队伍编号	21090130009
题号	B

基于群智能和机器学习的三维团簇稳态结构预测研究

摘 要

原子团簇是由数个至数千个原子相互作用形成的相对稳定聚集体，其物理和化学性质由它所包含的原子数和原子空间结构决定，寻找原子团簇最优结构问题是计算机、物理和化学等研究领域的一个非确定性多项式求解（NP-Hard）难题。

本文主要介绍了金原子团簇和硼原子团簇的最优结构预测方法：

文章提出一种基于势能函数、群智能优化和机器学习的混合模型对原子团簇的稳态结构进行预测。首先，基于给定的Au_{20}和B_{45}^{-}团簇结构及其能量值，利用改进的粒子群优化算法（IPSO）优化势能模型的参数，使模型最大程度匹配所给能量值。其次，为满足机器学习模型的输入结构，建立库伦矩阵进行团簇结构信息的数据表征，并将转换后的数据与第一步势能计算结果同时放入机器学习集成算法（Stacking）和神经网络门控循环单元（GRU）中，利用所给Au_{20}和B_{45}^{-}原子团簇的能量值训练模型，并将训练好的模型分别用于后续预测 Au 团簇和 B^{-}团簇的最优结构。此外，考虑到已有文献表明高度对称的团簇结构具有较低的能量值，本文在第一题和第三题分别生成 2 种基于有限对称的初始团簇结构，并使用模拟退火算法（SA）逐步改变新解来使目标函数值（能量）减小，迭代找到全局最优结构。进一步地，在第二题和第三题生成 4 种基于有限对称和随机生成的初始团簇结构，并对 4 种结构类型的最优结构进行比较分析并验证其稳定性。结果表明，本文提出的新型混合算法具有良好的收敛特性，并且从团簇的三维结构图中可以得出，相比于其他 3 种结构类型，全局最优的团簇结构具有明显的对称特性，并且这种结构特性具有较高的稳定性。

关键词：群智能；机器学习；纳米团簇；结构预测

目 录

1 问题重述

1.1 研究背景

团簇（Cluster）这一名词最初是由 Cotton 于 1966 年提出，他认为团簇是具有金属-金属键的多核化合物[1]。目前关于团簇的定义和范围，多数学者认为，团簇由数个至数千个原子、分子或离子组成，其空间尺度为几至几百埃，通过物理或化学相互作用结合在一起相对稳定的微观或亚微观聚集体，其物理和化学性质随所含原子数而变化。

预测原子团簇最稳定结构的一般方法是合理简化原子团簇的内部结构，基于原子之间的相互作用势建立物理模型，将物理模型转换为数学模型，然后对数学模型进行求解。研究原子间的势能模型对预测原子团簇最稳态结构有很大的作用，其中应用比较广泛的势能模型有经验势、自洽势和紧束缚势等。研究原子团簇结构的第二步是在势能面上寻找能量最低的团簇结构，主要方法包括蒙特卡洛算法、分子动力学、模拟退火算法、遗传算法等。

1.2 研究问题

由于团簇势能曲面上局部极小值的数量非常庞大，且构型空间随着体系尺寸呈指数性增长，因此多元团簇的结构优化相当困难。传统的数值迭代求解薛定谔方程计算非常耗时，因此，需要对这种方法加以改进，如结合机器学习等方法，训练团簇结构和能量的关系，从而预测新型团簇的全局最优结构，有利于发现新型团簇材料的结构和性能。

鉴于上述背景，本文需要研究解决以下问题：

问题 1：针对金属团簇，给定 1000 个金团簇Au_{20}的结构，建立金团簇能量预测的数学模型，并预测金团簇Au_{20}的全局最优结构，描述形状；

问题 2：在问题 1 的基础上，设计算法，产生金团簇不同结构的异构体，自动搜索和预测金团簇Au_{32}的全局最优结构，并描述其几何形状，分析稳定性；

问题 3：针对非金属团簇，给定 3751 个硼团簇B_{45}^{-}的结构，建立硼团簇能量预测的数学模型，并预测硼团簇B_{45}^{-}的全局最优结构，描述形状；

问题 4：在问题 3 的基础上，设计算法，产生硼团簇不同结构的异构体，自动搜索和预测硼团簇B_{40}^{-}的全局最优结构，并描述其几何形状，分析稳定性。

2 问题分析

2.1 针对金团簇的问题分析

问题 1 和问题 2 都关于金属团簇，关于问题 1，赛题已经给出了Au_{20}的 1000 个结构信息和对应的能量信息，我们假设所给的能量信息是可以由金属势能函数计算得到的，而主流的计算金属团簇的函数公式会有一些未知参数，例如 Gupta 函数的 A，p，q，r0 等，本文尝试带入前人设定的值但与给定的能量值出现较大偏差，鉴于此，我们使用一种智能寻优的方法找到适合的参数值，不过考虑到预测结果对设定的参数值极为敏感，我们加入一种集成学习算法，训练团簇位置结构和刚刚预测得到的能量与真实值之间的关系，能够得到更稳定的预测结果。在预测最优结构上，由于我们之前已经训练好了预测模型，那么只需要通过生成团簇结构并带入到训练好的模型中就可以得到预测的能量，通过能量值的大小就可以反推出哪种结构是最优的， 在这个分析方法上，重要的是如何生成结构，参考前人文献，认为具有对称特性的团簇往往在能量上表现的更低，因此我们考虑生成 2 种基于对称性限制的初始构型并带入到预测模型中，在这个流程中，需要一种智能优化算法来迭代寻找最低的能量，由于模拟退火算法在连续优化问题上的较优表现，我们使用该方法来预测能量最低的团簇结构。对于问题 2，我们同样利用刚才训练好的模型来进行预测，同时为了验证其稳定性，我们在限制性条件假定下加入了随机生成的初始构型来做对比研究。

2.2 针对硼元素的问题分析

问题 3 和问题 4 都关于非金属团簇，关于问题 3，首先对硼团簇（B_{45}^{-}）进行能量预测数学模型的构建，本文构想使用一种新型 Tersoff-IPSO-GRU 算法：建立传统的 Tersoff 势能模型，同样利用粒子群优化算法（IPSO）找出势能模型的 2 项最优参数，并将 3751 个已知数据的位置向量代入此 Tersoff 方程求解，将计算结果作为一个变量；另外，原始数据经过库伦矩阵的计算，将特征值与前者结果放入门控循环单元模型（GRU）预测各团簇的能量。其次，运用模拟退火算法，预测B_{45}^{-}的全局最优结构，描述其形状，并对其稳定性进行验证分析。而问题 4 在一开始首先需随机生成不同结构的B_{40}^{-}形状，后续与问题 3 同理构建使用基于深度学习的自动搜索及预测全局最优结构的算法。

3 模型假设

1. 假设Au_{20}和Au_{32}计算势能公式 Gupta 的参数相同；

2. 假设B_{40}^-和B_{45}^-计算势能公式 Gupta 的参数相同；

3. 假设Au_{20}、Au_{32}、B_{40}^-、B_{45}^-中原子在空间中呈现有限对称结构；

4. 假设Au_{20}、Au_{32}、B_{40}^-、B_{45}^-原子团簇的中心点位于笛卡尔坐标系原点处。

4 符号定义与评价指标

4.1 符号定义

表 4-1 符号定义与说明

符号	符号说明
V, E_b	势能
r_{ij}	两原子之间的距离
A_{ij}	斥力对项的系数
ξ_{ij}	i 和 j 原子距离的跃迁积分
p_{ij}	原子间距离的相互排斥项
q_{ij}	原子间距离的相互吸引项
w	惯性因子
c_1、c_2	加速因子
T_i	迭代次数
α	动量因子
b_{ij}^{π}	修正项

4.2 评价指标

MSE（Mean Square Error）均方误差，是真实值与预测值的差值的平方然后求和平均。MSE 也常被用做线性回归的损失函数。数学表达式如下：

$$MSE=\frac{1}{m}\sum_{i=1}^{m}(y_i-\hat{y}_i)^2, \quad (1)$$

MAE（Mean Absolute Error）平均绝对误差是绝对误差的平均值，可以更好的反映预测值误差的实际情况。数学表达式如下：

$$MAE(\mathrm{X},\ \mathrm{h}) = \frac{1}{m}\sum_{i=1}^{m}\left|h(x_i)-y_i\right|, \tag{2}$$

MAPE(Mean Absolute Percentage Error)平均绝对百分比误差,其范围为[0,+∞),MAPE 为 0%表示完美模型，MAPE 大于 100%表示劣质模型。数学表达式如下：

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right|, \tag{3}$$

5 问题一的模型建立、求解与验证

关于问题 1，首先对金团簇（Au_{20}）进行能量预测数学模型的构建，本文构想使用一种新型 Gupta-IPSO-Stacking 算法：首先，建立传统的 Gupta 势能模型，利用改进的粒子群优化算法（IPSO）找出势能模型的 5 项最优参数，并将 1000 个已知数据的位置向量代入此 Gupta 方程求解，将计算结果作为一个变量；其次，为满足机器学习的输入结构，将团簇的位置信息进行库伦矩阵转换，将特征值与势能方程结果同时放入机器学习集成算法（Stacking）预测各团簇的能量。此外，运用模拟退火算法（SA）预测Au_{20}的全局最优结构，描述其形状，对其稳定性进行分析。

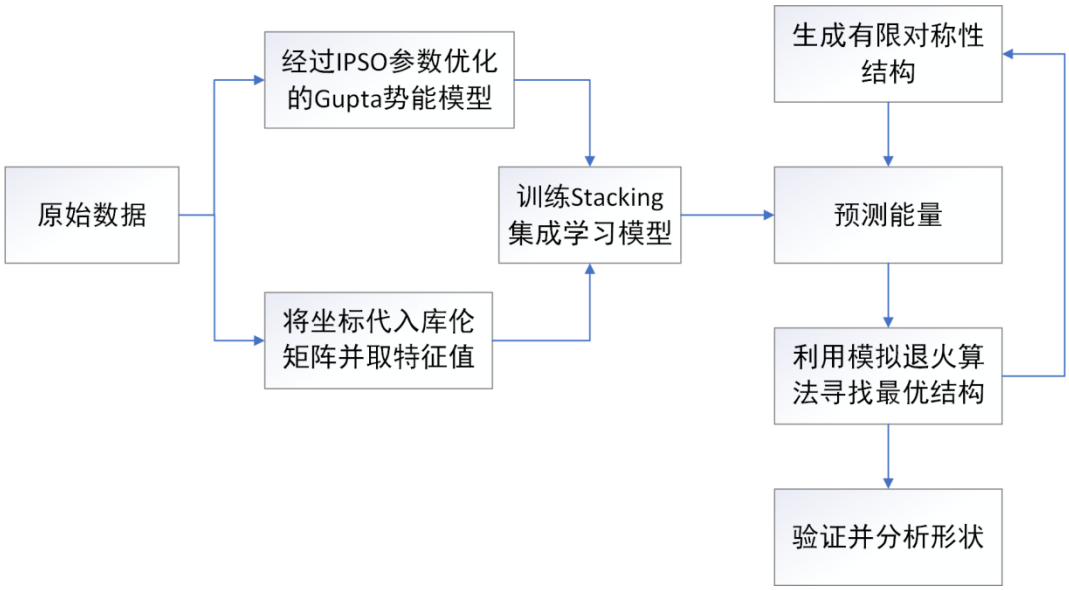


图 5-1 关于金团簇Au_{20}全局最优结构的思路流程图

5.1 Gupta-IPSO-Stacking 能量预测模型

5.1.1 Gupta 势能函数

近年来一系列精度不同的相互作用势被提出，如第一原理多体势、应用广泛的 Morse 势、Tersoff 势、EAM 势、Gupta 势等。不同的原子适用于不同的传统势能模型，其中，与对势仅考虑两两原子间相互作用不同，多体势(Mutli-body Potential)理论认为：在多原子系统中，一个原子的位置改变，必将影响到整体系统中一定范围的电子云分布，从而影响到其他原子间的相互作用，故多原子金属体系的势函数用多体势 Gupta 模型计算更为准确。

Gupta 势能函数常被用于描述二元、三元及多元金属团簇原子之间的相互作用，Gupta 势能函数由排斥项$\mathrm{V^r(i)}$和吸引项$\mathrm{V^m(i)}$两项组成，对于总原子的相互作用势$\mathrm{V_n}$可表述为：

总势能

$$\mathrm{V_n} = \sum_{\mathrm{i\neq 1}}^{\mathrm{n}}(\mathrm{V^r(i)} - \mathrm{V^m(i)}), \tag{4}$$

排斥势

$$\mathrm{V^r(i)} = \sum_{\mathrm{i\neq 1(j\neq 1)}}^{\mathrm{n}} \mathrm{A_{ij}}\exp(-\mathrm{p_{ij}}(\frac{\mathrm{r_{ij}}}{\mathrm{r_{ij}^{(0)}}} - 1)), \tag{5}$$

吸引势：

$$\mathrm{V^r(i)} = \sum_{\mathrm{i\neq 1(j\neq 1)}}^{\mathrm{n}}[\xi_{\mathrm{ij}}^2\exp(-2\mathrm{q_{ij}}(\frac{\mathrm{r_{ij}}}{\mathrm{r_{ij}^{(0)}}} - 1))]^{\frac{1}{2}}, \tag{6}$$

其中，$\mathrm{r_{ij}}$为体系中两原子之间的距离，$\mathrm{r_{ij}^{(0)}}$为平衡的第一近邻距离。$\mathrm{A_{ij}}$表示斥力对项的系数，ξ_{ij}表示 i 和 j 原子距离的跃迁积分，$\mathrm{p_{ij}}$和$\mathrm{q_{ij}}$分别代表原子间距离的相互排斥项和相互吸引项。

5.1.2 改进粒子群优化算法（IPS0）参数调优

由 Kennedy(1995)提出的 PSO 算法是基于粒子迭代寻找解空间的最优值的一种全局动态寻优方法，通过在每次迭代过程中追寻全局极值P_{gbest}和个体极值P_{ibest}来不断调整粒子的位置和速度，并在迭代时向着全局最优的粒子移动，迭代完成后可以趋近于解空间范围内的最优解。迭代的公式如下：

$$V_i^{t+1} = w_i V_i^t + c_1 r_1 (P_{ibest}^t - X_i^t) + c_2 r_2 (P_{gbest}^t - X_i^t), \tag{7}$$

$$X_i^{t+1} = X_i^t + V_i^{t+1}, \tag{8}$$

式(7)为速度更新公式，式(8)为位置更新公式，w为惯性因子，c_1和c_2为加速因子，一般为常数；r_1和r_2为 0 到 1 之间的随机数，基于 PSO 的原理，w惯性因子会随着迭代的增加而逐步减小，使其能够逐渐趋向于最优解，w的变化公式如下：

$$w = \frac{T_{\max} - T_i}{T_{\max}}(w_{\max} - w_{\min}) + w, \tag{9}$$

其中，$w_{\max}$、$w_{\min}$分别为最大和最小惯性权重，$T_{\max}$、T_i为最大迭代次数和当前迭代次数。

这种基本的 PSO 算法存在一个明显的缺陷，算法容易陷入到局部最优解。而已有自适应惯性权重法提出，通过余弦函数改变惯性权重在迭代前期以及迭代后期的数值以避免过早陷入局部最优解，但是这样牺牲了后期迭代的速度。为了避免该问题，本文使用了一种增加动量项的办法，能够使后期粒子收敛的速度加快，加入动量项的公式如下：

$$V_i^{t+1} = w_i V_i^t + \triangle v_i^t + \alpha v_i^{t-1}, \tag{10}$$

上式中，$\alpha \triangle v_i^{t-1}$为新引入的动量项，$\alpha$为动量因子，取值范围为$[-1, 1]$。加入该动量项后可以平滑算法当前的速度，若当前时刻的$\triangle v_i^t$与前一时刻的$\triangle v_i^{t-1}$同号，则引入动量可以加快提高粒子速度从而加快收敛；若当前时刻的$\triangle v_i^t$与前一时刻的$\triangle v_i^{t-1}$异号，说明算法存在一定的动荡，则加入动量项可以起到缓解动荡从而加快收敛的目的。接下来采用余弦函数来控制惯性权重的非线性变化，避免粒子过早地陷入局部最优解：

$$w' = [(w_{\max} - w_{\min})/2]\cos(\pi T_i / T_{\max}) + (w_{\max} + w_{\min})/2, \tag{11}$$

改进后的速度和位置公式如下：

$$V_i^{i+1} = w' V_i^t + \triangle v_i^t + \alpha \triangle v_i^{t-1}, \tag{12}$$

$$X_i^{t+1} = X_i^t + V_i^{t+1}, \tag{13}$$

其中，$w_{\max}$为 0.9，是惯性权重的最大值也是初始惯性权重值，$w_{\min}$为 0.4，是惯性权重的最小值。利用余弦函数可以解决惯性权重容易过早的使算法陷入到局部最优解的问题，同时加入动量项来解决粒子迭代速度变慢的问题。

本文基于自适应权重的粒子群优化参数的具体步骤如下：

Step1：初始化 PSO 算法的参数，包括 A，p，q，epilson，r0，同时随机初始化 Gupta 模型的 5 个参数在解空间的位置以及粒子的初始速度和位置，设置粒子最大速度为 V_{max} 以及 Gupta 模型 5 个参数的取值范围。

Step2：根据式(7)计算出第一次迭代时各个粒子的适应度值，将适应度值最小的粒子的 P_{ibest} 作为全局最优的 P_{gbest}。

Step3：根据式(10)确定粒子新的速度以及新的位置，并根据粒子位置确定新的适应度值。

Step4：通过不断迭代找到最优的适应度值，并将对应的 5 个参数输出。

因此，本文使用的是改进的 PSO 算法，选取的参数值如下：[A=0.06821, p=9.8346, q=2.3542, epilson=1.7903, r0=3.214]Gupta 模型的最优参数损失收敛情况如下。

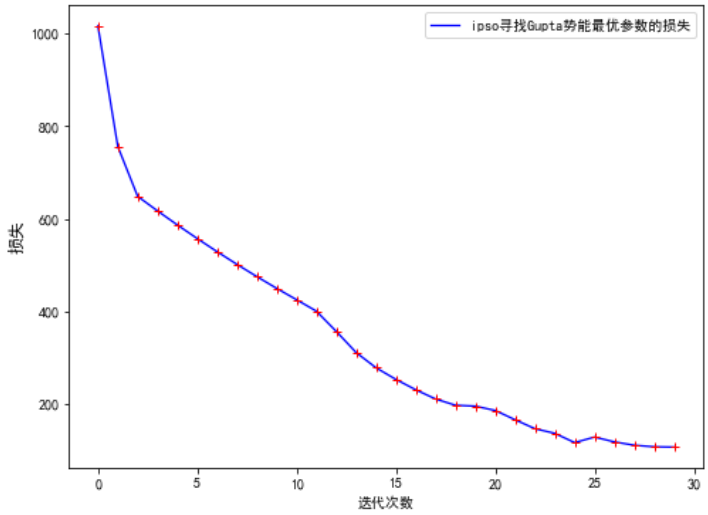


图 5-2 IPSO 寻找 Gupta 势能最优参数的损失图

5.1.3 库伦矩阵数据表征

对于一个由 20 个原子组成的团簇结构，存在函数 $E^{(N)}$：$R^{3N} \times Z \rightarrow R$，根据结构中原子的三维坐标以及对应的原子数目 Z 计算出该团簇结构的总能量，即存在 3N + N 维结构信息到一维能量信息的映射。若要在该结构预测过程中引入机器学习方法，就需要一种描述方法把随机生成的结构信息表征成数值向量的模式（机器学习方法的输入通常是向量），同时还要保证信息不能丢失，因此表征方法的选择对机器学习方法的性能表现十分重要。本文利用 Rupp 等[3]提出的库伦矩阵（Coulomb matrix）方法，该方法能把由团簇结构的原子坐标信息和原子核电荷数构成的 4N 维数据转化成 N × N 维的数值矩

