Python を利用したデータ解析入門

「観測データ解析」

2020 年後期

担当:

千葉大学環境リモートセンシング研究センター

市井 和仁, 楊 偉

http://ichiilab.weebly.com

1. Python の基礎

1.1. なぜ Python ?

現在では、地球環境・地域環境の把握・変動などの解析を行う際に、「コンピュータ」によるデータ解析は不可欠である。さらに、データが大量であることが多く、これらのデータ処理能力が研究や業務の効率を左右するといっても過言ではない。

例えば、気象データの解析などでは、気象庁などを通して様々な気候データ(気温・降水量・日照時間等)を様々な時間スケール(日・月・季節・年)で入手することが可能である(例: http://www.data.jma.go.jp/obd/stats/etrn/index.php では、日本全国の、気象観測点の過去のデータの閲覧・取得が可能)。日本全体の気候変化などを議論するには、一点のみならず、なるべく多くの地点の観測値を利用した解析が必要になる。さらに、例えば、点の観測データを平面(2 次元)データにしたうえで解析を行うことも有用である。また、世界の気候(地球温暖化などの広域現象)の変動を把握したい場合には、http://cdiac.ornl.gov/　(二酸化炭素情報分析センター; Carbon Dioxide Information Analysis Center; アメリカ)や、http://www.cru.uea.ac.uk/　(Climatic Research Unit, University of East Anglia; 英国)などの機関より、様々なデータが公開されている。例えば、過去約 100 年程度の地球各地の気温や降水量などの基本的な値は、全球 0.5 度程度の空間解像度でデータが準備されている。その他、様々な物理・化学・生物実験・水循環や気象現象などの観測などにおいても膨大なデータが作成されることが多い。

以前は、実験・観測などを中心に行っていた実務者・研究者には、高度なデータ解析能力などはあまり必要とされてはいなかったが、近年のデータ量の増加と、他分野との融合の機会の増加により、高度なデータ解析能力が必要とされるようになってきた。また、逆に、高度なデータ処理能力を持つことで、様々な業務を効率的にこなすことができるようになる。また、本テキストで扱うようなデータ処理手法は、環境分野のみならず、他の様々な分野での応用にも使える。

コンピュータを使って(大量の)データ処理を行うには、以下の 3 つの方法がある。

① GUI ベースで行えるソフトウェア (Excel, Arc-GIS, QGIS, ENVI その他様々)の利用
② インタプリタ的なソフトウェア (Python, R, MATLAB, OCTAVE, その他様々)の利用
③ 伝統的なプログラミング (C 言語/FORTRAN 等)の利用

①については、実際に Excel などを用いて、大量データの解析を行うのは容易ではない。複数のデータがあれば、複数のシートを開いたり、データの貼り付けが面倒であったり、グラフの作成・修正が困難だったり、大量のグラフを作成するにも、すべて手作業で行うために非常に時間がかかりそうだ(もちろんマクロなどを利用して自動化することも可能ではあるが)。Excel は、データ形式がテキストであれば可能だが、バイナリであれば、基本的には何もできない。

③については、処理速度を求める際(大型計算、大型データ処理)、または、きめ細かい解析を行う際に効率的であろう。また、既存の資産として、様々な数値モデルは C 言語や FORTRAN などで提供されていることが多い。ただし、伝統的なプログラミングを学ぶこと自体が、かなりの労力を必要とする。ただし将来的にコンピュータを扱う専門化、データ解析を行う専門家になるとしたら、避けては通れない道である。ただ、データの可視化(グラフ・画像の作成)においては、C 言語などでは対応しにくい部分があ

って(もちろん可能ではあるが)、必然と①や②との併用になるだろう。

私個人的な意見としては、中間的な②は、「そこそこ」きめ細かい解析ができ、結果の可視化なども容易であるため、理工系の学生であれば、習得すべき技術であると考える。私自身も、実際の研究の場面で、非常に多用している。元々、私自身は、②に対応する技術は持ち合わせていなかった（あまり気にしていなかった)。しかし、あるとき思いついて少し時間をかけて勉強することにより、容易に習得できることが分かり、今では欠かせないツールになった。あまりに便利であるので、簡単な処理時間を必要としない処理であれば、②で処理を行うことが多い。③に比較すると実行速度の遅さ、という欠点はあるが、プログラミング技術の習得に関しては、③に比較して容易である。C 言語などの本格的なプログラミングに関する授業については、本テキストでは扱わないが、特にスピードを要する場合(例えば数値モデル計算)や、過去の財産に依存する場合(C や FORTRAN にはこれまでの長い歴史の中で成熟されたプログラム(関数)が大量に存在する)には必要である。データ処理やモデル計算を通したプログラミングを極めたい場合には是非学習して欲しい項目である(個別に、授業用のテキストなども所有しているので希望者はお知らせください)。

以上のことから、本実習では、②タイプの言語を用いたプログラミング・データ解析手法を学び、いくつかの環境データを図示(グラフ・画像表示)し、簡単な統計解析を行えるようにすることを目標とすることとした。言語は、"Python"とする。Python は、最近注目されているプログラミング言語であり、データ処理、機械学習、その他諸々で非常に強力であると言われている。ググればすぐにわかるだろう。

本実習資料については、Python を用いたデータ解析の導入部分を学ぶということを目的に構成した。特に衛星データや様々な環境データについて、データを読み、統計解析を行い、グラフや図などの表示をするという基本的な部分に特化している。さらに勉強をしたい受講生は、ネットで様々な教材を探してみることをお勧めする。巻末に役に立ちそうな URL を示す。

1.2. Python をさわってみよう（→　インストール～実行が最も困難かもしれません！）

早速、Python を利用してみよう。以下うちインストール手順に関しては、Windows を想定して記載しているが、その他の OS(Linux, Mac 等)でも同様にできるはずである。

0) Python Web ページ. (visit: **https://www.python.org/**)

ソフトウェアのダウンロード:

Python Web	**https://www.python.org**	(Linux, Windows, Mac)
Anaconda	**https://www.anaconda.com/download/**	(Linux, Windows, Mac)

Anaconda: Python を使いやすいようにまとめあげたもの

初心者の方、もしくは、Windows の方は、anaconda などが便利です。

（Anaconda から起動した console (qtconsole), Spyder, jupiter notebook を使うと便利）

市井のおすすめ”は Spyder” (でも Jupyter Notebook もよく使われます)

Tips: ディレクトリ操作の基本；(Anaconda-ipython etc.などにはあるが、素の Python にはない)

```
>>> pwd                現在のディレクトリ位置を表示
>>> ls                 現在のディレクトリ上のファイルを表示
>>> mkdir dir          ディレクトリ dir を作成
>>> cd dir                    ディレクトリ dir へ移動する
>>> cd ..                     一つ上のディレクトリへ移動する
>>> pwd                現在のディレクトリ位置を表示
```

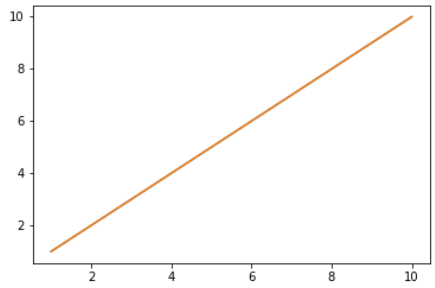
1)　行列の扱い(Python の大きな利点)；　重要なのでじっくりと考えて進めてください

```
>>> import numpy as np              行列（配列）のライブラリを読む（必須&重要）
>>> A=np.array([[ 1, 1, 2], [3, 5, 8], [13, 21, 24]])      2 次元配列を作成する
>>> A          行列を表示
>>> A.size     行列 A のサイズ(要素数)を表示
>>> A.shape    行列 A のサイズ(行、列)を表示
>>> A.ndim     行列 A の次元を表示
>>> A.dtype    行列 A の数値のタイプを表示
        (例: int32   4 バイト(32 ビット)整数, float64  浮動小数点 8 バイト形式など)
>>> B = A + 0.5       行列 A の要素すべてに 0.5 を足す
>>> B          行列 B を表示
>>> B.dtype    行列 B の数値のタイプを表示(先ほど 0.5 を足したもの)

>>> A*2                2 倍
>>> A*A                行列の要素ごとの掛け算(手計算の結果と比較してみよう)
>>> A@A                行列の掛け算(上との違いを見てみよう)
>>> np.dot(A,A)        行列の掛け算
>>> np.transpose(A) 転置（意外と後で使うかもー２次元画像の縦横変換）
>>> np.linalg.det(A)          行列式
>>> np.linalg.inv(A)          逆行列
>>> np.dot(A,np.linalg.inv(A))    元の行列 x 逆行列(単位行列になるか確認)
>>> [v,lamda]=np.linalg.eig(A)    行列 A の固有値・固有値ベクトル計算
>>> v          v は固有値
>>> lamda              v は固有ベクトル

>>> B=A.reshape(-1,)                  行列 A を 1 次元配列にする(B)
>>> B=np.ravel(A)             別のやり方
>>> B[7]                      1 次元表記した場合（配列のどこが表示されたか確認）
>>> A[2,1]                    2 次元表記した場合（配列のどこが表示されたか確認）
```

```
>>> idx=np.where(B>10)      B>10 をみたす配列要素を表示する (MATLAB->find)
>>> idx                     対応する配列の要素番号が返ってくる
>>> B[idx]                  上記で抽出された要素(idx)のみを表示
>>> idx=np.where(A>10)      A>10 をみたす配列要素を表示する (MATLAB->find)
>>> idx                     対応する配列の要素番号が返ってくる(A は 2 次元なので 2 次元)
>>> A[idx]                  上記で抽出された要素(idx)のみを表示
```

2) 行列の作成・グラフの描画

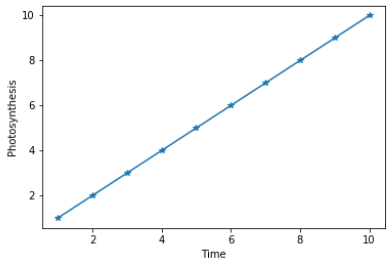
```
>>> import matplotlib.pyplot as plt     作図などのライブラリ読み込み（必須）
>>> import numpy as np                  数値処理のライブラリ読み込み（必須）
>>> t = np.array(range(1,11))           (from 1 to 10 step 1)
>>> t = np.array(range(1,11,2))         (from 1 to 10 step 2)
>>> x = np.array(range(1,11))
>>> y = np.array(range(1,11))
>>> plt.plot(x,y)
>>> plt.show()
```



Output example

```
>>> plt.plot(x,y,'*')
>>> plt.show()

>>> plt.plot(x,y,'*-')
>>> plt.show()

>>> plt.plot(x,y,'*-')
>>> plt.xlabel("Time")
>>> plt.ylabel("Photosynthesis")
>>> plt.show()

>>> plt.plot(x,y,'*-')
```

```
>>> plt.xlabel(“Time”)
>>> plt.ylabel(“Photosynthesis”)
>>> plt.savefig(“test.png”)          画像ファイルとして保存
>>> plt.savefig(“test.pdf”)          画像ファイルとして保存
```



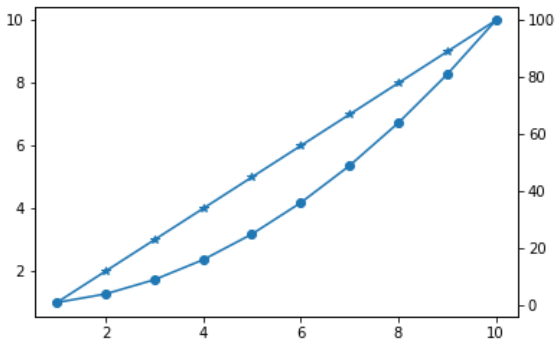
output example…

以下、続けてやってみて、どのようになるかを試してください…

```
>>> y2=y*y
>>> plt.plot(x,y,"*-",x,y2,"o-")
>>> plt.show()

>>> plt.plot(x,y,"*-",x,y2,"o-")
>>> plt.xlim(1,13)
>>> plt.ylim(1,120)
>>> plt.show()
```

#2 軸グラフの作成

```
>>> fig, ax1 = plt.subplots()
>>> ax1.plot(x,y,"*-")
>>> ax2 = ax1.twinx()
>>> ax2.plot(x, y2, "o-")
>>> plt.show()
```



output example…

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x=np.array(range(1,11))
>>> y=np.array(range(1,11))
>>> plt.subplot(2,3,1)
>>> plt.plot(x,y)
>>> plt.subplot(2,3,2)
>>> plt.plot(x,y*2)
>>> plt.subplot(2,3,3)
>>> plt.plot(x,y*3)
>>> plt.subplot(2,3,4)
>>> plt.plot(x,y*4)
>>> plt.subplot(2,3,5)
>>> plt.plot(x,y*5)
>>> plt.subplot(2,3,6)
>>> plt.plot(x,y*6)
>>> plt.show()
```

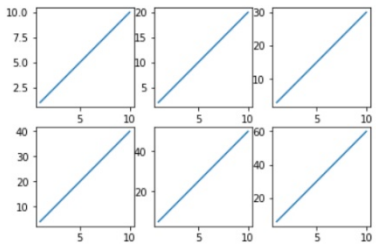
###補足開始###

また、plt.subplot, plt.plot をまとめて一行で

