Acta Physica Sinica

自动微分以及它在物理模拟中的应用*

刘金国[1)]　许开来[2)]

1) (哈佛大学物理系, 坎布里奇　02138)

2) (斯坦福大学, 斯坦福　94305)

自动微分是利用计算机自动化求导的技术，最近几十年因为它在机器学习研究中的应用而被很多人了解。如今越来越多的科学工作者意识到高效的，自动化的求导可以为很多科学问题的求解提供新的思路，其中自动微分在物理模拟中的应用尤为重要且具有挑战性。物理系统的可微分模拟可以帮助解决混沌理论，电磁学，地震学，海洋学等领域中的很多重要问题，但又因为其对计算时间和空间的苛刻要求而对自动微分技术本身提出了挑战。本文将会回顾将自动微分技术运用到物理模拟中的几种方法，并横向对比它们在计算时间，空间和精度方面的优势和劣势。这些自动微分技术包括伴随状态法，前向自动微分，后向自动微分以及可逆计算自动微分。

关键词: 自动微分，科学计算，可逆计算，最优检查点，物理模拟

PACS: 02.60.Pn, 02.30.Jr, 91.30.-f

1 名 词 表

伴随变量 adjoint

核函数 kernel function

原子函数 primitive function

可逆计算 reversible computing

可逆编程 reversible programming

鹅卵石游戏 pebble game

伴随状态法 adjoint state method

自动微分 automatic differentiation (AD)

---

*

† 通信作者. E-mail: jinguoliu@g.harvard.edu

前向自动微分 forward mode AD

后向自动微分 reverse mode AD

地震学 seismic

检查点 checkpoint

拓展动力学 augmented dynamics

损失 loss

控制流 control flow

辛积分器 symplectic integrator

洛伦茨 Lorenz

龙格库塔 Runge-Kutta

2 引　言

自动微分 [1]是指自动获取一段计算机程序导数的技术，很多人对它的了解源自它在机器学习中的成功应用，人们可以用它优化包含数千亿参数的神经元网络 [2]。与很多人印象不同的是，自动微分其实是个很古老的技术。 Nolan 早在他1953年的博士论文中就提出过计算机自动化求导的构想 [3]，后来针对这一构想又出现了两种不同的实践，分别是1964年由 Wengert 实现的前向自动微分 [4]和1970年 Linnainmaa 实现的后向自动微分 [5]。而最近十几年，由于后向自动微分在机器学习中的广泛应用，相关技术越来越成熟并在科学计算中的也得到了越来越多的应用。科学家们利用方便的，自动化的计算机辅助求导解决了很多重要的物理问题，其中包括帮助变分蒙特卡洛设计更加通用的多体物理波函数 [6, 7, 8, 9]，加速变分量子算法的模拟 [10]，拓展变分张量网络算法 [11]，以及求解自旋玻璃基态构型 [12]等。

对物理模拟过程的自动微分是自动微分重要的应用之一，其中最大的技术挑战是对电磁学，海洋学 [13]和地震学 [14, 15]等问题中最核心的微分方程求解过程的自动微分。这些微分方程的常见的求解方法是先将问题的时空坐标离散化，并以数值积分的形式完成求解。要得到精确的结果，离散化后的网格需要设计的很稠密，从而对存储空间和计算时间的需求巨大。与此同时，在自动微分计算过程中，机器学习中主流的后向自动微分库，如 Jax [16] 和 PyTorch [17]，需要很多额外的空间来缓存中间结果，导致人们经常无法直接用它们对具有一定规模的实际物理模拟问题求导。一种传统的解决常微分方程求导的方案叫做伴随状态法 [18, 19]，它假设了在较短时间内积分器可逆，并通过逆向积分来帮助自动微分回溯状态。事实上，除了 Leapfrog 等少数辛积分器可外，大多数的积分器并不能保证严格的时间反演对称，所以伴随状态法往往存在与积分步长有关的系统性误差。后来，有人把机器学习中的最优检查点算法带入到了物理模拟的反向自动微分中 [14]，仅引入对数的额外时间和空间开销就做到了严格求导。也有一些人直接利用基于最优检查点算法 [20, 21]的自动微分编译器，如 Tapenade [22], OpenAD [23] 和 TAF [13] 等，编译模拟代码实现类似的时间和空间的权衡。在最近几年，可逆计算开始被用作一种新的全编程语言的微分的方案

[24]，它提供了一种不同的时间和空间的交换方案叫做 Bennett 算法 [25]，可以做到对数的额外空间开销，但是对应的时间额外开销为多项式。可逆计算的优势在与提供了传统自动微分框架所不具有的内存控制的灵活性，因此可以被用来微分 GPU 核函数，复杂的控制流等。

本文将会回顾共轭态方法，前向自动微分以及基于最优检查点算法和可逆计算的后向自动微分等自动微分方法在处理物理模拟问题中的应用，并对比不同方法的优劣以及适用的场景。章节3介绍了几种自动微分方法的基本理论。章节4介绍了不同自动微分技术在波的传播模拟中的应用。关于如何在后向自动微分中权衡程序的运行时间和空间的理论，虽然重要，但受篇幅限制我们将其放在附录。

3 自动微分方法

物理模拟过程的常见求解方案是先将偏微分方程的空间部分离散化并作差分处理 [26]，从而将其转换为对时间的常微分方程

$$\frac{\mathrm{d}s}{\mathrm{d}t} = f(s, t, \theta)$$

其中s为状态，t为时间，θ为控制参数。一个常微分方程的数值求解器会把时间维度离散化，作n步叠代求解，每步仅做从时刻t_i到时刻$t_{i+1} = t_i + \Delta t$的演化。

$$\begin{aligned} s_n &= \mathrm{ODESolve}(f, s_0, \theta, t_0, t_n) \\ &= (s_{i+1} = \mathrm{ODEStep}(f, s_i, \theta, t_i, \Delta t) \text{ for } i = 0, 1, \ldots, n-1) \end{aligned} \tag{1}$$

其中s_i为完成第i步积分后的状态。为了简便，下文我们将单步运算简记为$\mathrm{ODEStep}(s_i)$。最后我们还会定义一个损失函数$\mathcal{L} = \mathrm{loss}(s_n)$。自动微分的目标则是求解损失量对输入状态$\frac{\partial \mathcal{L}}{\partial s_0}$和控制参数的导数和$\frac{\partial \mathcal{L}}{\partial \theta}$。

3.1 共轭态方法

共轭态方法 [18, 19]是专门针对常微分方程的反向传播方法。在研究中，人们发现积分过程的导数的反向传播同样是一个积分过程，只不过方向相反。于是人们通过构造一个可以同时回溯函数值和回传导数的拓展函数，以对拓展函数的逆向积分的形式完成导数的计算，如算法 1所示。该算法的描述来自文献 [19]，其中可以找到详细的推导过程，这里对原算法中的符号做了替换以方便读者理解。算法中的$\frac{\partial q}{\partial s}$，$\frac{\partial q}{\partial \theta}$以及$\frac{\partial \mathcal{L}}{\partial s_n}$为局域导数，它们可以通过手动推导或者借助其它自动微分库来实现。该方案在积分器严格可逆的时候梯度也严格，而当积分器反向积分误差不可忽略时，则需要额外的处理保证误差在可控范围，这

