# 最適化プログラム解説

T. Watanabe

# Index

# [0]ルーチン使用方法

**[0].1ルーチン使用例**

**import sgsearch as sgs としてそのメンバ関数pascalを使って**
**①極値を見出したい関数をfunc**
**(望小設定のみですので最大値を求める場合は正負を反転してください)**
**②上下限をupperx,lowerx(パラメータ数分のnumpyの配列,16変数まで対応)**
**③追跡したい峰候補の数をCands**
**で渡せば峰候補の関数値とその際のパラメータ値をsgsのメンバ変数**
**min_value[ ]、min_position[ ]から取り出せます。**

**使用例**

```
import sgsearch as sgs
import numpy as np
import sgsearch as sgs
import math as mathf
def objectFunc(x):
    x=x.reshape(-1,1)
    A = x[0] :  B = x[1] : C = x[2] ; D = x[3]
    return ( -mathf.exp(-1.*((A+1.)**2 + (B-1.)**2 + (C+1.)**2 + (D-1.)**2))
          - mathf.exp(-1.*((A-1.)**2 + (B+1.)**2 + (C-1.)**2 + (D+1.)**2)) )
```

目的関数

```
sgs.pascal(func=objectFunc,lowerx=np.array([-2., -2., -2., -2.]),
          upperx=np.array([2., 2., 2., 2.]),Cands=2,pattern="cross")
print(sgs.min_value[0]);print(sgs.max_position[0])
print(sgs.min_value[1]);print(sgs.max_position[1])
```

目的関数と
上下限等を渡して
峰探索、結果取出

**objecctFuncは(1,-1,1,-1)、(-1,1,-1,1)近辺で極小値-1、-1を取る関数で、実際それに近い結果が返ってくる**

# [0]ルーチン使用方法

[0].1最適化設定
pascal(func,upperx,lowerx,depth=40,linearmode_start=21,
pattern="all",Cands=1)
オプション設定の内容(上記はデフォルト値)
- func：最適化したい関数
- depth:パターンファイル使用、深さ
- linearmode_start:
  パターンファイルによる検索から線形性を仮定した検索に切替
  (変動幅が小さくなり線形性が高くなれば、こちらの方が少ない試行数で済む)
- pattern：パターンファイル選択(下記の6つから選択可能)
  - seed (離散sin関数状,最大10試行,2項目網羅)
  - cross1 (cross演算1回,最大22試行,3項目網羅)
  - multi1(multi演算1回,最大58試行,3項目網羅)
  - cross1_multi1(cross演算1回、multi演算1回、最大222試行,4項目網羅)
  - cross1_multi1_cross1(cross演算1回、multi演算1回、cross演算1回、最大2016試行)
  - cross1_multi2(cross演算1回、multi演算2回、最大7584試行)
  - all(2のパラメータ数乗全て、最大65536試行)
- Cands：追跡する最適ノード候補の数

# [1]パターンファイル読込、成形

**[1].1パターンファイル書式**
**0,1を16個並べた文字列データを1行に1つだけ配置**
**seedは2項網羅cross1はseedに演算①を1回適用,multi1は演算②を1回適用**
**allは0000000000000000から1111111111111111まで全パターン**
**[1].2データ読込**
**all、その他のパターンファイルとも頭から要素数分(以降N)の文字列を切り取り**
**重複分をマージしたあとnumpyの要素数Nの整数型変数として保持**

All

| パターン<br>(3項目網羅で先頭が0の16個) |
| --- |
| 0000000000000000<br>1000000000000000<br>0100000000000000<br>1100000000000000<br>1010000000000000<br>.<br>.<br>.<br>1111111111111111 |

cross1(3項目網羅)

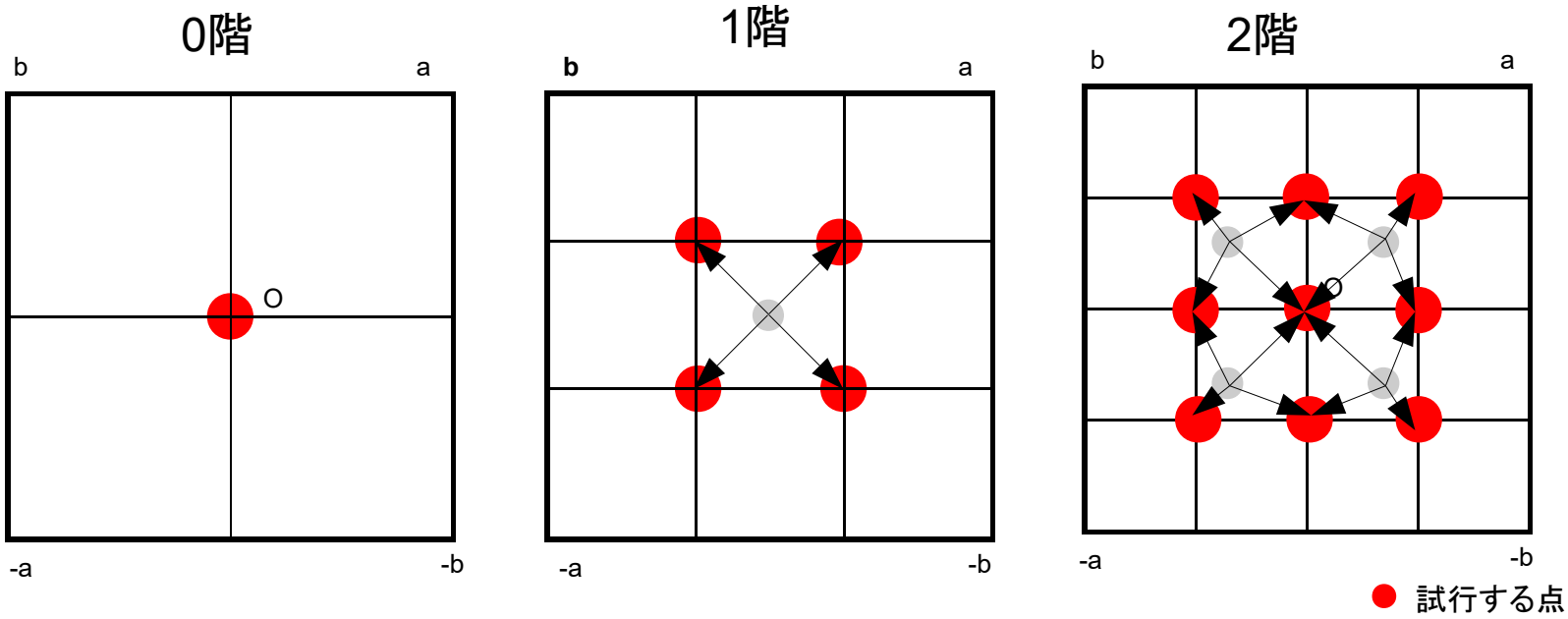
| パターン<br>(3項目網羅で先頭が0の16個) |
| --- |
| 0000000000000000<br>1111111100000000<br>1111000011110000<br>0000111111110000<br>1100110011001100<br>.<br>.<br>.<br>1111111111111111 |

(All以外は
切り取った単位で
重複が発生するので
マージする)

# [2]パターンデータと木を用いた検索

**[2].パスカル木**
**上下限値を探索階数で等間隔に分けた地点(以下ノードと呼ぶ)を**
**パターンデータの0,1に基づいて増減移動**

2次元でのイメージ図

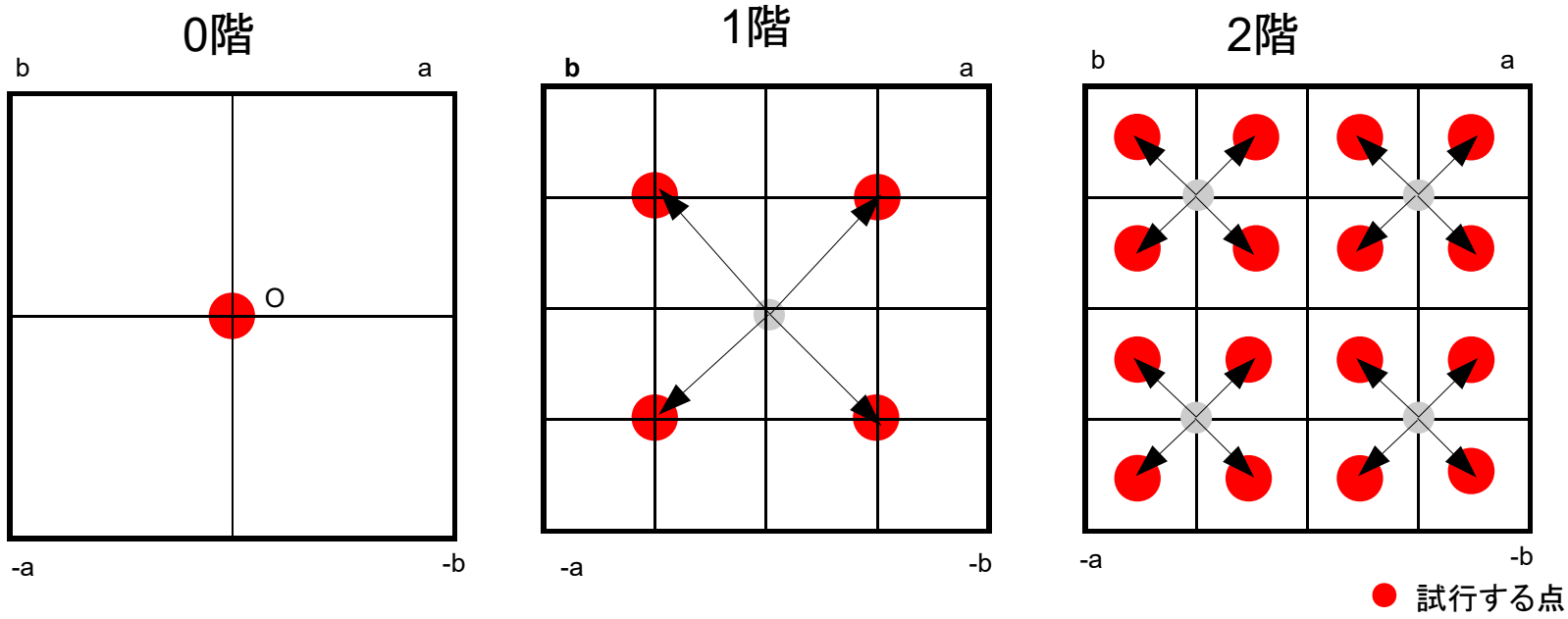



いくつかのテスト関数でうまく最適パラメータに移動することを確認

# [2]パターンデータと木を用いた検索

**[2].2分木のノード群上をパターンデータの0,1に基づいて移動**
**前の階で移動した半分の長さを1/2倍したベクトル分**
**移動した点で再度試行を行う。これを繰り返す。**
**(今後繰り返し回数の事を[階]とよぶ。)**

2次元でのイメージ図

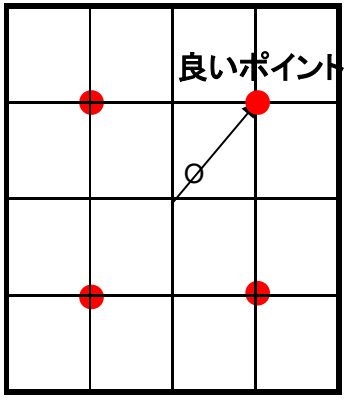



汎用性が低いので後回し。(前回のはモーフィング用に作成の8変数固定バージョン)