```
>>> plt.subplot(2,3,1), plt.plot(x,y)
```

と記載することも可能。

###補足終了###



3) Python によるプログラムの作成

Python では、一連のコマンドをファイルに書くことによって、スクリプト(プログラム)が書ける。例えば、テキストエディタを用いて、以下の内容を持つ sample1.py のファイルを作成してみよう。

以下のテキストを持つファイルを作成する　名前： sample1.py

#####スクリプト開始#####

```
# sample1.py                    （コメント行は#で始める）
```

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

x=np.array(range(1,11))
y=np.array(range(1,11))

plt.subplot(2,3,1)
plt.plot(x,y)
plt.subplot(2,3,2)
plt.plot(x,y*2)
plt.subplot(2,3,3)
plt.plot(x,y*3)
plt.subplot(2,3,4)
plt.plot(x,y*4)
plt.subplot(2,3,5)
plt.plot(x,y*5)
plt.subplot(2,3,6)
plt.plot(x,y*6)

plt.show()
#####スクリプト終了#####
```

そして、Python 上にて以下のコマンドを実行

Anaconda – JupyterNotebook, qtconsole の場合：Python のコマンドプロンプトに入った後に
>>> `run –i sample1.py`

Python3 本体の場合:
`python3 sample1.py`
`python3 –i sample1.py`
　　(`-i` を付けると終了後に python3 の実行したまま終了。変数などの確認が可能)

また、IF や FOR や WHILE などの基本的なループ構造を利用することができる。

(`sample1.py` と同様に `sample2.py` というファイル名で以下の内容のファイルを作成しよう。)

```
#####スクリプト開始#####
#PYTHON sample program: sample2.py
```

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

x = np.array(range(1,11))
y = np.array(range(1,11))

for i in range(0,6):
    plt.subplot(2,3,i+1)
    plt.plot(x,y*(i+1))

plt.show()
#####スクリプト終了#####
```

`for i in range(0,6)` として、i=0 から 5 まで (step 1)のループを示している(0 から 1 刻みで 6 は含まない)。各ループ内で、`subplot()`, `plot()` が実行され、これまで得られたグラフと同じ結果を得ることができる。これは後で絶対使える！
また、例えば、1 つ飛びにしたければ、`for i in range(0,6,2)`のように表現可能(1, 3, 5 の順に実行される)。

4) 便利な関数(統計ツールとして); C 言語や FORTRAN などのプログラミングより簡単！

```
>>> import numpy as np
>>> testdata = np.array([ 1, 1, 2, 3, 5, 8, 13, 21, 24])
>>> np.average(testdata)          平均を求める
>>> np.mean(testdata)             平均を求める
>>> sum(testdata)                 合計を求める
>>> np.max(testdata)              最大を求める
>>> np.min(testdata)              最小を求める
>>> np.mean(testdata[3:5])        要素 4-5 の平均を求める
>>> np.std(testdata)              標準偏差を求める
>>> t=np.array(range(1,10))       適当に作る(各点に対して時間軸を仮定した)
>>> np.corrcoef(t,testdata)       相関係数を求める
>>> np.polyfit(t,testdata,1)      直線回帰(最小自乗法) 出力：最初が傾き、次が切片
>>> p=np.polyfit(t,testdata,1)    結果を変数 p に格納
>>> p.size                        変数 p のサイズチェック(double 2 要素では?)
>>> p[0]                          p(0)のチェック
>>> p[1]                          p(1)のチェック
```

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(t,testdata,t,t*p[0]+p[1])  回帰直線をもグラフ表示
>>> plt.show()                          一旦表示

>>> plt.plot(t,testdata,t,t*p[0]+p[1])  plt.showすると図は書き直し
>>> plt.xlim(1,10)                      軸の設定
>>> plt.xlabel('Time')                  軸ラベルの設定
>>> plt.ylabel('CO2 Conc.')             軸ラベルの設定
>>> plt.title('CO2 Time-series')        軸ラベルの設定
>>> plt.title('CO2 Time-series',fontsize=18)
          軸ラベルの設定（フォントサイズを指定したい場合）
>>> plt.show()

>>> plt.plot(t,testdata, label="original")
>>> plt.plot(t,t*p[0]+p[1], label="linear regression")
>>> plt.legend()                        凡例を作る
>>> plt.show()

>>> plt.plot(t,testdata,t,t*p[0]+p[1])
>>> plt.grid()                          グリッドラインの作成
>>> plt.show()

>>> plt.plot(t,testdata,t,t*p[0]+p[1])
>>> plt.grid(color='r', linestyle='--') グリッドラインの作成
>>> plt.show()


>>> %reset                  メモリに格納された変数をすべてリセット
>>> testdata3=np.array([1,1,2,3,5,8,13,21,24,30,33,40]).reshape(4,3)
                            もう一度定義　（配列の作り方　その 2）
>>> np.mean(testdata3)      平均をとったが、どうなった?
>>> np.mean(testdata3,0)    平均をとったが、どうなった?
>>> np.mean(testdata3,1)    平均をとったが、どうなった?
```

Testdata3 を画像表示してみよう。 ごく導入です。

```
>>> import matplotlib.pyplot as plt
>>> plt.imshow(testdata3)  表示する画像の設定
```

```
>>> plt.colorbar()          カラーバーの付与
>>> plt.show()  画像表示
```

今回は、さっと Python の機能について、配列操作・グラフ描画・統計処理について流してみた。今後は、これらを基礎として、さらに複雑な解析をおこなっていく予定である。特に配列の扱いについては、Python での大きな利点の一つ(C や FORTRAN ならば、FOR ループを使わないといけない)である。是非有効利用してもらいたい。

レポート課題 1

何でもいいので、t, testdata, その線形回帰などを使って、
かっこいいグラフを作ってみてください。　作った命令も記載してください。

補足：

リモートセンシング、他環境データ（観測データ）を扱う上で重要なポイント：

① 基本的なプログラミング構造
② テキストデータの読み書きとグラフ表示
③ バイナリデータ（画像データ）の読み書きとグラフ表示
④ データを取得(wget など)したり、フォーマット変換したり
⑤ 統計知識、データをどう調理するか？

My suggestion:

Japanese books, URL:

詳細！ Python 入門ノート (大重美幸) (ソーテック社)
みんなの Python 4th edition (柴田淳) (SB Creative)
ゼロから始めるデータサイエンス (Joel Grus) (O'REILLY) 少し難しいかも
http://bicycle1885.hatenablog.com/entry/2014/02/14/023734

English:

https://www.python.org/about/gettingstarted/
https://www.learnpython.org/
http://blog.rtwilson.com/resources-for-learning-python-for-remote-sensing-or-switching-from-idl/ (FOR IDL users)
https://jakevdp.github.io/PythonDataScienceHandbook/index.html

2. Python によるテキストデータの解析

2.1. はじめに

いよいよ、実際のデータ(私自身の研究の中でも使うことがある)を利用した解析に入ってゆく。本セクションでは、インターネット上で入手できるいくつかの観測データ(テキスト形式)をダウンロードし、自分で加工し、意味のある結果を構築できる(統計計算やグラフ描画)ようになることを目標とする。IPCC(気候変動に関する政府間パネル)報告書などでも示されている全球気温の変化(温暖化の一つの観測例)などについても、以下の知識があれば、自分で図を作成することも可能である。

まずは、今回の解析で用いるデータを見ていこう。

1) Global Temperature (2020/09/05 確認)

http://cdiac.ess-dive.lbl.gov/　　Carbon Dioxide Information Analysis Center (CDIAC), US

Data: Climate → Temperature

→ Global and hemispheric temperature anomalies - land and marine instrumental records

(全球・半球気温のアノマリ - 陸・海の測器による記録)

と辿ると、以下のサイトになる。

w

ここから、データの直接閲覧(Digital Data)・説明・グラフ(Graphics)などを確認できる。

今回は、Table - Global Monthly and Annual Temperature Anomalies, 1850-2015 を利用する。

2) Atmospheric CO_2 concentration (2020/09/05 確認)

http://cdiac.ess-dive.lbl.gov/　　Carbon Dioxide Information Analysis Center (CDIAC)

Data: Atmospheric Trace Gases and Aerosols → Carbon Dioxide

→ Scripps Institution of Oceanography Network (長期データがある)

と辿ると、以下のサイトになる。

http://cdiac.ess-dive.lbl.gov/trends/co2/sio-keel.html

ここでは、アメリカのスクリプス海洋研究所で観測されたいくつかの観測地点での CO_2 濃度データが掲載されている。とりあえず、最長のデータ期間を持つ[Mauna Loa, Hawaii]を選択し、図やデータや説明を見てみよう。

3) Southern Oscillation Index (南方振動係数)　(2020/09/05 確認)

http://www.cru.uea.ac.uk/　Climate Research Unit, University of East Anglia, UK.

様々な気候データがあるサイト

Data → Pressure and Circulation Indices

→ Pressure data, NAO, SOI and other circulation indices, etc.

→ Southern Oscillation Index (SOI)

課題 2.1. Southern Oscillation Index (南方振動係数)は何か調べ、できるだけ具体的に説明すること。自身でデータを作成できればさらにすばらしいです（全く同じ値になるかはわかりませんが、近い値は出せるはず)。

2.2. 解析例 1: Global Temperature Record

では、いよいよ、データ解析に移る。Python に限らず、一般的なデータ解析では、

1) データの取得
2) データの形式変換
3) データの読み込み(必要に応じてデータを読み込める形に加工)
4) データの解析

となり、実際の解析の前に、データの取得や加工・読み込みが必要となる。データの形式は、データ配布機関やデータの種類によって異なるので、使用するソフトウェアに適切な形式に変換を行う必要がある(フォーマット変換)。実際、実務や研究の場面では、フォーマット変換などのデータ準備に非常に多くの時間を割かれることが多い(一説には研究時間の約 80%はデータ準備に時間を費やされるとか…)。

0)　作業ディレクトリへ移動
作業ディレクトリ(`¥MyDocument` など、どこでもよい)を作成し、移動する。
本章は、そこで作業を行う。

1)　データの取得
先ほど確認をした Global Temperature のデータをダウンロードしよう。

`https://cdiac.ess-dive.lbl.gov/ftp/trends/temp/jonescru/global.txt`

(名前をつけて保存→ファイルの種類：テキストファイル(*.txt)　とする)
一応、Notepad++または Wordpad (メモ帳では改行されない; 本ファイルのフォーマットが Windows 用ではないため)などで、ファイルを確認してみよう。

このデータは、1850 年から最近(2010 年)までの全球(海+陸)平均気温が格納されている。相対値として、1961-1990 年の平均値からの偏差で格納されている(コメント行の一部; ダウンロードしたファイルの 2, 4 行目)。データについては、最初の列では年、次に 1-12 月、最後に年平均の値が格納されている。

2)　フォーマット変換
テキストファイルの場合は、コメント行(今回のファイルでは、1-17 行目)が曲者で、こういったコメント行を何らかの方法で回避する必要がある。Python では、`skiprows` を指定することで、先頭から指定した行までをヘッダとみなして読み飛ばすことができる（フッターは指定不可のため、`comments` で読み飛ばす行の先頭の文字を指定するか、あらかじめ元データからフッターを削除する必要がある)。

3) Python で読む

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

>>> din_t = np.loadtxt("global.txt", skiprows=17, usecols=range(0, 14))
```

np.loadtxt(ファイル名)はテキストファイルを読む命令。結果が din_t (名前は任意)に格納される。 skiprows は読み飛ばす行の設定、usecols は読み込む列の設定。

```
>>> din_t.shape
```

166 x 14 data;　166 years, 14 data (year, jan,feb,,,, dec, ann);

形式は double(浮動小数点実数倍精度)

```
>>> din_t          すべてのデータを表示する
>>> din_t[:,0]     1 列目をすべて表示する　(年が出る)
>>> din_t[:,13]    14 列目をすべて表示する　(年平均値が出る)
>>> din_t[2,:]     3 行目をすべて表示する　(1852 年の値が出る)
>>> din_t[2,9]     3 行目 10 列目を表示する　(1852 年 9 月の値が出る)
```

以上のような形でデータ値の参照が可能である。元データのファイルと比較してみよう。

4) グラフの描画(意味を考えながらやってみてください。plt.show()は適宜入れてください)

例 1:

```
>>> plt.plot(din_t[:, 13])
>>> plt.plot(din_t[:, 0], din_t[:, 13])
>>> plt.plot(din_t[0:150, 0], din_t[0:150, 13])  (1850-1999 年まで)
>>> plt.plot(din_t[131:161, 0], din_t[131:161, 13]) (1981-2010 年まで)
```

例 2:

```
>>> plt.plot(din_t[:, 0], din_t[:, 13])
>>> plt.xlim(1950,2000)
>>> plt.ylim(-0.6,0.6)
>>> plt.xlabel("Year")
>>> plt.ylabel("Temperature")
>>> plt.show()
```

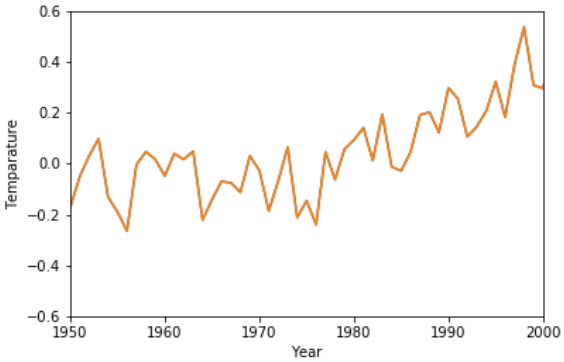


図 1. 全球平均気温の推移

例 3: 以下のスクリプトを作成して実行してください(行番号は不要): ファイル名 global_temp1.py とする。

```
01  #
02  # Python Sample: global_temp1.py
03  #
04
05  import matplotlib.pyplot as plt
06  import numpy as np
07  din_t = np.loadtxt("global.txt", skiprows=17, usecols=range(0, 14))
08  fig = plt.figure(figsize=(10,8))
09
10  for i in range(0,12):
11            plt.subplot(4,3,i+1)
12            plt.plot(din_t[:, 0], din_t[:, i+1])
13            plt.xlim(1950,2000)
14            plt.ylim(-0.6,0.6)
15            plt.xlabel("Year")
16            plt.ylabel("Temperature")
17            buf="Month"+"{0:02d}".format(i+1)
18            plt.title(buf)
19
20  fig.suptitle('Global Temperature', fontsize=12)
21  plt.tight_layout()
22  plt.subplots_adjust(top=0.90)
23
24  plt.show()
```

10-17 行目: 一つのループ(i=0 から 11 まで、計 12 回)

12 行目: y 軸の項目は `din_t(:,i+1)`。`i` が変わる毎に表示する項目が変わる(月が変わる)

13,14 行目: x 軸・y 軸表示範囲の設定

15,16 行目: x 軸・y 軸ラベルの付与

17 行目: `buf="Month"+"{0:02d}".format(i+1)`は、文字列を作成している。

例えば、実行後に

```
buf.shape
```

と入力することで、変数 `buf` の大きさを見ることができる。また、`buf` と入力すると、`buf` に格納された文字列が表示される。

`%02d` で書かれた部分に、2 桁の整数(d)で、2 桁に満たない場合は 0 を付ける、という形式で i の値を入れる、ということを示す。ちなみに小数の場合は`'d'`の代わりに`'f'`とする。【sprintf に対応する、str.format()を使っても書ける（省略）】

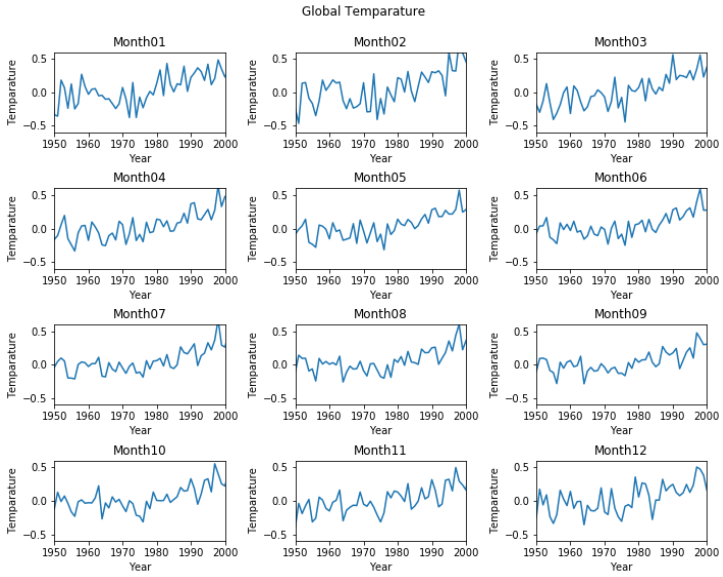


図 2. 結果の一例

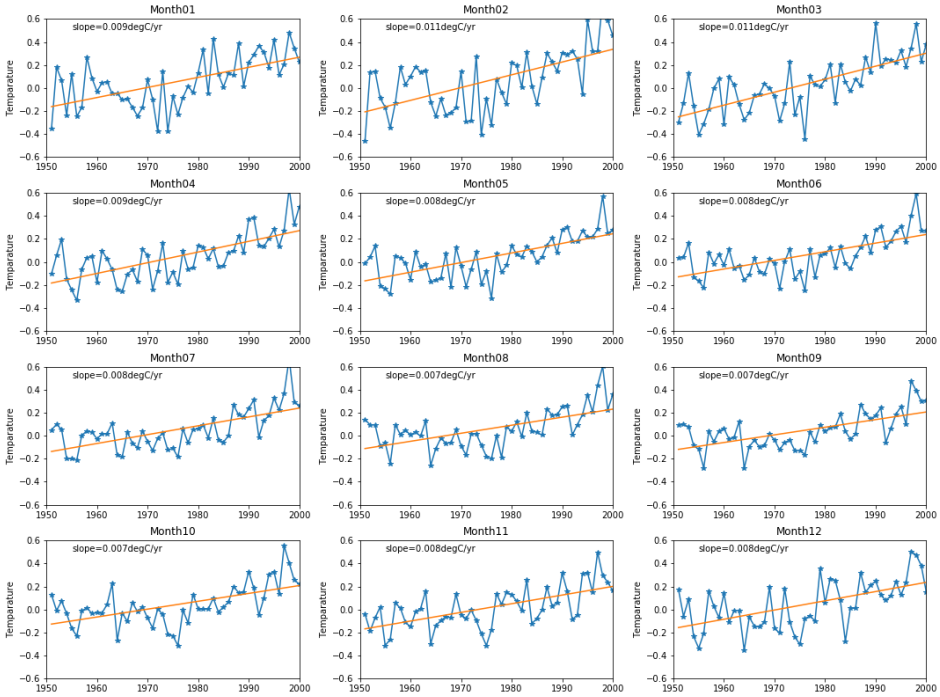
例 4: さらに応用: ファイル名 `global_temp2.py`

定量的な解析(例えば温度は一年あたり何度上がっている?)をしてみよう。(1951-2000)

```
01  #
02  # Python Sample: global_temp2.py
03  #   with Linear Regression Line
04  #
05
06  import matplotlib.pyplot as plt
07  import numpy as np
08
```

```
09 din_t = np.loadtxt("global.txt", skiprows=17, usecols=range(0, 14))
10
11 xdata = din_t[(1951-1850):(2000-1850+1), 0]
12 fig = plt.figure(figsize=(15,12))  #figsize=(15,12)は各自で調整（図の大きさ）
13
14 for i in range(0,12):
15     ydata = din_t[(1951-1850):(2000-1850+1), i+1]
16
17     #linear regression
18     p = np.polyfit(xdata, ydata, 1)
19     ydata_reg = xdata*p[0]+p[1]
20
21     plt.subplot(4,3,i+1)
22     plt.plot(xdata, ydata,"*-",xdata,ydata_reg,"-")
23     plt.xlim(1950,2000)
24     plt.ylim(-0.6,0.6)
26     plt.ylabel("Temperature")
27     buf = "Month"+"{0:02d}".format(i+1)
27     plt.title(buf)                                    (タイトルを付けた)
28     buf = "slope="+"{0:.3f}".format(p[0])+"degC/yr"
29     plt.text(1955, 0.5, buf)                          (傾きを表示)
30

31 plt.tight_layout()
32 plt.subplots_adjust(top=0.90)
33
34 plt.show()
```

11 行目: 1950-2000 年に対応する部分のデータを別に作成した。
11 行目: xdata に関しては一度作成するだけで、すべての月に利用できる。
18-19 行目: 1950-2000 年の部分のみを用いて回帰をする。
22 行目: plot()では、元データ・回帰直線の両者をグラフ化
28 行目: グラフに表示させる文字列を作成
29 行目: グラフ中の(x,y)=(1955,0.5)に buf を書く(結果を参照)



さらに、別の年代についても解析がしたくなる(1900-2000 年での温度上昇は何度とか)。もう少し進化させてみよう。プログラムを組む際は、とりあえず作ってしまうのではなく、いろいろと汎用性を持たせるように作成すると、後の作業が楽になる場合が多い。ファイル名 global_temp3.py。

```
01  #
02  # Python Sample: global_temp3.py
03  #   with Linear Regression Line
04  #
05
06  import matplotlib.pyplot as plt
07  import numpy as np
08
09  start_year = 1951
10  end_year = 2000
11
12  infile = "global.txt"
13  outfile = “global_temp_"+"{0:04d}".format(start_year)+"_"+"{0:04d}".format(end_year)+".pdf"
14  din_t = np.loadtxt(infile, skiprows=17, usecols=range(0, 14))
15  xdata = din_t[(start_year-1850):(end_year-1850+1), 0]
16  fig = plt.figure(figsize=(15,12))
17
```

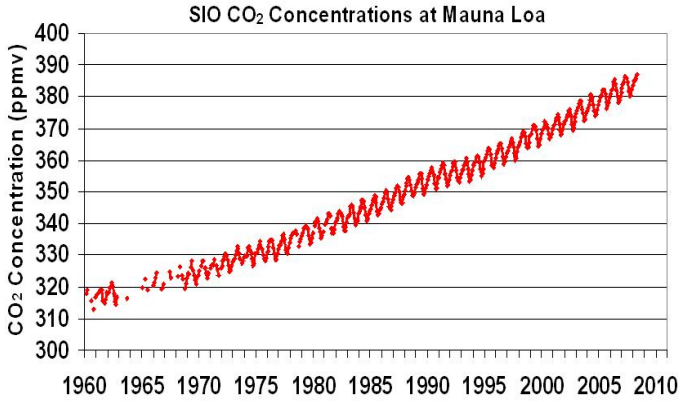
```
18 for i in range(0,12):
19     ydata = din_t[(start_year-1850):(end_year-1850+1), i+1]
20
21     #linear regression
22     p = np.polyfit(xdata, ydata, 1)
23     ydata_reg = xdata*p[0]+p[1]
24
25     plt.subplot(4,3,i+1)
26     plt.plot(xdata, ydata,"*-",xdata,ydata_reg,"-")
27     plt.xticks(np.arange(start_year, end_year+1, 10))
28     plt.ylim(-0.6,0.6)
29     #plt.xlabel("Year")
30     plt.ylabel("Temperature")
31     buf="Month"+"{0:02d}".format(i+1)
32     plt.title(buf)
33     buf = "slope="+"{0:.3f}".format(p[0])+"degC/yr"
34     pos_x = (end_year-start_year)*0.1+start_year
35     plt.text(pos_x, 0.5, buf)
36
37 fig.suptitle('Global Temperature', fontsize=14)
38 plt.tight_layout()        (例えば重ならないように調整)
39 plt.subplots_adjust(top=0.90) (例えば重ならないように調整)
40
41 plt.savefig(outfile)
42 plt.show()
```

9-10 行目:　　開始、終了年度をここで定義するように変更した。プログラムの変更が容易に。
12 行目:　　入力ファイルの名前を定義した。
13 行目:　　結果の出力ファイルの名前を定義した。
34 行目:　　開始、終了年度の変更にも対応できるように、35 行目の X 座標を算出した。
(どうしてこのようになるのか、考えてみてください)
41 行目:　　結果をファイルに出力。

課題 2.2. 上述のプログラムでは、1951-2000 年までのデータを解析している。これを 1900-2000 年, 1850-2000 年と解析期間を変えて、気温の上昇速度を含めてグラフを描いてください。月ごとに上昇速度などの違いが見えるか? 期間の違いによって気温上昇速度が変わるのかどうか？を議論して下さい。

2.3. 解析例 2: 大気 CO_2 濃度増加量と SOI(南方振動係数)の関係

地球温暖化問題の一つの要因として、大気 CO_2 濃度の上昇が挙げられる。人為的に排出された(化石燃料の燃焼や土地利用変化など)CO_2 は、すべてが大気に残留するのではなく、その一部が海洋や陸域に吸収される。大気 CO_2 濃度を把握するために、世界中のいくつかの観測点にて長期観測がされている。いくつかの観測点のデータは、オンライン・無料で入手することができる。その中で最も長期の観測記録を持つ観測点に、ハワイのマウナロアの観測点が挙げられる(1958 年～現在)。



大気 CO_2 濃度の時系列変化

(http://cdiac.ess-dive.lbl.gov/trends/co2/graphics/SIOMLOINSITUTHRU2008.JPG より)

大気 CO_2 濃度の変化を追うことで、地球温暖化問題にも関連して、大気 CO_2 濃度を決定するメカニズムに対して重要な知見を与えることが可能である。例えば、大気 CO_2 濃度は季節変化していることが知られているが、季節変動を詳細に解析することにより、陸域植生の活動(例えば光合成がいつ始まるのか; 実は温暖化によって寒い地域の植生の活動時期が早まっていると言われている)の変化を推定することができる。また、大気 CO_2 濃度の増加速度(例えば、今年—昨年; 専門用語では CO_2 Growth Rate と呼ばれる)を計算したり、人為的に放出された CO_2 のうちの大気に残留する割合(専門用語で Airborne Fraction と呼ばれる)を求めることにより、陸や海洋の CO_2 吸収能力や、これらの変化がどのような原因によって引き起こされているのか考察することも可能である。

まずは、大気 CO_2 濃度の観測データから見てゆこう。今回は、前述した Mauna Loa の CO_2 濃度を利用することとする。前述の URL のより Mauna Loa の CO_2 濃度を取得しよう。

