Python によるひまわり８号
衛星データの可視化

JPGU2022
入門 JpGU スーパーレッスン

2022 年 5 月 29 日

V20220527v1.3

目次

はじめに

本書は JpGU 2022 入門スーパーレッスン「Python によるひまわり８号衛星データの可視化」にて実施されるプログラミング演習の教科書として作成されました．本演習では，千葉大学でアーカイブ・公開を行っているひまわり８号データを使って，取得から可視化までの手順を紹介する．ひまわり８号は従来のひまわり７号と比べて性能が向上し，より詳細に地球を観測できるようになり，気象現象のみならず“地球環境を観測する衛星”としても利用の幅が広がっている．本テキストは初めてひまわりデータを扱ってみたい方や Python の基本を学びたい初心者を対象としている．

演習には各自のペースで進められるように，オンデマンド形式の動画教材と web ブラウザを用いた Python 環境（Google Colab）を用意した．データファイルの構造や扱い方を理解して，参加者の必要に応じて自身で取得・処理できるようにすることが目的である．

本演習が皆様の研究の一助となれば幸いである．

東京大学生産技術研究所
千葉大学環境リモートセンシング研究センター
担当：豊嶋紘一

1. 演習環境の準備

1-1. Google Colaboratory へのアクセス

演習を行うにあたり，Python プログラムを実行する環境を用意する．Python を最初に使う際には各自の PC 環境へのインストール作業が必要となるが，今回はクラウド上で Python 環境が提供されている **Google Colaboratory**（以下 Colab）を利用する．Colab に関する情報は Google の公式[1]ページを参照のこと．Colab は，Google が機械学習の教育研究を目的として開発したツールで，**Jupyter Notebook** といわれるノートブック形式で作成したプログラムを実行し，その結果を確認しながらデータ解析を行う機能をベースに作られている．両者はショートカットキーが若干異なるものの操作性は似通っている（キー操作の詳細は「ツール」タブ中の「キーボードショートカット…」から参照できる）．Google アカウントを取得していれば，素早く Colab 環境で Python をはじめることができるので，演習を始めるにあたって事前に Google アカウントを取得してください．Google にログインした状態で，Colab[1] にアクセスします．（「colab」と検索すると素早くアクセスできる．）

公式には推奨 web ブラウザに Google Chrome，Firefox，Safari が指定されており，これら 3 つの最新バージョンでは完全に動作することが検証済み[2] であるため，最新バージョンでの使用をお勧めする．

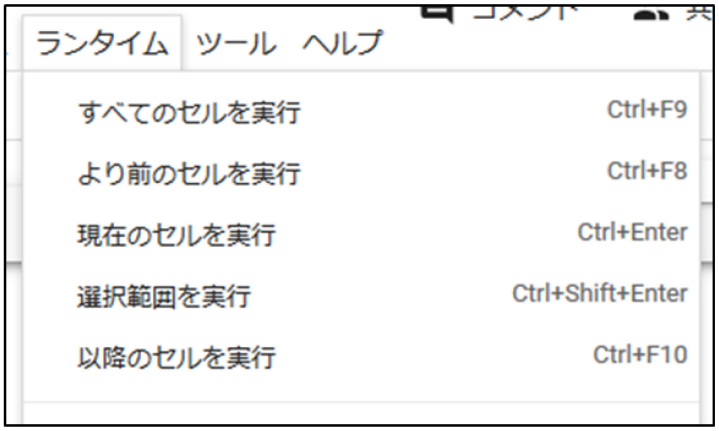
[1] https://colab.research.google.com/notebooks/intro.ipynb

[2] https://research.google.com/colaboratory/faq.html?hl=ja

1-2. Google Colaboratory の使い方

Colab のページが表示されると，画面上部にメニューバーを確認することができる．Python を実行するためノートブックを新規に開くには，メニュー「ファイル」から「ノートブックを新規作成」を選択する．すると新しいタブウインドウが開いてコード入力状態のセル（コードを入力する枠）が表示されていることがわかる．ここに Python コードを入力して，セル左側にある**実行ボタン**（三角マーク）を押すとその**コードを実行する**ことができる（作成されたコードはドライブ内に作られた「Colab Notebooks」内に保存される）．

同様のことはメニューの「ランタイム」を選択し「現在のセルを実行」を選択，または「Ctrl ボタンを押しながら Enter を押す」ことでも実行可能である．**新しくコードを入力**したい場合には「＋コード」を押すと，新しいセルが下に追加される．



例として「2×2」を実行してみる．セルに「2*2」と入力して実行ボタンを押すと次の行に「4」と表示される．

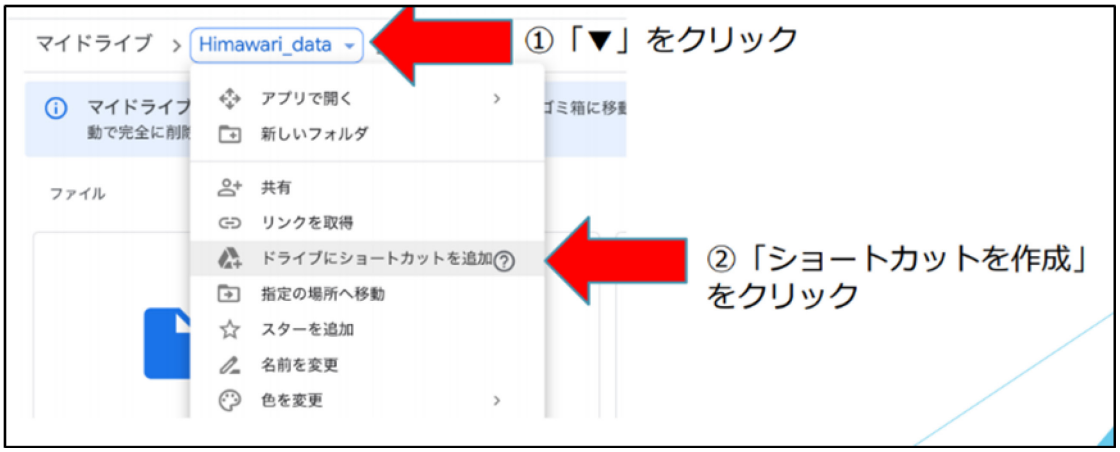
セル右上に表示されている↑・↓マークを押すとセルを一行前・後に移動させることができ，ごみ箱マークでセルを削除できる．

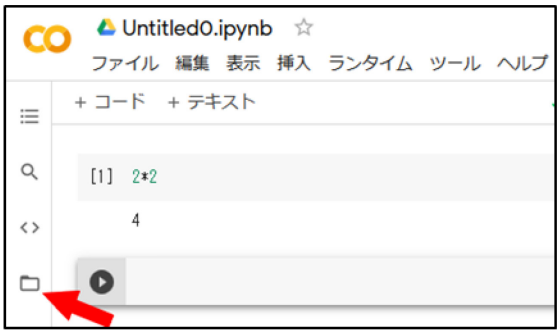
プログラムのファイル名は現在のプログラム名のところでクリックすることで直接変更できる．

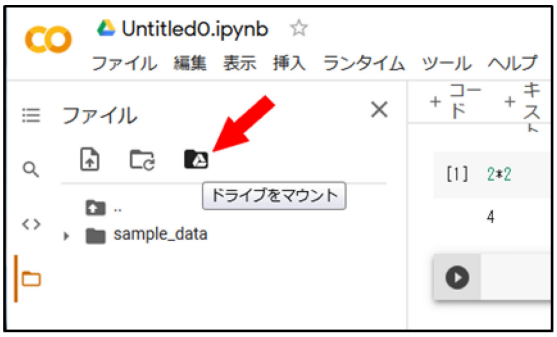
1-3. Google Drive のマウント

今回の演習で使用するデータを保存した共有設定済みの Google ドライブを各ユーザーの環境にマウント（リンクづけを）する方法を説明する．使用するひまわりデータはファイルサイズが大きく，個々にダウンロード・ファイルの解凍を行うと処理に時間がかかってしまうほか，アクセス集中によってデータベースに負荷がかかってしまうので，今回はあらかじめ演習で使用するデータを共有の Google ドライブに準備した．共有ドライブをマウントして自分の環境の一部として認識させることで円滑に演習を進めることができる．

では本演習で用いる共有の Google ドライブ [3]に移動する.
以下のような画面のように，「Himawari_Data」をクリックして表示されるメニューの「ドライブにショートカットを追加」を選択する.







メイン画面左側の「フォルダ」のマークを選択して，サイドバーを表示させる．上部の「ドライブをマウント」マークを選択する.

3 https://drive.google.com/drive/folders/1XvW_A4EemkFZ9p7R7HEVVio6gjnI2SNG?usp=sharing

ウインドウ左下に「Google ドライブをマウントしています…」の表示とともにノートブックに対する Google ドライブのファイルアクセス許可を求める確認画面が表示される．「GOOGLE ドライブに接続」を選択すると完了する．

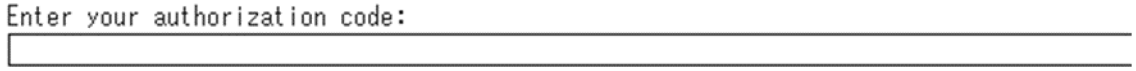
「drive」が表示され，マウントされていることを確認する．
「/content/drive/MyDrive/Himawari_Data」フォルダの中に今回演習で使用するデータが確認できる．

⚠ マウント作業はコマンドでも実行可能である．セルに次のコードを入れて実行する．

```
from google.colab import drive
drive.mount('/content/drive')
```

Go to this URL in a browser: https://accounts …と表示されるのでリンクをクリック，Colab にドライブのアクセスを許可し，認証コードをコピーし

て以下のセルに張り付け Enter を押すと”Mount at /content/drive”と表示され，マウントが完了する.



⚠ Colab ノートブックを 90 分以上操作しなかった場合，そのインスタンスは**自動的に終了**する．メモリ変数に入っていた結果は消えてしまうことに注意．その都度モジュールの import をし直す必要がある．**共有ドライブのマウントも解除**されてしまうので，上記マウント方法の説明に従って**再マウント**を行う必要がある．またインスタンスの最大は 12 時間とされており，モジュールを追加インストールした場合などは，**再び設定**する必要があることに注意．

2. Python を用いた計算とデータの入出力

ここでは変数を使った例を示す．変数 x に 5 を入れ 2 倍してみる．以下の2行を入力して実行する.
結果は以下のようになる.

```
[2]  X = 5
     X*2

     10
```

⚠ 本書に掲載のプログラムコードは Word にてフォーマットしている．PDF 形式へ変換の際に文字が変わっている場合がある．表記されている文字列と実際にコピーされる文字列が異なることがあるため PDF からプログラムコードをコピー＆ペーストする際にはその都度コードが正しいか確認が必要である．例えばインデントを用いている部分（Appendix-4）などには注意が必要である.

2-1. 科学計算モジュール(Numpy)を使う

☞ Python で数値計算を行うには，NumPy というモジュールを使うことが多い．モジュールとは機能単位に分けられた部品であり，通常はメインのプログラムから呼び出して使用する．Python では科学計算を主な用途とした NumPy など様々なモジュールがオープンソースで提供されている．NumPy を使用するた

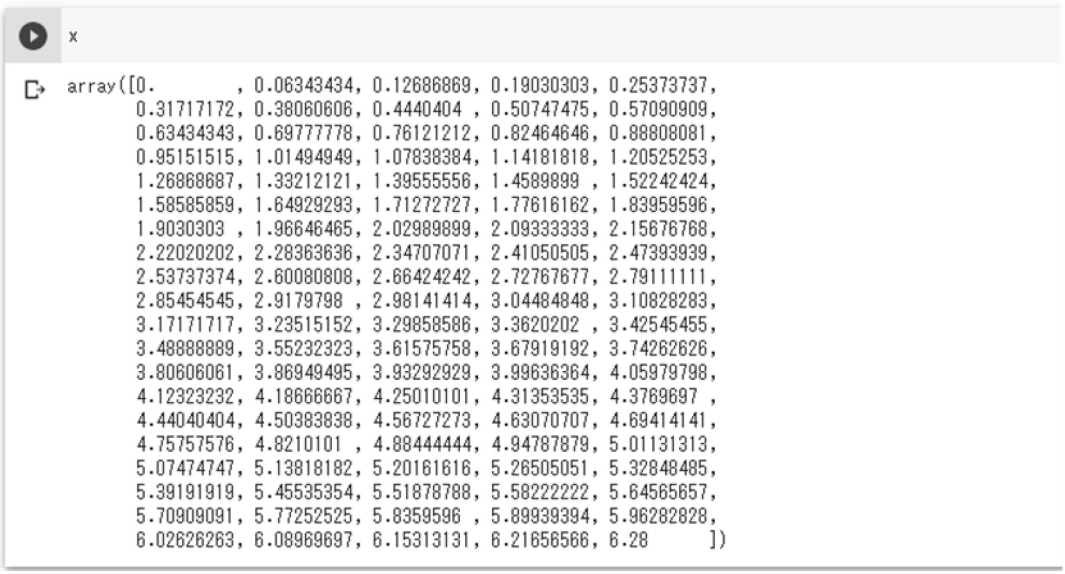
めには，以下のように import 文を使って事前に読み込むことが必要である．as np として NumPy を np と短縮して表している．

```
import numpy as np
```

☞ 例としてサンプルデータを作成する．
NumPy の関数の一つで，数列を作りたい場合に用いられる linspace() を利用する．
以下の例で x には，0.0 から 6.28 までの範囲を 100 分割した値，y には sin(x)の値が入る．

```
x = np.linspace(0.0,6.28,100)
y = np.sin(x)
```

☞ 変数の中身を確かめてみよう．x と入力して実行すると以下のように表示される．



```
x

array([0.        , 0.06343434, 0.12686869, 0.19030303, 0.25373737,
       0.31717172, 0.38060606, 0.4440404 , 0.50747475, 0.57090909,
       0.63434343, 0.69777778, 0.76121212, 0.82464646, 0.88808081,
       0.95151515, 1.01494949, 1.07838384, 1.14181818, 1.20525253,
       1.26868687, 1.33212121, 1.39555556, 1.4589899 , 1.52242424,
       1.58585859, 1.64929293, 1.71272727, 1.77616162, 1.83959596,
       1.9030303 , 1.96646465, 2.02989899, 2.09333333, 2.15676768,
       2.22020202, 2.28363636, 2.34707071, 2.41050505, 2.47393939,
       2.53737374, 2.60080808, 2.66424242, 2.72767677, 2.79111111,
       2.85454545, 2.9179798 , 2.98141414, 3.04484848, 3.10828283,
       3.17171717, 3.23515152, 3.29858586, 3.3620202 , 3.42545455,
       3.48888889, 3.55232323, 3.61575758, 3.67919192, 3.74262626,
       3.80606061, 3.86949495, 3.93292929, 3.99636364, 4.05979798,
       4.12323232, 4.18666667, 4.25010101, 4.31353535, 4.3769697 ,
       4.44040404, 4.50383838, 4.56727273, 4.63070707, 4.69414141,
       4.75757576, 4.8210101 , 4.88444444, 4.94787879, 5.01131313,
       5.07474747, 5.13818182, 5.20161616, 5.26505051, 5.32848485,
       5.39191919, 5.45535354, 5.51878788, 5.58222222, 5.64565657,
       5.70909091, 5.77252525, 5.8359596 , 5.89939394, 5.96282828,
       6.02626263, 6.08969697, 6.15313131, 6.21656566, 6.28      ])
```

☞ 主なモジュールを次に示す．

テキストで使用するモジュール一覧		
ライブラリ	**モジュール**	**主な使用用途**
標準	sys	システムパラメータに関する関数を含む
標準	bz2	bzip2ファイルなどの圧縮と展開
標準	os	ファイル操作に関する関数を含む
標準	tarfile	tarファイルの読み書き
標準	zipfile	ZIPファイルを圧縮の展開
numpy	numpy	数値計算，多次元配列を扱う
matplotlib	pyplot	グラフなどの描画
matplotlib	cm	色に関する描画
wget	wget	データのダウンロード
Pillow(PIL)	Image	画像処理
xarray	xarray	多次元データ解析
pandas	pandas	データ分析
geopandas	geopandas	地理情報を持つデータを扱うデータ分析

＊Python の標準ライブラリとは、Python にあらかじめ標準で用意されているライブラリのこと.

2-2. グラフを描く

☞ Colab で簡単なグラフを描いてみる．準備として描画の前に次の行を書いておく．この行はおまじない的なもので，これを書かないと図が Web ページ中に埋め込み表示されない.

