队伍编号	21090090037
题号	B

三维团簇的能量预测

摘 要

本文从分子结构出发，利用机器学习和粒子群算法分析团簇结构和能量之间的关系，找到相关特征去量化这种关系。根据附件中团簇坐标及能量信息，训练数据，建立团簇结构预测模型，结合粒子群算法，找到团簇全局最优结构，并对新型团簇进行结构预测。

针对问题一，根据附件中提供的金团簇Au_{20}的数据，进行数据预处理。我们用库伦矩阵将团簇中 N×3 维原子坐标转换为 N×1 维的矩阵特征值变量，结合这些原子衍生出的特征变量，利用多种机器学习算法，得到 19 个与团簇能量相关的特征变量，最终选择了线性回归模型作为我们的预测模型。通过模型预测，得到金团簇Au_{20}的全局最优结构，通过 VMD 对团簇结构进行可视化。

针对问题二，通过改进动态惯性权重、莱维飞行、差分进化变异、普通粒子学习、初始化结构对称化，我们改进了粒子群算法，并利用该算法初始化产生金团簇不同异构体。结合由问题一得到的线性回归预测模型，我们找到了金团簇Au_{32}的全局最优结构，通过 VMD 对团簇结构进行可视化。改变某些相关特征变量，利用我们的预测模型预测金团簇Au_{32}的能量，将其与未改变特征前的能量做比较，发现能量只有轻微偏差。因此，利用我们的优化算法求得的最优结构具有很好的稳定性。

针对问题三，根据附件中提供的硼团簇B_{45}-的数据，与问题一类似，我们用同样的方法对数据进行预处理，找到其库伦矩阵特征值向量，结合衍生特征变量，通过机器学习算法，得到 32 个与团簇能量相关的特征变量，最终选择了线性回归模型作为我们的预测模型。用模型预测得到硼团簇B_{45}-的全局最优结构，最后由 VMD 对硼团簇结构进行可视化。

针对问题四，与问题二类似，仍然利用我们改进的粒子群优化算法，产生硼团簇不同异构体。结合线性回归预测模型，预测出硼团簇B_{40}-的全局最优结构。稳定性分析方法与问题二保持不变，得出的结论是我们的最优化结构的稳定的，由 VMD 对硼团簇B_{40}-结构进行可视化。

关键词：改进粒子群算法、机器学习、团簇、结构预测

目录

一、问题提出

1.1 背景

团簇是多原子、分子通过物理或者化学结合力组成的集聚体，其物理和化学性质随所含的原子数目变化而变化，势能面非常复杂。传统方法通过数值迭代求解团簇能量效率低，且随着原子数的增加，计算时间呈指数增长，如何加速搜索团簇结构最优及预测过程成为相关研究人员亟待解决的问题。因此，本文尝试使用机器学习的方法，训练出团簇结构和能量的关系，再结合粒子群算法，从而预测团簇全局最优结构。此外，经查阅资料，我们发现，越稳定的物质通常在几何构型上也具有对称性。因此，我们考虑借助参数敏感性分析来探求三维团簇能量预测全局最优结构的稳定性。

本文在团簇结构和能量预测的基础上，通过机器学习和粒子群算法构建三维团簇能量预测的数学模型，有利于预测新型团簇的全局最优结构，有利于发现新型材料的结构和性能，对加快推动我国化学分子研究领域的发展有重要意义。

1.2 问题重述

（1）针对金属团簇，附件给出了 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；

（2）在问题 1 的基础上，请你们设计算法，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；

（3）针对非金属团簇，附件给出了 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;

（4）在问题 3 的基础上，请你们设计算法，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。

二、基本假设

（1）假设每个团簇中原子间特征无强烈影响。

（2）假设外部环境对团簇能量无影响。

（3）假设随机构造出的团簇结构能够理想化存在。

（4）假设数据的预处理对实际结果无显著影响错误无丢失。

三、符号说明

表 1　符号说明表

符号	意义	符号	意义
N	团簇原子个数	$scale$	经典步长
i,j	第i,j个原子	K	缩放因子
Z_i	第i个原子的核电荷数	c	学习因子
σ_{st}	节点对依赖值	r	0-1 之间的随机数
ω^0	初始惯性权重	r^j	随机整数
e^{t}	粒子群的进化速度	m	种群大小
R_i^k	由所有个体最优化形成的组合体	CR	交叉概率常数

四、问题分析

4.1 问题一的分析

为预测出金团簇Au_{20}的最优结构，我们对附件中Au_{20}的原子数据进行特征处理，根据数据处理得到的特征向量以及原子自身演变出来得到的特征向量，对比多个模型的交叉验证结果，进而选取一个较为适合的模型做最终的预测。采用的模型包括：通过机器学习的线性回归（$Linear\ Regression, LR$）、岭回归（$Ridge\ Regression, RR$）、线性支持向量机（$Linear\ SVM, LSVM$）、支持向量机（SVM）、自适应提升方法（$AdaBoostRegressor$），梯度提升决策树（$GradientBoostingRegressor$）、$XGBRegressor$。考虑模型可接受性，最终预测模型选择线性回归模型。通过该模型对该模型优化预测金团簇Au_{20}全局最优结构，最后由VMD软件对该最优结构进行展示。

4.2 问题二的分析

在问题一的基础上，我们加入了改进的粒子群算法，并由该算法初始种群产生对称的金团簇Au_{20}异构体（一般来说，团簇对称时能量相对来说也会更低），通过该算法全局搜索找到最优结构，由VMD展示。通过改变自变量，分析团簇的稳定性。

4.3 问题三的分析

与问题一类似，我们分析了硼元素的特征向量，使用不同的机器学习预测方法，最终得到硼团能量的数学预测模型。通过该数学模型对硼元素进行全局最优结构预测，由VMD对预测到的最优结构进行展示。

4.4 问题四的分析

与问题二类似，在问题三的基础上，我们同样利用粒子群算法对得到的数学模型进行全局寻优，找到最优解，其最优结构由VMD展示。通过改变相关特征变量，我们发现其有较好的稳定性。

五、模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 数据预处理

1. 构造特征变量：

附件提供了 1000 个金团簇的原子数、能量及坐标数据信息，但是实际上只有 999 个有效数据，因此这里将未显示的数据行不做考虑。考虑到团簇的原子坐标是三维欧氏空间的相对坐标位置，不便于将其直接纳入数学预测模型作为特征变量。因此通过变换之后产生如下若干衍生特征变量：

（1）库伦矩阵方法：通过库伦矩阵计算公式将团簇的 N×3 维的坐标矩阵转换为 N×N 维的库伦矩阵，本题中金（Au）的核电荷数维 79，故所有$Z_i = 79, i \in (1,79)$。为满足机器学习模型对特征变量输入要求，继续对库伦矩阵进行降维。为尽可能多的表现矩阵的信息，通过计算矩阵的特征值(N×1 维)来代替矩阵信息。即库伦矩阵的特征值作为新的特征变量[1]。

$$C_{ij} = \begin{cases} 0.5Z_i^{2.4}, & \forall i = j \\ \dfrac{Z_i Z_j}{R_i - R_j}, & \forall i \neq j \end{cases}$$

（2）距离矩阵指标：团簇结构的结构稳定性必然和其键长、原子之间的距离、最大距离、最小距离等有一定联系。因此，通过计算所有原子之间的距离可

以正对每个样本团簇得到一个距离矩阵。通过聚合计算距离矩阵的平均距离、中位数距离、最小距离、最大距离纳入为特征变量。距离计算公式采用欧式距离：

$$r_{ij} = \sqrt{\left((x_i, y_i, z_i) - (x_j, y_j, z_j)\right)^2}$$

（3）网络结构指标:将团簇的三维结构可以看作一个网络结构数据，即可通过网络结构指标进一步描述结构特征。因此，通过距离矩阵构建邻接矩阵，当原子之间的某两个原子之间的距离大于某个指标即该原子间存在连接的无向边（本报告采用平均矩阵）。通过邻接矩阵进而得到其网络结构，计算相应网络分析指标：

网络密度用于刻画网络中节点间相互连边的密集程度，网络密度越大说明网络越复杂。计算公式如下：

$$\text{网络密度} = \frac{\text{实际边数}}{\text{理论边数}} = \frac{E}{C_N^2}$$

度中心性衡量的是节点对促进网络传播过程中发挥的作用，计算公式如下：

$$C_D(i) = \frac{k_i}{N-1}$$

接近中心性度量的思想是如果一个节点和其它很多节点都很接近，则节点处于网络的中心位置，计算公式如下：

$$C_{Cl}(i) = \frac{N-1}{\sum_j dist(i,j)}$$

中介中心性指的是一个结点担任其它两个结点之间最短路的桥梁的次数，计算公式如下：

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

（4）原子中对称函数：为了更加有效的描述物质的势能面，团簇结构的对称性也需要考虑到特征中。原子中心对称函数是一组径向和角函数，可以用于描述原子在截断半径r_{cut}内的配对特征。本报告采用如下的径向函数：

$$G_i^1 = \sum_j f_c(r_{ij})$$

$$G_i^2 = \sum_j e_c^{\left(-\eta(r_{ij}-r_s)^2\right)} f_c(r_{ij})$$

上式公式中r_{ij}是原子i和j之间的欧式距离，η, r_s, r_{cut}为可调整的参数，本报告中采用的参数设置为：$\eta = 1, r_s = 3, r_{cut} = 6$。$f_c(r_{ij})$是截断函数，定义如下：

$$f_c(r_{ij}) = \begin{cases} \frac{1}{2}\left(\cos\left(\frac{\pi r_{ij}}{r_{cut}}\right) + \right) & , r_{ij} > r_{cut} \\ 0 & , r_{ij} > r_{cut} \end{cases}$$

（5）最终的构造的数据变量如下表：

表 2　问题一的变量属性表

变量名称	属性名
X0-X19	库伦矩阵特征值
X20-X23	距离矩阵指标
X25-X29	网络结构描述指标
X30-X72	原子中对称函数G^1, G^2

2. 离群值处理：

通过观察目标变量及能量分布，分布图如下。将$Au20$能量值大于-1530 的确定为离群值（$Au20$样本中共一个离群值）。本模型通过删除的方式处理离群值。

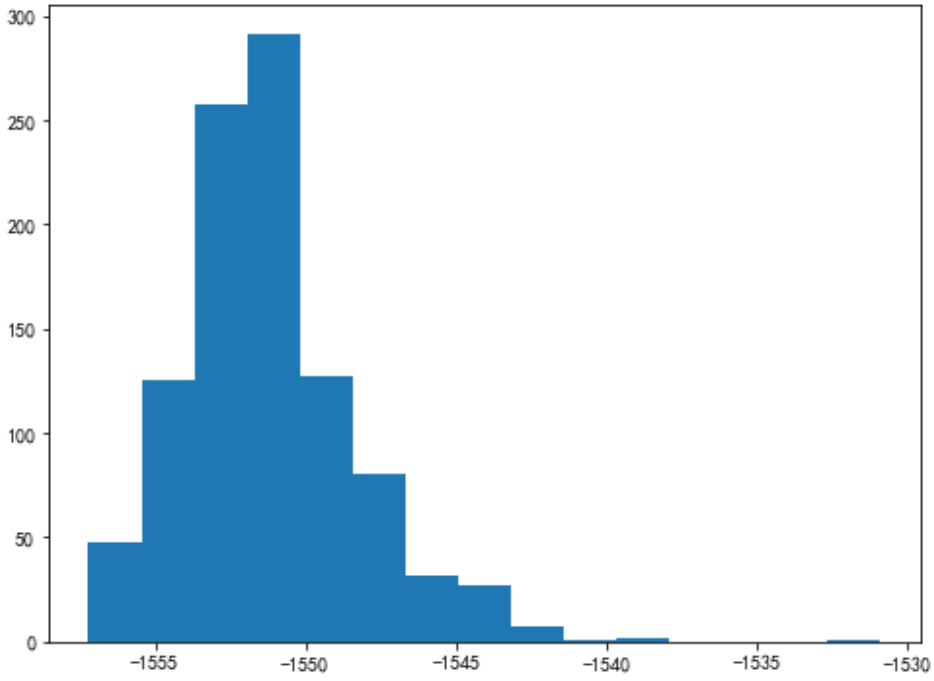

图 1　Au20 能量值分布直方图

3. 数据标准化：

考虑到衍生特征变量存在不同量纲和单位的情况，因此对所有特征变量进行数据标准化，将原始数据映射到（0，1）区间内，标准化公式如下：

$$X^* = \frac{X - \min\limits_{i\in(1,n)} X_i}{\max\limits_{i\in(1,n)} X_i - \min\limits_{i\in(1,n)} X_i}$$

