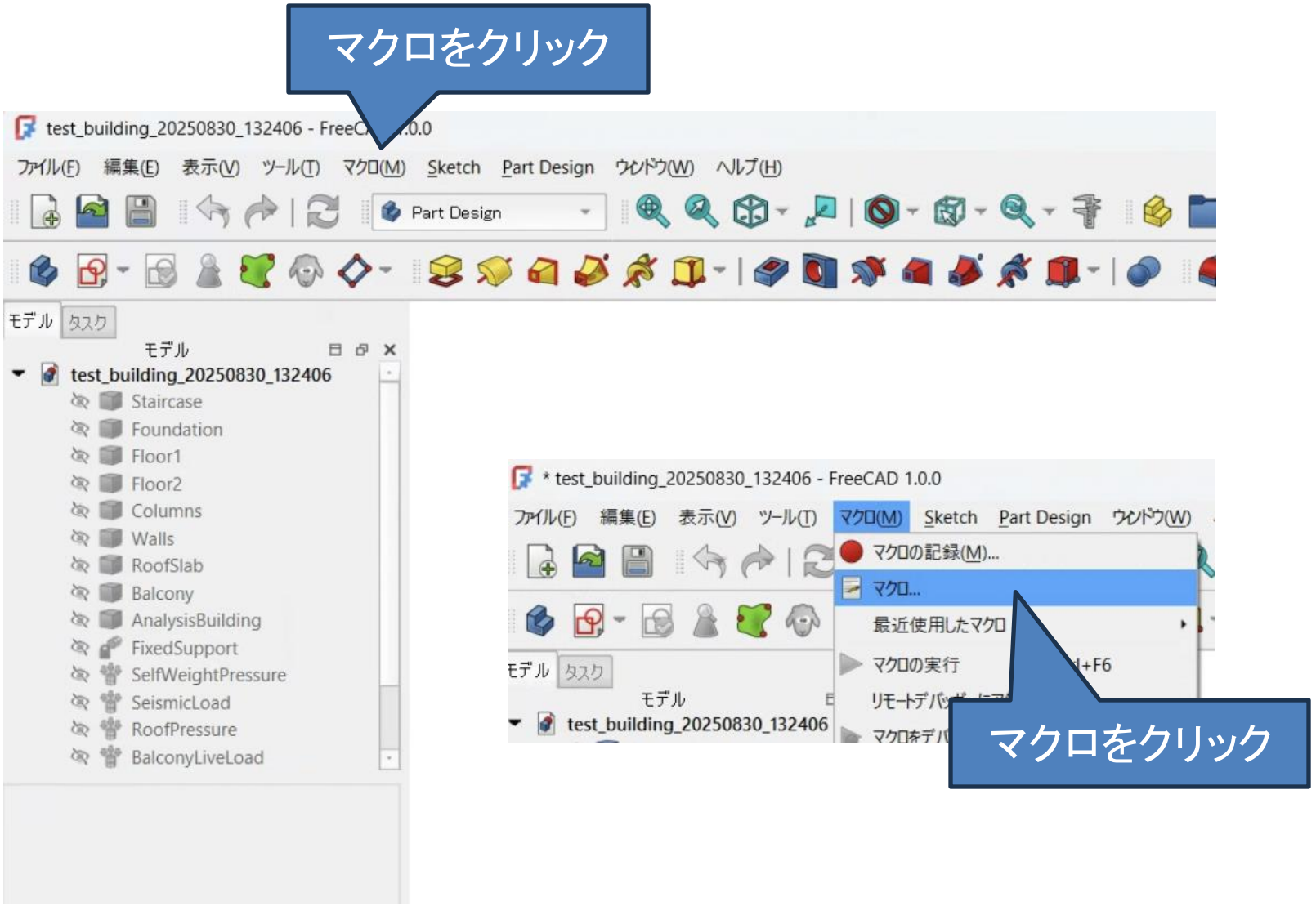
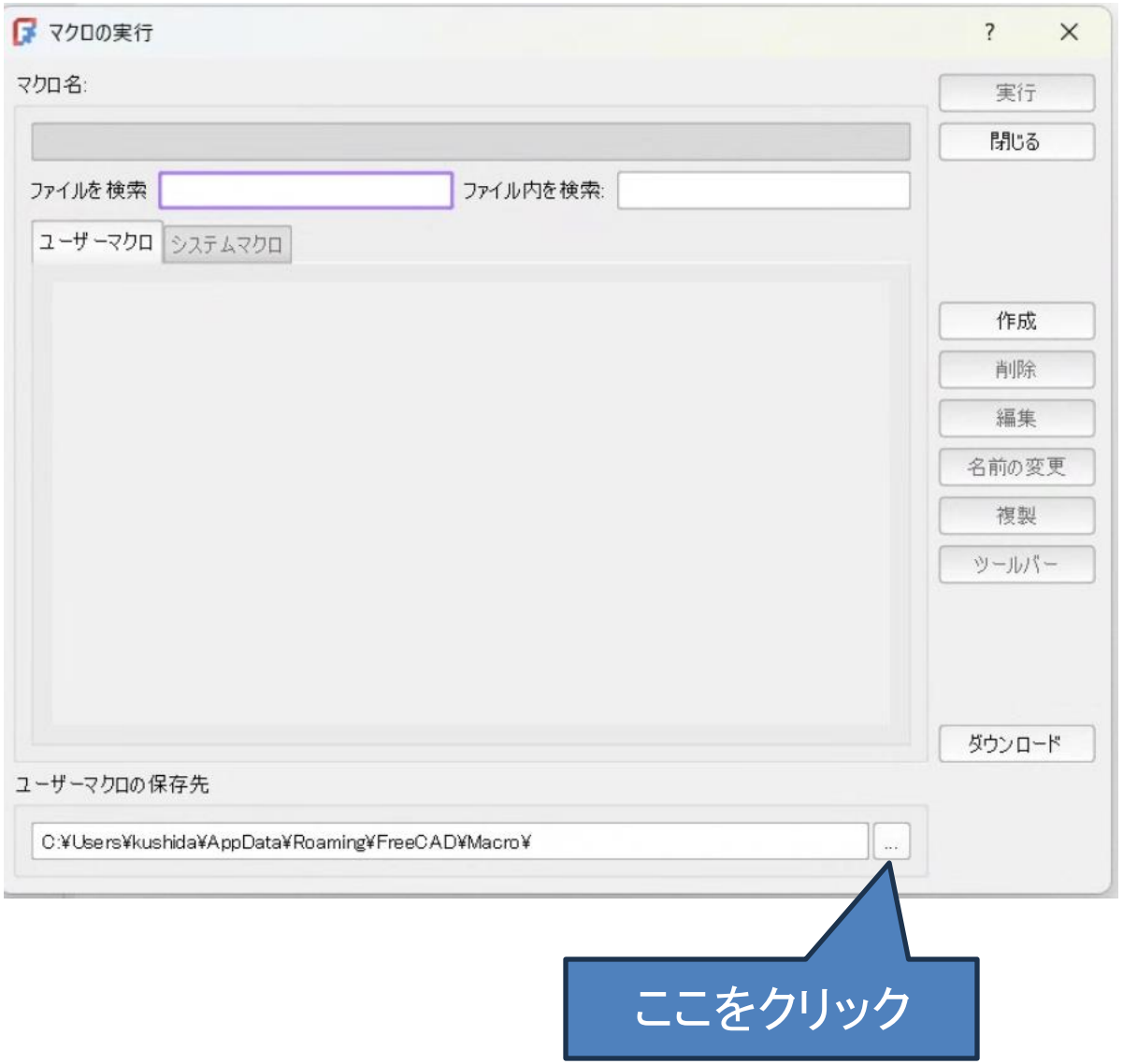
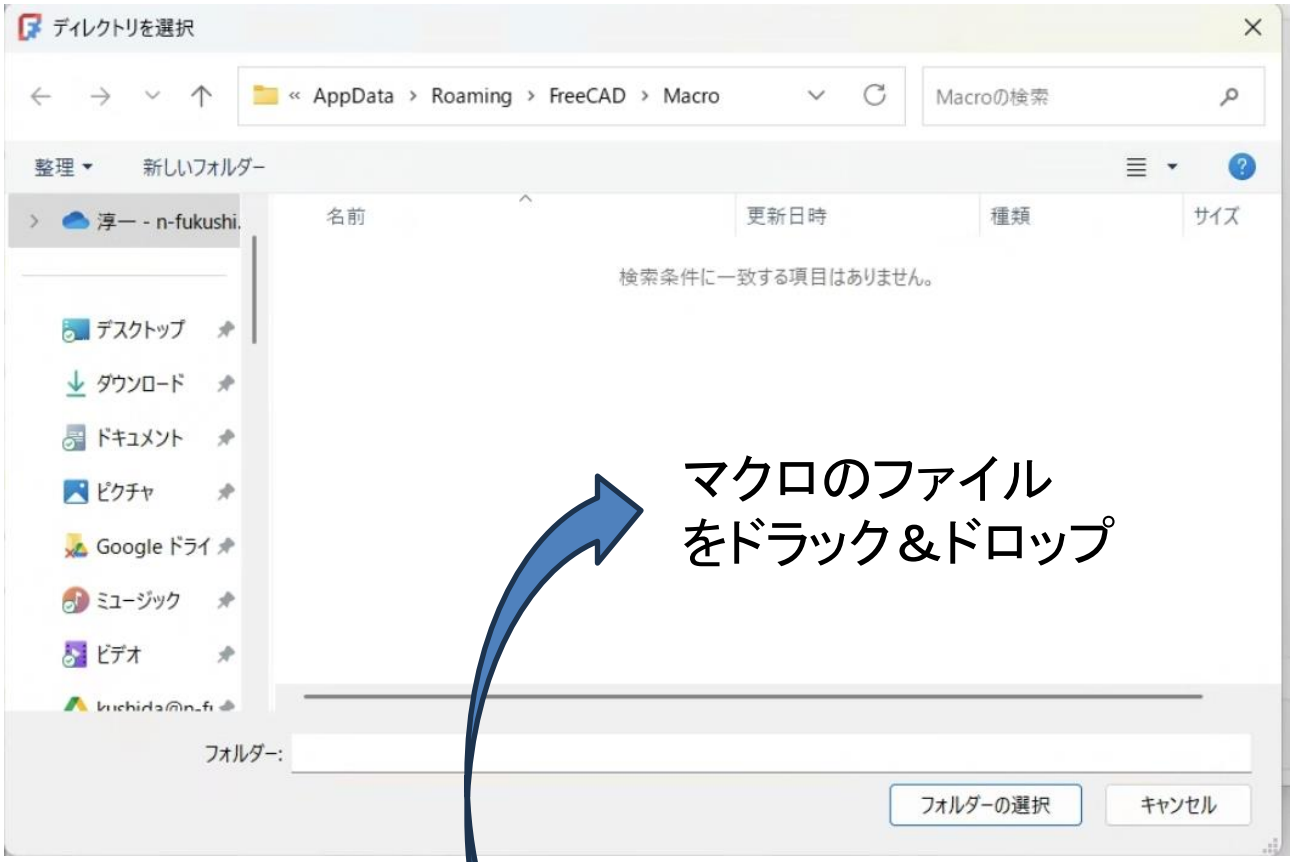
FCstdファイルをダブルクリックするとFreeCADのウインドが立ち上がる



