

ベイズ最適化の チュートリアル

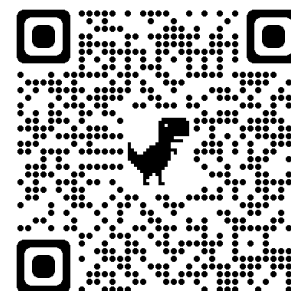
理化学研究所仁科加速器科学研究センター

森田 泰之

はじめに

実行環境

Google Colaboratory
<https://colab.google/>



今回使用するファイル

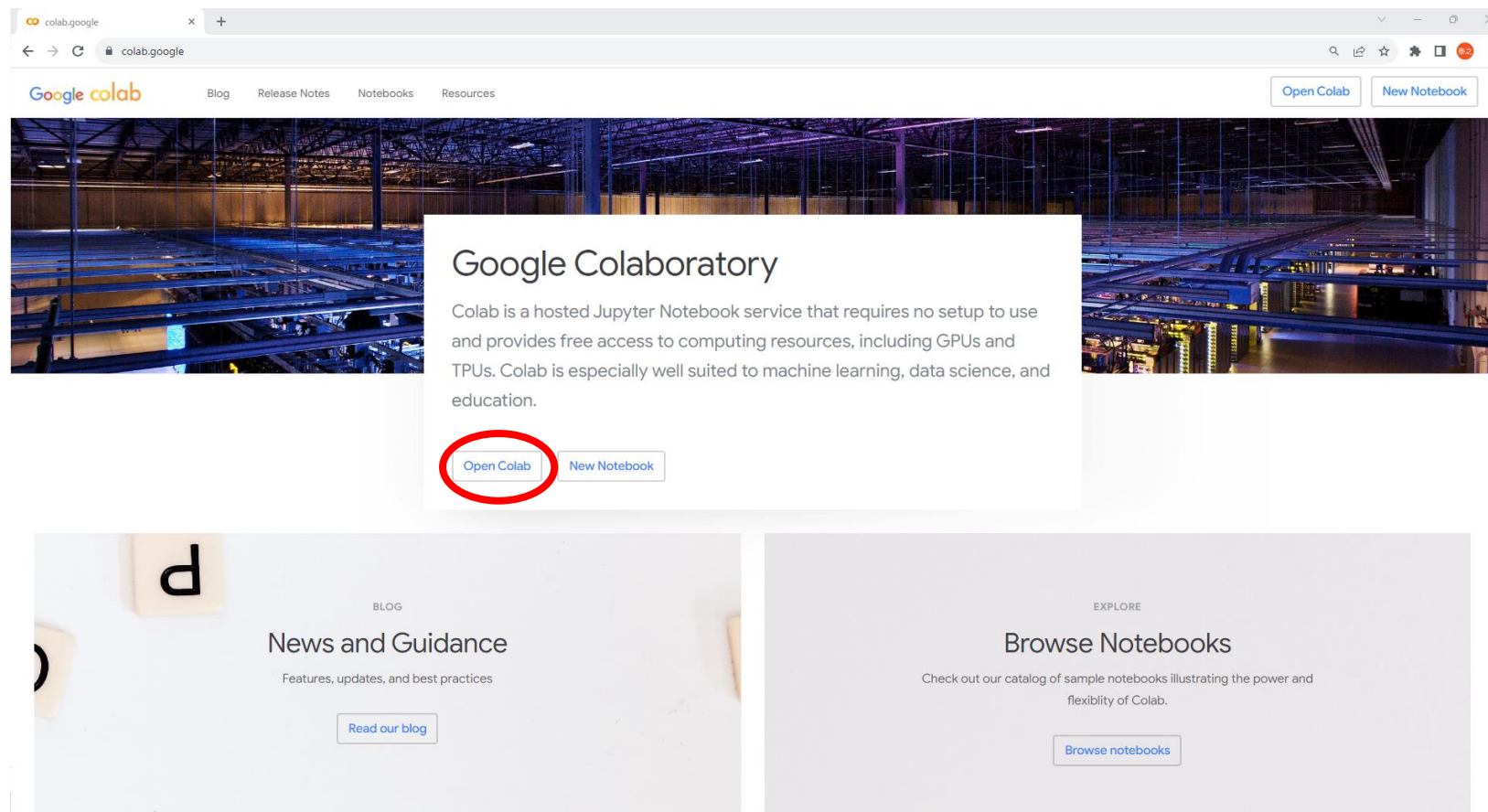
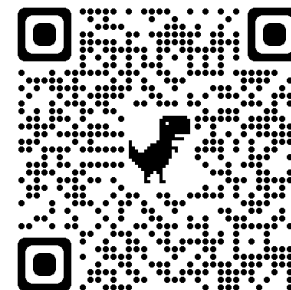
<https://github.com/Morita1116/Machine-Learning-Workshop-on-Accelerator-and-Beam-Physics-Tutorial-Materials>

