



中国研究生创新实践系列大赛  
“华为杯”第二十一届中国研究生  
数学建模竞赛

学 校 上海交通大学

---

参赛队号 24102480006

---

1. 左润衡

---

队员姓名

2. 朱家杰

---

3. 吕德淞

---

**中国研究生创新实践系列大赛**  
**“华为杯”第二十一届中国研究生**  
**数学建模竞赛**

题 目： 数据物理耦合的磁芯原件损耗建模

---

**摘 要：**

磁芯材料对构建高效、可靠的电能变换系统起到了关键性作用。在实际应用中，磁芯材料会受温度、励磁波形、励磁频率、磁芯材料、磁通密度峰值等因素影响，产生不同程度的老化。现有的方法还不足以支撑跨材料不同工况下的磁芯损耗精确预测。研究相关因素对磁芯损耗的相关性至关重要，并且根据关联关系对现有的磁芯损耗模型进行修正仍是亟待解决的问题，修正方程的方式方法影响着模型预测的精度和泛化性。随着机器学习算法的发展，基于数据驱动的方法可以实现跨材料不同工况下的预测，在此基础上研究磁芯损耗和传输磁能的多目标优化问题为磁芯材料的实际应用具有极高的指导意义。

针对问题一，首先利用数据的二阶导数特征进行初步筛除正弦波，再利用一阶导特征基于 DBSCAN 聚类方法区分三角波和梯形波，经检验，该预测模型在训练集中的准确率为 100%，证明该分类方法的高效性，并预测了附件二中包含的 80 例未标注激励波形的测试数据。

针对问题二，首先将每种温度下的频率、磁通密度峰值作为自变量，将磁芯损耗作为因变量，根据传统的斯坦麦茨方程形式进行拟合。其次结合磁滞损耗和涡流损耗的实际物理背景，采用对数函数将各拟合系数关于温度进行拟合，得到修正后的斯坦麦茨方程。通过对原始斯坦麦茨方程与引入温度修正后的拟合结果，发现 MSE 减小至修正前的 7.19%，与不同温度下拟合 MSE 结果在同一量级，表明了改进的方法的合理性，模型的预测精度得到了显著提升。

针对问题三，本文采用非参数回归分析定性探讨温度、励磁波形和磁芯材料对磁芯损耗的影响，并通过响应面方法考察三者之间的协同效应。结合线性回归与随机森林算法计算指标信息的重要性，并利用 BFGS 求解器优化，最终确定最小磁芯损耗的最佳方案为 25°C、正弦波和材料 1。这一过程在选择 79400Hz 作为参考频率的前提下进行，以确保训练集中数据的完整性和可靠性。

针对问题四，本文构建了跨材料多工况的磁芯损耗预测模型，重点分析了温度、励磁波形、磁芯材料、频率和磁通密度峰值对损耗的影响，考虑到模型泛化性能，本文重点采

用物理信息嵌入的神经网络模型实现高精度预测，并以全连接神经网络、残差一维卷积网络和多任务学习器网络作为对比模型。以附件三材料数据为测试集，分别验证了以上 4 种模型的预测性能。

针对问题五，本文解决了磁芯损耗最小化与传输磁能最大化的多目标优化问题。首先通过物理信息嵌入的神经网络建立磁芯损耗目标函数，与传输磁能协同构建多目标函数，并结合物理知识构建物理量约束。其次，利用加权法将多目标优化问题转化为单目标优化问题。最后，由于变量中含有离散和连续型变量，因此采用树状结构 TPE 算法计算不同权重因子影响下的最优解和最优值。实验结果发现材料 1 和材料 3 下的高频率、高温、正弦波条件下能够实现多目标问题最优。

综上所述，本文基于数据驱动建模，研究了各工况因素对磁芯损耗的关联性，对传统磁芯损耗模型进行修正，并且创新性地将物理信息嵌入到数据驱动方法中，建立了物理约束神经网络磁芯损耗预测模型，最终确定了不同工作场景需求下的磁芯损耗和传输磁能最优值和相应的最优工作条件。

**关键词：**励磁波形分类 多因素关联度分析 物理约束神经网络 磁芯损耗预测 多目标优化

# 目录

<b>1 问题重述</b>	<b>5</b>
1.1 问题背景	5
1.2 问题分析	7
<b>2 模型假设</b>	<b>8</b>
<b>3 符号说明</b>	<b>8</b>
<b>4 励磁波形分类</b>	<b>9</b>
4.1 利用二阶导特征进行初步筛选以识别正弦波	9
4.2 利用一阶导特征基于 DBSCAN 聚类划分三角波和梯形波	11
4.3 基于导数和聚类的波形分类方法有效性检验与预测	12
<b>5 斯坦麦茨方程修正</b>	<b>14</b>
5.1 探究温度对斯坦麦茨方程的影响	14
5.2 基于实际物理背景采用对数函数拟合斯坦麦茨方程系数	16
5.3 引入温度修正的斯坦麦茨方程拟合效果检验	16
<b>6 磁芯损耗因素分析</b>	<b>18</b>
6.1 数据预处理	18
6.2 基于非参数回归和响应面的单因素和双因素耦合分析	19
6.3 基于多元回归随机森林的最优条件探索	21
6.4 结果分析	22
<b>7 基于数据驱动的磁芯损耗预测模型</b>	<b>23</b>
7.1 数据预处理	23
7.2 基于数据驱动的预测模型	23
7.2.1 全连接神经网络	24
7.2.2 残差一维卷积网络	24
7.2.3 多任务学习器	24
7.2.4 物理约束神经网络	26
7.3 结果分析	28
<b>8 磁性元件的最优化条件</b>	<b>31</b>
8.1 多目标优化问题构建	31

8.2 基于 TPE 的优化问题求解 . . . . .	32
8.3 不同权重下的最优组合解 . . . . .	33
<b>9 结论 . . . . .</b>	<b>35</b>
<b>参考文献 . . . . .</b>	<b>36</b>
<b>附录 A Python 源程序 . . . . .</b>	<b>37</b>
A.1 第 1 问程序 . . . . .	37
A.2 第 2 问程序 . . . . .	39
A.3 第 3 问程序 . . . . .	46
A.4 第 4 问程序 . . . . .	65
A.5 第 5 问程序 . . . . .	84

# 1 问题重述

## 1.1 问题背景

铁磁性材料在现代电力电子技术中扮演着不可或缺的角色。铁磁性材料的深入研究对构建高效、可靠的电能变换系统起到了关键性作用 [1]。此外，随着新兴应用场景，如电动航空、高功率无线充电等的出现，对铁磁性材料在极端条件下的性能提出了更高要求，更加凸显了深入研究铁磁性材料的必要性和紧迫性 [2]。

铁磁性材料能够在交变磁场作用下高效地存储、传递和转换磁能，是实现电能变换的重要介质。随着第三代功率半导体技术的迅速发展，电力电子系统正朝着高频率、高功率密度和高可靠性的方向快速演进。这一趋势对磁性材料提出了更高的挑战：在更高的频率下保持优异的磁性能和较低的损耗 [3]。

然而在实际应用中，铁磁性材料的使用面临着许多复杂的技术挑战。最突出的问题是磁芯损耗，这是磁性材料在高频交变磁通作用下产生的功率损耗。磁芯损耗不仅降低了系统的整体效率，还会导致元件温度升高，影响系统的可靠性和寿命。磁芯损耗与多个因素密切相关，包括工作频率、磁通密度、励磁波形、工作温度以及磁芯材料本身的特性 [4]，这些因素之间存在复杂的非线性关系和相互作用，形成了一个多维度、高度耦合的物理系统，使得准确预测和控制磁芯损耗变得异常困难。此外，高频磁性材料（如铁氧体、合金磁粉芯、非晶/纳米晶等）的微观结构复杂，其损耗机制涉及多尺度的物理过程，从原子级的磁矩排列到宏观的磁畴运动，这进一步增加了建模和分析的难度。

为应对这些问题，研究人员提出了多种磁芯损耗模型和计算方法。主要分为两大类：损耗分离模型和经验计算模型。损耗分离模型试图将总损耗分解为磁滞损耗、涡流损耗和剩余损耗三个部分，分别进行计算后求和。这种方法基于对损耗机制的物理理解，能够提供对损耗构成的深入理解，但在实际应用中常常面临计算复杂和参数确定困难的问题。另一方面，经验计算模型，如著名的斯坦麦茨方程（Steinmetz equation），基于实验数据或理论推导，提供了相对简单的损耗估算方法，但这些模型通常只适用于特定条件（如正弦波励磁），在复杂的实际工作环境中可能产生较大误差。尽管已有一些改进模型，如修正的斯坦麦茨方程（MSE）和改进的广义斯坦麦茨方程（iGSE），试图扩展模型的适用范围，但仍然无法全面准确地描述所有工况下的磁芯损耗行为 [5]。这些模型在特定条件下表现良好，但在面对广泛的材料类型、复杂的励磁波形和变化的环境条件时，其精度和适用性仍存在显著局限。近年来，随着机器学习（ML）算法的快速发展，一些学者开始尝试将ML应用于磁芯损耗预测，这些新方法展现出了处理复杂非线性问题的潜力，但同时也面临着数据质量、模型泛化性和物理解释性等挑战。

因此，开发一个既普遍适用又高度精确的磁芯损耗预测模型成为当前的迫切需求。这不仅是理论上的突破，更是推动电力电子技术实际应用的关键。一个理想的模型应能准确

描述不同材料在各种工作条件下的损耗行为。预测模型不仅需要能够处理复杂的非线性关系，还应具备良好的可解释性和泛化性能。

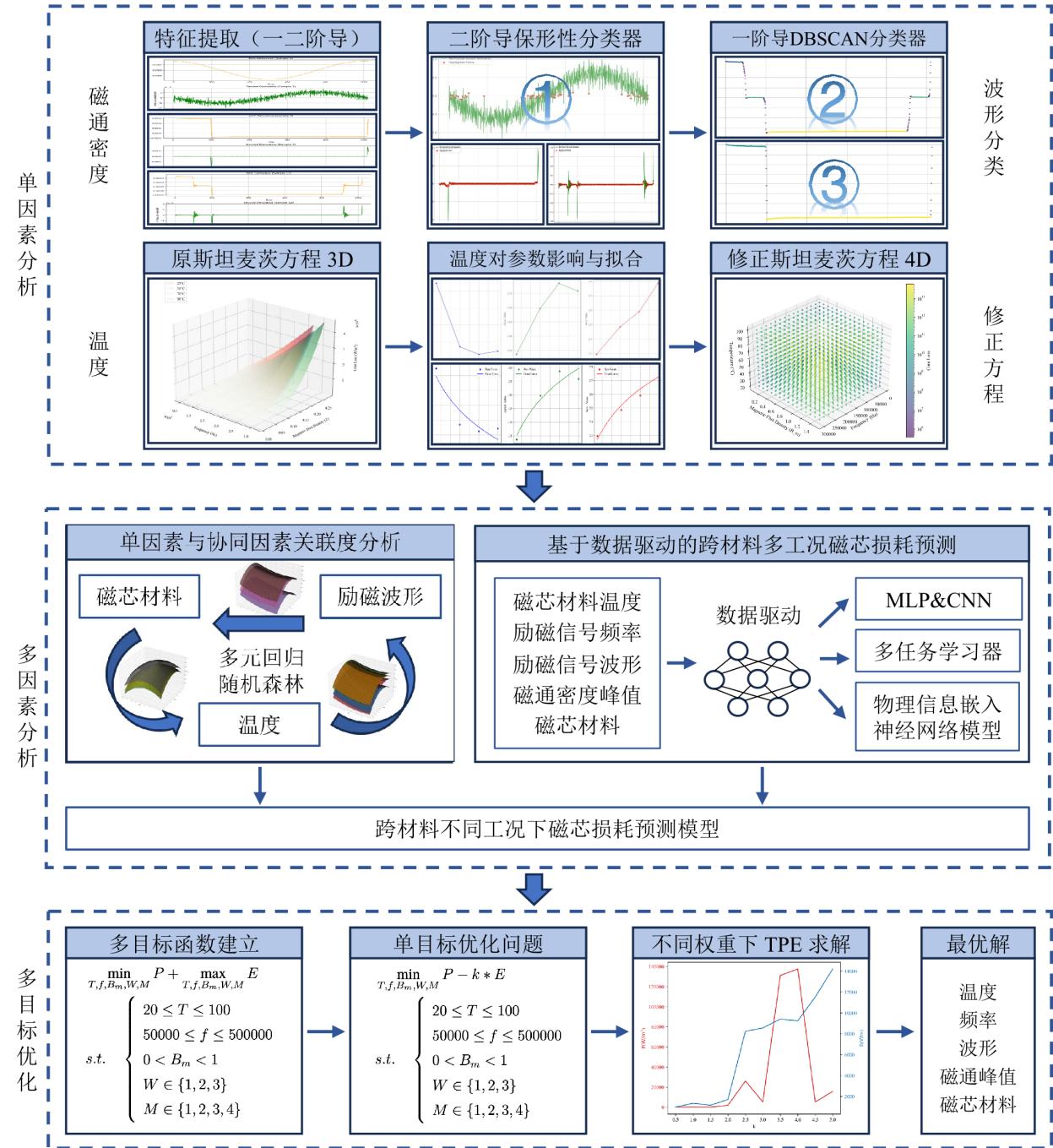


图 1.1 本文研究框架

## 1.2 问题分析

**问题一：**本题旨在对三种励磁波形根据磁通响应数据进行分类。首先跟举正弦波的求导保形性，利用二阶导数数据进行初步筛除正弦波，再利用一阶导数据根据导数曲线平台的个数采用无监督学习的聚类方法进一步划分三角波和梯形波。在分步分类的时候尤需注意正弦波与三角波在导数数据聚类上的混淆问题。

**问题二：**本题旨在对传统的斯坦麦茨方程添加温度影响因素，以提高其在不同温度下的预测能力。将每种温度下的频率、磁通密度峰值作为自变量，将磁芯损耗作为因变量，根据传统的斯坦麦茨方程形式进行拟合。其次结合磁滞损耗和涡流损耗的实际物理背景设计各拟合系数关于温度的拟合方程形式并拟合，得到修正后的斯坦麦茨方程。该拟合思路在较少数据集上较直接添加温度幂次项的你和方法相比更具合理性。

**问题三：**本题旨在磁芯损耗的因素分析，特别是温度、材料类型和励磁波形这三大要素对损耗的耦合影响。本文首先定性确定各单因素和两两因素协同对磁芯损耗的正负影响趋势，其次，为了量化各个因素的影响程度，由于变量中存在离散和连续变量，因此采用树状结构算法计算因素的信息重要性，进而量化影响程度。

**问题四：**本题旨在建立能够跨材料不同工况下的磁芯损耗预测模型。由于磁芯损耗主要受温度、频率、波形、材料、磁通密度峰值的影响，因此仅需要研究以上五种因素和磁芯损耗之间的映射关系，由于训练样本较小，模型泛化能力要求较高，可以采用结合物理信息的神经网络方法求解。

**问题五：**本题旨在求解最小磁芯损耗和最大传输磁能的多目标优化问题。为了最小化磁芯损耗和最大化传输磁能，可以将多目标优化问题转化为单目标优化问题，在转化的过程中需要注意转化因子对两者最优值的影响程度，以满足实际运行过程中的传输磁能和磁芯损耗需求。

## 2 模型假设

1. 本例数据中各波形类别未出现极端情况，如三角波的波形偏度过大、梯形波的平缓区过窄等。
2. 以实际物理现象为支撑，假设原斯坦麦茨方程中温度对各系数的影响为对数形式；
3. 假设不同频率下的使磁芯损耗最小的温度、励磁波形、磁芯材料最优组合近似；
4. 假设磁芯材料主要受温度、励磁频率、励磁波形、磁芯材料、磁通密度峰值的影响，而没有其他外来干扰。

## 3 符号说明

符号	意义	量纲
$P_{\text{loss}}$	磁芯损耗	$\text{W/m}^3$
$f$	频率	Hz
$B_m$	磁通密度峰值	T
$T$	温度	$^{\circ}\text{C}$
$k_1, \alpha_1, \beta_1$	原斯坦麦茨方程拟合系数	-
$k_2, k_3$	$k_1$ 关于温度的拟合系数	-
$\alpha_2, \alpha_3$	$\alpha_1$ 关于温度的拟合系数	-
$\beta_2, \beta_3$	$\beta_1$ 关于温度的拟合系数	-

## 4 励磁波形分类

本节主要介绍基于导数特征和聚类算法的励磁波形分类方法，以区分正弦波、三角波和梯形波。首先，由于不同波形的导数特征存在显著差异，可以利用这些特征进行初步分类。因此，对原始波形数据进行一阶和二阶导数计算，为后续分类提供基础。其次，本节介绍了两种不同的分类步骤：利用二阶导特征识别正弦波，以及基于 DBSCAN 聚类算法区分三角波和梯形波。最后，利用已标注的训练数据验证模型准确性，并对未标注的测试样本进行分类预测，展示了该方法的有效性和实用性。

### 4.1 利用二阶导特征进行初步筛选以识别正弦波

首先对波形数据进行二阶导数计算，每种波形的处理结果分别如图4.2，图4.3，图4.4所示。

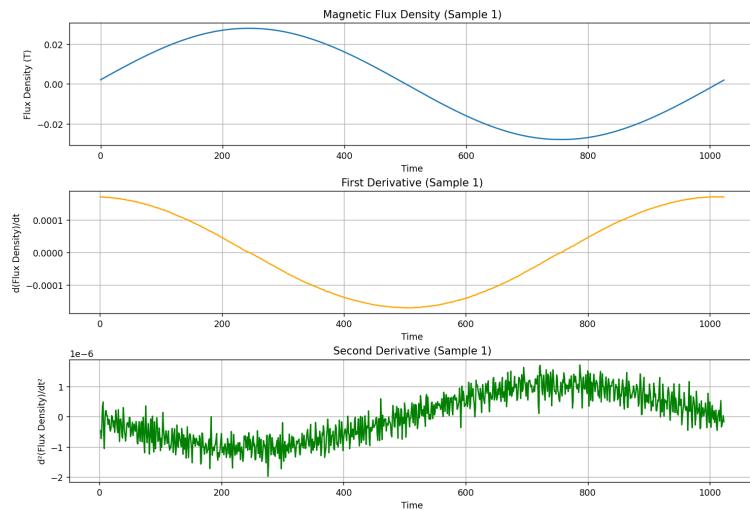


图 4.2 正弦波一阶导和二阶导

由于三角波和梯形波的二阶导绝大多数数据位于零点附近，而由于正弦波独有的特性，其二阶导整体仍呈现出正弦型。基于这一特性，对导数数据进行归一化处理，并统计位于区间  $(-0.1, 0.1)$  内的点数，如图4.5，图4.6，图4.7所示。设定阈值为采样点数的一半（约为 500）：小于该阈值的值判定为正弦波，反之则为三角波或梯形波。