```
http://cdiac.ess-dive.lbl.gov/ftp/trends/co2/maunaloa.co2
```

終わりの 4 行を削除しよう。これで Python で読めるようになった。変更後のファイル名を `maunaloa.co2.python.txt` とする。

早速データを読み込んで、グラフを描いてみよう。(`numpy`, `matplotlib.pyplot` のインポートを忘れずに)

```
>>> din_co2 = np.loadtxt("maunaloa.co2.python.txt", skiprows=15, usecols=range(0, 14))
>>> din_co2.shape                #一応データサイズの確認(51x14 になってるか?)
>>> plt.plot(din_co2[:,0],din_co2[:,13])
```

[:,0]は年・[:,13]は年平均濃度に対応する(元のデータ参照)。

今回のグラフでは、一部、エラー値(-99.0)が入っているため、これを飛ばしてグラフを作成したい。とりあえずは、エラー値が含まれない 1970 年以降だけに着目してグラフを作成しよう。

```
>>> plt.plot(din_co2[:,0],din_co2[:,13])
>>> plt.xlim(1970,2010)
>>> plt.ylim(300,400)
>>> plt.xlabel("Year")
>>> plt.ylabel("CO2 Concentration (ppmv)")
>>> plt.show()
```

次に、各年の CO_2 濃度増加量(CO_2 Growth Rate)を求めてみよう。CO_2 Growth Rate は、1 年間に増加した大気 CO_2 量 (ppmv/year)で表され、次年の CO_2 濃度から現在の CO_2 濃度を引けばよい。すなわち、CO2(y+1)-CO2(y)を計算すればよい。

```
>>> din_co2g = din_co2[1:51,13]-din_co2[0:50,13]
```
(次年の CO_2 - 現在の CO_2)

```
>>> din_co2gyr = din_co2[0:50,0]
```
(対応する年を格納)

このように Python では配列の要素同士の四則演算は一気にできる。

これらを、一応、グラフにしておこう。

```
>>> plt.plot(din_co2gyr, din_co2g, "o-")
>>> plt.xlim(1970,2010)
>>> plt.ylim(0,4)
>>> plt.show()
```

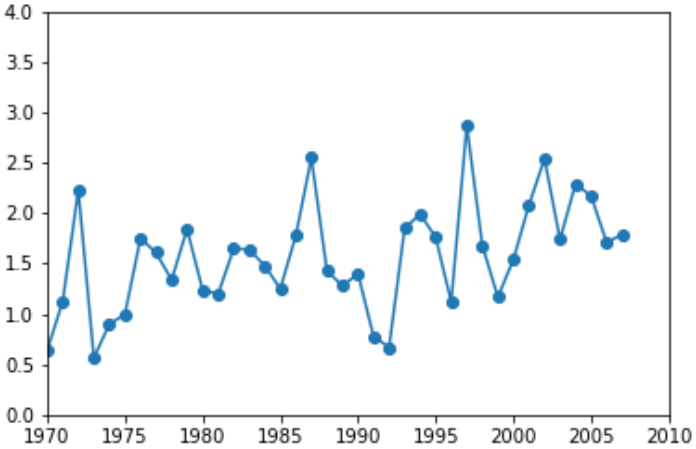


図 3. CO_2 Growth Rate の経年変動

課題 2.3：CO_2 Growth Rate が高い年・低い年、というのは、大気 CO_2 濃度を決定するうえで、どのような意味をもつだろうか？大気 CO_2 濃度の変化は、化石燃料などによる人為的排出、陸域での吸収、海洋での吸収の差によって決まると仮定せよ。
(地球の炭素循環については、
http://ja.wikipedia.org/wiki/%E7%82%AD%E7%B4%A0%E5%BE%AA%E7%92%B0 などを参考に。)
(下は Wikipedia における炭素循環の説明です)

季節変動を含めて表示させたい場合、元々2 次元だったデータを 1 次元に変換する必要がある。

```
>>> din_co2_month = din_co2[:,1:13]
```

1 は 1 月、12 は 12 月のデータに対応(0:年, 13:年平均の位置に相当 ― 1 から開始で 1 刻みで 13 の前で終了)。すなわち、余分な年と年平均を取ったデータを作成したことになる。
din_co2_month.shape として変数のサイズを確認しよう。

```
>>> din_co2_1dim = din_co2_month.reshape(-1)
```

として、1 次元にしたものを保存しておく。

```
>>> din_co2_1dim.shape
```

としてサイズを確認しておこう。(612x1 の double 型)
以下を実行してデータ確認をしよう。余分なデータ(-99.0)は入っているが、季節変動が描画できた。

```
>>> plt.plot(din_co2_1dim)
>>> plt.ylim(300,400)
>>> plt.show()
```

次に SOI(南方振動係数)データを見てみよう。こちらは、前述した Climate Research Unit のデータを用いよう。前述の URL より SOI を取得しよう。ファイル名はそのまま soi.dat とする。

http://www.cru.uea.ac.uk/cru/data/soi/soi.dat

このデータは特にコメントなどはないため、Python では何も変換せずに読めそうだ。
では、早速データを読み、解析をしよう。データのフォーマットはデータを取得する際の Web を参考にすること。

```
>>> din_soi = np.loadtxt("soi.dat", usecols=range(0, 14))
>>> plt.plot(din_soi[:,0],din_soi[:,13])
>>> plt.xlabel("Year")
>>> plt.ylabel("SOI")
>>> plt.show()                    #年平均の SOI をグラフに表示
```

例えば、もう少し近年に絞って見ると

```
>>> plt.plot(din_soi[:,0],din_soi[:,13])
>>> plt.xlabel("Year")
>>> plt.ylabel("SOI")
>>> plt.xlim(1980,2010)       #表示範囲を 1980 年から 2010 年に設定
>>> plt.ylim(-2,2)
>>> plt.show()
```

課題 2.4： 1980 年以降、エルニーニョ・ラニーニャと言われている年(Web などで調べること)について、SOI の値を調べてみよう。

最後に、CO_2 Growth Rate と、SOI の両者を比較してみよう。

```
>>> din_soi = np.loadtxt("soi.dat", usecols=range(0, 14))  #SOI データを読む
>>> din_buf = np.loadtxt("soi.dat", usecols=range(0, 14))  #すべて
>>> din_soiyr = din_buf[:,0]  #年のみ
>>> din_soi = din_buf[:,13]   #年平均のみ

>>> din_co2 = np.loadtxt("maunaloa.co2.python.txt", skiprows=15, usecols=range(0, 14))
          #CO2 データを読む
>>> din_co2gyr = din_co2[1:50,0]   #年のみ
>>> din_co2g = din_co2[2:51,13]-din_co2[1:50,13] #CO2 Growth Rate 計算

>>> idx_soi = np.where((din_soiyr>1970) & (din_soiyr<2008))
                SOI に関して、対応する年に対するインデックスを取得
```

```
>>> idx_co2g = np.where((din_co2gyr>1970) & (din_co2gyr<2008))
```
CO_2 Growth Rate に関して、対応する年に対するインデックスを取得
```
>>> plt.plot(din_soi[idx_soi], din_co2g[idx_co2g], "o")
```
必要なデータのみ(idx_soi,idx_co2g)を使って散布図を描く
```
>>> plt.xlabel("SOI")
>>> plt.ylabel("CO2 Growth Rate")
```
ラベルを付加
```
>>> plt.xlim(-1.5,1.5)
>>> plt.ylim(0.5,3.0)
```
表示範囲を設定
```
>>> plt.show()
```

これらの作業を一連のスクリプトにしてみよう。例えば、以下のようになる。ファイル名は、co2_soi.py とする。

```
01 #
02 # Python Sample: co2_soi.py
03 #
04
05 import matplotlib.pyplot as plt
06 import numpy as np
07
08 start_year = 1970
09 end_year = 2008
10
11 co2file = "maunaloa.co2.python.txt"
12 soifile = "soi.dat"
13
14 din_co2 = np.loadtxt(co2file, skiprows=15, usecols=range(0, 14))
15 din_co2g = din_co2[2:51,13]-din_co2[1:50,13]
16 din_co2gyr = din_co2[1:50,0]
17
18 din_buf = np.loadtxt(soifile, usecols=range(0, 14))
19 din_soi = din_buf[:,13] #年平均のみ
20 din_soiyr = din_buf[:,0] #年のみ
21
22 idx_soi = np.where((din_soiyr>start_year) & (din_soiyr<end_year))
23 idx_co2g = np.where((din_co2gyr>start_year) & (din_co2gyr<end_year))
24
25 plt.plot(din_soi[idx_soi], din_co2g[idx_co2g], "o")
```

```
26  plt.xlabel("SOI")
27  plt.ylabel("CO2 Growth Rate")
28  plt.xlim(-1.5,1.5)
29  plt.ylim(0.5,3.0)
30  plt.show()
```

15,16 行目: CO_2 growth rate とそれに対応する年を求める
22,23 行目: start_year, end_year の範囲内にあるデータのインデックスを取得
25 行目: 16, 17 行目で取得した位置にある配列だけを選択してプロットする

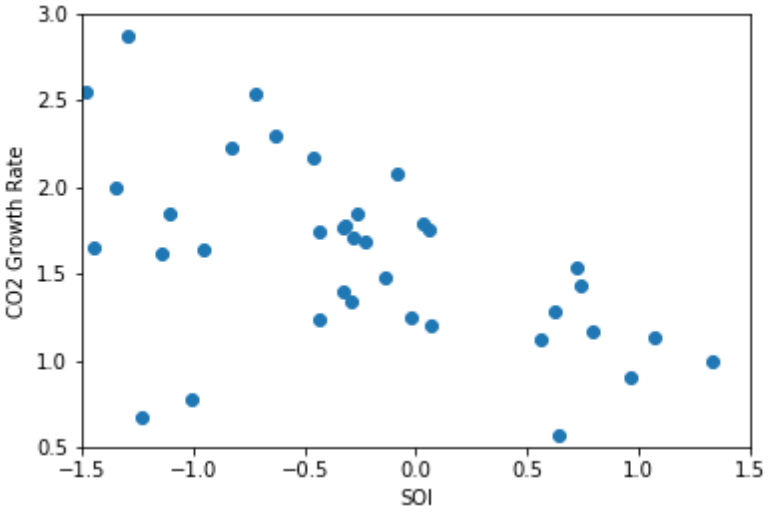


図 4. CO_2 Growth Rate と SOI の関連

課題 2.5. CO_2 Growth Rate と SOI の関連(エルニーニョ/ラニーニャとの関連)を議論せよ。また、SOI<-0.8, CO_2 Growth Rate<1 である 2 点はどのような年であるか調べよ。

課題 2.6. CO_2 Growth Rate と SOI の散布図に対して、直線回帰を行い、直線とその回帰式を入れなさい。つぎに、また、SOI<-0.8, CO_2 Growth Rate<1 である 2 点を除いた回帰を行い、直線とその回帰式を入れなさい。

3. Python によるバイナリデータの解析

3.1.はじめに

本章では、いよいよバイナリデータを扱う。ここでは、2 次元(画像としても閲覧可能なもの)のバイナリデータを対象とする。バイナリデータは、テキストデータと異なり、直接データを目で見ることは困難である(注: 直接見る方法もあるが、今回は割愛)。データを見るには、一般的には、少し高度な画像処理ソフトウェア(Adobe Photoshop や GIMP 等)、さらに特殊な科学データ解析用のソフトウェア(ENVI, Arc-GIS)、またはプログラミング(C, FORTRAN, Python, R, MATLAB 等)などが利用できる。

まず、科学データ解析用のソフトウェアの利用に関しては、それ自体が高価であったりして、本当の専門家でないと扱う(購入する)のが困難である。次に、少し高度な画像処理ソフトウェア(Adobe Photoshop や GIMP 等)を用いる方法では、読めるデータ形式(バイナリ形式)が限られている。例えば、Photoshop では、符号なし整数(1 バイト)と符号なし整数(2 バイト)の 2 種類のみである。従って、実数データや、符号付きデータについては読むことが困難である。また、GUI を介した操作を行うソフトの利用自体、手作業が発生する。そのため、データ数が増えてきたときに、非常に煩雑である。従って、バイナリデータに関しても、プログラミングなどによる扱い方を学ぶことは有意義である。

但し、従来のコンピュータ言語である C や FORTRAN に関しては、画面に表示させるための命令等の扱いが大変複雑である(できないことはないが結構大変)。ここでも、Python, R, MATLAB(Octave)などのデータ処理＋データ表示の両方に利点を持つソフトウェアが便利である。

本章では、まず、画像処理ソフトウェア(GIMP, Adobe Photoshop)を用いて、符号なし整数(1 バイト)のデータを読み、表示させてみる。そのデータを通して、今回のデータのフォーマット・地図投影・緯度経度と画像中の座標との対応、などについて学ぶ。次に、それらの画像を Python で表示するためのプログラミングについて学ぶ。さらに、一般の画像処理ソフトウェアでは読み込み・表示が困難な実数データを含むデータに関して、Python を用いて、画像読み込み・表示・解析を行う。

3.2. 2 次元データのデータ格納方法　～ 画像処理ソフトでの解析を通して ～

今回用いるデータ(3.2, 3.3 章共通)は、