阵，其中 N 为团簇中的原子个数。

$$C=\begin{cases}0.5Z_i^{2.4} & ,\forall i=j\\ \frac{Z_iZ_j}{|R_i-R_j|} & ,\forall i\neq j\end{cases}, \tag{14}$$

第 999 个金团簇的库伦矩阵表征结果如图 5-3 所示，其中 R_i为第 i 个原子的坐标向量，Z_i为第 i 个原子的核电荷，金是元素周期表中第 79 个元素，核电荷数量为 79，因此本研究中Z_i的取值均为 79。为满足机器学习算法的向量输入需求，继续对得到的库伦矩阵做降维处理，通过提取矩阵的特征值把 N×N 维的库伦矩阵降维成为 N×1 维的特征向量，并对特征值排序，特征值提取过程如图 5-4 所示。当库伦矩阵为稀疏矩阵时，可以只取前 n 个较大的特征值，以此来降低计算复杂度。

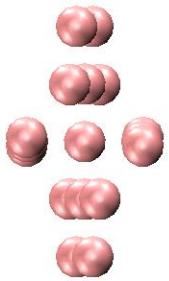


图 5-3 第 999 个金团簇

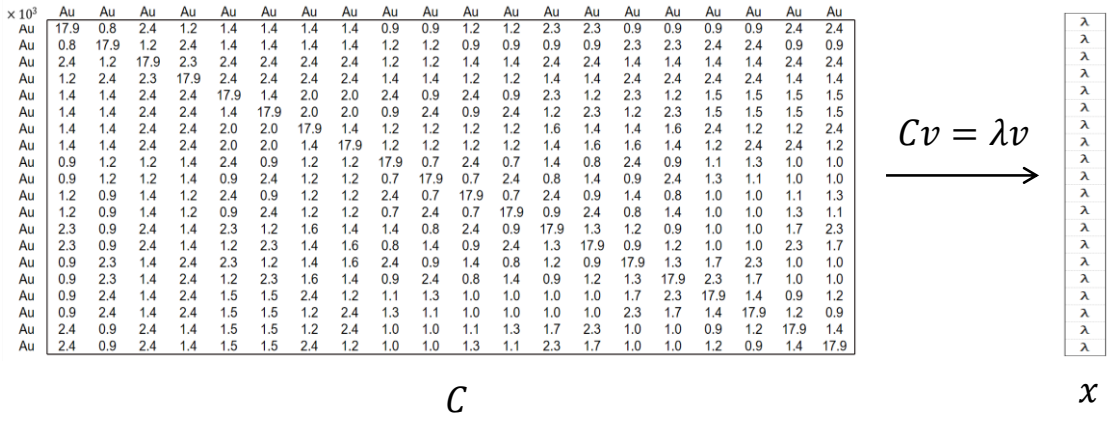


图 5-4 第 999 个金团簇的库伦矩阵计算及特征值提取结果

本文中样例原子数目都为 20，因此得到的特征向量的维度相同，机器学习算法要求输入维度是一致的，我们会得到 20×1 维的特征向量，对于 Au32 而言，存在 32 个输入数据，我们考虑将后续 12 个特征删除来进行预测，因为特征值越靠前其保留的信息越重要，并且这种重要性呈指数级递减形式。

5.1.4 集成算法（Stacking）预测能量

集成学习是指将一个问题分解到多种不同的方法中，连结多个学习器来完成学习任务。集成学习包括：序列化方法和并行化方法两种。前者指学习器之间存在较强的依赖关系，需要串行生成。比如 Boosting[4]；后者指学习器之间没有强依赖关系，可以同时生成。比如 RF[5]。下文对 RF 算法、XGBoost 和 Boosting 做简要介绍。

RF 算法是一种基于决策树的组合分类器[6]，其主要思想是：第一步从训练数据集中运用自主抽样法（Bootstrap Method，Bootstrapping）提取多个样本以创建 N 个训练子集；第二步对提取出的每个样本建模生成 N 棵决策树进而形成“森林”；最后将这些决策树进行组合并采用投票的方式获得最终分类或预测的结果。RF 算法中随机性的引入克服了单一决策模型容易过拟合的问题。

XGBoost 是 Boosting 族中的一种提升树模型，有训练速度快、能有效处理大规模数据等优点。XGBoost 的主要思想是集成多种弱分类器从而形成强分类器，不断拟合上一棵树的残差来产生新树，每加入一棵决策树，模型整体的性能都必须有所提升，直到性能不再提升或下降时结束算法。XGBoost 的决策函数为其中，$\mathrm{f_k}(x_i)$是第 k 棵树在x_i节点的预测值，XGBoost 在预测某个样本的分数时，会根据样本的不同特征在每棵树中映射到相应的叶子节点，该样本的预测值即为每棵树对应叶节点的得分之和。其中，$\mathrm{f_k}(x_i)$是第 k 棵树在x_i节点的预测值，XGBoost 在预测某个样本的分数时，会根据样本的不同特征在每棵树中映射到相应的叶子节点，该样本的预测值即为每棵树对应叶节点的得分之和。

Adaboost[7]是一种经典的提升算法，运用单层决策作为基分类器。该算法采用迭代的思想，每次迭代仅训练一个基分类器，在训练的过程中把预测结果中分类错误的样本权重提高，分类正确的样本权重降低，促使后续的训练过程中基分类器更加关注被错分的样本，训练完成的基分类器会加入下一轮的迭代，直到迭代次数达到预先设置的最大值或错误率充分小时才能确定最终的强分类器。

支持向量回归（SVR）是使用支持向量机模型（SVM）处理连续变量的回归问题，该算法原理为找到一个回归平面，使得所有特征变量数据到该平面的欧氏距离最近。SVR 设置了一个真实值的误差范围，并且认为只要预测的值在该范围之内便认为预测正确，仅对误差范围外的预测值计算损失。因此，SVR 通过最大化误差范围和最小化损失值双重手段来优化模型，但是由于在现实任务中，误差范围往往难以确定，SVR 会加入一个松弛变量，使得该范围具有弹性。SVR 模型的预测容错度相对较高。

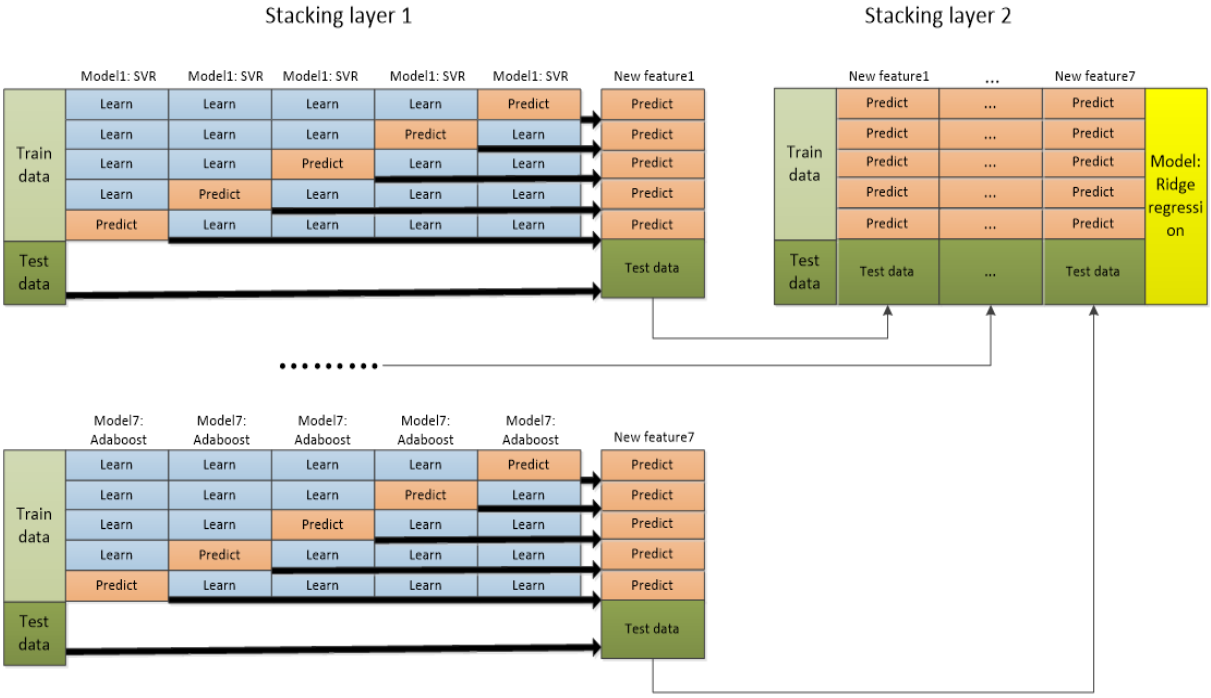


图 5-5 Stacking 预测过程图

本文使用 5 折交叉验证 Stacking 集成模型进行模型训练与预测的过程如图 5-5 所示。首先对数据集打乱顺序进行训练集与测试集划分，将训练集进行 5 折交叉验证并输入多个模型分别进行训练，训练完成后将这些模型作为基层模型，然后在其基础上建立学习器，再学习器以每个模型的预测结果作为特征，通过多次迭代训练分别给各个特征赋予不同的权重，以提高最后预测结果的精准度。

图 5-6 和图 5-7 展示了 Stacking 集成模型以及各个子模型的预测结果对应的评价指标的对比。

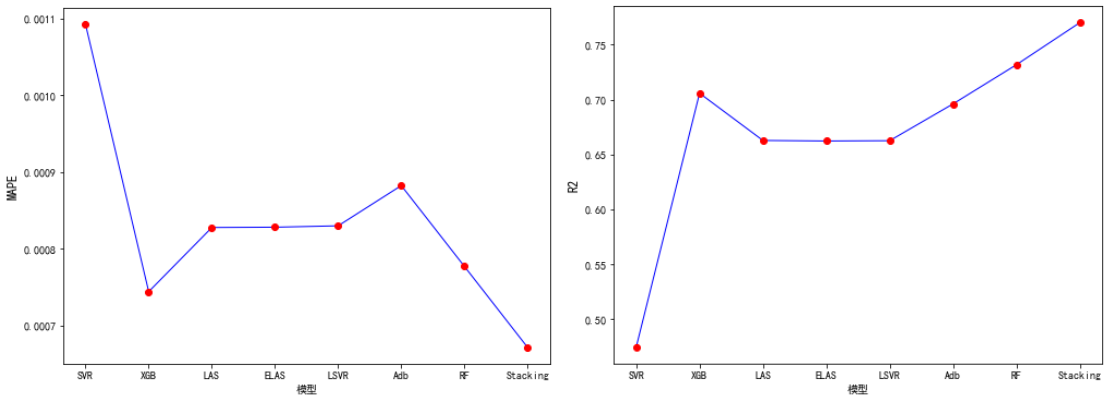


图 5-6 Stacking 模型预测能量结果 MAPE、R2 对比图

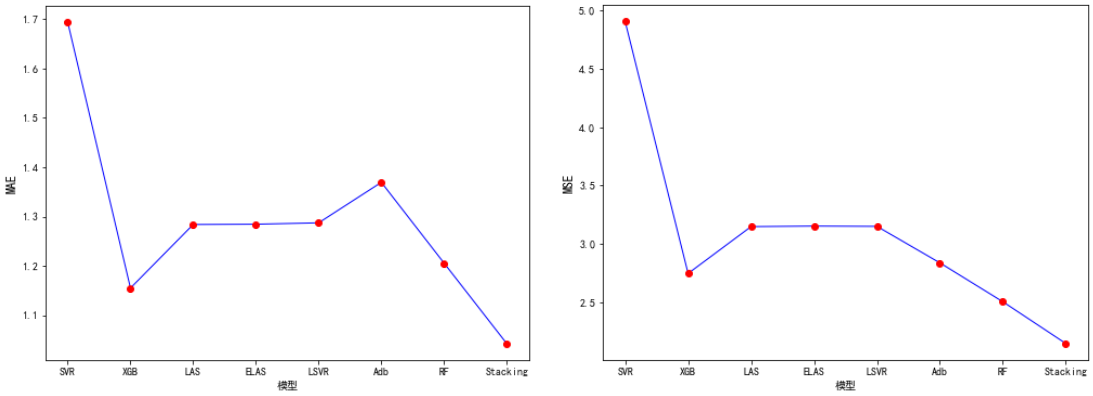


图 5-7 Stacking 模型预测能量结果 MAE、MSE 对比图

从图 5-6 和图 5-7 可以明显看出，在 Stacking 集成模型内的各个子模型的预测结果中，SVR 模型的预测结果表现相对最差，其对应的 MAPE、MAE 和 MSE 值均远大于其他模型，R^2 远小于其他模型；XGB 模型和 LAS 模型等其他子模型的预测性能相当，其对应的多项评价指标的值相差较小。Stacking 集成模型预测性能最好，各项评价指标均为最佳。集成模型预测结果对应的 MAPE 值为 0.000635，小于该指标表现次优的 XGB 模型（0.000731），集成模型的预测精度较子模型的平均精度（0.000852）提高 25.47%；集成模型预测结果对应的 R^2 为 0.767，大于该指标表现次优的 XGB 模型（0.712），集成模型的预测精度较子模型的平均精度（0.621）提高 23.51%。因此，借助于直观图示和精确的数值计算，说明了本文使用的 Stacking 集成模型的预测性能是最优的。

5.2 模拟退火算法预测全局最优结构

5.2.1 模拟退火算法

模拟退火算法（Simulate Anneal，SA）是一种群智能通用概率演算法，用来在一个大的搜寻空间内找寻命题的最优解。模拟退火是由 S.Kirkpatrick，C.D.Gelatt 和 M.P.Vecchi 在 1983 年所发明的，该算法的出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。

模拟退火算法新解的产生和接受可分为如下四个步骤：

第一步是由一个产生函数从当前解产生一个位于解空间的新解；为便于后续的计算和接受，减少算法耗时，通常选择由当前新解经过简单地变换即可产生新解的方法，如对构成新解的全部或部分元素进行置换、互换等。

第二步是计算与新解所对应的目标函数差。因为目标函数差仅由变换部分产生，所以目标函数差的计算最好按增量计算。事实表明，对大多数应用而言，这是计算目标函

数差的最快方法。

第三步是判断新解是否被接受，判断的依据是一个接受准则，最常用的接受准则是 Metropolis 准则。

第四步是当新解被确定接受时，用新解代替当前解，这只需将当前解中对应于产生新解时的变换部分予以实现，同时修正目标函数值即可。此时，当前解实现了一次迭代。可在此基础上开始下一轮试验。而当新解被判定为舍弃时，则在原当前解的基础上继续下一轮试验。

模拟退火算法与初始值无关，算法求得的解与初始解状态 S(算法迭代的起点) 无关；且具有渐近收敛性，已在理论上被证明是一种以概率收敛于全局最优解的全局优化算法；同时，模拟退火算法还具有并行性。

5.2.2 限制性对称初始结构生成

Kryachko & Remacle[16]和 Philipp 等[17]的研究表明，金原子团簇的理论最稳态结构为对称结构，但是现实中往往会存在一些误差因素，导致理论与现实之间存在一定偏差，这种状态可以用有限对称来描述。有限对称是一种逼近对称的非对称状态，广泛存在于自然界中。

本文考虑现实误差因素，将生成的随机结构设定为限制性对称结构。限制性对称是通过对原子坐标添加噪声项实现的，本文首先随机生成 10 个点，然后产生这 10 个点的对称点，再对对称点的坐标添加噪声，具体坐标设置如表 5-1 所示。

表 5-1 基于限制性对称的金原子团簇结构坐标生成

原子序号	原子坐标	原子序号	原子坐标	原子序号	原子坐标
1	(x1,y1,z1)	8	(-x4,-y4,z4+ξ)	15	(x8,y8,z8)
2	(-x1,-y1,z1+ξ)	9	(x5,y5,z5)	16	(-x8,-y8,z8+ξ)
3	(x2,y2,z2)	10	(-x5,-y5,z5+ξ)	17	(x9,y9,z9)
4	(-x2,-y2,z2+ξ)	11	(x6,y6,z6)	18	(-x9,-y9,z9+ξ)
5	(x3,y3,z3)	12	(-x6,-y6,z6+ξ)	19	(x10,y10,z10)
6	(-x3,-y3,z3+ξ)	13	(x7,y7,z7)	20	(-x10,-y10,z10+ξ)
7	(x4,y4,z4)	14	(-x7,-y7,z7+ξ)		

注：ξ为趋近于 0 的实数。

5.2.3 结构初始范围设定

金原子团簇由许多原子组成，这些原子之间存在着相互作用力，团簇势能由原子间相互作用的势能决定。图 5-8 为原子势能与原子间距的函数关系图，由图可知当原子间距离 r 为 $2^{1/6}$ 时，势能值最小。根据 Gupta 势能计算公式可以计算出该最小值。

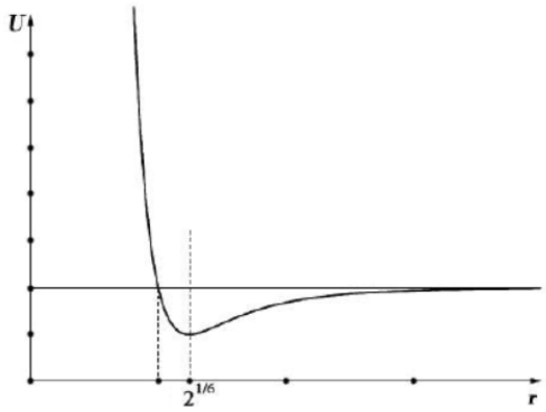


图 5-8 原子势能与原子间距函数关系图

根据高斯猜想，在一个三维空间内，当四个大小相同的球体为最紧密布局状态时，各个球的球心两两相连应该构成一个正四面体，此时他们对空间的占有密度为 $\rho=\frac{\sqrt{2}\pi}{6}\approx 0.74048$。

