

第10回VL講習会@千葉大・CEReS

～ 日本の地球観測衛星を知ろう ～

2016年9月20 - 21日

PartA テキスト

ひまわり8号衛星データの画像化

千葉大学環境リモートセンシング研究センター

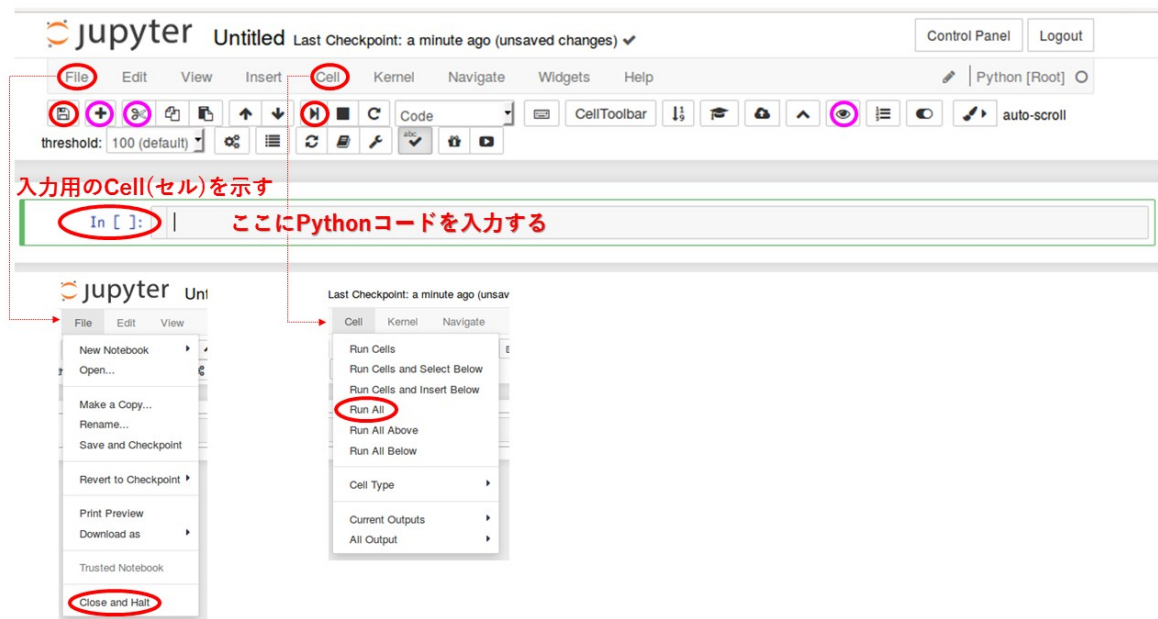
担当： 眞子 直弘，豊嶋 紘一，岡本 浩

1 Jupyter Notebookの簡単なイントロダクション

Jupyter Notebookとは・・・

Ipython Notebookと呼ばれるPythonのWebインターフェイスが進化したもので、Pythonに加えてRubyやBashといった様々なスクリプト言語に対応したWebベースの解析ツールである。






1.1 Jupyter Notebookのインターフェイス




◎ メニューバーの主なメニュー

File > Close and Halt	ノートブックを閉じる
Cell > Run All	全てのCellを実行する

◎ ツールバーの主なアイコン

	Save and Checkpoint	ノートブックを保存
	Insert cell below	Cellを追加
	Cut selected cells	Cellを削除
	run cell, select below	Cellを実行
	Hide codecell inputs	Cellを隠す (Code Cellのみ)

※ブラウザでページの再読み込みを行うと、最後に保存したノートブックが読み込まれる。

※CellにはPythonコードを入力するCode Cellの他、HTMLスクリプトを記述するMarkdown Cell等がある。Markdown CellをダブルクリックするとHTMLスクリプトが表示される。その場合、Shift+Enterキーを押すか、ツールバーのrunボタン()を押すと元の表示に戻る。

1.2 Jupyter Notebookを電卓として使う

☞ 下の入力欄(Cell=セルと呼ばれる)に適当な数式を入力してみよう！
入力後にキーボードの**Shiftキー**と**Enterキー**を同時に押すか、
ツールバーの**runアイコン**("run cell, select below")をクリックすると
入力した式が評価される。

入力例:

```
1+1
```

```
In [1]: 1+1
```

```
Out[1]: 2
```

1.3 変数を使う

☞ 下のCellで変数を使った計算を試してみよう！

入力例:

```
x = 10  
x*2
```

```
In [2]: x = 10  
x*2
```

```
Out[2]: 20
```

1.4 科学計算モジュール(Numpy)を使う

☞ Pythonで複雑な科学計算を行うには、Numpyというモジュールを使うと良い。そのための準備として以下の記述が必要。

入力コード:

```
import numpy as np
```

```
In [3]: import numpy as np
```

☞ 次に、Numpyを使って適当なデータを用意する。
以下の例では、xには[0.0,3.14]の範囲を100分割した値、yにはsin(x)の値が入る。

入力例:

```
x = np.linspace(0.0, 3.14, 100)  
y = np.sin(x)
```

```
In [4]: x = np.linspace(0.0, 3.14, 100)  
y = np.sin(x)
```

☞ 変数の中身を確認してみよう！

入力例:

```
x
```

【参考】このように、変数の中身を知りたい場合は、変数名を入力してShift+Enterキーを押すと表示される。

In [5]:

```
x
```

```
Out[5]: array([[ 0.          ,  0.03171717,  0.06343434,  0.09515152,  0.12686869,
  0.15858586,  0.19030303,  0.2220202 ,  0.25373737,  0.28545455,
  0.31717172,  0.34888889,  0.38060606,  0.41232323,  0.4440404 ,
  0.47575758,  0.50747475,  0.53919192,  0.57090909,  0.60262626,
  0.63434343,  0.66606061,  0.69777778,  0.72949495,  0.76121212,
  0.79292929,  0.82464646,  0.85636364,  0.88808081,  0.91979798,
  0.95151515,  0.98323232,  1.01494949,  1.04666667,  1.07838384,
  1.11010101,  1.14181818,  1.17353535,  1.20525253,  1.2369697 ,
  1.26868687,  1.30040404,  1.33212121,  1.36383838,  1.39555556,
  1.42727273,  1.4589899 ,  1.49070707,  1.52242424,  1.55414141,
  1.58585859,  1.61757576,  1.64929293,  1.6810101 ,  1.71272727,
  1.74444444,  1.77616162,  1.80787879,  1.83959596,  1.87131313,
  1.9030303 ,  1.93474747,  1.96646465,  1.99818182,  2.02989899,
  2.06161616,  2.09333333,  2.12505051,  2.15676768,  2.18848485,
  2.22020202,  2.25191919,  2.28363636,  2.31535354,  2.34707071,
  2.37878788,  2.41050505,  2.44222222,  2.47393939,  2.50565657,
  2.53737374,  2.56909091,  2.60080808,  2.63252525,  2.66424242,
  2.6959596 ,  2.72767677,  2.75939394,  2.79111111,  2.82282828,
  2.85454545,  2.88626263,  2.9179798 ,  2.94969697,  2.98141414,
  3.01313131,  3.04484848,  3.07656566,  3.10828283,  3.14       ]])
```

1.5 グラフを描く

☞ Jupyter Notebookで簡単なグラフを描いてみよう！

まず始めの準備として描画の前に次の一行を書いておく。(これで図がWebページに埋め込まれるようになる。)

入コード:

```
%matplotlib inline
```

In [6]:

```
%matplotlib inline
```

☞ また、Pythonの描画モジュール(Matplotlib)を使うために以下の記述が必要。

入コード:

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

In [7]:

