2024 年第十四届武汉理工大学研究生数学建模竞赛

承　诺　书

我们仔细阅读了2024年第十四届武汉理工大学研究生数学建模竞赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与本队以外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其它公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们愿意承担由此引起的一切后果。

参赛题号（从 A/B 中选择一项填写）：C

参赛队号：39

参赛队员：　队员 1 姓名：毕云飞

　　队员 2 姓名：刘鸿振

　　队员 3 姓名：朱思胥

联系方式：　Email：2277241439@qq.com　联系电话：15725332307

日期：　2024 年 6 月 11 日

2024 年武汉理工大学第 14 届研究生数学建模竞赛

题 目： 基于时间序列和异常检测的设备维护模型

摘 要：

如何尽可能的延长设备的使用寿命是一个备受关注的问题，除了改进设备制造工艺之外，合理的安排设备维护同样可以有效延长设备的使用寿命。因此如何安排设备的维护周期成为了一个很重要的问题，本文基于时间序列分析和异常检测技术建立一系列模型对如何合理安排设备维护周期延长设备使用寿命进行求解。

针对问题一，我们将设备的性能曲线分解为自然状态（无异常-无维护）性能曲线，异常性能曲线以及维护性能曲线，利用题目给出的第一联数据分别拟合。我们认为设备异常是指设备的性能变化率在一定时间内持续大于某一阈值，设备的维护是将设备的重置回一定时间以前。我们基于自然状态，异常以及维护性能曲线和异常检测规则构建了设备的性能仿真模型，利用 python 的 curve_fit 函数进行数据拟合，将模型代入第一联的实际监测数据，模型可以精准的识别出设备的异常时刻并进行维护。

针对问题二，我们基于问题一拟合的自然状态，异常以及维护三种性能曲线进行组合。探讨有异常-有维护、有异常-无维护、无异常-无维护和无异常-有维护四类模型，通过仿真发现，有异常-无维护的设备使用寿命最短，仅 1863 小时；有异常-有维护的设备的使用寿命是最长的，使用寿命达到了 2884 小时。

针对问题三，可以看作是对设备定期维护和维护间隔的优化问题，我们对基于问题一建立设备性能仿真模型建立对设备维护次数和维护时间点的优化模型，以获得最长的设备使用寿命。通过粒子群算法对优化模型求解，求解结果，通过优化新方案的设备使用寿命达到了 2907 小时。

针对问题四，关于异常识别，我们基于 LSTM 模型训练了对异常检测的分类模型；关于设备运行周期分析，我们基于问题一的模型，构建了新的自然状态，异常状态，老化状态，维护状态的性能模拟模型并分别模拟验证了我们的猜想；关于优化设备使用寿命，我们基于问题一的维护规则，训练了新数据集的模型，实验结果表示优化方案可以延长 300 小时的使用寿命。

针对问题五，本文选择基于随机森林的递归特征消失模型进行特征选择，最终选择了归一化流量，归一化温度，设备前端可能的影响因素 2。

关键词：时间序列分析；异常检测；LSTM。

目录

1. 问题重述

1.1 引言

在设备的日常运行过程中，由于长时间的工作和各种外界因素的影响，不可避免地会出现一定程度的损耗和异常情况。这些损耗和异常可能会影响设备的正常运行，甚至导致设备的提前报废。然而，通过定期对设备进行维护和保养，可以有效地减缓设备的损耗速度，预防和解决设备的异常情况，从而延长设备的使用寿命。设备的维护并不是随意进行的，而是需要根据设备的实际运行情况来制定合理的维护计划。这就需要利用设备运行监测数据来进行分析和判断。通过对设备运行数据的分析，可以了解设备的运行状态，发现设备的潜在的故障隐患，从而确定最佳的设备维护周期。总的来说，设备维护是一项非常重要的工作，它直接关系到设备的使用寿命和企业的运营成本。因此，企业应该重视设备的维护工作，合理规划设备维护周期，尽可能提高设备的使用寿命，从而提高企业的经济效益。

1.2 问题的提出

1.2.1 问题一

对于第一联数据，建立数学模型，识别什么时候设备运行异常 (或设备的性能发生了转折)，应该停机维护，给出模型的具体方法和分析结果。

1.2.2 问题二

建立数学模型，关于第一联数据的运行周期进行分析。

1.2.3 问题三

关于第一联数据，给出设备维护周期优化方案，即在什么时候维护，能提高设备的使用寿命，给出详细模型分析及结果分析。

1.2.4 问题四

运用第二联多特征数据，建立数学模型，识别何时设备异常需要进行维护，对设备运行周期进行分析，给出设备维护周期的优化方案，以延长设备的使用寿命。

1.2.5 问题五

对第 4 问中模型的特征进行分析，哪些检测指标对模型的影响比较大，通过筛选特征将模型优化，完成第 4 问中的任务。

2. 模型的假设

- 设备维护相当于将设备状态重置到一定时间以前，但是老化状态不会被重置
- 设备维护存在两种类型：定期的设备维护和设备异常的设备维护；
- 设备的维护次数与设备性能变化率没有关系；
- 设备出现异常会使性能变化率增加。

3. 问题分析

3.1 问题一分析

题目要求建立模型识别设备运行异常，应该停机维护。属于时间序列分析下的监测问题，题目提供的数据是设备在运行期间进行了数次维护的性能监测曲线。为了获得设备在自然条件（不进行维护且不出现异常）下的性能变化曲线，本文首先要剔除维护对设备性能的影响。然后，为了模拟维护的效果，本文拟合了维护对设备性能提升的函数。再然后，本文认为设备需要维护是在设备的性能变化率达到一定阈值且距离上次维护间隔一定时间的时候，所以本文计算了每次维护时的性能变化率和时间间隔。本文认为需要根据设备的性能变化率和维护时间间隔将设备分为待维护和非待维护两大类 [1]。最后，本文利用建立的模型对数据进行仿真，结果与实际数据进行对比。

3.2 问题二分析

题目要求建立模型，对设备的运行周期进行分析。本文借助问题一建立的模型分析了当设备出现异常时，有无设备维护对设备运行周期的影响。

3.3 问题三分析

题目要求给出可以延长设备使用寿命的维护周期优化方案，本质上是一个优化问题，本文在问题一建立的性能，维护模型的基础上，通过对维护时间的选择和维修次数进行最优化，从而确定设备维护的最优化方案。

3.4 问题四分析

题目要求识别设备在何时出现异常并需要进行维护，相较于问题一，该问的数据特征较多，本文基于长短期记忆（LSTM）神经网络训练一个分类模型，以区分设备的正常与异常状态。进一步地，为了对设备的运行周期进行了深入分析。该分析专注于在无任何人工干预的情况下，对设备的运行过程进行建模。关于设计设备维护周期的优化方案，关键在于如何界定维护的最佳时机。根据问题一的分析，维护后设备性能的显著下降为本文提供了一个明确的判断标准。

3.5 问题五分析

题目要求分析哪些检测指标对模型的影响较大，属于一个特征选择问题，本文选择基于随机森林的递归特征消失模型来寻找重要程度更高的特征。

4. 问题一的模型建立与求解

4.1 数据预处理

在构建模型之前，需要对时间序列数据进行一定的预处理方便后续的训练与计算。本文依次对数据进行了数据平滑，异常值处理，缺失值填充的处理。

由于时间序列数据在记录时因为各种因素容易导致记录的数据出现小幅度的波动，这种波动对于数据训练存在一定影响，所以要进行数据平滑，本文选择滑动窗口平均作为数据平滑的方案，计算公式如下：

$$x_t = \frac{1}{k} \sum_{j=t-k+1}^{t} x_j \tag{1}$$

其中，x_t 表示 t 时刻的性能指标，k 表示滑动窗口的大小，计算 t 时刻前 k 小时内的平均值作为新的 t 时刻的性能指标。

关于异常值的处理，本文选择 z-score 作为评判标准，在上述进行数据平滑时，当窗口内的数据值大于 z-score 时，认为是数据异常点，对于异常点，本文使用当前滑动窗口的数据平均值作为替换，z-score 的计算公式如下所示：

$$\mathrm{z_t} = \frac{\mathrm{x_t} - \overline{\mathrm{x_t}}}{\mathrm{s_t}} \tag{2}$$

其中 z_t 是 x_t 的 z-score，$\overline{x_t}$ 是计算 x_t 时的滑动窗口的数据平均值，s_t 是计算 x_t 时的滑动窗口的数据标准差。

关于缺失值的填充，本文选择前填后填，利用数据前一小时和后一小时的有效数据进行填充。

4.2 模型建立

为了确定维护设备的评判规则以及后续对规则进行验证，本文需要分别拟合出每次维护后性能提升的效果，设备在自然条件下的性能变化曲线以及需要维护的标准。

4.2.1 设备维护效果

本文认为，设备每次维护相当于将设备状态重置到一定时间以前。实际数据中在不同的运行时间时进行维修对设备的重置时间是不同的，为了简化模型本文使用一个定值作为任意时刻对设备维修时对设备的重置时间，这是一个优化

问题，所以本文选择粒子群算法（PSO）计算维护设备会将设备重置时间。PSO的适应度函数为：

$$g\left(\Delta t\right)=\sum_{i=2}^{11}\left(f\left(t_i\right)-f\left(t_i-\Delta t\right)-\theta_i\right)^2 \tag{3}$$

其中，$f(t)$ 表示设备在自然状态下 t 时刻的性能指标，$f\left(t_i\right)$ 表示第 i 段运行时间的起点的设备性能，即设备经过维护后第一时间的性能,Δt 表示对设备维修时对设备的重置时间，θ_i 表示第 i-1 段和第 i 段运行时间之间设备经过维护的性能提升值。优化目标为：

$$\min_{\Delta t\in[0,1000]} g\left(\Delta t\right) \tag{4}$$

4.2.2 设备在自然状态下的性能曲线

为了拟合设备在自然状态下的性能曲线，本文需要把数据中由于设备维护带来的影响给剔除，利用上节计算出的 Δt，把每次维护后的数据都减去设备维护带来性能提升，计算公式为：

$$x_t=x_t+\left(x_i-x_{i-\Delta t}\right)\quad i=2,3...,11. \tag{5}$$

关于性能曲线的拟合，通过对数据的观察本文选择三次多项式进行拟合，即：

$$\hat{x}_t=f\left(t\right)=a+bt+ct^2+dt^3 \tag{6}$$

$\hat{x}_t$ 为性能曲线预测 t 时刻时的性能指标。

每进行一次维护，设备的性能函数就会进行一次分段，具体公式为：

$$f_h\left(t\right)=\begin{cases}a+bt+ct^2+dt^3 & t\in[0,k_1]\\ a+bt+ct^2+dt^3-\left(f\left(k_1\right)-f\left(k_1-\Delta t\right)\right) & t\in(k_1,k_2]\\ \vdots & \\ a+bt+ct^2+dt^3-\sum_{i=1}^{n}f\left(k_i\right)-f\left(k_i-\Delta t\right) & t\in(k_n,+\infty]\end{cases} \tag{7}$$

其中，$f_h\left(t\right)$ 表示存在维护情况下设备性能的函数，k_i 表示第 i 次维护时的时间，且 $k_i-k_{i-1}\leq L,\ i=2,3,...,n$。

4.2.3 维护的标准

为了排除传感器在某一刻出现误报的情况，本文认为设备的性能变化率出现明显转折一段时间后需要进行维护，同时，因为两次维护之间存在一定时间间隔 L，所以可能出现设备性能变化率以及出现了明显转折一段时间但是没有立即维护的情况。

