arXiv:2304.02650v1 [cs.MS] 4 Apr 2023

使用 RooFit 和 Clad 对分箱似然进行自动微分

Garima Singh*、**Jonas Rembser**†、**Lorenzo Moneta**†、**David Lange***、**Vassil Vassilev***

* 物理系，普林斯顿大学，新泽西州普林斯顿 08544，美国
† EP-SFT，CERN，粒子大道 1 号，1211 梅兰，瑞士

电子邮件：garima.singh@cern.ch, jonas.rembser@cern.ch, lorenzo.moneta@cern.ch, david.lange@cern.ch, vassil.vassilev@cern.ch

摘要。 随着下一代实验的数据集不断增长，物理分析的处理需求在计算上变得更加苛刻，因此需要对 RooFit 进行性能优化。加速最小化并增加稳定性的一种可能方法是使用自动微分（AD）。与数值微分不同，计算成本随参数数量线性扩展，这使得 AD 对于具有许多参数的统计模型特别有吸引力。在本文中，我们报告了在 RooFit 中实现 AD 的一种可能方法。我们的方法是添加一个设施，以自动为完整的 RooFit 模型生成 C++ 代码。与原始 RooFit 模型不同，此生成的代码没有虚函数调用和其他 RooFit 特定的开销。特别是，然后使用此代码通过 Clad 自动生成梯度。Clad 是一种源转换 AD 工具，作为 clang 编译器的插件实现，它会自动为输入的 C++函数生成导数代码。我们展示了将此代码生成策略应用于 HistFactory 和其他常用 RooFit 模型时观察到的改进结果。HistFactory 是 RooFit 的子组件，它实现了基于直方图模板的概率密度的分箱似然模型。这些模型通常具有非常大量的自由参数，因此是 RooFit 中 AD 支持的一个有趣的首要目标。

1. 引言

在大型强子对撞机收集的数据分析中，分析的复杂性持续增长。诸如联合希格斯测量等尖端统计分析通常使用RooFit 库 [1], ，它是数据分析框架 ROOT [2]的一部分。在这些研究中，必须对包含数千个自由参数的似然函数进行最小化；这些参数分布在数百个似然分量中，每个分量代表不同的测量通道。对于最大的模型，似然最小化可能需要数小时才能完成 [3]。为了确保这一时间不再进一步增长，并希望甚至将其缩短到几分钟的量级，RooFit 的性能在过去几年中得到了显著提升。例如，这包括对似然评估代码进行彻底重写，以实现更容易的向量化 [4]。

为了迭代地找到似然函数的最小值，通常需要知道似然函数相对于所有自由参数的梯度。朴素地，可以通过一次改变一个参数并重新评估完整似然函数来数值地找到梯度。然而，当只改变一个参数时，没有必要重新评估完整的数学表达式，

这就是 RooFit 包含复杂缓存机制的原因。计算图中的各个表达式或表达式组都由对象表示，这些对象会缓存其上一次的结果。

这种缓存方法存在一些性能缺陷。它需要大量簿记工作（例如跟踪因参数值变化而需要重新计算的模型部分），这会带来显著开销。以内存占用相对较大的对象来表示计算图，并通过虚函数调用相互求值，对性能也不理想。此外，由于单个 RooFit 对象通常将许多数学运算组合在一起，因此仍存在一些冗余计算。换句话说，对于计算图的某些子图，我们会回退到完全冗余的重新求值。

偏离这种缓存范式，我们相信通过在最细粒度级别更好地分析似然函数的完整表达式图，可以自动生成无开销且无冗余计算的完整梯度表示，从而缓解所有这些性能惩罚。这是编译器可以完成的工作。在本文中，我们为分箱的 HistFactory 风格似然函数 [5] 使用 Clad [6], 一种基于编译器的源代码转换 AD 工具生成了梯度。

2. 背景

2.1. 数值微分

数值微分旨在估算数学函数的导数，具体而言，是利用函数值或其他元信息进行估算。它广泛应用于各种流行的数学软件和库中，包括 RooFit；在 RooFit 中，它通过最小化库 MINUIT [7] 实现导数计算。

数值微分可以通过多种不同的技术实现，其中最著名的方法之一是牛顿有限差分法。在该方法中，实值多元函数 $f(x_0, x_1, ..., x_n)$ 关于输入 x_i 的导数可写为：

$$\frac{\delta f(x_0, x_1, ..., x_n)}{\delta x_i} \equiv \frac{f(x_0, ..., x_i + h, ..., x_n) - f(x_0, ..., x_i - h, ..., x_n)}{2h} \tag{1}$$

