SaddleScape V1.0：基于高指标鞍点动力学构造解景观的 Python 软件包

Yuyang Liu* Hua Su† Zixiang Xiao‡ Lei Zhang§ Jin Zhao¶2026年1月6日

arXiv:2601.01081v1 [math.NA] 3 Jan 2026

摘要

我们介绍 SaddleScape V1.0，这是一款专为探索和构造复杂系统中的解景观而设计的 Python 软件包。该软件包实现了高指标鞍点动力学（HiSD）框架及其变体，包括适用于非梯度系统的广义HiSD和加速HiSD。SaddleScape V1.0 通过动态更新状态估计及用于表征鞍点局部流形的相关子空间，能够系统地识别临界点，包括局部极小值和高指标鞍点。它既支持由能量函数/泛函定义的梯度系统，也支持一般的非梯度自治动力系统。其主要功能包括对符号输入进行自动微分、海森矩阵-向量乘积的数值近似技术、多种特征值求解器，以及用于构造解景观的算法。该软件提供易于使用的接口、灵活的参数配置、轨迹与景观可视化工具，以及数据导出功能。通过提供高效且易于使用的先进鞍点动力学实现，SaddleScape V1.0 促进了解景观的构造，帮助各个科学领域的研究人员深入洞察复杂系统的层级结构。源代码可在代码仓库 https://github.com/HiSDpackage/saddlescape 中获取。该软件包的介绍网站为 https://hisdpackage.github.io/saddlescape。

关键词： SaddleScape、解景观、高指标鞍点动力学、鞍点庞加莱 ts

1 引言

物理学、化学、计算机科学及相关领域的许多实际问题，都可以表述为多变量非线性函数/泛函的最小化问题，这些函数/泛函可表示蛋白质折叠中的氨基酸位置 [15, 21, 26], 、Lennard-Jones团簇中的原子坐标 [36], ，或人工神经网络的参数 [7–9]。能量函数/泛函的每个局部极小值都对应一个亚稳态，不同的极小值点之间由能垒隔开。位于两个极小值之间能量最低的能垒处的状态称为过渡态。

*北京大学数学科学学院，中国北京，100871。（liuyuyang@stu.pku.edu.cn）。

†北京大学北京国际数学研究中心，中国北京，100871。（suhua@pku.edu.cn）。

‡北京大学数学科学学院，中国北京，100871。（xiaozixiang@stu.pku.edu.cn）。§北京大学北京国际数学研究中心、定量生物学中心、机器学习研究中心，中国北京，100871。（zhangl@math.pku.edu.cn）。

¶北京，中国，邮编 100048，首都师范大学多学科研究院和北京应用数学国家研究中心。（zjin@cnu.edu.cn）。

过渡态在许多科学领域都发挥着关键作用，例如在相变中寻找临界晶核和跃迁路径 [6, 52], 、计算化学反应的反应速率 [1, 14],，以及分析生物动力学 [35, 50, 53]。

除了局部极小值和过渡态之外，能量函数/泛函的驻点还包括高指标鞍点。从数学上看，这些驻点是 $\nabla E(\boldsymbol{x}) = 0$ 的解。对于非退化鞍点，莫尔斯理论 [24] 通过莫尔斯指标刻画其性质。莫尔斯指标定义为海森矩阵在其上负定的子空间的最大维数。例如，亚稳态的莫尔斯指标为 0，而过渡态的莫尔斯指标为 1。本文将莫尔斯指标为 $k > 0$ 的非退化临界点称为指标-k 鞍点（为方便起见，有时也将极小值点称为指标-0 鞍点）。

因此，要深入理解能量景观，不仅要定位极小值点，还必须识别连接它们的指标为1的鞍点。此外，较低指标和较高指标的鞍点可以通过适当算法生成的动力学搜索路径相互连接。揭示这些连接关系有助于深入洞察能量景观的结构，进而解释各种物理和化学现象。然而，大多数传统的鞍点搜索数值方法，例如同伦方法 [3, 22] 和去膨胀技术 [10], ，并未明确处理莫尔斯指标或鞍点之间的连通性。为此，Zhang等人提出并分析了高指标鞍点动力学（HiSD）算法 [47], ，该算法适用于搜索指定指标的高指标鞍点。该算法能够系统地搜索指定指标的鞍点，并揭示它们之间的相互联系。

在此基础上，Zhang等人提出了解景观的概念[45, 46]：一张由所有驻点及其相互连接构成的路径图。该图描绘了一种层级结构，其中鞍点按指标组织，展示了高指标鞍点与低指标鞍点之间的关系。这种层级结构加深了我们对不同指标鞍点之间关系的理解，并使鞍点搜索更加系统、全面，从而有助于深入认识能量景观。

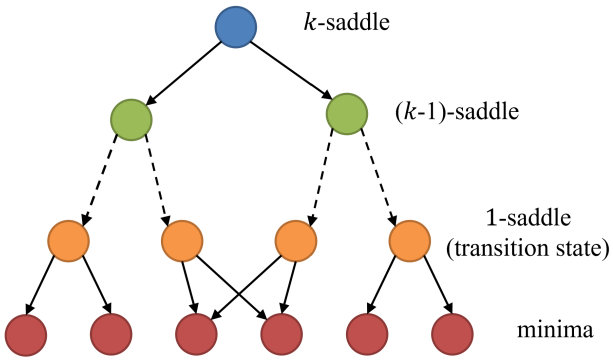


图 1：解景观示意图。节点表示具有不同 Morse 指标的鞍点，有向边表示鞍点之间的连接关系。

后续研究将解景观概念和 HiSD 方法从能量景观（梯度系统）拓展到了非梯度自治系统 $\dot{\boldsymbol{x}} = \boldsymbol{F}(\boldsymbol{x})$。这类系统广泛存在于描述物理、化学和生物动力学的模型中，例如动力学方程[4], 、与纳维–斯托克斯方程 [34], 相关的降阶模型，以及生化反应 [5, 25, 27]。由此提出了广义高指标鞍点动力学（GHiSD） [46]。该框架推广了能量景观（梯度系统）的情形，在该情形中，$\boldsymbol{F}(\boldsymbol{x}) = -\nabla E(\boldsymbol{x})$。此外，针对具有非线性等式约束的系统，还发展了约束高指标鞍点动力学（CHiSD） [43] 。综述文章 [51]系统阐述了这些鞍点动力学算法以及解景观的构造方法。

近年来，将 HiSD 与其他数值方法结合，为构建解景观提供了更高效的实现方式和坚实的理论支持。特别是，具有代表性的变体包括加速高指数鞍点动力学（AHiSD） [19], ，该方法引入动量以加快收敛；改进的高指数鞍点动力学（iHiSD） [33],

该方法结合鞍点动力学与梯度流，为算法的全局收敛性提供理论依据；此外，还有数据驱动的 HiSD 变体，它们采用基于采样数据构建的代理模型，例如基于高斯过程的模型 [58] 和基于神经网络的模型 [17], 等。此外，研究人员将不同的数值格式与 HiSD 相结合，提出了各种有效的算法，并开展了理论收敛性和误差分析[18, 20, 23, 32, 54–57, 59]。

除了理论和数值方法的发展之外，这些算法还已成功应用于研究各种系统的结构。在凝聚态物理学中，解景观概念和鞍点动力学已被用于描述晶体势能面 [16], ，计算准晶的相变和成核 [44, 65], ，以及构建和分析液晶系统中的缺陷景观 [11–13, 28–31, 38, 45, 48]。在这些复杂系统中，HiSD 算法提供了一种系统化的方法来定位成族的鞍点和极小值点，并将其组织成解景观，从而阐明驻态之间的相互联系。

在理论物理学中，研究人员通过绘制不同旋转频率下相应的解景观，应用这些算法探索旋转玻色–爱因斯坦凝聚体的激发态 [42]。在生物学中，解景观框架已被用于描述细胞命运分化 [40, 62], ，并分析肿瘤微环境中的癌症进展动力学 [37]。相关技术还被应用于其他具有多个平衡态的复杂系统，包括二嵌段共聚物-均聚物模型 [39], 、由朗道–布拉佐夫斯基模型描述的柱状相–螺旋相界面系统 [41], 、空间分数阶相场模型 [49], 、多相流体流动系统 [64], ，以及粗糙表面上的液滴润湿和定向输运 [63]。通过构建揭示这些系统中鞍点与极小值点之间连通性的解景观，鞍点动力学方法能够系统地研究鞍点问题，并深入揭示其背后的物理机制。这些实例还凸显了对实用计算框架的需求，即以系统化和自动化的方式构建此类解景观。

尽管方法和应用方面取得了这些进展，在具体问题中实际构造解景观时，往往仍停留在手动组装鞍点之间各条连接的层面。在典型工作流程中，针对景观中的每条边（即两个临界点之间的每条直接连接），都必须分别设计并运行基于 HiSD 的搜索，为不同的运行指定临时的停止准则，并且在探索和完善景观时高度依赖个人对 HiSD 及其变体的经验。此外，HiSD 的不同变体（如 GHiSD、CHiSD、AHiSD 和数据驱动扩展）通常采用略有差异的记号、约定和实现细节，实践中很容易因此产生混淆和不一致。目前，仍缺乏用于构造解景观的整体自动化算法框架，也缺乏能够将这些方法整合到单一工具中的系统化、一体化且用户友好的软件实现。

为促进解景观的研究与应用，我们开发了 SaddleScape V1.0 软件包，其中实现了一系列用于构造解景观的 HiSD 算法。SaddleScape V1.0 的核心贡献是实现了一套系统化、分层且自动化的解景观构造流程。我们的软件包还提供了高效且用户友好的 HiSD 算法实现，使研究人员能够轻松探索和应用 HiSD 方法。具体而言，我们的软件包具有以下主要优势：

- **自动化解景观构建：**我们实现了一种系统的向下搜索流程，该流程基于广度优先搜索（BFS），采用先进先出（FIFO）队列调度，并结合随机扰动。我们还通过重启接口支持从极小值点或较低指标的鞍点开始向上搜索，使用户能够完善或扩展已有的部分构建的解景观。

- **HiSD 变体与系统类型的统一框架：** 该软件包在同一接口中统一实现 HiSD 及其变体（例如用于非梯度系统的 GHiSD，以及采用动量加速的 AHiSD）。它同时支持梯度系统和一般的非梯度自治系统，包括经过空间离散化后的泛函系统。

- **灵活高效的数值实现：** 求解过程中的大多数步骤均可由用户配置。该系统支持数值函数和符号表达式，支持自定义梯度及自动微分，并集成了无矩阵的海森矩阵/雅可比矩阵向量乘积、多种特征值求解器，以及多种步长和加速策略（包括 Barzilai–Borwein、重球法和 Nesterov 方法），以提高稳健性和效率。

- **集成式后处理与可视化：** 我们提供用于搜索轨迹可视化、解景观构建以及以多种格式保存数据的封装方法。这些后处理模块有助于快速、直观地了解搜索过程和最终得到的解景观，并为后续分析提供便利。

- **模块化架构与用户友好型接口：** 该软件包由多个模块组成，分别用于参数检查、符号与数值导数处理、海森矩阵/雅可比算子、特征值求解器、迭代鞍点动力学求解器以及解景观构建。用户可通过一组清晰的选项配置系统定义、求解器参数、特征值方法和加速方案。许多参数均提供默认值，降低了新用户的使用门槛，同时也允许高级用户微调数值细节，并通过替换相应模块来尝试扩展算法。

总之，该软件包简化了鞍点搜索过程，提供灵活的参数设置，以及多种可视化和数据处理工具，从而降低了实际构建和分析解景观的门槛。

为便于阅读，常用符号的定义列于表1中。

2 高指标鞍点动力学

本节简要介绍一系列 HiSD 算法。详情请参阅我们的网站 https://hisdpackage.github.io/saddlescape 或相关参考文献。这些算法构成了此软件包的理论基础。本节内容分为三部分：

- 适用于能量函数二阶 Fr´échet 可微的梯度系统的基本 HiSD[47] 算法；

- 推广至非梯度系统，得到 GHiSD[46] 算法；

- 动量加速 AHiSD[19]算法。

2.1 梯度系统的 HiSD

2.1.1 系统描述

给定定义在带有内积 $\langle\cdot,\ \cdot\rangle$ 的实希尔伯特空间 $\mathcal{H}$上的二次 Fr´echet 可微能量泛函 $E(\boldsymbol{x})$，自然力定义为 $\boldsymbol{F}(\boldsymbol{x}) = -\nabla E(\boldsymbol{x})$，海森矩阵定义为$\mathbb{H}(\boldsymbol{x}) = \nabla^2 E(\boldsymbol{x})$。

- 若点 $\hat{\boldsymbol{x}} \in \mathcal{H}$ 满足 $\|\boldsymbol{F}(\hat{\boldsymbol{x}})\| = 0$，则称其为 $E(\boldsymbol{x})$ 的临界点。若临界点的海森矩阵 $\mathbb{H}(\hat{\boldsymbol{x}})$ 可逆且其逆有界，则称该临界点为非退化临界点。

- 非局部极值的临界点称为鞍点。按照惯例，局部极小值是指标-0 鞍点，局部极大值是指标-d 鞍点，其中 d 表示系统维数。

- 对于非退化临界点 $\hat{\boldsymbol{x}}$，其莫尔斯指标是满足以下条件的子空间$\mathcal{K} \subset \mathcal{H}$ 的最大维数：海森矩阵 $\mathbb{H}(\hat{\boldsymbol{x}})$ 在 $\mathcal{K}$ 上负定。目标是在势能面（PES）上寻找指标-k 鞍点（k-鞍点）。

表1：　　HiSD 框架中使用的数学符号和关键定义汇总　　.

符号	描述
$\boldsymbol{a}$	粗体表示向量。
$\mathbb{A}$	黑板粗体表示标准代数对象（例如矩阵或数域）。特别地，$\mathbb{R}$ 和 $\mathbb{C}$ 表示实数域和复数数域。
$\mathcal{A}$	花体字表示空间。
d	系统维数。
$\langle \cdot, \cdot \rangle$	希尔伯特空间中的内积 $\mathcal{H}$；在有限维情况下写作 $\boldsymbol{x}^\top \boldsymbol{y}$ 。
$E(\boldsymbol{x})$	定义在实希尔伯特空间 $\mathcal{H}$。
$\boldsymbol{F}(\boldsymbol{x})$	向量场 $\boldsymbol{F}(\boldsymbol{x})$；在梯度系统中，$\boldsymbol{F}(\boldsymbol{x}) = -\nabla E(\boldsymbol{x})$。
$\mathbb{H}(\boldsymbol{x})$	海森矩阵：$\mathbb{H}(\boldsymbol{x}) = \nabla^2 E(\boldsymbol{x})$。
$\mathbb{J}(\boldsymbol{x})$	雅可比矩阵：$\mathbb{J}(\boldsymbol{x}) = \nabla \boldsymbol{F}(\boldsymbol{x})$。在梯度系统中，$\boldsymbol{F}(\boldsymbol{x}) = -\nabla E(\boldsymbol{x})$，我们有 $\mathbb{J}(\boldsymbol{x}) = -\mathbb{H}(\boldsymbol{x})$。
$\hat{\boldsymbol{x}}$	临界点（梯度系统）或平衡点（非梯度系统）。
$\mathcal{W}^u(\hat{\boldsymbol{x}})$	由特征值对应正实部的特征向量张成的不稳定子空间为正实部。
$\mathcal{W}^s(\hat{\boldsymbol{x}})$	由对应于负实部特征值的特征向量张成的稳定子空间具有负实部。
$\mathcal{W}^c(\hat{\boldsymbol{x}})$	由对应于零实部特征值的特征向量张成的中心子空间至零实部。
k_u	不稳定子空间维数（非梯度系统）。
k_s	稳定子空间维数。
k_c	中心子空间维数。
k-鞍点	指标为 k 的鞍点（梯度系统中的莫尔斯指标 k，不稳定非梯度系统中的维度 $k_u = k$ ）。

- 为简便起见，我们假设 $\mathcal{H}$ 的维数为 d，并将内积 $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ 记作$\boldsymbol{x}^\top \boldsymbol{y}$。

等价地，鞍点的指标对应于其海森矩阵的负特征值个数。在实对称矩阵的合同标准形中，这个量也称为负惯性指标（负特征值的个数）。

2.1.2 指标-k HiSD 的推导

$\hat{}$ 对于非退化临界点 $\hat{}\boldsymbol{x}$，令 $\mathcal{V}$ 为由 $\mathbb{H}(\hat{}\boldsymbol{x})$ $\hat{}$对应负特征值的特征向量张成的子空间。由于 $\mathbb{H}(\hat{}\boldsymbol{x})$ 在 $\mathcal{V}$ 上负定，在 $\hat{}\mathcal{V}^\perp$上正定，因此 $\hat{}\boldsymbol{x}$ 是仿射空间 $\hat{}\boldsymbol{x} + \hat{}\mathcal{V}$中的局部极大值，也是 $\hat{}\boldsymbol{x} + \hat{}\mathcal{V}^\perp$中的局部极小值。

$\hat{}\boldsymbol{x}$的投影到 $\mathcal{V}$ 和 $\hat{}\mathcal{V}^\perp$分别记为 $\hat{}\boldsymbol{x}_{\hat{}\mathcal{V}}$和 $\hat{}\boldsymbol{x}_{\hat{}\mathcal{V}^\perp}$。数对 $(\boldsymbol{v}, \boldsymbol{w}) = (\hat{}\boldsymbol{x}_{\hat{}\mathcal{V}}, \hat{}\boldsymbol{x}_{\hat{}\mathcal{V}^\perp})$ 满足极小极大问题

$$\min_{\boldsymbol{w} \in \hat{\mathcal{V}}^\perp} \max_{\boldsymbol{v} \in \hat{\mathcal{V}}} E(\boldsymbol{v} + \boldsymbol{w}).$$

