下面给你提供一个完整的、通用的代码示例，演示如何从仿真数据和真实数据中识别关键波段（特征），分析 CD 值与波段强度之间的关系，并比较真实数据与仿真数据的趋势差异。

你可以按下面步骤进行（Python 代码）：

完整代码示例：

假设数据结构为：

• 光谱数据（X）：每一行是一个样本，每一列是某个波段的强度。

• CD 值（y）：每个样本对应的目标值（连续数值）。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import mutual_info_regression
from scipy.stats import pearsonr

# 假设仿真数据为：X_sim, y_sim
# 假设真实数据为：X_real, y_real
# 假设波长为 wavelengths（例如：[400, 401, ..., 800] nm）

# 示例数据加载（这里请换成你实际的数据）
# X_sim, y_sim = ...
# X_real, y_real = ...
# wavelengths = np.arange(400, 801)  # 假设 400 到 800 nm 的光谱

# --- 第一步：特征重要性分析（仿真数据）---
def feature_importance_analysis(X, y, wavelengths, top_n=10):
    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    rf.fit(X, y)
    importances = rf.feature_importances_

    # 选出最重要的 top_n 个波段
    indices = np.argsort(importances)[::-1][:top_n]
    important_wavelengths = wavelengths[indices]

    plt.figure(figsize=(10,5))
    plt.bar(important_wavelengths, importances[indices])
    plt.xlabel('Wavelength (nm)')
    plt.ylabel('Feature Importance')
    plt.title('Top Important Wavelengths from Simulation Data')
    plt.show()
```

```
    return important_wavelengths

# 使用仿真数据进行特征重要性分析
important_wavelengths = feature_importance_analysis(X_sim, y_sim,
wavelengths)

# --- 第二步：趋势分析（对比仿真 vs. 真实）---
def plot_trend_comparison(X_sim, y_sim, X_real, y_real, wavelengths,
selected_wavelengths):
    for wl in selected_wavelengths:
        idx = np.where(wavelengths == wl)[0][0]
        plt.figure(figsize=(7,5))
        plt.scatter(y_sim, X_sim[:, idx], alpha=0.3, label='Simulation',
color='blue')
        plt.scatter(y_real, X_real[:, idx], alpha=0.9, label='Real',
color='red')
        plt.xlabel('CD Value')
        plt.ylabel(f'Intensity at {wl} nm')
        plt.title(f'Comparison at Wavelength {wl} nm')
        plt.legend()
        plt.show()

# 对比趋势
plot_trend_comparison(X_sim, y_sim, X_real, y_real, wavelengths,
important_wavelengths)

# --- 第三步：定量趋势差异分析（相关性对比）---
def quantitative_difference(X_sim, y_sim, X_real, y_real, wavelengths,
selected_wavelengths):
    results = []
    for wl in selected_wavelengths:
        idx = np.where(wavelengths == wl)[0][0]
        # 仿真数据相关性
        corr_sim, _ = pearsonr(X_sim[:, idx], y_sim)
        # 真实数据相关性
        corr_real, _ = pearsonr(X_real[:, idx], y_real)
        results.append({
            'Wavelength': wl,
            'Corr_sim': corr_sim,
            'Corr_real': corr_real,
            'Corr_diff': abs(corr_sim - corr_real)
        })
    df_results = pd.DataFrame(results)
    print(df_results.sort_values(by='Corr_diff', ascending=False))
    return df_results

# 定量分析相关性差异
df_diff = quantitative_difference(X_sim, y_sim, X_real, y_real,
wavelengths, important_wavelengths)
```

上述代码的功能：

• **第一步：**

使用仿真数据训练一个随机森林回归器，自动识别最重要的几个波段。

• **第二步：**

绘制真实数据和仿真数据在关键波段上的 CD 值与波段强度趋势图，直观看出域差异。

• **第三步：**

进一步定量分析仿真与真实数据的趋势差异，通过比较相关系数发现最明显差异的波段。

如何进一步利用上述结果？

上述分析完成后，你可以：

• 对发现趋势有明显差异的波段（比如明显的偏差、尺度差异）做简单的数据校正或变换（如线性或非线性拟合校正）。

• 再将处理后的数据应用于迁移学习模型中，以提高真实数据的回归精度。

下一步建议：

1. **替换示例数据**

将上述示例中的数据加载部分替换为你自己的数据加载方式。

2. **运行上述代码，获得关键波段和差异趋势**。

3. 根据趋势差异，明确修正策略，执行特征空间变换或数据修正，提升模型性能。

以上代码和思路可直接用于实践中，后续有任何问题，随时告诉我。

以下给出一个简单、清晰的**数据校正（趋势修正）和迁移学习（微调）**的代码实现，具体包括：

• **趋势差异的拟合和修正**（仿真数据→真实数据）。

• **用修正后的数据进行迁移学习**（仿真预训练→真实数据微调）。

第一步：趋势修正（波段校正）

对前面步骤发现的趋势差异大的关键波段（假设已明确），用简单的**多项式拟合**或**线性回归**修正：