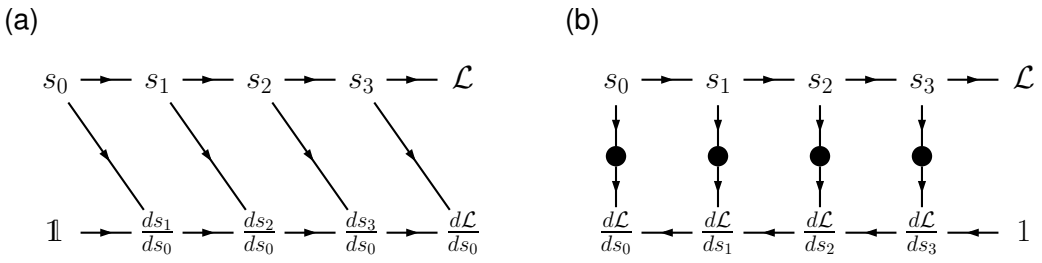


图 1: 物理模拟程序中对状态的 (a) 前向自动微分和 (b) 后向自动微分过程。其中线条上的箭头代表运算的方向，圆圈为后向自动微分中缓存的变量。

一点我们会在随后的例子中涉及。

算法 1: 伴随状态法

输入: 动力学参数θ，开始时间t_0，结束时间t_n，末态s_n，以及需要回传的导数$\frac{\partial \mathcal{L}}{\partial s_n}$

输出: $\frac{\partial \mathcal{L}}{\partial s_0}$, $\frac{\partial \mathcal{L}}{\partial \theta}$

function aug_dynamics($(s,\ a,\ \text{-}),\ t,\ \theta$)
  $q = f(s, t, \theta)$ # 定义拓展动力学函数
  return $(q,\ -a^T \frac{\partial q}{\partial s},\ -a^T \frac{\partial q}{\partial \theta})$
end
$S_0 = (s_n,\ \frac{\partial \mathcal{L}}{\partial s_n},\ 0)$ # 计算拓展动力学函数的初始状态
$(s_0,\ \frac{\partial \mathcal{L}}{\partial s_0},\ \frac{\partial \mathcal{L}}{\partial \theta}) = \text{ODESolve}(\text{aug\_dynamics},\ S_0,\ \theta,\ t_n,\ t_0)$ # 对拓展动力学反向积分

3.2 前向自动微分

顾名思义，前向自动微分是指向前（指与程序运行方向相同）传播导数。这里导数传播的规则和数学分析中的无穷小量的运算规则有关。 数学中，在对一个输入变量p求导时，会让它携带一个无穷小量$\mathrm{d}p$，并通过对这个无穷小量的运算完成对程序的求导。比如当作用函数f时，会有如下链式法则

$$f\left(\vec{x} + \frac{\mathrm{d}\vec{x}}{\mathrm{d}p}\mathrm{d}p\right) = f(\vec{x}) + \left(\frac{\mathrm{d}f(\vec{x})}{\mathrm{d}\vec{x}}\frac{\mathrm{d}\vec{x}}{\mathrm{d}p}\right)\mathrm{d}p \tag{2}$$

其中$\vec{x}$为输入函数f的参数的集合，可以包括p本身。$\frac{\mathrm{d}f(\vec{x})}{\mathrm{d}\vec{x}}$为局域雅可比矩阵，前向传播中我们将它与梯度矢量相乘得到新的梯度矢量。实际程序实现中，这个局域雅可比矩阵并不需要构造出来，考虑到任何程序都具有可拆分为基础指令的特点，人们把程序拆解为基础标量指令，并在这些基础指令上通过代码变换或者是算符重载的方式实现梯度矢量的计算。以标量的乘法为例，变量会同时记录它的数值和一阶小量的系数$(v, \dot{v})$，其中$\dot{v} = \frac{\mathrm{d}v}{\mathrm{d}p}$，人们重新定义它的基本运算规则如下

$$\dot{*} : ((a, \dot{a}), (b, \dot{b})) \mapsto (a * b, a\dot{b} + b\dot{a})$$

使得其在计算同时更新一阶小量的系数，而自动微分所要做的就是将所有的运算指令都引入小量的系数并

作上述运算规则的替换。简单的运算规则的替换对于人类来说尚可手动，但真实的程序可能会包含数以亿计的这样的基础操作，虽然结果依然是解析的，但是人们很难再通过人力得到具体的导数表达式，而计算机恰恰很擅长这样繁琐但是规则简单的任务。如图 1 (a) 所示，在求解常微分方程中，单步前向自动微分可以形式化的表达为

$$\mathrm{ODEStep}^{\mathrm{F}}:\left(s_i,\frac{\mathrm{d}s_i}{\mathrm{d}s_0},\frac{\mathrm{d}s_i}{\mathrm{d}\theta}\right)\mapsto\left(\mathrm{ODEStep}\left(s_i\right),\frac{\partial s_{i+1}}{\partial s_i}\frac{\mathrm{d}s_i}{\mathrm{d}s_0},\frac{\partial s_{i+1}}{\partial s_i}\frac{\mathrm{d}s_i}{\mathrm{d}\theta}+\frac{\partial s_{i+1}}{\partial\theta}\right)$$

这里为了简洁略去了时间等常数参量。由于状态s_i, s_{i+1}和控制参数θ均可包含多个变量，上述偏微分均解释为雅可比行列式。一般前向自动微分在一次前向运行中只对一个或者若干个变量求导，因此计算空间会随着一次求导的变量数目线性增加。无论是一次求导多少个变量，前向自动微分求导的时间都会随着需求导的变量的数目线性增长，这是限制前向自动微分应用场景的最主要因素。

3.3 后向自动微分

后向自动微分与前向自动微分梯度传播方向相反，它解决了前向自动微分中计算开销随着需要求导的变量数目线性增长的问题。后向自动微分包括正向计算和梯度反向传播两个过程。正向计算过程中，程序进行普通的计算并获取所需的运行时信息，最后得到一个标量损失$\mathcal{L}$。梯度回传的过程是计算导数的过程，可表示为更新变量的梯度$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}v}$的过程。从$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\mathcal{L}}=1$出发，梯度对应如下链式法则

