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) (斯坦佛大学, 斯坦佛　94305)

自动微分是利用计算机自动化求导的技术，最近几十年因为被用于机器学习研究而被很多人了解。如今越来越多的物理学者意识到高效的，自动化的求导可以对很多物理问题的求解提供新的思路。其中自动微分在物理模拟问题中的应用尤为重要且具有挑战性。本文介绍如何将自动微分技术运用到物理模拟的求导中，介绍了共轭态法，前向自动微分，后向自动微分以及可逆计算自动微分的基本概念，并横向对比了它们在物理模拟中的优势和劣势。

关键词：自动微分，科学计算，可逆计算，Treeverse，物理模拟

PACS: 02.60.Pn, 02.30.Jr, 91.30.−f

1 引　言

自动微分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，人们可以用它优化带有千亿计参数的神经元网络 [Rosset(2019)]。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 [Nolan(1953)]，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 [Wengert(1964)] 和 1970 年 Linnainmaa 实现的后向自动微分 [Linnainmaa(1976)]。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,也带动了相关技术在科学计算中的越来越广泛的应用。科学家们利用方便的,自动化

*

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

的计算机辅助求导解决了很多重要的物理问题,其中包括变分蒙特卡洛求解多体物理波函数 [Gutzwiller(1963), Carleo and Troyer(2017), Deng *et al.*(2017)Deng, Li, and Das Sarma, Cai and Liu(2018)]，变分量子模拟的模拟 [Luo *et al.*(2019)Luo, Liu, Zhang, and Wang],多种张量网络算法 [Liao *et al.*(2019)Liao, Liu, Wang, and Xiang] 甚至是自旋玻璃基态构型 [Liu *et al.*(2020)Liu, Wang, and Zhang] 等。

本文回顾并探讨自动微分重要应用之一,对物理模拟过程的自动微分。更具体的说是对海洋学 [Heimbach *et al.*(200 和地震学 [Symes(2007), Zhu *et al.*(2020)Zhu, Xu, Darve, and Beroza] 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将问题的空间部分网格化，从而转换为对时间的常微分方程。很多现有的包括 Jax, PyTorch 在内的面向机器学习的自动微分框架中处理后向自动微分对于处理这类问题是棘手的。因为这些框架需要存储程序每一步计算的信息以便在后向传播过程中取出用于回传梯度。这对本身内存空间消耗巨大，且计算步骤数很多的物理模拟过程的积分来说并不实际。之所以说空间开销巨大, 是因为要模拟的足够精细, 空间网格必须要非常稠密, 而步骤数多是为了让对时间的常微分更加准确而要求积分步长足够小。一种传统的解决自动微分对空间的需求的方案叫做共轭态法 [Plessix(2006), Chen *et al.*(2018)Chen, Rubanova, Bettencourt, and Duvenaud], 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 Leap frog 等少数积分器在时间反演不变的哈密顿量问题中可以做到时间反演对称,大多数的积分器并不能保证严格可逆,所以共轭态法往往存在由积分步长带来的系统性误差。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状态回溯中 [Symes(2007)], 仅在对数的额外时间和空间开销下避免了系统误差。而本文作者也发现与此类似的可逆计算 [] 中的 Bennett 算法也可以用于用利用较少的空间回溯中间状态。本文将会介绍共轭态方法，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在处理物理模拟问题中的应用，以及不同方法的优劣。

章节 2 介绍了共轭态法和自动微分的基本原理。章节 3 介绍了基于检查点和可逆编程的两种后向自动微分的基础理论，尤其两者如何权衡程序的运行时间和空间。章节 4 介绍了不同自动微分技术在地震波模拟过程中的应用。

2 共轭态法与自动微分方法

物理模拟过程的常见求解方案是将偏微分方程的空间部分离散并作差分处理 [Grote and Sim(2010)], 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。假设我们已经拥有一个常微分方程方程求解器来得到末态这个常微分方程求解器在求解过程中会把时间离散化，作 n 步叠代，每步仅做从时刻 t_i 到时刻 $t_i + \Delta t$ 的演化。

$$\begin{aligned} s_n &= \text{ODESolve}(f, s_0, \theta, t_0, t_n) \\ &= (s_{i+1} = \text{ODEStep}(f, s_i, \theta, t_i, \Delta t) \text{ for } i = 0, 2, \ldots, n-1) \end{aligned} \tag{1}$$

其中 s_i 为完成第 i 步积分后的状态。最后我们还会定义一个损失函数 $\mathcal{L} = \text{loss}(s_n)$。自动微分的目标则是求解损失量对参数的导数 $\frac{\partial \mathcal{L}}{\partial s_0}$, $\frac{\partial \mathcal{L}}{\partial \theta}$, $\frac{\partial \mathcal{L}}{\partial t_0}$ 和 $\frac{\partial \mathcal{L}}{\partial t_n}$。

2.1 共轭态方法

共轭态方法 [Plessix(2006), Chen *et al.*(2018)Chen, Rubanova, Bettencourt, and Duvenaud] 是专门针对积分过程反向传播的传统方法。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更新原函数和导数的拓展函数，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。

