|  |
| --- |
| 附录1：问题一的参考代码 |
| 介绍：该代码是用python编写的，用于对数据做归一化处理。 |
| import pandas as pd import numpy as np  df=pd.read\_excel('data.xlsx')  # 获取除第一列外的数据 data\_to\_scale = df.iloc[:, 1:]  # 计算最小值和最大值 min\_vals = data\_to\_scale.min() max\_vals = data\_to\_scale.max()  # 进行标准化处理 scaled\_data = (data\_to\_scale - min\_vals) / (max\_vals - min\_vals)  # 将标准化的数据替换原数据 df.iloc[:, 1:] = scaled\_data  # 保存结果到新Excel文件 df.to\_excel('scaled\_data.xlsx', index=False, engine='openpyxl') |

|  |
| --- |
| 附录2：问题一的参考代码 |
| 介绍：该代码是用python编写的，用于求解工艺参数与产品性能之间的相关性系数。 |
| import pandas as pd import numpy as np  df = pd.read\_excel('scaled\_data.xlsx')  # Python中列索引从0开始，所以第2列是索引1，第3列是索引2，第4列是索引3，以此类推 cols\_x = df.iloc[:, 1:4] # 第2-4列 cols\_y = df.iloc[:, 4:11] # 第5-11列  # 计算相关性系数（R方值） r\_squared = pd.DataFrame(index=cols\_x.columns, columns=cols\_y.columns)  for x\_col in cols\_x.columns:  for y\_col in cols\_y.columns:  # 计算相关性系数  correlation\_matrix = np.corrcoef(cols\_x[x\_col], cols\_y[y\_col])  correlation = correlation\_matrix[0, 1]  # 计算R方值  r\_squared.loc[x\_col, y\_col] = correlation \*\* 2  print(r\_squared) |

|  |
| --- |
| 附录3：问题一的参考代码 |
| 介绍：该代码是用python编写的，用于求解不同产品性能之间的相关性系数。 |
| import pandas as pd import numpy as np  df = pd.read\_excel('scaled\_data.xlsx')  # Python中列索引从0开始，所以第2列是索引1，第3列是索引2，第4列是索引3，以此类推 cols\_x = df.iloc[:, 4:11] # 第2-4列 cols\_y = df.iloc[:, 4:11] # 第5-11列  # 计算相关性系数（R方值） r\_squared = pd.DataFrame(index=cols\_x.columns, columns=cols\_y.columns)  for x\_col in cols\_x.columns:  for y\_col in cols\_y.columns:  # 计算相关性系数  correlation\_matrix = np.corrcoef(cols\_x[x\_col], cols\_y[y\_col])  correlation = correlation\_matrix[0, 1]  # 计算R方值  r\_squared.loc[x\_col, y\_col] = correlation \*\* 2  print(r\_squared) |

|  |
| --- |
| 附录4：问题一的参考代码 |
| 介绍：该代码是用python编写的，用于绘制工艺参数与产品性能之间的斯皮尔曼相关性热力图。 |
| import pandas as pd from scipy.stats import spearmanr import seaborn as sns import matplotlib.pyplot as plt  # 设置中文字体 plt.rcParams['font.family'] = ['Microsoft YaHei']  # 读取Excel文件 df = pd.read\_excel('scaled\_data.xlsx')  # 选择要分析的列 columns = ['树脂含量（wt%）', '固化温度(℃)', '减量程度（%）', '断裂强力', '断裂伸长量', '撕裂强力', '透气率', '透湿率', '柔软度', '折皱回复率'] data = df[columns]  # 计算斯皮尔曼相关系数 correlation\_matrix, p\_values = spearmanr(data)  # 将相关系数矩阵转换为DataFrame以便于查看 correlation\_df = pd.DataFrame(correlation\_matrix, index=columns, columns=columns)  # 绘制热力图 plt.figure(figsize=(10, 8)) sns.heatmap(correlation\_df, annot=True, cmap='coolwarm', vmin=-1, vmax=1) plt.title('工艺参数与产品性能之间的斯皮尔曼相关性热力图') plt.show() |

|  |
| --- |
| 附录5：问题一的参考代码 |
| 介绍：该代码是用python编写的，用于绘制不同产品性能之间的斯皮尔曼相关性热力图。 |
| import pandas as pd from scipy.stats import spearmanr import seaborn as sns import matplotlib.pyplot as plt  # 设置中文字体 plt.rcParams['font.family'] = ['Microsoft YaHei']  # 读取Excel文件 df = pd.read\_excel('scaled\_data.xlsx')  # 选择要分析的列 columns = ['断裂强力', '断裂伸长量', '撕裂强力', '透气率', '透湿率', '柔软度', '折皱回复率'] data = df[columns]  # 计算斯皮尔曼相关系数 correlation\_matrix, p\_values = spearmanr(data)  # 将相关系数矩阵转换为DataFrame以便于查看 correlation\_df = pd.DataFrame(correlation\_matrix, index=columns, columns=columns)  # 绘制热力图 plt.figure(figsize=(10, 8)) sns.heatmap(correlation\_df, annot=True, cmap='coolwarm', vmin=-1, vmax=1) plt.title('不同产品性能的斯皮尔曼相关性热力图') plt.show() |

|  |
| --- |
| 附录6：问题一的参考代码 |
| 介绍：该代码是用python编写的，用于求解不同工艺参数之间可能存在着交互作用，并绘制图像。 |
| import pandas as pd  import statsmodels.api as sm  from statsmodels.formula.api import ols  # 数据准备  data = {  '树脂含量': [15, 15, 15, 15, 20, 20, 20, 20, 25, 25, 25, 25, 30, 30, 30, 30],  '固化温度': [100, 110, 120, 130, 100, 110, 120, 130, 100, 110, 120, 130, 100, 110, 120, 130],  '减量程度': [0, 10, 20, 30, 10, 0, 30, 20, 20, 30, 0, 10, 30, 20, 10, 0],  '断裂强力': [1781.9181, 1142.4154, 1199.74011, 1218.922, 1164.56934, 1652.4389, 1211.408, 1233.107,  1182.962, 1248.54871, 1759.6982, 1214.9815, 1249.2589, 1250.4569, 1207.59863, 1730.8907]  }  # 创建 DataFrame  df = pd.DataFrame(data)  # 方差分析 (ANOVA) 函数  def anova(df, formula):  model = ols(formula, data=df).fit()  anova\_table = sm.stats.anova\_lm(model, typ=2)  return anova\_table  # 定义模型公式  formulas = {  '树脂含量': '断裂强力 ~ C(树脂含量)',  '固化温度': '断裂强力 ~ C(固化温度)',  '减量程度': '断裂强力 ~ C(减量程度)',  '交互效应：树脂含量\*固化温度': '断裂强力 ~ C(树脂含量) \* C(固化温度)',  '交互效应：树脂含量\*减量程度': '断裂强力 ~ C(树脂含量) \* C(减量程度)',  '交互效应：固化温度\*减量程度': '断裂强力 ~ C(固化温度) \* C(减量程度)',  }  # 计算 ANOVA  results = []  for key, formula in formulas.items():  try:  table = anova(df, formula)  for source in table.index:  results.append([key, source, table.loc[source, 'sum\_sq'], table.loc[source, 'df'],  table.loc[source, 'mean\_sq'] if 'mean\_sq' in table.columns else 'N/A',  table.loc[source, 'F'] if 'F' in table.columns else 'N/A'])  except Exception as e:  print(f"Error processing formula '{key}': {e}")  # 随机误差计算  model = ols('断裂强力 ~ C(树脂含量) \* C(固化温度) \* C(减量程度)', data=df).fit()  anova\_table = sm.stats.anova\_lm(model, typ=2)  ss\_total = anova\_table['sum\_sq'].sum()  ss\_between = sum([result[2] for result in results if '交互效应' in result[0]])  ss\_within = ss\_total - ss\_between  df\_within = len(df) - len(anova\_table.index)  ms\_within = ss\_within / df\_within  results.append(['随机误差', '剩余', ss\_within, df\_within, ms\_within, 'N/A'])  # 创建 DataFrame 并输出  results\_df = pd.DataFrame(results, columns=['方差来源', '离差平方和', '自由度', '均方', 'F值'])  print(results\_df) |

