

PythonのDEAライブラリと事例紹介

森嶋 武史

分析・開発手法が高度化していく中で、オープンソースソフトウェアの重要性はますます増している。Pythonでは、R言語ほどオペレーションズ・リサーチの分野が扱われることは多くなく、提供されているライブラリ数が少ないのが現状である。しかし、Pythonはプログラマー人口の多くを占めるため、信頼性が高く便利なライブラリがあれば、データ包絡分析法(DEA)がより多くの人に広まり、さまざまな課題解決につながることを期待できる。本論では、こういった課題意識のもとで作成したDEAのライブラリを紹介し、その簡易的な適用事例を紹介する。

キーワード：データ包絡分析法, Python, オープンソースソフトウェア

1. はじめに

近年、データ分析手法や開発フレームワークの高度化により、オープンソースソフトウェア(Open source software; OSS)の重要性はますます高まっている。現在では、統計や機械学習を利用するプロジェクトにしても、一からモデルを構成するのではなく、scikit-learnなどのOSSを用いるのが一般的である[1]。手法が高度化、複雑化していく中で、理論を理解して利用することは必要であるが、それらを都度実装するのは、費用対効果面でも品質面でも効率的ではないためである。実際、米国防総省においても、2022年、プロプライエタリな製品を購入する前に優先して、既存の政府あるいはOSSのソリューションを採用し、既存のソリューションが十分でない場合にのみ、非商用のソフトウェアを作成しなくてはならない、という方針を明らかにした[2]。

筆者が2020年にデータ包絡分析法(Data envelopment analysis; DEA)の研究を始めた際も、まずはDEAを扱う既存のOSSライブラリを探した。しかし、使い慣れているPythonには残念ながら扱いやすいライブラリは当時提供されていなかった。ここでいう扱いやすいとは、メンテナンスされている、機能やドキュメントが充実していることを指す。一方で、R言語向けには多くの機能が実装されているDEAライブラリBenchmarkingが提供されていた[3]。

この状況の背景としては、DEAを利用するユーザーがPythonよりもR言語を主としていること、非プログラマーでも扱いやすいようなWebベースのアプリや

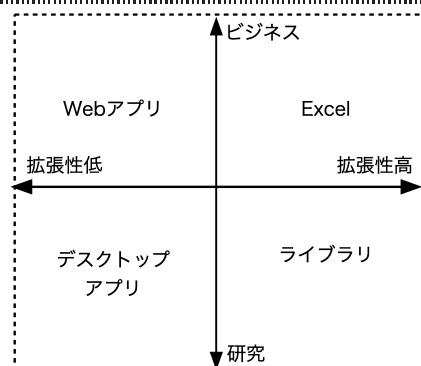


図1 DEAを提供するソフトウェアの分類

デスクトップアプリが用いられていることが可能性として挙げられる。これらのアプリはユーザビリティが高い面もあるが、コードベースで分析するには向いていない場合がある。コードベースで分析する場合は、データ加工や分析に柔軟性があり、システム化する場合においてもシナジーがある一方で、用いるプログラミング言語にある程度精通している必要がある。このように、同じDEAのソフトウェアといっても、用途や利用想定ユーザーによって提供するプラットフォームが異なる。図1は、これらDEAを提供するソフトウェアを、縦軸を用途、横軸を拡張性として4象限に分類したものである。

図1のように、アナリストや非プログラマー向けの領域はビジネス側であり、ExcelやWebアプリが該当し、機能が充実してより専門家側の領域はデスクトップアプリやライブラリが該当すると捉えることができる。Excelやライブラリはソースコードやファイルが提供されるため改変可能であるが、Webアプリやデスクトップアプリは改変が難しいため、このように分類することができる。ライブラリについては、Webアプリなどほかの提供方法とは異なりプログラミング言語

もりにぶ たけし
ダイキン工業(株)テクノロジー・イノベーションセンター
〒566-8585 大阪府摂津市西一津屋1番1号
takeshi.morinibu@daikin.co.jp