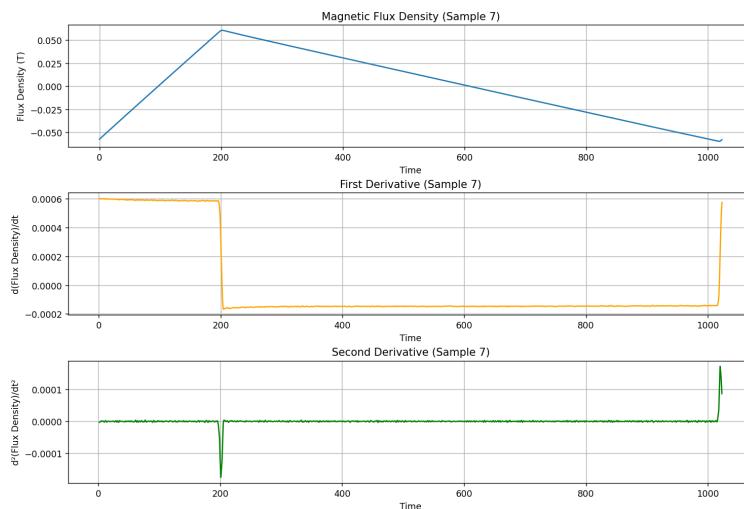


图 4.3 三角波一阶导和二阶导

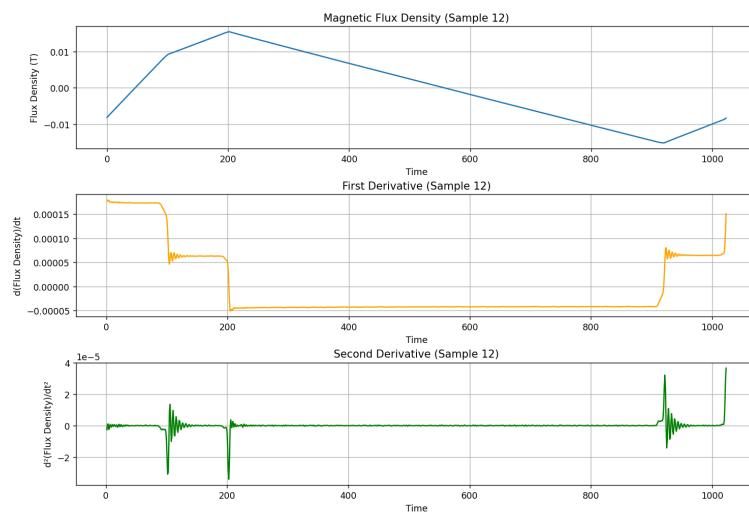


图 4.4 梯形波一阶导和二阶导

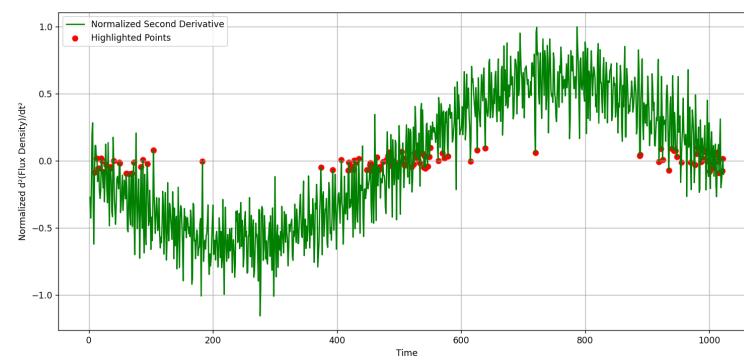


图 4.5 归一化后正弦波二阶导数据接近零值的个数统计

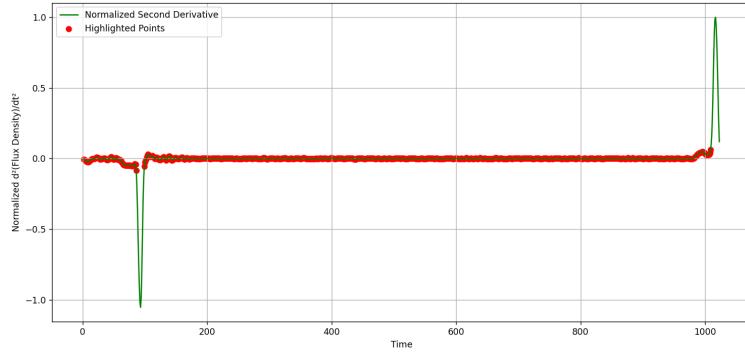


图 4.6 归一化后三角波二阶导数据接近零值的个数统计

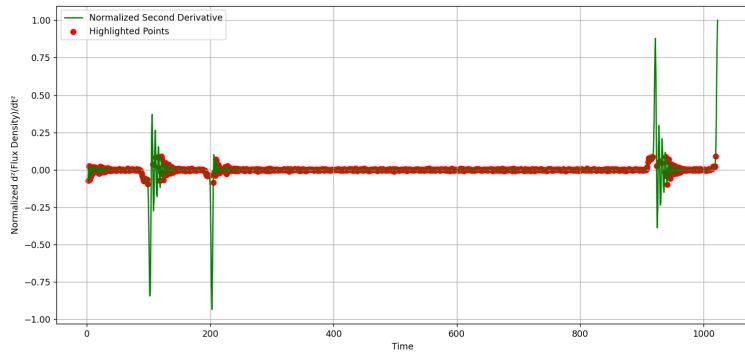


图 4.7 归一化后梯形波二阶导数据接近零值的个数统计

## 4.2 利用一阶导特征基于 DBSCAN 聚类划分三角波和梯形波

为实现有效分类，采用 DBSCAN 聚类算法 [6]，根据导数图像对聚类器的两个参数  $\epsilon$  和  $min_{samples}$  进行调整(设置为 0.00002 和 20)，以决定簇的形成。聚类结果显示，紫色部分被标记为噪声，其他颜色代表识别出的不同簇，如图4.8，图4.9所示。

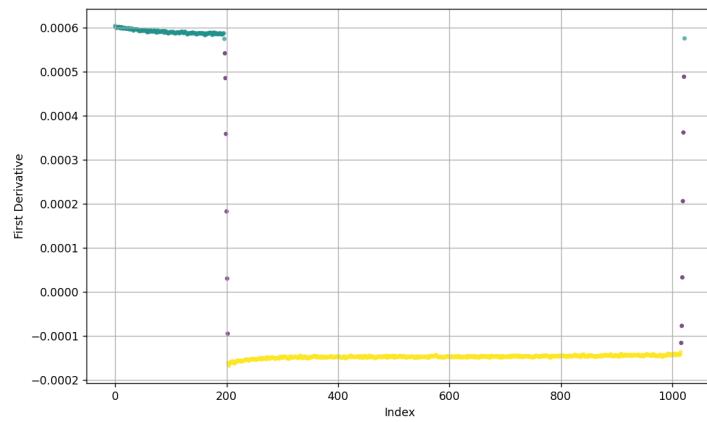


图 4.8 三角波聚类结果

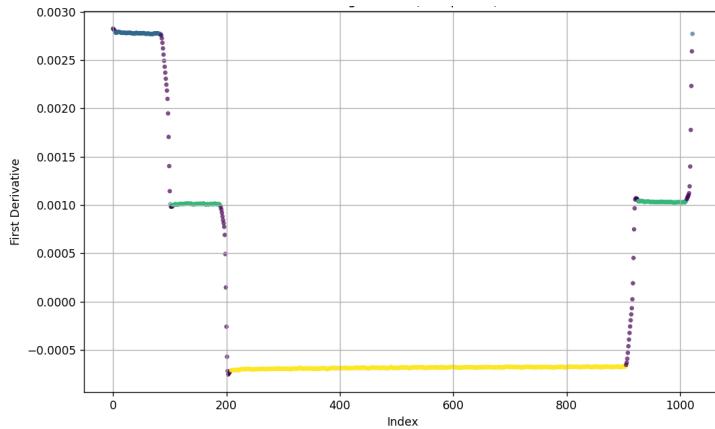


图 4.9 梯形波聚类结果

三角波和梯形波的导数由于具备明显的阶梯状分布，导致聚类效果良好。尽管正弦波的导数在一个周期内包含两个平滑段<sup>4.10</sup>，容易与三角波的两个平台混淆，但在前述步骤中已利用二阶导将其筛除，故不影响分类判断。

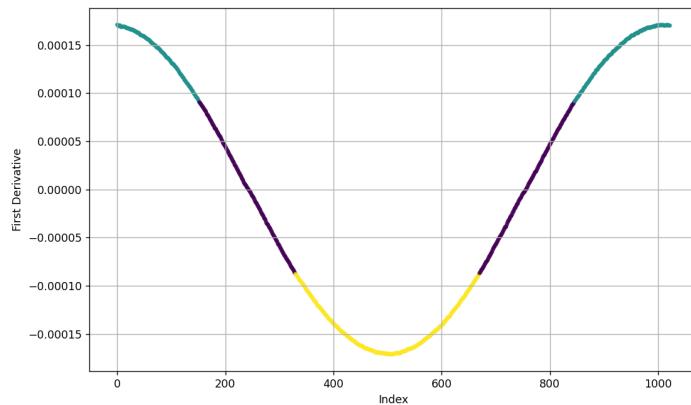


图 4.10 正弦波聚类结果

### 4.3 基于导数和聚类的波形分类方法有效性检验与预测

利用前述模型，根据波形的一阶导和二阶导数据和聚类算法，可以作为分辨正弦波、三角波和梯形波的方法。附件一中包含大量已标注波形类别的数据，基于此对上述提出的方法检验。经检验，该预测模型在训练集中的准确率为 100%，证明该分类方法的高效性，甚至胜过机器学习的方法。附件二中包含了 80 例未标注激励波形的测试数据，三种波形的各自数量为正弦波 19 例，三角波 45 例，梯形波 16 例，分布如图4.11所示，其中 1,2,3 分别表示正弦、三角、梯形波。部分具体分类结果展示如表4.1所示（完整分类结果见附件四）。

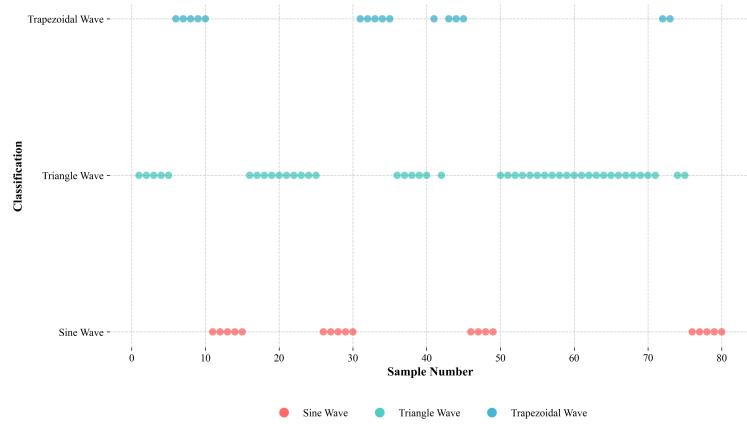


图 4.11 附件二波形分类结果

表 4.1 附件四中部分数据分类结果

波形编号	分类结果	对应编码
1	三角波	2
5	三角波	2
15	正弦波	1
25	三角波	2
35	梯形波	3
45	梯形波	3
55	三角波	2
65	三角波	2
75	三角波	2
80	正弦波	1

## 5 斯坦麦茨方程修正

本节主要介绍对斯坦麦茨方程进行修正以提高其在不同温度下的磁芯损耗预测精度。首先，通过分析不同温度下的斯坦麦茨方程拟合结果，发现温度对方程系数有显著影响，需要考虑温度因素。其次，基于磁滞损耗和涡流损耗的物理背景，采用对数函数对斯坦麦茨方程的系数进行温度相关的修正，提出了改进的斯坦麦茨方程模型。最后，通过对比修正前后的拟合效果，验证了改进模型的有效性。

### 5.1 探究温度对斯坦麦茨方程的影响

分别将每种温度 ( $25^{\circ}\text{C}$ 、 $50^{\circ}\text{C}$ 、 $70^{\circ}\text{C}$ 、 $90^{\circ}\text{C}$ ) 下的频率  $f$ 、磁通密度峰值  $B_m$  作为自变量，将磁芯损耗  $P_{\text{loss}}$  作为因变量，根据传统的斯坦麦茨方程形式5.1进行拟合，见图5.12和式5.2。

$$P_{\text{loss}} = k_1 \cdot f^{\alpha_1} \cdot B_m^{\beta_1} \quad (5.1)$$

其中： $k_1$ 、 $\alpha_1$ 、 $\beta_1$  为拟合系数， $1 < \alpha_1 < 3$ ， $2 < \beta_1 < 3$ 。

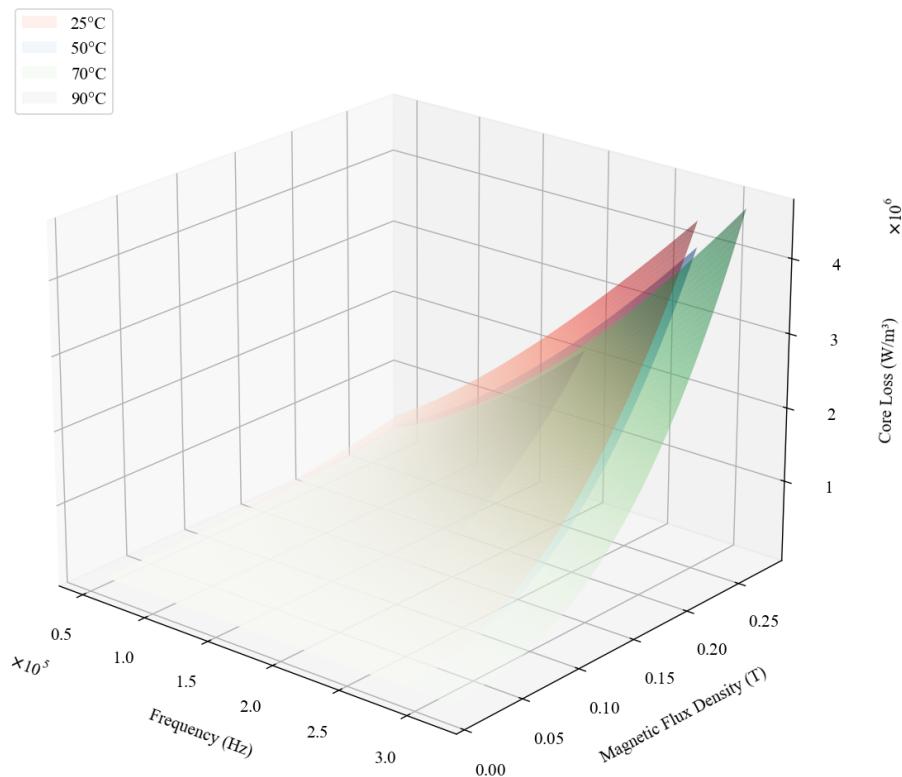


图 5.12 不同温度下的斯坦麦茨方程

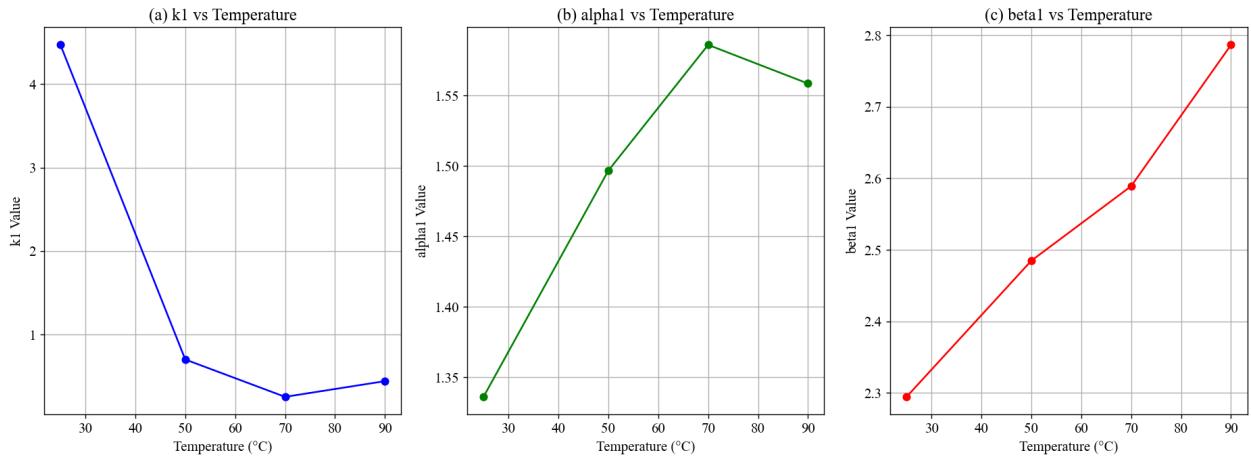


图 5.13 各拟合系数关于温度的函数

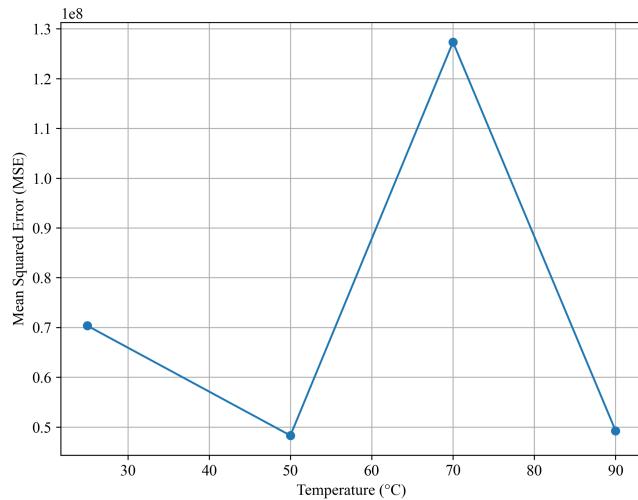


图 5.14 不同温度下拟合 MSE

$$P_{\text{loss}} = \begin{cases} 4.4699 \cdot f^{1.3361} \cdot B_m^{2.2943}, & \text{for } 25^\circ C \\ 0.7021 \cdot f^{1.4966} \cdot B_m^{2.4848}, & \text{for } 50^\circ C \\ 0.2546 \cdot f^{1.5859} \cdot B_m^{2.5891}, & \text{for } 70^\circ C \\ 0.4427 \cdot f^{1.5585} \cdot B_m^{2.7867}, & \text{for } 90^\circ C \end{cases} \quad (5.2)$$

得出结论：各拟合系数均为温度的函数5.3，因此不能忽略温度对预测结果的影响。

$$P_{\text{loss}} = k_1(T) \cdot f^{\alpha_1(T)} \cdot B_m^{\beta_1(T)} \quad (5.3)$$

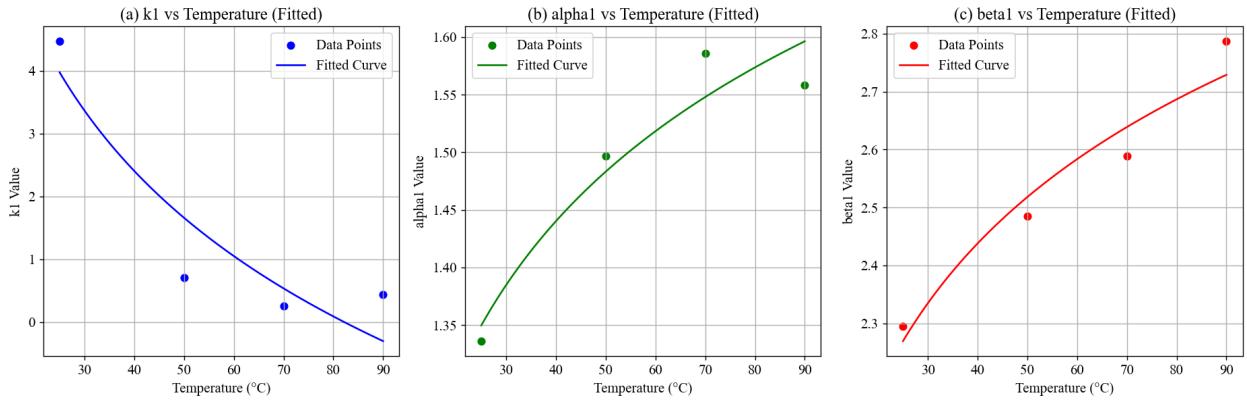


图 5.15 各系数关于温度拟合曲线

## 5.2 基于实际物理背景采用对数函数拟合斯坦麦茨方程系数

本实验各拟合系数关于温度的函数的关系符合磁滞损耗和涡流损耗的物理背景，因此具有很强的理论支持。根据磁滞损耗的特性 [7] 可知，温度升高会导致磁芯材料的磁导率（即材料的磁性能）下降，使得磁畴的翻转更为困难，高频磁场的作用将导致更多的磁滞损耗；根据涡流损耗的特性 [8] 可知，涡流损耗与频率的平方成正比。温度升高会导致材料电阻率增大，理论上电阻率的增加会减少涡流损耗，但频率增加的影响通常更大。随着温度上升，材料内的微观结构可能发生变化（如晶格畸变），这种变化使得材料内部电导率的不均匀性增加，从而放大了高频下的涡流效应。

基于以上探究，采用对数函数5.4将各拟合系数关于温度进行拟合5.15，得到修正后的斯坦麦茨方程，见图5.16和式5.5。

$$\begin{aligned} k_1 &= k_2 \cdot \log T + k_3 \\ \alpha_1 &= \alpha_2 \cdot \log T + \alpha_3 \\ \beta_1 &= \beta_2 \cdot \log T + \beta_3 \end{aligned} \quad (5.4)$$

$$P_{\text{loss}} = (0.7021 \cdot \log(T) + 0.4427) \cdot f^{(0.2498 \cdot \log(T) + 1.176)} \cdot B_m^{(0.1241 \cdot \log(T) + 2.189)} \quad (5.5)$$

## 5.3 引入温度修正的斯坦麦茨方程拟合效果检验

通过对比原始斯坦麦茨方程与引入温度修正后的拟合结果5.17，发现引入温度修正项后，MSE 减小至修正前的 7.19%，与不同温度下拟合 MSE 结果5.18在同一量级（图中橘黄色虚线为修正后的 MSE），这表明改进的方法符合预期，模型的预测精度得到了显著提升。