```
MOD12Q1.ASIA.QDEG.2001.byt
MOD12Q1.ASIA.QDEG.2001.ppm
```

というファイルである。本章で必要なファイルは、以下においたので取得すること。

https://www.dropbox.com/s/tkmv9ydrnr7dskb/MOD12Q1.ASIA.QDEG.2001.byt
https://www.dropbox.com/s/mys7j5a5khxqdc8/MOD12Q1.ASIA.QDEG.2001.ppm

自分の作業用フォルダに保存してください。

このデータは、米国航空宇宙局(NASA; National Aeronautics and Space Administration)の地球観測衛星 Terra に搭載されている中分解能撮像分光放射計(MODIS; Moderate Resolution Imaging Spectroradiometer)というセンサで観測された、様々な波長帯における地表面反射率を組み合わせて作成された「土地被覆分類」データである。Terra 衛星は 1999 年 12 月 18 日に打ち上げられ、現在まで 10 年以上のデータが蓄積されている。MODIS センサは、0.4μm～14.4μm の波長帯の間に 36 の観測波長帯を持つ。これらのデータを用いて作成されたデータである。今回は、図 1 に示す通り、アジア域広域に渡って合成処理をしたものである(NASA のサイトよりダウンロードしたものを当研究室で合成したもの)。カバーする範囲としては、(60E, 80N) – (180E, 10S)である。また、このデータは、経度(横)方向に 480 画素、緯度(縦)方向に 360 画素で構成されている。等緯度経度座標でデータが格納されている。各画素は、符号なし整数(1 バイト)(0-255)でデータがバイナリ形式で格納されている。画像の左上から横方向へそして右下への方向にデータが並んでいる(ファイル上では画素が並んでいるだけ)。

試しに、データのサイズ(480x360)と形式(符号なし整数 1 バイト)より、データの大きさは、

480 x 360 x 1 = 172800 バイト

となり、Windows 上で表示されるファイルサイズと同一になる(ファイルを右クリック→プロパティ)。

データの中身としては、各画素毎に土地被覆区分の番号が格納されている(表 1)。例えば、水域に対応する場所では 0、耕作地なら 12 などと、符号なし整数(1 バイト)で格納されている。

次に、1 画素辺りの緯度・経度を求めてみよう。経度方向には、480 画素あり、これが 90 度分に対応するので、

480 ピクセル / 120 度 = 4 (ピクセル/度)　(経度 1 度辺り 4 ピクセル)　または

1 度 / 4 ピクセル = 15 分　(1 ピクセル辺り 15 分 (=0.25 度))

となる。緯度方向には、360 画素あり、これが 90 度分に対応するので、

360 ピクセル / 90 度 = 4 (ピクセル/度)　(経度 1 度辺り 4 ピクセル)　または

1 度 / 4 ピクセル = 15 分　(1 ピクセル辺り 15 分 (=0.25 度))

従って、1 ピクセルは、経度方向 0.25 度・緯度方向 0.25 度の大きさを持つことが分かった。

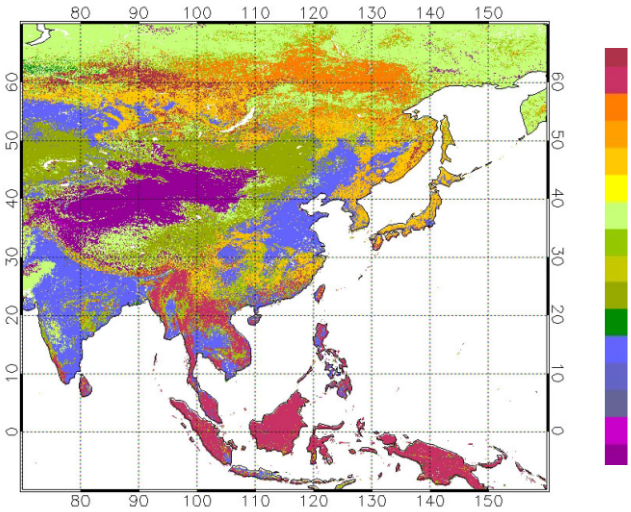


表 1. 土地被覆図のデジタル値

土地被覆の形態	デジタル値
水域	0
常緑針葉樹林	1
常緑広葉樹林	2
落葉針葉樹林	3
落葉広葉樹林	4
混合林	5
密な低木	6
疎な低木	7
サバンナ (Woody)	8
サバンナ	9
草地	10
湿地	11
耕作地	12
都市	13
耕作地・自然植生の混合	14
雪氷	15
裸地	16

図 1.　アジアの土地被覆図(Terra/MODIS データより構築。図表は大阪府立大・植山氏より借用)

さて、次は、データを画像として表示させ、データのイメージをつかむことを目標としたい。広く使われている高機能の画像処理ソフトウェアとして、アドビ社の Photoshop(商用) や GIMP(フリーソフトウェア)などがある。今回は GIMP を用いてデータを表示させる。

まずは、GIMP を起動させよう。(ない場合はインストールしよう。もしくは自分の好きな画像処理ソフトでもできる場合があるかもしれない。)

スタート　→　すべてのプログラム　→　GIMP　→　GIMP2

GIMP では、このままでは、“MOD12Q1.ASIA.QDEG.2001.byt”ファイルは読めない。このファイルは各ピクセルのデータが格納されているが、データのサイズ(縦横のピクセル数)やデータ形式(Binary? ASCII?等)の情報がないためである。GIMP などの画像処理ソフトでデータ(画像)を表示させるためには、必要な情報を持ったファイルを作成する必要がある。

本実習では､あらかじめ GIMP で読めるようにヘッダ情報を付加したファイルを利用することとする。画像ファイルの形式としては、最も簡単なフォーマットの一つである ppm 形式を用いた。
ファイル名: "MOD12Q1.ASIA.QDEG.2001.ppm"。ヘッダ情報といっても、それほど難しいものではなく、MOD12Q1.ASIA.QDEG.2001.byt　ファイルの頭に、以下の 4 行を付加したものである。

