインにもタブが付いていますが，通常は「IPython コンソール」を使用します。

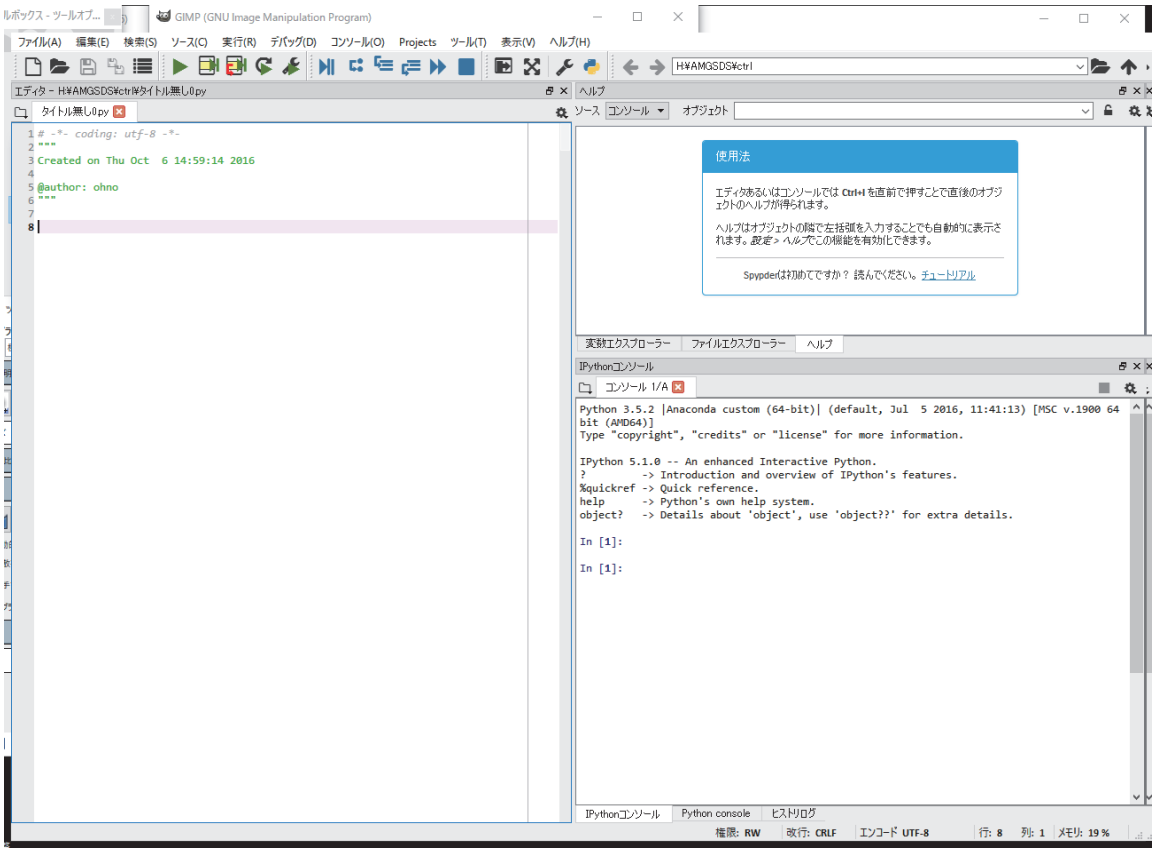


図 A8.　起動直後の Spyder の画面

6　Spyder へのサンプルプログラムの読み込み

サンプルプログラムを開いてみましょう。三分割されている Spyder 右上ペインの「ファイルエクスプローラー」をクリックします（図 A9）。続いて Spyder 右上端にある黒いフォルダの絵をクリックしてディレクトリの選択画面を表示させ，デスクトップ→ PythonWorks を選んで，