从物理学角度来看，势能值越小，对应的系统越稳定，因此对于金原子团簇来说，要想找到其最稳定结构，即求金原子团簇势能的最小值所对应的解。设想这样一种情况：金原子为具有一定直径的无重量小球，团簇内各个金原子两两相切，构成了一个最紧密布局状态，当小球直径为 $2^{1/6}$ 时，金原子团簇的整体势能最小，也即达到了最稳定状态。根据此猜想，原子个数为 N 时，这 N 个原子的空间结构将被确定在一个半径为 R 的球体内，R 可以借助于球体体积公式计算得出，如公式（11）所示。

$$R=\left(\frac{N}{\rho}\right)^{\frac{1}{3}}\times\frac{2^{\frac{1}{6}}}{2}\approx 0.62\times N^{\frac{1}{3}}, \qquad (15)$$

在本文的条件里，N 为 20，计算可得 R 为 1.6829。因此，本文将在一个半径为 1.6829 的球体范围内随机生成 20 个具有限制性对称特征的点作为金原子团簇的初始结构。

限制金原子团簇生成的范围可能会导致局部最优现象。本文在对所给实验数据进行描述性统计分析后发现，金原子团簇中原子坐标点的 x 值最小为-8.449219，最大为 7.461536，y 值最小为-8.716752，最大为 9.713382，z 值最小为-9.433710，最大为 9.242134。

x、y 和 z 的最大值和最小值确定了一个三维空间，因此，本文将此空间设定为全局范围，并在此空间内生成限制性对称 Au 原子团簇结构。表 5-2 展示了对所给实验数据进行描述性统计分析的结果。

表 5-2 Au_{20}原子团簇实验数据的描述性统计

	x	y	z
数量	19980	19980	19980
均值	-0.001299	-1.601602e-11	0.011941
标准差	2.234019	2.139016	2.217112
最小值	-8.449219	-8.716752	-9.433710
25%	-1.378895	-1.322093	-1.345737
50%	0.000000	0.000000	0.000000
75%	1.378856	1.321483	1.378422
最大值	7.461536	9.713382	9.242134

注：表中阴影部分为 x、y 和 z 的最大值和最小值。

综上所述，在此节本文共生成了两种初始结构，分别为由 R=1.6829 确定范围的限制性对称结构和由所给实验数据确定范围的限制性对称结构。

5.2.4 全局最优结构预测结果

本文 5.1.4 部分对 Stacking 集成预测模型进行训练并借助于评价指标证明了其良好的预测性能。在此部分我们将随机产生的两类限制性对称结构坐标数据进行库伦矩阵转换并输入 Stacking 集成模型预测对应的能量值，预测结果如图 5-9、5-10、5-11 和 5-12 所示。

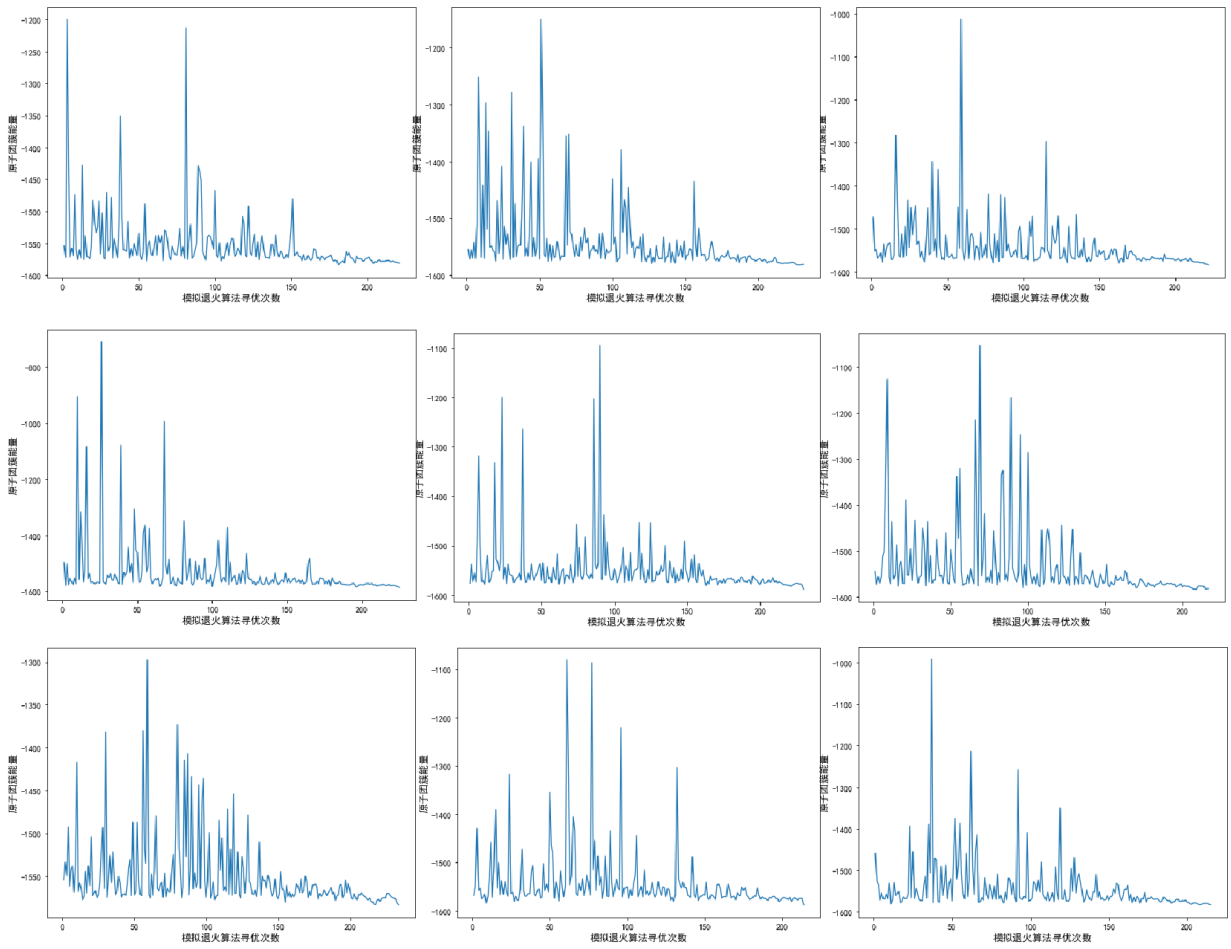


图 5-9 由 R 确定范围的限制性对称结构模拟退火算法寻优图

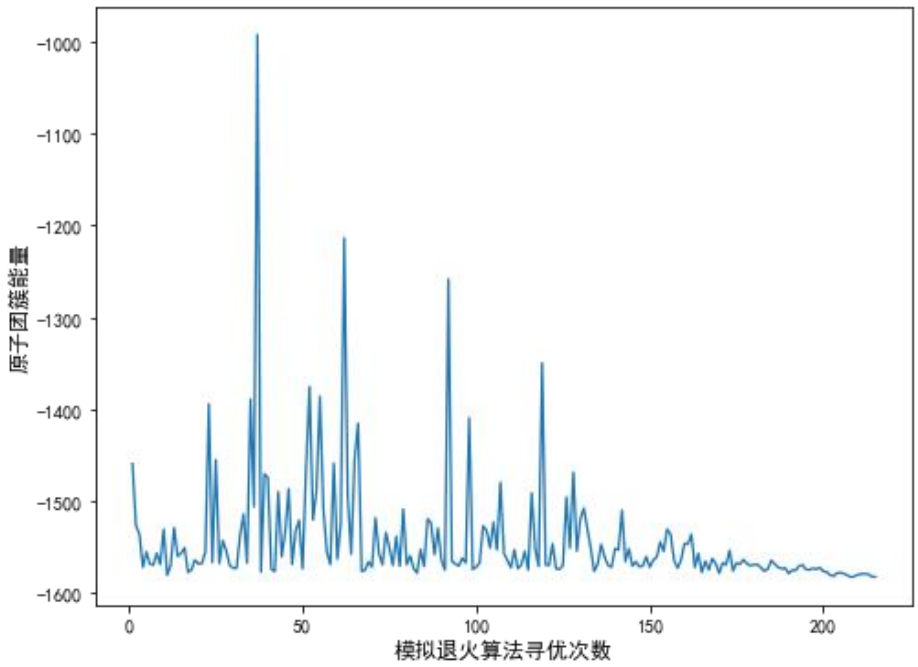
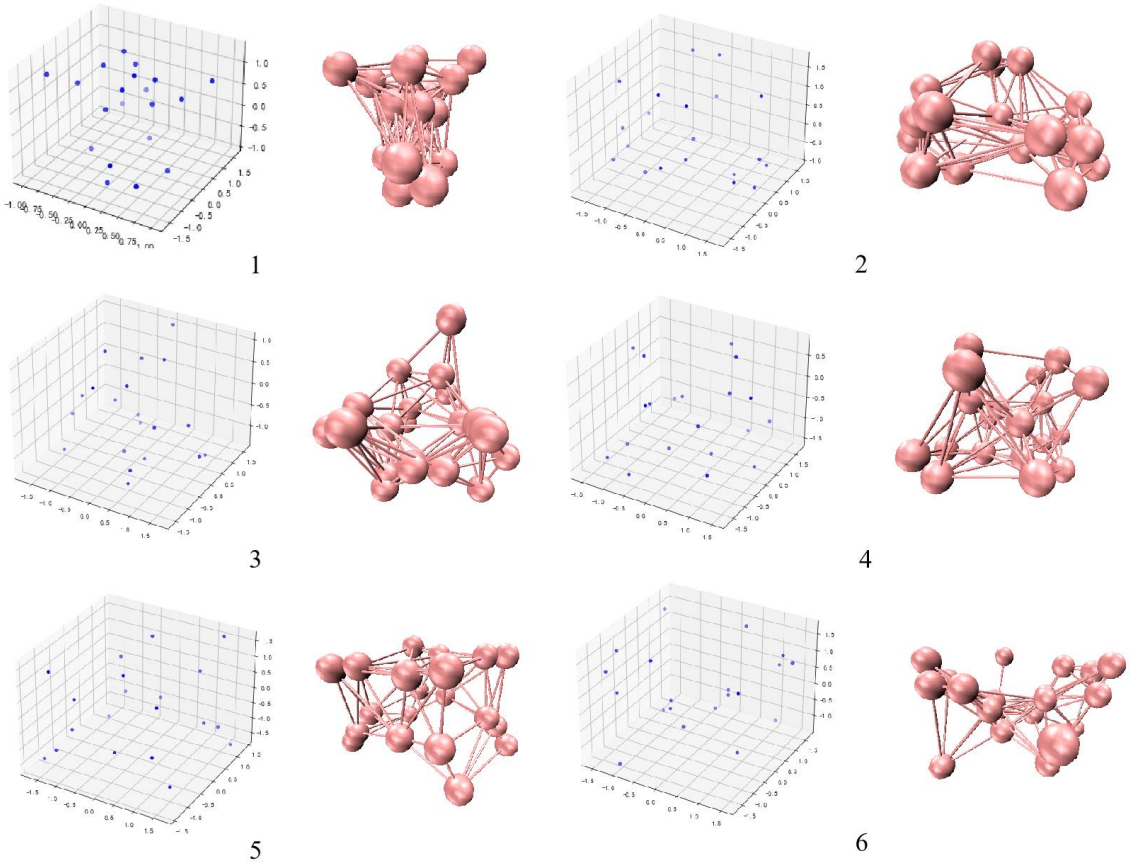


图 5-10 由 R 确定范围的限制性对称结构模拟退火算法最优图

本文将模拟退火寻优算法的迭代次数设置为 3000，每一次寻优结束都会输出一个相对最优结构，一次寻优过程如图 5-10 所示。由于是随机产生结构，一次算法寻优找到的可能是局部最优解，为了降低出现局部最优解的概率，本文在此进行 9 次寻优，如图 5-9 所示，9 次寻优过程中模拟退火算法共进行了 27000 次迭代。

对图 5-9 中寻优结果进行对比可以发现，在每一次寻优过程后期，随机结构的能量都趋向于向最优值收敛，即产生的随机结构接近于最稳定结构。图 5-10 为 9 次寻优过程结束确定的相对稳定结构中的最优结构，其对应的能量值为-1598.8573653797。



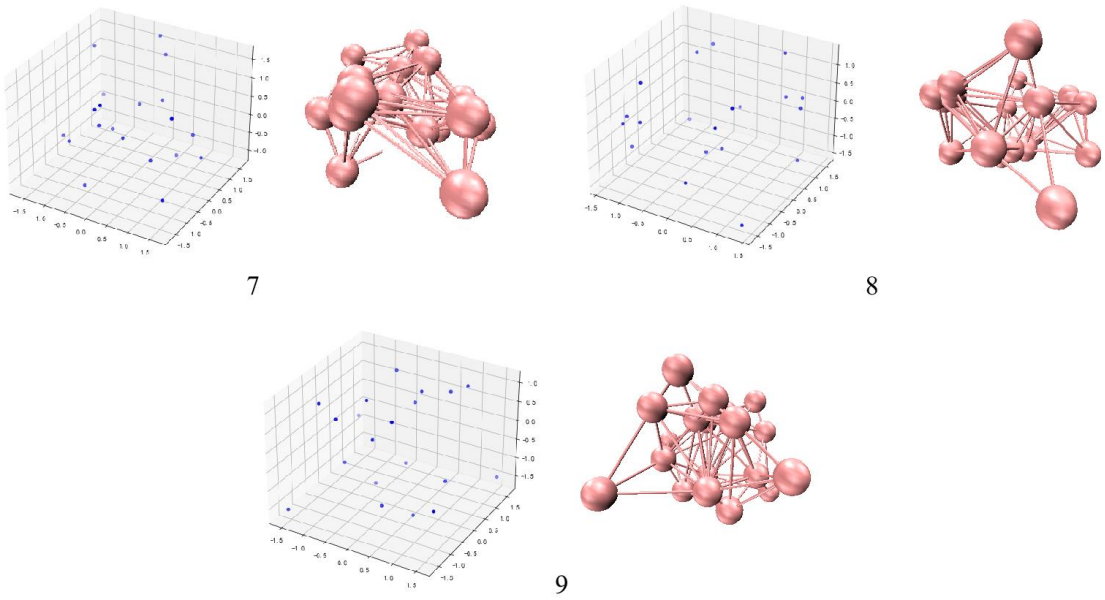


图 5-11 9 次寻优过程确定的相对稳定结构示意图

图 5-11 展示了 9 次模拟退火算法寻优结束确定的相对稳定结构的三维坐标图与结构示意图，Au_{20}团簇结构（1）-（9）对应的能量值如表 5-3 所示。

表 5-3 9 次寻优过程确定的相对稳定结构的能量

结构编号	能量	结构编号	能量
1	-1593.72819919776	6	-1598.25760712548
2	-1596.83596273478	7	-1594.01624620779
3	-1592.37190803227	8	-1591.4223318793
4	-1594.2009561987	9	-1598.8573653797
5	-1597.73322460756		

对表 5-3 中的数据进行对比分析发现，每一次寻优过程确定的相对最稳定结构的能量值差别较小，这与图 5-9 的结果是一致的。因此本文将这 9 个相对稳定结构中的最优结构作为全局最优结构，即图 5-11 中的结构 9。

