理学部学生にとても役立つ『Python講座』

講師　西濱大将
（理学部物理学科３年）
daisukenishihama63@gmail.com

2023/02/08

目次

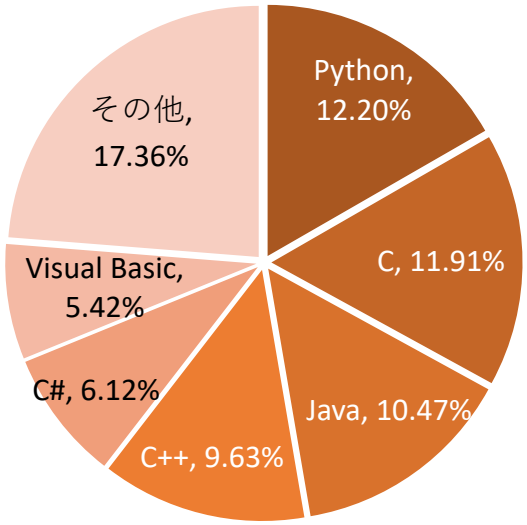
1. Pythonを使うメリット
2. 本日の講義の目標
3. Python環境の準備
4. Pythonの基本的操作を習得する
    1. 算術演算
    2. ビット演算
    3. 比較演算子・ブール演算子
    4. if文・while文・for文
5. 自然数 n の素数判定
6. 2つの電荷のポテンシャル
7. フーリエ解析（月は地球の空気を動かしている？）

Pythonを使うメリット

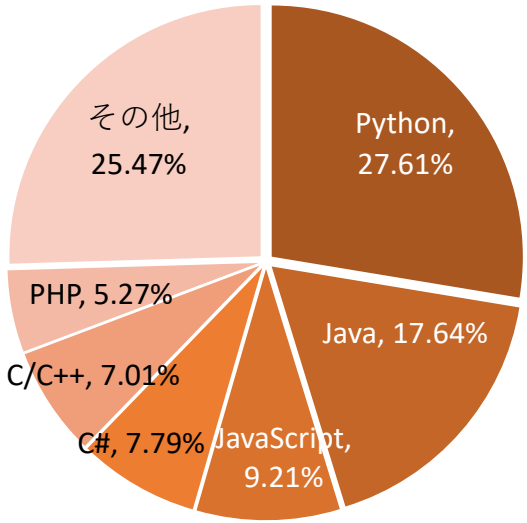
- 非常に大きなコミュニティ
  - 世界中で沢山の人が使っている
  - 調べれば何でも出てくる
- 豊富なライブラリー
  - 簡単に複雑なことをたくさんできる
  - 覚えることが少ない
  - 色んなことができる
- 簡単に素早くコードが書ける
  - 型（整数・小数・文字列）について考える必要がない
  - 直感的に書くことができる

Pythonを使うメリット

- 非常に大きなコミュニティ
  - 世界中で沢山の人が使っている
  - 調べれば何でも出てくる



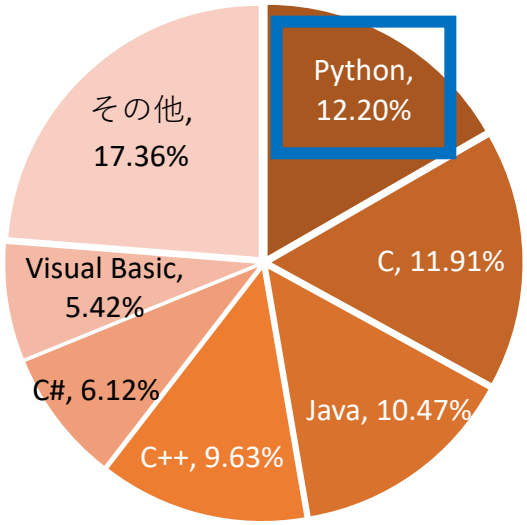
Source: TIOBE Index



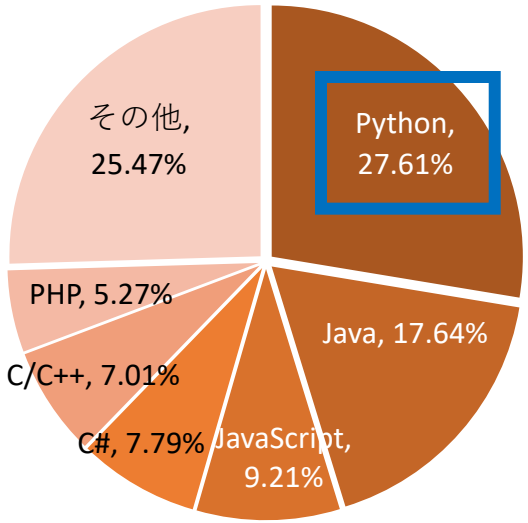
Source: PYPL PopularitY of Programming Language Index

Pythonを使うメリット

- 非常に大きなコミュニティ
  - 世界中で沢山の人が使っている
  - 調べれば何でも出てくる



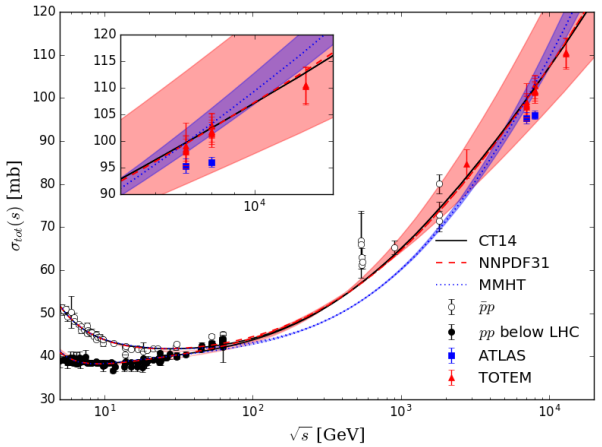
Source: TIOBE Index



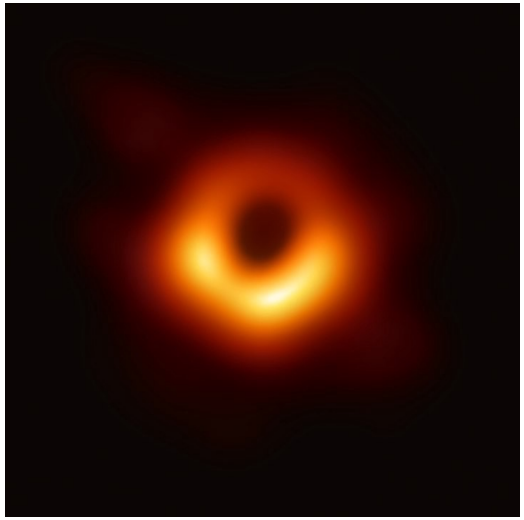
Source: PYPL PopularitY of Programming Language Index

Pythonを使うメリット

- 豊富なライブラリー
  - 簡単に複雑なことをたくさんできる
  - 覚えることが少ない
  - 色んなことができる



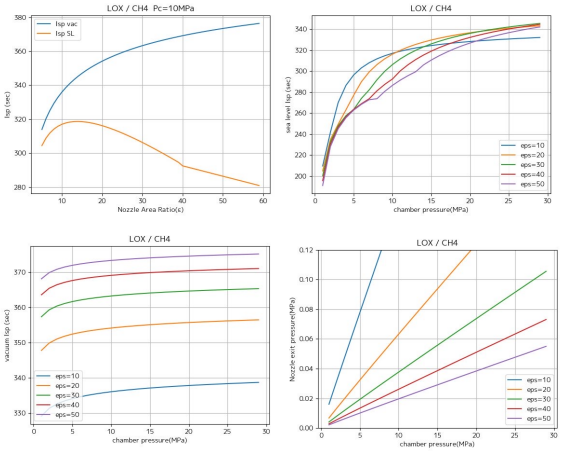
Mateus Broilo / How to create cool figures with Matplotlib



Black Hole M87 (Image Credits: Event Horizon Telescope Collaboration)

scikit-learn

pandas



いな | 稲川貴大（ロケット屋）
Twitter@ina111 / RocketCEA

Pythonを使うメリット

- 簡単に素早くコードが書ける
  - 型（整数・小数・文字列）について考える必要がない
  - 直感的に書くことができる

C++の場合

Pythonの場合

Pythonを使うメリット

- 簡単に素早くコードが書ける
  - 型（整数・小数・文字列）について考える必要がない
  - 直感的に書くことができる

C++の場合

```cpp
#include <iostream>

int main(){
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Pythonの場合

Pythonを使うメリット

- 簡単に素早くコードが書ける
  - 型（整数・小数・文字列）について考える必要がない
  - 直感的に書くことができる

C++の場合

```cpp
#include <iostream>

int main(){
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Pythonの場合

```python
print("Hello, World!")
```

Pythonを使うメリット

- 簡単に素早くコードが書ける
  - 型（整数・小数・文字列）について考える必要がない
  - 直感的に書くことができる

C++の場合

```cpp
#include <iostream>

int main(){
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

さらにコンパイル＆実行が必要！！

Pythonの場合

```python
print("Hello, World!")
```

インタープリタ言語のため実行だけで十分！

Pythonを使うメリット

- 簡単に素早くコードが書ける
  - 型（整数・小数・文字列）について考える必要がない
  - 直感的に書くことができる

C++の場合

```cpp
#include <iostream>

int main(){
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

機械語に翻訳する作業

↑

さらにコンパイル＆
実行が必要！！

Pythonの場合

```python
print("Hello, World!")
```

自動で一行ずつ機械語に
翻訳して実行する言語

↑

インタープリタ言語の
ため実行だけで十分！

Pythonを使うメリット

- プログラミング言語 Python は初学者にも学びやすい言語である一方で，さまざまな応用も可能です.
- 近年では学術研究にも利用が広がっています.