Worldwide, Sept 2023 :				
Rank	Change	Language	Share	1-year trend
1		Python	27.99 %	+0.1 %
2		Java	15.9 %	-1.1 %
3		JavaScript	9.36 %	-0.1 %
4		C#	6.67 %	-0.4 %
5		C/C++	6.54 %	+0.3 %
6		PHP	4.91 %	-0.4 %
7		R	4.4 %	+0.2 %
8		TypeScript	3.04 %	+0.2 %

図2 PYPL Popularity of Programming Language

依存性があるため、DEA ライブラリ Benchmarking は R 言語で扱うことができて、Python ユーザーにとってはアクセシビリティが悪いという状況であった。

一方で、Python はプログラマー人口の多くを占め、Google の検索エンジンでチュートリアルがどれほど検索されているかを示す指標によると、図 2 が示すように約 28% のシェアを誇り突出して 1 位であることがわかる [4]。そのため、整備された DEA ライブラリを Python ユーザーに対して提供することは、Python を通じて DEA の認知を広める可能性が期待できる。

しかし整備されたライブラリの提供は簡単ではなく、実際に Python において複数の DEA ライブラリを確認することができたが、そのどれもが継続的にメンテナンスされていないかったり、機能がごくわずかしかなかったり、信頼に足る状態ではなかった。この点が課題認識としてあったため、DEA をより多くの人に知ってもらい、有効活用してもらうために、利用人口の多い Python ユーザーにとって扱いやすい DEA ライブラリを展開することにした。

本稿では、まず 2 節で信頼性の高いライブラリとして満たすべき観点を述べ、次に 3 節で実際に開発した Python ライブラリである Pyfrontier を紹介する。最後に、4 節でこのライブラリを用いた DEA の適用事例を紹介し、その利便性などを説明する。

2. 信頼性の高いライブラリの条件

2.1 品質

まず、ライブラリの信頼性、つまり品質については、そのライブラリが正常にメンテナンスされているかどうかが重要である。プログラミング言語や依存性のある OSS のバージョンアップに対応していなかったり、バグ報告に対して長らく未対応であったりするのは

品質が高いとはいえず、分析や開発に用いるに足るものではない。

たとえば、Python は 2023 年 10 月時点ではバージョン 3.12 まで公開されているが、対応している Python のバージョンを記載していなかったり、現時点でコミュニティサポートが切れているバージョン 3.7 でライブラリの更新が止まっていると、ユーザーが扱っている Python のバージョンで問題なく動くことが保証されない。問題なく動くとは、記載された機能テストをパスしているとも言いかえられる。この場合、現在リリースされているライブラリは現行の Python バージョンでテストされているか、そのテストはどのような観点で記載されているか、不足していないかということも判断基準となる。たとえば、低すぎるテストの網羅率の場合は潜在的な問題がある可能性がある。

このような情報は、オープンソースであるためソースコードやリポジトリのドキュメントを見れば明らかであるし、それが不明確な場合は信頼性が低いといえる。またソースコード管理に加えてソーシャル機能サービスを提供する GitHub においては、スター数やコミット履歴も参考情報となる。コミット履歴の最新が数年前であった場合は、そのライブラリは長らくメンテナンスされていないといえる。逆に、これら条件をクリアしていれば信頼に足るライブラリとして認識されるということである。

2.2 ドキュメント

機能の充実は重要であるが、加えてその機能はどのように動作してどう扱えばいいのかをユーザーが理解できなければ、適切に扱うことは難しい。また、そもそも機能が認知されていない場合は、ユーザーには用いられない。そのためドキュメントは、提供している機能を網羅していること、機能の正しい使い方を示すこと、それらを認知してもらうためにわかりやすくあること、が必要となる。

たとえば、多くの機能が実装されているが、使い方やその例が記載されていない場合、ユーザーは手探りで挙動を追うことになってしまう。また、適切な使用方法がわからないまま扱ってしまうと、開発者の意図とは異なった方法で用いられてしまい、予期せぬ分析結果を発生させる可能性があり、その結果、そのライブラリは信頼に足るものとして扱われなくなってしまう。親切的な OSS のドキュメントは、ユーザーがアクセスしやすいよう、リポジトリを用いた別途ドキュメントページに詳細を記載することで、記載内容を充実させている。たとえば scikit-learn のドキュメントは

