**Pythonを利用した地球科学・数値解析入門**

**Python Training Course for Earth Science**

**千葉大学・環境リモートセンシング研究センター・小槻研究室**

**Kotsuki Laboratory, CEReS, Chiba University**

**https://kotsuki-lab.com/**

**contact information:   
shunji.kotsuki@chiba-u.jp**

**History of Development & Contributors**

* **2020.10.19 α1.0 日本語版 中村理絵・樺山修**
* **2021.03.25 β1.0 日本語＋英語版 樺山修・並木祐太・中村理絵**
* **2021.08.07 β2.0 日本語＋英語版 樺山修・飯塚かつら**

# 目次

[目次 3](#_Toc79157727)

[Outline 4](#_Toc79157728)

[1. はじめに (Introduction) 5](#_Toc79157729)

[1.1. なぜPython を推奨するのか？(Why Python?) 5](#_Toc79157730)

[1.2. 本資料の構成 (Structure of this manual) 8](#_Toc79157731)

[1.3. お願い と 感謝 (Request and appreciation) 8](#_Toc79157732)

[1.4. Pythonのダウンロード Downloading Python 9](#_Toc79157733)

[1.5. 本書の注意書き (Caution) 15](#_Toc79157734)

[1.6. スクリプトの実行の仕方 (How to run the script) 18](#_Toc79157741)

[1.7. ライブラリのインストールの仕方 (How to install the library) 20](#_Toc79157749)

[1.8. ノートブックの便利な機能の紹介 (Introducing useful features of notebooks) 21](#_Toc79157750)

[2. 基本演習 (Basic exercises) 24](#_Toc79157751)

[2.1. 基本演算 (Basic operations) 24](#_Toc79157753)

[2.2. 変数と配列 (Variables and arrays) 26](#_Toc79157760)

[2.3. 制御文 (Control statement) 37](#_Toc79157761)

[2.4. ファイルの入出力 (File input / output) 43](#_Toc79157762)

[2.5. グラフの描画 (Drawing a graph) 45](#_Toc79157763)

[2.6. scriptによる命令実行（復習）(Command execution by script (review)) 54](#_Toc79157764)

[2.7. Pythonの間違えやすい落とし穴 (Common mistakes in Python) 56](#_Toc79157771)

[3. Pythonによるテキストデータの解析 (Text data analysis with Python) 62](#_Toc79157772)

[3.1. 解析とは何か？ (What is analysis?) 62](#_Toc79157774)

[3.2. 解析の工程 (Analysis process) 63](#_Toc79157779)

[3.3. 積算棒グラフの作成 (Creating an integrated bar graph) 64](#_Toc79157780)

[4. Pythonによるバイナリデータの処理 (Processing binary data with Python) 81](#_Toc79157782)

[4.1. バイナリデータとは (What is binary data) 81](#_Toc79157783)

[4.2. 二次元データのデータ格納方法 (How to store 2D data) 82](#_Toc79157789)

[4.3. 衛星データを扱う工程 (Process of handling satellite data) 87](#_Toc79157790)

[4.4. Ctlファイル付きのバイナリデータ処理 (Binary data processing with Ctl file) 87](#_Toc79157792)

[5. Pythonによるデータの取得方法 (How to obtain data with Python) 102](#_Toc79157793)

[5.1. データの取得方法 (How to obtain data) 102](#_Toc79157794)

[5.2. データのダウンロード (Data download) 110](#_Toc79157801)

[5.3. 画像の作成 (Creating images) 116](#_Toc79157802)

[5.4. 国ごとの時系列グラフ (Time series graph by country) 130](#_Toc79157803)

[5.5. 加重平均 (Weighted average) 132](#_Toc79157804)

# Outline

[Outline](#_Toc54035403) 3

[1. Introduction 4](#_Toc54035404)

[1.1. Why Python? 4](#_Toc54035405)

[1.2. Structure of this manual 7](#_Toc54035406)

[1.3. Request and appreciation 7](#_Toc54035407)

[1.4. Downloading Python](#_Toc54035408) 8

[1.5. Caution 1](#_Toc54035409)4

[1.6. How to run the script 1](#_Toc54035416)7

[1.7. How to install the library](#_Toc54035424) 19

[1.8. Introducing useful features of notebooks 2](#_Toc54035425)0

[2. Basic exercises 2](#_Toc54035426)3

[2.1. Basic operations 2](#_Toc54035428)3

[2.2. Variables and arrays 2](#_Toc54035435)5

[2.3. Control statement 3](#_Toc54035436)6

[2.4. File input and output 4](#_Toc54035437)2

[2.5. Drawing a graph 4](#_Toc54035438)4

[2.6. Command execution by script (review) 5](#_Toc54035439)3

[2.7. Common mistakes in Python 5](#_Toc54035446)5

[3. Text data analysis with Python 6](#_Toc54035447)1

[3.1. What is analysis 6](#_Toc54035449)1

[3.2. Analysis process 6](#_Toc54035454)2

[3.3. Creating an integrated bar graph 6](#_Toc54035455)3

[4. Processing binary data by Python](#_Toc54035457) 80

[4.1. What is binary data](#_Toc54035458) 80

[4.2. Data storage method for two-dimentional data](#_Toc54035464) 81

[4.3. Process of handling satellite data 8](#_Toc54035465)6

[4.4. Dealing with binary data with ctl file 8](#_Toc54035467)6

[5. How to obtain data with Python](#_Toc54035468) 101

[5.1. How to obtain data](#_Toc54035469) 101

[5.2. Downloading data 10](#_Toc54035476)9

[5.3. Creating images 11](#_Toc54035477)5

[5.4. Time series graph by country 12](#_Toc54035478)9

[5.5. Weighted average 1](#_Toc54035479)31

# はじめに (Introduction)

## なぜPython を推奨するのか？(Why Python?)

地球科学分野において、「コンピュータ」を用いた数値解析（＝プログラミング）は必須の技術となってきている。特に近年では、衛星観測情報や気象予測情報など、一般に公開されている情報が爆発的に増加している。簡単な数値処理であればEXCELなどでも充分である。しかし、利用可能な情報を用いて計算しようとする場合は、ギガバイト～テラバイトのデータ処理を行う必要があり、そのような大規模データ処理はプログラミング無しに行うことはできない。プログラミングとは、簡単に言ってしまえば「繰り返し計算」により、大規模データを効率良く処理する仕組みである。１０～２０年前までの時代であれば、実験・観測を主としていた研究者には高度なデータ解析能力は必要とされなかったのかもしれない。しかし、近年では他分野との競合などにより、数値解析技術の習得は、もはや必須と言える状況と言える。

プログラミング言語は、CやFortranに代表されるコンパイル言語と、Python, BASH, PERLなどのスクリプト言語がある。このうち、今回扱うPythonはスクリプト言語である。スクリプト言語は、計算速度ではコンパイル言語に劣るものの、その習得の容易さや、試行錯誤の簡易さにメリットがあり、初学者が最初に取り組む言語として敷居が低い。今のところ、スーパーコンピュータなどのメニーコア　(many core)を用いる並列計算を使わないのであれば、Pythonのみで十分に研究を遂行することが出来る。小槻自身、もし大学生の自分に伝えることが出来るのであれば、自信をもってPythonからの習得を強く推奨するだろう。

具体的にPythonを用いるメリットには、下記の点が挙げられる。

* スクリプト言語であり、習得が比較的容易
* プログラミングによるデータ処理から、描画までを一括で行うことが出来る
  + また、工夫次第で論文掲載レベルのカッコいい絵を描くことも容易（大事！）
* jupyternotebook やgoogle coraboratoryなどのアプリを使い、WEBブラウザで実行可能
  + 数行ずつ開発・チェックが出来て、コーディングが容易～詳しくは後述する。
* 機械学習への接続性が高い
  + 多くの機械学習ライブラリがPythonで提供されており、それらを用いた計算が容易

　Pythonによる数値解析、などの書籍やマニュアルは、多数見られる。それでもなお、研究室としてこのマニュアルを整備しようと決めた理由は、「地球科学特有のデータ特性・処理技術」にあり、それらを一般書籍でカバーしきれないからである。例えば、「年・月・日・秒」などの時間方向のデータ処理や、緯度・経度・国・領域といった地域毎のデータ処理がある。それらの数値処理には、4次元を超える多次元変数を扱うことも頻発する。こう書くと大変に思えるが、この「多次元変数の処理」についても、本マニュアルを通した演習により、難しい概念ではないと理解して頂けると思う。

書籍からの習得では、結局何が必要な技術か分からず、どこに集中力を割いて良いのか分からなくなる。そのため、このマニュアルは、研究に役立つ技術を最短で学べるように意識して整備した。実際に、研究室に配属された学生・スタッフも用いているが、概ね１～３ヶ月の演習・試行錯誤で、データ処理技術やプログラミングに対する知識が格段に上がってきている様である。ただし、上述したように、本マニュアルが提供するのは、「最短経路」であり、より深いデータ処理の「体得」には、研究活動を通じて手を動かし、試行錯誤を続ける事が必須である。

本資料の目的は、マニュアルを公開の上、読者に興味を持って頂き、地球科学数値処理の理解を深めて頂くことである。また、本マニュアルに興味を持ち研究をしてみたいという方がいらっしゃれば、是非積極的にコンタクトを取って頂ければ幸いである。

In the field of earth science, numerical analysis (= programming) using a "computer" has become an essential technology. Especially in recent years, information that is open to the public, such as satellite observation information and weather forecast information, has increased explosively. For simple numerical processing, EXCEL is enough. However, when trying to calculate using open source data, it is necessary to perform gigabyte to terabyte data processing, and such large-scale data processing cannot be performed without programming. Simply, programming is a mechanism for efficiently processing large-scale data by "repetitive calculation". In1990s and 2000s, researchers who were mainly engaged in experiments and observations may not have needed advanced data analysis skills. However, in recent years, the acquisition of numerical analysis technology is essential due to competition with other fields.

Programming languages include compilation languages such as C and Fortran, and scripting languages such as Python, BASH, and PERL. Of these, Python, which we are dealing with this time, is a scripting language. Although the scripting language is inferior to the compiled language in terms of calculation speed, it has advantages in terms of ease of learning and ease of trial and error, and relatively easy to learn as the first language for beginners. For now, if you don't use parallel computing that uses many cores such as supercomputers, you can do enough research with Python alone. Kotsuki himself would strongly recommend learning from Python if he could have told himself when he was a college student.

Pros of using Python are as follows

* Scripting language is easier to learn
* From programming data processing to drawing, it is all in one
  + Capable of creating a cool graph for papers
* Runs on web browser using Jupyter Notebook or Google Coraboratory
  + Possible to code and debug line by line will be explained later.
* Works well with machine learning
  + Many machine learning Libraries are available

Many books and manuals such as numerical analysis by Python are seen today. Nevertheless, the laboratory decided to prepare this manual because "data characteristics and processing technology peculiar to earth science" is hard to be covered by general books. For example, there is data processing in the time direction such as "year / month / day / second" and data processing for each region such as latitude / longitude / country / region. In those numerical processing, multidimensional variables exceeding 4 dimensions are seen frequently. Multidimensional processing may seem difficult, but I think that you can understand that this "processing of multidimensional variables" is not a difficult concept through the exercises in this manual.

Learning from books can be hard for you to tell what is needed, and you don't know what to focus on. Therefore, this manual is designed with the intention of learning the important techniques for research in the shortest possible time. In fact, the students and staff assigned to the laboratory are also using this manual and it seems that their knowledge of data processing technology and programming has improved dramatically after about 1 to 3 months of practice using trial and error. However, as mentioned above, this manual is the "shortest path." Hands-on experience through research activities is essential and you must continue trial and error to master the deeper data processing.

The purpose of this material is to open the manual, to make readers interested, and to deepen their understanding of earth science numerical processing. Also, if you are interested in this manual and would like to do research, we would appreciate it if you would contact us.

## 本資料の構成 (Structure of this manual)

本資料は下記の構成となっている。 (This manual is structured as following)

* 1章：　環境の構築 (Building the environment)
* 2章：　基本演習 (Basic exercise)
* 3章：　テキストデータの処理 (Processing text data)
* 4章：　バイナリデータの処理 (Processing binary data)

千葉大学の講義において、これらのPython技術を前提として、実際に地球大気・リモートセンシングデータ処理演習を行っている。2020年の新型コロナウィルス感染拡大により、講義資料や動画を整備している。本資料を更に一歩進めて数値解析を行いたい場合は、下記のリンクから講義情報にアクセスすることが可能である。

<https://kotsuki-lab.com/internal-pages/>

In the lecture at Chiba University, we are actually conducting the Earth Atmosphere / Remote Sensing Data Processing Exercise on the premise of these Python techniques. Lecture materials and videos are being prepared due to the spread of the covid-19 infection in 2020. If you would like to take this material one step further and perform numerical analysis, you can access the lecture information from the link below.

<https://kotsuki-lab.com/internal-pages/>

## お願い と 感謝 (Request and appreciation)

この資料は、研究室のメンバーらにより作成された。可能な限り分かり易く記述したつもりであるが、Pythonを初めて扱う読者には難解な記述もあるかもしれない。また、記述したソースコードにバグがあるかもしれない。その場合は、難解であった場所・誤りがあった箇所を、作成者まで伝えて欲しい（超重要！）。それらのフィードバックを踏まえて、この資料は今後も良くしていきたいと考えている。

この資料の作成にあたり、千葉大学・環境リモートセンシング研究センター・市井研究室で整備されている資料に大きくinspiredされた（<http://ichiilab.weebly.com/classes.html>）。基本的に、学習コンテンツについては小槻研究室のスタッフで全て書き換えているが、この資料整備という発想はそもそも市井先生の研究室運営から得られたものであり、ここに記して謝意を示す。

This manual is created by the members of the lab. We've tried to make it as clear as possible, but it may be still difficult for readers new to Python. Also, there may be bugs in the written source code. In that case, it is very important for you to contact and let us know where the issues or errors are so this material can continue to be improved.

This manual is greatly inspired by the materials from Chiba University, Environmental Remote Sensing Research Center, Ichii Laboratory (<http://ichiilab.weebly.com/classes.html>). Basically, all the learning contents are rewritten by the staff of Kotsuki laboratory, but the idea of creating this manual was originally obtained from the laboratory management of Professor Ichii, and I would like to express my gratitude here.

## Pythonのダウンロード Downloading Python

早速、Pythonを利用してみよう。以下の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)でも同様にできる筈である。本書では、初心者にも取り組みやすいJupyter Notebookというソフトウェア（IDLE）を使う。

Let’s start using Python. The following instructions are for windows, but it should be almost the same with other OS (Linux, Mac, etc.). In this manual, we use Jupyter Notebook, which is an easy and nice software (IDLE) for beginners.

Anacondaのインストールページ <https://www.anaconda.com/download/>

URLをブラウザに貼って、WEBページにアクセスしましょう。DOWNLOADというボタンがあるので、クリックする。

Installation website for Anaconda <https://www.anaconda.com/download/>

Paste the url in web browser and access the website. Click download icon.

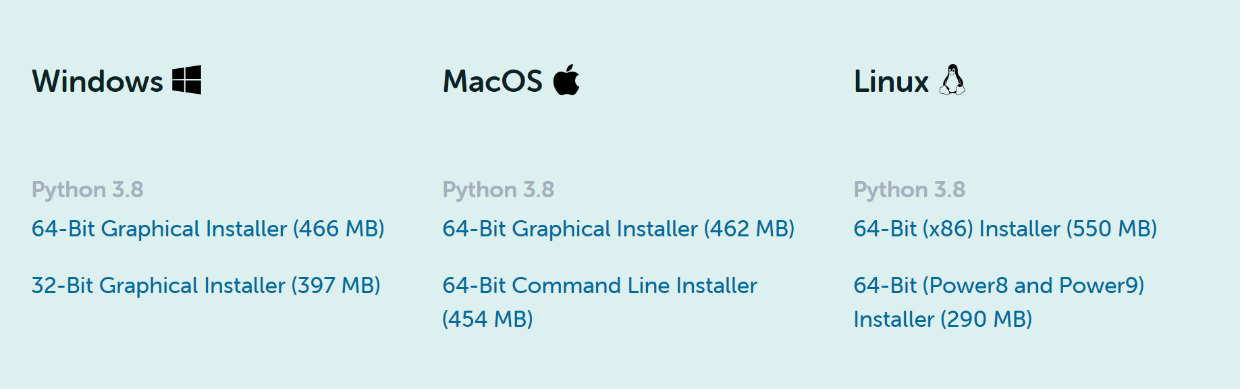


Figure 1: Figureanacondaのダウンロード (Download anaconda)

こういう画面にロールダウンされる。Python 3.8 64-Bit Graphical Installerをクリックしよう。

You’ll be scrolled down as above. Click Python 3.8 64-Bit Graphical Installer.

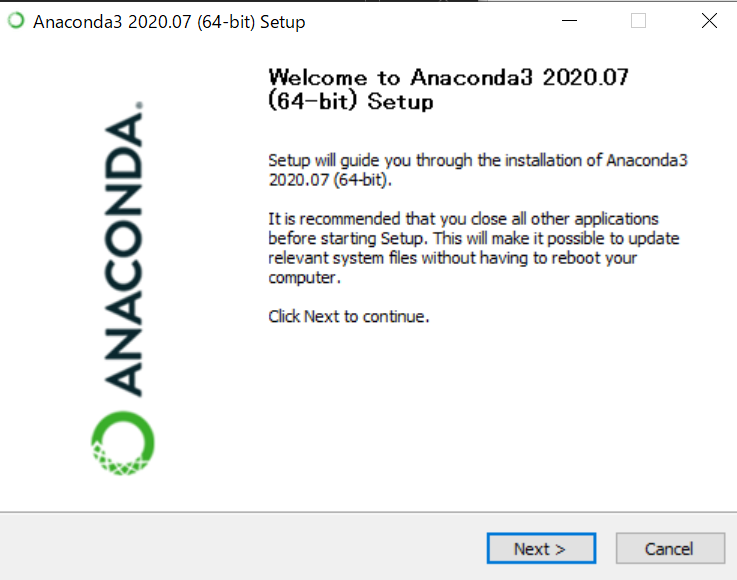


Figure 2: アナコンダのダウンロード、その２ (Download Anaconda part 2)

exeファイルがダウンロードされると思うので、自分のダウンロードフォルダに行って、インストールしよう (**エラー! 参照元が見つかりません。**)。NEXTで進み、I Agreeで利用規約に同意する。

exe file will be downloaded on your computer. Check in download folder click it to start the installation.

Click NEXT and agree with the policy

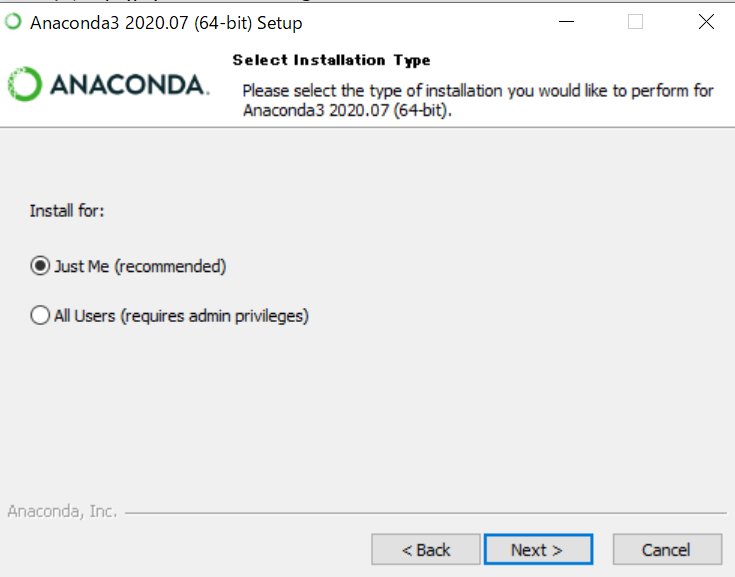


Figure 3: アナコンダのダウンロード、その３ (Download Anaoconda part 3)

ユーザーにインストールか全体にインストールするか聞かれるが、特に拘りがなければ、Just Meを選択する。続いて、ダウンロード先を聞かれるが、拘りが無ければ、そのままNEXT。

You will be asked if you want to install it to all users or just yourself. Click " Just me" unless you have a special reason. Next, you will be asked the download location. Click " Next" unless you have a special reason to change it

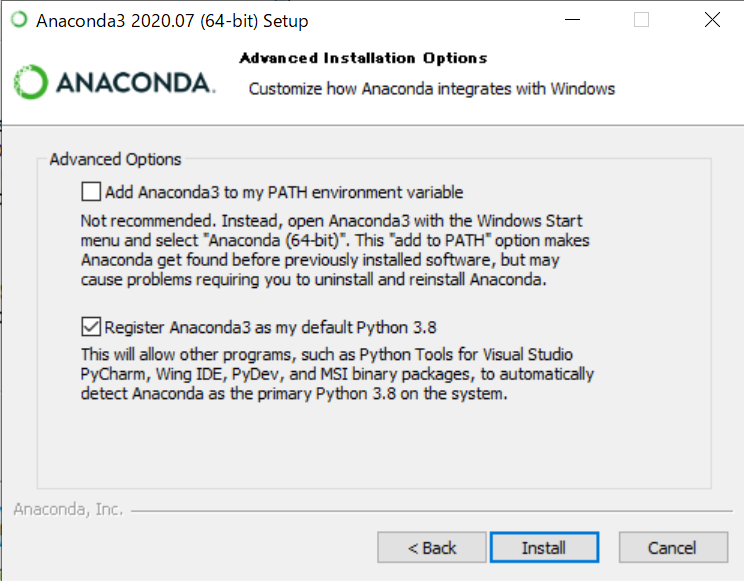


Figure 4:　アナコンダのダウンロード、その４ (Download Anaconda part 4)

選択肢「Add Anaconda to my PATH environment variable」は、環境変数 PATH に Anaconda のフォルダを追加するか否かを決める選択肢だが、チェックしなくて良い。 また「Register Anaconda as my default Python 3.7」は「Anaconda をデフォルトの Python 3.7 として登録するか」の選択肢である。チェックのままで問題なし。 このまま「Install」ボタンをクリックし、インストールを開始する。インストールが完了したら、Finishで閉じよう。

The top checkbox, "Add Anaconda to my PATH environment variable" is whether you add the path of Anaconda to Environmental variables? or not, which you can leave it unchecked.

"Register Anaconda as my default Python 3.7" is whether you reｇister Anaconda as the default Python3.7 or not, which you can keep it checked. Click "Install" to start installation. Click "Finish" when it’s done to close the window.

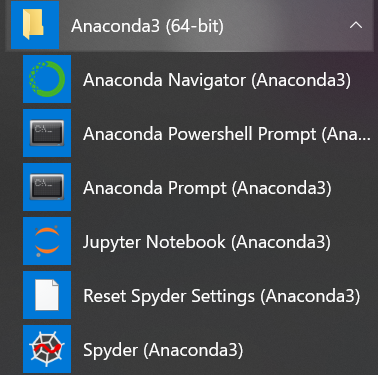


Figure 5:　Jupyter notebookの実行方法 (how to run Jupyter Notebook)

スタートメニューにフォルダが表示されていると思う。この中のJupyter NotebookとAnaconda　Promptを使う。Jupyter Notebookはプログラミング開発。Anaconda PromptはmoduleやLibrary（便利な機能を持ったPython専用の拡張機能みたいなもの）のダウンロードとスクリプトの実行に使う。使いやすいようにデスクトップに置くか、右クリックでスタートまたはタスクバーにピン止めしよう。ちなみに私はタスクバーにピン止めしている。それでは、Jupyter Notebookをクリックしよう。

Anaconda3 folder will be created in your start menu. Jupyter Notebook and Anaconda Prompt are what we use. Jupyter Notebook is for editing codes and Anaconda Prompt is for downloading module or Library (It’s sort of Python extensions with useful functions) and is also for executing a script. It’s recommended that you pin them in start menu or in task bar. I personally prefer task bar. Then, click Jupyter Notebook.

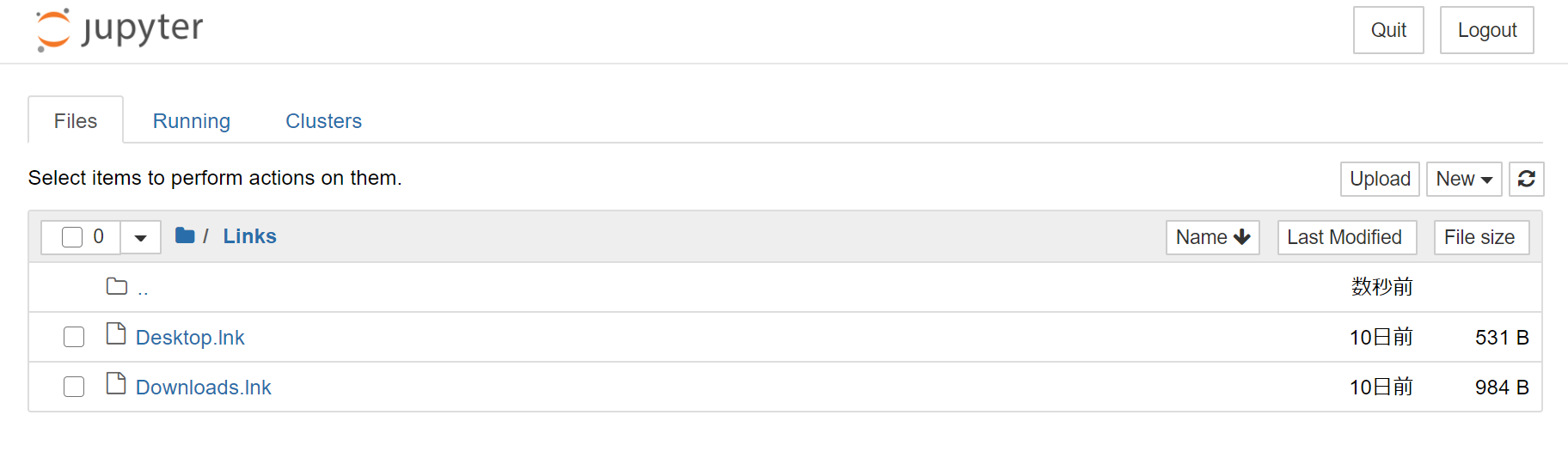


Figure 6:　Jupyter notebookの使い方 (How to use Jupyter Notebook)

この様な画面がWEBブラウザ（私の場合はGoogle Chrome）にて起動すると思う。ちなみに、WEBブラウザ上で起動しているように見えるからオンラインなのかと思うが、オフラインで実行されている。つまり、ネット―ワークに接続していなくてもJupyter notebookは利用できる。試しにネットワークを切ってみても、動くはずである。

Figure 6の右上に「New」というボタンがあるので、クリックする。

Web Browser (such as Google Chrome) will open shortly. You might think that Jupyter Notebook runs through the Internet but it runs locally, which means it’s possible to use Jupyter Notebook even when you are offline. You could try testing it.

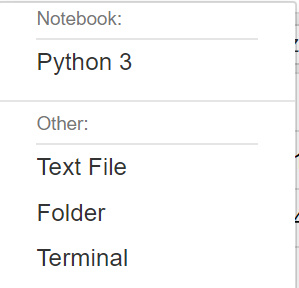


Figure 7:　Jupyter notebookの使い方、その２ (How to use Jupyter Notebook part 2)

すると、こういう画面(Figure 7)が出てきます。Python3をクリックしましょう。

Click "Python 3" in the top of the list in Figure 7.

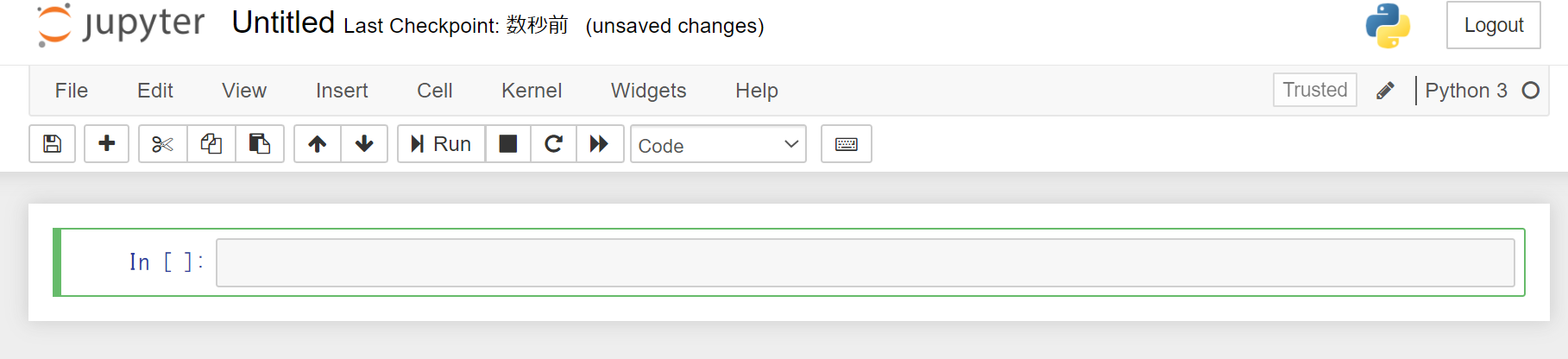


Figure 8:　Jupyter notebookの使い方、その３ (How to use Jupyter notebook part 3)

すると、やっとnotebookの画面（Figure 8）が出てきます。緑の枠の内側(セルという)、In[ ]:の右側の灰色の枠にプログラムを書いていきます。試しに、print(‘codeの結果はここだよ’)と打ってみましょう。実行するには、実行したいセルをクリックして、Shift+ENTERを押すか。上部にあるRunのボタンを押す。

Notebook window will open in Figure 8. In the green frame (called "Cell"), type your code in the gray box next to "In[ ]: " then type "print (‘Output appears here’) " as a first try. To run the code, click the cell that you’re working on and hit "Shift + ENTER" or click "Run" icon in the menu bar above.

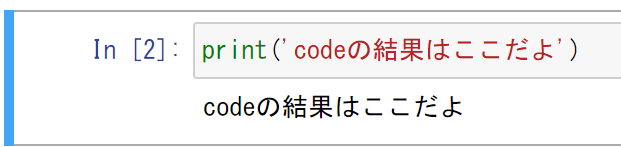


Figure 9:　Jupyter notebookの使い方、その４ (Jupyter Notebook part 4)

Figure 9という結果が出てくる。Wordと同じで、一行打った後にENTERを押したら、行を変えることができるので、長いプログラムを書く時には改行を適宜使おう。

Pythonはインタラクティブな（対話）言語と言われている。それは、一行書いて、その結果をすぐに見ることができるからだ（他の言語では、この間に１ステップないし、２ステップ入る）。このインタラクティブな動作が初心者に優しいということで、よく使われる言語になっており、私も開発に便利だと思っている。

Output will appear in Figure 9. You can start a new line by pressing ‘Enter’ like in Microsoft Word, which is handy when coding a lot.

Python is one of the interactive languages. That's because you can write a line and see the results right away (in other languages, you'll have one or two steps in between). This interactive behavior is beginner-friendly, making it a popular language, and useful for development.

## 本書の注意書き (Caution)

### 本書の使い方 (How to use this manual)

１章の最後に、Pythonを進める上で身に着けるべき考え方を概観する。とはいえ、おそらく一度読んだだけで全て腑に落ちる事は無い様に思う。プログラミングを学ぶには、実際に手を動かしながら体得することが一番の近道である。したがって、立ち止まり過ぎず、実際に２章の作業を進めながら振り返って習得するのが良いと思う。

At the end of Chapter 1, I'll give an overview of the ideas needed to get started with Python. However, understanding everything by reading it once is difficult. The best way to learn programming is to actually move your hands and practice it. Therefore, it is better not to stop to look back but learn while actually proceeding with the work of Chapter 2.

### 本書の扱い方 (How to handle this manual)

本書の使い方を説明する。打ち込むプログラムのことをコード（Code）と呼ぶ。これを本書では、テキストボックスの中に記述していくので、それを書き写してもらうのだが、基本的にコピペ（copy & taste）は推奨しない。自分で手を動かして初めて気づくことが多いからだ。実際に手を動かしながら打ち込んでいって、コードごとにどう動いているのか一つずつ地道に考えていって貰えると上達が早くなるだろう。ちなみに私はコピペしているが、数字などを色々変えて、どう動くのかなどを試しながら習得速度を高めている。応用的に考えたりもしている。例えば、forループを月で回しているが、これを年月日で回したらどうなるのだろうか？実は月だけで回すより年月日で回すのはずっと難しい。かなり調べた結果、出来るようになった。自分で気になったことやりたいことを調べて、習得するというのはモチベーション的にも単純作業よりもずっと集中できるので上達も早くなる。要は自分で思考しているかどうかが重要なのだ。その辺りを意識しながら、本書も一つのツールとして見て、各自で自分なりのやり方で実施してもらいたい。

本書では、ビジュアルも大事にしている。イメージを持って取り組んでほしいからだ。全体像をまず理解して、ステップを理解して、現在自分はどこにいるのか。何のためにこの作業をしているのか理解した上でやってほしい。もちろん、とりあえず進めて、完成した時に、完成図がわかった上で一から振り返って工程を理解するという方法でも良い。

本書では、左にナビゲーションをつけている。目次のように自分の読みたい章までワンクリックで飛べるようになっているので有効活用してほしい。ちなみに、そんなものは表示されないという方は、表示からナビゲーションウィンドウの欄のチェックボタンにチェックをつければ、見えるようになると思う。

一つ注意してほしいのは、Pythonはユーザーの多い言語なので、検索すれば、たいていのことは出てくる。しかし、何がわからないのか理解していて、検索できるというのは、もうすでに中級者なのだ。なので、初心者の方は、「何が分からないか、それが分からない」と思った時は、周りのPythonに詳しい人に聞いてほしい。

余談だが、こんな意見もあると思って聞いてほしい。「聞くは一時の恥、聞かぬは一生の恥。」ということわざがあるが、私は同意しない。分からないということは、恥ではない。人に聞くのも恥ではない。人は生まれた時、何も知らない。しかし、私もあなたも他人から学び、色んなことができるようになった。教えを乞うことは恥ではない。教えてもらう時間を取ることによって、他人の貴重な時間を奪ってしまうという意見もあり、それは正しい。だからこそ、自分で一度考えて、どうしてもわからなくなってから、相手に何が分からないのか分かりやすいように、質問の仕方も具体的にしっかり練ってから聞くようにしよう。しかし、こういう考え方もある。質問される側から考えた時に、質問に答えられるということは、しっかり自分がそのことについて理解していなくてはいけない。言い方を変えれば、質問されるというのは、自分が本当に理解しているのか確かめる良い機会でもあるのだ。だから、私は人に助けを求められたら全力で助けることにしている。世の中こういうポジティブなとらえ方で持ちつ持たれつになれば良いのにと筆者は考えている。

First, this manual will explain how to use this book. The program to be typed in is called "Code". I will write this in the text box, so copy it. However, I do not recommend copy & paste. This is because you often learn something only when you move your hand. By actually moving your hands and typing it in, and thinking about how each code works, one by one, your progress will be faster. By the way, I'm also copying, and increasing the learning speed by changing the numbers and trying out how it works. Also think about it in an applied way. For example, by turning a "for loop" by the month, yet what if it is turned by the date? Actually, it is much more difficult to turn by date than by turning only by month. After a lot of research, it was accomplished. Finding out what you are interested in and learning what you want to do is much more motivating than simple work, so you can improve faster. The point is to think for yourself. Keep this in mind. I would like you to see this manual as a tool and implement it in your own way.

In this book, visuals are also important. Proceed this lesson with an overall image of the wholeprocess. Understand the steps and know where you are at now. Please be aware of what you are doing this work for. It’s okay to proceed your work without complete understanding of Python. When the task is accomplished with more understanding of your completed drawing, look back the beginning and understand the process.

In this manual, the navigation is attached to the left. Like the table of contents, you can jump to the chapter you want to read with one click, so please make effective use of it. By the way, if you don't see such a thing, you can see it by checking the check button in the navigation window column from the display.One thing to keep in mind is that Python is a user-rich language, so a search will find most areas. However, if you know what you don't understand and can search, you are already in intermediate level. If you are a beginner and don't know what you don't understand, ask someone who is familiar with Python.

Here is my opinion. There is a proverb that hearing is a temporary shame, and not hearing is a lifetime shame, but I disagree. Not knowing is not a shame. It's not a shame to ask people. People know nothing when they are born. However, we have learned from others and have become to be able to do many things. Begging for teaching is not a shame. There is also an opinion that asking for help will take away valuable time from others, which is correct. That's why you should come up with the question that others can easily understand. However, if you think from the‘being asked’side, you must have the complete understanding of the topic. In other words, being asked is also a good opportunity to see if you really understand the concept or not. That's why I try my best to help people when they ask for my help. The author hopes that the world could function in this positive way.