- | | |
|---------------------|-------------------------------------|
| • Setup.txt | Google Colaboratory上に環境を構築するためのコマンド |
| • BO_2parameters.py | プログラム例(2パラメーターの最適化) |
| • BO_4parameters.py | プログラム例(4パラメーターの最適化) |

はじめに

実行環境

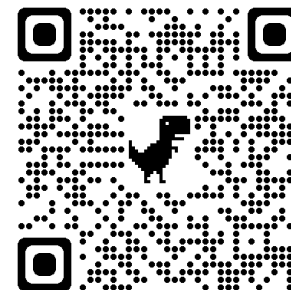
Google Colaboratory
<https://colab.google/>



はじめに

実行環境

Google Colaboratory
<https://colab.google/>



Open notebook

Examples >

Recent >

Google Drive >

GitHub >

Upload >

Search notebooks

Title	Last opened ▲	First opened ▼	
Copy of Untitled3.ipynb	November 2	November 2	
Untitled3.ipynb	November 2	November 2	
Welcome To Colaboratory	November 2	Feb 2, 2021	
Untitled2.ipynb	November 2	November 2	
Untitled1.ipynb	November 2	November 2	
BD_ECR.ipynb	July 5	July 5	
01.ベイズ最適化概説.ipynb	May 20, 2022	Mar 31, 2021	

+ New notebook

Cancel

実行ファイルの中身0

実行ファイルの中身1

```
import GPy
import GPyOpt
import matplotlib.pyplot as plt
import numpy as np
import time

from ocelot import *
```

各種ライブラリの読み込み

GPy, GPyOpt	ベイズ最適化計算用のライブラリ
matplotlib.pyplot	グラフ描写用のライブラリ
numpy	配列計算用のライブラリ
time	時間関連の情報や関数のライブラリ
Ocelot	ビーム輸送計算用ライブラリ

実行ファイルの中身2

`def calc_resp(x0,x1):` Ocelotによる軌道計算を行う関数

L20, L50

QM1, QM2

cell, lat

tws0

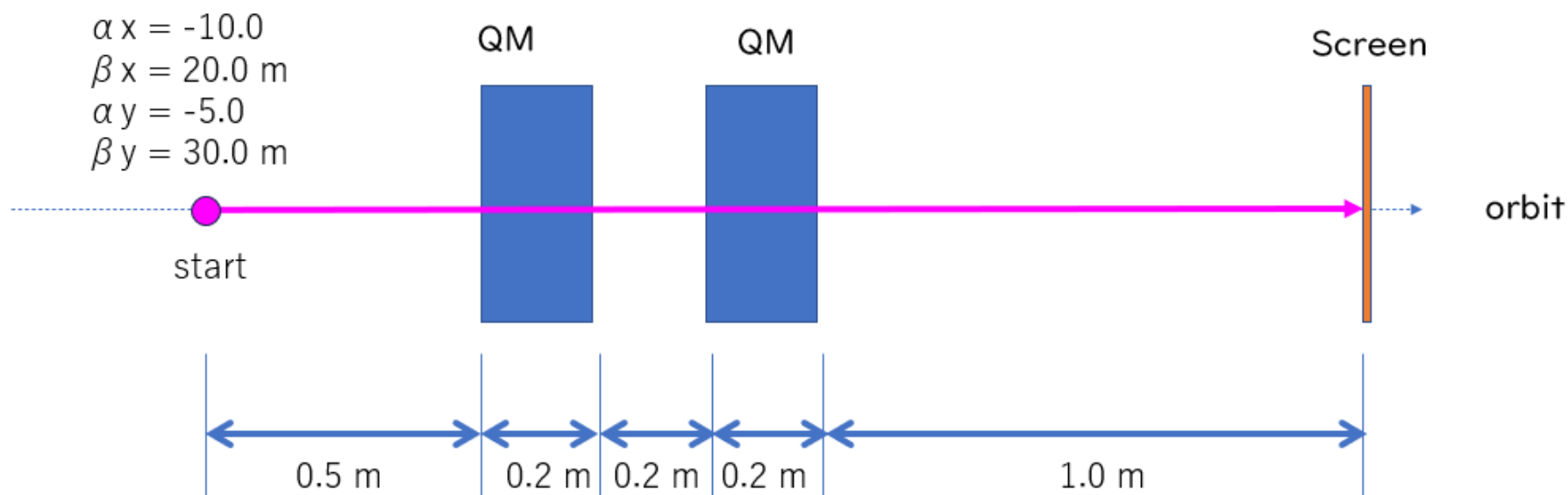
sx, sy

ドリフト空間。それぞれ0.2m,0.5m

四重極電磁石(有効磁場長0.2m) ←この磁場を最適化する
ビーム輸送系の配置

初期条件 $\beta_x, \beta_y, \alpha_x, \alpha_y$, Emittance (x, y)