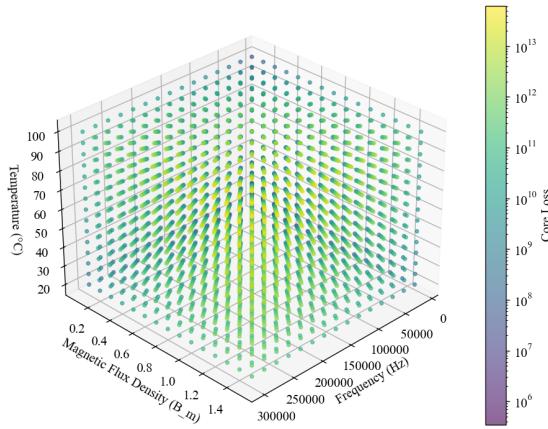


图 5.16 修正后的斯坦麦茨方程图

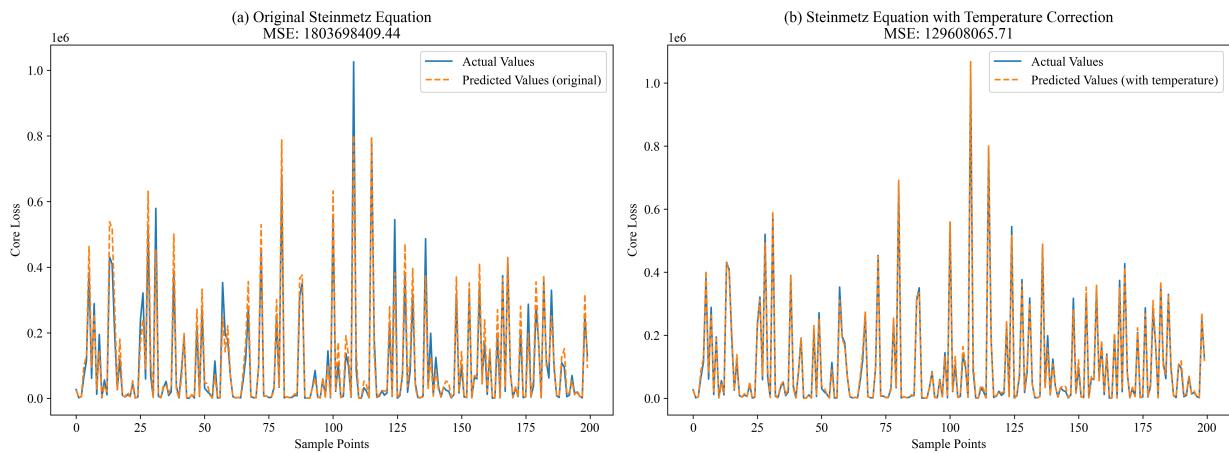


图 5.17 修正前后对比

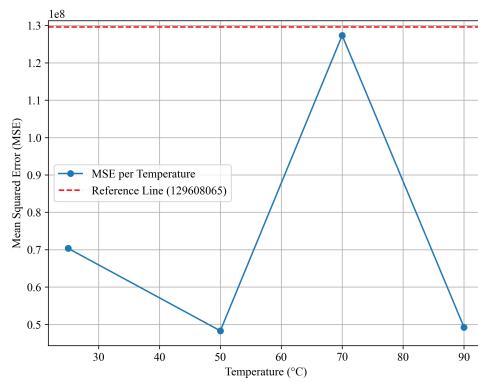


图 5.18 修正方程的 MSE 与原方程在不同温度下的 MSE 对比

## 6 磁芯损耗因素分析

### 6.1 数据预处理

本节旨在探究影响磁芯损耗的三种重要因素分别及两两协同的影响程度并确定这三个因素使得磁芯损耗最小的最优条件，首先根据式5.3，磁芯损耗与频率呈正比关系，频率越低，损耗越小。因此，我们从低到高选择了一个代表性的参考频率区间进行分析。因79400Hz附近的频率条件下，三个变量的所有类别均出现，因此选做研究对象。

首先对实验数据进行预处理，以观察在参考低频下温度、材料类型和励磁波形对磁芯损耗的影响。箱形图用于显示数据分布的概况，包括中位数、四分位数以及异常值，有助于快速识别数据的集中趋势和变异程度。它可以直观地比较不同组之间的分布差异。为了直观展示数据的分布，可以通过图6.19展示了三个因素对磁芯损耗的影响效果。即温度为90°C时磁芯损耗整体很低，励磁波形为正弦波时，磁芯损耗最低且很少出现磁芯损耗较高的情况。材料1和材料4表现都较好，但是材料1容易出现磁芯损耗较高的情况。

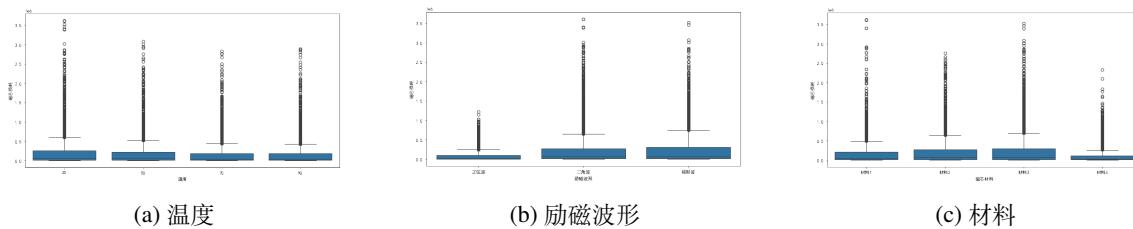


图 6.19 参考低频下三种因素与磁芯损耗箱形图

将两两因素绘制在坐标轴上可以通过三维散点图6.20定性分析励磁波形-温度，励磁波形-材料，材料-温度的三因素两两之间对磁芯损耗的协同影响。即通过分布可以看出温度与励磁波形对磁芯损耗的影响较小，材料与励磁波形的协同影响较为显著，温度与材料的协同作用对磁芯损耗影响小。

本节最后为了探寻在离散变量材料、励磁波形及连续变量温度共同作用下达到磁芯损耗最小的最优条件，需要对非这三个变量影响确定的不同值进行平均处理以简化分析，并且这步处理对控制磁通密度峰值有参考作用，最终得到了48个具有唯一性的点，以便于全局寻优。

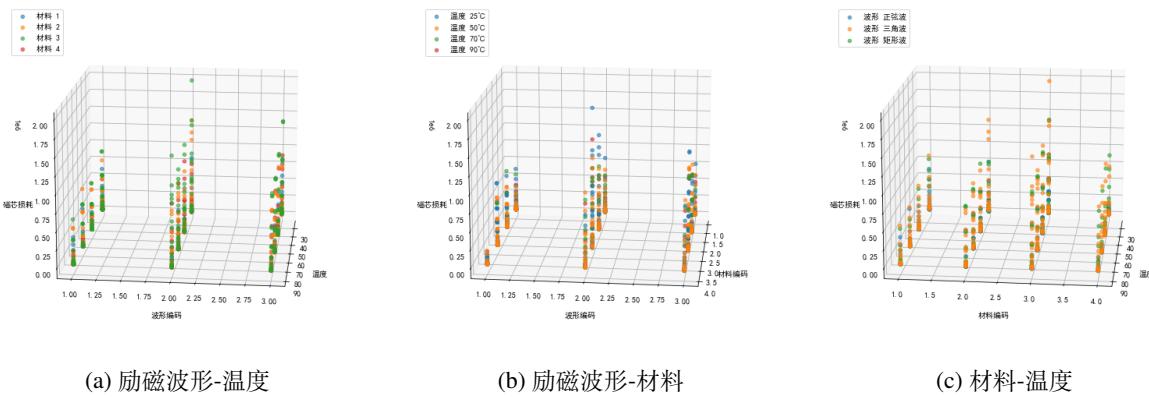


图 6.20 三种变量对磁芯损耗散点图

## 6.2 基于非参数回归和响应面的单因素和双因素耦合分析

本节在对单因素与磁芯损耗的相关性进行分析时，采用了参数回归方法进行定性分析。利用式6.6回归模型：

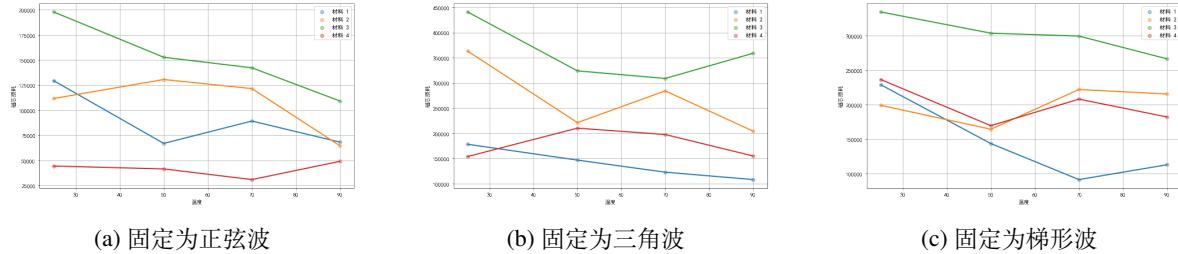


图 6.21 不同励磁波形下的温度对磁芯损耗的非参数回归

通过图6.21，温度对磁芯损耗的影响较小，但是不确定是正相关还是负相关，总体上温度上升磁芯损耗会降低。

通过图6.22，励磁波形对磁芯损耗的影响大，总体上正弦波对磁芯损耗的降低作用优于三角波和梯形波。

通过图6.23，材料类型对磁芯损耗的影响较大，总体上材料 1 和材料 4 对减少磁芯损耗效果优于其他材料。

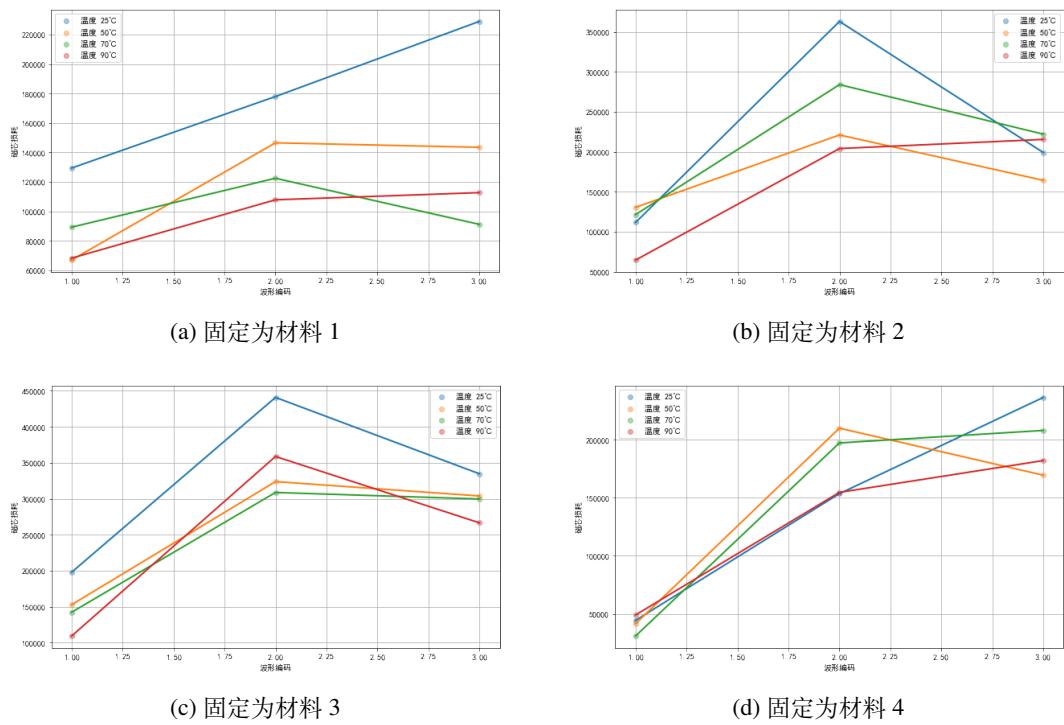


图 6.22 不同材料下的励磁波形对磁芯损耗的非参数回归

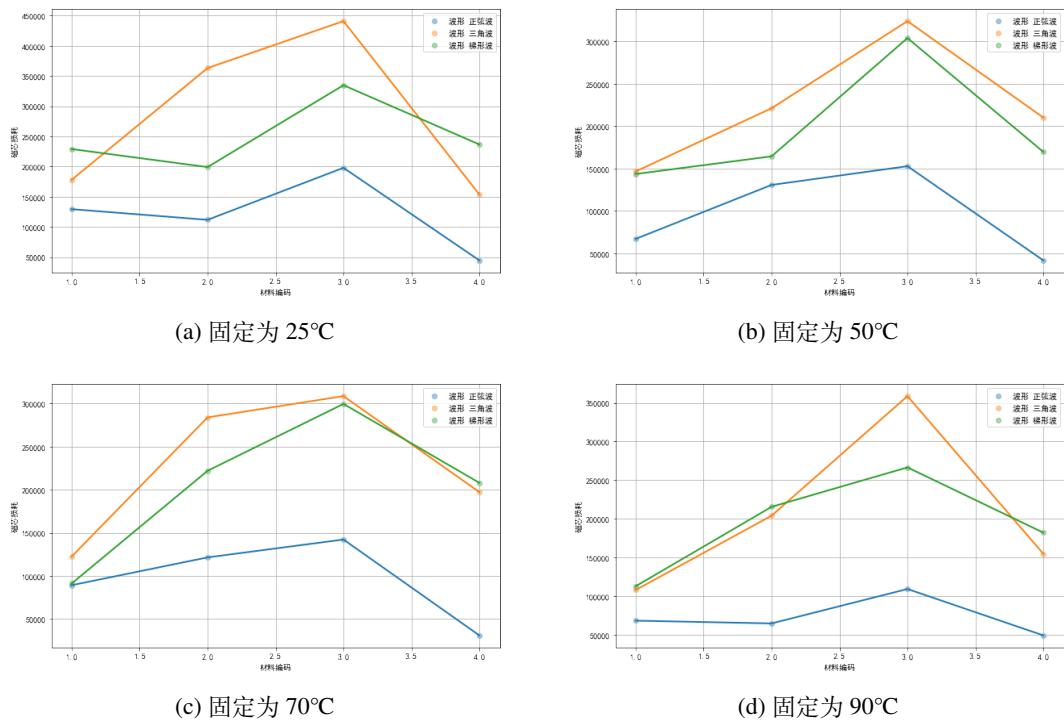


图 6.23 不同温度下材料类型对磁芯损耗的非参数回归

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon \quad (6.6)$$

其中， $y$  表示磁芯损耗， $x_1$ 、 $x_2$ 、 $x_3$  分别表示材料、温度和励磁波形，而  $\epsilon$  是误差项。利用式6.6中的公式得到的回归系数揭示了各个因素的独立影响。此外，对于三个因素之间的两两耦合影响，绘制了图6.24响应面，以根据其曲率定性描述温度与材料类型、温度与励磁波形、材料类型与励磁波形的交互作用。通过分析响应面，看出温度与励磁波形响应面以及温度与材料响应面沿温度轴的曲率变化较小，说明温度与另外两个指标的耦合关系并不明显，而材料与励磁波形的共同影响显著，沿响应面两个方向的曲率变化较大。

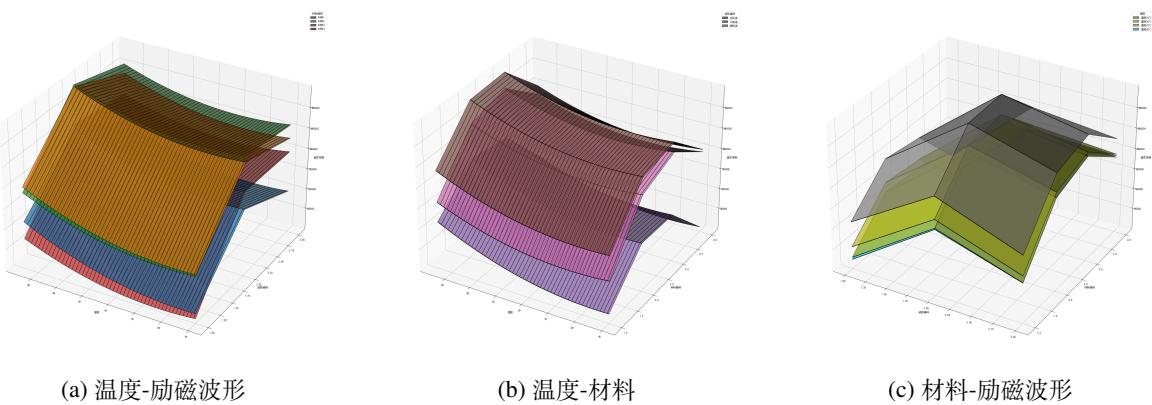


图 6.24 三因素两两协同对磁芯损耗的响应面

### 6.3 基于多元回归随机森林的最优条件探索

为了进一步定量分析各因素之间的复杂关系，本节采用了随机森林算法 [9] 与多元回归相结合的方式。随机森林模型的构建通过式6.7进行特征重要性评估：

$$I_j = \frac{1}{N} \sum_{t=1}^T \left( \frac{(y - \hat{y}_t)^2 - (y - \hat{y}_t^{(j)})^2}{\sigma^2} \right) \quad (6.7)$$

其中， $I_j$  表示特征  $j$  的重要性， $N$  是样本总数， $T$  是树的数量， $y$  是真实值， $\hat{y}_t$  是随机森林模型的预测值，而  $\hat{y}_t^{(j)}$  是特征  $j$  被打乱后的预测值， $\sigma^2$  是样本方差。

利用式6.7中的公式得到的特征重要性表6.2揭示了励磁波形对磁芯损耗的显著影响（0.4680），同时材料与温度的交互项（0.2079）也显示了重要性。

BFGS 算法，见式6.8是一种高效的迭代优化算法，它利用拟牛顿方法6.9来更新 Hessian 矩阵的近似值。在第  $k$  次迭代中，解的更新公式为：

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k) \quad (6.8)$$

其中,  $\mathbf{x}_k$  是第  $k$  次迭代的解,  $\alpha_k$  是步长 (学习率),  $\mathbf{H}_k$  是在第  $k$  次迭代计算的 Hessian 矩阵的近似值, 而  $\nabla f(\mathbf{x}_k)$  是目标函数在  $\mathbf{x}_k$  处的梯度。

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{H}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{H}_k}{\mathbf{s}_k^T \mathbf{H}_k \mathbf{s}_k} \quad (6.9)$$

其中,  $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$  表示变化的变量, 而  $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$  表示梯度的变化。

表 6.2 特征系数与特征重要性

特征	系数	重要性
材料	-16532.6157	-0.1748
温度	-1048.4859	-0.0585
励磁波形	2221.4885	0.4680
材料-温度	618.0173	0.2079
材料-励磁波形	-14360.0436	-0.0524
温度-励磁波形	-393.8105	-0.0384

## 6.4 结果分析

本节用离散网格搜索在参考低频相应面模型上求得的结果是材料 4,70°C, 正弦波情况下磁芯损耗最小。

利用 BFGS 寻优随机森林得到的影响因子求得的结果是材料 1,25°C, 正弦波情况下磁芯损耗可能取到最小。

对此现象的合理解释是, 即使温度升高往往代表着耗能增大, 但材料 4 中的特殊结构使得其是一种在高温上表现良好的低耗能材料。另外对于参考低频响应面的分析可能陷入局部最优, 但是整体趋势正确。温度, 材料以及励磁波形这三个因素之间的相关性不简单是线性关系, 因此不能通过斯皮尔曼相关系数等直接进行单因素分析, 本节通过多元回归与随机森林结合的方式求得的最优解综合评价了三种因素及两两协同对磁芯损耗的复杂非线性影响, 具有参考价值。

## 7 基于数据驱动的磁芯损耗预测模型

本章节主要介绍基于数据驱动方法建立跨材料不同工况下的磁芯损耗预测模型。首先，由于不同的输入数据具有不同的量纲和单位，导致数据之间的差异性较大，影响数据驱动模型对不同特征的抓取能力，因此需要先对原始数据进行预处理，从而提高模型的性能和稳定性。其次，本小节介绍了 4 中不同的局域数据驱动方法的磁芯损耗预测模型，分别为全连接神经网络 [10]、残差卷积神经网络 [11]、多任务学习器和物理约束的深度神经网络 [12]。最后，利用以上四个模型对测试集中的不同样本的磁芯损耗进行预测。

### 7.1 数据预处理

磁芯材料的损耗与诸多因素相关，其中主要包括温度、励磁频率、励磁波形、磁芯材料和磁通密度峰值。其中励磁波形和磁芯材料为离散变量，温度、励磁频率和磁通密度峰值为连续变量。

针对离散型变量，直接将类别编码为整数可能会引入不存在的顺序关系，而 one-hot 编码会将上述类别编码转化为二进制值，因此不会使模型误解分类数据之间的关系。因此，本文采用 one-hot 编码方法，将励磁波形和磁芯材料变量转换为模型可输入变量。