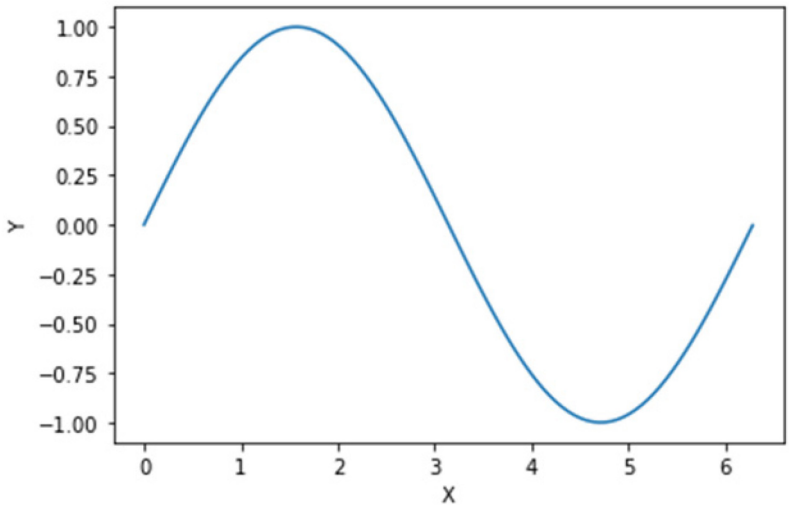
```
%matplotlib inline
```

☞ そのあと，Python の描画モジュール（Matplotlib）を使うために以下の記述をする．pyplot は Matplotlib パッケージ内のモジュールで，自動的に図形や軸を作るライブラリである.

```
import matplotlib.pyplot as plt
```

☞ 先ほど用意したデータを Matplotlib の plot メソッドでプロットしてみる．xlabel と ylabel はそれぞれ軸ラベルを書くメソッドである.

```
plt.plot(x,y)
plt.xlabel('X')
plt.ylabel('Y');
```



⚠ 描画コマンドの後ろにセミコロン（;）を付けると
[<matplotlib.lines.Line2D at 0x7fa83f4282b0>]
のような matplotlib のメッセージを非表示にできる.

2-3. ファイルの書き出し，読み込み

ここでは，基本的な Python による，テキストやバイナリファイルの入出力方法を確認する．まず NumPy をインポートする.

```
import numpy as np
```

はじめに NumPy 配列 a,b,c を生成し，a,b,c 3 列のカラムとなるように結合して 1 つの 2 次元配列 d を作成する．複数の配列を結合するための関数は様々あるが，ここでは np.stack を使用する．この関数を使うと，新たな軸に沿って配列を積み重ねる（stack する）ことができる．この例では結果の配列は元の配列より 1 次元多い 2 次元配列となる。第一引数には連結する配列のリスト，第二引数 axis には結合する軸（次元）を 0 始まりで指定する．第二引数 axis のデフォルトは 0 である．ただし，axis は結合された配列にとっての軸を指定する．今回の場合，a，b，c はそれぞれ 3 つの整数を持った 1 次元配列であるが，axis に 1 を指定し，横方向に結合(stack)することで，3 行 3 列の 2 次元配列を生成することができる.

（‘#’を付けると#以降の文字がコメントとして扱われる）

```
a = np.array([1,2,3])
b = np.array([4,5,6])
```

```
c = np.array([7,8,9])
d = np.stack([a,b,c],axis=1) # 横方向に結合する
```

この他，同様に配列を結合する関数の例として，numpy.vstack()（配列を縦（vertical）に結合する関数）や numpy.hstack()（配列を横（horizontal）に結合する関数）なども用意されている．ただし，これら 2 つの関数の第一引数は，リストではなくタプルで指定する必要があることに注意．

⚠ リストとタプル

上記の関数の第一引数として使用しているリストとタプルについて，簡単に紹介する．リストとは，[ ]の中でカンマによって区切られた要素で構成されている変数を指す（変数＝[要素 1，要素 2…]）．

一方，タプルとは，( )の中でカンマによって区切られた要素で構成される変数のことである（変数＝(要素 1，要素 2…)）．基本的にはリストと同じように扱うことができるが，要素の変更はできないことに注意．

テキスト出力．変数 d の 2 次元配列をテキスト出力する例．区切り文字（delimiter）をカンマに指定しているが，ここを変更することでスペース区切りにすることも可能である．次に［fmt］で，変数 d のデータ型を指定する．デフォルトは'%.18e'で，何も指定せずに書き込むと，小数点以下 18 桁の指数表記となる．指定可能な型として例を挙げると，10 進不浮動小数型（%f），符号あり 10 進整数型（%d），文字列型（%s）などがある．また，'%04d'のように桁のゼロ埋めも可能である．このような表記をした場合，04 は全体で 4 桁，残りを 0 で埋める，という意味である（例：‘0001’）．

```
# save as text
np.savetxt('text.csv', d, delimiter=',', fmt='%f')
```

テキスト入力．カンマで区切られた 3 列のテキストデータを x,y,z の変数に読み込む．区切り文字（delimiter）をカンマに指定している．

```
# open from text
x,y,z = np.loadtxt('text.csv', delimiter=',')
```

バイナリ出力．4 バイト実数型（float32）を指定してバイナリ出力する例．dtype オプションにはデータの型を指定する．ファイル全体を読み込む場合にはデータサイズを指定する必要はない．フォーマット文字の 'f' は浮動小数点

数を示し，後ろの４はバイト数を意味している． ’f’ 以外にも ’i’， ’u’ 等のフォーマット文字があり，それぞれ符号付き整数，符号なし整数を意味する．

```
# save as binary
f = np.array(d , dtype="f4") # 実数型
f.tofile("bin.dat")
```

その他のデータ型については巻末の Appendix-3(データ型の一覧)に付したので参照されたい．

バイナリ入力．3x3 の配列として４バイト実数型を指定して読み込みを行う．NumPy の np.fromfile()という関数を用いて’bin.dat’という名前のデータファイルを開いている．ここの dtype の指定には’<’が付記されている．これはバイトオーダー(データの記録純)の指定である．デフォルトのバイトオーダーは計算機に依存するが，フォーマット文字の前に ’<’， ’>’ を置くとそれぞれリトルエンディアン，ビッグエンディアンとして，入出力順を指定できる．ここではリトルエンディアンを指定する．

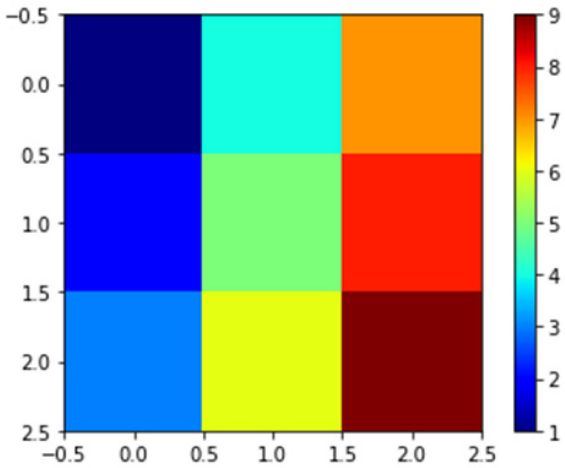
```
# open from binary
data = np.fromfile('bin.dat', dtype="<f4").reshape((3,3))
```

⚠ バイトオーダーとは・・・

２バイト整数，４バイト浮動小数点数といった 2byte 以上の変数をメモリに格納する際のデータの並べ方のことで，endian(エンディアン)とも呼ばれる．上位バイトから順に格納する方式を big endian，下位バイトから格納する方式を little endian で，Intel のプロセッサは little endian を採用している．たとえば，「ab cd ef gh」という 2byte データがある．２バイトデータとは，「ab」 や「cd」といった二つのバイトで一つの情報を表すデータ型のことを指す．big endian では下位から上位バイトに向かって読み込むので「ab cd ef gh」と読む．一方，「gh ef cd ab」と読むのが little endian である．もし2バイトという情報を与えないで１バイトとして読んでしまったら，「hgfedcba」と読んでしまうので，正しい情報が得られない．したがってデータの in，out の際にはデータ型とバイトオーダーを指定する必要がある．バイトオーダーは計算機(CPU のアーキテクチャー)依存であり，大型計算機で書き出されたデータは big endian，パソコンは little endian がデフォルトであることが多い．CEReS で公開のひまわりデータは big endian のバイトオーダーであるので，パソコンで読み込む際には，big endian のファイルであると指定する必要がある．

グラフの画像保存　読み込んだ変数（data）を図で確認してみる.

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
plt.imshow(data,cmap="jet")
plt.colorbar() #カラーバーの表示
plt.savefig("img.png", format="png",dpi=300)
#300dpi 保存
plt.show()
```



2-4. ユーザー定義関数

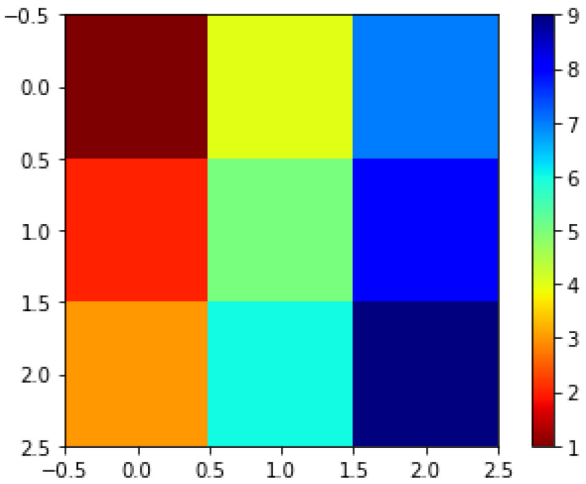
共通する部分のコードを簡略化する方法にユーザー定義関数があり，def 文を使って定義することでユーザーの目的に合わせた関数を作ることができる.

例えば，上記のような図を複数描画したい場合，同じコードを何度も書くのは煩雑である. そこで引数を与えるだけで上の図を作成する方法を紹介する. 引数とはコマンドにインプットする情報のことである. Python では関数を「def」文を使って定義する（ここでの関数名は mkfig とする）. mkfig 関数の引数には「データ名」，「カラーマップ名」を指定するものとする. なお，関数の範囲は コロン”:”から始まり，関数内のコードは左端に一定のスペースを空けて入力する(Colab の場合は”:”以下で改行すると自動的にスペースが作られる). 関数内でのスペースサイズに特段指定はないが，サイズは統一する必要がある.

```
def mkfig(data,color):
  plt.imshow(data,cmap=color)
  plt.colorbar()
  plt.show()
```

関数を使用する場合は mkfig（データ名，カラーマップ名）を指定する．「データ名」はユーザーが表示したいデータ変数を指定するが，「カラーマップ名」は mkfig 関数の中で使われている imshow 関数のパラメータ cmap の指定を意味することに注意．つまり，「カラーマップ名」には，matplotlib が指定する型のパラメータを指定する必要がある．このパラメータは自作することもできるが，matplotlib ライブラリの cmap(colormap)パラメータの中から指定するのが簡単である．ここでは“jet_r”という cmap パラメータを指定する．その他の cmap パラメータについて知りたい場合は matplotlib のサイト[4]を参照されたい．

```
mkfig(data,"jet_r")
#jet のカラーを反転させた jet_r を使用する
```



これをプログラムごとに毎回記述するのは大変なので，この関数を他のプログラムでも import して利用できるようにする．まず Google Colab にて「ファイル」→「ノートブックを新規作成」をクリックし，上記の関数のみのプログラムを新規作成する．Colab Notebooks ファイルがドライブ内に無い場合は自動的に作成される．プログラム名は deff.ipynb とし，作成後「ファイル」から「保存」をクリックすること．

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
#この関数を使用するためには2つのモジュールが必要
def mkfig(data,color):
  plt.imshow(data,cmap=color)
```

[4] https://matplotlib.org/3.5.0/tutorials/colors/colormaps.html

```
  plt.colorbar()
  plt.show()
```

「def mkfig」の次の行からインデントが挿入されていることに注意．詳しくは巻末の Appendix-4(Python におけるインデントの意味)を参照のこと．
Colab ではプログラムを作成すると.ipynb になってしまう．関数は.py であるので，プログラムの中で import するためには.py 形式に変換する必要がある．まず共有の GoogleDrive をマウントする．続いて deff.ipynb があるフォルダへ移動する．（以下を実行したあと，!ls で deff.ipynb があるかどうかを確認する）．

```
cd "/content/drive/MyDrive/Colab Notebooks"
```

次に以下の shell コマンドを実行する（deff.ipynb 以外のプログラムで実行する）．shell とは，Ubuntu をはじめとする Linux システムを動かすためのコマンドであり，Colab は ubuntu をベースとするシステムでできているので，python だけでなく，shell を用いて動作させることも可能である。colab 上 Shell コマンドを動かすためには，下記のようにコマンドの前に“！“を付記して，実行させることができる．同じセル内にシェルコマンドと Python コードを混在させるとうまく動かないので分けて実行する点に注意すること．

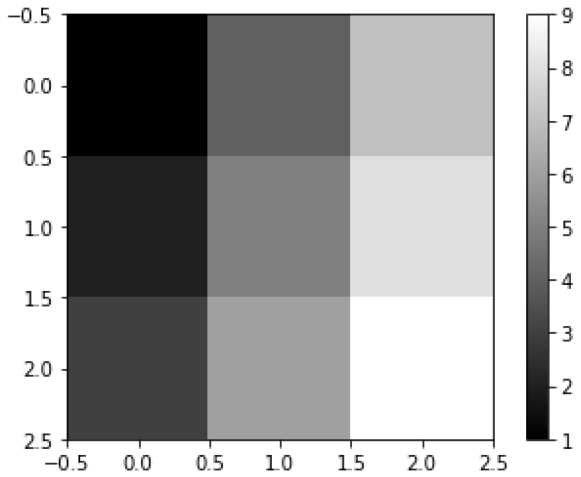
```
!jupyter nbconvert --to python deff.ipynb
```

同フォルダ内に deff.py が作成される．これを別プログラムで扱うには以下のコマンドを実行する．

```
#from google.colab import drive
#drive.mount('/content/drive')
#すでにマウント済みのユーザーはスキップ(コメントアウト)してよい
import sys
sys.path.append(¥
"/content/drive/MyDrive/Colab Notebooks")
import deff
```

sys モジュールのモジュール検索パスは標準ライブラリの sys モジュールの sys.path に格納されている．sys.path はディレクトリのパスを文字列として格納したリストオブジェクトとして返す．ここでは，sys.path で得たディレクトリのパスに append()メソッドを用いて新たなパス(″/content/drive/MyDrive/Colab Notebooks″)を追加している(sys.path.append)．これによって deff.py の関数のモジュールを import できるようになる．ユーザー作成の関数名は.py より前の部分の名称となる．ユーザー定義関数を使用する際には以下のように２次元配列である data 変数と，「カラーマップ名」に gray を指定する．

```
deff.mkfig(data,"gray")
```



☞本テキストでは表記の便宜上，1 行あたりのコマンドが長い場合には明示的に改行を行っている．Python では¥（またはバックスラッシュ）を付すことで改行していても，一連のコマンドと認識させることができる．

3. ひまわり 8 号データの読み方

3-1. 千葉大学で公開のデータセットについて

ひまわり 8 号データは気象庁によるデータ処理の後，研究用途として千葉大学，東京大学，JAXA，NICT の 4 機関にリアルタイムで提供されている．千葉大学環境リモートセンシング研究センター（CEReS）では，気象庁からリアルタイムに取得したデータに精密幾何補正処理を行い，緯度経度直交座標に変換したデータ（全球 Gridded データ）としてアーカイブしている．それらのデータは以下のリンクからダウンロードが可能である．しかしながら本演習ではダウンロードを行う時間を短縮するために予め（あらかじめ）サンプルデータを共有ドライブに準備しているので個々に取得する必要はない．基本的なデータの諸元は以下のサイトに記載しているので確認のこと．

千葉大学 Himawari-8/9 Full-disk gridded data 公開サイト [5]

千葉大 CEReS Gridded データについてデータの仕様は以下の通り．

東西方向	85° E to 205° E
南北方向	60° N to 60° S
データ型	2 バイト　符号なし整数型
バイトオーダー	Big endian data order
データ読込順	北から南方向

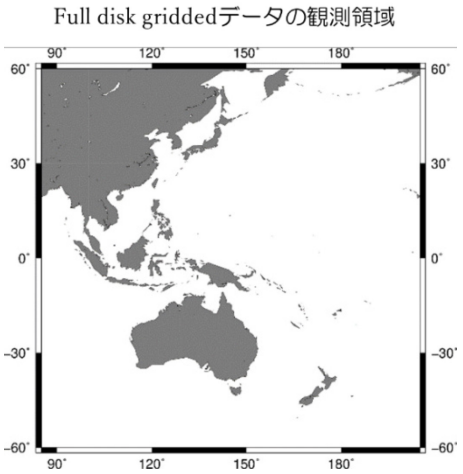
[5] http://www.cr.chiba-u.jp/databases/GEO/H8_9/FD/index_jp.html

