Parameter Estimator ユーザーズガイド 第6版

輸出する際の注意事項

本製品（ソフトウェアを含む）は、外国為替および外国貿易法で規定される規制貨物（または役務）に該当することがあります。

その場合、日本国外へ輸出する場合には日本国政府の輸出許可が必要です。

なお、輸出許可申請手続きにあたり資料等が必要な場合には、お買い上げの販売店またはお近くの当社営業拠点にご相談ください。

はしがき

◆本書は、 NEC Vector Annealingのご利用におけるParameter Estimatorの使用方法ついて説明したものです。

◆商標、著作権について

■記載されている会社名、製品名は、各社の登録商標または商標です。

用語定義

用語	説明
constraint mode	制約を満たすことを優先して解の探索を行います。VectorAnnealingの求解時にvector_mode=VectorAnnealing.VECTOR_MODE_CONSTRAINTを指定して実行します。
非constraint mode	エネルギーの最小化を優先して解の探索を行います。VectorAnnealingの求解時にvector_mode=VectorAnnealing.VECTOR_MODE_SPEEDを指定して実行します。

目次

1. Parameter Estimator の概要
2. 導入手順
3. 利用手順
4. 仕様
5. 使用時の注意事項

1.Parameter Estimator の概要

概要

- Parameter Estimator とは？
  - NEC Vector Annealing (VA)を利用する際、温度の逆数の入力パラメータ(ベータ)の値次第で求解性能が大きく変わります。また、ハミルトニアンが複数の項で構成される問題の場合、各項に掛ける重み係数の値によっても求解性能は大きく変わります。性能を引き出すためにはそれらパラメータをチューニングする必要があります。

  - Parameter Estimator はPyQUBO*を用いて作成した式を解析することで、それらパラメータの適切な値を推定するツールです。

*：PyQUBOとは
アニーリングマシンで組合せ最適化問題を解く際に、定式化した数式をQUBO形式に変換する
ドメイン固有言語（DSL）のことです。Pythonの他のライブラリと同様に使用することができます。
https://pyqubo.readthedocs.io/en/latest/index.html
https://github.com/recruit-communications/pyqubo

推定できる値

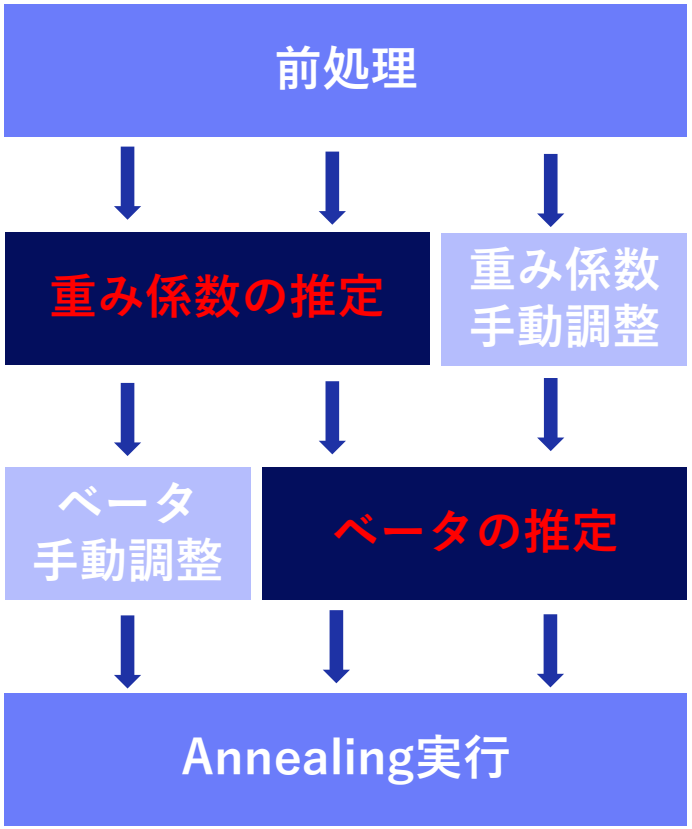
- Parameter Estimator では以下の値を推定することが出来ます。

出力項目	説明
ベータ (beta_list または beta_range)	自動推定した逆温度のデータ 推定した全逆温度のリスト形式(beta_list) または [最初、最後、総数] の形式(beta_range) で出力する
重み係数 (feed_dict)	ハミルトニアンの制約条件の項に掛ける重み係数のデータ {項名：重み係数, ...}の辞書形式で出力する

使用の流れ

- VAを用いた求解は以下の手順で行います。
  1. 前処理(インポート、式の作成)
  2. 重み係数の推定
  3. ベータの推定
  4. Annealing実行

Parameter Estimator を利用する部分



- Parameter Estimatorでは上記のうち重み係数の推定とベータの推定を行うことが出来ます。
- 求解に必要なsweep数の推定は行わないので、手動で調整してください。
- 重み係数・ベータの両方を推定することも、一方のみを推定し、もう一方を手動で調整することも可能です。
- 両方を推定する場合、重み係数の推定をベータの推定の前にしてください。

API一覧

- Parameter Estimatorは以下のAPIを提供します。
  - 重み係数を推定するAPI

API名	説明
get_feed()	非constraint modeの場合に用いるAPIです。 重み係数(feed_dict)を推定します。 なお、constraint modeでは使用できません。 high_orderには対応していません。

  - ベータを推定するAPI

API名	説明
get_beta()	非constraint modeの場合に用いるAPIです。 list形式([beta0, beta1, ..., betan])のbetaを推定します。
get_beta_range()	非constraint modeの場合に用いるAPIです。 range形式([start, end, nsteps])のbetaを推定します。
get_constraint_beta()	constraint modeの場合に用いるAPIです。 list形式([beta0, beta1, ..., betan])のbetaを推定します。
get_constraint_beta_range()	constraint modeの場合に用いるAPIです。 range形式([start, end, nsteps])のbetaを推定します。

