

报名序号：007202

论文题目：A 题 园区微电网风光储协调优化配置
周铠然 · 仅供求职展示

园区微电网智能化储能优化与经济效益提升

摘 要

本研究聚焦于三个独立接入主电网的园区微电网，深入探讨了储能系统的最优配置方案及其对经济性的深远影响。

针对问题一，基于电力系统的基本供需平衡原理详细分析了各园区在未配备储能系统时的购电情况、弃风弃光电量、总供电成本以及单位电量的平均供电成本，并识别出影响经济性的关键因素，包括装机容量和发电效率。引入了 50kW/100kWh 储能系统后，本文通过强化学习模型对储能的充放电策略进行优化。这一举措显著减少了购电和弃电的总量，从而降低了供电成本。进一步利用线性规划模型对不同储能配置方案下的总供电成本进行了深入分析。结果表明，存在比 50kW/100kWh 更为优化的配置方案。经过优化，园区 A、B、C 的储能配置方案分别实现了显著的成本降低，不仅提升了经济性，也增强了能源利用效率。

针对问题二，在未配置储能的情况下，通过合并三个园区的数据，建立了包括光伏发电、风电发电和负荷数据的联合数据集，并基于此进行了经济性分析。结果显示，高峰购电量主要出现在夜间和下午晚高峰期间，而弃电量在中午时段达到峰值，主要是因为此时段光伏发电量最大而负荷需求较低。总供电成本和单位电量平均供电成本均在一天中呈现显著波动，高峰时段成本较高。为了优化储能配置并提升经济性，建立了线性规划模型、混合整数线性规划模型和基于遗传算法的多目标优化模型，并对三个模型进行准确度的评估和对比。线性规划模型以最小化总供电成本为目标，考虑发电量、负荷需求及储能系统约束。MILP 模型则引入二进制决策变量，更精细地处理充放电决策。模型求解结果展示出总弃风弃光电量接近零，总供电成本和单位电量平均供电成本也相对较低。进一步制定的联合园区最优储能配置方案的最低总成本为 3862.2 元。为此制定了储能运行策略和购电计划，以实现成本效益最大化。联合运营相比于独立运营，能够带来成本分摊、优化资源配置等经济优势，可以提高整体的发电效率和经济效益。还分析了导致经济收益改变的主要因素，包括储能系统的协调使用，规模经济和负荷平衡等。

针对问题三，对园区的风光储能系统进行了优化配置，具体包括风电和光伏的配比优化模型以及季节性出力的考虑。基于园区的负荷需求和可再生能源的发电特性，建立了风光配比优化模型，目标是最小化总成本，包括初始投资成本、运维成本和电费节省。通过线性规划方法，确定了风电和光伏的最优配置比例，使得在满足负荷需求的同时，最大限度地利用可再生能源，减少购电成本。在优化过程中，充分考虑了季节性出力的变化，根据全年的风光发电数据，分析各个季节中风电和光伏的发电量变化，调整风光配比，以应对不同季节的发电特性。

关键词：系统动力学模型；强化学习；线性规划；经济性分析；

一、 问题重述

在当前的能源转型背景下，越来越多的地区正在向可再生能源过渡，特别是风能和太阳能。然而，由于这些能源的间歇性和不稳定性，使得能源供需平衡成为一个挑战。为了提高可再生能源的利用率并减少对传统化石燃料的依赖，一种有效的策略是在微电网系统中整合储能技术。

本问题聚焦于三个具体的园区微电网，每个园区都独立接入主电网，并配置有风电和光伏发电设施。园区微电网面临的主要问题是风光发电的不稳定性及其与园区负荷的时序不匹配，导致发电过剩时需要弃电。为了优化能源利用并减少弃电，提出了引入储能系统的解决方案。储能系统，特别是电化学储能如磷酸铁锂电池，可以提供能量缓冲，使得可再生能源的生成更加符合消费模式。然而，储能系统的成本依然较高，需要仔细评估其经济性和配置方案。

本文分为三个主要的研究问题，如下：

问题 1：各园区独立运营储能配置方案及其经济性分析

- (1) 分析未配置储能时各园区的经济运行状况，包括购电量、弃风弃光电量、总供电成本和单位电量的平均供电成本，并分析影响其经济性的关键因素。
- (2) 探索各园区分别配置 50 kW/100 kWh 储能系统后，通过制定储能最优运行策略及购电计划，分析运行经济性的改善。
- (3) 假设风光功率波动特性保持上述条件不变，评估 50 kW/100 kWh 配置方案是否最优。

问题 2：联合园区储能配置方案及其经济性分析

- (1) 模拟三个园区合并为一个联合运营体的情景，分析未配置储能和配置储能后的经济性。
- (2) 设计并优化联合园区的总储能配置方案，制定储能运行策略及购电计划，并比较其与独立运营时的经济收益。

问题 3：园区风光储协调配置方案及其经济性分析

- (1) 针对三个园区最大负荷预期增长 50% 的情景，制定风光储协调的配置方案。
- (2) 使用全年 12 个月的风光发电数据和分时电价，为各园区独立运营制定并分析风光储协调配置方案的经济性。

二、 问题分析

2.1 问题一的分析

对于问题 1，首先进行数据的预处理，确保发电和负荷数据的一致性和准确性。鉴于风光发电的波动性及其与负荷需求的时序不匹配特点，本文提出采用系统动力学模型结合机器学习方法进行分析 and 优化。系统动力学模型能够模拟能源流动和存储动态，考虑到系统内各组成部分之间的复杂互动和反馈循环，为储能系统的运行提供了实时的调节机制。需要基于电力系统的基本供需平衡原理对各园区在未配备储能系统时的购电情况进行了详细评估。引入机器学习中的强化学习算法，旨在优化储能的充放电策略，以

应对能源生成和消费的实时变化。强化学习通过与环境的交互学习最佳策略，具有自我适应调整策略以最大化长期回报的能力。计算总供电成本以及单位电量的平均供电成本，这些指标能帮助了解在现有条件下供电的经济性，并识别出影响经济性的关键因素。引入 50kW/100kWh 的储能系统后将采用强化学习模型和线性规划模型对不同储能配置方案下的总供电成本进行了深入分析并比较不同配置方案的经济性。

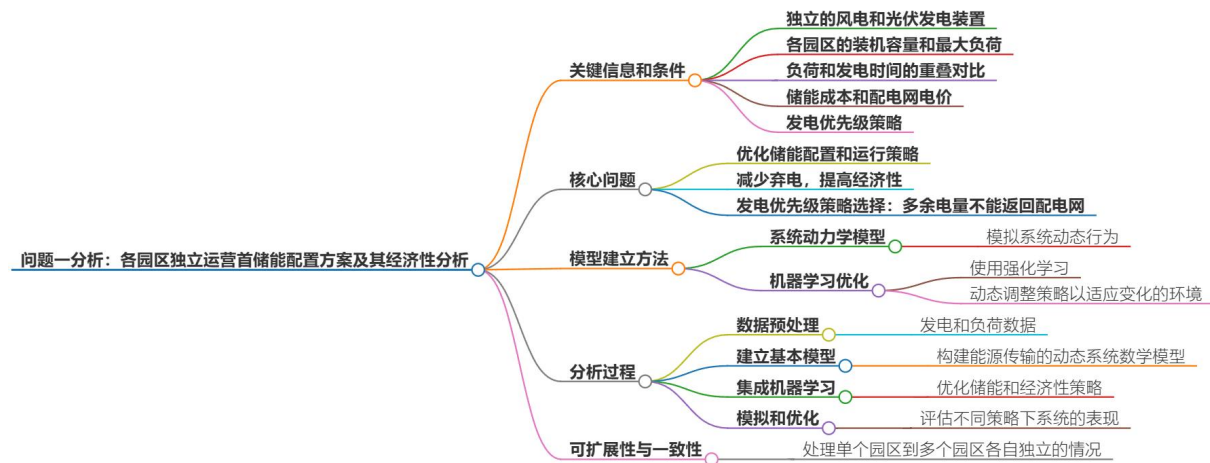


图 1 问题一分析过程

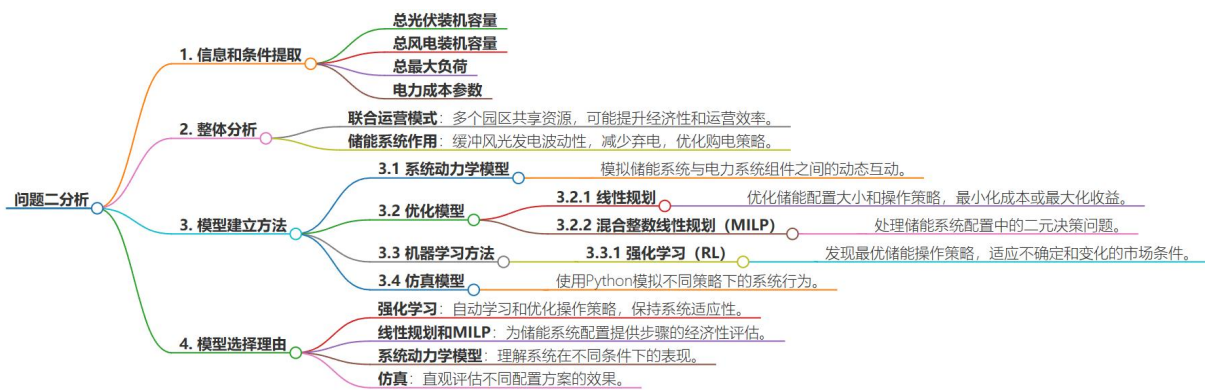
2.2 问题二的分析

对于问题二，需要分析了三个园区作为一个联合体进行储能配置及其经济性情况。首先提取题目中提供的关键信息，包括总光伏装机容量、总风电装机容量、总最大负荷，以及电力成本参数。整体分析表明，联合运营模式提供了一种在多个园区之间共享资源的方式，这可能带来经济性的提升和运营效率的改善。在这种模式中，储能系统发挥关键作用，通过缓冲风光发电的波动性，减少弃电，并利用电力市场策略优化购电成本。

为了深入分析，本文考虑了以下几种不同的模型和方法：

1. 系统动力学模型：用于模拟储能系统与园区内其他电力系统成分之间的动态互动。
2. 线性规划和混合整数线性规划 (MILP)：适用于优化储能配置的规模和操作策略，以最小化总成本或最大化经济收益。
3. 强化学习：这种机器学习方法适合动态调整策略，以实现长期的成本效益，特别适用于处理不确定和变化的市场条件。
4. 基于遗传算法的多目标优化：能够同时处理多个目标，通过模拟自然选择的过程来搜索最优解，目标包括最小化总供电成本和最大化可再生能源的使用。

上述几个模型支持复杂决策的制定。



2.3 问题三的分析

对于问题三，解决方案涉及到负荷特性分析、发电特性分析、电价分析、风光储协调配置方案制定、运营维护成本分析以及经济性分析。具体过程见下图，其能够为园区风光储能系统的协调配置提供一个经济有效的方案。



三、模型的建立与求解

3.1 问题一：各园区独立运营储能配置方案及其经济性分析

3.1.1 分析未配置储能时各园区运行的经济性

首先，分析未配置储能时各园区的经济运行状况。从附件提供的数据中获得各园区的光伏和风电发电量及负荷需求。通过计算每小时的实际发电量和负荷需求差异，我们可以确定购电量和弃电量。

由题目可知，各园区的风光装机容量和最大负荷参数如下：

园区	光伏装机容量(P_{pv})	风电装机容量(P_w)	最大负荷 (P_{Lmax})
园区 A	750 kW	-	447 kW
园区 B	-	1000 kW	419 kW
园区 C	600 kW	500 kW	506 kW

注意：园区 A 没有风电装机容量，园区 B 没有光伏装机容量，在表格中这些数据用短横线 (-) 表示

具体的计算过程如下：

① 购电量 $Q_{purchase}$ 的计算：

如果在某时刻的发电量 $Q_{generation}$ 小于负荷需求 Q_{demand} 意味着电力供应不足以满足需求，因此需要从外部购买电力以补足差额^[1]。购电量可以表示为：

$$Q_{purchase} = \max(0, Q_{demand} - Q_{generation}) \quad (1)$$

② 弃电量 $Q_{curtailment}$ 的计算：

弃电量的计算则是在发电量超过负荷需求时进行的。这意味着有多余的发电量未能被消耗，因此需要被“弃掉”或不用于供电。如果发电量 $Q_{generation}$ 大于负荷需求 Q_{demand} ，弃电量 $Q_{curtailment}$ 可以表示为：

$$Q_{curtailment} = \max(0, Q_{generation} - Q_{demand}) \quad (2)$$

③ 总供电成本 C_{total} 的计算

总供电成本的计算涉及到购电量和电价。如果购电量不为零，则需要计算购买这部分电力的成本。假设电价为 P 元/kWh，总供电成本 C_{total} 可以表示为购电量与电价的乘积：

$$C_{total} = P \times Q_{purchase} \quad (3)$$

④ 单位电量平均供电成本 $C_{average}$ 的计算

单位电量平均供电成本是指将总供电成本分摊到每单位的电量需求上。这可以反映出供电的平均成本。单位电量平均供电成本 $C_{average}$ 是总供电成本除以总电量需求：

$$C_{average} = \frac{C_{total}}{Q_{demand}} \quad (4)$$

以上计算方法是基于电力系统供需平衡的原理^[2]，通过量化购电和弃电的情况，以及计算相关的成本，从而对电力系统的经济性进行评估。这些计算方法在电力系统规划和运行中是常见且必要的，有助于电力系统运营商做出经济有效的决策。

计算结果的数据可视化如下：

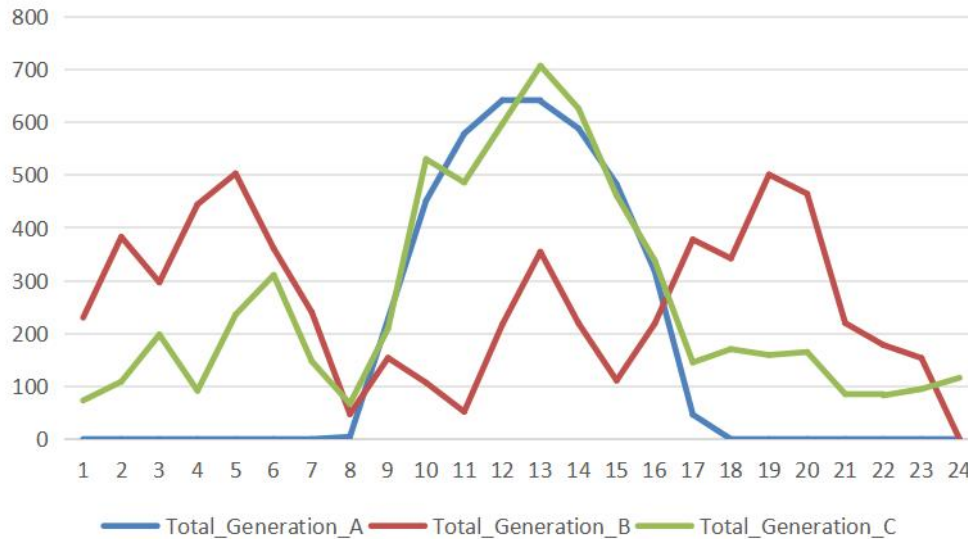


图 1 总发电量图

从图表中可以看出，园区 B 在大部分时间内发电量最高；园区 C 的发电量在某些时段也相对较高；园区 A 通常发电量较低。这表明园区 B 和 C 可能有更大的风电和光伏装机容量，或者更有利的气象条件。

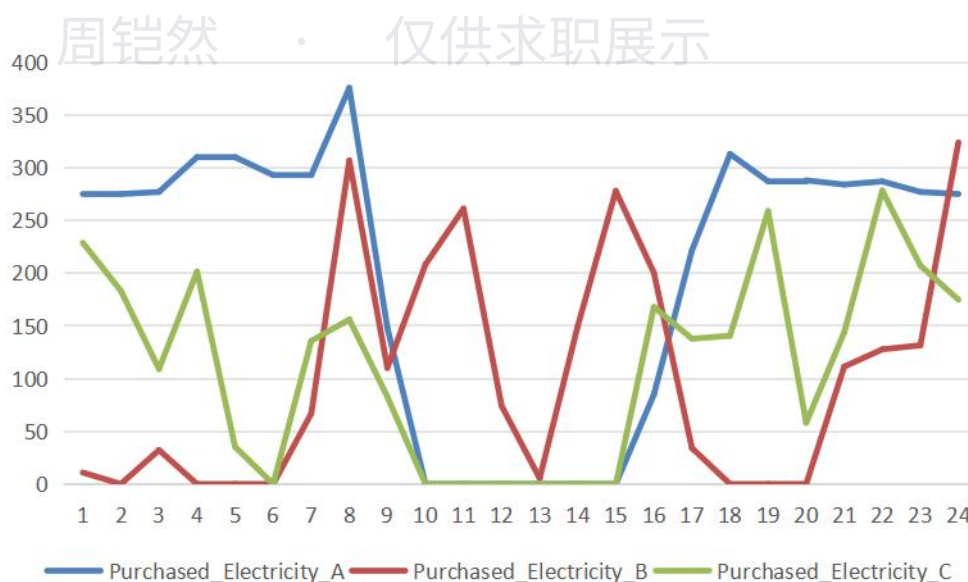


图 2 购电量图

购电量的图表显示，园区 A 在大多数时段需要购买较多的电力以满足负载需求，园区 C 在某些峰值时段也需要较多的电力，而园区 B 的购电量相对较少。这可能是因为园区 B 的发电能力更强，能够更好地满足自身的电力需求。

