高校生のデータサイエンス・
Python でも 77 本ノック

NAIST STELLA プログラム

「共創」が育む主体性の未来　学習教材

付録に「Google Colaboratory の使い方」も収録！

金谷重彦 編著

国立大学法人
奈良先端科学技術大学院大学
NARA INSTITUTE of SCIENCE and TECHNOLOGY

ISBN 978-4-902874-06-8

高校生のデータサイエンス・77 本ノック

NAIST STELLA プログラム

「共創」が育む主体性の未来 学習教材

付録に「Google Colaboratory の使い方」も収録！

金谷重彦 編著

奈良先端科学技術大学院大学・データ駆動型サイエンス創造センター

先端科学技術研究科・情報科学領域

ISBN 978-4-902874-06-8

目次

はじめに

「高校生のデータサイエンス・77 本ノック」は、高校生の方々に「いきなり大学院」（高院連携）、すなわち、ちょっと研究にむけたプログラミングを通した知識発見を実現しようと作成しました。

ここで使っているプログラミング言語は Python 言語で、コードが読みやすく、世界中で使われている言語です。それと、様々なパッケージがあるので、統計解析や機械学習だけでなく、ウェブやゲームの開発にも用いられています。

皆さんは、Python 言語との初めての出会いだと思います。巻末には付録として、Google Colaboratory の使い方を収録しました。Google Colaboratory を使うとインストールの手間なく、プログラミングが行えるので、これを使いこなしましょう。

その上で、「I.プログラミング基礎」と「II.データ整理の基礎」を実習しましょう。その後は、「III 知識発見」のテーマ、「3.妖怪」、「4.関係性とは」、「5.確率・統計」、「6.集合」、「7.フィボナッチ数」、「8.素数」、「9.無限級数の和」のどこを読んでもすぐに実習できるようになっています。まあ、興味のありそうなところから読んでください。まあ、内容としては大学院のものもありますが、驚きながら楽しみましょう。

「おっと、面白じゃないか！」
と思っていただきたく、よろしくね！

I. プログラミング基礎

1. Python プログラミング基礎 [情報 1]

本テキストで使うプログラム文法を理解しましょう。実際に実行して出力結果により動作原理を解読してください。

変数の定義と代入

変数 a に数値 1 を代入するには、a=1 とします。[]でくくることで、値が並んだリストに変換することができます。この場合、Python 言語では、a とリストの 0 番目の値を返す [a][0] は同じです。Python では 0 から数えるため、0 番目は普段で言う 1 番目と同じことに気をつけてください。では、このことを確認しましょう。

ノック 1 本目

RK01.py のプログラムを実行し、a と [a][0] が同じことを確認しましょう。

RK01.py

```
1  # 変数 a に 1 を代入する
2  a = 1
3
4  # リスト[a]の長さを取得する
5  # Python では単一の整数の長さを取得する方法はないので、リストに変換して長さを
   取得します
6  length_of_a = len([a])
7
8  # リスト[a]の最初の要素を取得する
9  # Python では単一の整数はインデックスを持たないので、リストに変換して最初の要
   素を取得します
10 first_element = [a][0]
11
12 # 変数 a を表示する
13 print("変数 a:", a)
14
15 # 結果を表示する
16 print("リスト[a]の長さ:", length_of_a)
```

```
17 print("リスト[a]の最初の要素:", first_element)
```

四則演算など

足し算しよう！数値の足し算をする場合、Python でも「+」により足し算ができます。

ノック２本目

RK02.py を実行し、a + b の結果を代入した c が、足し算の結果と同じになることを確認しよう。

RK02.py

```
1  # 変数 a に 5 を代入する
2  a = 5
3
4  # 変数 b に 2 を代入する
5  b = 2
6
7  # 変数 a と変数 b の和を計算し、変数 c に代入する
8  c = a + b
9
10 # 結果を表示する
11 print("a + b =", a + b)
12 print("変数 c の値:", c)
```

演算を楽しもう！

ノック３本目

以下の表は、演算記号を説明しています。この表を参考に RK03.py を実行して結果を考察しよう。Python では階乗を計算する際には math というパッケージを用います。

記号	演算
a + b	足し算
a - b	引き算
a * b	掛け算
a / b	割り算
a // b	割り算（商を整数で出力）
a % b	割り算の余り

a ** b	a^b
math.factorial(a)	$a(a-1)(a-2)\cdots 3\cdot 2\cdot 1$ (a の階乗)

RK03.py

```python
import math  # 数学関数のためのライブラリをインポート

# 変数 a に 13 を代入する
a = 13

# 変数 b に 3 を代入する
b = 3

# 変数 a と変数 b の和を計算する
sum_ab = a + b

# 変数 a と変数 b の差を計算する
diff_ab = a - b

# 変数 a と変数 b の積を計算する
prod_ab = a * b

# 変数 a と変数 b の商を計算する(小数)
div_ab = a / b

# 変数 a と変数 b の商を計算する(整数)
int_div_ab = a // b

# 変数 a と変数 b の剰余を計算する
mod_ab = a % b

# 変数 a の b 乗を計算する
power_ab = a ** b

# math パッケージを用いて、5 の階乗を計算する
factorial_5 = math.factorial(5)

```

```
# 結果を表示する
print("a + b =", sum_ab)
print("a - b =", diff_ab)
print("a * b =", prod_ab)
print("a / b =", div_ab)
print("a // b =", int_div_ab)
print("a % b =", mod_ab)
print("a ^ b =", power_ab)
print("factorial(5) =", factorial_5)
```

ベクトルの要素ごとの演算

ベクトルとは、a=(1, 2, 3)、b=(4, 5, 6) のように複数の数値からなる列です。これらの数値をそれぞれベクトルの要素とします。そこで要素ごとに演算についてもノック 3 本目で示した演算記号を活用できます。例えば、

a + b = (1, 2, 3) + (4, 5, 6) = (5, 7, 9)

となります。

ノック 4 本目

ノック 3 本目で示した演算記号をベクトルの要素の演算に適用してみよう。RK04.py の実行結果を考察しよう。

RK04.py

```
# ベクトルや行列演算のための numpy パッケージをインポートします
import numpy as np

# 変数 a に配列[10, 20, 30]を代入する
a = np.array([10, 20, 30])

# 変数 b に配列[3, 2, 1]を代入する
b = np.array([3, 2, 1])

# 変数 a の長さを取得する
length_a = len(a)
```

```
12
13  # 変数 b の長さを取得する
14  length_b = len(b)
15
16  # 変数 a と変数 b の和を計算する
17  sum_ab = a + b
18
19  # 変数 a と変数 b の差を計算する
20  diff_ab = a - b
21
22  # 変数 a と変数 b の積を計算する
23  prod_ab = a * b
24
25  # 変数 a と変数 b の商を計算する(小数)
26  div_ab = a / b
27
28  # 変数 a と変数 b の商を計算する(整数)
29  int_div_ab = a // b
30
31  # 変数 a と変数 b の剰余を計算する
32  mod_ab = a % b
33
34  # 変数 a の b 乗を計算する
35  power_ab = a ** b
36
37  # 結果を表示する
38  print("変数 a の長さ:", length_a)
39  print("変数 b の長さ:", length_b)
40  print("a + b =", sum_ab)
41  print("a - b =", diff_ab)
42  print("a * b =", prod_ab)
43  print("a / b =", div_ab)
44  print("a // b =", int_div_ab)
45  print("a % b =", mod_ab)
46  print("a ^ b =", power_ab)
```

条件分岐：if 条件: ~, else: ~

中央値を求めるプログラムを作ろう！

1.1、12.3、8.5、4.2、2.1、9.4、7.0、3.0

と 8 つの数値列が与えられたときに、この中央値を求めるにはどうしたらいいだろうか？

まず、数字を小さい順に並べます。

1.1、2.1、3.0、4.2、7.0、8.5、9.4、12.3

要素の数は 8 個なので、4 番目と 5 番目の平均値（ = (4.2 + 7.0) / 2 = 5.6 ）が中央値となります。

では

1.1、12.3、8.5、4.2、2.1、9.4、7.0

のときにはというと、

1.1、2.1、4.2、7.0、8.5、9.4、12.3

と並べ替えて、要素数が 7 個であるので、4 番目（7.0）が中央値となります。ということは、要素数が偶数か奇数かにより、中央値の求め方が異なります。これを if: ~, else:~で場合分けをすれば OK です。

ノック 5 本目

{1.1, 12.3, 8.5, 4.2, 2.1, 9.4, 7.0, 3.0} の中央値を求めるプログラムを作成しよう。

RK05.py

```
1  # 必要なパッケージをインポートします
2  import numpy as np
3
4  # 変数 x に配列[1.1, 12.3, 8.5, 4.2, 2.1, 9.4, 7.0, 3.0]を代入する
5  x = np.array([1.1, 12.3, 8.5, 4.2, 2.1, 9.4, 7.0, 3.0])
6
7  # 配列 x をソートする
8  sx = np.sort(x)
9
10 # ソートされた配列の長さを取得する
11 ns = len(sx)
12
13 # 中央値を計算する
14 if ns % 2 == 0:
```

```
    medv = (sx[ns // 2 - 1] + sx[ns // 2]) / 2
else:
    medv = sx[ns // 2]

# 結果を表示する
print("手動計算による中央値:", medv)

# numpy を使用して中央値を計算する
median_value = np.median(x)

# 結果を表示する
print("numpy による中央値:", median_value)
```

ループ for i in ~

a[0] = 1, a[1] = 2, a[2] = 3, a[3] = 4 とします。a[0] * a[1] * a[2] * a[3]を計算しましょう。

初期値として mp = 1 としましょう。

i の範囲は 1, 2, 3, 4 とします。

i = 0 のとき、

a[0] をもとに、mp * a[0] を計算してこれをまた mp という変数に代入します。

すると mp = 1 となります。

i = 1 のとき、

a[1] をもとに、mp * a[1] を計算してこれをまた mp という変数に代入します。

すると mp = 1 * 2 = 2 となります。

i = 2 のとき、

a[2] をもとに、mp * a[2] を計算してこれをまた mp という変数に代入します。

すると mp = 2 * 3 = 6 となります。

i = 3 のとき、

a[3] をもとに、mp * a[3] を計算してこれをまた mp という変数に代入します。

すると mp = 6 * 4 = 24 となり、mp = 24 (= a[0] * a[1] * a[2] * a[3])となりました。

このように、n 個の値が a[0], a[1], …, a[n-1]と n 個の数値が格納されているとき、

for i in range(n): ~

とし、i は 0, 1, …, n-1 として for i in range(n): ~ の ~ の中で演算をすれば、繰り返し演算ができます。

```
for i in range(14):
    mp *= a[i]
```

ノック 6 本目

RK06.py を実行し動作を理解しよう。

繰り返し演算(ループ処理)を活用し、1 x 2 x 3 x … x 14 を計算しよう。

RK06.py

```
1  # 必要なパッケージをインポートします
```

```
2  import math
3
4  # 変数 a にリスト[1, 2, ..., 14]を代入する
5  a = list(range(1, 15))
6
7  # 変数 mp を 1 に初期化する
8  mp = 1
9
10 # 0 から 13 までの各要素についてループを実行する
11 for i in range(14):
12     mp *= a[i]
13     print(f"round[{i+1}]={mp}")
14
15 # 結果を表示する
16 print("最終的な mp の値:", mp)
17
18 # 1*2*3*...*14 の結果を表示する
19 print("1*2*3*...*14 =", mp)
20
21 # math モジュールを使用して 14 の階乗を計算する
22 factorial_14 = math.factorial(14)
23
24 # 結果を表示する
25 print("factorial(14) =", factorial_14)
```

5 行目：a[0] = 1, a[1] = 2, …, a[13] = 14 とベクトルを作る。

8 行目：掛け算の結果を mp とし 1 を代入し初期化する。

11-13 行目：for ループにより、i = 0, …, 13 として、1-14 の掛け算を行う。

i = 0 について、mp x a[1] = 1 を mp = 1 とする。

i = 1 について、mp x 2 (=2) を計算し、これを mp に代入する。すなわち、mp = 2。

i = 2 について、mp x 3 (=6) を計算し、これを mp に代入する。

…

i = 13 について　mp x 14 (=8717829120) を計算し、これを mp に代入し、おしまい。

出力結果を以下に示す。

```
round[1]=1
round[2]=2
round[3]=6
```

```
round[4]=24
round[5]=120
round[6]=720
round[7]=5040
round[8]=40320
round[9]=362880
round[10]=3628800
round[11]=39916800
round[12]=479001600
round[13]=6227020800
round[14]=87178291200
最終的な mp の値： 87178291200
1*2*3*...*14 = 87178291200
factorial(14) = 87178291200
```

ノック 7 本目

1 x 2 x 3 x … x 14 を計算しよう。a[0] = 1, a[1] = 2, …, a[13] = 14 とベクトルを作らなくても 14! を計算できます。どうやればいいのでしょうか？for ループを活用して実現しよう。

RK07.py

```
1  # 変数 mp を 1 に初期化する
2  mp = 1
3
4  # 1 から 14 までの各要素についてループを実行する
5  # range(a, b)と書くと i=a,...,b-1 までとなる
6  for i in range(1, 15):
7      mp *= i
8
9  # 結果を表示する
   print("最終的な mp の値:", mp)
```

ノック 8 本目

1, 2, 3, 5, 8, 13 を [1, 2, 3, 5, 8, 13] として、これらの総和を求めるプログラムを、for ループを活用して作成しよう。

RK08.py

```
1  # 変数 ss を 0 に初期化する
2  ss = 0
3
4  # リスト[1, 2, 3, 5, 8, 13]の各要素についてループを実行する
5  for i in [1, 2, 3, 5, 8, 13]:
6      ss += i
7      print(f"SUM {i} = {ss}")
```

出力結果

```
SUM 1 = 1
SUM 2 = 3
SUM 3 = 6
SUM 5 = 11
SUM 8 = 19
SUM 13 = 32
```

ループ処理: while 条件: ~

1 x 2 x 3…x k について、1,000,000,000 より小さく、最大となる値と、そのときの k を求めてみよう。

while 条件: について条件が成り立つ間、while の内部の演算を行う。ここで注意すべきは

while mp < 1000000000:

について、i = 12 のとき

mp (= 479001600) < 1000000000

であるので、条件を満たし、

i = 13 のとき mp (= 6227020800) となり、

mp < 1000000000

を満たさず、while ループから脱出する。

そこで、求めたいのは、i = 12 のときの mp となる。つまり while ループを抜けたときの値の一個前をどうやったら得られるか？これを考えてみよう。

ノック 9 本目

1 x 2 x 3 x … x k について、1,000,000,000 より小さく、最大となる値と、そのときの k を求めてみよう。

RK09.py

```
1  # 変数 mp を 1 に初期化する
2  mp = 1
3
4  # 変数 i を 1 に初期化する
5  i = 1
6
7  # mp が 1000000000 未満の間ループを実行する
8  while mp < 1000000000:
9      mp *= i
10     print(f"round=[{i}] {mp}")
11     i += 1
12
13 # 結果を計算して表示する
14 result = mp / (i - 1)
```

```
print("mp / (i - 1) =", result)
print("k =", i - 2)
print(f"round {i - 2} : {result}")
```

出力結果

```
round=[1] 1
round=[2] 2
round=[3] 6
round=[4] 24
round=[5] 120
round=[6] 720
round=[7] 5040
round=[8] 40320
round=[9] 362880
round=[10] 3628800
round=[11] 39916800
round=[12] 479001600
round=[13] 6227020800
mp / (i - 1) = 479001600.0
k = 12
round 12 : 479001600.0
```

ノック 10 本目

1 x 2 x 3 x … x k について、1,000,000,000 より小さく、最大となる値と、そのときの k を求めてみよう。ノック９本目の改良版です。

RK10.py

```
# 変数 mp を 1 に初期化する
mp = 1

# 変数 i を 2 に初期化する
i = 2

# mp が 1000000000 未満の間ループを実行する
while mp < 1000000000:
    # 現在の mp の値を保存する
```

```
    mprev = mp
    # mp を i で掛ける
    mp *= i
    # i を 1 増やす
    i += 1

    # mp が 1000000000 を超えた場合
    if mp > 1000000000:
        # mp を前の値に戻す
        mp = mprev
        # i を 2 減らす
        i -= 2
        # ループを抜ける
        break

# 結果を表示する
print("最終的な mp の値:", mp)
print(f"round {i} : {mp}")
```

簡単な行列処理

データセットに対しての演算は DataFrame.apply(func, axis) を使えばできる。axis=0 で列ごとに、axis=1 で行ごとに適用する。例えば、dataSet.apply(lambda col: col.mean(), axis=0) と書けば、列ごとの平均を計算できる。これを用いて、RK11.py を書いてみる。しかしながら、Python で平均や分散を計算したい場合は、DataFrame.mean(axis=0)、DataFrame.var(axis=0) で書くのが実用的です。

ノック 11 本目

行が 6 個 (r_1, r_2, r_3, r_4, r_5) 、列が 3 個 (x, y, z) のデータがあるとしよう。

	x	y	z
r_1	1	2	3
r_2	1	3	5
r_3	1	4	7
r_4	1	5	9
r_5	1	6	11

このようなデータについて、行ごとの集計(演算)は apply(X, 1, F) 、列ごとの集計(演算)は apply(X, 2, F) を使えば簡単にできる。これを試してみよう。

RK11.py

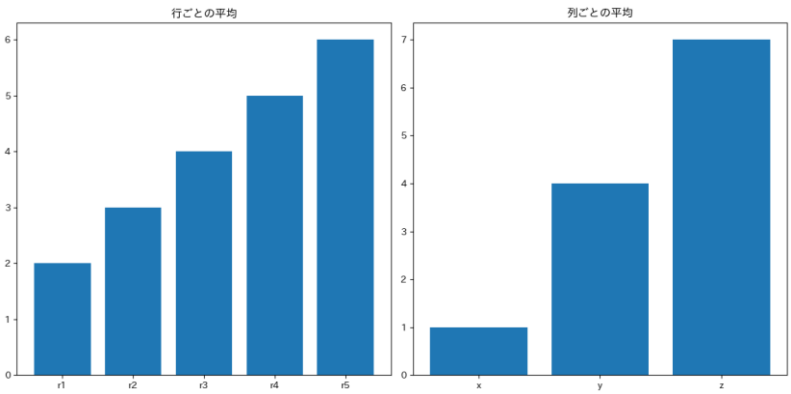
```
1  import japanize_matplotlib  # 日本語フォントを表示するためのライブラリをインポート
2  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
3  import pandas as pd  # データフレーム操作のためのライブラリをインポート
4
5  # x, y, z のデータを作成する
6  x = [1, 1, 1, 1, 1]
7  y = [2, 3, 4, 5, 6]
8  z = [3, 5, 7, 9, 11]
9
10 # データフレームを作成する
11 dataSet = pd.DataFrame({'x': x, 'y': y, 'z': z})
12 dataSet.index = ["r1", "r2", "r3", "r4", "r5"]
13
14 # データフレームを表示する
```

```
15 print(dataSet)
16
17 # プロットを 2 つ並べて表示する設定
18 fig, axes = plt.subplots(1, 2, figsize=(12, 6))
19
20 # 行ごとの平均を計算して棒グラフを作成する
21 rmean = dataSet.mean(axis=1)
22 axes[0].bar(rmean.index, rmean)
23 axes[0].set_title("行ごとの平均")
24
25 # 列ごとの平均を計算して棒グラフを作成する
26 cmean = dataSet.mean(axis=0)
27 axes[1].bar(cmean.index, cmean)
28 axes[1].set_title("列ごとの平均")
29
30 # グラフを表示する
31 plt.tight_layout()
32 plt.show()
```

15 行目：dataSet は

```
> print(dataset)
    x  y  z
r1  1  2  3
r2  1  3  5
r3  1  4  7
r4  1  5  9
r5  1  6 11
```

と定義されている。



18 行目：図を横に 2 枚並べて配置する。

21-23 行目：列ごと（x, y, z）の平均値を求めて、棒グラフに表す（左）。

26-28 行目：行ごと（r_1, r_2, r_3, r_4, r_5）の平均値を求めて、棒グラフに表す（右）。

論理演算 [数学 I]

論理演算とは、真と偽に関する演算です。「サイコロは六面体だ」真としよう。一方、「サイコロには 6 の目がある」も真であるとしよう。さらに、「サイコロは五面体だ」を偽としよう。

このとき、真を 1、偽を 0 とする。

『「サイコロは六面体だ」または「サイコロは五面体だ」』は「真」1 + 0=1

『「サイコロは六面体だ」かつ「サイコロには 6 の目がある」は「真」1 x 1 =1

『「サイコロは六面体だ」かつ「サイコロは五面体だ」』は「偽」1 x 0=0

と「かつ」は掛け算、「または」は足し算として真偽を 1,0 により判定できる演算で、ブール代数ともいいます。

ノック 12 本目

{1, 5, 6, 9, 8, 2, 4} それぞれについて、

(1) 2 で割った余りが 0 であることが真(1)か偽(0)か判定するプログラムを作成しよう。

(2) 3 で割った余りが 0 であることが真(1)か偽(0)か判定するプログラムを作成しよう。

(3) (1)を満たす要素を抜き出すプログラムを作成しよう。

(4) (2)を満たす要素を抜き出すプログラムを作成しよう。

(5) (1)かつ(2)を満たす要素を抜き出すプログラムを作成しよう。

(6) (1)または(2)を満たす要素を抜き出すプログラムを作成しよう。

RK12.py

```
1  import numpy as np  # 数値計算のためのライブラリをインポート
2
3  # 配列 X を作成する
4  X = np.array([1, 5, 6, 9, 8, 2, 4])
5
6  # TF1 は X の要素が 2 で割り切れるかどうかを判定するブール配列
7  TF1 = (X % 2 == 0)
8
```

```
9  # TF2 は X の要素が 3 で割り切れるかどうかを判定するブール配列
10 TF2 = (X % 3 == 0)
11
12 # TF1 が True の要素を取り出す
13 X_TF1 = X[TF1]
14
15 # TF2 が True の要素を取り出す
16 X_TF2 = X[TF2]
17
18 # TF1 と TF2 が両方 True の要素を取り出す
19 AND = TF1 & TF2
20 X_AND = X[AND]
21
22 # TF1 または TF2 が True の要素を取り出す
23 OR = TF1 | TF2
24 X_OR = X[OR]
25
26 # 結果を表示する
27 print("X[TF1]:", X_TF1)
28 print("X[TF2]:", X_TF2)
29 print("X[AND]:", X_AND)
30 print("X[OR]:", X_OR)
```

	i	1	2	3	4	5	6	7
4 行目	X[i]	1	5	6	9	8	2	4
7 行目	TF1[i] (X%2==0)	False	False	True	False	True	True	True
10 行目	TF2[i] (X%3==0)	False	False	True	True	False	False	False
13 行目	X[TF1]			6		8	2	4
16 行目	X[TF2]			6	9			
19 行目	AND (TF1&TF2)	False	False	True	False	False	False	False
20 行目	X[AND] (残る要素)			6				
23 行目	OR (TF1|TF2)	False	False	True	True	True	True	True
24 行目	X[OR] (残る要素)			6	9	8	2	4

7 行目：「X % 2 == 0」X[i] (i=1, …, 7) を 2 で割った余りが 0 であると True、0 でないと False となります。
13 行目：X[TF1] について、TF1 が True の要素のみを残す。その結果、X[TF1] の要素は、6, 8, 2, 4 となります。
16 行目：X[TF2] について、TF2 が True の要素のみを残す。その結果、X[TF2] の要素は、6,9 となります。
19 行目、**23 行目**: TF1 & TF2 とは、「TF1 が True」かつ「TF2 が True」という論理演算です。ここで、積(and)「&」演算と和(or)「|」を以下に示す。参考まで、~X は X の否定です。

X	True	True	False	False
Y	False	True	False	True

X&Y	False	True	False	False
X|Y	True	True	False	True
!X	False	False	True	True
!Y	True	False	True	False

論理演算と四則演算 [数学 I]

論理演算で得られる値には、真(True)と偽(False)があります。ここで「かつ」は「&」、「または」は「|」の記号を使います。つまり、True & True は True、True & False は False、False & False は False となります。また、True | True は True、True | False は True、False | False は False となります。Python 言語では、True は 1、False は 0 として扱われます。True + True は 1 + 1 = 2 となります。

ノック 13 本目

RK13.py を実行し True と False の演算「&、|、*、+、-、/、！」を理解しよう。

RK13.py

```
1  # 論理演算の結果を表示する
2
3  # AND 演算
4  print("True & True:", True & True)
5  print("True & False", True & False)
6  print("False & False:", False & False)
7
8  # OR 演算
9  print("True | True:", True | True)
10 print("True | False:", True | False)
11 print("False | False:", False | False)
12
13 # 論理値の乗算
14 print("True * True:", True * True)
15 print("True * False:", True * False)
16 print("False * False:", False * False)
17
18 # 論理値の加算
19 print("True + True:", True + True)
20 print("True + False:", True + False)
21 print("False + False:", False + False)
22
23 # 論理値の減算
```

```
print("True - True:", True - True)
print("True - False:", True - False)
print("False - False:", False - False)
print("False - True:", False - True)

# 論理値の除算
print("True / True:", True / True)
try:
    print("True / False:", True / False)
except ZeroDivisionError:
    print("True / False: ゼロ割りエラー")
try:
    print("False / False:", False / False)
except ZeroDivisionError:
    print("False / False: ゼロ割りエラー")
try:
    print("False / True:", False / True)
except ZeroDivisionError:
    print("False / True: ゼロ割りエラー")

# NOT 演算
print("!True:", not True)
```

論理演算「＊」「|」「！」では、True あるいは False を返す。一方、算術演算では、True は 1、False は 0 として算術演算を行い、数字を返す。

出力結果

```
True & True: True
True & False False
False & False: False
True | True: True
True | False: True
False | False: False
True * True: 1
True * False: 0
False * False: 0
True + True: 2
```

```
True + False: 1
False + False: 0
True - True: 0
True - False: 1
False - False: 0
False - True: -1
True / True: 1.0
True / False: ゼロ割りエラー
False / False: ゼロ割りエラー
False / True: 0.0
!True: False
```

II. データ整理の基礎

2. データの整理 [数学 1]

棒グラフ

Matplotlib の bar() を使うと縦棒、横棒も棒グラフを描画できる。

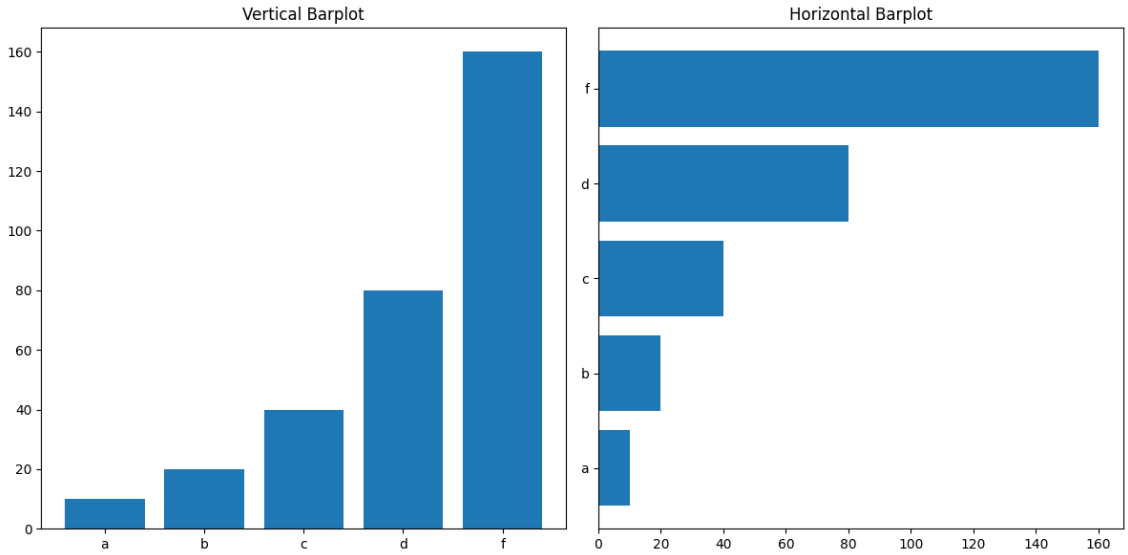
ノック 14 本目

a, b, c, d, f それぞれの値を縦、横の 2 種類の棒グラフを作成しよう。

name	a	b	c	d	f
xdata	10	20	40	80	160

RK14.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2
3  # データを作成する
4  xdata = [10, 20, 40, 80, 160]
5  labels = ["a", "b", "c", "d", "f"]
6
7  # プロットを 2 つ並べて表示する設定
8  fig, axes = plt.subplots(1, 2, figsize=(12, 6))
9
10 # 垂直の棒グラフを描く
11 axes[0].bar(labels, xdata)
12 axes[0].set_title("Vertical Barplot")
13
14 # 水平の棒グラフを描く
15 axes[1].barh(labels, xdata)
16 axes[1].set_title("Horizontal Barplot")
17
18 # グラフを表示する
19 plt.tight_layout()
20 plt.show()
```



bar() を使うと複数の列からなるデータを棒グラフ行あるいは列で重ねて作成できる。

ノック 15 本目

以下の行列 dm について

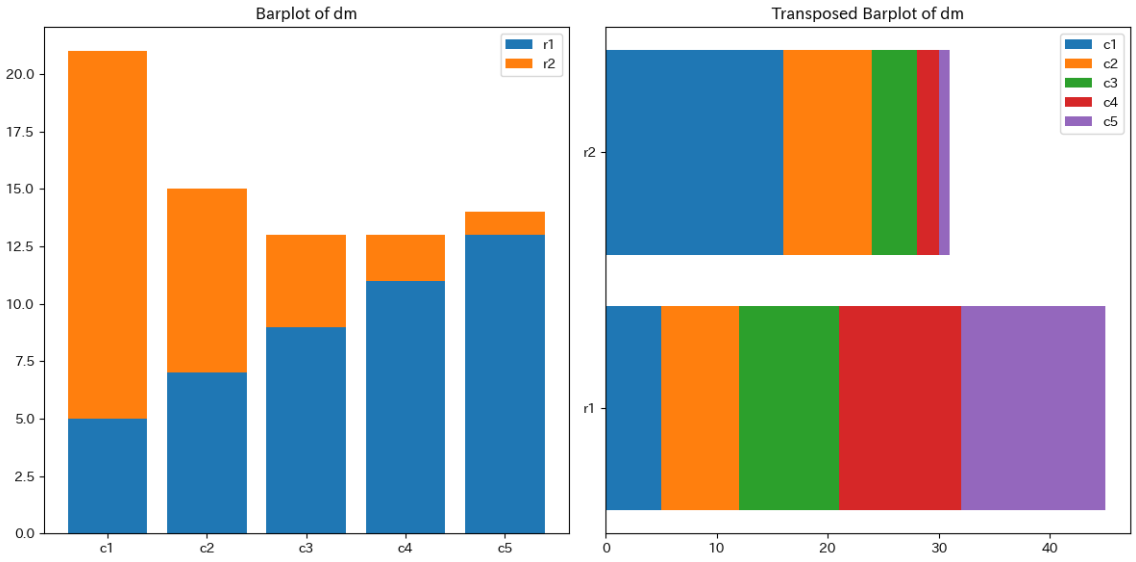
$$\mathrm{dm} = \begin{pmatrix} 5 & 7 & 9 & 11 & 13 \\ 16 & 8 & 4 & 2 & 1 \end{pmatrix}$$

行と列についてそれぞれの要素の棒グラフを重ねて描画しよう。

RK15.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # 行列 dm を作成する
5  dm = np.array([[5, 7, 9, 11, 13], [16, 8, 4, 2, 1]])
6  col_names = ["c1", "c2", "c3", "c4", "c5"]
7  row_names = ["r1", "r2"]
8
9  # プロットを 2 つ並べて表示する設定
10 fig, axes = plt.subplots(1, 2, figsize=(12, 6))
11
12 # 垂直の棒グラフを描く
```

```
13 axes[0].bar(col_names, dm[0], label=row_names[0])
14 axes[0].bar(col_names, dm[1], bottom=dm[0],
   label=row_names[1])
15 axes[0].set_title("Barplot of dm")
16 axes[0].legend()
17
18 # 行列を転置して水平の棒グラフを描く
19 dm_t = dm.T
20 for i in range(dm_t.shape[0]):
21     axes[1].barh(row_names, dm_t[i], left=np.sum(dm_t[:i],
   axis=0), label=col_names[i])
22 axes[1].set_title("Transposed Barplot of dm")
23 axes[1].legend()
24
25 # グラフを表示する
26 plt.tight_layout()
27 plt.show()
```



ノック 16 本目

以下の行列 dm について

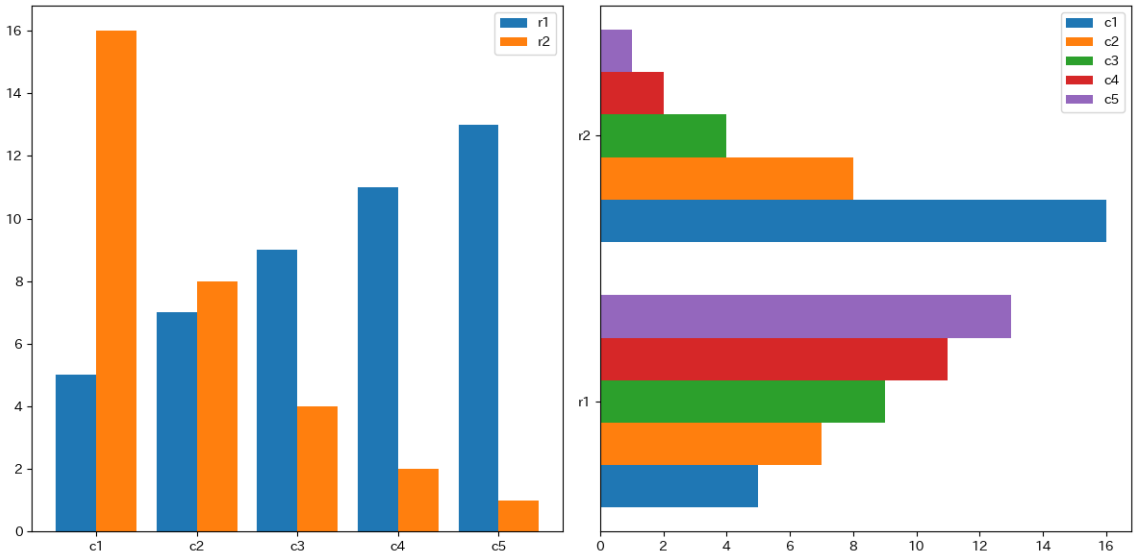
$$\mathrm{dm} = \begin{pmatrix} 5 & 7 & 9 & 11 & 13 \\ 16 & 8 & 4 & 2 & 1 \end{pmatrix}$$

行と列についてそれぞれの要素の棒グラフを重ねずに描画しよう。

RK16.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # データの設定
5  dm = np.array([[5, 7, 9, 11 13], [16, 8, 4, 2, 1]])
6  row_names = ["r1", "r2"]
7  col_names = ["c1", "c2", "c3", "c4", "c5"]
8
9  fig, axs = plt.subplots(1, 2, figsize=(12, 6))
10 total_width = 0.8
11
12 # 縦棒グラフの作成
13 x = np.arange(dm.shape[1])  # x 軸の位置を設定
14 width = total_width / dm.shape[0]  # 棒の幅を設定
15
16 # 最初の系列をプロット
17 axs[0].bar(x - width/2, dm[0], width=width, label='r1')
18 # 二つ目の系列をプロット
19 axs[0].bar(x + width/2, dm[1], width=width, label='r2')
20
21 # x 軸のラベルを設定
22 axs[0].set_xticks(x)
23 axs[0].set_xticklabels(col_names)
24 axs[0].legend()
25
26 # 横棒グラフの作成
27 dm_t = dm.T  # 行列を転置
28 y = np.arrange(dm_t.shape[1])  # y 軸の位置を設定
29 height = total_width / dm.shape[0]  # 棒の高さを設定
30
31 # 各系列をプロット
32 for i in range(dm_t.shape[0]):
33
```