由于各变量之间的量纲不同，导致数值差异明显，不利于作为基于数据驱动的模型的输入变量，因此需要对以上 5 种变量进行归一化处理，本文采用的是 Min-Max 归一化方法，具体公式描述如式 7.10。

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \quad (7.10)$$

其中  $X_{\min}$  和  $X_{\max}$  分别是特征的最小值和最大值。

基于以上数据处理，可以使模型正确地提取相应变量之间的特征。然而，选择一个合适的预测模型更为重要。

### 7.2 基于数据驱动的预测模型

由于铁磁损耗和温度、励磁频率、励磁波形、磁芯材料和磁通密度峰值相关，因此可以将这种关系转化为函数映射关系，其数学表达式如式 7.11。

$$P = f(T, f, W, M, B_m) \quad (7.11)$$

其中  $P$  为磁芯损耗， $T, f, W, M, B_m$  分别是温度、励磁频率、励磁波形、磁芯材料和磁通密度峰值。目前有大量基于数据驱动的方法可以用于函数关系映射，其中包括全连接神经网络、卷积神经网络等。本文在上述基本模型的基础上，分别建立了全连接神经网络、残差一维卷积网络、多任务学习器以及物理约束神经网络模型，不仅在现有的网络结构上进行改进，而且创新地将先验物理信息与数据驱动的方法结合起来，使得模型的泛化性能得到进一步的提升。

### 7.2.1 全连接神经网络

全连接神经网络（Fully Connected Neural Network, FCNN）是一种基础的深度学习模型，由输入层、隐藏层和输出层组成，每个神经元与前一层的所有神经元相连。它的主要作用是通过学习输入数据的复杂特征来进行分类和回归任务。其优点包括结构简单、易于实现、具有强大的表示能力和自适应学习能力，能够处理复杂的非线性数据分布。FCNN 的数学表达式如式7.12所示。

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}a^{(l)} = f(z^{(l)}) \quad (7.12)$$

其中： $z^{(l)}$  是第  $l$  层的加权输入， $W^{(l)}$  是第  $l$  层的权重矩阵， $a^{(l-1)}$  是第  $l-1$  层的激活输出， $b^{(l)}$  是第  $l$  层的偏置向量， $f$  是 ReLU 激活函数。

本文采用的 FCNN 的模型架构如表7.3所示。

### 7.2.2 残差一维卷积网络

一维卷积网络（1D Convolutional Neural Network, 1D-CNN）是一种专门用于处理序列数据的神经网络结构。与二维卷积网络不同，1D-CNN 只在一个维度上进行卷积操作，通常是序列长度维度。其主要作用是通过卷积核在输入序列上滑动，提取局部特征，从而捕捉数据中的模式和趋势。1D-CNN 能够有效处理长序列数据，减少参数数量，提高计算效率，并且通过共享卷积核权重，能够捕捉到序列中的局部依赖关系。此外，1D-CNN 还具有较强的平移不变性，即对输入数据的平移具有鲁棒性。1D-CNN 的数学表达式如式7.13所示。

$$y(t) = (x * w)(t) = \sum_{k=0}^{K-1} x(t+k) \cdot w(k) \quad (7.13)$$

其中， $y(t)$  是输出， $x(t)$  是输入序列， $w(k)$  是卷积核， $K$  是卷积核的长度。

残差网络（Residual Network, ResNet）核心思想是通过引入残差连接来解决深层网络训练中的退化问题。随着网络层数的增加，传统深度神经网络容易出现梯度消失或梯度爆炸的问题，导致训练困难和性能下降。残差网络通过在每个残差块中添加快捷连接，使得输入可以直接跳过若干层传递到输出，能够训练更深的网络，从而提高模型的表达能力。具体来说，残差块的输出是输入和通过若干层计算得到的变换之和，即  $H(x) = F(x) + x$ ，其中  $F(x)$  是残差函数， $x$  是输入。

因此，本文提出了基于残差网络架构的一维卷积网络模型，即残差一维卷积神经网络（Res-1D-CNN）模型。该模型的框架如图7.25所示。

### 7.2.3 多任务学习器

多任务学习（Multi-Task Learning, MTL）通过同时训练多个相关任务来提高模型的性能和泛化能力。其基本思想是利用不同任务之间的共享信息和特征，从而使模型能够更好

表 7.3 FCNN 模型架构

模型层	输入数据维度	输出数据维度
Linear	5	64
ReLU	64	64
Linear	64	256
ReLU	256	256
Linear	256	512
ReLU	512	512
Linear	512	1024
ReLU	1024	1024
Linear	1024	512
ReLU	512	512
Linear	512	256
ReLU	256	256
Linear	256	64
ReLU	64	64
Linear	64	1

地理解和学习数据的内在结构。多任务学习通过共享底层特征，多个任务可以相互补充，减少训练时间和计算资源。其次由于模型在多个任务上进行训练，它能够更好地适应新任务和未见数据，减少过拟合的风险，增强泛化能力。

本文将所有磁芯材料的数据经过 MTL 的共享层，首先提取数据共同特征。其次，将磁芯材料类型作为判断标签，根据标签将 4 种不同磁芯材料的数据作为 4 个子任务器的训练数据集，这样能够使网络模型更好地泛化到不同的磁芯材料中。本文采用的 MLT 框架如表7.4所示。

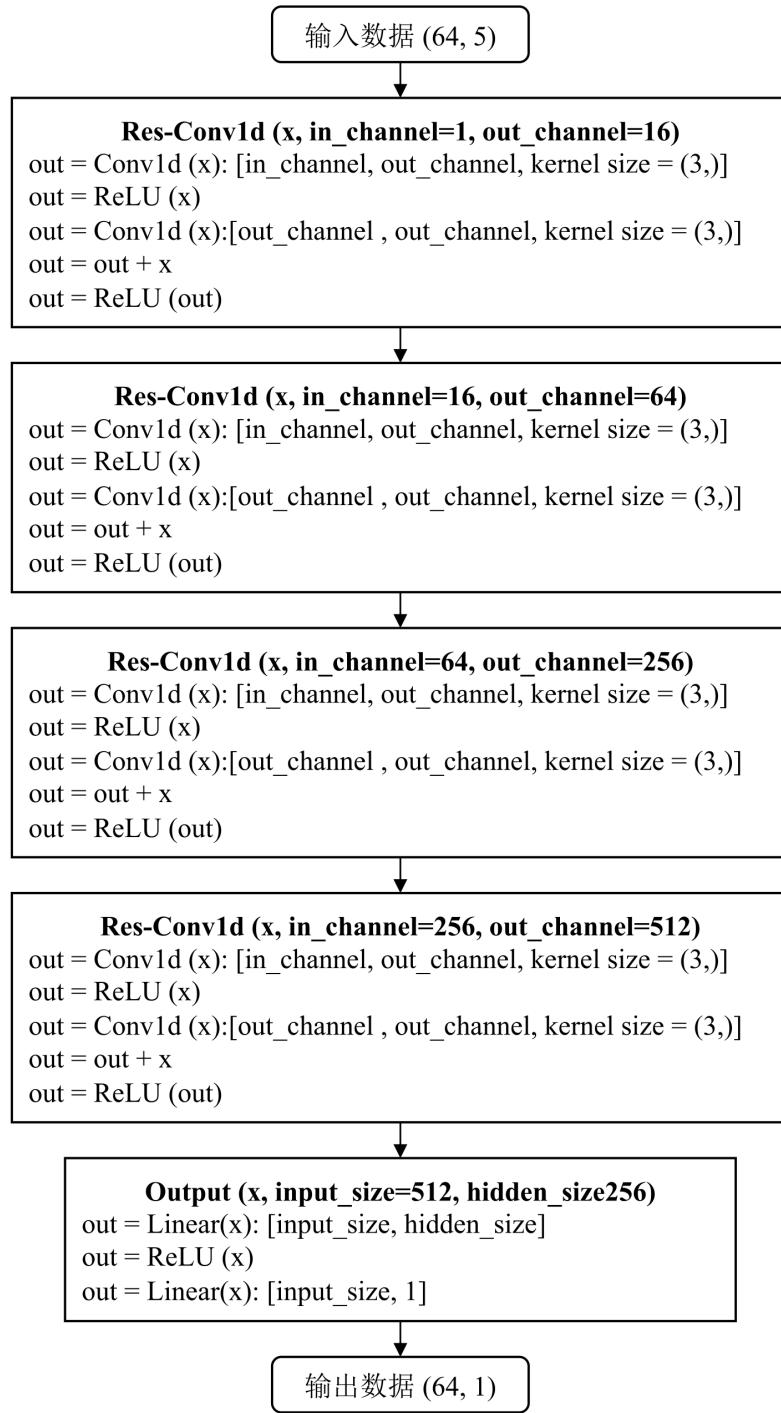


图 7.25 Res-1D-CNN 模型架构

#### 7.2.4 物理约束神经网络

物理约束神经网络（Physics-constrained Neural Network, PCNN）是一种结合了深度学习和物理学知识的机器学习模型。与传统的数据驱动神经网络不同，PCNN 在学习过程中利用物理法则对模型进行指导，从而提高模型的泛化能力，特别是在数据较少或噪声较大

表 7.4 MTL 模型架构

学习器	模型层	输入数据维度	输出数据维度
共享层	Linear	5	64
共享层	ReLU	64	64
共享层	Linear	64	256
共享层	ReLU	256	256
子任务	Linear	256	512
子任务	ReLU	512	512
子任务	Linear	512	1024
子任务	ReLU	1024	1024
子任务	Linear	1024	512
子任务	ReLU	512	512
子任务	Linear	512	256
子任务	ReLU	256	256
子任务	Linear	256	64
子任务	ReLU	64	64
子任务	Linear	64	1

的情况下。

PCNN 的工作原理是将物理知识引入损失函数中。具体来说，PCNN 模型通常由一个深度神经网络构成，其损失函数包含两部分：数据误差项和物理信息误差项。数据误差项用于衡量网络预测输出与实际观测数据之间的差异，而物理信息误差项则考量网络预测结果是否满足物理定律。由于引入了物理约束，PCNN 在数据稀缺或噪声较多的情况下仍能进行有效的训练和预测。其次，物理约束提供了额外的指导信息，使得 PCNN 对数据质量和数量的依赖相对较小。最后，通过结合物理法则，PCNN 模型的预测结果更具解释性，能够更好地反映实际物理现象。

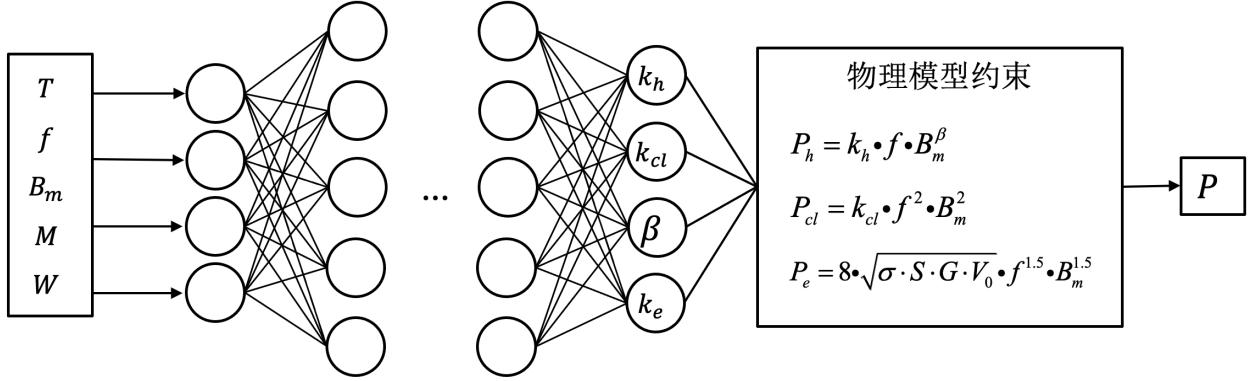


图 7.26 PCNN 模型架构示意图

本文基于磁芯损耗的物理机制，将总损耗分解为三个独立的成分：磁滞损耗，涡流损耗，剩余损耗。对应的数学表达如式7.14, 7.15, 7.16所示

$$P_h = k_h * f * B_m^\beta \quad (7.14)$$

$$P_{cl} = k_{cl} * f^2 * B_m^2 \quad (7.15)$$

$$P_e = k_e * f^{1.5} * B_m^{1.5} \quad (7.16)$$

其中， $k_h, k_{cl}, k_e, \beta$  与工况和材料相关， $f$  和  $B_m$  分别为励磁频率和磁通密度峰值。

因此，本文将  $k_h, k_{cl}, k_e, \beta$  设为 PCNN 模型内部输出的隐藏变量，将隐藏变量送入物理模型中得到输出，将该输出与实际输出的差作为网络模型的反馈值进行参数修正。由于以上四个参数在物理意义上必须为正，因此本文采用式7.17计算实际参数值。

$$x = x_0 * \exp^y \quad (7.17)$$

其中  $x$  为参数值， $x_0$  为参数基准值， $y$  为神经网络的隐藏输出变量。

PCNN 的模型框架如图7.26所示。

本文将采用以上 4 种数据驱动模型预测磁芯损耗，并且模型训练超参数如表7.5所示。

### 7.3 结果分析

本文利用附件三中的不同磁芯材料数据作为测试集，预测不同工况下的磁芯损耗，预测结果如图7.27所示。

将以上预测的特定 10 组样本的磁芯损耗预测值作为参考基准，如表7.6所示。

表 7.5 模型训练超参数

参数	设定值
Learning Rate	2e-3
Step Size	20
Gamma	0.5
Epochs	150
Batch Size	64

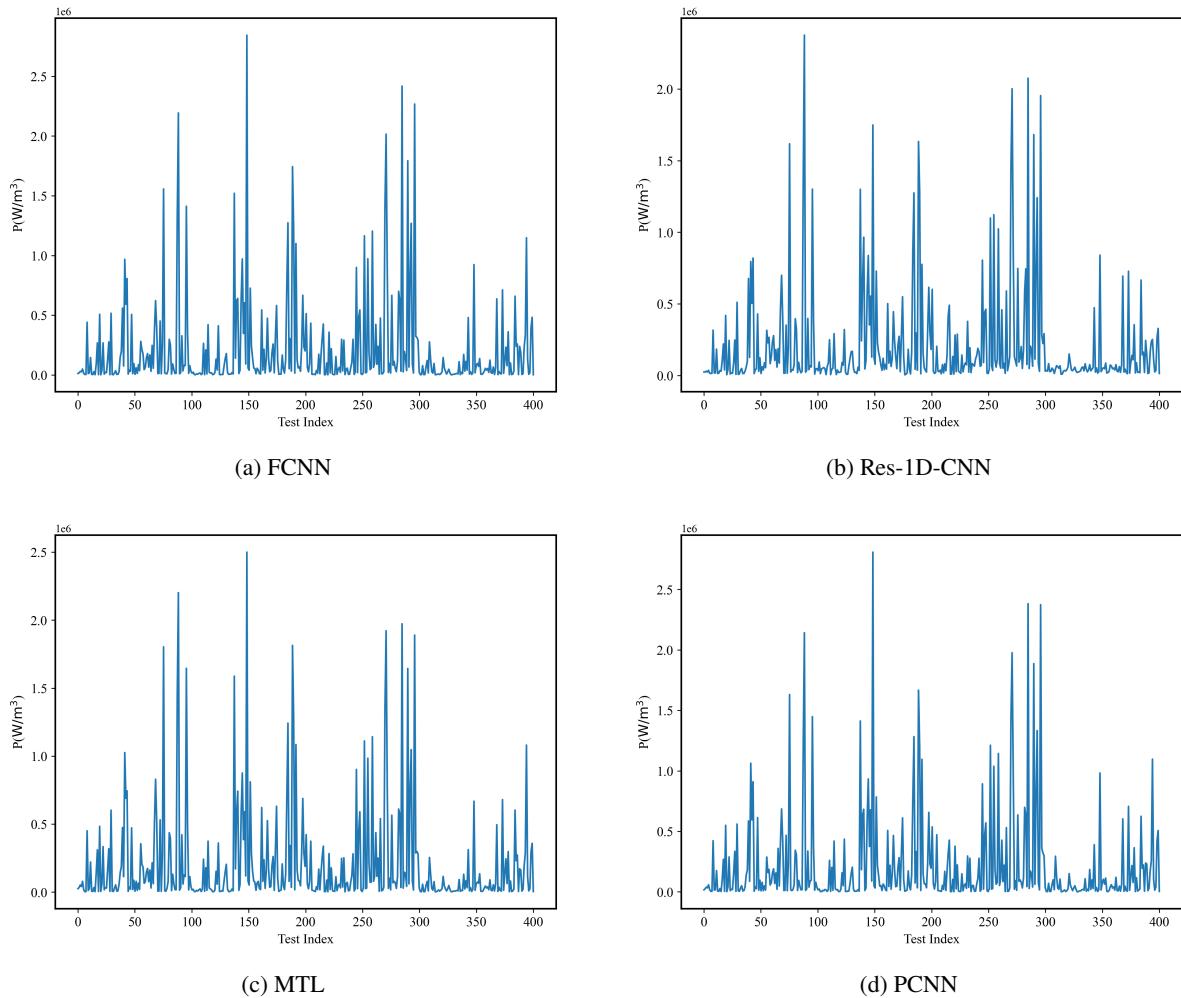


图 7.27 4 种预测模型的磁芯损耗预测结果

表 7.6 问题四励磁芯损耗预测结果

样本序号	FCNN	Res-1D-CNN	MTL	PCNN
16	2035.2516	29284.277	3488.1846	3342.6057
76	1556890.9	1617232.5	1803428.1	1630763.2
98	13484.493	9819.393	16302.925	14192.076
126	1832.5481	7579.2715	4823.7944	2839.191
168	143192.55	242262.69	173151.95	185814.88
230	63583.723	87209.5	57835.844	67042.8
271	2016077.8	2002435.8	1921618.1	1976048.8
338	10878.623	38314.746	10813.469	9290.566
348	923657.56	839963.8	668534.7	982625.56
379	2060.5757	17282.168	2907.8127	2888.5818

从测试结果可以看出，不同的数据驱动方法其对同一个样本的磁芯损耗预测在同一个量级范围内。然而，PCNN 模型基于先验物理模型知识，在小样本训练先能够更好地泛化到不同材料和工况条件，因此，本文最终选取 PCNN 磁芯材料损耗预测模型作为多目标优化问题的目标函数之一。

## 8 磁性元件的最优化条件

在本章节，主要介绍了考虑磁芯损耗和传输磁能两个目标函数的多目标优化问题的求解。首先，根据 PCNN 预测模型建立磁芯损耗目标函数，再结合传输磁能定义，构建多目标优化问题，并且最后将多目标优化问题转化为多个单目标优化问题。其次，本文采用基于树状结构的贝叶斯优化方法在高维参数空间中寻找该优化问题的最优解。最后，本文通过调整设定的权重值大小，讨论权重对最小磁芯损耗和最大传输磁能的影响。

### 8.1 多目标优化问题构建

在工程实践中，为了实现磁性元件整体性能的卓越与最优化，需要综合考虑多个评价指标，其中，传输磁能就是重要的评价指标之一，因此，需要同时考虑磁芯损耗与传输磁能这两个评价指标。

首先，利用 PCNN 预测模型构建磁芯损耗目标函数，即磁芯损耗可由式8.18表示。

$$P = P_h + P_{cl} + P_e = k_h * f * B_m^\beta + k_{cl} * f^2 * B_m^2 + k_e * f^{1.5} * B_m^{1.5} \quad (8.18)$$

$$k_h = PCNN_{k_h}(T, f, B_m, W, M) \quad (8.19)$$

$$\beta = PCNN_\beta(T, f, B_m, W, M) \quad (8.20)$$

$$k_{cl} = PCNN_{k_{cl}}(T, f, B_m, W, M) \quad (8.21)$$

$$k_e = PCNN_{k_e}(T, f, B_m, W, M) \quad (8.22)$$

其次，根据传输磁能的定义构建第二个目标函数，即传输磁能可由式8.23表示。

$$E = f * B_m \quad (8.23)$$

为了最小化磁芯损耗，并且最大化传输磁能，即构建多目标优化问题，其中优化目标和约束如式8.24表示。

$$\begin{aligned} & \min_{T, f, B_m, W, M} P + \max_{T, f, B_m, W, M} E \\ s.t. \quad & \left\{ \begin{array}{l} 20 \leq T \leq 100 \\ 50000 \leq f \leq 500000 \\ 0 < B_m < 1 \\ W \in \{1, 2, 3\} \\ M \in \{1, 2, 3, 4\} \end{array} \right. \end{aligned} \quad (8.24)$$

