基于广义回归神经网络与帝国竞争算法的熔喷材料性能问题

摘要

本文研究插层熔喷非织造材料的性能问题，从宏微观角度分析数据变化规律，建立各向异性广义回归神经网络，采用拟牛顿算法确定超参数，最后基于分析结果建立双目标规划模型并利用帝国竞争算法求解。

针对问题一，从**宏观角度**进行插层前后对比分析以及指标相关性分析，从**微观角度**对影响程度进行灰色关联度量化分析。首先，本文从宏观的角度对插层前后的变化趋势进行分析，并且对插层率与结构变量和产品性能的指标变化率进行**相关性分析**。其中，通过**方差分析**和**图像分析**插层前后各指标的变化。本文分别**控制**两个工艺参数指标**变量**，从微观的角度对插层率与结构变量以及产品性能的相关性进行分析。最后，对固定工艺参数指标后得到的 **Pearson 相关系数**进行**灰色关联度量化分析**，得出插层率以及插层率自身的变化对相应指标的影响程度。其中，插层率影响程度最大的指标为**厚度**。对厚度变化率与插层率指标进行回归分析。

针对问题二，利用**拟牛顿法** (BFGS 算法) 改进各向异性广义回归神经网络模型预测结构变量数据。首先，本文对各项指标数据进行相对误差判断。本文对工艺参数与结构变量进行**双因素方差分析**。本文对本文利用 data3 中的指标数据建立**各向异性广义回归神经网络 (AGRNN) 预测模型**，并利用 **BFGS 算法**确定的**超参数**从而对模型进行优化。由此得到预测模型结合已知数据对表 1 中的结构变量进行预测。最后，本文对模型进行**十折交叉验证**和**残差正态检验**。

针对问题三，利用逐步回归法筛选自变量指标，建立偏最小二乘法回归模型，建立以**最小化过滤效率**为目标的规划模型，求解对应工艺参数。首先，本文对结构变量与产品性能各指标数据进行预处理，降低数据的相对误差。在对各指标数据进行 **Pearson 相关性分析**后，本文选择一系列非线性指标进行后续的回归分析。在指标选取过程中，本文通过**逐步回归法**对指标进行筛选，得到 **19 个**自变量指标。为了减弱指标间的多重共线性，本文建立**偏最小二乘法回归模型**，得到结构变量与产品性能之间的回归方程。此外，本文建立以最大化过滤效率为目标的**规划模型**，求解该目标下对应的工艺参数为**接收距离 19.743cm**，**热风速度 1857.3968r/min**，此时对应**最大过滤效率为 91.094%**。

针对问题四，建立双目标规划模型，利用**理想点法**转化为单目标规划确定权重求解，最终得到不同目标权重下的 Parero 最优解集，确定满足目标的工艺参数。本文建立**双目标规划模型**，并利用理想点法将双目标规划模型转变为单目标规划。调整权重后，利用**帝国竞争算法**求解得到不同 **Parero 最优解集**，确定同时满足过滤效率最高以及过滤阻力最小这两个目标的工艺参数数据为 **12.271cm**、**1825.937r/min**。

本文的优势在于： 1. 利用十折交叉验证与残差正态检验验证预测模型的准确性与可靠性；2. 利用理想点法确定目标权重，得到 Parero 最优解集，对问题的分析更深入全面。

关键字：方差分析　灰色关联量化模型　AGRNN　偏最小二乘回归　理想点法

目录

一、 问题重述

1.1 问题背景

熔喷非织造材料具有许多良好的性能，因而常常被用来生产口罩。然而熔喷非织造材料的弹性较差，为了改善熔喷非织造材料的蓬松性，科学家们通过在聚丙烯 (PP) 熔喷制备过程中将短纤维 (PET) 作为插层结构引入到熔喷非织造材料中，形成高蓬松性和高孔隙率的插层熔喷非织造材料。插层熔喷非织造材料整体为“Z”型结构，具有优良的吸音、保暖和过滤性能。

制备插层熔喷非织造材料的过程中存在许多交互影响的工艺参数，并且在气流短纤插层后更加复杂。因此，通过工艺参数决定结构变量，而由结构变量决定最终的产品性能也变得较为复杂。了解工艺参数与结构变量的关系以及结构变量与产品性能的关系更有助于插层熔喷织造材料的制备[10]。

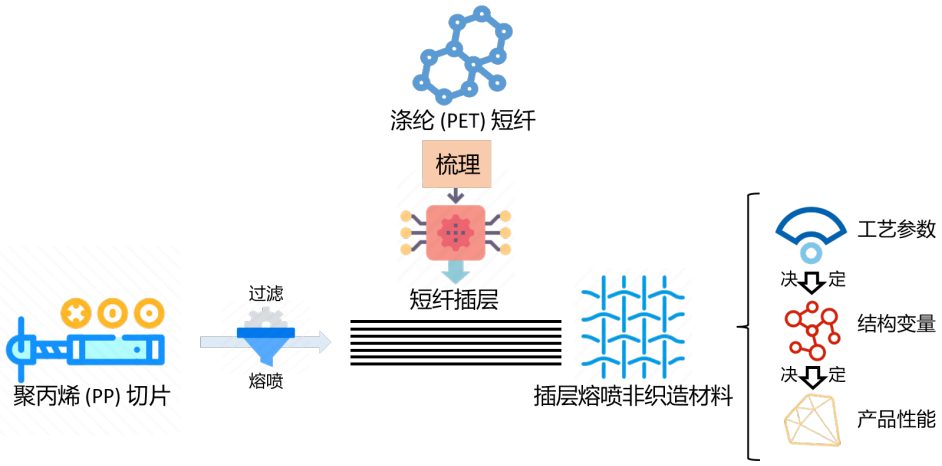


图 1　插层熔喷非织造材料背景图

1.2 问题提出

熔喷非织造材料具有许多良好的性能，因而常常被用来生产口罩。制备插层熔喷非织造材料的过程中存在许多交互影响的工艺参数，并且在气流短纤插层后更加复杂。因此，通过工艺参数决定结构变量，而由结构变量决定最终的产品性能也变得较为复杂。针对这些熔喷非织造材料的相关信息，我们对以下几个问题进行研究：

问题一：分析插层后结构变量、产品性能的变化规律，并考察插层率对其变化的影响。

问题二：分析工艺参数与结构变量的关系，得出预测的结构变量完成表 1。

问题三：分析结构变量与产品性能的关系以及各个结构变量之间、产品性能之间的关系。结合问题二，考察产品过滤效率达到最高时对于工艺参数的要求。

问题四：产品生产需要兼顾各方面的条件和要求。其中，包括对接受距离、热空气

速度、厚度、压缩回弹性率的要求。此外，为了防止熔喷非织造过滤材料因过滤阻力大使得大量颗粒堵塞孔隙而致使过滤效率迅速下降的现象发生，产品要同时保证过滤效率最高且过滤阻力最小。考察工艺参数为多少时能满足该目标。

二、 模型假设

（1）所有数据真实可靠。

（2）工艺参数与结构变量的联系以及结构变量与产品性能的联系并不受到插层率的影响。

（3）结构变量仅受到工艺参数的影响。

（4）产品性能仅受到结构变量的影响。

三、 符号说明

符号	含义	说明
a_1, a_2	接受距离、热空气速度	工艺参数对应指标
b_1, b_2, b_3	厚度、孔隙率、压缩回弹性率	结构变量对应指标
c_1, c_2, c_3	过滤阻力、过滤效率、透气性	产品性能对应指标
p_i	神经元传递函数	$p_i = \exp(\sum_{j=1}^{2} -(\frac{(x_j - x_{ij})^2}{2\sigma_j^2}))$
y_k	神经元 k 的输出对应结果第 k 个元素	

四、 问题一的模型建立与求解

4.1 问题一的描述与分析

问题一要求分析插层后结构变量、产品性能的变化规律，并考察插层率对其变化的影响。首先，本文从宏观的角度对插层前后的变化趋势进行分析，并且对插层率与结构变量和产品性能的指标变化率进行相关性分析。依据分析结果，发现工艺参数的变化对相关性分析的结果存在一定影响。

因此，本文分别**固定两个工艺参数指标**，从微观的角度对插层率与结构变量以及产品性能的相关性进行分析。对固定工艺参数指标后得到的相关系数进行灰色关联度量化分析，得出插层率以及插层率自身的变化对相应指标的影响程度。最后，对影响程度最大的指标进行回归拓展分析。

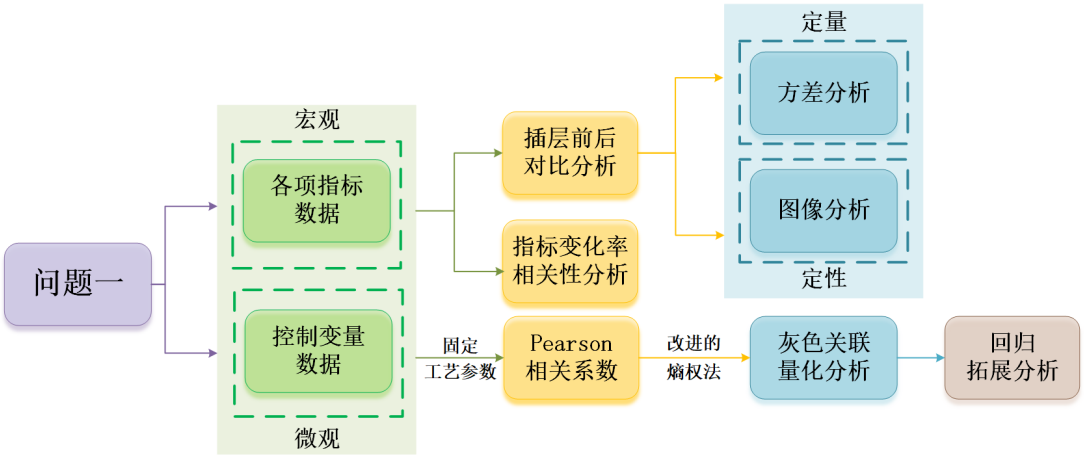


图 2　问题一思路图

4.2 各项指标数据宏观分析

本文从宏观的角度对插层后结构变量、产品性能的变化规律以及插层率对该变化的影响进行分析。首先，本文对插层前后的指标变化情况进行方差分析。为了尽量**消除工艺参数指标**带来的影响，本文选取结构变量、产品性能各个指标的变化率与插层率进行相关性分析，从而分析插层率对数据变化的影响。

4.2.1 插层前后对比分析

设工艺参数指标接受距离和热空气速度分别为 a_1、a_2。设结构变量指标厚度、孔隙率、压缩回弹性率分别为为 b_1、b_2、b_3。设产品性能指标过滤阻力、过滤效率、透气性分别为 c_1、c_2、c_3。在宏观分析中，本文首先对插层前后的结构变量、产品性能各项指标的变化进行**方差分析**。分析结果如下表1：

表 1　方差分析结果表

分析项	厚度 b_1	孔隙率 b_2	压缩回弹性率 b_3	过滤阻力 c_1	过滤效率 c_2	透气性 c_3
总离差	25.559	279.325	3203.729	12145.34	26296.94	2277936
偏 Eta 方	0.58	0.552	0.201	0.033	0.098	0.03
F 值	66.168	59.134	12.091	1.617	5.244	1.509
p 值 (双尾)	0.000***	0.000***	0.001***	0.21	0.026**	0.225

对方差分析结果进行分析，由方差分析中的 p 值可得厚度、孔隙率、压缩回弹性率以及过滤效率统计结果显著，即是否插层在该指标上存在显著差异，而是否插层在过滤

阻力和透气性上不存在显著差异。由 Eta 方的结果可知，厚度数据的差异有 58.0% 来源于不同组别间的差异，孔隙率数据的差异则有 55.2% 来源于不同组别间的差异，以此类推。

此外，本文结合图像对结构变量与产品性能各项指标的插层前后的对比与变化进行分析。下图3为结构变量厚度、孔隙变化率以及压缩回弹性率三项指标的插层前后数据对比图。其中，各指标从左至右的顺序为**插层率从小到大**排序的顺序。

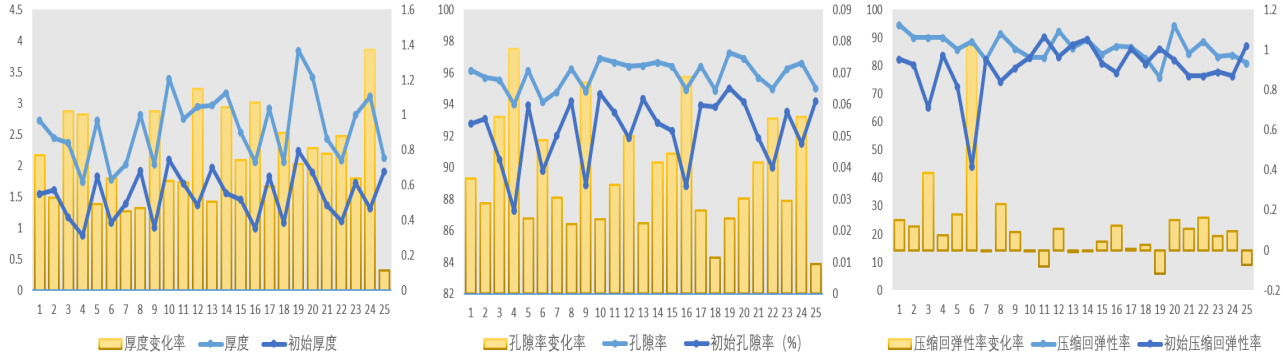


图 3　插层前后结构变量指标对比图

分析图像可知，插层熔喷法可以有效降低熔喷非织造材料的厚度和孔隙变化率，该结果印证了插层熔喷法对于熔喷非织造材料蓬松性较大问题的改善。而对于材料的压缩回弹性率，观察图像发现插层后该指标并没有显著的提升，仅在插层率较小时存在一定的提升，该结果也说明了适当的插层处理对于压缩回弹性率的增加是有利的，而插层率过大则会导致材料的蓬松性过差从而无法具备良好的压缩回弹性率。

