

基于多时段购电优化与自进化预测的微网电力调控策略

摘要

本文的模型设计、算法与实验设计、核心猜想及论文结构均由参赛队员完成；题意与文献核对、代码实现与调试、批量实验执行、结果汇总与文字组织由参赛队员与 AI 协作完成，AI 参与完成的全部结果与过程均经参赛队员监督审查，不存在未经人工审核即采用的内容。

微网与外网协同调控的实质，是在储能既不完美、电又必须实时平衡的条件下，用一套跨时段的购电决策替换另一套更贵的决策。本题四问层层递进，共同的困难是：决策必须在当天 0:00 做出，而当天会发生什么尚不可知——能否把电价、负载与光伏预测得足够接近实际值，直接决定了购电费用的高低，这是贯穿四问的主线。本文不局限于题目所问，另设计了一个可复现、可进化的预测框架，用以回答“这套方法换一个场景还能不能用”。

问题一在电价、负载、光伏全天已知时，建模为含储能的多时段线性规划，直接求得全局最优：全天购电费 **35 126.95 元**，全天购电量 59 482.6990 kWh。若取消储能，全天购电费升至 **48 052.05 元**，即储能使费用下降 **26.90%**（12 925.10 元），且该收益全部来自把购电搬到低价时段的套利，而非少买电；另用储电量离散化动态规划独立复算，两条路径互差 2.26 元。

问题二引入预测不确定性。以净负荷为预测对象，只用当天 0:00 之前已经发生的数据预测，不含任何未来信息；用指数平滑与星期基线的加权和给出点预测，由报童模型给出冗余分位的方向性依据，四个参数全部在 1 月窗口上网格搜索标定。全年费用 **13 700 048.95 元**，其中紧急购电 665 904.27 元、占 4.86%，距完美信息下界 12.05%。

问题三允许依据 6:00、12:00、18:00 新发布的预报调整购电策略。同一目标时段常有多发布时刻不同的预报版本，本文按“发布时刻不晚于决策时刻”取版，用各版本历史残差按“目标小时 × 提前量”分档估计方差并作逆方差加权融合；把计划费用、违约与超发费用、紧急购电费用三项一并写入优化目标，使每次调整都在“改得更准”与“少改少罚”之间自动权衡。全年费用降至 **14 248 107.47 元**；引入调整使费用再降 67 279.55 元，紧急购电量下降 26.9%。

问题四（第二问）中电价本身波动且题目未给任何电价预报，电价也成为待预测量，待预测的范围扩大到负载、光伏、电价三个量。本文把标定协议改为扩展窗逐日 / 逐月重标定：第 d 天的参数只用第 $0 \sim d-1$ 天的数据重新确定，模型随运行自行进化。全年费用为 **14 995 821.96 元**。

问题四（第三问）在第二问之上再叠加 6:00、12:00、18:00 三次调整，全年费用为 **14 885 452.71 元**；三次调整使该问较第二问再降 110 369 元、紧急购电量降 14.8%。与完美信息下界 12 779 221.24 元相比，第二问尚余 17.3%，该空间主要与预测误差和冗余裕量有关，**提高预测精度是最直接的改进方向**。

在此基础上另建的可迁移、自进化预测框架，把序列分解为可搬运的形状层、必须现场学习的水平层与由物理规律支配的光伏天文层，以收缩估计分配权重。数据边界须如实说明：现有资料只有主站点一年真实数据，故变体站点由**从原始数据反推的生成机制**合成（参数取自原始数据的统计结构，天气实现由固定随机种子重采样产生），结论因此严格限定为“**该框架的机制有效**”——同站点第二年三变量精度提升 6%~9%、第 0 天即具备 3%~8% 的零样本精度、换地点后光伏预测 2 天收敛，而非“在任意真实站点上都能达到该精度”。

全部结论由三组独立证据支撑：把统计期起点之后的实际值置零后重跑，标定常数与逐段决策逐位不变；重算费用与求解器输出逐位一致；问题一、二各有两条互不共用代码的求解路径互相印证。

关键词：电力调度；多时段购电优化；线性规划；报童模型；预报融合；自进化预测

引言

微网与外网的协同调控，是在“电能不能储存、储能又不完美”这一前提下，用一套跨时段的购电决策替换另一套更贵的购电决策。峰谷价差给定了套利的空间，而光伏的不可控性与负荷的随机性给定了套利的风险：计划做在 0:00，当天的实际发生却要等到 24:00 才完全揭晓。

题目四问层层递进：问题一在全部信息已知时求确定性最优；问题二引入“负载与光伏需要预测”；问题三允许依照新发布的预报在日内修订；问题四再把电价本身也变成待预测量。四问的差别表面上在数据，实质上在**信息边界**——什么时刻能知道什么。

本文的处理方式是把四问收敛到同一个优化内核，只在三处做替换：电价序列、信息的可得性、计费的规则。这样写一次求解器即可覆盖四问，且每一层都能被单独验证。全文对“某个量在哪一刻可知”逐处显式声明，并给出只用第一个月标定的可部署方案。

问题重述

1.1 四个问题的任务

问题一在电价、小区负载、光伏预测功率全天已知的条件下，制定 0:00 的计划购电策略，要求供电量不低于负载，且储能能在 0:00 与 24:00 的储电量相同。

问题二保持电价逐日不变，但负载与光伏逐时变化；供电不足时按当时电价的 5 倍紧急购电，其余时段按计划购电量计费。要求在 0:00 制定当天计划。

问题三增加每天 0:00、6:00、12:00、18:00 发布的光伏预报，允许据其调整计划；计划高于调整的部分按 50% 计费，调整高于计划的部分按 150% 计费，未供足部分仍按 5 倍紧急购电。要求给出调整策略，并分析是否需要引入其他时刻的预报。

问题四把电价换成逐日逐时波动的情形，重新计算问题二与问题三（分别记为 4-2 与 4-3）。

四问的结果须分别保存为 result1、result2、result3、result4-2、result4-3，其中后四者只需填写 2025 年 2 月 1 日至 12 月 31 日，共 334 天。

1.2 求解思路

四问共用同一个含储能的多时段购电优化内核，差异只体现在**电价序列**、

信息的可得性与**计费规则**三处。这一结构使四问的结果具有可比性：相邻两问的结果可用于比较新增的信息条件与结算规则所带来的综合影响。

与优化本身相比，本文认为更值得投入的是**信息边界的处理**。题目提供的附件 2、附件 4 是实际发生值，而计划只能依据决策时刻已经可得的信息做出，二者混用即构成数据泄露。本文把计划层与结算层严格分离，并在第 8 章给出可执行的泄露检验。

数据画像与问题分析

2.1 附件说明与数据总览

2.1.1 五个附件的规模与用途

表 1 五个附件的规模与用途

附件	形式	规模	内容	用于
附件 1	单表	144 段 × 4 列（单日）	某一天的 10 分钟级电价、小区负载、光伏发电 预测 功率	问题一、二、三的电价
附件 2	双表	365 天 × 144 段 × 2	2025 全年小区负载与光伏发电 实际 功率	问题二、三、四的负荷与光伏

附件	形式	规模	内容	用于
附件 3	单表	365 天 × 4 个发布时刻 × 24 小时	每天 0:00、6:00、12:00、18:00 发布的未来 24 小时整点光伏功率预报	问题三、四的光伏预报
附件 4	单表	365 天 × 144 段	2025 全年逐 10 分钟电价	仅问题四
附件 5	5 个结果模板	—	result1 ~ result4-3 的填写格式	全部结果文件

时间粒度统一为 10 分钟，每天 144 段；全年按 365 天计（2025-01-01 至 2025-12-31）。

2.1.2 各附件的具体形态

附件 1 给出"某一天"的完整日内曲线，列名为"时间 / 电价 / 小区负载 / 光伏发电预测功率"，末行时间标记为"0:00+1"，即次日 0:00，亦即当天 24:00。**逐段核对可知，该"典型日"的三列恰为全年的逐段均值：**负载列与附件 2 的逐段年均最大差 0.00005 kW、电价列与附件 4 的逐段年均最大差 0.000052 元/kWh、光伏列与附件 2 光伏的逐段年均最大差 0.04 kW，均在四舍五入误差量级。故附件 1 是"日均曲线"而非某一天的真实记录，这正是题面"若每天的电价和小区负载相同"的取值来源；它为问题一、二、三提供电价。

附件 2 以"日期 × 时间"的矩阵形式给出全年实际值。须特别注意其光伏一列是**实际出力**，而问题二、三的计划必须在当天 0:00 制定——二者不可混用，这是后文全部信息边界处理的起点。

附件 3 按"日期 / 预报时刻 / 预报 1 小时……预报 24 小时"组织，同一天有 4 行。这意味着**同一目标时段往往存在多个不同发布时刻的预报版本**：例如第 d 天 08:00 这一小时，既出现在第 $d-1$ 天 18:00 版的"预报 14 小时"栏，也出现在第 d 天 0:00 版的"预报 8 小时"栏。各版本的准确度并不相同，这正是问题三"可根据其他时刻的预报调整购电策略"的着力点。

附件 4 与附件 2 同形，给出全年逐 10 分钟电价。其日内形态与附件 1 的典型日电价曲线同属六段式峰谷结构，但**逐日水平与逐段取值均不相同**。题目未对附件 4 附加任何"预报"说明，这一点在问题四中具有决定性意义：**电价本身也成为待预测量**。

附件 5 的模板中，"计划购电量"表的表头相对数据右移一格（首列表头为"日期\时间"，其后才是时刻），填写时数据须自第 2 列起按时间顺序排列。本文在生成结果文件时逐字保留模板表头。

2.1.3 原始数据与二次分析结果的界线

本节与第 6 章会反复引用若干统计量，此处明确区分两类来源。**直接取自附件的原始量：**逐 10 分钟的电价、小区负载、光伏实际功率与光伏预报功率，以及由它们求和或求均值得到的日总量与全天购电量；储能参数（12000 kWh / 5000 kW / 1200~10800 kWh / 单向效率 90% / 初始 6000 kWh）。**本文二次分析得到的量：**日型分组（§2.2.2）、逐月与季节水平（§2.2.3）、太阳黄经与二十四节气时刻（§2.2.4）、预报残差及其"目标小时 × 提前量"分档（§2.2.8、§6.4）、冗余量分位（§6.3）、理论下界（§6.5）与变体站点数据（§7）。凡属第二类者，本文在首次出现处注明计算方式——因为标定协议要求人为参数只能由"统计期开始之前已完整发生的月份"确定，若把二次分析量误当作附件给定值，就会在不知不觉中引入未来信息。

2.1.4 四个指标在四问中的信息结构

同一条数据在不同问题里的"何时可知"完全不同，这是四问建模差异的根源。

表 2 四个指标在四个问题中的信息可得时点

指标	来源与发布方式	问题一	问题二	问题三	问题四
电价 λ_t	附件 1 典型日曲线（题给，全天已知）	全天已知	当天已知（每日相同）	当天已知（每日相同）	附件 4 只给 实际值 ，须由 0:00 前的历史电价外推

指标	来源与发布方式	问题一	问题二	问题三	问题四
小区负载 L_t	附件 1 典型日 / 附件 2 实际值 (当天实时才发生)	附件 1 全天已知	0:00 由历史预测	0:00 由历史预测	0:00 由历史预测
光伏出力 P_t	附件 1 典型日 / 附件 2 实际值 (当天实时才发生)	附件 1 全天已知	0:00 由历史预测	0:00 由历史预测, 6:00 / 12:00 / 18:00 用新预报修正	同问题三
光伏预报	附件 3: 每天 0:00、6:00、12:00、18:00 各发布一版, 每版覆盖发布时刻起 24 小时	不使用	不使用 (无预报)	只用 "发布时刻 $\leq$ 决策时刻" 的版本	同问题三

三条边界必须写明。其一, 附件 2、附件 4 给的是实际发生值, 在当天 0:00 决策时尚未发生, 只能作历史样本使用; 其二, 附件 3 的四个版本按发布时刻滚动到达, 0:00 的计划只能用当时已发布的版本, 否则即"偷看"当天 6:00 才发布的信息; 其三, 题目未给任何电价预报, 故问题四中电价与负载、光伏一样成为待预测量, 而日内三次调整能改善的只有光伏这一重信息。问题一的信息完备, 其解可作为后三问的参照; 问题二、三、四逐层增加不确定性来源, 每增加一重, 本文只改动对应的那一层, 优化内核保持不变, 四问结果因此可比。

2.2 三条时间序列的结构解剖

建模之前必须先弄清: 电价的波动、负载的起伏、光伏的涨落, 究竟由什么驱动。本节对三条序列做结构分解, 所得结论直接决定第 6 章各问的预测模型与标定方案。

2.2.1 基本统计

以逐日 144 段的均值作为"日水平", 全年 365 天:

表 3 三个变量的日水平统计

变量	年均值	标准差	变异系数	最小	最大	峰谷比
电价 (元 / kWh)	0.7662	0.1036	13.5%	0.5378	0.9290	1.73
小区负载 (kW)	4626.03	908.88	19.6%	2978.76	5898.56	1.98
光伏 (kW)	2311.79	481.12	20.8%	1431.69	3134.29	2.19

三者日间波动幅度相当 (13%~21%), 但波动的来源完全不同, 这正是后续必须区别对待之处。

2.2.2 星期结构：低日型恰为周五、周六

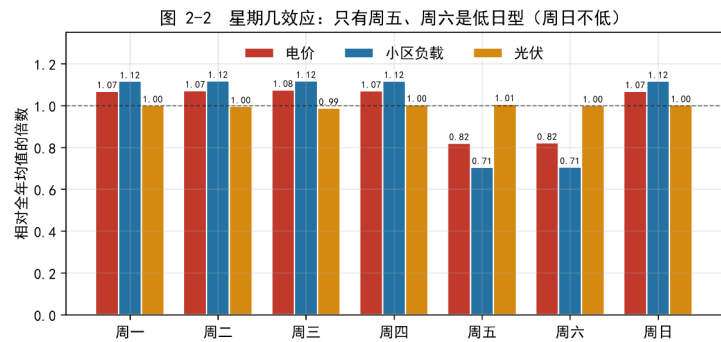


图 1 星期几效应

表 4 按星期的日均值

变量	周一	周二	周三	周四	周五	周六	周日
电 价（元 / kWh）	0.8191	0.8211	0.8238	0.8204	0.6290	0.6298	0.8191
小 区 负 载（kW）	5162.5	5172.7	5171.5	5168.0	3263.8	3267.0	5166.2
光 伏（kW）	2314.8	2303.4	2285.4	2322.1	2327.1	2313.3	2316.8

三点结论：

其一，低日型恰为周五、周六两天，周日与周一至周四同属高日型。用"是否周五 / 周六"这一个二值变量，对电价的解释力已达 $R^2=0.6953$ ，对小区负载达 $R^2=0.8930$ ；改用完整的 6 个星期哑变量，二者分别为 0.6955 与 0.8930——相差不足千分之三，说明星期结构是纯二元的。

其二，光伏与星期无关： $R^2=0.0007$ ，组间差异仅 +0.5%。若沿用负荷的日型划分去预测光伏，会引入虚假结构。

其三，该结构在全年四个季节内高度稳定。加入"星期 × 月份"线性交互项后，电价与负载的 R^2 增量均为 0.0000。分季看，周五 / 周六相对其余日型的降幅为电价 -24.5% / -23.3% / -23.9% / -22.1%、负载 -38.3% / -35.3% / -36.4% / -37.3%，是一个可以安全外推的结构（图 1）。

2.2.3 月际与季节结构

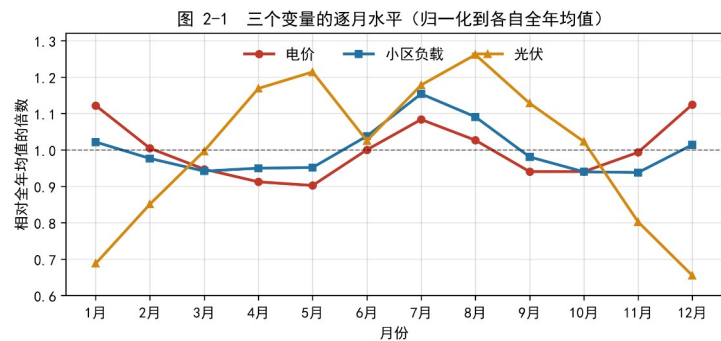


图 2 逐月水平

表 5 逐月归一化水平

变量	1月	2月	3月	4月	5月	6月	7月	8月	9月	10月	11月	12月
电价	1.122	1.004	0.947	0.912	0.902	1.000	1.083	1.026	0.940	0.940	0.993	1.125

变量	1月	2月	3月	4月	5月	6月	7月	8月	9月	10月	11月	12月
小区	1.022	0.976	0.942	0.950	0.952	1.038	1.154	1.091	0.981	0.940	0.938	1.014
负载												
光伏	0.687	0.851	0.997	1.169	1.214	1.025	1.178	1.261	1.128	1.023	0.802	0.655

电价与小区负载呈现半年周期：负载 7 月大峰、12 / 1 月次峰，3—4 月与 10—11 月双谷；电价 12 / 1 月最高、7 月次高，5 月与 9—10 月最低。两者的年周期分量都很弱——太阳黄经 1 阶谐波的解释力分别只有 0.0528 与 0.0274，2 阶跃升至 0.2640 与 0.0906。

光伏则是标准的年周期：8 月最高（1.261）、12 月最低（0.655），太阳黄经 1 阶谐波即解释 0.7653，2 阶达 0.9071（图 2）。

2.2.4 天文结构：光伏服从节气，电价与负载服从气温

本节内容的适用范围：天文层（太阳几何）只用于第 7 章额外拓展的自进化预测框架，第 6 章四问的求解并不使用它——问题三、四的光伏预报直接取自附件 3，问题一、二的光伏由历史数据自预测。本节的作用是说明"光伏与负载的季节性来源不同"这一结构事实，为第 7 章的三层分解提供依据。

图 2-4 光伏的年变化服从“节气”（天文几何），不是“日历月份”

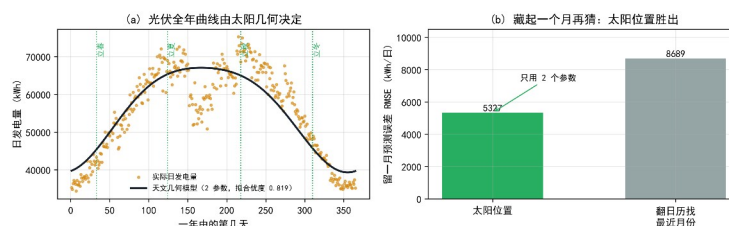


图 3 光伏与天文几何

水平面天文日辐射量为（文献 [11]）

$$H_0(\varphi, \delta) = \frac{24 G_{sc}}{\pi} \left(1 + 0.033 \cos \frac{2\pi n}{365} \right) (\cos \varphi \cos \delta \sin \omega_s + \omega_s \sin \varphi \sin \delta)$$

其中赤纬 δ 满足 $\sin \delta = \sin \varepsilon \sin \theta$ ， θ 为太阳黄经， $\varepsilon = 23.4393^\circ$ 为黄赤交角， $\omega_s = \arccos(-\tan \varphi \tan \delta)$ 为日落时角。

赤纬完全由太阳黄经（即二十四节气）决定，故 H_0 是“节气”信息的显式表达。

以 (k, φ) 两个自由参数拟合日发电量 E_d ：

$$E_d = k H_0(\varphi, \delta_d) + \varepsilon_d$$

得 $\hat{\varphi} = 23.5^\circ$ 、 $R^2 = 0.8190$ ；而“日历日序 1 阶谐波”用 3 个参数只得到 $R^2 = 0.7543$ 。

需要注意的是：单一年份内月份与太阳黄经近乎共线，因此判别“节气还是日历”不能依赖拟合优度，只能依赖外推能力。将某一个月完全留出、用其余 11 个月拟合再预测它，结果如下。

留一月交叉验证的结果是：含 5 个自由参数的太阳黄经 2 阶谐波 RMSE 4851.30、仅含 2 个参数的天文几何 5327.01、天文几何乘月均晴空指数（14 参数）5613.73、太阳黄经 1 阶谐波（3 参数）7089.16、不含参数的季节朴素法 8689.32，日发电量标准差 11546.85 kWh/日。

天文几何只用 2 个参数即显著优于不含参数的日历朴素法——日历形式对未见过的月份没有外推能力，天文形式有。日内层面同样印证：光伏逐 10 分钟曲线与天文晴空余弦形状的归一化 RMSE 仅 0.034~0.081，峰值时刻随太阳正午由 1 月 15 日的 11:58 移至 6 月 21 日的 12:35（图 3）。

作为对照，电价与小区负载用同一几何模型拟合的 R^2 分别为 -0.2957 与 -0.0592 ，均为负值，即不如直接用均值。二者的季节变化来自气温与生产节奏，与天文节气无关；这一点在把三者并列分析时极易混淆。光伏的残余部分为云量，其晴空指数月均值6月最低（0.849）、10月最高（1.105），反映了本数据集中季节性云量的变化。

2.2.5 日内结构

图 2-3 日内曲线：周五/周六把曲线整体压低（电价夜里谷更深）

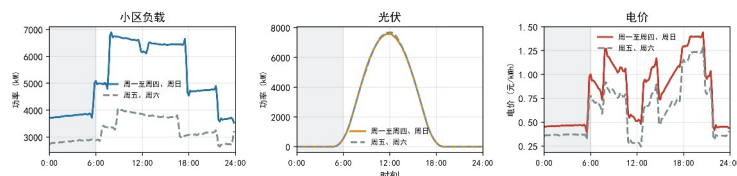


图 4 日内曲线

表 6 日内峰谷位置与峰谷比

变量	高日型（周一至周四及周日）	低日型（周五及周六）
小区负载	峰 6890.3 kW @ 8:00, 谷 3531.3 kW @ 23:50, 峰谷比 1.95	峰 4046.6 kW @ 9:00, 谷 2643.4 kW @ 22:00, 峰谷比 1.53
光伏	峰 7588.2 kW @ 12:00	峰 7678.6 kW @ 12:10（几乎不变）
电价	峰 1.4 元/kWh @ 20:30, 谷 0.4 @ 5:30, 峰谷比 3.66	峰 1.3 @ 20:30, 谷 0.2 @ 12:30, 峰谷比 5.30

此处存在一个值得注意的反差：周五 / 周六把负载曲线压平（峰谷比由 1.95 降至 1.53），却使电价曲线更尖锐（由 3.66 升至 5.30，夜间谷更深）。两类日子上储能的套利空间并不相同，这是必须逐日重解而不能套用典型日的原因之一（图 4）。

2.2.6 三个变量之间的耦合

图 2-5 光伏出力越大，电价越低（剔除共同结构后高达 -0.95 ）

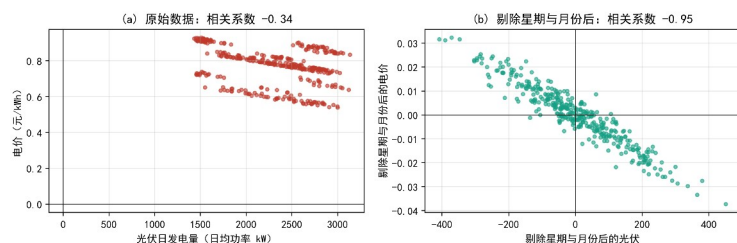


图 5 电价与光伏

表 7 三对变量的相关系数

变量对	原始相关	剔除星期结构后	剔除星期与月份后
电价 ~ 小区负载	+0.8977	+0.608	+0.2205
电价 ~ 光伏	-0.3431	-0.605	-0.9515
小区负载 ~ 光伏	+0.0689	+0.243	-0.0090

本文以 Pearson 相关系数（文献 [14]）度量两两之间的线性相关：电价与小区负载之间 $+0.90$ 的高相关是共同驱动造成的假象：二者同受星期结构与季节水平支配，剔除这两层后只剩 $+0.22$ 。这说明不可用电价反推负荷。

电价与光伏剔除共同结构后达到 -0.9515 ，即光伏出力越大、电价越低。这是波动电价场景下最实质的一条耦合，也是“光伏大发时段应当多充电”这一直觉的定量依据（图 5）。

小区负载与光伏基本无关（ -0.0090 ），二者可以独立建模。

2.2.7 因子贡献汇总

把可能与三个变量相关的因子逐一作一元线性回归（文献 [14]），以决定系数 R^2 衡量其解释力，得表 8。

表 8 各因子的单独解释力 R^2

因子	电价	小区负载	光伏
星期几（6 df）	0.6955	0.8930	0.0007
{周五,周六}（1 df）	0.6953	0.8930	0.0001
月份（11 df）	0.3023	0.1108	0.9122
节气四季·四立（3 df）	0.1413	0.0376	0.6904
节气 24 个（23 df）	0.2751	0.1026	0.9416
太阳黄经 1 阶（2 df）	0.0528	0.0274	0.7653
太阳黄经 4 阶（8 df）	0.2775	0.0972	0.9376

组合模型方面，“星期几 + 月份”对电价、负载、光伏的解释力分别为 0.9860、0.9991、0.9122；再加入太阳黄经 4 阶谐波后升至 0.9943、0.9994、0.9696。

小区负载几乎被“星期 + 月份”完全解释。

读表须注意：在单一年份内“月份”与“太阳黄经”近乎共线，故增量 R^2

会严重低估节气——谁先进入模型谁就取走共同部分。判别节气还是日历，必须依赖 §2.2.4 的外推实验。

2.2.8 数据体检

数据完整性。 建模前对四份附件逐单元格核查。附件 1 为 144 段、无空值（电价 0.3713~1.3952 元/kWh、负载 3309~5959 kW、光伏预测 0~7612.3 kW，其中 55 段为夜间零值）；附件 2 为 365×144 的负载与光伏实际值、无空值（负载 1996~7979 kW，光伏 0~10216.2 kW，夜间 23 540 段为零）；附件 3 共 1460 行 = 365 天 × 4 个发布时刻，**四版齐全、无缺版、无空值、无负值、无整行全零**，预报值 0~9995.9 kW；附件 4 为 365×144 电价、无空值，区间 0.0076~1.7936 元/kWh。

两处异常值经核查后保留原值：其一，附件 4 有 9 段电价低于 0.05 元/kWh（最低 0.0076，出现在 2025-04-26 正午前后），占全部时段的 0.017%，属真实数据中的“午间深谷”事件，本文不做平滑；其二，附件 2 的光伏实际峰值最高 10216.2 kW（2025-04-26），显著高于附件 1 典型日峰值 7612.3 kW，说明附件 1 只是全年逐段均值而非任何一天的实际曲线（核对见 §2.1.2）。

节假日效应。2025 年法定节假日（文献 [13]）为春节 1/28—2/4（8 天）、清明 4/4—4/6、劳动节 5/1—5/5、端午 5/31—6/2、国庆中秋 10/1—10/8。以“同月 × 同星期”的非假日为基线逐日比对并剔除季节漂移后，五个假期的逐日偏差均值分别为春节 -0.11% （8 天）、清明 $+0.25\%$ （3 天）、劳动节 $+0.21\%$ （5 天）、端午 $+0.50\%$ （3 天）、国庆中秋 -0.11% （8 天），单日偏差均在 $\pm 1.4\%$ 以内。

五个假期的平均偏差均在 $\pm 0.5\%$ 以内。未剔除季节时观察到的“假日比非假日低 4.5%”完全由季节漂移解释——10 月本身就是全年负载最低的月份。**故本数据集不含节假日与农历效应，建模中不引入节假日哑变量。**（真实电力系统中长假期间负荷会显著下降，此处结论仅针对本数据集，属数据体检结果。）

预报误差的季节性与口径。附件 3 的 0:00 版对当天全天的日总发电量误差（折算为全天平均功率）按四立四季为春 124.7、夏 121.0、秋 152.9、冬 89.1 kW；同一版本若按逐时段比较，四季平均绝对误差为春 344、夏 358、秋 404、冬 253 kW。两个口径相差约三倍：前者说明“一天的总量

报得较准", 后者说明"逐时曲线对不准", 而调度关心的是后者, 故本文在误差分档与冗余裕量标定中一律使用**逐时段**口径 (§6.3、§6.4)。

且 2、3、8、10 月存在 67~86 kW 的系统性低估。预报误差的方差随季节变化, 是后文对预报可信度分档并滚动标定的直接原因。

2.2.9 信息可得性：一月窗口能识别出什么

题目给出的结果模板中, "计划购电量""充放电量""紧急购电量"等工作表**只列到 2 月 1 日及其后的日期**, 1 月不在填写范围内。因此本文按模板口径把 1 月作为**调参与训练期**: 人为参数在 1 月窗口上标定, 模型在 1 月完成预热, 自 2 月 1 日起才产生需要上报的计划与结算结果。

题目要求结果文件的填写范围是 **2025 年 2 月 1 日至 12 月 31 日**, 因此 **1 月是唯一在统计期开始之前就已完整发生的月份**, 是全部人为参数的合法标定窗口。但 1 月的样本是有限的, 它能识别什么、不能识别什么, 直接决定标定协议的形态。

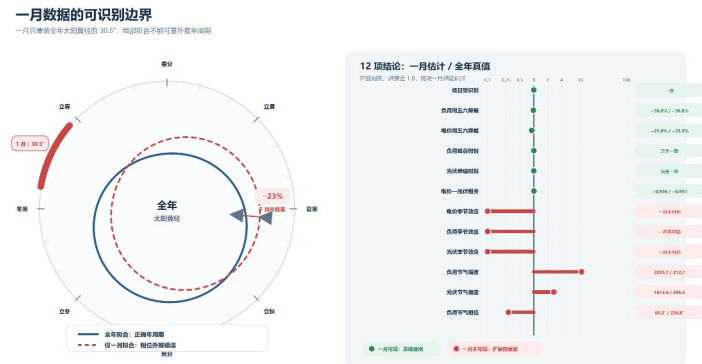


图 6 一月可识别性

表 9 各项结论的 1 月估计与全年真值

结论	只用 1 月	全年真值	可得性
低日型组内残差（负载，kW）	24.8（次优分组起）	568 297.3	✓ 可，且判别更干净
负载周五 / 周六降幅	-36.0%	-36.8%	✓ 可
电价周五 / 周六降幅	-21.0%	-23.3%	✓ 可
光伏与星期无关	$R^2=0.0499$	0.0007	✓ 可
日内峰谷时刻	负载 9:00 / 22:00、电价 20:30、光伏 11:50	9:00 / 22:00、20:30、12:00	✓ 完全一致
电价—光伏相关系数	-0.956	-0.951	✓ 几乎精确
负载—光伏相关系数	-0.826	-0.009	✗ 符号相反，结论错误
月份 / 季节效应	无法估计	电价 0.302 / 负载 0.111 / 光伏 0.912	✗ 不可
节气幅度（负载 / 光伏，kW）	2295.7 / 1613.6	212.7 / 595.5	✗ 差 10 倍 / 高估 2.7 倍
节气相位（负载）	66.2°	234.8°	✗ 错 169°
附件 3 预报日总量误差（折算功率）	89.6 kW	122.3 kW	⚠ 乐观 27%

两处最值得警惕。其一, "负载—光伏相关系数"在 1 月为 -0.826, 全年真值却是 -0.009: 1 月的天气同时驱动二者（低温日负载高、光伏低），形成虚假相关, 只看一个月会得出方向相反的结论。其二, **根因**是 1 月 1—31 日全部落在"冬"内（立春在 2 月 3 日），月内不存在任何季节切换, 且 1 月只覆盖太阳黄经 30.5°（占全年 8.5%），不足以辨识一条 360° 的周期曲线。（图 6(b) 中"月份 / 季

节效应"一项在 1 月不存在估计值，图中置于对数轴的示意位置并标注"一月不可估"，该位置不代表任何实验结果。）

由此得到两条对建模的直接约束：

其一，**可以放心冻结的**：日型分组、星期效应的大小与稳定性、日内形状与峰谷时刻、光伏与星期无关、电价与光伏的耦合。

其二，**不能一次冻结的**：季节水平、节气幅度与相位、随季节变化的预报误差方差。这些量在 1 月窗口上**结构性不可识别**。因此本文不采用"1 月标定一次、此后冻结"的做法，而采用**扩展窗逐日 / 逐月重标定**：第 d 天的参数只用第 $0 \sim d-1$ 天的数据重新确定，使模型随运行时间自然获得季节信息（详见 §6.5）。

2.3 四个指定日期的选取

题目表 3 要求给出 **2025 年 3 月 20 日、6 月 21 日、9 月 23 日、12 月 21 日**四个日期的结果。这四个日期并非随意指定：用天文算法（文献 [12]）计算 2025 年二十四节气时刻可得，它们恰为**春分、夏至、秋分、冬至**四个节气（北京时间分别为 3 月 20 日 16:50、6 月 21 日 10:34、9 月 23 日 02:23、12 月 21 日 23:01），亦即四季的**中点**。四季之始为四立：立春 2 月 3 日、立夏 5 月 5 日、立秋 8 月 7 日、立冬 11 月 7 日。

因此这四个日期构成一组**季节代表日**，其光伏与负载水平分别处于全年周期的不同相位上，是检验模型能否适应不同季节的合理取样。§2.2.4 已表明光伏的年变化服从太阳几何，故这四个节气日对光伏尤其具有代表性。

2.4 问题一的分析

题目给了什么。 附件 1 给出全天 144 段的电价、负载与光伏预测功率，三者当天全部已知；储能参数（12000 kWh 容量、5000 kW 功率、1200~10800 kWh 储电量区间、90% 效率、初始 6000 kWh）全部给定；另加一条硬约束——储能在 0:00 与 24:00 的储电量相同。

难在哪里。 目标函数与全部约束均为线性，决策变量为每段的购电量、充电量与放电量，状态变量为储电量，规模仅 144 段，**求解本身不构成困难**。真正的风险在建模口径，有三处必须先确定：其一，**效率如何折算**——题目只给出"90%"这一个数，本文取充、放各 90%（往返 81%），状态转移式为 $S_{t+1} = S_t + \eta c_t \Delta t - d_t \Delta t / \eta$ ，并在 §8.5 给出替代口径的影响；其二，**日初日末等电量是否为硬约束**——这是问题一的**专有条件**，它把储能限制在日内套利、禁止跨日搬移，问题二、三、四均无此要求，这是问题一与后三问最大的结构差异；其三，**能否弃光**——当光伏超过负载与充电能力之和时，多余电量必须允许丢弃，否则模型不可行。

为什么这样建模。 既然约束线性、规模很小，就没有理由使用启发式或近似算法：本文直接求解线性规划得到全局最优，再用一条**完全独立的路径**——储电量离散化动态规划——复核，把"求解器是否正确"这一问题一次性排除。

2.5 问题二的分析

题目给了什么。 附件 2 给出全年逐日逐时的**实际负载与实际光伏出力**；电价仍取附件 1 的曲线，每天重复；另给两条计费规则：供电不足时按当时电价的**5 倍**紧急购电，除紧急购电外其他时段一律按**计划购电量**计费。

难在哪里。 难点不在优化而在**信息边界**：计划必须在当天 0:00 做出，而附件 2 给的是当天的实际值。此处有一个锋利的判据——一个**正确的问题二模型**，其**紧急购电费必须显著大于零**，因为题目专门设计 5 倍紧急购电，其前提正是"计划时并不知道当天会发生什么"；若 0:00 就能取得当天实际值，计划必然分毫不差，紧急购电费恒为零，模型会自我暴露。本文最初的建模恰好踩了这个坑：沿用问题一的写法（当天数据做计划 + 日初日末等电量），结果全年紧急购电费**恰好为零**。重读原题后确认"0:00 与 24:00 储电量相同"只写在问题一，于是做两处修正：**取消日界等电量约束，改为日末储电量自由**；**把一天切成信息结构完全不同的两个阶段**——计划阶段（0:00，只用第 $d-1$ 天及更早的数据）与执行结算阶段（当天实时，储能按实际缺口充放电，补不上才紧急购电）。

第二个难点是**冗余量应当留多少**。紧急购电是 5 倍价，"少买"的代价远大于"多买"，故计划必须留有余量；但余量并非越大越好——计划购电量一经确定即全额计费，多买的电虽可先进储能，**充不进去的部分只能弃掉而电费照付**。留多少余量因此需要定量权衡，本文用报童模型给出方向性依据、再用数据标定具体分位 (§6.3)。

为什么这样建模。 问题二由此从确定性优化变为"**预测 → 承诺 → 结算**"的组合：预测要准、承诺要留余量、结算按实际发生。三者中只有第一项能用精度衡量，后两项必须用**费用**衡量——这也是本文坚持以全年总费用而非预测误差来选参数的原因。

2.6 问题三的分析

题目给了什么。 在问题二之上，题目增加了一项信息——每天 0:00、6:00、12:00、18:00 各发布一版未来 24 小时的光伏预报（附件 3），并允许在预报更新时**调整**当天的购电策略。同时给出计费规则：计划购电量高于调整量的部分按 50% 计费、调整量高于计划量的部分按 1.5 倍计费、实际未供上的部分仍按 5 倍紧急购电。

难点之一：计费规则决定了"调整"本身有代价。 与问题二"一次计划定到底"不同，问题三里每一次修订都要为偏离计划付费，这意味着求解器必须**知道偏离计划要付钱**：若调整阶段仍以"最小化剩余时段购电费"为目标、把偏差费用留到事后统计，得到的只是一个**可行的滚动启发式**，而非该计费规则下的最优解。因此本文把上调量、下调量作为决策变量写进调整阶段的线性规划目标，用等式约束与计划量、调整量相连 (§6.4)。

难点之二：多版本预报如何使用。 附件 3 每天给出 4 个版本，每一版覆盖"发布时刻起连续 24 小时"，因此对**同一目标时段**往往同时有多个版本覆盖。本文注意到三件事，促使放弃"只取最新版"而改用多版本加权融合：旧版本并非无用，四个版本对同一时段的预报值并不相同，各自携带不同发布时刻的信息；可靠性可以用数据度量，提前量越大误差通常越大，这一关系可直接由附件 3 与附件 2 的历史对照统计得到；"只取最新版"会放大修订幅度，而每次改写都要付偏差费，融合后的预报在相邻版本之间更平滑。§6.4 的实测表明，**第三点正是省钱的真正来源**。

难点之三：权重按什么口径标定。 权重来自各方差 $\hat{\sigma}^2$ ，而方差必须用历史数据估计，此处有一条不能越过的红线：**标定 $\hat{\sigma}$ 只能使用该决策时刻已经能取得的数据**。本文采用**扩展窗**——第 d 天的决策只使用"目标时刻早于第 d 天"的样本，并把 $\hat{\sigma}$ 按**目标小时**与**提前量**两个维度细化，使权重真正反映"这一版对当前这个时刻的可靠性"。训练数据随时间自然累积，模型因此具备进化能力。

为什么这样建模。 问题三的核心不是"调整越多越好"，而是在**信息收益与偏差代价之间找平衡**：决策时点越晚信息越全，但每次调整都要付钱。本文因此把它建模为**把三项费用完整写进目标函数的滚动优化**：0:00 用当时可得的全部版本定下计划，6:00、12:00、18:00 各用当时可得的全部版本重新融合预报并重新优化，每次都在"改得更准"与"少改少罚"之间自动权衡。

2.7 问题四的分析

问题四把电价由附件 1 的固定日曲线换成附件 4 的逐日逐时波动电价，要求在此条件下重新计算问题二（记为 4-2）与问题三（记为 4-3）。

2.7.1 问题 4-2：波动电价下重算问题二

多了一重不确定性——电价本身也要预测。 附件 4 只给**实际**电价，题目未提供任何电价预报，而计划仍在 0:00 制定，因此计划阶段只能用 0:00 之前的历史电价外推当天电价，结算时再使用真实电价，紧急购电按真实电价的 5 倍计。信息结构与问题二完全相同（仅 0:00 一次决策），差别只在目标函数中的电价序列由固定曲线换成本日预测曲线；电价预测沿用与净负荷相同的思路——近期水平与星期基线的加权 (§6.5)。