```
    axes[1].barh(y - total_width / 2 + height * (i + 0.5),
34  dm_t[i], height=height, label=col_names[i])
35
36  # y 軸のラベルを設定
37  axs[1].set_yticks(x)
38  axs[1].set_yticklabels(col_names)
39  axs[1].legend()
40
41  # グラフを表示する
42  plt.tight_layout()
43  plt.show()
```



ヒストグラム

hist() を使うとヒストグラムを描画できる。

ノック 17 本目

札幌の 4 月 1-30 日(spDay)の最低気温(spMax)と最高気温(spMin)のデータです。横軸に温度(1 度刻み)をとり spMax と spMin を重ねてヒストグラムで表してみよう。

spDay	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
spMax	21.9	24.5	23.4	26.2	15.3	22.4	21.8	16.8	19.9	19.2	21.9	25.9	20.9	18.8	22.1
spMin	8.3	13.0	8.4	7.9	7.0	3.7	6.1	8.5	8.6	11.9	12.1	14.4	7.0	10.5	6.6

spDay	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
spMax	20.0	15.0	16.0	22.2	26.4	26.0	28.3	18.7	21.3	22.5	25.0	22.0	26.1	25.6	25.7
spMin	10.6	16.6	19.1	20.1	19.8	24.5	12.6	16.4	13.0	13.3	14.1	14.4	17.0	21.3	24.5

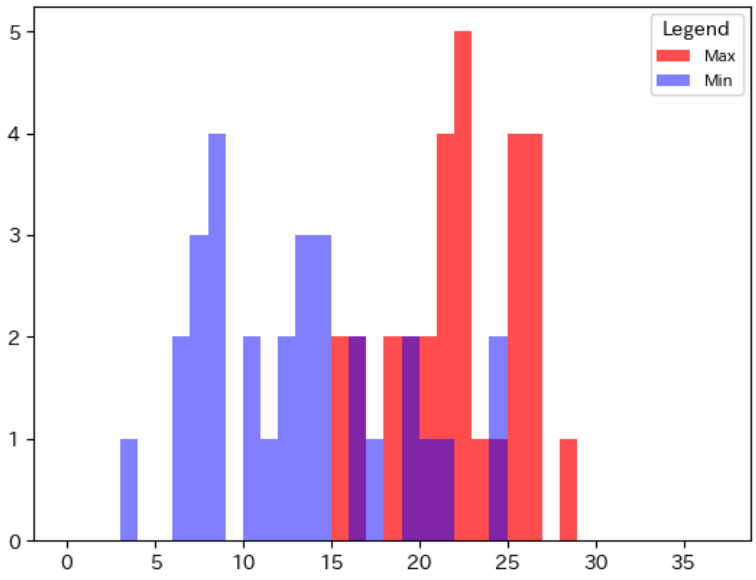
RK17.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # データを作成する
5  spMax = np.array([21.9, 24.5, 23.4, 26.2, 15.3, 22.4, 21.8,
   16.8, 19.9, 19.2,
6                    21.9, 25.9, 20.9, 18.8, 22.1, 20.0, 15.0,
   16.0, 22.2, 26.4,
7                    26.0, 28.3, 18.7, 21.3, 22.5, 25.0, 22.0,
   26.1, 25.6, 25.7])
8
9  spMin = np.array([8.3, 13.0, 8.4, 7.9, 7.0, 3.7, 6.1, 8.5,
   8.6, 11.9,
10                   12.1, 14.4, 7.0, 10.5, 6.6, 10.6, 16.6, 19.1,
   20.1, 19.8,
11                   24.5, 12.6, 16.4, 13.0, 13.3, 14.1, 14.4,
   17.0, 21.3, 24.5])
12
13 # ヒストグラムを描く
```

```
    plt.hist(spMax, bins=np.arange(0, 38, 1), color='red',
14  alpha=0.7, label='Max')
    plt.hist(spMin, bins=np.arange(0, 38, 1), color='blue',
15  alpha=0.5, label='Min')

16  # 凡例を追加する
17  plt.legend(loc='upper right', title='Legend',
18  fontsize='small')

19  # グラフを表示する
20  plt.show()
```

5-11 行目：Python では spMax = np.array([21.9, 24.5, …, 25.7]) のように数値をベクトルで表すときには、[] の中にカンマ区切って記述します(ファイルからの直接読み込むこともできます、後で説明します)。

13-21 行目：ヒストグラム（頻度表）を描画します。



ヒストグラムを重ねないで描画することもできる。

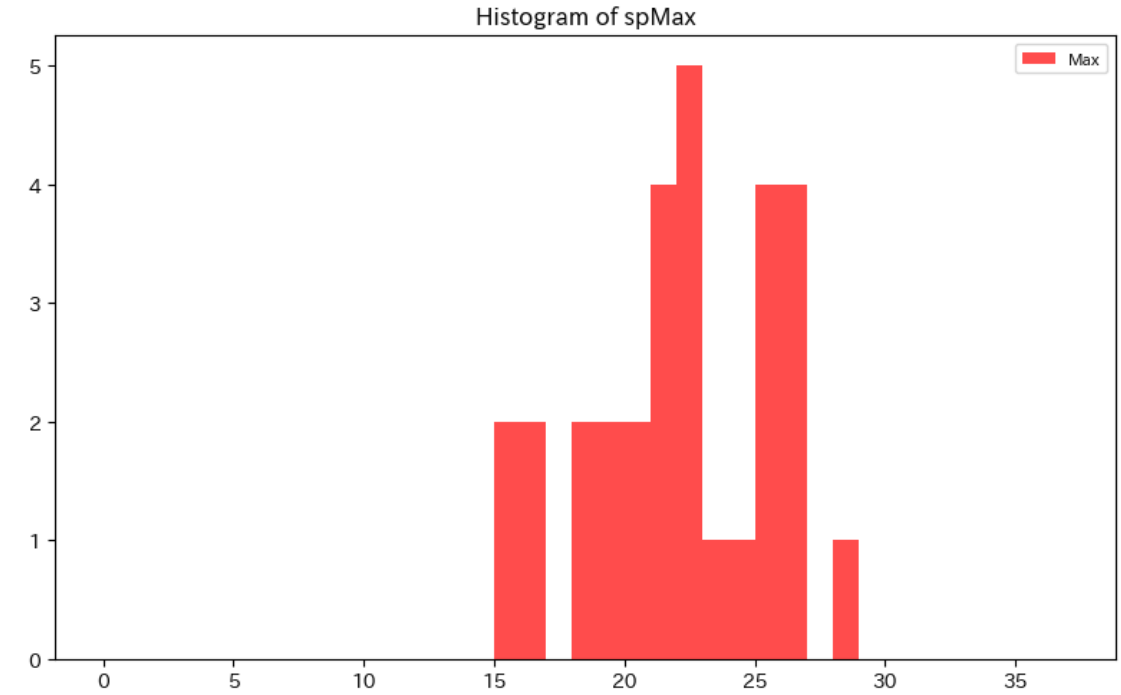
ノック 18 本目

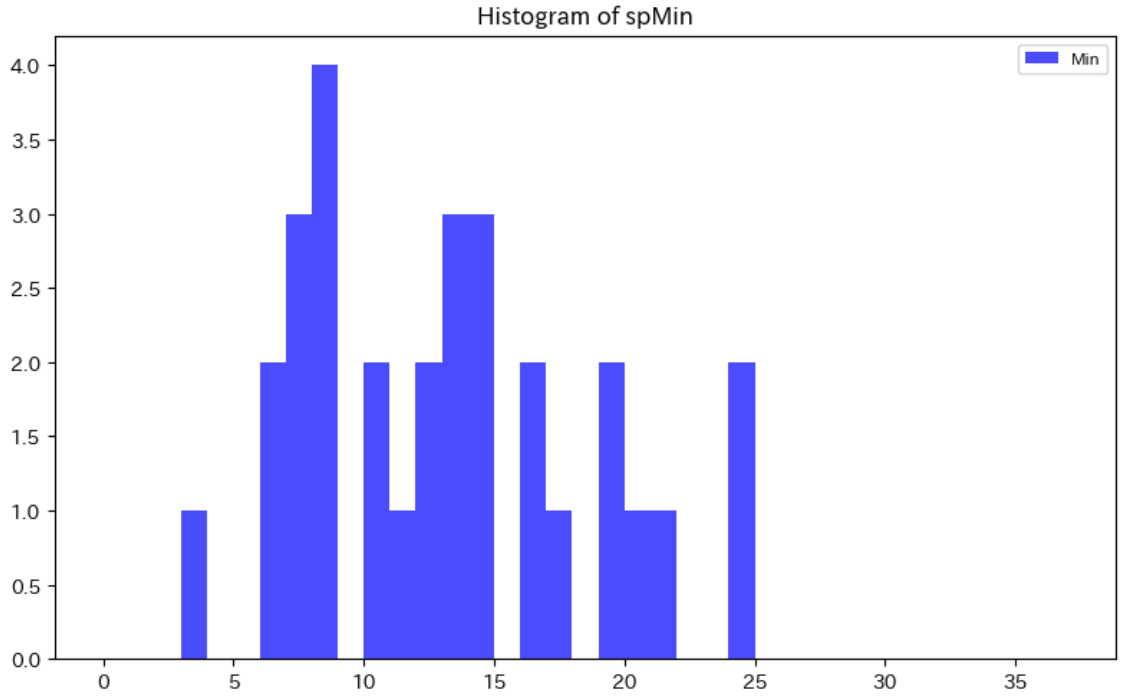
ノック 17 本目のプログラムを以下のプログラムに置き換えて実行してみよう。

RK18.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # データを作成する
5  spMax = np.array([21.9, 24.5, 23.4, 26.2, 15.3, 22.4, 21.8, 16.8, 19.9, 19.2,
6                    21.9, 25.9, 20.9, 18.8, 22.1, 20.0, 15.0, 16.0, 22.2, 26.4,
7                    26.0, 28.3, 18.7, 21.3, 22.5, 25.0, 22.0, 26.1, 25.6, 25.7])
8
9  spMin = np.array([8.3, 13.0, 8.4, 7.9, 7.0, 3.7, 6.1, 8.5, 8.6, 11.9,
10                   12.1, 14.4, 7.0, 10.5, 6.6, 10.6, 16.6, 19.1, 20.1, 19.8,
11                   24.5, 12.6, 16.4, 13.0, 13.3, 14.1, 14.4, 17.0, 21.3, 24.5])
12
13 # プロットを 2 つ縦に並べて表示する設定
14 fig, axes = plt.subplots(2, 1, figsize=(8, 10))
15
16 # 上のヒストグラムを描く
17 axes[0].hist(spMax, bins=np.arange(0, 38, 1), color='red', alpha=0.7, label='Max')
18 axes[0].set_title("Histogram of spMax")
19 axes[0].legend(loc='upper right', fontsize='small')
20
21 # 下のヒストグラムを描く
```

```
22 axes[1].hist(spMin, bins=np.arange(0, 38, 1), color='blue',
   alpha=0.7, label='Min')
23 axes[1].set_title("Histogram of spMin")
24 axes[1].legend(loc='upper right', fontsize='small')
25
26 # グラフを表示する
27 plt.tight_layout()
28 plt.show()
```


Histogram of spMax
Max
0
1
2
3
4
5
0
5
10
15
20
25
30
35


Histogram of spMin
Min
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
5
10
15
20
25
30
35

ヒストグラムと集計

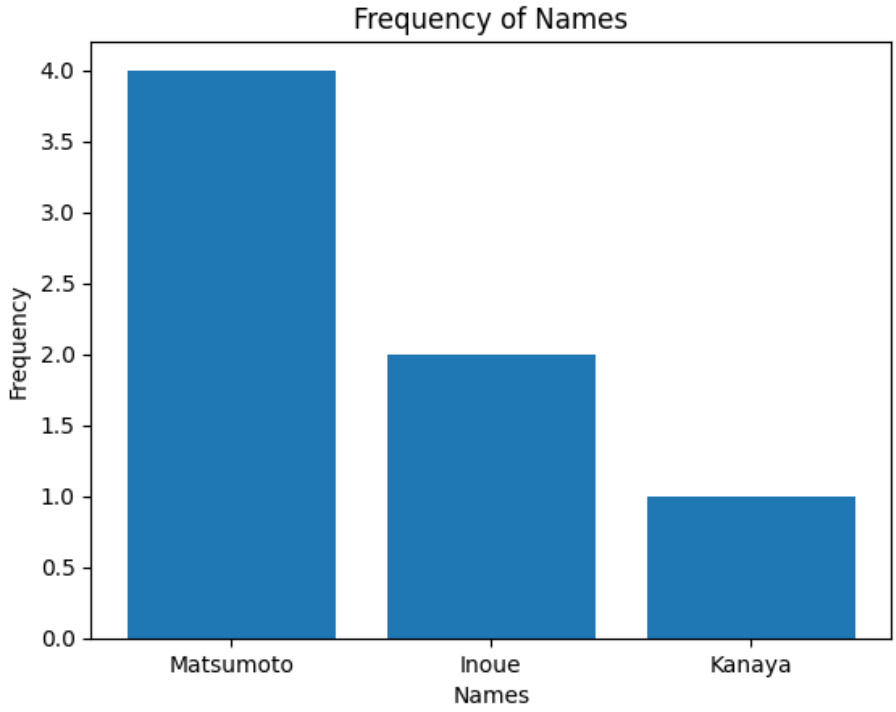
要素の値あるいは文字列を集計し、結果を棒グラフで表してみましょう。

ノック 19 本目

RK19.py では、dataC に含まれている 3 種の文字列(Inoue、Matsumoto、Kanaya)の出現数を集計しています。実行して確認しよう。

RK19.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2
3  # データを作成する
4  dataC = ["Inoue", "Inoue", "Matsumoto", "Matsumoto", "Kanaya", "Matsumoto", "Matsumoto"]
5
6  # データの頻度を計算する
7  dataC_table = {name: dataC.count(name) for name in set(dataC)}
8
9  # グラフを描く
10 plt.bar(dataC_table.keys(), dataC_table.values())
11 plt.xlabel('Names')
12 plt.ylabel('Frequency')
13 plt.title('Frequency of Names')
14
15 # グラフを表示する
16 plt.show()
```



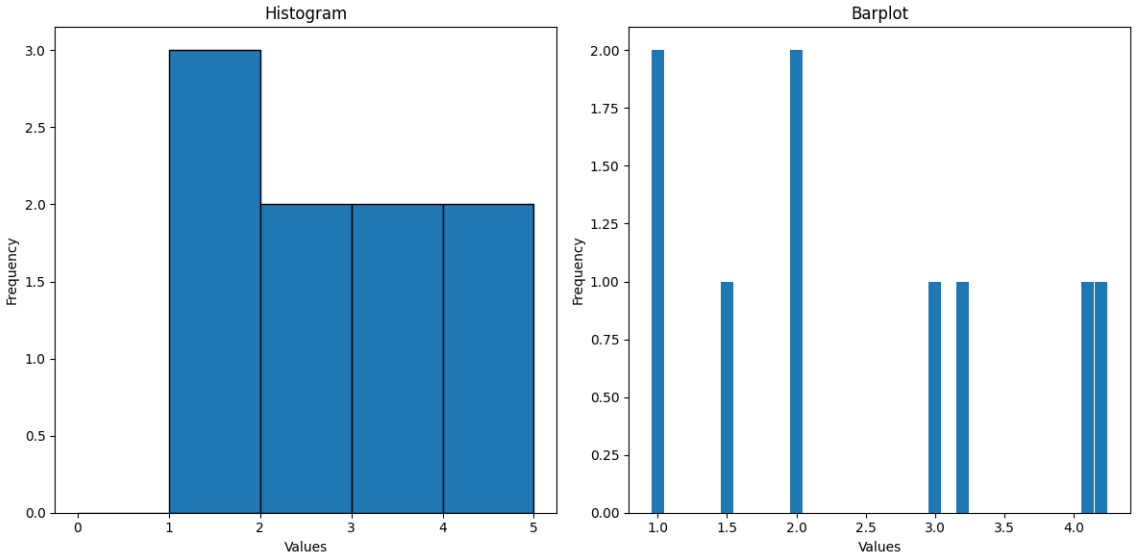
ノック 20 本目

RK20.py を実行し、hist() 関数と bar() を比較しよう。

RK20.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # データを作成する
5  dataC = [1.5, 2, 3, 4.1, 1, 3.2, 2, 1, 4.2]
6
7  # プロットを 2 つ並べて表示する設定
8  fig, axes = plt.subplots(1, 2, figsize=(12, 6))
9
10 # ヒストグラムを描く
11 axes[0].hist(dataC, bins=np.arange(0, 6, 1),
   edgecolor='black')
12 axes[0].set_title("Histogram")
13 axes[0].set_xlabel('Values')
```

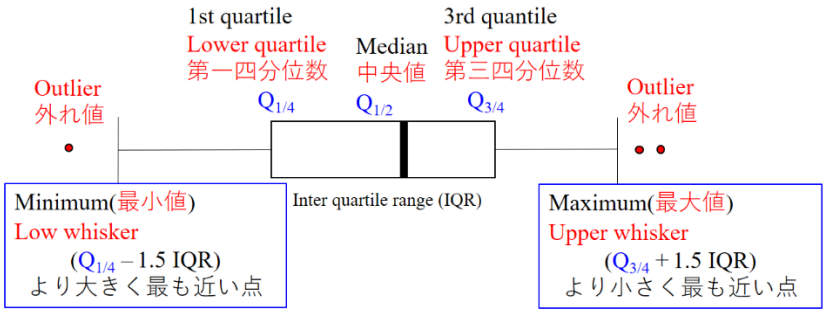
```
14 axes[0].set_ylabel('Frequency')
15
16 # データの頻度を計算する
17 dataC_table = {value: dataC.count(value) for value in
   set(dataC)}
18
19 # バープロットを描く
20 axes[1].bar(dataC_table.keys(), dataC_table.values(),
   width=0.09)
21 axes[1].set_title("Barplot")
22 axes[1].set_xlabel('Values')
23 axes[1].set_ylabel('Frequency')
24
25 # グラフを表示する
26 plt.tight_layout()
27 plt.show()
```



中央値、平均値、箱ひげ図

奇数個の要素からなる集合 {1, 4, 5, 8, 11} があったとき、これらの中央値は 8 となります。一方、偶数個の要素 {1, 4, 5, 6, 8, 11} があった時、中央は一意に決まらないので、5 と 6 の平均値をとり 5.5 とします。平均値は、数字の合計を要素数で割った値です。つまり、集合 {1, 4, 5, 8, 11} についての平均値は (1 + 4 + 5 + 8 + 11) / 5 = 5.8 となります。

箱ひげ図とは、以下に示すように、数値を小さい順に並べて、$Q_{1/4}$（第一四分位数；小さい方から 1/4 位の数)、$Q_{1/2}$（中央値；小さい方から 1/2 位の数)、$Q_{3/4}$（第三四分位数；小さい方から 3/4 位の数)をもとに、最小値（$Q_{1/4}$ - 1.5 IQR, IQR = $Q_{3/4}$ - $Q_{1/4}$）と最大値（$Q_{3/4}$ + IQR, IQR = $Q_{3/4}$ - $Q_{1/4}$）として、データの分布をみる方法です。



ノック 21 本目

札幌の 4 月 1-30 日(spDay)の最低気温(spMax)と最高気温(spMin)のデータです。spMax と spMin について中央値、平均値を求め、箱ひげ図を描画しよう。

spDay	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
spMax	21.9	24.5	23.4	26.2	15.3	22.4	21.8	16.8	19.9	19.2	21.9	25.9	20.9	18.8	22.1
spMin	8.3	13.0	8.4	7.9	7.0	3.7	6.1	8.5	8.6	11.9	12.1	14.4	7.0	10.5	6.6

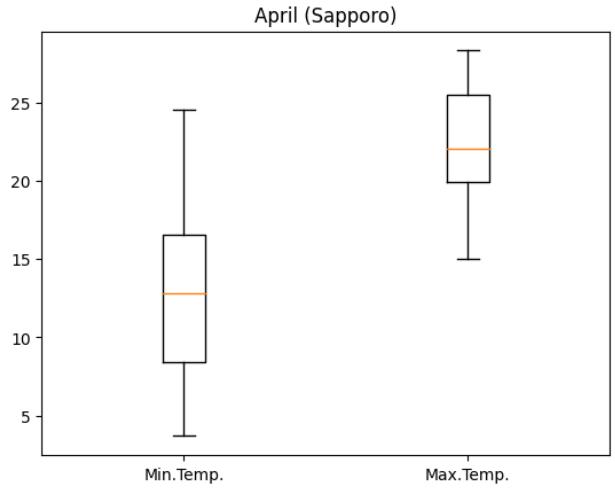
spDay	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
spMax	20.0	15.0	16.0	22.2	26.4	26.0	28.3	18.7	21.3	22.5	25.0	22.0	26.1	25.6	25.7
spMin	10.6	16.6	19.1	20.1	19.8	24.5	12.6	16.4	13.0	13.3	14.1	14.4	17.0	21.3	24.5

RK21.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # データを作成する
```

```
    spMax = np.array([21.9, 24.5, 23.4, 26.2, 15.3, 22.4, 21.8,
5   16.8, 19.9, 19.2,
                  21.9, 25.9, 20.9, 18.8, 22.1, 20.0, 15.0,
6   16.0, 22.2, 26.4,
                  26.0, 28.3, 18.7, 21.3, 22.5, 25.0, 22.0,
7   26.1, 25.6, 25.7])

8   spMin = np.array([8.3, 13.0, 8.4, 7.9, 7.0, 3.7, 6.1, 8.5,
9   8.6, 11.9,
                  12.1, 14.4, 7.0, 10.5, 6.6, 10.6, 16.6, 19.1,
10  20.1, 19.8,
                  24.5, 12.6, 16.4, 13.0, 13.3, 14.1, 14.4,
11  17.0, 21.3, 24.5])

12  # 平均を計算して表示する
13  mean_spMax = np.mean(spMax)
14  mean_spMin = np.mean(spMin)
15  print("Mean of spMax:", mean_spMax)
16  print("Mean of spMin:", mean_spMin)
17
18  # 中央値を計算して表示する
19  median_spMax = np.median(spMax)
20  median_spMin = np.median(spMin)
21  print("Median of spMax:", median_spMax)
22  print("Median of spMin:", median_spMin)
23
24  # 箱ひげ図を描く
25  plt.boxplot([spMin, spMax], labels=["Min.Temp.",
26  "Max.Temp."])
    plt.title("April (Sapporo)")
27  plt.show()
```

```
Mean of spMax: 22.060000000000002
Mean of spMin: 13.043333333333333
Median of spMax: 22.05
Median of spMin: 12.8
```


April (Sapporo)
25
20
15
10
5
Min.Temp.
Max.Temp.

散布図

二つの軸 x と y の値に従ってデータをプロットする図を散布図といいます。

ノック 22 本目

札幌の 4 月 1-30 日(spDay)の最低気温(spMax)と最高気温(spMin)のデータです。
spMax と spMin をそれぞれ(x,y)として、1 日目から 30 日までを散布図にプロットしてみよう。

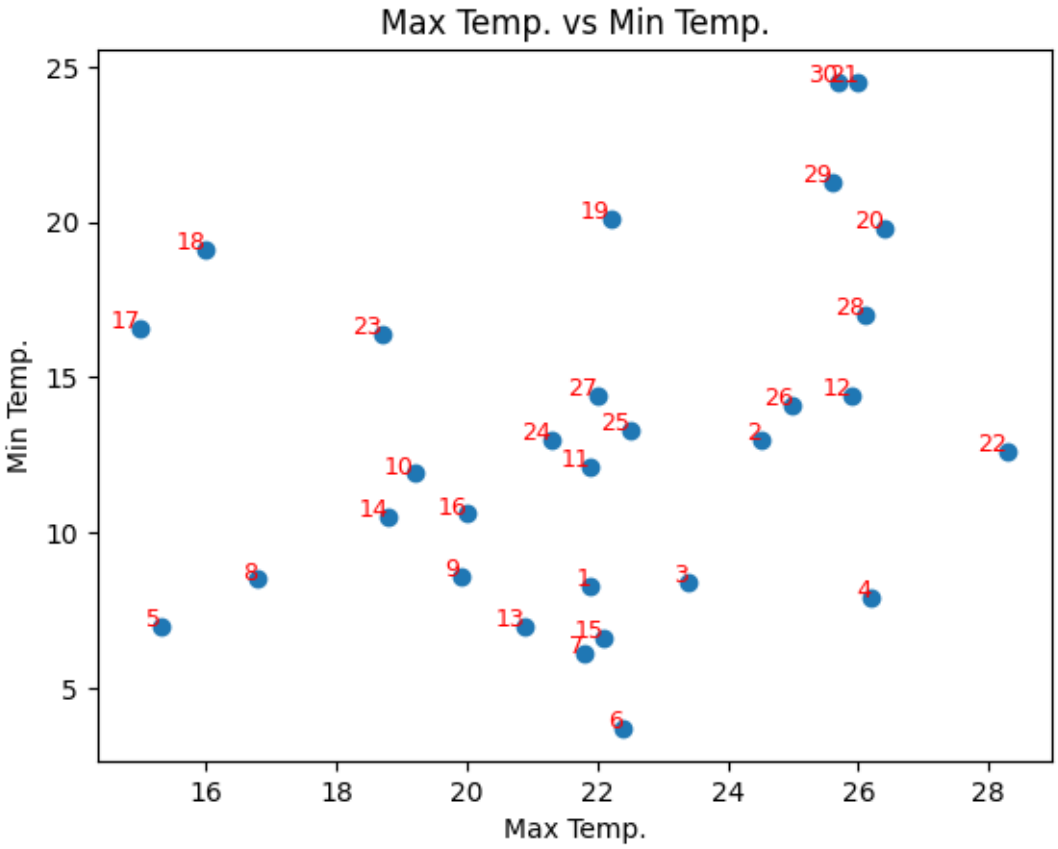
spDay	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
spMax	21.9	24.5	23.4	26.2	15.3	22.4	21.8	16.8	19.9	19.2	21.9	25.9	20.9	18.8	22.1
spMin	8.3	13.0	8.4	7.9	7.0	3.7	6.1	8.5	8.6	11.9	12.1	14.4	7.0	10.5	6.6

spDay	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
spMax	20.0	15.0	16.0	22.2	26.4	26.0	28.3	18.7	21.3	22.5	25.0	22.0	26.1	25.6	25.7
spMin	10.6	16.6	19.1	20.1	19.8	24.5	12.6	16.4	13.0	13.3	14.1	14.4	17.0	21.3	24.5

RK22.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2
3  # データを作成する
4  spDay = range(1, 31)
5  spMax = [21.9, 24.5, 23.4, 26.2, 15.3, 22.4, 21.8, 16.8,
6  19.9, 19.2,
7          21.9, 25.9, 20.9, 18.8, 22.1, 20.0, 15.0, 16.0, 22.2,
8  26.4,
           26.0, 28.3, 18.7, 21.3, 22.5, 25.0, 22.0, 26.1, 25.6,
9  25.7]

10 spMin = [8.3, 13.0, 8.4, 7.9, 7.0, 3.7, 6.1, 8.5, 8.6, 11.9,
           12.1, 14.4, 7.0, 10.5, 6.6, 10.6, 16.6, 19.1, 20.1,
11 19.8,
12         24.5, 12.6, 16.4, 13.0, 13.3, 14.1, 14.4, 17.0, 21.3,
13 24.5]

14 # プロットを作成する
```

```
    plt.scatter(spMax, spMin)
15  plt.xlabel('Max Temp.')
16  plt.ylabel('Min Temp.')
17  plt.title('Max Temp. vs Min Temp.')
18
19  # 各ポイントにラベルを追加する
20  for i in range(len(spDay)):
        plt.text(spMax[i], spMin[i], str(spDay[i]), color='red',
21  fontsize=9, ha='right')
22
23  # プロットを表示する
24  plt.show()
```



折れ線グラフ

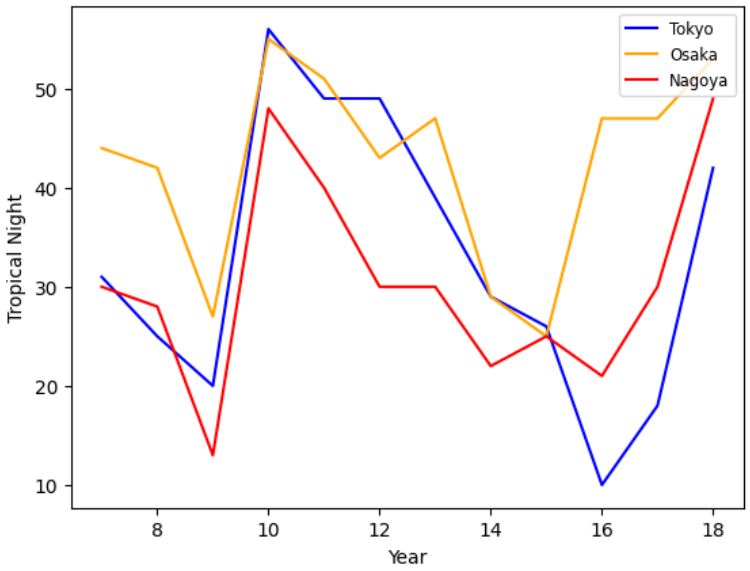
ノック 23 本目

2007 年から 2018 年(Year)のそれぞれの年に、最低気温が 25℃以上（熱帯夜）であった日数を Tokyo、Osaka、Nagoya で集計した。年ごとにどのような変化があるか、3 つの地点で折れ線グラフに表してみよう。

Year	7	8	9	10	11	12	13	14	15	16	17	18
Tokyo	31	25	20	56	49	49	39	29	26	10	18	42
Osaka	44	42	27	55	51	43	47	29	25	47	47	53
Nagoya	30	28	13	48	40	30	30	22	25	21	30	49

RK23.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2
3  # データを作成する
4  Year = range(7, 19)
5  Tokyo = [31, 25, 20, 56, 49, 49, 39, 29, 26, 10, 18, 42]
6  Osaka = [44, 42, 27, 55, 51, 43, 47, 29, 25, 47, 47, 53]
7  Nagoya = [30, 28, 13, 48, 40, 30, 30, 22, 25, 21, 30, 49]
8
9  # プロットを作成する
10 plt.plot(Year, Tokyo, label='Tokyo', color='blue')
11 plt.plot(Year, Osaka, label='Osaka', color='orange')
12 plt.plot(Year, Nagoya, label='Nagoya', color='red')
13 plt.xlabel('Year')
14 plt.ylabel('Tropical Night')
15 plt.legend(loc='upper right', fontsize='small')
16
17 # プロットを表示する
18 plt.show()
```


Tokyo
Osaka
Nagoya
Tropical Night
50
40
30
20
10
8
10
12
14
16
18
Year

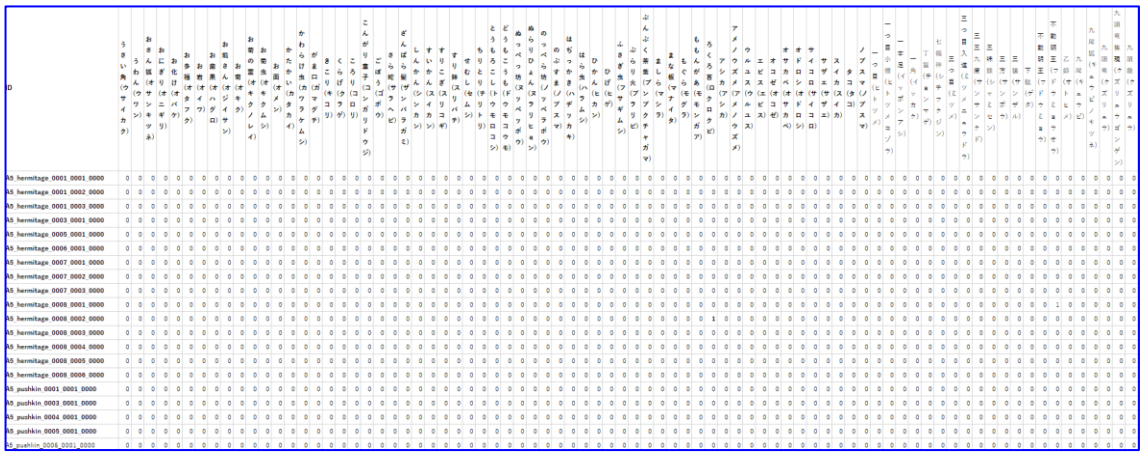
III.知識発見

3. 妖怪

NAIST STELLA プログラム「共創」では、「高院の連携実績と広範な学術・国際・地域ネットワークで挑む」、次世代型理数系人材の育成を進めております。その中で、国際日本文化研究センターは、日本文化を国際的な視野で、学際的かつ総合的に研究していこうとする研究機関であり、さまざまな資料を有し、公開を行っています。その中に、

怪異・妖怪画像データベース(DB と略す)(https://www.nichibun.ac.jp/YoukaiGazou/)という妖怪に関する DB を作成し、公開しています。

怪異・妖怪画像 DB の「あ」のところの葵の上(アオイノウエ)をクリックすると、検索結果:1 件と表示され、画像が出力されます(中央の図)さらに、この画像をクリックすると下のほうに、書誌情報が記載されています。葵の上についての資源識別子(U426_nichibunken_0051_0009_0000)がついており、この番号は一つの絵とユニーク(一対一)に対応付けられています。また、主題をみると「鬼：オニ」「鬼女：キジョ」がつけられています。それでは、主題となる用語で、妖怪画がどのくらいあるか、プログラムをつくって検討してみましょう。DataYokai.csv ファイルには、行が資源識別子、列が主題となっています。妖怪の絵（資源識別子）に使われている主題については 1 をそうでなければ 0 となっています。



ノック 24 本目

DataYokai.csv をもとに、主題キーワードの個数と、妖怪の絵の数(サンプル数)を求めてみましょう。妖怪の絵にはそれぞれ、主題となるキーワードが使われていれますが、一つの妖怪の絵あたりにどのくらい使われているか検討してみましょう。また最大の主題キーワード数となっている妖怪の絵を探してみよう。

RK24.py

```python
import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
import pandas as pd  # データフレーム操作のためのライブラリをインポート

# CSV ファイルを読み込む（エンコーディングを指定）
dataYokai = pd.read_csv('DataYokai.csv', header=0, index_col=0, encoding='shift-jis')

# データフレームの次元を表示する
print("Dimensions of dataYokai:", dataYokai.shape)

# 行ごとの合計を計算する
sumrow = dataYokai.apply(sum, axis=1)

# ヒストグラムを描く
plt.hist(sumrow, bins=range(0, 16, 1), edgecolor='black')
plt.xlabel('Sum of Rows')
```

```
plt.ylabel('Frequency')
plt.title('Histogram of Sum of Rows')
plt.show()

# 最大値を表示する
max_sumrow = sumrow.max()
print("Max sum of rows:", max_sumrow)

# 最大値に該当する行名を表示する
names_max_sumrow = sumrow[sumrow ==
max_sumrow].index.tolist()
print("Names with max sum of rows:", names_max_sumrow)
```

> print("Dimensions of dataYokai:", dataYokai.shape)

```
Dimensions of dataYokai: (2861, 872)
```

全ての妖怪の絵は 2861 枚、使われている主題キーワードは 872 個ということがわかる。

> sumrow = dataYokai.apply(sum, axis=1)

行(妖怪の絵)ごとの使われている主題キーワードの数を数えている。

> plt.hist(sumrow, bins=range(0, 16, 1), edgecolor='black')