Screen上での β_x, β_y



実行ファイルの中身3

`def show_beta(x0,x1):` Ocelotの計算結果を描写する関数
matplotlib.pyplotを使って描写される

`def eval_func(x):` ベイズ最適化の評価関数。この関数の値を指標に最適化を行う。
実機ではビーム強度などの測定値を使う。

評価関数の例

- `val = np.abs(sx) + np.abs(sy)` ターゲット上の β_x, β_y の和
- `val = np.abs(sx*sy)` ターゲット上の β_x, β_y の積
- `val = np.log(np.abs((sx+10)*(sy+10)))` ターゲット上の β_x, β_y の積の対数
- `val = -1.0/np.abs((sx+10)*(sy+10))` ターゲット上の β_x, β_y の積の逆数

その他、

`beta_x = [p.beta_x for p in tws], beta_y = [p.beta_y for p in tws]`

で全体を計算し、`np.max(beta_x)`で最大値を考慮するなどの工夫も可能

実行ファイルの中身4

```
# ===== Main =====
```

```
bounds = ~
```

各変数(今回は電磁石)の設定範囲

continuous: 連続的な変数。設定可能範囲をdomainに入力。

discrete: 離散的な変数。設定可能な値を配列としてdomainに入力。

```
myBopt = ~
```

ベイズ最適化のモデル

- **f**
- bounds
- **initial_design_number**
- acquisition_type
- acquisition_weight, jitter
- de_duplication
- normalize_Y
- maximize

評価関数 (eval_func)

変数の設定範囲

初期条件用のデータサンプリング数

獲得関数 (LCB,PI,EI)

大きいほど探索重視、小さいほど予測重視

重複したデータのサンプル(Trueで重複しない)

評価関数の規格化

評価関数の最大化(True) or 最小化(False)

```
myBopt.run_optimization(max_iter=**)
```

```
myBopt.plot_acquisition()
```

```
myBopt.plot_convergence()
```

最適化の実行。()内はイタレーション数

評価関数や獲得関数の描写。3次元以上は不可
最適化結果の推移の描写

Setup

Setup.txt中のコマンド

```
pip install GPy GPyOpt git+https://github.com/ocelot-collab/ocelot.git pyfftw numexpr numba
```

をGoogle Colaboratoryで実行



```
pip install GPy GPyOpt git+https://github.com/ocelot-collab/ocelot.git pyfftw numexpr numba
```