2.導入手順

事前準備

◆ 用意しておくもの

■ NEC Vector Annealing (VA)の実行環境

- NEC Vector Annealingがインストールされている環境

■ parameter_estimator-3.0.0

- 別途提供されているparameter_estimator-3.0.0-cp311-cp311-linux_x86_64.whl
- 実行環境のOSに合わせてインストールするwheelパッケージを選択ください。
  - OSがRHEL 8.8/8.10またはRocky Linux 8.8/8.10の場合、rhel8フォルダ内のwheelパッケージ
  - OSがRHEL 9.2/9.4またはRocky Linux 9.2/9.4の場合、rhel9フォルダ内のwheelパッケージ

■ Python3.11，PyQUBO (バージョンは1.4.0以上)

- Python3.11が動作し、PyQUBOがインストールされていることが前提になります。
- PyQUBOのバージョンは1.4.0以降となります。
- **推奨はPyQUBO 1.4.0であり、Pythonは3.11のみとなります。**

Parameter Estimator のインストール

◆ インストール手順

■ pipコマンドでインストールします

- 以下のようにwheelパッケージを任意のフォルダ(***DIR***)に置いて、pipコマンドでインストールします。

```
$ pip3.11 install --user DIR/parameter_estimator-3.0.0-cp311-cp311-linux_x86_64.whl
```

  - ユーザの個人環境にインストールする場合は、「--user」オプションを指定してください。

- pipコマンドで、インストールされていることを確認できます。

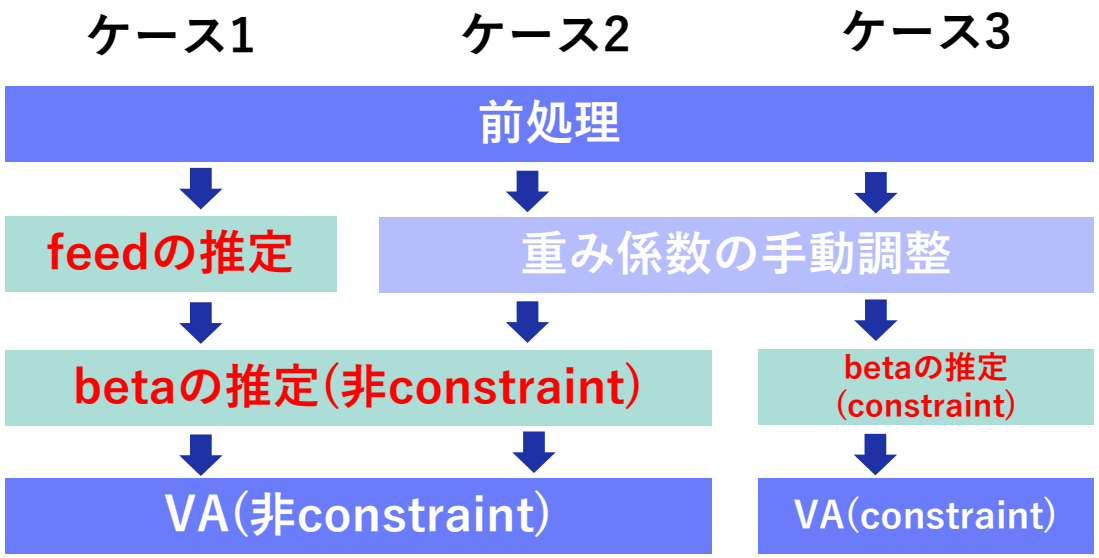
```
$ pip3.11 show parameter_estimator
Name: parameter-estimator
Version: 3.0.0   バージョンを確認
Summary: Parameter Estimator for NEC Vector Annealing
 .
 .
```

3.利用手順

利用手順

- 以下の3つのケースでの利用を想定しています。それぞれのケースにおける利用例を次頁以降に記します。

ケース	mode	betaの推定	feedの推定	備考
ケース1	非constraint	行う	行う	high_orderは非対応です。
ケース2	非constraint	行う	行わない	
ケース3	constraint	行う	行わない	constraint modeでfeedの推定はできません。



ケース1

- 非constraint modeでfeedとbetaを推定するケースです。以下の手順で利用します。
  - このケースで高次項を含むmodelを推定することはできません。

1. モジュールのimport
2. model作成
3. Estimatorインスタンスの作成
4. feedの推定
5. betaの推定
6. Annealingの実行

ケース1 - 前準備

- モジュールのインポート
  - parameter_estimator, pyqubo, VectorAnnealingをimportします。
    - インストールしたparameter_estimatorをimportしてください。

```
import VectorAnnealing
from parameter_estimator import parameter_estimator
from pyqubo import Array, Placeholder, Constraint
```

Array.createのvartype=‘BINARY‘のみに対応しています。
UnaryEncIntegerやLogEncInteger等の他には未対応です。

ケース1 - model作成

- spinを定義して、目的関数と制約条件を設定します。
  - 制約条件を制約毎にConstraint()で定義してください

    ※ 複数の制約を一括でラベル付けすると、正しく動作しません。

    ※ MIN(), MAX(), MINMAX(), FREE()は予約語のためラベル名に使用できません。

```
n = 5
x = Array.create('x', shape=n, vartype='BINARY')

# 目的関数Hwを定義する
Hw = sum(sum(x[i]*x[j] for i in range(n)) for j in range(n))

# 制約条件Hcを定義する
Hc = Constraint((sum(x[i] for i in range(n)) - 1) ** 2, label="onehot")

param_hw = Placeholder('hw')
param_hc = Placeholder("hc")
H = param_hw * Hw + param_hc * Hc
```

ケース1 - model作成

- フリップオプションの設定
  - VAのユーザーズガイドに従って各フリップオプションを設定します。
  - 例えばonehotとfixedを扱う場合は以下のようにconstflipにフリップオプションを設定してください。

```
onehot = [[f'x[{i}]' for i in range(n)]]
fixed = [['x[0]', 0]]

constflip = {"onehot":onehot, "fixed":fixed}
```

(参考)Paramter_Estimatorで対応しているフリップオプション一覧

  - フリップオプションのみに含まれるspinがある場合は、それらのspinをspin_listに追加する必要があります。

ケース1 - model作成

- model, quboの作成
  - 仮の重み係数をfeed_dictに指定して、ハミルトニアンからmodelとquboを作成して下さい。
    - Placeholder()で設定した各重み係数名をkeyに、正の実数をvalueとした辞書を指定してください。
  - 目的関数の重み係数は推定されませんので、指定した値に固定されます。
    (複数の目的関数を定義した場合は、各目的関数の比率を手動で調整してください)

```
model = H.compile()
feed_dict = {"hw":1, "hc":1}
qubo, offset = model.to_qubo(feed_dict=feed_dict)
```