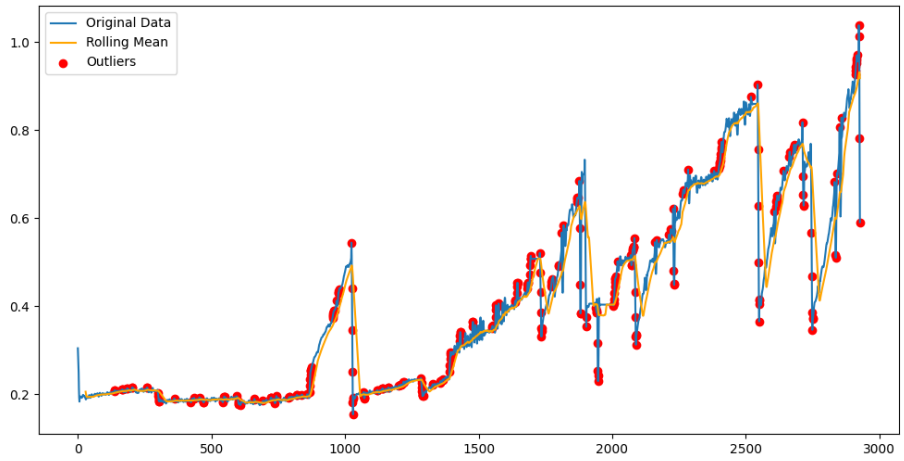


图 1　数据预处理后时间序列示意图

本文选择 30 小时作为一个时间滑动窗口，认为如果在 30 小时内性能的变化率持续大于阈值就认为属于设备异常，需要进行维护。在题目给出的数据中，本文认为第 4，7，9，10，11 段均为出现设备异常情况。

本文选择每次设备的性能变化率出现明显转折时与自然状态下设备性能变化率差值的平均值作为决定维护的阈值，计算公式如下：

$$\mu = \frac{1}{N}\sum_{j=1}^{N}(\mu_j - f'(t_j)) \tag{8}$$

其中，μ 是决定维护的性能变化率阈值，μ_j 表示第 j 次设备的性能变化率出现明显转折时的性能变化率，$f'(t_j)$ 表示自然状态下此时的设备性能变化率。

除了通过设备的性能变化率决定是否要维护外，本文认为还需要考虑两个指标，一个是每隔一定时间 T 需要对设备进行一次定期维护，另一个是由于维护成本等原因本文认为两次维护之间存在一个最小时间间隔 L。当设备的性能变化率超过阈值时，还要考虑此时离上一次维护的时间间隔是否大于 L。关于这两个指标的求解，

4.3 模型求解

利用第一联的数据对上述模型进行求解。经过预处理后设备性能指标时间序列图如图1所示

关于两次维护的最短时间间隔，通过计算可知 $L = 200h$，且定期维护的时间间隔为 300 小时。关于 Δt，通过粒子群算法，本文得出最优的 Δt 为 93h，关于设备在自然状态下的性能拟合曲线如下图2所示：

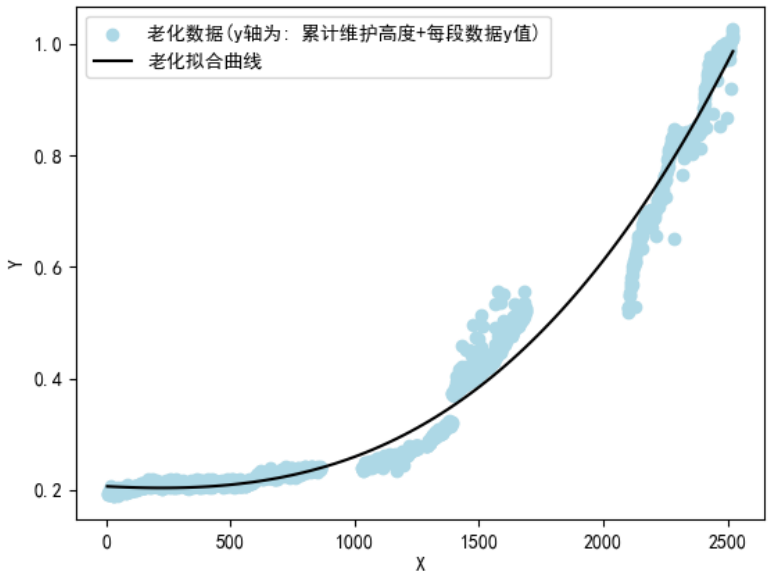


图 2　性能模拟曲线

具体表达式为：

$$f(t)=0.2054612+7.556058\times10^{-6}\times t-4.081754\times10^{-8}\times t^{2}+7.932633\times10^{-11}\times t^{3}, \tag{9}$$

接下来对实际数据进行仿真，测试设备在出现异常情况后模型能否识别并进行维护。关于设备的异常仿真，本文模拟了题目提供的设备异常时间段的性能函数，设备异常的模拟函数如下

$$f_g(t)=f(t)+\mu_{mean}(t-t_0),\ t\geq t_0, \tag{10}$$

$$\mu_{mean}=\frac{1}{n-30}\sum_{i=30}^{n-30}(f'(t_i)-\mu_{t_i}). \tag{11}$$

其中，$f_g(t)$ 表示存在异常时的设备性能函数，μ_{mean} 是计算在发生异常时间段内，设备理论上在自然状态下该时刻 t_i 的变化率和实际变化率 μ_{t_i} 差值的平均值，然后 t_0 表示异常开始的时刻。

将异常情况加入的设备性能模拟监测图如图3所示

从图中本文可以看出，当异常发生时，模型可以快速识别出设备异常并进行维护。

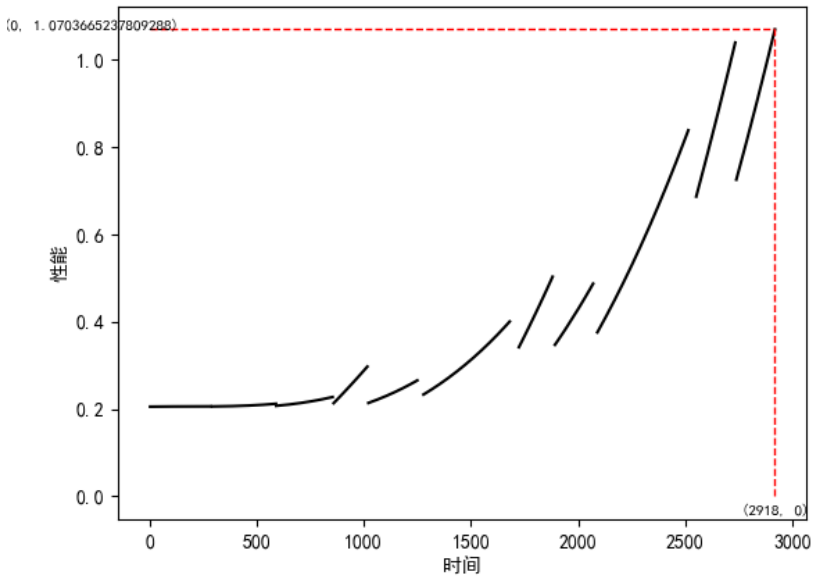


图 3　设备性能在存在异常和维护情况下的模拟时间序列

5. 问题二的模型建立与求解

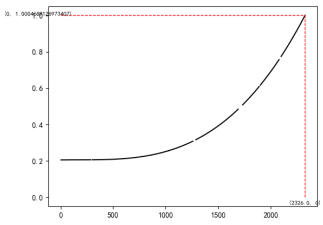
5.1 模型建立

基于问题一的模型，本文考虑两类情况的叠加，即有无设备异常和有无维护设备两类情况，依次为有异常-有维护、有异常-无维护、无异常-无维护和无异常-有维护四类模型。其中有异常-无维护的模型即公式（10），无异常-有维护的模型即公式（7），无异常-无维护的模型即公式（6），有异常-有维护的模型如下所示：

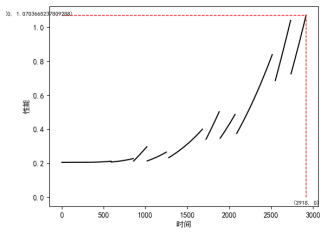
$$f_{gh}(t) = f_h(t) + f_g(t) - f(t) \tag{12}$$

5.2 模型求解

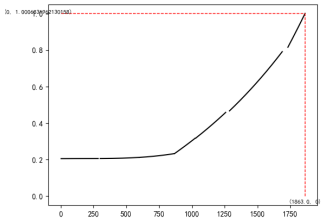
对于四类模型，四种模拟性能示意图如下图12所示:



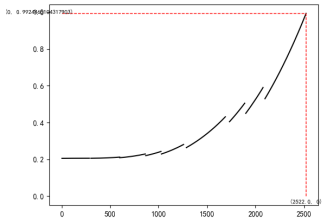
(a) 无异常-无维护



(b) 有异常-有维护



(c) 有异常-无维护



(d) 无异常-有维护

图 4　不同情况下的设备性能曲线

其中，无异常-无维护情况的设备运行了 2326 小时达到了性能的使用寿命；有异常-有维护的情况设备总共运行了 2884 小时; 有异常-无维护情况下的设备运行了 1863 小时达到了设备的使用寿命；无异常-有维护的设备总共运行了 2522 小时达到了设备的使用寿命。从性能曲线图可以看出，无异常-有维护的情况下比无异常-无维护明显延长了设备的使用寿命。但是出现了一个违反自觉的情况，有异常-有维护情况的使用寿命是最长的，本文认为是由于异常导致设备的维护相比与无异常-有维护情况更加频繁，所以适当的增加设备维护次数可以有效的延长设备的使用寿命。

6. 问题三的模型建立与求解

6.1 模型建立

由问题二的结果可知，通过调整维修的时间点可以有效延长设备的使用寿命。因此，建立对维修时间点和维修次数的最优化模型可以找到理论上延长设备使用寿命的最佳维护时间点和维护次数。

优化目标函数为：

$$\max\ Len\left(f_{gh}\right)=t_{\max}-n\times 200,\ \ f_{gh}\left(t_{\max}\right)=1\ and\ f_{gh}\left(t\right)<1\ t\in[0,t_{\max}) \quad (13)$$

约束条件

$$\begin{cases} T \in [T_l, T_u] \\ L \in [L_l, L_u] \end{cases} \tag{14}$$

其中，$Len\left(f_{gh}\right)$ 是给定 T, L 下设备在有维护-有异常情况下的使用寿命，n 表示插入时间点中违反约束的时间点个数。L_l, L_u, T_l, T_u 分别是维修次数和维修时间点所能选择的上下界。

本文选择使用粒子群算法（PSO）对优化模型求解，粒子群算法中，每个解决方案被视为搜索空间中的一个“粒子”，每个粒子代表了问题的潜在解。粒子在搜索空间中飞行，通过跟踪两个“极值”来更新自己的位置和速度：个体极值（pbest）和全局极值（gbest）。个体极值是粒子自身所找到的最优解，而全局极值是整个粒子群中所有粒子所找到的最优解。

这里给出粒子群算法的基本计算公式，第 i 个粒子在第 t 次迭代的位置 $x_i(t)$ 和速度 $v_i(t)$ 可以通过以下公式更新：

$$v_i(t+1) = w \cdot v_i(t) + c_1 \cdot r_1 \cdot (pbest_i - x_i(t)) + c_2 \cdot r_2 \cdot (gbest - x_i(t)) \tag{15}$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

其中：w 是惯性权重，用于平衡探索和开发。c_1 和 c_2 是学习因子，用于调整粒子向个体极值和全局极值的移动步长。当 $c_1 = 0$ 时，则粒子没有了认知能力，变为只有社会的模型 (social-only)，称为全局 PSO 算法。粒子有扩展搜索空间的能力，具有较快的收敛速度，但由于缺少局部搜索，对于复杂问题比标准 PSO 更易陷入局部最优。当 $c_2 = 0$ 时，则粒子之间没有社会信息，模型变为只有认知 (cognition-only) 模型，称为局部 PSO 算法。由于个体之间没有信息的交流，整个群体相当于多个粒子进行盲目的随机搜索，收敛速度慢，因而得到最优解的可能性小。r_1 和 r_2 是 [0, 1] 区间内的随机数，为算法引入随机性。$pbest_i$ 是第 i 个粒子迄今为止找到的最优位置。$gbest$ 是整个粒子群迄今为止找到的最优位置。图5给出了基本粒子群算法的流程示意图，