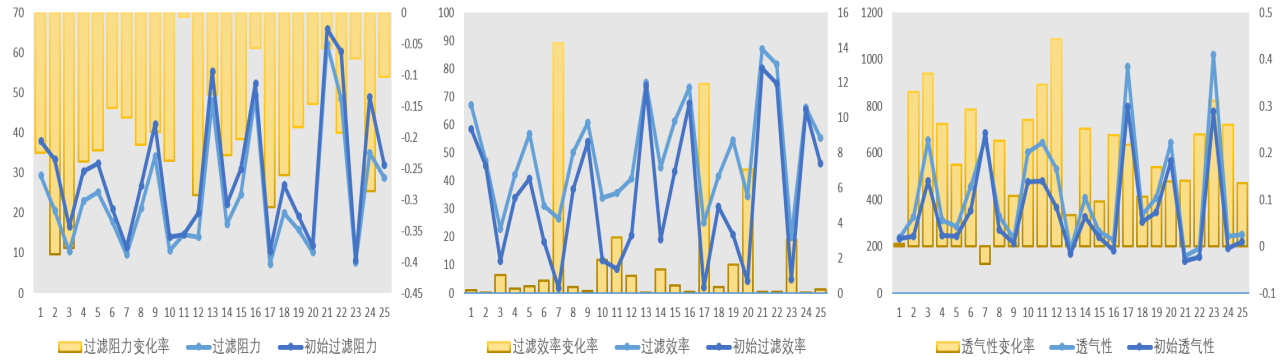


图 4　插层前后产品性能指标对比图

上图4为产品性能过滤阻力、过滤效率以及透气性三项指标的插层前后数据对比图。分析图像发现，插层处理后过滤阻力存在一定的减小，过滤效率和透气性基本都有一定提升。该结果说明了插层熔喷处理能够较好地提高产品性能。此外，观察图像可以发现插层率不同时产品性能的提升程度也各不相同。因此，在现实生产中，应该确定最有利于产品性能提升的插层率，对熔喷非织造材料进行插层处理。

4.2.2 指标变化率相关性分析

在对各指标数据的宏观分析中，本文还利用 **Pearson 相关系数**对插层率以及各指标变化率数据进行了相关性分析。Pearson 相关系数计算公式如下：

$$P_{x,y} = \frac{\mathrm{cov}(x,y)}{\sigma_x \sigma_y} = \frac{E[(x - x_i)(y - y_i)]}{\sigma_x \sigma_y} \tag{1}$$

其中，x、y 表示需要计算相关系数的两个指标数据对象。

对插层率和各项指标变化率进行 Pearson 相关性分析如下图：

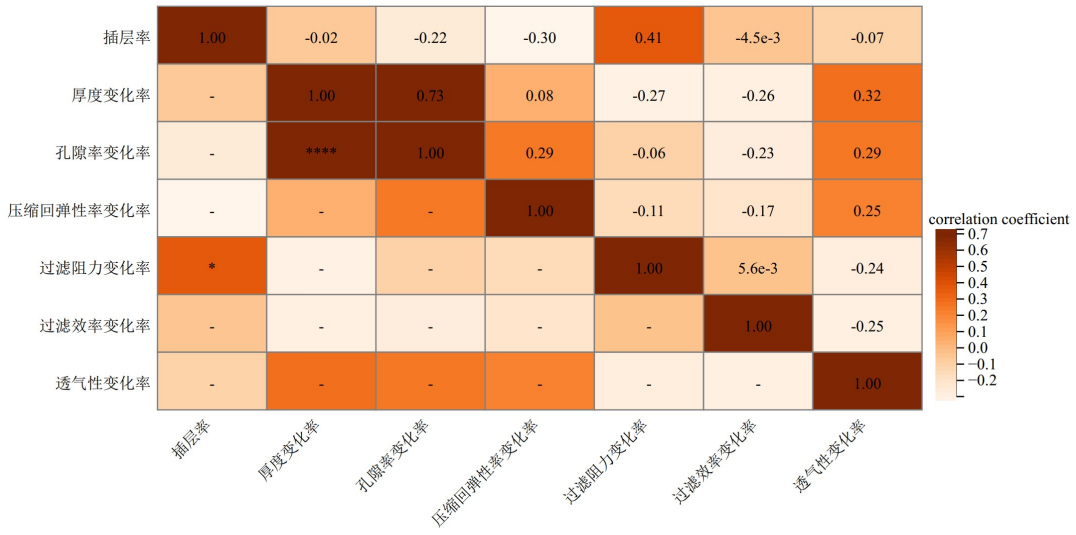


图 5　宏观相关性分析结果图

由图可知，插层率与其他六个指标变化率的相关性均较弱。该结果表明，插层率与结构变量和产品性能各项指标的相关性可能仍受到工艺参数的影响，即变化率指标的选取并未将工艺参数的影响完全消除。因此，本文对各项数据指标进行微观分析，即固定工艺参数后，再对数据进行相关性分析，得出插层率以及插层率自身的变化对相应指标的影响程度。

4.3 控制变量数据微观分析

基于宏观分析的结果，本文固定工艺参数后，再对不同情况下插层率与六项指标变化率的相关性进行分析。再利用所得相关系数建立灰色关联度量化模型得出插层率以及插层率自身的变化对相应指标的影响程度。

4.3.1 多层量化目标树的建立

由宏观分析中相关性分析的结果可知，插层率与结构变量和产品性能六项指标变化率的相关性较小，因此需要进一步消除工艺参数对结构变量和产品性能数据变化的影

响。分析工艺参数数据发现，接受距离存在 20cm、25cm、30cm、35cm、40cm 五种情况，热空气速度存在 800r/min、900r/min、1000r/min、1100r/min、1200r/min 五种情况，且各情况下数据个数相同。因此，对这十种工艺参数**分别进行固定**，求出每种情况下插层率与结构变量和产品性能对应六项指标的 Pearson 相关系数。

利用计算得到的各项指标相关系数建立多层量化目标树，本文将控制接受距离变量 a_1 和控制热空气速度变量 a_2 作为中间目标。对于控制接受距离变量 a_1 的中间目标，本文将其对应五种情况下，即接受距离为 20cm、25cm、30cm、35cm、40cm 时对应的插层率与结构变量以及产品性能六项指标的相关系数作为底层量化指标。而对于控制热空气速度变量 a_2 的中间目标，本文将其对应五种情况下，即热空气速度为 800r/min、900r/min、1000r/min、1100r/min、1200r/min 时对应的插层率与结构变量以及产品性能六项指标的相关系数作为其底层量化指标。由此得到多层量化目标树如下图6：

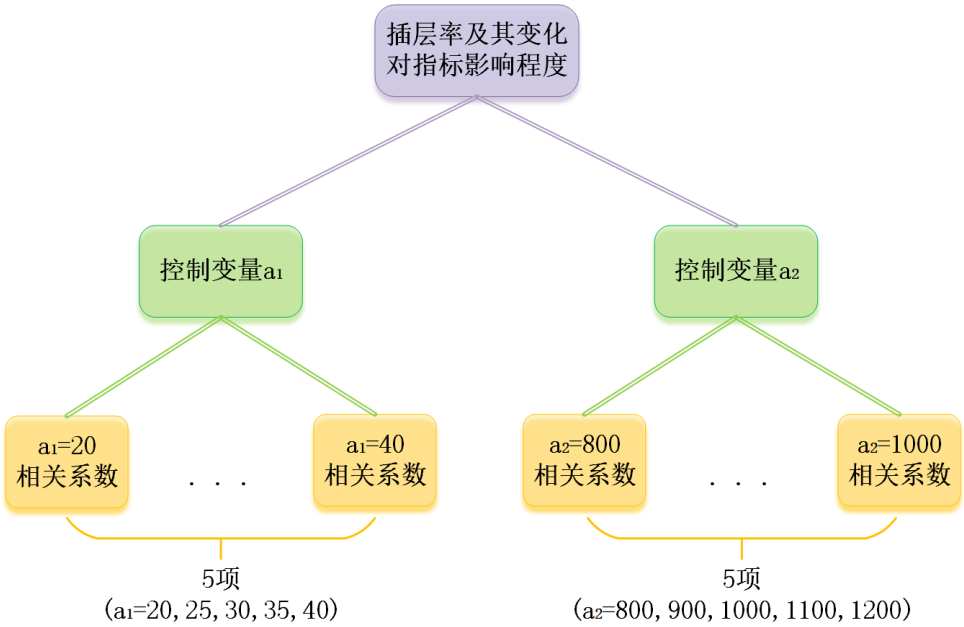


图 6　多层量化目标树图

4.3.2 改进的熵权法定权

得出不同工艺参数情况下对应的插层率与结构变量和产品性能对应六项指标的 Pearson 相关系数后，本文对相关系数进行**取绝对值处理**，从而仅考虑相关程度。本文选择改进的熵权法计算固定接受距离或热空气速度的各情况权重，以及固定接受距离和固定热空气速度两个中间目标的权重。

传统熵权法在所有熵值趋近于 1 时，会过度放大权重差距，赋权不合理。因此本文对熵权法中的权重计算进行改进，从而克服传统赋权确定的不合理性。改进的熵权法具体步骤如下[8]：

Step1：建立初始量化矩阵和指标数据标准化。针对本题量化固定接受距离参数和

热空气速度参数这两种情况，均分别对应 5 个量化指标。我们有 6 个量化结果，即插层率与 6 个结构变量和产品性能变化率指标的相关系数。对于固定接受距离参数情况的量化分析，$r_{i,j}(i=1,2,\cdots,6;j=1,2,\cdots,5)$ 是第 i 个结果中第 j 个量化指标，则初始量化矩阵为 $R=(r_{i,j})_{6\times5}$。由于各项指标量纲和单位不统一，因此对矩阵 M 进行标准化处理，得到标准化矩阵 $T=(t_{i,j})_{6\times5}$。依据下式对矩阵进行标准化：

$$t_{i,j}=\frac{m_{i,j}-m_{\min}}{m_{\max}-m_{\min}}. \tag{2}$$

Step2：计算各项指标的信息熵。第 j 项指标的信息熵为：

$$e_j=-\frac{1}{\ln 6}\sum_{i=1}^{n}p_{i,j}\ln p_{i,j}. \tag{3}$$

其中，$p_{i,j}$ 为第 j 项指标下第 i 个结果占该指标的比重。

Step3：确定各指标权重。第 j 项指标的权重为：

$$q_j=\begin{cases}(1-\overline{e}^{\alpha})q_j+\overline{e}^{\alpha}\overline{q_j}, & e_j<1\\ 0, & e_j=1\end{cases}. \tag{4}$$

其中，$\overline{q_j}$ 为指标 r_j 的平均权重，e_j 为对应的熵值，$\overline{e}$ 为全部不为 1 的熵权的平均值，α 为熵值指数，一般取评价方案的个数。

针对固定接受距离参数情况的量化分析，我们均有 6 个量化结果，即 6 个结构变量和产品性能对应指标，5 个量化指标。对于固定热空气速度参数的情况以及中间目标权重的确定，仍重复上述过程。设控制变量 a_1、a_2 时对应的十组相关系数为 $g_1,g_2,\cdots,g_10$。最终得到各层指标权重结果如下7：

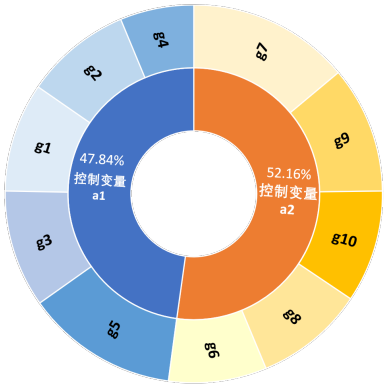


图 7　各层指标权重图

4.3.3 灰色关联度量化

结合改进的熵权法得到的各层指标的权重，建立灰色关联度量化模型对插层率以及插层率自身变化对各个指标的影响程度进行量化分析。根据灰色关联决策的理论，以各个结果指标向量与相对最优结果指标向量的关联度作为量化影响程度的准则。

针对本题量化插层率以及插层率自身变化对各个指标的影响程度结果，其相对最优结果均为每个指标的最小值 $u_0=(f_{01},f_{02},\cdots,f_{06})$，规范化处理后为 $u_0=(1,1,\cdots,1)$，结果 u_i 的量化指标 v_j 与相对最优结果 u_0 的量化指标 v_j 之间的灰色关联度为：

$$\gamma_{ij}=\frac{\xi \max_i |f_{ij}-1|}{|f_{ij}-1|+\xi \max_i |f_{ij}-1|} \tag{5}$$

其中，$\xi\in(0.1)$ 为分辨系数，一般取作 0.5。

设得到的灰色关联度矩阵分别为 γ_1 和 γ_2，固定接受距离参数情况对应的指标权重向量为 W_1，固定热空气速度参数对应的指标权重向量为 W_2，则固定接受距离参数时的量化结果为 $\gamma_1 W_1^T$，固定热空气速度参数时的量化结果为为 $\gamma_2 W_2^T$。对于插层率对各项指标的影响程度结果，结合中间目标权重同样利用上述灰色关联度模型进行量化。

最终，计算得到插层率及插层率自身的变化对各项指标的影响程度量化结果如下图8：

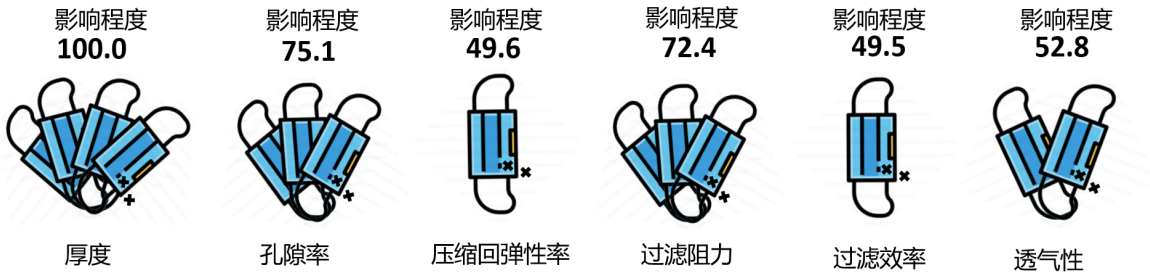
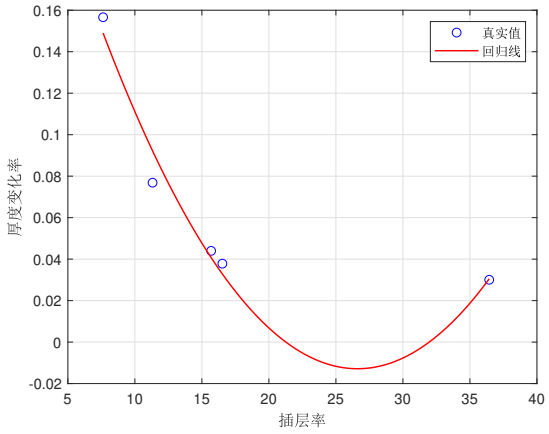