```
P5
# EnvMdl Land Cover (480x360 uint8)
480 360
255
```

この情報の説明としては、http://pen.envr.tsukuba.ac.jp/~nishida/lecture/image_anal/format.html などを参照:。ppm 形式の詳細については、ウィキペディアなどを参照すること。

1 行目： 画像の種類
- P1 ・・・ 白黒２値画像　（ASCII 形式）
- P2 ・・・ モノクロ濃淡画像　（ASCII 形式）
- P3 ・・・ RGB カラー画像　（ASCII 形式）
- P4 ・・・ 白黒２値画像　（Binary 形式）
- P5 ・・・ モノクロ濃淡画像　（Binary 形式）
- P6 ・・・ RGB カラー画像　（Binary 形式）

2 行目： #からはじまる, コメント行。
3 行目： 画像の横ピクセル数と縦ピクセル数
4 行目： データの最大値

例えば、

MOD12Q1.ASIA.QDEG.2001.ppm

`MOD12Q1.ASIA.QDEG.2001.byt`

の 2 つのファイルでは、前者(.ppm)が 39 バイト大きい。この 39 バイトが画像データの情報を表すヘッダ部分に当たる。Windows 上では、例えば、`MOD12Q1.ASIA.QDEG.2001.ppm` をワードパッドなどで開くと、ヘッダ部分がテキスト形式で格納されていることが分かる。

次は、GIMP でファイルを読む。ファイル→開く、とするとファイル選択ウィンドゥが表示される。ファイルを選択し、開く。すると以下のウィンドゥが現れる(図 2)。

ただし、このデータは、256 階調のうち、ごく小さい値しか格納されていないので、ほぼ真っ黒な画像が表示される。これにストレッチ処理を行う(メニューより: 色→レベル ; 図 3)。結果の一例を図 4 に示す。

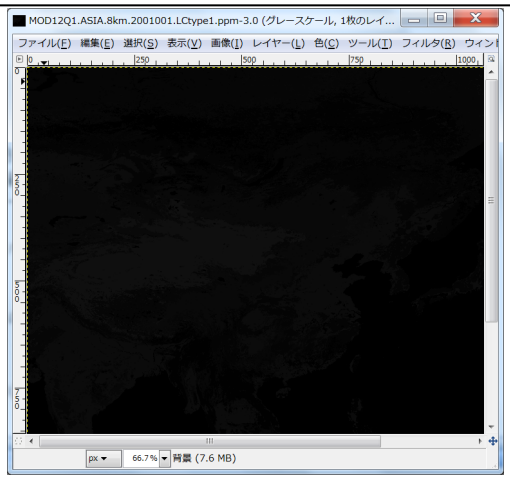


図 2. GIMP で該当ファイルを開いた結果

図 3.　ストレッチ処理の画面。最大値を小さくすればよい(図の矢印)。

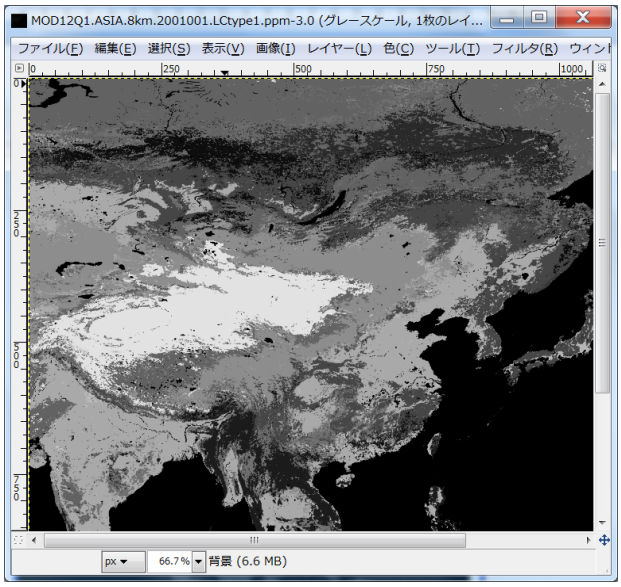


図 4. ストレッチ処理後の画像

ここで、今回用いた画像(データ)の形式について説明する(図 5)。今回のデータは、横方向に 480 ピクセル、縦方向に 360 ピクセルで構成される。データを画像で見る際には、上記などのようにして 2 次元データとして見ることができるが、もとのファイルにはそのような情報は含まれていない。データの先頭から終わりまで、各点のデータ値が順に格納されているだけである(従って、画像処理ソフトで表示させる際には、画像サイズ(縦横のピクセル数)を入力する必要がある)。一般に、画像の左上を先頭に、まずは右方向へ、そして下方向へ値が格納されており、画像の右下が一番最後になる。画像の左上を(0,0)としたとき(Python では配列の添字は 0 から始まるので)、点(X,Y)は、データの先頭から数えて、Y×480 + X + 1 番目の位置にあることが分かる。自分の手で確かめてみるとよい。この概念は必ず理解してほしい。

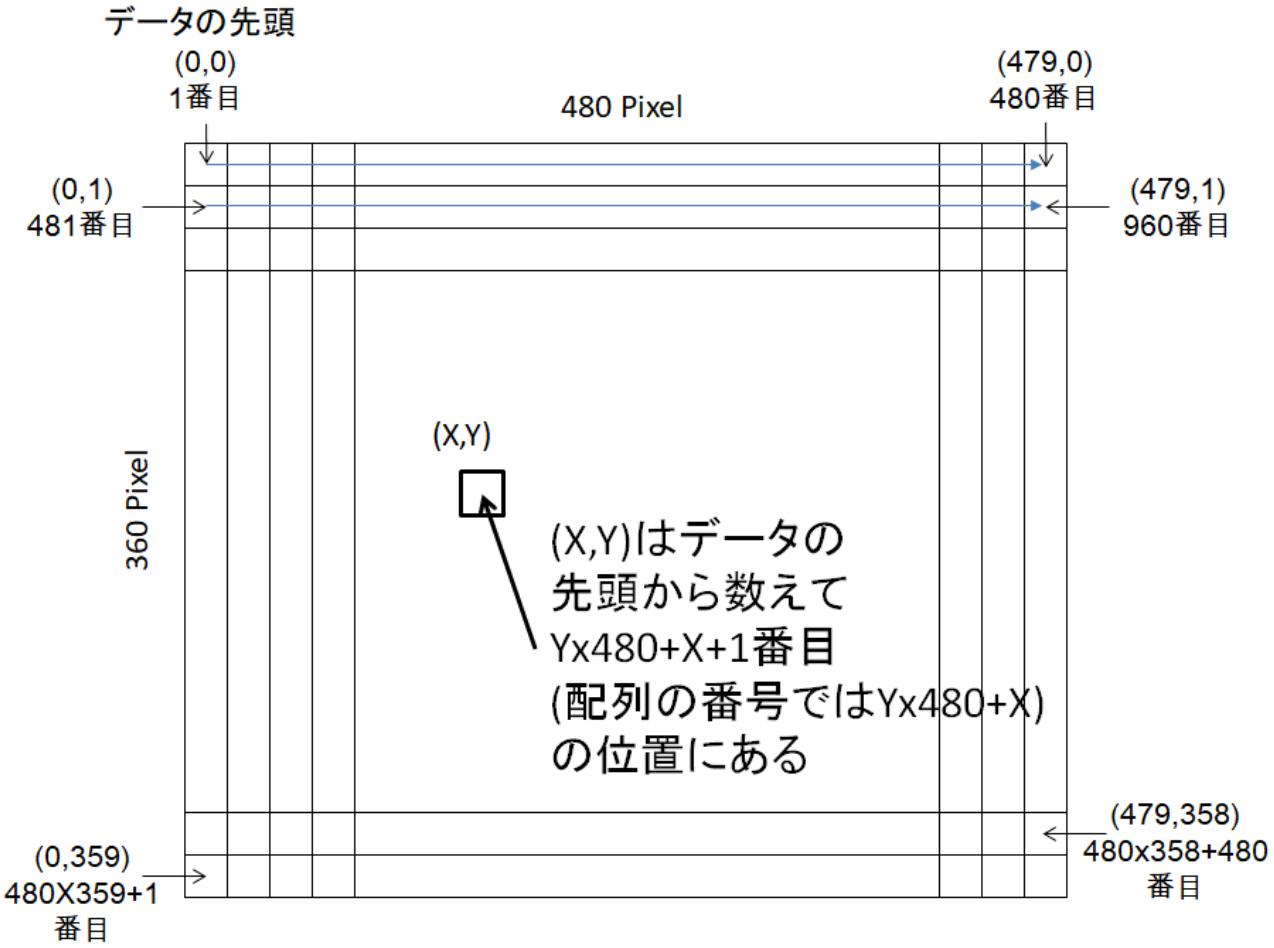


図 5. 今回用いた 2 次元データの概要。左上を(0,0)とし各座標が先頭から数えて何番目に来るかを示した。

次にデータの各点と、緯度経度の対応を説明する(図 6)。前述のように 1 点は、0.25 度の範囲をカバーする。今回のデータの範囲は、(60E, 80N) - (180E, 10S)の範囲をカバーしている。例えば、(0,0)の中心点での緯度経度は、(60.125E, 79.875N)である。すなわち、(60E, 80N)から 0.125 (=1/8) 度だけずらした緯度経度となる。言い換えると、(0,0)の点は、(60E, 80N) - (60+0.25E, 80-0.25N)の範囲をカバーしていることになる。前と同様に、各点(X,Y)の緯度経度を求めることができる。逆に、緯度経度から、データ中での座標を計算することも可能である。

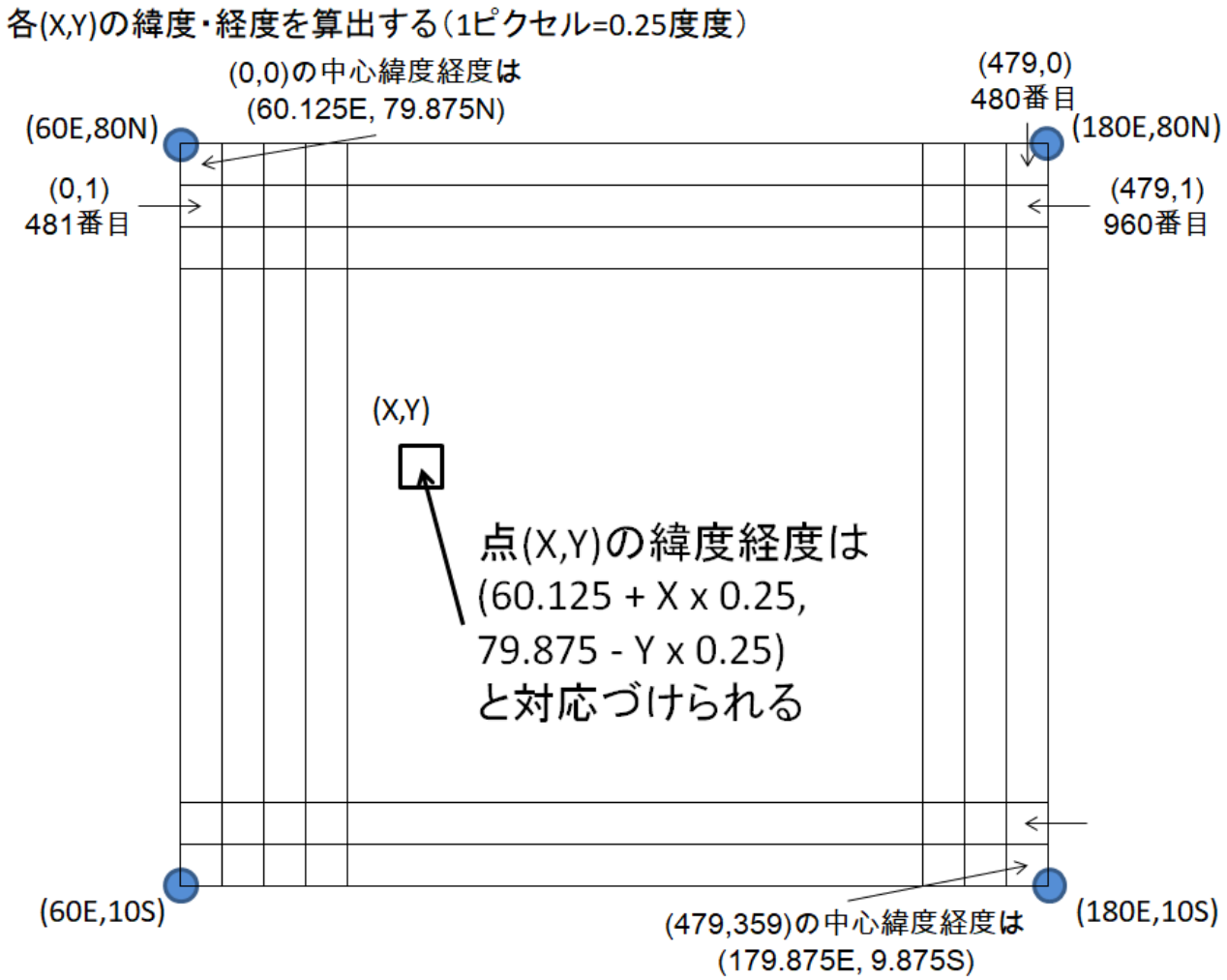


図 6. 今回用いた 2 次元データと各座標の中心点と緯度経度との関連。座標(X,Y)の中心緯度経度を求めた。

3.3. Python を用いたバイナリデータの解析

(1) 土地被覆分類データの解析

Python を用いた場合(一般的なプログラミングの場合も同様)のバイナリデータの扱いについては、

⓪　下準備
①　ファイルのオープン
②　ファイルからデータの読み込み
③　ファイルのクローズ
④　データの解析

のような手順となる。③④は必要に応じて順序が入れ替わることも多い。早速、①－④の手順に従って、先ほどの土地被覆分類図を表示させてみよう。以下は、今回の実習で最も重要な部分であるので、確実に理解すること。

⓪　下準備

必要なモジュールを import しよう。

```
>>> import numpy as np                      #numpy
>>> import matplotlib.pyplot as plt         #matplotlib
>>> import matplotlib.cm as cm              #カラーマップの読み込み（カラーマップを変えないので今回はなくてもよい）
```

①　ファイルのオープン

データが格納されているファイルを開く

```
>>> fin=open('MOD12Q1.ASIA.QDEG.2001.byt','rb')
```

ファイルの開き方の基本：

```
fin=open(infile, mode)
```

fin: ファイル ID
infile: 入力ファイル
mode: ファイル入出力のモード　(r: テキスト読み込み, rb: バイナリ読み込み
w: テキスト書き込み, wb: バイナリ書き込み)

②　ファイルの読み込み

```
>>> dbuf=np.fromfile(fin,dtype=np.uint8,count=-1)
```

dbuf: データ格納用メモリ(読んだデータが格納される)　1 次元配列として確保される
fin: ファイル ID (②で作ったもの)
dtype: データのタイプ　例:　np.int8: 符号付 1 バイト整数
np.uint8: 符号なし 1 バイト整数
np.int16: 符号付 2 バイト整数

np.uint16: 符号なし 2 バイト整数
np.int32: 符号付 4 バイト整数
np.uint32: 符号なし 4 バイト整数
np.float32: 4 バイト小数
np.float64: 8 バイト小数　など

count:　読み込むデータの数　　(-1 ならばファイルの終わりまで)

③　ファイルのクローズ

```
>>> fin.close()
```

fin: ファイル ID

④　データの加工・解析

```
>>> dlc=dbuf.reshape(360,480)
```

先ほどは 1 次元配列として読み込んだ。ここでは 360 行 480 列の 2 次元データに変換

```
>>> plt.imshow(dlc,clim=(0.0,16.0))
```

まずは 2 次元画像として正しく読めているか確認しよう。clim=(min, max)は、画像表示の際の引き延ばしの際の最小・最大値

imshow はデータを画像として表示する。

```
>>> plt.show()
```

例によって画面表示

試しに領域の中のどこか一部を取ってみよう。

```
>>> plt.imshow(dlc[0:99,0:479],clim=(0.0,16.0))
```

まずは 2 次元画像として正しく読めているか確認しよう。clim=(min, max)は、画像表示の際の引き延ばしの際の最小・最大値

imshow はデータを画像として表示する。

```
>>> plt.show()
```

例によって画面表示

やる前にどの地域か予想してみよう。

```
>>> plt.imshow(dlc[120:239,240:359],clim=(0.0,16.0))
```

まずは 2 次元画像として正しく読めているか確認しよう。clim=(min, max)は、画像表示の際の引き延ばしの際の最小・最大値

imshow はデータを画像として表示する。

```
>>> plt.show()
```

例によって画面表示

iPython, Anaconda (Jupiter Note)のような環境であれば、ファイルがきちんと読めたかどうか確認も可能である。メモリに一時保存されている変数の一覧を確認できる。(Python 本体のみよりも圧倒的に便利)

```
>>> whos
```

次に簡単なデータ解析を行う。各土地被覆区分が何点づつ存在するかを求めてみよう。
例えば、土地利用区分 1 の常緑針葉樹林のピクセル数を数えてみる。

```
>>> idx=np.where(dlc==1)   #dlc=1 となる点の配列番号の一覧を idx に格納する。
>>> whos                   #idx がどんな型・次元になっているか調べよう(タプル型 n=2)
                           #タプル(tuple)型とは何か調べてみよう
>>> idx[0]                 #タプル型の[0]
>>> idx[1]                 #タプル型の[1]
>>> idx[0].size            #idx[0]のサイズを取得 (すなわち dlc==1 となった要素数)