6.2 模型求解

本文给定约束 $L_l = 0, L_u = 4000, T_l = 10, T_u = 15$，模型训练参数 $'c1' : 0.5, 'c2' : 0.3, 'w' : 0.9$，最大迭代数为 100。经过迭代 150 轮后，模型的求解迭代效果如下图14模型求解结果为: 26, 230, 785, 908, 994, 1077, 1202, 1702, 2381, 2388 优化后的最佳设备使用寿命为 2907 小时。

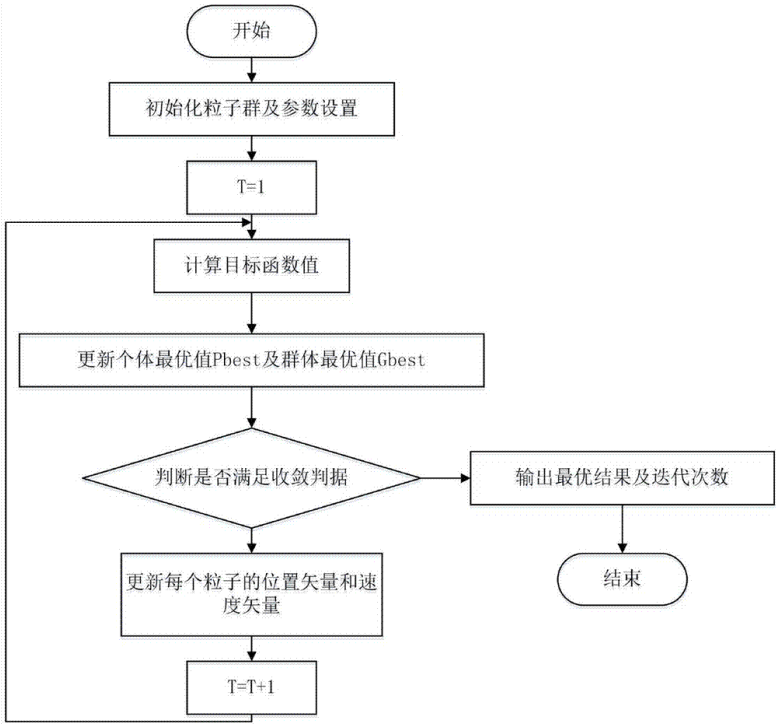


图 5　基本粒子群算法流程图

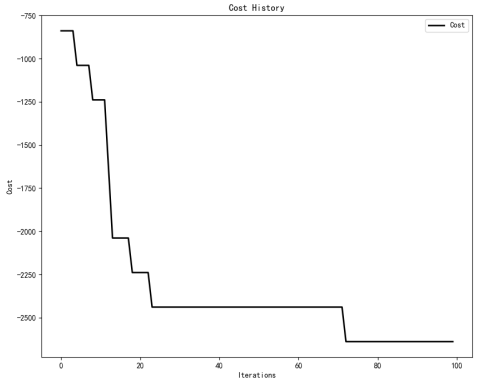


图 6　PSO 适应度迭代示意图

7. 问题四的模型建立与求解

7.1 模型建立

7.1.1 设备异常维护

从图像中可以看出，sheet2 中设备老化和异常对设备性能带来的影响不是单纯的上升，而是体现在设备性能的波动上（可能是由于出现异常的种类不同），基于问题一的模型，正常情况下，模型性能的波动性会随着设备的运行进行分段或连续的增长。异常情况则是在短时间内出现了线性的，不正常的增长，维修则

可以将不正常的增长还原到当前的基准线上。

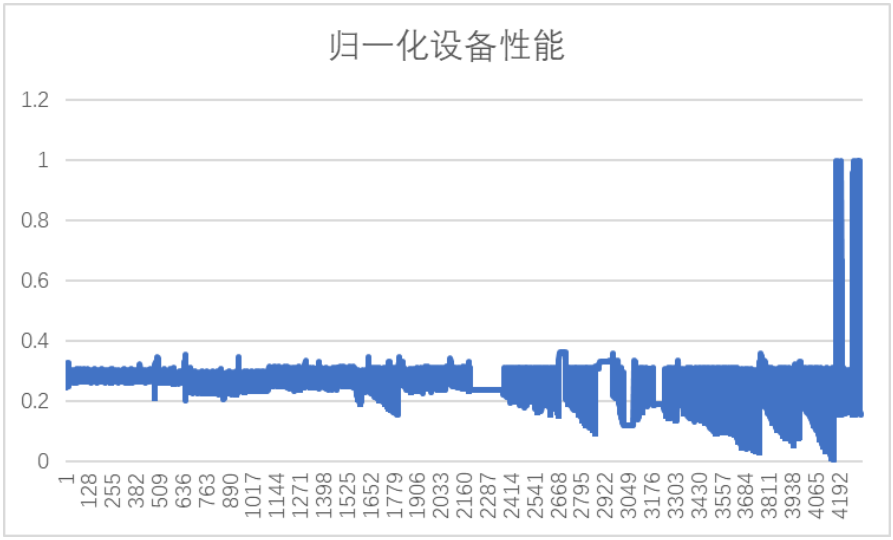


图 7　归一化设备性能

为了降低数据正常波动对后续分析带来的影响，需要先通过滑动窗口进行平滑，关于滑动窗口的大小，本文选择使用 acf 系数估计大小，关于归一化设备性能的 acf 系数图如图8所示。

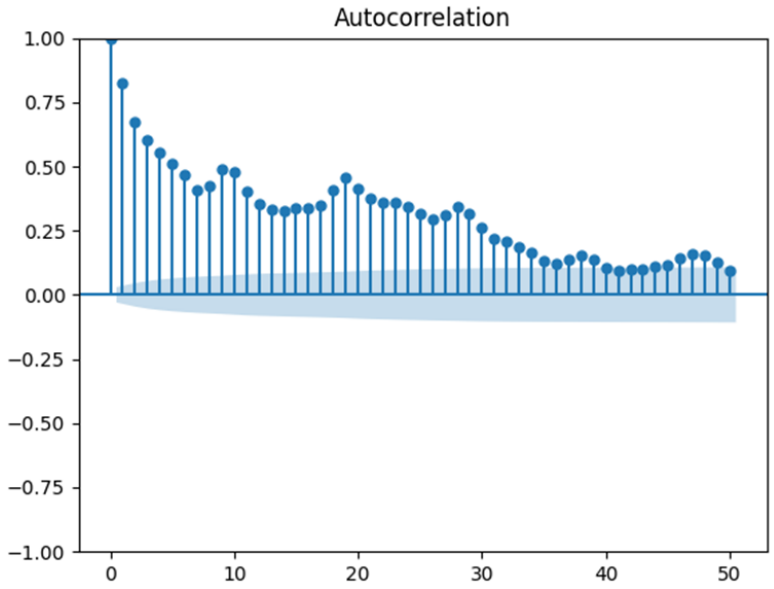


图 8　设备归一化性能的 ACF 系数

从 acf 系数中可以明显的发现，每隔 10 个时间节点，相关系数会有明显的提升，本文认为原始数据中，性能以 10 为周期。采用以 10 作为宽度的滑动窗口采集其一定范围内设备性能的最大，最小，平均值，标准差。绘制如下图9所示

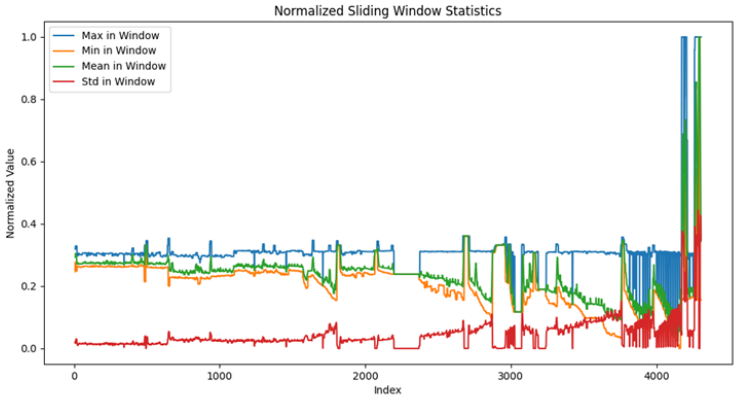


图 9　设备性能的最大，最小，平均值，标准差

基于问题一的分析，本文认为数据集中标准差突变的数据点便是设备异常的时间点，本文选择利用 LSTM 神经网络进行训练，以时间序号，归一化流量，液位，设备前端的三种可能影响因素作为模型输入，归一化的设备性能作为输出，LSTM 的具体解释见 [2]。

7.1.2 设备运行周期分析

同样，假设在设备的运行周期中存在以下几个因素：

1. 正常运行: 对周期性数据，采用 SARIMA 模型作为基础模型，以一开始设备没有故障的运行过程作为基础进行拟合。
2. 设备老化: 以最开始的运行过程作为基准和平均线，以相对于平均值的倍数 k 作为异常情况和老化过程的量化，构建时间序列分解的乘法模型。在图中可以看到，k 值的增加（曲线向下的扩展）在正常情况下并不是连续的，而是运行一段时间后就会自动下降一个幅度，设置为定期增加的分段函数。
3. 异常情况：在之前的模型中可以发现一些异常情况，在这些情况下模型的性能会不断提升，直到达到一个阈值后骤然下降。从图中数据来看，提升速度并没有一个稳定的趋势，而是在一定范围内波动。
4. 人工维护：从图中来看，人工维护的作用是在变异之后把 k 压制回变异之前。从图中可以看到，本次设备的运行周期中，设备的损坏是由于设备在运行过程中不断的出现超载情况，最终由于超载导致设备性能达到 0 导致设备损坏。

鉴于机器超载本身没有危害，最大的危害来自于超载过后的机器崩溃，（结论间接来自性能达到 1 后迅速下降）本文的模型只需要识别崩溃前的预兆时间。

即通过当前数据对周期进行分类：输入前几个周期的数据，识别下一个周期是说明类别（1. 正常 2.k 增加 3. 崩溃）

综上所述，本文构建的模型方程为：

$$e(x)_{base} = f(x) \tag{16}$$

其中：f(x) 为使用 SARIMA 模型根据最开始的最正常的数据拟合的模型，SARIMA 具体模型解释见文献 [3]。观察图7可以发现，效率波动的上界基本没有变化，但下界随着时间增大不断降低，本文认为老化带来的左右会影响到效率低于平均值 (SARIMA 模型拟合的效率的平均值 arg) 的时间中的效率值，使其和平均值的差值增大一定倍数，对这一倍数进行估计，取老化分界点。

$$e_{old} = \begin{cases} e_{base}, e_{base} > avg \\ avg - w_{oldi}(avg - e_{base}), e_{base} < avg \end{cases} \tag{17}$$

其中，$w_i = \frac{avg - e_{real}}{avg - e_{base}}$。本文认为图7中，效率的下界出现不正常的下降为异常，下界下降到一定范围之后就会使机器因超载而损坏，损坏之后下一次的下界就会的更低，直到最后异常使效率达到 0，机器彻底损坏。通过给 w_{old} 上加一个斜率的方式来表现这种异常。

$$w_{errorj} = w_{oldi} + k_j \times (x_t - x_{0j}) \tag{18}$$

$$e_{error} = \begin{cases} e_{base}, & e_{base} > avg \\ avg - w_{error}(avg - e_{base}), & e_{base} < avg \end{cases} \tag{19}$$