$\hat{}$由于 $\mathcal{V}$ 未知，因此得到一个迭代方案，同时更新当前迭代值 $\boldsymbol{x}$（临界点的一个估计值 $\hat{}$）以及对 $\mathcal{V}$ 的近似 $\mathcal{V}$。

$\boldsymbol{x}$ 的动力学

直观而言，$\boldsymbol{x}$ 的更新经过设计，使得 $\dot{\boldsymbol{x}}$ 在空间 $\mathcal{V}$, $\mathcal{P}_{\mathcal{V}}\dot{\boldsymbol{x}}$ 上的投影是能量函数 $E(\boldsymbol{x})$ 的上升方向，而在补空间 $\mathcal{V}^\perp$, $\mathcal{P}_{\mathcal{V}^\perp}\dot{\boldsymbol{x}}$ 上的投影则是下降方向。这里，$\mathcal{P}_{\mathcal{V}}$ 和 $\mathcal{P}_{\mathcal{V}^\perp}$ 分别表示到子空间 $\mathcal{V}$ 及其正交补 $\mathcal{V}^\perp$ 上的正交投影算子。

注意到自然力 $\boldsymbol{F}(\boldsymbol{x}) = -\nabla E(\boldsymbol{x})$ 是最速下降方向，因此可以设定

$$\mathcal{P}_{\mathcal{V}}\dot{\boldsymbol{x}} = -\mathcal{P}_{\mathcal{V}}\boldsymbol{F}(\boldsymbol{x}), \quad \mathcal{P}_{\mathcal{V}^\perp}\dot{\boldsymbol{x}} = \mathcal{P}_{\mathcal{V}^\perp}\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x}) - \mathcal{P}_{\mathcal{V}}\boldsymbol{F}(\boldsymbol{x}).$$

然后，给定两个正松弛参数 $\beta_{\mathcal{V}}$ 和 $\beta_{\mathcal{V}^\perp}$，$\boldsymbol{x}$ 的动力学可表示为

$$\dot{\boldsymbol{x}} = \beta_{\mathcal{V}}(-\mathcal{P}_{\mathcal{V}}\boldsymbol{F}(\boldsymbol{x})) + \beta_{\mathcal{V}^\perp}(\boldsymbol{F}(\boldsymbol{x}) - \mathcal{P}_{\mathcal{V}}\boldsymbol{F}(\boldsymbol{x})).$$

为简单起见，令 $\beta_{\mathcal{V}} = \beta_{\mathcal{V}^\perp} = \beta > 0$，并取 $\mathcal{V}$ 的标准正交基 $\{\boldsymbol{v}_i\}_{i=1}^k$，其中每个 $\boldsymbol{v}_i$均视为列向量，上述方程可简化为

$$\beta^{-1}\dot{\boldsymbol{x}} = \boldsymbol{F}(\boldsymbol{x}) - 2\mathcal{P}_{\mathcal{V}}\boldsymbol{F}(\boldsymbol{x}) = \left(\mathbb{I} - 2\sum_{i=1}^{k} \boldsymbol{v}_i \boldsymbol{v}_i^\top\right) \boldsymbol{F}(\boldsymbol{x}). \tag{2.1}$$

子空间 $\mathcal{V}$的更新

为了将近似定义为由 $\mathbb{H}(\boldsymbol{x})$的 k 个最小特征值所对应的特征向量张成的空间。该子空间通过同时瑞利商迭代最小化（SIRQIT）方法计算。考虑同时求解以下 k 约束优化问题（$i = 1,2,\ldots,k$）：

$$\min_{\boldsymbol{v}_i} \langle \boldsymbol{v}_i, \mathbb{H}(\boldsymbol{x})\boldsymbol{v}_i \rangle, \qquad \text{s.t.} \quad \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \delta_{ij} \quad j = 1, 2, \ldots, i, \tag{2.2}$$

其中

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

根据拉格朗日乘子法，将瑞利商的负梯度作为驱动项，并通过拉格朗日乘子施加正交归一约束，可得带有拉格朗日乘子 $\xi_j^{(i)}$ 和松弛参数 $\gamma > 0$ 的 $\dot{\boldsymbol{v}}_i$ 演化方程：

$$\gamma^{-1}\dot{\boldsymbol{v}}_i = -\mathbb{H}(\boldsymbol{x})\boldsymbol{v}_i + \sum_{j=1}^{i} \xi_j^{(i)} \boldsymbol{v}_j. \tag{2.3}$$

由于约束条件 $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \delta_{ij}$ 始终成立，因此有

$$\langle \dot{\boldsymbol{v}}_i, \boldsymbol{v}_j \rangle + \langle \boldsymbol{v}_i, \dot{\boldsymbol{v}}_j \rangle = 0, \quad i, j = 1, \ldots, k. \tag{2.4}$$

将式 (2.3) 代入式 (2.4)，并依次求解 $\dot{\boldsymbol{v}}_i$ 中针对$i = 1$，2、.....、 k的拉格朗日乘子，我们得到

$$\xi_i^{(i)} = \langle \boldsymbol{v}_i, \mathbb{H}(\boldsymbol{x})\boldsymbol{v}_i \rangle, \quad \xi_j^{(i)} = 2\langle \boldsymbol{v}_j, \mathbb{H}(\boldsymbol{x})\boldsymbol{v}_i \rangle \quad j = 1, 2, \ldots, i-1.$$

代回式 (2.3)，即可得到 $\boldsymbol{v}_i$的最终动力学：

$$\gamma^{-1}\dot{\boldsymbol{v}}_i = -\left(\mathbb{I} - \boldsymbol{v}_i\boldsymbol{v}_i^\top - 2\sum_{j=1}^{i-1} \boldsymbol{v}_j\boldsymbol{v}_j^\top \right) \mathbb{H}(\boldsymbol{x})\boldsymbol{v}_i, \quad i = 1, \ldots, k. \tag{2.5}$$

整体动力系统

完整的 HiSD 动力系统由式 (2.1) 和式 (2.5) 组合而成：

$$\begin{cases} \beta^{-1}\dot{\boldsymbol{x}} = \left(\mathbb{I} - 2\sum_{i=1}^{k} \boldsymbol{v}_i\boldsymbol{v}_i^\top \right) \boldsymbol{F}(\boldsymbol{x}), \\ \gamma^{-1}\dot{\boldsymbol{v}}_i = -\left(\mathbb{I} - \boldsymbol{v}_i\boldsymbol{v}_i^\top - 2\sum_{j=1}^{i-1} \boldsymbol{v}_j\boldsymbol{v}_j^\top \right) \mathbb{H}(\boldsymbol{x})\boldsymbol{v}_i, \quad i = 1, \ldots, k. \end{cases} \tag{2.6}$$

该框架通过动态更新临界点估计及其对应的子空间来定位鞍点。这些动力学的线性稳定性分析及更多细节见[47]。

2.1.3 HiSD 方法的离散化

采用近似海森矩阵的 SIRQIT

当精确海森矩阵可用时，可以直接对式 (2.6) 离散化以进行数值求解。然而，对于许多问题，计算海森矩阵并不可行，或代价过高，因此需要采用数值近似方法。值得注意的是，式 (2.6) 只需要海森矩阵-向量乘积 $\mathbb{H}(\boldsymbol{x})\boldsymbol{v}_i$，可通过有限差分高效计算。

海森矩阵-向量乘积 $\mathbb{H}(\boldsymbol{x})\boldsymbol{v}$ 可以利用泰勒展开近似：

$$-\boldsymbol{F}(\boldsymbol{x} \pm l\boldsymbol{v}) = \nabla E(\boldsymbol{x} \pm l\boldsymbol{v}) = \nabla E(\boldsymbol{x}) \pm \mathbb{H}(\boldsymbol{x})l\boldsymbol{v} + \mathcal{O}(\|l\boldsymbol{v}\|^2).$$

结合这两个展开式，可得如下近似（也称为二聚体方法）：

$$\boldsymbol{H}(\boldsymbol{x}, \boldsymbol{v}, l) = -\frac{\boldsymbol{F}(\boldsymbol{x} + l\boldsymbol{v}) - \boldsymbol{F}(\boldsymbol{x} - l\boldsymbol{v})}{2l} \approx \mathbb{H}(\boldsymbol{x})\boldsymbol{v}. \tag{2.7}$$

当 $l \to 0$ 时，理论上可以得到精确梯度；但在实际数值实验中，应避免使用过小的 l 值，以防止数值不稳定性。考虑这一点，可得到具有近似海森矩阵的完整系统：

$$\begin{cases} \beta^{-1}\dot{\boldsymbol{x}} = \left(\mathbb{I} - 2\sum_{i=1}^{k} \boldsymbol{v}_i\boldsymbol{v}_i^\top \right) \boldsymbol{F}(\boldsymbol{x}), \\ \gamma^{-1}\dot{\boldsymbol{v}}_i = -\left(\mathbb{I} - \boldsymbol{v}_i\boldsymbol{v}_i^\top - 2\sum_{j=1}^{i-1} \boldsymbol{v}_j\boldsymbol{v}_j^\top \right) \boldsymbol{H}(\boldsymbol{x}, \boldsymbol{v}_i, l) \quad i = 1, \ldots, k. \end{cases} \tag{2.8}$$

这里， l 取为固定常数，其选取可同时考虑目标精度和数值稳定性。

对上述动力学进行直接离散化可得到以下算法：

算法 1：使用近似海森矩阵求解 k-鞍点的 HiSD。
输入： $k \in \mathbb{N}$, $l > 0$, $\boldsymbol{x}^{(0)} \in \mathcal{H}$, $\{\boldsymbol{v}_i^{(0)}\}_{i=1}^k \subset \mathcal{H}$ 满足 $\left\langle \boldsymbol{v}_j^{(0)}, \boldsymbol{v}_i^{(0)} \right\rangle = \delta_{ij}$。

1: Set $n = 0$, compute $\boldsymbol{f}^{(0)} = \boldsymbol{F}(\boldsymbol{x}^{(0)})$;
2: **repeat**
3: $\quad \boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \beta^{(n)} \boldsymbol{g}^{(n)}$;
4: $\quad$ **for** $i = 1 : k$ **do**
5: $\quad\quad \boldsymbol{v}_i^* = \boldsymbol{v}_i^{(n)} + \gamma_i^{(n)} \boldsymbol{d}_i^{(n)}$;
6: $\quad\quad \boldsymbol{v}_i^* = \boldsymbol{v}_i^* - \sum_{j=1}^{i-1} \left\langle \boldsymbol{v}_j^{(n+1)}, \boldsymbol{v}_i^* \right\rangle \boldsymbol{v}_j^{(n+1)}$;
7: $\quad\quad \boldsymbol{v}_i^{(n+1)} = \boldsymbol{v}_i^* / \|\boldsymbol{v}_i^*\|$;
8: $\quad$ **end for**
9: $\quad \boldsymbol{f}^{(n+1)} = \boldsymbol{F}(\boldsymbol{x}^{(n+1)})$;
10: $\quad n := n + 1$;
11: **until** $\|\boldsymbol{f}^{(n)}\| < \varepsilon_{\boldsymbol{F}}$
12: **Output :** $\boldsymbol{x}^{(n)}, \boldsymbol{v}_1^{(n)}, \ldots, \boldsymbol{v}_k^{(n)}$.

其中

$$\boldsymbol{g}^{(n)} = \boldsymbol{f}^{(n)} - 2 \sum_{i=1}^{k} \left\langle \boldsymbol{v}_i^{(n)}, \boldsymbol{f}^{(n)} \right\rangle \boldsymbol{v}_i^{(n)},$$

$$\boldsymbol{u}_i^{(n)} = \boldsymbol{H}\left(\boldsymbol{x}^{(n+1)}, \boldsymbol{v}_i^{(n)}, l\right),$$

$$\boldsymbol{d}_i^{(n)} = -\boldsymbol{u}_i^{(n)} + \left\langle \boldsymbol{v}_i^{(n)}, \boldsymbol{u}_i^{(n)} \right\rangle \boldsymbol{v}_i^{(n)} + \sum_{j=1}^{i-1} 2 \left\langle \boldsymbol{v}_j^{(n)}, \boldsymbol{u}_i^{(n)} \right\rangle \boldsymbol{v}_j^{(n)}.$$

步长 $\beta^{(n)}$ 和 $\gamma_i^{(n)}$ 的选取方法将在第 2.1.4 节讨论。此外，尽管连续动力学能够保持 $\boldsymbol{v}_i$ 的正交归一性，但由于采用有限时间步长进行离散化，建议在上述算法的第 6 行和第 7 行应用格拉姆–施密特正交化，以满足正交归一约束。实际更新 $\boldsymbol{v}_i^{(n)}$ 时，可以减小时间步长并增加子步数；也就是说，以步长 $\gamma_i^{(n)}$ 将算法的第 4–8 行重复执行 N 次

$\overline{N}$，从而更精细地更新特征子空间。

结合 LOBPCG 方法

或者，可以使用成熟的特征值求解器（例如局部最优块预条件共轭梯度（LOBPCG）方法）来更新 $\mathcal{V}$。该方法能够高效计算最小的 k 个特征向量，并在精度与效率之间取得良好平衡。

该算法的核心思想是在一个子空间上近似求解瑞利商优化问题 (2.2)，并逐步扩充该子空间以改进近似，最终收敛到真实的 k 最小特征向量。通过这一子空间降维，原问题被转化为一个维度更低、更易处理的特征值问题。

有关实现细节，请参阅 [47]。这里仅指出，在 LOBPCG 的数值实现中，海森矩阵与矩阵的乘法可以分解为多个海森矩阵-向量乘积。随后可利用式 (2.7) 对这些乘积进行近似，以提高计算效率。

2.1.4 迭代步长的选择

简而言之，选择步长的常用方法如下：

显式欧拉法

将 $\beta^{(n)} = \beta \Delta t, \gamma_i^{(n)} = \gamma \Delta t$ 设为固定常数，其中 $\beta, \gamma, \Delta t > 0$。这里，$\Delta t$ 不应过小，以免收敛缓慢；也不应过大，以防算法发散。

BB梯度法

Fo对于 $\beta^{(n)}$的选择，我们采用以下 Barzilai–Borwein (BB) 步长[2] :

令 $\Delta \boldsymbol{x}^{(n)} = \boldsymbol{x}^{(n)} - \boldsymbol{x}^{(n-1)}$ 和 $\Delta \boldsymbol{g}^{(n)} = \boldsymbol{g}^{(n)} - \boldsymbol{g}^{(n-1)}$。根据拟牛顿框架，我们最小化以下范数

$$\min_{\beta^{(n)}} \|\Delta \boldsymbol{x}^{(n)}/\beta^{(n)} - \boldsymbol{g}^{(n)}\|, \quad \min_{\beta^{(n)}} \|\Delta \boldsymbol{x}^{(n)} - \beta^{(n)} \boldsymbol{g}^{(n)}\|.$$

以求得 BB1 和 BB2 步长：

$$\beta^{(n)}_{\mathrm{BB1}} = \frac{\langle \Delta \boldsymbol{x}^{(n)}, \Delta \boldsymbol{x}^{(n)} \rangle}{\langle \Delta \boldsymbol{x}^{(n)}, \Delta \boldsymbol{g}^{(n)} \rangle}, \quad \beta^{(n)}_{\mathrm{BB2}} = \frac{\langle \Delta \boldsymbol{x}^{(n)}, \Delta \boldsymbol{g}^{(n)} \rangle}{\langle \Delta \boldsymbol{g}^{(n)}, \Delta \boldsymbol{g}^{(n)} \rangle}.$$

在此情况下，BB1步长的分母可能变得非常小，甚至为零，从而导致数值不稳定性；因此，我们采用BB2步长。此外，我们可以为 $\beta^{(n)}\|\boldsymbol{g}^{(n)}\|$设置上界$\tau$ ，并取 $\beta^{(n)}$ BB2 的绝对值，以避免出现负步长。结合这些约束条件可得：

$$\beta^{(n)} = \min \left\{ \frac{\tau}{\|\boldsymbol{g}^{(n)}\|}, \left| \frac{\langle \Delta \boldsymbol{x}^{(n)}, \Delta \boldsymbol{g}^{(n)} \rangle}{\langle \Delta \boldsymbol{g}^{(n)}, \Delta \boldsymbol{g}^{(n)} \rangle} \right| \right\}. \tag{2.9}$$

2.2 非梯度系统的 GHiSD

2.2.1 非梯度自治动力系统中的 k-鞍点描述

HiSD算法作用于二次 Fr´echet 可微的能量泛函 $E(\boldsymbol{x})$。相比之下，GHiSD算法针对一般 d 维非梯度自治动力系统中的指标为 k 的鞍点：

$$\dot{\boldsymbol{x}} = \boldsymbol{F}(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d, \quad \boldsymbol{F} \in \mathcal{C}^r(\mathbb{R}^d, \mathbb{R}^d), \quad r \geq 2. \tag{2.10}$$

我们首先定义非梯度系统中鞍点的指标。

在动力系统的语境下，将 $\boldsymbol{F}(\boldsymbol{x})$ 的雅可比矩阵记为 $\mathbb{J}(\boldsymbol{x}) = \nabla \boldsymbol{F}(\boldsymbol{x})$，内积记为 $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\top \boldsymbol{y}$。满足 $\boldsymbol{F}(\hat{\boldsymbol{x}}) = \boldsymbol{0}$ 的点 $\hat{\boldsymbol{x}}$ 称为平衡态（或驻点）。