算法 1: 共轭态法

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

$\frac{\partial \mathcal{L}}{\partial t_n} = \frac{\partial \mathcal{L}}{\partial s_n}^T f(s_n, t_n, \theta)$ # 计算损失函数对终了时间的导数

function aug_dynamics([s, a, -], t, θ) # 定义拓展动力学函数

  $s' = f(s, t, \theta)$

  return (s', $-a^T \frac{\partial s'}{\partial s}$, $-a^T \frac{\partial s'}{\partial \theta}$)

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, $\frac{\partial \mathcal{L}}{\partial s_0}$, $\frac{\partial \mathcal{L}}{\partial \theta}$) = ODESolve(aug_dynamics, S_0, t_n, t_0, θ) # 对拓展动力学反向积分

该算法的描述来自文献 [Chen *et al.*(2018)Chen, Rubanova, Bettencourt, and Duvenaud]，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。

2.2 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。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

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

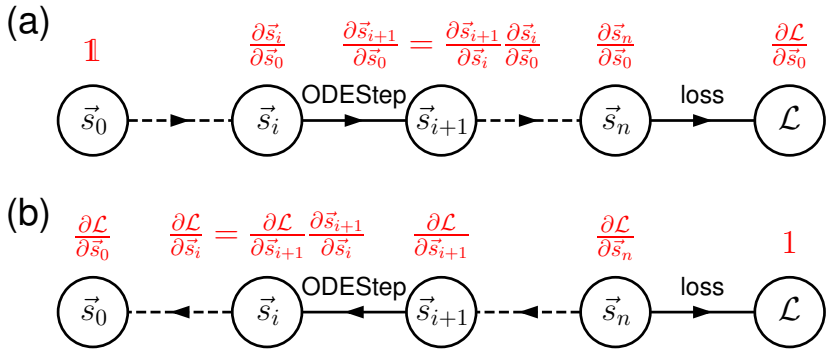

图 1: (a) 前向自动微分和 (b) 后向自动微分再常微分方程中的应用，其中圆圈为变量，线条上的箭头梯度传播的方向。

其中 $\vec{x}$ 为输入函数 f 的参数的集合，包括 p 本身。$\frac{\mathrm{d}f(\vec{x})}{\mathrm{d}\vec{x}}$ 为局域雅可比矩阵，实际程序实现中，这个局域雅可比矩阵并不需要构造出来。由于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实现自动微分的局域变换规则。具体来说一个标量会同时记录它的数值和一阶小量的系数 $(v, \dot{v})$，其中 $\dot{v} = \frac{\mathrm{d}v}{\mathrm{d}p}$，人们通过重新定义它的基本运算规则，如

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

使得其在计算同时更新一阶小量的系数。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

$$\begin{aligned}\mathrm{ODEStep}^{\mathrm{F}} : \left(\left(s_{i+1}, \frac{\mathrm{d}s_{i+1}}{\mathrm{d}s_0}\right), \left(\theta, \frac{\mathrm{d}s_{i+1}}{\mathrm{d}\theta}\right)\right) \mapsto \Bigg(&\left(\mathrm{ODEStep}\left(s_i, t_i, \theta, \Delta t\right), \frac{\partial s_{i+1}}{\partial s_i}\frac{\mathrm{d}s_i}{\mathrm{d}s_0}\right), \\ &\left(\theta, \frac{\partial s_{i+1}}{\partial s_i}\frac{\mathrm{d}s_i}{\mathrm{d}\theta} + \frac{\partial s_{i+1}}{\partial \theta}\right)\Bigg)\end{aligned}$$

这里为了简洁略去了积分函数和时间等参量。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。真实的程序计算中，仅通过基本运算的代码替换或者算符重载即可实现该变换。这里还有一个与计算时间有关的要素，那就是一次对多少个变量求导。如果同时更行整个雅可比行列式，计算空间随着求导变量数目线性增加。在实践中，在内存大小有限且需要求导变量较多的情况下，一般是重复运行多次计算过程，一次只处理若干个变量的导数这样分批次求导。无论是哪种情况，求导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。

2.3 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。后向自动微分包括前向计算和梯度后向传播两个过程。前向计算过程中，程序进行普通的计算，最后计算得到一个标量为损失 $\mathcal{L}$。梯度回传的过程是计算导数的过程，可表示为更新一个变量的对偶量的过程 $\overline{v} = \frac{\partial \mathcal{L}}{\partial v}$。从 $\overline{\mathcal{L}} = 1$ 出发，梯度回传即应用如下链式法则

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x} = \sum_{y} \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial x}$$

为了实现该链式法则，人们对于一类基础函数 $y = f(x)$ 定义了对偶量的局域运算规则

$$\overline{f} : \overline{x} \mapsto \overline{x} + \overline{y} \frac{\partial y}{\partial x}$$

其中，$\frac{\partial y}{\partial x}$ 为局域雅可比矩阵，它的数值不需要具体计算出来，而是以函数的形式，连同所需的中间变量一起存放在栈中，并在后向传播中按照后进先出的顺序调用。所有由这一类基础函数构成的代码便可以利用上述导数回传规则更新对偶量。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