### Codeの注意書き (Code Note)

これから、codeを書き、その解説という形で本書は進めていく。コードの打ち込み先というのが、Jupyter Notebook、メモ帳、アナコンダプロンプトの３つになる。それぞれ、ノートブック、スクリプト、プロンプトと簡易的に呼んでいく。そして、それぞれ塗りつぶしを灰色(Figure 10)、青色(Figure 11)、黒色(Figure 12)として目で見て分かりやすくする。ノートブックでは、最初、分かりやすい表示のためにInとOutを使っているが、後半はスクリプトにしているので、この"In Out"が無くなる。ノートブックでは改行できるので、一つのセル（文字を打ち込む枠内のこと）に何行もコードを書くことができる。本書では、スクリプトとして書くが、もちろんノートブックで改行しながら、書き込む開発を基本的には推奨する。なぜなら、ノートブックはすぐに結果が見えるので、開発にとても便利だからだ。スクリプト実行は、コードが長くなり扱うデータが増えてくると便利になるが、本書を進めていく上ではその様な長いコードとは、おそらく出会わない・・・。anacondaプロンプトはスクリプトの実行とライブラリのインストールでしか使わないので、登場回数は少ないだろう。

Now the code will be written with the explanation. There are three code destinations: Jupyter Notebook, Notepad, and Anaconda Prompt. They are simply called notebook, script, and prompt. Then, make the fills gray (Fig. 10), blue (Fig. 11), and black (Fig. 12) to make them easier to see. In the notebook, In and Out are used for easy display at first, but since the latter half is a script, this "In Out" disappears. Notebooks allow line breaks, so you can write many lines of code in a single cell (inside the box where you type characters). In this book, it is written as a script, but of course it is recommended to write it while breaking the line in the notebook and develop it. Notebooks are very convenient for development because you can see the results immediately. Script execution becomes convenient when the code becomes long and the amount of data to be handled increases, but in the course of this book, such long code will probably not be encountered. The anaconda prompt is only used for script execution and library installation, so it will appear less often.

|  |
| --- |
| In [1]: import numpy as np  In [2]: import os |

Figure 10:　ノートブックの例。背景が灰色。 "In Out"で行番号が分かる。

(Notebook example. The background is gray. "In Out" tells the line number.)

|  |
| --- |
| import numpy as np  import os |

Figure 11:　スクリプトの例。背景が水色。

(Script example. The background is light blue.)

|  |
| --- |
| >jupyter notebook |

Figure 12:　アナコンダプロンプトで実行する場合。背景が黒色。

(When running at the anaconda prompt. The background is black.)



## スクリプトの実行の仕方 (How to run the script)

本書では、ノートブックだけでなく、スクリプト実行を用いる。１章のFigure 11ではスクリプトは青い枠で囲むと注意書きしてあり、このボックスが登場した場合に用いる。この実行はいたってシンプルだ。メモ帳にノートブックのようにコードを書いて、それを保存するときに.pyという拡張子を名前の最後につけて保存する。ファイルのアイコンがテキストファイルではなくて、Pythonのマークになると思うので、そうであることを確認してほしい。その後、実行はアナコンダプロンプトで行う。

In this manual, we use not only notebooks but also run scripts. In Figure 11 of Chapter 1, it is noted that the scripts should be surrounded by a blue frame, which will be used when this box appears. This execution is pretty simple. Write the code in Notepad like a notebook. When you save it, save it with the .py extension at the end of the name. The file icon will be a Python mark, not a text file, so make sure it is .py. After that, you can run at the anaconda prompt.

|  |
| --- |
| >cd <directory>  >python <file name.py> |

上記の様に打つ。この最初の一行は、<directory>の部分に先ほどメモ帳で作成したコードのファイルの保存先フォルダ名を入れる。保存先フォルダの確認方法については、Figure 13を参照して欲しい。cdとは、change directoryの略で、ディレクトリ（フォルダ）間を移動するという意味だ。ファイルを指定するときに、ファイル名だけでは、どこにあるかわからない。その保存先フォルダまで指定してやっとコンピュータはそのフォルダがどこにあるか判断できるのだ。<file name.py>は、先ほどメモ帳で作成したコードのファイルの名前をそのまま打てばよい。

Type the scripts as shown above. In this first line, insert the location of the destination folder where the code file created in Notepad to the <directory> part. Here is how to check where the destination folder is. Right clink on the code file created in Notepad. Click on “property,” then your “path” will be seen as “C:\Users\kotsuki-lab-W02\Documents” shown in Figure 13. “cd” is an abbreviation for change directory, which means to move between directories, that is, folders. It is necessary to know where to move when changing directories. From Figure 13, you can copy “C:\Users\kotsuki-lab-W02\Documents” and paste onto anaconda prompt by right clicking <directory>. Just remember that “cd” is Linux and not Python language. In the second line of anaconda prompt, the command “python” means to run the script with python. When you specify a file, you won't know where the file is just by looking at the file name. Only by specifying the destination folder can determine where the targeted folder is. For <file name.py>, just type the name of the code file created in Notepad.



Figure 13 : directory確認方法 (Confirming your directory)

ノートブックとアナコンダプロンプトでは、動作がだいぶ違う。なぜならOSが違うからだ。ノートブックはWindowsでアナコンダプロンプトはLinuxコマンドを使わないといけない。だから、最初は、戸惑うだろうが、困ったらLinuxコマンドで検索してみよう。

The behavior is quite different between the notebook and the anaconda prompt. Because the OS is different. Notebooks must use Windows and anaconda prompts must use Linux commands. So, at first, you may be confused, but if you are in trouble, try searching with “Linux commands”.



## ライブラリのインストールの仕方 (How to install the library)

codeの注意書きとして、アナコンダプロンプトが登場した。スクリプトの実行を1.6で説明したが、それに続き、ライブラリ(Library)やモジュール(module)と呼ばれるもののインストールの仕方を説明する。では、ライブラリとは、何か。ライブラリとは、便利な機能のパッケージだ。例えば、平均を出したいとする。(1+2+3+4+5+6+7+8+9)/9=5。普通に計算するとこうなる。しかし、ライブラリを使えば、np.mean(np.arange(1,9+1))と実行すれば、同じ数が出てくる。これくらいの少ない数なら、手間は一緒だが、数が増えてくればライブラリを使ったほうが圧倒的に速い。つまり、ライブラリというのは、よく使われる処理を関数として用意してくれる便利なものなのだ。これを使わない手はない。実は、アナコンダというのは、すでによく使われるライブラリがインストールされたパッケージとなっている。なので、matplotlibなどの良く用いるライブラリを自分でインストールする必要はない。しかし、本書では、インストールする必要があるものも使うので、インストールの仕方を説明しておく。

Anaconda prompt had appeared as a note of code. This manual explained the script run in 1.6, but after that, this manual will explain how to install what is called a “library” and “module”. So, what is a library? A library is a package of useful features. For example, suppose you want to average (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9) / 9 = 5. If you calculate normally, it will be like this. However, if you use the library, you can run np.mean (np.arange (1,9 + 1)) and you will get the same number. With such a small number, the effort is the same, but as the amount of number increases, it is overwhelmingly faster to use the library. In other words, a library is a convenient thing that prepares frequently used processes as functions. There is no way not to use this. In fact, anaconda is a package that already has commonly used libraries installed. So, you don't have to install frequently used libraries such as matplotlib yourself. However, this manual also uses the ones that need to be installed, so this manual will explain how to install them.

|  |
| --- |
| >conda install cartopy |

至って、やり方はシンプルだ。アナコンダプロンプトに一行書くだけ。今回は例として4.4章、 Ctlファイル付きのバイナリデータ処理に登場する、Cartopyをインストールしている。この部分を自分のほしいライブラリの名前に変えれば良い。

余談だが、pipという別のパッケージ管理コマンドを使っている人もいる。もちろんpipでもいいのだが、anacondaをメインで使うのであれば、pipは使わないほうがいい。なぜなら、pipとanacondaは基本的に互換性がなく、すでにインストールされているライブラリであってもバージョンの違いや微妙な名前の違い(例えば、 pyqtとPyQt5など)によって、環境が上書きされたりする可能性があるらしいのだ。正直なところ、まだ私は実感できていないという話だ。実はanacondaではインストールできないライブラリがたまにある。そういうライブラリはpipでインストールしている。そういう時だけ使うことを推奨したいが、くれぐれもバージョンの違いには気を付けてほしい。

The method is very simple. Just write a line at the anaconda prompt. This time, as an example, I have installed Cartopy, which appears in 4.4. binary data processing with Ctl files. You can change this part to the name of the library you want.

However,, some people use another package management command called pip. Of course, pip is fine, but if you mainly use anaconda, you shouldn't use pip. This is because pip and anaconda are basically incompatible, and even if the library is already installed, the environment can be overwritten by different versions or subtle name differences (for example, pyqt and PyQt5).

To be honest, I haven't had a problem with it yet. Yet, there are some libraries that cannot be installed with anaconda. Such libraries are installed with pip. I recommend using it only in such cases, but please be careful about the difference in version.

## ノートブックの便利な機能の紹介 (Introducing useful features of notebooks)

ノートブックには、色々な便利な機能がある。それを紹介していきたいと思う。コードを開発する効率に大きく影響していくので、習得してほしい。

<https://qiita.com/zawawahoge/items/baa2a5318df079c5f7e5>

ここにノートブックのショートカットの一覧があるので、読んで、便利なものを使っていこう。この中でも便利なものを抜粋して、紹介する。

Notebooks have many useful features. I would like to introduce them. They will greatly affect the efficiency of code development, so please learn them.

<https://qiita.com/zawawahoge/items/baa2a5318df079c5f7e5>

Here's a list of notebook shortcuts, read them and use the ones that are convenient. Here are some of the most useful ones.

### 一括実行と停止 (Batch execution and stop)



Figure 14:　一括実行と停止 (Batch execution and stop)

上部にこんなアイコンがあるだろう。>>のようなアイコンで、一括で停止して、再度すべてのセル(cell)を実行してくれる。これを押せば全てのセルを再度実行してくれる。そして、黒い四角のボタンが停止ボタンだ。間違って実行を押してしまったときは、これを押そう。

There will be an icon like this at the top. With an icon like >>, it will stop all at once and run all cells again. If you press this, all cells will run again. And the black square button is the stop button. If you accidentally press run, press this stop button.

### コメントアウト (Comment out)



Figure 15:　コメントアウト (Comment out)

これがコメントアウトだ。[Ctrl + / ]を押すと、コメントアウトできる。＃の後の文字は実行しても、実行されない。なので、これを使い、メモ書きすることができる便利なものだ。何個か試したい時も、これで現行のコードをコメントアウトしておいて、他のコードを書き足していき実行。やっぱり前のコードに戻したい時は[Ctrl + Z ]で戻せる。いろんな使い方ができるので、ぜひ使ってほしい。

This is a comment out. You can comment out by pressing [Ctrl + /]. The character after the # is run, but not run. So, it's a convenient way to use it to write notes. If you want to try a few, comment out the current code and add other code to run it. If you want to go back to the previous code, you can go back with [Ctrl + Z]. You can use it in various ways, so please use it.

### セルの追加 (Add cell)

セルにコードを書いて、実行すれば、セルは自動追加されていくのだが、実行する前にセルを追加したい時がある。そんなときは上部の+ボタンを押せばよいのだが、Aを押しても、今選択しているセルの上にセルを追加できる。ちなみに、Figure 16のように、ノートブックの上の欄に上下ボタンがある。押すとセルが上下に移動するので、セルを整理したい時には使ってみよう。

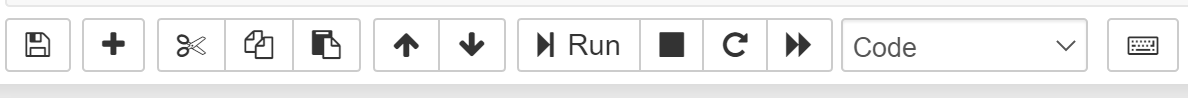


Figure 16:　セルの上下 (Up/down button of cell)

#### If you write code in a cell and run it, the cell will be added automatically, but there are times when you want to add a cell before running it. In such a case, just press the + button at the top. Alternatively, you can press A to add a cell above the currently selected cell. By the way, as shown in Figure 16, there are up and down buttons in the upper column of the notebook. When you press, the cells move up and down, so use it when you want to organize the cells.

# 基本演習 (Basic exercises)



## 基本演算 (Basic operations)

まずは、Pythonを扱うことに慣れてほしい。ということで、基本演算でJupyter Notebookで開発するというのは、どういうことなのか練習しながら慣れよう。演算は後々、衛星データを扱っていく時にも使うので、それを見据えて励んでほしい。数字等を変えてどうなるか自分で試してみて欲しい。



基本演算は数式を入力するだけだ。もし、errorが出てきてパニックになった場合は、2.7章の「errorに慣れよう」を読んでみよう。

To get used to working with Python, use basic operations for practicing what it is to develop with Jupyter Notebook. You will use calculations later when dealing with satellite data, so please work hard with that in mind. Try for yourself what happens by changing the numbers. The basic operation is just to enter the formula as it is. If you get an error and panic, read Chapter 2.7, "Getting used to the error."

|  |
| --- |
| In [1]: 1+3  Out[1]: 4  In [2]: 2\*4  Out[2]: 8  In [3]: 10/4  Out[3]: 2  In [4]: 10./4  Out[4]: 2.5  In [5]: 10%3  Out[5]: 1  In [6]: 2\*\*3  Out[6]: 8 |

【解説】[3]と[4]の違いは小数点の有無であり，どちらにも小数点がない場合は整数として答えが返される。他の言語では、整数なのか実数なのか前もって宣言しないといけない。しかし、Pythonでは自動的に判断してくれるので、その特性が確認できる。 [5]は剰余計算。これは、forを回すときによく使われる。 [6]はべき乗計算。

[Explanation] The difference between [3] and [4] is the presence or absence of a decimal point, and if neither has a decimal point, the return value is an integer. In other languages, you have to declare in advance whether it is an integer or a real number. However, Python will automatically determine that, so you can see its characteristics. [5] is the remainder calculation. This is useful when turning a "for". [6] is a power calculation.

|  |
| --- |
| In [7]: int(12.3)  Out[7]: 12  In [8]: int(10./4)  Out[8]: 2  In [9]: float(10/4)  Out[9]: 2.5 |

【解説】ここで注意してほしいのは、先ほど話した通りほかの言語は前もって型を宣言するが、それはPythonでは不要だが、わざと変換したりもする。int()を使うと整数に変換する。[8]は[4]の結果をint()を使って変換したので結果が整数になっている。float()は小数表示にする関数で[9]は[4]と同じ結果を得ている。

＜参考＞

本書では、演習はこのくらいにするが、もっと高度な演算もある。下記のURLにその他の例もあるので、参照にされたい。

<http://www.tohoho-web.com/python/operators.html>

<http://www.tohoho-web.com/python/operators.html>

　[Explanation] Be aware of the fact that other languages declare the types first, but it is not necessary in Python while conversion occurs purposely. Using Int () converts the value into an integer. [8] used int() to convert the result of [4], therefore the result is in integer.　float () is a function to display decimal numbers, and [9] has the same result as [4].

<Reference>

, The exercises are explained in this manual , but there are more advanced calculations. Please refer to the URL below for other examples.

<http://www.tohoho-web.com/python/operators.html>



## 変数と配列 (Variables and arrays)

### 変数 (Variables)

|  |
| --- |
| In [1]: x1 = 1  In [2]: x1, x2, x3, x4 = 1, 1.1, "1", "abc"  In [3]: x1  Out[3]: 1  In [4]: x2  Out[4]: 1.1  In [5]: x3  Out[5]: '1'  In [6]: x4  Out[6]: "abc"  In [7]: type(x1)  Out[8]: int  In [8]: type(x2)  Out[8]: float  In [9]: type(x3)  Out[9]: str  In [10]: type(x4)  Out[10]: str |

【解説】[1]1つの変数を定義。[2]複数の変数を一行で定義することもできる。[3]~[6]それぞれの変数の値が入っていることを確認。[7]~[10]type()関数は変数の種類を返す関数。[7]は整数(integer)，[8]小数(float)，[9]と[10]は文字列(string)。[2]において""で囲うと文字列と認識される。たとえ、"1"のように数字でも""で囲うと整数ではなく文字列として認識されるので覚えておいてほしい。余談だが、小数は英語でDecimalだ。なぜ代わりにfloatが使われるのか私は知らない。知っている人がいたらぜひ教えてほしい。

[Explanation] [1] Define one variable. [2] Define multiple variables in one line. [3] ~ [6] Confirm the value of each variable. [7] ~ [10] type () function returns to the type of variable. [7] is an integer (integer), [8] is a decimal (float). [9] [10] are strings (string). In [2], it is recognized as a character string when enclosed in "". Even if a number like "1" is enclosed in "", it will be recognized as a letter character string instead of an integer, so keep this in mind. Besides this, the decimal is truly a "decimal" in English. I don't know why it is a "float" instead. If anyone knows, please inform us.

|  |
| --- |
| In [1]: x1 = (1,"abc",3)  In [2]: x2 = [1,"abc",3]  In [3]: type(x1)  Out[3]: tuple  In [4]: type(x2)  Out[4]: list  In [5]: x1[1]  Out[5]: 'abc'  In [6]: x1[0]  Out[6]: 1  In [7]: x1[1] = "xyz"  ---------------------------------------------------------------------------  TypeError Traceback (most recent call last)  <ipython-input-174-96bdbf661f56> in <module>  ----> 1 x1[1]="xyz"  TypeError: 'tuple' object does not support item assignment  In [8]: x2[1] = "xyz"  In [9]: x2  Out[9]: [1, 'xyz', 3] |

### リスト (List)

【解説】まず、list[　]とtuple(　)というものが出てきたが、これらを配列という。高校数学で行列というものをやったと思うが、ああいうものだと思ってほしい。数字などを順番にして、格納していける便利なものだ。[1] " ( ) "で定義するとtuple，[2] " [ ] "で定義するとlistという配列が定義される．[5]pythonでは配列の1個目の要素を0番目と数える。よってx1[1]は1番目ではなく2番目の要素を指定しているので1ではなく"abc"が返される。[7]と[8]tupleでは値の変更ができないのに対して，listでは可能である。

[Explanation] First, list [　] and tuple (　) came out, but these are called arrays. Probably this procession was learned in high school mathematics. It's a convenient way to store numbers in order. [1] a tuple array is defined by"(　)". [2] a list array is defined by"[　]". [5] In Python, the first element of the array is counted as 0th. Therefore, x1 [1] specifies the second element instead of the first, so the return value is "abc" instead of 1. [7] [8] the value in a tuple array can’t be changed, but it can in a list array.

### Numpy

数値計算にはモジュールのnumpyarrayという配列を用いると便利である。行列の扱いは、Pythonの大きな利点であり、重要なのでじっくりと考えて進めてください。

For numerical calculation, the "numpyarray" in the numpy module is useful. Working with matrices is a big advantage of Python and is important, so learn it carefully.

|  |
| --- |
| In [1]: import numpy as np  In [2]: x = np.array([100,10,1])  In [3]: x  Out[3]: array([100, 10, 1])  In [4]: np.array(range(1,10))  Out[4]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])  In [5]: np.arange(1,10)  Out[5]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])  In [6]: np.arange(1,10,2)  Out[6]: array([1, 3, 5, 7, 9])  In [7]: np.ones(5)  Out[8]: array([1., 1., 1., 1., 1.])  In [8]: np.zeros(5)  Out[8]: array([0., 0., 0., 0., 0.]) |

【解説】[1]重要！ numpy（数値計算を効率的に行うモジュール）を読み込む。[2] 1次元配列を作成。[4]range関数を使って、連続する数字を作っている。注意してほしいのは、1から10と指定しているのに、9までしか作られていないことだ。ほしい数に+1することを忘れないでほしい。ちなみに、0から始めて良い場合は、最初の引数を省略できて、range(10)で良い。[5][6]arange関数を使って配列を作成。第一引数が初項，第二引数が末項，第三引数が項差。先ほどより少ないコードで、同じ結果が得られる。後から見直す上でも、処理の速さでも、短いコードのほうが良いので、意識してみてほしい。[4][5]1から9までの連番の配列。[6] 1から9まで、項差2の配列。[7][8]すべて1もしくは0の配列はonesとzerosで作成可能。

[1] Important! Read numpy (a module that performs numerical calculations efficiently.) [2] Create a one-dimensional array. [4] The range function creates consecutive numbers. Note that although 1 to 10 is specified, only 1 to 9 are created. Don't forget to add +1 to the desired number. Also, if starting from 0, omit the first argument and use range (10). [5] [6] Create an array by the arrange function. The first argument is the first term, the second argument is the last term, and the third argument is the term difference. You can get the same result with less code than before. Shorter code is better for reviewing later and for faster processing, so be aware of it. [4] [5] An array of serial numbers from 1 to 9. [6] An array with a term difference of 2 from 1 to 9. [7] [8] All 1 or 0 arrays can be created with ones and zeros.

### 多次元配列 (Multidimensional array)

数値計算を進めるためには多次元の配列（行列）を作成しよう。

Let's create a multidimensional array (matrix) to proceed with numerical calculation.

|  |
| --- |
| In [1]: A=np.array([[ 1, 1, 2], [3, 5, 8], [13, 21, 24]])  In [2]: A  Out[2]: array([[ 1, 1, 2],  [ 3, 5, 8],  [13, 21, 24]])  In [3]: A.size  Out[3]: 9  In [4]: A.shape  Out[4]: (3, 3)  In [5]: A.ndim  Out[5]: 2  In [6]: A.dtype  Out[6]: dtype('int32') |

【解説】[1][2]2次元配列(行列)を作成し、表示。[3] 行列Aのサイズ(要素数)を表示、[4] 行列Aのサイズ(行、列)を表示、[5]行列Aの次元を表示、[6]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)。

[Explanation] [1] [2] Create and display a two-dimensional array (matrix). [3] Display the size (number of elements) of matrix A, [4] Display the size (rows, columns) of matrix A, [5] Display the dimensions of matrix A, [6] Display the numeric type of matrix A (Example: int32 4-byte (32-bit) integer, float64 floating-point 8-byte format).

### 配列の計算 (Array calculation)

上の「多次元配列」に続いて、下記を実施してみよう。

After following the "multidimensional array" above, do the following:

|  |
| --- |
| In [1]: B=A+0.5 　 　行列Aの要素すべてに0.5を足す  In [2]: B 　　行列Bを表示  Out[2]: array([[ 1.5, 1.5, 2.5],  [ 3.5, 5.5, 8.5],  [13.5, 21.5, 24.5]])  In [3]: 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)  Out[3]: dtype('float64')  In [4]: A\*2 　　2倍  Out[4]: array([[ 2, 2, 4],  [ 6, 10, 16],  [26, 42, 48]])  In [5]: 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)  Out[5]: array([[ 1, 1, 4],  [ 9, 25, 64],  [169, 441, 576]])  In [6]: A@A 　　行列の掛け算(上との違いを見てみよう)  Out[6]: array([[ 30, 48, 58],  [122, 196, 238],  [388, 622, 770]])  In [7]: np.dot(A,A) 　　行列の掛け算  Out[7]: array([[ 30, 48, 58],  [122, 196, 238],  [388, 622, 770]])  In [8]: np.transpose(A) 転置（意外と後で使うかも－２次元画像の縦横変換）  Out[8]: array([[ 1, 3, 13],  [ 1, 5, 21],  [ 2, 8, 24]]) |

【解説】[1][2]行列Aの要素すべてに0.5を足したものを行列Bとする。[3]行列Bの数値のタイプを表示、[4]2倍、[5]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）。[6]行列の掛け算([5]との違いを見てみよう)。[7]行列の掛け算。[8]転置（意外と後で使うかも－２次元画像の縦横変換）。

[Explanation] [1] [2] Let matrix B be the sum of all the elements of matrix A plus 0.5. [3] Show the type of numbers in matrix B. [4] Double. [5] Multiply by each element of the matrix (compare with the result of manual calculation). [6] Matrix multiplication (see the difference from [5]). [7] Matrix multiplication. [8] Translocation (might use this later -vertical / horizontal conversion of 2D image).

配列の要素を確認してみよう。

Let's check the elements of the array.

|  |
| --- |
| In [1]: B=A.reshape(-1,) 行列A を1次元配列にする(B)  In [2]: B=np.ravel(A) 別のやり方  In [3]: 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）  Out[3]: 21  In [4]: 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）  Out[4]: 21  In [5]: A[2,:] 2次元表記した場合（配列のどこが表示されたか確認）  Out[5]: array([13,21,24])  In [6]: A[:,1] 2次元表記した場合（配列のどこが表示されたか確認）  Out[6]: array([1,5,21])  In [7]: idx=np.where(B>10) B>10をみたす配列要素を表示する (MATLAB->find)  In [8]: idx 対応する配列の要素番号が返ってくる  Out[8]: (array([6, 7, 8], dtype=int64),)  In [9]: B[idx] 上記で抽出された要素(idx)のみを表示  Out[9]: array([13, 21, 24])  In [10]: idx=np.where(A>10) A>10をみたす配列要素を表示する (MATLAB->find)  In [11]: 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)  Out[11]: (array([2, 2, 2], dtype=int64), array([0, 1, 2], dtype=int64))  In [12]: A[idx] 上記で抽出された要素(idx)のみを表示  Out[12]: array([13, 21, 24]) |

【解説】[1]行列Aを１次元配列にしてBとする、[2][1]の別の方法。[3]１次元表記した場合（配列Bのどこが表示されたか確認）[4] 2次元表記した場合（配列のどこが表示されたか確認）[5][6] ある次元の全体がほしい場合は":"を使うとよい [7] B>10をみたす配列要素を表示する (MATLAB->find)。[8] 対応する配列の要素番号が表示される。[9][8]で抽出された要素(idx)のみを表示。[10] A>10をみたす配列要素を表示する (MATLAB->find)。[11] 対応する配列の要素番号が表示される (Aは2次元なので2次元)。 [12][11]で抽出された要素(idx)のみを表示。

[Explanation] [1] Let the matrix A be a one-dimensional array and be B. [2] Another method of [1]. [3] One-dimensional notation (check where in array B was displayed). [4] Two-dimensional notation (check where in the array was displayed). [5] [6] If you want the entire dimension, use ": ". [7] Display array elements that satisfy B> 10 (MATLAB-> find). [8] Display the element number of the corresponding array. [9] Display only the elements (idx) extracted from [8]. [10] Display array elements that satisfy A> 10 (MATLAB-> find). [11] Display the element number of the corresponding array (2D because A is 2D). [12] Display only the elements (idx) got from [11] are displayed.

### 行列の演算 (Matrix operation)

In [9]: np.linalg.det(A) 　　　　　　　　行列式 Determinant

Out[9]: -20.000000000000007

In [10]: np.linalg.inv(A) 　　　　逆行列 Inverse Matrix

Out[10]: array([[ 2.4, -0.9, 0.1],

[-1.6, 0.1, 0.1],

[ 0.1, 0.4, -0.1]])

In [11]: 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)

Out[11]: array([[ 1.00000000e+00, 0.00000000e+00, 0.00000000e+00],

[-2.22044605e-16, 1.00000000e+00, 0.00000000e+00],

[-2.44249065e-15, 4.44089210e-16, 1.00000000e+00]])

In [12]: [v,lamda]=np.linalg.eig(A) 行列Aの固有値・固有値ベクトル計算

In [13]: v 　　　　　　　　vは固有値

Out[13]: array([31.50348902, 0.34368697, -1.84717599])

In [14]: lamda 　　　　　　　　vは固有ベクトル

Out[14]: array([[-0.07214919, -0.86781198, -0.23523234],

[-0.29569257, 0.49551157, -0.69307077],

[-0.95255467, 0.03702237, 0.68140932]])

【解説】私はまだ実戦で使ったことがないので、解説はしない。こういうものがあるという紹介だけでとどめておく。

[Explanation] This hasn’t been used yet, so this is just an introduction.

### 便利な関数 (Useful function)

|  |
| --- |
| In [1]: testdata = np.array([ 1, 1, 2, 3, 5, 8, 13, 21, 24])  In [2]: np.average(testdata) 平均を求める  Out[2]: 8.666666666666666  In [3]: np.mean(testdata) 平均を求める  Out[3]: 8.666666666666666  In [4]: sum(testdata) 合計を求める  Out[5]: 78  In [5]: np.max(testdata) 最大を求める  Out[5]: 24  In [6]: np.min(testdata) 最小を求める  Out[6]: 1  In [7]: np.mean(testdata[3:5]) 要素3-4の平均を求める  Out[7]: 4.0  In [8]: np.std(testdata)　　　　　　　　標準偏差を求める  Out[8]: 8.259674462242579  In [9]: 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)  In [10]: np.corrcoef(t,testdata) 相関係数を求める  Out[10]: array([[1. , 0.9325953],  [0.9325953, 1. ]])  In [11]: 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  Out[11]: array([ 2.98333333, -6.25 ])  In [12]: p=np.polyfit(t,testdata,1) 結果を変数pに格納  In [13]: p.size 　　　変数pのサイズチェック(double要素では?)  Out[13]: 2  In [14]: p[0] p(0)のチェック  Out[14]: 2.983333333333333  In [15]: p[1] p(1)のチェック  Out[15]: -6.249999999999989 |

【解説】[2][3] 平均を求める。実はこの関数間に違いがあるのだが、私は実感できていない。詳しくは調べてみてほしい。 [4] 合計。よく使うので、覚えておこう。 [5][6] 最大、最小を求める。よく使う。 [7]要素 3-4の平均。 [8] 標準偏差。 [10] tとtestdataの相関係数を求める。 [11] 最小二乗法で直線回帰（傾き、切片）。 [12][11] を変数pに入れる。 [13] 変数ｐのサイズチェック (2要素であると確認) [14][15] p(0), p[1]を確認。結果を変数に格納しておいて、後で使うことはよくあるので、できるようにしておきたい。行列に格納しておいて、[0]などで、指定することもよくある。

[Explanation] [2] [3] Calculate the average. Actually, there should be a difference between these functions, but it hasn’t been noticed. Check it out. [4] Total. This is used a lot, so remember it. [5] [6] Find the maximum and minimum. This too is often used. [7] Mean of elements 3-4 [8] Standard deviation [10] Find the correlation coefficient between t and test data [11] Linear regression (slope, intercept) using the least squares method. [12] puts [11] in the variable p [13] Checks the size of the variable p (confirms that it is two elements) [14] [15] Checks p (0) and p [1]. To store the result in a variable so that it can be used later. It is often stored in a matrix and specified by [0].

### リスト内包表記 (List comprehension)

In [1]: a=[1,2,3]

In [2]:print([x\*3 for x in a])

Out[2]:[3,6,9]

In [3]: print([x\*4 for x in a if x == 3])

Out[4]:[12]

【解説】リストの中身を計算する時には非常に便利。特に、処理が速いので、大量の計算をする時にこれが使えると大幅な計算コストの短縮ができる。例えば、日本の領域で、雨が年間1500ｍ以上降る場所とそうでない場所に分けたとしよう。値としては、1か0になる。１の値を取るindexだけ取得したいとなったら、リスト内包表記の出番だ。もう読者なら書き方がわかるだろう。では、リスト内包表記を使わなかった場合、removeという関数を使うことになる。これは、リスト内の最初の0を消すという関数なので、それをwhileで０が残っている間ずっと消すというコードになる。これだと一回一回最初の０まで計算するので、相当な時間を要する。ということで、リスト内包表記を習得しよう。5章Pythonによるデータの取得方法で登場するので、そこでまた詳しく解説する。

　[Explanation] This is very useful when calculating the contents of the list. In particular, since the processing is fast, this significantly reduces the calculation cost, when performing a large amount of calculation. For example, dividing the area of ​​Japan into places where it rains more than 1500m a year and places where it doesn't, the value can be either 1 or 0. If you want to get only the index that takes a value of 1, it's time for list comprehension. Readers will know how to write it by now. So, if not using list comprehensions, use a function called "remove". This is a function that erases the first 0 in the list, so the code erases 0 the entire time while the 0 remains. In this case, it takes a considerable amount of time to calculate up to the first 0 each time. So learn the list comprehension notation. Since it will be in “Chapter 5 How to get data by Python”, this manual will explain the details there.

## 制御文 (Control statement)

### if

条件分岐(if, elif, else)は次のように表す。

Conditional branches (if, elif, else) are as follows.

|  |
| --- |
| In [1]: a = 10  In [2]: if a==10:print("yes")  else:print("no")  Out[2]: yes  In [3]: a = 10  In [4]: if a>10:print("yes")  else:print("no")  Out[4]: no  In [5]: a = 10  In [6]: if a>10:  print("no")  elif a==10:  print("yes")  else:  print("else")  Out[6]: yes |

【解説】[1]はaを10であると定義して，[2]は10ならば"yes"と表示される。（print文）そうでなければ"no"と表示するように記載している。このときaが10と等しいという条件は"＝"2個で表し、その条件にあてはまる場合には、頭に空白があるブロックが実行され，空白がないと次の条件(else.それ以外)である場合の処理が始まる。[6]複数の条件はelif文を使って表現する。ちなみに、以上や以下は">=","<="を用いる。[1]~[4]では、制御文:実行文だったのが、[6]では制御文:　改行　インデント　実行文という構造になる。制御文の条件を満たしたときに実行文を行いますというのを分かりやすく示しているのが改行とインデントだ。これから実行文が長くなれば重要になるので、インデントを忘れないように。ちなみに、notebookでは、:を打った後にENTERを押せば自動でインデントしてくれるが、能動的にしたい場合はtabキーを押そう。

[Explanation] [1] “a” is defined as 10. [2] if “a” is 10, "yes" is displayed (print statement). Otherwise, "no" will be displayed. At this time, the condition that “a” is equal to 10 is represented by two "=". If that condition is met, the block with a blank at the beginning is run, and if there is no blank, the processing for the next condition (else. Other) starts. [6] "elif" statement is for multiple conditions. By the way, "> =" and "<=" are for the above and below. For [1]~[4], the code shows “control statement: execution statement” while [6] uses such code structure of “control statement: new line/indentation/execution statement”. The reason for having “new line/indentation” is because those indicate the fact that execution statement will run when conditions for control statement have met. When the execution statement gets longer, knowing how to use indentation with new line will be important. By the way, indentation occurs automatically afer you press “:” then “ENTER” in notebook. However, you could press “Tab” key manually and get the same result.

### for

繰り返し処理(for文)は次のように表す。

The iterative process (for statement) is as follows.

|  |
| --- |
| In [1]: for i in range(1,4):  print(i)  Out[2]: 1  2  3 |

【解説】来ました。Pythonを使う理由はこれと言っても過言ではない。forは必須の制御文です。大きなデータを扱う時はこれなしでは、無限に時間があっても足らない。そして、どうforを上手く使えるかがデータの処理速度に影響してくるので、上手い使い方を学んでいってほしい。とは言うが、千里の道も一歩より、一から学んでいこう。for文はあらかじめ決められた回数、処理を実行する命令文である。この場合iという変数がその後ろに記載された条件(range)で指定された数字を順番に入れて、その都度forで括られたブロックのコードを実行する。上の例では[1,2,3]が作成される。ここで注意が必要なのは1から4の1個前（つまり3番目）まで実行されるということである。ここが、間違えやすいポイントである。私は間違えないようにいつも4+1などにしている。

　[Explanation] Here we are. It's no exaggeration to say that this is the reason for using Python. "for" is a control statement that is essential for our field. Without it, when dealing with large amounts of data, infinite time is not enough. And how to use "for" greatly affects the data processing speed, so learn how to use it efficiently. With that said, a journey of a thousand miles begins with a single step. Take this first step. A "for" statement is a statement that runs processing a predetermined number of times. In this case, put the numbers specified by the condition (range) described after the variable "i" in order, and execute the code of the block enclosed by for each time. In the above example, [1,2,3] is created. The point to note here is that it is run from 1 to one before 4 (That is 3). Beginners tend to make a mistake here. I always type 4 + 1 to not to make a mistake.

条件はrangeで作成した配列ではなく、どのような書き方でもよい。

The condition does not have to be an array created by range, and you can write it in any way.

|  |
| --- |
| In [1]: for i in ["abc ", "XYZ ",123]:  print (i)  Out[1]: abc  XYZ  123 |

【解説】ここでは直接、配列で繰り返し条件を指定している。

[Explanation] Here, the repetition condition is directly specified in the array.

