

arXiv:2305.01582v3 [astro-ph.IM] 5 May 2023

使用 PySR 和 SymbolicRegression.jl 进行面向科学的可解释机器学习

Miles Cranmer[1,2]

[1]普林斯顿大学，新泽西州普林斯顿，美国[2]
Flatiron Institute，美国纽约州纽约市

2023年5月2日

PySR[3] 是一个面向实用符号回归的开源库。符号回归是一种机器学习，旨在发现人类可理解的符号模型。PySR 的开发旨在让科学领域更容易使用并推广符号回归；它基于一个高性能分布式后端、灵活的搜索算法，并提供与多个深度学习软件包的接口。PySR 的内部搜索算法是一种多种群进化算法，由独特的进化-简化-优化循环构成，专为优化新发现的经验表达式中的未知标量常数而设计。PySR 的后端是高度优化的 Julia 库 SymbolicRegression.jl[4]，，也可直接从 Julia 调用。它能够在运行时将用户定义运算符融合进 SIMD内核、执行自动微分，并将表达式种群分布到集群中的数千个核心上。在介绍这款软件的同时，我们还推出了一项名为“*EmpiricalBench*”的新基准测试，用于量化符号回归算法在科学领域的适用性。该基准测试衡量从原始数据集和合成数据集中恢复历史经验方程的能力。

1 引言

约翰内斯·开普勒通过研究第谷·布拉赫凭借肉眼观测获得的三十年数据，从中寻找规律，发现了著名的行星运动第三定律：（周期）$^2 \propto$（半径）^{3},。开普勒并没有借助计算机上的遗传算法，在所有可能的关系中进行搜索，而是凭借自己的几何直觉发现了这一规律。“正是开普勒第三定律，而不是一颗苹果，引导艾萨克·牛顿发现了万有引力定律” [1]。同样，普朗克定律并非从第一性原理推导而来，而是通过对数据进行拟合得到的一种符号形式 [2]。这种符号关系启发了量子力学的发展。

现在，第一步——发现经验关系——从数据中或基于人类直觉发现经验关系，既困难又耗时，即使对于已被证明属于 NP 难题 [3]的低维数据也是如此。在现代高维数据集中，如果不借助自动化工具，要发现简单的符号关系似乎是不可能完成的任务。

这引出了一个称为“符号回归”（SR）的优化问题。本文介绍一种用于执行符号回归的算法，并介绍专门用于科学领域方程发现的软件。

符号回归。 符号回归描述了一项监督学习任务，其中模型空间由解析表达式构成。通常通过多目标优化框架来解决这一问题，同时最小化预测误差和模型

[3]github.com/MilesCranmer/PySR[4] github.com/MilesCranmer/SymbolicRegression.jl

复杂度。在这一类算法中，人们不是在某些过参数化的通用模型中拟合具体参数，而是在简单解析表达式构成的空间中搜索，以寻找准确且可解释的模型。在科学发展史上，科学家常常凭借直觉并通过反复试验“手动”进行符号回归，以找到简单而准确的经验表达式。这些经验表达式可能进而带来新的理论见解，例如前文提到的开普勒和普朗克的发现，分别推动了经典力学和量子力学的发展。符号回归算法将经验方程的发现自动化，借助现代计算的强大能力检验远超人类直觉所能筛选的表达式。

符号回归早在 20 世纪 70 年代和 80 年代就开始作为科学研究工具发展，例如 [4],以及工具 BACON 的创建，时间是 [5] ，之后又有进一步发展，例如 [6, 7]。通过结合多种启发式策略，在可能的表达式树上进行暴力搜索，BACON及其后续工具，例如FAHRENHEIT [8], ，展示了从理想化数据中发现多种简单物理定律的能力。遗传算法能够提供更灵活的搜索空间，并减少对这些表达式的先验假设； [9] 的研究使其广为人知。最后，随着用户友好型工具 EUREQA 于 [10, 11, 12], 问世，以及 HeuristicLab 于 [13], 推出，符号回归开始适用于真实世界中含噪声的科学数据和经验方程。

面向科学的方程发现。 如今，Eureqa 是一款闭源专有工具，在科学领域已不再广泛使用。除 [14, 15, 16, 17], 等早期研究外，近年来，符号回归（SR）以及面向科学的符号回归领域也涌现出大量令人振奋的新进展。这些进展主要聚焦于对以树表示的表达式使用遗传算法，其中包括遗传算法方面的核心进展，例如 [18, 19], 提出了用于选择待变异个体的改进策略， [20, 21] 则采用基于新兴统计模式的自适应重组方法。此外，符号回归领域还涌现了全新的策略：特别是，我们此前的研究 [22, 23, 24] 提出将符号回归用于解释神经网络。神经网络能够高效地在高维数据中发现低维模式，因此，这种混合策略也提供了一种对高维或非表格数据进行符号回归的方法。

其他技术包括 [25, 26] ，它提出了类似马尔可夫链蒙特卡洛的符号回归采样方法； [27, 28] ，它训练深度生成模型来提出变异后的表达式； [29, 30], 它在数十亿个合成样本上预训练大型 Transformer 模型，以便从数据中自回归地生成表达式； [31, 32, 33], 它开发了利用符号回归发现对称性的技术； [34] ，它通过随机梯度下降，在一个非常大的预定义表达式中优化一组稀疏系数；最后，[35, 36], 它将遗传算法应用于灵活的微分方程空间。

除了树之外，还有一些其他数据结构用于表示符号表达式，例如命令式表示（“线性遗传编程”） [, 如 37, 38, 39, 40], ；以及序列表示 [, 如 29, 30, 32]。在树上的遗传算法之外，最受欢迎的替代方法之一，尤其是在偏微分方程研究中，是“SINDy”（非线性动力学稀疏辨识）[41, 42, 43] ，它将表达式表示为固定非线性表达式字典的线性组合。术语线性基的类似思想也用于“Operon”遗传算法的初始化 [44], 以及“FFX” [45]。该策略还可以与深度学习 [（如 43, 46, 47, 48, 49]）或其他技术结合，以在高维数据中寻找表达式 [（如 36]）。其他技术包括混合整数非线性规划 [50]、贪心树搜索 [51], ，以及将理论作为对模型空间的一种约束来结合使用 [52]。

然而，我们认为，许多现有算法缺少某些特性，因此难以在广泛的场景中实用地发现科学中的可解释方程。“科学符号回归”的要求远比应用于合成数据集的符号回归更为严苛，而许多算法实际上是通过合成基准测试进行训练的。我们将在下文列出这些要求。

1. Science 中的经验方程通常包含未知实常数[5] 一组

[5]有人可能会认为，科学模型中的任何常量都必须依赖于有限个已知的物理常数（例如光速）和数学常数（例如 π）。然而，这会引入显著更高的复杂度，而这

某些符号回归算法（例如 [31]）所考虑的可能表达式集合——由运算符、整数和变量构成——在给定最大规模下是离散且有限的。这个集合甚至可以通过暴力搜索来搜索，正如 [31]中结合对称性和量纲分析模块所采用的方法。然而，如果包含任意实常数，则必须搜索的方程集合将是不可数无限的。若干因素进一步加剧了搜索这一集合的难度：

2. 发现的方程必须能够深入揭示问题的本质。最能带来洞见、因而被科学家采用的方程，往往并非最准确的方程（许多其他机器学习算法都可以用来提高准确率），而是一个在准确率与简洁性之间取得平衡的方程。

3. 科学中的经验方程是从含有噪声的数据中发现的，而这些数据通常具有异方差结构。

4. 许多经验方程不可微，而是由仅在参数空间的部分区域内生效的不同表达式组成（例如，当 $x < 0$ 时为 $\exp(x)$，当 $x \geq 0$ 时为 $x^2 + 1$）。

5. 在科学研究中发现的表达式必须满足已知约束条件，例如质量守恒。

6. 经验关系通常由某个特定科学领域独有的运算符组成，因此方程搜索策略必须支持自定义运算符。

7. 观测数据通常具有高维特征，符号回归算法必须能够在内部处理这一问题，或与特征选择算法结合使用。

8. 搜索工具必须能够在非表格数据集中发现关系，例如序列、网格和图（例如，通过使用聚合算子，如 $\sum$）。

9. 最后，也是最重要的一点，任何对科学家有用的符号回归软件包都必须是开源的，

跨平台的，

跨平台的，易于使用，并具有用户界面

使用常见工具。更多讨论，

参考文献 [53, 54] 从天体物理学界的角度来看待这一问题。

贡献。本文介绍 PySR，这是一个用于科学符号回归的 Python 和 Julia 软件包。Python 软件包可通过 PyPI 和 Conda 以 pysr 的名称获取[6]，Julia 库则可通过 Julia 软件包注册表以 SymbolicRegression 的名称获取[7]。PySR 的每个部分都以科学发现为出发点，因为创建这个软件包的初衷，正是让本文作者能够在自己的领域中使用该工具进行方程发现。PySR 满足了上述所有科学经验方程发现要求。我们将在接下来的章节中对此进行详细讨论。

2 方法

首先，我们在 PySR 的第 2.1 节深入讨论其内部算法，然后在第 2.2 节介绍其软件实现。接着，我们将在第2.3 节、第2.4 节和第2.5 节讨论一些其他实现特性。

2.1 算法

PySR 是一种多种群进化算法，多个进化过程异步进行。PySR 的主循环针对每个种群独立运行，是一种经典的进化算法：通过锦标赛选择来选择个体（最早见于 [55] 的论文，并在 [56] 中得到推广），再通过多种变异和交叉操作生成新个体。用于符号回归的进化算法最早在 [9]，中得到广泛推广，此后一直主导着灵活符号回归算法的发展。PySR 对这种经典方法做了若干改动，其中一些改动受到近期研究成果的启发。

一个简单的进化算法按以下步骤进行：

1. 假设有一个由个体组成的种群、一个适应度函数以及一组变异算子。2. 随机从种群中选取一个大小为 n_s 的个体子集（例如，经典的

可能无法简化，因为基本常数与导出常数之间的关系可能极其复杂。

[6] github.com/MilesCranmer/PySR [7] github.com/MilesCranmer/SymbolicRegression.jl

