Numpy および Python

プログラミング言語

C言語

- ✦**1972**年に開発された汎用高級言語
- ✦**機械語に近い構成による高速な実行速度**
- ✦**ポインタのポインタ**に代表される難解な仕様
  - ✦**習得難易度が高い**

Python

- ✦**1991**年に開発された汎用高級言語
- ✦豊富なライブラリ(モジュール)による拡張
- ✦**簡易な文法により記述することができる**
  - ✦習得難易度が低い
- ✦速度がC言語と比較して**劣る**

プログラミング言語

- C言語
- Python

✦ 科学計算用に設計された言語ではない

✦ 自分で多くを定義する必要がある

e.g. 構造体，クラス，関数

✦ 本来の趣旨から離れた作業が必要

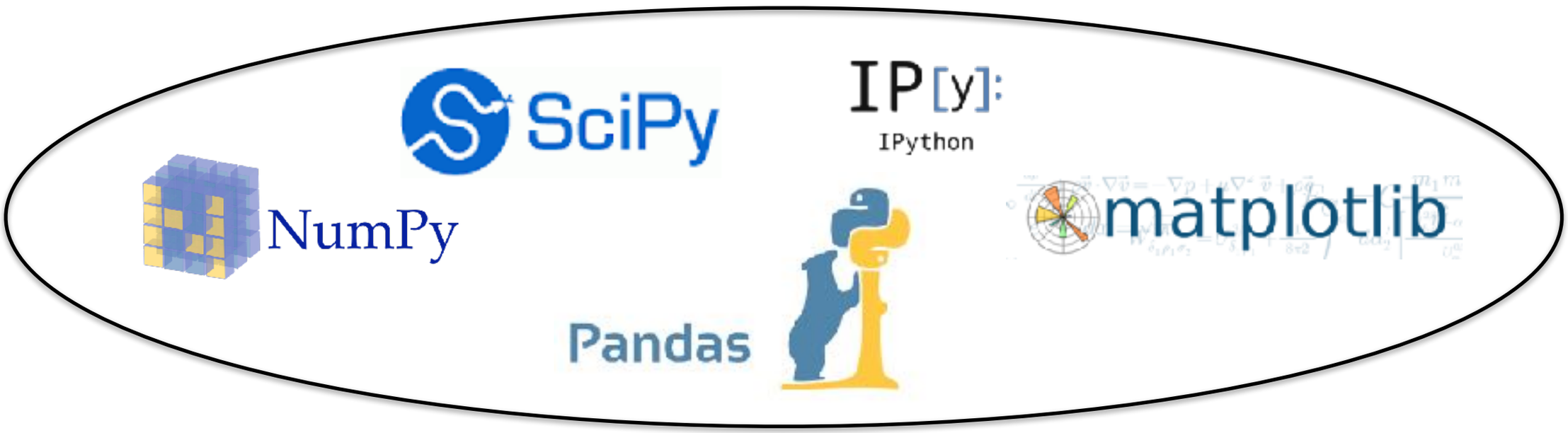
Numpyを使おう！

✦ 科学計算用に設計されたPythonのライブラリ

✦ 行列演算など算術計算が高速

Numpy

- Pythonで用いられる科学計算用ライブラリ
  - ✦内部はCで実装されている
  - ✦その他の数学・解析系ライブラリにおける基盤
  - ✦JuliaやRubyなど多言語での利用も可能



本編の前に…

- Pythonのヴァージョン

```
Python 3.6.4
```

- 表記

  - コード名

```
#include <stdio.h>

int main(void){
  return 0;
}
```

- リンク

https://github.com/Scstechr/D-WiSE_LT/tree/master/180301/codes

Hello World

C言語

❖ hello.c

```
#include <stdio.h>

int main(void){
  printf(“Hello World\n”);

  return 0;
}
```

✤ コンパイル

```
$ gcc hello.c
```

✤ 実行

```
$ ./a.out
Hello World
```

Hello World

Python

❖ hello.py

```
print(“Hello World”)
```

❖ 実行

```
$ python hello.py
Hello World
```

逐次コンパイルを行うため事前コンパイルは不要

❖ hello_main.py

```
def main():
    print("Hello World")

if __name__ == "__main__":
    main()
```

Hello World

比較

❖ hello.c

```
#include <stdio.h>

int main(void){
  printf("Hello World\n");

  return 0;
}
```

❖ hello.py

```
print(“Hello World”)
```

同じ処理を少ない行数で記述することが可能

デバッグの負担を軽減する

Hello World

比較

❖ hello.c

```
#include <stdio.h>

int main(void){
  printf("Hello World\n");

  return 0;
}
```

❖ hello_main.py

```
def main():
    print("Hello World")

if __name__ == "__main__":
    main()
```

配列の初期化

C言語

❖ array.c

```
int main(void){

  int array[10];

  for(int i = 0; i < 10; i++){
    array[i] = 0;
  }

  return 0;
}
```

```
array[] = {0,0,0,0,0,0,0,0,0,0}
```