```
from sklearn.linear_model import LinearRegression

def correct_simulation_data(X_sim, y_sim, X_real, y_real, wavelengths,
selected_wavelengths):
    X_sim_corrected = X_sim.copy()

    for wl in selected_wavelengths:
        idx = np.where(wavelengths == wl)[0][0]

        # 用真实数据拟合真实关系
        model_real = LinearRegression().fit(y_real.reshape(-1,1), X_real[:,
idx])

        # 用仿真数据拟合仿真关系
        model_sim = LinearRegression().fit(y_sim.reshape(-1,1), X_sim[:,
idx])

        # 计算仿真到真实的校正关系
        # 真实强度 = a_real * CD + b_real
        # 仿真强度 = a_sim * CD + b_sim
        # 校正：X_corrected = X_sim - (仿真趋势 - 真实趋势)

        # 预测仿真和真实趋势的差值
        trend_diff = (model_sim.predict(y_sim.reshape(-1,1)) -
                      model_real.predict(y_sim.reshape(-1,1)))

        # 仿真数据减去趋势差异，完成校正
        X_sim_corrected[:, idx] -= trend_diff

    return X_sim_corrected

# 假设已知重要波段为 important_wavelengths
X_sim_corrected = correct_simulation_data(X_sim, y_sim, X_real, y_real,
wavelengths, important_wavelengths)
```

以上步骤，将仿真数据修正到更贴近真实数据的分布。

第二步：迁移学习建模（预训练→微调）

典型实现过程为：

1. 使用修正后的仿真数据**预训练**模型。

2. 使用少量真实数据进行**微调**。

以简单的深度学习模型（如 1D-CNN）为例实现迁移：

```
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers

# 定义简单的 1D-CNN 回归模型
def build_model(input_shape):
    model = models.Sequential([
        layers.Conv1D(32, kernel_size=3, activation='relu',
input_shape=input_shape),
        layers.MaxPooling1D(2),
        layers.Conv1D(64, kernel_size=3, activation='relu'),
        layers.GlobalAveragePooling1D(),
        layers.Dense(32, activation='relu'),
        layers.Dense(1)
    ])
    model.compile(loss='mse', optimizer='adam')
    return model

# 数据准备
X_sim_corrected_exp = X_sim_corrected[..., np.newaxis]  # CNN 输入需增加通道维
X_real_exp = X_real[..., np.newaxis]

# --- 仿真数据预训练 ---
model = build_model(input_shape=X_sim_corrected_exp.shape[1:])
model.fit(X_sim_corrected_exp, y_sim, epochs=50, batch_size=64,
validation_split=0.1)

# --- 真实数据微调 ---
# 用较小的学习率进行微调
model.compile(loss='mse', optimizer=optimizers.Adam(1e-4))

model.fit(X_real_exp, y_real, epochs=20, batch_size=4,
validation_split=0.2)
```

第三步：评估模型效果（真实数据测试）

使用留一法或交叉验证评估：