# [3]線形性仮定での検索

## [3].1 線形仮定による次階試行ベクトル選択

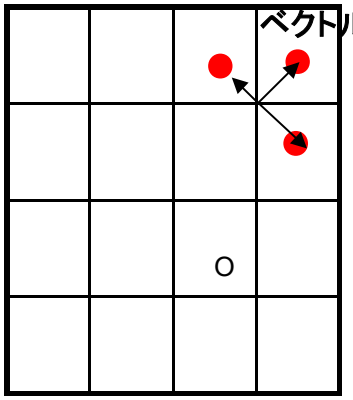
N階　(1,1,-1,-1,1,1,-1,-1)




● 試行した点

N階でスコアが上がった
ベクトル方向について
目的関数が線形に近ければ
同じ方向でまたスコアが上が
ると推測(2匹目のドジョウ?)
それでN+1階ではそのベクト
ルと各因子1つずつずらした
ベクトルを試行

N+1階




● 試行する点

N階のベストが
(1,1,-1,-1,1,1,-1,-1)　なら

(1,1,-1,-1,1,1,-1,1)
(1,1,-1,-1,1,1,1,-1)
(1,1,-1,-1,1,-1,-1,-1)
(1,1,-1,-1,-1,1,-1,-1)
(1,1,-1,1,1,1,-1,-1)
(1,1,1,-1,1,1,-1,-1)
(1,-1,-1,-1,1,1,-1,-1)
(-1,1,-1,-1,1,1,-1,-1)を試行

**前回モーフィングでうまく機能したがテスト関数でもうまく機能することを確認**

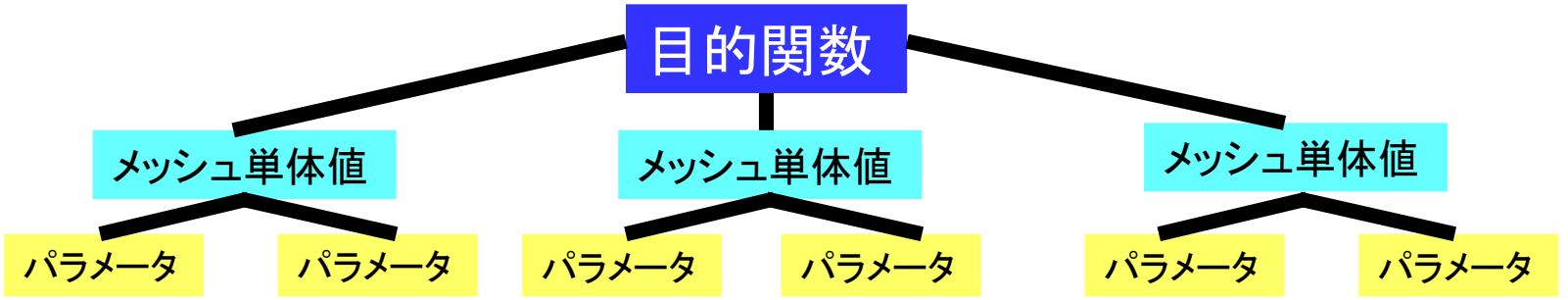
# [4]今後の方針

## [4].2　今後の方針

・Freecad or Openscadモデルのパラメータとリンクさせstl作成から一貫して通すこともできる形にしたい。

・自動的にいくつかのメッシュ単体をサンプル点、中間ノードとして
(ニューラルネットワーク、ベイジリアンネットワークといったライブラリが適用できそう)

その物理値によりパラメータを分離評価できるようにしたい。
(分離評価により少数の施行だけでパラメータをどちらに振るか判別可能)



# [2]網羅パターンの生成方法

**[2].2 n項目中3項目の組み合わせの網羅パターンの生成方法①**
**2項目網羅パターンから任意の2つを選びそれぞれの桁に下記表の演算を施したパターンを作成する。**

演算①

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

**証明**
**2項目網羅である3項目間に(0,0,1)という組み合わせが形成できなかったとする。**
**この場合2項目は網羅していることから(0,0,1)から1項目だけ異なる**
**(1,0,1)(0,1,1)が存在することになり**
**それでこれらに演算①が施されるとされるとそれが(0,0,1)となる。**
**2項目網羅である3項目間に(0,1,1)という組み合わせが形成できなかった場合は対称性より(1,0,0)が存在しないことになり上記と同様の議論と演算①で(0,1,1)が形成される。**
**この演算で3項目の網羅が確保されることがわかる。**

# [2]網羅パターンの生成方法

**[2].3 n項目中3項目の組み合わせの網羅パターンの生成方法②**
**2項目網羅パターンから任意の2つを選びそれぞれの桁に下記の演算②、③を施したパターンを作成する。**

演算②

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

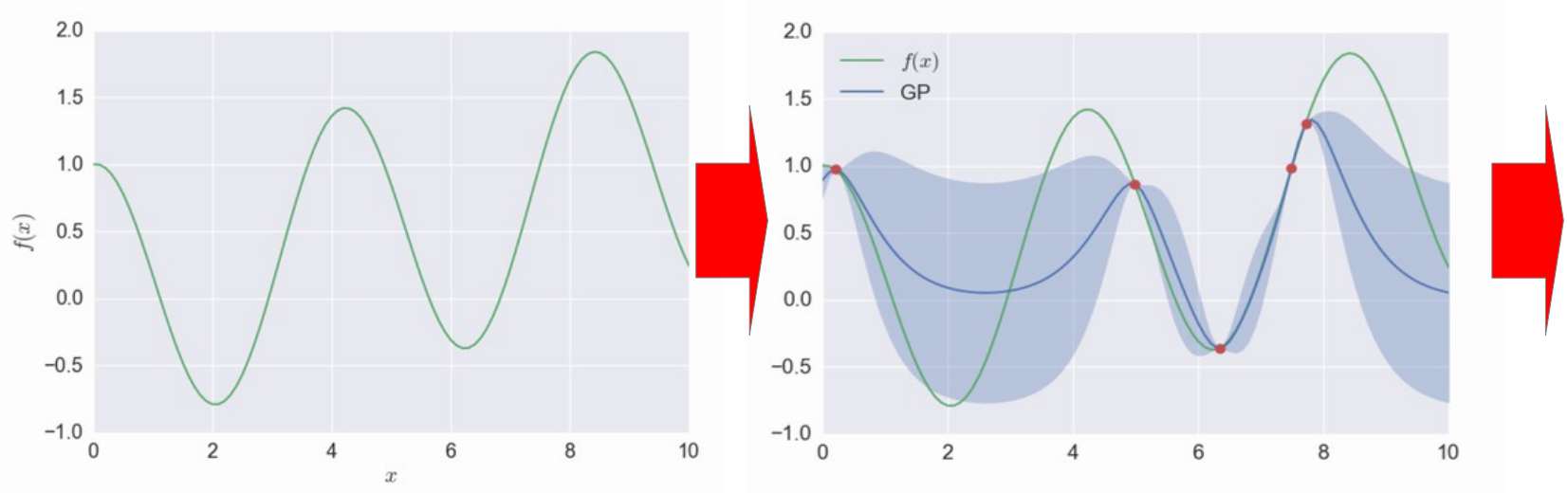
演算③

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**証明**
**2項目網羅である3項目間に(0,0,1)という組み合わせが形成できなかったとする。**
**この場合2項目は網羅していることから(0,1,1),(1,0,1)が存在することになる。**
**それで(0,1,1) (1,0,1) に演算③を行うとそれが(0,0,1)が生成される。**
**2項目網羅である3項目間に(0,1,1)という組み合わせが形成できなかった場合は上記と同様の議論と演算②で(0,1,1)が形成される。**
**この演算で3項目の網羅が確保されることがわかる。**