CEReS 公開 Gridded データ

	CEReS Gridded data		ひまわり8,9号対応バンド	波長（μm）	pixel x line	空間解像度	階調	想定用途
可視	EXT	01	Band 03	0.64	24000 x 24000	0.005 degree (500 m 相当)	2048(11bit)	下層雲・霧
	VIS	01	Band 01	0.47	12000 x 12000	0.01 degree (1km 相当)		植生・エアロゾル
		02	Band 02	0.51				
近赤外		03	Band 04	0.86				
	SIR	01	Band 05	1.6	6000 x 6000	0.02 degree (2km 相当)		雲相判別
		02	Band 06	2.3				雲粒有効半径
赤外	TIR	01	Band 13	10.4			4096(12bit)	雲画像
		02	Band 14	11.2				雲画像・水温
		03	Band 15	12.4				
		04	Band 16	13.3			2048(11bit)	雲頂高度
		05	Band 07	3.9			16384(14bit)	下層雲・霧・自然火災
		06	Band 08	6.2			2048(11bit)	上・中層水蒸気
		07	Band 09	6.9				中層水蒸気量
		08	Band 10	7.3			4096(12bit)	
		09	Band 11	8.6				雲相判別・SO2
		10	Band 12	9.6				オゾン

また，ひまわり８号データの DN 値を物理量に変換するためにはルックアップテーブル（LUT）が必要である．DN 値とは Digital number のことで，データの大小を相対的に整数値で表している．LUT ファイルと，Gridded data の取得からデータの読み込み・物理量変換・画像化の一連の作業を自動で行うサンプルコード（Fortran，C 言語，Python）を含んだサンプルコード集[6]は千葉大学のデータ公開サイトにて提供を行っている．

ひまわり８号 Gridded データは，次のルールでバンドごとのファイル名が付けられている．例えば最も高精細な可視(赤色)バンドであるバンド 03 は Gridded data の ext01 に，熱赤外であるバンド 13 は Goridded data の tir01 に対応するので取得する際にはこのバンド種別を確認すること（詳細は表「CEReS 公開 Gridded データ」を参照のこと）．グリッドデータのファイル名は yyyymmddhhnn.XXX.ZZ.ｆｌｄ.geoss のようになっている．ここで，yyyy，mm，dd，hh，nn はそれぞれ年，月，日，時，分，XXX.ZZ はグリッドデータのバンド名，fld は観測領域名(ここでは fulldisk の意味)を示し，時刻は世界協定時刻（UTC）である．

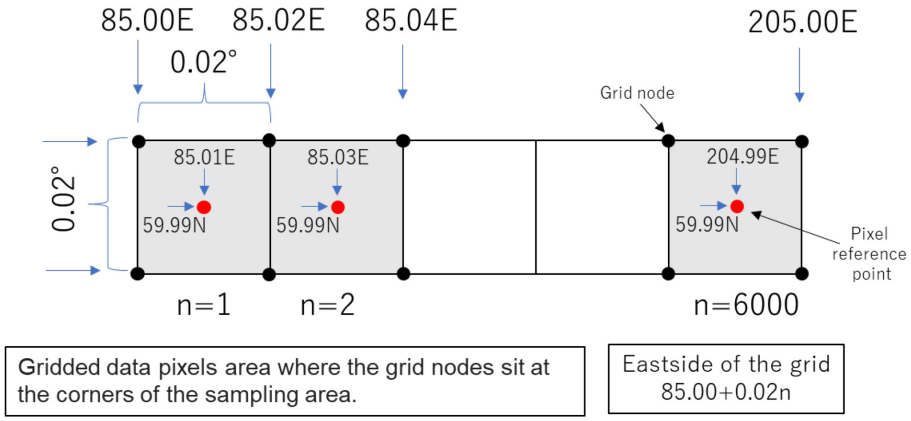


【例】202007040000.ext.01.fld.geoss

[6] http://www.cr.chiba-u.jp/databases/GEO/H8_9/FD/count2tbb_v102.tgz

全球スキャン（フルディスク:fld）Gridded data の観観測範囲は東経 85°～205°，北緯 60°～南緯 60°（120°×120°四方の領域）であり，各バンドの空間分解能に対応したグリッド（格子）に区切られている（図参照）．グリッドデータは西から東，北から南の順に記録されている．

例）解像度0.02°/grid　のデータの場合



⚠ デジタルナンバー（DN 値）とは・・・連続量（アナログ信号）をデジタル変換して得られる，0, 1, 2, 3, …のような整数値で，データの値の大小を相対的に整数値で表している．

3-2. ひまわりデータのダウンロード方法（参考）

本演習ではサンプルデータを用意しているが，演習の後に各自で任意のデータを利用する際のデータ取得方法を説明する．

⚠ 演習時は次の‘3-3. データの描画’へスキップしてください．

例．2020 年 7 月 7 日 12 時（UTC 時間では 3 時）の赤外バンド（Band13）を取得する
データは FTP[7] にて公開されており，年月（202007），バンド（TIR）のサブディレクトリに分類され，3-1 節のファイル名規則によってデータが保存され

[7] ftp://hmwr829gr.cr.chiba-u.ac.jp/gridded/FD/V20190123

ている（202007070300.tir.01.fld.geoss.bz2 ）.
例えば，ダウンロードするデータファイルの絶対パスは以下のとおり.

データをダウンロードするコマンドの一つに wget がある．wget は，一般的には Sell 版の wget が使用され，Colab 上でも下記のように実行することができる.

```
!wget
```

しかし，ここでは python 演習であるため，敢えて Python コードで Download する例を示す．まずは wget をインストールする.

```
!pip install wget
```

wget.download()として任意のファイルをダウンロードすることができる.

```
import wget
file = 'ftp://hmwr829gr.cr.chiba-u.ac.jp/gridded/FD/V20190123/202007/TIR/202007070300.tir.01.fld.geoss.bz2'
wget.download(file)
```

時系列に伴う解析を行うには，時間方向に連続したファイルをダウンロードする必要がある．以下に Band13（TIR01）の 2020 年 7 月 7 日 03 UTC から 06 UTC までの 10 分間隔のデータをダウンロードするサンプルを示す．このコードを実行すると，指定時間範囲のデータがダウンロードされ，解凍した後のデータのみ保存される．ひまわりデータは（bz2）形式で圧縮されているので「bz2」モジュールのコマンドで解凍する．解凍前の bz2 データは os モジュールの remove を用いて削除している.

```
import wget #wget モジュールの import
import bz2  #解凍用モジュール
import os
FTP="ftp://hmwr829gr.cr.chiba-u.ac.jp/gridded/FD/V20190123/"
B="TIR"
b="tir"
bno=1
for y in range(2020,2021) :  #Year
  for m in range(7,8) :  #Month
    for d in range(7,8) :  #Day
      for h in range(3,7) :  #Hour
        for mi in range(0,60,10):  #Minute
```

```
tt=FTP+"{year:04}{month:02}/{band}/".format(year=y,¥
  month=m,band=B)
          # 書式指定文字列は{}内に:に続けて書く． month:02 は一桁の
場合，十の位に 0 を埋める

DATE="{year:04}{month:02}{day:02}{hour:02}{min:02}".¥
  format(year=y,month=m,day=d,hour=h,min=mi)
          fname=DATE+".{band}.{bandno:02}.fld.geoss".¥
  format(band=b,bandno=bno)
          fnamebz=fname+".bz2"
          ttt=tt+fnamebz
          print(ttt)
          wget.download(ttt)  #データダウンロード
          zipfile = bz2.BZ2File(fnamebz)    #圧縮データオープン
          data = zipfile.read()    #圧縮データ読み込み
          open(fname, "wb").write(data)    #解凍データ書き出し
          os.remove(fnamebz)    #圧縮データ削除
```

3-3. ひまわり 8 号 Gridded データの描画

それではデータの描画を行う．共有ドライブがマウントされていることを確認して，科学計算用モジュール（NumPy）と描画モジュール（Matplotlib）を使う準備をする．ここでは matplotlib のカラーマップを使うため，Matplotlib.cm も設定する．

```
# 図を表示するためのオプション
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

例として 2019 年 9 月 9 日 3 時 00 分(JST)における，10.4 μm 赤外バンド（tir.01）のデータを使う．これは 2019 年に千葉県に甚大な被害を引き起こした台風 15 号事例である． ひまわり 8 号データのファイル名（"201909081800.tir01.fld.geoss"）の日付は協定標準時（UTC）であることに注意されたい．日本標準時（JST）は UTC よりも 9 時間進んでいるので，取得の際には 9 月 8 日 18 時（UTC）とする．後の作業を簡単にするために，共有フォルダのパスや日時ファイル名，拡張子の変数を作る．

```
datapath = '/content/drive/MyDrive/Himawari_Data/'
YYYYMMDDHHmm = '201909081800'
ext = '.tir.01.fld.geoss'
file = datapath + YYYYMMDDHHmm + ext
```

変数「datapath」にはひまわりサンプルデータが保存されている共有フォルダのパスを指定している．「/content/drive/MyDrive」が自身のホームディ

レクトリで，そこに「Himawari_Data」という共有フォルダがマウントされた状態を意味する．「YYYYMMDDHHmm」はデータの年月日時刻（UTC）を指定，「ext」にはひまわりの観測バンドの種別を指定している．これらの文字列の変数を「＋」を用いて連結して「file」というフルパスを示す変数としている．以下のようにして確認してみよう！

```
file
```

```
'/content/drive/MyDrive/Himawari_Data/201909081800.tir.01.fld.geoss'
```

つまりこれが，ひまわりデータの置き場所である．

それではひまわりデータを np.fromfile()という関数を用いて変数「file」を読み込んでみる．ひまわりデータの形式は，2byte 符号なし整数型（Big endian）であるため，「dtype」には'>u2'を指定している．（Appendix-3(データ型の一覧)参照）

```
dataDN = np.fromfile(file,dtype='>u2')
```

適切に読み込まれたか確かめてみよう．

```
dataDN
```

```
array([2401, 2401, 2401, ..., 3031, 3031, 3031], dtype=uint16)
```

2 バイト符号なし整数型（unit16）でデータが読み込まれたことがわかる．
つぎに変数配列の形状を確認してみる．

```
dataDN.shape
```

```
(36000000,)
```

dataDN は要素数 36000000 の 1 次元配列（一連のデータの羅列）として読み込まれたことがわかる．
Gridded データの記録順は 85° E, 60° N から経度方向に 205° E, 60° S までである．データ範囲は緯度幅⊿120°，経度幅⊿120°である．熱赤外バンド（tir.01）のグリッド間隔は 0.02°，データの個数は 6000 × 6000 ＝ 36000000 となっている(3-1 節参照)．
図として可視化するために，読み込んだ 1 次元配列データに reshape()（全要素数を保ったまま配列の次元数を変更すること）を行って，2 次元配列に変換する．

```
dataDN = dataDN.reshape(6000,6000)
dataDN
```

```
array([[2401, 2401, 2401, ..., 3454, 3454, 3454],
       [2403, 2403, 2403, ..., 3432, 3432, 3457],
       [2395, 2401, 2401, ..., 3457, 3457, 3457],
       ...,
       [3190, 3171, 3171, ..., 3027, 3027, 3027],
       [3171, 3171, 3171, ..., 3056, 3056, 3056],
       [3165, 3165, 3165, ..., 3031, 3031, 3031]], dtype=uint16)
```

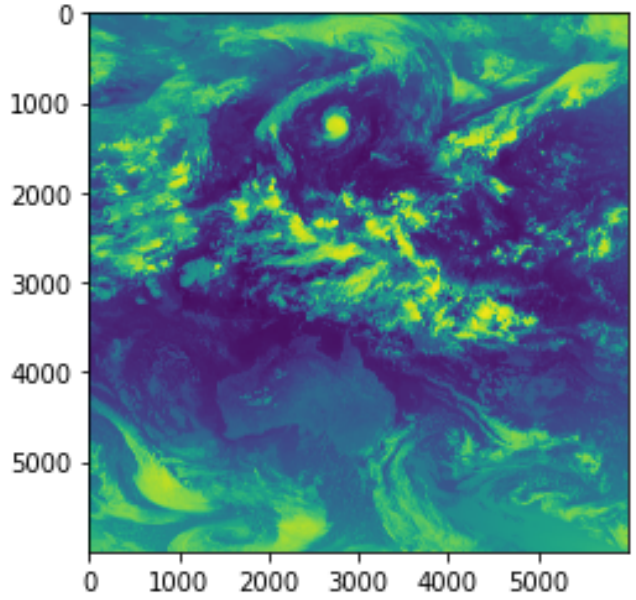
```
dataDN.shape
```

```
(6000, 6000)
```

このようにして 6000x6000 の 2 次元配列となった．この変数を画像で表示して確認してみる．2 次元データを図として表示するには matplotlib の plt.imshow()を使う．

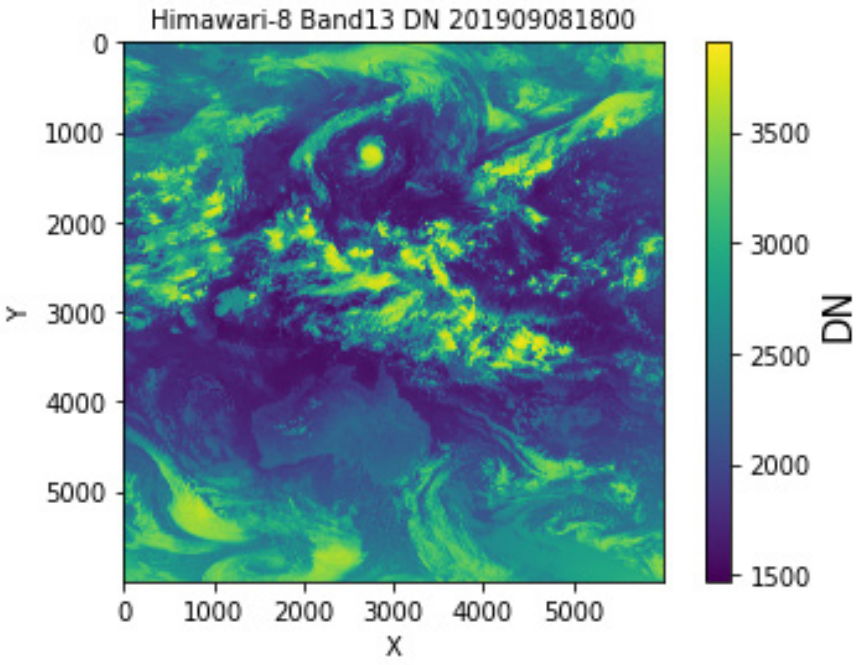
```
plt.imshow(dataDN)
```

```
<matplotlib.image.AxesImage at 0x7fc26eabe780>
```



上図のようにデータ全体を画像化することができた．オーストラリアの輪郭や，台風の丸い雲域を確認することができる．plt.imshow()のデフォルト設定のカラーマップでは，青色（黄色）ほど小さい（大きい）値となっている．x,y 軸の値はそれぞれデータ要素の行番号，列番号がそれぞれ対応している．x,y 軸にラベルを付記するには plt.xlabel, plt.ylabel といった関数を使って表示する．plt.colorbar()はカラーバーを表示するメソッドであり，set_label()をつけ加えることでカラーバーにラベルを表示している．

```
plt.imshow(dataDN)
plt.xlabel('X')
plt.ylabel('Y')
#font10
plt.title("Himawari-8 Band13 DN 201909081800", size=10)
# font15
plt.colorbar().set_label('DN',size=15);
```



表示の方法をカスタマイズするために，いくつかオプションを設定することができる．主なオプションを以下に挙げる．

imshow の主なオプション

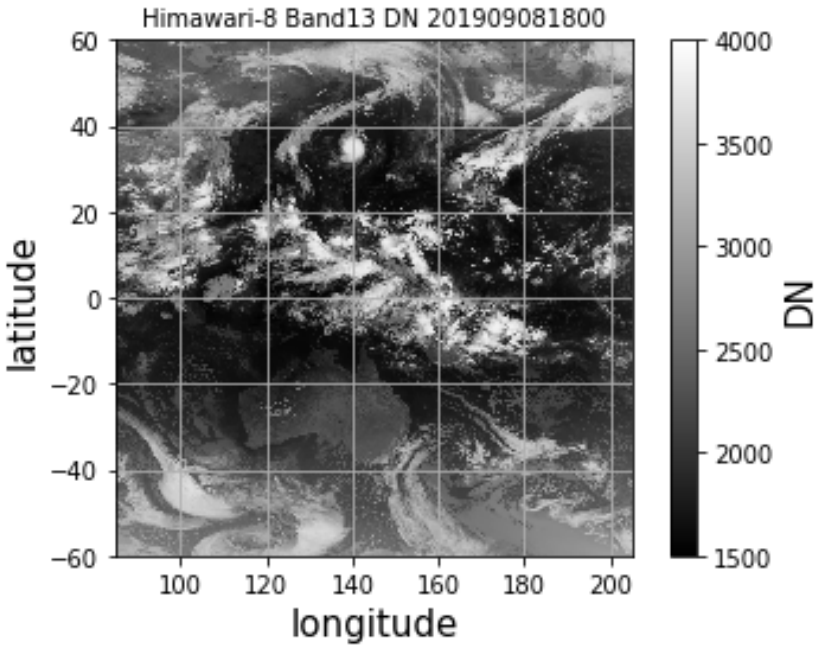
オプション	説明	値
vmin	カラースケールの最小値	例：1500
vmax	カラースケールの最大値	例：4000
extent	画像の左右下上座標指定	例：(85，205，-60，60)
cmap	カラーマップ	‘jet’，’ gray’，’ Greys’ など
Interpolation	補間方法	’none’，’nearest’，’bilinear’など
Origin	画像の上下起点	’upper’（上→下），’lower’（下→上）

◎ imshow の引数でオプション=値のように指定する．
【例】plt.imshow (data，vmin=0，interpolation=’none’)
カラーマップをグレースケールにして表示してみる．
ここでも 2-4 節のユーザー定義を利用して作図する．
なお，関数は同一プログラムで複数定義可能であるため，deff.ipynb に以下を追記する．（「ファイル」→「保存」したあと.py に変換すること，やり方は 2-4 節を参照)