有限差分法通过向函数引入一个微小扰动 h，并测量其对函数结果的影响来计算导数。这个微小扰动通常称为步长。步长越小，导数估算就越准确。理论上，步长应为一个非常小的值；然而，由于浮点运算的特性，除以较小的浮点数可能会导致结果产生较大的数值误差。这使得计算 h 的值成为一个相当困难的优化问题：值过小会导致较大的数值误差，而值过大则会使导数估算变差。此外，步长的选择还取决于所求解的问题，这意味着不存在适用于所有使用场景的单一 h 值。因此，在保持导数准确性的同时选择合适的步长极具挑战性。

数值微分的另一个不足之处是性能较差。例如，在方程 1 中，函数 f 对所有输入的完整导数是逐个计算的。这意味着，由于每次导数计算至少需要调用函数 2 次（使用其他更精确的方法时，调用次数还会更多），对于一个有 n 个参数的函数，至少需要调用 f $2 * n$ 次，才能得到完整导数。因此，将这些方法用于参数众多的问题时，其性能往往会随着参数数量增加而急剧下降。不过，数值微分的一个优点是，只需付出很少的努力，就能支持复杂的函数和参数。如方程 1 所示，大多数数值微分技术都不太依赖具体实现，这意味着只要函数接口符合数值微分算法的要求，就能处理任意复杂的函数实现。在某些应用中，这一优点可能胜过前面提到的一些问题。

RooFit 正是出于这个原因而采用数值微分：即使面对复杂的代码库，实现起来也相当简单。至于其他缺点，RooFit 通过使用复杂的缓存机制和其他数值逻辑来加以应对，在保持性能令人满意的同时，降低导数的最终误差。这些改进使 RooFit 拥有高效的数值微分流程，能够有效处理大规模分析；代价是代码库变得非常复杂，从而妨碍维护和新功能的开发。

2.2. 自动微分与 *Clad*

自动微分（AD）是一组用于计算机程序精确导数的技术。与数值微分不同，自动微分能够生成真正的导数，而不是估计值。它利用函数计算图中的微分链式法则对函数进行微分。

AD主要定义了两种用于计算函数导数的递归关系—— f-前向累积和反向累积。对于任意复杂的函数$y = f(g(x))$，可以将其拆分为如下中间步骤：

$$w_0 = x \quad w_1 = g(x) \quad w_2 = f(g(x)) = y \tag{2}$$

前向和反向累积模式定义如下：

$$\frac{\partial w_i}{\partial x} = \frac{\partial w_i}{\partial w_{i-1}} * \frac{\partial w_{i-1}}{\partial x} \quad \text{and} \quad \frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial w_{i+1}} * \frac{\partial w_{i+1}}{\partial w_i} \tag{3}$$

如上所述，前向累积模式会计算函数中每个子表达式相对于某个固定输入的导数（与数值微分颇为相似），而反向累积模式则会计算输出相对于每个子表达式的导数。反向模式的这一特性使其成为需要对大量参数求导时的理想选择，因为与前向累积模式相比，它所需的计算量更少。因此，我们的工作主要使用反向累积模式（或反向模式）。具体而言，我们使用 Clad [6], ，这是一种基于 LLVM 编译器基础设施 [8] 构建、作为 C/C++ Clang 编译器插件的源代码转换自动微分工具。Clad 在编译器层面检查目标函数的源代码，并生成另一个用于计算其导数的函数。图 1 展示了使用 Clad 求导时，一个示例函数及其导数的形式。Clad 特别适用于 HEP 工具，因为它通过 Cling [9], （ROOT 的 C++ 解释器）集成在 ROOT 中。

尽管自动微分（AD）在效率方面相较于数值微分具有明显优势，但它也存在一个局限：对带有副作用的代码（例如类和其他面向对象编程构造）的支持有限。在许多情况下，这些构造会隐藏 AD 生成正确导数所需的关键可微信息。尽管这种“隐藏”是为了让目标应用更易于使用，但它却成为 AD 工具的障碍。这使得将AD集成到 RooFit 中尤其困难，因为类和面向对象编程是其重要的设计元素。

