自動微分ライブラリの性能比較

01308414　関西大学　　*檀 寛成　　DAN Hiroshige
01111894　大阪公立大学　楠木 祥文　KUSUNOKI Yoshifumi

1. はじめに

自動微分とは，関数の（偏）導関数値を求める技術である．自動微分は，導関数値を必要とする工学の諸分野で古くから使われている技術であり，最適化分野では非線形最適化におけるコア技術の一つである．我々のグループでも自動微分ライブラリを開発している．

一方，近年，機械学習分野で自動微分の重要性が再注目されている [2]．元来，ニューラルネットワークにおける誤差逆伝播が自動微分と本質的に同等のものであることは知られていたが，2010 年代以降の深層学習の爆発的な広がりを背景に再び注目を集めている．機械学習分野ではライブラリの開発と公開が盛んに行われており，その流れで，自動微分ライブラリも開発･公開されるようになったようである．

そこで本発表では，我々が開発している自動微分ライブラリと，最適化や機械学習などで広く使われている自動微分ライブラリとの性能比較を行い，それぞれの特徴と優位性を確認する．

2. 自動微分ライブラリの性能比較

2.1. 比較対象と計算環境

本発表では，以下の自動微分ライブラリを比較する（カッコ内は実装言語（とバージョン番号））．

- `NOA_ad`: 我々が開発するライブラリ (C++)
- `ADOL-C`: COIN-OR [1] に収録されている自動微分ライブラリ (C++, Version 2.7)
- `PyTorch`: 自動微分を備えている機械学習ライブラリ (Python, Version 1.13.1)
- `ReverseDiff`: 近年注目されている科学技術計算言語 Julia で実装されている自動微分ライブラリ (Julia, Version 1.14.4)

また計算環境は表 1 の通りである．

本発表では，非線形最適化のベンチマーク問題に現れる目的関数と制約式の勾配を 100 回計算し，その際の計算時間と使用メモリ量を測定することで性能評価に用いた．ただし，勾配を 100 回計算するのに要する時間が 1 時間を超える（と判断される）ケースは計算を行わなかった．

表 1: 計算環境

OS	Ubuntu 20.04 LTS
CPU	Intel Core i9-9900K（16 スレッド）
Memory	64GB
Python	Version 3.7.9
Julia	Version 1.6.7

2.2. 計算対象

本発表は，非線形最適化のベンチマーク問題集 CUTEr に収録されている問題から `aug2d`，`mancino` を選択した[2]．

これらの問題は，いずれも変数や制約の数を変更することが可能である．ここでは，表 2 のように問題規模を変更しながら計算を行った．

表 2: 計算対象

	問題	row.	col.	nnz.	den.
(a1)	aug2d	97	240	564	2.4e-2
(a2)	aug2d	397	880	2344	6.7e-3
(a3)	aug2d	897	1920	5324	3.1e-3
(a4)	aug2d	1597	3360	9504	1.8e-3
(a5)	aug2d	2497	5200	14884	1.1e-3
(a6)	aug2d	3597	7440	21464	8.0e-4
(a7)	aug2d	4897	10080	29244	5.9e-4
(a8)	aug2d	6397	13120	38224	4.6e-4
(a9)	aug2d	8097	16560	48404	3.6e-4
(a10)	aug2d	9997	20400	59784	2.9e-4
(m1)	mancino	111	120	330	2.5e-2
(m2)	mancino	421	440	1260	6.8e-3
(m3)	mancino	931	960	2790	3.1e-3
(m4)	mancino	1641	1680	4920	1.8e-3
(m5)	mancino	2551	2600	7650	1.2e-3
(m6)	mancino	3661	3720	10980	8.1e-4
(m7)	mancino	4971	5040	14910	6.0e-4
(m8)	mancino	6481	6560	19440	4.6e-4
(m9)	mancino	8191	8280	24570	3.6e-4
(m10)	mancino	10101	10200	30300	2.9e-4

row.: 式の数（= 目的関数 + 制約条件数），col.: 変数の数，
nnz.: 勾配の非零要素数，
den.: 非零要素の密度 (= nnz. / (row. × col.)))