线性化 在平衡态附近对系统进行线性化，令 $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{y}$ 并代入式（2.10），得到 ds

$$\dot{\boldsymbol{y}} = \mathbb{J}(\hat{\boldsymbol{x}})\boldsymbol{y} + \mathcal{O}(\|\boldsymbol{y}\|^2).$$

忽略高阶项后，得到线性化系统：

$$\dot{\boldsymbol{y}} = \mathbb{J}(\hat{\boldsymbol{x}})\boldsymbol{y}.$$

我们根据 $\mathbb{J}(\hat{\boldsymbol{x}})$ 的右（广义）特征向量对应的特征值实部的符号进行分类，由此定义三个关键子空间：

- **不稳定子空间 $\mathcal{W}^u(\hat{\boldsymbol{x}})$：** 由对应特征值具有正实部的特征向量张成，其维数记为 k_u。沿这些方向的扰动会指数增长，使系统偏离平衡态。

- **稳定子空间 $\mathcal{W}^s(\hat{\boldsymbol{x}})$：** 由对应特征值实部为负的特征向量张成，其维数记为 k_s。此处的扰动会指数衰减并趋近平衡态。

- **中心子空间 $\mathcal{W}^c(\hat{\boldsymbol{x}})$：** 由零实部特征值对应的特征向量张成，其维数记为 k_c。扰动表现出中性动力学，例如振荡，不会发生指数增长或衰减。

根据 $\mathbb{J}(\hat{\boldsymbol{x}})$的谱分解，$\mathbb{R}^d = \mathcal{W}^u \oplus \mathcal{W}^s \oplus \mathcal{W}^c$，其中 $k_u + k_s + k_c = d$。当所有特征值的实部均非零时（$\mathbb{R}^d = \mathcal{W}^u \oplus \mathcal{W}^s$），该平衡态是双曲型的，这也是我们重点研究的对象。

在梯度系统中，其中 $\boldsymbol{F} = -\nabla E$，关系 $\mathbb{J} = -\mathbb{H}$ 将海森矩阵与雅可比矩阵联系起来。平衡态的指标 k_u 定义为其不稳定子空间的维数，是梯度系统中莫尔斯指标概念的推广。因此，我们将指标为-k_u 的平衡态称为指标为-k的鞍点，而汇点（$k_u = 0$）和源点（$k_u = d$）则是特殊情况。后续章节将详细介绍在非梯度系统中搜索这些- k鞍点的方法。

2.2.2 指标为-k 的 GHiSD 推导

$\boldsymbol{x}$的动力学

$\boldsymbol{x}$动力学的构造原理与 HiSD 类似。在平衡点附近，沿系统力的方向运动会使轨迹沿稳定子空间收敛至平衡点，并沿不稳定子空间远离平衡点。因此，我们在稳定子空间中保持力的方向，在不稳定子空间中反转其方向。由于 k鞍点处的精确不稳定子空间$\mathcal{W}^{\mathrm{u}}(\hat{\boldsymbol{x}})$未知，与 HiSD 类似，我们定义 $\mathcal{V}$，以近似表示 $\mathcal{W}^{\mathrm{u}}(\hat{\boldsymbol{x}})$，即由与具有最大实部的 k个特征值相关的 $\mathbb{J}(\boldsymbol{x})$ 的 k （广义）特征向量张成的实不变子空间。如果这些特征值（及特征向量）是复数，则将 $\mathcal{V}$ 取为由相应复广义特征向量的实部和虚部张成的实空间。由此得到

$$\beta^{-1}\dot{\boldsymbol{x}} = \left(\mathbb{I} - 2\mathcal{P}_{\mathcal{V}}\right)\boldsymbol{F}(\boldsymbol{x}) = \left(\mathbb{I} - 2\sum_{j=1}^{k}\boldsymbol{v}_j\boldsymbol{v}_j^{\top}\right)\boldsymbol{F}(\boldsymbol{x}). \tag{2.11}$$

这与 (2.1) 形式相同，其中 $\{\boldsymbol{v}_i\}_{i=1}^k$ 是 $\mathcal{V}$的标准正交基，$\beta > 0$ 是松弛参数。

子空间 $\mathcal{V}$的更新

虽然高效的瑞利商优化不适用于非对称矩阵，但请注意，动力学 $\dot{\boldsymbol{v}} = \mathbb{J}\boldsymbol{v}$ 自然与 $\boldsymbol{v}$ 对齐，该不变子空间对应于 $\mathbb{J}$ 实部最大的特征值。因此，可以类似地提出一个系数 $\xi_j^{(i)}$ 和松弛常数 $\gamma > 0$未定的动力学方程：

$$\gamma^{-1}\dot{\boldsymbol{v}}_i = \mathbb{J}(\boldsymbol{x})\boldsymbol{v}_i + \sum_{j=1}^{i}\xi_j^{(i)}\boldsymbol{v}_j.$$

与 HiSD 类似，利用正交归一约束 (2.4)，可得

$$\xi_i^{(i)} = -\langle\mathbb{J}(\boldsymbol{x})\boldsymbol{v}_i, \boldsymbol{v}_i\rangle, \quad \xi_j^{(i)} = -\langle\mathbb{J}(\boldsymbol{x})\boldsymbol{v}_i, \boldsymbol{v}_j\rangle - \langle\boldsymbol{v}_i, \mathbb{J}(\boldsymbol{x})\boldsymbol{v}_j\rangle, \quad j = 1, \cdots, i-1.$$

将这些结果与式 (2.11) 结合，得到完整的动力系统：

$$\begin{cases} \beta^{-1}\dot{\boldsymbol{x}} = \left(\mathbb{I} - 2\sum_{j=1}^{k}\boldsymbol{v}_j\boldsymbol{v}_j^{\top}\right)\boldsymbol{F}(\boldsymbol{x}), \\ \gamma^{-1}\dot{\boldsymbol{v}}_i = \left(\mathbb{I} - \boldsymbol{v}_i\boldsymbol{v}_i^{\top}\right)\mathbb{J}(\boldsymbol{x})\boldsymbol{v}_i - \sum_{j=1}^{i-1}\boldsymbol{v}_j\boldsymbol{v}_j^{\top}\left(\mathbb{J}(\boldsymbol{x}) + \mathbb{J}^{\top}(\boldsymbol{x})\right)\boldsymbol{v}_i, \quad i = 1, \cdots, k. \end{cases} \tag{2.12}$$

该系统的线性稳定性将在 [46]中讨论。

2.2.3 GHiSD 的一种简化离散形式

对于 GHiSD方法的算法实现，与 HiSD算法类似，由于时间步长导致的

离散化误差，对（2.12）进行显式欧拉离散化需要额外执行强制正交归一化操作。后续研究 [60] 表明，（2.12）的显式欧拉离散化等价于以下更简洁的离散化方法；由于效率更高，该方法在实际计算中更受青睐：

$$\begin{cases} \tilde{\boldsymbol{v}}_i^{(n+1)} = \boldsymbol{v}_i^{(n)} + \gamma \mathbb{J}(\boldsymbol{x}) \boldsymbol{v}_i^{(n)} \quad i = 1, \ldots, k, \\ \left[\boldsymbol{v}_1^{(n+1)}, \ldots, \boldsymbol{v}_k^{(n+1)} \right] = \operatorname{orth} \left(\tilde{\boldsymbol{v}}_1^{(n+1)}, \ldots, \tilde{\boldsymbol{v}}_k^{(n+1)} \right), \end{cases} \tag{2.13}$$

其中，orth(·) 表示格拉姆–施密特正交化。

与 HiSD 类似，我们采用二聚体方法（2.7）来近似 $\mathbb{J}(\boldsymbol{x})\boldsymbol{v}_i^{(n)}$ as

$$\mathbb{J}(\boldsymbol{x}) \boldsymbol{v}_i^{(n)} \approx \frac{\boldsymbol{F}(\boldsymbol{x} + l\boldsymbol{v}_i^{(n)}) - \boldsymbol{F}(\boldsymbol{x} - l\boldsymbol{v}_i^{(n)})}{2l}.$$

组合该方法与离散化的 $\boldsymbol{x}$-动力学，可得到完整的离散化方法：

$$\begin{cases} \boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \beta \left(\boldsymbol{F}(\boldsymbol{x}^{(n)}) - 2 \sum_{j=1}^{k} \left\langle \boldsymbol{F}(\boldsymbol{x}^{(n)}), \boldsymbol{v}_j^{(n)} \right\rangle \boldsymbol{v}_j^{(n)} \right), \\ \tilde{\boldsymbol{v}}_i^{(n+1)} = \boldsymbol{v}_i^{(n)} + \gamma \dfrac{\boldsymbol{F}(\boldsymbol{x}^{(n+1)} + l\boldsymbol{v}_i^{(n)}) - \boldsymbol{F}(\boldsymbol{x}^{(n+1)} - l\boldsymbol{v}_i^{(n)})}{2l}, \quad i = 1, \ldots, k, \\ \left[\boldsymbol{v}_1^{(n+1)}, \ldots, \boldsymbol{v}_k^{(n+1)} \right] = \operatorname{orth} \left(\tilde{\boldsymbol{v}}_1^{(n+1)}, \ldots, \tilde{\boldsymbol{v}}_k^{(n+1)} \right). \end{cases} \tag{2.14}$$

2.3 AHiSD

本小节介绍 HiSD 的两种动量加速变体。第一种变体采用重球动量[19]:

$$\begin{cases} \boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \beta^{(n)} \left(\mathbb{I} - 2 \sum_{i=1}^{k} \boldsymbol{v}_i^{(n)} \boldsymbol{v}_i^{(n)\top} \right) \boldsymbol{F}(\boldsymbol{x}^{(n)}) + \alpha(\boldsymbol{x}^{(n)} - \boldsymbol{x}^{(n-1)}), \\ \left\{ \boldsymbol{v}_i^{(n+1)} \right\}_{i=1}^{k} = \operatorname{EigenSol} \left(\mathbb{H}(\boldsymbol{x}^{(n+1)}), \left\{ \boldsymbol{v}_i^{(n)} \right\}_{i=1}^{k} \right). \end{cases} \tag{2.15}$$

利用动量加速所产生的外推迭代方向，我们得到 Nesterov 加速变体 [17]:

$$\begin{cases} \boldsymbol{w}^{(n)} = \boldsymbol{x}^{(n)} + \alpha^{(n)}(\boldsymbol{x}^{(n)} - \boldsymbol{x}^{(n-1)}), \\ \boldsymbol{x}^{(n+1)} = \boldsymbol{w}^{(n)} + \beta^{(n)} \left(\mathbb{I} - 2 \sum_{i=1}^{k} \boldsymbol{v}_i^{(n)} \boldsymbol{v}_i^{(n)\top} \right) \boldsymbol{F}(\boldsymbol{w}^{(n)}), \\ \left\{ \boldsymbol{v}_i^{(n+1)} \right\}_{i=1}^{k} = \operatorname{EigenSol} \left(\mathbb{H}(\boldsymbol{x}^{(n+1)}), \left\{ \boldsymbol{v}_i^{(n)} \right\}_{i=1}^{k} \right). \end{cases} \tag{2.16}$$

这些方法的主要特点包括:

- **特征值求解器：** $\operatorname{EigenSol}\left(\mathbb{H}(\boldsymbol{x}^{(n+1)}), \{\boldsymbol{v}_i^{(n)}\}_{i=1}^{k}\right)$ 表示特征值求解器，它可以实现 HiSD 动力学的直接离散化，也可以采用 LOBPCG 等算法。

- **步长：** 步长 $\beta^{(n)}$ 采用与 HiSD 相同的选择准则，包括欧拉方法或 BB步长方法。

- **动量：** 动量项 $\alpha(\boldsymbol{x}^{(n)} - \boldsymbol{x}^{(n-1)})$ 构成了核心加速机制。对于重球加速，$\alpha \in [0, 1)$ 通常为常数。在 Nesterov 加速中，$\alpha^{(n)}$有多种形式，例如

$$\alpha^{(n)} = \frac{n}{n+3}, \tag{2.17}$$

or

$$\alpha^{(n)} = \theta_{n+1}^{-1}(\theta_n - 1) \quad \text{with} \quad \theta_{n+1} = \frac{1 + \sqrt{1 + 4\theta_n^2}}{2}, \ \theta_0 = 1. \tag{2.18}$$

数值实验表明，这些选择在性能上的差异可以忽略不计；因此，我们倾向于采用更简单的第一种表述。

3 解景观的构造

利用上述 HiSD算法套件准确定位单个指标为k 的鞍点，为实现一个更具雄心的目标奠定了基础：绘制解空间的全局拓扑。本节详细介绍将离散鞍点转化为路径图的算法框架，该路径图由所有驻点及其相互连接构成，称为**解景观**。

Co从概念上讲，我们将解景观建模为一个有向图 G =(V, E)，其中 e:

- 顶点（V）是系统的鞍点，每个顶点都由其坐标、莫尔斯指标及其他性质（例如其不稳定流形）来刻画。
- 边（E）表示这些顶点之间的层级连接。从指标为 k 的鞍点指向指标为 m 的鞍点的一条边（其中 $m < k$）意味着后一个鞍点可以从前一个鞍点的不稳定流形到达。

解景观的构造涉及两种主要操作：向下搜索和向上搜索。它们统一于 GHiSD 框架之下，其中梯度系统包含于 $\boldsymbol{F}(\boldsymbol{x}) = -\nabla E(\boldsymbol{x})$。

3.1 向下搜索：遍历层级结构

向下搜索是揭示解景观的基本机制。该搜索从指标-k 鞍点出发，定位指标-m 鞍点（$m < k$），并最终到达指标-0鞍点（汇点），同时记录层级连接。此过程直接应对景观探索中的核心挑战：如何可靠地离开鞍点的局部区域，并转向与之相连、指标更低的临界点。具体方法如下：

1. **初始扰动：** 从给定的指标-k 鞍点出发，施加一个幅度为 $\mathcal{O}$（ε）的小扰动。为确保逃逸方向不带偏向，首先在全空间中生成一个小的随机扰动。然后将该向量投影到不稳定子空间 $\mathcal{W}^u$ ，并缩放至$\mathcal{O}$（ε），以提取驱动系统远离鞍点的分量。

2. **不稳定子空间初始化：** 从初始鞍点的 k 个不稳定方向中选取 m 个方向，构成 $\mathcal{V}_m^{(0)}$。

3. **HiSD算法：** 随后应用 HiSD 动力学，由$\dot{\boldsymbol{x}} = \left(\mathbb{I} - 2\mathcal{P}_{\mathcal{V}_m(\boldsymbol{x})}\right)\boldsymbol{F}$（$\boldsymbol{x}$）控制，确保沿不稳定方向的力投影 $\boldsymbol{v}_i$ 保持为正。根据 HiSD算法理论，这会驱动系统远离初始鞍点，并趋向与之相连、指标更低的鞍点。

为确保进行全面探索，该实现加入了多项增强措施，例如从多种统计分布（如均匀分布、高斯分布）中采样扰动，并系统地探索初始子空间的所有组合可能性（k选 m）。对于汇搜索（$m = 0$），算法则进行了简洁优雅的简化。由于子空间 $\mathcal{V}^{(0)}$ 为空，HiSD 动力学自然退化为标准的梯度下降动力学，从而避免了计算开销较大的海森矩阵和特征向量计算。

3.2 向上搜索：从吸引域出发进行探索

向下搜索构建了主要的层级结构，而向上搜索则提供了一种互补的策略来探索解景观，使我们能够从低指标鞍点发现高指标鞍点。这对于从已发现的极小值点启动新的搜索或扩展不完整景观尤其有用。其原理与向下搜索对称：

1. **初始扰动：** 从初始指标为 k 的鞍点出发，施加一个幅度为 $\mathcal{O}$(ε)的小扰动。重新启动时，用户可指定此扰动。

2. **子空间初始化：** 将 k 个不稳定方向与稳定子空间中的 $m-k$ 个方向组合，构成 $\mathcal{V}_m^{(0)}$（通常选择对应于最小正特征值的方向）。

3. **HiSD算法：** 随后的 HiSD 动力学沿稳定方向 $\boldsymbol{v}_j$产生负投影，使系统得以逃逸。

在实际实现中，整体策略首先使用 HiSD 定位一个初始高指标鞍点，随后以此为起点进行全面的向下搜索，从而构造解景观。向上搜索也是重要工具，可从新发现的区域重新开始探索，以确保完备性。

3.3 解景观构造的算法框架

在构造完整解景观时，除了逃逸问题之外，还面临一些关键挑战。

- **识别问题：** 搜索过程常会从不同路径收敛到同一个鞍点。我们需要一种稳健而高效的方法，判断新发现的临界点究竟是新点，还是列表中已有临界点的重复项。

- **完备性问题：** 搜索必须足够系统且全面，以发现几乎所有重要关联，并构建近乎完整的解景观表示，避免沿着单一路径过早终止。

为应对这些挑战，我们设计了一种基于广度优先搜索算法的全局搜索策略，并使用先进先出队列进行管理。我们选择广度优先搜索，而非深度优先搜索（DFS）等其他方法，原因至关重要：它逐层探索的方式与解景观的自然层级结构完美对应。深度优先搜索会迅速沿着一条路径搜索到某个极小值，可能因此错过其他重要分支；而广度优先搜索能够确保，在搜索从同一个初始鞍点进入指标-($k-2$)层之前，先找到与特定指标-k鞍点相连的所有指标-($k-1$)鞍点。这种有条不紊的方法对于确保搜索的系统性和完整性至关重要。

