电力系统储能调度优化的综合研究

摘　要

在实现“双碳”目标的背景下，储能技术在中国新型电力系统的构建中扮演着重要角色。针对工商业储能系统的调度优化问题，本文使用快速傅里叶变换（FFT）预测模型、混合整数规划模型、遗传算法等解决了对储能设备的预测、谷套利收益的最大化以及需量电价成本的最小化等问题。

对于问题一，以 15 分为周期，15 秒为间隔，预测三个储能设备未来一天的负荷。我们首先对历史数据进行充分预处理，对于小面积的缺失值采取了**线性插值**，大面积的缺失值直接剔除。然后。我们选择**快速傅里叶变换（FFT）预测模型**，以提取负荷数据的周期性特征，通过将时域数据转换至频域并识别主要频率成分，再利用逆傅里叶变换回到时域，有效预测未来的负荷变化趋势。最后将原始的历史负荷数据与模型预测后的负荷数据进行对比，两者数据基本保持一致。

对于问题二，根据负荷预测结果，制定和优化储能系统的充放电策略，以在满足实际负荷需求的同时，最大化经济效益。我们建立了**混合整数规划模型**，处理连续变量和二进制决策变量之间的复杂关系。在优化过程中，模型的目标是其满足实际负荷的前提下，尽可能的获得更大的收益。此外，我们还建立了多种约束条件，如储能系统不能同时充电和放电、电力负荷必须得到满足等。通过问题一预测的数据与优化模型相结合，使用**遗传算法**对充放电策略进行求解，最终求得最大收益：单柜收益为 **272.68 元**，收益提升 **75.20 元**, 双并柜收益为 **671.06 元**，收益提升 **149.40 元**，三并柜收益为 **1076.30 元**，收益提升 **244.94 元**。

对于问题三，要求在假设最大需量可以超过合同规定的 105% 的情况下，重新设计储能系统的充放电策略，以进一步优化经济效益。我们在现有模型的基础上重新建立**混合整数规划模型**，引入允许最大需量超限的条件，并重新调整优化目标函数，以综合考虑超过 105% 部分的需量电费。其次，需对充放电策略进行重新设计，以便在提高峰谷套利收益的同时，合理控制最大需量的超限程度，避免额外的电价成本过高。最终，使用**遗传算法**对充放电策略进行求解，最终求得最大收益：单柜收益为 **272.94 元**，收益提升 **75.46 元**，双并柜收益为 **670.11 元**，收益提升 **148.45 元**，三并柜收益为 **1075.72 元**，收益提升 **244.36 元**。

关键词:　**快速傅里叶变换预测 混合整数规划模型 遗传算法 电量负荷**

一　问题的背景和重述

1.1　问题背景

在“双碳”目标的引领下，中国计划到 2035 年基本建成新型电力系统，储能技术在其中起着关键作用。当前，工商业储能公司通过在低电价时段存储电能、高电价时段释放电能，实现峰谷套利以获取利润。然而，储能系统的经济效益不仅依赖电价差异，还受到储能容量、充放电功率上限以及需量电价等因素的影响。因此，优化储能系统的调度策略，以在满足电力需求的同时最大化经济收益，成为储能公司面临的核心问题。

1.2　问题重述

为了优化电力系统中的储能调度，以实现峰谷套利的最大化和需量电价成本的最小化。本文根据附件的数据建立数学模型，解决以下几个关键问题：

1. 根据历史的电力数据，预测未来一天三个储能设备的负荷。这需要在 15 分钟为周期、15 秒为间隔的条件下，通过分析和处理历史数据，得出未来的负荷预测结果。

2. 利用预测的负荷数据，建立并优化储能系统的充放电策略，确保满足实际负荷的同时，尽可能提高收益。并比较新策略与原策略的收益差异。

3. 假设最大需量可以根据情况超过设定的合同最大需量的 105%，请重新建模和设计三个储能设备的充放电策略，并将结果保存在 预测结果.xlsx 的对应位置中，同时请指出相对于原来充放电策略的收益提升。

二　问题分析

2.1　问题一的分析

问题一要求对三个储能设备未来一天的负荷进行预测，这一任务在电力系统储能调度中至关重要。负荷预测通过分析历史用电数据来推测未来的电力需求，是制定合理的储能策略、实现经济效益最大化的关键步骤。首先，需对历史数据进行充分预处理，确保数据的完整性和准确性。接着，选择快速傅里叶变换（FFT），以提取负荷数据的周期性特征。通过将时域数据转换至频域并识别主要频率成分，再利用逆傅里叶变换回到时域，能够有效预测未来的负荷变化趋势。这种方法特别适用于处理具有周期性波动的电力负荷数据，有助于更精确地预测未来的需求，从而为储能调度策略的制定提供数据支持。

2.2　问题二的分析

问题二要求根据负荷预测结果，制定和优化储能系统的充放电策略，以在满足实际负荷需求的同时，最大化经济效益。该问题的核心在于平衡峰谷套利收益与需量电价成本之间的关系。首先，需要根据储能系统的特性，如充放电功率上限、容量限制等，建立合适的数学模型。混合整数规划模型是一种有效的选择，它能够处理连续变量（如充放电功率）和二进制决策变量（如充放电状态）之间的复杂关系。在优化过程中，模型的目标是通过在低电价时段充电、高电价时段放电，最大化峰谷套利收益，同时通过合理控制最大需量，尽可能降低需量电价成本。此外，优化策略还需满足多种约束条件，如储能系统不能同时充电和放电、电力负荷必须得到满足等。通过问题一预测的数据与优化模型相结合，制定出最优的充放电策略，从而在经济效益和系统安全性之间取得最佳平衡。

2.3 问题三的分析

问题三要求在假设最大需量可以超过合同规定的 105% 的情况下，重新设计储能系统的充放电策略，以进一步优化经济效益。在于需量电价成本将随着最大需量的增加而显著提高，因此需要在追求更高收益和控制成本之间找到新的平衡。首先，需要在现有模型的基础上，引入允许最大需量超限的条件，并重新调整优化目标函数，以综合考虑超过 105% 部分的需量电费。其次，需对充放电策略进行重新设计，以便在提高峰谷套利收益的同时，合理控制最大需量的超限程度，避免额外的电价成本过高。为此，可以采用混合整数规划模型，将不同储能设备的充放电行为与实际负荷需求相匹配，并引入新的约束条件以限制超出部分的影响。

三 模型假设

1. 假设在模型预测期间，电力系统的负荷需求保持相对稳定，不会出现极端的波动或突发性需求变化。
2. 假设储能系统的充放电效率稳定，不会因为设备老化或其他外部因素导致性能波动，且储能系统的容量和功率上限不会被超出。
3. 假设采用的负荷预测模型能够准确反映未来的负荷变化趋势，预测结果可以为优化策略提供可靠的参考。
4. 假设储能系统能够灵活调整放电策略，在不超过合同最大需量 105% 的情况下，确保最大化收益的同时尽量减少需量电价成本。

四 符号说明

符号	含义
$P_c^{(i)}(t)$	时间 t 时储能设备 i 的充电功率（kW）
$P_d^{(i)}(t)$	时间 t 时储能设备 i 的放电功率（kW）
$E^{(i)}(t)$	时间 t 时储能设备 i 的电量存储水平（kWh）
$C_s(t)$	时间 t 时电力售电价格（元/kWh）
$C_b(t)$	时间 t 时电力购电价格（元/kWh）
D_m	最大需量（kW）
D_c	合同规定的最大需量（kW）
α	单位需量电价（元/kW）
η_c	充电效率
η_d	放电效率
$L(t)$	时间 t 时的实际负荷需求（kW）
$P_g(t)$	时间 t 时从电网获取的功率（kW）