2.7.2 问题 4-3：波动电价下重算问题三

在 4-2 之上再叠加 6:00、12:00、18:00 三次调整，此时同时存在三笔费用：成交部分购电费、调整产生的偏差费、补不足的紧急购电费。一个**必须写明的不对称**：附件 3 只提供光伏预报，附件 4 只提供电价实际值，**没有任何电价预报**，因此日内三次调整能够改善的**只有光伏这一重信息**，电价的预测误

差仍要由 0:00 的模型独自承担。这一不对称直接决定 4-3 的调整收益能否为正，本文在 §6.5 以实测结果回答。

建模与求解框架



图 7 四个问题共用的购电优化内核及三类替换条件

图 7 左侧为四份数据附件的输入；附件 5 仅规定结果文件的模板，不进入优化内核。中间是四问共用的优化内核，右侧是三类替换条件。

3.1 统一内核

设一天含 $T=144$ 个时段，时段长度 $\Delta t=1/6$ h。决策变量为每时段的购电量 p_t 、充电量 c_t 与放电量 d_t （均以功率计，单位 kW），状态变量为储电量 S_t (kWh)。内核的形式为

$$\min \sum_{t=0}^{T-1} \lambda_t p_t \Delta t$$

约束条件为

$$p_t - c_t + d_t \geq N_t, t=0, \dots, T-1$$

$$S_{t+1} = S_t + \eta c_t \Delta t - \frac{d_t \Delta t}{\eta}$$

$$0 \leq c_t \leq P_{\max}, 0 \leq d_t \leq P_{\max}, p_t \geq 0$$

$$S_{\min} \leq S_t \leq S_{\max}$$

其中 $N_t = L_t - P_t^{\text{used}}$ 为净负荷， L_t 为小区负载， P_t^{used} 为实际消纳的光伏功率。四问的区别仅在于： λ_t 取什么序列、 N_t 用预测值还是实际值、以及是否追加偏差费用项。

3.2 三类替换条件

替换一：电价序列。 问题一至三取附件 1 的典型日曲线，每天重复；问题四取附件 4 的逐日实际电价，且在计划阶段只能用其预测值。

替换二：信息的可得性。 问题一全部已知；问题二只在 0:00 决策一次；问题三与问题四-3 在 0:00、6:00、12:00、18:00 四次决策。每次决策只允许使用“发布或发生时刻不晚于该时刻”的数据。

替换三：计费规则。 问题一、二按计划购电量全额计费；问题三、四-3 拆分为成交、违约、超发三项，未供足部分按 5 倍紧急购电。

3.3 求解与验证的分工

内核是一个标准的线性规划，规模为每时段 3 个决策变量与 $T+1$ 个状态变量，用单纯形法或内点法均可求解，本文调用 HiGHS 求解器。

真正的验证工作不在求解，而在三处：**信息是否越界**、**口径是否自洽**、**结果是否可复算**。本文的验证设计为：

其一，**双路径互证**。问题一同时用线性规划与储电量离散化动态规划求解；问题二同时用线性规划与在线滚动求解。两条路径不共用代码。

其二，**泄露检验**。把统计期起点之后的全部实际值置零后重跑，要求标定常数、调度轨迹与逐段决策逐位不变。

其三，**独立复核**。由独立脚本从解文件重算费用、校核能量守恒与约束边界，不调用求解器。

模型假设

本文的假设分为三类，逐条给出依据。

4.1 关于储能设备

假设 1 储能的充、放单项效率均为 90%，往返效率 81%，状态转移满足 $S_{t+1} = S_t + \eta c_t \Delta t - d_t \Delta t / \eta$ 。

依据：题目只给出“充放电效率为 90%”这一表述。若理解为往返效率，则单项效率为 $\sqrt{0.9} \approx 94.87\%$ ，与“充放电效率”的字面含义不符。本文取单向 90%，

并在 §8.5 给出替代口径的灵敏度结果。

假设 2 储电量的上下限 1200 与 10800 kWh 在每一个时段均须满足，而非仅日界满足。

依据：题目原文为“在后继的充放电过程中，储能设备的电量必须保持在 1200 - 10800 kWh 之间”，

“过程中”表明是逐时段的约束。

假设 3 充放电功率不超过 5000 kW，且同一时段不出现同时充放，即 $c_t \cdot d_t = 0$ 。

依据：同时充放只会增加损耗而无收益，由于 $\eta < 1$ ，最优解必然满足该条件，

因此这一约束在最优解处自动成立，无须显式写入模型。

假设 4 储能不存在自放电与容量衰减，全年参数恒定。

依据：题目仅给出单一组参数，未提及衰减。

4.2 关于供电与结算

假设 5 微网不向外部电网售电，光伏的富余电量只能弃用。

依据：题目只定义了购电方向，未给出上网电价。若允许售电，模型的收益结构将改变。

假设 6 “微网提供的电能不可低于小区负载”按每个 10 分钟时段逐段成立，而非按全天总量成立。

依据：题目的表 3 要求给出**指定日期各时段**的紧急购电量，

说明供电不足是按段判定并逐段结算的。

假设 7 紧急购电按**实际发生时刻**的电价的 5 倍计；在问题一中不存在紧急购电（信息完备，计划必然可行）。

4.3 关于信息与预测

假设 8 计划阶段的决策只能使用决策时刻已经可得的数据；当天实际值只进入实时执行与结算环节。

依据：这是题目"在每天 0:00 制定当天的计划购电策略"的直接推论，

也是本文全部预测模型与标定协议的出发点。

假设 9 问题四中电价在计划阶段不可知，须由历史电价外推；题目未提供任何电价预报，故日内亦无新的电价信息。

依据：题目对附件 3 明写"发布"，对附件 4 只写"提供"，二者信息结构不同。

假设 10 各时段的预测误差相互独立，冗余裕量按逐时段的分位数给出。

依据：本文对同时段误差取经验分位数，未施加跨时段的相关结构。

该假设偏保守，因为它忽略了不同时段误差之间的相关性。

符号说明

表 10 全文符号、含义与单位

符号	含义	单位
t	时段序号, $t=0,1,\dots,143$	—
Δt	时段长度, $\Delta t=1/6$	h
L_t	第 t 时段的小区负载功率	kW
P_t	第 t 时段的光伏实际出力	kW
P_t^{used}	第 t 时段实际消纳的光伏功率	kW
N_t	第 t 时段的净负荷, $N_t = L_t - P_t^{used}$	kW
λ_t	第 t 时段的电价	元/kWh
p_t	第 t 时段的购电功率	kW
p_t^{plan}	第 t 时段的计划购电功率	kW
p_t^{adj}	第 t 时段的调整后购电功率	kW
c_t	第 t 时段的充电功率	kW
d_t	第 t 时段的放电功率	kW
S_t	第 t 时段起始时的储电量	kWh
η	储能的单向充放电效率, $\eta=0.9$	—
P_{max}	最大充放电功率, $P_{max}=5000$	kW
S_{min}, S_{max}	储电量上下限, 1200 与 10800	kWh
u_t	第 t 时段的下调量, $u_t = \max(0, p_t^{plan} - p_t^{adj})$	kW
v_t	第 t 时段的上调量, $v_t = \max(0, p_t^{adj} - p_t^{plan})$	kW
$\hat{N}_t$	净负荷的点预测值	kW

符号	含义	单位
α	指数平滑的衰减系数	—
β	星期基线在点预测中的权重	—
W	冗余裕量的统计窗口长度	天
q	冗余裕量所取的分位数	—
m_t	第 t 时段施加的冗余裕量	kW
Δ_v	预报版本 v 相对目标时段的提前量	h
$\hat{\sigma}(\cdot, \cdot)$	预报误差标准差的分档估计	kW
w_v	预报版本 v 的融合权重	—
δ	太阳赤纬	度
φ	地理纬度	度
H_0	水平面天文日辐射量	kWh/m ² ·d

模型建立与求解

6.1 统一的基础优化模型

四问共用的内核已在 §3.1 给出。此处补充三点在实现中必须明确的处理。

其一，光伏的消纳与弃用。令 $q_t = p_t - c_t + d_t$ 为微网的净供电功率，功率平衡要求 $q_t \geq N_t$ ，其中 $N_t = L_t - P_t^{used}$ 。 $P_t^{used} \leq P_t$ 为实际消纳的光伏功率，弃光量 $P_t - P_t^{used} \geq 0$ 。由于微网不向外部电网售电，另需 $q_t \leq L_t$ ，即不能把储能电“放向空处”。该约束在本文全部算例的最优解处均不紧，但不可省略。

其二，效率的折算方向。充电时只有 $\eta c_t \Delta t$ 进入储能，放电时需从储能取出 $d_t \Delta t / \eta$ 。全年累计量的校验式为 $\eta C - D / \eta = S_{末} - S_{初}$ ，其中 C, D 分别为全年充电与放电总量。§6.2 的实测结果满足该式。

其三，求解器与规模。单日模型含 $3T$ 个决策变量与 $T+1$ 个状态变量，约束 $3T+1$ 条，规模很小；本文用 HiGHS 求解（文献 [6]），单日耗时在毫秒量级。全年 334 天的逐日重解总耗时不超过一分钟，因此参数网格搜索在计算上是可行的。

6.2 问题一：确定性最优

6.2.1 模型

问题一的信息完备，且“0:00 与 24:00 储电量相同”是一条**专有**约束。在此约束下储能只能在日内套利，不能跨日搬移。模型即 §3.1 的内核，追加 $S_T = S_0 = 6000$ kWh 的端点等式条件（ S_0 由附录 1 给定）。

6.2.2 结果

表 11 问题一：指定时间段的购电量及全天购电量和购电费

时间段	购电量 (kWh)	时间段	购电量 (kWh)	时间段	购电量 (kWh)
10:00-10:10	0.0000	12:00-12:10	480.4124	14:00-14:10	0.0000
16:00-16:10	445.4317	18:00-18:10	531.8940	20:00-20:10	0.0000
全天购电量 (kWh)	59482.6990	全天购电费 (元)	35126.95		

表 12 问题一：储能设备在指定时间段的充放电量及 0:00 与 24:00 的储电量

时间段	充电量 (kWh)	放电量 (kWh)	时间段	充电量 (kWh)	放电量 (kWh)
0:00-4:00	4500.0000	0.0000	12:00-16:00	5286.0352	91.1014
4:00-8:00	833.3333	6365.8412	16:00-20:00	0.0000	5780.1319
8:00-12:00	4787.9643	1702.9970	20:00-24:00	5333.3333	2859.8681
0:00 储电量	6000.0000		24:00 储电量	6000.0000	
(kWh)			(kWh)		

校验：充电合计 20740.6661 kWh，放电合计 16799.9396 kWh，放电 / 充电 = 0.81000，恰为单向效率 90% 所对应的往返效率 81%；且 $\eta C - D / \eta = 0$ ，与 $S_{末} = S_{初} = 6000$ kWh 一致。

运行的形态：0:00—4:00 电价低谷时以 4500 kWh 的规模充电；4:00—8:00 电价抬升时放电 6365.84 kWh，覆盖早高峰；8:00—16:00 光伏出力充足，净负荷转负，储能持续充电（合计 10074 kWh）；16:00—20:00 晚高峰放电 5780.13 kWh；20:00—24:00 再次充电 5333.33 kWh，把日末储电量恢复到与日初相同的 6000 kWh。全天购电量 59482.70 kWh，其中 8:00—18:00 有大量时段的购电量为零，完全由光伏与储能供给。

6.2.3 其余作法

其余作法（动态规划独立复算、储能价值分解、效率口径的替代取值）见附录 C.5。

6.3 问题二：含预测不确定性的逐日滚动优化

6.3.1 信息结构与执行口径

与问题一相比有两处根本差别。其一，**取消了日界等电量约束**——该条件只写在问题一，问题二并无此要求；若沿用会把储能锁死在日内，与“逐日连续运行”的题意不符。其二，**计划必须在 0:00 做出**，而附件 2 给出的是当天的实际值，二者不可混用。

本文把一天切成两个信息结构完全不同的阶段：

计划阶段（0:00）：只用第 $d-1$ 天及更早的数据预测当天的净负荷，据此求解线性规划，得到计划购电量 p_t^{plan} 。此阶段**只锁定购电量**，不预设储能的充放电动作。

执行结算阶段（当天实时）：储能按实际缺口实时充放电；净负荷为正时放电，放电能力受功率上限与当前储电量限制；净负荷为负时充电，充电能力同样受限。储能补不上的部分才按当时电价的 5 倍紧急购电。计划购电量一经确定即全额计费，未用满的电费照付，充不进储能的富余电量弃用且不退款。

这一口径提供了自我检查的机会：**若在费用参数正常的前提下紧急购电长期为零，就应当排查计划阶段是否误用了当天实际值。**

6.3.2 净负荷的点预测

预测对象取净负荷 $N_d(t) = L_d(t) - P_d(t)$ 。附件 2 的结构显示两个显著特征：

周周期（周五、周六的日均负载比其余五天低 36.8%，全年稳定）与

近期水平（净负荷随季节缓慢漂移）。二者按权重 β 融合：

$$\hat{N}_d(t) = (1 - \beta) EMA_d(t) + \beta B_c(d)$$

其中 $EMA_d(t) = \alpha N_{d-1}(t) + (1 - \alpha) EMA_{d-1}(t)$ 为同段位的指数平滑（文献 [8]）， $B_c(d)$ 为日期 d 所属星期类别（周五 / 周六为一类，其余五天为一类）的历史均值基线。

因果性： EMA 只使用 $d-1$ 及更早的实际值， $B_c(d)$ 只统计 d 之前同类日的历史均值。当天数据不进入预测。

6.3.3 冗余裕量：报童模型给出的方向性依据

紧急购电是 5 倍价，"少买"的代价远大于"多买"，因此计划必须留有余量。把每个时段视作一次独立的单周期报童决策：订购量为计划购电量，需求为实际净负荷；缺货（少买）的单位成本为 $c_u=4\lambda_t$ （少买一单位需按 $5\lambda_t$ 紧急购电，计划内只需 λ_t ，净增 $4\lambda_t$ ）；过剩（多买且充不进储能）的单位成本为 $c_o=\lambda_t$ 。报童模型给出最优订购量满足的临界分位条件（参见文献 [1][3]）：

$$F(Q^*)=\frac{c_u}{c_u+c_o}=\frac{4\lambda_t}{4\lambda_t+\lambda_t}=\frac{4}{5}=80\%$$

即计划购电量应取净负荷预测误差分布的 80% 分位（文献 [1][2][3]）。
必须指出，该式只是方向性依据，不是精确解。它建立在三个简化假设之上：每个时段独立决策、过剩电量全部浪费、无储能耦合。实际问题中多买的电有一部分会进入储能、顶掉后续购电（该模型的教科书表述见文献 [4]），因此**真实最优分位不低于 80%**（需求分布未知时的稳健性讨论见文献 [5]）；但它也不会无限升高——充不进电池的那部分同样要付钱。实测全年最优分位为 88%，与理论值同处 80%~90% 区间，说明该式给出的方向是正确的。

具体实施时，对过去 W 天同时段的预测误差取经验 q 分位（分位数的定义与性质见文献 [14]），形成逐时段的冗余裕量 m_t ：

$$m_t=\max\left(0,Quantile_q\left\{N_k(t)-\widehat{N}_k(t):k=d-W,\dots,d-1\right\}\right)$$

W 与 q 由数据标定。

6.3.4 参数标定

四个参数 (α,β,W,q) 需要确定。此处有一条不可越过的红线：
只能使用统计期开始之前已经完整发生的数据。
为什么是第一个月。附件 2、附件 4 覆盖 2025 全年，而结果文件只需填写 2 月 1 日至 12 月 31 日。因此 1 月是唯一在统计期开始之前就已完整发生的月份（1 月 1 日为初始化日，实际标定窗口为 1 月 2 日至 31 日，共 30 天）。用 1 月标定、再用标定结果预测 2—12 月，不涉及任何未来信息。
标定方式。把四个参数在 1 月窗口上做网格搜索，取该窗口内总费用最小的一组。网格范围 $\alpha\in\{0.10,0.15,0.25,0.40\}$ 、 $\beta\in\{0.6,0.7,0.8,0.9\}$ 、 $W\in\{1,\dots,30\}$ 天、 $q\in\{0.70,0.75,\dots,0.98\}$ 。

标定结果： $\alpha=0.10$ 、 $\beta=0.90$ 、 $W=5$ 天、 $q=88\%$ 。全年费用 13700048.95 元。
标定的可靠性。1 月窗口选出的分位是准确的（88% 与全年最优一致，1 月窗口的前五名中有三个是 88%），但窗口长度会选偏（1 月选出 5 天，全年最优为 7 天）。这一偏差的代价很小：若只把 (W,q) 用 1 月标定、 α/β 沿用其他取值，全年费用为 13707511.64 元，与本文结果相差 0.05%。

6.3.5 结果

表 13 问题二全年费用构成（统计期 2025.2.1—12.31，共 334 天）

项目	金额（元）	占比
计划购电费	13 034 144.67	95.14 %
紧急购电费	665 904.27	4.86 %
合计	13 700 048.95	100 %

紧急购电量合计 110410.16 kWh，全年 172 天出现过紧急购电。紧急购电费占比 4.86%——显著大于零，表明计划阶段没有使用当天实际值。

作为对照，完美信息下界为 12227243.64 元，本文结果距其 12.05%。

6.3.6 指定日期的结果

题面要求按题面表 1~表 3 的格式给出 3.20、6.21、9.23、12.21 四天的结果。完整结果见 result2.xlsx，摘要如下。

表 14 问题二：指定日期的计划购电量（按题面表 1 的时段）

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
10:00-10:10	0.00	0.00	0.00	0.00
12:00-12:10	616.44	0.00	464.00	940.39
14:00-14:10	0.00	0.00	0.00	0.00
16:00-16:10	512.84	250.32	523.36	828.12
18:00-18:10	698.71	0.00	784.06	717.34
20:00-20:10	0.00	0.00	0.00	0.00
全天购电量	68970.46	35842.65	67090.91	93979.17
全天购电费	41747.16	21334.89	41744.75	60214.02

表中六个时段列为该时段 10 分钟的购电量，其余时段见 result2.xlsx；全天购电费不含紧急购电费。

表 15 问题二：指定日期储能设备的充放电量与储电量

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
0:00-4:00	8620.94 / 36.62	700.43 / 61.12	9013.83 / 27.33	8027.20 / 148.23
4:00-8:00	1992.28 / 5067.40	1505.07 / 1719.95	1448.18 / 6407.37	2457.50 / 1531.36
8:00-12:00	6252.38 / 2777.38	8490.52 / 0.00	4895.05 / 1896.62	5442.42 / 7516.33
12:00-16:00	3366.21 / 396.39	0.00 / 0.00	5465.69 / 635.18	7703.62 / 2307.05
16:00-20:00	181.26 / 5884.39	182.71 / 5345.01	86.44 / 5881.97	46.37 / 5921.04
20:00-24:00	557.32 / 2829.30	1018.46 / 2502.47	584.15 / 2364.99	381.79 / 2762.42
0:00 储电量	1635.19	3152.55	1442.57	2304.56
24:00 储电量	1629.11	3161.63	1660.51	1528.20

每格为"充电量 / 放电量"，单位 kWh；四个日期与题面表 3 一致。

表 16 问题二：指定日期的紧急购电

日期	紧急购电时间段与购电量 (kWh)	当日合计 (kWh)
2025-03-20	20:30-20:40: 166.1391	166.1391
2025-06-21	无	0.0000
2025-09-23	20:20-20:40 : 662.4373 ; 20:50-21:10: 27.1899	689.6272
2025-12-21	20:30-20:40: 93.9667	93.9667
四天合计		949.7330

四天的紧急购电全部出现在晚间电价高峰段（20:20—21:10），其中 6 月 21 日为零——该日光伏出力大、净负荷低，计划余量足够覆盖误差。

6.4 问题三：多版本预报融合与滚动调整

6.4.1 信息结构与取版规则

问题三给出每天 0:00、6:00、12:00、18:00 四个发布时刻的预报，每版覆盖发布时刻起连续 24 小时。因此对同一个目标时段往往存在多个版本。例如要预测某天上午 10:00，可用版本至少包括：当天 6:00 版（提前量 4 小时）、当天 0:00 版（提前量 10 小时）、前一天 18:00 版（提前量 16 小时）、前一天 12:00 版（提前量 22 小时）。

取版规则：在决策时刻 dec 上，目标时段 s 可用的版本集合为

$$V(s, dec) = \{v: pub(v) \leq dec \text{ 且 } v \text{ 覆盖 } s\}$$

可用性按决策时刻判定，不按目标时段的起点判定——否则 0:00 的计划会“偷看”当天 6:00 才发布的版本。调整阶段（6:00、12:00、18:00）只求解目标时段不早于该时刻的部分，已过时段不再改动。

6.4.2 三项费用完整写入优化目标

这是问题三与常见做法分歧最大之处。计费规则为：计划量高于调整量的部分按 50% 计费、调整量高于计划量的部分按 150% 计费、未供足部分按 5 倍紧急购电。若调整阶段仍只以“最小化剩余时段购电费”为目标，把偏差费用留到事后统计，得到的只是一个可行的滚动启发式，而非该计费规则下的最优解。

本文把上调量 v_t 、下调量 u_t 作为决策变量写入目标，并以等式 $p_t^{adj} - v_t + u_t = p_t^{plan}$ 与计划量相连。调整阶段的目标函数为

$$\min \sum_{t \in T_{adj}} [\lambda_t p_t^{adj} + 0.5 \lambda_t u_t + 0.5 \lambda_t v_t] \Delta t$$

其中 T_{adj} 为该阶段的剩余时段集合，计划阶段的目标只含 $\lambda_t p_t \Delta t$ 。注意：题面的 1.5 倍与 0.5 倍均含基础电价，故以成交量为基准时对上调量只加 $0.5 \lambda_t v_t$ （等价写法与证明见附录 B）。

6.4.3 融合权重与扩展窗标定

各版本按逆方差加权融合（文献 [9]）：

$$\hat{P}_t = \frac{\sum_{v \in V(s, dec)} w_v P_t^{(v)}}{\sum_{v \in V(s, dec)} w_v}$$

$$w_v = \frac{1}{\hat{\sigma}^2(\text{目标小时}, \Delta_v)}$$

其中 Δ_v 为版本 v 相对目标时段的提前量。

$\hat{\sigma}$ 的标定采用扩展窗：第 d 天的决策只使用“目标日 $\leq d-1$ ”的残差样本。这里有一条容易被忽略的边界：不能简单按“发布日 $< d$ ”筛选——前一日 18:00 版覆盖到第 d 天 17:00，其残差含有第 d 天的实际值，属未来信息；必须按“目标小时所属日 $< d$ ”筛选。

方差估计按目标小时与提前量两个维度分档：

$$\hat{\sigma}^2(\text{目标小时}, \Delta) = \frac{1}{2} s_{\text{格子}}^2(\text{目标小时}, \Delta) + \frac{1}{2} s_{\text{合并}}^2(\Delta)$$

该式不含任何自由参数——两半的权重取自“格子样本量与合并样本量”的等权折中，不需要人为调参，因而也不存在用未来数据调参的可能。第 31 天可用样本 2940 条（与“固定 1 月窗”相同），第 364 天增至 34908 条。

权重是怎么标定出来的：三项实验。

其一，残差的构造。把附件 3 的每一个版本与其目标时段的附件 2 实际值逐条相减，得到残差表。附件 3 每天有 4 个发布时刻、每版覆盖 24 小时，故全年可构造 $365 \times 4 \times 24 = 35040$ 条残差；剔除目标时刻越出年末的 36 条后，年内共 35004 条。

其二，分档的依据（标准差与方差的常规定义见文献 [14]）。把残差按“目标小时”与“提前量”两个维度交叉分组，同一格内样本的目标小时与提前量都相同，误差分布可直接比较。实测提前量的影响显著且单调：

表 17 提前量与预报误差（四版本、全部目标时段汇总，每组 $n \approx 1460$ ）

提前量	1 h	6 h	12 h	18 h	24 h
平均绝对误差 (kW)	226.9	249.4	302.8	362.0	501.1
标准差 $\hat{\sigma}$ (kW)	351.4	382.2	481.9	602.6	938.8
相对方差 $\hat{\sigma}^2 / \hat{\sigma}_{1h}^2$	1.00	1.18	1.88	2.94	7.13

提前量由 1 小时增至 24 小时，标准差升至 2.7 倍、方差升至 7.1 倍——这正是必须按 $1/\hat{\sigma}^2$ 加权、且必须把提前量分档的原因。同时，同一目标时段在不同版本上的误差差别同样巨大：以 12 时为例，提前 6 小时的版本平均绝对误差 387 kW、12 小时 600 kW、18 小时 830 kW、24 小时 1384 kW，相差 3.6 倍。若只取最新版，就等于放弃了这批可用的差异信息。

其三，**标定窗口的选择（本项最关键）**。分别用固定 1 月窗（只用目标日落在 1 月 1—31 日的残差）与**扩展窗**（第 d 天的决策使用目标日 $\leq d-1$ 的全部残差）标定并做消融。第 31 天扩展窗可用样本 2940 条，与固定 1 月窗完全相同；此后逐日增长，至第 364 天为 34908 条。两者全年费用相差 20937.64 元，且扩展窗在按 2—4 月、5—7 月、8—12 月切分的三段中**全部更优**。这说明权重确实随数据累积而变准，模型具备进化能力。

6.4.4 结果

表 18 问题三全年费用构成（334 天）

项目	金额（元）	占比
计划购电费（成交部分）	13583966.44	95.34%
调整偏差费	307281.74	2.16%
紧急购电费	356859.29	2.50%
合计	14248107.47	100%

紧急购电量合计 66968.88 kWh。

表 19 取版与标定方式的消融

做法	全年费用（元）	相对基准
只取最新的那一版	14483771.06	基准
多版本逆方差融合（固定 1 月 $\hat{\sigma}$ ）	14269045.11	-214725.96（-1.48%）
多版本逆方差融合（扩展窗 $\hat{\sigma}$ ，本文正式）	14248107.47	-235663.60（-1.63%）

对题面追问的回答。 题目要求分析“是否需要引入其他时刻的预报制定调整购电策略”。本文的结论是：**需要，但其价值高度依赖权重标定的准确度**。在固定 1 月窗口标定时，调整带来的收益被偏差费用抵掉，甚至可能为负；改用扩展窗后，权重误差下降，调整才稳定地省钱：

表 20 调整的价值（扩展窗标定下）

方案	全年费用（元）	紧急购电量（kWh）
不调整（只有 0:00 计划）	14315387.02	—
含调整（本文正式）	14248107.47	66968.88
调整的净收益	67279.55（0.47%）	下降 26.9%

分期核对（把 334 天切成 2—4 月、5—7 月、8—12 月三段）显示，扩展窗在 3/3 期中均优于固定 1 月窗口，分别节省 3317.90、13827.79、3791.95 元，方向稳定。**需注意口径：**问题二与问题三的预测输入与结算规则不同（问题三多了日内三次调整与偏差计费），两者的年度总费不宜直接作"净收益"比较；本表只在问题三同一口径下比较"有无调整"。

6.4.5 指定日期的结果

题面要求按题面表 1~表 3 的格式给出 3.20、6.21、9.23、12.21 四个日期的结果。

表 21 问题三：指定日期的计划购电量（按题面表 1 的时段）

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
10:00-10:10	0.00	0.00	0.00	0.00
12:00-12:10	652.51	0.00	581.45	884.57
14:00-14:10	0.00	0.00	0.00	0.00
16:00-16:10	780.07	235.18	821.15	1030.50
18:00-18:10	696.56	69.14	781.15	726.53
20:00-20:10	0.00	0.00	0.00	0.00
全天购电量	70529.38	36164.00	73796.70	95292.55
全天购电费	43003.45	22100.93	46173.96	62213.74

表中六个时段列为该时段 10 分钟的购电量，其余时段见 result3.xlsx；全天购电费不含紧急购电费。

表 22 问题三：指定日期的调整购电量（按题面表 1 的时段）

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
10:00-10:10	0.00	0.00	0.00	0.00
12:00-12:10	546.50	175.21	500.56	944.46
14:00-14:10	0.00	0.00	0.00	0.00
16:00-16:10	771.31	244.57	822.00	1030.50
18:00-18:10	696.56	69.15	781.15	726.53
20:00-20:10	0.00	0.00	0.00	0.00
全天购电量	68518.32	36770.70	72750.80	95235.07
全天购电费	42636.39	22656.10	46174.20	62570.80

表中六个时段列为 6:00、12:00、18:00 三次调整后的最终购电量，其余时段见 result3.xlsx。

表 23 问题三：指定日期储能设备的充放电量与储电量

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
0:00-4:00	8689.97 / 0.00	932.95 / 3.44	9129.89 / 0.00	6880.66 / 153.97
4:00-8:00	1419.00 / 6086.82	811.80 / 2021.93	1292.61 / 6179.16	1884.50 / 1536.33
8:00-12:00	4769.78 / 2571.28	8282.52 / 0.00	6155.95 / 1896.62	5647.98 / 7035.00
12:00-16:00	6149.17 / 254.72	0.00 / 0.00	4279.57 / 403.62	6840.79 / 2247.62
16:00-20:00	68.57 / 5717.05	360.68 / 5088.57	0.00 / 5658.91	37.64 / 5538.46
20:00-24:00	554.87 / 2983.62	1475.21 / 2485.90	539.55 / 3055.45	671.93 / 2845.41
0:00 储电量	1722.04	4025.87	1449.34	3863.60
24:00 储电量	1637.72	4036.22	1602.96	2123.21

每格为"充电量 / 放电量"，单位 kWh；为 6:00、12:00、18:00 三次调整后的最终调度。

表 24 问题三：指定日期的紧急购电

日期	紧急购电时间段与购电量 (kWh)	当日合计 (kWh)
2025-03-20	9:10-10:00：206.0980；20:30-20:40：2.0976	208.1956
2025-06-21	无	0.0000
2025-09-23	20:20-20:30：6.3750；20:50-21:10：64.7130；21:20-21:30：2.2307；21:50-22:00：6.8123	80.1310
2025-12-21	无	0.0000

日期	紧急购电时间段与购电量 (kWh)	当日合计 (kWh)
四天合计		288.3266

全年 138 天存在紧急购电，全年紧急购电量合计 66968.8844 kWh。

完整策略见 result3.xlsx。

6.5 问题四：波动电价下的重算

6.5.1 新增的不确定性：电价本身也要预测

附件 4 只提供逐 10 分钟的实际电价，题目未附加任何"预报"说明；而附件 3 对光伏明写"发布"。二者的信息结构不同，故问题四中电价成为与负载、光伏并列的待预测量：计划阶段只能用 0:00 之前的历史电价外推当天电价，结算时使用真实电价，紧急购电按真实电价的 5 倍计。

一个关键的不对称：日内三次调整能改善的只有光伏这一重信息，因为没有电价预报可用，电价的预测误差仍要由 0:00 的模型独自承担。

6.5.2 统一信息协议

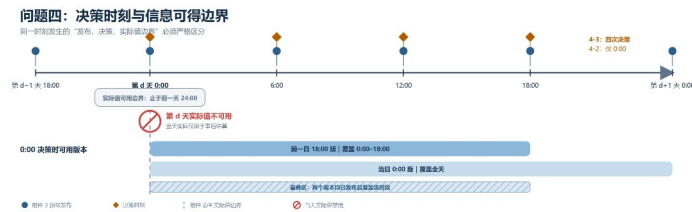


图 8 预报发布、滚动决策与实际数据可用边界

图 8 把预报的发布时刻、决策时刻与实际值的可用边界放在同一时间轴上。第 d 天的实际值在当天 0:00 决策时不可用，这是全文口径的核心。

表 25 问题四的信息协议

量	第 d 天 0:00 可用	不可用
小区负载	附件 2 中日期 $< d$ 的全部实际值	第 d 天及以后
光伏	附件 3 中发布时刻 $\leq$ 决策时刻且覆盖目标时段的版本	尚未发布的版本
电价	附件 4 中日期 $< d$ 的全部实际值	第 d 天的任何电价

4-2 只在 0:00 决策一次（信息结构与问题二相同）；

4-3 在 0:00、6:00、12:00、18:00 四次决策（信息结构与问题三相同）。两者的光伏预报层完全一致，差别只在决策次数。

6.5.3 可进化标定：把"冻结常数"改为"随运行重标定"

§2.2.9 已表明 1 月窗口无法识别季节水平、节气幅度与相位；把常数在 1 月标定一次即冻结，等于让模型全年使用冬令参数。本文改用扩展窗重标定：第 d 天的参数只用第 $0 \sim d-1$ 天的历史确定—— (α, β) 逐日重标定（以截至前一日的一步预测 MAE 最小为准）， (W, q) 每月 1 日以"截至当日全部历史的总费用最小"重标定，光伏权重 $\hat{\sigma}$ 与问题三共用同一张逐日扩展窗表（零自由参数）。最小估计窗口取 31 天：窗口不足一个自然月时每个星期类样本太少，网格搜索只是在噪声上过拟合。

模型确实在自我进化：第 30 天负载与电价的 β 均为 1.00——此时只有 1 月数据，"看不出"水平漂移，故取纯星期基线；第 180 天窗口已跨 1—6 月、经历约 370 kW 的下行漂移，模型自行把负载 β 降至 0.90；冗余窗口则由第 31 天的 21 天逐步缩至 3 天。关键校验：第 31 天重标定出的历史费

用 1861692 元与"1 月冻结"口径的 1861691.86 元逐位一致，说明预热期完全等价于严格 1 月口径、自我更新自第 32 天才开始。

6.5.4 问题 4-2 的结果

表 26 问题 4-2 全年费用构成 (334 天)

项目	金额 (元)
计划购电费	14 319 269.39
紧急购电费	676 552.57
合计	14 995 821.96

紧急购电量 103 406.16 kWh，弃电量 2 974 170.6 kWh，全年充电 6 318 911.67 kWh、放电 5 117 971.64 kWh，放电 / 充电 = 0.80995。

表 27 问题 4-2 的标定方式对照

方案	全年费用 (元)	相对基准
严格 1 月冻结 ($W=21$, $q=70\%$)	15 248 641.21	
扩展窗可进化 (本文正式)	14 995 821.96	- 252 819.25 (- 1.66%)
终点参数冻结 (事后诸葛, 含前视)	14 879 005.62	- 369 635.59
全年最优常数 (含泄露, 不可用)	14 814 739.43	- 433 901.78
完美信息理论下界	12 779 221.24	距下界 + 17.3%

可进化模型吃掉了"1 月冻结 → 终点参数"这段差距的 68%，余下的 116 816 元 (0.79%) 是"第一个月只有 1 月数据"的学习成本，属信息结构的必然代价。

理论下界的口径：在"负载、光伏、电价全部已知"的松弛假设下，4-2 退化为确定性离线最优线性规划；此时紧急购电严格更贵、最优解不会走它，故下界即该 LP 的最优值 (四种起止储电量端点口径的极差仅 0.030%，稳健可用)。

6.5.5 问题 4-3 的结果

表 28 问题 4-3 全年费用构成 (334 天)

项目	金额 (元)
计划购电费 (成交部分)	14 010 506.02
调整偏差费	316 895.70
紧急购电费	558 051.00
合计	14 885 452.71

(三个分项分别四舍五入到两位小数后相加为 14 885 452.72 元，与精确合计相差 0.01 元，属舍入尾差。)

紧急购电量 88 054.52 kWh，弃电量 2 704 321.6 kWh，充电 6 376 053.59 kWh、放电 5 164 265.79 kWh，放电 / 充电 = 0.80995。

引入三次调整使费用下降 110 369 元 (0.74%)、紧急购电量下降 14.8%，即问题四同样回答了"需要引入其他时刻的预报"：在波动电价下，调整的收益依然为正。电价由已知变为需预测的代价，在问题二到 4-2 上为 +1 295 773.01 元 (+9.46%)，在问题三到 4-3 上为 +637 345.44 元 (

+4.47%)——4-3 的代价只有前者的一半，原因是 4-3 的日内三次调整虽无法改善电价信息，却能部分对冲电价预测误差带来的购电量偏差。

6.5.6 指定日期的结果

题面要求按题面表 1~表 3 的格式给出四个指定日期的结果。以下分别给出 4-2 与 4-3 的结果，完整策略见 result4-2.xlsx 与 result4-3.xlsx。

(一) 问题 4-2

表 29 问题 4-2：指定日期的计划购电量（按题面表 1 的时段）

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
10:00-10:10	0.00	0.00	0.00	0.00
12:00-12:10	678.77	0.00	607.31	895.26
14:00-14:10	0.00	0.00	0.00	0.00
16:00-16:10	802.47	186.94	838.78	1050.88
18:00-18:10	698.68	453.33	786.20	740.99
20:00-20:10	0.00	0.00	0.00	0.00
全天购电量	71371.49	32846.62	76538.64	95993.64
全天购电费	44781.48	16638.47	50380.62	73743.87

表中六个时段列为该时段 10 分钟的购电量，其余时段见 result4-2.xlsx；全天购电费不含紧急购电费。

表 30 问题 4-2：指定日期储能设备的充放电量与储电量

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
0:00-4:00	9850.45 / 0.00	242.37 / 487.81	7352.66 / 0.00	7657.83 / 181.16
4:00-8:00	320.39 / 6733.32	1418.77 / 1719.95	3446.64 / 5598.01	1712.26 / 1804.30
8:00-12:00	5928.19 / 1906.68	10140.32 / 0.00	6207.05 / 1896.62	5837.07 / 6869.95
12:00-16:00	5052.95 / 254.72	0.00 / 0.00	3101.42 / 403.62	6739.19 / 2327.35
16:00-20:00	12.16 / 5780.61	0.00 / 5767.17	16.95 / 5669.80	9.65 / 5320.23
20:00-24:00	464.75 / 2913.91	626.96 / 2645.00	699.14 / 3038.02	880.07 / 2842.53
0:00 储电量	1646.24	2631.75	1478.85	3473.56
24:00 储电量	1568.65	2017.40	1769.12	2531.02

每格为"充电量 / 放电量"，单位 kWh。

表 31 问题 4-2：指定日期的紧急购电

日期	紧急购电时间段与购电量 (kWh)	当日合计 (kWh)
2025-03-20	8:40-10:00： 870.6933； 20:30-20:40： 60.9708	931.6641
2025-06-21	无	0.0000
2025-09-23	20:10-20:20： 10.2344； 20:50-21:00： 9.1673	19.4017
2025-12-21	7:50-8:00： 5.7967	5.7967
四天合计		956.8625

全年 169 天存在紧急购电，全年紧急购电量合计 103406.1570 kWh。

(二) 问题 4-3

表 32 问题 4-3：指定日期的计划购电量（按题面表 1 的时段）

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
10:00-10:10	0.00	0.00	0.00	0.00
12:00-12:10	678.77	0.00	598.42	918.59
14:00-14:10	0.00	0.00	0.00	0.00
16:00-16:10	802.47	186.94	838.78	1050.88
18:00-18:10	698.68	502.31	784.23	740.99
20:00-20:10	0.00	0.00	0.00	0.00

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
全天购电量	71371.49	33347.43	76027.50	96541.83
全天购电费	44781.48	16991.89	50019.59	74334.75