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

这里也同样略却了积分函数和时间等参量。考虑到该后向传播过程需要知道函数的输入，整个前向计算（左侧）和后向梯度传播（右侧）可表达为

$$\begin{array}{ll} \cdots & \cdots \\ \mathtt{push}(\Sigma, s_i) & s_i = \mathtt{pop}(\Sigma) \\ s_{i+1} = \mathrm{ODEStep}(s_i, t_i, \theta, \Delta t) & (\overline{s_i}, \overline{\theta}) = \overline{\mathrm{ODEStep}}^{B}(\overline{s_{i+1}}, \overline{\theta}, s_i) \\ \cdots & \cdots \end{array}$$

前向计算过程中除了运算本身，还会将部分运算的中间结果通过入栈操作 `push(Σ,`s_i`)` 将状态 s_i 的值放入一个全局堆栈 Σ 中，而后向过程将这些缓存的结果通过出栈操作 `x = pop(Σ)` 取出 s_i 的值并利用后向传播规则更新 $\overline{s}_i$ 和 $\overline{\theta}$。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 (i.e. $\mathcal{O}(T)$) 的额外空间开销。如何设计算法在保证对计算状态逆

序的访问的前提下，减少计算中使用的堆栈 Σ 的大小，是我们在章节 3中要讨论的一个至关重要的问题。

3 时间与空间的权衡

3.1 重计算与反计算

在反向自动微分中，比起将每一步的计算状态保存到堆栈，有时候我们也可以通过“重计算”的方式来得到一部分中间状态。

	$s_{i+2} = \texttt{pop}(\Sigma)$
	$(\overline{s_{i+2}}, \overline{\theta}) = \overline{\text{ODEStep}}^B(\overline{s_{i+3}}, \overline{\theta}, s_{i+2})$
$\texttt{push}(\Sigma, s_i)$	$s_i = \texttt{read}(\Sigma, \texttt{i})$
$s_{i+1} = \text{ODEStep}(s_i, t_i, \theta, \Delta t)$	$s_{i+1} = \text{ODEStep}(s_i, t_i, \theta, \Delta t)$ # 重计算
$s_{i+2} = \text{ODEStep}(s_{i+1}, t_i, \theta, \Delta t)$	$(\overline{s_{i+1}}, \overline{\theta}) = \overline{\text{ODEStep}}^B(\overline{s_{i+2}}, \overline{\theta}, s_{i+1})$
$\texttt{push}(\Sigma, s_{i+2})$	$s_i = \texttt{pop}(\Sigma)$
	$(\overline{s_i}, \overline{\theta}) = \overline{\text{ODEStep}}^B(\overline{s_{i+1}}, \overline{\theta}, s_i)$

这里，`read` 函数仅读取数据而不对数据出栈。在正向计算过程中，我们选择性的没有存储状态 s_{i+1}。在反向计算过程中，由于状态 s_{i+1} 不存在，程序通过读取 s_i 的状态重新计算得到 s_{i+1}。通过这种方式，堆栈的大小可以减少 1。如何最优的选择存储和释放状态成了检查点方案的关键，这个最优方案在 1992 年被 Griewank [Griewank(1992)] 提出，被成为 Treeverse 算法，该算法仅需对数多个存储空间，即表 1所示的空间复杂度 $\mathcal{O}(S\log(T))$，其中 S 为单个状态的空间大小，T 为时间，或这里的步数。以及对数的额外时间开销，即时间复杂度 $\mathcal{O}(T\log(T))$。

还有一种可以保证变量方案叫做可逆编程，意思是让代码在结构上具有可逆性，比如在可逆编程语言的框架下书写代码。可逆的书写意味着赋值和清除变量这些常见的操作不再被允许，取而代之的是累加 (`+=`) 和累减 (`-=`)，从内存空“借”一段空内存 ($\leftarrow$) 和将一个已经清除的变量“归还”给内存 ($\rightarrow$) 等。可逆计算赋予用户更高的自由度去控制内存的分配与释放，以如下的程序计算过程（左侧）和它的反过程（右侧）为例

$s_{i+1} \leftarrow 0$

s_{i+1} += ODEStep$(s_i, t_i, \theta, \Delta t)$

$s_{i+2} \leftarrow 0$

s_{i+2} += ODEStep$(s_{i+1}, t_i, \theta, \Delta t)$

s_{i+1} −= ODEStep$(s_i, t_i, \theta, \Delta t)$ # 反计算

$s_{i+1} \rightarrow 0$ # 归还 s_{i+1}

$s_{i+1} \leftarrow 0$

s_{i+1} += ODEStep$(s_i, t_i, \theta, \Delta t)$ # 计算 s_{i+1}

s_{i+2} −= ODEStep$(s_{i+1}, t_i, \theta, \Delta t)$

$s_{i+2} \rightarrow 0$

s_{i+1} −= ODEStep$(s_i, t_i, \theta, \Delta t)$

$s_{i+1} \rightarrow 0$

这里我们先是从状态 s_i 经过两步计算得到了状态 s_{i+1} 和 s_{i+2}。由于 s_{i+1} 在接下来的运算中不会被用到，我们可以利用已有信息 s_i 将它反计算为 0，并将存储空间“归还”给系统。有了反过程，仅需要在反过程中插入求导程序即可完成自动微分。