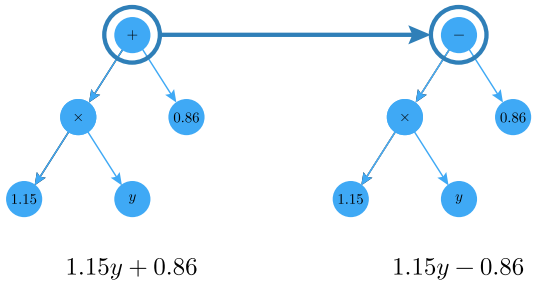

图 1：对表达式树应用变异操作。

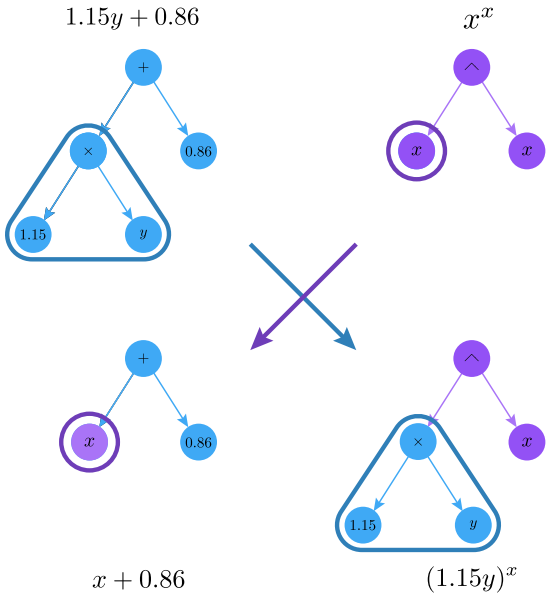

图 2：两棵表达式树之间的交叉操作。

锦标赛选择使用 $n_s = 2$，但也允许更大的锦标赛）。3. 通过评估该子集中每个个体的适应度来进行锦标赛。4. 以 p 的概率选出适应度最高的个体作为获胜者。否则，将该个体从子集中移除，并再次重复此步骤。如果只剩下一个个体，则选出它。因此，第一、第二、第三名及后续个体被选中的概率大致为 p，$p(1-p)$ $p(1-p)^2$，.....。5. 复制选中的个体，并从一组可能的变异中随机选择一种应用于副本。

6. 用变异后的个体替换种群中的一个成员。这里，可以替换种群或子集中的最弱个体。

该算法计算效率高，并且通过将步骤 2.拆分为多个个体组，实现大规模并行化。这些组可以独立演化，并在组间异步“迁移”。对于第一项修改，我们在步骤 5.中替换种群中年龄最大的成员，而不是适应度最低的个体，这种方法称为“年龄正则化”进化 [57]。我们根据个体在步骤 5.中创建时的 Unix 时间记录其年龄。将其纳入PySR 的动机，源于神经架构搜索取得的出色成果 [57, 58] ——不过值得注意的是，符号回归（SR）中更早也采用过类似的正则化方法，例如 [59] 和 [60] ——这些研究发现，这种简单的修改能够防止过早收敛，同时不损害性能。它可以避免种群过快地专门化，并陷入搜索空间的局部最小值。 [58] 中的实现使用 $p = 1$ 并探索 $4 \leq n_s \leq 64$，不过在此我们允许 p 小于 1，此外还有若干其他差异。

修改。我们在 PySR 中的算法最初于 2020 [61]，首次公开，它对这一进化算法进行了三项改动。我们将模拟退火 [62] 应用于步骤 4.：给定温度 $T \in [0, 1]$，变异会以一定概率被拒绝，该概率与变异后适应度的变化有关。拒绝概率为 $p = \exp\left(\frac{L_F - L_E}{\alpha T}\right)$，其中 L_F和 L_E分别表示变异后个体和原始个体的适应度，而 α 是一个超参数。这使进化过程能够在高温 和低温 阶段之间交替进行：高温阶段增加个体的多样性，而低温阶段则逐渐聚焦于适应度最高的个体。 α 控制温度的尺度，其中$\alpha \to \infty$ 等价于常规锦标赛选择，而 $\alpha \to 0$ 则拒绝所有会降低适应度的变异。由于 p 的取值也控制着种群的多样性，因此我们还可以将 p 的取值与温度关联起来：在较高温度下将 p 设得更大，在较低温度下将 p 设得更小。模拟退火是一种简单而强大的策略，可用于在离散

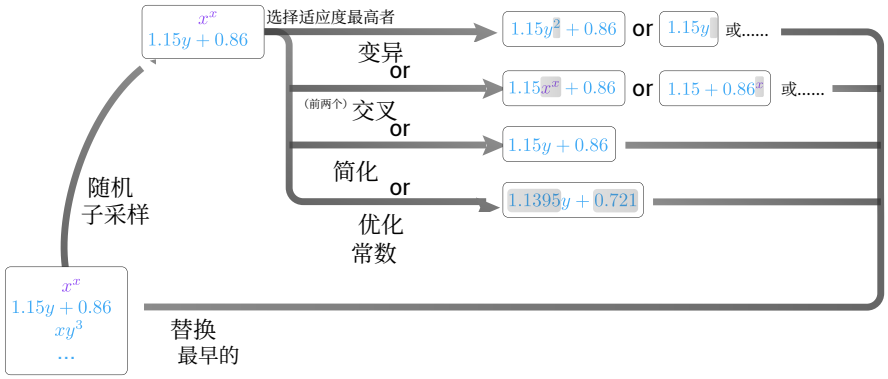

图 3: PySR 的内层循环。表达式种群会被随机子采样。从该子样本中进行锦标赛选择，并选出胜者进行繁殖：繁殖方式可以是变异、交叉、简化或显式优化。变异和交叉操作的示例见图 1 和 2。

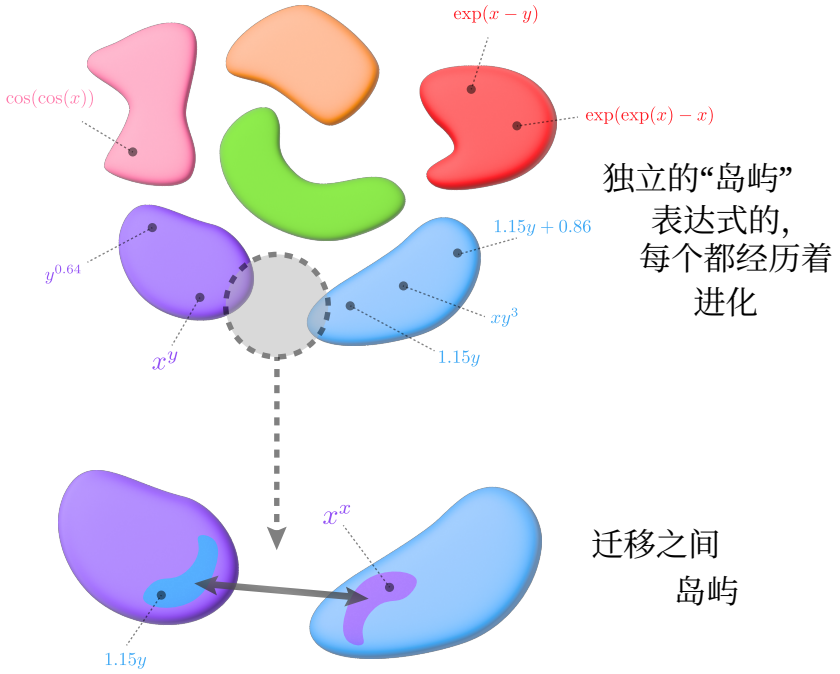

图 4: PySR 的外层循环。多个种群按照图 3 所述算法独立演化。经过指定轮数的进化后，岛屿之间会进行迁移。

空间中进行优化，我们在第 3 节中发现，模拟退火可以显著加快搜索过程。

我们对锦标赛选择所做的第二项修改针对的是我们希望纳入考量的个体：数学表达式。我们将遗传算法嵌入在进化-简化-优化循环中。“进化”是指基于锦标赛选择的进化过程，重复执行固定次数的变异。换言之，这是种群的一次完整进化。“简化”在此指方程简化——

在循环过程中，利用一组代数等价关系将方程简化为等价形式（由于这一过程发生得不够频繁，通常不会因限制搜索空间而损害发现过程）。该算法中的“优化”部分，是执行若干次经典优化算法迭代（默认使用 BFGS [63]，不过也可以使用`Optim.jl`软件包中的任何优化器 [64]），以显式优化每个方程中的常量。算法的这一部分显著提升了含有实常数的方程的发现效果，而这对于符号回归在科学领域的实际应用至关重要。至少从 [65]以来，已有研究表明，对表达式中的数值常量执行局部梯度搜索能大幅提升符号回归的性能——我们将这种搜索的优化版和随机版紧密集成在算法中。

在进入简化-优化 阶段之前执行多次变异（以及交叉操作），是因为有些方程只能通过冗余的中间状态才能得到。例如，$x*x-x*x$ 通常会简化为 0，但如果跳过这一步简化，下一次变异可能会将其中一个 x变为 y，从而得到$x*x-x*y$。偶尔才进行简化，可以在缩小搜索空间的同时，允许存在冗余但必要的中间表达式。

最后，我们所做的第三项改进是采用一种新颖的自适应简约性 指标。不过，我们首先给出复杂度的定义。

> **复杂度**：默认情况下，PySR 中的复杂度等于表达式树中的节点数，而不考虑各节点的内容。
>
> 不过，PySR 中的复杂度完全可以由用户配置。我们认为，“简单”表达式的最佳定义，是对用户而言最容易解释的表达式。含有二重对数函数的表达式，在粒子物理学家看来或许很常见——事实上，对于某些粒子物理数据而言，它可能是模型空间的一个很好的先验——但对流体动力学家来说却极为罕见；也许并非巧合的是，对于流体数据而言，它也不是一个好的先验选择。

惩罚复杂度的传统机制是使用一个固定的简约性，即添加一个等于.....复杂度的损失项。

一个表达式乘以一个常量因子：

$$\ell(E) = \ell_{\text{pred}}(E) + (\text{parsimony}) \cdot C(E), \qquad (1)$$

对于 $C(E)$，表达式 E 的复杂度以及 ℓ_{pred} 预测损失。相反，我们会自适应地调整每复杂度惩罚，使得每个复杂度级别上的表达式数量大致相同。这可以近似表示为：

$$\ell(E) = \ell_{\text{pred}}(E) \cdot \exp(\text{frecency}[C(E)]), \qquad (2)$$