喜多, 一 ...[et al]. プログラミング演習 Python 2021. 2021: 1- 239

本日の講義の目標

- Pythonの基本的操作を習得する
- 理学的な応用を実際にやってみる
  - 自然数 n の素数判定
  - ２つの電荷のポテンシャル
  - フーリエ解析（月は地球の空気を動かしている？）
  - ~~ライブラリーを入れてみる（化合物の描画）~~
- 機械学習を実践してみる
  - ~~白黒画像をカラー画像に？~~

Python環境の準備

- PC内で実行環境を整える必要がある
  - →初心者にとっては大変＆難しい…
- 時間の都合上で，Google Colabを使用
  - https://colab.research.google.com/
  - 環境を整える必要がない
  - オンライン上でできる

算術演算


```
+a              # 正数
-a              # 負数
a + b           # 加算
a - b           # 減算
a * b           # 乗算
a / b           # 除算
a % b           # a を b で割った余り
a ** b          # a の b 乗
a // b          # 切り捨て除算
```

⇐Pythonの算術演算

【参考文献】とほほのPython入門 - 演算子　https://www.tohoho-web.com/python/operators.html

算術演算


```
+a          # 正数
-a          # 負数
a + b       # 加算
a - b       # 減算
a * b       # 乗算
a / b       # 除算
a % b       # a を b で割った余り
a ** b      # a の b 乗
a // b      # 切り捨て除算
```

⇐Pythonの算術演算

⚠️注意

一般に a の b 乗は a^b と記述することが多いが，Python では異なる！

算術演算

問題　次のPythonで計算し13で割ったときの余りはいくつか

$$\left[\left(2^{100}-4\right)\div 2\right]+\left[7118^{9}\div 3\right]$$

※　[x]はガウス記号でxそれを超えない最大の整数値を表す

```
+a          # 正数
-a          # 負数
a + b       # 加算
a - b       # 減算
a * b       # 乗算
a / b       # 除算
a % b       # a を b で割った余り
a ** b      # a の b 乗
a // b      # 切り捨て除算
```

⇐Pythonの算術演算

⚠️注意

一般に a の b 乗は a^b と記述することが多いが，Pythonでは異なる！

算術演算

問題　次のPythonで計算し13で割ったときの余りはいくつか

$$\left[\left(2^{100}-4\right)\div 2\right]+\left[7118^{9}\div 3\right]$$

※　[x]はガウス記号でxを超えない最大の整数値を表す

```
+a              # 正数
-a              # 負数
a + b           # 加算
a - b           # 減算
a * b           # 乗算
a / b           # 除算
a % b           # a を b で割った余り
a ** b          # a の b 乗
a // b          # 切り捨て除算
```

⇐Pythonの算術演算

⚠注意

一般に a の b 乗は a^b と記述することが多いが，Python では異なる！

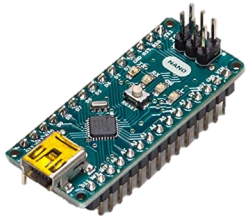
Answer：　8

```
# 計算はこれで十分
((2**100 - 4) // 2 + 7118**9 // 3) % 13
```

ビット演算

- コンピュータは電流が流れている状態と流れていない状態の２つしか認識できない.
  - １：流れている状態，０：流れていない状態
    - ２進数に対応
    - ２進数の１桁 = 1 bit (ビット)
    - 8桁 (8bit) = 1 byte (バイト) を１つの塊と考える

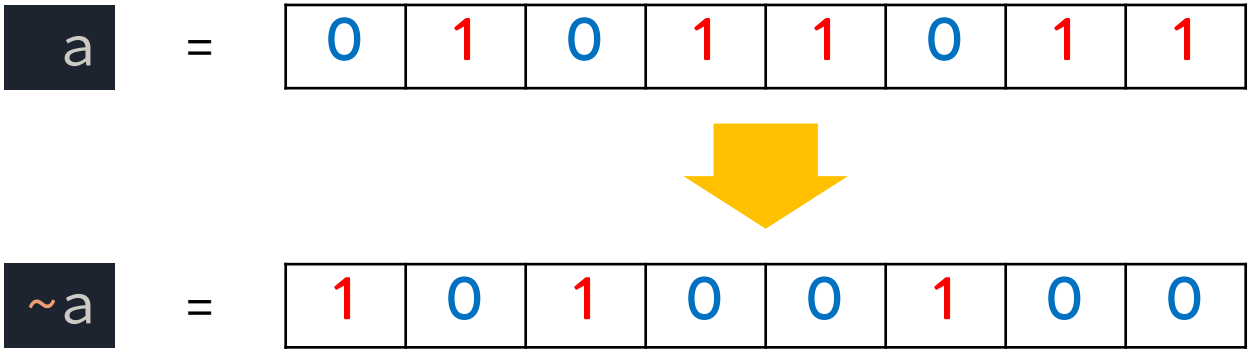
✅ バイナリーレベルでのシステム開発時に必要不可欠



ビット演算


```
~a          #  ビット反転
a  &  b     #  AND:論理積(aもbも1のビットが1)
a  |  b     #  OR:論理和(aまたはbが1のビットが1)
a  ^  b     #  XOR:排他的論理和(aまたはbが1のビットが1)
a  <<  b    #  b  ビット左シフト
a  >>  b    #  b  ビット右シフト
```

イメージ図

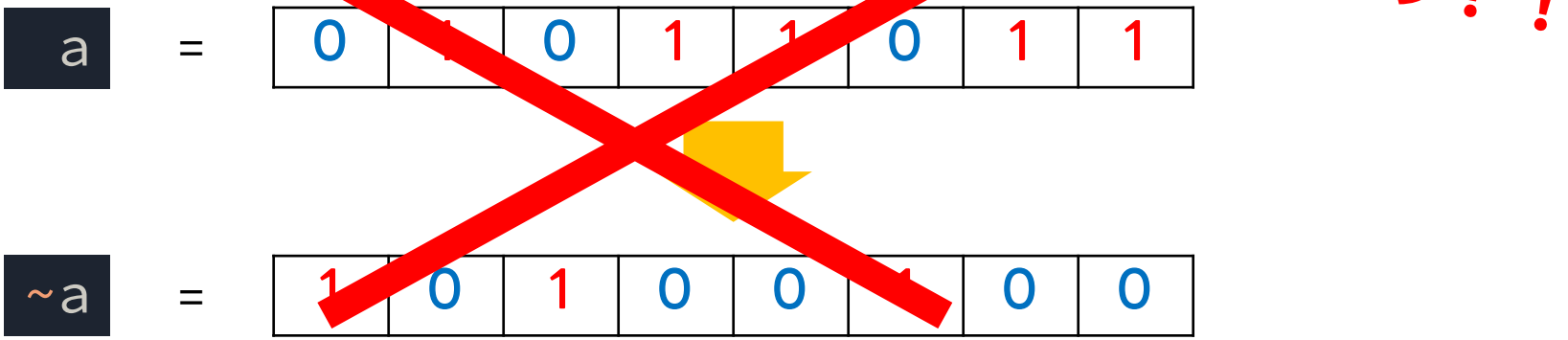


ビット演算


```
~a          #  ビット反転
a & b       #  AND:論理積(aもbも1のビットが1)
a | b       #  OR:論理和(aまたはbが1のビットが1)
a ^ b       #  XOR:排他的論理和(aまたはbが1のビットが1)
a << b      #  b ビット左シフト
a >> b      #  b ビット右シフト
```

イメージ図

実際はもう少し違う！！



ビット演算


```
~a          #  ビット反転
```

- ビット「反転」ということで `~~a = a` を満たす必要がある
  - 一対一対応する必要がある

$$0101 \longleftrightarrow 1010$$

- 最上位ビットは符号を意味する
  - 反転することによって負の数字を加算で計算したい

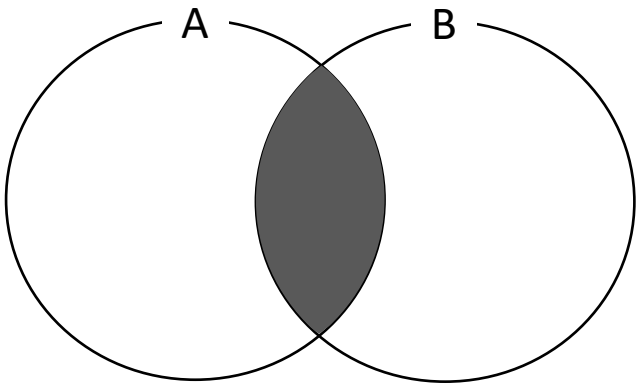
$$\boxed{0}001 \rightarrow$$ 数字：0～7　（有限体）

$$7 - 1 = 7 + \underbrace{(-1)}_{1000)_2} = 7 + \underbrace{7}_{0110)_2} = 6$$

ビット演算


```
~a          #  ビット反転
a  &  b     #  AND:論理積(aもbも1のビットが1)
a  |  b     #  OR:論理和(aまたはbが1のビットが1)
a  ^  b     #  XOR:排他的論理和(aまたはbが1のビットが1)
a  <<  b    #  b  ビット左シフト
a  >>  b    #  b  ビット右シフト
```