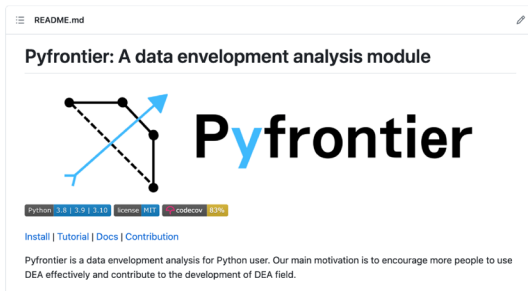


図 3 Pyfrontier

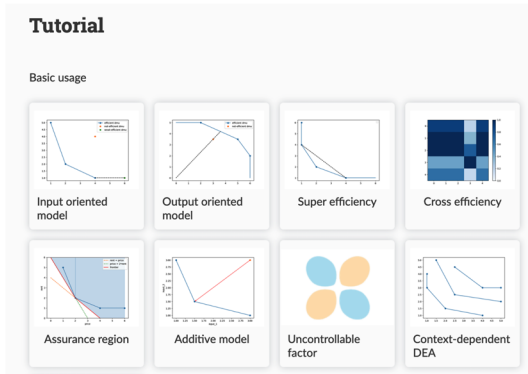


図 4 DEA の各分析手法のチュートリアル

読むだけでも、その分野の勉強になるほどである。

2.3 継続的開発

ここで取り上げた品質やドキュメントの観点を保守し続けるためには、継続的な開発が必要であり、そのためには Continuous Integration/Continuous Delivery (CICD) と呼ばれる、自動化されたビルドやリリースを支える仕組みが有効である。

3. Pyfrontier

2 節で触れた品質、ドキュメントについての観点を網羅して、DEA の Python ライブラリとして Pyfrontier をリリースした [5]。図 3 はトップ画面であり、ロゴのデザインは効率的フロンティアを意識して作成した。実装のロジック面では書籍を参考にした [6]。このプロジェクトでは多くの OSS に見られるように、信頼性を高めるために、単体テストを機能修正や追加の度に動かしてそのテストカバレッジを出力し、Python の対応バージョンなどの情報も表示するようにした。またドキュメントについては、Sphinx という OSS を用いて、Python の実行コードが説明として表記されるようにした。これらはチュートリアルとして、図 4 のように表示される。各チュートリアルの中身は図 5 の



図 5 Sphinx による Jupyter Notebook 形式のチュートリアル

References

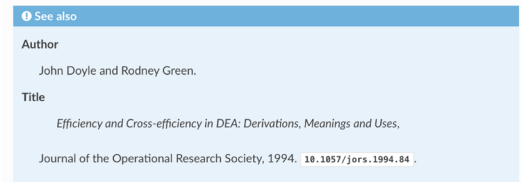


図 6 bibtexparser を用いた参考文献の表示

ようにデータ分析者にとっては馴染み深い、Jupyter Notebook 形式で表示される。ライブラリの設計面では、図 5 のサンプルコードのように、モデルをインスタンス化してフィットするという、scikit-learn ユーザーには馴染み深いものにした。このようにドキュメントを充実させることで、興味をもったユーザーがすぐに実践できる例を提供し、DEA が広く展開することを期待している。

さらに Pyfrontier 独自の要素として、OSS である bibtexparser を用いて参考文献を bib ファイルから抽出し、図 6 のようにドキュメントの補足に追記する機能を追加した。この機能によって、もとなる論文や応用事例にアクセスしやすくなるため、興味をもったユーザーが情報にアクセスすることを促進する。情報源は信頼できる唯一のソースとして bib ファイルをもとにしているため、記載ミスなどがないような設計となっている。一方、Pyfrontier の機能としてはまだ、Benchmarking は多くは網羅できていない。その点は、今後継続的に機能追加を想定しており、OSS として効果的に運用して価値を提供したいと考えている。