…

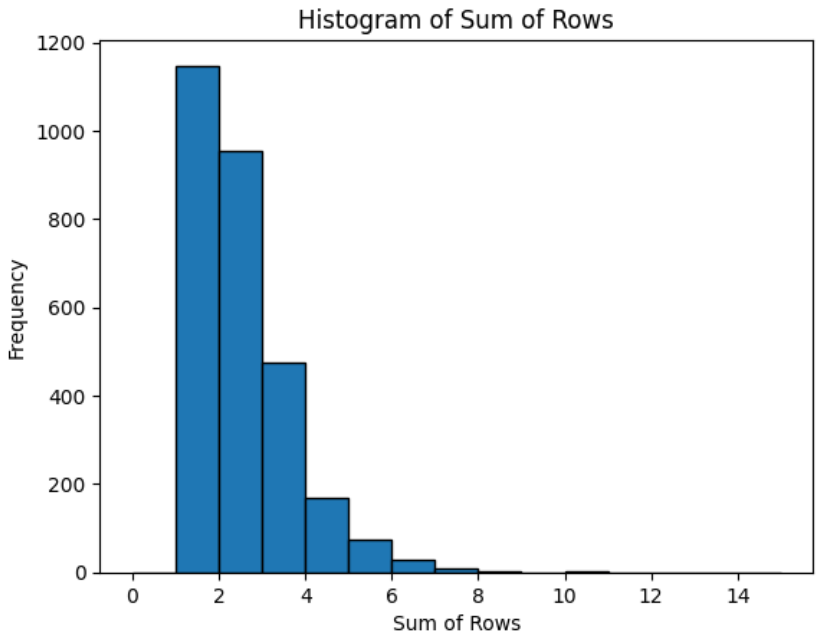
ヒストグラムを書くと大半の妖怪の絵には 2-3 個の主題キーワードが活用されている。

> max_sumrow = sumrow.max()

```
Max sum of rows: 10
```

> names_max_sumrow = sumrow[sumrow == max_sumrow].index.tolist()

```
Names with max sum of rows: ['U426_nichibunken_0104_0006_0000']
```



主題キーワードが最も多く使われいる絵の資源識別子は、"U426_nichibunken_0104_0006_0000"であり、なんと10 個の主題キーワードが活用されている。

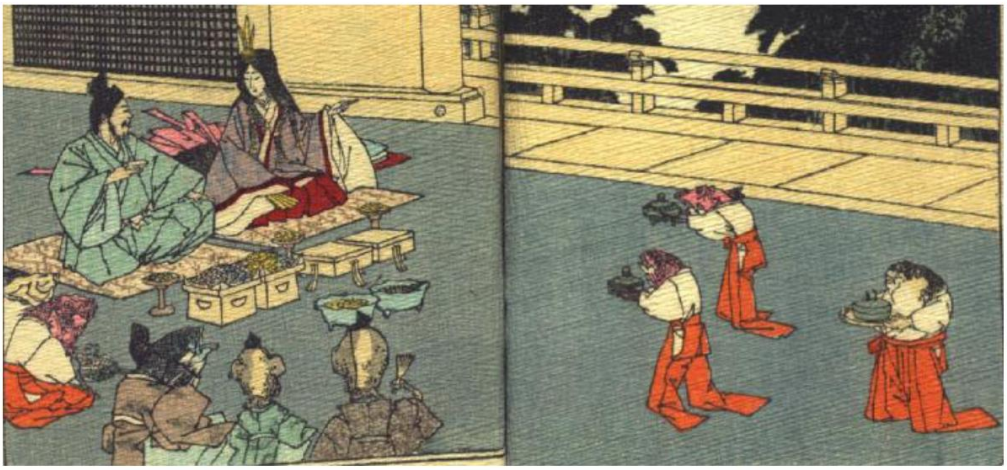
U426_nichibunken_0104_0006_0000 を Google 検索してみると、この絵には、「乙姫；オトヒメ」，「女；オンナ」，「魚；サカナ」，「河豚；フグ」，「蛸；タコ」，「女官；ニョカン」，「冠；カンムリ」，「扇子；センス」，「烏帽子；エボシ」，「食べ物；タベモノ」と 10 個の主題キーワードがついている。

国際日本文化研究センター
怪異・妖怪画像データベース

検索

乙姫;オトヒメ, 女;オンナ, 魚;サカナ, 河豚;フグ, 蛸;タコ, 女官;ニョカン, 冠;カンムリ, 扇子;センス, 烏帽子;エボシ, 食べ物;タベモノ

J.Dautremer



タイトル	
著作者	J.Dautremer
主題	乙姫;オトヒメ, 女;オンナ, 魚;サカナ, 河豚;フグ, 蛸;タコ, 女官;ニョカン, 冠;カンムリ, 扇子;センス, 烏帽子;エボシ, 食べ物;タベモノ
内容記述	乙姫は女房装束に冠を着け、片手に扇子を持って座り、浦島太郎に話しかけている。浦島太郎は公家装束に烏帽子をかぶって座り、盃を片手に乙姫の話を聞いている。彼の前に食べ物が供されている。白い着物に赤い袴という女官のような装束の魚たちが器を運んでいる。そのうちの1体は河豚のようである。同じ装束の魚2体が浦島太郎の脇に座って控えている。公家装束に烏帽子の魚1体と蛸2体も同席している。蛸の1体は片手に扇子を持っている。
公開者	所蔵者:国際日本文化研究センター
寄与者	
日付	2006
資源タイプ	画像
フォーマット	
資源識別子	U426_nichibunken_0104_0006_0000
情報源	親書誌:U426_nichibunken_0104:Ourasima
言語	フランス語

ノック 25 本目

主題キーワードの使用頻度から、妖怪画について検討してみよう。

DataYokai.csv をもとに、主題キーワードの使用回数の表と棒グラフで表してみよう。

RK25.py

```
import re # 正規表現を扱うためのライブラリをインポート

import japanize_matplotlib  # 日本語を表示するためのライブラリをインポート
import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
import pandas as pd  # データフレーム操作のためのライブラリをインポート

# CSV ファイルを読み込む（エンコーディングを指定）
dataYokai = pd.read_csv('DataYokai.csv', header=0,
index_col=0, encoding='shift-jis')

# 列ごとの合計を計算する
Yokailabel = dataYokai.apply(sum, axis=0)

# 合計を昇順にソートする
sYokai = Yokailabel.sort_values()

# 合計が 9 を超えるものを抽出する
s20 = sYokai[sYokai > 9]

# 抽出したデータを降順にソートする
dataSS = s20.sort_values(ascending=False)

# データフレームを全て表示するための設定
pd.set_option('display.max_rows', None)

# データフレームとして表示する
result_df = pd.DataFrame({'names': dataSS.index, 'dataSS':
dataSS.values})
print(result_df)
```

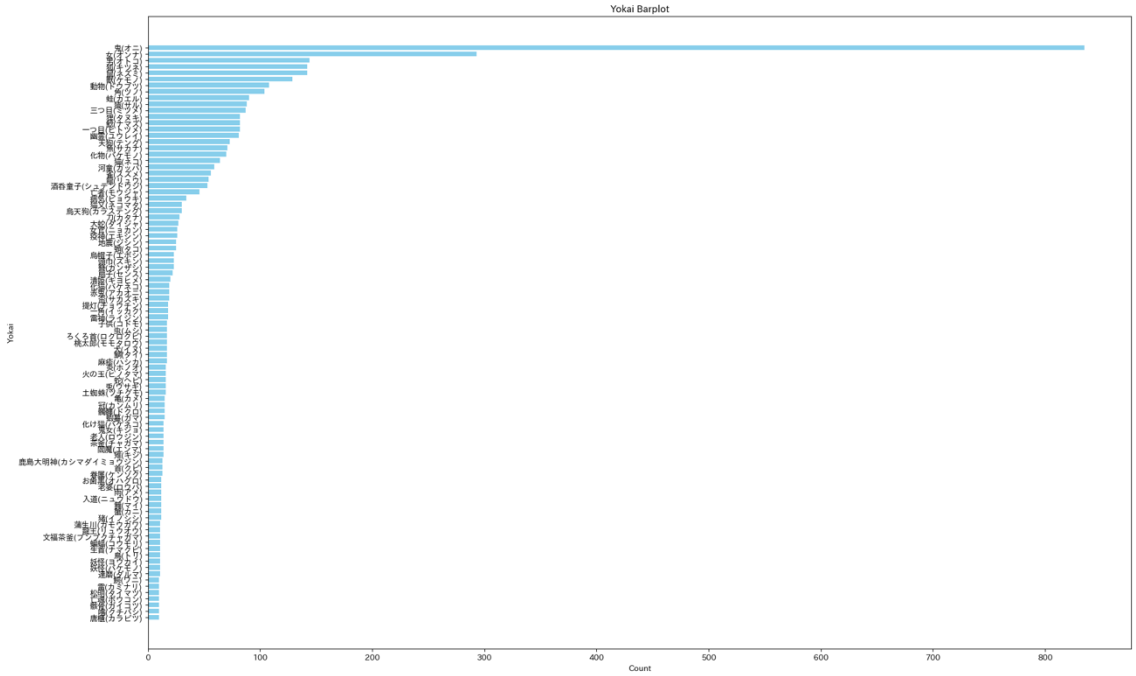
```
28
29  # プロットの設定を行う
30  plt.figure(figsize=(20, 12))
31  plt.barh(dataSS.index, dataSS.values, color='skyblue')
32  plt.xlabel('Count')
33  plt.ylabel('Yokai')
34  plt.title('Yokai Barplot')
35  plt.gca().invert_yaxis()
36  plt.tight_layout()
37
    # グラフを表示する
    plt.show()
```

ここでは、10 種以上の妖怪画像で活用されている主題キーワードを以下に示す。主題キーワードの総数 872 個のうち、10 種以上の妖怪画像で活用されている主題キーワードは 92 個だった。

ほとんどの主題キーワードは数枚の妖怪画像の説明に使われている。

```
> print(result_df)
                 names  dataSS
0              鬼(オニ)     835
1              女(オンナ)     293
2              男(オトコ)     144
3              狐(キツネ)     142
4              鼠(ネズミ)     142
5              獣(ケモノ)     129
6           動物(ドウブツ)     108
7              角(ツノ)     104
8              蛙(カエル)      90
9              猿(サル)      88
10          三つ目(ミツメ)      87
11             狸(タヌキ)      82
12             鯰(ナマズ)      82
13         一つ目(ヒトツメ)      82
14          幽霊(ユウレイ)      81
15           天狗(テング)      73
16             魚(サカナ)      71
17          化物(バケモノ)      70
18             猫(ネコ)      64
19           河童(カッパ)      59
20             雀(スズメ)      56
21             龍(リュウ)      54
22     酒呑童子(シュテンドウジ)      53
23          亡者(モウジャ)      46
24          病気(ビョウキ)      34
25          猫又(ネコマタ)      30
26       烏天狗(カラステング)      30
27             刀(カタナ)      28
28          大蛇(ダイジャ)      27
29          女官(ニョカン)      26
```

30 疫神(エキシン) 26
31 地震(ジシン) 25
32 蛸(タコ) 25
33 烏帽子(エボシ) 23
34 頭巾(ズキン) 23
35 簪(カンザシ) 23
36 扇子(センス) 22
37 清姫(キヨヒメ) 20
38 化猫(バケネコ) 19
39 赤鬼(アカオニ) 19
40 盃(サカズキ) 19
41 提灯(チョウチン) 18
42 一角(イッカク) 18
43 雷神(ライジン) 18
44 子供(コドモ) 17
45 虫(ムシ) 17
46 ろくろ首(ロクロクビ) 17
47 桃太郎(モモタロウ) 17
48 犬(イヌ) 17
49 鯛(タイ) 17
50 麻疹(ハシカ) 17
51 炎(ホノオ) 16
52 火の玉(ヒノタマ) 16
53 蛇(ヘビ) 16
54 兎(ウサギ) 16
55 土蜘蛛(ツチグモ) 16
56 亀(カメ) 15
57 冠(カンムリ) 15
58 髑髏(ドクロ) 15
59 蝦蟇(ガマ) 15
60 化け猫(バケネコ) 14
61 鬼女(キジョ) 14
62 老人(ロウジン) 14
63 茶釜(チャガマ) 14
64 閻魔(エンマ) 14
65 雉(キジ) 14
66 鹿島大明神(カシマダイミョウジン) 13
67 首(クビ) 13
68 眷属(ケンゾク) 13
69 お歯黒(オハグロ) 12
70 老婆(ロウバ) 12
71 雨(アメ) 12
72 入道(ニュウドウ) 12
73 舞(マイ) 12
74 蟹(カニ) 12
75 猪(イノシシ) 12
76 蒲生川(ガモウガワ) 11
77 龍王(リュウオウ) 11
78 文福茶釜(ブンブクチャガマ) 11
79 蝙蝠(コウモリ) 11
80 生首(ナマクビ) 11
81 鳥(トリ) 11
82 妖怪(ヨウカイ) 11
83 妖怪(バケモノ) 11
84 達磨(ダルマ) 11
85 鰐(ワニ) 10
86 雷(カミナリ) 10
87 松明(タイマツ) 10
88 亡魂(ボウコン) 10
89 骸骨(ガイコツ) 10
90 嘴(クチバシ) 10
91 唐櫃(カラビツ) 10



このように、非常に多くの妖怪画像の説明に使われている主題キーワードと、ほんのわずかの妖怪画像の説明に使われている主題キーワードが存在していることがわかる。

パワーロー（ベキ乗則）[数学 I]

1 つの妖怪画に使われている主題キーワードの種類数を $f(1)$、

2 つの妖怪画に使われている主題キーワードの種類数を $f(2)$、

…

u 個の妖怪画に使われている主題キーワードの種類数を $f(u)$

としよう。

ここで、

$f(u) = au^{-k}$

が成り立つ場合、これを冪乗則（ベキ乗則、power law）という。この式の両辺について対数をとると、

$\log f(u) = \log a - k \log u$

となるので、u 個の妖怪画に使われている主題キーワードの種類数である $f(u)$ について対数をとると線形の関係になる。このことをプログラミングにより実験してみよう。

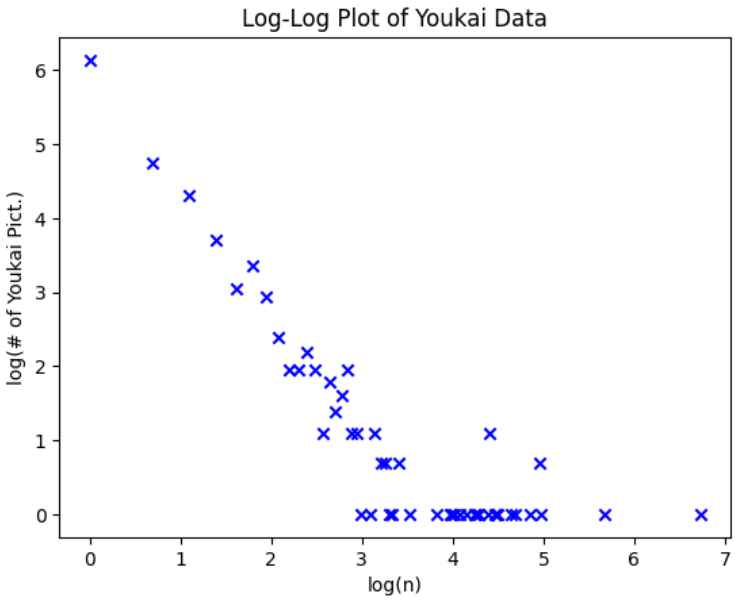
ノック 26 本目

u 個の妖怪画に使われている主題キーワードの種類数である $f(u)$ について対数をとったとき線形の関係になればパワーローということになる。この法則が妖怪画データの場合に成り立つか検討しよう。

RK26.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3  import statsmodels.api as sm  # 線形回帰のためのライブラリをインポート
4
5  # CSV ファイルを読み込む（エンコーディングを指定）
6  dataYokai = pd.read_csv('DataYokai.csv', header=0,
   index_col=0, encoding='shift-jis')
7
8  # 列ごとの合計を計算する
9  Yokailabel = dataYokai.apply(sum, axis=0)
10
11 # 合計を降順にソートする
12 sYokai = Yokailabel.sort_values(ascending=False)
```

```
13
14  # 合計の頻度を計算する
15  ndata = sYokai.value_counts()
16
17  # ndata のインデックスを整数に変換する
18  nn = ndata.index.astype(int)
19
20  # nn の対数を計算する
21  logn = np.log(nn)
22
23  # ndata の対数を計算する
24  logfn = np.log(ndata.values)
25
26  # 線形回帰を実行し、結果を表示する
27  X = sm.add_constant(logn)
28  model = sm.OLS(logfn, X).fit()
29  print(model.summary())
30
31  # 散布図を描く
32  plt.scatter(logn, logfn, c='blue', marker='x')
33  plt.xlabel('log(n)')
34  plt.ylabel('log(# of Youkai Pict.)')
35  plt.title('Log-Log Plot of Youkai Data')
36  plt.show()
```




```
> print(model.summary())
```

OLS Regression Results

Dep. Variable:	logfn	R-squared:	0.746
Model:	OLS	Adj. R-squared:	0.741
Method:	Least Squares	F-statistic:	138.4
Date:	Sat, 01 Jun 2024	Prob (F-statistic):	1.32e-15
Time:	09:35:40	Log-Likelihood:	-54.440
No. Observations:	49	AIC:	112.9
Df Residuals:	47	BIC:	116.7
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.3599	0.297	14.657	0.000	3.762	4.958
logn	-0.9740	0.083	-11.763	0.000	-1.141	-0.807

Omnibus:	5.932	Durbin-Watson:	1.172
Prob(Omnibus):	0.051	Jarque-Bera (JB):	4.773
Skew:	0.668	Prob(JB):	0.0919
Kurtosis:	3.745	Cond. No.	10.6

べキ分布の特徴は、中央値・最頻値が分布の左端に位置します。平均や分散という概念が事実上意味をなさないという点で、正規分布とは異なります。また、べキ分布はどの尺度で拡大・縮小しても、常に同じような分布になるという「スケールの不変性」があります。となると、妖怪画を説明するための文字表現としての主題キーワードの使い方にはべキ乗則でできあがっているのかもしれません。また、妖怪画の多様性には、限界がないことを示しているのかもしれません。

ノック 27 本目

主題キーワードに「天狗」が入っている妖怪画の資源識別子をリストアップして、「天狗」を含む用語を列挙しよう。

RK27.py

```
import re  # 正規表現を扱うためのライブラリをインポート

import pandas as pd  # データ操作のためのライブラリをインポート

# CSV ファイルを読み込む（エンコーディングを指定）
dataYokai = pd.read_csv('DataYokai.csv', header=0,
index_col=0, encoding='shift-jis')

# 列名を取得する
Yokailabel = dataYokai.columns
print(Yokailabel)

# "天狗"を含む列名を検索する
targetcol = [col for col in Yokailabel if re.search("天狗",
col)]

# 結果を表示する
print(targetcol)
```

dataYokai には、行が各々の妖怪画と対応し、列が主題キーワードとなっている。例えば、資源識別子「A5_hermitage_0008_0001_0000」の妖怪画については、不動明王の列が 1 となっているため、この妖怪画はこの主題キーワードで説明がされていることを示している。

	不動明王_フドウミョウ_	不動明王_フドウミョウオウ_	乙姫_オトヒメ_	九尾_キュウビ_	九尾狐_キュウビノキツネ_	九頭竜_クズリュウ_	九頭竜権現_クズリュウゴンゲン_	九頭龍_クズリュウ_	亀_カメ_	事触れ_コトブレ_	五徳_ゴトク_	五輪塔_ゴリントウ_	五重塔_ゴジュウノトウ_
A5_hermitage_0001_0001_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0001_0002_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0001_0003_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0003_0001_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0005_0001_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0006_0001_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0007_0001_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0007_0002_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0007_0003_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0008_0001_0000	0	1	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0008_0002_0000	0	0	0	0	0	0	0	0	0	0	0	0	0
A5_hermitage_0008_0003_0000	0	0	0	0	0	0	0	0	0	0	0	0	0

Yokailabel には、列名が格納されている。

```
> print(Yokailabel)
Index(['うさい角(ウサイカク)', 'うわん(ウワン)', 'おさん狐(オサンキツネ)', 'おにぎり(オニギリ)', 'お
化け(オバケ)',
       'お多福(オタフク)', 'お岩(オイワ)', 'お歯黒(オハグロ)', 'お爺さん(オジイサン)', 'お菊(オキク)
',
       ...
       '鼓(ツヅミ)', '鼠(ネズミ)', '鼻(ハナ)', '鼻毛(ハナゲ)', '鼻高(ハナダカ)', '龍(リュウ)',
       '龍女(リュウジョ)', '龍女(リュウニョ)', '龍王(リュウオウ)', '龍神(リュウジン)'],
      dtype='object', length=872)
```

re.search("天狗", col)により、Yokailabel に格納されている、各々の文字列について「天狗」が含まれている列名を抜き出すことができます。その結果を targetcol に格納します。

```
> print(targetcol)
['大天狗(ダイテング)', '天狗(テング)', '烏天狗(カラステング)', '青天狗(アオテング)', '鞍馬天狗(クラマテング)']
```

ノック 28 本目

主題キーワードに「蛸」が入っている妖怪画の資源識別子をリストアップして、Google 検索してみよう。まず「蛸」が含まれる列名をピックアップし、その列名ごとに 1 となっている妖怪画の資源識別子を列挙してみましょう。

RK28.py

```
import re  # 正規表現を扱うためのライブラリをインポート

import pandas as pd  # データ操作のためのライブラリをインポート

# CSV ファイルを読み込む（エンコーディングを指定）
dataYokai = pd.read_csv('DataYokai.csv', header=0, index_col=0, encoding='shift-jis')

# 列名を取得する
Yokailabel = dataYokai.columns

# "蛸"を含む列名を検索する
targetcol = [col for col in Yokailabel if re.search("蛸", col)]

# 該当する列のみを抽出する
targetmat = dataYokai[targetcol]

# 行ごとの合計を計算し、0 でない行のみを抽出する
sumtc = targetmat.apply(sum, axis=1)
targetmat = targetmat[sumtc != 0]

# 抽出されたデータフレームの列数を取得する
nf = targetmat.shape[1]

# リスト uu を初期化する
uu = []

# 各列について処理を行う
```

```
for i in range(nf):
    TFget = targetmat.iloc[:, i] != 0
    rnames = targetmat.index[TFget].tolist()
    uu.append([targetmat.columns[i]] + rnames)
    print(uu[i])
```

出力結果

['大蛸(オオダコ)', 'U426_nichibunken_0140_0001_0000']
['蛸(タコ)', 'A5_pushkin_0018_0006_0000', 'A5_pushkin_0041_0001_0000',
'A5_pushkin_0041_0024_0000', 'U426_nichibunken_0053_0021_0000',
'U426_nichibunken_0053_0021_0004', 'U426_nichibunken_0073_0025_0000',
'U426_nichibunken_0092_0003_0003', 'U426_nichibunken_0094_0006_0000',
'U426_nichibunken_0094_0006_0001', 'U426_nichibunken_0097_0010_0000',
'U426_nichibunken_0097_0021_0000', 'U426_nichibunken_0104_0006_0000',
'U426_nichibunken_0107_0008_0006', 'U426_nichibunken_0108_0001_0005',
'U426_nichibunken_0108_0002_0004', 'U426_nichibunken_0123_0003_0002',
'U426_nichibunken_0140_0001_0000', 'U426_nichibunken_0142_0010_0000',
'U426_nichibunken_0227_0009_0000', 'U426_nichibunken_0228_0010_0000',
'U426_nichibunken_0230_0001_0000', 'U426_nichibunken_0230_0010_0000',
'U426_nichibunken_0254_0008_0000', 'U426_nichibunken_0321_0003_0000',
'U426_nichibunken_0423_0001_0033']

このようにして得られた資源識別子をググると「蛸」の妖怪に到達できます。やってみよう!

ノック 29 本目

ノック 25 本目の結果を参考に、主題キーワードの好きな文字列により妖怪画の資源識別子をリストアップして、Google 検索してみよう。

ノック 30 本目

主題キーワードに「天狗」を含んでいる妖怪画について、主題キーワードの包含関係による使用の類似性を検討してみよう。

RK30.py

```
import re  # 正規表現を扱うためのライブラリをインポート

import japanize_matplotlib  # 日本語を表示するためのライブラリをインポート

```

```
import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
import numpy as np  # 数値計算のためのライブラリをインポート
import pandas as pd  # データ操作のためのライブラリをインポート
from scipy.cluster.hierarchy import linkage, dendrogram  # クラスタリングをするためのライブラリをインポート
from scipy.spatial.distance import pdist  # 距離行列を計算するライブラリをインポート

# CSV ファイルを読み込む（エンコーディングを指定）
dataYokai = pd.read_csv('DataYokai.csv', header=0, index_col=0, encoding='shift-jis')

# 列名を取得する
Yokailabel = dataYokai.columns

# "天狗"を含む列名を検索する
targetcol = [col for col in Yokailabel if re.search("天狗", col)]

# 該当する列のみを抽出する
targetmat = dataYokai[targetcol]

# 行ごとの合計を計算し、0 でない行のみを抽出する
sumtc = targetmat.apply(sum, axis=1)
targetmat = targetmat[sumtc != 0]

# 行列を転置する
DataS = targetmat.T

# 各行の合計を計算し、新しい行名を作成する
nn = DataS.apply(sum, axis=1)
newL = [f"{rowname}{n}" for rowname, n in zip(DataS.index, nn)]
DataS.index = newL

```

```
35 # Simpson 距離を計算する
36 def simpson_dist(u, v):
37     return 1 - np.dot(u, v) / min(u.sum(),v.sum())
38
39 # 距離行列を計算する
40 DD = pdist(DataS, metric=simpson_dist)
41
42 # 階層型クラスタリングを実行する
43 hclustName = 'ward'
44 Z = linkage(DD, method=hclustName)
45
46 # デンドログラムをプロットする
47 plt.figure(figsize=(10, 7))
48 dendrogram(Z, labels=DataS.index, leaf_rotation=90)
49 plt.title(f"dist type=Simpson clust type={hclustName}")
50 plt.show()
```

1-9 行目は、RK29.25 と同じです。targetmat は、行には妖怪画の資源識別子、列には「天狗」が入った主題キーワードからなる行列となっています。

	大天狗. ダイテング.	天狗. テング.	烏天狗. カラステング.	青天狗. アオテング.	鞍馬天狗. クラマテング.
A5_hermitage_0007_0001_0000	0	0	1	0	0
A5_pushkin_0005_0001_0000	0	1	0	0	0
…					
…					
U426_nichibunken_0442_0036_0000	0	1	0	0	0

ここで、天狗に関わる妖怪画は、88 作品ありました。
これを転置（行と列をひっくりかえす）すると、下の表になります。

	A5_hermitage_0007_0001_0000	A5_pushkin_0005_0001_0000	…	…	U426_nichibunken_0442_0036_0000
大天狗. ダイテング.	0	0	…	…	0
天狗. テング.	0	1	…	…	1
烏天狗. カラステング.	1	0	…	…	0
青天狗. アオテング.	0	0	…	…	0
鞍馬天狗.	0	0	…	…	0

クラマテング.					

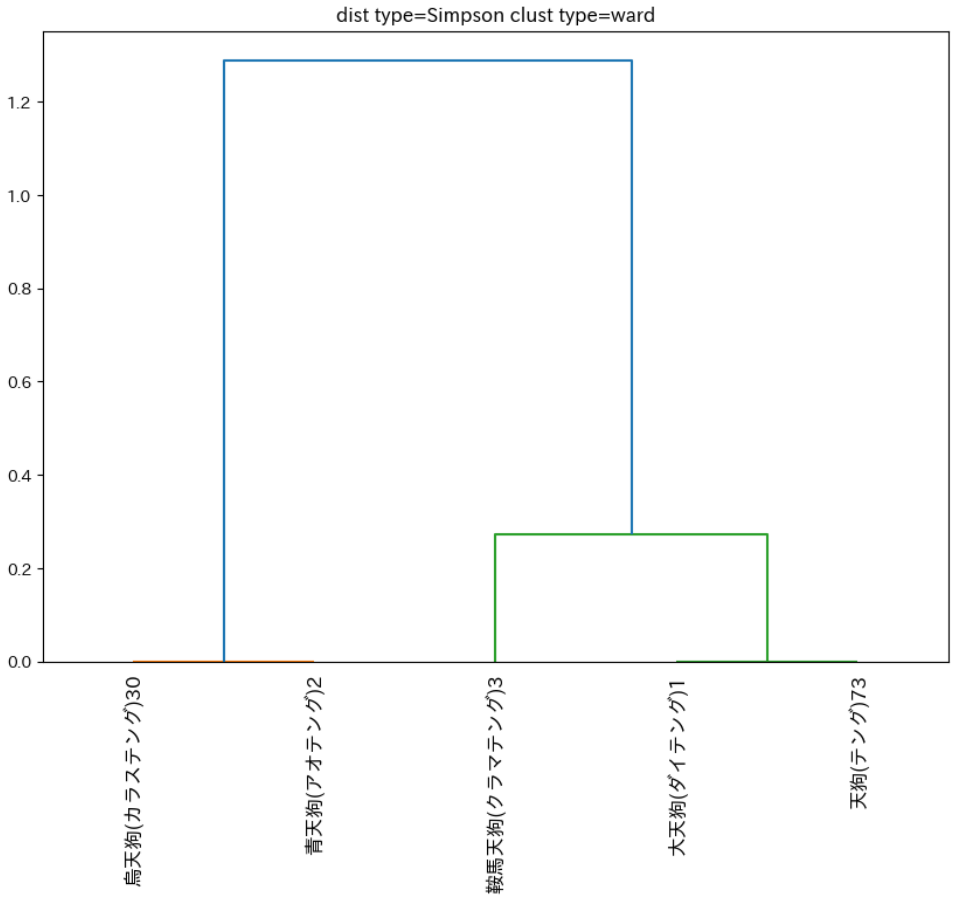
主題キーワード間の類似性として、ここでは、Simpson 係数（Overlap 係数）で表しました。二つの主題キーワード A, B について、A と B を使われている妖怪画の総数をそれぞれN_AとN_Bととします。また A、B それぞれの両方のキーワードが用いられている妖怪画の総数を$N_{A\cap B}$とします。

$$overlap係数(A,B) = \frac{N_{A\cap B}}{\min(N_A, N_B)}$$

二つの主題キーワード A, B の使用頻度（N_AとN_B）の影響を受けにくいからですが、他の距離係数を用いることも可能です。

怪異・妖怪画像データベースには、天狗を含む主題キーワードとして 5 種類あり、青天狗は烏天狗に含まれ、一方、鞍馬天狗、大天狗は天狗に含まれるという傾向がみられます。つまり、妖怪画を説明するための主題キーワードの使い方には大きく二つのグループに従って説明されていることがわかります。

（烏天狗と青天狗）（鞍馬天狗、大天狗、天狗）はそれぞれ共通に妖怪画の説明に用いられ



ていることがわかります。

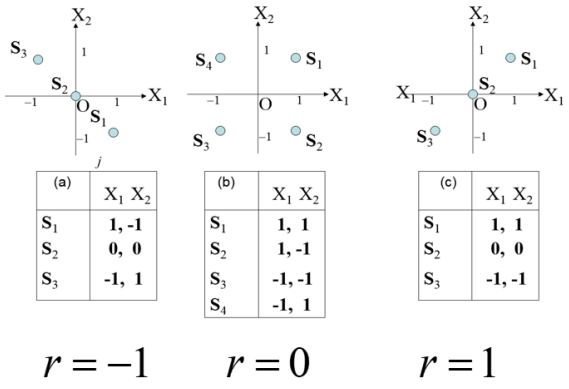
4. 関係性とは [数学 I]

「関係がある」というのを表す指標に、ピアソン相関係数があります。一般には相関係数と呼ばれています。以下の行列について二つの列 j と j' (変数 j と j') の相関係数を求めます。相関係数 $r_{jj'}$ の式をみると、分母は、j と j' 番目の列それぞれについて、平均値との差の 2 乗の和をルートを取った形になっています。一方、分子は、列ごとに j 番目と j' 番目の平均値との差をとり掛け合わせた合計となっています。二つの変数間の関係をプロットした場合に、このピアソン相関では、右下がりの直線上にプロットされると −1 、右上がりにプロットされると 1 となります。また、$r = 0$ のときには二つの変数間に関係がないことがわかります。ピアソン相関係数を活用してデータ間の相関をみてみましょう。

$$\begin{pmatrix} x_{11} & x_{11} & \cdots & x_{1j} & \cdots & x_{1j'} & \cdots & \cdots & x_{1M} \\ x_{21} & x_{21} & \cdots & x_{2j} & \cdots & x_{2j'} & \cdots & \cdots & x_{2M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & \cdots & x_{ij} & \cdots & x_{ij'} & \cdots & \cdots & x_{iM} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{N1} & x_{N1} & \cdots & x_{Nj} & \cdots & x_{Nj'} & \cdots & \cdots & x_{NM} \end{pmatrix}$$

ピアソンの相関係数

$$r_{jj'} = \frac{\sum_{i=1}^{N}(x_{ij} - \bar{x}_j)(x_{ij'} - \bar{x}_{j'})}{\sqrt{\sum_{i=1}^{N}(x_{ij} - \bar{x}_j)^2 \sum_{i=1}^{N}(x_{i'j} - \bar{x}_{j'})^2}} \qquad \bar{x}_j = \frac{\sum_{i=1}^{N} x_{ij}}{N}$$



ノック 31 本目

2007 年から 2018 年(Year)のそれぞれの年に、最低気温が 25℃以上（熱帯夜）であった日数を Tokyo、Osaka、Nagoya で集計した。Tokyo と Osaka, Tokyo と Nagoya、Osaka と Nagoya の間に相関があるだろうか。

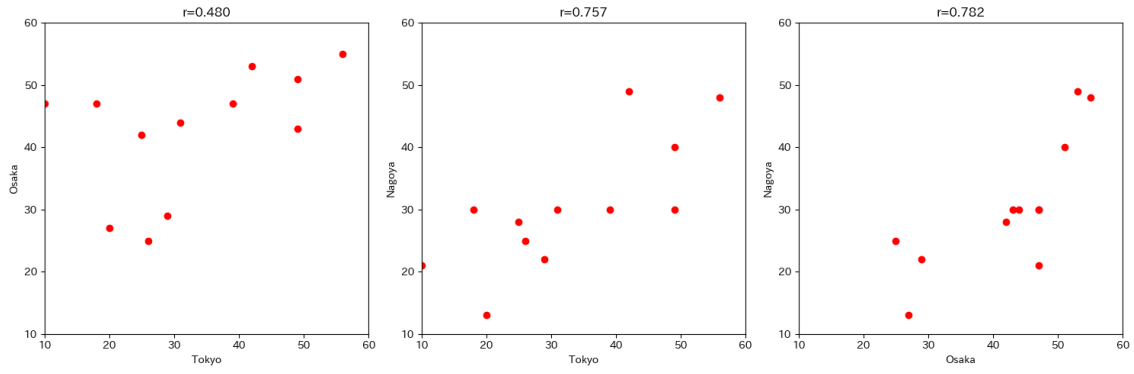
Year	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
Tokyo	31	25	20	56	49	49	39	29	26	10	18	42
Osaka	44	42	27	55	51	43	47	29	25	47	47	53
Nagoya	30	28	13	48	40	30	30	22	25	21	30	49

RK31.py

```
1 import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2 import numpy as np  # 数値計算のためのライブラリをインポート
3 import pandas as pd  # データ操作のためのライブラリをインポート
```

```
4
5  # データを作成する
6  Year = [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
7  Tokyo = [31, 25, 20, 56, 49, 49, 39, 29, 26, 10, 18, 42]
8  Osaka = [44, 42, 27, 55, 51, 43, 47, 29, 25, 47, 47, 53]
9  Nagoya = [30, 28, 13, 48, 40, 30, 30, 22, 25, 21, 30, 49]
10
11 # データフレームを作成する
12 dataSet = pd.DataFrame({'Year': Year, 'Tokyo': Tokyo,
   'Osaka': Osaka, 'Nagoya': Nagoya})
13
14 # 相関係数を計算する
15 TO = "r=" + str(np.corrcoef(dataSet['Tokyo'],
   dataSet['Osaka'])[0, 1])[:5]
16 TN = "r=" + str(np.corrcoef(dataSet['Tokyo'],
   dataSet['Nagoya'])[0, 1])[:5]
17 ON = "r=" + str(np.corrcoef(dataSet['Osaka'],
   dataSet['Nagoya'])[0, 1])[:5]
18
19 # プロットを 3 つ並べて表示する設定
20 fig, axs = plt.subplots(1, 3, figsize=(15, 5))
21
22 # 東京と大阪の散布図
23 axs[0].scatter(dataSet['Tokyo'], dataSet['Osaka'],
   color='red')
24 axs[0].set_xlabel('Tokyo')
25 axs[0].set_ylabel('Osaka')
26 axs[0].set_xlim(10, 60)
27 axs[0].set_ylim(10, 60)
28 axs[0].set_title(TO)
29
30 # 東京と名古屋の散布図
31 axs[1].scatter(dataSet['Tokyo'], dataSet['Nagoya'],
   color='red')
32 axs[1].set_xlabel('Tokyo')
33 axs[1].set_ylabel('Nagoya')
```