为了求解上述多目标优化问题，本文采用加权法，通过引入权重因子将上面的多目标优化问题转化为单目标优化问题，从而采用有效地算法进行求解。转化后的单目标优化问

题如式8.25表示。

$$\begin{aligned} & \min_{T,f,B_m,W,M} P - k * E \\ s.t. \quad & \left\{ \begin{array}{l} 20 \leq T \leq 100 \\ 50000 \leq f \leq 500000 \\ 0 < B_m < 1 \\ W \in \{1, 2, 3\} \\ M \in \{1, 2, 3, 4\} \end{array} \right. \end{aligned} \quad (8.25)$$

## 8.2 基于 TPE 的优化问题求解

TPE (Tree-structured Parzen Estimator) 算法是一种用于贝叶斯优化的算法，主要用于高维参数空间中寻找最优参数配置。它通过对搜索空间的建模，采用概率密度估计的方式，有选择地在参数空间中采样，从而在较短的时间内找到性能较好的参数配置。TPE 算法的核心思想是使用两个不同的概率密度函数来建模参数的条件概率分布：一个用于建模表现好的参数配置的概率密度函数，另一个用于建模未知的参数配置的概率密度函数。

TPE 算法在超参数优化、机器学习模型调优等领域得到了广泛的应用，尤其在搜索空间较大、目标函数难以优化的情况下，其效果往往优于随机搜索等传统优化方法。

TPE 算法通过对条件概率分布  $p(x|y)$  和边缘概率分布  $p(y)$  进行建模，然后通过贝叶斯公式来计算后验概率分布  $p(y|x)$ 。具体来说，TPE 使用两个密度函数来定义  $p(x|y)$ ：

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{otherwise} \end{cases} \quad (8.26)$$

其中， $l(x)$  是使用观测空间  $\{x^{(i)}\}$  来建立的，该观测空间对应的损失  $f(x^{(i)})$  小于  $y^*$ ，使用剩下的观测来建立  $g(x)$ 。TPE 选择期望改进 (Expected Improvement, EI) 作为采集函数：

$$EI(x) = \int_{-\infty}^{y^*} (y^* - y)p(y|x) dy \quad (8.27)$$

为了简化上式，我们首先对分母进行构造：

$$p(x) = \gamma l(x) + (1 - \gamma)g(x) \quad (8.28)$$

最后，EI 可以化简为：

$$EI(x) = \frac{l(x)}{g(x)} \quad (8.29)$$

最大化 EI 的点  $x^*$  即为下一次迭代的候选参数：

$$x^* = \arg \max EI(x) \quad (8.30)$$

基于上述讨论，本文利用 TPE 算法对优化问题进行求解，通过调整权重因子研究不同权重值下的最优解。

### 8.3 不同权重下的最优组合解

不同的权重因子对优化问题的意义不同，越大的权重因子证明对传输磁能的重要性评估越高，因此，实际应用中可以根据磁芯材料目前的需求，合理地调整权重因子，实现满足传输磁能要求下的最小磁芯损耗。本文利用 TPE 算法分别求解了权重因子  $k$  从 0.5 到 5 的最优解变化趋势，如表8.7所示。

表 8.7 不同权重因子下的最优解

权重	B	F	T	M	W
0.5	0.001964	494527.672	89.827	材料 3	正弦波
1.0	.002919	459453.329	84.669	材料 1	正弦波
1.5	0.002427	476001.751	89.315	材料 2	正弦波
2.0	0.003817	444310.256	72.436	材料 2	三角波
2.5	0.016492	499456.045	62.160	材料 3	正弦波
3.0	0.017134	499571.368	93.030	材料 3	正弦波
3.5	0.018811	499776.253	25.462	材料 1	正弦波
4.0	0.018478	499819.578	28.518	材料 3	正弦波
4.5	0.023042	499929.852	98.238	材料 1	正弦波
5.0	0.028436	499457.455	91.452	材料 3	正弦波

其次，本文研究不同权重因子下的最小磁芯损耗和最大传输磁能趋势，寻找最佳的权重因子选择。不同权重因子下的最小磁芯损耗和最大传输磁能趋势如图8.28 所示。

从图中可以看出，当权重值为 0.45 或 0.5 时，可以既满足较低的最小磁芯损耗，又可以满足较高的最大传输磁能，因此本文选择这两个权重值下的最优解作为最终的组合，如表8.8所示。

从表中可以看出，最优解可以选择材料 1 和材料 3 下的正弦波，并且温度较高的情况下可以保持较小的磁芯损失。其次励磁频率需要较大，这样可以保证较大的传输磁能，这是因为传输磁能表达式中励磁频率对其为一阶影响，而磁芯损耗表达式中励磁频率对其为高阶影响，因此在多目标优化求解后计算求得频率应当保持较高的水准。

表 8.8 最终确定权重因子下的最优解

权重	B	F	T	M	W
4.5	0.023042	499929.852	98.238	材料 1	正弦波
5.0	0.028436	499457.455	91.452	材料 3	正弦波

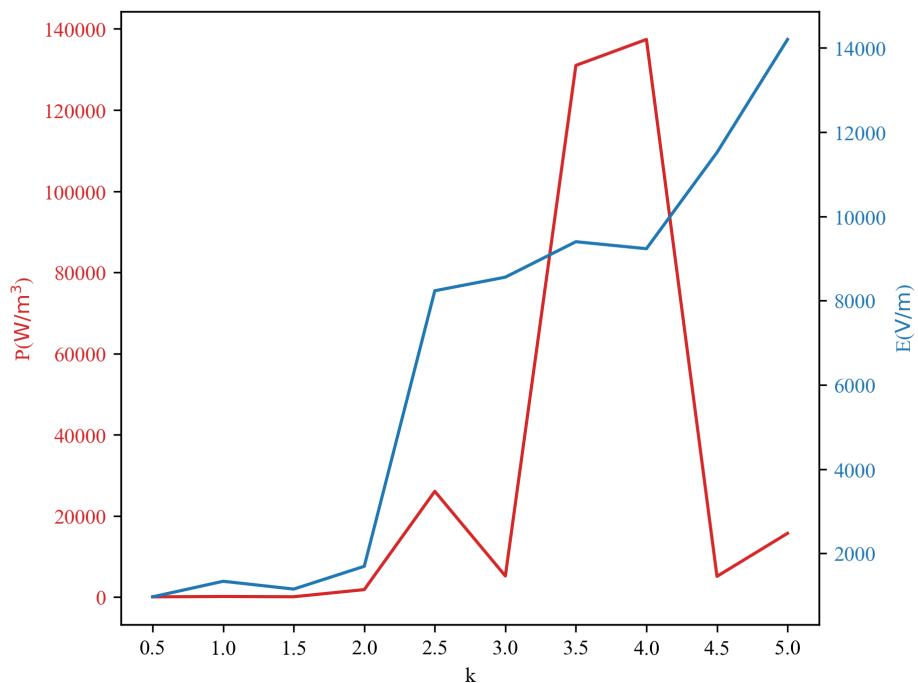


图 8.28 权重因子对的最小磁芯损耗和最大传输磁能影响趋势

## 9 结论

本文根据曲线的一阶导数和二阶导数提取曲线的隐含特征，利用对 DBSCAN 聚类方法对励磁波形分类，实现了较高的分类准确性，相较于传统机器学习算法性能提升明显。

其次，本文研究斯坦麦茨方程参数与温度之间的内在关系，通过结合先验物理信息，拟合参数与温度之间的映射曲线，通过对参数的修正进而实现在不同温度下磁芯损耗预测更加精准的斯坦麦茨方程。

为了进一步探究工况因素对磁芯损耗的影响，本文针对三个重要影响因素温度、励磁波形、磁芯材料对于达成最小磁芯损耗的单目标求解，分别使用离散网格搜索和 BFGS 优化相结合的结果揭示了磁芯损耗的复杂性。在离散网格搜索中，材料 4 在 70°C 下的表现优异，显示其在高温条件下的低能耗特性，这可能得益于其特殊的材料结构。然而，BFGS 优化表明，材料 1 在 25°C 时可能实现更低的损耗，表明温度、材料和励磁波形之间的关系并非简单的线性相关。因此，利用多元回归和随机森林的方法综合评估这三种因素及其协同效应，能够更全面地反映其对磁芯损耗的复杂非线性影响。尽管结果具有参考价值，但仍需进一步验证不同材料在实际工作条件下的表现，并考虑其他潜在因素对损耗的影响，以优化模型的准确性和适用性。

最后，为了实现跨材料多工况下的磁芯损耗预测，本文建立了物理信息嵌入的神经网络模型，并根据预测模型输出的磁芯损耗和传输磁能构建多目标优化问题，通过加权法转化为单目标问题求解。由于存在离散和连续型变量，因此本文采用树状结构 TPE 搜索算法计算不同权重因子下的最优解，实现了在不同工作需求下的磁芯材料最优工况求解，更加满足实际工程需要。

## 参考文献

- [1] Wang X, Pei Y, Chen L, et al., A Review of Magnetic Core Loss Prediction Methods for High-Frequency Power Electronic Transformers, *IEEE Transactions on Power Electronics*, 38(11):13515-13531, 2023.
- [2] 王兴勇, 陈乾宏, 郑大江, 等, 软磁材料在电力电子中的应用研究进展, *电工技术学报*, 31(21):1-13, 2016.
- [3] 郑大江, 王兴勇, 黄金, 高频磁性元件设计方法研究综述, *中国电机工程学报*, 33(3): 1-15, 2013.
- [4] Liu M, Wang C, Wang C, et al., An Improved Loss Prediction Model for Soft Magnetic Composite Materials Considering Temperature Effects, *IEEE Transactions on Industrial Electronics*, 69(12):12697-12707, 2022.
- [5] Muhlethaler J, Biela J, Kolar J W, et al., Modeling and multi-objective optimization of inductive power components, *IEEE Transactions on Power Electronics*, 28(1):246-255, 2012.
- [6] Ester M, Kriegel H P, Sander J, et al., DBSCAN: A density-based algorithm for discovering clusters in large spatial databases with noise, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*:226-231, 1996.
- [7] Bertotti G, General properties of power losses in soft ferromagnetic materials, *IEEE Transactions on magnetics*, 24(1):621-630, 1988.
- [8] Steinmetz C P, On the law of hysteresis, *Transactions of the American Institute of Electrical Engineers*, 9(1):3-64, 1892.
- [9] Breiman L, Random forests, *Machine learning*, 45(1):5-32, 2001.
- [10] Hornik K, Stinchcombe M, White H, Multilayer feedforward networks are universal approximators, *Neural networks*, 2(5):359-366, 1989.
- [11] He K, Zhang X, Ren S, et al., Deep residual learning for image recognition, *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770-778, 2016.
- [12] Raissi M, Perdikaris P, Karniadakis G E, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *Journal of Computational Physics*, 378:686-707, 2019.

## 附录 A Python 源程序

### A.1 第1问程序

二阶导初筛正弦波并校验.py

```
import pandas as pd
import numpy as np

# 读取 Excel 文件
file_path = r'C:\Users\ASUS\Desktop\fenlei\xiaoyang.xlsx'
data = pd.read_excel(file_path, sheet_name='材料1')
# 提取磁通密度数据 (假设从第5列开始)
flux_density_data = data.iloc[:, 4:]

# 计算一阶导数 (磁通密度的变化率)
flux_density_first_derivative = flux_density_data.diff(axis=1)

# 计算二阶导数 (磁通密度变化率的变化率, 即加速度)
flux_density_second_derivative =
    flux_density_first_derivative.diff(axis=1)

# 归一化处理
normalized_second_derivative =
    flux_density_second_derivative.copy()
max_values = normalized_second_derivative.max(axis=1) #
    # 计算每行的最大值
normalized_second_derivative =
    normalized_second_derivative.div(max_values, axis=0) # 归一化

# 设置阈值
threshold = 500

# 遍历样本并统计分类结果
classification_results = []

for sample_index in
    range(normalized_second_derivative.shape[0]):
    # 统计位于区间 (-0.1, 0.1) 内的二阶导数值的点数
    count_in_range =
        ((normalized_second_derivative.iloc[sample_index, 2:] >=
```

```

-0.1) & (normalized_second_derivative.iloc[sample_index,
2:] <= 0.1)).sum()

# 判定分类
if count_in_range < threshold:
    classification_results.append(0) # 正弦波
else:
    # 进一步区分三角波和梯形波
    classification_results.append(1)

# 将结果保存到向量文件
result_vector = np.array(classification_results)
np.savetxt('shaichu_zhengxianbo.txt', result_vector, fmt='%d')

# 输出分类结果
for i, result in enumerate(classification_results):
    print(f"样本 {i + 1}: {result}")

```

### 一阶导再 DBSCAN 聚类求类别数.py

```

import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN

# 读取 Excel 文件
file_path = r'C:\Users\ASUS\Desktop\fengei\xiaoyang.xlsx'
data = pd.read_excel(file_path, sheet_name='材料1')

# 提取磁通密度数据 (假设从第5列开始)
flux_density_data = data.iloc[:, 4:]
# print(flux_density_data)
# 计算一阶导数 (磁通密度的变化率)
flux_density_first_derivative =
    flux_density_data.diff(axis=1).dropna(axis=1, how='all') #
    去除空列
print("Number of samples:",
      flux_density_first_derivative.shape[0])

# 设置要处理的样本数
num_samples = data.shape[0] # 动态获取行数

```

```

cluster_counts = []

# 遍历每一行的导数数据并进行DBSCAN聚类
for sample_index in range(num_samples):
    # 获取当前行的一阶导数数据
    first_derivative_sample =
        flux_density_first_derivative.iloc[sample_index]
    .dropna().values.reshape(-1, 1)

    # 使用DBSCAN进行聚类
    # dbSCAN = DBSCAN(eps=0.00001, min_samples=20) #
    #     eps和min_samples可以调整
    dbSCAN = DBSCAN(eps=0.00002, min_samples=20) #
    #     eps和min_samples可以调整
    dbSCAN.fit(first_derivative_sample)

    # 获取聚类标签 (-1 表示噪声点)
    labels = dbSCAN.labels_

    # 统计去除噪声点后的类别数
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

    # 将类别数保存
    cluster_counts.append(n_clusters)

# 将聚类类别数结果输出
print("Each sample's DBSCAN cluster counts:", cluster_counts)

# 将结果保存到向量文件
result_vector = np.array(cluster_counts)
np.savetxt(r'C:\Users\ASUS\Desktop\fengei\cluster_counts.txt',
           result_vector, fmt='%d')

```

## A.2 第2问程序

原斯坦麦茨方程系数关于温度拟合.py

```

import pandas as pd
import numpy as np

```

```

from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# 设置中文字体
plt.rcParams['axes.unicode_minus'] = False # 解决负号显示问题
# 设置字体和字号
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['font.size'] = 12 # 设置字体大小
# 定义斯坦麦茨方程
def steinmetz_eq(frequency, flux_density, k1, alpha1, beta1):
    return k1 * (frequency ** alpha1) * (flux_density ** beta1)

# 定义温度依赖的拟合方程 (修正后的形式)
def temp_dependent_eq(T, a, b):
    return a * np.log(T) + b

# 存储温度和对应的拟合参数
temperatures = [25, 50, 70, 90]
k1_values = []
alpha1_values = []
beta1_values = []

# 创建一个图, 用来对比四个温度的拟合曲线
plt.figure(figsize=(10, 8))

# 遍历每个温度
for temp in temperatures:
    # 动态读取不同温度下的训练集数据
    X_train = pd.read_excel(f'X_train_{temp}_processed.xlsx')
    y_train = pd.read_excel(f'y_train_{temp}_processed.xlsx')

    # 提取频率和磁通密度
    frequency_train = X_train['频率, Hz']
    flux_density_train = X_train['Magnetic_flux_density_peak']

    # 使用curve_fit拟合斯坦麦茨方程参数
    popt, pcov = curve_fit(lambda f, k1, alpha1, beta1:
                           steinmetz_eq(f, flux_density_train, k1, alpha1, beta1),

```

```

frequency_train, y_train.values.flatten()))

# 输出拟合参数
k1, alpha1, beta1 = popt
k1_values.append(k1)
alpha1_values.append(alpha1)
beta1_values.append(beta1)
print(f"Fitted parameters for {temp}°C: k1={k1},
      alpha1={alpha1}, beta1={beta1}")

# 生成频率范围（假设频率范围为训练集中频率的范围）
freq_range = np.linspace(frequency_train.min(),
                          frequency_train.max(), 100)

# 使用拟合的参数计算曲线上的预测值
y_curve = steinmetz_eq(freq_range,
                       np.mean(flux_density_train), k1, alpha1, beta1)

# 绘制不同温度下的拟合曲线
plt.plot(freq_range, y_curve, label=f'{temp}°C')

# 调整图像细节
plt.xlabel('Frequency (Hz)')
plt.ylabel('Core Loss')
plt.title('Fitted Steinmetz Curves at Different Temperatures')
plt.legend()
plt.grid(True)
plt.savefig('fitted_curves_comparison.png')
plt.show()

# 创建三个子图，显示k1、alpha1、beta1随温度变化的折线图
plt.figure(figsize=(15, 6))

# k1 参数随温度变化的图
plt.subplot(1, 3, 1)
plt.plot(temperatures, k1_values, marker='o', color='blue')
plt.title('k1 vs Temperature')
plt.xlabel('Temperature (°C)')
plt.ylabel('k1 Value')

```

```

plt.grid(True)

# alpha1 参数随温度变化的图
plt.subplot(1, 3, 2)
plt.plot(temperatures, alpha1_values, marker='o',
         color='green')
plt.title('alpha1 vs Temperature')
plt.xlabel('Temperature (°C)')
plt.ylabel('alpha1 Value')
plt.grid(True)

# beta1 参数随温度变化的图
plt.subplot(1, 3, 3)
plt.plot(temperatures, beta1_values, marker='o', color='red')
plt.title('beta1 vs Temperature')
plt.xlabel('Temperature (°C)')
plt.ylabel('beta1 Value')
plt.grid(True)

# 保存参数变化图
plt.tight_layout()
plt.savefig('parameter_vs_temperature.png')
plt.show()

# 对 k1、alpha1、beta1 分别进行温度拟合
popt_k1, _ = curve_fit(temp_dependent_eq, temperatures,
                       k1_values)
popt_alpha1, _ = curve_fit(temp_dependent_eq, temperatures,
                           alpha1_values)
popt_beta1, _ = curve_fit(temp_dependent_eq, temperatures,
                          beta1_values)

# 生成温度范围，用于拟合曲线的绘制
temp_range = np.linspace(min(temperatures),
                         max(temperatures), 100)

# 绘制 k1 拟合结果
plt.figure(figsize=(15, 5))

```

```

plt.subplot(1, 3, 1)
plt.scatter(temperatures, k1_values, label='Data Points',
            color='blue')
plt.plot(temp_range, temp_dependent_eq(temp_range, *popt_k1),
         label='Fitted Curve', color='blue')
plt.title('k1 vs Temperature (Fitted)')
plt.xlabel('Temperature (°C)')
plt.ylabel('k1 Value')
plt.legend()
plt.grid(True)

# 绘制 alpha1 拟合结果
plt.subplot(1, 3, 2)
plt.scatter(temperatures, alpha1_values, label='Data Points',
            color='green')
plt.plot(temp_range, temp_dependent_eq(temp_range,
                                         *popt_alpha1), label='Fitted Curve', color='green')
plt.title('alpha1 vs Temperature (Fitted)')
plt.xlabel('Temperature (°C)')
plt.ylabel('alpha1 Value')
plt.legend()
plt.grid(True)

# 绘制 beta1 拟合结果
plt.subplot(1, 3, 3)
plt.scatter(temperatures, beta1_values, label='Data Points',
            color='red')
plt.plot(temp_range, temp_dependent_eq(temp_range,
                                         *popt_beta1), label='Fitted Curve', color='red')
plt.title('beta1 vs Temperature (Fitted)')
plt.xlabel('Temperature (°C)')
plt.ylabel('beta1 Value')
plt.legend()
plt.grid(True)

# 保存拟合曲线
plt.tight_layout()
plt.savefig('fitted_parameters_vs_temperature.png', dpi=300)
plt.show()

```

```

# 打印拟合结果
print(f"Fitted equation for k1: k1 = {popt_k1[0]} * log(T) +
      {popt_k1[1]}")
print(f"Fitted equation for alpha1: alpha1 = {popt_alpha1[0]} *
      log(T) + {popt_alpha1[1]}")
print(f"Fitted equation for beta1: beta1 = {popt_beta1[0]} *
      log(T) + {popt_beta1[1]}")