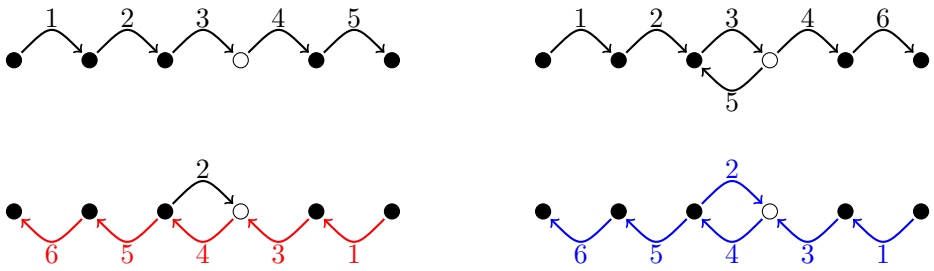
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\vec{x}}=\sum_i\frac{\partial\mathcal{L}}{\partial\vec{y}_i}\frac{\mathrm{d}\vec{y}_i}{\mathrm{d}\vec{x}}$$

为了实现该链式法则，软件包设计者们一般会在软件中规定一类可微分的函数集合，叫做原子函数$\{f_p\}$，并定义了梯度在原子函数$\vec{y}=f_p(\vec{x})$上的回传规则，可表示为伴随变量$\overline{v}=\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}v}$的更新规则

$$\overline{f}_p:(\overline{x},\overline{y})\mapsto(\overline{x}+\overline{y}\frac{\partial\vec{y}}{\partial\vec{x}},\overline{y})$$

其中，$\frac{\partial\vec{y}}{\partial\vec{x}}$为局域雅可比矩阵，它的数值不需要具体计算出来，而是通过后向传播函数实现。根据实现不同，这些原子函数可以是 Tapenade 和 NiLang [24] 那样仅包含有限个基础标量指令，也可以是向 Jax 和 Pytorch 中那样的可拓展的常用函数集合。用户将程序表示为原子函数的组合，在前向计算中，计算机将遇到的原子函数的后向传播函数连同所需的中间变量一起存放在堆栈中*，并在反向传播中按照后进先出的顺序调用，这样代码便可以利用上述梯度回传规则更新伴随变量。具体到求解常微分方程的自动微分过程中，其反向传播过程如图 1 (b) 所示，可以形式化的表达为

*除了堆栈方案，也有一些软件包通过静态编译等手段的实现状态回溯。



(a) (b)

图 2: (a) 检查点方案和 (b) 反计算方案避免缓存全部中间状态。图中黑箭头为正常计算过程，红箭头为梯度回传过程，蓝箭头为梯度回传和反向计算的复合过程，箭头上的数字代表了执行顺序。黑色和白色的圆点为被缓存和未被缓存 (或通过反计算消除) 的状态。

$$\overline{\text{ODEStep}}^{\text{B}} : (\overline{s_{i+1}}, \overline{\theta}, s_i) \mapsto \left(\overline{s_{i+1}} \frac{\partial s_{i+1}}{\partial s_i}, \overline{\theta} + \overline{s_{i+1}} \frac{\partial s_{i+1}}{\partial \theta} \right)$$

# 正向计算（单步）	# 反向传播（单步）
$\texttt{push}(\Sigma, s_i)$	$s_i = \texttt{pop}(\Sigma)$
$s_{i+1} = \text{ODEStep}(s_i)$	$(\overline{s_i}, \overline{\theta}) = \overline{\text{ODEStep}}^{B}(\overline{s_{i+1}}, \overline{\theta}, s_i)$

这里也同样略去了时间等参量，`push` 和 `pop` 分别是对全局堆栈Σ的入栈和出栈操作。虽然导数回传的计算复杂度与需要求导的变量数目无关，但后向自动微分向堆栈中存储数据带来了正比于计算步骤数的额外空间开销。如何设计一个可以对计算状态逆序的访问的算法，使得缓存带来的时间和空间开销更小，是反向自动微分设计复杂性的来源，也被叫做“时间与空间的权衡”问题。实际应用中，人们可以用检查点 [20]或者反向计算 [24]技巧来避免缓存全部中间结果。如图 2 (a) 所示的检查点方案中，正向计算中程序可以选择性的缓存（黑色圆点）或者不缓存部分状态（空心圆点），这些被缓存的状态被称作检查点。在图中下方的反向传播过程中，当需要的数据没有被缓存时，程序会从最近的检查点出发重新计算该数据（下方步骤2）。图 (b) 所示的反计算方案是可逆计算中常用的避免缓存的方案。当运算在数学层面不可逆，可逆计算要求保留输入状态以使其在程序层面可逆，因此默认也需要缓存每步的结果。而反计算通过反向运行（上方步骤5）清零已分配的内存的方式并将空间资源归还给系统控。在反向传播过程中，执行顺序和前向镜像对称，每个指令变成原指令的逆，因此可以自然的获得运行时的状态信息。不论是检查点方案还是可逆计算方案，节省内存都需要消耗更多的时间，那么如何权衡两者的额外开销才是最好呢？我们在附录中引入一个简单的理论模型叫做鹅卵石游戏并详细讨论了如何在检查点方案中用Treeverse算

方法	时间	空间	是否严格
伴随状态法	$\mathcal{O}(T)$	$\mathcal{O}(S)$	否
前向自动微分	$\mathcal{O}(NT)$	$\mathcal{O}(S)$	是
基于最优检查点的后向自动微分	$\mathcal{O}(T\log T)$	$\mathcal{O}(S\log T)$	是
基于可逆计算的后向自动微分	$\mathcal{O}(T^{1+\epsilon})$	$\mathcal{O}(S\log T)$	是

表 1: 不同自动微分方案中，时间与空间复杂度的关系。这里伴随状态法没有考虑了缓存部分中间结果以保证反向积分的正确性，考虑止之后应为$\mathcal{O}(TS)$。前向自动微分时间复杂度中的N代表了需要求导的参数个数。可逆计算中的多项式时间复杂度中的$\epsilon > 0$。

法 [20]（也叫做最优检查点算法）实现仅需对数的额外时间和空间逆向遍历状态，以及如何在可逆计算中用Bennett算法 [25]实现对数的额外空间和多项式的额外时间实现逆向遍历状态，这里仅把基本结论列于表 1中。

4 自动微分在物理模拟中的应用

本章节有两个案例，其一是用前向自动微分和伴随状态法求解参数较少的洛伦茨系统模拟的导数，其二是用最基于最优检查点算法和可逆计算的后向微分对步骤较多且内存消耗巨大的地震学模拟的求导。这些例子的源代码可以在 Github 仓库中找到：https://github.com/GiggleLiu/WuLiXueBao。

4.1 洛伦茨方程求解

洛伦茨系统 [27]是人们研究混沌问题的经典模型，它描述了一个定义在三维空间中的动力学

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y-x),\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= x(\rho-z)-y,\\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy-\beta z.\end{aligned}$$

