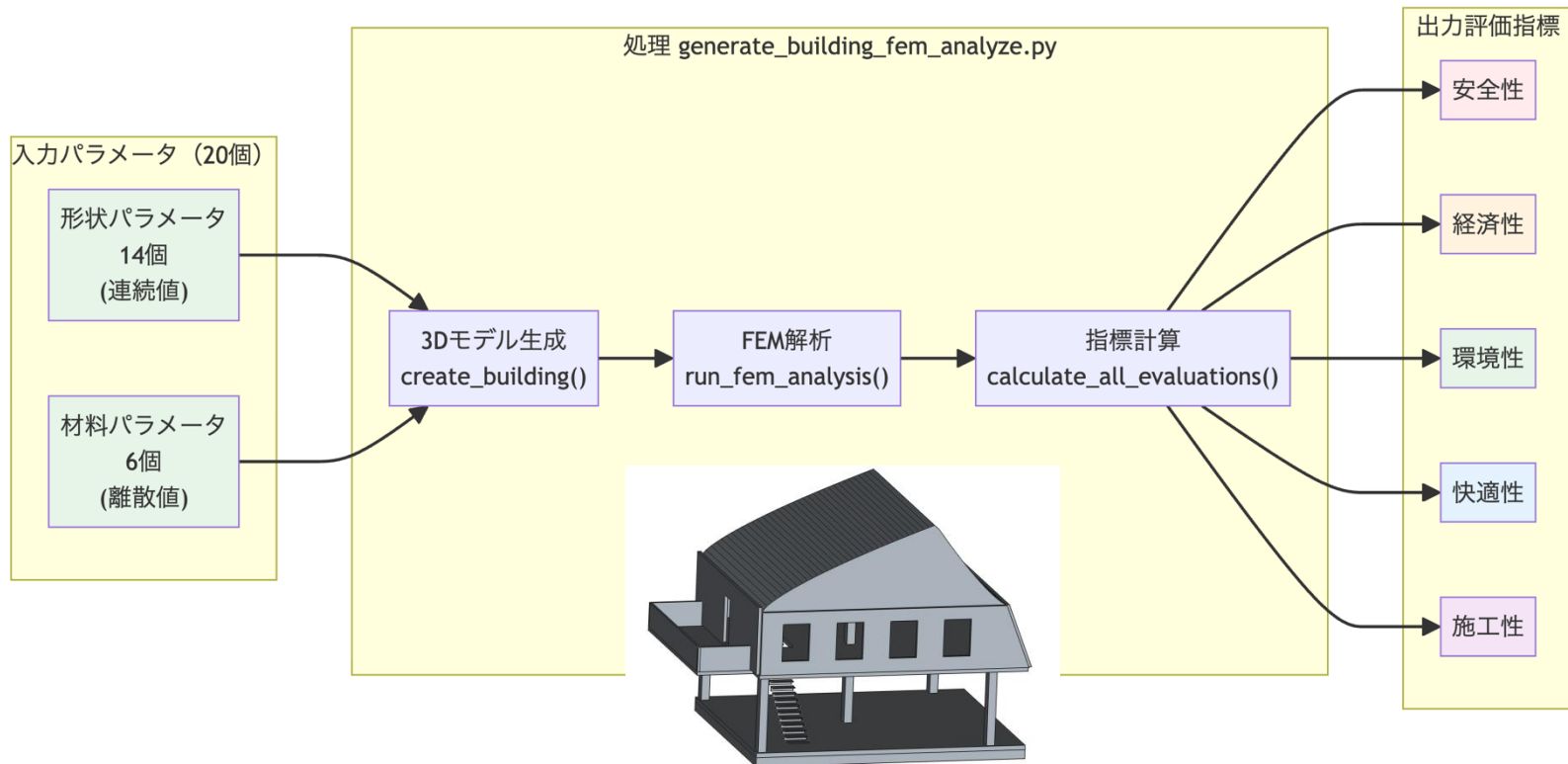


建築デザイン設計問題へのPSOの適用

- PSOによる建築設計の最適化
 - 何を最適化するか？
 - 何を目的として最適化を行うか？

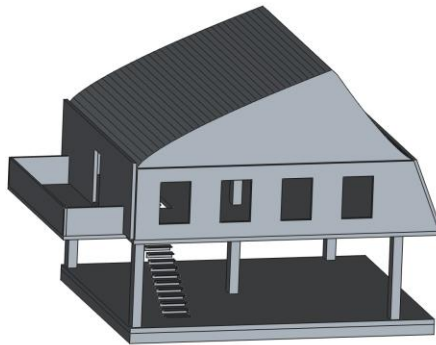


ブラックボックス最適化としての定式化

設計変数

x_1
x_2
x_3
\vdots
x_{20}

入力



設計変数に従い
建物を設計

FEM
解析

出力

評価指標

f_1 : 安全率
f_2 : 建築コスト
f_3 : CO2排出量
f_4 : 快適性
f_5 : 施工性

$$\mathbf{y} = (f_1, \dots, f_5)$$

$$\mathbf{y} = f(\mathbf{x})$$

最適化の
対象