```
def mkfigure(data,slon,elon,slat,elat,valmin,valmax,¥
valname,yyyy,mm,dd,hh,tt,ccmap,tname):
  plt.imshow(data,vmin=valmin,vmax=valmax,extent=(slon,¥
  elon,slat,elat),interpolation="None",origin="upper",¥
    cmap=ccmap)
  plt.colorbar().set_label(valname,size=15)
  TT="{title} {vname}
{year:04}{month:02}{day:02}{hour:02}{time:02}"¥
    .format(title=tname,vname=valname,year=yyyy,month=mm,¥
    day=dd,hour=hh,time=tt)
  plt.title(TT,size=10)
  plt.grid(True)
  plt.xlabel("longitude",size=15)
  plt.ylabel("latitude",size=15);
```

変換が終わったら別のプログラムで import し，関数を実行する．

```
from google.colab import drive
drive.mount('/content/drive')
import sys
sys.path.append("/content/drive/MyDrive/¥
Colab Notebooks")
import deff
deff.mkfigure(dataDN, 85,205,-60,60,1500,4000,¥
"DN",2019,9,8,18,0,"gray","Himawari-8 Band13")
```



3-4. 輝度温度変換

続いてルックアップテーブル（LUT）を用いてカウント値の物理量変換を行う．LUT は CEReS の FTP[8]で公開している．LUT のファイル名はグリッドデータのバンド名と同じである．DN 値と物理量（バンド 1～6：反射率（%），バンド 7～16：輝度温度（K））の対応関係が記録されている．
LUT のテキストデータは np.loadtxt() を用いて以下のように読み込むことができる．

```
DN,tbb = np.loadtxt(datapath + 'tir.01', unpack = True)
print(DN,tbb)
```

```
[0.000e+00 1.000e+00 2.000e+00 ... 4.048e+03 4.049e+03 4.050e+03] [330.966134
330.946875 330.927613 ... 117.385241 110.857281  69.998635]
```

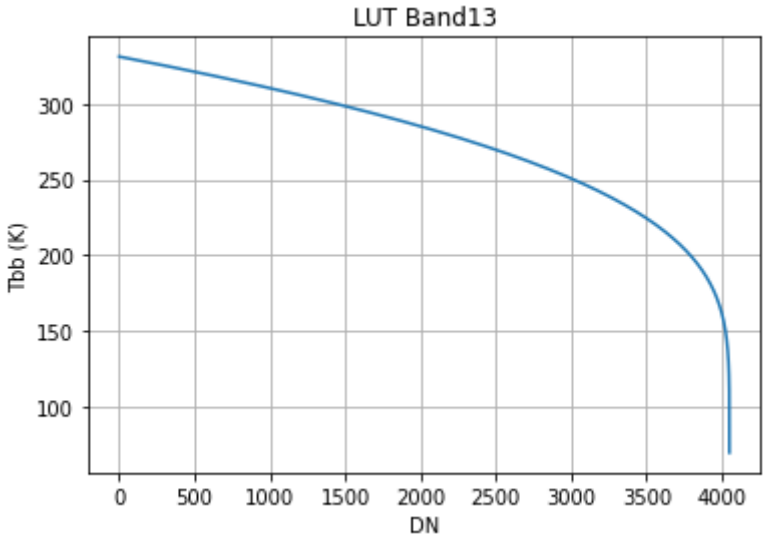
```
tbb.shape # Tbb(輝度温度の配列)
```

```
(4051,)
```

np.loadtxt() は CSV（カンマ区切り）や TSV（タブ区切り）などのテキストファイルを配列として読み込むことができる．unpack = True のオプションを付けることで要素ごとに配列へ格納することができる．
ルックアップテーブルの tir.01 ファイルには，1 カラム目には（DN 値），2 カラム目には（輝度温度値）が入っている．それぞれのカラムを変数「DN」，「tbb」 に代入する．LUT のデータをグラフで確認する．

```
plt.plot(DN,tbb)
plt.title("LUT Band13")
plt.xlabel('DN')
plt.ylabel('Tbb (K)')
plt.grid(True)
```

[8] ftp://hmwr829gr.cr.chiba-u.ac.jp/gridded/FD/support/count2tbb_v102.tgz



ルックアップテーブルを使うと DN 値に対応する物理量（tbb）が分かる．変数「tbb」のインデックス（0 から始まる要素番号）は対応する DN 値と同じである．ルックアップテーブルの一部を表にすると以下のようになる．

index	DN	tbb
0	0	330.9661
1	1	330.9469
2	2	330.9276
3	3	330.9083
4	4	330.8891
5	5	330.8698
6	6	330.8505
7	7	330.8313
8	8	330.812
9	9	330.7927
10	10	330.7734

NumPy 配列（ndarray）はインデックスを指定することによって要素を取り出せるため，tbb の要素番号に DN 値を入れると対応する物理量 tbb が得られる．

☞ DN 値が 0，1000，4050 のときの，対応する物理量を求めてみよう！

```
tbb[0],tbb[1000],tbb[4050]
```

```
(330.966134, 310.194917, 69.998635)
```

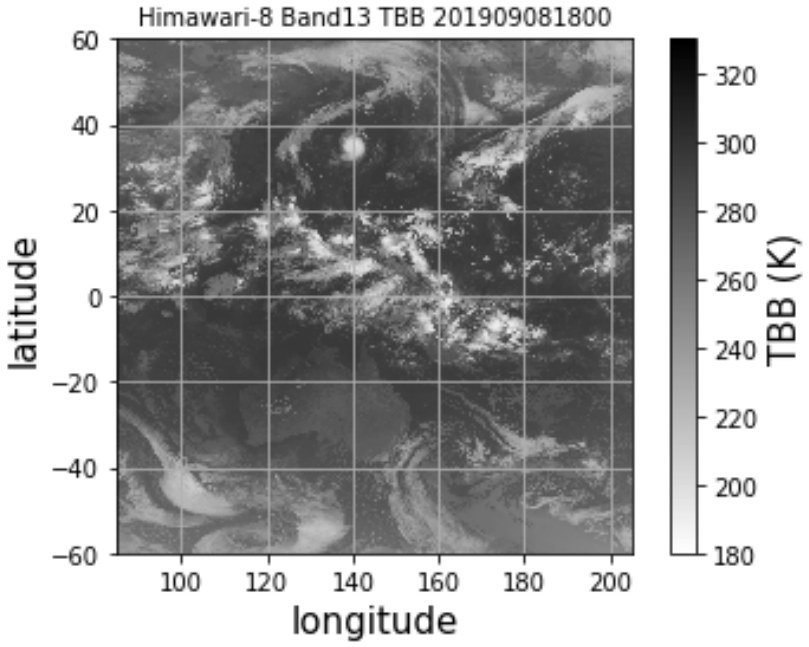
任意の DN 値から物理量変換の確認ができた．では次に Gridded データ全体の物理量変換を行う．

```
dataTBB = tbb[dataDN]
dataTBB
```

```
array([[272.876214, 272.876214, 272.876214, ..., 227.384489, 227.384489,
        227.384489],
       [272.811286, 272.811286, 272.811286, ..., 228.745101, 228.745101,
        227.196343],
       [273.070711, 272.876214, 272.876214, ..., 227.196343, 227.196343,
        227.196343],
       ...,
       [241.935802, 242.861431, 242.861431, ..., 249.485984, 249.485984,
        249.485984],
       [242.861431, 242.861431, 242.861431, ..., 248.202945, 248.202945,
        248.202945],
       [243.151015, 243.151015, 243.151015, ..., 249.310421, 249.310421,
        249.310421]])
```

変数「dataTBB」には物理量変換後の値が入っている．このデータを図で表示してみる．

```
deff.mkfigure(dataTBB,85,205,-60,60,180,330,"TBB (K)",¥
2019,9,8,18,0,"gray_r","Himawari-8 Band13")
```



ひまわりの全球の輝度温度データが得られた．先程の DN 値とは逆で小さい値ほど白い（明るい）色とするために色変化の向きを反転させている．

3-5. 画像の切り出し

ここでは，データをより詳細にみるために，領域の切り出しを行っていく．例として北緯 30°から 40°，東経 135°から 145°の領域を切り出す方法を示す．以下のようにして，領域のパラメータからグリッドのピクセル数に換算している．今回の Band13 は解像度が 0.02°であるため，resolution に 0.02 を指定している．（Band03 では 0.005，その他では 0.01 とする．）

```
lat_min = 30 # 切り出す範囲
lat_max = 40
lon_min = 135
lon_max = 145

resolution = 0.02
y_min = int((60 - lat_max)/resolution)
y_max = int((60 - lat_min)/resolution)
x_min = int((lon_min - 85.0)/resolution)
x_max = int((lon_max - 85.0)/resolution)
print(y_min,y_max,x_min,x_max)
```

```
1000 1500 2500 3000
```

上記のように切り出すピクセル値が表示された．この値を用いて配列から切り出しを行う．

```
dataTBB_tc=dataTBB[y_min:y_max,x_min:x_max]
```

```
array([[291.635065, 291.501853, 291.18144 , ..., 292.060192, 291.927527,
        292.033672],
       [290.187154, 288.498945, 286.865629, ..., 292.351457, 292.589159,
        292.325012],
       [290.187154, 288.498945, 286.865629, ..., 292.747326, 292.720982,
        293.036677],
       ...,
       [285.658048, 285.148102, 284.863641, ..., 289.970895, 286.305806,
        286.389982],
       [285.290021, 285.148102, 285.573242, ..., 289.073922, 284.206201,
        281.854397],
       [285.290021, 285.148102, 285.573242, ..., 286.893537, 283.890187,
        285.005975]])
```

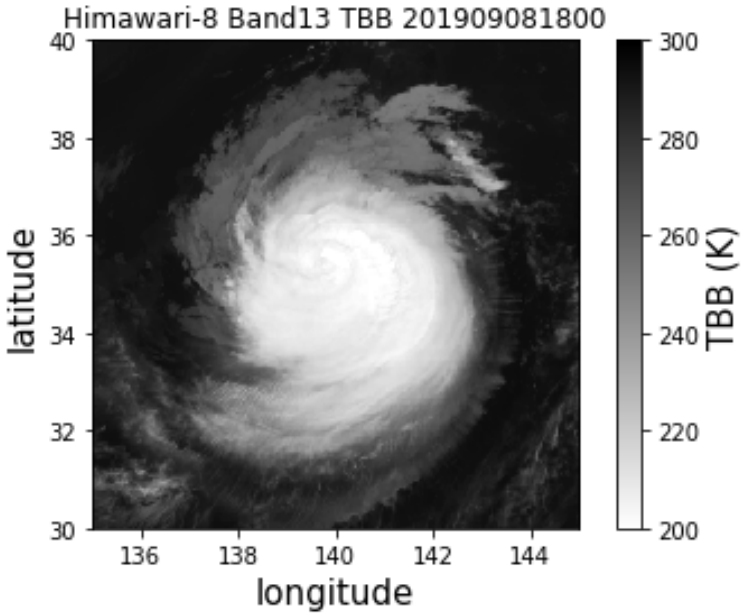
データの配列を確認してみよう．500x500 の領域が確認できる．

```
dataTBB_tc.shape
```

```
(500, 500)
```

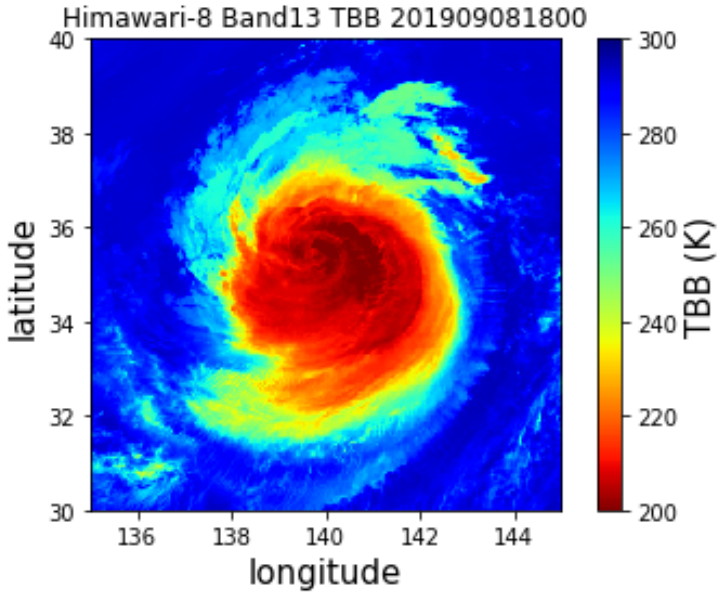
では画像を出してみよう．

```
deff.mkfigure(dataTBB_tc,lon_min,lon_max,¥
lat_min,lat_max,200,300,"TBB (k)",2019,9,8,18,0,¥
"gray_r","Himawari-8 Band13")
```



台風エリアが表示され，データの切り出しが確認できる.
台風中心付近の雲分布をより詳細に見るためにカラーマップを変更してみよう． 活発な雲域ほど赤く強調されるように配色してみる.

```
deff.mkfigure(dataTBB_tc,lon_min,lon_max,¥
lat_min,lat_max,200,300,"TBB (k)",2019,9,8,18,0,¥
"jet_r","Himawari-8 Band13")
```



台風中心付近から外側に螺旋状にのびる雲の様子が見やすくなった.

3-6. ひまわり動画の作成

ここまでは，ある時間のスナップショットの観測データを可視化する方法を紹介してきた．雲が時間とともに移り変わっていく様子はアニメーションとして眺めるとわかりやすい．ここでは一定時間間隔の複数のひまわり画像を作成し，gif 形式のアニメーション画像を作成する方法を紹介する．まずひまわりデータをダウンロードするにあたって，Google ドライブをマウントして wget モジュールをインストールする．

```
!pip install wget
```

続いて画像を作成する．まず画像読み込みに使われる PIL の Image を import する．

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from PIL import Image
import wget
import bz2
import os
```

画像を作成して保存する一連の作業を行う関数を定義する（ユーザー定義関数）(2.4 参照)．この関数の引数には（2 次元配列に変換済みのひまわりデータ，データの最大値，最小値，データ要素名，日時，カラースケール名，画像タイトル）を与えると動作し，対応する画像が出力される仕組みとなっている．

```
def mkfigure(data,slon,elon,slat,elat,valmin,valmax,¥
  valname,yyyy,mm,dd,hh,tt,ccmap,tname):
  plt.imshow(data,vmin=valmin,vmax=valmax,extent=(slon,¥
    elon,slat,elat),interpolation="None",origin="upper",¥
    cmap=ccmap)
  plt.colorbar().set_label(valname,size=15)
  TT="{title} ¥
  {vname} ¥
  {year:04}{month:02}{day:02}{hour:02}{time:02}".format(¥
    title=tname,vname=valname,year=yyyy,month=mm,day=dd,¥
    hour=hh,time=tt)
  fname="{year:04}{month:02}{day:02}{hour:02}¥
    {minute:02}.png".format(year=yyyy,month=mm,day=dd,¥
    hour=hh,minute=tt)
  plt.title(TT,size=10)
  plt.grid(True)
  plt.xlabel("longitude",size=15)
  plt.ylabel("latitude",size=15)
  plt.savefig(fname);          #画像の保存
  plt.show()                   #図を分けるため，必ず使用する．
```

続いて，輝度温度に変換するための LookUpTable の読み込み，切り出し範囲の指定+ピクセル番号の変換，解像度，年月の指定，gif 動画作成のための配列を作成する．images は画像をまとめるための配列である．概念としては写真を時間順に重ね，束にした画像をパラパラ漫画のように見せることである．

```
lut=np.loadtxt("/content/drive/MyDrive/Himawari_Data/¥
    tir.01",usecols=(1,)) #LUT 読み込み
images=[]   #画像をまとめて動画にするためのもの
lat_min = 21.5 # 南端の緯度
lat_max = 48.5 # 北端の緯度
lon_max = 152 # 東端の経度
lon_min = 119 # 西端の経度
resolution = 0.02 # 1 グリッドあたりの解像度 (度)
y_min = int((60 - lat_max)/resolution)
y_max = int((60 - lat_min)/resolution)
x_min = int((lon_min - 85.0)/resolution)
x_max = int((lon_max - 85.0)/resolution)
```