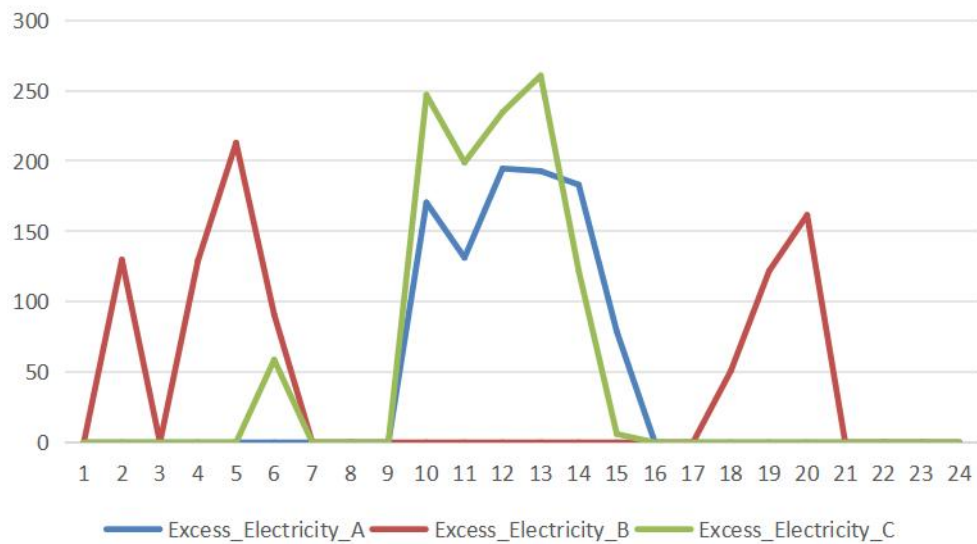


图 3 弃电量图

弃电量的图表显示园区 B 在某些高发电时段有较大量的电力被浪费，这可能是因为发电量远大于实际需求，且未配置储能系统以存储过剩电力。园区 A 和 C 的弃电量相对较少，但仍有峰值存在。

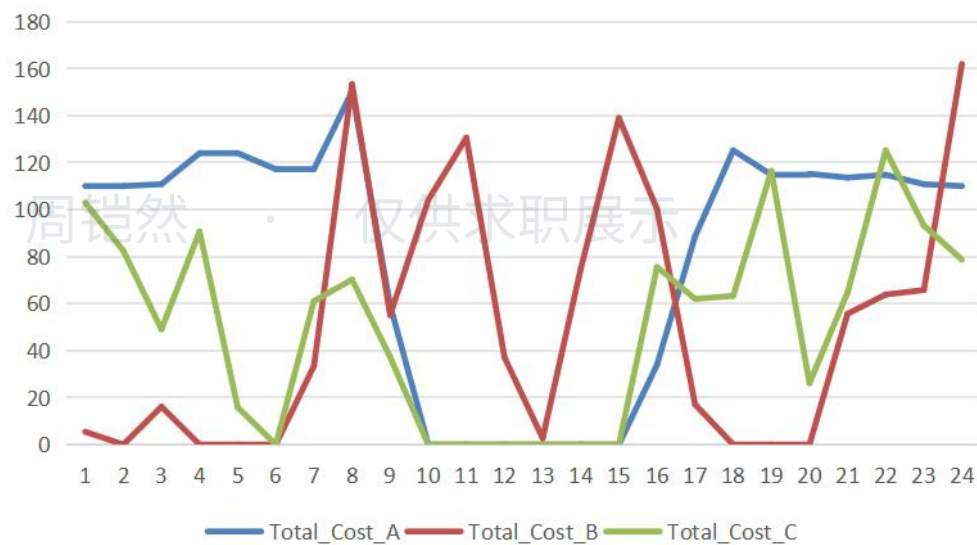


图 4 总成本图

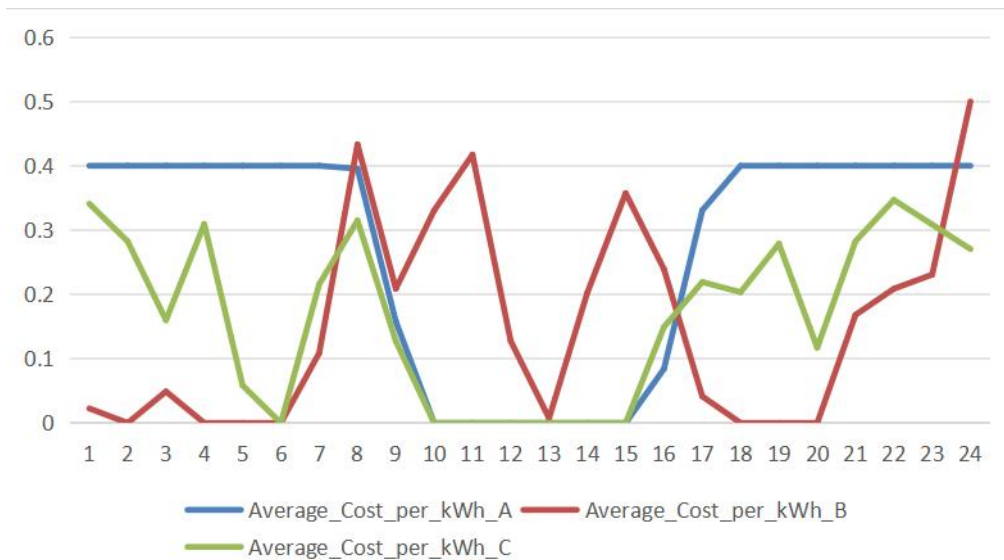


图 5 平均供电成本图

总成本和平均成本图表显示园区 A 的成本波动较大,这可能是由于较高的购电需求和价格波动。园区 C 在某些时段的平均成本显著高于其他园区,这可能与购电量的增加和发电能力的不足有关。

影响经济性的关键因素

1. 装机容量和发电效率.园区 B 的较高发电量表明其可能拥有更高的装机容量或更高效的设备。
2. 负荷需求波动.需求峰值直接影响购电量,尤其在发电不足以满足需求时。
3. 天气条件.风力和光照条件直接影响风电和光伏发电的效率,进而影响经济性。

以上这些分析表明,各园区应考虑增加储能容量以减少弃电,同时也可以通过优化设备和增加装机容量来提高自身发电能力,减少对外购电的依赖,从而提高经济性。在未来的策略中,应特别关注提高园区 B 的储能能力,以充分利用其高发电量,减少弃电情况。同时,园区 A 和 C 需要提高其发电效率或增加储能以应对高峰时段的电力需求。

3.1.2 系统动力学模型建立

对于园区微电网的储能系统优化,本文建立的系统动力学模型基于以下方程和约束条件来确保能量供应的有效性和系统的经济运行。这些方程和约束条件直接影响模型的输出和最终决策。

表 1 变量定义

$P_{gen,t}$	t 时刻的总发电量（风电和光伏之和）
$L_{dem,t}$	t 时刻的负荷需求
$E_{sto,t}$	t 时刻储能的能量状态
$C_{cha,t}$	t 时刻储能的充电
$C_{dis,t}$	t 时刻储能的放电量
$Storage_{A,B,C}$	储能系统电量

Total_Generation _{A,B,C}	各个园区的总发电量
Purchased_Electricity _{A,B,C}	各个园区的购电量
Excess_Electricity _{A,B,C}	各个园区的弃电量

接着，将进行方程建立过程。

(1) 方程一：能量平衡方程

这个方程是系统动力学模型的核心，确保了储能系统，即电池的能量状态在任意时间点都是前一时间点的能量加上当前时间段内的净充电量，因需考虑效率。并且确保任一时刻能量的输入输出平衡。

$$E_{sto,t} = E_{sto,t-1} + \eta_{cha} \times C_{cha,t} - \frac{1}{\eta_{dis}} \times C_{dis,t} \quad (5)$$

每个园区的能量平衡由以下方程确定：

$$\text{Storage}_{i,t+1} = \text{Storage}_{i,t} + \text{Charge}_{i,t} - \text{Discharge}_{i,t} \quad (6)$$

其中， i 表示 A,B,C 园区， t 表示时间同步

(2) 方程二：负荷满足方程

该方程确保在任何给定时间点，园区的能源需求得到满足。这里，确保任何时刻园区的能源需求得到满足，如果储能和发电不能满足需求，则需要从电网购买额外电力：

$$P_{gen,t} + C_{dis,t} - C_{cha,t} \geq L_{dem,t} \quad (7)$$

(3) 约束条件

各园区分别配置 50kW/100kWh 储能。

系统的容量限制： $10\% \leq \frac{E_{sto,i,t}}{100kWh} \leq 90\%$

需要对充电和放电流量进行控制，故充电和放电的操作需要基于当前发电量、负载和储能状态。

充电条件：当发电量大于负载需求时，

$$\text{Charge}_{i,t} = \min(\text{Excess Generation}_{i,t}, \text{Charge Power Limit} - \text{Storage}_{i,t}) \quad (8)$$

储能放电条件：当负载需求大于发电量时，

$$\text{Discharge}_{i,t} = \min(\text{Load Deficit}_{i,t}, \text{Storage}_{i,t}) \quad (9)$$

弃电量的最小化：尽量避免发电量超过负荷加储能容量的情况。

3.1.3 机器学习优化

强化学习（Reinforcement Learning, RL）是一种机器学习方法，它使得智能体能够在环境中通过试错来学习最佳行为策略，以最大化某种累积奖励。这种方法特别适合于

那些需要长期策略和决策的问题，如电力系统的能源管理。将机器学习特别是强化学习（RL）整合到电力系统管理中，可以极大地提升操作策略的动态调整能力和适应性。

首先，需要定义强化学习的环境，这涉及到对电力系统的建模，包括输入、输出和状态的详细定义。

表 2 环境定义

环境	包括电力市场的电价、发电量、负荷需求和天气条件
状态	系统的当前状态，例如储能的电量、当前电价、实时电价、当前时段的发电量、负荷和天气条件等
行动	智能体可以采取的行动，如多少电力从储能中放电、充电或保持当前储能电量不变
奖励	基于行动的结果来定义奖励，通常是成本节约的量度，例如通过智能调度减少的电力购买成本。

(1) 数据收集与预处理

根据以上方案，现在进行算法的具体实现，使用的是 Python 编程语言，并对结果数据进行可视化。

收集风光发电数据、负荷需求数据和电价数据。导入必要的 pandas 库，并设置数据文件路径。使用 pandas 的 read_excel 函数读取园区负荷数据和风光发电数据。根据光伏和风电的出力比例以及各自的装机容量，计算出每个园区的实际发电量。使用 pandas 的 merge 函数将负荷数据和发电数据合并到一个 DataFrame 中。读取 Excel 文件并整理为 DataFrame 格式，计算各园区的实际发电量和总发电量。

初始化储能状态，储能状态 Storage_{A,B,C} 和购买电力 Purchased_Electricity_{A,B,C} 初始设为为 0。

接着定义储能运行策略参数。定义充电和放电时机的规则，确保储能系统在发电量超过负荷需求时充电，在负荷需求超过发电量时放电。设置储能参数，定义储能容量、充电功率限制、放电功率限制和初始储能状态。

表 3 储能状态设定

参数名称	描述	设定数值	单位
storage_capacity	储能系统能够存储的最大电量	100	kWh
charge_power_limit	储能系统在单位时间内能充入的最大电力量	50	kW
discharge_power_limit	储能系统在单位时间内能放出的最大电力量	50	kW
initial_storage	开始时储能系统中已有的电量	50	kWh

定义充电和放电函数，并创建 charge 和 discharge 函数（公式 4、5）来计算充电量和放电量。

遍历每个时间步，对于每个园区（A, B, C），计算总发电量和负荷，根据当前状态决定是否充电或放电，然后根据充电和放电函数更新储能状态。以及计算每个园区的总成本和平均供电成本，具体过程见公式 3、4。

(2) 强化学习模型训练

在本文的园区微电网储能优化研究中，引入了强化学习模型，以进一步提升系统的

动态调整能力和适应性。

以下是强化学习模型的详细训练和优化过程。首先，需要定义强化学习的环境，包括状态空间、动作空间和奖励函数。

Step1.环境定义

状态空间包括当前储能系统的电量、园区 A 的负荷需求、园区 A 的总发电量、园区 B 的总发电量和园区 C 的总发电量，这五个变量构成了状态空间。

动作空间有充电、放电和保持当前状态这三个动作。充电是在发电量超过负荷需求时，将多余的电力存储到储能系统中；放电是在负荷需求超过发电量时，从储能系统释放电力；保持状态即不进行充电或放电操作。

奖励函数基于购电成本来定义。模型的目标是通过减少高成本电力的购买来最大化奖励。因此，当负荷需求高于发电量时，购电成本较高，奖励值较低；反之，当储能系统有效运行以减少购电需求时，奖励值较高。

Step2.模型定义

使用神经网络来构建强化学习模型。模型采用前馈神经网络结构，包含输入层、两个隐藏层和输出层。

输入层：接收当前状态空间的输入（即 5 个变量）。

隐藏层：两个隐藏层分别包含若干神经元，并采用 ReLU 激活函数，以捕捉状态与动作之间的复杂关系。

输出层：输出层有三个神经元，分别对应三个动作（充电、放电和保持状态），通过线性激活函数输出各动作的价值。

Step3.经验回放

为了更高效的完成这道解题，加上电脑配置等因素，代码使用了经验回放来提升训练效率，经验回放可以提高训练的效率和稳定性。

经验回放缓冲区存储过去的状态、动作、奖励和下一状态等信息，通过随机抽样的方式对模型进行训练，避免模型参数更新的高相关性。

在每一步中，记录当前状态、选择的动作、获得的奖励和下一状态，存储在经验回放缓冲区中。

在每个训练周期中，从缓冲区中随机抽取一小部分经验进行训练，更新模型参数。这种方式可以打破数据之间的相关性，提高训练效率。

Step4.模型训练

模型训练通过与环境的不断交互来进行，每个训练周期包括以下步骤：

- 状态初始化。从环境中获取初始状态，包括储能电量、负荷需求和发电量等信息。
- 选择动作。基于当前状态，使用 `epsilon-greedy` 策略选择动作。`epsilon-greedy` 策略在大部分情况下选择当前模型认为最优的动作，但偶尔会选择随机动作，以探索新的策略。
- 执行动作。在环境中执行所选动作，并记录获得的奖励和下一状态。
- 存储经验。将当前状态、动作、奖励和下一状态存储到经验回放缓冲区。

- e. 更新模型。从经验回放缓冲区中随机抽取一小部分经验，使用这些经验对模型进行训练。训练时，通过计算目标值和预测值的差异来更新模型参数，目标值由当前奖励和下一状态的最大预期奖励构成。
- f. 重复训练。重复上述过程，直到达到设定的训练周期数。

奖励函数定义如下：

$$R_t = -P_{\text{purchase},t} \cdot C_{\text{electricity}} \quad (10)$$

其中， R_t 表示时刻 t 的奖励值， $C_{\text{electricity}}$ 表示电价。

目标值如下：

$$y = R_t + \gamma \cdot \max Q(s_{t+1}, a') \quad (11)$$

其中， y 表示目标值， γ 表示折扣因子， $Q(s_{t+1}, a')$ 表示下一状态 s_{t+1} 下的最大预期奖励。

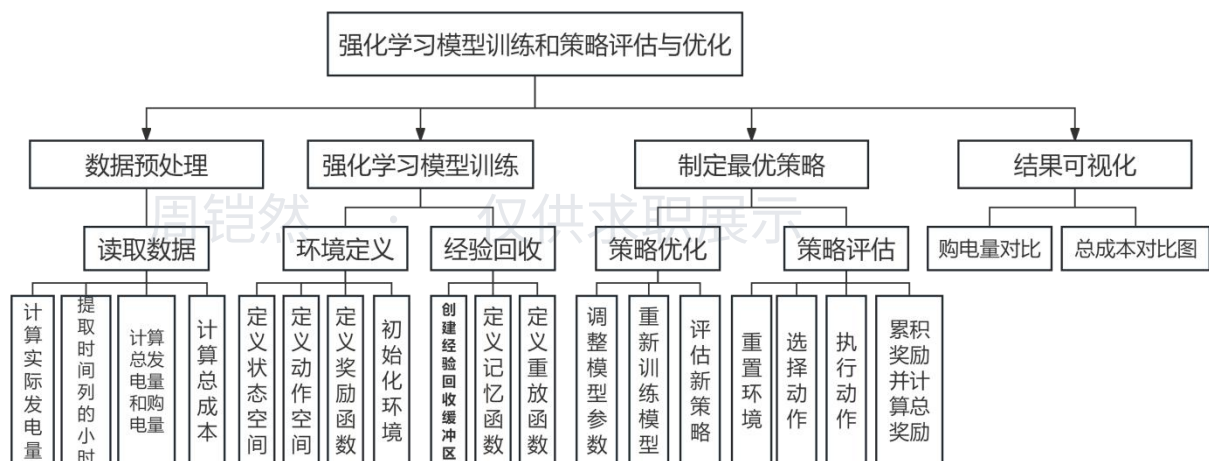


图 6 强化学习模型训练和策略评估与优化

(3) 策略评估和优化

在模型训练完成后，使用评估函数对训练好的策略进行评估。评估过程中，记录总奖励、购电量和总成本，判断储能系统对园区微电网运行经济性的改善效果。

评估训练好的模型策略在实际数据上的表现，计算优化前后的总成本。

评估步骤如下：

- a. 初始化评估环境。与训练过程相同，获取初始状态。
- b. 执行策略。按照训练好的模型策略执行动作，并记录每一步的奖励和购电量。
- c. 计算总奖励和总成本。累计计算整个评估过程中的总奖励和总成本，以评估储能系统的经济性。