五　数据预处理

由于数据集中存在部分缺失值，直接使用这些数据进行分析可能会导致结果偏差。为确保数据的完整性和连续性，我们检查了数据集中缺失值的分布情况。为了合理填补这些缺失值，我们对小范围连续的数据采用了线性插值，对大范围缺失的数据进行剔除，部分处理数据如下所示：

表 1: 部分处理数据

时间	CN	SD
2023-11-03 00:00:00	0.0	73.8
2023-11-03 00:00:15	0.0	73.2
2023-11-03 00:00:30	0.0	74.1
2023-11-03 00:00:45	0.1	75.6
2023-11-03 00:01:00	49.9	125.4
2023-11-03 00:01:15	50.0	126.0
2023-11-03 00:01:30	49.9	126.9
2023-11-03 00:01:45	50.0	127.2
2023-11-03 00:02:00	49.9	126.6
2023-11-03 00:02:15	49.7	126.3
2023-11-03 00:02:30	49.7	126.6
⋮	⋮	⋮

六　模型的建立与求解

6.1　问题一模型的建立与求解

6.1.1　模型建立

为了解决问题一中对三个储能设备未来一天负荷的预测任务，我们采用了快速傅里叶变换（FFT）[1] 方法。FFT 擅长处理周期性时间序列数据，能够有效地识别数据中的主要周期成分并进行预测。我们通过对历史负荷数据进行预处理，将其转换到频域后提取出主要的频率成分，随后通过逆傅里叶变换（IFFT）重构时域数据，从而实现对未来一天负荷情况的准确预测。

6.1.2　快速傅里叶变换预测

快速傅里叶变换（FFT）是一种高效的频域分析方法，能够将时间序列数据从时域转换到频域，从而识别出数据中的主要周期成分。利用 FFT 进行预测时，可以分析负荷数据中的周期性变化趋势，从而对未来的负荷进行预测。以下是快速傅里叶预测的具体过程：

1. 时域到频域转换

通过快速傅里叶变换，我们将历史负荷数据从时域转换到频域。频域中的每个频率分量对应着时域数据中的一种周期性变化。快速傅里叶变换的数学表达式为：

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-j2\pi kn/N}, \quad k = 0, 1, 2, \ldots, N-1 \tag{1}$$

其中，x_n 为时域信号，X_k 为频域信号，N 为信号的采样点数，k 为频率索引。

2. 频域特征提取与预测

在频域中，我们可以识别出负荷数据的主要周期成分。这些周期成分代表了负荷变化的规律性。通过分析这些主要成分，可以预测出未来时间段内的负荷趋势。

为了更好地预测，我们可以选择性地保留主要的频率成分，忽略噪声和较小的成分。这使得预测更具稳定性和准确性。

3. 逆傅里叶变换（IFFT）

在提取到主要频率成分后，我们使用逆傅里叶变换（IFFT）将这些频域信息转换回时域，从而生成预测的负荷数据。逆傅里叶变换的数学表达式为：

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{j2\pi kn/N}, \quad n = 0, 1, 2, \ldots, N-1 \tag{2}$$

通过 IFFT，我们可以得到未来时刻的负荷预测数据，这些数据为储能设备的优化调度提供了准确的依据。

通过快速傅里叶预测方法，我们不仅能够有效地捕捉负荷数据中的周期性趋势，还能够基于这些趋势对未来的负荷变化进行准确预测。这为储能系统的优化管理提供了坚实的数据支持。

6.1.3 模型求解

对于三个储能设备未来一天的负荷预测，我们利用快速傅里叶变换（FFT）对三个储能设备的未来一天负荷进行预测。我们将预处理后的历史数据转换到频域中，通过 FFT 提取出主要的频率成分。接着，通过逆傅里叶变换（IFFT）将频域数据转换回时域，从而预测未来一天的负荷情况。

当频域分析和逆变换过程完成后，我们得到了未来一天的负荷预测结果。预测结果以 15 分钟为周期，提供未来 24 小时内各时段的负荷值，最终将这些预测数据按要求存储到指定的 Excel 文件中。单柜的未来一天的负荷预测结果如下所示：

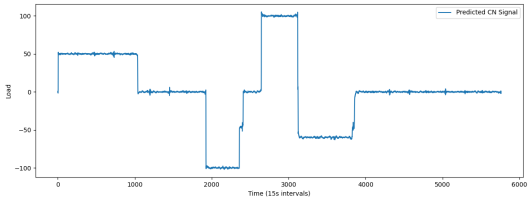

图 1: 未来一天的单柜的储能用电（CN）

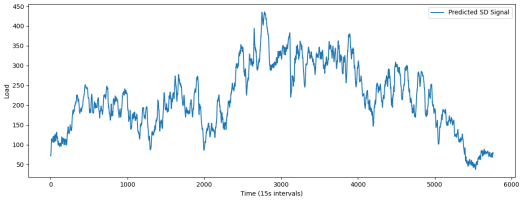

图 2: 未来一天的单柜的市电（SD）

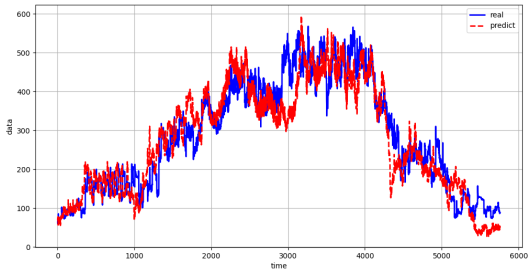

图 3: 未来一天的单柜的负荷

二并柜的未来一天的负荷预测结果如下所示：

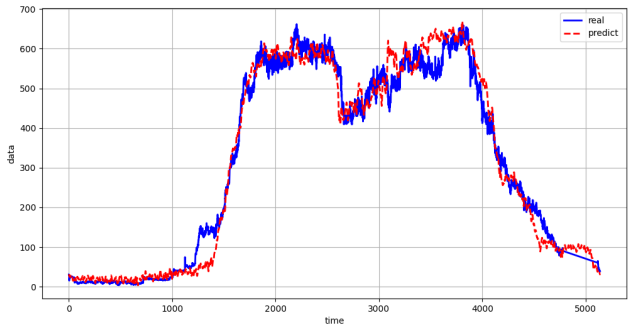

图 4: 未来一天的二并柜的负荷

三并柜的未来一天的负荷预测结果如下所示：

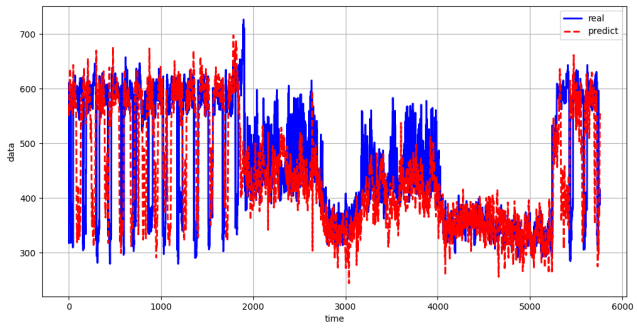

图 5: 未来一天的三并柜的负荷

其中，负载的计算公式如下：

$$\text{Load} = \text{SD} - \text{CN} \tag{3}$$

其中，Load 表示负荷，SD 表示市电功率，CN 表示储能功率。

我们计算出来三个储能设备未来一天的负荷的预测数据并将其填入了相应的 excel 表格，单柜的未来一天的负荷部分预测结果如下所示：

表 2: 单柜未来一天的负荷的部分预测数据

时间	预测的负载	真实的负载
2024-03-18 00:00:00	72.1926	63.6
2024-03-18 00:00:15	71.5487	60.0
2024-03-18 00:00:30	70.2663	67.2
2024-03-18 00:00:45	73.8635	66.6
2024-03-18 00:01:00	76.4903	66.9
2024-03-18 00:01:15	75.6353	70.1
2024-03-18 00:01:30	39.2429	70.7
2024-03-18 00:01:45	39.3064	69.3