入れ子状に制御文を書くことも可能。

You can write control statements in a nested manner.

|  |
| --- |
| In [1]: for i in range(1,10):  if i == 5:  print(i)  Out[1]: 5 |

【解説】１から9までのうち5に該当する場合にのみprintが実行される。インデントに気を付けてほしい。二つ目の：の後は、２回分のtabが入る。

[Explanation] Of 1 to 9, print is run only when it is equal to 5. Be careful about indentation. After the second colon “: ”, there are two tabs.

### while

繰り返し処理(while文)は次のように表す。

Loop statements (while statement) is as follows.

|  |
| --- |
| In [1]: x = 0  In [2]: while x<3 :  　　　　 print(x)  x += 1  Out[2]: 0  1  2 |

【解説】xが3未満の間はprintが実行される．"x +=1" は "x = x+1" と同じであり，それまでのｘに１を加えたものを新しくxとするという命令文である。数学的に考えるとxがx+1と等しいと考えてしまうが，ここでは等しいという意味ではなく，代入（定義）として=を使っているといえる。等しいという条件は"= ="のように等号を2つ並べて表される。

[Explanation] "print" is run while x is less than 3. "x += 1" is the same as "x = x + 1", and it is a statement that the previous x plus 1 is the new x. Mathematically, we think that x is equal to x + 1, but here it does not mean that it is equal, but = is an assignment (definition). The condition of equality is shown as two equal signs "= =".

### def

制御文ではないのだが、defという便利なものがある。後々コードを整理していくときに必要不可欠なので、ここで説明しておく。

Although it is not a control statement, there is a convenient one called def. It is essential when organizing the code later, so here is the explanation.

In [1]: def readbook(d):

v=“朗読を始めます”

print(v)

In [2]: readbook(1)

Out[3]: 朗読を始めます

defというのは、関数を作る制御文みたいなものだ。今回は、readbookという関数を作っている。関数なので引数、つまり()が必要なのだが、上のように必ずしも変数を使わなくてもよいのだ。例えば、前の結果を関数の中で使いたいときには、dをdef内で使ってあげればよい。

このdefの大きな特徴として、def内で定義した変数は、defの内部でしか定義されない。どういうことかというと、def外でシンプルにvと打って、実行してみてほしい。定義されてないとエラーが出るだろう。これはメリットでもデメリットでもあるので、よく覚えておいてほしい。

A def is like a control statement that creates a function. This time, the function is named readbook. Since it is a function, it needs an argument, that is (). However, it does not always necessarily have to be a variable as shown above. For example, if you want to use the previous result in a function, you can use d in def.

A major feature of this def is that the variables defined inside the def are defined only inside the def. What does that mean? Simply type v outside the def and run it. You will get an error that it is not defined. This is both a merit and a demerit, so keep that in mind.

### 制御文を複数実行する時 (When running multiple control statements)

制御文を扱う上で、インデントの存在は重要である。ノートブックでは、tabキーを押せばインデントできるので、利用していこう。Python では特別な文字を使わずに同じインデントがされている文を同じブロックとして扱う。言葉だけでは、わかりにくいと思うので、Figure 17 を用意した。コードは、forでiとjが回されていて、ifで場合分けがされている。ブロック分けによって、実行される順番が決められている。rangeがそれぞれ2なので、それぞれ二回実行される。3rdブロックの場合分け処理が２ブロックのforの２回分実行され、それがさらに１ブロックの２回分実行される。それは、樹形図で表すことができるので、Figure 17の右の樹形図のようになる。iとjがそれぞれ２回ずつで、計四回、３ブロックが実行されるという仕組みになっている。後は、使いながら慣れてもらいたい。私の目安だが、カンマ(:)が出てきたら、そのあとはインデントが入るので、必ず余白に気を配ろう。

Indentation is important in control statements. In notebooks, pressing the tab key indents the code, so use it. In Python, instead of separating blocks by special characters, statements with the same indentation are recognized as the same block. It's difficult to understand with just words, so Figure 17 is prepared. In the code, the "for loop" manages i and j, and "if" makes a conditional branch. The order of execution is decided by block division. Since each range is 2, each "for loop" runs twice. Case-by-case processing in 3rd block runs twice which is ordered by the "for loop" in the 2nd block. Then, the two runs run twice again because of the "for loop" in the 1st block. The diagram on the right of Figure 17 explains this. i and j run twice each, and three blocks run four times in total. Get used to it through using it several times. My personal guide is that the comma (:) is followed by indentation, so don't miss the indentation.

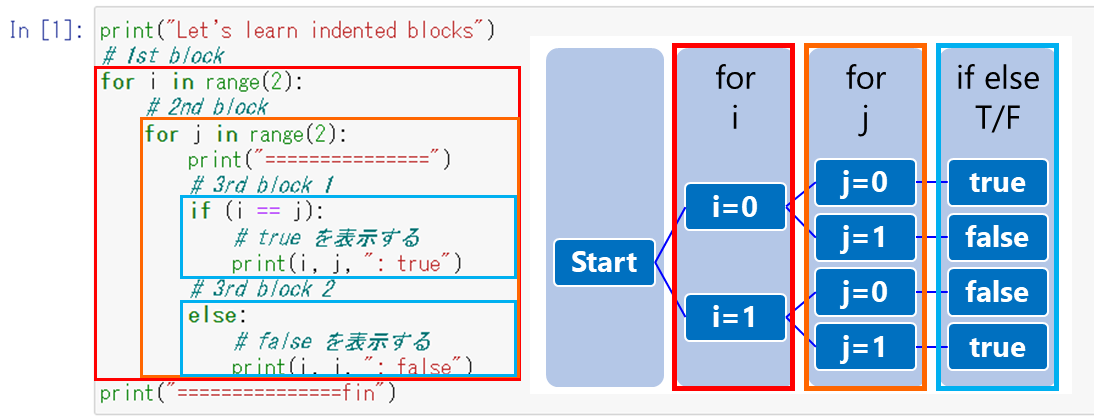


Figure 17: Pythonの改行されたブロック (Indented blocks in Python)

## ファイルの入出力 (File input / output)

4.3章にて具体的なデータ解析（テキストファイル、バイナリファイル）について述べるので、ここではテキストファイルの入出力について簡単に紹介する。なお、"lesson2-1.csv"などのファイルは準備していないので、今のところは、この様に入出力を行うのだ、という点を理解してもらえればよい。実際に下記の作業を実行すると、エラーが出力される。

This manual will explain specific data analysis (text file, binary file) in Chapter 4.3, so here is a brief introduction of the input and output of text files. However, a file such as "lesson2-1.csv" hasn’t yet been prepared, so for now, understand that I / O is done in this way. If you actually run the following work, an error will be displayed.

|  |
| --- |
| In [1]: x = np.loadtxt("lesson2-1.csv", delimiter=",", dtype=("float"))  In [2]: x  Out[2]: array([[ 1.2, 3.2, 3.9, 4.2],  [ 5.9, 2.7, 6.6, 9.1],  [ 3.3, 4.1, 1.5, 8.2]]) |

テキストファイルの入力（ファイルの中身がすべて数値の場合）。

Input of text file (when the contents of the file are all numerical values).

【解説】 delimiterは元のファイルの区切り文字を指定している。ここでは開こうとするファイルが数字のみなのでデータ形式(dtype)をfloatとして小数で一括して読み込むことができた。ファイルに存在するデータの形式が一定でない場合は工夫が必要である。

Here, since there are only numbers in the opened file, the data format (dtype) was set to float and it was possible to read all at once with decimal numbers. If the data format of the file is not constant, you will need an extra process.

少し強引な方法としてはすべて文字列として読み込んで，数値に変換するという方法がある。

A slightly brute force method is to read everything as a character string and convert it to numbers.

|  |
| --- |
| In [1]: x = np.loadtxt("lesson2-2.csv",delimiter=",",skiprows=1,  dtype=("S1"))  In [2]: x  Out[2]:array([[b'a', b'1'],  　　　　　　[b'b', b'2'],  　　　　　　[b'c', b'3']], dtype='|S1')  In [3]: x1 = x[:,0]  In [4]: x2 = x[:,1].astype("float")  In [5]: x1  Out[5]: array([b'a', b'b', b'c'], dtype='|S1')  In [6]: x2  Out[6]: array([ 1., 2., 3.]) |

【解説】 xに文字列としてファイルを読み込み（skiprowsで頭の数行を読み飛ばしている），その後にx1に1列目，x2に2列目を小数に変換(astype("float"))したものを代入する。

[Explanation] Read the file as a character string in x (skiprows skips the first few lines), then substitute x1 for the first column and x2 for the second column converted to decimal numbers (astype("float")).

テキストファイルへの出力（ファイルの中身がすべて数値の場合）。

Output to a text file (when the contents of the file are all numerical values).

|  |
| --- |
| In [1]: np.savetxt("test.txt", x, delimiter=",", fmt="%f") |

## グラフの描画 (Drawing a graph)

Pythonには、matplotlibと呼ばれるライブラリがあり、綺麗な絵を簡単に作成できる。ここでは、基本的な作図方法を見ていく。

Python has the matplotlib library for easily creating beautiful pictures. Here, we will look at the basic drawing method.

In [1]: import matplotlib.pyplot as plt

In [2]: import numpy as np

In [3]: t = np. array(range(1,11))

In [4]: t = np.arange(1,11,2)

In [5]: x = np.arange(1,11)

In [6]: y = np.arange(1,11)

In [7]: plt.plot(x,y)

In [8]: plt.show()

【解説】[1][2]matplotlib（作図などのモジュール読み込み）[3][4]xとyの配列を作成している。[4]は、[3]のコードを短くして、配列の数字も２個飛ばしにしている。[5][6]x, yをプロット。[7]はｘとｙをグラフ化している。Figure 18のように表示されるだろう。[8]はpltがここで終わりという指示。

[Explanation] [1] [2] matplotlib (loading modules such as drawing) [3] [4] Create an array of x and y. [4] the code in [3] is shortened and the numbers in the array are skipped by two. [5][6] Plot x and y. [7] Graphs x and y. It should look like Figure 18. [8] Indicates that plt ends here.

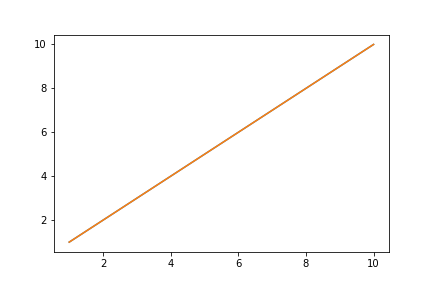


Figure 18:　線グラフ (Line graph)

In [8]: plt.plot(x,y,’\*’)

plt.show()

In [9]: plt.plot(x,y,’\*-’)

plt.show()

In [10]: plt.plot(x,y,’\*-’)

plt.xlabel(“Time”)

plt.ylabel(“Photosynthesis”)

In [11]: plt.plot(x,y,’\*-’)

plt.xlabel(“Time”)

plt.ylabel(“Photosynthesis”)

plt.savefig(“test.png”)

plt.savefig(“test.pdf”)

【解説】[8][9]プロットのマーカー、線のスタイルを確認してみよう。[10]x軸、y軸のラベル表示。[11]画像ファイルとして保存（ファイル名: test.png、test.pdf）、ファイルの中身を確認してみよう。

[Explanation] [8] [9] Let's check the plot markers and line styles. [10] Label display of x-axis and y-axis [11] Save as an image file (file name: test.png, test.pdf), and check the contents of the file.

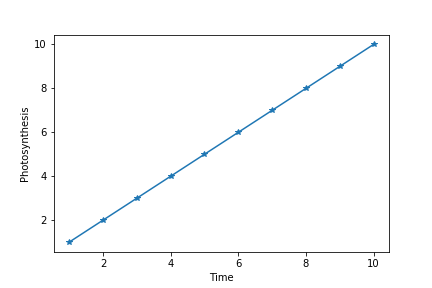


Figure 19:　線グラフ２ (Line graph 2)

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…。

Continue to try as shown below and see what happens...

In [12]: y2=y\*y

In [13]: plt.plot(x,y,"\*-",x,y2,"o-")

In [14]: plt.plot(x,y,"\*-",x,y2,"o-")

plt.xlim(1,13)

plt.ylim(1,120)

plt.show()

【解説】[13][14]1つのグラフエリアに2つのデータがプロットされる。[14]x軸、y軸の範囲を設定。

[Explanation] [13][14] Plot two data in one graph area. [14] Set x-axis and y-axis ranges.

2軸グラフを作成しよう。

Create a 2-axis graph.

In [1]: fig, ax1 = plt.subplots()

ax1.plot(x,y,"\*-")

ax2 = ax1.twinx()

ax2.plot(x, y2, "o-")

plt.show()

【解説】\*1 \*2 (x,y)のプロット、\*3 \*4(x,y2)のプロット

[Explanation] Plot \* 1 \* 2 (x, y), plot \* 3 \* 4 (x, y2)

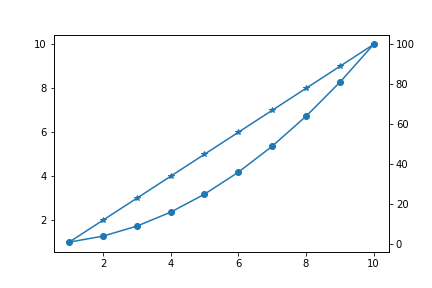


Figure 20:　線グラフの二重プロット (Double plot of line graph)

### グラフ描画の参考資料 (Reference material for graph drawing)

いくつか参考になるURLを用意したので、ぜひグラフを書くときには参考にしてほしい。

Some URLs are prepared as references, so refer to them when drawing graphs.

<https://qiita.com/ceptree/items/5fb5e9e6f29d214153c9>

<https://qiita.com/skotaro/items/08dc0b8c5704c94eafb9>

<https://qiita.com/kenichiro_nishioka/items/8e307e164a4e0a279734>

こちらには、色々なグラフが載っている。自分の書きたいスタイルに合う画像を選んでそこからコードを取ってくれば、自分の好きなものが書けてしまう。ぜひ、活用してもらいたい。

There are various graphs here in the link below. Select an image that suits your style and take the code from it, then write what you like. Make use of it.

<https://matplotlib.org/gallery.html>

### グラフの構造 (Graph structure)

これから複数のグラフを一つの画像として描いてもらう。その場合には、グラフの構造を理解してもらう必要がある。グラフを描くmatplotlibは下記のような階層構造を持っている。

From now on, multiple graphs will be drawn as one image. In that case, it is necessary to understand the structure of the graph. Matplotlib, which draws graphs, has the following hierarchical structure.

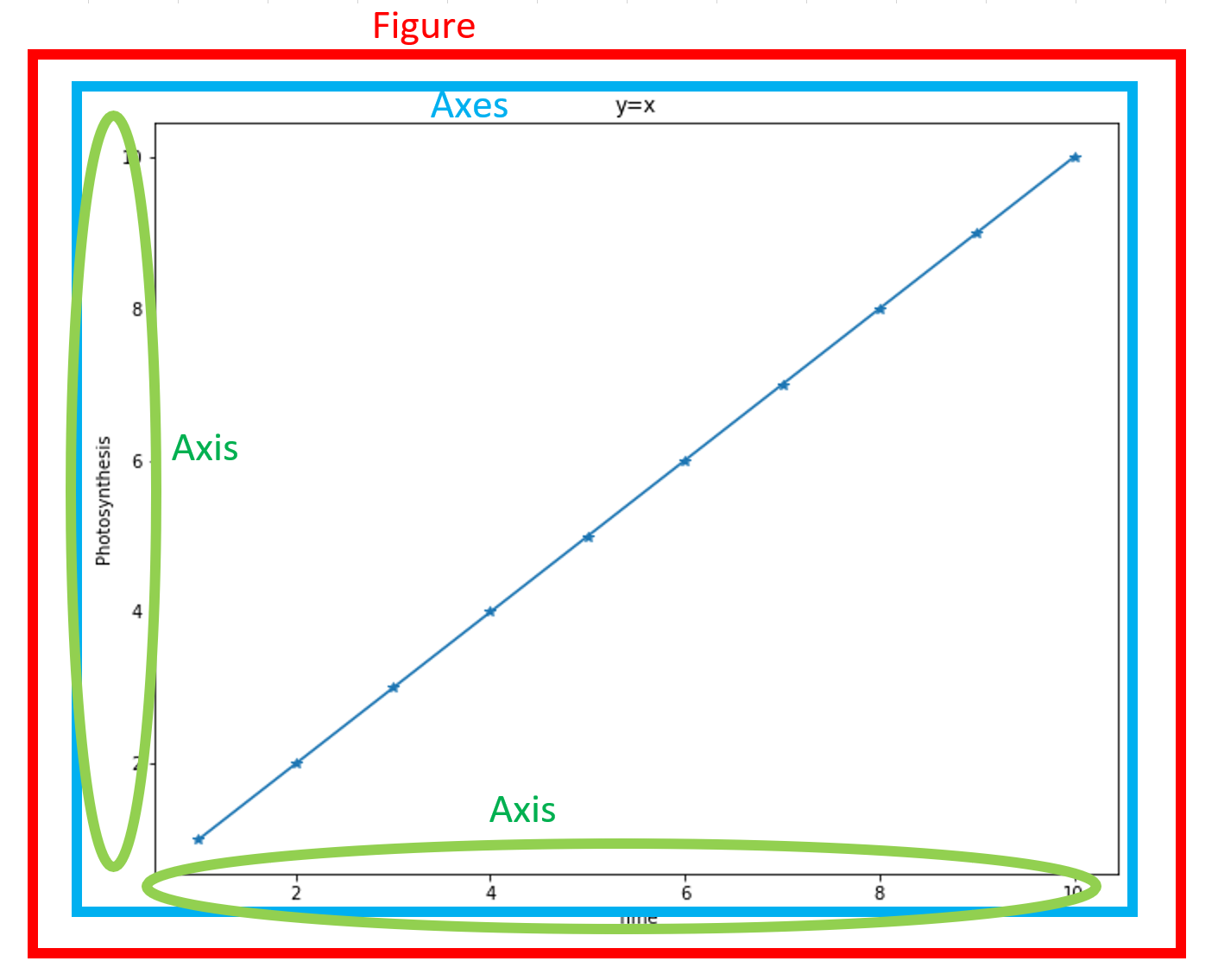


Figure 21:　グラフの構造 (Graph structure)

Figureで全体の描画領域を確保し、その中にAxes領域を確保する。さらにその中の下部や左右にグラフの軸(Axis)を確保している。これを念頭に置いておいてほしい。

Allocate the entire drawing area by the Figure, and allocate the Axes area in it. Furthermore, allocate the axis of the graph to the bottom and left and right in the Axes area.

次に複数のグラフを表示してみよう。

Next, display multiple graphs.

In [1]: import matplotlib.pyplot as plt

In [2]: import numpy as np

In [3]: x=np.array(range(1,11))

In [4]: y=np.array(range(1,11))

In [5]: 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()

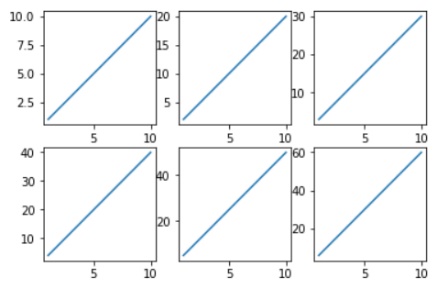


Figure 22:　サブプロット (Subplot)

結果はこの通りである。一枚の画像に複数のグラフを表示させたい時はこのコードを使ってほしい。それでは、参考程度に、かっこいいグラフを一つ紹介する。

The result is above. Use this code when wanting to display multiple graphs on one image. Then, for reference, here is one cool graph.

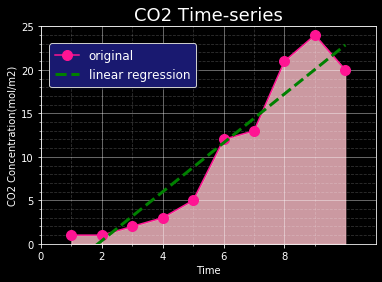


Figure 23:　かっこいいグラフ (Cool graph)

スタイルをいじれば、こんなグラフも書けるようになる。参考までに、ソースコードを以下に示す。

If you play with the style, you will be able to draw such a graph. For reference, the source code is below.

#明暗反転 (Light-dark reversal)

plt.style.use('dark\_background')

testdata = np.array([1,1,2,3,5,12,13,21,24,20])

t = np.array(range(1,11))

p = np.polyfit(t,testdata,1)

fig, ax = plt.subplots(facecolor='midnightblue')

ax.plot(t, testdata, 'o-', color="deeppink", markersize=10,

label="original")

ax.plot(t, t\*p[0]+p[1], '--g', linewidth=3, label="linear regression")

ax.fill\_between(t, testdata, color="pink", alpha=0.8)

ax.set\_xlim(0,11)

ax.set\_ylim(0,25)

#legend and title

fig.legend(loc="upper left", bbox\_to\_anchor=(0.1,0.85),

borderpad=0.5, shadow=True, ncol=1, fontsize=12, facecolor='midnightblue')

ax.set\_title('CO2 Time-series', color='white', fontsize=18)

#x axis

major\_xticks = np.arange(0,10,2)

minor\_xticks = np.arange(0,10,1)

ax.set\_xticks(major\_xticks)

ax.set\_xticks(minor\_xticks, minor=True)

ax.set\_xlabel('Time')

#y axis

major\_yticks = np.arange(0,26,5)

minor\_yticks = np.arange(0,26,1)

ax.set\_yticks(major\_yticks)

ax.set\_yticks(minor\_yticks, minor=True)

ax.set\_ylabel('CO2 Concentration(mol/m2)')

#grid

ax.grid(ls="-", which='major', alpha=0.5)

ax.grid(ls="--", which='minor', alpha=0.2)

#save as png

save\_dir = r‘C:\Users\kotsuki-lab-W02\Documents’

plt.savefig(save\_dir+"/"+'cool\_graph.png',bbox\_inches="tight")

plt.show()

ここで一つポイントがある。PATHについての説明だ。**エラー! 参照元が見つかりません。**章で出てきたように、「cool\_graph.png」だけだと、現在いる場所に保存する。これが相対PATHだ。相対、つまり現在位置によって場所情報が変わる。それに対して、今回のコードは絶対PATHと呼ばれるものを使っている。「C:\Users\kotsuki-lab-W02\Documents\cool\_graph.png」これが絶対PATHだ。ファイル名（cool\_graph.png）の前に場所（C:\Users\kotsuki-lab-W02\Documents\）が書かれている。エラーが少なくなるので、どこにいても動作する絶対PATHをなるべく使おう。

上のコードでは、save\_dirという変数を使っている。今回は一回しか使われないのであまり意味がないが、複数saveする場合は、何回も絶対PATHを打つのは手間である。なので、場所情報を関数でおいて、関数＋ファイル名で保存するようにしている。

r‘C:\Users\kotsuki-lab-W02\Documents’で、シングルクォーテーションの前にrがあるが、これはwindows用だ。‘C:\\Users\\kotsuki-lab-W02\\Documents’実はこれと同じ意味である。しかし、いちいち打つのは手間である。コピペした後、同じ意味で使えるrだけを打っている。ちなみに、MacとLinuxはPATHが/data02/user/というようにスラッシュで区切られるので、rはいらない。Windowユーザーは気を付けよう。

Here is an important fact, “PATH”. As shown in Chapter 2.5, if there is only “cool\_graph.png”, the file will be saved at your current location, which is called a relative PATH. A relative PATH means that your location information changes depending on where the file is. In contrast, an absolute PATH, 「C:\Users\kotsuki-lab-W02\Documents\cool\_graph.png」, is used here. File location （C:\Users\kotsuki-lab-W02\Documents\） is mentioned before the file name （cool\_graph.png）. In order to reduce error, try using an absolute PATH as they work better anywhere.

A variable called “save\_dir” is used in the previous code. It is only used one time here, but this function is useful for saving files multiple times and avoiding typing an absolute PATH each time. Therefore, it can be saved as “function + file name” if your location information is replaced with function.

For r‘C:\Users\kotsuki-lab-W02\Documents’, the reason for having “r” in front of a single quation is for Windows users. It is technically the same meaning as ‘C:\\Users\\kotsuki-lab-W02\\Documents’. However, typing all these file names out is time-consuming, so having “r” at the begging of your file location will help to have the same function afer copying and pasting. By the way, there is no need for using “r” for Mac and Linux users as their PATH is deliminated by slash as /data02/user/. Be cautious with your PATH.

## scriptによる命令実行（復習）(Command execution by script (review))

3章からはスクリプトでの実行も想定して、コードを書いていくので、簡単なものから慣れていこう。先ほどのグラフのコードで練習してみよう。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。

From Chapter 3, the code will be written assuming execution with a script, so get used to the simple ones. Practice with the code in the graph above. In Python, writing a series of commands in a file writes a script (program). For example, use a text editor to create a sample1.py file with the following contents.

以下のテキストを持つファイルを作成する。　名前： sample1.py

Create a file with the following text name: sample1.py

#PYTHON sample program: sample1.py

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)

コマンドプロンプトを起動し、1章で説明したとおりに実行してみよう。

Start a command prompt and run it as explained in Chapter 1.

>cd C:\Users\kotsuki-lab-W02\Documents\python\_training

>python sample1.py

こんな感じになるだろう。

ところで、先ほどのコードは長いと感じないだろうか？私は同じものを何度もやっていて、長いと思う。そんな時は、forを使って同じことの繰り返しを省略できる。

It would look something like the above.

By the way, having to repeat the same thing over and over I think it's too long. In such a case, use "for" to omit the repetition of the same thing.

#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()

for i in range(0,6)として、i=0から5まで 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。

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)。

As for i in range(0,6), the above is the loop from i=0 to 5 (starting from 0, in 1increments, not including 6). In each loop, subplot () and plot () are run, and the result is the same as the previous graph.

Also, for example, if you want to go two by two, write "for i in range (0,6,2) " (run in the order of 1, 3, 5).



## Pythonの間違えやすい落とし穴 (Common mistakes in Python)

### Errorに慣れよう (Get used to Error)

どうだろう？2章でコードを打っていて、errorには、出会っただろうか？絶賛パニック中だったら、一旦落ち着こう。プログラミングをやる上でerrorは避けて通れない。必ず思わぬ間違いが出てくる。しかし、それは普通のことなので、安心してほしい。これから、errorと出会った時にどうすればよいのかを教えていく。

How did it go? Did you come across an error while typing the code in Chapter 2? If you're in a panic, calm down. Error is inevitable when programming. There will always be unexpected mistakes. But that's normal, so don't worry. From now on, instruction will be given as what to do when an error occurs.

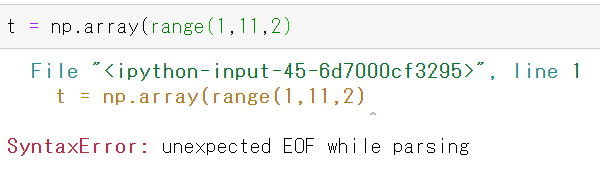


Figure 24:　括弧( )が足りないエラー (Error when parentheses ( ) are missing)

よくあるのが、こういうerrorだ。Errorメッセージであるunexpected EOF while parsingで調べてみれば、Figure 25:のように、回答が載っているサイトが結構ある。Pythonを付け足してあげれば、Pythonの対処方法が出てくるので、そこを読んで調べてほしい。

This is a common error. Search for the Error message “unexpected EOF while parsing”, quite a few websites with answers, as shown in Figure 25:, will be found. If word “Python” is added in your search, there will be an answer for using Python, read it and find out.



Figure 25:　エラー対処の検索 (Search for error handling)

### Pythonは０から始まる言語 (Python is a 0-based language)

先ほど、listを数える時に、Pythonでは０から始まると話した。実は言語によって、０から数え始める言語と１から数え始める言語がある。

<https://ja.wikipedia.org/wiki/%E3%82%AA%E3%83%AA%E3%82%B8%E3%83%B3>

これはオリジンの問題である。ウィキにも載っている様に、時間の言葉にもこの問題がある。例えば、年月日は１オリジンで、時分秒は０オリジンである。つまり、プログラミング言語の話だけではないようだ。

プログラミング言語ではC言語などは０オリジンで、FORTRANは１オリジンである。

<https://jp.quora.com/%E3%83%97%E3%83%AD%E3%82%B0%E3%83%A9%E3%83%A0%E3%81%A7%E3%81%AF%EF%BC%90%E3%81%8B%E3%82%89%E6%95%B0%E3%81%88%E3%82%8B%E3%81%AE%E3%81%AF%E4%BD%95%E6%95%85%E3%81%A7%E3%81%99%E3%81%8B>

どうやら、コードを書く上で、メリットがあるようだが、私はほかの言語を使ったことがないので、こういうものと思っているが、ほかの言語を使ったことがある人はどう思っているのだろうか？

話を戻そう。０からなのか１からなのかは、我らにはどうでもよいことだ。Pythonは０から数え始める。そして、０と１の問題は混在していることが重要な事実だ。つまり、数え間違えて、エラーを起こすことがよくあるのだ。これを防ぐためには、毎回確認するしかない。どうか、エラーが起きないように毎回試しながら、数字を追って行ってほしい。

As mentioned earlier, when counting the list, Python starts from 0. There are languages that start counting from 0 and others that start counting from 1.

[Zero-based numbering - Wikipedia](https://en.wikipedia.org/wiki/Zero-based_numbering)

The issue here is about “origin.” As seen on the wiki, the words of time also have this problem. For example, the date is 1 origin and the hour, minute, and second are 0 origin. So, it's not just about programming languages.

In programming languages, C language has 0 origin and FORTRAN has 1 origin.

<https://jp.quora.com/%E3%83%97%E3%83%AD%E3%82%B0%E3%83%A9%E3%83%A0%E3%81%A7%E3%81%AF%EF%BC%90%E3%81%8B%E3%82%89%E6%95%B0%E3%81%88%E3%82%8B%E3%81%AE%E3%81%AF%E4%BD%95%E6%95%85%E3%81%A7%E3%81%99%E3%81%8B>

Apparently, there are benefits to writing code, but having never used any other language, it's hard to imagine. What do people think if they have used other languages?

Back to the topic. It doesn't matter whether it begins from 0 or 1. Python starts counting from 0. It is important to know that the problems of 0 and 1 are mixed. Therefore, miscounting and making errors are common. The only way to prevent this is to check every time. Follow the numbers, trying each time to avoid errors.

### 配列が上書きされてしまう恐ろしい罠 (A terrifying trap that the array is overwritten)

もうすでに経験している方がいたら、おそらく気を付けながらやっていると思うが、これはよくある恐ろしいエラーだ。とりあえず、例を見てみてほしい。

Those who have already experienced this common and horrifying error are probably using caution. Take a look at an example.

a=np.arange(20)

a=a.reshape(4,5)

b=a[1:3]

b[1]=1

a

これを実行すると、

And run the code above…

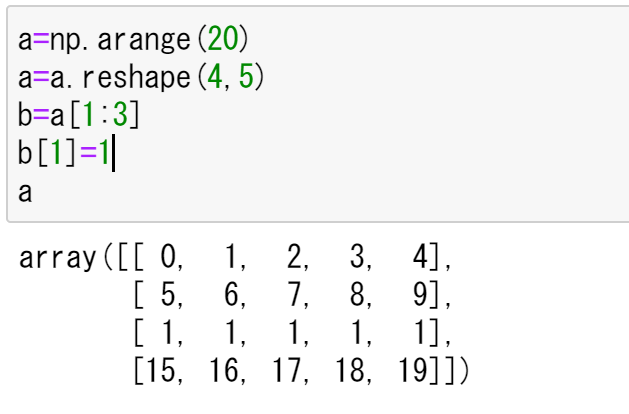


Figure 26:　配列が上書きされてしまう恐ろしい罠

(A terrifying trap that the array is overwritten)

こんな結果が出る。aを定義して、aの一部を取ってきて、bとしたとしても、bを編集してしまうと、aの一部も編集されてしまうという例だ。実はこれを回避する方法がある。

The result is above. In this example, when “a” is defined and a part of “a” is assigned to “b”, if “b” is edited, a part of “a” is also edited. Actually, there is a way to avoid this.

a=np.arange(20)

a=a.reshape(4,5)

b=np.copy(a[1:3])

b[1]=1

a

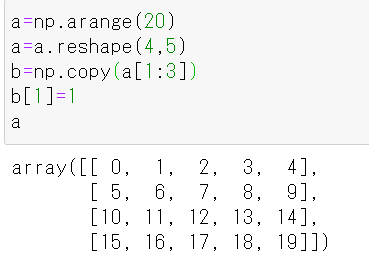


Figure 27:　罠の回避方法 (How to avoid traps)

実は、先ほどの間違いは = を使ったことだ、しかし、np.copyを使うことで、上書きされてしまう問題を回避できる。なので、なるべく使うようにしよう。ちなみに、基本演算でも回避できる。例えば、１をかけてあげれば、回避できるので、こちらも選択肢として考えてみてほしい。このPythonの特性により便利な += や \*= などが使えるので、批判だけはできない。

In fact, the mistake made earlier was the use of “=”, but np.copy avoids the overwritten problem. So, use it as much as possible. In addition, basic operations can avoid this trap too. For example, it can be avoided by multiplying by 1. Remember this as an option as well. Don’t just criticize this issue, as useful function of += and \*= are available in Python.

### コピペする時の注意点 (Caution when Copying & Pasting)

print(“コピペする時は注意”)

例えば、この上のコードをcontrol+cでコピーして、ノートブックにcontrol+vで貼り付けする時、エラーが出る。

For example, when copying the code above with control + c and pasting it into notebook with control + v, an error will be shown.

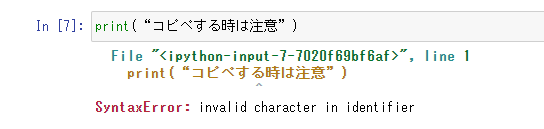


Figure 28:　"　" の書式がwordとノートブックで違う問題

(Problem that the format of "　"is different between word and notebook)

こんなエラーが出るだろう。これの解決方法はシンプル。エラーは"　"のところに出ている。おそらく、文字の書式に問題、wordの形式とノートブックの形式が違うのだろう。したがって、ノートブック内で打ち直せばよい。

You will get an error like this. The solution is simple. The error appears in "　". Perhaps there is a problem with the character format, and the word format and the notebook format are different. Therefore, retype it in notebook.

### 拡張子を表示させよう (Display filename extension)

データサイエンスでは、拡張子というのがとても重要だ。Pythonでファイルを読み込む時は、必ずそのファイルの場所と名前を指定する。その時の名前とは名前＋拡張子なのだ。では、拡張子とは何か？例えば、Pythonコードの拡張子は.pyだ。ｔool.pyというファイルがあった時に、toolだけでは読み込めない。.pyまで必要なのだ。

もう一つ拡張子が大事な理由がある。これから、データを扱っていく時に、データがどんな風に格納されているかという情報が拡張子になる。例えば、ＷＥＢサイトで.ncという拡張子のファイルと、.grdの両方が提供されている場合、拡張子によって、読み込みの方法が変わってくる。合致しない方法で読み込むとエラーが起こる。なので、拡張子が見えないとデータサイエンスは始まらない。

拡張子を見えるようにするのは簡単だ。エクスプローラーで表示をクリックして、ファイル名拡張子のところにチェックをいれよう (**エラー! 参照元が見つかりません。**)。

In data science, filnename extension is very important. Specifying file location and name is a must when reading files in Python. “Name” used here refers to “filename + extension”. By the way, what is “filnename extension”? For example, “.py” is the filename extension for Python. Reading “tool.py” file isn’t possible with only reading “tool” without “.py”.

Another important reason why filnename extension is necessary here is because of the information about how your data is stored, which is written in filename extensions. For example, how to read data provided on a website between .nc and .grd files depends on the type of filename extension. If the data is read with wrong extension, there will be an error. Therefore, it is impossible to deal data science without seeing your filename extensions.

It’s easy to display filename extension. Open “File Explorer”, select the “View” tab, then enable “File name extensions” (Figure 29).