配列の初期化

Python

❖ array.py

```
def main():
    lst = []

    for i in range(10):
        lst.append(0)

if __name__ == "__main__":
    main()
```

リスト (list)

- ✤ 多機能なPythonの配列
- ✤ lst.append() は()の中身を末尾に追加

lst = [0 0 0 0 0 0 0 0 0 0]

配列の初期化

比較

❖ array.c

```
int main(void){

  int array[10];

  for(int i = 0; i < 10; i++){
    array[i] = 0;
  }

  return 0;
}
```

❖ array.py

```
def main():

    lst = []

    for i in range(10):
      lst.append(0)

if __name__ == "__main__":
    main()
```

配列の初期化

比較

❖ array.c

```c
int main(void){

  int array[10];

  for(int i = 0; i <
  10; i++){
    array[i] = 0;
```

❖ array.py

```python
def main():

   lst = []

   for i in
    range(10):
      lst.append(0)
```

変数の型付け

- ✤ C言語： 静的型付け （変数宣言時に型宣言を行う）
- ✤ Python： 動的型付け （右辺により型変化）

配列の初期化

リスト内包表記

❖ array_comp.py

```
def main():

    lst = [0 for i in range(10)]

if __name__ == "__main__":
    main()
```

リストの初期化は「リスト内包表記」で一文で書ける

配列の初期化

コンストラクタ

❖ array_const.py

```
def main():

    lst = list(range(10))

if __name__ == "__main__":
    main()
```

for文を使わない分リスト内包表記より(おそらく)速い

配列を出力する

C言語

❖ array_print.c

```
#include <stdio.h>

int main(void){

   int array[10];

   for(int i = 0; i < 10; i++){
      array[i] = i;
   }

   for(int i = 0; i < 10; i++){
      printf(“%d\n”, array[i]);
   }

   return 0;
}
```

```
$ gcc array_print.c
$ ./a.out
0
1
2
3
4
5
6
7
8
9
```

配列を出力する

Python

❖ array_print.py

```
def main():
  lst = [i for i in range(10)]

  for i in lst:
    print(i)

  # [print(i) for i in lst]

if __name__ == '__main__':
  main()
```

```
$ python array_print2.py
0
1
2
3
4
5
6
7
8
9
```

Pythonにおける出力

print()

❖ array_print2.py

```
def main():
  lst = [i for i in range(10)]

  print(lst)

if __name__ == '__main__':
  main()
```

```
$ python array_print2.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

❖ hello_main.py

```
def main():
  print('Hello World')

if __name__ == '__main__':
  main()
```

```
$ python hello_main.py
Hello World
```

print()

✤ ()の中身に合わせて出力を変化

配列を出力する

C言語で同様の動作を実装する (関数篇)

❖ array_print2.c (1/2)

```
#include <stdio.h>

void print(int *array, int size){
  printf("[");
  for(int i = 0; i < size; i++){
    printf("%d,", array[i]);
  }
  printf("\b]\n");
}
```

配列のポインタとサイズを受け取る関数を定義

配列を出力する

C言語で同様の動作を実装する (関数篇)

❖ array_print2.c (2/2)

```
int main(void){

  int array[10];

  for(int i = 0; i < 10; i++){
    array[i] = i;
  }

  print(array, 10);

  return 0;
}
```

配列を出力する

C言語で同様の動作を実装する (構造体篇)

❖ array_print3.c (1/3)

```
#include <stdio.h>
#include <stdlib.h>

#define ARRAY_SIZE 10

typedef struct{
   int array[ARRAY_SIZE];
   int size;
} Array;
```

配列とサイズで構成される構造体を定義

配列を出力する

C言語で同様の動作を実装する (構造体篇)

❖ array_print3.c (2/3)

```
void print(Array *array){
  printf("[");
  for(int i = 0; i < array->size; i++){
    printf("%d,", array->array[i]);
  }
  printf("\b]\n");
}
```

構造体のポインタを受け取る関数を定義

配列を出力する

C言語で同様の動作を実装する (構造体篇)

❖ array_print3.c (3/3)

```
int main(void){

  Array *array = (Array*)malloc(sizeof(Array));
  array->size = 10;

  for(int i = 0; i < array->size; i++){
    array->array[i] = i;
  }

  print(array);
  free(array);
  return 0;
}
```

配列を出力する

int型の(一次元の)配列にしか有効ではない

❖ array_print2.c

```
#include <stdio.h>

void print(int *array, int size){
    printf("[");
    for(int i = 0; i < size; i++){
        printf("%d,", array[i]);
    }
    printf("\b]\n");
}

int main(void){

    int array[10];

    for(int i = 0; i < 10; i++){
        array[i] = i;
    }

    print(array, 10);

    return 0;
}
```

❖ array_print3.c

```
#include <stdio.h>
#include <stdlib.h>

#define ARRAY_SIZE 10

typedef struct{
     int array[ARRAY_SIZE];
     int size;
} Array;

void print(Array *array){
     printf("[");
     for(int i = 0; i < array->size; i++){
          printf("%d,", array->array[i]);
     }
     printf("\b]\n");
}

int main(void){

     Array *array = (Array*)malloc(sizeof(Array));
     array->size = 10;

     for(int i = 0; i < array->size; i++){
          array->array[i] = i;
     }

     print(array);
     free(array);
     return 0;
}
```

配列を出力する

Pythonのprint()の凄さ

❖ array_print2.c