图 8　插层率对各项指标影响量化结果图

由图可知，插层率以及插层率自身的变化对厚度的影响最大，这是由于插层技术直接改善了熔喷非织造材料较大的蓬松性，使得材料厚度大幅降低。其次，插层率及其变化对过滤阻力与孔隙率也有着较大影响，即改变插层率对材料起到一定程度的降低过滤阻力以及增大孔隙率的效果。而由量化结果可知，插层率对过滤效率、压缩回弹性率以及透气性的影响程度较小。总的来说，插层率及其自身变化对结构变量对应指标的影响较为显著，对产品性能对应指标的影响较弱。

4.4 回归拓展分析

由上述灰色关联度量化分析发现插层率及其自身变化对厚度的影响最大，本文针对厚度变化率与插层率的指标数据进行回归拓展分析，回归结果见附录。通过回归可以发现回归中一半的 R_2 大于 90%，说明对厚度变化率的回归效果良好，验证了灰色关联量化模型的可行性。



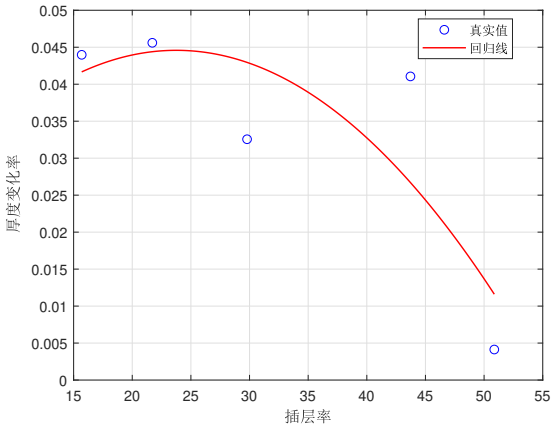


图 9　$a_2 = 900$ 时厚度变化率回归图　　**图 10　$a_1 = 30$ 时厚度变化率回归图**

由图9图10可知插层率与指标变化率在不同工艺参数下存在不同的关联，应该针对不同工艺参数分别进行回归分析，这也印证了本文**微观分析**的合理性。

五、 问题二的模型建立与求解

5.1 问题二的描述与分析

问题二要求分析工艺参数与结构变量的关系，得出预测的结构变量完成表 1。本文利用 data3 中的指标数据建立各向异性广义回归神经网络 (AGRNN) 预测模型，并利用 BFGS 算法确定的超参数从而对模型进行优化。由此得到预测模型利用已知数据对表 1 中的结构变量进行预测。最后，本文对模型进行十折交叉验证和残差检验。其中，利用 S-W 正态检验进行残差验证。

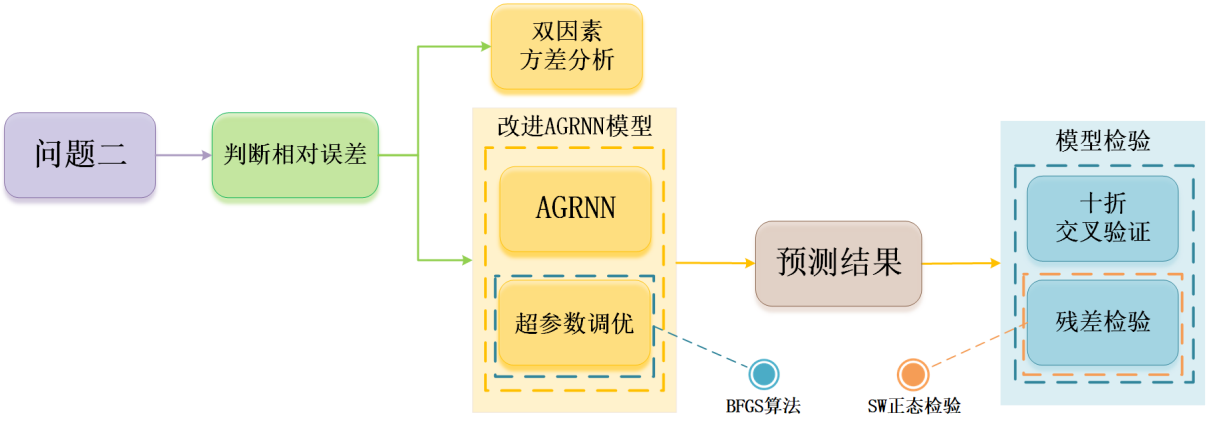


图 11　问题二思路图

5.2 相对误差判断

本文对结构变量厚度 b_1、孔隙率 b_2、压缩回弹性率 b_3 的相对误差进行计算。计算得到厚度 b_1 的**最大相对误差**为 0.870%，孔隙率 b_2 的**最大相对误差**为 0.858%，压缩回弹性率 b_3 的**最大相对误差**为 0.865%。参考罗兹-哥特里法[11] 中对于最大相对误差的规定，结构变量各项指标的数据误差较小，不需要进行数据预处理。

5.3 双因素方差分析

首先，为确定工艺参数与结构变量之间的关系，本文将接收距离和热空气速度作为自变量，对三项结构变量数据指标进行双因素分析。将孔隙率作为因变量时的方差分析结果如下表:

表 2　双因素方差分析结果 (孔隙率)

差异源	平方和	df	均方	F	p
Intercept	689461.2	1	689461.2	6128292	0.000**
接收距离	27.017	4	6.754	60.035	0.000**
热风速度	15.339	4	3.835	34.086	0.000**
Residual	7.425	66	0.113		

由上表可知，利用双因素分研究接收距离和热风速度对孔隙率的影响关系，接收距离呈现出显著性 ($F = 60.035, p = 0.000 < 0.05$)，说明主效应存在，接收距离会对孔隙率产生差异关系。热风速度呈现出显著性 ($F = 34.086, p = 0.000 < 0.05$) 说明主效应存在，热风速度会对孔隙率产生差异关系。

此外，因变量为厚度以及压缩回弹性率的情况与上述情况类似。最终得出结论：工艺参数各指标会对结构变量各项指标产生差异关系，即工艺参数的变化会对结构变量产生影响。后续本文选择建立预测模型对工艺参数与结构变量的关系进行进一步分析。

5.4 用于材料生产优化的改进 AGRNN 模型

分析题目数据发现，data3 中提供的工艺参数与结构变量对应指标的样本数据较少，且部分数据起伏较大。在预测方法中，径向基神经网络较适用此种情况。各向异性广义回归神经网络 (Anisotropic Generalized Regression Neural Network,AGRNN) 是径向基神经网络的一种，具有很强的非线性映射能力和学习速度，样本数据少时预测效果很好，

还可以处理不稳定数据。因此，本文选择各向异性广义回归神经网络 (AGRNN) 对结构变量进行预测。

AGRNN 由四层构成，分别为输入层、模式层、求和层和输出层。AGRNN 的网络结构如下图12：

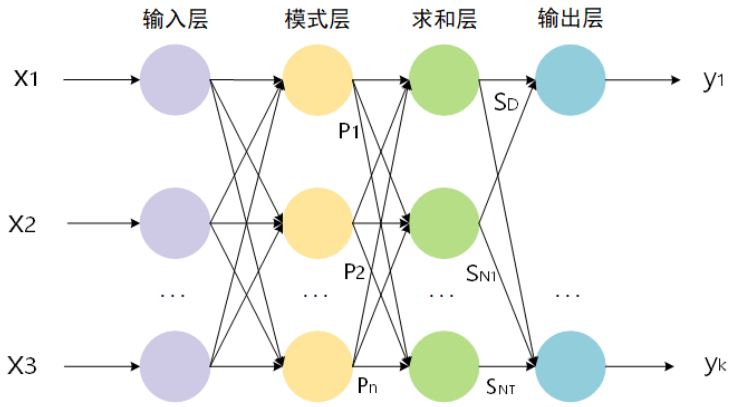


图 12　各向异性广义回归神经网络结构图

输入层：神经元数目表示学习样本中输入向量的维数，本文中表示工艺参数接受距离 a_1 与热空气速度 a_2。

模式层：模式层神经元数目即为学习样本数量 2，权重设为 1，模式层神经元传递函数如下：

$$p_i = \exp(\sum_{j=1}^{2} -(\frac{(x_j - x_{ij})^2}{2\sigma_j^2})) \quad i = 1, 2; j = 1, 2. \tag{6}$$

其中，x_j 表示带宽第 j 维时的输入变量，x_{ij} 为带宽第 j 维时第 i 个神经元对应的学习样本，σ_1 和 σ_2 是本文建立 AGRNN 模型中的超参数[2]。

求和层：在不同情况下结合模式层计算结果分别对神经元进行算术求和与加权求和，即有：

$$S_D = \sum_{i=1}^{2} p_i; \quad S_{N_k} = \sum_{i=1}^{2} y_{ik} p_i, \quad k = 1, 2, \cdots, k_0. \tag{7}$$

输出层：输出神经元数目等于学习样本中输出向量维数 k_0，各神经元将求和层输出对象相除，得到神经元 k 的输出对应估计结果 $\hat{y}(x)$ 的第 k 个元素：

$$y_k = \frac{S_{N_k}}{S_D}, \quad k = 1, 2, \cdots, k_0. \tag{8}$$

5.5 BFGS 算法改进 AGRNN 模型

平滑参数 σ_1 和 σ_2 是本文建立的 AGRNN 模型对应的超参数，较高的 σ_1 和 σ_2 值会使数据过于平滑，而较低的 σ_1 和 σ_2 值会导致数据过拟合，因此选择最优的 σ_1 和 σ_2

对预测效果十分重要。本文选择拟牛顿法 (BFGS 算法) 对 σ_1 和 σ_2 进行超参数调优，最大限度地利用现有的小样本数据集，利用最优的输入输出组合和最优的 σ_1 和 σ_2 建立 AGRNN 网络。

BFGS 算法有效利用了有限的数据，并且评估结果能够尽可能接近模型在测试集上的表现，可以作为模型优化的指标使用[4]。AGRNN 混合 BFGS 算法可以有效搜索最优超参 σ_1 和 σ_2。BFGS 算法的搜索流程见伪代码如下：

Algorithm 1 Broyden Fletcher Goldfarb Shanno Algorithm

1: **Initialize** /* Given parameters $\delta \in (0,1)$, $\sigma \in (0,0.5)$, initial point $x_0 \in R^n$, termination error $0 < \varepsilon \ll 1$. The initial symmetric positive define matrix B_0 (usually taken as $G(x_0)$ or identity matrix I_n). Let $k := 0$ */

2: calculate $g_k = \bigtriangledown f(x_k)$.

3: If $\parallel g_k \parallel \leq \omega$, stop the calculation and output X_k as the approximate minimum point;

4: Solve the system of linear equation to obtain d_k: $B_k d = -g_k$;

5: Let m_k be the smallest nonnegative integer m that satisfies the following inequality: $f(x_k + \delta^m d_k) \leq f(x_k) + \sigma\delta^m g_k^T d_k$. Let $\alpha_k = \delta^{m_k}$, $x_{k+1} = x_k + \alpha_k d_k$;

6: B_{k+1} is determined by the correction formula;

7: Let $k := k + 1$, and then turn to step 1;

8: iteration ← iteration + 1;

9: until meet the conditions.

10: **end**

5.6 求解结果与分析

本文利用上述 BFGS 算法对 AGRNN 模型进行超参数调优，得到针对三项数据指标的最优超参数如下表：

表 3　最优超参数结果表

	厚度	孔隙率	压缩回弹性率
σ_1	0.09264742	0.37992253	0.2346002
σ_2	0.12055773	0.39232725	0.29283169
R^2	0.995	0.909	0.974

根据上述最优超参数得到针对厚度、孔隙率、压缩回弹性率的三种预测模型，基于题目所给表中的接收距离与热风速度数据即可对这三种指标进行预测。本文结合样本数

据建立基于 BFGS 算法改进的 AGRNN 模型，对表 1 中结构变量数据预测结果如下表：

表 4　问题二的结果表

接收距离 (cm)	热风速度 (r/min)	厚度 mm	孔隙率（%）	压缩回弹性率（%）
38	850	2.903	96.275	85.613
33	950	2.808	96.250	87.580
28	1150	2.814	96.264	86.870
23	1250	2.609	95.932	85.925
38	1250	3.516	96.702	84.260
33	1150	3.111	96.483	86.534
28	950	2.506	95.687	88.272
23	850	2.019	94.597	87.378

观察表中结果可知，在接收距离与热风速度均较大时，材料的结构变量表现较差，而接受距离与热风速度均较小时，材料的结构变量表现较好。接收距离与热风速度的变化均对厚度、孔隙率以及压缩回弹性率这三项指标产生影响，也印证了工艺参数与结构变量间存在着一定的关系[7]。

5.7 模型检验

模型检验是确定模型的正确性、有效性和可信性的研究与测试过程。本文通过十折交叉检验和残差检验对改进的 AGRNN 模型进行检验。

5.7.1 十折交叉检验

交叉验证是一种统计学上将数据样本切割成较小子集的实用方法，因此本文选择该方法对模型的预测精度进行验证。其中，十折交叉验证较为常用，此方法的优势在于同时重复运用随机产生的子样本进行多次训练和验证，每次的结果验证一次[5]。

本文将 data3 中厚度、孔隙率与压缩回弹性率对应数据用作训练和验证数据。样本数为 75，随机选取 68 组结果分布较为均匀的数据作为训练样本，剩下 7 组数据作为验证样本进行 10 折交叉验证。在十折交叉验证中，10 次的结果的准确率的平均值作为对算法精度的估计，由此得到三项数据指标对应的准确率如下表：

表 5　十折交叉验证结果表

结构变量	厚度	孔隙率	压缩回弹性率
准确率均值	99.90%	68.30%	90.10%
准确率标准差	0.001	0.366	0.096

由上表知，本文针对该问题建立的改进的 AGRNN 模型对三项数据指标的预测过程中，在厚度与压缩回弹性率中均具有较高的准确率。在孔隙率中的预测准确率不佳，本文分析可能是因为数据量较少且波动较大导致的。