图 5-12、5-13 和 5-14 展示了由所给实验数据确定的范围内生成的限制性对称结构的多次模拟退火算法寻优过程以及每一次寻优所确定的相对稳态结构。

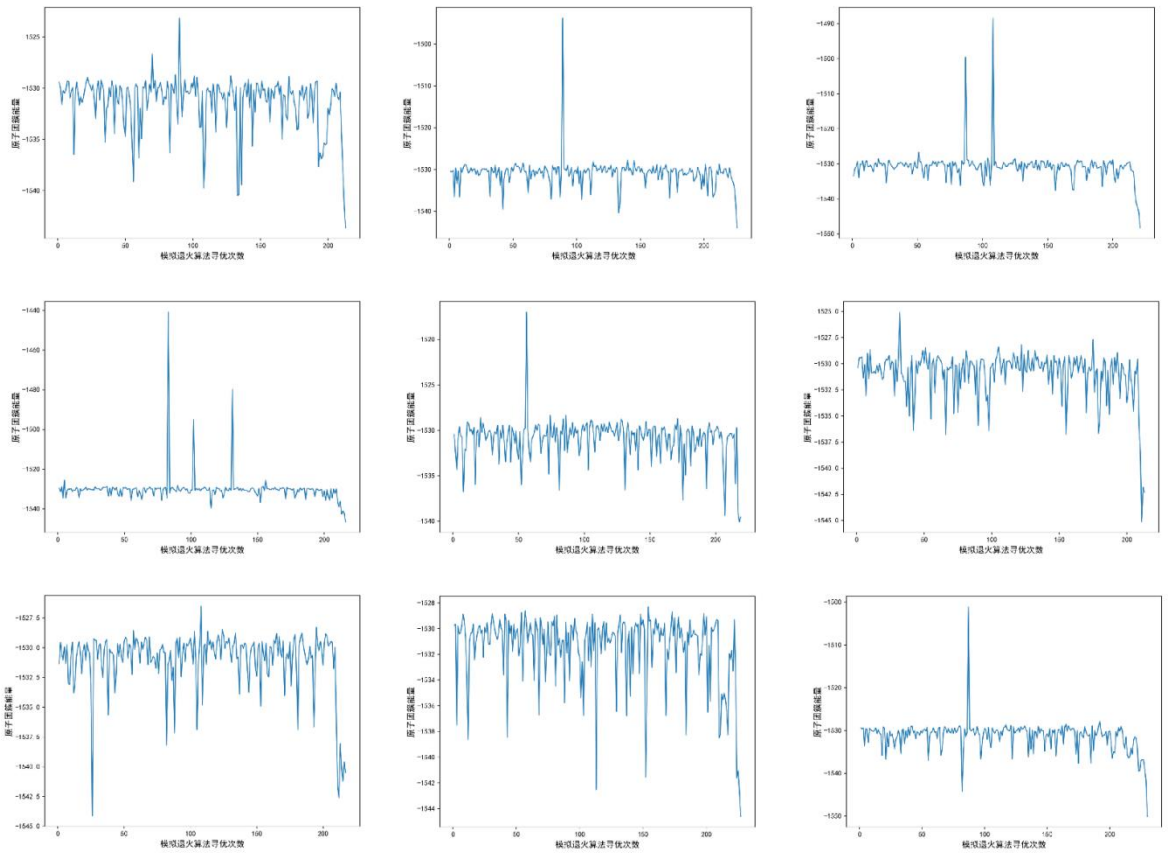


图 5-12 由所给实验数据确定范围的限制性对称结构模拟退火算法寻优图

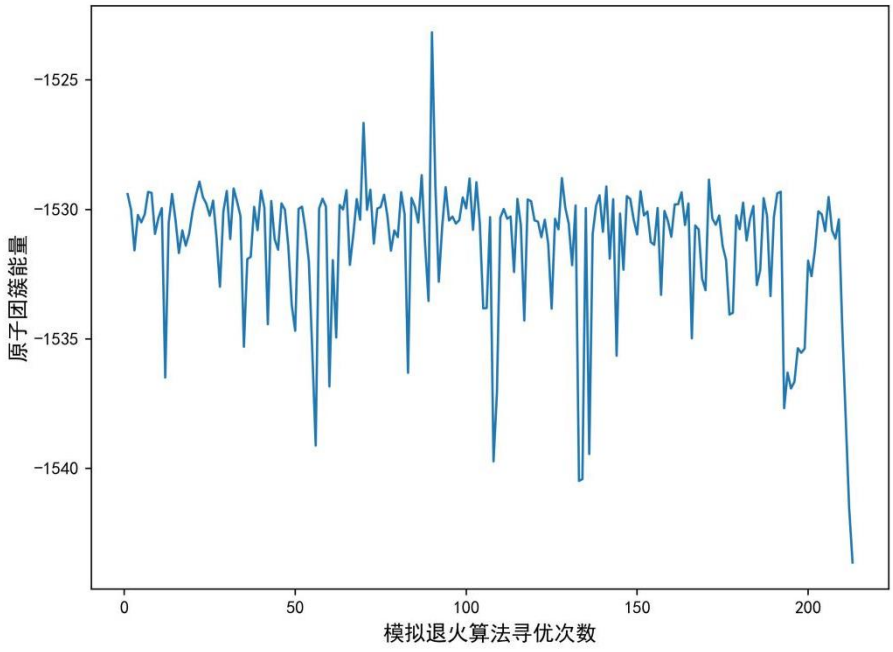


图 5-13 由所给实验数据确定范围的限制性对称结构模拟退火算法最优图

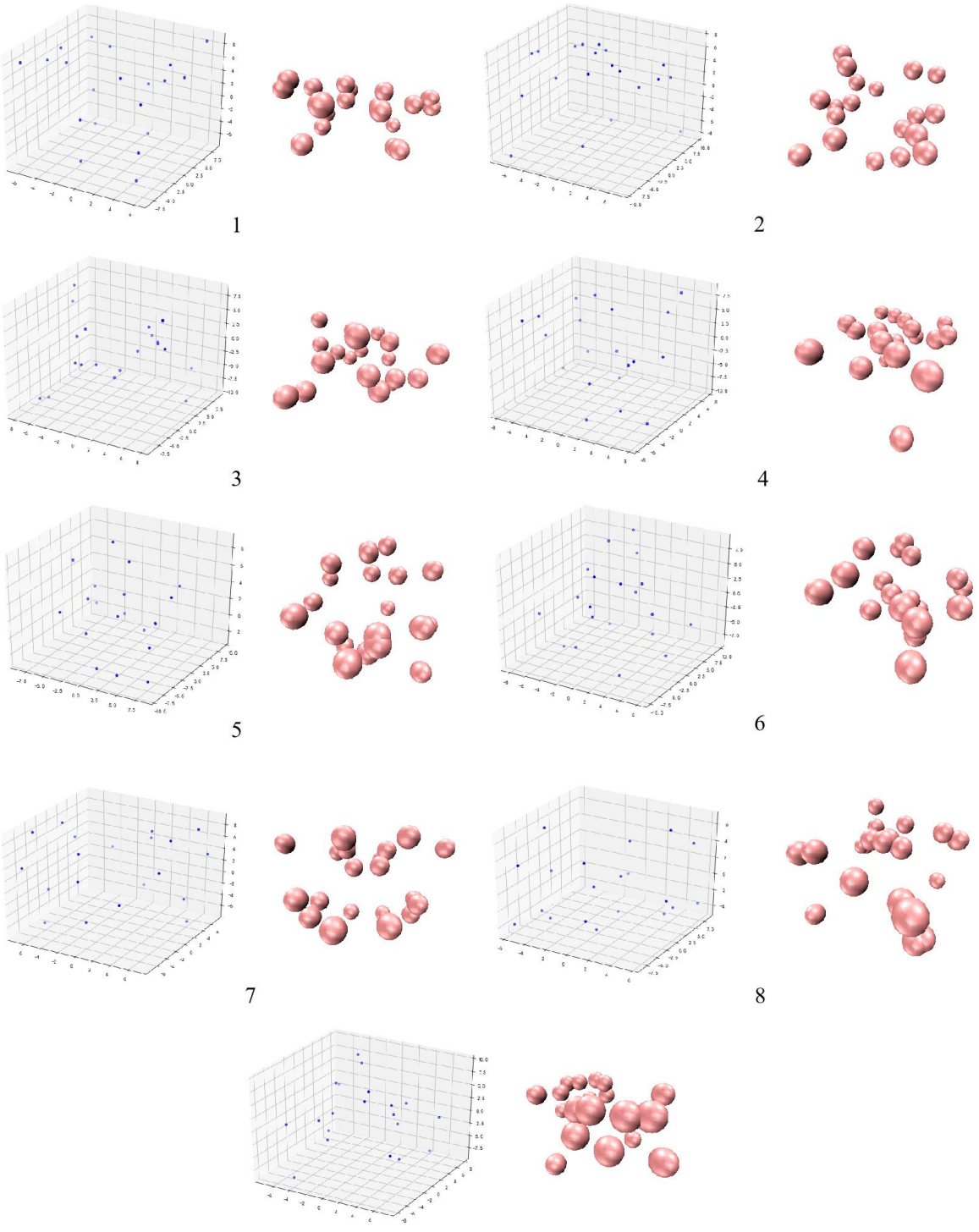


图 5-14 9 次寻优过程确定的相对稳定结构示意图

表 5-4 9 次寻优过程确定的相对稳定结构的能量

结构编号	能量	结构编号	能量
1	-1543.73479573	6	-1536.32194487
2	-1530.69690459	7	-1535.18810031
3	-1531.38804263	8	-1538.24992737
4	-1530.05231209	9	-1542.34875422
5	-1531.08838329		

对比表 5-4 中的数据可得，在由所给实验数据确定范围内生成的限制性对称结构中，能量最小值为-1543.73479573，对应的结构图为图 5-14 中结构 1。

通过对比表 5-3 和 5-4 可知，较实验数据确定的空间范围来看，基于高斯猜想以及最紧密布局假设确定的限制性对称结构范围内生成的结构具有较小的势能，既验证了本文提出的假设的合理性，也有助于找到全局范围内的最优Au_{20}团簇结构。

综上所述，针对问题一，本文首先假设了稳态原子团簇具有限制性对称特征，然后基于高斯猜想与最紧密布局假设确定了一个生成结构的空间范围，同时在由所给实验数据确定的另一个空间范围内生成结构，对两种结构分别使用模拟退火算法进行迭代寻优，表 5-3 和表 5-4 展示了两种结构分别进行 9 次寻优所确定的相对稳定结构对应的能量，表 5-3 中结构 9 对应的能量最小，为-1598.8573653797，其对应的Au_{20}团簇结构为图 5-9 中的结构 9。

6 问题二的模型建立、求解与验证

关于问题 2，首先对金团簇（Au_{32}）进行初始结构的生成，本对问题 1 的结构进行了拓展，增加了 2 种随机结构不同空间范围的生成。关于能量预测数学模型的构建，本文同样使用了 Gupta-IPSO-Stacking 算法。随后，运用模拟退火算法（SA），预测Au_{32}的全局最优结构，描述其形状，并对其稳定性进行分析。

6.1 Au_{20}原子团簇初始结构生成

本文 5.2.2 部分和 5.2.3 部分详细描述了限制性对称结构生成的方法与参数设置，此处使用同样的方法生成限制性对称结构。

6.1.1 限制性对称结构的生成

当 N 为 32 时，计算得 R 为 1.9683，因此本文生成Au_{32}原子团簇限制性对称结构的

空间范围有两个，分别是由 R=1.9683 确定的球体和由所给实验数据确定的长方体空间。基于限制性对称的Au_{32}原子团簇结构坐标生成如表 6-1 所示。

表 6-1 基于限制性对称的Au_{32}原子团簇结构坐标生成

原子序号	原子坐标	原子序号	原子坐标	原子序号	原子坐标
1	(x1,y1,z1)	12	(-x6,-y6,z6+ξ)	23	(x12,y12,z12)
2	(-x1,-y1,z1+ξ)	13	(x7,y7,z7)	24	(-x12,-y12,z12+ξ)
3	(x2,y2,z2)	14	(-x7,-y7,z7+ξ)	25	(x13,y13,z13)
4	(-x2,-y2,z2+ξ)	15	(x8,y8,z8)	26	(-x13,-y13,z13+ξ)
5	(x3,y3,z3)	16	(-x8,-y8,z8+ξ)	27	(x14,y14,z14)
6	(-x3,-y3,z3+ξ)	17	(x9,y9,z9)	28	(-x14,-y14,z14+ξ)
7	(x4,y4,z4)	18	(-x9,-y9,z9+ξ)	29	(x15,y15,z15)
8	(-x4,-y4,z4+ξ)	19	(x10,y10,z10)	30	(-x15,-y15,z15+ξ)
9	(x5,y5,z5)	20	(-x10,-y10,z10+ξ)	31	(x16,y16,z16)
10	(-x5,-y5,z5+ξ)	21	(x11,y11,z11)	32	(-x16,-y16,z16+ξ)
11	(x6,y6,z6)	22	(-x11,-y11,z11+ξ)		

注：ξ为趋近于 0 的实数。

6.1.2 随机结构的生成

为了验证本文前面提出的预测原子团簇能量的 Stacking 集成模型的有效性，也为了群智能算法尽可能地在随机空间内进行全局寻优，本文还设置了在上述两个空间范围内随机生成Au_{32}原子团簇结构。Au_{32}原子团簇结构坐标随机生成如表 6-2 所示。

表 6-2 Au_{32}原子团簇结构坐标随机生成

原子序号	原子坐标	原子序号	原子坐标	原子序号	原子坐标
1	(x1,y1,z1)	12	(x12,y12,z12)	23	(x23,y23,z23)
2	(x2,y2,z2)	13	(x13,y13,z13)	24	(x24,y24,z24)
3	(x3,y3,z3)	14	(x14,y14,z14)	25	(x25,y25,z125)
4	(x4,y4,z4)	15	(x15,y15,z15)	26	(x26,y26,z26)
5	(x5,y5,z5)	16	(x16,y16,z16)	27	(x27,y27,z27)
6	(x6,y6,z6)	17	(x17,y17,z17)	28	(x28,y28,z28)
7	(x7,y7,z7)	18	(x18,y18,z18)	29	(x29,y29,z29)

8	（x8, y8, z8）	19	（x19, y19, z19）	30	（x30, y30, z30）
9	（x9, y9, z9）	20	（x20, y20, z20）	31	（x31, y31, z31）
10	（x10, y10, z10）	21	（x21, y21, z21）	32	（x32, y32, z32）
11	（x11, y11, z11）	22	（x22, y22, z22）		

使用 Stacking 集成模型和模拟退火算法对随机产生的金原子团簇结构进行能量预测并寻找最稳态结构，去除了对限制性对称结构所施加的约束条件，能够在给定的空间范围内考虑到更多种原子结构组合情况，实现全局范围内的最稳态结构寻找，同时也能够验证限制性对称结构寻优的有效性。

6.2 模拟退火算法预测全局最优结构

限制性对称结构和随机结构在两个空间范围内生成Au_{32}原子团簇的方式一共有 4 种，针对生成的 4 种原子团簇结构，本文在此进行了与 5.2.4 部分同样的模拟退火算法多次迭代寻优，每一次寻优设置的迭代次数为 3000 次，每一种结构进行 9 次寻优并通过能量对比确定最优结构。受篇幅限制，本文在此仅展示最优结构的坐标图与对应的结构图，如图 6-1 至图 6-4 所示。

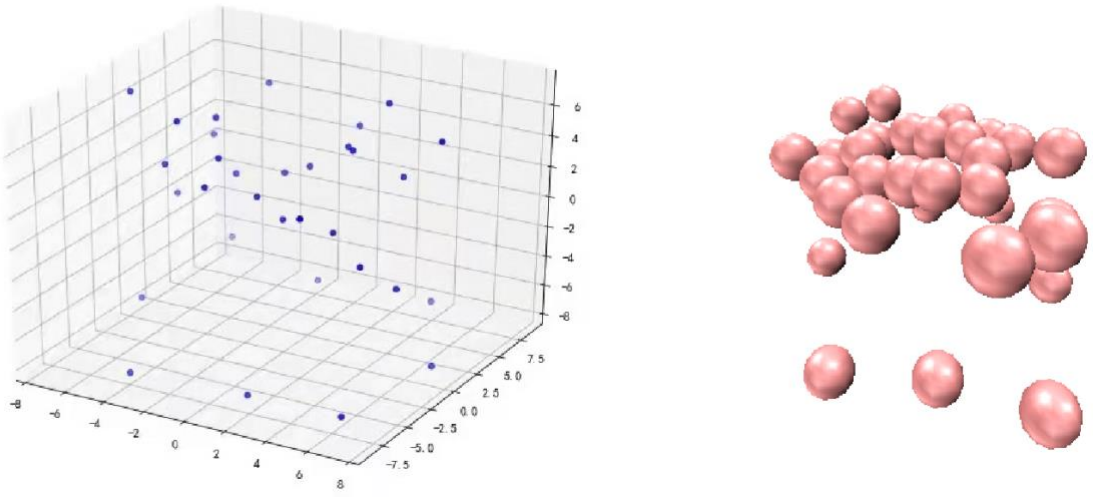


图 6-1 限制性对称结构的Au_{32}团簇在由 R 确定的空间范围内最优结构图

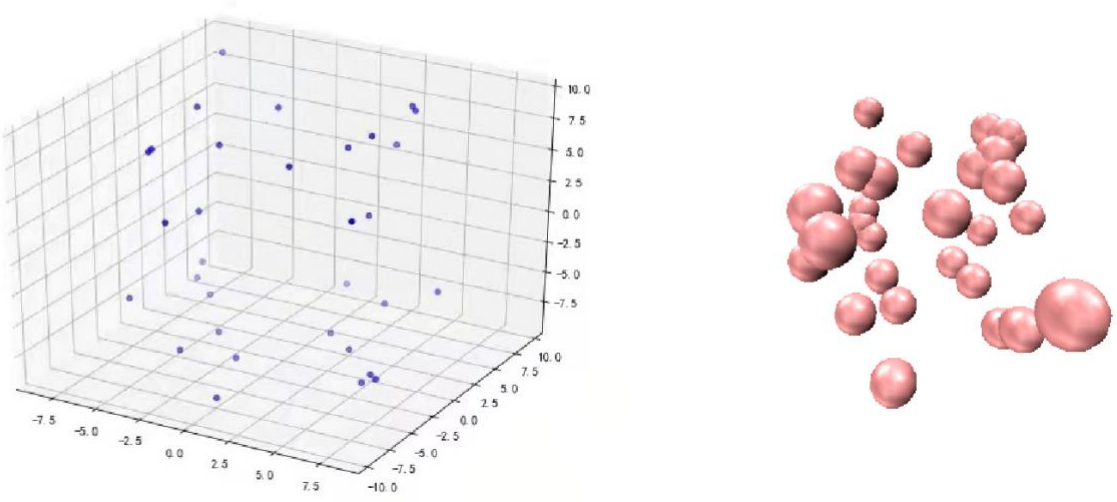


图 6-2 限制性对称结构的Au_{32}团簇在由实验数据确定的空间范围内最优结构图

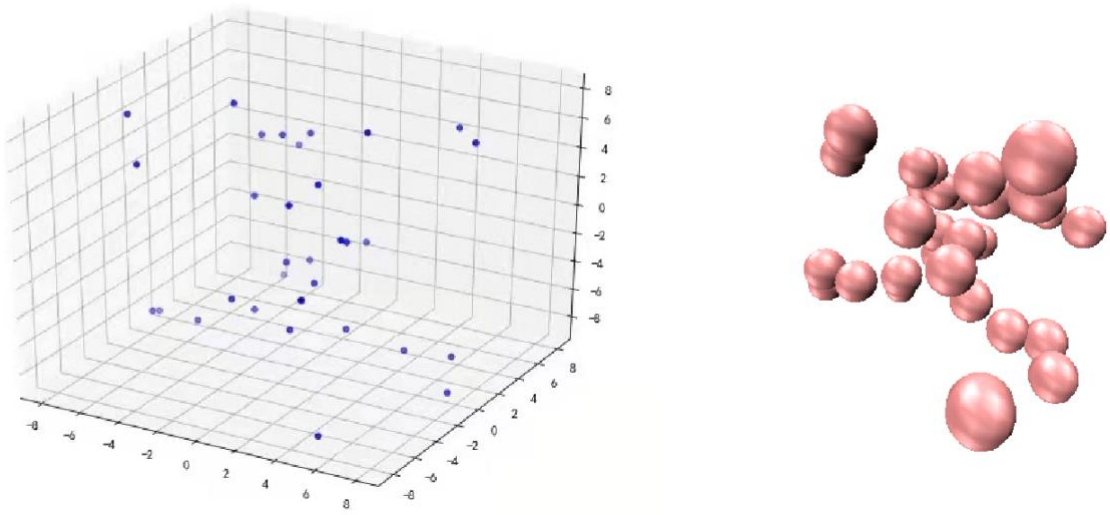


图 6-3 随机结构的Au_{32}团簇在由 R 确定的空间范围内最优结构图

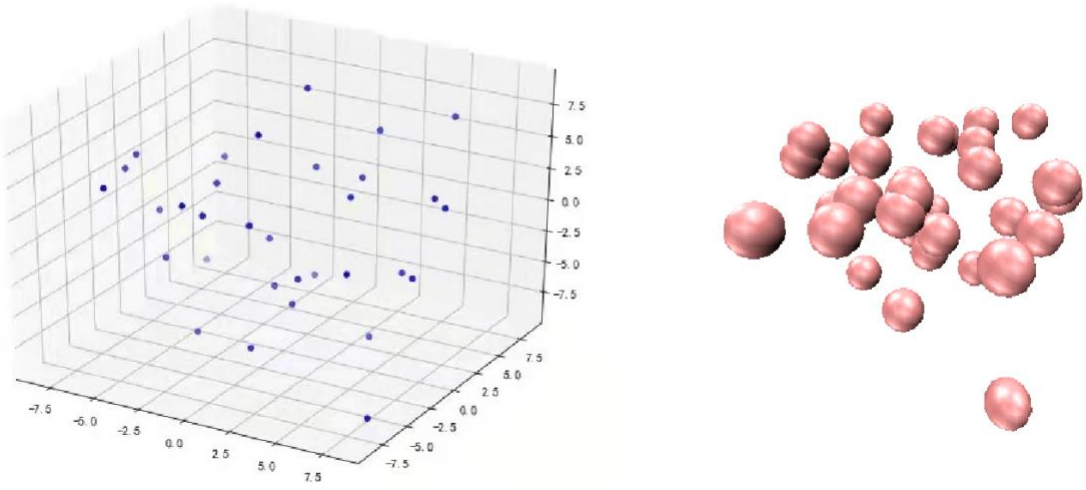


图 6-4 随机结构的Au_{32}团簇在由实验数据确定的空间范围内最优结构图

表 6-3 4 种结构生成的Au_{32}团簇中相对最优结构的能量

结构	能量
限制性对称-R 范围	-5724.8207238540328
限制性对称-实验数据范围	-5009.3344526115595
随机结构-R 范围	-6031.1860358164727
随机结构-实验数据范围	-5473.2178851454821