|  |
| --- |
| 附录7：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用多元线性回归建立工艺参数与人造革7种性能之间的关系模型。 |
| import numpy as np import pandas as pd from sklearn.linear\_model import LinearRegression from sklearn.preprocessing import PolynomialFeatures from sklearn.metrics import r2\_score import matplotlib.pyplot as plt  # 读取数据 df = pd.read\_excel("data.xlsx")  # 特征和目标变量 X = df[['树脂含量', '固化温度', '减量程度']] y = df['断裂强力']  # 生成二次项特征 poly = PolynomialFeatures(degree=1, include\_bias=False) X\_poly = poly.fit\_transform(X)  # 建立模型并训练 model = LinearRegression() model.fit(X\_poly, y)  # 预测和计算R-squared y\_pred = model.predict(X\_poly) r\_squared = r2\_score(y, y\_pred)  # 打印回归方程 feature\_names = poly.get\_feature\_names\_out() coef\_dict = dict(zip(feature\_names, model.coef\_)) print("回归方程的系数:", coef\_dict) print("回归方程的截距:", model.intercept\_) print("R-squared:", r\_squared) |
| 附录8：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用K折交叉验证检验多元线性回归建立工艺参数与人造革7种性能之间的关系模型是否存在过拟合。 |
| import numpy as np import pandas as pd from sklearn.linear\_model import LinearRegression from sklearn.preprocessing import PolynomialFeatures from sklearn.metrics import r2\_score from sklearn.model\_selection import KFold, cross\_val\_score import matplotlib.pyplot as plt  # 读取数据 df = pd.read\_excel("data.xlsx")  # 特征和目标变量 X = df[['树脂含量', '固化温度', '减量程度']] y = df['折皱回复率']  # 生成二次项特征 poly = PolynomialFeatures(degree=3, include\_bias=False) X\_poly = poly.fit\_transform(X)  # 建立模型 model = LinearRegression()  # K折交叉验证 kf = KFold(n\_splits=5, shuffle=True, random\_state=42) # 这里选择5折交叉验证，你可以根据需要调整 cross\_val\_scores = cross\_val\_score(model, X\_poly, y, cv=kf, scoring='r2')  # 输出交叉验证结果 print("交叉验证R-squared的平均值:", cross\_val\_scores.mean())  # 训练整个模型以输出回归方程 model.fit(X\_poly, y) y\_pred = model.predict(X\_poly) r\_squared = r2\_score(y, y\_pred)  # 打印回归方程 feature\_names = poly.get\_feature\_names\_out() coef\_dict = {name: round(coef, 6) for name, coef in zip(feature\_names, model.coef\_)} intercept = round(model.intercept\_, 6)  print("回归方程的系数:", coef\_dict) print("回归方程的截距:", intercept) print("训练集上的R-squared:", r\_squared) |

|  |
| --- |
| 附录9：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解断裂强力最佳工艺参数。 |
| import numpy as np  # 系数和截距 intercept = 240.830123 coefficients = {  '树脂含量': -3.392196,  '固化温度': -0.00856,  '减量程度': -15.491499,  '树脂含量^2': 0.068722,  '树脂含量 固化温度': -0.00807,  '树脂含量 减量程度': 0.110209,  '固化温度^2': -0.001647,  '固化温度 减量程度': 0.067183,  '减量程度^2': 0.110893 }  # 定义回归方程 def regression\_function(x1, x2, x3):  return (intercept +  coefficients['树脂含量'] \* x1 +  coefficients['固化温度'] \* x2 +  coefficients['减量程度'] \* x3 +  coefficients['树脂含量^2'] \* x1\*\*2 +  coefficients['树脂含量 固化温度'] \* x1 \* x2 +  coefficients['树脂含量 减量程度'] \* x1 \* x3 +  coefficients['固化温度^2'] \* x2\*\*2 +  coefficients['固化温度 减量程度'] \* x2 \* x3 +  coefficients['减量程度^2'] \* x3\*\*2)  # 定义搜索范围 x1\_range = np.linspace(15, 30, 150) # 树脂含量范围，取150个值 x2\_range = np.linspace(100, 130, 300) # 固化温度范围，取300个值 x3\_range = np.linspace(0, 30, 300) # 减量程度范围，取300个值  # 初始化最大值和对应的参数 max\_strength = -np.inf best\_params = (None, None, None)  # 遍历所有可能的参数组合 for x1 in x1\_range:  for x2 in x2\_range:  for x3 in x3\_range:  strength = regression\_function(x1, x2, x3)  if strength > max\_strength:  max\_strength = strength  best\_params = (x1, x2, x3)  print(f'Optimal Parameters:') print(f'树脂含量 = {best\_params[0]}') print(f'固化温度 = {best\_params[1]}') print(f'减量程度 = {best\_params[2]}') print(f'撕裂强力 = {max\_strength}') |

|  |
| --- |
| 附录10：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解断裂伸长量最佳工艺参数。 |
| import numpy as np  # 系数和截距 intercept = 0.117955 coefficients = {  '树脂含量': 0.042182,  '固化温度': 0.008205,  '减量程度': -0.029932,  '树脂含量^2': -0.0003,  '树脂含量 固化温度': -0.000305,  '树脂含量 减量程度': 0.000273,  '固化温度^2': 0.0,  '固化温度 减量程度': 2.5e-05,  '减量程度^2': 0.00055 }  # 定义回归方程 def regression\_function(x1, x2, x3):  return (intercept +  coefficients['树脂含量'] \* x1 +  coefficients['固化温度'] \* x2 +  coefficients['减量程度'] \* x3 +  coefficients['树脂含量^2'] \* x1\*\*2 +  coefficients['树脂含量 固化温度'] \* x1 \* x2 +  coefficients['树脂含量 减量程度'] \* x1 \* x3 +  coefficients['固化温度^2'] \* x2\*\*2 +  coefficients['固化温度 减量程度'] \* x2 \* x3 +  coefficients['减量程度^2'] \* x3\*\*2)  # 定义搜索范围 x1\_range = np.linspace(15, 30, 150) # 树脂含量范围，取150个值 x2\_range = np.linspace(100, 130, 300) # 固化温度范围，取300个值 x3\_range = np.linspace(0, 30, 300) # 减量程度范围，取300个值  # 初始化最大值和对应的参数 max\_strength = -np.inf best\_params = (None, None, None)  # 遍历所有可能的参数组合 for x1 in x1\_range:  for x2 in x2\_range:  for x3 in x3\_range:  strength = regression\_function(x1, x2, x3)  if strength > max\_strength:  max\_strength = strength  best\_params = (x1, x2, x3)  print(f'Optimal Parameters:') print(f'树脂含量 = {best\_params[0]}') print(f'固化温度 = {best\_params[1]}') print(f'减量程度 = {best\_params[2]}') print(f'断裂伸长量 = {max\_strength}') |