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

☞ では、先ほど用意したデータをプロットしてみよう！

Matplotlibのplotというメソッドを使ってみる。

入コード:

```
plt.plot(x, y)
```

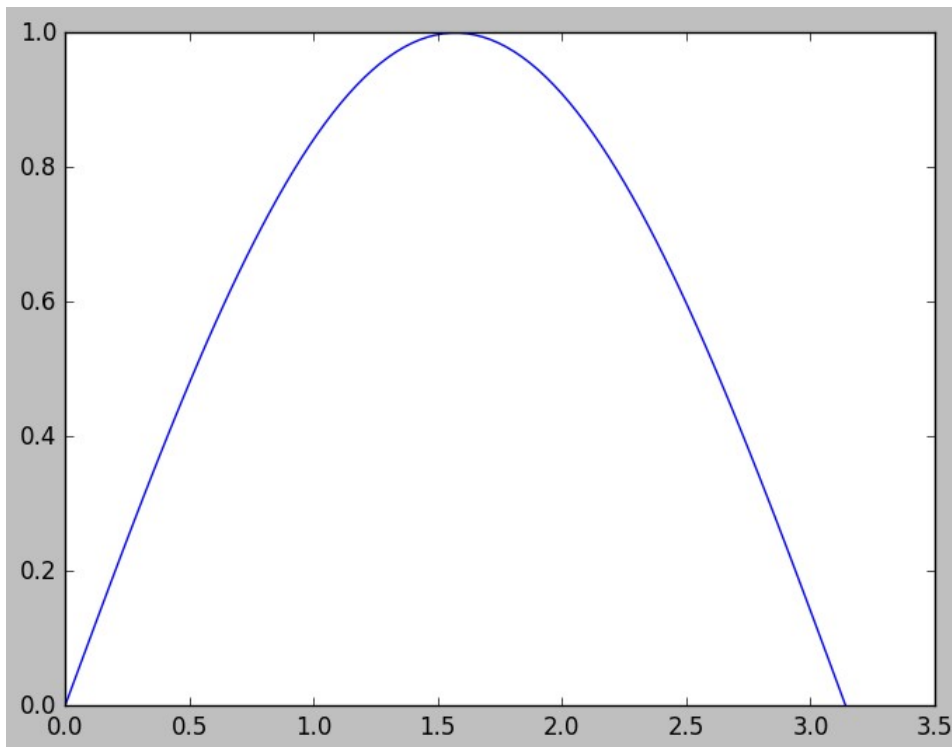
【参考】描画コマンドの後ろにセミコロン(;)を付けると
[<matplotlib.lines.Line2D at 0x7fa83f4282b0>]

のようなmatplotlibのメッセージを非表示にできる。

例: plt.plot(x, y);

```
In [8]: plt.plot(x, y)
```

```
Out[8]: <matplotlib.lines.Line2D at 0x7fa83f4282b0>
```



1.6 その他の機能

🔗 **Jupyterのヘルプ機能**を使ってみよう!

コマンドの後ろに**クエスチョンマーク(?)**を付けるとヘルプ画面が表示される。ヘルプ画面は右上の×印をクリックして閉じることができる。

入力例:

```
np.linspace?
```

```
In [9]: np.linspace?
```

🔗 **Jupyterの補完機能**を使ってみよう!

コマンドを途中まで入力して**Tabキー**を押すと、補完候補が表示される。矢印キーで選択してEnterキーを押すと入力できる。

入力例:

```
np.lin[Tab]
```

```
In [10]: np.linspace
```

```
Out[10]: <function numpy.core.function_base.linspace>
```

🔗 **シェルコマンド**を使ってみよう!

Code Cell内では先頭に**ビックリマーク(!)**を付けてシェルコマンドを実行することができる。なお、ls、catのように良く使われるシェルコマンドの中には直接実行できるものもある。

入力例:

```
!echo hello
```

```
In [11]: !echo hello
```

```
hello
```

👉 スクロールバーを使ってみよう！

出力の行数が多いときはCellの左側をクリックするとスクロールバーが表示される。
Cellの左側をダブルクリックすると出力を隠すことができる。
もう一度クリックすると元の表示に戻る。

入力例:

```
cat vis.01
```

In [12]: cat vis.01

```
0 -1.176471
1 -1.117647
2 -1.058824
3 -1.000000
4 -0.941176
5 -0.882353
6 -0.823529
7 -0.764706
8 -0.705882
9 -0.647059
10 -0.588235
11 -0.529412
12 -0.470588
13 -0.411765
14 -0.352941
15 -0.294118
16 -0.235294
17 -0.176471
18 -0.117647
19 -0.058824
```

2 ひまわり8号グリッドデータの描画

千葉大学ではひまわり8号全球スキャンの**グリッドデータ(緯度経度直交座標系精密幾何補正済データ)**を公開している。

ひまわり8号データ処理の手始めとして、まずは扱いやすいグリッドデータを見てみよう！

なお、グリッドデータの詳細は以下のページで説明されている。

[ひまわり8号グリッドデータ公開ページ \(http://www.cr.chiba-u.jp/databases/GEO/H8_9/FD/index_jp.html\)](http://www.cr.chiba-u.jp/databases/GEO/H8_9/FD/index_jp.html)

2.1 ひまわり8号グリッドデータについて

ひまわり8号のデータとは・・・

簡単に言うと、気象衛星ひまわり8号からデジタルカメラで撮影した**地球のカラー写真**。ただし、JPEGのような一般的な画像フォーマットになっていないため、画像として表示するためには**データ処理が必要**(このページのメインテーマ)。

ひまわり8号の「カメラ」はAHI(Advanced Himawari Imager)と呼ばれている。普通のカラーカメラは可視領域の3色(赤緑青)しか記録しないのに対し、AHIは可視～赤外領域の16色(16バンド)を記録することができる。

グリッドデータとは・・・

ひまわり8号のデータを等緯度経度グリッド上に並べて使いやすくしたデータ。

(それに対し、ひまわり8号の標準データはAHIの画像座標上に並んでおり、緯度経度を得るためには幾何補正が必要。)

グリッドデータにはこれまでの静止気象衛星データとの関連性から気象庁とは異なるルールでバンド名が付けられている。表2.1にひまわり8号のバンド名とグリッドデータのバンド名の対応を示す。(例えばひまわり8号のバンド03はグリッドデータのext01に対応する。)

グリッドデータのファイル名はyyyyymmddhhnn.XXX.ZZ.cccc.geossのようにになっている。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、ccccは観測領域名。

【例】201605120400.ext.01.fld.geoss

全球スキャン(フルディスク:fld)グリッドデータの観測範囲は東経85°～205°、北緯60°～南緯60°(120°×120°)であり、各バンドの空間分解能に合わせたグリッド(格子)に区切られている。

ただし、これではデータサイズが大き過ぎるため、本演習では**東経138°～141°、北緯34.5°～37.5°**

(3°×3°)の関東周辺(kanto)を切り出したデータを使用する(図2.1参照)。

グリッドデータには等緯度経度グリッド上のひまわり8号観測値(DN値)が西から東、北から南の順に**2バイト符号なし整数**のバイナリ形式で記録されている(バイトオーダーはビッグエンディアン)。

詳細を非表示

◎ 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, ... のようなカウント値

◎ 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のように多バイトから成るデータをメモリに格納する順番のこと。上位バイトから順に格納する方式をビッグエンディアン、下位バイトから格納する方式をリトルエンディアンという。Intelのプロセッサはリトルエンディアンを採用している。

表2.1. ひまわり8号のバンド名とグリッドデータのバンド名

ひまわり8号バンド名	グリッドデータバンド名	中心波長(μm)	グリッド間隔(度※)	有効ビット数
01	vis	01	0.47	11
02		02	0.51	11
03	ext	01	0.64	11
04	vis	03	0.86	11
05	sir	01	1.6	11
06		02	2.3	11
07	tir	05	3.9	14
08		06	6.2	11
09		07	6.9	11
10		08	7.3	12
11		09	8.6	12
12		10	9.6	12
13		01	10.4	12
14		02	11.2	12
15		03	12.4	12
16		04	13.3	12

※ グリッド間隔0.01度は空間分解能約1 kmに相当する。

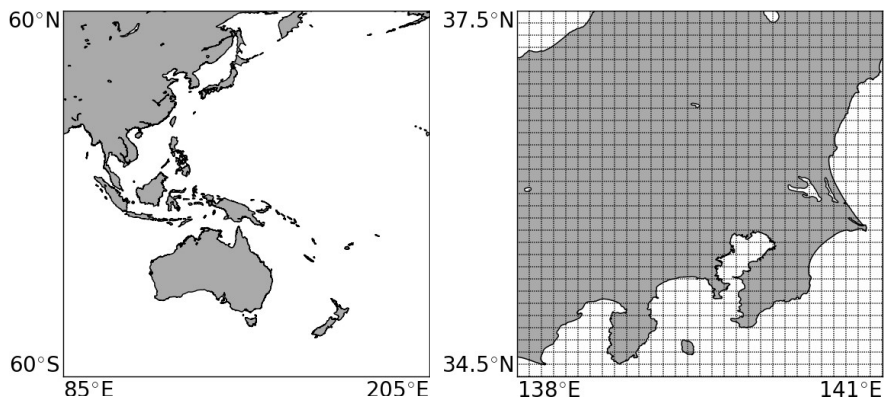


図2.1 (左) フルディスクグリッドデータの観測領域、(右) グリッドデータから切り出した関東周辺領域
(図のグリッド間隔は0.1度、実際のグリッドデータのグリッド間隔は表2.1の通り)