以下のコードは複数データを取得するサンプルコードを示している．今回は 2019 年 10 月 5 日 0 時 00 分から 10 月 14 日 18 時 00 分（UTC）の範囲の赤外バンド（Band13）データを 6 時間間隔でダウンロードする(3.2 節を参照)．今回は 6 時間毎のデータを取得するため，時間のループ（変数：h）のレンジを 0,24,6 と設定しており，時間間隔を変更したい場合は 6 の部分を任意に変更していただきたい．for ループを用いてまとめてデータを読み込み→輝度温度変換→画像化+保存→画像読み込み→images 変数にスタックという流れで進める．os モジュールの os.remove はディスクを圧迫しないためにファイルやデータを削除するために用いている．

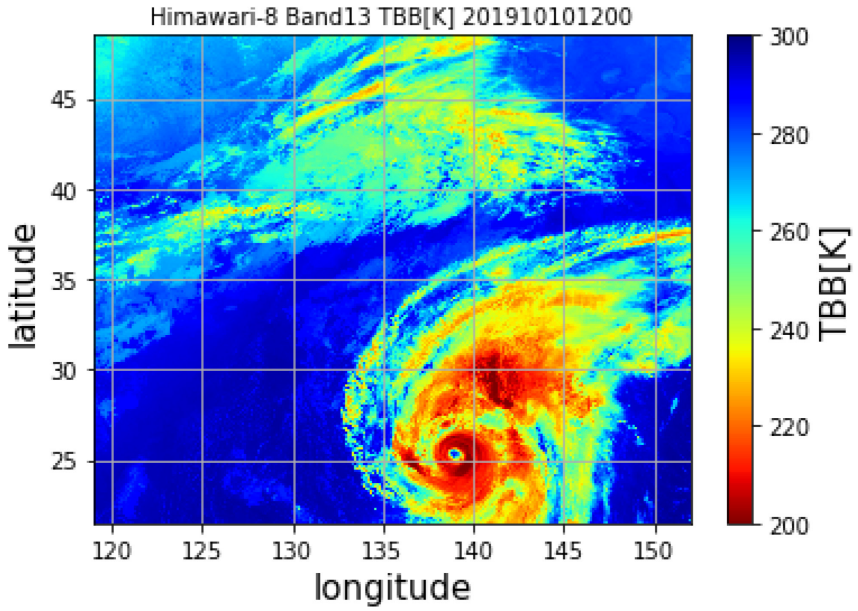
```
FTP="ftp://hmwr829gr.cr.chiba-
u.ac.jp/gridded/FD/V20190123/"
B="TIR"
b="tir"
bno=1
for y in range(2019,2020) :
  for m in range(10,11) :
    for d in range(5,15) :
      for h in range(0,24,6) :
        for mi in range(0,10,10):
          tt=FTP+"{year:04}{month:02}/{band}/".format(¥
            year=y,month=m,band=B)
          DATE="{year:04}{month:02}{day:02}{hour:02}¥
            {min:02}".format(year=y,month=m,day=d,hour=h,¥
            min=mi)
          fname=DATE+¥
            ".{band}.{bandno:02}.fld.geoss".format(¥
```

```
            band=b,bandno=bno)
        fnamebz=fname+".bz2"
        ttt=tt+fnamebz
        print(ttt)
        wget.download(ttt)
        zipfile = bz2.BZ2File(fnamebz)
        data = zipfile.read()
        open(fname, "wb").write(data)
        os.remove(fnamebz)
        #データ読み込み+6000*6000 に変換
        data=np.fromfile(fname,dtype=">u2").reshape(6000,
        6000)
        datatbb=lut[data] #物理量変換
        #画像の切り出し(日本域)
        datatbbjp=datatbb[y_min:y_max,x_min:x_max]
        mkfigure(datatbbjp,lon_min,lon_max,lat_min,¥
          lat_max,200,300,"TBB[K]",y,m,d,h,mi,"jet_r",¥
          "Himawari-8 Band13")   #画像化+保存
        os.remove(fname)
        fname="{year:04}{month:02}{day:02}{hour:02}¥
          {minute:02}.png".format(year=y,month=m,day=d,¥
          hour=h,minute=mi)  #画像データ名
        img=Image.open(fname)  #画像を開く
        images.append(img)  #開いた画像を images に加える
        os.remove(fname)
        del data,datatbb,datatbbjp
```

gif 動画を保存する．Duration は動画再生時の 1 フレームあたりの持続時間を示し，値を小さくすると動きが早く，値を大きくするとゆっくりと画像を確認することができる．loop を指定すると再生回数を指定することができ，loop=0 とすると無限ループ再生となる．

```
#gif データ書き出し，loop=0 で無限ループ再生，duration で間隔指定)
images[0].save("/content/drive/MyDrive/¥
    Hagibismovement.gif",save_all=True,¥
  append_images=images[1:],optimize=False,duration=200,¥
  loop=0)
```

Google ドライブに移動すると gif 画像が作成されているのが確認できる．



課題：ひまわりデータを用いた 2022 年トンガ火山噴火事例の描画

2022 年 1 月 15 日(日本時間)午後 1 時ごろ，トンガ諸島付近のフンガ・トンガ-フンガ・ハアパイ火山で大規模な噴火が発生した[9]．この噴火では噴煙が高度 16km にまで達した．日本でも岩手県久慈港で最大 1.1m の津波を観測し，日本各地で噴火の衝撃波による気圧変化が観測された[10]．本演習の課題ではひまわり 8 号のデータを用いて，噴煙の広がりや衝撃波の伝達を描画していただきたい．

第 3 章で学んできたひまわりの可視化とアニメーションの方法を応用して，トンガ火山噴火事例を作成してみてください．ルックアップテーブルは’/content/drive/MyDrive/Himawari_Data’フォルダに置いています．使用するバンドは Band13(10.4μm)とし，切り出しに使用する緯度経度は緯度-25°N から 15°S，経度 180°E から 190°E としてください．データは

[9] https://www.jma.go.jp/jma/press/2201/16b/kaisetsu202201161415.pdf

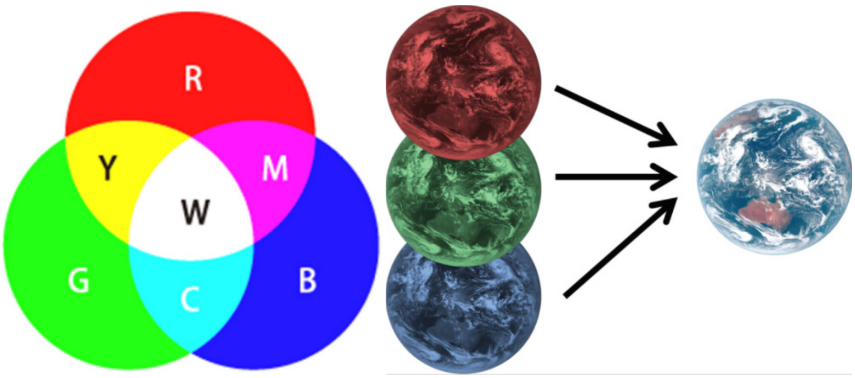
[10] https://jp.weathernews.com/news/38708/

ルックアップテーブル同様，/content/drive/MyDrive/Himawari_Data に置いてあります．
以下の手順でやってみてください．
・ディレクトリパスを確認する（3-3 節）
・物理量変換を行う（3-4 節）
・画像化する（3-4 節）
・切り出しを行う（3-5 節）
・アニメーション化する（3-6 節）

Appendix-1 ひまわりデータの画像合成

1-1. RGB 合成について

RGB 合成画像は，光の三原色（赤・緑・青）の性質を利用して 3 種類の衛星画像に赤(R)，緑(G)，青(B)を割り当てて 3 種類の衛星画像を 1 つのカラー衛星画像として表現する方法のこと．3種類の衛星画像の特徴が、1 枚のカラー画像だけでわかりやすく表現することができる点で有用な方法である．



☞ この章を始めるにあたって，事前に共有ドライブを Colab にマウントし，以下のコードを入力して準備を行う．

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

1-2. Gridded データの読み込みと物理量変換

例として 2020 年 7 月 4 日 00 時 00 分（UTC）の観測データを用いて，可視画像のカラー合成を行う．可視青バンド（vis.01），緑バンド（vis.02），赤バンド（ext.01）の gridded データを読み込んでいく．まずはディレクトリ名を含めたファイル名の変数を作る．

```
datapath = '/content/drive/MyDrive/Himawari_Data/'
YYYYMMDDHHmm = '202007040000'
```

今回は可視バンドの赤(0.64 μm, Band03, 0.005°解像度)，緑(0.51 μm, Band02, 0.01°解像度)，青(0.47 μm, Band01, 0.01°解像度)をそれぞれ画像の赤・緑・青チャンネルに割当ててカラー合成を行う．次のようにして各バンドのデータファイルの拡張子を指定する．

```
# Band03
file_R = datapath + YYYYMMDDHHmm + '.ext.01.fld.geoss'
# Band02
file_G = datapath + YYYYMMDDHHmm + '.vis.02.fld.geoss'
# Band01
file_B = datapath + YYYYMMDDHHmm + '.vis.01.fld.geoss'
```

R・G・B 各バンドの変数にデータを読み込む.

```
dataDN_R = np.fromfile( file_R, dtype='>u2')
dataDN_G = np.fromfile( file_G, dtype='>u2')
dataDN_B = np.fromfile( file_B, dtype='>u2')
```

配列を 2 次元に変換する.

```
dataDN_G = dataDN_G.reshape(12000,12000)
dataDN_B = dataDN_B.reshape(12000,12000)
dataDN_R = dataDN_R.reshape(24000,24000)
```

それぞれのバンドに対応した．物理量変換のためのルックアップテーブル（LUT）を読み込む.

```
albedo_R = np.loadtxt(datapath + 'ext.01',usecols=(1,))
albedo_G = np.loadtxt(datapath + 'vis.02',usecols=(1,))
albedo_B = np.loadtxt(datapath + 'vis.01',usecols=(1,))
```

LUT のテキストデータは DN 値（0 列目）とそれに対応する albedo（1 列目）の2カラムで構成され，DN 値は 0 から 1 ずつ順番に増加する形式で書かれている．usecols=(1,)として albedo（1 列目）を順に読む．インデックス番号が DN 値に対応している（0 行目の DN 値は 0）.

```
albedo_R
```

```
array([ -1.176471,  -1.117647,  -1.058824, ..., 119.117646, 119.176469,
       119.235293])
```

各バンドの DN 値を物理量に変換する.

```
dataALBEDO_R = albedo_R[dataDN_R]
del dataDN_R #サイズが大きいため，メモリ確保のため配列を削除する
dataALBEDO_G = albedo_G[dataDN_G]
del dataDN_G
dataALBEDO_B = albedo_B[dataDN_B]
del dataDN_B
```

次に 0.005°解像度（24000x24000 のサイズ）の高い Band03 データをその他のデータと同じ解像度の 0.01°解像度（12000x12000 のサイズ）へ変換する．縦，横のデータ数をそれぞれ半分にするには，単に 1 つおきに間引くだけでも良いが，ここでは縦，横それぞれで 2 画素ずつのペアを組み（2×2），各ペアの平均値を計算するビニング（binning，複数のピクセルを

結合して，単一のピクセルに変換すること）を行う．ビニングの概念は以下の図の通りである．

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

0.5	2.5	4.5
6.5	8.5	10.5
12.5	14.5	16.5
18.5	20.5	22.5

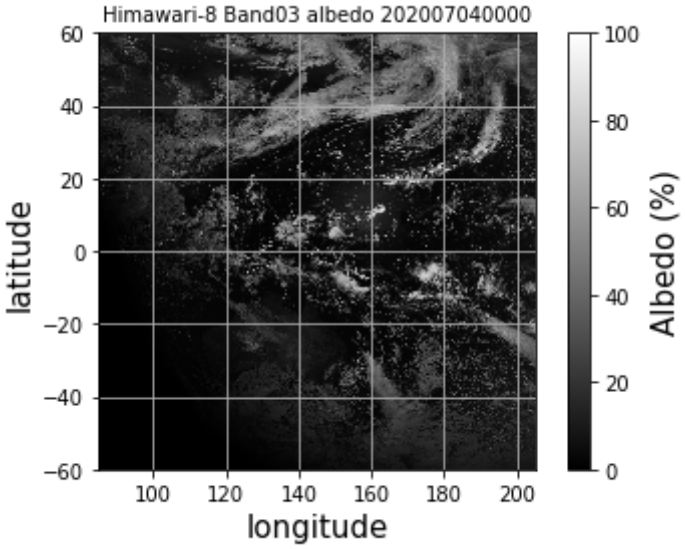
3.5	5.5	7.5
15.5	17.5	19.5

```
dataALBEDO_R=dataALBEDO_R.reshape(12000,2,12000,2).mean¥
  (-1).mean(1)
dataALBEDO_R.shape
```

```
(12000, 12000)
```

もともと 24000x24000 のサイズであった band03 のデータは Band01，Band02 のサイズと同様に 12000x12000 のデータサイズとなった．解像度変換を行ったデータを表示して確認する．3-3 節の関数（deff.ipynb）を用いて表示する．なお deff.ipynb は各自で”Colab Notebooks”フォルダに作成しないと動かない仕様になっています．

```
import sys
sys.path.append("/content/drive/MyDrive/Colab Notebooks")
import deff
deff.mkfigure(dataALBEDO_R,85,205,-
60,60,0,100,"Albedo",2020,7,4,0,0,"gray",¥
  "Himawari-8 Band03")
```



図のように，リサイズしたデータが正常にできていることを確認した．このままではデータサイズ（領域）が大きいので，表示範囲を切り出す．

```
lat_max = -5 # 範囲の指定
lat_min = -35 # 緯度の最小
lon_max = 160 # 経度の最大
lon_min = 110 # 経度の最小
resolution = 0.01
y_min = int((60 - lat_max)/resolution)
y_max = int((60 - lat_min)/resolution)
x_min = int((lon_min - 85.0)/resolution)
x_max = int((lon_max - 85.0)/resolution)
print(y_min,y_max,x_min,x_max)
```

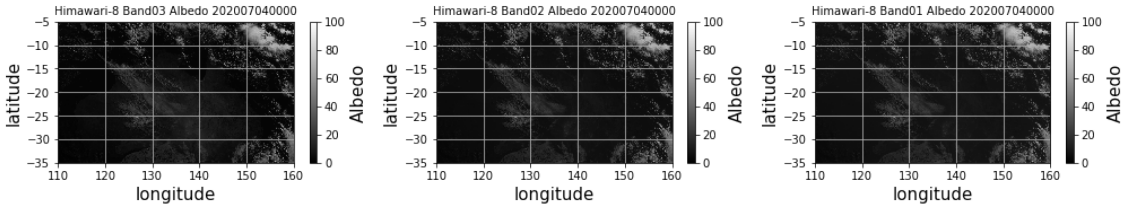
6500 9500 2500 7500 と表示され，x，y の最大，最小が確認できた．
各バンドについて，指定した領域で切り出しを行う．領域を限定したデータは変数に_area1 を付して表示することにする．

```
dataALBEDO_R_area1 = dataALBEDO_R¥
  [y_min:y_max,x_min:x_max]
dataALBEDO_G_area1 = dataALBEDO_G¥
  [y_min:y_max,x_min:x_max]
dataALBEDO_B_area1 = dataALBEDO_B¥
  [y_min:y_max,x_min:x_max]
```

それぞれのバンドを画像で表示してみる．

```
plt.figure(figsize=(17.0,2.3))
#左
plt.subplot(1,3,1)
deff.mkfigure(dataALBEDO_R_area1,lon_min,lon_max,¥
```

```
  lat_min,lat_max,0,100,¥
  "Albedo",2020,7,4,0,0,"gray","Himawari-8 Band03")
#中
plt.subplot(1,3,2)
deff.mkfigure(dataALBEDO_G_area1,lon_min,lon_max,¥
  lat_min,lat_max,0,100,¥
  "Albedo",2020,7,4,0,0,"gray","Himawari-8 Band02")
#右
plt.subplot(1,3,3)
deff.mkfigure(dataALBEDO_B_area1,lon_min,lon_max,¥
  lat_min,lat_max,0,100,¥
  "Albedo",2020,7,4,0,0,"gray","Himawari-8 Band01")
```

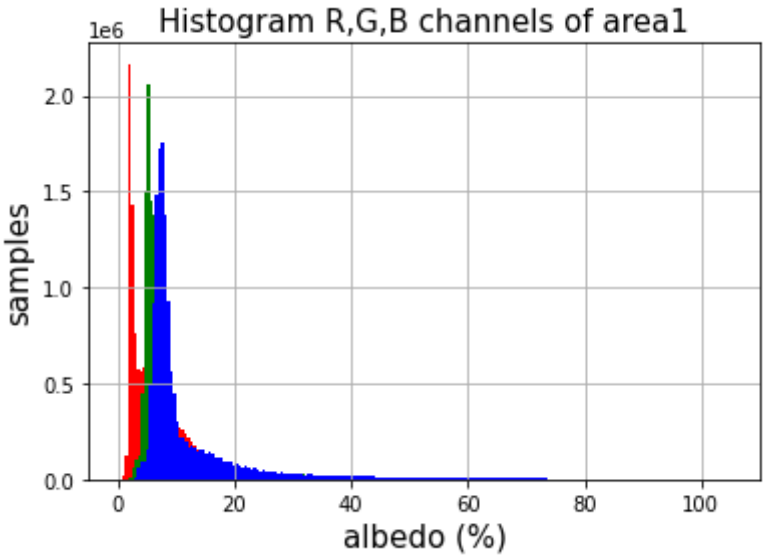


上図のように領域の切り出しを確認した.