|  |
| --- |
| 附录11：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解柔软度最佳工艺参数。 |
| import numpy as np  # 更新后的系数和截距 intercept = 1.49625 coefficients = {  '树脂含量': -0.04875,  '固化温度': 0.005625,  '减量程度': 0.067875,  '树脂含量^2': 0, # 如果新的回归方程没有二次项，将其设置为0  '树脂含量 固化温度': 0, # 如果新的回归方程没有交互项，将其设置为0  '树脂含量 减量程度': 0,  '固化温度^2': 0,  '固化温度 减量程度': 0,  '减量程度^2': 0 }  # 更新后的回归方程 def regression\_function(x1, x2, x3):  return (intercept +  coefficients['树脂含量'] \* x1 +  coefficients['固化温度'] \* x2 +  coefficients['减量程度'] \* x3)  # 定义搜索范围 x1\_range = np.linspace(15, 30, 150) # 树脂含量范围，取150个值 x2\_range = np.linspace(100, 130, 300) # 固化温度范围，取300个值 x3\_range = np.linspace(0, 30, 300) # 减量程度范围，取300个值  # 初始化最大值和对应的参数 max\_strength = -np.inf best\_params = (None, None, None)  # 遍历所有可能的参数组合 for x1 in x1\_range:  for x2 in x2\_range:  for x3 in x3\_range:  strength = regression\_function(x1, x2, x3)  if strength > max\_strength:  max\_strength = strength  best\_params = (x1, x2, x3)  print(f'Optimal Parameters:') print(f'树脂含量 = {best\_params[0]}') print(f'固化温度 = {best\_params[1]}') print(f'减量程度 = {best\_params[2]}') print(f'柔软度 = {max\_strength}') |

|  |
| --- |
| 附录12：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解透气率最佳工艺参数。 |
| import numpy as np  # 系数和截距 intercept = 654.596534 coefficients = {  '树脂含量': 0.042795,  '固化温度': -4.425432,  '减量程度': -20.034932,  '树脂含量^2': -0.370575,  '树脂含量 固化温度': 0.227257,  '树脂含量 减量程度': -0.135007,  '固化温度^2': -0.026606,  '固化温度 减量程度': 0.281106,  '减量程度^2': -0.393256 }  # 定义回归方程 def regression\_function(x1, x2, x3):  return (intercept +  coefficients['树脂含量'] \* x1 +  coefficients['固化温度'] \* x2 +  coefficients['减量程度'] \* x3 +  coefficients['树脂含量^2'] \* x1\*\*2 +  coefficients['树脂含量 固化温度'] \* x1 \* x2 +  coefficients['树脂含量 减量程度'] \* x1 \* x3 +  coefficients['固化温度^2'] \* x2\*\*2 +  coefficients['固化温度 减量程度'] \* x2 \* x3 +  coefficients['减量程度^2'] \* x3\*\*2)  # 定义搜索范围 x1\_range = np.linspace(15, 30, 150) # 树脂含量范围，取150个值 x2\_range = np.linspace(100, 130, 300) # 固化温度范围，取300个值 x3\_range = np.linspace(0, 30, 300) # 减量程度范围，取300个值  # 初始化最大值和对应的参数 max\_strength = -np.inf best\_params = (None, None, None)  # 遍历所有可能的参数组合 for x1 in x1\_range:  for x2 in x2\_range:  for x3 in x3\_range:  strength = regression\_function(x1, x2, x3)  if strength > max\_strength:  max\_strength = strength  best\_params = (x1, x2, x3) print(f'Optimal Parameters:') print(f'树脂含量 = {best\_params[0]}') print(f'固化温度 = {best\_params[1]}') print(f'减量程度 = {best\_params[2]}') print(f'透气率 = {max\_strength}') |

|  |
| --- |
| 附录13：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解撕裂强力最佳工艺参数。 |
| import numpy as np  # 系数和截距 intercept = 240.830123 coefficients = {  '树脂含量': -3.392196,  '固化温度': -0.00856,  '减量程度': -15.491499,  '树脂含量^2': 0.068722,  '树脂含量 固化温度': -0.00807,  '树脂含量 减量程度': 0.110209,  '固化温度^2': -0.001647,  '固化温度 减量程度': 0.067183,  '减量程度^2': 0.110893 }  # 定义回归方程 def regression\_function(x1, x2, x3):  return (intercept +  coefficients['树脂含量'] \* x1 +  coefficients['固化温度'] \* x2 +  coefficients['减量程度'] \* x3 +  coefficients['树脂含量^2'] \* x1\*\*2 +  coefficients['树脂含量 固化温度'] \* x1 \* x2 +  coefficients['树脂含量 减量程度'] \* x1 \* x3 +  coefficients['固化温度^2'] \* x2\*\*2 +  coefficients['固化温度 减量程度'] \* x2 \* x3 +  coefficients['减量程度^2'] \* x3\*\*2)  # 定义搜索范围 x1\_range = np.linspace(15, 30, 150) # 树脂含量范围，取150个值 x2\_range = np.linspace(100, 130, 300) # 固化温度范围，取300个值 x3\_range = np.linspace(0, 30, 300) # 减量程度范围，取300个值  # 初始化最大值和对应的参数 max\_strength = -np.inf best\_params = (None, None, None)  # 遍历所有可能的参数组合 for x1 in x1\_range:  for x2 in x2\_range:  for x3 in x3\_range:  strength = regression\_function(x1, x2, x3)  if strength > max\_strength:  max\_strength = strength  best\_params = (x1, x2, x3)  print(f'Optimal Parameters:') print(f'树脂含量 = {best\_params[0]}') print(f'固化温度 = {best\_params[1]}') print(f'减量程度 = {best\_params[2]}') print(f'撕裂强力 = {max\_strength}') |

|  |
| --- |
| 附录14：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解折皱回复率最佳工艺参数。 |
| import numpy as np  # 更新后的系数和截距 intercept = 88.232665 coefficients = {  '树脂含量': 2.362457,  '固化温度': 0.567772,  '减量程度': 0.653567,  '树脂含量^2': -0.038973,  '树脂含量 固化温度': -0.005583,  '树脂含量 减量程度': 0.004328,  '固化温度^2': -0.001423,  '固化温度 减量程度': 0.000496,  '减量程度^2': -0.013223 }  # 更新后的回归方程 def regression\_function(x1, x2, x3):  return (intercept +  coefficients['树脂含量'] \* x1 +  coefficients['固化温度'] \* x2 +  coefficients['减量程度'] \* x3 +  coefficients['树脂含量^2'] \* x1\*\*2 +  coefficients['树脂含量 固化温度'] \* x1 \* x2 +  coefficients['树脂含量 减量程度'] \* x1 \* x3 +  coefficients['固化温度^2'] \* x2\*\*2 +  coefficients['固化温度 减量程度'] \* x2 \* x3 +  coefficients['减量程度^2'] \* x3\*\*2)  # 定义搜索范围 x1\_range = np.linspace(15, 30, 150) # 树脂含量范围，取150个值 x2\_range = np.linspace(100, 130, 300) # 固化温度范围，取300个值 x3\_range = np.linspace(0, 30, 300) # 减量程度范围，取300个值  # 初始化最大值和对应的参数 max\_strength = -np.inf best\_params = (None, None, None)  # 遍历所有可能的参数组合 for x1 in x1\_range:  for x2 in x2\_range:  for x3 in x3\_range:  strength = regression\_function(x1, x2, x3)  if strength > max\_strength:  max\_strength = strength  best\_params = (x1, x2, x3)  print(f'Optimal Parameters:') print(f'树脂含量 = {best\_params[0]}') print(f'固化温度 = {best\_params[1]}') print(f'减量程度 = {best\_params[2]}') print(f'折皱回复率 = {max\_strength}') |

|  |
| --- |
| 附录15：问题二的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解透湿率最佳工艺参数。 |
| import numpy as np  #系数和截距 intercept = 10622.53453 coefficients = {  '树脂含量': -233.822679,  '固化温度': -88.785737,  '减量程度': 20.464182,  '树脂含量^2': 2.374558,  '树脂含量 固化温度': 0.354642,  '树脂含量 减量程度': 1.728879,  '固化温度^2': 0.386926,  '固化温度 减量程度': -0.477996,  '减量程度^2': 0.063604 }  # 更新后的回归方程 def regression\_function(x1, x2, x3):  return (intercept +  coefficients['树脂含量'] \* x1 +  coefficients['固化温度'] \* x2 +  coefficients['减量程度'] \* x3 +  coefficients['树脂含量^2'] \* x1\*\*2 +  coefficients['树脂含量 固化温度'] \* x1 \* x2 +  coefficients['树脂含量 减量程度'] \* x1 \* x3 +  coefficients['固化温度^2'] \* x2\*\*2 +  coefficients['固化温度 减量程度'] \* x2 \* x3 +  coefficients['减量程度^2'] \* x3\*\*2)  # 定义搜索范围 x1\_range = np.linspace(15, 30, 150) # 树脂含量范围，取150个值 x2\_range = np.linspace(100, 130, 300) # 固化温度范围，取300个值 x3\_range = np.linspace(0, 30, 300) # 减量程度范围，取300个值  # 初始化最大值和对应的参数 max\_strength = -np.inf best\_params = (None, None, None)  # 遍历所有可能的参数组合 for x1 in x1\_range:  for x2 in x2\_range:  for x3 in x3\_range:  strength = regression\_function(x1, x2, x3)  if strength > max\_strength:  max\_strength = strength  best\_params = (x1, x2, x3)  print(f'Optimal Parameters:') print(f'树脂含量 = {best\_params[0]}') print(f'固化温度 = {best\_params[1]}') print(f'减量程度 = {best\_params[2]}') print(f'透湿率 = {max\_strength}') |