```

### 修正前后的斯坦麦茨方程的预测效果对比.py

```

import pandas as pd
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用黑体
plt.rcParams['axes.unicode_minus'] = False # 解决负号显示问题

# 定义新的斯坦麦茨方程，包含温度修正项
def steinmetz_eq_with_temp(frequency, flux_density,
                           temperature):
    k = 0.7021 * np.log(temperature) + 0.4427
    alpha = 0.2498 * np.log(temperature) + 1.176
    beta = 0.1241 * np.log(temperature) + 2.189
    return k * (frequency ** alpha) * (flux_density ** beta)

# 定义原始的斯坦麦茨方程（不带温度项）
def steinmetz_eq(frequency, flux_density, k, alpha, beta):
    return k * (frequency ** alpha) * (flux_density ** beta)

# 读取训练集和测试集数据
X_train = pd.read_excel('X_train_processed.xlsx')
y_train = pd.read_excel('y_train_processed.xlsx')
X_test = pd.read_excel('X_test_processed.xlsx')
y_test = pd.read_excel('y_test_processed.xlsx')

# 提取频率、磁通密度和温度

```

```

frequency_train = X_train['频率, Hz']
flux_density_train = X_train['Magnetic_flux_density_peak']
temperature_train = X_train['温度, oC']

# 1. 使用修正后的斯坦麦茨方程（包含温度修正项）进行拟合
popt_with_temp, _ = curve_fit(
    lambda f, k, alpha, beta, gamma: steinmetz_eq_with_temp(f,
        flux_density_train, temperature_train),
    frequency_train,
    y_train.values.flatten()
)

# 输出修正方程的拟合参数
k_with_temp, alpha_with_temp, beta_with_temp, gamma_with_temp
= popt_with_temp
print(f"Fitted parameters (with temperature correction):"
      f"\nk={k_with_temp}, alpha={alpha_with_temp},"
      f"\nbeta={beta_with_temp}, gamma={gamma_with_temp}")

# 使用修正方程的参数对测试集进行预测
frequency_test = X_test['频率, Hz']
flux_density_test = X_test['Magnetic_flux_density_peak']
temperature_test = X_test['温度, oC']
y_pred_with_temp = steinmetz_eq_with_temp(frequency_test,
    flux_density_test, temperature_test)

# 计算修正方程的均方误差 (MSE)
mse_with_temp = mean_squared_error(y_test, y_pred_with_temp)
print(f"Mean Squared Error (with temperature correction):"
      f"\n{mse_with_temp}")

# 2. 使用原始斯坦麦茨方程（不带温度项）进行拟合
popt_original, _ = curve_fit(
    lambda f, k, alpha, beta: steinmetz_eq(f,
        flux_density_train, k, alpha, beta),
    frequency_train,
    y_train.values.flatten()
)

```

```

# 输出原始方程的拟合参数
k_original, alpha_original, beta_original = popt_original
print(f"Fitted parameters (original): k={k_original},
      alpha={alpha_original}, beta={beta_original}")

# 使用原始方程的参数对测试集进行预测
y_pred_original = steinmetz_eq(frequency_test,
                               flux_density_test, k_original, alpha_original, beta_original)

# 计算原始方程的均方误差 (MSE)
mse_original = mean_squared_error(y_test, y_pred_original)
print(f"Mean Squared Error (original): {mse_original}")

# 创建一个图表，分别展示两种方程的结果对比
fig, ax = plt.subplots(1, 2, figsize=(16, 6))

# 1. 左图：修正后的斯坦麦茨方程实际值与预测值的对比图
ax[0].plot(y_test, label='Actual Values')
ax[0].plot(y_pred_with_temp, label='Predicted Values (with
    temperature)', linestyle='--')
ax[0].set_xlabel('Sample Points')
ax[0].set_ylabel('Core Loss')
ax[0].legend()
ax[0].set_title(f'Steinmetz Equation with Temperature
    Correction\nMSE: {mse_with_temp:.2f}')

# 2. 右图：原始斯坦麦茨方程实际值与预测值的对比图
ax[1].plot(y_test, label='Actual Values')
ax[1].plot(y_pred_original, label='Predicted Values
    (original)', linestyle='--')
ax[1].set_xlabel('Sample Points')
ax[1].set_ylabel('Core Loss')
ax[1].legend()
ax[1].set_title(f'Original Steinmetz Equation\nMSE:
    {mse_original:.2f}')

```

### A.3 第3问程序

```

# 频率分析
import matplotlib
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# 设置中文字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['axes.unicode_minus'] = False

# 检查 Excel 文件中的所有工作表名称
def check_sheet_names(file_path):
    xl = pd.ExcelFile(file_path)
    print(xl.sheet_names)

file_path = '附件一（训练集）.xlsx'
check_sheet_names(file_path) # 打印所有的工作表名称，便于检查

# 加载并准备数据
def load_and_prepare_data(file_path):
    materials = {}
    for i in range(1, 5):
        sheet_name = f'材料{i}'
        # 加载每个工作表的数据
        materials[i] = pd.read_excel(file_path,
                                      sheet_name=sheet_name)
        # 重新命名列名
        materials[i].columns = ['温度', '频率', '磁芯损耗',
                               '励磁波形'] + ['磁通密度{}'.format(j) for j in range(1,
                                                                           1025)]
        # 添加材料类型列
        materials[i]['材料类型'] = f'材料{i}'

    # 合并所有材料的数据
    data = pd.concat(materials.values(), ignore_index=True)
    return data

# 加载数据
data = load_and_prepare_data(file_path)
print("原始数据预览：")

```

```

print(data.head())

# 筛选频率在50000附近的数据，定义一个合理的范围
frequency_range = 500 # 可以根据需要调整范围

filtered_data = data[(data['频率'] >= 79400 - frequency_range)
                     & (data['频率'] <= 79400 + frequency_range)]
print(f"\n筛选后频率在{79400±frequency_range}范围内的数据: ")
print(filtered_data.head())

# 如果需要保存筛选后的数据，可以使用以下代码将其保存为 Excel 文件
filtered_data.to_excel("筛选后的数据.xlsx", index=False)

# 可视化筛选后的数据频率分布
plt.figure(figsize=(10, 6))
sns.histplot(data=filtered_data, x='频率', bins=30, kde=True)
plt.title(f'筛选后的数据频率分布 ({50000±frequency_range})')
plt.xlabel('频率')
plt.ylabel('样本数量')
plt.grid(True)
plt.show()

# 绘制整体情况
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 定义文件路径
file_path = '附件一（训练集）.xlsx'
save_file = '筛选后的合并数据.xlsx'

# 加载和筛选数据
def load_and_filter_data_for_sheet(file_path, sheet_name,
                                    frequency_range=1000, target_frequency=79000):

```

```

data = pd.read_excel(file_path, sheet_name=sheet_name)
data.columns = ['温度', '频率', '磁芯损耗', '励磁波形'] +
['磁通密度{}'.format(j) for j in range(1, 1025)]
filtered_data = data[(data['频率'] >= target_frequency -
frequency_range) &
                      (data['频率'] <= target_frequency +
frequency_range)].copy()
filtered_data['材料类型'] = sheet_name
filtered_data['材料编码'] = int(sheet_name[-1])

# 修改波形编码
waveform_mapping = {'正弦波': 1, '三角波': 2, '梯形波': 3}
filtered_data['波形编码'] =
    filtered_data['励磁波形'].map(waveform_mapping)

return filtered_data

# 合并所有筛选后的数据
def merge_filtered_data(file_path):
    xl = pd.ExcelFile(file_path)
    merged_data = pd.DataFrame()

    for sheet_name in xl.sheet_names:
        filtered_data = load_and_filter_data_for_sheet(file_path,
sheet_name)
        merged_data = pd.concat([merged_data, filtered_data],
ignore_index=True)

    return merged_data

# 加载合并数据并保存
merged_data = merge_filtered_data(file_path)
merged_data.to_excel(save_file, index=False)
print(f"筛选后的数据已保存至: {save_file}")

# 提取相关变量和目标变量
X = merged_data[['材料编码', '温度', '波形编码']]
y = merged_data['磁芯损耗']

```

```

# 创建绘图
fig = plt.figure(figsize=(18, 6)) # 调整整个图形的大小

# 绘制材料组合响应曲面
ax1 = fig.add_subplot(131, projection='3d') # 修改为131
for 材料编码_fixed in X['材料编码'].unique():
    subset = X[X['材料编码'] == 材料编码_fixed]
    ax1.scatter(subset['温度'], subset['波形编码'],
                y[subset.index], label=f'材料编码 {材料编码_fixed}', alpha=0.6)
ax1.set_xlabel('温度', labelpad=10)
ax1.set_ylabel('波形编码', labelpad=10)
ax1.set_zlabel('磁芯损耗', labelpad=10)
ax1.set_title('温度、波形编码与磁芯损耗 (不同材料编码)')
ax1.legend(loc='upper left')

# 绘制温度组合响应曲面
ax2 = fig.add_subplot(132, projection='3d') # 修改为132
for 温度_fixed in [25, 50, 70, 90]:
    subset = X[X['温度'] == 温度_fixed]
    ax2.scatter(subset['材料编码'], subset['波形编码'],
                y[subset.index], label=f'温度 {温度_fixed}℃', alpha=0.6)
ax2.set_xlabel('材料编码', labelpad=10)
ax2.set_ylabel('波形编码', labelpad=10)
ax2.set_zlabel('磁芯损耗', labelpad=10)
ax2.set_title('材料编码、波形编码与磁芯损耗 (不同温度)')
ax2.legend(loc='upper left')

# 绘制波形组合响应曲面
ax3 = fig.add_subplot(133, projection='3d') # 修改为133
for 波形编码_fixed in [1, 2, 3]:
    subset = X[X['波形编码'] == 波形编码_fixed]
    ax3.scatter(subset['温度'], subset['材料编码'],
                y[subset.index], label=f'波形编码 {波形编码_fixed}', alpha=0.6)
ax3.set_xlabel('温度', labelpad=10)
ax3.set_ylabel('材料编码', labelpad=10)
ax3.set_zlabel('磁芯损耗', labelpad=10)
ax3.set_title('温度、材料编码与磁芯损耗 (不同波形编码)')

```

```

ax3.legend(loc='upper left')

# 调整整体布局
plt.tight_layout()
plt.show()

```

## 单因素与双因素分析.py

```

# 修改绘图函数，替换图例中的编码
def plot_loess_by_group(average_data, fixed_waveforms, xlabel,
                       ylabel):
    waveform_mapping = {1: '正弦波', 2: '三角波', 3: '梯形波'}
    for fixed_waveform in fixed_waveforms:
        fig, ax = plt.subplots(figsize=(10, 6))
        fixed_data = average_data[average_data['波形编码'] == fixed_waveform]
        groups = fixed_data.groupby('材料编码')

        for name, group in groups:
            lowess_results = lowess(group['磁芯损耗'],
                                    group[xlabel], frac=0.3)
            ax.scatter(group[xlabel], group['磁芯损耗'],
                       alpha=0.4, label=f'材料 {name}') # 修改为材料
            ax.plot(lowess_results[:, 0], lowess_results[:, 1],
                    linewidth=2)

        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.legend()
        ax.grid(True)
        plt.show()

# 修改材料编码的图例
def plot_loess_by_material(average_data, fixed_material,
                           xlabel, ylabel):
    fig, ax = plt.subplots(figsize=(10, 6))
    fixed_data = average_data[average_data['材料编码'] == fixed_material]
    groups = fixed_data.groupby('温度')

```

```

for name, group in groups:
    lowess_results = lowess(group['磁芯损耗'], group[xlabel],
                           frac=0.3)
    ax.scatter(group[xlabel], group['磁芯损耗'], alpha=0.4,
               label=f'温度 {name}℃')
    ax.plot(lowess_results[:, 0], lowess_results[:, 1],
            linewidth=2)

    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.legend()
    ax.grid(True)
    plt.show()

# 修改波形编码的图例
def plot_loess_by_temperature(average_data, fixed_temperature,
                               xlabel, ylabel):
    fig, ax = plt.subplots(figsize=(10, 6))
    fixed_data = average_data[average_data['温度'] ==
                               fixed_temperature]
    groups = fixed_data.groupby('波形编码')

    waveform_mapping = {1: '正弦波', 2: '三角波', 3: '梯形波'}

    for name, group in groups:
        lowess_results = lowess(group['磁芯损耗'], group[xlabel],
                               frac=0.3)
        ax.scatter(group[xlabel], group['磁芯损耗'], alpha=0.4,
                   label=f'波形 {waveform_mapping[name]}') # 修改为波形名称
        ax.plot(lowess_results[:, 0], lowess_results[:, 1],
                linewidth=2)

        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.legend()
        ax.grid(True)
        plt.show()

# 绘制不同波形编码下的温度与磁芯损耗关系

```

```

plot_loess_by_group(average_data, [1, 2, 3], '温度', '磁芯损耗')
for fixed_material in range(1, 5): # 假设材料编码为1到4
    plot_loess_by_material(average_data, fixed_material,
                           '波形编码', '磁芯损耗')


for fixed_temperature in [25, 50, 70, 90]: # 固定的温度值
    plot_loess_by_temperature(average_data, fixed_temperature,
                             '材料编码', '磁芯损耗')


# 响应曲面
import pandas as pd
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 定义文件路径
file_path = '附件一（训练集）.xlsx'
save_file = '筛选后的合并数据.xlsx'

# 加载和筛选数据
def load_and_filter_data_for_sheet(file_path, sheet_name,
                                    frequency_range=1000, target_frequency=79000):
    data = pd.read_excel(file_path, sheet_name=sheet_name)
    data.columns = ['温度', '频率', '磁芯损耗', '励磁波形'] +
                  ['磁通密度{}'.format(j) for j in range(1, 1025)]
    filtered_data = data[(data['频率'] >= target_frequency -
                           frequency_range) &
                          (data['频率'] <= target_frequency +
                           frequency_range)].copy()

```

```

filtered_data['材料类型'] = sheet_name
filtered_data['材料编码'] = int(sheet_name[-1])

# 修改波形编码为 1, 2, 3
waveform_mapping = {'正弦波': 1, '三角波': 2, '梯形波': 3}
filtered_data['波形编码'] =
    filtered_data['励磁波形'].map(waveform_mapping)

return filtered_data

# 合并所有筛选后的数据
def merge_filtered_data(file_path):
    xl = pd.ExcelFile(file_path)
    merged_data = pd.DataFrame()

    for sheet_name in xl.sheet_names:
        filtered_data = load_and_filter_data_for_sheet(file_path,
            sheet_name)
        merged_data = pd.concat([merged_data, filtered_data],
            ignore_index=True)

    return merged_data

# 加载合并数据并保存
merged_data = merge_filtered_data(file_path)
merged_data.to_excel(save_file, index=False)
print(f"筛选后的数据已保存至: {save_file}")

# 提取相关变量和目标变量
X = merged_data[['材料编码', '温度', '波形编码']]
y = merged_data['磁芯损耗']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

# 使用多项式特征生成
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly_train = poly.fit_transform(X_train)

```

```

X_poly_test = poly.transform(X_test)

# 构建多项式回归模型
model = LinearRegression()
model.fit(X_poly_train, y_train)

# 计算训练集和测试集误差
train_pred = model.predict(X_poly_train)
test_pred = model.predict(X_poly_test)
train_mse = mean_squared_error(y_train, train_pred)
test_mse = mean_squared_error(y_test, test_pred)
print(f"\n训练集均方误差 (MSE) : {train_mse}")
print(f"测试集均方误差 (MSE) : {test_mse}")

# 构建用于绘制不同组合的网格数据
温度_range = np.linspace(X['温度'].min(), X['温度'].max(), 100)
    # 增加样本数量
波形编码_range = np.array([1, 2, 3])
材料编码_range = np.linspace(X['材料编码'].min(),
    X['材料编码'].max(), 100, dtype=int)

# 创建绘图
fig = plt.figure(figsize=(18, 6)) # 调整整个图形的大小

# 绘制材料组合响应曲面
ax1 = fig.add_subplot(131, projection='3d') # 横排布局
波形编码_fixed = 1 # 固定波形编码为1 (正弦波)
for 材料编码_fixed in 材料编码_range:
    温度_grid, 波形编码_grid = np.meshgrid(温度_range,
        波形编码_range)
    X_grid = np.c_[np.full(temperature_grid.size, 材料编码_fixed),
        temperature_grid.ravel(), 波形编码_grid.ravel()]
    X_poly_grid = poly.transform(X_grid)
    y_grid_pred = model.predict(X_poly_grid)
    y_grid_pred = np.maximum(y_grid_pred, 0) # 确保预测值不小于0
    y_grid_pred = y_grid_pred.reshape(temperature_grid.shape)
    ax1.plot_surface(temperature_grid, 波形编码_grid, y_grid_pred,
        alpha=0.7)
ax1.set_xlabel('温度')

```

```

ax1.set_ylabel('波形编码')
ax1.set_zlabel('磁芯损耗')
ax1.set_title(f'温度、波形编码与磁芯损耗\n(不同材料编码)')

# 绘制温度组合响应曲面
ax2 = fig.add_subplot(132, projection='3d') # 横排布局
材料编码_fixed = 2 # 固定材料编码为2
for 温度_fixed in [25, 50, 70, 90]: # 四种温度
    材料编码_grid, 波形编码_grid = np.meshgrid(材料编码_range,
                                                波形编码_range)
    X_grid = np.c_[材料编码_grid.ravel(),
                   np.full(材料编码_grid.size, 温度_fixed),
                   波形编码_grid.ravel()]
    X_poly_grid = poly.transform(X_grid)
    y_grid_pred = model.predict(X_poly_grid)
    y_grid_pred = np.maximum(y_grid_pred, 0) # 确保预测值不小于0
    y_grid_pred = y_grid_pred.reshape(材料编码_grid.shape)
    ax2.plot_surface(材料编码_grid, 波形编码_grid, y_grid_pred,
                     alpha=0.7)
ax2.set_xlabel('材料编码')
ax2.set_ylabel('波形编码')
ax2.set_zlabel('磁芯损耗')
ax2.set_title(f'材料编码、波形编码与磁芯损耗\n(不同温度)')

# 绘制波形组合响应曲面
ax3 = fig.add_subplot(133, projection='3d') # 横排布局
温度_fixed = 50 # 固定温度为50℃
for 波形编码_fixed in [1, 2, 3]: # 波形编码 [1, 2, 3]
    温度_grid, 材料编码_grid = np.meshgrid(温度_range,
                                              材料编码_range)
    X_grid = np.c_[材料编码_grid.ravel(), 温度_grid.ravel(),
                   np.full(温度_grid.size, 波形编码_fixed)]
    X_poly_grid = poly.transform(X_grid)
    y_grid_pred = model.predict(X_poly_grid)
    y_grid_pred = np.maximum(y_grid_pred, 0) # 确保预测值不小于0
    y_grid_pred = y_grid_pred.reshape(温度_grid.shape)
    ax3.plot_surface(温度_grid, 材料编码_grid, y_grid_pred,
                     alpha=0.7)
ax3.set_xlabel('温度')

```

```

ax3.set_ylabel('材料编码')
ax3.set_zlabel('磁芯损耗')
ax3.set_title(f'温度、材料编码与磁芯损耗\n(不同波形编码)')

# 调整整体布局
plt.tight_layout()
plt.show()

```

### 随机森林与线性回归.py

```

import pandas as pd

from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split,
    GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures

data = {
    '材料编码': [1] * 12 + [2] * 12 + [3] * 12 + [4] * 12,
    '温度': [25, 25, 25, 50, 50, 50, 70, 70, 70, 90, 90, 90] *
        4,
    '波形编码': [0, 1, 2] * 16,
    '磁芯损耗': [
        177901.107230, 228905.582918, 129350.250903,
        146479.094092, 143455.968748,
        66913.563687, 122450.203260, 91153.304033,
        89214.622771, 107804.260402,
        112686.618460, 68195.610245, 363018.185326,
        199014.490964, 111638.252447,
        221011.515584, 164278.788740, 130569.724177,
        283994.123919, 221991.914669,
        121495.990844, 204033.072037, 215465.995439,
        64602.843121, 440803.350926,
        334653.181748, 197810.803829, 323792.325398,
        303835.239849, 152688.426489,
        308679.472039, 299646.718253, 142231.108224,
        358734.455387, 266497.715088,
        109146.546874, 153577.430534, 236468.301644,
        44400.664569, 209975.070697,
    ]
}

```

```

    169474.502312, 41529.183501, 197256.544041,
    208035.109255, 30936.778928,
    154675.070252, 182068.149663, 49082.615708
]
}

df = pd.DataFrame(data)
# df['波形编码'] = df['波形编码'].replace({0: 2, 1: 3, 2: 1})

# 特征和标签
X = df[['材料编码', '温度', '波形编码']]
y = df['磁芯损耗']

# 生成交互项
poly = PolynomialFeatures(interaction_only=True,
                           include_bias=False)
X_poly = poly.fit_transform(X)

# 数据分割
X_train, X_test, y_train, y_test = train_test_split(X_poly,
                                                    y, test_size=0.2, random_state=42)

# 使用线性回归判断正负相关性
lin_model = LinearRegression()
lin_model.fit(X_train, y_train)

# 预测
y_pred = lin_model.predict(X_test)

# 性能评估
mse = mean_squared_error(y_test, y_pred)
print(f"均方误差: {mse:.2f}")

# 模型系数
print("特征系数 (正负相关性) : ")
feature_names = poly.get_feature_names(['材料编码', '温度',
                                         '波形编码'])
coef_signs = lin_model.coef_