其中，σ, ρ和β为三个控制参数。该系统的含时演化的曲线如图 3 (b)所示。在$\rho > 1$时，系统有两个吸引子 [28]，但仅当$\rho < \sigma\frac{\sigma+\beta+3}{\sigma-\beta-1}$时才会出现图 (b) 中橘色曲线所示的粒子稳定的围绕其中一个吸引子运动的情况，这时候系统较为稳定并表现处对初值较为不敏感的特点。末了位置坐标对控制参数和初始坐标的导数反映了末态对控制参数和初始位置的敏感度，在一定程度的反映了系统出现了浑沌现象。在数值模拟中，我们用4阶龙格库塔方法对时间积分并得到末了位置，模拟中固定初始位置为$(x_0, y_0, z_0) = (1, 0, 0)$，控制参数为$\beta = 8/3$，积分时间区间为$[0, T = 30]$以及积分步长为$3 \times 10^{-3}$。由于该过程所含参数仅有6个，包括初始位置的三个坐标(x_0, y_0, z_0)和三个控制参数(σ, ρ, β)，因此用前向自动微分工具 ForwardDiff [29] 求导比起后向自动微分有很大优势。我们把导数的绝对值的平均与初始ρ, σ的关系画在图 3中。可以看出只有在理论预言的黑线下方才会有较小的导数，说明稳定吸引子参数下系统动力学的确对初值依赖性较低。

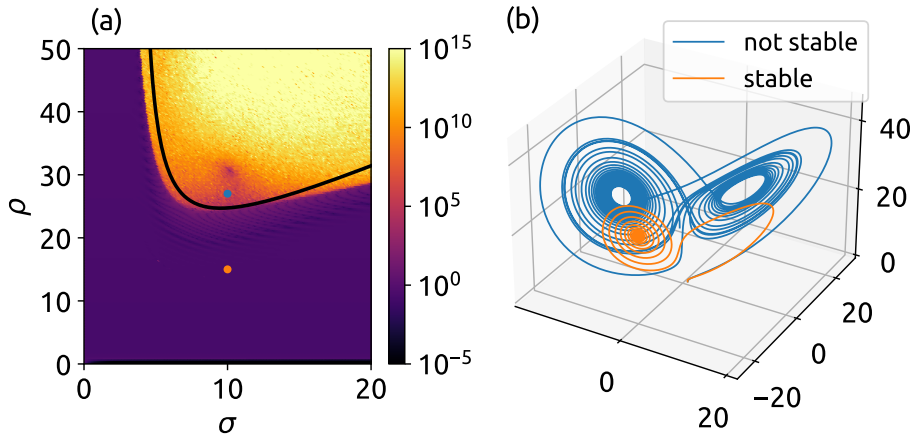


图 3: (a) 洛伦茨系统中固定参数$\beta = 8/3$，梯度大小与参数ρ和σ的关系。图中颜色代表了平均梯度大小，黑线是理论上会不会存在稳定吸引子分界线。(b) 中的蓝色和黄色的线分别对应 (a) 图标识的蓝点处参数$(\sigma = 10, \rho = 27)$ 和黄点处参数$(\sigma = 10, \rho = 15)$ 对应的动力学模拟。

方法	Julia	ForwardDiff	NiLang	Neural ODE + NiLang
时间	1.90ms	2.88ms	6.70ms	34.3ms
空间 (估计)	1	6	10^4	50

表 2: 不同方法对洛伦茨系统的微分所需时间和空间，其中空间部分以状态数目为单位。这里，NeuralODE中单步计算的微分由NiLang完成，检查点的步长为200步。

表 2对比了不同方法的时间和空间的消耗，可以看到前向自动微分所用的时间仅为原代码的不到2倍。这里的高效来自 ForwardDiff 中允许一次对多个变量求导，代价是用了正比于参数数目倍数的空间。该技术虽然没有改变随着输入变量数目增加，计算复杂度线性增加的本质，但是减少了线性部分的系数，对处理实际问题很有帮助。基于可逆计算的后向自动微分库NiLang [24]需的计算时间为原计算的约3.5倍，其中包含了前向计算和反向传播过程，因此这个速度并不算差。但是由于龙格库塔积分器不可逆，在不利用额外的计算时间交换空间的情况下，需要缓存每一步的计算以保证可逆性，因此要求有10^4倍于状态大小的缓存空间。好在该问题的单个状态空间仅有三个维度，不作任何缓存求导仍然很容易。表中最后一列的伴随状态法的单步求导利用了 NiLang 的自动微分，虽然伴随状态法在理论上可以做到无额外内存消耗，但是会引入由于积分器不可逆带来的系统误差，这对于研究混沌问题非常致命。我们以前向自动微分的导数是严格的导数作为基准，把伴随状态法求得的导数的l^2相对误差与积分步长的关系绘制于图 4 (a)中，可以看出相对误差随着积分步长指数增加。因此，我们需要每隔一段积分，就设置一个检查点重新加载正确的坐标。图 (b) 显示检查点越密，误差越小，消耗的额外空间也越多。在表格中的模拟中，我们选择了检查点步长为200，对应检查点数目为50，相对误差约为10^{-5}。

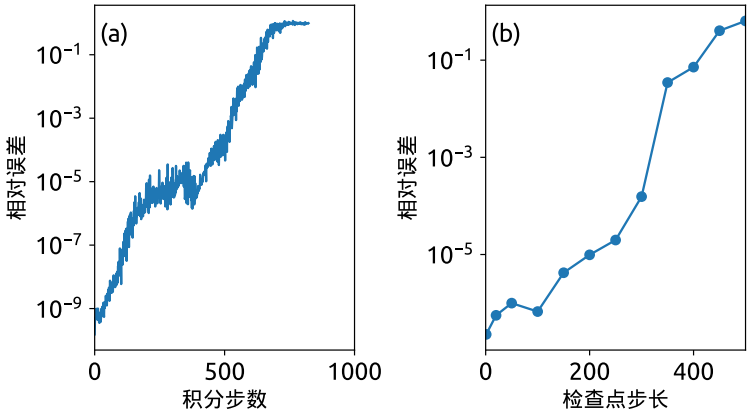


图 4: (a) 利用共轭态方法对$\rho = 27, \sigma = 10, \beta = 8/3$的洛伦茨系统求导时，相对$l^2$误差与积分步数的关系。其中一个点代表了在该步长下，对100个随机初始点计算得到的中位数，缺失的数据代表该处出现数值溢出。 (b) 每隔固定步长设置检查点后，10^4步积分的误差。

4.2 波的传播方程的微分

考虑由如下方程决定的波函数$u(x_1, x_2, t)$在非均匀二维介质中的传播过程

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} - \nabla \cdot (c^2 \nabla u) = f & t > 0, \\ u = u_0 & t = 0, \\ \frac{\partial u}{\partial t} = v_0 & t = 0. \end{cases} \tag{3}$$

