京都大学防災研究所年報 第 68 号 B
DPRI Annuals, No. 68 B, 2025

自動微分を用いた数値最適化と変分法データ同化

Numerical Optimization and Variational Data Assimilation Using Automatic Differentiation

榎本剛・中下早織

Takeshi ENOMOTO and Saori NAKASHITA

Synopsis

Gradients play an important role in optimization in both variational data assimilation and machine learning. Traditionally, adjoint models have been coded by hand for computational performance. However, recent advances in hardware and software now allow for the automated construction of adjoints using automatic differentiation (AD). This article describes the fundamentals of adjoints and AD. Using a simple nonlinear model as an example, we demonstrate how to systematically construct its adjoint. The Lagrange multiplier method is applied to derive the adjoint directly from the model's code, a process analogous to how AD works with a computational graph. We also show that AD can be applied to a discontinuous model, a common feature in physics schemes with switches, such as convective parametrization. A clear understanding of adjoints and automatic differentiation is essential for the effective use of machine learning frameworks in data assimilation and machine learning.

キーワード: 随伴モデル，計算グラフ，誤差逆伝播，勾配，ラグランジュの未定乗数法

Keywords: adjoint model, computational graph, back propagation, gradient, Lagrange multiplier

1. はじめに

データ同化は，数値天気予報に不可欠な初期状態を推定するために用いられる．最適な初期状態とは，過去の初期状態から予報モデルを用いて求めた第一推定値及び観測との乖離が最も少ないものを指す．最適値の推定には，第一推定値及び観測の誤差共分散を考慮し，誤差のより小さいものに大きな重みをつける（Lorenc 1986）．

乖離の最小化には，数値最適化が用いられる（Nocedal and Wright 2006）．大気や海洋モデルなどの問題は大規模になるため，最適化を効率よく行うために，共軛勾配法など勾配に基づく手法が用いられることが多い（Navon and Legler 1991）．機械学習でも，勾配に基づく数値最適化により，ニューラルネットワークのパラメタ推定が行われる（Amari 1967, 1993）．さらに勾配はハミルトニアン・モンテカルロ（Hamiltonian Monte Carlo, HMC）法や変分推論などベイズ推定にも不可欠である（Baydin et al. 2018）．

変分法データ同化では，人間が書いた随伴モデルが用いられてきた．随伴を自動生成する自動微分（Iri 1984, 1991）は変分法が適用された頃から知られており，随伴モデルの作成に利用されている．これまで使われてきた自動微分は，モデルのソースコードから随伴モデルのソースコードを作成する翻訳器である．翻訳された随伴のソースは，通常は最終的に人間が確認し手を加える．人間の介入が必要とされてきたのは，正確性の確保と計算効率のためである．

一方機械学習においては，通常自動微分が用いられ，人間が介入することは稀である．近年の機械学習の興隆により自動微分を含む機械学習フレームワークやライブラリが充実してきた．これらの自動微分はソース翻訳ではなく，後で述べるように計算グラフを用いたものである．

本稿では，まず第2節で簡単に随伴について述べた後，随伴モデルを直接モデルのコードから作成する方法（Lawson et al. 1995）を示す．次に，第3節において自動微分の特徴や仕組みについて述べる．さらに，第4節で自動微分の適用例を示す．その中で自動微分が積雲対流パラメタリゼーションなど，作動・不作動の

スイッチがある物理過程に共通する不連続モデルにも適用できることも示す．第5節に結語を述べる．

2. 随伴モデル

この節では，Luchini and Bottaro (2014)に基づき随伴法の背景について述べるとともに，具体的に随伴モデルを構築する手順（Lawson et al. 1995）を示す．

2.1 背景

随伴法はLagrange (1763)が一般微分方程式の階数を下げるために考案し，流体の運動や弦の振動，衛星の軌道計算に適用した．部分積分により得られた補助方程式をéquation adjointeと呼んだ．補助方程式は元の方程式の弱形式であり，有限要素法による数値解法にも応用されている．

随伴方程式は内積が定義できるヒルベルト空間で定義されることが多いが内積は必須ではない．ベクトル空間のうち，内積を必要せず，より一般的なバナッハ空間において，線型形式を用いて随伴方程式が定義できる．

2.1.1 瞬時離散線型系

まず，随伴が行列の転置であることを確認するために，瞬時離散線型系

$$\mathbf{x} = \mathbf{H}\mathbf{u} \tag{1}$$

を考える．ここで $\mathbf{x}$ は要素数 Mの実数または複素数のベクトル，入力 $\mathbf{u}$ は要素数 N のベクトル， $\mathbf{H}$ は $M \times N$ の行列を表す．ここで注目する量，目的 J は $\mathbf{x}$ のスカラー線型函数で以下のように定義する．

$$J = \mathbf{y}^{\mathrm{T}}\mathbf{x} = \mathbf{y}^{\mathrm{T}}\mathbf{H}\mathbf{u} = \mathbf{v}^{\mathrm{T}}\mathbf{u} \tag{2}$$

ここで,

$$\mathbf{v} = \mathbf{H}^{\mathrm{T}}\mathbf{y} \tag{3}$$

は随伴方程式である．式(3)を式(1)と比較すると，$\mathbf{H}$ の転置で表されており，目的 J における$\mathbf{x}$の係数 $\mathbf{y}$ は随伴変数であることが分かる．また，

$$\mathbf{v} = \frac{\partial J}{\partial \mathbf{u}} \tag{4}$$

は入力に対する目的の感度を表している．

2.1.2 離散時間力学系

次に離散時間力学系

$$\mathbf{x}_{n+1} = \mathbf{A}_n\mathbf{x}_n,\ n = 0,\ldots,N-1 \tag{5}$$

を考える．$\mathbf{x}_n$ は時間ステップn における要素数M のベクトル，$\mathbf{A}$は$M \times M$ の時間遷移行列である．目的を $J = \mathbf{y}^{\mathrm{T}}\mathbf{x}_N$ と表し式(5)を繰り返し用いることにより、

$$\begin{aligned} J &= \mathbf{y}^{\mathrm{T}}\mathbf{x}_N = \mathbf{y}^{\mathrm{T}}(\mathbf{A}_{N-1}(\mathbf{A}_{N-2}(\ldots\mathbf{A}_1(\mathbf{A}_0\mathbf{x}_0)))) \\ &= ((((\mathbf{y}^{\mathrm{T}}\mathbf{A}_{N-1})\mathbf{A}_{N-2}\ldots)\mathbf{A}_1)\mathbf{A}_0)\mathbf{x}_0 = \mathbf{v}_0^{\mathrm{T}}\mathbf{x}_0 \end{aligned} \tag{6}$$

が得られる。これは入力$\mathbf{x}_0$の線型形式になっている．随伴方程式は以下のように $\mathbf{v}_N = \mathbf{y}$ から初めて時間逆方向に進める．

$$\mathbf{v}_n = \mathbf{A}_n^{\mathrm{T}}\mathbf{v}_{n+1},\ n = N-1,\ldots,0 \tag{7}$$

式(7)の随伴方程式で初期感度 $\mathbf{v}_0$ を計算すれば，異なる初期条件に対する目的は式(6)の最右辺 $J = \mathbf{v}_0^{\mathrm{T}}\mathbf{x}_0$ を計算することにより$O(M)$ の計算量で求めることができる．また，目的

$$J = \mathbf{v}_n^{\mathrm{T}}\mathbf{x}_n = \text{const}.\ n = 0,\ldots,N \tag{8}$$