其中，w_{error} 为总权值，k_j，$x_0 j$ 为第 j 次变异的权值斜率和开始时间。当 $e_{error} < {}_j$ 时，机器过载，进入损坏阶段（表现为原始数据中，每次一次结束之后都会有一段没有周期的平坦数据）。通过维护可以使得异常情况导致的下界下降恢复，返回到老化水平即使 w_{errorj} 恢复到 w_{oldi}，如果维修发生在崩溃之前，则没有负面影响，如果维修发生在崩溃之后则下一次的 ${}_j$ 等距增长，直到达到 0 后，机器会因为异常损坏，寿命结束。

$$\theta_j = \theta_0 + \Delta\theta_{mean} \times n_{fail} \tag{20}$$

其中，θ_0 和 $\Delta\theta_{mean}$ 为根据原始数据估计的下界和每次损坏的阈值增长值，n_{fail} 为维修不及时导致机器损坏的次数。通过对原始数据中机器崩坏之前最后一个超载数据的绘图，可以发现其大致随损坏次数线性下降，模型假设合理。通过拟合得到斜率和截距，确定初始阈值为 0.166，每发生一次损坏阈值下降 0.16。

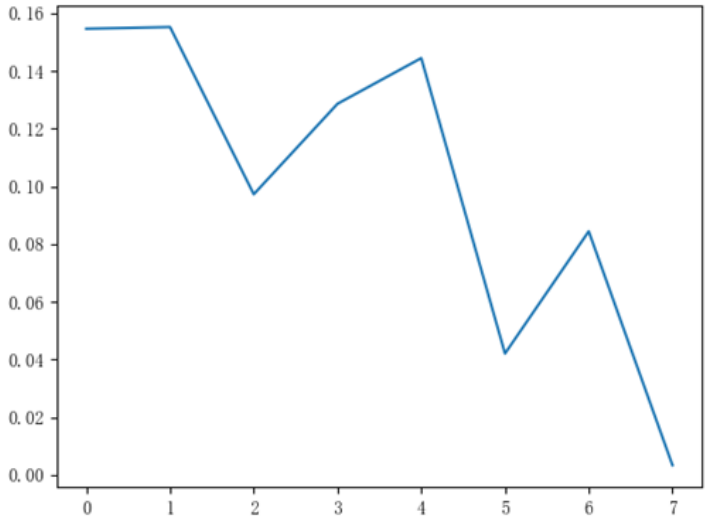


图 10　阈值分析示意图

7.1.3 设备维护周期优化

构建处理体系，本文的目标仍然是通过优化维护时间，减少异常的出现情况，每当模型识别到性能出现异常，便开始判断此时能否进行维护，使阈值尽可能晚的到达 1，进而使机器运行的时间尽可能长。

7.2 模型求解

关于设备异常的识别，基于 LSTM 的训练结果如图所示

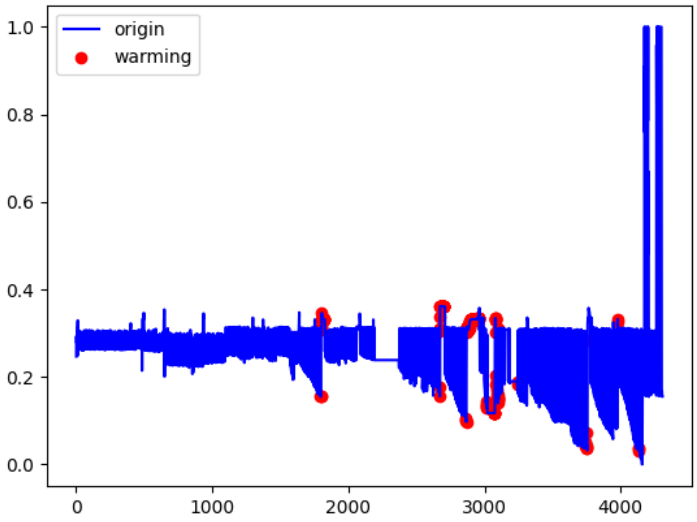
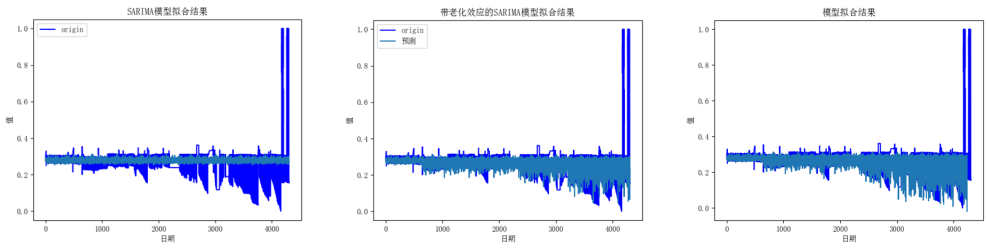


图 11　LSTM 异常检测结果

关于设备运行周期的分析，首先关于拟合 SARIMA 模型，使用 pmdarima 的 auto_arima 包自动确定其它参数，参数为 order=(1,0,2);seasonal_order=(0,0,2,10)，在自然条件，老化条件以及考虑全部条件下的拟合结果示意图如下所示



(a) 自然状态下的性能曲线　　(b) 考虑老化的性能曲线　　(c) 考虑所有情况的性能曲线

图 12　不同情况下的设备性能曲线

关于设备维护周期的优化结果如下图所示：

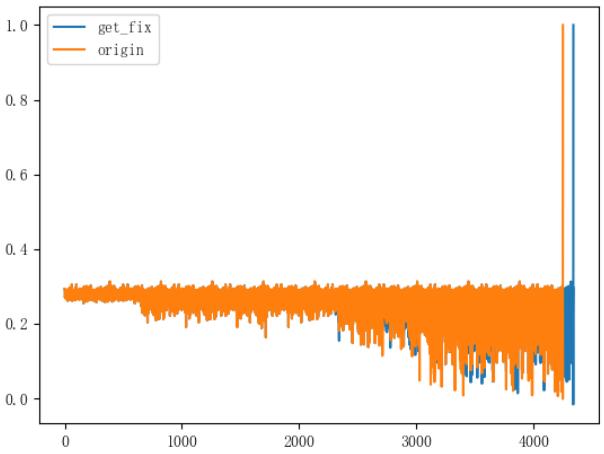


图 13　设备优化与原始方案对比图

原始方案的设备使用寿命为 4243 小时，经过优化后的维护方案，设备的使用寿命达到了 4606 小时。

8. 问题五的模型建立与求解

8.1 模型建立

随机森林的 REF 递归特征消除法是一种基于模型的特征选择方法。它通过构建随机森林模型，并反复训练模型来评估每个特征的重要性，从而递归地消除不重要的特征。REF 方法通过以下步骤实现：1. 计算初始模型在所有特征上的特征重要性得分。2. 去掉得分最低的特征，并重新训练模型。3. 重复步骤 1 和步骤 2，直到选择的特征数达到所需的数量或者达到预定义的停止标准。这种方法的优点是可以在不需要先验知识的情况下对特征进行选择，而且它可以对高维数据集进行有效的特征筛选。此外，REF 方法还可以通过交叉验证来评估模型的性能，以避免过拟合。但是，该方法的缺点是计算成本较高，特别是在处理大型数据集时。本文首先定义了一个自己实现的随机森林分类器，然后定义了一个 REF 函数来选择最重要的特征。在 REF 函数的实现基于以下步骤：

1. 定义一个随机森林分类器，使用 RandomForestClassifier 类来实现。

2. 定义一个包含特征索引和得分的列表 feature_scores。

3. 计算每个特征在随机森林模型中的重要性得分。对于每个特征，本文从原始数据集中删除该特征，然后训练随机森林模型并计算特征重要性得分。本文把特征得分添加到 feature_scores 列表中。

4. 将特征得分转换为平均得分，使用 np.mean 函数实现。

5. 选择前 n_selected_features 个特征的索引，使用 np.argsort 和 [::-1] 来实现。

6. 返回所选特征的索引。

8.2 模型求解

模型的求解结果如下所示

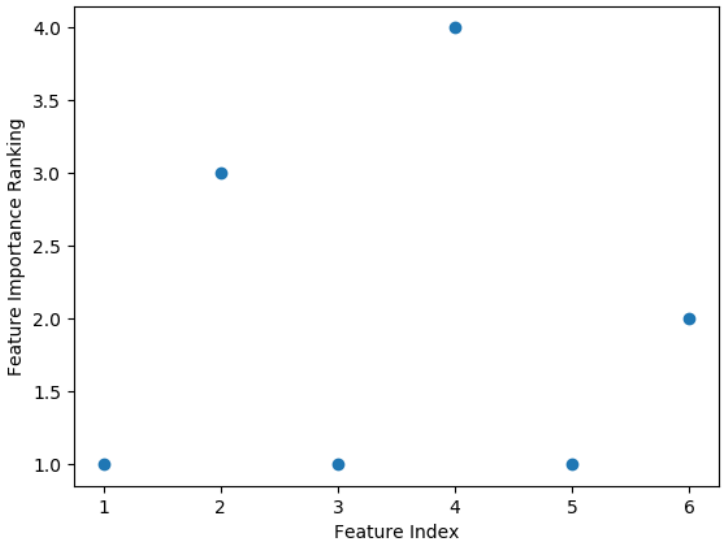


图 14　随机森林的递归特征消失模型

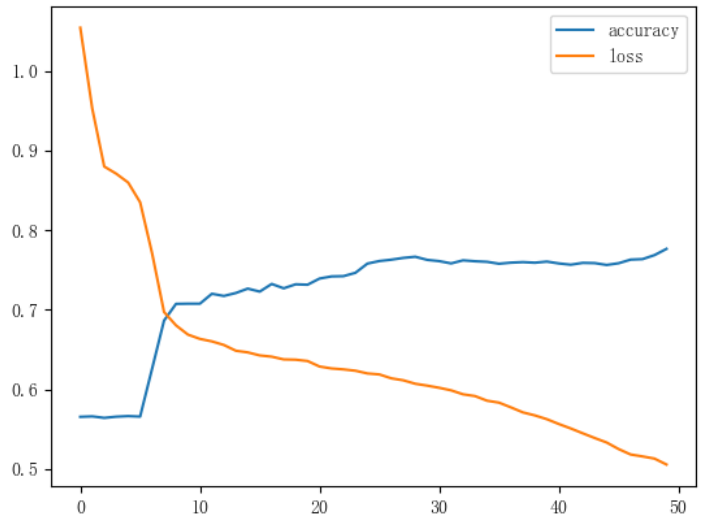


图 15　模型迭代示意图

选取特征重要程度最高的前三个点，分别为归一化流量，归一化温度，设备

前端可能的影响因素 2。经过迭代，模型的准确率达到 75%，交叉熵达到 50%。

9. 模型优缺点

9.1 模型优点

模型简单便捷，通过将实际性能曲线拆分为自然状态，维护状态，异常状态三部分的性能曲线，可以自由组合模拟出不同情况下设备性能曲线，方便探索合适的设备维护策略为工厂生产期间设备维护提供指导。

9.2 模型缺点

关于模型的缺点有以下几部分：

1. 假设条件过于苛刻，假设设备维护次数与设备运行时长没有关系其实不符合实际生产状况，应该添加更具体的有关维护次数的老化因子；
2. 仿真时异常的插入时间的人为固定，没有深入探究设备在不同运行时长时发生异常的不同。

参考文献

[1] 许易经，韩学山，杨明，朱星旭，徐波, 基于设备在线监测的电网状态检修决策模型, 电力系统自动化:72-83, 2020.

[2] Mark, LSTM - 长短期记忆递归神经网络, https://zhuanlan.zhihu.com/p/123857569, 2024.06.10.

[3] Ma Z Z Q W, Lisha;Teng, Online assessment of failure probability for smart meters based on SARIMA-LTFRLS, Electric Power Systems Research:1-11, 2023.