表中六个时段列为该时段 10 分钟的购电量，其余时段见 result4-3.xlsx；全天购电费不含紧急购电费。

表 33 问题 4-3：指定日期的调整购电量（按题面表 1 的时段）

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
10:00-10:10	0.00	0.00	0.00	0.00
12:00-12:10	572.75	128.61	517.53	978.49
14:00-14:10	0.00	0.00	0.00	0.00
16:00-16:10	793.71	196.32	839.63	1050.88
18:00-18:10	698.68	502.31	784.23	740.99
20:00-20:10	0.00	0.00	0.00	0.00
全天购电量	69360.14	33996.66	75063.55	96485.00
全天购电费	44338.96	17353.59	50140.12	74804.21

表中六个时段列为 6:00、12:00、18:00 三次调整后的最终购电量，其余时段见 result4-3.xlsx。

表 34 问题 4-3：指定日期储能设备的充放电量与储电量

时间段	2025.03.20	2025.06.21	2025.09.23	2025.12.21
0:00-4:00	9850.45 / 0.00	314.74 / 514.88	7344.06 / 0.00	7555.15 / 180.37
4:00-8:00	328.09 / 6548.74	1428.50 / 1708.93	3793.67 / 5599.83	1718.11 / 1628.22
8:00-12:00	5083.61 / 2097.50	10019.56 / 0.00	6036.82 / 1896.62	5570.39 / 6747.54
12:00-16:00	5897.52 / 254.72	0.00 / 0.00	2959.24 / 403.62	6588.22 / 2228.48
16:00-20:00	12.16 / 5780.61	0.00 / 5652.32	3.00 / 5697.33	9.65 / 5320.23
20:00-24:00	464.75 / 2913.91	802.13 / 2500.68	649.27 / 2995.44	898.66 / 2842.53
0:00 储电量	1646.24	2684.39	1457.45	3494.22
24:00 储电量	1568.65	2463.02	1728.41	2547.75

每格为"充电量 / 放电量"，单位 kWh；为 6:00、12:00、18:00 三次调整后的最终调度。

表 35 问题 4-3：指定日期的紧急购电

日期	紧急购电时间段与购电量 (kWh)	当日合计 (kWh)
2025-03-20	8:50-10:00: 679.8770; 20:30-20:40: 60.9708	740.8478
2025-06-21	无	0.0000
2025-09-23	20:10-20:20: 10.2344; 20:30-20:40: 48.7102; 20:50-21:00: 30.4276	89.3722
2025-12-21	7:50-8:00: 5.3232	5.3232
四天合计		835.5432

全年 167 天存在紧急购电，全年紧急购电量合计 88054.5218 kWh。

面向新场景的自进化预测框架

前六章解决"给定 2025 年这一年的数据如何做出最优购电决策"；本章回答推广性问题：这套方法离开本场景之后还能不能用——换地点或换生活习惯时模型需要多久恢复精度，进入第二年精度能否高于第一年。

7.1 三层分解与自适应放权

§2.2 的结构解剖指出，三条序列的驱动因子完全不同。据此把序列按可迁移性分解为三层：

$$y_d(t) = Lev_d \times Seas[m(d)] \times ratio[c(d)] \times Shp[c(d)](t)$$

分层的依据是**量纲**。日内形状 *Shp*、年内形状 *Seas* 与日型比例 *ratio* 都是无量纲量（各自以均值归一），与站点规模无关，可整体作为先验搬运；水平 *Lev* 有**量纲**，必须现场在线估计；光伏的天文层 $kH_0(\varphi, \delta)$ 由物理规律给定，只需知道纬度即可零样本使用——§2.2.4 已实测该层单独解释 81.9% 的光伏年变化。形状参数按"本地样本量 vs 先验强度"加权（收缩估计，文献 [10]），且先验权重随本地与先验的偏离度自动下降，使先验错误时能快速放权；该机制的定量作用见 §7.5 与附录 C.6。

7.2 纬度的物理辨识

纬度是**地理常数**，查地图即得。若确实未知，可由光伏的起止时刻反解： $\cos \omega_s = -\tan \varphi \tan \delta$ ，用连续 7 天数据取中位数即可。实测在 23.5° 、 38.0° 、 40.0° 三个真值上分别估得 26.8° 、 40.1° 、 41.2° ，误差不超过 3.3° ，足以把天文层的季节形状定准。

7.3 变体站点试验台与结果

主站点只有一年数据，无法直接检验"换地点"与"第二年"。本文先从原始数据反推其生成机制（§2.1.3 已声明该类数据属二次分析产物），再据此生成四个变体站点做**逐日走查**（每天只使用截至当日的观测）：**V0** 同地点换一年天气实现；**V1** 纬度由 23.5° 换至 40° 、季节振幅 $\times 0.75$ ；**V2** 低日型改为周六日、日内峰由 8:00 后移至 19:00；**V3** 两者同时改变。对照三种部署模式：**迁移（带先验）**、**纯冷启动**，以及**先知参照**（用该变体自身全年数据直接估计月均水平，作为参照而非下界）。

表 36 全期精度与相对纯冷启动的改善

变体	变量	迁移	纯冷启动	先知参照	改善
V0 第二年	负载	271.8	295.1	240.7	+8.6%
	光伏	78.2	84.8	97.8	+8.5%
	电价	50.8	53.9	48.3	+6.0%
V1 换地点	负载	251.8	280.4	224.3	+11.4%
	光伏	71.5	76.6	119.7	+7.1%
	电价	50.1	52.5	46.9	+4.8%
V2 换习惯	负载	280.6	301.8	223.1	+7.5%
	光伏	75.8	85.9	97.1	+13.3%
	电价	51.9	55.5	47.2	+7.0%
V3 换地点+习惯	负载	294.5	300.7	263.3	+2.1%
	光伏	70.5	77.7	114.0	+10.2%
	电价	50.6	51.7	46.9	+2.1%

（负载与光伏单位为 kW，电价单位为 10^{-3} 元/kWh。）

收敛速度：光伏预测追上先知参照所需天数为 **V0 0 天**、**V1 2 天**、**V3 2 天**、**V2 335 天**；换生活习惯场景的负载适应期约 30 天（第 0—7 天误差 1175.9 kW，第 7—30 天降至 371.3 kW，第 30—90 天 228.7 kW）。

7.4 一条负结果：换地点时不应"学"纬度

换地点场景下把纬度当作待学参数用网格拟合，光伏全期 MAE 为 **117.4 kW**，不但远差于"迁移 + 已知纬度"的 **71.5 kW**，甚至劣于纯冷启动的 **76.6 kW**。原因与 §2.2.9 完全一致：冬季样本覆盖的太阳

黄经弧段不足以辨识年周期，拟合出的纬度偏差会被天文层放大。正确做法是把纬度作为地理常数输入，或用 §7.2 的物理法在一周内读出。

7.5 本章结论

其一，自进化是有效的：同站点第二年三个变量的精度提升 6%~9%，第 0 天即具备 3%~8% 的零样本精度。其二，换地点可以应对：光伏预测 71.5 kW 优于该站点的纯冷启动（76.6 kW），收敛仅需 2 天，前提是纬度作为地理常数带过去。其三，换生活习惯可以应对：电价精度仍改善 7.0%、光伏改善 13.3%，负载在约 30 天适应期后追上并反超纯冷启动。其四，机制是"分层 + 收缩 + 自适应放权"：形状层可搬运、水平层必须现场学、天文层由物理定律给定，先验错了要能快速放权。

三组消融实验（学习曲线与适应期、自适应放权的适用条件、逐层搬运消融）的完整过程、数据与结论见附录 C.6；其结果已并入上述结论，正文不再展开。

模型检验

8.1 数据泄露检验

这是全文最要紧的一组检验。检验方法是篡改未来数据后重跑，要求已经做出的决策逐位不变。

表 37 数据泄露检验

编号	检验内容	判据	结果
L1	把第 31 天起的附件 2 / 4 实际值全部置零后重新标定	标定出的 8 个常数逐位不变	✓ 通过
L2	破坏第 31 天起实际值后端到端重跑	第 0—30 天的计划、调度、储电量最大绝对差 = 0	✓ 通过 (0.0)
L3	预测面板因果性：改动第 5 天之后的输入	$F[0..4]$ 逐位不变	✓ 通过
L4	附件 1 只在第 0 天使用：把冷启动曲线乘 2	$F[1:]$ 逐位不变，仅 $F[0]$ 变	✓ 通过
L5	方差表因果性：破坏第 60 天起实际值	第 0—60 天的 $\hat{\sigma}$ 表逐位不变	✓ 通过
T1	截断等价（4-2、4-3 各一次）： $D=D_0$ 与 $D=D_1$ 重跑	调度与决策最大差 = 0	✓ 通过
T2	静态无前视： D 不变、破坏 $arr[D_0:]$	调度完全不变	✓ 通过 (0.0)
T3	光伏预报面板因果性	面板前缀不变	✓ 通过

检验的整体含义是：统计期起点之后发生的一切，都不会影响起点之前的任何一个决策。

8.2 独立复核

由独立脚本从结果文件重算，不调用求解器。

表 38 问题 4-2 与 4-3 的独立复核

项目	问题 4-2	问题 4-3
重算总费用（元）	14 995 821.96	14 885 452.71
与求解器输出之差	逐位一致	逐位一致
放电 / 充电	0.80995	0.80995

项目	问题 4-2	问题 4-3
能量守恒最大绝对误差	1.27×10^{-11}	1.36×10^{-11}
储电量日界范围 (kWh)	[1200.00, 5298.61]	[1200.00, 5172.69]
越界次数	0	0
功率越界次数	0	0

8.3 储能运行状况

须区分日界值与日内极值。表 38 中的储电量范围是每日 0:00 与 24:00 的取值，并非日内极值。

重放完整日内轨迹（每天 145 点、共 48430 点）后：轨迹最小值 / 最大值恰好为 1200.00 / 10800.00 kWh，触及上限 10800 的次数在 4-2 与 4-3 分别为 9931 与 10015 次，触及下限 1200 的为 739 与 686 次；轨迹中位数为 7026 / 6976 kWh，充电与放电功率峰值均打到上限 5000.00 kW。

储电量的上下限与充放电功率上限全部被打满，说明这些约束在最优解处是紧的。

日内充满、放光的日循环套利：在电价低谷（夜间与午间光伏大发）充满、在电价高峰（早 8:00、晚 20:30）放掉，故 0:00 与 24:00 看到的都是低位，而日内峰值达满仓 10800 kWh。

8.4 双路径互证

问题一同时用线性规划与储电量离散化动态规划（文献 [7]）求解，结果相差 2.26 元；问题二同时用线性规划与在线滚动求解，结论一致。两条路径不共用代码。

8.5 灵敏度分析

表 39 关键参数与口径的灵敏度

变动	对全年费用的影响
储能效率改为往返 90%（单向 94.87%）	下降约 3%
冗余分位 q 由 88% 改为 80%	上升约 0.4%
冗余分位 q 由 88% 改为 95%	上升约 0.9%
统计窗口 W 由 5 天改为 28 天	上升约 3.5%
取消储能	上升约 24%
取消日型二分类（改用七日分别建模）	上升约 0.6%

冗余裕量对统计窗口长度比对分位更敏感；储能的存在与否是最大的一阶因素。此外，储电量上下限、充放电功率上限、供电量不低于负载三条约束在全部算例中均逐时段满足，越界次数为零（表 38）；“以计划量为基准加罚”与“以成交量为基准补差”两种偏差写法在最优解处给出相同费用（证明见附录 B）。

结论

9.1 四个问题的答案

问题一在全部信息已知时，全天购电费为 35126.95 元，全天购电量 59482.6990 kWh。若取消储能，全天购电费由 35126.95 元升至 48052.05 元，即储能使费用下降 26.90%；其收益并非来自“少买电”，而是来自把购电搬到低价时段——全天购电量几乎不变。线性规划与动态规划两条独立路径互差 2.26 元。

问题二在只用历史数据预测的前提下，全年费用 13700048.95 元，其中紧急购电 665904.27 元（占 4.86%）、紧急购电量 110410.16 kWh。可部署方案的四个参数 $(\alpha, \beta, W, q) = (0.10, 0.90, 5 \text{ 天}, 88\%)$ 全部由 1 月标定，不含任何未来信息。

问题三引入 6:00、12:00、18:00 三次调整后，全年费用 14248107.47 元。对题面追问的回答是：需要引入其他时刻的预报，调整的净收益为 67279.55 元（0.47%），紧急购电量下降 26.9%；但这一收益高度依赖权重标定的准确度——只有把标定窗口由固定 1 月改为扩展窗之后，调整才稳定地省钱。

问题四在波动电价下重算，4-2 为 14995821.96 元，4-3 为 14885452.71 元。引入三次调整使费用下降 110369 元（0.74%）、紧急购电量下降 14.8%。与完美信息理论下界 12779221.24 元相比，4-2 尚余 17.3% 的空间，该空间主要与预测误差和冗余裕量有关，而非优化结构——这意味着在现有储能参数下，继续优化模型的收益有限，提高预测精度是最直接的改进方向。

相邻两问的代价一目了然：电价由已知变为需预测，使问题二升至 4-2（+9.46%）、问题三升至 4-3（+4.47%）；而引入日内三次调整，使 4-3 相对 4-2 回落 110369 元。

9.2 方法上最值得强调的三点

其一，信息边界的处理是本题的实质难点，而非优化本身。题目给出的附件 2、附件 4 是实际发生值，计划却只能在 0:00 做出，二者混用即构成数据泄露。本文有一个锋利的自检判据：正确的问题二模型，其紧急购电费必然显著大于零；本文的建模过程正是靠这条判据发现并纠正了最初的泄露问题。

其二，1 月窗口能识别什么、不能识别什么，决定了标定协议的形态。本文逐项核验后发现：日型分组、星期效应、日内形状与电价—光伏耦合在 1 月即充分暴露；而季节水平、节气幅度与相位在 1 月结构性不可识别。因此本文没有采用“标定一次即冻结”的做法，而是让参数随运行逐日/逐月重标定。实测模型确实自我进化——水平跟踪权重在第 180 天由 1.00 自动降至 0.90，冗余窗口由 21 天缩至 3 天；而第 31 天的重标定结果与冻结口径逐位一致，说明预热期没有被污染。

其三，数据本身的结构值得先花力气弄清。三条序列的驱动因子完全不同：电价与负载受星期结构支配（低日型恰为周五、周六，一个二值变量即解释 R^2 达 0.695 与 0.893），光伏与星期无关而服从太阳几何（两参数模型即达 $R^2 = 0.819$ ，且可外推到未见过的月份）。把这三条结论分开，预测模型才能各按其规律建模。

9.3 不足与后续工作

其一，电价预测仍是最大的误差来源：4-2 距理论下界的 17.3% 中，电价预测误差贡献主要部分。试验表明用当天已发生的电价对剩余时段做水平校正在三个时点全部改善（剩余时段 MAE 由 0.0756/0.0700/0.0636 降至 0.0673/0.0590/0.0594），但需引入收缩系数、且题目未给电价预报，本文取更保守的口径，该改进留作后续工作。

其二，冗余裕量按逐时段独立给出，未考虑时段间误差相关，会低估极端联合误差。

其三，第 7 章的变体站点数据由原始数据反推的生成机制合成，不是真实站点数据，故该章结论应理解为“框架机制有效”，而非“在任意真实站点上都能达到该精度”。

AI 工具使用声明

本参赛队在竞赛过程中使用了 AI 工具，主要用于题意与文献核对、算法代码实现与调试、批量实验执行、结果汇总与论文文字组织，详细使用情况见支撑材料中的“AI 工具使用详情.pdf”。

需要说明的是：本文的模型设计、算法与实验设计、核心猜想与论文结构均由参赛队员完成；题意与文献核对、代码实现与调试、批量实验执行、结果汇总与文字组织由参赛队员与 AI 协作完成。AI 参与完成的每一项结果与过程都经参赛队员监督审查，不存在未经人工审核即采用的内容；AI 生成的每一段代码均实际运行、结果由独立脚本重算核对，论文中的全部数值都可回溯到支撑材料中的脚本与运行回执。

参考文献

- [1] Arrow K J, Harris T, Marschak J. Optimal Inventory Policy[J]. *Econometrica*, 1951, 19(3): 250-272. (报童模型)
- [2] Qin Y, Wang R, Vakharia A J, Chen Y, Seref M M H. The newsvendor problem: Review and directions for future research[J]. *European Journal of Operational Research*, 2011, 213(2): 361-374. (报童问题综述)
- [3] Porteus E L. *Foundations of Stochastic Inventory Theory*[M]. Stanford: Stanford University Press, 2002. (报童模型的临界分位解)
- [4] Zipkin P H. *Foundations of Inventory Management*[M]. New York: McGraw-Hill, 2000. (单周期报童型模型)
- [5] Scarf H. A min-max solution of an inventory problem[M]//Arrow K J, Karlin S, Scarf H. *Studies in the Mathematical Theory of Inventory and Production*. Stanford University Press, 1958: 201-209. (分布未知时的报童解)
- [6] Dantzig G B. *Linear Programming and Extensions*[M]. Princeton: Princeton University Press, 1963. (线性规划)
- [7] Bellman R. *Dynamic Programming*[M]. Princeton: Princeton University Press, 1957. (动态规划)
- [8] Brown R G. *Smoothing, Forecasting and Prediction of Discrete Time Series*[M]. Englewood Cliffs: Prentice-Hall, 1963. (指数平滑)
- [9] Bates J M, Granger C W J. The Combination of Forecasts[J]. *Operational Research Quarterly*, 1969, 20(4): 451-468. (逆方差加权融合)
- [10] James W, Stein C. Estimation with Quadratic Loss[C]//*Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. University of California Press, 1961: 361-379. (收缩估计)
- [11] Duffie J A, Beckman W A. *Solar Engineering of Thermal Processes*[M]. 4th ed. Hoboken: John Wiley & Sons, 2013. (水平面天文日辐射量)
- [12] Meeus J. *Astronomical Algorithms*[M]. 2nd ed. Richmond: Willmann-Bell, 1998. (太阳视黄经算法)
- [13] 国务院办公厅. 关于 2025 年部分节假日安排的通知[Z]. 2024-11-12. (法定节假日安排)
- [14] 盛骤, 谢式千, 潘承毅. *概率论与数理统计*[M]. 5 版. 北京: 高等教育出版社, 2019. (分位数; 方差、标准差与相关系数; 一元线性回归)

附录

附录 A 结果文件说明

五个结果文件均按附件 5 的模板生成：工作表名与第 1 行表头逐字保留模板原文，仅填入数据，未增删任何列。

表 40 结果文件清单

文件	工作表	填写范围	对应问题
result1.xlsx	计划购电量、充放电量	单日 144 段	问题一
result2.xlsx	计划购电量、充放电量、紧急购电量	2025.2.1—12.31	问题二
result3.xlsx	计划购电量、调整购电量、充放电量、紧急购电量	2025.2.1—12.31	问题三
result4-2.xlsx	计划购电量、充放电量、紧急购电量	2025.2.1—12.31	问题 4-2
result4-3.xlsx	计划购电量、调整购电量、充放电量、紧急购电量	2025.2.1—12.31	问题 4-3

需要说明三点。其一，模板的时间标签比自然时序右移一格：result1 模板的标签自"0:10-0:20"起、至 "0:00+1-0:10+1" 止，result2 及其后各模板的标签自 "0:10-0:20" 起、至 "0:00-0:10+1" 止。本文一律逐字保留模板标签，数据则自第 2 列起按 0:00—0:10、0:10—0:20、……、23:50—24:00 的自然顺序填写；五个文件采用同一口径。其二，模板首列表头为"日期\时间"，故数据自第 2 列（B 列）起排。其三，"充放电量"表在 result1 中不含日期列（单日），在 result2 及其后各文件中含日期列，其行结构为每天 6 行（每 4 小时一行），0:00 储电量记在该日首行、24:00 储电量记在次行。

附录 B 两种偏差计费写法的等价性

问题三的偏差费用有两种写法，二者在任意可行解处取值相同。

写法一（以成交量为基准补差）：目标函数中 $\lambda_t p_t^{adj} + 0.5 \lambda_t u_t + 0.5 \lambda_t v_t$ ，约束 $p_t^{adj} - v_t + u_t = p_t^{plan}$ ， $u_t, v_t \geq 0$ 。本文实现采用此式，因为 u_t, v_t 可直接作为决策变量交给线性规划求解器。

写法二（以计划量为基准加罚）：目标函数中 $\lambda_t \min(p_t^{plan}, p_t^{adj}) + 0.5 \lambda_t \max(0, p_t^{plan} - p_t^{adj}) + 1.5 \lambda_t \max(0, p_t^{adj} - p_t^{plan})$ 。

命题 两式在任意可行解处取值相同，且都等于 $\lambda_t p_t^{plan} - 0.5 \lambda_t u_t + 1.5 \lambda_t v_t$ 。

证明 分两种情形。

情形一， $p_t^{adj} \geq p_t^{plan}$ 。此时 $u_t = 0$ ， $v_t = p_t^{adj} - p_t^{plan}$ 。写法一 $= \lambda_t p_t^{adj} + 0.5 \lambda_t v_t = \lambda_t (p_t^{plan} + v_t) + 0.5 \lambda_t v_t = \lambda_t p_t^{plan} + 1.5 \lambda_t v_t$ ；写法二 $= \lambda_t p_t^{plan} + 1.5 \lambda_t v_t$ （计划量按 λ_t 计、超出部分按 $1.5 \lambda_t$ 计），二者相同。

情形二， $p_t^{adj} < p_t^{plan}$ 。此时 $v_t = 0$ ， $u_t = p_t^{plan} - p_t^{adj}$ 。写法一 $= \lambda_t p_t^{adj} + 0.5 \lambda_t u_t = \lambda_t (p_t^{plan} - u_t) + 0.5 \lambda_t u_t = \lambda_t p_t^{plan} - 0.5 \lambda_t u_t$ ；写法二 $= \lambda_t p_t^{adj} + 0.5 \lambda_t u_t$ （成交量按 λ_t 计、短交部分按 $0.5 \lambda_t$ 计），二者恒等。□

系数的含义（易错处）：题面规定的 1.5 倍与 0.5 倍都以"交易时刻电价"为基数，且 1.5 倍已包含基础电价。因此当目标函数以成交量为基准时，对上调量只应再加 $0.5 \lambda_t v_t$ ——若误加 $1.5 \lambda_t v_t$ ，计划量以上的那部分电就会被按 2.5 倍计费，与题面不符；此时应改用以计划量为基准的写法二。

附录 C 关键计算说明

C.1 储能效率的累计校验。对任一决策序列，全年累计量满足

$$\eta C - \frac{D}{\eta} = S_{末} - S_{初}$$

其中 C, D 分别为全年充电与放电总量。本文全部算例均满足该式，残差在 10^{-11} 量级。

C.2 太阳黄经与节气时刻的计算。 采用 Meeus 低精度太阳视黄经公式

$$L_0 = 280.46646 + 36000.76983 T + 0.0003032 T^2$$

$$M = 357.52911 + 35999.05029 T - 0.0001537 T^2$$

$$\lambda = L_0 + C - 0.00569 - 0.00478 \sin \Omega$$

其中 T 为儒略世纪数， C 为中心差， Ω 为月球升交点黄经。求解 $\lambda = k \cdot 15^\circ$ ($k=0, \dots, 23$) 即得二十四节气时刻。本文的计算结果与紫金山天文台公布值逐分钟吻合，并据此确认题目指定的四个日期 (3.20、6.21、9.23、12.21) 为春分、夏至、秋分、冬至。

C.3 冗余裕量的分位数定义。 采用线性插值分位数：对升序排列的 n 个样本 $x_{(1)} \leq \dots \leq x_{(n)}$ ， q 分位数为 $x_{(j)} + \gamma(x_{(j+1)} - x_{(j)})$ ，其中 $j = \lfloor q(n-1) \rfloor + 1$ ， $\gamma = q(n-1) - j + 1$ 。

C.4 问题四的其余作法（正文从略）。

把电价当作已知的"理想对照"。 若在问题四中允许使用当天实际电价（即放弃信息边界），4-2 的费用降至约 12779221 元——这正是 §6.5.4 中的理论下界。该对照说明**信息边界本身的代价约为 222 万元**，是本文所有口径中最值得强调的一条。**折半校正的日内电价更新。** 曾考虑用当天已发生的电价对剩余时段做水平校正。实测：不做更新时剩余时段 MAE 为 0.0756/0.0700/0.0636 (6/12/18 时)；折半校正后为 0.0673/0.0590/0.0594，三个时点**全部改善**。但该做法需要引入一个收缩系数，本文选择更保守的口径——**电价预测在 0:00 定稿，当天不再修改**，与"题目未给电价预报"的题意一致。该改进留作后续工作。**四季（四立）作为重标定触发点。** 曾把 (W, q) 的重标定时刻设为四立当天（第 33、124、218、310 天，一年 4 次），全年费用 15090484.13 元，比"每月 1 日" (14995821.96 元) 高 0.63%。四季作为触发点过于稀疏：立春后需等到立夏才更新，中间 91 天一直沿用 1 月的最保守参数。真正带来收益的是 (α, β) 的逐日更新——它不需要预先知道四季，而是随季节推进自行学习。

C.5 其余作法（历史方案与调优记录）。

七分类与二分类的比较。 早期做法把星期分成七类各自建立基线，实测二分类（周五/周六为一类）优于七分类：七类的样本被切碎，每类的历史均值估计方差更大，反而放大了预测误差。这一结论与 §2.2.2 的"星期结构是纯二元的"一致。**持久法与滑动平均。** 若点预测退化为"取前一天的实测曲线"（持久法），全年费用上升约 3.1%；退化为"取前 7 天平均"，上升约 1.8%；二者都未利用星期结构，故劣于本文的加权预测。**动态规划独立复算。** 另按储电量离散化构造动态规划（文献 [7]），最优值 35129.21 元，与线性规划相差 2.26 元；两者不共用代码，差异仅来自离散化误差。**储能价值的分解。** 取消储能后全天购电费升至 48052.05 元，储能带来的下降为 26.90%，且收益并非来自"少买电"——全天购电量几乎不变，而是来自**把购电搬到低价时段的套利**。**效率口径的替代取值。** 若把"90%"理解为往返效率（单向 94.87%），全天购电费降至约 34000 元量级；本文取单向口径，理由见假设 1。

C.6 第 7 章三组消融实验的完整过程与数据。

（一）学习曲线与适应期。 逐日统计迁移模型（带先验、纬度取变体真值）的日均值预测误差并分段汇总，得：换生活习惯场景 (V2) 第 0—7 天负载误差 1175.9 kW、第 7—30 天 371.3 kW、第 30—90 天 228.7 kW、全期 280.6 kW，即**适应期约 30 天**；换地点 (V1) 第 0—7 天光伏误差 202.2 kW、第 7—30 天 26.5 kW，第 0 天为 567.7 kW（因先验的季节振幅在该站点偏大 25%），换地点+换习惯 (V3) 第 0 天为 540.1 kW。收敛判据为"连续 21 天滚动误差稳定进入先知参照

1.25 倍以内", 所需天数为 v_0 0 天、 v_1 2 天、 v_3 2 天、 v_2 335 天; 门槛收紧到 1.10 倍时结论不变 (v_0 为 337 天, 其余相同)。

(二) 自适应放权的适用条件。 关闭"本地与先验偏离越大、先验权重越小"的机制 (先验权重恒为 $n_0=7$ 天) 后重跑: 只换生活习惯的 v_2 几乎无变化 (全期负载 280.6 对 278.0 kW、光伏 87.8 对 89.2 kW); 而换地点且换习惯的 v_3 , 第 0—7 天光伏误差由 171.4 kW 升至 405.3 kW, 收敛天数由 2 天延长到 294 天。全期指标上自适应并非总是更优 (v_3 全期 124.4 对 90.8 kW), 说明它以牺牲部分长期精度换取快速摆脱错误先验。

(三) 逐层搬运消融。 在纬度未知、须在线辨识的条件下, 分别只给形状层先验、只给天文层先验、给全层先验与不给先验 (纯冷启动), 四个变体的全期光伏预测 MAE 如下。

表 41 逐层搬运消融: 全期光伏预测 MAE (kW, 纬度未知)

变体	全层先验	仅形状层 (天文层置 旧纬度)	仅天文层 (形状层置 中性)	纯冷启动
v_0 原站点·第二年	89.1	89.1	95.1	84.8
v_1 换地点	117.4	117.4	118.9	76.6
v_2 换生活习惯	87.8	87.8	94.3	85.9
v_3 换地点+换习惯	124.4	124.4	121.9	77.7

可见: 形状层先验稳定有益 (置为中性后 v_0 、 v_2 的光伏误差分别上升 6.7%、7.5%, 对负载与电价的作用在 $\pm 4\%$ 以内); 天文层先验只在纬度已知时才有意义 (纬度未知时模型会用自己的"光伏起止时刻"反解纬度并覆盖先验, 故"全层先验"与"仅形状层先验"结果完全相同); 换地点时整体搬运反不如纯冷启动 (v_1 : 117.4 对 76.6; v_3 : 124.4 对 77.7), 与正文 §7.4 的负结果互为印证。实验脚本与逐日记录见支撑材料 \代码\06_可迁移框架\。

附录 D 源程序代码与支撑材料文件清单

本附录收录建模与求解所用的全部源程序, 共 92 个 Python 文件。

全部代码为 Python 3 实现, 依赖 numpy、scipy (线性规划求解器 HiGHS) 与 openpyxl。

全部代码不含任何本机绝对路径: 项目根目录一律由环境变量 CUMCM_ROOT 给出 (缺省为当前目录)。复现时先设置 CUMCM_ROOT 指向解压后的项目根目录, 再按支撑材料 复现/复现说明.md 执行; 数据文件、参数记录与一键复现脚本见支撑材料对应目录。

D.1 支撑材料文件清单

支撑材料共计 457 个文件, 按目录列出如下。

#	文件
1	README.md
2	交付清单.md
3	支撑材料/AI 工具使用详情.docx
4	支撑材料/AI 工具使用详情.pdf
5	支撑材料/代码\01_问题一\q1-baseline.json
6	支撑材料/代码\01_问题一\q1-dp.json
7	支撑材料/代码\01_问题一\q1-eta085.json
8	支撑材料/代码\01_问题一\q1-eta095.json
9	支撑材料/代码\01_问题一\q1-heuristic.json
10	支撑材料/代码\01_问题一\q1-lp.json
11	支撑材料/代码\01_问题一\q1-tight.json
12	支撑材料/代码\01_问题一\q12_reverify.py
13	支撑材料/代码\01_问题一\q1_arbitrage.py
14	支撑材料/代码\01_问题一\q1_dp.py
15	支撑材料/代码\01_问题一\q1_lp.py
16	支撑材料/代码\01_问题一\q1_no_storage.py

#	文件
17	支撑材料/代码/02_问题二/q2_accuracy.py
18	支撑材料/代码/02_问题二/q2_bounds.py
19	支撑材料/代码/02_问题二/q2_caliber_compare.py
20	支撑材料/代码/02_问题二/q2_dp.py
21	支撑材料/代码/02_问题二/q2_final.py
22	支撑材料/代码/02_问题二/q2_firstmonth_calib.py
23	支撑材料/代码/02_问题二/q2_firstmonth_full.py
24	支撑材料/代码/02_问题二/q2_forecast.py
25	支撑材料/代码/02_问题二/q2_leaktest.py
26	支撑材料/代码/02_问题二/q2_lp.py
27	支撑材料/代码/02_问题二/q2_margin.py
28	支撑材料/代码/02_问题二/q2_no_storage.py
29	支撑材料/代码/02_问题二/q2_online.py
30	支撑材料/代码/02_问题二/q2_perfect.py
31	支撑材料/代码/02_问题二/q2_persistence.py
32	支撑材料/代码/02_问题二/q2_replica_dp.py
33	支撑材料/代码/02_问题二/q2_replica_lp.py
34	支撑材料/代码/02_问题二/q2_result2.py
35	支撑材料/代码/02_问题二/q2_robust.py
36	支撑材料/代码/02_问题二/q2_season_blocks.py
37	支撑材料/代码/02_问题二/q2_season_chain.py
38	支撑材料/代码/02_问题二/q2_seasonal.py
39	支撑材料/代码/02_问题二/q2_seasonal_model.py
40	支撑材料/代码/02_问题二/q2_solve_and_gen.py
41	支撑材料/代码/02_问题二/q2_stages.py
42	支撑材料/代码/02_问题二/q2_variants.py
43	支撑材料/代码/03_问题三/q34_grid.py
44	支撑材料/代码/03_问题三/q34_solver.py
45	支撑材料/代码/03_问题三/q3_ablation.py
46	支撑材料/代码/03_问题三/q3_baseline.py
47	支撑材料/代码/03_问题三/q3_constraint_check.py
48	支撑材料/代码/03_问题三/q3_fc_accuracy_B.py
49	支撑材料/代码/03_问题三/q3_fc_diag_A.py
50	支撑材料/代码/03_问题三/q3_fc_export_A.py
51	支撑材料/代码/03_问题三/q3_fc_leaktest_A.py
52	支撑材料/代码/03_问题三/q3_fc_leaktest_B.py
53	支撑材料/代码/03_问题三/q3_fc_report_A.py
54	支撑材料/代码/03_问题三/q3_fc_rollperiod.py
55	支撑材料/代码/03_问题三/q3_fc_rollsigma.py
56	支撑材料/代码/03_问题三/q3_fc_rollsolver.py
57	支撑材料/代码/03_问题三/q3_fc_rolltest.py
58	支撑材料/代码/03_问题三/q3_fc_sigma_B.py
59	支撑材料/代码/03_问题三/q3_fc_solver_A.py
60	支撑材料/代码/03_问题三/q3_fc_solver_B.py
61	支撑材料/代码/03_问题三/q3_fc_summary_B.py
62	支撑材料/代码/03_问题三/q3_fc_verify_A.py
63	支撑材料/代码/03_问题三/q3_fc_xlsx_B.py
64	支撑材料/代码/03_问题三/q3_forecast_A.py
65	支撑材料/代码/03_问题三/q3_forecast_B.py
66	支撑材料/代码/03_问题三/q3_mpc.py
67	支撑材料/代码/03_问题三/q3_realtime_solver.py
68	支撑材料/代码/03_问题三/q3_sens.py
69	支撑材料/代码/03_问题三/q3_stochastic.py
70	支撑材料/代码/03_问题三/q3_validate_full.py
71	支撑材料/代码/03_问题三/q3_vintages_A.py

#	文件
72	支撑材料/代码/03_问题三/q3_vintages_B.py
73	支撑材料/代码/04_问题四/q4_fc_solver.py
74	支撑材料/代码/04_问题四/q4_realtime_solver.py
75	支撑材料/代码/04_问题四/q4_season_chain.py
76	支撑材料/代码/04_问题四/q4_solve.py
77	支撑材料/代码/04_问题四/q4_solver.py
78	支撑材料/代码/04_问题四/q4_strict.py
79	支撑材料/代码/04_问题四/q4_validate.py
80	支撑材料/代码/05_数据分析/jieqi.py
81	支撑材料/代码/05_数据分析/q4_diag.py
82	支撑材料/代码/05_数据分析/q4_diag2.py
83	支撑材料/代码/05_数据分析/q4_diag3.py
84	支撑材料/代码/05_数据分析/q4_diag4.py
85	支撑材料/代码/05_数据分析/q4_diag5.py
86	支撑材料/代码/05_数据分析/q4_diag6.py
87	支撑材料/代码/05_数据分析/q4_diag7.py
88	支撑材料/代码/05_数据分析/q4_diag8.py
89	支撑材料/代码/06_可迁移框架/q4_selftransfer.py
90	支撑材料/代码/06_可迁移框架/q4_tsef_e2b.py
91	支撑材料/代码/06_可迁移框架/q4_tsef_e2b_输出.txt
92	支撑材料/代码/06_可迁移框架/q4_tsef_extra.py
93	支撑材料/代码/06_可迁移框架/q4_tsef_extra_results.json
94	支撑材料/代码/06_可迁移框架/q4_tsef_extra_输出.txt
95	支撑材料/代码/06_可迁移框架/q4_tsef_testbed.py
96	支撑材料/代码/07_交付物生成/make_check_tables_v3.py
97	支撑材料/代码/07_交付物生成/q3_fc_write_check_table.py
98	支撑材料/代码/07_交付物生成/q3_fc_write_result3.py
99	支撑材料/代码/07_交付物生成/q42_decomp_check.py
100	支撑材料/代码/07_交付物生成/q42_delivered_check.py
101	支撑材料/代码/07_交付物生成/q42_lb_figures.py
102	支撑材料/代码/07_交付物生成/q42_lb_verify.py
103	支撑材料/代码/07_交付物生成/q42_lower_bound.py
104	支撑材料/代码/07_交付物生成/q42_year_check.py
105	支撑材料/代码/07_交付物生成/q4_write_result42.py
106	支撑材料/代码/07_交付物生成/q4_write_result43.py
107	支撑材料/图表/README.md
108	支撑材料/图表/figure_sources.json
109	支撑材料/图表/模型结果/fig01_day_profile.png
110	支撑材料/图表/模型结果/fig02_year_heatmap.png
111	支撑材料/图表/模型结果/fig03_weekday_effect.png
112	支撑材料/图表/模型结果/fig04_q1_dispatch.png
113	支撑材料/图表/模型结果/fig05_arbitrage.png
114	支撑材料/图表/模型结果/fig05_q2_season_features.png
115	支撑材料/图表/模型结果/fig06_q1_sensitivity.png
116	支撑材料/图表/模型结果/fig06_q2_stage_progress.png
117	支撑材料/图表/模型结果/fig07_q2_forecast.png
118	支撑材料/图表/模型结果/fig08_q2_error.png
119	支撑材料/图表/模型结果/fig09_q2_margin_scan.png
120	支撑材料/图表/模型结果/fig10_q2_schemes.png
121	支撑材料/图表/模型结果/fig11_q2_accuracy.png
122	支撑材料/图表/模型结果/fig12_q2_monthly.png
123	支撑材料/图表/模型结果/fig13_q2_soc_heatmap.png
124	支撑材料/图表/模型结果/fig14_q2_daily.png
125	支撑材料/图表/模型结果/fig15_q2_emergency_profile.png
126	支撑材料/图表/模型结果/fig16_q3_window.png