であることも分かる．この関係は，随伴モデルの確認や感度解析（Fujii et al. 2008）に利用できる．

2.1.3 連続時間力学系

本稿では詳細を省略するが，より一般的な線型力学系や連続時間力学系

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{9}$$

についても同様に随伴方程式を求めることができる．式(9)の随伴方程式は

$$-\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \mathbf{A}^{\mathrm{T}}\mathbf{v} + \mathbf{y} \tag{10}$$

となる．このとき，初期及び終端条件

$$\mathbf{x}(0) = \mathbf{x}_0, \mathbf{v}(T) = 0 \tag{11}$$

であり，目的は

$$J = \int_0^T \mathbf{y}^{\mathrm{T}}(t)\mathbf{x}(t)\mathrm{d}t \tag{12}$$

とする．式(10)は式(9)に未知のベクトル函数$\mathbf{v}(t)$をかけて部分積分することにより$\mathrm{d}\mathbf{x}/\mathrm{d}t$のない形にし式(11)を適用すると，$\mathbf{x}$ の係数として式(10)が導かれる．

式(7)や(9)で表される随伴方程式は時間逆方向，すなわち最終ステップ N または最終時刻 T から随伴変数 $\mathbf{v}$ を積分することに注意する．

2.1.4 非線型方程式の随伴

式(3)や(7)は，非線型方程式の接線型であるとみなすこともできる．式(1)で表される線型系の出力に対する入力の微分は一定で行列の転置と結び付いている．より一般的な非線型問題において，随伴は状態の函数となり線型代数として解釈することできなくなるものの，その微分は意味を持つ．

式(5)を一般化した時間遷移方程式は

$$\mathbf{x}_{n+1} = f_n(\mathbf{x}_n),\ n = 0,\ldots,N-1 \tag{13}$$

で初期条件は $\mathbf{x}_0$，目的は $J(\mathbf{x}_N)$ で表される．

目的 J の初期状態 $\mathbf{x}_0$ に関する微分は連鎖律を用いて次のように表すことができる．

$$\left(\frac{\partial J}{\partial \mathbf{x}_0}\right)^{\mathrm{T}} = \left(\frac{\partial J}{\partial \mathbf{x}_N}\right)^{\mathrm{T}} \frac{\partial \mathbf{x}_N}{\partial \mathbf{x}_{N-1}} \cdots \frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0} \tag{14}$$

ここで $\partial \mathbf{x}_{n+1}/\partial \mathbf{x}_n$ は正方ヤコビ行列である．式(6)のように連続したヤコビ行列の積を左に結び付けることにより，逆向きの漸化式として随伴方程式

$$\mathbf{v}_n = \left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right)^{\mathrm{T}} \mathbf{v}_{n+1}, n = N-1,\ldots,0 \tag{15}$$

が得られる．終端条件 $\mathbf{v}_N = \partial J/\partial \mathbf{x}_N$ から逆順に式(15)を計算すると，$\mathbf{v}_0 = \partial J/\partial \mathbf{x}_0$ が得られる．

2.2 随伴モデルの構築

随伴の作り方には、大別して二つの方法がある。

1. 支配方程式系に随伴函数を掛けて、部分積分をして、随伴方程式を求めてから離散化して随伴モデルを作る。
2. 支配方程式系を離散化し、その接線型モデルを作り、接線型モデルを元に随伴モデルを作成する。

両者は同じ解に収束すると考えられるが，離散化誤差は同じではない．弱い意味での（積分値としての）収束であり，各点での勾配は異なる可能性があるため，境界付近では特に注意が必要である．

前者の方法では離散化したモデルが厳密に対応しない可能性がある．後者の方法は式(14)のヤコビ行列を明示的に求めて転置するのは非効率であり，大規模問題ではメモリの制約からヤコビ行列を扱うことが事実上できない．仮に扱えたとしても，ヤコビ行列の積の計算量は膨大となる．

2.2.1 未定乗数法による随伴の導出

式(15)を素朴に適用する代わりに，ここでは，離散化された順行モデルのプログラムの各行から随伴モデルを作成する．随伴方程式と随伴変数は，ラグランジュの未定乗数法を用いて導出する点は，前者の方法に類似しているが，後者の方法とは異なり，接線型モデルを経ずに直接随伴を作成する．

初期値やパラメタなど制御変数を $\mathbf{x} = (x_1, \ldots, x_m)^{\mathrm{T}}$ とし，時間ステップ n における状態変数の一つを $z_n, n = 1,\ldots,N$ で表す．目的は制御変数と状態に依存するものとし $J = J(\mathbf{x}, z_1, \ldots, z_N)$ で表す．

次の状態がその一つ前の時間ステップのみに依存するとし，モデルを $f_n(\mathbf{x}, z_{n-1})$で表すと，順行計算は次のように表される．

$$z_1 = f_1(\mathbf{x}),\ z_n = f_n(\mathbf{x}, z_{n-1}),\ n = 2,\ldots,N \tag{16}$$

未定乗数を λ_n とするとラグランジュ函数は

$$\begin{aligned} L(\mathbf{x}, z_1, \ldots, z_N, \lambda_1, \ldots, \lambda_N) &= J(\mathbf{x}, z_1, \ldots, z_N) \\ -\lambda_1(z_1 - f_1(\mathbf{x})) &- \sum_{n=2}^{N} \lambda_n (z_n - f_n(\mathbf{x}, z_{n-1})) \end{aligned} \tag{17}$$

ラグランジュ函数の鞍点では，L の微分が同時に0になる．$\partial L/\partial \lambda_n, n = 1,\ldots,N$ から支配方程式(16)が得られる．一方，$\partial L/\partial z_n$ から随伴方程式

$$\begin{aligned} \lambda_N &= \frac{\partial J}{\partial z_N}, \\ \lambda_n &= \frac{\partial J}{\partial z_n} + \frac{\partial f_{n+1}}{\partial z_n}\lambda_{n+1}, n = N-1,\ldots,1 \end{aligned} \tag{18}$$

が得られ，状態の随伴変数を表す未定乗数が求まる．さらに，$\partial L/\partial \mathbf{x}$ から目的の制御変数に対する微分

$$\begin{aligned} (\nabla_{\mathbf{x}} J)_k &= (\nabla_{\mathbf{x}} L)_k \\ &= \frac{\partial L}{\partial x_k} = \sum_{i=1}^{N} \frac{\partial f_i}{\partial x_k}\lambda_i,\ k = 1,\ldots,m \end{aligned} \tag{19}$$

を計算することができる．式(19)は，モデルの制御変数についての微分に式(18)で求めた各時間ステップでの未定乗数を掛けたものの総和が，目的の制御変数についての微分を表すことを示している．

順行方程式のコードから随伴を作成する場合，式(18)及び(19)にはラグランジュ函数は現れないので，式(17)のラグランジュ函数を明示的に作る必要はない．具体的な手順は次のとおりである．

- 随伴変数は0に初期化する．
- 時間を逆行するので，コードを逆順に辿り，ループは逆順に回す．
- モデルの状態についての微分を計算し，一つ前のステップの未定乗数を掛ける（式(18)の二番目の式の右辺第2項）．観測があるステップでは，目的が状態に依存するので，目的の状態についての微分を加える（同第1項）．
- モデルの制御変数についての微分を計算し，そのステップの未定乗数を掛けて和をとることで，制御変数の随伴を計算する（式(19)）．