>>> y=idx[0].size          #メモリに確保するため y という変数に入れてみよう
>>> y                      #y の中身を確認しよう
```

以上より、土地被覆項目 1 のピクセル数を得ることができた。

課題 3.1. 同様に din=2, 3, 4, 5, 10, 12, 14 のピクセル数を求めてみよう。
次に、FOR ループを用いたプログラムを書いて 1-16 すべての項目のピクセル数を一気に求めよう。

(2) NDVI データの解析

同様に、MODIS 衛星で観測された NDVI(正規化植生指数; Normalized Difference Vegetation Index)データを見てみよう。NDVI は、可視域(赤)バンド・近赤外域バンドを組み合わせ、植物量や活性度に関連のあるとされている。以下のように計算される。

$$NDVI = \frac{NIR - RED}{NIR + RED} \qquad (1)$$

ここで、NIR は近赤外バンドの地表面反射率、RED は赤色バンドの地表面反射率を表す。NDVI は-1 から 1 までの値を取るために、今回のデータでは、浮動小数点実数(4 バイト)の形式で格納されている。地図投影法は、土地被覆分類データと同じである。また、水域は、-9999.0 が格納されている。

まずは、データの確認。以下の URL よりデータを取得し、解凍すること。

URL: https://www.dropbox.com/s/4jkvuxjgyl3vhoq/MOD13A2.C5.tar.gz

ファイルの中身は、Terra 衛星搭載 MODIS センサより観測された NDVI であり、

MOD13A2.C5.NDVI.YYYY.Asia.QDEG.bsq.flt （YYYY 年）

がすべて NDVI データである(2000-2015 年 1-12 月のデータがある)。

これらの一つのファイルのサイズ・フォーマットは、

サイズ:

480 x 360 x 12 (ヶ月分) x 4 (4byte データなので) = 8,294,400 バイト

フォーマット:

(480x360) (1 月)
(480x360) (2 月)
(3-11 月が順に格納)
(480x360) (12 月)

の順にデータが格納されている。

次に、これを読み、表示させてみよう(植生が最も活動的そうな 7 月を選択)。

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> fin=open('MOD13A2.C5.NDVI.2000.Asia.QDEG.bsq.flt','rb')
>>> dbuf= np.fromfile(fin,dtype=np.float32)
```

```
>>> dndvi_mon=dbuf.reshape(12,360,480)
>>> dndvi_out=dndvi_mon[6,:,:]
      https://matplotlib.org/users/colormaps.html #例えば 7 月を選択
>>> plt.imshow(dndvi_out,clim=(0.0,1.0))
>>> plt.show()
```

色付けについては、python に含まれるカラーマップ(数値の大小によって色を割り当てる)を利用してみて、少し見やすいようにアレンジができる。

```
>>> import matplotlib.cm as cm
>>> plt.imshow(dndvi_out,clim=(0.0,1.0),cmap=cm.jet)
>>> plt.show()
```

カラーマップの種類は以下に記載： viridis, plasma などがカラーマップの名前に相当
https://matplotlib.org/users/colormaps.html

これまでの部分をスクリプトにまとめて実行してみよう。多少後々での変更が容易なように変更。
ファイル名は、disp_ndvi.py として保存しよう。

```
##スクリプト開始##
# NDVI data analysis
# Format: 480 pixel x 360 line x 12 months (float)
# Missing Value: -9999.0

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

pixel=480
line=360
month=12
imon=6

indir='./'
infile='MOD13A2.C5.NDVI.2000.Asia.QDEG.bsq.flt'

fin=open(indir+infile,'rb')
dbuf=np.fromfile(fin,dtype=np.float32)
```

```
dndvi_mon=dbuf.reshape(month,line,pixel)
dndvi_out=dndvi_mon[imon,:,:]
plt.imshow(dndvi_out,clim=(-0.2,1.0),cmap=cm.jet)

plt.colorbar()
plt.axis('off')
plt.show()
##スクリプト終了##
```

また、例えば、
`dndvi_out=dndvi_mon[imon:,:]`
の行を、
`dndvi_out=np.mean(dndvi_mon,0)`
に変更すると、年平均値となる。（やってみよう。平均を求めるのは非常に楽！）

次の段階として、もう少しきれいにデータを出力してみよう。例えば、これまでの結果では、海域も青く描かれており、NDVI が低い地域との区別がつきにくい。この際、海域を白くしよう。これは、技術的な問題だけであるので、例としてのスクリプトを示す。その他カラーバーを表示などといくつか飾ってみた。このスクリプトを作成し、実行すること。ファイル名を `disp_ndvi_clean.py` とする。

```
##スクリプト開始##
# NDVI data analysis
# Format: 480 pixel x 360 line x 12 months (float)
# Missing Value: -9999.0

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

pixel=480
line=360
month=12

indir='./'
infile='MOD13A2.C5.NDVI.2000.Asia.QDEG.bsq.flt'

fin=open(indir+infile,'rb')
```

```
dbuf=np.fromfile(fin,dtype=np.float32)
dndvi_mon=dbuf.reshape(month,line,pixel)

dndvi_final=np.mean(dndvi_mon,0)  # Calculate Average (direction: dimension 1)

fig=plt.imshow(dndvi_final,clim=(0.0,1.0),cmap=cm.jet)
fig.cmap.set_under('w')
#clb=plt.colorbar(extend='min',orientation="horizontal")
clb=plt.colorbar(extend='min')
clb.ax.set_title('NDVI')
plt.title('MODIS NDVI (mean)')
plt.axis('off')
plt.show()
##スクリプト終了##
```

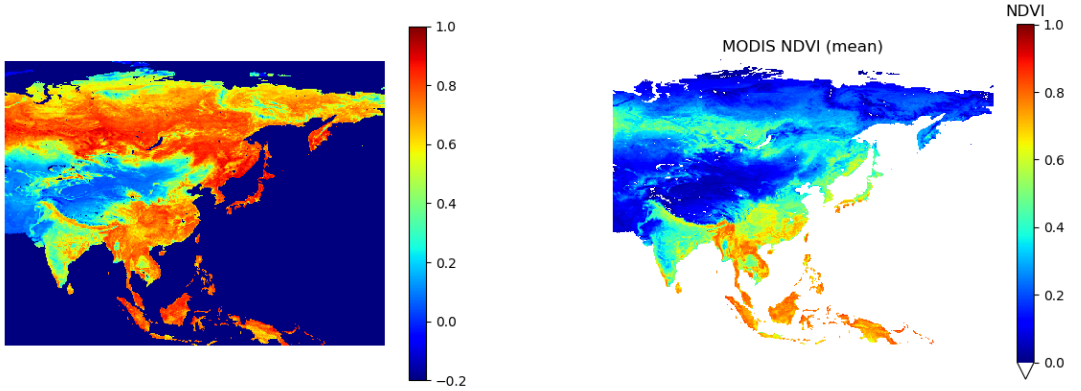


図 9. (a) 元の画像(disp_ndvi.py 実行時)　(b) 海を白くした場合(disp_ndvi_clean.py 実行時)

次に、グラフの時と同様に、複数の画像を同一画面に表示させてみよう。for ループを用いて、subplot()を利用すればよい。ファイル名を disp_ndvi_multi.py とする。

```
##スクリプト開始##
# NDVI data analysis
# Format: 480 pixel x 360 line x 12 months (float)
# Missing Value: -9999.0

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

```
pixel=480
line=360
month=12

indir='./'
infile='MOD13A2.C5.NDVI.2000.Asia.QDEG.bsq.flt'

fin=open(indir+infile,'rb')
dbuf=np.fromfile(fin,dtype=np.float32)
dndvi_mon=dbuf.reshape(month,line,pixel)

#plt.figure(figsize=(15,12))

for imon in range(0,4):

        dndvi_out=dndvi_mon[imon*3+1,:,:]
        idx_minus=np.where((dndvi_out>-990.0) & (dndvi_out<0.0))
        dndvi_out[idx_minus]=0.0

        plt.subplot(2,2,imon+1)
        fig=plt.imshow(dndvi_out,clim=(0.0,1.0),cmap=cm.jet)
        fig.cmap.set_under('w')
        clb=plt.colorbar(extend='min')
        clb.ax.set_title('NDVI')
        stitle="MODIS NDVI: Month "+"{0:02d}".format(imon*3+1)
        plt.title(stitle)
        plt.axis('off')

plt.show()
##スクリプト終了##
```

課題 3.2.　同様に 1-12 月までの NDVI を同一画面に出力させてみよう。

(3) 応用

最後に、これまでのまとめとして、いくつかサンプルプログラム・解析を紹介する。

① データ中の特定の緯度経度範囲のデータをピックアップ。ファイル名:disp_ndvi_pickup.py。

```
##スクリプト開始##
# NDVI data analysis
# Format: 480 pixel x 360 line x 12 months (float)
# Missing Value: -9999.0

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

pixel=480
line=360
month=12

slon=125.0  # Start Longitude
elon=150.0  # End   Longitude
slat=50.0   # Start Latitude
elat=30.0   # End   Latitude

indir='./'
infile='MOD13A2.C5.NDVI.2000.Asia.QDEG.bsq.flt'

# Calculate Corresponding Pixel (math.floor: truncation)
# You can also use np.floor() - however it returns double
spixel=math.floor((slon-60.0)*4.0)
sline =math.floor((80.0-slat)*4.0)
epixel=math.floor((elon-60.0)*4.0)
eline =math.floor((80.0-elat)*4.0)

fin=open(indir+infile,'rb')
dbuf=np.fromfile(fin,dtype=np.float32)
dndvi_mon=dbuf.reshape(month,line,pixel)
```

```
plt.figure(figsize=(15,12))

for imon in range(0,4):

        dndvi_out=dndvi_mon[imon*3,sline:eline,spixel:epixel]
        idx_minus=np.where((dndvi_out>-990.0) & (dndvi_out<0.0))
        dndvi_out[idx_minus]=0.0

        plt.subplot(2,2,imon+1)
        fig=plt.imshow(dndvi_out,clim=(0.0,1.0),cmap=cm.jet)
        fig.cmap.set_under('w')
        clb=plt.colorbar(extend='min')
        clb.ax.set_title('NDVI')
        stitle="MODIS NDVI: Month "+"{0:02d}".format(imon*3+1)
        plt.title(stitle)
        plt.axis('off')

plt.show()
##スクリプト終了##

import math as math          math モジュールを読み込み(切り捨て作業で必要)
math.floor は切り捨て
```

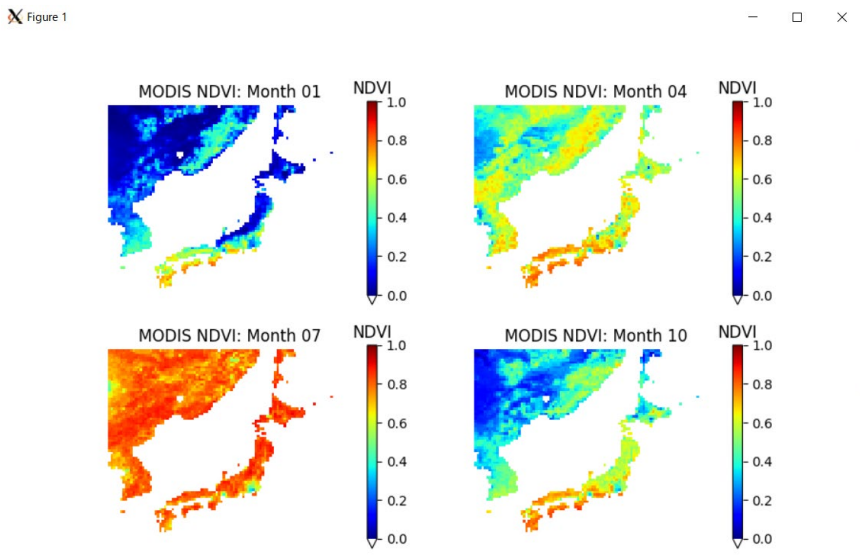


図 10.　結果の一例 (上のスクリプトの場合)

② 2000-2015 年の各年の年平均 NDVI の偏差

ファイル名、disp_ndvi_anomaly.py として以下を作成してみよう。

##スクリプト開始##

```
# NDVI data analysis
# Format: 480 pixel x 360 line x 12 months (float)
# Missing Value: -9999.0

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

pixel=480
line=360
month=12
nyear=16

indir='./'
infile='MOD13A2.C5.NDVI.2000.Asia.QDEG.bsq.flt'

# Prepare annual NDVI from 2000 to 2015
dndvi_ann=np.zeros((nyear,line,pixel))
dndvi_anom=np.zeros((nyear,line,pixel))

for iyear in range(2000,2016):

       infile=indir+"MOD13A2.C5.NDVI."+"{0:04d}".format(iyear)+".Asia.QDEG.bs
q.flt"
       fin=open(infile,'rb')
       dbuf=np.fromfile(fin,dtype=np.float32)
       dndvi_buf=dbuf.reshape(month,line,pixel)
       dndvi_ann[iyear-2000,:,:]=np.mean(dndvi_buf,0)

# Calculate NDVI average (2000-2015 average)

dndvi_ave=np.mean(dndvi_ann,0)
idx_err=np.where(dndvi_ave<-990.0)

# Calculate NDVI anomaly in each year

for iyear in range(2000,2016):

       dndvi_anom[iyear-2000,:,:]=dndvi_ann[iyear-2000,:,:]-dndvi_ave

# Display NDVI anomaly

for iyear in range(2000,2016):

       plt.subplot(4,4,iyear-2000+1)
       dbuf=dndvi_anom[iyear-2000,:,:]
       dbuf[idx_err]=-9999.0

       fig=plt.imshow(dbuf,clim=(-0.1,0.1),cmap=cm.jet)
       fig.cmap.set_under('w')
```