# [1]比較対象

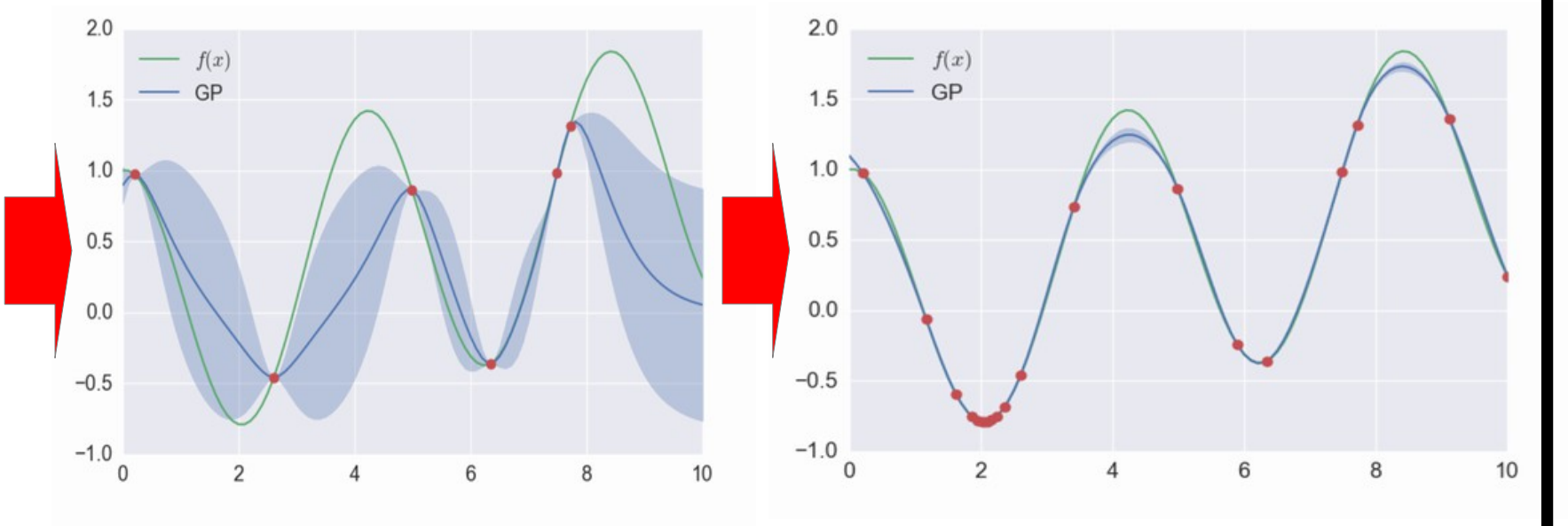
**[1].1 ベイズ最適化の理論概要**
**(https://qiita.com/marshi/items/51b82a7b990d51bd98cdより引用)**




**①未知の関数に対し最初ランダムに試行(赤点)する。**
**②①を通る滑らかな曲線(青線。複数選択肢がある模様、)と**
**ガウス過程を想定した関数枠(青領域)を見出す。**

# [1]比較対象

## [1].1 ベイズ最適化の理論概要



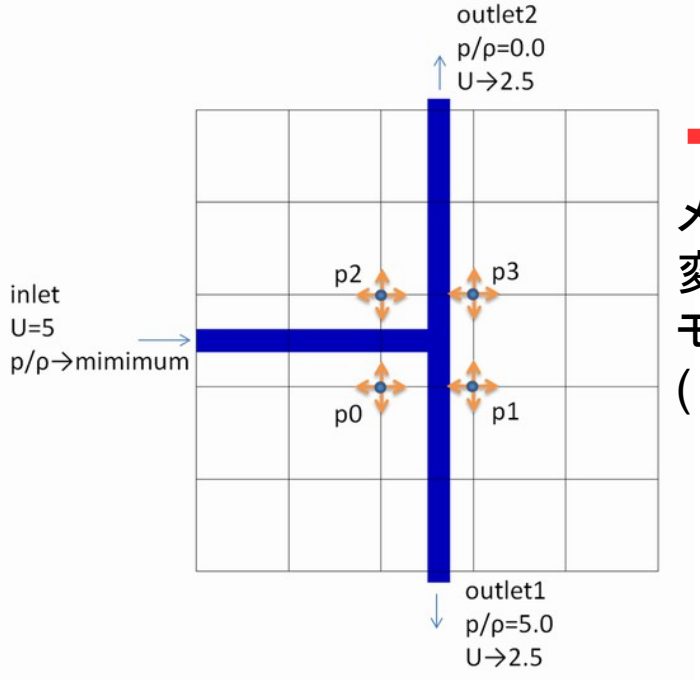

③関数枠を参考に極小値を取りそうと思われる点を試行する。
④①～③により最適値を見出す。
(機械学習のパラメータ最適化(4～10因子あたり)ではかなりメジャーな様子)

# [1]比較対象

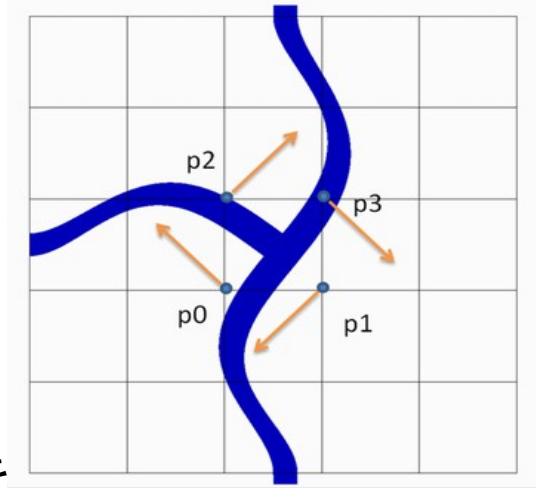
**[1].2 ベイズ最適化の例(OPENCAE関西片山様作成 T字管モデル)**
**https://qiita.com/TatsuyaKatayama/items/afbc6667ca66775d6937より引用**

T字菅モデル(OpenFoamのTutorialより作成)

outlet1,2の流速を一定とし、最小のinlet圧力となる形状を求める



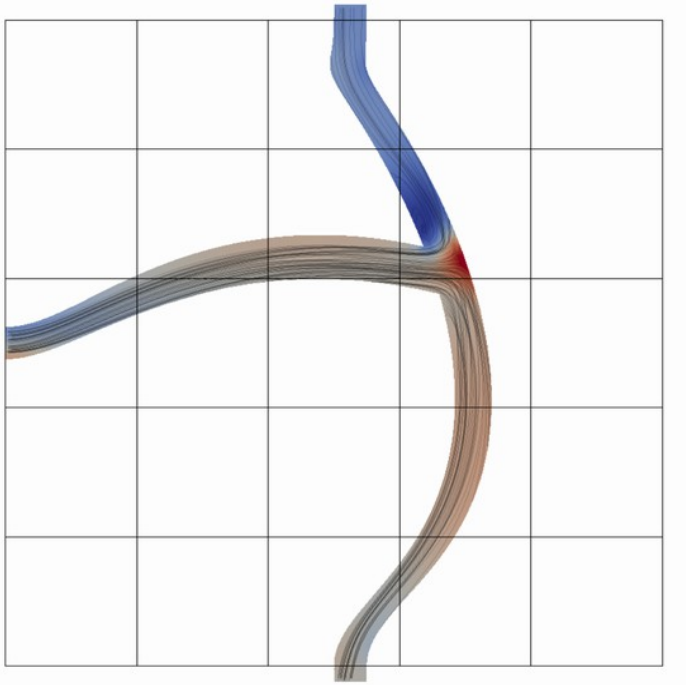

メッシュ作成後p0～p3に
変動をかけて
モーフィング形状を作成
( 右例では
p0:(-1,1)、p1(-1,-1)
p2:(1,1)、p3(1,-1))
以後p0～p3のx,y座標を
パラメータとしますが
その幅は-1～1です。



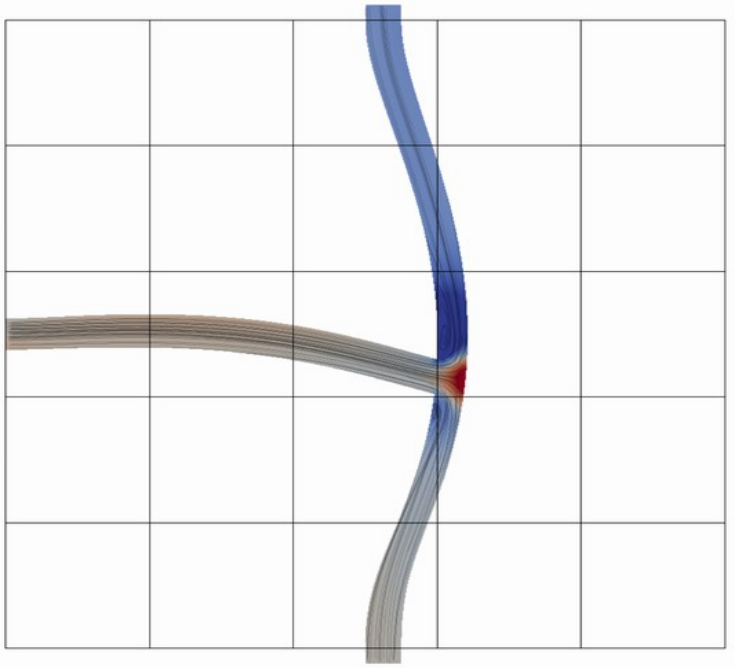
# [1]比較対象

**[1].2 前ページのモデルに(計250試行)**

峰①(スコア4.167)



峰②(スコア5.895)



**上記2つがスコアの良い峰近傍と目される結果となった**

# [2]自分の手法の適用結果

**[2].1 3項目網羅①2階分試行(計28試行)**

1階

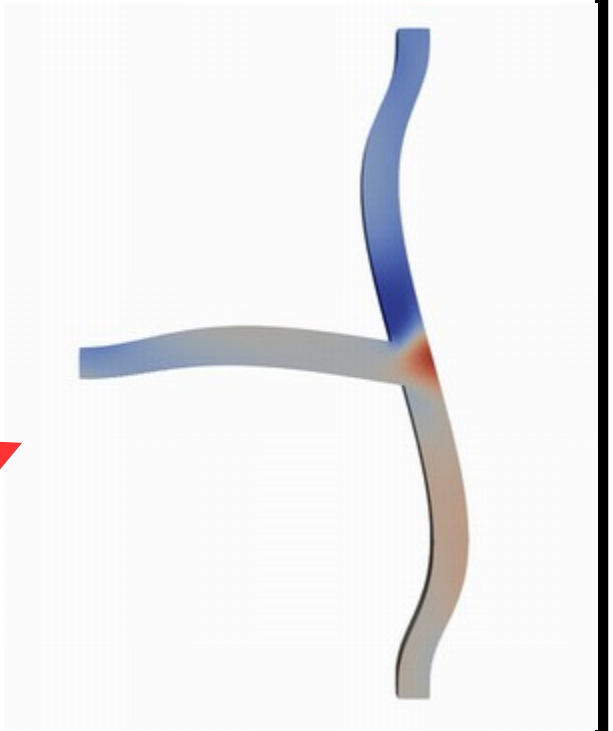
| | |
|---|---|
| 00000000 | 10000.0 |
| 11111111 | 71.70794359999992 |
| 00001111 | 702.3295141000003 |
| 11110000 | 1669.5241276000002 |
| 00110011 | 1674.9678220999995 |
| 11001100 | 691.4603001 |
| 01010101 | 10000.0 |
| 10101010 | 206.11907040000017 |
| 00111100 | 244.93639039999994 |
| 11000011 | 10000.0 |
| 01011010 | 4128.285608099998 |
| 10100101 | 12.810175599999996 |
| 01100110 | 34.45323809999995 |
| 10011001 | 10000.0 |