1-3. 各バンドの反射率の分布と RGB 合成

次に各バンドに含まれる反射率の分布について plt.hist() を用いてヒストグラムで確認する.

```
plt.hist(dataALBEDO_R_area1.flatten(),bins=200, ¥
  color='r') # 200bin で表示
plt.hist(dataALBEDO_G_area1.flatten(),bins=200, ¥
  color='g')
plt.hist(dataALBEDO_B_area1.flatten(),bins=200, ¥
  color='b')
plt.title("Histogram R,G,B channels of area1", size=15)
plt.xlabel('albedo (%)',size=15)
plt.ylabel('samples',size=15)
plt.grid(True)
```

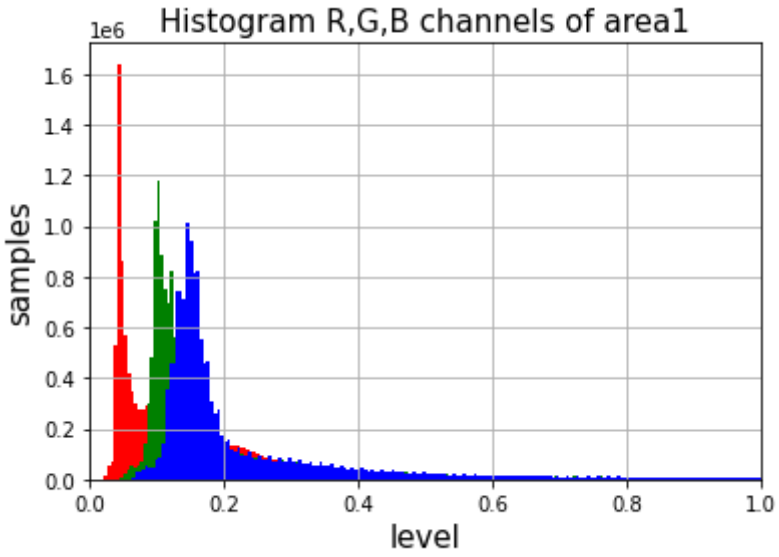


各バンドとも albedo 値の分布はおよそ 0 から 75%に収まっていることがわかる．比較的輝度の小さい部分にサンプルが分布しているため，画像としては暗い画像となる．imshow()を用いて RGB 合成するには 1 バイト符号なし整数または 0 以上 1 以下の浮動小数点数データに限られる．そこで RGB カラー画像表示を行うためにスケーリングを行い，0 から 1 に分布するように補正する．ここでは0から1に分布するようにするために 50 分の 1 してスケーリングを行った．

```
dataALBEDO_R_area1 = dataALBEDO_R¥
  [y_min:y_max,x_min:x_max] * 0.02
dataALBEDO_G_area1 = dataALBEDO_G¥
  [y_min:y_max,x_min:x_max] * 0.02
dataALBEDO_B_area1 = dataALBEDO_B¥
  [y_min:y_max,x_min:x_max] * 0.02
```

スケーリング後の反射率の分布について plt.hist()を用いてヒストグラムで確認する．

```
plt.xlim(0.0, 1.0) # 0から1に限定する
plt.hist(dataALBEDO_R_area1.flatten(),bins=400, ¥
  color='r')
plt.hist(dataALBEDO_G_area1.flatten(),bins=400, ¥
  color='g')
plt.hist(dataALBEDO_B_area1.flatten(),bins=400, ¥
  color='b')
plt.title("Histogram R,G,B channels of area1", size=15)
plt.xlabel('level',size=15)
plt.ylabel('samples',size=15)
plt.grid(True)
```



各バンドを0から１に割り当てて画像作成を行った．ではこれら３つのデータを np.dstack() を使って合成(stack)する．

```
dataRGB_area1=np.dstack((dataALBEDO_R_area1,¥
  dataALBEDO_G_area1,dataALBEDO_B_area1))
```

```
dataRGB_area1.shape
```

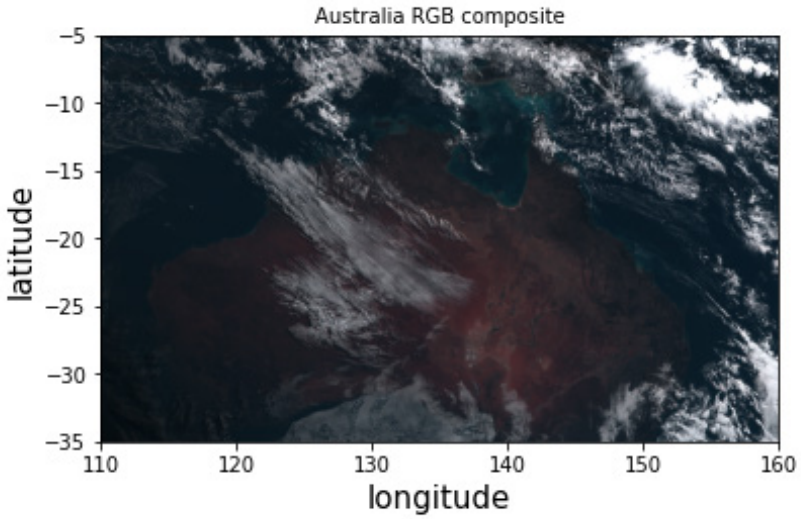
```
(3000, 5000, 3)
```

データ範囲を０から１に収めるために clip(0,1)とする．画素毎に R，G，B の反射率の最大値を求め，最大値が１より大きい場合はその画素を「白」にするために R，G，B の全反射率を１に修正している．

```
dataRGB_area1=dataRGB_area1.clip (0,1)
```

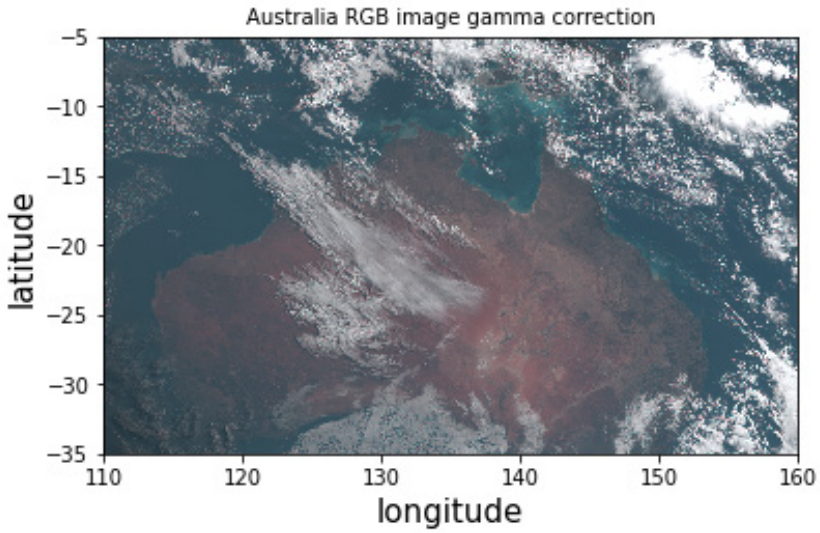
重ねた（Stack）データを画像で確認する．

```
def mkrgb(data,lon_min,lon_max,lat_min,lat_max):
  plt.imshow(data,extent=(lon_min,lon_max,lat_min,¥
    lat_max),origin='upper',interpolation='none')
  plt.title("Australia RGB composite", size=10)
  plt.xlabel('longitude',size=15)
  plt.ylabel('latitude',size=15)

mkrgb(dataRGB_area1,lon_min,lon_max,lat_min,lat_max)
```



データをそのまま描画すると，やや暗い画像が得られる．
画像全体に対して明るさ（輝度）を増加させると，もともと暗く不明瞭であった部分は見やすくなる一方，明るかった部分は白飛びして（図の右上の雲の部分）明るい部分の詳細を判別することは難しい．ピクセル値の最小値と最大値を変えずに，画像全体を明るく（暗く）表示したい場合は，$x' = x^{1/\gamma}$のような関係式で表されるガンマ補正によって補正できる．γ の値が 1 より小さい（大きい）場合，画像全体が暗く（明るく）なり，ピクセル輝度の高い（低い）部分が強調される．標準的なディスプレイの γ の値としては 2.2 程度が適当である．累乗を計算する np.power() を用いて RGB データにガンマ補正を施す．

```
mkrgb(np.power(dataRGB_area1,(1.0/2.0)),lon_min,lon_max,lat_min,lat_max)
```



雲だけでなく，地表面の様子が鮮明に確認できるようになった．参考文献に掲載している気象庁資料「第６章ひまわり８号 RGB 合成画像の基礎」では RGB 合成画像の基礎やバンドの組み合わせに関する説明が詳しいので合成画像を作成する際には参考にされたい．

Appendix-2：PyGMT によるデータの描画

2-1. GMT について

Shell 上で動作するマップ描画ツールには GMT(Generic Mapping Tool)[11]があり地球物理の分野でよく用いられてきた．PostScript (EPS)として出力するためきれいな図が得られるのが特徴である．Python でマップを描画するには Basemap や cartopy などを用いる方法があるが，GMT6.x 系ではこれまでの GMT4.x 系や GMT5.x と比べて機能強化がなされ Python インターフェースによる PyGMT でのきれいな図の描画も可能となったことで，データの読み込みから可視化までの一連の作業がしやすくなった．
データテーブルには配列(numpy.ndarray)や pandas.DataFrame，グリッドには xarray.DataArray を使用することができる．使用方法やサンプルは PyGMT のサイト[12]に掲載されているので参考にされたい．Colab 環境でも PyGMT を利用することができるので，ここでは Colab 環境への環境設定と PyGMT による描画の例を紹介する．

2-2. PyGMT の導入

以下のコードは GMT6 を Colab にインストールするものである．Colab 上で apt-get で入るのは GMT5.x であり PyGMT による描画はできないため，ソースから GMT6.x を入れている．それ以外のライブラリや海岸線データは apt コマンドを使用して導入している．

```
# GMT6 and PyGMT の準備　(May 2022)
# 最初とランタイムが切れたときのみ実行する

# load libraries and coastlines
!sudo apt install -y gmt-gshhg gmt-dcw build-essential¥
  cmake libcurl4-gnutls-dev libnetcdf-dev gdal-bin¥
    libgdal-dev libfftw3-dev libpcre3-dev liblapack-dev¥
 libblas-dev libglib2.0-dev ghostscript  graphicsmagick¥
   ffmpeg xdg-utils
GMT_VERSION = "6.3.0"   #現在の最新バージョンは 6.3.0 です
# download the code
#Shell コマンドですので、クォーテーションマークは要らないです
!wget
```

[11] https://www.generic-mapping-tools.org/

[12] https://www.pygmt.org/latest/

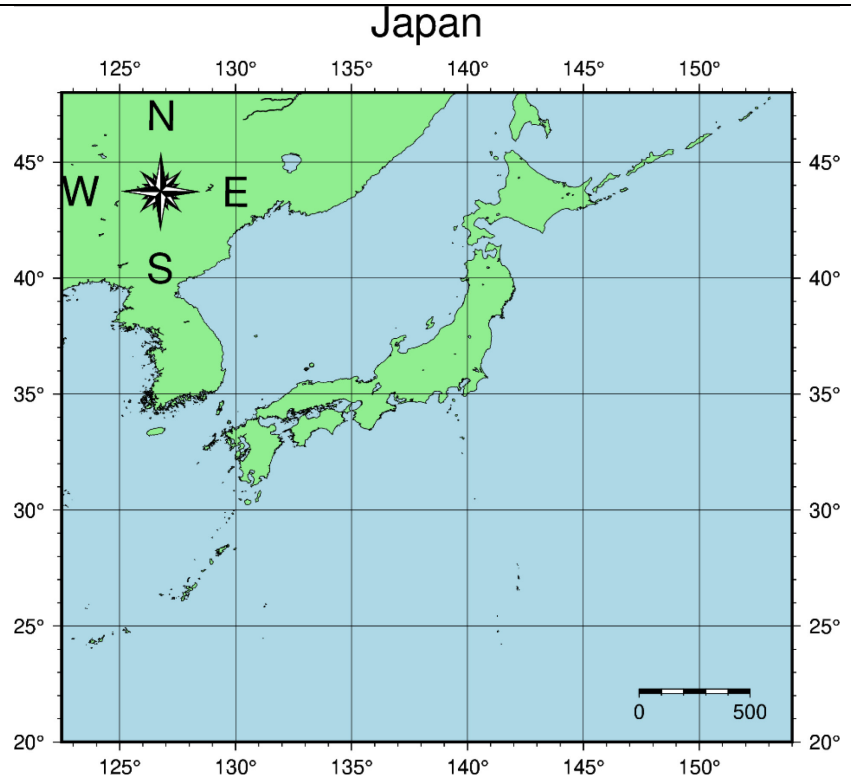
```
https://github.com/GenericMappingTools/gmt/releases/¥
  download/{GMT_VERSION}/gmt-{GMT_VERSION}-src.tar.gz
!tar xzvf gmt-{GMT_VERSION}-src.tar.gz

# build and install
%cd gmt-{GMT_VERSION}
!cat ./cmake/ConfigUserTemplate.cmake
> ./cmake/ConfigUser.cmake
!mkdir build
%cd build
!sudo cmake -DCMAKE_INSTALL_PREFIX=/usr/local¥
   -DCMAKE_BUILD_TYPE=RelWithDebInfo ..
# j オプションで同時に実行できるジョブの数（並行処理の数）を指定
!sudo make -j2
!sudo make -j2 install

# load the PyGMT
!pip install pygmt
```

2-3. 地図の描画

GMT6 がインストールされたら，マップの描画方法を確認する．最初に，pygmt をインポートする．config を使用して gmt の設定を指定するが，ここでは地図のフレームを実線にするために `MAP_FRAME_TYPE="plain"`と指定している(設定しない場合，黒白の破線フレームになる)．次に，basemap を使用して描画する領域を指定する．描画領域の lat,lon の最大・最小を指定して `region=[122.5, 154, 20, 48]`とし `projection` で地図投影の種類を指定している．`Q15c` は横幅 15cm で図を描画することを意味する．`fig.coast` で海岸線を描き，`land="lightgreen", water="lightblue"`で陸，海を指定した色で塗りつぶす．shorelines=True で海岸線を描画する．map_scale の x はスケールの位置，c20 は 20° N におけるスケール，w500 は 500km のスケール，+f でスケールバーを黒白のフレームに変更している．グリッドラインを自動で付加するため `frame=["ag"]`としている．図タイトルは `frame=["+tJapan"]`とすることで加えることができる．`rose` は図左上の方位記号を作成し，`jTL` は左上(TopLeft の意味)，`w0.6i` は幅 0.6 インチ，`f` は方角の細かさで，3 にすると 16 方位になる．+l で方角ラベルの表示，`o0.5i` で対角線上に 0.5 インチだけ全体的に位置をずらしている．

```
# 地図の描画確認
import pygmt
fig = pygmt.Figure()
```

```
pygmt.config(MAP_FRAME_TYPE="plain") # set the frame solid
line
fig.basemap(region=[122.5, 154, 20, 48], ¥
  projection="Q15c", frame=True)
fig.coast(land="lightgreen", water="lightblue", ¥
  shorelines=True, map_scale="x13/1+c20+w500+f")
fig.basemap(frame=["ag", "+tJapan"])
fig.basemap(rose="jTL+w0.6i+f3+l+o0.5i")
fig.show()
```



GMT6 の導入がうまくいくと，上のようなマップが表示される．
以下に，PyGMT で使える主なコマンドを簡単に示す．GMT6 オリジナルのコマンドと同じか，”ps”を省略したものが多い．

