# -\*- coding: utf-8 -\*-

# A 题：园区微电网风光储协调优化配置

print(

"""

##### 问题1：各园区独立运营储能配置方案及其经济性分析 #####

"""

)

import pandas as pd

# 读取附件1：各园区典型日负荷数据

load\_data = pd.read\_excel("./mnt/data/附件1：各园区典型日负荷数据.xlsx")

print("各园区典型日负荷数据：")

print(load\_data.head())

# 读取附件2：各园区典型日风光发电数据

generation\_data = pd.read\_excel("./mnt/data/附件2：各园区典型日风光发电数据.xlsx")

print("\n各园区典型日风光发电数据：")

print(generation\_data)

print(

"""

###############################

# 计算各园区的购电量和弃风弃光电量

###############################

"""

)

import numpy as np

# 参数设置

P\_pv\_A = 750 # 园区A光伏装机容量 (kW)

P\_w\_B = 1000 # 园区B风电装机容量 (kW)

P\_pv\_C = 600 # 园区C光伏装机容量 (kW)

P\_w\_C = 500 # 园区C风电装机容量 (kW)

# 读取负荷数据

load\_data["时间（h）"] = pd.to\_datetime(load\_data["时间（h）"], format="%H:%M:%S")

load\_data.set\_index("时间（h）", inplace=True)

# 读取发电数据

generation\_data["时间（h）"] = pd.to\_datetime(

generation\_data["时间（h）"], format="%H:%M:%S"

)

generation\_data.set\_index("时间（h）", inplace=True)

# 计算每个园区的光伏和风电实际出力 (kW)

generation\_data["园区A 光伏出力(kW)"] = (

generation\_data["园区A 光伏出力（p.u.）"] \* P\_pv\_A

)

generation\_data["园区B风电出力(kW)"] = generation\_data["园区B风电出力（p.u.）"] \* P\_w\_B

generation\_data["园区C 光伏出力(kW)"] = (

generation\_data["园区C 光伏出力（p.u.）"] \* P\_pv\_C

)

generation\_data["园区C 风电出力(kW)"] = (

generation\_data["园区C 风电出力（p.u.）"] \* P\_w\_C

)

# 计算每个园区的总出力

generation\_data["园区A 总出力(kW)"] = generation\_data["园区A 光伏出力(kW)"]

generation\_data["园区B 总出力(kW)"] = generation\_data["园区B风电出力(kW)"]

generation\_data["园区C 总出力(kW)"] = (

generation\_data["园区C 光伏出力(kW)"] + generation\_data["园区C 风电出力(kW)"]

)

# 合并负荷和发电数据

data = load\_data.join(

generation\_data[["园区A 总出力(kW)", "园区B 总出力(kW)", "园区C 总出力(kW)"]]

)

# 计算购电量和弃风弃光电量

data["园区A 购电量(kW)"] = np.maximum(

data["园区A负荷(kW)"] - data["园区A 总出力(kW)"], 0

)

data["园区B 购电量(kW)"] = np.maximum(

data["园区B负荷(kW)"] - data["园区B 总出力(kW)"], 0

)

data["园区C 购电量(kW)"] = np.maximum(

data["园区C负荷(kW)"] - data["园区C 总出力(kW)"], 0

)

data["园区A 弃光电量(kW)"] = np.maximum(

data["园区A 总出力(kW)"] - data["园区A负荷(kW)"], 0

)

data["园区B 弃风电量(kW)"] = np.maximum(

data["园区B 总出力(kW)"] - data["园区B负荷(kW)"], 0

)

data["园区C 弃光弃风电量(kW)"] = np.maximum(

data["园区C 总出力(kW)"] - data["园区C负荷(kW)"], 0

)

# 计算总购电量和弃风弃光电量

total\_purchase\_A = data["园区A 购电量(kW)"].sum()

total\_purchase\_B = data["园区B 购电量(kW)"].sum()

total\_purchase\_C = data["园区C 购电量(kW)"].sum()

total\_waste\_A = data["园区A 弃光电量(kW)"].sum()

total\_waste\_B = data["园区B 弃风电量(kW)"].sum()

total\_waste\_C = data["园区C 弃光弃风电量(kW)"].sum()

# 计算总供电成本和单位电量平均供电成本

purchase\_cost\_A = total\_purchase\_A \* 1 # 园区A的购电成本

purchase\_cost\_B = total\_purchase\_B \* 1 # 园区B的购电成本

purchase\_cost\_C = total\_purchase\_C \* 1 # 园区C的购电成本

average\_cost\_A = purchase\_cost\_A / data["园区A负荷(kW)"].sum()

average\_cost\_B = purchase\_cost\_B / data["园区B负荷(kW)"].sum()

average\_cost\_C = purchase\_cost\_C / data["园区C负荷(kW)"].sum()

# 输出结果

result = {

"园区A": {

"总购电量(kWh)": total\_purchase\_A,

"弃光电量(kWh)": total\_waste\_A,

"总供电成本(元)": purchase\_cost\_A,

"单位电量平均供电成本(元/kWh)": average\_cost\_A,

},

"园区B": {

"总购电量(kWh)": total\_purchase\_B,

"弃风电量(kWh)": total\_waste\_B,

"总供电成本(元)": purchase\_cost\_B,

"单位电量平均供电成本(元/kWh)": average\_cost\_B,

},

"园区C": {

"总购电量(kWh)": total\_purchase\_C,

"弃光弃风电量(kWh)": total\_waste\_C,

"总供电成本(元)": purchase\_cost\_C,

"单位电量平均供电成本(元/kWh)": average\_cost\_C,

},

}

for key, value in result.items():

print(f"\n{key} 结果：")

for sub\_key, sub\_value in value.items():

print(f"{sub\_key}: {sub\_value:.2f}")

print(

"""