2.2.2 簡単な非線型モデルの随伴の例

随伴の例として，Lorenz (1963)モデル

$$\begin{aligned} \dot{X} &= -\sigma X + \sigma Y \\ \dot{Y} &= -XZ + rX - Y \\ \dot{Z} &= XY - \beta Z \end{aligned} \tag{20}$$

の随伴を示す．時間にEuler法を用いると，式(20)はList. 1に示すように離散化される．

上述の手順に従って，このコードの随伴を作るとList. 2のようになる．例えば，List. 1の3行目を `b` で偏微分したものに，左辺の変数の随伴 `az[n + 1]`

```
x[n + 1] ← x[n] + dt * s * (-x[n] + y[n])
y[n + 1] ← y[n] + dt * (-x[n] * z[n] + r * x[n] - y[n])
z[n + 1] ← z[n] + dt * (x[n] * y[n] - b * z[n])
```

List. 1 Lorenz (1963) model discretized in time using the Euler method in the R language.

```
ab ← ab - dt * z[n] * az[n + 1]
az[n] ← az[n] + (1 - dt * b) * az[n + 1]
ay[n] ← ay[n] + dt * x[n] * az[n + 1]
ax[n] ← ax[n] + dt * y[n] * az[n + 1]
ar ← ar + dt * x[n] * ay[n + 1]
az[n] ← az[n] - dt * x[n] * ay[n + 1]
ay[n] ← ay[n] + (1 - dt) * ay[n + 1]
ax[n] ← ax[n] + dt * (-z[n] + r) * ay[n + 1]
as ← as + dt * (-x[n] + y[n]) * ax[n + 1]
ay[n] ← ay[n] + s * dt * ax[n + 1]
ax[n] ← ax[n] + (1 - s * dt) * ax[n + 1]
```

List.2 Adjoint of List. 1 without the gradient from innovation.

```
ar ← ar + dt * x[n] * ay[n + 1]
as ← as + dt * (-x[n] + y[n]) * ax[n + 1]
ab ← ab - dt * z[n] * az[n + 1]
ax[n] ← (1 - s * dt) * ax[n + 1] + dt * (-z[n] + r) * ay[n + 1]
                                 + dt * y[n] * az[n + 1]
ay[n] ← s * dt * ax[n + 1] + (1 - dt) * ay[n + 1] + dt * x[n] *
az[n + 1]
az[n] ← -dt * x[n] * ay[n + 1] + (1 - dt * b) * az[n + 1]
```

List.3 As in List. 2 but for the refactored version.

を乗じたものを足し込んでパラメタ β の随伴 `ab` （随伴変数には `a` をつけている）が得られる．状態変数 z の随伴 `az` も同様にList. 1の3行目を `z[n]` で偏微分したものに `az[n + 1]` を乗じて足し込むことにより `az[n]` が得られる．

さらに整理したものをList. 3に示す．状態の随伴変数の `dt` が掛かった項を取り出すと

$$-\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\dot{X}^{\mathrm{a}}\\ \dot{Y}^{\mathrm{a}}\\ \dot{Z}^{\mathrm{a}}\end{bmatrix} = \begin{bmatrix}-\sigma & r-Z & Y\\ \sigma & -1 & X\\ 0 & -X & -\beta\end{bmatrix} \tag{21}$$

となる．式(21)の右辺は，式(20)を線型化し行列で表したものを転置したものと一致している．

List. 1は冗長であるが，順行モデルから直接機械的に生成できる．この手順は，自動微分が計算グラフを用いて行なっていることと類似している．

List 2を用いた初期状態及びパラメタ推定とList. 3つまり式(21)に基づく初期状態の推定結果は，第3節に述べる自動微分の比較対象として，第4.2節に示す．

3. 自動微分

この節では，まず微分を計算する様々な方法について述べ，自動微分の利点を明らかにする．次に自動微分の種類と計算グラフを用いた自動微分の仕組みについて述べる．

3.1 微分の計算方法

微分の計算方法は四つに分類できる（Margossian et al. 2019）

1. 解析的微分: 人間が予め解析的に計算する．厳密で最も高速であるが，導出とコード化に時間を要し誤りやすい．
2. 差分: コード化は容易だが，浮動小数点数誤差の影響を受け，自由度の数だけ評価が必要で大規模問題では低速．
3. 記号微分: 厳密だが，メモリ集約的で低速．

4. 自動微分: （機械精度の範囲で）厳密で速度は解析的微分に匹敵しうる．実装に注意が必要だが，既に良いパッケージが複数存在する．

3.2 複素ステップ微分

上記の微分の計算方法2に示したように，実函数$f(x)$の中央差分

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h} \tag{22}$$

はステップ幅hが小さいほど正確に求められるが，あまり小さくしすぎると浮動小数点数誤差の影響を受ける．一方 虚部に摂動を与えてテーラー展開

$$f(x+ih) = f(x) + ihf'(x) - \frac{h^2}{2!}f''(x) - \frac{ih^3}{3!}f^{(3)}(x) + \ldots \tag{23}$$

すると，虚部から微分が$O(h^2)$の精度で計算できる（Lyness 1967, Lyness and Moler 1967, Squire and Trapp 1998）．

$$f'(x) \approx \mathrm{Im}\frac{f(x+ih)}{h} \tag{24}$$

RやPythonのような言語では，函数を書き換えることなく，虚数の摂動を与えて式(24)を計算するだけで高精度で微分が得られる．FortranやCなど強い型付け言語でも変数を複素数に書き換える必要があるが，比較的簡単な変更で済む．なお，複素数に対しては式(24)は成り立たないが，実部と虚部をそれぞれ複素数とすることによりスペクトルモデルに適用できるようにする拡張が提案されている（Cerviño and Bewley 2003）．

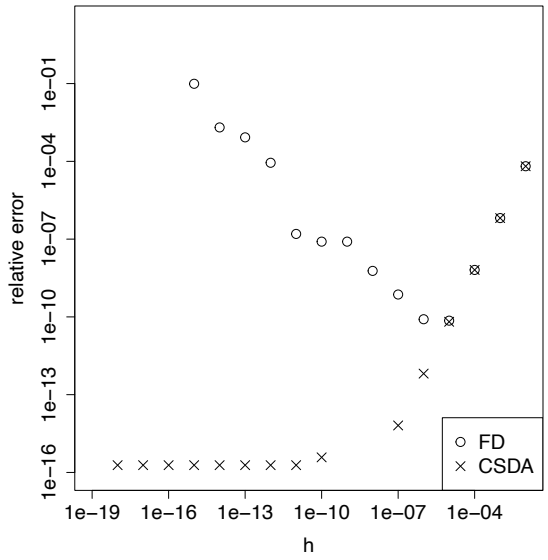

Fig. 1 Relative error of finite differences (FD) and complex step derivative approximation (CSDA) for the numerical derivative of $f(x) = x^{9/2}$ at $x = 1.5$. The missing values with $h \leq 1 \times 10^{-16}$ of FD are due to the loss of derivatives, i.e. $f(x+h) - f(x) = 0$. The missing values of CSDA at $h = 1 \times 10^{-8}$,1×10^{-9}, and 1×10^{-19} are due to perfect agreement with the analytic value up to the machine precision.