```
#include <stdio.h>

void print(int *array, int size){
    printf("[");
    for(int i = 0; i < size; i++){
        printf("%d,", array[i]);
    }
    printf("\b]\n");
}

int main(void){

    int array[10];

    for(int i = 0; i < 10; i++){
        array[i] = i;
    }

    print(array, 10);

    return 0;
}
```

❖ array_print2.py

```
def main():
    lst = [i for i in range(10)]

    print(lst)

if __name__ == '__main__':
    main()
```

Pythonのデバッグのお供

Pythonのlistの凄さ

- 対話型インタープリタの起動

```
$ python
Python 3.6.2 |Anaconda custom (64-bit)| (default,
Jul 20 2017, 13:14:59)
[GCC 4.2.1 Compatible Apple LLVM 6.0
(clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license"
for more information.
>>> exit()
$
```

簡単な動作確認をすることができる

Pythonのデバッグのお供

Pythonのlistの凄さ

- 充実した既存の関数群

```
>>> dir(list)
['__add__', '__class__', '__contains__',
'__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__',
'__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'clear',
'copy', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
```

Pythonのデバッグのお供

その他の有用な機能

- help(): 関数その他の詳細をviで出力 (qで抜け出す)

```
>>> help(list.append)
Help on method_descriptor:

append(...)
    L.append(object) -> None -- append object to end
```

- type(): インスタンスの種類を出力

```
>>> lst = list(range(10))
>>> lst
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> type(lst)
<class 'list'>
```

実行時間

比較

❖ array_print3.c

```
$ time ./a.out
[0,1,2,3,4,5,6,7,8,9]

real 0m0.007s
user 0m0.002s
sys  0m0.002s
```

❖ array_print2.py

```
$ time python array_print2.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

real 0m0.062s
user 0m0.039s
sys  0m0.015s
```

Pythonは遅い!

Numpy

PythonのNumpyの凄さ

- 対話型インタープリタの起動

```
$ python
Python 3.6.2 |Anaconda custom (64-bit)| (default,
Jul 20 2017, 13:14:59)
[GCC 4.2.1 Compatible Apple LLVM 6.0
(clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license"
for more information.
>>>
```

Numpy

Pythonのライブラリ利用法

✤ importする

```
>>> import numpy as np
```

numpyを書くのは長いのでnpと表記する（慣例）

★ C言語の類似表記:

```
#include <stdio.h>
#include “myheader.h”
```

★ Pythonの場合は出自を記す必要がある

```
import numpy as np
#numpyで定義されたsum()を使いたい

np.sum()
```

Numpy

PythonのNumpyの凄さ

- 豊富な関数群

```
>>> dir(np)
['ALLOW_THREADS', 'BUFSIZE', 'CLIP', 'ComplexWarning', 'DataSource',
'ERR_CALL', 'ERR_DEFAULT', 'ERR_IGNORE', 'ERR_LOG', 'ERR_PRINT',
'ERR_RAISE', 'ERR_WARN', 'FLOATING_POINT_SUPPORT', 'FPE_DIVIDEBYZERO',
'FPE_INVALID', 'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf',
'Infinity', 'MAXDIMS', 'MAY_SHARE_BOUNDS', 'MAY_SHARE_EXACT', 'MachAr',
'ModuleDeprecationWarning', 'NAN', 'NINF', 'NZERO', 'NaN', 'PINF',
'PZERO', 'PackageLoader', 'RAISE', 'RankWarning', 'SHIFT_DIVIDEBYZERO',
'SHIFT_INVALID', 'SHIFT_OVERFLOW', 'SHIFT_UNDERFLOW', 'ScalarType',
'Tester', 'TooHardError', 'True_', 'UFUNC_BUFSIZE_DEFAULT',
'UFUNC_PYVALS_NAME', 'VisibleDeprecationWarning', 'WRAP', '_NoValue',
'__NUMPY_SETUP__', '__all__', '__builtins__', '__cached__',
'__config__', '__doc__', '__file__', '__git_revision__', '__loader__',
'__name__', '__package__', '__path__', '__spec__', '__version__',
'_import_tools', '_mat', 'abs', 'absolute', 'absolute_import', 'add',
'add_docstring', 'add_newdoc', 'add_newdoc_ufunc', 'add_newdocs',
'alen', 'all', 'allclose', 'alltrue', 'alterdot', 'amax', 'amin',
'angle', 'any', 'append', 'apply_along_axis', 'apply_over_axes',
'arange', 'arccos', 'arccosh', 'arcsin', 'arcsinh', 'arctan', 'arctan2',
'arctanh', 'argmax', 'argmin', 'argpartition', 'argsort', 'argwhere',
```

Numpy

PythonのNumpyの凄さ

- 多次元行列(np.ndarray)に限っても豊富