############################

# 配置50kW/100kWh储能后的分析

############################

"""

)

# 储能参数设置

storage\_power = 50 # 储能功率 (kW)

storage\_capacity = 100 # 储能容量 (kWh)

charge\_efficiency = 0.95

discharge\_efficiency = 0.95

SOC\_min = 0.10

SOC\_max = 0.90

SOC\_initial = 0.50 \* storage\_capacity

# 初始化储能SOC

data["SOC\_A"] = SOC\_initial

data["SOC\_B"] = SOC\_initial

data["SOC\_C"] = SOC\_initial

# 储能充放电策略

for index, row in data.iterrows():

# 园区A储能策略

if row["园区A负荷(kW)"] > row["园区A 总出力(kW)"]: # 负荷大于发电，放电

discharge = min(

(row["园区A负荷(kW)"] - row["园区A 总出力(kW)"]) / discharge\_efficiency,

storage\_power,

)

actual\_discharge = min(discharge, (row["SOC\_A"] - storage\_capacity \* SOC\_min))

data.at[index, "SOC\_A"] -= actual\_discharge

data.at[index, "园区A 购电量(kW)"] = max(

row["园区A负荷(kW)"]

- row["园区A 总出力(kW)"]

- actual\_discharge \* discharge\_efficiency,

0,

)

else: # 负荷小于发电，充电

charge = min(

(row["园区A 总出力(kW)"] - row["园区A负荷(kW)"]) \* charge\_efficiency,

storage\_power,

)

actual\_charge = min(charge, (storage\_capacity \* SOC\_max - row["SOC\_A"]))

data.at[index, "SOC\_A"] += actual\_charge

data.at[index, "园区A 弃光电量(kW)"] = max(

row["园区A 总出力(kW)"]

- row["园区A负荷(kW)"]

- actual\_charge / charge\_efficiency,

0,

)

# 园区B储能策略

if row["园区B负荷(kW)"] > row["园区B 总出力(kW)"]: # 负荷大于发电，放电

discharge = min(

(row["园区B负荷(kW)"] - row["园区B 总出力(kW)"]) / discharge\_efficiency,

storage\_power,

)

actual\_discharge = min(discharge, (row["SOC\_B"] - storage\_capacity \* SOC\_min))

data.at[index, "SOC\_B"] -= actual\_discharge

data.at[index, "园区B 购电量(kW)"] = max(

row["园区B负荷(kW)"]

- row["园区B 总出力(kW)"]

- actual\_discharge \* discharge\_efficiency,

0,

)

else: # 负荷小于发电，充电

charge = min(

(row["园区B 总出力(kW)"] - row["园区B负荷(kW)"]) \* charge\_efficiency,

storage\_power,

)

actual\_charge = min(charge, (storage\_capacity \* SOC\_max - row["SOC\_B"]))

data.at[index, "SOC\_B"] += actual\_charge

data.at[index, "园区B 弃风电量(kW)"] = max(

row["园区B 总出力(kW)"]

- row["园区B负荷(kW)"]

- actual\_charge / charge\_efficiency,

0,

)

# 园区C储能策略

if row["园区C负荷(kW)"] > row["园区C 总出力(kW)"]: # 负荷大于发电，放电

discharge = min(

(row["园区C负荷(kW)"] - row["园区C 总出力(kW)"]) / discharge\_efficiency,

storage\_power,

)

actual\_discharge = min(discharge, (row["SOC\_C"] - storage\_capacity \* SOC\_min))

data.at[index, "SOC\_C"] -= actual\_discharge

data.at[index, "园区C 购电量(kW)"] = max(

row["园区C负荷(kW)"]

- row["园区C 总出力(kW)"]

- actual\_discharge \* discharge\_efficiency,

0,

)

else: # 负荷小于发电，充电

charge = min(

(row["园区C 总出力(kW)"] - row["园区C负荷(kW)"]) \* charge\_efficiency,

storage\_power,

)

actual\_charge = min(charge, (storage\_capacity \* SOC\_max - row["SOC\_C"]))

data.at[index, "SOC\_C"] += actual\_charge

data.at[index, "园区C 弃光弃风电量(kW)"] = max(

row["园区C 总出力(kW)"]

- row["园区C负荷(kW)"]

- actual\_charge / charge\_efficiency,

0,

)

# 计算配置储能后的购电量和弃电量

total\_purchase\_A\_storage = data["园区A 购电量(kW)"].sum()

total\_purchase\_B\_storage = data["园区B 购电量(kW)"].sum()

total\_purchase\_C\_storage = data["园区C 购电量(kW)"].sum()

total\_waste\_A\_storage = data["园区A 弃光电量(kW)"].sum()

total\_waste\_B\_storage = data["园区B 弃风电量(kW)"].sum()

total\_waste\_C\_storage = data["园区C 弃光弃风电量(kW)"].sum()

# 计算配置储能后的总供电成本和单位电量平均供电成本

purchase\_cost\_A\_storage = total\_purchase\_A\_storage \* 1 # 园区A的购电成本

purchase\_cost\_B\_storage = total\_purchase\_B\_storage \* 1 # 园区B的购电成本

purchase\_cost\_C\_storage = total\_purchase\_C\_storage \* 1 # 园区C的购电成本

average\_cost\_A\_storage = purchase\_cost\_A\_storage / data["园区A负荷(kW)"].sum()

average\_cost\_B\_storage = purchase\_cost\_B\_storage / data["园区B负荷(kW)"].sum()

average\_cost\_C\_storage = purchase\_cost\_C\_storage / data["园区C负荷(kW)"].sum()