|  |
| --- |
| 附录16：问题三的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解最优力学性能的最佳工艺参数。 |
| import numpy as np  # 定义回归方程 def fracture\_strength(resin\_content, curing\_temp, reduction\_degree):  return 1840.838757 + (24.828373 \* resin\_content) + (-7.187842 \* curing\_temp) + (-79.999763 \* reduction\_degree) \  + (0.283718 \* resin\_content \*\* 2) + (-0.336917 \* resin\_content \* curing\_temp) + (  0.360148 \* resin\_content \* reduction\_degree) \  + (0.06519 \* curing\_temp \*\* 2) + (0.116717 \* curing\_temp \* reduction\_degree) + (  1.410783 \* reduction\_degree \*\* 2)  def elongation(resin\_content, curing\_temp, reduction\_degree):  return 0.117955 + (0.042182 \* resin\_content) + (0.008205 \* curing\_temp) + (-0.029932 \* reduction\_degree) \  + (-0.0003 \* resin\_content \*\* 2) + (-0.000305 \* resin\_content \* curing\_temp) + (  0.000273 \* resin\_content \* reduction\_degree) \  + (0.0 \* curing\_temp \*\* 2) + (2.5e-05 \* curing\_temp \* reduction\_degree) + (0.00055 \* reduction\_degree \*\* 2)  def tear\_strength(resin\_content, curing\_temp, reduction\_degree):  return 240.830123 + (-3.392196 \* resin\_content) + (-0.00856 \* curing\_temp) + (-15.491499 \* reduction\_degree) \  + (0.068722 \* resin\_content \*\* 2) + (-0.00807 \* resin\_content \* curing\_temp) + (  0.110209 \* resin\_content \* reduction\_degree) \  + (-0.001647 \* curing\_temp \*\* 2) + (0.067183 \* curing\_temp \* reduction\_degree) + (  0.110893 \* reduction\_degree \*\* 2)  # 目标函数 def objective(x):  resin\_content, curing\_temp, reduction\_degree = x  fracture = fracture\_strength(resin\_content, curing\_temp, reduction\_degree)  elongation\_val = elongation(resin\_content, curing\_temp, reduction\_degree)  tear = tear\_strength(resin\_content, curing\_temp, reduction\_degree)   # 归一化到 [0, 1] 范围  fracture\_normalized = (fracture - 1840) / (2000 - 1840)  elongation\_normalized = (elongation\_val - 0) / (0.2 - 0)  tear\_normalized = (tear - 240) / (300 - 240)   # 返回所有指标的负平均值，目标是最大化这些值  return - (fracture\_normalized + elongation\_normalized + tear\_normalized) / 3  # 遍历范围的网格搜索 def grid\_search(bounds, step\_size=1.0):  resin\_range = np.arange(bounds[0][0], bounds[0][1] + step\_size, step\_size)  temp\_range = np.arange(bounds[1][0], bounds[1][1] + step\_size, step\_size)  reduction\_range = np.arange(bounds[2][0], bounds[2][1] + step\_size, step\_size)   best\_score = float('inf')  best\_params = None   for resin\_content in resin\_range:  for curing\_temp in temp\_range:  for reduction\_degree in reduction\_range:  score = objective([resin\_content, curing\_temp, reduction\_degree])  if score < best\_score:  best\_score = score  best\_params = (resin\_content, curing\_temp, reduction\_degree)   return best\_params, -best\_score  # 变量范围 bounds = [(15, 30), (100, 130), (0, 30)]  # 执行网格搜索 best\_params, best\_score = grid\_search(bounds, step\_size=0.5)  # 输出结果 print(f"树脂含量: {best\_params[0]}") print(f"固化温度: {best\_params[1]}") print(f"减量程度: {best\_params[2]}")  # 计算优化后的目标值 print(f"断裂强力: {fracture\_strength(\*best\_params)}") print(f"断裂伸长量: {elongation(\*best\_params)}") print(f"撕裂强力: {tear\_strength(\*best\_params)}") |

|  |
| --- |
| 附录17：问题三的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解最优柔软性能的最佳工艺参数。 |
| import numpy as np  # 定义透气率回归方程 def air\_permeability(resin\_content, curing\_temp, reduction\_degree):  return 654.596534 + (0.042795 \* resin\_content) + (-4.425432 \* curing\_temp) + (-20.034932 \* reduction\_degree) \  + (-0.370575 \* resin\_content \*\* 2) + (0.227257 \* resin\_content \* curing\_temp) + (-0.135007 \* resin\_content \* reduction\_degree) \  + (-0.026606 \* curing\_temp \*\* 2) + (0.281106 \* curing\_temp \* reduction\_degree) + (-0.393256 \* reduction\_degree \*\* 2)  # 定义透湿率回归方程 def moisture\_permeability(resin\_content, curing\_temp, reduction\_degree):  return 10622.534531 + (-233.822679 \* resin\_content) + (-88.785737 \* curing\_temp) + (20.464182 \* reduction\_degree) \  + (2.374558 \* resin\_content \*\* 2) + (0.354642 \* resin\_content \* curing\_temp) + (1.728879 \* resin\_content \* reduction\_degree) \  + (0.386926 \* curing\_temp \*\* 2) + (-0.477996 \* curing\_temp \* reduction\_degree) + (0.063604 \* reduction\_degree \*\* 2)  # 目标函数 def objective(x):  resin\_content, curing\_temp, reduction\_degree = x  air\_perm = air\_permeability(resin\_content, curing\_temp, reduction\_degree)  moisture\_perm = moisture\_permeability(resin\_content, curing\_temp, reduction\_degree)   # 对数据标准化  air\_perm\_normalized = (air\_perm - 600) / (700 - 600) # 示例范围  moisture\_perm\_normalized = (moisture\_perm - 10000) / (12000 - 10000) # 示例范围   # 返回所有指标的负平均值，目标是最大化这些值  return - (air\_perm\_normalized + moisture\_perm\_normalized) / 2  # 遍历范围的网格搜索 def grid\_search(bounds, step\_size=1.0):  resin\_range = np.arange(bounds[0][0], bounds[0][1] + step\_size, step\_size)  temp\_range = np.arange(bounds[1][0], bounds[1][1] + step\_size, step\_size)  reduction\_range = np.arange(bounds[2][0], bounds[2][1] + step\_size, step\_size)   best\_score = float('inf')  best\_params = None   for resin\_content in resin\_range:  for curing\_temp in temp\_range:  for reduction\_degree in reduction\_range:  score = objective([resin\_content, curing\_temp, reduction\_degree])  if score < best\_score:  best\_score = score  best\_params = (resin\_content, curing\_temp, reduction\_degree)   return best\_params, -best\_score  # 变量范围 bounds = [(15, 30), (100, 130), (0, 30)]  # 执行网格搜索 best\_params, best\_score = grid\_search(bounds, step\_size=0.5)  # 输出结果 print(f"树脂含量: {best\_params[0]}") print(f"固化温度: {best\_params[1]}") print(f"减量程度: {best\_params[2]}")  # 计算优化后的目标值 print(f"透气率: {air\_permeability(\*best\_params)}") print(f"透湿率: {moisture\_permeability(\*best\_params)}") |