$$(\overline{s_{i+2}}, \overline{\theta}) = \overline{\text{ODEStep}}^B(\overline{s_{i+3}}, \overline{\theta}, s_{i+2})$$

$$s_{i+1} \leftarrow 0$$

$$s_{i+1} \mathrel{+}= \text{ODEStep}(s_i, t_i, \theta, \Delta t) \text{ \# 计算 } s_{i+1}$$

$$(\overline{s_{i+1}}, \overline{\theta}) = \overline{\text{ODEStep}}^B(\overline{s_{i+2}}, \overline{\theta}, s_{i+1})$$

$$s_{i+2} \mathrel{-}= \text{ODEStep}(s_{i+1}, t_i, \theta, \Delta t)$$

$$s_{i+2} \rightarrow 0$$

$$s_{i+1} \mathrel{-}= \text{ODEStep}(s_i, t_i, \theta, \Delta t)$$

$$s_{i+1} \rightarrow 0$$

$$(\overline{s_i}, \overline{\theta}) = \overline{\text{ODEStep}}^B(\overline{s_{i+1}}, \overline{\theta}, s_i)$$

在可逆计算的时间和空间的交换规则下，最优的算法是 Bennett 算法，其空间的额外开销也是对数，但是时间的额外开销变为了多项式，即表 1所示的时间复杂度 $\mathcal{O}(T^{1+\epsilon})$，其中 $\epsilon > 0$，高于最优检查点方案的对数复杂度。Treeverse 算法和 Bennett 算法分别是源代码后向自动微分（一种通过源代码变换实现的反向传播技术）和可逆计算中权衡时间与空间的核心的算法。为了更好的理解这些算法以方便更好的对物理模拟过程微分，我们用鹅卵石游戏模型来进行说明。

3.2 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,它最初被提出描述可逆计算中的时间与空间的权衡。游戏开始，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 $0, 1, 2 \ldots n$，并且在 0 号格子上有一个预先布置的鹅卵石。其规则为

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。

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，即可逆计算的一个单位的运算时间。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的可行，在这里对应回收规则中要求前一个格点中存在鹅卵石。游戏目标是从自己的堆中取尽可能最少的鹅卵石，或是使用尽可能少的步骤数触发游戏结束。可逆计算版本的鹅卵石游戏最省步骤数的玩法，也是最消耗空间的玩法是用 n 个鹅卵石依次铺至终点格子 n，此时时间复杂度和空间复杂度均为 $\mathcal{O}(T)$，此处 T 为格子数。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。

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**
    $S[i+1] \leftarrow 0$
    $S[i+1]$ += $f_i(S[i])$
  else
    $l = \lceil \frac{L}{k} \rceil$
    $k' = \lceil \frac{L}{l} \rceil$
    for $j = 1, 2, ..., k'$ **do**
      bennett(S, k, $i + (j-1)l$, $\min(\frac{L}{k}, L - (j-1)l)$) # forward for k' sub-blocks
    end
    for $j = k' - 1, k' - 2, ..., 1$ **do**
      $\sim$bennett(S, k, $i + \frac{j-1}{k}L$, $\frac{L}{k}$) # backward for k'-1 sub-blocks
    end
  end
end

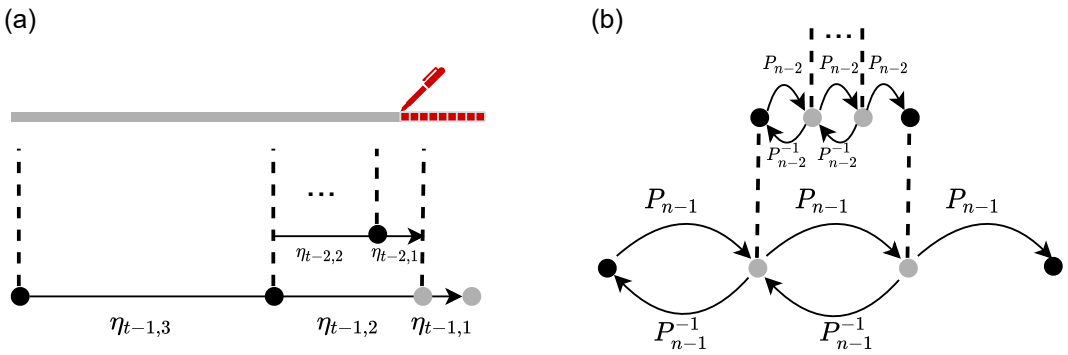

图 2: (a) 广义自动微分中常见的 Treeverse 算法 [Griewank(1992)]，其中 $\eta(\tau,\delta) \equiv \binom{\tau+\delta}{\delta} = \frac{(\tau+\delta)!}{\tau!\delta!}$。(b) 可逆计算时空交换的 Bennett 算法。 [Bennett(1973), Levine and Sherman(1990)] 其中，P 和 Q 分别代表了计算和反计算。