完整的自动化工作流程详述如下算法。

算法 2 通过 HiSD 搜索构造 SolutionLandscape。输入：初始点 $\boldsymbol{x}^{(0)}$，初始不稳定子空间 $\boldsymbol{E}^{(0)}$（可自动构造 ），最大莫尔斯指标 M，单个点的搜索深度 J1：扰动初始点，以避免陷入初始鞍点：$\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}^{(0)} + \boldsymbol{x}_{\epsilon}$2：初始化 HiSD 求解器 3：对于 $i = M$，递减至 0，执行 4：使用 $\boldsymbol{x}^{(0)}$ 和 $\boldsymbol{E}^{(0)}$启动 HiSD 搜索，寻找指标为 i 的鞍点 5：如果找到新的鞍点 $\boldsymbol{x}_{\mathbf{new}}$，则 6：将 $\boldsymbol{x}_{\mathbf{new}}$ 添加到鞍点列表，并初始化其连接数据 7：跳出 8：结束 if 9：结束 for 10：// 使用先进先出队列调度实现广度优先搜索 11：当队列非空时执行 12：出队队首元素 $\boldsymbol{x}_{\mathbf{head}}$（指标为 k） 13：对于 $i = k - 1$，递减至 max(0, $k - J$)，执行 14：生成对称扰动集 $\boldsymbol{P}$（均匀分布/高斯分布） 15：在 $i > 0$ 时生成初始子空间 $\boldsymbol{E}$（无需 $i = 0$）// 尝试$\binom{k}{i}$ 个不稳定方向的所有组合（如果参数为“all”），// 否则选择 i 个最不稳定方向（如果参数为“min”）。16：对于所有 $\boldsymbol{p} \in \boldsymbol{P}$、$\boldsymbol{E_i} \in \boldsymbol{E}$，执行 17：从 $\boldsymbol{x}_{\mathbf{head}} + \boldsymbol{p}$ 启动 HiSD 搜索，使用 $\boldsymbol{E}_i$ 18：如果找到新的鞍点 $\boldsymbol{x}_{\mathbf{new}}$，则 19：与已有点进行比较 20：如果 $\boldsymbol{x}_{\mathbf{new}}$ 是新点，则 21：将其加入队列并更新连接信息 22：否则 23：记录与 $\boldsymbol{x}_{\mathbf{head}}$的连接 24：结束 if 25：结束 if 26：结束 for 27：结束 for 28：结束 while 29：（可选）从新的初始点重复此过程 输出：鞍点列表（ID、坐标、莫尔斯指标、相互连接关系）

4 软件实现

4.1 概述

本节系统介绍软件包中各组件的功能，并展示核心代码的实现细节。我们首先概述 Python软件包的依赖项和输入参数规范。

4.1.1 依赖项

我们的软件包需要以下稳定版本的 Python 库：

- **标准库模块：** copy、sys、warni[illegible]

• **外部依赖项：** NumPy (v2.2.3) py (v1.15.3 nPy (v1.13.3 plotlib(v3 networkx (v3.4.2,

4.1.2 输入参数配置

本节概述了可用参数，这些参数使求解器具备灵活性和可定制性。详细参数列表请参阅附录 A。

• **系统参数：** 这些参数与系统设置及其一般属性有关，包括 **维度**、**能量函数**、**梯度**、**自动微分**、**数值梯度**、**二聚体长度**、**对称性检查**和**梯度系统**。

> **注意：** 为确保梯度系统（由 $-\nabla E$ 控制）与非梯度系统（由 $\boldsymbol{F}$ 控制）之间的公式形式统一，该软件包采用内部符号约定。具体而言，为非梯度系统提供输入时，用户必须输入向量场的负值$\boldsymbol{F}(\boldsymbol{x})$。这样可以确保 HiSD 和 GHiSD 的核心迭代机制不受系统类型影响，始终保持一致。

• **景观参数：** 这些参数与解景观的构建和探索有关，包括 **最大指标**、**最大指标间隔**、**相同性判定**、**初始特征向量**、**扰动方法**、**扰动半径**、**扰动数量**和**特征向量组合**。这些参数控制鞍点的分层搜索，包括指标限制、扰动策略和等价判据。

• **求解器参数：** 这些参数与求解器过程相关，用于控制 HiSD 算法的行为，包括**初始点**、**容差**、**搜索区域**、**时间步长**、**最大迭代次数**、**保存轨迹**、**详细输出**和**报告间隔**。它们定义了鞍点搜索的初始条件、收敛准则和输出设置。

• **特征参数：** 这些参数用于控制特征值对（特征值和特征向量）的计算，包括**特征值方法**、**特征值最大迭代次数**、**特征值步长**、**精度容差**和**特征向量统一**。它们定义了特征值求解器的选择、迭代上限、步长以及特征值分类的容差。

• **海森参数：** 这些参数与海森矩阵相关，包括**精确海森矩阵**和**海森二聚体长度**。它们控制海森矩阵的计算方法，例如解析推导，或通过基于二聚体的方法进行数值近似。

• **加速参数：** 这些参数与提升算法速度和效率相关，包括**BB步长**、**加速**、**Nesterov选择**、**Nesterov重启**和**动量**。它们支持自适应步长和基于动量的加速技术，从而提高收敛速度。

4.2 核心程序

该程序将软件包的所有功能封装在 `Landscape` 类中，该类主要包含三类函数：

• __ __类定义的核心函数，用于执行参数验证和初始化（参见第4.3节）。

• 使用鞍点动力学构建解景观的函数（参见第4.5节）：

– ：基本求解器。– 从点重启：从新的初始点重新开始计算的函数。– 从鞍点重启：从 重新开始计算的函数，并使用小幅随机扰动，以避免在原始鞍点附近陷入困境

- 用于可视化和保存的函数:

  – I 用于绘制解轨迹（参见第 4.6.1 节）。– DrawConnection: 通过绘制鞍点对间的连接来可视化解景观（参见第 4.6.2 节）。– Save: 用于保存原始计算数据（参见第 4.6.3 节）。

4.3 参数验证模块

4.3.1 参数验证程序

该程序主要通过 **景观参数**验证用户输入，确保其符合要求，所用函数为 LandscapeCheckParam。其核心功能包括:

- 检测到无效输入时立即报错。
- 为缺失的参数自动分配默认值 .

4.3.2 参数验证程序

其余参数通过调用函数 HiSDCheckParam 进行验证，该函数位于文件 HiSD .py 中，并在 Solver 逻辑和功能与 LandscapeCheckP 类似。

4.4 指标-k 鞍点搜索的架构与实现

求解器架构旨在适应附录A中详细介绍的各种配置。本节将详细说明指标-k 鞍点搜索程序如何满足这些不同需求，包括:

- 不同参数配置下算法分支的全面说明，包括功能规范和建议使用场景;
- 鞍点搜索算法核心的具体实现细节。具有指定配置的最终集成求解器封装在 Solver 类中:
- __ __ 执行参数验证和初始化（通过上述 HiSDCheckParam 函数）
- 运行: 执行指标为k 的鞍点搜索（目标搜索指标在求解器 初始化时指定）。

4.4.1 符号计算与自动微分 AnaCal.p y

该程序为能量/梯度输入实现了双接口设计，并支持自动符号-数值转换:

1. **直接使用可调用对象:** 当函数对象作为能量函数和梯度函数提供时，会直接使用它们而不进行额外处理（此情况下无需调用特定函数）。

2. **符号能量函数处理:**

   - I 将符号能量函数转换为可调用的数值函数，避免重复编译符号表达式，从而显著提高计算效率。结果存储在 instance.EnergyFunctionSolve。

- FunctionAnalysis 编译的函数计算特定点处的能量函数值。

3. **符号梯度处理：**

- 号能量函数执行自动微分，返回符号微分结果和可调用的数值梯度函数。符号结果存储在 `instance.GradFunction sympy`中，而数值函数则存储在 `instance.GradFunction numpy`。
- 提供的符号梯度函数编译为可调用的数值函数，其输出和存储方式与自动求导相同。
- 使用 `instance.GradFunction numpy`中预先存储的已编译梯度函数，计算特定点处的梯度。

4. **数值梯度计算：**

- **：** 仅提供数值能量函数时，使用中心差商计算梯度函数的数值近似。

4.4.2 HessianMatrix 处理

注意：对于非梯度系统，此处的“海森矩阵”指向量场函数的**负**雅可比矩阵。这是因为我们之前要求用户在输入的力前加上负号。

1. **将海森矩阵视为算子：**在 HiSD算法中，我们只需处理海森矩阵-向量乘积或海森矩阵结构，从而避免显式计算完整海森矩阵。

- ，通过中心有限差分近似计算海森矩阵-向量乘积 $\pm L\mathbf{v}$（实现公式 (2.7)）。
- 算子重载，使用无矩阵方法计算海森矩阵乘积。

2. **显式海森矩阵计算：**

- 号微分和对称性检查，判断系统是否具有梯度保持性质，从而确定后续要使用的 HiSD 算法。

- 矩阵，不进行对称性检查。
- 使用符号计算结果计算特定点处的海森矩阵。
- **：** 显式海森矩阵计算的混合方法：– 在指定时使用精确符号计算。– 否则，通过批量海森向量积 采用数值近似（海森矩阵乘以单位矩阵的各列，即对每个标准基向量应用该算子，以重构完整的海森矩阵）。

- **显式海森矩阵计算的适用场景：**
  – 迭代次数可能非常多时。– 频繁复用海森矩阵时。– 解的精度优先于计算时间时（此时仅使用符号显式计算）。

注：在高维情形下，由于计算成本高昂，仍不建议显式计算海森矩阵。

4.4.3 特征值求解器

本程序实现了多种用于计算 HiSD 算法特征值对的方法，支持梯度系统和非梯度系统。

1. **LOBPCG 方法：** （由于要求算子对称，因此仅适用于梯度系统）此方法使用 `scipy.sparse.linalg.lobpcg` 实现，对应第 2.1.3

- 使用无矩阵算子（海森矩阵-向量乘积和海森矩阵乘积）实现 LOBPCG。
- 当精确海森矩阵可用时，使用显式海森矩阵进行 LOBPCG 计算。

• 两个函数均返回：– 当前点处对应于最小 k 个特征值的特征向量。– 一个标志 `whetherkindex` ，用于指示该点是否满足指标k 条件（至少有 k 个负特征值）。由于输出特征值按升序排列，只需检查第 k 个特征值是否为负（在精度容差范围内）。

2. **幂法：** （适用于非梯度系统）该方法基于对理论动力系统 $\dot{\boldsymbol{v}} = \mathbb{J}(\boldsymbol{x})\boldsymbol{v}$ 的直接离散化（参见第 2.2.3 节中的公式（2.13））。每次更新近似特征向量后，都会应用 QR 分解进行正交归一化。

- 使用海森矩阵向量乘积和海森矩阵乘积的无矩阵实现。
- 当精确海森矩阵可用时，采用显式矩阵实现。

• 输出与 LOBPCG 一致。为降低重复进行精确特征值求解的计算成本，我们使用 $\mathbb{V}^\top\mathbb{H}\mathbb{V}$ 的对角元素进行近似，其中：– $\mathbb{H}$是该点处的海森矩阵（此处为负雅可比矩阵）。– $\mathbb{V}$ 的各列包含正交近似单位特征向量。

注意： 实际上，由于梯度系统本质上是非梯度系统的一种特殊情况，因此也可以将幂法应用于梯度系统，前提是接口保持一致。

3. **显式欧拉离散化：** （适用于梯度系统和非梯度系统）此方法直接对理论框架中的连续 HiSD 和 GHiSD 动力学进行欧拉离散化。详情请参阅第 2.1.3 节中的算法 1 和第 2.2.3 节中的公式（2.14）。

- 梯度系统中连续 HiSD 动力学的无矩阵欧拉离散化。
- 梯度系统的显式海森矩阵离散化。
- **：** 连续 GHiSD 动力学的无矩阵欧拉离散化。
- 连续 GHiSD 动力学的欧拉离散化。
- 输出和特征值近似方法与幂法保持一致。

注意： 鉴于第 2.2.3 节所述的直接显式欧拉离散与幂法等价，并出于计算效率考虑，该软件包默认对非梯度系统采用幂法作为特征值计算方法，对梯度系统则采用专为其优化的 LOBPCG。不过，如果用户更关注 HiSD 的理论保证，也可以切换为直接显式欧拉离散化。

4. **鞍点指标验证程序：** 提供准确的鞍点指标验证。

- (提供一种精确的稀疏方法，用于计算最小的 k 特征值，避免近似方法带来的误差。此函数仅在 HiSD 迭代接近新的鞍点时调用，而非在每次迭代时调用，以降低计算成本。
- I 迭代完成后重新检查莫尔斯指标（由于精度限制，终止条件只能保证指标至少为 k ）。该函数计算海森矩阵的完整谱，并将实部足够小的特征值视为近似零值（以考虑数值误差），从而帮助检测潜在的鞍点退化。

5. **初始特征向量选择程序：** 当未提供初始特征向量时，为初始迭代点提供近似特征向量。

- (对梯度系统和非梯度系统统一应用对称化，以加快计算，因为算法的最低要求只是获取一组正交单位向量，作为迭代的初始特征向量。

6. **特征向量规范化程序：** 由于不同硬件平台上基础算法（如 LAPACK）的实现存在显著差异，因此即使多个特征值对应的特征向量构成同一特征子空间的有效基，不同设备上的结果也可能存在较大差异。将此参数设为 `True` 时，将启用规范化过程（先进行高斯消元，将矩阵化为行阶梯形，再进行格拉姆－施密特正交化），以统一特征子空间的基向量，从而提高跨设备的可复现性。

具体而言，会根据特征值重数，对每个特征值块应用相同的标准化算法。只有在具有多个特征值的复杂系统中，才需要执行此过程。对于重数简单的情况，不同设备之间可能仅出现可忽略的符号差异。正如式 (2.6) 和式 (2.12) 所示，这些细微差异不会影响解迭代的收敛行为。

注： O 由于舍入误差传播的差异，不同设备上的输出仍可能有所不同 n.

- C 对多个特征值对应的特征向量进行标准化，以提高跨设备稳定性。
- (使用高斯消元和 Gram-Schmidt 正交化对行空间进行规范化。

4.4.4 迭代搜索程序 HiSDIter.py

现在，我们来介绍求解器的关键部分：搜索迭代。**准备程序**

- I 迭代所需的初始点和初始特征向量。
- I 根据公式（2.9）实现 Barzilai-Borwein 自适应步长策略。

HiSD 迭代：**用于搜索非指标0 鞍点**

完成准备步骤后，程序结构如下。本节主要关注：

1. **基于动量的加速方案：**
   - 重球加速（包括无加速这一特殊情况，相当于动量系数为 0.0 的重球法）。

- Nesterov 加速算法。

2. **步长选择策略：**

- 使用固定时间步长 d
- 采用 BB 步长 dt=BBS _ _

3. **构建完整的迭代框架：** 根据参数配置调用相应函数。

4. **返回新发现的鞍点信息和迭代记录：** 当 instance.ReportInterval 步间隔时，记录迭代次数和梯度范数；当 instance.Verbose=True 时。