4. 主成分降维：

通过构造特征变量一共生成了 72 个变量，由于 Au20 只有 997 个样本，故可能存在“维度爆炸”问题。因此，通过主成分方法构建“综合指标”进行降维处理，本报告按照累积贡献度达到 90%的标准进行主成分选择。新的主成分表示如下，A 为特征变换矩阵，Z 为主成分变量，X 为原特征变量：

$$Z = A^T X$$

5. 最终数据结构：

最终结果共有 19 个特征变量，997 个样本纳入到机器学习模型中。

5.1.2 机器学习预测模型

1. 评估准则：

对模型拟合效果的评估方法较多，本报告采用回归问题中较为常见的R^2决定系数进行模型评价，当然，最终预测模型的选择还需要参考多方面因素考虑。决定系数的评价只作为客观评价的参考标准，不作为最终的决定因素。计算公式如下：

$$R^2 = 1 - \frac{\sum\left(Y - \hat{Y}\right)^2}{\sum(Y - \bar{Y})^2}$$

2. 交叉验证：

由于样本量的原因，首先采用交叉验证的方式对多种机器学习算法进行比较。本报告采用 k=10 折交叉验证。

交叉验证结果如下表：

表 3　6 种机器学习模型对 Au20 预测的交叉验证评估结果

LR	RR	LSVM	SVM	XGBoost	AdaBoost
0.604	0.604	0.597	0.684	0.692	0.576
0.640	0.640	0.635	0.757	0.728	0.620
0.629	0.629	0.608	0.723	0.683	0.580
0.582	0.583	0.603	0.744	0.668	0.604

0.642	0.642	0.645	0.731	0.675	0.615
0.603	0.603	0.596	0.713	0.707	0.629
0.596	0.596	0.593	0.709	0.664	0.603
0.645	0.645	0.616	0.724	0.725	0.667
0.651	0.651	0.645	0.742	0.716	0.630
0.602	0.602	0.572	0.686	0.628	0.636
0.615	0.615	0.574	0.713	0.666	0.606
0.644	0.644	0.632	0.712	0.716	0.629
0.627	0.627	0.580	0.724	0.675	0.632
0.633	0.633	0.600	0.694	0.696	0.613
0.607	0.607	0.632	0.719	0.692	0.617
0.631	0.631	0.640	0.739	0.728	0.624
0.598	0.598	0.567	0.664	0.676	0.592
0.566	0.566	0.551	0.674	0.690	0.578
0.600	0.600	0.591	0.696	0.672	0.578
0.634	0.634	0.603	0.711	0.658	0.593

通过绘制 6 个模型的 10 折交叉验证结果的箱线图可以发现，黑箱算法（$SVM/XGBoost/AdaBoost$）普遍比白箱算法（LR，RR）预测效果较好。但白箱算法的预测效果的平均值(中位数)也在 0.68 附近，且预测效果的稳定性更强，多次预测结果的分布集中。在应用中完全可以接受。黑箱算法虽然有着较高的预测效果，但模型解释较难，且预测结果分布离散，不稳定。

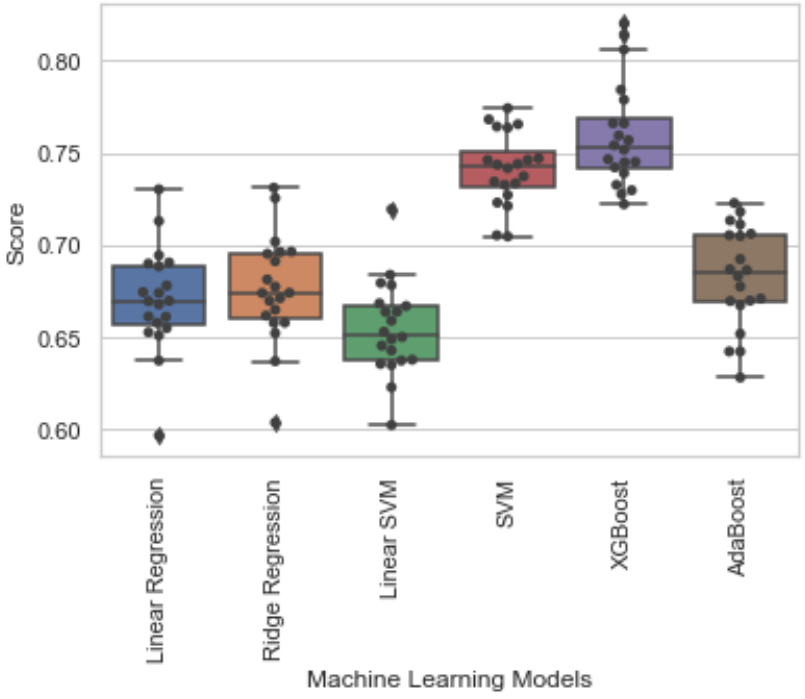

图 2　6 种机器学习模型对 Au20 预测的交叉验证结果箱线图

3. 最终预测模型结果：

考虑模型可解释性，最终预测模型选择线性回归模型。按照 7：3 的比例，将数据集划分为训练集和测试集。通过线性回归模型，在训练集上拟合模型，在测试集上评估模型。拟合得到的模型结果如下：

$$\begin{aligned} Y = {} & 0.151Z_1 - 0.043Z_2 + 0.051Z_3 - 0.0117Z_4 - 0.047Z_5 + 0.063Z_6 + 0.0148Z_7 \\ & + 0.027Z_8 - 0.0393Z_9 + 0.011Z_{10} + 0.204Z_{11} + 0.236Z_{12} \\ & - 0.0129Z_{13} + 0.198Z_{14} + 0.147Z_{15} - 0.046Z_{16} - 0.010Z_{17} \end{aligned}$$

上式模型公式中，自变量为特征变量的主成分。模型在训练集的决定系数为：0.66，残差平方和为：0.34；在测试集的决定系数为：0.62 残差平方和为：0.38。模型的残差分析：

通过绘制残差图可以发现，由于数据中存在异常值，所以可以发现残差图中有一处明显的凸起。但从整体而言，无异常现象。且通过参加检验服从正态分布，则满足高斯马尔科夫假设，线性模型结果可行。

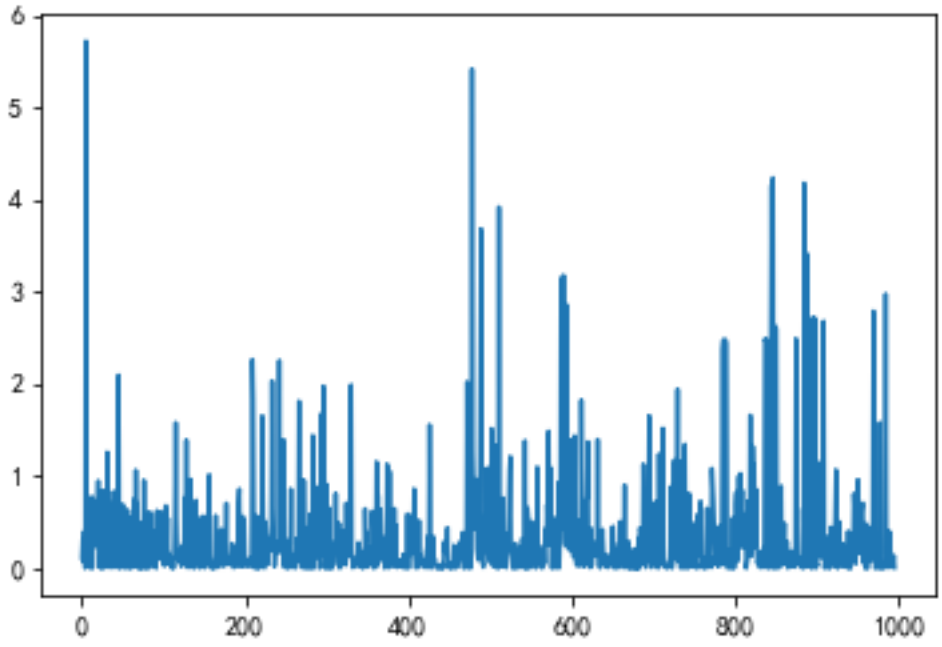

图 3 Au20 线性回归模型预测残差图

5.1.3 预测 Au20 全局最优结构

1. 预测算法

采用粒子群算法（PSO）来对 Au20 搜索全局最优结构。PSO 算法是一种复杂适应性系统的理论，此处采用的粒子群算法通过随机构造产生初始粒子[2]。

性能测试

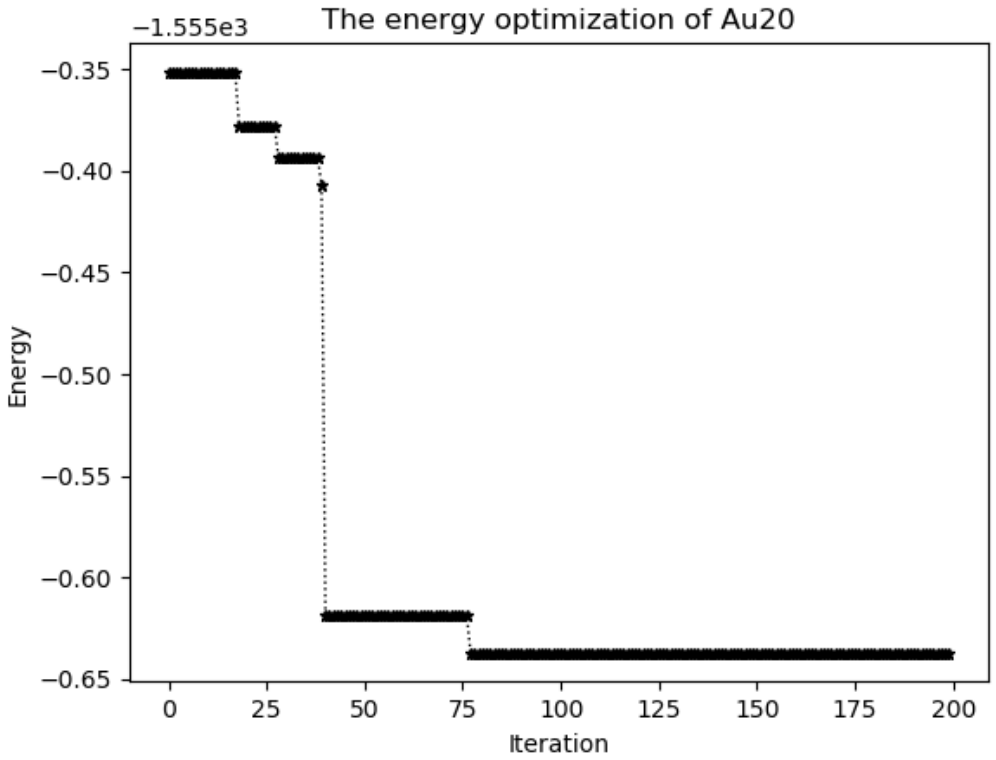

图 4　粒子群算法对 Au20 最优化结构预测的收敛过程

从$Au20$最优化结构迭代优化图可以发现，粒子群算法在经过差不多 40 次迭代之后，就有一个较大的提升，而在 75 次迭代之后便已经找到了模型的最优解。从算法性能上来说，可以进一步采纳为最优化求解的算法。