```

```

for feature, coef in zip(feature_names, coef_signs):
    print(f"{feature}: {coef:.4f}")

# 创建随机森林模型进行特征重要性评估
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf,
                           param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=5,
                           n_jobs=-1)

grid_search.fit(X_train, y_train)

# 获取最佳参数和模型
best_rf = grid_search.best_estimator_

# 预测
y_pred_rf = best_rf.predict(X_test)

# 性能评估
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f"优化后的均方误差: {mse_rf:.2f}")

# 特征重要性
importance = best_rf.feature_importances_
print("特征重要性 (优化后) : ")

# 输出特征重要性, 乘以对应的符号
for feature, imp, sign in zip(feature_names, importance,
                               coef_signs):
    adjusted_importance = imp * (1 if sign > 0 else -1) #
        根据线性回归系数的符号调整重要性
    print(f"{feature}: {adjusted_importance:.4f}")

```

## 优化求解.py

```
# 系数法最小值
import numpy as np
from scipy.optimize import minimize

# 系数
coefficients = {
    '材料编码': -0.1748,
    '温度': -0.0585,
    '波形编码': 0.4680,
    '材料编码 温度': 0.2079,
    '材料编码 波形编码': -0.0524,
    '温度 波形编码': -0.0384
}

# 定义目标函数
def objective_function(params):
    material, temperature, waveform = params
    # 计算磁芯损耗
    loss = (coefficients['材料编码'] * material +
            coefficients['温度'] * temperature +
            coefficients['波形编码'] * waveform +
            coefficients['材料编码 温度'] * material * temperature +
            coefficients['材料编码 波形编码'] * material * waveform +
            coefficients['温度 波形编码'] * temperature * waveform)
    return loss

# 初始猜测（可以是任何合理的值）
initial_guess = [4, 20, 1] # 材料编码, 温度, 波形编码

# 定义约束条件和边界
bounds = [(1, 4), (25, 90), (0, 2)] #
    材料编码范围, 温度范围, 波形编码范围
# 此处编码为三角波, 梯形波, 正弦波

# 优化
result = minimize(objective_function, initial_guess,
                   bounds=bounds)
```

```

# 打印结果
optimal_material = round(result.x[0])
optimal_temperature = round(result.x[1])
optimal_waveform = round(result.x[2])
optimal_loss = result.fun

print(f"目标函数最小值: {optimal_loss:.2f} 对应的材料编码:
{optimal_material}, 温度: {optimal_temperature}, 波形编码:
{optimal_waveform}")

# 网格法最小值
# 定义离散点
材料编码_range = np.arange(1, 5) # 材料编码 1 到 4
温度_range = [25, 50, 70, 90] # 温度
波形编码_range = [1, 2, 3] # 波形编码

# 搜索所有组合的最小磁芯损耗
results = []

for 材料编码 in 材料编码_range:
    for 温度 in 温度_range:
        for 波形编码 in 波形编码_range:
            # 创建输入
            X_test = np.array([[材料编码, 温度, 波形编码]])
            X_poly_test = poly.transform(X_test)
            y_pred = model.predict(X_poly_test)
            results.append((材料编码, 温度, 波形编码, y_pred[0]))

# 将结果转换为DataFrame并查找最小值
results_df = pd.DataFrame(results, columns=['材料编码', '温度',
                                             '波形编码', '磁芯损耗'])
min_loss_row = results_df.loc[results_df['磁芯损耗'].idxmin()]

print(f"最小磁芯损耗:
{min_loss_row['磁芯损耗']:.4f}, 对应的参数为: 材料编码={min_loss_row['材料编码']},
温度={min_loss_row['温度']},
波形编码={min_loss_row['波形编码']}")

# 可视化结果

```

```

fig = plt.figure(figsize=(10, 6))
sc = plt.scatter(results_df['温度'], results_df['材料编码'],
                 c=results_df['磁芯损耗'], cmap='viridis')
plt.colorbar(sc, label='磁芯损耗')
plt.xlabel('温度')
plt.ylabel('材料编码')
plt.title('材料编码与温度对磁芯损耗的影响')
plt.show()

```

### 其他对比方法.py

```

# 肯德尔秩系数，效果较差
import pandas as pd
import numpy as np
from scipy.stats import kendalltau

# 定义文件路径
file_path = '附件一（训练集）.xlsx'

# 读取单个 sheet 的数据并进行筛选
def load_and_filter_data_for_sheet(file_path, sheet_name,
                                    frequency_range=1000, target_frequency=79000):
    # 加载数据
    data = pd.read_excel(file_path, sheet_name=sheet_name)
    # 重命名列名
    data.columns = ['温度', '频率', '磁芯损耗', '励磁波形'] +
                  ['磁通密度{}'.format(j) for j in range(1, 1025)]
    # 筛选频率在目标频率附近的数据
    filtered_data = data[(data['频率'] >= target_frequency -
                           frequency_range) &
                          (data['频率'] <= target_frequency +
                           frequency_range)].copy()
    # 添加材料类型列，编码材料
    filtered_data['材料类型'] = sheet_name
    filtered_data['材料编码'] = sheet_name[-1] # 假设材料编号在
                                                # '材料1', '材料2' ... 的最后一位
    # 编码励磁波形
    filtered_data['波形编码'] =
        filtered_data['励磁波形'].astype('category').cat.codes
return filtered_data

```

```

# 合并所有筛选后的数据
def merge_filtered_data(file_path):
    xl = pd.ExcelFile(file_path)
    merged_data = pd.DataFrame()

    for sheet_name in xl.sheet_names:
        filtered_data = load_and_filter_data_for_sheet(file_path,
            sheet_name)
        merged_data = pd.concat([merged_data, filtered_data],
            ignore_index=True)

    return merged_data

# 计算整体的肯德尔秩相关系数
def calculate_overall_kendall(merged_data):
    variables = ['材料编码', '温度', '波形编码']
    target = '磁芯损耗'
    results = []

    for var in variables:
        tau, p_value = kendalltau(merged_data[var],
            merged_data[target])
        results.append({
            '自变量': var,
            '肯德尔秩相关系数 (Tau)': tau,
            '显著性水平 (p-value)': p_value,
            '相关性程度': '强相关' if abs(tau) >= 0.5 else ('中等相关'
                if abs(tau) >= 0.3 else '弱相关')
        })
    return pd.DataFrame(results)

# 主函数，筛选数据并计算整体相关性
def main(file_path):
    # 合并筛选后的数据
    merged_data = merge_filtered_data(file_path)

    # 打印筛选后的数据预览
    print("筛选后的合并数据预览: ")

```

```

print(merged_data.head())

# 计算整体的肯德尔秩相关系数
overall_results = calculate_overall_kendall(merged_data)

# 输出整体相关性结果
print("\n整体单因素非线性相关性分析结果: ")
print(overall_results)

# 保存结果到 Excel 文件
overall_results.to_excel('整体单因素非线性相关性分析结果.xlsx',
                         index=False)

# 执行主函数
main(file_path)

# 方差分析
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
import matplotlib.pyplot as plt
import seaborn as sns
model_temp=ols('磁芯损耗~温度', data=data).fit()
anova_temp=anova_lm(model_temp)
print('温度单因素方差分析结果: ')
print(anova_temp)

model_material=ols('磁芯损耗~C(磁芯材料)', data=data).fit()
anova_material=anova_lm(model_material)
print('磁芯材料单因素方差分析结果: ')
print(anova_material)

model_waveform=ols('磁芯损耗~C(励磁波形)', data=data).fit()
anova_waveform=anova_lm(model_waveform)
print('励磁波形单因素方差分析结果: ')
print(anova_waveform)

model_interaction=ols('磁芯损耗 ~ C(温度) + C(磁芯材料) +
C(励磁波形) + C(温度):C(励磁波形) + C(磁芯材料):C(励磁波形) +

```

```

C(温度):C(磁芯材料)', data=data).fit()
anova_interaction=anova_lm(model_interaction)
print('交互作用方差分析结果: ')
print(anova_interaction)

plt.figure(figsize=(12, 6))
plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定中文字体
sns.boxplot(x='磁芯材料', y='磁芯损耗', data=data)
#plt.title('不同磁芯材料磁芯损耗箱线图')
plt.show()

plt.figure(figsize=(12, 6))
plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定中文字体
sns.boxplot(x='励磁波形', y='磁芯损耗', data=data)
#plt.title('不同励磁波形磁芯损耗箱线图')
plt.show()

plt.figure(figsize=(12, 6))
plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定中文字体
sns.boxplot(x='温度', y='磁芯损耗', data=data)
#plt.title('不同温度磁芯损耗箱线图')
plt.show()

```

#### A.4 第4问程序

question4.py

```

# 导入第三方库
import torch
import torch.nn as nn
from torch.utils.data.dataloader import DataLoader
from torch.optim.lr_scheduler import StepLR
import random
import numpy as np
import copy
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from math import sqrt
import os

```

```

from matplotlib import rcParams

# 设置全局字体为Times New Roman
rcParams['font.family'] = 'serif'
rcParams['font.serif'] = ['Times New Roman']

# 导入自定义库
from read_data_new import *
from models import *

# 设置随机数种子
def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True

setup_seed(27)

device = torch.device("cuda:0" if torch.cuda.is_available()
    else "cpu")

# 定义超参数
batch_size = 50 # 批次大小
num_epochs = 150 # 训练轮数
step_size = 10 # 学习率变化周期
gamma = 0.5 # 学习率更新值
learning_rate = 0.002 # 学习率
P0 = 1e-3

# 模型输出迭代初始值
k_h0 = 1
k_c10 = 1
k_e0 = 4
b0 = 2

'''

1. 输入数据：根据训练集和测试集相同数据，选取模型输入数据为[0温度， 1频率， 2磁芯材料， 4励磁波形， 磁通密度]

```

```

2. 材料类型采用数字代替，为1, 2, 3, 4，励磁波形同理，dict = ["正弦波":1,
   "三角波":2, "梯形波":3]
3. 模型采用全连接神经网络
    ...
# 建立字典
material_dict = {'材料1': 1, '材料2': 2, '材料3': 3, '材料4': 4}
wave_dict = {'正弦波': 1, '三角波': 2, '梯形波': 3}

# 读取训练数据
# [0温度, 1频率, 2磁芯材料, 3磁芯损耗, 4励磁波形, 磁通密度]
directory = f'data/'
file_name = 'train'
file_path_train = os.path.join(directory, file_name)
train_data = create_train_data(file_path_train)

# 转换第3,5列的数据
for row in train_data:
    row[2] = float(material_dict[row[2]])
    row[4] = wave_dict[row[4]]

# 提取磁通密度的统计学特征
B = train_data[:, 5:]
B_max = calculate_max(B).reshape(-1, 1)
train_data = np.concatenate((train_data[:, :5], B_max),
                           axis=1)

# 分离输入数据和标签数据
train_x = np.concatenate((train_data[:, :3], train_data[:, 4:]), axis=1)
train_y = train_data[:, 3].reshape(-1, 1)
train_x, valid_x, train_y, valid_y = \
    train_test_split(train_x, train_y, test_size=0.2,
                    random_state=420)

# 归一化训练数据
scaler_x, train_x_scaled = scale_train(train_x)
scaler_y, train_y_scaled = scale_train(train_y)
valid_x_scaled = scaler_x.transform(valid_x)
valid_x_scaled = torch.from_numpy(valid_x_scaled)

```

```

valid_y_scaled = scaler_y.transform(valid_y)
valid_y_scaled = torch.from_numpy(valid_y_scaled)

# 加载数据
val_batch_size = valid_x_scaled.shape[0]
input_dim = train_x_scaled.shape[1]

train_loader = DataLoader(TensorDataset(train_x_scaled,
                                         train_y_scaled),
                           batch_size=batch_size,
                           shuffle=True)

val_loader = DataLoader(TensorDataset(valid_x_scaled,
                                         valid_y_scaled),
                           batch_size=val_batch_size,
                           shuffle=True)

# 读取测试数据
# [0序号, 1温度, 2频率, 3磁芯材料, 4励磁波形, 磁通密度]
directory = f'data/'
file_name = 'test2'
file_path_test = os.path.join(directory, file_name)
test_data = create_test_data(file_path_test)

# 转换第4,5列的数据
for row in test_data:
    row[3] = float(material_dict[row[3]])
    row[4] = wave_dict[row[4]]

# 提取磁通密度的统计学特征
B = test_data[:, 5:]
B_max = calculate_max(B).reshape(-1, 1)
test_data = np.concatenate((test_data[:, :5], B_max), axis=1)
test_x = test_data[:, 1:]

# 数据归一化
test_x_scaled = scaler_x.transform(test_x)
test_x_scaled = torch.from_numpy(test_x_scaled)

```

```

# 加载数据
test_batch_size = test_x_scaled.shape[0]
test_loader = DataLoader(TensorDataset(test_x_scaled),
                        batch_size=test_batch_size,
                        shuffle=False)

# 创建模型实例
# net = MultiTaskModel(input_dim=input_dim,
#                       shared_dim=shared_dim).to(device)
net = FCNN(input_dim=input_dim, output_dim=1).to(device)

# 定义损失函数和优化器
loss_function = nn.MSELoss().to(device)
optimizer_net = torch.optim.Adam(net.parameters(),
                                 lr=learning_rate)
scheduler_net = StepLR(optimizer_net, step_size=step_size,
                       gamma=gamma)

min_epochs = 10
best_model_q4 = None
min_loss = 0.1
average_train_loss = []
average_val_loss = []

file_path_q4 = 'result/model_q4_fcnn_new.pth' # 模型保存路径

if os.path.exists(file_path_q4):
    print("FCNN 最佳模型文件存在, 跳过执行模型训练。")
else:
    print("FCNN 最佳模型文件不存在, 开始训练模型。")
    for epoch in range(num_epochs):
        net.train()
        train_loss = []
        for i, data in enumerate(train_loader, 0):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            inputs = inputs.float()
            labels = labels.float()


```

```

# 梯度清零
optimizer_net.zero_grad()

# 前向传播
y_pred = net(inputs)
y_pred = P0*torch.exp(y_pred)

loss = loss_function(y_pred, labels)

train_loss.append(loss.cpu().item())

# 更新梯度
loss.backward()

# 优化参数
optimizer_net.step() # 更新每个网络的权重

scheduler_net.step() # 调整学习率

net.eval()
with torch.no_grad():
    val_loss = []
    for i, data in enumerate(val_loader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device),
        labels.to(device)
        inputs = inputs.float()
        labels = labels.float()

        # 前向传播
        y_pred = net(inputs)
        y_pred = P0*torch.exp(y_pred)

        # 计算损失
        loss = loss_function(labels, y_pred)
        val_loss.append(loss.cpu().item())

if epoch > min_epochs and np.mean(val_loss) < min_loss:
    min_loss = np.mean(val_loss)

```

```

best_model_q4 = copy.deepcopy(net)

if (epoch == 0) | ((epoch + 1) % 20 == 0):
    print('epoch {:03d}'.format(epoch), 'train_loss {:.8f}'.format(np.mean(train_loss)), 'val_loss {:.8f}'.format(np.mean(val_loss)))

average_train_loss.append(np.mean(train_loss))
average_val_loss.append(np.mean(val_loss))

torch.save(best_model_q4, file_path_q4)

# 绘制训练集和验证集损失曲线
plt.figure(figsize=(8, 6))
plt.plot(average_train_loss, label='train loss')
plt.plot(average_val_loss, label='val loss')
plt.legend()
plt.savefig(f'./result/q4_loss_fcnn_new.png')
plt.show()

# 测试模型
with torch.no_grad():
    model = torch.load(file_path_q4)
    # print(model)

    # 预测测试数据
    for i, data in enumerate(test_loader, 0):
        inputs = data[0]
        inputs = inputs.to(device)
        inputs = inputs.float()

        # y_pred 为预测值
        y_pred = model(inputs)
        y_pred = P0*torch.exp(y_pred)

        y_pred = invert_scale(scaler_y,
                             y_pred.cpu().detach().numpy())

# 创建模型实例

```

```

net = CNN(input_dim=input_dim,
           output_dim=output_dim).to(device)

# 定义损失函数和优化器
loss_function = nn.MSELoss().to(device)
optimizer_net = torch.optim.Adam(net.parameters(),
                                 lr=learning_rate)
scheduler_net = StepLR(optimizer_net, step_size=step_size,
                       gamma=gamma)

train_loader = DataLoader(TensorDataset(train_x_scaled,
                                         train_y_scaled),
                          batch_size=batch_size,
                          shuffle=True)

val_loader = DataLoader(TensorDataset(valid_x_scaled,
                                       valid_y_scaled),
                        batch_size=val_batch_size,
                        shuffle=True)

min_epochs = 10
best_model_q4 = None
min_loss = 0.1
average_train_loss = []
average_val_loss = []
file_path_q4 = 'result/model_q4_cnn_new.pth'

if os.path.exists(file_path_q4):
    print("CNN 最佳模型文件存在, 跳过执行模型训练。")
else:
    print("CNN 最佳模型文件不存在, 开始训练模型。")
    for epoch in range(num_epochs):
        net.train()
        train_loss = []
        for i, data in enumerate(train_loader, 0):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            inputs = inputs.float()
            labels = labels.float()

```

```

# 梯度清零
optimizer_net.zero_grad()

# 前向传播
y_pred = net(inputs)
y_pred = P0*torch.exp(y_pred)

loss = loss_function(y_pred, labels)

train_loss.append(loss.cpu().item())

# 更新梯度
loss.backward()

# 优化参数
optimizer_net.step() # 更新每个网络的权重

scheduler_net.step() # 调整学习率

net.eval()
with torch.no_grad():
    val_loss = []
    for i, data in enumerate(val_loader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device),
                    labels.to(device)
        inputs = inputs.float()
        labels = labels.float()

        # 前向传播
        y_pred = net(inputs)
        y_pred = P0*torch.exp(y_pred)

        # 计算损失
        loss = loss_function(labels, y_pred)
        val_loss.append(loss.cpu().item())

if epoch > min_epochs and np.mean(val_loss) < min_loss:

```

```

min_loss = np.mean(val_loss)
best_model_q4 = copy.deepcopy(net)

if (epoch == 0) | ((epoch + 1) % 20 == 0):
    print('epoch {:03d}'.format(epoch), 'train_loss {:.8f}'.format(np.mean(train_loss)), 'val_loss {:.8f}'.format(np.mean(val_loss)))

average_train_loss.append(np.mean(train_loss))
average_val_loss.append(np.mean(val_loss))

torch.save(best_model_q4, file_path_q4)

# 绘制训练集和验证集损失曲线
plt.figure(figsize=(8, 6))
plt.plot(average_train_loss, label='train loss')
plt.plot(average_val_loss, label='val loss')
plt.legend()
plt.savefig(f'./result/q4_loss_cnn_new.png')
plt.show()

# 测试模型
with torch.no_grad():
    model = torch.load(file_path_q4)
    # print(model)

    # 预测测试数据
    for i, data in enumerate(test_loader, 0):
        inputs = data[0]
        inputs = inputs.to(device)
        inputs = inputs.float()

        # y_pred 为预测值
        y_pred = model(inputs)
        y_pred = P0*torch.exp(y_pred)

        y_pred = invert_scale(scaler_y,
                             y_pred.cpu().detach().numpy())

```

```

# 创建模型实例
net = MultiTaskModel(input_dim=input_dim,
                      output_dim=output_dim, shared_dim=shared_dim).to(device)

# 定义损失函数和优化器
loss_function = nn.MSELoss().to(device)
optimizer_net = torch.optim.Adam(net.parameters(),
                                 lr=learning_rate)
scheduler_net = StepLR(optimizer_net, step_size=step_size,
                       gamma=gamma)

min_epochs = 10
best_model_q4 = None
min_loss = 0.1
average_train_loss = []
average_val_loss = []
file_path_q4 = 'result/model_q4_mt_new.pth'

if os.path.exists(file_path_q4):
    print("MTNN 最佳模型文件存在，跳过执行模型训练。")
else:
    print("MTNN 最佳模型文件不存在，开始训练模型。")
    for epoch in range(num_epochs):
        net.train()
        train_loss = []
        for i, data in enumerate(train_loader, 0):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            inputs = inputs.float()
            labels = labels.float()
            inputs_train = torch.cat([inputs[:, :2], inputs[:, 3:]], dim=1) # 第2列是材料类型
            train_type = invert_scale(scaler_x,
                                      inputs.cpu().numpy())[:, 2]

            # 梯度清零
            optimizer_net.zero_grad()

            # 前向传播