f はブラックボックス関数

建物のパラメータ (設計変数 \mathbf{x}) を
入れると評価指標 \mathbf{y} が返ってくる

ただし、内部の挙動
(数式) はわからない

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. 設計変数 x を定義し, 目的関数・制約条件を設計変数の関数 $f(x)$ として表現する

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

内部の数式を知らなくても,
試行錯誤的に良い解を探す手法

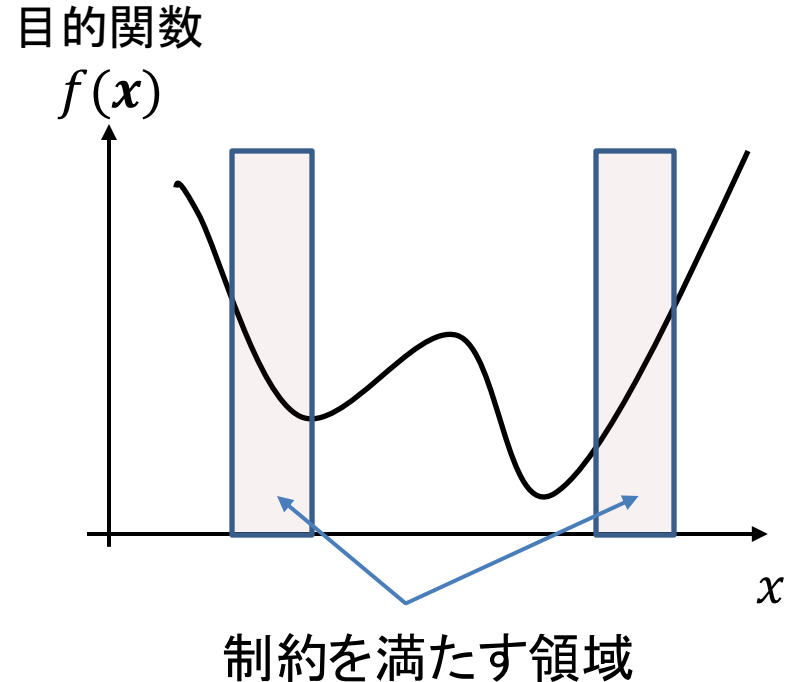
5. 解の検証:

- 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)