```
from sklearn.metrics import mean_absolute_error

# 模型预测真实数据
y_pred_real = model.predict(X_real_exp).flatten()

# 评估指标
mae = mean_absolute_error(y_real, y_pred_real)
print(f"Real Data MAE after transfer learning: {mae:.4f}")

# 结果绘图对比
plt.figure(figsize=(7,5))
plt.scatter(y_real, y_pred_real, c='red', label='Predicted vs True')
plt.plot([y_real.min(), y_real.max()], [y_real.min(), y_real.max()], 'k--')
plt.xlabel('True CD Value')
plt.ylabel('Predicted CD Value')
plt.title('Real Data Predictions after Transfer Learning')
plt.legend()
plt.show()
```

✅ 代码使用流程总结：

步骤	功能	说明
1	趋势差异修正	修正仿真数据，使仿真数据贴近真实数据
2	仿真数据预训练模型	使用修正数据训练
3	真实数据微调模型	用少量真实数据微调模型，获得更准确预测

📌 注意事项与后续优化：

• 若趋势差异较为复杂，可考虑更复杂的回归（多项式回归或非线性回归）。

• 微调时要格外注意避免过拟合，可增加 early stopping。

• 在真实数据极少时（<10 条），尽量使用更简单的模型结构，避免过拟合。

你可以直接使用以上代码实现『**数据修正 + 迁移学习微调**』流程，进一步提高模型精度。

这里的校正关系，具体推导和计算方式如下：

📌 **原理解析（推导过程）：**

你有两组数据（仿真与真实），在**同一波段**下：

• 仿真数据趋势为：

$$X_{\text{sim}} = a_{\text{sim}} \cdot \text{CD} + b_{\text{sim}}$$

• 真实数据趋势为：

$$X_{\text{real}} = a_{\text{real}} \cdot \text{CD} + b_{\text{real}}$$

但实际情况是，你想让仿真数据“贴近”真实数据趋势，以更好地为真实数据的回归任务服务，因此需要修正仿真数据。

🚩 **修正的推导思路：**

希望仿真数据修正后的新趋势（记为 $X_{\text{sim\_corrected}}$）满足：

$$X_{\text{sim\_corrected}} \approx X_{\text{real}}$$

即：

$$X_{\text{sim\_corrected}} = X_{\text{sim}} - \text{趋势差异}$$

这里『趋势差异』定义为在**相同 CD 值**下，仿真和真实数据趋势（线性关系）的差值：

$$\text{趋势差异} = X_{\text{sim 趋势预测}} - X_{\text{real 趋势预测}}$$

用公式写清楚，就是在每个 CD 值对应的点上计算：

$$X_{\text{sim\_corrected}} = X_{\text{sim}} - \left[(a_{\text{sim}} \cdot \text{CD} + b_{\text{sim}}) - (a_{\text{real}} \cdot \text{CD} + b_{\text{real}}) \right]$$

即：

$$X_{\text{sim\_corrected}} = X_{\text{sim}} - (a_{\text{sim}} - a_{\text{real}})\cdot\text{CD} - (b_{\text{sim}} - b_{\text{real}})$$

代码如何实现这个公式的？

对应的 Python 代码解释（与你之前的代码一致）：

```
# 拟合真实趋势
model_real = LinearRegression().fit(y_real.reshape(-1,1), X_real[:, idx])
# 拟合仿真趋势
model_sim = LinearRegression().fit(y_sim.reshape(-1,1), X_sim[:, idx])

# 预测仿真趋势的数值（仿真趋势线上的点）
X_sim_trend = model_sim.predict(y_sim.reshape(-1,1))

# 预测真实趋势的数值（真实趋势线上的点）
X_real_trend = model_real.predict(y_sim.reshape(-1,1))

# 趋势差异：仿真趋势 - 真实趋势
trend_diff = X_sim_trend - X_real_trend

# 校正：仿真数据减去趋势差异
X_sim_corrected[:, idx] -= trend_diff
```

• 上述代码的实质即**在每个仿真数据的 CD 点上，计算并减去了“仿真与真实趋势的差异”**。

总结（核心思想）：

• 拟合真实和仿真数据趋势。

• 找出仿真数据与真实数据的趋势差异。

• 将这个差异应用于仿真数据，实现校正。

通过这个过程，你使仿真数据的趋势更贴近真实数据，从而提高了用仿真数据训练的模型对真实数据的预测精度。

这就是你代码中“趋势差异校正”的本质和详细计算过程的解析。

这里提供几种常见且有效的校正备选方案（线性与非线性），并给出详细的 Python 代码实现：

备选方案概述：

假设你已经明确了哪些波段趋势差异显著，针对这些波段：

方法	场景	说明
1. 线性回归校正	线性偏差明显	仿真-真实趋势存在明显线性差异
2. 多项式回归校正	非线性差异明显	趋势存在弯曲、非线性差异
3. 分段线性校正	趋势在不同 CD 范围表现不同	适合趋势有折点
4. 基于高斯过程回归(GPR)校正	复杂趋势，少量真实数据	非线性复杂趋势适合

代码示例：

▶️ 方法 1: 线性回归校正