GPy, GPyOpt
Ocelot
その他

ベイズ最適化計算用
ビーム軌道計算用
Ocelot計算に必要なあれこれ

がInstallされる

ただしGoogle Colaboratoryの場合、
12時間でリセットされてしまうため注意

実行

Google Colaboratoryにファイルの中身をコピー＆ペーストとして実行

実行すると数字の羅列が表示される
(eval_func中のprint部分)
左から

- iteration
- QM1設定値
- QM2設定値
- Screen上のサイズ(x)
- Screen上のサイズ(y)
- 評価関数の値

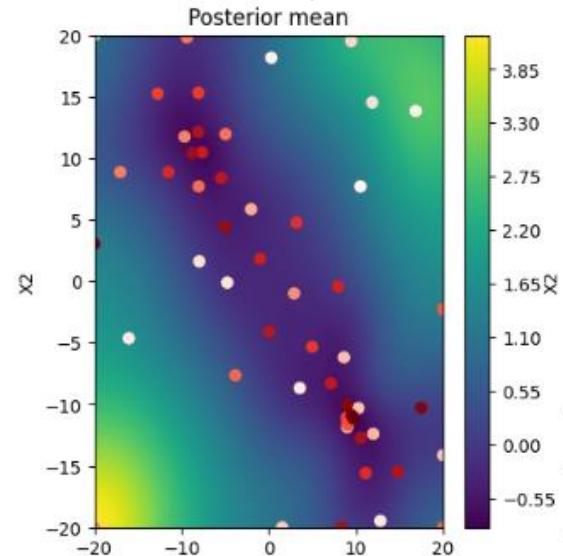
```
==== start ====  
# cnt, x0, x1, sx, sy, eval_val  
0: +10.508, +7.714, +11.392, +48.520, +59.912  
1: +0.300, +18.167, +14.704, +37.158, +51.862  
2: +16.858, +13.840, +16.079, +91.605, +107.683  
3: +3.560, -8.673, +13.402, +0.720, +14.121  
4: -16.036, -4.628, +55.522, +18.101, +73.623  
5: +12.796, -19.454, +5.122, +16.845, +21.967  
6: -4.751, -0.100, +17.908, +1.049, +18.958  
7: -7.952, +1.625, +20.017, +5.258, +25.276  
8: +11.868, +14.532, +14.526, +73.900, +88.426  
9: +9.450, +19.526, +15.920, +79.295, +95.215  
10: -20.000, +20.000, +17.762, +34.210, +51.972  
11: +1.548, -20.000, +36.385, +17.401, +53.786  
12: +20.000, -14.131, +26.241, +0.511, +26.751  
13: +10.285, -10.301, +2.157, +2.536, +4.693  
14: +8.626, -6.174, +0.967, +9.840, +10.807  
15: +11.990, -12.394, +5.492, +0.920, +6.412
```