2.　预测结果

从最优化迭代结果可以看到，通过粒子群算法得到的最小化能量值为：-1555。通过VMD软件对预测的$Au20$团簇结构进行可视化绘制。结果如下图：

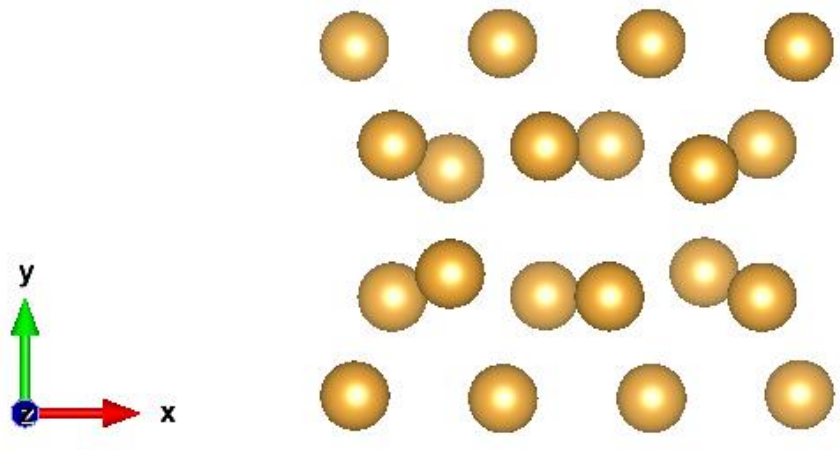

图 5　预测最优化的 Au20 结构图

5.2 问题二模型的建立与求解

5.2.1 算法设计

1. 算法改进：

由于常见的 PSO 算法在问题求解上存在较弱的鲁棒性问题。因此，本报告对 PSO 算法进行了一系列的改进，使得改进后的 PSO 算法能够较好的适应本问题的求解[3]。

（1）改进惯性权重。惯性权重对算法性能有重要的影响，惯性权重较大时有利于全局搜索，惯性权重较小时有利于算法的局部开发，从而加快收敛速度，因此，我们这里采用如下的惯性权重改进[4]：

$$\omega^t = \omega^0 - 0.5e^t + 0.1\alpha^t$$

其中ω^0为初始惯性权重，一般取值为 0.9，新引入变量e, α，对于第t代，

$$e^t = \frac{Gbest^t}{Gbest^{t-1}},$$

$$\alpha^t = \frac{mGbest^t}{\sum_{k=1}^{m} P_k best^t}$$

这里 m 为种群大小，e^t是粒子群的进化速度，当经过若干次迭代后，e^t值保持为 1，则说明算法停止或找到最优。α^t是粒子聚合度，用来描述当前粒子的聚散性，值越大表明粒子越分散。$e^t \in (0,1], \alpha^t \in (0,1]$，所以惯性权重$\omega^t \in (\omega^0 - 0.5, \omega^0 + 0.1]$。

（2）$Levy$飞行。$Levy$飞行是一个随机移动过程，其移动距离结合了短距离搜索和偶尔的长距离搜索。步长$S=\frac{u}{|v|^{\frac{1}{\beta}}}$,$\beta$是$Levy$指数，$u,v$服从$u\sim N(0,\ \sigma_u^2)$，$v\sim N(0,\ \sigma_v^2)$正态分布。步长计算公式[4]为：

$$stepsize\ =\ scale\times S$$

控制参数为ε，如果随机值小于ε，使用基本 PSO 算法更新速度和位置。否则，用以下公式更新：

$$V_i^{k+1}=\omega V_i^k+c_1r_1\left(P_i^k-X_i^k\right)+c_2r_2\left(G^k-X_i^k\right)+c_3r_3\left(R_i^k-X_i^k\right)\times stepsize$$

$$X_i^{k+1}=X_i^k+V_i^{k+1}$$

（3）加入差分进化变异。这里我们引用的差分化个体变异操作[4]。按照如下公式对 t 代的个体X_i^t进行变异操作得到变异个体：

$$V_i^{t+1}=X_{r_i}^t+K(X_{r_2}^t-X_{r_3}^t)$$

以增强种群的多样性，按照以下公式对X_i^t和得到的变异个体V_i^{t+1}实施交叉操作，得到了个体u_i^{t+1}，即：

$$u_{i,j}^{t+1}=\begin{cases}v_{i,j}^{t+1}, & if(rand(j)\le CR)\ or\ j=rand(i)\\ x_{i,j}^t, & otherwise\end{cases}$$

根据下面公式对试验个体u_i^{t+1}和X_i^t进行比较竞争，对于最小化问题，选择目标函数值较低的个体作为子代，即如下：

$$x_i^{t+1}=\begin{cases}u_i^{t+1}, & if(u_i^{t+1})<f(x_i^t)\\ x_i^t, & otherwise\end{cases}$$

（4）加入对普通粒子的学习[2]。我们采用随机学习算子改进规则对 PSO 速度公式进行了更新，具体形式如下：

$$V_i^{k+1}=\omega V_i^k+c_1r_1\left(P_i^k-X_i^k\right)+c_2r_2\left(G^k-X_i^k\right)+c_3r_3\left(R_i^k-X_i^k\right)$$

（5）初始化结构对称化。当无限制初始化时，完全随机生成的初始值团簇结构不存在任何对称性。但是在团簇结构中，存在一定的原子对称性限制。因此，在初始化结构时，算法考虑了原子的对称性约束条件。

2. 参数设置：

通过问题一中对$Au20$的最优化结构预测，进一步对$Au32$团簇进行最优化搜索。此处我们考虑了$Au20$已经得到的最优结构，在该结构上进一步引入新的原

子，通过上述改进的粒子群算法对新引入的原子进行最优化搜索。相关粒子群算法参数设置如下：

$$\omega^0 = 0.9, \mathrm{e}^{\mathrm{t}} = 1, \sigma_v = \sigma_u = 1$$

3. 测试函数：

这里我们使用$Rastrigin's$函数和$Schaffer$函数测试我们的算法。

$Rastrigin's$函数：

$$Ras(x) = 20 + x_1^2 + x_2^2 - 10(cos2\pi x_1 + cos2\pi x_2)$$

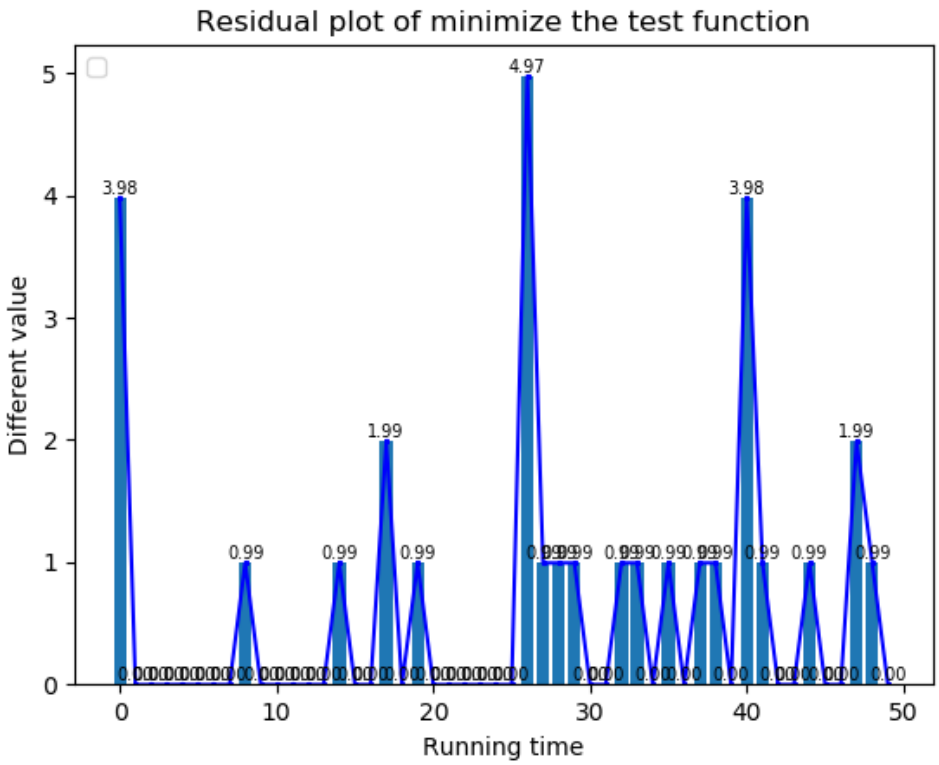

图 6　改进算法测试$Rastrigin's$函数残差

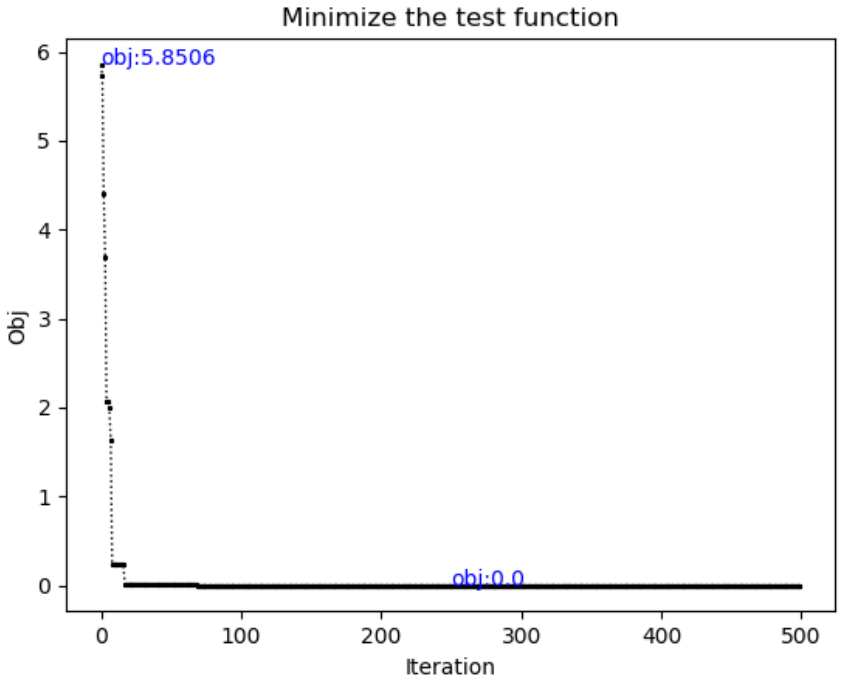

图 7　改进算法测试$Rastrigin's$函数最小值

$Schaffer$函数：

$$f(x) = -0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{\left(1 + 0.001(x_1^2 + x_2^2)\right)^2}$$