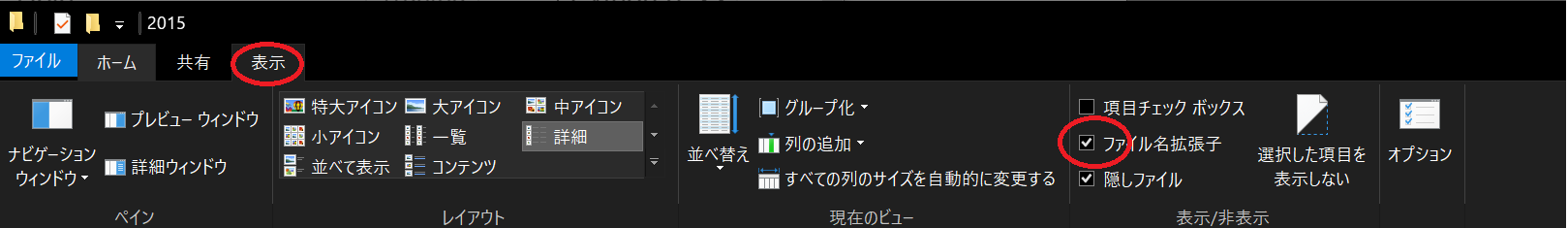


Figure 29: 拡張子を表示させよう (Display filename extension)

# Pythonによるテキストデータの解析 (Text data analysis with Python)



## 解析とは何か？ (What is analysis?)

いよいよ、全米が待ち望んでいた、実際に研究で使われるデータを用いてのデータ解析に入ってゆく（拍手）。2章で、ある程度配列の扱いや、グラフの書き方などを身に着けてもらったと思う。それを実際のデータ解析の流れで使いながら、再確認しようというのが本章の目的である。では、実際に研究に使われるデータとは何か説明しよう。本章では、テキストファイルのデータを扱う。例えば、気象データなどはこの形式で提供されることが多い。気温や湿度などのデータだ。しかし、今回扱うデータはテキストファイルなのだが、その中身はバイナリデータという面白い形式のものだ。これを扱おうと思ったのは、よく提供される気象データも扱えるようにしつつ、4.1章で扱うバイナリデータにも対応できるようにするというのが目的だ。衛星データを扱う場合は、NASAやJAXAなどからバイナリデータで提供されることが多い。本章では、バイナリデータを扱う前に、読者に馴染みがあるだろうテキストファイルで慣れてもらうというのが趣旨となる。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。

この章から、ほぼすべてスクリプトで表現するが、開発はノートブックでしよう。もう一つポイントとなるのが、for文だ。繰り返しの作業が増えてくるので、forを効率的に回すためのリストが多用されるので、コードは複雑になる。頑張って慣れてほしい。

Finally, the data analysis with the data for actual research that everyone has been waiting for will be explained (applause). In Chapter 2, it was taught how to handle arrays and how to draw graphs to some extent. The purpose of this chapter is to apply them to actual data analysis and reconfirm them. Now the data for actual research will be explained. This chapter deals with text file data. For example, meteorological data such as temperature and humidity are often in this format. However, although the data this time is a text file, the contents are in an interesting format called binary data. The purpose of learning binary data is to be able to handle common weather data and to be able to handle it better for Chapter 4.1. NASA and JAXA often publish satellite data in binary data. In this chapter, get used to simple text files before dealing with binary data. The goal is to be able to download observation data (text format) available on the Internet, process it, and create meaningful results (statistical calculation and graph drawing).

From this chapter, almost everything is shown with a script but develop it in notebook. Another point is the "for" statement. The code gets complicated because it requires more repetitive work, and there are many lists to run "for loop" efficiently. Do your best and get used to it.



## 解析の工程 (Analysis process)

それでは、データ解析に移る。Pythonに限らず、一般的なデータ解析では、

Figure 30:　解析のステップ (Steps for analysis)

というステップを踏む。今回は、取得するデータは、形式を変えずに使えるので、データの形式変換は省略する。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…)。

Now moving on to data analysis. Not limited to Python, but general data analysis follows the steps above. This time, it is not necessary to convert the format of the acquired data, so omit the format conversion of the data. Before the actual analysis, acquiring, processing, and reading the data are necessary. The format of the data depends on the agency distributing the data and the type of data, therefore it is necessary to convert it to the format appropriate for the software (format conversion). In fact, in work and research situations, a great deal of time is spent on formatting data (one theory mentions that about 80% of your research time is used for prepping data...).

## 積算棒グラフの作成 (Creating an integrated bar graph)

### データの取得 (Data acquisition )

積算棒グラフの作成を目標にしてもらう。先ほどのステップ通りに進めると、まずは、データの取得をしてもらう。今回は簡潔に手でダウンロードしてもらう。実は、Pythonを使って、一括でデータをダウンロードすることもできるのだが、それは5章にて扱う。今回扱うデータは、

<https://www.nies.go.jp/REAS/index.html#data%20sets>

このURLにあるので、コピーして、WEBブラウザに貼り付けよう。

The goal is to create an integrated bar graph. Follow the steps above to get the data. Download it manually this time. Actually, Python can also download data in bulk, but that is in Chapter 5.

The data for this time is at the following URL, copy it and paste it into your web browser.

<https://www.nies.go.jp/REAS/index.html#data%20sets>

<https://www.nies.go.jp/REAS/index.html#data%20sets>

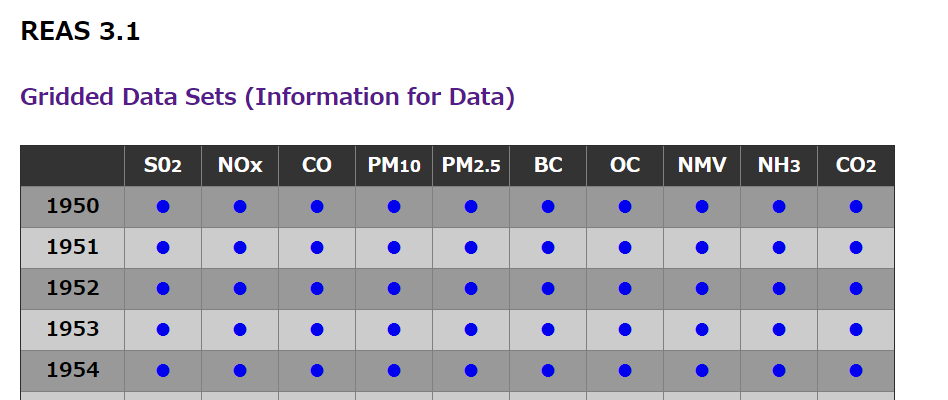


Figure 31: WEBサイトのイメージ (How the website looks like)

下にスクロールしていくと、このデータセットがあるだろう。ちなみに、information for dataというのがあるだろう。ここもクリックして読んでおこう。どういうデータなのかがわかる。英語なので、頑張って読めるようにしよう・・・。今回の目的のデータがテーブルで表示されていると思うが、列が要素、行が年数を表している。今回扱うデータはNitrogen Oxides（窒素化合物）のデータである。これらは主に工場から排出される排気ガスで、経済活動を目算するのに使われる。しかし、NOxは火山など自然に作られるものも空気中に存在する。では、どれくらいが排気ガスから排出されて、どれくらいが自然に発生しているものなのか確かめたくはないだろうか？そんな時に役立つのがこの排出データ(inventory)。それでは、実際にダウンロードしてみよう。２列目にNOxというデータがあるので、これに注目しながら、下にスクロールしていき、任意のデータをクリックしてダウンロードしてみよう。お使いのPCのスペックとネット環境によるが、すべてのデータをダウンロードするのには、時間がかかると思うので、10個くらいに絞ってダウンロードしよう。過去と現在を比べてみたければ、1950年から10年ごとにダウンロード。最近の期間に絞って、詳しく分析してみたければ、２００５年から２０１５年までのデータをダウンロードすると良いだろう。ダウンロードしたファイルは作業フォルダに整理しておこう。本書は解説上2005年から2015年までのデータを扱うことにする。

ダウンロードすると、NOX\_2005\_GRID.tar.gzという圧縮ファイルが保存されている。解凍しよう。一個一個解凍するのは、面倒だという人もいるだろう。一括でやってくれる無料ソフトがある。

<https://forest.watch.impress.co.jp/library/software/lhaplus/>

ここでダウンロードできるLhaplusさん。余談だが、解凍だけでは無くて、ファイルに鍵をつけたりもできる。最近個人情報がうるさいが、メールで送るときにこれで鍵をかけると便利だ。

解凍すると、NOX\_2005\_GRID.tarというフォルダがダウンロードされる。こちらをクリックするとNOXというフォルダになり、またクリックすると年数のフォルダが続く。その後に、やっと我々が欲しいREASv3.1\_NOX\_DOMESTIC\_2005\_0.25x0.25というファイル達が表れる。年数分けはしたほうが良いので、年数のフォルダを一つのNOXのフォルダに格納すると便利だろう。整理しておこう。

Scroll down to see this dataset (Figure 31). There will be information for data. Click here to read it as well as to see what kind of data it is. The data of your interest is displayed in the table.　Columns represent elements and rows represent years. The data is for Nitrogen Oxides (nitrogen compound). These are mainly exhaust gases emitted from factories, which can estimate economic activity. However, NOx naturally produced by volcanoes is also in the air. So, see how much is emitted from the factories and how much is naturally generated. This emission data (inventory) is useful in such cases. Download it. There is NOx data in the second column, so keep an eye on this, scroll down and click on any data to download. If all the data is downloaded, it will take a long time depending on your PC performance and the Internet speed, so download about only 10 data. Download every 10 years from 1950 to compare the past with the present. To focus on the recent period and analyze it in detail, download the data from 2005 to 2015. Organize the downloaded files in your working folder. The explanation of this book is the data from 2005 to 2015.

After downloading, a compressed file is saved under the name NOX\_2005\_GRID.tar.gz. Unzip it. Some people may find it troublesome to decompress one by one. There is free software that does it all at once.

<https://forest.watch.impress.co.jp/library/software/lhaplus/>

Download Lhaplus from the link above. As an aside, Lhaplus can decompress not only files but also lock files. Since managing personal information is strict these days, it is convenient to lock the files with this function when sending by e-mail. After unzipping the file, a folder named NOX\_2005\_GRID.tar will be downloaded. Click here to go to the NOX folder, and click to go to the years folder. After that, the files wanted will be here, REASv3.1\_NOX\_DOMESTIC\_2005\_0.25x0.25. It's better to divide by years, so it's convenient to store the years' folder in one NOX folder. Organize it.

### データの読み込み (Reading data)

次は、データの読み込みだ。一般に、Linuxでは、ユーザーのディレクトリ（windowでいうフォルダ）が作成される。ここでは、このユーザーディレクトリを"/data02/username"として記述する。

Next is data reading. Generally, Linux creates a user's directory (folder in window). Here, the user directory is "/ data02 / username".

# 最初の各設定　(First each setting)-----------------------------

# 扱うデータの取得先dir (Location to get data)

datadir=r'/data02/username/NO2analysis/inventory/NOX'

# 完成した画像の保存先directory (Location to save a created image)

save\_dir=r'/data02/username/NO2analysis/pictures/'

# 扱うデータの期間 (始まりの年,月,終わりの年,月) (Data duration (starting year, month, ending year, month))

Term=[2005,1,2015,12]

# 設定終了---------------------------------

このように先に設定しておきたいことを区切って設定するとわかりやすくなるので、便利だ。datadirは先ほどダウンロードしたファイルたちをいれておいたフォルダを指定する。Save\_dirは完成したファイルの保存先だ。完成したグラフを画像ファイルとして保存するので、そのつもりでフォルダを作っておこう。フォルダの保存先のdirectoryの取り方は、対象のファイルを右クリックして、プロパティを表示させよう。その中に場所という項目があり、似たような文字列があると思う。それがディレクトリだ。コピーして貼り付けよう。

As shown above, it is easy to understand and convenient to separate the settings first. Datadir specifies the folder for the files that were downloaded earlier. Save\_dir is the destination for the created file. Create a folder to save the completed graph as an image file. To get the directory where the data is saved, right-click the target file and open the properties. If “location” is clicked in the properties, a similar string will be seen. That is the directory. Copy and paste.

# import libraries

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from tqdm.notebook import tqdm

使うライブラリもこのように一括でimportしておく。

Import the necessary libraries.

# read data

FILENAME=datadir+r"/2005/REASv3.1\_NOX\_TOTAL\_2005\_0.25x0.25"

fin=pd.read\_table(FILENAME,header=None,skiprows=9,delim\_whitespace=True)

fin

まずは、とてもシンプルな読み込みをする。pd.read\_tableというライブラリの関数を使っている。（　）の中は引数と呼ばれる。設定みたいなものだ。FILENAMEというのは、なんのファイルを読み込むのか指定している。FILENAMEに直接、＝の後を入力しても良いのだが、長くなりそうだったので、別にした。datadirというのは最初に設定した、フォルダだ。headerというのは、

First, do a very simple read. Use a function in the library called pd.read\_table. The inside of (　) is arguments. It's like a setting. FILENAME specifies what file to read. Enter the string directly instead of FILENAME, but it will be long, so I did it above. The datadir is the first folder you set up.

|  |  |  |
| --- | --- | --- |
|  | ヘッダー1 | ヘッダー2 |
| ヘッダーA | 1 | 3 |
| ヘッダーB | 2 | 4 |

このような表のことだ。このヘッダーが含まれているのかどうかという話で、元のデータをクリックして開いてみてほしいが、ヘッダーはないので、Noneに指定する。skiprowsというのは、これも元のデータを開いてみてほしいのだが、

The header is a table like the one above. Click on the original data to open it to see if this header exists. Since there is no header, specify header = None. Open the original data again to explain the skiprows

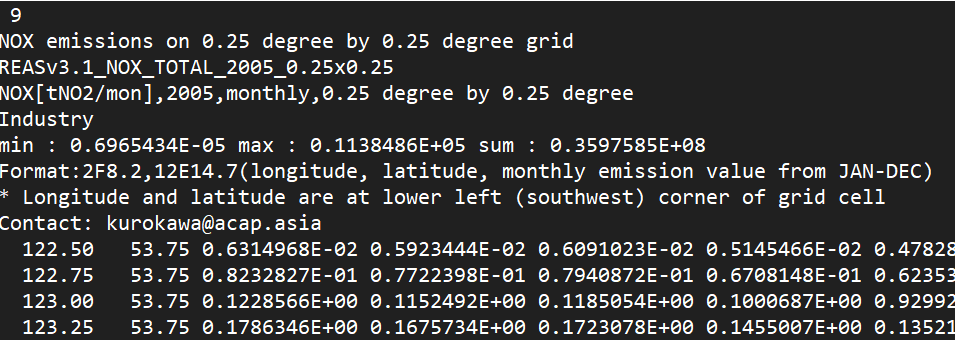


Figure 32:　テキストファイルをクリックで開く (Click to open text file)

このように、最初の9行は、データの説明になっており、欲しいのは9マス以降だ。そういう時に、skiprowsで飛ばしたい行を飛ばす。delim\_whitespace=Trueというのは、表のコマとコマの間をどう定義するかという話だ。今回のデータは間が空白になっている。例えば、/で分けたデータもある。しかし、今回は空白なので、delim\_whitespace=Trueで良い。こういう設定は、いろいろ変えてみて遊んでほしい。「python pd.read\_table」で検索してもらえれば、色々な設定が出てくるので、ぜひ。

The first 9 lines are the data description, and the 9th line and beyond are all that are needed. In this case, skiprows skips unnecessary lines. delim\_whitespace = True is the delimiter between each piece of data in the table. This time, the delimiter is blank. For example, some data is separated by “/”. However, this time, blanks separate the data, so set delim\_whitespace = True. Try changing these settings and see what happens. Search for "python pd.read\_table", there are various settings, so please try.

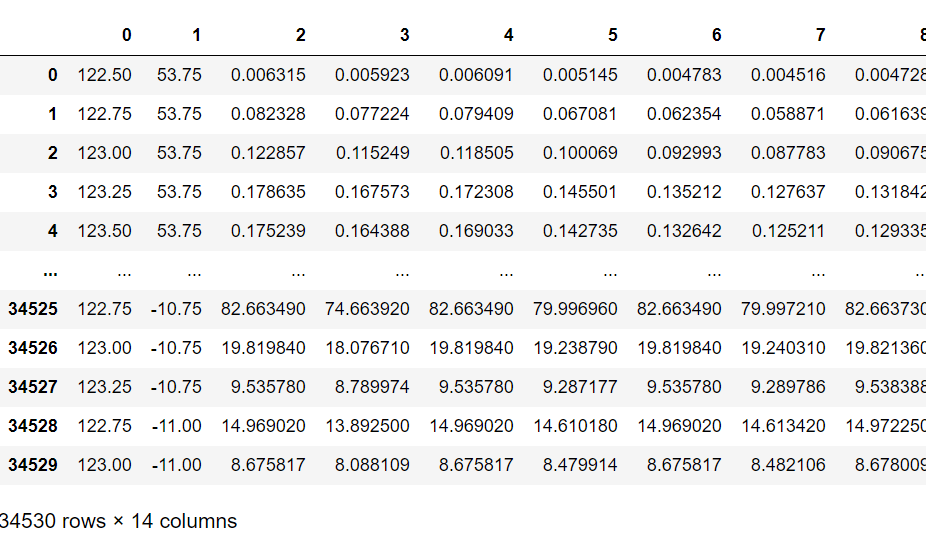


Figure 33:　テーブルデータの結果 (Table data result)

このような結果が出るだろう。ちゃんとテーブルとして読み込まれていることがわかる。ここで、データの構造を分析しよう。まずは、列から。もともと、月平均のデータだったので、12個分あるはずだが、14個ある。0と1列目とそれ以降では、数字の桁数がだいぶ違う。なるほど。0と1列目の数字を見ると、どうやら緯度経度を表しているようだ。それ以降がデータなら12か月分のデータとして辻褄があう。

余談だが、緯度経度のデータは、0.25度刻みだ。全球のデータだと仮定すると、経度は、360/0.25=1,440ピクセル。つまり、緯度経度で、1,440 x 720のデータだと予想できる。全球分のピクセル数は、1,440 x 720=1,036,800あるはずだ。しかし、行数を見ると、34,530で、全然足りない。つまり、全球データではないことがわかる。Informationを見れば、アジア圏内のデータとなっているので、納得。数字から逆算して、確かめるというのは、二重チェックという意味でとても大事なので、欠かさずにやろう。

テーブルのままでは、扱いにくいので、配列に変換しよう。そして、緯度経度のデータはいらないので、分離しておこう。なぜ、tableのままで、数値計算をしないで、配列に変換するのか説明しておく。なぜなら、numpyというライブラリが数値計算に特化していて、とても便利なのだ。しかし、これは配列でしか機能しない。なので、配列にして数値計算を行う。

Here are the results. The data was read as a table. Now analyze the data structure. First, focus on the columns. Originally, the data was monthly average, so there should only be 12 columns, but there are 14 columns. The number of digits in the 0 th and 1st columns, and the number of digits after that, are quite different. Therefore, it is easy to assume that the numbers in the first two columns are latitude and longitude, and the following data represent 12 months of NOx emission data.

As an aside, latitude and longitude data is in 0.25-degree increments. Assuming it is global data, the longitude would include 360 / 0.25 = 1,440 pixels. In other words, it would be expected to see 1,440 x 720 data in latitude and longitude. The number of pixels for the whole sphere should be 1,440 x 720 = 1,036,800. However, looking at the number of lines, 34,530 is not enough at all for 1,036,800. It turns out that this is not global data. Looking at the information, it makes sense because this data only focuses on Asia. It is always important to calcutae back from your numbers and be sure to double check your results.

Tables are inconvenient, so convert them to arrays. Since latitude and longitude data aren’t necessary, separate them here. The reason why we convert data to an array without performing numerical calculations in the table format will be explained. The numpy library is good at numerical calculations and very convenien, yet this only works with arrays. Therefore, make the data into an array and perform numerical calculations.

NO2array=fin.values

NO2array.shape

.valuesで簡単にtableから配列へと変換できる。.shapeは、配列が何ｘ何なのか教えてくれる便利な関数なので、覚えておこう。結果は、(34530, 14)と出てくるであろう。これは、配列が二次元配列であると表していて、さらに、34530 x 14のデータがあるということを表している。

".values" easily converts a table to an array. Remember that ".shape" is a useful function that tells the shape of an array (number x number). The result would be (34530, 14). This means that the array is a two-dimensional array and that there are 34530 x 14 data.

NO2data=NO2array[:,2:13+1]

NO2data.shape

緯度経度のデータはいらないので、[ ]リストを使って、分離した。さきほどのshape関数のおかげで、データがどう格納しているかわかったので、[2:13+1,:]なのか、[:,2:13+1]なのか迷わなくて済む。終わりの数字は未満という意味になるので、13+1になる。結果は、(34530, 12)と出てくるであろう。緯度経度が削除できたことがわかる。

　Latitude and longitude data are not needed here, so the [ ] list separated them. The previous shape function tells how the data is stored, so it’s not necessary to wonder if it's [2: 13 + 1,:] or [:, 2: 13 + 1]. The ending number means less than, so it needs to be 13 + 1. The result will be (34530, 12). Notice that the latitude and longitude have been deleted.

### グラフの作成 (Create graphs)

読み込みまでできたので、解析に移ろう。まずは、グラフを作るための数値の準備。今私たちが持っているデータをおさらいしよう。ひと月平均の12か月分のデータを座標ごとに持っている。これが、さらに要素ごとにあるという状況だ。そして、私たちが作りたい積算棒グラフというのは、要素ごとの棒グラフを縦に積んでいくことになる。しかし、こんな膨大な量のデータを一杯積んでも、私たちの目と脳では判断できないだろう。今回は年平均が無難だろう。そして、座標もこんなに一杯は表示できない。なら、座標も平均値化させよう。これで、一本の棒グラフを縦に積んで表示できる。これで、要素ごとにどれくらいの割合で排出されているかわかる。方針が立っただろうか？前回は、TOTALだけ読み込んだが、他に６つ、全部で７つのファイルがあったと思う。この中のTOTAL以外で、グラフを作りたい。しかし、その前に、チェックをしてみよう。TOTAL以外を足すと本当にTOTALになるのか調べてみる。

Now that the data has been read, move on to analysis. First, prepare the numerical values ​​to make a graph. Review the acquired data. The data is for 12 months of the average of one month for each element such as coordinate. Furthermore, there will be more data for other element. The integrated bar graph is a stack of bar graphs for each element. But our eyes and brain will not understand such a huge amount of data. The annual average is enough for this time since the entire coordinates can’t be displayed. Now stack a single bar chart vertically and see how much NOx each element emits. Hopefully the next step will be clear. Only TOTAL was read in previous section, but there were 6 other files, 7 in total. Make a graph with 6 of them excluding TOTAL. But before that, check to see if adding anything other than TOTAL really makes it TOTAL.

meandata=np.mean(NO2data)

meandata

先ほど、配列にしておいたので、numpyの関数が使える。すると関数一つで、年平均が出せる。結果は、86.82267871927607だ。それでは、2005年の年平均を要素ごとに出してみよう。今回は繰り返しの作業にforを使っていて、複雑になっているので、頑張ってついてきてほしい。

Earlier, the data was made into an array, so the functions in the numpy library could be used here. One function calculates the annual average. The result is 86.82267871927607. Get the 2005 annual average for each element. This time, the "for loop" repeats the same work, so it gets more complicated. Try hard to follow the manual.

#データの読み込み (read data)

term=np.arange(Term[0],Term[2]+1)

variablelist=["TOTAL","ROAD\_TRANSPORT","POWER\_PLANTS\_NON-POINT","OTHER\_TRANSPORT","INDUSTRY","DOMESTIC"]

meanNO=[]

for i in tqdm(range(1)):

for s in range(len(variablelist)):

FILENAME=datadir+ '/' +str(term[i])+"/REASv3.1\_NOX\_"+variablelist[s]+"\_"+str(term[i])+"\_0.25x0.25"fin=pd.read\_table(FILENAME,header=None,skiprows=9,delim\_whitespace=True)

NO2array=fin.values

NO2data=NO2array[:,2:13+1]

meandata=np.mean(NO2data)

meanNO.append(meandata)

meanNO

termというのは、最初に設定した期間だ。それにarangeを使い、年数の配列にしておく。次に、variablelistとして、要素のリストを作る。forを回すときに使う。ここで一つだけ注意点がある。なぜかREASv3.1\_NOX\_POWER\_PLANTS\_POINT\_2005だけ語尾に\_0.25 x 0.25がない。不親切ではないか。実はこれには理由がある。Nonpointデータというのもあるが、pointとnonpointは何が違うのか。Pointのほうは、窒素ガスの出どころがわかっているデータだけを含んでいる。Nonpointのほうは、わかっていないデータも含んでいる。つまり、より多くのデータが含まれているデータだけ今回はつかえばよいのだ。つまり、わかりやすく名前を変えてくれていて、実は親切だったのだ。リストにはnonpointだけいれた。

forで回すのだが、tqdmというのを入れている。これは、ライブラリにあるが、forを大量に回す時に時間がかかる場合がある。そんな時に、見やすいバーで、どれくらいダウンロードが進んでいるかわかるようにするものだ。lenという関数は長さを取得する関数だ。variablelistは要素６個なので、６の数字が出てくる。str( )というのは、番号を文字列に変換する関数だ。dirの指定には文字列しか使えないので変換している。これで６つの数字が格納された配列が結果で見えるだろう。そして、１番上の数字がさきほど出したmeandataと同じはずだ。

"term" is the period set at the beginning. "arange" makes the format an array of years. Next, create a list of elements as a "variablelist". These are needed for "for loop". Be careful here. For some reasons, only REAS v3.1\_NOX\_POWER\_PLANTS\_POINT\_2005 does not have \_0.25 x 0.25 at the end, making it more difficult. However there is a reason for this non-point data. What is the difference between point and nonpoint data? Point data confirm the source of the nitrogen gas data at point while nonpoint data includes everything even the data of unknown origin. In other words, it’s only needed to use data that contains more data for this exercise. So, the names were clearly distinguished and actually user-friendly. Add only nonpoint data to the list.

Add a "tqdm" function to the "for loop". It's in the library. When there are a lot of "for loop", it may take some time. In such cases, "tqdm" is an easy-to-read bar that permits seeing how far the process has gone. The function "len" is a function to get the length. Since the "variablelist" has 6 elements, the return value of "len" is 6. "str ( )" is a function that converts numbers to strings. Only strings can be specified as "dir", so they need to be converted. Now an array of 6 numbers in the result will be seen. And the number at the top should be the same as the "meandata" calculated earlier.

np.sum(meanNO[1:6])

TOTAL以外を足して、TOTALになるか確かめてみたが、結果が94.20682419903399なので、だいぶTOTALより大きい。余分なものが含まれているか、重複しているのだろう。不備があるとわかったが、どうしようもないので、信ぴょう性は怪しいなと思いながら、作業を続けよう。

いよいよ、先ほど作成した年平均から積算棒グラフを作成する。

　Now to sum up everything except TOTAL and see if it is equal to TOTAL. The result is 94.20682419903399, which is much larger than TOTAL. Maybe it contains extras or is duplicated. The data includes incomplete data set, so proceed thjs exercise with caution. Finally, create an integrated bar graph from the annual average created earlier.

# 積算棒グラフの作成 (Create a bar graph)

fig=plt.figure(figsize=(12,10))

ind=np.arange(1)

width=0.35

colorlist=["k","b","r","c","y","g"]

# グラフの装飾 (Graph decoration)

plt.title("Yearly Mean Inventory Data of NOx (ton/month)", fontsize = 18)

plt.xlabel("Years", fontsize = 15)

plt.ylabel("Amount of NOx(Ton/month)", fontsize = 15)

plt.tick\_params(labelsize=10)

# グラフ描画 (Draw a graph)

plt.bar(ind,meanNO[1],width,label=variablelist[1],align="center"

,color =colorlist[1])

plt.bar(ind,meanNO[2],width,bottom = meanNO[1]

,label=variablelist[2],align="center",color =colorlist[2])

plt.bar(ind,meanNO[3],width,bottom = meanNO[1]+meanNO[2]

,label=variablelist[3],align="center",color =colorlist[3])

plt.bar(ind,meanNO[4],width,bottom = meanNO[1]+meanNO[2]+meanNO[3],label=variablelist[4],align="center",color =colorlist[4])

plt.bar(ind,meanNO[5],width

,bottom = meanNO[1]+meanNO[2]+meanNO[3]+meanNO[4]

,label=variablelist[5],align="center",color =colorlist[5])

plt.legend(loc="upper left",bbox\_to\_anchor=(1, 1),fontsize=10)

fig.savefig(save\_dir+"/example\_bar\_stacked.png",bbox\_inches="tight")

plt.show()

2.5章で話したが、まずはplt.figureで全体の画像を貼る。グラフの装飾までは、復習なので、解説はしない。plt.barだが、forで回さないで地道にコピーしながら書いた。bottomの指定で前の棒グラフの上に置いていくというイメージだ。labelで名前をlegendとして表示できるようにしてある。色分けもして見やすいようにしている。残念ながらwidthは一個しかデータがないので機能していない。legendだが、位置の指定ができる右下とかにもできるのでいじってみてほしい。bbox\_to\_anchor=(1, 1)これを消せば、legendがグラフ枠内に入るようになる。savefigで画像を任意のフォルダに保存している。bbox\_inches="tight"は画像の外側の余白を無くすもので便利である。

As mentioned in Chapter 2.5, "plt.figure" will put the whole image first. This is the second time to decorate the graph, so it will not be explained here. Copy and paste "plt.bar" instead of a "for loop" repeatedly. Place the next bar graph on top of the previous bar graph by specifying bottom. "label" displays the name as legend. It was color-coded to make it easier to see. Unfortunately, "width" doesn't work here because it has only one data. "legend" specifies the position. It can also be done in the lower right, so give it a try. Delete "bbox\_to\_anchor = (1, 1) ", the "legend" will be inside the graph frame. "savefig" saves in a folder. "bbox\_inches = "tight" is convenient because it eliminates the outer margins of the image.

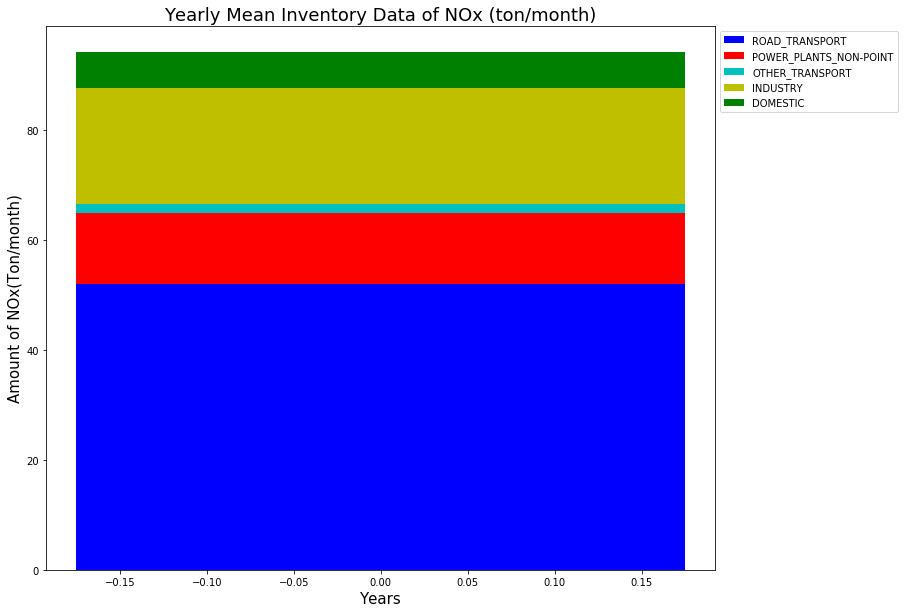


Figure 34:　一年分の積算棒グラフ (One year’s total bar graph)

結果はこんな感じである。横長いが、積算棒グラフは作れた。見た目が良くないが、これに他の年数を追加すると見栄えが良くなるので、ここは我慢しよう。工業排出(Industry)が一番割合は多いのかと思っていたが、まさか輸送業からの排出が一番多いとは。意外な結果だった。興味深い。

The result is as above. Although the width is too wide, an integrated bar graph was created. It doesn't look good as is, but adding other years to it, it will look better, so be patient. It was thought that the ratio of industrial emissions would be the highest. However, the transportation industry emits the most NOx. It turned out to be a surprising result.

### [課題]2005年から2015年までの積算棒グラフを書こう (Draw an integrated bar graph from 2005 to 2015)

それでは、ここで課題を出そう。2005年から2015年までの年平均をそれぞれ要素ごとにだし、要素ごとに格納する。配列は11年のデータx TOTALを抜いて5個の平均データを二次元の配列として格納してほしい。11 x 5の配列になる。それを積算棒グラフにして11個分の棒グラフを書こう。つまり、今までは2005年だけだったが、2015年まで全部出してほしいということだ。解答は次ページに出すが、まずは自分の力で頑張ってほしい。

Here is our new challenge. Calculate the annual average from 2005 to 2015 for each element and store them for each element. Store 5 average data excluding 11-year data x TOTAL in a two-dimensional array. There will be 11 x 5 arrays. Make it an integrated bar graph and draw 11 bar graphs. In other words, last time it was only 2005, but this time include everything from 2005 to 2015 in one graph. The answer is on the next page, but try it out without looking ahead of the manual.

# データの読み込み (Read the data)

def dataread(dir):

alldata=[]

term=np.arange(Term[0],Term[2]+1)

var=["ROAD\_TRANSPORT","POWER\_PLANTS\_NON-POINT",

"OTHER\_TRANSPORT","INDUSTRY","DOMESTIC"]

for s in tqdm(range(len(var))):

vd=[]

for i in tqdm(range(len(term))):

FILENAME=dir+str(term[i])+"/REASv3.1\_NOX\_"+

var[s]+"\_"+str(term[i])+"\_0.25x0.25"

fin=pd.read\_table(FILENAME,header=None,

skiprows=9,delim\_whitespace=True)

NO2array=fin.values

NOdata=[]

NO2data=NO2array[:,2:13+1]

md=np.mean(NO2data)

vd.append(md)

alldata.append(vd)

meandataarray=np.array(alldata)

return meandataarray

meandata2=dataread(datadir)

meandata2

解答である。まずは、データの読み込みから。今回からdefを用いた。defというのは、defの内部で定義した名前をその外側で反映しないという特性があるので、自由に短い名前を与えても、名前管理しなくてよいという利点がある。この特性を用いるとコードがだいぶ短くなるので、実感してほしい。注意としては、ここで定義したものを後で使いたい時は定義しなおさないといけないことだ。後のコードは前回説明したものと一緒なので、説明はしない。

The above is the answer. First, reading the data is focused. "def" was used this time. Since "def" has the property that the name defined inside "def" is not reflected outside. It is an advantage that there is no need to manage even any short variable names that are given. Taking advantage of this characteristic, the code will be much shorter. Note that it would be necessary for you to redefine what you define in "def" when using it later. The rest of the code is the same as explained earlier.

# 積算棒グラフの作成 (Create a total bar graph)

def bar\_stacked\_graph(d):

fig=plt.figure(figsize=(12,10))

ind = np.arange(1)

width=0.35

cl=["b","r","m","y","g"]

lb=["ROAD\_TRANSPORT","POWER\_PLANTS\_NON-POINT"

,"OTHER\_TRANSPORT","INDUSTRY","DOMESTIC"]

term=np.arange(Term[0],Term[2]+1)

# グラフの装飾 (Decorate a graph)

plt.title("Yearly Mean Inventory Data of NOx (ton/month)"

,fontsize = 18)

plt.xlabel("Years", fontsize = 15)

plt.ylabel("Amount of NOx(Ton/month)", fontsize = 15)

plt.grid(True)

plt.tick\_params(labelsize=10)

# グラフ描画 (Draw a graph)

s=np.zeros(len(d[0]))

for g in tqdm(range(len(cl))):

plt.bar(term,d[g],width,bottom = s,label=lb[g],align="center",color =cl[g])

s+=d[g]

plt.legend(loc="upper left",bbox\_to\_anchor=(1, 1),fontsize=10)

fig.savefig(save\_dir+"example\_11year\_bar\_stacked.png"

,bbox\_inches="tight")

plt.show()

bar\_stacked\_graph(meandata2)

グラフの描画である。こちらもdefを使っているおかげで、名前をかなり省略してコードがすっきりしている。そして、注目すべきは、グラフ描画のところだ。forを回すことによって、コードが少なくなった。もう一つのポイントは、bottomの部分だ。先ほどのコードと比べてみてほしいが、sの中身は最初0、2回目がd[0]、三回目がd[0]+d[1]となるようにしたい。そういう時は、最初にsを0の配列として作っておいて、そこにd[g]を足していくという方式が使える。

Above is a graph drawing code. Again, thanks to "def", the variable names are much shorter and the code is cleaner. The point of this particular code here is the graph drawing part. The "for loop" shortened the code. Another point here is "bottom". Unlike the previous code, "s " is to be 0 for the first time, "d [0] " for the second time, and "d [0] + d [1] " for the third time. In such a case, first declare "s " as an array of 0, and add "d [g] " to it every time.