5. **迭代过程中的警告和错误处理：**

- 当搜索超出指定边界或在最大迭代次数内未能收敛时发出警告，且不会更新实例中的结果。
- 对于可能存在的退化鞍点发出警告（同时仍会记录实例中的结果）；对于非退化情形，则报告找 指标。

```
def HiSDIteration(instance):
  """
  Main iteration for HiSD method for high-index saddle points.
  """
  boundarytempsave = copy.deepcopy(instance.DataBoundary)
  xini = instance.PrimaryInitialPoint.reshape(
    -1,
  )
  tau = 0.5
  x = instance.x
  x_pre = instance.x_pre
  v = instance.v
  WhetherBB = instance.BBStep
  mom = instance.Momentum
  dt = instance.TimeStep
  if instance.SaveTrajectory:
    x_record = copy.deepcopy(x.reshape(1, instance.Dim))
  else:
    x_record = None
  gnorm_record = []
  timestep_record = [0]
  g = instance.Grad(x)
  gnorm_record.append(np.linalg.norm(g))

  # Check if Nesterov acceleration is used
  if instance.Acceleration == "nesterov":
    if instance.NesterovChoice == 2:
      NesterovTheta = 1
      NesterovTheta = (1 + (1 + 4 * NesterovTheta**2) ** (1 / 2)) / 2
    elif instance.NesterovChoice == 1:
      if instance.NesterovRestart is not None:
        plusbase = 0

    # Main iteration loop
    for j in range(1, instance.MaxIter + 1):
      if WhetherBB and j > 1:
        dt = BBStepSize(x, x_pre, g, g_pre,    tau)
      timestep_record.append(dt)
      if instance.NesterovChoice == 2:
        NesterovThetaNew = (1 + (1 + 4 * NesterovTheta**2) ** (1 / 2)) / 2
        NesterovGamma = (NesterovTheta - 1) / NesterovThetaNew
        NesterovTheta = NesterovThetaNew
      elif instance.NesterovChoice == 1:
        if instance.NesterovRestart is not None:
          NesterovGamma = (j - plusbase) / (j - plusbase + 3)
        else:
          NesterovGamma = j / (j + 3)
      w_temp = x + NesterovGamma * (x - x_pre)
      gw = instance.Grad(w_temp)
```

```
      dxw = dt * (gw - 2.0 * np.matmul(v, np.matmul(v.T, gw)))
      x_temp = w_temp - dxw
      x_pre = x
      x = x_temp
      if WhetherBB:
        g_pre = g
      v, whetherkindex = instance.EigVecMethod(x, v)
      g = instance.Grad(x)
      if instance.Verbose:
        if j % instance.ReportInterval == 0:
          print("Iteration: "+ str(j)+ f"|| Norm of gradient: {np.linalg.norm(g):.6f}")
      xnow = x.reshape(
        -1,
      )
      if instance.SaveTrajectory:
        x_record = np.append(x_record, x.reshape(1, instance.Dim), axis=0)
      if np.linalg.norm(xnow - xini) > instance.SearchArea:
        print("[WARNING] Iteration diverged: Search point exceeds feasible region. Skipping to
            next search.")
        instance.flag = False
        instance.DataBoundary = boundarytempsave
        return
      instance.DataBoundary = [[min(instance.DataBoundary[i][0], x[i, 0]),max(instance.
          DataBoundary[i][1],  x[i, 0]), ]for i in range(instance.Dim)]
      gnorm_record.append(np.linalg.norm(g))
      if np.linalg.norm(g) < instance.Tolerance:
        if whetherkindex:
          break
        if j==1 or gnorm_record[-2]>=instance.Tolerance:
          whetherk = CheckIndexk(instance, x, instance.SaddleIndex)
          if whetherk:
            break
      if instance.NesterovRestart is not None:
        if j % instance.NesterovRestart == 0:
          if instance.NesterovChoice == 2:
            NesterovTheta = 1
            NesterovTheta = (1 + (1 + 4 * NesterovTheta**2) ** (1 / 2)) / 2
          elif instance.NesterovChoice == 1:
            plusbase = j
  else:
    ... # Standard/Heavy-ball iteration logic goes here

  # Check for convergence and report results
  if j == instance.MaxIter:
    if gnorm_record[-1] >= instance.Tolerance:
      print("[WARNING] Iteration not converged: Maximum iterations reached without convergence
          . Skipping to next search.")
      instance.flag = False
      instance.DataBoundary = boundarytempsave
      return
    negnum,  zeronum,  posnum,  negvecs = FindIndex(instance, x)
    if negnum+zeronum >= instance.SaddleIndex:
      print("[Note] Due to eigenvalue approximation inaccuracies during iterations, "
        "the search trajectory may reach a qualifying saddle point without triggering a report
            .")
    else:
      print("[WARNING] Iteration not converged: Maximum iterations reached without convergence
          . Skipping to next search.")
      instance.flag = False
      instance.DataBoundary = boundarytempsave
      return
  else:
     negnum,  zeronum,  posnum,  negvecs = FindIndex(instance, x)

    print(f"[WARNING] Degenerate saddle point detected under precision tol={instance.
        PrecisionTol}: Hessian matrix may contain zero eigenvalue(s).")
    print(f"Eigenvalue spectrum: negative={negnum}, zero={zeronum}, positive={posnum}. ")
  else:
    print(f"Non-degenerate saddle point identified: Morse index ={negnum} (number of negative
        eigenvalues).")
  instance.finalindex = int(negnum)
  instance.negvecs = negvecs
  instance.IterNum = j
  instance.x = x
  instance.x_record = x_record
  instance.timestep_record = np.array(timestep_record).reshape(-1, 1)
```

```
    instance.timestep_record = np.cumsum(instance.timestep_record)
    instance.gnorm_record = gnorm_record
```

代码清单 1：HiSD 核心迭代求解器的实现，包含可选的动量加速。

`SDIteration`：用于搜索指标-0 鞍点

此函数是 `HiSDIteration` 的简化版本，专[illegible]鞍点（极小值点）。需要注意的是，迭代过程中不会使用特征向量。该函数基于梯度下降（可选用加速算法），无需海森矩阵信息来分离特征方向（例如，无需计算特征向量），从而显著减少不必要的计算。

4.4.5 总结

其实现包括以下预备模块：（1）符号计算与自动微分，（2）海森矩阵处理，以及（3）特征值求解。这些模块整合到了参数验证程序（`HiSDSolverCheckParam.py`）和迭代搜索程序（`HiSDIter.py`）中。最终整合的 `HiSDSolv[illegible]`该心模块。

4.5 解景观构建 LandscapeRun.py

该程序是核心编排模块，负责构建解景观。此前的所有组件均为实现这一目标而设计。鉴于程序篇幅较长，我们将采用分层分解方法。

4.5.1 准备：函数

```
def checkwhetherexist(instance, x, index):
  """
  Check if a saddle point already exists.
  """
  for i in range(len(instance.SaddleList)):
    if instance.SameJudgementMethod(x, instance.SaddleList[i][1]) and (index == instance.
        SaddleList[i][2]):
      return True, i
  return False, None
```

列表 2：使用用户定义的等价度量唯一标识鞍点，以避免重复的函数。

此函数用于在向上和向下搜索过程中区分不同的鞍点并合并重复项。

4.5.2 CoreCode：函数

从初始点搜索一个指标不超[illegible]标的鞍点

该算法部分旨在从给定的初始点出发，找到一个指标尽可能高且不超过`instance.MaxIndex`的鞍点。

找到的鞍点将作为向下搜索的起点。找到合适的鞍点后，循环会立即终止；否则，该过程将报告失败。

注意： 由于此函数也用于重启程序 `RestartFromPoint` 和[illegible]，因[illegible]

- 该点应满足与现有鞍点的分离条件（通过以下方式检查[illegible]by `LandscapeRun`）。
- 对于被判定为相同的点，保留梯度范数较小的那个。

在不存在鞍点列表的初始景观构建过程中，会跳过这些邻近性检查。

```
def LandscapeRun(instance):
  """
  Do the downward search automatically of HiSD to construct the solution landscape
  """
  tempList = []  # Temporary list to hold intermediate saddle points
  k = instance.MaxIndex
  np.random.seed(1121)
  # Searching for saddle points starting from MaxIndex
  while k >= 0:
    instance.calHiSD.SaddleIndex = k
    if instance.InitialEigenVectors is not None:
      instance.calHiSD.InitialEigenVectors = instance.InitialEigenVectors[:, 0:k]
    ... # Formatted output
    instance.calHiSD.run()
    if instance.calHiSD.flag:
      whetherinlist, smallestind = checkwhetherexist(instance, instance.calHiSD.x, instance.
          calHiSD.finalindex)
      if whetherinlist:
        xold = instance.SaddleList[smallestind][1]
        xnew = instance.calHiSD.x
        gold = instance.calHiSD.Grad(xold)
        gnew = instance.calHiSD.Grad(xnew)
        if np.linalg.norm(gold) > np.linalg.norm(gnew):
          instance.SaddleList[smallestind][1] = instance.calHiSD.x
        if instance.Continue:
          tempList.append([instance.SaddleList[smallestind][0], instance.SaddleList[
              smallestind][1],
              instance.SaddleList[smallestind][2], instance.SaddleList[smallestind][3]])
        if instance.BeginID != -1 and (
          instance.BeginID not in instance.SaddleList[smallestind][4]
        ):
          instance.SaddleList[smallestind][4].append(instance.BeginID)
          if instance.SaveTrajectory:
            instance.DetailRecord.append([smallestind, instance.BeginID,
                instance.calHiSD.x_record, instance.calHiSD.timestep_record])
          else:
            instance.DetailRecord.append([smallestind, instance.BeginID])
        print("Search an existing saddle point.")
      else:  # Save the saddle point and its information
        instance.SaddleList.append([instance.saddleind, instance.calHiSD.x,
            instance.calHiSD.finalindex, instance.calHiSD.negvecs, [instance.BeginID]])
        tempList.append([instance.saddleind, instance.calHiSD.x,
            instance.calHiSD.finalindex, instance.calHiSD.negvecs])
        if instance.SaveTrajectory:
          instance.DetailRecord.append([instance.saddleind, instance.BeginID,
              instance.calHiSD.x_record, instance.calHiSD.timestep_record])
        else:
          instance.DetailRecord.append([instance.saddleind, instance.BeginID])
        instance.saddleind = instance.saddleind + 1
      if instance.calHiSD.finalindex > instance.MaxIndex:
        instance.MaxIndex = instance.calHiSD.finalindex
        print("Warning! 'MaxIndex' updated due to a larger saddle point index.")
      break
    k = k - 1
  if k == -1:
    raise RuntimeError("No more saddle points found in the search area!")
```

列表 3：开始或重新启动时定位初始鞍点的算法。该程序尝试从初始猜测出发，寻找指定最大指标的鞍点；若未能收敛，则逐步降低目标指标，以此为解景观提供种子。

从高指标鞍点出发的广度优先向下搜索

从指标为 k 的鞍点定位指标为 j （$j < k$）的鞍点，核心策略包括以下步骤：

1. **方向选择：** 从 k 个可用模态中选择 j 个不稳定方向，构成维持子空间。在梯度系统中，这些不稳定方向对应海森矩阵的负特征方向；在非梯度系统中，则对应雅可比矩阵的正实部特征方向。请注意，我们的约定将后者视为“海森矩阵”的负方向，从而统一了这两种情况。

2. **扰动：** 对当前鞍点的坐标施加扰动。

3. **逃逸机制：** 当扰动沿剩余的 $k-j$ 个不稳定方向将状态从平衡态移开时：

- 系统力驱动动力学沿这些 $k-j$ 个方向演化。
- 因此， $\boldsymbol{x}$ 会逃离当前指标为 k 的鞍点邻域。
- 随后，HiSD 迭代引导系统演化至目标指标为 j 的鞍点。

为提高计算效率，实现中区分了一般鞍点搜索（$j>0$）和局部极小值搜索（$j=0$）：

- **对于 $j>0$ 情况：**

  – 从不稳定模态中选取 j 个特征向量（使用所有组合子集或最不稳定的模态），以此初始化不稳定子空间。– 生成初始随机扰动（例如，采用均匀分布或高斯分布）。– 执行标准的 HiSD 搜索，并在收敛后更新数据结构。

- **对于 $j=0$ 种情况：**

  – 省略特征子空间分析，直接应用梯度下降法。对于局部极小值，这一简化仍能保持搜索效果，同时显著降低计算成本。

注： 考虑到解景观近似对称，我们沿相反方向施加扰动（即 $\pm\boldsymbol{p}$），以更全面地识别连通鞍点。

```
    ... # continued from the preceding codes
  while tempList:
    tempsearchinitial = tempList.pop(0)
    for j in range(tempsearchinitial[2] - 1,
        max(0, tempsearchinitial[2] - instance.MaxIndexGap) - 1, -1):
      if j > 0:
        # Try all combinations of eigenvectors
        all_combinations = list(combinations(range(tempsearchinitial[2]), j))
        for r in range(len(all_combinations)):
          if instance.EigenCombination == "min":
            if r != 0:
              continue
          instance.calHiSD.SaddleIndex = j
          instance.calHiSD.InitialEigenVectors = tempsearchinitial[3][:, all_combinations[r]]
          # Generate perturbations
          PerturbationList = []
          for p in range(instance.PerturbationNumber):
              pertemp = instance.PerMethod(instance.Dim, instance.PerturbationRadius)
              pertemp_projection=tempsearchinitial[3]@(tempsearchinitial[3].T@pertemp)
              pertemp_projection_normalized=instance.PerturbationRadius*pertemp_projection/np.
                  clip(np.linalg.norm(pertemp_projection),1e-10,None)
              PerturbationList.append(pertemp_projection_normalized)
              PerturbationList.append(-pertemp_projection_normalized)
          for per in PerturbationList:
            instance.calHiSD.InitialPoint = tempsearchinitial[1] + per
            ... # Formatted output
            instance.calHiSD.run()
            if not instance.calHiSD.flag:
              continue
            whetherinlist, smallestind = checkwhetherexist(instance,
                instance.calHiSD.x, instance.calHiSD.finalindex)
            if instance.calHiSD.finalindex > j:
              print("Sorry! Because of the relaxed abort criteria, we find a saddle point with
                  higher index.")
              continue
            if whetherinlist:  # If the point is already in the list
              xold = instance.SaddleList[smallestind][1]
              xnew = instance.calHiSD.x
```

```
                gold = instance.calHiSD.Grad(xold)
                gnew = instance.calHiSD.Grad(xnew)
                if np.linalg.norm(gold) > np.linalg.norm(gnew):
                  instance.SaddleList[smallestind][1] = instance.calHiSD.x
                if (tempsearchinitial[0] not in instance.SaddleList[smallestind][4]):
                  instance.SaddleList[smallestind][4].append(tempsearchinitial[0])
                  if instance.SaveTrajectory:
                    instance.DetailRecord.append([smallestind, tempsearchinitial[0],
                        instance.calHiSD.x_record, instance.calHiSD.timestep_record])
                  else:
                    instance.DetailRecord.append([smallestind, tempsearchinitial[0]])
              else:  # Add the new saddle point
                instance.SaddleList.append([instance.saddleind, instance.calHiSD.x,
                    instance.calHiSD.finalindex, instance.calHiSD.negvecs, [tempsearchinitial
                        [0]]])
                if instance.SaveTrajectory:
                  instance.DetailRecord.append([instance.saddleind, tempsearchinitial[0],
                      instance.calHiSD.x_record, instance.calHiSD.timestep_record])
                else:
                  instance.DetailRecord.append([instance.saddleind, tempsearchinitial[0]])
                tempList.append([instance.saddleind, instance.calHiSD.x,
                    instance.calHiSD.finalindex, instance.calHiSD.negvecs])
                instance.saddleind = instance.saddleind + 1
        else:
          ... # for index-0, no eigenvectors, perturbation only
```

清单 4：用于系统性向下构造解景观的广度优先搜索（BFS）逻辑实现。

扰动

如上所述，在进行向下搜索时，我们调用 Perturbation.p 扰动方法（从已有鞍点出发搜索新的鞍点时，需要通过该方法逃离当前鞍点）。

```
def gaussianper(Dim, PerturbationRadius):
  """
  Generate Gaussian-distributed random vector and normalize it.
  """
  v = np.random.randn(Dim)
  v_normalized = PerturbationRadius * v / np.clip(np.linalg.norm(v), 1e-10, None)
  return v_normalized.reshape(-1, 1)


def uniformper(Dim, PerturbationRadius):
  """
  Generate uniform-distributed random vector and normalize it.
  """
  v = np.random.uniform(-1, 1, Dim)
  v_normalized = PerturbationRadius * v / np.clip(np.linalg.norm(v), 1e-10, None)
  return v_normalized.reshape(-1, 1)

... # Called in the core loop.
pertemp = instance.PerMethod(instance.Dim, instance.PerturbationRadius)
pertemp_projection = tempsearchinitial[3] @ (tempsearchinitial[3].T @ pertemp)
pertemp_projection_normalized = instance.PerturbationRadius * pertemp_projection / np.clip(np.
    linalg.norm(pertemp_projection), 1e-10, None)
PerturbationList.append(pertemp_projection_normalized)
PerturbationList.append(-pertemp_projection_normalized)
...
```

清单 5：生成归一化的随机扰动，以沿不稳定方向从临界点逃离。

重新启动向上搜索

我们可 使用向上搜索构建完整的解景观。有两种选择 ：

- 重启函数 从点重启 用 重新求解。
- 函数 从鞍点重 有的鞍点继续求解（请注意，重启时需要施加小幅随机扰动，以便脱离该鞍点）。

```
class Landscape:
  ... # other methods

  def RestartFromPoint(self, RestartPoint, MaxIndex):
    """
    Restart the landscape process.
    """
    RestartPoint = np.array(RestartPoint).reshape(-1, 1)
    self.calHiSD.InitialPoint = RestartPoint
    self.InitialPoint = RestartPoint
    historyMaxIndex = self.MaxIndex
    self.MaxIndex = MaxIndex
    self.BeginID = -1
    self.Continue = True
    LandscapeRun(self)
    self.Continue = False
    self.MaxIndex = historyMaxIndex

  def RestartFromSaddle(self, BeginID, Perturbation, MaxIndex):
    """
    Run the landscape process.
    """
    if BeginID >= len(self.SaddleList):
      raise ValueError("Invalid saddle ID")
    RestartPoint = self.SaddleList[BeginID][1] + np.array(Perturbation).reshape(
      -1, 1
    )
    self.calHiSD.InitialPoint = RestartPoint
    self.InitialPoint = RestartPoint
    historyMaxIndex = self.MaxIndex
    self.MaxIndex = MaxIndex
    self.BeginID = BeginID
    self.Continue = True
    LandscapeRun(self)
    self.Continue = False
    self.MaxIndex = historyMaxIndex
```

列表 6：用于从特定点或现有鞍点重新开始 Landscape 构造的方法。

4.6 数据处理模块

此模块负责保存和可视化搜索结果。由于其主要涉及后处理，而非数值计算，因此我们省略代码细节，重点介绍主要工作流程。实现细节请参见 GitHub 代码仓库。

4.6.1 搜索轨迹绘图程序

此程序　可视化不同数据维度下采用不同方法的搜索轨迹　s:

- 高维数据：使用用户定义的投影映射到二维。
- 2维数据：生成网格点以计算能量值，绘制等高线（contour/contourf），根据鞍点的指标用不同颜色标记，并绘制搜索轨迹（详细轨迹或直线连接）。
- 1维数据：绘制函数曲线，标记鞍点位置，并用直线连接相关鞍点。

4.6.2 解景观绘图程序

该程序通过以下步骤生成解景观的拓扑连接图：

1. 配置绘图参数。

2. 构建图结构。3. 计算节点坐标（用于绘图）。4. 绘制各元素：按莫尔斯指标从高到低绘制节点（确保高指标节点显示在上方）、动态调整节点大小，并绘制边。5. 添加并调整节点标签。