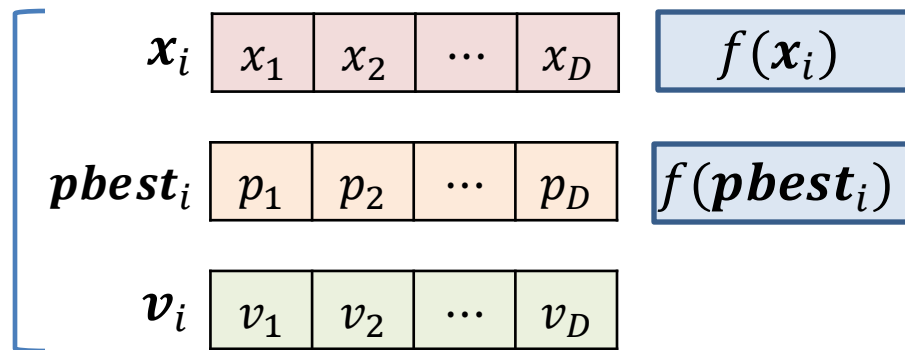


制約条件 (Constraints)

- 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

PSOの設計変数 x と適応度関数 f

各粒子 i の持つ情報 ($i = 1, \dots, N$):



D 次元ベクトル

N は粒子数

D は問題の次元数

最小化問題

達成したい目的に応じた
適応度関数を設定
(制約条件も考慮)

最適化の対象

$x = (x_1, x_2, \dots, x_{20})$ $D = 20$

設計変数 x はすべて連続値とし、
事前に決めた範囲内でランダムに生成
(探索中もはみ出さないように修正)

20
個の
変数

適応度関数 f

適応度

この値を小さくする
ような x を探索

設計変数 x の扱い: 形状パラメータ

変数	変数名	意味	単位
x_1	Lx	X方向長さ	m
x_2	Ly	Y方向長さ	m
x_3	H1	1階高さ	m
x_4	H2	2階高さ	m
x_5	tf	床スラブ厚	mm
x_6	tr	屋根スラブ厚	mm
x_7	bc	柱幅	mm

変数	変数名	意味	単位
x_8	hc	柱奥行き	mm
x_9	tw_ext	外壁厚	mm
x_{10}	wall_tilt_angle	壁傾斜角度	度
x_{11}	window_ratio_2f	2階窓面積比	-
x_{12}	roof_morph	屋根形状係数	-
x_{13}	roof_shift	屋根ずれ係数	-
x_{14}	balcony_depth	バルコニー奥行き	m

- mm 単位の形状変数は評価直前に整数へ丸める(四捨五入)
- 各変数の定義域 (下限, 上限) は pso_config.py で指定する

```
pso_config.py 14 # =====
15 # 設計変数の範囲定義
16 # =====
17 variable_ranges = {
18     "Lx": (8, 12), # 建物幅 [m]
19     "Ly": (6, 12), # 建物奥行 [m]
20     "H1": (2.6, 3.5), # 1階高さ [m]
21     "H2": (2.6, 3.2), # 2階高さ [m]
```

設計変数 x の扱い: 材質パラメータ

変数番号	部材	材料
x_{15}	柱	0 or 1
x_{16}	1階床	0 or 1
x_{17}	2階床	0 or 1
x_{18}	屋根	0 or 1
x_{19}	壁	0 or 1
x_{20}	バルコニー	0 or 1