其中c为波在介质中的传播速度。理想匹配层 (PML) [30, 31, 32]是模拟波在介质中运动的一种准确可靠的方案，为了在有限尺寸模拟该动力学，PML方法引入了吸收层防止边界反射的影响。在引入辅助场并对空间和时间进行离散化后，上述方程可变形为如下数值计算过程

$$\begin{cases} u_{i,j}^{n+1} \approx & \frac{\Delta t^2}{1+(\zeta_{1i}+\zeta_{2j})\Delta t/2} \Bigg((2 - \zeta_{1i}\zeta_{2j})\, u_{i,j}^n - \frac{1-(\zeta_{1i}+\zeta_{2j})\Delta t/2}{\Delta t^2} u_{i,j}^{n-1} \\ & + c_{i,j}^2 \frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\Delta x^2} + c_{i,j}^2 \frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\Delta y^2} \\ & + \frac{(\phi_x)_{i+1,j} - (\phi_x)_{i-1,j}}{2\Delta x} + \frac{(\phi_y)_{i,j+1} - (\phi_y)_{i,j-1}}{2\Delta y} \Bigg) \\ (\phi_x)_{i,j}^{n+1} = & (1 - \Delta t \zeta_{1i})(\phi_x)_{i,j}^n + \Delta t c_{i,j}^2 (\zeta_{1i} - \zeta_{2j}) \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} \\ (\phi_y)_{i,j}^{n+1} = & (1 - \Delta t \zeta_{2j})(\phi_y)_{i,j}^n + \Delta t c_{i,j}^2 (\zeta_{2j} - \zeta_{1i}) \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} \end{cases} \tag{4}$$

这里的第一项为近似，因为它忽略了原式中介质传播速度c的空间梯度项的贡献。ζ_1和ζ_2分别是x和y方向的衰减系数，ϕ_x和ϕ_y分别为引入的辅助场的x和y方向的分量，Δx, Δy和Δt分别是空间和时间的离散化参数。该方程的详细推导可参考文献 [26]。 PML 模拟的自动微分在地震学中有着重要的应用 [15]，而且人

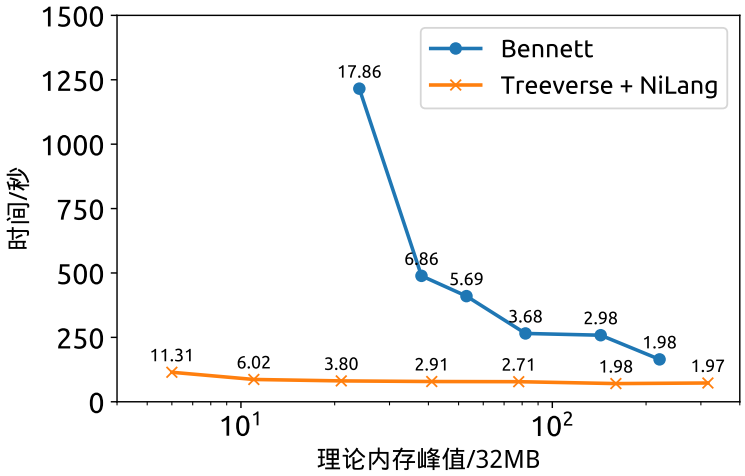


图 5: Bennett算法和Treeverse算法实际应用与PML求解过程的微分的时间与空间开销，其中图中标记的数字为函数的实际前向运行步骤数与原始模拟步骤数 (10^4) 的比值，纵轴的总时间是前向计算和后向传播时间的总和。在Bennett算法中，后向运行次数和前向运行次数一样，而Treeverse算法中，反向传播的次数固定为10^4。横轴的空间的数值的实际含义为检查点的数目或可逆计算中的最大状态数。
们很早就认识到检查点方案可以用于地震波模拟中 [14]来让回溯中间状态的内存需求大大减少。

在数值模拟中，我们用双精度浮点数模拟了1000 × 1000的二维格点上的PML方程求解，每个状态要存储4个矩阵$s_n = \{u^{n-1}, u^n, \phi_x^n, \phi_y^n\}$，占用存储空间为 32MB。虽然前向自动微分可以仅用常数倍空间开销微分该程序，但由于仅传播速度c就包含10^6个参数，前向自动微分带来的线性时间复杂度的增加是不可接受的。同时，若不作任何的内存优化对该程序后向自动微分，积分10^4步需要存储空间至少为320G，远超出了普通GPU的存储能力。此时，我们需要用附录中描述的 Bennett 算法和 Treeverse 算法来节省后向自动微分的缓存。图 5中展示了这两种时间空间交换方案下，实际程序在 GPU 上运行的时间与空间的关系，计算设备为 Nvidia Tesla V100。纯可逆计算实现的 Bennett 算法中，计算梯度的部分随着前向计算的步骤数的增加而增加，额外开销和理论模型几乎一致。 Bennett 算法在可逆计算的意义下是最优的时空交换策略，但对普通硬件并不是最优的。 Treeverse+NiLang 的方案是指用可逆计算处理单步运算的微分，同时用 Treeverse 算法处理步骤间的微分。这里随着检查点数目的减少，计算时间的减少并不明显。这是因为增加的计算时间是前向计算，而这里单步后向计算梯度的时间是前向时间的二十多倍，因此即使仅用5个检查点，额外的增加的时间也不到一倍。这里单步计算梯度的之所比前向计算慢很多是因为当使用 NiLang 微分并行运行的GPU核函数，必须要避免变量的共享读取以避免程序在后向计算中同时并行更新同一块内存的梯度，这些额外的设计带来了很多性能的损失。与此对比，在单线程版本的CPU上，前向和后向的单步运算时间差距在四倍以内。此外，虽然 Treeverse 算法在可以做到高效的时间和空间的交换，但由于内存的管理需要用到系统内存中的全局堆栈，无法直接用于微分 GPU 的核函数。而可逆计算则非

常适合微分这种非线性且具有一定可逆性的程序，这也是为什么这里选择用可逆计算对单步运算求导来避免手动求导 GPU 核函数的麻烦。

致谢

感谢王磊老师的讨论以及计算设备的支持。感谢彩云天气 CTO 苑明理老师关于自动微分关于自动微分在天气预报中应用的讨论。

附录：时间与空间的交换，鹅卵石游戏

鹅卵石游戏是一个定义在一维格子上的单人游戏。人们最初提出这个模型是为了描述可逆计算中的时间与空间的交换关系。游戏开始时，玩家拥有一堆鹅卵石以及一个一维排布的n个格子，标记为$0, 1, 2 \dots n$，并且在0号格子上有一个预先布置的鹅卵石。游戏规则为

鹅卵石游戏-可逆计算版本

放置规则：如果第i个格子上有鹅卵石，则可以从自己堆中取一个鹅卵石放置于第$i+1$个格子中，
回收规则：仅当第i个格子上有鹅卵石，才可以把第$i+1$个格子上的鹅卵石取下放入自己的堆中，
结束条件：第n个格子上有鹅卵石。
游戏目标：在固定可使用鹅卵石数目为S (不包括初始鹅卵石) 的前提下，使用尽可能少的步骤数触发游戏结束。

这里一个鹅卵石代表了一个单位的内存，而放置和取回鹅卵石的过程分别代表了计算和反计算，因此均需要一个步骤数，对应计算中的一个单位的运算时间。这里对应回收规则中要求前一个格点中存在鹅卵石，对应可逆计算在释放内存时，要求其前置状态存在以保证反计算可行。当鹅卵石数目充足 ($S \geq n$)，我们用n个鹅卵石依次铺至终点格子，此时时间复杂度和空间复杂度均为$\mathcal{O}(n)$。最少的鹅卵石数目的玩法则需要用到可逆计算框架下时间和空间最优交换方案Bennett算法。