5.7.2 残差检验

本文通过对厚度、孔隙率与压缩回弹性率回归得到的标准化残差得到的残差进行 S-W 正态性检验，从而验证模型的可靠性。

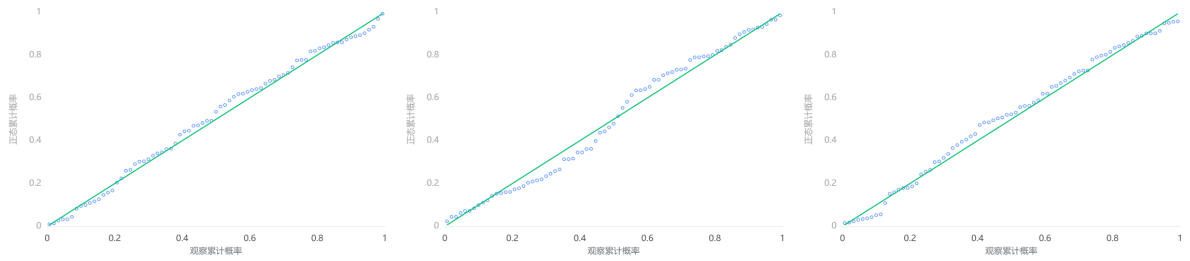


图 13　回归标准化残差的正态 P-P 图

根据验证结果可知，厚度的残差数据样本显著性 P 值为 0.616，孔隙率残差数据样本显著性 P 值为 0.155，压缩回弹性率残差数据样本显著性 P 值为 0.068。三项指标残差数据均不呈现显著性，不能拒绝原假设，即数据满足正态分布。因此，厚度、孔隙率和压缩回弹性率的残差数据均为**白噪声数据**，表明该回归模型具有良好的预测效果及模型性能。

六、 问题三的模型建立与求解

6.1 问题三的描述与分析

问题三要求分析结构变量与产品性能的关系以及各个结构变量之间、产品性能之间的关系。结合问题二，考察产品过滤效率达到最高时对于工艺参数的要求。首先，本文

对结构变量与产品性能各指标数据进行预处理，降低数据的相对误差。在对各指标数据进行 Pearson 相关性分析后，本文选择一系列非线性指标进行后续的回归分析。

在指标选取过程中，本文通过逐步回归法对指标进行筛选，得到 19 个自变量指标。为了减弱指标间的多重共线性，本文建立偏最小二乘法回归模型，对结构变量与产品性能的关系以及各个结构变量之间、产品性能之间的关系进行分析。此外，本文建立以最大化过滤效率为目标的规划模型，求解该目标下对应的工艺参数。

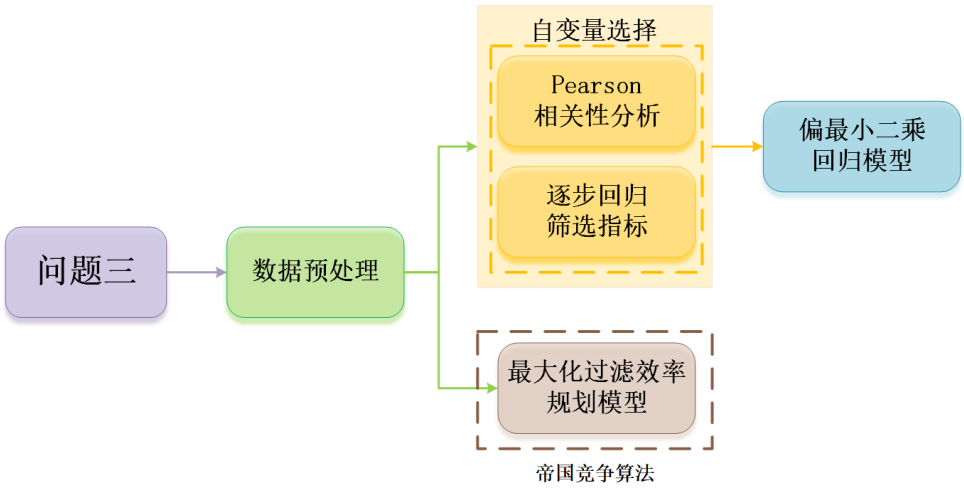


图 14　问题三思路图

6.2 准备工作

6.2.1 数据预处理

本问题要求结合结构变量以及产品性能对应的指标数据进行分析。本文对结构变量与产品性能的指标数据进行初步分析发现，同一重复实验组中数据间的相对误差较大，可能会对后续的模型建立以及模型效果造成一定影响。因此，本文对各项指标数据进行预处理，从而降低数据的相对误差，对数据中的离群值进行处理。数据预处理流程如下图15。数据预处理结果见附录。

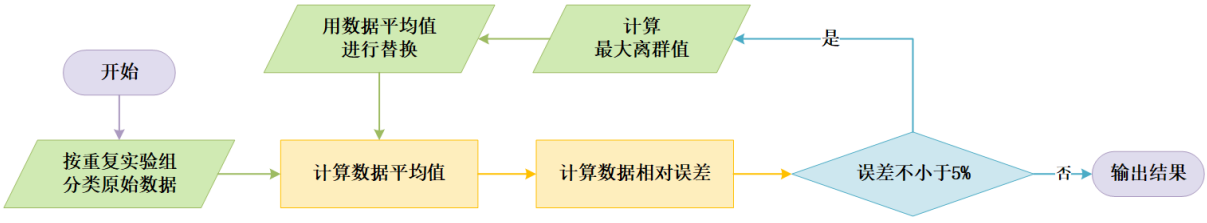


图 15　数据预处理流程图

6.2.2 模型的选择

问题三要求分析结构变量与产品性能的关系，即利用结构变量数据指标对产品性能相关指标进行预测。为了更好的选择与本题数据特征相符的模型，本文首先对结构变量与产品性能对应的各项指标即 b_1、b_2、b_3、c_1、c_2、c_3 进行 Pearson 相关性分析。分析结果如下图：

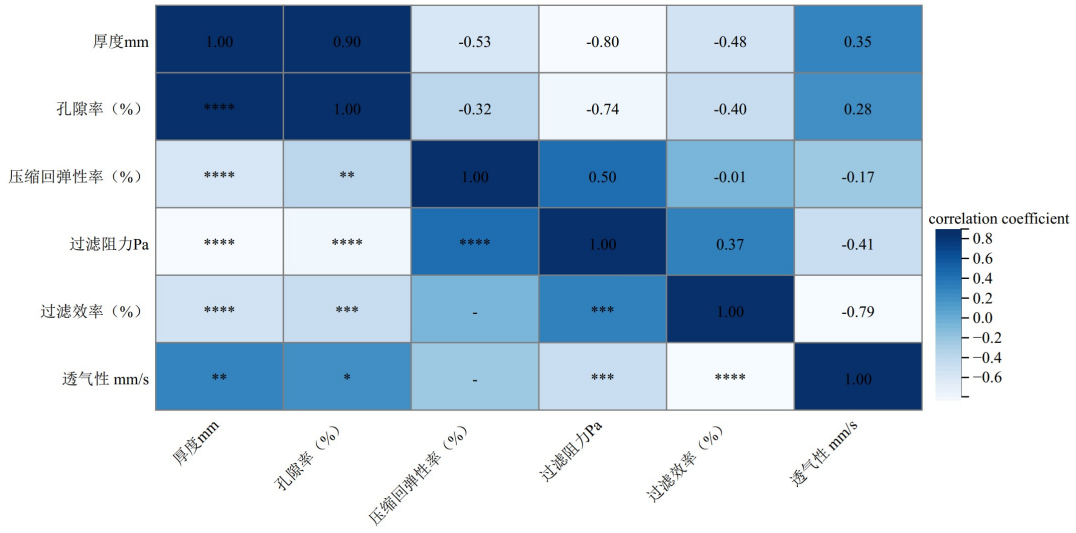


图 16　结构变量与产品性能指标相关分析结果图

由图可知，分析各结构变量指标之间的相关系数发现，**厚度与孔隙率**的相关性较强，而其他结构变量指标间相关性较弱。分析各产品性能指标间的相关系数发现，**过滤阻力与过滤效率**的相关性较强，而其他产品性能指标间相关性较弱。

观察上图，结构变量与产品性能各指标间的相关系数大部分较小，又由于本文选用 Pearson 相关性分析，则由此可知各指标数据间线性相关性不强。因此，后续针对本问题选择预测模型时，本文选择非线性指标进行回归分析。

6.3 指标的确定

6.3.1 指标的初步选择

基于上述 Pearson 相关性分析，本文基于 b_1、b_2、b_3 对数据进行负指数处理，确定以下非线性指标进行后续的回归分析 $(i, j, k = 1, 2, 3)$：

1. 负一次幂：$\frac{1}{b_i}$.
2. 负二次幂交互项：$\frac{1}{b_i b_j} \quad 1 \le i \le j \le 3$.
3. 一次幂与负一次幂交互项：$\frac{b_i}{b_j} \quad i \ne j$.
4. 一次幂与负二次幂交互项：$\frac{b_k}{b_i b_j} \quad i \le j \quad k \ne i, j$.

6.3.2 逐步回归指标筛选

初步选择 24 个非线性指标后，为降低指标间的多重共线性，本文利用逐步回归对指标进行筛选。逐步回归是一种线性回归模型自变量选择方法，其基本思想是将变量一个一个引入，引入的条件是其偏回归平方和经验是显著的。最终得到回归模型中所有变量对因变量都是显著的。

本文将过滤阻力、过滤效率和透气性作为因变量分别进行逐步回归，取三个因变量对应结果的公共部分，选择自变量指标如下：b_1、b_2、b_3、$\frac{1}{b_1}$、$\frac{1}{b_2}$、$\frac{b_1}{b_2}$、$\frac{b_1}{b_3}$、$\frac{b_2}{b_3}$、$\frac{b_3}{b_1}$、$\frac{b_3}{b_2}$、$\frac{1}{b_1^2}$、$\frac{1}{b_3^2}$、$\frac{1}{b_1b_2}$、$\frac{1}{b_1b_3}$、$\frac{1}{b_2b_3}$、$\frac{b_1}{b_2^2}$、$\frac{b_1}{b_3^2}$、$\frac{b_2}{b_1^2}$、$\frac{b_1}{b_2b_3}$.

最终确定以上包括结构变量本身以及非线性组合的 19 个自变量，进行后续的偏最小二乘回归。

6.4 偏最小二乘法模型建立与求解

由于筛选后指标数量仍较大，本文建立偏最小二乘法回归模型进行后续的预测分析以减弱指标间的多重共线性。本文将上述结构变量 19 个相关指标作为自变量，产品性能指标 c_1、c_2、c_3 作为因变量建立偏最小二乘法模型[3]。设自变量观测数据矩阵为 $B=(b_{ij})_{75\times19}$，因变量观测数据矩阵为 $C=(c_{ij})_{75\times3}$。

Step1：数据标准化：将各指标 b_{ij} 转化为指标值 b_{ij}^*，有：

$$b_{ij}^*=\frac{b_{ij}-u_j^{(1)}}{s_j^{(1)}},\quad i=1,2,\cdots,75;j=1,2,3. \tag{9}$$

其中，$u_j^{(1)}=\frac{1}{75}\sum_{i=1}^{75}b_{ij}$，$s_j^{(1)}=\sqrt{\frac{1}{75-1}\sum_{i=1}^{75}\left(b_{ij}-u_j^{(1)}\right)^2}, j=1,2,3$，即 $u_j^{(1)}$、$s_j^{(1)}$ 为第 j 个自变量 x_j 的样本均值和样本标准差。即称 x_j^* 为标准化变量，满足下式：

$$x_j^*=\frac{x_j-u_j^{(1)}}{s_j^{(1)}},\quad j=1,2,3. \tag{10}$$

同理可得标准化指标值 c_{ij}^* 以及对应标准化变量满足下式：

$$y_j^*=\frac{y_j-u_j^{(2)}}{s_j^{(2)}},\quad j=1,2,3. \tag{11}$$

Step2：求相关系数矩阵：计算标准化处理后的 19 个自变量与三个因变量之间的相关系数矩阵，根据相关系数矩阵对各自变量与因变量的相关性进行初步判断。完整的相关系数表格见附录。

Step3：分别提取自变量组和因变量组的成分。本文共提取 8 个主成分，取前五个作为自变量组和因变量组的出成分。

Step4：求解两个成分对时标准化指标变量与成分变量之间的回归方程。

Step5：求解因变量组与自变量组之间的回归方程：将之前得到的 u_i 代入 Step3 中 y_j^* 的回归方程，得到标准化指标变量之间的回归方程。之后，将标准化变量 y_j^*、x_j^* 分别还原成原始变量 y_j、x_j，得到回归方程。

6.5 最大化过滤效率规划模型

问题三要求研究当工艺参数为多少时，产品的过滤效率将会达到最高。由此，本文建立以最大化过滤效率为目标的规划模型如下：

$$\max P_2(G(a)). \tag{12}$$

其中，P 表示通过偏最小二乘回归得到的结构变量与过滤效率之间的函数关系，G 表示问题二分析的工艺参数与结构变量之间的函数关系。

该规划模型约束条件为指标数据自身的数据特征需要满足的实际情况：各指标数据均不小于 0，百分制数据默认不大于 100%.

6.6 帝国竞争算法求解规划模型

本文利用帝国竞争算法对上述规划模型进行求解，算法伪代码如下[1]：

Algorithm 2 Function Imperialist Competitve Algorithm(x)

Initialize /* set multiple initial national solutions. $X_n, n = 1, \cdots, n_{max}$。Select N solutions with the best quality from them as colonial countries of N empires, $X_1, X_2, \cdots, X_N$, and distribute them to national colonies according to the proportion of national power, thus obtaining N empires $E_1, E_2, \cdots, E_N$ */
iteration = 1; **Repeat for** $i \in [1, N]$ **do**
    Empire Assimilate(E_i)
    Colonies Revolve(E_i)
    Empire Posses(E_i)
    Caculate New Cost(E_i)
end
Empire Unite($E_1, E_2, \cdots, E_N$) EmpireCompetition($E_1, E_2, \cdots, E_N$)
EmpireCompetition($E_1, E_2, \cdots, E_N$)
iteration ← iteration + 1; Make a new iteration.
until There is only one empire left or the number of iterations has been reached.