```
34  axs[1].set_xlim(10, 60)
35  axs[1].set_ylim(10, 60)
36  axs[1].set_title(TN)
37
38  # 大阪と名古屋の散布図
39  axs[2].scatter(dataSet['Osaka'], dataSet['Nagoya'],
    color='red')
40  axs[2].set_xlabel('Osaka')
41  axs[2].set_ylabel('Nagoya')
42  axs[2].set_xlim(10, 60)
43  axs[2].set_ylim(10, 60)
44  axs[2].set_title(ON)
45
46  # グラフを表示する
47  plt.tight_layout()
48  plt.show()
49
50  # データセットの相関行列を表示する
51  print(dataSet.corr())
```




```
> print(dataSet.corr())
             Year      Tokyo      Osaka     Nagoya
Year     1.00000000 -0.2061740 0.08951652 0.1350897
Tokyo   -0.20617398  1.0000000 0.48094702 0.7571719
Osaka    0.08951652  0.4809470 1.00000000 0.7828882
Nagoya   0.13508968  0.7571719 0.78288818 1.0000000
```

ノック 32 本目

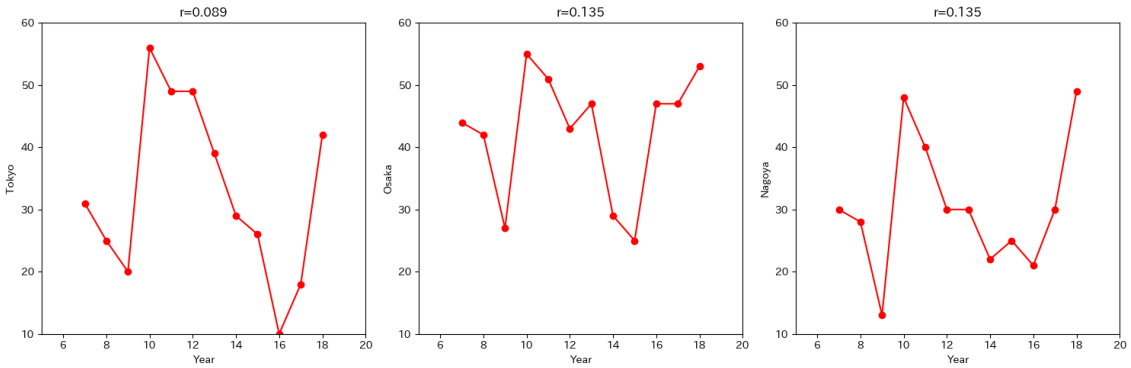
2007 年から 2018 年(Year)のそれぞれの年に、最低気温が 25℃以上（熱帯夜）であった日数を Tokyo、Osaka、Nagoya で集計した。年度と Tokyo、年度と Osaka，年度と Tokyo の間に相関があるだろうか。

Year	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
Tokyo	31	25	20	56	49	49	39	29	26	10	18	42
Osaka	44	42	27	55	51	43	47	29	25	47	47	53
Nagoya	30	28	13	48	40	30	30	22	25	21	30	49

RK32.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3  import pandas as pd  # データ操作のためのライブラリをインポート
4
5  # データを作成する
6  Year = [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
7  Tokyo = [31, 25, 20, 56, 49, 49, 39, 29, 26, 10, 18, 42]
8  Osaka = [44, 42, 27, 55, 51, 43, 47, 29, 25, 47, 47, 53]
9  Nagoya = [30, 28, 13, 48, 40, 30, 30, 22, 25, 21, 30, 49]
10
11 # データフレームを作成する
12 dataSet = pd.DataFrame({'Year': Year, 'Tokyo': Tokyo,
   'Osaka': Osaka, 'Nagoya': Nagoya})
13
14 # 相関係数を計算する
15 TO = "r=" + str(np.corrcoef(dataSet['Year'],
   dataSet['Tokyo'])[0, 1])[:5]
16 TN = "r=" + str(np.corrcoef(dataSet['Year'],
   dataSet['Osaka'])[0, 1])[:5]
17 ON = "r=" + str(np.corrcoef(dataSet['Year'],
   dataSet['Nagoya'])[0, 1])[:5]
18
19 # プロットを 3 つ並べて表示する設定
20 fig, axs = plt.subplots(1, 3, figsize=(15, 5))
21
```

```
# 年と東京の散布図
axs[0].plot(dataSet['Year'], dataSet['Tokyo'], '-o',
color='red')
axs[0].set_xlabel('Year')
axs[0].set_ylabel('Tokyo')
axs[0].set_xlim(5, 20)
axs[0].set_ylim(10, 60)
axs[0].set_title(TN)

# 年と大阪の散布図
axs[1].plot(dataSet['Year'], dataSet['Osaka'], '-o',
color='red')
axs[1].set_xlabel('Year')
axs[1].set_ylabel('Osaka')
axs[1].set_xlim(5, 20)
axs[1].set_ylim(10, 60)
axs[1].set_title(ON)

# 年と名古屋の散布図
axs[2].plot(dataSet['Year'], dataSet['Nagoya'], '-o',
color='red')
axs[2].set_xlabel('Year')
axs[2].set_ylabel('Nagoya')
axs[2].set_xlim(5, 20)
axs[2].set_ylim(10, 60)
axs[2].set_title(ON)

# グラフを表示する
plt.tight_layout()
plt.show()

# データセットの相関行列を表示する
print(dataSet.corr())
```



以下の結果より、Year と 3 つの地域(Tokyo，Osaka，Nagoya)の相関係数は低いことがわかる。熱帯夜の日数が増えるか年とともに増えるかというとそうでもない。

```
> print(dataSet.corr())
              Year      Tokyo      Osaka     Nagoya
Year     1.00000000 -0.2061740 0.08951652 0.1350897
Tokyo   -0.20617398  1.0000000 0.48094702 0.7571719
Osaka    0.08951652  0.4809470 1.00000000 0.7828882
Nagoya   0.13508968  0.7571719 0.78288818 1.0000000
```

単回帰モデル：対数変換の面白さ [数学 I，[数学 B]

$100000 = 10^5$

について底を 10 として対数をとると

$\log_{10}100000 = 5$

となる。なんてことはない、100000 の桁数を表したことと変わりはない。

ところが、仮に

$x^n x^m = 5$

が成り立っているとしよう。この関係を見つけるのはグラフに書くとちょっと面倒だ。

ところが両辺、対数をとってみよう。

$\log_{10}(x^n y^m) = \log_{10} 5$

これを変形すると、

$n\log_{10}x + m\log_{10}y = \log_{10} 5$

$\log_{10}x$ と $\log_{10}y$ の関係は線形になり、これらの関係の式をみつけるのが楽になる。

なお、Python では、底を 2 とする場合は、np.log2()、底を 10 とする場合は、np.log10()、自然対数は np.log() として計算できます。

ノック 33 本目

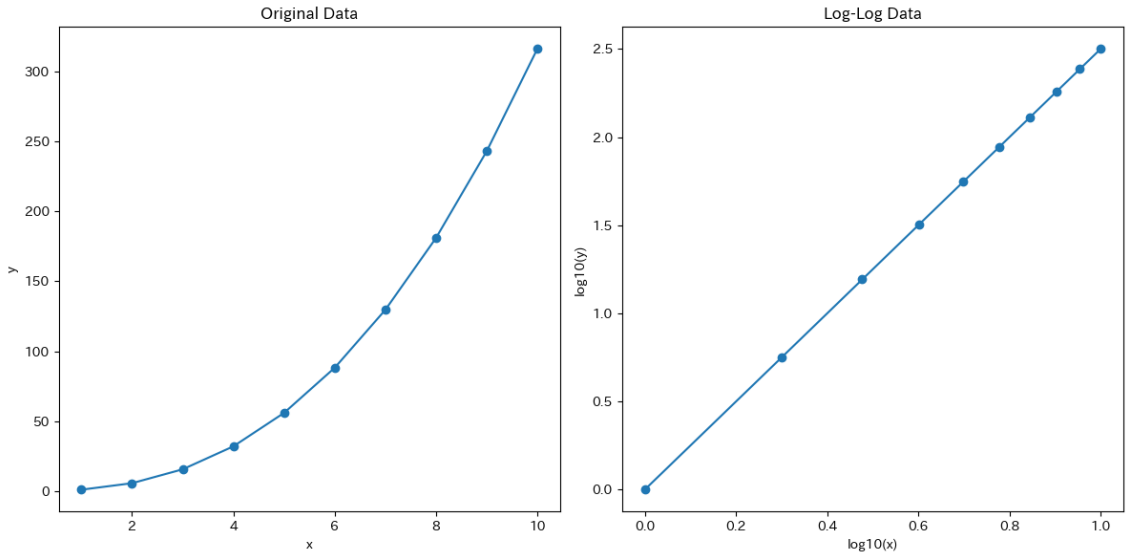
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1.00, 5.65, 15.58, 32.00, 55.90, 88.18, 129.64,181.01, 243.00, 316.22)
について $\log_{10}x$、$\log_{10}y$ と各変数を対数変換して、x と y の関係を式で表してみよう。

RK33.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3  import statsmodels.api as sm  # 線形回帰のためのライブラリをインポート
4
5  # データを作成する
6  x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
7  y = np.array([1.00, 5.65, 15.58, 32.00, 55.90, 88.18, 129.64,
   181.01, 243.00, 316.22])
8
9  # プロットを 2 つ並べて表示する設定
10 fig, axes = plt.subplots(1, 2, figsize=(12, 6))
11
12 # 元のデータをプロットする
```

```
13 axes[0].plot(x, y, 'o-')
14 axes[0].set_xlabel('x')
15 axes[0].set_ylabel('y')
16 axes[0].set_title('Original Data')
17
18 # 対数を取ったデータをプロットする
19 logx = np.log10(x)
20 logy = np.log10(y)
21 axes[1].plot(logx, logy, 'o-')
22 axes[1].set_xlabel('log10(x)')
23 axes[1].set_ylabel('log10(y)')
24 axes[1].set_title('Log-Log Data')
25
26 # グラフを表示する
27 plt.tight_layout()
28 plt.show()
29
30 # 線形回帰を実行する
31 X = sm.add_constant(logx)
32 model = sm.OLS(logy, X).fit()
33
34 # 結果を表示する
35 print(model.summary())
```

10-28 行目：x と y の関係、$\log_{10}x$ 、$\log_{10}y$ の関係を図に示しました。



この図をみると、$\log_{10}x$ と $\log_{10}y$ の間には線形の関係があることがわかります。

32 行目：

$$\log_{10}y = [\text{切片}] + [\text{傾き}] \cdot \log_{10}x$$

として、回帰モデルを作成した結果を以下に示す。この結果の見方を説明すると、赤字のところの(Intercept)が[切片] $= -0.0002$、$\log_{10}x$ の[傾き]が $\log x$ の[傾き]$= 2.5002$ です。そこで、これら係数の統計的に有意性は $\Pr(>|t|)$ に示されていています。切片 -0.0002 には $\Pr(>|t|) = 0.090$ であり、基準 p 値を 0.05 と設定すると、この p 値より大きいので、符号に意味はありません。つまり、0 とみなせます。一方、2.5002 は $\Pr(>|t|) = 0.000$ であり、$p = 0.05$ より小さいので、この係数は有意に正であるとなります。式全体は、分散分析により Prob (F-statistic): $< 2.2e-16$ となっているので、式は成り立っていることになります(Prob (F-statistic)の項目を見て、0.05 より小さければ OK と確認することだけ理解しましょう。)

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   logy   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 2.128e+08
Date:                Sat, 01 Jun 2024   Prob (F-statistic):           5.46e-31
Time:                        11:01:12   Log-Likelihood:                 74.102
No. Observations:                  10   AIC:                            -144.2
Df Residuals:                       8   BIC:                            -143.6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
```

```
              coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -0.0002      0.000     -1.926      0.090      -0.001     4.7e-05
logx          2.5002      0.000   1.46e+04      0.000       2.500       2.501
==============================================================================
Omnibus:                        7.414   Durbin-Watson:                   2.172
Prob(Omnibus):                  0.025   Jarque-Bera (JB):                2.688
Skew:                          -1.058   Prob(JB):                        0.261
Kurtosis:                       4.406   Cond. No.                         4.83
==============================================================================
```

そこで、おおよそ、

$$\log_{10} y = 0.000 + 2.500 \cdot \log_{10} x$$

となります。これをもとに、

$$\log_{10} y - 2.500 \log_{10} x = 0$$

$$\log_{10}\left(\frac{y}{x^{2.5}}\right) = 0$$

$$\frac{y}{x^{2.5}} = 10^0 = 1$$

結局

$$y = x^{2.5} = x^{5/2} = x^2\sqrt{x}$$

という関係が導かれました。

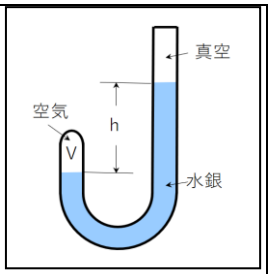
ノック 34 本目

x<-c(1.000,0.933,0.895,0.870,0.851,0.835,0.823,0.812)
y<-c(1.000,2.143,3.348,4.594,5.873,7.177,8.503,9.849)
について $\log_{10} x$、$\log_{10} y$と各変数を対数変換して、x と y の関係を式で表してみよう。

RK33.py を参考にプログラムをつくって x と y の関係を式で表してみよう。

気体の体積と圧力の関係 [理数探究基礎]

かつて 17 世紀にイギリスの科学者ボイルは、気体の体積と圧力の関係を調べる実験を行った。J 字状の管の先端に閉じ込めた空気の体積 V と、管に入れた水銀の量の関係を調べた。温度は一定である。液面の高さの差 h から閉じ込められた空気にかかる圧力 p を求めた(表)。この表をもとに p と V の関係を求めてみよう。



体積 V(cm^3)	48.0	40.0	32.0	24.0	20.0	16.0	12.0
圧力 p(x10^3hPa)	1.01	1.23	1.54	2.04	2.46	3.04	4.09

法則性を見出すときには、対数プロットが役に立つことがある。底を 10 として二つの変数 V と p の対数をとり、直線関係が得られた。

$$\log_{10}V = a + b\log_{10}p$$

この式は、

$$\log_{10}V + \log_{10}p^{-b} = a$$

となり、さらに変形すると

$$\log_{10}(Vp^{-b}) = a$$

$$Vp^{-b} = 10^a$$

と変形できるため、a と b が得られれば V と p の関係を式で表現できます。

(a)いま、x と y が比例関係にあるとき

$$y = x$$

対数をとると、傾き 1、切片(定数)0 の直線が得られます。

$$\log_{10}y = \log_{10}x$$
$$\log_{10}\frac{y}{x} = 0$$
$$\frac{y}{x} = 10^0 = 1$$

となる。

(b) $y = x^m$のとき

$$\log_{10}y = m\log_{10}x$$

となるため、y と x の傾きが m、切片（定数）は 0 となります。よって、傾きから m を求められます。

ノック 35 本目

手計算で上記の枠内式を誘導して、y と x の関係を式であらわしてみましょう。

(Python プログラムはありません)

ノック 36 本目

ノック 35 本目をもとに、

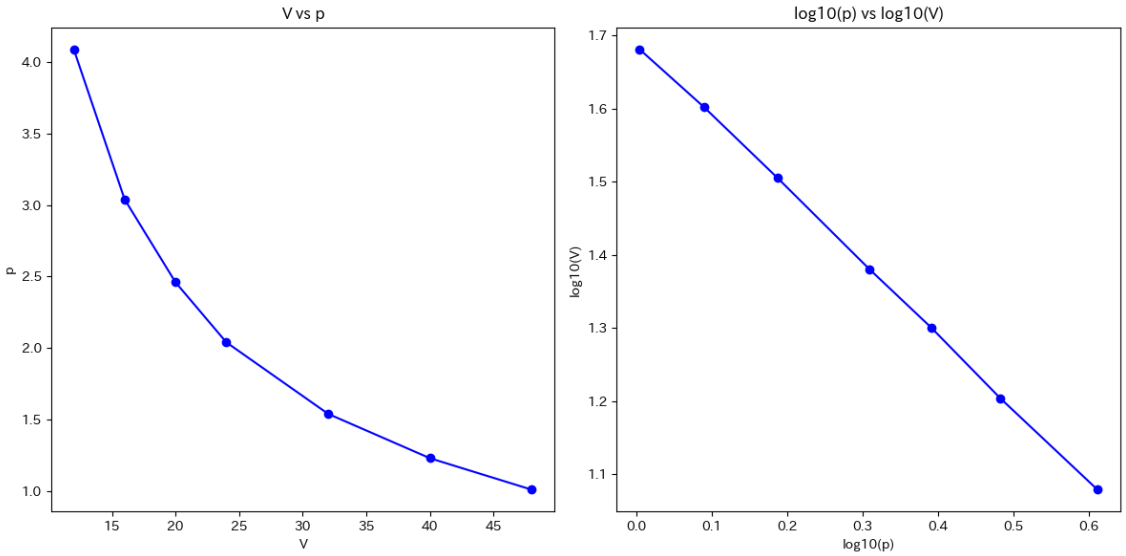
[1] V と p を、x 軸と y 軸に設定しプロットしてみましょう。

[2] $\log_{10}V$ と $\log_{10}p$ を、x 軸と y 軸に設定しプロットしてみましょう。

RK36.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # データを作成する
5  V = np.array([48.0, 40.0, 32.0, 24.0, 20.0, 16.0, 12.0])
6  p = np.array([1.01, 1.23, 1.54, 2.04, 2.46, 3.04, 4.09])
7
8  # プロットを 2 つ並べて表示する設定
9  fig, axes = plt.subplots(1, 2, figsize=(12, 6))
10
11 # 元のデータをプロットする
12 axes[0].plot(V, p, 'b-o')
13 axes[0].set_xlabel('V')
14 axes[0].set_ylabel('p')
15 axes[0].set_title('V vs p')
16
17 # 対数を取ったデータをプロットする
18 logp = np.log10(p)
19 logV = np.log10(V)
20 axes[1].plot(logp, logV, 'b-o')
21 axes[1].set_xlabel('log10(p)')
22 axes[1].set_ylabel('log10(V)')
23 axes[1].set_title('log10(p) vs log10(V)')
24
```

```
# グラフを表示する
plt.tight_layout()
plt.show()
```



左側の図が[1] V と p の関係、右側の図が [2] $\log_{10}V$ と $\log_{10}p$ の関係となります。この図をみていただくと、$\log_{10}V$ と $\log_{10}p$ の間には、

$$\log_{10}V = a - b\log_{10} p$$

という関係が成り立つ。ここで、a と b をどのように求めるのでしょうか。ここでよく使われるのが最小二乗法です。

最小二乗法 [数学 B]

いま、y と x の測定値があるとします。ここで、 y と x の間には、

$$y = a + bx \tag{1}$$

という関係があるとします。N 個の測定値 $y = \{y_1, y_2, \cdots, y_i, \cdots, y_N\}$ と $x = \{x_1, x_2, \cdots, x_i, \cdots, x_N\}$ を(1)に代入する。

$$y_1 = a + bx_1 + e_1$$
$$\cdots$$
$$y_i = a + bx_i + e_i$$
$$\cdots$$
$$y_N = a + bx_N + e_N$$

ここで、e_i は残差である。測定値 y_i と x_i がぴったり(1)の式に合うことはほぼないので残差を入れた式とする。では、この式を残差 e_i により式を変形してみましょう。

$$e_i = y_i - (a + bx_i)$$

ここで $i = 1, 2, 3, \cdots, N$ である。

そこで、全体の 2 乗誤差を定義する。

$$E(a, b) = {e_1}^2 + {e_2}^2 + \cdots + {e_N}^2 = \sum_{i=1}^{N} {e_i}^2$$

この式の $E(a, b)$ を最小とする a, b を求める。

この式に、$e_i = y_i - (a + bx_i)$ を代入してみましょう。

$$E(a, b) = \{y_1 - (a + bx_1)\}^2 + \cdots + \{y_i - (a + bx_i)\}^2 + \cdots + \{y_N - (a + bx_N)\}^2$$
$$= \sum_{i=1}^{N} \{y_i - (a + bx_i)\}^2$$

$E(a, b)$ を最小にする係数 a, b を求めてみましょう。最後の $\sum_{i=1}^{N} \{y_i - (a + bx_i)\}^2$ の項を見てみましょう。

$$E(a, b) = \sum_{i=1}^{N} \{y_i - (a + bx_i)\}^2 = \sum_{i=1}^{N} \{y_i - a - bx_i\}^2$$
$$= \sum_{i=1}^{N} \{{y_i}^2 + a^2 + (bx_i)^2 - 2ay_i - 2bx_iy_i + 2abx_i\}$$
$$= \sum_{i=1}^{N} y^2 + \sum_{i=1}^{N} a^2 + b^2 \sum_{i=1}^{N} {x_i}^2 - 2a \sum_{i=1}^{N} y_i - 2b \sum_{i=1}^{N} x_iy_i + 2ab \sum_{i=1}^{N} x_i$$

ここで、$\sum_{i=1}^{N} y^2$、$\sum_{i=1}^{N} y_i$、$\sum_{i=1}^{N} {x_i}^2$、$\sum_{i=1}^{N} x_i$、$\sum_{i=1}^{N} x_iy_i$ は実験値から計算できるので定数とみなせる。

$Y_2 = \sum_{i=1}^{N} y^2$、$Y_1 = \sum_{i=1}^{N} y_i$、$X_2 = \sum_{i=1}^{N} {x_i}^2$、$X_1 = \sum_{i=1}^{N} x_i$、$Z = \sum_{i=1}^{N} x_i y_i$

また、

$$\sum_{i=1}^{N} a^2 = a^2 + a^2 + \cdots + a^2 = Na^2$$

となる。

$$E(a,b) = Y_2 + Na^2 + b^2 X_2 - 2aY_1 - 2bZ_1 + 2abX_1$$

この式をみると、$E(a,b)$ は変数 a と b についての 2 次式となる。a^2 と b^2 の係数は N と $X_1 = \sum_{i=1}^{N} x_i$ はともに正なので最小値が存在する。これを簡単に求めるには、
$E(a,b)$ を片方の変数 b を固定して、a で微分した式を作り 0 とおく。これを偏微分といい、$\frac{\partial E(a,b)}{\partial a}$ とする。

$$\frac{\partial \mathrm{E(a,b)}}{\partial a} = 2Na - 2Y_1 + 2X_1 b = 0$$

同様に、変数 a を固定して、 b で微分した式を作り 0 とおく。

$$\frac{\partial E(a,b)}{\partial b} = 2X_2 b - 2Z_1 + 2X_1 a = 0$$

これら二つの式をもとに a, b を算出する。すると、

$$a = \frac{X_2 Y_1 - X_1 Z}{N X_2 - {X_1}^2} = \frac{\sum_{i=1}^{N} {x_i}^2 \sum_{i=1}^{N} y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i y_i}{N \sum_{i=1}^{N} {x_i}^2 - \left(\sum_{i=1}^{N} x_i\right)^2}$$

$$b = \frac{X_1 Y_1 - N Z_1}{N X_2 - {X_1}^2} = \frac{\sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i - N \sum_{i=1}^{N} x_i y_i}{N \sum_{i=1}^{N} {x_i}^2 - \left(\sum_{i=1}^{N} x_i\right)^2}$$

と求まります。

Python スクリプトにより a, b の係数を求めてみましょう。

$$\log_{10} V = a + b \log_{10} p$$

のように、$y = a + bx$ として a, b を求め、このモデルに従って、任意の x から y を予測するモデルのこと線形回帰モデル（linear regression model）といいます。Python プログラムの statsmodels ライブラリの sm.OLS()に x と y とデータセット指定すると簡単に求めることができます。sm.OLS()と式を誘導して求めた回帰係数（ a と b ）を比較してみよう。

ノック 37 本目

V<-c(48.0,40.0,32.0,24.0,20.0,16.0,12.0)
p<-c(1.01,1.23,1.54,2.04,2.46,3.04,4.09)
について$x = \log_{10} V$, $y = \log_{10} p$ と変換した後、

[1]sm.OLS()を用いて $y = a + bx$ としたときの回帰係数 a, b を求めよう。

[2]最小二乗法により導いた式

$$a = \frac{X_2 Y_1 - X_1 Z}{N X_2 - {X_1}^2} = \frac{\sum_{i=1}^{N} {x_i}^2 \sum_{i=1}^{N} y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i y_i}{N \sum_{i=1}^{N} {x_i}^2 - \left(\sum_{i=1}^{N} x_i\right)^2}$$

$$b = \frac{X_1 Y_1 - N Z_1}{N X_2 - {X_1}^2} = \frac{\sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i - N \sum_{i=1}^{N} x_i y_i}{N \sum_{i=1}^{N} {x_i}^2 - \left(\sum_{i=1}^{N} x_i\right)^2}$$

を用いて a, b を求めてみよう。

RK37.py

```python
import numpy as np  # 数値計算のためのライブラリをインポート
import statsmodels.api as sm  # 線形回帰のためのライブラリをインポート

# データを作成する
V = np.array([48.0, 40.0, 32.0, 24.0, 20.0, 16.0, 12.0])
p = np.array([1.01, 1.23, 1.54, 2.04, 2.46, 3.04, 4.09])
logp = np.log10(p)
logV = np.log10(V)

# 回帰モデル
X = sm.add_constant(logV)  # 切片項を追加
model = sm.OLS(logp, X).fit()
print(model.summary())
conf_int = model.conf_int(alpha=0.05)
print("95% Confidence Intervals:")
print(conf_int)

a1 = model.params[0]
b1 = model.params[1]
print("a1:", a1)
print("b1:", b1)
print("10^a1:", 10**a1)
```

```
24  # 手計算による導出
25  N = len(V)
26  a2 = (np.sum(logV**2) * np.sum(logp) - np.sum(logV) *
    np.sum(logp * logV)) / (N * np.sum(logV**2) -
    (np.sum(logV)**2))
27  b2 = (N * np.sum(logV * logp) - np.sum(logV) * np.sum(logp))
    / (N * np.sum(logV**2) - (np.sum(logV)**2))
28
29  print("a2:", a2)
30  print("b2:", b2)
31  print("10^a2:", 10**a2)
```

5-8 行目：logp には $\log_{10} p$ 、logV には $\log_{10} V$ の値が格納されています。

7-10 行目：

logp を y (目的変数)、logV を x (説明変数)として、回帰モデルを作るスクリプトは、

model = sm.OLS(logp, X).fit()

となる。model の中を見てみましょう。sm.OLS() の左側が目的変数、右側が説明変数となる。ここでは、$\log p = a + b \log V$ という回帰モデルをつくる。得られた回帰モデルは model に格納される。

print(model.summary())

により、回帰モデルを得ることができる。また

print(conf_int)

により、係数 a, b の 95%信頼区間が得らえる。

結果を見てみましょう。residuals はそれぞれのサンプルの残差を表している。また、coefficients から、

$$\log p = 1.6923 - 1.0016 \log V$$

というモデルができたことがわかります。ここで、係数の右側にある***は係数の統計有意水準を表す。***とは p<0.001 の統計有意性を示している（とってもいいモデルができた）。

F-statistic とは回帰モデル全体における統計有意性を示す指標である。これも p-value: 1.003e-10 と非常に小さい p 値なので $\log p$ と$\log V$ には、線形回帰モデルとして関連づいていることがわかる。

```
> print(model.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                   logp   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                3.241e+04
Date:                Sat, 01 Jun 2024   Prob (F-statistic):          1.00e-10
Time:                        11:01:13   Log-Likelihood:                 32.014
```

```
No. Observations:                 7   AIC:                          -60.03
Df Residuals:                     5   BIC:                          -60.14
Df Model:                         1
Covariance Type:          nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.6923      0.008    216.082      0.000       1.672       1.712
logV          -1.0016      0.006   -180.038      0.000      -1.016      -0.987
==============================================================================
Omnibus:                        nan   Durbin-Watson:                   2.127
Prob(Omnibus):                  nan   Jarque-Bera (JB):                0.774
Skew:                        -0.508   Prob(JB):                        0.679
Kurtosis:                     1.728   Cond. No.                         14.8
==============================================================================
```

つづいて、confint(reg,level=0.95)により係数 a, b の区間推定を行ってみましょう。95%信頼区間において、

切片 a は、$1.672143 < a < 1.7124062$

係数 b は $-1.015944 < b < -0.9873409$

となった。注目すべきは、係数 b は -1 とみなせるという点です。

```
> print(conf_int)
 95% Confidence Intervals:
             0         1
const  1.672143  1.712406
logV  -1.015944 -0.987341
a1: 1.6922744015429625
b1: -1.0016423434722865
10^a1: 49.23505206842694
a2: 1.6922744015429712
b2: -1.0016423434722912
10^a2: 49.23505206842792
```

なお[1] sm.OLS()に x と y 求めた係数が a_1 (切片）、b_1 (傾き)、[2] a と b の値を求める式を誘導した場合の係数が a_2 (切片）、b_2 (傾き)であり、確かに、$a_1 = a_2, b_1 = b_2$ となる。

それでは、

$$\log p = 1.6923 - 1.0016 \log V$$

の式から p と V の関係を導いてみましょう。

$$\log_{10} p = 1.6923 - 1.0016 \log_{10} V$$

$$\log_{10} p + \log_{10} V^{1.0016} = 1.6923$$

$$\log_{10} pV^{1.0016} = 1.6923$$

$$pV^{1.0016} = 10^{1.6923} = 49.238$$

いま、係数 1.0016 は 95%信頼区間で 1 とみなせるので、おおよそ、この実験では、

$$PV = 49.238 (= 一定)$$

という結果が得られた。多分、化学の授業で気体の状態方程式として勉強されることと思います。

ノック 38 本目

2007 年から 2018 年(Year)のそれぞれの年に、最低気温が 25℃以上（熱帯夜）であった日数を Tokyo、Osaka、Nagoya で集計した。以下の二つの回帰モデルを作成しよう。

$$(熱帯夜の日数)_{\mathrm{Nagoya}} = a_0 + a_1 (熱帯夜の日数)_{\mathrm{Osaka}}$$

$$(熱帯夜の日数)_{\mathrm{Nagoya}} = a_0 + a_1 (熱帯夜の日数)_{\mathrm{Tokyo}}$$

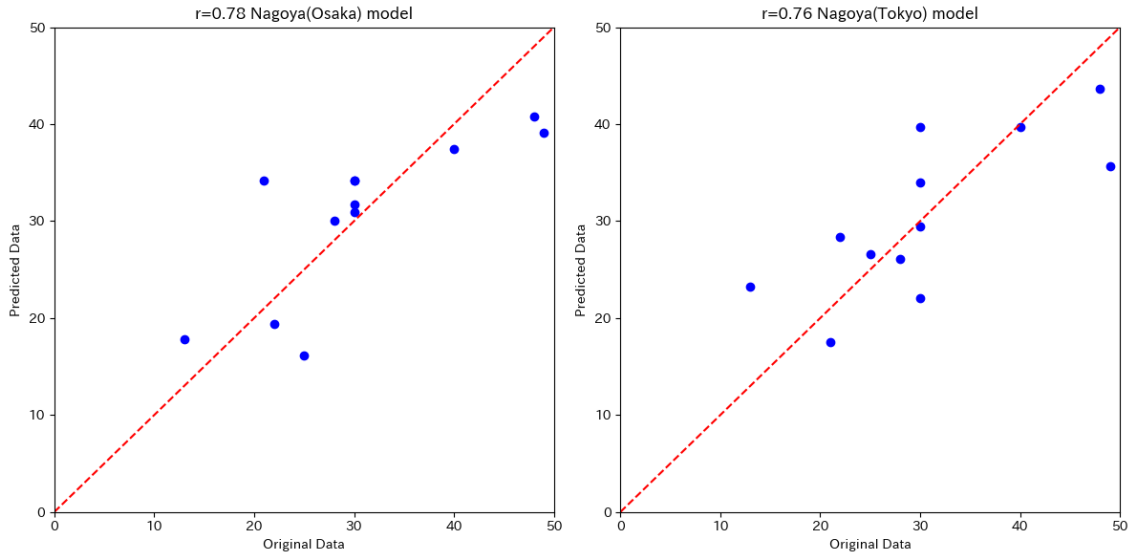
Year	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
Tokyo	31	25	20	56	49	49	39	29	26	10	18	42
Osaka	44	42	27	55	51	43	47	29	25	47	47	53
Nagoya	30	28	13	48	40	30	30	22	25	21	30	49

RK38.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3  import statsmodels.api as sm  # 線形回帰のためのライブラリをインポート
4
5  # データを作成する
6  Year = [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
7  Tokyo = [31, 25, 20, 56, 49, 49, 39, 29, 26, 10, 18, 42]
8  Osaka = [44, 42, 27, 55, 51, 43, 47, 29, 25, 47, 47, 53]
```

```
Nagoya = [30, 28, 13, 48, 40, 30, 30, 22, 25, 21, 30, 49]

# 大阪を説明変数、名古屋を目的変数とした線形回帰モデルを作成する
model_NO = sm.OLS(Nagoya, sm.add_constant(Osaka)).fit()
print(model_NO.summary())
print("95% Confidence Intervals (Nagoya ~ Osaka):")
print(model_NO.conf_int(alpha=0.05))

# 東京を説明変数、名古屋を目的変数とした線形回帰モデルを作成する
model_NT = sm.OLS(Nagoya, sm.add_constant(Tokyo)).fit()
print(model_NT.summary())
print("95% Confidence Intervals (Nagoya ~ Tokyo):")
print(model_NT.conf_int(alpha=0.05))

# 相関係数を計算する
cor_NO = np.corrcoef(Nagoya, model_NO.fittedvalues)[0, 1]
NO = f"r={cor_NO:.2f} Nagoya(Osaka) model"

cor_NT = np.corrcoef(Nagoya, model_NT.fittedvalues)[0, 1]
NT = f"r={cor_NT:.2f} Nagoya(Tokyo) model"

# プロットを 2 つ並べて表示する設定
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

# 名古屋の実際の値と大阪を説明変数としたモデルの予測値をプロットする
axs[0].scatter(Nagoya, model_NO.fittedvalues, color='blue')
axs[0].plot([0, 50], [0, 50], color='red', linestyle='--')
axs[0].set_xlim(0, 50)
axs[0].set_ylim(0, 50)
axs[0].set_xlabel('Original Data')
axs[0].set_ylabel('Predicted Data')
axs[0].set_title(NO)

# 名古屋の実際の値と東京を説明変数としたモデルの予測値をプロットする
axs[1].scatter(Nagoya, model_NT.fittedvalues, color='blue')
axs[1].plot([0, 50], [0, 50], color='red', linestyle='--')
```

```
axs[1].set_xlim(0, 50)
axs[1].set_ylim(0, 50)
axs[1].set_xlabel('Original Data')
axs[1].set_ylabel('Predicted Data')
axs[1].set_title(NT)

# グラフを表示する
plt.tight_layout()
plt.show()
```




```
> print(model_NO.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Nagoya   R-squared:                       0.613
Model:                            OLS   Adj. R-squared:                  0.574
Method:                 Least Squares   F-statistic:                     15.83
Date:                Sat, 01 Jun 2024   Prob (F-statistic):           0.00260
Time:                        11:01:13   Log-Likelihood:                -39.185
No. Observations:                  12   AIC:                             82.37
Df Residuals:                      10   BIC:                             83.34
Df Model:                           1
Covariance Type:            nonrobust
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -4.4094      8.999     -0.490      0.635     -24.460      15.642
Osaka         0.8214      0.206      3.979      0.003       0.361       1.281
==============================================================================
Omnibus:                        0.171   Durbin-Watson:                   2.174
Prob(Omnibus):                  0.918   Jarque-Bera (JB):                0.139
Skew:                          -0.162   Prob(JB):                        0.933
Kurtosis:                       2.585   Cond. No.                         196.
==============================================================================
> print(model_NO.conf_int(alpha=0.05))
                0          1
const -24.46024814  15.64150367
Osaka   0.36145887   1.28133512
```