二并柜的未来一天的负荷部分预测结果如下所示：

表 3: 二并柜未来一天的负荷的部分预测数据

时间	预测的负载	真实的负载
2024-03-18 00:00:00	30.8	29.4300
2024-03-18 00:00:15	30.8	29.0387
2024-03-18 00:00:30	30.8	29.6527
2024-03-18 00:00:45	29.6	29.8417
2024-03-18 00:01:00	15.4	28.1960
2024-03-18 00:01:15	20.0	27.2208
2024-03-18 00:01:30	22.2	27.3113
2024-03-18 00:01:45	21.0	27.4274

三并柜的未来一天的负荷部分预测结果如下所示：

表 4: 三并柜未来一天的负荷的部分预测数据

时间	预测的负载	真实的负载
2024-03-18 00:00:00	321.3	571.3533
2024-03-18 00:00:15	317.4	551.2207
2024-03-18 00:00:30	579.5	569.1884
2024-03-18 00:00:45	590.5	558.9060
2024-03-18 00:01:00	611.3	586.8409
2024-03-18 00:01:15	582.7	604.8841
2024-03-18 00:01:30	409.4	606.5477
2024-03-18 00:01:45	431.7	603.9339

6.1.4 模型检验

为了验证通过快速傅里叶变换（FFT）得到的预测模型的准确性，我们将原始的历史负荷数据与模型重构后的负荷数据进行对比，以单柜的未来一天的负荷部分预测结果为例，如下图所示：

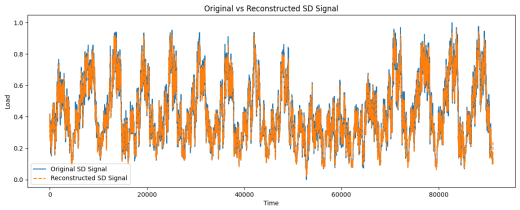

图 6: 负荷（SD）原始信号与重构信号对比

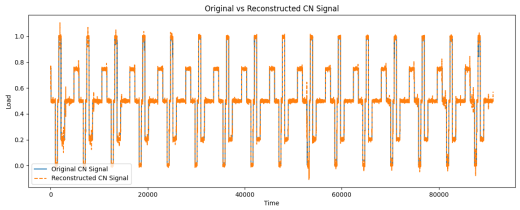

图 7: 负荷（CN）原始信号与重构信号对比

从图中可以看出，重构信号与原始信号的变化趋势基本一致，尤其在频率成分上，重构信号能够很好地捕捉到原始信号的主要特征，说明模型在提取主要频率成分方面具有较好的表现。同时，市电负荷（SD）和储能负荷（CN）的重构信号均能够较好地拟合原始信号，表明模型在不同负载情况下均具备较高的适用性。

6.2 问题二模型的建立与求解

6.2.1 混合整数规划模型

在电力系统储能调度中，储能公司通过峰谷套利以获取最大收益，并且需要控制最大需量以降低需量电价成本。本问题要求在满足储能系统及电力负荷需求的前提下，制定并优化储能系统的充放电策略，使得峰谷套利收益最大化，同时需量电价成本最小化。因此我们建立了混合整数规划模型 [2]，具体如下：

1. 决策变量

- $P_c(t)$：储能系统在时间段 t 的充电功率（单位：kW），这是一个连续变量。
- $P_d(t)$：储能系统在时间段 t 的放电功率（单位：kW），这是一个连续变量。
- $E(t)$：储能系统在时间段 t 的电量存储水平（单位：kWh），这是一个连续变量。
- $u_c(t)$：储能系统在时间段 t 是否进行充电的二进制变量，取值为 0 或 1。
- $u_d(t)$：储能系统在时间段 t 是否进行放电的二进制变量，取值为 0 或 1。

2. 目标函数

优化目标是通过制定合适的充放电策略来最大化储能系统的总收益。具体定义为：

$$\text{Max } Z = R_a - 0.003 \times C_m \tag{4}$$

其中：

- **峰谷套利收益** R_a 表示通过在电价低谷时充电和电价高峰时放电所获得的净收益，计算公式为：

$$R_a = \sum_{t=1}^{T} (P_d(t) \times C(t)) - \sum_{t=1}^{T} (P_c(t) \times C(t)) \tag{5}$$

其中，$P_d(t)$ 和 $P_c(t)$ 分别表示时刻 t 的放电功率和充电功率，$C(t)$ 表示时刻 t 的电价。

- **需量电价成本** C_m 是根据最大需量计算的成本，具体为：

$$C_m = \alpha \times D_m \tag{6}$$

其中，D_m 是结算周期（通常为一个月）内的最大需量，α 表示单位需量电价。

3. 约束条件

在优化过程中，需要满足以下约束条件：

a. 储能系统充放电功率约束

$$0 \leq P_c(t) \leq P_{c,\max} \tag{7}$$

$$0 \leq P_d(t) \leq P_{d,\max} \tag{8}$$

其中，$P_{c,\max}$ 和 $P_{d,\max}$ 分别表示储能系统的最大充电和放电功率，对于单柜、二并柜、三并柜的功率限制分别为 101.4 kW、201.5 kW 和 308.1 kW。

b. 储能系统容量约束

$$0 \leq E(t) \leq E_{\max} \tag{9}$$

其中，$E(t)$ 表示时刻 t 的储能状态（即剩余能量），$E_{\max}$ 为储能系统的总容量限制，每个电柜的容量为 215 kWh。

c. 储能系统同时充放电限制

储能系统在任一时刻不能同时进行充电和放电，即：

$$P_c(t) \times P_d(t) = 0 \tag{10}$$

d. 电力负荷需求约束

在每个时刻 t，储能系统与市电结合需要满足公司所有的用电需求：

$$P_s(t) + P_d(t) \geq L(t) \tag{11}$$

其中，$P_s(t)$ 表示时刻 t 从市电获取的功率，$L(t)$ 表示时刻 t 的实际负荷需求。

e. 需量电价约束

储能系统放电应控制最大需量不超过合同最大需量的 105%，即：

$$D_m \leq 1.05 \times D_c \tag{12}$$

其中，D_c 表示合同规定的最大需量。

4. 优化模型的整合呈现

$$\text{Max } Z = R_a - 0.003 \times C_m \tag{13}$$

$$\text{s.t.} \begin{cases} C_m = \alpha \times D_m \\ R_a = \sum_{t=1}^{T} \left(P_d(t) \times C(t)\right) - \sum_{t=1}^{T} \left(P_c(t) \times C(t)\right) \\ 0 \leq P_c(t) \leq P_{c,\max} \\ 0 \leq P_d(t) \leq P_{d,\max} \\ 0 \leq E(t) \leq E_{\max} \\ P_c(t) \times P_d(t) = 0 \\ P_s(t) + P_d(t) \geq L(t) \\ D_m \leq 1.05 \times D_c \end{cases} \tag{14}$$

6.2.2 遗传算法求解

由于本问题的约束条件较多，传统的遍历方法时间成本较高，因此我们采用遗传算法对模型进行求解，具体算法步骤如下：

1. 编码

根据模型特点，采用实数和二进制混合编码方式。染色体的每个基因表示储能系统在某个时间段的充电功率 $P_c(t)$、放电功率 $P_d(t)$ 以及充放电状态 $u_c(t)$ 和 $u_d(t)$。染色体长度为 $3T$，其中 T 表示调度周期内的时间段数量。

2. 种群初始化

随机生成初始种群，确保每个个体的充放电策略满足基本约束条件。即在任意时间段 t，充电功率 $P_c(t)$ 和放电功率 $P_d(t)$ 应满足功率限制，同时保证不能同时进行充电和放电。