2階

| | |
|---|---|
| 00000000 | 10000.0 |
| 11111111 | 10000.0 |
| 00001111 | 10000.0 |
| 11110000 | 10000.0 |
| 00110011 | 173.85340939999986 |
| 11001100 | 102.9531384000001 |
| 01010101 | 6.060708099999989 |
| 10101010 | 10000.0 |
| 00111100 | 10.6287304 |
| 11000011 | 15.830222500000017 |
| 01011010 | 57.546999999999926 |
| 10100101 | 10000.0 |
| 01100110 | 3.529013600000003 |
| 10011001 | 10000.0 |

**0はマイナス方向移動(1階なら-0.5、2階なら-0.25)**
**1はプラス方向移動(1階なら0.5、2階なら0.25)**
**左から順にp0～p3のx,yに対応**

峰①近辺？(スコア3.529)



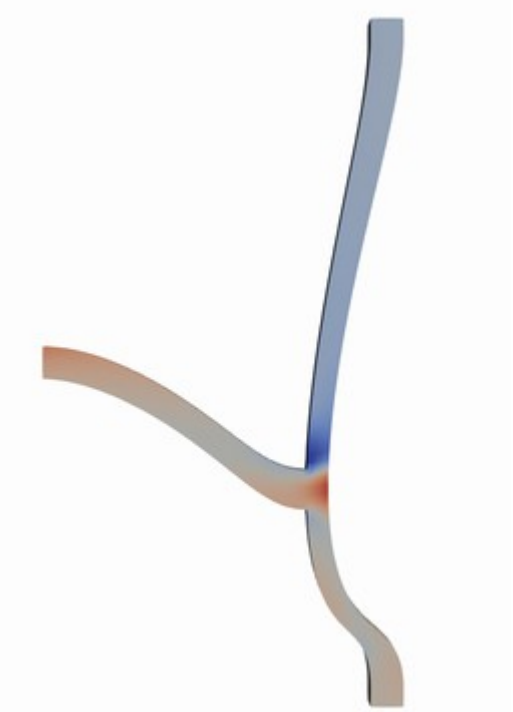
**この段階で**
**ベイズ最適化よりも**
**良いスコアを発見できた。**

# [2]自分の手法の適用結果

## [2].2 3項目網羅②2階試行(計64試行)

| | |
|---|---|
| 00000000 | 10000.0 |
| 11111111 | 71.70794359999 |
| 00001111 | 702.3295141000 |
| 11110000 | 1669.524127600 |
| 00110011 | 1674.967822099 |
| 11001100 | 691.4603001 |
| 01010101 | 10000.0 |
| 10101010 | 206.1190704000 |
| 00000011 | 10000.0 |
| 11111100 | 454.9172400000 |
| 00000101 | 232.1282025000 |
| 11111010 | 141.1647499999 |
| 00001010 | 10000.0 |
| 11110101 | 77.91781360000 |
| 00001100 | 67.84238960000 |
| 11110011 | 160.6131360999 |
| 00010001 | 10000.0 |
| 11101110 | 71.6841581 |
| 00100010 | 242.0003300999 |
| 11011101 | 10000.0 |
| 00110000 | 97.92050999999 |
| 11001111 | 10000.0 |
| 01000100 | 10000.0 |
| 10111011 | 10000.0 |
| 01010000 | 723.3837999999 |
| 10101111 | 10000.0 |
| 10001000 | 11.43855160000 |
| 01110111 | 1111.576506899 |
| 10100000 | 10000.0 |
| 01011111 | 2181.0189225 |
| 11000000 | 10000.0 |
| 00111111 | 1056.207059600 |

| | |
|---|---|
| 00000000 | 138.01805639999998 |
| 11111111 | 31.957057599999978 |
| 00001111 | 59.16927759999993 |
| 11110000 | 279.5568025 |
| 00110011 | 391.08573889999985 |
| 11001100 | 686.4683689000001 |
| 01010101 | 22.447246100000008 |
| 10101010 | 198.99212839999984 |
| 00000011 | 203.01991639999997 |
| 11111100 | 232.12197560000007 |
| 00000101 | 9.900260899999987 |
| 11111010 | 24.3255081 |
| 00001010 | 105.39308559999995 |
| 11110101 | 54.88802439999996 |
| 00001100 | 53.98880089999997 |
| 11110011 | 59.74273290000004 |
| 00010001 | 213.5886225000001 |
| 11101110 | 189.04304000000002 |
| 00100010 | 10.267040099999997 |
| 11011101 | 18.82442040000001 |
| 00110000 | 12.76818810000001 |
| 11001111 | 30.73052959999998 |
| 01000100 | 253.67555610000016 |
| 10111011 | 329.1617696 |
| 01010000 | 19.22707240000002 |
| 10101111 | 17.20978999999997 |
| 10001000 | 158.15429960000014 |
| 01110111 | 85.42567960000014 |
| 10100000 | 308.40492439999997 |
| 01011111 | 228.3557601 |
| 11000000 | 652.0665600000002 |
| 00111111 | 253.23046999999985 |



こちらでは良い値を見つけられなかった。
(最高でスコア9.900)
その他ポイントも含め探った感じでは
このモデルは非線形性が強く
1階は全て(2^8)実施するくらいの難物

# [2]自分の手法の適用結果

## [2].3 1階全パターン試行(計256試行)

| | |
|---|---|
| 00000000 | 10000.0 |
| 00000001 | 313.0818116000 |
| 00000010 | 810.3402369000 |
| 00000011 | 10000.0 |
| 00000100 | 10000.0 |
| 00000101 | 232.1282025000 |
| 00000110 | 378.1807836000 |
| 00000111 | 694.6644826 |
| 00001000 | 365.9504721000 |
| 00001001 | 858.5475108999 |
| 00001010 | 10000.0 |
| 00001011 | 1517.7844129 |
| 00001100 | 67.84238960000 |
| 00001101 | 10000.0 |
| 00001110 | 399.6833683999 |
| 00001111 | 702.3295141000 |
| 00010000 | 3.253550137599 |
| 00010001 | 10000.0 |
| 00010010 | 2677.6778969 |
| 00010011 | 3397.017915600 |
| 00010100 | 10000.0 |
| 00010101 | 2.3450634496e+ |
| 00010110 | 1060.952222499 |
| 00010111 | 1680.6417441 |
| 00011000 | 2298.033264100 |
| 00011001 | 3730.293834399 |
| 00011010 | 3454.4879744 |
| 00011011 | 4544.674433600 |
| 00011100 | 10000.0 |
| 00011101 | 10000.0 |
| 00011110 | 1357.398776100 |
| 00011111 | 1818.618209999 |
| 00100000 | 10000.0 |
| 00100001 | 10000.0 |
| 00100010 | 242.0003300999 |
| 00100011 | 664.4092943999 |
| 00100100 | 4.2883029 |
| 00100101 | 107.3067613999 |
| 00100110 | 137.8523856000 |
| 00100111 | 434.2638465000 |
| 00101000 | 88.62023009999 |
| 00101001 | 10000.0 |
| 00101010 | 10000.0 |
| 00101011 | 924.0113236000 |
| 00101100 | 12.07622000000 |
| 00101101 | 246.4591380999 |
| 00101110 | 10000.0 |
| 00101111 | 438.8245520999 |
| 00110000 | 97.92050999999 |
| 00110001 | 10000.0 |
| 00110010 | 1047.812252899 |
| 00110011 | 1674.967822099 |
| 00110100 | 37.35907359999 |
| 00110101 | 282.8247335999 |
| 00110110 | 471.0549129000 |
| 00110111 | 959.3414995999 |
| 00111000 | 10000.0 |
| 00111001 | 10000.0 |
| 00111010 | 10000.0 |
| 00111011 | 2490.549660099 |
| 00111100 | 244.9363903999 |
| 00111101 | 10000.0 |
| 00111110 | 774.2504736 |
| 00111111 | 1056.207059600 |
| 01000000 | 9.779331599999 |
| 01000001 | 10000.0 |
| 01000010 | 614.1269829 |
| 01000011 | 1097.7946164 |
| 01000100 | 10000.0 |
| 01000101 | 194.5347025000 |
| 01000110 | 307.0913400999 |
| 01000111 | 813.5157409000 |
| 01001000 | 120.5605603999 |
| 01001001 | 10000.0 |
| 01001010 | 1053.034068900 |
| 01001011 | 1445.85584 |
| 01001100 | 6.408672500000 |
| 01001101 | 10000.0 |
| 01001110 | 391.6087361000 |
| 01001111 | 740.5087341000 |
| 01010000 | 723.3837999999 |
| 01010001 | 10000.0 |
| 01010010 | 3601.004980100 |
| 01010011 | 4329.985920399 |
| 01010100 | 10000.0 |
| 01010101 | 10000.0 |
| 01010110 | 1608.289776099 |
| 01010111 | 2423.113282399 |
| 01011000 | 2552.766162499 |
| 01011001 | 10000.0 |
| 01011010 | 4128.285608099 |
| 01011011 | 5176.347567600 |
| 01011100 | 10000.0 |
| 01011101 | 10000.0 |
| 01011110 | 1766.322122500 |
| 01011111 | 2181.0189225 |
| 01100000 | 462.3147899999 |
| 01100001 | 39.85662810000 |
| 01100010 | 57.73968040000 |
| 01100011 | 478.4560115999 |
| 01100100 | 339.5487023999 |
| 01100101 | 14.69106639999 |
| 01100110 | 34.45323809999 |
| 01100111 | 396.8358800000 |
| 01101000 | 14.55096760000 |
| 01101001 | 256.3353788999 |
| 01101010 | 447.9184775999 |
| 01101011 | 793.8059604000 |
| 01101100 | 10000.0 |
| 01101101 | 121.6278956000 |
| 01101110 | 155.4712235999 |
| 01101111 | 369.5795203999 |
| 01110000 | 10000.0 |
| 01110001 | 289.1946001000 |
| 01110010 | 590.1251415999 |
| 01110011 | 1674.2596921 |
| 01110100 | 168.0399600000 |
| 01110101 | 50.40203560000 |
| 01110110 | 290.3259000999 |
| 01110111 | 1111.576506899 |
| 01111000 | 347.4697881000 |
| 01111001 | 1462.178640899 |
| 01111010 | 10000.0 |
| 01111011 | 2625.124503599 |
| 01111100 | 42.7387584 |
| 01111101 | 10000.0 |
| 01111110 | 886.7064528999 |
| 01111111 | 1026.574367600 |
| 10000000 | 102.8986908999 |
| 10000001 | 183.0076183999 |
| 10000010 | 106.6268003999 |
| 10000011 | 593.5814775999 |
| 10000100 | 149.3398929000 |
| 10000101 | 10000.0 |
| 10000110 | 10000.0 |
| 10000111 | 10000.0 |
| 10001000 | 11.43855160000 |
| 10001001 | 368.2485228999 |
| 10001010 | 257.9352703999 |
| 10001011 | 795.3616299999 |
| 10001100 | 10000.0 |
| 10001101 | 98.37365239999 |
| 10001110 | 38.66251000000 |
| 10001111 | 10000.0 |
| 10010000 | 10000.0 |
| 10010001 | 10000.0 |
| 10010010 | 10000.0 |
| 10010011 | 10000.0 |
| 10010100 | 6.716979999999 |
| 10010101 | 10000.0 |