```
>>> dir(np.ndarray)
['T', '__abs__', '__add__', '__and__', '__array__',
'__array_finalize__', '__array_interface__', '__array_prepare__',
'__array_priority__', '__array_struct__', '__array_wrap__', '__bool__',
'__class__', '__contains__', '__copy__', '__deepcopy__', '__delattr__',
'__delitem__', '__dir__', '__divmod__', '__doc__', '__eq__',
'__float__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__iand__',
'__ifloordiv__', '__ilshift__', '__imatmul__', '__imod__', '__imul__',
'__index__', '__init__', '__init_subclass__', '__int__', '__invert__',
'__ior__', '__ipow__', '__irshift__', '__isub__', '__iter__',
'__itruediv__', '__ixor__', '__le__', '__len__', '__lshift__', '__lt__',
'__matmul__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__',
'__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__',
'__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__',
'__rlshift__', '__rmatmul__', '__rmod__', '__rmul__', '__ror__',
'__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__',
'__rxor__', '__setattr__', '__setitem__', '__setstate__', '__sizeof__',
'__str__', '__sub__', '__subclasshook__', '__truediv__', '__xor__',
'all', 'any', 'argmax', 'argmin', 'argpartition', 'argsort', 'astype',
```

Numpy

listからnp.ndarrayへの変換

❖ np_array.py

```
import numpy as np

def main():
    lst = [i for i in range(10)]
    print(lst, type(lst))

    array = np.array(lst)
    print(array, type(array))

if __name__ == "__main__":
    main()
```

```
$ python np_array.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>
[0 1 2 3 4 5 6 7 8 9] <class ‘numpy.ndarray'>
```

Numpy

numpyで定義された関数を用いる

❖ np_array2.py

```
import numpy as np

def main():
    #array = np.array([i for i in range(10)])
    array = np.arange(10)
    print(array)

if __name__ == "__main__":
    main()
```

np.arange(n): 0~n-1で初期化された一次元配列

実行時間の計測: Python

比較

❖ array_print2.py

```
$ time python array_print2.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

real  0m0.062s
user  0m0.039s
sys   0m0.015s
```

❖ np_array2.py

```
$ time python np_array2.py
[0 1 2 3 4 5 6 7 8 9]

real  0m0.239s
user  0m0.161s
sys   0m0.062s
```

実行時間の計測: Python

numpyをimportするオーバーヘッド

❖ import_numpy.py

```
$ cat import_numpy.py
import numpy as np
$ time python import_numpy.py

real 0m0.226s
user 0m0.149s
sys  0m0.062s
```

❖ array_print2.py

```
real 0m0.062s
user 0m0.039s
sys  0m0.015s
```

❖ array_print2.py

```
real 0m0.239s
user 0m0.161s
sys  0m0.062s
```

→

```
real 0m0.013s
user 0m0.012s
sys  0m0.000s
```

実行時間の計測: Python

- ipythonを利用する
  - 多機能の対話型インタープリタの起動

```
$ ipython
Python 3.6.2 |Anaconda custom (64-bit)| (default,
Jul 20 2017, 13:14:59)
Type 'copyright', 'credits' or 'license' for more
information
IPython 6.2.1 -- An enhanced Interactive Python.
Type '?' for help.

In [1]: list.append?
Docstring: L.append(object) -> None -- append
object to end
Type:      method_descriptor

In [2]:
```

実行時間の計測: Python

- ipythonを利用する

  - Numpyをimport

```
In [1]: import numpy as np
```

  - nを引数とする関数を定義

```
In [2]: def range_list(n):
   ...:     lst = list(range(n))
   ...:
In [3]: def arange_np(n):
   ...:     array = np.arange(n)
   ...:
```

実行時間の計測: Python

- ipythonを利用する
  - %timeitで計測 (要素数10の1次元配列)

```
In [4]: %timeit range_list(10)
878 ns ± 31.8 ns per loop (mean ± std.
dev. of 7 runs, 1000000 loops each)

In [5]: %timeit arange_np(10)
872 ns ± 32.8 ns per loop (mean ± std.
dev. of 7 runs, 1000000 loops each)
```

実行時間の計測: Python

- ipythonを利用する
  - %timeitで計測 (要素数10000の1次元配列)

```
In [6]: %timeit range_list(10000)
238 µs ± 9.11 µs per loop (mean ± std.
dev. of 7 runs, 1000 loops each)

In [7]: %timeit arange_np(10000)
5.87 µs ± 179 ns per loop (mean ± std.
dev. of 7 runs, 100000 loops each)
```

実行時間: C

time.hを利用する

❖ array_10k.c (1/3)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ARRAY_SIZE 10000
#define TRIAL 100000

typedef struct{
   int data[ARRAY_SIZE];
   int size;
} Array;
```

