アルゴリズムと計算量

計算量

- コンピュータが計算するための手数
- 現実的な計算可能性を見積もるために重要

線形演算

- 基本演算（Level 1）: 代入，スカラー積，加減算，内積
- ベクトルのサイズ n ➡️ 計算量 $O(n)$

```
## 内積の計算：n回のループなので計算量はO(n)
n = len(x)
s = np.float64(0)
for i in range(n):
        s += x[i] * y[i]
```

- 基本演算（Level 2）: 行列・ベクトル積，列ベクトルと行ベクトルの積
- 行列のサイズ (m, n) ➡️ 計算量 $O(mn)$
  - 次元nの正方行列 ➡️ 計算量 $O(n^2)$

```
## 行列・ベクトル積の計算：m回のループ中にO(n)の内積計算があるので計算量はO(mn)
m, n = np.shape(A) # m 行 n 列
for i in range(m):
        y[i] = np.dot(A[i,:], x)
```

- 基本演算（Level 3）: 行列積
- 行列のサイズ $(m, k), (k, n)$ ➡️ 計算量 $O(mnk)$
  - 次元 n の正方行列 ➡️ 計算量 $O(n^3)$

```
## 行列積の計算：n回のループ中にO(mk)のベクトル・行列積計算があるので計算量はO(mnk)
k, n = np.shape(B) # k 行 n 列
for j in range(n):
        C[:,j] = np.dot(A, B[:,j])
```

サーチ（探索）

- ある条件を満たすデータをデータ集合から探す
- データ処理の基本的な操作の一つ

データ構造とサーチアルゴリズム

- サーチアルゴリズムは「データ構造」に依存する
- 順番に「次のデータ」にしかアクセスできない
  ➡️ データの長さより手数を小さくできない
- 任意の場所のデータにアクセスできる
  ➡️ 二分探索により効率よく探索できる

二分探索

- データが適切にソートされていることが条件
- 条件を満たす値が存在する可能性がある領域を狭めていく
- データの長さ n ➡️ 最悪計算量 $O(\log n)$

```
[1 3 9 10 15 29] で [3] が格納されている位置を見つける

[ 1  3  9 *10  15  29]
  ↓
[ 1 *3  9] 10  15  29
```

ソート

- データ集合を何らかの順番で整列させる
  - 昇順：小さいものから順番にならべる
  - 降順：大きいものから順番にならべる
- データ処理の基本

ソートアルゴリズムと計算量

- バブルソート：シンプルなアルゴリズム
- クイックソート：応用上よく使われるアルゴリズム
- 最悪計算量：最も手間がかかるときの手数

バブルソート

- 全ての要素に対して，隣の要素と比較して順序が逆の場合入れ替える
- データの長さ n ➡️ 最悪計算量 $O(n^2)$

```
[3  1] 5  2  4
 ↓
 1 [3  5] 2  4
 ↓
 1  3 [5  2] 4
 ↓
 1  3  2 [5  4]
 ↓
[1  3  2  4] 5    # 次に最初の4要素について同じように行う
```

クイックソート

- 基準値よりも小さいグループ，大きいグループに分割する
- データの長さ n ➡️ 最悪計算量 $O(n^2)$
- 平均計算量 $O(n \log n)$

```
[3] 1  5  2  4
 ↓
 1  2 [3] 5  4
 ↓
[1  2] 3  5  4   # [1 2] に対してクイックソートを適用
 ↓
 1  2  3 [5  4]  # [5 4] に対してクイックソートを適用
 ↓
 1  2  3  4  5
```