4.6.3 数据保存程序

该程序以指定格式保存实例对象中的 SaddleList 数据，包括 JSON（.json）、MATLAB（.mat）和 Python Pickle（.pkl）格式。它会提取每个鞍点的 ID、位置、莫尔斯指标和父节点，并根据用户选择的输出格式进行序列化。

5 示例集

本节介绍四个示例，以展示我们实现方案的功能。

5.1 蝴蝶函数

此示例展示了一个能量热图形似蝴蝶的系统。其能量函数定义如下：

$$E(x,y) = x^4 - 1.5x^2y^2 + y^4 - 2y^3 + y^2 + x^2y - 2x^2.$$

首先，我们通过导入 Landscape 类来初始化该软件包。

```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.getcwd(), '..', 'saddlescape-1.0')))
from saddlescape import Landscape
import numpy as np
# import packages needed
```

代码清单 7：导入所需的库和主要的 Landscape 类。

然后，我们定义符号能量函数并初始化参数。

```
# given energy function
energyfunction = 'x1**4 -1.5*x1**2*x2**2+ x2**4 - 2*x2**3 + x2**2 + x1**2*x2 - 2*x1**2'
# parameter initialization
x0 = np.array([0.1, 0.1]) # initial point
dt = 1e-2 # time step
k = 2 # the maximum index of saddle point
acceme = 'none'
maxiter = 10000 # max iter
```

代码清单 8：符号能量函数的定义及系统参数的初始化。

接下来，我们实例化求解器并执行 运行 方法，[illegible]建解景观。

```
MyLandscape = Landscape(MaxIndex=k, AutoDiff=True, EnergyFunction=energyfunction, EigenMethod=
'e
```

代码清单 9：求解器的实例化与解景观构建的执行。

部分输出如下：

```
  HiSD Solver Configuration:
  ------------------------------
  [HiSD] Current parameters (initialized):
  [Config Sync] ‘Dim‘ parameter auto-adjusted to 2 based on ‘InitialPoint‘ dimensionality.
  Parameter ‘NumericalGrad‘ not specified - using default value False.
  Using ‘EnergyFunction‘ instead of ‘Grad‘ - enabling auto-differentiation mode.
  Parameter ‘Momentum‘ not specified - using default value 0.0.
  Parameter ‘BBStep‘ not specified - using default value False.
  Parameter ‘DimerLength‘ not specified - using default value 1e-05.
  Parameter ‘Tolerance‘ not specified - using default value 1e-06.
  Parameter ‘NesterovChoice‘ not specified - using default value 1.
  Parameter ‘SearchArea‘ not specified - using default value 1000.0.

  ... # more parameter checks

  Gradient system detected. Activating HiSD algorithm.

  Landscape Configuration:
  ------------------------------
  [Landscape] Current parameters (initialized):
  Parameter ‘SameJudgementMethod‘ not specified - using default value <function
      LandscapeCheckParam.<locals>.<lambda> at 0x000002EBD1A2F240>.
  Parameter ‘PerturbationMethod‘ not specified - using default value uniform.
  Parameter ‘InitialEigenVectors‘ not specified - using default value None.
  Parameter ‘SaveTrajectory‘ not specified - using default value True.
  Parameter ‘MaxIndexGap‘ not specified - using default value 1.

  Start running:
  ------------------------------

  From initial point search index-2:
  ------------------------------

  Iteration: 100|| Norm of gradient: 0.176823
  Iteration: 200|| Norm of gradient: 0.084819
  ...
  Iteration: 1200|| Norm of gradient: 0.000004
  Iteration: 1300|| Norm of gradient: 0.000001
  Non-degenerate saddle point identified: Morse index =2 (number of negative eigenvalues).

  From saddle point (index-2, ID-0) search index-1:
  ------------------------------

  Iteration: 100|| Norm of gradient: 0.023111
  Iteration: 200|| Norm of gradient: 0.061258
  ...
  Iteration: 900|| Norm of gradient: 0.000024
  Iteration: 1000|| Norm of gradient: 0.000003
  Non-degenerate saddle point identified: Morse index =1 (number of negative eigenvalues).

  ... # more log messages during searches
```

清单 10：展示初始化和迭代进度的代表性控制台输出。

搜索完成后，可利用这些数据进行各种后处理。例如，我们绘制搜索轨迹。

```
MyLandscape.DrawTrajectory(ContourGridNum=100, ContourGridOut=25, DetailedTraj=True)
# Draw the search path.
```

清单 11：用于可视化搜索轨迹的命令。

此外，还支持绘制解景观并保存数据。

```
MyLandscape.DrawConnection()
MyLandscape.Save('output/Ex_Butterfly')
# Save the data
```

清单 12：用于可视化连通图并保存结果的命令 **s.**

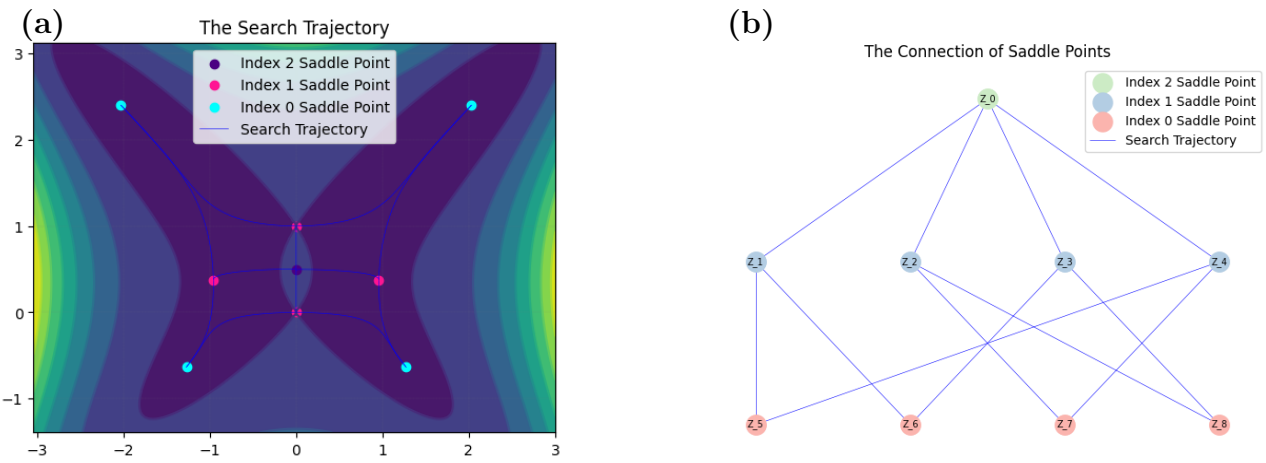


图 2：蝴蝶势能面的解景观重构。(a)从 $x_0 = (0.1, 0.1)$ 出发的搜索轨迹的二维投影。（**b**）解景观展示了指标-2（极大值）、指标-1（过渡态）和指标-0（极小值点）鞍点之间的连通关系。

此外，还可以通过特定的后处理生成搜索过程的动画图像。详情请参见

https://github.com/HiSDpackage/saddlescape/blob/main/ Examples/Ex 1 Butterfly.ipynb.

最终输出作为补充材料提供，可通过
https://github.com/HiSDpackage/saddlescape/blob/main/Examples/output/Ex Butterfly.mp4 访问。

5.2 M¨uller-Brown 势

在此示例中，我们演示如何通过向上搜索重启算法，以获得完整的解景观。

W我们将 M¨uller-Brown 势定义为高斯势的线性组合：

$$E_{MB}(x,y) = \sum_{i=1}^{4} A_i \exp[a_i(x-\bar{x}_i)^2 + b_i(x-\bar{x}_i)(y-\bar{y}_i) + c_i(y-\bar{y}_i)^2].$$

参数设置如下：

$$\begin{aligned} &A = [-200, -100, -170, 15], \\ &a = [-1, -1, -6.5, 0.7],\ b = [0, 0, 11, 0.6],\ c = [-10, -10, -6.5, 0.7], \\ &\bar{x} = [1, 0, -0.5, -1],\ \bar{y} = [0, 0.5, 1.5, 1]. \end{aligned}$$

与第一个示例类似，我们初始化求解器（此处从 (0.15,0.25) 开始），搜索鞍点，然后绘制搜索轨迹和解景观。

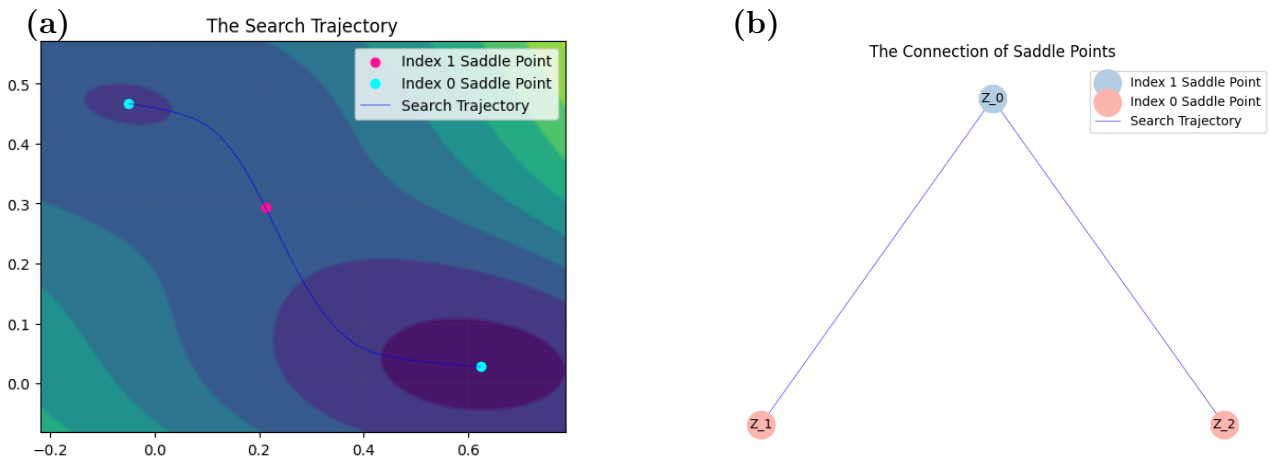


图 3：M¨uller-Brown 势能景观的初步探索。 图 (a) 和(b)展示了通过单次初始搜索得到的不完整临界点及其连接，凸显了迭代重启的必要性。

然而，M¨uller-Brown 势呈现多峰分布，导致初始搜索不完整。因此，我们从一个局部极小值重新开始搜索：

```
MyLandscape.RestartFromSaddle(1,-np.array([[-0.01],[0]]),1)
# restart the search using an upward search
```

代码清单 13：从特定鞍点重新启动搜索，通过向上搜索扩展景观。

此外，我们绘制图形以进行分析并保存数据。

```
MyLandscape.DrawTrajectory(ContourGridNum=100, ContourGridOut=25, DetailedTraj=True)
# Draw the search path.
MyLandscape.DrawConnection()
MyLandscape.Save('output/Ex_MBP','pickle')
# Save the data
```

代码清单 14：用于可视化和数据存储的后处理命令。

现在，我们得到了一个完整的解景观。

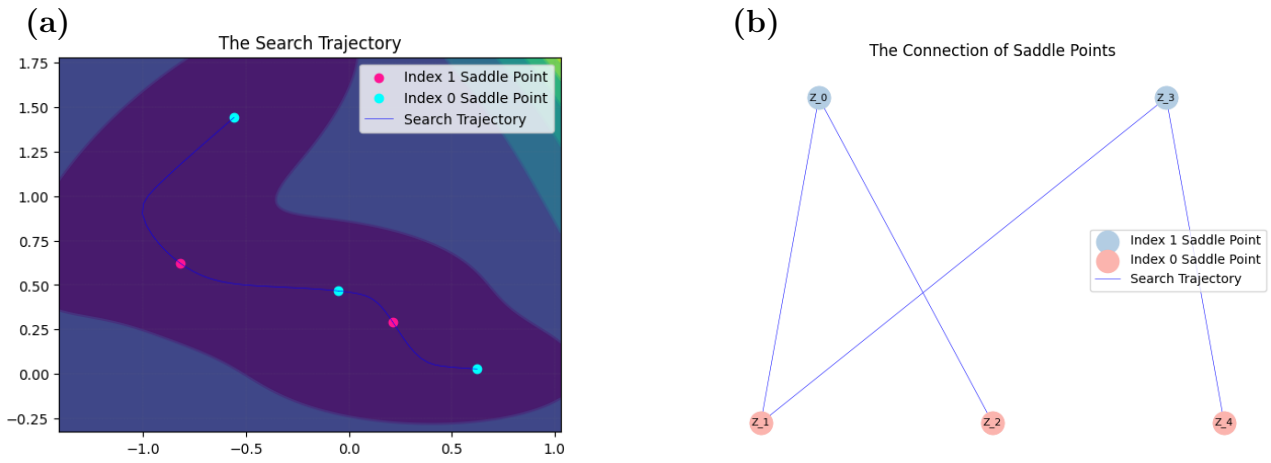


图 4：完整的 M¨uller-Brown 势能解景观。 这些完整的图 (a) 和(b)通过从初步阶段（图 3）识别出的局部极小值重新启动搜索算法获得，成功恢复了全局拓扑。

5.3 三次函数

在本例中，我们演示如何通过投影来可视化搜索轨迹，所使用的系统具有能量函数

$$E(x) = \sum_{j=1}^{n} j(x_j^2 - 1)^2.$$

所有鞍点都位于 n 维超立方体的顶点、面中心和体心处。

特别是，对于 $n = 3$，共有 27 个鞍点

$$(a, b, c), \quad \text{where } a, b, c \in \{-1, 0, 1\},$$

它们是立方体的顶点、面中心和体心。

我们沿用前面示例中的方法。由于该系统是三维的，我们定义一个投影函数来可视化轨迹。

```
def proj_func(input):
  output = np.hstack((1.0 * input[:, [0]]+ 1.5 * input[:, [1]], 1.0 * input[:, [0]]+ 2.5 *
      input[:, [2]]))
  return output
```

代码清单 15：De　将三维坐标映射到二维以供可视化的投影函数的定义 .

然后，我们可以通过投影绘制搜索轨迹和解景观 .

```
MyLandscape.DrawTrajectory(ContourGridNum=100, ContourGridOut=25, Projection=proj_func)
# Draw the search path.
MyLandscape.DrawConnection()
MyLandscape.Save('output/Ex_Cubic','mat')
# Save the data
```

列表16：生　成投影轨迹图和该 **e**
三次系统的命令。

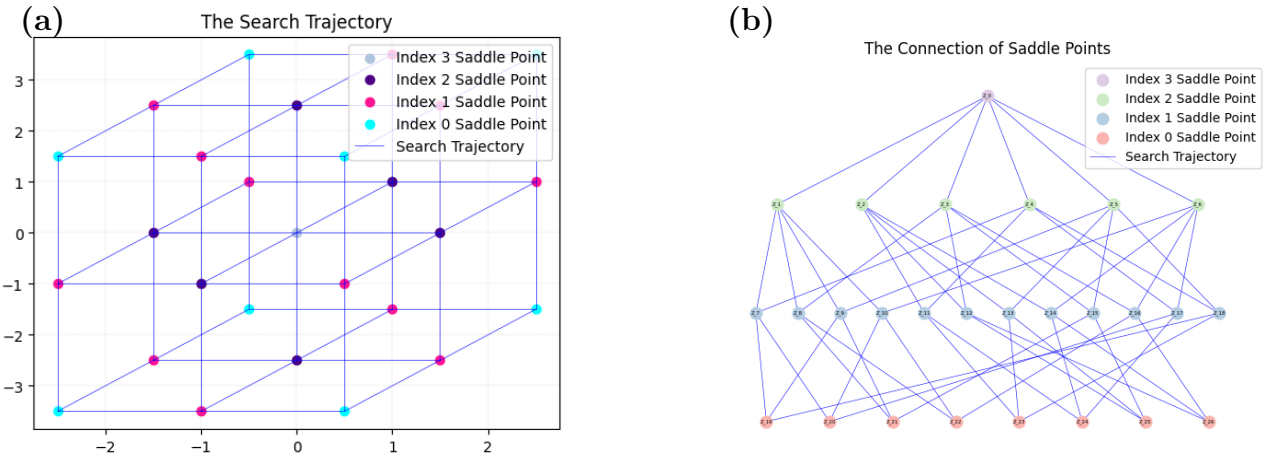


图5：三维三次函数景观的可视化（$n = 3$）。**（a）**使用坐标的线性组合将搜索轨迹投影到二维平面上。**（b）**全连通解景观，呈现出超立方体顶点和面中心所特有的对称结构。

5.4 相场模型

在本示例中，我们构建一个泛函系统的解景观。具体而言，我们考虑一个二维相场模型：

$$E(\phi) = \int_{\Omega} \left(\frac{\kappa}{2} |\nabla \phi|^2 + \frac{1}{4} (1 - \phi^2)^2 \right) dx.$$

Allen-Cahn方程为

$$\dot{\phi} = \kappa\Delta\phi + \phi - \phi^3.$$

我们考虑 $\Omega = [0,1]^2$，并施加周期边界条件，使用 64×64 网格上的有限差分格式对区域进行离散化。空间导数采用通过卷积核实现的中心有限差分进行近似。

```python
from scipy.ndimage import convolve

def gradient_PF(x, opt):
  kappa = opt['kappa']
  n2 = len(x)
  n = int(np.sqrt(n2))
  phi = x.reshape(n, n)
  h = opt['h']
  D2 = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]]) / h**2
  conv_term = convolve(phi, D2, mode='wrap')
  F = -(kappa * conv_term + phi - phi**3)
  return F.reshape(n2, 1)
```