2.3. 計算結果と考察

平均計算時間と使用メモリ量を表 3, 4 に示す．

[1] OR 関係のオープンソースソフトウエアが多数収録されているサイト．`https://www.coin-or.org`

[2] 発表当日までに計算対象を増やしたいと考えている．

表 3: 平均計算時間 (単位: 秒) ((⋆): 0.001 秒未満)

	NOA_ad	ADOL-C	PyTorch	RevDiff
(a1)	(⋆)	(⋆)	0.025	0.002
(a2)	(⋆)	0.002	0.161	0.019
(a3)	(⋆)	0.022	3.783	0.104
(a4)	(⋆)	0.077	32.060	0.318
(a5)	0.001	0.154	–	0.859
(a6)	0.003	0.371	–	2.079
(a7)	0.005	0.608	–	4.671
(a8)	0.008	1.261	–	8.798
(a9)	0.013	1.824	–	15.431
(a10)	0.015	2.561	–	27.591
(m1)	(⋆)	(⋆)	0.021	0.003
(m2)	(⋆)	(⋆)	0.104	0.025
(m3)	(⋆)	(⋆)	0.719	0.129
(m4)	(⋆)	(⋆)	11.120	0.392
(m5)	0.002	0.001	–	0.983
(m6)	0.005	0.002	–	2.483
(m7)	0.009	0.004	–	5.690
(m8)	0.012	0.006	–	11.103
(m9)	0.019	0.011	–	19.845
(m10)	0.023	0.014	–	32.568

表 4: 使用メモリ量 (単位: MB)

	NOA_ad	ADOL-C	PyTorch	RevDiff
(a1)	5.14	6.13	3.96	276.68
(a2)	7.56	24.82	27.46	283.41
(a3)	11.40	97.59	92.73	307.32
(a4)	16.75	319.41	376.16	367.15
(a5)	23.86	600.84	–	488.54
(a6)	32.87	1488.99	–	705.51
(a7)	43.00	2319.76	–	1057.81
(a8)	54.66	5051.90	–	1594.80
(a9)	68.57	7029.29	–	2369.82
(a10)	83.29	9553.64	–	3446.62
(m1)	5.67	5.57	3.03	323.56
(m2)	9.53	6.19	15.98	328.00
(m3)	16.02	7.31	74.99	341.82
(m4)	25.30	9.70	181.15	374.55
(m5)	36.63	11.80	–	439.39
(m6)	50.90	18.11	–	552.32
(m7)	67.60	22.27	–	734.32
(m8)	86.64	26.77	–	1010.37
(m9)	108.64	44.00	–	1406.26
(m10)	132.62	51.18	–	1954.51

計算時間・メモリ使用量ともに，C++ で実装されている NOA_ad，ADOL-C が優位であり，続いて Julia で実装されている ReverseDiff，Python で実装されている PyTorch（の自動微分機能）という結果となった．これには，プログラミング言語の計算速度差が大きく影響している可能性がある．

一方で，ReverseDiff，PyTorch には，スクリプト言語 (Julia, Python) 特有の使いやすさがある．また，これらの言語で書かれた他のライブラリ（特に機械学習向けライブラリ）との関係を考えると，それぞれの言語の「エコシステム」内で重要な役割を果たすライブラリであるといえよう．

NOA_ad と ADOL-C では，問題によって優劣がわかれた．特に，ADOL-C は，問題によって大きく性能差のある結果となった．現時点では，この原因の詳細は不明である[3]．

非線形最適化問題の求解に際しては，目的関数や制約式の勾配（やヘッセ行列）を繰り返し計算する必要がある．その観点からは，問題が中規模以上であれば，C++ で実装された自動微分ライブラリが必須になりそうである．

3. 「おわりに」に代えて：最適化分野における無償ソルバ整備の現状

近年，機械学習が広範に支持を得ているのは，深層学習を中心とするコア技術のライブラリが実装され，オープンソースとして無償で提供されていることが大きく寄与しているように思われる．これにより，いわゆる「参入障壁」が低くなっている現状がある．

一方，最適化分野でもこの流れが始まっているようである．例えば，AMPL は 2021 年より Community Edition [1] というバージョンの無償配布を始めている．AMPL 自体はソルバを呼び出すためのモデリング言語ライブラリであるが，Community Edition はオープンソースのソルバをバンドルして配布しているため，非常に使いやすくなっている．また，2022 年 11 月より，Zuse Institute Berlin が作成している最適化ソルバ SCIP [3] が，Apache 2.0 ライセンスの下で無償で配布されている．このライセンスの下では，商用利用や改変・再配布も可能である．

我々の開発する NOA_ad（を含む一連のソルバ・ライブラリ）は，現時点では未公開だが，この流れに寄与できるように頑張りたい（一緒に開発して頂ける方を絶賛募集中です）．

[3] ライブラリの使い方が適切かを検証する必要もある．

参考文献

[1] AMPL Community Edition, https://ampl.com/ce/

[2] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic Differentiation in Machine Learning: a Survey, Journal of Machine Learning Research, **18** (2018), 1-43.

[3] SCIP Optimization Suite, https://www.scipopt.org