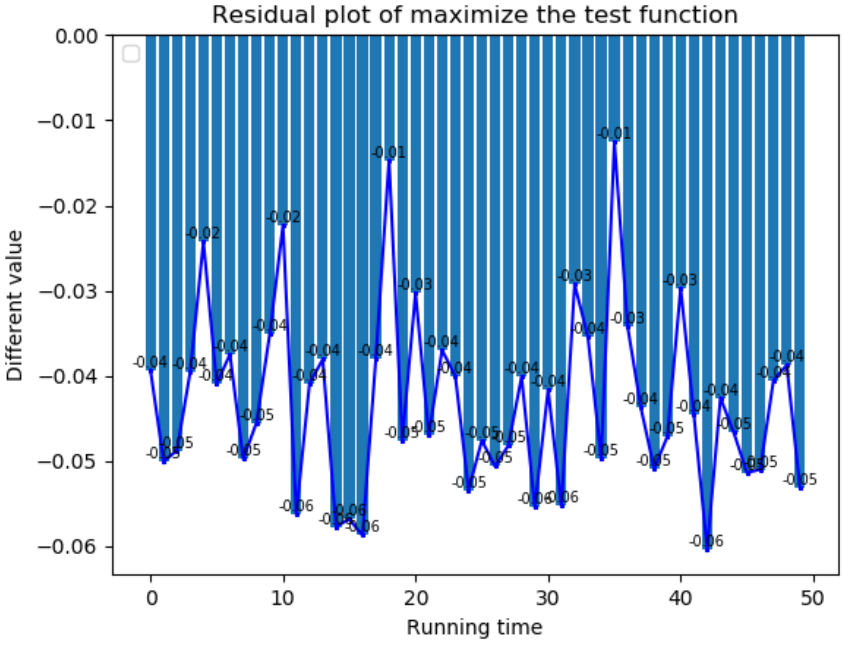

图 8　改进算法测试$Schaffer$函数残差

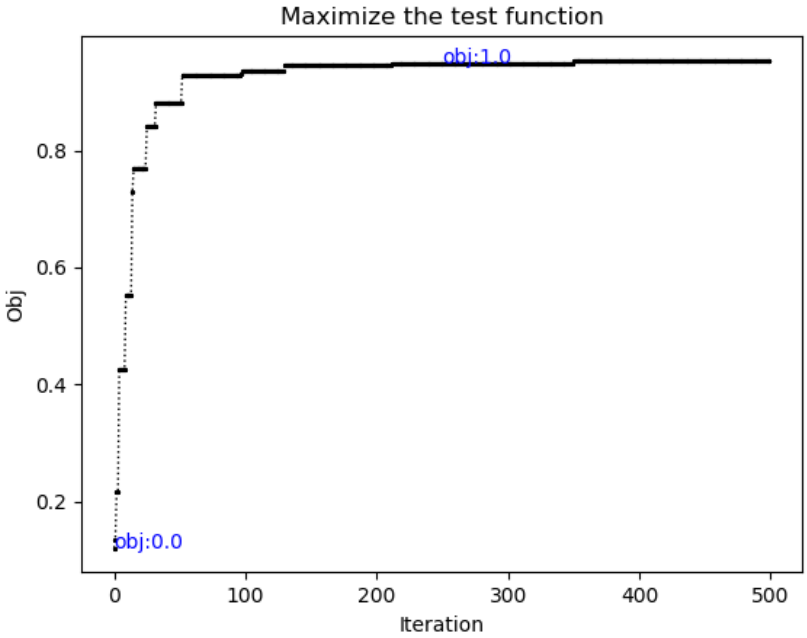

图 9　改进算法测试$Schaffer$函数最大值

4. 性能测试：

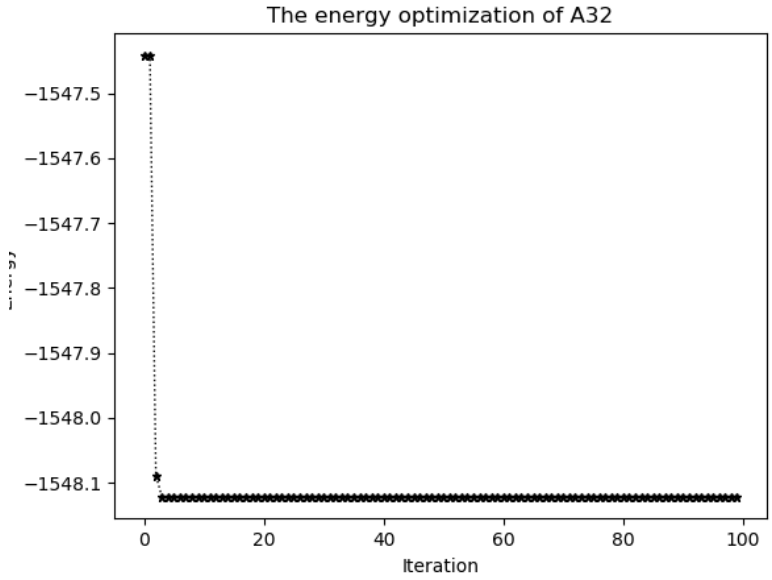

图 10　粒子群算法对 Au32 最优化结构预测的收敛过程

通过对改进后的粒子群算法性能进行评估，结果如上图。可以发现，经过了较少的迭代次数算法便已收敛。说明了算法的可行性。

5. 计算结果：

通过改进的粒子群算法得到$Au32$最低能量为：-1548；通过VMD软件可视化预测得到的$Au32$坐标点得到结构图如下：

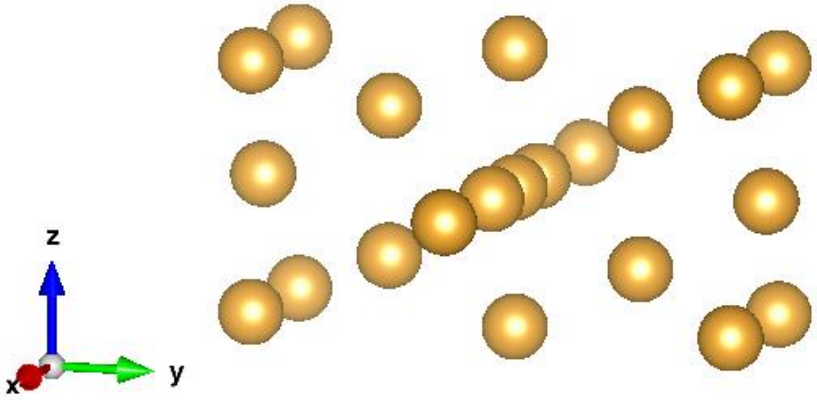

图 10　预测最优化的 Au32 结构图

5.2.2 金团簇稳定性分析

物质在对称的情况下一般都具有很好的稳定性，通过观察分析，我们求出的最优结构具有一定的对称性，因此，可以初步判定达到稳定。还有一种方法可以判断无知的稳定性，就是改变某些相关特征变量，利用我们的预测模型预测金团簇Au_{32}的能量，将其与未改变特征前的能量做比较，这里不做讨论。

5.3 问题三模型的建立与求解

5.3.1 数据预处理

1. 构造特征变量：

附件提供了 3751 个金团簇的原子数、能量及坐标数据信息。通过变换之后产生如下若干衍生特征变量。（由于本题基本与第一题相似，故构造特征过程同问题一类似，此处不再重述。）

最终的构造的数据变量如下表，共有 3751 个 B45 团簇样本，147 个特征变量。

表 4　问题三的变量属性表

变量名称	属性名
X0-X44	库伦矩阵特征值
X45-X48	距离矩阵指标
X49-X53	网络结构描述指标
X54-X147	原子中对称函数G^1, G^2

2. 离群值处理：

通过观察目标变量及能量分布，分布图如下。将 B45 能量值大于-92000 的确定为离群值（B45 样本中共 2 个离群值）。本模型通过删除的方式处理离群值。

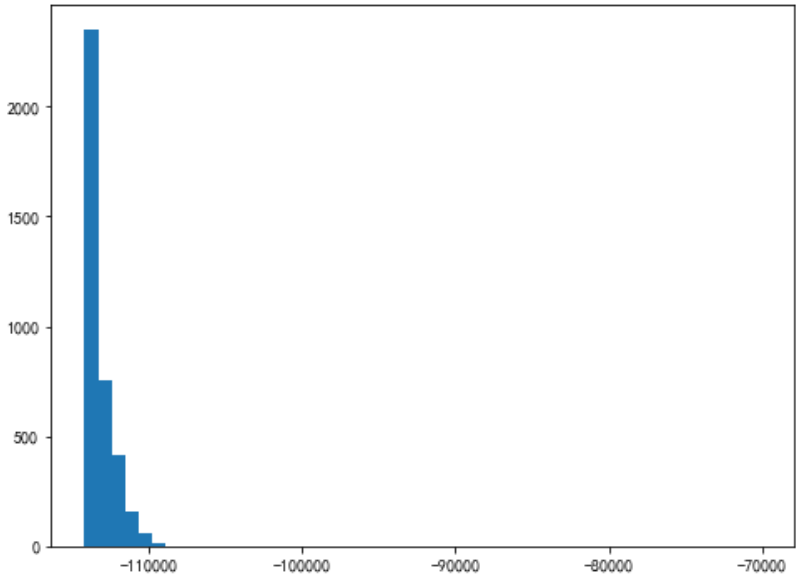

图 11　B45-能量值分布直方图

3. 数据标准化：

同问题一类似，此处不再赘述。

4. 主成分降维：

同问题一类似，此处不再赘述。

5. 最终数据结构：

最终结果共有 32 个特征变量，3749 个样本纳入到机器学习模型中。

5.3.2 机器学习预测模型

1. 评估准则：

同问题一类似，此处不再赘述。

2. 交叉验证：

首先采用交叉验证的方式对多种机器学习算法进行比较。本报告采用 k=10 折交叉验证。

交叉验证结果如下表：

表 5　6 种机器学习模型对 B45-预测的交叉验证评估结果

LR	RR	LSVM	SVM	XGBoost	AdaBoost
0.907	0.907	0.895	0.934	0.907	0.864
0.880	0.880	0.858	0.898	0.889	0.839