マクロの実行のウインドが立ち上がる



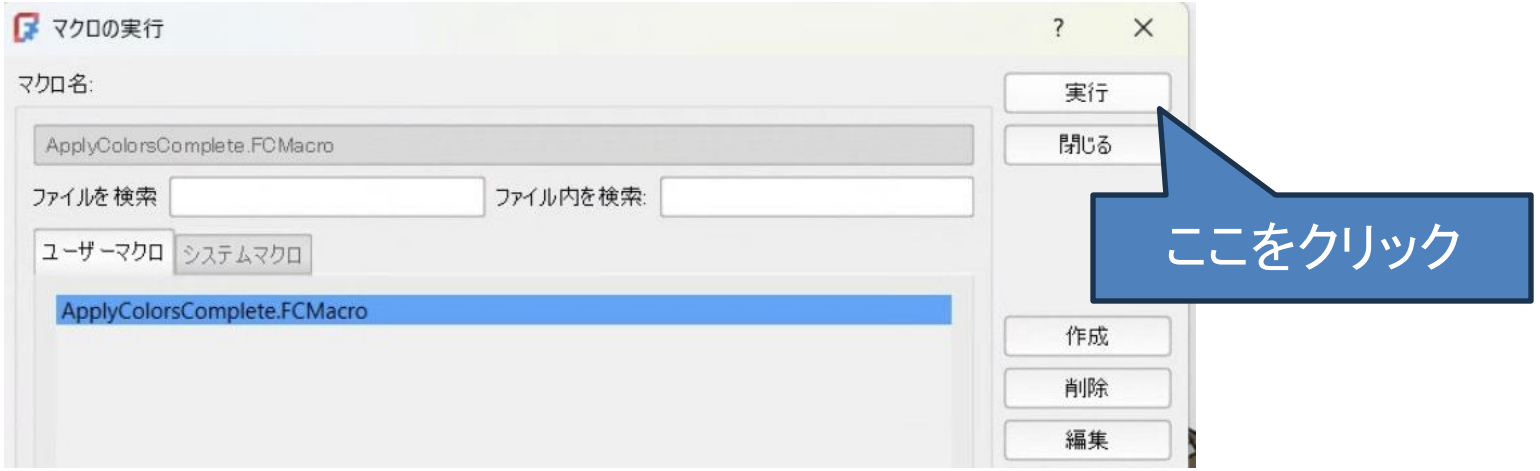
マクロの保存場所のウインドが立ち上がる



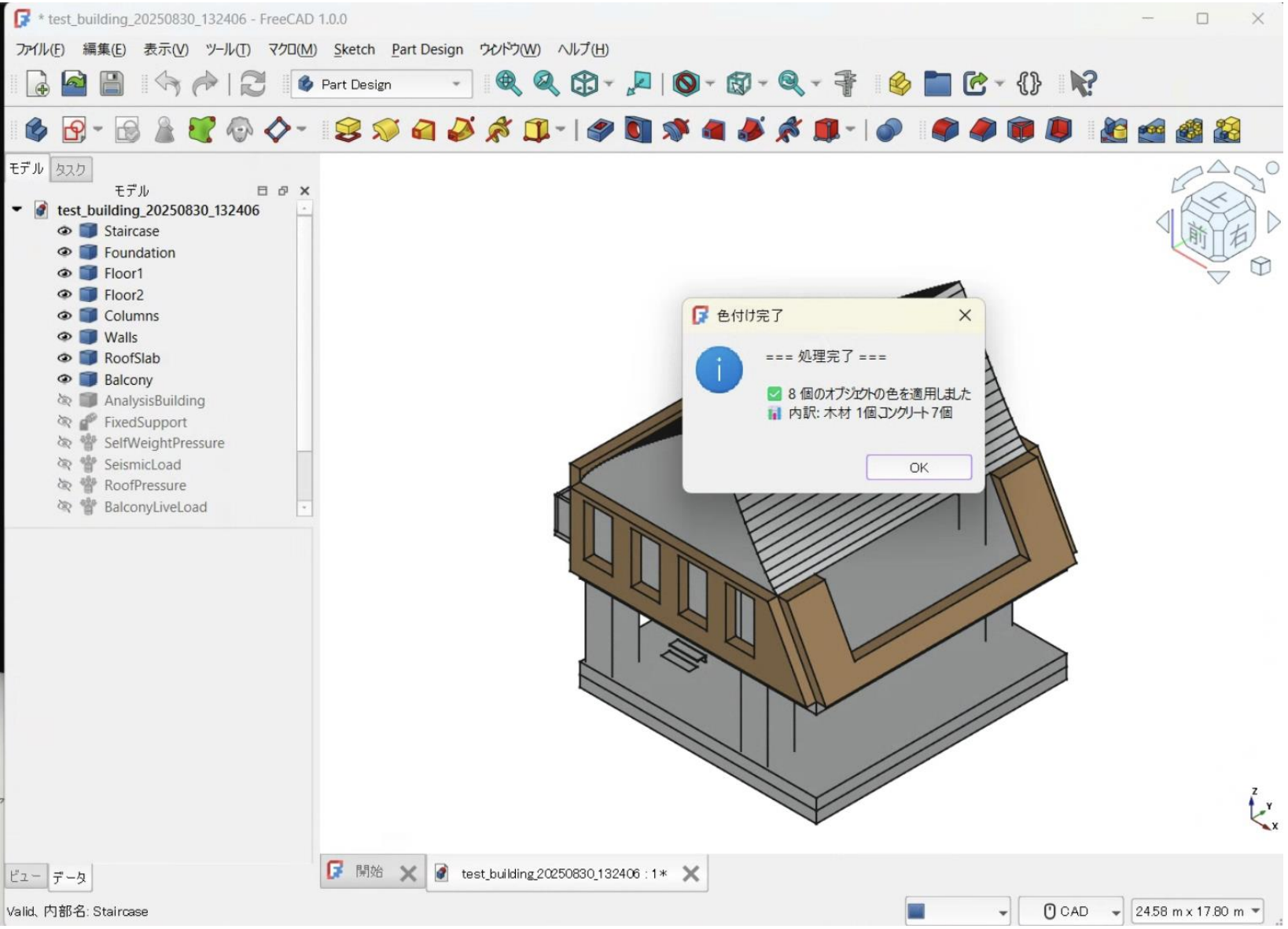
マクロのファイル
をドラック＆ドロップ

マクロのファイルをDLしたフォルダ

1. マクロのファイルをドラック＆ドロップしたら，
   一旦「キャンセル」を押してマクロのウインドを閉じる
2. マクロの実行のウインドもｘで閉じる
3. 再度，マクロの実行のウインドを立ち上げる