3. 适应度计算

适应度函数基于总收益 Z 进行评估：

$$f(\text{染色体}) = R_a - 0.003 \times C_m \tag{15}$$

若染色体违反了功率、容量或充放电互斥约束，则对其适应度值进行惩罚。

4. 遗传操作

- **选择复制**：采用轮盘赌选择法，保留父代中适应度最高的个体以替换子代中最差的个体。步骤如下：
  - 计算当前种群中每个染色体的适应度 $\text{fit}(i)$；
  - 计算种群中所有染色体的适应度之和：

$$\text{sumfit} = \sum \text{fit}(i) \tag{16}$$

  - 计算每个染色体的选择概率：

$$p(i) = \frac{\text{fit}(i)}{\text{sumfit}} \tag{17}$$

  - 计算累计概率 $ps(i)$ 并执行选择操作。
- **交叉**：交叉操作通过交换两个染色体的部分基因序列生成新的个体。改进后的交叉步骤如下：
  - Step 1：在两个父代染色体上各自随机选一段时间序列（子路径 A 与子路径 B）；
  - Step 2：将子路径 A 和 B 前置于新的子代染色体上；
  - Step 3：将剩余的时间序列按父代顺序依次添加，确保子代染色体符合功率和充放电状态的约束；
  - Step 4：对生成的染色体进行解码和重新编码，并计算适应度，保留适应度较高的个体。
- **变异**：在染色体中随机选取某个时间段 t 的充电功率 $P_c(t)$ 或放电功率 $P_d(t)$ 进行小幅度扰动，同时交换充放电状态。若变异后的染色体满足约束且适应度提高，则保留该变异。

5. 进化机制

随着代数的增加，种群逐渐优化，算法逐步逼近最优解。采用精英保留策略，防止陷入局部最优，持续改进直到满足终止条件。

6.2.3 模型求解

在满足实际负荷需求的前提下，本模型的优化目标是尽可能地获得更大的经济收益。具体来说，优化目标分为两个方面：一是最大化峰谷套利收益，策略是在电价较低的谷电时段尽可能地充电达到储能系统的容量上限，在电价较高的峰电时段尽可能地放电；二是最小化需量电价成本，即通过优化策略，尽可能降低月度最大需量，进而减少需量电价的支出。

为此，我们将收益划分为两部分：固定最优利益和可优化利益。固定最优利益是指通过峰谷套利在固定条件下可以获得的收益，这部分收益主要取决于电价差异，不受最大需量的影响。可优化利益则需要综合考虑多个因素，包括当前时间的负荷情况、储能系统的充放电策略以及月度最大需量的影响。因

此，可优化利益的核心在于对特定时间段（如 11:00-13:00）的充放电策略进行精细调整，以实现整体收益的最优化。

模型求解中采用了遗传算法来解决该优化问题，依据实际的数据求解结果如下：

表 5: 基于实际数据的三并柜收益情况

储能设备	需量电价/元	峰谷套利收益/元	总收益/元	收益提升/元
单柜	25886.44	320.93	243.28	70.28
双并柜	38482.15	748.64	643.71	122.05
三并柜	35263.54	1136.72	1030.93	199.57

此外，我们通过问题一求得的三并柜未来一天的负荷预测数据对模型重新求解，求解结果如下：

表 6: 基于预测数据的三并柜收益情况

储能设备	需量电价/元	峰谷套利收益/元	总收益/元	收益提升/元
单柜	22562.03	435.51	272.68	75.20
双并柜	35504.52	777.58	671.06	149.40
三并柜	34427.54	1179.58	1076.30	244.94

为了使总收益最大化，我们制定了如下的储能调整策略，以单柜为例，部分时间储能时间安排如下所示：

表 7: 未来一天的单柜部分储能策略

时间	储能	时间	储能
2024-03-18 11:00:00	96.91818895	2024-03-18 11:00:15	99.8207336
2024-03-18 11:00:30	99.43858618	2024-03-18 11:00:45	97.18626379
2024-03-18 11:01:00	98.96409412	2024-03-18 11:01:15	96.29145835
2024-03-18 11:01:30	99.65428201	2024-03-18 11:01:45	100.85167861
2024-03-18 11:02:00	99.78881866	2024-03-18 11:02:15	98.65706403
2024-03-18 11:02:30	99.68375451	2024-03-18 11:02:45	98.65117013
2024-03-18 11:03:00	100.28378437	2024-03-18 11:03:15	96.53505404
2024-03-18 11:03:30	97.30889469	2024-03-18 11:03:45	97.88565202
⋮	⋮	⋮	⋮
2024-03-18 12:59:45	95.25718767	2024-03-18 13:00:00	100.99519034

6.3 问题三模型的建立与求解

6.3.1 混合整数规划模型

在电力系统储能调度中，储能公司通过峰谷套利以获取最大收益，并且需要优化储能系统的充放电策略来控制需量电价成本。在问题三中，假设最大需量可以根据情况超过设定的合同最大需量的 105%，但超过部分将增加需量电费。因此我们建立了混合整数规划模型，具体如下：

1. 决策变量

- $P_c^{(i)}(t)$：储能设备 i 在时间段 t 的充电功率（单位：kW），这是一个连续变量。
- $P_d^{(i)}(t)$：储能设备 i 在时间段 t 的放电功率（单位：kW），这是一个连续变量。
- $E^{(i)}(t)$：储能设备 i 在时间段 t 的电量存储水平（单位：kWh），这是一个连续变量。
- $u_c^{(i)}(t)$：储能设备 i 在时间段 t 是否进行充电的二进制变量，取值为 0 或 1。
- $u_d^{(i)}(t)$：储能设备 i 在时间段 t 是否进行放电的二进制变量，取值为 0 或 1。

2. 目标函数

优化目标是通过制定合适的充放电策略来最大化储能系统的总收益。具体定义为：

$$\text{Max } Z = R_a - 0.003 \times C_m \tag{18}$$

其中：

- **峰谷套利收益** R_a 表示通过在电价低谷时充电和电价高峰时放电所获得的净收益，计算公式为：

$$R_a = \sum_{t=1}^{T} \sum_{i=1}^{N} \left(P_d^{(i)}(t) \times C_s(t) - P_c^{(i)}(t) \times C_b(t) \right) \tag{19}$$

其中，$P_d^{(i)}(t)$ 和 $P_c^{(i)}(t)$ 分别表示储能设备 i 在时刻 t 的放电功率和充电功率，$C_s(t)$ 和 $C_b(t)$ 分别表示时刻 t 的售电价格和购电价格。

- **需量电价成本** C_m 是根据最大需量计算的成本，具体为：

$$C_m = \alpha \times \max_t \left(\sum_{i=1}^{N} \left(P_g(t) + P_d^{(i)}(t) - P_c^{(i)}(t) \right) \right) \tag{20}$$

其中 C_m 是考虑到实际需量可能超过合同规定 105% 的部分，需量电价将根据超出部分进行计算。

3. 约束条件

在优化过程中，需要满足以下约束条件：

a. 能量平衡约束

$$P_g(t) + P_d^{(i)}(t) - P_c^{(i)}(t) = P_l(t) \quad \forall t, \forall i \tag{21}$$

该约束确保在每个时刻 t，通过电网购买的功率 $P_g(t)$ 和储能系统的放电功率 $P_d^{(i)}(t)$ 总和减去充电功率 $P_c^{(i)}(t)$ 后，能够满足实际负荷需求 $P_l(t)$。

b. 储能功率限制

$$0 \leq P_c^{(i)}(t) \leq P_c^{(i),\max} \quad \forall t, \forall i \tag{22}$$

$$0 \leq P_d^{(i)}(t) \leq P_d^{(i),\max} \quad \forall t, \forall i \tag{23}$$

该约束规定了储能设备 i 在任一时刻 t 的充放电功率必须在其最大充电功率 $P_c^{(i),\max}$ 和最大放电功率 $P_d^{(i),\max}$ 范围内。

c. 储能容量限制

$$0 \leq E^{(i)}(t) \leq E^{(i),\max} \quad \forall t, \forall i \tag{24}$$

该约束确保储能设备 i 在任一时刻 t 的储能状态 $E^{(i)}(t)$ 必须在其最大储能容量 $E^{(i),\max}$ 范围内。

d. 储能状态更新

$$E^{(i)}(t+1) = E^{(i)}(t) + \eta_c \cdot P_c^{(i)}(t) \cdot \Delta t - \frac{P_d^{(i)}(t) \cdot \Delta t}{\eta_d} \quad \forall t, \forall i \tag{25}$$