0.901	0.901	0.880	0.933	0.910	0.840
0.903	0.903	0.880	0.940	0.914	0.854
0.909	0.909	0.883	0.939	0.902	0.866
0.888	0.888	0.865	0.913	0.886	0.825
0.892	0.892	0.880	0.912	0.890	0.827
0.886	0.886	0.859	0.911	0.891	0.837
0.914	0.914	0.893	0.940	0.929	0.853
0.900	0.900	0.876	0.941	0.863	0.837
0.886	0.886	0.866	0.912	0.875	0.849
0.908	0.908	0.894	0.936	0.926	0.861
0.895	0.895	0.866	0.923	0.895	0.839
0.890	0.890	0.878	0.913	0.887	0.843
0.900	0.900	0.883	0.937	0.862	0.860
0.881	0.881	0.867	0.910	0.879	0.827
0.896	0.896	0.881	0.925	0.902	0.862
0.909	0.909	0.885	0.941	0.925	0.856
0.885	0.885	0.858	0.904	0.885	0.834
0.902	0.902	0.891	0.940	0.913	0.851

同样通过绘制 6 个模型的 10 折交叉验证结果的箱线图可以发现，当样本量较大、且特征变量足够时（相对于$Au20$），各个机器学习模型的预测效果明显较优（决定系数都大于 0.8）。黑箱算法（$SVM/XGBoost$）普遍比白箱算法（LR，RR）预测效果较好，但也存在较差效果的$AdaBoost$。但白箱算法的预测效果的平均值(中位数)也在 0.9 附近，且预测效果的稳定性更强，多次预测结果的分布集中。优于其可解释性的优势，故在应用中完全可以接受稍逊一筹的预测效果。

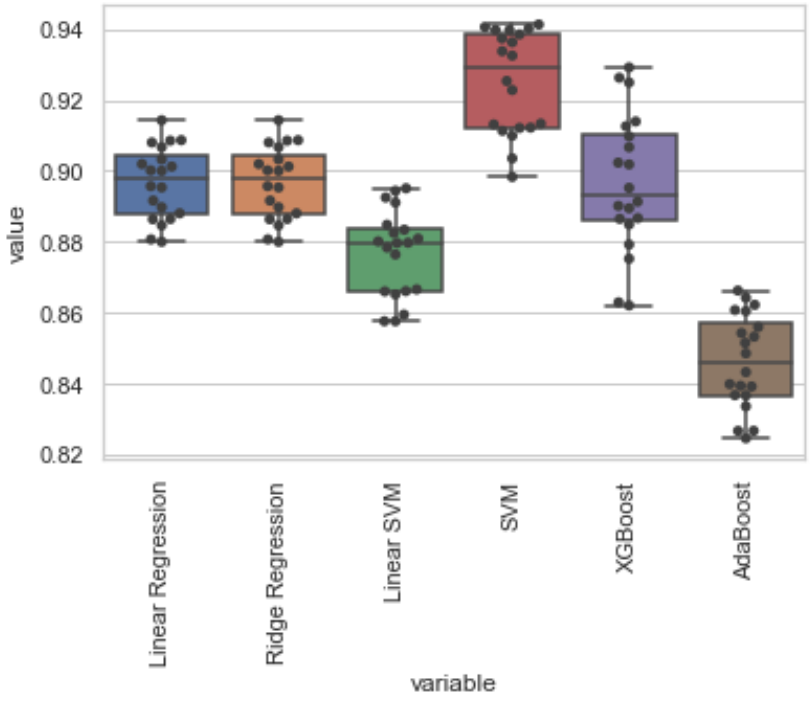

图 12　6 种机器学习模型对 B45-预测的交叉验证结果箱线图

3.　最终预测模型结果：

考虑模型可解释性，最终预测模型选择线性回归模型。按照 7：3 的比例，将数据集划分为训练集和测试集。通过线性回归模型，在训练集上拟合模型，在测试集上评估模型。拟合得到的模型结果如下：

$$\begin{aligned} Y = &-0.097Z_1 + 0.004Z_2 + 0.082Z_3 + 0.116Z_4 - 0.144Z_5 + 0.026Z_6 + 0.011Z_7 + 0.047Z_8 \\ &- 0.009Z_9 - 0.009Z_{10} - 0.077Z_{11} + 0.035Z_{12} - 0.073Z_{13} + 0.025Z_{14} \\ &- 0.065Z_{15} + 0.039Z_{16} - 0.125Z_{17} - 0.136Z_{18} + 0.021Z_{19} + 0.051Z_{20} \\ &- 0.047Z_{21} + 0.046Z_{22} - 0.076Z_{23} - 0.116Z_{24} + 0.033Z_{25} + 0.133Z_{26} \\ &+ 0.072Z_{27} + 0.024Z_{28} + 0.007Z_{29} - 0.014Z_{30} + 0.074Z_{31} \end{aligned}$$

上式模型公式中，自变量为特征变量的主成分。模型在训练集的决定系数为：0.91，残差平方和为：0.89；在测试集的决定系数为：0.1 残差平方和为：0.1。模型的残差分析：

通过绘制残差图可以发现，可以发现残差图中有一处明显的凸起，这是因为数据中存在一个明显的离群点，故对利群点的处理是非常有必要的。但从整体而言，无异常现象，整体的预测效果都较好。且通过参加检验服从正态分布，则满足高斯马尔科夫假设，线性模型结果可行。

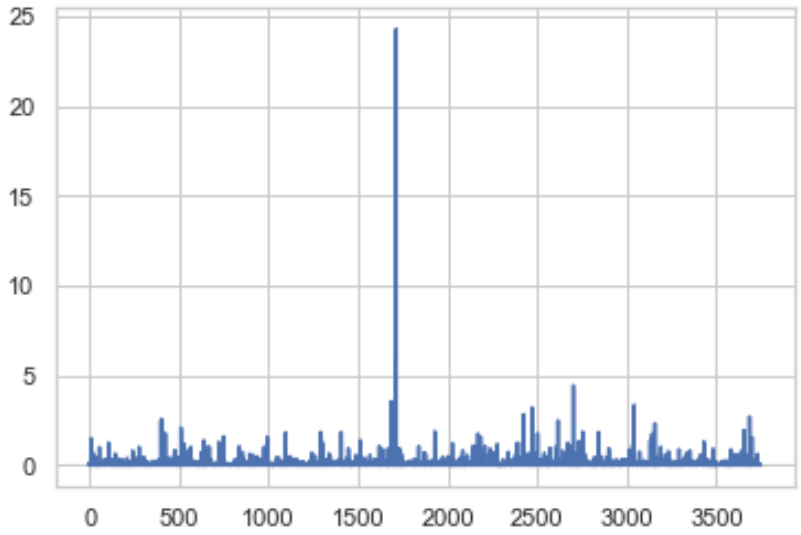

图 13　B45-线性回归模型预测残差图

5.3.3　预测 B45 全局最优结构

1. 预测算法：

同问题一类似，此处不再赘述。

2. 性能测试：

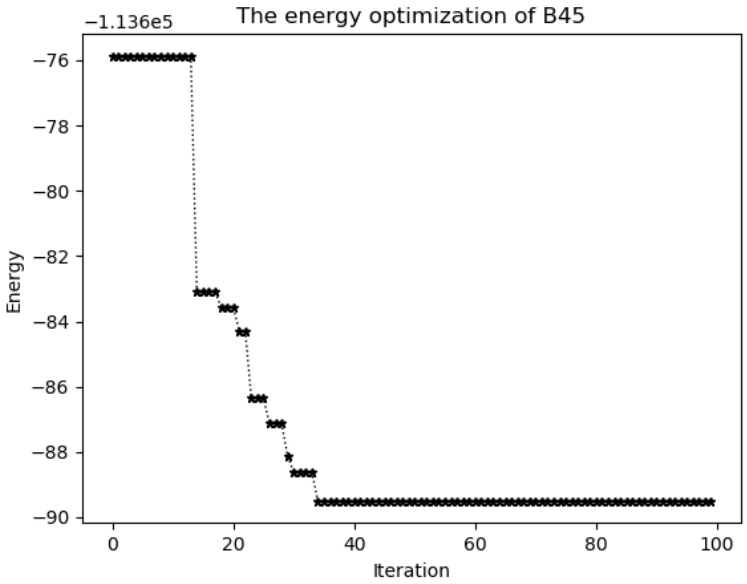

图 14　粒子群算法对 B45-最优化结构预测的收敛过程

从 B45 最优化结构迭代优化图可以发现，粒子群算法在经过差不多 20 次迭代之后，就有一个较大的提升，而在 35 次迭代之后便已经找到了模型的最优解。

3. 预测结果

从最优化迭代结果可以看到，通过粒子群算法得到的最小化能量值为：-1555。通过 VMD 软件对预测的 B45 团簇结构进行可视化绘制。结果如下图：

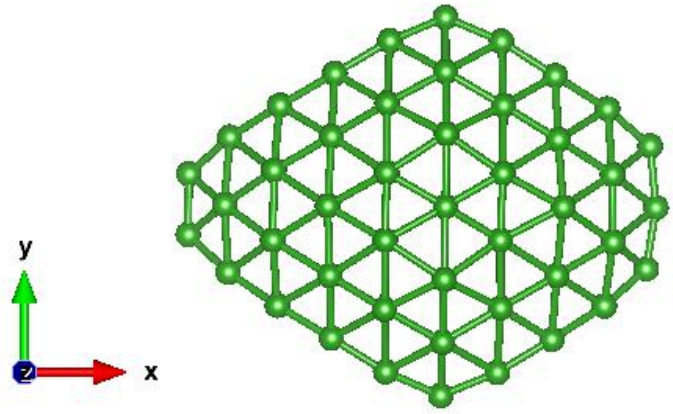

图 15　预测最优化的 B45-结构图

5.4 问题四模型的建立与求解

5.4.1 算法设计

1. 算法改进：

同问题二介绍，此处不再赘述。

2. 参数设置：

同问题二介绍，此处不再赘述。

3. 性能评价：

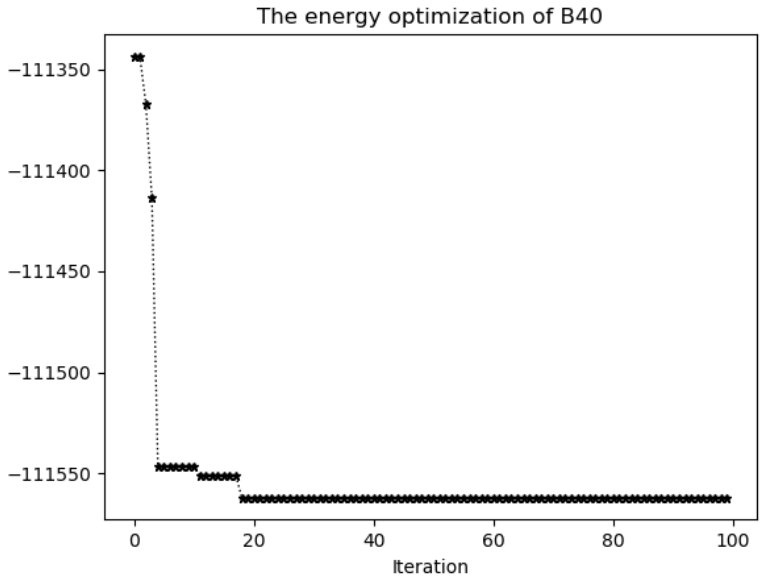

图 16　粒子群算法对 B40 最优化结构预测的收敛过程

通过对改进后的粒子群算法性能进行评估，结果如上图。可以发现，经过了较少的迭代次数算法便已收敛。说明了算法的可行性。

4. 计算结果：

通过改进的粒子群算法得到 B40-最低能量为：-111560；通过 VMD 软件可视化预测得到的 B40-坐标点得到结构图如下：

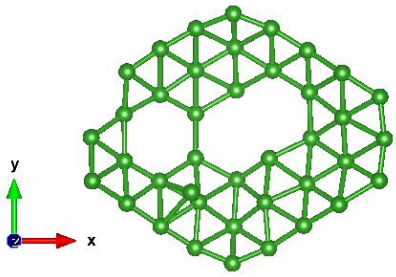

图 17　预测最优化的 B40 结构图

5.4.2 硼团簇稳定性分析

物质在对称的情况下一般都具有很好的稳定性，通过观察分析，我们求出的最优结构具有一定的对称性，因此，可以初步判定达到稳定。还有一种方法可以判断无知的稳定性，就是改变某些相关特征变量，利用我们的预测模型预测硼团簇 B45 的能量，将其与未改变特征前的能量做比较，这里不做讨论。