由表 6-3 可知，图 6-3 所示结构对应的能量最小，该结构即为Au_{32}原子团簇在全局范围内的最稳定结构。

6.3 稳定性分析

根据文献[5]的研究，团簇结构的稳定性常常采用势能量二阶有限差分的方法，这种方法被用于衡量该团簇与其邻近团簇相比的结构稳定性情况。$\Delta_2 E = E(i_{best}+N)+E(i_{best}-N)-2E(i_{best})$ 表示了势能量二阶有限差分随最优结构个数改变的变化公式，若该差分值大于 0 则表明该团簇具有更为稳定的结构，E 为前文构造的势函数 Gupta。图 6-5 表示了Au_{32}原子团簇结构的势能量二阶差分变化图，由图可知，Gupta 的势能量二阶差分值具有多个向上的峰型，表明相较于其他较优结构而言，该结构的稳定性更好。

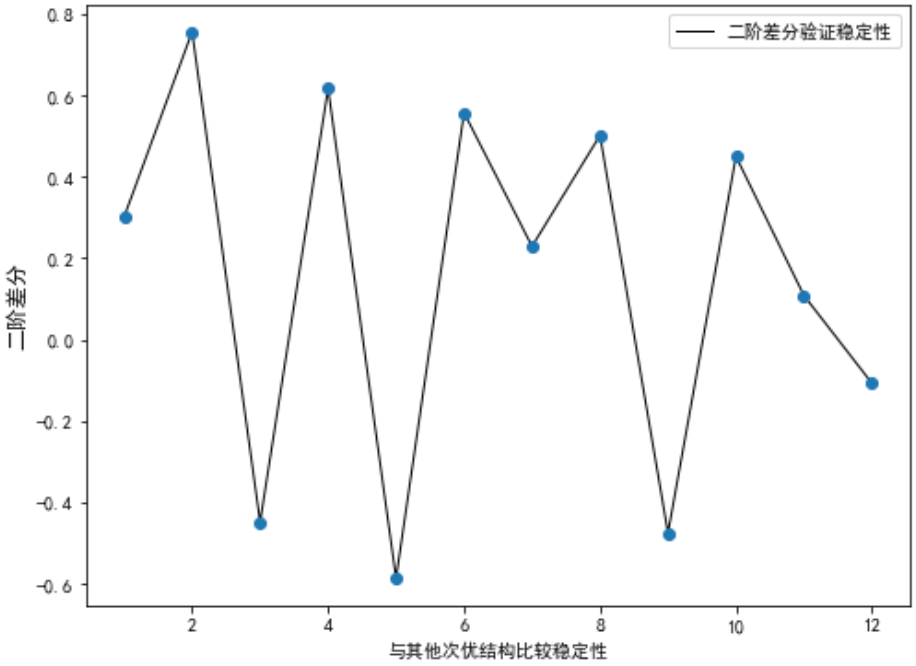


图 6-5 结构稳定性比较

7 问题三的模型建立、求解与验证

关于问题 3，首先对硼团簇（B_{45}^{-}）进行能量预测数学模型的构建，本文构想使用一种新型 Tersoff-IPSO-GRU 算法：首先，建立传统的 Gupta 势能模型，利用改进的粒子群优化算法（IPSO）找出势能模型的 2 项最优参数，并将 1000 个已知数据的位置向量代入此 Tersoff 方程求解，将计算结果作为一个变量其次，为满足机器学习的输入结构，将团簇的位置信息进行库伦矩阵转换，将特征值与势能方程结果放入门控循环单元模型（GRU）预测各团簇的能量，关于 IPSO 优化算法和库伦矩阵下文不再赘述。其次，运用模拟退火算法预测B_{45}^{-}的全局最优结构，描述其形状，并对其稳定性进行分析。

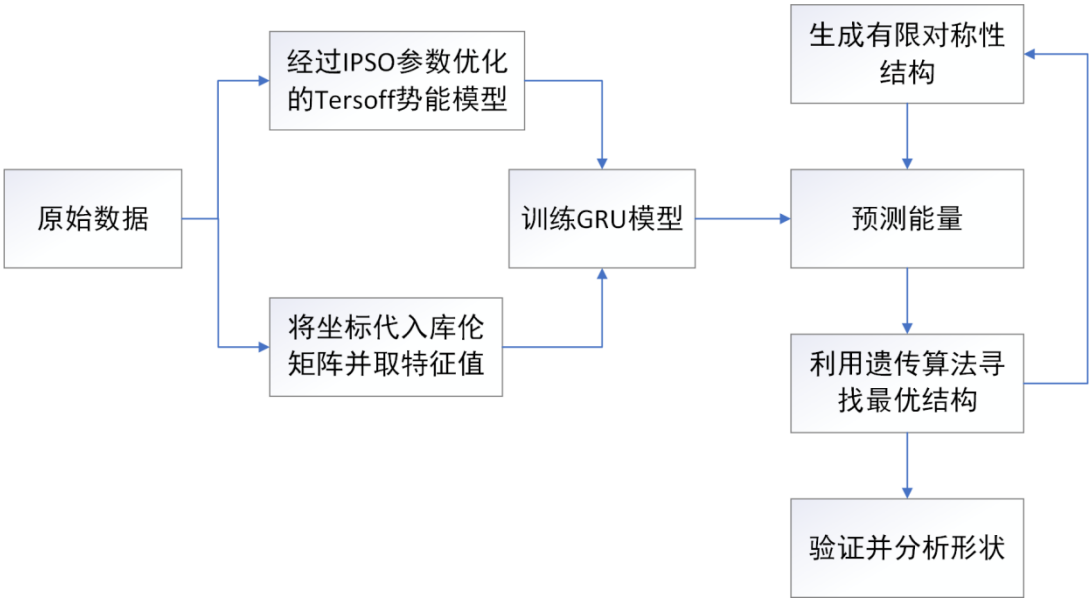


图 7-1 关于硼团簇B_{45}^{-}全局最优结构的思路流程图

7.1 Tersoff-IPSO-GRU 能量预测模型

7.1.1 Tersoff 势能函数

近年来一系列精度不同的相互作用势被提出，如第一原理多体势、应用广泛的 Morse 势、Tersoff 势、EAM 势、Gupta 势等。不同的原子适用于不同的传统势能模型，其中，键序势(Bond-order Potential)与多体势相比，引进了键序的概念来描述共价键系统一两个原子之间键的强度不是常量而是取决于原子的局部环境，可以通过一个与原子配位数相关的键序函数计算键的能量。因此，对于碳、硅、硼这样的非金属团簇较为合适，因此关于硼的势能计算本文选取 Tersoff 模型。

键序势最初是在 80 年代末期由 Tersoff 提出，称为 Tersoff 势网，它不仅可以计算

相应晶格常数、键能、键角、弹性模量和空位形成能，还可以描述系统中化学键的形成和断裂以及原子之间化合键变化的动态过程。在 2002 年, Brenner 等同再次对键序势进行改进，提出了所谓的第二代 Brenner 势函数——REBO（Reactive empirical bond order）势。与 Brenner 势相比，REBO 势扩大了应用范围，正确地反映了共价键的形成和断裂过程，更好地描述和预测固体原子系统中的键长、键角和力场常数，可作为解决大体系复杂分子的有力工具。Tersoff 势能公式为：

总势能

$$E_b = \sum_i \sum_{j(>1)} [V^R(r_{ij}) - b_{ij} V^A(r_{ij})], \tag{16}$$

排斥势

$$V^R(r_{ij}) = f^c(r_{ij})(1 + Q/r_{ij}) A e^{-\alpha r_{ij}}, \tag{17}$$

吸引势

$$V^A(r_{ij}) = f^c(r_{ij}) \sum_{n=1,3} B_n e^{-\beta_n r_{ij}}, \tag{18}$$

键序

$$b_{ij}^{-} = \frac{1}{2}[b_{ij}^{\sigma-\pi} + b_{ij}^{\sigma-\pi}] + b_{ij}^{\pi}, \tag{19}$$

其中，修正项b_{ij}^{π}包括了非局域效应和 C-C 二面角的影响：

$$b_{ij}^{\pi} = \pi_{ij}^{RC} + b_{ij}^{DH}, \tag{20}$$

同样利用改进的粒子群算法来寻找 Tersoff 的参数，ipso 寻优损失值的迭代次数的关系如下图所示，得到的最优参数值分别为 Q=1.3022，b=0.6812。

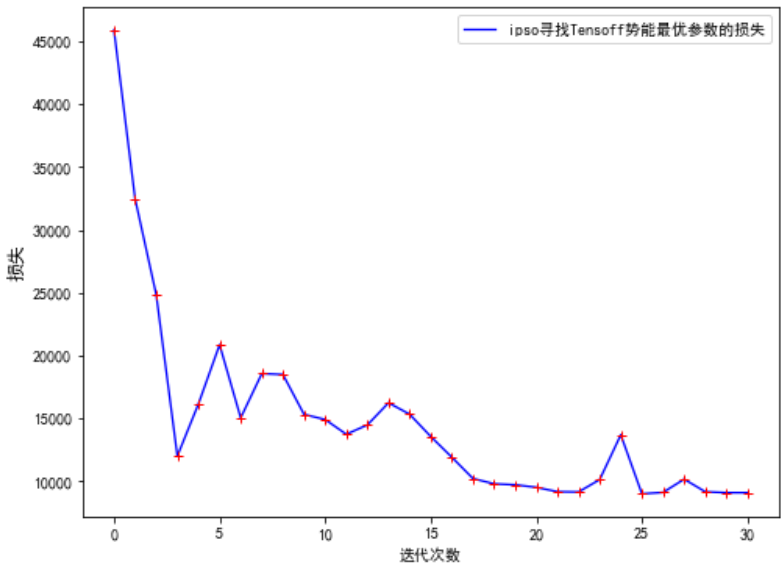


图 7-2 IPSO 迭代寻优过程

7.1.2 门控循环单元（GRU）预测能量

门控循环单元结构（Gate Recurrent Unit，GRU）是循环神经网络（Recurrent Neural Network, RNN）的一种，和长短期记忆人工神经网络（Long-Short Term Memory，LSTM）一样，也是为了解决长期记忆和反向传播中的梯度等问题而提出来的，它能够有效捕捉长序列之间的语义关联, 缓解梯度消失或爆炸现象，且在计算上较 LSTM 更简单，能够大幅提高计算效率。

LSTM 的内部结构有 3 个“门”，分别为输入门、输出门和遗忘门，GRU 对 LSTM 的三个“门”做了优化，仅保留两个，分别为重置门和更新门。重置门决定了如何将新的输入信息与前面的记忆相结合，更新门定义了前面记忆保存到当前时间步的量。若将重置门设置为 1，更新门设置为 0，结果即是标准的 RNN 模型，图 7-3 为 GRU 的结构图。

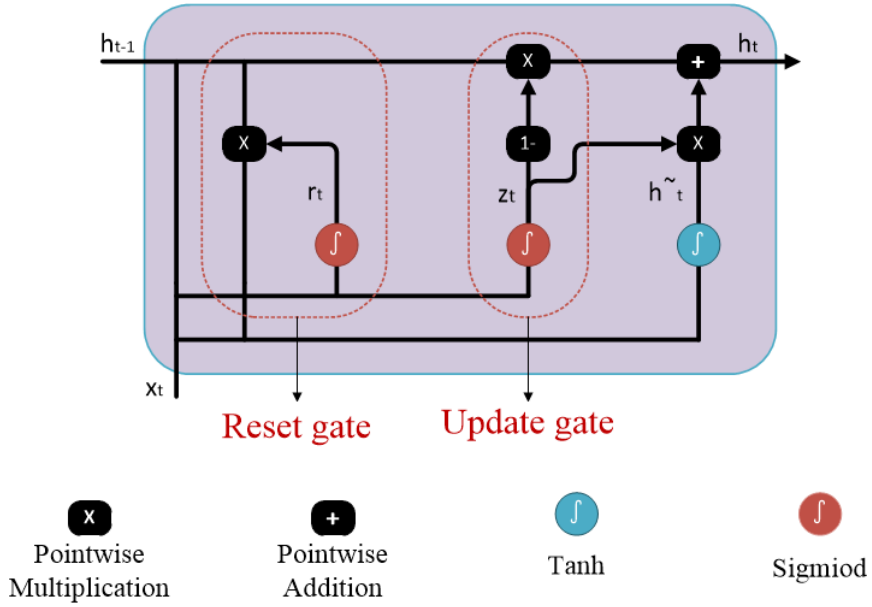


图 7-3 GRU 神经网络单元结构图

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}), \tag{21}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}), \tag{22}$$

$$\tilde{h}_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{t-1} + b_{hn})), \tag{23}$$

$$h_t = (1 - z_t) * \tilde{h}_t + z_t * h_{t-1}, \tag{24}$$

上式中，W 参数均为权重系数，b 参数均为误差项，σ 为 Sigmoid 函数，tanh 为 Tanh 函数，r_t 为重置门的输出结果，z_t 为更新门的输出结果，$\tilde{h}_t$ 为上一个神经元的输出信息

在该神经元中被保留的部分，h_t 为该神经元的输出信息。

7.1.3 模型预测结果对比

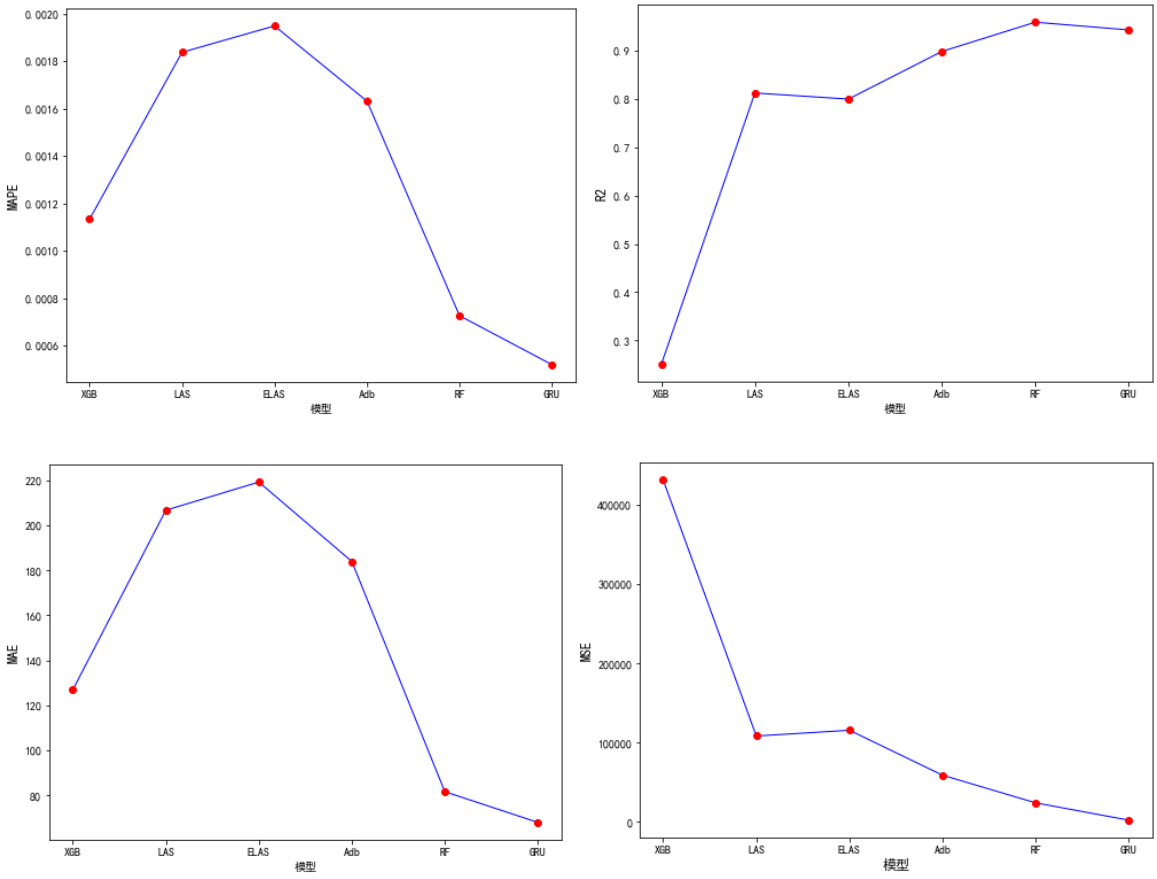


图 7-4 多种模型预测结果的评价指标对比

从图 7-4 可以明显看出，相较于其他机器学习模型，GRU 模型的四个评价指标均是最优的。GRU 模型的 MAPE、MAE 和 MSE 均远小于其他模型，R^2 大于其他模型的平均水平。总体来看，GRU 模型比其他模型的预测性能更好，因此，本文在此部分使用 GRU 模型预测 B^- 原子团簇的能量。

7.2 基于模拟退火算法预测全局最优结构

7.2.1 B_{45}^- 原子团簇初始结构生成

表 7-1 B_{45}^- 原子团簇实验数据的描述性统计

	x	y	z
数量	168795	168795	168795

均值	9.87E-20	9.54E-09	7.7E-09
标准差	3.331249	2.420738	0.705281
最小值	-6.90263	-5.11455	-2.79003
25%	-2.58703	-1.75673	-0.50146
50%	0.00665	-0.01451	-0.01418
75%	2.59215	1.798655	0.485035
最大值	7.07114	5.48891	3.75261

当 N 为 45 时，计算 R 为 2.2053。由表 7-1 可得由所给的实验数据确定的空间范围。

B_{45}^-原子团簇初始结构的生成方法参照本文 6.1 部分Au_{32}原子团簇初始结构的生成办法。由于 5.2.4 部分的预测结果说明由 R 确定范围的限制性对称结构生成的原子团簇具有较小的能量，所以在此部分仅生成由 R 确定范围的限制性对称结构的B_{45}^-原子团簇初始结构。