如算法 2（图 5 (b)）所示，Bennett 算法将格子均匀的分割为 $k \geq 2$ 等份，先是像前执行 k 个区块得到计算结果，然后从最后第 $k-1$ 个区块开始依次收回中间 $k-1$ 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，直到程序无法再分割。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与 k 和 l 的关系如下

$$T_r = (2k-1)^l, S_r = l(k-1). \tag{3}$$

其中，k 与 l 满足 $T = k^l$。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 3 (b) 展示了 $n = 16$，$k = 2$ $(l = 4)$ 时候的游戏解法，对应步骤数为 $(T_r + 1)/2 = 41$，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的过程。

我们稍微修改可以得到检查点版本的规则，它为用户增加了一支画笔用于涂鸦格点，改变后的规则描述为

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 $i+1$ 个格子被涂鸦或 $i = n$，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。

检查点版本的鹅卵石游戏中，涂鸦过程代表了梯度反向传播的过程，因此它要求按照以与程序正常运行方向相反的顺序访问计算状态。它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

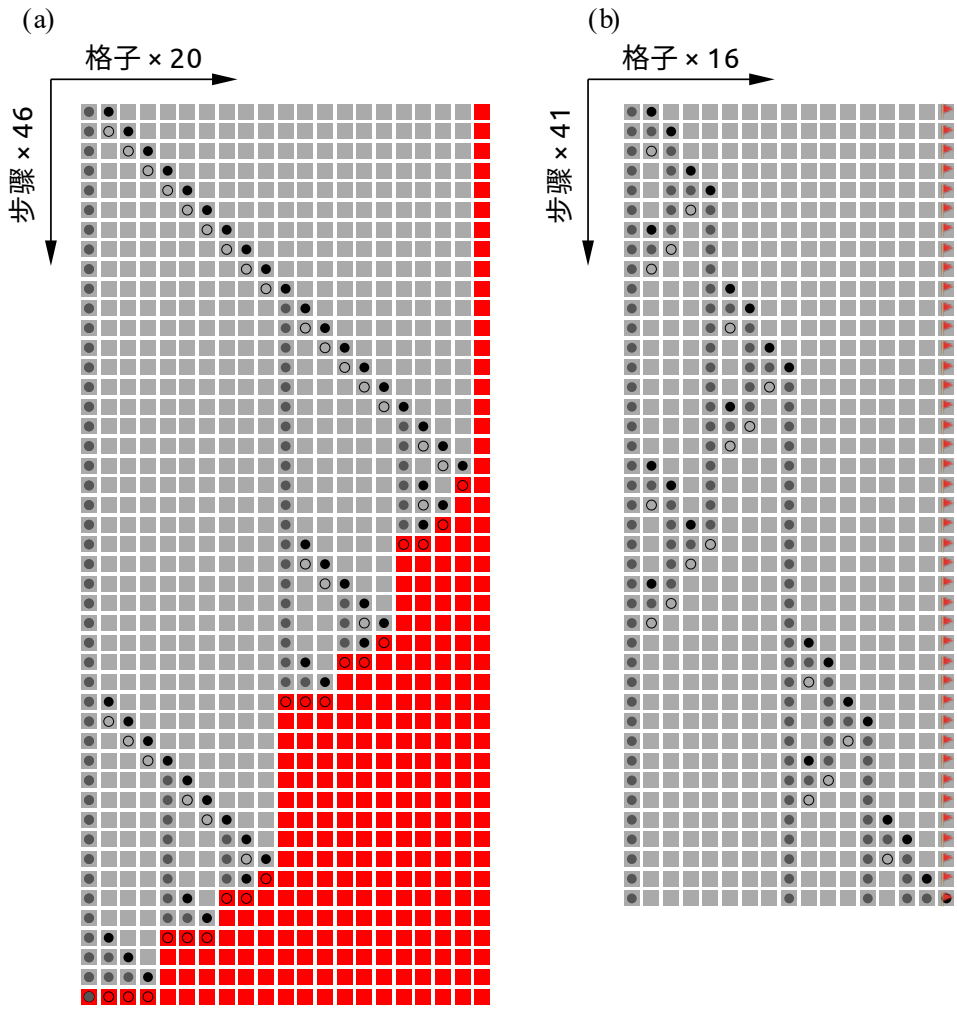

图 3: (a) Treeverse 算法 ($t = 3$, $d = 3$) 和 (b) Bennett 算法 ($k = 2$, $n = 4$) 对应的时间空间交换策略下的鹅卵石游戏，横向是一维棋盘的格子，纵向是步骤。其中“○”为在这一步中收回的鹅卵石，“●”为在这一步中放上的鹅卵石，而颜色稍淡的“●”则对应遗留在棋盘上未收回的鹅卵石。红色格子代表已被涂鸦，带旗帜的格点代表终点。

算法 3: Treeverse 算法

输入: 初始状态 $s = s_0$, 状态缓存集合 $S = \{\}$，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 δ, 扫描次数 τ, 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, S, $\overline{s_\phi}$, δ, τ, β, σ, ϕ)
  if $\sigma > \beta$ **then**
    $\delta = \delta - 1$
    $S[\beta] = s_\beta$ # 缓存状态 s_β 至状态集合 S
    for $j = \beta, \beta + 1, ..., \sigma - 1$ **do**
      $s_{j+1} = f_j(s_j)$ # 计算 s_σ
    end
  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**
    $\overline{s_\kappa} = \text{treeverse}(s_\sigma, S, \overline{s_\phi}, \delta, \tau, \sigma, \kappa, \phi)$
    $\tau = \tau - 1$
    $\phi = \kappa$
  end
  if $\phi - \sigma \neq 1$ **then**
    `error("treeverse fails!")` # 由于总步长 $n \neq \eta(\tau + \delta, \delta)$，实际实现应处理该异常
  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($S[\beta]$) # 从缓存的状态集合中移除 s_β
  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