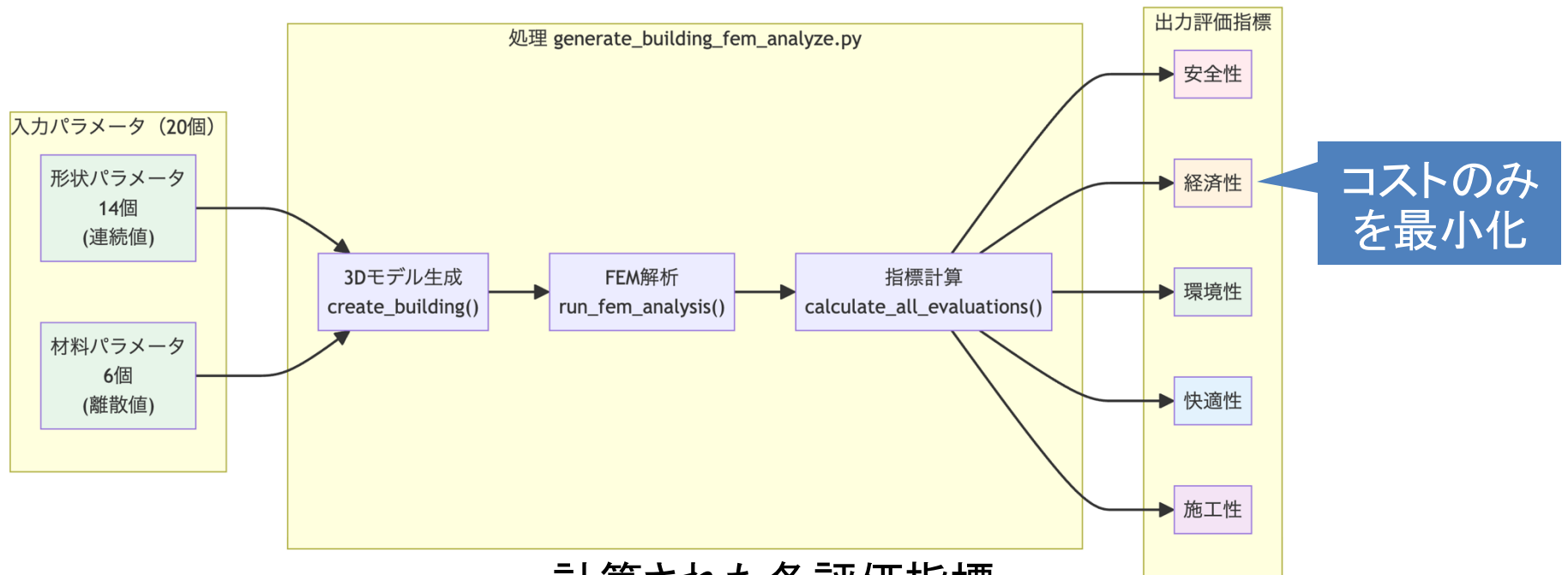
0 = 木材
1 = コンクリート

- 材質パラメータ $x_{15} \sim x_{20}$: 連続値 $[0, 1]$ として保持し, 評価時に

$$m_i = \begin{cases} 0 & (x_i < 0.5) \\ 1 & (x_i \geq 0.5) \end{cases} \quad (i = 15, \dots, 20)$$

と離散化してFEMに渡す (0=木材, 1=コンクリート).

適応度関数の意味(1/2)



pso_config.py で
指定する

計算された各評価指標

コスト

安全性

環境性

快適性

施工性

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

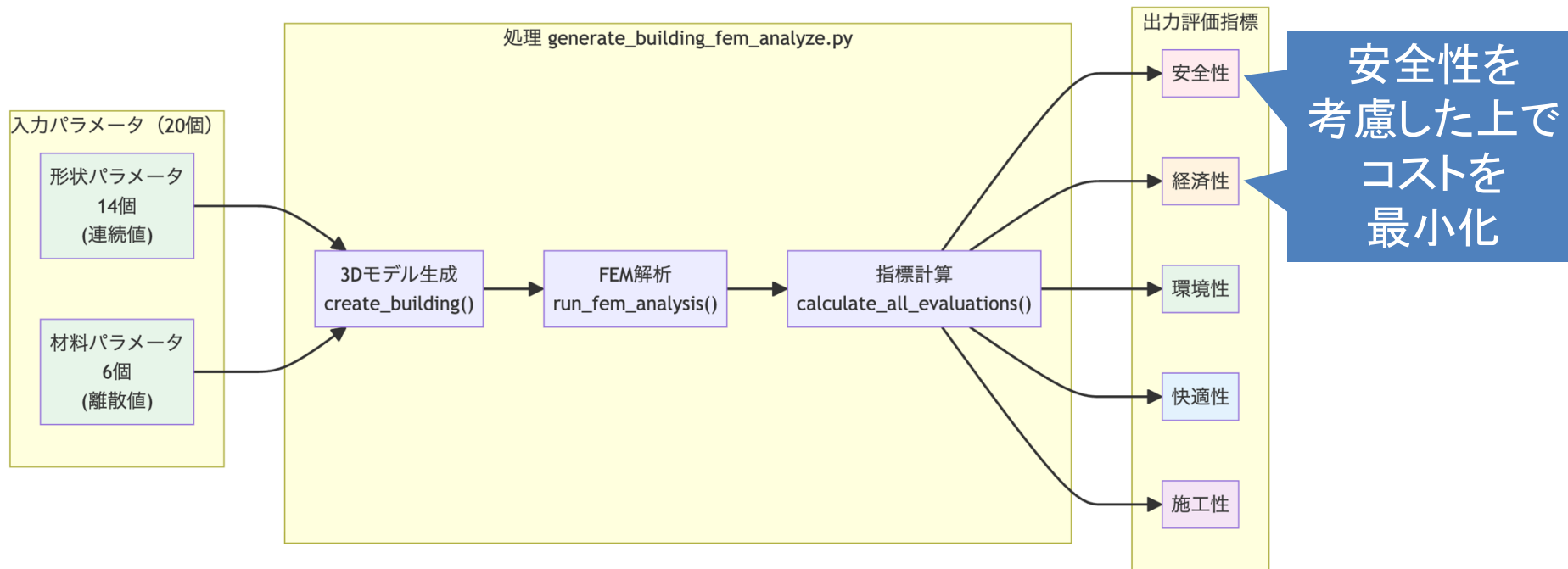
```
# 基本適応度：コストのみ
fitness = cost
```

```
return fitness
```