マクロが登録されているので実行を押す



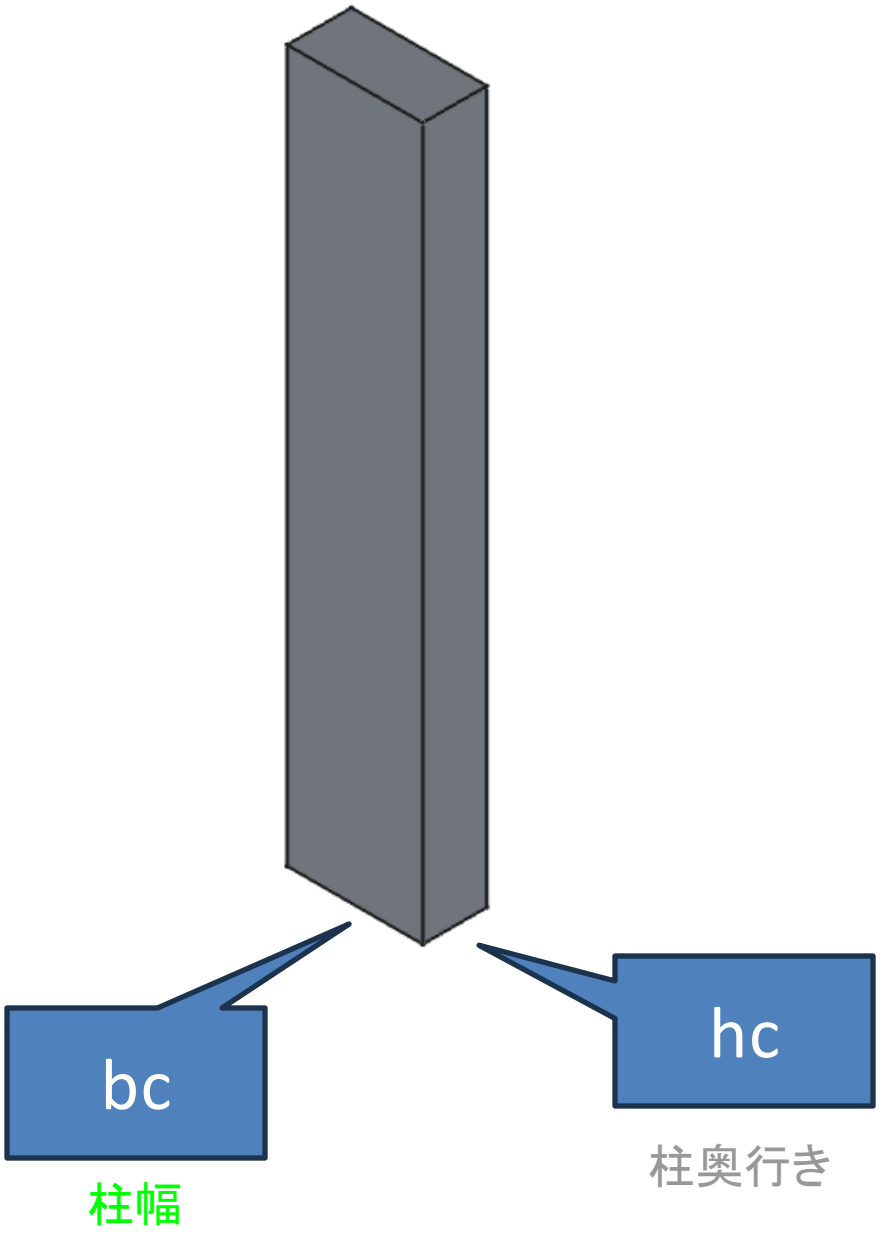
このように建物が表示されたら成功，OKを押す

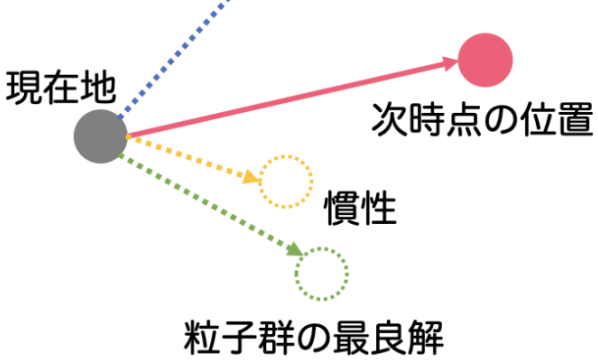
FreeCADの基本操作（標準設定の場合）

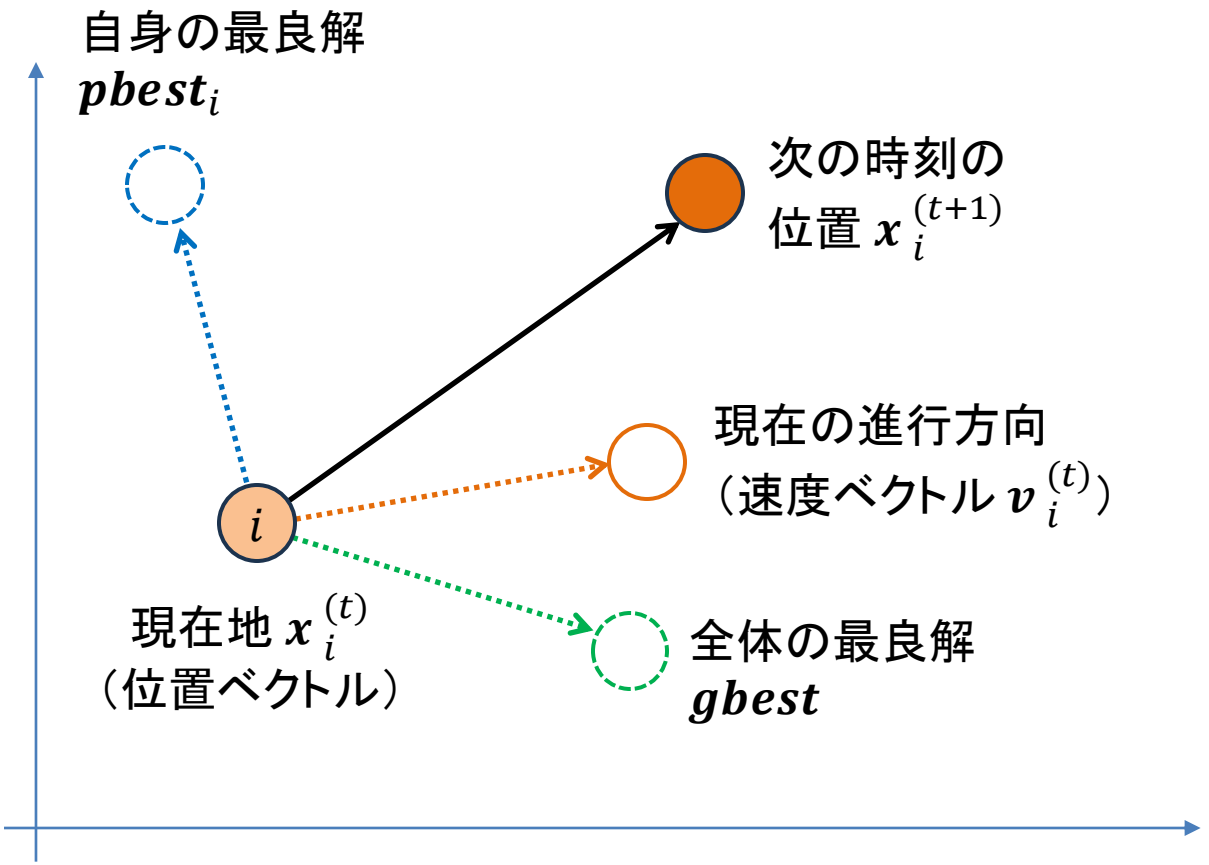
- **回転**
  - マウス中央ボタン（ホイール）を押しながらドラッグ
  - または右クリックと組み合わせると自由回転
- **移動（パン）**
  - Shiftキー ＋ マウス中央ボタンを押しながらドラッグ
- **拡大・縮小**
  - マウスホイールを回転
- **選択**
  - 左クリックでオブジェクトや要素を選択

