最適化ルーチン使用方法、理論概説

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,5項目網羅)
  - 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 (3項目網羅で先頭が0の16個)
0000000000000000 1000000000000000 0100000000000000 1100000000000000 1010000000000000 . . . 1111111111111111

cross1(3項目網羅)

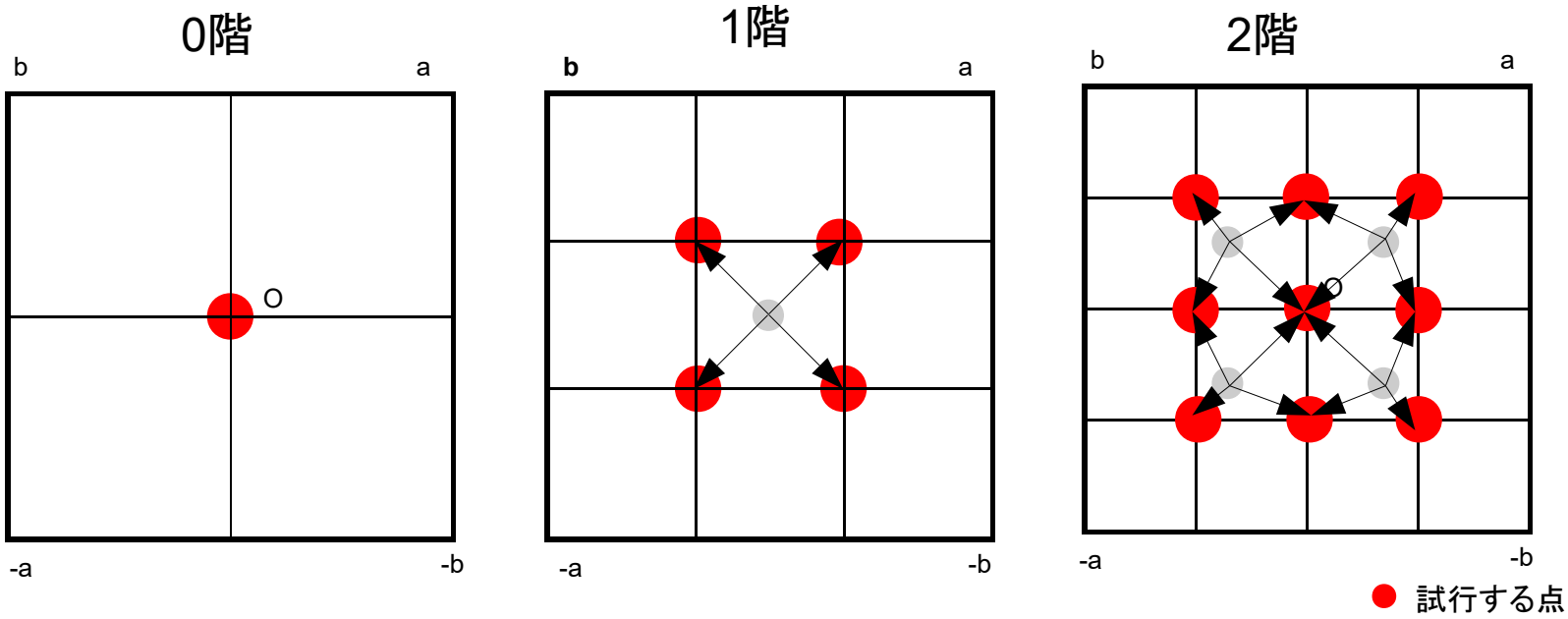
パターン (3項目網羅で先頭が0の16個)
0000000000000000 1111111100000000 1111000011110000 0000111111110000 1100110011001100 . . . 1111111111111111