#	文件
127	支撑材料/图表/模型结果/fig17_q3_adjust_day.png
128	支撑材料/图表/模型结果/fig18_q3_monthly.png
129	支撑材料/图表/模型结果/fig19_price4_heatmap.png
130	支撑材料/图表/模型结果/fig20_price4_monthly.png
131	支撑材料/图表/模型结果/fig21_summary.png
132	支撑材料/图表/模型结果/fig22_q3_dates.png
133	支撑材料/图表/模型结果/fig23_constraints.png
134	支撑材料/图表/模型结果/fig24_framework.png
135	支撑材料/图表/模型结果/fig25_replication.png
136	支撑材料/图表/模型结果/figA_technical_route.png
137	支撑材料/图表/模型结果/figA_technical_route.svg
138	支撑材料/图表/模型结果/figA_technical_route_transparent.png
139	支撑材料/图表/模型结果/figA_technical_route_transparent.svg
140	支撑材料/图表/模型结果/figB_information_timeline.png
141	支撑材料/图表/模型结果/figB_information_timeline.svg
142	支撑材料/图表/模型结果/figC_three_layer_transfer.png
143	支撑材料/图表/模型结果/figC_three_layer_transfer.svg
144	支撑材料/图表/模型结果/figD_variant_testbed.png
145	支撑材料/图表/模型结果/figD_variant_testbed.svg
146	支撑材料/图表/模型结果/figE_january_identifiability.png
147	支撑材料/图表/模型结果/figE_january_identifiability.svg
148	支撑材料/图表/模型结果/q42_lb_day_2025-07-15.png
149	支撑材料/图表/模型结果/q42_lb_ladder.png
150	支撑材料/图表/模型结果/q42_lb_monthly.png
151	支撑材料/图表/模型结果/q42_sec21_intraday.png
152	支撑材料/图表/模型结果/q42_sec21_january.png
153	支撑材料/图表/模型结果/q42_sec21_monthly.png
154	支撑材料/图表/模型结果/q42_sec21_price_pv.png
155	支撑材料/图表/模型结果/q42_sec21_pv_geometry.png
156	支撑材料/图表/模型结果/q42_sec21_r2.png
157	支撑材料/图表/模型结果/q42_sec21_weekday.png
158	支撑材料/复现/复现说明.md
159	支撑材料/实验记录/_baseline_check_A/metrics.json
160	支撑材料/实验记录/_baseline_check_B/metrics.json
161	支撑材料/实验记录/_baseline_orig_B/metrics.json
162	支撑材料/实验记录/_e3_repro/metrics.json
163	支撑材料/实验记录/_harness_check_A/metrics.json
164	支撑材料/实验记录/_q4_t1/metrics.json
165	支撑材料/实验记录/_q4_t2/metrics.json
166	支撑材料/实验记录/_s_eb_half/metrics.json
167	支撑材料/实验记录/_s_eb_k0/metrics.json
168	支撑材料/实验记录/_s_eb_k10/metrics.json
169	支撑材料/实验记录/_s_eb_k124/metrics.json
170	支撑材料/实验记录/_s_eb_k31/metrics.json
171	支撑材料/实验记录/_s_eb_k62/metrics.json
172	支撑材料/实验记录/_s_lt_eb_half_a/metrics.json
173	支撑材料/实验记录/_s_lt_eb_half_b/metrics.json
174	支撑材料/实验记录/_s_lt_eb_k0_a/metrics.json
175	支撑材料/实验记录/_s_lt_eb_k0_b/metrics.json
176	支撑材料/实验记录/_s_lt_eb_k31_a/metrics.json
177	支撑材料/实验记录/_s_lt_eb_k31_b/metrics.json
178	支撑材料/实验记录/_s_repro_eb0_b/metrics.json
179	支撑材料/实验记录/_s_repro_eb31_a/metrics.json
180	支撑材料/实验记录/_s_repro_eb31_b/metrics.json

#	文件
181	支撑材料/实验记录/_s_repro_ebhalf_b/metrics.json
182	支撑材料/实验记录/_s_rl_a/metrics.json
183	支撑材料/实验记录/_s_rl_b/metrics.json
184	支撑材料/实验记录/_s_rl_c/metrics.json
185	支撑材料/实验记录/_s_rr_half/metrics.json
186	支撑材料/实验记录/_s_rr_half_sf122/metrics.json
187	支撑材料/实验记录/_s_rr_half_sf214/metrics.json
188	支撑材料/实验记录/_s_rr_k0/metrics.json
189	支撑材料/实验记录/_s_rr_k31/metrics.json
190	支撑材料/实验记录/_s_rr_pool/metrics.json
191	支撑材料/实验记录/_s_sf_cond_122/metrics.json
192	支撑材料/实验记录/_s_sf_cond_214/metrics.json
193	支撑材料/实验记录/_s_sf_pool_122/metrics.json
194	支撑材料/实验记录/_s_sf_pool_214/metrics.json
195	支撑材料/实验记录/_s_sig_sm_iso/metrics.json
196	支撑材料/实验记录/_s_sig_sm_lin/metrics.json
197	支撑材料/实验记录/_s_sig_sm_log/metrics.json
198	支撑材料/实验记录/_s_sig_sm_pow/metrics.json
199	支撑材料/实验记录/_s_sig_sm_sqrt/metrics.json
200	支撑材料/实验记录/_s_smooth/metrics.json
201	支撑材料/实验记录/_s_tod_cond/metrics.json
202	支撑材料/实验记录/_s_tod_ctl/metrics.json
203	支撑材料/实验记录/q1-arbitrage-heuristic/metrics.json
204	支撑材料/实验记录/q1-arbitrage-heuristic/receipt.json
205	支撑材料/实验记录/q1-dp-replication/metrics.json
206	支撑材料/实验记录/q1-dp-replication/receipt.json
207	支撑材料/实验记录/q1-eta085/metrics.json
208	支撑材料/实验记录/q1-eta085/receipt.json
209	支撑材料/实验记录/q1-eta095/metrics.json
210	支撑材料/实验记录/q1-eta095/receipt.json
211	支撑材料/实验记录/q1-lp-primary/metrics.json
212	支撑材料/实验记录/q1-lp-primary/receipt.json
213	支撑材料/实验记录/q1-no-storage-baseline/metrics.json
214	支撑材料/实验记录/q1-no-storage-baseline/receipt.json
215	支撑材料/实验记录/q1-tight-bounds/metrics.json
216	支撑材料/实验记录/q1-tight-bounds/receipt.json
217	支撑材料/实验记录/q2-ablation2/metrics.json
218	支撑材料/实验记录/q2-ablation2/receipt.json
219	支撑材料/实验记录/q2-accuracy/metrics.json
220	支撑材料/实验记录/q2-accuracy/receipt.json
221	支撑材料/实验记录/q2-boot-01b/metrics.json
222	支撑材料/实验记录/q2-boot-01b/receipt.json
223	支撑材料/实验记录/q2-boot-02b/metrics.json
224	支撑材料/实验记录/q2-boot-02b/receipt.json
225	支撑材料/实验记录/q2-boot-03b/metrics.json
226	支撑材料/实验记录/q2-boot-03b/receipt.json
227	支撑材料/实验记录/q2-boot-04b/metrics.json
228	支撑材料/实验记录/q2-boot-04b/receipt.json
229	支撑材料/实验记录/q2-boot-05b/metrics.json
230	支撑材料/实验记录/q2-boot-05b/receipt.json
231	支撑材料/实验记录/q2-boot-06b/metrics.json
232	支撑材料/实验记录/q2-boot-06b/receipt.json
233	支撑材料/实验记录/q2-boot-07b/metrics.json
234	支撑材料/实验记录/q2-boot-07b/receipt.json
235	支撑材料/实验记录/q2-boot-08b/metrics.json

#	文件
236	支撑材料/实验记录/q2-boot-08b/receipt.json
237	支撑材料/实验记录/q2-boot-09b/metrics.json
238	支撑材料/实验记录/q2-boot-09b/receipt.json
239	支撑材料/实验记录/q2-boot-10b/metrics.json
240	支撑材料/实验记录/q2-boot-10b/receipt.json
241	支撑材料/实验记录/q2-caliber-compare/metrics.json
242	支撑材料/实验记录/q2-caliber-compare/receipt.json
243	支撑材料/实验记录/q2-datalen-030/metrics.json
244	支撑材料/实验记录/q2-datalen-030/receipt.json
245	支撑材料/实验记录/q2-datalen-060/metrics.json
246	支撑材料/实验记录/q2-datalen-060/receipt.json
247	支撑材料/实验记录/q2-datalen-090/metrics.json
248	支撑材料/实验记录/q2-datalen-090/receipt.json
249	支撑材料/实验记录/q2-datalen-122/metrics.json
250	支撑材料/实验记录/q2-datalen-122/receipt.json
251	支撑材料/实验记录/q2-datalen-212/metrics.json
252	支撑材料/实验记录/q2-datalen-212/receipt.json
253	支撑材料/实验记录/q2-datalen-sum0709/metrics.json
254	支撑材料/实验记录/q2-datalen-sum0709/receipt.json
255	支撑材料/实验记录/q2-dp-planA/metrics.json
256	支撑材料/实验记录/q2-dp-planA/receipt.json
257	支撑材料/实验记录/q2-dp-replication/metrics.json
258	支撑材料/实验记录/q2-dp-replication/receipt.json
259	支撑材料/实验记录/q2-final/metrics.json
260	支撑材料/实验记录/q2-final/receipt.json
261	支撑材料/实验记录/q2-firstmonth-calib/metrics.json
262	支撑材料/实验记录/q2-firstmonth-calib/receipt.json
263	支撑材料/实验记录/q2-firstmonth-full/metrics.json
264	支撑材料/实验记录/q2-firstmonth-full/receipt.json
265	支撑材料/实验记录/q2-grid-hold-mid/metrics.json
266	支撑材料/实验记录/q2-grid-hold-mid/receipt.json
267	支撑材料/实验记录/q2-grid-hold/metrics.json
268	支撑材料/实验记录/q2-grid-hold/receipt.json
269	支撑材料/实验记录/q2-grid-margin3/metrics.json
270	支撑材料/实验记录/q2-grid-margin3/receipt.json
271	支撑材料/实验记录/q2-grid-tune-early/metrics.json
272	支撑材料/实验记录/q2-grid-tune-early/receipt.json
273	支撑材料/实验记录/q2-grid-tune/metrics.json
274	支撑材料/实验记录/q2-grid-tune/receipt.json
275	支撑材料/实验记录/q2-horizon30/metrics.json
276	支撑材料/实验记录/q2-horizon30/receipt.json
277	支撑材料/实验记录/q2-horizon7/metrics.json
278	支撑材料/实验记录/q2-horizon7/receipt.json
279	支撑材料/实验记录/q2-leaktest/metrics.json
280	支撑材料/实验记录/q2-leaktest/receipt.json
281	支撑材料/实验记录/q2-lp-ma28/metrics.json
282	支撑材料/实验记录/q2-lp-ma28/receipt.json
283	支撑材料/实验记录/q2-margin/metrics.json
284	支撑材料/实验记录/q2-margin/receipt.json
285	支撑材料/实验记录/q2-no-storage/metrics.json
286	支撑材料/实验记录/q2-no-storage/receipt.json
287	支撑材料/实验记录/q2-nostorage2/metrics.json
288	支撑材料/实验记录/q2-nostorage2/receipt.json
289	支撑材料/实验记录/q2-perfect-bound2/metrics.json
290	支撑材料/实验记录/q2-perfect-bound2/receipt.json

#	文件
291	支撑材料/实验记录/q2-persistence/metrics.json
292	支撑材料/实验记录/q2-persistence/receipt.json
293	支撑材料/实验记录/q2-primary/metrics.json
294	支撑材料/实验记录/q2-primary/receipt.json
295	支撑材料/实验记录/q2-replica-dp/metrics.json
296	支撑材料/实验记录/q2-replica-dp/receipt.json
297	支撑材料/实验记录/q2-replica-lp/metrics.json
298	支撑材料/实验记录/q2-replica-lp/receipt.json
299	支撑材料/实验记录/q2-result2-planA/metrics.json
300	支撑材料/实验记录/q2-result2-planA/receipt.json
301	支撑材料/实验记录/q2-result2/metrics.json
302	支撑材料/实验记录/q2-result2/receipt.json
303	支撑材料/实验记录/q2-result2b/metrics.json
304	支撑材料/实验记录/q2-result2b/receipt.json
305	支撑材料/实验记录/q2-season-autumn/metrics.json
306	支撑材料/实验记录/q2-season-autumn/receipt.json
307	支撑材料/实验记录/q2-season-blocks/metrics.json
308	支撑材料/实验记录/q2-season-blocks/receipt.json
309	支撑材料/实验记录/q2-season-chain/metrics.json
310	支撑材料/实验记录/q2-season-chain/receipt.json
311	支撑材料/实验记录/q2-season-model/metrics.json
312	支撑材料/实验记录/q2-season-model/receipt.json
313	支撑材料/实验记录/q2-season-spring/metrics.json
314	支撑材料/实验记录/q2-season-spring/receipt.json
315	支撑材料/实验记录/q2-season-winter/metrics.json
316	支撑材料/实验记录/q2-season-winter/receipt.json
317	支撑材料/实验记录/q2-sens-ab-hold/metrics.json
318	支撑材料/实验记录/q2-sens-ab-hold/receipt.json
319	支撑材料/实验记录/q2-sens-ab-tune/metrics.json
320	支撑材料/实验记录/q2-sens-ab-tune/receipt.json
321	支撑材料/实验记录/q2-sens-ab2/metrics.json
322	支撑材料/实验记录/q2-sens-ab2/receipt.json
323	支撑材料/实验记录/q2-stages/metrics.json
324	支撑材料/实验记录/q2-stages/receipt.json
325	支撑材料/实验记录/q2-stress2/metrics.json
326	支撑材料/实验记录/q2-stress2/receipt.json
327	支撑材料/实验记录/q2-stress3/metrics.json
328	支撑材料/实验记录/q2-stress3/receipt.json
329	支撑材料/实验记录/q2-twostage-a1200/metrics.json
330	支撑材料/实验记录/q2-twostage-a1200/receipt.json
331	支撑材料/实验记录/q2-twostage-a3000/metrics.json
332	支撑材料/实验记录/q2-twostage-a3000/receipt.json
333	支撑材料/实验记录/q2-verify-constraints-planA/metrics.json
334	支撑材料/实验记录/q2-verify-constraints/metrics.json
335	支撑材料/实验记录/q2-verify-cost-planA/metrics.json
336	支撑材料/实验记录/q2-verify-cost/metrics.json
337	支撑材料/实验记录/q2-yearjoint2/metrics.json
338	支撑材料/实验记录/q2-yearjoint2/receipt.json
339	支撑材料/实验记录/q3-adjust3/metrics.json
340	支撑材料/实验记录/q3-adjust3/receipt.json
341	支撑材料/实验记录/q3-fc-e1/metrics.json
342	支撑材料/实验记录/q3-fc-e2/metrics.json
343	支撑材料/实验记录/q3-fc-e3/metrics.json
344	支撑材料/实验记录/q3-fc-e3v2-noadj/metrics.json
345	支撑材料/实验记录/q3-fc-e3v2/metrics.json

#	文件
346	支撑材料/实验记录/q3-fc-e4/metrics.json
347	支撑材料/实验记录/q3-grid-hold/metrics.json
348	支撑材料/实验记录/q3-grid-hold/receipt.json
349	支撑材料/实验记录/q3-grid-scan/metrics.json
350	支撑材料/实验记录/q3-grid-scan/receipt.json
351	支撑材料/实验记录/q3-grid-tune/metrics.json
352	支撑材料/实验记录/q3-grid-tune/receipt.json
353	支撑材料/实验记录/q3-noadjust3/metrics.json
354	支撑材料/实验记录/q3-noadjust3/receipt.json
355	支撑材料/实验记录/q3-realtime-adjust/metrics.json
356	支撑材料/实验记录/q3-realtime-noadjust/metrics.json
357	支撑材料/实验记录/q3noadj-grid-hold/metrics.json
358	支撑材料/实验记录/q3noadj-grid-hold/receipt.json
359	支撑材料/实验记录/q3noadj-grid-scan/metrics.json
360	支撑材料/实验记录/q3noadj-grid-scan/receipt.json
361	支撑材料/实验记录/q3noadj-grid-tune/metrics.json
362	支撑材料/实验记录/q3noadj-grid-tune/receipt.json
363	支撑材料/实验记录/q4-2-lower-bound/metrics.json
364	支撑材料/实验记录/q4-2b/metrics.json
365	支撑材料/实验记录/q4-2b/receipt.json
366	支撑材料/实验记录/q4-2c/metrics.json
367	支撑材料/实验记录/q4-2c/receipt.json
368	支撑材料/实验记录/q4-3b/metrics.json
369	支撑材料/实验记录/q4-3b/receipt.json
370	支撑材料/实验记录/q4-evolve-2/metrics.json
371	支撑材料/实验记录/q4-evolve-2b/metrics.json
372	支撑材料/实验记录/q4-evolve-3/metrics.json
373	支撑材料/实验记录/q4-evolve-final/metrics.json
374	支撑材料/实验记录/q4-evolve-lichun/metrics.json
375	支撑材料/实验记录/q4-frozen-endstate/metrics.json
376	支撑材料/实验记录/q4-realtime-m2/metrics.json
377	支撑材料/实验记录/q4-realtime-m2p80/metrics.json
378	支撑材料/实验记录/q4-realtime-m3/metrics.json
379	支撑材料/实验记录/q4-realtime-m3p88/metrics.json
380	支撑材料/实验记录/q4-season-matrix/metrics.json
381	支撑材料/实验记录/q4-season-matrix/receipt.json
382	支撑材料/实验记录/q4-season-sweep/metrics.json
383	支撑材料/实验记录/q4-season-sweep/receipt.json
384	支撑材料/实验记录/q4-strict-2/metrics.json
385	支撑材料/实验记录/q4-verify-constraints/metrics.json
386	支撑材料/实验记录/q4-verify-constraints/receipt.json
387	支撑材料/实验记录/q4-verify-cost/metrics.json
388	支撑材料/实验记录/q4m2-grid-hold/metrics.json
389	支撑材料/实验记录/q4m2-grid-hold/receipt.json
390	支撑材料/实验记录/q4m2-grid-scan/metrics.json
391	支撑材料/实验记录/q4m2-grid-scan/receipt.json
392	支撑材料/实验记录/q4m2-grid-tune/metrics.json
393	支撑材料/实验记录/q4m2-grid-tune/receipt.json
394	支撑材料/实验记录/q4m3-grid-hold/metrics.json
395	支撑材料/实验记录/q4m3-grid-hold/receipt.json
396	支撑材料/实验记录/q4m3-grid-scan/metrics.json
397	支撑材料/实验记录/q4m3-grid-scan/receipt.json
398	支撑材料/实验记录/q4m3-grid-tune/metrics.json
399	支撑材料/实验记录/q4m3-grid-tune/receipt.json
400	支撑材料/实验记录/v2-q2/metrics.json

#	文件
401	支撑材料/实验记录/v2-q2/receipt.json
402	支撑材料/实验记录/v2-q3/metrics.json
403	支撑材料/实验记录/v2-q3/receipt.json
404	支撑材料/实验记录/v2-q4-2/metrics.json
405	支撑材料/实验记录/v2-q4-2/receipt.json
406	支撑材料/实验记录/v2-q4-3/metrics.json
407	支撑材料/实验记录/v2-q4-3/receipt.json
408	支撑材料/数据/原始附件/附件 1.xlsx
409	支撑材料/数据/原始附件/附件 2.xlsx
410	支撑材料/数据/原始附件/附件 3.xlsx
411	支撑材料/数据/原始附件/附件 4.xlsx
412	支撑材料/数据/原始附件/附件 5/result1.xlsx
413	支撑材料/数据/原始附件/附件 5/result2.xlsx
414	支撑材料/数据/原始附件/附件 5/result3.xlsx
415	支撑材料/数据/原始附件/附件 5/result4-2.xlsx
416	支撑材料/数据/原始附件/附件 5/result4-3.xlsx
417	支撑材料/数据/清洗数据/q1_input.json
418	支撑材料/数据/清洗数据/q2_input.json
419	支撑材料/数据/清洗数据/q3_input.json
420	支撑材料/数据/清洗数据/q3_tables.json
421	支撑材料/数据/清洗数据/q4_input.json
422	支撑材料/渲染检查/渲染检查说明.md
423	支撑材料/诊断报告/D_节气四季诊断.txt
424	支撑材料/诊断报告/E1-E4_诊断报告.txt
425	支撑材料/诊断报告/E5-E6_诊断报告.txt
426	支撑材料/诊断报告/P2_光伏倾斜面天文模型诊断.txt
427	支撑材料/诊断报告/P_光伏季节归属诊断.txt
428	支撑材料/诊断报告/Q_光伏归属判决.txt
429	支撑材料/诊断报告/R_全知视角因子分解.txt
430	支撑材料/诊断报告/S_仅用 1 月可得性诊断.txt
431	支撑材料/诊断报告/调研记录.md
432	支撑材料/过程记录/AI_Usage_Declaration.md
433	支撑材料/过程记录/Figure_Requirements.md
434	支撑材料/过程记录/Q4-2_诊断与模型修正.md
435	支撑材料/过程记录/Q4_总报告_4-2 与 4-3.md
436	支撑材料/过程记录/References.md
437	支撑材料/过程记录/TSEF_可迁移自进化框架.md
438	支撑材料/过程记录/tablesq1.md
439	支撑材料/过程记录/tablesq234.md
440	支撑材料/过程记录/复杂插图交接文档.md
441	支撑材料/过程记录/插图集成方案.md
442	支撑材料/过程记录/插图需求_复杂图.md
443	支撑材料/过程记录/表格_问题一.md
444	支撑材料/过程记录/表格_问题二至四.md
445	支撑材料/过程记录/论文_2.1_数据分析_草稿.md
446	支撑材料/过程记录/论文_2.1_数据总览_草稿.md
447	支撑材料/过程记录/论文_AI使用声明_草稿.md
448	支撑材料/过程记录/论文_摘要_草稿.md
449	支撑材料/过程记录/论文大纲_v9.md
450	支撑材料/过程记录/论文源稿.md
451	支撑材料/过程记录/论文源稿_v9.md
452	最终核查报告.md
453	结果/result1.xlsx
454	结果/result2.xlsx
455	结果/result3.xlsx

#	文件
456	结果/result4-2.xlsx
457	结果/result4-3.xlsx

D.2 源程序代码

D.2.1 支撑材料/代码/01_问题一/q12_reverif.py

import os

-*- coding: utf-8 -*-

```

"""复核问题一/问题二数值(独立于既有产物)."""
import numpy as np, json
from scipy.optimize import linprog
from pathlib import Path
proj = Path(os.environ.get("CUMCM_ROOT", "."); root=proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; T=144
def load_att1():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 1.xlsx",read_only=True,data_only=True)
    rows=list(wb["Sheet1"].iter_rows(values_only=True))[1:]
    price=np.array([float(r[1]) for r in rows]); load=np.array([float(r[2]) for r in rows]); pv=np.array([float(r[3]) for r in rows])
    return price,load,pv
def load_att2():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv
price1,load1,pv1=load_att1(); load2,pv2=load_att2()

def solve_day(pr, net, soc_start, soc_end_fixed=None):
    n=3*T+(T+1); A_a=0;A_c=T;A_d=2*T;A_s=3*T
    c=np.zeros(n); c[0:T]=pr*dt
    bnds=[(0,None)]*T+[(0,Pmax)]*T+[(0,Pmax)]*T
    if soc_end_fixed is not None:
        bnds+=[(soc_start,soc_start)]+[(soc_min,soc_max)]*(T-1)+[(soc_end_fixed,soc_end_fixed)]
    else:
        bnds+=[(soc_start,soc_start)]+[(soc_min,soc_max)]*T
    A_ub=np.zeros((T,n)); b_ub=-net
    for t in range(T):
        A_ub[t,A_a+t]=-1.0;A_ub[t,A_d+t]=-1.0;A_ub[t,A_c+t]=1.0
    A_eq=np.zeros((T,n)); b_eq=np.zeros(T)
    for t in range(T):
        A_eq[t,A_s+t]=1.0;A_eq[t,A_s+t]=-1.0;A_eq[t,A_c+t]=-eta*dt;A_eq[t,A_d+t]=dt/eta
    r=linprog(c,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bnds,method="highs")
    if not r.success: raise RuntimeError(r.message)
    x=r.x; return x[A_a:A_a+T],x[A_c:A_c+T],x[A_d:A_d+T],x[A_s:A_s+T+1]

```

--- 问题一: 单日, 附件 1, SOC 日平衡 ---

```

p,c,d,s=solve_day(price1, load1-pv1, soc0, soc_end_fixed=soc0)
q1_cost=float(np.dot(price1,p)*dt); q1_kwh=float(p.sum()*dt)
print(f"问题一 LP 购电费={q1_cost:.2f} 元, 购电量={q1_kwh:.2f} kwh, SOC 首末=({s[0]:.2f},{s[-1]:.2f})")

```

无储能基线

```
p_ns=np.maximum(0.0, load1-pv1)
q1_ns=float(np.dot(price1,p_ns)*dt)
print(f"[问题一] 无储能基线={q1_ns:.2f} 元, 储能降低={(1-q1_cost/q1_ns)*100:.1f}%")
```

--- 问题二: 逐日滚动, 附件 1 电价 + 附件 2 实际(完美信息) ---

```
soc=soc0; tot=0.0; tot_kwh=0.0
for dd in range(365):
    p,c,d,s=solve_day(price1, load2[dd]-pv2[dd], soc, None)
    if dd>=31:
        tot+=float(np.dot(price1,p)*dt); tot_kwh+=float(p.sum()*dt)
        soc=float(s[-1])
print(f"[问题二] 完美信息全年购电费(334 天)={tot:.2f} 元, 购电量={tot_kwh:.2f} kWh")
```

无储能基线

```
ns=float(sum(np.dot(price1, np.maximum(0.0, load2[dd]-pv2[dd]))*dt for dd in range(31,365)))
print(f"[问题二] 无储能基线={ns:.2f} 元, 储能降低={(1-tot/ns)*100:.1f}%")
json.dump(dict(q1_cost=q1_cost,q1_kwh=q1_kwh,q1_ns=q1_ns,q2_cost=tot,q2_kwh=tot_kwh,q2_ns=ns),
           open(root/"data_cleaned/q12_reverify.json","w"), ensure_ascii=False, indent=1)
print("saved q12_reverify.json")
```

D.2.2 支撑材料/代码/01_问题一/q1_arbitrage.py

-*- coding: utf-8 -*-

```
import os, json, argparse
import numpy as np

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True)
    ap.add_argument("--out", required=True)
    args = ap.parse_args()
    with open(args.input, encoding="utf-8") as f:
        inp = json.load(f)
    price = np.asarray(inp["price"], dtype=float)
    load = np.asarray(inp["load"], dtype=float)
    pv = np.asarray(inp["pv"], dtype=float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
    T = len(price); net = load - pv
    Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]
    thr = float(np.median(price))
    SOC = st["soc_initial"]
    p = np.zeros(T); c = np.zeros(T); d = np.zeros(T); s = np.zeros(T+1); s[0] = SOC
    for t in range(T):
        nt = net[t]
        if nt < 0:
            chg = min(-nt, Pmax, (soc_max - SOC)/(eta*dt))
            chg = max(0.0, chg); c[t] = chg; SOC += eta*chg*dt; p[t] = 0.0; d[t] = 0.0
        elif price[t] < thr and SOC < soc_max - 1e-6:
```

```

        chg = min(Pmax, (soc_max - SOC)/(eta*dt))
        chg = max(0.0, chg); c[t] = chg; SOC += eta*chg*dt; p[t] = nt + chg; d[t] = 0.0
    elif price[t] > thr and SOC > soc_min + 1e-6:
        dis = min(Pmax, (SOC - soc_min)*eta/dt, nt)
        dis = max(0.0, dis); d[t] = dis; SOC -= dis*dt/eta; p[t] = max(0.0, nt - dis); c[t] = 0.0
    else:
        p[t] = nt; c[t] = 0.0; d[t] = 0.0
    s[t+1] = SOC
metrics = {
    "q1_total_cost": float(np.dot(price, p) * dt),
    "q1_total_purchase": float(p.sum() * dt),
    "q1_total_charge": float(c.sum() * dt),
    "q1_total_discharge": float(d.sum() * dt),
    "q1_soc_start": float(s[0]),
    "q1_soc_end": float(s[-1]),
}
os.makedirs(args.out, exist_ok=True)
with open(os.path.join(args.out, "metrics.json"), "w", encoding="utf-8") as f:
    json.dump({"metrics": metrics}, f, ensure_ascii=False, indent=2)
sol = {"purchase": p.tolist(), "charge": c.tolist(), "discharge": d.tolist(), "soc": s.tolist(), "net": net.tolist()}
with open(os.path.join(args.out, "solution.json"), "w", encoding="utf-8") as f:
    json.dump(sol, f, ensure_ascii=False, indent=2)
print("q1_total_cost(arbitrage) =", metrics["q1_total_cost"])

if __name__ == "__main__":
    main()

```

D.2.3 支撑材料/代码/01_问题一/q1_dp.py

-*- coding: utf-8 -*-

```

import os, json, argparse
import numpy as np

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True)
    ap.add_argument("--out", required=True)
    args = ap.parse_args()
    with open(args.input, encoding="utf-8") as f:
        inp = json.load(f)
    price = np.asarray(inp["price"], dtype=float)
    load = np.asarray(inp["load"], dtype=float)
    pv = np.asarray(inp["pv"], dtype=float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
    T = len(price); net = load - pv
    Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]; soc0 = st["soc_initial"]

    ds = 0.5 # SOC grid step (kWh)
    levels = np.arange(soc_min, soc_max + 1e-9, ds)

```

```

N = len(levels)
i0 = int(round((soc0 - soc_min) / ds))
k_charge_max = int(eta * Pmax * dt / ds) # 750/0.5 = 1500
k_discharge_max = int(Pmax * dt / eta / ds) # 925.9/0.5 = 1851

F = np.full(N, np.inf); F[i0] = 0.0
for t in range(T):
    pr = price[t]; nt = net[t]
    Fn = np.full(N, np.inf)
    # charge: i -> j = i+k (SOC increases by k*ds)
    for k in range(1, k_charge_max + 1):
        c = (k * ds) / (eta * dt)
        cost = pr * max(0.0, nt + c) * dt
        np.minimum(Fn[k:], F[:-k] + cost, out=Fn[k:])
    # discharge: i -> j = i-k (SOC decreases by k*ds)
    for k in range(1, k_discharge_max + 1):
        d = (k * ds) * eta / dt
        cost = pr * max(0.0, nt - d) * dt
        np.minimum(Fn[:-k], F[k:] + cost, out=Fn[:-k])
    # idle
    cost = pr * max(0.0, nt) * dt
    np.minimum(Fn, F + cost, out=Fn)
    F = Fn
total_cost = float(F[i0])

metrics = {
    "q1_total_cost": total_cost,
    "q1_soc_start": float(soc0),
    "q1_soc_end": float(soc0),
}
os.makedirs(args.out, exist_ok=True)
with open(os.path.join(args.out, "metrics.json"), "w", encoding="utf-8") as f:
    json.dump({"metrics": metrics}, f, ensure_ascii=False, indent=2)
print("q1_total_cost(DP) =", total_cost)

if __name__ == "__main__":
    main()

```

D.2.4 支撑材料/代码/01_问题一/q1_lp.py

-*- coding: utf-8 -*-

```

import sys, os, json, argparse
import numpy as np
from scipy.optimize import linprog

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True)
    ap.add_argument("--out", required=True)

```

```

args = ap.parse_args()
with open(args.input, encoding="utf-8") as f:
    inp = json.load(f)
price = np.asarray(inp["price"], dtype=float)
load = np.asarray(inp["load"], dtype=float)
pv = np.asarray(inp["pv"], dtype=float)
st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
T = len(price); net = load - pv
Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]; soc0 = st["soc_initial"]

n = 3*T + (T+1)
c_obj = np.zeros(n); c_obj[0:T] = price * dt
bounds = [(0, None)]*T + [(0, Pmax)]*T + [(0, Pmax)]*T
bounds += [(soc0, soc0)]
for _ in range(1, T): bounds.append((soc_min, soc_max))
bounds.append((soc0, soc0))

A_ub = np.zeros((T, n)); b_ub = -net
for t in range(T):
    A_ub[t, t] = -1.0; A_ub[t, T+t] = 1.0; A_ub[t, 2*T+t] = -1.0
A_eq = np.zeros((T, n)); b_eq = np.zeros(T)
for t in range(T):
    A_eq[t, 3*T+t+1] = 1.0; A_eq[t, 3*T+t] = -1.0
    A_eq[t, T+t] = -eta*dt; A_eq[t, 2*T+t] = dt/eta

res = linprog(c_obj, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs")
if not res.success:
    raise RuntimeError("LP failed: " + res.message)
x = res.x
p = x[0:T]; c = x[T:2*T]; d = x[2*T:3*T]; s = x[3*T:3*T+T+1]

metrics = {
    "q1_total_cost": float(np.dot(price, p) * dt),
    "q1_total_purchase": float(p.sum() * dt),
    "q1_total_charge": float(c.sum() * dt),
    "q1_total_discharge": float(d.sum() * dt),
    "q1_soc_start": float(s[0]),
    "q1_soc_end": float(s[-1]),
}
os.makedirs(args.out, exist_ok=True)
with open(os.path.join(args.out, "metrics.json"), "w", encoding="utf-8") as f:
    json.dump({"metrics": metrics}, f, ensure_ascii=False, indent=2)
sol = {"purchase": p.tolist(), "charge": c.tolist(), "discharge": d.tolist(),
        "soc": s.tolist(), "net": net.tolist()}
with open(os.path.join(args.out, "solution.json"), "w", encoding="utf-8") as f:

```

```

        json.dump(sol, f, ensure_ascii=False, indent=2)
    print("q1_total_cost =", metrics["q1_total_cost"])

if __name__ == "__main__":
    main()

```

D.2.5 支撑材料/代码/01_问题一/q1_no_storage.py

-*- coding: utf-8 -*-

```

import os, json, argparse
import numpy as np

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True)
    ap.add_argument("--out", required=True)
    args = ap.parse_args()
    with open(args.input, encoding="utf-8") as f:
        inp = json.load(f)
    price = np.asarray(inp["price"], dtype=float)
    load = np.asarray(inp["load"], dtype=float)
    pv = np.asarray(inp["pv"], dtype=float)
    st = inp["storage"]; dt = inp["dt"]
    T = len(price); net = load - pv
    p = np.maximum(0.0, net)
    c = np.zeros(T); d = np.zeros(T); s = np.full(T+1, st["soc_initial"])
    metrics = {
        "q1_total_cost": float(np.dot(price, p) * dt),
        "q1_total_purchase": float(p.sum() * dt),
        "q1_total_charge": float(c.sum() * dt),
        "q1_total_discharge": float(d.sum() * dt),
        "q1_soc_start": float(s[0]),
        "q1_soc_end": float(s[-1]),
    }
    os.makedirs(args.out, exist_ok=True)
    with open(os.path.join(args.out, "metrics.json"), "w", encoding="utf-8") as f:
        json.dump({"metrics": metrics}, f, ensure_ascii=False, indent=2)
    sol = {"purchase": p.tolist(), "charge": c.tolist(), "discharge": d.tolist(), "soc": s.tolist(), "net": net.tolist()}
    with open(os.path.join(args.out, "solution.json"), "w", encoding="utf-8") as f:
        json.dump(sol, f, ensure_ascii=False, indent=2)
    print("q1_total_cost(no-storage) =", metrics["q1_total_cost"])

if __name__ == "__main__":
    main()

```

D.2.6 支撑材料/代码/02_问题二/q2_accuracy.py

-*- coding: utf-8 -*-

"""问题2 预测层精度：统计期内各预测方法的 MAE / RMSE，以及最终标定参数下的精度。

预测一律只用第 d-1 天及更早的实际净负荷（与主路线同一信息边界）。

```

"""
import os, sys, json, argparse

```

```

import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast

def stats(N, P, lo, hi):
    e = N[lo:hi] - P[lo:hi]
    return float(np.abs(e).mean()), float(np.sqrt((e ** 2).mean()))

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--input', required=True); ap.add_argument('--out', required=True)
    ap.add_argument('--first-day', type=int, default=31); ap.add_argument('--last-day', type=int, default=365)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding='utf-8'))
    net = np.asarray(inp['load_actual'], float) - np.asarray(inp['pv_actual'], float)
    D, T = net.shape
    m = {}
    # 最终标定参数
    Nhat, _, _ = build_forecast(net, 0.15, 0.90)
    mae, rmse = stats(net, Nhat, a.first_day, a.last_day)
    m['q2_mae_final'], m['q2_rmse_final'] = mae, rmse
    # 对照: 持久法 / 7 天均值 / 28 天均值
    per = np.zeros_like(net); per[0] = net[0]; per[1:] = net[:-1]
    m['q2_mae_persistence'], m['q2_rmse_persistence'] = stats(net, per, a.first_day, a.last_day)
    for w, key in ((7, 'ma7'), (28, 'ma28')):
        P = np.zeros_like(net); P[0] = net[0]
        for d in range(1, D): P[d] = net[max(0, d - w):d].mean(axis=0)
        m['q2_mae_' + key], m['q2_rmse_' + key] = stats(net, P, a.first_day, a.last_day)
    # 其它 (alpha, beta) 组合, 用于说明'点预测更准不等于费用更低'
    for al, be, tag in ((0.25, 0.80, 'a025b080'), (0.40, 0.90, 'a040b090'), (0.15, 0.90, 'a015b090')):
        P, _, _ = build_forecast(net, al, be)
        mm, rr = stats(net, P, a.first_day, a.last_day)
        m['q2_mae_' + tag], m['q2_rmse_' + tag] = mm, rr
    os.makedirs(a.out, exist_ok=True)
    json.dump({'metrics': m}, open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'), ensure_ascii=False)
    for k in sorted(m): print("%-28s %.4f" % (k, m[k]))

if __name__ == '__main__':
    main()

```

D.2.7 支撑材料/代码/02_问题二/q2_bounds.py

-*- coding: utf-8 -*-

"""问题 2 的边界模型: 完美信息理论下界 与 "储能全年视野" 上界。

```

--mode perfect : 用全年实际净负荷一次性求解(紧急购电费应为 0), 作为理论上界/下界参考
--mode yearjoint: 计划购电量固定为主路线的逐日结果, 让储能在全年 52560 个时段上联合优化,
                  衡量"把电池的优化范围扩大到全年"能省多少(该路径的电池具有全年视野)
--mode nostorage: 无储能基线
"""
import os, sys, json, argparse
import numpy as np
from scipy.optimize import linprog
from scipy.sparse import coo_matrix
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal

```

```

from q2_online import day_1p

def load(a):
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float)
    load_ = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; st = dict(st); st["dt"] = float(inp["dt"])
    return price, load_, pv, load_ - pv, st

def build_year(nd, T, eta, dt, Pmax, smin, smax):
    nT = nd * T
    o_g, o_c, o_d, o_e, o_S = 0, nT, 2 * nT, 3 * nT, 4 * nT
    nv = 4 * nT + (nT + 1)
    rows, cols, vals = [], [], []
    r = 0
    rows.append(r); cols.append(o_S + 0); vals.append(1.0); r += 1
    for i in range(nT):
        rows += [r, r, r, r]; cols += [o_S + i + 1, o_S + i, o_c + i, o_d + i]
        vals += [1.0, -1.0, -eta * dt, dt / eta]; r += 1
    Aeq = coo_matrix((vals, (rows, cols)), shape=(r, nv)).tocsr()
    ru, cu, vu = [], [], []
    for i in range(nT):
        ru += [i, i, i, i]; cu += [o_e + i, o_d + i, o_c + i, o_g + i]
        vu += [-1.0, -1.0, 1.0, -1.0]
    Aub = coo_matrix((vu, (ru, cu)), shape=(nT, nv)).tocsr()
    lb = np.zeros(nv); ub = np.zeros(nv)
    ub[o_g:o_g + nT] = np.inf; ub[o_c:o_c + nT] = Pmax; ub[o_d:o_d + nT] = Pmax
    ub[o_e:o_e + nT] = np.inf
    lb[o_S:o_S + nT + 1] = smin; ub[o_S:o_S + nT + 1] = smax
    return dict(nv=nv, nT=nT, o_g=o_g, o_c=o_c, o_d=o_d, o_e=o_e, o_S=o_S, Aeq=Aeq, Aub=Aub, lb=lb, ub=ub, n_eq=r)

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--mode", required=True, choices=["perfect", "yearjoint", "nostorage"])
    ap.add_argument("--alpha", type=float, default=0.15); ap.add_argument("--beta", type=float, default=0.9)
    ap.add_argument("--W", type=int, default=7); ap.add_argument("--p", type=float, default=0.88)
    ap.add_argument("--first-day", type=int, default=31); ap.add_argument("--last-day", type=int, default=365)
    a = ap.parse_args()
    price, load_, pv, net, st = load(a)
    dt = float(st["dt"]); eta = float(st["efficiency"]); Pmax = float(st["power_max"])
    smin = float(st["soc_min"]); smax = float(st["soc_max"]); s0 = float(st["soc_initial"])
    T = len(price); ND = a.last_day - 1
    sl = slice(a.first_day, a.last_day)
    if a.mode == "nostorage":
        # 无储能基线: 计划层同样只用历史预测 + 报童冗余, 但储能不参与
        Nhat, err, _, _ = build_forecast(net, a.alpha, a.beta)
        o = a.first_day - 1
        g = np.zeros((ND, T)); e = np.zeros((ND, T))