# 输出结果

result\_storage = {

"园区A": {

"总购电量(kWh)": total\_purchase\_A\_storage,

"弃光电量(kWh)": total\_waste\_A\_storage,

"总供电成本(元)": purchase\_cost\_A\_storage,

"单位电量平均供电成本(元/kWh)": average\_cost\_A\_storage,

},

"园区B": {

"总购电量(kWh)": total\_purchase\_B\_storage,

"弃风电量(kWh)": total\_waste\_B\_storage,

"总供电成本(元)": purchase\_cost\_B\_storage,

"单位电量平均供电成本(元/kWh)": average\_cost\_B\_storage,

},

"园区C": {

"总购电量(kWh)": total\_purchase\_C\_storage,

"弃光弃风电量(kWh)": total\_waste\_C\_storage,

"总供电成本(元)": purchase\_cost\_C\_storage,

"单位电量平均供电成本(元/kWh)": average\_cost\_C\_storage,

},

}

for key, value in result\_storage.items():

print(f"\n{key} 结果（配置储能后）：")

for sub\_key, sub\_value in value.items():

print(f"{sub\_key}: {sub\_value:.2f}")

print()

print(

"""

#######################################

# 判断50kW/100kWh的方案是否最优

# 并制定各园区最优的储能功率和容量配置方案

#######################################

"""

)

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import GridSearchCV

# 准备数据

# 生成一些特征和目标数据，用于机器学习模型的训练

# 特征包括：负荷、发电、储能参数等

# 目标包括：总供电成本、弃电量等

# 生成特征数据

features = data[

[

"园区A负荷(kW)",

"园区A 总出力(kW)",

"园区B负荷(kW)",

"园区B 总出力(kW)",

"园区C负荷(kW)",

"园区C 总出力(kW)",

]

]

# 生成目标数据

# 这里我们假设目标数据是购电量、弃电量和总供电成本的加权和

data["总供电成本"] = (

data["园区A 购电量(kW)"] \* 1

+ data["园区B 购电量(kW)"] \* 1

+ data["园区C 购电量(kW)"] \* 1

)

data["弃电量"] = (

data["园区A 弃光电量(kW)"]

+ data["园区B 弃风电量(kW)"]

+ data["园区C 弃光弃风电量(kW)"]

)

data["目标"] = data["总供电成本"] + data["弃电量"] \* 0.1

# 拆分训练集和测试集

X = features.values

y = data["目标"].values

# 创建随机森林回归模型

rf = RandomForestRegressor()

# 定义参数网格用于网格搜索

param\_grid = {

"n\_estimators": [100, 200, 300],

"max\_depth": [10, 20, 30],

"min\_samples\_split": [2, 5, 10],

}

# 网格搜索

grid\_search = GridSearchCV(

estimator=rf,

param\_grid=param\_grid,

cv=5,

scoring="neg\_mean\_squared\_error",

n\_jobs=-1,

)

grid\_search.fit(X, y)

# 最优参数

best\_params = grid\_search.best\_params\_

best\_rf = grid\_search.best\_estimator\_

# 输出最优参数

print("最优参数：", best\_params)

print()

# 使用最优模型进行预测

y\_pred = best\_rf.predict(X)

# 计算最优储能配置方案

optimal\_storage\_power\_A = np.mean(y\_pred[: len(data["园区A负荷(kW)"])]) \* storage\_power

optimal\_storage\_capacity\_A = (

np.mean(y\_pred[: len(data["园区A负荷(kW)"])]) \* storage\_capacity

)

optimal\_storage\_power\_B = np.mean(y\_pred[: len(data["园区B负荷(kW)"])]) \* storage\_power

optimal\_storage\_capacity\_B = (

np.mean(y\_pred[: len(data["园区B负荷(kW)"])]) \* storage\_capacity

)

optimal\_storage\_power\_C = np.mean(y\_pred[: len(data["园区C负荷(kW)"])]) \* storage\_power

optimal\_storage\_capacity\_C = (

np.mean(y\_pred[: len(data["园区C负荷(kW)"])]) \* storage\_capacity

)

# 输出结果

print(f"最优储能配置方案：")

print(f"园区A储能功率: {optimal\_storage\_power\_A:.2f} kW")

print(f"园区A储能容量: {optimal\_storage\_capacity\_A:.2f} kWh")

print(f"园区B储能功率: {optimal\_storage\_power\_B:.2f} kW")

print(f"园区B储能容量: {optimal\_storage\_capacity\_B:.2f} kWh")

print(f"园区C储能功率: {optimal\_storage\_power\_C:.2f} kW")

print(f"园区C储能容量: {optimal\_storage\_capacity\_C:.2f} kWh")

print()

print(

"""

##### 问题2：联合园区储能配置方案及其经济性分析 #####

"""

)

print(

"""

#####################################################################

# 若未配置储能，分析联合园区运行经济性

# 包括：联合园区的总购电量、总弃风弃光电量、总供电成本和单位电量平均供电成本

#####################################################################

"""

)

import pandas as pd

# 读取负荷数据

load\_data = pd.read\_excel("./mnt/data/附件1：各园区典型日负荷数据.xlsx")

generation\_data = pd.read\_excel("./mnt/data/附件2：各园区典型日风光发电数据.xlsx")