ケース1 - model作成

- spin_listの作成
  - modelに含まれるspinの一覧を作成してください。
  - PyQUBOで作成したモデルで使用しているスピンの一覧はmodel.variablesに定義されていますので、それをコピーしてください。

```
spin_list = model.variables
```

  - フリップオプションのみに含まれるspinがある場合、手動で追加してください。

```
y = Array.create('y', shape=3, vartype='BINARY')
orone = ['x[0]', 'y[0]', 'y[1]', 'y[2]']
constflip["orone"] = [orone]

# orone flip optionに指定したy[0], y[1], y[2]をspin_listに追加する
spin_list = list(set(spin_list) | set(orone))
```

  - 余計なspinが含まれると誤った推定をする場合があります。

ケース1 - インスタンス作成

- Estimatorクラス のインスタンスを作成
  - 以下のようにしてインスタンスを作成してください。

```
#インスタンス生成
est = parameter_estimator.Estimator()
```

※ハミルトニアンを変更した場合、再度、インスタンスを作成し直す必要があります。
インスタンスを作成し直さない場合、誤った推定をする可能性があります。

ケース1 - 重み係数の推定

- Parameter Estimator で、重み係数を自動推定
  - get_feed()関数を用いて、重み係数(feed_dict)を推定します。
  - model.to_qubo()に指定した仮のfeed_dictと同じfeed_dictを引数に指定し、constflipに定義したフリップオプションを以下のように記述してください。
  - cal_all_feedをTrueにすると目的関数を考慮した推定をします。

```
#feedの推定
feed_dict = est.get_feed(Hc, feed_dict, spin_list, qubo, **constflip, cal_all_feed=True)

#quboの再計算
qubo, offset = model.to_qubo(feed_dict=feed_dict)
```

  - feed_dictを推定した後 、そのfeed_dictを用いてquboを再計算し更新して下さい(推定したfeed_dictが係数となったquboが作られます)。

ケース1 - ベータの推定

- Parameter Estimator で、ベータを自動推定
  - get_beta()関数またはget_beta_range()関数を用いて、ベータを自動推定します。
    - beta_rangeを推定する場合

```
beta_range = est.get_beta_range(Hc, feed_dict, spin_list, qubo, num_sweeps=1000, **constflip)
```

    - beta_listを推定する場合

```
beta_list = est.get_beta(Hc, feed_dict, spin_list, qubo, num_sweeps=1000, **constflip)
```

ケース1 - VAの実行

- 再計算したquboを用いてva_modelを計算し、va_samplerにva_modelとbetaを入力して、求解を行って下さい。
  - beta_listとbeta_rangeを間違えないように注意してください。
    - beta_listを推定する場合

```
feed_dict = est.get_feed(Hc, feed_dict, spin_list, qubo, **constflip, cal_all_feed=True)
qubo, offset = model.to_qubo(feed_dict=feed_dict)
beta_list = est.get_beta(Hc, feed_dict, spin_list, qubo, num_sweeps=1000, **constflip)

va_model = VectorAnnealing.model(qubo, offset, **constflip)
va       = VectorAnnealing.sampler()
results = va.sample(va_model,num_sweeps=1000,
beta_list=beta_list,vector_mode=VectorAnnealing.VECTOR_MODE_SPEED)
```

    - beta_rangeを推定する場合

```
feed_dict = est.get_feed(Hc, feed_dict, spin_list, qubo, **constflip, cal_all_feed=True)
qubo, offset = model.to_qubo(feed_dict=feed_dict)
beta_range = est.get_beta_range(Hc, feed_dict, spin_list, qubo, num_sweeps=1000, **constflip)

va_model = VectorAnnealing.model(qubo, offset, **constflip)
va       = VectorAnnealing.sampler()
results = va.sample(va_model, num_sweeps=1000, beta_range=beta_range,
vector_mode=VectorAnnealing.VECTOR_MODE_SPEED)
```

ケース1 - sampleコード

```
import VectorAnnealing
from parameter_estimator import parameter_estimator
from pyqubo import Array,Placeholder,Constraint


n= 5
x = Array.create('x', shape=n, vartype='BINARY')
Hw = sum(sum(x[i]*x[j] for i in range(n)) for j in range(n))
Hc = Constraint((sum(x[i] for i in range(1,n)) - 1) ** 2, label="onehot")
param_hw = Placeholder('hw')
param_hc = Placeholder("hc")
H = param_hw * Hw + param_hc * Hc

# Flip Option
onehot = [[f'x[{i}]' for i in range(1,n)]]
fixed = [['x[0]', 0]]
constflip = {"onehot":onehot, "fixed":fixed}

# model化
model = H.compile()
feed_dict = {"hw":1, "hc":1}
qubo, offset = model.to_qubo(feed_dict=feed_dict)
spin_list = model.variables

# paramter_estimator
est = parameter_estimator.Estimator()
feed_dict = est.get_feed(Hc, feed_dict, spin_list, qubo, **constflip, cal_all_feed=True)
qubo, offset = model.to_qubo(feed_dict=feed_dict)
beta_range = est.get_beta_range(Hc, feed_dict,spin_list, qubo, num_sweeps=1000, **constflip)

#VA
sa_model = VectorAnnealing.model(qubo, offset, **constflip)
sampler = VectorAnnealing.sampler()
results = sampler.sample(sa_model, num_sweeps=1000, beta_range=beta_range, vector_mode=VectorAnnealing.VECTOR_MODE_SPEED)
```

ケース2

- 非constraint modeでbetaを推定するケースです。以下の手順で利用します。
  - このケースでは高次項を含むmodelも推定可能です。

1. モジュールのimport
2. model作成
3. Estimatorインスタンスの作成
4. betaの推定
5. Annealingの実行

ケース2 - 前準備

- モジュールのインポート
  - parameter_estimator, pyqubo, VectorAnnealingをimportします。
    - インストールしたparameter_estimatorをimportしてください。

```
import VectorAnnealing
from parameter_estimator import parameter_estimator
from pyqubo import Array, Placeholder, Constraint
```