```
from sklearn.linear_model import LinearRegression

def linear_correction(X_sim, y_sim, X_real, y_real, wavelengths,
selected_wavelengths):
    X_sim_corrected = X_sim.copy()
    for wl in selected_wavelengths:
        idx = np.where(wavelengths == wl)[0][0]
        model_real = LinearRegression().fit(y_real.reshape(-1,1), X_real[:,
idx])
        model_sim = LinearRegression().fit(y_sim.reshape(-1,1), X_sim[:,
idx])

        # 计算趋势差异
        diff = model_sim.predict(y_sim.reshape(-1,1)) -
model_real.predict(y_sim.reshape(-1,1))
        X_sim_corrected[:, idx] -= diff

    return X_sim_corrected
```

▶️ 方法 2: 多项式回归校正（非线性）

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

def polynomial_correction(X_sim, y_sim, X_real, y_real, wavelengths,
selected_wavelengths, degree=2):
    X_sim_corrected = X_sim.copy()
    for wl in selected_wavelengths:
        idx = np.where(wavelengths == wl)[0][0]

        # 多项式拟合真实趋势
        poly_real = make_pipeline(PolynomialFeatures(degree),
LinearRegression())
        poly_real.fit(y_real.reshape(-1,1), X_real[:, idx])

        # 多项式拟合仿真趋势
        poly_sim = make_pipeline(PolynomialFeatures(degree),
LinearRegression())
        poly_sim.fit(y_sim.reshape(-1,1), X_sim[:, idx])

        # 计算趋势差异
        diff = poly_sim.predict(y_sim.reshape(-1,1)) -
poly_real.predict(y_sim.reshape(-1,1))
        X_sim_corrected[:, idx] -= diff

    return X_sim_corrected
```

▶️ 方法 3: 分段线性校正

```
from sklearn.linear_model import LinearRegression
import numpy as np

def piecewise_linear_correction(X_sim, y_sim, X_real, y_real, wavelengths,
selected_wavelengths, breakpoint):
    X_sim_corrected = X_sim.copy()
    for wl in selected_wavelengths:
        idx = np.where(wavelengths == wl)[0][0]

        # 分段校正
        mask_low = y_sim <= breakpoint
        mask_high = y_sim > breakpoint

        # 低段校正
        model_real_low = LinearRegression().fit(y_real[y_real <=
breakpoint].reshape(-1,1), X_real[y_real <= breakpoint, idx])
        model_sim_low = LinearRegression().fit(y_sim[mask_low].reshape(-
1,1), X_sim[mask_low, idx])

        diff_low = model_sim_low.predict(y_sim[mask_low].reshape(-1,1)) -
model_real_low.predict(y_sim[mask_low].reshape(-1,1))
        X_sim_corrected[mask_low, idx] -= diff_low

        # 高段校正
        model_real_high = LinearRegression().fit(y_real[y_real >
breakpoint].reshape(-1,1), X_real[y_real > breakpoint, idx])
        model_sim_high = LinearRegression().fit(y_sim[mask_high].reshape(-
1,1), X_sim[mask_high, idx])

        diff_high = model_sim_high.predict(y_sim[mask_high].reshape(-1,1))
- model_real_high.predict(y_sim[mask_high].reshape(-1,1))
        X_sim_corrected[mask_high, idx] -= diff_high

    return X_sim_corrected
```

使用时需指定一个合适的 breakpoint（例如中间 CD 值，如 110）。

▶️ 方法 4: 高斯过程回归(GPR)校正（推荐小样本真实数据时使用）

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF

def gpr_correction(X_sim, y_sim, X_real, y_real, wavelengths,
selected_wavelengths):
    X_sim_corrected = X_sim.copy()
    kernel = RBF(length_scale=1.0)

    for wl in selected_wavelengths:
        idx = np.where(wavelengths == wl)[0][0]
```

```
        # 用真实数据训练 GPR
        gpr_real = GaussianProcessRegressor(kernel=kernel).fit(y_real.reshape(-1,1), X_real[:, idx])