# 将时间转换为datetime类型

load\_data["时间（h）"] = pd.to\_datetime(load\_data["时间（h）"], format="%H:%M:%S")

generation\_data["时间（h）"] = pd.to\_datetime(

generation\_data["时间（h）"], format="%H:%M:%S"

)

# 设置时间为索引

load\_data.set\_index("时间（h）", inplace=True)

generation\_data.set\_index("时间（h）", inplace=True)

# 计算各园区的总负荷和总发电

load\_data["总负荷(kW)"] = (

load\_data["园区A负荷(kW)"] + load\_data["园区B负荷(kW)"] + load\_data["园区C负荷(kW)"]

)

generation\_data["总光伏出力(kW)"] = (

generation\_data["园区A 光伏出力（p.u.）"] \* 750

+ generation\_data["园区C 光伏出力（p.u.）"] \* 600

)

generation\_data["总风电出力(kW)"] = (

generation\_data["园区B风电出力（p.u.）"] \* 1000

+ generation\_data["园区C 风电出力（p.u.）"] \* 500

)

generation\_data["总发电(kW)"] = (

generation\_data["总光伏出力(kW)"] + generation\_data["总风电出力(kW)"]

)

# 合并负荷和发电数据

combined\_data = pd.concat([load\_data, generation\_data], axis=1)

# 计算购电量和弃电量

combined\_data["购电量(kW)"] = np.maximum(

combined\_data["总负荷(kW)"] - combined\_data["总发电(kW)"], 0

)

combined\_data["弃电量(kW)"] = np.maximum(

combined\_data["总发电(kW)"] - combined\_data["总负荷(kW)"], 0

)

# 计算总购电量、总弃电量、总供电成本和单位电量平均供电成本

total\_purchase = combined\_data["购电量(kW)"].sum()

total\_waste = combined\_data["弃电量(kW)"].sum()

total\_cost = total\_purchase \* 1 # 购电成本为1元/kWh

average\_cost = total\_cost / combined\_data["总负荷(kW)"].sum()

# 输出结果

print(f"联合园区未配置储能时的经济性分析：")

print(f"总购电量(kWh): {total\_purchase:.2f}")

print(f"总弃电量(kWh): {total\_waste:.2f}")

print(f"总供电成本(元): {total\_cost:.2f}")

print(f"单位电量平均供电成本(元/kWh): {average\_cost:.2f}")

print()

print(

"""

#####################################

# 假设风光荷功率波动特性保持上述条件不变

# 制定联合园区的总储能最优配置方案

#####################################

"""

)

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split, GridSearchCV

# 加载数据

load\_data = pd.read\_excel('./mnt/data/附件1：各园区典型日负荷数据.xlsx')

generation\_data = pd.read\_excel('./mnt/data/附件2：各园区典型日风光发电数据.xlsx')

# 处理数据

load\_data['时间（h）'] = pd.to\_datetime(load\_data['时间（h）'], format='%H:%M:%S')

generation\_data['时间（h）'] = pd.to\_datetime(generation\_data['时间（h）'], format='%H:%M:%S')

load\_data.set\_index('时间（h）', inplace=True)

generation\_data.set\_index('时间（h）', inplace=True)

# 合并负荷数据

load\_data['总负荷(kW)'] = load\_data['园区A负荷(kW)'] + load\_data['园区B负荷(kW)'] + load\_data['园区C负荷(kW)']

# 合并发电数据

generation\_data['总光伏出力(kW)'] = generation\_data['园区A 光伏出力（p.u.）'] \* 750 + generation\_data['园区C 光伏出力（p.u.）'] \* 600

generation\_data['总风电出力(kW)'] = generation\_data['园区B风电出力（p.u.）'] \* 1000 + generation\_data['园区C 风电出力（p.u.）'] \* 500

generation\_data['总发电(kW)'] = generation\_data['总光伏出力(kW)'] + generation\_data['总风电出力(kW)']

# 合并数据

combined\_data = pd.concat([load\_data, generation\_data], axis=1)

# 计算购电量和弃电量

combined\_data['购电量(kW)'] = np.maximum(combined\_data['总负荷(kW)'] - combined\_data['总发电(kW)'], 0)

combined\_data['弃电量(kW)'] = np.maximum(combined\_data['总发电(kW)'] - combined\_data['总负荷(kW)'], 0)

# 储能参数设置

charge\_efficiency = 0.95

discharge\_efficiency = 0.95

SOC\_min = 0.10

SOC\_max = 0.90

# 生成模拟数据

def simulate\_storage\_operation(storage\_power, storage\_capacity):

combined\_data['SOC'] = 0.50 \* storage\_capacity # 初始SOC为50%的storage\_capacity

combined\_data['购电量\_储能后(kW)'] = combined\_data['购电量(kW)']

combined\_data['弃电量\_储能后(kW)'] = combined\_data['弃电量(kW)']

for index, row in combined\_data.iterrows():

# 如果负荷大于发电，放电

if row['总负荷(kW)'] > row['总发电(kW)']:

discharge = min((row['总负荷(kW)'] - row['总发电(kW)']) / discharge\_efficiency, storage\_power)

actual\_discharge = min(discharge, (row['SOC'] - storage\_capacity \* SOC\_min))

combined\_data.at[index, 'SOC'] -= actual\_discharge

combined\_data.at[index, '购电量\_储能后(kW)'] = max(row['总负荷(kW)'] - row['总发电(kW)'] - actual\_discharge \* discharge\_efficiency, 0)