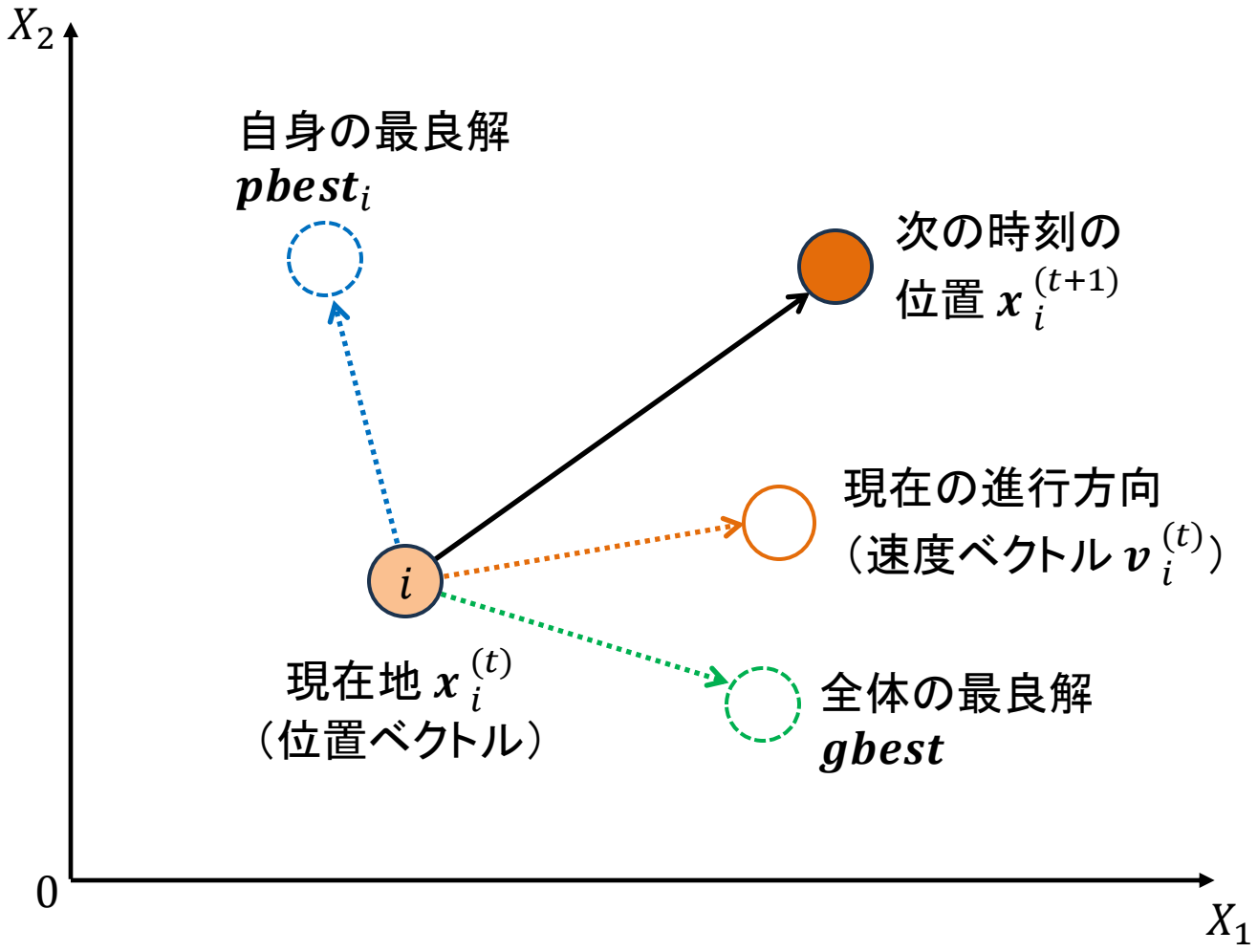
一度マクロを登録すると，以後は「最近使用したマクロ」で
呼び出すことが可能

PCをシャットダウンすると登録したマクロが消えるので，
その際はもう一度マクロを登録する

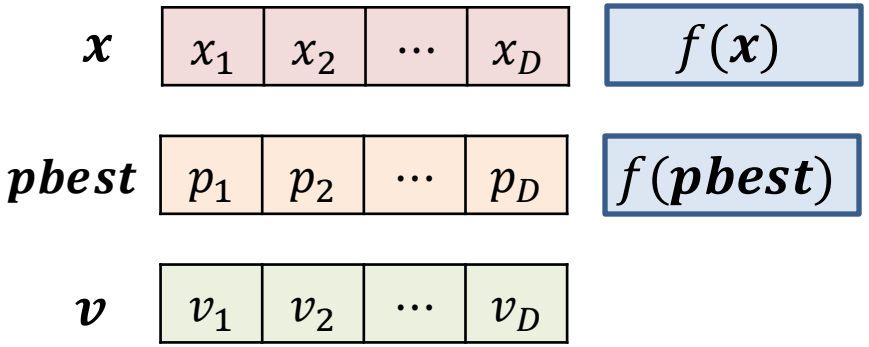

bc
hc
柱幅
柱奥行き


現在地
次時点の位置
慣性
粒子群の最良解

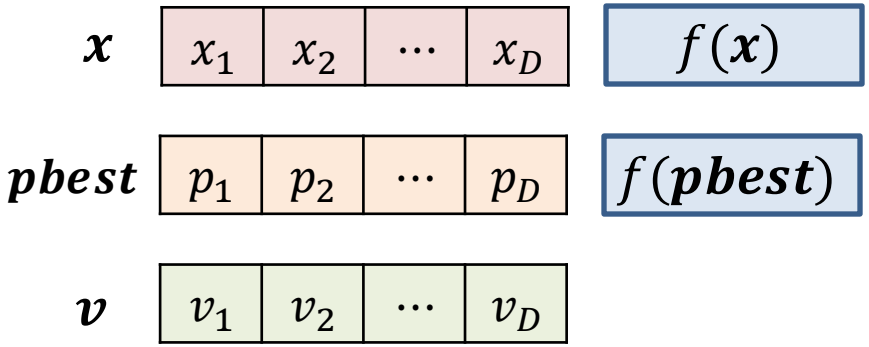

自身の最良解
$pbest_i$
次の時刻の
位置 $x_i^{(t+1)}$
現在の進行方向
（速度ベクトル $v_i^{(t)}$）
i
現在地 $x_i^{(t)}$
（位置ベクトル）
全体の最良解
$gbest$


X_2
自身の最良解
$\boldsymbol{pbest}_i$
次の時刻の
位置 $\boldsymbol{x}_i^{(t+1)}$
現在の進行方向
（速度ベクトル $\boldsymbol{v}_i^{(t)}$）
i
現在地 $\boldsymbol{x}_i^{(t)}$
（位置ベクトル）
全体の最良解
$\boldsymbol{gbest}$
0
X_1

ステップ t



ステップ $t+1$



3. 速度更新式の詳細

完全な速度更新方程式

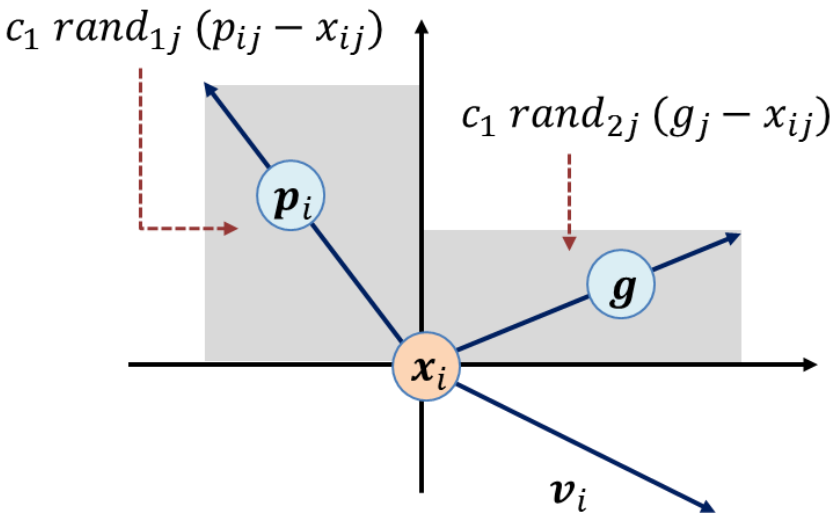
$$\mathbf{v}_i(t+1) = \underbrace{w \cdot \mathbf{v}_i(t)}_{\text{慣性項}} + \underbrace{c_1 \cdot r_1 \cdot (\mathbf{p}_i - \mathbf{x}_i(t))}_{\text{認知成分}} + \underbrace{c_2 \cdot r_2 \cdot (\mathbf{g} - \mathbf{x}_i(t))}_{\text{社会成分}}$$

各成分の役割

- **慣性項**: 現在の速度を維持（探索能力）
- **認知成分**: 個体の経験への回帰（利用能力）
- **社会成分**: 群れの成功への追従（協調）

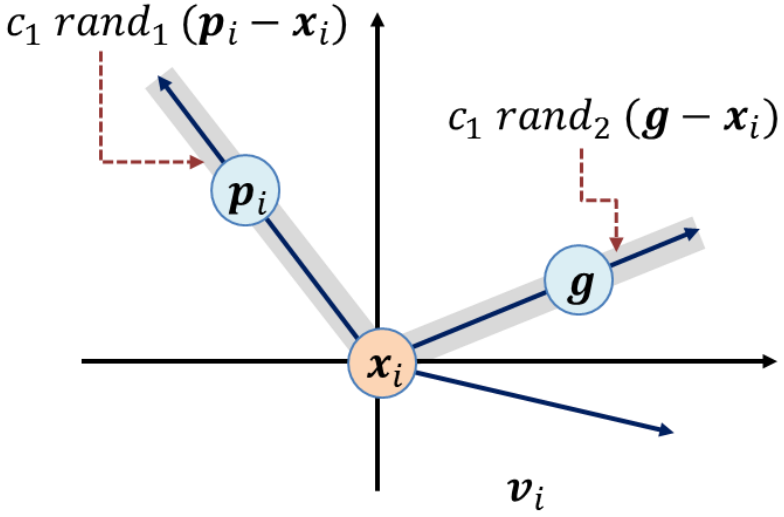
PSOの回転不変性

通常のPSO(乱数非同期型)



回転不変性を持たない
→ 灰色の領域は座標軸の取り方に依存

乱数同期型PSO
（各次元の乱数が同じ）



回転不変性を持つ
(ただし，多様性は低下)

PSOの擬似コード

初期位置，初期速度をランダムで初期化

○PSO の擬似コード

Initialize A(0)={ $(\boldsymbol{x}_i, \boldsymbol{v}_i) \mid i=1,2,...,N$ }，$(\boldsymbol{x}_i^*=\boldsymbol{x}_i)$，$N$は個体数（集団サイズ）

Evaluate A(0)，$(pbest_i=f(\boldsymbol{x}_i^*))$

G=A(0)の最良エージェント

for(t=1; 終了条件を満足しない; t++) { // t は反復回数

  for(i=1; i≦N; i++) {

    for(各次元 j) {

      $v_{ij} = wv_{ij} + c_1 rand_{1ij}\left(x_{ij}^* - x_{ij}\right) + c_2 rand_{2ij}\left(x_{Gj}^* - x_{ij}\right);$

      if($v_{ij} < -V_j^{max}$) $v_{ij} = -V_j^{max}$; else if($v_{ij} > V_j^{max}$) $v_{ij} = V_j^{max}$;