```
        clb=plt.colorbar(extend='min')
        #clb.ax.set_title('NDVI')
        stitle='NDVI anomaly (YR:'+"{0:04d}".format(iyear)+')'
        plt.title(stitle)
        plt.axis('off')

plt.show()
##スクリプト終了##
```

図のサイズが大きすぎる・小さすぎるなどの場合には、2.2 章でやった

```
fig = plt.figure(figsize=(15,12))
plt.tight_layout()
```

などをして調整をしてみよう。

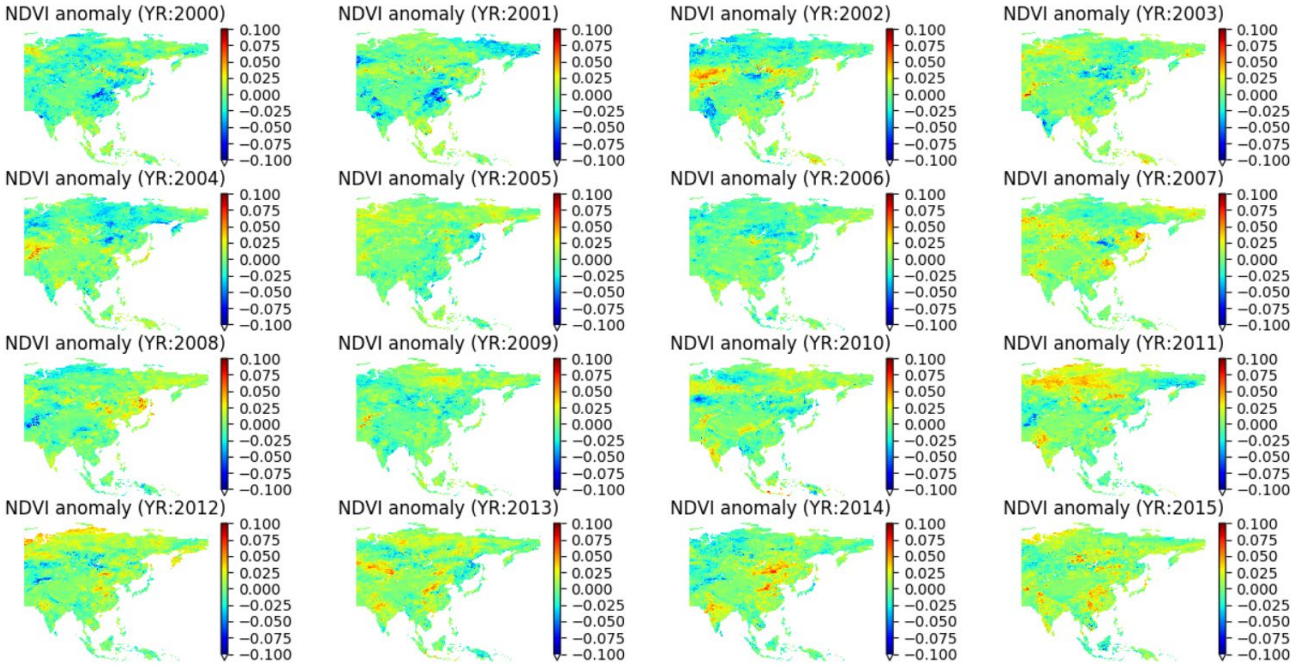


図 11. 実行結果

課題 3.3. 上で作成した disp_ndvi_anomaly.py について、特定の地域を取得できるように変更しよう。

課題 3.4. さらに、特定の地域についての NDVI 平均値を算出し、時系列のグラフを出すようなスクリプトを作成しよう。

③ 2000-2015 年の各年の年平均 NDVI の増減傾向

最後の例として、各ピクセル毎に計算をする例として、NDVI の増減傾向(線形回帰をした傾き)を計算し、表示するスクリプトを作成してみよう。

まずは、これまで利用した月別の NDVI のデータだが、データ量が大きいので、処理を簡略化するために、月別→年間平均、に変換する。

```
# NDVI data conversion (monthly -> yearly)
# Input: 480 pixel x 360 line x 12 months (float)
# Output: 480 pixel x 360 line x 1 (float)
# Missing Value: -9999.0

import numpy as np

pixel=480
line=360
month=12

indir='./'

for iyear in range(2000,2016):

        infile=indir+"MOD13A2.C5.NDVI."+"{0:04d}".format(iyear)+".Asia.QDEG.bsq.flt"

        fin=open(infile,'rb')
        dbuf=np.fromfile(fin,dtype=np.float32)
        fin.close

        dndvi_buf=dbuf.reshape(month,line,pixel)
        dndvi_ann=np.mean(dndvi_buf,0)

        outfile=indir+"MOD13A2.C5.NDVI."+"{0:04d}".format(iyear)+".Asia.QDEG.ann.flt"
        fout=open(outfile,'wb')
        dndvi_ann.tofile(fout)
        fout.close()
```

ファイルができたかどうか、確認してみよう。

課題 3.5. 上で構築したデータを図示してみよう。Python での描画がベスト。他の方法でもよしとする。

次に、上記のデータを用いて、各点毎に線形回帰を行い、傾き（増加・減少率）と傾きの統計的有意性(傾きを 0 とするとした帰無仮説を棄却できない確率)を計算してみよう。これにより、2000-2015 年の期間においてどこでどの程度植生が増減したかを推測することが可能になる。傾きの算出については、2.2 章で `np.polyfit()`を利用したが、これでは有意確率(p 値)を算出できない。そこで、`scipy` に含まれ

る linregress を使おう。必要に応じて、scipy をインストールしよう。サンプルプログラムを以下に提示する。ちょっと時間がかかるので、進捗が分かるように表示を加えた。

```
# NDVI data trend analysis
# Format: 480 pixel x 360 line (annual) (float)
# Missing Value: -9999.0

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy.stats import linregress

pixel=480
line=360
im_min=-0.01
im_max=0.01
nyear=16

indir='./'

# Prepare annual NDVI from 2000 to 2015
dndvi_ann=np.zeros((nyear,line,pixel))
dndvi_trend=np.zeros((line,pixel))
dndvi_pvalue=np.zeros((line,pixel))

# Store 16-years NDVI
for iyear in range(2000,2016):

    infile=indir+"MOD13A2.C5.NDVI."+"{0:04d}".format(iyear)+".Asia.QDEG.ann.flt"
    fin=open(infile,'rb')
    dbuf=np.fromfile(fin,dtype=np.float32)
    dndvi_buf=dbuf.reshape(line,pixel)
    dndvi_ann[iyear-2000,:,:]=dndvi_buf

# Calculate linear trend
for iline in range(0,line):
    print(iline)                          #この行は単に進捗を示すため
    for ipixel in range(0,pixel):

        dndvi=dndvi_ann[:,iline,ipixel]
        idx_err=np.where(dndvi<-990.0)

        if len(idx_err[0])>0:
            dndvi_trend[iline,ipixel]=-999.0
            dndvi_pvalue[iline,ipixel]=-999.0
        else:
            #Originally, polyfit was used for linear regression
            #However, I found that linregress (scipy) is better
            #Use scipy.linregress to get p-value
            slope, intercept, r_value, p_value, std_err = linregress(range(0,16),dndvi)
            dndvi_trend[iline,ipixel]=slope
            dndvi_pvalue[iline,ipixel]=p_value

# Adjust min-value
idx_min=np.where(((dndvi_trend<im_min) & (dndvi_trend>-990.0)).all())
dndvi_trend[idx_min]=im_min
```

```
# Display output
plt.figure(figsize=(20,12))

plt.subplot(1,2,1)
fig=plt.imshow(dndvi_trend,clim=(-0.01,0.01),cmap=cm.jet)
fig.cmap.set_under('w')
stitle='NDVI Linear Trend (2000-15) (NDVI/year)'
clb=plt.colorbar()
plt.title(stitle)
plt.axis('off')

plt.subplot(1,2,2)
fig=plt.imshow( dndvi_pvalue,clim=(0.0,0.1),cmap=cm.jet)
fig.cmap.set_under('w')
stitle='trend p-value (significance)'
clb=plt.colorbar()
plt.title(stitle)
plt.axis('off')

plt.show()
```

補足. ファイルフォーマット変換

第 3 章でバイナリデータを扱ってきたが、これらのファイルを GIS ソフトウェアなどで読むには多少の加工が必要である。例えば、オープンソースである QGIS で読み込ませるには、地図座標を情報として格納できるファイルフォーマットにする必要がある。

これまで扱ってきたデータ（単なるバイナリファイル）を GIS やリモセンソフトウェアで読み込ませる方法の一つであり、元のデータの形を壊さない便利な方法の一つとして、ENVI フォーマットがある。ENVI は、リモートセンシングデータ解析ソフトウェア（商用）の一つである。ENVI フォーマットでは、テキスト形式のヘッダファイルを一つ作成するだけで、GIS データフォーマットとなり、ソフトウェアでも読み込みが可能となる。その上で、このファイルを geotiff などのより広く認識されているフォーマットに変換すればよい。

例えば、3.2 章で用いた土地被覆分類図(MOD12Q1.ASIA.QDEG.2001.byt)については、MOD12Q1.ASIA.QDEG.2001.byt.hdr というファイル名で以下の内容を含むテキストファイルを作成してみよう。

```
####以下開始
ENVI
description = {
  File Imported into ENVI.}
samples = 480
lines   = 360
bands   = 1
header offset = 0
file type = ENVI Standard
data type = 1
interleave = bsq
sensor type = Unknown
byte order = 0
map info = {Geographic Lat/Lon, 1.0, 1.0, 60.125, 79.875, 0.25, 0.25, WGS-84,
units=Degrees}
wavelength units = Unknown
####以下終了
```

以下、ENVI フォーマットにおけるヘッダファイルの説明を付ける。

ENVI フォーマットにおけるヘッダファイルの説明

```
ENVI
```

```
description = {
  File Imported into ENVI.}
samples = 480          # Pixel Number (Column)
lines   = 360         # Line  Number (Row)
bands   = 1          # Band  Number
header offset = 0
file type = ENVI Standard
data type = 1          # Type:    1: uint8, 2: int16, 3: int32, 4: float,
                                  5: double, 12: uint16, 13: uint32,
                                  14: int64, 15: uint64
interleave = bsq       # bsq/bil/bip
sensor type = Unknown
byte order = 0         # 0: intel, 1: IEEE
map info = {Geographic Lat/Lon, 1.0, 1.0, 60.125, 79.875, 0.25, 0.25, WGS-84,
units=Degrees}
wavelength units = Unknown

## README for map info ##
1. Projection name: "Geographic Lat/Lon"
2. Reference-X: Grid X of (X,Y) for Pixel Easting
3. Reference-Y: Grid Y of (X,Y) for Pixel Northing
4. Pixel Easting
5. Pixel Northing
6. x pixel size
7. y pixel size
8. Datum: "WGS-84"
9. Units: Degrees
```

自分の PC に GIS ソフトウェアがインストールされている場合は、作成した ENVI 形式のファイルを表示できるか確認してみよう。QGIS などでも読めるはずである。

最後に、ENVI 形式のファイルを geotiff 形式に変換してみよう。ここでは、GDAL という地図投影法の変換などソフトウェアが多く含まれており、リモートセンシングや GIS の業界ではよく用いられているフリーソフトを利用する。以下の

```
#### スクリプト開始
#Import gdal
from osgeo import gdal
```

```
infile='MOD12Q1.ASIA.QDEG.2001.byt'
outfile='MOD12Q1.ASIA.QDEG.2001.byt.tif'
#Open existing dataset
src_ds = gdal.Open( src_filename )

#Open output format driver, see gdal_translate --formats for list
format = "GTiff"
driver = gdal.GetDriverByName( format )

#Output to new format
dst_ds = driver.CreateCopy( outfile, src_ds, 0 )

#Properly close the datasets to flush to disk
dst_ds = None
src_ds = None
#### スクリプト終了
```

Geotiff になれば、リモートセンシングや GIS ソフトウェアでなく、通常の画像ビューワーでも表示ができるようになる。

自由課題

今回の実習で学んだ技術を用いて、自分自身で何らかの解析をしてみよう。これらをまとめた上で最終課題のレポートとする。課題の案としては

① 今回の実習で用いたデータを利用した解析

・CO_2 growth rate, 地球の平均気温, SOI の指標の時系列変化を解析することによる、地球規模 CO_2 循環の理解

・アジア域で 2001-2006 年において、植生指数がどこで増加・減少したのか?

・アジア域において、各植生区分毎に植生指数はどのように季節変化しているか?

・アジア域において、様々な観測サイトにおける植生指数の月別変化を把握する

② その他提供可能なデータ

・1982 年以降の衛星からの植生指数グローバルデータ(0.5 deg x 0.5 deg など)

・2000 年以降の様々な MODIS センサからのプロダクト

(アジア域・日本域；雪・地表面温度・植生指数など)

・グローバル・アジア・日本などの気候データ(気温・降水量・日射など; 1900 年以降)を利用した解析(異常気象と考えられる年を探す、近年の気候変動傾向を抽出するなど)

→　その他希望があれば応相談。提供可能ならば提供します。