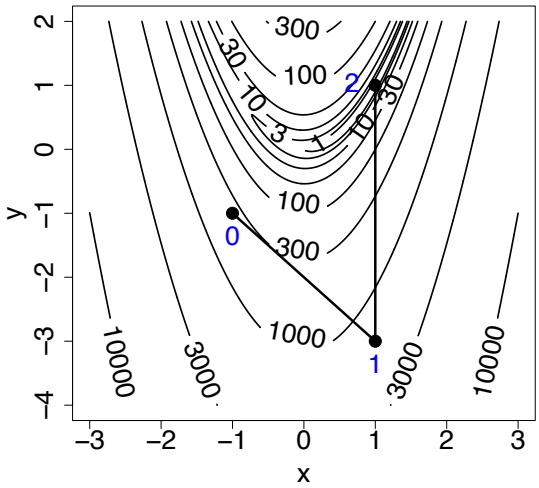

Fig. 2 Optimization of the Rosenbrock function $(1-x)^2 + 100(y-x^2)^2$ from $(x, y) = (-1, -1)$ using the Gauss–Newton method. The derivatives are computed using CSDA. The numbers of iterations are marked in blue.

例として，Fig. 1に$f(x) = x^{9/2}$の$x = 1.5$における微

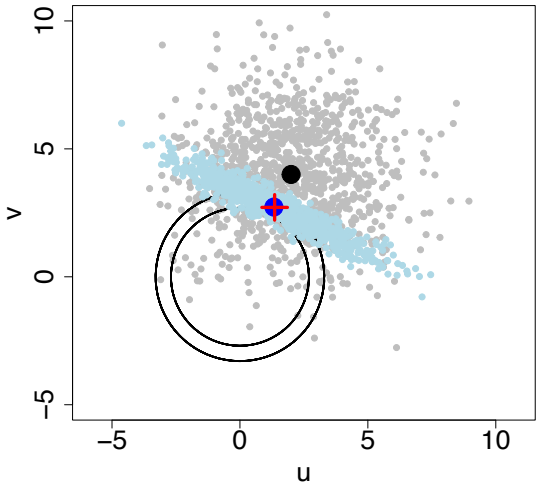

Fig. 3 Assimilation of a wind speed of 3 m s^{-1} with 10% observational error using the maximum likelihood ensemble filter (MLEF). The black dot indicates the first guess at (2, 4). The the gray dots are 1000 ensemble members generated by adding Gaussian noise with standard deviation of 2. The blue dot indicates the MLEF analysis using CSDA derivatives and the red + is the analytic solution. The donut-shaped area spanned by the two circles indicate the wind speed with one standard deviation. The Gauss–Newton optimization is used with tolerance of 10^{-4}.

分を差分（FD）と複素ステップ微分（CSDA: complex step derivative approximation）を用いて，数値的に求めた結果を比較する (Squire and Trapp 1998). FDの相対誤差は，丸め誤差と切断誤差がバランスする最適値

$$h_{\mathrm{opt}} = 2\left|\frac{uf(x)}{f''(x)}\right|^{1/2} \tag{25}$$

程度まで減少する．ここでuは相対精度で倍精度では$u \approx 1.11\times10^{-16}, \sqrt{u} \approx 1.05\times10^{-8}$，である．この例で見積ると，$h_{\mathrm{opt}} \approx 1.25\times10^{-7}$となるが，既に$h = 1\times10^{-7}$では誤差は増大に転じている．

複素ステップ微分を使って，Rosenbrock函数を最適化した例をFig. 2に示す．Gauss–Newton法を用いると，解析的に求めた微分をCSDAに置き換えても2ステップで収束する (Enomoto and Nakashita 2024)．風速の同化でも2ステップで解析解からの相対誤差0.0375%の値に収束した (Fig. 3).

3.3 計算グラフ

自動微分には，順行と逆行とがある（Iri 1984, 1991, Baydin 2018, Margossian 2019）．式(6)の $\mathbf{A}_n$ や式(14)の $\partial\mathbf{x}_{n+1}/\partial\mathbf{x}_n$ の積において，行列ベクトル積を右に結びつけた走査を順行（forward mode）呼び，式(5)に対応する．逆行（reverse mode）は左に結びつけたもので，随伴方程式(7)あるいは(15)に対応する．

計算の過程は計算グラフで表すことができる．これを用いて，一組の順行と逆行走査により自動微分により勾配が計算されることを示す.

Fig. 4は，式(20)に示したLorenz (1963)の$\dot{Z}$の右辺第一項の計算グラフである．円で示される節には変数を格納する葉と計算を表す節とがあり，葉と節をつなぐ枝はデータの流れを示す．葉には，vで表される変数

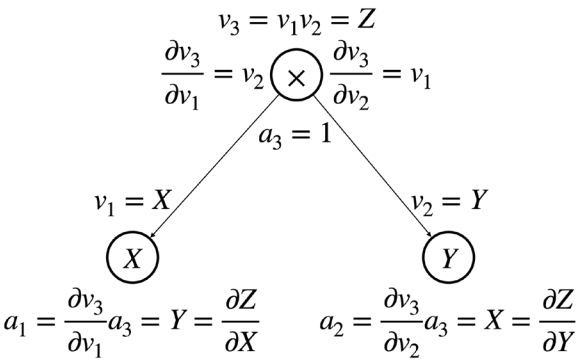

Fig. 4 Computation graph for a nonlinear term XY that appears in the tendency of $\dot{Z}$ of the Lorenz (1963) model. See Fig. A1 for the whole computational graphs for the three prognostic equations. Arrows represent the back propagation.

に加えて随伴変数aを用意する．順行では，節に向かって入力データを送り，計算（$v_3 = v_1 v_2 = Z$）するとともに，枝で表される関係について微分（$\partial v_3/\partial v_1$及び$\partial v_2/\partial v_1$）を計算する．逆行では，グラフを逆に辿る．まず随伴変数$a_3 = 1$に初期化し，順行と逆向きに随伴を伝播させながら，連鎖律により随伴変数の値を求める．付録のFig. A1にLorenz (1963)モデルの三つの方程式の計算グラフを示す.

計算グラフによる自動微分は，第2.1節に述べた順行と随伴の計算そのものであり，第2.2節に示したコードの各行から随伴モデルを構築する方法を機械的に行うものである.

3.4 自動微分パッケージ

自動微分の機能を含むパッケージは多数あり，https://www.autodiff.org/にまとめられている．そのうちのいくつかをTbl. 1に示す.