# 如果负荷小于发电，充电

else:

charge = min((row['总发电(kW)'] - row['总负荷(kW)']) \* charge\_efficiency, storage\_power)

actual\_charge = min(charge, (storage\_capacity \* SOC\_max - row['SOC']))

combined\_data.at[index, 'SOC'] += actual\_charge

combined\_data.at[index, '弃电量\_储能后(kW)'] = max(row['总发电(kW)'] - row['总负荷(kW)'] - actual\_charge / charge\_efficiency, 0)

total\_purchase\_storage = combined\_data['购电量\_储能后(kW)'].sum()

total\_waste\_storage = combined\_data['弃电量\_储能后(kW)'].sum()

total\_cost\_storage = total\_purchase\_storage \* 1 # 购电成本为1元/kWh

average\_cost\_storage = total\_cost\_storage / combined\_data['总负荷(kW)'].sum()

return total\_purchase\_storage, total\_waste\_storage, total\_cost\_storage, average\_cost\_storage

# 生成特征数据和目标数据

storage\_powers = np.linspace(0, 200, 20) # 储能功率从0到200kW

storage\_capacities = np.linspace(0, 500, 20) # 储能容量从0到500kWh

X = []

y = []

for storage\_power in storage\_powers:

for storage\_capacity in storage\_capacities:

total\_purchase\_storage, total\_waste\_storage, total\_cost\_storage, average\_cost\_storage = simulate\_storage\_operation(storage\_power, storage\_capacity)

X.append([storage\_power, storage\_capacity])

y.append(total\_cost\_storage)

X = np.array(X)

y = np.array(y)

# 拆分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 创建随机森林回归模型

rf = RandomForestRegressor()

# 定义参数网格用于网格搜索

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [10, 20, 30],

'min\_samples\_split': [2, 5, 10]

}

# 网格搜索

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5, scoring='neg\_mean\_squared\_error', n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# 最优参数

best\_params = grid\_search.best\_params\_

best\_rf = grid\_search.best\_estimator\_

# 使用最优模型进行预测

y\_pred = best\_rf.predict(X\_test)

# 寻找最优储能配置

min\_cost\_index = np.argmin(y\_pred)

optimal\_storage\_power = X\_test[min\_cost\_index, 0]

optimal\_storage\_capacity = X\_test[min\_cost\_index, 1]

# 输出结果

print("最优参数：", best\_params)

print(f"最优储能配置方案：")

print(f"储能功率: {optimal\_storage\_power:.2f} kW")

print(f"储能容量: {optimal\_storage\_capacity:.2f} kWh")

# # 线性规划

# from scipy.optimize import linprog

# # 储能参数设置

# storage\_cost\_power = 800 # 储能功率单价 (元/kW)

# storage\_cost\_capacity = 1800 # 储能容量单价 (元/kWh)

# years = 10 # 运行寿命 (年)

# # 目标函数系数 (总成本)

# c = [

# storage\_cost\_power \* years, # 联合园区储能功率成本

# storage\_cost\_capacity \* years, # 联合园区储能容量成本

# ]

# # 约束条件 (购电量减少)

# # 计算配置储能后的购电量和弃电量

# # 我们假设储能效率为95%，SOC范围为10%-90%

# charge\_efficiency = 0.95

# discharge\_efficiency = 0.95

# SOC\_min = 0.10

# SOC\_max = 0.90

# # 初始化数据

# combined\_data["SOC"] = 0.50 \* 100 # 假设初始SOC为50%的100kWh

# combined\_data["购电量\_储能后(kW)"] = combined\_data["购电量(kW)"]

# combined\_data["弃电量\_储能后(kW)"] = combined\_data["弃电量(kW)"]

# # 储能充放电策略模拟

# for index, row in combined\_data.iterrows():

# # 如果负荷大于发电，放电

# if row["总负荷(kW)"] > row["总发电(kW)"]:

# discharge = min(

# (row["总负荷(kW)"] - row["总发电(kW)"]) / discharge\_efficiency, 50

# )

# actual\_discharge = min(discharge, (row["SOC"] - 100 \* SOC\_min))

# combined\_data.at[index, "SOC"] -= actual\_discharge

# combined\_data.at[index, "购电量\_储能后(kW)"] = max(

# row["总负荷(kW)"]

# - row["总发电(kW)"]

# - actual\_discharge \* discharge\_efficiency,

# 0,

# )

# # 如果负荷小于发电，充电

# else:

# charge = min((row["总发电(kW)"] - row["总负荷(kW)"]) \* charge\_efficiency, 50)

# actual\_charge = min(charge, (100 \* SOC\_max - row["SOC"]))

# combined\_data.at[index, "SOC"] += actual\_charge

# combined\_data.at[index, "弃电量\_储能后(kW)"] = max(

# row["总发电(kW)"] - row["总负荷(kW)"] - actual\_charge / charge\_efficiency, 0

# )

# # 计算配置储能后的购电量和弃电量

# total\_purchase\_storage = combined\_data["购电量\_储能后(kW)"].sum()

# total\_waste\_storage = combined\_data["弃电量\_储能后(kW)"].sum()

# total\_cost\_storage = total\_purchase\_storage \* 1 # 购电成本为1元/kWh

# average\_cost\_storage = total\_cost\_storage / combined\_data["总负荷(kW)"].sum()

# # 输出结果

# print(f"联合园区配置储能后的经济性分析：")

# print(f"总购电量(kWh): {total\_purchase\_storage:.2f}")

# print(f"总弃电量(kWh): {total\_waste\_storage:.2f}")

# print(f"总供电成本(元): {total\_cost\_storage:.2f}")

# print(f"单位电量平均供电成本(元/kWh): {average\_cost\_storage:.2f}")

# print()

print(

"""