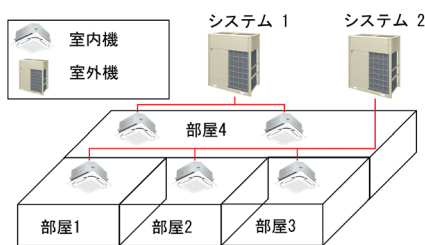


図 7 VRF システムの導入イメージ

4. Pyfrontier 適用事例

4.1 事前知識

本節では、この Pyfrontier を用いた DEA の適用事例として、オフィスビルの空調機器運用を改善する取り組みを紹介する。対象とする空調システムは、Variable refrigerant flow (VRF) である。VRF は 1980 年代に日本で導入されたビル用マルチエアコンであり、図 7 が示すように、一つの室外機に複数の室内機を紐付けて構成するシステムである。この VRF システムは、中小規模のオフィスビルにおいて個別制御性を重視したりメンテナンスを容易にするために、とくにアジア圏でシェアが広がっている [7]。オフィスビルにおける VRF システムの各室内機は、ユーザー制御によって多様なオンオフサイクルをもつことが指摘されている [8]。この場合、継続して運転し続けると仮定した場合のエネルギー使用量は、ユーザーの室温快適性の要望の違いがもたらすリモコン操作によって、実際のエネルギー使用量よりも 1.8 倍ほど多くなることを示した。つまり、ユーザーに空調の制御権がある場合、継続して運転するよりもエネルギー使用量が少なく収まるということである。

表 1 は、あるオフィスビルの 2019 年における冷房期間の運転データである。たとえば、一つ目のデータは 7 月 1 日に空調機器が ON であった時間を示し、室内機 A は 854 分、B は 611 分、C は 716 分運用していたことを示す。こうして見ると、ユーザーに制御権がある場合は冷房期間の間でも使われ方に違いがあることがわかる。このデータから、ユーザーがなるべく少ない運用時間で済ませている日と、そうでない日を分析し特定することが期待できる。少ない運用時間で済んでいる日を効率的な運用日として特定すれば、長く運用している日は効率的な運用日を目指して使い方を改善することで、よりエネルギーを抑えた運用ができる可能性がある。気温や気候の影響もあるため、一概に結論付けることはできないが、エネルギーの使い

表 1 各室内機の冷房期間における日ごとの総運用時間

DMU	日付	室内機 A	室内機 B	室内機 C
1	7-1	854	611	716
2	7-2	828	690	688
3	7-5	865	903	565
4	7-6	830	727	793
5	7-8	868	765	482
6	7-9	968	611	757
7	7-12	869	814	732
8	7-13	845	678	729
9	7-14	868	656	716
10	7-19	862	733	750
11	7-20	862	650	765
12	7-21	873	802	761
13	7-24	825	666	785
14	7-26	936	819	818
15	7-27	873	662	688
16	7-29	852	812	745
17	8-2	927	873	699
18	8-3	868	778	786
19	8-4	865	811	761
20	8-5	831	931	821
21	8-6	940	821	808
22	8-17	905	778	829
23	8-19	778	648	590
24	8-20	947	736	913
25	8-23	859	798	807
26	8-24	869	733	755
27	8-26	829	818	777
28	8-27	915	726	853
29	8-30	866	766	716
30	8-31	899	759	719

方を見直す機会となるデータや、議論を行うには十分な情報を提供できると期待できる。

4.2 DEA の適用

本分析の目的は、効率的な運用日を特定し、非効率的な運用日に対し改善案を提示することである。効率的な運用日とは短い運用時間で済んでいる日であるので、その日を特定するために、入力には各室内機の運用時間、出力は一定値とし、入力指向 Charnes-Cooper-Rhodes (CCR) モデルを用いて DEA を適用する。本論では出力一定のため CCR モデルを用いているが、一般に DEA のモデルを決める際には、入力の変化に対して出力がどのように影響を受けるかという、規模の収穫を考慮する必要がある。これは同特集号の DEA の例解 [9] により理論面の解説が行われているため、興味のある読者にとっては参考となるだろう。図 8 はこの定義を図示したものである。このようにモデル化することで、より少ない入力値の組合せであるほど効率