代码清单 17：使用有限差分近似实现相场模型的离散梯度流。

为考虑周期边界条件，我们定义了一个具有平移不变性的等价谓词。

```python
def SameSaddle(x, y):
  n= 64
  A = x.reshape(n,n)
  B = y.reshape(n,n)
  epsilon = 0.05
  row_indices = np.array([(np.arange(n) - dx) % n for dx in range(n)])
  col_indices = np.array([(np.arange(n) - dy) % n for dy in range(n)])
  row_shifted = np.zeros((n, n, n), dtype=B.dtype)
  for dx in range(n):
    row_shifted[dx] = B[row_indices[dx], :]
  for dx in range(n):
    for dy in range(n):
      shifted_B = row_shifted[dx][:, col_indices[dy]]
      max_diff = np.max(np.abs(A - shifted_B))
      if max_diff <= epsilon:
        return True
  return False
```

代码清单 18：针对周期边界条件，使用空间平移方法进行鞍点等价性检查。

或者，也可以使用快速傅里叶变换（FFT）算法计算归一化互相关，以定义平移不变的等价谓词。

```python
from numpy.fft import fft2, ifft2

def SameSaddle(x, y):
    n = 64
    A = x.reshape(n, n)
    B = y.reshape(n, n)
    # 2D Fourier transform
    fft_A = fft2(A)
    fft_B = fft2(B)
    cross_corr_spectrum = np.conj(fft_A) * fft_B
    real_space_corr = np.real(ifft2(cross_corr_spectrum))
    max_corr = np.max(real_space_corr)
    norm_factor = np.sum(A**2)
    if norm_factor < 1e-7:
        return np.sum(B**2) < 1e-7
    normalized_max_corr = max_corr / norm_factor
    if normalized_max_corr > 0.99:
        return True
    return False
```

清单 19：使用 FFT算法处理周期性偏移的鞍点等价性检查。

我们关注具有 $\kappa = 0.02$ 的系统。

```
N = 64
opt = {'kappa': 0.02, 'N': N, 'h': 1/N,}
GradFunc = lambda x: gradient_PF(x, opt)
```

清单 20：相场系统参数和梯度函数的配置。

我们初始化求解器并求解该系统。

```
# parameter initialization
x0 = np.array([0 for i in range(N**2)]) # initial point
dt = 1e-3 # time step
k = 5
acceme = 'nesterov'
neschoice = 1
nesres = 200
mom = 0.8
maxiter = 2000 # max iter

MyLandscape = Landscape(MaxIndex=k, AutoDiff=False, Grad=GradFunc, DimerLength=1e-3,
    HessianDimerLength=1e-3, EigenStepSize=1e-7, InitialPoint=x0, TimeStep=dt, Acceleration=
    acceme, SearchArea=1e4,
                        SymmetryCheck=False,Tolerance=1e-4, MaxIndexGap=3,
    EigenCombination='min',BBStep=True, NesterovChoice=neschoice, NesterovRestart=nesres,
    Momentum=mom, MaxIter=maxiter, Verbose=True, ReportInterval=10, EigenMaxIter=2,
    PerturbationNumber=2, EigvecUnified=True, SaveTrajectory=False, SameJudgementMethod=
    SameSaddle, PerturbationRadius=5.0)
# Instantiation
MyLandscape.Run()
# Calculate
```

List图 21：相场模型的 HiSD 求解器初始化与执行 **l.**

我们可以绘制解景观并保存数据。

```
MyLandscape.DrawConnection()
MyLandscape.Save('output/Ex_PhaseField','mat')
# Save the data
```

代码清单 22：保存和绘制相场景观结果的命令。

表示函数的鞍点可以通过后处理进行可视化：

```
import matplotlib.pyplot as plt
from scipy.interpolate import griddata

def plot_phi_heatmap(phi_vector, N):
  if len(phi_vector) != N * N:
    raise ValueError(f"Input shape must be {N * N}, but got {len(phi_vector)}")
  h=1/N
  x = np.linspace(h/2, 1-h/2, N)
  y = np.linspace(h/2, 1-h/2, N)
  X, Y = np.meshgrid(x, y)
  phi = phi_vector.reshape((N, N))
  grid_x, grid_y = np.mgrid[0:1:N*10j, 0:1:N*10j]
  grid_phi = griddata((X.flatten(), Y.flatten()), phi.flatten(), (grid_x, grid_y), method='
      cubic')
  # Draw heatmap
  plt.figure(figsize=(8, 6))
  plt.imshow(grid_phi, extent=(0, 1, 0, 1), origin='lower', cmap='viridis', vmin=-1, vmax=1)
  plt.colorbar(label='$\phi$ value')
  plt.title('phase field$\phi$ heatmap')
  plt.xlabel('x')
  plt.ylabel('y')
  plt.show()

for i in range(len(MyLandscape.SaddleList)):
  plot_phi_heatmap(MyLandscape.SaddleList[i][1],N)
```

代码清单 23：He lp 函数，用于将空间相场构型 $\phi(x)$ 可视化为热力图 **.**

通过将鞍点相场与解景观并列可视化，我们展示了它们之间的对应关系。

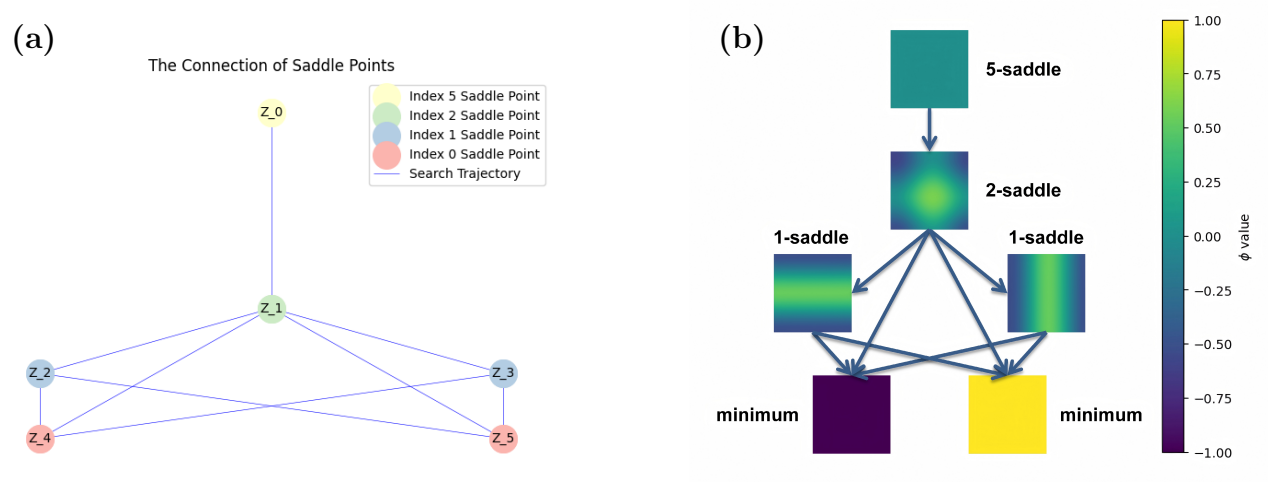


图 6：相互作用参数为 κ =0。02时相场模型的计算解景观。 (a) 解景观。(b) 插图展示了与选定鞍点对应的空间相位构型 $\phi(\boldsymbol{x})$。

此外，对于具有 $\kappa = 0.01$ 、解景观更为复杂的系统，我们可以先采用快速 BB 步长并结合 Nesterov 加速，高效地获得不完整的解景观。随后，我们使用固定小步长重新启动搜索，在局部进行搜索并构建完整的解景观。这种两阶段策略，即先加速搜索、再精细探索，有效地平衡了计算效率与解的精度。通过这种方法得到的最终解景观如下：

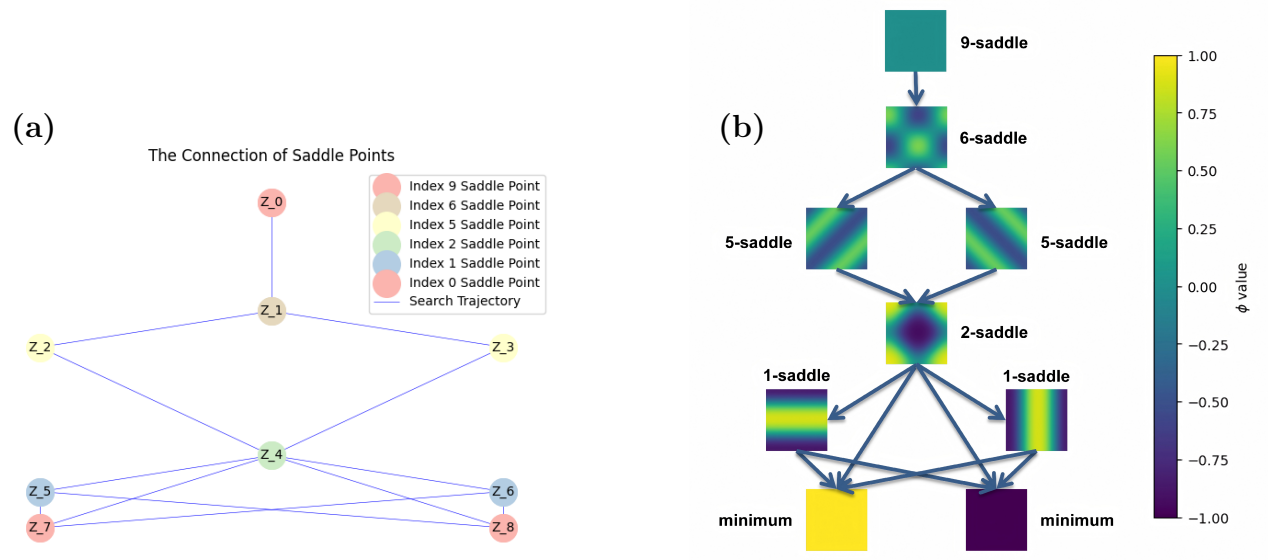


图 7：相场模型的解景观复杂度增加 $\kappa = 0.01$。 与$\kappa = 0.02$ 情形相比，参数值降低导致亚稳态和高指标鞍点大量增加。

6 结论与未来工作

6.1 结论

软件包 SaddleScape V1.0 引入了一个统一的计算框架，通过先进的鞍点动力学算法（即 HiSD 及其变

体）探索解景观。该软件包旨在兼顾易用性与灵活性，使新手用户和领域专家都能轻松分析复杂系统并构建解景观。该软件包的主要成果包括：

- **自动化解景观构建：** 一种系统化、分层且高度自动化的解景观构建流程，基于向下搜索和向上搜索、随机扰动，以及采用先进先出队列调度的广度优先搜索策略。该流程取代了传统做法：针对鞍点之间的每种连接，手动设计独立的搜索。

- **统一框架与通用输入支持：** 在单一接口中统一实现 HiSD 及其变体（例如用于非梯度系统的 GHiSD，以及采用动量加速的 AHiSD）。该软件包可通过解析表达式或 Python 可调用对象兼容梯度系统和非梯度系统，也支持空间离散化后的泛函系统，并提供默认参数以减少用户干预。

- **灵活高效的数值实现：** 解耦的数值模块（例如 `AnaCal`、`HessianMatrix`、`EigMethod`）使用户能[illegible]算子、特征值求解器，以及步长或加速策略。符号计算、自动微分，以及基于二聚体型有限差分的无矩阵海森向量积/雅可比向量积，与高效的向量化线性代数程序相结合，从而提升稳健性和计算效率，尤其适用于高维问题。

- **景观级控制、模块化与易用性：** 用户可以配置系统定义、求解器参数、最大指标、扰动策略以及用于识别不同鞍点的等价判据，还可以设置从现有鞍点或新的初始点重新启动的流程，从而细化或扩展部分构建的景观。该软件包由多个模块组成，分别用于参数检查、导数处理、特征值计算、迭代鞍点动力学求解器以及解景观构建。许多参数都提供了默认值，降低了新用户的使用门槛，同时也允许高级用户微调数值细节，并通过替换相应模块来尝试更先进的算法。

- **集成后处理、可视化与数据导出：** 针对搜索轨迹可视化、解景观构建以及多种格式（JSON、MATLAB 和 Python Pickle）的数据导出，软件包提供了自动化工作流程，使用户能够直观地了解搜索过程和最终得到的解景观，并便于后续分析及与其他计算工具集成。

通过将数值和算法层面的复杂性封装为直观的接口与自动化工作流程，SaddleScape V1.0 将理论上的鞍点动力学与凝聚态物理、材料科学、生物物理建模和非线性动力学等领域的实际应用联系起来，使研究人员能够专注于特定领域的洞见，而不必纠结于实现细节。

6.2 未来工作

为推进 SaddleScape 软件包的发展，仍有多个方向值得进一步探索。在算法方面，一个自然的拓展方向是集成 CHiSD [43], 基于代理模型 [17, 58], 的数据驱动形式，以及 Func- tional HiSD [61] 等无限维扩展。另一个重要方向是利用分布式计算框架开发并行 HiSD 算法，从而更高效地构建高维复杂系统的解景观。

从技术角度来看，可将 GPU 加速线性代数和稀疏矩阵运算用于大规模问题，并且值得将自动微分支持扩展到随机和隐式动力系统。就易用性而言，未来的工作包括开发

图形用户界面（GUI），并加强与 Jupyter Notebooks 的集成，同时建设协作平台，以便共享预配置的工作流和用户贡献的模块。

最后，跨学科应用具有巨大潜力。值得探索的方向包括材料科学（如相变）、生物化学（如蛋白质折叠）和机器学习（如损失景观分析）。在此背景下，逆向设计框架为实验验证提供了重要途径：首先指定期望的宏观性质或目标构型，然后在解景观上利用优化或基于学习的算法，识别候选系统参数，以便随后在实验室中实现并测试。

SaddleScape 软件包将继续发展为一个开源平台，促进计算数学与应用科学之间的合作。其模块化设计和社区驱动的开发旨在使其成为跨学科解景观探索的核心工具包。

7 致谢

我们感谢张平文教授和殷建远教授对 HiSD 算法的贡献。我们还要感谢于兵博士、罗跃博士、郑相成教授、施宝明博士、刘元凯和王浩然，他们为该算法的推广和实际应用作出了重要贡献。

8 资助

本研究得到国家自然科学基金委员会（编号：12225102、T2321001 和 12288101）、国家重点研发计划（2024YFA0919500）、JR25003 以及北京市杰出青年科学家计划（编号：JWZQ20240101027）的资助。

9 代码可用性

源代码可在代码仓库 https://github.com/HiSDpackage/saddlescape 中获取。该软件包的介绍网站位于 https://hisdpackage.github.io/saddlescape。

参考文献

[1] J. Baker，一种定位过渡态的算法，《计算化学杂志》，*7*（*1986*），第*385–395*页。[2] J. Barzilai and J. M. Borwein，两点步长梯度方法，《*IMA*数值分析杂志》，*8*（*1988*），第*141–148*页。[3] C. Chen and Z. Xie，求解非线性问题多个解的搜索扩展法，《计算机与应用数学》，*47*（*2004*），第*327–343*页。[4] H. Chen, S. Kandasamy, S. Orszag, R. Shock, S. Succi, and V. Yakhot，用于湍流的扩展玻尔兹曼动理学方程，《科学》，*301*（*2003*），第*633–636*页。[5] W. W. Chen, M. Niepel, and P. K. Sorger，生化反应建模的经典与当代方法，《基因与发育》，*24*（*2010*），第*1861–1875*页。[6] X. Cheng, L. Lin, W. E, P. Zhang, and A.-C. Shi，嵌段共聚物中有序相的成核，《物理评论快报》，*104*（*2010*），第*148301*页。[7] R. Das and D. J. Wales，用于预测患者出院的机器学习能量景观，《物理评论*E*》，*93*（*2016*），第*063310*页。