伝統的なツールはソース翻訳によるものである（ADIFOR, Tapenade, Adept）．生成されたコードは，

Tbl. 1 Selected list of packages for automatic differentiation.

package	language	feature	reference
ADIFOR	Fortran	source-to-source	Bischof et al. (1996)
Tapenade	C++	source-to-source	Hascoet and Pascual (2013)
Stan Math	C++	operator overloading	Carpenter et al. (2015)
Adept	C++	operator overloading	Hogan et al. (2017)
CoDiPack	C++	expression template	Sagebaum et al. (2018)
PyTorch (libtorch)	Python (C++)	eager execution	Paszke et al. (2017)
JAX	Python	XLA compiler	Frostig et al. (2018)
Enzyme	LLVM IR	optimization of IR	Moses et al. (2020)

Tbl. 2 Variational initial state and parameter estimation of the Lorenz (1963) model. n refers to the number of iterations.

	gradient	optimization	n	r	σ	β	X	Y	Z
truth				32	10	2.667	1	3	5
first guess				30	11	2	1.1	3.3	5.5
estimate	hand-coded adjoint	nvm	365	32	10	2.667	1	3	5
	hand-coded adjoint	BFGS	39	32.00	10.00	2.667	1.235	2.702	4.975
	torch for R	L-BFGS	62	32	10.00	2.667	0.8989	3.128	5.012
	CodiPack	nvm	358	32	10	2.667	1	3	5

人間が確認する必要があることが多いが，コンパイルして実行できるので，速度やメモリ使用量の点で有利である．Adeptは大気海洋の研究者が作成して公開しているものである．次に登場したのは，演算子の多重定義やC++の言語機能を活用したものである．Stan Mathはベイズデータ解析用の言語のための数学ライブラリである．CoDiPackは並列計算が可能であり，ヘッダのみで構成されている．Baumgartner et al. (2019)は，欧州中期予報センター（ECMWF: European Centre for Medium-range Weather Forecasts）の全球予報システム（IFS: Integrated Forecast System）の雲スキームの勾配計算にCoDiPackを用いた．

PyTorchは機械学習で人気のあるライブラリの一つで，柔軟で使いやすいことで知られている．基本的な機能はlibtorchとしてC++で実装されており，Rのインターフェースもある．JAXは高性能計算を目的としたPythonライブラリである．スペクトル力学コアと機械学習とのハイブリッド大気大循環モデル（Kochkov et al. 2024）はJAXで記述されている．

Enzyme（Moses et al. 2020）は，コンパイラのプラグインで中間言語（IR: intermediate representation）の勾配を生成する．C/C++だけでなくFortranを含む様々な言語に適用可能で，コンパイルにより高速であることを特長としている．

4. 自動微分の適用例

この節では，非線型モデル（Lorenz 1963）と不連続モデル（Zhang et al. 2000）に対する4次元変分法データ同化に自動微分を適用した例について示す．

4.1 非線型モデル

Lorenz (1963)のモデルを用いて，自動微分の検証を行う．時間ステップの幅は0.01は数は200，誤差を考慮しない観測が60, 120, 180ステップ目に得られるものとする（Huang and Yang 1996）．

Tbl. 2に初期状態及びパラメタを同時を推定した結果を示す．基準とする随伴モデルを用いた場合でも，最適化手法やそれらのパラメタにより反復回数や誤差は異なる（Nash 2020）．

自動微分により求めた勾配計算には，torch for RとCodiPackを用いた．torch for Rの自動微分を勾配計算に，L-BFGSを最適化に用いた場合は随伴モデルとR組込のBFGSと同程度の精度であるが，この実験設定では強ウルフ条件（Nocedal and Wright 2006）を課す必要がある．勾配をCodiPackで計算し，最適化にnvmを用いた場合は，随伴を用いた場合と同様に機械精度で厳密に推定できている．この結果は，自動微分は勾配を正確に推定しており，推定の精度には最適化が重要であることを示唆している．

次にパラメタは正しい値に固定し，初期状態を推定する問題を解く．JAXで書いた順行モデルに対する自動微分を適用した結果は，随伴モデルを用いた結果（Huang and Yang 1996）を再現できる（Fig. 5）．いくつかのハードウェアで簡単なベンチマークを行った（Tbl. 3）．パソコンでは随伴モデルが桁違いに速いが，JAXもJust In Time compilation (JIT) により57倍高速化される．JAXはGPUやTPUを用いるとJITの効果がさらに大きく，T4では5350倍，A100では498倍，v6e1 TPUでは1228倍高速化され，T4とA100では随伴モデルの実行時間を大きく下回った．一方TPUでは随伴モデルの方がJITありJAXよりも速い．Macでは，EnzymeはJITありJAXの5054倍速い．単一スレッドだが，コンパイルの効果は大きい．

4.2 不連続モデル

自動微分が不連続なモデルにも適用できるか確認する．ここでは，閾値において時間変化が異なるモデル（Zhang et al. 2000）を用いる．

$$f(x) = \frac{\partial x}{\partial t} = \begin{cases} f_1(x) = 2x - 2 & x < 1 \\ f_2(x) = x - 4 & x \geq 1 \end{cases} \tag{26}$$

このモデルを時間離散化すると

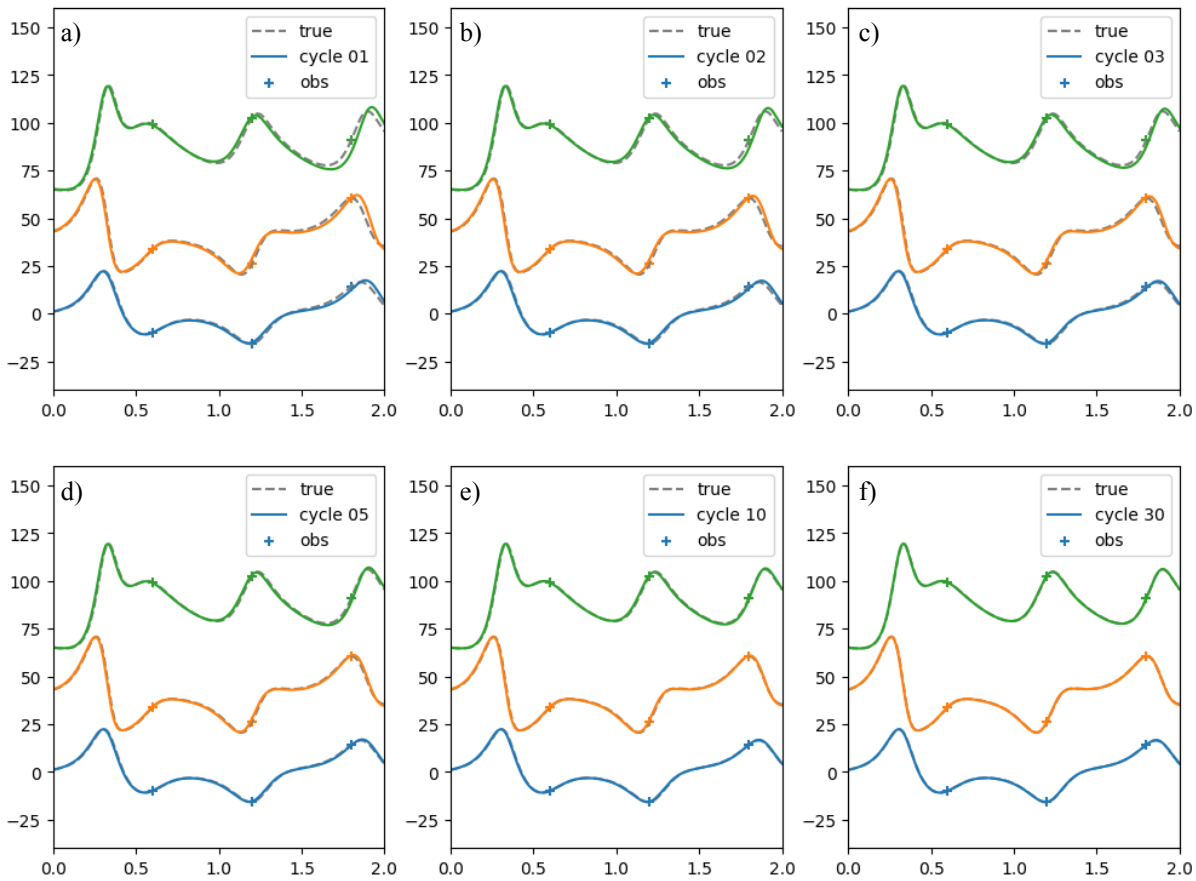

Fig. 5 Variational assimilation of the observations at $t = 0.6, 1.2, 1.8$ (step 60, 120, 180) using gradients with automatic differentiation in JAX. Cycles a) 1, b) 2, c) 3, d) 5, e) 10, and f) 30 are shown. Blue, orange, green, grey dotted curves represent X, Y, Z and true values. Plus marks are observations.