# [2]自分の手法の適用結果

## [2].3 1階全パターン試行(計256試行)

| | |
|---|---|
| 10010110 | 10000.0 |
| 10010111 | 840.6357064000 |
| 10011000 | 10000.0 |
| 10011001 | 10000.0 |
| 10011010 | 10000.0 |
| 10011011 | 10000.0 |
| 10011100 | 90.82836160000 |
| 10011101 | 831.8647524999 |
| 10011110 | 10000.0 |
| 10011111 | 10000.0 |
| 10100000 | 10000.0 |
| 10100001 | 66.90415000000 |
| 10100010 | 10000.0 |
| 10100011 | 10000.0 |
| 10100100 | 215.3630899999 |
| 10100101 | 12.81017559999 |
| 10100110 | 10000.0 |
| 10100111 | 179.6975999999 |
| 10101000 | 24.57518609999 |
| 10101001 | 196.0981099999 |
| 10101010 | 206.1190704000 |
| 10101011 | 511.1046255999 |
| 10101100 | 107.4524428999 |
| 10101101 | 47.83564690000 |
| 10101110 | 10000.0 |
| 10101111 | 10000.0 |
| 10110000 | 60.37997839999 |
| 10110001 | 254.9991364000 |
| 10110010 | 68.81615559999 |
| 10110011 | 661.9925649000 |
| 10110100 | 10000.0 |
| 10110101 | 10000.0 |
| 10110110 | 40.59760560000 |
| 10110111 | 422.7906788999 |
| 10111000 | 10000.0 |
| 10111001 | 787.2709943999 |
| 10111010 | 547.1552100000 |
| 10111011 | 10000.0 |
| 10111100 | 8.622316400000 |
| 10111101 | 339.8503556000 |
| 10111110 | 175.7235303999 |
| 10111111 | 10000.0 |
| 11000000 | 10000.0 |
| 11000001 | 10000.0 |
| 11000010 | 10000.0 |
| 11000011 | 10000.0 |
| 11000100 | 10000.0 |
| 11000101 | 10000.0 |
| 11000110 | 54.99638160000 |
| 11000111 | 142.7643180999 |

| | |
|---|---|
| 11001000 | 339.8523120999 |
| 11001001 | 98.65455239999 |
| 11001010 | 95.71815959999 |
| 11001011 | 10000.0 |
| 11001100 | 691.4603001 |
| 11001101 | 8.608312499999 |
| 11001110 | 8.450382400000 |
| 11001111 | 10000.0 |
| 11010000 | 10000.0 |
| 11010001 | 10000.0 |
| 11010010 | 10000.0 |
| 11010011 | 10000.0 |
| 11010100 | 500.9492321 |
| 11010101 | 10000.0 |
| 11010110 | 42.68909160000 |
| 11010111 | 10000.0 |
| 11011000 | 10000.0 |
| 11011001 | 10000.0 |
| 11011010 | 10000.0 |
| 11011011 | 10000.0 |
| 11011100 | 10000.0 |
| 11011101 | 10000.0 |
| 11011110 | 10000.0 |
| 11011111 | 10000.0 |
| 11100000 | 10000.0 |
| 11100001 | 75.59516609999 |
| 11100010 | 306.8939449000 |
| 11100011 | 10000.0 |
| 11100100 | 10000.0 |
| 11100101 | 119.3533584000 |
| 11100110 | 10000.0 |
| 11100111 | 27.68885689999 |
| 11101000 | 653.2265224999 |
| 11101001 | 9.372858899999 |
| 11101010 | 10000.0 |
| 11101011 | 10000.0 |
| 11101100 | 10000.0 |
| 11101101 | 10000.0 |
| 11101110 | 71.6841581 |
| 11101111 | 10000.0 |
| 11110000 | 1669.524127600 |
| 11110001 | 23.1432024 |
| 11110010 | 321.5334560999 |
| 11110011 | 160.6131360999 |
| 11110100 | 1388.9576361 |
| 11110101 | 77.91781360000 |
| 11110110 | 197.1004016 |
| 11110111 | 129.9867584 |
| 11111000 | 325.7030184 |
| 11111001 | 10000.0 |

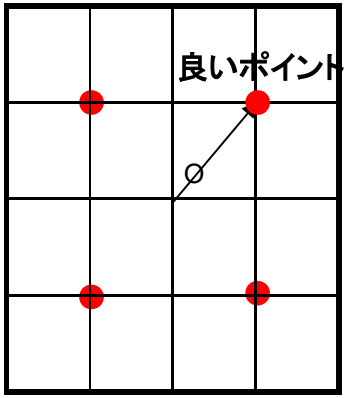
| | |
|---|---|
| 11111010 | 141.1647499999 |
| 11111011 | 792.5301625 |
| 11111100 | 454.9172400000 |
| 11111101 | 10000.0 |
| 11111110 | 10000.0 |
| 11111111 | 71.70794359999 |

**ベストスコアの00100100**
**に対し次ページの線形仮定の**
**探索をかける**

# [2]自分の手法の適用結果

## [2].4 線形仮定による次階試行ベクトル選択

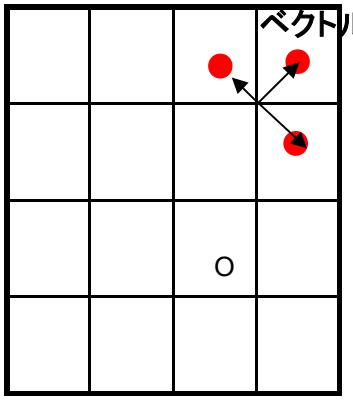
N階　(1,1,-1,-1,1,1,-1,-1)




● 試行した点

N階でスコアが上がった
ベクトル方向について
目的関数が線形に近ければ
同じ方向でまたスコアが上が
ると推測(2匹目のドジョウ?)
それでN+1階ではそのベクト
ルと各因子1つずつずらした
ベクトルを試行

N+1階




● 試行する点

N階のベストが
(1,1,-1,-1,1,1,-1,-1)　なら

(1,1,-1,-1,1,1,-1,1)
(1,1,-1,-1,1,1,1,-1)
(1,1,-1,-1,1,-1,-1,-1)
(1,1,-1,-1,-1,1,-1,-1)
(1,1,-1,1,1,1,-1,-1)
(1,1,1,-1,1,1,-1,-1)
(1,-1,-1,-1,1,1,-1,-1)
(-1,1,-1,-1,1,1,-1,-1)を試行

**階が降りるに従い線形に近づく可能性が高いので前の階のベクトルと同じ向き**
**をまず試行しあとは各因子1つづつ反転させた物を試行する。**

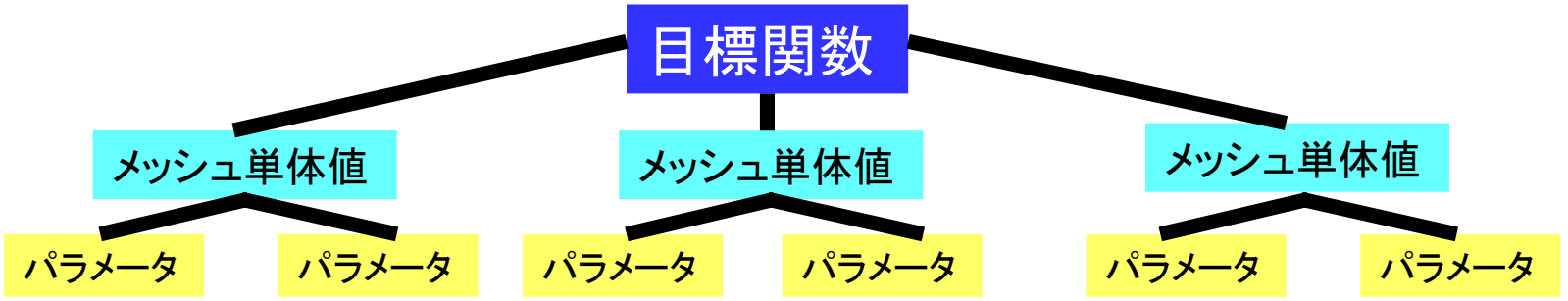
# [5]載せたい手法

## [5].1　さしあたり載せたい機能

・2分木以外にパスカル木選択もできるようにする。
・パラメータ数として8と16まで対応できるようにする。
・パラメータを2グループにわけてそれぞれ別の目標関数を追跡できるようにする。
・2,3,4,5項目網羅くらいまでのパターンファイルを選択できるようにする。