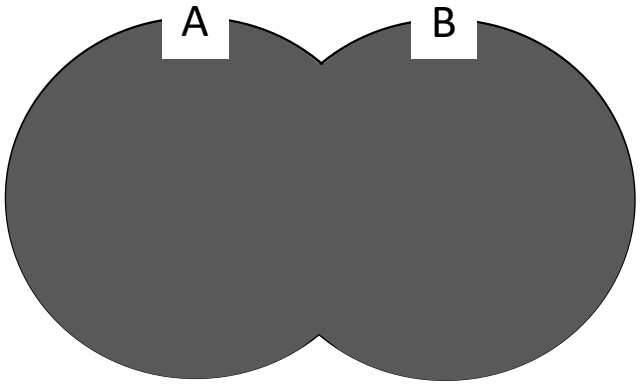
イメージ図



ビット演算


```
~a          #  ビット反転
a  &  b     #  AND:論理積(aもbも1のビットが1)
a  |  b     #  OR:論理和(aまたはbが1のビットが1)
a  ^  b     #  XOR:排他的論理和(aまたはbが1のビットが1)
a  <<  b    #  b  ビット左シフト
a  >>  b    #  b  ビット右シフト
```

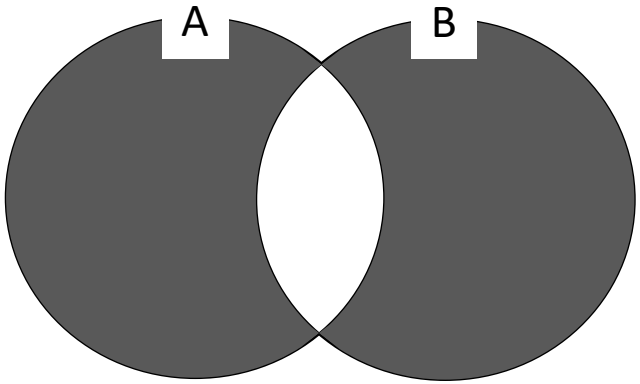
イメージ図



ビット演算


```
~a          #  ビット反転
a  &  b     #  AND:論理積(aもbも1のビットが1)
a  |  b     #  OR:論理和(aまたはbが1のビットが1)
a  ^  b     #  XOR:排他的論理和(aまたはbが1のビットが1)
a  <<  b    #  b  ビット左シフト
a  >>  b    #  b  ビット右シフト
```

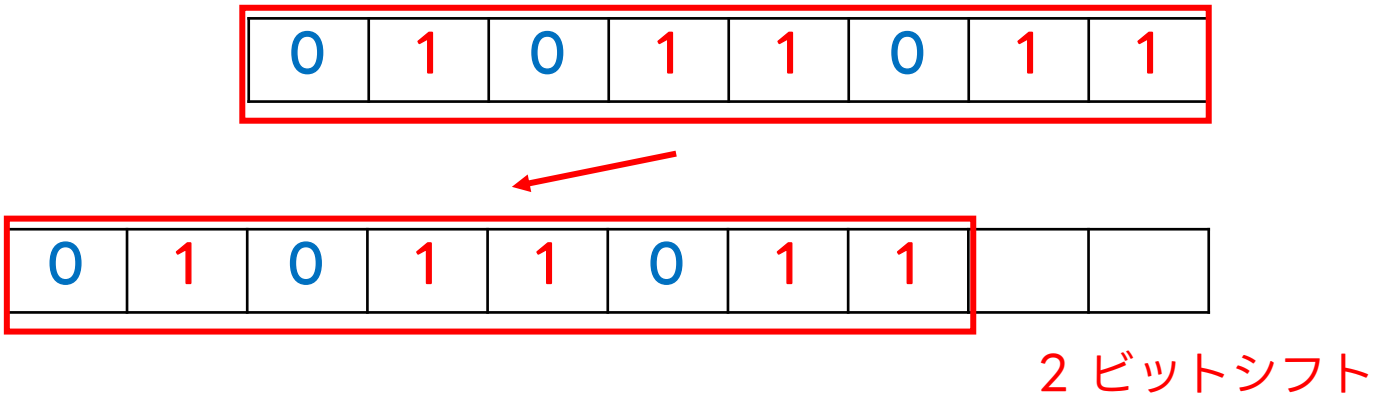
イメージ図



ビット演算


```
~a          #  ビット反転
a  &  b     #  AND:論理積(aもbも1のビットが1)
a  |  b     #  OR:論理和(aまたはbが1のビットが1)
a  ^  b     #  XOR:排他的論理和(aまたはbが1のビットが1)
a  <<  b    #  b  ビット左シフト
a  >>  b    #  b  ビット右シフト
```

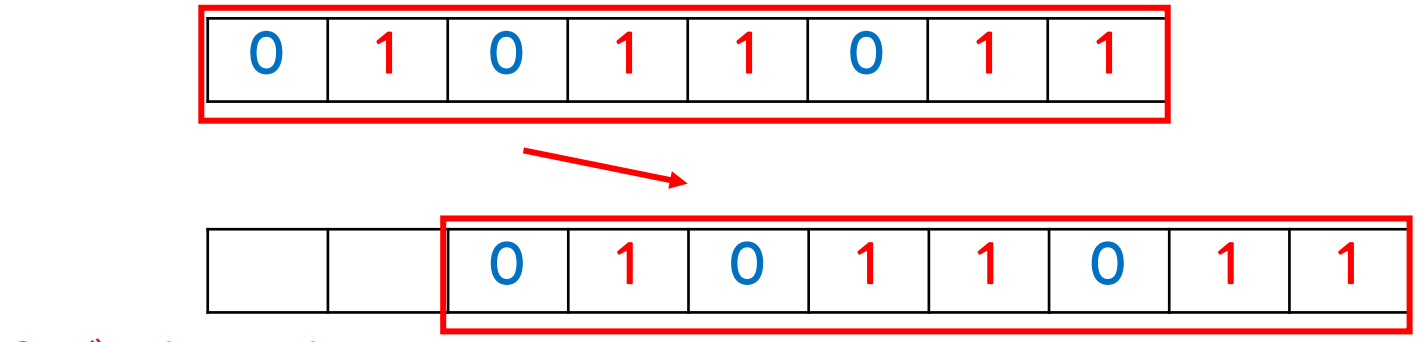
イメージ図



ビット演算


```
~a          #  ビット反転
a  &  b     #  AND:論理積(aもbも1のビットが1)
a  |  b     #  OR:論理和(aまたはbが1のビットが1)
a  ^  b     #  XOR:排他的論理和(aまたはbが1のビットが1)
a  <<  b    #  b  ビット左シフト
a  >>  b    #  b  ビット右シフト
```

イメージ図



比較演算子・ブール演算子

比較演算子

```
a == b       # a が b と等しい
a != b       # a が b と異なる
a < b        # a が b よりも小さい
a > b        # a が b よりも大きい
a <= b       # a が b 以下である
a >= b       # a が b 以上である
a is b       # a が b と等しい
a is not b   # a が b と異なる
a in b       # a が b に含まれる
             (a, b は共に文字列、または、b はリストやタプル)
a not in b   # a が b に含まれない
             (a, b は共に文字列、または、b はリストやタプル)
```

- “a = b”ではbをaに代入するという意味になるので注意
- a < x < bと書くと a < x and x < b と等価になる

ブール演算子

```
a and b      # a も b も真であれば真
a or b       # a または b が真であれば真
not a        # a が偽であれば真
```

if文・while文・for文

とほほのPython入門 - 制御構文
https://www.tohoho-web.com/python/control.html#break

コロン忘れずに！

```
if expr1:
    suite1...
elif expr2:
    suite2...
else:
    suite3...
```

Trueだったら

Falseだったら

インデントまたは空白4文字

break: ループを抜ける
continue: 次のループ処理に移動

Trueだったら

```
while expr:
    suite1...
else:
    suite2...
```

Falseだったら

- リスト
- タプルの各要素
- 辞書のキー
- 文字列の各文字
- ファイルの各行

etc

```
for var in expr:
    suite1...
else:
    suite2...
```

自然数 n の素数判定


```python
def is_prime(i):
    if i <= 1:
        return False
    for j in range(2, int(i**0.5) + 1):
        if i % j == 0:
            return False
    return True

def is_prime(num):
    if num <= 1:
        return False
    else:
        for i in range(2, num):
            if num % i == 0:
                return False
            else:
    return True

print(is_prime(100))
```

総当りで計算

素数ではない場合、2からnの平方根までの間に必ず約数が存在するので、その間に約数がなければ素数と判断できる

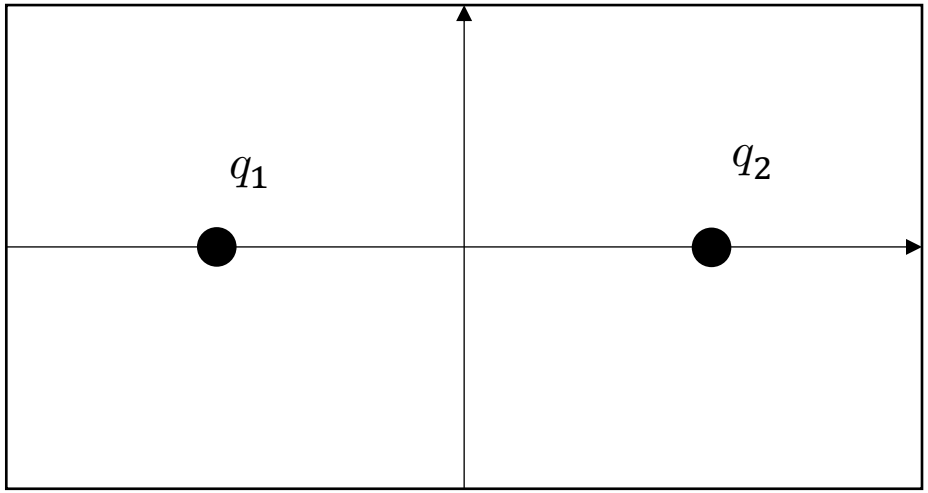
2つの電荷のポテンシャル

- 2つ位置 $\mathbf{r}_1$, $\mathbf{r}_2$ の電荷 q_1 q_2 のポテンシャルは

$$\phi(\mathbf{r}) = \frac{1}{4\pi\varepsilon}\frac{q_1}{|\mathbf{r}-\mathbf{r}_1|} + \frac{1}{4\pi\varepsilon}\frac{q_2}{|\mathbf{r}-\mathbf{r}_2|}$$