如算法 2 (图6 (b)) 所示，Bennett 算法将格子均匀的分割为$k \geq 2$等份，先是像前执子k个区块使最后一个区块的末尾存在鹅卵石，然后从最后第$k-1$个区块开始反向执子收回中间$k-1$个鹅卵石到自由堆中。每个区块又递归的均匀分割为k个子分块做同样的放置鹅卵石-保留最后的鹅卵石-取回鹅卵石的操作，直到格子无法再分割。假设次过程的递归次数为l，我们可以得到步骤数和鹅卵石数为

$$T = (2k-1)^l, S = l(k-1)+1. \tag{5}$$

算法 2: Bennett算法

输入: 初始状态集合$S = \{0 : s_0\}$, 子分块数目k, 分块起点$i = 0$, 分块长度$L = n$

输出: 末态$S[n]$

```
function bennett(S, k, i, L)
    if L = 1 then
        S[i+1] ← 0
        S[i+1] += f_i(S[i])
    else
        l ← ⌈L/k⌉
        k' ← ⌈L/l⌉
        for j = 1, 2, ..., k' do
            bennett(S, k, i + (j−1)l, min(l, L − (j−1)l))      # 向前执行 k' 个分块
        end
        for j = k'−1, k'−2, ..., 1 do
            ~bennett(S, k, i + (j−1)l, l)                      # 向后执行 k'-1 个分块
        end
    end
end
```

其中，k与l满足总格子数$n = k^l$。可以看出可逆计算的时间复杂度和原时间为多项式关系。同时可以看出k越小，使用的总鹅卵石数目越小，因此最省空间的鹅卵石游戏解法对应$k = 2$。作为例子，图 7 (b) 展示了$n = 16$，$k = 2$ $(l = 4)$ 时候的游戏解法，对应步骤数为$(T + 1)/2 = 41$，这里的实际操作数少了大约一半是因为最外层的 Bennett 过程不需要取回鹅卵石。

我们稍微修改游戏设定可以得到检查点版本的鹅卵石游戏。该版本中用户多了一支画笔可用于涂鸦格点，且初始状态的n号格子已经被涂鸦，新的规则描述为

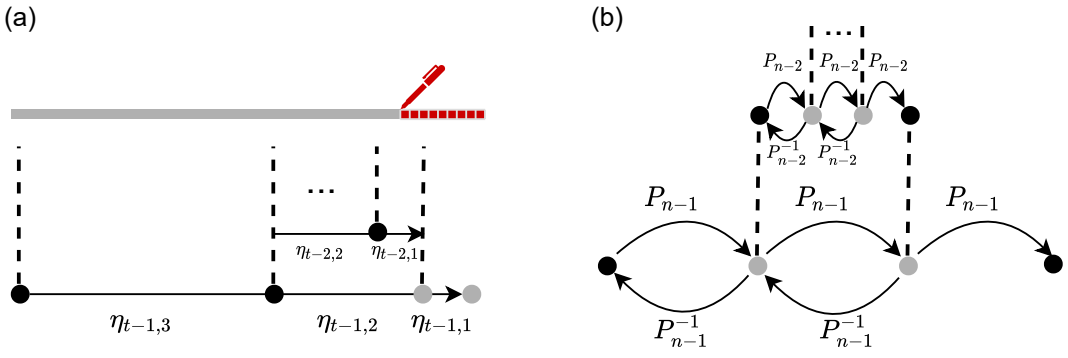


图 6: (a) Treeverse算法 [20]示意图，其中$\eta(\tau, \delta) \equiv \binom{\tau+\delta}{\delta} = \frac{(\tau+\delta)!}{\tau!\delta!}$。(b) Bennett算法对应$k = 3$的示意图 [33, 25]，其中$P$和$Q$分别代表了计算和反计算。

鹅卵石游戏-检查点版本

放置规则：如果第i个格子上有鹅卵石，则可以从自己堆中取一个鹅卵石放置于第$i+1$个格子中，

回收规则：可以随意把格子上的鹅卵石取下放入自己的堆中，收回鹅卵石不计步骤数，

涂鸦规则：当第i个格子有鹅卵石，且第$i+1$个格子被涂鸦，可以涂鸦第i个格子，涂鸦不记入步骤数，

结束条件：涂鸦完所有的格点。

游戏目标：在固定可使用鹅卵石数目为S (不包括初始鹅卵石) 的前提下，使用尽可能少的步骤数触发游戏结束。

检查点版本的鹅卵石游戏中，鹅卵石可以被随时取下，代表不可逆的内存擦除。涂鸦过程则代表了梯度反向传播的过程，它要求前置计算状态和后置梯度均都存在。在鹅卵石充足的情况下，最节省步骤数的解法和可逆计算版本一样，即计算过程中不取下任何鹅卵石。而用最少鹅卵石的解法则仅需要两枚鹅卵石交替使用，每当我们需要涂鸦一个格子i，我们总是从初始鹅卵石开始扫描（依次放置一个鹅卵石并取下前一个鹅卵石）i步至指定格子，因此总步骤数$\leq \frac{n(n-1)}{2}$。鹅卵石数目为$2 < \delta < n$的最优解最难，需要用到如算法 3 (图6 (a)) 所示的 Treeverse 算法。该算法在完成第一遍从格子0开始的扫描会在格子上留下$\delta - 1 = S - 2$个鹅卵石 (不包括初始鹅卵石)，把格子分割成δ个区块。我们把这些没有被取下的鹅卵石称为检查点，我们总可以从任意一个检查点出发扫描后方的格子。区块的大小由二项分布函数$\eta(\tau - 1, \delta), \ldots, \eta(\tau-1, 2), \eta(\tau-1, 1)$确定，其中$\tau$的取值满足$\eta(\tau, \delta) = n$。为了涂鸦$n-1$号格点，我们从离$n-1$号格点最近的检查点 (最后一个区块的开始处) 出发扫描至该点，重复该过程直至涂鸦至最后一个区块的开始处。由于最后一个区块尺寸最小，仅为$\tau - 1$，因此我们并不担心这样的扫描会使得步骤数增加太多。当我们完成了最后一个区块的涂鸦，我们便可把格子上用于标记最后一个区块起点的鹅卵石取下以便重复利用。为了涂鸦倒数第二个区块，我们先是扫描这个区块，并用刚回收的鹅卵石将其分割为大小分别是$\eta(\tau - 2, 2)$和$\eta(\tau - 2, 1)$的两个子区间。用同样的方式计算最后一个区间并递归的分割前一个子区间直至区块大小为1而无法继续分割。整个算法的步骤数和鹅卵石数目的关系是

$$T \approx \tau n, S = (\delta + 1), \tag{6}$$

由二项分布的性质可得，τ和δ的大小可以都是$\propto \log(n)$。图 7 (a) 展示了 Treeverse 算法仅用4个鹅卵石，46个步骤数涂鸦完所有20个格子。[†]