六、评价与改进

6.1 模型的评价

6.1.1 模型优点

（1）通过库伦矩阵特征值、网络结构指标、距离矩阵指标、原子中对称函数构建特征，尽可能多的考虑团簇自身物理和化学性质。

（2）采用主成分分析可以尽可能的消除“维度爆炸”的问题，并且可以降低模型过拟合的风险。

（3）通过交叉验证的方式可以降低数据量问题产生的影响。

（4）采用线性回归模型进行预测能够显示的表现特征与能量之间的关系，且线性回归的预测模型便于解释。

（5）点群对称性约束的改进机制可以大大减少不必要的计算，提高收敛速度。

6.1.2 模型缺点

（1）将三维坐标点映射到特征变量之后，难以将特征变量逆过程推导为坐标点。

（2）线性回归模型的预测效果相对于其它模型来说较差。

（3）预测模型的评估标准较为单一，容易产生偏差。

（4）对团簇结构的预测没考虑键对之间的影响。

6.2 模型的改进

（1）改进特征构造的方式，进一步考虑团簇结构上对能量的影响因素，可得到更加全面的预测模型特征变量。

（2）在粒子群算法中，可以加入关于原子键对之间的约束。

参考文献

1. 魏晓辉, et al., 机器学习加速CALYPSO结构预测的可行性. 吉林大学学报(工学版), 2021. 51(02): p. 667-676.

2. 周营成, 基于群智能和机器学习的新型纳米团簇结构预测研究. 2020, 北京化工大学.

3. 范天娥, 基于群智能算法的金属纳米团簇结构优化研究. 2018, 厦门大学.

4. 曾思琴, 基于智能优化算法的原子团簇结构研究. 2010, 浙江师范大学.

附录

1. 机器学习模型预测

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from scipy.spatial.distance import cdist
import networkx as nx
import warnings
warnings.filterwarnings("ignore")
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False


import os
import re
import math

from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR,LinearSVR
from sklearn.ensemble import
AdaBoostRegressor,ExtraTreesRegressor,RandomForestRegressor,GradientB
oostingRegressor
from xgboost import XGBRegressor


from sklearn.model_selection import
train_test_split,cross_val_score,cross_val_predict # 交叉验证所需的函
数
from sklearn.model_selection import
KFold,LeaveOneOut,LeavePOut,ShuffleSplit # 交叉验证所需的子集划分方法
```

```
from sklearn.model_selection import
StratifiedKFold,StratifiedShuffleSplit # 分层分割
from sklearn.model_selection import
GroupKFold,LeaveOneGroupOut,LeavePGroupsOut,GroupShuffleSplit # 分组
分割


from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn import preprocessing
from sklearn.metrics import
accuracy_score,mean_squared_error,mean_absolute_error
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures


# In[7]:


path = "../A2_data/A1_rawdata/Au20_OPT_1000/"
# path = "../A2_data/A1_rawdata/B45-_OPT_3751/"

# 读取文件名
def get_allfile(path):
    filelist = []
    get_dir = os.listdir(path)
    for i in get_dir:
        filelist.append(os.path.join(path,i))

    return filelist

# 测试
filelist = get_allfile(path)
# filelist


# In[8]:



# 读取团簇数据
def read_one_xyz(filename):
    xyz = []
```

```
    with open(filename, 'r') as f: # 读取每一行
        flines = f.readlines()
        for fl in range(len(flines)):
            if fl == 0: # 第一行为原子个数
                num = flines[fl]
            elif fl == 1: # 第二行为能量
                power =
np.array(re.findall(r"[-]?\d+\.?\d*",str(flines[fl]))[-
1]).astype(float)
            else: # 其余行为坐标点
                xyz.append(flines[fl].split())

    axis_xyz = []
    for i in xyz[1:]: # 划分坐标点为 x,y,z
        axis_xyz.append(i[1:4])
    axis_xyz_num = np.array(axis_xyz).astype(float)

    return axis_xyz_num,power

# 测试
xyz,power = read_one_xyz("../A2_data/A1_rawdata/Au20_OPT_1000/2.xyz")
# xyz,power = read_one_xyz("../A2_data/A1_rawdata/B45-
_OPT_3751/2.xyz")
power
# xyz


# In[9]:


# 生成衍生变量

def distmatrix(xyz_matrix):
    """计算原子距离矩阵"""
    dist=cdist(xyz_matrix,xyz_matrix,metric='euclidean')
    return dist
dist = distmatrix(xyz) # 测试


def dict_index(distmatrix):
    """
```

```
    距离矩阵相关指标：最大距离，最小距离，平均距离，平均距离，中位数距离
    """
    temp = []

    for i in range(distmatrix.shape[0]): # 遍历距离矩阵所有行

        for j in range(distmatrix.shape[1]): # 遍历距离矩阵所有列
            if j >= i: # 不考虑原子与自身的距离
                continue
            temp.append(distmatrix[i,j]) # 将每个距离添加到列表，便于计算统计指标
    mindist = min(temp)
    maxdist = max(temp)
    meandist = np.mean(temp)
    meddist = np.median(temp)
    sumdist = sum(temp)
    return mindist,maxdist,meandist,meddist,sumdist
mindist,maxdist,meandist,meddist,sumdist = dict_index(dist) # 测试


def close_matrix(dist,index):
    """
    根据距离矩阵得到邻接矩阵
    """
    matrix = np.where(dist>index,1,0)     # 根据距离与 index 的相对来判断是否原子之间有边
    return matrix


def netgraph_index(matrix):
    """
    网络图结构指标分析
    """
    G = nx.Graph(matrix)  # 根据邻接矩阵构建网络结构

    Gnum =  G.number_of_edges()    # 网络边的数量
    mean_cluster = nx.average_clustering(G) # 平均聚类系数
    netrans = nx.transitivity(G) # 网络传递性
    mean_degrcenter =
np.average(list(nx.degree_centrality(G).values())) # 平均度中心性
```

```
    mean_closcenter =
np.average(list(nx.closeness_centrality(G).values())) # 平均接近中心
性
    mean_betwcenter =
np.average(list(nx.betweenness_centrality(G).values())) # 平均中介中
心性

    return
Gnum,mean_cluster,netrans,mean_degrcenter,mean_closcenter,mean_betwce
nter


def f_c(r_ij,r_cut=6):
    """分子性质衍生特征的切断函数 f_c"""
    if r_cut > r_ij:  # r_ij:原子之间的距离(or 距离矩阵？)
        f = (np.cos(np.pi*r_ij/r_cut)+1)/2
    else:
        f = 0
    return f


def G_func(xyz,yita=1,r_s=3):
    """G1,G2 函数（改写为了所有原子的和、均值）"""
    g1_temp,g2_temp = [],[]
    for j in range(xyz.shape[0]):
        f_c_ls1 = []
        f_c_ls2 = []
        for k in range(xyz.shape[0]):
            if k == j:
                continue
            dist_vec = np.linalg.norm(np.asarray(xyz[j]) -
np.asarray(xyz[k]))
            exp_dist_vec = np.exp(-yita * (dist_vec - r_s)**2)
            f_c_ls1.append(f_c(dist_vec))
            f_c_ls2.append(exp_dist_vec * f_c(dist_vec))
        g1_temp.append(sum(f_c_ls1))  # 文献中给出的 G1,每个团簇有 n
个 G1
        g2_temp.append(sum(f_c_ls2))  # 文献中给出的 G2,每个团簇有 n
个 G2

    g1_sum = np.sum(g1_temp)  # 对每个团簇的 G1 求和
    g2_sum = np.sum(g2_temp)  # 对每个团簇的 G2 求和
```

```
    g1 = np.average(g1_temp)  # 对每个团簇的G1求平均
    g2 = np.average(g2_temp)  # 对每个团簇的G2求平均
    return g1_sum,g2_sum,g1,g2,g1_temp,g2_temp
g1_sum,g2_sum,g1,g2,g1_temp,g2_temp = G_func(xyz) # 测试


def get_coulombMatrix(mole,zi,eigpercent=1):
    """
    计算库伦矩阵特征值,每个团簇的库伦矩阵的特征值大小排序
    """
    cij = np.zeros((mole.shape[0],mole.shape[0]))  # 初始化库伦矩阵
    for i in range(mole.shape[0]): # 遍历所有原子
        for j in range(mole.shape[0]): # 计算库伦矩阵
            if i == j:
                cij[i][j] = 0.5 * zi ** 2.4
            else:
                cij[i][j] = zi * zi / (np.linalg.norm((mole[i] -
mole[j]),ord=1))

    eig,_ = np.linalg.eig((cij + cij.T)/2) # 计算特征值，特征向量(这
里为了尽可能消除特征值为复数的写法)
    eig_sorted = np.sort(eig)[::-1]
    eigvalue = eig_sorted[:int(eig.shape[0]* eigpercent)]  # 选取最大
的几个特征值（按照百分比计算，默认全部）
    return eigvalue
eigvalue = get_coulombMatrix(xyz,79,1) # 测试


# In[22]:


# 生成分析数据集(耗时较旧)
def output_dataframe(path,zi,eigpercent=1):
    """
    path:文件夹路径
    zi:原子电荷数
    eigpercent:库伦矩阵特征值个数比例
    """
    filelist = get_allfile(path)
    X = []   # 特征向量
    Y = []   #预测目标(power)
    for i in range(len(filelist)):
```

```
        sample = []
        file,power = read_one_xyz(filelist[i])  # 读取数据

         # 将库伦矩阵特征值添加到特征向量中
        eigvalue = get_coulombMatrix(file,zi,eigpercent)  # 计算库伦
矩阵
        if isinstance(eigvalue[0],complex):   # 删除复数特征值样本
            continue
        sample = [x for x in eigvalue]

        # 将原子间的距离指标添加到特征向量中
        dist = distmatrix(file)  # 距离矩阵
        mindist,maxdist,meandist,meddist,sumdist = dict_index(dist)
# 距离指标
        sample = sample + [mindist,maxdist,meandist,meddist,sumdist]

        # 将团簇网络结构指标添加到特征向量中
        closematrix = np.where(dist>meandist,1,0)   # 计算邻接矩阵

Gnum,mean_cluster,netrans,mean_degrcenter,mean_closcenter,mean_betwce
nter = netgraph_index(closematrix) # 网络结构指标
        sample = sample +
[Gnum,mean_cluster,netrans,mean_degrcenter,mean_closcenter,mean_betwc
enter]