        # 用仿真数据训练 GPR
        gpr_sim = GaussianProcessRegressor(kernel=kernel).fit(y_sim.reshape(-1,1), X_sim[:, idx])

        # 计算趋势差异
        diff = gpr_sim.predict(y_sim.reshape(-1,1)) - gpr_real.predict(y_sim.reshape(-1,1))
        X_sim_corrected[:, idx] -= diff

    return X_sim_corrected
```

🚩 如何选择合适的方法？

方法	适合情景	样本需求	稳定性
线性回归	明显线性差异	少量数据即可	高
多项式回归	明显非线性趋势	较多	中等
分段线性	趋势有明显折点	较多	中等
高斯过程	非线性复杂，真实样本少	少量真实数据即可	较高

一般建议先尝试『线性』或『多项式回归』，当发现更复杂时再考虑『分段线性』或『高斯过程』。

✅ 推荐实践步骤（示例）：

```
# 举例调用（多项式回归校正）
selected_wavelengths = [500, 600, 720]  # 举例波段
X_sim_corrected_poly = polynomial_correction(X_sim, y_sim, X_real, y_real, wavelengths, selected_wavelengths, degree=2)

# 或线性回归校正
X_sim_corrected_linear = linear_correction(X_sim, y_sim, X_real, y_real, wavelengths, selected_wavelengths)
```

📌 **下一步使用：**

• 上述修正后的数据（如 X_sim_corrected_linear）再用来进行后续迁移学习、微调等。

以上代码均可直接使用，帮助你快速进行数据校正，提高模型预测效果。

你目前的问题本质上是：

在完整光谱范围（如 550 维数据）中，不同区间可能有不同类型的域差异（噪声、偏移 Shift、尺度 Scale 变化），如何**自动化识别这些有用区间**，再分别**实施差异校正或变换**。

这需要结合**区间化分析**（Segment Analysis）与**统计/机器学习方法**实现自动区间判断，然后再进行区间级别的变换与校正。

🚩 **一、自动识别差异区间的方法：**

以下推荐一种清晰可执行的路线：

📌 **（1）光谱自动分段：**

建议将 550 维的完整光谱分成多个更小的区间，比如：

• 均匀划分：如每 10~20 个波段作为一个区间

• 也可以基于光谱形态、专家知识人为区间化（如果有）

代码示例（简单均匀划分）：

```
def split_spectrum_intervals(wavelengths, interval_size=20):
    intervals = []
```

```
    num_intervals = len(wavelengths) // interval_size
    for i in range(num_intervals):
        start = i * interval_size
        end = start + interval_size
        intervals.append((start, end))
    return intervals

intervals = split_spectrum_intervals(wavelengths, interval_size=20)
```

📌（2）区间差异类型判断：

自动判断每个区间的差异类型。常见方法是：

• 噪声区间判断：

计算真实数据和仿真数据的信噪比或标准差大小，区间内数据波动大且无明显趋势，定义为噪声区间。

• shift 或 scale 区间判断：

计算真实数据和仿真数据间的线性回归拟合优度（如决定系数 R^2）和回归系数的变化：

• 拟合度高，但截距（intercept）显著不为零 → shift

• 拟合度高，斜率明显偏离 1 → scale（尺度差异）

• 非线性区间判断：

拟合优度（R^2）明显偏低或多项式拟合明显优于线性拟合 → 非线性差异。

代码示例：

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

def classify_interval(X_sim, y_sim, X_real, y_real, start, end):
    interval_type = 'noise'
    sim_interval = X_sim[:, start:end].mean(axis=1)
    real_interval = X_real[:, start:end].mean(axis=1)

    # 线性拟合
    lr = LinearRegression().fit(sim_interval.reshape(-1,1), real_interval)
```

```
    r2 = lr.score(sim_interval.reshape(-1,1), real_interval)
    slope = lr.coef_[0]
    intercept = lr.intercept_

    if r2 < 0.2:
        interval_type = 'nonlinear'
    elif abs(intercept) > 0.1 and abs(slope - 1) < 0.2:
        interval_type = 'shift'
    elif abs(slope - 1) > 0.2:
        interval_type = 'scale'
    else:
        interval_type = 'usable_linear'

    return interval_type, r2, slope, intercept
```