[1] $(\text{熱帯夜の日数})_{\text{Nagoya}} = a_0 + a_1(\text{熱帯夜の日数})_{\text{Osaka}}$

model_NO.summary()による a_0 と a_1 により

$(\text{熱帯夜の日数})_{\text{Nagoya}} = -4.4094 + 0.8214 \text{ x } (\text{熱帯夜の日数})_{\text{Osaka}}$

という関係が得られました。

F-statistic: 15.83 on 1 and 10 DF, p-value: 0.00260

により、F 統計量における p 値=0.00260 は、$p < 0.05$ であるので、ここで得られた回帰モデルは統計的に有意です。

続いて、それぞれの係数の t 統計量における p 値は、0.633 および 0.003 である。いま、$p < 0.05$ のとき、係数が統計的に有意だとすると、a_0 は統計的に有意でない、a_1 は統計的に有意である。つまり、$a_0 = -4.44094$ は 0 とみなせる。一方で $a_1 = -0.8214$ は 0 とみなせない。ということを意味している。

model_NO.conf_int(alpha=0.05)の結果をみると、95%区間推定において

$-24.460248 < a_0 < 15.641504$

$0.361459 < a_1 < 1.281335$

であり、a_0 の区間は 0 をまたいでいるので、0 と見做されます。一方、a_1 の区間は正と見做せます。同様に、$(\text{熱帯夜の日数})_{\text{Nagoya}} = a_0 + a_1(\text{熱帯夜の日数})_{\text{Tokyo}}$についても統計的に解釈してみてください。

```
> print(model_NT.summary())
```

```
OLS Regression Results
==============================================================================
Dep. Variable:                 Nagoya   R-squared:                       0.573
Model:                            OLS   Adj. R-squared:                  0.531
Method:                 Least Squares   F-statistic:                     13.44
Date:                Sat, 01 Jun 2024   Prob (F-statistic):           0.00435
Time:                        11:01:13   Log-Likelihood:                -39.769
No. Observations:                  12   AIC:                             83.54
Df Residuals:                      10   BIC:                             84.51
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         11.8560      5.504      2.154      0.057      -0.408      24.120
Tokyo          0.5678      0.155      3.666      0.004       0.223       0.913
==============================================================================
Omnibus:                        0.138   Durbin-Watson:                   1.153
Prob(Omnibus):                  0.933   Jarque-Bera (JB):                0.236
Skew:                           0.194   Prob(JB):                        0.889
Kurtosis:                       2.433   Cond. No.                         93.0
==============================================================================
```

```
> print(model_NT.conf_int(alpha=0.05))
```

```
                0          1
const -0.40826373  24.12035528
Tokyo  0.22267065   0.91300169
```

重回帰モデルを作成しよう

最小二乗法では、x と y のデータから

$y = a + bx$

の式を導いた。これを単回帰分析と言います。この単回帰分析をさらに発展させ、x_1 、x_2 , y のデータから

$y = a + b_1x_1 + b_2x_2$

という式を導くことができます。この場合、複数の説明変数 $(x1,\ x2)$ から目的変数 y を求める重回帰モデルといいます。熱帯夜のデータを用いて重回帰モデルを構築してみましょう。

ノック 39 本目

複数の説明変数、$(\text{熱帯夜の日数})_{\text{Osaka}}$、$(\text{熱帯夜の日数})_{\text{Tokyo}}$から$(\text{熱帯夜の日数})_{\text{Nagoya}}$を説明する回帰モデルをつくってみよう。

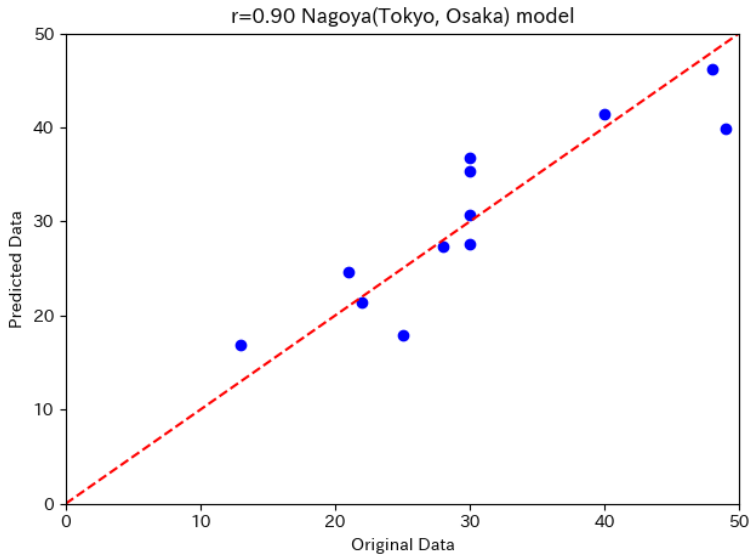
$$(\text{熱帯夜の日数})_{\text{Nagoya}} = a_0 + a_1(\text{熱帯夜の日数})_{\text{Osaka}} + a_2(\text{熱帯夜の日数})_{\text{Tokyo}}$$

Year	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
Tokyo	31	25	20	56	49	49	39	29	26	10	18	42
Osaka	44	42	27	55	51	43	47	29	25	47	47	53
Nagoya	30	28	13	48	40	30	30	22	25	21	30	49

RK39.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3  import pandas as pd  # データ操作のためのライブラリをインポート
4  import statsmodels.api as sm  # 線形回帰のためのライブラリをインポート
5
6  # データを作成する
7  Year = [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
8  Tokyo = [31, 25, 20, 56, 49, 49, 39, 29, 26, 10, 18, 42]
9  Osaka = [44, 42, 27, 55, 51, 43, 47, 29, 25, 47, 47, 53]
10 Nagoya = [30, 28, 13, 48, 40, 30, 30, 22, 25, 21, 30, 49]
11
12 # データフレームを作成する
```

```
13  dataSet = pd.DataFrame({'Year': Year, 'Tokyo': Tokyo,
    'Osaka': Osaka, 'Nagoya': Nagoya})
14
15  # 東京と大阪を説明変数、名古屋を目的変数とした線形回帰モデルを作成する
    X = sm.add_constant(dataSet[['Tokyo', 'Osaka']])
16  model_NTO = sm.OLS(dataSet['Nagoya'], X).fit()
17  print(model_NTO.summary())
18
19  # 相関係数を計算する
20  cor_NTO = np.corrcoef(dataSet['Nagoya'],
21  model_NTO.fittedvalues)[0, 1]
    NTO = f"r={cor_NTO:.2f} Nagoya(Tokyo, Osaka) model"
22
23  # 名古屋の実際の値と予測値をプロットする
24  plt.scatter(dataSet['Nagoya'], model_NTO.fittedvalues,
25  color='blue')
    plt.plot([0, 50], [0, 50], color='red', linestyle='--')
26  plt.xlim(0, 50)
27  plt.ylim(0, 50)
28  plt.xlabel('Original Data')
29  plt.ylabel('Predicted Data')
30  plt.title(NTO)
31
32  # グラフを表示する
33  plt.tight_layout()
34  plt.show()
35
```




```
> print(model_NTO.summary())
  OLS Regression Results
==============================================================================
Dep. Variable:                 Nagoya   R-squared:                       0.801
Model:                            OLS   Adj. R-squared:                  0.757
Method:                 Least Squares   F-statistic:                     18.16
Date:                Sat, 01 Jun 2024   Prob (F-statistic):           0.000693
Time:                        11:01:14   Log-Likelihood:                -35.180
No. Observations:                  12   AIC:                             76.36
Df Residuals:                       9   BIC:                             77.82
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -5.9828      6.816     -0.878      0.403     -21.401       9.435
Tokyo          0.3714      0.127      2.923      0.017       0.084       0.659
Osaka          0.5715      0.178      3.215      0.011       0.169       0.974
==============================================================================
Omnibus:                        0.849   Durbin-Watson:                   1.482
Prob(Omnibus):                  0.654   Jarque-Bera (JB):                0.605
Skew:                           0.487   Prob(JB):                        0.739
Kurtosis:                       2.491   Cond. No.                         250.
==============================================================================
```

$(\text{熱帯夜の日数})_{\text{Nagoya}} = -5.9828 + 0.3714\ \left(\text{熱帯夜の日数}\right)_{\text{Tokyo}} + 0.5715\ (\text{熱帯夜の日数})_{\text{Osaka}}$ という回帰モデルが得られました。このように複数の変数[$\left(\text{熱帯夜の日数}\right)_{\text{Osaka}}$ と $(\text{熱帯夜の日数})_{\text{Tokyo}}$]により一つの変数[$(\text{熱帯夜の日数})_{\text{Nagoya}}$] を説明する回帰モデルを、重回帰モデルと呼びます。

統計検定よりそれぞれの係数が 0 と見做せるかどうかを検定すると、p 値は 0.403, 0.017, 0.011 です。つまり、95%信頼区間では -5.9828 は 0 と見做せます。一方、0.3714 と 0.5715 は 0 と見做せません。すなわち、$\left(\text{熱帯夜の日数}\right)_{\text{Osaka}}$ と$(\text{熱帯夜の日数})_{\text{Tokyo}}$は $(\text{熱帯夜の日数})_{\text{Nagoya}}$ に正に寄与していることがわかります。

5. 確率・統計[数学 I, 数学 A]

計算機では、さまざまな乱数を発生させることができます。例えば、コインを投げて表と裏のでる確率を乱数を使って実験することができます。すると、100,000 回試行したときに、表と裏の出る確率が実際にどのくらいばらつくのか？についてもシミュレーション通して推定することができます。ここでは、乱数を使って、確率を理解しましょう。ちなみに高校では　数学 I ならびに数学 A の発展的内容です。

ノック 40 本目

おみくじシステムをつくろう。いま、4 種類のくじ、大吉(Daikichi)、吉(Kichi)、小吉(Shokichi)、凶(Kyo)、を考える。これをランダムに選ぶプログラムを作成しよう。

このことは、sample 関数を用いるとできる。以下のプログラムをみてみよう。

2 行目で c("Daikichi","Kichi","Shokichi","Kyo")で、4 つの事象、大吉(Daikichi)、吉(Kichi)、小吉(Shokichi)、凶(Kyo)を定義し、その起こる確率を prob=c(1/4, 1/4, 1/4, 1/4)としているので、それぞれの起こる確率 P(Daikichi)=P(Kichi)=P(shokichi)=P(Kyo)=1/4 です。この確率で、ランダムに大吉(Daikichi)、吉(Kichi)、小吉(Shokichi)、凶(Kyo)を選び表示してくれます。ここで、replace=True なので、くじを引いて後、元に戻す仕様です。

以下のプログラムを実行しよう。Daikichi が出るまで何度も実行してみよう。

RK40.py

```
1  import numpy as np  # 数値計算のためのライブラリをインポート
2
3  # サンプルサイズを設定する
4  n = 1
5
6  # サンプリングを実行する
7  result = np.random.choice(["Daikichi", "Kichi", "Shokichi",
   "Kyo"], size=n, replace=True, p=[1/4, 1/4, 1/4, 1/4])
8
9  # 結果を表示する
10 print(result)
```

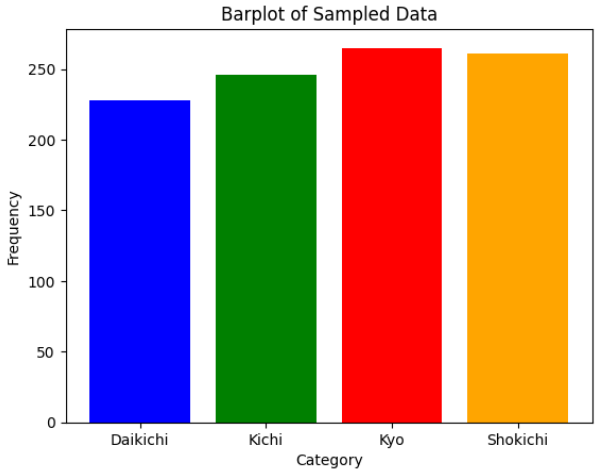
ノック 41 本目

1000 回おみくじを引いてみよう。発生する乱数が毎回異なるので、大吉(Daikichi)、吉(Kichi)、小吉(Shokichi)、凶(Kyo)の出現数が異なることを確かめよう。ただし、それぞれのおみくじの出現確率は 1/4 なので、おおよそ 250 回づつ出現することが確認できるでしょう。

RK41.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # サンプルサイズを設定する
5  n = 1000
6
7  # サンプリングを実行する
8  sData = np.random.choice(["Daikichi", "Kichi", "Shokichi", "Kyo"], size=n, replace=True, p=[1/4, 1/4, 1/4, 1/4])
9
10 # 結果を集計する
11 unique, counts = np.unique(sData, return_counts=True)
12 shukei = dict(zip(unique, counts))
13
14 # 結果を表示する
15 print(shukei)
16
17 # 棒グラフを作成する
18 plt.bar(shukei.keys(), shukei.values(), color=['blue', 'green', 'red', 'orange'])
19 plt.xlabel('Category')
20 plt.ylabel('Frequency')
21 plt.title('Barplot of Sampled Data')
22 plt.show()
```

1000 回ランダムに選ぶと、Daikichi,、Kichi、Shokichi、Kyo はだいたい 250 回ずつになります。ただ乱数を使っているので毎回微妙にかわります。



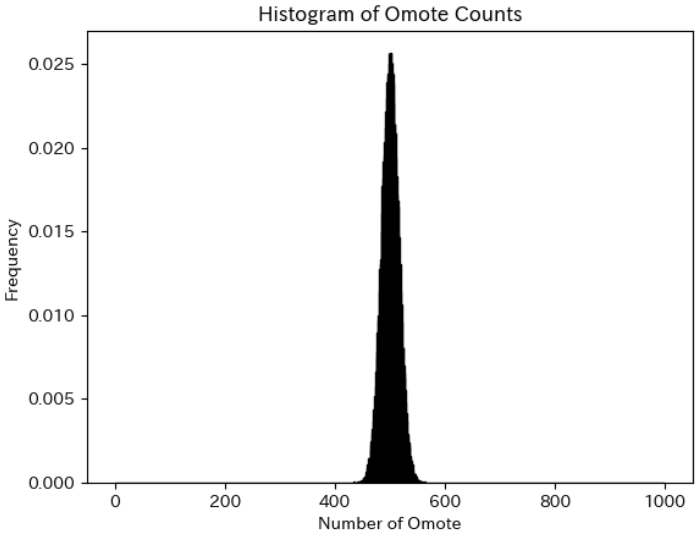
ノック 42 本目

乱数を使って「コインを 1000 回投げる」これを 100000 回行ったとき、0-1000 回表が出る回数についてヒストグラムで表してみよう。

RK42.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # サンプルサイズと試行回数を設定する
5  n = 1000
6  m = 100000
7
8  # 結果を格納する配列を初期化する
9  Omote = np.zeros(m)
10
11 # 試行を実行する
12 for i in range(m):
13     OU = np.random.choice(["Omote", "Ura"], size=n, replace=True, p=[1/2, 1/2])
14     Omote[i] = np.sum(OU == "Omote")
15
16 # ヒストグラムを描く
```

```
17 plt.hist(Omote, bins=np.arange(0, n + 2) - 0.5, density=True,
   edgecolor='black')
18 plt.xlabel('Number of Omote')
19 plt.ylabel('Frequency')
20 plt.title('Histogram of Omote Counts')
21 plt.show()
```



実は、これは二項分布で記述できます。

コインの表と裏の出る確率をそれぞれ $p, q = 1 - p$ としましょう。 N =1000 回コインを投げたとき、k 回表が出る確率を $P(k)$ とします。

$$P(k) = {}_N C_k p^k q^{N-k}$$

いま、$p = q = \frac{1}{2}$ なので

$$P(k) = {}_N C_k p^N$$

となります。

5 回、コイントスをしてみましょう $(N = 1)$

一回も表が出ない確率は

$$P(0) = {}_5 C_0 \left(\frac{1}{2}\right)^5 = \left(\frac{1}{2}\right)^5$$

1 回から 5 回表が出る確率は、

$$P(1) = {}_5C_1\left(\frac{1}{2}\right)^5 = 5\left(\frac{1}{2}\right)^5$$
$$P(2) = {}_5C_2\left(\frac{1}{2}\right)^5 = \frac{5}{2}\cdot\frac{4}{1}\left(\frac{1}{2}\right)^5$$
$$P(3) = {}_5C_3\left(\frac{1}{2}\right)^5 = \frac{5}{2}\cdot\frac{4}{1}\left(\frac{1}{2}\right)^5$$
$$P(4) = {}_5C_4\left(\frac{1}{2}\right)^5 = 5\left(\frac{1}{2}\right)^5$$
$$P(5) = {}_5C_5\left(\frac{1}{2}\right)^5 = \left(\frac{1}{2}\right)^5$$

となります。ここで、${}_NC_k = \frac{N!}{k!(N-k)!}$ です。

$$P(0) + P(1) + \cdots + P(5) = (1 + 5 + 10 + 10 + 5 + 1)\left(\frac{1}{2}\right)^5 = 32\left(\frac{1}{2}\right)^5 = 1$$

と 0-5 回表が出る確率の合計も 1 となります。

これは、実は、$(a+b)^n$ を展開すると、

$$(a+b)^N = {}_NC_0a^Nb^0 + {}_NC_1a^{N-1}b^1 + \cdots + {}_NC_ka^{N-k}b^k + \cdots + {}_NC_{N-1}a^1b^{N-1} + {}_NC_0a^0b^N$$

ここで、$a = p$、$b = q = 1 - p$ とおくと、

$$1 = {}_NC_0p^N(1-p)^0 + {}_NC_1p^{N-1}(1-p)^1 + \cdots + {}_NC_kp^{N-k}(1-p)^k + \cdots + {}_NC_{N-1}p^1(1-p)^{N-1} + {}_NC_Np^0(1-p)^N$$

となり、第 1 項から $N+1$ 項がそれぞれ、$P(0)$, $P(1)$,…, $P(N)$ と対応します。

ノック 43 本目

乱数を使って「コインを 1000 回投げる」これを 100000 回行ったとき、0-1000 回表が出る回数についてヒストグラムを作成し、二項分布、正規分布と比較しよう。

二項分布の平均値は、

$$E(X) = Np$$

分散は、

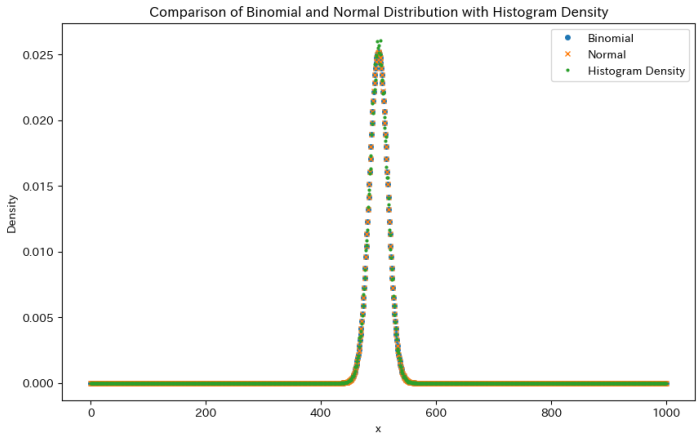
$$V(X) = Np(1-p)$$

となります。これを使って正規分布の密度関数と比較できます。これを確認しましょう。

RK43.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3  from scipy.stats import binom, norm  # 統計のためのライブラリをインポート
4
5  # サンプルサイズと試行回数を設定する
6  n = 1000
7  m = 100000
8
9  # 結果を格納する配列を初期化する
10 Omote = np.zeros(m)
11
12 # 試行を実行する
13 for i in range(m):
14     OU = np.random.choice(["Omote", "Ura"], size=n, replace=True, p=[1/2, 1/2])
15     Omote[i] = np.sum(OU == "Omote")
16
17 # ヒストグラムを描く
18 hist_values, bin_edges = np.histogram(Omote, bins=np.arange(0, n + 2) - 0.5, density=True)
19
20 # 平均と標準偏差を計算する
21 av = n * 1/2
22 sd = np.sqrt(n * 1/2 * (1 - 1/2))
23
24 # x 軸の値を作成する
25 xaxis = np.arange(0, n + 1)
26
27 # 二項分布の確率質量関数と正規分布の確率密度関数を計算する
28 bi = binom.pmf(xaxis, n, 1 / 2)
29 dn = norm.pdf(xaxis, av, sd)
30
31 # ヒストグラムの密度を取得する
```

```
32  rdens = hist_values
33
34  # matplot を使用してプロットする
35  plt.figure(figsize=(10, 6))
36  plt.plot(xaxis, bi, 'o', label='Binomial', markersize=4)
37  plt.plot(xaxis, dn, 'x', label='Normal', markersize=4)
38  plt.plot(xaxis, rdens, '.', label='Histogram Density',
    markersize=4)
39  plt.legend()
40  plt.xlabel('x')
41  plt.ylabel('Density')
42  plt.title('Comparison of Binomial and Normal Distribution
    with Histogram Density')
43  plt.show()
```



プログラム中の bi 理論分布から導いた二項分布の密度分布(1 と表記)、dn は正規分布の密度分布(2 と表記)、の rdens はコインを 1000 回投げた時のコインの表と裏の相対数を 10000 回行った場合の分布(3 と表記)を示しています。

図をみると、黒(数字は 1）がランダムサンプリングによるコイントスで表が出る相対頻度、赤(数字の 2）と緑(数字の 3)は、二項分布と正規分布の理論頻度です。結局のところ、試行回数を増やすと、ランダムサンプリングによるコイントスで表が出る相対頻度は、二項分布および正規分布とほぼ同一になります。このように試行数を 1000 回にするとほぼ同じ分布となります。こうやって、計算機実験ができる楽しさを実感しよう。

二項分布による統計検定 [数学 I]

ノック 44 本目

T 社と K 社でマスクを開発し、どちらを好むか調査した。20 人を無作為に抽出し、T 社と K 社のマスクしてもらい、どちらがよいか訊いた。T 社製がよいと言った人は 5 人、K 社製がよいと言った人は 15 人だった。このとき K 社と T 社のマスクの好みは同等か？

この課題を確率値（p 値）で評価する。まず仮説をたてる。

帰無仮説 H_0：T 社と K 社のマスクに着け心地などの差はない。

とする。これに対する対立仮説は 3 つ設定できる。

対立仮説 H_1:

(1) $x_T(=x) \neq x_K(=N-x)$

(2) $x_T(=x) < x_K(=N-x)$

(3) $x_T(=x) > x_K(=N-x)$

ところで、$x_T = 5$、$x_K = 15$であるので、

いま(3)は成り立たないので、(1)と(2)の検定を行う。

「T 社製がよい人は 5 人、K 社製がよい人は 15 人」ということは、見かけ上、K 社製が優勢である。これを統計検定する。まず、「K 社製がよい人が 15 人」ということを、「K 社製がよい人は少なくとも 15 人」と捉える。そこで、少なくとも 15 人だから、K 社製がよい人が 15-20 人の確率を求めることになる。ここで $p = \frac{1}{2}$ とする。

$$P(k) = {}_N C_k p^k (1-p)^{N-k}$$

について、$N = 20,\ p = 0.5$ を代入すると

$$P(k) = {}_N C_k 0.5^k (0.5)^{N-k} = {}_{20} C_k 0.5^{20}$$

となる。そこで、$P(15) - P(20)$ の総和を求める。

$$P(15) + \cdots + P(20) = 0.5^{20}\{{}_{20}C_{15} + {}_{20}C_{16} + {}_{20}C_{17} + {}_{20}C_{18} + {}_{20}C_{19} + {}_{20}C_{20}\}0.5^{20}$$

これを Python で計算してみよう。

RK44.py

```
import math  # 数学関数のためのライブラリをインポート

# 二項係数を計算し、確率を求める
Pr15 = math.comb(20, 15) * (0.5 ** 20)
Pr16 = math.comb(20, 16) * (0.5 ** 20)
Pr17 = math.comb(20, 17) * (0.5 ** 20)
Pr18 = math.comb(20, 18) * (0.5 ** 20)
Pr19 = math.comb(20, 19) * (0.5 ** 20)
Pr20 = math.comb(20, 20) * (0.5 ** 20)

# 確率の合計を計算する
result = Pr15 + Pr16 + Pr17 + Pr18 + Pr19 + Pr20
print(f"{result:.8f}")
```

```
> result
[1] 0.02069473
```

となり p <0.05 となり、H_0 は棄却され、H_1 (2) $x_T(=x) < x_K(=N-x)$が採択される。

一方、

$$P(15)+\cdots+P(20)=0.5^{20}\{{}_{20}C_{15}+{}_{20}C_{16}+{}_{20}C_{17}+{}_{20}C_{18}+{}_{20}C_{19}+{}_{20}C_{20}\}0.5^{20}$$

と

$$P(0)+\cdots+P(5)=0.5^{20}\{{}_{20}C_{0}+{}_{20}C_{1}+{}_{20}C_{2}+{}_{20}C_{3}+{}_{20}C_{4}+{}_{20}C_{5}\}0.5^{20}$$

の和を求めると、

0.02069473・2= 0.04138946

となり、これも $p < 0.05$ であるので、H_0 は棄却され、H_1 (1) $x_T(=x) \neq x_K(=N-x)$が採択される。

この検定を二項検定といい、binomtest() により検定できる。

帰無仮説 H_0：T 社と K 社のマスクに着け心地などの差はない。

とする。これに対する対立仮説は 3 つ設定できる。

対立仮説 H_1:

(1) $x_T(=x) \neq x_K(=N-x)$

(2) $x_T(=x) < x_K(=N-x)$

(3) $x_T(=x) > x_K(=N-x)$

ところで、$x_T = 5$、$x_K = 15$ である。binomtest() の関数は以下のように定義されている。

```
binomtest(
    15, # 成功数
    20,  # 全数
    p=0.5, # 成功の仮定された確率
    alternative='two-sided' # 対立仮説:"two-sided", "greater",
"less"
)
```

3 種の対立仮説"two.sided", "greater", "less"で定義できる。以下のプログラムの[A]、[B]、[C]は上の対立仮説のどれと対応するか？プログラムを実行して検討しよう。

ノック 45 本目

T 社と K 社でマスクを開発し、どちらを好むか調査した。20 人を無作為に抽出し、T 社と K 社のマスクしてもらい、どちらがよいか訊いた。T 社製がよいと言った人は 5 人、K 社製がよいと言った人は 15 人だった。このとき K 社と T 社のマスクの好みは同等か？

これを 3 種の対立仮説"two.sided", "greater", "less"で定義して二項検定しよう。

RK45.py

```
1  from scipy.stats import binomtest  # 統計のためのライブラリをインポート
2
3  # [A] 両側検定
4  result_A = binomtest(15, 20, p=0.5, alternative='two-sided')
5  print("Two-sided test p-value:", result_A)
6
7  # [B] 片側検定(greater)
8  result_B = binomtest(15, 20, p=0.5, alternative='greater')
9  print("Greater test p-value:", result_B)
10
11 # [C] 片側検定(less)
12 result_C = binomtest(15, 20, p=0.5, alternative='less')
13 print("Less test p-value:", result_C)
```

```
> binomtest(15, 20, p=0.5, alternative='two-sided')
Two-sided test p-value: 0.04138946533203125

> binomtest(15, 20, p=0.5, alternative='greater')
Greater test p-value: 0.020694732666015625

> binomtest(15,20,,p=0.5,alternative="less")
Less test p-value: 0.9940910339355469
```

6. 集合 [数学 I]

集合といえば、以下のような感じです。

1 から 1000 までの整数について、2 の倍数、3 の倍数、 2 かつ 3 の倍数、 2 あるいは 3 のいずれの倍数でもない数を求めてみよう。

では、計算機では集合の問題をどうやって扱うのでしょうか？課題を解決しながら理解しましょう。

ノック 46 本目

1 から 1000 までの整数について、2 の倍数の数を求めよう。

RK46.py

```
1  import numpy as np  # 数値計算のためのライブラリをインポート
2
3  # 1 から 1000 までの数を作成する
4  N = np.arange(1, 1001)
5
6  # 2 で割り切れるかどうかのブール配列を作成する
7  TF2 = (N % 2 == 0)
8
9  # 2 で割り切れる数を抽出する
10 fold2 = N[TF2]
11
12 # 2 で割り切れる数の個数を計算する
13 Nfold2 = len(fold2)
14 Nfold2
```

1 行目：N には 1 から 1000 の数字が格納されている。N[1]=1, N[2]=2, … , N[1000]=1000 である。
c(1:1000) は 1 から 1000 からなるベクトルを定義して N に格納する。
2 行目: N%2==0 は、N を 2 で割った余り(%)が 0 のとき True(真)、0 でないときに False(偽)となる。
TF2 には、False, True, False, True, …, False, True が格納される。
「A == B」は A と B が同じとき True となる。一方「A!=B」とすると、A と B が異なるとき True となる。
3 行目：N[TF2]により、TF2 が True の要素が fold2 に格納される。fold2[1]=2, fold2[2]=4, …,
fold2[500]=1000 となる。

4 行目：length()により要素の数を求めることできる。要素数 Nfold2 は 500 となる。

このようにして 1 から 1000 までの整数では、2 で割り切れる整数は 500 個あることがわかった。

ノック 47 本目

1 から 20 までの整数について、2 の倍数からなる集合を A、3 の倍数からなる集合を B とします。二つの集合の要素数によりベン図を作成しよう。

RK47.py

```
1  import numpy as np  # 数値計算のためのライブラリをインポート
2
3  # 1 から 20 までの数を作成する
4  N = np.arange(1, 21)
5
6  # 各条件を満たす数を抽出する
7  even_not_div3 = N[(N%2==0) & (N%3!=0)]
8  not_even_div3 = N[(N%2!=0) & (N%3==0)]
9  even_div3 = N[(N%2==0) & (N%3==0)]
10 not_even_not_div3 = N[(N%2!=0) & (N%3!=0)]
11
12 # 各条件を満たす数の個数を計算する
13 len_even_not_div3 = len(even_not_div3)
14 len_not_even_div3 = len(not_even_div3)
15 len_even_div3 = len(even_div3)
16 len_not_even_not_div3 = len(not_even_not_div3)
17
18 # 結果を表示する
19 print("N[N%2==0 & N%3!=0]:", even_not_div3)
20 print("N[N%2!=0 & N%3==0]:", not_even_div3)
21 print("N[N%2==0 & N%3==0]:", even_div3)
22 print("N[N%2!=0 & N%3!=0]:", not_even_not_div3)
23 print("length(N[N%2==0 & N%3!=0]):", len_even_not_div3)
24 print("length(N[N%2!=0 & N%3==0]):", len_not_even_div3)
25 print("length(N[N%2==0 & N%3==0]):", len_even_div3)
26 print("length(N[N%2!=0 & N%3!=0]):", len_not_even_not_div3)
```

4 行目：N には 1,2,3,…,20 が格納されます。

7 行目：length(N[N%2==0 & N%3!=0])について、

N%2==0 はベクトル N の要素について 2 で割った余りが 0 であれば True、

N%3!=0 はベクトル N の要素について 3 で割った余りが 0 でなければ True となります。

これらの二つの条件が「&」であるので、「N%2==0」かつ「N%3!=0」が成り立てば True となります。以下の表では、T は True、F は False を表しています。空欄をうめてみよう。

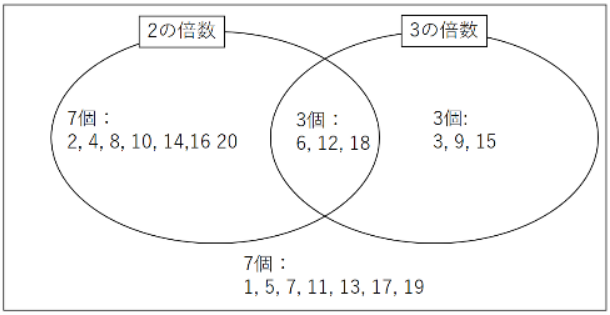
要素	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
N%2==0	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
N%3!=0	T	T	F	T	T	F	T	T	F	T	T	F	T	T	F	T	T	F	T	T
N%2==0& N%3!=0	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T
N%2!=0 & N%3==0																				
N%2==0& N%3==0																				
N%2!=0 & N%3!=0																				

となり、N[N%2==0 & N%3!=0]は{2,4,8,10,14,16,20}の 7 個の要素からなるベクトルとなります。同様に、N[N%2!=0 & N%3==0]、N[N%2!=0 & N%3==0]、N[N%2==0 & N%3==0]、N[N%2!=0 & N%3!=0]の要素を求めよう。

出力結果をみると

```
N[N%2==0 & N%3!=0]: [ 2  4  8 10 14 16 20]
N[N%2!=0 & N%3==0]: [ 3  9 15]
N[N%2==0 & N%3==0]: [ 6 12 18]
N[N%2!=0 & N%3!=0]: [ 1  5  7 11 13 17 19]
length(N[N%2==0 & N%3!=0]): 7
length(N[N%2!=0 & N%3==0]): 3
length(N[N%2==0 & N%3==0]): 3
length(N[N%2!=0 & N%3!=0]): 7
```

となる。というわけで、ベン図に表すととなります。

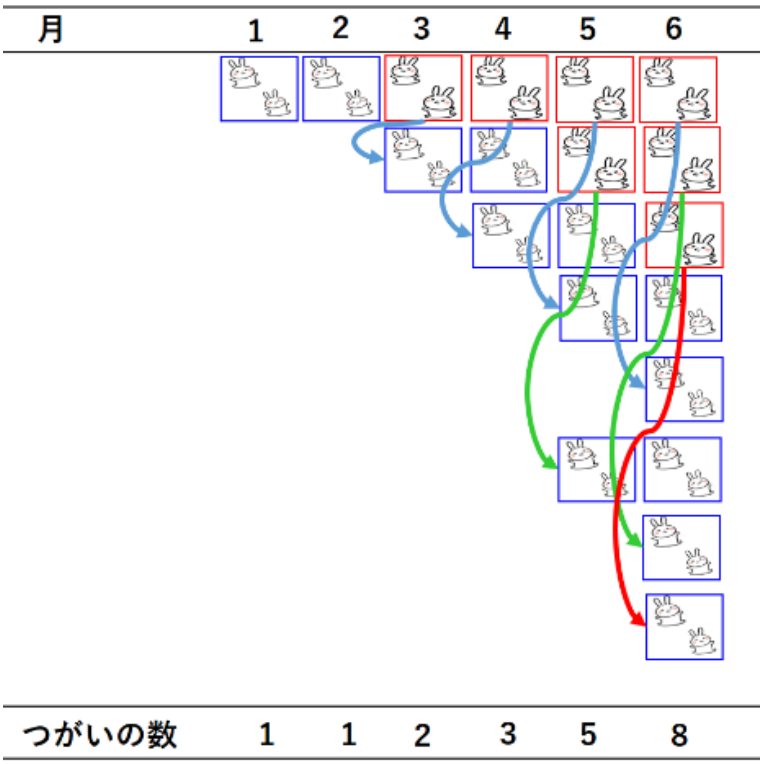


7. フィボナッチ数列 [数学 B]

レオナルド・フィボナッチは次の問題を考案した数列です。高校の数学の教科書でも、フィボナッチ数列は数学 B で取り上げられています。さらにこのフィボナッチ数は、黄金比と関係があります。黄金比については、数学 I で取り上げられています。異なった章で学ぶことが実は関係があるということを発見するとなんとも嬉しくなります。ここでは、フィボナッチ数と黄金比の関係を軽く理解しましょう。

ノック 48 本目

1 つがいの兎は、産まれて 2 か月後から毎月 1 つがいずつの兎を産む。 兎が死なずずっと生み続ける。この条件の下で、産まれたばかりの 1 つがいの兎は 1 年の間に何つがいの兎になるか？



6 カ月目までを図示し、1 から 6 カ月の兎のつがいの数を数えると、1、 1 、2、3、5、8 となる。この数列をフィボナッチ数 (Fibonacci sequence) (F_n) といい、次の式で定義される。このフィボナッチ数を計算しよう。