Array.createのvartype=‘BINARY‘のみに対応しています。
UnaryEncIntegerやLogEncInteger等の他には未対応です。

ケース2 - model作成

- spinを定義して、目的関数と制約条件を設定します。
  - 制約条件を制約毎にConstraint()で定義してください

※ 複数の制約を一括でラベル付けすると、正しく動作しません。

※ MIN(), MAX(), MINMAX(), FREE()は予約語のためラベル名に使用できません。

```
n = 5
x = Array.create('x', shape=n, vartype='BINARY')

# 目的関数Hwを定義する
Hw = sum(sum(x[i]*x[j] for i in range(n)) for j in range(n))

# 制約条件Hcを定義する
Hc = Constraint((sum(x[i] for i in range(n)) - 1) ** 2, label="onehot")

param_hw = Placeholder('hw')
param_hc = Placeholder("hc")
H = param_hw * Hw + param_hc * Hc
```

ケース2 - model作成

- フリップオプションの設定
  - VAのユーザーズガイドに従って各フリップオプションを設定します。
  - 例えばonehotとfixedを扱う場合は以下のようにconstflipにフリップオプションを設定してください。

```
onehot = [[f'x[{i}]' for i in range(n)]]
fixed = [['x[0]', 0]]

constflip = {"onehot":onehot, "fixed":fixed}
```

(参考)Paramter_Estimatorで対応しているフリップオプション一覧

  - フリップオプションのみに含まれるspinがある場合は、それらのspinをspin_listに追加する必要があります。

ケース2 - model作成

- model, quboの作成
  - 作成したハミルトニアンからmodelとquboを作成して下さい。
  - feed_dictを別途用意してください。

```
model = H.compile()
feed_dict = {"hw":2, "hc":1}
qubo, offset = model.to_qubo(feed_dict=feed_dict)
```

- 高次項の作成
  - betaのみを推定する場合高次項も指定可能です。(必須ではありません)

```
high_order = {('z[0]', 'z[1]', 'z[2]'):1, ('z[0]', 'z[1]', 'x[0]'):2}
```

  - 高次項のみに含まれるspinがある場合後でspin_listに追加する必要があります。

ケース2 - model作成

- spin_listの作成
  - modelに含まれるspinの一覧を作成してください。
  - PyQUBOで作成したモデルで使用しているスピンの一覧はmodel.variablesに定義されていますので、それをコピーしてください

```
spin_list = model.variables
```

  - flip optionや高次項にのみ含まれるspinがある場合、手動で追加してください。

```
y = Array.create('y', shape=3, vartype='BINARY')
high_order = {('z[0]', 'z[1]', 'z[2]'):1, ('z[0]', 'z[1]', 'x[0]'):2}
orone = ['x[0]', 'y[0]', 'y[1]', 'y[2]']
constflip["orone"] = [orone]

# y[0], y[1], y[2], z[0], z[1], z[2]を追加する
spin_list = list(set(spin_list) | set(orone) | {'z[0]', 'z[1]', 'z[2]'})
```

  - 余計なspinが含まれると誤った推定をする場合があります。

ケース2 - インスタンス作成

- Estimatorクラス のインスタンスを作成
  - 以下のようにしてインスタンスを作成してください。

```
#インスタンス生成
est = parameter_estimator.Estimator()
```

※ハミルトニアンを変更した場合、再度、インスタンスを作成し直す必要があります。
インスタンスを作成し直さない場合、誤った推定をする可能性があります。

ケース2 - ベータの推定

- Parameter Estimator で、ベータを自動推定
  - get_beta()関数またはget_beta_range()関数を用いて、ベータを自動推定します。
    - beta_rangeを推定する場合

```
beta_range = est.get_beta_range(Hc, feed_dict, spin_list, qubo, num_sweeps=1000,
high_order=high_order, **constflip)
```

    - beta_listを推定する場合

```
beta_list = est.get_beta(Hc, feed_dict, spin_list, qubo, num_sweeps=1000,
high_order=high_order,**constflip)
```

ケース2 - VAの実行

- va_modelを計算し、va_samplerにva_modelとbetaを入力して、求解を行って下さい。
  - beta_listとbeta_rangeを間違えないように注意してください。
    - beta_listを推定する場合

```
beta_list = est.get_beta(Hc, feed_dict, spin_list, qubo, num_sweeps=1000,
                         high_order=high_order, **constflip)

va_model = VectorAnnealing.model(qubo, offset, high_order=high_order, **constflip)
va       = VectorAnnealing.sampler()
results = va.sample(va_model, beta_list=beta_list, num_sweeps=1000,
          vector_mode=VectorAnnealing.VECTOR_MODE_SPEED)
```

    - beta_rangeを推定する場合

```
beta_range = est.get_beta_range(Hc, feed_dict, spin_list, qubo,
                                num_sweeps=1000,high_order=high_order,**constflip)

va_model = VectorAnnealing.model(qubo, offset, high_order=high_order, **constflip)
va       = VectorAnnealing.sampler()
results = va.sample(va_model, beta_range=beta_range, num_sweeps=1000,
                    vector_mode=VectorAnnealing.VECTOR_MODE_SPEED)
```

ケース2 - sampleコード

```
import VectorAnnealing
from parameter_estimator import parameter_estimator
from pyqubo import Array,Placeholder,Constraint


n= 5
x = Array.create('x', shape=n, vartype='BINARY')
Hw = sum(sum(x[i]*x[j] for i in range(n)) for j in range(n))
Hc = Constraint((sum(x[i] for i in range(1,n)) - 1) ** 2, label="onehot")
param_hw = Placeholder('hw')
param_hc = Placeholder("hc")
H = param_hw * Hw + param_hc * Hc

# Flip Option
onehot = [[f'x[{i}]' for i in range(1,n)]]
fixed = [['x[0]', 0]]
constflip = {"onehot" : onehot, "fixed":fixed}

# model化
model = H.compile()
feed_dict = {"hw":1, "hc":2}
qubo, offset = model.to_qubo(feed_dict=feed_dict)
z = Array.create('z', shape=3, vartype='BINARY')
high_order = {('z[0]', 'z[1]', 'z[2]'): 1, ('z[0]', 'z[1]', 'x[0]'):2}
spin_list = model.variables
spin_list = list(set(spin_list) |{'z[0]', 'z[1]', 'z[2]'})

# paramter_estimator
est = parameter_estimator.Estimator()
beta_range = est.get_beta_range(Hc, feed_dict,spin_list, qubo, num_sweeps=1000, high_order=high_order, **constflip)

#VA
sa_model = VectorAnnealing.model(qubo, offset, **constflip, high_order=high_order)
sampler = VectorAnnealing.sampler()
results = sampler.sample(sa_model, num_sweeps=1000, beta_range=beta_range, vector_mode=VectorAnnealing.VECTOR_MODE_SPEED)
```