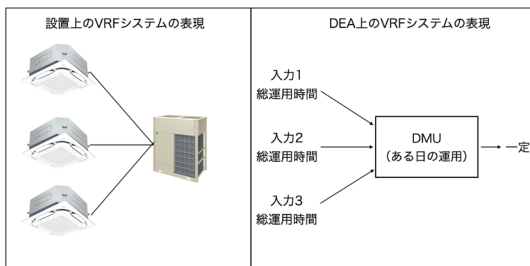


図8 DEAで扱う入出力データの構造
これは入出力の関係をモデル化したものであり、Decision Making Unit (DMU) と呼ばれる。

```

1 import pandas as pd
2 from Pyfrontier import frontier_model
3
4 data = pd.read_csv("./dataset.csv")
5 dea = frontier_model.MultipleDEA("CRS", "in")
6
7 dea.fit(
8     data[["A", "B", "C"]].to_numpy(),
9     data[["output"]].to_numpy(),
10 )
11
12 dea.results

```

図9 Pyfrontier のサンプルコード

的と判定される。本データにおいては3次元であるが、実際には紐付く室内機数に依存するモデルである。

本分析にはPyfrontierを用いる。図9の5行目でモデル定義し、7行目から10行目までで入出力データを引数にフィットすることで、効率値などの結果を取得（12行目）することができる。ここではさらに、追加の分析のためにクロス効率値 [10] を算出している。これもPyfrontierによって算出できる。

4.3 適用結果

表2はこの出力をまとめたものである。この結果は、7月1日、7月9日、8月19日が最も効率良い運用をしている日であり、それ以外の日に対してはこれらの日が構成する効率的フロンティアを目指して空調運用の改善を提案できる。ユーザーやビル管理者は、この情報をもとにエネルギーをより抑えた運用を行うことが期待できる。

さらに、このようなDEAの結果はさまざまなデータ分析手法と組み合わせることで、より興味深い分析を行うことができ、たとえばクロス効率値を用いたアプローチが存在する [10]。これは、クロス効率値によって全データを順位付けし、それを階層的クラスタリングにより分類することで、効率値が低く改善が効果的なグループと、効率値が高く改善の目標とできるグルー

表2 各室内機の冷房期間における効率値とクロス効率値

DMU	日付	効率値	クロス効率値
1	7-1	1	0.935
2	7-2	0.939	0.935
3	7-5	0.899	0.854
4	7-6	0.937	0.923
5	7-8	0.896	0.881
6	7-9	1	0.845
7	7-12	0.895	0.869
8	7-13	0.942	0.923
9	7-14	0.951	0.910
10	7-19	0.902	0.894
11	7-20	0.959	0.916
12	7-21	0.891	0.869
13	7-24	0.961	0.945
14	7-26	0.831	0.819
15	7-27	0.944	0.904
16	7-29	0.913	0.883
17	8-2	0.839	0.814
18	8-3	0.896	0.878
19	8-4	0.899	0.873
20	8-5	0.936	0.876
21	8-6	0.827	0.815
22	8-17	0.859	0.850
23	8-19	1	0.996
24	8-20	0.857	0.829
25	8-23	0.905	0.881
26	8-24	0.895	0.889
27	8-26	0.938	0.901
28	8-27	0.876	0.855
29	8-30	0.898	0.883
30	8-31	0.865	0.859

プを特定し、対応施策を考えやすくする方法である。クロス効率値は、重みの値が不確定になることからクロス効率値が不確定になるという課題がある。本稿で採用した方法は、具体的な効率値そのものではなく、それによって全データを分類することに目的があるため、値の変動によって結果が大きく変わることは抑制されている。この手法は、Pyfrontierにより入力指向CCRモデルを適応したのちクロス効率値を取得し、ScipyのLinkageによってウォード法によりクロス効率値を分類することで容易に実現できる。クロス効率値の算出には各DMUの最適ウェイトを用いるため乗数形を解く必要がある。先に挙げた同特集号のDEAの例解 [9] を参照することで、DEAの乗数形や包絡形についての理解をより深めることができるだろう。