2.2 準備

それでは、グリッドデータの描画をする準備をしよう！

👉 まず、描画の準備

入コード:

```
%matplotlib inline
```

In [1]: `%matplotlib inline`

👉 次に、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。

入コード:

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

In [2]: `import numpy as np`
`import matplotlib.pyplot as plt`
`import matplotlib.cm as cm`

2.3 グリッドデータの読み込み

ここでは例として青バンド(vis.01)のグリッドデータ("201605120400.vis.01.kanto.geoss")を使う。
☞ このデータは以下のようなコードで読み込むことができる。

入カコード:

```
data = np.fromfile('201605120400.vis.01.kanto.geoss', dtype='>u2')
```

詳細を非表示

◎ ここではdtypeオプションでビッグエンディアン2バイト符号なし整数を指定している。ファイルを全部読み込む場合、データサイズを指定する必要はない。ただし、もしファイルサイズが2バイトの整数倍でない場合はエラーとなる。

◎ フォーマット文字の'u'は符号なし整数を意味し、後ろの2はバイト数を意味している。'u'以外にも'i'、'f'等のフォーマット文字があり、それぞれ符号付き整数、浮動小数点数を意味する。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を指定できる。

```
In [3]: data = np.fromfile('201605120400.vis.01.kanto.geoss', dtype='>u2')
```

☞ どの変数が読み込まれたか確かめてみよう！

入力例:

```
data
```

```
In [4]: data
```

```
Out[4]: array([204, 204, 200, ..., 199, 198, 197], dtype=uint16)
```

☞ データサイズを確かめてみよう！

入カコード:

```
data.shape
```

```
In [5]: data.shape
```

```
Out[5]: (90000,)
```

この例では、vis.01バンドのグリッド間隔が0.01°であり、切り出したデータの範囲が3°×3°なので、画像サイズは300×300になっている。

☞ 読み込んだデータに**reshape**(全要素数を保ったまま配列の次元数や各次元の要素数を変更すること)を行って2次元配列に変換しよう！

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

 1次元データ: shape=(9,) **reshape** ⇒

0	1	2
3	4	5
6	7	8

 2次元データ: shape=(3, 3)

入カコード:

```
data = data.reshape(300, 300)
```

```
In [6]: data = data.reshape(300, 300)
```

☞ reshapeの効果を確かめるために、もう一度データサイズを確かめてみよう！

入カコード:

```
data.shape
```

```
In [7]: data.shape
```

```
Out[7]: (300, 300)
```

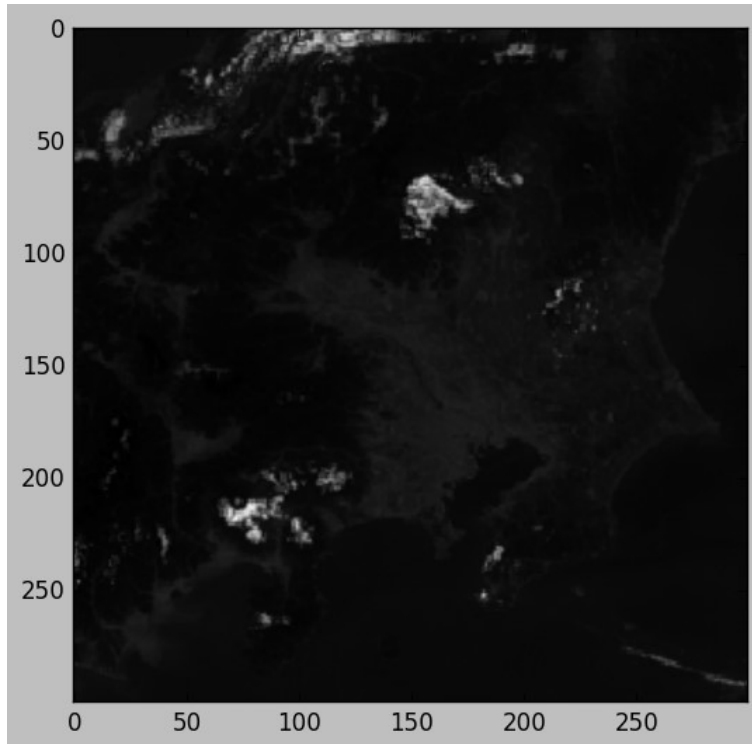
2.4 描画

☞ 等間隔グリッドの2次元データはimshowというメソッドを使って画像表示することができる。

入コード:

```
plt.imshow(data);
```

In [8]: plt.imshow(data);



一応、ひまわり8号のグリッドデータを画像として表示できたが・・・ imshowのデフォルト設定では

- ・ 画像が暗い
 - ・ 画像の列番号、行番号がそれぞれ横軸、縦軸になる
- ↓
- ・ カラースケールの下限、上限はそれぞれ vmin、vmax オプションで与えられる。
 - ・ 横軸、縦軸の値は extent オプションで左、右、下、上の座標を与えられる。

imshowの主なオプションを表2.2に挙げる。

表2.2 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: 100
vmax	カラースケールの最大値	例: 500
extent	画像の左右下上座標	例: (138,141,34.5,37.5)
cmap	カラーマップ	cm. jet cm. gray etc.
interpolation	補間方法	'none' 'nearest' 'bilinear' etc.
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)

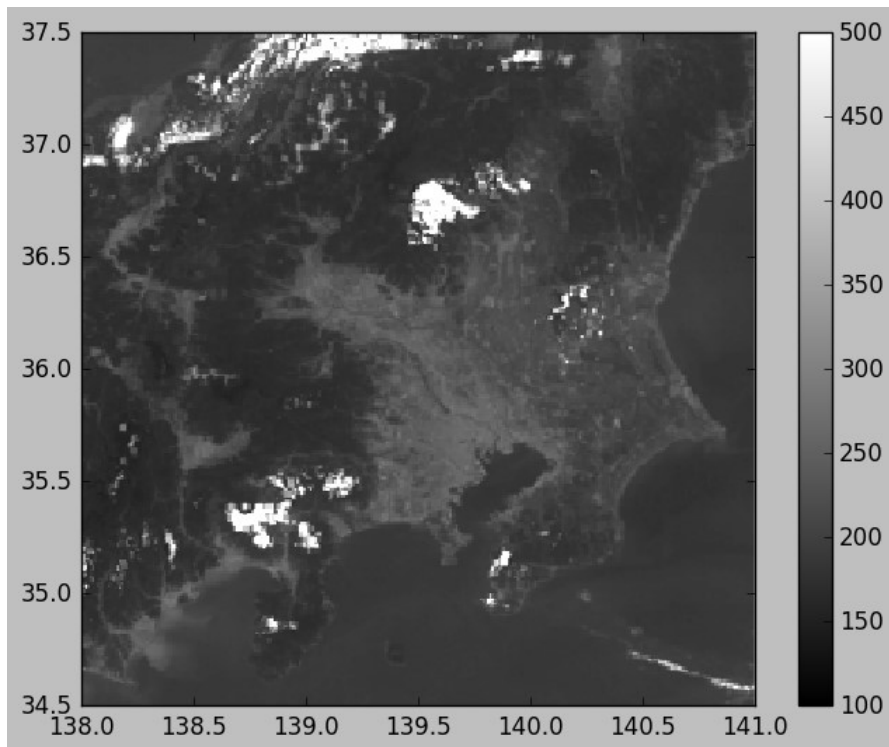
その他、詳細についてはplt.imshow?で表示される ヘルプ画面参照。

👁️ imshowのオプションを使ってみよう！

入力例:

```
plt.imshow(data, vmin=100, vmax=500, extent=(138, 141, 34.5, 37.5), interpolation='none')
plt.colorbar():
```

```
In [9]: plt.imshow(data, vmin=100, vmax=500, extent=(138, 141, 34.5, 37.5), interpolation='none')
plt.colorbar():
```



カラーバーはDN値であることに注意！

3 グリッドデータの校正

グリッドデータに格納されているのはひまわり8号のセンサーから得られた生データ(DN値)であり、物理量(反射率または輝度温度)を得るためには**校正**が必要である。

3.1 準備

☞ まず、いつもの準備を行う。

入コード:

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

```
In [1]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

3.2 グリッドデータの読み込み

☞ 前章と同様に、青バンド(vis.01)のグリッドデータを読み込んでみよう！ここではreshapeまで一気にやってみる。

入コード:

```
data = np.fromfile('201605120400.vis.01.kanto.geoss', dtype='>u2').reshape(300, 300)
```

```
In [2]: data = np.fromfile('201605120400.vis.01.kanto.geoss', dtype='>u2').reshape(300, 300)
```

3.3 ルックアップテーブルの読み込み

[千葉大学のグリッドデータ公開サイト \(http://www.cr.chiba-u.jp/databases/GEO/H8_9/FD/index_jp.html\)](http://www.cr.chiba-u.jp/databases/GEO/H8_9/FD/index_jp.html)では、グリッドデータの取得から読み込み、校正、描画まで全自動で行ってくれる 便利なスクリプトを含むサンプルプログラム(Fortran、C言語)のセット([count2tbb_v101.tgz](http://www.cr.chiba-u.jp/databases/GEO/H8_9/FD/count2tbb_v101.tgz) (http://www.cr.chiba-u.jp/databases/GEO/H8_9/FD/count2tbb_v101.tgz))も提供されている。

サンプルプログラムセットには **DN値を物理量に変換するためのルックアップテーブル**が含まれており、ここではこれを使って校正を行う。

ルックアップテーブルのファイル名はグリッドデータのバンド名と同じであり、内容にはDN値と物理量(バンド1~6: 反射率、単位は%、バンド7~16: 輝度温度、単位はK)が書かれている。

ルックアップテーブル(vis.01)の内容

0列目(DN値)	1列目(反射率)
0	-1.176471
1	-1.117647
2	-1.058824
.	.
.	.
2047	119.235294

👉 ルックアップテーブルは以下のようなコードで読み込むことができる。

入コード:

```
count, value = np.loadtxt('vis.01', unpack=True)
```

詳細を非表示

◎ loadtxtにunpack=Trueのオプションを付けて0列目、1列目をそれぞれcount、valueに代入している。

◎ このオプションを付けないとloadtxtは1つの2次元ndarrayを返す。

In [3]:

```
count, value = np.loadtxt('vis.01', unpack=True)
```

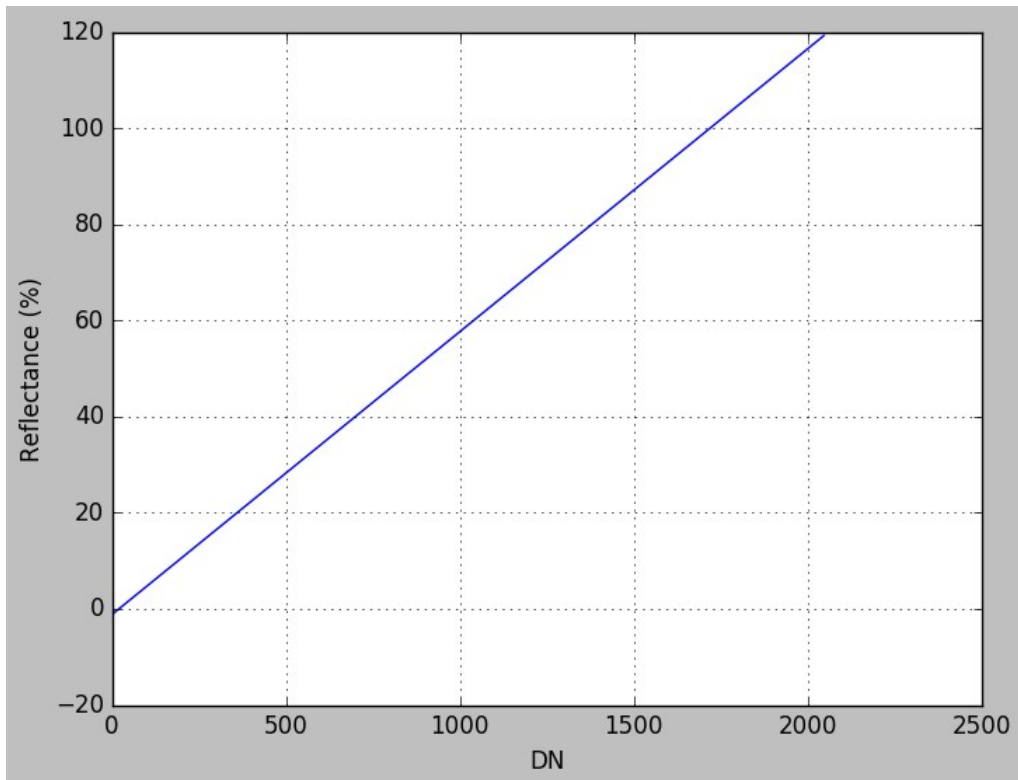
👉 ルックアップテーブルの内容をグラフで確かめてみよう！

入コード:

```
plt.plot(count, value)
plt.xlabel('DN')
plt.ylabel('Reflectance (%)')
plt.grid(True)
```

In [4]:

```
plt.plot(count, value)
plt.xlabel('DN')
plt.ylabel('Reflectance (%)')
plt.grid(True)
```



3.4 校正

ルックアップテーブルを使うとDN値(count)に対応する物理量(value)が直ちに分かる。

今の場合、配列valueのインデックス(0から始まる要素番号)は対応するcountの値に等しい。

Numpyの配列(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、valueのインデックスを指定することによってDN値に対応する物理量を得ることができる。

◎ 例えば、value[0]でvalueの最初の要素を取り出せる。

◎ value[[0, 1, 2]]のように要素のインデックスを入れた配列を使って複数の要素も取り出せる。

☞ DN値が0、500、1000、1500、2000に対応する物理量を求めてみよう！

入コード:

```
value[[0, 500, 1500, 2000]]
```

```
In [5]: value[[0, 500, 1500, 2000]]
```

```
Out[5]: array([-1.176471,  28.235294,  87.058824, 116.470588])
```

☞ 同様に、dataの中身はDN値なので、valueのインデックスに指定すれば物理量に変換できる。

入コード:

```
value_b = value[data]
```

```
In [6]: value_b = value[data]
```

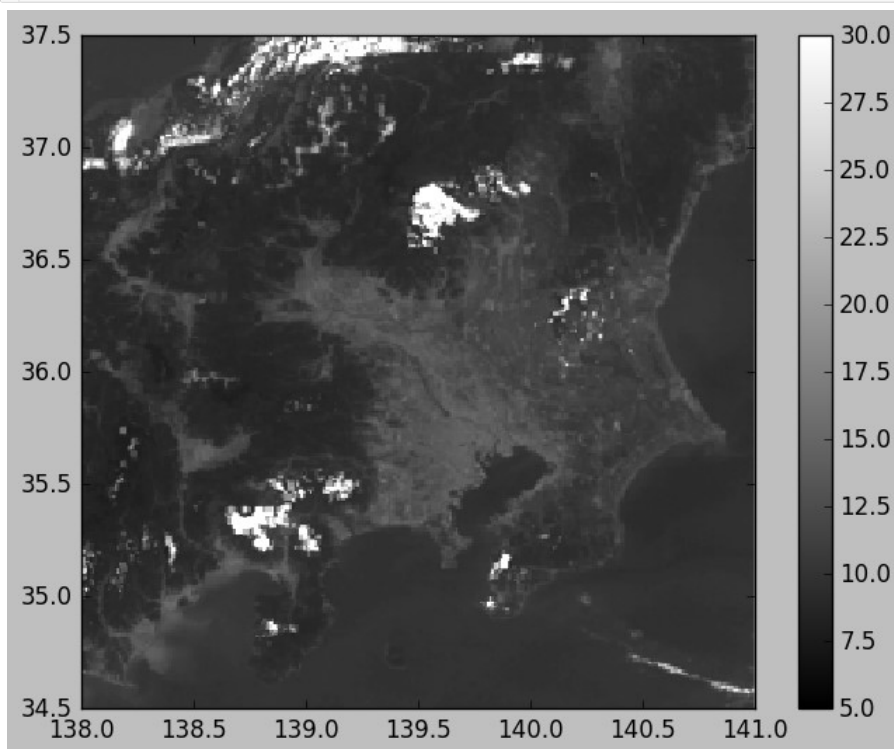
3.5 描画

☞ 校正済みデータをプロットしてみよう！

入コード:

```
plt.imshow(value_b, vmin=5, vmax=30, extent=(138, 141, 34.5, 37.5), interpolation='none')  
plt.colorbar();
```