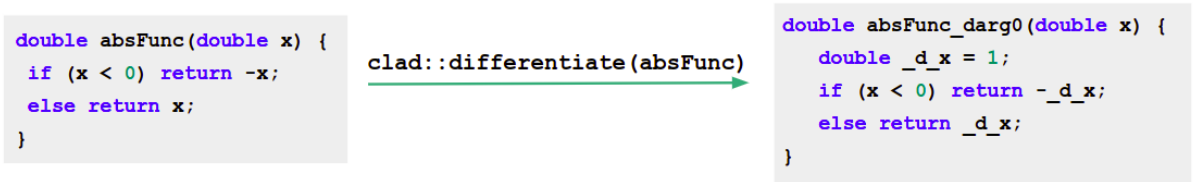


图 1：**一个由 Clad 求导的示例函数。**

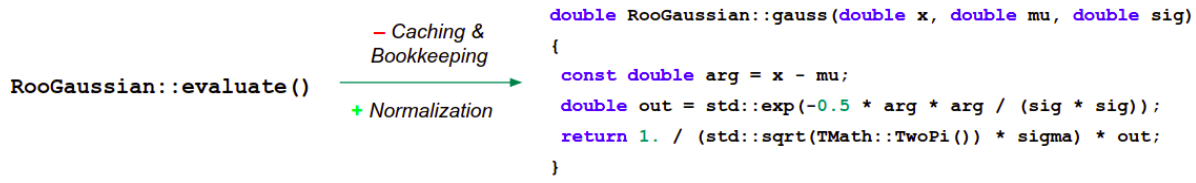


图 2: **一个示例，展示如何将 RooFit 类转换以暴露其微分属性。**

3. 设计与实现

正如前几节所述，RooFit 的面向对象设计使得直接使用 AD 工具对统计模型求导变得困难。我们需要引入一个中间步骤或转换，使这些模型更适合进行 AD。使用 Clad 这样的工具，意味着生成状态无关的代码，以一种 Clad 能够理解的方式展现 RooFit 模型的可微分性质。

构建此类变换的第一步，是公开 RooFit 类的相关微分属性。我们可以通过引入一个静态函数来实现这一点，该函数将类的数学含义转换为代码。图2 展示了如何将 `RooGaussian::evaluate()`转换为可由 Clad 求导的函数。

既然我们现在可以独立地转换每个类，就需要将统计模型中的所有这些转换组合成一段代码，以便之后由 Clad 对其进行微分。虽然可以将模型中各个节点/类的独立导数组合起来，从而得到完整模型的导数，但这种方式通常效率不高。先组合代码，再生成导数的一大优势在于，编译器能够更好地对组合后的代码和生成的导数执行编译器优化。此外，这种组合并微分 方法还带来了更好的调试体验，因为导数中的错误原因可以直接归因于某一行无状态代码，而不必追踪成员函数的深层调用栈。基于这些原因，在本工作中，我们采用组合并微分 方法。

实现组合并求导（或代码压缩）方法的一种方式，是为每个类引入一个“translate”函数。此“translate”函数会返回一个`std::string`，其中以代码形式表示该类底层的数学记法，之后可将其拼接成一个表示整个模型的字符串。

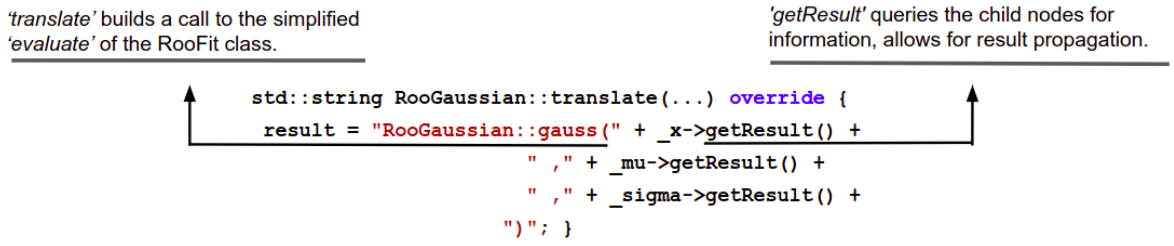
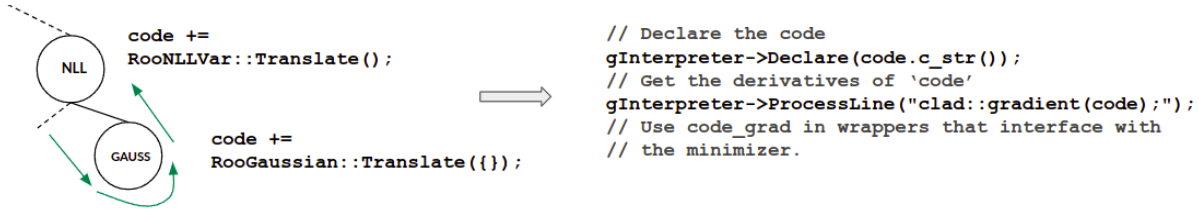