其中，$C(E)$ 的“频率新近度”（frecency）是一个综合指标，用于衡量种群中复杂度为 $C(E)$ 的表达式出现的频率及其时效性。在 PySR 中，我们采用一种简单的方法：统计时间滑动窗口内表达式出现的次数，并将其除以一个可调节的常量。因此，这是一种根据表达式在种群中的出现频率，自适应地惩罚不同复杂度的简单方法。这种方法鼓励进化搜索既从简单但精度较低的表达式出发探索问题，也从复杂但精度较高的表达式出发进行探索。从定性角度来看，这似乎有助于缓解搜索过早陷入错误函数形式的情况，避免搜索只能对其进行小幅迭代改进，却难以重新开始从头探索的问题。

外层循环的伪代码见 PySR 算法 1，其中包括图 4所示的迁移步骤。算法 2 给出了主 进化-简化-优化循环的伪代码，将图 3中的部分示意图形式化。算法 3概述了变异集合，并补充了模拟退火。最后，算法 4描述了锦标赛选择策略。

附加功能。 PySR 包含多种附加功能，适用于不同类型的数据和场景：

- 含噪数据。 PySR 支持多种处理含噪数据的策略：– 首先，PySR 提供可选的去噪预处理步骤，通过在输入数据集上优化一个高斯过程，其核函数形式为 $k(x,x') = \sigma^2 \exp\left(-|x-x'|^2/2l^2\right) + \alpha\,\delta(x-x') + C$。该核函数由高斯核（用于插值数据集的标准核函数）、白噪声核（用于考虑固有噪声）和常数核（用于

算法 1： PySR。

```
输入    ： 𝒳，要为其寻找表达式的数据集
输出    ： 每种复杂度下的最佳表达式
参数    ：n_p，种群数量（=40）
参数    ：L，每个种群中的表达式数量（=1000）
参数    ：α_H，使用来自 H 的表达式进行替换的比例（=0.05）
参数    ： α_M，使用来自 ∪_i M_i (=0.05) 的表达式作为替换比例
函数 pysr(𝒳)
    对于i 在 range(n_p) // [代码]
        创建集合 P_i ，其中包含 L 个复杂度为 3 的随机表达式
        // 例如，(3.2 + x_1) 的大小为 3
        创建空集合 M_i // 用于存储在 P_i 中发现的最佳表达式
    end
    创建空集合 H // 用于存储整体最佳表达式
    对于 n 在 range(n_iter) // [代码]
        // 以下循环在多个工作进程上并行执行：
        对于 i 在 range(n_p)
            // 演化-简化-优化：
            P_i ← 演化(P_i, 𝒳)
            对于 E 在 P_i
                简化 E
                优化 E中的常量
                已将更新的 E 存储到 P_i
            end
            M_i ← 在 P_i 中每种复杂度下最准确表达式
            //（实际上， M_i 会在整个演化过程中不断更新）
            H ← 在 M_i ∪ H 中每种复杂度下的最准确表达式
            // 迁移：
            对于 E 中的 P_i
                如果rand() < α_H
                    将 E 中的 P_i 替换为来自 H的随机表达式
                end
                如果rand() < α_M
                    将 E 替换为 P_i 中的一个来自 ∪_{j≠i} M_j的随机表达式
                end
            end
        end
    end
    返回 H
结束
```

修改高斯过程的均值）叠加而成。优化此核函数后，高斯过程会为每个输入点预测去噪后的目标值，然后将这些值传递给 PySR 主算法。

– 第二，PySR 允许用户为每个输入数据点指定一组权重。例如，如果用户事先知道测量的不确定性为$\sigma > 0$，比如 σ 测量值的标准差，那么就可以根据 $1/\sigma^2$信噪比对损失中的每个数据点赋予权重。

– 第三，用户可以定义要优化的自定义似然函数，该函数也可以将权重纳入考量。这一点将在下一条中说明。

• 自定义损失函数。PySR 支持任意用户定义的损失函数，可通过字符串或函数对象传入。这些函数接受一个标量预测值和一个标量目标值，还可选择接受一个标量权重，并应返回大于 0 的实数。随后，将数据集中各点的损失相加。借助这一功能，用户可以定义任意回归目标、自定义似然函数（定义为单个数据点的对数似然），以及基于分类的损失函数。此外，还可以定义隐式目标。

• 自定义运算符。 PySR 最强大的功能之一是能够定义自定义运算符。不同的科学领域都有本领域特有的函数，它们会出现在许多公式中，并具有特定含义。因此，

算法 2： 进化。

输入 ： P，一组表达式
输入 ： $\mathcal{X}$ 如算法 1
输出 ： P，演化得到的表达式集
参数 ：n_c，每次调用 evolve() 时的变异数量 (=300000)
参数 ：p_{cross}，交叉的概率（=0.01）

函数 演化($P, \mathcal{X}$)
  对 k **在** 范围(n_c) // *[代码]*
    如果 随机数() $> p_{cross}$ // *[代码]*
      // 变异
      $E \leftarrow$ 锦标赛选择($P, \mathcal{X}$)
      $T \leftarrow 1 - \frac{k}{n_c}$ // 退火温度
      $E^* \leftarrow$ 突变(E, T)
      将最旧的表达式替换为 E^*
    else
      do
        // 交叉
        $E_1 \leftarrow$ 锦标赛选择(P, $\mathcal{X}$)
        $E_2 \leftarrow$ 锦标赛选择(P, $\mathcal{X}$)
        E_1^*, $E_2^* \leftarrow$ 交叉(E_1, E_2)
        将 P 中最旧的两个表达式替换为 E_1^* 和 $E^*{}_2$
      直到 满足_约束条件(E_1^*) 并且 满足_约束条件(E_2^*)
    end
  end
  返回 P
结束

这些运算符能够在符号回归库中使用非常重要，因此 PySR 允许使用自定义的一元或二元函数。只要函数可以定义为 $f : \mathbb{R} \to \mathbb{R}$ 或 $\mathbb{R}^2 \to \mathbb{R}$，就可以将其用作用户定义运算符。PySR 对内置运算符和用户定义运算符一视同仁。除了简化策略之外，PySR 无法区分+ 运算符和贝塞尔函数。

• 特征选择。与用于去噪的高斯过程预处理步骤类似，PySR 也会使用简单的降维预处理步骤。给定用户指定的要选择的特征数量，PySR 会先用梯度提升树算法拟合数据集，然后选出最重要的特征。这些特征会被送入主搜索循环。

•约束条件。PySR 允许为发现的表达式指定各种硬性约束。这些约束会在每次变异时强制执行：如果违反约束，则拒绝该变异。通过应用程序接口可以设置以下几种约束：

– 表达式的最大规模（

算法 3： 变异。

输入 ： E，一个表达式
输入 ：T，退火温度 $\in [0, 1]$
输出 ：E 的变异版本
参数 ：m_i，对于 $i = 1$、.....、8，概率权重为变异类型 i
参数 ：f，常数扰动尺度（=1）
参数 ：ϵ，最小扰动（=0.1）
参数 ：α，温度尺度（=0.1）

函数 变异(E, T) // *[代码]*
  根据约束条件调整权重 w_1，.....，w_8
  $i \leftarrow$ 按 w_1，.....，w_8 加权随机选择 1，.....，8
  do
    $E^* \leftarrow$复制(E)
    切换 i
      情况1 // 变异常量
        $a \leftarrow (1 + f \times T + \epsilon)^{2\times\text{rand}()-1}$
        如果 rand() < 0.5
          $a \leftarrow -a$
        end
        在 E^* 中乘以 a 个随机常量
      情况2 // 变异运算符
        随机将 E^* 中的一个运算符替换为一个度相同的运算符
      情况3 // 追加/前置节点
        将随机节点添加到 E^*的根节点或叶节点其中一处
      情形4 // 插入节点
        在 E^* 内插入一个随机节点
      案例5 // 删除子树
        从 E^*替换一个节点及其子节点使用常量或变量
      情况6 // 简化树
        简化 E^*
      情况7 // 完全创建新树
        $E^* \leftarrow$ 随机表达式
      情况8 // 无变异
        不执行任何操作
    end
  直到 满足_约束条件(E^*)
  C，$C^* \leftarrow$ 表达式的复杂度 E，E^*，分别
  L，$L^* \leftarrow$ 表达式的准确率 E，E^*，分别
  $q_{\text{anneal}} \leftarrow$ 指数$\left(-\frac{L^* - L}{\alpha \times T}\right)$
  $C \leftarrow$ 表达式的复杂度 E
  $q_{\text{parsimony}} \leftarrow \frac{\text{frecency}[C]}{\text{frecency}[C]}$
  如果 rand() $< q_{anneal} \cdot q_{parsimony}$
    返回 E^*
  else
    返回 E
  end
结束

运算符、变量和常数的出现次数），这为搜索空间设定了一个整体上限。

– 表达式的最大深度，可用于控制最终表达式的嵌套深度。

– 特定运算符中子表达式的最大大小。这指定了最大

算法 4: 锦标赛选择。

输入 : P，一个由表达式组成的种群
输入 : $\mathcal{X}$ 如算法 1
输出 : 一个表达式（获胜者是锦标赛选择）
参数 : n_s，锦标赛选择大小（=12）
参数 : $p_{\text{tournament}}$，选择的概率适应度最高的个体 (=0.9)

函数 锦标赛选择($P, \mathcal{X}$) // *[代码]*
  $Q \leftarrow$ 一个大小为 n_s 的随机子集，取自 P
  当 length(Q) > 1
    $E \leftarrow$获取_适应度最高者(Q)
    如果 rand() $< p_{tournament}$
      中断
    end
    从 Q 中移除 E
  end
  返回 E
结束

function 获取_最优个体(P) // *[代码]*
  $\ell_{\text{best}} \leftarrow \infty$
  对于 E，**在** P
    $C \leftarrow$ 复杂度为 E
    $\ell \leftarrow$准确率（ E）
    // 包含自适应简约性：
    $\ell \leftarrow \ell \times$exp(f时效性[$C$])
    如果 $\ell < \ell_{best}$
      $\ell_{\text{best}} \leftarrow \ell$
      $E^* \leftarrow E$
    end
  end
  返回 E^*
末尾