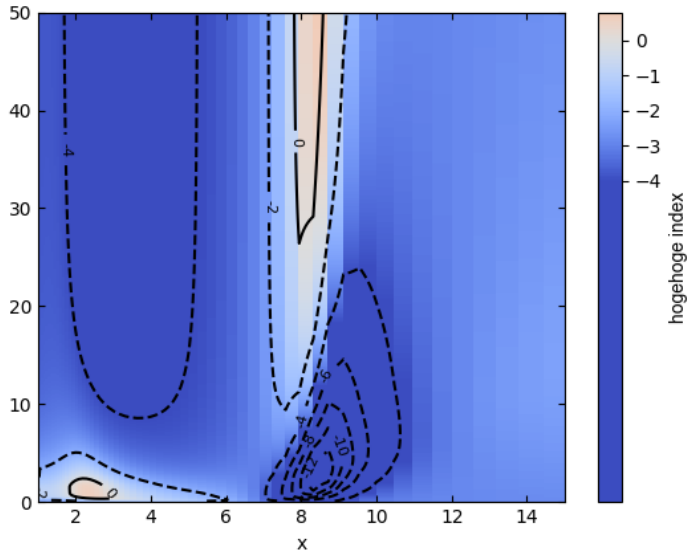
  で与えられる.

- 計算機上では係数は規格化する $(4\pi\varepsilon = 1)$.



2つの電荷のポテンシャル

- 今回の方針
  - xy平面上に2つの電荷をおいたときのポテンシャルをz軸にとり、それを色の濃さで2次元上に表現する.



←イメージ図

Kosuke @kumamupooh
matplotlibでカラーバーの範囲を思い通りにする – Qiita
https://qiita.com/kumamupooh/items/c793a6781a753eca6d8e

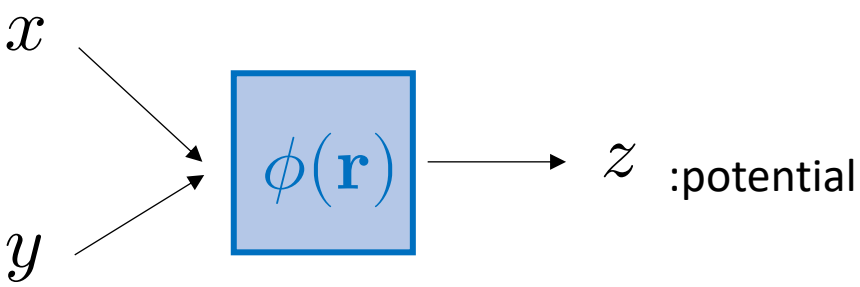
2つの電荷のポテンシャル

任意の位置ベクトルをxy座標で書き下す

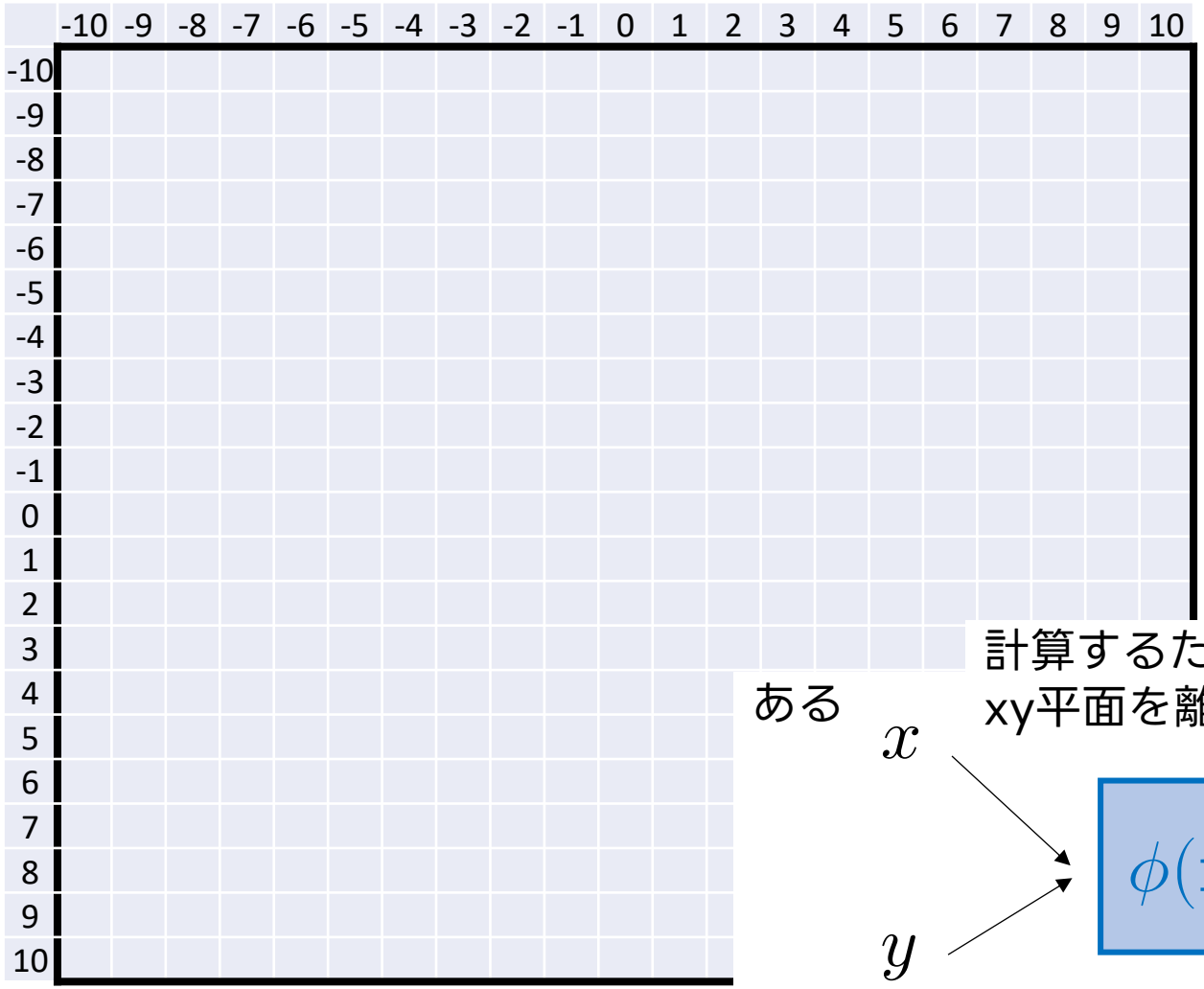
$$\phi(\mathbf{r}) = \frac{1}{4\pi\varepsilon}\frac{q_1}{|\mathbf{r}-\mathbf{r}_1|} + \frac{1}{4\pi\varepsilon}\frac{q_2}{|\mathbf{r}-\mathbf{r}_2|}$$

$$= \frac{q_1}{\sqrt{(x-x_1)^2+(y-y_1)^2}} + \frac{q_2}{\sqrt{(x-x_2)^2+(y-y_2)^2}} = \phi(x,y)$$

ある

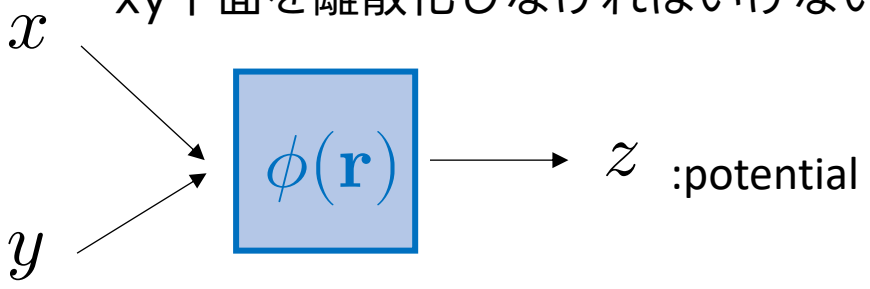


2つの電荷のポテンシャル



ある

計算するためには
xy平面を離散化しなければいけない



2つの電荷のポテンシャル


```
import numpy as np
```

NumPy（ナムパイ）とは、多次元配列を効率的に扱うライブラリです。 Pythonの標準ライブラリではありませんが、科学技術計算や機械学習など、ベクトルや行列の演算が多用される分野では、事実上の標準ライブラリとしての地位を確立しています。

- ✓ Pythonは遅いと言われがちですが，Numpyの内部実装はC言語（とFortran）なので非常に高速
- ✓ 世界で初めて撮影されたM87 Black Holeの大量の解析でも使用された
- ✓ 重力波の解析LIGOでも使用された