7.2.2 全局最优结构预测结果

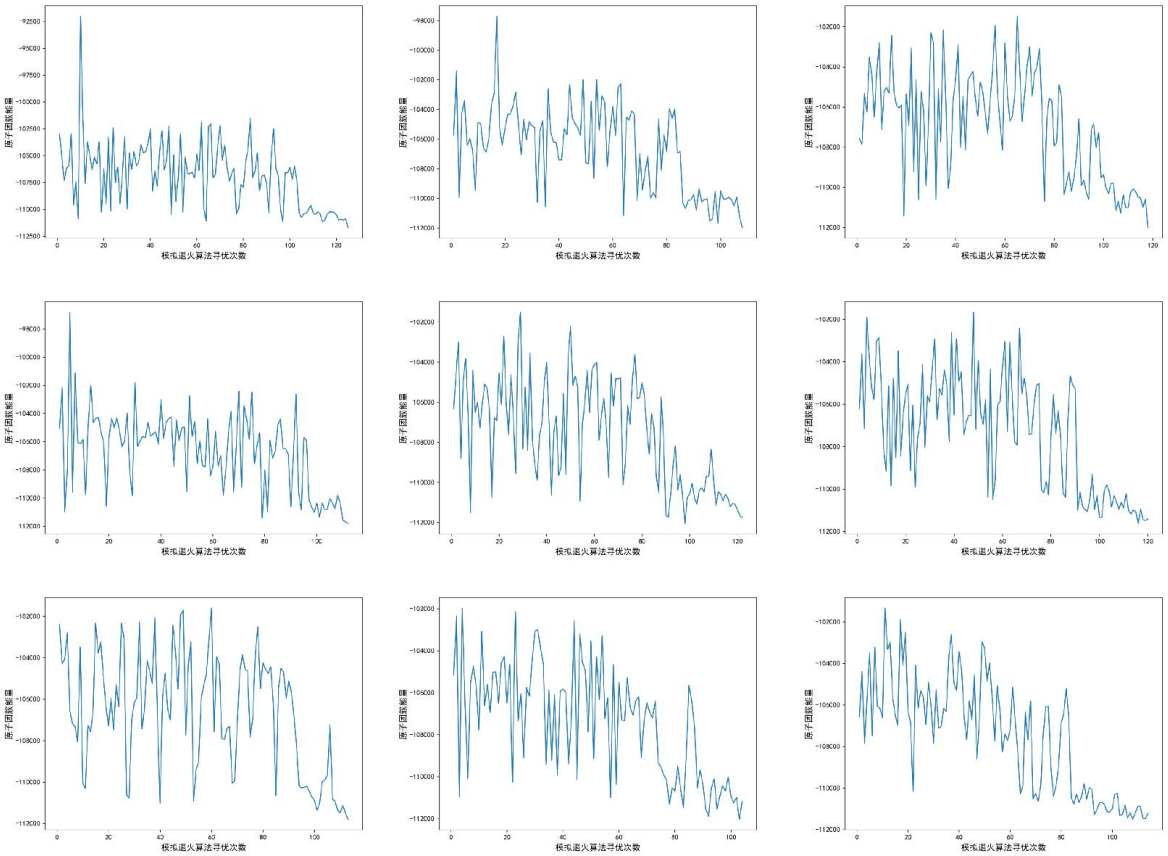


图 7-5 由 R 确定范围的B_{45}^-限制性对称结构模拟退火算法寻优图

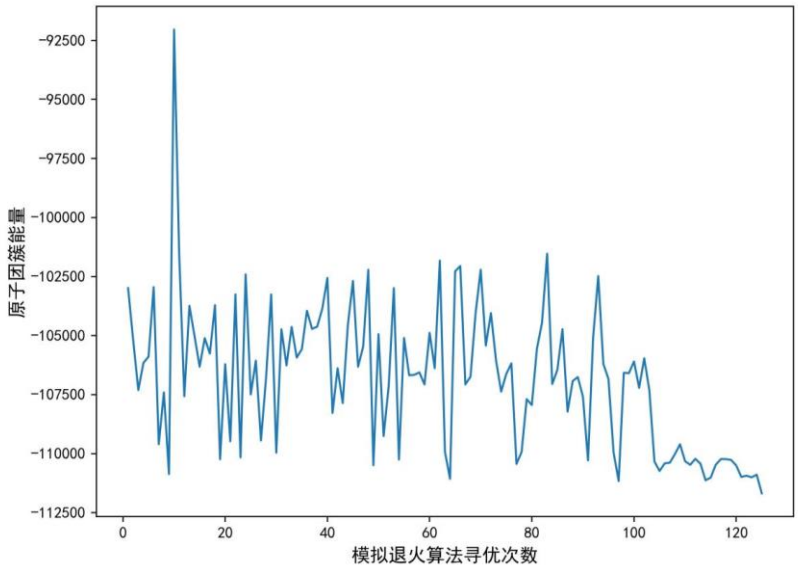
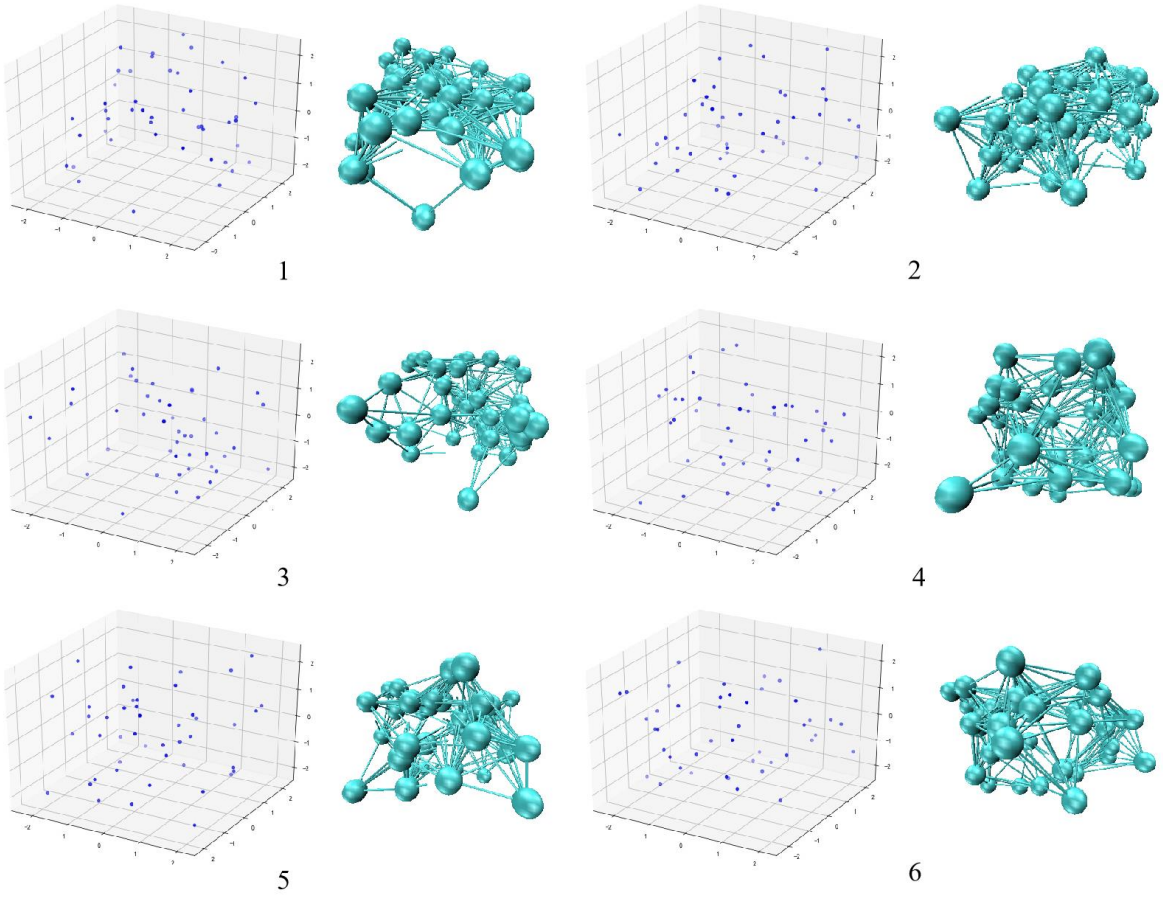


图 7-6 由 R 确定范围的B_{45}^{-}限制性对称结构模拟退火算法最优图



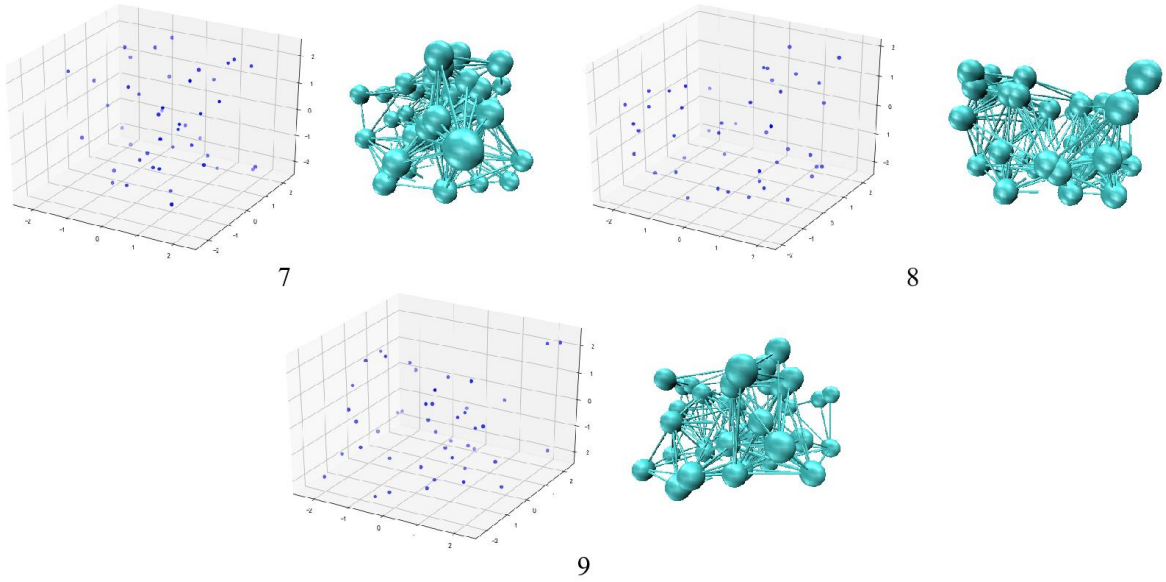


图 7-7 9 次寻优过程确定的B_{45}^-原子团簇相对稳定结构示意图

表 7-2 9 次寻优过程确定的B_{45}^-原子团簇相对稳定结构的能量

结构编号	能量	结构编号	能量
1	-113816.15022984	6	-114093.74694839
2	-113500.62980152	7	-114236.10428152
3	-113889.3245107	8	-113468.32803897
4	-113866.25149028	9	-110942.10536825
5	-113463.41660823		

由表 7-2 可得，9 次寻优过程找到的限制性对称相对稳定结构中结构 7 对应的能量最小，为-114236.10428152，因此认为图 7-7 中结构 7 为B_{45}^-原子团簇全局稳定结构。

8 问题四的模型建立、求解与验证

关于问题 4，首先对硼团簇（B_{40}^-）进行初始结构的生成，本对问题 3 的结构进行了拓展，增加了 3 种随机结构不同空间范围的生成。关于能量预测数学模型的构建，本文同样使用了 Tersoff-IPSO-GRU 算法。然后使用模拟退火算法预测B_{40}^-的全局最优结构，描述其形状，并对其稳定性进行分析。

8.1 硼B_{40}^{-}原子团簇初始结构生成

B_{40}^{-}原子团簇初始结构的生成方法参照本文 6.1 部分Au_{32}原子团簇初始结构的生成办法，共生成 4 种类型的B_{40}^{-}原子团簇初始结构。

当 N 为 40 时，计算 R 为 2.1204。

表 8-1 基于限制性对称的B_{40}^{-}原子团簇结构坐标生成

原子序号	原子坐标	原子序号	原子坐标	原子序号	原子坐标
1	(x1,y1,z1)	15	(x8,y8,z8)	29	(x15,y15,z15)
2	(-x1,-y1,z1+ξ)	16	(-x8,-y8,z8+ξ)	30	(-x15,-y15,z15+ξ)
3	(x2,y2,z2)	17	(x9,y9,z9)	31	(x16,y16,z16)
4	(-x2,-y2,z2+ξ)	18	(-x9,-y9,z9+ξ)	32	(-x16,-y16,z16+ξ)
5	(x3,y3,z3)	19	(x10,y10,z10)	33	(x17,y17,z17)
6	(-x3,-y3,z3+ξ)	20	(-x10,-y10,z10+ξ)	34	(-x17,-y17,z17+ξ)
7	(x4,y4,z4)	21	(x11,y11,z11)	35	(x18,y18,z18)
8	(-x4,-y4,z4+ξ)	22	(-x11,-y11,z11+ξ)	36	(-x18,-y18,z18+ξ)
9	(x5,y5,z5)	23	(x12,y12,z12)	37	(x19,y19,z19)
10	(-x5,-y5,z5+ξ)	24	(-x12,-y12,z12+ξ)	38	(-x19,-y19,z19+ξ)
11	(x6,y6,z6)	25	(x13,y13,z13)	39	(x20,y20,z20)
12	(-x6,-y6,z6+ξ)	26	(-x13,-y13,z13+ξ)	40	(-x20,-y20,z20+ξ)
13	(x7,y7,z7)	27	(x14,y14,z14)		
14	(-x7,-y7,z7+ξ)	28	(-x14,-y14,z14+ξ)		

表 8-2 B_{40}^{-}原子团簇结构随机坐标生成

原子序号	原子坐标	原子序号	原子坐标	原子序号	原子坐标
1	(x1,y1,z1)	15	(x15,y15,z15)	29	(x29,y29,z29)
2	(x2,y2,z2)	16	(x16,y16,z16)	30	(x30,y30,z30)
3	(x3,y3,z3)	17	(x17,y17,z17)	31	(x31,y31,z31)
4	(x4,y4,z4)	18	(x18,y18,z18)	32	(x32,y32,z32)
5	(x5,y5,z5)	19	(x19,y19,z19)	33	(x33,y33,z33)
6	(x6,y6,z6)	20	(x20,y20,z20)	34	(x34,y34,z34)

7	（x7, y7, z7）	21	（x21, y21, z21）	35	（x35, y35, z35）
8	（x8, y8, z8）	22	（x22, y22, z22）	36	（x36, y36, z36）
9	（x9, y9, z9）	23	（x23, y23, z23）	37	（x37, y37, z37）
10	（x10, y10, z10）	24	（x24, y24, z24）	38	（x38, y38, z38）
11	（x11, y11, z11）	25	（x25, y25, z25）	39	（x39, y39, z39）
12	（x12, y12, z12）	26	（x26, y26, z26）	40	（x40, y40, z40）
13	（x13, y13, z13）	27	（x27, y27, z27）		
14	（x14, y14, z14）	28	（x28, y28, z28）		