ケース3

- constraint modeでbetaを推定するケースです。以下の手順で利用します。
  - このケースではfeedを推定できません (constraint modeでの求解には不要) 。
  - このケースでは高次項を含むmodelも推定可能です。

1. モジュールのimport
2. model作成
3. Estimatorインスタンスの作成
4. betaの推定
5. Annealingの実行

ケース3 - 前準備

- モジュールのインポート
  - parameter_estimator, pyqubo, VectorAnnealingをimportします。
    - インストールしたparameter_estimatorをimportしてください。

```
import VectorAnnealing
from parameter_estimator import parameter_estimator
from pyqubo import Array, Placeholder, Constraint
```

Array.createのvartype=‘BINARY‘のみに対応しています。
UnaryEncIntegerやLogEncInteger等の他には未対応です。

ケース3 - model作成

- spinを定義して、目的関数と制約条件を設定します。
  - 制約条件を制約毎にConstraint()で定義してください

    ※ 複数の制約を一括でラベル付けすると、正しく動作しません。

    ※ MIN(), MAX(), MINMAX(), FREE()は予約語のためラベル名に使用できません。

```
n = 5
x = Array.create('x', shape=n, vartype='BINARY')

# 目的関数Hwを定義する
Hw = sum(sum(x[i]*x[j] for i in range(n)) for j in range(n))

# 制約条件Hcを定義する
Hc = Constraint((sum(x[i] for i in range(n)) - 1) ** 2, label="onehot")

param_hw = Placeholder('hw')
param_hc = Placeholder("hc")
H = param_hw * Hw + param_hc * Hc
```

ケース3 - model作成

- フリップオプションの設定
  - VAのユーザーズガイドに従って各フリップオプションを設定します。
  - constraint mode専用のflip optionも設定できます。
  - 1つ以上のフリップオプションを設定する必要があります。
  - 例えばonehotとfixedとweighted_sum を扱う場合は以下のようにconstflipにフリップオプションを設定してください。

```
onehot = [[f'x[{i}]' for i in range(n)]]
fixed = [['x[0]', 0]]
weighted_sum = [{"comparison":("EQ", 3), "weight":{f'x[{i}]':(i+1) for i in range(n)}}]
constflip = {"onehot":onehot, "fixed":fixed, "weighted_sum":weighted_sum}
```

(参考)Paramter_Estimatorで対応しているフリップオプション一覧

  - flip optionのみに含まれるspinがある場合は、推定前にspin_listに追加する必要があります。

ケース3 - model作成

- model, quboの作成
  - 作成したハミルトニアンからmodelとquboを作成して下さい。
  - feed_dictを別途用意してください。

```
model = H.compile()
feed_dict = {"hw":2, "hc":1}
qubo, offset = model.to_qubo(feed_dict=feed_dict)
```

- 高次項の作成
  - betaのみを推定する場合高次項も指定可能です(必須ではありません)。

```
high_order = {('z[0]', 'z[1]', 'z[2]'):1, ('z[0]', 'z[1]', 'x[0]'):2}
```

  - 高次項のみに含まれるspinがある場合後でspin_listに追加する必要があります。

ケース3 - model作成

- spin_listの作成
  - modelに含まれるspinの一覧を作成してください。
  - PyQUBOで作成したモデルで使用しているスピンの一覧はmodel.variablesに定義されていますので、それをコピーしてください

```
spin_list = model.variables
```

  - flip optionや高次項にのみ含まれるspinがある場合、手動で追加してください。

```
y = Array.create('y', shape=3, vartype='BINARY')
high_order = {('z[0]', 'z[1]', 'z[2]'):1, ('z[0]', 'z[1]', 'x[0]'):2}
orone = ['x[0]', 'y[0]', 'y[1]', 'y[2]']
constflip["orone"] = [orone]

# y[0], y[1], y[2], z[0], z[1], z[2]を追加する
spin_list = list(set(spin_list) | set(orone) | {'z[0]', 'z[1]', 'z[2]'})
```

  - 余計なspinが含まれると誤った推定をする場合があります。

ケース3 - インスタンス作成

- Estimatorクラス のインスタンスを作成
  - 以下のようにしてインスタンスを作成してください。

```
#インスタンス生成
est = parameter_estimator.Estimator()
```

※ハミルトニアンを変更した場合、再度、インスタンスを作成し直す必要があります。
インスタンスを作成し直さない場合、誤った推定をする可能性があります。

ケース3 - ベータの推定

- Parameter Estimator で、ベータを自動推定
  - get_beta()関数またはget_beta_range()関数を用いて、ベータを自動推定します。
  - Hとfeedは不要です。
  - 1つ以上のフリップオプションを指定する必要があります。
    - beta_rangeを推定する場合

```
beta_range = est.get_constraint_beta_range(spin_list, qubo, num_sweeps=1000,
                                           high_order=high_order, **constflip)
```

    - beta_listを推定する場合

```
beta_list = est.get_constraint_beta(spin_list, qubo, num_sweeps=1000,
high_order=high_order,**constflip)
```

ケース3 - VAの実行

- va_modelを計算し、va_samplerにva_modelとbetaを入力して、求解を行って下さい。
  - beta_listとbeta_rangeを間違えないように注意してください。
    - beta_listを推定する場合

```
beta_list = est.get_constraint_beta(spin_list, qubo, num_sweeps=1000,
high_order=high_order,**constflip)

va_model = VectorAnnealing.model(qubo, offset, high_order=high_order, **constflip)
va       = VectorAnnealing.sampler()
results = va.sample(va_model, beta_list=beta_list, num_sweeps=1000,
                    vector_mode=VectorAnnealing.VECTOR_MODE_CONSTRAINT)
```