######################

# 与各园区独立运营相比

# 联合运营的经济收益分析

######################

"""

)

# 各园区独立运营总量

total\_purchase\_independent = 4874.12 + 2432.30 + 2699.39

total\_waste\_independent = 951.20 + 897.50 + 1128.02

total\_cost\_independent = 4874.12 + 2432.30 + 2699.39

average\_cost\_independent = total\_cost\_independent / (

data["园区A负荷(kW)"].sum()

+ data["园区B负荷(kW)"].sum()

+ data["园区C负荷(kW)"].sum()

)

# 联合运营总量

total\_purchase\_joint = total\_purchase\_storage

total\_waste\_joint = total\_waste\_storage

total\_cost\_joint = total\_cost\_storage

average\_cost\_joint = average\_cost\_storage

# 计算经济收益变化

savings\_purchase = total\_purchase\_independent - total\_purchase\_joint

savings\_waste = total\_waste\_independent - total\_waste\_joint

savings\_cost = total\_cost\_independent - total\_cost\_joint

savings\_average\_cost = average\_cost\_independent - average\_cost\_joint

# 输出结果

print(f"各园区独立运营 vs 联合运营的经济性对比：")

print(f"总购电量节省(kWh): {savings\_purchase:.2f}")

print(f"总弃电量减少(kWh): {savings\_waste:.2f}")

print(f"总供电成本节省(元): {savings\_cost:.2f}")

print(f"单位电量平均供电成本减少(元/kWh): {savings\_average\_cost:.2f}")

print()

########## 可视化##########

import matplotlib.pyplot as plt

# Comparison of Purchased and Wasted Energy Before and After Energy Storage Implementation

parks = ['Park A', 'Park B', 'Park C']

before\_purchase = [total\_purchase\_A, total\_purchase\_B, total\_purchase\_C]

after\_purchase = [total\_purchase\_A\_storage, total\_purchase\_B\_storage, total\_purchase\_C\_storage]

before\_waste = [total\_waste\_A, total\_waste\_B, total\_waste\_C]

after\_waste = [total\_waste\_A\_storage, total\_waste\_B\_storage, total\_waste\_C\_storage]

plt.figure(figsize=(10, 6))

bar\_width = 0.35

index = range(len(parks))

plt.bar(index, before\_purchase, bar\_width, label='Before Energy Storage', color='b')

plt.bar(index, before\_waste, bar\_width, label='Before Energy Storage', color='r', bottom=before\_purchase)

plt.bar([i + bar\_width for i in index], after\_purchase, bar\_width, label='After Energy Storage', color='g')

plt.bar([i + bar\_width for i in index], after\_waste, bar\_width, label='After Energy Storage', color='y', bottom=after\_purchase)

plt.xlabel('Parks')

plt.ylabel('Energy (kWh)')

plt.title('Comparison of Purchased and Wasted Energy Before and After Energy Storage Implementation')

plt.xticks([i + bar\_width / 2 for i in index], parks)

plt.legend(['Before Purchase', 'Before Waste', 'After Purchase', 'After Waste'])

plt.tight\_layout()

plt.show()

# Optimal Energy Storage Configuration

storage\_powers = np.linspace(0, 200, 20)

storage\_capacities = np.linspace(0, 500, 20)

costs = []

for power in storage\_powers:

for capacity in storage\_capacities:

total\_cost = simulate\_storage\_operation(power, capacity)[2]

costs.append(total\_cost)

storage\_powers, storage\_capacities = np.meshgrid(storage\_powers, storage\_capacities)

costs = np.array(costs).reshape(storage\_powers.shape)

fig = plt.figure(figsize=(10, 6))

ax = fig.add\_subplot(111, projection='3d')

surf = ax.plot\_surface(storage\_powers, storage\_capacities, costs, cmap='viridis')

fig.colorbar(surf, ax=ax, shrink=0.5, aspect=5)

ax.set\_xlabel('Storage Power (kW)')

ax.set\_ylabel('Storage Capacity (kWh)')

ax.set\_zlabel('Total Cost')

ax.set\_title('Optimal Energy Storage Configuration')

plt.show()

# main-3-1 三（1）题

import pandas as pd

# 读取各园区典型日负荷数据

load\_data = pd.read\_excel('./mnt/data/附件1：各园区典型日负荷数据.xlsx')

# 读取各园区典型日风光发电数据

generation\_data = pd.read\_excel('./mnt/data/附件2：各园区典型日风光发电数据.xlsx')

# 各园区装机容量

Pv\_A = 750 # kW

Pv\_C = 600 # kW

Pw\_B = 1000 # kW

Pw\_C = 500 # kW

# 计算各园区的光伏和风电实际出力

generation\_data['园区A 光伏出力(kW)'] = generation\_data['园区A 光伏出力（p.u.）'] \* Pv\_A

generation\_data['园区C 光伏出力(kW)'] = generation\_data['园区C 光伏出力（p.u.）'] \* Pv\_C

generation\_data['园区B 风电出力(kW)'] = generation\_data['园区B风电出力（p.u.）'] \* Pw\_B

generation\_data['园区C 风电出力(kW)'] = generation\_data['园区C 风电出力（p.u.）'] \* Pw\_C

# 合并负荷数据和发电数据

merged\_data = pd.merge(load\_data, generation\_data, on='时间（h）')

# 计算每日总负荷和总发电量

daily\_load = merged\_data[['园区A负荷(kW)', '园区B负荷(kW)', '园区C负荷(kW)']].sum()

daily\_generation\_A = merged\_data['园区A 光伏出力(kW)'].sum()

daily\_generation\_B = merged\_data['园区B 风电出力(kW)'].sum()

daily\_generation\_C = merged\_data[['园区C 光伏出力(kW)', '园区C 风电出力(kW)']].sum().sum()