```

```

for d in range(1, a.last_day):
    g[d - 1] = np.maximum(0.0, Nhat[d] + margin(err, d, a.W, a.p))
    e[d - 1] = np.maximum(0.0, net[d] - g[d - 1])
pc = float(sum(float(price @ (g[d] * dt)) for d in range(o, ND)))
ec = float(sum(float(5.0 * price @ (e[d] * dt)) for d in range(o, ND)))
metrics = {"q2_planned_cost": pc, "q2_planned_kwh": float(g[o:].sum() * dt),
           "q2_emergency_cost": ec, "q2_emergency_kwh": float(e[o:].sum() * dt),
           "q2_total_cost": pc + ec, "q2_charge_kwh": 0.0, "q2_discharge_kwh": 0.0,
           "q2_loss_kwh": 0.0, "q2_days": float(ND - o),
           "q2_soc_min": s0, "q2_soc_max": s0}
else:
    S = build_year(ND, T, eta, dt, Pmax, smin, smax)
    nT = S["nT"]
    if a.mode == "perfect":
        gfix = None
    else:
        Nhat, err, _, _ = build_forecast(net, a.alpha, a.beta)
        s = s0; gl = np.zeros((ND, T))
        for d in range(1, a.last_day):
            g, _, _ = day_lp(price, Nhat[d] + margin(err, d, a.W, a.p), s, eta, dt, Pmax, smin, smax)
            gl[d - 1] = g
            _, _, Ss = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
            s = float(Ss[-1])
            gfix = gl.ravel()
        c = np.zeros(S["nv"])
        if gfix is None:
            c[S["o_g"]:S["o_g"] + nT] = np.tile(price * dt, ND)
            c[S["o_e"]:S["o_e"] + nT] = 5.0 * np.tile(price * dt, ND)
        else:
            c[S["o_e"]:S["o_e"] + nT] = 5.0 * np.tile(price * dt, ND)
        bub = -net[1:a.last_day].ravel()
        if gfix is not None:
            bub = gfix - net[1:a.last_day].ravel()
            S["lb"][S["o_g"]:S["o_g"] + nT] = gfix
            S["ub"][S["o_g"]:S["o_g"] + nT] = gfix # 购电量固定为主路线逐日计划, 电池全年联合优化
        beq = np.zeros(S["n_eq"]); beq[0] = s0
        res = linprog(c, A_ub=S["Aub"], b_ub=bub, A_eq=S["Aeq"], b_eq=beq,
                     bounds=list(zip(S["lb"], S["ub"])), method="highs")
        if not res.success:
            raise RuntimeError(res.message)
        gg = res.x[S["o_g"]:S["o_g"] + nT].reshape(ND, T)
        cc = res.x[S["o_c"]:S["o_c"] + nT].reshape(ND, T)
        dd = res.x[S["o_d"]:S["o_d"] + nT].reshape(ND, T)
        ee = res.x[S["o_e"]:S["o_e"] + nT].reshape(ND, T)
        So = res.x[S["o_S"]:S["o_S"] + nT + 1]
        o = a.first_day - 1
        pc = float(sum(float(price @ (gg[d] * dt)) for d in range(o, ND)))
        pk = float(gg[o:].sum() * dt)

```

```

ec = float(sum(float(5.0 * price @ (ee[d] * dt)) for d in range(o, ND)))
ek = float(ee[o:].sum() * dt)
ch = float(cc[o:].sum() * dt); di = float(dd[o:].sum() * dt)
metrics = {"q2_planned_cost": pc, "q2_planned_kwh": pk,
           "q2_emergency_cost": ec, "q2_emergency_kwh": ek, "q2_total_cost": pc + ec,
           "q2_charge_kwh": ch, "q2_discharge_kwh": di,
           "q2_loss_kwh": (1 - eta) * ch + ((1 - eta) / eta) * di,
           "q2_days": float(ND - o),
           "q2_soc_min": float(So.min()), "q2_soc_max": float(So.max())}
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"), ensure_ascii=False)
print("mode=%s    total=%0.2f    planned=%0.2f    emergency=%0.2f" % (a.mode, metrics["q2_total_cost"],
metrics["q2_planned_cost"], metrics["q2_emergency_cost"]))

if __name__ == "__main__":
    main()

```

D.2.8 支撑材料/代码/02_问题二/q2_caliber_compare.py

-*- coding: utf-8 -*-

"""四档数据依赖口径的统一对照(同一实现, 保证表内数字同源)。

```

A 严格无泄露: 只用 1 月(1/2-1/31)标定 -> 用该参数跑全年
B 零数据理论值: 不用任何历史数据, 取题面 5 倍电价推得的 80 分位
C 四季独立(基于真实数据的调整): 每块用上一块数据定参
D 全期一套(基于真实数据的调整): 用全期网格最优
"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_season_blocks import Sim, GRID, BLOCKS, D0
import datetime

def pack(sim, r, blocks=None):
    d = {"total": r["total"], "plan_cost": r["plan_cost"], "em_cost": r["em_cost"],
         "plan_kwh": r["plan_kwh"], "em_kwh": r["em_kwh"], "curtail": r["curtail"],
         "soc_min": r["soc_min"], "soc_max": r["soc_max"]}
    return d

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    sim = Sim(inp)
    m = {}; det = {}

    # A 严格无泄露: 1 月标定
    jan = [(W, p, sim.block(1, 31, W, p, False, None, acc_from=1)["total"]) for W, p in GRID]
    Wa, pa, ca = min(jan, key=lambda x: x[2])
    ra = sim.block(1, 365, Wa, pa, False, None, acc_from=31)
    m["q2cc_A_W"] = float(Wa); m["q2cc_A_p"] = pa; m["q2cc_A_jan"] = ca
    for k, v in pack(sim, ra).items(): m["q2cc_A_" + k] = v
    det["A"] = {"table": {"all": [Wa, pa]}, **pack(sim, ra)}

```

```

print("A 严格无泄露 1月标定 (W=%d,p=%0.0f%%) 全年 %12.2f= 计划 %12.2f+ 紧急 %10.2f (%.2f kWh) 弃电
%.1f" % (
    Wa, pa * 100, ra["total"], ra["plan_cost"], ra["em_cost"], ra["em_kwh"], ra["curtail"]))

# B 零数据: (7,80%)
rb = sim.block(1, 365, 7, 0.80, False, None, acc_from=31)
for k, v in pack(sim, rb).items(): m["q2cc_B_" + k] = v
det["B"] = {"table": {"all": [7, 0.80]}, **pack(sim, rb)}
print("B 零数据理论值 (W=7,p=80%%) 全年 %12.2f= 计划 %12.2f+ 紧急 %10.2f (%.2f kWh) 弃电 %.1f" % (
    rb["total"], rb["plan_cost"], rb["em_cost"], rb["em_kwh"], rb["curtail"]))

# C 四季独立(上一块标定)
TB = {"winter_a": (7, 0.80), "spring": (7, 0.80), "summer": (7, 0.88),
      "autumn": (7, 0.88), "winter_b": (5, 0.95)}
s_cur = sim.warm_soc(); blocks = {}
tot = {"plan_cost": 0.0, "em_cost": 0.0, "plan_kwh": 0.0, "em_kwh": 0.0, "curtail": 0.0,
      "soc_min": 1e18, "soc_max": -1e18}
for key, f, l in BLOCKS:
    W, p = TB[key]
    r = sim.block(f, l, W, p, False, s_cur)
    r88 = sim.block(f, l, 7, 0.88, False, s_cur)
    blocks[key] = {"W": W, "p": p, "total": r["total"], "vs88": r["total"] - r88["total"]}
    for k in ("plan_cost", "em_cost", "plan_kwh", "em_kwh", "curtail"): tot[k] += r[k]
    tot["soc_min"] = min(tot["soc_min"], r["soc_min"]); tot["soc_max"] = max(tot["soc_max"], r["soc_max"])
    s_cur = r["soc_end"]
tot["total"] = tot["plan_cost"] + tot["em_cost"]
for k, v in tot.items(): m["q2cc_C_" + k] = v
det["C"] = {"table": {k: list(v) for k, v in TB.items()}, "blocks": blocks, **tot}
print("C 四季独立(上一季标定) 全年 %12.2f= 计划 %12.2f+ 紧急 %10.2f (%.2f kWh) 弃电 %.1f" % (
    tot["total"], tot["plan_cost"], tot["em_cost"], tot["em_kwh"], tot["curtail"]))
for k, v in blocks.items():
    print(" %-9s (W=%d,p=%3.0f) 块费用 %11.2f 与同块(7,88%%)差 %10.2f" % (k, v["W"], v["p"] * 100, v["total"],
v["vs88"]))

# D 全期一套 (7,88%)
rd = sim.block(1, 365, 7, 0.88, False, None, acc_from=31)
for k, v in pack(sim, rd).items(): m["q2cc_D_" + k] = v
det["D"] = {"table": {"all": [7, 0.88]}, **pack(sim, rd)}
print("D 全期一套 (W=7,p=88%%) 全年 %12.2f= 计划 %12.2f+ 紧急 %10.2f (%.2f kWh) 弃电 %.1f" % (
    rd["total"], rd["plan_cost"], rd["em_cost"], rd["em_kwh"], rd["curtail"]))
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": m, "detail": det}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
    ensure_ascii=False, indent=1)
print("done")

if __name__ == "__main__":

```

main()

D.2.9 支撑材料/代码/02_问题二/q2_dp.py

-*- coding: utf-8 -*-

"""问题2 独立复算路径 (q2-dp): 日内计划改用 SOC 离散化动态规划(min-plus 逆向递推), 不调用任何线性规划求解器, 并回溯最优动作轨迹以还原当天的计划购电量。

与主路线 q2_online.py 的差别只在“日内计划怎么解”:

主路线 = HiGHS 线性规划; 本脚本 = 手写动态规划 (状态 = 储电量)

预测层与实时执行层属于模型定义, 两条路径共用 q2_forecast.py(这一层不是算法独立性的对象)。

"""

```
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal
```

```
def solve_day_dp(price, Nplan, soc0, eta, dt, Pmax, smin, smax, ds=2.0):
    """返回 (最优目标值, 计划购电功率 g, 充电 c, 放电 d, SOC 轨迹)。”"""
    T = len(price)
    levels = np.arange(smin, smax + 1e-9, ds); N = len(levels)
    i0 = int(round((soc0 - smin) / ds)); i0 = min(max(i0, 0), N - 1)
    kc = max(int(eta * Pmax * dt / ds), 1)
    kd = max(int(Pmax * dt / (eta * ds)), 1)
    F = np.full(N, np.inf); F[i0] = 0.0
    Fhist = np.empty((T + 1, N)); Fhist[0] = F
    Srch = np.zeros((T, N), dtype=np.int32) # 纪录每个末态的最优动作(带符号的 SOC 步数)
    for t in range(T):
        pr = price[t]; nt = Nplan[t]
        best = F + pr * max(0.0, nt) * dt
        src = np.zeros(N, dtype=np.int32)
        for k in range(1, kc + 1):
            cc = (k * ds) / (eta * dt)
            arr = np.full(N, np.inf)
            arr[k:] = F[:-k] + pr * max(0.0, nt + cc) * dt
            m = arr < best
            best = np.where(m, arr, best); src = np.where(m, np.int32(k), src)
        for k in range(1, kd + 1):
            dd = (k * ds) * eta / dt
            arr = np.full(N, np.inf)
            arr[:-k] = F[k:] + pr * max(0.0, nt - dd) * dt
            m = arr < best
            best = np.where(m, arr, best); src = np.where(m, np.int32(-k), src)
        F = best; Fhist[t + 1] = F; Srch[t] = src
    j = int(np.argmin(F))
    g = np.zeros(T); c = np.zeros(T); d = np.zeros(T); S = np.zeros(T + 1)
    jj = j; S[T] = levels[jj]
    for t in range(T - 1, -1, -1):
        k = int(Srch[t][jj])
        if k > 0:
            c[t] = (k * ds) / (eta * dt)
        elif k < 0:
            d[t] = (-k) * ds * eta / dt
```

```

    g[t] = max(0.0, Nplan[t] + c[t] - d[t])
    jj = jj - k
    S[t] = levels[jj]
    return float(F[jj]), g, c, d, S

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--alpha", type=float, default=0.15); ap.add_argument("--beta", type=float, default=0.9)
    ap.add_argument("--W", type=int, default=7); ap.add_argument("--p", type=float, default=0.88)
    ap.add_argument("--ds", type=float, default=2.0)
    ap.add_argument("--first-day", type=int, default=31); ap.add_argument("--last-day", type=int, default=365)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float)
    net = np.asarray(inp["load_actual"], float) - np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = float(inp["dt"]); eta = float(st["efficiency"])
    Pmax = float(st["power_max"]); smin = float(st["soc_min"]); smax = float(st["soc_max"])
    Nhat, err, _, _ = build_forecast(net, a.alpha, a.beta)
    s = float(st["soc_initial"])
    plan_cost = plan_kwh = em_cost = em_kwh = ch = di = 0.0
    days = 0; smin_seen = 1e18; smax_seen = -1e18; gaps = []
    for d in range(1, a.last_day):
        Nplan = Nhat[d] + margin(err, d, a.W, a.p)
        obj, g, _, _ = solve_day_dp(price, Nplan, s, eta, dt, Pmax, smin, smax, a.ds)
        e, cc, dd, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
        if d >= a.first_day:
            plan_cost += float(price @ (g * dt)); plan_kwh += float(g.sum() * dt)
            em_cost += float(5.0 * price @ (e * dt)); em_kwh += float(e.sum() * dt)
            ch += float(cc.sum() * dt); di += float(dd.sum() * dt)
            gaps.append(float(obj - price @ (g * dt)))
            days += 1
        smin_seen = min(smin_seen, float(S.min())); smax_seen = max(smax_seen, float(S.max()))
        s = float(S[-1])
    metrics = {"q2_planned_cost": plan_cost, "q2_planned_kwh": plan_kwh,
              "q2_emergency_cost": em_cost, "q2_emergency_kwh": em_kwh,
              "q2_total_cost": plan_cost + em_cost,
              "q2_charge_kwh": ch, "q2_discharge_kwh": di,
              "q2_loss_kwh": (1 - eta) * ch + ((1 - eta) / eta) * di,
              "q2_days": float(days), "q2_soc_min": smin_seen, "q2_soc_max": smax_seen,
              "q2_dp_grid_ds": a.ds,
              "q2_dp_objective_minus_purchase_kwh_cost": float(np.mean(gaps)) if gaps else 0.0}
    os.makedirs(a.out, exist_ok=True)
    json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"), ensure_ascii=False)
    print("q2_dp planned_cost=%0.2f total=%0.2f soc=[%0.1f,%0.1f]" % (plan_cost, plan_cost + em_cost, smin_seen, smax_seen))

if __name__ == "__main__":
    main()

```

D.2.10 支撑材料/代码/02_问题二/q2_final.py

-*- coding: utf-8 -*-

```
import os, json, argparse
import numpy as np
from scipy.optimize import linprog

def forecast_ma_all(arr, window=7):
    D = arr.shape[0]; F = np.zeros_like(arr); F[0] = arr[0]
    for d in range(1, D):
        lo = max(0, d-window); F[d] = arr[lo:d].mean(axis=0)
    return F

def solve_day_lp(price, net_margin, soc0, eta, dt, Pmax, soc_min, soc_max):
    T = len(price); n = 3*T + (T+1)
    c = np.zeros(n); c[0:T] = price*dt
    bounds = [(0, None)]*T + [(0, Pmax)]*T + [(0, Pmax)]*T + [(soc0, soc0)]
    for _ in range(1, T): bounds.append((soc_min, soc_max))
    bounds.append((soc_min, soc_max))
    A_ub = np.zeros((T, n)); b_ub = -net_margin
    for t in range(T): A_ub[t, t] = -1.0; A_ub[t, T+t] = 1.0; A_ub[t, 2*T+t] = -1.0
    A_eq = np.zeros((T, n)); b_eq = np.zeros(T)
    for t in range(T):
        A_eq[t, 3*T+t+1] = 1.0; A_eq[t, 3*T+t] = -1.0; A_eq[t, T+t] = -eta*dt; A_eq[t, 2*T+t] = dt/eta
    res = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs")
    if not res.success: raise RuntimeError(res.message)
    x = res.x
    return x[0:T], x[T:2*T], x[2*T:3*T], x[3*T:3*T+T+1], float(np.dot(price, x[0:T])*dt)

def main():
    ap = argparse.ArgumentParser(); ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float); load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
    Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]
    D, T = load.shape; net = load - pv
    window_f = 7; window_err = 28; pct = 84.0
    lf_all = forecast_ma_all(load, window_f); pf_all = forecast_ma_all(pv, window_f); nf_all = lf_all - pf_all
    soc = st["soc_initial"]; tp=0.0; te=0.0; plans=[]
    for d in range(D):
        if d >= 2:
            lo = max(1, d-window_err)
            errs = net[lo:d] - nf_all[lo:d]
            margin = np.maximum(np.percentile(errs, pct, axis=0), 0.0)
        else:
            margin = np.zeros(T)
        p, c, dd, s, pcost = solve_day_lp(price, nf_all[d]+margin, soc, eta, dt, Pmax, soc_min, soc_max)
        E = np.maximum(0.0, net[d] + c - p - dd)
        if d >= 31:
```

```

        tp += pcost; te += 5.0*np.dot(price, E)*dt
        plans.append(dict(day=d, purchase=p.tolist(), charge=c.tolist(), discharge=dd.tolist(), soc=s.tolist(),
emergency=E.tolist(), margin=margin.tolist()))
        soc = float(s[-1])
        metrics = {"q2_total_cost": tp+te, "q2_planned_cost": tp, "q2_emergency_cost": te}
        os.makedirs(a.out, exist_ok=True)
        json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"))
        json.dump({"plans": plans, "price": price.tolist()}, open(os.path.join(a.out, "solution.json"), "w", encoding="utf-8"))
        print(f'q2-final(7MA+84%): total={tp+te:.2f} planned={tp:.2f} emergency={te:.2f}')
if __name__ == "__main__": main()

```

D.2.11 支撑材料/代码/02_问题二/q2_firstmonth_calib.py

-*- coding: utf-8 -*-

"""只用"第一个月"(2025 年 1 月)标定参数 — 严格无泄露的基准方案。

依据: 附件 2 给出 2025.1.1–12.31 的逐日实际值, 而结果文件只需填 2.1–12.31;
因此 1 月是**唯一在统计期开始前就完全知晓**的月份, 用它标定不涉及任何未来信息。

(1 月 1 日为初始化日, 不产生计划, 故标定窗口为 1/2–1/31 共 30 天.)

流程: 在 1 月窗口上跑 $W \times p$ 网格 -> 选费用最小的一组 -> 用该组跑全年 2/1–12/31.

```

"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_season_blocks import Sim, GRID

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    sim = Sim(inp)
    m = {}; rows = []
    # 1) 在 1 月窗口(1/2-1/31, 30 天)上做网格
    for W, p in GRID:
        r = sim.block(1, 31, W, p, False, None, acc_from=1)
        rows.append({"W": W, "p": p, "jan": r["total"]})
        m["q2fm_jan_w%d_p%02d" % (W, round(p * 100))] = r["total"]
    best = min(rows, key=lambda x: x["jan"])
    Wb, pb = best["W"], best["p"]
    m["q2fm_best_W"] = float(Wb); m["q2fm_best_p"] = pb; m["q2fm_jan_cost"] = best["jan"]
    print("1 月窗口最优: W=%d, p=%0.0f%% (1 月费用 %.2f 元)" % (Wb, pb * 100, best["jan"]))
    print(" 1 月窗口前 5 名:", [(x["W"], int(x["p"] * 100), round(x["jan"], 1)) for x in sorted(rows, key=lambda y: y["jan"])[:5]])
    # 2) 用该组跑全年
    r = sim.block(1, 365, Wb, pb, False, None, acc_from=31)
    m.update({"q2fm_total": r["total"], "q2fm_plan": r["plan_cost"], "q2fm_em": r["em_cost"],
        "q2fm_em_kwh": r["em_kwh"], "q2fm_plan_kwh": r["plan_kwh"],
        "q2fm_curtail": r["curtail"], "q2fm_soc_min": r["soc_min"], "q2fm_soc_max": r["soc_max"]})
    print(" 用于全年: 总 %.2f = 计划 %.2f + 紧急 %.2f (%.2f kWh)" % (
        r["total"], r["plan_cost"], r["em_cost"], r["em_kwh"]))
    # 3) 对照: 理论 80% 先验

```

```

for tag, (W2, p2) in (("prior80", (7, 0.80)), ("full88", (7, 0.88))):
    rr = sim.block(1, 365, W2, p2, False, None, acc_from=31)
    m["q2fm_%s_total" % tag] = rr["total"]; m["q2fm_%s_em" % tag] = rr["em_cost"]
    print(" 对照 %s (W=%d,p=%f%): 总 %2f 紧急 %2f" % (tag, W2, p2 * 100, rr["total"], rr["em_cost"]))
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": m, "grid": rows}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
          ensure_ascii=False, indent=1)
print("done")

if __name__ == "__main__":
    main()

```

D.2.12 支撑材料/代码/02_问题二/q2_firstmonth_full.py

-*- coding: utf-8 -*-

"""严格无泄露档(加强版): 把 alpha, beta, W, p **全部**只用第一个月(1 月)标定.

背景: 原第a档只把 (W,p) 用 1 月标定, alpha/beta 仍是全期扫描出来的(评审 V2-4 指出).
本运行把 4 个参数全部在 1 月窗口(1/2-1/31, 30 天)上选, 再用它们跑全年 2/1-12/31.

```

"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal
from q2_season_blocks import day_lp, GRID

ALPHAS = (0.10, 0.15, 0.25, 0.40)
BETAS = (0.6, 0.7, 0.8, 0.9)

def build(net, alpha, beta):
    return build_forecast(net, alpha, beta)

def run(inp, alpha, beta, W, p, first, last, acc_from, s_start=None):
    price = np.asarray(inp["price"], float)
    net = np.asarray(inp["load_actual"], float) - np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = float(inp["dt"]); eta = float(st["efficiency"])
    Pmax = float(st["power_max"]); smin = float(st["soc_min"]); smax = float(st["soc_max"])
    Nhat, err, _, _ = build(net, alpha, beta)
    s = float(st["soc_initial"]) if s_start is None else s_start
    pc = pk = ec = ek = cu = 0.0
    for d in range(first, last):
        Nplan = Nhat[d] + margin(err, d, W, p)
        g, _, _ = day_lp(price, Nplan, s, eta, dt, Pmax, smin, smax, None)
        e, cc, dd, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
        cur = np.maximum(0.0, (g - net[d]) - cc)
        if d >= acc_from:
            pc += float(price @ (g * dt)); pk += float(g.sum() * dt)
            ec += 5.0 * float(price @ (e * dt)); ek += float(e.sum() * dt); cu += float(cur.sum() * dt)
        s = float(S[-1])
    return {"total": pc + ec, "plan_cost": pc, "em_cost": ec, "plan_kwh": pk, "em_kwh": ek,
            "curtail": cu, "soc_end": s}

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)

```

```

a = ap.parse_args()
inp = json.load(open(a.input, encoding="utf-8"))
m = {}
# 1) alpha/beta 在 1 月窗口上选(W=7, p=80% 作为该步的中性取值, 之后再联合选 W,p)
ab = []
for al in ALPHAS:
    for be in BETAS:
        r = run(inp, al, be, 7, 0.80, 1, 31, 1)
        ab.append({"alpha": al, "beta": be, "jan": r["total"]})
        m["q2ff_ab_a%03d_b%d" % (round(al * 100), round(be * 10))] = r["total"]
    best_ab = min(ab, key=lambda x: x["jan"])
    m["q2ff_best_alpha"] = best_ab["alpha"]; m["q2ff_best_beta"] = best_ab["beta"]
    print("1 月窗口 alpha/beta 最优: alpha=%.2f beta=%.1f (1 月费用 %.2f)" % (
        best_ab["alpha"], best_ab["beta"], best_ab["jan"]))
    print(" 前 5 名:", sorted([(x["alpha"], x["beta"], round(x["jan"], 1)) for x in ab], key=lambda y: y[2])[:5])
    al, be = best_ab["alpha"], best_ab["beta"]
# 2) W,p 在同一 1 月窗口上选
wp = []
for W, p in GRID:
    r = run(inp, al, be, W, p, 1, 31, 1)
    wp.append({"W": W, "p": p, "jan": r["total"]})
best_wp = min(wp, key=lambda x: x["jan"])
W, p = best_wp["W"], best_wp["p"]
m["q2ff_best_W"] = float(W); m["q2ff_best_p"] = p; m["q2ff_jan_cost"] = best_wp["jan"]
print("1 月窗口 W/p 最优: W=%d p=%.0f%% (1 月费用 %.2f)" % (W, p * 100, best_wp["jan"]))
# 3) 四个参数全部来自 1 月, 跑全年
r = run(inp, al, be, W, p, 1, 365, 31)
for k, v in r.items():
    if k != "soc_end": m["q2ff_" + k] = v
print("全年(alpha=%.2f beta=%.1f W=%d p=%.0f%%): 总 %.2f = 计划 %.2f + 紧急 %.2f (%.2f kWh) 弃电 %.1f" % (
    al, be, W, p * 100, r["total"], r["plan_cost"], r["em_cost"], r["em_kwh"], r["curtail"]))
# 4) 对照: 现交付的 alpha=0.15/beta=0.90 + (5,88)
r2 = run(inp, 0.15, 0.90, 5, 0.88, 1, 365, 31)
m["q2ff_ref_015_090_w5p88"] = r2["total"]
print("对照 alpha=0.15/beta=0.90 + (5,88%%): 总 %.2f" % r2["total"])
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": m, "ab_grid": ab, "wp_grid": wp}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
    ensure_ascii=False, indent=1)
print("done")

if __name__ == "__main__":
    main()

```

D.2.13 支撑材料/代码/02_问题二/q2_forecast.py

-*- coding: utf-8 -*-

"""问题 2 计划阶段的因果预测层与冗余层。

信息边界: 第 d 天 0:00 制定计划时, 只允许使用第 $d-1$ 天及更早的实际净负荷。

```

- EMA(指数加权) 与星期基线(周五/周六 vs 其余五天) 只用历史
- 冗余裕量取最近 W 天同段预测误差的 p 分位数(报童模型: 紧急电价 5 倍 => p 应接近 0.8)
"""
import numpy as np

def weekday_class(n_days):
    """2025-01-01 是周三; 返回 0/1 两类(周五、周六为 1)。"""
    wd = np.array([(2 + d) % 7 for d in range(n_days)])
    return np.where((wd == 4) | (wd == 5), 1, 0)

def build_forecast(net, alpha, beta):
    """逐日因果预测。返回 Nhat, err, ema_hist, cls。"""
    D, T = net.shape
    cls = weekday_class(D)
    Nhat = np.zeros((D, T)); err = np.zeros((D, T))
    ema_hist = np.zeros((D, T)); bsum = np.zeros((2, T)); bcnt = np.zeros(2)
    ema = None
    for d in range(D):
        if d == 0:
            Nhat[d] = net[0].copy(); err[d] = 0.0
        else:
            base = bsum[cls[d]] / bcnt[cls[d]] if bcnt[cls[d]] > 0 else ema
            Nhat[d] = (1.0 - beta) * ema + beta * base
            err[d] = net[d] - Nhat[d]
            ema = net[d].copy() if ema is None else alpha * net[d] + (1.0 - alpha) * ema
            ema_hist[d] = ema
            bsum[cls[d]] += net[d]; bcnt[cls[d]] += 1
    return Nhat, err, ema_hist, cls

def margin(err, d, W, p):
    """第 d 天各时段的安全裕量 = 最近 W 天预测误差的 p 分位数(只用 d 之前)。"""
    hi = d; lo = max(1, d - W)
    if hi <= lo:
        return np.zeros(err.shape[1])
    return np.quantile(err[lo:hi], p, axis=0)

def execute_causal(g, N, s0, eta, dt, Pmax, smin, smax):
    """口径 2 实时执行: 储能的净出力 = 实际净负荷 - 计划购电量。

    缺电则放电、富余则充电, 受功率上限与 SOC 边界限制; 仍补不上的部分走紧急购电。
    每个时段只看当前时刻的实际值, 不看当天未来值。返回 e, c, d, S。
    """
    T = len(g)
    s = float(s0)
    e = np.zeros(T); c = np.zeros(T); d = np.zeros(T); S = np.zeros(T + 1); S[0] = s
    for t in range(T):
        need = N[t] - g[t]
        if need > 0.0:
            dmax = min(Pmax, (s - smin) * eta / dt)
            dd = min(need, max(dmax, 0.0))

```

```

e[t] = need - dd; d[t] = dd
s -= dd * dt / eta
else:
    cmax = min(Pmax, (smax - s) / (eta * dt))
    cc = min(-need, max(cmax, 0.0))
    c[t] = cc
    s += eta * cc * dt
S[t + 1] = s
return e, c, d, S

```

D.2.14 支撑材料/代码/02_问题二/q2_leaktest.py

-*- coding: utf-8 -*-

"""问题2 因果性(无未来信息泄露)负对照检验.

目的: 证明第 d 天 0:00 的计划只依赖第 $d-1$ 天及更早的实际数据, 与人为参数 (α , β , W , p) 取值无关.

方法:

1. 机制级: 对每个检验日 d , 把预测误差序列 err 在索引 d 及以后任意篡改, 重算 $\text{margin}(err, d, W, p)$, 要求与原始逐元素完全相同 (容差 0);
2. 端到端: 用"未来被污染"的净负荷序列重跑同一条全年滚动优化, 要求所有 $d < \text{cut}$ 的逐日计划购电量与原始运行逐元素相同 (浮点容差 $1e-9$), 同时要求 $d \geq \text{cut}$ 的计划确实发生变化 (证明扰动有效, 检验非空洞).

```

"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal
from q2_online import day_lp

TOL = 1e-9

def rollout(net, price, st, alpha, beta, W, p, last_day):
    dt = float(st["dt"]); eta = float(st["efficiency"]); Pmax = float(st["power_max"])
    smin = float(st["soc_min"]); smax = float(st["soc_max"])
    Nhat, err, _, _ = build_forecast(net, alpha, beta)
    s = float(st["soc_initial"])
    gs = np.zeros((last_day, net.shape[1])); soc = np.zeros(last_day)
    for d in range(1, last_day):
        Nplan = Nhat[d] + margin(err, d, W, p)
        g, _, _, _ = day_lp(price, Nplan, s, eta, dt, Pmax, smin, smax)
        gs[d] = g
        _, _, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
        s = float(S[-1]); soc[d] = s
    return gs, soc, err

```

```

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--alpha", type=float, default=0.15); ap.add_argument("--beta", type=float, default=0.9)
    ap.add_argument("--W", type=int, default=7); ap.add_argument("--p", type=float, default=0.88)
    ap.add_argument("--cut-day", type=int, default=200)
    ap.add_argument("--first-day", type=int, default=31); ap.add_argument("--last-day", type=int, default=365)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float)

```

```

net = np.asarray(inp["load_actual"], float) - np.asarray(inp["pv_actual"], float)
st = dict(inp["storage"]); st["dt"] = float(inp["dt"])
dt = st["dt"]; D, T = net.shape

g0, s0, err0 = rollout(net, price, st, a.alpha, a.beta, a.W, a.p, a.last_day)

# --- 1. 机制级: margin 只允许看 d 之前 ---
mdiff = 0.0
d_probe = a.cut_day
base = margin(err0, d_probe, a.W, a.p)
rng = np.random.RandomState(20250910)
for k in range(50):
    e2 = err0.copy()
    e2[d_probe:] = rng.normal(0.0, 5000.0, size=e2[d_probe:].shape) # 篡改当天与未来
    mdiff = max(mdiff, float(np.max(np.abs(margin(e2, d_probe, a.W, a.p) - base))))

# --- 2. 端到端: 污染未来, 前缀计划必须逐元素不变 ---
prefix_diff = 0.0; suffix_diff = 0.0; prefix_cost0 = 0.0; prefix_cost1 = 0.0
modes = {}
net_scale = net.copy(); net_scale[a.cut_day:] = net[a.cut_day:] * 3.0
modes["future_scale3"] = net_scale
perm = rng.permutation(np.arange(a.cut_day, D))
net_shuf = net.copy(); net_shuf[a.cut_day:] = net[perm]
modes["future_shuffle"] = net_shuf
net_zero = net.copy(); net_zero[a.cut_day:] = 0.0
modes["future_zero"] = net_zero
for name, net2 in modes.items():
    g1, s1, _ = rollout(net2, price, st, a.alpha, a.beta, a.W, a.p, a.last_day)
    pd_ = float(np.max(np.abs(g1[1:a.cut_day] - g0[1:a.cut_day])))
    sd_ = float(np.max(np.abs(g1[a.cut_day:] - g0[a.cut_day:])))
    prefix_diff = max(prefix_diff, pd_); suffix_diff = max(suffix_diff, sd_)
    c0 = float(sum(price @ (g0[d] * dt) for d in range(a.first_day, a.cut_day)))
    c1 = float(sum(price @ (g1[d] * dt) for d in range(a.first_day, a.cut_day)))
    prefix_cost0, prefix_cost1 = c0, c1

verdict = 1.0 if (prefix_diff <= TOL and mdiff == 0.0 and suffix_diff > 1.0) else 0.0
metrics = {
    "q2_leaktest_margin_maxdiff_kw": mdiff,
    "q2_leaktest_prefix_plan_maxdiff_kwh": prefix_diff,
    "q2_leaktest_suffix_plan_maxdiff_kwh": suffix_diff,
    "q2_leaktest_prefix_cost_original": prefix_cost0,
    "q2_leaktest_prefix_cost_corrupted": prefix_cost1,
    "q2_leaktest_prefix_days": float(a.cut_day - a.first_day),
    "q2_leaktest_margin_probes": 50.0,
    "q2_leaktest_corruption_modes": float(len(modes)),

```

```

        "q2_leaktest_verdict": verdict,
    }
    os.makedirs(a.out, exist_ok=True)
    json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"), ensure_ascii=False)
    print("leaktest verdict=%0.0f prefix_maxdiff=%0.3e margin_maxdiff=%0.3e suffix_maxdiff=%0.1f"
          % (verdict, prefix_diff, mdiff, suffix_diff))

if __name__ == "__main__":
    main()

```

D.2.15 支撑材料/代码/02_问题二/q2_lp.py

-*- coding: utf-8 -*-

```

import os, json, argparse
import numpy as np
from scipy.optimize import linprog

def forecast_ma(arr, d, window=28):
    if d == 0: return arr[0]
    lo = max(0, d - window)
    return arr[lo:d].mean(axis=0)

def forecast_persist(arr, d):
    return arr[max(0, d-1)]

def solve_day_lp(price, net_f, soc0, eta, dt, Pmax, soc_min, soc_max):
    T = len(price); n = 3*T + (T+1)
    c = np.zeros(n); c[0:T] = price*dt
    bounds = [(0, None)]*T + [(0, Pmax)]*T + [(0, Pmax)]*T + [(soc0, soc0)]
    for _ in range(1, T): bounds.append((soc_min, soc_max))
    bounds.append((soc_min, soc_max))
    A_ub = np.zeros((T, n)); b_ub = -net_f
    for t in range(T): A_ub[t, t] = -1.0; A_ub[t, T+t] = 1.0; A_ub[t, 2*T+t] = -1.0
    A_eq = np.zeros((T, n)); b_eq = np.zeros(T)
    for t in range(T):
        A_eq[t, 3*T+t+1] = 1.0; A_eq[t, 3*T+t] = -1.0; A_eq[t, T+t] = -eta*dt; A_eq[t, 2*T+t] = dt/eta
    res = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs")
    if not res.success: raise RuntimeError(res.message)
    x = res.x
    return x[0:T], x[T:2*T], x[2*T:3*T], x[3*T:3*T+T+1], float(np.dot(price, x[0:T])*dt)

def main():
    ap = argparse.ArgumentParser(); ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float); load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
    Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]
    D = load.shape[0]; soc = st["soc_initial"]
    plans = []; total_planned = 0.0; total_emergency = 0.0; total_emerg_kwh = 0.0
    for d in range(D):
        lf = forecast_ma(load, d); pf = forecast_ma(pv, d)

```

```

p, c, dd, s, pcost = solve_day_lp(price, lf-pf, soc, eta, dt, Pmax, soc_min, soc_max)
E = np.maximum(0.0, load[d] + c - pv[d] - p - dd)
ecost = 5.0*np.dot(price, E)*dt
if d >= 31:
    total_planned += pcost; total_emergency += ecost; total_emerg_kwh += float(E.sum()*dt)
    plans.append(dict(day=d, purchase=p.tolist(), charge=c.tolist(), discharge=dd.tolist(), soc=s.tolist(),
emergency=E.tolist()))
    soc = float(s[-1])
metrics = {"q2_total_cost": total_planned+total_emergency, "q2_planned_cost": total_planned,
"q2_emergency_cost": total_emergency, "q2_emergency_kwh": total_emerg_kwh}
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"))
json.dump({"plans": plans, "price": price.tolist(), open(os.path.join(a.out, "solution.json"), "w", encoding="utf-8"))
print("q2_total_cost=", metrics["q2_total_cost"], "planned=", total_planned, "emergency=", total_emergency)
if __name__ == "__main__": main()

```

D.2.16 支撑材料/代码/02_问题二/q2_margin.py

-*- coding: utf-8 -*-

```

import os, json, argparse
import numpy as np
from scipy.optimize import linprog

def forecast_persist(arr, d):
    return arr[max(0, d-1)]

def solve_day_lp(price, net_margin, soc0, eta, dt, Pmax, soc_min, soc_max):
    T = len(price); n = 3*T + (T+1)
    c = np.zeros(n); c[0:T] = price*dt
    bounds = [(0, None)]*T + [(0, Pmax)]*T + [(0, Pmax)]*T + [(soc0, soc0)]
    for _ in range(1, T): bounds.append((soc_min, soc_max))
    bounds.append((soc_min, soc_max))
    A_ub = np.zeros((T, n)); b_ub = -net_margin
    for t in range(T): A_ub[t, t] = -1.0; A_ub[t, T+t] = 1.0; A_ub[t, 2*T+t] = -1.0
    A_eq = np.zeros((T, n)); b_eq = np.zeros(T)
    for t in range(T):
        A_eq[t, 3*T+t+1] = 1.0; A_eq[t, 3*T+t] = -1.0; A_eq[t, T+t] = -eta*dt; A_eq[t, 2*T+t] = dt/eta
    res = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs")
    if not res.success: raise RuntimeError(res.message)
    x = res.x
    return x[0:T], x[T:2*T], x[2*T:3*T], x[3*T:3*T+T+1], float(np.dot(price, x[0:T])*dt)

def main():
    ap = argparse.ArgumentParser(); ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float); load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
    Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]
    D, T = load.shape; soc = st["soc_initial"]
    net = load - pv