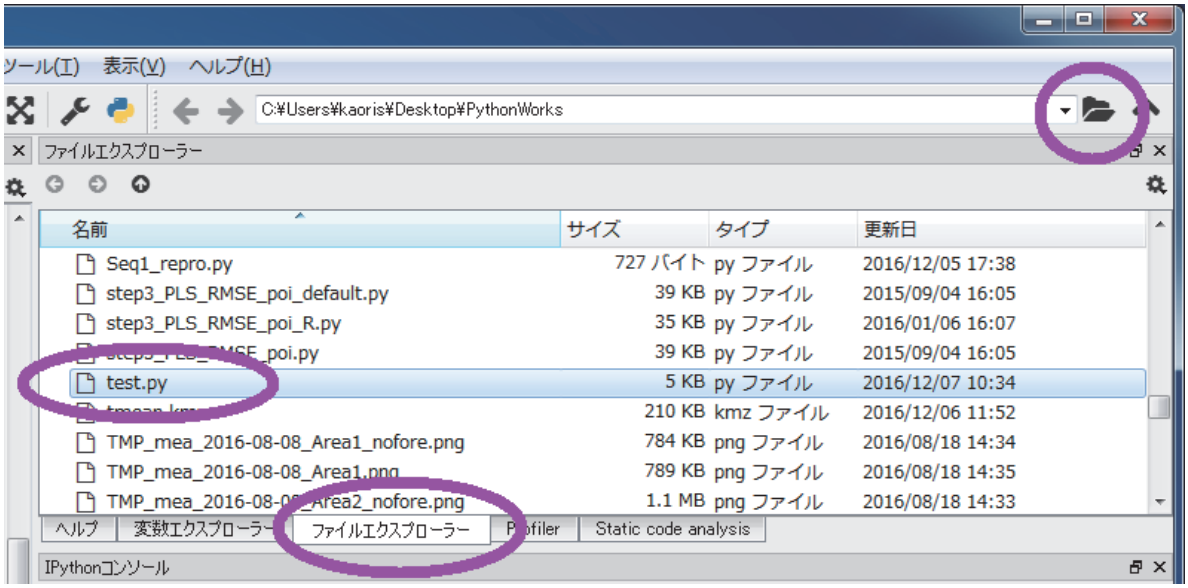


図 A9.　Spyder 右上ペインの拡大

「フォルダーの選択」ボタンを押します。画面右上ペインには作業フォルダの中身が一覧表示されるので，サンプルプログラム「**test.py**」をダブルクリックして開きます。Spyder メニューの中の黒いフォルダの絵をクリックしても，同様に開くことができます。これにより，プログラムエディタに内容が表示されます。

7　Spyder でのプログラムの編集

Spyder の左側のペインには，読み込まれたプログラムが表示されます（図 A10）。Spyder は読み込んだテキストを解析し，Python の文法に基づいて自動的に色分けして表示します。誤った文法を使用すると警告も表示されます。編集により内容が変更されると，シートのファイル名の横に * が付きます。編集は保存するまで有効にならないので，プログラムを修正したら，実行前に必ず保存して下さい。保存はメインメニューの「ファイル」から保存します。ツールバーのフロッピーディスクマークでも保存できます。