Figure 35: 2005年から2015年のNOx積算棒グラフ   
(NOx integration bar graph 2005 - 2015)

結果はこんな感じである。先ほどと違ってちゃんと細い棒が並んでいる。アジア圏内のデータだが、2011年までは上昇しているが、2011年からは横ばいだ。NOx排出削減の取り組みから減ったのだろうか？Pow-plantとdomesticの排出量はほぼ変わらない。これも興味深い結果だ。

The result is as above. Unlike the previous diagram, multiple graphs are lined up properly. This is the data for Asia, which NOx level had risen until 2011 yet been flat after 2011. Is it because of the efforts to reduce NOx emissions? Amount of emissions from power plants and domestics are almost the same over time. This is also an interesting result.



# Pythonによるバイナリデータの処理 (Processing binary data with Python)

## バイナリデータとは (What is binary data)

本章では、いよいよバイナリデータを扱う。容易にデータを目で見ることが出来るテキストデータ（.txtなど）と異なり、バイナリデータ（.bin, .nc, .hdfなど数多い）を直接目で見ることは困難である(注: ワンクリックで表示してくれるソフトもあるが、今回は割愛)。不便そうなバイナリデータであるが、それでも使う理由は、データの圧縮効率にある。同じ情報量のデータを格納するとき、テキストデータに必要なデータサイズは、バイナリデータの何倍にもなる。そのため、大規模なデータを使用する必要がある分野では、バイナリデータでデータが公開されているケースが多い。陸域系であれば二次元（地図として可視化出来るもの）、気象系であれば三次元（地図＋鉛直次元）のデータの扱いを理解する必要がある。とはいえ、バイナリデータを扱う際に障壁となるのは、その読み込みである。データさえ読み込めれば、これまでに学んできた数値処理技術でデータ解析を行うことが出来る。

コンパイル言語であるC やFORTRANでも、各種バイナリデータを読み込むことは可能なのだが、ライブラリの準備などの事前処理が複雑になる(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。そこで、ここではPythonを使ったバイナリデータの処理方法を学んでいこう。

本章では、イメージを持って取り組んで頂くために、まずピクセルとは何かを説明する。次に、どういう工程で衛星データから画像を描くのかを大まかに説明する。次に、それらの画像を実際にPythonで表示するためのプログラミングについて学ぶ。

In this chapter, binary data will be learned. While text data (such as .txt) is easy to see, binary data (such as .bin, .nc, .hdf) is difficult to see directly. (Note: There is a software that displays the contents of binary data with one click, but it will not be explained here). Binary data seems to be inconvenient, but the reason for commonality is its efficiency of data compression. When storing the same amount of information, the data size required for text data is many times more than binary data size. Therefore, in the field of handling large-scale data, the data is often published as binary data. Understanding how to handle 2D data (visualized as a map) for land systems and 3D data (map + vertical dimension) for meteorological systems is necessary here. How to read binary data could be a problem when dealing with this type of data. As long as the data is read properly, the data can be analyzed using the numerical processing technology earned so far.

Compiled languages C and FORTRAN can read various binary data, but pre-processing such as library preparation is complicated (possible, but quite difficult). Again, software, such as Python, R, and MATLAB (Octave), has useful advantages in both data processing and data display. So, learn how to process binary data with Python.

First, in this chapter, this manual will explain what a pixel is, so that it can easily be imagined. Next, this manual will roughly explain the process of drawing an image with satellite data. Then, how to program the actual displayed image in Python will be explained.



## 二次元データのデータ格納方法 (How to store 2D data)

### ピクセルとは？ (What is a pixel)

まず、ピクセルとは、なんぞやという人向けに説明したい。今、見ているPCの画面がどういう構造になっているのか知っているだろうか？ Figure 36を見てほしい。花の画像だ。よく見ると、単一の色の四角形のパネルが32 x 24個あり、これが集まって画像が作られている。この四角形一つをピクセル(Pixel)という。余談だが、この一つの四角形をさらに4分割すると、62 x 48になり、細かくなるだろう。そうすると、画像はさらになめらかになるのだ。これを高解像度化という。４Kが今話題になっていると思うが、旧来のフルHDが1,920 x 1,080、4 Kは、4,000 x 2,000になるので、フルHDの１ピクセルを4分割すると、4 Kになるということだ。

先ほど、単一の色の四角形と言ったが、ここが実はポイントなのだ。一つ一つのピクセルは一つの色しか持てない。そして、その色の集まりが、Figure 36なのだ。どういうことかと言うと、数字に対する配色を変えれば、赤い花にも変えることができる。NASAやJAXAで配布しているデータは数字しか格納されていないので、それを画像に変えるのは私たちのプログラミングということになる。ここで、Pythonが必要なのだ。

First, what is a pixel? What is the structure of the screen of our PC? See Figure 36. It's an image of a flower. Look closely, 32 x 24 single-colored square panels come together to form an image. A single rectangular panel is called a pixel. Furthermore, if this one rectangle is divided into four, the finer image will result in 62 x 48. Then the image looks even smoother. This is called high resolution. 4K is a hot topic right now. The old full HD is 1,920 x 1,080, and 4K is 4,000 x 2,000, so if one pixel of full HD is divided into four, it will be 4K.

Actually, the point is the single-colored rectangle explained earlier. Each pixel can only have one color. And the set of colors is shown in Figure 36. In other words, changing the color scheme for the numbers can change it to a red flower. The data published by NASA and JAXA only stores numbers, so the task of making images by using programming can be done with Python.



Figure 36: ピクセル例　引用元　[**http://www.infonet.co.jp/ueyama/ip/glossary/pixel.html**](http://www.infonet.co.jp/ueyama/ip/glossary/pixel.html)

**(Example of pixelated image)**

### バイナリデータと配列 (Binary data and arrays)

ピクセルの話がわかってもらえたところで、これから扱うバイナリデータの説明をする。先ほどの花の画像は色というデータの集合体であるが、バイナリデータというのは数字の集まりなのである。例えば、温度。数字が温度を表していれば、それを画像にすれば、温度分布の画像が作れる。もう一つ、例を紹介する。Figure 37を見てほしい。とても荒い日本地図だ。一つ一つのピクセルに値が見えると思う。そして、数字に合わせて、色が配色されている。数字から色を割り振るのがPythonを扱う目的である。ちなみにこのデータの用途はデータ解析をしたい時に、日本だけのデータがほしいと思うことがあるが、ほとんどのデータは全球データである。日本を抽出するには、このマスクデータから番号に合うインデックスを取得し、扱いたいデータのそのインデックスの部分だけを取得すればよいのだ。今はわからないと思うが、またあとで説明するので、安心してほしい。

Now that the pixels are understood, binary data will be explained. The flower image mentioned earlier is a set of color data, but binary data is a set of numbers. For instance, temperature. If the numbers represent temperatures, a temperature distribution image can be created by changing the numbers to images. Here is another example. See Figure 37. It's a very rough map of Japan. The value in each pixel is displayed, and the colors are arranged according to the numbers. The purpose of working with Python is to assign colors according to numbers. Such method is convenient when only data for Japan is needed while most of the data is global. To get only data for Japan, the index that matches the number from this mask data is needed, and select only the data for that index from the whole data analyzed. If this is still unclear, it will be explained again later.

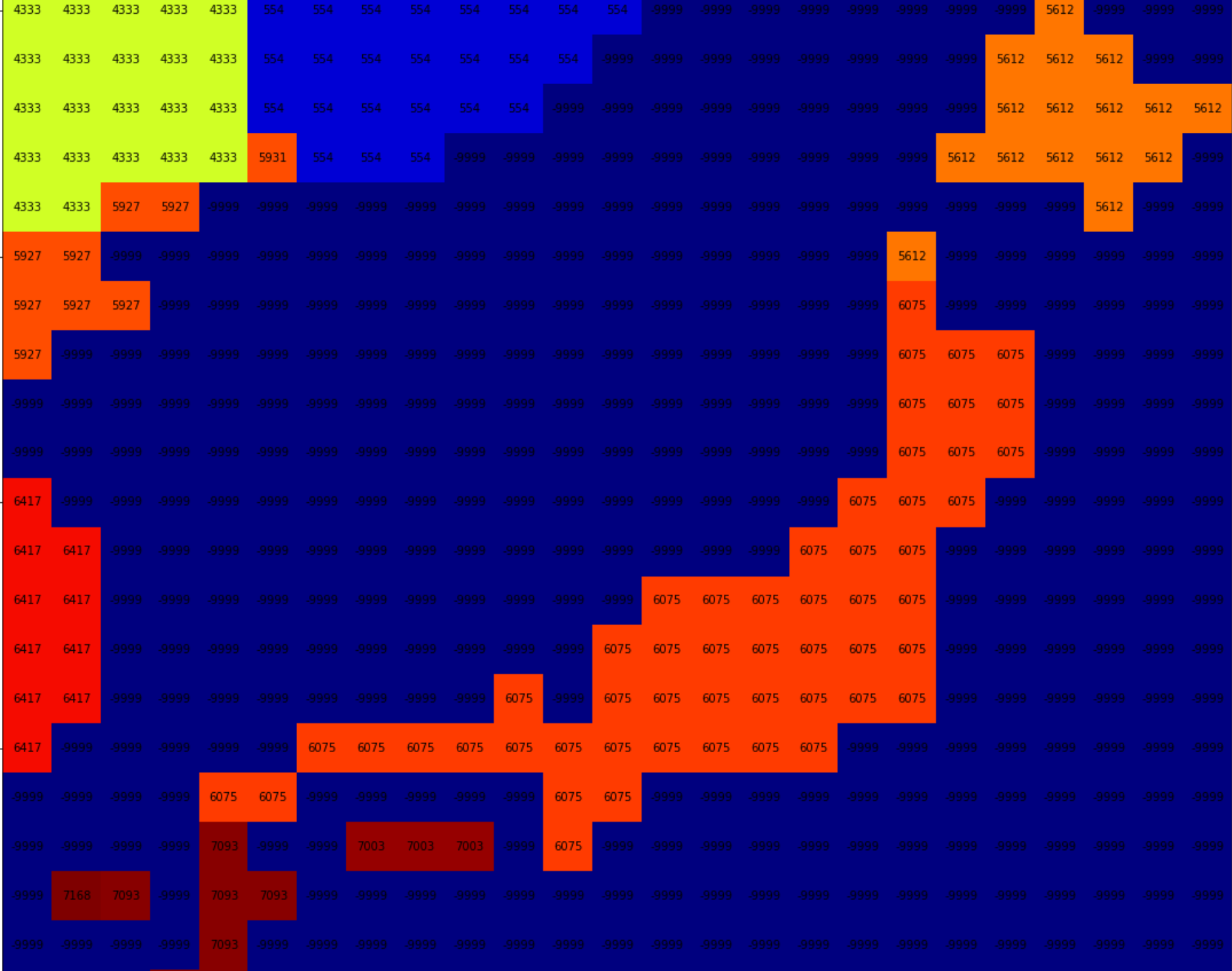


Figure 37: ピクセルと数値データの関係

(Relationship between pixels and numerical data)

バイナリデータとは、数字の集まりであるとわかってもらえたと思う。では、その数字はどのようにファイルに格納されているのだろうか。一般的に配列として格納されている。[1,2,3]こういうやつだ。

Understand that binary data is a set of numbers. So how are those numbers stored in a file? It is generally stored as an array. It looks like [1,2,3].

|  |  |  |
| --- | --- | --- |
| 20 | 21 | 20 |
| 12 | 16 | 17 |
| 10 | 15 | 17 |

Figure38: 簡易版　配列と数値データの関係

(A simple example. Relationship between array and numerical data)

ちょっとわかりにくいかもしれないが、Figure38が3 x 3の画像だと思ってほしい。それぞれ1つのピクセルに１つの数字が入っている。これを１行目と２行目、２行目と３行目の間をハサミで切ったとしようすると[20,21,20]、[12,16,17]、[10,15,17]という3数字が3つ分になるだろう。それを1行に並べたものは、[20,21,20,12,16,17,10,15,17]だ。実はこれが、バイナリファイルとして、提供されるものになっている。つまり、2列目3行目の15という数字は、バイナリファイルでは、8番目になっている。しかし、Pythonでは0から数えるので、Pythonでは、7番目だ。ちなみに、なぜ左上から数えているのか疑問に思わないだろうか？学校で習うグラフだと左下から普通数える。だが、Pythonでは左上から数える。私も理由は知らない。

It may be a little confusing, but think of Figure38 as a 3 x 3 image. Each pixel contains one number. If this is cut with scissors between the 1st and 2nd lines, then the 2nd and 3rd lines, it will be 3 sets of 3 numbers [20,21,20], [12,16,17], [10,15,17]. Then, these lines make one line [20,21,20,12,16,17,10,15,17]. This is actually a binary file format. In other words, the number 15 is in the second column and third row, so 8th in the binary file. However, Python counts from 0, so it's actually 7th in Python. It’s unknown why this is counting from the top left. If it is a graph learned at school, the usual count starts from the bottom left. But Python counts from the top left. The reason is unknown to this manual.

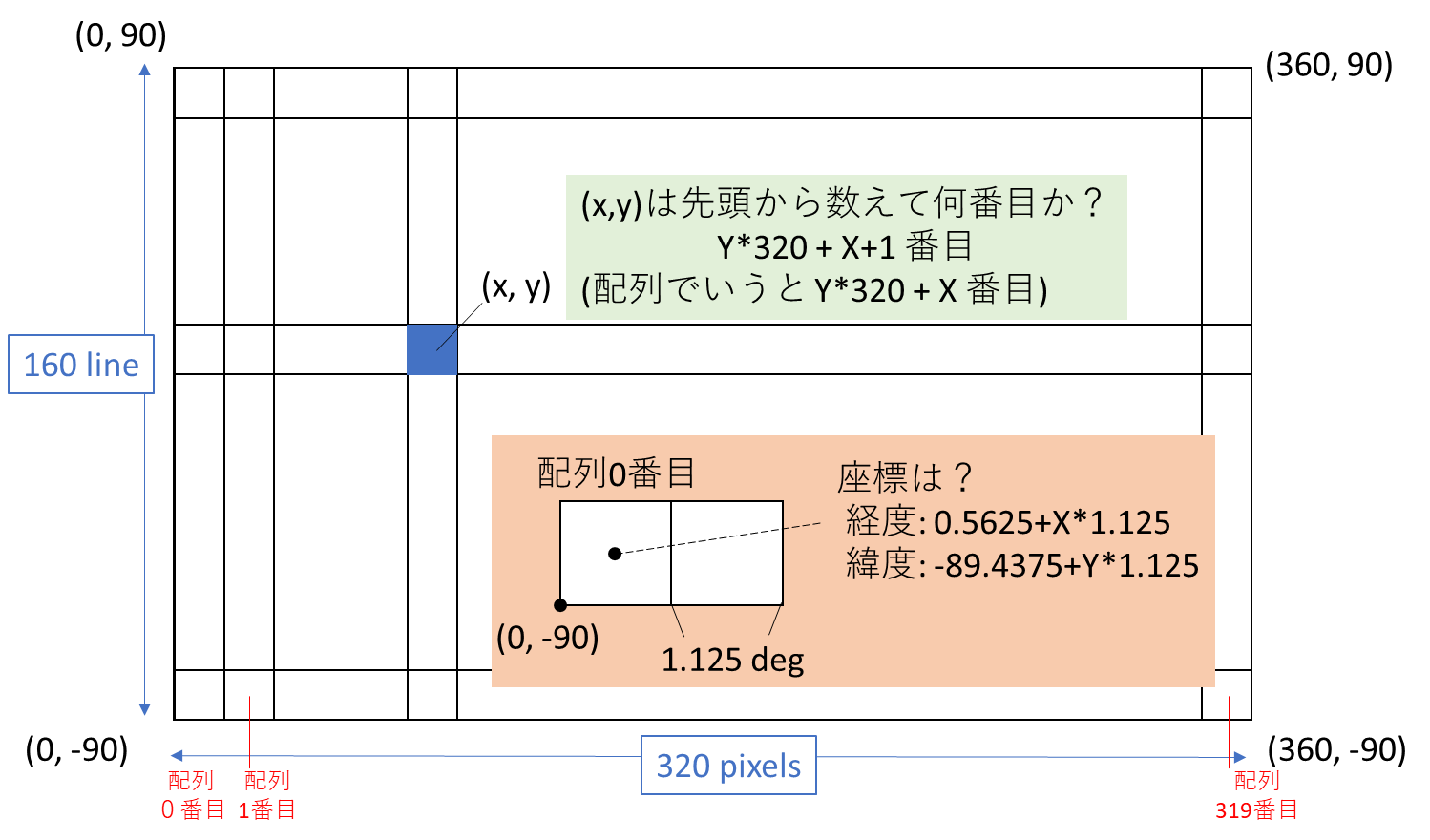


Figure39: 配列と数値データの関係

(Relationship between array and numerical data)

まずは、緯度経度と座標の関係性を説明しよう。Figure 39はよくある全球データだ。つまり緯度がー90から90度まで、経度が-180から180度までのデータ。ややこしいのが、全球データを表しているのに、ピクセル数は360 x 180ではなくて、320 x 160になっている。つまり、１ピクセルが１度ではないということだ。混同しないようにしてほしい。最初は慣れないかもしれないが、みんなもすぐに慣れると思うので、安心してほしい。つまり、１座標が360/320=1.125度ということだ。緯度経度と座標の関係は分かってもらえたと思う。次は、座標と配列の関係だ。(x,y)という座標が、一本の配列にすると何番目になるのかという計算式が載っている。Y\*320+X番目。試しに先ほどの3 x 3の場合の例を考えてみよう。2列目3行目の15という数字だったので、X=2,Y=3ということになる。おっと気を付けてほしい。Pythonは0から始まる。つまり、正しいのは、X=1,Y=2。つまり、2\*3+1=7。ほしい座標が配列のどこに格納されているか知りたい時はこの式を使おう。

First, this manual will explain the relationship between latitude, longitude, and coordinates. Figure 39 is common global data with latitudes from -90 to 90 degrees and longitudes from -180 to 180 degrees. It's confusing because the number of pixels in the data for the entire globe is 320 x 160 instead of 360 x 180. In other words, one pixel is not one degree in this case.　It may be difficult at first, but everyone will get used to it soon, so don't worry. The point here is that one coordinate is 360/320 = 1.125 degrees. The relationship between latitude, longitude, and coordinates are understood now. Next is the relationship between coordinates and arrays. The following formula calculates the index number of the coordinates (x, y) in one-line array, Y \* 320 + X. Consider the example of the 3 x 3 case above. 15 is in the 2nd column and 3rd row, which means X = 2 and Y = 3. Oops, be careful now. Since Python starts at 0, X = 1 and Y = 2 are correct. That is, 2 \* 3 + 1 = 7. Use this formula to find out where the coordinates are stored in the array.

## 衛星データを扱う工程 (Process of handling satellite data)

バイナリデータの扱いに少しは慣れたのではないか？ここでは、工程、ステップについて話したい。基本的に、衛星データはバイナリファイル、つまり数字の配列として提供されている。それを目的のサイズの画像ファイルの大きさ、例えば320 x 160などの配列に一度直し、それから画像を書く。それでは、後はプログラミングに接しながら、覚えていってほしいと思う。獅子は我が子を千尋の谷に落とす。

Is using binary data a little more familiar now? The processes and steps will be explained here. Basically, satellite data is a binary file, an array of numbers. Resize it to an array of appropriate image file size, for example 320 x 160, and then plot the data on the image. Then, practice the method while pursuing programming. Here’s a proverb, “Practice makes perfect.”



## Ctlファイル付きのバイナリデータ処理 (Binary data processing with Ctl file)

### ファイルのダウンロード

まずは、この章で使うファイルをダウンロードしてこよう。必要なファイルは、

1982010100.grd

1982010100.ctl

という二つのファイルだ。

下記のURLからダウンロードしてこよう。

<https://drive.google.com/drive/folders/1fQBEKdDs57cfO-hR6u7aq7mGit9qJT8A?usp=sharing>

First, download the needed files. They are the following two files.

1982010100.grd

1982010100.ctl

Download those from the following URL.

<https://drive.google.com/drive/folders/1fQBEKdDs57cfO-hR6u7aq7mGit9qJT8A?usp=sharing>

<https://www.dropbox.com/sh/5xvtf7vr4cbr0b8/AADNRvGdlzMRnankZdCGmsura?dl=0>

### Ctlファイルの読み方 (How to read the Ctl file)

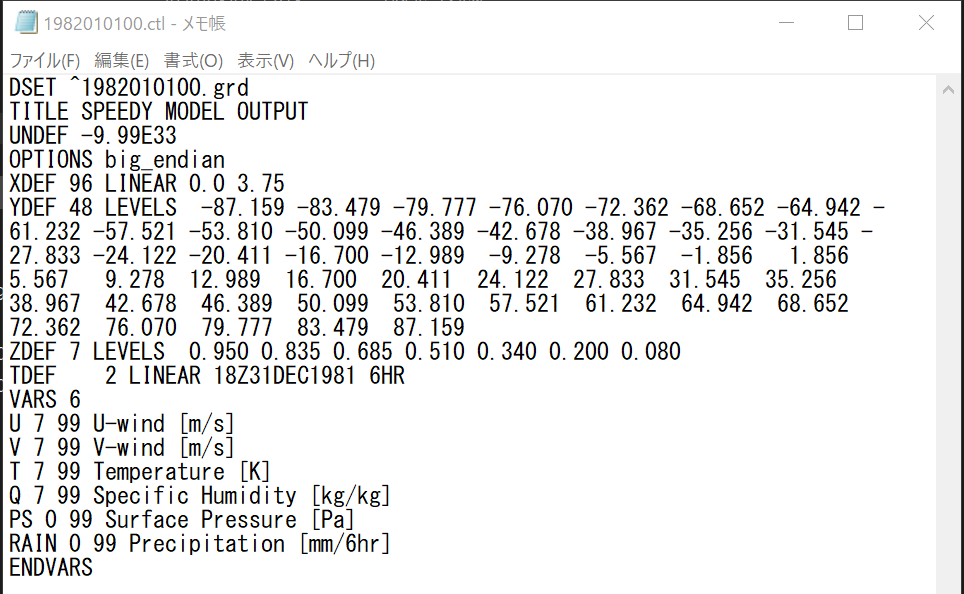


Figure 40: Ctlファイルの読み方 (How to read the Ctl file)

バイナリデータを扱っていると、Ctlデータがついてくることが多いので、まずはこの形式のデータの扱い方を学ぼう。Ctlファイルとは、データの説明が書いているファイルだ。Ctlファイルの読み方を説明していく。Ctlという拡張子のついたほうのデータをメモ帳で開いてみてほしい。Figure 40のような文字列が見えるだろう。X、Y、Z、TDEFというのは、それぞれX軸、Y軸、Z軸、そして、時間軸を表している。つまり、4次元のデータになっていることがわかる。それぞれ96、48、7、2個のデータが含まれている。XDEFの右側に0と3.75という数字があるが、これは、3.75刻みでメモリが打たれていることを示している。確かめてみよう。96 x 3.75=360。これは全球の経度に値するので合っていることがわかる。YDEFのほうは数字がならいんでいるが、こちらは48個しかデータないから全部表示させてくれている。緯度-90から90までを分割している。ZDEFは、7個あり、0.950から0.080まであるが、これは実は標高を表している。単位を消して、1から0までに分割しているので、単位は気にせず7層あるということだけ気にすればよいだろう。VARSというのは、variableのことで、このデータの場合、6個のvariableがあるので、欲しい値を指定する必要がある。それでは、実際にプログラミングに移る。適宜、コメントアウトしたり数字いじったりして、遊んでみてほしい。

Binary data often comes with Ctl data, so first learn how to handle Ctl format data. A Ctl file is a data description file. How to read the Ctl file will be explained. Open the data with the Ctl extension in Notepad. Strings are described as shown in Figure 40. X, Y, Z, and TDEF represent the X-axis, Y-axis, Z-axis, and time axis, respectively. In other words, this data is four-dimensional. Each contains 96, 48, 7, and 2 data. The number 0 and 3.75 to the right of XDEF indicate that the coordinates are in 3.75 increments. Check the equation, 96 x 3.75 = 360. This makes perfect sense because XDEF is the longitude of the entire globe. YDEF includes multiple numbers. Since there are only 48 data, all of them are displayed. It divides latitude -90 to 90. ZDEF is 7 (from 0.950 to 0.080), which represents altitude. Since the actual figures are scaled from 1 to 0, don't worry about the unit, just know that there are 7 layers. VARS means variable. For this data, there are 6 variables, so it is necessary to specify the value you want. Now, move on to the actual programming. Comment out some code or change the numbers and test what happens for fun.

### データの読み込み (Read data)

まず、モジュールやライブラリ類をインポートしておく。

First, import modules and libraries.

import numpy as np

import matplotlib.pyplot as plt

import cartopy.crs as ccrs

from tqdm.notebook import tqdm

from cartopy.mpl.ticker import LongitudeFormatter,

LatitudeFormatter

次に、データの読み込み先フォルダと完成したデータの保存先フォルダの指定。

Next, specify the folder to read the data and the folder to save the created data.

save\_dir = r'C:\Users\kotsuki-lab-W02\Documents\manual\_ep4/'

次にファイルの読み込み

Next, read the file

fin = open(r"C:\Users\kotsuki-lab-W02\

Documents\manual\_ep4/1982010100.grd","rb")

dbuf = np.fromfile(fin,dtype='float32').byteswap()

fin.close()

print(dbuf.shape)

print(dbuf)

最後のコードのdbufでdbufの中身を見ることができるので、数字が羅列されていれば、正確に読み込めていることがわかる。byteswap()というのが追加されている。これは、Ctlの中に「OPTIONS big\_endian」という書き込みがあるからだ。これは、bitの書きかたが順番通りなのか、もしくは順番が逆になっているのかというのを表している。今回はスワップが必要。詳細が気になる方は、big endianで検索してみてほしい。

次に、一直線の配列としてデータは格納されているので、画像にするために、reshape関数によって、3次元にする。そのためには、y軸、ｘ軸、ｚ軸がそれぞれいくつにしなければならないのか調べる。データの容量を見たい。右クリックでプロパティから見ても良いがせっかくなので、Pythonで確かめてみよう。

See the contents of "dbuf" with "dbuf" in the last code. If a list of numbers is displayed, it was read correctly. "Byteswap ()" was added. This is because there is a note in Ctl that says "OPTIONS big\_endian". This indicates whether the bits are written in the order or in the reverse order. This time a swap is needed. The details are available by searching for big endian.

Next, since the data is stored as a straight array, the reshape function makes the data three-dimensional in order to create an image. To do this, find out how many data must be on the y-axis, x-axis, and z-axis. Look at the total amount of data. See it in the properties by right-clicking, but this time check it with Python.

%cd C:\Users\username\Documents\python\_training

%ls

cdはdirectory changeの略で、Linuxのコマンドである。フォルダをファイルの保存先まで移動し、lsというコマンドで、ファイルのリストを見ることができる。これらは本来Linuxコマンドだが、%を頭につけることで、ノートブックでもLinuxコマンドが使える。ちなみに、スクリプト実行する場合は、アナコンダプロンプトで実行するので、%はいらない。その場合は、削除して実行しよう。

"cd" stands for "change directory" and is a Linux command. Go to the file destination and see the list of files with the command "ls". These are originally Linux commands, but Linux commands can be used in a notebook by prefixing them with %. By the way, running the script with the anaconda prompt won’t require %. In that case, delete % and run it.



Figure 41:　ファイルのデータ量 (Amount of data in the file)

"1982010100.grd"というファイルは、552,960 バイトとなっているはずだ。これから軸の数字を逆算したい。Ctlファイルから48ｘ96の画像が書けることがすでにわかっているので、割ってみよう。552960/48/96=120。残りは120ある。ここで、variableの数だが、7層あるデータが４つ、そして、１層のみのデータが２つの、合わせて30個要素がある。残りの4は何を表しているかというと１つの数字に何byte使われているかだ。実は、触れていなかったが、dbufのとこで、dtype=float32と指定している。このデータ型は１pixelのデータ量が4byteなので、辻褄が合うことが確かめられる。ここで、疑問が出てくるだろう。「あれ、4次元じゃないの？」と。逆算から、実は、TDEFは1個だったようだとわかってしまった。Ctlも残念ながら間違っていることがあるので、数字から逆算しよう。科学では、逆算はとても大事なので覚えておこう。30 x 48 x 96だとわかったので、さっそくreshapeしてみよう。

The file "1982010100.grd" should be 552,960 bytes. Calculate the numbers on the axis back. It’s already known that Ctl files can create 48x96 image, so divide them. 552960 / 48 / 96 = 120. The rest is 120. Here, regarding the number of variables, there are a total of 30 elements, 4 with 7 layers of data and 2 with only 1 layer of data. The remaining 4 (= 120 / 30) is how many bytes one number is. Actually, not mentioned before, but in dbuf, dtype = float32 was specified. This data type is 4 bytes / pixel, so it makes sense. Here, the question will come up. ‟That's four-dimensional, isn't it?”" Actually, it was found out that there is only one TDEF by back calculation. Unfortunately, Ctl may be wrong sometimes, so calculate back from the numbers. Back Calculation is always important in the science field. Now that 30 x 48 x 96 is known. Reshape it.

reshape\_data = dbuf.reshape(30,48,96)

reshape\_data

それでは、データの準備は整ったので、画像を書いていこう。

Now that the data is ready, make an image.

### 図を書く (Draw a diagram)

# draw image

fig=plt.figure(figsize=(12,10))

for i in range(0,7):

dataout = reshape\_data[i,:,:]

dataout = dataout[::-1]

ax = plt.subplot(4,2,i%7+1,projection=

ccrs.PlateCarree(central\_longitude=-180.0))

ax.coastlines(resolution='110m')

ax.set\_extent([-180,180,-90,90], ccrs.

PlateCarree(central\_longitude=-180.0))

# setting grid of latitude and longitude

ax.set\_xticks([-180,-120,-60,0, 60, 120, 180], crs=

ccrs.PlateCarree(central\_longitude=-180.0))

ax.set\_yticks([-90, -60, -30, 0, 30, 60, 90], crs=

ccrs.PlateCarree(central\_longitude=-180.0))

lon\_formatter = LongitudeFormatter(dateline\_direction\_label

=True)

lat\_formatter = LatitudeFormatter()

ax.xaxis.set\_major\_formatter(lon\_formatter,)

ax.yaxis.set\_major\_formatter(lat\_formatter)

im = plt.imshow(dataout,cmap='jet',extent=[-180,180,-90,90])

clb = plt.colorbar(extend='min')

clb.ax.set\_title('Wind \nSpeed(m/s)')

plt.title("U-WIND: ELEVATION LEVEL"+str(i%7+1))

plt.ylabel('Latitude')

plt.xlabel('Longitude')

plt.tight\_layout()

plt.savefig(save\_dir + "u-wind-image.png",bbox\_inches="tight")

plt.show()

これが、コードの全体像だ。まずは、大まかに全体像を説明してから、小分けにして説明していく。ちなみに、コロン(:)の後ろはインデント(indent)があるので、忘れないように。キーボードのtabキーを押すと、インデントできる。話を戻す。plt.figureで全体の画像を作る。subplotでその一枚大きな画像を分割しながら、一つ一つに小さい画像を貼っていくという構造になっている。その一つ一つというのが、forで回した7階層になっているデータ。dataout=reshapedata[i,:,:]では、reshapeされたデータのi番目、つまり、u-windの一層目から7層目を指定している。dataout=dataout[::-1]は、順番を逆にするコードだ。なぜか。私も書いてみてからわかったのだが、Pythonでは画像を書くときに左上から書く。しかし、このデータは左下からのデータだったのだ。だから上下を逆にする必要があった。試しにこれをコメントアウトしてみてほしい。全球の地形図と合わなくなる。surface pressureが標高の高さに影響されるので、とてもこの影響が見やすいので、見てみてほしい。

This is the whole code. First, the whole thing will be explained briefly, and explained in smaller details. By the way, don't forget to indent after the colon (:). Indenting can be done by pressing the tab key on the keyboard. Make the whole image with "plt.figure". The "subplot" divides the whole image and puts a small image on each one. Each one represents different 7 layers of data in a "for loop". "dataout = reshapedata [i,:,:] " specifies the i-th (1st to 7th layers of u-wind) of the reshaped data. "dataout = dataout [:: -1] " is the code to reverse the order. Why? As it was found out after writing it, Python creates an image from the upper left. However, this data was from the bottom left. So it had to be turned upside down. Comment this out as a test. It does not match the global topographic map. The height of the altitude affects the surface pressure, so it is very easy to see the outcome, take a look.

fig=plt.figure(figsize=(12,10))

for i in range(0,7):

dataout = reshape\_data[i,:,:]

dataout = dataout[::-1]

ax = plt.subplot(4,2,i%7+1,projection=

ccrs.PlateCarree(central\_longitude=-180.0))

ax.coastlines(resolution='110m')

ax.set\_extent([-180,180,-90,90], ccrs.

PlateCarree(central\_longitude=-180.0))

ここは、subplotで4ｘ2マスに分けて、i%7+1番目に置くように指定している。ちなみに、%は割った余りだ。0～7番目だと、あまり恩恵はないが、7～14番目だと、グラフをプロットするときにその番号にプロットされてしまう。それを割ったあまり+1にすることで、1～7の番号に変換することができるのだ。便利なのでぜひ覚えてほしい。projectionを指定している。CartoPyを使っているので、詳しくは調べてください。central\_longitudeは、基本値が0度で、イギリスが真ん中にくる地図が作成される。しかし、今回与えられたデータは太平洋が真ん中になっている。つまり、-180を真ん中に地図を設定したいので、コードのような値に設定している。coastlinesで海岸線を書いている。extentで全球を指定。

Here, the "subplot" divides the area into 4x2 squares and puts the graph at "i% 7 + 1". By the way, % is a remainder. From 0th to 7th, it does not make much difference, but from the 7th to 14th, it cannot plot the graph to the correct numbers without the function of %. By making the remainder +1, it is then possible to to convert the numbers of 1~7. It's convenient, so remember it. Projection is specified. CartoPy is used here, so check it out for more information. "central\_longitude" has a basic value of 0 degree, and a map with the United Kingdom in the center is created. However, the data given at this time has the center in the middle of the Pacific Ocean. In other words, to set -180 in the middle of a map is wanted here, so the value is set as the code indicates. "coastlines" drew the coastlines. "extent" specifies global.

# setting grid of latitude and longitude

ax.set\_xticks([-180,-120,-60,0, 60, 120, 180], crs=

ccrs.PlateCarree(central\_longitude=-180.0))

ax.set\_yticks([-90, -60, -30, 0, 30, 60, 90], crs=

ccrs.PlateCarree(central\_longitude=-180.0))

lon\_formatter = LongitudeFormatter(dateline\_direction\_label

=True)

lat\_formatter = LatitudeFormatter()

ax.xaxis.set\_major\_formatter(lon\_formatter,)

ax.yaxis.set\_major\_formatter(lat\_formatter)

im = plt.imshow(dataout,cmap='jet',extent=[-180,180,-90,90])

clb = plt.colorbar(extend='min')

clb.ax.set\_title('Wind \nSpeed(m/s)')

plt.title("U-WIND: ELEVATION LEVEL"+str(i%7+1))

plt.ylabel('Latitude')

plt.xlabel('Longitude')

ここは、軸の表記を調整しています。メモリ表記する間隔を自分で指定して、ただの数字を“S”, “N”, “E”, “W”表記に変換。

Here, adjusts the notation of the axis. Specify the coordinate spacing and convert it to “S”, “N”, “E”, “W” notation.

plt.tight\_layout()

plt.savefig(save\_dir + "u-wind-image.png",bbox\_inches="tight")

plt.show()

画像の描画からカラーバーの追加、カラーバーのタイトル指定、subplotごとのタイトル指定、ｙ、ｘ軸のラベル追加をやっている。試しに、コメントアウトしてみてほしいが、plt.tight\_layout()をいれることで、グラフごとの間隔が適切にとれるようになっている。そのあと、画像をセーブして完成。セーブする時のbbox\_inches="tight"というコードも重要で、これがあると余白が消すことができるので、後々論文に掲載したりするときに便利である。それでは、お楽しみの結果を見てみよう。

Image drawing, color bar addition, color bar title specification, title specification for each subplot, y and x-axis labels were added. "plt.tight\_layout ()" keeps the space between the graphs nice. Comment it out as a test. After that, save the image and finish. The code "bbox\_inches = "tight"" is also important, which can remove the margins when saving the image. This is convenient when wanting to publish it in a paper later. See the fun results.