```
In [7]: plt.imshow(value_b, vmin=5, vmax=30, extent=(138, 141, 34.5, 37.5), interpolation='none')  
plt.colorbar();
```



DN値が物理量に変わった！

4 グリッドデータのRGB合成

ひまわり8号の観測バンドには赤、緑、青のバンドが含まれており、これらのデータを合成することでカラー写真のように見た目に近い画像が得られる。

4.1 RGB合成について

RGB合成とは・・・

赤、緑、青の3色を混ぜ合わせて様々な色を作り出すこと。
下のスライダーを操作して実際にRGB合成をしてみよう！

× R G B

RGB合成結果:



4.2 準備

👉 まず、いつもの準備を行う。

入コード:

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

```
In [3]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

4.3 グリッドデータの読み込み

👉 前章と同様に、青、緑、赤バンド(vis.01、vis.02、ext.01)のグリッドデータを読み込んでみよう！
ここでは校正まで一気にいき、さらに0.01を掛けることで反射率の百分率を割合に変換する。
赤バンド(ext.01)だけ空間分解能が異なることに注意！

入コード:

```
fnam_b = '201605120400.vis.01.kanto.geoss'
fnam_g = '201605120400.vis.02.kanto.geoss'
fnam_r = '201605120400.ext.01.kanto.geoss'
value_b = np.loadtxt('vis.01', usecols=(1,)) [np.fromfile(fnam_b, dtype='>u2').reshape(300, 300)]*0.01
value_g = np.loadtxt('vis.02', usecols=(1,)) [np.fromfile(fnam_g, dtype='>u2').reshape(300, 300)]*0.01
value_r = np.loadtxt('ext.01', usecols=(1,)) [np.fromfile(fnam_r, dtype='>u2').reshape(600, 600)]*0.01
```

詳細を非表示

loadtxtにusecols=(1,)オプションを付けて1列目だけを読み込んでいる。
usecolsオプションの値にはタプル(Pythonの配列の一種)を与えるが、
要素数が1のタプルは要素の後にコンマを付けるという規則がある。

```
In [4]: fnam_b = '201605120400.vis.01.kanto.geoss'
fnam_g = '201605120400.vis.02.kanto.geoss'
fnam_r = '201605120400.ext.01.kanto.geoss'
value_b = np.loadtxt('vis.01', usecols=(1,)) [np.fromfile(fnam_b, dtype='>u2').reshape(300, 300)]*0.01
value_g = np.loadtxt('vis.02', usecols=(1,)) [np.fromfile(fnam_g, dtype='>u2').reshape(300, 300)]*0.01
value_r = np.loadtxt('ext.01', usecols=(1,)) [np.fromfile(fnam_r, dtype='>u2').reshape(600, 600)]*0.01
```



```
In [7]: rgb.shape
```

```
Out[7]: (300, 300, 3)
```

合成して出来たRGBデータは今まで同様imshowを使って描画できるが、RGBカラーで正しく表示できるのは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。

反射率は本来0以上1以下の値を取るが、ノイズ等のためにその範囲を逸脱することもあり得る。

☞ 以下のコードで反射率を0から1までの範囲に限定する。

入コード:

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

```
In [8]: rgb = rgb.clip(0, 1)
```

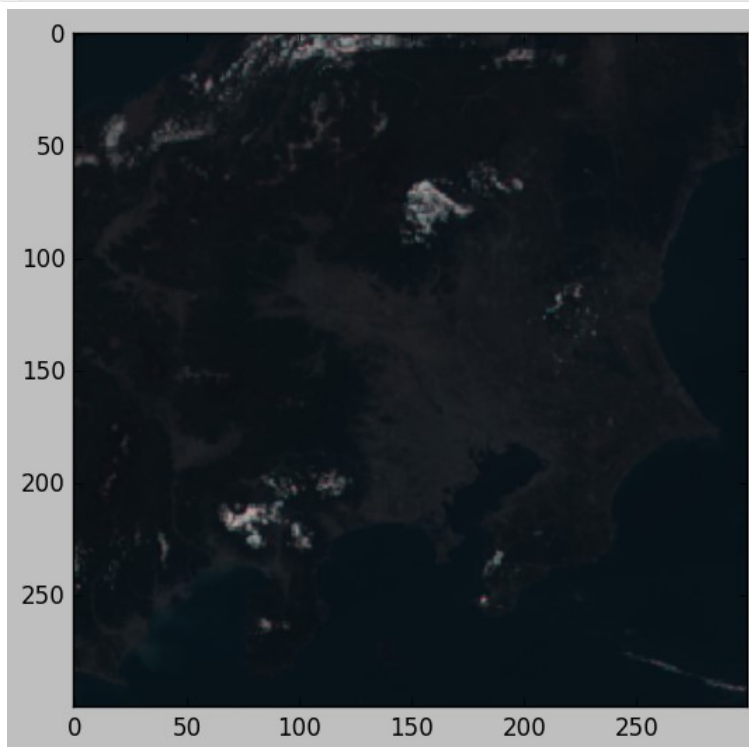
4.6 描画

☞ まずはimshowを使ってRGBデータをそのまま描画してみよう！

入コード:

```
plt.imshow(rgb);
```

```
In [9]: plt.imshow(rgb);
```



RGBデータをそのまま描画すると、しばしば暗い画像が得られる。原因としては、

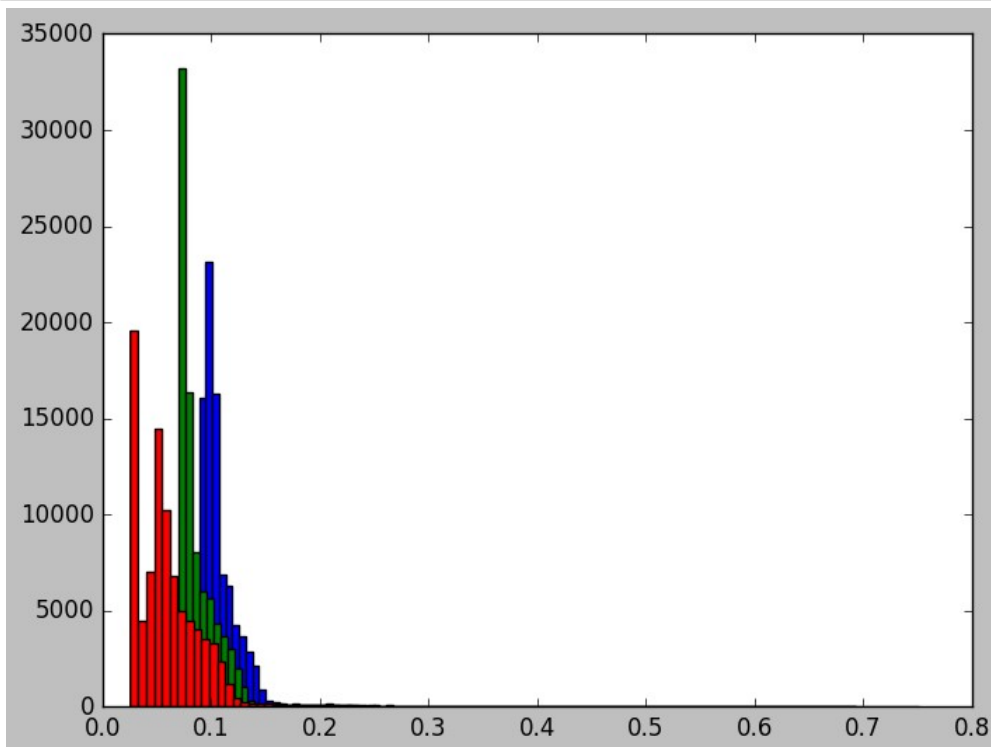
- ◎ 信号強度が小さい
- ◎ ディスプレイの特性が考えられる。

☞ 信号強度を確認するために、各色の反射率のヒストグラムを確認してみよう！

入コード:

```
plt.hist(value_b.flatten(), bins=100)  
plt.hist(value_g.flatten(), bins=100)  
plt.hist(value_r.flatten(), bins=100);
```

```
In [10]: plt.hist(value_b.flatten(),bins=100)
plt.hist(value_g.flatten(),bins=100)
plt.hist(value_r.flatten(),bins=100);
```