• 可以调整阈值（如 R^2, intercept, slope）适合你的数据特征。

二、基于区间类型进行特征变换和校正：

针对不同区间类型，采用不同的校正策略：

• 噪声(noise)区间：舍弃（不使用该区间特征）。

• shift 区间：减去 shift（即减去线性拟合的截距）。

• scale 区间：线性缩放（scale 到真实趋势）。

• 非线性(nonlinear)：用非线性回归（如多项式）或高斯过程校正。

• usable_linear：不需额外处理，可直接使用。

示例代码实现：

```
def correct_interval(X_sim, y_sim, X_real, y_real, start, end,
interval_type):
    corrected_interval = X_sim[:, start:end].copy()
    sim_interval = X_sim[:, start:end].mean(axis=1)
    real_interval = X_real[:, start:end].mean(axis=1)

    if interval_type == 'shift':
        lr = LinearRegression().fit(sim_interval.reshape(-1,1),
real_interval)
        shift = lr.intercept_
        corrected_interval -= shift

    elif interval_type == 'scale':
        lr = LinearRegression().fit(sim_interval.reshape(-1,1),
real_interval)
```

```
        scale = lr.coef_[0]
        intercept = lr.intercept_
        corrected_interval = corrected_interval * scale + intercept

    elif interval_type == 'nonlinear':
        poly = make_pipeline(PolynomialFeatures(2), LinearRegression())
        poly.fit(sim_interval.reshape(-1,1), real_interval)
        correction = poly.predict(sim_interval.reshape(-1,1)) -
sim_interval
        corrected_interval += correction.reshape(-1,1)

    # noise 区间不作处理，因为后续可能会舍弃

    return corrected_interval
```

三、完整自动流程示例代码（综合以上两步）：

```
interval_results = []
corrected_X_sim = []

for start, end in intervals:
    interval_type, r2, slope, intercept = classify_interval(X_sim, y_sim,
X_real, y_real, start, end)
    print(f"Interval {start}-{end}: type={interval_type}, R²={r2:.2f},
slope={slope:.2f}, intercept={intercept:.2f}")
    if interval_type != 'noise':
        corrected_interval = correct_interval(X_sim, y_sim, X_real, y_real,
start, end, interval_type)
        corrected_X_sim.append(corrected_interval)
        interval_results.append((start, end, interval_type))
    else:
        print(f"Interval {start}-{end} discarded due to noise.")

# 合并所有有效区间，得到校正后的完整数据
corrected_X_sim = np.hstack(corrected_X_sim)
```

四、后续步骤：

• 使用校正后的特征（corrected_X_sim）进行迁移学习、回归建模。

• 真实数据同样以对应区间（除噪声区间外）拼接，形成匹配的训练和验证集。

五、总结和推荐做法：

• **自动区间划分 → 差异类型自动判断 → 针对性校正或舍弃**。

• 对每个小区间单独分析，比整体分析更加细致有效。

• 以上代码示例易于实现，效果较稳健。

推荐先以此为基础，在你数据上测试，具体参数（区间大小、判断阈值）根据实验调整。

你现在遇到的情况是：

• 用少量真实数据（~10 个样本）做 PLS 回归，效果不错（$R^2 \approx 0.85$）。

• 手上还有大量仿真数据（~6000 个样本），但仿真光谱与真实光谱有差异。

因此下一步明确的目标是：

如何有效将仿真数据融入当前 PLS 建模中，以进一步提升真实数据预测精度？

🚩 为什么 PLS 回归已有良好效果？

• PLS（偏最小二乘）擅长在少量样本下对高维特征进行建模，已挖掘到真实数据中的有效信息。

• 但是，样本太少，模型可能仍未挖掘完全的信息，进一步提升可能性仍存在。

🚩 如何将仿真数据有效融入 PLS 中？

这里有三种明确且易实施的策略：

📌（一）仿真数据预训练+真实数据微调（迁移学习式 PLS）

思路：

• **第一步**：用仿真数据训练 PLS 回归模型，得到初始 PLS 权重。

• **第二步**：再用真实数据进一步调整或修正 PLS 模型权重（类似迁移学习的思路）。

具体实践：

• PLS 虽然本身无『微调』机制，但可实现**组合式迁移**：

1. 用仿真数据确定 PLS 的潜变量数量和主要权重。

2. 用真实数据再做一次 PLS，但初始化时或通过权重修正时参考仿真 PLS 的结果。

代码示例（简单实现）：

```
from sklearn.cross_decomposition import PLSRegression

# Step1: 仿真数据预训练
pls_sim = PLSRegression(n_components=5)
pls_sim.fit(X_sim, y_sim)

# 提取仿真 PLS 权重
X_weights_sim = pls_sim.x_weights_

# Step2: 真实数据微调
pls_real = PLSRegression(n_components=5)
pls_real.fit(X_real, y_real)

# 微调（融合仿真权重）：用简单的加权平均权重示例
alpha = 0.7  # 控制权重融合程度
pls_real.x_weights_ = alpha * pls_real.x_weights_ + (1 - alpha) *
X_weights_sim
```