图 3：`RooGaussian` **的一个“translate”函数示例。**

此外，它还包含一些额外逻辑，用于将所需信息（例如传递给其他节点的输入、当前节点对其他节点产生的副作用等）正确传播给子节点。图 3 给出了 RooGaussian 类的 translate 函数示例。

为保留控制流，我们对模型执行深度优先搜索，并对访问到的每个节点调用 translate。搜索完成后，使用 ROOT 的 C++ 解释器 Cling 对生成的代码进行即时编译。图 4 描述了如何

一个示例 RooFit 模型可以被转换为代码，然后由 Clad 求导。



Fi图4: **一个示例 RooFit 模型的代码生成和微分。**

4. 结果

我们使用一个基于 HistFactory 模板 [5] 的示例测试了这项工作。负对数似然函数如下:

$$l = \sum_i \text{pois}(n_i|\alpha \cdot \nu_i) + l_{\text{aux}}. \tag{4}$$

此处，pois$(n_i|\alpha \cdot \nu_i)$（在公式4 中）表示泊松分布的负对数，其期望值为 $\alpha \cdot \nu_i$，在箱 i 中的观测值为 n_i。期望值是全局光度缩放常数 α 与三个缩放后的模板直方图 S、B_1 和 B_2 之和的乘积，分别代表一个信号样本和两个背景样本:

$$\nu_i = \mu S_i + \gamma_{1,i} B_{1,i} + \gamma_{1,2} B_{2,i}. \tag{5}$$

信号的缩放因子 μ（在公式 5 中）与箱无关，而 γ-因子则针对每个箱和背景分别独立。参数 α 受高斯分布约束，而每个 γ 都受泊松分布约束。所有这些约束都包含在辅助似然 l_{aux}中。这是对数据驱动的背景估计之上预测信号进行测量时的典型做法。在本研究中，我们通过改变箱数来改变模型中的参数数量。