8.2 全局最优结构预测结果

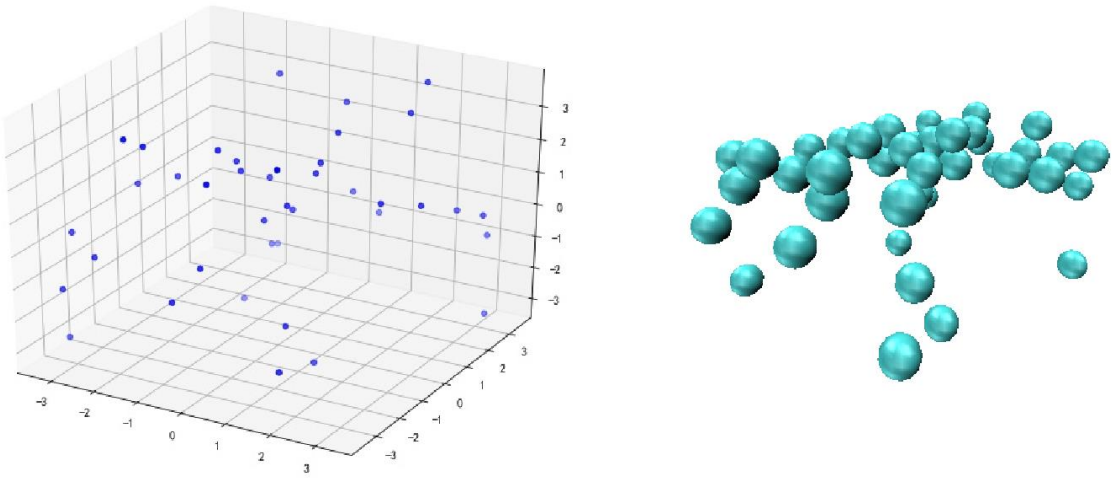


图 8-1 对于B_{40}^{-}对称给定范围的全局最优结构图

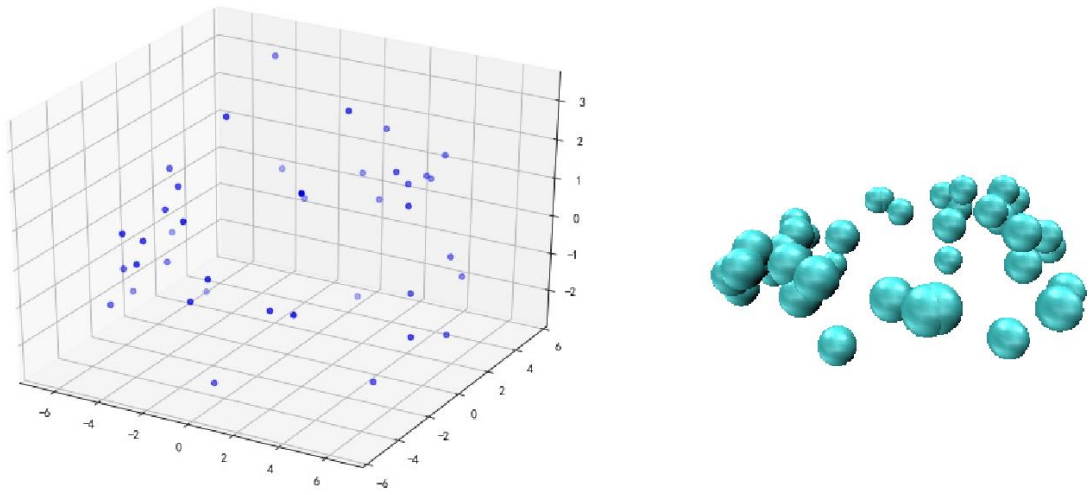


图 8-2 对于B_{40}^{-}对称题目范围的全局最优结构图

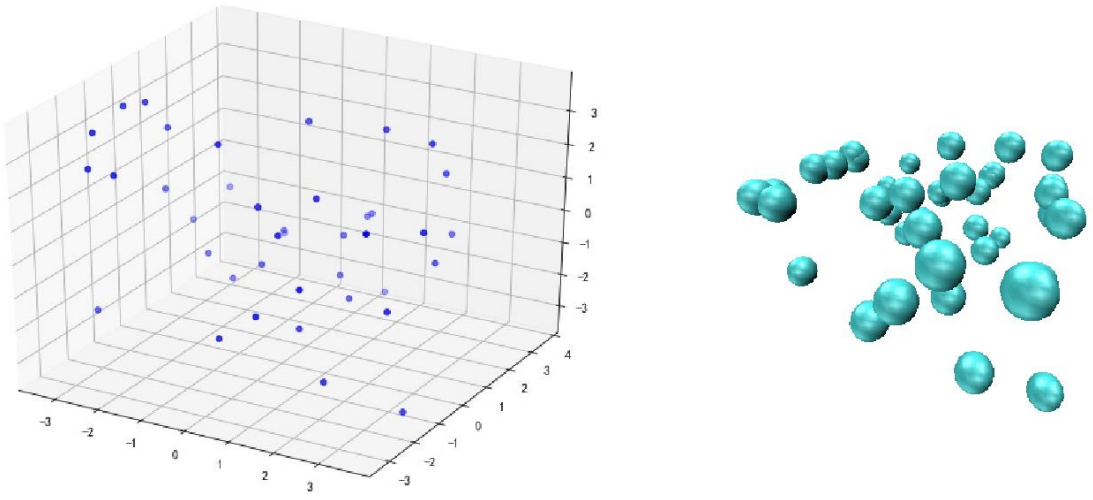


图 8-3 对于B_{40}^{-}随机给定范围的全局最优结构图

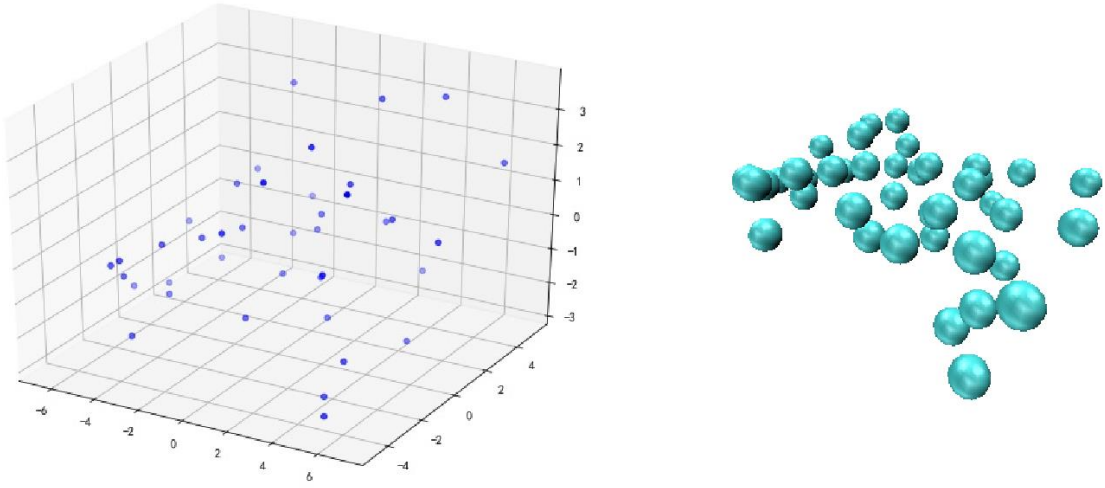


图 8-4 对于B_{40}^{-}随机题目范围的全局最优结构图

表 8-3 4 种结构生成的Au_{32}团簇中相对最优结构的能量

结构	能量
限制性对称-R 范围	-114191.57476693
限制性对称-实验数据范围	-113292.33895437
随机结构-R 范围	-114535.08152818
随机结构-实验数据范围	-113571.02244547

由表 8-3 可知，图 8-3 所示结构对应的能量最小，该结构即为B_{40}^{-}原子团簇在全局范围内的最稳定结构。

8.3 稳定性比较

在这里采用与 6.3 同样的方法来比较B_{40}^{-}的稳定性，通过选取其他 10 个生成的次优结构来验证最优结构的稳定性，Tensoff 势能量二阶差分的图如 8-5 所示，通过该图同样可以得出我们寻找到的最优结构具有一定的稳定性，具有更多向上的峰型。

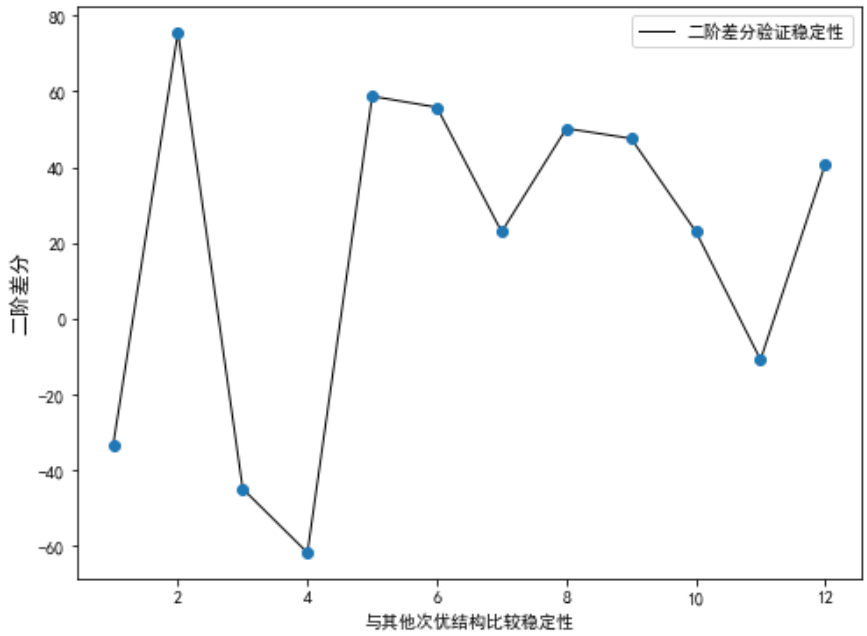


图 8-5 结构稳定性比较

9 参考文献

[1] Cotton F A. Quarterly Review, Chemistry Society, 1966, 20: 389.

[2] Pyykkao P. Angew. Chem. Int. Ed., 2004, 43: 4412.

[3] Rupp M, Tkatchenko A, Müller K, et al. Fast and accurate modeling of molecular atomization energies with machine learning[J]. Physical Review Letters, 2012, 108(5): 058301.

[4] 蔡毅,朱秀芳,孙章丽,陈阿娇. 半监督集成学习综述[J]. 计算机科学, 2017, 44(S1):7-13.

[5] Cai Y, Zhu X F, Sun Z L,Chen A J. Overview of Semi-supervised integrated Learning[J].Computer Science, 2017, 44(S1):7-13.

[6] 周志华.机器学习[M]. 北京, 清华大学出版社, 2016. ZHOU Z H. Machine Learning[M]. BEI JING, Tsinghua University Press,2016.

[7] Breiman L. Random forest[J]. Machine Learning, 2001, 45:5-32.

[8] Chen T , Guestrin C . XGBoost: A scalable tree boosting sys- tem[C]. Proceedings of the 22nd ACM SIGKDD Internation- al Conference on Knowledge Discovery and Data Mining. New York: ACM, 2016, 785-794.

[9] Freund Y , Schapire R A . A decision- theoretic generalization of on-line learning and an application to boosting[C]. Pro- ceedings of the Second European Conference on Computational Learning Theory. Berlin: Springer-Verlag, 1995. 23-37.

[10]毛华平,王红艳,倪羽,徐国亮,马美仲,朱正和,唐永建.Au_n(n=2—9)团簇的几何结构和电子特性[J].物理学报,2004(06):1766-1771.

[11]曹欣伟.B_n 及 B_nN 小团簇结构与稳定性的密度泛函理论研究[D].西北大学,2009.

[12]Kryachko, E. S., and Françoise Remacle. "The magic gold cluster Au20." International Journal of Quantum Chemistry 107.14 (2007): 2922-2934.

[13]魏晓辉,周长宝,沈笑先,刘圆圆,童群超.机器学习加速 CALYPSO 结构预测的可行性[J].吉林大学学报(工学版),2021,51(02):667-676.

[14]高朋越,吕健,王彦超,马琰铭.基于智能全局优化算法的理论结构预测[J].物理,2017,46(09):582-589.

[15]刘绍军. 原子间相互作用势函数的若干理论与应用研究[D].北京师范大学,2001.

[16]Kryachko E S , F Remacle. The magic gold cluster Au20[J]. International Journal of Quantum Chemistry, 2007, 107(14):2922-2934.

[17]Philipp Gruene,David M. Rayner,Britta Redlich,Alexander F. G. van der Meer,Jonathan T. Lyon,Gerard Meijer,Andre Fielicke. ChemInform Abstract: Structures of Neutral Au7, Au19, and Au20Clusters in the Gas Phase.[J]. ChemInform,2008,39(42).

10 代码

代码一：导入 Au20 数据、导入 B45 数据、生成 Gupta 势能公式、利用改进的粒子群算法寻找 Gupta 势能参数

```
import random
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import pandas as pd
from sklearn.model_selection import GridSearchCV
import re
import numpy as np
import matplotlib.pyplot as plt
```

```
#导入 Au20 结构信息
xyz = []
AU_20 = []
energy_20 = []
xyz_i = []
for i in range(1000):
    if i != 155:     #不存在 155 个文件
        filename = 'Au20_OPT_1000//%d.xyz'%i
        with open(filename, 'r') as f:
            content = f.readlines()
            AU_20.append(float(content[0].strip()))
            energy_20.append(float(content[1].strip()))
            for i in range(2,22,1):
                a = content[i][3:].strip()
                data_a = re.split('\s+',a.strip())
                xyz.append([float(i)    for i in data_a])
    else:
        pass
#导出结构信息和能量
df_xyz = pd.DataFrame(xyz)
df_xyz['energy'] = energy_20
df_xyz.to_excel('坐标.xls')

#导入 B45 数据
xyz_b45 = []
B_45 = []
energy_B45 = []
xyz_b_45 = []

for i in range(3751):
    filename = 'B45-_OPT_3751//%d.xyz'%i
    with open(filename, 'r') as f:
        content = f.readlines()
        B_45.append(float(content[0].strip()))
        energy_B45.append(float(content[1][24:].strip()))
        for i in range(2,47,1):
            a = content[i][3:].strip()
            data_a = re.split('\s+',a.strip())
            xyz_b45.append([float(i)    for i in data_a])


#定义 Gupta 势能公式
def V_r_m(A,p,q,N,epsilon,r0,xyz_i):
    V_r = 0
    V_m = 0
```

```
    V_i = 0
    r = np.array(r_ij(xyz_i))
    for i in range(1,N):
        V_r_i = 0
        V_m_i = 0
        V_r_m_i = 0
        for j in range(1,N):
            if i == j:
                pass
            else:
                V_r_ii = A*np.exp((-p*((r[i-1,j-1]/r0)-1)))
                V_r_i +=V_r_ii
                V_m_ii = np.sqrt((epsilon**2)*np.exp(-2*q*((r[i-1,j-1]/r0)-1)))
                V_m_i +=V_m_ii
            V_r += V_r_i
            V_m +=V_m_i
            V_r_m_i += V_r-V_m
    V_i +=V_r_m_i
     return V_i

#定义 tensoff 势能函数
def V_r_m(mu,epilson,N,xyz_i):
    V_r = 0
    V_r_i =0
    V_i = 0
    r = np.array(r_ij(xyz_i))
    for i in range(1,N):
        for j in range(i+1,N+1):
            V_r = (mu/r[i-1,j-1])**12-(mu/r[i-1,j-1])**6
            V_r_i +=V_r
    V_i = 4*epilson*V_r_i
    return V_i

#定义欧几里得距离
def eucldist_forloop(coords1, coords2):
    dist = 0
    for (x, y) in zip(coords1, coords2):
        dist += (x - y)**2
    return dist**0.5

#定义 20 个原子 Au20 的欧几里得距离矩阵
def r_ij(xyz_i):
    r_ij = []
    ojld_i = xyz_i
    for j in range(20):
```

```
            a_ij = []
            for h in range(20):
                dist = eucldist_forloop(ojld_i[j],ojld_i[h])
                a_ij.append(dist)
            r_ij.append(a_ij)
        return r_ij

#IPSO 寻找 Gupta 最优实验参数、IPSO 寻找 Tensoff 最优实验参数

#定义适应度函数
def fitness_function(position,N):
    energy_pred = []
    for i in range(0,10000,20):
        xyz_i = xyz[i:i+20]
        energy_pred.append(V_r_m(position[0],position[1],position[2],N,position[3],4.00132,xyz_i))
    energy_wucha = mean_absolute_error(energy_pred, energy_true)
     return energy_wucha

#定义粒子群算法的参数
class args:
    r1 = random.random()    #局部学习因子
    r2 = random.random()    #全局学习因子
    c1 = 1.5 #  局部学习因子
    c2 = 1.5 #  全局学习因子
    n_iterations = 20 #  迭代步数
    n_particles = 30 #  粒子数
    wmax = 0.9
    wmin = 0.4
    w = wmax
for i in range(args.n_particles):
    max_velocity = np.array([0.03,0.03])
    min_velocity = np.array([-0.03,-0.03])
def eval_velocity(new_velocity):
    for j in range(0,2):
        if(new_velocity[j] > max_velocity[j]):
            new_velocity[j] = max_velocity[j]
        elif(new_velocity[j] < min_velocity[j]):
            new_velocity[j] = min_velocity[j]
    return new_velocity
def eval_position(particle_position_vector):
    for j in range(0,2):
        if(particle_position_vector[j] > max_position[j]):
            particle_position_vector[j] = max_position[j]
        elif(particle_position_vector[j] < min_position[j]):
            particle_position_vector[j] = min_position[j]
```

```
    return particle_position_vector

#定义粒子群迭代寻优过程
def pso_gupta(N):
    #初始化参数
    #随机初始化每个粒子的位置信息
    particle_position_vector = np.array([np.array([np.random.uniform(0.07,0.2),
np.random.uniform(8.5,11.5),np.random.uniform(0.8,2.5),np.random.uniform(1,2.5)]) for _ in range(args.n_particles)])
    #粒子个体位置信息
    pbest_position = particle_position_vector
    #粒子个体最优适应度值
    pbest_fitness_value = np.array([float('inf') for _ in range(args.n_particles)])
    #群体最优适应度值
    gbest_fitness_value = np.array([float('inf')])
    #群体中最优位置
    gbest_position = np.array([float('inf'), float('inf'), float('inf'), float('inf')])
    #初始化每个粒子的速度信息
    #velocity_vector = ([np.array([random.random()*0.3,random.random()*0.3]) for _ in range(args.n_particles)])
    velocity_vector=np.array([np.array([(-1)** (bool(random.getrandbits(1)))*random.random()*0.2,(-1) **
(bool(random.getrandbits(1))) * random.random()*0.2,(-1) ** (bool(random.getrandbits(1))) * random.random()*0.2,(-1) **
(bool(random.getrandbits(1))) * random.random()*0.2]) for _ in range(args.n_particles)])
    iteration = 0
    mae_i = []
    #迭代寻找最优适应度值
    while iteration < args.n_iterations:
        for i in range(args.n_particles):
            fitness_cadidate = fitness_function(particle_position_vector[i], N)
            print("error of particle-", i, "is (test)", fitness_cadidate, " At (A,p,q,epsilon): ",
                  particle_position_vector[i])
            if (pbest_fitness_value[i] > fitness_cadidate):
                pbest_fitness_value[i] = fitness_cadidate
                pbest_position[i] = particle_position_vector[i]
            if (gbest_fitness_value > fitness_cadidate):
                gbest_fitness_value = fitness_cadidate
                gbest_position = particle_position_vector[i]
        g_best_mae = fitness_function(gbest_position, N)
        mae_i.append(g_best_mae)
        print("The best position in iteration number", iteration + 1, "is", gbest_position, "with mae:", g_best_mae)
        for i in range(args.n_particles):
            if iteration == 0:
                w_s = args.w
            else:
                w_s = w_s
            new_velocity = (w_s * velocity_vector[i]) + (args.c1 * args.r1) * (
                    pbest_position[i] - particle_position_vector[i]) + (args.c2 * args.r2) * (
```

```
                                                gbest_position - particle_position_vector[i])
                new_velocity = eval_velocity(new_velocity)
                new_position = new_velocity + particle_position_vector[i]
                particle_position_vector[i] = new_position
            w_s = (args.wmax - args.wmin)*(args.n_iterations - iteration)/args.n_iterations + args.wmin
            iteration = iteration + 1
#画出随着迭代次数增加适应度函数变化图
plt.rcParams['figure.figsize'] =(8.0,6.0)
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False
plt.plot(mae_i,label='ipso 寻找 Gupta 势能最优参数的损失',color='blue',linewidth=1.3)
plt.plot(mae_i,'r+',linewidth = 3)
plt.xlabel('迭代次数')
plt.ylabel('损失',fontsize = 12)
plt.legend()
plt.show()
```

代码二：定义库伦矩阵、计算特征值、定义欧几里得距离

```
#定义库伦矩阵
def get_coulombmatrix(molecule, largest_mol_size=None):
    numberAtoms = len(molecule)
    if largest_mol_size == None or largest_mol_size == 0:
        largest_mol_size = numberAtoms
    cij = np.zeros((largest_mol_size, largest_mol_size))
    xyzmatrix = molecule
    chargearray = 79
    for i in range(numberAtoms):
        for j in range(numberAtoms):
            if i == j:
                cij[i][j] = 0.5 * chargearray ** 2.4
            else:
                dist = np.linalg.norm(np.array(xyzmatrix[i]) - np.array(xyzmatrix[j]))
                dist = dist.real
                cij[i][j] = chargearray * chargearray / dist
    return cij
#循环计算库伦矩阵的特征值
tezheng_Au20 = []
for i in range(0,19980,20):
    kulun_i = xyz[i:i+20]
    kulun_juzhen = get_coulombmatrix(kulun_i, largest_mol_size=None)
    tezhengzhi_juzhen= np.linalg.eig(kulun_juzhen)
    tezheng_Au20.append(tezhengzhi_juzhen[0])
```