コストが小さいほど良い設計
⇒ コストがそのまま粒子の適応度(良さ)

他の評価指標は(計算したが)無視

適応度関数の意味(2/2)



```
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) * 10000
```

```
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

+= は「加算して代入」
x += 3 は
x = x + 3 と同じ意味

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) * 10000  
  
    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 **10000** を掛けた値を fitness に加算（コストに上乗せ）する

代入

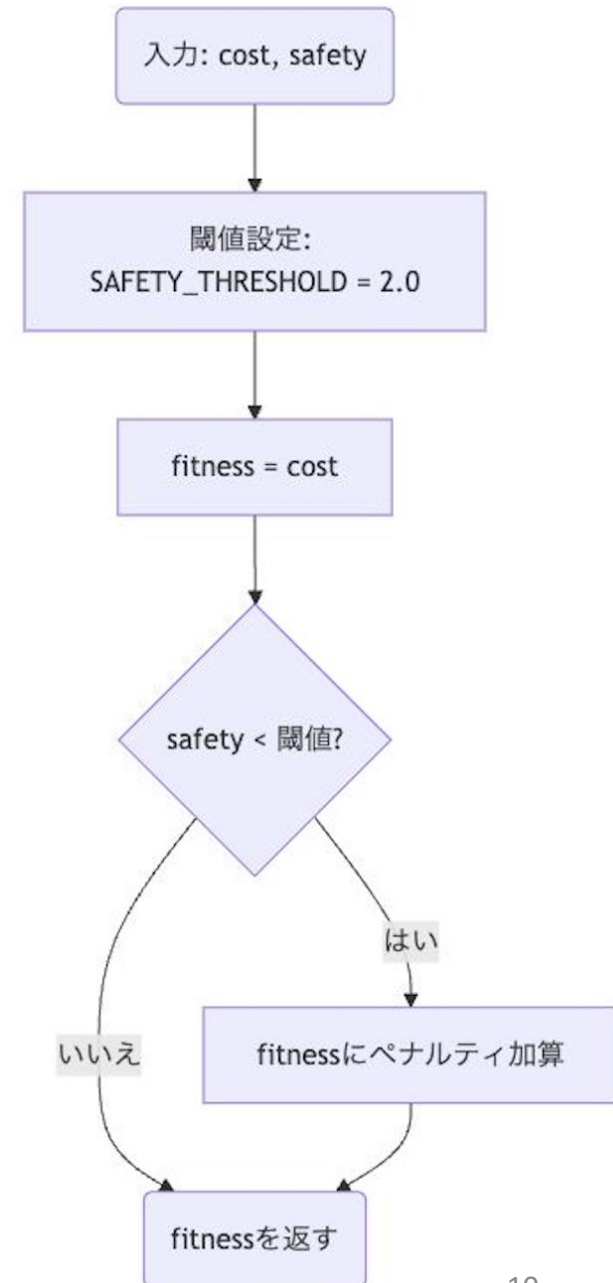
$$\text{fitness} = \text{fitness} + \text{不足分} \times \text{重み}$$

適応度:

「コスト」と「安全率」を1つの評価指標に合成

自由に設定可能

フローチャート



適応度関数の切り替え (1/2)

まとめてコメント化

[Notepad++の設定](#)