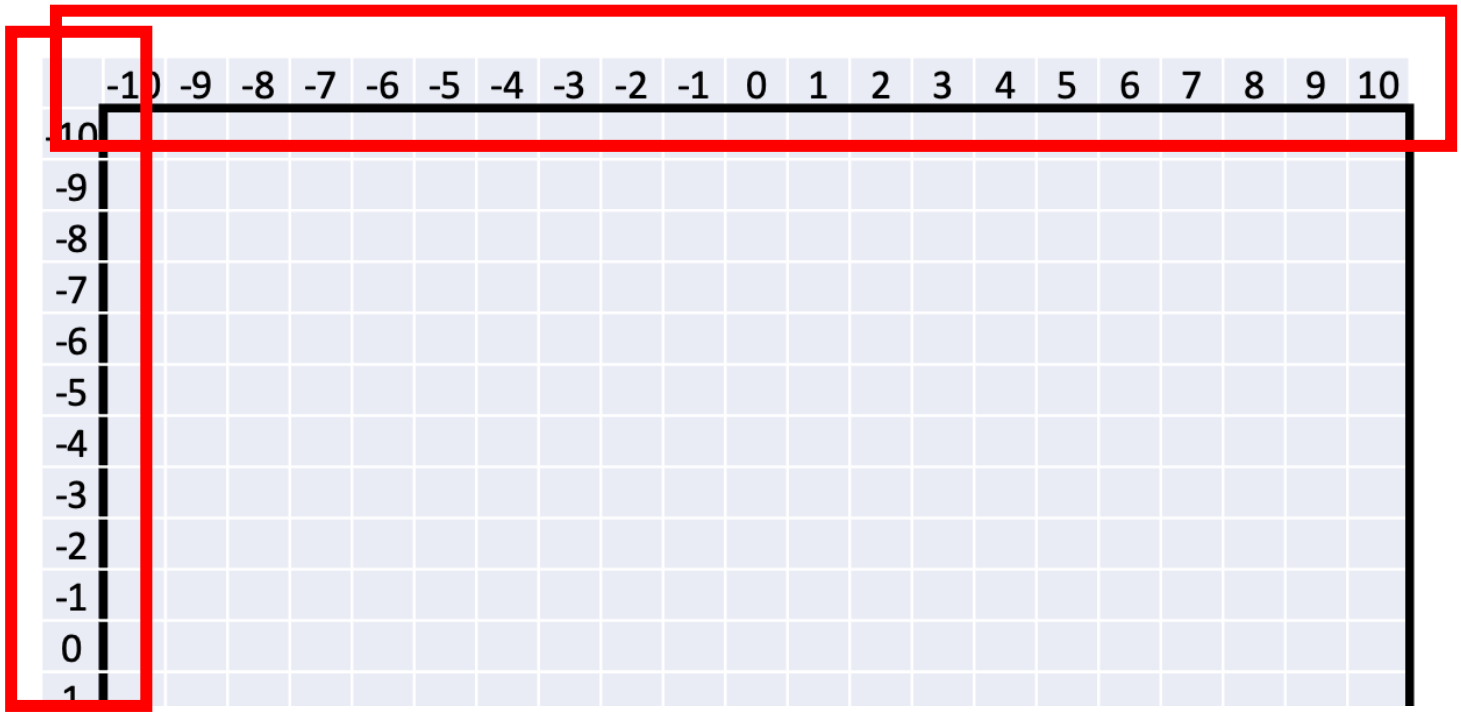
```
a = np.array([[1,2,3],
              [0,0,4],
              [6,4,3]])
b = np.linalg.inv(a)    →逆行列
print(b)

c = a@b     →行列演算
print(c)
```

今回は使いませんが，
試しにやってみよう！

2つの電荷のポテンシャル


```
X = np.linspace(-2,2,50)
Y = np.linspace(-2,2,50)
```

np.linspace(-2,2,50) は-2から2を50分割したベクトルを生成する
- ✓ 50はデフォルトなのでnp.linspace(-2,2)でも良い



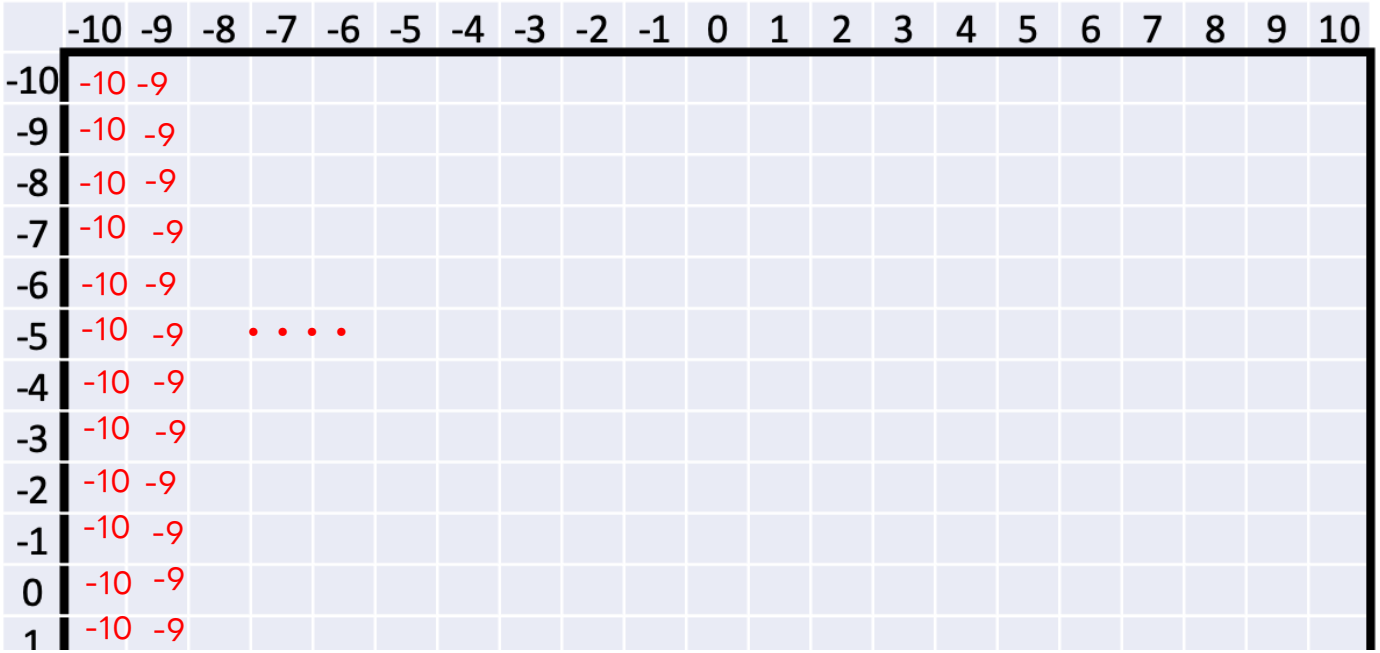
２つの電荷のポテンシャル


```
x, y = np.meshgrid(X, Y)
```

np.meshgridは指定した各軸の要素列から各軸方向の格子列を返します。

By Documentation

X座標に注目したとき以下のような2次元配列になっている
このような２次元配列を生成してくれる



2つの電荷のポテンシャル


```
z = 1/np.sqrt((x-1)**2 + (y)**2)
        - 1/np.sqrt((x+1)**2 + (y)**2)
```

なぜこのような記述で済むのか？

四則演算は次のように定義されています：

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + 10 = \begin{bmatrix} 11 & 12 \\ 13 & 14 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} = \begin{bmatrix} 11 & 22 \\ 33 & 44 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 10 & 20 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \boxed{\begin{bmatrix} 10 & 20 \\ 10 & 20 \end{bmatrix}} = \begin{bmatrix} 11 & 22 \\ 13 & 14 \end{bmatrix}$$

2つの電荷のポテンシャル

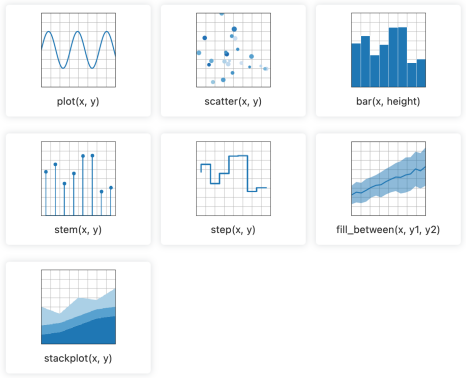

```
import matplotlib.pyplot as plt
```

matplotlib

様々グラフを描画することができる優れたライブラリ

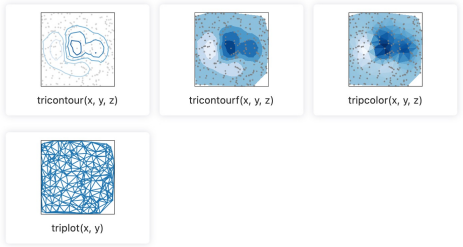
Basic

Basic plot types, usually y versus x.