附录 A Python 源程序

```
import numpy as np
import matplotlib.pyplot as plt
import random
import function as F
from scipy.optimize import curve_fit
from pyswarm import pso
import pyswarms as ps
from parameters import Env_parameters
# 设置字体
from pylab import mpl
mpl.rcParams['font.sans-serif'] = ['SimHei']
mpl.rcParams['axes.unicode_minus'] = False

#
    ======================================================================
# 超参数设置
parameters = Env_parameters()
para = parameters.parse_args()
# 初始化超参数
data_path = para.data_path
figure_path = para.figure_path
interval = para.interval
interval_segment = para.interval_segment
normal_segment = para.normal_segment
abnormal_segment = para.abnormal_segment
SLD = para.SLD
SRW = para.SRW
MMI = para.MMI
MTE = para.MTE
min_dis = para.min_dis

# ====================================================

# 读取Excel文件
data = pd.read_excel(data_path)
# 对每段数据进行数据清洗，删除离群点
data_segments = []
```

```
for segment in interval_segment:
    segment_data = data.iloc[segment[0]:segment[1]].dropna()
    segment_data = F.Remove_outliers(segment_data)
    data_segments.append(segment_data)


# 首先拟合模型f(x), 利用三次函数
age_x = np.concatenate([data_segments[i].iloc[:, 0] for i in [0,
    1, 2, 4, 5, 7, 8]])
age_y = np.concatenate([
    data_segments[0].iloc[:, 1],
    data_segments[1].iloc[:, 1] + (data_segments[0].iloc[-1, 1] -
        data_segments[1].iloc[0, 1]),
    data_segments[2].iloc[:, 1] + (data_segments[0].iloc[-1, 1] -
        data_segments[1].iloc[0, 1] + data_segments[1].iloc[-1, 1]
        - data_segments[2].iloc[0, 1]),
    data_segments[4].iloc[:, 1] + (data_segments[0].iloc[-1, 1] -
        data_segments[1].iloc[0, 1] + data_segments[1].iloc[-1, 1]
        - data_segments[2].iloc[0, 1] + data_segments[2].iloc[-1,
        1] - data_segments[4].iloc[0, 1]),
    data_segments[5].iloc[:, 1] + (data_segments[0].iloc[-1, 1] -
        data_segments[1].iloc[0, 1] + data_segments[1].iloc[-1, 1]
        - data_segments[2].iloc[0, 1] + data_segments[2].iloc[-1,
        1] - data_segments[4].iloc[0, 1] +
        data_segments[4].iloc[-1, 1] - data_segments[5].iloc[0,
        1]),
    data_segments[7].iloc[:, 1] + (data_segments[0].iloc[-1, 1] -
        data_segments[1].iloc[0, 1] + data_segments[1].iloc[-1, 1]
        - data_segments[2].iloc[0, 1] + data_segments[2].iloc[-1,
        1] - data_segments[4].iloc[0, 1] +
        data_segments[4].iloc[-1, 1] - data_segments[5].iloc[0, 1]
        + data_segments[5].iloc[-1, 1] - data_segments[7].iloc[0,
        1]),
    data_segments[8].iloc[:, 1] + (data_segments[0].iloc[-1, 1] -
        data_segments[1].iloc[0, 1] + data_segments[1].iloc[-1, 1]
        - data_segments[2].iloc[0, 1] + data_segments[2].iloc[-1,
        1] - data_segments[4].iloc[0, 1] +
        data_segments[4].iloc[-1, 1] - data_segments[5].iloc[0, 1]
        + data_segments[5].iloc[-1, 1] - data_segments[7].iloc[0,
```

```
        1] + data_segments[7].iloc[-1, 1] -
        data_segments[8].iloc[0, 1])
])
age_x = np.array(age_x); age_y = np.array(age_y)
age_params, age_params_covariance = curve_fit(F.f, age_x, age_y)
age_x_fit = np.linspace(min(age_x), max(age_x), 100); age_y_fit
    = F.f(age_x_fit, age_params)
# 画出数据点和拟合曲线
plt.scatter(age_x, age_y, label='老化数据(y轴为:
    累计维护高度+每段数据y值)', color='lightblue')
plt.plot(age_x_fit, age_y_fit, color='black', label='老化拟合曲线')
# 添加标签和图例
plt.xlabel('X'); plt.ylabel('Y'); plt.legend()
plt.savefig('figure/123569段数据清洗后拟合性能趋势图')
#plt.show()

# ================================================
# ================================================

# 利用PSO算法求解回退时间
# t的下界与上界（根据实际情况调整）
lb = [0]; ub = [1000]
delta_x = [data_segments[i].iloc[0, 0] for i in
    normal_segment[1:6]]
delta_y = [data_segments[i].iloc[-1, 1] - data_segments[i +
    1].iloc[0, 1] for i in normal_segment[0:5]]
backoff_time, backoff_error = pso(lambda t: F.pso_goal(t,
    delta_x, delta_y, age_params), lb, ub)
print('回退时间', backoff_time, '回退误差', backoff_error)


# 拟合异常数据段的斜率(4, 7, 10, 11)
local_segments = [data_segments[i].iloc[0:SLD] for i in
    abnormal_segment]
coefficients = [np.polyfit(segment['时间'], segment['性能'], 1)
    for segment in local_segments]
polys = [np.poly1d(coef) for coef in coefficients]
print('================================================')
print('================================================')
```

```
# 计算4条线各自斜率之差
threshold_all = []
for i, segment in enumerate(abnormal_segment):
    plt.scatter(data_segments[segment]['时间'],
        data_segments[segment]['性能'], label=f'数据集{segment +
        1}全部点', color='blue')
    plt.scatter(local_segments[i]['时间'],
        local_segments[i]['性能'], label=f'数据集{segment + 1}局部点',
              color='cyan')
    plt.plot(local_segments[i]['时间'],
        polys[i](local_segments[i]['时间']),
        label=f'拟合曲线{segment+1}',
            color='black')
    print(f'清洗后数据集{abnormal_segment[i] + 1}的局部斜率:
        {coefficients[i][0]}')
    threshold_all.append(coefficients[i][0] -
        F.derivative_f(data_segments[segment].iloc[0,0],
        age_params))
plt.xlabel('时间'); plt.ylabel('性能'); plt.legend()
plt.savefig('figure/在清洗数据后得到的第4_7_10_11段起始点的斜率')
#plt.show()
print('4条线各自的斜率之差:',threshold_all)

# ====================================================
# ====================================================

# 下面计算拟合曲线在3, 6, 9,
    10段的末尾点的斜率(直接代入末尾点的x坐标算就行), 前面拟合的是4, 7, 10,
    11起始点的局部斜率(取了30个点)
# 绘制这四点和它们的导数
points_x = [data_segments[segment-1].iloc[-1,0] for i, segment
    in enumerate(abnormal_segment)]
points_y = [F.f(x, age_params) for x in points_x]
points_slope =
    [F.derivative_f(data_segments[segment-1].iloc[-1,0],
    age_params) for i, segment in enumerate(abnormal_segment)]

plt.scatter(age_x, age_y, label='老化数据(y轴为:
    累计维护高度+每段数据y值)', color='lightblue')
```

```
plt.plot(age_x_fit, age_y_fit, color='black', label='老化拟合曲线')
for (px, py, slope) in zip(points_x, points_y, points_slope):
    plt.scatter(px, py, color='red') # 点的位置
    x_range = np.linspace(px - 10, px + 10, 100)
    y_range = slope (x_range - px) + py
    plt.plot(x_range, y_range, color='red') # 绘制导数直线
# 添加标签和图例
plt.xlabel('X'); plt.ylabel('Y'); plt.legend()
plt.savefig('figure/123569段数据清洗后拟合性能趋势图_带导数')
#plt.show()

# 输出导数值
print('=====================================================')
print('=====================================================')
print('拟合曲线在第3, 6, 9, 10段的末尾点的x坐标和斜率分别为:')
for i, segment in enumerate(abnormal_segment):
    print(f'第{segment}段末尾x坐标 = {points_x[i]},
        第{segment}段末尾斜率 = {points_slope[i]}')
# 将data_segment中多个dataframe合并为一个dataframe
data_all = pd.concat(data_segments)




# ================================================
# ================================================

# 存储4, 7, 10, 11段上每个x的斜率
slope_memory = [[] for i in range(len(abnormal_segment))]
for i, segment in enumerate(abnormal_segment):
    for num in range(interval_segment[segment][1] -
        interval_segment[segment][0] + 1 - SLD):
        x = data_segments[segment].iloc[i, 0]
        slope = F.derivative_f(x, age_params)
        slope_memory[i].append(slope)
# 下面计算k_mean
k_mean = [0 for i in range(len(interval_segment))]
for i, segment in enumerate(abnormal_segment):
```

```
    k_mean[segment] = 1 / (interval_segment[segment][1] -
        interval_segment[segment][0] + 1 - SLD)
        (coefficients[i][0] (interval_segment[segment][1] -
        interval_segment[segment][0] + 1 - SLD) -
        sum(slope_memory[i]))
print('=====================================================')
print('=====================================================')
print('k_mean: ', k_mean)

# ================================================
# ================================================

# 计算斜率阈值
# 用平均值作为斜率
#threshold = np.average(threshold_all[:len(abnormal_segment)])
# 用最小值作为斜率阈值
threshold = min(threshold_all)
print('斜率阈值:', threshold)
print('=====================================================')
print('=====================================================')


# 设置slope_RW用于存放选取时间节点前30个节点的x坐标在拟合曲线上的斜率
slope_fx = []
print('请输入需要判断是否异常的时间节点(整数):')
any_time = int(input())
# 设置slope_ex用于存放选取时间节点前30个节点的x坐标在模型e(x)上的斜率
slope_ex = []
if any_time < SRW:
    slope_ex = [F.derivative_ex(k_mean, backoff_time, x,
        age_params) for x in range(0,any_time)]
else:
    slope_ex = [F.derivative_ex(k_mean, backoff_time, x,
        age_params) for x in range(any_time-SRW,any_time)]
# 筛选出大于斜率阈值min_slope的数据
slope_ex_1 = [slope_ex[i] for i in range(SRW) if slope_ex[i] >
    threshold]
if len(slope_ex_1) >= len(slope_ex)/2:
    print(f'{any_time}时的机器需要维修')
```

```
else:
    print(f'{any_time}时的机器正常，无需维修')
print('=====================================================')
print('=====================================================')

# ======================================
# 下面计算g1(x)
g1 = []
for i, segment in enumerate(interval):
    g1x = F.g1x(data_segments[segment].iloc[:, 0],
        k_mean[segment], data_segments[segment].iloc[0, 0])
    g1.append(g1x)
# 下面计算g2(x)
g2 = []
for i, segment in enumerate(interval):
    g2x = F.g2x(g2, segment, data_segments[segment].iloc[:, 0],
        sum(k_mean[:segment + 1]), data_segments[segment].iloc[0,
        0])
    g2.append(g2x)
# 下面计算h(x)
h = []
for i, segment in enumerate(interval):
    if segment == 0:
        hx = 0
    else:
        x = data_segments[segment].iloc[0, 0]
        hx = F.hx(segment, backoff_time[0], h, x, age_params)
    h.append(hx)
# 下面计算e(x)
e = []
for num, segment in enumerate(interval):
    x = data_segments[segment].iloc[:, 0]
    ex = F.ex(data_segments, h, segment, x, k_mean, age_params)
    e.append(ex)
# 画图
for i, segment in enumerate(interval):
    #plt.scatter(data_segments[segment]['时间'],
        data_segments[segment]['性能'], label=f'数据集{segment +
        1}全部点', color='blue')
```

```
    plt.plot(e[i].index[:], e[i].values, label=f'第{segment +
        1}段的e曲线', color='black')
plt.xlabel('时间'); plt.ylabel('性能'); plt.legend()
plt.savefig('figure/模型e(x).png')
#plt.show()

# ====================================
# ====================================
# 下面计算无异常
Wuyichang = []
for i, segment in enumerate(interval):
    x_total = data_segments[segment].iloc[:, 0]
    # judge用来判断性能是否达到1
    judge = 0
    for num, x in enumerate(x_total):
        Wuyichangx = F.Wuyichangx(h, segment, x, age_params)
        Wuyichang.append((x, Wuyichangx))
        if Wuyichangx >= 1:
            judge = 1
            break
    if judge == 1:
        break
    curve_x, curve_y = zip(Wuyichang)
    plt.plot(curve_x, curve_y, label=f'无异常曲线', color='black')
    Wuyichang = []
# 使用虚线画出对应的 x, y 坐标
plt.plot([curve_x[-1], curve_x[-1]], [0, curve_y[-1]], 'r--',
    linewidth=1)
plt.plot([0, curve_x[-1]], [curve_y[-1], curve_y[-1]], 'r--',