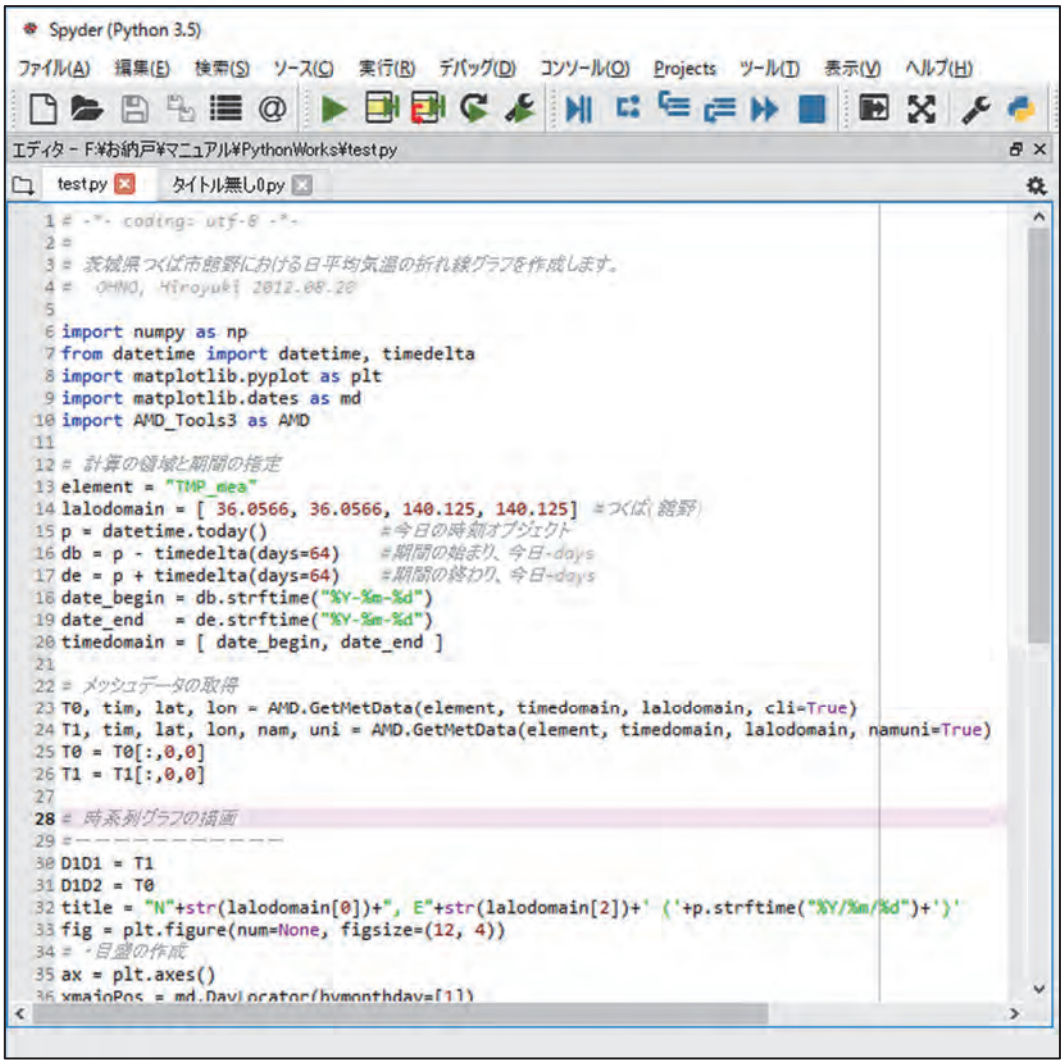


図 A10.　Spyder の左ペインの拡大

Spyder は，読み込んだプログラムを解釈し，文法的なエラーなどがあればそれを表示する。

8　Spyder でのプログラムの実行

Spyder でプログラムを実行するには，ツールバーの実行ボタン（▸）を押します。いくつかのプログラムを同時に開いて編集しているときは，今エディターで表示されているプログラムが実行されます。実行ボタンを押すと，Spyder の右下ペインにある「IPython コンソール」に「**runfile ('*Program_file_name*')**」と自動的に入力され，実行されます。実行結果もここに表示されます。IPython コンソールのプロンプト（「In [1]:」などと表示されている場所）に，キーボードから直接「**runfile ('*Program_file_name*')**」と入力しても実行されます。

プログラムに何らかの問題がありうまく実行できないと，Spyder はその内容を報告します。これを表示させるには，メインメニューから，「実行」→「Profile」を選択します。

PythonWorks.zip に梱包されているサンプルプログラム **test.py** は，メッシュ農業気象データ配信サーバーから，茨城県つくば市館野における最近の日平均気温を取得して，折れ線グラフで示すとともに，それを CSV ファイルで出力するものです。実行すると，Spyder の右下ペインにある「IPython コンソール」に図 A11 のようなグラフが表示されます。

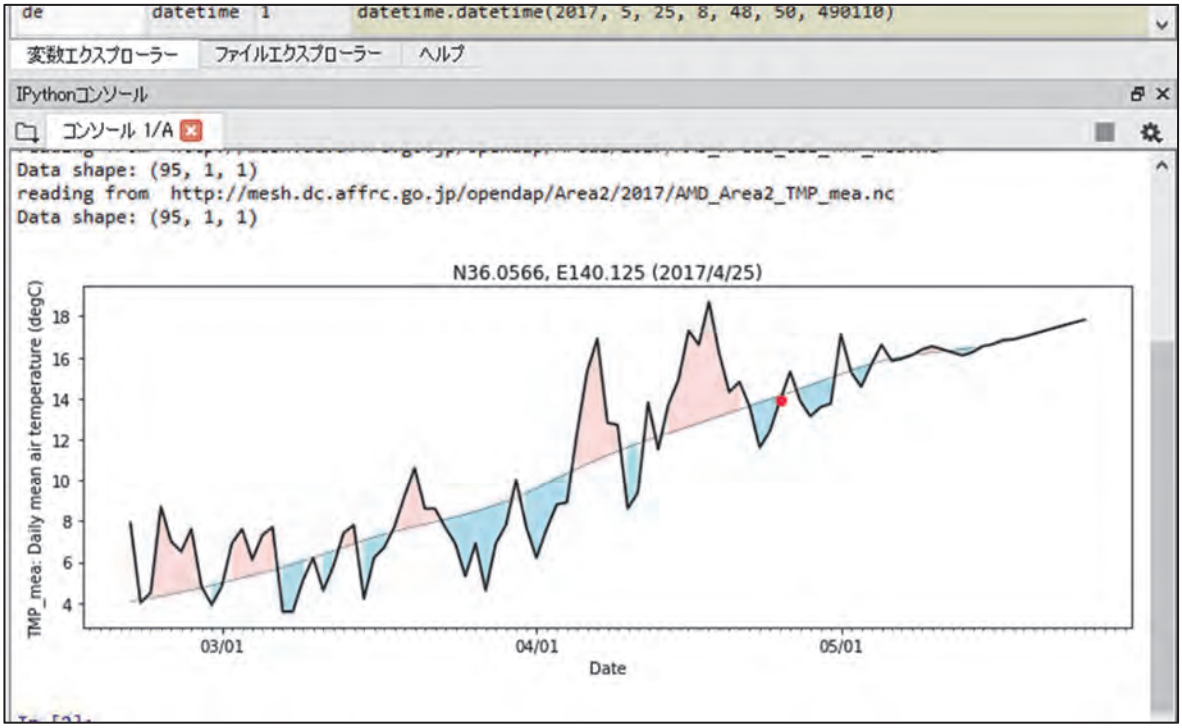


図 A11．プログラム test.py の実行結果

平成 29 年 7 月

技術マニュアル
「メッシュ農業気象データ利用マニュアル（2017年版）」

著　者　大野宏之　佐々木華織
発行者　（研）農業・食品産業技術総合研究機構　農業環境変動研究センター
〒 305-8604　茨城県つくば市観音台 3-1-3
問い合わせ先　農研機構農業環境変動研究センター　大野宏之
Tel：029-838-8191　E-mail：ohno(at)affrc.go.jp