# 打印每日总负荷和总发电量

print(f"园区A每日总负荷: {daily\_load['园区A负荷(kW)']} kWh")

print(f"园区B每日总负荷: {daily\_load['园区B负荷(kW)']} kWh")

print(f"园区C每日总负荷: {daily\_load['园区C负荷(kW)']} kWh")

print(f"园区A每日总光伏发电量: {daily\_generation\_A} kWh")

print(f"园区B每日总风电发电量: {daily\_generation\_B} kWh")

print(f"园区C每日总发电量: {daily\_generation\_C} kWh")

print("##### 独立运营方案计算 #####")

# 储能系统参数

power\_unit\_cost = 800 # 元/kW

energy\_unit\_cost = 1800 # 元/kWh

soc\_min = 0.1

soc\_max = 0.9

efficiency = 0.95

# 计算各园区储能需求

def calculate\_storage(daily\_load, daily\_generation):

# 需要储能的能量 = (每日负荷 - 每日发电) \* 充/放电效率

energy\_storage\_needed = max(0, (daily\_load - daily\_generation) / efficiency)

# 储能系统能量配置 = 储能需求 / (SOC最大值 - SOC最小值)

energy\_storage\_capacity = energy\_storage\_needed / (soc\_max - soc\_min)

# 储能系统功率配置 = 每日最大负荷

power\_storage\_capacity = daily\_load / 24 # 假设最大负荷平均分布

return energy\_storage\_capacity, power\_storage\_capacity

# 园区A储能配置

energy\_storage\_A, power\_storage\_A = calculate\_storage(daily\_load['园区A负荷(kW)'], daily\_generation\_A)

# 园区B储能配置

energy\_storage\_B, power\_storage\_B = calculate\_storage(daily\_load['园区B负荷(kW)'], daily\_generation\_B)

# 园区C储能配置

energy\_storage\_C, power\_storage\_C = calculate\_storage(daily\_load['园区C负荷(kW)'], daily\_generation\_C)

# 计算投资成本

def calculate\_investment\_cost(energy\_storage, power\_storage):

return energy\_storage \* energy\_unit\_cost + power\_storage \* power\_unit\_cost

# 园区A投资成本

investment\_cost\_A = calculate\_investment\_cost(energy\_storage\_A, power\_storage\_A)

# 园区B投资成本

investment\_cost\_B = calculate\_investment\_cost(energy\_storage\_B, power\_storage\_B)

# 园区C投资成本

investment\_cost\_C = calculate\_investment\_cost(energy\_storage\_C, power\_storage\_C)

# 打印结果

print(f"园区A储能系统能量配置: {energy\_storage\_A:.2f} kWh, 功率配置: {power\_storage\_A:.2f} kW, 投资成本: {investment\_cost\_A:.2f} 元")

print(f"园区B储能系统能量配置: {energy\_storage\_B:.2f} kWh, 功率配置: {power\_storage\_B:.2f} kW, 投资成本: {investment\_cost\_B:.2f} 元")

print(f"园区C储能系统能量配置: {energy\_storage\_C:.2f} kWh, 功率配置: {power\_storage\_C:.2f} kW, 投资成本: {investment\_cost\_C:.2f} 元")

print("##### 联合运营方案计算 #####")

# 计算整体储能需求

total\_daily\_load = daily\_load.sum()

total\_daily\_generation = daily\_generation\_A + daily\_generation\_B + daily\_generation\_C

# 联合运营储能配置

total\_energy\_storage, total\_power\_storage = calculate\_storage(total\_daily\_load, total\_daily\_generation)

# 联合运营投资成本

total\_investment\_cost = calculate\_investment\_cost(total\_energy\_storage, total\_power\_storage)

# 打印结果

print(f"联合运营储能系统能量配置: {total\_energy\_storage:.2f} kWh, 功率配置: {total\_power\_storage:.2f} kW, 投资成本: {total\_investment\_cost:.2f} 元")

print("##### 购电成本计算 #####")

import numpy as np

# 分时电价

peak\_price = 1.0 # 元/kWh

off\_peak\_price = 0.4 # 元/kWh

# 定义分时电价时段

def get\_electricity\_price(hour):

if 7 <= hour < 22:

return peak\_price

else:

return off\_peak\_price

# 计算购电成本

def calculate\_electricity\_cost(load\_data, generation\_data, daily\_load, daily\_generation):

total\_cost = 0

for hour in range(24):

load = load\_data.iloc[hour, 1:4].sum() # 每小时总负荷

generation = generation\_data.iloc[hour, 1:5].sum() # 每小时总发电量

net\_load = load - generation # 净负荷

if net\_load > 0:

price = get\_electricity\_price(hour)

total\_cost += net\_load \* price

return total\_cost

# 各园区独立运营购电成本

electricity\_cost\_A = calculate\_electricity\_cost(load\_data[['时间（h）', '园区A负荷(kW)']], generation\_data[['时间（h）', '园区A 光伏出力(kW)']], daily\_load['园区A负荷(kW)'], daily\_generation\_A)

electricity\_cost\_B = calculate\_electricity\_cost(load\_data[['时间（h）', '园区B负荷(kW)']], generation\_data[['时间（h）', '园区B 风电出力(kW)']], daily\_load['园区B负荷(kW)'], daily\_generation\_B)

electricity\_cost\_C = calculate\_electricity\_cost(load\_data[['时间（h）', '园区C负荷(kW)']], generation\_data[['时间（h）', '园区C 光伏出力(kW)', '园区C 风电出力(kW)']], daily\_load['园区C负荷(kW)'], daily\_generation\_C)