☞ 信号強度の最小値～最大値が0～1の範囲に収まるようにオフセットやゲインを調節してみよう！

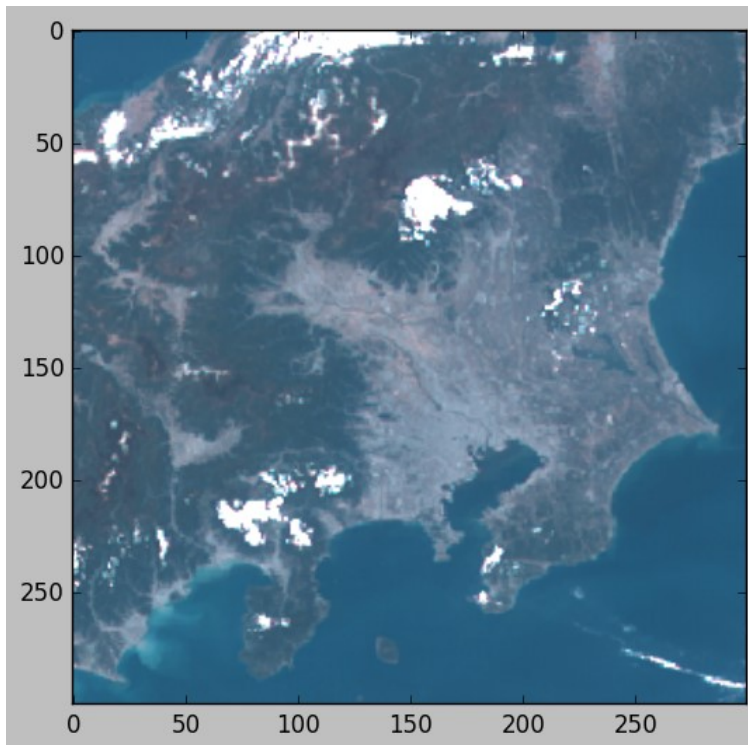
入コード:

```
rgb2 = np.dstack((value_r*5, value_g*5, value_b*5))
rgb2[(rgb2.max(axis=2) > 1.0)] = 1.0
plt.imshow(rgb2);
```

◎ 反射率の値はどの色もだいたい0.2より小さいので、1行目で各色を5倍にしている。

◎ 2行目では、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。

```
In [11]: rgb2 = np.dstack((value_r*5, value_g*5, value_b*5))
rgb2[(rgb2.max(axis=2) > 1.0)] = 1.0
plt.imshow(rgb2);
```



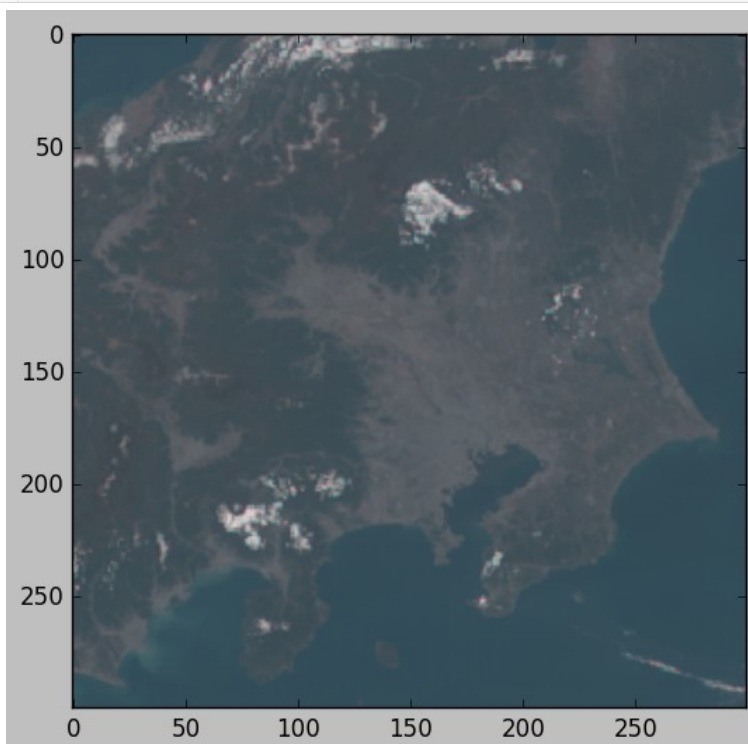
ディスプレイの特性として、信号強度と明るさは必ずしも比例しない。
この特性は $x' = x^{1/\gamma}$ のようなガンマ補正によって補正できる。
標準的なディスプレイの γ の値としては2.2程度が適当である。

👉 RGBデータにガンマ補正をかけてみよう！

入コード:

```
plt.imshow(np.power(rgb, 1.0/2.2));
```

```
In [12]: plt.imshow(np.power(rgb, 1.0/2.2));
```



5 ひまわり8号NetCDFデータの描画

日本域および機動観測域のひまわり8号データにはNetCDF形式のものが用意されている。このデータは等緯度経度座標系かつ校正済みのため、簡単に取り扱うことができる。

5.1 ひまわり8号NetCDFデータについて

NetCDF (Network Common Data Form) データ形式は多次元配列データの格納に適したデータ形式の一種であり、HDF (Hierarchical Data Format) と並んで衛星観測データの保存に広く使われている。NetCDFには自己記述的でポータブルという特徴があり、あらかじめどのようなデータが入っているか知らなくても決められた手順に従って読み出すことができる。現在のNetCDFはバージョン4が主流であり、PythonではnetCDF4というモジュールを使って扱える。

ひまわり8号のNetCDFデータは観測バンド毎の空間分解能に応じた等緯度経度座標系になっており、各座標の物理量(バンド1~6: 反射率, バンド7~16: 輝度温度)が格納されている。

NetCDFデータのファイル名はNC_H08_yyyymmdd_hhnn_Bbb_cccc_Rjj.ncのようになっており、yyyy、mm、dd、hh、nn、bb、cccc、jjにはそれぞれ年、月、日、時、分、バンド、観測領域、空間分解能に基づく値が入っている。

5.2 準備

☞ まず、いつもの準備を行う。

入コード:

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

```
In [1]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

☞ NetCDF形式のデータは、netCDF4というPythonモジュールを使って簡単に読み出せる。まず、netCDF4モジュールを使う準備をする。

入コード:

```
from netCDF4 import Dataset
```

```
In [2]: from netCDF4 import Dataset
```

5.3 NetCDFデータの読み込み

☞ ここでは例として近赤外バンド(バンド4)のNetCDFデータ("NC_H08_20160512_0400_B04_JP01_R10.nc")を使う。以下のようなコマンドでファイルを読み込み用にオープンする。

入コード:

```
nc = Dataset('NC_H08_20160512_0400_B04_JP01_R10.nc', 'r')
```

```
In [3]: nc = Dataset('NC_H08_20160512_0400_B04_JP01_R10.nc', 'r')
```

☞ データを変数に読み込んでみよう! バンド1~バンド6までは' albedo'、バンド7~バンド16までは' tbb' という変数名になっている。

入コード:

```
val = nc.variables['albedo'][:]
```

```
In [4]: val = nc.variables['albedo'][:]
```

☞ どの変数が読み込まれたか確かめてみよう！

入力例:

```
val.shape
```

In [5]: val.shape

Out[5]: (2701, 3301)

☞ 緯度経度を読み込んでみよう！ 緯度、経度はそれぞれ 'latitude'、'longitude' という変数名になっている。

入カコード:

```
lat = nc.variables['latitude'][:]  
lon = nc.variables['longitude'][:]
```

In [6]: lat = nc.variables['latitude'][:]
lon = nc.variables['longitude'][:]

☞ どの変数が読み込まれたか確かめてみよう！

入力例:

```
lat.shape
```

In [7]: lat.shape

Out[7]: (2701,)

☞ 読み終わったらファイルを閉じる。

入カコード:

```
nc.close()
```

In [8]: nc.close()

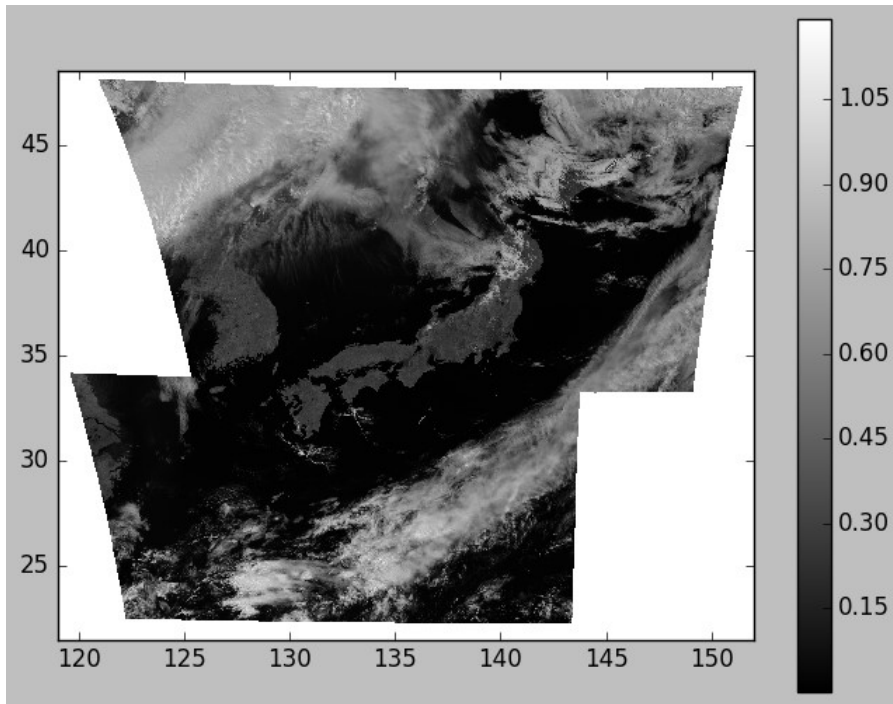
5.4 描画

☞ Matplotlibの imshowを使って2次元データを描画する。

入カコード:

```
plt.imshow(val, extent=(lon[0], lon[-1], lat[-1], lat[0]), interpolation='none')  
plt.colorbar();
```

```
In [9]: plt.imshow(val, extent=(lon[0], lon[-1], lat[-1], lat[0]), interpolation='none')
plt.colorbar();
```



6 ひまわり標準データの描画

フルディスクのひまわり標準データはグリッドデータに比べて取り扱いが難しいが、グリッドデータにはないヘッダー情報やディスク周辺部の観測データを含む。

6.1 ひまわり標準データについて

ひまわり標準データはひまわり独自のバイナリデータ形式になっており、可視赤外放射計(AHI)の観測で得られたDN値と様々なヘッダー情報が含まれている。

フルディスクのひまわり標準データに収められているAHI観測データの列数および有効ビット数を表6.1にまとめる。ここで使うひまわり標準データは図6.1のように10個のセグメントに分割されており、1セグメント(1ファイル)当たりの行数は列数の1/10である。

ひまわり標準データにはAHIの各画素で得られたDN値が有効ビット数に関わらず2バイトの符号なし整数として記録されている。

表6.1. フルディスクひまわり標準データ

ひまわり8号バンド名	中心波長(μm)	空間分解能(km)	列数(fulldisk)	有効ビット数
01	0.47	1	11000	11
02	0.51	1	11000	11
03	0.64	0.5	22000	11
04	0.86	1	11000	11
05	1.6	2	5500	11
06	2.3	2	5500	11
07	3.9	2	5500	14
08	6.2	2	5500	11
09	6.9	2	5500	11
10	7.3	2	5500	12
11	8.6	2	5500	12
12	9.6	2	5500	12
13	10.4	2	5500	12
14	11.2	2	5500	12
15	12.4	2	5500	12
16	13.3	2	5500	12

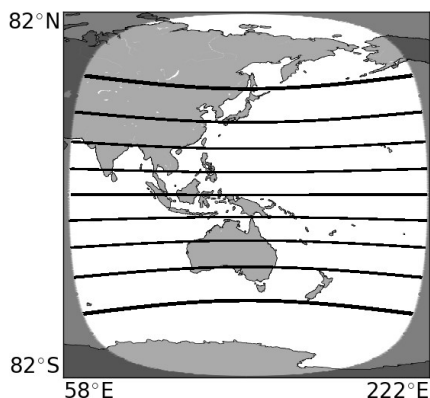


図6.1 フルディスクひまわり標準データの観測領域

6.2 準備

☞ まず、いつもの準備を行う。

入コード:

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

```
In [1]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

☞ 次に、ヘッダー情報のサイズを計算するため(ファイルサイズを取得するため)にosモジュールを使う準備をする。また、今回は幾何補正で次元補間関数を使うため、その準備をする。

入コード:

```
import os
from scipy.interpolate import RectBivariateSpline
```

```
In [2]: import os
from scipy.interpolate import RectBivariateSpline
```

6.3 ひまわり標準データの読み込み

ここでは例として近赤外バンド(バンド4)のひまわり標準データ("HS_H08_20160512_0400_B04_FLDK_R01_SD210.DAT")を使う。

☞ このデータは以下のようなコードで読み込むことができる。

入コード:

```
fnam = 'HS_H08_20160512_0400_B04_FLDK_R10_S0210.DAT'
NCOL = 11000
NLIN = NCOL//10
hsiz = os.path.getsize(fnam)- NCOL*NLIN*2
with open(fnam,'rb') as fp:
    head = fp.read(hsiz)
    data = np.fromstring(fp.read(), dtype='u2').reshape(NLIN, NCOL)
```

詳細を非表示

◎ ここではAHI観測データの列数、行数とファイルサイズからヘッダーサイズを計算しているが、これらの値はヘッダー情報を解析して得ることもできる。

◎ 3行目では整数の商を得るために切り捨て除算演算子//を用いている。(Python3では整数同士に除算演算子/を使うと浮動小数点数の商が得られる。)

◎ 7行目ではfromstringの引数にdtypeを与えて変数の型を指定している。'u'は符号なし整数を意味し、後ろの2はバイト数を意味している。'u'以外にも'i'、'f'等のフォーマット文字があり、それぞれ符号付き整数、浮動小数点数を意味する。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を指定できる。型の指定方法はいくつかあるが、ここではコードが一番短いものを使用している。(例えば'u2'の代わりにnp.dtype('u2')やnp.uint16が使える。)fromstringのデフォルトは'f8'である。

```
In [3]: fnam = 'HS_H08_20160512_0400_B04_FLDK_R10_S0210.DAT'
NCOL = 11000
NLIN = NCOL//10
hsiz = os.path.getsize(fnam)- NCOL*NLIN*2
with open(fnam,'rb') as fp:
    head = fp.read(hsiz)
    data = np.fromstring(fp.read(), dtype='u2').reshape(NLIN, NCOL)
```

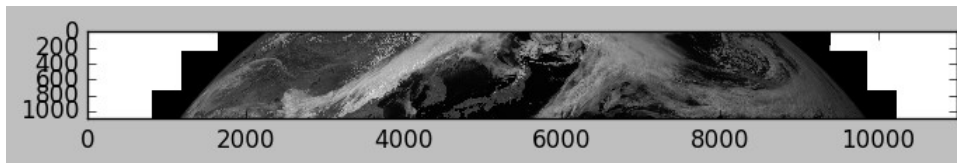
☞ これでデータが読み込まれたので、例えば以下のようなコードで描画できる。

入コード:

```
plt.imshow(data, vmax=2**11);
```



```
In [4]: plt.imshow(data, vmax=2**11);
```



6.4 ひまわり標準データのヘッダー情報の解析

前節で読み込んだデータはDN値であり、物理量を得るには校正が必要である。また、AHI独自の空間座標系になっているために幾何補正も必要となる。そのためにヘッダー情報の解析を行う。ここでは[ひまわり標準データのユーザーガイド \(http://www.data.jma.go.jp/mscweb/en/himawari89/space_segment/hsd_sample/HS_D_users_guide_en_v12.pdf\)](http://www.data.jma.go.jp/mscweb/en/himawari89/space_segment/hsd_sample/HS_D_users_guide_en_v12.pdf)に従って必要最低限の情報のみ取り出すことにする。

入コード:

```
imax, = np.fromstring(head[3:5], dtype='u2')
j, h = 0, []
for i in range(imax):
    n, = np.fromstring(head[j+1:j+3], dtype='u2')
    h.append(head[j:j+n])
    j += n
sub_lon, = np.fromstring(h[2][3:11])
cfac, lfac = np.fromstring(h[2][11:19], dtype='u4')
coff, loff = np.fromstring(h[2][19:27], dtype='f4')
p1, p2, p3, p4, p5, p6, p7 = np.fromstring(h[2][27:83])
band, = np.fromstring(h[4][3:5], dtype='u2')
wlen, = np.fromstring(h[4][5:13])
verr, vout = np.fromstring(h[4][15:19], dtype='u2')
gain, cnst = np.fromstring(h[4][19:35])
lnum, = np.fromstring(h[6][5:7], dtype='u2')
```