根据评估结果，对模型进行进一步优化。可以通过调整模型参数、增加训练数据量或采用不同的奖励函数等方式优化策略。

优化方法说明如下：

- 调整模型参数。比如调整神经网络的层数和神经元数量，修改学习率等。
- 增加训练数据量。通过增加训练周期数或使用更多的历史数据来提升模型的泛化能力。在本问中是通过增加训练周期数来优化。
- 改进奖励函数。优化奖励函数，使其更能反映购电成本和储能系统运行的实际情况，从而提升模型决策的合理性。

(4) 经济性考量

比较储能系统运行前后的总成本，评估其经济性。

计算储能系统运行前的总成本 首先，我们需要计算在没有储能系统的情况下，各园区的购电量和总成本。

经过计算可知储能系统运行前总成本为 4492.29，接下来，需要在引入储能系统的情况下，计算各园区的购电量和总成本。经过计算得知，储能系统运行后总成本为 4214.97。总成本共减少了 277.311。这也可以说明储能系统的引入改善了经济性。

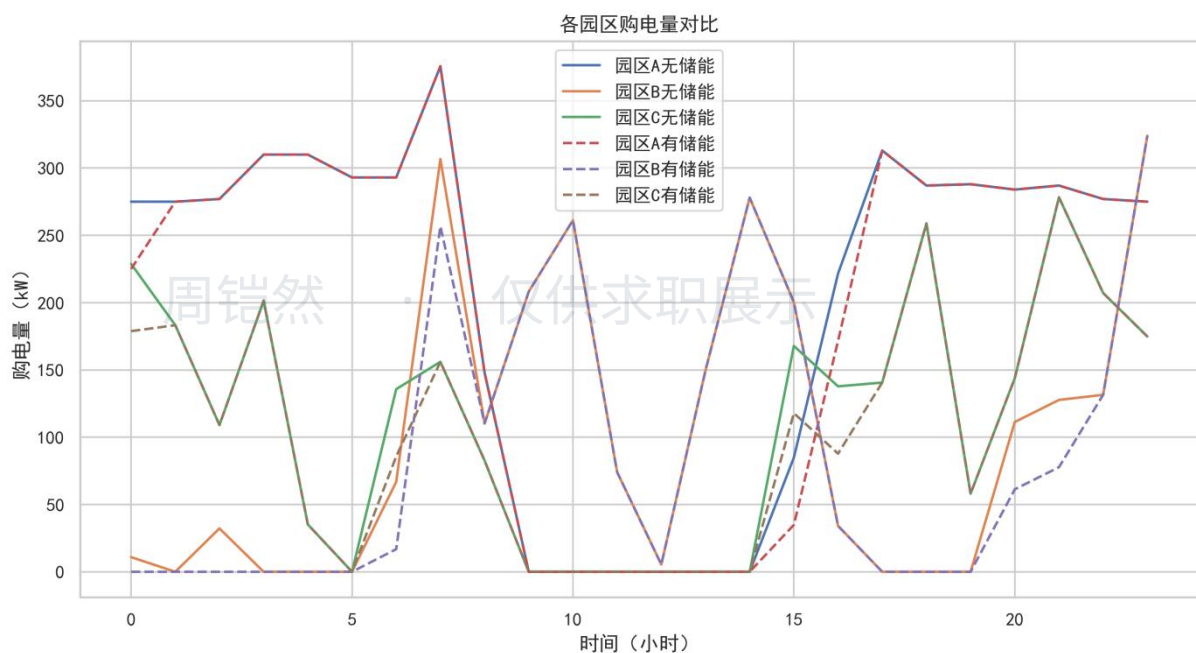


图 7 购电量对比图

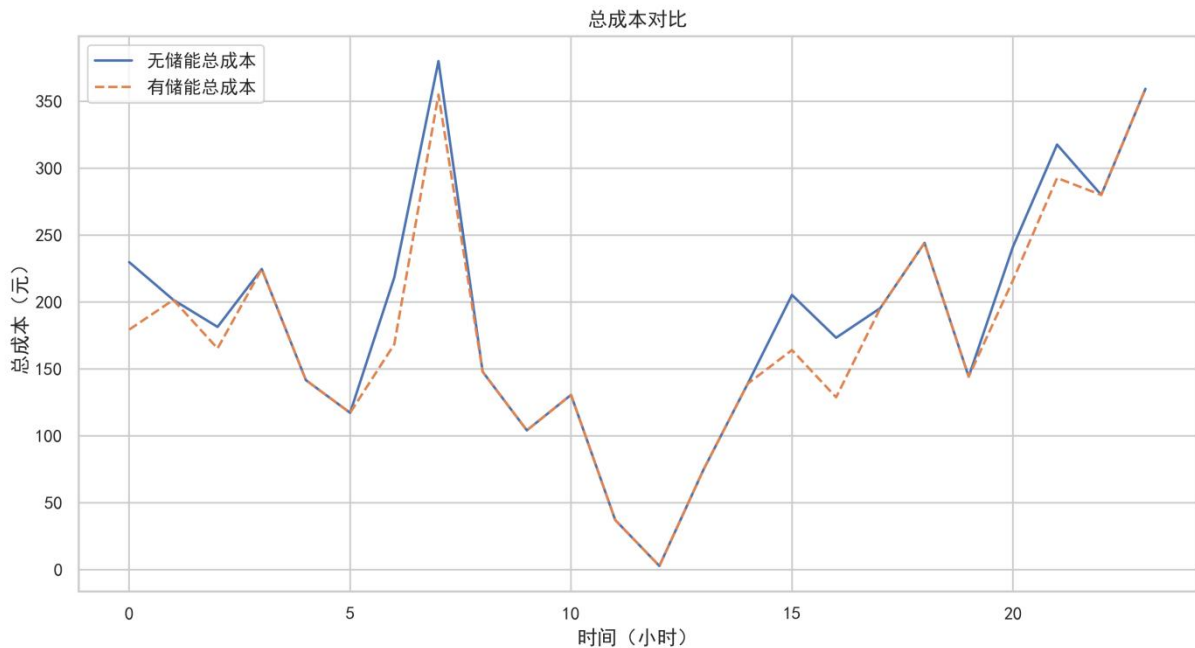


图 8 有无储能下的总成本对比图

制定的优化策略如下：

- 充电策略。在发电量超过负荷需求时，多余的电能存入储能系统，直到储能满载。
- 放电策略。在发电量无法满足负荷需求时，从储能系统释发电能补充差额。

通过强化学习模型的训练和优化，本文实现了储能系统充放电策略的自适应调整，提高了园区微电网的运行经济性。强化学习模型通过大量的模拟和交互，学到了在不同环境条件下的最优策略，进一步提升了储能系统的灵活性和效率。

3.1.4 储能最优运行策略

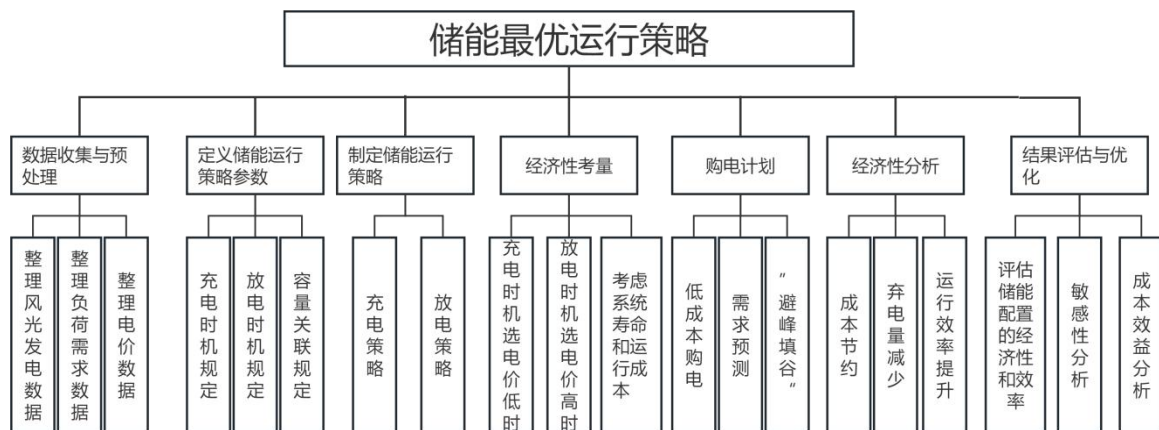


图 9 储能最优运行策略方案

为了分析储能系统对各园区运行经济性的影响，我们首先需要定义储能系统的运行策略。在这种情况下，在第（2）问中各园区分别配置了 50 kW/100 kWh 的储能系统，并通过充电和放电操作来平衡过剩电力和电力需求不足的情况。

储能运行策略可以分为充电操作和放电操作。充电操作是当园区的总发电量超过其负荷需求时，将多余的电力存储到储能系统中，直到储能满载或没有多余的电力；放电操作是当园区的总发电量无法满足负荷需求时，从储能系统释放电力，以满足剩余的负荷。



图 10 更新储能电量

先对储能系统使用情况进行分析，可知储能系统在园区 B 表现得尤为显著，几乎完全充满和完全放空，这表明园区 B 的储能系统被高效利用，可能是因为园区 B 的发电量与负荷波动较大，使得储能系统在调节峰谷差异中发挥了重要作用。

而园区 A 和 C 的储能使用较少，这可能是因为这些园区的总发电量与负荷需求更为匹配，或者发电量较少导致较低的弃电量，从而减少了储能系统的使用需求。

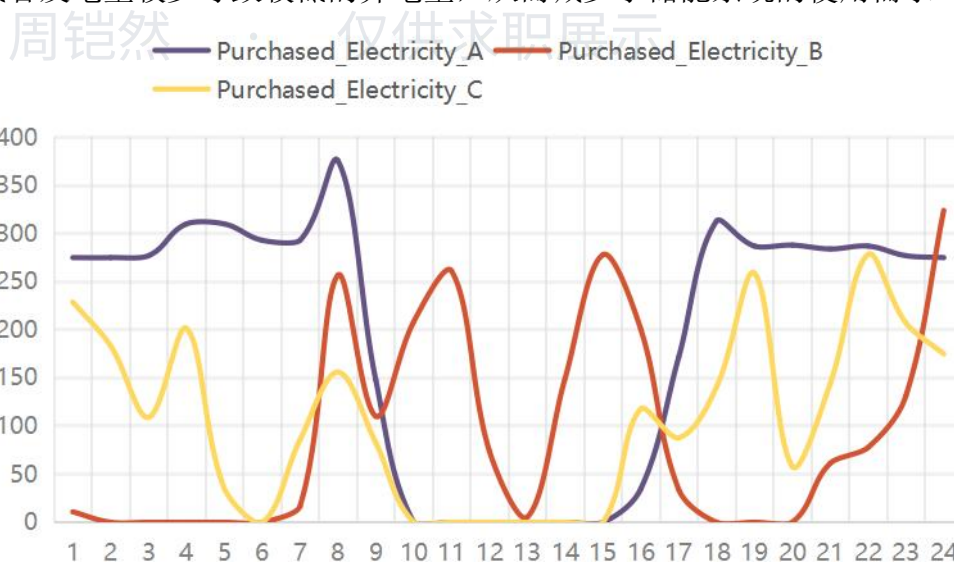


图 11 更新购电量

然后分析购电量的变化，从购电量的图表中可以看出，各园区的购电量波动仍然存在，但园区 B 的购电需求在晚上时段显著减少，这表明储能的放电在高需求时段有效地减少了对外购电的依赖。

园区 A 和 C 的购电量仍有显著波动，表明储能系统的容量可能不足以在需求高峰时完全替代购电，或者发电量不足以充分充电。

可见，储能系统的配置对园区 B 的运行经济性改善最为显著，通过平衡发电与负荷之间的波动，有效减少了高峰时段的购电需求，从而可能降低了电力成本和提高了能源自给自足率。对于园区 A 和 C，虽然储能系统的效果不如园区 B 显著，但在特定时段内仍有助于减少购电量和弃电量，提升能源使用效率。

为了进一步提升储能系统的经济效益，会考虑优化储能容量和放电策略，并制定储能最优运行策略，确保在电价最高或需求最大时使用储能电力，以及在发电量最高时进行充电，从而最大化经济效益。

为了制定储能最优运行策略并分析其对各园区运行经济性的影响，需要考虑以下三个关键因素：

- (1) **充电时机**：在电价较低或发电量高于需求，即弃电情况的时段进行充电，确保利用过剩的可再生能源，从而减少对高价电网电力的依赖。
- (2) **放电时机**：在电价较高或负荷需求超过发电量的时段进行放电，这样可以减少购买昂贵的电网电力，同时提供更经济的电力支持。
- (3) **容量管理**：维持储能在一个安全的运行区间，如 20%至 80%的 SOC，以延长储能系统的寿命并提供充足的反应余地。

综合考虑了上述三个关键因素，认为最优的储能运行策略应最大化使用可再生能源，并在高成本时段减少对外购电。基于此，本文将制定以下两个策略作为最优运行策略：

① **充电策略**：在日间或风力较强时段，即当发电量超过当前负荷需求时，进行储能充电。充电应控制在不起引起设备过度负荷的范围内，同时保证储能设备不超过其最大容量。

② **放电策略**：在高需求时段，特别是当电价较高或外购电力成本增加时，从储能设备中释放电力。确保储能设备在关键时刻有足够的存储能量来满足需求。

3.1.5 制定购电计划

购电计划需要考虑到电价波动、总需求和储能状态这几个因素。

1. 我们需要进行低成本购电。也就是在电价最低的时段，这个时间段通常是夜间或需求低谷期，积极购电，以备不时之需，同时考虑将额外的低成本电力存储于储能设备中。
2. 对需求进行预测，可以根据历史数据和天气预报来预测日负荷曲线，结合储能容量和状态来决定实际购电量。
3. 需要“避峰填谷”，即在电力需求高峰时段，优先使用储能设备中的电力，减少高峰时段的购电量，从而降低成本。
4. 通过储能系统的配置，可以显著减少弃电量，同时减少高峰时段的购电需求。这将降低总供电成本，并提高单位电量的经济性。

3.1.6 分析各园区运行经济性是否改善，并解释其原因

回答：各园区运行经济性确实得到了改善。

从经济性的角度分析，经过计算可知储能系统运行前总成本为 4492.29，储能系统运行后总成本为 4214.97。总成本共减少了 277.311。这也可以说明储能系统的引入改善了经济性。具体过程见(2)经济性考量部分。

通过储能设备的充电和放电操作，各园区可以在电价较低时储存能量，并在电价高时使用存储的电力，从而减少了高成本的电力购买，降低了电力成本。

通过储能系统的应用，可再生能源的过剩电力不再被浪费，而是被储存起来，增加了能源的利用效率，减少了弃电。

储能系统提供了更多的能量管理灵活性，使得园区可以更有效地管理其能源需求和供应，提高了运行效率和能源自给自足率。

具体的解释如下：

- (1) 对于园区 B，由于其较大的发电波动，通过储能系统的配置，可以更有效地管理这些波动，尤其是在高发电量时存储能量，在高需求时释放，减少了购电成本。
- (2) 对于园区 A 和 C，尽管这些园区的发电量相对平稳，但在需求峰值时刻，储能仍能发挥关键作用，通过提供必要的电力来减少昂贵的电力购买。

通过合理设计和运行储能系统，可以明显提高园区的运行经济性。这不仅降低了能源成本，而且提高了可再生能源的利用效率和能源自给自足率。未来，可以通过数据分析和更智能的能源管理系统，进一步优化储能运行策略，实现能源使用和经济效益的双重优化。