该约束描述了储能设备在不同时刻的能量状态变化，其中 η_c 和 η_d 分别表示充电和放电效率，Δt 表示时间间隔。

e. 储能设备不能同时充放电

$$P_c^{(i)}(t) \times P_d^{(i)}(t) = 0 \quad \forall t, \forall i \tag{26}$$

该约束确保储能设备在任一时刻不能同时进行充电和放电操作。

f. 电力负荷需求约束

$$P_g(t) + \sum_{i=1}^{N} P_d^{(i)}(t) \geq L(t) \quad \forall t \tag{27}$$

该约束确保在每个时刻 t，储能系统与市电结合能够满足公司所有的用电需求 $L(t)$。

g. 需量电价约束

$$\max_t \left(P_g(t) + \sum_{i=1}^{N} P_d^{(i)}(t) - \sum_{i=1}^{N} P_c^{(i)}(t) \right) \geq 1.05 \times D_c \tag{28}$$

该约束允许最大需量超过合同规定的 105

h. 需量电费计算

$$C_m = \alpha \times \max_t \left(P_g(t) + \sum_{i=1}^{N} P_d^{(i)}(t) - \sum_{i=1}^{N} P_c^{(i)}(t) \right) \tag{29}$$

需量电费 C_m 的计算基于实际超过合同最大需量 105

4. 优化模型的整合呈现

$$\text{Max } Z = R_a - 0.003 \times C_m \tag{30}$$

$$
\text{s.t.}\begin{cases}
R_a = \sum_{t=1}^{T}\sum_{i=1}^{N}\left(P_d^{(i)}(t)\times C_s(t) - P_c^{(i)}(t)\times C_b(t)\right) & \\
C_m = \alpha \times \max_t\left(\sum_{i=1}^{N}\left(P_g(t) + P_d^{(i)}(t) - P_c^{(i)}(t)\right)\right) & \\
0 \le P_c^{(i)}(t) \le P_c^{(i),\max}, & \forall t, \forall i \\
0 \le P_d^{(i)}(t) \le P_d^{(i),\max}, & \forall t, \forall i \\
0 \le E^{(i)}(t) \le E^{(i),\max}, & \forall t, \forall i \\
u_c^{(i)}(t) + u_d^{(i)}(t) \le 1, & \forall t, \forall i \\
P_g(t) + \sum_{i=1}^{N} P_d^{(i)}(t) \ge L(t), & \forall t \\
E^{(i)}(t+1) = E^{(i)}(t) + \eta_c \cdot P_c^{(i)}(t)\cdot \Delta t - \frac{P_d^{(i)}(t)\cdot\Delta t}{\eta_d}, & \forall t, \forall i \\
P_c^{(i)}(t) \times P_d^{(i)}(t) = 0, & \forall t, \forall i \\
\max_t\left(P_g(t) + \sum_{i=1}^{N} P_d^{(i)}(t) - \sum_{i=1}^{N} P_c^{(i)}(t)\right) > 1.05 \times D_c, & \forall t \\
C_m = \alpha \times \max_t\left(P_g(t) + \sum_{i=1}^{N} P_d^{(i)}(t) - \sum_{i=1}^{N} P_c^{(i)}(t)\right) &
\end{cases} \tag{31}
$$

6.3.2 模型求解

针对问题三中建立的混合整数规划模型，求解的目标是找到最优的储能调度策略，使得储能系统在满足实际负荷需求的前提下，实现峰谷套利收益最大化，同时合理控制需量电价成本。我们采用混合整数规划的方法对模型进行优化, 最终通过遗传算法对最大收益进行求解，求解结果如下：

表 8: 基于实际数据的三并柜收益情况

储能设备	需量电价/元	峰谷套利收益/元	总收益/元	收益提升/元
单柜	29195.66	327.28	247.66	74.66
双并柜	34983.84	752.02	647.07	125.41
三并柜	40729.42	1136.56	1025.49	205.01

此外，我们通过问题一求得的三井柜未来一天的负荷预测数据对模型重新求解，求解结果如下：

表 9: 基于预测数据的三并柜收益情况

储能设备	需量电价/元	峰谷套利收益/元	总收益/元	收益提升/元
单柜	22543.16	340.73	272.94	75.46
双并柜	35434.81	776.68	670.11	148.45
三并柜	34389.52	1179.15	1075.72	244.36

为了使总收益最大化，我们制定了如下的储能调整策略，以单柜为例，部分时间储能时间安排如下所示：

表 10: 未来一天的单柜部分储能策略

时间	储能	时间	储能
2024-03-18 11:00:00	99.32462841	2024-03-18 11:00:15	99.1788357
2024-03-18 11:00:30	99.38667603	2024-03-18 11:00:45	101.29229224
2024-03-18 11:01:00	100.98772975	2024-03-18 11:01:15	100.23109933
2024-03-18 11:01:30	101.11746334	2024-03-18 11:01:45	98.92691602
2024-03-18 11:02:00	100.0278739	2024-03-18 11:02:15	101.35468309
2024-03-18 11:02:30	100.87732193	2024-03-18 11:02:45	97.77125734
2024-03-18 11:03:00	99.42215478	2024-03-18 11:03:15	97.61442286
2024-03-18 11:03:30	101.33841668	2024-03-18 11:03:45	98.00426891
⋮	⋮	⋮	⋮
2024-03-18 12:59:45	96.20339222	2024-03-18 13:00:00	100.36531192

七　模型的优缺点及推广

7.1　模型的优点

1. 模型不仅考虑了峰谷套利收益的最大化，还有效地将需量电价成本的最小化纳入优化目标，通过混合整数规划模型实现了多目标的综合优化。

2. 模型能够灵活应对不同的充放电策略及最大需量的限制变化，尤其在假设最大需量可以超过合同最大需量的情况下，模型仍能有效平衡收益与成本，确保经济效益的最大化。

3. 模型在优化过程中严格遵守了储能系统的功率限制、容量限制、以及充放电互斥等关键约束条件，确保了模型解的实际可操作性和可靠性。

4. 通过遗传算法进行求解，能够在高维复杂搜索空间中迅速收敛，找到接近最优的解决方案，适用于时间紧迫且解空间较大的实际应用场景。

5. 模型结构清晰，可以根据不同的储能设备和电力系统需求进行调整和扩展，适应不同规模和条件下的储能调度问题。

7.2　模型的缺点

1. 模型的求解结果依赖于多个参数的准确性，如电价、储能系统的效率以及最大需量的设定值等。如果这些参数在实际应用中发生偏差，可能会影响优化结果的有效性。

2. 由于模型涉及多个约束条件和大量时间段的数据，求解过程可能会耗费较多的计算资源，尤其是在处理大规模储能系统或长时间预测时，计算复杂度进一步增加。

3. 虽然模型对不同的储能设备和策略具有一定的适应性，但在面对突发性电力需求或电价剧烈波动时，模型可能难以快速调整，缺乏实时响应能力。

4. 模型主要基于确定性的输入数据进行优化，未充分考虑电力市场价格波动、设备故障或外部环境变化等不确定性因素，这可能导致优化策略在实际执行中出现偏差。

7.3 模型的推广

该优化模型具备良好的通用性，能够推广应用于更广泛的储能调度和电力优化场景。首先，模型的结构和算法不仅适用于当前的工商业储能系统，也可扩展至包括风能、太阳能等在内的多能源系统中，实现综合能源管理。其次，模型中的混合整数规划方法和遗传算法求解策略，可以适应不同规模的储能系统和复杂的电力市场环境，从而为多种用电场景下的经济效益最大化提供优化支持。最后，随着电力市场的发展和储能技术的进步，模型还可以通过引入更多动态参数和不确定性因素，进一步增强对未来电力系统的适应能力，使其在智能电网和分布式能源管理领域得到更广泛的应用。

参考文献

[1] 杨丽娟, 张白桦, 叶旭桢. 快速傅里叶变换 FFT 及其应用 [J]. 光电工程, 2004, 31(S1):1-3.