|  |
| --- |
| 附录18：问题三的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解最优的热湿舒适度的最佳工艺参数。 |
| import numpy as np  # 定义柔软度回归方程 def softness(resin\_content, curing\_temp, reduction\_degree):  return 1.49625 - (0.04875 \* resin\_content) + (0.005625 \* curing\_temp) + (0.067875 \* reduction\_degree)  # 定义折皱回复率回归方程 def wrinkling\_recovery\_rate(resin\_content, curing\_temp, reduction\_degree):  return 88.232665 + (2.362457 \* resin\_content) + (0.567772 \* curing\_temp) + (0.653567 \* reduction\_degree) \  + (-0.038973 \* resin\_content \*\* 2) + (-0.005583 \* resin\_content \* curing\_temp) + (0.004328 \* resin\_content \* reduction\_degree) \  + (-0.001423 \* curing\_temp \*\* 2) + (0.000496 \* curing\_temp \* reduction\_degree) + (-0.013223 \* reduction\_degree \*\* 2)  # 目标函数 def objective(x):  resin\_content, curing\_temp, reduction\_degree = x  softness\_val = softness(resin\_content, curing\_temp, reduction\_degree)  wrinkling\_rate\_val = wrinkling\_recovery\_rate(resin\_content, curing\_temp, reduction\_degree)   # 对数据标准化  softness\_normalized = (softness\_val - 1) / (2 - 1)  wrinkling\_rate\_normalized = (wrinkling\_rate\_val - 80) / (100 - 80)   # 返回所有指标的负平均值，目标是最大化这些值  return - (softness\_normalized + wrinkling\_rate\_normalized) / 2  # 遍历范围的网格搜索 def grid\_search(bounds, step\_size=1.0):  resin\_range = np.arange(bounds[0][0], bounds[0][1] + step\_size, step\_size)  temp\_range = np.arange(bounds[1][0], bounds[1][1] + step\_size, step\_size)  reduction\_range = np.arange(bounds[2][0], bounds[2][1] + step\_size, step\_size)   best\_score = float('inf')  best\_params = None   for resin\_content in resin\_range:  for curing\_temp in temp\_range:  for reduction\_degree in reduction\_range:  score = objective([resin\_content, curing\_temp, reduction\_degree])  if score < best\_score:  best\_score = score  best\_params = (resin\_content, curing\_temp, reduction\_degree)   return best\_params, -best\_score  # 变量范围 bounds = [(15, 30), (100, 130), (0, 30)]  # 执行网格搜索 best\_params, best\_score = grid\_search(bounds, step\_size=0.5)  # 输出结果 print(f"树脂含量: {best\_params[0]}") print(f"固化温度: {best\_params[1]}") print(f"减量程度: {best\_params[2]}")  # 计算优化后的目标值 print(f"柔软度: {softness(\*best\_params)}") print(f"折皱回复率: {wrinkling\_recovery\_rate(\*best\_params)}") |

|  |
| --- |
| 附录19：问题三的参考代码 |
| 介绍：该代码是用python编写的，使用粒子群算法求得的方程求解最优力学性能的最佳工艺参数。 |
| import pandas as pd import numpy as np from pyswarm import pso from sklearn.preprocessing import MinMaxScaler  # 读取Excel文件 df = pd.read\_excel('data.xlsx')  # 提取数据列 resin\_content = df['树脂含量'].values curing\_temperature = df['固化温度'].values reduction\_degree = df['减量程度'].values fracture\_strength = df['断裂强力'].values fracture\_elongation = df['断裂伸长量'].values tear\_strength = df['撕裂强力'].values  # 数据归一化 scaler = MinMaxScaler() target\_data = np.vstack([fracture\_strength, fracture\_elongation, tear\_strength]).T target\_data\_normalized = scaler.fit\_transform(target\_data)  # 反归一化 def denormalize(value, scaler):  return scaler.inverse\_transform(np.array([[value]]))[0, 0]  # 计算目标函数（最大化归一化后的断裂强力、断裂伸长量和撕裂强力的加权和） def objective\_function(x):  resin, temp, reduction = x  distances = np.abs(resin\_content - resin) + np.abs(curing\_temperature - temp) + np.abs(reduction\_degree - reduction)  idx = np.argmin(distances)   # 获取归一化后的目标值  normalized\_targets = target\_data\_normalized[idx]   # 目标是最大化这三个指标的加权和  return -np.sum(normalized\_targets) # 使用负号因为pso最小化目标函数  # 边界条件 lb = [resin\_content.min(), curing\_temperature.min(), reduction\_degree.min()] ub = [resin\_content.max(), curing\_temperature.max(), reduction\_degree.max()]  # 粒子群优化 xopt, fopt = pso(objective\_function, lb, ub, swarmsize=50, maxiter=100)  # 输出结果 optimal\_resin, optimal\_temp, optimal\_reduction = xopt optimal\_idx = np.argmin(np.abs(resin\_content - optimal\_resin) + np.abs(curing\_temperature - optimal\_temp) + np.abs(reduction\_degree - optimal\_reduction))  # 反归一化目标值 final\_targets\_normalized = target\_data\_normalized[optimal\_idx] final\_targets = scaler.inverse\_transform(final\_targets\_normalized.reshape(1, -1))  print(f"树脂含量: {optimal\_resin}") print(f"固化温度: {optimal\_temp}") print(f"减量程度: {optimal\_reduction}")  print(f"断裂强力: {final\_targets[0, 0]}") print(f"断裂伸长量: {final\_targets[0, 1]}") print(f"撕裂强力: {final\_targets[0, 2]}") |

|  |
| --- |
| 附录20：问题三的参考代码 |
| 介绍：该代码是用python编写的，使用粒子群算法求得的方程求解最优柔软性能的最佳工艺参数。 |
| import pandas as pd import numpy as np from pyswarm import pso from sklearn.preprocessing import MinMaxScaler  # 读取Excel文件 df = pd.read\_excel('data.xlsx')  # 提取数据列 resin\_content = df['树脂含量'].values curing\_temperature = df['固化温度'].values reduction\_degree = df['减量程度'].values softness = df['柔软度'].values wrinkle\_recovery\_rate = df['折皱回复率'].values  # 数据归一化 scaler = MinMaxScaler() target\_data = np.vstack([softness, wrinkle\_recovery\_rate]).T target\_data\_normalized = scaler.fit\_transform(target\_data)  # 反归一化 def denormalize(value, scaler):  return scaler.inverse\_transform(np.array([[value]]))[0, 0]  # 计算目标函数（最大化归一化后的柔软度和折皱回复率的加权和） def objective\_function(x):  resin, temp, reduction = x  distances = np.abs(resin\_content - resin) + np.abs(curing\_temperature - temp) + np.abs(reduction\_degree - reduction)  idx = np.argmin(distances)   # 获取归一化后的目标值  normalized\_targets = target\_data\_normalized[idx]   # 目标是最大化这两个指标的加权和  return -np.sum(normalized\_targets) # 使用负号因为pso最小化目标函数  # 边界条件 lb = [resin\_content.min(), curing\_temperature.min(), reduction\_degree.min()] ub = [resin\_content.max(), curing\_temperature.max(), reduction\_degree.max()]  # 粒子群优化 xopt, fopt = pso(objective\_function, lb, ub, swarmsize=50, maxiter=100)  # 输出结果 optimal\_resin, optimal\_temp, optimal\_reduction = xopt optimal\_idx = np.argmin(np.abs(resin\_content - optimal\_resin) + np.abs(curing\_temperature - optimal\_temp) + np.abs(reduction\_degree - optimal\_reduction))  # 反归一化目标值 final\_targets\_normalized = target\_data\_normalized[optimal\_idx] final\_targets = scaler.inverse\_transform(final\_targets\_normalized.reshape(1, -1))  print(f"树脂含量: {optimal\_resin}") print(f"固化温度: {optimal\_temp}") print(f"减量程度: {optimal\_reduction}")  print(f"柔软度: {final\_targets[0, 0]}") print(f"折皱回复率: {final\_targets[0, 1]}") |