3.1.7 模型的建立与求解

假设风光荷功率波动特性保持上述条件不变，为了判断 50kW/100kWh 的方案是否最优，接下来会进行线性规划模型的建立，以分析该方案是否为最优的。

在模型的建立之前我们从经济性角度可以得知引入 50kW/100kWh 的储能系统后，可以看到明显的经济性改善。具体来说，储能系统运行前的总成本为 4492.29 元，而运行后的总成本降低到了 4214.97 元，实现了成本减少 277.311 元。这一成本的降低直接证明了储能系统在经济性方面的积极作用。从这方面确实可以判断这个方案是有优化的。

具体是否为最优方案，请看以下分析过程。

- a. 对于线性规划模型的建立过程，详细过程在问题三的 5.2.3 模型一的建立这一部分。

决策变量的定义和该部分一样，目标函数是最小化各个园区的总供电成本 C_{total} ，目的是找出成本最低或效益最大的配置方案。

目前通过计算得知 50kW/100kWh 这个方案得到的各个园区的一天的总供电成本数据如下：

表 4 50kW/100kWh 方案的数据表

园区	一天的总供电成本	未配置储能系统	相差
园区 A	1889.65 元	1949.65 元	60.00 元
园区 B	1094.61 元	1216.15 元	121.54 元
园区 C	1249.69 元	1214.73 元	-34.96 元

通过对比可知，园区 C 的差额是负数，这意味着园区 C 配置储能系统的成本比未配置储能系统的成本要高，这可能是由于储能系统的固定成本导致，或者是因为储能系统在园区 C 的运行策略没有优化到最佳状态。因此 50kW/100kWh 的方案并不是最优的。

根据以上制定的储能最优运行策略，现在需要对方案进行优化，制定各园区最优的储能功率、容量配置方案。

读取负荷和发电数据，计算实际发电量，并合并负荷和发电数据。

线性规划求解函数 `run_lp` 函数定义了线性规划问题，包括充电功率、放电功率、储能量和购电量的决策变量，并定义了优化目标和约束条件（具体见 5.2.3 部分）。

此时通过引入了强化学习模型，并得到前十个最优解的方案，从中再进一步筛选，还需要进行多种储能方案的比较，使用 `optimize_park` 函数对多种储能方案进行比较，包括 50kW/100kWh、100kW/200kWh、200kW/400kWh 和 400kW/800kWh，选择总供电成本最低的方案。

以下找到每个园区的最优储能功率和容量配置方案的思维过程脑图



图 12 寻找最优储能配置方案过程脑图

由此前分析可知，园区 C 在某些时段的平均成本显著高于其他园区，这可能与购电量的增加和发电能力的不足有关。园区 A 和 C 的弃电量相对较少，但仍有峰值存在。园区 C 在某些峰值时段也需要较多的电力。结合表 4 数据，故只为园区 C 的配置找到最优的配置方案，其他两个园区的配置方案还是不变。

园区 C 最优储能配置：功率 50kW，容量 50kWh，最低总成本：1386.315 元

优化配置方案考虑了园区 C 在某些时段的峰值电力需求，确保系统在负荷高峰期能够提供足够的电力。

3.2 问题二：联合园区储能配置方案及其经济性分析

3.2.1 数据预处理

在没有储能的情况下，联合园区的运行经济性主要包括总购电量、总弃风弃光电量、总供电成本和单位电量平均供电成本。

在问题二中，本研究是将三个园区合并为一个联合运营体，并分析未配置储能和配置储能后的经济性。

首先，将三个园区的数据合并成一个联合数据集。包括各园区的光伏发电、风电发电和负荷数据。

合并后的总发电量 $P_{gen, total}$:

$$P_{gen, total, t} = P_{gen, A, t} + P_{gen, B, t} + P_{gen, C, t} \quad (1)$$



图1 联合园区总发电量

合并后的总负荷需求 $L_{dem, total}$:

$$L_{dem, total, t} = L_{dem, A, t} + L_{dem, B, t} + L_{dem, C, t} \quad (2)$$



图2 联合园区总负荷需求

3.2.2 计算未配置储能时的经济性

问题二的分析是基于问题一的计算过程，进一步按要求进行合并分析。

从问题二的图1联合园区总发电量中可以看到，不同时段의总购电量存在显著波动。高峰购电量出现在夜间和下午晚高峰期间，而在中午和夜间的购电量较低。这表明在发电量不足以满足负荷需求时，需要从电网购入电量。高购电量时段对应着发电量低、负荷需求高的时间段。

从问题二的图2联合园区总负荷需求中可以看到，总弃风弃光电量在中午达到峰值。这是因为在中午太阳辐射最强，光伏发电量最大，而此时负荷需求相对较低，导致部分发电量无法被消纳，从而产生弃电现象。

了解以上两个现象后，开始进行联合园区的相关计算和经济性分析。

- a. 计算未配置储能时的购电量。购电量是在发电量小于负荷需求时需要从电网购入的电量。具体购电量计算公式见问题一的公式(1)。

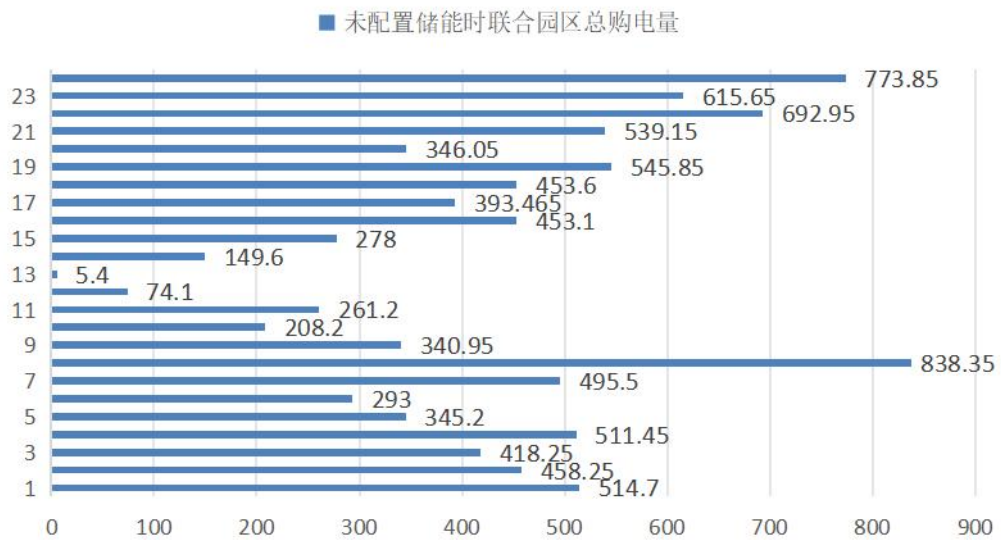


图 3 联合园区总购电量

高峰购电量出现在夜间和下午晚高峰期间，购电量直接影响总供电成本。

- b. 计算未配置储能时的弃电量。弃电量是在发电量大于负荷需求时多余的电量。弃电量计算公式具体见问题一的公式(2)

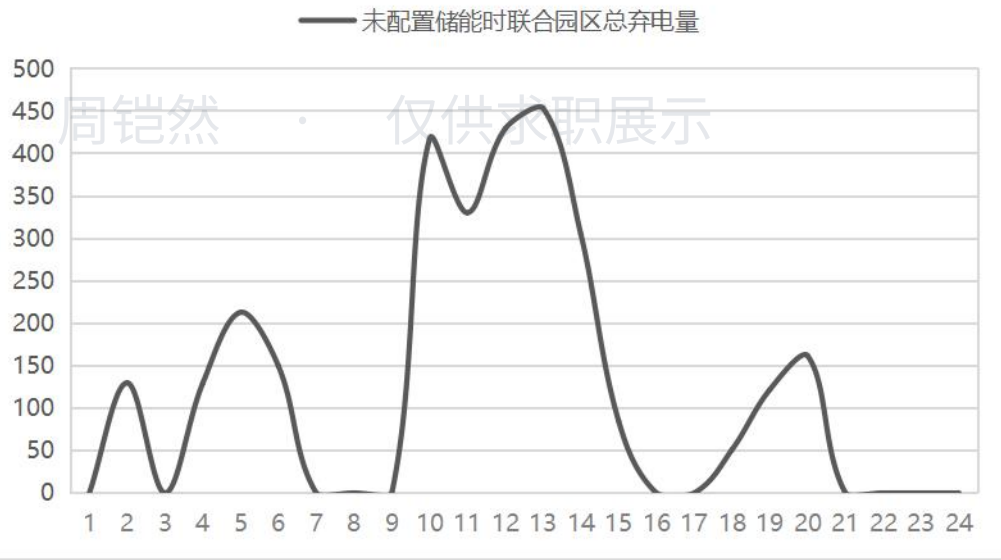


图 4 联合园区总弃电量

中午的弃电量最大，表明此时光伏发电量大，负荷需求低，导致弃电现象严重。从图 4 中可以看到，总弃风弃光电量在中午达到峰值。

- c. 计算未配置储能时的总供电成本。购电量乘以电价得出总成本。总供电成本计算公式具体见问题一的公式(3)

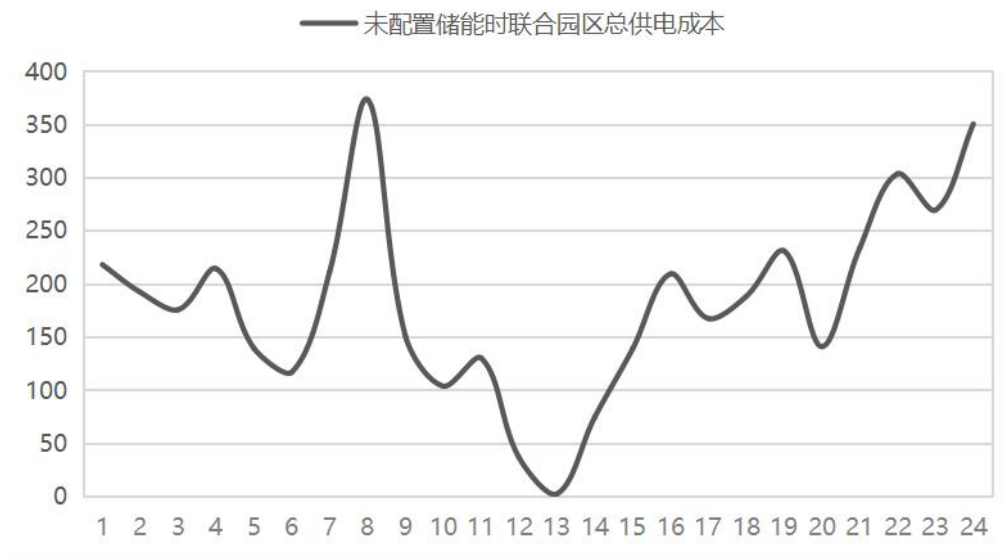


图 5 联合园区总供电成本

高峰供电成本与购电量直接相关，高购电量时段供电成本较高。

从图 5 可以看到，总供电成本在一天内有显著波动。高峰供电成本出现在购电量高的时段，如夜间和下午晚高峰期间，而在中午和夜间的供电成本较低。这表明购电成本与购电量直接相关。

- d. 计算单位电量的平均供电成本。单位电量平均供电成本计算公式具体见问题一的公式(4)

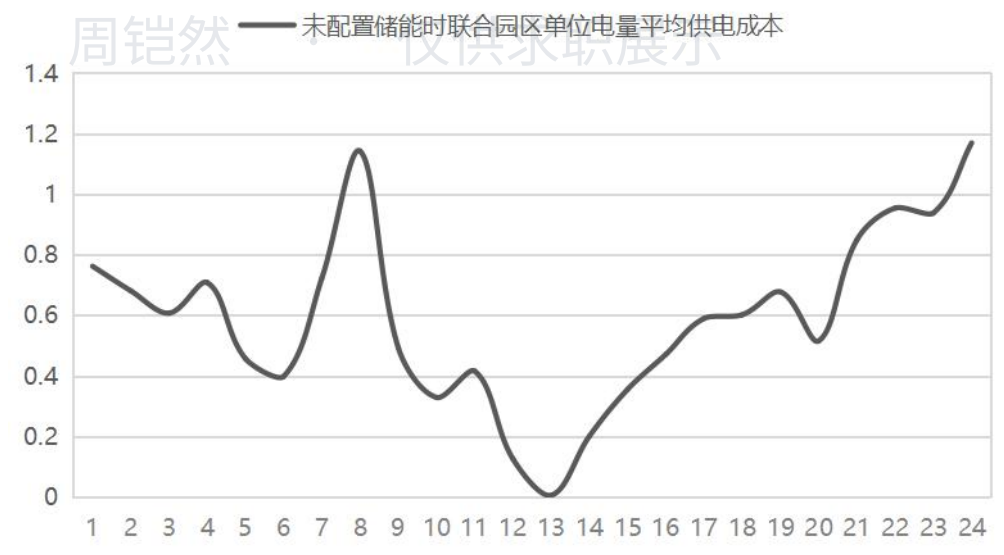


图 6 联合园区单位电量平均供电成本

单位电量平均供电成本在高峰时段较高，表明购电成本较高。

从图 6 可以看到，单位电量平均供电成本在一天内的波动较大。高峰时段单位电量平均供电成本较高，低谷时段单位电量平均供电成本较低。这表明单位电量平均供电成本受购电量和总供电成本的影响较大。

通过以上的分析，得到以下经济性的总结：

1. 总购电量分析，高峰购电量出现在夜间和下午晚高峰期间，购电量直接影响总供电成本。
2. 总弃风弃光电量分析，中午的弃电量最大，表明此时光伏发电量大，负荷需求低，导致弃电现象严重。
3. 总供电成本分析，高峰供电成本与购电量直接相关，高购电量时段供电成本较高。
4. 单位电量平均供电成本分析，单位电量平均供电成本在高峰时段较高，表明购电成本较高。

通过优化发电和储能系统的运行策略，可以减少高峰购电量和弃电量，从而降低总供电成本和单位电量平均供电成本，提升联合园区的经济性。

故接下来会制定联合园区的总储能最有配置方案，并给出储能运行策略及购电计划，并重新分析其经济性。

3.2.3 模型一的建立（线性规划模型）

三个园区独立运营时，各园区独立实现发电（风光发电、网购电）与负荷均衡。若三个园区总发电与总负荷相均衡，形成联合运营园区。以典型日风光发电和负荷功率为依据，设园区从风电、光伏的购电成本分别为 0.5 元/kWh、0.4 元/kWh。

基于上述思想需要建立线性规划模型，该模型以最小化总供电成本为目标，并且考虑到了发电量、负荷需求以及储能系统的约束。

表 1 决策变量

变量	说明
$C_{cha,t}$	储能系统的充电功率
$C_{dis,t}$	储能系统的放电功率
$P_{purchase,t}$	电网购电量
$E_{sto,t}$	为状态变量，储能系统的当前电量

注意：后续模型有相同决策不重复编写

b. 约束条件：

1. 储能系统的容量限制：

$$0 \leq E_{sto,t} \leq E_{sto,max} \quad (3)$$

2. 储能系统的充放电功率限制：

$$\begin{aligned} 0 \leq C_{cha,t} \leq C_{cha,max} \\ 0 \leq C_{dis,t} \leq C_{dis,max} \end{aligned} \quad (4)$$

3. 储能系统的能量平衡方程：

$$E_{sto,t} = E_{sto,t-1} + \eta_{cha} \cdot C_{cha,t} - \frac{1}{\eta_{dis}} \cdot C_{dis,t} \quad (5)$$

4. 电力平衡约束:

$$P_{\text{purchase},t} = L_{\text{dem,total},t} - P_{\text{gen,total},t} - C_{\text{dis},t} + C_{\text{cha},t} \quad (6)$$

c. 目标函数:

最小化总供电成本 C_{total} :

$$C_{\text{total}} = \sum (P_{\text{purchase},t} \times C_{\text{electricity}}) \quad (7)$$

其中, 购电量 $P_{\text{purchase},t}$ 和弃电量 $P_{\text{curtail},t}$ 为:

$$\begin{aligned} P_{\text{purchase},t} &= \max(0, L_{\text{dem,total},t} - P_{\text{gen,total},t} - C_{\text{dis},t}) \\ P_{\text{curtail},t} &= \max(0, P_{\text{gen,total},t} - L_{\text{dem,total},t} - C_{\text{cha},t}) \end{aligned} \quad (8)$$

3.2.4 模型一的求解与结果分析

根据以上思想和模型求解的原理, 用 python 语言进行求解, 代码见附录二, 以下是使用线性规划 (LP) 模型求解储能系统的最优运行策略的结果:

表 2 模型一的求解结果

总购电量	6375.869559999999 kWh
总弃风弃光电量	7.105427357601002e-14 kWh
总供电成本	3187.9347799999996 元
单位电量平均供电成本	0.14 元/kWh

以下是对结果进行分析:

1. 从结果可以发现, 总弃风弃光电量是一个非常小的数字, 接近于零。因此能系统的配置和运行策略似乎非常有效, 因为几乎没有可再生能源被弃用。这意味着在这一天内, 几乎没有风能和太阳能被浪费或"弃掉"。这可能是因为储能系统有效地存储了过剩的可再生能源, 并且园区的运行策略很好地匹配了能源生产和消费。
2. 总供电成本相对较低, 这可能意味着园区通过优化储能使用和购电计划, 实现了成本节约。
3. 平均供电成本为 0.14 元/kWh, 这个成本可能低于某些地区的零售电价, 显示出园区微电网优化配置的经济效益。

模型一的结果中, 联合园区的总供电成本约为 3187.9347 元, 而未配置储能时的联合园区总供电成本为 4380.5255 元, 可见该模型可以显著降低总供电成本和单位电量平均供电成本, 同时提高可再生能源利用率, 从而实现了经济性和环保效益的双重目标。

3.2.5 模型二的建立 (混合整数线性规划模型)

混合整数线性规划 (MILP) 可以同时处理连续和离散变量。

模型二的建立将考虑包括储能的充电和放电决策在内的不同变量, 并引入二进制决策变量来处理特定操作的开/关状态。

根据题目描述, 各园区可再生能源发电优先供给本园区负荷, 不足部分从主电网购电, 多余电量不允许向主电网出售。配置储能系统的目的是优化能源使用, 减少购电成本, 并最大限度地利用可再生能源。

模型二的目标是最小化联合园区的总供电成本，同时确保电力系统的稳定性和储能系统的有效利用。

模型二的决策变量是在表 1 的基础上再加入一个变量 $x_{on,t}$ ，这个是二进制变量，表示储能系统在第 t 时段是否放电（1 表示放电，0 表示不放电）。

a. 约束条件：

模型二的约束条件包括了模型一的三个约束（公式 4，5，6），其中储能系统容量约束为：

$$0.1 \times E_{sto,max} \leq E_{sto,t} \leq 0.9 \times E_{sto,max} \quad (9)$$

以及增加了一个约束，避免同时充电和放电：

$$x_{on,t} + x_{cha,t} \leq 1 \quad (10)$$

其中 $x_{cha,t}$ 是当 $C_{cha,t} > 0$ 时为 1，否则为 0 的二进制变量

3.2.6 模型二的求解与结果分析

MILP 模型通过引入二进制决策变量，可以更灵活地处理操作模式的切换，如启动或关闭储能系统的充放电，这有助于实现更精细的能源管理和成本控制。该模型特别适合处理有开关行为的系统，比如储能系统，可以使得整体系统运行更加高效和经济。



图 7 混合整数线性规划模型求解过程

模型二的最优解结果如下：

表 3 模型二的求解结果

时间/h	充电功率 (kW)	放电功率 (kW)	储能电量 (kWh)	购电量 (kWh)	放电状态
0	0	189.47	180	514.7	关闭
1	0	0	0	328.45	关闭
2	0	0	0	418.25	关闭
3	0	0	0	382.05	关闭
4	0	0	0	132.3	关闭
5	0	0	0	143.35	关闭

6	0	0	0	495.5	关闭
7	0	0	0	838.35	关闭
8	0	0	0	340.95	关闭
9	209.29	0	0	0	关闭
10	69.08	0	198.83	0	关闭
11	355.23	0	264.46	0	关闭
12	448.12	0	601.92	0	关闭
13	155.46	0	1027.63	0	关闭
14	0	0	1175.32	193.21	关闭
15	0	0	1175.32	453.1	关闭
16	0	800	1175.32	393.46	关闭
17	0	247.7	415.32	402.5	关闭
18	0	0	180	424.05	关闭
19	0	0	180	184.45	关闭
20	0	0	180	539.15	关闭
21	0	0	180	692.95	关闭
22	0	0	180	615.65	关闭
23	0	0	180	773.85	关闭

可以看到在一天 24 小时中，储能系统的运行策略如下：

时间 0-8: 储能系统未进行放电，而是从主电网购电以满足负荷需求。这可能是因为在这些时间段内，电价较低，或者风光发电不足以满足负荷。

时间 9: 开始充电，功率为 209.29 kW，这可能是在电价较低的时段，利用风光发电或从主电网购买电力来充电。

时间 10-13: 继续充电，充电功率逐渐增加至 448.12 kW，储能电量也随之增加。

时间 14: 储能系统充满至 1175.32 kWh，然后停止充电。

时间 15: 储能系统保持充满状态，没有放电，继续从主电网购电。

时间 16: 开始放电，功率为 800.00 kW，同时从主电网购电 393.46 kWh，可能是为了在高峰时段提供额外的电力支持。

时间 17: 放电功率减少至 247.70 kW，储能电量减少至 415.32 kWh，同时购电量为 402.50 kWh。

时间 18-23: 储能系统不再放电，储能电量维持在 180.00 kWh，从主电网购电以满足负荷需求。

从结果中可以看出储能系统在一天中的运行策略是：

1. 在电价较低的时段充电。
2. 在高峰负荷时段放电，以减少从主电网的购电量。
3. 保持一定的储能电量作为备用。

3.2.7 模型三的建立（基于遗传算法的多目标优化模型）

基于遗传算法的多目标优化是一种强大的搜索技术，它可以同时处理多个目标，并且适用于解决复杂的优化问题，特别是当问题具有多个决策变量和非线性目标时。

遗传算法通过模拟自然选择的过程来搜索最优解。其基本步骤包括初始化种群、选择、交叉（杂交）、变异和代替。

模型三的目标包括最小化总供电成本和最大化可再生能源的使用。



图 8 遗传算法优化过程

模型三的参数有两个，一是各园区的可再生能源发电能力和需求；二是储能系统的容量和效率。

约束条件同模型一。

目标函数为 C_{total} （公式 7）和最大化可再生能源利用率 $R_{utilization}$ ：

$$R_{utilization} = \frac{\sum_t (P_{gen, total, t} - P_{curtail, t})}{\sum_t P_{gen, total, t}} \quad (11)$$

3.2.8 模型三的求解与结果分析

遗传操作有种群初始化、选择、交叉和变异，一下是具体的求解过程：

1. 初始化与种群设定

在初始代，算法随机生成 50 个个体作为起始种群。这些个体代表不同的储能系统操作策略，每个个体由 24 个决策变量组成，这些变量控制每小时的储能系统充电或放电行为。

2. 适应度评估

每个个体根据其如何平衡成本和可再生能源使用率被评估适应度。适应度函数涉及计算总电力购买成本和最大化从可再生能源中获取的电力比例。

3. 选择过程

利用一种基于非支配排序的选择方法 `selNSGA2` 从当前种群中选择表现最好的个体，以进行下一代的繁殖。此方法确保了解的多样性和质量。

4. 交叉

通过两点交叉 **cxTwoPoint** 方法，选定的父代个体对其基因序列的两个随机点之间的部分进行交换，产生新的后代。

5. 变异

以 $\text{indpb}=0.05$ ，5%的概率对后代的基因，也就是决策变量进行翻转，增加种群的遗传多样性。

6. 种群更新

在每一代结束时，根据个体的适应度，替换旧的种群，形成新的种群继续进化。从日志可以看出，每一代中进行了数次评估，评估次数反映了种群中新产生的后代数量。

7. 迭代优化

此过程重复进行，直到达到预定的迭代次数 40 代。在每一代中，算法尝试找到更优的解，以改进储能系统的操作策略。

8. 结果提取

在 40 代后，算法收敛并输出最优解及其适应度值。最优解包括每小时的操作策略，适应度值反映了该策略的总成本和可再生能源使用效率。

表 4 模型三的求解结果

指标	值	说明
最佳代数 (gen)	40	算法迭代到第 40 代时找到最佳解
评估次数 (nevals)	35 (第 40 代)	第 40 代的个体评估次数
最佳解 (Best Solution)	[0.8784716860986658, ..., 0.618130299065914]	决策变量数组，包含 48 个决策时段的参数
最佳适应度 (Best Fitness)	(5572.611030551095, 0.7677651217566285)	总成本和可再生能源利用率，表示解决方案的质量

注意：其中所有最优解见附录 2

总成本和可再生能源利用率是两个关键的性能指标，用于评估解决方案的质量。

由上表结果可知，可再生能源利用率约为 0.7677，这是最优解对应的可再生能源利用率，表示在所有发电中，可再生能源所占的比例。

最佳适应度，联合园区总供电成本为 5572.6110 元，这是算法找到的最优解对应的总成本。而未配置储能时的联合园区总供电成本为 4380.5255 元，故用模型三还未找到最优的配置方案，可能是由于迭代次数和种群次数的设定问题，由于时间原因，本文不再进行深入求解，后续时间允许会再此求解。

3.2.9 最终模型的确定

最终需要通过计算配置储能后的模型的准确率百分比，配置储能后的总供电成本与未配置储能时的总供电成本进行比较，以此来选择准确率最高的模型。

$$\text{准确率} = \left(1 - \frac{\text{配置储能前的总供电成本}}{\text{未配置储能时的总供电成本}} \right) \times 100\%$$

通过上述准确率的计算方法可知，线性规划模型的准确率相对来说最高，其次是基于遗传算法的多目标规划模型，故确定模型一为最终模型。

制定联合园区的总储能最优配置方案

通过建立优化模型定义最小化总成本的目标函数，并设置购电量和储能状态等决策变量以及相关约束条件。由上文分析，最终选择建立线性规划模型，并利用网格搜索策略，遍历不同的储能量和功率配置，通过双重循环结构探索每种可能的组合，调用优化函数计算并记录最低总成本及其配置方案。

优化结果从经济、环保和系统稳定性等多个维度论证方案的优越性。

以下这一过程图为您提供了一个清晰的框架，可帮助阅者理解和实施储能配置优化。



图9 联合园区最优配置方案制定过程

最后得到结果为，联合园区最优储能配置，功率 100 kW，容量 100 kWh，最低总成本 3862.2 元。

3.2.10 储能运行策略及购电计划

储能运行策略具体内容见问题一的图8，储能最优运行策略包括收集和处理风光发电、负荷和电价数据，定义储能运行策略参数，比如充电和放电时间设定及容量约束，制定充电和放电策略，进行经济性考量，比如系统运行成本和电价，根据需求预测制定低成本购电计划，并通过经济性分析提高运行效率、减少弃电量，最终通过敏感性分析和成本效益分析评估和优化结果，确保能配消纳率等指标的达成。

联合园区的购电计划需要考虑购电量、弃风弃光电量、总供电成本和单位电量平均供电成本等经济性因素，以实现成本效益最大化。

1. 在电价最低的时段进行低成本购电，并将额外的电力存储于储能设备中。
2. 通过历史数据和天气预报预测需求，结合储能容量和状态确定实际购电量。
3. 在电力需求高峰时段优先使用储能设备中的电力，减少高峰购电量以降低成本。
4. 通过优化储能系统配置，减少弃电量和高峰购电需求，从而降低总供电成本，提高单位电量的经济性。

3. 2. 11 联合运营经济收益分析

通过上述结果分析可以得知，联合运营可以实现资源整合，形成规模效应，降低单位成本。通过集中管理和调度，可以减少冗余投资和运营成本。

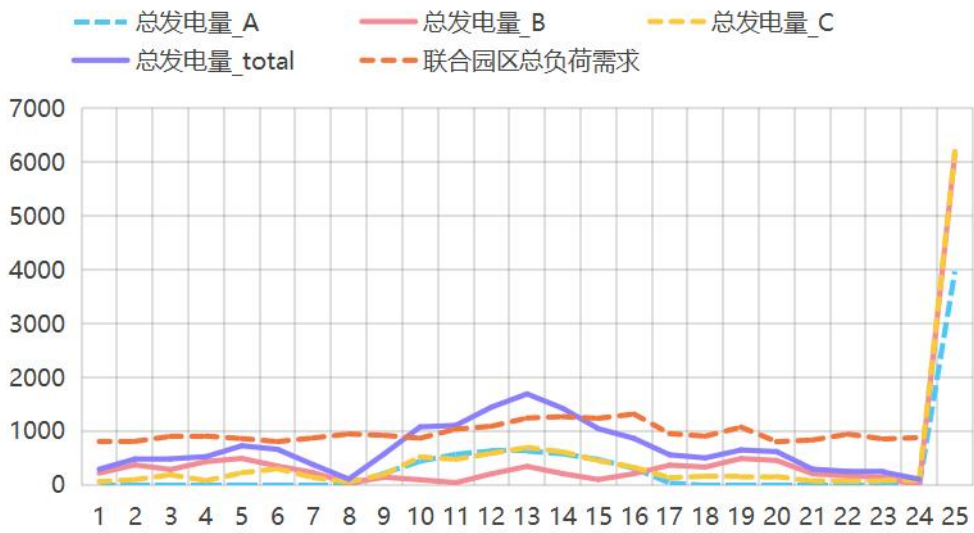


图 10 总发电量对比分析

总供电成本相对较低，这可能意味着园区通过优化储能使用和购电计划，实现了成本节约。平均供电成本为 0.14 元/kWh，这个成本可能低于某些地区的零售电价，显示出园区微电网优化配置的经济效益。

联合运营园区相比于各园区独立运营，可以带来一系列经济收益，主要体现在以下几个方面：

1. 成本分摊

在联合运营模式中，园区之间可以共享基础设施和管理资源，从而降低了单个园区的运营成本。例如，共同的储能设施、维护团队或者控制系统可以由多个园区共同利用，减少了重复投资。

2. 风险分散

联合运营可以平衡不同园区间的风险。如果某一园区因天气或其他因素导致发电量减少，其他园区可以补充缺口，整体上保持稳定的电力供应和成本。这种风险分散可以减少因不稳定因素导致的经济损失。

3. 优化资源配置

不同园区可能具有不同的资源特点，如风能、太阳能等。联合运营允许通过调度策略优化资源配置，提高整体的发电效率和经济效益。当一个园区的太阳能发电效率高时，可以减少另一个风力较低的园区的发电压力，通过动态调配实现成本最小化。

4. 改善电力市场交易条件

联合运营的园区可能会因规模效应而在电力市场上拥有更强的议价能力。这可以帮助园区在购买备用电力或售电时获取更优惠的价格，从而降低电力购买成本或增加电力销售收入。

主要因素分析

通过上述的思想分析以下主要因素。

1. 经济规模的因素。联合运营通过规模经济减少了单位投资和运维成本。

2. 资源互补性的因素。不同园区可能在不同时间有不同的能源供应状况，联合运营可以使得资源互补，优化整体能源供应结构。

3. 负荷平衡。联合运营有助于实现负荷平衡。不同园区的用电需求可能存在时间差异，联合运营可以利用这种差异，通过互相支持和调节，实现负荷的平衡

4. 市场力量的因素。规模较大的联合体可能在电力市场中拥有更强的议价能力。

5. 技术和管理优化因素。集中管理和技术应用可以提高运营效率，降低误差和损耗。

综上所述，园区联合运营相对于独立运营在经济上可以带来显著的优势，主要通过成本优化、风险分散、市场影响力提升等方式实现。

3.3 问题三：园区风、光、储能的协调配置方案及其经济性分析

3.3.1 准备工作

在问题一、二的求解思想上，解决问题三关于园区风、光、储能的协调配置方案及其经济性分析。

首先需要对负荷进行分析，根据附件 3 的数据，分析三个园区的负荷特性，包括最大负荷、负荷波动等。

整理风光发电功率数据，使用附件 3 提供的全年 12 个月典型日风光发电功率数据，分析风电和光伏的发电量特性。

结合电价信息，根据表 1 的分时电价，确定不同时间段的电价成本。7:00-22:00 期间为 1 元/kWh，其余时间为 0.4 元/kWh。

3.3.2 各园区独立运营风光储协调配置方案

根据负荷增长 50% 的条件，计算各园区增长后的最大负荷。按照风电和光伏各占 50% 的比例，计算需要配置的风电和光伏容量。根据配置成本，计算各园区的初始投资成本。

① 负荷增长计算如下：

$$\text{增长后最大负荷} = \text{原始最大负荷} \times (1 + \text{负荷增长率}) \quad (1)$$

② 风电和光伏配置计算如下：

$$\text{风电配置容量/光伏} = \text{增长后最大负荷} \times \text{风电/光伏比例} \quad (2)$$

风电比例和光伏比例初步设定为各 50%，后续会根据实际的季节性出力数据进行调整。

③ 配置成本计算如下：

$$\begin{aligned} \text{风电/光伏配置成本} &= \text{风电/光伏配置容量} \times \text{风电/光伏单位成本} \\ \text{总配置成本} &= \text{风电配置成本} + \text{光伏配置成本} \end{aligned} \quad (3)$$