Tbl. 3 Benchmark for the initial state estimation with %timeit.

gradient	Hardware	time
hand-coded adjoint	M1 MacBook Pro 2021 32GB	92.5 ms
JAX without JIT		263 s
JAX with JIT		4.64 s
Enzyme AD		918 µs
	AMD Ryzen 5 5600G 32GB	279 µs
hand-coded adjoint	Colab T4	1.09 s
JAX without JIT		626 s
JAX with JIT		117 ms
hand-coded adjoint	Colab A100	641 ms
JAX without JIT		319 s
JAX with JIT		44.8 ms
hand-coded adjoint	Colab v6e1 TPU	162 ms
JAX without JIT		199 s
JAX with JIT		258 ms

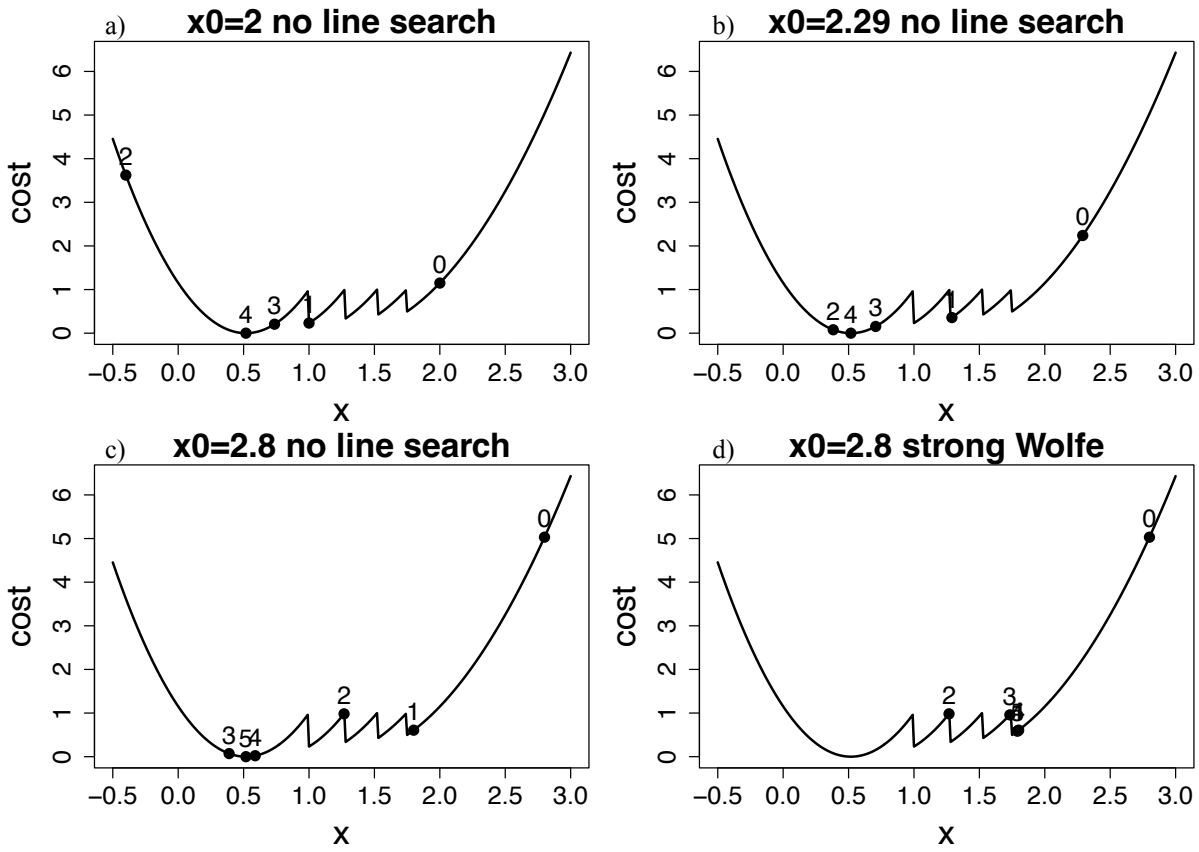

Fig. 6 Minimization of the discontinuous cost function from different first guesses: a) $x_0 = 2$, b) $x_0 = 2.29$, c, d) $x_0 = 2.8$ without (a-c) and with (d) the strong Wolfe line search.

$$x_{n+1} = x_n + f(x)\Delta t \tag{27}$$

同化窓は時間ステップ4つ分

$$t_\mathrm{R} = t_0 + 4\Delta t, \Delta t = 0.1 \tag{28}$$

とし，目的函数は二乗量とする．

$$J_1(x_0) = x^2(t_\mathrm{R}) \tag{29}$$

目的函数の初期時刻における勾配は次のように表される．

$$\begin{aligned}\nabla J_1(x_0) &= \frac{\partial x^2(t_\mathrm{R})}{\partial x_0} = 2x(t_\mathrm{R})\frac{x(t_\mathrm{R})}{\partial x_0} \\ &= x(t_\mathrm{R})\frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial x_1}\frac{\partial x_1}{\partial x_0} \\ &= x(t_\mathrm{R})\prod_{i=0}^{3}\left(1 + \Delta t \left.\frac{\partial f}{\partial x}\right|_{x_i}\right)\end{aligned} \tag{30}$$

勾配の計算にはtorch for Rの自動微分，最適化にはL-BFGSを用いた．目的函数はモデルの不連続の影響で鋸型となっており，最小値0.5の他に複数の極小が存在する（Fig. 6）．$x_0 = 2, 2.29$から（a, b）では線型探索による差異は見られないが，$x_0 = 2.8$は線型探索なしでは4ステップ目以降1.797で停滞する．

Tbl. 4 The number of successful convergence to the global minimum at 0.5 from x_0 varied between 0.01 and 3.