运算符参数中表达式的大小。例如，指定 $\wedge$ 运算符的最大参数大小为 $(-1, 3)$，意味着其底数可以是任意大小的表达式（以 -1 表示），而指数中的表达式最大只能为 3。这种运算符特定约束可以大幅降低所发现表达式的复杂度。

– 特定运算符组合的最大嵌套次数。例如，嵌套约束 {sin: {sin: 0, cos: 1}} 表示允许 sin(cos(x))，但不允许 sin(cos(cos(x))) 和 sin(sin(x))。

- 附加数据结构。请参阅深度学习接口部分2.3。

•积分与动力系统。参见深度学习接口部分2.3。

2.2 软件实现

其底层搜索算法PySR本身（如算法1 至4中以伪代码形式所描述的内容）采用纯 Julia 编写，并以 SymbolicRegression.jl[8]这一库的名称发布。Julia 兼具高级[9] 界面与媲美 C 等语言的极高性能++ 和 Rust[10]。然而，在这项工作中使用 Julia 的关键优势在于，它是一种即时编译（JIT）语言。这使得 PySR 能够利用静态编译库无法实现的优化：例如，将用户定义的运算符和损失函数编译到核心函数中。从性能角度来看，使用即时编译技术为PySR带来的最显著优势是，可以将多个运算符合并为单个编译后的运算符。例如，如果用户声明 + 和 – 为有效运算符，那么 PySR 就会为 +、–以及它们的组合编译支持 SIMD 的内核，例如：(a – b) + c，等等。对于每一种运算符组合，系统都会自动执行这一过程，组合深度最多可达两个运算符，即使是用户定义的运算符也不例外。仅通过这种方式融合运算符，便能显著提升表达式求值代码的执行速度，而该代码始终是PySR中的性能瓶颈。由于 Julia 采用即时编译，这些运算符无需预先定义在库中：它们同样能够获得与预定义运算符相当的性能。

PySR 为这个纯 Julia 后端搜索算法提供了一个简单的 Python 前端应用程序接口。该前端采用了流行的 Scikit-learn 机器学习库风格 [66]:

[8] https://github.com/MilesCranmer/SymbolicRegression.jl [9] docs.julialang.org/en/v1/ [10]julialang.org/benchmarks/

```
自pysr导入 PySRRegressor

# 声明搜索选项：模型 = PySRRegressor(模型
_选择="最佳"，一元_运算符=["cos"，"sin"],二
元_运算符=["+"，"-"，"/"，"*"],)



# 加载数据
X, y =加载_数据()
# X 形状：(n_行, n_个特征)
# y 形状：(n_行) 或 (n_行, n_个目标)

# 运行搜索：
模型。fit(X, y)

# 查看发现的表达式：print(model)


# 评估：使用帕累托前沿上的第 5 个表达式：y_预
测值 = model。predict(X, 5)#（未指定 `5`时,
它会选择一个表达式 ↪# 使复杂度与误差达到平衡)
```

并行化 PySR 通过彼此独立演化的表达式种群实现并行化，这一点被描述为允许大规模并行化的基础，见 [56]。在这一框架中，一个表达式种群会被分派给一个工作进程（如果是单节点，则为线程；如果是多节点，则为进程），随后该种群会经过许多次迭代进行演化。在一批迭代完成后，该种群会返回到主工作进程，主工作进程会执行以下几项任务：(1) 将种群中发现的每种复杂度下的最佳表达式记录为全局“名人堂”的一部分；(2) 记录当前表达式，以便它们能够“迁移”到其他种群；最后，(3) 从其他种群以及全局名人堂中随机“迁移”表达式到当前种群。在这些步骤完成后，该种群会再次被分派给一个工作进程，并进行进一步演化。这种并行化是异步的，不同种群可以以不同的速度完成演化，而不会拖慢其他种群。该技术与 [11]，中描述的方法大致相似，不过这里我们修改了迁移步骤，使其也能基于全局、永久性的精英档案进行迁移，而不是仅迁移当前种群成员。重新引入精英档案成员可以显著加快搜索速度。

2.3 接口

PySR 支持导出到多个常用 Python 库及其他格式。默认情况下，PySR 会将发现的表达式导出为 SymPy [67],；此外，SymPy 还可将表达式转换为 LaTeX，以便纳入研究报告。该功能的实现机制相当简单：PySR 使用恢复出的表达式的字符串表示，将符号名称替换为对应的 SymPy 名称（例如，cos 会变为 sympy.cos），然后通过 Python 解释器对其求值。需要注意的是，用户必须使用 SymPy 函数以及 Julia 函数来定义任何自定义运算符。

导出表达式时也采用了类似的方法，即对表达式的字符串表示求值，并将其转换为可调用的 NumPy、PyTorch 和 JAX 格式 [68, 69, 70]；它们都接受向量输入。对于 NumPy，表达式中的常量会嵌入可调用函数中。不过，对于 PyTorch 和 JAX，由于用户可能希望重新优化表达式中的常数（例如，[71]中就是如此），这些常数是可训练的。在 PyTorch 中，这意味着将这些常数创建为 PyTorch nn.Parameter，使 PyTorch 能够跟踪相对于这些参数的梯度，从而通过梯度下降进行训练。由于 JAX 是一种函数式语言，参数会以向量形式导出，用户随后将该向量传入可调用函数。这与使用 JAX 深度学习框架进行训练的方式类似。

2.4 自定义运算符

特定数学运算符在一组模型中的出现频率，会因这些模型所描述的科学领域而异。例如，流行病学模型通常会使用指数函数，以表示传染病的指数增长或衰减，但很少使用贝塞尔函数——后者在经典物理学应用中更为常见。符号回归能够生成可解释模型的根本原因是

模型的特点是，生成的表达式会使用特定领域中常见的运算符，领域科学家或许能够发现它们与现有模型之间的联系。在某些方面，这一进程与语言有关：黑箱机器学习模型用科学家无法理解的语言表示函数，而符号回归则使用领域内的语言。此外，鉴于科学建模具有模块化特性——许多复杂模型都建立在针对特定子系统的现有简单模型之上——从提高模型准确率的角度来看，使用常见运算符也合乎情理（一个有趣的具体示例见 [26]）。

严格来说，PySR 并没有任何内置运算符。由于 Julia 采用即时编译机制，Julia 基础库完整语言中的任何实标量函数都可用，并且能够在运行时编译到搜索代码中。这包括常用的运算符，例如 +、-、*、/、 ^、exp、log、sin、cos、tan、abs，以及许多其他运算符。

此外，由于许多科学领域都有各自独特的运算符，因此可以将任意 Julia 函数作为运算符传入，无论它是二元运算符（具有两个参数）还是一元运算符（具有一个参数）。任何形式为 $f : \mathbb{R} \to \mathbb{R}$ 或 $\mathbb{R}^2 \to \mathbb{R}$ 的函数，无论是否连续，都可以作为用户自定义运算符使用。该函数只需针对标量输入和输出进行设计，PySR 就会利用 Julia 自动编译并向量化它，并在可能的情况下生成 SIMD（单指令多数据）指令。在 PySR 中，一个示例如下：

```
运算符 = "special(x, y) = cos(x)* (x + y)"模型 = PySRRegressor(二元_运算符=[运算符])
```

其中，字符串 special(x, y) = cos(x) * (x+y) 是 Julia 代码，用于定义一个函数。这将定义一个二元运算符 special，并将其编译到搜索代码中。若要在各种导出功能中启用自定义运算符，用户还必须在 SymPy 中定义等效运算符（此处为 lambda x, y: sympy.cos(x) * (x + y)）；如果用户希望将其导出到这些框架，也需要定义 JAX 或 PyTorch 版本。

作为 Science 中的一个例子，在天体物理学中，经常会看到“折断幂律”：其指数在参数空间中取不同的值。可以在 PySR 中启用幂律运算符 ^，然后提供字符串 cond(x, y) = x < 0 ? 0 : y，该字符串定义了一个表达式的条件分支，其条件取决于 x 变量中定义的某个表达式（使用三元运算符 condition ? value1 : value2）。例如，pow(x, cond(x - 5, 3.4) - 2.1) 将定义以下折断幂律：

$$\begin{cases} x \geq 5, & x^{1.3} \\ x < 5, & x^{4.5}. \end{cases}$$

定义自定义运算符的重要性在于，该库并未专门针对某一固定的运算符集合进行调优；特定领域中常见的任何运算符都可以实现（只要它接受 1 到 2 个标量参数）。

2.5 自定义损失函数

在许多用于基准测试回归算法的机器学习玩具数据集中，均方误差（MSE 或 L_2 损失）通常用作学习目标 [72]。在贝叶斯框架中，MSE 等价于假设每个数据点都服从高斯分布，且各点的方差相同。最小化 MSE 等价于最大化高斯对数似然。然而，在科学研究中，通常使用针对特定问题定义的似然函数，而且它往往不服从高斯分布。因此，符号回归软件包允许使用自定义损失函数十分重要。 PySR 以与自定义运算符非常相似的方式实现了这一功能（见第 2.4节）。给定类似“loss(x, y) = abs(x - y)”这样的字符串，PySR 会将其传递给 Julia 后端，后端会自动将其向量化，并在整个搜索过程中将其用作损失函数。在贝叶斯语境下，这样便可以定义任意似然函数，即使其中包含非常复杂的分支逻辑也可以实现。加权损失也同样适用，例如“loss(x, y, w) = abs(x - y)* w”