代码三：建立集成学习模型（stacking）

```
#导入集成学习包
```

```
from sklearn.model_selection import cross_val_score, RandomizedSearchCV
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import StackingRegressor
from sklearn.svm import LinearSVR
from sklearn.linear_model import RidgeCV

#生成初始学习器
SVR = SVR()
las = Lasso()
elastic = ElasticNet()
rf = RandomForestRegressor()
xgbr = XGBRegressor()
adb = AdaBoostRegressor()
lsvr = LinearSVR()

#划分训练集和测试集
df = pd.read_excel('data_Au20.xls')
cut = int(0.7*len(df))
X_train = df.iloc[:cut,:-1].values
X_test    = df.iloc[cut:,:-1].values
y_train = df.iloc[:cut,-1].values
y_test = df.iloc[cut:,-1].values

#1.SVR
SVR.fit(X_train,y_train)
cv = cross_val_score(SVR, X_train, y_train,cv=5,    n_jobs=-1)
print('Cross val:', cv )
print('Cross val mean:', cv.mean())
#2.Lasso
las.fit(X_train,y_train)
cv = cross_val_score(las, X_train, y_train,cv=5, error_score='neg_root_mean_squared_error', n_jobs=-1)
print('Cross val:', cv )
print('Cross val mean:', cv.mean())
#3.ElasticNet
elastic.fit(X_train,y_train)
cv = cross_val_score(elastic, X_train, y_train,cv=5, error_score='neg_root_mean_squared_error', n_jobs=-1)
print('Cross val:', cv )
print('Cross val mean:', cv.mean())
#4.RandomForest
rf.fit(X_train,y_train)
```

```
cv = cross_val_score(rf, X_train, y_train,cv=5, error_score='neg_root_mean_squared_error', n_jobs=-1)
print('Cross val:', cv )
print('Cross val mean:', cv.mean())
#5.XGBRegressor
xgbr.fit(X_train,y_train)
cv=cross_val_score(xgbr,X_train,y_train,cv=5,error_score='neg_root_mean_squared_error',n_jobs=-1)
print('Cross val:', cv )
print('Cross val mean:', cv.mean())
#6.adbRegressor
adb.fit(X_train,y_train)
cv=cross_val_score(adb,X_train,y_train,cv=5,error_score='neg_root_mean_squared_error',n_jobs=-1)
print('Cross val:', cv )
print('Cross val mean:', cv.mean())
#7.LinearSVR
lsvr.fit(X_train,y_train)
cv=cross_val_score(lsvr,X_train,y_train,cv=5,error_score='neg_root_mean_squared_error',n_jobs=-1)
print('Cross val:', cv )
print('Cross val mean:', cv.mean())

#利用网格搜索寻找最优学习器
#1.SVR
param_grid = {'C':[0.6, 0.7, 0.8, 0.9, 1],'epsilon':[0.3,0.4,0.5], 'coef0':[0.1, 0.001, 0.0001]}
SVR_grid = GridSearchCV(SVR, param_grid, cv=5, verbose=True, return_train_score=True,
error_score='neg_root_mean_squared_error', n_jobs=-1)
SVR_grid.fit(X_train, y_train)
print (SVR_grid.best_score_, SVR_grid.best_params_)
#2.Lasso
param_grid = { 'alpha': [0.0001,0.001, 0.01, 0.1, 1], 'max_iter' : [30, 40, 45]}
las_grid = GridSearchCV(las, param_grid,    cv=5, verbose=True, return_train_score=True, n_jobs=-1)
las_grid.fit(X_train, y_train)
print (las_grid.best_score_, las_grid.best_params_)
#3.ElasticNet
param_grid = {'alpha': [0.001, 0.5, 1.0], 'max_iter' : [150, 170, 175], 'l1_ratio': [0.3, 0.4, 0.45] }
elastic_grid = GridSearchCV(elastic, param_grid,    cv=5, verbose=True, return_train_score=True, n_jobs=-1)
elastic_grid.fit(X_train, y_train)
print (elastic_grid.best_score_, elastic_grid.best_params_)
#4.RandomForest
param_grid = { 'n_estimators': [325, 350, 400],'max_features': ['sqrt'],'max_depth' : [15, 20, 25],}
rf_grid = GridSearchCV(rf, param_grid,    cv=5, verbose=True, return_train_score=True,n_jobs=-1)
rf_grid.fit(X_train, y_train)
print (rf_grid.best_score_, rf_grid.best_params_)
#5.XGBRegressor
param_grid = {
        'num_boost_round': [10, 15, 20],
        'eta': [0.01, 0.02, 0.03],
```

```
        'max_depth': [3, 5, 7],
        'subsample': [0.3, 0.5, 0.9],
        'colsample_bytree': [0.3, 0.5, 0.9],
    }
xgbr_grid = GridSearchCV(xgbr, param_grid, cv=5, verbose=True, return_train_score=True, error_score='neg_root_mean_squared_error', n_jobs=-1)
xgbr_grid.fit(X_train, y_train)
print (xgbr_grid.best_score_, xgbr_grid.best_params_)
#5.AdaboostRegressor
param_grid = {
        'n_estimators': [50, 100, 150, 200],
        'learning_rate': [0.3, 0.5, 0.1, 0.13],
        'loss': ['linear', 'square']
    }
adb_grid = GridSearchCV(adb, param_grid, cv=5, verbose=True, return_train_score=True, error_score='neg_root_mean_squared_error', n_jobs=-1)
adb_grid.fit(X_train, y_train)
print (adb_grid.best_score_, adb_grid.best_params_)
#1.linearSVR
param_grid = {'C':[0.6, 0.7, 0.8, 0.9, 1],'epsilon':[0.3,0.4,0.5,0.1]}
lsvr_grid = GridSearchCV(lsvr, param_grid, cv=5, verbose=True, return_train_score=True, error_score='neg_root_mean_squared_error', n_jobs=-1)
lsvr_grid.fit(X_train, y_train)
print (lsvr_grid.best_score_, lsvr_grid.best_params_)

#得到 stacking 融合模型
SVR_best = SVR_grid.best_estimator_
xgbr_best = xgbr_grid.best_estimator_
rf_best = rf_grid.best_estimator_
las_best = las_grid.best_estimator_
elastic_best = elastic_grid.best_estimator_
lsvr_best = lsvr_grid.best_estimator_
adb_best = adb_grid.best_estimator_
estimators = [('SVR_best', SVR_best),
              ('xgbr_best', xgbr_best),
              ('rf_best', rf_best),
              ('las_best', las_best),
              ('elastic_best', elastic_best),
              ('adb_best', adb_best),
              ('lsvr_best', lsvr_best)
              ]
reg = StackingRegressor(estimators=estimators,final_estimator=RidgeCV())
reg.fit(X_train, y_train).score(X_test, y_test)

#各个机器学习模型结果对比
```

```
SVR_predict = SVR_best.predict(X_test)
xgbr_predict = xgbr_best.predict(X_test)
rf_predict = rf_best.predict(X_test)
las_predict = las_best.predict(X_test)
elastic_predict = elastic_best.predict(X_test)
lsvr_predict = lsvr_best.predict(X_test)
adb_predict = adb_best.predict(X_test)
stacking_predict =    reg.predict(X_test)

#建立评价指标
#输出结果
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
import math
from sklearn.metrics import r2_score as R2

#导入平均绝对误差 MAS，均方误差 MSE 函数
#定义平均绝对百分比误差
def mape(y_true, y_pred):
    return np.mean(np.abs((y_pred - y_true) / y_true))
MSE_SVR = mean_squared_error(SVR_predict, y_test)
MSE_xgbr = mean_squared_error(xgbr_predict, y_test)
MSE_rf = mean_squared_error(rf_predict, y_test)
MSE_las = mean_squared_error(las_predict, y_test)
MSE_elastic = mean_squared_error(elastic_predict, y_test)
MSE_lsvr = mean_squared_error(lsvr_predict, y_test)
MSE_adb = mean_squared_error(adb_predict, y_test)
MSE_stacking = mean_squared_error(stacking_predict, y_test)

MAE_SVR = mean_absolute_error(SVR_predict, y_test)
MAE_xgbr = mean_absolute_error(xgbr_predict, y_test)
MAE_rf = mean_absolute_error(rf_predict, y_test)
MAE_las = mean_absolute_error(las_predict, y_test)
MAE_elastic = mean_absolute_error(elastic_predict, y_test)
MAE_lsvr = mean_absolute_error(lsvr_predict, y_test)
MAE_adb = mean_absolute_error(adb_predict, y_test)
MAE_stacking = mean_absolute_error(stacking_predict, y_test)

MAPE_SVR = mape( y_test,SVR_predict)
MAPE_xgbr = mape( y_test,xgbr_predict)
MAPE_rf = mape( y_test,rf_predict)
MAPE_las = mape( y_test,las_predict)
MAPE_elastic = mape( y_test,elastic_predict)
MAPE_lsvr = mape( y_test,lsvr_predict)
MAPE_adb = mape( y_test,adb_predict)
```

```
MAPE_stacking = mape( y_test,stacking_predict)

R2_SVR = R2( y_test,SVR_predict)
R2_xgbr = R2( y_test,xgbr_predict)
R2_rf = R2( y_test,rf_predict)
R2_las = R2(y_test,las_predict)
R2_elastic = R2( y_test,elastic_predict)
R2_lsvr = R2( y_test,lsvr_predict)
R2_adb = R2( y_test,adb_predict)
R2_stacking = R2( y_test,stacking_predict)
#绘图
mse = [MSE_SVR, MSE_xgbr, MSE_las, MSE_elastic, MSE_lsvr, MSE_adb,MSE_rf,MSE_stacking]
mae = [MAE_SVR, MAE_xgbr, MAE_las, MAE_elastic, MAE_lsvr, MAE_adb,MAE_rf,MAE_stacking]
mape=[MAPE_SVR,MAPE_xgbr,MAPE_las,MAPE_elastic,MAPE_lsvr,MAPE_adb,MAPE_rf,MAPE_stacking]
R2 = [R2_SVR, R2_xgbr, R2_las,R2_elastic, R2_lsvr, R2_adb,R2_rf,R2_stacking]
mz = ['SVR', 'XGB', 'LAS', 'ELAS', 'LSVR','Adb','RF','Stacking']
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

plt.rcParams['figure.figsize'] = (8.0, 6.0)
plt.figure(12)
plt.plot(mz, mse,label='各个模型测结果的 MSE 比较',color = 'blue',linewidth=1)
plt.plot(mse,'ro')
plt.xlabel('模型',fontsize = 12)
plt.ylabel('MSE',fontsize = 12)
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False
plt.rcParams['figure.figsize'] = (8.0, 6.0)
plt.figure(12)
plt.plot(mz, mae,label='各个模型测结果的 MAE 比较',color = 'blue',linewidth=1)
plt.plot(mae,'ro')
plt.xlabel('模型')
plt.ylabel('MAE',fontsize = 12)
plt.grid(False)
plt.rcParams['figure.figsize'] = (8.0, 6.0)
plt.figure(8)
plt.plot(mz, mape,label='各个模型测结果的 MAPE 比较',color = 'blue',linewidth=1)
plt.plot(mape,'ro')
plt.xlabel('模型')
plt.ylabel('MAPE',fontsize = 12)
plt.grid(False)
plt.rcParams['figure.figsize'] = (8.0, 6.0)
plt.figure(12)
plt.plot(mz, R2,label='各个模型测结果的 R2 比较',color = 'blue',linewidth=1)
plt.plot(R2,'ro')
```

```
plt.xlabel('模型')
plt.ylabel('R2',fontsize = 12)
plt.grid(False)
#保存训练好的 stacking 模型
import pickle
saved_model = pickle.dumps(reg)
```

代码三：基于有限对称性、随机性生成初始结构、模拟退火算法寻找最优结构

```
#模拟退火寻找最优结构
def ObjFun(t):
    #设置粒子群找到的最优参数
    A=0.08985951
    p = 10.68708327
    q = 2.11183077
    N = 20
    epsilon = 1.8097
    r0=1
    #随机产生 xyz_i
    X = []
    Y = []
    Z = []
    for i in range(16):
        x = np.random.uniform(-8.449219,7.4615367) #Au32 给定-4.968377，4.968377 : -8.449219    7.461536
        X.append(x)
        #xt = -x
        xt = np.random.uniform(-8.449219,7.461536)
        X.append(xt)
        y = np.random.uniform(-8.716752,9.71338) #-8.716752 9.71338
        Y.append(y)
        #yt = -y
        yt = np.random.uniform(-8.7167527,9.71338)
        Y.append(yt)
        z = np.random.uniform(-9.433710,9.242134) #-9.433710    9.242134
        Z.append(z)
        #zt = z
        zt = np.random.uniform(-9.4337107,9.242134)
        Z.append(zt)
    X1 = pd.DataFrame(X)
    Y1 = pd.DataFrame(Y)
    Z1 = pd.DataFrame(Z)
    X1.index = Y1.index
    frame = [X1,Y1,Z1]
    result1 = pd.concat(frame, axis=1)
    xyz_i = result1.values.tolist()
    #xyz_i
```

```
    kulun_juzhen_i = get_coulombmatrix(xyz_i, largest_mol_size=None)
    tezhengzhi_juzhen= np.linalg.eig(kulun_juzhen_i)[0].tolist()[0:20]
    energy_gupta = V_r_m(A,p,q,N,epsilon,r0,xyz_i)
    tezhengzhi_juzhen.append(energy_gupta)
    tezheng_shuru = np.array(tezhengzhi_juzhen).reshape(1,-1)
    energy = reg.predict(tezheng_shuru)
     return energy,result1    #返回模型预测的能量以及对应的团簇位置结构

#判断函数
#返回 1 表示接受当前解，返回 0 表示拒绝当前
#当系统内能减少时（目标函数值减小，新解更理想），接收该解；
#当系统内能增加时（新解比旧解更差），以一定的概率接受当前解（当 T 变小时，probability 在减小，于是接受更差的解的概率值越小，退火过程趋于稳定）
#为当前解添加随机扰动
def Disturbance(low,high,x_old):
    if np.random.random()>0.5:
        x_new = x_old + (high - x_old) * np.random.random()
    else:
        x_new = x_old - (x_old - low) * np.random.random()
     return x_new

from mpl_toolkits.mplot3d import Axes3D#绘制三维图
#设置模拟退火参数
low = 0.3
high = 9.9
tmp = 1e5
tmp_min = 1e-3
alpha = 0.9
#初始化
x_old = (high-low) * np.random.random() + low
x_new = x_old + np.random.uniform()
value_old,a = ObjFun(x_old)
value_new = value_old
counter = 0
record_x = pd.DataFrame()
record_y = []
while(counter <= 1000):
    x_new = Disturbance(low,high,x_old)
    value_new,xyz_i = ObjFun(x_new)
    deltaE = value_new - value_old
    if Judge(deltaE,tmp)==1:
        value_old=value_new
        record_y.append(value_new)
        x_old=x_new
    if deltaE < 0:
```

```
            tmp=tmp*alpha #deltaE < 0 说明新解较为理想，继续沿着降温的方向寻找，减少跳出可能性；当温度减小，当前内能变化量的概率值会变小
        else:
            counter+=1
        if counter >=989:
            print(counter)
            a = counter-988
            file = open('./Au32_模拟退火_随机_题目范围//最优结构%a.txt'%a,'w')
            file.write(str(xyz_i))
            file.close()
            xx1 = pd.DataFrame(xyz_i)
            x1 = xx1.iloc[:,0]
            y1 = xx1.iloc[:,1]
            z1 = xx1.iloc[:,2]
            x = x1.values.tolist()
            y = y1.values.tolist()
            z = z1.values.tolist()
            fig = plt.figure()
            ax3d = Axes3D(fig)
            ax3d.scatter(x,y,z,c="b",marker="o")
            plt.savefig('Au32_模拟退火_随机_题目范围//最优结构%a.jpg'%a)
            plt.show()

energy_best_suiji = record_y[-10:]
#观察 x 的变化以及目标函数值的变化
length=len(record_y)
index=[i+1 for i in range(length)]
#plt.title('Average Return of Models',Fontsize = '20')
plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode_minus']=False
plt.rcParams['figure.figsize'] = (8.0, 6.0)
plt.xlabel('模拟退火算法寻优次数',Fontsize = '12')
plt.ylabel('原子团簇能量',Fontsize = '12')
plt.plot(index,record_y,linewidth = 1.3)
    #plt.grid(False)
plt.savefig(r"Au32_模拟退火_随机_题目范围//寻优图 1.jpg",dpi=400)
print(plt.show)

#将得到的最优的 10 个位置信息导入 xyz 文件中
import re
for i in range(1,11,1):
    filename = 'Au32_模拟退火_随机_题目范围//最优结构%i.txt'%i
    with open(filename, 'r') as f:
        content = f.readlines()
```

```
        for j in range(1,21,1):
            a = content[j][3:].strip()
            data_a = re.split('\s+',a.strip())
            xyz.append([float(j)    for j in data_a])
        result = pd.DataFrame(xyz)
        result = np.array(result)
        xyz = []
        with open('./Au32_模拟退火_随机_题目范围/Au_32_best_%i.xyz'%i,'w') as f_new:
            f_new.write('20'+'\n')
            f_new.write(str(energy_best_suiji[i-1])+'\n')
            for i in range(len(result)):
                f_new.write(str('Au')+''+str(result[i][0])+' '+str(result[i][1])+str(result[i][2])+'\n')
        f_new.close()
```

代码四：对 B45 生成 Tensoff 势能函数、建立 GRU 模型

```
#定义 Tensoff 势能函数
def V_r_m(mu,epilson,N,xyz_i):
    V_r = 0
    V_r_i =0
    V_i = 0
    r = np.array(r_ij(xyz_i))
    for i in range(1,N):
        for j in range(i+1,N+1):
            V_r = (mu/r[i-1,j-1])**12-(mu/r[i-1,j-1])**6
            V_r_i +=V_r
    V_i = 4*epilson*V_r_i
    return V_i

#建立 GRU 模型
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from math import sqrt
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.recurrent import GRU
import tensorflow as tf

def create_model(dropout=0.1,lr=0.001):
    model = Sequential()
    model.add(GRU(20,input_shape=(46,1), return_sequences=True))
    #model.add(GRU(18,input_shape=(window, input_size), return_sequences=False))
    model.add(Dropout(0.1))
    model.add(Dense(6,activation='sigmoid'))
    model.add(Dense(1))
    adam = Adam(lr=0.01, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

```
    model.compile(loss = 'mse',optimizer='adam')
    model.fit(X_train, y_train,validation_data=(X_test, y_test), epochs = 100, batch_size = 238)
    return model

#网格搜索调整参数
model = KerasRegressor(build_fn=create_model,verbose=1,epochs=50)
batch_size = [128,196,256]
dropout=[0.08,0.1,0.12,0.16,0.2]
lr = [0.001,0.005,0.01,0.015]
param_grid = dict(batch_size=batch_size,dropout=dropout,lr = lr)
grid=GridSearchCV(estimator=model,param_grid=param_grid,refit=False,cv=5,scoring='neg_mean_squared_error')#n_jobs
为-1 时表示使用并行计算占用所有计算资源，CV 表示交叉验证的折数
time_start = time.time()
grid_result = grid.fit(X_train,y_train) #进行网格搜索
time_end = time.time()
time_c= time_end - time_start
print('time cost', time_c/3600, 'h') #运行所花时间
print("Best: %f using %s" % (-grid_result.best_score_, grid_result.best_params_))#CV 的最佳平均得分及对应参数
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params): #输出每个参数配置下的得分、标差
    print("%f (%f) with: %r" % (-mean, stdev, param))
model.predict(X_test,y_test)
#在测试集上的预测
y_test_predict=model.predict(X_test)
draw=pd.concat([pd.DataFrame(y_test),pd.DataFrame(y_test_predict)],axis=1);
draw.iloc[:,0].plot(figsize=(12,6))
draw.iloc[:,1].plot(figsize=(12,6))
plt.axhline(y=0.0, color='r', linestyle='-',lw=1)
plt.legend(('real','predict'),loc='upper right',fontsize='15')
#plt.legend(('predict'),loc='upper right',fontsize='15')
plt.title("Test Data",fontsize='30') #添加标题
plt.show()
```