    linewidth=1)
plt.annotate(f'({curve_x[-1]}, 0)', (curve_x[-1], 0),
    textcoords="offset points", xytext=(0, -10), ha='center',
    fontsize=8)
plt.annotate(f'(0, {curve_y[-1]})', (0, curve_y[-1]),
    textcoords="offset points", xytext=(-30, 0), ha='center',
    fontsize=8)
plt.savefig('figure/无异常')
#plt.show()
```

```
# 下面计算无维护
Wuweihu = []
for i, segment in enumerate(interval):
    x_total = data_segments[segment].iloc[:, 0]
    # judge用来判断性能是否达到1
    judge = 0
    for num, x in enumerate(x_total):
        Wuweihux = F.Wuweihux(g2, segment, num, x, age_params)
        Wuweihu.append((x, Wuweihux))
        if Wuweihux >= 1:
            judge = 1
            break
    curve_x, curve_y = zip(Wuweihu)
    plt.plot(curve_x, curve_y, label=f'无维护曲线', color='black')
    if judge == 1:
        break
    Wuweihu = []
# 使用虚线画出对应的 x, y 坐标
plt.plot([curve_x[-1], curve_x[-1]], [0, curve_y[-1]], 'r--',
    linewidth=1)
plt.plot([0, curve_x[-1]], [curve_y[-1], curve_y[-1]], 'r--',
    linewidth=1)
plt.annotate(f'({curve_x[-1]}, 0)', (curve_x[-1], 0),
    textcoords="offset points", xytext=(0, -10), ha='center',
    fontsize=8)
plt.annotate(f'(0, {curve_y[-1]})', (0, curve_y[-1]),
    textcoords="offset points", xytext=(-30, 0), ha='center',
    fontsize=8)
plt.savefig('figure/无维护')
#plt.show()

# 下面计算都无
Douwu = []
for i, segment in enumerate(interval):
    x_total = data_segments[segment].iloc[:, 0]
    # judge用来判断性能是否达到1
    judge = 0
    for num, x in enumerate(x_total):
        Douwux = F.Douwux(x, age_params)
```

```
        Douwu.append((x, Douwux))
        if Douwux >= 1:
            judge = 1
            break
    curve_x, curve_y = zip(Douwu)
    plt.plot(curve_x, curve_y, label=f'无维护曲线', color='black')
    if judge == 1:
        break
    Douwu = []
# 使用虚线画出对应的 x, y 坐标
plt.plot([curve_x[-1], curve_x[-1]], [0, curve_y[-1]], 'r--',
    linewidth=1)
plt.plot([0, curve_x[-1]], [curve_y[-1], curve_y[-1]], 'r--',
    linewidth=1)
plt.annotate(f'({curve_x[-1]}, 0)', (curve_x[-1], 0),
    textcoords="offset points", xytext=(0, -10), ha='center',
    fontsize=8)
plt.annotate(f'(0, {curve_y[-1]})', (0, curve_y[-1]),
    textcoords="offset points", xytext=(-30, 0), ha='center',
    fontsize=8)
plt.savefig('figure/都无')
#plt.show()

# 利用pso算法求解最优维护次数
# 给出确定的维修的点的个数
n = 10
# 计算生成的维修时间, 以及各个时间节点之差
T, T_length = F.interval_generate(n)
x = 0
for i in range(5000):
    # e1(x) = g(x,T) + + f(x,T) + h(x,T)与y=1交点的横坐标
    if F.e1x(T, i, k_mean, backoff_time, age_params) >= 1:
        x = i
        break
print('=====================================================')
print('=====================================================')
print('e1(x) = g(x,T) + f(x,T) + h(x,T)与y=1交点的横坐标为: ', x)

# ==========================================
```

```
# ========================================

# t的下界与上界（根据实际情况调整）
def pso_goal2(T):
    num = 0
    for i in range(n-1):
        if np.any(T[i+1] - T[i] > min_dis):
            num += 1
    return num 200 -x
def fitness_function(T):
    return pso_goal2(T),

lb = [0 for i in range(n)]; ub = [x for i in range(n)]
dimensions = n; n_particles = 10
# 设置PSO算法的参数
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}
optimizer = ps.single.GlobalBestPSO(n_particles=n_particles,
    dimensions=dimensions, bounds=(lb, ub), options=options)
best_solution, best_T = optimizer.optimize(fitness_function,
    iters=100)

print('=====================================================')
print('=====================================================')
print('最佳维护时间:')
best_T = np.sort(best_T)
print(best_T)

# ===================================
# ===================================

print('=====================================================')
print('=====================================================')
# 现在将best_T带入e1x模型中
print('现在将best_T带入e1x模型中:')
x = 0
for i in range(5000):
    # e1(x) = g(x,T) + + f(x,T) + h(x,T)与y=1交点的横坐标
    if F.e1x(best_T, i, k_mean, backoff_time, age_params) >= 1:
        x = i
```

```
        break
print('==================================================')
print('==================================================')
print('best_T时, e1(x) = g(x,T) + f(x,T) +
    h(x,T)与y=1交点的横坐标为: ', x)

# 下面计算原始数据带入e1x模型中
print('下面计算原始数据带入e1x模型中:')
T_origin = [segment[1] for segment in interval_segment if
    segment[0] < interval_segment[-1][0]]
print('原始维护点为:', T_origin)
x = 0
for i in range(5000):
    # e1(x) = g(x,T) + + f(x,T) + h(x,T)与y=1交点的横坐标
    if F.e1x(T_origin, i, k_mean, backoff_time, age_params) >= 1:
        x = i
        break
print('==================================================')
print('==================================================')
print('T_origin时, e1(x) = g(x,T) + f(x,T) +
    h(x,T)与y=1交点的横坐标为: ', x)
```

```
    import pandas as pd
import numpy as np
import function as F
import random
from parameters import Env_parameters
from scipy.optimize import curve_fit, fsolve
from numpy.polynomial.polynomial import Polynomial


# 超参数设置
parameters = Env_parameters()
para = parameters.parse_args()
# 初始化超参数
data_path = para.data_path
figure_path = para.figure_path
interval = para.interval
interval_segment = para.interval_segment
```

```
normal_segment = para.normal_segment
abnormal_segment = para.abnormal_segment
SLD = para.SLD
SRW = para.SRW
MMI = para.MMI
MTE = para.MTE
min_dis = para.min_dis


# 删除数据中的离群点
def Remove_outliers(data):
    mean = data['性能'].mean()
    std_dev = data['性能'].std()
    # 定义阈值，例如，超过均值加减2倍标准差的点视为离群点
    threshold = 2 std_dev
    # 根据阈值删除离群点
    cleaned_data = data[(data['性能'] >= mean - threshold) &
        (data['性能'] <= mean + threshold)]

    return cleaned_data

# 定义模型函数f，例如，三次函数(可修改)
def f(x, a, b, c, d):
    return a x3 + b x2 + c x + d

# 定义模型函数f的导数derivative_f
def derivative_f(x, a, b, c, d):
    return 3 a x2 + 2 b x + c

# 定义目标函数 pso_goal(t)
def pso_goal(t, x_value, delta_y, a, b, c, d):
    g_val = 0
    for i in range(len(x_value)):
        xi = x_value[i]
        delta_y_i = delta_y[i]
        g_val += (f(xi, a, b, c, d) - f(xi - t, a, b, c, d) -
            delta_y_i)2
    return g_val
```

```
# 定义函数gx
def g1x(x, k, x0):
    g1x = k (x - x0)
    return g1x

def g2x(g2, segment, x, k, x0):
    g2x = k (x - x0)
    if segment != 0:
        g2x = g2x + g2[segment-1].values[-1] - g2x.values[0]
    return g2x

def hx(segment, backoff_time, h, x, a, b, c, d):
    hx = F.f(x, a, b, c, d) - F.f(x - backoff_time, a, b, c, d) +
        h[segment - 1]
    return hx

def h2x(T, backoff_time, x, a, b, c, d):
    x_i0, x_i1 = F.find_interval(T, x)
    h2x = 0
    for i, segment in enumerate(T):
        if x > segment:
            if segment > backoff_time:
                h2x = h2x - (F.f(segment, a, b, c, d) - F.f(segment
                    - backoff_time, a, b, c, d))
            else:
                h2x = h2x - F.f(segment, a, b, c, d)
    return h2x
def ex(data_segments, h, segment, x, k_mean, a, b, c, d):
    if segment == 3 or segment == 6 or segment == 9 or segment ==
        10:
        e = F.f(x, a, b, c, d) + F.g1x(x, k_mean[segment],
            data_segments[segment].iloc[0, 0]) - h[segment]
    else:
        e = F.f(x, a, b, c, d) - h[segment]

    return e

def derivative_ex(k_mean, backoff_time, x, a, b, c, d):
    k = 0
```

```
    for i, (start, end) in enumerate(interval_segment, start=1):
        if start <= x <= end:
            k = k_mean[i-1]
            break
    return k

def Wuyichangx(h, segment, x, a, b, c, d):
    wuyichang = F.f(x, a, b, c, d) - h[segment]

    return wuyichang

def Wuweihux(g, segment, num, x, a, b, c, d):
    wuweihu = F.f(x, a, b, c, d) + g[segment].values[num]

    return wuweihu

def Douwux(x, a, b, c, d):
    douwu = F.f(x, a, b, c, d)
    return douwu

def generate_random_points(n):
    points = [random.uniform(0, 5000) for _ in range(n)]
    return sorted(points)

def find_interval(points, value):
    for i in range(len(points) - 1):
        if points[i] <= value < points[i + 1]:
            return points[i], points[i+1]
    return 0,0
def find_interval2(points, value):
    for i in range(len(points) - 1):
        if points[i][0] <= value <= points[i][1]:
            return points[i][0]
    a=1
    return 0

def g_val2(T, k, backoff_time, t_i0, x_i0, x, a, b, c, d):
    return F.g1x(x, k, max(t_i0, x_i0)) + F.f(x, a, b, c, d) +
        h2x(T, backoff_time, x, a, b, c, d)
```

```
def interval_generate(n):
    # 先生成 1 到 5000 之间的 10 个不同的随机数
    random_numbers = random.sample(range(1, 5001), n)
    # 从小到大排序
    sorted_numbers = sorted(random_numbers)
    # 计算相邻点之间的差值
    differences = [sorted_numbers[i + 1] - sorted_numbers[i] for
        i in range(len(sorted_numbers) - 1)]

    return sorted_numbers, differences


def e1x(T, x, k_mean, backoff_time, a ,b, c, d):
    t_i0, t_i1 = F.find_interval(T, x)
    x_i0 = F.find_interval2(interval_segment, x)
    k = 0
    if 860<=x_i0<= 1018 or 1724<=x_i0<= 1882 or 2539<=x_i0<=2734
        or 2738<=x_i0<=2919:
        if 860<=x_i0<= 1018: k = k_mean[3]
        elif 1724<=x_i0<= 1882: k = k_mean[6]
        elif 2539<=x_i0<=2734: k = k_mean[9]
        else: k = k_mean[10]
    g_val2 = F.g_val2(T, k, backoff_time, t_i0, x_i0, x, a, b, c,
        d)

    return g_val2

def pso_goal2(T, n, x):
    num = 0
    for i in range(n-1):
        if np.any(T[i+1] - T[i] > min_dis):
            num += 1
    return num 200 - x
```

```
    import argparse
import numpy as np

class Env_parameters():
```

```
def parse_args(self):
    parser = argparse.ArgumentParser('Env_parameters')

    # 定义模型参数
    parser.add_argument('--data_path', default='data/C1.xlsx',
                        help='Data file path')
    parser.add_argument('--figure_path', default='figure',
                        help='Figure file path')
    parser.add_argument('--interval', default=[0, 1, 2, 3, 4,
        5, 6, 7, 8, 9, 10],
                        help='interval')
    parser.add_argument('--interval_segment', type=list,
        default=[(0, 287), (293, 592), (592, 856), (860, 1018),
        (1022, 1251), (1279, 1685), (1724, 1882), (1892, 2071),
        (2078, 2515), (2539, 2734), (2738, 2919)],
        help='Interval of each segment')

    parser.add_argument('--normal_segment', default=[0, 1, 2,
        4, 5, 7, 8],
                        help='Segments use for polynomial fitting')
    parser.add_argument('--abnormal_segment', default=[3, 6,
        9, 10],
                        help='Segments use for computing k_mean')
    parser.add_argument('--SLD', type=int, default=30,
                        help='Size of local data for fitting k_mean
                            model ')
    parser.add_argument('--SRW', type=int, default=30,