6.7 求解结果与分析

本文利用上述偏最小二乘回归模型得到结构变量与产品性能之间的关系表现为回归方程如下：

$$\begin{cases} c_1 = -285011.95 - 20752.97b_1 - 1490.70b_2 + 2729.19b_3 + \cdots, \\ c_2 = -2434448.73 - 226172.29b_1 - 25636.35b_2 + 28223.76b_3 + \cdots, \\ c_3 = 20730981.70 - 5501309.17b_1 + 400319.34b_2 - 391072.86b_3 + \cdots. \end{cases} \tag{13}$$

其中，三个回归方程的完整自变量即系数见附录。

此外，本文还对各项指标进行了最小二乘回归，将最小二乘回归效果与偏最小二乘回归效果相比较如下表：

表 6　PLS 回归与 OLS 回归效果表

	PLS 回归		OLS 回归	
	R^2	$R^2_{adjusted}$	R^2	$R^2_{adjusted}$
过滤阻力 Pa	0.937	0.917	0.937	0.917
过滤效率（%）	0.846	0.796	0.838	0.786
透气性 mm/s	0.790	0.722	0.761	0.684

由表中的 R^2 与 $R^2_{adjusted}$ 结果可以发现，偏最小二乘回归模型在过滤效率和透气性的预测效果优于传统的最小二乘回归模型。下图17图表示数据预处理后的过滤效率数据与其原始数据，图18表示预测后的过滤效率数据与真实数据的曲线。由 PLS 回归效果图可以发现，预测得到的结果与真实值较为接近，验证了回归模型的准确性。

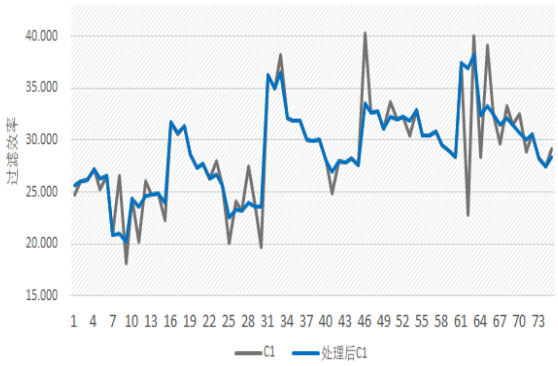


图 17　数据预处理效果图

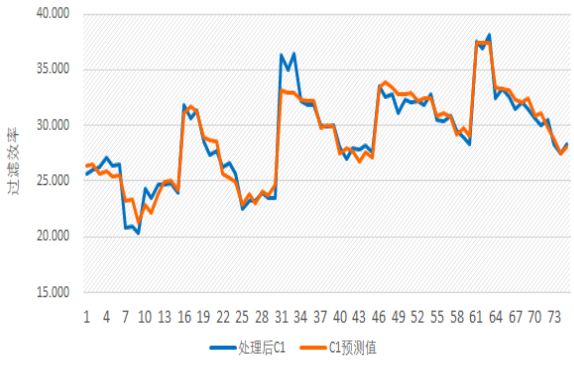


图 18　PLS 回归效果图

此外，利用帝国竞争算法求解以过滤效率最大化为目标的规划模型得到最优解的迭代过程如下图19：

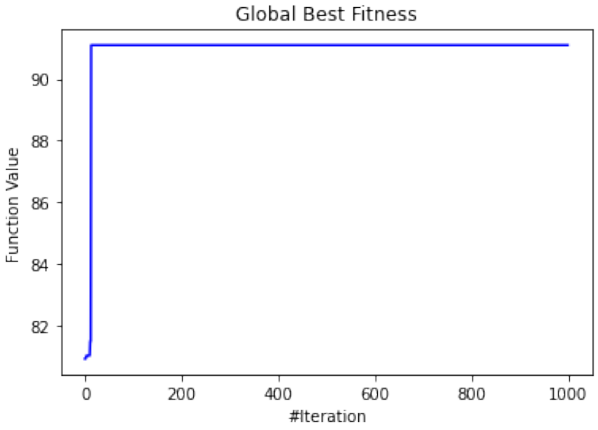


图 19　规划模型求解迭代曲线图

对以过滤效率最大化为目标的规划模型进行求解得到材料工艺参数的结果为：接收距离 19.743cm，热风速度 1857.3968r/min。此时结构变量结果为：厚度 2.333mm、孔隙率 95.886%、压缩回弹性率 85.179%。达到过滤效率最大化的目标，过滤效率达到 91.094%。

七、 问题四的模型建立与求解

7.1 问题四的描述与分析

问题四要求综合考虑各项指标的限制，考察能够满足材料同时具有高过滤效率与小过滤阻力两个目标的工艺参数。本文建立双目标规划模型，并利用理想点法将双目标规划模型转变为单目标规划，最终得到 Parero 最优解集，并结合不同目标权重对 Parero 最优解集进行分析，确定同时满足过滤效率最高以及过滤阻力最小这两个目标的工艺参数数据。

7.2 产品性能双目标规划模型

本文建立以过滤效率最高和过滤阻力最小为目标的双目标规划模型，并基于题目内容以及相关数据特征确定该规划模型的约束条件[9]。

基于问题二的相关分析可以得到工艺参数与结构变量之间的关系，记为 $G(a)=b$，则有：

$$b_j = G_j(a), \quad j = 1, 2, 3. \tag{14}$$

其中，a 表示工艺参数指标。

而基于问题三可以得到结构变量与产品性能之间的关系，记为 $P(b)=c$，则有：

$$c_k = P_k(b), \quad k=1,2,3. \tag{15}$$

其中，b 表示结构变量指标。

由上述关系可以得到工艺参数与产品性能之间存在关系如下：

$$c_k = P_k(G_j(a)), \quad j=1,2,3; k=1,2,3. \tag{16}$$

其中，a 表示工艺参数指标。

目标函数

首先，本问题需保证过滤阻力最小化，该目标函数为：

$$\min P_1(G(a)). \tag{17}$$

此外，本文还需保证过滤效率最大化，则有目标函数：

$$\max P_2(G(a)). \tag{18}$$

结合过滤阻力和过滤效率，得到产品性能双目标规划模型的总目标函数为：

$$\min Z = (P_1(G(a)), -P_2(G(a))). \tag{19}$$

约束条件

由题可知接收距离 a_1 不大于 100cm，则有：

$$0 < a_1 \leq 100. \tag{20}$$

由题可知热空气速度不大于 2000r/min，则有：

$$0 < a_2 \leq 2000. \tag{21}$$

此外，按照应用的要求，厚度尽量不要超过 3mm，则有：

$$0 < G_1(a) \leq 3. \tag{22}$$

基于生产应用的要求，压缩回弹性率尽量不要低于 85%，则有：

$$G_3(a) \geq 85. \tag{23}$$

综上所述，该产品性能双目标规划模型建立如下：

$$\min Z = (P_1(G(a)), -P_2(G(a))).$$

$$\begin{cases} 0 < a_1 \leq 100, \\ 0 < a_2 \leq 2000, \\ 0 < G_1(a) \leq 3, \\ G_3(a) \geq 85. \end{cases} \tag{24}$$

7.3 理想点法转化单目标规划

建立上述产品性能双目标规划模型后，本文利用理想点法确定目标权重，将双目标规划模型转化为单目标规划模型[6]。

先分别求解上述 2 个单目标优化问题，令

$$P_m^* = \begin{cases} \min\limits_{a\in\Omega} P_1(a), \\ \min\limits_{a\in\Omega}(-P_2(a)). \end{cases} \tag{25}$$

$$\begin{cases} \min\limits_{a} \sum\limits_{m=1}^{2} \omega_m (P_m(a) - P_m^*)^2, \\ \text{s.t. } a \in \Omega \end{cases}. \tag{26}$$

其中，权重 $\omega_m \geq 0, m = 1, 2, \sum\limits_{m=1}^{2} \omega_m = 1$。

该单目标规划模型的最优解 a^* 即为原双目标规划模型的非支配解。求解该单目标该规划模型得到 Parero 最优解集。

7.4 模型求解

首先，理想点法中先对两个单目标优化问题分别进行求解，该求解过程中，本文结合问题三中建立的单目标优化模型的求解过程进行求解分析。

之后，本文利用帝国竞争算法对双目标规划模型进行求解。

7.5 求解结果与分析

首先，基于理想点法，本文对两个单目标问题进行分别求解，下图20图21为两个单目标问题的求解迭代曲线。根据理想点法求出工艺参数结果对应两点为：[32.691, 1053.067]、[12.271, 1825.937]. 对应最小过滤阻力为 23.853Pa，最大过滤效率 91.93%

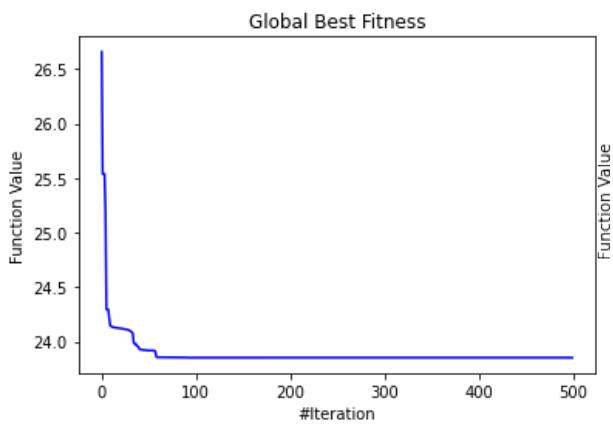


图 20　目标 1 优化求解迭代图

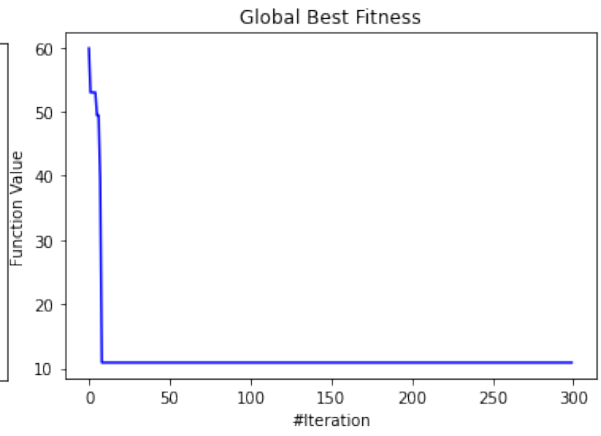


图 21　目标 2 优化求解迭代图

利用帝国竞争算法，结合上述理想点法求解结果得出不同目标权重下双目标规划模型 Parero 最优解集如下图：

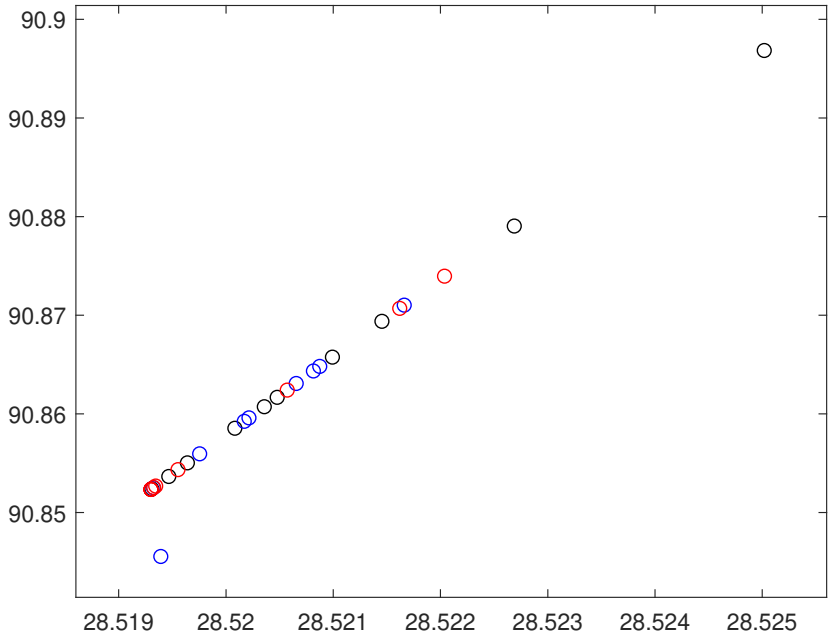


图 22　不同权重最优解图

图中红色为权重 0.4、0.6，蓝色为权重 0.5，0.5，黑色为权重 0.6、0.4。可以看出调节权重后理想点分布并未受到太大影响，在 28.5 和 90.8 之间浮动，可以认为两个目标之间存在一定正相关。

八、 模型评价

8.1 模型总结

本文通过单因素方差分析，灰色关联量化模型反应插层率对指标本身及其变化率的影响，利用 AGRNN 广义回归神经网络建立工艺参数与结构变量的关系，建立基于逐步回归的偏最小二乘回归模型反应结构变量与产品性能的关系，建立双目标规划模型设计最佳工艺参数，最后综合分析结果，提出有针对性的意见。

8.2 模型优缺点分析

优点：

（1）第一问从多方面入手，基于宏观和微观用控制变量的方法对工艺参数进行控制，并引入灰色关联量化模型让结果的展示更为全面；

（2）本文在第二问中利用在非线性和小样本表现优异的各向异性广义回归神经网络，并用十折交叉验证和残差正态检验来完善模型；

（3）在第三问利用迭代均值法改善数据，再用逐步回归和偏最小二乘法回归有效的降低了数据中的多重共线性，有效的提升了预测的精确程度；

（4）在第四问建立基于理想点法的双目标优化，并通过调整权重对规划进行更为详细的讨论，最终给出合理意见。

缺点：

（1）在问题二中的神经网络模型只单纯引入了工艺参数的指标，并没有考虑工艺参数中的交互影响；

（2）问题三模型中虽然利用偏最小二乘法消除了一定的多重共线性，但引入的指标中仍存在多重共线性和过拟合的问题。

8.3 模型改进与展望

首先，本文通过单因素方差分析，灰色关联量化模型反应插层率对指标本身及其变化率的影响，可以将方差和相关系数的量化结果带入之后的模型中对模型进行改进。AGRNN 广义回归神经网络模型中可以引入工艺参数间的交互影响使模型更加完善。在基于逐步回归的偏最小二乘回归模型中，可以利用 Lasso 回归对变量进行删选，使多重共线性进一步降低。规划模型可以对目标的关系进行更深层次的探讨，使结果更具有现实意义。

参考文献

[1] Salam Karim Yasir Mahdi Ahmed B Jade Catalan Opulencia Maria Fakhriddinovich Uktamov Khusniddin Thaeer Hammid Ali Abdalkareem Jasim Saade, Mireya Romero Parra Rosario. Optimization of doubly-fed induction generator (dfig)based wind turbine to achieve maximum power generation with imperialist competitive algorithm (ica). *[J]. Science progress*, 105(3), 2022.

[2] Federico Amato, Fabian Guignard, Philippe Jacquet, and Mikhail Kanevski. On feature selection using anisotropic general regression neural network. 10 2020.

[3] Lin Z. Groves K. et al Guan, T. Sparse functional partial least squares regression with a locally sparse slope function. *Stat Comput*, 32(30), 2022.

[4] Wang B. Hu W Li, X. A modified nonmonotone bfgs algorithm for unconstrained optimization. 01 2020.

[5] Saha Sunil Roy Jagabandhu. Ensemble hybrid machine learning methods for gully erosion susceptibility mapping: K-fold cross validation approach. *[J]. Artificial Intelligence in Geosciences*, 03 2022.