ヘッダのincludeと定数・構造体の定義

実行時間: C

time.hを利用する

❖ array_10k.c (2/3)

```
void process(){
   Array *array = (Array*)malloc(sizeof(Array));
   array->size = 10000;

   for(int i = 0; i < array->size; i++){
      array->data[i] = i;
   }
}
```

配列の初期化を関数として切り出す

実行時間: C

time.hを利用する

❖ array_10k.c (3/3)

```
int main(){
   clock_t t1, t2;
   double sum; sum = 0;

   for(int i = 0; i < TRIAL; i++){
      t1 = clock();process();t2 = clock();

      sum += (double)(t2-t1)/CLOCKS_PER_SEC;
   }

   printf("%f\n", (double)sum / TRIAL);

   return 0;
}
```

実行時間

time.hを利用する

- array_10k.c

```
$ ./array_10k.out
0.000034
```

比較

種類		実行時間
Python	list	238 [μs]
	numpy	5.86 [μs]
C		34 [μs]

~~Pythonは遅い!~~

Numpy - 配列の初期化

- ipythonを利用する
  - ✤ 多機能型の対話型インタープリタの起動

```
$ ipython
Python 3.6.2 |Anaconda custom (64-bit)| (default,
Jul 20 2017, 13:14:59)
Type 'copyright', 'credits' or 'license' for more
information
IPython 6.2.1 -- An enhanced Interactive Python.
Type '?' for help.
```

  - ✤ Numpyをimport

```
In [1] : import numpy as np
```

Numpy - 配列の初期化

全零行列

- np.zeros()

```
In [2]: np.zeros(10)
Out[2]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,
0.,  0.,  0.])
```

全一行列

- np.ones()

```
In [3]: np.ones(10)
Out[3]: array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,
1.,  1.,  1.])
```

Numpy - 多次元行列を定義

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

- listを変換

```
In [4]: np.array([[0,1],[2,3]])
Out[4]:
array([[0, 1],
       [2, 3]])
```

- np.reshape()で1次元配列を多次元へ変換

```
In [5]: np.arange(4).reshape([2,2])
Out[5]:
array([[0, 1],
       [2, 3]])
```

Numpy - 多次元行列を定義

全零行列 np.zeros()

```
In [6]: np.zeros([2,2])
Out[6]:
array([[ 0.,  0.],
       [ 0.,  0.]])
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

全一行列 np.ones()

```
In [7]: np.ones([2,2])
Out[7]:
array([[ 1.,  1.],
       [ 1.,  1.]])
```

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

単位行列 np.identity()

```
In [8]: np.identity(2)
Out[8]:
array([[ 1.,  0.],
       [ 0.,  1.]])
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Numpy - 行列演算

$$a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

```
In [9]: a = np.array([1,2,3])
In [9]: b = np.array([4,5,6])
```

加算

```
In [10]: a + b
Out[10]: array([5, 7, 9])
```

減算

```
In [11]: a - b
Out[11]: array([-3, -3, -3])
```

Numpy - 行列演算

$$a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

```
In [9]: a = np.array([1,2,3])
In [9]: b = np.array([4,5,6])
```

乗算

```
In [12]: a * b
Out[12]: array([ 4, 10, 18])
```

除算

```
In [13]: a / b
Out[13]: array([ 0.25,  0.4 ,  0.5 ])
```

Numpy - 線形代数

np.linalg

```
>>> dir(np.linalg)
['LinAlgError', '__builtins__', '__cached__',
'__doc__', '__file__', '__loader__', '__name__',
'__package__', '__path__', '__spec__',
'_numpy_tester', '_umath_linalg',
'absolute_import', 'bench', 'cholesky', 'cond',
'det', 'division', 'eig', 'eigh', 'eigvals',
'eigvalsh', 'info', 'inv', 'lapack_lite', 'linalg',
'lstsq', 'matrix_power', 'matrix_rank',
'multi_dot', 'norm', 'pinv', 'print_function',
'qr', 'slogdet', 'solve', 'svd', 'tensorinv',
'tensorsolve', 'test']
>>>
```

Numpy - 線形代数

$$c = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
In [14]: c = np.arange(1,5).reshape(2,2)
```

- 行列式： $\det \mathbf{c}$

```
In [15]: np.linalg.det(c)
Out[15]: -2.0000000000000004
```

- ノルム： $\|\mathbf{c}\|$

```
In [16]: np.linalg.norm(c)
Out[16]: 5.477225575051661 2
```

Numpy - 線形代数

$$c = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
In [14]: c = np.arange(1,5).reshape(2,2)
```

- 逆行列： $\mathbf{c}^{-1}$

```
In [17]: np.linalg.inv(c)
Out[17]:
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Numpy - 乱数生成

random（既存ライブラリ）