3 评估

在第 1 节中，我们讨论了在科学领域发现符号表达式时的实际问题。在此，我们将 PySR 与现有工具在解决这些问题方面进行比较。

3.1 高层次比较

首先，我们对现有的各种符号回归工具进行高层次比较。
这些定性观点依据第 1 节中列出的理想特性得出。

✓	✓	-
✓	✓	✓
×	×	-
✓	×	✓
×	✓	-
×	×	✓
×	×	✓
×	×	✓
×	×	✓
×	✓	×
✓	✓	-
✓	✓	-
∗III	×	-
1a	3	6
✓	✓	✓
✓	✓	-
×	×	-
×	×	-
×	×	✓
×	×	✓
×	×	-
×	×	×
[30]	[21]	[23]
🔓	🔓	🔓

m. (2) Flexible basis of expressions,
allowed. (5) Any scalar tree, with

g this general method to a specific
methods which have deep learning

ed algorithms *could* support
practical use in science. Open

3.2 EmpiricalBench：经验科学符号回归基准

与历史上用于发现著名经验方程的数据集相比，现有的符号回归基准做了某些简化。在发现新表达式时，人们实际上并不知道表达式中的物理常数，因此必须学习实常数。例如，在费曼基准数据集 [73]，中，所有表达式都列出了相关的物理常数——但这些常数最初也必须与方程一起被发现。同样，在常用的合成“Nguyen”基准 [76]，中，其中包含诸如 $F_9 = \sin(x) + \sin(y^2)$之类的表达式，并不存在非整数常数。由 [77] 举办的“SRBench”竞赛在比较不同方法时非常严谨，但只包含一个现实世界任务（其真实值未知），其余表达式均为合成表达式。该基准的改进之处在于包含了可调节的噪声量。然而，其噪声模型非常简单：完全符合高斯分布，且不包含异方差性。尽管这类合成基准在符号回归研究中有其用途，但这类基准通常无法完全反映现实世界科学发现所面临的挑战。因此，我们在此引入一个新的基准，旨在准确呈现科学中的经验方程发现步骤。该数据集中的每个方程都是真实的经验表达式，科学家曾在某个时刻利用实验获得的、有噪声且不完美的数据发现了它。

如果一个算法要被认为对科学领域的方程发现有用，那么它应该能够发现这类关系。该基准测试的数据如图所示：5，方程及引用文献见表2。在使用原始数据的情况下，数据要么：(a) 取自公开可用的原始数据集；要么 (b) 使用`WebPlotDigitizer` [78]从表格或图表中数字化提取。例如，哈勃定律的数据是从原始的 1929 年论文中手动提取的 [79] 。在原始数据不可用的情况下，则根据方程在变量的合理范围内生成数据，并施加噪声。用于生成和评估该基准测试的代码和数据集可在论文代码仓库中获取，其基于 SRBench 竞赛的一个分支 [77, 80]。

如“回归目标”列所述，许多可以用对数单位轻松表示的定律，实际上却被设定为一个以观测单位为单位的问题。搜索算法必须自行发现这种更合适的单位。

许多现有的物理表达式搜索基准测试都基于理论推导出的方程，而非经验观测并经经验表述的方程。虽然这些基准测试可能有助于评估符号回归领域的某些方面，但它们并未针对符号回归最终的用途：经验发现。因此，本数据集中的所有方程在（若有）理论推导之前，都先经过了经验表述。因此，此处的物理变量直接对应于可观测量。此外，该数据集不包含任何相关的物理常数——算法必须像发现方程的科学家那样，自动找出这些常数。

3.2.1 EmpiricalBench 上的结果

我们分叉了 SRBench 竞赛代码库 [77]，，因为其作者成功地将大量现有方法汇集到一个具有通用应用程序接口的代码库中，这项工作令人印象深刻。不过，我们在此旨在研究每种算法的整个帕累托前沿，而不是像 SRBench 中那样只考察单个表达式。通过审查整个帕累托前沿中发现的表达式，我们不受选择标准这一略显武断的选择影响；不同方法采用的选择标准各不相同。这样，我们就能更准确地评估算法本身的性能。因此，我们修改了每种算法的界面，使其返回表达式的帕累托前沿；若无法做到，则改为返回若干候选表达式的列表。我们成功地在 EmpiricalBench 上测试了 PySR、Operon [44]，DSR [27]，EQL [34]，QLattice [75]，和 SR-Transformer [30] 。其他代码要么原本就未纳入 SRBench，要么与我们的测试不兼容，要么尽管付出了大量努力，仍无法在我们的系统上完成配置。

我们允许每种方法在运行 Rocky Linux 的 AMD Rome CPU 上使用 8 个核心，并允许它们搜索最长 1 小时。这些约束条件与 SRBench竞赛中的设置类似；该竞赛的作者曾针对这种设置逐一调整代码，因此我们认为这些设置最为公平。

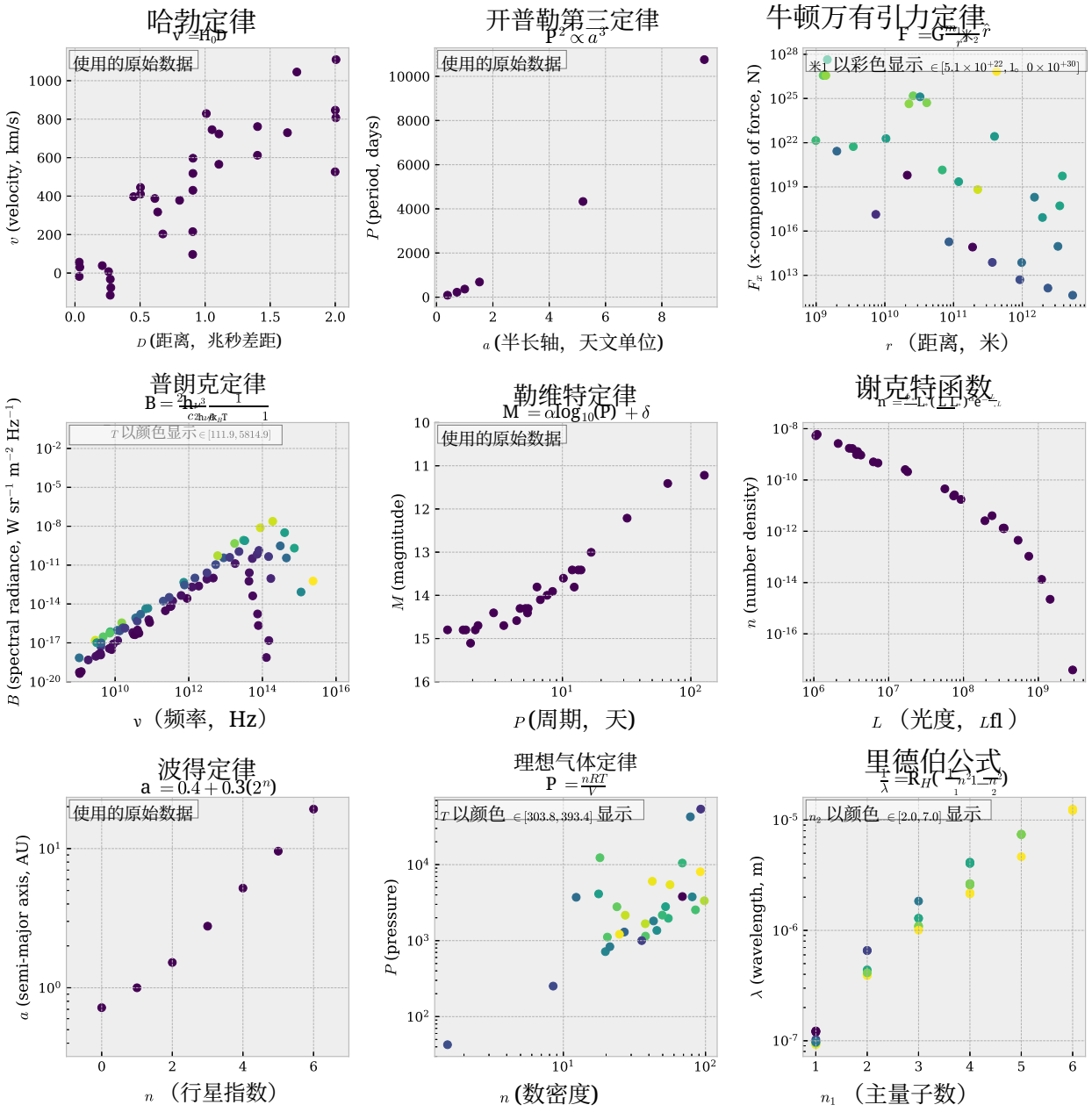

图 5：EmpiricalBench 中所有数据的可视化。EmpiricalBench 是一个面向科学领域的符号回归基准测试。对于依赖两个以上输入的关系，使用颜色标示额外的变量。在条件允许时，使用发现每条定律时的原始数据；否则，根据公式和变量的现实取值范围生成数据，并加入一定程度的噪声。

对于 EmpiricalBench 中的 9 个方程，我们对 6 种算法分别进行 5 次试验，并记录每次试验的帕累托前沿。每种算法都使用完整的数据集进行训练。这是因为我们的测试对象是输出表达式，而不是单独的测试数据集。之后，每个帕累托前沿通过肉眼分析，以确认其中是否包含真实表达式。这一过程由人工完成，因为使用 SymPy 自动检查等式往往会产生错误结果。此外，肉眼检查表达式可以忽略恢复出的表达式中的小误差，只要

Name	Law	提前引用
哈勃定律	$v = H_0 D$	[79]
开普勒第三定律	$P^2 \propto a^3$	[81]
牛顿万有引力定律	$\mathbf{F} = G\frac{m_1 m_2}{r^2}\hat{r}$	[82]
普朗克定律	$B = \frac{2h\nu^3}{c^2}\frac{1}{e^{h\nu/k_B T}-1}$	[83]
勒维特定律	$M = \alpha \text{对数}_{10}(P) + \delta$	[84]
谢克特函数	$n = \frac{\phi_*}{L_*}(\frac{L}{L_*})^\alpha e^{-\frac{L}{L_*}}$	[85]
波德定律	$a = 0.4 + 0.3(2^n)$	[86]
理想气体定律	$P = \frac{nRT}{V}$	[87]
里德伯公式	$\frac{1}{\lambda} = R_H(\frac{1}{n_1^2} - \frac{1}{n_2^2})$	[88] ¥

表 2：EmpiricalBench 中的表达式，以及与图 5 中数据集相关的表达式。这些表达式最初都是通过经验发现的。

	PySR	Operon	DSR	EQL	QLattice	SR-Transformer
哈勃	5/5 (5, 0, 0, 0)	0/5 (0, 5, 0, 0)	1/5 (1, 0, 4, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 5, 0, 0)	0/5 (0, 0, 0, 5)
开普勒	5/5 (5, 0, 0, 0)	0/5 (0, 5, 0, 0)	4/5 (4, 1, 0, 0)	0/5 (0, 0, 2, 3)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
牛顿	5/5 (5, 0, 0, 0)	1/5 (1, 2, 0, 2)	1/5 (1, 0, 4, 0)	0/5 (0, 0, 5, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
普朗克	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 1, 4)	0/5 (0, 0, 5, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
Leavitt	5/5 (5, 0, 0, 0)	0/5 (0, 0, 0, 5)	5/5 (5, 0, 0, 0)	0/5 (0, 0, 5, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
Schechter	5/5 (5, 0, 0, 0)	5/5 (5, 0, 0, 0)	5/5 (5, 0, 0, 0)	0/5 (0, 0, 4, 1)	5/5 (5, 0, 0, 0)	0/5 (0, 0, 0, 5)
Bode	5/5 (5, 0, 0, 0)	3/5 (3, 0, 0, 2)	1/5 (1, 0, 3, 1)	0/5 (0, 0, 4, 1)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
理想气体	5/5 (5, 0, 0, 0)	0/5 (0, 0, 0, 5)	5/5 (5, 0, 0, 0)	0/5 (0, 0, 4, 1)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
里德伯	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 5, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)