(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

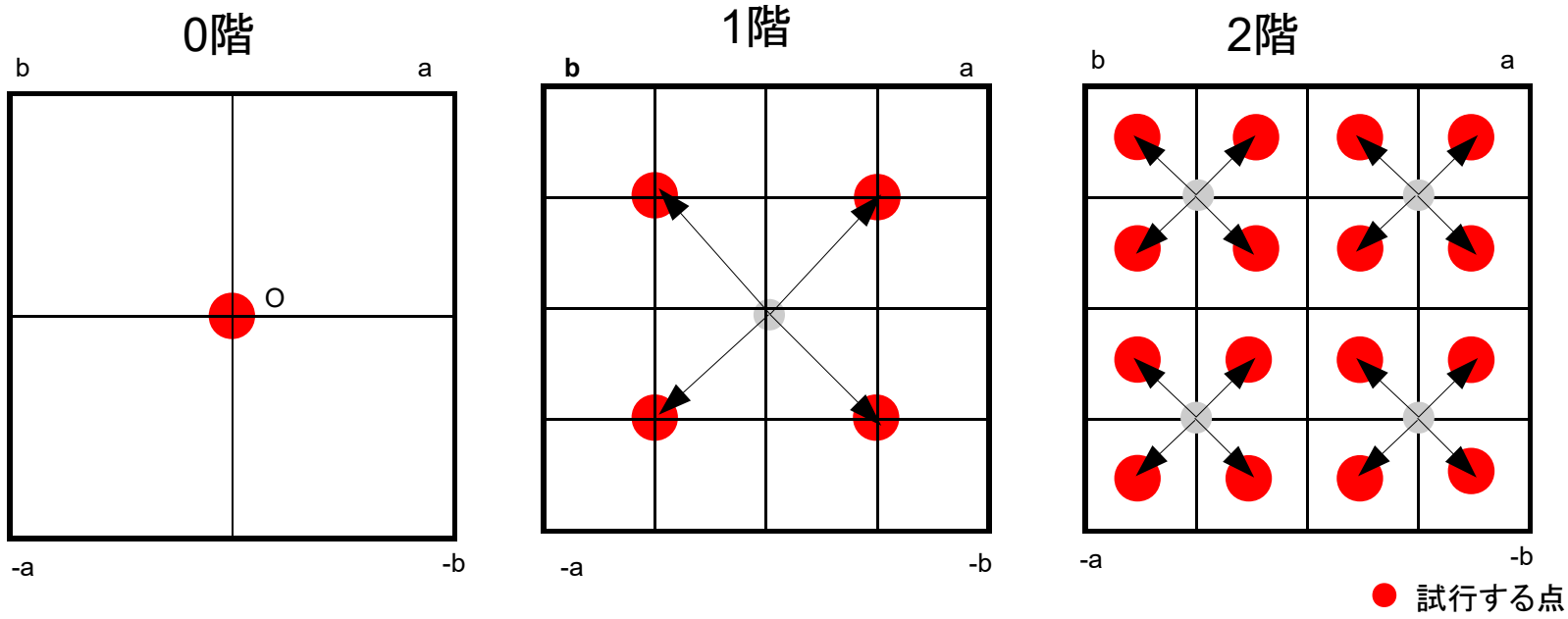


いくつかのテスト関数でうまく最適パラメータに移動することを確認

[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

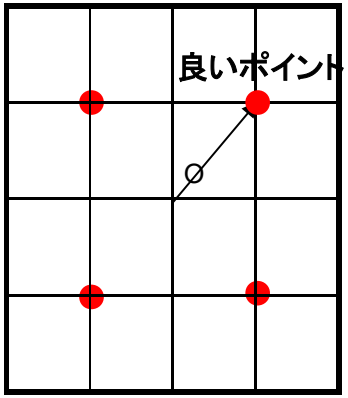


汎用性が低いので後日実装

[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

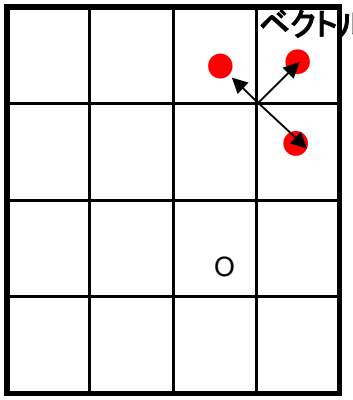
N階　(1,1,-1,-1,1,1,-1,-1)



● 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)