# 联合运营购电成本

electricity\_cost\_joint = calculate\_electricity\_cost(load\_data, generation\_data, total\_daily\_load, total\_daily\_generation)

# 打印购电成本

print(f"园区A每日购电成本: {electricity\_cost\_A:.2f} 元")

print(f"园区B每日购电成本: {electricity\_cost\_B:.2f} 元")

print(f"园区C每日购电成本: {electricity\_cost\_C:.2f} 元")

print(f"联合运营每日购电成本: {electricity\_cost\_joint:.2f} 元")

# 计算年购电成本

days\_per\_year = 365

annual\_cost\_A = electricity\_cost\_A \* days\_per\_year

annual\_cost\_B = electricity\_cost\_B \* days\_per\_year

annual\_cost\_C = electricity\_cost\_C \* days\_per\_year

annual\_cost\_joint = electricity\_cost\_joint \* days\_per\_year

print("##### 五年总购电成本和总投资成本 #####")

# 计算5年总购电成本

total\_cost\_5\_years\_A = annual\_cost\_A \* 5

total\_cost\_5\_years\_B = annual\_cost\_B \* 5

total\_cost\_5\_years\_C = annual\_cost\_C \* 5

total\_cost\_5\_years\_joint = annual\_cost\_joint \* 5

# 计算5年总投资成本（包括储能系统投资和购电成本）

total\_investment\_5\_years\_A = investment\_cost\_A + total\_cost\_5\_years\_A

total\_investment\_5\_years\_B = investment\_cost\_B + total\_cost\_5\_years\_B

total\_investment\_5\_years\_C = investment\_cost\_C + total\_cost\_5\_years\_C

total\_investment\_5\_years\_joint = total\_investment\_cost + total\_cost\_5\_years\_joint

# 打印5年总成本

print(f"园区A储能系统5年总成本: {total\_investment\_5\_years\_A:.2f} 元")

print(f"园区B储能系统5年总成本: {total\_investment\_5\_years\_B:.2f} 元")

print(f"园区C储能系统5年总成本: {total\_investment\_5\_years\_C:.2f} 元")

print(f"联合运营储能系统5年总成本: {total\_investment\_5\_years\_joint:.2f} 元")

# 比较不同方案的5年总成本

print("\n##### 5年总成本比较 #####")

print(f"园区A储能系统5年总成本: {total\_investment\_5\_years\_A:.2f} 元")

print(f"园区B储能系统5年总成本: {total\_investment\_5\_years\_B:.2f} 元")

print(f"园区C储能系统5年总成本: {total\_investment\_5\_years\_C:.2f} 元")

print(f"联合运营储能系统5年总成本: {total\_investment\_5\_years\_joint:.2f} 元")

# 总结最佳方案

if total\_investment\_5\_years\_joint < min(total\_investment\_5\_years\_A, total\_investment\_5\_years\_B, total\_investment\_5\_years\_C):

print("联合运营方案在经济上最优。")

else:

min\_cost = min(total\_investment\_5\_years\_A, total\_investment\_5\_years\_B, total\_investment\_5\_years\_C)

if min\_cost == total\_investment\_5\_years\_A:

print("园区A的独立运营方案在经济上最优。")

elif min\_cost == total\_investment\_5\_years\_B:

print("园区B的独立运营方案在经济上最优。")

else:

print("园区C的独立运营方案在经济上最优。")

# main-3-2.py

import pandas as pd

# 读取全年12个月的典型日风光发电数据

monthly\_generation\_data = pd.read\_excel('./mnt/data/附件3：12个月各园区典型日风光发电数据.xlsx')

# 查看数据结构

print("Monthly Generation Data:\n", monthly\_generation\_data.head())

from sklearn.preprocessing import MinMaxScaler

# 提取特征（去掉时间列）

features = monthly\_generation\_data.iloc[:, 1:]

# 数据标准化

scaler = MinMaxScaler()

features\_scaled = scaler.fit\_transform(features)

# 显示标准化后的数据

features\_scaled\_df = pd.DataFrame(features\_scaled, columns=features.columns)

print("Scaled Features:\n", features\_scaled\_df.head())

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

# 生成虚拟标签数据，这里假设我们需要预测的储能容量

# 假设使用每个月的总发电量的平均值作为储能需求的虚拟标签

labels = features\_scaled\_df.mean(axis=1)

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features\_scaled, labels, test\_size=0.2, random\_state=42)

# 训练随机森林回归模型

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# 预测和评估模型

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}")

# 使用训练好的模型进行全年储能需求预测

storage\_predictions = model.predict(features\_scaled)

print("Storage Predictions:\n", storage\_predictions)

# 经济性分析函数

def economic\_analysis(storage\_predictions, power\_price=800, energy\_price=1800, years=5):

power\_cost = storage\_predictions \* power\_price

energy\_cost = storage\_predictions \* energy\_price

total\_cost = power\_cost + energy\_cost

annual\_savings = storage\_predictions \* 365 \* (1 - 0.4) # 以节约的电费计算

return total\_cost, annual\_savings

# 计算经济性

total\_cost, annual\_savings = economic\_analysis(storage\_predictions)

# 输出经济性分析结果

for i, month in enumerate(monthly\_generation\_data.columns[1:], start=1):

print(f"{month} - 总投资成本：{total\_cost[i-1]:.2f} 元，年度节约成本：{annual\_savings[i-1]:.2f} 元")