表 3：各算法在 EmpiricalBench 上的结果。给出的比例为重新发现的正确表达式数量除以试验总次数。括号中详细列出了五次试验的情况，顺序为：1）正确重新发现的数量；2）近似正确重新发现的数量；3）未能生成定义良好的表达式的运行次数；4）错误重新发现的数量。

整体函数形式是正确的。它也优于通过数值方式检查表达式，因为可以正确惩罚那些过于复杂但准确的表达式。

对于此项分析，每个帕累托前沿被归入以下四类之一：(1) 正确，如果帕累托前沿包含真实表达式；(2)近似正确，如果帕累托前沿中的某个表达式在某处与真实表达式相差一个常量因子；(3)失败，如果搜索未产生任何结果（例如，由于数值不稳定性、段错误、定义域错误等）；(4) 错误，如果帕累托前沿中未找到真实表达式。此项分析的所有原始数据以及发现的所有帕累托前沿均可在线获取，网址为 github.com/MilesCranmer/pysr_论文。

我们强调，符号回归算法及其软件实现本身的稳定性对于科学领域的实际应用具有核心重要性，因此，那些产生未定义表达式或导致程序崩溃的不稳定运行，也被纳入结果数据集和主要评分中。然而，还应注意，由于我们是PySR的作者，因此，与运行我们不熟悉的软件包相比，我们更有可能在稳定的环境中运行。例如，如表3所示，如果忽略运行失败的情况，DSR 的结果会有所改善，尽管它仍然在某些表达式上表现不佳。因此，表3详细列出了这些结果，以便更细致地理解实验结果。还需注意的是，该基准测试并未明确衡量其他基准测试中考虑的指标，例如评估次数（例如，遗传算法在这方面的效率非常低），也未考虑表达式的数值准确性。

3.3 讨论

还需要特别指出，这些结果来自专为本次竞赛准备的数据集。许多这类算法在针对每个表达式生成的合成数据集上，可能表现非常出色。例如，哈勃定律不过是一种线性关系，因此许多算法无法发现它，可能会让人感到意外。然而，要在基准测试中取得良好表现，每种算法都必须从图中所示的带噪声、有时还存在偏差的数据集中找出这些表达式。5。

另一个值得关注的现象是，在所有测试过的算法中，EQL 和 SR-Transformer 这两种纯深度学习方法找回的表达式最少。其中一种方法 EQL 以在线方式学习表达式，而 SR-Transformer 则基于预训练权重进行快速推理。尤其是 SR-Transformer，它在图形处理器集群上，利用数周时间预训练，所用数据是数十亿个随机生成的合成表达式。然而，采用经典启发式方法编写的“未训练”算法——PySR、Operon、DSR 和 QLattice——表现都胜过了这些现代深度学习模型。或许，这揭示了为真实世界数据和意料之外的“长尾”示例准备深度学习系统有多么困难；而手工编写的标准算法往往能在意料之外的示例上同样表现出色。尤其是在符号回归中，潜在表达式的空间极其庞大且具有非线性，因此插值技术可能难以奏效；试想一下，x/y和 $x \times y$ 只相差一次变异，但它们的数据表示却截然不同。尽管如此，纯深度学习策略似乎仍需改进架构或训练方式，才能在真实世界的符号回归场景中发挥作用，但它在例如 [23, 27, 30] 中取得的成功表明，此类混合方法确实具有很大的长期潜力，我们应继续探索这些方向。

4 结论

本文介绍了 PySR，这是一款面向实用符号回归的开源库。我们开发 PySR，旨在推动符号回归在科学领域的普及。借助强大的多种群进化算法、独特的进化-简化-优化循环、高性能分布式后端，以及与深度学习软件包的集成，PySR 能够加速从数据中发现可解释符号模型。

此外，通过推出新的基准测试 EmpiricalBench，我们提供了一种定量评估符号回归算法在科学应用中表现的方法。

我们的结果表明，尽管深度学习取得了进展，基于进化和其他未经训练的符号技术的经典算法（如 PySR、Operon、DSR 和 QLattice），在发现历史经验方程方面仍优于纯粹基于深度学习的方法 EQL 和 SR-Transformer。

来自原始数据集和合成数据集的表达式。这凸显了深度学习方法在应用于具有偏差和异方差性等特征的真实世界数据，以及表达式空间具有非线性时所面临的挑战。

然而，我们强调，深度学习技术——包括诸如 [30],这样的生成模型、诸如 [27]，以及符号蒸馏 [23]——都有望改进符号回归，我们鼓励进一步探索此类混合方法。

自 2020年 发布 PySR 以来的过去几年中，PySR 已被应用于多个子领域，帮助发现新模型，取得了令人振奋的成果。此处无法一一详述，下面列举一些示例： [89]使用 PySR 发现云层形成过程的全新符号参数化； [90] 将 PySR 与其他机器学习技术结合，发现各种材料中的电子转移规则； [91, 92, 93] 将 PySR 与量纲分析结合，发现新的天体物理关系； [94]在经济学领域使用 PySR，找出支配国际贸易的有效规则； [95] 展示如何使用 PySR 提取神经微分方程学得的动力学方程；最后， [96] 使用 PySR 发现可解释的引力波源种群模型。

从某种意义上说，这份近期应用列表为我们在第 1 节中倡导的以科学和用户为中心的方法提供了迄今最有力的验证，PySR 已成功应用于多个领域的模型发现。我们希望PySR 能继续发展成为社区工具，为研究人员创造价值，帮助他们在数据中发现可解释的符号关系，并最终为各自领域带来新的见解、理论和进展。

致谢 本软件使用 Python [97]和 Julia [98] 编程语言开发。PySR 的直接依赖项包括 numpy [68]、sympy [67]、sklearn [99]、和 pandas [100]，导出功能则由 jax [70] 和 py- torch [69]提供。SymbolicRegression.jl 的主要依赖项包括 Optim.jl [64]、LoopVectoriza- tion.jl [101]、Zygote.jl [102]、和 SymbolicU- tils.jl [103]。此外，制作本文稿时还使用了 matplotlib [104] 和 showyourwork [105]。部分图示的艺术灵感来自 Quanta杂志 [106]。

Miles Cranmer 感谢西蒙斯基金会为开展这项研究提供资源；感谢 Shirley Ho 和 David Spergel 与我进行了无数次富有启发性的讨论，内容涉及 PySR、对本文手稿的反馈、将其推广为科学领域的工具，以及对本项目的支持；感谢在 PySR 开发过程中始终提供反馈的研究合作者，包括 Pablo Lemos、Peter Battaglia、Steve Brunton、Jay Wadekar、Paco Villaescusa-Navarro、Kaze Wong、Elaine Cui、Christina Kreisch、Nathan Kutz、Drummond Fielding、Keaton Burns、Dima Kochkov、Alvaro Sanchez-Gonzalez、Christian Jespersen、Patrick Kidger、Kyle Cranmer、Niall Jeffrey、Ana Maria Delgado、Keming Zhang、Pierre-Alexandre Kamienny、Michael Douglas、Francois Charton；感谢所有出色的开源代码贡献者，包括 Mark Kittisopikul、T Coxon、Dhananjay Ashok、Johan Blåbäck、Julius Martensen、GitHub 用户 @ngam、Christopher Rackauckas、Jerry Ling、Charles Fox、Johann Brehmer、Marius Millea、GitHub 用户 @Coba、Pietro Monticone、Mateusz Kubica、GitHub 用户 @Jgmedina95、Michael Abbott、Oscar Smith，以及其他许多人；感谢 Marco Virgolín 对本文初稿提出极具价值的意见，以及提供的一般性反馈；感谢 Bill La Cava 提供反馈，并牵头发起 SRBench 项目，同时感谢其他所有 SRBench 组织者；感谢 Brenden Petersen 对 PySR 提供反馈，并就符号回归领域分享了富有见地的讨论；还要感谢许多其他人，他们通过电子邮件、Twitter、GitHub 问题反馈以及面对面交流等方式为本项目提供了支持。PySR 周围正在形成的社区及其所获得的积极反馈令我深受感动。谢谢大家。

参考文献

[1] 斯蒂芬·霍金。巨人的肩膀：物理学与天文学的伟大著作。Running，费城，宾夕法尼亚州；伦敦，2004年。[2]普朗克。关于改进维恩光谱方程。Friedr. Vieweg & Sohn，1900年。[3] Marco Virgolin 和 Solon P. Pissis。符号回归是NP难问题，2022年7月。

[4] Donald Gerwin。信息处理、数据推断与科学概括。行为科学，19(5):314–325，1974年。[5] Pat Langley。BACON：一种能够发现经验定律的产生式系统。载于国际人工智能联合会议，1977年。[6] Pat Langley。利用 BACON.3 重新发现物理学。载于第六届国际人工智能联合会议论文集——第*1*卷，IJCAI'79，第505–507页，1979年，美国加利福尼亚州旧金山。Morgan Kaufmann 出版公司。[7] Pat Langley、Gary L. Bradshaw 和 Herbert A. Simon。BACON.5：守恒定律的发现。载于国际人工智能联合会议，1981年。[8] Pat Langley 和 Jan M. Zytkow。用于经验发现的数据驱动方法。人工智能，40(1):283–312，1989年。[9]John R. Koza。将遗传编程作为通过自然选择对计算机进行编程的一种手段。统计与计算，4(2):87–112，1994年6月。

[10] Josh Bongard 和 Hod Lipson。《封面文章：非线性动力系统的自动化逆向工程》。美国国家科学院院刊，104(24):9943–9948，2007年6月。