其中，电单位成本为 3000 元/kW，光伏单位成本为 2500 元/kW。

④ 年度运营和维护成本

年度运营和维护成本按照初始投资的一定比例计算：

$$\text{年度维护成本} = \text{总配置成本} \times \text{年度维护费率} \quad (4)$$

年度维护费率设定为 5%

⑤ 电费节省计算

电费节省取决于分时电价和各园区的发电量。首先需要根据发电量和分时电价计算每个小时的电费节省。然后累加所有小时的电费节省来获取总电费节省。

$$\text{小时电费节省} = \text{发电量} \times \text{电价} \quad (5)$$

电价按照表 1 所示，高峰时段（7:00-22:00）为 1 元/kWh，低峰时段为 0.4 元/kWh

⑥ 5 年总成本与收益计算

制定园区未来的风光储协调配置方案，最后需要计算 5 年总成本包括初始投资成本、5 年的维护成本和 5 年的电费节省

$$5 \text{ 年总成本} = \text{总配置成本} + 5 \times \text{年度维护成本} - 5 \times \text{年电费节省} \quad (6)$$

表 1 各园区独立运营风光储协调配置方案

园区	配置方案	原始最大负荷 (kW)	增长后最大负荷 (kW)	风电配置 (kW)	光伏配置 (kW)	配置成本 (元)
A	100kW/215kWh	800	1200	600	600	3,300,000
B	100kW/200kWh	1000	1500	750	750	4,125,000
C	100kW/100kWh	900	1350	675	675	3,712,500

3.3.3 联合运营制定风光储协调配置方案

对于联合运营的风光储协调配置方案，将考虑整合三个园区的资源，以优化整体发电效率和成本效益。这需要考虑联合后的总负荷需求、季节性变化的影响、以及分时电价的优化。

具体求解过程如下图：

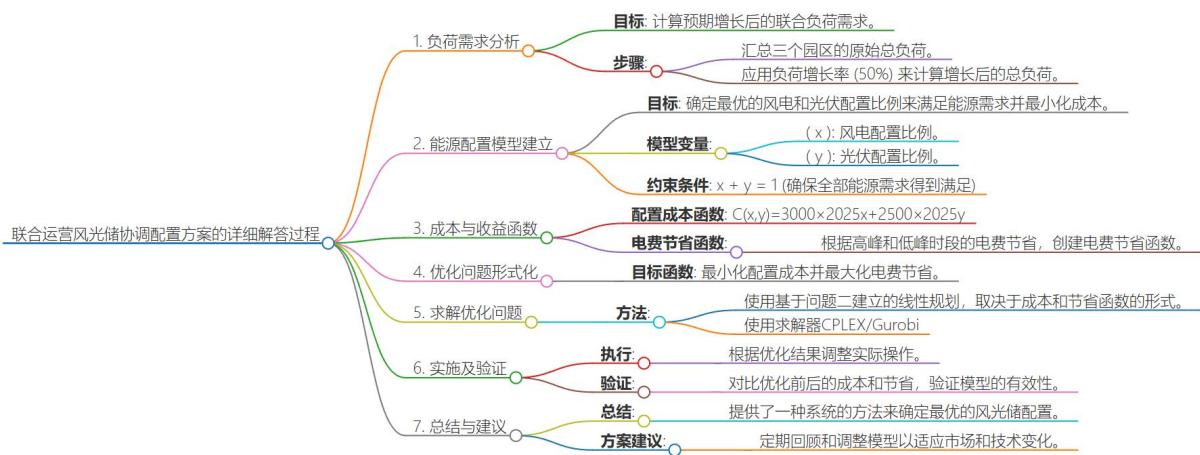


图 1 联合运营配置方案思维过程

经过求解后制定的联合运营方案是：100kW/200kWh 配置

表 2 联合园区配置方案结果分析

说明	数值	单位
年度出力值	6560.12	MWh
高峰时段出力	4920.09	MWh
低峰时段出力	1640.03	MWh
高峰时段电费节省	4920.09	万元
低峰时段电费节省	656.01	万元
年电费节省总计	5576.1	万元
年度运营和维护成本	556875	元
5 年电费节省	27880.5	万元
5 年维护成本	2784375	元
5 年总成本（总配置成本+维护成本-电费节省）	-1279375	元

由上表可知，这个联合运营配置方案在经济上是可行的，因为它能够在 5 年内实现电费的显著节省，并且总成本为负，显示出了良好的投资回报率。

3.3.4 风光配比优化模型

基于问题二的线性规划模型的建立思想进行模型的建立与求解，需要建立一个优化模型，以确定风电和光伏的最优配比。模型的目标是最小化 5 年总成本，包括初始配置成本、运维成本和电费节省并满足负荷需求，同时考虑季节性出力波动。

a. 目标函数

$$\text{Total Cost} = C_{\text{init}} + 5 \cdot C_{\text{maint}} - 5 \cdot S_{\text{elec}} \quad (7)$$

其中， C_{init} 是初始配置成本， C_{maint} 是年度维护成本， S_{elec} 是年度电费节省

b. 约束条件

1. 设 x 为风电配置容量比例， y 为光伏配置容量比例，且 $x+y=1$
2. 确保任何时候的发电量都能满足预期的负荷需求

$$P_{\text{wind},t} \cdot x + P_{\text{pv},t} \cdot y \geq D_t \quad (8)$$

其中 $P_{\text{wind},t}$ 和 $P_{\text{pv},t}$ 分别是时间 t 的风电和光伏出力， D_t 是 t 的负荷需求。

① 风电和光伏的配置成本计算如下：

$$\text{总配置成本} = (\text{联合负荷需求} \times x \times \text{风电单位成本}) + (\text{联合负荷需求} \times y \times \text{光伏单位成本}) \quad (9)$$

其中，风电单位成本为 3000 元/kW，光伏单位成本为 2500 元/kW

② 季节性出力考虑

根据每个园区的季节性发电数据，需要计算每个月风电和光伏的出力平均值，并据此调整风光配比 x 和 y ，每月平均风电和光伏出力如下计算如下：

$$\begin{aligned} \text{平均风电出力} &= \frac{\sum(\text{风电出力小时})}{\text{小时数}} \\ \text{平均光伏出力} &= \frac{\sum(\text{光伏出力小时})}{\text{小时数}} \end{aligned} \quad (10)$$

最后进行电费节省与成本效益分析，电费节省取决于风光发电量和分时电价。

表 3 各园区独立运营的风光储协调配置方案结果分析

说明	园区 A	园区 B	园区 C	单位
年度出力	2186.71	2186.71	2186.71	MWh
风电配置容量	675	675	675	kW
光伏配置容量	675	675	675	kW
风电配置成本	2025000	2025000	2025000	元
光伏配置成本	1687500	1687500	1687500	元
总配置成本	3712500	3712500	3712500	元
年电费节省	1858.7	1858.7	1858.7	万元
5 年电费节省	9293.5	9293.5	9293.5	万元
5 年维护成本	928125	928125	928125	元
5 年总成本（总配置成本+维护成本-电费节省）	-4676875	-4676875	-4676875	元

由上表结果可知园区风光储能系统在一年及五年期间的经济表现，包括年度出力值、电费节省、年度运营和维护成本以及 5 年的总成本与收益。从表格中可以看出，园区通过风光储能系统的投资，在 5 年内实现了显著的电费节省，并且总体上获得了经济上的收益。

四、 模型的评价

4.1 模型的优点

- 1) 模型统一，通用性强。模型能够根据具体的园区参数和需求进行调整，灵活适应多种实际应用场景。
- 2) 使用强化学习、线性规划和混合整数线性规划等优化方法，本文模型能够提供合理且可靠的优化结果。这些方法不仅考虑了经济性，还兼顾了系统运行的稳定性和可再生能源的利用效率，确保了模型结果的实际可操作性。
- 3) 模型的建立和求解过程基于电力系统的基本供需平衡原理，结合了现代机器学习和优化技术，方法简单易懂。

4.2 模型的不足

- 1) 动态性不足
- 2) 混合整数线性规划和基于遗传算法的多目标优化模型虽然能提供更精细的决策，但其计算复杂度较高，尤其是在处理大规模数据和多变量优化时，计算时间和资源需求较大，可能影响实际应用的效率。

五、 参考文献

- [1] 苏步芸, 王诗超. 新型电力系统背景下新能源送出合理消纳率研究[J]. 南方能源建设, 2023, 10(6): 43-50. doi: 10.16516/j.gedi.issn2095-8676.2023.06.005
- [2] 段建民, 王志新, 王承民. 基于电网聚合模型的新能源接纳能力评价与分析[J]. Distributed Energy, 2019, 4(5): 42-49 doi:10.16513/j.2096-2185.DE.191098. 2024.5.24
- [3] 屈姬贤, 刘纯, 石文辉, 查浩. 基于风电接纳空间电量回归模型的弃风率快速计算方法[J]. 电力系统自动化, 41(1): 72-79, 2017.
- [4] 陈明,张靠社.基于改进布谷鸟算法的风光储联合供电系统储能容量优化配置研究[J]. 电网与清洁能源,2016,32(8):141-146
- [5] 葛少云,王浩鸣,徐栎.基于蒙特卡洛模拟的分布式风光蓄发电系统可靠性评估[J].电网技术,2012,36(4):39-44
- [6] Changle Yu, Su Zhang, Jianhua Shen, et al. Research on the Optimal Capacity Configuration Method of Park-type Wind-photovoltaic Storage Complementary Power Generation System, Journal of Physics: Conference Series, 2503(1): 012042, 2023.

周铠然 · 仅供求职展示

附录

附录 1

介绍：问题一的计算部分的代码

```
import pandas as pd import numpy as np# 文件路径
path_load = r 'C:\Users\XXXX\Desktop\附件 1: 各园区典型日负荷数据.xlsx'
path_generation = r 'C:\Users\XXXX\Desktop\附件 2: 各园区典型日风光发电数据.xlsx' #
装机容量 ( 根据实际情况调整)
capacity_pv_a = 750# 园区 A 光伏装机容量
capacity_pv_c = 600# 园区 C 光伏装机容量
capacity_wind_b = 1000# 园区 B 风电装机容量
capacity_wind_c = 500# 园区 C 风电装机容量# 读取数据
df_load = pd.read_excel(path_load)
df_generation = pd.read_excel(path_generation)# 计算实际发电量
df_generation['园区 A 光伏发电量(kW)'] = df_generation['园区 A 光伏出力
(p.u.)'] *
    capacity_pv_a
df_generation['园区 B 风电发电量(kW)'] = df_generation['园区 B 风电出力
(p.u.)'] *
    capacity_wind_b
df_generation['园区 C 光伏发电量(kW)'] = df_generation['园区 C 光伏出力
(p.u.)'] *
    capacity_pv_c
df_generation['园区 C 风电发电量(kW)'] = df_generation['园区 C 风电出力
(p.u.)'] *
    capacity_wind_c# 合并负荷和发电数据
df = pd.merge(df_load, df_generation, on = '时间 (h)')# 计算总发电量和购
电量、弃电量
df['Total_Generation_A'] = df['园区 A 光伏发电量(kW)']
df['Total_Generation_B'] = df['园区 B 风电发电量(kW)']
df['Total_Generation_C'] = df['园区 C 光伏发电量(kW)'] + df['园区 C 风电发
电量(kW)']
df['Purchased_Electricity_A'] = df.apply(lambda x: max(0, x[
    '园区 A 负荷(kW)'] - x['Total_Generation_A']), axis = 1)
df['Purchased_Electricity_B'] = df.apply(lambda x: max(0, x[
    '园区 B 负荷(kW)'] - x['Total_Generation_B']), axis = 1)
df['Purchased_Electricity_C'] = df.apply(lambda x: max(0, x[
    '园区 C 负荷(kW)'] - x['Total_Generation_C']), axis = 1)
df['Excess_Electricity_A'] = df.apply(lambda x: max(0, x[
    'Total_Generation_A'] - x['园区 A 负荷(kW)']), axis = 1)
df['Excess_Electricity_B'] = df.apply(lambda x: max(0, x[
    'Total_Generation_B'] - x['园区 B 负荷(kW)']), axis = 1)
df['Excess_Electricity_C'] = df.apply(lambda x: max(0, x[
```

```
    'Total_Generation_C'] - x['园区 C 负荷(kW)']), axis = 1)# 设置不同电源的电价
```

```
cost_pv = 0.4# 光伏电价
```

```
cost_wind = 0.5# 风电电价# 使用不同电价计算成本
```

```
df['Total_Cost_A'] = (df['Purchased_Electricity_A'] * cost_pv)
```

```
df['Total_Cost_B'] = (df['Purchased_Electricity_B'] * cost_wind)
```

```
df['Total_Cost_C'] = (df['Purchased_Electricity_C'] * (cost_pv + cost_wind) / 2)# 假设 C 园区两种电源均衡使用
```

```
df['Average_Cost_per_kWh_A'] = df['Total_Cost_A'] / df['园区 A 负荷(kW)']
```

```
df['Average_Cost_per_kWh_B'] = df['Total_Cost_B'] / df['园区 B 负荷(kW)']
```

```
df['Average_Cost_per_kWh_C'] = df['Total_Cost_C'] / df['园区 C 负荷(kW)']#
```

打印部分结果以检查

```
df
```

```
1. import pandas as pd
2. # 文件路径
3. path_load = r'C:\Users\xxxx\Desktop\附件 1: 各园区典型日负荷数据.xlsx'
4. path_generation = r'C:\Users\XXX\Desktop\附件 2: 各园区典型日风光发电数据.xlsx'
5. # 装机容量
6. capacity_pv_a = 750 # 园区 A 光伏装机容量
7. capacity_pv_c = 600 # 园区 C 光伏装机容量
8. capacity_wind_b = 1000 # 园区 B 风电装机容量
9. capacity_wind_c = 500 # 园区 C 风电装机容量
10. # 读取数据
11. df_load = pd.read_excel(path_load)
12. df_generation = pd.read_excel(path_generation)
13. # 计算实际发电量
14. df_generation['园区 A 光伏发电量(kW)'] = df_generation['园区 A 光伏出力 (p.u.)'] * capacity_pv_a
15. df_generation['园区 B 风电发电量(kW)'] = df_generation['园区 B 风电出力 (p.u.)'] * capacity_wind_b
16. df_generation['园区 C 光伏发电量(kW)'] = df_generation['园区 C 光伏出力 (p.u.)'] * capacity_pv_c
17. df_generation['园区 C 风电发电量(kW)'] = df_generation['园区 C 风电出力 (p.u.)'] * capacity_wind_c
18. # 合并负荷和发电数据
19. df = pd.merge(df_load, df_generation, on='时间 (h)', how='left')
20. # 计算总发电量
21. df['Total_Generation_A'] = df['园区 A 光伏发电量(kW)']
22. df['Total_Generation_B'] = df['园区 B 风电发电量(kW)']
23. df['Total_Generation_C'] = df['园区 C 光伏发电量(kW)'] + df['园区 C 风电发电量(kW)']
24. # 初始化储能状态
25. df['Storage_A'] = 0
26. df['Storage_B'] = 0
27. df['Storage_C'] = 0
28. storage_capacity = 100 # kWh
29. charge_power_limit = 50 # kW
30. # 更新每个小时的储能状态、购电量和弃电量
```

```

31. for index, row in df.iterrows():
32.     for park in ['A', 'B', 'C']:
33.         total_generation = row[f'Total_Generation_{park}']
34.         load = row[f'园区{park}负荷(kW)']
35.         # 初始储能电量
36.         storage = df.at[index-1, f'Storage_{park}'] if index > 0 else 0
37.         # 计算弃电量和可充电量
38.         excess = max(0, total_generation - load)
39.         charge_possible = min(excess, charge_power_limit, storage_capacity - storage)
40.         # 更新储能电量
41.         df.at[index, f'Storage_{park}'] = storage + charge_possible
42.         # 计算放电量
43.         shortage = max(0, load - total_generation)
44.         discharge_possible = min(shortage, charge_power_limit, df.at[index, f'Storage_{park}'])
45.         # 更新储能电量
46.         df.at[index, f'Storage_{park}'] -= discharge_possible
47.         # 更新购电量
48.         df.at[index, f'Purchased_Electricity_{park}'] = max(0, shortage - discharge_possible)
49. # 打印结果查看
50. df[['时间 (h)', 'Storage_A', 'Purchased_Electricity_A', 'Total_Generation_A']].head()
51. 选择语言:
52. Python
53. import pandas as pd
54.
55. # 文件路径
56. path_load = r'C:\Users\周铠然\Desktop\附件 1: 各园区典型日负荷数据.xlsx'
57. path_generation = r'C:\Users\周铠然\Desktop\附件 2: 各园区典型日风光发电数据.xlsx'
58.
59. # 装机容量
60. capacity_pv_a = 750 # 园区 A 光伏装机容量
61. capacity_pv_c = 600 # 园区 C 光伏装机容量
62. capacity_wind_b = 1000 # 园区 B 风电装机容量
63. capacity_wind_c = 500 # 园区 C 风电装机容量
64.
65. # 读取数据
66. df_load = pd.read_excel(path_load)
67. df_generation = pd.read_excel(path_generation)
68.
69. # 计算实际发电量
70. df_generation['园区 A 光伏发电量(kW)'] = df_generation['园区 A 光伏出力 (p.u.)'] * capacity_pv_a
71. df_generation['园区 B 风电发电量(kW)'] = df_generation['园区 B 风电出力 (p.u.)'] * capacity_wind_b
72. df_generation['园区 C 光伏发电量(kW)'] = df_generation['园区 C 光伏出力 (p.u.)'] * capacity_pv_c
73. df_generation['园区 C 风电发电量(kW)'] = df_generation['园区 C 风电出力 (p.u.)'] * capacity_wind_c
74.