                        help='Size of rolling window')
    parser.add_argument('--MMI', type=int, default=200,
                        help='Minimum maintenance
                            interval(最小维护时间间隔)')
    parser.add_argument('--MTE', type=int, default=30,
                        help='Maintenance time error(维护时间误差)')
    parser.add_argument('--min_dis', type=int, default=158,
                        help='原始数据所有维护间隔中的最小距离')

    return parser.parse_args()
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
import numpy as np
from matplotlib.font_manager import FontProperties
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pmdarima import auto_arima
import pickle
import bisect
import math

# 设置中文字体，例如SimHei或者微软雅黑
font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc",
    size=14)
plt.rcParams['font.family'] = font.get_name()


def preview(df, window_size):
    dataframe = df.copy()
    dataframe['max_in_window'] =
        dataframe['归一化设备性能'].rolling(window=window_size).max()
    dataframe['min_in_window'] =
        dataframe['归一化设备性能'].rolling(window=window_size).min()
    dataframe['mean_in_window'] =
        dataframe['归一化设备性能'].rolling(window=window_size).mean()
    dataframe['std_in_window'] =
        dataframe['归一化设备性能'].rolling(window=window_size).std()
    # 绘制折线图
    plt.figure(figsize=(12, 6))
    plt.plot(dataframe.index, dataframe['max_in_window'],
        label='Max in Window')
    plt.plot(dataframe.index, dataframe['min_in_window'],
        label='Min in Window')
    plt.plot(dataframe.index, dataframe['mean_in_window'],
        label='Mean in Window')
    plt.plot(dataframe.index, dataframe['std_in_window'],
        label='Std in Window')

    plt.title('Normalized Sliding Window Statistics')
    plt.xlabel('Index')
```

```
    plt.ylabel('Normalized Value')
    plt.legend()
    plt.show()


def corridor_view(df, window_size):
    dataframe = df.copy()
    dataframe['max_in_window'] =
        dataframe['归一化设备性能'].rolling(window=window_size).max()
    dataframe['min_in_window'] =
        dataframe['归一化设备性能'].rolling(window=window_size).min()
    dataframe['mean_in_window'] =
        dataframe['归一化设备性能'].rolling(window=window_size).mean()
    # 绘制折线图
    plt.figure(figsize=(12, 6))
    plt.plot(dataframe.index, dataframe['max_in_window'],
        label='Max in Window')
    plt.title('Normalized Sliding Window Statistics')
    plt.xlabel('Index')
    plt.ylabel('Normalized Value')
    plt.legend()
    plt.show()


def window_view(dataframe, center, width):
    selected_df = dataframe.iloc[center - width:center + width]
    # 绘制折线图
    plt.figure(figsize=(10, 6))
    for column in selected_df.columns:
        plt.plot(selected_df.index, selected_df[column],
            label=column)

    # 添加图例
    plt.legend(loc='upper right')

    # 添加标题和标签
    plt.title('Line Plot Around Time' + str(center))
    plt.xlabel('Time')
    plt.ylabel('Value')
```

```
    plt.show()


def base_function(model, train_length, pre_x,
    origin_y0=0.29339298):
    fitted_values = model.fittedvalues.clip(lower=0.26)
    fitted_values.iloc[0] = origin_y0 # 第一个值空缺，默认为0，换成原值
    pre_y = fitted_values.iloc[pre_x % train_length]
    return pre_y


def old_function(x, index, ws):
    if x <= max(index):
        position_left = bisect.bisect_left(index, x)
        w = ws[position_left]
    else:
        width = round(max(index) / len(index))
        position = math.floor(x / width)
        w_width = (max(ws) - min(ws)) / len(ws)
        w = w_width (position - len(index) + 1) + max(ws)
    return w


def min_mean(series):
    mins = series.rolling(window=10).min().dropna()
    means = series.rolling(window=10).mean().dropna()
    return mins.mean(), means.mean()


def error_function(x, indexes, w_errors, threshold=2300):
    find = False
    for i in range(len(indexes)):
        index = indexes[i]
        if min(index) <= x <= max(index):
            w_error = w_errors[i] (x-index[0])
            find = True
    if not find:
        if x <= threshold:
            w_error = 0
```

```
        else:
            w_error = np.random.uniform(min(w_errors),
                max(w_errors))
    return w_error


class SimulationModel(object):
    def __init__(self, sarima):
        # 模型固定参数
        self.theta_slope = -0.01861380145238094
        self.theta_intercept = 0.1664436513333333
        self.sarima = sarima
        self.w_olds = [648, 2371, 3240, 4263]
        self.ws = [ 0.9180667,3.95835864,5.72486277,10.9876563 ]
        self.err_index = [[1644, 1797, 1827], [2371, 2671, 2709],
            [2710, 2866, 2964], [2965, 3017, 3082],
                    [3242, 3311, 3318], [3319, 3753, 3779], [3780,
                        3974, 3981], [3982, 4158, 4181]]
        self.threshold = 2300
        _, self.mean_model = min_mean(sarima.fittedvalues)
        self.w_error = [0.043744198823861186,
            0.012574428517164268, 0.05979140014409433,
            0.16395539389911531,
                     0.032229129285617164, 0.02910982731208004,
                        0.05165442925138477, 0.09937570560821894]

        # 模拟可变参数
        self.trace = []
        self.w_old = 1
        self.theta = 0.1664436513333333
        self.in_error = False
        self.error_k = 0
        self.error_start = 0
        self.broke = False


    def base_function(self, pre_x, origin_y0=0.29339298):
        fitted_values = self.sarima.fittedvalues.clip(lower=0.25)
        train_length = len(fitted_values)
```

```
        fitted_values.iloc[0] = origin_y0 #
            第一个值空缺，默认为0，换成原值
        pre_y = fitted_values.iloc[pre_x % train_length]
        return pre_y

    def get_w_old(self, x):
        ws = self.ws
        index = self.w_olds
        if x <= max(index):
            position_left = bisect.bisect_left(index, x)
            w = ws[position_left]
        else:
            width = round(max(index) / len(index))
            position = math.floor(x / width)
            w_width = (max(ws) - min(ws)) / len(ws)
            w = w_width (position - len(index) + 1) + max(ws)
        return w

    def get_error_k(self, x):
        find = False
        indexes = self.err_index
        w_errors = self.w_error
        for i in range(len(indexes)):
            index = indexes[i]
            if min(index) <= x <= max(index):
                k = w_errors[i]
                find = True
        if not find:
                k = np.random.uniform(min(w_errors), max(w_errors))
        self.error_k = k
        return k

    def get_w_error(self,x):
        if self.in_error:
            w_error = (x-self.error_start) self.error_k
        else:
            w_error = 0
        return w_error
```

```
def predict(self,x):
    base = self.base_function(x)
    if base < self.mean_model:
        w_old = self.get_w_old(x)
        _ = self.get_error_k(x)
        w_error = self.get_w_error(x)
        pre = self.mean_model - (self.mean_model - base)
            (w_error + w_old)
    else:
        pre = base
    return pre

def state_renew(self,x,pre):
    # 检测异常状态
    if self.in_error:
        # 判断是否需要结束
        if pre < self.theta:
            self.theta = self.theta + self.theta_slope
            self.in_error = False
    else:
        # 判断是否需要开始
        for index in self.err_index:
            if x == index[0]:
                self.in_error = True
                self.error_start = x
        if x >= self.threshold:
            self.in_error = True
            self.error_start = x

def simulation(self):
    x = 0
    while not self.broke:
        pre = self.predict(x)
        self.state_renew(x, pre)
        if pre < self.theta:
            pre = self.predict(x)
        if pre < 0:
            self.broke = True
        self.trace.append(pre)
```

```
            x = x + 1
        return self.trace


if __name__ == '__main__':
    # 导入数据
    features =
        pd.read_csv(r'../2024研究生校内赛题目/C题附件/feature.csv',
        index_col=0)
    features.drop(features.index[0], inplace=True) #
        第一行的0是个异常值
    miss_corridor = [[2192, 2370], [3084, 3241]]
    error_corridor = [[1644, 1797, 1827], [2371, 2671, 2709],
        [2710, 2866, 2964], [2965, 3017, 3082],
                  [3242, 3311, 3318], [3319, 3753, 3779], [3780,
                     3974, 3981], [3982, 4158, 4181]]
    old_corridor = [648, 2371, 3240, 4263]
    base_corridor = [[15, 452], [648, 698], [2371, 2421], [3240,
        3290]]
    # 选择任务
    view = False
    fit_sarima = False
    get_w1 = False
    get_w_error = False
    test = False
    get_theta = False
    model_test = True
    if view:
        # ACF
        plot_acf(features['归一化设备性能'], lags=50)
        plt.show()
        plt.scatter(features.index, features['归一化设备性能'])
        preview(features, 10)
        window_view(features, 151, 150)
        indexes = [650, 1700, 2308, 2732, 3106, 3352, 3726, 3875,
            4150]
        for index in indexes:
            window_view(features, index, 150)
    if fit_sarima:
```

```
    get_para = False
    train = features['归一化设备性能'][15:452]
    # plt.plot(train)
    # plt.show()
    if get_para:
        # 自动选择最优参数
        auto_model = auto_arima(train, seasonal=True, m=10,
            trace=True)
        #
        # 输出最优参数(1, 0, 2)x(0, 0, 2, 10)
        print(auto_model.summary())
    else:
        # 使用自动选择的参数来拟合SARIMA模型
        model = SARIMAX(train,
                        order=(1, 0, 2),
                        seasonal_order=(0, 0, 2, 10))

        sarima_model = model.fit()

        # 进行预测
        fitted_values = sarima_model.fittedvalues

        # 可视化结果
        plt.figure(figsize=(10, 6))
        plt.plot(train.index, train, label='原始数据')
        plt.plot(train.index, fitted_values, label='预测值',
            color='blue')

        plt.xlabel('日期')
        plt.ylabel('值')
        plt.title('SARIMA模型预测')
        plt.legend()
        plt.show()

        # 保存模型
        with open('sarima_model.pkl', 'wb') as pkl:
            pickle.dump(sarima_model, pkl)
if get_w1:
    # 加载模型
```

```
with open('sarima_model.pkl', 'rb') as pkl:
    sarima_model = pickle.load(pkl)

origin = features['归一化设备性能']
plt.plot(origin, color='blue', label='origin')
pred = np.zeros(len(features))
for i in range(len(features)):
    pred[i] = base_function(sarima_model,
        len(features['归一化设备性能'][15:452]), i)
plt.plot(pred)
plt.xlabel('日期')
plt.ylabel('值')
plt.title('SARIMA模型拟合结果')
plt.legend()
plt.show()

mins = []
for index in base_corridor:
    start = index[0]
    end = index[1]
    series = features['归一化设备性能'][start:end]
    min_s, _ = min_mean(series)
    mins.append(min_s)
min_model, mean_model = min_mean(sarima_model.fittedvalues)
wid0 = mean_model - min_model
wids = mean_model - np.array(mins)