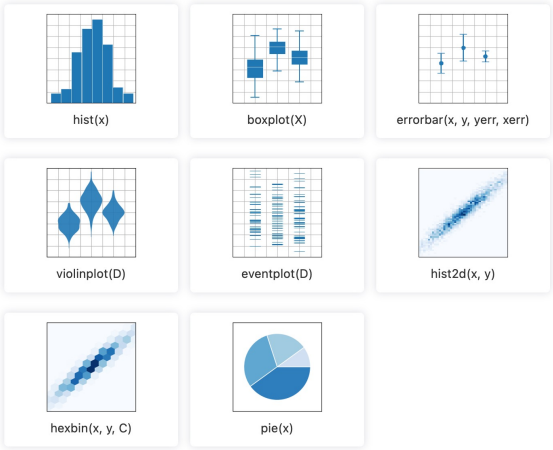
Unstructured coordinates

Sometimes we collect data `z` at coordinates `(x,y)` and want to visualize as a contour. Instead of gridding the data and then using `contour`, we can use a triangulation algorithm and fill the triangles.



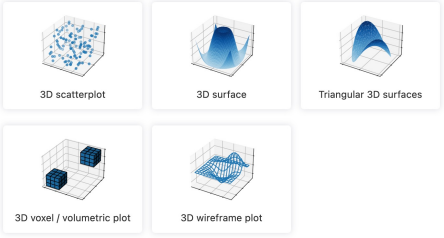
Statistics plots

Plots for statistical analysis.



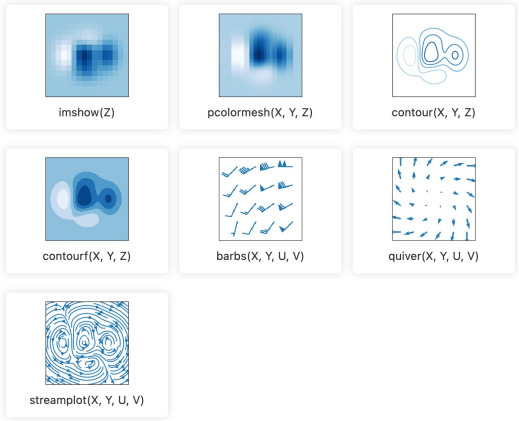
3D

3D plots using the `mpl_toolkits.mplot3d` library.



Plots of arrays and fields

Plotting for arrays of data `Z(x, y)` and fields `U(x, y), V(x, y)`.



２つの電荷のポテンシャル

たったこれだけで描画できてしまう！！

```
plt.pcolormesh(x, y, z)
plt.show()
```

少し書く量を増やしてみましょう！
→少し凝った図ができあがります

```
plt.contour(x, y, z, levels=[-2,-1,-0.5,-0.1,0,0.1,0.5,1,2], colors='k',
linewidths=1)
plt.pcolormesh(x, y, z, cmap='coolwarm')
plt.colorbar()
plt.clim(-3,3)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Potential of two charges')
plt.show()
```

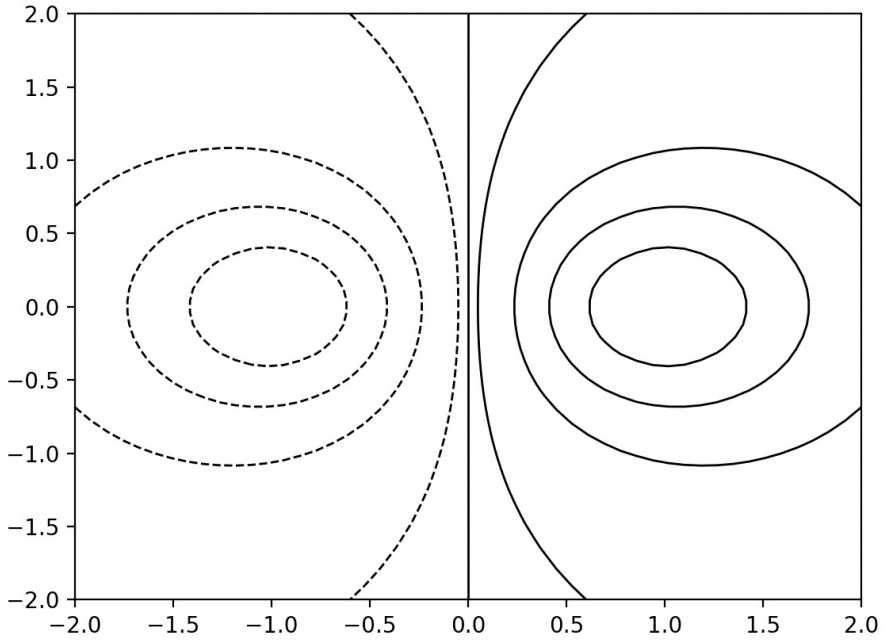
2つの電荷のポテンシャル

等高線を描画する関数

`levels` 引数で等高線の間隔を指定できます。 整数を指定した場合は、描画範囲を levels 段階に分けるように等高線を作成します。

```
plt.contour(x, y, z, levels=[-2,-1,-0.5,-0.1,0,0.1,0.5,1,2],
colors='k', linewidths=1)
```

リストを指定した場合、内側から順に指定したものが使用される。それを繰り返します.

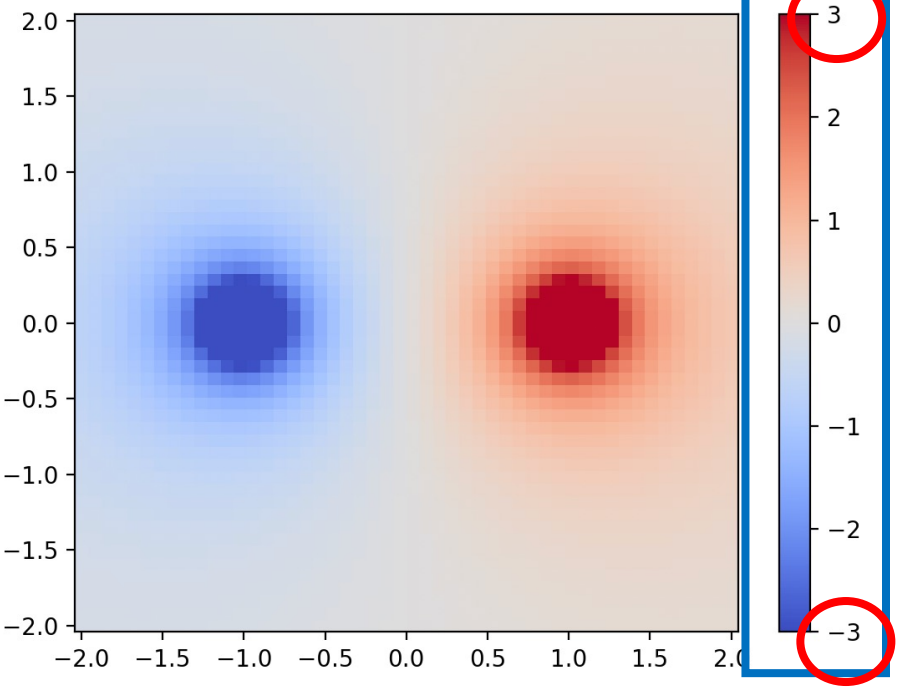


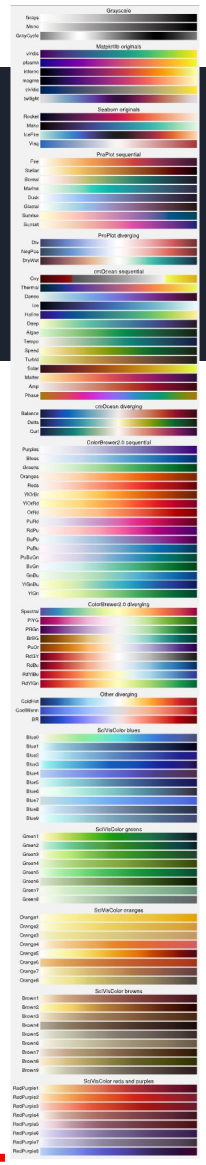
- ✓ `colors='k'`のkはblack
- ✓ `colors`ではなく`cmap`でも指定することができる
- ✓ 他には`linestyles=["-", "--", "-", "--", "-"]`などもある

2つの電荷のポテンシャル


```
plt.pcolormesh(x, y, z, cmap='coolwarm')
plt.colorbar()
plt.clim(-3,3)
```

カラーバーの上限・加減を設定できる



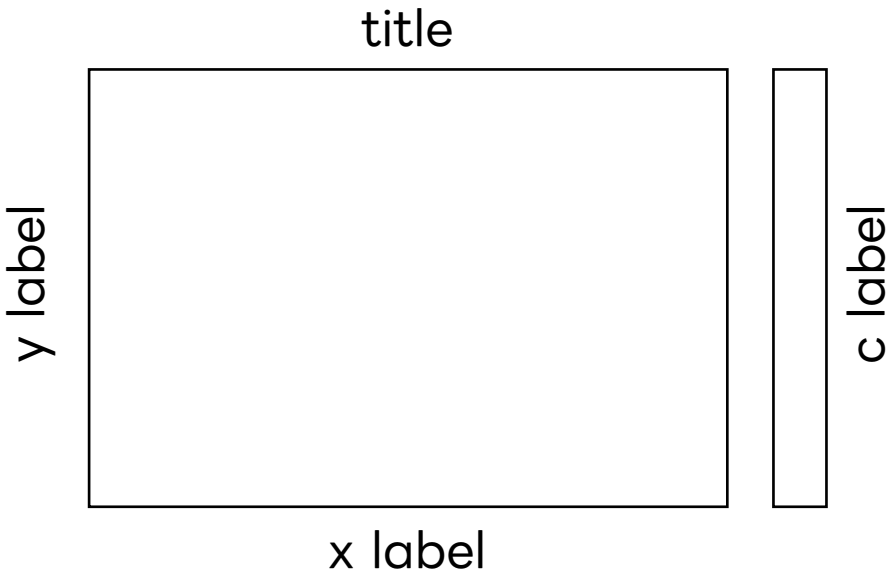


たくさんある！！
（自分でも作れる）

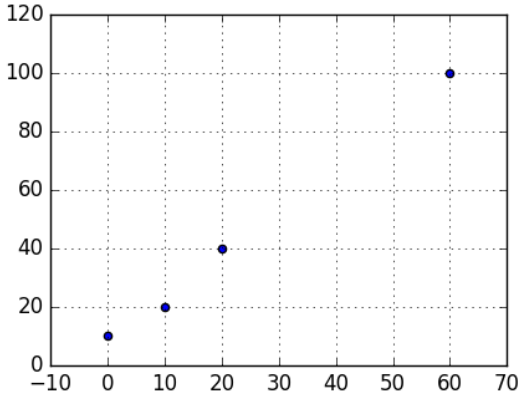
https://matplotlib.org/stable/tutorials/colors/colormaps.html

