
Nulavis

リリース 1.0.0

NTT DATA Mathematical Systems Inc.

2026 年 09 月 18 日

目次

第 1 章 概要	1
1.1 概要	1
1.2 インストール	2
1.3 API リファレンス	5
1.4 Rosenbrock 関数の最適化	8
索引	15

第1章 概要

Nulavis (ニュラビス) は、CMA-ES (Covariance Matrix Adaptation Evolution Strategy) を用いたブラックボックス最適化を Python から実行するためのモジュールです。

目的関数の微分情報を必要とせず、目的関数の値を評価するだけで連続変数の最適化問題を解くことができます。

本マニュアルでは、本モジュールのインストール方法、基本的な使用方法、各種パラメータおよび使用例について説明します。

1.1 概要

1.1.1 CMA-ES とは

CMA-ES (Covariance Matrix Adaptation Evolution Strategy) は、連続変数を対象としたブラックボックス最適化アルゴリズムです。

目的関数の勾配を必要としないため、以下のような問題に適用できます。

- 目的関数の微分が困難な問題
- 数値シミュレーションを含む問題
- 複雑な非線形性を持つ問題
- 多峰性を持つ問題
- 目的関数がブラックボックスとして与えられる問題

本モジュールでは、C++で実装された CMA-ES 計算ルーチンを Python から利用できるインターフェースを提供します。

1.1.2 主な特徴

本モジュールでは、以下の機能を利用できます。

- CMA-ES によるブラックボックス最適化
- 変数ごとの上下限の指定
- 最小化・最大化の切り替え
- 初期解の指定
- 集団サイズの指定
- 重み付き親個体数の指定

- 最大反復回数の指定
- 最大計算時間の指定
- 乱数シードの指定
- 最適化過程のログ出力間隔の指定
- 配列形式および辞書形式による変数指定

1.1.3 最適化問題

本モジュールでは、次の形式の最適化問題を扱います。

$$\text{minimize}_{\mathbf{x}} f(\mathbf{x})$$

または、

$$\text{maximize}_{\mathbf{x}} f(\mathbf{x})$$

ここで、 f はユーザーが定義する目的関数です。

目的関数は Python の関数として指定します。

1.2 インストール

1.2.1 動作環境

Nulavis 1.0.0 は以下の環境で動作します。

- OS
 - Windows : Windows 11, これら OS の Server 版
- Python
 - 3.11, 3.12, 3.13, 3.14 (いずれも 64bit のみ)
 - 依存モジュール・ライブラリ
 - * `numpy`
 - * Microsoft Visual C++ 再頒布可能パッケージ
- Nuorium Optimizer : V28 開発版のライセンスが有効である環境

注釈

Microsoft Visual C++ 再頒布可能パッケージをインストールするために、Visual Studio 本体をインストールする必要はありません。

1.2.2 Microsoft Visual C++ 再頒布可能パッケージのインストール

本モジュールを使用する前に、Microsoft Visual C++ 再頒布可能パッケージをインストールしてください。

Microsoft の公式ダウンロードページから最新版をダウンロードできます。

サポートされている最新の [Visual C++ 再頒布可能パッケージのダウンロード | Microsoft Learn](#)

上記のリンクの「サポートされている最新の再頒布可能パッケージ バージョン」から X64 アーキテクチャのインストーラーをダウンロードしてください。

インストール手順

1. 上記のダウンロードページからインストーラーをダウンロードします。
2. ダウンロードした `vc_redist.x64.exe` を実行します。
3. 使用許諾契約を確認し、同意します。
4. **Install** ボタンをクリックします。
5. インストールが完了したら、必要に応じて Windows を再起動します。

1.2.3 Python のインストール

Python がインストールされていない場合は、Python をインストールしてください。

Python の公式 Web サイトからダウンロードできます。

[Python 公式ダウンロードページ](#)

注釈

本モジュールが対応している Python のバージョンについては、動作環境 を参照してください。

1.2.4 本モジュールのインストール

必要なソフトウェアをインストールした後、Python の `pip` コマンドを使用して本モジュールをインストールします。

```
pip install nulavis-1.0.0-cp***-cp***-win_amd64.whl
```