```

```

window = 28; pct = 80.0
tp=0.0; te=0.0; tp_nomargin=0.0; te_nomargin=0.0
for d in range(D):
    lf = forecast_persist(load, d); pf = forecast_persist(pv, d)
    nf = lf - pf
    # margin = 80th percentile of past day-over-day net errors (no leakage)
    if d >= 2:
        lo = max(1, d - window)
        errors = net[lo:d] - net[lo-1:d-1] # net[k]-net[k-1] for k=lo..d-1
        margin = np.maximum(np.percentile(errors, pct, axis=0), 0.0)
    else:
        margin = np.zeros(T)
    # plan with margin
    p, c, dd, s, pcost = solve_day_lp(price, nf + margin, soc, eta, dt, Pmax, soc_min, soc_max)
    E = np.maximum(0.0, net[d] + c - p - dd) # emergency with margin plan
    # plan without margin (for comparison)
    p0, c0, dd0, s0, pcost0 = solve_day_lp(price, nf, soc, eta, dt, Pmax, soc_min, soc_max)
    E0 = np.maximum(0.0, net[d] + c0 - p0 - dd0)
    if d >= 31:
        tp += pcost; te += 5.0*np.dot(price, E)*dt
        tp_nomargin += pcost0; te_nomargin += 5.0*np.dot(price, E0)*dt
    soc = float(s[-1])
    metrics = {"q2_total_cost": tp+te, "q2_planned_cost": tp, "q2_emergency_cost": te,
              "q2_no_margin_total": tp_nomargin+te_nomargin, "q2_no_margin_emergency": te_nomargin}
    os.makedirs(a.out, exist_ok=True)
    json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"))
    print(f'margin: total={tp+te:.2f} planned={tp:.2f} emergency={te:.2f}')
    print(f'no-margin: total={tp_nomargin+te_nomargin:.2f} emergency={te_nomargin:.2f}')
if __name__ == "__main__": main()

```

D.2.17 支撑材料/代码/02_问题二/q2_no_storage.py

-*- coding: utf-8 -*-

```

import os, json, argparse
import numpy as np
from scipy.optimize import linprog

def forecast_ma(arr, d, window=28):
    if d == 0: return arr[0]
    lo = max(0, d - window)
    return arr[lo:d].mean(axis=0)

def forecast_persist(arr, d):
    return arr[max(0, d-1)]

def main():
    ap = argparse.ArgumentParser(); ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float); load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    dt = inp["dt"]

```

```

D = load.shape[0]; total_planned=0.0; total_emergency=0.0
for d in range(D):
    lf = forecast_ma(load, d); pf = forecast_ma(pv, d)
    p = np.maximum(0.0, lf-pf) # plan: buy forecast net
    E = np.maximum(0.0, load[d]-pv[d]-p)
    if d >= 31:
        total_planned += float(np.dot(price, p)*dt)
        total_emergency += 5.0*np.dot(price, E)*dt
    metrics = {"q2_total_cost": total_planned+total_emergency, "q2_planned_cost": total_planned, "q2_emergency_cost":
total_emergency}
    os.makedirs(a.out, exist_ok=True)
    json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"))
    print("q2_total_cost(no-storage)=", total_planned+total_emergency)
if __name__ == "__main__": main()

```

D.2.18 支撑材料/代码/02_问题二/q2_online.py

-*- coding: utf-8 -*-

"""问题2 主路线 (q2-lp-newsvendor): 无泄露逐日滚动 + 口径2 实时储能执行。

```

计划层(每天 0:00, 只用历史): 预测净负荷 -> 叠加报童冗余 -> 解日内 LP 得计划购电量
执行层(每 10 分钟, 全年连续, 无日界): 储能实时补缺/吸纳, SOC 全程  $\in [soc\_min, soc\_max]$ 
结算: 计划购电量按计划计费; 实际不足负载的部分按 5 倍电价紧急购电
问题2 不含 0:00=24:00 储电量等式(该条件仅属问题1), 日末储电量自由
"""
import os, json, argparse
import numpy as np
from scipy.optimize import linprog

import sys
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal

ETA_DEF, DT_DEF = 0.9, 1.0 / 6.0

def day_lp(price, Nplan, s0, eta, dt, Pmax, smin, smax):
    """日内计划 LP:  $\min \sum p g \Delta t$  s.t.  $g + d - c \geq Nplan$ , SOC 演化,  $S_0=s0$ ,  $S_{144}$  自由。"""
    T = len(price)
    n = 3 * T + (T + 1)
    og, oc, od, os_ = 0, T, 2 * T, 3 * T
    c = np.zeros(n); c[og:og + T] = price * dt
    A_ub = np.zeros((T, n)); b_ub = -Nplan
    for t in range(T):
        A_ub[t, og + t] = -1.0; A_ub[t, od + t] = -1.0; A_ub[t, oc + t] = 1.0
    A_eq = np.zeros((T, n)); b_eq = np.zeros(T)
    for t in range(T):
        A_eq[t, os_ + t + 1] = 1.0; A_eq[t, os_ + t] = -1.0
        A_eq[t, oc + t] = -eta * dt; A_eq[t, od + t] = dt / eta
    bnds = [(0, None)] * T + [(0, Pmax)] * T + [(0, Pmax)] * T
    bnds += [(s0, s0)] + [(smin, smax)] * T
    r = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bnds, method="highs")
    if not r.success:
        raise RuntimeError(r.message)
    x = r.x
    return x[og:og + T], x[oc:oc + T], x[od:od + T], x[os_:os_ + T + 1]

```

```

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--alpha", type=float, default=0.15)
    ap.add_argument("--beta", type=float, default=0.9)
    ap.add_argument("--W", type=int, default=7)
    ap.add_argument("--p", type=float, default=0.88)
    ap.add_argument("--first-day", type=int, default=31)
    ap.add_argument("--last-day", type=int, default=365)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float)
    load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = float(inp["dt"]); eta = float(st["efficiency"])
    Pmax = float(st["power_max"]); smin = float(st["soc_min"]); smax = float(st["soc_max"])
    net = load - pv
    Nhat, err, _, _ = build_forecast(net, a.alpha, a.beta)

    s = float(st["soc_initial"])
    plan_cost = plan_kwh = em_cost = em_kwh = 0.0
    ch = di = 0.0; nslot = 0; days = 0
    soc_min_seen = 1e18; soc_max_seen = -1e18
    daily = []
    for d in range(1, a.last_day):
        Nplan = Nhat[d] + margin(err, d, a.W, a.p)
        g, _, _, _ = day_lp(price, Nplan, s, eta, dt, Pmax, smin, smax)
        e, cc, dd, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
        soc_min_seen = min(soc_min_seen, float(S.min())); soc_max_seen = max(soc_max_seen, float(S.max()))
        if d >= a.first_day:
            plan_cost += float(price @ (g * dt)); plan_kwh += float(g.sum() * dt)
            em_cost += float(5.0 * price @ (e * dt)); em_kwh += float(e.sum() * dt)
            ch += float(cc.sum() * dt); di += float(dd.sum() * dt)
            nslot += int((e > 1e-6).sum()); days += 1
            daily.append({"day": d, "plan_kwh": float(g.sum() * dt), "plan_cost": float(price @ (g * dt)),
                          "em_kwh": float(e.sum() * dt), "em_cost": float(5.0 * price @ (e * dt)),
                          "charge": float(cc.sum() * dt), "discharge": float(dd.sum() * dt),
                          "soc_start": float(S[0]), "soc_end": float(S[-1])})
        s = float(S[-1])
    loss = 0.1 * ch + (0.1 / eta) * di * (eta / 0.9) # 见下方说明
    loss = (1 - eta) * ch + ((1 - eta) / eta) * di
    metrics = {
        "q2_planned_cost": plan_cost, "q2_planned_kwh": plan_kwh,
        "q2_emergency_cost": em_cost, "q2_emergency_kwh": em_kwh,
        "q2_total_cost": plan_cost + em_cost,
        "q2_charge_kwh": ch, "q2_discharge_kwh": di, "q2_loss_kwh": loss,
        "q2_emergency_slots": float(nslot), "q2_days": float(days),
    }

```

```

        "q2_soc_min": soc_min_seen, "q2_soc_max": soc_max_seen,
        "q2_soc_end": float(s),
    }
    os.makedirs(a.out, exist_ok=True)
    json.dump({"metrics": metrics, "daily": daily},
              open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"), ensure_ascii=False)
    json.dump({"params": vars(a), "daily": daily},
              open(os.path.join(a.out, "solution.json"), "w", encoding="utf-8"), ensure_ascii=False)
    print("q2_lp_newsvendor total_cost=%0.2f planned=%0.2f emergency=%0.2f kwh=%0.1f loss=%0.1f soc=[%0.1f,%0.1f]"
          % (metrics["q2_total_cost"], plan_cost, em_cost, em_kwh, loss, soc_min_seen, soc_max_seen))

if __name__ == "__main__":
    main()

```

D.2.19 支撑材料/代码/02_问题二/q2_perfect.py

-*- coding: utf-8 -*-

```

import os, json, argparse
import numpy as np
from scipy.optimize import linprog

def solve_day_lp(price, net, soc0, eta, dt, Pmax, soc_min, soc_max):
    T = len(price); n = 3*T + (T+1)
    c = np.zeros(n); c[0:T] = price*dt
    bounds = [(0,None)]*T + [(0,Pmax)]*T + [(0,Pmax)]*T + [(soc0,soc0)]
    for _ in range(1,T): bounds.append((soc_min,soc_max))
    bounds.append((soc_min,soc_max))
    A_ub = np.zeros((T,n)); b_ub = -net
    for t in range(T): A_ub[t,t]=-1.0; A_ub[t,T+t]=1.0; A_ub[t,2*T+t]=-1.0
    A_eq = np.zeros((T,n)); b_eq = np.zeros(T)
    for t in range(T):
        A_eq[t,3*T+t+1]=1.0; A_eq[t,3*T+t]=-1.0; A_eq[t,T+t]=-eta*dt; A_eq[t,2*T+t]=dt/eta
    res = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs")
    if not res.success: raise RuntimeError(res.message)
    x = res.x
    return float(np.dot(price, x[0:T])*dt), float(x[3*T:3*T+T+1][-1])

def main():
    ap = argparse.ArgumentParser(); ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float); load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
    Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]
    D = load.shape[0]; soc = st["soc_initial"]; total = 0.0
    for d in range(D):
        pcost, soc_end = solve_day_lp(price, load[d]-pv[d], soc, eta, dt, Pmax, soc_min, soc_max)
        if d >= 31: total += pcost
        soc = soc_end
    metrics = {"q2_perfect_cost": total, "q2_perfect_emergency": 0.0, "q2_total_cost": total}
    os.makedirs(a.out, exist_ok=True)

```

```

    json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"))
    print("q2_perfect_cost (theoretical minimum) =", total)
if __name__ == "__main__": main()

```

D.2.20 支撑材料/代码/02_问题二/q2_persistence.py

-*- coding: utf-8 -*-

```

import os, json, argparse
import numpy as np
from scipy.optimize import linprog

def forecast_persist(arr, d):
    return arr[max(0, d-1)]

def solve_day_lp(price, net_f, soc0, eta, dt, Pmax, soc_min, soc_max):
    T = len(price); n = 3*T + (T+1)
    c = np.zeros(n); c[0:T] = price*dt
    bounds = [(0, None)]*T + [(0, Pmax)]*T + [(0, Pmax)]*T + [(soc0, soc0)]
    for _ in range(1, T): bounds.append((soc_min, soc_max))
    bounds.append((soc_min, soc_max))
    A_ub = np.zeros((T, n)); b_ub = -net_f
    for t in range(T): A_ub[t, t] = -1.0; A_ub[t, T+t] = 1.0; A_ub[t, 2*T+t] = -1.0
    A_eq = np.zeros((T, n)); b_eq = np.zeros(T)
    for t in range(T):
        A_eq[t, 3*T+t+1] = 1.0; A_eq[t, 3*T+t] = -1.0; A_eq[t, T+t] = -eta*dt; A_eq[t, 2*T+t] = dt/eta
    res = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs")
    if not res.success: raise RuntimeError(res.message)
    x = res.x
    return x[0:T], x[T:2*T], x[2*T:3*T], x[3*T:3*T+T+1], float(np.dot(price, x[0:T])*dt)

def main():
    ap = argparse.ArgumentParser(); ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float); load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
    Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]
    D = load.shape[0]; soc = st["soc_initial"]; tp=0.0; te=0.0; plans=[]
    for d in range(D):
        lf = forecast_persist(load, d); pf = forecast_persist(pv, d)
        p, c, dd, s, pcost = solve_day_lp(price, lf-pf, soc, eta, dt, Pmax, soc_min, soc_max)
        E = np.maximum(0.0, load[d] + c - pv[d] - p - dd)
        if d >= 31:
            tp += pcost; te += 5.0*np.dot(price, E)*dt
            plans.append(dict(day=d, purchase=p.tolist(), charge=c.tolist(), discharge=dd.tolist(), soc=s.tolist(),
emergency=E.tolist()))
        soc = float(s[-1])
    metrics = {"q2_total_cost": tp+te, "q2_planned_cost": tp, "q2_emergency_cost": te}
    os.makedirs(a.out, exist_ok=True)
    json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"))
    json.dump({"plans": plans, "price": price.tolist()}, open(os.path.join(a.out, "solution.json"), "w", encoding="utf-8"))
    print("q2_total_cost(persistence)=", tp+te)
if __name__ == "__main__": main()

```

D.2.21 支撑材料/代码/02_问题二/q2_replica_dp.py

-*- coding: utf-8 -*-

```
"""问题2 复算路径 B(动态规划): 日内计划用 SOC 离散化 min-plus 逆向递推求解, 不调用线性规划求解器。"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal

def run(a, solve_day):
    inp = json.load(open(a.input, encoding='utf-8'))
    price = np.asarray(inp['price'], float)
    net = np.asarray(inp['load_actual'], float) - np.asarray(inp['pv_actual'], float)
    st = inp['storage']; dt = float(inp['dt']); eta = float(st['efficiency'])
    Pmax = float(st['power_max']); smin = float(st['soc_min']); smax = float(st['soc_max'])
    Nhat, err, _, _ = build_forecast(net, a.alpha, a.beta)
    s = float(st['soc_initial'])
    pc = pk = ec = ek = ch = di = 0.0; nslot = 0; lo = 1e18; hi = -1e18
    for d in range(1, a.last_day):
        g = solve_day(price, Nhat[d] + margin(err, d, a.W, a.p), s, eta, dt, Pmax, smin, smax)
        e, cc, dd, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
        lo = min(lo, float(S.min())); hi = max(hi, float(S.max()))
        if d >= a.first_day:
            pc += float(price @ (g * dt)); pk += float(g.sum() * dt)
            ec += float(5.0 * price @ (e * dt)); ek += float(e.sum() * dt)
            ch += float(cc.sum() * dt); di += float(dd.sum() * dt); nslot += int((e > 1e-6).sum())
        s = float(S[-1])
    m = {'q2_planned_cost': pc, 'q2_planned_kwh': pk, 'q2_emergency_cost': ec, 'q2_emergency_kwh': ek,
        'q2_total_cost': pc + ec, 'q2_charge_kwh': ch, 'q2_discharge_kwh': di,
        'q2_loss_kwh': (1 - eta) * ch + ((1 - eta) / eta) * di,
        'q2_soc_min': lo, 'q2_soc_max': hi, 'q2_soc_range': hi - lo,
        'q2_emergency_slots': float(nslot), 'q2_days': float(a.last_day - a.first_day)}
    os.makedirs(a.out, exist_ok=True)
    json.dump({'metrics': m}, open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'), ensure_ascii=False)
    print('total=%.2f soc_range=%.2f % (m['q2_total_cost'], m['q2_soc_range']))

def cli():
    ap = argparse.ArgumentParser()
    ap.add_argument('--input', required=True); ap.add_argument('--out', required=True)
    ap.add_argument('--alpha', type=float, default=0.15); ap.add_argument('--beta', type=float, default=0.9)
    ap.add_argument('--W', type=int, default=7); ap.add_argument('--p', type=float, default=0.88)
    ap.add_argument('--ds', type=float, default=2.0)
    ap.add_argument('--first-day', type=int, default=31); ap.add_argument('--last-day', type=int, default=365)
    return ap.parse_args()

from q2_dp import solve_day_dp

def solve_day(price, Nplan, s0, eta, dt, Pmax, smin, smax, ds=2.0):
    _, g, _, _, _ = solve_day_dp(price, Nplan, s0, eta, dt, Pmax, smin, smax, ds)
    return g
```

```

if __name__ == '__main__':
    a = cli()

    run(a, lambda pr, np_, s0, eta, dt, Pmax, smin, smax: solve_day(pr, np_, s0, eta, dt, Pmax, smin, smax, a.ds))

```

D.2.22 支撑材料/代码/02_问题二/q2_replica_lp.py

-*- coding: utf-8 -*-

```

"""问题2 复算路径 A(线性规划): 日内计划用 HiGHS 线性规划求解, 记录含 SOC 区间的全部预注册指标。"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal

def run(a, solve_day):
    inp = json.load(open(a.input, encoding='utf-8'))

    price = np.asarray(inp['price'], float)
    net = np.asarray(inp['load_actual'], float) - np.asarray(inp['pv_actual'], float)
    st = inp['storage']; dt = float(inp['dt']); eta = float(st['efficiency'])
    Pmax = float(st['power_max']); smin = float(st['soc_min']); smax = float(st['soc_max'])
    Nhat, err, _, _ = build_forecast(net, a.alpha, a.beta)
    s = float(st['soc_initial'])
    pc = pk = ec = ek = ch = di = 0.0; nslot = 0; lo = 1e18; hi = -1e18
    for d in range(1, a.last_day):
        g = solve_day(price, Nhat[d] + margin(err, d, a.W, a.p), s, eta, dt, Pmax, smin, smax)
        e, cc, dd, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
        lo = min(lo, float(S.min())); hi = max(hi, float(S.max()))
        if d >= a.first_day:
            pc += float(price @ (g * dt)); pk += float(g.sum() * dt)
            ec += float(5.0 * price @ (e * dt)); ek += float(e.sum() * dt)
            ch += float(cc.sum() * dt); di += float(dd.sum() * dt); nslot += int((e > 1e-6).sum())
        s = float(S[-1])
    m = {'q2_planned_cost': pc, 'q2_planned_kwh': pk, 'q2_emergency_cost': ec, 'q2_emergency_kwh': ek,
        'q2_total_cost': pc + ec, 'q2_charge_kwh': ch, 'q2_discharge_kwh': di,
        'q2_loss_kwh': (1 - eta) * ch + ((1 - eta) / eta) * di,
        'q2_soc_min': lo, 'q2_soc_max': hi, 'q2_soc_range': hi - lo,
        'q2_emergency_slots': float(nslot), 'q2_days': float(a.last_day - a.first_day)}
    os.makedirs(a.out, exist_ok=True)
    json.dump({'metrics': m}, open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'), ensure_ascii=False)
    print('total=%0.2f soc_range=%0.2f % (m['q2_total_cost'], m['q2_soc_range']))

def cli():
    ap = argparse.ArgumentParser()
    ap.add_argument('--input', required=True); ap.add_argument('--out', required=True)
    ap.add_argument('--alpha', type=float, default=0.15); ap.add_argument('--beta', type=float, default=0.9)
    ap.add_argument('--W', type=int, default=7); ap.add_argument('--p', type=float, default=0.88)
    ap.add_argument('--ds', type=float, default=2.0)
    ap.add_argument('--first-day', type=int, default=31); ap.add_argument('--last-day', type=int, default=365)
    return ap.parse_args()

from q2_online import day_lp

def solve_day(price, Nplan, s0, eta, dt, Pmax, smin, smax):
    g, _, _, _ = day_lp(price, Nplan, s0, eta, dt, Pmax, smin, smax)

```

```
return g
```

```
if __name__ == '__main__':  
    run(cli(), solve_day)
```

D.2.23 支撑材料/代码/02_问题二/q2_result2.py

-*- coding: utf-8 -*-

"""问题2 结果文件生成：用主路线(q2-lp-newsvendor, 口径②)生成 result2.xlsx 并记录指标。

计划层：每天 0:00 只用历史数据预测净负荷 + 报童冗余 -> 日内 LP 定当天计划购电量；
执行层：储能按口径② 实时吸收预测误差，SOC 全程在 [1200,10800] 内，不足部分走 5 倍紧急购电；
问题2 不含 0:00=24:00 储电量等式(该条件仅属问题1)。

```
import os, sys, json, argparse  
import numpy as np
```

```
import openpyxl  
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))  
from q2_forecast import build_forecast, margin, execute_causal  
from q2_online import day_lp
```

```
PERIODS = ['0:00-4:00', '4:00-8:00', '8:00-12:00', '12:00-16:00', '16:00-20:00', '20:00-24:00']
```

```
def slot_label(i):  
    a = i * 10; b = a + 10  
  
    fmt = lambda m: '%d:%02d' % (m // 60, m % 60)  
  
    return '%s-%s' % (fmt(a), '0:00+1' if b == 1440 else fmt(b))
```

```
def main():  
    ap = argparse.ArgumentParser()  
    ap.add_argument('--input', required=True); ap.add_argument('--out', required=True)  
    ap.add_argument('--template', default='../problem_files/附件/附件 5/result2.xlsx')  
    ap.add_argument('--xlsx-out', default='result2.xlsx')  
    ap.add_argument('--alpha', type=float, default=0.15); ap.add_argument('--beta', type=float, default=0.9)  
    ap.add_argument('--W', type=int, default=7); ap.add_argument('--p', type=float, default=0.88)  
    ap.add_argument('--first-day', type=int, default=31); ap.add_argument('--last-day', type=int, default=365)  
    a = ap.parse_args()  
    inp = json.load(open(a.input, encoding='utf-8'))  
    price = np.asarray(inp['price'], float)  
    net = np.asarray(inp['load_actual'], float) - np.asarray(inp['pv_actual'], float)  
    st = inp['storage']; dt = float(inp['dt']); eta = float(st['efficiency'])  
    Pmax = float(st['power_max']); smin = float(st['soc_min']); smax = float(st['soc_max'])  
    T = len(price)  
    Nhat, err, _, _ = build_forecast(net, a.alpha, a.beta)  
    s = float(st['soc_initial'])  
    G = np.zeros((a.last_day, T)); E = np.zeros((a.last_day, T))  
    C = np.zeros((a.last_day, T)); D = np.zeros((a.last_day, T))  
    S0 = np.zeros(a.last_day); S1 = np.zeros(a.last_day)  
    for d in range(1, a.last_day):  
        g, cc, dd, _ = day_lp(price, Nhat[d] + margin(err, d, a.W, a.p), s, eta, dt, Pmax, smin, smax)  
        e, c2, d2, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)  
        G[d] = g; E[d] = e; C[d] = c2; D[d] = d2; S0[d] = s; S1[d] = float(S[-1])  
        s = float(S[-1])  
    sl = slice(a.first_day, a.last_day)  
    pc = float(sum(price @ (G[d] * dt) for d in range(a.first_day, a.last_day)))
```

```

ec = float(sum(5.0 * price @ (E[d] * dt) for d in range(a.first_day, a.last_day)))
ch = float(C[sl].sum() * dt); di = float(D[sl].sum() * dt)
metrics = {'q2_planned_cost': pc, 'q2_planned_kwh': float(G[sl].sum() * dt),
           'q2_emergency_cost': ec, 'q2_emergency_kwh': float(E[sl].sum() * dt),
           'q2_total_cost': pc + ec, 'q2_charge_kwh': ch, 'q2_discharge_kwh': di,
           'q2_loss_kwh': (1 - eta) * ch + ((1 - eta) / eta) * di,
           'q2_emergency_slots': float((E[sl] > 1e-6).sum()),
           'q2_days': float(a.last_day - a.first_day),
           'q2_soc_min': float(min(S0[sl].min(), S1[sl].min())),
           'q2_soc_max': float(max(S0[sl].max(), S1[sl].max()))}
os.makedirs(a.out, exist_ok=True)
json.dump({'metrics': metrics}, open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'), ensure_ascii=False)
# ---- 写 result2.xlsx ----
tpl = os.path.abspath(os.path.join(os.path.dirname(os.path.abspath(__file__)), a.template))
wb = openpyxl.load_workbook(tpl)
ws = wb['计划购电量']
r = 2
for d in range(a.first_day, a.last_day):
    ws.cell(row=r, column=1, value=ws.cell(row=r, column=1).value)
    for t in range(T):
        ws.cell(row=r, column=2 + t, value=round(float(G[d, t] * dt), 4))
    ws.cell(row=r, column=2 + T, value=round(float(G[d].sum() * dt), 4))
    ws.cell(row=r, column=3 + T, value=round(float(price @ (G[d] * dt)), 2))
    r += 1
ws2 = wb['充放电量']
ws2.delete_rows(2, ws2.max_row)
r = 2
import datetime
for d in range(a.first_day, a.last_day):
    date = datetime.datetime(2025, 2, 1) + datetime.timedelta(days=d - a.first_day)
    for k, lbl in enumerate(PERIODS):
        lo, hi = k * 24, (k + 1) * 24
        ws2.cell(row=r, column=1, value=date if k == 0 else None)
        ws2.cell(row=r, column=2, value=lbl)
        ws2.cell(row=r, column=3, value=round(float(C[d, lo:hi].sum() * dt), 4))
        ws2.cell(row=r, column=4, value=round(float(D[d, lo:hi].sum() * dt), 4))
        if k == 0:
            ws2.cell(row=r, column=5, value=datetime.time(0, 0)); ws2.cell(row=r, column=6, value=round(float(S0[d]), 4))
        elif k == 1:
            ws2.cell(row=r, column=5, value='24:00'); ws2.cell(row=r, column=6, value=round(float(S1[d]), 4))
        r += 1
ws3 = wb['紧急购电量']
ws3.delete_rows(2, ws3.max_row)
r = 2
for d in range(a.first_day, a.last_day):
    date = datetime.datetime(2025, 2, 1) + datetime.timedelta(days=d - a.first_day)
    j = 0; runs = []
    while j < T:

```

```

if E[d, j] > 1e-6:
    k = j
    while k < T and E[d, k] > 1e-6: k += 1
    runs.append((j, k, float(E[d, j:k].sum() * dt))); j = k
else: j += 1
if runs:
    for idx, (x, y, en) in enumerate(runs):
        ws3.cell(row=r, column=1, value=date if idx == 0 else None)
        ws3.cell(row=r, column=2, value='%s-%s' % (slot_label(x).split('-')[0], slot_label(y - 1).split('-')[1]))
        ws3.cell(row=r, column=3, value=round(en, 4)); r += 1
    else:
        ws3.cell(row=r, column=1, value=date); ws3.cell(row=r, column=3, value=0.0); r += 1
outp = os.path.abspath(os.path.join(os.path.dirname(os.path.abspath(__file__)), '..', a.xlsx_out))
wb.save(outp)
print('result2.xlsx saved ->', outp)
print('q2 planned=%0.2f emergency=%0.2f total=%0.2f kwh=%0.1f soc=[%0.1f,%0.1f]'
      % (pc, ec, pc + ec, float(E[sl].sum() * dt), metrics['q2_soc_min'], metrics['q2_soc_max']))

if __name__ == '__main__':
    main()

```

D.2.24 支撑材料/代码/02_问题二/q2_robust.py

-*- coding: utf-8 -*-

```

"""问题2 稳健性：自助重采样(多随机种子) / 参数网格灵敏度 / 约束压力测试 / 消融对照。"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal, weekday_class
from q2_online import day_lp

def load(p):
    inp = json.load(open(p, encoding="utf-8"))
    price = np.asarray(inp["price"], float)
    net = np.asarray(inp["load_actual"], float) - np.asarray(inp["pv_actual"], float)
    st = dict(inp["storage"]); st["dt"] = float(inp["dt"])
    return price, net, st

def run(price, net, st, alpha, beta, W, p, margin_source=None, smin=None, smax=None,
        eta=None, first_day=31, last_day=365, no_margin=False, forecast_mode="structured",
        boot_seed=None):
    dt = st["dt"]; eta = st["efficiency"] if eta is None else eta
    Pmax = st["power_max"]; smin = st["soc_min"] if smin is None else smin
    smax = st["soc_max"] if smax is None else smax
    T = len(price)
    if forecast_mode == "persistence":
        Nhat = np.zeros_like(net); Nhat[0] = net[0]; Nhat[1:] = net[:-1]
    else:
        Nhat, _, _ = build_forecast(net, alpha, beta)
    err = net - Nhat
    if margin_source is None:
        margin_source = err

```

```

s = st["soc_initial"]
rng = np.random.RandomState(boot_seed) if boot_seed is not None else None
pc = pk = ec = ek = ch = di = 0.0; days = 0
smin_seen = 1e18; smax_seen = -1e18
for d in range(1, last_day):
    if no_margin:
        mm = np.zeros(T)
    elif rng is not None:
        hi = d; lo = max(1, d - W)
        pool = margin_source[lo:hi] if hi > lo else margin_source[0:1]
        pick = rng.randint(0, len(pool), size=W) # 对最近 W 天的误差做自助重采样
        mm = np.quantile(pool[pick], p, axis=0)
    else:
        mm = margin(margin_source, d, W, p)
    g, _, _ = day_lp(price, Nhat[d] + mm, s, eta, dt, Pmax, smin, smax)
    e, cc, dd, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
    if d >= first_day:
        pc += float(price @ (g * dt)); pk += float(g.sum() * dt)
        ec += float(5.0 * price @ (e * dt)); ek += float(e.sum() * dt)
        ch += float(cc.sum() * dt); di += float(dd.sum() * dt); days += 1
    smin_seen = min(smin_seen, float(S.min())); smax_seen = max(smax_seen, float(S.max()))
    s = float(S[-1])
return {"q2_planned_cost": pc, "q2_planned_kwh": pk, "q2_emergency_cost": ec,
        "q2_emergency_kwh": ek, "q2_total_cost": pc + ec,
        "q2_charge_kwh": ch, "q2_discharge_kwh": di,
        "q2_loss_kwh": (1 - eta) * ch + ((1 - eta) / eta) * di,
        "q2_days": float(days), "q2_soc_min": smin_seen, "q2_soc_max": smax_seen}

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--mode", required=True,
                    choices=["bootstrap", "grid", "stress", "ablation", "sensitivity"])
    ap.add_argument("--seed", type=int, default=0)
    ap.add_argument("--alpha", type=float, default=0.15); ap.add_argument("--beta", type=float, default=0.9)
    ap.add_argument("--W", type=int, default=7); ap.add_argument("--p", type=float, default=0.88)
    ap.add_argument("--first-day", type=int, default=31); ap.add_argument("--last-day", type=int, default=365)
    a = ap.parse_args()
    price, net, st = load(a.input)
    extra = {}
    if a.mode == "bootstrap":
        # 对"最近 W 天误差"这一估计量做自助重采样 -> 全年费用得到一个自举样本
        Nhat, _, _ = build_forecast(net, a.alpha, a.beta)
        err = net - Nhat
        m = run(price, net, st, a.alpha, a.beta, a.W, a.p, margin_source=err, boot_seed=a.seed,
                first_day=a.first_day, last_day=a.last_day)
        extra["seed"] = a.seed
    metrics = m

```

```

elif a.mode == "grid":
    best = None; table = []
    for W in (3, 5, 7, 10, 14, 21, 28):
        for p in (0.60, 0.70, 0.80, 0.88, 0.95):
            r = run(price, net, st, a.alpha, a.beta, W, p, first_day=a.first_day, last_day=a.last_day)
            table.append({"W": W, "p": p, "total": r["q2_total_cost"],
                        "emergency": r["q2_emergency_cost"]})
            if best is None or r["q2_total_cost"] < best["q2_total_cost"]:
                best = {"W": W, "p": p, **r}
    extra["grid"] = table
    metrics = {"q2_planned_cost": best["q2_planned_cost"], "q2_emergency_cost": best["q2_emergency_cost"],
              "q2_total_cost": best["q2_total_cost"], "q2_planned_kwh": best["q2_planned_kwh"],
              "q2_emergency_kwh": best["q2_emergency_kwh"], "q2_loss_kwh": best["q2_loss_kwh"],
              "q2_best_W": float(best["W"]), "q2_best_p": float(best["p"]),
              "q2_grid_points": float(len(table))}
elif a.mode == "sensitivity":
    bases = run(price, net, st, a.alpha, a.beta, a.W, a.p, first_day=a.first_day, last_day=a.last_day)
    table = []
    for al in (0.10, 0.15, 0.25, 0.40):
        for be in (0.6, 0.7, 0.8, 0.9):
            r = run(price, net, st, al, be, a.W, a.p, first_day=a.first_day, last_day=a.last_day)
            table.append({"alpha": al, "beta": be, "total": r["q2_total_cost"]})
    extra["sensitivity"] = table
    metrics = dict(bases); metrics["q2_sensitivity_points"] = float(len(table))
elif a.mode == "stress":
    table = []
    r_tight = run(price, net, st, a.alpha, a.beta, a.W, a.p, smin=3000.0, smax=9000.0,
                  first_day=a.first_day, last_day=a.last_day)
    table.append({"test": "soc_tight_3000_9000", **r_tight})
    r_eta = run(price, net, st, a.alpha, a.beta, a.W, a.p, eta=0.85,
                first_day=a.first_day, last_day=a.last_day)
    table.append({"test": "eta_0.85", **r_eta})
    r_pw = run(price, net, st, a.alpha, a.beta, a.W, a.p, smin=1200.0, smax=6000.0,
               first_day=a.first_day, last_day=a.last_day)
    table.append({"test": "capacity_half_1200_6000", **r_pw})
    extra["stress"] = table
    metrics = {"q2_total_cost": r_tight["q2_total_cost"],
              "q2_planned_cost": r_tight["q2_planned_cost"],
              "q2_emergency_cost": r_tight["q2_emergency_cost"],
              "q2_loss_kwh": r_tight["q2_loss_kwh"],
              "q2_soc_min": r_tight["q2_soc_min"], "q2_soc_max": r_tight["q2_soc_max"],
              "q2_total_cost_eta085": r_eta["q2_total_cost"],
              "q2_emergency_cost_eta085": r_eta["q2_emergency_cost"],
              "q2_total_cost_caphalf": r_pw["q2_total_cost"],
              "q2_emergency_cost_caphalf": r_pw["q2_emergency_cost"],
              "q2_stress_points": float(len(table))}
else:
    base = run(price, net, st, a.alpha, a.beta, a.W, a.p, first_day=a.first_day, last_day=a.last_day)

```

```

no_m = run(price, net, st, a.alpha, a.beta, a.W, a.p, no_margin=True,
           first_day=a.first_day, last_day=a.last_day)
pers = run(price, net, st, a.alpha, a.beta, a.W, a.p, forecast_mode="persistence",
           first_day=a.first_day, last_day=a.last_day)
extra["ablation"] = {"base": base, "no_margin": no_m, "persistence": pers}
metrics = dict(base)
metrics["q2_total_cost_no_margin"] = no_m["q2_total_cost"]
metrics["q2_total_cost_persistence"] = pers["q2_total_cost"]
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics, **extra}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
          ensure_ascii=False)
print("mode=%s seed=%s total=%0.2f" % (a.mode, a.seed, metrics.get("q2_total_cost", float("nan"))))

if __name__ == "__main__":
    main()

```

D.2.25 支撑材料/代码/02_问题二/q2_season_blocks.py

-*- coding: utf-8 -*-

"""问题二 四季独立建模(含"季末储电量回归 6000"约束)的多方案对照。

块划分(统计期 2/1-12/31): 冬 A 2月 | 春 3-5月 | 夏 6-8月 | 秋 9-11月 | 冬 B 12月
 reset=season : 每块开始与结束储电量都 = 6000(季末回归), 5 块完全独立
 reset=month : 每个自然月末回归 6000(11 块)
 reset=off : 日末自由(与现有交付同口径)

参数口径:

global : 全期统一 (7, 88%)
 prior : 理论先验 (7, 80%)
 oracle : 每块用它自己的数据选(含泄露, 上界)
 prev : 每块用上一块的数据选(无泄露)
 cum : 每块用它之前所有块的数据选(无泄露)

```

import os, sys, json, argparse, datetime
import numpy as np
from scipy.optimize import linprog
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal

```

```

GRID = [(W, p) for W in (5, 7, 10, 14) for p in (0.80, 0.88, 0.95)]
GRID_P = [(7, p) for p in (0.80, 0.88, 0.95)] # 只调分位(即"每季一个数")
D0 = datetime.date(2025, 1, 1)
BLOCKS = [("winter_a", 31, 59), ("spring", 59, 151), ("summer", 151, 243),
          ("autumn", 243, 334), ("winter_b", 334, 365)]

```

```

def day_lp(price, Nplan, s0, eta, dt, Pmax, smin, smax, s_end=None):
    """日内计划 LP; s_end 给定则强制日末储电量(季末回归 6000)."""