[6] 康艳，伊丽，龚家国. 基于分期设权理想点法的水文模型参数多目标优化. *[J]*. 天津大学学报*(*自然科学与工程技术版*)*, 54(05):458–467, 2021.

[7] 张小鹏，钱晓明. 玻璃纤维棉毡及插层熔喷的工艺研究进展. *[J]*. 化工新型材料, 50(01):287–291, 2022.

[8] 翟芸，胡冰，施端阳. 基于改进 ahp-熵权法的雷达装备可靠性评估指标赋权方法. *[J/OL]*. 现代防御技术, 08(10):1–10, 2022.

[9] 邹志伟. 插层熔喷气流场模拟及生产工艺参数的优化. *[D]*. 天津工业大学, 2020.

[10] 韩玲，胡梦缘，马英博，郝栋连. 医用非织造口罩材料及其新技术的研究现状. *[J]*. 西安工程大学学报, 34(02):20–25, 2020.

[11] 黄晓. 罗兹-哥特里法脂肪含量的测定. *[J]*. 广东化工, 48(07):172–173, 2021.

附录 A　支撑材料清单

- **图表**

-厚度变化率与插层率回归结果表

-偏最小二乘回归成分矩阵

-偏最小二乘回归因子载荷系数表

-偏最小二乘回归模型系数表

- **代码**

-问题一 matlab 源程序

-问题二 python 源程序

-问题三 matlab 源程序

-问题三 python 源程序

-问题四 python 源程序

附录 B　图表

厚度变化率与插层率回归结果表

a	b	c	R 方
5.09E-05	-0.004364583	0.119321793	0.996766632
#######	0.002130829	0.019276098	0.679992197
0.002627	-0.081118604	0.600113145	0.843967266
0.001786	-0.061945045	0.578582239	0.947290495
#######	-0.000656085	0.082462946	0.660611272
0.000449	-0.023903569	0.305244227	0.969691619
2.08E-05	-0.003889818	0.129417283	0.57705078
8.12E-05	-0.00813359	0.209915535	0.968616394
0.000123	-0.008413381	0.173306008	0.701367444
0.000934	-0.046382488	0.572339176	0.96793605

偏最小二乘回归成分矩阵

变量	因子 1	因子 2	因子 3	因子 4	因子 5
厚度 mm	0.265	-0.179	0.351	0.161	0.09
孔隙率（%）	0.234	-0.047	-0.204	0.413	0.006
压缩回弹性率（%）	-0.118	-0.452	0.104	0.021	-0.244
1/b1	-0.253	0.039	0.123	0.222	-0.31
1/b2	-0.233	0.04	0.225	-0.388	0.077
b1/b2	0.265	-0.179	0.356	0.131	0.074
b2/b3	0.196	0.316	-0.166	0.166	-0.093
b1/b3	0.262	-0.141	0.339	0.156	-0.122
b3/b2	-0.196	-0.323	0.186	-0.162	-0.176
b3/b1	-0.253	0.002	0.138	0.224	-0.395
1/b1^2	-0.241	-0.044	0.391	0.486	0.441
1/b3^2	0.119	0.448	-0.101	-0.036	-0.251
1/b1b2	-0.251	0.032	0.15	0.219	-0.21
1/b2b3	-0.024	0.482	0.03	-0.273	0.003
1/b1b3	-0.252	0.078	0.106	0.217	-0.238
b2/b1^2	-0.242	-0.039	0.371	0.48	0.34
b1/b2^2	0.264	-0.179	0.362	0.099	0.058
b1/b3^2	0.259	-0.106	0.328	0.15	-0.346
b1/b2b3	0.262	-0.139	0.343	0.127	-0.139
过滤阻力 Pa	-4.302	0.899	-0.163	1.164	0.019
过滤效率（%）	-1.552	0.544	-0.927	-0.238	0.107
透气性 mm/s	-3.116	0.737	-0.45	0.59	0.199

偏最小二乘回归因子载荷系数表

变量	因子 1	因子 2	因子 3	因子 4	因子 5
厚度 mm	0.253	-0.033	0.252	0.015	0.183
孔隙率（%）	0.236	-0.146	-0.441	0.501	0.006
压缩回弹性率（%）	-0.145	-0.521	-0.111	-0.018	-0.246
1/b1	-0.251	0.098	0.073	0.186	-0.239
1/b2	-0.235	0.148	0.452	-0.488	0.056
b1/b2	0.253	-0.03	0.272	-0.017	0.177
b2/b3	0.217	0.325	-0.109	0.24	-0.199
b1/b3	0.253	0.01	0.261	0.017	-0.166
b3/b2	-0.218	-0.324	0.128	-0.24	-0.269
b3/b1	-0.253	0.06	0.072	0.179	-0.531
1/b1^2	-0.246	0.129	0.235	0.296	0.321
1/b3^2	0.146	0.519	0.121	0.016	-0.307
1/b1b2	-0.251	0.103	0.103	0.167	-0.164
1/b2b3	0.002	0.617	0.394	-0.284	0.061
1/b1b3	-0.248	0.138	0.074	0.193	0.049
b2/b1^2	-0.247	0.126	0.216	0.304	0.241
b1/b2^2	0.252	-0.026	0.293	-0.052	0.172
b1/b3^2	0.252	0.048	0.269	0.018	-0.514
b1/b2b3	0.253	0.014	0.279	-0.014	-0.174
过滤阻力 Pa	-0.235	0.005	-0.074	0.043	-3.985
过滤效率（%）	-0.106	0.211	-0.914	-0.641	-2.611
透气性 mm/s	0.088	-0.047	0.526	0.322	6.8

偏最小二乘回归模型系数表

	过滤阻力	过滤效率	透气性	过滤阻力 Pa	过滤效率	透气性
常数	-285011.951	-2434448.73	20730981.7	0	0	0
厚度 mm	-20752.965	226172.287	-5501309.17	-2422.718	9965.219	-30868.779
孔隙率	-1490.698	-25636.348	400319.34	-302.263	-1961.895	3901.505
压缩回弹性率	2729.186	28223.757	-391072.86	820.883	3203.966	-5653.751
1/b1	119824.694	-4567311.61	68203824.9	2231.637	-32104.259	61054.356
1/b2	14650550.7	253792517	-3995981715	325.276	2126.674	-4264.337
b1/b2	3439307.64	31915409.3	5477377.84	4018.825	14075.157	307.632
b2/b3	180242.024	2993546.66	-38225696.9	927.877	5816.278	-9458.46
b1/b3	420624.425	-66162582.2	939729986	596.174	-35392.859	64019.488
b3/b2	-143059.621	-2477767.74	39747536.3	-595.69	-3893.933	7955.07
b3/b1	-715.935	27172.063	-398446.174	-1188.455	17023.813	-31791.411
1/b1^2	-623.562	-2759.963	-491714.195	-9.843	-16.443	-373.084
1/b3^2	-516372624	-1.3957E+10	1.6494E+11	-483.419	-4931.377	7422.097
1/b1b2	-551849.245	-17923717.4	144835493	-112.431	-1378.217	1418.308
1/b2b3	314548178	4887765284	7060152862	130.638	766.154	140.937
1/b1b3	-4619904.9	207312637	-3047632594	-968.179	16397.337	-30698.424
b2/b1^2	17.163	97.945	4999.161	25.237	54.356	353.32
b1/b2^2	-216533416	-2519075380	1.3446E+10	-2528.801	-11103.391	7547.959
b1/b3^2	-51086228.9	2170074474	-3.0512E+10	-881.299	14129.24	-25300.126
b1/b2b3	85606953.5	1571669130	-2.325E+10	1217.16	8433.823	-15888.569

附录 C　问题一--matlab 源程序

```
%%第一问
%数据预处理

%第一列 插层率
```

```
%第二、三列 a1 a2 工艺参数
%第四、五、六列 b1 b2 b3 结构变量
%第七、八、九 c1 c2 c3 产品性能
clc;clear
load('x_s')
load('x_cge_s')
chaceng=x(:,1);
cc1=zeros(5,7);%%a1
cc2=zeros(5,7);
cc3=zeros(5,7);
cc4=zeros(5,7);
cc5=zeros(5,7);
k=1;%计数器

a1=x(:,2);
a2=x(:,3);%a指标
b1=x(:,4);
b2=x(:,5);%b指标
b3=x(:,6);
b1_new=x_cge(:,4);
b2_new=x_cge(:,5);%b插层值
b3_new=x_cge(:,6);
b1_cge=x_cge(:,4)-x(:,4);%%变化了多少
b2_cge=x_cge(:,5)-x(:,5);
b3_cge=x_cge(:,6)-x(:,6);
b1_lv=b1_cge./x(:,4);
b2_lv=b2_cge./x(:,5);%变化率
b3_lv=b3_cge./x(:,6);
 k1=1;

 %% 定a1
for i=1:25
   for j=1:6
 if a1(i)==20

    k=ceil(k1/6);
    cc1(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j);
    cc1(k,1)=chaceng(i);
     k1=k1+1;

 end
   end
end

 k1=1;
cor1=corr(cc1);
cor(:,1)=cor1(2:7,1);
```

```
for i=1:25
   for j=1:6
 if a1(i)==25

    k=ceil(k1/6);
    cc2(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j);

    cc2(k,1)=chaceng(i);
     k1=k1+1;

 end
   end
end

 k1=1;
cor2=corr(cc2);
cor(:,2)=cor2(2:7,1);

for i=1:25
   for j=1:6
 if a1(i)==30

    k=ceil(k1/6);
    cc3(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j);
    cc3(k,1)=chaceng(i);
     k1=k1+1;

 end
   end
end

 k1=1;
cor3=corr(cc3);
cor(:,3)=cor3(2:7,1);

for i=1:25
   for j=1:6
 if a1(i)==35

    k=ceil(k1/6)
    cc4(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j)
    cc4(k,1)=chaceng(i)
     k1=k1+1;

 end
   end
end
```

```
cor4=corr(cc4)
cor(:,4)=cor4(2:7,1)

 k1=1;

for i=1:25
   for j=1:6
 if a1(i)==40

    k=ceil(k1/6);
    cc5(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j);
    cc5(k,1)=chaceng(i);
     k1=k1+1;

 end
   end
end
cor5=corr(cc5);
cor(:,5)=cor5(2:7,1);



 huigui1=[ cc1(:,2) cc2(:,2) cc3(:,2) cc4(:,2) cc5(:,2) ];
reg1=[cc1(:,1) cc2(:,1) cc3(:,1) cc4(:,1) cc5(:,1)];

y1=cc1(:,2);
x1=cc1(:,1);
[fitresult, gof]=createFit(x1,y1);
 k1=1;

%% 定a2
for i=1:25
   for j=1:6
 if a2(i)==800

    k=ceil(k1/6);
    cc1(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j);
    cc1(k,1)=chaceng(i);
     k1=k1+1;

 end
   end
end

 k1=1;
cor1=corr(cc1);
cor(:,6)=cor1(2:7,1);
```

```
for i=1:25
   for j=1:6
 if a2(i)==900

    k=ceil(k1/6);
    cc2(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j);
    cc2(k,1)=chaceng(i);
     k1=k1+1;

 end
   end
end

 k1=1;
cor2=corr(cc2);
cor(:,7)=cor2(2:7,1);

for i=1:25
   for j=1:6
 if a2(i)==1000

    k=ceil(k1/6);
    cc3(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j);
    cc3(k,1)=chaceng(i);
     k1=k1+1;

 end
   end
end

 k1=1;
cor3=corr(cc3);
cor(:,8)=cor3(2:7,1);

for i=1:25
   for j=1:6
 if a2(i)==1100

    k=ceil(k1/6)
    cc4(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j)
    cc4(k,1)=chaceng(i)
     k1=k1+1;
 end
   end
end
cor4=corr(cc4)
cor(:,9)=cor4(2:7,1)
```

```
 k1=1;

for i=1:25
   for j=1:6
 if a2(i)==1200

    k=ceil(k1/6);
    cc5(k,j+1)=(x_cge(i,j+3)-x(i,j+3))./x(i,j);
    cc5(k,1)=chaceng(i);
     k1=k1+1;

 end
   end
end
cor5=corr(cc5);
cor(:,10)=cor5(2:7,1);
 k1=1;

huigui2=[ cc1(:,2) cc2(:,2) cc3(:,2) cc4(:,2) cc5(:,2) ]
reg2=[ cc1(:,1) cc2(:,1) cc3(:,1) cc4(:,1) cc5(:,1) ]

huigui=[huigui1 huigui2]
reg=[reg1 reg2]
for i=1:10
   z1=huigui(:,i)
   z2=reg(:,i)
[fitresult, gof]=createFit(z2,z1)
xishu(1,i)=fitresult.a
xishu(2,i)=fitresult.b
xishu(3,i)=fitresult.c
R(i)=gof.rsquare
figure(i)
end
figure(1)

subplot(2,1,1)
figure(2)

subplot(2,1,2)
%%回归
function [fitresult, gof] = createFit(x, huigui1)


%% Fit: 'untitled fit 1'.
[xData, yData] = prepareCurveData( x, huigui1 );
```

```
ft = fittype( 'a*x^2+b*x+c', 'independent', 'x', 'dependent', 'y' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
opts.StartPoint = [0.91875420971507 0.97984104882752 0.998338142101944];
[fitresult, gof] = fit( xData, yData, ft, opts );
figure( 'Name', 'untitled fit 1' );
h = plot( fitresult, xData, yData );
legend( h, 'huigui1 vs. x', 'untitled fit 1', 'Location', 'NorthEast', 'Interpreter', 'none' );
%画坐标
xlabel( 'x', 'Interpreter', 'none' );
ylabel( 'huigui1', 'Interpreter', 'none' );
grid on




%% 灰色关联量化模型
clear;clc
load('hs1')
load('hs2')
load('hs')
%X=abs(hs(:,1:5));
%X=abs(hs(:,6:10));
X=[hs1,hs2]
% X=abs(hs);

%% 判断是否需要正向化
[n,m] = size(X);
disp(['共有' num2str(n) '个评价对象，' num2str(m) '个评价指标'])
Position = input('输入位置：');
disp(['极小为1，中间为2，区间为3'])
    Type = input('输入处理形式：') %极小值正向化处理
    % 注意，Position和Type是两个同维度的行向量
    for i = 1 : size(Position,2)
        %这里需要对这些列分别处理，因此我们需要知道一共要处理的次数，即循环的次数
       X(:,Position(i)) = Positivization(X(:,Position(i)),Type(i),Position(i));

    end
    disp('正向化后的矩阵 X = ')
    disp(X)


%% 对正向化后的矩阵进行预处理
z = X ./ repmat(sum(X.*X) .^ 0.5, n, 1);
disp('标准化矩阵 Z = ')
disp(z)
```

```
%% 构造母序列和子序列
Y = max(z,[],2); % 母序列为虚拟的，用每一行的最大值构成的列向量表示母序列
X = z; % 子序列就是预处理后的数据矩阵

%% 计算得分
absX0_Xi = abs(X - repmat(Y,1,size(X,2))) % 计算|X0-Xi|矩阵
a = min(min(absX0_Xi)) % 计算两级最小差a
b = max(max(absX0_Xi)) % 计算两级最大差b
rho = 0.5; % 分辨系数取0.5
gamma = (a+rho*b) ./ (absX0_Xi + rho*b) % 计算子序列中各个指标与母序列的关联系数

Z=X;
       if sum(sum(Z<0)) >0 % 如果之前标准化后的Z矩阵中存在负数，则重新对X进行标准化
           disp('原来标准化得到的Z矩阵中存在负数，所以需要对X重新标准化')
           for i = 1:n
               for j = 1:m
                   Z(i,j) = [X(i,j) - min(X(:,j))] / [max(X(:,j)) - min(X(:,j))];
               end
           end
           disp('X重新进行标准化得到的标准化矩阵Z为：')
           disp(Z)
       end
       weight = Entropy_Method(Z);
       disp('熵权法确定的权重为：')
       disp(weight)
score = sum(X .* repmat(weight,size(X,1),1),2); % 未归一化的得分
stand_S = score / sum(score); % 归一化后的得分
[sorted_S,index] = sort(stand_S ,'descend') % 进行排序



function [W] = Entropy_Method(Z)
% 输出
% W：熵权，1*m的行向量

 x=input('是否要用改进的熵权法 用请输入1，不用输入2 =  ')

 if x==1
%% 改进的计算熵权

    [n,m] = size(Z);
    alpha=10
    D = zeros(1,m); % 初始化保存信息效用值的行向量
    for i = 1:m
        x = Z(:,i); % 取出第i列的指标
        p = x / sum(x);
```

```
        % 注意，p有可能为0，此时计算ln(p)*p时，Matlab会返回NaN，所以这里我们自己定义一个函数
        e(i) = -sum(p .* mylog(p)) / log(n); % 计算信息熵
        D(i) = 1- e(i); % 计算信息效用值
    end
    W = D ./ sum(D); % 将信息效用值归一化，得到权重
    %%权重改进
    ae=mean(e)
    w=mean(W);
    for i=1:m
        if W(i)~=0
    W(i)=(1-ae^alpha)*W(i)+ae^alpha*w;

        else
            W(i)=0;
        end
    end

 end

 if x==2
    %% 原本的熵权法
       [n,m] = size(Z);
    D = zeros(1,m); % 初始化保存信息效用值的行向量
    for i = 1:m
        x = Z(:,i); % 取出第i列的指标
        p = x / sum(x);
        % 注意，p有可能为0，此时计算ln(p)*p时，Matlab会返回NaN，所以这里我们自己定义一个函数
        e = -sum(p .* mylog(p)) / log(n); % 计算信息熵
        D(i) = 1- e; % 计算信息效用值
    end
    W = D ./ sum(D); % 将信息效用值归一化，得到权重

 end
end

% 重新定义一个mylog函数，当输入的p中元素为0时，返回0
function [lnp] = mylog(p)
n = length(p); % 向量的长度
lnp = zeros(n,1); % 初始化最后的结果
    for i = 1:n % 开始循环
        if p(i) == 0 % 如果第i个元素为0
            lnp(i) = 0; % 那么返回的第i个结果也为0
        else
            lnp(i) = log(p(i));
        end
    end
end
```