注釈

上記の `.whl` ファイル名はインストールする Python のバージョンにより異なります。ご利用になる Python のバージョンに合わせた `.whl` ファイルをお選びください。

1.2.5 インストールの確認

インストールが正常に完了したことを確認するため、Python から `nulavis` をインポートします。

```
import nulavis
```

エラーが発生しなければ、インストールは完了です。

さらに、`run()` が利用可能であることを確認できます。

```
import nulavis

print(nulavis.cmaes.run)
```

以下のように表示されれば、正常にインストールされています。

```
<cyfunction run at 0x...>
```

1.2.6 トラブルシューティング

ImportError が発生する場合

`nulavis` のインポート時に、DLL に関するエラーが発生する場合は、Microsoft Visual C++ 再頒布可能パッケージがインストールされているか確認してください。

例えば、以下のようなエラーが発生する場合があります。

```
ImportError: DLL load failed
```

この場合は、Microsoft Visual C++ 再頒布可能パッケージを最新版に更新した上で、再度インポートを試してください。

Microsoft Visual C++ 再頒布可能パッケージは、Visual Studio でビルドされた C/C++アプリケーションの実行に必要なランタイムコンポーネントを提供します。

警告

Visual C++ Redistributable は、必ず Microsoft の公式サイトからダウンロードしてください。

1.2.7 インストール手順のまとめ

インストールは以下の手順で行います。

1. Microsoft Visual C++ 再頒布可能パッケージをインストール
↓
2. Python をインストール
↓
3. pip で `nulavis` をインストール

(次のページに続く)

(前のページからの続き)

↓

4. `import nulavis` で動作確認

これで、`nulvais` を使用する準備が整います。

1.3 API リファレンス

1.3.1 `nulavis`

1.3.2 `run`

`nulavis.cmaes.run(func, n, param)`

CMA-ES によるブラックボックス最適化を実行します。

パラメータ

- **func** (*callable*) -- 最適化する目的関数です。`xinit` および `bound` をリスト形式で指定した場合は、目的関数に変数をリストとして渡します。辞書形式で指定した場合は、変数名をキーとする辞書を渡します。

目的関数は、変数を引数として受け取り、目的関数値を返す `callable` である必要があります。

- **n** (*int*) -- 最適化する変数の数です。`xinit` および `bound` の要素数と一致している必要があります。
- **param** (*dict*) -- 最適化に使用するパラメータを指定する辞書です。

xinit

[list or dict] 最適化の初期点です。必須パラメータです。

リスト形式の場合、`n` 個の `float` を指定します。

例:

```
"xinit": [1.0, 2.0, 3.0]
```

辞書形式の場合、変数名をキーとして初期値を指定します。

例:

```
"xinit": {
    "x": 1.0,
    "y": 2.0,
    "z": 3.0,
}
```

要素数は `n` と一致している必要があります。

bound

[list or dict] 各変数の探索範囲です。必須パラメータです。

リスト形式の場合、各変数について [下限, 上限] の形式で指定します。下限および上限は float である必要があります。

例:

```
"bound": [  
    [-5.0, 5.0],  
    [-10.0, 10.0],  
    [0.0, 100.0],  
]
```

辞書形式の場合、`xinit` と同じキーを使用して指定します。

例:

```
"bound": {  
    "x": [-5.0, 5.0],  
    "y": [-10.0, 10.0],  
    "z": [0.0, 100.0],  
}
```