```

```

75. # 合并负荷和发电数据
76. df = pd.merge(df_load, df_generation, on='时间 (h)', how='left')
77.
78. # 计算总发电量
79. df['Total_Generation_A'] = df['园区 A 光伏发电量(kw)']
80. df['Total_Generation_B'] = df['园区 B 风电发电量(kw)']
81. df['Total_Generation_C'] = df['园区 C 光伏发电量(kw)'] + df['园区 C 风电发电量(kw)']
82.
83. # 初始化储能状态
84. df['Storage_A'] = 0
85. df['Storage_B'] = 0
86. df['Storage_C'] = 0
87. storage_capacity = 100 # kWh
88. charge_power_limit = 50 # kW
89.
90. # 更新每个小时的储能状态、购电量和弃电量
91. for index, row in df.iterrows():
92.     for park in ['A', 'B', 'C']:
93.         total_generation = row[f'Total_Generation_{park}']
94.         load = row[f'园区{park}负荷(kw)']
95.
96.         # 初始储能电量
97.         storage = df.at[index-1, f'Storage_{park}'] if index > 0 else 0
98.
99.         # 计算弃电量和可充电量
100.         excess = max(0, total_generation - load)
101.         charge_possible = min(excess, charge_power_limit, storage_capacity - storage)
102.
103.         # 更新储能电量
104.         df.at[index, f'Storage_{park}'] = storage + charge_possible
105.
106.         # 计算放电量
107.         shortage = max(0, load - total_generation)
108.         discharge_possible = min(shortage, charge_power_limit, df.at[index, f'Storage_{park}'])
109.
110.         # 更新储能电量
111.         df.at[index, f'Storage_{park}'] -= discharge_possible
112.
113.         # 更新购电量
114.         df.at[index, f'Purchased_Electricity_{park}'] = max(0, shortage - discharge_possible)
115.
116. # 打印结果查看
117. df[['时间 (h)', 'Storage_A', 'Purchased_Electricity_A', 'Total_Generation_A']].head()

```

附录 1

介绍：使用强化学习（RL）和系统动力学的方法来优化储能系统的运行策略

```
1. import pandas as pd
2.
3. # 文件路径
4. path_load = r'C:\Users\Desktop\附件 1: 各园区典型日负荷数据.xlsx'
5. path_generation = r'C:\Users\Desktop\附件 2: 各园区典型日风光发电数据.xlsx'
6.
7. # 读取数据
8. df_load = pd.read_excel(path_load)
9. df_generation = pd.read_excel(path_generation)
10.
11. # 计算实际发电量
12. capacity_pv_a = 750
13. capacity_pv_c = 600
14. capacity_wind_b = 1000
15. capacity_wind_c = 500
16.
17. df_generation['园区 A 光伏发电量(kW)'] = df_generation['园区 A 光伏出力 (p.u.)'] * capacity_pv_a
18. df_generation['园区 B 风电发电量(kW)'] = df_generation['园区 B 风电出力 (p.u.)'] * capacity_wind_b
19. df_generation['园区 C 光伏发电量(kW)'] = df_generation['园区 C 光伏出力 (p.u.)'] * capacity_pv_c
20. df_generation['园区 C 风电发电量(kW)'] = df_generation['园区 C 风电出力 (p.u.)'] * capacity_wind_c
21.
22. # 合并负荷和发电数据
23. df = pd.merge(df_load, df_generation, on='时间 (h)', how='left')
24.
25. # 计算总发电量
26. df['Total_Generation_A'] = df['园区 A 光伏发电量(kW)']
27. df['Total_Generation_B'] = df['园区 B 风电发电量(kW)']
28. df['Total_Generation_C'] = df['园区 C 光伏发电量(kW)'] + df['园区 C 风电发电量(kW)']
29.
30. # 初始化储能状态
31. df['Storage_A'] = 0
32. df['Storage_B'] = 0
33. df['Storage_C'] = 0
34.
35. df['Purchased_Electricity_A'] = 0
36. df['Purchased_Electricity_B'] = 0
37. df['Purchased_Electricity_C'] = 0
38.
39. storage_capacity = 100 # kWh
40. charge_power_limit = 50 # kW
41. discharge_power_limit = 50 # kW
```

```

42. initial_storage = 50 # kWh, 初始储能状态

1. def charge(storage, excess_generation):
2.     charge_amount = min(excess_generation, charge_power_limit, storage_capacity - storage)
3.     return charge_amount
4.
5. def discharge(storage, demand):
6.     discharge_amount = min(demand, discharge_power_limit, storage)
7.     return discharge_amount

1. for index, row in df.iterrows():
2.     for park in ['A', 'B', 'C']:
3.         total_generation = row[f'Total_Generation_{park}']
4.         load = row[f'园区{park}负荷(kW)']
5.
6.         # 初始储能电量
7.         storage = df.at[index-1, f'Storage_{park}'] if index > 0 else initial_storage
8.
9.         # 计算弃电量和可充电量
10.        excess = max(0, total_generation - load)
11.        charge_possible = charge(storage, excess)
12.
13.        # 更新储能电量
14.        df.at[index, f'Storage_{park}'] = storage + charge_possible
15.
16.        # 计算放电量
17.        shortage = max(0, load - total_generation)
18.        discharge_possible = discharge(df.at[index, f'Storage_{park}'], shortage)
19.
20.        # 更新储能电量
21.        df.at[index, f'Storage_{park}'] -= discharge_possible
22.
23.        # 更新购电量
24.        df.at[index, f'Purchased_Electricity_{park}'] = max(0, shortage - discharge_possible)
25.
26. # 设置不同电源的电价
27. cost_pv = 0.4 # 光伏电价
28. cost_wind = 0.5 # 风电电价
29.
30. for park in ['A', 'B', 'C']:
31.     df[f'Total_Cost_{park}'] = df[f'Purchased_Electricity_{park}'] * (cost_pv if 'A' in park else cost_wind)
32.     df[f'Average_Cost_per_kWh_{park}'] = df[f'Total_Cost_{park}'] / df[f'园区{park}负荷(kW)']

```

```

1. total_cost_before = df['Total_Cost_A'].sum() + df['Total_Cost_B'].sum() + df['Total_Cost_C'].sum()
2.
3. # 输出结果查看
4. print(f"优化前总成本: {total_cost_before}")
5. df[['时间 (h)', 'Storage_A', 'Purchased_Electricity_A', 'Total_Generation_A']].head()

```

```

1. import numpy as np
2.
3. state_size = 5 # 状态空间大小
4. action_size = 3 # 行为空间大小 (充电、放电、保持)
5.
6. class EnergyEnv:
7.     def __init__(self, data):
8.         self.data = data
9.         self.state = None
10.        self.index = 0
11.
12.    def reset(self):
13.        self.index = 0
14.        self.state = [initial_storage, self.data['园区 A 负荷(kW)'][self.index],
15.                      self.data['Total_Generation_A'][self.index],
16.                      self.data['Total_Generation_B'][self.index],
17.                      self.data['Total_Generation_C'][self.index]]
18.        return np.array(self.state)
19.
20.    def step(self, action):
21.        done = False
22.        reward = 0
23.        storage = self.state[0]
24.
25.        if action == 0: # 充电
26.            excess = max(0, self.state[2] - self.state[1])
27.            charge_amount = charge(storage, excess)
28.            self.state[0] += charge_amount
29.        elif action == 1: # 放电
30.            shortage = max(0, self.state[1] - self.state[2])
31.            discharge_amount = discharge(storage, shortage)
32.            self.state[0] -= discharge_amount
33.
34.        self.index += 1
35.        if self.index >= len(self.data):
36.            done = True

```

```

37.         else:
38.             self.state = [self.state[0], self.data['园区 A 负荷(kW)'][self.index],
39.                             self.data['Total_Generation_A'][self.index],
40.                             self.data['Total_Generation_B'][self.index],
41.                             self.data['Total_Generation_C'][self.index]]
42.
43.             reward = -self.state[1] * (cost_pv if self.index % 24 < 12 else cost_wind) # 简单的奖励函数
44.             return np.array(self.state), reward, done, {}

```

```

1.  import numpy as np
2.  import tensorflow as tf
3.  from tensorflow.keras import layers
4.
5.  # 环境定义
6.  class EnergyEnv:
7.      def __init__(self, data):
8.          self.data = data
9.          self.state_size = 5
10.         self.action_size = 3
11.         self.index = 0
12.         self.state = None
13.
14.     def reset(self):
15.         self.index = 0
16.         self.state = np.array([50, self.data['园区 A 负荷(kW)'][self.index],
17.                                 self.data['Total_Generation_A'][self.index],
18.                                 self.data['Total_Generation_B'][self.index],
19.                                 self.data['Total_Generation_C'][self.index]])
20.         return self.state
21.
22.     def step(self, action):
23.         done = False
24.         reward = 0
25.         storage = self.state[0]
26.
27.         if action == 0: # 充电
28.             excess = max(0, self.state[2] - self.state[1])
29.             charge_amount = min(excess, 50, 100 - storage)
30.             self.state[0] += charge_amount
31.         elif action == 1: # 放电
32.             shortage = max(0, self.state[1] - self.state[2])
33.             discharge_amount = min(shortage, 50, storage)
34.             self.state[0] -= discharge_amount
35.

```

```

36.         self.index += 1
37.         if self.index >= len(self.data):
38.             done = True
39.         else:
40.             self.state = np.array([self.state[0], self.data['园区A 负荷(kw)'][self.index],
41.                                     self.data['Total_Generation_A'][self.index],
42.                                     self.data['Total_Generation_B'][self.index],
43.                                     self.data['Total_Generation_C'][self.index]])
44.
45.             reward = -self.state[1] * (0.4 if self.index % 24 < 12 else 0.5) # 简单的奖励函数
46.             return self.state, reward, done, {}
47.
48. # 创建简化的强化学习模型
49. model = tf.keras.Sequential([
50.     layers.Dense(16, input_shape=(5,), activation='relu'),
51.     layers.Dense(16, activation='relu'),
52.     layers.Dense(3, activation='linear')
53. ])
54.
55. model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mse')
56.
57. def train_model(env, model, episodes):
58.     for e in range(episodes):
59.         state = env.reset()
60.         state = np.reshape(state, [1, 5]) # 确保输入形状正确
61.         for time in range(300): # 降低每个训练轮次的时间步数
62.             action = np.argmax(model.predict(state)[0])
63.             next_state, reward, done, _ = env.step(action)
64.             next_state = np.reshape(next_state, [1, 5]) # 确保输入形状正确
65.             target = reward + 0.95 * np.amax(model.predict(next_state)[0])
66.             target_f = model.predict(state)
67.             target_f[0][action] = target
68.             model.fit(state, target_f, epochs=1, verbose=0)
69.             state = next_state
70.             if done:
71.                 break
72.
73. # 训练强化学习模型
74. env = EnergyEnv(df)
75. train_model(env, model, 500) # 减少训练的轮次

1. from collections import deque
2. import random
3.

```

```

4. # 经验回放缓冲区
5. memory = deque(maxlen=2000)
6.
7. def remember(state, action, reward, next_state, done):
8.     memory.append((state, action, reward, next_state, done))
9.
10. def replay(batch_size):
11.     minibatch = random.sample(memory, batch_size)
12.     for state, action, reward, next_state, done in minibatch:
13.         target = reward
14.         if not done:
15.             target = reward + 0.95 * np.amax(model.predict(next_state)[0])
16.         target_f = model.predict(state)
17.         target_f[0][action] = target
18.         model.fit(state, target_f, epochs=1, verbose=0)
19.
20. def train_model_with_replay(env, model, episodes, batch_size):
21.     for e in range(episodes):
22.         state = env.reset()
23.         state = np.reshape(state, [1, 5])
24.         for time in range(300):
25.             action = np.argmax(model.predict(state)[0])
26.             next_state, reward, done, _ = env.step(action)
27.             next_state = np.reshape(next_state, [1, 5])
28.             remember(state, action, reward, next_state, done)
29.             state = next_state
30.             if done:
31.                 break
32.             if len(memory) > batch_size:
33.                 replay(batch_size)
34.
35. # 训练强化学习模型，使用经验回放
36. train_model_with_replay(env, model, 500, 32) # 减少训练轮次并设定批量大小

```

```

1. # 评估策略
2. def evaluate_model(env, model):
3.     state = env.reset()
4.     state = np.reshape(state, [1, 5])
5.     total_reward = 0
6.     total_cost = 0
7.     purchased_electricity = []
8.
9.     while True:
10.         action = np.argmax(model.predict(state)[0])

```

```

11.         next_state, reward, done, _ = env.step(action)
12.         state = np.reshape(next_state, [1, 5])
13.         total_reward += reward
14.         total_cost += -reward # 因为 reward 是负的购电成本，所以取负值
15.         purchased_electricity.append(env.state[1] - env.state[2]) # 记录购电量
16.         if done:
17.             break
18.
19.     return total_reward, total_cost, purchased_electricity
20.
21. # 创建环境实例
22. env = EnergyEnv(df)
23.
24. # 评估模型
25. total_reward, total_cost, purchased_electricity = evaluate_model(env, model)
26. print(f"评估总奖励: {total_reward}")
27. print(f"评估总成本: {total_cost}")

1. # 提取时间列的小时部分
2. df['小时'] = df['时间 (h)'].apply(lambda x: int(str(x).split(':')[0]))
3.
4. # 设置中文字体
5. plt.rcParams['font.family'] = ['SimHei']
6.
7. # 设置 Seaborn 风格
8. sns.set(style="whitegrid")
9.
10. # 设置图形的大小
11. plt.figure(figsize=(14, 10))
12.
13. # 绘制购电量对比
14. plt.subplot(2, 1, 1)
15. sns.lineplot(x='小时', y='Purchased_Electricity_A_no_storage', data=df, label='园区 A 无储能', color='b', linewidth=2.5)
16. sns.lineplot(x='小时', y='Purchased_Electricity_B_no_storage', data=df, label='园区 B 无储能', color='g', linewidth=2.5)
17. sns.lineplot(x='小时', y='Purchased_Electricity_C_no_storage', data=df, label='园区 C 无储能', color='r', linewidth=2.5)
18. plt.plot(df['小时'], purchased_electricity, label='有储能购电量', linestyle='--', color='k', linewidth=2.5)
19. plt.xlabel('时间 (小时)')
20. plt.ylabel('购电量 (kW)')
21. plt.title('各园区购电量对比')
22. plt.legend()

```

```

23.
24. # 绘制总成本对比
25. plt.subplot(2, 1, 2)
26. total_cost_before_series = df['Total_Cost_A_no_storage'] + df['Total_Cost_B_no_storage'] + df['Total_
Cost_C_no_storage']
27. total_cost_after_series = df['Total_Cost_A'] + df['Total_Cost_B'] + df['Total_Cost_C']
28. sns.lineplot(x=df[' 小 时 '], y=total_cost_before_series, label=' 无 储 能 总 成 本
', color='b', linewidth=2.5)
29. plt.plot(df[' 小 时 '], total_cost_after_series, label=' 有 储 能 总 成 本
', linestyle='--', color='k', linewidth=2.5)
30. plt.xlabel('时间 (小时)')
31. plt.ylabel('总成本 (元)')
32. plt.title('总成本对比')
33. plt.legend()
34.
35. # 调整布局并显示图形
36. plt.tight_layout()
37. plt.show()

1. # 输出优化后的策略
2. optimized_strategy = {
3.     '小时': df['小时'].tolist(),
4.     '购电量_有储能': purchased_electricity,
5.     '总成本_有储能': total_cost_after_series.tolist()
6. }
7.
8. print("优化后的策略:", optimized_strategy)