|  |
| --- |
| 附录21：问题三的参考代码 |
| 介绍：该代码是用python编写的，使用粒子群算法求得的方程求解最优的热湿舒适度的最佳工艺参数。 |
| import pandas as pd import numpy as np from pyswarm import pso from sklearn.preprocessing import MinMaxScaler  # 读取Excel文件 df = pd.read\_excel('data.xlsx')  # 提取数据列 resin\_content = df['树脂含量'].values curing\_temperature = df['固化温度'].values reduction\_degree = df['减量程度'].values air\_permeability = df['透气率'].values # 更新为透气率 moisture\_permeability = df['透湿率'].values # 更新为透湿率  # 数据归一化 scaler = MinMaxScaler() target\_data = np.vstack([air\_permeability, moisture\_permeability]).T target\_data\_normalized = scaler.fit\_transform(target\_data)  # 反归一化 def denormalize(value, scaler):  return scaler.inverse\_transform(np.array([[value]]))[0, 0]  # 计算目标函数（最大化归一化后的透气率和透湿率的加权和） def objective\_function(x):  resin, temp, reduction = x  distances = np.abs(resin\_content - resin) + np.abs(curing\_temperature - temp) + np.abs(reduction\_degree - reduction)  idx = np.argmin(distances)   # 获取归一化后的目标值  normalized\_targets = target\_data\_normalized[idx]   # 目标是最大化这两个指标的加权和  return -np.sum(normalized\_targets) # 使用负号因为pso最小化目标函数  # 边界条件 lb = [resin\_content.min(), curing\_temperature.min(), reduction\_degree.min()] ub = [resin\_content.max(), curing\_temperature.max(), reduction\_degree.max()]  # 粒子群优化 xopt, fopt = pso(objective\_function, lb, ub, swarmsize=50, maxiter=100)  # 输出结果 optimal\_resin, optimal\_temp, optimal\_reduction = xopt optimal\_idx = np.argmin(np.abs(resin\_content - optimal\_resin) + np.abs(curing\_temperature - optimal\_temp) + np.abs(reduction\_degree - optimal\_reduction))  # 反归一化目标值 final\_targets\_normalized = target\_data\_normalized[optimal\_idx] final\_targets = scaler.inverse\_transform(final\_targets\_normalized.reshape(1, -1))  print(f"树脂含量: {optimal\_resin}") print(f"固化温度: {optimal\_temp}") print(f"减量程度: {optimal\_reduction}")  print(f"透气率: {final\_targets[0, 0]}") print(f"透湿率: {final\_targets[0, 1]}") |

|  |
| --- |
| 附录22：问题三的参考代码 |
| 介绍：该代码是用python编写的，使用多元线性回归模型求得的方程求解人造革7项指标综合性能最优所需要的工艺参数。 |
| import numpy as np  # 断裂强力回归方程 def fracture\_strength(resin\_content, curing\_temp, reduction\_degree):  return 1840.003261 + (0.269734 \* resin\_content) + (-2.442865 \* curing\_temp) + (-6.512045 \* reduction\_degree) \  + (-0.002773 \* resin\_content \*\* 2) + (0.002722 \* resin\_content \* curing\_temp) + (  -0.020057 \* resin\_content \* reduction\_degree) \  + (0.049659 \* curing\_temp \*\* 2) + (0.034738 \* curing\_temp \* reduction\_degree) + (  0.015217 \* reduction\_degree \*\* 2)  # 断裂伸长量回归方程 def elongation(resin\_content, curing\_temp, reduction\_degree):  return 0.074934 + (-0.000312 \* resin\_content) + (0.000065 \* curing\_temp) + (0.000410 \* reduction\_degree) \  + (0.000000 \* resin\_content \*\* 2) + (-0.000000 \* resin\_content \* curing\_temp) + (  -0.000000 \* resin\_content \* reduction\_degree) \  + (0.000000 \* curing\_temp \*\* 2) + (0.000000 \* curing\_temp \* reduction\_degree) + (  0.000000 \* reduction\_degree \*\* 2)  # 撕裂强力回归方程 def tear\_strength(resin\_content, curing\_temp, reduction\_degree):  return 240.809183 + (-3.401894 \* resin\_content) + (0.551013 \* curing\_temp) + (-3.170159 \* reduction\_degree) \  + (-0.022140 \* resin\_content \*\* 2) + (-0.020169 \* resin\_content \* curing\_temp) + (  0.073350 \* resin\_content \* reduction\_degree) \  + (-0.008494 \* curing\_temp \*\* 2) + (0.014356 \* curing\_temp \* reduction\_degree) + (  -0.008118 \* reduction\_degree \*\* 2)  # 透气率回归方程 def air\_permeability(resin\_content, curing\_temp, reduction\_degree):  return 654.596534 + (0.042795 \* resin\_content) + (-4.425432 \* curing\_temp) + (-20.034932 \* reduction\_degree) \  + (-0.370575 \* resin\_content \*\* 2) + (0.227257 \* resin\_content \* curing\_temp) + (  -0.135007 \* resin\_content \* reduction\_degree) \  + (-0.026606 \* curing\_temp \*\* 2) + (0.281106 \* curing\_temp \* reduction\_degree) + (  -0.393256 \* reduction\_degree \*\* 2)  # 透湿率回归方程 def moisture\_permeability(resin\_content, curing\_temp, reduction\_degree):  return 10622.534531 + (-233.822679 \* resin\_content) + (-88.785737 \* curing\_temp) + (20.464182 \* reduction\_degree) \  + (2.374558 \* resin\_content \*\* 2) + (0.354642 \* resin\_content \* curing\_temp) + (  1.728879 \* resin\_content \* reduction\_degree) \  + (0.386926 \* curing\_temp \*\* 2) + (-0.477996 \* curing\_temp \* reduction\_degree) + (  0.063604 \* reduction\_degree \*\* 2)  # 柔软度回归方程 def softness(resin\_content, curing\_temp, reduction\_degree):  return 1.49625 + (-0.04875 \* resin\_content) + (0.005625 \* curing\_temp) + (0.067875 \* reduction\_degree)  # 折皱回复率回归方程 def wrinkle\_recovery(resin\_content, curing\_temp, reduction\_degree):  return 88.232665 + (2.362457 \* resin\_content) + (0.567772 \* curing\_temp) + (0.653567 \* reduction\_degree) \  + (-0.038973 \* resin\_content \*\* 2) + (-0.005583 \* resin\_content \* curing\_temp) + (  0.004328 \* resin\_content \* reduction\_degree) \  + (-0.001423 \* curing\_temp \*\* 2) + (0.000496 \* curing\_temp \* reduction\_degree) + (  -0.013223 \* reduction\_degree \*\* 2)  # 目标函数 def objective(x):  resin\_content, curing\_temp, reduction\_degree = x  fracture = fracture\_strength(resin\_content, curing\_temp, reduction\_degree)  elongation\_val = elongation(resin\_content, curing\_temp, reduction\_degree)  tear = tear\_strength(resin\_content, curing\_temp, reduction\_degree)  air\_perm = air\_permeability(resin\_content, curing\_temp, reduction\_degree)  moisture\_perm = moisture\_permeability(resin\_content, curing\_temp, reduction\_degree)  soft = softness(resin\_content, curing\_temp, reduction\_degree)  wrinkle = wrinkle\_recovery(resin\_content, curing\_temp, reduction\_degree)   # 归一化到 [0, 1] 范围  fracture\_normalized = (fracture - 1840) / (2000 - 1840)  elongation\_normalized = (elongation\_val - 0) / (0.2 - 0)  tear\_normalized = (tear - 240) / (300 - 240)  air\_perm\_normalized = (air\_perm - 654) / (700 - 654)  moisture\_perm\_normalized = (moisture\_perm - 10622) / (11000 - 10622)  soft\_normalized = (soft - 1.49625) / (2 - 1.49625)  wrinkle\_normalized = (wrinkle - 88.232665) / (100 - 88.232665)   # 返回所有指标的负平均值  return - (fracture\_normalized + elongation\_normalized + tear\_normalized + air\_perm\_normalized +  moisture\_perm\_normalized + soft\_normalized + wrinkle\_normalized) / 7  # 网格搜索函数 def grid\_search(bounds, step\_size=0.5):  best\_score = float('inf')  best\_params = None  for resin\_content in np.arange(bounds[0][0], bounds[0][1], step\_size):  for curing\_temp in np.arange(bounds[1][0], bounds[1][1], step\_size):  for reduction\_degree in np.arange(bounds[2][0], bounds[2][1], step\_size):  score = objective((resin\_content, curing\_temp, reduction\_degree))  if score < best\_score:  best\_score = score  best\_params = (resin\_content, curing\_temp, reduction\_degree)  return best\_params, best\_score  # 定义搜索范围 bounds = [(15, 30), (100, 130), (0, 30)]  # 执行网格搜索 best\_params, best\_score = grid\_search(bounds, step\_size=0.5)  # 输出结果 print(f"树脂含量: {best\_params[0]}") print(f"固化温度: {best\_params[1]}") print(f"减量程度: {best\_params[2]}")  # 计算优化后的目标值 print(f"断裂强力: {fracture\_strength(\*best\_params)}") print(f"断裂伸长量: {elongation(\*best\_params)\*10}") print(f"撕裂强力: {tear\_strength(\*best\_params)}") print(f"透气率: {air\_permeability(\*best\_params)}") print(f"透湿率: {moisture\_permeability(\*best\_params)}") print(f"柔软度: {softness(\*best\_params)}") print(f"折皱回复率: {wrinkle\_recovery(\*best\_params)}") |