[2] 张明佳. 混合整数规划方法的工程应用研究 [D]. 华中科技大学,2005.

[3] 吉根林. 遗传算法研究综述 [J]. 计算机应用与软件, 2004, 21(2):5.

附录 1 问题一代码

```
%数据预处理
import pandas as pd
import numpy as np

df_his = pd.read_csv('./历史用电数据_三并柜.csv')

df_his['date'] = pd.to_datetime(df_his['date'])

start_date = pd.to_datetime('2023-5-21')
end_date = df_his['date'].max()
all_dates = pd.date_range(start=start_date, end=end_date)

missing_dates = all_dates.difference(df_his['date'])
available_dates = all_dates.intersection(df_his['date'])
print("缺失的日期:")
print(missing_dates)

time_frames = []
for date in available_dates:
    start_time = pd.to_datetime(date)
    end_time = pd.to_datetime(date) + pd.DateOffset(days=1) - pd.Timedelta(seconds=15)
    time_frame = pd.date_range(start=start_time, end=end_time, freq='15s')
    time_frames.append(time_frame)

full_time_series = pd.concat([pd.Series(times) for times in time_frames]).reset_index(drop=True)
print(full_time_series)
df_expand = df_his['data'].str.split(',', expand=True)

df = pd.DataFrame()
df.index.name = 'Index'
df['Time'] = np.random.random(518400)
df['CN'] = np.random.random(518400)
df['SD'] = np.random.random(518400)
print(df)
df_transposed_CN = pd.DataFrame()
df_transposed_SD = pd.DataFrame()
for i in range(270):
    if i % 3 == 1:
        df_transpose = df_expand.loc[i].transpose()
        df_transposed_CN = pd.concat((df_transposed_CN, df_transpose), ignore_index=True)
    if i % 3 == 2:
        df_transpose = df_expand.loc[i].transpose()
        df_transposed_SD = pd.concat((df_transposed_SD, df_transpose), ignore_index=True)

```

```
df.loc[:, 'Time'] = full_time_series.values
df.loc[:, 'CN'] = df_transposed_CN.values
df.loc[:, 'SD'] = df_transposed_SD.values

print(df)
df.to_excel('./origin_data_three.xlsx')


%绘图
import pandas as pd
import matplotlib.pyplot as plt

# 读取 Excel 文件
file_path = '预测结果_单柜_2024-03-18.xlsx' # 将此路径替换为你的文件路径
df = pd.read_excel(file_path)

# 假设真实值和预测值列名为 '真实值' 和 '预测值'
real_values = df['FH']
predicted_values = df['FH_预测']

# 绘制图表
plt.figure(figsize=(12, 6))
plt.plot(real_values, label='real', color='blue', linewidth=2)
plt.plot(predicted_values, label='predict', color='red', linestyle='--', linewidth=2)
plt.xlabel('time')
plt.ylabel('data')
plt.legend()
plt.grid(True)
plt.show()


%FFT预测
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# 1. 数据加载和预处理
df = pd.read_excel('./origin_data_two.xlsx')
df = df.interpolate(method='linear')
print(df.head())

data_CN = df['CN'].values[270720:] # 599040 639360 633600 *627840 *668160 *673920
data_SD = df['SD'].values[270720:] # 599040 639360 633600 *627840 *668160 *673920

# 460800 *455040
```

```
# *270720
# 归一化数据
scaler_CN = MinMaxScaler(feature_range=(0, 1))
data_CN = scaler_CN.fit_transform(data_CN.reshape(-1, 1)).flatten()

scaler_SD = MinMaxScaler(feature_range=(0, 1))
data_SD = scaler_SD.fit_transform(data_SD.reshape(-1, 1)).flatten()

# 2. 快速傅里叶变换（FFT）分析
def apply_fft(data, sampling_rate=1):
    fft_result = np.fft.fft(data)
    fft_freq = np.fft.fftfreq(len(data), d=sampling_rate)

    # 只保留正频率分量
    positive_freqs = fft_freq[np.where(fft_freq >= 0)]
    positive_fft_result = np.abs(fft_result[np.where(fft_freq >= 0)])

    return positive_freqs, positive_fft_result, fft_result

# 设置采样率（每15秒一个数据点）
sampling_rate = 1 / 15

# 对 CN 数据进行 FFT 分析
freqs_CN, fft_values_CN, fft_result_CN = apply_fft(data_CN, sampling_rate)

# 对 SD 数据进行 FFT 分析
freqs_SD, fft_values_SD, fft_result_SD = apply_fft(data_SD, sampling_rate)

# 3. 信号重构
def reconstruct_signal(fft_result, num_components=5760):
    indices = np.argsort(np.abs(fft_result))[-num_components:]
    filtered_fft = np.zeros_like(fft_result)
    filtered_fft[indices] = fft_result[indices]
    return np.fft.ifft(filtered_fft).real

# 重构 CN 和 SD 信号
reconstructed_cn_signal = reconstruct_signal(fft_result_CN)
reconstructed_sd_signal = reconstruct_signal(fft_result_SD)

# 4. 预测未来5760个数据点
# 获取重构信号的长度和周期
signal_length = len(reconstructed_cn_signal)
predicted_cn_signal = np.tile(reconstructed_cn_signal, 5760 // signal_length + 1)[:5760]
predicted_sd_signal = np.tile(reconstructed_sd_signal, 5760 // signal_length + 1)[:5760]

# 反归一化数据
```

```
predicted_cn_signal = scaler_CN.inverse_transform(predicted_cn_signal.reshape(-1, 1)).flatten()
predicted_sd_signal = scaler_SD.inverse_transform(predicted_sd_signal.reshape(-1, 1)).flatten()

# 5. 保存预测数据到 Excel 文件
predicted_data = pd.DataFrame({
    'Predicted_CN': predicted_cn_signal,
    'Predicted_SD': predicted_sd_signal
})

predicted_data.to_excel('predicted_results_fft.xlsx', index=False)
print("预测结果已保存到 predicted_results_fft.xlsx 文件中")

# 6. 可视化预测结果
plt.figure(figsize=(14, 5))
plt.plot(predicted_cn_signal, label='Predicted CN Signal')
plt.xlabel('Time (15s intervals)')
plt.ylabel('Load')
plt.legend()
plt.show()

plt.figure(figsize=(14, 5))
plt.plot(predicted_sd_signal, label='Predicted SD Signal')
plt.xlabel('Time (15s intervals)')
plt.ylabel('Load')
plt.legend()
plt.show()
```