## [5].2　将来

・Freecad or Openscadモデルのパラメータとリンクさせstl作成から一貫して
通すこともできる形にしたい。
・目標関数の分割というより、自動的にいくつかのメッシュ単体をサンプル点、中間ノードとして
(私も勉強中ですがpythonの機械学習系ライブラリにそういった物がありそうです)
その物理値によりパラメータを分離評価できるようにしたい。
(分離評価により少数の施行だけでパラメータをどちらに振るか判別可能)



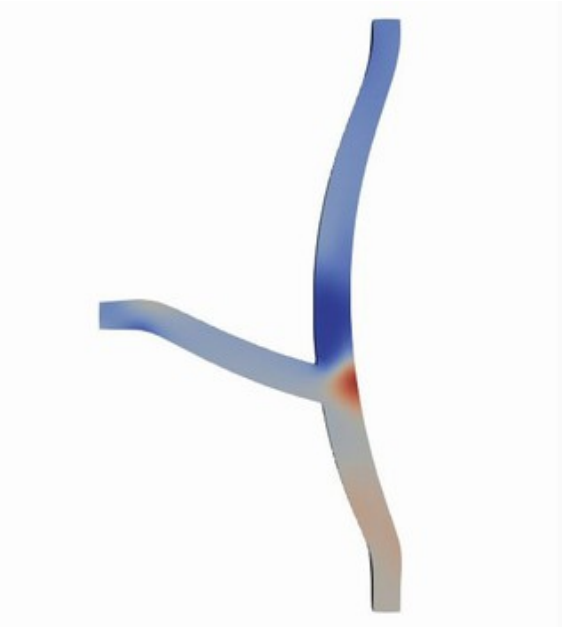
# [2]自分の手法の適用結果

**[2].5 1階全試行のベストノードに対する[2].4適用(+9)試行)**

| | |
|---|---|
| 00100100 | 4.2883029 |

| | |
|---|---|
| 00000100 | 6.41225 |
| 00100000 | 121.56318360000002 |
| 00100100 | 43.41689960000001 |
| 00100101 | 0.8518184000000004 |
| 00100110 | 1.912130399999995 |
| 00101100 | 27.163440000000012 |
| 00110100 | 20.84254250000002 |
| 01100100 | 218.35394610000006 |
| 10100100 | 197.56148839999994 |

峰③(スコア0.852)



**これまでのベストスコアを大きく更新する峰を発見**

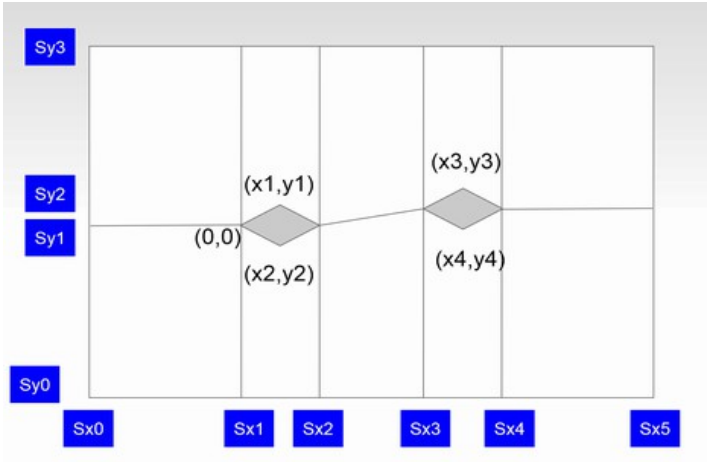
# [2]自分の手法の適用結果

**[2].6 まとめ**

**・本問題に対しては自分の手法の長所が生きにくい**
**(目標関数が分離しにくい、8パラメータすべて寄与度が高そう)であったが**
**ベイズ最適化よりも少数でより良いポイントを見出したり、**
**未発見の峰を見つけることができ同等以上のパフォーマンスとなった。**
**ただしOpenFoamの計算が成立しないポイントが相応数あり**
**ベイズ最適化の真価が発揮できない可能性もあったのでこれの解決策も調査する。**
**(自分の手法は計算が成立しないポイントが多くても相応に使える事もわかった)**

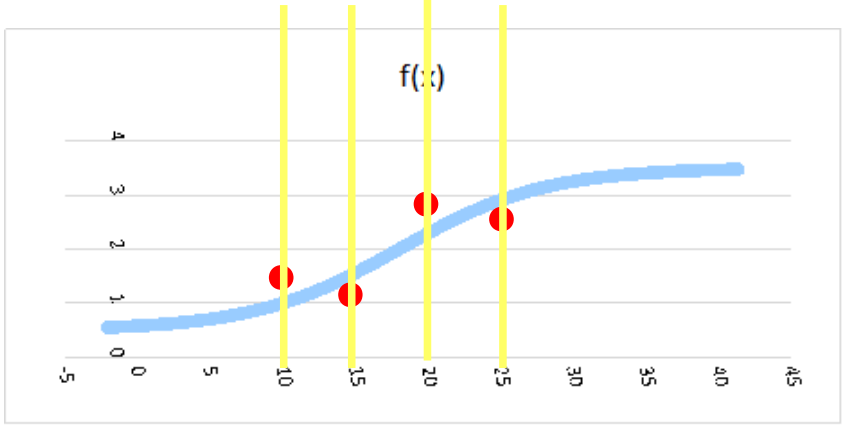
# [3]今後の方針

[3].1 網羅パターン作成
2項目網羅のラベル名とパラメータ列(ベクトル)からn項目網羅を自動生成する
プログラムを作成したのでさしあたり8因子と16因子の生成段階ごとの
ファイルを用意する。
[3].2 ルーチン汎用化
複数の子ノード自動作成と線形過程部分はまだ自動化出来ていないので作成する。
今回比較対象のベイズ最適化についてgpyoptというライブラリを使いましたが
これに似たインターフェースで上下限値設定をできるように作って
公開しようと考えております。
[3].3目標関数分割対応(2翼モデルでの例示)
野村様に作って頂いた右のモデルで
14変数まで設定可能なのと
目的関数分割に対応しているので
これでテストして例示とする予定。



# ベイズ最適化検証

**[X].ベイズ最適化検証**
**・人間の肺の付加圧力(m・H2O)と膨張容積(L)は4パラメータのシグモイド関数で表せることがわかっている。**
**それで疑似的にパラメータを決めて測定結果を作り、(A=0.5 B=3 C=18 D=5)**
**ベイズ最適化で測定結果x=10,15,20,25からパラメータの逆算を試みたところ300試行で0.528、2.758、17.492、4.608となり上下限幅に対し±5～10%ぐらいの誤差(こちらにはまだ自分の最適化を適用していないので今後確認する)**




膨張容積=f(x)=A+B/(1+exp(-(x-C)/D
評価するスコアは試行した赤点と真値(青線と黄線との交点)との差の2乗の足し合わせ。
ベイズ最適化を適用するにあたり上下限の設定はA[-1,1],B[1,5],C[15,30],D[2,10]とした。

# 前回報告分Index

# [1]SGS法の概要

## [1].1 SGS法の狙い

①.n個の0,1因子に対しに対し任意のk因子間の組み合わせを網羅する
パターンを作成する。

パターン1:0101□□1001
パターン2:0001□□0100
パターン3:0101□□0100
:
パターンm:1000□□1010

2因子網羅(どの2列を選択しても(0,0),(0,1),(1,0)(1,1)が存在する)

②.①とフラクタルの概念を活用し連続量の因子で非線形、交互作用がある
目的関数に対しても因子数に対し線形相当のオーダー内で停留点に漸近する。

$f(x_1,x_2,x_3,\square,x_n)=x_1+2*x_2+(x_3-0.5)^2+\square$ ← 非線形項

$g(x_1,x_2,x_3,\square,x_n)=x_1+(x_2+0.5)*(x_3+0.5)+\square$ ← 交互作用項

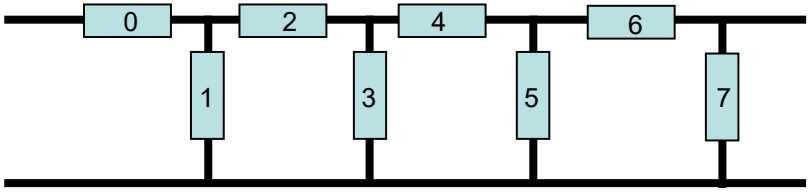
目的関数

# [1]SGS法の概要

**[1].2　SGS法の適用例**

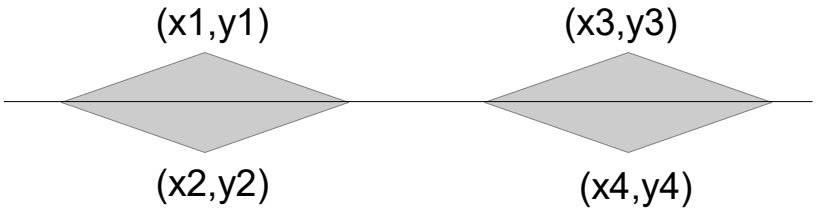
**①ソフトのテストパターン作成(前ページの①参照)**

**②電子回路の素子の特性値について設計値からの**
**ばらつきに対する変動を良く行われるモンテカルロ法と違い乱数に頼らず**
**明確にk素子間のパターンを網羅して確認できる。**




K個間の素子の特性値のばらつきの組み合わせを網羅して不具合発生のリスクが無いか確認

**③翼形状の各種寸法をどのようにすれば空気抵抗等の特性が良くなるか検索する。**




各寸法の組み合わせを網羅しつつ
最適な設計値を検索

# [2]網羅パターンの生成方法

[2].1　n項目中2項目の組み合わせの網羅パターンの生成方法
以下簡略のためnは2の累乗とする
　　(2の累乗に該当しない数はそれよりも大きい2の累乗でパターン生成)
ここで因子に0から2^i-1の2進数を割り振り
それぞれの桁(と0,1を反転させた物)をパターンとすれば
任意の2つの因子について(0,0)(0,1)(1,0)(1,1)が少なくとも1回は出現する
証明
(0,0)(1,1)はオール0,オール1の時に全項目について発生する。
(0,1)(1,0)は同一の数で無い限り2進表記した際に異なる桁が
少なくとも1つ存在するのでそこで形成される。