|  |
| --- |
| 附录23：问题三的参考代码 |
| 介绍：该代码是用python编写的，使用粒子群优化算法求得的方程求解人造革7项指标综合性能最优所需要的工艺参数。 |
| import pandas as pd import numpy as np from pyswarm import pso from sklearn.preprocessing import MinMaxScaler  # 读取Excel文件 df = pd.read\_excel('data.xlsx')  # 提取数据列 resin\_content = df['树脂含量'].values curing\_temperature = df['固化温度'].values reduction\_degree = df['减量程度'].values breaking\_strength = df['断裂强力'].values breaking\_extension = df['断裂伸长量'].values tear\_strength = df['撕裂强力'].values air\_permeability = df['透气率'].values moisture\_permeability = df['透湿率'].values softness = df['柔软度'].values wrinkle\_recovery\_rate = df['折皱回复率'].values  # 数据归一化 scaler = MinMaxScaler() target\_data = np.vstack([  breaking\_strength, breaking\_extension, tear\_strength, air\_permeability,  moisture\_permeability, softness, wrinkle\_recovery\_rate ]).T target\_data\_normalized = scaler.fit\_transform(target\_data)  # 反归一化 def denormalize(value, scaler):  return scaler.inverse\_transform(np.array([[value]]))[0, 0]  # 计算目标函数（最大化归一化后的指标的加权和） def objective\_function(x):  resin, temp, reduction = x  distances = np.abs(resin\_content - resin) + np.abs(curing\_temperature - temp) + np.abs(reduction\_degree - reduction)  idx = np.argmin(distances)   # 获取归一化后的目标值  normalized\_targets = target\_data\_normalized[idx]   # 目标是最大化这几个指标的加权和  return -np.sum(normalized\_targets) # 使用负号因为pso最小化目标函数  # 边界条件 lb = [resin\_content.min(), curing\_temperature.min(), reduction\_degree.min()] ub = [resin\_content.max(), curing\_temperature.max(), reduction\_degree.max()]  # 粒子群优化 xopt, fopt = pso(objective\_function, lb, ub, swarmsize=50, maxiter=100)  # 输出结果 optimal\_resin, optimal\_temp, optimal\_reduction = xopt optimal\_idx = np.argmin(np.abs(resin\_content - optimal\_resin) + np.abs(curing\_temperature - optimal\_temp) + np.abs(reduction\_degree - optimal\_reduction))  # 反归一化目标值 final\_targets\_normalized = target\_data\_normalized[optimal\_idx] final\_targets = scaler.inverse\_transform(final\_targets\_normalized.reshape(1, -1))  print(f"树脂含量: {optimal\_resin}") print(f"固化温度: {optimal\_temp}") print(f"减量程度: {optimal\_reduction}")  print(f"断裂强力: {final\_targets[0, 0]}") print(f"断裂伸长量: {final\_targets[0, 1]}") print(f"撕裂强力: {final\_targets[0, 2]}") print(f"透气率: {final\_targets[0, 3]}") print(f"透湿率: {final\_targets[0, 4]}") print(f"柔软度: {final\_targets[0, 5]}") print(f"折皱回复率: {final\_targets[0, 6]}") |