```
46 def calculate_fitness(cost, safety, co2, comfort, constructability):  
47  
48     # 基本適応度：コストのみ  
49     fitness = cost  
50  
51     return fitness  
52
```



```
46 def calculate_fitness(cost, safety, co2, comfort, constructability):  
47  
48     # 基本適応度：コストのみ  
49     fitness = cost  
50  
51     return fitness  
52
```

マウスでこれらの行範囲
指定して, Ctrl + K



```
46 # def calculate_fitness(cost, safety, co2, comfort, constructability):  
47  
48     ## 基本適応度：コストのみ  
49     # fitness = cost  
50  
51     # return fitness  
52
```

まとめてコメント化される
(この部分は無視される)

適応度関数の切り替え (2/2)

まとめてコメント化を解除

```
54 | # def calculate_fitness(cost, safety, co2, comfort, constructability):
55 |
56 |     # # 安全率の閾値
57 |     # SAFETY_THRESHOLD = 2.0
58 |
59 |     # # 基本適応度：コストのみ
60 |     # fitness = cost
61 |
62 |     # # 安全率ペナルティ
63 |     # if safety < SAFETY_THRESHOLD:
64 |     |     # fitness += (SAFETY_THRESHOLD - safety) * 100000
65 |
66 |     # return fitness
67 |
```