[11] Michael Schmidt 和 Hod Lipson。从实验数据中提炼自由形式的自然定律。*Science*，324(5923):81–85，2009年4月。

[12] M. Schmidt 和 H. Lipson。隐式方程的符号回归。遗传编程理论与实践 *VII*，第 73–85 页，2010年。

[13] S. Wagner 和 M. Affenzeller。HeuristicLab：通用且可扩展的优化环境。载于 Bernardete Ribeiro、Rudolf F. Albrecht、Andrej Dobnikar、David W. Pearson 和 Nigel C. Steele（编），*Adaptive and Natural Computing Algorithms*，第 538–541 页，维也纳，2005 年。Springer。

[14] Kalyanmoy Deb、Samir Agrawal、AmritPratap 和 T. Meyarivan。《一种快速的精英非支配排序遗传算法，用于多目标优化：NSGA-II。载于 Marc Schoenauer、Kalyanmoy Deb、Günther Rudolph、Xin Yao、Evelyne Lutton、Juan Julian Merelo 和 Hans-Paul Schwefel 编，《自然界并行问题求解 *PPSN VI*》，《计算机科学讲义》，第849–858页，柏林、海德堡，2000年。Springer。[15] K. Deb、A. Pratap、S. Agarwal 和 T. Meyari-van。《一种快速且精英化的多目标遗传算法：NSGA-II》。《*IEEE*进化计算汇刊》，6(2)：182–197，2002年4月。[16]J.W. Davidson、D.A. Savic 和 G.A. Wal-ters。《符号与数值回归：实验与应用》。《信息科学》，150(1)：95–117，2003年。[17] Kyle Cranmer 和 R. Sean Bowman。《PhysicsGP：一种用于事件选择的遗传编程方法》。《计算物理学通讯》，167(3)：165–176，2005年5月。[18] William La Cava、Lee Spector 和 Kourosh Danai。《用于回归的 Epsilon-Lexicase 选择》。载于*2016*年遗传与进化计算会议论文集，GECCO '16，第741–748页，美国纽约州纽约市，2016年7月。计算机协会。[19] William La Cava、Thomas Helmuth、Lee Spector 和 Jason H. Moore。《对 Lexicase 选择和 epsilon-Lexicase 选择的概率与多目标分析》，2018年4月。[20] Marco Virgolin、Tanja Alderliesten、Cees Witteveen 和 Peter A. N. Bosman。《通过基因池最优混合和基于输入空间熵的构件学习实现可扩展的遗传编程》。载于遗传与进化计算会议论文集，GECCO '17，第1041–1048页，美国纽约州纽约市，2017年7月。计算机协会。[21] Marco Virgolin、Tanja Alderliesten、Cees Witteveen 和 Peter A. N. Bosman。《改进基于模型的遗传编程，以实现小型表达式的符号回归》。《进化-

计算》，29(2)：211–237，2021年6月。

[22]Miles D. Cranmer、Rui Xu、Peter Battaglia 和 Shirley Ho。利用图网络学习符号物理。*NeurIPS 2019 ML4Physics*研讨会，2019年11月。

[23] Miles Cranmer、Alvaro Sanchez-Gonzalez、Pe- ter Battaglia、Rui Xu、Kyle Cranmer、David Spergel 和 Shirley Ho。利用归纳偏置从深度学习中发现符号模型。*NeurIPS*，2020年6月。

[24] Miles Cranmer、Can Cui、Drummond B Field- ing、Shirley Ho、Alvaro Sanchez-Gonzalez、Kimberly Stachenfeld、Tobias Pfaff 等。用于可解释人工智能的解耦稀疏网络。稀疏神经网络研讨会，第 7 页，2021年7月。

[25] Ying Jin、Weilin Fu、Jian Kang、Jiadong Guo 和 Jian Guo。贝叶斯符号回归。2020年1月。

[26]Roger Guimerà、Ignasi Reichardt、Antoni Aguilar-Mogas、Francesco A. Massucci、Manuel Miranda、Jordi Pallarès 和 Marta Sales-Pardo。一位贝叶斯机器科学家，旨在协助解决具有挑战性的科学问题。*Science Advances*，6(5)，2020年。

[27]Brenden K. Petersen、Mikel Landajuela、T. Nathan Mundhenk、Claudio P. Santiago、Soo K. Kim 和 Joanne T. Kim。《深度符号回归：通过风险偏好策略梯度从数据中恢复数学表达式》，2021年4月。

[28] Li Li、MinjieFan、RishabhSingh 和 Patrick Riley。带有渐近约束的神经网络引导的符号回归。*arXiv*预印本 *arXiv:1901.07714*，2019年。

[29] Stéphane d'Ascoli、Pierre-Alexandre Kamienny、Guillaume Lample 和 François Charton。用于循环序列的深度符号回归，2022年6月。

[30] Pierre-AlexandreKamienny、Stéphane d' Ascoli、Guillaume Lample 和 François Charton。使用 Transformer 进行端到端符号回归。2022年4月。

[31]Silviu-Marian Udrescu、Andrew Tan、Jiahai Feng、Orisvaldo Neto、Tailin Wu 和 Max Tegmark。AI Feynman 2.0：利用图模块化的帕累托最优符号回归。发表于神经信息处理系统进展，第 33 卷，第 4860–4871 页。CurranAssociates,Inc.，2020年。

[32] Ziming Liu 和 Max Tegmark。AI庞加莱：从轨迹中进行机器学习守恒定律。*arXiv* 电子论文，页码 arXiv:2011.04698，2020年11月。

[33] Sebastian J. Wetzel、Roger G. Melko、Joseph Scott、Maysum Panju 和 Vijay Ganesh。通过解释孪生神经网络发现对称性不变量和守恒量。*Physical Review Research*，2(3):033499，2020年9月。

[34] Subham Sahoo、Christoph Lampert 和 Georg Martius。学习用于外推和控制的方程。机器学习研究论文集第80卷，第4442–4450页，瑞典斯德哥尔摩斯德哥尔摩展览中心，2018年7月。PMLR。

[35] Steven Atkinson、Waad Subber、Liping Wang、Genghis Khan、Philippe Hawi 和 Roger Ghanem。数据驱动发现自由形式的控制微分方程。*arXiv*预印本 *arXiv:1910.05117*，2019年。

[36]Andrew Slavin Ross、Ziwei Li、Pavel Perezhogin、Carlos Fernandez-Granda 和 Laure Zanna。《理想化模型中海洋次网格参数化的机器学习基准测试》，2022年10月。

[37] M. Brameier 和 W. Banzhaf。《线性遗传编程与神经网络在医疗数据挖掘中的比较》。*IEEE* 进化计算汇刊，5(1):17–26，2001年2月。

[38] Aytac Guven。《用于日流量时间序列建模的线性遗传编程》。地球系统科学杂志，118(2):137–146，2009年4月。

[39] He Ma、ArunachalamNarayanaswamy、Patrick Riley 和 Li Li。演化符号密度

泛函。《科学进展》，8(36):eabq0279，2022年9月。

[40]Douglas Mota Dias 和 Marco Aurélio C. Pacheco。《从符号回归问题出发描述量子启发式线性遗传编程》。载于*2012*年 *IEEE* 进化计算大会，第 1–8 页，2012年6月。

[41]Steven L. Brunton、Joshua L. Proctor 和 J. Nathan Kutz。通过稀疏识别非线性动力系统，从数据中发现控制方程。《美国国家科学院院刊》，113(15):3932–3937，2016年。[42] Samuel H Rudy、Steven L Brunton、Joshua L Proctor 和 J Nathan Kutz。数据驱动的偏微分方程发现。《科学进展》，3(4):e1602614，2017年。[43] Kathleen Champion、Bethany Lusch、J. Nathan Kutz 和 Steven L. Brunton。数据驱动的坐标和控制方程发现。*arXiv* 电子预印本，第 arXiv:1904.02107 页，2019年3月。[44]Bogdan Burlacu、Gabriel Kronberger 和 Michael Kommenda。Operon C++：一种高效的符号回归遗传编程框架。载于*2020*年遗传与进化计算大会伴随会议论文集，GECCO '20，第1562–1570页，美国纽约州纽约市，2020年7月。计算机协会。[45] Trent McConaghy。FFX：快速、可扩展、确定性的符号回归技术。载于 Rick Riolo、Ekaterina Vladislavleva 和 Jason H. Moore 编，《遗传编程理论与实践 IX》，遗传与进化计算，第235–260页。Springer，纽约州纽约市，2011年。[46] Bethany Lusch、J. Nathan Kutz 和 Steven L. Brunton。用于非线性动力学通用线性嵌入的深度学习。《自然通讯》，9:4950，2018年11月。[47]Gert-Jan Both、Subham Choudhury、Pierre Sens 和 Remy Kusters。DeepMoD：深度用于在含噪数据中进行模型发现的学习。2019年。[48]Zhao Chen、Yang Liu 和 Hao Sun。从稀缺数据中深度学习物理定律。2020年。[49] Christopher Rackauckas、Yingbo Ma、Julius Martensen、Collin Warner、Kirill Zubov、Ro- hit Supekar、Dominic Skinner 和 Ali Ramadhan。用于科学机器学习的通用微分方程。*arXiv*预印本 *arXiv:2001.04385*，2020年。[50] Alison Cozad 和 Nikolaos V. Sahinidis。一种用于符号回归的全局混合整数非线性规划方法。*MathematicalProgramming*，170(1):97–119，2018年7月。[51] Fabricio Olivetti de Franca。一种用于符号回归的贪心搜索树启发式算法。信息科学，442–443:18–32，2018年5月。[52] Cristina Cornelio、Sanjeeb Dash、Vernon Austel、Tyler Josephson、Joao Goncalves、Kenneth Clarkson、Nimrod Megiddo 等。AI Descartes：结合数据与理论以实现可推导的科学发现。*arXiv:2109.01634 [cs]*，2021年10月。[53]A. M. Price-Whelan、B. M. Sip'ocz、H. M. G"unther、P. L. Lim、S. M. Crawford、S. Con- seil、D. L. Shupe 等。Astropy项目：构建开放科学项目及v2.0核心软件包的状态。*aj*，156:123，2018年9月。[54]Astropy Collaboration、Adrian M. Price-Whelan、Pey Lian Lim、Nicholas Earl、Nathaniel Starkman、Larry Bradley、David L. Shupe 等。Astropy项目：维系并发展一个以社区为导向的开源项目，以及核心软件包的最新主要版本（v5.0）。《天体物理学杂志》，935:167，2022年8月。[55] Anne Brindle。用于函数优化的遗传算法。博士论文，1980年。[56]David E. Goldberg 和 Kalyanmoy Deb。遗传算法中所用选择方案的比较分析。载于 GREGORY J. E. Rawlins 编，遗传学基础