詳細を非表示

◎ ひまわり標準データのヘッダーは11 (= imax)個のブロックに分かれており、各ブロックの先頭にブロック番号とブロックサイズが格納されている。

◎ 2~6行目でヘッダーをブロック毎に分け、それ以後はブロック毎に必要なパラメータを取り出している。ここではシーケンスのアンパックという手法が使われている。シーケンスとはリストやタプル、ndarrayといったPythonの配列型のことである。例えば8行目の左辺は要素数2のタプル、右辺は要素数2のndarrayであり、右側のndarrayがアンパック(開梱)されて左辺のそれぞれの要素に代入されている。(Pythonのタプルは丸括弧で囲んで表すが、ここでは丸括弧が省略されている。)要素数1のタプルは値の後ろにコンマを付けるという規則があり、1行目の左辺にはコンマが付いている。(コンマを付けるとimaxは整数値になるが、そうしないとndarrayになる。)

```
In [5]: imax, = np.fromstring(head[3:5], dtype='u2')
j, h = 0, []
for i in range(imax):
    n, = np.fromstring(head[j+1:j+3], dtype='u2')
    h.append(head[j:j+n])
    j += n
sub_lon, = np.fromstring(h[2][3:11])
cfac, lfac = np.fromstring(h[2][11:19], dtype='u4')
coff, loff = np.fromstring(h[2][19:27], dtype='f4')
p1, p2, p3, p4, p5, p6, p7 = np.fromstring(h[2][27:83])
band, = np.fromstring(h[4][3:5], dtype='u2')
wlen, = np.fromstring(h[4][5:13])
verr, vout = np.fromstring(h[4][15:19], dtype='u2')
gain, cnst = np.fromstring(h[4][19:35])
lnum, = np.fromstring(h[6][5:7], dtype='u2')
```

6.5 ひまわり標準データの校正

前節で得られたヘッダー情報を使って校正を行う。まず、DN値(data)を放射輝度(lrad)に変換する。次に放射輝度を反射率(バンド1~6)または輝度温度(バンド7~16)に変換する。変換に用いる数式はひまわり標準データのユーザーガイド(http://www.data.jma.go.jp/mscweb/en/himawari89/space_segment/hsd_sample/HS_D_users_guide_en_v12.pdf)で説明されている。

入カコード:

```
LMIN = 1.0e-60
lrad = gain*data+cnst
lrad[lrad < LMIN] = LMIN
if band > 6:
    wlen *= 1.0e-6
    lrad *= 1.0e6
    c0, c1, c2, c_0, c_1, c_2, c_c, c_h, c_k = np.fromstring(h[4][35:107])
    t_e = c_h*c_c/(c_k*wlen*np.log(2*c_h*c_c**2/(wlen**5*lrad)+1))
    val = c0+c1*t_e+c2*t_e**2
else:
    coef, = np.fromstring(h[4][35:43])
    val = coef*lrad
```

詳細を非表示

◎ 3行目では $lrad < LMIN$ が成立する $lrad$ の要素の値を $LMIN$ に変更している。 $cond = (lrad < LMIN)$ とすると $cond$ は $lrad$ が $LMIN$ より小さい要素はTrue、そうでない要素はFalseのndarrayになる。(複数条件のandやorを取る場合は各条件を括弧で囲んでおくとよい。)このようにndarrayの要素を選択するためにはTrueまたはFalseのフラグが入った配列が使えるほか、欲しい要素のインデックスを入れた配列を使うこともできる。(例えば、 $indy, indx = np.where(lrad < LMIN)$ とすると $lrad < LMIN$ が成立するインデックスを取得でき、 $lrad[indy, indx]$ で要素を選択することができる。)

◎ 4~12行目では追加のヘッダー情報を取得して校正された放射輝度($lrad$)を反射率(バンド1~6)または輝度温度(バンド7~16)に変換している。

```
In [6]: LMIN = 1.0e-60
lrad = gain*data+cnst
lrad[lrad < LMIN] = LMIN
▼ if band > 6:
    wlen *= 1.0e-6
    lrad *= 1.0e6
    c0, c1, c2, c_0, c_1, c_2, c_c, c_h, c_k = np.fromstring(h[4][35:107])
    t_e = c_h*c_c/(c_k*wlen*np.log(2*c_h*c_c**2/(wlen**5*lrad)+1))
    val = c0+c1*t_e+c2*t_e**2
▼ else:
    coef, = np.fromstring(h[4][35:43])
    val = coef*lrad
```

6.6 ひまわり標準データの幾何補正

標準データの幾何補正方法としては、AHI画像座標(列番号,行番号)を緯度経度に変換する方法と、適当な等緯度経度座標をAHI画像座標に変換し、その格子点におけるAHI観測値を2次元補間によって求める方法が考えられる。ここでは結果が等緯度経度座標系になる後者の方法を採用する。変換に用いる数式は[LRIT/HRIT Global Specification5](http://www.cgms-info.org/documents/cgms-lrit-hrit-global-specification-(v2-8-of-30-oct-2013).pdf) ([http://www.cgms-info.org/documents/cgms-lrit-hrit-global-specification-\(v2-8-of-30-oct-2013\).pdf](http://www.cgms-info.org/documents/cgms-lrit-hrit-global-specification-(v2-8-of-30-oct-2013).pdf))で説明されている。

入コード:

```
lon = np.arange(138.0, 141.001, 0.01)
lat = np.arange(37.5, 34.499, -0.01)
col = np.arange(NCOL)+1
lin = np.arange(NLIN)+1num
rad_lat = np.radians(lat)
c_lat = np.arctan(p5*np.tan(rad_lat))
c_lon = np.radians(lon-sub_lon)
cos_lat = np.cos(c_lat).reshape(-1,1)
sin_lat = np.sin(c_lat).reshape(-1,1)
cos_lon = np.cos(c_lon).reshape(1,-1)
sin_lon = np.sin(c_lon).reshape(1,-1)
r1 = p3/np.sqrt(1.0-p4*cos_lat**2)
r1 = p1-r1*cos_lat*cos_lon
r2 = -r1*cos_lat*sin_lon
r3 = r1*sin_lat
rn = np.sqrt(r1*r1+r2*r2+r3*r3)
x = np.degrees(np.arctan(-r2/r1))
y = np.degrees(np.arcsin(-r3/rn))
col_out = coff+x/65536*cfac
lin_out = loff+y/65536*lfac
val_out = RectBivariateSpline(lin,col,val).ev(lin_out,col_out)
```

詳細を非表示

◎ ここではndarrayのブロードキャストという手法が使われている。これを使うと、例えば要素数が(1, nx)と(ny, 1)のndarrayの2項演算をするとあたかもそれぞれが要素数(ny, nx)のndarrayであるかのように扱われる(足りない行または列には同じ値が使われる)。9~12行目ではブロードキャストが行われるようにreshape(全要素数を保ったままndarrayの次元数や各次元の要素数を変更すること)を行っている。ここで、1つの次元の要素数に-1を指定すると、その次元の要素数は全要素数が保たれるように自動計算される。

◎ この例ではバンド4観測データの空間分解能が1 kmであることから東経138°~141°、北緯34.5°~37.5°の範囲を0.01度間隔で区切り、緯度経度座標をAHI画像座標に変換して格子点上のAHI観測値をRectBivariateSplineという2次元補間メソッドを用いて求めている。

```
In [7]: lon = np.arange(138.0, 141.001, 0.01)
lat = np.arange(37.5, 34.499, -0.01)
col = np.arange(NCOL)+1
lin = np.arange(NLIN)+1num
rad_lat = np.radians(lat)
c_lat = np.arctan(p5*np.tan(rad_lat))
c_lon = np.radians(lon-sub_lon)
cos_lat = np.cos(c_lat).reshape(-1,1)
sin_lat = np.sin(c_lat).reshape(-1,1)
cos_lon = np.cos(c_lon).reshape(1,-1)
sin_lon = np.sin(c_lon).reshape(1,-1)
r1 = p3/np.sqrt(1.0-p4*cos_lat**2)
r1 = p1-r1*cos_lat*cos_lon
r2 = -r1*cos_lat*sin_lon
r3 = r1*sin_lat
rn = np.sqrt(r1*r1+r2*r2+r3*r3)
x = np.degrees(np.arctan(-r2/r1))
y = np.degrees(np.arcsin(-r3/rn))
col_out = coff+x/65536*cfac
lin_out = loff+y/65536*lfac
val_out = RectBivariateSpline(lin,col,val).ev(lin_out,col_out)
```

6.7 描画

☞ Matplotlibの `imshow` を使って2次元データを描画する。

入コード:

```
plt.imshow(val_out, extent=(lon[0], lon[-1], lat[-1], lat[0]), interpolation='none')  
plt.colorbar();
```

In [8]: `plt.imshow(val_out, extent=(lon[0], lon[-1], lat[-1], lat[0]), interpolation='none')`
`plt.colorbar();`