附录 D　问题二–python 源程序

```
# 问题二：AGRNN模型求解与验证

# 引入数据处理的包
import numpy as np
import pandas as pd
import seaborn as sns
# 引入机器学习的包
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from pyGRNN import GRNN
from sklearn.model_selection import KFold
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import joblib

# 引入附件中data3
data3 = pd.read_excel("第9题数据.xlsx",sheet_name="data3")
# 准备相关数据
# 工业参数
InPara = data3.iloc[:,0:2]
# 结构变量
Stru = data3.iloc[:,2:5]
# 产品性能
PerFor = data3.iloc[:,5:8]
# 厚度
thickness = Stru.iloc[:,0]
# 孔隙率
poreRate = Stru.iloc[:,1]
# 回弹率
reboRate = Stru.iloc[:,2]
X_train = InPara.copy()
# 改变Y_train，拟合不同的结构变量
Y_train = thickness.copy()

# 在训练集上的效果
# 并保存相关模型
res = Stru.copy()
for i in range(3):
    Y_train = Stru.iloc[:,i].copy()
    pipeline = make_pipeline(StandardScaler(),GRNN(calibration="gradient_search"))
```

```
    pipeline.fit(X_train,Y_train)
    if i == 0:
        res["厚度mm（预测）"] = pipeline.predict(X_train)
        res["厚度mm（差值）"] = res["厚度mm（预测）"] - res["厚度mm"]
        joblib.dump(pipeline,"thick.joblib.dat")
    elif i == 1:
        res["孔隙率（%）（预测）"] = pipeline.predict(X_train)
        res["孔隙率（%）（差值）"] = res["孔隙率（%）（预测）"] -res["孔隙率（%）"]
        joblib.dump(pipeline,"pore.joblib.dat")
    elif i == 2:
        res["压缩回弹性率（%）（预测）"] = pipeline.predict(X_train)
        res["压缩回弹性率（%）（差值）"] = res["压缩回弹性率（%）（预测）"] -
            res["压缩回弹性率（%）"]
        joblib.dump(pipeline,"rebo.joblib.dat")
    score = pipeline.score(X_train,Y_train)
    print("\n---------------------------------------")
    print("第%1d个结构变量" % (i + 1))
    print('sigma: %s, Accuracy: %.3f' % (pipeline.get_params()['grnn'].sigma, score))
res.to_excel("拟合值及差值.xlsx")

# 十折交叉验证
for i in range(3):
    Y_train = Stru.iloc[:,i].copy()
    kf = KFold(n_splits=10,shuffle=True,random_state=2)
    kfold = kf.split(X_train,Y_train)
    # 记录每次的性能
    print("\n---------------------------------------")
    print("第%1d个结构变量" % (i + 1))
    scores = []
    for k,(train,test) in enumerate(kfold):
        # 加入标准化
        pipeline = make_pipeline(StandardScaler(),GRNN(calibration="gradient_search"))
        pipeline.fit(X_train.iloc[train,:],Y_train.iloc[train])
        score = pipeline.score(X_train.iloc[test,:],Y_train.iloc[test])
        scores.append(score)
        # print(pipeline.get_params()['grnn'].sigma)
        print('Fold: %2d, sigma: %s, Accuracy: %.3f' % (k+1,
            pipeline.get_params()['grnn'].sigma, score))
        # print('Fold: %2d, Training/Test Split Distribution: %s, Accuracy: %.3f' % (k+1,
            np.bincount(Y_train.iloc[train]), score))
    print('Cross-Validation accuracy: %.3f +/- %.3f' %(np.mean(scores), np.std(scores)))

# 预测集导入
# 并导出预测结果
X_pre = pd.DataFrame(
    {'接收距离(cm)':[38,33,28,23,38,33,28,23],
    '热风速度(r/min)':[850,950,1150,1250,1250,1150,950,850]}
```

```
)
res = X_pre.copy()
for i in range(3):
    Y_train = Stru.iloc[:,i].copy()
    pipeline = make_pipeline(StandardScaler(),GRNN(calibration="gradient_search"))
    pipeline.fit(X_train,Y_train)
    if i == 0:
        res["厚度mm"] = pipeline.predict(X_pre)
    elif i == 1:
        res["孔隙率（%）"] = pipeline.predict(X_pre)
    elif i == 2:
        res["压缩回弹性率（%）"] = pipeline.predict(X_pre)
# 输出预测结果
res.to_excel("问题二预测结果.xlsx")
```

附录 E　问题三–matlab 源程序

```
%%问题三
%%数据预处理
clc;clear
load('data3')%data3的数据
x=data3(:,6:8);%b1 b2 b3
x_min=zeros( 25 ,3 );
y1=zeros(3,25);
y2=zeros(3,25);
y3=zeros(3,25);
a=5
 %% y1的迭代修正
for i=1:75
          y1(mod(i-1,3)+1,ceil(i/3))=x(i,1);
end

y1_er=(max(y1)-min(y1))./mean(y1)*100;


for j=1:25
while y1_er(j)>=a
     y1_am=max(abs(y1(:,j)-mean(y1(:,j)))); %%离平均值最大的数
  for i=1:3

    if abs(y1(i,j)-mean(y1(:,j)))==y1_am
       y1(i,j)=mean(y1(:,j));
    end
 end
  y1_er=(max(y1)-min(y1))./mean(y1)*100;
```

```
 end
end



%% y2的迭代修正
   for i=1:75
          y2(mod(i,3)+1,ceil(i/3))=x(i,2);
   end
   y2_er=(max(y2)-min(y2))./mean(y2)*100;
for j=1:25
while y2_er(j)>=a
    y2_am=max(abs(y2(:,j)-mean(y2(:,j)))) ;%%离平均值最大的数
  for i=1:3

    if abs(y2(i,j)-mean(y2(:,j)))==y2_am
       y2(i,j)=mean(y2(:,j));
    end
 end
  y2_er=(max(y2)-min(y2))./mean(y2)*100;

 end
end

%% y3的迭代修正
  for i=1:75
          y3(mod(i,3)+1,ceil(i/3))=x(i,3);
  end
   y3_er=(max(y3)-min(y3))./mean(y3)*100;

for j=1:25
while y3_er(j)>=a
    y3_am=max(abs(y3(:,j)-mean(y3(:,j)))) ;%%离平均值最大的数
  for i=1:3

    if abs(y3(i,j)-mean(y3(:,j)))==y3_am
       y3(i,j)=mean(y3(:,j));
    end
 end
  y3_er=(max(y3)-min(y3))./mean(y3)*100;

 end
end



y1=y1(:);
```

```
y2=y2(:);
y3=y3(:);
y=[y1,y2,y3]
% y1=y1'
% y2=y2'
% y3=y3'
% y=[y1,y2,y3];
```

附录 F　问题三–python 源程序

```
# 问题三：帝国竞争算法求解
import numpy as np
import pandas as pd
import joblib
from mealpy.human_based.ICA import BaseICA
import warnings
warnings.filterwarnings('ignore')
# 导入问题二中训练的数据结构
# 厚度
thick_model = joblib.load("thick.joblib.dat")
# 孔隙率
pore_model = joblib.load("pore.joblib.dat")
# 回弹率
rebo_model = joblib.load("rebo.joblib.dat")

pls = pd.read_excel("pls系数.xlsx")
def effiencyFor(b1,b2,b3):
    # 用于存放各项
    temp = np.array([
        b1, b2, b3, 1/b1, 1/b2,
        b1/b2, b2/b3, b1/b3, b3/b2, b3/b1,
        1/(b1**2), 1/(b3**2), 1/(b1*b2), 1/(b2*b3), 1/(b1*b3),
        b2/(b1**2), b1/(b2**2), b1/(b3**2), b1/(b2*b3)
    ])

    after = pls["过滤阻力Pa"].values[1:]*temp
    res1 = after.sum() + pls["过滤阻力Pa"].values[0]

    after = pls["过滤效率（%）"].values[1:]*temp
    res2 = after.sum() + pls["过滤效率（%）"].values[0]

    after = pls["透气性 mm/s"].values[1:]*temp
    res3 = after.sum() + pls["透气性 mm/s"].values[0]

    res = np.array([res1,res2,res3])
```

```
    return res

def fitness_function(solution):
    X_pre = pd.DataFrame(
        {'接收距离(cm)':[solution[0]],
        '热风速度(r/min)':[solution[1]]}
    )
    # 厚度
    thick = thick_model.predict(X_pre)[0]
    # 孔隙率
    pore = pore_model.predict(X_pre)[0]
    # 回弹率
    rebo = rebo_model.predict(X_pre)[0]

    if thick<=0:
        return -1/(solution[0]*solution[1])

    # 产品性能
    effiency = effiencyFor(thick, pore, rebo)


    return effiency[1]

problem_dict1 = {
    "fit_func": fitness_function,
    "lb": [5, 100],
    "ub": [100, 2000],
    "minmax": "max",
}

epoch = 1000
pop_size = 100
empire_count = 8
assimilation_coeff = 2.0
revolution_prob = 0.05
revolution_rate = 0.1
revolution_step_size = 0.1
zeta = 0.2


model = BaseICA(problem_dict1, epoch, pop_size, empire_count, assimilation_coeff,
    revolution_prob, revolution_rate, revolution_step_size, zeta)
best_position, best_fitness = model.solve()
print(f"Solution: {best_position}, Fitness: {best_fitness}")
resX = pd.DataFrame(
        {'接收距离(cm)':[best_position[0]],
        '热风速度(r/min)':[best_position[1]]}
```

```
    )
# 厚度
thick = thick_model.predict(resX)[0]
# 孔隙率
pore = pore_model.predict(resX)[0]
# 回弹率
rebo = rebo_model.predict(resX)[0]

effRes = effiencyFor(thick, pore, rebo)

print("第三问最终结果：")
print(best_position)
print("厚度：%.3f; 孔隙率：%.3f; 回弹率：%.3f" % (thick, pore, rebo))
print("产品性能：",effRes)

model.history.save_global_best_fitness_chart(filename="problem2/first")
```