実行回数は

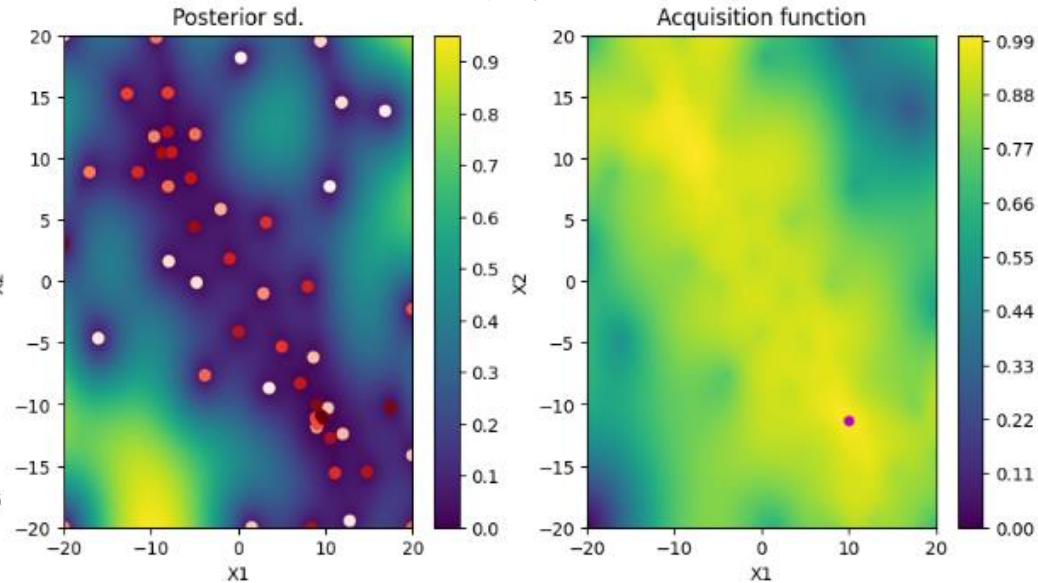
initial_design_number + max_iter

実行結果例

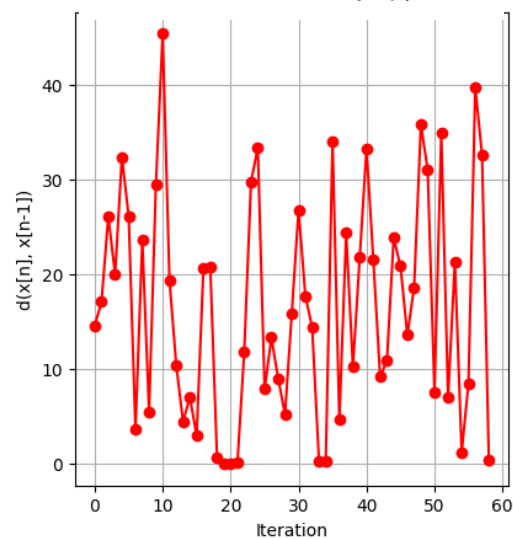
予測値



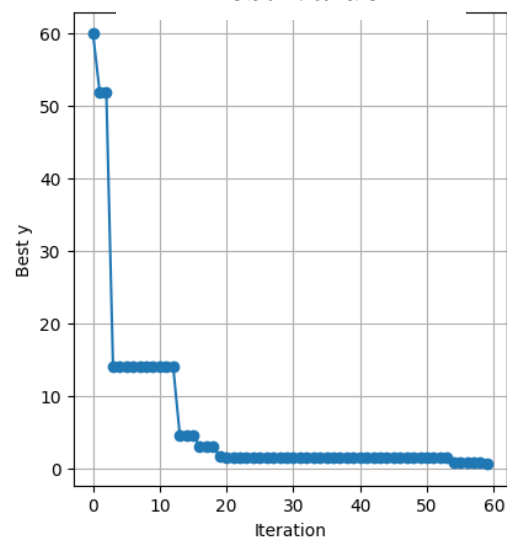
獲得関数



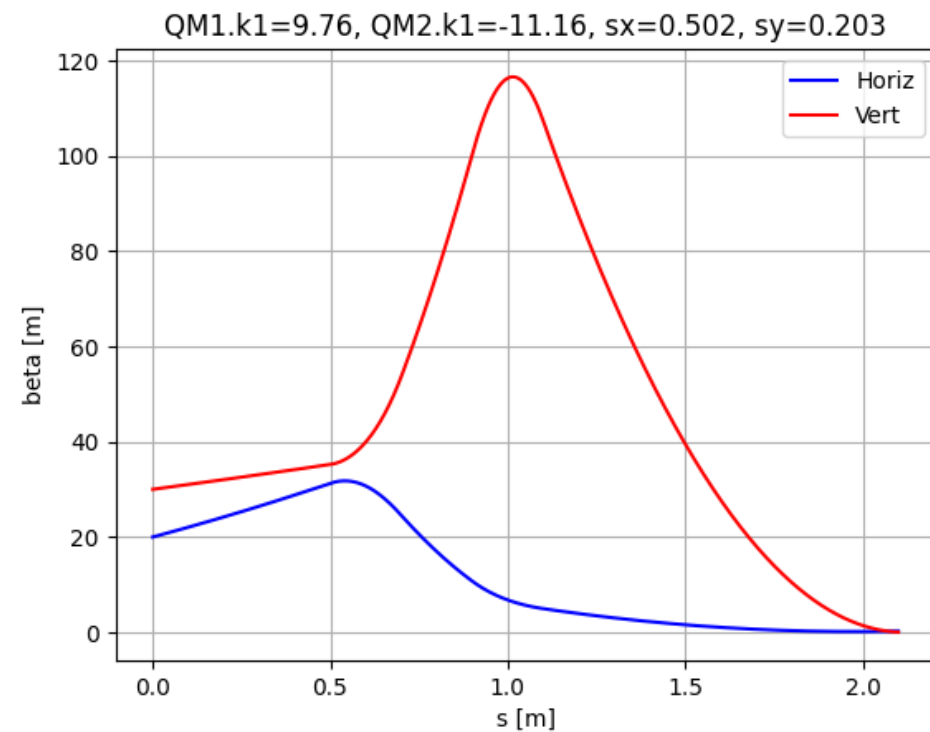
パラメータ変化量



Best Value



Beam envelope



Best Valueでのenvelope

各種設定値を変更すると収束までのiteration数やビームの形状などが変化します。
皆様で各種設定を変更してみてください。

- ビーム初期条件
- 評価関数
- 獲得関数
- acquisition_weight (LCB) / jitter (PI,EI)
- max_iter
- QMの設定範囲
- QMの数(4台: BO_4parameters.py)
etc.

12時ごろまではZoomでも質問を受け付けておりますので、お気軽にご質問ください。

実行ファイルの中身4

```
# ===== Main =====
```

```
bounds = ~
```

各変数(今回は電磁石)の設定範囲

continuous: 連続的な変数。設定可能範囲をdomainに入力。

discrete: 離散的な変数。設定可能な値を配列としてdomainに入力。

```
myBopt = ~
```

ベイズ最適化のモデル

- f
- bounds
- initial_design_number
- acquisition_type
- acquisition_weight, jitter
- de_duplication
- normalize_Y
- maximize

評価関数 (eval_func)

変数の設定範囲

初期条件用のデータサンプリング数

獲得関数 (LCB,PI,EI)

大きいほど探索重視、小さいほど予測重視

重複したデータのサンプル(Trueで重複しない)

評価関数の規格化

評価関数の最大化(True) or 最小化(False)

```
myBopt.run_optimization(max_iter=**)
```

最適化の実行。()内はイタレーション数

```
myBopt.plot_acquisition()
```

評価関数や獲得関数の描写。3次元以上は不可

```
myBopt.plot_convergence()
```

最適化結果の推移の描写