ws = wids / wid0
print(ws)
origin = features['归一化设备性能']
plt.plot(origin, color='blue', label='origin')
diff = np.zeros(len(features))
for i in range(len(features)):
    base_i = base_function(sarima_model,
        len(features['归一化设备性能'][15:452]), i)
    if base_i < mean_model:
        pred[i] = mean_model - (mean_model - base_i)
            old_function(i, old_corridor, ws)
        diff[i] = (mean_model - base_i) old_function(i,
            old_corridor, ws)
```

```
    plt.plot(pred,label='预测')
    # plt.plot(diff, label='老化')
    plt.xlabel('日期')
    plt.ylabel('值')
    plt.legend()
    plt.title('带老化效应的SARIMA模型拟合结果')
    plt.show()
    print('done')
if get_w_error:
    # 加载模型
    with open('sarima_model.pkl', 'rb') as pkl:
        sarima_model = pickle.load(pkl)
    w_old = [ 0.9180667,3.95835864,5.72486277,10.9876563 ]
    min_model, mean_model = min_mean(sarima_model.fittedvalues)
    wid0 = mean_model - min_model
    w_errors = []
    # 计算异常斜率
    for index in error_corridor:
        features['min_in_window'] =
            features['归一化设备性能'].rolling(window=10).min()
        x_data = features.index[index[0]:index[1]]
        y_data = features['min_in_window'][index[0]:index[1]]
        # 使用 numpy.polyfit 进行一阶线性拟合
        slope, intercept = np.polyfit(x_data, y_data, 1)
        w_error = - slope / wid0
        w_errors.append(w_error)
    print(w_errors)
    pred = np.zeros(len(features))
    for i in range(len(features)):
        base_i = base_function(sarima_model,
            len(features['归一化设备性能'][15:452]), i)
        if base_i < mean_model:
            pred[i] = mean_model - (mean_model - base_i)
                (old_function(i, old_corridor, w_old) +
                    error_function(i, error_corridor, w_errors))
            if pred[i]<0:
                print('crash')
        else:
            pred[i] = base_i
```

```
    plt.plot(pred, label='model')
    origin = features['归一化设备性能']
    plt.plot(origin, color='blue', label='origin')
    plt.xlabel('日期')
    plt.ylabel('值')
    plt.title('老化和变异效应的SARIMA模型拟合结果')
    plt.show()
    print('done')
    #
       [0.043743515207174025,0.012711859850471364,0.05975965184957367,0.171464
    # 0.02904503212758782, 0.050794175309296814,
       0.07693958886971206]
if test:
    with open('sarima_model.pkl', 'rb') as pkl:
       sarima_model = pickle.load(pkl)

    min_model, mean_model = min_mean(sarima_model.fittedvalues)
    w_old = [0.97141064, 4.20797395, 6.61174103, 10.89324291]

    diff = np.zeros(len(features))
    w = np.zeros(len(features))
    for i in range(len(features)):
       base_i = base_function(sarima_model,
          len(features['归一化设备性能'][15:452]), i)
       if base_i < mean_model:
          w[i] = old_function(i, old_corridor, w_old)
          diff[i] = (mean_model - base_i) old_function(i,
             old_corridor, w_old)
    plt.plot(w, label='权值')
    plt.xlabel('日期')
    plt.ylabel('值')
    plt.show()
    plt.plot(diff, label='下方差值')
    plt.xlabel('日期')
    plt.ylabel('值')
    plt.show()
    print('done')
    move_mean =
       sarima_model.fittedvalues.rolling(window=10).mean()
```

```
        mean = move_mean.mean()
        plt.plot(sarima_model.fittedvalues, label='拟合值')
        plt.plot(move_mean, label='滑动平均')
        plt.plot(np.ones(len(move_mean))mean, label='滑动平均的平均')
        plt.legend()
        plt.xlabel('日期')
        plt.ylabel('值')
        plt.show()
    if get_theta:
        thresholds = []
        for index in error_corridor:
            thresholds.append(features['归一化设备性能'][index[1]])
        plt.plot(np.array(thresholds))
        plt.show()
        slope, intercept = np.polyfit(np.arange(len(thresholds)),
            np.array(thresholds), 1)
        print(slope, intercept) # -0.01861380145238094
            0.1664436513333333
    if model_test:
        with open('sarima_model.pkl', 'rb') as pkl:
            sarima_model = pickle.load(pkl)

        model = SimulationModel(sarima_model)
        trace = model.simulation()

        origin = features['归一化设备性能']
        plt.plot(origin, color='blue', label='origin')
        plt.plot(np.array(trace), label='model')
        plt.xlabel('日期')
        plt.ylabel('值')
        plt.title('模型拟合结果')
        plt.show()
```

```
    import torch
import pandas as pd
import matplotlib.pyplot as plt
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.dataset import random_split
import torch.optim as optim
```

```
import torch.nn as nn
import numpy as np
import pickle
from pylab import mpl
import copy
mpl.rcParams['font.sans-serif'] = ['SimHei']
mpl.rcParams['axes.unicode_minus'] = False


from question4_1 import SimulationModel


class LSTMClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTMClassifier, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size,
            batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, (h_n, _) = self.lstm(x)
        out = self.fc(h_n[-1])
        return out


def make_dataset(dataframe, indexes, timesteps=10):
    # 输入数据集，和产生变异，崩溃的时间点
    # 默认值，正常数据标记为0
    dataframe['label'] = np.zeros(len(dataframe))
    for i in range(len(dataframe)):
        # 遍历一遍，根据点的未来是否有变异和崩溃设置label
        # 变异为1，崩溃为2
        for index in indexes:
            if (i+10) <= index[0] or i > index[2]:
                # 不在区间内
                continue
            elif (i+10) <= index[1]:
                # 有异常
```

```
            dataframe.loc[i, 'label'] = 1
        elif i <= index[2]:
            # 有损坏
            dataframe.loc[i, 'label'] = 2
    data_array = dataframe.values
    print((dataframe['label'] == 0).sum())
    print((dataframe['label'] == 1).sum())
    print((dataframe['label'] == 2).sum())

    # 计算样本数
    num_samples = data_array.shape[0] - timesteps + 1

    # 初始化 LSTM 输入数据的张量
    X = np.zeros((num_samples, timesteps, data_array.shape[1]-1))
    Y = np.zeros((num_samples, 1))
    # 创建时间步长序列
    for i in range(num_samples):
        X[i] = data_array[i:i + timesteps, :-1]
        Y[i] = data_array[i + timesteps-1, -1]

    # 将 NumPy 数组转换为 PyTorch 张量
    X_tensor = torch.tensor(X, dtype=torch.float32)
    Y_tensor = torch.tensor(Y, dtype=torch.int)

    return X_tensor, Y_tensor


if __name__ == '__main__':
    train = False
    use = True

    features =
        pd.read_csv(r'../2024研究生校内赛题目/C题附件/feature.csv',
        index_col=0)
    select_columns = ['归一化流量', '归一化温度',
        '设备前端可能的影响因素2']
    select_features = features[select_columns]
    error_corridor = [[1644, 1797, 1827], [2371, 2671, 2709],
        [2710, 2866, 2964], [2965, 3017, 3082],
```

```
                [3242, 3311, 3318], [3319, 3753, 3779], [3780,
                    3974, 3981], [3982, 4158, 4181]]
X_tensor, Y_tensor = make_dataset(features, error_corridor)
if train:
    # 创建 TensorDataset 对象
    dataset = TensorDataset(X_tensor, Y_tensor.long())

    # 定义训练集大小和验证集大小
    train_size = int(0.8 len(dataset))
    val_size = len(dataset) - train_size

    # 随机划分数据集为训练集和验证集
    train_dataset, val_dataset = random_split(dataset,
        [train_size, val_size])

    # 创建 DataLoader 对象
    train_loader = DataLoader(train_dataset, batch_size=32,
        shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=32,
        shuffle=False)
    # 初始化模型、损失函数和优化器
    model = LSTMClassifier(input_size=X_tensor.shape[2],
        hidden_size=64, output_size=3)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.0001)

    # 训练模型
    num_epochs = 50
    accuracies = []
    losses = []
    for epoch in range(num_epochs):
        model.train()
        accuracy = 0.0
        val_loss = 0.0
        for inputs, targets in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs.squeeze(),
                targets.squeeze())
```

```
            loss.backward()
            optimizer.step()
            # 获取每个样本的最大概率对应的类别索引
            predicted_classes = torch.argmax(outputs, dim=1)
            # 计算预测正确的样本数量
            correct = (predicted_classes ==
                targets.squeeze()).sum().item()
            # 计算准确率
            accuracy = accuracy + correct / len(targets)
            val_loss = val_loss + loss.item()

        val_loss /= len(train_loader)
        print(f"Epoch [{epoch + 1}/{num_epochs}], Train Loss:
            {loss.item():.4f}")
        accuracy /= len(train_loader)
        losses.append(val_loss)
        accuracies.append(accuracy)
        # 打印准确率
        print("准确率:", accuracy)



# 验证模型
plt.plot(np.array(accuracies), label='accuracy')
plt.plot(np.array(losses), label='loss')
plt.legend()
plt.show()
model.eval()
with torch.no_grad():
    val_loss = 0.0
    accuracy = 0.0
    for inputs, targets in val_loader:
        outputs = model(inputs)
        val_loss += criterion(outputs.squeeze(),
            targets.squeeze()).item()
        # 获取每个样本的最大概率对应的类别索引
        predicted_classes = torch.argmax(outputs, dim=1)
        # 计算预测正确的样本数量
        correct = (predicted_classes ==
```

```
                targets.squeeze()).sum().item()
            # 计算准确率
            accuracy = accuracy + correct / len(targets)
        val_loss /= len(val_loader)
        print(f"Validation Loss: {val_loss:.4f}")
        accuracy /= len(val_loader)
        print("准确率:", accuracy)

    torch.save({
        'epoch': num_epochs,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
    }, 'trained_model1.pth')
if use:
    # 加载模型
    loaded_model = model =
        LSTMClassifier(input_size=X_tensor.shape[2],
        hidden_size=64, output_size=3) #
        这里需要根据你的模型类型来初始化一个新的模型对象
    checkpoint = torch.load('trained_model1.pth')
    loaded_model.load_state_dict(checkpoint['model_state_dict'])
    loaded_optimizer = optim.Adam(loaded_model.parameters()) #
        重新初始化优化器对象
    loaded_optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    epoch = checkpoint['epoch']

    outputs = loaded_model(X_tensor)
    # 获取每个样本的最大概率对应的类别索引
    predicted_classes = torch.argmax(outputs, dim=1)
    indices = (predicted_classes ==
        2).nonzero(as_tuple=True)[0]
    origin = features['归一化设备性能']
    plt.plot(origin, color='blue', label='origin')
    plt.scatter(indices.tolist(), origin[indices.tolist()],
        color='red', label='warming')
    plt.legend()
    plt.show()

    with open('sarima_model.pkl', 'rb') as pkl:
```

```
        sarima_model = pickle.load(pkl)

    simulation_model0 = SimulationModel(sarima_model)
    trace0 = simulation_model0.simulation()

    simulation_model1 = SimulationModel(sarima_model)
    simulation_model1.get_lstm(loaded_model, X_tensor)
    simulation_model1.use_fix = True
    trace1 = copy.deepcopy(simulation_model1.simulation())

    plt.plot(np.array(trace1), label='get_fix')
    plt.plot(np.array(trace0), label='origin')
    plt.legend()
    plt.show()
```