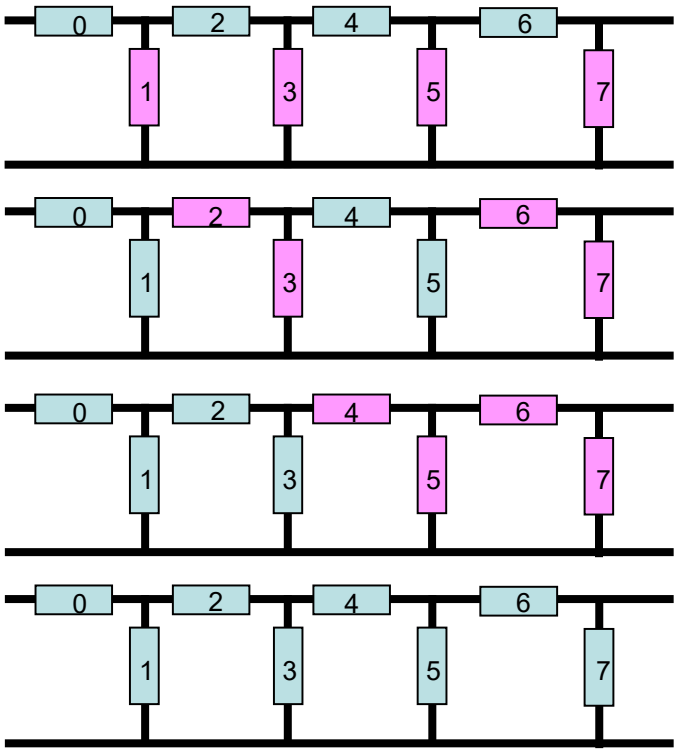
# [2]網羅パターンの生成方法

例(8因子)

| | | 10進表記(因子に0から7の番号を割り付けたもの) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2進表記 順 | 1桁 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | 2桁 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | 3桁 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | 4桁 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2進表記 反 | 1桁 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| | 2桁 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| | 3桁 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| | 4桁 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**表の8行分のパターンにより任意の2因子間で(0,0)(0,1)(1,0)(1,1)が網羅されている。**
**試行のオーダーはlog n**

**回路での例示**
**(付番順に表の値の0を青、1を赤として表示)**




**上記4つとそれの赤、青反転の**
**8つ用意すればどの2素子間でも**
**(赤、赤)、(赤、青)、(青、赤)、(青、青)が存在する**

# [2]網羅パターンの生成方法

例(16因子)

| | | 10進表記(因子に0から15の番号を割り付けたもの) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 2進表記 順 | 1桁 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | 2桁 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | 3桁 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | 4桁 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 5桁 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2進表記 反 | 1桁 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| | 2桁 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| | 3桁 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| | 4桁 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 5桁 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**8因子に比べ2パターンだけ増える要領となる。**

# [2]網羅パターンの生成方法

**[2].2 n項目中3項目の組み合わせの網羅パターンの生成方法①**
**2項目網羅パターンから任意の2つを選びそれぞれの桁に下記表の演算を施したパターンを作成する。**

演算①

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

**証明**
**2項目網羅である3項目間に(0,0,1)という組み合わせが形成できなかったとする。**
**この場合2項目は網羅していることから(0,0,1)から1項目だけ異なる**
**(1,0,1)(0,1,1)が存在することになり**
**それでこれらに演算①が施されるとされるとそれが(0,0,1)となる。**
**2項目網羅である3項目間に(0,1,1)という組み合わせが形成できなかった場合は対称性より(1,0,0)が存在しないことになり上記と同様の議論と演算①で(0,1,1)が形成される。**
**この演算で3項目の網羅が確保されることがわかる。**

# [2]網羅パターンの生成方法

例(8因子)、演算①　(赤字：3項目網羅で新規に出現したパターン)

| | | 2項目網羅パターン | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 00000000 | 00001111 | 00110011 | 01010101 | 11111111 | 11110000 | 11001100 | 10101010 |
| 2項目網羅パターン | 00000000 | 00000000 | 00001111 | 00110011 | 01010101 | 11111111 | 11110000 | 11001100 | 10101010 |
| | 00001111 | - | 00000000 | 00111100 | 01011010 | 11110000 | 11111111 | 11000011 | 10100101 |
| | 00110011 | - | - | 00000000 | 01100110 | 11001100 | 11000011 | 11111111 | 10011001 |
| | 01010101 | - | - | - | 00000000 | 10101010 | 10100101 | 10011001 | 11111111 |
| | 11111111 | - | - | - | - | 00000000 | 00001111 | 00110011 | 01010101 |
| | 11110000 | - | - | - | - | - | 00000000 | 00111100 | 01011010 |
| | 11001100 | - | - | - | - | - | - | 00000000 | 01100110 |
| | 10101010 | - | - | - | - | - | - | - | 00000000 |

赤字が3項目網羅で新規に出現したものであり6である(2項目網羅は8)

試行のオーダーは(log n)^2

(ただし数学者の友人曰く、もっとオーダーが少ないアルゴリズムがある可能性があるとのこと。現在探索中)

# [2]網羅パターンの生成方法

**[2].3 n項目中3項目の組み合わせの網羅パターンの生成方法②**
**2項目網羅パターンから任意の2つを選びそれぞれの桁に下記の演算②、③を施したパターンを作成する。**

演算②

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

演算③

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**証明**
**2項目網羅である3項目間に(0,0,1)という組み合わせが形成できなかったとする。**
**この場合2項目は網羅していることから(0,1,1),(1,0,1)が存在することになる。**
**それで(0,1,1) (1,0,1) に演算③を行うとそれが(0,0,1)が生成される。**
**2項目網羅である3項目間に(0,1,1)という組み合わせが形成できなかった場合は上記と同様の議論と演算②で(0,1,1)が形成される。**
**この演算で3項目の網羅が確保されることがわかる。**

# [2]網羅パターンの生成方法

例(8因子)、演算② (赤字:3項目網羅で新規に出現したパターン)

| | | 2項目網羅パターン | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 00000000 | 00001111 | 00110011 | 01010101 | 11111111 | 11110000 | 11001100 | 10101010 |
| 2項目網羅パターン | 00000000 | 00000000 | 00001111 | 00110011 | 01010101 | 11111111 | 11110000 | 11001100 | 10101010 |
| | 00001111 | - | 00001111 | 00111111 | 01011111 | 11111111 | 11111111 | 11001111 | 10101111 |
| | 00110011 | - | - | 00110011 | 01110111 | 11111111 | 11110011 | 11111111 | 10111011 |
| | 01010101 | - | - | - | 01010101 | 11111111 | 11110101 | 11011101 | 11111111 |
| | 11111111 | - | - | - | - | 11111111 | 11111111 | 11111111 | 11111111 |
| | 11110000 | - | - | - | - | - | 11110000 | 11111100 | 11111010 |
| | 11001100 | - | - | - | - | - | - | 11001100 | 11101110 |
| | 10101010 | - | - | - | - | - | - | - | 10101010 |

# [2]網羅パターンの生成方法

例(8因子)、演算③ (赤字:3項目網羅で新規に出現したパターン)

| | | 2項目網羅パターン | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 00000000 | 00001111 | 00110011 | 01010101 | 11111111 | 11110000 | 11001100 | 10101010 |
| 2項目網羅パターン | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| | 00001111 | - | 00001111 | 00000011 | 00000101 | 00001111 | 00000000 | 00001100 | 00001010 |
| | 00110011 | - | - | 00110011 | 00010001 | 00110011 | 00110000 | 00000000 | 00100010 |
| | 01010101 | - | - | - | 01010101 | 01010101 | 01010000 | 01000100 | 00000000 |
| | 11111111 | - | - | - | - | 11111111 | 00001111 | 11001100 | 10101010 |
| | 11110000 | - | - | - | - | - | 11110000 | 11000000 | 10100000 |
| | 11001100 | - | - | - | - | - | - | 11001100 | 10001000 |
| | 10101010 | - | - | - | - | - | - | - | 10101010 |

**演算②、③で3項目網羅として新しく24必要となり演算①より必要なパターン数は多くなるが3項目網羅を再び②、③にかければ4項目網羅以上のパターンを作成していける。(つまり繰り返せば全ての8ビットパターン(総数2^8)をつくれる。)**

# [2]網羅パターンの生成方法

**[2].4 n項目中4項目の組み合わせの網羅パターンの生成方法**
**3項目網羅パターンから任意の2つを選び演算②、③を施す。**

**証明**
**・3項目網羅内にある4項目間に(0,0,0,1)という組み合わせが形成できなかったとする**
**この場合3項目は網羅していることから(0,0,1,1),(0,1,0,1)が存在することになる。**
**それで(0,0,1,1),(0,1,0,1)に演算②を行うと(0,0,0,1)が生成される。**
**・3項目網羅内に(0,0,1,1)というパターンが形成できなかったとする**
**この場合3項目は網羅していることから(0,0,0,1),(0,0,1,0)が存在することになる。**
**それで(0,0,0,1),(0,0,1,0)に演算③を行うと(0,0,1,1)が生成される。**