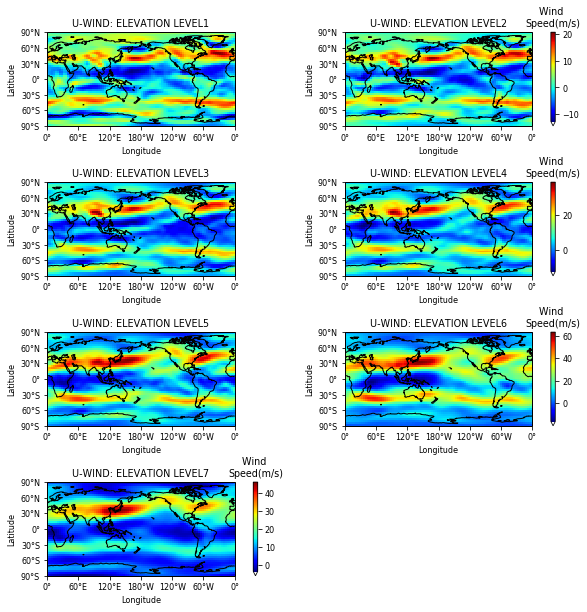


Figure 42:　SPEEDY全球データ画像　7層 (SPEEDY global data image 7 layers)

2が結果だ。U-windというのは東西の風の強さを表している。高度が高くなるにつれて、日本辺りの風が強くなっていることが見える。これは、みんなも知っていると思うが、偏西風を表している。偏西風というのは高い高度で吹いている風なので、高度が高くなるほど、強くなるのは、納得がいくだろう。

2 is the result. U-wind represents the strength of the east and west winds. The higher the altitude, the stronger the wind around Japan. This, as we all know, represents a westerly wind. The westerlies are blowing at high altitudes, so it makes sense that the higher the altitude, the stronger the wind.

### [課題]自分で図を書いてみよう（Task：Try to draw an image）

先ほど30個の要素があると言ったが、まだ7個しか画像にしていない。残りは、課題とする。ポイントになるのは、forの回し方で、他はタイトルなどが変わるだけなので、挑戦してほしいと思う。

As mentioned earlier, there are 30 elements yet there are only 7 images created so far. The rest will be homework. The point here is "for loop". Other than that, only change needed for this task is the label, title, etc. so give it a try.

### 緯度平均のグラフを書く (Draw a graph of latitude average)

次に、要素(variables)ごとに緯度平均を計算して、線グラフを作ってみよう。最初に大まかな方針を示す。ｘ軸にそって横一列が同緯度なので、それを一つの配列として切り出し、平均値化する。それをYとする。緯度は、－90から90まで作れるので緯度をXとして、グラフを書く。

Next, calculate the latitude average for each variable and make a line graph. Numerical calculations are coming out and the loops are diverse, which makes it complicated. First, a brief whole plan. The idea is that the horizontal rows of the x-axis have the same latitude, so average them for each latitude. This is the Y axis. Since the latitude is from -90 to 90, draw a graph with the latitude as the X axis.

x = np.arange(-90,90,3.75)

print(x.shape)

print(x)

y = np.copy(reshape\_data)

y = np.mean(y[:,:,:],axis=2)

print(y.shape)

print(y)

まず、np.arangeで等間隔の配列を作る。-90から90まで、3.75毎に区切って、配列を作るという意味だ。緯度を48個に分割すると、一つが3.75度である。np.copyで配列をコピーしたのは、元のデータの上書きを未然に防ぐため。次に、np.meanで同緯度に対して平均化した。y[:,:,:]と記述しているのは、ｙは三次元の配列であると、わかりやすくするため。同緯度は３番目の配列。つまり、axis=2である。

First, create an evenly spaced array with np.arange meaning to make an array by separating every 3.75 from -90 to 90. Dividing the latitude into 48, one is 3.75 degrees. The reason for copying the array with np.copy is to prevent overwriting of the original data. Next, it was averaged for the same latitude with np.mean. The reason for describing y [:,:,:] is to make it easier to understand that y is a three-dimensional array. The same latitude is the third array. That is, axis = 2.

# list

legendlist=['1st Elevation','2nd Elevation','3rd Elevation',

'4th Elevation','5th Elevation','6th Elevation',

'7th Elevation']

g\_list = [0, 7, 14, 21, 28, 29, 30]

t\_list = ["U-WIND", "V-WIND", "TEMPERATURE",

"SPECIFIC HUMIDITY ",

"SURFACE PRESSURE", "MEAN PRECIPITATION"]

ylabel\_list = ["Wind Speed(m/s)", "Wind Speed(m/s)",

'Mean Temperature(K)', 'Specific Humidity(kg/kg)',

'Surface Pressure(Pa)','precipitation(mm/6hr)']

今回は、二重forループは変則的に合計３０回（7 × 4 + 1 \* 2）実行するので、codeを簡潔化するためにリスト管理をしている。legendやラベルなどを管理する。

This time, the double "for loop" irregularly runs 30 times in total (7 x 4 + 1 \* 2), so the list was made to simplify the code. Now manage legends and labels.

#グラフ描写

fig=plt.figure(figsize=(10,10))

plt.rcParams["font.size"] = 8

for i in range(6):

ax = plt.subplot(3, 2, i + 1)

for s in range(g\_list[i], g\_list[i+1]):

ax.plot(x,y[s])

ax.set\_xticks([-90, -60, -30, 0, 30, 60, 90])

ax.set\_xticklabels(["90°S", "60°S", "30°S", 0,"30°N"

, "60°N", "90°N"])

ax.set\_xlim(-90, 90) #x軸の範囲

ax.set\_title('LATITUDE MEAN '+ t\_list[i] +

' in ELEVATIONS')

if i%2 != 0:

plt.legend(legendlist,bbox\_to\_anchor=(1.05,1),

loc='upper left', borderaxespad=0,

fontsize=8)

ax.set\_xlabel('latitude(°)')

ax.set\_ylabel(ylabel\_list[i])

plt.tight\_layout()

plt.savefig(save\_dir+"/"+'lattitude-mean-graph.png',

bbox\_inches="tight")

plt.show()

ここからは、グラフの描写だ。まず、大枠の画像の大きさと文字の大きさを決める。次にforで回す作業に入っていく。rangeが6なのは、全部で6要素だからだ。さらにsをforで回している。こちらは、先ほどリスト（g\_list）で作ったように、階層が7つだが、最後の2要素は、階層が１つしかなくて、ほかは7層あるという数の違いがあるので、単純に%で回せないときにはこうやってリストを活用する。tickを使って、ｘ軸のラベルを作る。ちなみに、先ほど図を描いたときとcodeが変わっている。なぜなら、projectionが無くなっているからだ。地形図を書かない時はいらない。tickはラベルを変換する。ｙ軸の値は何も与えなければ、勝手にplotのmoduleが設定してくれる。xlimでｘ軸の最小最大値を決めている。これがないと、0から始まったりしてしまう。ifを使って、奇数の時だけ右側にlegendを表示するようにしている。それでは、結果を見てみよう。

The code here draws a graph. First, specify the size of the outline image and the font size. Then the 1st "for loop" starts with "i". The range is 6, because there are 6 elements in total. Then the 2nd "for loop" starts with "s". There are 7 layers, as created with a list (g\_list), but the last 2 elements have only 1 layer and the others have 7 layers, so irregular "for loop" utilize the list like the one created earlier.

"tick" ​​makes an x-axis label. By the way, the code is different from the image drawn earlier. The reason is that the projection isn’t included here to draw topographic maps. "tick" converts label. If y-axis value is not given, the plot module will set it automatically. "xlim" specifies the minimum and maximum values ​​for the x-axis. Without it, the x-axis starts at 0. "If" (conditional branch) is for displaying the legend on the right side only when "i" is an odd number. See the results.

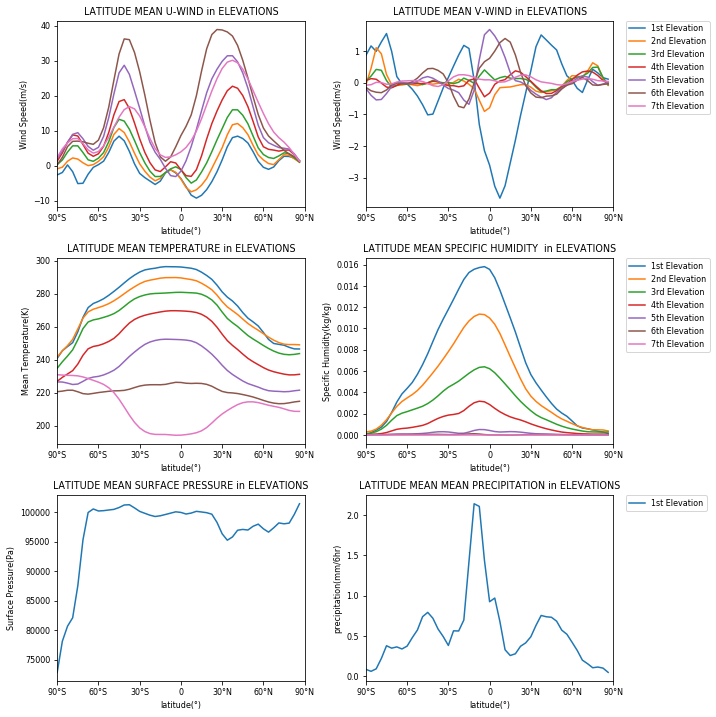


Figure43: 全球データ　６要素７層　等緯度線グラフ

(Global data 6 elements 7 layers equi-latitude line graph)

結果グラフはこの通りだ。気温（Temperature）を見ると、赤道地域が高くなるなど、現実と照らし合わせてほしい。

そういえば、気づいただろうか？今回は配列を逆転させていない。画像を書くときは左上から書いていくつまり90度から書いていく。しかし、グラフにするときは－90からプロットしていく。だから、今回は逆転させる必要はない。

解説は以上となる。数字など変えて遊んでみてほしい。リスト管理とforで回してcodeを短縮しているのが今回の肝である。これからcodeが複雑で長くなればなるほど、重要になってくる。ぜひ覚えて簡潔なcodeを書けるようになってほしい。

The graph looks like this. Temperature graph shows that the temperature is high near the equator. Please compare it with reality.

By the way, was it noticeable that this time the array was not reversed? Python creates an image from the top left, that is, from 90 degrees this time. However, Python plots on the graph from -90 degrees. Therefore, there is no need to reverse the array this time.

That is all for this explanation. Change the numbers and see what happens during your self-study. List management and the use of "for loop" to shorten the code are the keys for this lesson. From now on, the more complex and longer the code gets, the more important to have your code simple. Hopefully this knowledge will be remembered, and the ability to write concise code will be obtained.

ここまでがPythonを用いた地球科学・数値解析入門のコアパートである。衛星データのダウンロード方法など、実際の研究に関わってくるコンテンツに関心のある方は、5章に進んで頂きたい。別の発展的な内容として、下記の学習コンテンツも存在する。研究室のHPで公開しているため、関心のある読者は是非、こちらを訪問して更なる学習を進めて頂きたい。 🡺 <https://kotsuki-lab.com/internal-pages/>

* Pythonによる地球観測データ処理 (地球科学的知識の獲得)
  + (1) 全球大気再解析を用いた三次元大気データ処理 (ハドレー循環の可視化など)
  + (2) 地球の水収支 (降水量と蒸発散量の空間分布)
  + (3) 地球の放射 (地球の短波・長波放射収支)
  + (4) 衛星観測降水マップの処理 (降水量の時系列、平成30年7月豪雨の解析)
  + (5) 海表面温度SSTの解析 (固有直行分解による次元削減、基底モードの解析)
  + (6) Lorenz 63 の3変数モデルを用いたカオス力学系の理解
* Pythonによるデータ同化 (プログラムスキル、線形代数・統計数学の知識獲得)
  + カルマンフィルタ、変分法、アンサンブルカルマンフィルタなどのデータ同化手法をLorenz 96として知られる低次元力学モデルを用いて「自分の手で」実装し、データ同化やカオス力学系への理解を深めると共に、使えるデータ同化技術を習得する。

So far approaching the core part of the introduction for earth science and the numerical analysis using Python have been completed. If more interests remain for actual research contents, such as how to download satellite data, please proceed to Chapter 5. Other developmental contents and the following learning materials are available at our laboratory website:

🡺 <https://kotsuki-lab.com/internal-pages/>

* Earth observation data processing by Python (acquisition of geoscientific knowledge)
* (1) Three-dimensional atmospheric data processing using global atmospheric reanalysis (visualization of Hadley circulation, etc.)
* (2) Earth's water balance (spatial distribution of precipitation and evapotranspiration)
* (3) Earth's radiation (Earth's short-wave and long-wave radiation balance)
* (4) Processing of satellite observation precipitation map (time series of precipitation, analysis of heavy rainfall in July 2018)
* (5) Analysis of sea surface temperature SST (dimension reduction by intrinsic orthogonal decomposition, analysis of basis mode)
* (6) Understanding of Chaotic Dynamical System using Lorenz 63 three-variable model
* Data assimilation by Python (programming skills, knowledge acquisition of linear algebra/ statistical mathematics)
* Implement data assimilation methods such as Kalman filters, variational methods, and ensemble Kalman filters "with your own hands" using a low-dimensional dynamical model known as Lorenz 96, and use them to deepen your understanding of data assimilation and chaotic dynamical systems that lead to aquireing usable data assimilation techniques.

# Pythonによるデータの取得方法 (How to obtain data with Python)

## データの取得方法 (How to obtain data)

ここまでは、用意されたデータをダウンロードして、そのデータを解析する技術を身に着けてきた。では、そのデータ自体を取得するにはどうしたら良いのだろうかという疑問が出てくるだろう。本章では、それを解決したいと思う。この章を終えれば、データを取ってきて、解析するという一連の流れができるようになり、自分で研究を進めることができるだろう。楽しみにして頂きたい。ここでは、人工衛星データに焦点を絞り、データ取得の一連の流れを説明する。

衛星を扱っている主要な宇宙機関(Space Agency)は、日本のJAXA、アメリカのNASA、欧州連合のESAがある。この3機関は衛星データを無料で公開している。ありがたい話だ。「どこに何のデータがあるのか」、こればっかりは知識だ。知っているか知らないかについては、残酷だが、経験を積んで行ってほしい。私は人脈という言い方は、人との繋がりを価値あるいは物として捉えている点について、好きではない。しかし、豊富な知識のある人に教えてもらえるというのは、とても大事なことだ。だからこそ、質問上手になり、多くの知識を得ていきたい。ぜひ百戦錬磨の方に教えを受けよう。そして、あなたが教える立場になった時に、その知識を惜しみなく伝授すると良いのではないだろうか。

では、本書は君たちに何を伝えられるのか。経験の積み方だ。経験を積めと言われても、始めたばっかりの方は、何をしたらわからないだろう。入り口と進み方を教えるので、後は自分でゴールまで突き進んでほしい。それでは、調べ方の一例と、知った後、データを扱える形に形式変換、を進めていこう。

So far, how to download the prepared data and analyze it have been explained and illustrated. How to get the data yourself will be shown here. By the end of this chapter, the process of acquiring and analyzing data will be understood, so that your personal research can be done. Look forward to it. There will also be a focus on artificial satellite data, and then the flow of data acquisition will be explained.

The major space agencies that handle satellites are JAXA in Japan, NASA in the United States, and ESA in the European Union. These three institutions publish satellite data free of charge. Thanks to their work. “Where and what kind of data to find?” This type of question won’t lead you to the data of your interest. Gaining experience on searching data would be the most preferred way. The phrase “a network of connection” isn’t preferred from my personal point of view as the connection with people should not be seen as a value or thing. That’s why asking questions and gaining a lot of information from knowledgeable people is very important. Learn from experienced people. Someday this will be reversed, and your knowledge and experience can be helpful to others.

So, what can be learned from this manual? It's a way of gaining experience. Being told to gain experience on the first day is incomprehensible. How to enter and proceed will be taught here. Then, it’s necessary to proceed further to reach goals on your personal level. Now, an example of research methods and then data format conversion will be shown.

Figure 44: データ解析のステップ (Steps for data analysis)

Figure44を覚えているだろうか。前回は、このデータの形式変換を飛ばした。読み込みと解析はもう習得したので、本章では、データの取得とデータの形式変換をメインにやる。それでは、データの調べ方を学ぼう。何をやるか。まずは、最初に思い浮かぶのが、検索だろう。

[satellite database nasa]で検索してみよう。

Remember Figure 44? Last time, the data format conversion was skipped. Now that reading and analysis of data have been mastered, this chapter will focus on data acquisition and data format conversion. How to find data? The first thing is searching on the Internet.

First search for [satellite database nasa].

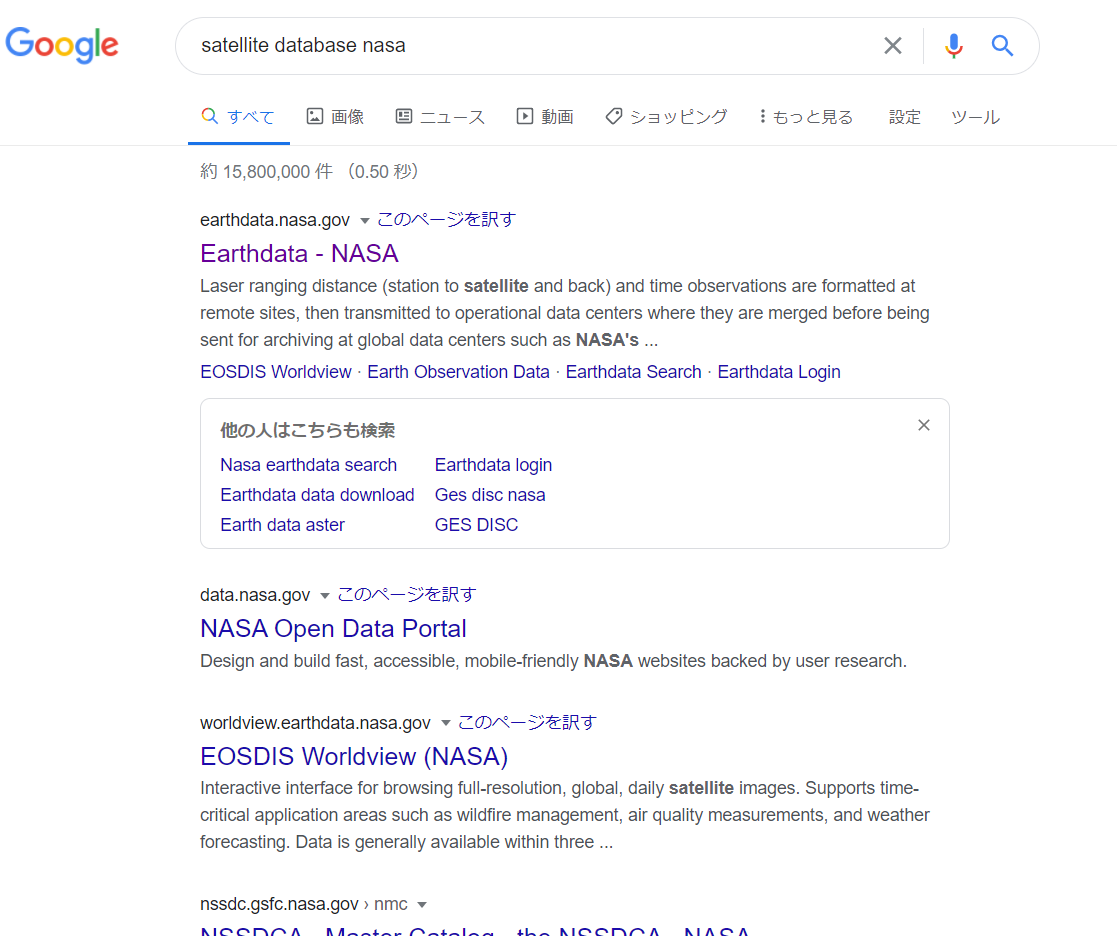


Figure 45:　NASAを検索 (Search for NASA)

おそらく、Earthdata NASAというのが、一番上に来るだろう。

I’m sure Earthdata NASA would be listed at the top.



Figure 46: Earthdata Search

次に、「DATA」の「Earthdata Search」をクリックしてもらうとデータのカタログページに行くことができる。

Click “DATA” of “Earthdata Search” to proceed to the data catalog page.

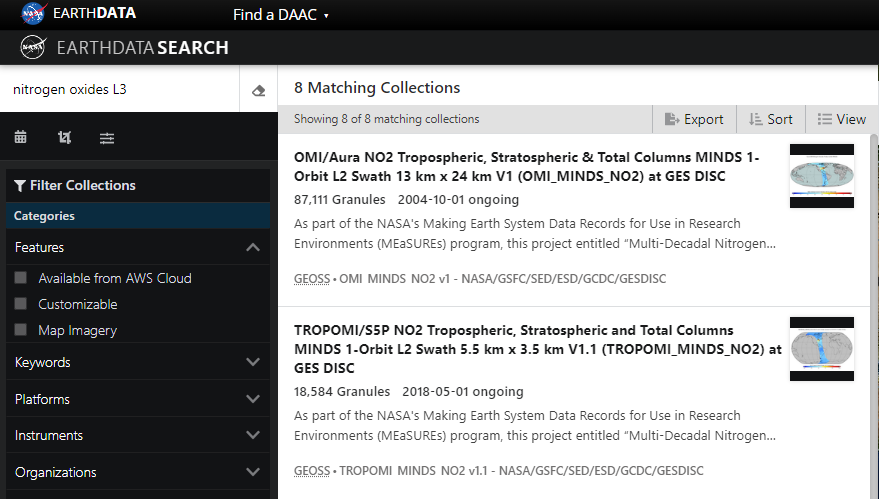


Figure 47: OMI data

今回は、3章で窒素酸化物を扱ったというのもあって、せっかくなので、このデータを見てみよう。「nitrogen oxides L3」と検索枠に入力すると、OMI/Auraのデータが出てくる。これをクリック。次にview collection detailsをクリックしてみよう。詳細が見える。次に、View All Related URLsをクリックしよう（[20220714情報更新] View All Related URLsをクリックすると、エラーページになってしまうので、Data Set Landing Pageをクリックして欲しい。Figure49のサイトが表示される）。URLを個別にではなくて、一括でダウンロードしたいからだ。ちなみに上の画面の右上に隠れているEarthdata Loginをクリックすると、Registerというのがあるので、登録しよう。無料ではあるのだが、会員登録をしないと使えないのだ。データをダウンロードする時にログインを求められるので、前もって登録しておこう。

The nitrogen oxide data was used in Chapter 3, so let’s take a look at this data. Enter “nitrogen oxides L3” in the search box and OMI / Aura data will appear. Click on this. Next, click on “view collection details” to see the details. Next, click on “View All Related URLs” ([20220714 update] If you click on “View All Related URLs,” an error page will appear, so please click on “Data Set Landing Page” to avoid the error. The site in Figure 49 will be displayed). The reason is that we want to download the URLs in a batch, not individually. By the way, the service is free, but registering as a member of this site to use their data is necessary, so register/login into “Earth data Login” at the top right of the screen. Downloading the data will require logging in, so register in advance.

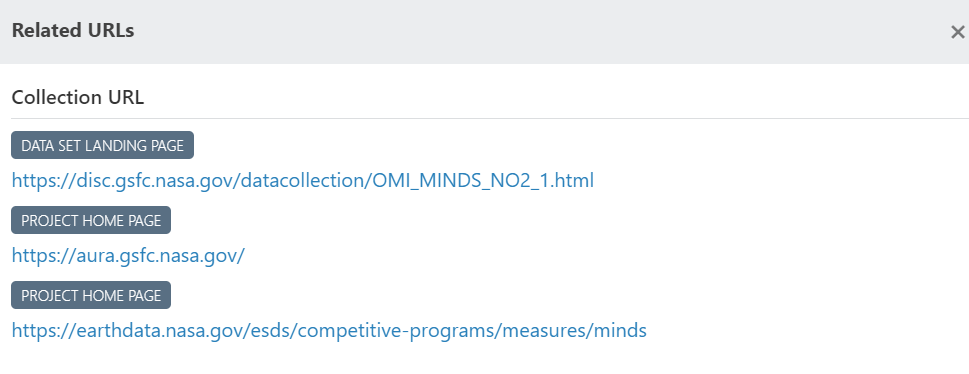


Figure 48: Collection URL

どれでも良いのだが、私は一番上のページを使っている。クリックしよう。もちろん、他のページもチェックしておいてほしい。

Any above URL is fine, but the top link is used here. Click the link. Of course, check other pages as well.

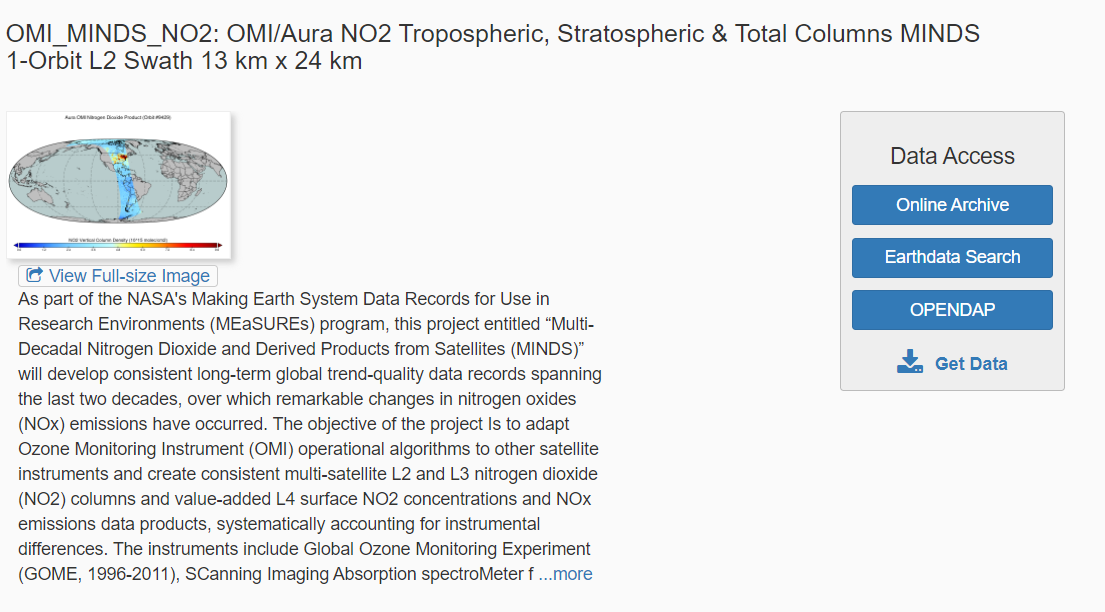


Figure 49: GES DISCのページ (GES DISC webpage)

Data Accessの枠内にあるGetDataをクリックしよう。GES DISCというサイトに到着。ちなみに見切れているが、上に検索バーがあるので、sentinel-2などなど検索してもらえれば、ほかのデータも手に入る。ぜひ、色々検索してみてほしい。

それでは、進もう。Get Dataをクリックしよう。

Click on “Get Data” under “Data Access” box. Here is the GES DISC website. There is a search bar at the top, so if searching for “sentinel-2” is used, for example, other data will be seen. Do a lot of searching. Move on. Click “Get Data”.

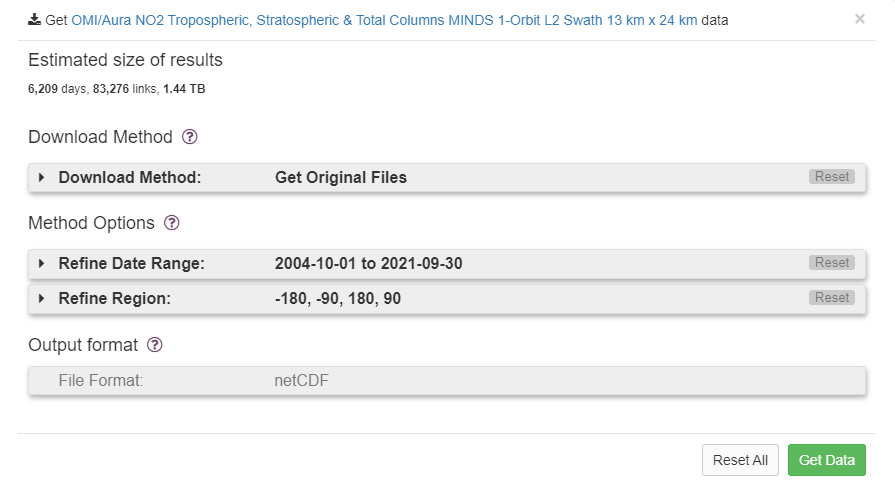


Figure 50: GES DISCのダウンロードページ (GEO DISC download page)

ここで、問題に気付いた。日付だ。日付が、2021年9月までしかない。月平均のデータなら別だが、日ごとや時間ごとの衛星データというのは、通常3日後にはアップロードされるので、3日前までくらいのデータまでは普通、手に入るのだ。つまり、このデータベースは古いデータなのではないかと疑念を抱いてほしい。この疑念から、このデータを使うのは一旦保留して、他のも検索してみようという方針になる。上の検索欄にまたnitrogenと打ってみよう。

At this point, a problem was seen. It was the date. The date is listed only until September 2021. Except for monthly average data, daily and hourly satellite data are often uploaded about 3 days after satellite observations, so the latest data is usually about 3 days ago. So be suspicious that this database is out of date. Therefore, put this data on hold and search for others. Type “nitrogen” again in the search field above.

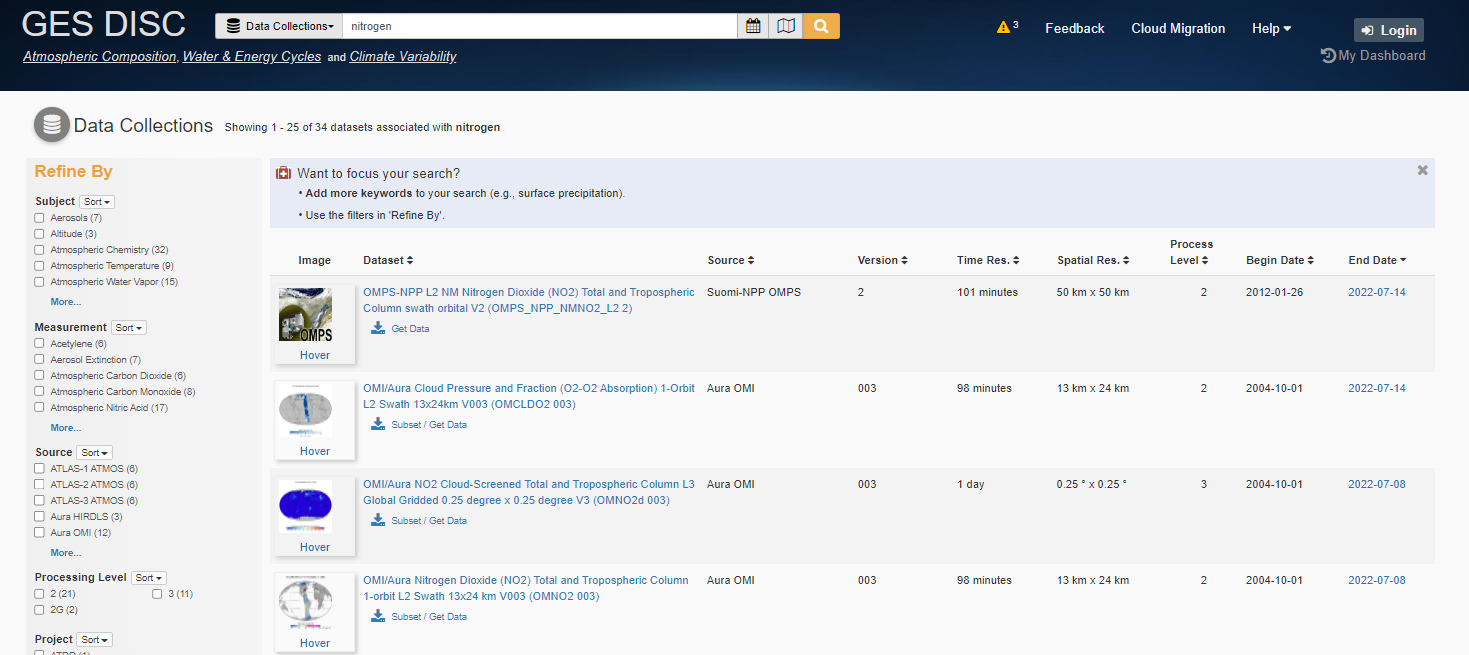


Figure 51: 再度検索 (Retrieval for the second time)

たくさん検索結果が出てきた。おそらくさきほどのデータは古いデータなのだろう。End Dateが最近のものであることを確認して安心できる。しかし、たくさんデータがあるが、どれを使ったらいいのだろうか。データの種類の見方を説明しよう。重要なのは項目が読めるかだ。Tim Resというのは、何時間ごとに衛星写真を撮っているのかということ。Spatial Resというのは、空間解像度。そして、一番大事なのが、Process Level。これは、どの程度衛星データが加工されているかというのを示している。L1だと、swathのデータになるので、

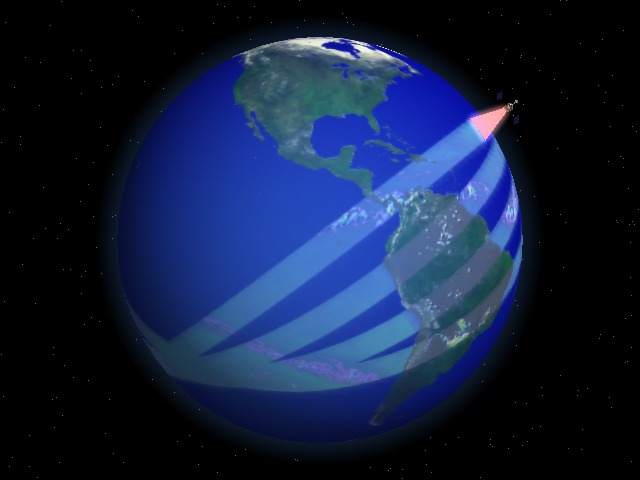


Figure 52: Swathのデータ　引用元　[**https://svs.gsfc.nasa.gov/174**](https://svs.gsfc.nasa.gov/174)

**(Reference source for swath data)**

Figure52のようなぶつ切りのデータになる。これを一枚の全球データに合成する技術が必要になってくる。極部が重複するので、大変だ。技術があるなら、どうぞお使いください。私は技術を持ち合わせていないのと、加工に時間を費やしたくないので、基本的にL2、もしくは、L3の全球データを使っている。それでは、OMI/Aura NO2 Cloud-Screened Total and Tropospheric Column L3 Global Gridded 0.25 degree x 0.25 degree V3 (OMNO2d 003)をクリックしよう。このデータは雲の影響を修正されたデータだ。衛星データは雲の影響をもろに受けるのでそれを修正されているのはありがたい。

ちなみに余談だが、少しスクロールすると、HIRDLS/Aura Level 3 Nitrogen Dioxide (NO2) 1deg Lat Zonal Fourier Coefficients V007 (H3ZFCNO2 007)というL3データもある。何が違うだろうか？End Dateが2008年までしかない。これは、同じAuraという衛星だが、古いセンサー（HIRDLS）のデータだ。もし、古いデータが欲しい場合は、こちらを選ぶ。

Figure 52 shows a lot of search results. Perhaps the previous data was the old one. Make sure the End Date is recent. However, there is a lot of data, which one is better? How to distinguish between the types of data will be explained. The important key is to read the labels. “Tim Res” is the interval between satellite photographs. “Spatial Res” is a spatial resolution. And the most important thing is “Process Level”. This shows how satellite data is processed. L1 is swath data, which is chopped data as shown in Figure 52.

Each piece needs to be synthesized, and it's a complicated process as the poles overlap. Try it if confident. Having no such skills and not wanting to spend much time processing the data, just use L2 or L3 data. Now, click on “OMI / Aura NO2 Cloud-Screened Total and Tropospheric Column L3 Global Gridded 0.25 degree x 0.25 degree V3 (OMNO2d 003)”. This is the corrected data without the influence of clouds. Clouds have a huge impact on satellite data, so revised data will be helpful. By the way, if scrolling down a little, there is also L3 data called “HIRDLS / Aura Level 3 Nitrogen Dioxide (NO2) 1deg Lat Zonal Fourier Coefficients V007 (H3ZFCNO2 007)”. What's the difference? End Date is 2008. This is the same satellite Aura, but data is from an older sensor called “HIRDLS”. If old data is wanted, select this.