鹅卵石。而用最少鹅卵石的解法则是每当我们需要涂鸦一个格子 i，我们总是从初始鹅卵石 s_0 开始扫描（依次放置一个鹅卵石并取下前一个鹅卵石）i 步至格子 i，因此只需要 2 个鹅卵石即可涂鸦全部格子，步骤数为 $\frac{n(n-1)}{2}$。如算法 3 （图 5 (a)）所示，完成第一遍从 s_0 到 s_n 的扫描后会在棋盘上留下 δ 个鹅卵石（不包括初始鹅卵石），把格点分割成 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通过放置-取回鹅卵石的操作扫描后方的格子。每个区块的大小被证明有一个最优的取值，即大小为二项分布函数 $\eta(\tau, d)$，其中 $d = 1, 2, \ldots, \delta$ 为从末尾开始数的区块的指标，而 τ 的取值满足 $\eta(\tau, \delta) = n$。由于最后的 n 号格子可以直接被涂色，拥有状态 s_{n-1} 后，$n-1$ 号格子满足涂鸦规则，因此我们可以在第一遍扫描时给它涂上颜色。为了继续涂鸦 $n-2$ 号格点，我们从离 $n-2$ 号格点最近的检查点出发扫描至该点，依次类推直至达到最后一个检查点处。由于最后一个区块尺寸最小，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重复利用。为了涂鸦倒数第二个区块，我们先是扫描整个区间，并把这个区间用回收的鹅卵石依据二项分布 $\eta(\tau - 1, d = 2, 1)$ 分割为两个子区间。随后依然是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，同时也意味着进行了 τ 次递归。整个算法的时间和空间开销的关系是

$$T_c \approx \tau T, S_c = (\delta + 1)S, \tag{4}$$

其中，$T = \eta(\tau, \delta)$ 是初始计算时间。选择适当的 t 或者 d，在时间和空间维度上的额外复杂度可以都是 $\log(T)$。图 3 (a) 展示了如何只用 4 个鹅卵石，步骤数 46 涂鸦完所有 20 个格子。

图 4 展示了在固定额外时间开销的情况下，Bennett 算法和 Treeverse 算法得到的最优的空间开销。由于可逆性的限制，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。但可逆计算也有优点，其一是可以利用可逆性节省内存，比如稀疏矩阵的运算中，大多数运算都是可逆的基础操作。同时由于没有全局堆栈以及对程序自动设置检查点的问题，程序设计上也更加自由，比如在 GPU 编程的设备函数中是不允许访问全局堆栈的。基于鹅卵石模型的讨论对于通用的程序显然是过于理想化的，但这样理想化的描述恰巧非常合适用于常微分方程微分的描述。

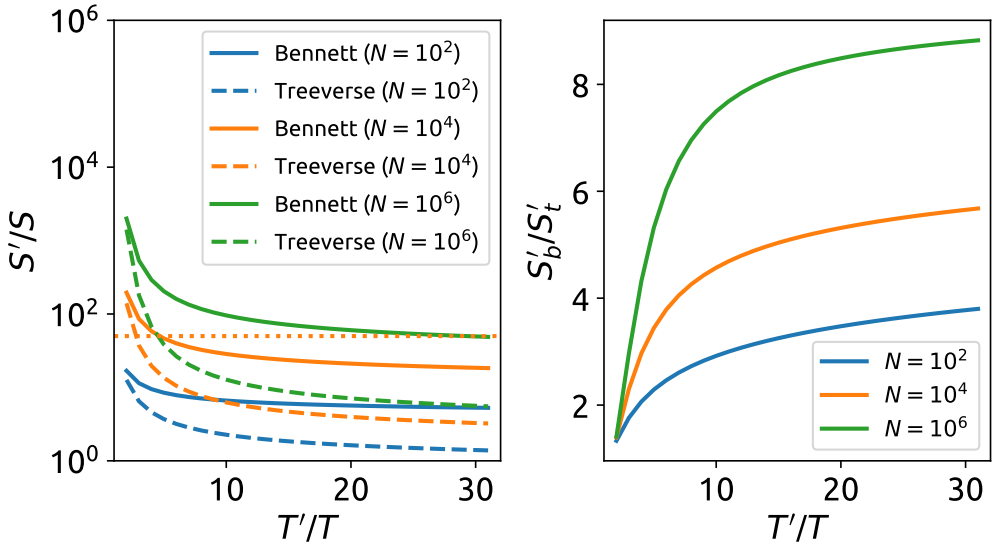

图 4: 为了回溯中间状态，时间和空间在两种最优时间-空间交换策略下的关系。(a) 固定横轴为状态回溯的计算时间与原函数计算时间的比值，对比再允许固定时间开销下，内存的额外开销。其中 Bennett 算法代表了可逆计算下的最优策略，而 Treeverse 则是传统计算允许的最优策略，黄色点状横线对应 $S'/S = 50$。(b) 对比 Bennett 算法与 Treeverse 算法空间开销的比值。