2つの電荷のポテンシャル


```
plt.xlabel('x')
plt.ylabel('y')
plt.title('Potential of two charges')
```

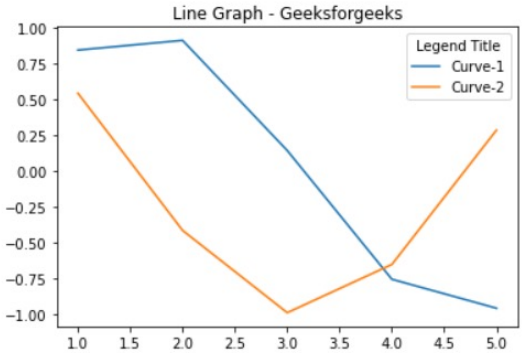



```
plt.clabel('c')
plt.grid()
plt.legend()
```



grid

https://stackoverflow.com/questions/8209568/how-do-i-draw-a-grid-onto-a-plot-in-python



legend

https://www.geeksforgeeks.org/how-to-add-a-title-to-a-matplotlib-legend/

そもそもフーリエ解析とは？

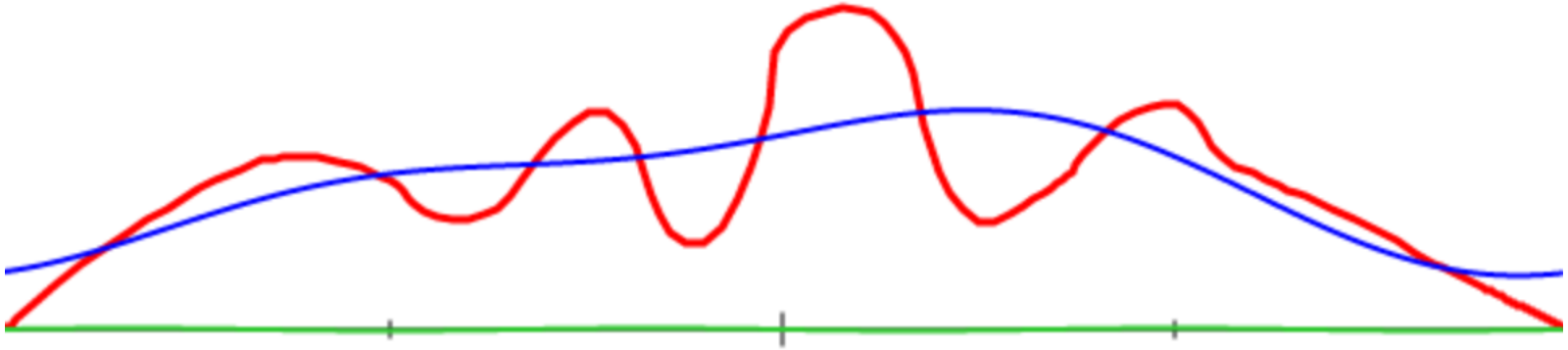
- 不連続点を除けば，周期 2π の任意の周期関数はフーリエ級数展開できる：

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx)$$

- 非周期関数を周期 $T \to \infty$ の周期関数と考えて任意の曲線がどんな周期の正弦波からなっているかを解析する方法を**フーリエ解析**という．

そもそもフーリエ解析とは？

5項

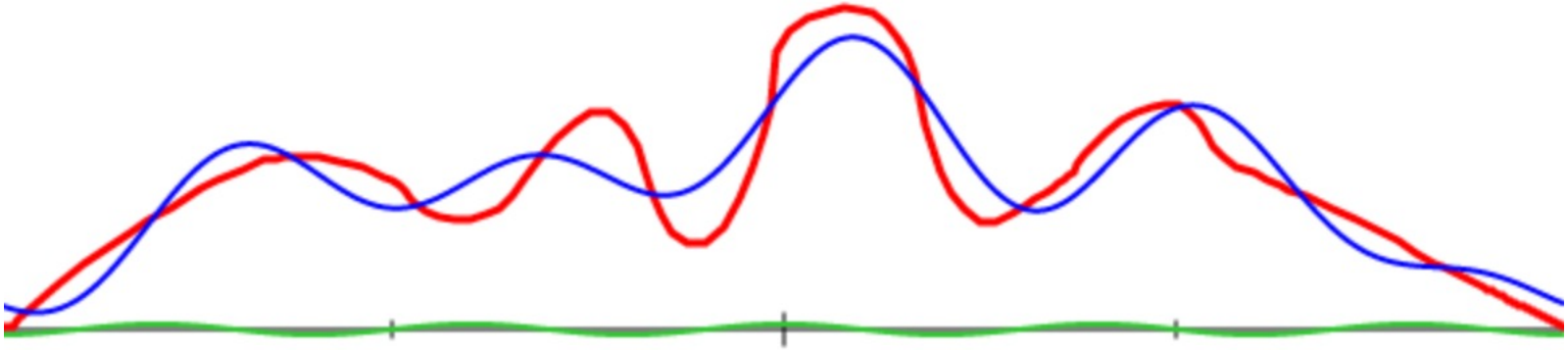


赤線…適当に描いた曲線

青線…正弦波のみでFittingした曲線

そもそもフーリエ解析とは？

10項

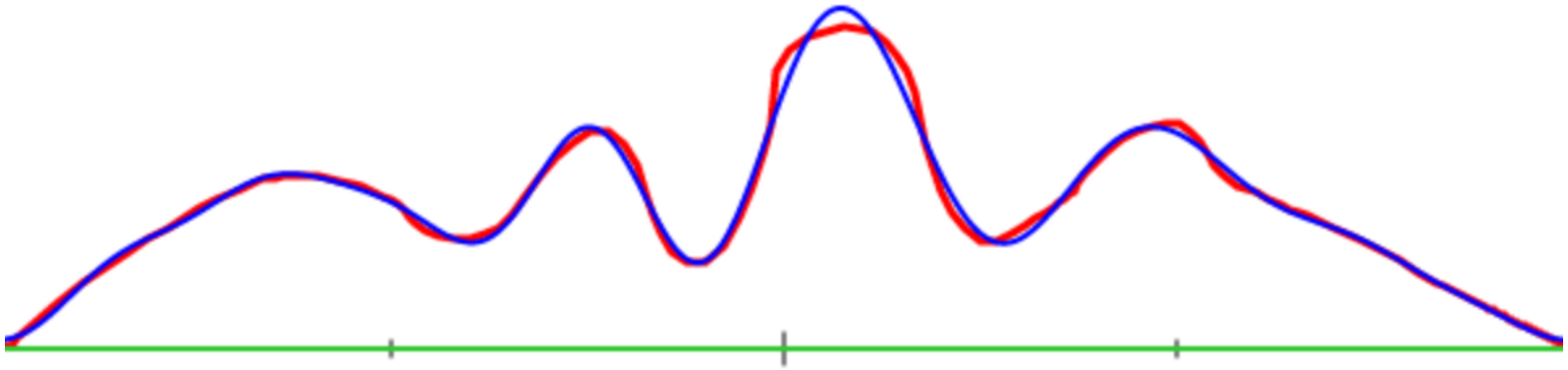


赤線…適当に描いた曲線

青線…正弦波のみでFittingした曲線

そもそもフーリエ解析とは？

20項

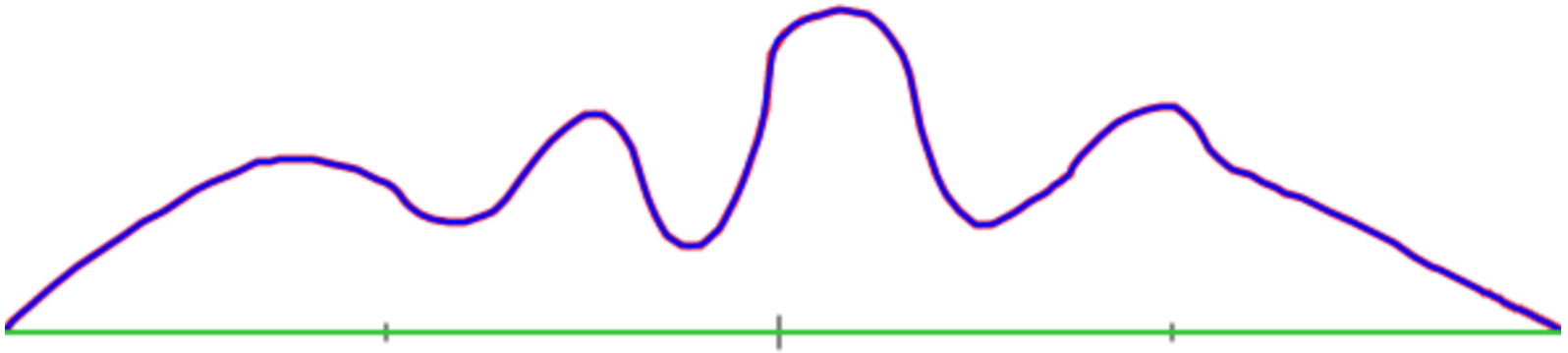


赤線…適当に描いた曲線

青線…正弦波のみでFittingした曲線

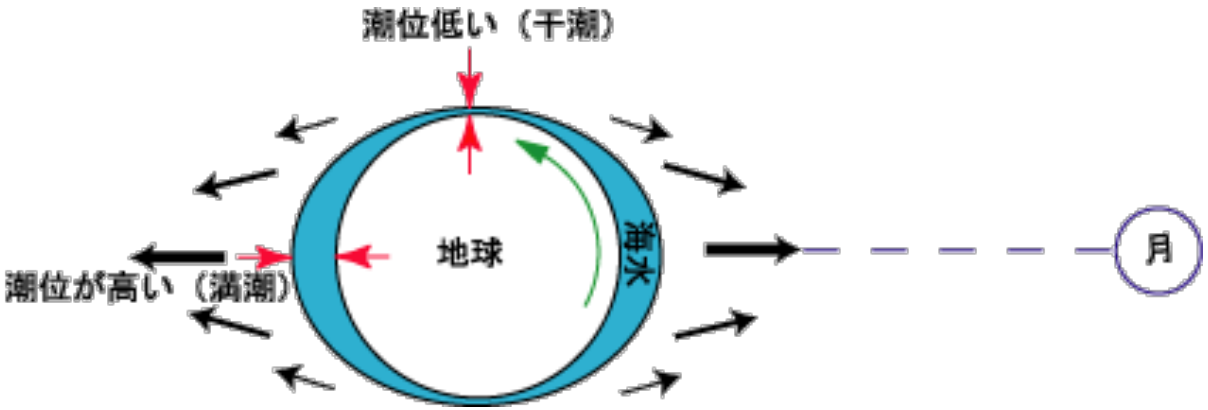
そもそもフーリエ解析とは？

180項



- 任意の曲線が正弦波のみで表せるイメージは掴めたかな？
  - →逆に任意の曲線がどんな周期の正弦波から成っているか解析するのを**フーリエ解析**

月は地球の空気を動かしている？



https://www1.kaiho.mlit.go.jp/KAN8/sv/teach/kaisyo/tide1.html

by Satoshi Shimada

→ 空気（気圧）にも影響を及ぼしている？

→ 周期性が存在するはず！

→ フーリエ解析

Pandasをインポートする


```
import pandas as pd
```

Pandas(パンダス)とは、データ解析を容易にする機能を提供するPythonのデータ解析ライブラリです。
Pandasの特徴には、データフレーム(DataFrame)などの独自のデータ構造が提供されており、様々な処理が可能です。

引用：https://aiacademy.jp/media/?p=152

- 事前に私がデータ整形した気象庁の2010年から2020年のデータを使用します.
- CSV形式，またはParquet形式を用意しました.
  - 自由な方をお使いください
  - Parquetの場合はpyarrowのインストールを忘れずに

Apache Parquetは、データの保存と検索を効率的に行うために設計された、オープンソースの列指向データファイルフォーマットです。**複雑なデータを大量に扱うために**、効率的なデータ圧縮とエンコーディングのスキームを提供し、性能を向上させています。Parquetは、Java、C++、Pythonなど、複数の言語で利用可能です。

データを読み込む

サーバー上にデータを上げているので好きな方をお使いください.

```
csv_url = 'https://ddd3h.github.io/data/tokyo_pres.csv'
parquet_url = 'https://ddd3h.github.io/data/tokyo_pres.parquet'

df = pd.read_parquet(parquet_url)

df.head()
```

今回は使わない

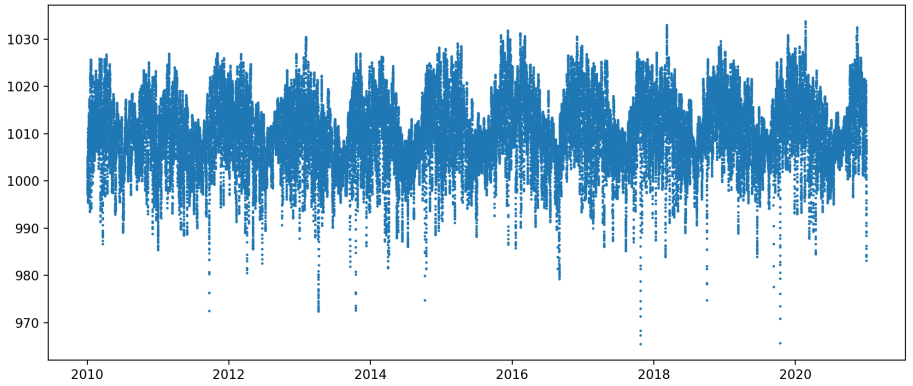
```
In [20]: df
Out[20]:
                Date&Time  LocalPressure[hPa]  QualityInfrom  HomogeneousNumber
0        2010/1/1 1:00:00               998.3              8                  1
1        2010/1/1 2:00:00               998.6              8                  1
2        2010/1/1 3:00:00               998.3              8                  1
3        2010/1/1 4:00:00               998.2              8                  1
4        2010/1/1 5:00:00               998.5              8                  1
...                   ...                 ...            ...                ...
8779  2020/12/31 20:00:00              1008.4              8                  1
8780  2020/12/31 21:00:00              1008.5              8                  1