鹅卵石游戏是对程序的时间和空间的交换关系的非常理想化的描述，它恰巧非常合适用于描述常微分

[†]这里的46步并不是严格的最优解，因为图中扫描过程的最后一步并不需要立即释放内存从而可以减少步骤数。

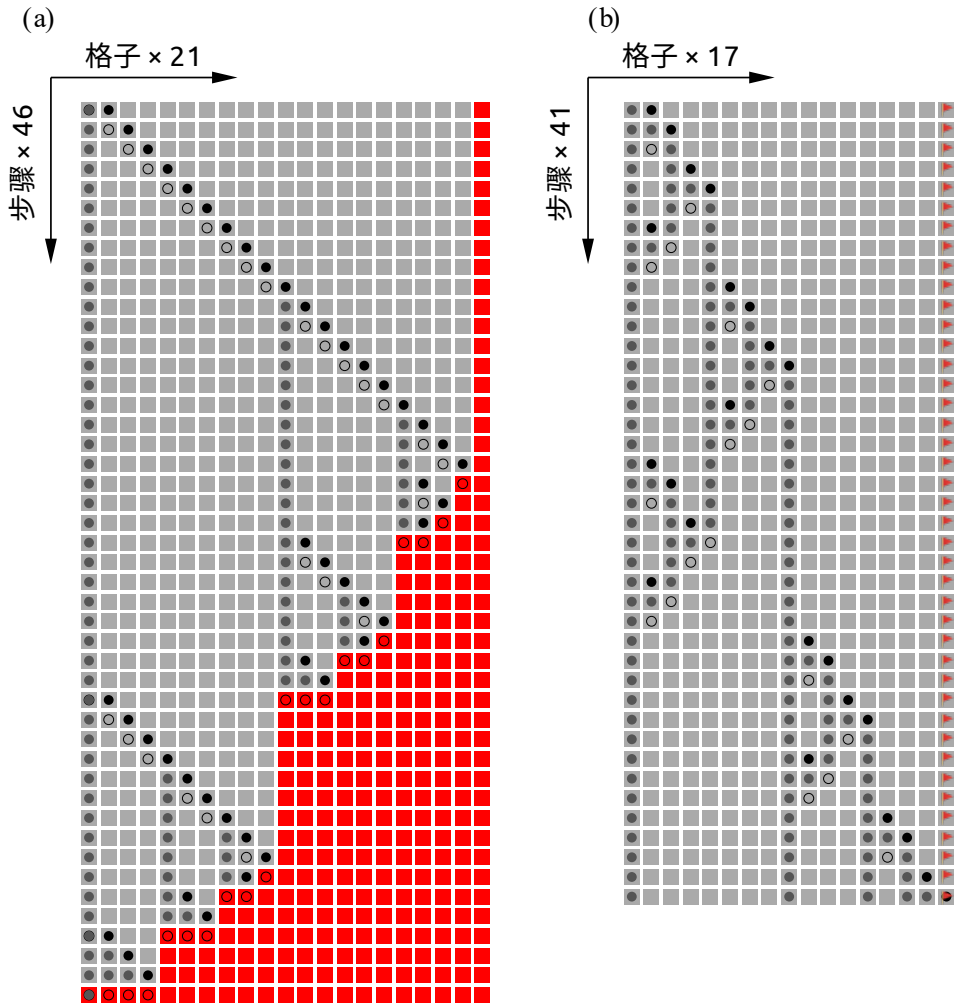


图 7: (a) Treeverse算法 ($\tau = 3, \delta = 3$) 和 (b) Bennett算法 ($k = 2, n = 4$) 对应的最优时间空间交换策略下的鹅卵石游戏解法，横向是一维棋盘的格子，纵向是步骤。其中“○”为在这一步中收回的鹅卵石，“●”为在这一步中放上的鹅卵石，而颜色稍淡的“●”则对应之前步骤中遗留在棋盘上未收回的鹅卵石。(a) 中的红色格子代表已被涂鸦。(b) 中带旗帜的格点代表终点。

算法 3: Treeverse算法

输入: 状态缓存集合$S = \{0 : s_0\}$，需回传的梯度$\overline{s_n} \equiv \frac{\partial \mathcal{L}}{\partial s_n}$，允许缓存的状态数$\delta$, 扫描次数$\tau$, 分块起点$\beta = 0$，分开终点$\phi = n$，以及把分块分割为两部分的分割点$\sigma = 0$

输出: 回传的梯度$\overline{s_0} \equiv \frac{\partial \mathcal{L}}{\partial s_0}$

function treeverse(S, $\overline{s_\phi}$, δ, τ, β, σ, ϕ)
  if $\sigma > \beta$ **then**
    $\delta = \delta - 1$
    $s = S[\beta]$ # 加载初始状态 s_β
    for $j = \beta, \beta + 1, ..., \sigma - 1$ **do**
      $s_{j+1} = f_j(s_j)$ # 计算s_σ
    end
    $S[\sigma] = s_\sigma$
  end
  # 以κ为最优分割点（二项分布），递归调用Treeverse算法
  while $\tau > 0$ **and** $\kappa = \text{mid}(\delta, \tau, \sigma, \phi) < \phi$ **do**
    $\overline{s_\kappa} = \text{treeverse}(S, \overline{s_\phi}, \delta, \tau, \sigma, \kappa, \phi)$
    $\tau = \tau - 1$
    $\phi = \kappa$
  end
  $\overline{s_\sigma} = \overline{f_\sigma}(\overline{s_{\sigma+1}}, s_\sigma)$ # 利用已有的s_σ和$\overline{s_\phi}$回传导数
  if $\sigma > \beta$ **then**
    remove($S[\sigma]$) # 从缓存的状态集合中移除s_σ
  end
  return $\overline{s_\sigma}$
end
function mid(δ, τ, σ, ϕ) # 选取二项分布分割点
  $\kappa = \lceil (\delta\sigma + \tau\phi)/(\tau + \delta) \rceil$
  if $\kappa \geq \phi$ **and** $\delta > 0$ **then**
    $\kappa = \max(\sigma + 1, \phi - 1)$
  end
end