```
54 | # def calculate_fitness(cost, safety, co2, comfort, constructability):
55 |
56 |     # # 安全率の閾値
57 |     # SAFETY_THRESHOLD = 2.0
58 |
59 |     # # 基本適応度：コストのみ
60 |     # fitness = cost
61 |
62 |     # # 安全率ペナルティ
63 |     # if safety < SAFETY_THRESHOLD:
64 |     |     # fitness += (SAFETY_THRESHOLD - safety) * 100000
65 |
66 |     # return fitness
67 |
```

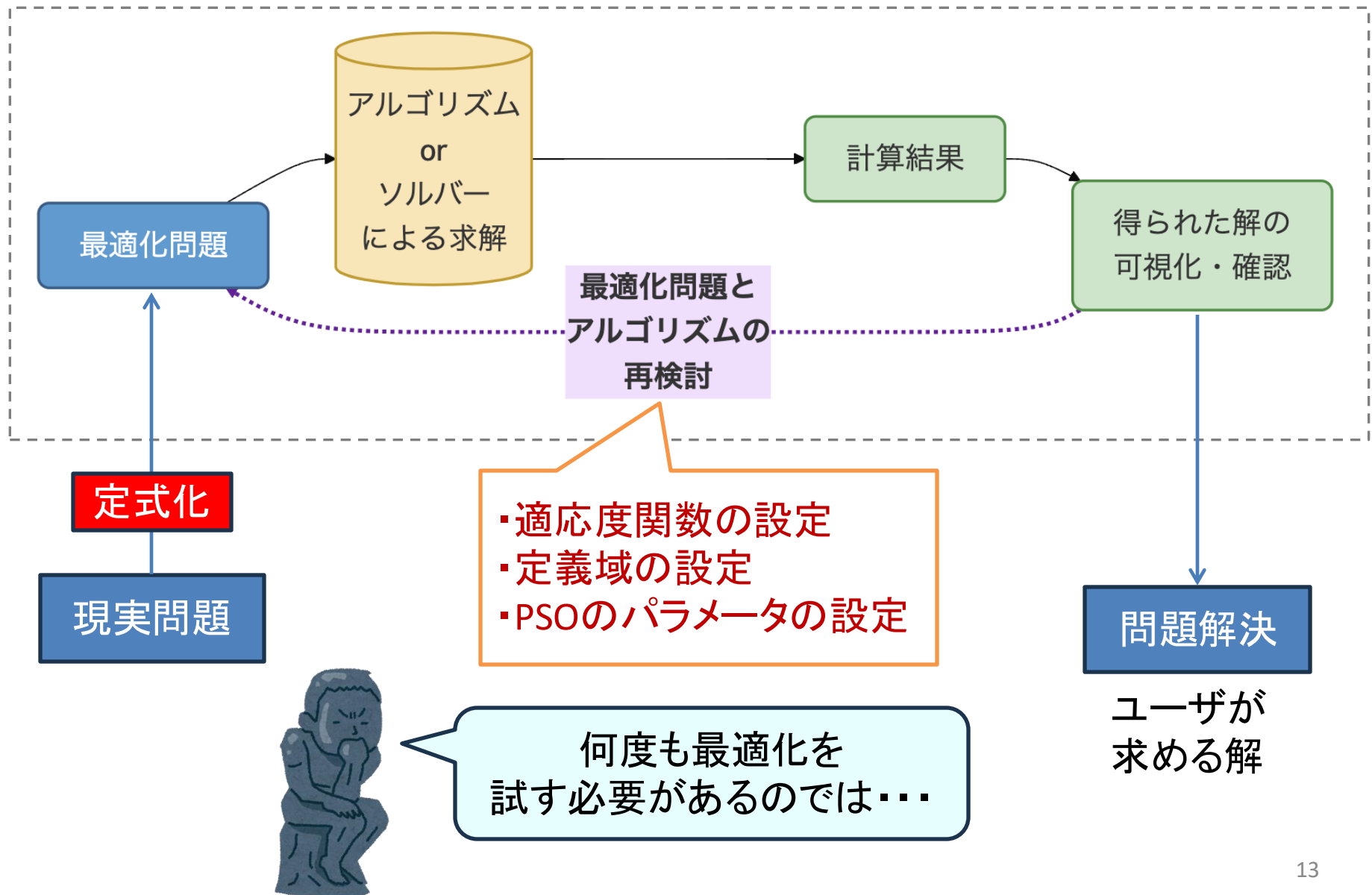
マウスで範囲指定して,
Ctrl + Shift + K