8781  2020/12/31 22:00:00              1008.8              8                  1
8782  2020/12/31 23:00:00              1008.4              8                  1
8783    2021/1/1 00:00:00              1008.5              8                  1

[96432 rows x 4 columns]
```

→1時間ごとの気圧データが格納されている

データの確認


```
time = pd.to_datetime(df['Date&Time'])
pres = df['LocalPressure[hPa]']

fig, ax = plt.subplots(figsize=(12,5))
ax.scatter(time, pres,s=1)
plt.show()
```

- ✓ ParquetでもDate&Timeのデータは文字列として格納してあるので，時間データに変換してください
- ✓ 時間データ，圧力データは使いやすいようにtime，presという変数に入れとく
- ✓ どんなデータなのかをグラフにして確認してみよう．
- ✓ 欠損値が存在するか確認しよう．



欠損値の処理

✓ データに欠損が存在するか確認しよう

```
df.isna().any()
```

```
Date&Time              False
LocalPressure[hPa]      True
QualityInfrom          False
HomogeneousNumber      False
dtype: bool
```

デフォルトなので設定しなくても良い

✓ 欠損データを補完する（今回は線形補間を使用する）

```
df = df.interpolate(method='linear')
```

引数methodに指定できる補間方法としては、そのほか、'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'piecewise_polynomial', 'from_derivatives', 'pchip', 'akima'がある。

FFT

- 今回使用するのは

FFT = 高速フーリエ変換

※　より高機能のFFTを使用したい場合はScipyのほうを使用すると良い

numpy.fft.fft

`fft.fft(a, n=None, axis=-1, norm=None)` [source]

Compute the one-dimensional discrete Fourier Transform.

This function computes the one-dimensional *n*-point discrete Fourier Transform (DFT) with the efficient Fast Fourier Transform (FFT) algorithm [CT].

By Documentation (https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html)

Discrete Fourier Transform (`numpy.fft`)

- numpy.fft.fft
- numpy.fft.ifft
- numpy.fft.fft2
- numpy.fft.ifft2
- numpy.fft.fftn
- numpy.fft.ifftn
- numpy.fft.rfft
- numpy.fft.irfft
- numpy.fft.rfft2
- numpy.fft.irfft2
- numpy.fft.rfftn
- numpy.fft.irfftn
- numpy.fft.hfft
- numpy.fft.ihfft
- numpy.fft.fftfreq
- numpy.fft.rfftfreq
- numpy.fft.fftshift
- numpy.fft.ifftshift

See also

`numpy.fft`
for definition of the DFT and conventions used.

`ifft`
The inverse of `fft`. →fftの逆関数.

`fft2`
The two-dimensional FFT. →2次元FFT.

`fftn`
The *n*-dimensional FFT. →n次元FFT.

`rfftn`
The *n*-dimensional FFT of real input. →実入力のn次元FFT.

`fftfreq`
Frequency bins for given FFT parameters. → 与えられたFFTパラメータに対応する周波数ビン.

フーリエ変換


```
N = len(pres)
```

↓高速フーリエ変換

```
fk = np.fft.fft(pres)
freq = np.fft.fftfreq(N,d=1)
```

サンプリング周期[h]→サンプリング周波数[1/h]

```
plt.plot(1/freq, np.abs(fk)/(N/2),lw=1)
```

周波数[1/h]→軸を時間[h]

正規化係数・1 周期がデータ数 N/2 に対応している

```
plt.xlim(2,15)
plt.ylim(-0.1,0.8)
plt.show()
```

結果は複素数なので振幅スペクトルを計算する場合はその絶対値

📖 分からない箇所があれば離散フーリエ変換等を参照

python

```
1 numpy.fft.fftfreq(n, d=1.0)
```

引数の説明は以下の通り。

`n` : FFTを行うデータ点数。

`d` : サンプリング周期（デフォルト値は `1.0` ）。