    - beta_rangeを推定する場合

```
beta_range = est.get_constraint_beta_range(spin_list, qubo, num_sweeps=1000,
                                           high_order=high_order,
                                           high_order=high_order, **constflip)

va_model = VectorAnnealing.model(qubo, offset, high_order=high_order, **constflip)
va       = VectorAnnealing.sampler()
results = va.sample(va_model, beta_range=beta_range, num_sweeps=1000,
                    vector_mode=VectorAnnealing.VECTOR_MODE_CONSTRAINT)
```

ケース3 - sampleコード

```
import VectorAnnealing
from parameter_estimator import parameter_estimator
from pyqubo import Array,Placeholder,Constraint

n= 5
x = Array.create('x', shape=n, vartype='BINARY')
Hw = sum(sum(x[i]*x[j] for i in range(n)) for j in range(n))
Hc = Constraint((sum(x[i] for i in range(n)) - 1) ** 2, label="onehot")
param_hw = Placeholder('hw')
param_hc = Placeholder("hc")
H = param_hw * Hw + param_hc * Hc

# Flip Option
onehot = [[f'x[{i}]' for i in range(n)]]
fixed = [['x[0]',0]]
weighted_sum = [{"comparison":("EQ",3), "weight":{f'x[{i}]':(i+1) for i in range(n)}}]
constflip = {"onehot":onehot, "fixed":fixed, "weighted_sum":weighted_sum}

# model化
model = H.compile()
feed_dict = {"hw":1, "hc":2}
qubo, offset = model.to_qubo(feed_dict=feed_dict)
z = Array.create('z', shape=3, vartype='BINARY')
high_order = {('z[0]', 'z[1]', 'z[2]'):1, ('z[0]', 'z[1]', 'x[0]'):2}
spin_list = model.variables
spin_list = list(set(spin_list) |{'z[0]', 'z[1]', 'z[2]'})

# paramter_estimator
est = parameter_estimator.Estimator()
beta_range = est.get_constraint_beta_range(spin_list, qubo, num_sweeps=1000, high_order=high_order, **constflip)

#VA
sa_model = VectorAnnealing.model(qubo, offset, **constflip, high_order=high_order)
sampler = VectorAnnealing.sampler()
results = sampler.sample(sa_model, num_sweeps=1000, beta_range=beta_range, vector_mode=VectorAnnealing.VECTOR_MODE_CONSTRAINT)
```

4.仕様

Estimator クラス

- Parameter Estimatorを使用する場合は、Estimatorクラスのインスタンスを作成します。
- Estimatorインスタンスの初期化時には以下の引数を設定できます。

引数	必須/任意	説明
threads	任意	並列計算のthread数を指定する。 defaultは論理コア数の半分(切り上げ)。

- parameterを推定する場合、Estimatorインスタンスのメソッドを実行してください。

API仕様- get_feed()

- get_feed()は以下の引数を受け付けます。

引数	必須/任意	説明
H	必須	ハミルトニアン (目的関数を省略し、制約条件のみを指定すると高速化します。目的関数のみの場合は推定しないため、get_feedは使用しないでください。)
feed	必須	手動で用意した目的関数/制約条件の値（正の値）
spin_list	必須	H、flip option, high_orderで使用しているスピンのリスト。PyQUBOのmodel.variablesを指定する。ただし、flip optionのみに含まれるspinがあれば、PyQUBOのmodel.variablesにそれらのspinを追加して指定する。
qubo	必須	PyQUBOのコンパイル機能で出力したQUBO
**flip option	任意	各種制約フリップオプション。指定できるflip optionは後述します。 スピン変数は文字列で指定してください。constflip = {"onehot": [['x[0]', 'x[1]']], "fixed": [['x[0', 0]]}のように辞書で定義し、**constflipとパックした形で指定することを推奨します。
high_order	任意	get_feed()では未対応です。引数としては存在していますが、指定するとerrorとなります。
cal_all_feed	任意	重み係数の推定に目的関数を考慮するかを指定するパラメータ。 ・Trueの場合Constraint()を付けていない項も考慮した重み係数を計算します。 ・Falseの場合Constraint()を付けていない項を無視した重み係数を計算します。 defaultはFalse

- 重み係数推定機能はconstraint modeには対応しません。
- 重み係数の調整は複数の重み係数を比較し相対的に行います。そのためcal_all_feed=Falseで目的関数を考慮せず、制約条件の重み係数が1つのみになってしまう場合は、重み係数の調整ができず、制約を満たせない可能性があります。

API仕様- get_beta()

- get_beta()は以下の引数を受け付けます。

引数	必須/任意	説明
H	必須	ハミルトニアン (目的関数を省略し、制約条件のみを指定すると高速化します。ただし、目的関数のみのケースでは省略できません。)
feed	必須	手動で用意した目的関数/制約条件の値（正の値）
spin_list	必須	H、flip option, high_orderで使用しているスピンのリスト。PyQUBOのmodel.variablesを指定する。ただし、flip optionやhigh_orderのみに含まれるspinがあれば、PyQUBOのmodel.variablesにそれらのspinを追加して指定する。
qubo	必須	PyQUBOのコンパイル機能で出力したQUBO
sweeps	任意	VectorAnnealingのスイープ数
**flip option	任意	各種制約フリップオプション。指定できるflip optionは後述します。 スピン変数は文字列で指定してください。constflip = {"onehot": [['x[0]', 'x[1]']], "fixed": [['x[0', 0]]}のように辞書で定義し、**constflipとパックした形で指定することを推奨します。
high_order	任意	高次項

API仕様- get_beta_range()

- get_beta_range()は以下の引数を受け付けます。

引数	必須/任意	説明
H	必須	ハミルトニアン (目的関数を省略し、制約条件のみを指定すると高速化します。ただし、目的関数のみのケースでは省略できません。)
feed	必須	手動で用意した目的関数/制約条件の値（正の値）
spin_list	必須	H、flip option, high_orderで使用しているスピンのリスト。PyQUBOのmodel.variablesを指定する。ただし、flip optionやhigh_orderのみに含まれるspinがあれば、PyQUBOのmodel.variablesにそれらのspinを追加して指定する。
qubo	必須	PyQUBOのコンパイル機能で出力したQUBO
sweeps	任意	VectorAnnealingのスイープ数
**flip option	任意	各種制約フリップオプション。指定できるflip optionは後述します。 スピン変数は文字列で指定してください。constflip = {"onehot": [['x[0]', 'x[1]']], "fixed": [['x[0', 0]]}のように辞書で定義し、**constflipとパックした形で指定することを推奨します。
high_order	任意	高次項
sweep_step_rate	任意	sweep数に対するbeta_rangeのstep数の比です。defaultは0.8