```
>>> import random
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random',
'SG_MAGICCONST', 'SystemRandom', 'TWOPI',
'_BuiltinMethodType', '_MethodType', '_Sequence', '_Set',
'__all__', '__builtins__', '__cached__', '__doc__',
'__file__', '__loader__', '__name__', '__package__',
'__spec__', '_acos', '_bisect', '_ceil', '_cos', '_e',
'_exp', '_inst', '_itertools', '_log', '_pi', '_random',
'_sha512', '_sin', '_sqrt', '_test', '_test_generator',
'_urandom', '_warn', 'betavariate', 'choice', 'choices',
'expovariate', 'gammavariate', 'gauss', 'getrandbits',
'getstate', 'lognormvariate', 'normalvariate',
'paretovariate', 'randint', 'random', 'randrange', 'sample',
'seed', 'setstate', 'shuffle', 'triangular', 'uniform',
'vonmisesvariate', 'weibullvariate']
```

Numpy - 乱数生成

np.random

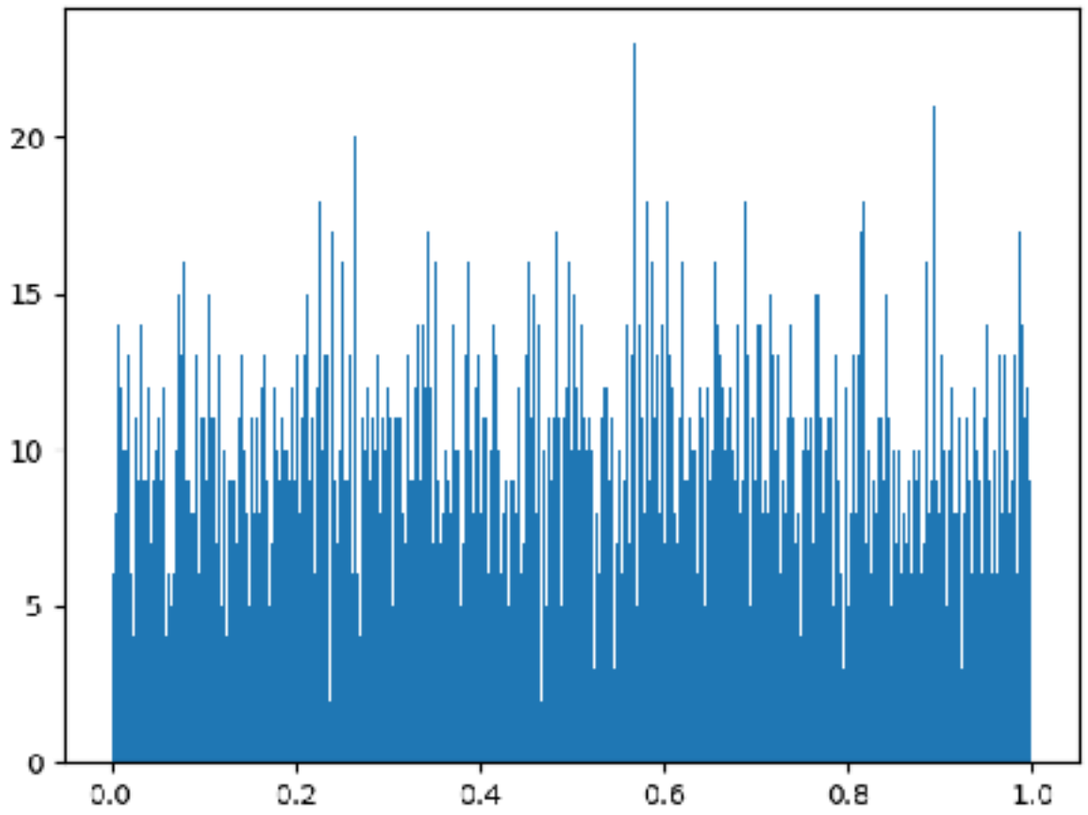
```
>>> dir(np.random)
['Lock', 'RandomState', '__RandomState_ctor', '__all__',
'__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__path__',
'__spec__', '_numpy_tester', 'absolute_import', 'bench',
'beta', 'binomial', 'bytes', 'chisquare', 'choice',
'dirichlet', 'division', 'exponential', 'f', 'gamma',
'geometric', 'get_state', 'gumbel', 'hypergeometric', 'info',
'laplace', 'logistic', 'lognormal', 'logseries', 'mtrand',
'multinomial', 'multivariate_normal', 'negative_binomial',
'noncentral_chisquare', 'noncentral_f', 'normal', 'np',
'operator', 'pareto', 'permutation', 'poisson', 'power',
'print_function', 'rand', 'randint', 'randn', 'random',
'random_integers', 'random_sample', 'ranf', 'rayleigh',
'sample', 'seed', 'set_state', 'shuffle', 'standard_cauchy',
'standard_exponential', 'standard_gamma', 'standard_normal',
'standard_t', 'test', 'triangular', 'uniform', 'vonmises',
'wald', 'warnings', 'weibull', 'zipf']
```

Numpy - 異なる分布による乱数生成

一様分布: np.random.rand()

❖ np_rand.py

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

def main():
    R = np.random.rand(10000)

    plt.hist(R, bins=1000)

    plt.show()

if __name__ == "__main__":
    main()
```