附录 2 问题二代码

```
import pandas as pd
import numpy as np
from math import floor
import time
import matplotlib.pyplot as plt


class GA(object):
    def __init__(self, FH,
                 maxgen=2000,
                 size_pop=1000,
                 cross_prob=0.80,
                 pmuta_prob=0.02,
                 select_prob=0.8):
        self.maxgen = maxgen # 最大迭代次数
        self.size_pop = size_pop # 群体个数
        self.cross_prob = cross_prob # 交叉概率
        self.pmuta_prob = pmuta_prob # 变异概率
        self.select_prob = select_prob # 选择概率

        self.FH = FH # 城市的左边数据
        self.num = 480 # 城市个数 对应染色体长度

        # 通过选择概率确定子代的选择个数
        self.select_num = max(floor(self.size_pop * self.select_prob + 0.5), 2)

        # 父代和子代群体的初始化（不直接用np.zeros是为了保证单个染色体的编码为整数，np.zeros对应的数据类型为浮点型）
        self.chrom = np.zeros(self.size_pop * self.num).reshape(self.size_pop,
                                                                 self.num) # 父 print(chrom.shape)(200, 14)
        self.sub_sel = np.zeros(self.select_num * self.num).reshape(self.select_num, self.num) # 子 (160, 14)

        # 存储群体中每个染色体的路径总长度，对应单个染色体的适应度就是其倒数 #print(fitness.shape)#(200,)
        self.fitness = np.zeros(self.size_pop)

        self.best_fit = [] ##最优距离
        self.best_path = [] # 最优路径

    def rand_chrom(self):
        for i in range(self.size_pop):
            rand_ch = np.zeros(self.num)
            for j in range(self.num):
                random_number = np.random.uniform(0, 101.4)
                rand_ch[j] = random_number
```

```
            self.chrom[i, :] = rand_ch
            self.fitness[i] = self.comp_fit(rand_ch)

    def comp_fit(self, CN):
        res = 177.5255
        SD = self.FH + CN
        max = 552.94
        for i in range(480 - 15):
            demand = 0
            for j in range(i, i + 15):
                demand += SD[j]
            demand /= 15
            if demand > max:
                max = demand

        charge = np.sum(CN) / 240
        res = res - 0.003 * 48 * max / 1.05 + charge * 0.8257
        return res

    def select_sub(self):
        fit = self.fitness # 适应度函数
        cumsum_fit = np.cumsum(fit) # 累积求和 a = np.array([1,2,3]) b = np.cumsum(a) b=1 3 6
        pick = cumsum_fit[-1] / self.select_num * (
                np.random.rand() + np.array(range(int(self.select_num)))) # select_num 为子代选择个数 160
        i, j = 0, 0
        index = []
        while i < self.size_pop and j < self.select_num:
            if cumsum_fit[i] >= pick[j]:
                index.append(i)
                j += 1
            else:
                i += 1
        self.sub_sel = self.chrom[index, :]

    def cross_sub(self):
        if self.select_num % 2 == 0: # select_num160
            num = range(0, int(self.select_num), 2)
        else:
            num = range(0, int(self.select_num + 1), 2)
        for i in num:
            if self.cross_prob >= np.random.rand():
                self.sub_sel[i, :], self.sub_sel[i + 1, :] = self.intercross(self.sub_sel[i, :], self.sub_sel[i
                    + 1, :])

    def intercross(self, ind_a, ind_b):
        r1 = np.random.randint(self.num)
```

```
        r2 = np.random.randint(self.num)
        while r2 == r1: # 如果r1==r2
            r2 = np.random.randint(self.num) # r2重新生成
        left, right = min(r1, r2), max(r1, r2) # left 为r1,r2小值 , r2为大值
        ind_a1 = ind_a.copy() # 父亲
        ind_b1 = ind_b.copy() # 母亲
        for i in range(left, right + 1):
            ind_a2 = ind_a.copy()
            ind_b2 = ind_b.copy()
            ind_a[i] = ind_b1[i] # 交叉 （即ind_a （1,14） 中有个元素 和ind_b互换
            ind_b[i] = ind_a1[i]
            x = np.argwhere(ind_a == ind_a[i])
            y = np.argwhere(ind_b == ind_b[i])
            if len(x) == 2:
                ind_a[x[x != i]] = ind_a2[i] # 查找ind_a 中元素=- ind_a[i] 的索引
            if len(y) == 2:
                ind_b[y[y != i]] = ind_b2[i]
        return ind_a, ind_b

    # 变异模块 在变异概率的控制下，对单个染色体随机交换两个点的位置。
    def mutation_sub(self):
        for i in range(int(self.select_num)): # 遍历每一个 选择的子代
            if np.random.rand() <= self.pmuta_prob: # 如果随机数小于变异概率
                r1 = np.random.randint(self.num) # 随机生成小于num==可设置 的数
                r2 = np.random.randint(self.num)
                while r2 == r1: # 如果相同
                    r2 = np.random.randint(self.num) # r2再生成一次
                self.sub_sel[i, [r1, r2]] = self.sub_sel[i, [r2, r1]] # 随机交换两个点的位置。

    # 进化逆转 将选择的染色体随机选择两个位置r1:r2 ， 将 r1:r2 的元素翻转为 r2:r1 ，如果翻转后的适应度更高，则替换
        原染色体，否则不变
    def reverse_sub(self):
        for i in range(int(self.select_num)): # 遍历每一个 选择的子代
            r1 = np.random.randint(self.num) # 随机生成小于num==14 的数
            r2 = np.random.randint(self.num)
            while r2 == r1: # 如果相同
                r2 = np.random.randint(self.num) # r2再生成一次
            left, right = min(r1, r2), max(r1, r2) # left取r1 r2中小值，r2取大值
            sel = self.sub_sel[i, :].copy() # sel 为父辈染色体 shape=（1,14）

            sel[left:right + 1] = self.sub_sel[i, left:right + 1][::-1] # 将染色体中(r1:r2)片段 翻转为（r2:r1)
            if self.comp_fit(sel) > self.comp_fit(self.sub_sel[i, :]): # 如果翻转后的适应度小于原染色体，则不变
                self.sub_sel[i, :] = sel

    # 子代插入父代，得到相同规模的新群体
    def reins(self):
```

```
        index = np.argsort(self.fitness)[::] # 替换最差的（倒序）
        self.chrom[index[:self.select_num], :] = self.sub_sel

    def info(self, CN):
        res = 177.5255
        SD = self.FH + CN
        max = 552.94
        for i in range(480 - 15):
            demand = 0
            for j in range(i, i + 15):
                demand += SD[j]
            demand /= 15
            if demand > max:
                max = demand

        charge = np.sum(CN) / 240
        res = res - 0.003 * 48 * max / 1.05 + charge * 0.8257
        demand_price = 48 * max / 1.05
        charge_price = charge * 0.8257
        return demand_price, charge_price

if __name__ == "__main__":
    df = pd.read_excel('./预测结果_单柜_2024-03-18.xlsx')
    FH = df['FH'].values[2640:3120]
    print(FH)
    module = GA(FH)
    module.rand_chrom()
    for i in range(module.maxgen):
        module.select_sub()
        module.cross_sub()
        module.mutation_sub()
        module.reverse_sub()
        module.reins()

        for j in range(module.size_pop):
            module.fitness[j] = module.comp_fit(module.chrom[j])

        index = module.fitness.argmax()
        if (i + 1) % 10 == 0:
            print('第' + str(i + 1) + '代后的最短的路程: ' + str(module.fitness[index]))
            print('第' + str(i + 1) + '代后的最优路径:')
            print(module.chrom[index])
            demand_price, charge_price = module.info(module.chrom[index])
            print(demand_price)
            print(charge_price)

```

```
180         module.best_fit.append(module.fitness[index])
181         module.best_path.append(module.chrom[index])
182 
183     best = module.chrom[0]
184     print(best)
185     demand_price, charge_price = module.info(best)
186     print(demand_price)
187     print(charge_price)
```