$F_0 = 0,$

$F_1 = 1,$

$F_2 = F_0 + F_1 = 0+1=1$

$F_3 = F_1 + F_2 = 1+1=2$

$F_4 = F_2 + F_3 = 1+2=3$

$F_5 = F_3 + F_4 = 2+3=5$

…

$F_{n+2} = F_n + F_{n+1}\ (n \geq 0)$

これをプログラミングしてみよう。

RK48.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2
3  # フィボナッチ数列の長さを設定する
4  ns = 13
5
6  # フィボナッチ数列を初期化する
7  Fib = [0] * ns
8  Fib[0] = 0
9  Fib[1] = 1
10
11 # フィボナッチ数列を計算する
12 for i in range(2, ns):
13     Fib[i] = Fib[i-1] + Fib[i-2]
14
15 # フィボナッチ数列を表示する
16 print(Fib)
17
18 # フィボナッチ数列をプロットする
19 plt.plot(range(0, ns), Fib, marker='o')
20 plt.xlabel('i')
21 plt.ylabel('Fibonacci sequence')
22 plt.title('Fibonacci Sequence')
23 plt.grid(True)
24 plt.show()
```

> print(Fib)

```
[  0.   1.   1.   2.   3.   5.   8.  13.  21.  34.  55.  89. 144.]
```

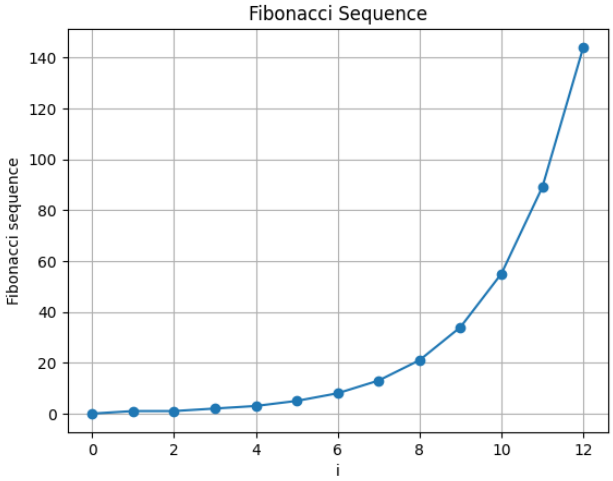
はじめを 0 カ月としているので、Fib[12]が 12 か月後の兎のつがいの数 144 となる。

4 行目：ns = 13 ここで、 0 番目のフィボナッチ数 Fib[0]、12 番目を Fib[12]とした。

7 行目：Fib[0] から Fib[ns] のベクトルを定義する。とりあえず 0 が入っています。

8-9 行目：Fib[0] に 0 、Fib[1] に 1 を代入する。

12-13 行目： i を順次 2, 3, … , ns-1 として、Fib[2], Fib[3], … , Fib[ns-1] と求めます。



ノック 49 本目

i 番目のフィボナッチ数を F_i としよう

$$r_i = \frac{F_i}{F_{i-1}}$$

について、i を大きくすると黄金比に近づくという。そこで、

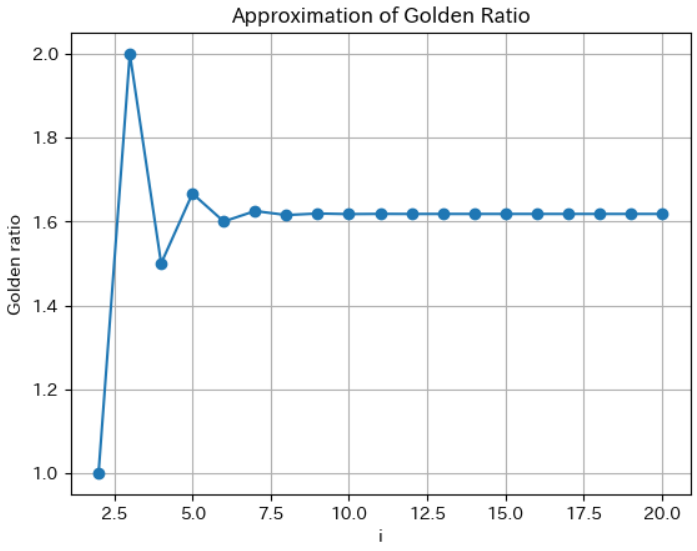
$i = 1, 2, \cdots, 100$ を横軸に、r_i を縦軸に図に表してみよう。

ちなみに、黄金比は、$1 : \frac{1+\sqrt{5}}{2} \cong 1 : 1.618\ldots$ で、「人間が最も美しいと感じる比率」なのだそうです。フィボナッチ数と黄金比の関係を図示しよう。

RK49.py

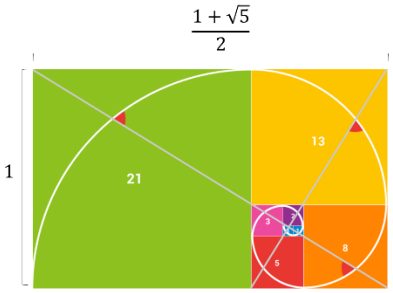
```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # フィボナッチ数列の長さを設定する
5  ns = 21
6
7  # フィボナッチ数列を初期化する
8  Fib = np.zeros(ns)
9  Fib[0] = 0
10 Fib[1] = 1
11
12 # フィボナッチ数列を計算する
13 for i in range(2, ns):
```

```
    Fib[i] = Fib[i-1] + Fib[i-2]

# 黄金比の近似値を格納する配列を初期化する
r = np.zeros(ns-1)

# 黄金比の近似値を計算する
for i in range(ns-1):
    if Fib[i] == 0:
        r[i] = np.inf # 分母が 0 の場合、無限大を代入
    else:
        r[i] = Fib[i+1] / Fib[i]

# 黄金比の近似値をプロットする
plt.plot(range(1, ns), r, marker='o')
plt.xlabel('i')
plt.ylabel('Golden ratio')
plt.title('Approximation of Golden Ratio')
plt.grid(True)
plt.show()

# 最後の黄金比の近似値と理論値を表示する
print("最後の黄金比の近似値:", r[ns-2])
print("理論的な黄金比:", (1 + np.sqrt(5)) / 2)
```



```
> print("最後の黄金比の近似値:", r[ns-2])
最後の黄金比の近似値： 1.6180339631667064
> print("理論的な黄金比:", (1 + np.sqrt(5)) / 2)
理論的な黄金比： 1.618033988749895
```

ちなみに、黄金比はオウムガイの形とも関係あるのだそうです。



たねあかし

フィボナッチ数列とは

$$F_0 = 0$$
$$F_1 = 1$$
$$F_{n+2} = F_n + F_{n+1}$$

です。これは実は、

$$F_n = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right\}$$

と書けます。

$$F_0 = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^0 - \left(\frac{1-\sqrt{5}}{2}\right)^0\right\} = 0$$
$$F_1 = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^1 - \left(\frac{1-\sqrt{5}}{2}\right)^1\right\} = \frac{1}{\sqrt{5}}\left\{\frac{1+\sqrt{5}-1+\sqrt{5}}{2}\right\} = \frac{1}{\sqrt{5}}\left\{\frac{2\sqrt{5}}{2}\right\} = 1$$
$$F_2 = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^2 - \left(\frac{1-\sqrt{5}}{2}\right)^2\right\} = \frac{1}{\sqrt{5}}\left\{\frac{6+2\sqrt{5}-6+2\sqrt{5}}{4}\right\} = \frac{1}{\sqrt{5}}\left\{\frac{4\sqrt{5}}{4}\right\} = 1$$
$$F_3 = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^3 - \left(\frac{1-\sqrt{5}}{2}\right)^3\right\} = \frac{1}{\sqrt{5}}\left\{\frac{16+8\sqrt{5}-16+8\sqrt{5}}{8}\right\} = \frac{1}{\sqrt{5}}\left\{\frac{8\sqrt{5}}{4}\right\} = 2$$

…

では

$$F_k + F_{k+1} = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k\right\} = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^{k+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{k+1}\right\}$$
$$= \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^k\left(1+\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)^k\left(1-\frac{1-\sqrt{5}}{2}\right)\right\}$$
$$= \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^k\left(\frac{3+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)^k\left(\frac{3-\sqrt{5}}{2}\right)\right\}$$

ここで、

$$\left(\frac{1+\sqrt{5}}{2}\right)^2 = \frac{6+2\sqrt{5}}{4} = \frac{3+\sqrt{5}}{2}$$
$$\left(\frac{1-\sqrt{5}}{2}\right)^2 = \frac{6-2\sqrt{5}}{4} = \frac{3-\sqrt{5}}{2}$$

であることを活用すると、

$$F_k + F_{k+1} = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^k\left(\frac{3+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)^k\left(\frac{3-\sqrt{5}}{2}\right)\right\}$$
$$= \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^k\left(\frac{1+\sqrt{5}}{2}\right)^2 - \left(\frac{1-\sqrt{5}}{2}\right)^k\left(\frac{1-\sqrt{5}}{2}\right)^2\right\} = F_{k+2}$$

ということで、

$$F_k + F_{k+1} = F_{k+2}$$

が成り立っています。

では、

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n}$$

としたときの極限値を求めてみましょう。式を一生懸命変形すると、

$$\frac{F_{n+1}}{F_n} = \frac{\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)\left(\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)^n}{1-\left(\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)^n}$$

$n \to \infty$ とすると、$\left(\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)^n \to 0$ となるので、

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \frac{1+\sqrt{5}}{2}$$

となります。これが黄金比となる訳だ、なるほど！

8. 素数 [数学 A]

素数とは、1 とそれ自身以外に正の約数をもたない自然数です。NEXT 数学 A「第 3 章 数学と人間の活動」でも取り上げられています。素数の分布に関する法則としては「リーマン予想」というのがあって、1859 年にドイツの数学者ベルンハルト・リーマン（1826～1866）によって提唱されました。その予想の具体的な内容は大変難しいのでここでは割愛しますが、一見ランダムに見える素数のすべてに共通する秩序があることになるのだそうです。リーマン予想の証明は現在も未解決であり、アメリカのクレイ数学研究所によって 100 万ドル(1 億 4 千万円)の懸賞金がかけられているのだそうで、相当、数学者が討ち死にしたのだろうと予想がつきます。リーマン予想と同じく、これまでに例外は見つかっていないものの、正しいことが証明されていない素数に関する「法則」として「4 以上の偶数はすべて、2 つの素数の足し算で表せる」というのもありますが、ここでは深入りするのはやめましょう。素数を求める単純なプログラムを作ってみましょう。

ノック 50 本目

RK50.py は、ある自然数について素数かどうか判定するプログラムを作成しよう。

いま、ある自然数を NN としよう。素数の定義から 2, 3, …, NN-1 で割り算をしたときの余りが 0 とならなければ素数である。プログラムでは、79190 が素数であるかを判定する。

NN を 79190 とすると

```
    79190 is NOT Prime Number yo!
```

と出力され素数(Prime Number)でないことがわかり、約数が表示される。

```
    > np.where(disc == 1)[0]
[    2     5    10  7919 15838 39595]
```

では、NN を 7919 としてみよう。すると

```
    7919 is Prime Number yo!
```

となり素数とわかります。

```
    > np.where(disc == 1)[0]
     []
```

となり 1 と自分自身以外の約数はないことがわかります。

RK50.py

```
1  import numpy as np  # 数値計算のためのライブラリをインポート
2
3  # チェックする数を設定する
4  NN = 79190
```

```
5
6  # 分割可能性を記録する配列を初期化する
7  disc = np.zeros(NN)
8
9  # 2 から NN-1 までの数で割り切れるかどうかをチェックする
10 for i in range(2, NN):
11     if NN % i == 0:
12         disc[i] = 1
13
14 # 素数かどうかを判定する
15 if np.sum(disc) == 0:
16     pp = f"{NN} is Prime Number yo!"
17 else:
18     pp = f"{NN} is NOT Prime Number yo!"
19 print(pp)
20
21 # 割り切れる数を表示する
22 factors = np.where(disc == 1)[0]
23 print(factors)
```

ノック 51 本目

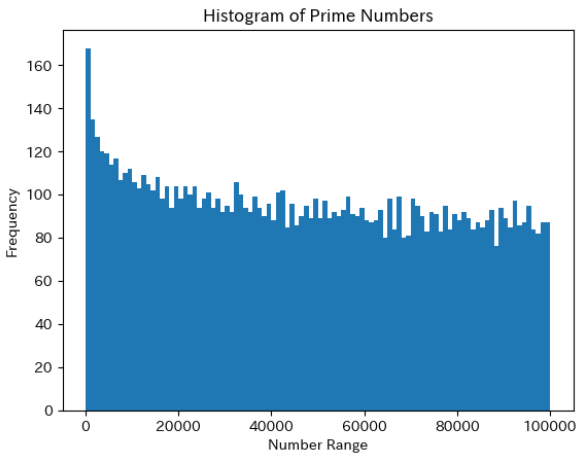
100000 以下の素数の個数を求めて、1000 ごとに区切ってヒストグラムをつくってみよう。

RK51.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  # 最大数を設定する
5  maxN = 100000
6
7  # 素数のリストを初期化する
8  pnum = [2]
9
```

```
# 3 から maxN までの数について素数かどうかをチェックする
for i in range(3, maxN + 1):
    is_prime = True
    for k in range(len(pnum)):
        if i % pnum[k] == 0:
            is_prime += False
            break
    if is_prime:
        pnum.append(i)

# 素数の個数を表示する
ns = len(pnum)
print("Number of primes:", ns)

# 素数のヒストグラムを作成する
his = plt.hist(pnum, bins=np.arange(0, maxN + 1000, 1000))
plt.xlabel('Number Range')
plt.ylabel('Frequency')
plt.title('Histogram of Prime Numbers')
plt.show()

# ヒストグラムのカウントを表示する
print("Histogram counts:", his[0])
```

1000 ずつに区切って素数の数を求めると徐々に減っているようには見えるが…。



自然界の素数の話題

素数ゼミ

北アメリカで、17 年あるいは 13 年という素数の年周期で大発生するセミがいる。セブテンデシム(*Magicicada septendecim*)、カッシーニ(*Magicicada cassini*)、セブテンデキュラ(*Magicicada septendecula*)は 17 年周期で大発生する。一方、トレデシム(*Magicicada tredecim*)、ネオトレデシム(*Magicicada neotredecim*)、トレデキュラ(*Magicicada tredecula*)は 13 年周期で大発生する。この周期性は、それぞれの種の交雑を防ぐことと関連するらしい。生物の発生周期が素数で説明できる？(吉村仁著、「17 年と 13 年だけ大発生？：素数ゼミの秘密に迫る！」(サイエンス・アイ新書)。

最大公約数(greatest common measure)

ユークリッドの互除法

二つの整数 (とりあえず共に正)としよう。

まず、二つの整数の大きい方を r_0、残りを r_1 とする。

(第 1 回)　$r_0 \div r_1 = p_1$ 余り $r_2 \neq 0$

(第 2 回)　$r_1 \div r_2 = p_2$ 余り $r_3 \neq 0$

(第 3 回)　$r_2 \div r_3 = p_3$ 余り $r_4 \neq 0$

:

のように割り算の余り r_n-1 が 0 でなければ、その余り r_{n-1} で除数 r_n を割っていく ことを繰り返していくと，あるところで必ず

(第 k 回)　$r_{k-1} \div r_k = p_k$　余り $r_{k+1} \neq 0$

(第 k+1 回) $r_k \div r_{k+1} = p_{k+1}$ 余り $r_{k+2} = 0$

と割りきれるので，このとき

最大公約数は、gcd(A,B) = r_{k+1}

つまり、r_{k+2}=0 となったときの、r_{k+1} が最大公約数となる。

これをプログラミングしよう。

ノック 52 本目

6006 と 391391 の最大公約数を求めよう。

RK52.py

```
1  # 最大公約数 (GCD) を計算するプログラム
2
3  # 2 つの数を設定する
4  x1 = 6006
5  x2 = 391391
6
7  # 初期値を設定する
8  r0 = max(x1, x2)
9  r1 = min(x1, x2)
10 r2 = r0 % r1
11 bingo = r1
12
```

```
13 # ユークリッドの互除法を使用して GCD を計算する
14 while r2 != 0:
15     r3 = r1 % r2
16     print(f"[r1 % r2] = {r3}, [r2] = {r2}")
17     bingo = r2
18     r1 = r2
19     r2 = r3
20
21 # 結果を表示する
22 print("GCD:", bingo)
```

```
> print("GCD:", bingo)
GCD: 1001
```

最小公倍数(least common multiple)を求めよう。

ノック 53 本目

ノック 52 本目で作成した最大公約数を求めるプログラムをもとに、最小公倍数を求めるプログラムをつくってみよう。

たねあかし

a と b の最大公約数 G 、b と r の最大公約数を g としたときに、$G = g$ となります。まずこれを示しましょう。

a と b で割ったときの商を q 余りを r とすると

$$a = bq + r \tag{1}$$

と表すことができます。

a と b の最大公約数は G 、b と r の最大公約数は g であるので

$$a = Ga' \tag{2}$$

$$b = Gb' = gb'' \tag{3}$$

$$r = gr'' \tag{4}$$

とあらわすことができます。ここで、a', b', b'', r'' は自然数です。
(1)が成り立つので、これに(2)と(3)式をもとに

$$a = bq + r = g(b''q + r'')$$

と変形できて、a は g で割り切れることがわかります。また b はもともと g で割り切れるので、g は a と b の公約数です。G は a と b との最大公約数なので

$$g \leq G \qquad (5)$$

が成り立ちます。
次に(1)を変形して

$$r = a - bq = G(a' - b'q)$$

と表せるので、r は G でも割り切れます。つまり G は b と r の約数となります。
b と r の最大公約数は g なので、

$$G \leq g \qquad (6)$$

が成り立ちます。
そこで、(5)と(6)の条件を満たすのは、

$$G = g \qquad (7)$$

となります。
これで a と b の最大公約数 G と b と r の最大公約数を g としたときに、$G = g$となることが示されました。
ということは、

$$a_1 = b_1 q_1 + r_1$$

a_1 を b_1 で割ったときの余りを r_1 を求めると、$a_1 > b_1, a_1 > r_1$ となります。ここで $r_1 = 0$ となれば、q_1 が最大公約数となります。

$r_1 = 0$ でないときには、b_1 を r_1 で割った余り r_2 を求めます。

$$b_1 = r_1 q_2 + r_2$$

ここで $r_2 = 0$ となれば、q_2 が最大公約数となります。このように順次 r_i が 0 になるまで割り算をしていき、$r_i = 0$ となれば、r_{i-1} が最大公約数となります。

つまり

$$b_1 > r_1 > r_2 > \cdots > r_n > r_{n+1} = 0$$

と順次 r_i を求めていき r_{n+1} が 0 となったときの r_n が最大公約数となります。

ノック 54 本目

37037037 と 86419753 の最大公約数と最小公倍数を求めよう。

ノック 55 本目

148005 と 376805 の最大公約数を求めよう。

ノック 56 本目

6006 と 391391 の最小公倍数を求めよう。

9. 無限数列の和 [数学 III]

無限等比級数などさまざまな規則性のある級数の総和が意外にも π に近づいたり、整数になったりと思いがけない数値と等しくなることがあります。この章では、規則性のある数列の和の収束する値をプログラミングを通して観察してみましょう。数学 B「数列」の発展形だと思えば、たいして難しくありません。いえいえ、証明しろと言われれば結構むずかしいものもありますが、計算機実験でなるほどと思うには面白い課題だと思います。

例えば、初項が 1 の等比級数の和を

$$S_n = 1 + r + r^2 + \cdots + r^{n-1} = \sum_{k=0}^{n-1} r^k$$

とします。この等比級数の和をもとに

$$rS_n = r + r^2 + \cdots + r^{n-1} + r^n$$

をつくり、上式から下式を引くと、

$$S_n = \frac{1-r^n}{1-r}$$

が得られます。ここで、$|r| < 1$ のとき、n をどんどん大きく $(n \to \infty)$ すれば、$r^n \to 0$ となるので、等比級数の和 S_n は $\frac{1}{1-r}$ に近づきます。この r を $\frac{1}{2}$ とすると

$$S_n = \left(\frac{1}{2}\right)^0 + \cdots + \left(\frac{1}{2}\right)^{n-1}$$

の n を無限大に多きくすると 2 に近づきます。これは

$$S_n = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots$$

と同じ式で、なんとも分母が 2 倍ずつ増えていて規則性のある級数です。また $\frac{1}{1-r} = \frac{1}{1-\frac{1}{2}} = 2$ となるので、S_n は 2 に近づきます。以下ではいろいろな級数の例を紹介します。終息する値はなんだろうと楽しんでください！

$\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$ ただし、$\lvert r \rvert < 1$

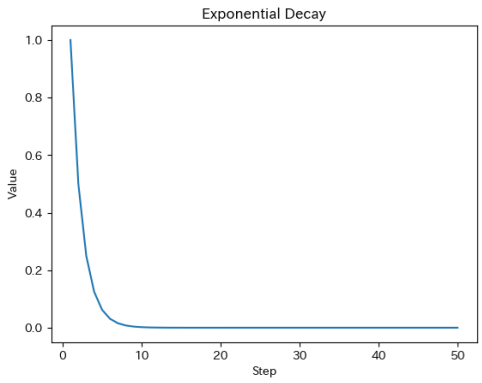
ということを、高校数学 B[第 1 章　数列]で習います。これを図に表すとどうなるのでしょうか？いろいろな規則性のある数列について図に表しながら考えてみましょう。

ノック 57 本目

$r = \frac{1}{2}$ としたときの級数 $\left(\frac{1}{2}\right)^0$, $\left(\frac{1}{2}\right)^1$, $\left(\frac{1}{2}\right)^2$, …, $\left(\frac{1}{2}\right)^{49}$ を縦軸、横軸を $0, 1, \ldots, 49$ として散布図をつくってみよう。

RK57.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2
3  ns = 50  # ステップ数の設定
4  r = 1 / 2  # 減衰率の設定
5  x = [0] * ns  # 長さ ns のリストを 0 で初期化
6
7  for i in range(ns):  # 0 から ns-1 までのループ
8      x[i] = r ** i  # 減衰計算
9
10 plt.plot(range(ns), x)  # プロットの作成
11 plt.xlabel('Step')  # x 軸ラベル
12 plt.ylabel('Value')  # y 軸ラベル
13 plt.title('Exponential Decay')  # タイトル
14 plt.show()  # プロットの表示
```

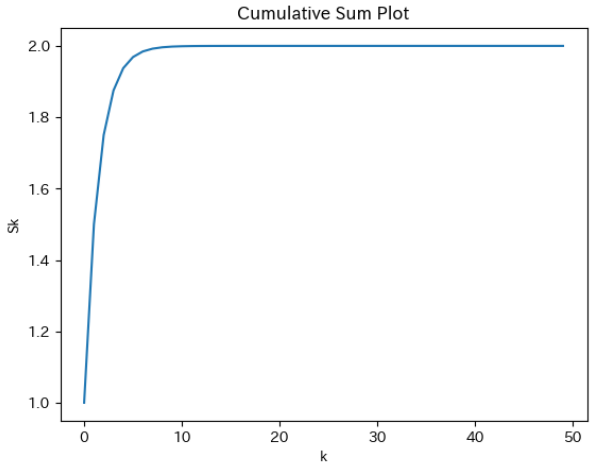


ノック 58 本目

$r=\frac{1}{2}$ としたときの級数 $\left(\frac{1}{2}\right)^0$, $\left(\frac{1}{2}\right)^1$, $\left(\frac{1}{2}\right)^2$, $\cdots$, $\left(\frac{1}{2}\right)^{49}$ について、
$S_0=\left(\frac{1}{2}\right)^0$、$S_1=\left(\frac{1}{2}\right)^0+\left(\frac{1}{2}\right)^1$、$S_2=\left(\frac{1}{2}\right)^0+\left(\frac{1}{2}\right)^1+\left(\frac{1}{2}\right)^2$、$S_{50}=\left(\frac{1}{2}\right)^0+\cdots+\left(\frac{1}{2}\right)^{49}$
を求めて、横軸を $0, 1, \ldots, 49$、縦軸を $S_0, S_1, \cdots, S_{49}$ として値をプロットしてみよう。

RK58.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  ns = 50  # ステップ数の設定
5  r = 1 / 2  # 減衰率の設定
6  x = [0] * ns  # 長さ ns のリストを 0 で初期化
7
8  for i in range(ns):  # 0 から ns-1 までのループ
9      x[i] = r ** i  # 減衰計算
10
11 sx = [0] * ns  # 累積和のリストを 0 で初期化
12
13 for k in range(ns):  # 0 から ns-1 までのループ
14     sx[k] = sum(x[:k+1])  # x の最初から k までの要素の和を計算
15
16 print(len(sx))  # sx の長さを出力
17 plt.plot(range(ns), sx)  # プロットの作成と範囲設定
18 plt.xlabel('k')  # x 軸ラベル
19 plt.ylabel('Sk')  # y 軸ラベル
20 plt.title('Cumulative Sum Plot')  # タイトルの設定
21 plt.show()  # プロットの表示
22
23 print(sx)  # sx を出力
```



$r=\frac{1}{2}$のとき、

$\Sigma_{k=0}^{\infty}\left(\frac{1}{2}\right)^k=\frac{1}{1-\frac{1}{2}}=2$

となるはずで、実際に、S_{21} から 2 となった。なるほど！

```
> sx[1:50]
 [1] 1.000000 1.500000 1.750000 1.875000 1.937500 1.968750 1.984375 1.992188
 [9] 1.996094 1.998047 1.999023 1.999512 1.999756 1.999878 1.999939 1.999969
[17] 1.999985 1.999992 1.999996 1.999998 1.999999 2.000000 2.000000 2.000000
[25] 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000
[33] 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000
[41] 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000
[49] 2.000000 2.000000
```

ノック 59 本目

級数 $1,\ \frac{1}{1},\ \frac{1}{1\cdot2},\ \frac{1}{1\cdot2\cdot3},\ \cdots,\ \frac{1}{n!}$ ついて、

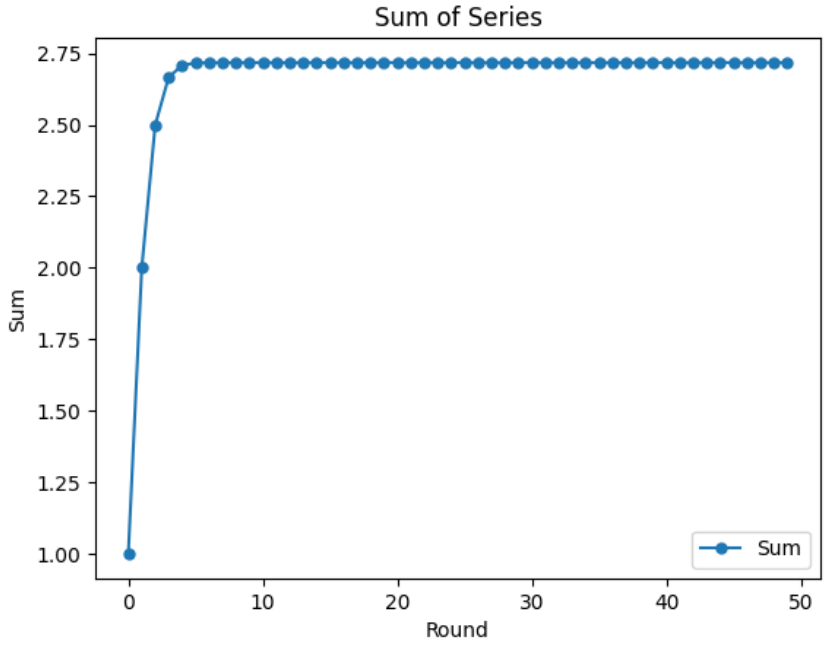
$S_0=1, S_1=1+\frac{1}{1},\ S_2=1+\frac{1}{1}+\frac{1}{1\cdot2},\ S_3=1+\frac{1}{1}+\frac{1}{1\cdot2}+\frac{1}{1\cdot2\cdot3},\ S_{49}=\frac{1}{1}+\frac{1}{1\cdot2}+\frac{1}{1\cdot2\cdot3}+\cdots+\frac{1}{1\cdot2\cdot3\cdot\ldots\cdot49}$、

を求めて、横軸を $0,1,\ldots,49$ 、縦軸を $S_0,S_1,\cdots,S_{49}$ として値をプロットしてみよう。

RK59.py

```
1  import math  # 数学関数のためのライブラリをインポート
2
3  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
4
5  ns = 50  # ステップ数の設定
6  a = [0] * ns  # 長さ ns のリストを 0 で初期化
7  sumv = 0  # 初期値の設定
8
9  for i in range(ns):  # 0 から ns-1 までのループ
```

```
10      sumv += 1 / math.factorial(i)  # 1/factorial(i) を sumv に加
        算
11      a[i] = sumv  # a の i 番目の要素に sumv を設定
12
13  # プロットの作成
14  plt.plot(range(ns), a, marker='o', markersize=5, label='Sum')
15  plt.xlabel('Round')  # x 軸ラベル
16  plt.ylabel('Sum')  # y 軸ラベル
17  plt.title('Sum of Series')  # タイトルの設定
18  plt.legend()  # 凡例の表示
19  plt.show()  # プロットの表示
20
21  print(len(a))  # a の長さを出力
22  print(a[ns - 1])  # a の最後の要素を出力
```




```
> print(len(a))
50
> print(a[ns-1])
2.7182818284590455
```

ネイピア数(数学定数の一つであり、自然対数の底)。

e = 2.71828 18284 59045 23536 02874 71352 …

に近づきます。ちなみにネイピア数の定義は、

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

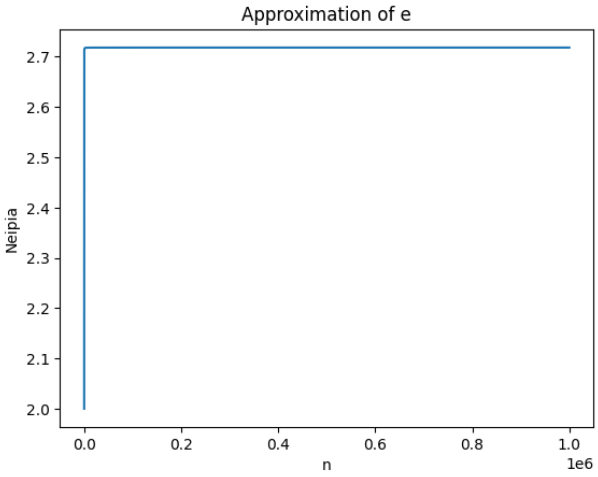
です。

実際にこの式により、$n = 1, 2, \cdots, 1000000$ と $\left(1+\frac{1}{n}\right)^n$ を計算してみましょう（program A を実行してみましょう)。すると、$n = 1000000$ では $\left(1+\frac{1}{n}\right)^n$ は、2.71828、ネイピア数と小数点以下 5 桁で一致します。また、$n = 1, 2, \cdots, 1000000$ のときの $\left(1+\frac{1}{n}\right)^n$ の値は以下のグラフのようになります。

program A

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  n = 1000000  # 計算するステップ数の設定
5  neipia = np.zeros(n)  # 長さ n のリストを 0 で初期化
6
7  for i in range(1, n + 1):  # 1 から n までのループ
8      neipia[i - 1] = (1 + 1 / i) ** i  # ネイピア数の近似値を計算しリストに保存
9
10 # プロットの作成
11 plt.plot(range(1, n + 1), neipia)
12 plt.xlabel('n')  # x 軸ラベル
13 plt.ylabel('Neipia')  # y 軸ラベル
14 plt.title('Approximation of e')  # タイトルの設定
15 plt.show()  # プロットの表示
16
17 print(np.exp(1))  # ネイピア数 e を出力
18 print(neipia[-1])  # neipia の最後の要素を出力
```

```
> print(np.exp(1))
2.718281828459045
> print(neipia[-1])
2.7182804690957534
```



たねあかし

マクローリン展開という便利な方法があります。

$$f(x) = f(0) + \frac{1}{1!}f'(0)x + \frac{1}{2!}f^{(2)}(0)x^2 + \frac{1}{3!}f^{(3)}(0)x^3 + \cdots + \frac{1}{n!}f^{(n)}(0)x^n$$

と展開できます。$f^{(2)}(0)$、$f^{(3)}(0)$、…、$f^{(n)}(0)$はそれぞれ、$f(x)$の 2 階、3 階、…、n 階微分に 0 を代入した値です。

$$f(x) = x^3 + x^2 + x + 1$$

をマクローリン展開をしてみましょう。

$$f'(x) = 3x^2 + 2x + 1$$
$$f^{(2)}(x) = 6x + 2$$
$$f^{(3)}(x) = 6$$

となります。すると

$$f(0) = 1$$
$$f'(0) = 1$$
$$f^{(2)}(0) = 2$$
$$f^{(3)}(0) = 6$$

となり、

$$f(x) = 1 + \frac{1}{1!}1x + \frac{1}{2!}2x^2 + \frac{1}{3!}6x^3 = 1 + x + x^2 + x^3$$

と元の式が再現できます。

続いて

$$f(x) = e^x$$

とすると

$$e^x = 1 + \frac{1}{1!}x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots + \frac{1}{n!}x^n$$

となるので、$x = 1$を代入すると、

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

となります。

ノック 60 本目

級数 $\frac{1}{1}, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \cdots, (-1)^{n-1}\frac{1}{n}$ について、
$S_1 = \frac{1}{1}$, $S_2 = \frac{1}{1} - \frac{1}{2}$, $S_3 = \frac{1}{1} - \frac{1}{2} + \frac{1}{3}$, $S_4 = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4}, \cdots, S_n = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + (-1)^{n-1}\frac{1}{n}$
を $n = 50$ まで求めて、横軸を $1, 2, \ldots, 50$ 、縦軸を $S_1, S_2, \cdots, S_{50}$ として値をプロットしてみよう。

実は、これは $\log_e 2 = 0.69314718$ …に近づきます。はじめて小数第三桁まで一致するときの n を求めよう。

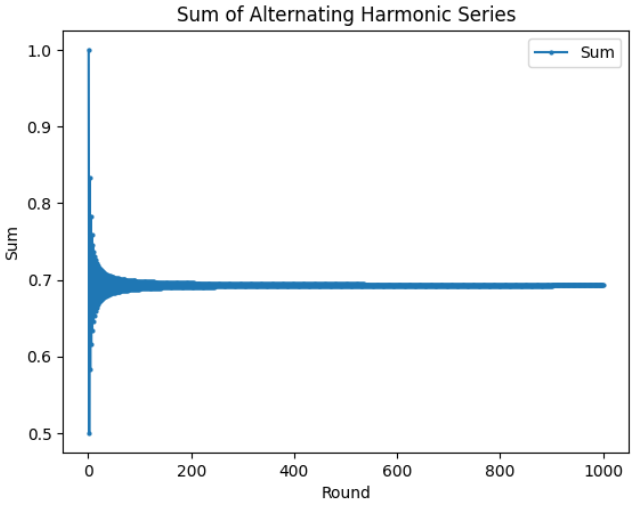
RK60.py

```python
import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
import numpy as np  # 数値計算のためのライブラリをインポート

ns = 1000  # ステップ数の設定
S = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
sumv = 0  # 初期値の設定

for i in range(1, ns + 1):  # 1 から ns までのループ
    addv = ((-1) ** (i - 1)) / i  # 交互級数の計算
    sumv += addv  # 累積和を更新
    S[i - 1] = sumv  # 結果をリストに保存

# プロットの作成
plt.plot(range(1, ns + 1), S, marker='o', markersize=2, label='Sum')
plt.xlabel('Round')  # x 軸ラベル
plt.ylabel('Sum')  # y 軸ラベル
plt.title('Sum of Alternating Harmonic Series')  # タイトルの設定
plt.legend()  # 凡例の表示
plt.show()  # プロットの表示

# 指定された範囲内の値を持つインデックスを検索
select = np.where((0.693 < S) & (S < 0.694))[0]

# 選択されたインデックスの周辺の値を出力
```

```
25 print(S[select[0] - 2])
26 print(S[select[0] - 1])
27 print(S[select[0]])
28 print(S[select[0] + 1])
29 print(S[select[0] + 2])
```

8-11, 22 行目：S[0], S[1]… には 1 番目、2 番目… n 番目までの級数和が格納してあるので、級数和が $0.693 < S[i] < 0.694$ となる級数和を求めて、それが何番目であるかを np.where 関数で使って select に格納する。そこで select[0](=586) が最初にこの条件を満たします。これが「はじめて小数第三桁まで一致するときの n 」と対照します。

24-29 行目：確認のため、select[0] の前後 2 個の級数和を出力した。