      $x_{ij} = x_{ij} + v_{ij};$

    }

    if($f(\boldsymbol{x}_i)$<$pbest_i$) {

      $\boldsymbol{x}_i^* = \boldsymbol{x}_i$; $pbest_i=f(\boldsymbol{x}_i)$;

      if($pbest_i$<$gbest$) { $G=i$; $gbest = pbest_i$; }

    }

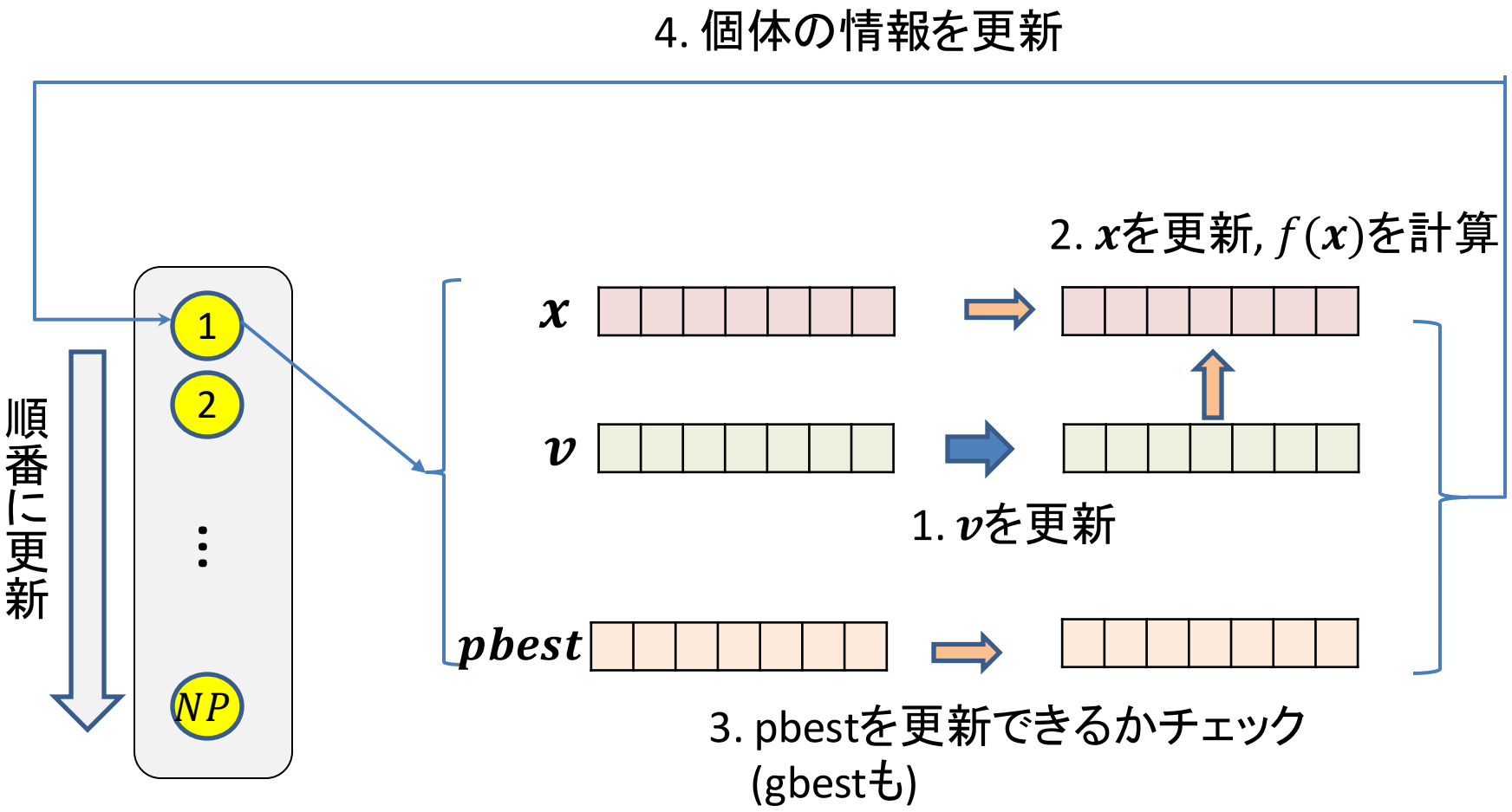
  }

}

- 速度の上限下限をチェック
- $V^{\max}$は定義域を基準に設定

- 移動後の位置を評価：$f(x)$を計算
- Pbestを超えていれば置換え
- さらにgbestもチェック

PSOの処理手順

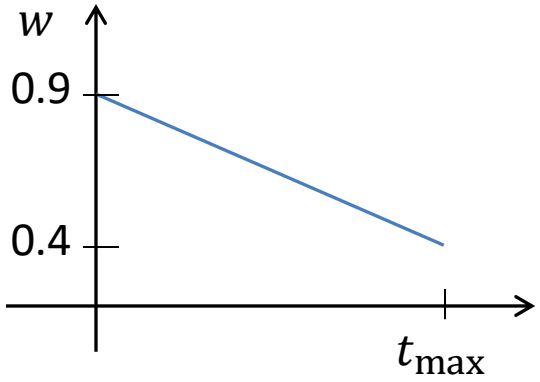


- pbest, gbestの更新は個体ごとに行われる （ **Unsynchronous** model）
- 最後の方の番号の個体は，最初の個体よりも良いgbestを使える場合がある

PSOのパラメータ設定

- Linearly Decreasing Inertia Weight Method(LDIWM)
  - 不安定領域から安定領域へ
  - $c_1 = c_2 = 2.0$ とし，wを0.9から0.4へ減少させる

サンプルプログラムのデフォルト設定



- Construction Method (CM)
  - パラメータを弱い安定領域に設定
  - $w = 0.729,\ c_1 = c_2 = 1.4955$

- Random Inertia Weight Method (RIWM)
  - $c_1 = c_2 = 1.4955$ とし，$0.5 \leq w \leq 1.0$ の間で状態更新毎にランダムに w を決定

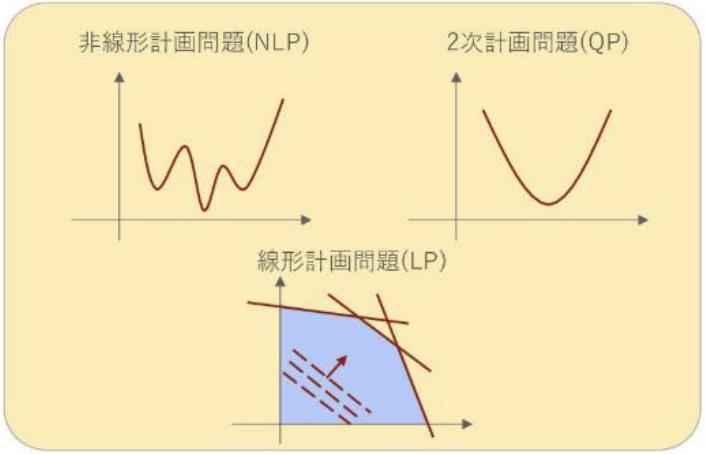
最適化問題の基礎知識｜代表的な最適化問題

ひとくちに最適化問題と言ってもさまざまな最適化問題が存在します

決定変数、目的関数、制約条件の性質によって以下のように分類されることがあります

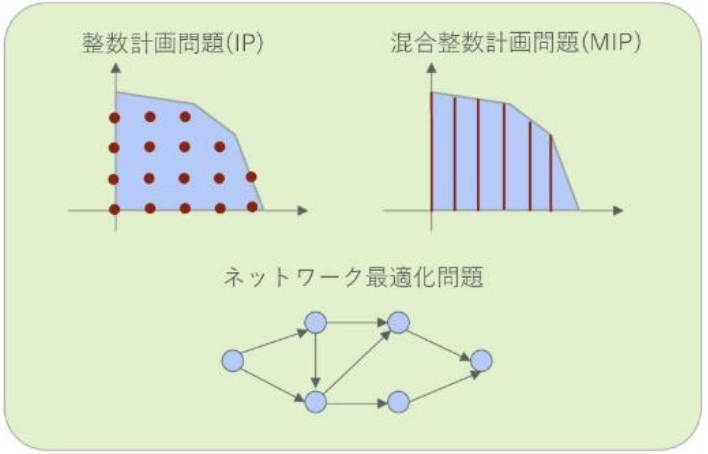
連続最適化

決定変数が実数値のような連続的な値を取る最適化問題



離散最適化（組合せ最適化）

決定変数が整数値や{0, 1}の2値のような離散的な値をとる最適化問題や、最適解を含む解の集合が順列やネットワークなど組合せ的な構造を持つ最適化問題



引用元：梅谷俊治「しっかり学ぶ数理最適化」

*ステップは0から開始

処理中のステップ	最良適応度	粒子数	評価回数	経過時間
4 / 19	4.113e+5	15	75	3.2 min

処理中のステップ	粒子数	評価回数	評価回数	経過時間	

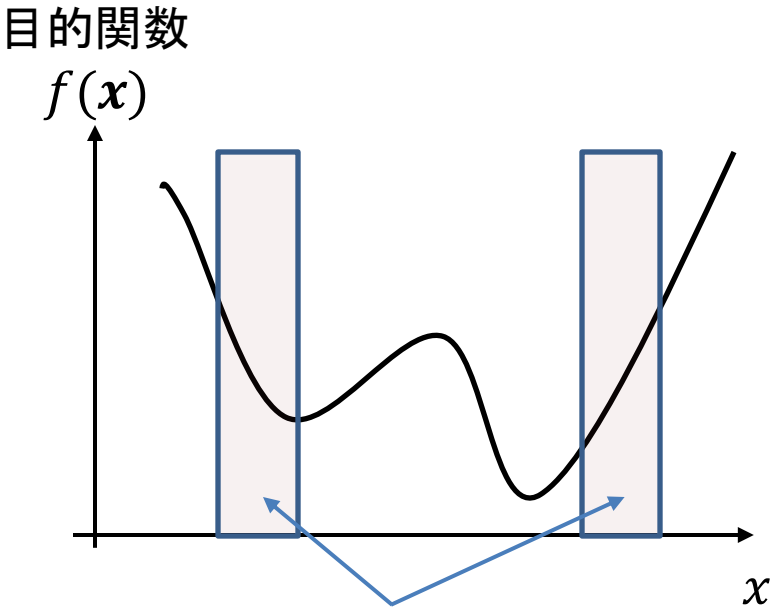
最適化問題を解く手順

1. 問題の定義:
   - 最適化問題を明確に定義する（達成したい目的，守るべき制約など）
2. 定式化:
   - 問題を数学的に表現する．目的関数と制約条件を数式に変換する
3. 解法(solver)の選択:
   - 定式化された問題に対して，適切な解法(最適化手法)を選択する
4. 解の探索:
   - 選択した解法を使って，問題の解を求める
5. 解の検証:
   - 得られた解が適切か（制約条件を満たしているか）を確認する