        # 将 G 函数指标添加到特征向量中
        g1_sum,g2_sum,g1,g2,g1_temp,g2_temp = G_func(file)  # 计算
G1, G2
        sample = sample + [g1_sum,g2_sum,g1,g2] + g1_temp + g2_temp

        X.append(sample)

        # 将预测能量添加到目标变量中
        Y.append(power)
    return X,Y

X,Y = output_dataframe(path,79,1)


# In[24]:
```

```
df_X = pd.DataFrame(X).add_prefix("X")
df_Y = pd.DataFrame(Y,columns=["Y"])
data = pd.concat([df_X,df_Y],axis=1)

#
data.to_csv("../A2_data/A2_output/Au20_alldata.csv",index=False,heade
r=True)
#
data.to_csv("../A2_data/A2_output/B45_alldata.csv",index=False,header
=True)
df_X.shape
# df_Y.shape

# In[25]:

data.describe()

# In[32]:

# plt.hist(df_Y,bins=30)  # 查看目标变量分布情况
# B40 存在异常值： data["Y"] > -92000

plt.figure(figsize=(8,6))
plt.hist(df_Y,bins=15,)  # 查看目标变量分布情况
plt.xlabel = "团簇能量值"
plt.ylabel = "团簇频数"
plt.show()
# Au20 存在离群值，-1530.908363

# 删除离群点
data_drop = data[(data["Y"] < -1535)]
# data_drop = data[(data["Y"] < -92000)]
data_drop.shape



# In[33]:


# 数据标准化
zscore = preprocessing.StandardScaler()
data_scale = zscore.fit_transform(data_drop)
data_scale.shape
```

```
# In[34]:


# 主成分降维
pca = PCA(n_components=0.9)
pca.fit(data_scale[:,0:data_scale.shape[1]-1])
data_scale_pca = pca.transform(data_scale[:,0:data_scale.shape[1]-1])
data_scale_pca.shape


# In[35]:


# 划分数据集（测试发现：为标准化，进行降维之后效果较差）

# 下面是标准化，未进行主成分降维的数据
# X_data = data_scale[:,range(data_scale.shape[1]-1)]
# Y_data = data_scale[:,-1]

# 下面是标准化，进行主成分降维的数据
X_data = data_scale_pca[:,range(data_scale_pca.shape[1]-1)]
Y_data = data_scale[:,-1]

# 下面是不标准化，未进行主成分降维的数据
# X_data = data_drop.iloc[:,range(data_drop.shape[1]-1)]
# Y_data = data_drop.iloc[:,-1]

X_train,X_test,y_train,y_test =
train_test_split(X_data,Y_data,random_state=200,test_size=0.7)


# In[38]:


def cross_valid_allmodel(model,X,Y,kfold=5):
    """
    构建一个交叉验证方法
    返回交叉验证的评分
    """
    cv = ShuffleSplit(n_splits=20, test_size=.3, random_state=0)
```

```
    scores_CV = cross_val_score(model, X_data, Y_data, cv=cv)
    return scores_CV

models = {
    "Linear Regression":LinearRegression(),
    "Ridge Regression":Ridge(),
    "Linear SVM":LinearSVR(),
    "SVM":SVR(),
#     "Random Forest":RandomForestRegressor(), # 计算时间较长
    "XGBoost":XGBRegressor(),
    "AdaBoost":AdaBoostRegressor()
}

scores_dict = {}
for name,model in models.items():
    scores_CV = cross_valid_allmodel(model,X_data, Y_data,kfold=10)
    print(name,":",scores_CV)
    scores_dict[name] = scores_CV  # 保存交叉验证评分结果
df_scoresCV = pd.DataFrame(scores_dict)
df_scoresCV.to_csv("../A2_data/A2_output/df_scoresCV.csv",index=False
,header=True)


# In[47]:


score_plotdata = pd.DataFrame(scores_dict).melt()

# 可视化绘制箱线图
sns.set(style="whitegrid", color_codes=True)
sns.boxplot(x="variable", y="value",data=score_plotdata)
sns.swarmplot(x="variable", y="value", data=score_plotdata,
color=".25")
plt.xticks(rotation=90)
plt.xlabel("Machine Learning Models")
plt.ylabel("Score")
plt.show()

"""
从不同模型交叉验证的箱线图可以发现，黑箱算法（SVM/XGBoost/AdaBoost）
普遍比白箱算法（线性回归，岭回归）预测效果较好。
但白箱算法的预测评分的中位数也在 0.91 附近，在应用中完全可以接受。
```

故考虑模型可接受性，最终预测模型选择线性回归模型
"""

In[39]:

"""线性回归"""
lr =
LinearRegression(fit_intercept=False,normalize=True,n_jobs=10).fit(X_
train,y_train)
lr_coef = np.round(lr.coef_,5)
print(lr_coef)
print('Training set score:{:.10f}'.format(lr.score(X_train,y_train)))
print('Test set score:{:.10f}'.format(lr.score(X_test,y_test)))
print("Training set
MSE:{:.2f}".format(mean_squared_error(y_train,lr.predict(X_train))))
print("Test set
MSE:{:.2f}".format(mean_squared_error(y_test,lr.predict(X_test))))
lr_predict = lr.predict(X_data)
predict_data =
pd.concat([pd.DataFrame(X_data),pd.DataFrame(Y_data,columns=["Y"]),pd
.DataFrame(lr_predict,columns=["predict"])],axis=1)
predict_data.to_csv("../A2_data/A2_output/Au20_predictdata.csv",index
=False,header=True)

# 残差图
plt.plot((lr.predict(X_data)-Y_data)**2)
plt.show()
"由于数据中存在异常值，所以可以发现残差图中有一处明显凸起。但整体上而言，无异常现象。"

2. 改进粒子群算法

```
from pyswarm.pso import *
import numpy as np
from pyswarm import pso
import matplotlib.pyplot as plt
import math
import random
from bisect import bisect_left
import timeit
from machine_learning_predict import machine_learning,ridge

"""基础粒子群算法案例"""
# print('*'*65)
# print('Example minimization of 4th-order banana function (no
constraints)')
# def myfunc(x):
#     # 自适应函数
#     x1 = x[0]
#     x2 = x[1]
#     return x1**4 - 2*x2*x1**2 + x2**2 + x1**2 - 2*x1 + 5
#
# lb = [-3, -1]
# ub = [2, 6]
#
# xopt1, fopt1 = pso(myfunc, lb, ub)
#
# print('The optimum is at:')
# print('    {}'.format(xopt1))
# print('Optimal function value:')
# print('    myfunc: {}'.format(fopt1))

"""轮盘赌三种选择方式"""
class Choice_p():

    def __init__(self,index):
        self.index = index

    """
    Basic roulette wheel selection: O(N)
    """
    def basic(self,fitness):
        '''
```

```
    Input: a list of N fitness values (list or tuple)
    Output: selected index
    '''
    sumFits = sum(fitness)
    # generate a random number
    rndPoint = random.uniform(0, sumFits)
    # calculate the index: O(N)
    accumulator = 0.0
    for ind, val in enumerate(fitness):
        accumulator += val
        if accumulator >= rndPoint:
            self.index = ind
    return self.index
"""
Bisecting search roulette wheel selection: O(N + logN)
"""
def bisectSearch(self,fitness):
    '''
    Input: a list of N fitness values (list or tuple)
    Output: selected index
    '''
    sumFits = sum(fitness)
    # generate a random number
    rndPoint = random.uniform(0, sumFits)
    # calculate the accumulator: O(N)
    accumulator = []
    accsum = 0.0
    for fit in fitness:
        accsum += fit
        accumulator.append(accsum)
    self.index = bisect_left(accumulator, rndPoint)  # O(logN)
    return self.index

"""
Stochastic Acceptance: O(1) if given the N and maxFit before
"""
def stochasticAccept(self,fitness):
    '''
    Input: a list of N fitness values (list or tuple)
    Output: selected index
    '''
    # calculate N and max fitness value
```

```
        N = len(fitness)
        maxFit = max(fitness)
        # select: O(1)
        while True:
            # randomly select an individual with uniform probability
            ind = int(N * random.random())
            # with probability wi/wmax to accept the selection
            if random.random() <= fitness[ind] / maxFit:
                self.index =  ind
            return self.index

"""基础粒子群算法"""
def getweight():
    # 惯性权重
    weight_0 = 1
    e_t = gbestfitness/gbestfitness_last
    af_t = sizepop*gbestfitness/sum(pbestfitness)
    weight_t = weight_0 - 0.5*e_t + 0.1*af_t
    return weight_t

def getlearningrate():
    # 分别是粒子的个体和社会的学习因子，也称为加速常数
    lr = (0.49445,1.49445,0.1)
    return lr

def getmaxgen():
    # 最大迭代次数
    maxgen = 500
    return maxgen

def getsizepop():
    # 种群规模
    sizepop = 50
    return sizepop

def getrangepop():
    # 粒子的位置的范围限制,x、y方向的限制相同
    rangepop = (-2*math.pi , 2*math.pi)
    return rangepop

def getrangespeed():
    # 粒子的速度范围限制
```

```
    rangespeed = (-0.5,0.5)
    return rangespeed

# 机器学习训练的方法
method = ridge()
def func(x):
    # l = [3.73270000*10**(-1), -3.24580000*10**(-1), -
1.78660000*10**(-1), - 3.00000000*10**(-3),
    #  1.18220000*10**(-1),  1.08890000*10**(-1),  1.03040000e-
01*10**(-1),  5.91900000*10**(-2),
    #  9.70900000*10**(-2), -1.95900000*10**(-2),  2.91000000*10**(-
2),  1.11930000*10**(-1),
    #  6.62500000*10**(-2), -2.17070000*10**(-1), - 1.22470000*10**(-
1), - 6.65785720*10**12,
    #  3.08590000*10**(-1),  6.65785720*10**12,  4.16224446*10**12,
2.05190000*10**(-1),
    #  -4.19970000*10**(-1), -4.16224446*10**12, - 2.86280000*10**(-
1),  5.30000000*10**(-4),
    #  -8.45594120*10**12,  4.05885115e+11, - 5.35638659*10**12,
2.68703067*10**12,
    #  1.88686196*10**12,  1.76396883*10**12,  1.76790267*10**12,
1.71237372*10**12,
    #  1.71579553*10**12,  1.6634948*10**12,  1.67104727*10**12,
1.37758059*10**12,
    #  1.38786907*10**12, 1.39354072*10**12,  1.41509287*10**12,
1.29690053*10**12,
    #  1.31075496*10**12,  1.25871102*10**12,  1.24285077*10**12,
1.19969065*10**12,
    #  1.19314020*10**12,  1.16821616*10**12,  1.19796968*10**12, -
4.82269577*10**12,
    #  -4.30005919*10**11, -4.33435059*10**11, - 4.15944547*10**11, -
4.18099957*10**11,
    #  -4.03970062*10**11, -4.04740363*10**11, - 3.30891838*10**11, -
3.34063652*10**11,
    #  -3.31951179*10**11, -3.36062300*10**11, - 3.04030130*10**11, -
3.07748448*10**11,
    #  -2.93184773*10**11, -2.89707401*10**11, - 2.78641486*10**11, -
2.76861889*10**11,
    #  -2.74400608*10**11, - 2.80574051*10**12]
    # y = sum([a*b for a,b in zip(l,list(x))])
    y = machine_learning(x,method)
    return y
```