下限は上限以下である必要があります。また、`xinit` の各値は対応する上下限の範囲内である必要があります。

lambda

[int, optional] CMA-ES の集団サイズです。1 以上の整数を指定します。

mu

[int, optional] CMA-ES で選択する親個体数です。1 以上の整数を指定します。

sigma

[float, optional] 初期ステップサイズです。0 より大きい float を指定します。

C

[list of float, optional] 初期共分散行列です。行列を 1 次元のリストとして指定します。

要素数は $n * n$ である必要があります、各要素は float である必要があります。

例えば、2 変数の場合は以下のように指定します。

```
"C": [  
    1.0, 0.0,  
    0.0, 1.0,  
]
```

max_iter

[int, optional] 最適化の最大反復回数です。0 以上の整数を指定します。

max_time

[int, optional] 最適化の最大計算時間です。0 以上の整数を指定します。

log_interval

[int, optional] ログを出力する間隔です。1 以上の整数を指定します。

seed

[int, optional] 乱数シードです。0 以上の整数を指定します。

silent

[bool, optional] ログ出力を抑制するかどうかを指定します。

True の場合はログ出力を抑制し、False の場合はログを出力します。

direction

[str, optional] 最適化方向です。

"minimize" を指定すると目的関数を最小化し、"maximize" を指定すると目的関数を最大化します。

大文字・小文字は区別されません。例えば、"MINIMIZE" も指定できます。

戻り値

result -- 最適化結果を格納した辞書です。

best_x

[list or dict] 得られた最良解です。xinit がリスト形式の場合はリスト、辞書形式の場合は xinit と同じキーを持つ辞書として返されます。

best_f

[float] 最良解における目的関数値です。

best_penalty

[float] 最良解における制約違反に対するペナルティ値です。

eval_count

[int] 目的関数の評価回数です。

generation

[int] 最適化終了時の世代数です。

message

[str] 最適化計算を終了した理由を記載したメッセージです。

sigma

[float] 最適化終了時のステップサイズです。

C

最適化終了時の共分散行列です。NumPy 配列として返されます。

status

[int] 最適化計算の終了ステータスです。

time

[float] 最適化計算にかかった計算時間 (秒) です。

戻り値の型

dict

例外

- **TypeError** -- 引数またはパラメータの型が不正な場合に発生します。
- **ValueError** -- `param` が辞書型でない場合、または初期値が指定された上下限の範囲外の場合に発生します。
- **KeyError** -- 必須パラメータが指定されていない場合、または `xinit` と `bound` のキーが一致しない場合に発生します。
- **RuntimeError** -- `xinit` または `bound` の要素数が `n` と一致しない場合に発生します。

1.4 Rosenbrock 関数の最適化

このチュートリアルでは、Rosenbrock 関数を例として、CMA-ES を用いた最適化の基本的な方法を説明します。

1.4.1 Rosenbrock 関数とは

Rosenbrock 関数は、最適化アルゴリズムの性能評価によく使用されるベンチマーク関数です。

2 変数の場合、Rosenbrock 関数は次式で定義されます。

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

この関数の大域的最適解は、

$$(x_1, x_2) = (1, 1)$$

であり、そのときの目的関数値は、

$$f(1, 1) = 0$$

となります。

Rosenbrock 関数は、最適解の周辺に細長く湾曲した谷を持つため、単純な最適化手法では最適解を見つけることが難しい問題として知られています。

1.4.2 目的関数を定義する

まず、Rosenbrock 関数を Python の関数として定義します。

```
def rosenbrock(x):  
    return (  
        (1.0 - x[0]) ** 2  
        + 100.0 * (x[1] - x[0] ** 2) ** 2  
    )
```

`rosenbrock()` は、2つの要素を持つリストを引数として受け取り、Rosenbrock 関数の値を返します。

例えば、

```
print(rosenbrock([1.0, 1.0]))
```

を実行すると、

```
0.0
```

と表示されます。

1.4.3 初期値と探索範囲を設定する

次に、CMA-ES に渡すパラメータを設定します。

ここでは、初期点を $(-1.0, 1.0)$ とし、各変数の探索範囲を $[-5.0, 5.0]$ とします。

```
param = {
    "xinit": [-1.0, 1.0],
    "bound": [
        [-5.0, 5.0],
        [-5.0, 5.0],
    ],
}
```

`xinit` は最適化の初期点を指定するパラメータです。

`bound` は各変数の探索範囲を指定します。

この例では、

```
x[0] : -5.0 ~ 5.0
x[1] : -5.0 ~ 5.0
```

となります。

注釈

`xinit` と `bound` に指定する数値は `float` である必要があります。そのため、`-1` や `5` ではなく、`-1.0` や `5.0` と指定します。

1.4.4 最適化を実行する

目的関数とパラメータを用意したら、`nulavis.cmaes.run()` を呼び出して最適化を実行します。

```
import nulavis

result = nulavis.cmaes.run(
    rosenbrock,
    2,
```

(次のページに続く)