目的関数と制約条件

目的関数（Objective Function）

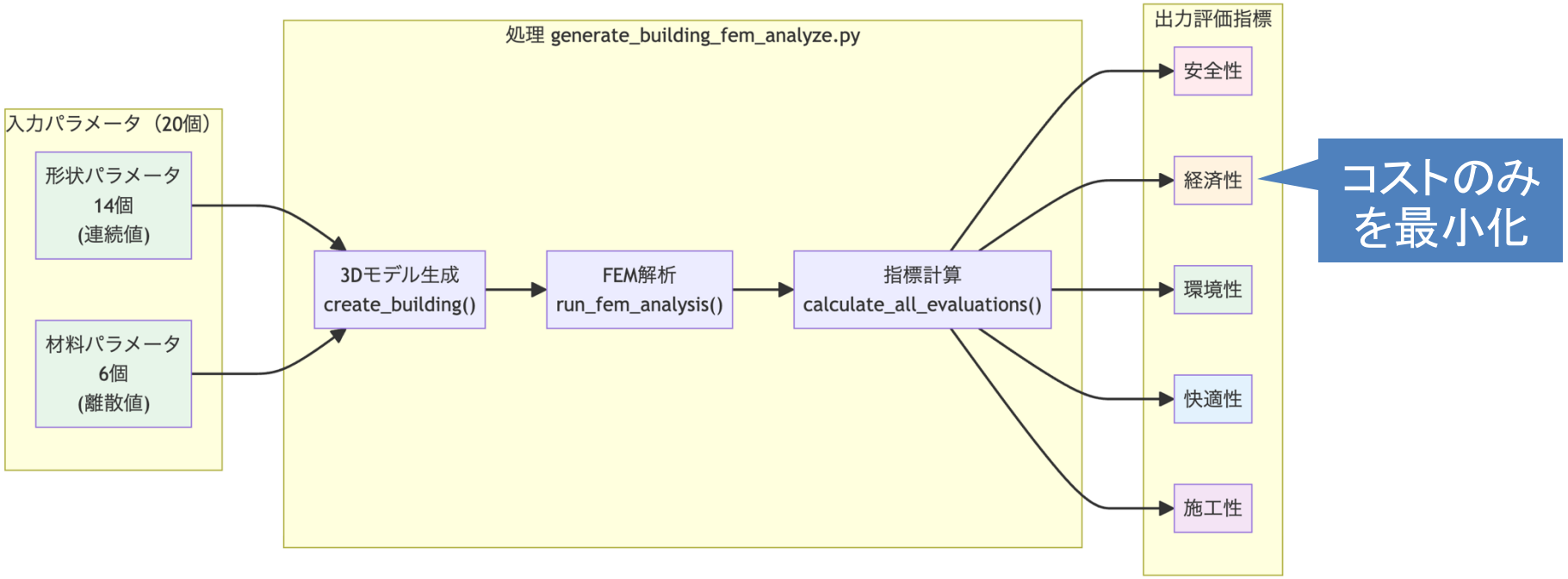
- 最適化問題で求めたい結果を表す数式
- 値を変化させるものが変数
- 最小 or 最大にしたい関数値が目的関数
  - （例：利益，コスト，効率など）



制約条件（Constraints）

- 最適化問題の解が満たすべき条件を表す式
- 制約条件は，目的関数を最大化または最小化すると同時に満たす必要がある
- （実世界の問題ならば）制約条件は，資源の制限，法律的制限，技術的制限など

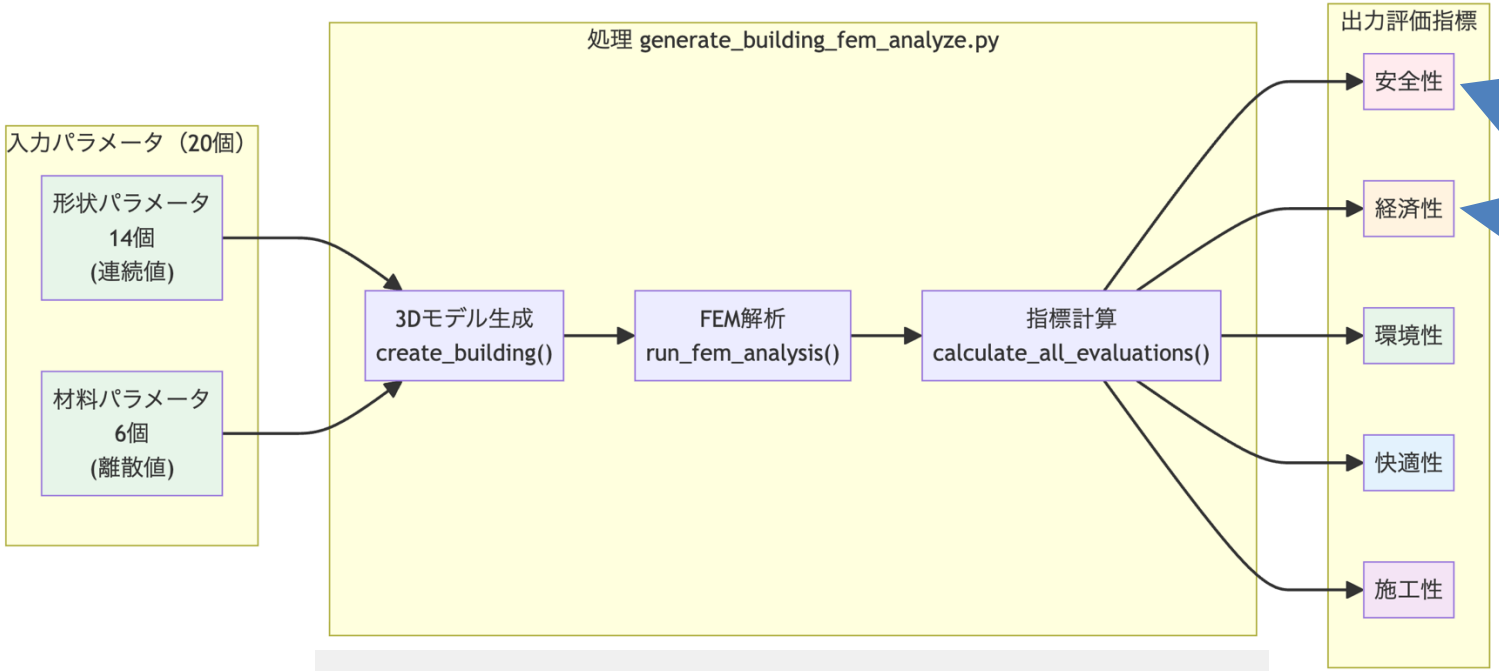
適応度関数の意味




```
def calculate_fitness(cost, safety, co2, comfort, constructability):

    # 基本適応度：コストのみ
    fitness = cost

    return fitness
```

コストが小さいほど良い設計

適応度関数の意味



安全性を
考慮した上で
コストを
最小化

```
def calculate_fitness(cost, safety, co2, comfort, constructability):

    # 安全率の閾値
    SAFETY_THRESHOLD = 2.0

    # 基本適応度：コストのみ
    fitness = cost

    # 安全率ペナルティ
    if safety < SAFETY_THRESHOLD:
        fitness += (SAFETY_THRESHOLD - safety) * 100000

    return fitness
```