```
> print(S[select[0] - 2])
 0.6940011509021636
> print(S[select[0] - 1])
 0.6922946662605254
> print(S[select[0]])
 0.6939982437733022      ←ここではじめて 0.693 と小数三桁が一致する。
> print(S[select[0] + 1])
 0.6922975635011934
> print(S[select[0] + 2])
 0.6939953563704634
```

たねあかし

$\log_e x$ を $\ln x$ と略して書きます。$\ln x$ の微分は $\frac{1}{x}$ となります。また、$\ln(x+1)$ の微分は $\frac{1}{(x+1)}$ となります。

$\frac{1}{x+1}$ をマクローリン展開すると、

$$\frac{1}{x+1} = 1 - x + x^2 - x^3 + x^4 + \cdots$$

これを積分すると

$$\ln(x+1) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \cdots$$

となります。そこで、$x = 1$を代入すると

$$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots$$

となります。

ノック 61 本目

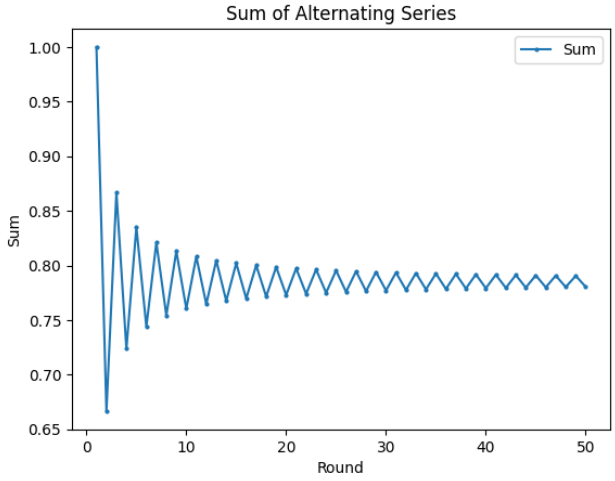
級数$\frac{1}{1}, -\frac{1}{3}, \frac{1}{5}, -\frac{1}{7}, \cdots, (-1)^{n-1}\frac{1}{2n-1}$について、
$S_1 = \frac{1}{1}$, $S_2 = \frac{1}{1} - \frac{1}{3}$, $S_3 = \frac{1}{1} - \frac{1}{3} + \frac{1}{5}$, $S_4 = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7}, \cdots, S_n = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + (-1)^{n-1}\frac{1}{2n-1}$
を$n = 50$まで求めて、横軸を$1, 2, \ldots, 50$、縦軸を$S_1, S_2, \cdots, S_{50}$として値をプロットしてみよう。

実は、これは$\frac{\pi}{4} = 0.78539816\ldots$に近づきます。そこで小数第二桁まで一致するときの$n$を求めよう。

RK61.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  ns = 50  # ステップ数の設定
5  S = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
6  sumv = 0  # 初期値の設定
7
8  for i in range(1, ns + 1):  # 1 から ns までのループ
9      sumv += ((-1) ** (i - 1)) * 1 / (2 * i - 1)  # 交互級数の計算
10     S[i - 1] = sumv  # 結果をリストに保存
11
12 # プロットの作成
13 plt.plot(range(1, ns + 1), S, marker='o', markersize=2,
   label='Sum')
14 plt.xlabel('Round')  # x 軸ラベル
15 plt.ylabel('Sum')  # y 軸ラベル
16 plt.title('Sum of Alternating Series')  # タイトルの設定
17 plt.legend()  # 凡例の表示
18 plt.show()  # プロットの表示
19
```

```
20 print(S[ns - 1])  # S の最後の要素を出力
21 print(np.pi / 4)  # π/4 を出力
```




```
> print(S[ns - 1])
0.7803986631477527
> print(np.pi / 4)
0.7853981633974483
```

たねあかし

$x = \tan y$ とおいて x について微分します。

$$1 = \frac{d}{dx}\tan y = \frac{1}{\cos^2 y}\frac{dy}{dx}$$

すると、

$$\frac{dy}{dx} = \cos^2 y$$

$x = \tan y$ を用いて $\cos^2 y$ を x で表すと

$$\cos^2 y = \frac{1}{1+x^2}$$

となります。

$\frac{1}{x+1}$ をマクローリン展開すると、

$$\frac{1}{x+1} = 1 - x + x^2 - x^3 + x^4 - \cdots$$

これを利用すると、

$$\frac{1}{x^2+1} = 1 - x^2 + x^4 - x^6 + x^8 - \cdots$$

すると

$$\frac{dy}{dx} = \frac{1}{x^2+1} = 1 - x^2 + x^4 - x^6 + x^8 - \cdots$$

となると

$$\int \frac{dy}{dx}dx = \int \frac{1}{x^2+1}dx = \int (1 - x^2 + x^4 - x^6 + x^8 - \cdots)dx = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \frac{1}{9}x^9 - \cdots$$

$$\int \frac{dy}{dx}dx = y$$

結局

$$y = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \frac{1}{9}x^9 - \cdots$$

$x = 1$ とおくと、$x = \tan y$ により、$y = \frac{\pi}{4}$

よって

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

となります。

ノック 62 本目

級数 $\frac{1}{1\cdot3}, \frac{1}{5\cdot7}, \frac{1}{9\cdot11}, \frac{1}{13\cdot15}, \cdots, \frac{1}{(4n-3)(4n-1)}$ について、

$S_1 = \frac{1}{1\cdot3}$, $S_2 = \frac{1}{1\cdot3} + \frac{1}{5\cdot7}$, $S_3 = \frac{1}{1\cdot3} + \frac{1}{5\cdot7} + \frac{1}{9\cdot11}$, $S_4 = \frac{1}{1\cdot3} + \frac{1}{5\cdot7} + \frac{1}{9\cdot11} + \frac{1}{13\cdot15}, \cdots,$

$S_n = \frac{1}{1\cdot3} + \frac{1}{5\cdot7} + \frac{1}{9\cdot11} + \frac{1}{13\cdot15} + \cdots + \frac{1}{(4n-3)(4n-1)}$

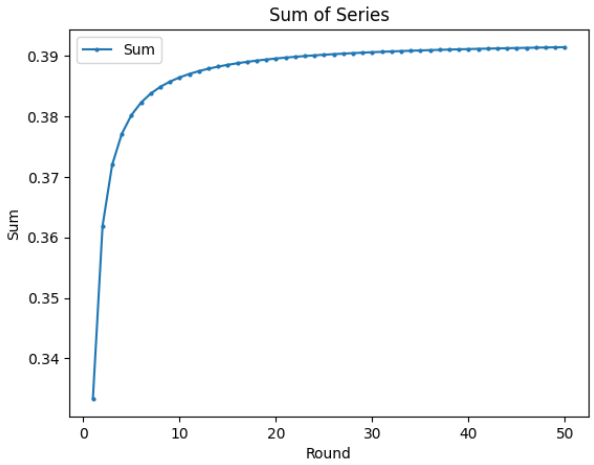
を $n = 50$ まで求めて、横軸を $1, 2, \ldots, 50$ 、縦軸を $S_1, S_2, \cdots, S_{50}$ として値をプロットしてみよう。

実は、これは $\frac{\pi}{2\cdot4} = 0.3926991\ldots$ に近づきます。そこで小数第二桁まで一致するときの n を求めよう。

RK62.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  ns = 50  # ステップ数の設定
5  S = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
6  sumv = 0  # 初期値の設定
7
8  for i in range(1, ns + 1):  # 1 から ns までのループ
9      sumv += 1 / ((4 * i - 3) * (4 * i - 1))  # 特定の級数の計算
10     S[i - 1] = sumv  # 結果をリストに保存
11
12 # プロットの作成
13 plt.plot(range(1, ns + 1), S, marker='o', markersize=2,
   label='Sum')
14 plt.xlabel('Round')  # x 軸ラベル
15 plt.ylabel('Sum')  # y 軸ラベル
16 plt.title('Sum of Series')  # タイトルの設定
17 plt.legend()  # 凡例の表示
```

```
18  plt.show()  # プロットの表示
19
20  print(S[ns - 1])  # S の最後の要素を出力
21  print(np.pi / (2 * 4))  # π/(2*4) を出力
```




```
> print(S[ns - 1])
0.391449112944819
> print(np.pi / (2 * 4))
0.39269908169872414
```

たねあかし

$\frac{1}{1\cdot3}=\frac{1}{2}\left(\frac{1}{1}-\frac{1}{3}\right)$、
$\frac{1}{5\cdot7}=\frac{1}{2}\left(\frac{1}{5}-\frac{1}{7}\right)$、
$\frac{1}{9\cdot11}=\frac{1}{2}\left(\frac{1}{9}-\frac{1}{11}\right)$、
…

を足すと

$$\frac{1}{1\cdot3}+\frac{1}{5\cdot7}+\frac{1}{9\cdot11}+\cdots=\frac{1}{2}\left(\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots\right)$$

ノック 61 本目のたねあかしより

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots$$

を用いると

$$\frac{1}{1\cdot3}+\frac{1}{5\cdot7}+\frac{1}{9\cdot11}+\cdots=\frac{1}{2}\cdot\frac{\pi}{4}$$

ノック 63 本目

(1) $S_n=\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\cdots+\frac{1}{n^2}$

(2) $S_n = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \cdots + \frac{1}{(2n-1)^2}$

(3) $S_n = \frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{6^2} + \cdots + \frac{1}{(2n)^2}$

について $n = 50$ まで求めて、横軸を $1, 2, \ldots, 50$ 、縦軸を $S_1, S_2, \cdots, S_{50}$ として値をプロットしてみよう。

$$\frac{\pi^2}{6} = 1.625133$$
$$\frac{\pi^2}{8} = 1.233701$$
$$\frac{\pi^2}{24} = 0.4112335$$

に近づきます。

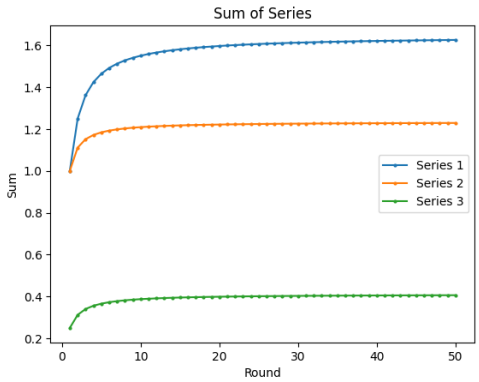
RK63.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  ns = 50  # ステップ数の設定
5  S = np.zeros((ns, 3))  # 長さ ns の 3 列の行列を 0 で初期化
6  sumv1 = 0  # 初期値の設定
7  sumv2 = 0  # 初期値の設定
8  sumv3 = 0  # 初期値の設定
9
10 for i in range(1, ns + 1):  # 1 から ns までのループ
11     sumv1 += 1 / (i ** 2)  # シリーズ 1 の計算
12     S[i - 1, 0] = sumv1  # 結果を行列に保存
13
14     sumv2 += 1 / ((2 * i - 1) ** 2)  # シリーズ 2 の計算
15     S[i - 1, 1] = sumv2  # 結果を行列に保存
16
17     sumv3 += 1 / ((2 * i) ** 2)  # シリーズ 3 の計算
18     S[i - 1, 2] = sumv3  # 結果を行列に保存
19
20 # プロットの作成
21 plt.plot(range(1, ns + 1), S[:, 0], marker='o', markersize=2,
   label='Series 1')
22 plt.plot(range(1, ns + 1), S[:, 1], marker='o', markersize=2,
   label='Series 2')
```

```
23  plt.plot(range(1, ns + 1), S[:, 2], marker='o', markersize=2,
    label='Series 3')
24  plt.xlabel('Round')  # x 軸ラベル
25  plt.ylabel('Sum')  # y 軸ラベル
26  plt.title('Sum of Series')  # タイトルの設定
27  plt.legend()  # 凡例の表示
28  plt.show()  # プロットの表示
29
30  print(S[ns - 1, 0])  # S の最後の要素を出力（シリーズ 1）
31  print(np.pi ** 2 / 6)  # π^2/6 を出力
32
33  print(S[ns - 1, 1])  # S の最後の要素を出力（シリーズ 2）
34  print(np.pi ** 2 / 8)  # π^2/8 を出力
35
36  print(S[ns - 1, 2])  # S の最後の要素を出力（シリーズ 3）
37  print(np.pi ** 2 / 24)  # π^2/24 を出力
```

15-19 行目：(1)、(2)、(3)の S_n の値をそれぞれ、行列 S の 1-3 列目に格納した。行は i＝0,1,2,…,ns-1 です。

```
> print(S[ns - 1, 0])
1.625132733621529
> print(np.pi ** 2 / 6)
1.6449340668482264
> print(S[ns - 1, 1])
1.2287007167795103
> print(np.pi ** 2 / 8)
1.2337005501361697
> print(S[ns - 1, 2])
0.4062831834053823
> print(np.pi ** 2 / 24)
0.4112335167120566
```



たねあかしはノック 65 本目の説明を参考にしてください。

ノック 64 本目

$$S_n = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \cdots + (-1)^{n-1}\frac{1}{n^2}$$

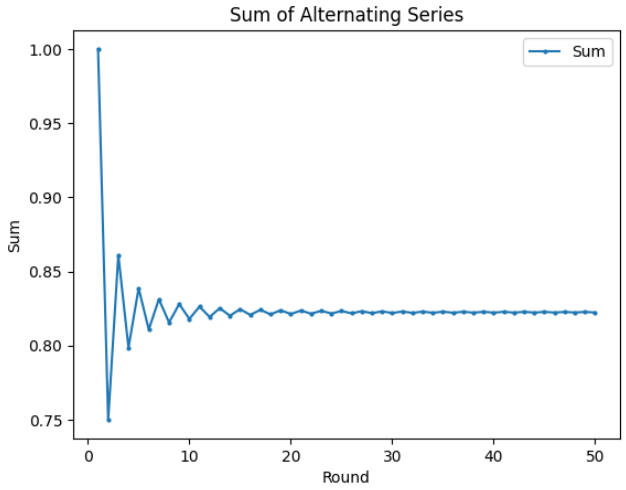
について $n = 50$ まで求めて、横軸を $1, 2, \ldots, 50$、縦軸を $S_1, S_2, \cdots, S_{50}$ として値をプロットしてみよう。

実は、$\frac{\pi^2}{12} = 0.822467$

に近づきます。そこで小数第三桁まで一致するときの n を求めよう。

RK64.py

```python
import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
import numpy as np  # 数値計算のためのライブラリをインポート

ns = 50  # ステップ数の設定
S = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
sumv = 0  # 初期値の設定

for i in range(1, ns + 1):  # 1 から ns までのループ
    sumv += ((-1) ** (i - 1)) / (i ** 2)  # 交互級数の計算
    S[i - 1] = sumv  # 結果をリストに保存

# プロットの作成
plt.plot(range(1, ns + 1), S, marker='o', markersize=2, label='Sum')
plt.xlabel('Round')  # x 軸ラベル
plt.ylabel('Sum')  # y 軸ラベル
plt.title('Sum of Alternating Series')  # タイトルの設定
plt.legend()  # 凡例の表示
plt.show()  # プロットの表示

print(S[ns - 1])  # S の最後の要素を出力
print(np.pi ** 2 / 12)  # π^2/12 を出力
```




```
> print(S[ns - 1])
0.8222710318260295
> print(np.pi ** 2 / 12)
0.8224670334241132
```

ノック 65 本目

$$S_n = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \cdots + (-1)^{n-1}\frac{1}{n^2}$$

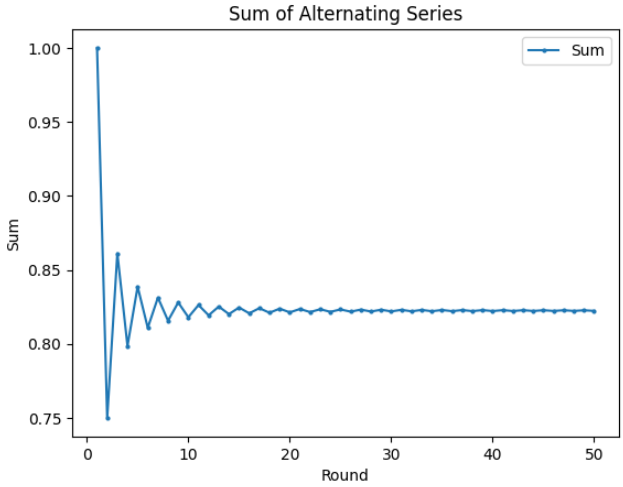
について $n = 50$ まで求めて、横軸を $1, 2, \ldots, 50$、縦軸を $S_1, S_2, \cdots, S_{50}$ として値をプロットしてみよう。

実は、$\frac{\pi^2}{12} = 0.822467$ に近づきます。

RK65.py

```python
import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
import numpy as np  # 数値計算のためのライブラリをインポート

ns = 50  # ステップ数の設定
S = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
sumv = 0  # 初期値の設定

for i in range(1, ns + 1):  # 1 から ns までのループ
    sumv += ((-1) ** (i - 1)) / (i ** 2)  # 交互級数の計算
    S[i - 1] = sumv  # 結果をリストに保存

# プロットの作成
```

```
13  plt.plot(range(1, ns + 1), S, marker='o', markersize=2,
    label='Sum')
14  plt.xlabel('Round')  # x 軸ラベル
15  plt.ylabel('Sum')  # y 軸ラベル
16  plt.title('Sum of Alternating Series')  # タイトルの設定
17  plt.legend()  # 凡例の表示
18  plt.show()  # プロットの表示
19
20  print(S[ns - 1])  # S の最後の要素を出力
21  print(np.pi ** 2 / 12)  # π^2/12 を出力
```




```
> print(S[ns - 1])
0.8222710318260295
> print(np.pi ** 2 / 12)
0.8224670334241132
```

たねあかし

(1) $\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\cdots=\frac{\pi^2}{6}$ を示してみよう。

例えば、

$$(a+x)^n = a_0x + a_1x^1 + a_2x^2 + a_3x^3 + \cdots + a_nx^n$$

のように、展開できる。初項を 1 としてこの発想を使うと、

$$f(x) = a_0 + a_1x^1 + a_2x^2 + a_3x^3 + \cdots + x^n$$

となり、

$$g(x) = 1 + b_1x^1 + b_2x^2 + b_3x^3 + \cdots + b_nx^n$$

とおくと、これは

$$f(x) = (x-r_1)(x-r_2)\cdots(x-r_n)$$

と書くことができる。ここで、

$$g(x) = \left(1-\frac{x}{r_1}\right)\left(1-\frac{x}{r_2}\right)\cdots\left(1-\frac{x}{r_n}\right)$$

とも書ける。

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

であるので

$$\frac{\sin x}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \cdots$$

一方で

$$\frac{\sin x}{x} = \left(1 - \frac{x^2}{r_1}\right)\left(1 - \frac{x^2}{r_2}\right)\ldots$$

と書けます。

x^2の項に注目すると

$$-\frac{1}{3!} = -\left(\frac{1}{r_1} + \frac{1}{r_2} + \cdots\right)$$

ここで、$r_1 = \pi^2, r_2 = (2\pi)^2, r_2 = (3\pi)^2, \cdots$

すると

$$-\frac{1}{3!} = -\left(\frac{1}{r_1} + \frac{1}{r_2} + \cdots\right) = -\left(\frac{1}{\pi^2} + \frac{1}{(2\pi)^2} + \frac{1}{(3\pi)^2}\ldots\right) = -\frac{1}{\pi^2}\left(1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots\right)$$

つまり

$$\mathbf{1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots = \frac{\pi^2}{3!} = \frac{\pi^2}{6}}$$

(2) $\mathbf{1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \cdots = \frac{\pi^2}{8}}$ を示してみよう。

$1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots$を偶数と奇数の項に分ける

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots = \left(1 + \frac{1}{3^2} + \frac{1}{5^2} + \cdots\right) + \left(\frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{6^2}\ldots\right)$$
$$= \left(1 + \frac{1}{3^2} + \frac{1}{5^2} + \cdots\right) + \frac{1}{2^2}\left(\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2}\cdots\right)$$

となり、

$$\frac{\pi^2}{6} = \left(1 + \frac{1}{3^2} + \frac{1}{5^2} + \cdots\right) + \frac{1}{4}\cdot\frac{\pi^2}{6}$$

すなわち

$$\left(1 + \frac{1}{3^2} + \frac{1}{5^2} + \cdots\right) = \frac{\pi^2}{6} - \frac{\pi^2}{24} = \frac{3\pi^2}{24} = \frac{\pi^2}{8}$$

(3) $\frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{6^2} + \cdots = \frac{\pi^2}{24}$ を示してみよう。

(1)と(2)を利用して

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots = \left(\frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \cdots\right) + \frac{1}{4}\left(\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2}\ldots\right)$$
$$\frac{3}{4}\left(\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2}\ldots\right) = \frac{\pi^2}{8}$$
$$\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2}\ldots = \frac{\pi^2}{6}$$

両辺を、$\frac{1}{2^2}$倍すると、

$$\frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{6^2}\ldots = \frac{\pi^2}{24}$$

(4) $\frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{34^2}\ldots = \frac{\pi^2}{12}$ を示してみよう。

(2)と(3)を利用して

$$\left(1 + \frac{1}{3^2} + \frac{1}{5^2} + \cdots\right) - \left(\frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{6^2} + \cdots\right) = \frac{\pi^2}{8} - \frac{\pi^2}{24} = \frac{\pi^2}{12}$$

ノック 66 本目

$a_1 = \frac{\sqrt{2}}{2},\ a_2 = \frac{\sqrt{2+\sqrt{2}}}{2},\ a_3 = \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2},\ \cdots$

とする。

$S_n = a_1 a_2 \cdots a_n$

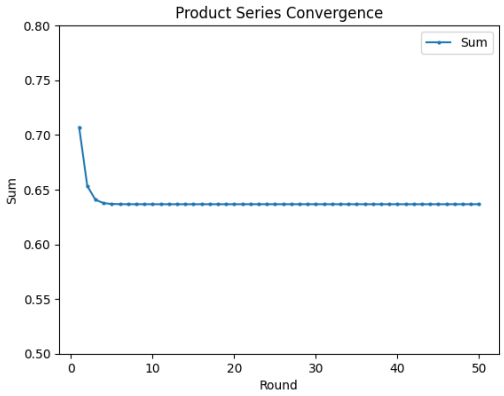
について $n = 50$ まで求めて、横軸を $1, 2, \ldots, 50$、縦軸を $S_1, S_2, \cdots, S_{50}$ として値をプロットしてみよう。

実は、$\frac{2}{\pi} = 0.6366198$ に近づきます。そこで小数第三桁まで一致するときの n を求めよう。

RK66.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  ns = 50  # ステップ数の設定
5  a = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
6
7  a[0] = np.sqrt(2)  # 最初の要素を √2 に設定
8  for i in range(1, ns):  # 1 から ns-1 までのループ
9      a[i] = np.sqrt(2 + a[i - 1])  # a[i] を計算
10
11 S = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
12 mulv = 1  # 初期値の設定
13 for i in range(ns):  # 0 から ns-1 までのループ
14     mulv *= a[i]  # mulv に a[i] を掛ける
15     S[i] = mulv / (2 ** (i + 1))  # S[i] に計算結果を保存
16
17 # プロットの作成
18 plt.plot(range(1, ns + 1), S, marker='o', markersize=2,
   label='Sum')
19 plt.xlabel('Round')  # x 軸ラベル
20 plt.ylabel('Sum')  # y 軸ラベル
21 plt.ylim(0.5, 0.8)  # y 軸の範囲を設定
22 plt.title('Product Series Convergence')  # タイトルの設定
23 plt.legend()  # 凡例の表示
24 plt.show()  # プロットの表示
```

```
25
26 print(S[ns - 1])  # S の最後の要素を出力
27 print(2 / np.pi)  # 2/π を出力
```




```
> print(S[ns - 1])
0.6366197723675812
> print(2 / np.pi)
0.6366197723675814
```

たねあかし

三角関数の２倍角の公式を使ってみよう。

$$\sin 2y = 2\cos y \sin y$$

この式に $2y = x$ を代入すると、

$$\sin x = 2\cos \frac{x}{2} \sin \frac{x}{2}$$

さらに$\sin \frac{x}{2}$をじゃんじゃか半角にしていくと

$$\sin x = 2\cos \frac{x}{2}\left(2\cos \frac{x}{2^2} \sin \frac{x}{2^2}\right) = 2\cos \frac{x}{2}\left(2\cos \frac{x}{2^2}\left(2\cos \frac{x}{2^3} \sin \frac{x}{2^3}\right)\right)$$

$$= 2^n \cdot \cos \frac{x}{2} \cos \frac{x}{2^2} \cdots \cos \frac{x}{2^n} \cdot \sin \frac{x}{2^n}$$

$$\lim_{n\to\infty} \frac{\sin x}{x} = \lim_{n\to\infty} \frac{2^n \cdot \cos \frac{x}{2} \cos \frac{x}{2^2} \cdots \cos \frac{x}{2^n} \cdot \sin \frac{x}{2^n}}{x} = \lim_{n\to\infty} \frac{\cos \frac{x}{2} \cos \frac{x}{2^2} \cdots \cos \frac{x}{2^n} \cdot \sin \frac{x}{2^n}}{\frac{x}{2^n}}$$

$$= \lim_{n\to\infty} \left(\cos \frac{x}{2} \cos \frac{x}{2^2} \cdots \cos \frac{x}{2^n}\right) \frac{\sin \frac{x}{2^n}}{\frac{x}{2^n}}$$

$$= \cos \frac{x}{2} \cos \frac{x}{2^2} \cdots \cos \frac{x}{2^n}$$

よって

$$\frac{\sin x}{x} = \cos \frac{x}{2} \cos \frac{x}{2^2} \cdots \cos \frac{x}{2^n}$$

$x = \frac{\pi}{2}$を代入すると

$$\cos \frac{\pi}{2} = \frac{\sqrt{2}}{2}$$

$$\cos \frac{\pi}{2^2} = \frac{\sqrt{2+\sqrt{2}}}{2}$$

$\cos\frac{\pi}{2^3}=\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}$

…

$\cos\frac{\pi}{2^n}$ の計算には、半角の公式 $\cos\frac{x}{2}=\frac{\sqrt{2+2\cos x}}{2}$ を使いましょう。

よって

$\frac{2}{\pi}=\frac{\sqrt{2}}{2}\frac{\sqrt{2+\sqrt{2}}}{2}\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}\cdots$

ノック 67 本目

$a_1=\frac{2\cdot2}{1\cdot3}$, $a_2=\frac{4\cdot4}{3\cdot5}$, $a_3=\frac{6\cdot6}{5\cdot7}$, $\cdots$, $a_n=\frac{(2n)\cdot(2n)}{(2n-1)(2n+1)}$,

とする。

$S_n=a_1a_2\cdots a_n$

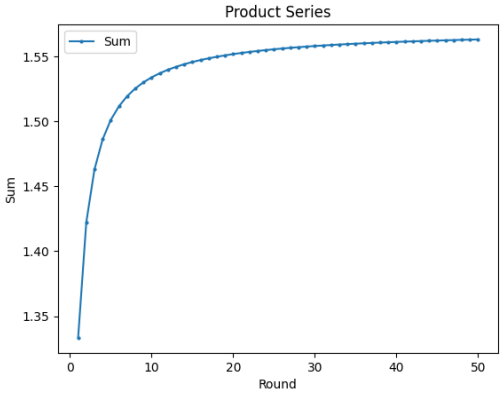
について $n=50$ まで求めて、横軸を $1,2,\ldots,50$ 、縦軸を $S_1,S_2,\cdots,S_{50}$ として値をプロットしてみよう。

実は、$\frac{\pi}{2}$ =1.570796 に近づきます。

RK67.py

```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  ns = 50  # ステップ数の設定
5  a = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
6
7  for i in range(1, ns + 1):  # 1 から ns までのループ
8      a[i - 1] = 2 * i * 2 * i / ((2 * i - 1) * (2 * i + 1))  # a[i] を計算
9
10 S = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
11 mulv = 1  # 初期値の設定
12 for i in range(ns):  # 0 から ns-1 までのループ
13     mulv *= a[i]  # mulv に a[i] を掛ける
14     S[i] = mulv  # S[i] に計算結果を保存
```

```
15
16  # プロットの作成
17  plt.plot(range(1, ns + 1), S, marker='o', markersize=2,
18  label='Sum')
19  plt.xlabel('Round')  # x 軸ラベル
20  plt.ylabel('Sum')  # y 軸ラベル
21  plt.title('Product Series')  # タイトルの設定
22  plt.legend()  # 凡例の表示
23  plt.show()  # プロットの表示
24
25  print(S[ns - 1])  # S の最後の要素を出力
26  print(np.pi / 2)  # π/2 を出力
```




```
> print(S[ns - 1])
1.5630394501077054
> print(np.pi / 2)
1.5707963267948966
```

たねあかし

ノック 65 のたねあかしより

$$\frac{\sin x}{x} = \left(1 - \frac{x^2}{r_1}\right)\left(1 - \frac{x^2}{r_2}\right)\cdots$$

ここで、$r_1 = \pi^2, r_2 = (2\pi)^2,\ r_2 = (3\pi)^2, \cdots$

すなわち

$$\frac{\sin x}{x} = \left(1 - \frac{x^2}{\pi^2}\right)\left(1 - \frac{x^2}{(2\pi)^2}\right)\left(1 - \frac{x^2}{(3\pi)^2}\right)\cdots$$

$x = \frac{\pi}{2}$ を代入すると

$$\left(1 - \frac{\left(\frac{\pi}{2}\right)^2}{\pi^2}\right) = \frac{3}{4} = \frac{1\cdot 3}{2\cdot 2}$$

$$\left(1 - \frac{\left(\frac{\pi}{2}\right)^2}{(2\pi)^2}\right) = \frac{15}{16} = \frac{3\cdot 5}{4\cdot 4}$$

$$\left(1 - \frac{\left(\frac{\pi}{2}\right)^2}{(3\pi)^2}\right) = \frac{35}{36} = \frac{5\cdot 7}{6\cdot 6}$$

…

$$\frac{2}{\pi}=\frac{1\cdot3}{2\cdot2}\cdot\frac{3\cdot5}{4\cdot4}\cdot\frac{5\cdot7}{6\cdot6}\cdots$$

よって

$$\frac{\pi}{2}=\frac{2\cdot2}{1\cdot3}\cdot\frac{4\cdot4}{3\cdot5}\cdot\frac{6\cdot6}{5\cdot7}\cdots$$

ノック 68 本目

$$S_n=\frac{3\cdot3}{2\cdot4}\cdot\frac{6\cdot6}{5\cdot7}\cdot\frac{9\cdot9}{8\cdot10}\cdots\frac{(3n)(3n)}{(3n-1)(3n+1)}$$
$$S_n=\frac{4\cdot4}{3\cdot5}\cdot\frac{8\cdot8}{7\cdot9}\cdot\frac{12\cdot12}{11\cdot13}\cdots\frac{(4n)(4n)}{(4n-1)(4n+1)}$$
$$S_n=\frac{6\cdot6}{5\cdot7}\cdot\frac{12\cdot12}{11\cdot13}\cdot\frac{18\cdot18}{17\cdot19}\cdots\frac{(6n)(6n)}{(6n-1)(6n+1)}$$

について $n=50$ まで求めて、横軸を $1,2,\ldots,50$ 、縦軸を $S_1,S_2,\cdots,S_{50}$ として値をプロットしてみよう。

上から順に、

$\frac{2\pi}{3\sqrt{3}}$ =1.2092

$\frac{\pi}{2\sqrt{2}}$ =1.110721

$\frac{\pi}{3}$ =1.047198

に近づきます。そこで小数第三桁まで一致するときの n を求めよう。

これは各自でプログラミングしてみてください。

たねあかし

ノック 67 のたねあかしより

$$x=\frac{\pi}{3},\ \frac{\pi}{4},\ \frac{\pi}{6}$$

を代入する。

ノック 69 本目

$$\tan(x)=\frac{1}{5}$$
$$\tan(y)=\frac{1}{239}$$

となる x と y により、

$$4(4x-y)=\pi$$

となる。ここで

$$x=\tan^{-1}\left(\frac{1}{5}\right)=\arctan\left(\frac{1}{5}\right)$$
$$y=\tan^{-1}\left(\frac{1}{239}\right)=\arctan\left(\frac{1}{239}\right)$$

と表す。Python では、math.atan(1/5), math.atan(1/239)とすると求められる。プログラムを書いてみよう。

RK69.py

```
import math  # 数学関数のためのライブラリをインポート

# 値を計算
result = 4 * (4 * math.atan(1/5) - math.atan(1/239))

# 結果を出力
print(result)
```

> print(result)
3.1415926535897936

ノック 70 本目

$$S_n = \frac{1}{2^2}\tan\left(\frac{\pi}{2^2}\right) + \frac{1}{2^3}\tan\left(\frac{\pi}{2^3}\right) + \cdots + \frac{1}{2^{n+1}}\tan\left(\frac{\pi}{2^{n+1}}\right)$$

について $n = 50$ まで求めて、横軸を $1, 2, \ldots, 50$ 、縦軸を $S_1, S_2, \cdots, S_{50}$ として値をプロットしてみよう。

これは

$$\frac{1}{\pi} = 1.11072$$

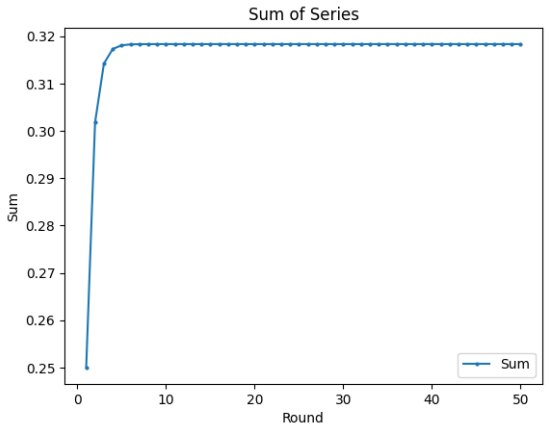
に近づきます。そこで小数第三桁まで一致するときの n を求めよう。

RK70.py

```
import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
import numpy as np  # 数値計算のためのライブラリをインポート

ns = 50  # ステップ数の設定
S = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
sumv = 0  # 初期値の設定

# 各ステップでの累積和の計算
for i in range(1, ns + 1):
    sumv += 1 / (2 ** (i + 1)) * np.tan(np.pi / 2 ** (i + 1))
    S[i - 1] = sumv

# プロットの作成
```

```
14  plt.plot(range(1, ns + 1), S, marker='o', markersize=2,
    label='Sum')
15  plt.xlabel('Round')  # x 軸ラベル
16  plt.ylabel('Sum')  # y 軸ラベル
17  plt.title('Sum of Series')  # タイトルの設定
18  plt.legend()  # 凡例の表示
19  plt.show()  # プロットの表示
20
21  # 結果の出力
22  print(1 / np.pi)
23  print(S[ns - 1])
```