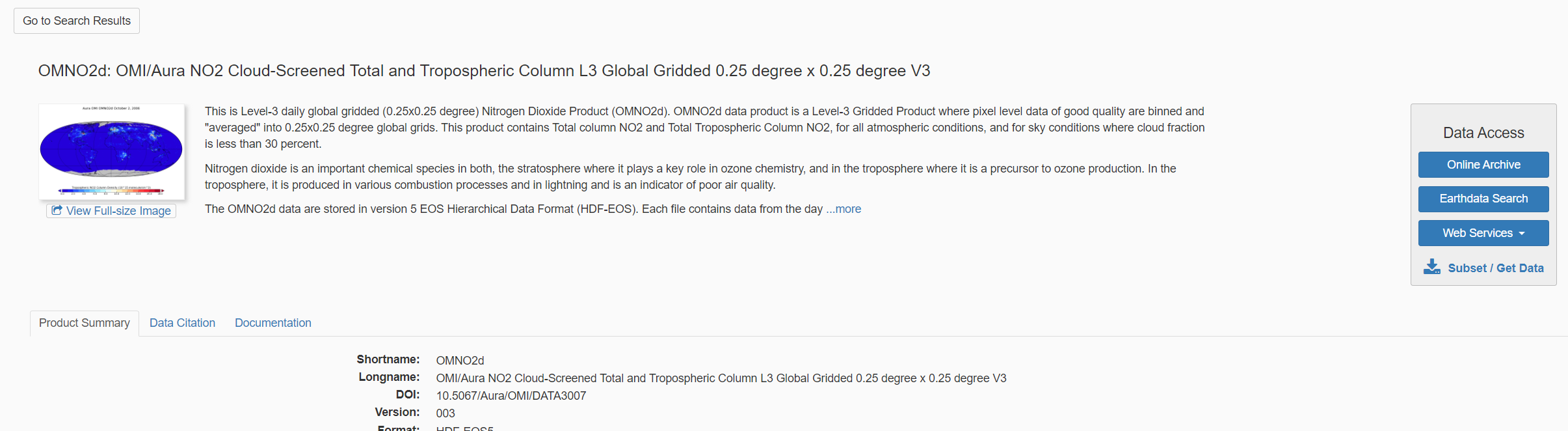


Figure 53: GES　DISC詳細説明ページ (GES DISC detailed explanation page)

詳細情報が見えるページに来た。Documentationのタブをクリックすると、詳細な説明でどんなデータなのかを書いてあるので、よく読んでほしい。Subset/Get Dataをクリックしてダウンロードページがポップアップする。

On the detailed information page, click the “Documentation” tab and read it carefully what kind of data it is. Click “Subset / Get Data” and the download page will pop up.

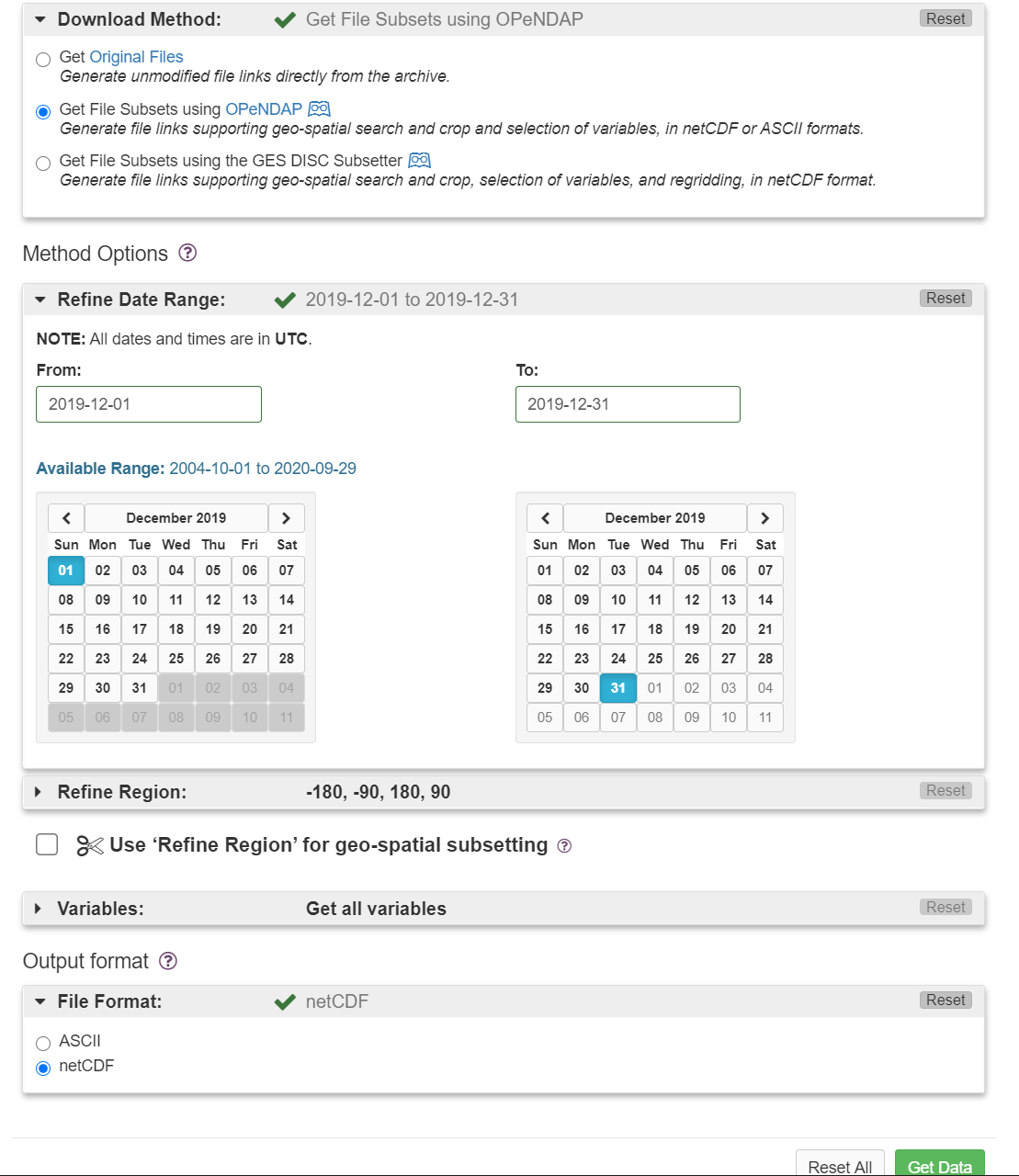


Figure 54: GES DISCダウンロードの設定ページ (GES DISC download settings page)

設定画面が出てくる。OPeNDAPを使おう。今回は、2019年の12月、1カ月分のデータに絞った。そして、File Formatだが、netCDFを選ぼう。よく使われるデータ形式で、とても便利だ。設定が終わったら下にスクロールして、Get Dataを押そう。

The setting screen appears. Check “OPeNDAP”. This time, the data for a whole month of December 2019 will be acquired. Then select “netCDF” in File Format. This is a popular data format and is very useful. When done, scroll down and press “Get Data”.

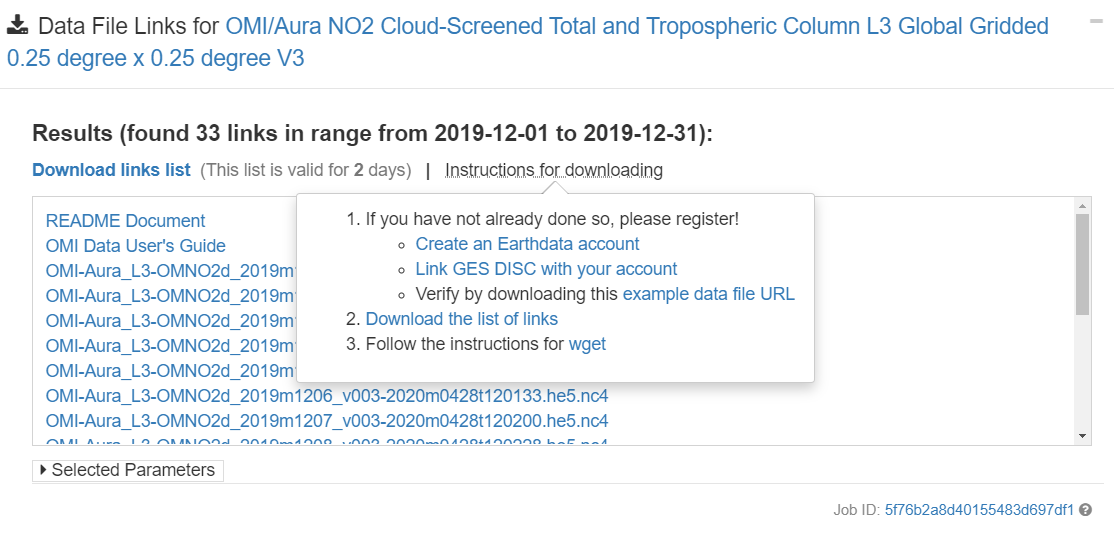


Figure 55: ダウンロード画面 (Download page)

ダウンロード画面が出てきた。同じような名前が羅列されているのが、データのURL。これをクリックすれば、パスワード認証があり、その後ダウンロードができる。

The download screen showed up. The URLs of the data are listed with similar names. If clicked on one of these, downloading can be done with password authentication.



## データのダウンロード (Data download)

先ほどのデータのダウンロードの続きをしよう。もちろん、URLをクリックしてダウンロードもできるのだが、例えば、2004年から2020年までのデータが欲しいと仮定する。5,000個ぐらいのデータがある。手動でやりたいだろうか？もちろん、やりたくない。そんなときのために、Pythonで一括ダウンロードを身に着けよう。Download Links Listをクリックする。すると、リストがダウンロードできる。作業フォルダに移しておこう。今度は、Instructions for downloadingにカーソルを合わせると、ポップアップが出てくる。Follow the instructions for wgetをクリックしよう。

Continue downloading the data. Of course, clicking the URL to download, but if data from 2004 to 2020 is requested, there are about 5,000 data available. That is too many, so learn to download at once with Python. Click “Download Links List”. Then download the list. Move it to the working folder. Next, move the cursor to “Instructions for downloading” and a pop-up will appear. Click “Follow the instructions for wget”.

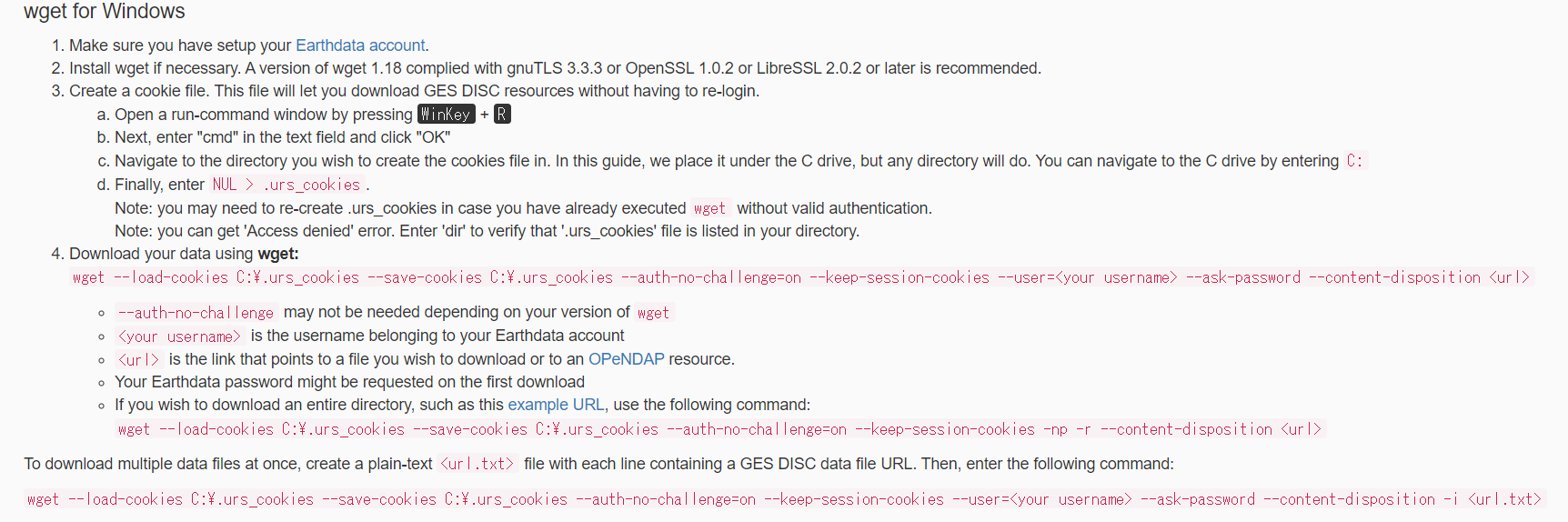


Figure 56: ダウンロード方法の説明サイト (Website explaining how to download)

NASAが自ら色々なダウンロードの方法を説明してくれているサイトに飛ぶ。ダウンロードの方法は3種類ある。そのうちの二つは私も使えたので、紹介しよう。一つ目は、Linuxコマンドを使う方法だ。本書では、アナコンダプロンプトをあまり使用していないので、この方法はあまり馴染みないかもしれない。しかし、あまり応用が効かない代わりにとても簡単なので、楽をしたいという方にはおすすめである。

Jump to a website where NASA will explain various download methods. There are three ways to download. Two of them will be introduced now. The first one is a Linux command. This method may not be familiar because this manual does not use many anaconda prompts. Although it can’t be useful to develop more complex order, it’s very easy and recommended for people who prefer fewer steps.

>cd $HOME

>touch .netrc

>echo "machine urs.earthdata.nasa.gov login <uid> password <password>" >> .netrc

>chmod 0600 .netrc

>wget --load-cookies C:\.urs\_cookies --save-cookies C:\.urs\_cookies --auth-no-challenge=on --keep-session-cookies --user=<your username> --ask-password --content-disposition -i <url.txt>

1行目はdirectoryの移動。2行目は、touchで.netrcという新規ファイルを作っている。echoでそれ以下を.netrcに記述している。ちなみに、ややこしいが、次の行の>>.netrcまでが一行だ。<uid>と<password>はそれぞれ、作成したGES DISCのユーザーIDとパス。次の行では、chmodという関数で、.netrcのファイルのアクセス権を変えている。自分しかアクセスできないようにしている。個人情報なので、見られないようにしているのだ。最後の<url.txt>の部分を消して、先ほどのDownload link listからダウンロードしたファイルのpathをいれると実行できる。そうするとファイル名がNASAの付けたファイル名で、作業しているdirectoryにダウンロードされる。シンプルで簡単なので便利だが、名前を変えたい場合などなど、Linuxに私も慣れていないので、コードの変え方がよくわからないのが、不便なところだ。

The first line "cd" is the command to move between directories. The second line "touch" creates a new file under the name of ".netrc". "Echo" writes the code in .netrc. Note that one line is "echo ~ ~ >> .netrc". <uid> and <password> are the user ID and password in GES DISC created. In the next line, "chmod" changes the permissions of the .netrc files so that only you can access them. This is personal information, so make sure no one else sees it. Run wget command after replacing the last <url,txt> part with the file from the “Download link list”. Then, the data will be downloaded to the working directory with the file name given by NASA. This method is convenient because it is simple and easy. However, when changing code, such as renaming, it can be difficult if not familiar with Linux.

それでは、二つ目の方法をやってみよう。スクロールしていくと、Python using Requestsという項目になる。ちなみに、すぐ下の項目のPython using Pydapが3つ目の方法だが、私はエラーが出て、できなかった。話を戻すが、１つ目の方法でも作成した.netrcをスクリプト内に書くような指示になっている。簡単でありがたい…。ウェブサイトにあるコードをそのまま使っても良いのだが、私は少しアレンジして使っているので、紹介する。

Now the second method is tried out. While scrolling, find “Python using Requests”. By the way, “Python using Pydap” below is the third method, but this one wasn’t successful due to the error occurred during the process. Back to the second method, write the same ".netrc" file as the first method in the script. This is easier. Use the code on the website as is, but a few changes can be made as follows.

# import libraries

import numpy as np

import pandas as pd

import glob

import requests

from tqdm.notebook import tqdm

import matplotlib as mpl

import matplotlib.pyplot as plt

import netCDF4 as nc

from cartopy.mpl.ticker import LongitudeFormatter, LatitudeFormatter

import cartopy.crs as ccrs

# ----------------------------Setting-----------------------------

# save directory of download list

list\_dir=r'C:\Users\kotsuki-lab-W02\Documents\

manual\_ep5\subset\_OMNO2d\_003\_20210719\_042905.txt'

# save directory of download data

data\_dir=r'C:\Users\kotsuki-lab-W02\Documents\manual\_ep5\data/'

# save directory of figure you plot

fig\_dir=r'C:\Users\kotsuki-lab-W02\Documents\manual\_ep5\fig/'

# name the download data

Dataname='\_NO2data'

# file extension of download data

fileextension='.he5.nc4'

# time range

# you need to write one next day

dateStart = '2019-12-01'

dateEnd = '2020-01-01'

dates = np.arange(dateStart,dateEnd,dtype='datetime64[D]')

print(dates)

# ---------------------------Setting End----------------------------

まずは、最初の設定、ダウンロードリストの名前と保存先などを指定して、いつも通りデータの期間も設定する。名前をつけるというのは、取ってきたデータ一つ一つにどんな名前をつけるかだ。今回はNO2のデータなので、こんな名前にした。そして、データの拡張子は、基本的に、元のデータのままが良い。なぜなら拡張子を変えたら読めなくなってしまったという場合があるからだ。ライブラリだが、netCDFをインストールする必要がある。アナコンダプロンプトでインストールしよう。

First, prepare the settings. Specify the name and save the destination of the downloaded list, and set the data term as usual. To name here is what kind of name to give to each downloaded data. This time it is NO2 data, Basically, the data extension should be the same as the original data. This is because changing the extension may make it unable to read. To install the "netCDF" library, download it with the Anaconda prompt.

# Read download list

with open(list\_dir,encoding='utf-8')as f:

datalist= f.readlines()

del datalist[0:2]

datalist

# create file list already downloaded

files = glob.glob(data\_dir+"\*")

file\_list1 = np.array(files)

file\_list = ["0"]\*len(file\_list1)

len\_data\_dir = int(len(data\_dir))

for i in range(len(file\_list1)):

file\_list[i] = file\_list1[i][len\_data\_dir:]

file\_list

ダウンロードリストの読み込みを行おう。先ほど、Download link listをダウンロードしたが、それを読み込む。このファイルはテキストファイルなので、読み込んで配列に変換しておく。readlineという関数で、行ごとに配列に落とし込む。最初の二行は、Readmeなどなので、私は保存する必要はないと考え、delで最初の2列を消去。保存しておきたかったら保存しよう。

Load download list. Load the "Download link list" you downloaded earlier. Since this file is a text file, read it and convert it to an array. The readline function converts each line as an array. For the first two rows, delete the first two columns with "del" because it is not necessary to save the "readme". If you want to save it, save it.

%cd /data02/username

!touch .netrc

!echo "machine urs.earthdata.nasa.gov login <username> password <pass> " > .netrc

!chmod 0600 .netrc

!cat .netrc

先ほどコマンドプロンプトで行った、ログイン情報のファイル作りだ。先ほど、手動でやってなかった場合は、このコードを使えばやってくれる。注目してほしいのは、Linuxコマンドをノートブックで使う時は、%や!の記号を頭につける。

Create a login information file that was done at the command prompt earlier. If it was not done manually earlier, this code will do it. Note that when using Linux commands in notebook, prefix them with the "%" or "!" symbol.

# Download NO2 data from GEO DISC

for idx,URL in tqdm(enumerate(datalist)):

FILENAME = str(dates[idx])+ Dataname+'.he5.nc4'

if FILENAME in file\_list:

print(FILENAME+" is already downloaded")

continue

else:

result = requests.get(URL)

try:

result.raise\_for\_status()

f = open(data\_dir+FILENAME,'wb')

f.write(result.content)

f.close()

print('contents of URL written to '+FILENAME)

except:

print('requests.get() returned an error code '+

str(result.status\_code))

ダウンロードをしよう。NASAのWEBサイトに掲載されているコードをアレンジしている。今までは、forといえば、rangeで数字を回していたが、今回は、enumerateを使うことで、idxとURLという2変数を回す。URLは、datalistの中身が順に回っている。idxには、datelistのインデックス番号、つまり、0, 1, 2, 3という数字番号が回っている。idxで、最初に作ったdatesを回し、それでファイル名を作成している。Datanameというのは、最初の設定で与えていた名前だ。今回は\_NO2。If文を使い、ファイル名がフォルダに存在していればスキップするようになっている。何回もコードを実行しても、２回目は途中からダウンロードしてくれるようになっている。残りは、NASAから取ってきたコードがそのまま使われている。PCの性能とネット回線の速さで前後するが、３０個のデータだと、２０分ぐらいかかる。気長に待とう。もし、ダウンロードデータが増えた時は、スクリプトを分割すると、同時に平行してダウンロードできるので速くなる。しかし、4つ以上に分割するのはおすすめしない。NASAからシャットアウトされたりする（経験談）。NASAにアクセス拒否を直してほしいとメールを送ると直してもらえる。

Let's download. Arranged the code posted on the NASA website. Until now, "for" was used to rotate numbers with "range", but this time, by using "enumerate", two variables, "idx" and "URL", are rotated. As for the "URL", the contents of the datalist go around in order. The "idx" has the index number of the datelist, that is, the number 0, 1, 2, 3. With "idx", I turn the dates I created first and create the file name with it. Dataname is the name given in the initial settings. This time "\_NO2". It uses an "if" statement and skips if the file name exists in the folder. Even if you execute the code many times, the second time it will be downloaded from the middle. For the rest, the code taken from NASA is used as it is. It depends on the performance of the PC and the speed of the internet line, but it takes about 20 minutes for 30 pieces of data. Please wait patiently. If the download data increases, splitting the script will make it faster because it can be downloaded in parallel at the same time. However, it is not recommended to divide it into four or more. It will shut out from NASA (real story). You can get it fixed by sending an email to NASA asking for reaccessing again.

## 画像の作成 (Creating images)

それでは、解析に移ろう。本章の趣旨は、データを取ってくるところまでなので、ここまででメインは終わっている。ここからは解析の一例としての紹介として読んでもらいたい。今までの章でやったことの繰り返しになるので、解説も少なめになる。自分で調べてみてほしい。なぜなら、調べることがとても勉強になるからだ。

Now moving on to analysis. Since the purpose of this chapter is to acquire data, the main part is over. From here, read the rest as an analysis example. Most of the code process has been explained in the previous chapters, so there is less explanation. Look it up to learn more by doing your own research.

まず、画像作成の工程だが、月で分けた後に、ピクセルごとに平均計算をして、画像を作ろう。先ほどダウンロードしたデータには何が入っているのか知りたい。しかし、4章で扱ったファイルと違って、Ctlファイルが見当たらない。netCDFファイルというのは、とても便利な形式のファイルでファイルの中にCtlファイルと似た情報が内包されている。そして、それを簡単に呼び出す魔法の関数がある。ncdumpだ。

The first process is the image creation. After dividing by month, make an image by averaging each pixel. What's in the data downloaded earlier? Unlike the files in Chapter 4, there is no Ctl file. However, a netCDF file is a very convenient file format with information similar to Ctl. And there is a magical function to get it easily. It's called "ncdump".

!ncdump /data02/username/NO2analysis/NO2data/2019-12-01\_NO2data.he5.nc4

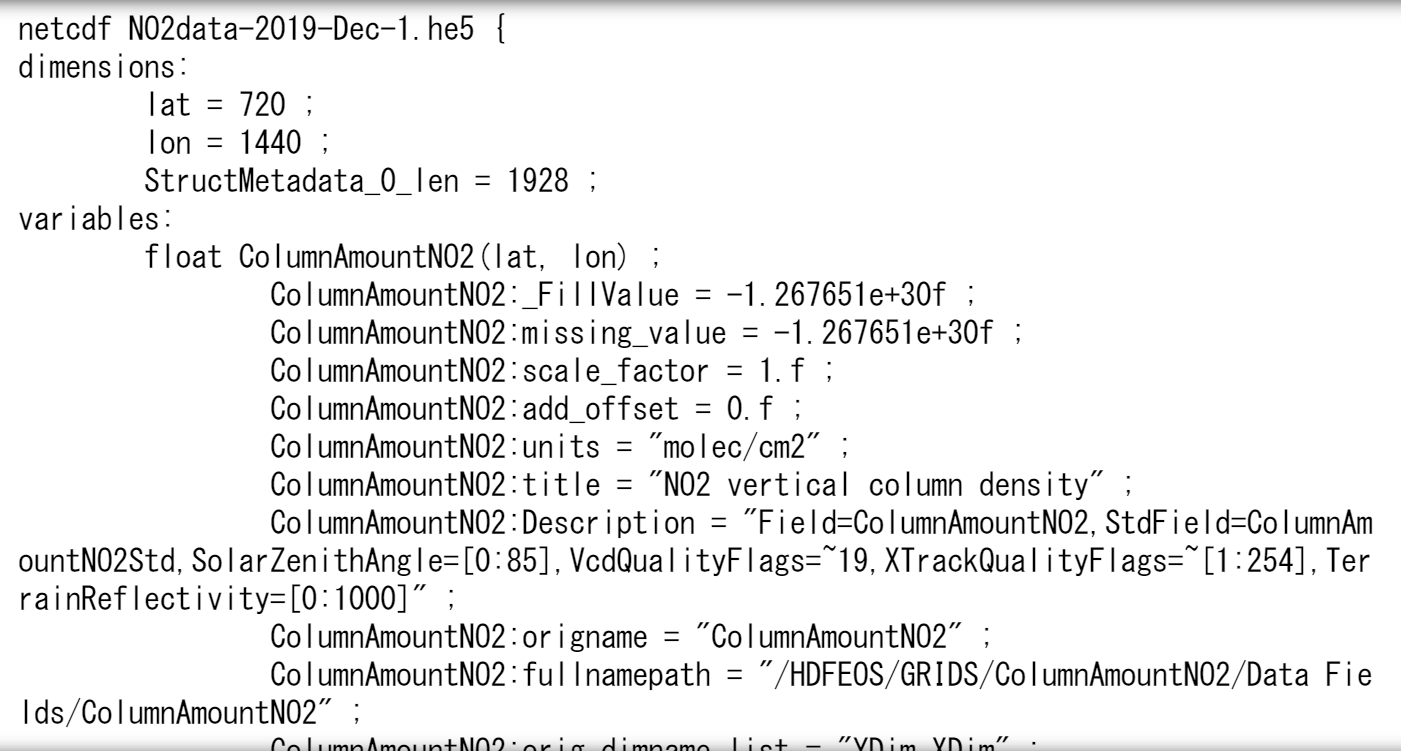


Figure 57: ncdumpの情報 (ncdump information)

少し処理に時間がかかるので、気長にまとう。結果はFigure 57。latやlonという要素に緯度経度情報が格納されていて、720 x 1440のデータだとわかる。もう一つ知っておきたいのが、どんな要素が含まれているかだ。variablesという項目があり、floatのあとの記述が、要素だ。その後の数行には、その要素の説明が入る。なので、floatに着目して追っていくと、

ColumnAmountNO2,

ColumnAmountNO2CloudScreened,

ColumnAmountNO2Trop,

ColumnAmountNO2TropCloudScreened,

Weight

Lon

lat

という７つの要素が含まれていることがわかる。７つの要素中、一番下二つは緯度経度情報なので、絶対必要だとわかる。しかし、NO2のつくデータは４つある。どれが必要なデータなのだろうか。まずは、cloudScreenedとは、雲の処理がしてあるかないかだ。Tropというのは、Troposhereの略で、対流圏という意味だ。無印のほうは、Troposhereとstratosphere(成層圏)の両方が含まれているデータになっている。経済活動の目算がしたい場合は、地表に近い対流圏のデータだけほしいので、Troposhereのデータを選ぼう。ということで、今回は、ColumnAmountNO2TropCloudScreenedが一番適している。

データ情報の中に、fillvalueやmissing\_valueというのがあるが、これは観測のないピクセルを何の値で埋めているのかという情報だ。平均値計算をするときに重要になってくるので、気に留めておこう。

It takes a little time to process, so be patient. The result is Figure 57. The elements "lat" and "lon" are latitude and longitude, which is 720 x 1440 data. Another thing to check is what elements are included. Below "variables", the string after "float" is the element. The next few lines are a description of the element. It contains seven elements. Of the seven elements, the bottom two are latitude and longitude information, so they are absolutely necessary. However, there are four data with "NO2". Which data do we need?　First of all, "cloudScreened" is whether or not the cloud effect has been processed. "Trop" is an abbreviation for "Troposhere". The top one is data that includes both Troposhere and stratosphere. When estimating economic activity, only data on the troposphere near the surface of the earth is required, so select "Troposhere" data. So, this time, "ColumnAmountNO2TropCloudScreened" is the most suitable.

In the data information, "fillvalue" and "missing\_value" are information about what value was used for unobserved pixels. Keep in mind that it will be important when calculating the mean value.

# Read data and change units

'''

change unit 1molec/cm^2 --> 1molec/cm^2\*100cm/1m\*100/1m-->10000molecules/m^2

change unit (10000molecules/m^2)/6.02e23mol^-1 -->10000/6.02e23 mol/m^2

'''

unitchange=10000/6.02e23

# create zero array of size equal to the NO2 data

all\_data = np.zeros((len(dates),720,1440),dtype=float)

for i in tqdm(range(len(dates))):

FILENAME = str(dates[i])+ Dataname+'.he5.nc4'

Data　=　nc.Dataset(data\_dir+FILENAME,'r',dtype='float32')

NO2 = data.

variables["ColumnAmountNO2TropCloudScreened"][:,:].data

lon = data.variables["lon"][:].data

lat = data.variables["lat"][:].data

vmin = 0

np.place(NO2, NO2 <= vmin, vmin)

NO2 = NO2 \* unitchange

all\_data[i] = NO2

all\_data.shape

まず、単位変換の式を書いている。WEBサイトからREADMEを読んでもらうと、元のデータの単位がmolec/cm2になっている。あまり馴染みのない単位なので、mol/m2に変換する。次にデータの読み込みのブロックがあるが、先ほどのダウンロードと同じ形で名前を作って読み込む。読み込む関数は、nc.Datasetを使う。これはnetCDF専用の読み込み関数だ。その後、データから、要素を抜き出す。データの抜けが負の大きい数字になっているので、平均する時に支障になってくる。０にしておこう。緯度経度も取得する。平均計算だ。結果は、 (31, 720, 1440)と出てくるだろう。31日分の720 x 1440のデータになっていることが分かる。

Start the analysis. First is the unit conversion formula. According to the "README" on the website, the original data unit is “molec/ cm2”. This is an unfamiliar unit, so change it to “mol / m2”. The next block is reading the data, giving the data a name and reading it in the same way as the previous download. The read function is "nc.Dataset". This is a convenient read function for netCDF. After that, the elements are extracted from the data. The missing data is a large negative number, so set it to 0 for averaging. Also get the latitude and longitude with an average calculation. The result will be (31, 720, 1440). This data includes 720 x 1440 each day for 31 days.

monthly\_mean\_data=np.mean((np.copy(all\_data)),axis=0)

monthly\_mean\_data.shape

# data check

'''

The idea here:

the longitude data goes left to right

the latitude data goes bottom to top

'''

print(lon.shape)

print(lon[710])

print(lat.shape)

print(lat[350])

# data check

'''

argmax finds the index of max value

we know the max value in the 115E would be in CHINA

513/720, which is before half -> counting from bottom

Now, we know NO2 data and longitude and latitude are matched

'''

CHA\_index = [(115+180)/360\*1440]

CHA\_index = np.array(CHA\_index,dtype=int)

print(np.argmax(monthly\_mean\_data[:,CHA\_index]))

# imshow plot left to right and top to dottom -> needs flip

monthly\_mean\_data2 = np.flipud(monthly\_mean\_data)

データの配列の順番をチェックした。緯度の中央が720の半分の360、経度の中央が1440の半分の720、それらの少し下を指定した時にマイナスの値になるかをチェックした。それによりデータの配列は南から北の順番、西から東の順番に並んでいることがわかる。次に、中国辺りに一番大きい値があることがWEBサイトからわかっているので、その経度に値を絞り、その中で、最大値のindexを出した。中国は北半球にあるので、値が中央値より下なら上から数えていることがわかる。これで、緯度経度NO2と３つのデータが同じ順番に数えられていることが確かめられた。ただし、pyplotを使う時は右上から数えるので、その時のために上下逆にしたデータを作成した。

Checked the order of the data array. Also checked if the center of latitude is 360, which is half of 720, the center of longitude is 720, which is half of 1440, and a little below those numbers is a negative value. It can be seen that the data is arranged in the order of south to north and west to east. Next, since I know from the website that there is the largest value around China, I narrowed down the value to that longitude and put out the index of the maximum value among them. Since China is in the Northern Hemisphere, if the value is below the median, we know that we are counting from the top. This confirms that latitude/ longitude NO2 and the three data are counted in the same order. However, when using pyplot, it counts from the upper right, so I created the data upside down for that matter.

画像にするには、720 x 1440の二次元の配列にする必要がある。なので、時間軸に平均する。Axisを決めてmeanすると、その軸に対して平均することができる。

To make an image, it needs to be a 720 x 1440 two-dimensional array. So, average to the time axis. By running the "mean" function with "axis" specified, only that axis is averaged.

# Plot figure

def draw\_picture(data):

fig = plt.figure(figsize=(15, 10))

plt.rcParams["font.size"] = 12

ax = plt.subplot(1, 1, 1, projection=ccrs.PlateCarree())

ax.coastlines(resolution="110m",color='grey')

ax.set\_extent([-180,180,-90,90], ccrs.PlateCarree())

ax.set\_title("NO$\_2$ 2019,Dec Monthly Mean (mol/m2)")

plt.ylabel('Latitude')

plt.xlabel('Longitude')

# setting grid of latitude and longitude

ax.set\_xticks([-180,-120,-60,0, 60, 120, 180],

crs=ccrs.PlateCarree())

ax.set\_yticks([-90, -60, -30, 0, 30, 60, 90],

crs=ccrs.PlateCarree())

lon\_formatter = LongitudeFormatter

(dateline\_direction\_label=True)

lat\_formatter = LatitudeFormatter()

ax.xaxis.set\_major\_formatter(lon\_formatter)

ax.yaxis.set\_major\_formatter(lat\_formatter)

vmin = 0

vmax = 0.0002

np.place(data, data <= vmin, vmin)

np.place(data, data >= vmax, vmax)

im = plt.imshow(data,cmap='jet',extent=[-180,180,-90,90])

clb = plt.colorbar(aspect=20,shrink=0.6)

clb.set\_label("NO$\_2$(mol/m2)")

plt.tight\_layout()

fig.savefig(fig\_dir+

"NO2\_2019\_Dec\_monthly\_mean\_image.jpg",

bbox\_inches="tight")

plt.show()

draw\_figure(monthly\_mean\_data2)

それでは、画像の描画に取り掛かる。前半部分はほぼ前章と同じなので、解説しない。#NO2からがnetCDFの新しい描画の仕方だ。Vmaxを決めることで、色の違いを強調して見やすくしている。残りのコードで気になったところは調べてみてほしい。ここまで来たみなさんなら、調べられると思う。関数名で検索したり、色々数字を変えたりして、動作を確かめてもらいたい。

Now, start drawing the image. The first half is almost the same as the previous chapter, so it will not be explained. Below "#NO2" is a new way of drawing with netCDF file. Setting "vmax" emphasizes the difference in color and makes it easier to see. If there are any questions about the rest of the code, find out for yourself. Anyone who has reached this point can find the answer themselves. Check the operation of the code by searching the Internet for the function name or changing the value of the variable in the code.