(前のページからの続き)

```
    param,
)
```

`run()` の引数は次の 3 つです。

引数	説明
<code>rosenbrock</code>	最適化する目的関数
<code>2</code>	最適化する変数の数
<code>param</code>	CMA-ES のパラメータ

1.4.5 最適化結果を確認する

`run()` の戻り値には、最適化によって得られた結果が辞書形式で格納されています。

最良解は `best_x` から取得できます。

```
print("最適解:", result["best_x"])
```

最良目的関数値は `best_f` から取得できます。

```
print("目的関数値:", result["best_f"])
```

実行すると、例えば次のような結果が得られます。

```
最適解: [1.000..., 1.000...]
目的関数値: 0.000...
```

CMA-ES は乱数を使用するため、実行するたびに結果が多少異なる場合があります。

1.4.6 完全なプログラム

ここまでの内容をまとめると、プログラム全体は次のようになります。

```
import nulavis

def rosenbrock(x):
    return (
        (1.0 - x[0]) ** 2
        + 100.0 * (x[1] - x[0] ** 2) ** 2
    )

param = {
    "xinit": [-1.0, 1.0],
```

(次のページに続く)

(前のページからの続き)

```

    "bound": [
        [-5.0, 5.0],
        [-5.0, 5.0],
    ],
}

result = nulavis.cmaes.run(
    rosenbrock,
    2,
    param,
)

print("最適解:", result["best_x"])
print("目的関数値:", result["best_f"])

```

このプログラムを実行すると、Rosenbrock 関数の最適解である

$$(x_1, x_2) = (1, 1)$$

に近い解が得られます。

1.4.7 乱数シードを指定する

最適化結果を再現したい場合は、`seed` を指定します。

```

param = {
    "xinit": [-1.0, 1.0],
    "bound": [
        [-5.0, 5.0],
        [-5.0, 5.0],
    ],
    "seed": 12345,
}

```

同じ環境・同じパラメータで実行する場合、同じ乱数系列を使用して最適化を実行できます。

1.4.8 最大反復回数を指定する

最適化の最大反復回数は `max_iter` で指定できます。

```

param = {
    "xinit": [-1.0, 1.0],
    "bound": [
        [-5.0, 5.0],
        [-5.0, 5.0],
    ],
}

```

(次のページに続く)

(前のページからの続き)

```
],  
    "max_iter": 1000,  
}
```

この例では、最大 1000 回まで反復して最適化を行います。

1.4.9 ログ出力を抑制する

最適化中のログ出力が不要な場合は、`silent` に `True` を指定します。

```
param = {  
    "xinit": [-1.0, 1.0],  
    "bound": [  
        [-5.0, 5.0],  
        [-5.0, 5.0],  
    ],  
    "silent": True,  
}
```

1.4.10 辞書を引数とする関数の最適化

先ほどの Rosenbrock 関数は辞書を引数として渡すこともできます。CMA-ES ではこのような関数を最適化することも可能です。

```
import nulavis  
  
def rosenbrock(x):  
    return (  
        (1.0 - x[0]) ** 2  
        + 100.0 * (x[1] - x[0] ** 2) ** 2  
    )  
  
param = {  
    "xinit": {0: -1.0, 1: 1.0},  
    "bound": {  
        0: [-5.0, 5.0],  
        1: [-5.0, 5.0],  
    },  
}  
  
result = nulavis.cmaes.run(  
    rosenbrock,
```

(次のページに続く)

(前のページからの続き)

```
    2,  
    param,  
)  
  
print("最適解:", result["best_x"])  
print("目的関数値:", result["best_f"])
```

1.4.11 まとめ

このチュートリアルでは、Rosenbrock 関数を題材として、以下の手順で CMA-ES による最適化を実行しました。

1. 目的関数を Python の関数として定義する
2. `xinit` と `bound` を指定する
3. `nulavis.cmaes.run()` を呼び出して最適化する
4. `best_x` と `best_f` から最適化結果を取得する

基本的な使い方を理解したら、`lambda`、`mu`、`sigma`、`max_iter` などのパラメータを変更することで、CMA-ES の動作を調整できます。

各パラメータの詳細については、[API リファレンス](#) を参照してください。

索引

R

`run()` (*nulavis.cmaes* モジュール), 5