```
# 团簇原子个数 n，自变量维度 D
n = 20
# D = 66
D = n * 3

def initpopvfit(sizepop,D):
    # 初始化种群
    pop = np.zeros((sizepop,D))
    v = np.zeros((sizepop,D))
    fitness = np.zeros(sizepop)

    for i in range(sizepop):
        axis_of_symmetry = random.choice([0, 1, 2])   # 对称轴
        half_pop = [rangepop[0]+(rangepop[1]-rangepop[0])*j/D for j
in range(int(D/2))]
        half_pop_ = [0]*(int(D/2))
        if axis_of_symmetry == 0:
            half_pop_ = np.multiply(np.array(half_pop), np.array([-1,
1, 1]*(int(D/6))))
            half_pop_ = list(half_pop_)
        if axis_of_symmetry == 1:
            half_pop_ = np.multiply(np.array(half_pop), np.array([-1,
1, 1]*(int(D/6))))
            half_pop_ = list(half_pop_)
        if axis_of_symmetry == 2:
            half_pop_ = np.multiply(np.array(half_pop), np.array([-1,
1, 1]*(int(D/6))))
            half_pop_ = list(half_pop_)
        pop[i] = half_pop + half_pop_
        # pop[i] = [rangepop[0]+(rangepop[1]-rangepop[0])*j/D for j
in range(D)]
        v[i] = [rangepop[0]+(rangepop[1]-rangepop[0])*j/D for j in
range(D)]
        fitness[i] = func(pop[i])

    return pop,v,fitness

def getinitbest(fitness,pop):
    # 群体最优的粒子位置及其适应度值
    gbestpop,gbestfitness =
pop[fitness.argmin()].copy(),fitness.min()
```

```
    #个体最优的粒子位置及其适应度值,使用 copy()使得对 pop 的改变不影响
pbestpop, pbestfitness 类似
    pbestpop,pbestfitness = pop.copy(),fitness.copy()
    return gbestpop,gbestfitness,pbestpop,pbestfitness

def DE(n,K,CR):
    # 随机选择 n 个个体交叉
    rand_num = set()
    while len(rand_num) <= n:
        rand_num.add(np.random.randint(0,sizepop-1))
    # 得到变异个体
    V = pop[list(rand_num)[0]] + K*(pop[list(rand_num)[1]]-
pop[list(rand_num)[2]])
    # 交叉操作
    pop_com = pop.copy()
    for i in range(pop.shape[0]):
        u = V.copy()
        for j in range(pop.shape[1]):
            if np.random.rand() > CR:
                u[j] = pop[i][j]
        pop_com[i] = u
    # 选择操作
    pop_new = pop.copy()
    for i in range(pop.shape[0]):
        if func(pop[i]) > func(pop_com[i]):
            pop_new[i] = pop_com[i]
    return pop_new

lr = getlearningrate()
maxgen = getmaxgen()
sizepop = getsizepop()
rangepop = getrangepop()
rangespeed = getrangespeed()

pop,v,fitness = initpopvfit(sizepop,D)
gbestpop,gbestfitness,pbestpop,pbestfitness =
getinitbest(fitness,pop)
gbestfitness_last = gbestfitness.copy()

## 算法参数
stepsize = 1 # 莱维飞行
kesi = 1 # 算法精度
```

```
## 粒子群主程序
result = np.zeros(maxgen)
for i in range(maxgen):

        # 更新惯性权重
        w = getweight()

        # 速度更新
        for j in range(sizepop):
            fitness = list(map(func, pop))
            choice = Choice_p(2)
            pop_index = choice.basic(fitness)
            # 加入莱维飞行
            if np.random.rand() < kesi:
                v[j] = w * v[j] + lr[0] * np.random.rand() *
(pbestpop[j] - pop[j]) + lr[1] * np.random.rand() * (
                            gbestpop - pop[j]) \
                       + lr[2] * np.random.rand() * (pop[pop_index] -
pop[j])
            else:
                v[j] = w * v[j] + lr[0] * np.random.rand() *
(pbestpop[j] - pop[j]) + lr[1] * np.random.rand() * (
                            gbestpop - pop[j]) \
                       + lr[2] * np.random.rand() * (pop[pop_index] -
pop[j]) * stepsize
        v[v<rangespeed[0]] = rangespeed[0]
        v[v>rangespeed[1]] = rangespeed[1]

        #粒子位置更新
        for j in range(sizepop):
            pop[j] += v[j]
        pop[pop < rangepop[0]] = rangepop[0]
        pop[pop > rangepop[1]] = rangepop[1]

        #粒子种群差分变异
        pop = DE(3,1,0.5)
        pop[pop < rangepop[0]] = rangepop[0]
        pop[pop > rangepop[1]] = rangepop[1]

        #适应度更新
        for j in range(sizepop):
```

```
            fitness[j] = func(pop[j])

        for j in range(sizepop):
            if fitness[j] < pbestfitness[j]:
                pbestfitness[j] = fitness[j]
                pbestpop[j] = pop[j].copy()

        if pbestfitness.min() < gbestfitness :
            gbestfitness_last = gbestfitness.copy()
            gbestfitness = pbestfitness.min()
            gbestpop = pop[pbestfitness.argmin()].copy()

        result[i] = gbestfitness

print(gbestpop)
plt.plot(result)
plt.show()
```

3. 预测结构坐标

```
from pyswarm import pso
import numpy as np
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt
import math
import pandas as pd
G1 = 6
r_cut = 0.6
def f_c(r_ij):
    """分子性质衍生特征的切断函数 f_c"""
    if r_cut > r_ij:
        f = (np.cos(np.pi*r_ij/r_cut)+1)/2
    else:
        f = 0
    return f

def distmatrix(xyz):
    """计算单个团簇样本的距离矩阵"""
    dist=cdist(xyz,xyz,metric='euclidean')
    return dist

def G1_fuc(x):
    G1_ = 0
    sample = []
    x = list(x)
    for i in range(20):
        sample.append([x[0 + 3 * i], x[1 + 3 * i], x[2 + 3 * i]])
    dist = distmatrix(np.array(sample))
    for i in range(len(sample)):
        for j in range(len(sample)):
            G1_ += f_c(dist[i,j])
    return G1_

def getweight():
    # 惯性权重
    weight_0 = 1
    e_t = gbestfitness/gbestfitness_last
    af_t = sizepop*gbestfitness/sum(pbestfitness)
    weight_t = weight_0 - 0.5*e_t + 0.1*af_t
    return weight_t
```

```
def getlearningrate():
    # 分别是粒子的个体和社会的学习因子，也称为加速常数
    lr = (0.49445,1.49445,0.1)
    return lr

def getmaxgen():
    # 最大迭代次数
    maxgen = 500
    return maxgen

def getsizepop():
    # 种群规模
    sizepop = 50
    return sizepop

def getrangepop():
    # 粒子的位置的范围限制,x、y 方向的限制相同
    rangepop = (-6 , 6)
    return rangepop

def getrangespeed():
    # 粒子的速度范围限制
    rangespeed = (-0.5,0.5)
    return rangespeed

def func(x):
    G1_ = G1_fuc(x)
    y = G1_ - G1
    return y

# 团簇原子个数 n，自变量维度 D
n = 20
D = n * 3

def initpopvfit(sizepop,D):
    # 初始化种群
    pop = np.zeros((sizepop,D))
    v = np.zeros((sizepop,D))
    fitness = np.zeros(sizepop)

    for i in range(sizepop):
        pop[i] = [rangepop[0]+(rangepop[1]-rangepop[0])*j/D for j in
```

```
range(D)]
        v[i] = [rangepop[0]+(rangepop[1]-rangepop[0])*j/D for j in
range(D)]
        fitness[i] = func(pop[i])
    return pop,v,fitness

def getinitbest(fitness,pop):
    # 群体最优的粒子位置及其适应度值
    gbestpop,gbestfitness =
pop[fitness.argmin()].copy(),fitness.min()
    #个体最优的粒子位置及其适应度值,使用 copy()使得对 pop 的改变不影响
pbestpop, pbestfitness 类似
    pbestpop,pbestfitness = pop.copy(),fitness.copy()
    return gbestpop,gbestfitness,pbestpop,pbestfitness

def DE(n,K,CR):
    # 随机选择 n 个个体交叉
    rand_num = set()
    while len(rand_num) <= n:
        rand_num.add(np.random.randint(0,sizepop-1))
    # 得到变异个体
    V = pop[list(rand_num)[0]] + K*(pop[list(rand_num)[1]]-
pop[list(rand_num)[2]])
    # 交叉操作
    pop_com = pop.copy()
    for i in range(pop.shape[0]):
        u = V.copy()
        for j in range(pop.shape[1]):
            if np.random.rand() > CR:
                u[j] = pop[i][j]
        pop_com[i] = u
    # 选择操作
    pop_new = pop.copy()
    for i in range(pop.shape[0]):
        if func(pop[i]) > func(pop_com[i]):
            pop_new[i] = pop_com[i]
    return pop_new


lr = getlearningrate()
maxgen = getmaxgen()
sizepop = getsizepop()
```

```
rangepop = getrangepop()
rangespeed = getrangespeed()

pop,v,fitness = initpopvfit(sizepop,D)
gbestpop,gbestfitness,pbestpop,pbestfitness =
getinitbest(fitness,pop)
gbestfitness_last = gbestfitness.copy()

## 算法参数
stepsize = 1 # 莱维飞行
kesi = 1 # 算法精度

## 粒子群主程序
result = np.zeros(maxgen)
for i in range(maxgen):

        # 更新惯性权重
        w = getweight()

        # 速度更新
        for j in range(sizepop):
            v[j] = w * v[j] + lr[0] * np.random.rand() * (pbestpop[j]
- pop[j]) + \
                   lr[1] * np.random.rand() * (gbestpop - pop[j])

        v[v<rangespeed[0]] = rangespeed[0]
        v[v>rangespeed[1]] = rangespeed[1]

        #粒子位置更新
        for j in range(sizepop):
            pop[j] += v[j]
        pop[pop < rangepop[0]] = rangepop[0]
        pop[pop > rangepop[1]] = rangepop[1]

        #粒子种群差分变异
        pop = DE(3,1,0.5)
        pop[pop < rangepop[0]] = rangepop[0]
        pop[pop > rangepop[1]] = rangepop[1]

        #适应度更新
        for j in range(sizepop):
            fitness[j] = func(pop[j])
```

```
        for j in range(sizepop):
            if fitness[j] < pbestfitness[j]:
                pbestfitness[j] = fitness[j]
                pbestpop[j] = pop[j].copy()

        if pbestfitness.min() < gbestfitness :
            gbestfitness_last = gbestfitness.copy()
            gbestfitness = pbestfitness.min()
            gbestpop = pop[pbestfitness.argmin()].copy()

        result[i] = gbestfitness

print(gbestpop)
print(gbestfitness)
xyz = []
for i in range(20):
   xyz.append([gbestpop[0 + 3 * i], gbestpop[1 + 3 * i], gbestpop[2 +
3 * i]])
print(xyz)
plt.plot(result)
plt.show()

df = pd.DataFrame(xyz)
df.to_excel(r'C:\Users\manin\Desktop\MathorCup 高校数学建模挑战
赛.xlsx')
```