方法	时间	空间
共轭态法	$\mathcal{O}(T)$	$\mathcal{O}(TS)$
前向自动微分	$\mathcal{O}(NT)$	$\mathcal{O}(S)$
基于检查点的后向自动微分	$\mathcal{O}(T\log T)$	$\mathcal{O}(S\log T)$
基于可逆计算的后向自动微分	$\mathcal{O}(T^{1+\epsilon})$	$\mathcal{O}(S\log T)$

表 1: 不同方案的时间与空间复杂度。其中共轭态法考虑了缓存部分中间结果以保证反向积分的正确性，因此空间会有与时间线性增长。前向自动微分中的 N 代表了需要求导的参数个数。可逆计算中的时间复杂度为多项式，且 $\epsilon > 0$。

4 自动微分在物理模拟中的应用

4.1 地震波的模拟

Perfectly matched layer (PML) 方程是模拟波在介质中运动的一种准确可靠的方案，在介质中，波场 $u(\vec{x}, t)$ 的传播可描述为

$$\begin{cases} u_{tt} - \nabla\cdot(c^2\nabla u) = f & t > 0, \\ u = u_0 & t = 0, \\ u_t = v_0 & t = 0. \end{cases} \tag{5}$$

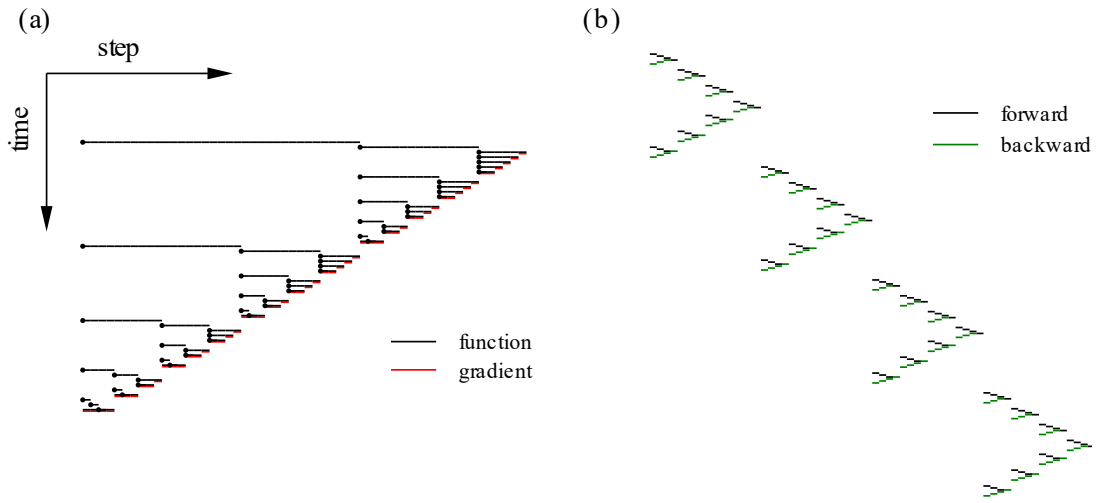

图 5: (a) Treeverse 算法 ($t = 5, d = 3$) 和 (b) Bennett 算法 ($k = 4, n = 3$) 中，函数的执行过程，其中横向是鹅卵石游戏的格子，纵向随着进入 Treeverse 函数或者 Bennett 函数的次数向下延展。

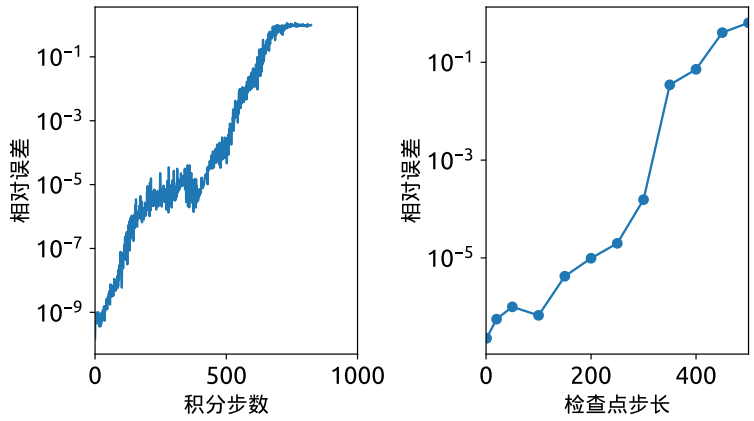

图 6: 利用共轭态方法求导时，l^2 误差与积分步长的关系。其中一个点代表了在该步长下，对 100 个随机初始点计算得到的中位数。缺失的数据代表该处出现数值溢出的情况。

经过对空间的离散化处理后来在地震学的模拟中，它被用来微分地震波传播的过程 [Symes(2007)]。自动微分也在其中发挥着重要的作用 [Zhu *et al.*(2020)Zhu, Xu, Darve, and Beroza]。 [Grote and Sim(2010)]

5 结　论

致谢

感谢王磊老师的讨论。感谢彩云天气 CTO 苑明理老师的讨论。

附录 A1

标题排列和编号方式为 A1, A2, A3.