**[2].5 n項目中5項目以上の組み合わせの網羅パターンの生成方法**
**k-1項目網羅パターンから任意の2つを選び演算②、③を施せば**
**4項目網羅と同じ要領でk項目網羅作成できる。**
**(つまり繰り返せば全ての2^nビットパターンをつくれる。)**

# [3]連続変数の最適条件検索

**[3].1 連続変数への網羅パターンの適用方法概要**
**[2]で作成した網羅パターンを活用し連続量の因子に対する最適条件検索を以下の手順で行う。**

**前提**
**n個の連続量の因子(機械寸法、素子の特性値等)の上限値、下限値をそれぞれ1,-1とする(中央値が0)。**
**手順①**
**[2]で作成したk項目網羅のパターンの1を1/2、0を-1/2に対応させてシミュレーションや実験を試行する。**
**(k項目の組み合わせを網羅していることによりk因子間の交互作用をどこかのパターンに取り込める。)**
**(因子数n全項目相当の最適値検索を行う場合はk項目間網羅の施行数をn(or 2n)を超えるまで進め線形代数と仮定(後ページに記載)して各々の因子が1/2,-1/2どちらが好ましいか推測できる。)**

# [3]連続変数の最適条件検索

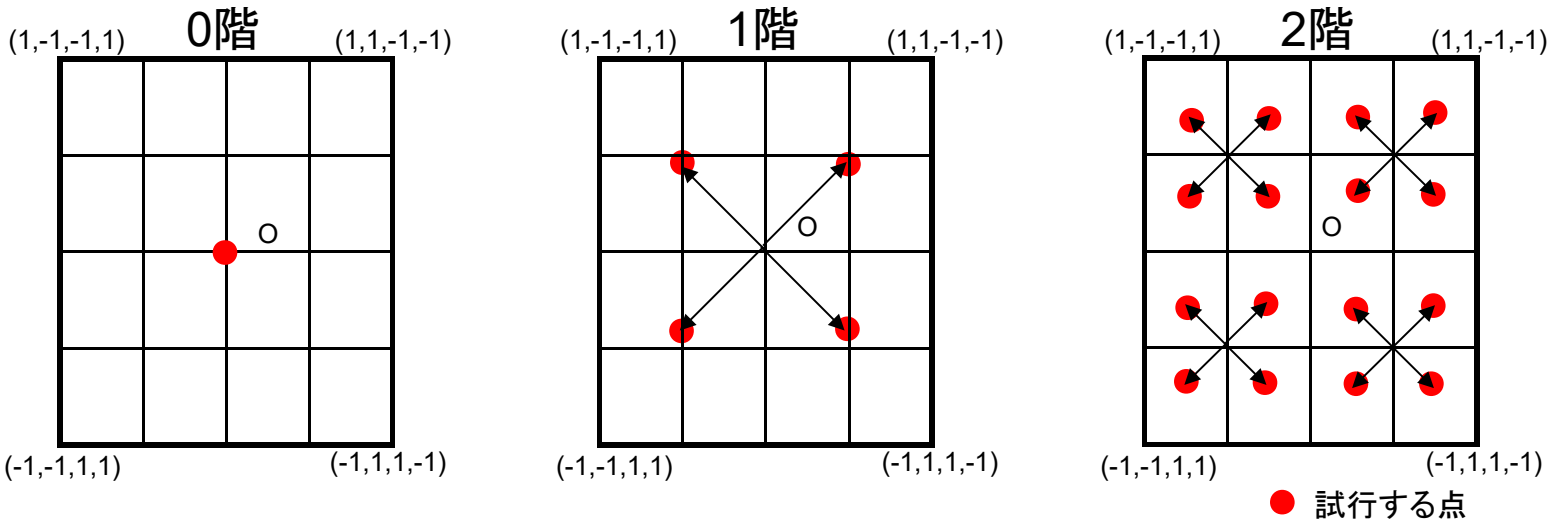
**手順②**

**①で試行した各点に対し用意したパターンの長さを1/2倍したベクトル分移動した点で再度試行を行う。これを繰り返す。**

**(今後繰り返し回数の事を[階]とよぶ。)**

**目標関数が非線形でも各設計因子による変化が微分可能であるならばどこかの階以降で目標関数の変異が線形に近づくのでそれを見極める。**

例:4因子に対し2パターン(1,1,0,0),(1,0,0,1)とそれらの逆を使用した場合




試行回数を減らす時はある階で目標関数の値が悪かった点を次の階で試行しない。

# [3]連続変数の最適条件検索

**[3].2 線形代数を仮定しての最適な因子変動探索**
**ある階において項目数のn以上のパターンを実施した後、そのパターンからn個抽出し線形に振る舞うと仮定して目的関数の変動量から各々の因子の係数(c)を解く。**
**それでその階で改めて正係数の因子には1、負係数の因子には-1をあてがい再度試行する**
**(階が進むに従い変動は線形に近づくと思われるが階が少ないうちは nより多くのパターンを施行し、線形代数を複数解いて係数の正負が揺れる 因子についていくつか拾い0,1両方施行)**

例:8因子に対する3項目間網羅結果使用

| パターン (3項目網羅で先頭が0の16個) | 試行結果 (前階の値を引いたもの) |
|---|---|
| (0,0,0,0,0,0,0,0) | y1 |
| (0,0,0,0,1,1,1,1) | y2 |
| (0,0,1,1,0,0,1,1) | y3 |
| (0,1,0,1,0,1,0,1) | y4 |
| (0,0,0,1,0,1,0,1) | y5 |
| ・ | ・ |
| ・ | ・ |
| ・ | ・ |
| (0,0,0,1,0,1,0,1) | y16 |

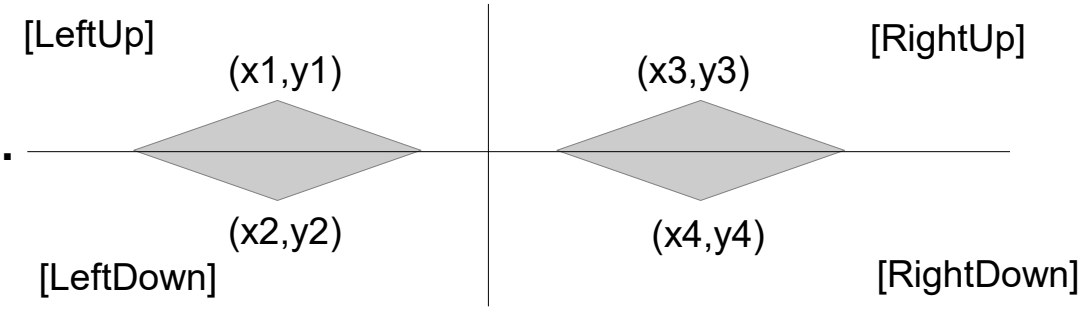
8パターン選択

c=M^(-1)*y

c :係数ベクトル
M^(-1) :パターンの逆行列
(1は1/2、0は-1/2に置き換え)
y :試行結果ベクトル

# [3]連続変数の最適条件検索

**[3].3 目的関数の分割**
**因子をその位置等で2や4のグループに分け、その近辺のエリアで**
**目的関数を分割設定しそれぞれのエリア独立で探索していく。**



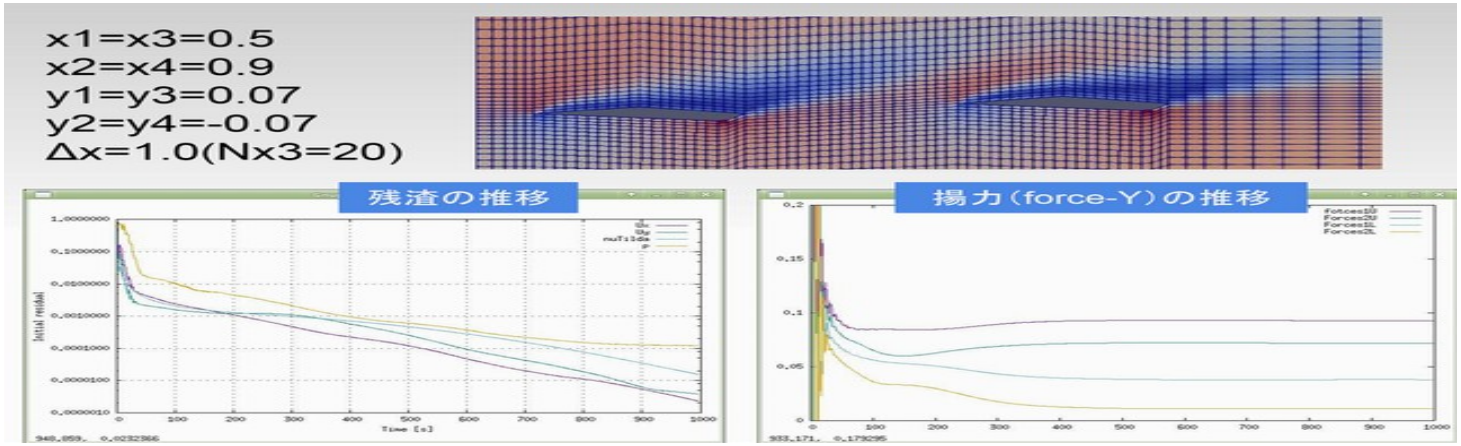

**目的関数の分離性があれば近辺の因子のみで評価できる。**
**(上の例の場合、仮に縦 or 横棒で区切ったエリアで分離できれば4因子間変動のみで評価できる、さらに縦、横双方で区切れれば2因子間変動のみで探索できる)**

**分割した際に本手法のスタートに用意したk項目網羅パターンが生きてくる要領**
**(分割できたエリア内の因子間でも当然k項目網羅がされているので**
**因子間の網羅率が高い)**

# [4]現在進行中の活動

**[4].1 効果検証**

**野村様に2翼モデルの作成と8パラメータ2階分(約2の16乗)の流体シミュレーションを実施して頂きました。**




**[4].2 最適化ルーチンの汎用化**

**BlockMeshについて(1)パラメータ指定(2)Mesh作成(3)SImpleFoam実行を繰り返せるプログラムは作れました。**

**。**