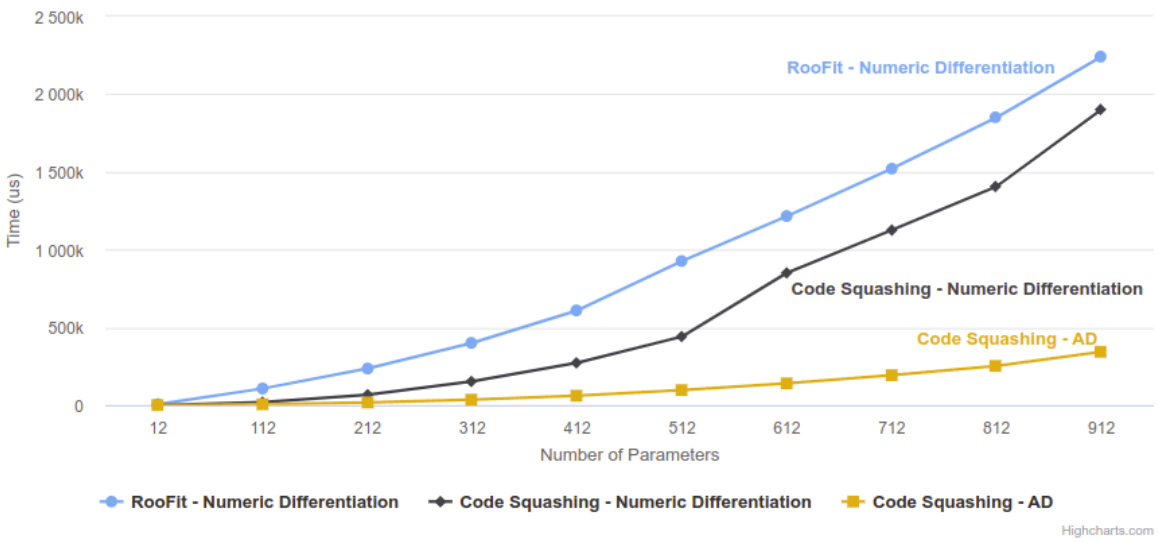


图5: **HistFactory模板上AD与数值微分的性能比较（定义于方程4）。**

图 5 展示了在不同计算配置下，对公式 4 中定义的模型进行最小化的性能比较。对于不同数量的参数，我们将结合代码（或代码压缩）并使用 AD 和数值微分的方法，与常规 RooFit 配置进行了比较。当参数数量约为 1,000 时，最小化速度提升了**5.5 倍** 。

5. 结论与未来工作

本研究表明，与使用带缓存的数值微分以避免重复计算相比，为解析梯度生成经过优化的可调用对象，能够显著提升 RooFit 的最小化性能。未来的工作将致力于在 ROOT 上游实现这一逻辑，使其适用于通用的 RooFit 模型。为此，未来还需在三个方面开展工作。第一，开发一个代码生成引擎，使其能够接收任意 RooFit 似然函数，并据此生成简单的 C++ 函数，以便 Clad 对其进行处理。第二，改进 RooFit 与 ROOT 所附带的最小化算法之间的接口，因为众多（且往往不必要的）抽象层使解析梯度的传递变得十分复杂。第三，研究最小化算法本身。现在，似然函数的一阶导数以及可能的二阶导数均可表示为可调用对象，且其计算成本低于数值近似，因此有必要重新审视最小化策略，以充分发挥解析梯度和 Hessian 矩阵的优势。

6. 致谢

本项目获得美国国家科学基金会资助，资助编号为 OAC-1931408，并依据合作协议 OAC-1836650 获得资助。

7. 参考文献

[1] WVęrkerke 和D.P.Kirkby，《用于数据建模的RooFit 工具包》，

eConf 0303241, 007, ，第 C 卷，第 2003 MOLT 页。[2] R. Brun、F.Rademakers、P. Canal、A. Naumann、O. Couet、L. Moneta、V. Vassilev、S. Linev、D. Piparo、G. GANIS、B. Bellenot、E. Guiraud、G.Amadio、wverkerke、P.Mato、TimurP、M.Tadel、wlav、E. Tejedor、J.Blomer、A. Gheata、S.Hageboeck、S. Roiser、marsupial、S. Wunsch、O. Shadura、A. Bose、CristinaCristescu、X. Valls和R. Isemann，《root-project/root：v6.18/02》，2019 年 8 月。 [在线]。可访问：https://doi.org/10 5281 3895860 。/zenodo。C. Burgard、J. Rembser、L. Moneta、P. Bos、W. Verkerke 和 [3] Z. Wolffs，《加速分析的新 RooFit 开发》，2022 年。 [在线]。可访问：https://agenda.infn.it/event/28874/contributions/169205/[4] E. Michalainas、J. Rembser、S.Hageboeck 和 L. Moneta，《使用GPU 加速及其他 RooFit 新闻》，*J. Phys. Conf. Ser.*，第2438卷，第 1 期，第012066 页，2023 年。[5] K. Cranmer、G. Lewis、L. Moneta、A. Shibata和 W. Verkerke，《HistFactory：用于创建供 RooFit 和 RooStats使用的统计模型的工具》，1456844，纽约大学，纽约，技术报告，2012 年。 [在线]。可访问：https://cds.cern.ch/record/[6]

V.Vassilev、M.Vassilev、A. Penev、L. Moneta 和 V. Ilieva，《Clad——使用Clang和 LLVM 进行自动微分》，第 608 卷，第1 期。IOP Publishing，2015 年 5 月，第 012055 页。 [在线]。可访问：https://iopscience.iop.org/article/10.1088/1742-6596/608/1/012055/pdf[7] F. James和 M. Roos，《Minuit：用于函数最小化以及参数误差和相关性分析的系统》， *Comput. Phys. Commun.*，第 10 卷，第 343–367 页，1975 年。[8] C. Lattner和 V.Adve，《LLVM：用于终身程序分析与转换的编译框架》，美国加利福尼亚州圣何塞，2004年3 月，第 75–88 页。[9] V. Vassilev、P. Canal、A. Naumann、L.Moneta和 P. Russo，《Cling——ROOT 6 的全新交互式解释器》。