• 实践中 alpha 可调，类似迁移学习的微调因子。

• 最后用调整后的 pls_real 进行预测。

📌（二）先用仿真数据校正真实数据的光谱，再用 PLS 建模（数据融合式）

思路：

• 根据前文『区间校正』，先**校正仿真数据到真实数据域**。

• 然后用**真实数据+校正后的仿真数据**共同训练 PLS 模型，增加有效数据量。

步骤明确：

• 执行前面讨论的区间校正或趋势校正（线性/多项式）。

• 校正后数据：X_sim_corrected、y_sim。

• 合并真实数据和校正仿真数据，训练 PLS：

```
X_combined = np.vstack([X_real, X_sim_corrected])
y_combined = np.concatenate([y_real, y_sim])

pls_combined = PLSRegression(n_components=5)
pls_combined.fit(X_combined, y_combined)
```

这样做优点明显：

• 利用大量校正仿真数据，显著降低过拟合风险。

• 提升 PLS 稳定性和泛化能力。

📌（三）Stacking 融合 PLS 模型（更稳健方案）

思路：

• 用仿真数据和真实数据分别训练两个独立 PLS 模型。

• 再用真实数据训练一个简单的 Stacking 模型融合两个 PLS 模型预测结果。

步骤：

1. 用仿真数据训练 PLS (pls_sim)，真实数据训练 PLS (pls_real)。

2. 再用真实数据训练融合模型（例如线性回归）：

```
from sklearn.linear_model import LinearRegression
```

```
# 分别训练仿真和真实 PLS
pls_sim = PLSRegression(n_components=5).fit(X_sim, y_sim)
pls_real = PLSRegression(n_components=5).fit(X_real, y_real)

# 用真实数据训练融合模型
pred_real_from_sim = pls_sim.predict(X_real)
pred_real_from_real = pls_real.predict(X_real)

stacker = LinearRegression().fit(
    np.hstack([pred_real_from_sim, pred_real_from_real]),
    y_real
)

# 最终预测新数据:
def predict_stacking(X_new):
    pred1 = pls_sim.predict(X_new)
    pred2 = pls_real.predict(X_new)
    final_pred = stacker.predict(np.hstack([pred1, pred2]))
    return final_pred
```

• 这种方法稳健可靠，融合不同域特征。

推荐实践流程（最佳方案）：

推荐第二种（数据校正融合后 PLS 建模）方案：

• 原因：

• 直观有效，充分发挥 PLS 模型特性。

• 已有前面的特征校正代码可复用，实施简便。

• 对少量真实数据提升效果明显。

具体推荐实践步骤：

1. **校正仿真数据：**

```
X_sim_corrected = polynomial_correction(X_sim, y_sim, X_real, y_real,
wavelengths, selected_wavelengths, degree=2)
```

2. **合并训练 PLS 模型：**

```
from sklearn.cross_decomposition import PLSRegression

X_combined = np.vstack([X_real, X_sim_corrected])
y_combined = np.concatenate([y_real, y_sim])

pls_final = PLSRegression(n_components=5)
pls_final.fit(X_combined, y_combined)
```

3. **评估模型效果：**

```
y_pred = pls_final.predict(X_real)
from sklearn.metrics import r2_score
r2 = r2_score(y_real, y_pred)
print("融合仿真数据后的真实数据 R²: ", r2)
```