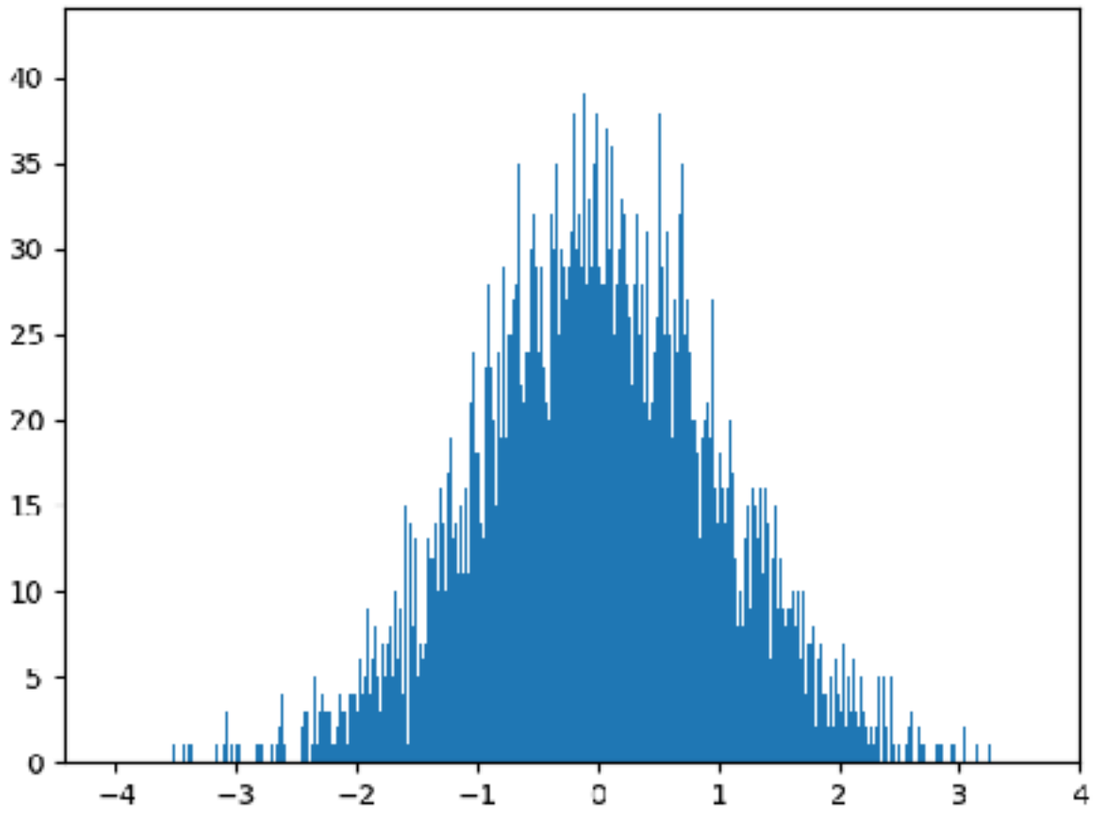
一様分布

Numpy - 異なる分布による乱数生成

標準正規分布 $\mathcal{N}(0,1)$: np.random.randn()

❖ np_randn.py

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

def main():
    R = np.random.randn(10000)

    plt.hist(R, bins=1000)

    plt.show()

if __name__ == "__main__":
    main()
```



標準正規分布

Numpy - 分布関数で決定

ランダムに選択: np.random.choice()

$$\Lambda_8(x) = 0.5x^2 + 0.28x^3 + 0.22x^8$$

❖ np_rand_choice.py

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

def main():
    array = np.arange(9)
    pmf = np.zeros(9)

    pmf[2] = 0.5
    pmf[3] = 0.28
    pmf[8] = 0.22

    R = np.random.choice(array, size = 10000, p=pmf)

    #print(R)
    plt.hist(R, bins=100)

    plt.show()

if __name__ == "__main__":
    main()
```