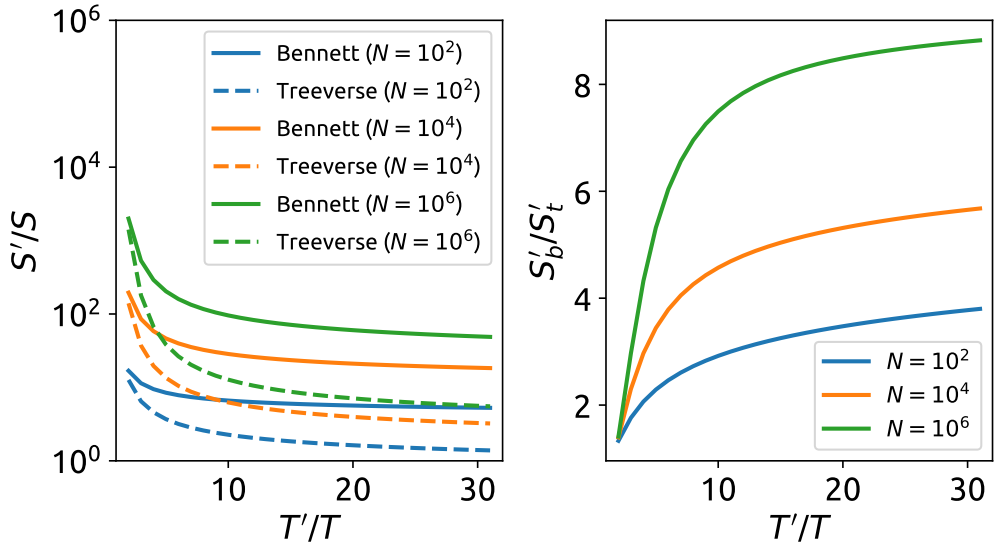


图 8: 为了回溯中间状态，时间和空间在两种最优时间-空间交换策略下的关系。(a) 固定横轴为状态回溯的计算时间与原函数计算时间的比值，对比再允许固定时间开销下，内存的额外开销。其中Bennett算法代表了可逆计算下的最优策略，而Treeverse则是传统计算允许的最优策略。(b) 对比Bennett算法与Treeverse算法空间开销的比值。

方程求解这样的不可逆线性程序。图 8 展示了在固定额外时间开销下，程序应用 Bennett 算法和 Treeverse 算法得到的最优的空间开销。可以看出，可逆计算整体上需要更多的空间开销。尤其是当步骤数更多，或是允许的时间的额外开销更大的时候，该差别愈加明显。当程序具有一定结构，可逆计算也有不错的优点，比如可以利用可逆性节省内存。另外由于可逆计算不需要对程序自动设置检查点，因此不需要借助全局堆栈，对于微分运行在GPU设备上的核函数，避免全局堆栈操作是必要的。

[1] Andreas Griewank and Andrea Walther 2008 SIAM , *Evaluating derivatives: principles and techniques of algorithmic differentiation.*

[2] C Rosset 2019 Microsoft Blog , *Turing-nlg: A 17-billion-parameter language model by microsoft.*

[3] John F Nolan 1953 , PhD thesis: *Analytical differentiation on a digital computer.*

[4] Robert Edwin Wengert 1964 *Communications of the ACM* **7** 463.

[5] Seppo Linnainmaa 1976 *BIT Numerical Mathematics* **16** 146.

[6] Martin C. Gutzwiller 1963 *Phys. Rev. Lett.* **10** 159.

[7] Giuseppe Carleo and Matthias Troyer 2017 *Science* **355** 602.

[8] Dong-Ling Deng, Xiaopeng Li, and S. Das Sarma 2017 *Phys. Rev. X* **7** 021021.

[9] Zi Cai and Jinguo Liu 2018 *Phys. Rev. B* **97** 035116.

[10] Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, and Lei Wang 2020 *Quantum* **4** 341.

[11] Hai-Jun Liao, Jin-Guo Liu, Lei Wang, and Tao Xiang 2019 *Phys. Rev. X* **9** 031041.

[12] Jin-Guo Liu, Lei Wang, and Pan Zhang 2021 *Phys. Rev. Lett.* **126** 090506.

[13] Patrick Heimbach, Chris Hill, and Ralf Giering 2005 *Future Generation Computer Systems* **21** 1356.

[14] William W Symes 2007 *Geophysics* **72** SM213.

[15] Weiqiang Zhu, Kailai Xu, Eric Darve, and Gregory C. Beroza 2021 *Computers & Geosciences* **151** 104751.

[16] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang 2018 Software , *JAX: composable transformations of Python+NumPy programs*.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala 2019 *arXiv* **1912.01703**.

[18] R.-E. Plessix 2006 *Geophysical Journal International* **167** 495.

[19] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud 2018 *Advances in Neural Information Processing Systems* **31**.

[20] Andreas Griewank 1992 *Optimization Methods and software* **1** 35.

[21] GAEL Forget, J-M Campin, Patrick Heimbach, Christopher N Hill, Rui M Ponte, and Carl Wunsch 2015 *Geoscientific Model Development* **8** 3071.

[22] Laurent Hascoet and Valérie Pascual 2013 *ACM Transactions on Mathematical Software (TOMS)* **39** 20.

[23] Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch 2008 *ACM Transactions on Mathematical Software (TOMS)* **34** 1.

[24] Jin-Guo Liu and Taine Zhao 2020 *arXiv* **2003.04617**.

[25] Robert Y Levine and Alan T Sherman 1990 *SIAM Journal on Computing* **19** 673.

[26] Marcus J. Grote and Imbo Sim 2010 *arXiv* **1001.0319**.

[27] Edward N Lorenz 1963 *Journal of atmospheric sciences* **20** 130.

[28] Morris W Hirsch, Stephen Smale, and Robert L Devaney 2012 Academic press , *Differential equations, dynamical systems, and an introduction to chaos*.

[29] Jarrett Revels, Miles Lubin, and Theodore Papamarkou 2016 *arXiv* **1607.07892**.

[30] Jean-Pierre Berenger 1994 *Journal of computational physics* **114** 185.

[31] J. A. Roden and S. D. Gedney 2000 *Microwave and Optical Technology Letters* **27** 334.

[32] Roland Martin, Dimitri Komatitsch, and Abdelaâziz Ezziani 2008 *Geophysics* **73** T51.

[33] Charles H Bennett 1973 *IBM journal of Research and Development* **17** 525.

Automatic differentiation and its applications in physics simulation *

Jin-Guo Liu[1)] Kai-Lai Xu[2)]

1) (Harvard University, Cambridge 02138)

2) (Stanford University, Stanford 94305)

1) (*Massachusetts Hall, Cambridge, MA 02138*)

2) (*450 Serra Mall, Stanford, CA 94305*)

Abstract

Automatic differentiation is a technology to differentiate a computer program automatically. It is known to many people for its use in machine learning in recent decades. Nowadays, researchers are becoming increasingly aware of its importance in scientific computing, especially in the physics simulation. Differentiating physics simulation can help us solve many important issues in chaos theory, electromagnetism, seismic and oceanographic. Meanwhile, it is also challenging because these applications often require a lot of computing time and space. This paper will review several automatic differentiation strategies for physics simulation, and compare their pros and cons. These methods include adjoint state methods, forward mode automatic differentiation, reverse mode automatic differentiation, and reversible programming automatic differentiation.

Keywords: automatic differentiation, scientific computing, reversible programming, optimal checkpointing, physics simulation

PACS: 02.60.Pn, 02.30.Jr, 91.30.-f

---

*

† Corresponding author. E-mail: jinguoliu@g.harvard.edu