Algorithms，第1卷，第69–93页。爱思唯尔，1991年1月。[57] Esteban Real、Alok Aggarwal、Yanping Huang 和 Quoc V. Le。用于图像分类器架构搜索的正则化进化。人工智能 *AAAI* 大会论文集，33(01):4780–4789，2019年7月。[58] Esteban Real、Chen Liang、David So 和 Quoc Le。AutoML-Zero：从零开始进化机器学习算法。载于国际机器学习大会，第8007–8019页。PMLR，2020年11月。[59]Gregory S. Hornby。ALPS：用于减少过早收敛问题的分层年龄种群结构。载于第八届遗传与进化计算年会论文集，GECCO '06，第815–822页，美国纽约州纽约市，2006年7月。计算机协会。[60]Michael D. Schmidt 和 Hod Lipson。年龄-适应度帕累托优化。载于第十二届遗传与进化计算年会论文集，GECCO '10，第543–544页，美国纽约州纽约市，2010年7月。计算机协会。[61]Miles Cranmer。PySR：Python/Julia中的快速并行符号回归。Zenodo，2020年9月。[62] S. Kirkpatrick、C. D. Gelatt 和 M. P. Vecchi。模拟退火优化。*Science*，220:671–680，1983年5月。[63] C. G. Broyden。一类双秩最小化算法的收敛性 1：一般考虑。*IMA*应用数学杂志，6(1):76–90，1970年3月。[64] Patrick Kofod Mogensen 和 Asbjørn Nilsen Riseth。用于Julia的数学优化软件包Optim。开源软件杂志，3(24):615，2018年。[65] Alexander Topchy 和 W. F. Punch。基于数值叶节点值的局部梯度搜索加速遗传编程。载于第三届遗传与年会论文集

进化计算，GECCO'01，第155–162页，美国加利福尼亚州旧金山，2001年7月。Morgan Kaufmann 出版公司。[66] Fabian Pedregosa、Gaël Varoquaux、Alexandre Gramfort、Vincent Michel、Bertrand Thirion、Olivier Grisel、Mathieu Blondel 等。Scikit-learn：Python 中的机器学习。机器学习研究期刊，12：2825–2830，2011年11月。[67] Aaron Meurer、Christopher P. Smith、Mateusz Paprocki、Ondřej Čertík、Sergey B. Kirpichev、Matthew Rocklin、AMiT Kumar 等。Sympy：Python 中的符号计算。*PeerJ* 计算机科学，3：e103，2017年1月。[68]Charles R. Harris、K. Jarrod Millman、Stéfan J. van der Walt、Ralf Gommers、Pauli Virtanen、David Cournapeau、Eric Wieser 等。使用 NumPy 进行数组编程。*Nature*，585(7825)：357–362，2020年9月。[69]Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen 等。PyTorch：一种命令式风格的高性能深度学习库。载于 H. Wallach、H. Larochelle、A. Beygelzimer、F. d'Alché-Buc、E. Fox 和 R. Garnett 编，神经信息处理系统进展*32*，第8024–8035页。Curran Associates, Inc.，2019年。[70] James Bradbury、Roy Frostig、Peter Hawkins、Matthew James Johnson、Chris Leary、Dougal Maclaurin、George Necula 等。JAX：Python+NumPy 程序的可组合变换，2018年。[71] Pablo Lemos、Niall Jeffrey、Miles Cranmer、Peter Battaglia 和 Shirley Ho。利用深度学习和归纳偏置重新发现牛顿引力和太阳系性质。投稿中，2022年。[72]Léo Grinsztajn、Edouard Oyallon 和 Gaël Varoquaux。为什么基于树的模型在表格数据上仍然优于深度学习？，2022年7月。[73]Silviu-Marian Udrescu 和 Max Tegmark。AI feynman：一种受物理学启发的方法

用于符号回归。《科学进展》，6(16)：eaay2631，2020年。

[74] Alan A. Kaptanoglu、Brian M. de Silva、Urban Fasel、Kadierdan Kaheman、Andy J. Goldschmidt、Jared L. Callaham、Charles B. Delahunt 等。《PySINDy：用于稳健稀疏系统辨识的综合性 Python 软件包》。2021年11月。

[75]Kevin René Broløs、Meera Vieira Machado、Chris Cave、Jaan Kasak、Valdemar Stentoft-Hansen、Victor Galindo Batanero、Tom Jelen 和 Casper Wilstrup。《一种使用 Feyn 进行符号回归的方法》。2021年4月。

[76] Nguyen Quang Uy、Nguyen Xuan Hoai、Michael O'Neill、R. I. McKay 和 Edgar Galván-López。《遗传编程中的基于语义的交叉：在实值符号回归中的应用》。《遗传编程与可进化机器》，12(2)：91–119，2011年6月。

[77] F. O. de Franca、M. Virgolin、M. Kommenda、M. S. Majumder、M. Cranmer、G. Espada、L. Ingelse 等。《面向数据科学的可解释符号回归：2022年竞赛分析》，2023年4月。

[78] Ankit Rohatgi。WebPlotDigitizer：4.6版，2022年。

[79]埃德温·哈勃。河外星云中距离与径向速度之间的关系。《美国国家科学院院刊》，15(3):168–173，1929年3月。

[80] William La Cava、Patryk Orzechowski、Bogdan Burlacu、Fabrício Olivetti de França、Marco Virgolin、Ying Jin、Michael Kommenda 和 Jason H. Moore。《当代符号回归方法及其相对性能》，2021年7月。

[81] 约翰内斯·开普勒。《世界的和谐》。奥地利林茨，1619年。

[82] 艾萨克·牛顿。《自然哲学的数学原理》。奉英国皇家学会之命，由 Joseph Streater 印刷，伦敦，英格兰，1687年。

[83] 马克斯·普朗克。《热辐射理论》。P. Blakiston's Son & Co，费城，1914年。

[84] Henrietta S. Leavitt 和 Edward C. Pickering。小麦哲伦云中 25 颗变星的周期。哈佛大学天文台通告，173:1–3，1912年3月。[85] William H. Press 和 Paul Schechter。通过自相似引力凝聚形成星系和星系团。《天体物理学杂志》，187:425–438，1974年2月。

[86] Charles Bonnet。《Contemplation de La Nature》。阿姆斯特丹，1764年。

[87]Émile Clapeyron。《论热的动力》。综合理工学院学报，1835年。

[88] Johannes Rydberg。《化学元素发射光谱结构研究》。《瑞典皇家科学院院刊》，1889年。

[89]Arthur Grundner、Tom Beucler、Pierre Gen- tine 和 Veronika Eyring。《基于数据驱动的云量参数化方程发现》，2023年4月。

[90] Yanzhang Li、Hongyu Wang、Yan Li、Huan Ye、Yanan Zhang、Rongzhang Yin、Haoning Jia 等人。《机器学习揭示的压力下矿物电子转移规则》。*NatureCommunications*，14(1):1815，2023年3月。

[91] Konstantin T. Matchev、Katia Matcheva 和 Alexander Roman。利用量纲分析和符号回归对系外行星凌星光谱进行解析建模。《天体物理学杂志》，930(1):33，2022年5月。

[92]Digvijay Wadekar、Leander Thiele、J. Colin Hill、Shivam Pandey、Francisco Villaescusa- Navarro、David N. Spergel、Miles Cranmer 等人。低晕质量下的 SZ 通量-质量（Y-M）关系：利用符号回归加以改进，并对重子反馈施加强约束。2022年9月。

[93]Digvijay Wadekar、Leander Thiele、Francisco Villaescusa-Navarro、J.Colin Hill、Miles Cran- mer、David N. Spergel、Nicholas Battaglia，

等。利用机器学习扩展天体物理标度关系：应用于减少SZ流量-质量散布，2022年1月。[94] Sergiy Verstyuk和Michael R. Douglas。利用机器学习求解国际贸易的引力方程。*SSRN*电子期刊，2022年。[95] Patrick Kidger。论神经微分方程，2022年2月。[96] Kaze W. K. Wong和Miles Cranmer。自动发现可解释的引力波种群模型。*ML4Astro*研讨会 @ *ICML 2022*，2022年7月。[97] Guido Van Rossum和Fred L. Drake。*Python 3*参考手册。CreateSpace，斯科茨谷，加利福尼亚州，2009年。[98] Jeff Bezanson、Alan Edelman、Stefan Karpin- ski和Viral B. Shah。Julia：数值计算的新方法。*arXiv:1411.1607 [cs]*,2015年7月。[99] Fabian Pedregosa、Gaël Varoquaux、Alexandre Gramfort、Vincent Michel、Bertrand Thirion、Olivier Grisel、Mathieu Blondel等。Scikit-learn：Python中的机器学习。机器学习研究杂志，12(85):2825–2830，2011年。[100] Wes McKinney。Python统计计算的数据结构。载于Stéfan van der Walt和Jarrod Millman编，第九届*Python*科学会议论文集，第56–61页，2010年。[101] Chris Elrod。LoopVectorization.jl：用于循环向量化的宏。JuliaHub，2022年。[102] Michael Innes。不要展开伴随：对SSA形式程序求导。*CoRR*，abs/1810.07951，2018年。[103] Shashi Gowda、Yingbo Ma、Alessandro Cheli、Maja Gwóźdź、Viral B. Shah、Alan Edelman和Christopher Rackauckas。通过多重分派实现高性能符号数值计算。*ACM*计算机代数通讯，55(3):92–96，2022年1月。[104] Michael Droettboom、John Hunter、Thomas A Caswell、Eric Firing、Jens Hedegaard Nielsen、Phil Elson、Benjamin Root 等。Matplotlib：Matplotlib v1.5.1，2016年1月。[105] Rodrigo Luger 和贡献者。展示你的- 工作！，2021年。[106] Charlie Wood。强大的“机器科学家”从原始数据中提炼物理定律。*Quanta*杂志，2022年5月。