PyGMT コマンド	GMT コマンド	用途
config	gmtset	GMT パラメータの設定を変更する
basemap	psbasemap	マップの描画設定を変更する
coast	pscoast	海岸線を描く
grdimage	grdimage	緯度経度データを含むデータを描画する
colorbar	psscale	カラーバーを描画する
text	pstext	文字列を描画する

savefig	(psconvert)	画像として保存する
makecpt	makecpt	カラーパレットを作成する
plot	psxy	テキストデータをプロットする
contour	grdcontour	等値線を描画する

2-4. ひまわりデータの描画

ひまわりデータをダウンロードして tbb 変換，GMT で描画する方法を示す．まずデータ取得と bz2 形式からの読み出しを行う．“3.3.ひまわり 8 号 Gridded データの描画”では np.fromfile を使用したが，ここでは別の関数(frombuffer 関数)を使用した．frombuffer 関数は，メモリのバイト列を直接読み込んで Numpy 配列に変換するため，大容量のデータをコピーせずに処理することが可能である．処理速度の高速化が期待できるため，大容量のデータを扱う方には便利な関数である．

```
# ひまわりの DL と読み込み
!pip install wget
import wget
import bz2
import numpy as np
import os
# 2015/7/7 00:00 Band13 を例にダウンロードする
wget.download("ftp://hmwr829gr.cr.¥
  chiba-u.ac.jp/gridded/FD/V20151105/201507/TIR/¥
  201507070000.tir.01.fld.geoss.bz2")
fname = "201507070000.tir.01.fld.geoss.bz2"

# bz2 形式を bigendian 2byte unsigned integer で読み，6000x6000 の numpy 配列にする
dbuf = bz2.BZ2File(fname).read()
dataDN =
np.frombuffer(dbuf,dtype='>u2').reshape(6000,6000)
os.remove(fname)
dataDN
```

データは dataDN として２次元の配列に入った．
続いて，Tbb 変換を行うため，ルックアップテーブル（LUT）をダウンロードする．tgz ファイルとしてアーカイブ・圧縮されているので，tarfile モジュールを用いて，展開・解凍する．r:gz は gzip 圧縮で読み込み用にオープンすることを意味する．extractall ですべての圧縮されたファイルをディレクトリ(path=’src’)に抽出する．with は開始と終了がセットになった処理に使うも

ので，ここでは圧縮ファイルからデータを抽出するために Open したあと，自動的にファイルを閉じてくれる(close)．今回は Band13 を利用するので，LUT は tir.01 を使用する．

```
# download the look-up table
import wget
import tarfile
wget.download('ftp://hmwr829gr.cr.chiba-
u.ac.jp/gridded/FD/support/count2tbb_v102.tgz')
# 解凍
with tarfile.open('count2tbb_v102.tgz', 'r:gz') as t:
    t.extractall(path='src')
%cd src
DN,tbb = np.loadtxt('tir.01', unpack = True)
print(DN,tbb)

# Tbb 変換
dataTBB = tbb[dataDN]
dataTBB
```

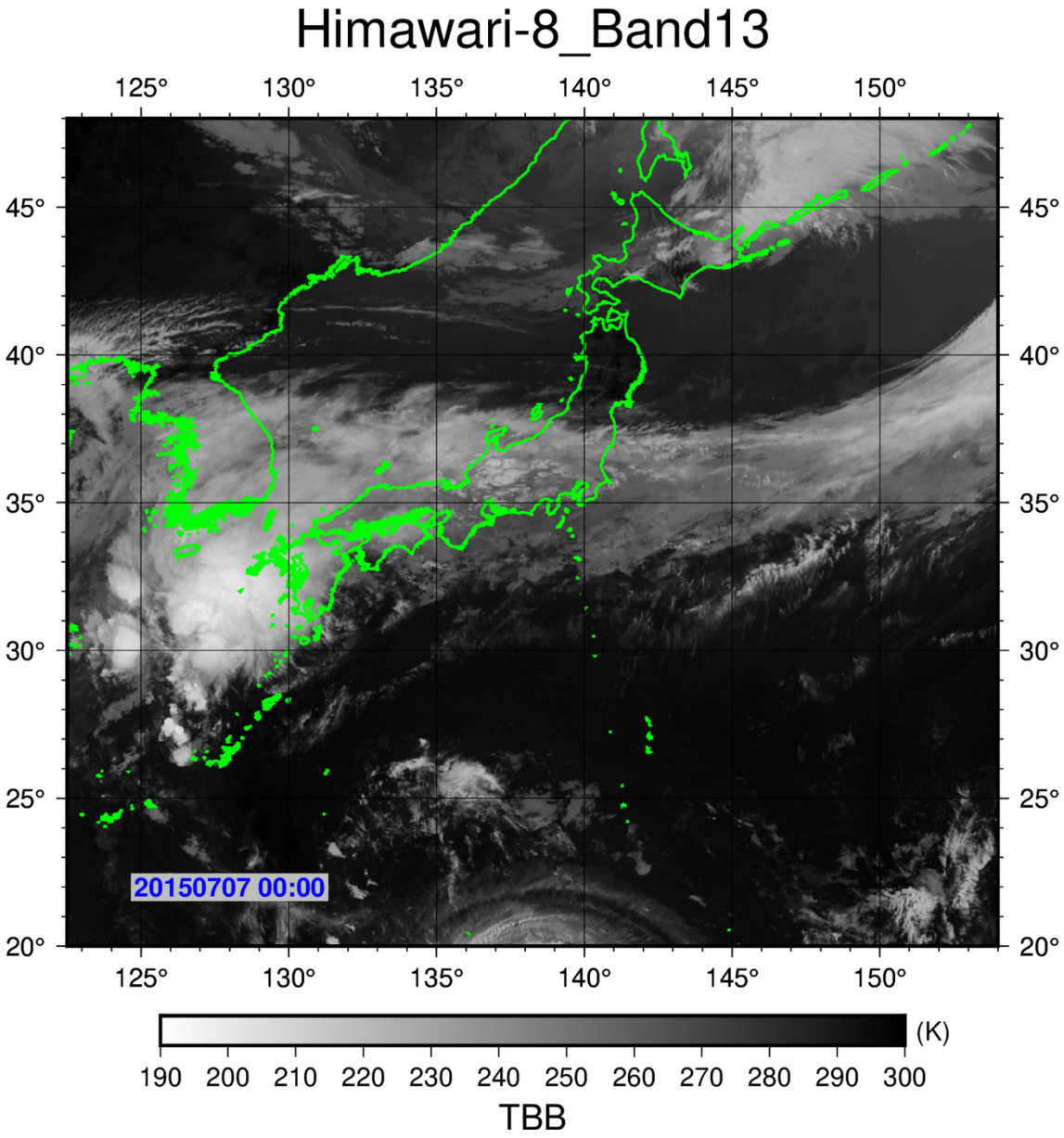
データは dataTBB として２次元の配列に入った．
PyGMT で描画を行うために，配列に lat,lon 情報を付加した xarray に変換する．

```
# TBB を xarray に変換する
import xarray as xr
# data に対応する緯度経度を定義する
lon1d = np.arange(85.0, 205.0, 0.02).astype(np.float64)
lat1d = np.arange(60.0, -60.0, -0.02).astype(np.float64)
xr_dataTBB = xr.DataArray(np.float32¥
  (dataTBB), name='Himawari-8',¥
    coords={
    'lat':('lat',lat1d,{'units':'Degrees_North'}),
    'lon':('lon',lon1d,{'units':'Degrees_East'})},
    dims=['lat','lon'])
xr_dataTBB
```

データは xr_dataTBB として xarray に変換された．このデータを PyGMT で読み込み，map 上にひまわりデータを表示してみる．

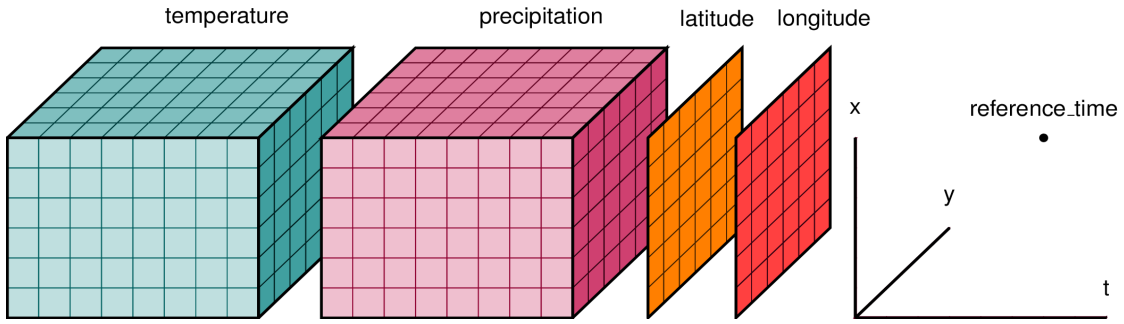
```
# PyGMT でひまわりを描く
import pygmt
fig = pygmt.Figure()
# set the frame solid line
```

```
pygmt.config(MAP_FRAME_TYPE="plain")
fig.basemap(region=[122.5, 154, 20, 48], ¥
  projection="Q15c", frame=True)
with pygmt.config(COLOR_FOREGROUND="red",¥
  COLOR_BACKGROUND="black"):
    pygmt.makecpt(
        cmap="gray", series=[190, 300, 10], ¥
  continuous=True, M=True, reverse=True,)
fig.grdimage(grid=xr_dataTBB, projection="Q15c")
fig.colorbar(frame=["a10", "x+lTBB", "y+l(K)"])
fig.coast(resolution="h", shorelines="1/1p,green")
fig.basemap(frame=["ag", "+tHimawari-8_Band13"])
fig.text(text="20150707 00:00", x=128, y=22, ¥
  font="12p,Helvetica-Bold,blue", angle=0, fill="gray")
fig.savefig("Himawari_tbb.png")
fig.show()
```


Himawari-8_Band13
125°
130°
135°
140°
145°
150°
45°
40°
35°
30°
25°
20°
20150707 00:00
(K)
190 200 210 220 230 240 250 260 270 280 290 300
TBB

2-5. xarray について

xarray はラベル付きの多次元配列である．Python にはもともと多次元配列を効率的に扱うことのできる NumPy，ラベル付き 1 次元配列（系列データ）のセットを扱う pandas があり，これら 2 つを組み合わせた xarray は pandas の多次元版と言える．



xarray の概念図　（xarray User Guide より引用 [13]）

衛星データは基本的に多次元で扱う場合が多い．たとえば衛星画像を時系列で見る時は X 方向×Y 方向×時間方向の 3 次元となる．このようなデータを扱う場合には Numpy 配列がよく使われる．ある一定時間ごとの，ある範囲の時系列変化を知りたいとする．配列を使用する場合 3 次元配列を作成し 1・2 次元目が空間情報，3 次元目が時間情報とすることもできるが，次元の順番に注意を払う必要がある．配列に地理情報は含まれないため，任意の範囲を切り出す際には，ピクセルサイズと緯度経度情報からその範囲がどのピクセルに相当するかを計算する必要がある．このように，配列を使うことは可能であるが，次元数が多い場合やプログラムが複雑な場合ではミスの原因となり得るのに対して，xarray はこのような負担を軽減することができる．上記の例の場合，データに時間情報を含めた 3 つの次元軸(dimension)を持つ 3 次元配列として格納し，各軸に対して緯度，経度，時間のラベル付けを行うことができる．地理情報と時間情報がデータと紐付けられるため，任意の時間範囲や空間範囲へのアクセスが容易である．例えば，緯度経度を指定するだけで，その範囲の任意の軸を切り出すことができる．xarray の特徴としては上記の他にも print したときにその概要を表示することや，位置インデクシング・スライシング（例：ある時間に最も近いデータを探す，など）が可能であること，またこれらの結果も xarray オブジェクトになるため軸に関する情報が保持されることなどが挙げられる．

---

[13] http://xarray.pydata.org/en/stable/user-guide/data-structures.html

2-6. PyGMT による行政区界の描画例

日本の行政区界を表すファイルは，国土交通省・国土数値情報サイトの「行政区域データ」から shp ファイルとして取得できる．ここでは千葉県の「行政区域」ファイルを用いて描画する例を紹介する．PyGMT で shp ファイルを読み込むためには，GeoPandas の to_file を使用して GMT フォーマットに変換する方法がある．GeoPandas は，python で地理空間情報の操作を容易にするライブラリである．Pandas と同様データフレーム型としてデータを読み込むが，地理情報が geometry という列に格納される．GeoPandas で読み込んだデータフレームはジオデータフレームと呼ばれる．今回使用する shp ファイルは国交省の国土数値情報データ [14]サイトからダウンロードし，マウント処理をした各自の Google Drive(/content/drive/MyDrive)の任意のフォルダに取得する．この shp ファイルには日本語の属性テーブルが含まれるため，読み込み(gpd.read_file)の際にはエンコーディングを明示的に指定する必要がある．まず，GeoPandas をインストールする．

```
# geopandas をインストールする
!pip install geopandas
```

次に，ひまわりデータをダウンロードする際にも使用した wget を使って，平成 31 年の千葉県のデータをダウンロードする．

```
import wget
shp_file = "https://nlftp.mlit.go.jp/ksj/gml/data/N03/¥
  N03-2020/N03-20200101_12_GML.zip"
wget.download(shp_file)
```

取得したデータを解凍するための zipfile をインポートし，zip ファイルを任意の Google Drive 上のフォルダに解凍する．

```
import zipfile
# 任意のドライブ・フォルダに shp ファイルを解凍する
# ここでは zip_path に以下のフォルダを指定する
zip_path = '/content/drive/MyDrive/gmt'
# 圧縮ファイルの中身をすべて解凍する
with zipfile.ZipFile(os.path.join(zip_path,¥
  'N03-20200101_12_GML.zip')) as existing_zip:
    existing_zip.extractall(zip_path)
```

[14] https://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-N03-v2_3.html#prefecture12

解凍が終了したら，フォルダ内に.dbf, prj, .shp, .shx, .xml の各拡張子を持つファイルができたことを確認する．これらのファイルは shp ファイルを構成するファイルである．たとえば，shp には図形の座標，dbf にはそれらの図形と紐付けられた番号や情報，shx には shp の図形と dbf の情報の対応関係などが保存されている. shp ファイルは最低限この 3 つが構成要素として必要であるが，今回のように投影法情報を持つ prj やメタデータが保存された xml が生成される場合もある．これらのファイルは通常単独では使用せず，shp ファイルとセットで使われるため同じディレクトリ上に置く必要がある．次に GeoPandas で shp ファイルを読み込む．

```
import os
import pygmt
import geopandas as gpd
# 読み込む shp ファイル名（千葉県の行政区界）
fnam = os.path.join(zip_path,'N03-20_12_200101.shp')
# フォルダの部分(zip_path)は各自の環境に合わせてください
# ファイル読込（encoding='SHIFT-JIS'と指定することで日本語データにも対応させる）
df_shp = gpd.read_file(fnam,encoding='SHIFT-JIS')
# 座標参照系(CRS:Coordinate Reference System)を世界測地系(WGS84)に明示的に設定する
gdf_wgs = df_shp.to_crs("EPSG:4326")
```

```
[31] gdf_wgs.head()
```

	N03_001	N03_002	N03_003	N03_004	N03_007	geometry
0	千葉県	None	千葉市	中央区	12101	POLYGON ((140.11667 35.55060, 140.11664 35.550...
1	千葉県	None	千葉市	中央区	12101	POLYGON ((140.10373 35.58333, 140.10373 35.583...
2	千葉県	None	千葉市	花見川区	12102	POLYGON ((140.11727 35.66667, 140.11727 35.666...
3	千葉県	None	千葉市	稲毛区	12103	POLYGON ((140.07380 35.64135, 140.07403 35.641...
4	千葉県	None	千葉市	若葉区	12104	POLYGON ((140.17500 35.58158, 140.17395 35.582...