附录 3 问题三代码

```
import pandas as pd
import numpy as np
from math import floor
import time
import matplotlib.pyplot as plt


class GA(object):
def __init__(self, FH,
             maxgen=1000,
             size_pop=500,
             cross_prob=0.80,
             pmuta_prob=0.02,
             select_prob=0.8):
    self.maxgen = maxgen # 最大迭代次数
    self.size_pop = size_pop # 群体个数
    self.cross_prob = cross_prob # 交叉概率
    self.pmuta_prob = pmuta_prob # 变异概率
    self.select_prob = select_prob # 选择概率

    self.FH = FH # 城市的左边数据
    self.num = 480 # 城市个数 对应染色体长度

    # 通过选择概率确定子代的选择个数
    self.select_num = max(floor(self.size_pop * self.select_prob + 0.5), 2)

    # 父代和子代群体的初始化（不直接用np.zeros是为了保证单个染色体的编码为整数，np.zeros对应的数据类型为浮点型）
    self.chrom = np.zeros(self.size_pop * self.num).reshape(self.size_pop,
                                                            self.num) # 父 print(chrom.shape)(200, 14)
    self.sub_sel = np.zeros(self.select_num * self.num).reshape(self.select_num, self.num) # 子 (160, 14)

    # 存储群体中每个染色体的路径总长度，对应单个染色体的适应度就是其倒数 #print(fitness.shape)#(200,)
    self.fitness = np.zeros(self.size_pop)

    self.best_fit = [] ##最优距离
    self.best_path = [] # 最优路径

def rand_chrom(self):
    for i in range(self.size_pop):
        rand_ch = np.zeros(self.num)
        for j in range(self.num):
            random_number = np.random.uniform(0, 101.4)
            rand_ch[j] = random_number
        self.chrom[i, :] = rand_ch
```

```
        self.fitness[i] = self.comp_fit(rand_ch)

def comp_fit(self, CN):
    fixed = 177.5255
    SD = self.FH + CN
    max = 580.587
    for i in range(480 - 15):
        demand = 0
        for j in range(i, i + 15):
            demand += SD[j]
        demand /= 15
        if demand > max:
            max = demand

    charge = np.sum(CN) / 240
    res = fixed - 0.003 * 48 * max / 1.05 + charge * 0.8257
    for i in range(floor(max) - 100):
        current = fixed - 0.003 * 48 * i / 1.05 - 0.003 * 96 * (max - i / 1.05) + charge * 0.8257
        if current > res:
            res = current
    return res

def select_sub(self):
    fit = self.fitness # 适应度函数
    cumsum_fit = np.cumsum(fit) # 累积求和 a = np.array([1,2,3]) b = np.cumsum(a) b=1 3 6
    pick = cumsum_fit[-1] / self.select_num * (
           np.random.rand() + np.array(range(int(self.select_num)))) # select_num 为子代选择个数 160
    i, j = 0, 0
    index = []
    while i < self.size_pop and j < self.select_num:
        if cumsum_fit[i] >= pick[j]:
            index.append(i)
            j += 1
        else:
            i += 1
    self.sub_sel = self.chrom[index, :]

def cross_sub(self):
    if self.select_num % 2 == 0: # select_num160
        num = range(0, int(self.select_num), 2)
    else:
        num = range(0, int(self.select_num + 1), 2)
    for i in num:
        if self.cross_prob >= np.random.rand():
            self.sub_sel[i, :], self.sub_sel[i + 1, :] = self.intercross(self.sub_sel[i, :], self.sub_sel[i +
                1, :])
```

```
90
91  def intercross(self, ind_a, ind_b):
92      r1 = np.random.randint(self.num)
93      r2 = np.random.randint(self.num)
94      while r2 == r1: # 如果r1==r2
95          r2 = np.random.randint(self.num) # r2重新生成
96      left, right = min(r1, r2), max(r1, r2) # left 为r1,r2小值 , r2为大值
97      ind_a1 = ind_a.copy() # 父亲
98      ind_b1 = ind_b.copy() # 母亲
99      for i in range(left, right + 1):
100         ind_a2 = ind_a.copy()
101         ind_b2 = ind_b.copy()
102         ind_a[i] = ind_b1[i] # 交叉 （即ind_a （1,14） 中有个元素 和ind_b互换
103         ind_b[i] = ind_a1[i]
104         x = np.argwhere(ind_a == ind_a[i])
105         y = np.argwhere(ind_b == ind_b[i])
106         if len(x) == 2:
107             ind_a[x[x != i]] = ind_a2[i] # 查找ind_a 中元素=- ind_a[i] 的索引
108         if len(y) == 2:
109             ind_b[y[y != i]] = ind_b2[i]
110     return ind_a, ind_b
111
112 # 变异模块 在变异概率的控制下，对单个染色体随机交换两个点的位置。
113 def mutation_sub(self):
114     for i in range(int(self.select_num)): # 遍历每一个 选择的子代
115         if np.random.rand() <= self.pmuta_prob: # 如果随机数小于变异概率
116             r1 = np.random.randint(self.num) # 随机生成小于num==可设置 的数
117             r2 = np.random.randint(self.num)
118             while r2 == r1: # 如果相同
119                 r2 = np.random.randint(self.num) # r2再生成一次
120             self.sub_sel[i, [r1, r2]] = self.sub_sel[i, [r2, r1]] # 随机交换两个点的位置。
121
122 # 进化逆转 将选择的染色体随机选择两个位置r1:r2 ， 将 r1:r2 的元素翻转为 r2:r1 ， 如果翻转后的适应度更高，则替换原染
        色体，否则不变
123 def reverse_sub(self):
124     for i in range(int(self.select_num)): # 遍历每一个 选择的子代
125         r1 = np.random.randint(self.num) # 随机生成小于num==14 的数
126         r2 = np.random.randint(self.num)
127         while r2 == r1: # 如果相同
128             r2 = np.random.randint(self.num) # r2再生成一次
129         left, right = min(r1, r2), max(r1, r2) # left取r1 r2中小值，r2取大值
130         sel = self.sub_sel[i, :].copy() # sel 为父辈染色体 shape=（1,14）
131
132         sel[left:right + 1] = self.sub_sel[i, left:right + 1][::-1] # 将染色体中(r1:r2)片段 翻转为（r2:r1)
133         if self.comp_fit(sel) > self.comp_fit(self.sub_sel[i, :]): # 如果翻转后的适应度小于原染色体，则不变
134             self.sub_sel[i, :] = sel
```

```
135
136 # 子代插入父代，得到相同规模的新群体
137 def reins(self):
138     index = np.argsort(self.fitness)[::] # 替换最差的（倒序）
139     self.chrom[index[:self.select_num], :] = self.sub_sel
140
141 def info(self, CN):
142     fixed = 177.5255
143     SD = self.FH + CN
144     max = 552.94
145     for i in range(480 - 15):
146         demand = 0
147         for j in range(i, i + 15):
148             demand += SD[j]
149         demand /= 15
150         if demand > max:
151             max = demand
152
153     charge = np.sum(CN) / 240
154     res = fixed - 0.003 * 48 * max / 1.05 + charge * 0.8257
155     flag = max
156     for i in range(floor(max) - 100):
157         current = fixed - 0.003 * 48 * i / 1.05 - 0.003 * 96 * (max - i / 1.05) + charge * 0.8257
158         if current > res:
159             res = current
160             flag = i
161     demand_price = 48 * flag / 1.05 + 96 * (max - flag / 1.05)
162     charge_price = charge * 0.8257 + fixed
163     return demand_price, charge_price
164
165 if __name__ == "__main__":
166 df = pd.read_excel('./预测结果_单柜_2024-03-18.xlsx')
167 FH = df['FH'].values[2640:3120]
168 print(FH)
169 module = GA(FH)
170 module.rand_chrom()
171 for i in range(module.maxgen):
172     module.select_sub()
173     module.cross_sub()
174     module.mutation_sub()
175     module.reverse_sub()
176     module.reins()
177
178     for j in range(module.size_pop):
179         module.fitness[j] = module.comp_fit(module.chrom[j])
180
```

```
    index = module.fitness.argmax()
    if (i + 1) % 10 == 0:
        print('第' + str(i + 1) + '代后的最短的路程: ' + str(module.fitness[index]))
        print('第' + str(i + 1) + '代后的最优路径:')
        print(module.chrom[index])
        demand_price, charge_price = module.info(module.chrom[index])
        print(demand_price)
        print(charge_price)

    module.best_fit.append(module.fitness[index])
    module.best_path.append(module.chrom[index])

best = module.chrom[0]
print(best)
demand_price, charge_price = module.info(best)
print(demand_price)
print(charge_price)
```