[8] F. Draxler, K. Veschgini, M. Salmhofer, and F. Hamprecht，神经网络能量景观中几乎不存在势垒，载于《第*35*届国际机器学习会议论文集》，机器学习研究论文集第*80*卷，*2018*年*7*月，第*1309–1318*页。[9] W. E, C. Ma, and L. Wu，梯度下降动力学下双层神经网络与随机特征模型优化及泛化性质的比较分析，中国科学：数学，*63*（*2020*），第*1235–1258*页。[10] P. E. Farrell, A. Birkisson, and S. W. Funke，寻找非线性偏微分方程不同解的去膨胀技术，*SIAM*科学计算杂志，*37*（*2015*），第*A2026–A2045*页。[11] Y. Han, J. Harris, A. Majumdar, and L. Zhang，约化*Landau-de Gennes*模型中的弹性各向异性，英国皇家学会会刊*A*辑：数学、物理与工程科学，*478*（*2022*），第*20210966*页。[12] Y. Han, J. Yin, Y. Hu, A. Majumdar, and L. Zhang，简化*Ericksen-Leslie*模型的解景观及其与约化*Landau-de Gennes*模型的比较，英国皇家学会会刊*A*辑：数学、物理与工程科学，*477*（*2021*），第*20210458*页。[13] Y. Han, J. Yin, P. Zhang, A. Majumdar, and L. Zhang，六边形上约化*Landau–de Gennes*模型的解景观，《非线性》，*34*（*2021*），第*2048–2069*页。[14] A. Heyden, A. T. Bell, and F. J. Keil，寻找化学反应过渡态的高效方法：改进的二聚体方法与分区有理函数优化方法的比较，《化学物理学杂志》，*123*（*2005*），第*224101*页。[15] D. T. Leeson, F. Gai, H. M. Rodriguez, L. M. Gregoret, and R. B. Dyer，复杂能量景观上的蛋白质折叠与解折叠，美国国家科学院院刊，*97*（*2000*），第*2527–2532*页。[16] L. Li, B. Yu, P. Gao, J. Lv, L. Zhang, Y. Wang, and Y. Ma，通过驻点网络表示晶体势能面，《材料学报》，*281*（*2024*），第*120403*页。[17] Y. Liu, L. Zhang, and J. Zhao，用于搜索鞍点和解景观的神经网络高指标鞍点动力学方法。*arXiv:2411.16200*，*2024*年。[18] Y. Luo, L. Zhang, P. Zhang, Z. Zhang, and X. Zheng，高指标鞍点动力学的半隐式方法及其在构造解景观中的应用，偏微分方程数值方法，*40*（*2024*），第*e23123*页。[19] Y. Luo, L. Zhang, and X. Zheng，用于搜索高指标鞍点的加速高指标鞍点动力学方法，《科学计算杂志》，*102*（*2025*），第*31*页。[20] Y. Luo, X. Zheng, X. Cheng, and L. Zhang，离散高指标鞍点动力学的收敛性分析，*SIAM*数值分析杂志，*60*（*2022*），第*2731–2750*页。[21] F. Mallamace, C. Corsaro, D. Mallamace, S. Vasi, C. Vasi, P. Baglioni, S. V.Buldyrev, S.-H. Chen, and H. E. Stanley，蛋白质折叠与解折叠中的能量景观，美国国家科学院院刊，*113*（*2016*），第*3159–3163*页。[22] D. Mehta，通过数值多项式同伦延拓法寻找势能景观的全部驻点，《物理评论*E*》，*84*（*2011*），第*025702*页。[23] S. Miao, L. Zhang, P. Zhang, and X. Zheng，高指标鞍点动力学正交归一化*Runge–Kutta*格式的构造与分析，非线性科学与数值模拟通讯，*145*（*2025*），第*108731*页。

[24] J. Milnor,《莫尔斯理论》。（*AM-51*），第*51*卷，普林斯顿大学出版社，普林斯顿，*1963*年。[25] Q. Nie, L. Qiao, Y. Qiu, L. Zhang, and W. Zhao, 噪声控制与效用：从调控网络到空间模式形成，《中国科学：数学》，*63*（*2020*年），第*425–440*页。[26] J. N. Onuchic, Z. Luthey-Schulten, and P. G. Wolynes, 蛋白质折叠理论：能量景观视角，《物理化学年度评论》，*48*（*1997*年），第*545–600*页。[27] L. Qiao, W. Zhao, C. Tang, Q. Nie, and L. Zhang, 能够实现适应与噪声衰减双重功能的网络拓扑结构，《细胞系统》，*9*（*2019*年），第*271–285.e7*页。[28] B. Shi, Y. Han, C. Ma, A. Majumdar, and L. Zhang, 一种改进的近晶相液晶*Landau–de Gennes*理论：相变与结构转变，*SIAM*应用数学杂志，*85*（*2025*年），第*821–847*页。[29] B. Shi, Y. Han, A. Majumdar, and L. Zhang, 具有简并平面边界条件的长方体中向列相液晶的多稳态，*SIAM*应用数学杂志，*84*（*2024*年），第*756–781*页。[30] B. Shi, Y. Han, J. Yin, A. Majumdar, and L. Zhang, 三维长方体上*Landau-de Gennes*自由能临界点的层级结构，《非线性》，*36*（*2023*年），第*2631*页。[31] B. Shi, Y. Han, and L. Zhang, 矩形限域中的向列相液晶：解景观与分岔，*SIAM*应用数学杂志，*82*（*2022*年），第*1808–1828*页。[32] M. Shuai, L. Ziqi, L. Zhang, P. Zhang, and X. Zheng, 高指标鞍点动力学的*Adams*显式离散化的构造与分析，*CSIAM*应用数学汇刊，*6*（*2025*年），第*625–650*页。[33] H. Su, H. Wang, L. Zhang, J. Zhao, and X. Zheng, 用于寻找鞍点和解景观的改进高指标鞍点动力学，*SIAM*数值分析杂志，*63*（*2025*年），第*1757–1775*页。[34] R. Temam,《纳维–斯托克斯方程：理论与数值分析》，《数学及其应用研究》第*2*卷，*Elsevier Science Publishers B.V.*，阿姆斯特丹，第*3*次修订版，*1984*年。[35] D. Wales,《能量景观：在团簇、生物分子和玻璃中的应用》，剑桥分子科学，剑桥大学出版社，剑桥，*2004*年。[36] D. J. Wales and J. P. K. Doye, 通过盆地跳跃进行全局优化及含有多达*110*个原子的*Lennard-Jones*团簇的最低能量结构，《物理化学杂志*A*辑》，*101*（*1997*年），第*5111–5116*页。[37] S. Wang, T. Wang, S. Wu, L. Zhang, and X. Zou, 数学建模与解景观揭示肿瘤生态微环境中的癌症进展动力学，*SIAM*应用数学杂志，*85*（*2025*年），第*50–77*页。[38] W. Wang, L. Zhang, and P. Zhang, 液晶的建模与计算，《数值学报》，*30*（*2021*年），第*765–851*页。[39] Z. Xu, Y. Han, J. Yin, B. Yu, Y. Nishiura, and L. Zhang, 二维限域下二嵌段共聚物-均聚物模型的解景观，《物理评论*E*》，*104*（*2021*年），第*014505*页。[40] G. Xue, X. Zhang, W. Li, L. Zhang, Z. Zhang, X. Zhou, D. Zhang, L. Zhang, andZ. Li, 融入逻辑的基因调控网络揭示细胞命运决策的原理，*eLife*，*12*（*2024*年），第*RP88742*页。

[41] X. Yao, J. Xu, and L. Zhang, 柱状-双连续陀螺界面中的跃迁路径，《计算物理通讯》，*32*（*2022*年），第*810–828*页。[42] J. Yin, Z. Huang, Y. Cai, Q. Du, and L. Zhang, 揭示旋转玻色-爱因斯坦凝聚体的激发态，《创新》，*5*（*2024*年）。[43] J. Yin, Z. Huang, and L. Zhang, 等式约束下解景观的约束高指标鞍点动力学，《科学计算杂志》，*91*（*2022*年），第*62*页。[44] J. Yin, K. Jiang, A.-C. Shi, P. Zhang, and L. Zhang, 连接晶体与准晶的跃迁路径，《美国国家科学院院刊》，*118*（*2021*年），第*e2106230118*页。[45] J. Yin, Y. Wang, J. Z. Y. Chen, P. Zhang, and L. Zhang, 在复杂能量景观上构建路径图，《物理评论快报》，*124*（*2020*年），第*090601*页。[46] J. Yin, B. Yu, and L. Zhang, 利用广义高指标鞍点动力学搜索解景观，《中国科学：数学》，*64*（*2021*年），第*1801–1816*页。[47] J. Yin, L. Zhang, and P. Zhang, 用于寻找高指标鞍点的基于高指标优化的收缩二聚体方法，*SIAM*科学计算杂志，*41*（*2019*年），第*A3576–A3595*页。[48] , *Onsager*模型的解景观识别出非轴对称临界点，《物理学*D*辑》，*430*（*2022*年），第*133081*页。[49] B. Yu, X. Zheng, P. Zhang, and L. Zhang, 通过快速近似算法计算非线性空间分数阶问题的解景观，《计算物理学杂志》，*468*（*2022*年），第*111513*页。[50] P. Yu, Q. Nie, C. Tang, and L. Zhang, *Nanog*诱导的中间态在调控干细胞分化与重编程中的作用，《*BMC*系统生物学》，*12*（*2018*年），第*22*页。[51] L. Zhang, 复杂系统解景观的构造，《中国科学数学》，*45*（*2023*年），第*267–283*页。[52] L. Zhang, L.-Q. Chen, and Q. Du, 固态相变中临界晶核的形态，《物理评论快报》，*98*（*2007*年），第*265703*页。[53] L. Zhang, K. Radtke, L. Zheng, A. Q. Cai, T. F. Schilling, and Q. Nie, 噪声驱动斑马鱼后脑基因表达边界变得更加锐利，《分子系统生物学》，*8*（*2012*年），第*613*页。[54] L. Zhang, P. Zhang, and X. Zheng, 高指标鞍点动力学欧拉离散化的误差估计，*SIAM*数值分析杂志，*60*（*2022*年），第*2925–2944*页。[55] ,高维球面上约束高指标鞍点动力学的离散化与指标稳健误差分析，《中国科学：数学》，*66*（*2023*年），第*2347–2360*页。[56] ,球面约束高指标鞍点动力学半隐式方法的误差估计，《数学学报》*B*辑，*44*（*2023*年），第*765–780*页。[57] ,具有局部*Lipschitz*条件的收缩二聚体鞍点动力学的数学与数值分析，*CSIAM*应用数学汇刊，*4*（*2023*年），第*157–176*页。[58] ,用于寻找鞍点和解景观的无模型收缩二聚体鞍点动力学，《日本工业与应用数学杂志》，*40*（*2023*年），第*1677–1693*页。

[59] ——,采用不准确模型对高指标鞍点动力学进行数值离散化的概率误差估计，《应用数学年刊》，*40*（*2024*年），第 *1–20* 页。[60] ，通过数值分析理解高指标鞍点动力学，《数学科学通讯》，*23*（*2025*年），第 *541–560* 页。[61] L. Zhang, X. Zheng, and S. Zhu, 若干半线性椭圆问题的鞍点动力学数值分析。*arXiv:2508.01534*，*2025*年。[62] X. Zhang, Z. Li, and L. Zhang, 基于超解景观构建细胞命运决定的整体图谱。*bioRxiv 2024.11.28.625944*，*2024*年。[63] Y. Zhang, X. Gu, Y. Wang, X. Xu, and L. Zhang, 粗糙表面上液滴的解景观：润湿转变与定向输运，《软物质》，*21*（*2025*年），第 *2729–2737* 页。[64] Y. Zhang, X. Yang, L. Zhang, Y. Li, T. Zhang, and S. Sun, 使用指数时间差分类高指标鞍点动力学分析两相多组分恒容恒温闪蒸系统的能量景观，《计算物理学杂志》，*477*（*2023*年），第 *111916* 页。[65] T. Zhou, L. Zhang, P. Zhang, A.-C. Shi, and K. Jiang, 十角准晶的成核与相变，《化学物理学杂志》，*161*（*2024*年），第 *164503* 页。

参数参考

以下是第 4.1.2 节中提到的详细参数列表。

System Parameters			
Parameter	**Description**	**Data Type**	**Default/Behavior**
Dim (Optional)	System dimension. Manual override must match dimensions of **InitialPoint**.	**int**(positive)	Inferred from **InitialPoint**.
EnergyFunction **(Conditional Required)**	Specifies the energy function for gradient systems.	Python: **Callable[[np.ndarray], float]** (**np.ndarray** shape: **(d,1)**) Symbolic: **str** using **x1, ..., xd** variables	Conditional required.
Grad **(Conditional Required)**	Specifies the gradient function for gradient systems or the negative of the vector field for non-gradient systems.	Python: **Callable[[np.ndarray], np.ndarray]** (**np.ndarray** shape: **(d,1)**) Symbolic: **list[str]**	Conditional required.
AutoDiff (Optional)	Enables automatic differentiation of the energy function.	**bool**	Default: **True** if **EnergyFunction** is provided. **False** if **Grad** is provided.
NumericalGrad (Optional)	Enables numerical gradient approximation. Only active when **Grad** is unspecified.	**bool**	**True**: Use finite difference approximation. **False**: (Default) Use analytical gradient from **EnergyFunction**.
DimerLength (Optional)	Displacement length for numerical gradient approximations.	**float**(positive)	Default: 1e-5
SymmetryCheck (Optional)	Verifies gradient system properties via Hessian symmetry.	**bool**	**True**: (Default) Auto-detect. **False**: Bypass checks (improves performance).
GradientSystem (Optional)	Explicit declaration of gradient system nature.	**bool**	Auto-detect if missing.
Landscape Parameters			
Parameter	**Description**	**Data Type**	**Default/Behavior**
MaxIndex (Optional)	Maximum saddle index (k) to compute. Index 0: Uses standard SD (Steepest Descent) method. Index ≥1: Uses HiSD (High-index Saddle Dynamics) method. **Constraint**: 0 ≤ value ≤ **Dim**.	int(non-negative)	Default: 6
MaxIndexGap (Optional)	Maximum allowed index difference between parent and child saddle points during hierarchical search.	int(positive)	Default: 1
SameJudgement (Optional)	Saddle point equivalence criterion. **float(positive)**: Threshold for 2-norm distance. **Callable[[np.ndarray, np.ndarray], bool]**: Custom comparison function.	**float**(positive) or **Callable[[np.ndarray, np.ndarray], bool]** (**np.ndarray** shape: **(d,1)**)	Default: 1e-3
InitialEigenVectors (Optional)	Initial guess for Hessian eigenvectors. **None**: Auto-initialize. **Manual input**: Column vectors must be orthonormal.	**None** or **np.ndarray** with shape **(d,k)** where **k** is **MaxIndex**.	Default: **None**
PerturbationMethod (Optional)	Statistical distribution for saddle point exploration. **uniform**: Uniform sampling. **gaussian**: Normally distributed displacements.	**str**	Default: **uniform**
PerturbationRadius (Optional)	Displacement magnitude for saddle perturbations.	**float(positive)**	Default: 1e-4
PerturbationNumber (Optional)	Number of directional probes per saddle point. **Note**: Actual probes=2×**PerturbationNumber** (bidirectional).	**int**(positive)	Default: 2
EigenCombination (Optional)	Strategy for eigenvector utilization in perturbations. **all**: High cost, exhaustive. **min**: Low cost, partial.	**str**	Default: **all**

Solver Parameters			
Parameter	**Description**	**Data Type**	**Default/Behavior**
InitialPoint (Required)	The starting coordinates for saddle point search.	**list** or **numpy.ndarray** (1D array)	Mandatory input
Tolerance (Optional)	Convergence threshold for saddle point iterations. Stop when $\|\|gradient\|\|_2$ < **Tolerance**.	**float**(positive)	Default: 1e-6
SearchArea (Optional)	Maximum exploration radius from initial point. Terminates if $\|\|x - x_0\|\|_2$ > **SearchArea**.	**float**(positive)	Default: 1e+3
TimeStep (Optional)	Temporal discretization interval for dynamics.	**float**(positive)	Default: 1e-4
MaxIter (Optional)	Maximum number of HiSD iterations permitted.	**int**(positive)	Default: 1000
SaveTrajectory (Optional)	Records full optimization path during computation.	**bool**	Default: **True**
Verbose (Optional)	Controls real-time progress reporting. Outputs iteration count and gradient norm.	**bool**	Default: **False**
ReportInterval (Optional)	Iteration frequency for console output when **Verbose=True**.	**int**(positive)	Default: 100
Eigen Parameters			
Parameter	**Description**	**Data Type**	**Default/Behavior**
EigenMethod (Optional)	Eigensolver selection for stability analysis. **lobpcg**(Gradient system): Locally Optimal Block PCG. **power**(Non-gradient system): Power Method. **euler**(Both): Explicit Euler Discretization.	**str**	Default: Gradient system: **lobpcg** Non-gradient system: **power**
EigenMaxIter (Optional)	Maximum iterations for eigenpair computation.	**int**(positive)	Default: 10
EigenStepSize (Optional)	Discretization step for Euler/power methods.	**float**(positive)	Default: 1e-7
PrecisionTol (Optional)	Precision tolerance for eigenvalues. (We treat eigenvalues as 0 if their absolute value is less than tolerance.)	**float**(non-negative)	Default: 1e-5
EigvecUnified (Optional)	Ensures eigenvectors corresponding to multiple eigenvalues are unified across different computational devices.	**bool**	Default: **False**
Hessian Parameters			
Parameter	**Description**	**Data Type**	**Default/Behavior**
ExactHessian (Optional)	Controls Hessian matrix computation method. Requires **Grad** as symbolic expressions.	**bool**	**True**: Analytical Hessian from symbolic gradient. **False**: (Default) Dimer-based approximation.
HessianDimerLength (Optional)	Displacement length for numerical Hessian-Vector product approximations.	**float**(positive)	Default: 1e-5
AccelerationParameters			
Parameter	**Description**	**Data Type**	**Default/Behavior**
BBStep (Optional)	Enables Barzilai-Borwein adaptive step sizing. **Trade-off**: May accelerate convergence but can cause instability in stiff systems.	**bool**	Default: **False**
Acceleration (Optional)	Convergence acceleration technique: **none**: No acceleration. **heavyball**: Momentum-based. **nesterov**: Nesterov acceleration.	**str**	Default: **none**
NesterovChoice (Optional)	Specifies the acceleration parameter sequence for Nesterov's method: **1**: refer to the equation (2.17)(default). **2**: refer to the equation (2.18).	**int**	Default: **1**
NesterovRestart (Optional)	Iteration interval for Nesterov momentum reset.	**int**(positive) or **None**	**None**: (Default) Disable restart. Integer **n**: Reset every **n** steps.
Momentum (Optional)	Momentum coefficient for heavy ball acceleration. Range: 0.0 ≤ Momentum < 1.0.	**float**(non-negative)	Default: 0.0 (Equivalent to no acceleration)