API仕様 - get_constaint_beta()

- get_constraint_beta()は以下の引数を受け付けます。

引数	必須/任意	説明
spin_list	必須	flip option, high_orderで使用しているスピンのリスト。PyQUBOのmodel.variablesを指定する。ただし、flip optionやhigh_orderのみに含まれるspinがあれば、PyQUBOのmodel.variablesにそれらのspinを追加して指定する。
qubo	必須	PyQUBOのコンパイル機能で出力したQUBO
sweeps	任意	VectorAnnealingのスイープ数
**flip option	任意	各種制約フリップオプション。 constraint mode専用のflip optionも指定することが出来ます。 指定できるflip optionは後述します。 スピン変数は文字列で指定してください。constflip = {"onehot": [['x[0]', 'x[1]']], "fixed": [['x[0', 0]]}のように辞書で定義し、**constflipとパックした形で指定することを推奨します。
high_order	任意	高次項

API仕様- get_constraint_beta_range()

- get_constraint_beta_range()は以下の引数を受け付けます。

引数	必須/任意	説明
spin_list	必須	flip option, high_orderで使用しているスピンのリスト。PyQUBOのmodel.variablesを指定する。ただし、flip optionやhigh_orderのみに含まれるspinがあれば、PyQUBOのmodel.variablesにそれらのspinを追加して指定する。
qubo	必須	PyQUBOのコンパイル機能で出力したQUBO
sweeps	任意	VectorAnnealingのスイープ数
**flip option	任意	各種制約フリップオプション。 constraint mode専用のflip optionも指定することが出来ます。 指定できるflip optionは後述します。 スピン変数は文字列で指定してください。constflip = {"onehot": [['x[0]', 'x[1]']], "fixed": [['x[0', 0]]}のように辞書で定義し、**constflipとパックした形で指定することを推奨します。
high_order	任意	高次項
sweep_step_rate	任意	sweep数に対するbeta_rangeのstep数の比です。defaultは0.8

flip option

- 指定できるflip optionは以下です(詳細はVAのユーザーズガイドを参照)。
  - constraintの是非に関わらず使用できるflip optionは以下となります。

引数	必須/任意	説明
onehot	任意	onehot制約フリップオプション
fixed	任意	fixed制約フリップオプション
andzero	任意	andzero制約フリップオプション
orone	任意	orone制約フリップオプション
supplement	任意	supplement制約フリップオプション
maxone	任意	maxone制約フリップオプション
minmaxone	任意	minmaxone制約フリップオプション(conditionを設定する場合はconstraint modeのみ)

  - constraint modeのみで使用できるflip optionは以下となります。

引数	必須/任意	説明
minmaxone	任意	minmaxone制約フリップオプション(conditionも設定可)
pattern	任意	pattern制約フリップオプション
weighted_sum	任意	weighted_sum制約フリップオプション

5.注意事項

使用時の注意事項 - 全般

◆ 以下、注意事項になります

■ 制約条件は、複数の制約をまとめて1つのラベル付けにしてはいけません。

- 以下のようにまとめて定義すると、正しい推定結果が取得できません。

```
Ha = param_time * Constraint(sum((sum(x[i, j] for i in range(point_num)) - 1) ** 2
                                 for j in range(point_num)), label="onehot1")
```

■ 式に係数0の項が含まれた場合、想定と異なるparameterを推定する場合があります。

使用時の注意事項 - 全般

- 以下、注意事項になります
  - ハミルトニアンやPlaceholderに、“nan”と“inf”の変数を使用すると動作を保証できませんので、使用しないようご注意をお願いいたします。
  - 本ツール使用時、サーバの論理コア数の半分を使用することに注意してください。

使用時の注意事項 - 全般

◆ 以下、注意事項になります

■ PyQUBO 1.4.0では以下のような問題が確認されていますので、旧バージョン時に作成したコードを利用される場合は注意してください。

- Placeholderの積が有る項と無い項の和を求めると、TypeError("__radd__(): incompatible function arguments.")例外を発行するため、以下のように記述してください。

–例外が発生するコード

```
HW += sum((dlength[j][k] - d_min[j]) * x[i, j] * x[(i+1)%point_num, k] for i in range(point_num))
H = param_hw * HW + HC
```

–正しく動作するコード

```
HW += param_hw * sum((dlength[j][k] - d_min[j]) * x[i, j] * x[(i+1)%point_num, k] for i in range(point_num))
H = HW + HC
```

使用時の注意事項 - 全般

- 以下、注意事項になります
  - 制約式の書き方で、Parameter Estimatorの実行時間が変わります。
    - 制約項(feed_dict): feed1, feed2
    - 制約式(Constraint関数): const1, const2, const3

① 実行時間が速くなる例

```
HC1 = feed1*(const1 + const2 + const3)
HC2 = feed2*(const4 + const5 + const6)
HC = HC1 + HC2
```

② 実行時間が遅くなる例

```
HC1 = feed1*const1 + feed1*const2 + feed1*const3
HC2 = feed2*const4 + feed2*const5 + feed2*const6
HC = HC1 + HC2
```

- ①の方法だと、constそれぞれに同じfeedを積算することを複数のCPUで並列化できるため、時間を短縮できます。(const1+const2+const3) を並列計算できるためです。
- ②の方法だと、constに逐一、feedを積算しているため、1つのCPUでしか計算できず並列化できません。feed1*const1, feed1*const2 …、それぞれ1並列となるためです。

①の方法で制約式を立てることを推奨します。

使用時の注意事項 - 全般

- 以下、注意事項になります
  - Constraint()のlabelの指定について
    - 以下のような場合、labelに予約語を用いた指定が必要なケースがあります。