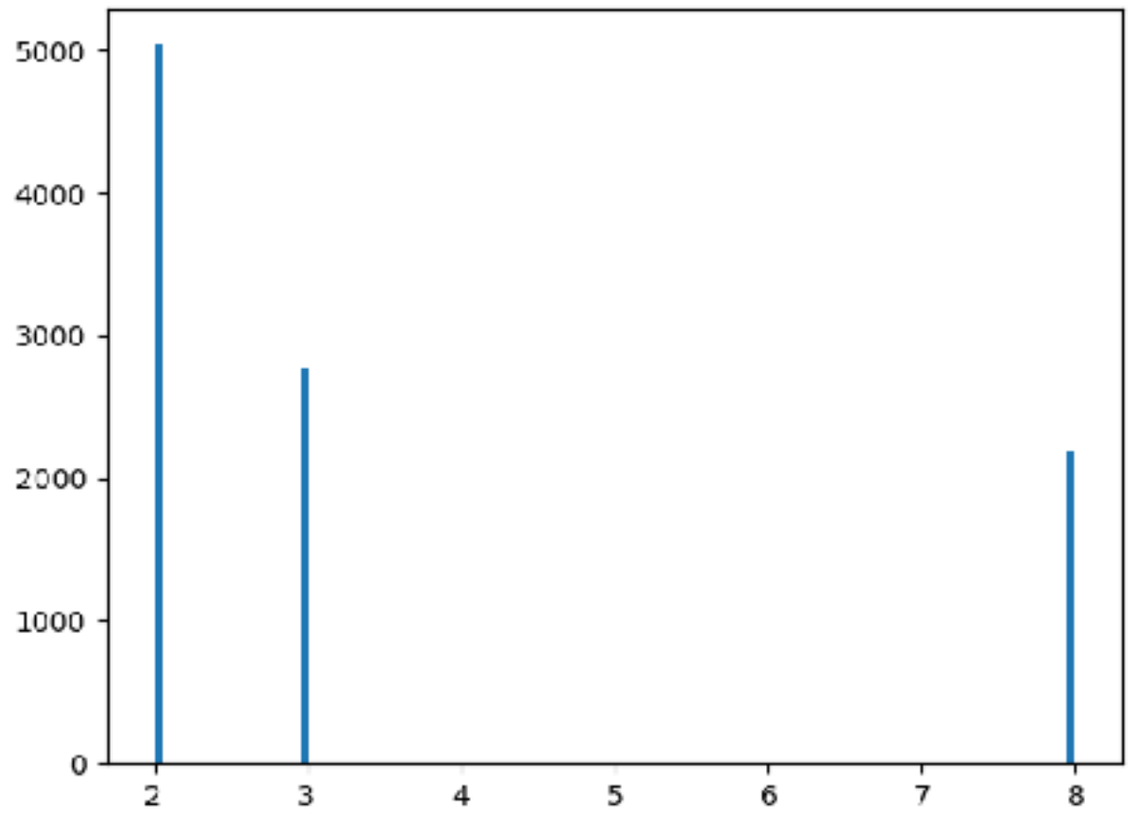
Numpy - 分布関数で決定

ランダムに選択: np.random.choice()

$$\Lambda_8(x) = 0.5x^2 + 0.28x^3 + 0.22x^8$$

❖ np_rand_choice.py

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

def main():
    array = np.arange(9)
    pmf = np.zeros(9)

    pmf[2] = 0.5
    pmf[3] = 0.28
    pmf[8] = 0.22

    R = np.random.choice(array, size =

    #print(R)
    plt.hist(R, bins=100)

    plt.show()

if __name__ == "__main__":
    main()
```



まとめ

Python

- 簡潔に記述できるためデバッグしやすい
- C言語と比較すると実行速度が遅い
- そのままでは科学計算用には適していない

Numpy

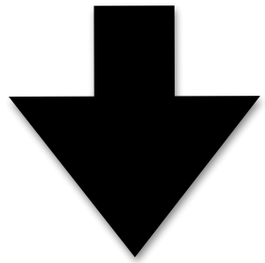
- Pythonの科学計算用ライブラリ
- 内部がCで記述されており高速な演算が可能
- 豊富なライブラリ関数による簡易な実装

その他の数学用ライブラリ

- Scipy： 算術計算用のライブラリ
  - 微分積分や疎行列などを定義している
  - NumpyにないものはおそらくScipyにある
- Sympy： 変数をシンボルとして扱うためのライブラリ
  - x, yなどに具体的に値を代入する必要がない
  - 極限などよりテーラー展開など応用数学分野に強い
  - 式の展開とか整理に利用？

NumpyおよびPythonの使い方

処理の流れをPythonで決める



実行速度に満足したか？

はい

いいえ

Cで書き直す？

はい

いいえ

修羅の道へ…

おすすめ

- YouTubeなどでカンファレンス動画を観る
  - PyData, Enthought, Next Day Video, PyCon
- オライリー本を買う
- GitHubなどでコードを漁る
- 公式ドキュメントを熟読する
- 新しいメソッドを知るごとに処理速度が変化

補助

まとめ

プログラミング言語年代記

1950s	Fortran		
1960s	BASIC		
1970s	C	SQL	SmallTalk
1980s	C++	Common Lisp	Perl
1990s	Python	Ruby	Haskell
2000s	C#	Go	Nim
2010s	Rust	Julia	Kotlin

Pythonを使いつつ速度を求める

Cython

- PythonをCに変換する
- コードが汚くなる代わりに速い

Numba

- JIT (Just-In-Time)コンパイラ
- 機械語に変換し速度向上を目指す

Pypy

- 速度向上を目指すPythonのヴァージョン