附录 G　问题四–python 源程序

```
# 问题四：理想点法求解
# 引入对应的包
import pandas as pd
import numpy as np
import seaborn as sns
from ICA import ImproveICA
import joblib
import warnings
warnings.filterwarnings('ignore')

# 导入问题二中训练的数据结构
# 厚度
thick_model = joblib.load("thick.joblib.dat")
# 孔隙率
pore_model = joblib.load("pore.joblib.dat")
# 回弹率
rebo_model = joblib.load("rebo.joblib.dat")

pls = pd.read_excel("pls系数.xlsx")
def effiencyFor(b1,b2,b3):
    # 用于存放各项
    temp = np.array([
        b1, b2, b3, 1/b1, 1/b2,
        b1/b2, b2/b3, b1/b3, b3/b2, b3/b1,
        1/(b1**2), 1/(b3**2), 1/(b1*b2), 1/(b2*b3), 1/(b1*b3),
        b2/(b1**2), b1/(b2**2), b1/(b3**2), b1/(b2*b3)
```

```
    ])

    after = pls["过滤阻力Pa"].values[1:]*temp
    res1 = after.sum() + pls["过滤阻力Pa"].values[0]

    after = pls["过滤效率 (%) "].values[1:]*temp
    res2 = after.sum() + pls["过滤效率 (%) "].values[0]

    after = pls["透气性 mm/s"].values[1:]*temp
    res3 = after.sum() + pls["透气性 mm/s"].values[0]

    res = np.array([res1,res2,res3])
    return res
# 用于计算最小过滤阻力
def fitness_function(solution):
    X_pre = pd.DataFrame(
        {'接收距离(cm)':[solution[0]],
        '热风速度(r/min)':[solution[1]]}
    )
    # 厚度
    thick = thick_model.predict(X_pre)[0]
    # 孔隙率
    pore = pore_model.predict(X_pre)[0]
    # 回弹率
    rebo = rebo_model.predict(X_pre)[0]

    if thick<=0:
        return 1e8/(solution[0]*solution[1])
    # 约束1:
    def g1():
        if rebo >= 100:
            return rebo - 100
        elif rebo <85:
            return 85 - rebo
        else:
            return 0
    # 约束2:
    def g2():
        if thick>=3:
            return thick - 3
        else:
            return 0
    def violate(value):

        if value > 0:
            return 1e5*value
        else:
```

```
            return 0
    # 产品性能
    effiency = effiencyFor(thick, pore, rebo)
    return effiency[0] + violate(g1()) + violate(g2())
problem_dict1 = {
    "fit_func": fitness_function,
    "lb": [5, 100],
    "ub": [100, 2000],
    "minmax": "min",
}

epoch = 1000
pop_size = 200
empire_count = 10
assimilation_coeff = 2.0
revolution_prob = 0.05
revolution_rate = 0.15
revolution_step_size = 0.15
zeta = 0.2
model = ImproveICA(problem_dict1, epoch, pop_size, empire_count, assimilation_coeff,
    revolution_prob, revolution_rate, revolution_step_size, zeta)
best_position, best_fitness = model.solve()
print(f"Solution: {best_position}, Fitness: {best_fitness}")
resX = pd.DataFrame(
       {'接收距离(cm)':[best_position[0]],
       '热风速度(r/min)':[best_position[1]]}
    )
# 厚度
thick = thick_model.predict(resX)[0]
# 孔隙率
pore = pore_model.predict(resX)[0]
# 回弹率
rebo = rebo_model.predict(resX)[0]

effRes = effiencyFor(thick, pore, rebo)
print("第四问计算最小过滤阻力：")
print(best_position)
print("厚度：%.3f; 孔隙率：%.3f; 回弹率：%.3f" % (thick, pore, rebo))
print("产品性能：",effRes)
model.history.save_global_best_fitness_chart(filename="problem4/first")
# 用于计算最大过滤效率
def fitness_function(solution):
    X_pre = pd.DataFrame(
       {'接收距离(cm)':[solution[0]],
       '热风速度(r/min)':[solution[1]]}
    )
    # 厚度
```

```
    thick = thick_model.predict(X_pre)[0]
    # 孔隙率
    pore = pore_model.predict(X_pre)[0]
    # 回弹率
    rebo = rebo_model.predict(X_pre)[0]
    if thick<=0:
        return -1/(solution[0]*solution[1])
    # 约束1:
    def g1():
        if rebo >= 100:
            return rebo - 100
        elif rebo <85:
            return 85 - rebo
        else:
            return 0
    # 约束2:
    def g2():
        if thick>=3:
            return thick - 3
        else:
            return 0
    def violate(value):
        if value > 0:
            return -1e5*value
        else:
            return 0
    # 产品性能
    effiency = effiencyFor(thick, pore, rebo)
    return effiency[1] + violate(g1()) + violate(g2())

problem_dict1 = {
    "fit_func": fitness_function,
    "lb": [5, 100],
    "ub": [100, 2000],
    "minmax": "max",
}

epoch = 1000
pop_size = 100
empire_count = 8
assimilation_coeff = 2.0
revolution_prob = 0.05
revolution_rate = 0.1
revolution_step_size = 0.1
zeta = 0.2
```

```
model = ImproveICA(problem_dict1, epoch, pop_size, empire_count, assimilation_coeff,
    revolution_prob, revolution_rate, revolution_step_size, zeta)
best_position, best_fitness = model.solve()
print(f"Solution: {best_position}, Fitness: {best_fitness}")

# 理想点法计算
# 用于计算最小过滤阻力
# 理想点第一个的权重
w = 0.5
def fitness_function(solution):
    X_pre = pd.DataFrame(
        {'接收距离(cm)':[solution[0]],
        '热风速度(r/min)':[solution[1]]}
    )
    # 厚度
    thick = thick_model.predict(X_pre)[0]
    # 孔隙率
    pore = pore_model.predict(X_pre)[0]
    # 回弹率
    rebo = rebo_model.predict(X_pre)[0]

    if thick<=0:
        return 1e8/(solution[0]*solution[1])

    # 约束1:
    def g1():
        if rebo >= 100:
            return rebo - 100
        elif rebo <85:
            return 85 - rebo
        else:
            return 0

    # 约束2:
    def g2():
        if thick>=3:
            return thick - 3
        else:
            return 0

    def violate(value):

        if value > 0:
            return 1e5*value
        else:
            return 0
```

```
    # 产品性能
    effiency = effiencyFor(thick, pore, rebo)

    fx = w*((effiency[0]-23.853014893014915)**2) + (1-w)*((effiency[1]-91.09381492715329)**2)

    return fx + violate(g1()) + violate(g2())

problem_dict1 = {
    "fit_func": fitness_function,
    "lb": [5, 100],
    "ub": [100, 2000],
    "minmax": "min",
}
epoch = 300
pop_size = 200
empire_count = 10
assimilation_coeff = 2.0
revolution_prob = 0.05
revolution_rate = 0.15
revolution_step_size = 0.15
zeta = 0.2
model = ImproveICA(problem_dict1, epoch, pop_size, empire_count, assimilation_coeff,
    revolution_prob, revolution_rate, revolution_step_size, zeta)
best_position, best_fitness = model.solve()
print(f"Solution: {best_position}, Fitness: {best_fitness}")
resX = pd.DataFrame(
       {'接收距离(cm)':[best_position[0]],
        '热风速度(r/min)':[best_position[1]]}
    )
# 厚度
thick = thick_model.predict(resX)[0]
# 孔隙率
pore = pore_model.predict(resX)[0]
# 回弹率
rebo = rebo_model.predict(resX)[0]

effRes = effiencyFor(thick, pore, rebo)

print("第四问理想点法结果: ")
print(best_position)
print("厚度: %.3f; 孔隙率: %.3f; 回弹率: %.3f" % (thick, pore, rebo))
print("产品性能: ",effRes)


# 帝国竞争算法
import numpy as np
from copy import deepcopy
```

```python
from mealpy.optimizer import Optimizer


class ImproveICA(Optimizer):
    def __init__(self, problem, epoch, pop_size, empire_count, assimilation_coeff,
                 revolution_prob, revolution_rate, revolution_step_size, zeta, **kwargs):
        '''
        epoch:最大迭代次数
        pop_size:种群大小
        empire_count:帝国数量
        assimilation_coeff:同化系数
        revolution_prob:革命概率
        revolution_rate:
        revolution_step_size:
        zeta:计算帝国总成本时，所需殖民地平均成本系数
        '''
        # 定义问题
        super().__init__(problem, kwargs)
        # 检查相关系数是否有问题
        self.epoch = self.validator.check_int("epoch", epoch, [1, 100000])
        self.pop_size = self.validator.check_int("pop_size", pop_size, [10, 10000])
        self.empire_count = self.validator.check_int("empire_count", empire_count, [2, 2 +
            int(self.pop_size / 5)])
        self.assimilation_coeff = self.validator.check_float("assimilation_coeff",
            assimilation_coeff, [1.0, 3.0])
        self.revolution_prob = self.validator.check_float("revolution_prob", revolution_prob,
            (0, 1.0))
        self.revolution_rate = self.validator.check_float("revolution_rate", revolution_rate,
            (0, 1.0))
        self.revolution_step_size = self.validator.check_float("revolution_step_size",
            revolution_step_size, (0, 1.0))
        self.zeta = self.validator.check_float("zeta", zeta, (0, 1.0))

        self.nfe_per_epoch = self.pop_size
        self.sort_flag = True
        self.pop_empires, self.pop_colonies, self.empires = None, None, None
        self.n_revoluted_variables, self.idx_list_variables = None, None

    # 革命
    def revolution_country__(self, position, idx_list_variables, n_revoluted):
        pos_new = position + self.revolution_step_size * np.random.normal(0, 1,
            self.problem.n_dims)
        idx_list = np.random.choice(idx_list_variables, n_revoluted, replace=False)
        position[idx_list] = pos_new[idx_list]
        return position

    def after_initialization(self):
```

```
        self.pop, self.g_best = self.get_global_best_solution(self.pop)
        # Initialization
        self.n_revoluted_variables = int(round(self.revolution_rate * self.problem.n_dims))
        self.idx_list_variables = list(range(0, self.problem.n_dims))

        # pop = Empires
        colony_count = self.pop_size - self.empire_count
        self.pop_empires = deepcopy(self.pop[:self.empire_count])
        self.pop_colonies = deepcopy(self.pop[self.empire_count:])

        cost_empires_list = np.array([solution[self.ID_TAR][self.ID_FIT] for solution in
            self.pop_empires])
        cost_empires_list_normalized = cost_empires_list - (np.max(cost_empires_list) +
            np.min(cost_empires_list))
        prob_empires_list = np.abs(cost_empires_list_normalized /
            np.sum(cost_empires_list_normalized))
        # 随机的选择殖民地给帝国
        self.empires = {}
        idx_already_selected = []
        for i in range(0, self.empire_count - 1):
            self.empires[i] = []
            n_colonies = int(round(prob_empires_list[i] * colony_count))
            idx_list = np.random.choice(list(set(range(0, colony_count)) -
                set(idx_already_selected)), n_colonies, replace=False).tolist()
            idx_already_selected += idx_list
            for idx in idx_list:
                self.empires[i].append(self.pop_colonies[idx])
        idx_last = list(set(range(0, colony_count)) - set(idx_already_selected))
        self.empires[self.empire_count - 1] = []
        for idx in idx_last:
            self.empires[self.empire_count - 1].append(self.pop_colonies[idx])

    def evolve(self, epoch):
        '''
            对问题进行求解
            epoch值当前的迭代
        '''
        nfe_epoch = 0

        '''
            同化
            对殖民地进行操作
            colonies是帝国的殖民地
        '''
        for idx, colonies in self.empires.items():
            for idx_colony, colony in enumerate(colonies):
                pos_new = colony[self.ID_POS] + self.assimilation_coeff * \
```

```
                np.random.uniform(0, 1, self.problem.n_dims) *
                    (self.pop_empires[idx][self.ID_POS] - colony[self.ID_POS])
            pos_new = self.amend_position(pos_new, self.problem.lb, self.problem.ub)
            self.empires[idx][idx_colony][self.ID_POS] = pos_new
            if self.mode not in self.AVAILABLE_MODES:
                self.empires[idx][idx_colony][self.ID_TAR] = self.get_target_wrapper(pos_new)
        self.empires[idx] = self.update_target_wrapper_population(self.empires[idx])
        nfe_epoch += len(self.empires[idx])

    '''
        革命
    '''
    for idx, colonies in self.empires.items():
        # 帝国主义国家革命
        pos_new_em = self.revolution_country__(self.pop_empires[idx][self.ID_POS],
            self.idx_list_variables, self.n_revoluted_variables)
        pos_new_em = self.amend_position(pos_new_em, self.problem.lb, self.problem.ub)
        self.pop_empires[idx][self.ID_POS] = pos_new_em
        if self.mode not in self.AVAILABLE_MODES:
            self.pop_empires[idx][self.ID_TAR] = self.get_target_wrapper(pos_new_em)

        # Apply revolution to Colonies
        for idx_colony, colony in enumerate(colonies):
            if np.random.rand() < self.revolution_prob:
                pos_new = self.revolution_country__(colony[self.ID_POS],
                    self.idx_list_variables, self.n_revoluted_variables)
                pos_new = self.amend_position(pos_new, self.problem.lb, self.problem.ub)
                self.empires[idx][idx_colony][self.ID_POS] = pos_new
                if self.mode not in self.AVAILABLE_MODES:
                    self.empires[idx][idx_colony][self.ID_TAR] =
                        self.get_target_wrapper(pos_new)
        self.empires[idx] = self.update_target_wrapper_population(self.empires[idx])
        nfe_epoch+= len(self.empires[idx])
    self.pop_empires = self.update_target_wrapper_population(self.pop_empires)
    self.update_global_best_solution(self.pop_empires, save=False)
    nfe_epoch += len(self.pop_empires)

    # 帝国内部竞争
    for idx, colonies in self.empires.items():
        for idx_colony, colony in enumerate(colonies):
            if self.compare_agent(colony, self.pop_empires[idx]):
                self.empires[idx][idx_colony], self.pop_empires[idx] =
                    deepcopy(self.pop_empires[idx]), deepcopy(colony)

    # 更新帝国的总客观价值
    cost_empires_list = []
    for idx, colonies in self.empires.items():
```

```
            fit_list = np.array([solution[self.ID_TAR][self.ID_FIT] for solution in colonies])
            fit_empire = self.pop_empires[idx][self.ID_TAR][self.ID_FIT] + self.zeta *
                np.mean(fit_list)
            cost_empires_list.append(fit_empire)
        cost_empires_list = np.array(cost_empires_list)

        # 根据其总权力找出每个帝国的拥有概率
        cost_empires_list_normalized = cost_empires_list - (np.max(cost_empires_list) +
            np.min(cost_empires_list))
        prob_empires_list = np.abs(cost_empires_list_normalized /
            np.sum(cost_empires_list_normalized))

        uniform_list = np.random.uniform(0, 1, len(prob_empires_list))
        vector_D = prob_empires_list - uniform_list
        idx_empire = np.argmax(vector_D)

        # 找到其中最弱的帝国和最弱的殖民地
        idx_weakest_empire = np.argmax(cost_empires_list)
        if len(self.empires[idx_weakest_empire]) > 0:
            colonies_sorted, best, worst =
                self.get_special_solutions(self.empires[idx_weakest_empire])
            self.empires[idx_empire].append(colonies_sorted.pop(-1))
        else:
            self.empires[idx_empire].append(self.pop_empires.pop(idx_weakest_empire))

        self.pop = self.pop_empires + self.pop_colonies
```