|  |
| --- |
| 附录24：问题四的参考代码 |
| 介绍：该代码是用python编写的，使用多元回归模型求得的方程求解权重调整之后的人造革7项指标综合性能最优所需要的工艺参数。 |
| import numpy as np  # 断裂强力回归方程 def fracture\_strength(resin\_content, curing\_temp, reduction\_degree):  return 1840.003261 + (0.269734 \* resin\_content) + (-2.442865 \* curing\_temp) + (-6.512045 \* reduction\_degree) \  + (-0.002773 \* resin\_content \*\* 2) + (0.002722 \* resin\_content \* curing\_temp) + (  -0.020057 \* resin\_content \* reduction\_degree) \  + (0.049659 \* curing\_temp \*\* 2) + (0.034738 \* curing\_temp \* reduction\_degree) + (  0.015217 \* reduction\_degree \*\* 2)  # 断裂伸长量回归方程 def elongation(resin\_content, curing\_temp, reduction\_degree):  return 0.074934 + (-0.000312 \* resin\_content) + (0.000065 \* curing\_temp) + (0.000410 \* reduction\_degree) \  + (0.000000 \* resin\_content \*\* 2) + (-0.000000 \* resin\_content \* curing\_temp) + (  -0.000000 \* resin\_content \* reduction\_degree) \  + (0.000000 \* curing\_temp \*\* 2) + (0.000000 \* curing\_temp \* reduction\_degree) + (  0.000000 \* reduction\_degree \*\* 2)  # 撕裂强力回归方程 def tear\_strength(resin\_content, curing\_temp, reduction\_degree):  return 240.809183 + (-3.401894 \* resin\_content) + (0.551013 \* curing\_temp) + (-3.170159 \* reduction\_degree) \  + (-0.022140 \* resin\_content \*\* 2) + (-0.020169 \* resin\_content \* curing\_temp) + (  0.073350 \* resin\_content \* reduction\_degree) \  + (-0.008494 \* curing\_temp \*\* 2) + (0.014356 \* curing\_temp \* reduction\_degree) + (  -0.008118 \* reduction\_degree \*\* 2)  # 透气率回归方程 def air\_permeability(resin\_content, curing\_temp, reduction\_degree):  return 654.596534 + (0.042795 \* resin\_content) + (-4.425432 \* curing\_temp) + (-20.034932 \* reduction\_degree) \  + (-0.370575 \* resin\_content \*\* 2) + (0.227257 \* resin\_content \* curing\_temp) + (  -0.135007 \* resin\_content \* reduction\_degree) \  + (-0.026606 \* curing\_temp \*\* 2) + (0.281106 \* curing\_temp \* reduction\_degree) + (  -0.393256 \* reduction\_degree \*\* 2)  # 透湿率回归方程 def moisture\_permeability(resin\_content, curing\_temp, reduction\_degree):  return 10622.534531 + (-233.822679 \* resin\_content) + (-88.785737 \* curing\_temp) + (20.464182 \* reduction\_degree) \  + (2.374558 \* resin\_content \*\* 2) + (0.354642 \* resin\_content \* curing\_temp) + (  1.728879 \* resin\_content \* reduction\_degree) \  + (0.386926 \* curing\_temp \*\* 2) + (-0.477996 \* curing\_temp \* reduction\_degree) + (  0.063604 \* reduction\_degree \*\* 2)  # 柔软度回归方程 def softness(resin\_content, curing\_temp, reduction\_degree):  return 1.49625 + (-0.04875 \* resin\_content) + (0.005625 \* curing\_temp) + (0.067875 \* reduction\_degree)  # 折皱回复率回归方程 def wrinkle\_recovery(resin\_content, curing\_temp, reduction\_degree):  return 88.232665 + (2.362457 \* resin\_content) + (0.567772 \* curing\_temp) + (0.653567 \* reduction\_degree) \  + (-0.038973 \* resin\_content \*\* 2) + (-0.005583 \* resin\_content \* curing\_temp) + (  0.004328 \* resin\_content \* reduction\_degree) \  + (-0.001423 \* curing\_temp \*\* 2) + (0.000496 \* curing\_temp \* reduction\_degree) + (  -0.013223 \* reduction\_degree \*\* 2)  # 目标函数，增加透气率和透湿率的权重 def objective(x):  resin\_content, curing\_temp, reduction\_degree = x  fracture = fracture\_strength(resin\_content, curing\_temp, reduction\_degree)  elongation\_val = elongation(resin\_content, curing\_temp, reduction\_degree)  tear = tear\_strength(resin\_content, curing\_temp, reduction\_degree)  air\_perm = air\_permeability(resin\_content, curing\_temp, reduction\_degree)  moisture\_perm = moisture\_permeability(resin\_content, curing\_temp, reduction\_degree)  soft = softness(resin\_content, curing\_temp, reduction\_degree)  wrinkle = wrinkle\_recovery(resin\_content, curing\_temp, reduction\_degree)   # 归一化到 [0, 1] 范围  fracture\_normalized = (fracture - 1840) / (2000 - 1840)  elongation\_normalized = (elongation\_val - 0) / (0.2 - 0)  tear\_normalized = (tear - 240) / (300 - 240)  air\_perm\_normalized = (air\_perm - 654) / (700 - 654)  moisture\_perm\_normalized = (moisture\_perm - 10622) / (11000 - 10622)  soft\_normalized = (soft - 1.49625) / (2 - 1.49625)  wrinkle\_normalized = (wrinkle - 88.232665) / (100 - 88.232665)   # 根据权重调整  weighted\_score = (fracture\_normalized + elongation\_normalized + tear\_normalized +  2 \* air\_perm\_normalized + 2 \* moisture\_perm\_normalized +  soft\_normalized + wrinkle\_normalized) / 8   # 返回负值，因为优化算法是最小化目标函数  return -weighted\_score  # 网格搜索函数 def grid\_search(bounds, step\_size=0.5):  best\_score = float('inf')  best\_params = None  for resin\_content in np.arange(bounds[0][0], bounds[0][1], step\_size):  for curing\_temp in np.arange(bounds[1][0], bounds[1][1], step\_size):  for reduction\_degree in np.arange(bounds[2][0], bounds[2][1], step\_size):  score = objective((resin\_content, curing\_temp, reduction\_degree))  if score < best\_score:  best\_score = score  best\_params = (resin\_content, curing\_temp, reduction\_degree)  return best\_params, best\_score  # 定义搜索范围 bounds = [(15, 30), (100, 130), (0, 30)]  # 执行网格搜索 best\_params, best\_score = grid\_search(bounds, step\_size=0.5)  # 输出结果 print(f"树脂含量: {best\_params[0]}") print(f"固化温度: {best\_params[1]}") print(f"减量程度: {best\_params[2]}")  # 计算优化后的目标值 print(f"断裂强力: {fracture\_strength(\*best\_params)}") print(f"断裂伸长量: {elongation(\*best\_params)\*100}") print(f"撕裂强力: {tear\_strength(\*best\_params)}") print(f"透气率: {air\_permeability(\*best\_params)}") print(f"透湿率: {moisture\_permeability(\*best\_params)}") print(f"柔软度: {softness(\*best\_params)}") print(f"折皱回复率: {wrinkle\_recovery(\*best\_params)}") |

|  |
| --- |
| 附录24：问题四的参考代码 |
| 介绍：该代码是用python编写的，使用粒子群算法求得的方程求解权重调整之后的人造革7项指标综合性能最优所需要的工艺参数。 |
| import pandas as pd  import numpy as np  from pyswarm import pso  from sklearn.preprocessing import MinMaxScaler  # 读取Excel文件  df = pd.read\_excel('data.xlsx')  # 提取数据列  resin\_content = df['树脂含量'].values  curing\_temperature = df['固化温度'].values  reduction\_degree = df['减量程度'].values  breaking\_strength = df['断裂强力'].values  breaking\_extension = df['断裂伸长量'].values  tear\_strength = df['撕裂强力'].values  air\_permeability = df['透气率'].values  moisture\_permeability = df['透湿率'].values  softness = df['柔软度'].values  wrinkle\_recovery\_rate = df['折皱回复率'].values  # 数据归一化  scaler = MinMaxScaler()  target\_data = np.vstack([  breaking\_strength, breaking\_extension, tear\_strength, air\_permeability,  moisture\_permeability, softness, wrinkle\_recovery\_rate  ]).T  target\_data\_normalized = scaler.fit\_transform(target\_data)  # 反归一化  def denormalize(value, scaler):  return scaler.inverse\_transform(np.array([[value]]))[0, 0]  # 计算目标函数（最大化归一化后的指标的加权和）  def objective\_function(x):  resin, temp, reduction = x  distances = np.abs(resin\_content - resin) + np.abs(curing\_temperature - temp) + np.abs(reduction\_degree - reduction)  idx = np.argmin(distances)  # 获取归一化后的目标值  normalized\_targets = target\_data\_normalized[idx]  # 设定透气率和透湿率的权重  weight\_air\_permeability = 2.0  weight\_moisture\_permeability = 2.0  # 计算目标值的加权和  weighted\_sum = (normalized\_targets[0] +  normalized\_targets[1] +  normalized\_targets[2] +  weight\_air\_permeability \* normalized\_targets[3] +  weight\_moisture\_permeability \* normalized\_targets[4] +  normalized\_targets[5] +  normalized\_targets[6])  return -weighted\_sum # 使用负号因为pso最小化目标函数  # 边界条件  lb = [resin\_content.min(), curing\_temperature.min(), reduction\_degree.min()]  ub = [resin\_content.max(), curing\_temperature.max(), reduction\_degree.max()]  # 粒子群优化  xopt, fopt = pso(objective\_function, lb, ub, swarmsize=50, maxiter=100)  # 输出结果  optimal\_resin, optimal\_temp, optimal\_reduction = xopt  optimal\_idx = np.argmin(np.abs(resin\_content - optimal\_resin) + np.abs(curing\_temperature - optimal\_temp) + np.abs(reduction\_degree - optimal\_reduction))  # 反归一化目标值  final\_targets\_normalized = target\_data\_normalized[optimal\_idx]  final\_targets = scaler.inverse\_transform(final\_targets\_normalized.reshape(1, -1))  print(f"树脂含量: {optimal\_resin}")  print(f"固化温度: {optimal\_temp}")  print(f"减量程度: {optimal\_reduction}")  print(f"断裂强力: {final\_targets[0, 0]}")  print(f"断裂伸长量: {final\_targets[0, 1]}")  print(f"撕裂强力: {final\_targets[0, 2]}")  print(f"透气率: {final\_targets[0, 3]}")  print(f"透湿率: {final\_targets[0, 4]}")  print(f"柔软度: {final\_targets[0, 5]}")  print(f"折皱回复率: {final\_targets[0, 6]}") |