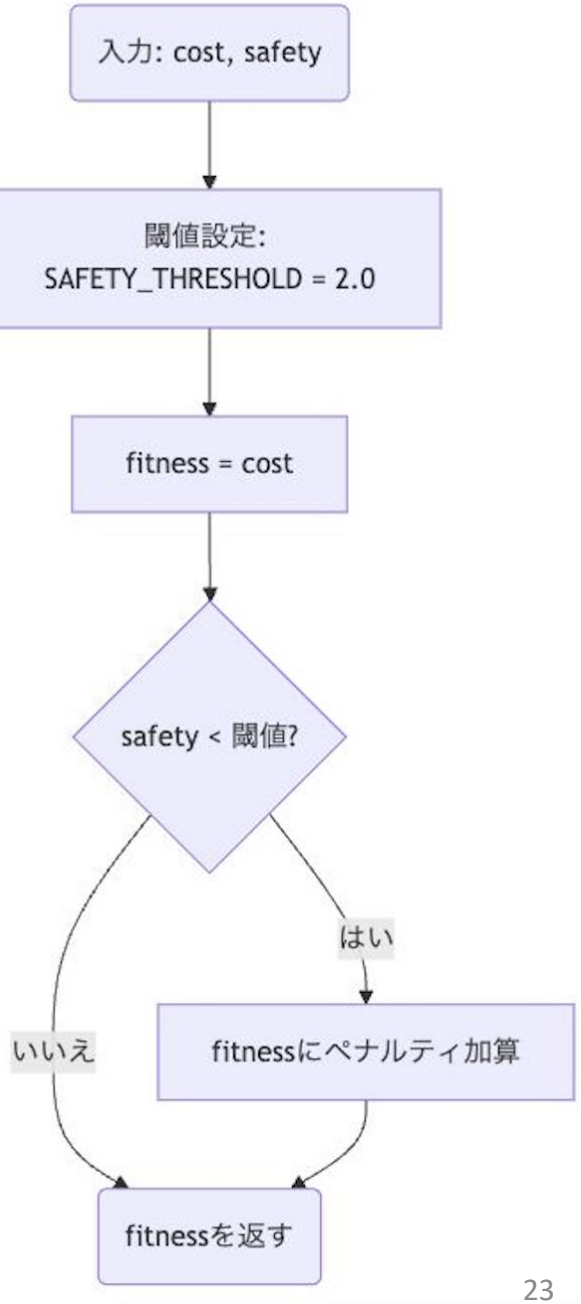
コストが小さくても
安全率が 2 より小さいと
ペナルティを加算

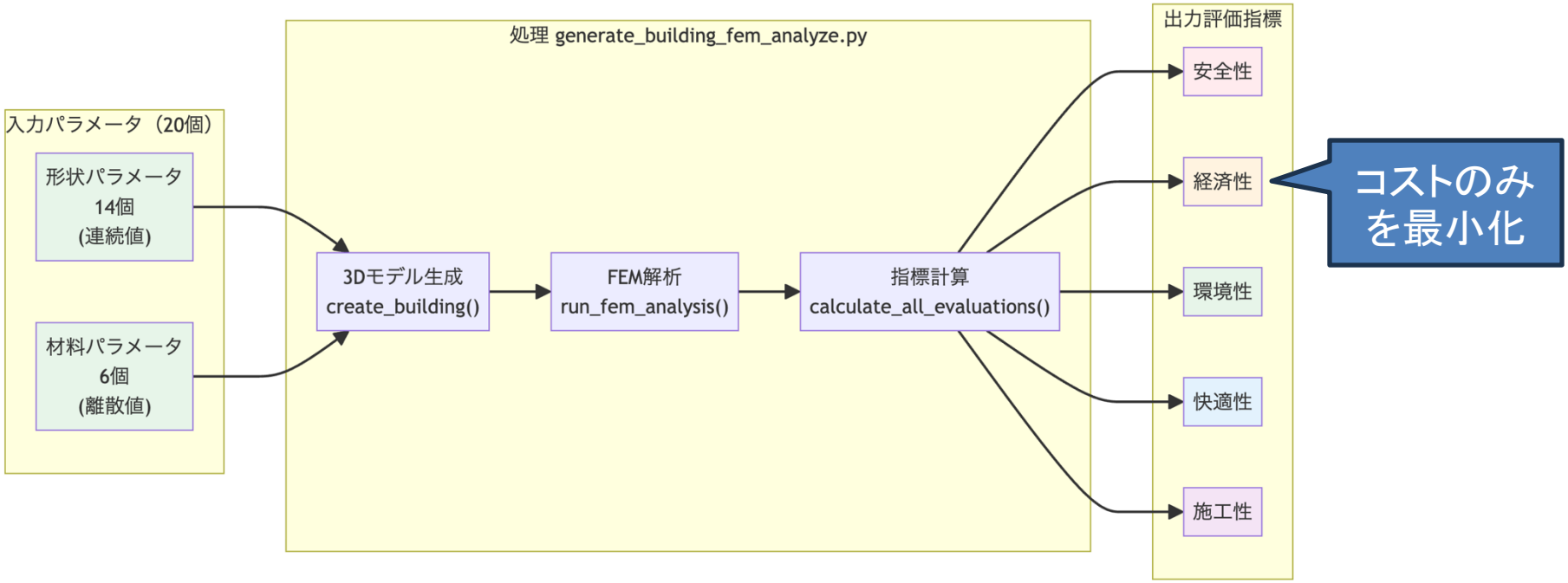
ペナルティの加算

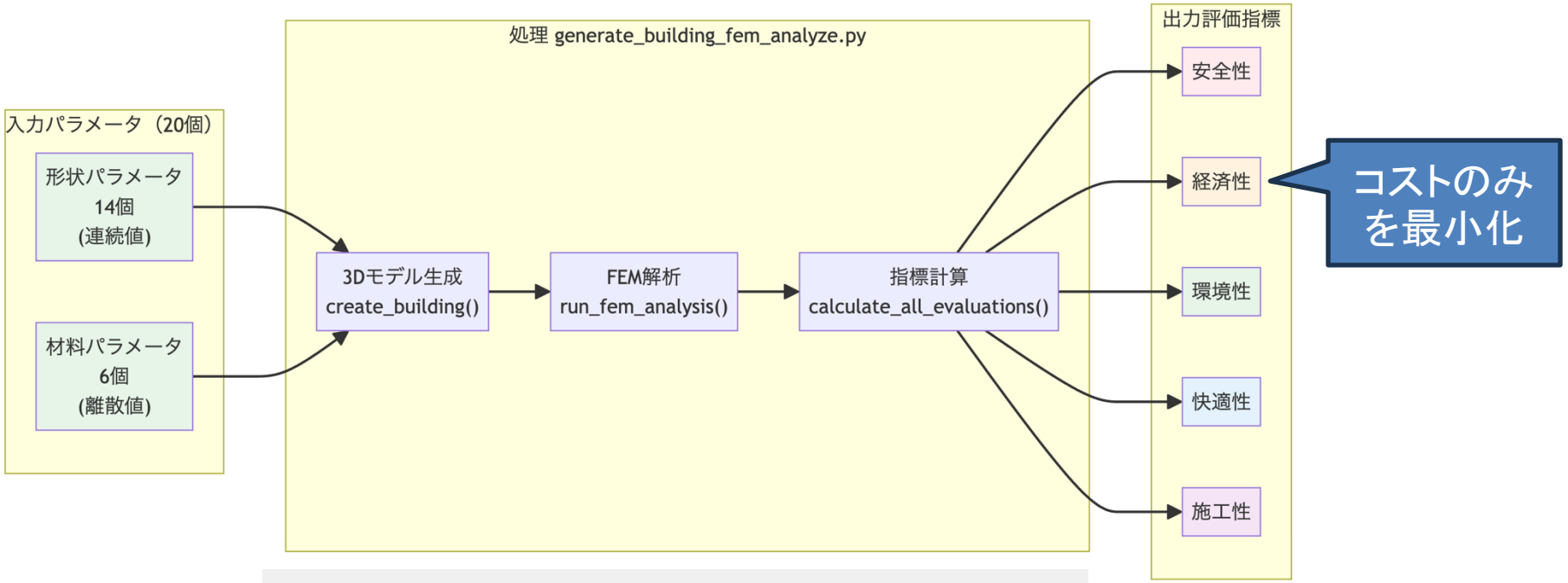
```
def calculate_fitness(cost, safety, co2, comfort, constructability):

    # 安全率の閾値
    SAFETY_THRESHOLD = 2.0

    # 基本適応度：コストのみ
    fitness = cost

    # 安全率ペナルティ
    if safety < SAFETY_THRESHOLD:
        fitness += (SAFETY_THRESHOLD - safety) * 100000

    return fitness
```

最低限確保すべき安全率を満たしていない場合は，その不足分に 100000 を掛けた値を fitness に加算（コストに上乗せ）する

「コストと安全率」を1つの指標に合成




入力パラメータ（20個）
形状パラメータ
14個
(連続値)
材料パラメータ
6個
(離散値)
処理 generate_building_fem_analyze.py
3Dモデル生成
create_building()
FEM解析
run_fem_analysis()
指標計算
calculate_all_evaluations()
出力評価指標
安全性
経済性
環境性
快適性
施工性
コストのみ
を最小化




```
def calculate_fitness(cost, safety, co2, comfort, constructability):

    # 安全率の閾値
    SAFETY_THRESHOLD = 2.0

    # 基本適応度：コストのみ
    fitness = cost

    # 安全率ペナルティ
    if safety < SAFETY_THRESHOLD:
        fitness += (SAFETY_THRESHOLD - safety) * 100000
```

事前に決めた上下限内の
範囲からランダムに値を決定

パラメータ	最小値	最大値	ランダムな値
Lx	8.0	12.0	9.5
Ly	6.0	12.0	7.5
H1	2.6	3.5	3.1
H2	2.6	3.2	2.9