```
54 | def calculate_fitness(cost, safety, co2, comfort, constructability):
55 |
56 |     # 安全率の閾値
57 |     SAFETY_THRESHOLD = 2.0
58 |
59 |     # 基本適応度：コストのみ
60 |     fitness = cost
61 |
62 |     # 安全率ペナルティ
63 |     if safety < SAFETY_THRESHOLD:
64 |         fitness += (SAFETY_THRESHOLD - safety) * 100000
65 |
66 |     return fitness
67 |
```

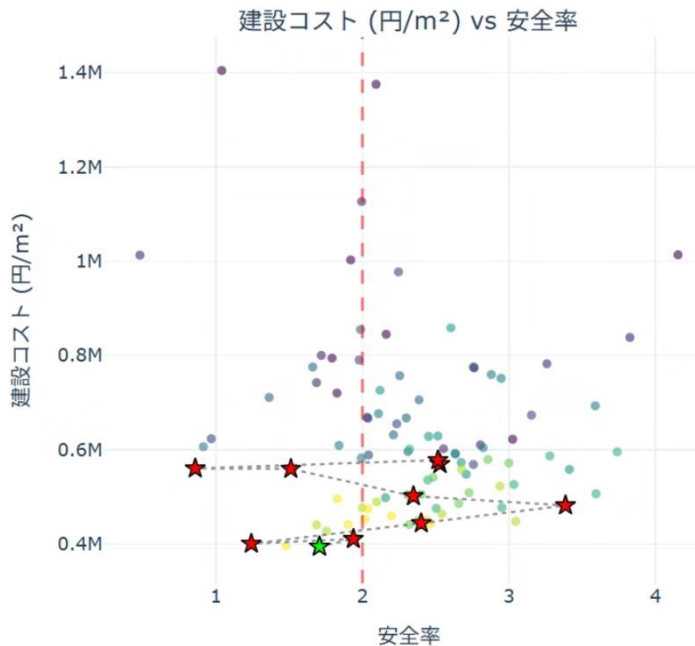
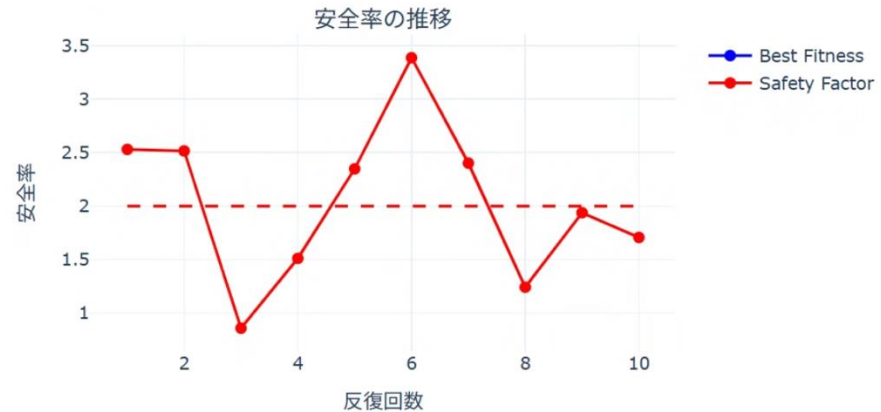
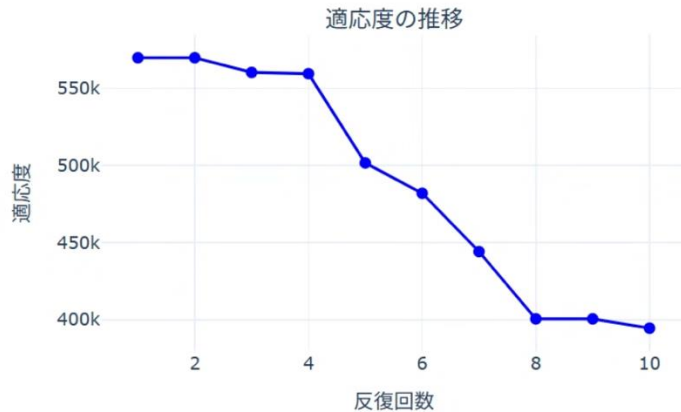
まとめてコメント化が
解除される
(コードとして認識される)

最適化の流れ



PSOの実行例

各世代のgbestの適応度と安全率



適応度関数を

基本適応度：コストのみ
fitness = cost

とした場合の結果

この設定だと
コストは下がるけど、
安全性が...



参考となる文献

日本建築学会・情報システム技術委員会
第37回情報・システム・利用・技術シンポジウム 2014

修正 PSO によるトラス構造物の最小重量設計

○菅谷 明誉^{*1}
曾我部 博之^{*2}

キーワード：粒子群最適化 高次元問題 多峰性関数 構造最適化 最小重量設計

<http://news-sv.aij.or.jp/jyoho/s1/proceedings/2014/pdf/H49.pdf>

- **対象構造物**: 某体育館の屋根トラス
- **モデル化**: 対称性と境界条件を考慮した立体トラス(設計変数324本の部材断面積)
- **目的関数**: 総重量の最小化
- **制約条件**: 引張・圧縮の許容応力度を満たすこと(鋼材SS400を想定)
- **評価方法**: 目的関数に制約違反部材数に比例したペナルティを加算