附录 A2

[Rosset(2019)] C. Rosset, Microsoft Blog (2019).

[Nolan(1953)] J. F. Nolan, *Analytical differentiation on a digital computer*, Ph.D. thesis, Massachusetts Institute of Technology (1953).

[Wengert(1964)] R. E. Wengert, Communications of the ACM **7**, 463 (1964).

[Linnainmaa(1976)] S. Linnainmaa, BIT Numerical Mathematics **16**, 146 (1976).

[Gutzwiller(1963)] M. C. Gutzwiller, \bibfield journal \bibinfo journal Phys. Rev. Lett.\ \textbf \bibinfo volume 10,\ \bibinfo pages 159 (\bibinfo year 1963).

[Carleo and Troyer(2017)] G. Carleo and M. Troyer, \bibfield journal \bibinfo journal Science\ \textbf \bibinfo volume 355,\ \bibinfo pages 602—606 (\bibinfo year 2017).

[Deng *et al.*(2017)Deng, Li, and Das Sarma] D.-L. Deng, X. Li, and S. Das Sarma, \bibfield journal \bibinfo journal Physical Review X\ \textbf \bibinfo volume 7 (\bibinfo year 2017),\ 10.1103/physrevx.7.021021.

[Cai and Liu(2018)] Z. Cai and J. Liu, \bibfield journal \bibinfo journal Phys. Rev. B\ \textbf \bibinfo volume 97,\ \bibinfo pages 035116 (\bibinfo year 2018).

[Luo *et al.*(2019)Luo, Liu, Zhang, and Wang] X.-Z. Luo, J.-G. Liu, P. Zhang, and L. Wang, "Yao.jl: Extensible, efficient framework for quantum algorithm design," (2019), arXiv:1912.10877 [quant-ph] .

[Liao *et al.*(2019)Liao, Liu, Wang, and Xiang] H.-J. Liao, J.-G. Liu, L. Wang, and T. Xiang, \bibfield journal \bibinfo journal Physical Review X\ \textbf \bibinfo volume 9 (\bibinfo year 2019),\ 10.1103/physrevx.9.031041.

[Liu *et al.*(2020)Liu, Wang, and Zhang] J.-G. Liu, L. Wang, and P. Zhang, "Tropical tensor network for ground states of spin glasses," (2020), arXiv:2008.06888 [cond-mat.stat-mech] .

[Heimbach *et al.*(2005)Heimbach, Hill, and Giering] P. Heimbach, C. Hill, and R. Giering, Future Generation Computer Systems **21**, 1356 (2005).

[Symes(2007)] W. W. Symes, \bibfield journal \bibinfo journal Geophysics\ \textbf \bibinfo volume 72,\ \bibinfo pages SM213 (\bibinfo year 2007).

[Zhu *et al.*(2020)Zhu, Xu, Darve, and Beroza] W. Zhu, K. Xu, E. Darve, and G. C. Beroza, "A general approach to seismic inversion with automatic differentiation," (2020), arXiv:2003.06027 [physics.comp-ph] .

[Plessix(2006)] R.-E. Plessix, \bibfield journal \bibinfo journal Geophysical Journal International\ \textbf \bibinfo volume 167,\ \bibinfo pages 495 (\bibinfo year 2006), https://academic.oup.com/gji/article-pdf/167/2/495/1492368/167-2-495.pdf .

[Chen *et al.*(2018)Chen, Rubanova, Bettencourt, and Duvenaud] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, in \emph \bibinfo booktitle Advances in Neural Information Processing Systems, Vol. 31, edited by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Curran Associates, Inc., 2018).

[Grote and Sim(2010)] M. J. Grote and I. Sim, "Efficient pml for the wave equation," (2010), arXiv:1001.0319 [math.NA] .

[Griewank(1992)] A. Griewank, \bibfield journal \bibinfo journal Optimization Methods and software\ \textbf \bibinfo volume 1,\ \bibinfo pages 35 (\bibinfo year 1992).

[Bennett(1973)] C. H. Bennett (1973).

[Levine and Sherman(1990)] R. Y. Levine and A. T. Sherman, \bibfield journal \bibinfo journal SIAM Journal on Computing\ \textbf \bibinfo volume 19,\ \bibinfo pages 673 (\bibinfo year 1990).

Automatic differention 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

To determine the probe made of amino acids arranged in a linear chain and joined together by peptide bonds between the carboxyl and amino groups of adjacent amino acid residues. The sequence of amino acids in a protein is defined by a gene and encoded in the genetic code. This can happen either before the protein is used in the cell, or as part of control mechanisms.

Keywords: automatic differentiation, scientific computing, reversible programming, Treeverse, physics simulation

PACS: 02.60.Pn, 02.30.Jr, 91.30.−f

* Project supported by the State Key Development Program for Basic Research of China (Grant No. 2011CB00000), the National Natural Science Foundation of China (Grant Nos. 123456, 567890), and the National High Technology Research and Development Program of China (Grant No. 2011AA06Z000).

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