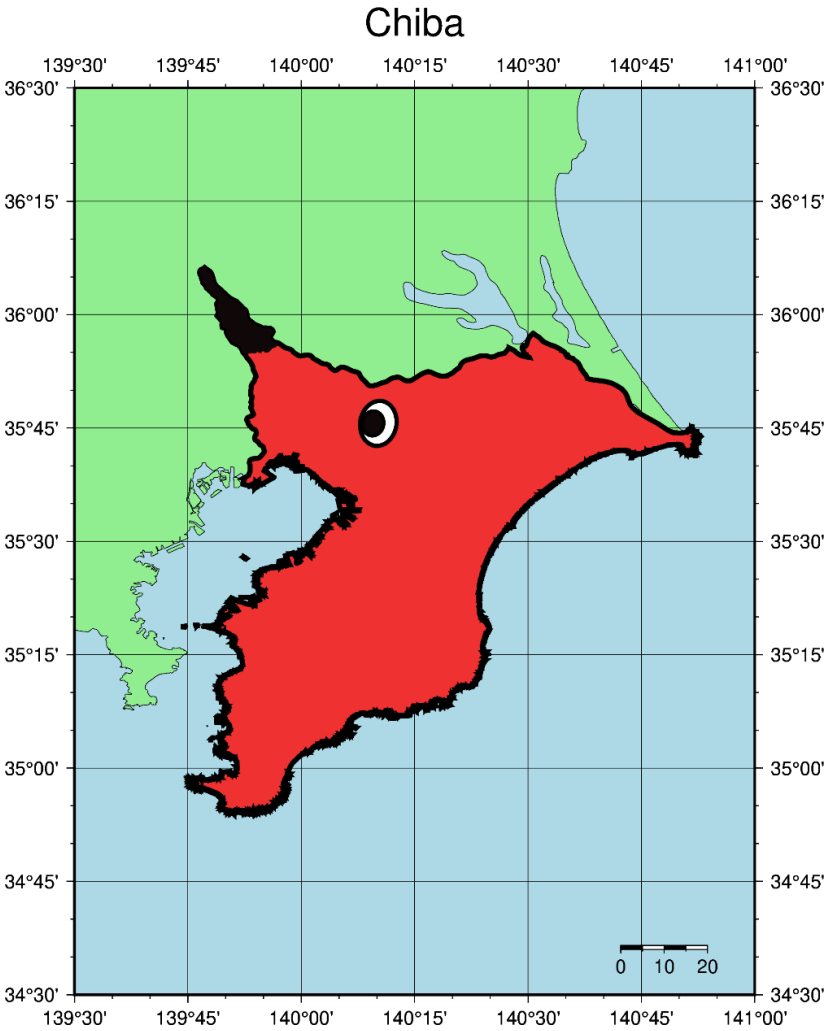
ジオデータフレームは，変数に続けて.head()と書くことで，上記のようにデータフレームの中身を一部表示させて確認することができる． ただし，データフレームの行数，列数表示にはデフォルトで制限があり，行数がその制限値を超える場合は中間の行が省略される．この制限値は，pandas の set_option を使って変更することが可能である．下記の例では，表示する行数，列数を 50 に設定している．行数を指定せず，shp ファイルに格納されているすべての行を表示するには，50 などの数値ではなく None を指定する．

```
import pandas as pd
# 表示したい列数の指定
pd.set_option('display.max_columns', 50)
# 表示したい行数の指定
pd.set_option('display.max_rows', 50)
```

必要な行列の情報が確認出来たら，表示したい行政区のポリゴンを抽出し，描画する．ここでは，千葉県と野田市の行政区界を抽出し，GMT 形式に変換する．

```
# 描画したい属性のポリゴンを選択する
# 千葉県の行政区界を抽出
chiba = gdf_wgs[df_shp['N03_001']=='千葉県']
# 使用するshpファイルは市町村単位で格納されている．
# そのままでは千葉県の外枠のみを描画できない．dissolveを使って市町村のポリゴンをまとめて，千葉県の情報（N03_001)を統合する
chibaken= chiba.dissolve(by='N03_001')
filename_chiba = "/content/drive/MyDrive/gmt/¥
  gdf_chibaken.gmt"
# GMTフォーマットに変換し，保存する
# この際，同じ名前のファイルが既にフォルダ上に存在すると，エラーとなる．ここではGMTファイルがまだ存在しない(初めて実行する)時にのみ変換を行う
if not os.path.isfile(filename_chiba):
  chibaken.to_file(filename_chiba, driver='OGR_GMT')
# 例として野田市の行政区界を抽出

noda = gdf_wgs[df_shp['N03_004']=='野田市']

filename_noda = "/content/drive/MyDrive/gmt/gdf_noda.gmt"
# GMTフォーマットに変換し，保存する
if not os.path.isfile(filename_noda):
  noda.to_file(filename_noda, driver='OGR_GMT')
# 千葉県境線を描画する
fig = pygmt.Figure()
pygmt.config(MAP_FRAME_TYPE="plain")
fig.basemap(region=[139.5, 141, 34.5, 36.5], ¥
    projection="Q15c", frame=True)
fig.coast(land="lightgreen", water="lightblue", ¥
    shorelines=True, map_scale="x13/1+c20+w20+f"))
fig.basemap(frame=["ag", "+tChiba"])
fig.plot(
    data=filename_chiba,  # GMTファイル
    pen="3p,black",  # 枠線の色と太さを指定
    color="#EF3131"   # 塗りつぶしの色を設定
)
```

```
fig.plot(
    data=filename_noda,   # GMT ファイル
    color="#100808"
)
# 任意の緯度経度地点に点（楕円）をプロットする例
# （経度，緯度，傾き，長径，短径）
data1 = [[140.17, 35.76, 80, 1.0, 0.8]]
fig.plot(data=data1, style="e", color="#FFFFFF", ¥
   pen="2.5p,black")
data2 = [[140.158, 35.76, 80, 0.55, 0.48]]
fig.plot(data=data2, style="e", color="#100808", ¥
   pen="2p,black")
fig.show()
```



Appendix-3：データ型の一覧

データ型dtype	型コード	説明
int8	i1	符号あり8ビット整数型
int16	i2	符号あり16ビット整数型
int32	i4	符号あり32ビット整数型
int64	i8	符号あり64ビット整数型
uint8	u1	符号なし8ビット整数型
uint16	u2	符号なし16ビット整数型
uint32	u4	符号なし32ビット整数型
uint64	u8	符号なし64ビット整数型
float16	f2	半精度浮動小数点型
float32	f4	単精度浮動小数点型
float64	f8	倍精度浮動小数点型
float128	f16	四倍精度浮動小数点型
complex64	c8	複素数（実部・虚部がそれぞれfloat32）
complex128	c16	複素数（実部・虚部がそれぞれfloat64）
complex256	c32	複素数（実部・虚部がそれぞれfloat128）
bool	?	ブール型（True or False）
unicode	U	Unicode文字列
object	O	Pythonオブジェクト型

＊データ型名の末尾の数字は bit で表し，型コード末尾の数字は byte で表す．同じ型でも値が違うので注意．

＊整数 int, uint や浮動小数点数 float の各データ型の取り得る値の範囲は np.iinfo(), np.finfo()で確認できる。

Appendix-4：Python におけるインデントの意味

プログラムにおいて，for 文や if 文などを書く時，どの範囲まで for や if が掛かっているのかをわかりやすくするため，インデント(字下げ)をすることがある．C や Fortran，Shell においてはインデントをしなくてもプログラムは問題なく実行できる．しかし，Python においては for や if，def などの関数において，ブロック(関数がかかる範囲)を定義するためにインデントを用いて管理しており，インデントがずれるとエラーになるため注意が必要である．例えば以下のコードはエラーになる．

```
a=2
 b=3
a+b
```

※b の左側の半角スペースがエラーの原因

```
def abc():
print("detect")
```

※print の左側にインデントが無いことがエラーの原因

for や if において，指定するブロック内のコードの前に適切なインデントが必要である．インデントのスペースは特に指定は無い（タブでも可）．例えば以下のコードの場合，

```
for i in range(1,3):
   for j in range(1,3):
      for k in range(1,3):
         print("A")   #for k ブロック内
      print("B")   #for j ブロック内
   print("C")    #for i ブロック内
print("D")   #どのブロック内にも属さない
```

AABAABCAABAABCD （実際には結果は縦並びに表示されるが，テキストの都合上，横書きに記している）

for i のインデントは半角スペース３つ，for j は２つ，for k は１つに指定している．for i のブロックに属すコードを書く場合は半角スペース３つ，for j の場合は 3+2 で５つ，for k の場合は 3+2+1 で６つ分インデントを設ける必要がある．

この場合，まず”A”が二回出力され，そのあとに”B”，”AAB”が二回続いた後に”C”が出力，”AABAABC”が二回出力されたあとに最後に”D”が出力される．

Appendix-5：NDVI の算出

ここまで，ひまわりデータを使って，台風やトンガ火山の噴煙といった大気に伴った現象を見てきたが，最後に地表面に着目して植生を可視化する方法を紹介する．植生域をより明瞭に区別するためには，植生指数(Vegetation Index)を計算する方法がある．植生指数は多々提唱されているが，最も一般的に使用されるのは，赤バンド(Red)と近赤外バンド(Near Infrared)を使用した正規化植生指数(NDVI：Normalized Difference Vegetation Index)である．NDVI は以下の式で表される．

$$\mathrm{NDVI} = \frac{\mathrm{NIR\ Band} - \mathrm{Red\ Band}}{\mathrm{NIR\ Band} + \mathrm{Red\ Band}}$$

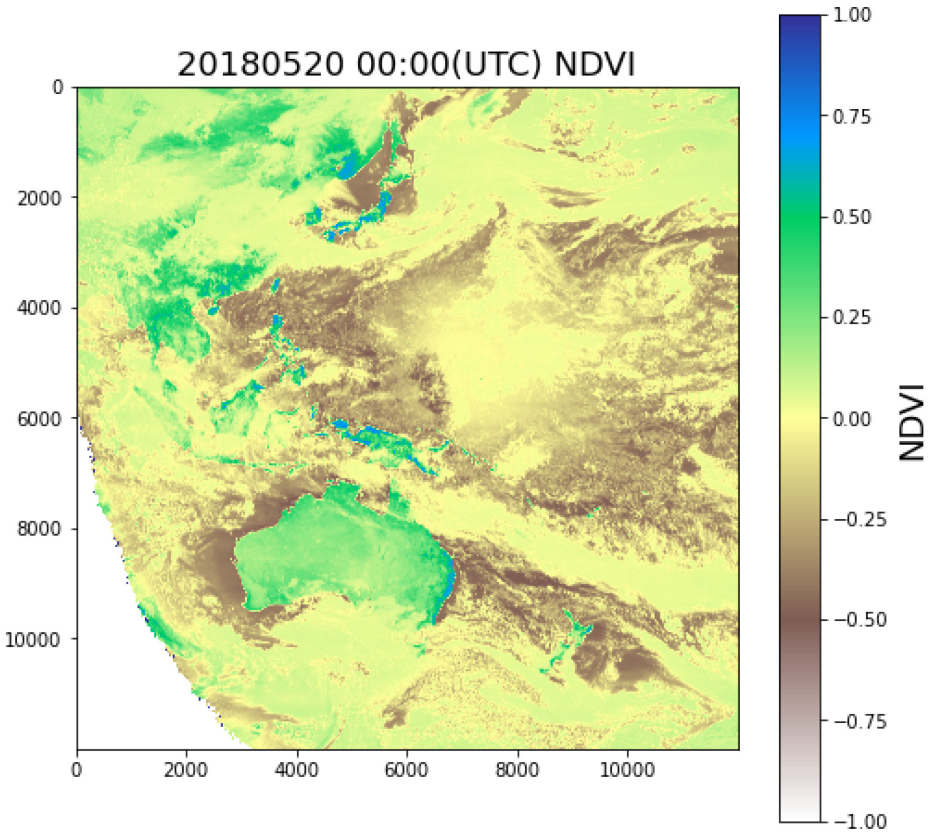
数値は-1～1 の範囲で表され，NDVI の値が大きい（小さい）ほど植生が多い（少ない）． 以下のコードで 2018 年 5 月 20 日の事例を基に NDVI の分布の可視化を紹介する．

```
!pip install wget
import wget
import os
import bz2
import tarfile
import numpy as np
import matplotlib.pyplot as plt

# CEReS Himawari-8 データの読み込み

FTP="ftp://hmwr829gr.cr.chiba¥
  -u.ac.jp/gridded/FD/V20190123/"
B03_tt =
FTP+"{year:04}{month:02}/{band}/".format(year=2018,¥
  month=5,band='EXT')
B04_tt =
FTP+"{year:04}{month:02}/{band}/".format(year=2018,¥
  month=5,band='VIS')

B03_fnam = '201805200000.ext.01.fld.geoss.bz2'
```

```
B04_fnam = '201805200000.vis.03.fld.geoss.bz2'
#ファイル名 (Band03/visible red, Band 04/near infrared)
B03 = '201805200000.ext.01.fld.geoss'
B04 = '201805200000.vis.03.fld.geoss'

B03_ttt = B03_tt+B03_fnam
B04_ttt = B04_tt+B04_fnam

# ダウンロード Band3, 4
wget.download(B03_ttt)
wget.download(B04_ttt)

# ダウンロード Lookup table
wget.download('http://www.cr.chiba¥
  -u.jp/databases/GEO/H8_9/FD/count2tbb_v102.tgz')

# tar.gz ファイルの解凍
with tarfile.open('count2tbb_v102.tgz', 'r:gz') as t:
    t.extractall()

# bz2 ファイル(AHI)の解凍
B03_zipfile = bz2.BZ2File(B03_fnam)
B03_data = B03_zipfile.read()
open(B03, "wb").write(B03_data)
os.remove(B03_fnam)

# Band3 の読み込み
with open( B03,'rb') as fp:
    H8_Band03 = ¥
  np.fromstring(fp.read(),dtype='>u2').reshape¥
  (24000, 24000)
os.remove(B03)
B04_zipfile = bz2.BZ2File(B04_fnam)
B04_data = B04_zipfile.read()
open(B04, "wb").write(B04_data)
os.remove(B04_fnam)
```

```
# Band4 の読み込み
with open( B04,'rb') as fp:
    H8_Band04 = ¥
  np.fromstring(fp.read(),dtype='>u2').reshape¥
  (12000, 12000)
os.remove(B04)

#ファイル名の定義
B03 = '201805200000.ext.01.fld.geoss'
B04 = '201805200000.vis.03.fld.geoss'

# LUT のデータパスの定義
LUT_Band03_fnam = 'count2tbb_v102/ext.01'
LUT_Band04_fnam = 'count2tbb_v102/vis.02'


# LUT の読み込み
LUT_Band03 = np.loadtxt(LUT_Band03_fnam, usecols=(1,))
LUT_Band04 = np.loadtxt(LUT_Band04_fnam, usecols=(1,))

# DN 値から反射率への変換(Band3)
H8_Band03_Ref = LUT_Band03[H8_Band03]

# リサンプルのための平均値算出
# Band3: 500m ==> 1km
H8_Band03_Ref = ¥
  H8_Band03_Ref.reshape(12000,2,12000,2).mean(-1).mean(1)

H8_Band03_Ref = ¥
  LUT_Band03[H8_Band03].reshape(12000,2,12000,2).mean¥
  (axis=(1,3))

# DN 値から反射率への変換(Band4)
H8_Band04_Ref = LUT_Band04[H8_Band04]
```

```
# 不要な変数の削除(メモリ使用量を減らすため)とNDVI計算式の分母分子の計算
del LUT_Band03, LUT_Band04
minus = H8_Band04_Ref - H8_Band03_Ref
del H8_Band04_Ref
plus = minus + H8_Band03_Ref*2

# NDVIの計算
NDVI = minus / plus

# 図の作成
fig = plt.figure(figsize=(8,8))
plt.imshow(NDVI,vmin = -1, vmax = 1, origin='upper',¥
  cmap = 'terrain_r')
plt.title('20180520 00:00(UTC) NDVI', fontsize=18)
plt.colorbar().set_label(label='NDVI', fontsize=18)
```



謝辞

本演習のテキスト資料作成に協力して頂いた鈴木遼太郎氏，大槻真由氏，本橋優登氏，祖父江侑紀氏に感謝申し上げます．また，実習をお手伝い頂いた李夢禹氏，劉治彦氏，李偉氏，白文宣氏，森貴之氏に心より御礼申し上げます．
本演習は独立行政法人日本学術振興会「研究拠点形成事業」（課題番号：JPJSCCA20220008）の助成を得て行われました．

参考文献

- 長谷川隆司，上田文夫，柿本太三．気象衛星画像の見方と使い方．オーム社，2006, 208p
- 新田尚．最新天気予報の技術．東京堂出版, 2011, 497p
- 伊東譲司，西村修司．ひまわり８号 気象衛星講座．東京堂出版, 2016, 272p
- 小倉義光．一般気象学 第２版補訂版．東京大学出版会, 2016, 320p
- 気象庁予報部（2016）「第６章ひまわり８号 RGB 合成画像の基礎」（量的予報技術資料），[online]<http://www.jma.go.jp/jma/kishou/books/yohkens/21/chapter6.pdf> May. 27, 2022 accessed
- 市井和仁，楊偉，Python を利用したデータ解析入門，2020，[online]<http://ichiilab.weebly.com/uploads/1/0/9/1/109128265/python_obsdataanalysis_2020_jpn_open.pdf> May. 27, 2022 accessed
- 大重美幸，詳細! Python 3 入門ノート，ソーテック社，2017，416p
- 喜多一,プログラミング演習 Python 2019,[online]<https://repository.kulib.kyoto-u.ac.jp/dspace/bitstream/2433/245698/1/Version2020_02_13_01.pdf> May. 27, 2022 accessed
- チーム・カルポ，Matplotlib&Seaborn 実装ハンドブック，秀和システム，2018，319p
- Choosing Colormaps in Matplotlib Version 3.3.4，[online] <https://matplotlib.org/stable/tutorials/colors/colormaps.html> May. 27, 2022 accessed