```

附录 1

介绍：问题一中运行成功后的部分截图

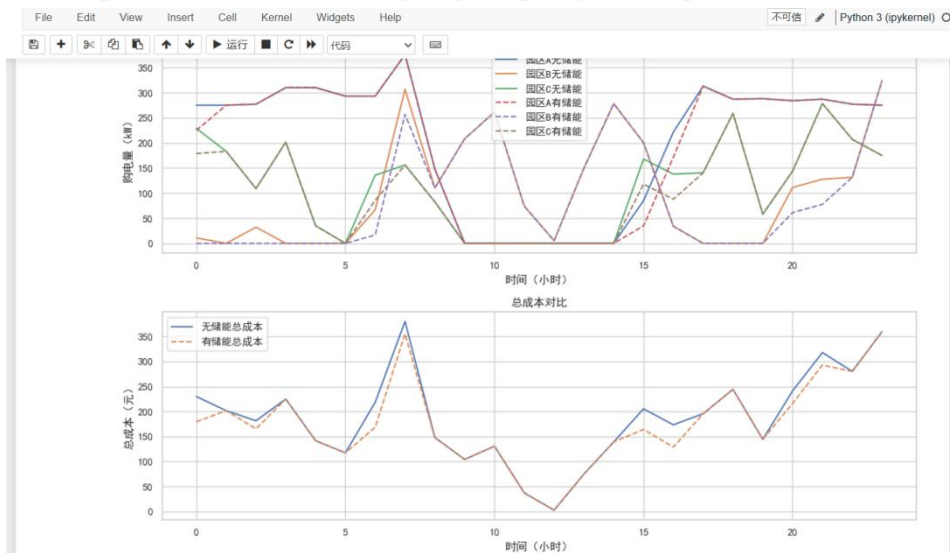
it[22]:

	时间 (h)	园区 A负荷 (kW)	园区 B负荷 (kW)	园区 C负荷 (kW)	园区A 光伏出力 (p.u.)	园区B风 电出力 (p.u.)	园区C光 伏出力 (p.u.)	园区C风 电出力 (p.u.)	园区A 光伏发电 电量 (kW)	园区 B风电 电量 (kW)	...	Purchased_Electricity_C	Excess_Electricity_A	Excess_Electricity_B	Exc
0	00:00:00	275	241	302	0.0000	0.2301	0.0000	0.1464	0.000	230.1	...	228.80	0.000		0.0
1	01:00:00	275	253	292	0.0000	0.3828	0.0000	0.2175	0.000	382.8	...	183.25	0.000		129.8
2	02:00:00	277	329	307	0.0000	0.2968	0.0000	0.3959	0.000	296.8	...	109.05	0.000		0.0
3	03:00:00	310	315	293	0.0000	0.4444	0.0000	0.1831	0.000	444.4	...	201.45	0.000		129.4
4	04:00:00	310	290	271	0.0000	0.5029	0.0000	0.4716	0.000	502.9	...	35.20	0.000		212.9
5	05:00:00	293	270	252	0.0000	0.3609	0.0000	0.6215	0.000	360.9	...	0.00	0.000		90.9
6	06:00:00	293	307	283	0.0000	0.2402	0.0000	0.2946	0.000	240.2	...	135.70	0.000		0.0
7	07:00:00	380	354	223	0.0058	0.0473	0.0105	0.1214	4.350	47.3	...	156.00	0.000		0.0
8	08:00:00	375	264	292	0.3026	0.1538	0.3280	0.0250	226.950	153.8	...	82.70	0.000		0.0
9	09:00:00	281	315	283	0.6020	0.1068	0.6314	0.3023	451.500	106.8	...	0.00	170.500		0.0
10	10:00:00	447	313	287	0.7711	0.0518	0.7936	0.0196	578.325	51.8	...	0.00	131.325		0.0
11	11:00:00	447	291	362	0.8555	0.2169	0.8925	0.1224	641.625	216.9	...	0.00	194.625		0.0
12	12:00:00	447	360	446	0.8531	0.3546	0.8999	0.3335	639.825	354.6	...	0.00	192.825		0.0
13	13:00:00	405	369	504	0.7842	0.2194	0.8221	0.2653	588.150	219.4	...	0.00	183.150		0.0
14	14:00:00	404	389	455	0.6437	0.1110	0.6667	0.1220	482.775	111.0	...	0.00	78.775		0.0
15	15:00:00	403	419	506	0.4242	0.2186	0.4275	0.1633	318.150	218.6	...	167.85	0.000		0.0

Out[27]:

	时间 (h)	Storage_A	Purchased_Electricity_A	Total_Generation_A
0	00:00:00	0	275.0	0.0
1	01:00:00	0	275.0	0.0
2	02:00:00	0	277.0	0.0
3	03:00:00	0	310.0	0.0
4	04:00:00	0	310.0	0.0

周铠然 · 仅供求职展示



附录 2

介绍：线性规划模型

```
1. import pandas as pd
2.
3. # 读取数据文件
4. file_path_load = 'C:\\Users\\XXX\\Desktop\\附件 1: 各园区典型日负荷数据.xlsx'
5. file_path_generation = 'C:\\Users\\XXX\\Desktop\\附件 2: 各园区典型日风光发电数据.xlsx'
6.
7. # 读取各园区负荷数据
8. df_load = pd.read_excel(file_path_load)
9. # 读取各园区风光发电数据
10. df_generation = pd.read_excel(file_path_generation)
11.
12. # 计算各园区总发电量（风电和光伏发电总和）
13. df_generation['总发电量_A'] = df_generation['园区 A 光伏出力 (p.u.)'] * 750 # 750 kW 为光伏装机容量
14. df_generation['总发电量_B'] = df_generation['园区 B 风电出力 (p.u.)'] * 1000 # 1000 kW 为风电装机容量
15. df_generation['总发电量_C'] = (df_generation['园区 C 光伏出力 (p.u.)'] * 600) + (df_generation['园区 C 风电出力 (p.u.)'] * 500) # 600 kW 光伏装机容量和 500 kW 风电装机容量
16.
17. # 合并三个园区的数据
18. df_combined = pd.merge(df_load, df_generation, on='时间 (h)')
19.
20. # 计算合并后的总发电量
21. df_combined['总发电量_total'] = df_combined['总发电量_A'] + df_combined['总发电量_B'] + df_combined['总发电量_C']
22.
23. # 计算合并后的总负荷需求
24. df_combined['总负荷需求_total'] = df_combined['园区 A 负荷 (kW)'] + df_combined['园区 B 负荷 (kW)'] + df_combined['园区 C 负荷 (kW)']
25.
26. # 假设电价随时间变化
27. df_combined['电价'] = 1.0 # 网购电价格为 1 元/kWh

1. from pulp import LpProblem, LpMinimize, LpVariable, LpSum, LpAffineExpression, value
2.
3. # 模型参数
4. T = len(df_combined) # 时间段数量
5. E_sto_max = 1800 # 储能系统最大容量 (kWh)
6. C_cha_max = 800 # 充电功率上限 (kW)
7. C_dis_max = 800 # 放电功率上限 (kW)
```

```

8.  eta_cha = 0.95 # 充电效率
9.  eta_dis = 0.95 # 放电效率
10.
11. # 创建线性规划问题
12. prob = LpProblem("Storage_Optimization", LpMinimize)
13.
14. # 决策变量
15. C_cha = LpVariable.dicts("C_cha", range(T), 0, C_cha_max)
16. C_dis = LpVariable.dicts("C_dis", range(T), 0, C_dis_max)
17. E_sto = LpVariable.dicts("E_sto", range(T+1), 0, E_sto_max)
18. P_purchase = LpVariable.dicts("P_purchase", range(T), 0)
19.
20. # 初始储能电量
21. prob += E_sto[0] == 0.1 * E_sto_max
22.
23. # 目标函数
24. prob += lpSum([P_purchase[t] * df_combined.at[t, '电价'] for t in range(T)])
25.
26. # 约束条件
27. for t in range(T):
28.     # 电力平衡约束
29.     prob += P_purchase[t] == df_combined.at[t, '总负荷需求_total'] - df_combined.at[t, '总发电量_total'] - C_dis[t] + C_cha[t]
30.
31. # 储能能量平衡约束
32. if t < T-1:
33.     # 构建储能能量平衡的表达式
34.     energy_balance_expr = E_sto[t] + eta_cha * C_cha[t] - C_dis[t] * (1 / eta_dis)
35.     prob += E_sto[t+1] == energy_balance_expr
36.
37. # 储能系统容量限制
38. prob += E_sto[t] >= 0.1 * E_sto_max
39. prob += E_sto[t] <= 0.9 * E_sto_max
40.
41. # 求解问题
42. prob.solve()
43.
44. # 提取结果
45. C_cha_opt = [value(C_cha[t]) for t in range(T)]
46. C_dis_opt = [value(C_dis[t]) for t in range(T)]
47. E_sto_opt = [value(E_sto[t]) for t in range(T+1)]
48. P_purchase_opt = [value(P_purchase[t]) for t in range(T)]
49.
50. # 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

```

```

51. total_purchase = sum(P_purchase_opt)
52. total_curtailment = sum([max(0, df_combined.at[t, '总发电量_total'] - df_combined.at[t, '总负荷需求_total'] - C_cha_opt[t]) for t in range(T)])
53. total_cost = sum([P_purchase_opt[t] * df_combined.at[t, '电价'] for t in range(T)])
54. average_cost_per_kWh = total_cost / sum(df_combined['总负荷需求_total'])
55.
56. # 输出结果
57. print(f"总购电量: {total_purchase} kWh")
58. print(f"总弃风弃光电量: {total_curtailment} kWh")
59. print(f"总供电成本: {total_cost} 元")
60. print(f"单位电量平均供电成本: {average_cost_per_kWh:.2f} 元/kWh")

```

附录 2

介绍：混合整数规划模型

```

1. import pandas as pd
2. from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpBinary, LpStatus
3.
4. # 读取数据文件
5. file_path_load = 'C:\Users\xxx\Desktop\附件 1: 各园区典型日负荷数据.xlsx'
6. file_path_generation = 'C:\xxx\Desktop\附件 2: 各园区典型日风光发电数据.xlsx'
7.
8. # 读取各园区负荷数据
9. df_load = pd.read_excel(file_path_load)
10. # 读取各园区风光发电数据
11. df_generation = pd.read_excel(file_path_generation)
12.
13. # 计算各园区总发电量（风电和光伏发电总和）
14. df_generation['总发电量_A'] = df_generation['园区 A 光伏出力 (p.u.)'] * 750 # 750 kW 为光伏装机容量
15. df_generation['总发电量_B'] = df_generation['园区 B 风电出力 (p.u.)'] * 1000 # 1000 kW 为风电装机容量
16. df_generation['总发电量_C'] = (df_generation['园区 C 光伏出力 (p.u.)'] * 600) + (df_generation['园区 C 风电出力 (p.u.)'] * 500) # 600 kW 光伏装机容量和 500 kW 风电装机容量
17.
18. # 合并三个园区的数据
19. df_combined = pd.merge(df_load, df_generation, on='时间 (h)')
20.
21. # 计算合并后的总发电量
22. df_combined['总发电量_total'] = df_combined['总发电量_A'] + df_combined['总发电量_B'] + df_combined['总发电量_C']
23.
24. # 计算合并后的总负荷需求

```

```

25. df_combined['总负荷需求_total'] = df_combined['园区 A 负荷(kw)'] + df_combined['园区 B 负荷(kw)'] + df_combined['园区 C 负荷(kw)']
26.
27. # 假设电价随时间变化
28. df_combined['电价'] = 1.0 # 网购电价格为 1 元/kWh
29.
30. # 模型参数
31. T = len(df_combined) # 时间段数量
32. E_sto_max = 1800 # 储能系统最大容量 (kWh)
33. C_cha_max = 800 # 充电功率上限 (kW)
34. C_dis_max = 800 # 放电功率上限 (kW)
35. eta_cha = 0.95 # 充电效率
36. eta_dis = 0.95 # 放电效率
37. electricity_cost = df_combined['电价'].to_numpy() # 电价数组
38.
39. # 创建 MILP 问题
40. prob = LpProblem("Storage_Optimization", LpMinimize)
41.
42. # 定义决策变量
43. C_cha = LpVariable.dicts("Charge_Power", range(T), 0, C_cha_max, cat='Continuous')
44. C_dis = LpVariable.dicts("Discharge_Power", range(T), 0, C_dis_max, cat='Continuous')
45. E_sto = LpVariable.dicts("Energy_Storage", range(T+1), 0, E_sto_max, cat='Continuous')
46. P_purchase = LpVariable.dicts("Purchased_Power", range(T), 0, None, cat='Continuous')
47. x_on = LpVariable.dicts("Discharge_On", range(T), 0, 1, cat=LpBinary)
48. 周铠然 · 仅供求职展示
49. # 定义一个大 M 值
50. M = 1000 # 这个值需要足够大，以确保逻辑约束的正确性
51.
52. # 目标函数：最小化总供电成本
53. prob += lpSum([P_purchase[t] * electricity_cost[t] for t in range(T)])
54.
55. # 约束条件
56. for t in range(T):
57.     # 电力平衡约束
58.     prob += P_purchase[t] + M * x_on[t] - C_cha[t] + df_combined['总发电量_total'].iloc[t] == df_combined['总负荷需求_total'].iloc[t]
59.
60.     # 储能能量平衡约束
61.     prob += E_sto[t+1] == E_sto[t] + eta_cha * C_cha[t] - C_dis[t] * eta_dis
62.
63.     # 储能系统容量限制
64.     prob += 0.1 * E_sto_max <= E_sto[t] <= 0.9 * E_sto_max
65.
66.     # 发电放电约束，确保放电只有在放电模式激活时才进行

```

```

67.     prob += C_dis[t] - C_dis_max * (1 - x_on[t]) <= 0
68.     prob += C_dis[t] - C_dis_max * x_on[t] >= 0
69.
70. # 初始和最终储能电量约束
71. prob += E_sto[0] == 0.1 * E_sto_max
72. prob += E_sto[T] == E_sto[0]
73.
74. # 求解问题
75. prob.solve()
76.
77. # 输出结果
78. if LpStatus[prob.status] == 'Optimal':
79.     print("Optimal Solution Found.")
80.     for t in range(T):
81.         print(f"Time {t}: Charge Power = {C_cho[t].varValue:.2f} kW, "
82.               f"Discharge Power = {C_dis[t].varValue:.2f} kW, "
83.               f"Storage Energy = {E_sto[t].varValue:.2f} kWh, "
84.               f"Purchased Power = {P_purchase[t].varValue:.2f} kWh, "
85.               f"Discharge On = {x_on[t].varValue}")
86. else:
87.     print("No optimal solution found.")

```

周铠然 · 仅供求职展示

附录 2

介绍：基于遗传算法的多目标规划

```

1. import numpy as np
2. import random
3. from deap import base, creator, tools, algorithms
4.
5. # 实际的发电数据
6. solar_generation = np.array([0, 0, 0, 0, 0, 0, 0, 0.0163, 0.6306, 1.2334, 1.5647, 1.748, 1.753, 1.606
3, 1.3104, 0.8517, 0.0835, 0, 0, 0, 0, 0, 0])
7. wind_generation = np.array([0.3765, 0.6003, 0.6927, 0.6275, 0.9745, 0.9824, 0.5348, 0.1687, 0.1788, 0
.4091, 0.0714, 0.3393, 0.6881, 0.4847, 0.233, 0.3819, 0.6424, 0.6829, 0.8191, 0.7945, 0.39, 0.3438, 0.3432, 0.
2323])
8. load_demand = np.random.uniform(300, 600, 24) # 假设的负荷需求
9.
10. # 定义适应度函数的多目标：最小化成本，最大化可再生能源利用率
11. creator.create("FitnessMulti", base.Fitness, weights=(-1.0, 1.0))

```

```

12. creator.create("Individual", list, fitness=creator.FitnessMulti)
13.
14. # 初始化种群和遗传操作
15. toolbox = base.Toolbox()
16. toolbox.register("attr_float", random.uniform, 0, 1)
17. toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_float, n=24)
18. toolbox.register("population", tools.initRepeat, list, toolbox.individual)
19.
20. # 评估函数: 计算总成本和可再生能源使用率
21. def evaluate(individual):
22.     cost = 0
23.     renewable_usage = 0
24.     for i in range(24):
25.         # 计算每小时的实际发电量
26.         solar_output = individual[i] * solar_generation[i]
27.         wind_output = individual[i] * wind_generation[i]
28.         total_generation = solar_output + wind_output
29.         renewable_usage += total_generation
30.
31.         # 如果总发电量小于负荷, 则需要购买电力
32.         if total_generation < load_demand[i]:
33.             cost += (load_demand[i] - total_generation) * 0.5 # 购电成本为 0.5 元/kWh
34.
35.         # 将总可再生发电量标准化为总发电量的比例
36.     total_possible_generation = sum(solar_generation + wind_generation)
37.     renewable_usage = renewable_usage / total_possible_generation
38.
39.     return (cost, renewable_usage)
40.
41. # 注册遗传算法需要的操作
42. toolbox.register("evaluate", evaluate)
43. toolbox.register("mate", tools.cxTwoPoint)
44. toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
45. toolbox.register("select", tools.selNSGA2)
46.
47. # 种群初始化
48. population = toolbox.population(n=50)
49.
50. # 运行遗传算法
51. result, log = algorithms.eaSimple(population, toolbox, cxpb=0.7, mutpb=0.2, ngen=40, verbose=True)
52.
53. # 输出最佳解
54. best_inds = tools.sortNondominated(population, len(population), first_front_only=True)[0]
55. best_ind = best_inds[0]

```

```
56. print('Best Solution:', best_ind)
57. print('Best Fitness:', best_ind.fitness.values)
```

周铠然 · 仅供求职展示