δx		0.5	1.0	1.5	2.0	2.5	3.0	3.1	3.2	3.3	3.4	3.5
no restart	no line search	300	300	300	299	300	298	300	300	299	300	299
	strong Wolfe	300	300	300	293	299	275	283	280	273	274	266
restart	no line search	300	300	300	300	300	300	300	300	299	300	300
	strong Wolfe	300	300	300	300	300	300	300	300	300	298	290

次に閾値での跳びに対する依存性について調べた．

$$f_2(x) = x - (1 + \delta x) \tag{31}$$

として δx を0.5から3.0まで0.5刻み、3.0から3.5まで0.1刻み，x_0を0.01から3まで0.01刻みに変化させた（Tbl. 4）．線型探索ありよりもなしの方が最小解に収束しない回数が少ない．線型探索なしの場合は跳びに対する依存性がほとんど見られないが，線型探索ありの場合は跳びが大きいほど収束しなくなる．最急降下方向から最適化をやり直すリスタートを最大10回許容すると，線型探索した場合でもより広い範囲で正しい値に収束するようになる．リスタートにより，停滞が打ち破られるためであると考えられる．いずれにしても，自動微分による勾配の計算は不連続なモデルにも適用できるが，収束は最適化手法に依存する．

5. まとめと今後の展望

本稿では，随伴を導入し，ラグランジュの未定乗数法による随伴モデルの構築方法について述べた．また，複素ステップ微分や自動微分の原理について紹介し，非線型モデルや不連続モデルの4次元変分法データ同化に適用した例を示した．

ラグランジュの未定乗数法に基づく導出は，冗長になるが順行モデルのコードから機械的に生成が可能で計算グラフを用いる自動微分と類似している．明確なルールがあるため，大規模言語モデル（いわゆる生成AI）に指示して自動化することも可能であろう（亀ら 2025）．

複素ステップ微分は自由度の少ない系の接線型モデルに限られるものの，機械精度で正確な微分が求められるため，検証に有用である．

自動微分は随伴よりもオーバーヘッドが大きいことが多いが，GPUやTPUを利用してJITを使うことにより実用的速度が得られ，随伴を上回る性能が得られる場合もある．Enzymeは群を抜いて実行時間が短い．Fortranからの利用や，並列化への対応（Moses et al. 2022）が成熟し，モデルの勾配計算が容易かつ高速に行えるようになることが期待される．

自動微分をうまく活用すれば，高次元のモデルでも随伴モデルを構築することなく，変分法データ同化やネットワークの学習ができる．現在，JAXで書かれた準地衡流モデル（榎本・中下 2022）や大気大循環モデル（Kochkov et al. 2024）に自動微分を適用して，データ同化や感度解析を試みている．自動微分は随伴作成の負担軽減のために簡略化した部分も含めることができるため，より順行モデルに忠実な随伴であると考えられる．モデルの誤差発展における各過程の寄与を定量化し，現象のメカニズムと予測可能性の理解につながることが期待される．

謝辞

本研究はJSPS科研費24H02226の助成を受けた．

Lorenz (1963)モデルの随伴は海洋データ同化夏の学校2025に際して準備したもので，https://github.com/tenomoto/dass2025から取得できる．

参考文献

榎本剛・中下早織 (2022): 準地衡流モデルへの決定論的アンサンブルデータ同化．京都大学防災研究所年報, Vol. 65B, pp. 126–133.

亀伸樹・平原和朗・加納将行・岡崎智久 (2025), データ同化Pythonコードの自動生成 ──大規模言語モデルのプログラミングへの適用──, Vol. 78, No. 2, pp. 13–30.

Amari, S. (1967): A theory of adaptive pattern classifiers. IEEE Transactions on Electronic Computers, Vol. EC-16, No. 3, pp. 299–307, doi:10.1109/PGEC.1967.264666.

Amari, S. (1993): Backpropagation and stochastic gradient descent method. Neurocomputing, Vol. 5, No.4, pp. 185–196, doi:10.1016/0925-2312(93)90006-O.

Baumgartner, M., M. Sagebaum, N. R. Gauger, P. Spichtinger, and A. Brinkmann (2019): Algorithmic differentiation for cloud schemes (IFS Cy43r3) using CoDiPack (v1.8.1). Geoscientific Model Development, 12, pp. 5197–5212, doi:10.5194/gmd-12-5197-2019.

Baydin, A. G., B. A. Pearlmutter, A. A. Radul, and J. M. Siskind (2018): Automatic differentiation in machine learning: a survey. J. Mach. Learn. Res., Vol. 18, No. 153, pp. 1–43.

Bischof, C., A. Carle, P. Khademi, and A. Mauer (1994): The ADIFOR 2.0 system for the automatic differentiation of Fortran 77 programs. Center for Researh on Parallel Computation, Rice University, https://www.mcs.anl.gov/research/projects/adifor/.

Cerviño, L. I., and T. R. Bewley (2003): On the extension of the complex-step derivative technique to pseudospectral algorithms. Journal of Computational Physics, 187, pp. 544–549, doi:10.1016/S0021-9991(03)00123-2.

Carpenter, B., M. D. Hoffman, M. Brubaker, D. Lee, P. Li, and M. Betancourt (2015): The Stan math library: reverse-mode automatic differentiation in C++, pp. 1–96, doi:10.48550/arXiv.1509.07164.

Enomoto, T., and S. Nakashita (2024): Application of exact Newton optimisation to the maximum likelihood ensemble filter. Tellus A, Vol. 76, No. 1, pp. 42–56, doi:10.16993/tellusa.3255.

Frostig, R., M. J. Johnson, D. Maclaurin, A. Paszke, and A. Radul (2021): Decomposing reverse-mode automatic differentiation, pp. 1–3, doi:10.48550/arXiv.2105.09469.

Fujii, Y., H. Tsujino, N. Usui, H. Nakano, and M.