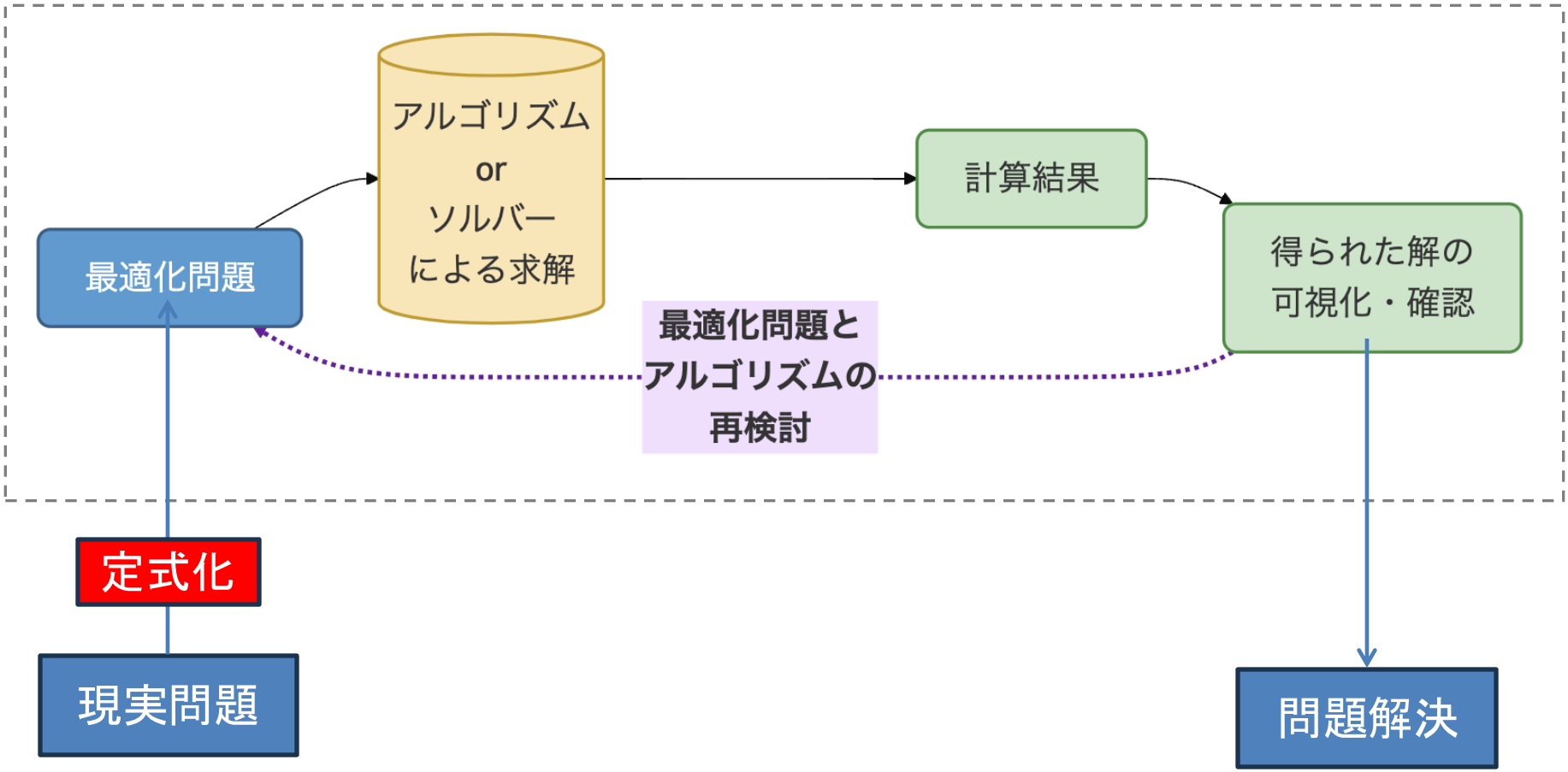
評価プログラム
generate_building_fem_analyze.py

5つの評価指標

サンプリング結果

production_freecad_random_fem_evaluation.csv

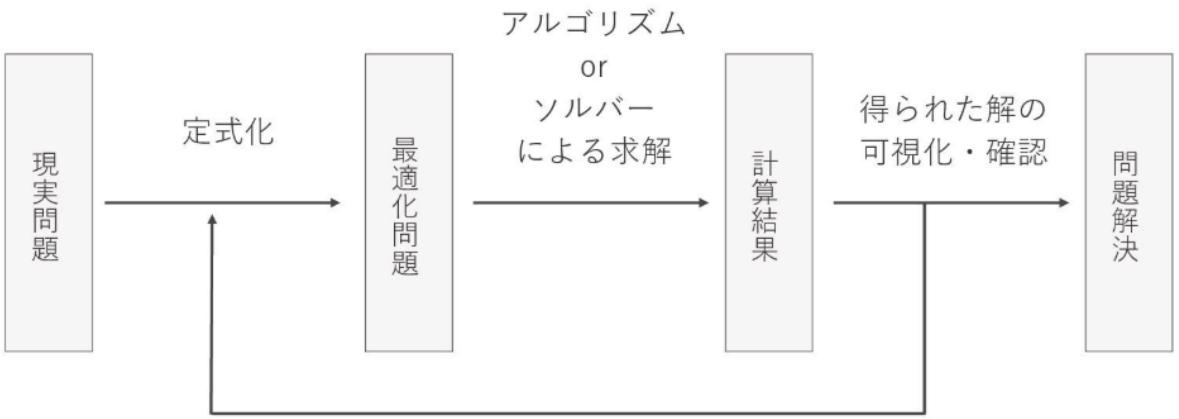
htmlで可視化


アルゴリズム
or
ソルバー
による求解
計算結果
最適化問題
得られた解の
可視化・確認
最適化問題と
アルゴリズムの
再検討
定式化
現実問題
問題解決

最適化案件の進め方

最適化案件は以下の一連の手続きからなります

定式化がアルゴリズムが一発で決まることは稀でお客さんのフィードバックをもらいながら何回もサイクルを回します



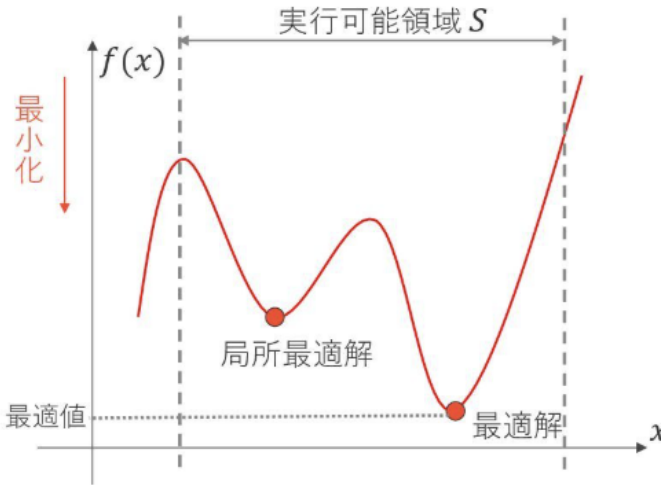
最適化問題の基礎知識｜用語

続いて基本的な最適化における用語について確認していきます

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{x} \in S. \end{aligned}$$

最適化問題は以上のような目的関数と制約条件から構成されます

用語	意味
決定変数	最適化対象の変数（上の問題のx）
解	決定変数に割り当てられた値
実行可能解	制約条件を満たす解
実行可能領域	実行可能解全体の集合
(大域)最適解	目的関数の値が最大or最小となる解
局所最適解	近傍内の任意の解よりも目的関数値が大きいor小さい解
最適値	最適解における目的関数の値



引用元：梅谷俊治「しっかり学ぶ数理最適化」

メリット ✅

- **実装が簡単**
  - 基本的な演算のみで実現可能
- **パラメータが少ない**
  - w, c_1, c_2, 粒子数程度
- **並列化が容易**
  - 各粒子の計算は独立
- **連続最適化に強い**
  - 実数値の最適化問題に適している
- **局所解からの脱出**
  - 確率的な動きにより可能

デメリット ✗

- **収束の保証なし**
  - 必ずしも最適解に到達しない
- **離散最適化は苦手**
  - 連続値を前提としたアルゴリズム
- **高次元では性能低下**
  - 次元の呪いの影響を受ける
- **パラメータ調整が必要**
  - 問題に応じた調整が重要

PSOのプログラム

プログラム中の変更可能なパラメータ

- 個体数(Nindividuals)
- 認知パラメータ(C1)
- 社会パラメータ(C2)
- 慣性係数の初期値(W_0)
- 慣性係数の最終値(W_T)
- 最大速度(MAX_V)

LDIWM
Wを除々に減少(W_0 → W_T)

PSOのパラメータ

```
/* 試行回数(平均性能を調べるため) */
#define RUN_MAX                 30
#define MAX_EVALUATIONS         200000
/* 個体数 */
#define Nparticles          20
/* 最大世代数(繰り返し回数) */
#define T_MAX               (MAX_EVALUATIONS/Nparticles)
/* パラメータ */
#define C1          2.0
#define C2          2.0
#define W_0         0.9
#define W_T         0.4
#define MAX_V       0.5*(UPPER-LOWER)
```

PSOの評価回数

#define MAX_EVALUATIONS　200000

DE

```
/* 個体数 *
#define Nindividuals　50
/* 最大世代数(繰り返し回数) */
#define T_MAX　　(MAX_EVALUATIONS/Nindividuals)
```

デフォルトでは
T_MAX=4000

PSO

```
/* 個体数 */
#define Nparticles　20
/* 最大世代数(繰り返し回数) */
#define T_MAX　　(MAX_EVALUATIONS/Nparticles)
```

デフォルトでは
T_MAX=10000