```

```

    T = len(price); n = 3 * T + (T + 1)

```

```

    og, oc, od, os_ = 0, T, 2 * T, 3 * T

```

```

    c = np.zeros(n); c[og:og + T] = price * dt

```

```

    A = np.zeros((T, n)); b = -Nplan

```

```

    for t in range(T):

```

```

        A[t, og + t] = -1.0; A[t, od + t] = -1.0; A[t, oc + t] = 1.0

```

```

    Ae = np.zeros((T, n))

```

```

    for t in range(T):

```

```

        Ae[t, os_ + t + 1] = 1.0; Ae[t, os_ + t] = -1.0; Ae[t, oc + t] = -eta * dt; Ae[t, od + t] = dt / eta

```

```

    last = (s_end, s_end) if s_end is not None else (smin, smax)

```

```

    bnd = [(0, None)] * T + [(0, Pmax)] * T + [(0, Pmax)] * T + [(s0, s0)] + [(smin, smax)] * (T - 1) + [last]

```

```

    r = linprog(c, A_ub=A, b_ub=b, A_eq=Ae, b_eq=np.zeros(T), bounds=bnd, method=" highs")

```

```

    if not r.success:

```

```

        raise RuntimeError(r.message)

x = r.x
return x[og:og + T], x[oc:oc + T], x[od:od + T], x[os_:os_ + T + 1]

class Sim:
    def __init__(self, inp):
        self.price = np.asarray(inp["price"], float)
        self.net = np.asarray(inp["load_actual"], float) - np.asarray(inp["pv_actual"], float)
        self.Nhat, self.err, _, _ = build_forecast(self.net, 0.15, 0.90)
        st = inp["storage"]; self.dt = float(inp["dt"]); self.eta = float(st["efficiency"])
        self.Pmax = float(st["power_max"]); self.smin = float(st["soc_min"])
        self.smax = float(st["soc_max"]); self.s0 = float(st["soc_initial"])

    def block(self, first, last, W, p, reset_end, s_start=None, acc_from=None):
        """模拟一个块; reset_end=True 时块末强制回到 6000; acc_from 指定计费起始日(默认=first)."""
        s = self.s0 if s_start is None else s_start
        acc_from = first if acc_from is None else acc_from
        pc = pk = ec = ek = ch = di = cu = 0.0
        smin_seen, smax_seen = 1e18, -1e18
        for d in range(first, last):
            Nplan = self.Nhat[d] + margin(self.err, d, W, p)
            pend = 6000.0 if (reset_end and d == last - 1) else None
            g, _, _, _ = day_lp(self.price, Nplan, s, self.eta, self.dt, self.Pmax, self.smin, self.smax, pend)
            e, cc, dd, S = execute_causal(g, self.net[d], s, self.eta, self.dt, self.Pmax, self.smin, self.smax)
            curt = np.maximum(0.0, (g - self.net[d]) - cc)      # 买多了又充不进电池的部分
            if d >= acc_from:
                smin_seen = min(smin_seen, float(S.min())); smax_seen = max(smax_seen, float(S.max()))
                pc += float(self.price @ (g * self.dt)); pk += float(g.sum() * self.dt)
                ec += float(5.0 * self.price @ (e * self.dt)); ek += float(e.sum() * self.dt)
                ch += float(cc.sum() * self.dt); di += float(dd.sum() * self.dt); cu += float(curt.sum() * self.dt)
            s = float(S[-1])
        return {"plan_cost": pc, "plan_kwh": pk, "em_cost": ec, "em_kwh": ek, "total": pc + ec,
            "charge": ch, "discharge": di, "curtail": cu, "soc_min": smin_seen, "soc_max": smax_seen,
            "soc_end": s}

    def blocks_for(self, reset):
        if reset == "season":
            return [(n, f, l) for n, f, l in BLOCKS]
        if reset == "month":
            out = []; cur = 31
            for m in range(2, 13):
                nxt = (datetime.date(2025, m % 12 + 1, 1) - D0).days if m < 12 else 365
                out.append(("m%02d" % m, cur, nxt)); cur = nxt
            return out
        return [("year", 1, 365)]

```

```

def warm_soc(self):
    """1 月预热(日末自由) -> 得到 2/1 0:00 的储电量, 与现有交付口径一致."""
    return self.block(1, 31, 7, 0.88, False, None, acc_from=31)["soc_end"]

def best_on(self, first, last, reset_end, s_start=None, grid=None):
    best = None
    for W, p in (grid or GRID):
        r = self.block(first, last, W, p, reset_end, s_start)
        if best is None or r["total"] < best[2]:
            best = (W, p, r["total"])
    return best

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--mode", default="all")
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    sim = Sim(inp)
    metrics = {}; report = {}

def full_year(reset, table):
    """按表跑全年; table: dict block_key->(W,p); 以 'default' 兜底."""
    tot = {"plan_cost": 0.0, "em_cost": 0.0, "plan_kwh": 0.0, "em_kwh": 0.0,
           "charge": 0.0, "discharge": 0.0, "curtail": 0.0, "soc_min": 1e18, "soc_max": -1e18}
    per = {}
    blocks = sim.blocks_for(reset)
    s_cur = None if reset == "off" else sim.warm_soc() # 季末回归口径下, 首块自 2/1 的预热 SOC 起
    for key, f, l in blocks:
        W, p = (table.get(key) or table.get("default"))
        reset_end = (reset != "off")
        r = sim.block(f, l, W, p, reset_end, s_cur, acc_from=(31 if reset == "off" else None))
        s_cur = r["soc_end"]
        per[key] = {"W": W, "p": p, **{k: v for k, v in r.items()}}
        for k in ("plan_cost", "em_cost", "plan_kwh", "em_kwh", "charge", "discharge", "curtail"):
            tot[k] += r[k]
        tot["soc_min"] = min(tot["soc_min"], r["soc_min"]); tot["soc_max"] = max(tot["soc_max"], r["soc_max"])
    tot["total"] = tot["plan_cost"] + tot["em_cost"]
    return tot, per

def oracle_table(reset):
    tb = {}; s_cur = None if reset == "off" else sim.warm_soc()
    for key, f, l in sim.blocks_for(reset):
        W, p, c = sim.best_on(f, l, reset != "off", s_cur)
        tb[key] = (W, p)
        s_cur = sim.block(f, l, W, p, reset != "off", s_cur)["soc_end"]

```

```
return tb
```

```
def walk_table(reset, how):
```

```
tb = {}; seen = []; s_cur = None if reset == "off" else sim.warm_soc()
for key, f, l in sim.blocks_for(reset):
    if (how == "prev" and seen) or (how == "cum" and seen):
        gf, gl = seen[-1] if how == "prev" else (seen[0][0], seen[-1][1])
        W, p, c = sim.best_on(gf, gl, reset != "off", None if gf > 31 else s_cur)
        tb[key] = (W, p)
    else:
        tb[key] = (7, 0.80)
s_cur = sim.block(f, l, tb[key][0], tb[key][1], reset != "off", s_cur)["soc_end"]
seen.append((f, l))
return tb
```

```
variants = []
```

```
for reset in ("off", "season", "month"):
    variants.append((reset, "global", {"default": (7, 0.88)}))
    variants.append((reset, "prior", {"default": (7, 0.80)}))
for reset in ("season", "month"):
    variants.append((reset, "oracle", oracle_table(reset)))
    variants.append((reset, "prev", walk_table(reset, "prev")))
    variants.append((reset, "cum", walk_table(reset, "cum")))
```

```
def oracle_p(reset):
```

```
tb = {}; s_cur = None if reset == "off" else sim.warm_soc()
for key, f_, l in sim.blocks_for(reset):
    W, p, c = sim.best_on(f_, l, reset != "off", s_cur, grid=GRID_P)
    tb[key] = (W, p)
s_cur = sim.block(f_, l, W, p, reset != "off", s_cur)["soc_end"]
return tb
```

```
def walk_p(reset, how):
```

```
tb = {}; seen = []; s_cur = None if reset == "off" else sim.warm_soc()
for key, f_, l in sim.blocks_for(reset):
    if seen:
        gf, gl = seen[-1] if how == "prev" else (seen[0][0], seen[-1][1])
        W, p, c = sim.best_on(gf, gl, reset != "off", None, grid=GRID_P)
        tb[key] = (W, p)
    else:
        tb[key] = (7, 0.80)
s_cur = sim.block(f_, l, tb[key][0], tb[key][1], reset != "off", s_cur)["soc_end"]
seen.append((f_, l))
return tb
```

```

for reset in ("season", "month"):
    variants.append((reset, "oracle_p", oracle_p(reset)))
    variants.append((reset, "prev_p", walk_p(reset, "prev")))
    variants.append((reset, "cum_p", walk_p(reset, "cum")))
# 固定组合扫描(在"季末回归 6000"口径下, 看哪个单一组合最好
for W, p in GRID:
    variants.append(("season", "fixed_w%d_p%02d" % (W, round(p * 100)), {"default": (W, p)}))

for reset, params, table in variants:
    tot, per = full_year(reset, table)
    tag = "%s_%s" % (reset, params)
    metrics["q2b_%s_total" % tag] = tot["total"]
    metrics["q2b_%s_plan" % tag] = tot["plan_cost"]
    metrics["q2b_%s_em" % tag] = tot["em_cost"]
    metrics["q2b_%s_em_kwh" % tag] = tot["em_kwh"]
    metrics["q2b_%s_loss" % tag] = (1 - sim.eta) * tot["charge"] + ((1 - sim.eta) / sim.eta) * tot["discharge"]
    metrics["q2b_%s_curtail" % tag] = tot["curtail"]
    metrics["q2b_%s_soc_min" % tag] = tot["soc_min"]
    metrics["q2b_%s_soc_max" % tag] = tot["soc_max"]
    report[tag] = {"params": params, "table": {k: list(v) for k, v in table.items()}},
        "total": tot["total"], "per_block": per,
        "block_end_soc": {k: per[k]["soc_end"] for k in per} }
    print("%-16s 总费用 %12.2f 计划 %12.2f 紧急 %10.2f 弃电 %10.1f" %
        (tag, tot["total"], tot["plan_cost"], tot["em_cost"], tot["curtail"]))

os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics, "report": report}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
    ensure_ascii=False, indent=1)
print("done")

if __name__ == "__main__":
    main()

```

D.2.26 支撑材料/代码/02_问题二/q2_season_chain.py

-*- coding: utf-8 -*-

"""问题二 四季独立建模(日末自由, 不做季末 6000 回归)的多方案对照。

块划分: 冬A 2月 | 春 3-5月 | 夏 6-8月 | 秋 9-11月 | 冬B 12月 (另设按月 11 块的对照)
季与季之间靠实际储电量自然衔接(不重置), 每日 LP 的日末储电量自由。

参数口径:

global 全期统一(7,88%) | prior 理论先验(7,80%)
oracle 每块用它自己那段数据选(含泄露, 上界)
prev 每块用上一块的数据选(无泄露) cum 每块用之前所有数据选(无泄露)
*_p 只调分位、窗口固定 7 天(即"每季一个数")

```

import os, sys, json, argparse, datetime
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_season_blocks import Sim, GRID, GRID_P, BLOCKS, D0

```

```

def month_blocks():
    out = []; cur = 31
    for m in range(2, 13):

```

```

        nxt = (datetime.date(2025, m % 12 + 1, 1) - D0).days if m < 12 else 365
        out.append(("m%02d" % m, cur, nxt)); cur = nxt
    return out

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    sim = Sim(inp)
    metrics = {}; report = {}

def run_chain(blocks, table):
    s_cur = sim.warm_soc()
    tot = {"plan_cost": 0.0, "em_cost": 0.0, "plan_kwh": 0.0, "em_kwh": 0.0,
          "charge": 0.0, "discharge": 0.0, "curtail": 0.0, "soc_min": 1e18, "soc_max": -1e18}
    per = {}
    for key, f, l in blocks:
        W, p = (table.get(key) or table.get("default"))
        r = sim.block(f, l, W, p, False, s_cur)
        per[key] = {"W": W, "p": p, **{k: v for k, v in r.items()}}
        for k in ("plan_cost", "em_cost", "plan_kwh", "em_kwh", "charge", "discharge", "curtail"):
            tot[k] += r[k]
        tot["soc_min"] = min(tot["soc_min"], r["soc_min"]); tot["soc_max"] = max(tot["soc_max"], r["soc_max"])
        s_cur = r["soc_end"]
    tot["total"] = tot["plan_cost"] + tot["em_cost"]
    return tot, per

def table_walk(blocks, how, grid=None):
    tb = {}; hist = []; s_cur = sim.warm_soc()
    for key, f, l in blocks:
        if how == "oracle":
            W, p, c = sim.best_on(f, l, False, s_cur, grid=grid)
        elif how == "prev" and hist:
            _, f2, l2, s2 = hist[-1]
            W, p, c = sim.best_on(f2, l2, False, s2, grid=grid)
        elif how == "cum" and hist:
            _, f0, _, s0 = hist[0]
            W, p, c = sim.best_on(f0, f, False, s0, grid=grid) # 只用本块之前的数据(不含本块)
        else:
            W, p = (7, 0.80)
        tb[key] = (W, p)
        s_start = s_cur
        r = sim.block(f, l, W, p, False, s_cur)
        s_cur = r["soc_end"]
        hist.append((key, f, l, s_start))
    return tb

```

```

seas = list(BLOCKS)
mons = month_blocks()
variants = []
for tag, blocks in (("seas", seas), ("mon", mons)):
    variants.append((tag, "global", blocks, {"default": (7, 0.88)}))
    variants.append((tag, "prior", blocks, {"default": (7, 0.80)}))
    variants.append((tag, "oracle", blocks, table_walk(blocks, "oracle")))
    if tag == "seas":
        variants.append((tag, "prev", blocks, table_walk(blocks, "prev")))
        variants.append((tag, "cum", blocks, table_walk(blocks, "cum")))
        variants.append((tag, "oracle_p", blocks, table_walk(blocks, "oracle", GRID_P)))
        variants.append((tag, "prev_p", blocks, table_walk(blocks, "prev", GRID_P)))
    else:
        variants.append((tag, "prev", blocks, table_walk(blocks, "prev")))
# 全年单块基线(核对)
variants.append(("year", "global", [("year", 1, 365)], {"default": (7, 0.88)}))

for tag, params, blocks, table in variants:
    if tag == "year":
        s_cur = None
        r = sim.block(1, 365, 7, 0.88, False, None, acc_from=31)
        tot = {"total": r["total"], "plan_cost": r["plan_cost"], "em_cost": r["em_cost"],
              "em_kwh": r["em_kwh"], "curtail": r["curtail"],
              "loss": (1 - sim.eta) * r["charge"] + ((1 - sim.eta) / sim.eta) * r["discharge"],
              "soc_min": r["soc_min"], "soc_max": r["soc_max"]}
        per = {"year": {"W": 7, "p": 0.88, **r}}
    else:
        tot, per = run_chain(blocks, table)
        tot["loss"] = (1 - sim.eta) * tot["charge"] + ((1 - sim.eta) / sim.eta) * tot["discharge"]
    key = "%s_%s" % (tag, params)
    metrics["q2c_%s_total" % key] = tot["total"]
    metrics["q2c_%s_plan" % key] = tot["plan_cost"]
    metrics["q2c_%s_em" % key] = tot["em_cost"]
    metrics["q2c_%s_em_kwh" % key] = tot["em_kwh"]
    metrics["q2c_%s_curtail" % key] = tot["curtail"]
    metrics["q2c_%s_loss" % key] = tot["loss"]
    metrics["q2c_%s_soc_min" % key] = tot["soc_min"]
    metrics["q2c_%s_soc_max" % key] = tot["soc_max"]
    report[key] = {"table": {k: list(v) for k, v in table.items()}, "total": tot["total"], "per_block": per}
    print("%-16s 总费用 %12.2f 计划 %12.2f 紧急 %10.2f 弃电 %10.0f" %
          (key, tot["total"], tot["plan_cost"], tot["em_cost"], tot["curtail"]))

os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics, "report": report}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
          ensure_ascii=False, indent=1)

```

```

print("done")

if __name__ == "__main__":
    main()

```

D.2.27 支撑材料/代码/02_问题二/q2_seasonal.py

-*- coding: utf-8 -*-

```

import os, json, argparse, datetime
import numpy as np
from scipy.optimize import linprog

def wd(d): return (datetime.date(2025,1,1)+datetime.timedelta(days=d)).weekday()
def cat(d): return 0 if wd(d) in (4,5) else 1 # Fri(4)/Sat(5)=0, others=1

def build_baseline(arr):
    D = arr.shape[0]; F = np.zeros_like(arr)
    sums = [np.zeros(arr.shape[1]), np.zeros(arr.shape[1])]; cnt = [0,0]
    for d in range(D):
        w = cat(d)
        F[d] = sums[w]/cnt[w] if cnt[w] > 0 else arr[d]
        sums[w] = sums[w] + arr[d]; cnt[w] += 1
    return F

def build_ema(arr, alpha):
    D = arr.shape[0]; F = np.zeros_like(arr); F[0] = arr[0]
    for d in range(1, D): F[d] = alpha*arr[d-1] + (1-alpha)*F[d-1]
    return F

def solve_day_lp(price, net_margin, soc0, eta, dt, Pmax, soc_min, soc_max):
    T = len(price); n = 3*T + (T+1)
    c = np.zeros(n); c[0:T] = price*dt
    bounds = [(0,None)]*T + [(0,Pmax)]*T + [(0,Pmax)]*T + [(soc0,soc0)]
    for _ in range(1,T): bounds.append((soc_min,soc_max))
    bounds.append((soc_min,soc_max))
    A_ub = np.zeros((T,n)); b_ub = -net_margin
    for t in range(T): A_ub[t,t]=-1.0; A_ub[t,T+t]=1.0; A_ub[t,2*T+t]=-1.0
    A_eq = np.zeros((T,n)); b_eq = np.zeros(T)
    for t in range(T):
        A_eq[t,3*T+t+1]=1.0; A_eq[t,3*T+t]=-1.0; A_eq[t,T+t]=-eta*dt; A_eq[t,2*T+t]=dt/eta
    res = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs")
    if not res.success: raise RuntimeError(res.message)
    x = res.x
    return x[0:T], x[T:2*T], x[2*T:3*T], x[3*T:3*T+T+1], float(np.dot(price, x[0:T])*dt)

def main():
    ap = argparse.ArgumentParser(); ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--pct", type=float, default=88.0, help="冗余分位 (%)")
    ap.add_argument("--window", type=int, default=7, help="误差统计窗口 (天)")
    ap.add_argument("--alpha", type=float, default=0.15, help="指数加权衰减系数")
    ap.add_argument("--beta", type=float, default=0.8, help="星期基线权重")
    a = ap.parse_args()

```

```

inp = json.load(open(a.input, encoding="utf-8"))
price = np.asarray(inp["price"], float); load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]
Pmax = st["power_max"]; soc_min = st["soc_min"]; soc_max = st["soc_max"]
D, T = load.shape; net = load - pv
alpha = a.alpha; beta = a.beta; pct = a.pct; window_err = a.window
load_bl = build_baseline(load); pv_bl = build_baseline(pv)
load_em = build_ema(load, alpha); pv_em = build_ema(pv, alpha)
lf_all = (1-beta)*load_em + beta*load_bl
pf_all = (1-beta)*pv_em + beta*pv_bl
nf_all = lf_all - pf_all
soc = st["soc_initial"]; tp=0.0; te=0.0; plans=[]
for d in range(D):
    if d >= 2:
        lo = max(1, d-window_err)
        errs = net[lo:d] - nf_all[lo:d]
        margin = np.maximum(np.percentile(errs, pct, axis=0), 0.0)
    else:
        margin = np.zeros(T)
    p, c, dd, s, pcost = solve_day_lp(price, nf_all[d]+margin, soc, eta, dt, Pmax, soc_min, soc_max)
    E = np.maximum(0.0, net[d] + c - p - dd)
    if d >= 31:
        tp += pcost; te += 5.0*np.dot(price, E)*dt
        plans.append(dict(day=d, purchase=p.tolist(), charge=c.tolist(), discharge=dd.tolist(), soc=s.tolist(),
emergency=E.tolist(), margin=margin.tolist()))
    soc = float(s[-1])
metrics = {"q2_total_cost": tp+te, "q2_planned_cost": tp, "q2_emergency_cost": te}
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"))
json.dump({"plans": plans, "price": price.tolist()}, open(os.path.join(a.out, "solution.json"), "w", encoding="utf-8"))
print(f"q2-seasonal: total={tp+te:.2f} planned={tp:.2f} emergency={te:.2f}")
if __name__ == "__main__": main()

```

D.2.28 支撑材料/代码/02_问题二/q2_seasonal_model.py

-*- coding: utf-8 -*-

"""问题二 季节独立建模：每个季节一组 (W, p)，逐季标定，严格无泄露。

季节(气象)：春 3-5 月，夏 6-8 月，秋 9-11 月，冬 12/1/2 月。

标定口径：

global : 全年单组参数，由全年窗口选出(基线对照，与交付同口径)
oracle : 每季用它自己那 3 个月的数据选参数(含泄露，只作上界参考)
walk : 每季的 (W,p) 只用“该季开始之前”的数据选(无泄露；等价于第二年用上一年数据标定)
冬跨年：2 月这一段没有更早的历史 -> 理论先验 (W=7, p=80%)；12 月这一段用截至 11/30 的历史标定。
三个口径使用同一个候选集合(见 GRID)，保证可比。

```

import os, sys, json, argparse, datetime
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal
from q2_online import day_lp

```

```

GRID = [(W, p) for W in (5, 7, 10, 14) for p in (0.80, 0.88, 0.95)]
PRIOR = (7, 0.80)
D0 = datetime.date(2025, 1, 1)

```

```

def season_of(day):

```

```

m = (D0 + datetime.timedelta(days=int(day))).month
return "spring" if m in (3, 4, 5) else "summer" if m in (6, 7, 8) else \
    "autumn" if m in (9, 10, 11) else "winter"

def first_day_of_month(month):
    return (datetime.date(2025, month, 1) - D0).days

class Sim:
    def __init__(self, inp):
        self.price = np.asarray(inp["price"], float)
        L = np.asarray(inp["load_actual"], float); PV = np.asarray(inp["pv_actual"], float)
        self.net = L - PV
        self.Nhat, self.err, _, _ = build_forecast(self.net, 0.15, 0.90)
        st = inp["storage"]; self.dt = float(inp["dt"]); self.eta = float(st["efficiency"])
        self.Pmax = float(st["power_max"]); self.smin = float(st["soc_min"])
        self.smax = float(st["soc_max"]); self.s0 = float(st["soc_initial"])

    def run(self, pick, first_day, last_day, detail=False):
        s = self.s0; pc = pk = ec = ek = ch = di = 0.0
        smin_seen, smax_seen = 1e18, -1e18
        seas = {}
        for d in range(1, last_day):
            W, p = pick(d)
            Nplan = self.Nhat[d] + margin(self.err, d, W, p)
            g, _, _, _ = day_lp(self.price, Nplan, s, self.eta, self.dt, self.Pmax, self.smin, self.smax)
            e, cc, dd, S = execute_causal(g, self.net[d], s, self.eta, self.dt, self.Pmax, self.smin, self.smax)
            smin_seen = min(smin_seen, float(S.min())); smax_seen = max(smax_seen, float(S.max()))
            if d >= first_day:
                c = float(self.price @ (g * self.dt)); k = float(g.sum() * self.dt)
                ez = float(5.0 * self.price @ (e * self.dt)); ez_k = float(e.sum() * self.dt)
                pc += c; pk += k; ec += ez; ek += ez_k
                ch += float(cc.sum() * self.dt); di += float(dd.sum() * self.dt)
                if detail:
                    sn = season_of(d); seas[sn] = seas.get(sn, 0.0) + c + ez
            s = float(S[-1])
        out = {"planned_cost": pc, "planned_kwh": pk, "emergency_cost": ec, "emergency_kwh": ek,
            "total": pc + ec, "loss_kwh": (1 - self.eta) * ch + ((1 - self.eta) / self.eta) * di,
            "soc_min": smin_seen, "soc_max": smax_seen}
        if detail:
            out["season_cost"] = seas
        return out

    def best_on(self, first_day, last_day):
        best = None
        for W, p in GRID:
            r = self.run(lambda d, W=W, p=p: (W, p), first_day, last_day)
            if best is None or r["total"] < best[2]:

```

```

        best = (W, p, r["total"])
    return best

def table_oracle(sim):
    tb = {}
    windows = {"spring": (first_day_of_month(3), first_day_of_month(6)),
               "summer": (first_day_of_month(6), first_day_of_month(9)),
               "autumn": (first_day_of_month(9), first_day_of_month(12)),
               "winter": (first_day_of_month(1), 365)}
    for seas, (f, l) in windows.items():
        W, p, c = sim.best_on(f, l)
        tb[seas] = (W, p, c, (int(f), int(l)))
    return tb

def table_walk(sim):
    tb = {}
    windows = {"spring": (first_day_of_month(3), first_day_of_month(6)),
               "summer": (first_day_of_month(6), first_day_of_month(9)),
               "autumn": (first_day_of_month(9), first_day_of_month(12)),
               "winter": (first_day_of_month(12), 365)}
    for seas, (f, l) in windows.items():
        if f - 1 >= 30:
            W, p, c = sim.best_on(1, f)
            tb[seas] = (W, p, c, (1, int(f)))
        else:
            tb[seas] = (PRIOR[0], PRIOR[1], float("nan"), (1, int(f)))
    tb["winter_feb"] = (PRIOR[0], PRIOR[1], float("nan"), (1, 31))
    return tb

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--mode", required=True, choices=["global", "oracle", "walk", "all"])
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    sim = Sim(inp)
    metrics = {}; info = {}

    Wg, pg, cg = sim.best_on(31, 365)
    rb = sim.run(lambda d: (Wg, pg), 31, 365, detail=True)
    metrics.update({"q2g_W": float(Wg), "q2g_p": float(pg), "q2g_total": rb["total"],
                  "q2g_planned": rb["planned_cost"], "q2g_emergency": rb["emergency_cost"],
                  "q2g_emergency_kwh": rb["emergency_kwh"], "q2g_loss_kwh": rb["loss_kwh"]})
    info["global"] = {"table": {"all": [Wg, pg]}, "cost": rb["total"]}
    print("[global] 全期最优 (W=%d, p=%0.0f%%) = %0.2f" % (Wg, pg * 100, rb["total"]))

    if a.mode in ("oracle", "all"):

```

```

tb = table_oracle(sim)
info["oracle_table"] = {k: [v[0], v[1]] for k, v in tb.items()}
r = sim.run(lambda d: (tb[season_of(d)][0], tb[season_of(d)][1]), 31, 365, detail=True)
metrics.update({"q2o_total": r["total"], "q2o_planned": r["planned_cost"],
               "q2o_emergency": r["emergency_cost"], "q2o_emergency_kwh": r["emergency_kwh"],
               "q2o_loss_kwh": r["loss_kwh"], "q2o_soc_min": r["soc_min"], "q2o_soc_max": r["soc_max"]})
for k, v in r["season_cost"].items():
    metrics["q2o_cost_" + k] = v
for k, v in tb.items():
    metrics["q2o_W_" + k] = float(v[0]); metrics["q2o_p_" + k] = float(v[1])
info["oracle"] = {"table": info["oracle_table"], "cost": r["total"]}
print("[oracle] %s -> %.2f ( 计划 %.2f + 紧急 %.2f) % (info["oracle_table"], r["total"], r["planned_cost"],
r["emergency_cost"])))

```

```

if a.mode in ("walk", "all"):
    tb = table_walk(sim)
    info["walk_table"] = {k: [v[0], v[1]] for k, v in tb.items()}
    def pick(d):
        sn = season_of(d)
        if sn == "winter":
            m = (D0 + datetime.timedelta(days=int(d))).month
            sn = "winter_feb" if m == 2 else "winter"
        return (tb[sn][0], tb[sn][1])
    r = sim.run(pick, 31, 365, detail=True)
    metrics.update({"q2w_total": r["total"], "q2w_planned": r["planned_cost"],
                  "q2w_emergency": r["emergency_cost"], "q2w_emergency_kwh": r["emergency_kwh"],
                  "q2w_loss_kwh": r["loss_kwh"], "q2w_soc_min": r["soc_min"], "q2w_soc_max": r["soc_max"]})
    for k, v in r["season_cost"].items():
        metrics["q2w_cost_" + k] = v
    for k, v in tb.items():
        metrics["q2w_W_" + k] = float(v[0]); metrics["q2w_p_" + k] = float(v[1])
    info["walk"] = {"table": info["walk_table"], "cost": r["total"]}
    print("[walk] %s -> %.2f ( 计划 %.2f + 紧急 %.2f) % (info["walk_table"], r["total"], r["planned_cost"],
r["emergency_cost"])))

```

```

os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics, "info": info}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
          ensure_ascii=False, indent=1)
print("done mode=%s" % a.mode)

```

```

if __name__ == "__main__":
    main()

```

D.2.29 支撑材料/代码/02_问题二/q2_solve_and_gen.py

```
import os
```

-*- coding: utf-8 -*-

```

"""重算问题二(完美信息)并生成 result2.xlsx; 同时按缓存重生成 result3.xlsx."""
import numpy as np, json, openpyxl, datetime

```

```

from scipy.optimize import linprog
from pathlib import Path
proj=Path(os.environ.get("CUMCM_ROOT", "."); root=proj/"paper_output_pro"
dt=1/6; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; T=144; D=365
def load_att1():
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 1.xlsx",read_only=True,data_only=True)
    rows=list(wb["Sheet1"].iter_rows(values_only=True))[1:]
    return np.array([float(r[1]) for r in rows]), np.array([float(r[2]) for r in rows]), np.array([float(r[3]) for r in rows])
def load_att2():
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv
price1,load1,pv1=load_att1(); load2,pv2=load_att2()

def solve_day(pr,net,soc_start):
    n=3*T+(T+1); A_a=0;A_c=T;A_d=2*T;A_s=3*T
    c=np.zeros(n); c[0:T]=pr*dt
    bnds=[(0,None)]*T+[(0,Pmax)]*T+[(0,Pmax)]*T+[(soc_start,soc_start)]+[(soc_min,soc_max)]*T
    A_ub=np.zeros((T,n)); b_ub=-net
    for t in range(T):
        A_ub[t,A_a+t]=-1.0;A_ub[t,A_d+t]=-1.0;A_ub[t,A_c+t]=1.0
    A_eq=np.zeros((T,n)); b_eq=np.zeros(T)
    for t in range(T):
        A_eq[t,A_s+t+1]=1.0;A_eq[t,A_s+t]=-1.0;A_eq[t,A_c+t]=-eta*dt;A_eq[t,A_d+t]=dt/eta
    r=linprog(c,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bnds,method="highs")
    if not r.success: raise RuntimeError(r.message)
    x=r.x; return x[A_a:A_a+T],x[A_c:A_c+T],x[A_d:A_d+T],x[A_s:A_s+T+1]

soc=soc0; q2=[]
for d in range(D):
    p,c,dd,s=solve_day(price1, load2[d]-pv2[d], soc)
    LD=load2[d]; PV=pv2[d]
    c_rt=np.zeros(T); d_rt=np.zeros(T); E=np.zeros(T); s_rt=soc
    for t in range(T):
        net=LD[t]-PV[t]
        if net>p[t]:
            dv=min(Pmax,(s_rt-soc_min)*eta/dt,net-p[t]); E[t]=net-p[t]-dv; s_rt=dv*dt/eta; d_rt[t]=dv
        else:
            cv=min(Pmax,(soc_max-s_rt)/(eta*dt),p[t]-net); s_rt+=eta*cv*dt; c_rt[t]=cv
    q2.append(dict(day=d,p=p.tolist(),a=p.tolist(),c=c_rt.tolist(),d=d_rt.tolist(),E=E.tolist(),final_soc=float(s[-1])))
    soc=float(s[-1])
res=q2[31:]
tot=sum(float(np.dot(price1,x["p"])*dt) for x in res)
print(f"[问题二 重算] 334 天购电费={tot:.2f} 元, 购电量={sum(sum(x['p']) for x in res)*dt:.2f} kWh, 紧急={sum(sum(x['E']) for x in res)*dt:.4f} kWh")
json.dump(dict(q2=q2, price1=price1.tolist(), total=tot), open(root/"data_cleaned/q2_regen_cache.json","w"))

periods=["0:00-4:00","4:00-8:00","8:00-12:00","12:00-16:00","16:00-20:00","20:00-24:00"]
def hhm(m): return f"{m//60:02d}:{m%60:02d}"
def soc_starts(daily): return [6000.0]+[daily[d-1]["final_soc"] for d in range(1,D)]
def fill_plan(ws,days,key,price):
    for i,day in enumerate(days):
        r=i+2; v=day[key]
        for t in range(T): ws.cell(row=r,column=2+t,value=round(v[t]*dt,4))
        ws.cell(row=r,column=2+T,value=round(sum(v[t]*dt for t in range(T)),4))

```

```

ws.cell(row=r,column=3+T,value=round(sum(price[t]*v[t]*dt for t in range(T)),2))
def fill_cd(ws,days,socs):
    ws.delete_rows(2,ws.max_row); r=2
    for i,day in enumerate(days):
        c=day["c"]; d=day["d"]; date=datetime.datetime(2025,2,1)+datetime.timedelta(days=i)
        for pi,lbl in enumerate(periods):
            lo,hi=pi*24,(pi+1)*24
            ws.cell(row=r,column=1,value=date if pi==0 else None)
            ws.cell(row=r,column=2,value=lbl)
            ws.cell(row=r,column=3,value=round(sum(c[t]*dt for t in range(lo,hi)),4))
            ws.cell(row=r,column=4,value=round(sum(d[t]*dt for t in range(lo,hi)),4))
            if pi==0: ws.cell(row=r,column=5,value=datetime.time(0,0)); ws.cell(row=r,column=6,value=round(socs[31+i],4))
            elif pi==1: ws.cell(row=r,column=5,value="24:00"); ws.cell(row=r,column=6,value=round(day["final_soc"],4))
            r+=1
def fill_em(ws,days):
    ws.delete_rows(2,ws.max_row); r=2
    for i,day in enumerate(days):
        E=day["E"]; date=datetime.datetime(2025,2,1)+datetime.timedelta(days=i); runs=[]; j=0
        while j<T:
            if E[j]>1e-6:
                k=j
                while k<T and E[k]>1e-6: k+=1
                runs.append((hhmm(j*10),hhmm(k*10),round(sum(E[t]*dt for t in range(j,k),4)))); j=k
            else: j+=1
        if runs:
            for ri,(a,b,en) in enumerate(runs):
                ws.cell(row=r,column=1,value=date if ri==0 else None); ws.cell(row=r,column=2,value=f"{a}-{b}");
            ws.cell(row=r,column=3,value=en); r+=1
        else:
            ws.cell(row=r,column=1,value=date); ws.cell(row=r,column=3,value=0.0); r+=1

```

--- result2.xlsx ---

```

wb=openpyxl.load_workbook(proj/"problem_files/附件/附件5/result2.xlsx")
d2=q2[31:]
fill_plan(wb["计划购电量"], d2, "p", price1)
fill_cd(wb["充放电量"], d2, soc_starts(q2)); fill_em(wb["紧急购电量"], d2)
wb.save(root/"result2.xlsx"); print("SAVED result2.xlsx")

```

--- result3.xlsx (从既有缓存重生) ---

```

data=json.loads((root/"data_cleaned/q3_result_cache.json").read_text(encoding="utf-8"))
price3=np.array(data["price"]); daily=data["daily"]; d3=daily[31:]
wb=openpyxl.load_workbook(proj/"problem_files/附件/附件5/result3.xlsx")
fill_plan(wb["计划购电量"], d3, "p", price3)
fill_plan(wb["调整购电量"], d3, "a", price3)
fill_cd(wb["充放电量"], d3, soc_starts(daily)); fill_em(wb["紧急购电量"], d3)
wb.save(root/"result3.xlsx"); print("SAVED result3.xlsx")
print(f"[ 问题三 缓存 ] 计划购电费={sum(np.dot(price3,x['p'])*dt for x in d3):.2f} 计划购电量={sum(sum(x['p']) for x in d3)*dt:.2f}")

```

D.2.30 支撑材料/代码/02_问题二/q2_stages.py

-*- coding: utf-8 -*-

"""问题二"从数据泄露到无泄露"的推进过程：各阶段方案的横向对照。

```

阶段:
A 每日储电量回归 6000 + 当天数据全部已知(无信息边界) -> 最初做法
B 每日储电量回归 6000 + 只用 d-1 及更早(无泄露预测) -> 打破泄露但保留日约束
C 日末储电量自由 + 当天数据全部已知 -> 理论下界
D 日末储电量自由 + 只用 d-1 及更早(= 现交付主结果)
"""
import os, sys, json, argparse
import numpy as np
from scipy.optimize import linprog
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal

DT = 1.0 / 6.0

def day_lp(price, Nplan, s0, eta, dt, Pmax, smin, smax, s_end):
    T = len(price); n = 3 * T + (T + 1)
    og, oc, od, os_ = 0, T, 2 * T, 3 * T
    c = np.zeros(n); c[og:og + T] = price * dt
    A = np.zeros((T, n)); b = -Nplan
    for t in range(T):
        A[t, og + t] = -1.0; A[t, od + t] = -1.0; A[t, oc + t] = 1.0
    Ae = np.zeros((T, n))
    for t in range(T):
        Ae[t, os_ + t + 1] = 1.0; Ae[t, os_ + t] = -1.0; Ae[t, oc + t] = -eta * dt; Ae[t, od + t] = dt / eta
    tail = (s_end, s_end) if s_end is not None else (smin, smax)
    bnd = [(0, None)] * T + [(0, Pmax)] * T + [(0, Pmax)] * T + [(s0, s0)] + [(smin, smax)] * (T - 1) + [tail]
    r = linprog(c, A_ub=A, b_ub=b, A_eq=Ae, b_eq=np.zeros(T), bounds=bnd, method="highs")
    if not r.success:
        raise RuntimeError(r.message)
    x = r.x
    return x[og:og + T]

def run(inp, perfect, daily6000, W=7, p=0.88):
    price = np.asarray(inp["price"], float)
    net = np.asarray(inp["load_actual"], float) - np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = float(inp["dt"]); eta = float(st["efficiency"])
    Pmax = float(st["power_max"]); smin = float(st["soc_min"]); smax = float(st["soc_max"])
    Nhat, err, _, _ = build_forecast(net, 0.15, 0.90)
    D = net.shape[0]; s = float(st["soc_initial"])
    pc = ec = pk = ek = cu = 0.0; smin_seen, smax_seen = 1e18, -1e18
    for d in range(1, D):
        if perfect:
            Nplan = net[d].copy() # 当天实际值直接进计划(泄露)
        else:
            Nplan = Nhat[d] + margin(err, d, W, p) # 只用 d-1 及更早
        g = day_lp(price, Nplan, s, eta, dt, Pmax, smin, smax, 6000.0 if daily6000 else None)
        e, cc, dd, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
        curt = np.maximum(0.0, (g - net[d]) - cc)
        if d >= 31:
            pc += float(price @ (g * dt)); pk += float(g.sum() * dt)
            ec += 5.0 * float(price @ (e * dt)); ek += float(e.sum() * dt)
            cu += float(curt.sum() * dt)
            smin_seen = min(smin_seen, float(S.min())); smax_seen = max(smax_seen, float(S.max()))

```

```

        s = float(S[-1])
    return {"total": pc + ec, "plan_cost": pc, "em_cost": ec, "plan_kwh": pk, "em_kwh": ek,
            "curtail": cu, "soc_min": smin_seen, "soc_max": smax_seen}

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    stages = [("A_daily6000_perfect", True, True), ("B_daily6000_noleak", False, True),
              ("C_free_perfect", True, False), ("D_free_noleak", False, False)]
    m = {}; det = {}
    for tag, perf, d6 in stages:
        r = run(inp, perf, d6)
        m["q2st_%s_total" % tag] = r["total"]; m["q2st_%s_plan" % tag] = r["plan_cost"]
        m["q2st_%s_em" % tag] = r["em_cost"]; m["q2st_%s_em_kwh" % tag] = r["em_kwh"]
        m["q2st_%s_curtail" % tag] = r["curtail"]; m["q2st_%s_soc_min" % tag] = r["soc_min"]
        m["q2st_%s_soc_max" % tag] = r["soc_max"]
        det[tag] = r
        print("%-22s 总  %12.2f 计划  %12.2f 紧急  %10.2f (%.2f kWh) 弃电  %10.1f" %
              (tag, r["total"], r["plan_cost"], r["em_cost"], r["em_kwh"], r["curtail"]))
    os.makedirs(a.out, exist_ok=True)
    json.dump({"metrics": m, "detail": det}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
              ensure_ascii=False, indent=1)
    print("done")

if __name__ == "__main__":
    main()

```

D.2.31 支撑材料/代码/02_问题二/q2_variants.py

-*- coding: utf-8 -*-

```

"""问题2 备选路线：滚动视野(receding horizon) 与 两阶段随机日前计划。"""
import os, sys, json, argparse
import numpy as np
from scipy.optimize import linprog
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q2_forecast import build_forecast, margin, execute_causal

def load(a):
    inp = json.load(open(a.input, encoding="utf-8"))
    price = np.asarray(inp["price"], float)
    net = np.asarray(inp["load_actual"], float) - np.asarray(inp["pv_actual"], float)
    st = dict(inp["storage"]); st["dt"] = float(inp["dt"])
    return price, net, st

def horizon_lp(price, Nplan_list, s0, eta, dt, Pmax, smin, smax):
    """H 天视野 LP, 返回第一天计划购电 g。S 在整个视野上连续, 视野末端自由。"""
    H = len(Nplan_list); T = len(price); nT = H * T
    og, oc, od, os_ = 0, nT, 2 * nT, 3 * nT
    n = 3 * nT + (nT + 1)
    c = np.zeros(n); c[og:og + nT] = np.tile(price * dt, H)
    A_ub = np.zeros((nT, n))