結果を図10に示す。たとえばデータを三つに分類すると、最もクロス効率値が低いのはグループ3である。グループ3の中でもさらにクロス効率値が低い日

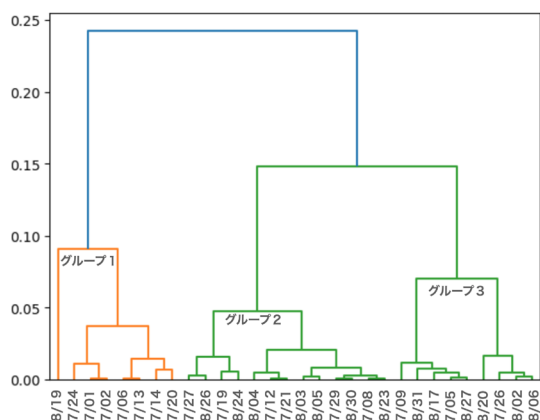


図 10 階層的クラスタリングによる空調運用日の分類

は 7 月 26 日、8 月 2 日や 8 月 6 日であるので、まずはそれらの日の運用を見直して改善することが有効と考えることができる。その際、参考としては最も効率値が高いグループ 1 の傾向を参考にすることもできるし、DEA の出力から参照集合が示す仮想 DMU を目指してもよい。

通常、DEA の目標は効率的数据の線形結合によって生成される仮想データであり、実際に存在するデータではないため、改善が実現可能であるかどうかは保証されない。また、いきなり効率的フロンティアを目指すことは、運用上難しい可能性もある。これらの問題に対応するために、たとえば、階層的 DEA により段階的な改善目標を策定する手法 [11] など、さまざまな提案がなされている。

5. おわりに

本稿では DEA の OSS ライブラリである Pyfrontier と、その分析例を紹介した。これらは Github で公開しており、pip 経由でインストール可能である [5]。ドキュメントの誤字やバグ報告、機能追加に関する要望など、Issue として投稿いただけると嬉しい。

本ライブラリが Python ユーザーに対しても信頼性の高い DEA ライブラリを提供することで、オペレー

ションズ・リサーチの分野がより多くの人に広まり、この分野の研究の進展に貢献することができれば幸いである。

謝辞 本ライブラリは、著者が大阪大学大学院の博士後期課程の一環として取り組んだものを含む。DEA を用いた分析の際、貴重な助言を賜った森田浩教授に感謝の意を表する。

参考文献

- [1] scikit-learn, <https://scikit-learn.org/stable/index.html> (2023 年 9 月 9 日閲覧)
- [2] Software Development and Open Source Software, <https://dodcio.defense.gov/Portals/0/Documents/Library/SoftwareDev-OpenSource.pdf> (2023 年 9 月 9 日閲覧)
- [3] Benchmarking, <https://cran.r-project.org/web/packages/Benchmarking/Benchmarking.pdf> (2023 年 9 月 9 日閲覧)
- [4] Popularity of Programming Language, <https://pypl.github.io/PYPL.html> (2023 年 9 月 9 日閲覧)
- [5] Pyfrontier, <https://github.com/NibuTake/PyDEA> (2023 年 9 月 9 日閲覧)
- [6] W. D. Cook and J. Zhu, *Data Envelopment Analysis Balanced Benchmarking*, 2013. (森田浩訳, 『データ包絡分析法—バランスの取れたベンチマーキング—』, 静岡学術出版, 2014.)
- [7] Z. Zhai and J. Rivas, “Promoting variable refrigerant flow system with a simple design and analysis tool,” *Journal of Building Engineering*, **15**, pp. 218–228, 2018.
- [8] G. Zhang, X. Li, W. Shi, B. Wang and Y. Cao, “Influence of occupant behavior on the energy performance of variable refrigerant flow systems for office buildings: A case study,” *Journal of Building Engineering*, **22**, 2019.
- [9] 森田浩, “DEA の例解,” オペレーションズ・リサーチ：経営の科学, **69**, pp. 12–19, 2024.
- [10] J. Wu, L. Liang and F. Yang, “Achievement and benchmarking of countries at the Summer Olympics using cross efficiency evaluation method,” *European Journal of Operational Research*, **197**, pp. 722–730, 2009.
- [11] T. Morinibu and H. Morita, “Improving operation management performance of air conditioners in offices using data envelopment analysis,” *Journal of Building Engineering*, **57**, 104702, 2022.