条件	予約語
範囲指定のある制約をConstraintに定義する場合	MIN()、MAX()、MINMAX()
補助スピンのある制約をConstraintに定義する場合	FREE()

使用時の注意事項 - 全般

- 以下、注意事項になります
  - 範囲指定のある制約をConstraintに定義する場合
    - 解く問題によっては、ベータを推定するときにConstraintの式に**1のスピン数の合計の範囲**を指定する必要があるケースが存在します。labelにMIN()またはMAX() を指定することで、Constraint()の値が0以外も許可するようになります。
    - 以下のシフト問題のサンプルコードは、従業員の人数を目標値に近づけるための制約条件の式に対して、labelのMIN({}), MAX({})に、1のスピン数の合計の最小, 最大の場合のエネルギー指定することで、制約を満たすエネルギーの範囲を指定しています。
    - 最小値と最大値が同じ場合にはMINMAX({})として設定することも可能です。

```
target_n = TARGET_WORK / float(D)                          # 目標値
energy_min = (int(target_n - 0.5) - target_n) ** 2         # 1のスピン数の最小
energy_max = (int(target_n + 1.5) - target_n) ** 2         # 1のスピン数の最大

Hc = (sum(Constraint((sum(spin_x[i,j] for i in range(N)) - target_n) ** 2,
                     label="employeers_in_day_{}_MIN({})_MAX({})".format(j, energy_min, energy_max))
          for j in range(D)) / (scale_n * D))
```

使用時の注意事項 - 全般

- 以下、注意事項になります
  - 補助スピンのある制約をConstraintに定義する場合
    - labelにFREE()を指定することで、Constraint()の値が0以外の場合でも指定したスピンがフリップしたら制約を満たす場合、制約違反にしないようになります。
    - 補助スピンが複数ある場合は"H1_FREE({})_FREE({})"などの形で追加してください。

```
H1_1 = y[0] * x[2]
H1_2 = 3 * y[0] + x[0] * x[1] - 2 * y[0] * x[0] - 2 * y[0] * x[1]
H1 += Constraint(H1_1 + H1_2, label="H1_FREE({})".format(y[0]))
```

使用時の注意事項 - 全般

- 以下、注意事項になります
  - Constraint()の数が多い(制約の数が多い)場合、pyqubo::compile()実行時にセグメンテーション違反が発生することがあります（※1）
    - 制約数が多い場合はpyqubo::compile()の入力に利用する定式とparameter_estimatorの引数に入力する定式の変数を分けてください。

```
# parameter_estimator用の変数
H = 0

#コンパイル用の変数
H_compile = 0
for k in range(N):
    for num,(i,j) in enumerate(M):
        # parameter_estimatorの引数に入力する定式はConstraintを利用
        H += Constraint(x[i,k] * x[j,k], f"H_{k}_{num}")
        # pyqubo::compile()の入力に利用する定式はConstraintを除きます
        H_compile += x[i,k] * x[j,k]
・・・
# コンパイル時はコンパイル用の変数を利用
Model = H_compile.compile()
・・・
# parameter_estimator利用時はConstraintを使った変数を利用
beta_list = est.get_beta(H, feed_dict, spin_list, qubo, sweeps, **constflip)
```

※1：制約が約60万個以上存在するケースで問題が発生することを確認

使用時の注意事項 - 全般

- 以下、注意事項になります
  - 大規模な問題や結合密度高い問題の推定を行う場合、メモリ不足で以下に示すエラーが発生し、 各推定APIを実行できないケースがあります。
    - C++ (Pybind)でメモリ不足が発生した場合
      - parameter_estimatorのc++で記述された処理で必要なメモリ量を確保できない場合、以下のエラーが発生することがあります。

```
terminate called without an active exception
```

    - Python側でメモリ不足が発生した場合
      - parameter_estimatorのPythonで記述された処理で必要なメモリ量を確保できない場合、以下のエラーが発生することがあります。

```
OSError: [Errno 12] Cannot allocate memory
```

      - get_beta()で呼び出したparameter_estimatorインスタンスをdelしてから、gc.collect()で未開放のobjectのメモリを解放することで、動作できるようになることがあります。

使用時の注意事項 - ベータ

- 以下、注意事項になります
  - beta_listとbeta_rangeを同時に指定することはできません。どちらの方が求解精度が良いかは問題に依存しているため、まずbeta_listを試し、精度が悪い場合、beta_rangeを使用してみることを推奨します。
  - get_beta()で推定したbeta_listの最初の値とget_beta_range()で推定したbeta_rangeの開始値(1番目の値)は一致するとは限りません。
  - get_beta()とget_beta_range()の戻り値はどちらもlist型です。get_beta_range()で推定したbeta_rangeを誤ってbeta_listに指定しても想定外のbetaのまま動作します。どちらのbetaを使っているか注意してください。

```
# 間違った記述例
beta_range = est.get_beta_range(HC, feed_dict, spin_list, qubo, sweeps, **constflip)
results = va.sample(va_model, sweeps, beta_list=beta_range)
```

使用時の注意事項 - feed

- 以下、注意事項になります
  - 制約条件が存在しない式を指定した場合、推定を行わず入力した初期の重み係数を返します。
  - QUBOの作成に使用するfeed_dictと、get_feed()の引数に指定するfeed_dictは同じものを使用して下さい。
  - 目的関数にも重み係数を設定してください。
  - デフォルトの設定では、制約条件以外の項は無視されます。目的関数を考慮した推定を行いたい場合は、cal_all_flip=Trueを指定してください。

発行履歴

◆発行履歴一覧

2022年	7月	初版
2022年	11月	2版
2023年	11月	3版
2024年	1月	4版
2024年	11月	5版
2025年	11月	6版

◆追加・変更点詳細

初版	新規作成
2版	PyQUBOのバージョンの注意事項を追記
3版	get_bata_range()とget_feed()の追加に伴う変更
4版	ベクトルホストの記述および記述ミスを変更
5版	constraint modeへの対応/high_order対応に伴う変更・記載の整理
6版	サンプルコードの記述を修正

Parameter Estimator
ユーザーズガイド

2024年　11月　 5版

日本電気株式会社
東京都港区芝五丁目７番１号
TEL(03)3454-1111（大代表）


Orchestrating a brighter world
NEC