```
> print(1 / np.pi)
0.3183098861837907
> print(S[ns - 1])
0.31830988618379075
```



たねあかし

$$\frac{1}{\tan A}=\frac{1-\tan^2\left(\frac{A}{2}\right)}{2\tan\left(\frac{A}{2}\right)}=\frac{1}{2\tan\left(\frac{A}{2}\right)}-\frac{1}{2}\tan\left(\frac{A}{2}\right)$$
$$=\frac{1}{2^2\tan\left(\frac{A}{2^2}\right)}-\frac{1}{2^2}\tan\left(\frac{A}{2^2}\right)-\frac{1}{2}\tan\left(\frac{A}{2}\right)$$
$$\cdots\cdots$$
$$=\frac{1}{2^n\tan\left(\frac{A}{2^n}\right)}-\frac{1}{2^n}\tan\left(\frac{A}{2^n}\right)-\cdots-\frac{1}{2}\tan\left(\frac{A}{2}\right)$$

ここで、$\frac{1}{2^n\tan\left(\frac{A}{2^n}\right)}=\frac{1}{A}\cdot\frac{A}{2^n}\cdot\frac{1}{\tan\left(\frac{A}{2^n}\right)}=\frac{1}{A}\cdot\frac{A}{2^n}\cdot\frac{\cos\left(\frac{A}{2^n}\right)}{\sin\left(\frac{A}{2^n}\right)}=\frac{1}{A}\cdot\frac{\cos\left(\frac{A}{2^n}\right)}{\frac{\sin\left(\frac{A}{2^n}\right)}{\frac{A}{2^n}}}$

$n\to\infty$ のとき、$\frac{\cos\left(\frac{A}{2^n}\right)}{\sin\left(\frac{A}{2^n}\right)}\to 1$、$\frac{\sin\left(\frac{A}{2^n}\right)}{\frac{A}{2^n}}\to 1$

となるので、

$$\frac{1}{2^n\tan\left(\frac{A}{2^n}\right)}=\frac{1}{A}$$

$A=\frac{\pi}{4}$ とおくと $\frac{1}{2^n\tan\left(\frac{A}{2^n}\right)}=\frac{4}{\pi}$、また $\frac{1}{\tan\left(\frac{\pi}{4}\right)}=\tan\left(\frac{\pi}{4}\right)$

$$\tan\left(\frac{\pi}{4}\right)+\frac{1}{2}\tan\left(\frac{\pi}{8}\right)+\frac{1}{4}\tan\left(\frac{\pi}{16}\right)+\cdots+\frac{1}{2^n}\tan\left(\frac{\pi}{2^{n+2}}\right)=\frac{4}{\pi}$$

$$\frac{1}{4}\tan\left(\frac{\pi}{4}\right)+\frac{1}{8}\tan\left(\frac{\pi}{8}\right)+\frac{1}{16}\tan\left(\frac{\pi}{16}\right)+\cdots+\frac{1}{2^{n+2}}\tan\left(\frac{\pi}{2^{n+2}}\right)=\frac{1}{\pi}$$

ノック 71 本目

$$e^x=\left(1+\frac{x}{n}\right)^n$$

一方で、

$$e^x=1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots$$

となることが知られている。では、$x=1$ としたときの

$$a_n=\left(1+\frac{1}{n}\right)^n$$

と

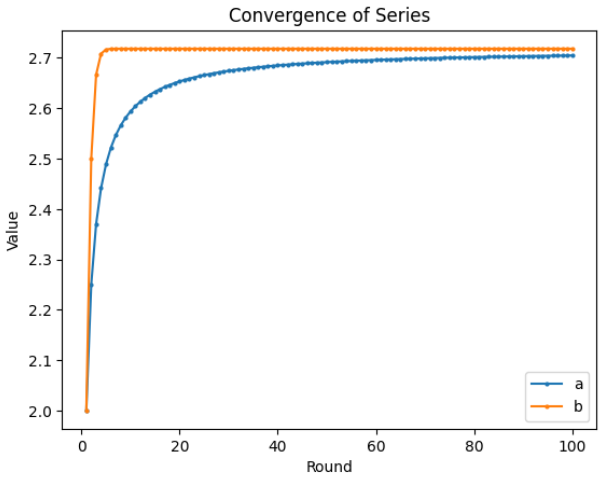
$$b_n=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}$$

について、$n=1,2,\cdots,100$ としたときの a_n と b_n を求めてみよう。すると、$e=2.718282$ に近づく。そこで a_n と b_n について$e=2.718282$ と小数第三桁まで一致するときの n を求めよう。

RK71.py

```python
import math  # 数学関数のためのライブラリをインポート

import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート

ns = 100  # ステップ数の設定
a = [0] * ns  # 長さ ns のリストを 0 で初期化
b = [0] * ns  # 長さ ns のリストを 0 で初期化
sv = 1  # 初期値の設定

# 各ステップでの計算
for i in range(1, ns + 1):
    a[i - 1] = (1 + 1 / i) ** i  # a[i - 1] の計算
    sv += 1 / math.factorial(i)  # 累積和 sv の計算
    b[i - 1] = sv  # b[i - 1] に sv を保存

# プロットの作成
plt.plot(range(1, ns + 1), a, marker='o', markersize=2, label='a')
```

```
18  plt.plot(range(1, ns + 1), b, marker='o', markersize=2,
    label='b')
19  plt.xlabel('Round')  # x 軸ラベル
20  plt.ylabel('Value')  # y 軸ラベル
21  plt.title('Convergence of Series')  # タイトルの設定
22  plt.legend()  # 凡例の表示
23  plt.show()  # プロットの表示
24
25  # 結果の出力
26  print(a[ns - 1])  # a の最後の要素を出力
27  print(b[ns - 1])  # b の最後の要素を出力
28  print(math.exp(1))  # e を出力
```




```
> print(a[ns - 1])
2.7048138294215285
> print(b[ns - 1])
2.7182818284590455
> print(math.exp(1))
2.718281828459045
```

ノック 72 本目

$$b_n = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$
$$c_n = \frac{1}{1!} + \frac{2}{2!} + \frac{3}{3!} + \cdots + \frac{n}{n!}$$

について、まあ、明らかに、b_n と c_n も $n \to \infty$ のとき、$e = 2.718282$ に近づく。

自力で作ってみよう！

たねあかし

$$c_n = \frac{1}{1!} + \frac{2}{2!} + \frac{3}{3!} + \cdots + \frac{n}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!} = b_n$$

となります。

ノック 73 本目

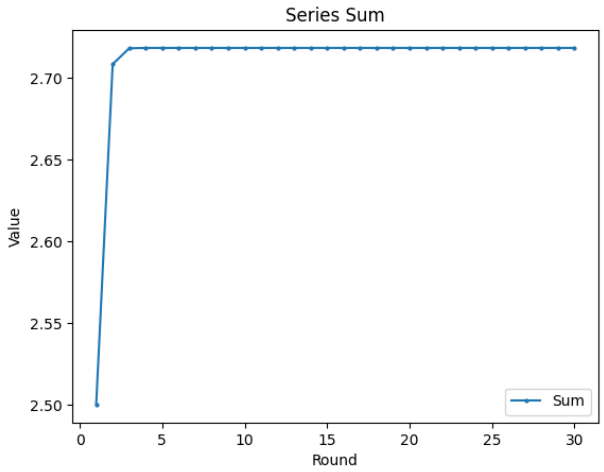
$$d_n = 1 + \frac{3}{2!} + \frac{5}{4!} + \cdots + \frac{2n+1}{(2n)!} = 1 + \sum_{i=1}^{n} \frac{(2i+1)}{(2i)!}$$

についてd_nも$n \to \infty$のとき、$e = 2.718282$ に近づく。

RK73.py

```
1  import math  # 数学関数のためのライブラリをインポート
2
3  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
4
5  ns = 30  # ステップ数の設定
6  d = [0] * ns  # 長さ ns のリストを 0 で初期化
7
8  sv1 = 1  # 初期値の設定
9
10 # 各ステップでの計算
11 for i in range(1, ns + 1):
12     sv1 += (2 * i + 1) / math.factorial(2 * i)  # 累積和 sv1 の計算
13     d[i - 1] = sv1  # d[i - 1] に sv1 を保存
14
15 # プロットの作成
16 plt.plot(range(1, ns + 1), d, marker='o', markersize=2, label='Sum')
17 plt.xlabel('Round')  # x 軸ラベル
18 plt.ylabel('Value')  # y 軸ラベル
19 plt.title('Series Sum')  # タイトルの設定
```

```
20  plt.legend()  # 凡例の表示
21  plt.show()  # プロットの表示
22
23  # 結果の出力
24  print(d[ns - 1])  # d の最後の要素を出力
25  print(math.exp(1))  # e を出力
```



> print(d[ns - 1])

2.7182818284590455

> print(math.exp(1))

2.718281828459045

たねあかし

$$d_n = 1 + \frac{3}{2!} + \frac{5}{4!} + \cdots + \frac{2n+1}{(2n)!} = 1 + \sum_{i=1}^{\infty} \frac{(2n+1)}{(2n)!}$$
$$= 1 + \sum_{i=1}^{\infty} \left[\frac{2n}{(2n)!} + \frac{1}{(2n)!}\right]$$
$$= 1 + \sum_{i=1}^{\infty} \left[\frac{1}{(2n-1)!} + \frac{1}{(2n)!}\right]$$
$$= \left(\frac{1}{1!} + \frac{1}{3!} + \cdots\right) + \left(\frac{1}{2!} + \frac{1}{4!} + \cdots\right)$$
$$= \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$
$$= e$$

ノック 74 本目

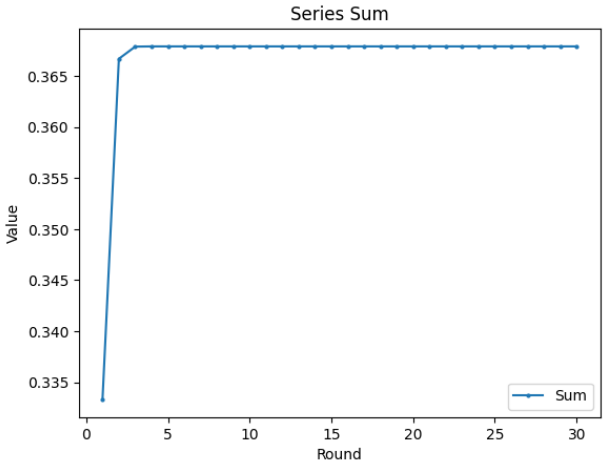
$$a_n = \frac{2}{3!} + \frac{4}{5!} + \frac{6}{7!} + \cdots + \frac{2n}{(2n+1)!}$$

について a_n も $n \to \infty$ のとき、$\frac{1}{e} = \frac{1}{2.718282} = 0.3678794$ に近づく。

RK74.py

```
import math  # 数学関数のためのライブラリをインポート

import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート

ns = 30  # ステップ数の設定
a = [0] * ns  # 長さ ns のリストを 0 で初期化

sv1 = 0  # 初期値の設定

# 各ステップでの計算
for i in range(1, ns + 1):
    sv1 += 2 * i / math.factorial(2 * i + 1)  # 累積和 sv1 の計算
    a[i - 1] = sv1  # a[i - 1] に sv1 を保存

# プロットの作成
plt.plot(range(1, ns + 1), a, marker='o', markersize=2, label='Sum')
plt.xlabel('Round')  # x 軸ラベル
plt.ylabel('Value')  # y 軸ラベル
plt.title('Series Sum')  # タイトルの設定
plt.legend()  # 凡例の表示
plt.show()  # プロットの表示

# 結果の出力
print(a[ns - 1])  # a の最後の要素を出力
print(1 / math.exp(1))  # 1/e を出力
```

```
> print(a[ns - 1])
0.36787944117144233
> print(1 / math.exp(1))
0.36787944117144233
```



たねあかし

$$a_n = \frac{2}{3!} + \frac{4}{5!} + \frac{6}{7!} + \cdots + \frac{2n}{(2n+1)!}$$

$$= \sum_{i=1}^{\infty} \left[\frac{2n}{(2n+1)!} \right]$$

$$= \sum_{i=1}^{\infty} \left[\frac{2n+1}{(2n+1)!} - \frac{1}{(2n+1)!} \right]$$

$$= \sum_{i=1}^{\infty} \left[\frac{1}{(2n)!} - \frac{1}{(2n+1)!} \right]$$

$$= \left(\frac{1}{2!} + \frac{1}{4!} + \cdots \right) - \left(\frac{1}{3!} + \frac{1}{5!} + \cdots \right)$$

$$= \left(\frac{1}{1!} + \frac{1}{2!} + \frac{1}{4!} + \cdots \right) - \left(\frac{1}{1!} + \frac{1}{3!} + \frac{1}{5!} + \cdots \right)$$

$$= \frac{1}{1!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \frac{1}{5!} + \cdots = \frac{1}{e}$$

ちなみに

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

について $x = -1$ を代入すると

$$\frac{1}{e} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{x^3}{3!} + \frac{1}{4!} - \frac{1}{5!} + \cdots$$

となります。

ノック 75 本目

$$a_n = \frac{1^2}{1!} + \frac{2^2}{2!} + \frac{3^2}{3!} + \cdots + \frac{n^2}{n!} = 1 + \frac{2}{1!} + \frac{3}{2!} + \cdots + \frac{n}{(n-1)!}$$

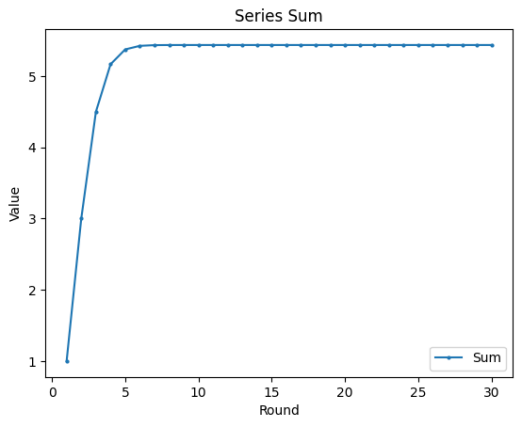
について a_n も $n \to \infty$ のとき、$2e = 2 \cdot 2.718282 = 5.436564$ に近づく。

RK75.py

```
1  import math  # 数学関数のためのライブラリをインポート
2
3  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
```

```
4
5  ns = 30  # ステップ数の設定
6  a = [0] * ns  # 長さ ns のリストを 0 で初期化
7
8  sv = 0  # 初期値の設定
9
10 # 各ステップでの計算
11 for i in range(1, ns + 1):
12     sv += i / math.factorial(i - 1)  # 累積和 sv の計算
13     a[i - 1] = sv  # a[i - 1] に sv を保存
14
15 # プロットの作成
16 plt.plot(range(1, ns + 1), a, marker='o', markersize=2,
   label='Sum')
17 plt.xlabel('Round')  # x 軸ラベル
18 plt.ylabel('Value')  # y 軸ラベル
19 plt.title('Series Sum')  # タイトルの設定
20 plt.legend()  # 凡例の表示
21 plt.show()  # プロットの表示
22
23 # 結果の出力
24 print(a[ns - 1])  # a の最後の要素を出力
25 print(2 * math.exp(1))  # 2*e を出力
```

```
> print(a[ns - 1])
5.43656365691809
> print(2 * math.exp(1))
5.43656365691809
```



たねあかし

$$\sum_{i=1}^{\infty} a_i = \frac{1^2}{1!} + \frac{2^2}{2!} + \frac{3^2}{3!} + \frac{4^2}{4!} + \cdots$$

$$= 1 + \frac{2}{1!} + \frac{3}{2!} + \frac{4}{3!} + \frac{5}{4!} + \cdots$$
$$= 1 + \sum_{i=1}^{\infty} \left[\frac{n+1}{n!}\right]$$
$$= 1 + \sum_{i=1}^{\infty} \left[\frac{1}{(n-1)!} + \frac{1}{1!}\right]$$
$$= 1 + e + (e - 1)$$
$$= 2e$$

ノック 76 本目

$$a_n = \frac{1^2}{1!} + \frac{2^2}{2!} + \frac{3^2}{3!} + \cdots + \frac{n^2}{n!} = 1 + \frac{2}{1!} + \frac{3}{2!} + \cdots + \frac{n}{(n-1)!}$$
$$b_n = \frac{1^2}{2!} + \frac{2^2}{3!} + \frac{3^2}{4!} + \cdots + \frac{n^2}{(n+1)!}$$

を比べてみよう。a_nは$n \to \infty$のとき、$2e = 2 \cdot 2.718282 = 5.436564$ に近づく。

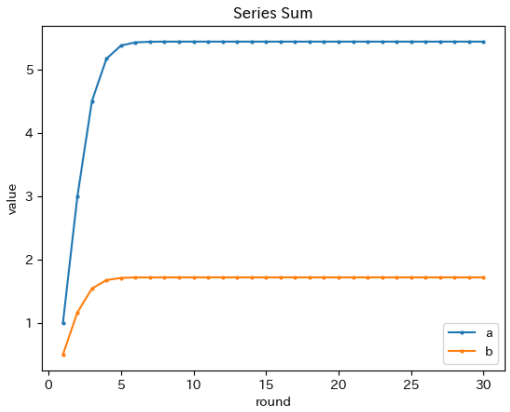
しかし、b_n は、$e - 1 = 2.718282 - 1 = 1.718282$ に近づく。面白っ！

RK76.py

```
1  import math  # 数学関数のためのライブラリをインポート
2
3  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
4
5  ns = 30  # ステップ数の設定
6  a = [0] * ns  # 長さ ns のリストを 0 で初期化
7  b = [0] * ns  # 長さ ns のリストを 0 で初期化
8  sv1 = 0  # 初期値の設定
9  sv2 = 0  # 初期値の設定
10
11 # 各ステップでの計算
12 for i in range(1, ns + 1):
13     sv1 += (i ** 2) / math.factorial(i)  # 累積和 sv1 の計算
14     a[i - 1] = sv1  # a[i - 1] に sv1 を保存
15
16     sv2 += (i ** 2) / math.factorial(i + 1)  # 累積和 sv2 の計算
17     b[i - 1] = sv2  # b[i - 1] に sv2 を保存
18
19 # プロットの作成
20 plt.plot(range(1, ns + 1), a, marker='o', markersize=2,
   label='a')
```

```
21  plt.plot(range(1, ns + 1), b, marker='o', markersize=2,
    label='b')
22  plt.xlabel('Round')  # x 軸ラベル
23  plt.ylabel('Value')  # y 軸ラベル
24  plt.title('Series Sum')  # タイトルの設定
25  plt.legend()  # 凡例の表示
26  plt.show()  # プロットの表示
27
28  # 結果の出力
29  print(a[ns - 1])  # a の最後の要素を出力
30  print(b[ns - 1])  # b の最後の要素を出力
31  print(2 * math.exp(1))  # 2*e を出力
32  print(math.exp(1) - 1)  # e-1 を出力
```

```
> print(a[ns - 1])
5.43656365691809
> print(b[ns - 1])
1.718281828459045
> print(2 * math.exp(1))
5.43656365691809
> print(math.exp(1) - 1)
1.718281828459045
```



ノック 77 本目

$$a_n = \frac{k}{(k+1)^1} + \frac{k}{(k+1)^2} + \frac{k}{(k+1)^3} + \cdots + \frac{k}{(k+1)^n}$$

について、$k = 1, 2, 3, 4, \cdots$ とどんな正の整数をいれても、1 に収束しまっせ！面白っ！

RK77.py

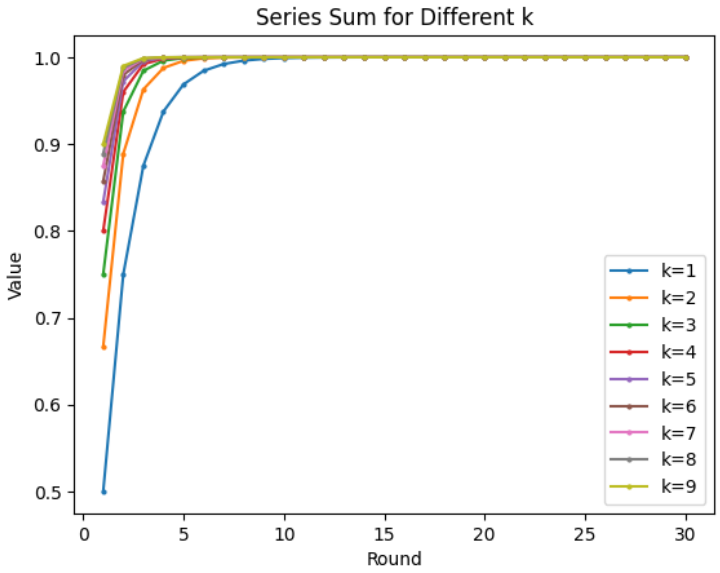
```
1  import matplotlib.pyplot as plt  # プロット作成のためのライブラリをインポート
2  import numpy as np  # 数値計算のためのライブラリをインポート
3
4  ns = 30  # ステップ数の設定
5  kmax = 9  # 最大 k の設定
6
7  # 長さ kmax, 幅 ns の行列を 0 で初期化
```

```
amat = np.zeros((kmax, ns))

# 各 k に対する計算
for k in range(1, kmax + 1):
    a = np.zeros(ns)  # 長さ ns のリストを 0 で初期化
    sv = 0  # 初期値の設定

    # 各ステップでの計算
    for i in range(1, ns + 1):
        sv += k / ((k + 1) ** i)  # 累積和 sv の計算
        a[i - 1] = sv  # a[i - 1] に sv を保存

    amat[k - 1] = a  # amat の k-1 列目に a を保存

# プロットの作成
for k in range(kmax):
    plt.plot(range(1, ns + 1), amat[k], marker='o',
markersize=2, label=f'k={k+1}')

plt.xlabel('Round')  # x 軸ラベル
plt.ylabel('Value')  # y 軸ラベル
plt.title('Series Sum for Different k')  # タイトルの設定
plt.legend()  # 凡例の表示
plt.show()  # プロットの表示

# 結果の出力
print(amat[:, ns - 1])  # amat の最後の行を出力
```

amat 行列として、ns (i＝1, 2, … , ns ; ノック 77 本目の n と体操) 行、kmat (k=1, 2, … , kmax; ノック 77 本目の k と体操) 列を定義する。それぞれの k について 1 に収束すること視覚化した。

```
> print(amat[ns - 1, :kmax])
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1.]
```


Series Sum for Different k
Value
Round
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30
k=1
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9

おわりに

プログラミングを楽しんでもらえたでしょうか？高校の数学の課題をもとにちょっと発展させた課題を解くことはいい機会だと思います。いろいろな課題を解きながら、いろんな分野を楽しむことができれば、人生楽しくなるでしょう。でも、プログラミングにはまるとあっという間に時間が過ぎます。適に時間を考えて遊ぶのがいいでしょう。例えば、試験前にはやっちゃだめです。こっちのほうが面白いからついついはまってしまいます。でも気分転換にはもってこいの分野です。

アルバート・アインシュタイン
は言った。

数学の難問に遭遇したとしても
けっして案ずるには及ばない
私達の知の容量は
それよりおおきいからである。

ともあれ、お疲れさまでした。

付録 Google Colaboratory 使い方

Python をデータ解析に活用するには、どうすればいいのだろうか？ 個人の PC で Python が使えるように環境構築することもできますが、パッケージのインストールエラーなどにてこずるなどの障害があるので、本書ではブラウザで Python を実行できる Google Colaboratory を用います。

Google アカウントの取得

まずは、以下のサイトから Google アカウントを作成してください。もうお持ちの方は飛ばしてください。

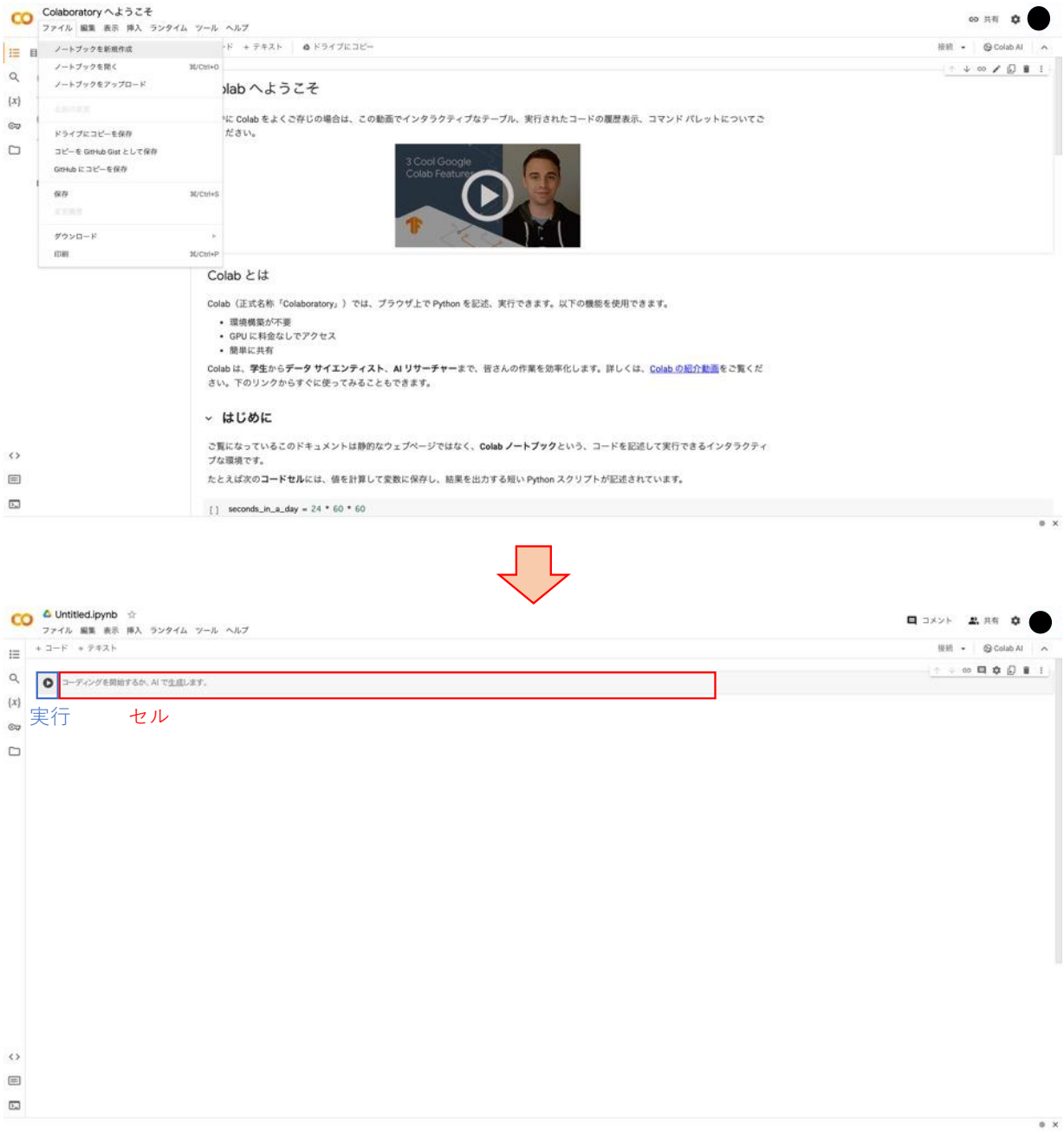
https://support.google.com/accounts/answer/27441?hl=ja

Google Colaboratory へのアクセス

まずは、以下の URL にアクセスしましょう。

https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja

そうすると、以下の画面が表示されるはずです。



Google Colaboratory でプログラムをつくる

上図に従って、「ファイル」から「ノートブックを新規作成」を選びます。そうすると何も書いてない画面が表示されます。プログラミングの準備が終わりました。赤で

示しているセルという部分にコードを書き、青で示している ▶ ボタンでそのセルを実行できます。

では実際にプログラムを作成してみましょう。まずセルに
「1+2+3+4+5+6+7+8+9+10」
と入力します。この 1 行を選択し、上部の Run ボタンをクリックさせると、プログラムが実行されます（以下の図参照）その結果、セルの下に
1+2+3+4+5+6+7+8+9+10 を実行した結果である 55 が表示されています。

二次方程式 $ax^2 + bx + c = 0$ の解は、$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ です。これをもとに、二次方程式を解くプログラムを作成してみましょう。ここで、二次方程式について、a, b, c が分っているときの x を求めるプログラムは以下のようになります。

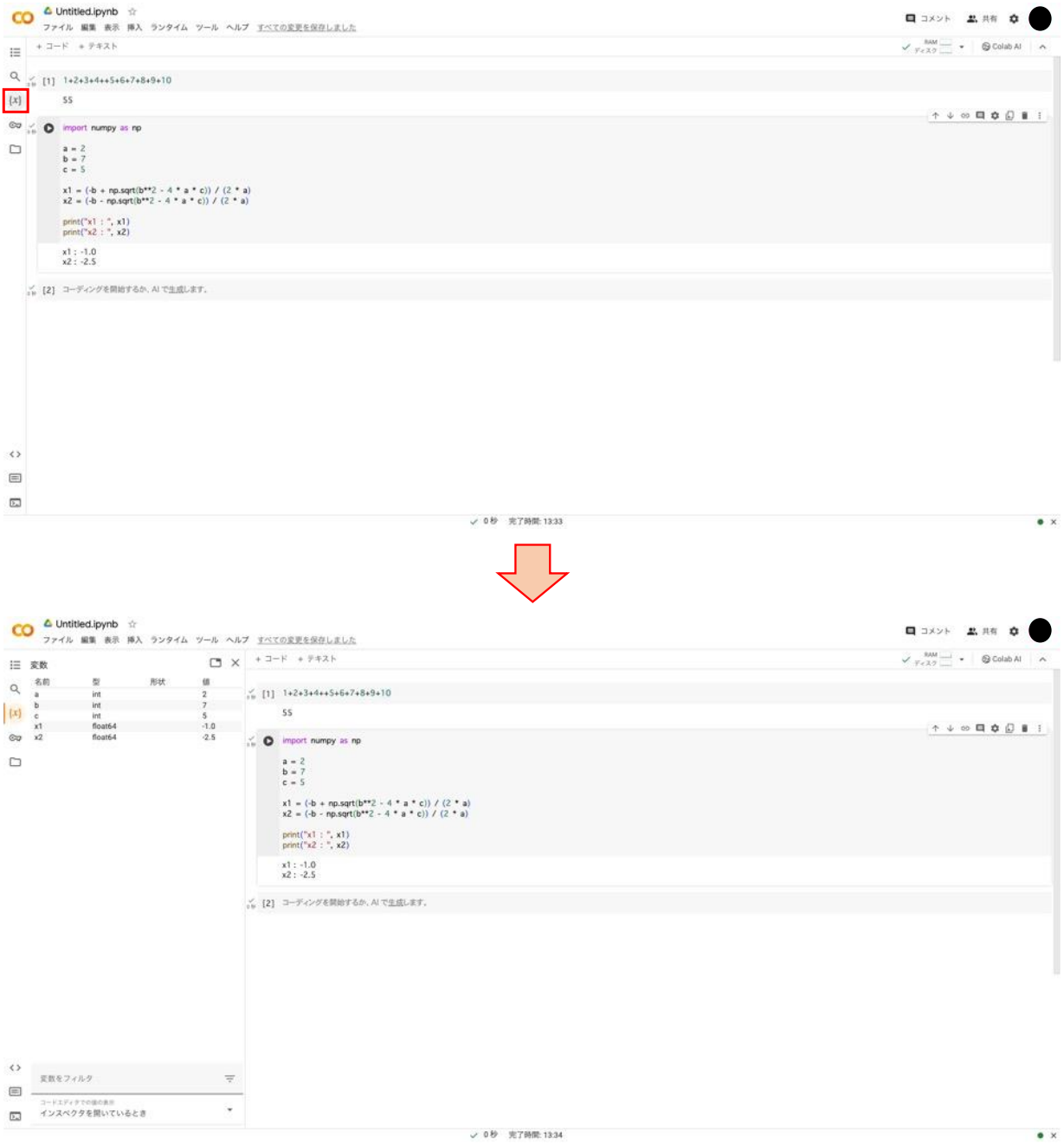
```
1  import numpy as np
2
3  a = 2
4  b = 7
5  c = 5
6
7  x1 = (-b + np.sqrt(b**2 - 4 * a * c)) / (2 * a)
8  x2 = (-b - np.sqrt(b**2 - 4 * a * c)) / (2 * a)
9
10 print("x1 : ", x1)
11 print("x2 : ", x2)
```

《 プログラムの概要 》
1 行目：numpy をインポートし、今後用いるときは np と書くことを宣言します。
3 行目：a = 2 とは、a に 2 を代入するという操作です。
3-5 行目：a=2, b=7, c=5 が代入されました。

7 行目：$x1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$

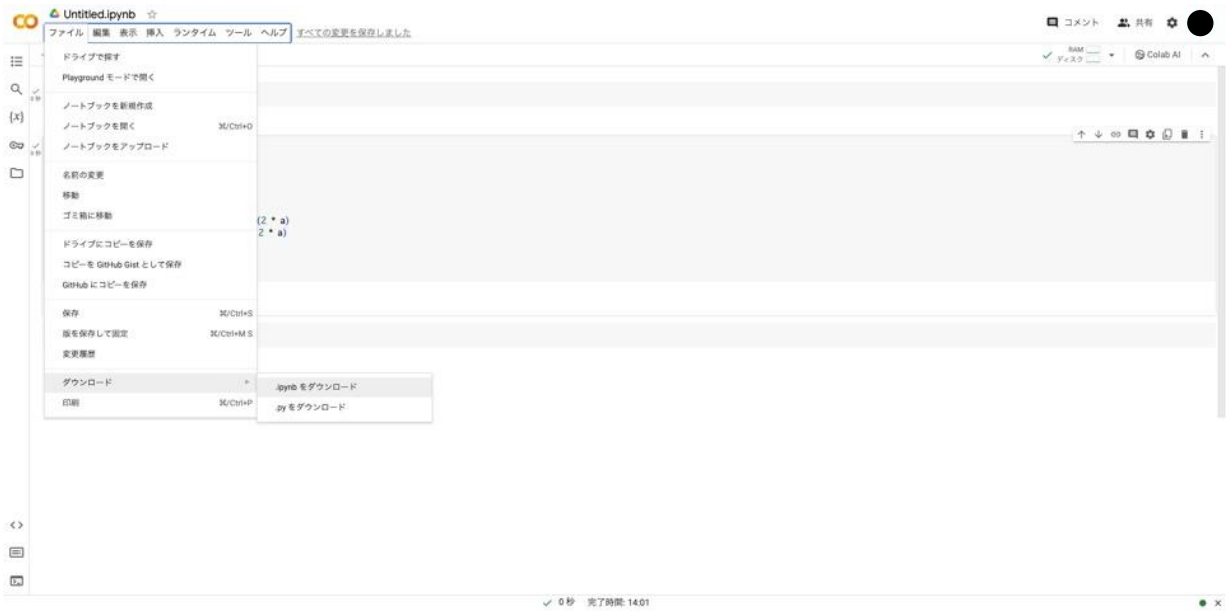
8 行目：$x2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$

を Python により作成しました。ここで np.sqrt()は√の計算を行う関数です。10,11 行目：x1 と x2 に格納された値（二次方程式の解）が出力されます。以下に実行結果を示しました。x1 と x2 を出力すると-1 と-2.5 となりました。$\{x\}$ を押すと各変数とその型と値を確認することもできます。



Google Colaboratory で作成したプログラムを保存する

ではこのプログラムをファイルにセーブしてみましょう。下図のようにメニューバーの「ファイル」から「ダウンロード」を選択し、「.ipynb をダウンロード」を選びダウンロードします。ダウンロードしたコードは個人の PC に環境を作れば、動かせますし、もう一度 Google Colaboratory にアップロードして動かすこともできます。



Google Colaboratory にプログラムをアップロードする

下図のようにメニューバーの「ファイル」から「ノートブックをアップロード」を選択し、アップロードしたい ipynb ファイルをドラック&ドロップします。そうすることで、これまでに作ってきたプログラムを Google Colaboratory 上で動かせるようになります。

著者（執筆時所属）

金谷　重彦　奈良先端科学技術大学院大学・データ駆動型サイエンス創造センター
平尾　俊貴　奈良先端科学技術大学院大学・先端科学技術研究科・情報科学領域
小笠原　司　奈良先端科学技術大学院大学・地域共創推進室
松本　健一　奈良先端科学技術大学院大学・先端科学技術研究科・情報科学領域
嶋利　一真　奈良先端科学技術大学院大学・先端科学技術研究科・情報科学領域
工藤　拓斗　奈良先端科学技術大学院大学・先端科学技術研究科・情報科学領域
田中　英武　奈良先端科学技術大学院大学・先端科学技術研究科・情報科学領域
山﨑　和真　奈良先端科学技術大学院大学・先端科学技術研究科・情報科学領域
張　凡　奈良先端科学技術大学院大学・先端科学技術研究科・情報科学領域

高校生のデータサイエンス・Python でも 77 本ノック
NAIST STELLA プログラム “「共創」が育む主体性の未来” 学習教材

2024 年 10 月 1 日　初版発行
編著者 金谷重彦
著作　奈良先端科学技術大学院大学
NAIST STELLA プログラム運営委員会
発行所 国立大学法人 奈良先端科学技術大学院大学
〒630-0192 奈良県生駒市高山町 8916-5
https://www.naist.jp/
電話 0743-72-5111（代表）

ISBN 978-4-902874-06-8