```

```

y_pred = net(inputs_train, train_type)
y_pred = P0*torch.exp(y_pred)

loss = loss_function(y_pred, labels)

train_loss.append(loss.cpu().item())

# 更新梯度
loss.backward()

# 优化参数
optimizer_net.step() # 更新每个网络的权重

scheduler_net.step() # 调整学习率

net.eval()
with torch.no_grad():
    val_loss = []
    for i, data in enumerate(val_loader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device),
                    labels.to(device)
        inputs = inputs.float()
        labels = labels.float()
        inputs_valid = torch.cat([inputs[:, :2],
                                inputs[:, 3:]], dim=1)
        valid_type = invert_scale(scaler_x,
                                  inputs.cpu().numpy())[:, 2]

        # 前向传播
        y_pred = net(inputs_valid, valid_type)
        y_pred = P0*torch.exp(y_pred)

        # 计算损失
        loss = loss_function(labels, y_pred)
        val_loss.append(loss.cpu().item())

if epoch > min_epochs and np.mean(val_loss) < min_loss:
    min_loss = np.mean(val_loss)

```

```

best_model_q4 = copy.deepcopy(net)

if (epoch == 0) | ((epoch + 1) % 20 == 0):
    print('epoch {:03d}'.format(epoch), 'train_loss {:.8f}'.format(np.mean(train_loss)), 'val_loss {:.8f}'.format(np.mean(val_loss)))

average_train_loss.append(np.mean(train_loss))
average_val_loss.append(np.mean(val_loss))

torch.save(best_model_q4, file_path_q4)

# 绘制训练集和验证集损失曲线
plt.figure(figsize=(8, 6))
plt.plot(average_train_loss, label='train loss')
plt.plot(average_val_loss, label='val loss')
plt.legend()
plt.savefig(f'./result/q4_loss_mt_new.png')
plt.show()

# 测试模型
with torch.no_grad():
    model = torch.load(file_path_q4)
    # print(model)

    # 预测测试数据
    for i, data in enumerate(test_loader, 0):
        inputs = data[0]
        inputs = inputs.to(device)
        inputs = inputs.float()
        inputs_test = torch.cat([inputs[:, :2], inputs[:, 3:]], dim=1)
        test_type = invert_scale(scaler_x,
                                 inputs.cpu().numpy()[:, 2])

        # y_pred 为预测值
        y_pred = model(inputs_test, test_type)
        y_pred = P0*torch.exp(y_pred)

```

```

y_pred = invert_scale(scaler_y,
                      y_pred.cpu().detach().numpy())

# 创建模型实例
net1 = PCNN(input_dim=input_dim, output_dim=1).to(device)
net2 = PCNN(input_dim=input_dim, output_dim=1).to(device)
net3 = PCNN(input_dim=input_dim, output_dim=1).to(device)
net4 = PCNN(input_dim=input_dim, output_dim=1).to(device)

# 定义损失函数和优化器
loss_function = nn.MSELoss().to(device)
optimizer_net = torch.optim.Adam(list(net1.parameters()) +
                                  list(net2.parameters()) + list(net3.parameters()) +
                                  list(net4.parameters()), lr=learning_rate)
scheduler_net = StepLR(optimizer_net, step_size=step_size,
                       gamma=gamma)

min_epochs = 10
best_model_q4_net1 = None
best_model_q4_net2 = None
best_model_q4_net3 = None
best_model_q4_net4 = None
min_loss = 1e10
average_train_loss = []
average_val_loss = []

file_path_q4_net1 = 'result/model_q4_pcnn_net1_new.pth' #
    模型保存路径
file_path_q4_net2 = 'result/model_q4_pcnn_net2_new.pth'
file_path_q4_net3 = 'result/model_q4_pcnn_net3_new.pth'
file_path_q4_net4 = 'result/model_q4_pcnn_net4_new.pth'

# 模型输入[0温度, 1频率, 2磁芯材料, 3励磁波形, 4方差, 5极差,
# 6偏度, 7峰度, 8最大幅值, 9最大幅值频率点, 10总能量, 11频谱中心]
if os.path.exists(file_path_q4_net1):
    print("PCNN 最佳模型文件存在, 跳过执行模型训练。")
else:
    print("PCNN 最佳模型文件不存在, 开始训练模型。")
    for epoch in range(num_epochs):

```

```

net1.train()
net2.train()
net3.train()
net4.train()
train_loss = []
for i, data in enumerate(train_loader, 0):
    inputs, labels = data
    inputs, labels = inputs.to(device), labels.to(device)
    inputs = inputs.float()
    labels = labels.float()

    # 梯度清零
    optimizer_net.zero_grad()

    # 前向传播
    k_h = net1(inputs)
    b = net2(inputs)
    k_cl = net3(inputs)
    k_e = net4(inputs)

    k_h = k_h0 * torch.exp(k_h)
    b = b0 * torch.exp(b)
    k_cl = k_cl0 * torch.exp(k_cl)
    k_e = k_e0 * torch.exp(k_e)

    input_phy = invert_scale(scaler_x,
                             inputs.cpu().numpy())
    p_real = invert_scale(scaler_y, labels.cpu().numpy())

    f = torch.from_numpy(input_phy[:, 1]).to(device)
    B_m = torch.from_numpy(input_phy[:, 4]).to(device)
    p_real = torch.from_numpy(p_real).to(device)

    f = f.reshape(-1, 1)
    B_m = B_m.reshape(-1, 1)
    p_real = p_real.reshape(-1, 1)

    p_h = P_h(k_h, f, B_m, b)
    p_cl = P_cl(k_cl, f, B_m)

```

```

p_e = P_e(k_e, f, B_m)

p_total = p_h + p_cl + p_e

loss = loss_function(p_total, p_real)

train_loss.append(loss.cpu().item())

# 更新梯度
loss.backward()

# 优化参数
optimizer_net.step() # 更新每个网络的权重

scheduler_net.step() # 调整学习率

net1.eval()
net2.eval()
net3.eval()
net4.eval()
with torch.no_grad():
    val_loss = []
    for i, data in enumerate(val_loader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device),
        labels.to(device)
        inputs = inputs.float()
        labels = labels.float()

        # 前向传播
        k_h = net1(inputs)
        b = net2(inputs)
        k_cl = net3(inputs)
        k_e = net4(inputs)

        k_h = k_h0 * torch.exp(k_h)
        b = b0 * torch.exp(b)
        k_cl = k_cl0 * torch.exp(k_cl)
        k_e = k_e0 * torch.exp(k_e)

```

```

        input_phy = invert_scale(scaler_x,
                                  inputs.cpu().numpy())
        p_real = invert_scale(scaler_y,
                               labels.cpu().numpy())

        f = torch.from_numpy(input_phy[:, 1]).to(device)
        B_m = torch.from_numpy(input_phy[:, 4]).to(device)
        p_real = torch.from_numpy(p_real).to(device)

        f = f.reshape(-1, 1)
        B_m = B_m.reshape(-1, 1)
        p_real = p_real.reshape(-1, 1)

        p_h = P_h(k_h, f, B_m, b)
        p_cl = P_cl(k_cl, f, B_m)
        p_e = P_e(k_e, f, B_m)

        p_total = p_h + p_cl + p_e

        loss = loss_function(p_total, p_real)
        val_loss.append(loss.cpu().item())

    if epoch > min_epochs and np.mean(val_loss) < min_loss:
        min_loss = np.mean(val_loss)
        best_model_q4_net1 = copy.deepcopy(net1)
        best_model_q4_net2 = copy.deepcopy(net2)
        best_model_q4_net3 = copy.deepcopy(net3)
        best_model_q4_net4 = copy.deepcopy(net4)

    if (epoch == 0) | ((epoch + 1) % 30 == 0):
        print('epoch {:03d}'.format(epoch), 'train_loss
              {}'.format(np.mean(train_loss)), 'val_loss
              {}'.format(np.mean(val_loss)))

    average_train_loss.append(np.mean(train_loss))
    average_val_loss.append(np.mean(val_loss))

torch.save(best_model_q4_net1, file_path_q4_net1)

```

```

torch.save(best_model_q4_net2, file_path_q4_net2)
torch.save(best_model_q4_net3, file_path_q4_net3)
torch.save(best_model_q4_net4, file_path_q4_net4)

# 绘制训练集和验证集损失曲线
plt.figure(figsize=(8, 6))
plt.plot(average_train_loss, label='train loss')
plt.plot(average_val_loss, label='val loss')
plt.legend()
plt.savefig(f'./result/q4_loss_pcnn_new.png')
plt.show()

# 测试模型
with torch.no_grad():
    net1 = torch.load(file_path_q4_net1)
    net2 = torch.load(file_path_q4_net2)
    net3 = torch.load(file_path_q4_net3)
    net4 = torch.load(file_path_q4_net4)

    # 预测测试数据
    for i, data in enumerate(test_loader, 0):
        inputs = data[0]
        inputs = inputs.to(device)
        inputs = inputs.float()

        k_h = net1(inputs)
        b = net2(inputs)
        k_cl = net3(inputs)
        k_e = net4(inputs)

        k_h = k_h0 * torch.exp(k_h)
        b = b0 * torch.exp(b)
        k_cl = k_cl0 * torch.exp(k_cl)
        k_e = k_e0 * torch.exp(k_e)

        input_phy = invert_scale(scaler_x, inputs.cpu().numpy())

        f = torch.from_numpy(input_phy[:, 1]).to(device)
        B_m = torch.from_numpy(input_phy[:, 4]).to(device)

```

```

f = f.reshape(-1, 1)
B_m = B_m.reshape(-1, 1)

p_h = P_h(k_h, f, B_m, b)
p_cl = P_cl(k_cl, f, B_m)
p_e = P_e(k_e, f, B_m)

y_pred = p_h + p_cl + p_e
y_pred = y_pred.cpu().detach().numpy()

results_index = [15, 75, 97, 125, 167, 229, 270, 337, 347,
                 378]
results = []
for i in range(len(results_index)):
    results.append(y_pred[results_index[i]])

results = np.array(results)
# 打开或创建文件，这里以写入模式为例
with open('result/q4_results_mtnn_new.txt', 'w') as file:
    # 将数组转换为字符串，并写入文件
    for item in results:
        file.write("%s\n" % item) # 每个元素后换行

# 绘制 Voltage 预测图像
# 创建横坐标
data_num = len(y_pred)
Cyc_X = np.linspace(0, data_num, data_num)

# 统一线宽设置
LINEWIDTH = 1.5
TICK_LABELSIZE = 12

# 使用with语句确保图表资源管理
with plt.style.context({'axes.linewidth': LINEWIDTH}): #
    # 设置全局线宽
    fig, sub = plt.subplots(figsize=(8, 6)) #
    # 直接设置子图大小，减少后续调整
    sub.plot(Cyc_X, y_pred, linewidth=LINEWIDTH)

```

```

# 设置坐标轴刻度和标签
sub.tick_params(axis='both', which='major',
                labelsize=TICK_LABELSIZE)
sub.set_ylabel('P ($\mathbf{W/m}^3$)',
                fontsize=TICK_LABELSIZE)
sub.set_xlabel('Test Index', fontsize=TICK_LABELSIZE)
# sub.set_title(f'MT Prediction Result',
#               fontsize=TICK_LABELSIZE + 2)

# 明确图例位置
# sub.legend()

# 保存图表，路径可作为参数传入或配置
save_path = 'result/q4_prediction_mt_new.png'
plt.savefig(save_path, dpi=300, bbox_inches='tight') #
    确保所有内容都被保存

plt.show()

```

## A.5 第5问程序

question5.py

```

# 导入第三方库
import torch
import random
import numpy as np
import matplotlib.pyplot as plt
import os
from hyperopt import fmin, tpe, hp, Trials
from matplotlib import rcParams

# 设置全局字体为Times New Roman
rcParams['font.family'] = 'serif'
rcParams['font.serif'] = ['Times New Roman']

# 导入自定义库
from read_data_new import *
from models import *

```

```

# 设置随机数种子
def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True

setup_seed(27)

device = torch.device("cuda:0" if torch.cuda.is_available()
else "cpu")

# 建立字典
material_dict = {'材料1': 1, '材料2': 2, '材料3': 3, '材料4': 4}
wave_dict = {'正弦波': 1, '三角波': 2, '梯形波': 3}

# 读取训练数据
# [0温度, 1频率, 2磁芯材料, 3磁芯损耗, 4励磁波形, 磁通密度]
directory = f'data/'
file_name = 'train'
file_path_train = os.path.join(directory, file_name)
train_data = create_train_data(file_path_train)

# 转换第3,5列的数据
for row in train_data:
    row[2] = float(material_dict[row[2]])
    row[4] = wave_dict[row[4]]

# 提取磁通密度的统计学特征
B = train_data[:, 5:]
B_max = calculate_max(B).reshape(-1, 1)
train_data = np.concatenate((train_data[:, :5], B_max),
axis=1)

# 分离输入数据和标签数据
train_x = np.concatenate((train_data[:, :3], train_data[:, 4:]), axis=1)

```

```

train_y = train_data[:, 3].reshape(-1, 1)
train_x, valid_x, train_y, valid_y = \
    train_test_split(train_x, train_y, test_size=0.2,
                     random_state=420)

# 归一化训练数据
scaler_x, train_x_scaled = scale_train(train_x)
scaler_y, train_y_scaled = scale_train(train_y)

for k in np.arange(2.5, 5.5, 0.5):
    def function(params):
        with torch.no_grad():
            file_path_q4_net1 =
                'result/model_q4_pcnn_net1_new.pth'
            file_path_q4_net2 =
                'result/model_q4_pcnn_net2_new.pth'
            file_path_q4_net3 =
                'result/model_q4_pcnn_net3_new.pth'
            file_path_q4_net4 =
                'result/model_q4_pcnn_net4_new.pth'
            net1 = torch.load(file_path_q4_net1)
            net2 = torch.load(file_path_q4_net2)
            net3 = torch.load(file_path_q4_net3)
            net4 = torch.load(file_path_q4_net4)
            T = params['T']
            F = params['F']
            B = params['B']
            W = params['W']
            M = params['M']
            inputs = np.array([T, F, B, W, M])
            inputs = inputs.reshape(1, -1)
            inputs = scaler_x.transform(inputs)
            inputs = torch.tensor(inputs,
                                  dtype=torch.float32).to(device)
            k_h = net1(inputs.unsqueeze(0))
            b = net2(inputs.unsqueeze(0))
            k_cl = net3(inputs.unsqueeze(0))
            k_e = net4(inputs.unsqueeze(0))
            # 模型输出迭代初始值

```

```

k_h0 = 1
k_c10 = 1
k_e0 = 4
b0 = 2
k_h = k_h0 * torch.exp(k_h).item()
b = b0 * torch.exp(b).item()
k_c1 = k_c10 * torch.exp(k_c1).item()
k_e = k_e0 * torch.exp(k_e).item()

f = torch.tensor(F, dtype=torch.float32)
B_m = torch.tensor(B, dtype=torch.float32)
k_h = torch.tensor(k_h, dtype=torch.float32)
b = torch.tensor(b, dtype=torch.float32)
k_c1 = torch.tensor(k_c1, dtype=torch.float32)
k_e = torch.tensor(k_e, dtype=torch.float32)

p_h = P_h(k_h, f, B_m, b)
p_c1 = P_c1(k_c1, f, B_m)
p_e = P_e(k_e, f, B_m)

y_pred = p_h + p_c1 + p_e
result = y_pred.item() - k*B*F
return result

# 定义超参数范围
space = {
    'T': hp.uniform('T', 20, 100),
    'F': hp.uniform('F', 50000, 500000),
    'B': hp.uniform('B', 0, 1),
    'W': hp.uniform('W', 1, 3),
    'M': hp.uniform('M', 1, 4),
}

# 初始化 Trials 对象
trials = Trials()

# 优化超参数
best_params = fmin(
    fn=lambda params: function(params),

```

```

        space=space,
        algo=tpe.suggest,
        max_evals=3000,
        trials=trials
    )

# 输出最优超参数
def create_inputs(x):
    inputs = np.array(x)
    inputs = inputs.reshape(1, -1)
    inputs = scaler_x.transform(inputs)
    inputs = torch.tensor(inputs,
                          dtype=torch.float32).to(device)
    return inputs

def pred_values(inputs):
    with torch.no_grad():
        file_path_q4_net1 =
            'result/model_q4_pcnn_net1_new.pth'
        file_path_q4_net2 =
            'result/model_q4_pcnn_net2_new.pth'
        file_path_q4_net3 =
            'result/model_q4_pcnn_net3_new.pth'
        file_path_q4_net4 =
            'result/model_q4_pcnn_net4_new.pth'
        net1 = torch.load(file_path_q4_net1)
        net2 = torch.load(file_path_q4_net2)
        net3 = torch.load(file_path_q4_net3)
        net4 = torch.load(file_path_q4_net4)
        k_h = net1(inputs.unsqueeze(0))
        b = net2(inputs.unsqueeze(0))
        k_cl = net3(inputs.unsqueeze(0))
        k_e = net4(inputs.unsqueeze(0))
        # 模型输出迭代初始值
        k_h0 = 1
        k_c10 = 1
        k_e0 = 4
        b0 = 2
        k_h = k_h0 * torch.exp(k_h).item()

```

```

        b = b0 * torch.exp(b).item()
        k_cl = k_cl0 * torch.exp(k_cl).item()
        k_e = k_e0 * torch.exp(k_e).item()

        f = torch.tensor(F, dtype=torch.float32)
        B_m = torch.tensor(B, dtype=torch.float32)
        k_h = torch.tensor(k_h, dtype=torch.float32)
        b = torch.tensor(b, dtype=torch.float32)
        k_cl = torch.tensor(k_cl, dtype=torch.float32)
        k_e = torch.tensor(k_e, dtype=torch.float32)

        p_h = P_h(k_h, f, B_m, b)
        p_cl = P_cl(k_cl, f, B_m)
        p_e = P_e(k_e, f, B_m)

        y_pred = p_h + p_cl + p_e

    return y_pred

B = best_params['B']
F = best_params['F']
M = best_params['M']
T = best_params['T']
W = best_params['W']

input_1 = create_inputs([B, F, int(M), T, int(W)])
input_2 = create_inputs([B, F, int(M)+1, T, int(W)])
input_3 = create_inputs([B, F, int(M), T, int(W)+1])
input_4 = create_inputs([B, F, int(M)+1, T, int(W)+1])

min_p = min(pred_values(input_1),
            pred_values(input_2),
            pred_values(input_3),
            pred_values(input_4))

min_index =
    torch.argmin(torch.tensor([pred_values(input_1),
                               pred_values(input_2),
                               pred_values(input_3),
                               pred_values(input_4)]))

print("最佳 B:", B)

```

```

print("最佳 F:", F)
print("最佳 T:", T)
# 建立字典
material_dict = {1: '材料1', 2: '材料2', 3: '材料3', 4: '材料4'}
wave_dict = {1: '正弦波', 2: '三角波', 3: '梯形波'}
if min_index == 0:
    m = material_dict[int(M)]
    w = wave_dict[int(W)]
    print("最佳 M:", material_dict[int(M)])
    print("最佳 W:", wave_dict[int(W)])
elif min_index == 1:
    m = material_dict[int(M) + 1]
    w = wave_dict[int(W)]
    print("最佳 M:", material_dict[int(M)+1])
    print("最佳 W:", wave_dict[int(W)])
elif min_index == 2:
    m = material_dict[int(M)]
    w = wave_dict[int(W) + 1]
    print("最佳 M:", material_dict[int(M)])
    print("最佳 W:", wave_dict[int(W)+1])
elif min_index == 3:
    m = material_dict[int(M) + 1]
    w = wave_dict[int(W) + 1]
    print("最佳 M:", material_dict[int(M)+1])
    print("最佳 W:", wave_dict[int(W)+1])
print("最小磁芯损耗:", min_p.item())
print("最大传输磁能:", best_params['B']*best_params['F'])

# 保存内容
content = """
最佳 B: {}
最佳 F: {}
最佳 T: {}
最佳 M: {}
最佳 W: {}
最小磁芯损耗: {}
最大传输磁能: {}
"""

```

```
# 可以修改的定义语句
definition = f"以下是权重为 {k} 最佳参数和相关计算结果: "

formatted_content = content.format(B, F, T, m, w,
min_p.item(), best_params['B'] * best_params['F'])

# 写入文件
with open('result/q5_results.txt', 'a', encoding='utf-8'):
    as file:
        file.write(definition + '\n')
        file.write(formatted_content + '\n\n')

print("内容已保存到 q5_results.txt 文件中")
```