🚩 **进一步提高精度的技巧：**

• 微调 n_components 潜变量数量。

• 仿真数据校正方式调整（非线性→分段校正等）。

• 增加数据增强（例如对真实数据的微小随机扰动）。

✅ **结论与推荐：**

• **PLS + 仿真数据融合（校正仿真数据后加入真实数据建模）** 是你当前场景下最有效的方式。

• 明确且易实现，预计可以进一步显著提高目前（0.85）的性能。

你可优先按上述第二种方法进行实践，进一步优化参数，有问题随时再讨论。

光谱数据的标准化方法选择至关重要，会显著影响模型表现和后续分析结果。以下推荐几种最适合光谱数据的标准化方法，并明确说明适用场景：

🚩 **光谱数据常用标准化方法（推荐顺序）：**

方法	原理	适用场景	推荐指数
矢量归一化（Vector Normalization）	每根光谱除以自身的欧氏长度，使每个光谱向量模为 1	**关注光谱形状而非绝对强度时**（如余弦相似度、PLS 模型）	⭐⭐⭐⭐⭐
标准正态变换（Standard Normal Variate, SNV）	每根光谱减去自身均值再除以自身标准差	去除样本间的整体偏移、尺度差异（适合 NIR、中红外光谱）	⭐⭐⭐⭐⭐
Min-Max 标准化	线性缩放到[0,1]区间	强调波段之间相对幅度变化时	⭐⭐⭐
均值中心化（Mean Centering）	每个波段减去所有样本的均值	PLS 回归的基本前处理步骤	⭐⭐⭐⭐⭐

📌 **1. 最推荐：矢量归一化（Vector Normalization）**

优点：

• **消除光谱强度的尺度影响**，聚焦光谱形态。

• 非常适合基于光谱形状的相似性评估（余弦相似度）。

实现代码：

```
import numpy as np

def vector_normalize(X):
    norms = np.linalg.norm(X, axis=1, keepdims=True)
    X_norm = X / norms
    return X_norm

X_norm = vector_normalize(X)
```

适用场景：

• 光谱形状、趋势比较（余弦相似度、相关性计算）。

• 基于形态的回归模型（PLS、CNN）。

📌 2. SNV（Standard Normal Variate）变换

SNV 非常适合光谱数据，是一种样本内标准化方法：

优点：

• **消除散射效应、样品厚度变化、仪器漂移等造成的偏移和尺度差异**。

• 在红外光谱（NIR、MIR）数据分析中表现出色。

实现代码：

```
def SNV(X):
    mean = np.mean(X, axis=1, keepdims=True)
    std = np.std(X, axis=1, keepdims=True)
    X_snv = (X - mean) / std
    return X_snv

X_snv = SNV(X)
```

适用场景：

• 特别推荐红外、近红外、拉曼光谱数据。

• 提高模型的鲁棒性，减小样本间整体差异。

📌 3. 均值中心化（Mean Centering）

几乎所有 PLS 回归任务中都会用到：

优点：

• **突出各波段的变化趋势**，去除波段整体偏移。

• 非常适合 PLS 回归的前处理。

实现代码：

```
def mean_centering(X):
    mean = np.mean(X, axis=0)
    X_mc = X - mean
    return X_mc

X_mc = mean_centering(X)
```

适用场景：

• PLS、PCA 前的常规处理。

📌 4. Min-Max 标准化

在波段特征的尺度变化较大时有帮助：

优点：

• 把所有波段压缩到统一尺度（0~1），直观简单。

缺点：

• 易受极值影响，不太适合有明显异常值的光谱数据。

实现代码：

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

适用场景：

• 部分机器学习方法（神经网络）对输入尺度敏感时使用。

• 在 PLS 中并非最佳选择。

🚩 推荐使用的完整组合方案（综合考虑）：

针对你的情况（PLS 建模 + 相似性判断），最佳组合：

• SNV + 均值中心化（推荐）

或

• 矢量归一化 + 均值中心化（次优选择）

代码示例（SNV + 均值中心化）：

```
# 先 SNV
X_snv = SNV(X)

# 再均值中心化
X_processed = mean_centering(X_snv)
```

🚩 各方法适用场景快速决策表：

场景	推荐标准化
强调光谱整体形状和趋势	矢量归一化
去除仪器漂移、散射效应	SNV
PLS/PCA 模型（一般）	均值中心化
波段尺度差异大	Min-Max

✅ 明确推荐：

• 如果你的光谱为**红外或近红外光谱**，优先考虑 **SNV**。

• 如果主要关注**光谱整体趋势**，特别是计算相似性，推荐使用 **矢量归一化**。

• 后续再对数据进行**均值中心化**是常规步骤。

示例：

```
# 最佳实践（推荐你当前场景使用）
X_final = mean_centering(SNV(X))
```

最终结论（明确给你当前情景的最佳推荐）：

你目前的情景（PLS 建模 + 相似性评估）：

最佳实践：

```
# SNV 标准化后均值中心化
X_processed = mean_centering(SNV(X))
```

标准化完成后的数据再用于：

• PLS 模型建模；

• 马氏距离或余弦相似度计算相似性；

• 有效提升模型性能与预测可信性评价。

以上建议清晰明确，强烈建议采用。