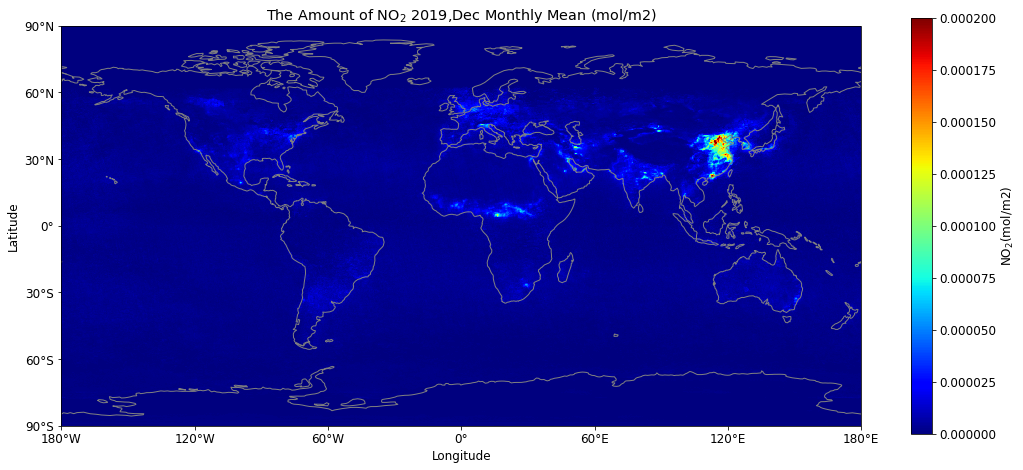


Figure 58: Oxの全球画像 (Global image of NOx)

結果の画像がFigure 58である。

The resulting image is Figure 58: 8.

余談だが、この数値に上限を決めて、画像にしたのは、以下のリンクにある画像と比べるためだ。

"vmax" was set, and an image was created to compare it with a certain image. It is at the link below.

**https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS\_S5P\_OFFL\_L3\_NO2**

上記のURLにある。Google Earth Engineを知っているだろうか？Googleが提供している簡易的な衛星解析サイトだ。Javascriptのコードを使って解析ができる。パッと画像化できるので、試しにという時には、Pythonより早いので、使ってみても良いだろう。もしかしたら、利用するのに、登録が必要だったかもしれない。Gmailを持っていれば、すぐに使えるようにできるので、やってみてほしい。

Do you know the Google Earth Engine? Google's simple satellite analysis website. Javascript code is used to analyze. Javascript creates images faster than Python so it’s worth trying. It may be necessary to register to use it. Having a Google account with Gmail will help to use immediately, so try it.

URLに行き、スクロールダウンすると、[Open in Code Editor]という青いボタンがあるので、クリックしよう。

Go to the URL, scroll down and see a blue button called [Open in Code Editor], click it.

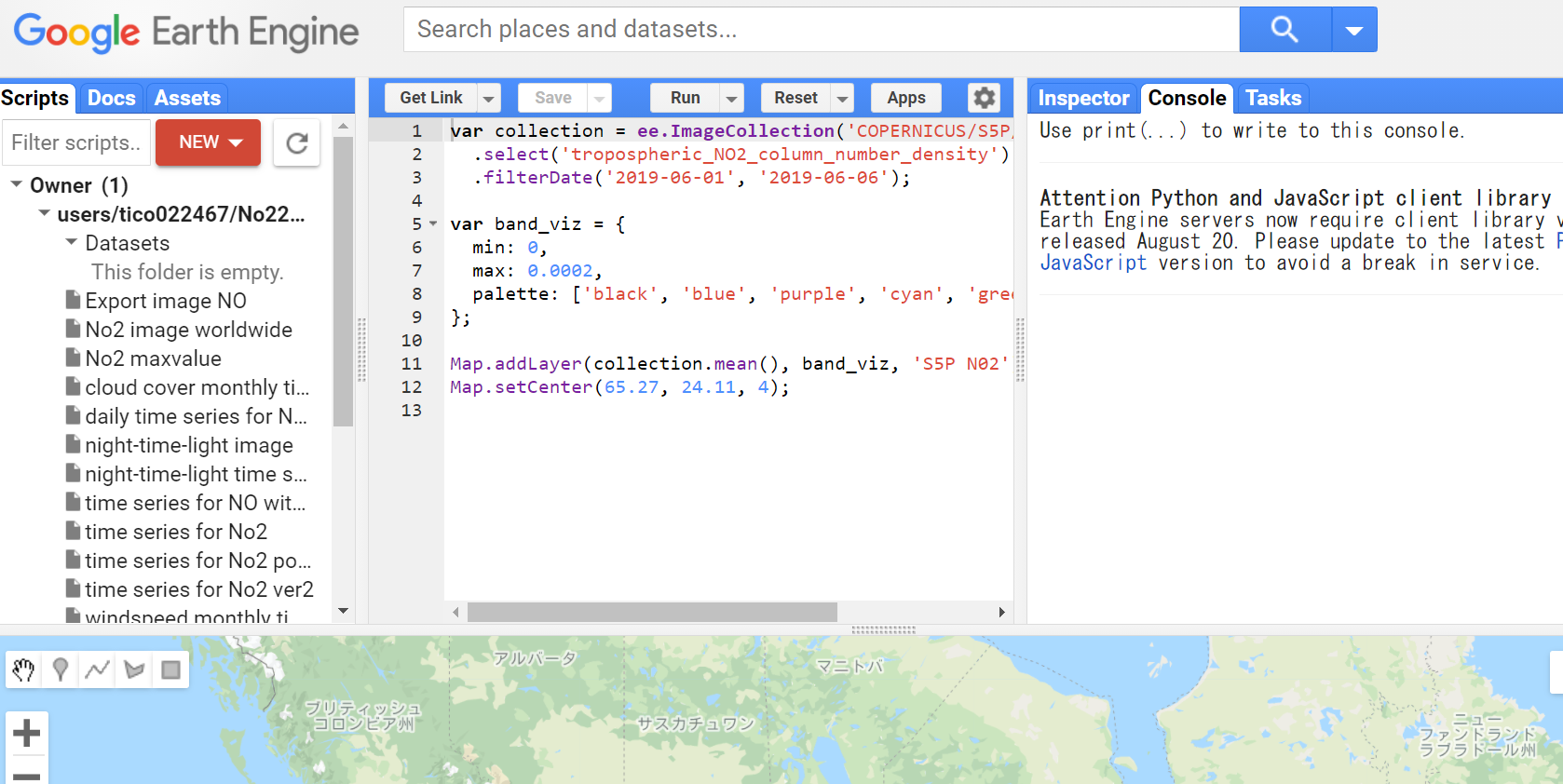


Figure 59: Google Earth Engine

Figure 59の画面になる。もう画像を見るコードが書かれていて、Runを押すだけで、画像を見ることができる。出てきた画像が先ほど作った画像の単位と数値が一緒になっているので、同じような画像になる。ぜひ、確かめてみてほしい。

Figure 59 is the link above. The code to see the image is written in advance, and the image is seen just by pressing "run". The image created by Google Earth and the image we created earlier have the same unit and value, so they look similar. Check it out.

### 5.3.1 Colormapの作成 (Creating Coloarmap)

Colormapは、グラフの配色を決めるものだ。色を変えたいこともあれば、配色の法則を変えたいことがある。例えば、データには、0から20まで値が存在するが、値はほとんど１０付近に集中している場合。１０付近の値を多く分割して図に表したいだろう。

Colormap determines the color scheme of your graph. Sometimes you want to change the color as well as the law of color scheme. For example, if the data has values from 0 to 20, but the values are mostly concentrated around 10. I would want to divide the values around 10 into many parts and show them in the figure.

s = np.unique(monthly\_mean\_data2)

vmin = 0.000025

np.place(s, s <= vmin, 0)

s = np.unique(s)

plt.hist(s)

試しに、上で扱ったデータの棒線グラフを書いてみよう。np.uniqueは重複する値を取り除いてくれる。さらに、２次元のデータだったのを１次元に変換してくれるので、棒グラフを書く場合は、二役できる。0.000025以下の値が多かったので、取り除いた。

As an experiment, let's draw a bar graph of the data dealt with above. "np.unique" removes duplicate values. Furthermore, since it converts two-dimensional data into one-dimensional data, it can be used for two purposes when drawing a bar graph. Many of the values were 0.000025 or less, so they were removed.

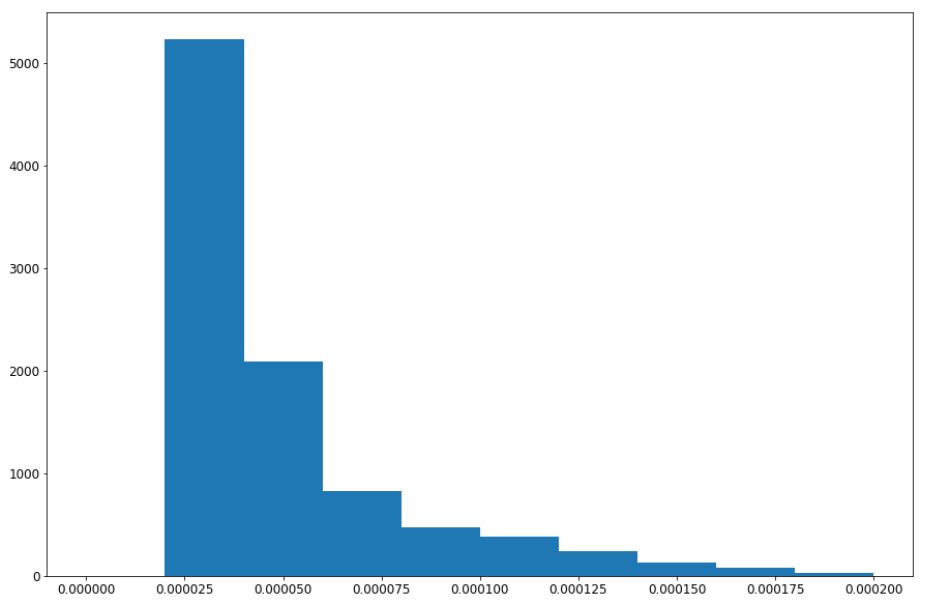


Figure 60:　棒グラフ (Bar graph)

棒グラフを見ると、データの値は、小さいほうに集中していることがわかる。つまり、小さい値をさらに分割したほうが、見やすい図になることが予想できる。

Looking at the bar graph, we can see that the values of the data are concentrated around the smaller values for x-axis near zero. In other words, it will be easier to see the results in detail if the smaller values are further divided in the figure to focuse on those high concentrated areas.

# custom colormap

custom\_cmap2 = mpl.colors.ListedColormap

(['white','lightgrey','skyblue',

'limegreen','lightgreen','gold','orchid',

'lightcoral','red'])

custom\_cmap2.set\_under('maroon')

custom\_cmap2.set\_over('darkgreen')

bounds = [0,0.1,1,2, 3, 5, 7.5, 10, 15 ,20]

bounds = np.array(bounds)\*0.00001

norm2 = mpl.colors.BoundaryNorm(bounds, custom\_cmap2.N)

mpl.colors.ListedColormapのリストを変更すれば、自由に色を変更できる。boundsのリストの値を変更すれば、自由に区切りも変更することができる。カラーリストの間をboundsの値で区切るので、考慮して個数を合わせないとエラーが出るので、そこだけ気を付けよう。

You can change the color into anything by changing the list of "mpl.colors.ListedColormap". You can also change the delimiter by changing the value of the "bounds" list. Since the color list is separated by the "bounds" value, an error will occur if the numbers are not matched without taking those into consideration.

def draw\_figure2(data):

fig = plt.figure(figsize=(15, 10))

plt.rcParams["font.size"] = 12

ax = plt.subplot(1, 1, 1, projection=ccrs.PlateCarree())

ax.coastlines(resolution="110m",color='grey')

ax.set\_extent([-180,180,-90,90], ccrs.PlateCarree())

ax.set\_title("NO$\_2$ 2019,Nov-Dec Mean (mol/m2)")

plt.ylabel('Latitude')

plt.xlabel('Longitude')

# setting grid of latitude and longitude

ax.set\_xticks([-180,-120,-60,0, 60, 120, 180],

crs=ccrs.PlateCarree())

ax.set\_yticks([-90, -60, -30, 0, 30, 60, 90],

crs=ccrs.PlateCarree())

lon\_formatter = LongitudeFormatter

(dateline\_direction\_label=True)

lat\_formatter = LatitudeFormatter()

ax.xaxis.set\_major\_formatter(lon\_formatter)

ax.yaxis.set\_major\_formatter(lat\_formatter)

im = plt.imshow(data,cmap=custom\_cmap2,norm=norm2,

extent=[-180,180,-90,90])

clb = plt.colorbar(aspect=20,shrink=0.6)

clb.set\_label("NO$\_2$(mol/m2)")

plt.tight\_layout()

fig.savefig(fig\_dir+"NO2\_2019\_Dec\_monthly\_mean\_image.jpg",

bbox\_inches="tight")

plt.show()

draw\_figure2(monthly\_mean\_data2)

変更点は、「cmap=custom\_cmap2,norm=norm2」のところだけだ。

The only change is "cmap = custom\_cmap2, norm = norm2".

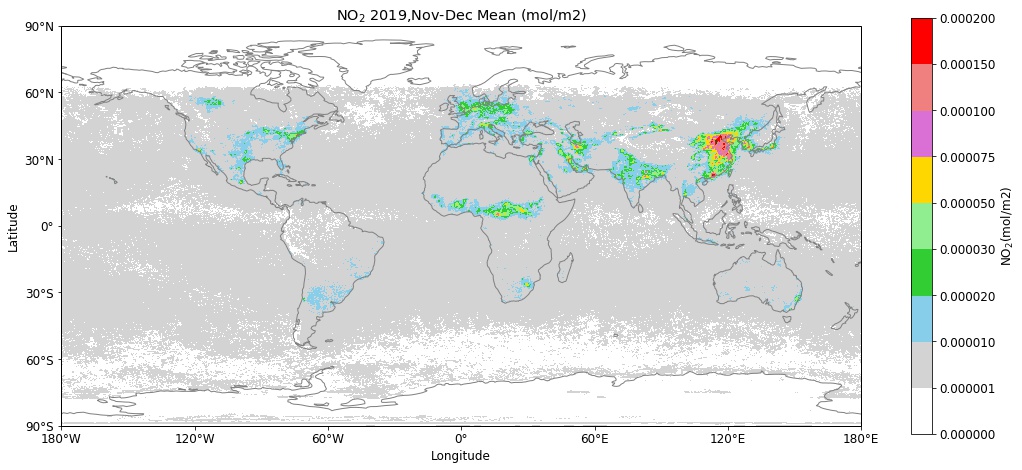


Figure 61:　Colormap変更後の図 (Figure after changing the colormap)

白と灰色の部分の差が明瞭になった。白の部分はほとんどデータなしの部分だろう。北極部のほうが白いのは、白夜のためだ。衛星は、光源が少ない夜には撮影できない。一日中夜である白夜の地域は、撮影ができないのだ。青や緑の部分の変化はこちらの図のほうが見やすくなっているだろう。デフォルトで提供されているcolormapで足りない時はこちらのコードを使おう。

The difference between the white and gray areas became obvious. The white area will be the part with almost no data. The Arctic looks whiter because of the midnight sun. Satellites cannot capture data at night when there are not enough light sources. It is not possible to shoot in the midnight sun area, where night continues all day long. The changes in the blue and green parts will be easier to see in this figure. If the colormap provided by default is not enough, use this code.

## 国ごとの時系列グラフ (Time series graph by country)

国ごとの時系列グラフを書いていこう。国ごとにデータを取るには、国ごとのマスクデータが必要だ。

dropboxから、

* global\_country\_mask.nc4
* convert\_FAO-COUNTRY.txt

のデータをダウンロードして、フォルダに整理しておこう。１つ目のデータがマスクデータだ。二つ目はCtlファイルのようなもので、国ごとの番号が載っている。つまり、ピクセル一つ一つに国の番号がついているので、その番号がついているindex（座標）を取得。NOxのデータの中でその座標に該当するデータを取ってきて、解析すれば、国ごとのデータが作れるという仕組みだ。

Write a time series graph for each country. To collect data for each country, mask data for each country is needed.

<https://www.dropbox.com/sh/5xvtf7vr4cbr0b8/AADNRvGdlzMRnankZdCGmsura?dl=0>

* global\_country\_mask.nc4
* convert\_FAO-COUNTRY.txt

Download the above two data from dropbox and organize them in a folder. The first data is mask data. The second is like a Ctl file, which contains country numbers. The point is that each pixel has a country number, so get the index (coordinates) of that number. Then, take the data of the coordinates from the NOx data and analyze it, creating the data for each country.

# ---------------setting--------------

# save directory of mask data

mask\_dir=r'C:\Users\kotsuki-lab-N05\Desktop\Python\Chapter5\maskdata/'

# save directory of image created

save\_dir=r'C:\Users\kotsuki-lab-N05\Desktop\Python\Chapter5/'

# choose country

# if you want, you can change the country from the country list below

country='Japan'

# country list for the mask data

countrycode={'Japan':110, 'Sweden':210,

'Norway':162,'Italy':106,'China':351,'Korea':117,

'England':229,'Germany':79,'US':231,'France':68,

'Brazil':21,'New Zealand':156,'Spain':203,

'Australia':11}

# -----------------setting end----------------

新たに、各設定に取得したデータをいれておこう。まずは、お馴染みの日本のデータからやろう。ちなみにcountrycodeは国コードの一覧をもちろん、読み込んでリスト化することもできるが、いくつかの国しか解析はしないので、手で打ったほうが早いと判断した。

Include the acquired data in the settings. Starting with Japan data. Also read "country code" and make it into a list. This time, only a few countries will be analyzed, so it's faster to type them by hand.

# read mask data and create mask index

countrymaskfile=nc.Dataset(mask\_dir+'global\_country\_mask.nc4','r')

maskdata = countrymaskfile.variables["mask1"][:].data[0]

mask=np.copy(maskdata)

mask = np.flipud(mask)

maskfocus=np.where(mask==countrycode[country],1,0)

maskindex=np.where(maskfocus==1)

maskindex

マスクデータも同じくnetCDFファイルになっているので、同じ読み込み方ができる。np.whereというのは、index（座標）を取り出せる便利な関数だ。一行目で、国別コードに該当するピクセルを1という数字に変換して、残りを0にしている。二行目で、その1の部分のindexを取ってくる。

次に、時系列グラフというのは、その名の通り、時系列の線グラフだ。今回はひと月分、つまり31日分のデータになる。線グラフにするので、31個分のデータがあればよい。ということは、720 x 1440の部分は平均化する必要がある。全球の緯度経度のデータを扱う時には、注意しなければいけないことがある。グラフを作成する前に、理解しておこう。

Since the mask data is also a netCDF file, it can be read in the same way. "np.where" is a convenient function to get the index (coordinates). The first line converts the pixel that matches the specified country code to 1 and if it does not match, to 0. The second line gets the index of the pixel of 1.

Next, the time series graph, as the name implies, is a line graph in time series. This time, the data is for a month (31 days). 31 data are required to make a line graph. This means that the 720 x 1440 part needs to be averaged. There are some things to be careful of when dealing with global latitude and longitude data. Understand them before creating the graphs.

## 加重平均 (Weighted average)

加重平均というのを聞いたことがあるだろうか？さっそくだが、一つ問題を出す。

Have you ever heard of a weighted average? Well, here is one question.

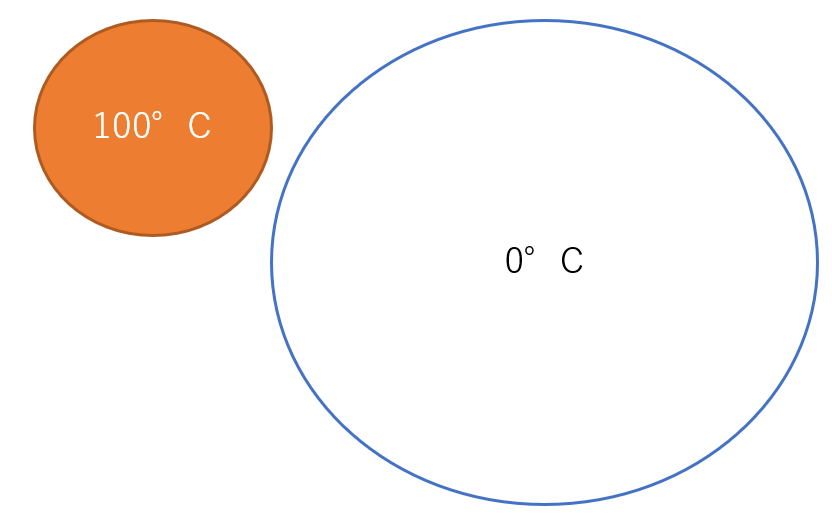


Figure 62: 100度のお風呂が1つ、0度の4倍の体積の水風呂が1つ

**(One 100 degree bath, one water bath with a volume four times that of 0 degree)**

上の図を見てほしい。100度の五右衛門風呂と0度の大きい水風呂がある。この二つの水をさらに大きいプールに入れるのを想像してほしい。この二つの水を合わせたら、温度はいくつになるだろうか？(100+0)/2＝50度だと思ったあなた。もう一度考えてほしい。

See the figure above. There is a “bathtub” at 100 ℃ and a cold water bath at 0 ℃. Imagine putting these two waters in a very large pool. What is the temperature of the water when these two waters are put together? If you think (100 + 0) / 2 = 50 ℃. Think again.

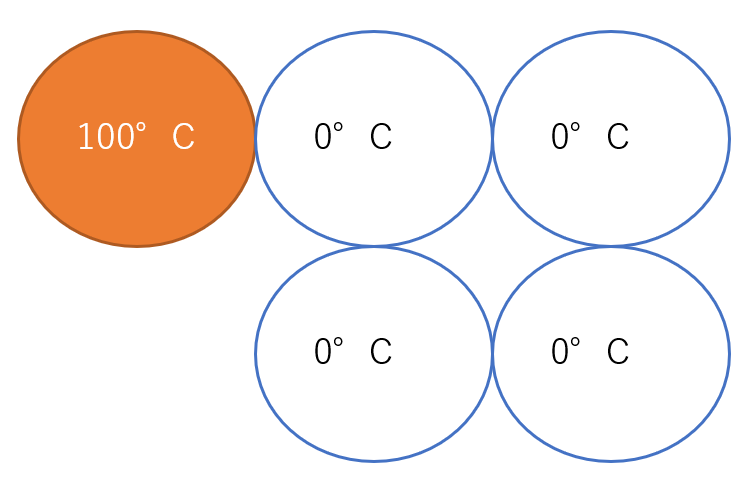


Figure 63: 100度のお風呂が1つ、0度の水風呂が4つ

**(One 100 degree bath, four 0 degree water baths)**

これならどうだろうか。5つの五右衛門風呂がある。先ほどの大きい水風呂を大きさの等しい４つの水風呂に分けた。体積は5つとも等しいとして、この5つの桶をプールに注いだとする。5つの合計で温度は何度になるか。(100+0+0+0+0)/5=20度だ。今回は等しい体積の元に計算をした。こちらが正しいことは感覚的にわかるだろう。これが重みづけ平均、つまり加重平均と言われるものだ。

How about this? There are 5 “bathtubs”. The large cold-water bath mentioned earlier was divided into four water baths of equal size. Assuming that all five have the same volume, pour these five tubs into one big pool. What is the temperature for the total of 5? (100 + 0 + 0 + 0 + 0) / 5 = 20 ℃. This time, the calculation was performed based on the same volume. You can see that this is correct. This is called “weighted average”.

では、地球の全球データの話をしよう。赤道での地球の一周の長さは、約40,075kmだ。では、一度分の距離は？これを360で割ればよい。だいたい111 kmぐらいだ。では、北極での一周の長さは？答えは、0 kmだ。北極点は点だからだ。仮に、今回は解像度が180 x 360の全球データの場合の話をしよう。1ピクセルが1度となる。赤道では、1ピクセルの大きさが、111 kmになり、北極付近では0 kmになってしまう。では、先ほどの桶の話からもわかるように、この二つの地点の気温をただ平均してしまったら、本当の平均値にならないことはわかってもらえるだろう。では、ある緯度のピクセルと赤道のピクセルの大きさの違いはどうやって求めたら良いのだろうか？

Now let's talk about global data of the earth. The length of the earth's circumference at the equator is about 40,075 km. Then, what is the distance of one degree? Divide the circumference by 360. It's about 111 km. Then, how long is the circumference at the North Pole? The answer is 0 km. Because the North Pole is a point. Considering the case of global data with a resolution of 180 x 360 at this time, 1 pixel is 1 degree. At the equator, one pixel is 111 km, and at the North Pole it is 0 km. Then, the story of the bathtubs earlier comes in handy. If just average of those two temperatures from equator and North Pole are considered, it will not be the true average value. So how is the difference in size between a pixel at a particular latitude and a pixel at the equator calculated?

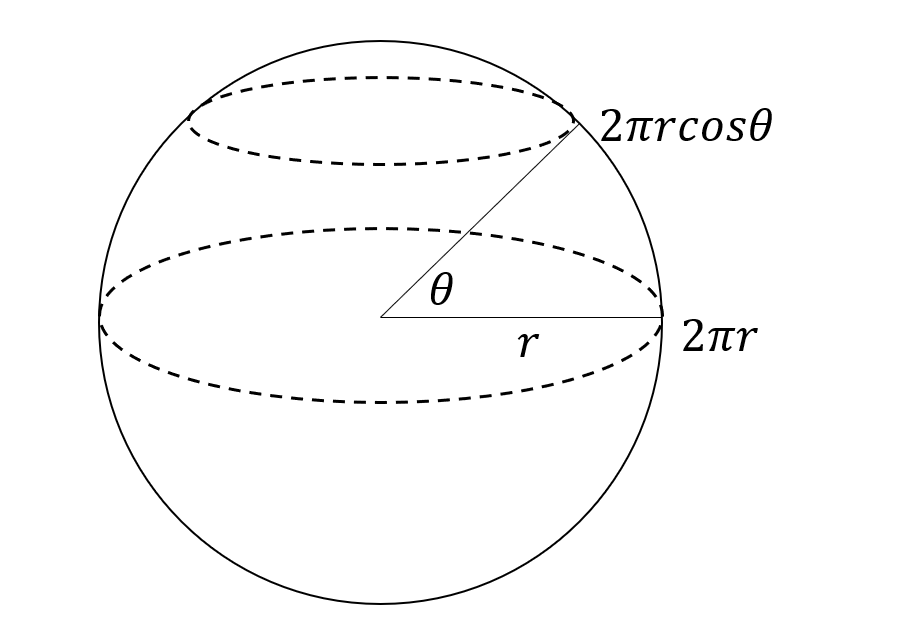


Figure 64: なぜ重みがcosθなのか？(Why is the weight cosθ?)

上の図を見てほしい。三角関数は覚えているだろうか？地球の中心から見て、赤道からある点までの角度をθとしている。そして、地球の半径をrとする。その時、赤道で円周の長さは2πrである。ある地点の円周の長さは、2πrcosθである。一周分を360で割れば、1度、つまり1ピクセルの経度方向の長さになる。ちなみに、緯度方向は中心から見た角度の弧の長さなので、一定である。ということは、赤道のピクセルサイズを1×1としたら、緯度θの点のピクセルサイズはcosθ×1ということになる。確かめだが、90度で、cos90°は0になる。北極点なので、0度で正解。

See the figure above. Remember trigonometric functions? The center of the earth is the vertex, and the angle from the equator to a certain point is θ. And the radius of the earth is “r”. At this time, the circumference is “2πr” at the equator. The circumference at a certain point is “2πrcosθ”. Dividing the circumference by 360 gives one degree, that is, the length of one pixel in the longitude direction. By the way, since the length of the arc at an angle from the center is constant, the latitude direction is constant. That is, if the pixel size at the equator is 1 × 1, the pixel size at the point of latitude θ is cos θ × 1. Confirm that this is correct. At 90 degrees, cos 90 °is 0, and 0 degrees at the North Pole is correct.

Figure 65: 加重平均の式 (Weighted average formula)

加重平均の式を紹介。*W*というのが重み、*X*というのが値を表している。数式に馴染みがないという人もいると思うので、わかりやすく先ほどのお風呂を例にして説明する。先ほどのお風呂の例を加重平均の式に当てはめると、(100x1+0x4)/(1+4) = 20となる。*X*1=100, *X*2=0、*W*1=1,*W*2=4だ。重みというのは、お風呂１つを１という基準にした時のそのほかの値の重みだ。全球で考えた時も、赤道を基準として１として考えた時の重みである。ちなみに、分母と分子両方に同じものをかけると単位は変わらない。なので、加重平均をしても単位は変わらないということも頭にいれておいてほしい。

余談だが、実は緯度による面積の違いを考慮していない平均の仕方を3章では、やってしまっている。なぜなら、ピクセルの説明をしてからではないと、理解してもらえないと考えたからだ。ピクセルの話は4章でしたと思う。なので、この章が理解できたら、3章に戻って、加重平均を実装してみてほしい。

それでは、加重平均を理解してもらったところで、プログラムに組み込んでいこう。

Figure 65 shows the weighted average formula. *W* is the weight and *X* is the value. Some people aren't familiar with mathematical formulas, so an easily understood bath example will be explained again. Applying the previous bath example to the weighted average equation, we get (100x1 + 0x4) / (1+4) = 20. *X*1 = 100, *X*2 = 0, *W*1 = 1, *W*2 = 4. The weight is the weight of other values when one bath is set to 1. Even when it comes to globe, it is the weight when the value of the equator is set to 1. Multiply both the denominator and the numerator by the same thing, the unit will not change. Therefore, keep in mind that the unit does not change even if a weighted average is performed.

Previously, the averaging method was done in Chapter 3 without considering the difference in area depending on the latitude. Because it would be difficult to understand until pixels were explained in Chapter 4. So, once understanding this chapter, go back to Chapter 3 and try to calculate the weighted average.

Now understanding the weighted average, create a program.

# apply mask

# calculate weighted mean

masked\_weight\_mean\_data = np.zeros((len(all\_data)),dtype=float)

for i in range(len(all\_data)):

# reset the values

weightcaldata = 0

denominator = 0

for t in tqdm(range(len(maskindex[0]))):

value = all\_data[i,maskindex[0][t],maskindex[1][t]]

latitude = lat[maskindex[0][t]]

if value > 0:

weightcaldata += value \* np.cos(np.radians(latitude))

denominator += np.cos(np.radians(latitude))

if denominator > 0:

weightmeandata = weightcaldata / denominator

else:

weightmeandata = 0

masked\_weight\_mean\_data[i] = weightmeandata

np.save(data\_dir+'masked\_weight\_mean\_data.npy', masked\_weight\_mean\_data)

masked\_weight\_mean\_data

Alldataとして読み込んでいたものを計算していく。iで31日分データを回して、重みの分母と分子を分けて計算する。まず、両方0にしておく。tで国別データを取ってきた座標を回し、座標に該当する値を取得。緯度も取得しておく。ここで注意しなければいけないのが、missing valueの座標があることだ。値がないところは平均には使えないので、if関数を使って、はじくようにしている。これだけで実行すると、0で計算できないというエラーが出てくる。なぜなら、全部データがない日があるからだ。分母が0になると計算できない。なので、ifを使って、denominatorが0になる場合は、値を0にするようにしている。

Read the data as "all data" and start the calculation. Repeat the "for loop" for 31 days with "i", and calculate the denominator and numerator of the weight separately. First, set both to 0. Repeat the "for loop" of the country-specific data coordinates with "t", and acquire the value and latitude of those coordinates. One thing to note here is that some coordinates have no value. If there is no value, it cannot be averaged, so the "if" function checks it. When this is run, an error still comes up that 0 cannot be calculated. Because there are days when there is no data at all. If the denominator is 0, it can’t be calculated. So, in such case, set the value to 0.

ここで、豆知識を紹介する。

Here are some tips.

# for second time, comment out the above code and apply this code

masked\_weight\_mean\_data=np.load(data\_dir+'masked\_weight\_mean\_data.npy')

masked\_weight\_mean\_data

今回はひと月分しかデータがないが、もし30年分のデータを扱いたいとなった時、おそらく計算は30分以上かかることになるだろう。その場合、毎回計算が必要となったら、いつまでたっても先に進めなくなってしまう。そんな時は、計算して作られた配列を保存すればよい。それが上のコードだ。有効活用してほしい。

This time only a month of data will be handled, but if 30 years of data is wanted, the calculation will probably take more than 30 minutes. In that case, every time the code runs, a calculation is needed, and it takes a long time. In such a case, saving the array created by the calculation can be done. The code above does that. Make an effective use of it.

それでは、加重平均をしたデータを使って、時系列グラフを書いてみよう。

Now, draw a time series graph using the weighted averaged data.

# plot time series

fig = plt.figure(figsize=(20, 10))

plt.rcParams["font.size"] = 15

ax = plt.subplot()

ax.plot(dates,masked\_weight\_mean\_data)

fig.autofmt\_xdate()

ax.set\_title("Time Series of NO$\_2$ Distribution in "+

country +" (mol/m2)")

plt.ylabel('Amount of NO$\_2$(mol/m2)')

plt.xlabel('Time')

Filename = "201912\_TimeSeries-NO2-"+ country +".png"

plt.savefig(fig\_dir+Filename,bbox\_inches="tight")

plt.show()

いつも通り、figureを作って、そのなかにsubplotで作っていく。pd.date\_rangeというのは、pandasで日にちデータを作れるもので、x軸のラベルに使っている。詳しくは調べてみてほしい。他は今までの知識で大丈夫だろう。

As usual, "figure" creates a plot area, and "subplot" creates a graph in it. "pd.date\_range" creates date data with pandas, which is the x-axis label. Please search for more details if interested. The other code has been explained so far, so it should be okay.

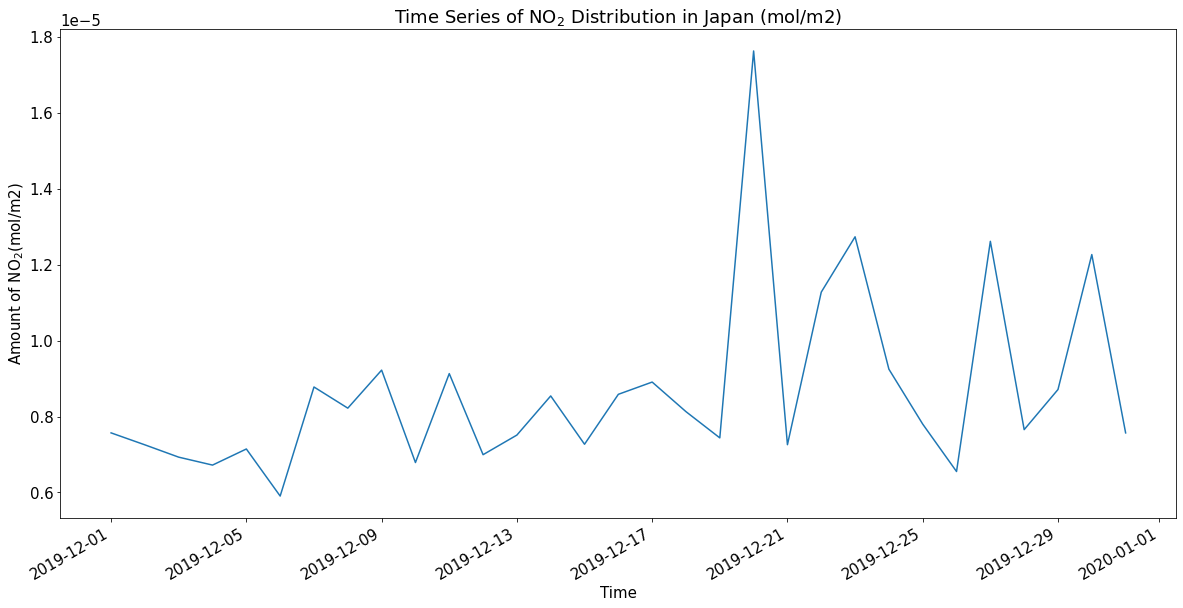


Figure 66: NO2の時系列グラフ　２０１９年１２月　日本

(NO2 time series graph December 2019 Japan)

これで、5章は終わりだ。これで、データを調べて、ダウンロードして、解析することまでできるようになったはずだ。これであなたも一人前のデータサイエンティストだろう。ここからは、自分で調べて、さらにできる技術を増やしていこう。ここまで読んで頂けたことに深く感謝する。何か気になった点などを見つけたら、執筆者まで連絡を頂けるとありがたい。不備があれば、修正する。一つの章として成り立ちそうな面白い質問が来たら、また章を追加しようと思っている。加筆がされるかは、あなた次第ということだ。ネタが溜まり次第、また加筆修正する可能性もがあるが、今のところ予定はない。それでは、また逢う日まで。

This is the end of Chapter 5. Now it should be possible to find, download, and analyze data. Now you are a data scientist. From now on, research on your own and expand your abilities. I am deeply grateful to you for reading until the end. If there are any questions or things to be aware of, please contact the author. If any problems with this manual are found, I'll fix them. If I receive an interesting question that I can explain as a new chapter, I will add it. So, it's up to you if this manual will be updated or not. I may update it as soon as I have a new idea. However, there are no plans so far. Well then, until the day we meet again.