Kamachi (2008): Application of singular vector analysis to the Kuroshio large meander. J. Geophys. Res. Oceans, 113, pp. 1–17, doi:10.1029/2007 JC004476.
Hascoet, L., and V. Pascual (2013): The Tapenade automatic differentiation tool: Principles, model, and specification. ACM Trans. Math. Softw., 39, pp. 20:1-20:43, doi:10.1145/2450153. 2450158.
Hogan, R. J. (2014): Fast Reverse-Mode Automatic Differentiation using Expression Templates in C++. ACM Trans. Math. Softw., 40, pp. 26:1-26:16, doi:10.1145/2560359.
Huang, X.-Y., and X. Yang (1996): Variational data assimilation with the Lorenz model. HIRLAM technical report, No. 26, Norrköpig, pp. 1–42, April 1996, https://www.researchgate.net/publication/2465513_Variational_Data_Assimilation_With_The_Lorenz_Model,.
Iri, M. (1984): Simultaneous computation of functions, partial derivatives and estimates of rounding errors —Complexity and practicality—. Japan J. Appl. Math., Vol. 1, No. 2, pp. 223–252, doi:10.1007/BF03167059.
Iri, M. (1991): History of automatic differentiation and rounding error estimation. Automatic differentiation of algorithms, Greiwank, Andreas and Corliss, George F., Ed., The first SIAM Workshop on Automatic Differentiation, Breckenridge, CO, USA, Society for Industrial and Applied Mathematics, pp. 3–16.
Kochkov, D., and Coauthors (2024): Neural general circulation models for weather and climate. Nature, 632, pp. 1060–1066, doi:10.1038/s41586-024-07744-y.
Lagrange, J. L. 1763. Solution de différents problèmes de calcul intégral. Miscellanea Taurinensia Vol. 3, pp. 179–380.
Lawson, L. M., Y. H. Spitz, E. E. Hofmann, and R. B. Long (1995): A data assimilation technique applied to a predator-prey model. Bull. Math. Biol., Vol. 57, No. 4, pp. 593–617, doi:10.1016/S0092-8240(05) 80759 -1.
Lorenc, A. C. (1986): Analysis methods for numerical weather prediction. Quart. J. Roy. Meteor. Soc., Vol. 112, No. 474, pp. 1177–1194.
Lorenz, E. N. (1963): Deterministic nonperiodic flow. J. Atmos. Sci., Vol. 20, No. 2, pp. 130–141, doi:10.1175/1520-0469(1963)020%253C0130:DNF%253E2.0.CO;2.
Lyness, J. N. (1967): Numerical algorithms based on the theory of complex variable. Proc. the 1967 22nd national conference, ACM '67, New York, NY, USA, Association for Computing Machinery, pp. 125–133, doi:10.1145/800196.805983.
Lyness, J., and C. Moler (1967): Numerical Differentiation of Analytic Functions. SIAM J. Numer. Anal., Vol. 4, No. 2, pp. 202–210, doi:10.1137/ 0704019.
Luchini, P., and A. Bottaro (2014): An Introduction to Adjoint Problems. Annu. Rev. Fluid Mech., Vol. 46, pp. 493–517, doi:10.1146/annurev-fluid-010313-141253.
Margossian, C. C. (2019): A review of automatic differentiation and its efficient implementation. WIREs Data Mining and Knowledge Discovery, Vol. 9, No. 4, e1305, pp. 1–19, doi:10.1002/widm.1305.
Moses, W., and V. Churavy (2020): Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. Advances in neural information processing systems, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, Eds, Vol. 33 of, Curran Associates, Inc., pp. 12472–12485.
Moses, W. S., S. H. K. Narayanan, L. Paehler, V. Churavy, M. Schanen, J. Hückelheim, J. Doerfert, and P. Hovland (2022): Scalable automatic differentiation of multiple parallel paradigms through compiler augmentation. SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–18, doi:10.1109/SC41404.2022.00065.
Nash, J. C. (2020): Provenance of R's Gradient Optimizers. R J., Vol.12, No. 1, 477–483.
Navon, I. M., and D. M. Legler (1987): Conjugate-gradient methods for large-scale minimization in meteorology. Mon. Wea. Rev., Vol. 115, No. 8, pp.1479–1502, doi:10.1175/1520-0493(1987)115%253 C1479:CGMFLS%253E2.0.CO;2.
Nocedal, J. and Stephen J. Wright (2006): Numerical optimization. 2nd edn Springer 664pp., https://doi.org/10.1007/978-0-387-40065-5.
Paszke, A., and Coauthors (2017): Automatic differentiation in PyTorch. 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA, pp. 1–31.
Sagebaum, M., T. Albring, and N. R. Gauger (2018): Expression templates for primal value taping in the reverse mode of algorithmic differentiation. Optimization Methods and Software, 33, pp. 1207–1231, doi:10.1080/10556788.2018.1471140.
Squire, W., and G. Trapp (1998): Using complex variables to estimate derivatives of real functions. SIAM Rev., Vol. 40, No. 1, pp. 110–112, doi:10.1137/S003614459631241X.
Zhang, S., X. Zou, J. Ahlquist, I. M. Navon, and J. G. Sela (2000): Use of Differentiable and Nondifferentiable Optimization Algorithms for Variational Data Assimilation with Discontinuous Cost Functions. Mon. Wea. Rev., 128, pp. 4031–4044, doi:10.1175/1520-0493(2000)129%253C4031:UODANO%253E2.0.CO;2.

（論文受理日：2025年8月31日）

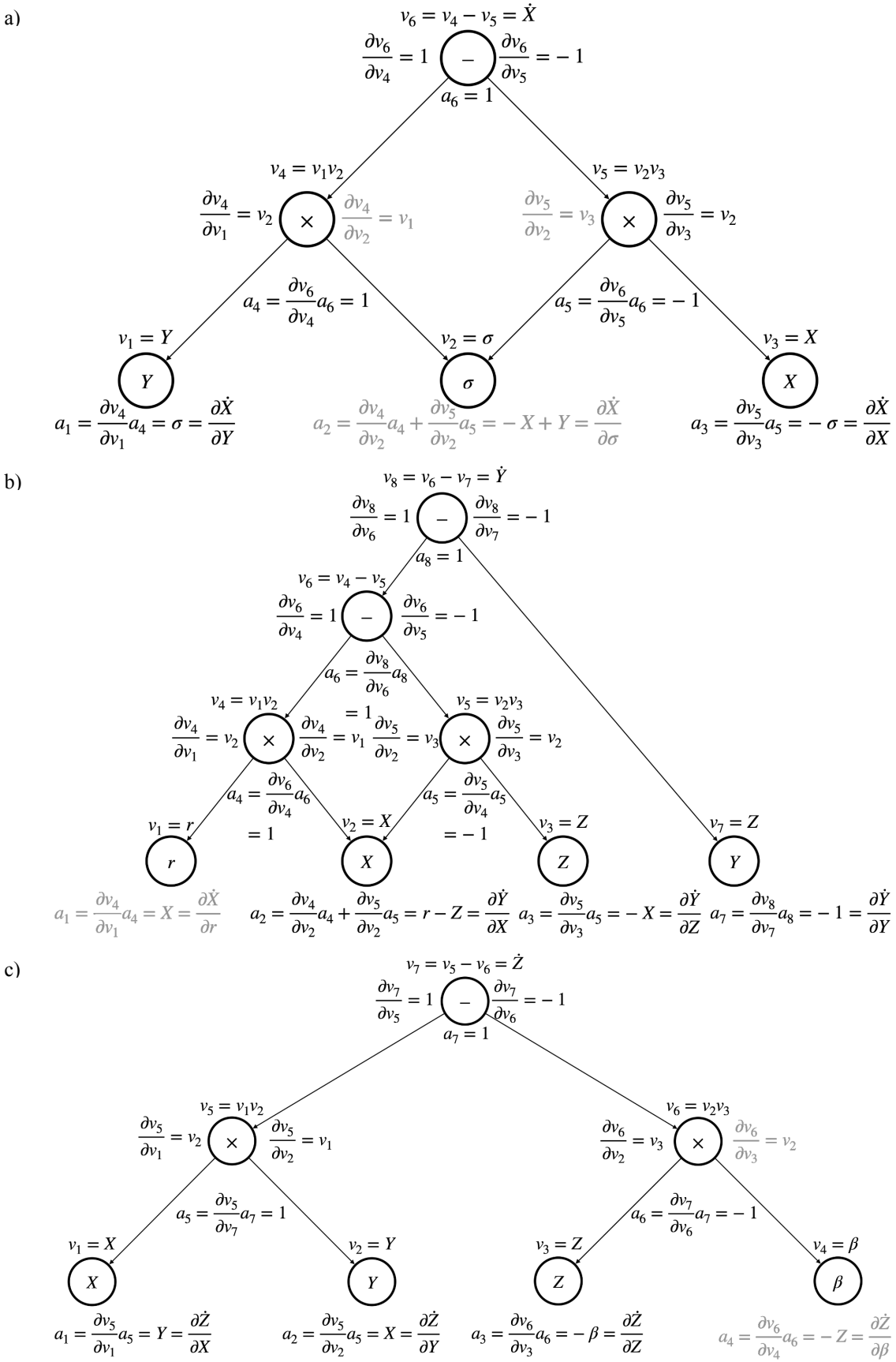

Fig. A1 As in Fig. 4 but for the computational graphs for the three equations of Lorenz (1963) model. Panels a), b), and c) corresponds to $\dot{X}$, $\dot{Y}$, and $\dot{Z}$, respectively. Adjoints for parameters can be omitted for state-only estimation and are coloured in grey.