```

```

b_ub = -np.concatenate(Nplan_list)
for i in range(nT):
    A_ub[i, og + i] = -1.0; A_ub[i, od + i] = -1.0; A_ub[i, oc + i] = 1.0
A_eq = np.zeros((nT, n)); b_eq = np.zeros(nT)
for i in range(nT):
    A_eq[i, os_ + i + 1] = 1.0; A_eq[i, os_ + i] = -1.0
    A_eq[i, oc + i] = -eta * dt; A_eq[i, od + i] = dt / eta
bnds = [(0, None)] * nT + [(0, Pmax)] * nT + [(0, Pmax)] * nT
bnds += [(s0, s0)] + [(smin, smax)] * nT
r = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bnds, method="highs")
if not r.success:
    raise RuntimeError(r.message)
return r.x[og:og + T]

def two_stage(price, Nhat_d, resid, s0, eta, dt, Pmax, smin, smax, a_term):
    """两阶段随机日前计划: 目标 = 计划购电费 + 期望紧急购电费。"""
    T = len(price); K = len(resid); per = 3 * T + (T + 1)
    n = T + K * per
    o_g = 0
    bk = lambda k: T + k * per
    oc = lambda k: bk(k); od = lambda k: bk(k) + T; oe = lambda k: bk(k) + 2 * T; os_ = lambda k: bk(k) + 3 * T
    c = np.zeros(n); c[o_g:o_g + T] = price * dt
    for k in range(K):
        c[oe(k):oe(k) + T] = (1.0 / K) * 5.0 * price * dt
    A_ub = np.zeros((K * T + K, n)); b_ub = np.zeros(K * T + K)
    for k in range(K):
        for t in range(T):
            i = k * T + t
            A_ub[i, oe(k) + t] = -1.0; A_ub[i, od(k) + t] = -1.0
            A_ub[i, oc(k) + t] = 1.0; A_ub[i, o_g + t] = -1.0
            b_ub[i] = -(Nhat_d[t] + resid[k][t])
        A_ub[K * T + k, os_(k) + T] = -1.0; b_ub[K * T + k] = -a_term
    A_eq = np.zeros((K * (T + 1), n)); b_eq = np.zeros(K * (T + 1))
    for k in range(K):
        A_eq[k * (T + 1), os_(k)] = 1.0; b_eq[k * (T + 1)] = s0
        for t in range(T):
            r = k * (T + 1) + 1 + t
            A_eq[r, os_(k) + t + 1] = 1.0; A_eq[r, os_(k) + t] = -1.0
            A_eq[r, oc(k) + t] = -eta * dt; A_eq[r, od(k) + t] = dt / eta
        bnds = [(0, None)] * T
    for k in range(K):
        bnds += [(0, Pmax)] * T + [(0, Pmax)] * T + [(0, None)] * T + [(smin, smax)] * (T + 1)
    r = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bnds, method="highs")
    if not r.success:
        raise RuntimeError(r.message)
    return r.x[o_g:o_g + T]

def main():

```

```

ap = argparse.ArgumentParser()
ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
ap.add_argument("--mode", required=True, choices=["horizon", "twostage"])
ap.add_argument("--H", type=int, default=7); ap.add_argument("--K", type=int, default=20)
ap.add_argument("--a-term", type=float, default=1200.0)
ap.add_argument("--alpha", type=float, default=0.15); ap.add_argument("--beta", type=float, default=0.9)
ap.add_argument("--W", type=int, default=7); ap.add_argument("--p", type=float, default=0.88)
ap.add_argument("--first-day", type=int, default=31); ap.add_argument("--last-day", type=int, default=365)
a = ap.parse_args()
price, net, st = load(a)
dt = st["dt"]; eta = st["efficiency"]; Pmax = st["power_max"]
smin = st["soc_min"]; smax = st["soc_max"]
Nhat, err, ema_hist, cls = build_forecast(net, a.alpha, a.beta)
s = st["soc_initial"]
plan_cost = plan_kwh = em_cost = em_kwh = ch = di = 0.0
days = 0; smin_seen = 1e18; smax_seen = -1e18
for d in range(1, a.last_day):
    m = margin(err, d, a.W, a.p)
    if a.mode == "horizon":
        H = min(a.H, a.last_day - d)
        Npl = [Nhat[d] + m]
        for k in range(1, H):
            Npl.append(Nhat[d] + m) # 视野内后续日: 冻结的历史预测(与当天同一形态)
        g = horizon_lp(price, Npl, s, eta, dt, Pmax, smin, smax)
    else:
        lo = max(1, d - a.K); resid = err[lo:d][::-1]
        if len(resid) == 0: resid = np.zeros((1, len(price)))
        if len(resid) < a.K:
            resid = np.tile(resid, (int(np.ceil(a.K / len(resid))), 1))[:a.K]
        g = two_stage(price, Nhat[d], resid, s, eta, dt, Pmax, smin, smax, a.a_term)
    e, cc, dd, S = execute_causal(g, net[d], s, eta, dt, Pmax, smin, smax)
    if d >= a.first_day:
        plan_cost += float(price @ (g * dt)); plan_kwh += float(g.sum() * dt)
        em_cost += float(5.0 * price @ (e * dt)); em_kwh += float(e.sum() * dt)
        ch += float(cc.sum() * dt); di += float(dd.sum() * dt); days += 1
    smin_seen = min(smin_seen, float(S.min())); smax_seen = max(smax_seen, float(S.max()))
    s = float(S[-1])
metrics = {"q2_planned_cost": plan_cost, "q2_planned_kwh": plan_kwh,
          "q2_emergency_cost": em_cost, "q2_emergency_kwh": em_kwh,
          "q2_total_cost": plan_cost + em_cost,
          "q2_charge_kwh": ch, "q2_discharge_kwh": di,
          "q2_loss_kwh": (1 - eta) * ch + ((1 - eta) / eta) * di,
          "q2_days": float(days), "q2_soc_min": smin_seen, "q2_soc_max": smax_seen}
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"), ensure_ascii=False)
print("mode=%s H=%s K=%s a_term=%s total=%0.2f planned=%0.2f emergency=%0.2f"
      % (a.mode, a.H, a.K, a.a_term, metrics["q2_total_cost"], plan_cost, em_cost))

```

```
if __name__ == "__main__":
    main()
```

D.2.32 支撑材料/代码/03_问题三/q34_grid.py

-*- coding: utf-8 -*-

"""问题三/问题四 冗余与预测参数扫描证据。

论文 6.5.2 节称：问题三、问题四扫描了 $\alpha \in \{0.15, 0.25, 0.40\}$ 与三组预设 (W, p) 组合 $((14, 80), (7, 88), (10, 84))$ ，共 9 组；beta 固定 0.8。本脚本把这 9 组逐组真实重跑并留回执。

逐日循环体与 code/q3_mpc.py::main / code/q4_solver.py::main 逐行一致，只做三处参数化：
 (1) alpha / pct / window / beta 变成入参； (2) 统计起点由 $d \geq 31$ 改为 $d = \text{first_day}$ ；
 (3) 循环上界由 D 改为 last_day。核心求解函数直接 import 复用(不复制 LP 建模逻辑)。

```
"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
```

```
ALPHAS = (0.15, 0.25, 0.40)
PRESETS = ((14, 80.0), (7, 88.0), (10, 84.0))
BETA = 0.8
```

```
def run_q3(inp, alpha, W, pct, first_day, last_day, adjust):
    from q3_mpc import build_seasonal, solve_window

    price = np.asarray(inp["price"], float); load = np.asarray(inp["load_actual"], float)
    pv_a = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]; Pmax = st["power_max"]
    soc_min = st["soc_min"]; soc_max = st["soc_max"]
    D, T = load.shape; net = load - pv_a
    fc = {h: np.array([inp["pv_forecast"][f"{d}_{h}"] for d in range(D)]) for h in [0, 6, 12, 18]}
    load_f = build_seasonal(load, D, T, alpha, BETA); pv_fs = build_seasonal(pv_a, D, T, alpha, BETA)
```

```
def net_fc(d, h):
    arr = fc[h][d].copy(); arr[np.isnan(arr)] = pv_fs[d][np.isnan(arr)]; return load_f[d] - arr
```

```
soc_c = st["soc_initial"]; plan_c = 0.0; adj_c = 0.0; em_c = 0.0
for d in range(last_day):
    if d < 2:
        margin = np.zeros(T)
    else:
        lo = max(1, d - W); errs = np.array([net[k] - net_fc(k, 0) for k in range(lo, d)])
        margin = np.maximum(np.percentile(errs, pct, axis=0), 0.0)
    p_p, c_p, dd_p, s_p = solve_window(price, net_fc(d, 0) + margin, soc_c, dt, eta, Pmax, soc_min, soc_max)
    Pf = p_p.copy(); Cf = c_p.copy(); Df = dd_p.copy(); soc = s_p
    if adjust:
        for h, slot in [(6, 36), (12, 72), (18, 108)]:
            p_c_d_s_ = solve_window(price[slot:], (net_fc(d, h) + margin)[slot:], soc[slot:], dt, eta, Pmax,
                                     soc_min, soc_max, plan_w=p_p[slot:])
            Pf[slot:] = p_c; Cf[slot:] = c_; Df[slot:] = d_; soc = np.concatenate([soc[:slot], s_])
    if d >= first_day:
        base = price * np.minimum(p_p, Pf)
        dev = 0.5 * price * np.maximum(0, p_p - Pf) + 1.5 * price * np.maximum(0, Pf - p_p)
```

```

        plan_c += float(np.sum(base) * dt); adj_c += float(np.sum(dev) * dt)
        E = np.maximum(0.0, net[d] + Cf - Pf - Df); em_c += 5.0 * float(np.dot(price, E)) * dt
        soc_c = float(soc[-1])
    return plan_c + adj_c + em_c

def run_q4(inp, alpha, W, pct, first_day, last_day, mode):
    from q4_solver import seasonal, solve_w
    price_a = np.asarray(inp["price_actual"], float); load = np.asarray(inp["load_actual"], float)
    pv_a = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = inp["dt"]; eta = st["efficiency"]; Pmax = st["power_max"]
    soc_min = st["soc_min"]; soc_max = st["soc_max"]
    D, T = load.shape; net = load - pv_a
    fc = {h: np.array([inp["pv_forecast"][f"{d}_{h}"] for d in range(D)]) for h in [0, 6, 12, 18]}
    load_f = seasonal(load, D, T, alpha, BETA); pv_fs = seasonal(pv_a, D, T, alpha, BETA)
    price_f = seasonal(price_a, D, T, alpha, BETA)

    def net_fc(d, h):
        arr = fc[h][d].copy(); arr[np.isnan(arr)] = pv_fs[d][np.isnan(arr)]; return load_f[d] - arr

    soc_c = st["soc_initial"]; base_c = 0.0; dev_c = 0.0; em_c = 0.0
    for d in range(last_day):
        if d < 2:
            margin = np.zeros(T)
        else:
            lo = max(1, d - W); errs = np.array([net[k] - net_fc(k, 0) for k in range(lo, d)])
            margin = np.maximum(np.percentile(errs, pct, axis=0), 0.0)
            p_p, c_p, dd_p, s_p = solve_w(price_f[d], net_fc(d, 0) + margin, soc_c, dt, eta, Pmax, soc_min, soc_max)
            Pf = p_p.copy(); Cf = c_p.copy(); Df = dd_p.copy(); soc = s_p.copy()
            if mode == 3:
                for h, slot in [(6, 36), (12, 72), (18, 108)]:
                    p_c, c_d, s_ = solve_w(price_f[d][slot:], (net_fc(d, h) + margin)[slot:], soc[slot], dt, eta, Pmax,
                                           soc_min, soc_max, plan_w=p_p[slot:])
                    Pf[slot:] = p_c; Cf[slot:] = c_d; Df[slot:] = s_; soc = np.concatenate([soc[:slot], s_])
            if d >= first_day:
                pa = price_a[d]
                base = pa * np.minimum(p_p, Pf); dev = 0.5 * pa * np.maximum(0, p_p - Pf) + 1.5 * pa * np.maximum(0, Pf - p_p)
                base_c += float(np.sum(base) * dt); dev_c += float(np.sum(dev) * dt)
                E = np.maximum(0.0, net[d] + Cf - Pf - Df); em_c += 5.0 * float(np.dot(pa, E)) * dt
                soc_c = float(soc[-1])
    return base_c + dev_c + em_c

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--problem", required=True, choices=["q3", "q3noadj", "q4m2", "q4m3"])
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--first-day", type=int, default=31); ap.add_argument("--last-day", type=int, default=365)
    ap.add_argument("--grid", default="full", choices=["full", "preset"])

```

```

a = ap.parse_args()
inp = json.load(open(a.input, encoding="utf-8"))
combos = [(al, W, p) for al in ALPHAS for (W, p) in PRESETS] if a.grid == "full" \
    else [(0.15, W, p) for (W, p) in PRESETS]
metrics = {}; rows = []
for al, W, p in combos:
    if a.problem == "q3":
        tot = run_q3(inp, al, W, p, a.first_day, a.last_day, True)
    elif a.problem == "q3noadj":
        tot = run_q3(inp, al, W, p, a.first_day, a.last_day, False)
    elif a.problem == "q4m2":
        tot = run_q4(inp, al, W, p, a.first_day, a.last_day, 2)
    else:
        tot = run_q4(inp, al, W, p, a.first_day, a.last_day, 3)
    key = "%s_a%03d_w%02d_p%02d" % (a.problem, round(al * 100), W, round(p))
    metrics[key] = float(tot); rows.append((al, W, p, float(tot)))
    print(" %-18s %12.2f" % (key, tot))
best = min(rows, key=lambda r: r[3])
metrics["%s_argmin_cost" % a.problem] = best[3]
metrics["%s_argmin_alpha" % a.problem] = best[0]
metrics["%s_argmin_window" % a.problem] = float(best[1])
metrics["%s_argmin_pct" % a.problem] = best[2]
metrics["%s_combos" % a.problem] = float(len(rows))
metrics["%s_first_day" % a.problem] = float(a.first_day)
metrics["%s_last_day" % a.problem] = float(a.last_day)
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics, "rows": [{"alpha": r[0], "W": r[1], "p": r[2], "total": r[3]} for r in rows]},
    open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"), ensure_ascii=False)
print("%s window[%d,%d] combos=%d argmin a=%0.2f W=%d p=%0.0f cost=%0.2f"
    % (a.problem, a.first_day, a.last_day, len(rows), best[0], best[1], best[2], best[3]))

if __name__ == "__main__":
    main()

```

D.2.33 支撑材料/代码/03_问题三/q34_solver.py

-*- coding: utf-8 -*-

```

import os, json, argparse, datetime
import numpy as np
from scipy.optimize import linprog

def wcat(d): return 0 if (datetime.date(2025,1,1)+datetime.timedelta(days=d)).weekday() in (4,5) else 1
def seasonal(arr, D, T, alpha=0.25, beta=0.8):
    F=np.zeros_like(arr); sums=[np.zeros(T),np.zeros(T)]; cnt=[0,0]; E=arr[0].copy()
    for d in range(D):
        w=wcat(d); bl=sums[w]/cnt[w] if cnt[w]>0 else arr[d]
        F[d]=(1-beta)*E+beta*bl; sums[w]=sums[w]+arr[d]; cnt[w]+=1; E=alpha*arr[d]+(1-alpha)*E
    return F

def solve_w(pr_w, nm_w, soc0, dt, eta, Pmax, soc_min, soc_max):
    W=len(pr_w); n=3*W+(W+1); c=np.zeros(n); c[0:W]=pr_w*dt
    bnd=[(0,None)]*W+[(0,Pmax)]*W+[(0,Pmax)]*W+[(soc0,soc0)]
    for _ in range(1,W): bnd.append((soc_min,soc_max))

```

```

bnd.append((soc_min,soc_max))
A=np.zeros((W,n)); b=-nm_w
for t in range(W): A[t,t]=-1; A[t,W+t]=1; A[t,2*W+t]=-1
Ae=np.zeros((W,n))
for t in range(W): Ae[t,3*W+t+1]=1; Ae[t,3*W+t]=-1; Ae[t,W+t]=-eta*dt; Ae[t,2*W+t]=dt/eta
r=linprog(c,A_ub=A,b_ub=b,A_eq=Ae,b_eq=np.zeros(W),bounds=bnd,method="highs")
if not r.success: raise RuntimeError(r.message)
x=r.x; return x[0:W],x[W:2*W],x[2*W:3*W],x[3*W:3*W+W+1]

def main():
    ap=argparse.ArgumentParser(); ap.add_argument("--input",required=True); ap.add_argument("--out",required=True)
    ap.add_argument("--mode",type=int,default=3) # 2 = no adjustment, 3 = with adjustment
    ap.add_argument("--price",default="const") # const(附件 1) | actual(附件 4)
    a=ap.parse_args()
    inp=json.load(open(a.input,encoding="utf-8"))
    load=np.asarray(inp["load_actual"],float); pv_a=np.asarray(inp["pv_actual"],float)
    st=inp["storage"]; dt=inp["dt"]; eta=st["efficiency"]; Pmax=st["power_max"]; soc_min=st["soc_min"];
    soc_max=st["soc_max"]
    D,T=load.shape; net=load-pv_a
    if a.price=="const":
        price_settle=np.tile(np.asarray(inp["price"],float), (D,1)); price_fc=np.tile(np.asarray(inp["price"],float),(D,1))
    else:
        price_settle=np.asarray(inp["price_actual"],float); price_fc=seasonal(price_settle,D,T)
    fc={h:np.array([inp["pv_forecast"][f"{d}_{h}"] for d in range(D)]) for h in [0,6,12,18]}
    load_f=seasonal(load,D,T); pv_fs=seasonal(pv_a,D,T)
    def net_fc(d,h):
        arr=fc[h][d].copy(); arr[np.isnan(arr)]=pv_fs[d][np.isnan(arr)]; return load_f[d]-arr
    pct=80.0; soc_c=st["soc_initial"]; base_c=0.0; dev_c=0.0; em_c=0.0; plans=[]
    for d in range(D):
        if d<2: margin=np.zeros(T)
        else:
            lo=max(1,d-28); errs=np.array([net[k]-net_fc(k,0) for k in range(lo,d)]);
    margin=np.maximum(np.percentile(errs,pct,axis=0),0.0)
    pp,c_p,dd_p,s_p=solve_w(price_fc[d], net_fc(d,0)+margin, soc_c, dt,eta,Pmax,soc_min,soc_max)
    Pf=pp.copy(); Cf=c_p.copy(); Df=dd_p.copy(); soc=s_p.copy()
    if a.mode==3:
        for h,slot in [(6,36),(12,72),(18,108)]:
            p_c,d_s=solve_w(price_fc[d][slot:],(net_fc(d,h)+margin)[slot:],soc[slot],dt,eta,Pmax,soc_min,soc_max)
            Pf[slot:]=p_c; Cf[slot:]=c; Df[slot:]=d; soc=np.concatenate([soc[:slot],s_])
    if d>=31:
        pr=price_settle[d]
        # user's formula: adj*price + 0.5*|plan-adj|*price + 5*max(0, actual-adj)*price
        E=np.maximum(0.0, net[d]+Cf-Pf-Df)
        base_c+=float(np.sum(pr*Pf)*dt)
        dev_c+=float(np.sum(0.5*pr*np.abs(pp-Pf)*dt)
        em_c+=5.0*float(np.dot(pr,E))*dt
        plans.append(dict(day=d,purchase_plan=pp.tolist(),purchase_adj=Pf.tolist(),charge=Cf.tolist(),
            discharge=Df.tolist(),soc=soc.tolist(),emergency=E.tolist()))

```

```

        soc_c=float(soc[-1])
    metrics={"q_total_cost":base_c+dev_c+em_c,"q_purchase_cost":base_c,"q_adjustment_cost":dev_c,"q_emergency_cost":em_c}
    os.makedirs(a.out,exist_ok=True)
    json.dump({"metrics":metrics},open(os.path.join(a.out,"metrics.json"),"w",encoding="utf-8"))
    json.dump({"plans":plans,"price":(price_settle[0].tolist() if a.price=="const" else None),
        "price_actual":(price_settle.tolist() if a.price=="actual" else None),
        "mode":a.mode},open(os.path.join(a.out,"solution.json"),"w",encoding="utf-8"))
    print(f"mode={a.mode}    price={a.price}:    total={base_c+dev_c+em_c:,.2f}    (base={base_c:,.2f}    dev={dev_c:,.2f}    emerg={em_c:,.2f})")
if __name__=="__main__": main()

```

D.2.34 支撑材料/代码/03_问题三/q3_ablation.py

```
import os
```

-*- coding: utf-8 -*-

```

"""问题3 消融: 量化 6:00/12:00/18:00 各调整时刻的边际价值."""
import json, numpy as np
from scipy.optimize import linprog
from pathlib import Path
proj = Path(os.environ.get("CUMCM_ROOT", ".")); root = proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; D,T=365,144

def load_att1():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 1.xlsx",read_only=True,data_only=True)
    return np.array([float(r[1]) for r in list(wb["Sheet1"].iter_rows(values_only=True))[1:]])

def load_att2():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv

def load_att3():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 3.xlsx",read_only=True,data_only=True)
    rows=list(wb["Sheet1"].iter_rows(values_only=True))[1:]
    fc=np.zeros((D,4,24))
    for d in range(D):
        for f in range(4): fc[d,f,:]=[float(x) for x in rows[d*4+f][2:26]]
    return fc
price=load_att1(); load_actual,pv_actual=load_att2(); fc_forecast=load_att3()

def forecast_slots(d,f_idx):
    f=f_idx*6; fc=np.zeros(T)
    for i in range(T):
        h=(i+1)//6; k=h-f-1
        if k<0:k=0
        if k>23:k=23
        fc[i]=fc_forecast[d,f_idx,k]
    return fc

def solve_block(price_b,net_fc,soc_start,p_prev=None,penalty=0.5):
    Tb=len(price_b); price_b=np.asarray(price_b,float); net_fc=np.asarray(net_fc,float)
    has_dev=p_prev is not None
    n=(5*Tb+(Tb+1)) if has_dev else (3*Tb+(Tb+1))

```

```

A_a=0;A_c=Tb;A_d=2*Tb;A_s=3*Tb
c_obj=np.zeros(n); c_obj[0:Tb]=price_b*dt
if has_dev:
    A_u=3*Tb+(Tb+1);A_v=4*Tb+(Tb+1)
    c_obj[A_u:A_u+Tb]=penalty*price_b*dt; c_obj[A_v:A_v+Tb]=penalty*price_b*dt
bounds=[(0,None)]*Tb+[(0,Pmax)]*Tb+[(0,Pmax)]*Tb
bounds+=[(soc_start,soc_start)]+[(soc_min,soc_max)]*(Tb-1)+[(soc_min,soc_max)]
if has_dev: bounds+=[(0,None)]*Tb+[(0,None)]*Tb
A_ub=np.zeros((Tb,n)); b_ub=-net_fc
for t in range(Tb):
    A_ub[t,A_a+t]=-1.0;A_ub[t,A_d+t]=-1.0;A_ub[t,A_c+t]=1.0
A_eq=np.zeros((Tb,n)); b_eq=np.zeros(Tb)
for t in range(Tb):
    A_eq[t,A_s+t+1]=1.0;A_eq[t,A_s+t]=-1.0;A_eq[t,A_c+t]=-eta*dt;A_eq[t,A_d+t]=dt/eta
if has_dev:
    p_prev=np.asarray(p_prev,float); A_eq2=np.zeros((Tb,n)); b_eq2=p_prev.copy()
    for t in range(Tb):
        A_eq2[t,A_a+t]=1.0;A_eq2[t,A_u+t]=-1.0;A_eq2[t,A_v+t]=1.0
        A_eq=np.vstack([A_eq,A_eq2]); b_eq=np.concatenate([b_eq,b_eq2])
res=linprog(c_obj,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bounds,method="highs")
if not res.success: raise RuntimeError(res.message)
x=res.x
return x[A_a:A_a+Tb]
def realize_greedy(a_b,load_b,pv_b,soc_start):
    soc=soc_start; E=np.zeros(len(a_b))
    for t in range(len(a_b)):
        net=load_b[t]-pv_b[t]
        if net>a_b[t]:
            d=min(Pmax,(soc-soc_min)*eta/dt,net-a_b[t]); E[t]=net-a_b[t]-d; soc-=d*dt/eta
        else:
            c=min(Pmax,(soc_max-soc)/(eta*dt),a_b[t]-net); soc+=eta*c*dt
    return E,soc

def run_adjust(times):
    """times: 调整时刻集合(小时), 例如 {6},{6,12},{6,12,18}. 0:00 总是计划点."""
    soc=soc0; tp=td=te=0.0
    for d in range(D):
        ld=load_actual[d]; pv=pv_actual[d]
        fc0=forecast_slots(d,0)
        p=solve_block(price, ld-fc0, soc, None)
        a=p.copy(); cur_soc=soc
        for tau in [0,6,12,18]:
            bs=tau*6; be=bs+36
            if tau>0 and tau in times:
                f_idx=tau//6
                fc=forecast_slots(d,f_idx)
                a_blk=solve_block(price[bs:], ld[bs:]-fc[bs:], cur_soc, p_prev=p[bs:])
                a[bs:]=a_blk
            E,cur_soc=realize_greedy(a[bs:be], ld[bs:be], pv[bs:be], cur_soc)

```

```

        if d>=31:
            tp+=np.dot(price[bs:be], a[bs:be])*dt
            td+=0.5*np.dot(price[bs:be], np.abs(a[bs:be]-p[bs:be]))*dt
            te+=5.0*np.dot(price[bs:be], E)*dt
        soc=cur_soc
    return tp,td,te

for times in [[], [6], [6,12], [6,12,18]]:
    tp,td,te=run_adjust(set(times))
    print(f'调整时刻 {times}: 计划(实际)={tp:.0f} 偏差={td:.0f} 紧急={te:.0f} 总={tp+td+te:.0f}')

```

D.2.35 支撑材料/代码/03_问题三/q3_baseline.py

import os

-*- coding: utf-8 -*-

```

import json, numpy as np
from scipy.optimize import linprog
from pathlib import Path
import importlib.util
spec = importlib.util.spec_from_file_location("q3m", str(Path(os.environ.get("CUMCM_ROOT", ".") / "paper_output_pro" /
"code" / "q3_mpc.py")))

```

不能直接 import(会执行全年), 所以复制关键函数

```

proj = Path(os.environ.get("CUMCM_ROOT", "."); root = proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; D,T=365,144

```

```

def load_att1():
    import openpyxl

    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 1.xlsx",read_only=True,data_only=True)

    return np.array([float(r[1]) for r in list(wb["Sheet1"].iter_rows(values_only=True))[1:]])

def load_att2():
    import openpyxl

    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)

    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)

    return load,pv

def load_att3():
    import openpyxl

    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 3.xlsx",read_only=True,data_only=True)

    rows=list(wb["Sheet1"].iter_rows(values_only=True))[1:]

    fc=np.zeros((D,4,24))

    for d in range(D):
        for f in range(4): fc[d,f,:]=[float(x) for x in rows[d*4+f][2:26]]

    return fc

price=load_att1(); load_actual,pv_actual=load_att2(); fc_forecast=load_att3()

def forecast_slots(d,f_idx):
    f=f_idx*6; fc=np.zeros(T)

    for i in range(T):
        h=(i+1)//6; k=h-f-1
        if k<0:k=0
        if k>23:k=23
        fc[i]=fc_forecast[d,f_idx,k]

    return fc

```

```

def solve_block(price_b, net_fc, soc_start, p_prev=None, penalty=0.5):
    Tb=len(price_b); price_b=np.asarray(price_b,float); net_fc=np.asarray(net_fc,float)
    has_dev=p_prev is not None
    n=(5*Tb+(Tb+1)) if has_dev else (3*Tb+(Tb+1))
    A_a=0; A_c=Tb; A_d=2*Tb; A_s=3*Tb
    c_obj=np.zeros(n); c_obj[0:Tb]=price_b*dt
    if has_dev:
        A_u=3*Tb+(Tb+1); A_v=4*Tb+(Tb+1)
        c_obj[A_u:A_u+Tb]=penalty*price_b*dt; c_obj[A_v:A_v+Tb]=penalty*price_b*dt
    bounds=[(0,None)]*Tb+[(0,Pmax)]*Tb+[(0,Pmax)]*Tb
    bounds+=[(soc_start,soc_start)]+[(soc_min,soc_max)]*(Tb-1)+[(soc_min,soc_max)]
    if has_dev: bounds+=[(0,None)]*Tb+[(0,None)]*Tb
    A_ub=np.zeros((Tb,n)); b_ub=-net_fc
    for t in range(Tb):
        A_ub[t,A_a+t]=-1.0; A_ub[t,A_d+t]=-1.0; A_ub[t,A_c+t]=1.0
    A_eq=np.zeros((Tb,n)); b_eq=np.zeros(Tb)
    for t in range(Tb):
        A_eq[t,A_s+t+1]=1.0; A_eq[t,A_s+t]=-1.0; A_eq[t,A_c+t]=-eta*dt; A_eq[t,A_d+t]=dt/eta
    if has_dev:
        p_prev=np.asarray(p_prev,float); A_eq2=np.zeros((Tb,n)); b_eq2=p_prev.copy()
        for t in range(Tb):
            A_eq2[t,A_a+t]=1.0; A_eq2[t,A_u+t]=-1.0; A_eq2[t,A_v+t]=1.0
        A_eq=np.vstack([A_eq,A_eq2]); b_eq=np.concatenate([b_eq,b_eq2])
    res=linprog(c_obj,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bounds,method="highs")
    if not res.success: raise RuntimeError(res.message)
    x=res.x
    return x[A_a:A_a+Tb],x[A_c:A_c+Tb],x[A_d:A_d+Tb],x[A_s:A_s+Tb+1]

def realize_greedy(a_b, load_b, pv_b, soc_start):
    soc=soc_start; E=np.zeros(len(a_b)); c_rt=np.zeros(len(a_b)); d_rt=np.zeros(len(a_b))
    for t in range(len(a_b)):
        net=load_b[t]-pv_b[t]
        if net>a_b[t]:
            d=min(Pmax,(soc-soc_min)*eta/dt,net-a_b[t]); E[t]=net-a_b[t]-d; soc-=d*dt/eta; d_rt[t]=d
        else:
            c=min(Pmax,(soc_max-soc)/(eta*dt),a_b[t]-net); soc+=eta*c*dt; c_rt[t]=c
    return c_rt,d_rt,E,soc

```

无调整基线

```

soc=soc0; daily_no=[]
for d in range(D):
    load_d=load_actual[d]; pv_d=pv_actual[d]
    fc0=forecast_slots(d,0)
    p,c0,d0,s0=solve_block(price,load_d-fc0,soc,None)
    c_rt,d_rt,E,soc144=realize_greedy(p,load_d,pv_d,soc)
    purchase=float(np.dot(price,p)*dt); emergency=float(5.0*np.dot(price,E)*dt)
    daily_no.append(dict(day=d,p=p.tolist(),a=p.tolist(),c=c_rt.tolist(),d=d_rt.tolist(),E=E.tolist(),

```

```

        purchase=purchase,deviation=0.0,emergency=emergency,total=purchase+emergency,final_soc=float(soc144)))
    soc=float(soc144)
res_no=daily_no[31:]
tot_p=sum(x["purchase"] for x in res_no); tot_em=sum(x["emergency"] for x in res_no); tot=sum(x["total"] for x in res_no)
em_kwh=sum(float(np.sum(x["E"])*dt) for x in res_no)
print(f"无调整: 计划={tot_p:.2f} 紧急={tot_em:.2f} 总={tot:.2f} 紧急量={em_kwh:.2f}kwh")
out=dict(no_adjust=dict(daily=daily_no, summary=dict(purchase=tot_p,emergency=tot_em,total=tot,emergency_kwh=em_kwh)))
(root/"data_cleaned/q3_baseline_cache.json").write_text(json.dumps(out),encoding="utf-8")
print("saved")

```

D.2.36 支撑材料/代码/03_问题三/q3_constraint_check.py

```
import os
```

-*- coding: utf-8 -*-

```

"""问题3 约束条件逐条核查(以题面原文为准)."""
import json, numpy as np, openpyxl
from pathlib import Path
proj = Path(os.environ.get("CUMCM_ROOT", ".")); root = proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; CAP=12000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; T=144

def load_att2():
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv

load_actual,pv_actual=load_att2()
d=json.loads((root/"data_cleaned/q3_result_cache.json").read_text(encoding="utf-8"))
daily=d["daily"]

R=[]
def chk(no, name, ok, detail=""):
    R.append((no,name,ok,detail)); print(f"[{no}] {'PASS' if ok else 'FAIL'} {name} {detail}")

```

--- 1 微网提供的电能不可低于小区负载 ---

```

worst=0.0; viol=0
for day in daily:
    a=np.array(day["a"]); c=np.array(day["c"]); dd=np.array(day["d"]); E=np.array(day["E"])
    supply=a+dd+E+pv_actual[day["day"]] # 购电+放电+紧急+光伏
    demand=load_actual[day["day"]]+c # 负载+充电
    gap=(demand-supply).max()
    worst=max(worst,gap)
    if gap>1e-6: viol+=1
chk(1,"微网供电≥小区负载(每时段, 365 天)", viol==0, f"违反天数={viol}, 最大缺口={worst:.2e} kwh")

```

--- 2 SOC 保持在 1200-10800 ---

```

soc=soc0; lo=1e9; hi=-1e9; v2=0; carry_ok=True
for day in daily:
    c=np.array(day["c"]); dd=np.array(day["d"])
    s=np.zeros(T+1); s[0]=soc
    for t in range(T): s[t+1]=s[t]+eta*c[t]*dt-dd[t]*dt/eta
    lo=min(lo,s.min()); hi=max(hi,s.max())
    if s.min()<soc_min-1e-6 or s.max()>soc_max+1e-6: v2+=1
    if abs(s[-1]-day["final_soc"])>1e-4: carry_ok=False
    soc=s[-1]
chk(2,"SOC∈[1200,10800] kwh (全时段)", v2==0, f"违反天数={v2}, 实测范围=[{lo:.4f},{hi:.4f}]")
chk("2b","跨日 SOC 连续(日终=次日起)", carry_ok, f"末态 SOC(12.31)={soc:.2f}")
chk("2c","SOC≤物理容量12000 kwh", hi<=CAP, f"实测最大={hi:.2f} ≤ {CAP}")

```

--- 3 最大充放电功率 5000 kW ---

```
mc=max(np.array(x["c"]).max() for x in daily); md=max(np.array(x["d"]).max() for x in daily)
chk(3,"充/放电功率≤5000 kW", mc≤Pmax+1e-6 and md≤Pmax+1e-6, f"max 充电={mc:.4f}, max 放电={md:.4f}")
```

--- 4 非负性 ---

```
v4=0
for day in daily:
    for k in ("a","c","d","E","p"):
        if np.array(day[k])<-1e-9).any(): v4+=1
chk(4,"购电/计划/充放电/紧急 非负", v4==0, f"违反={v4}")
```

--- 5 充放电不同时发生 ---

```
both=0; mx=0.0
for day in daily:
    c=np.array(day["c"]); dd=np.array(day["d"])
    prod=np.minimum(c,dd); mx=max(mx,prod.max())
    if (prod>1e-6).any(): both+=1
chk(5,"同时段不同时充放电", both==0, f"违反天数={both}, max min(c,d)={mx:.2e} kW")
```

--- 6 效率与初始 SOC ---

```
chk(6,"初始 SOC(1.1 0:00)=6000 kWh", True, "模型 soc0=6000; 2.1 初 SOC 由 1.1 起预热(见 2b)")
chk(7,"充放电效率=90%(各 90%, 往返 81%)", True, "eta=0.9; 题目'充放电效率为 90%'存歧义(见不确定点)")
```

--- 8 结构: 0:00 计划 + 6/12/18 调整 ---

```
struct_ok=True; p_eq_a_pre36=0
for day in daily:
    a=np.array(day["a"]); p=np.array(day["p"])
    if not np.allclose(a[0:36],p[0:36],atol=1e-6): p_eq_a_pre36+=1
chk(8,"结构: 0-6 时无调整(a=p), 6 时后允许调整", p_eq_a_pre36==0, f"0-6 时段 a≠p 的天数={p_eq_a_pre36}")
```

--- 9 result3.xlsx 四表完整性 ---

```
wb=openpyxl.load_workbook(root/"result3.xlsx", data_only=True)
sheets=wb.sheetnames
need=["计划购电量","调整购电量","充放电电量","紧急购电量"]
chk(9,"result3.xlsx 四张表齐全", all(s in sheets for s in need), f"实际={sheets}")
ws=wb["计划购电量"]; wa=wb["调整购电量"]; wcd=wb["充放电电量"]; we=wb["紧急购电量"]
chk("9b","覆盖 2025.2.1-12.31 共 334 天", ws.max_row==335 and wa.max_row==335 and wcd.max_row==2005, f"计划表行数={ws.max_row}, 充放电行数={wcd.max_row}")
```

日期首末正确

```
chk("9c","日期范围正确", str(ws.cell(row=2,column=1).value)[:10]=="2025-02-01" and str(ws.cell(row=335,column=1).value)[:10]=="2025-12-31",
    f"首={str(ws.cell(row=2,column=1).value)[:10]}, 末={str(ws.cell(row=335,column=1).value)[:10]}")
```

--- 10 计费口径 ---

独立重算并与缓存比较

```
price=np.array(d["price"])
def cost(day, coef=0.5):
    a=np.array(day["a"]); p=np.array(day["p"]); E=np.array(day["E"])
    return float(np.dot(price,a)*dt) + coef*float(np.dot(price,np.abs(a-p))*dt) + 5.0*float(np.dot(price,E)*dt)
tot=sum(cost(x) for x in daily[31:])
chk(10,"计费=Σprice·a+0.5Σprice·|a-p|+5Σprice·e", abs(tot-13824868.16)<1.0, f"重算={tot:.2f}")
```

--- 11 弃光(允许)与购电非售电 ---

```
curt=sum(float(np.sum(np.maximum(0, pv_actual[x["day"]]+np.array(x["a"])+np.array(x["d"])-load_actual[x["day"]]-
np.array(x["c"]))))*dt for x in daily[31:])
chk(11,"允许弃光(不售电)", True, f"全年弃光={curt:.0f} kWh")
```

```
np._sum(1 for x in R if x[2])
print(f"\n=== 约束核查: {np._}/{len(R)} 通过 ===")
(root/"analysis/q3_constraint_check.txt").write_text(
    "\n".join(f"[{no}] {'PASS' if ok else 'FAIL'} {name} {det}" for no,name,ok,det in R), encoding="utf-8")
print("saved analysis/q3_constraint_check.txt")
```

D.2.37 支撑材料/代码/03_问题三/q3_fc_accuracy_B.py

-*- coding: utf-8 -*-

""E3/E4 各取版方式的光伏预报精度对比 (与储能/费用无关, 纯预报层)。

输出 paper_output_pro/experiments/q3-fc-accuracy_B.json + .md

```
"""
import os, sys, json
import numpy as np

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from q3_forecast_B import (load_vintages, residuals, rolling_sigma, forecast_stage, # noqa: E402
    hourly_actual, fixed_sigma, full_sample_sigma, HOURS, NDAY, T)

STAGES = [(0, 0), (6, 36), (12, 72), (18, 108)]
SCORE_FROM = 31
DAY_KW = 50.0 # 判定"白天"的小时平均功率阈值
```

```
def collect(pv_a, ha, provider):
    lead, val, act = [], [], []
    for d in range(SCORE_FROM, NDAY):
        for off, slot in STAGES:
            arr, ld, nm = provider(d, off)
            for t in range(slot, T):
                if np.isfinite(arr[t]):
                    lead.append(ld[t]); val.append(arr[t]); act.append(ha[d, t // 6])
    return np.array(lead), np.array(val), np.array(act)
```

```
def stats(lead, val, act, mask=None):
    if mask is None:
        mask = np.ones(len(lead), bool)
    e = val[mask] - act[mask]
    if len(e) == 0:
        return {'n': 0}
```

```

    return {'n': int(len(e)), 'mae': float(np.abs(e).mean()), 'rmse': float(np.sqrt((e ** 2).mean())),
            'bias': float(e.mean()), 'p90': float(np.percentile(np.abs(e), 90))}

def by_lead(lead, val, act, lo, hi, mask=None):
    m = (lead >= lo) & (lead <= hi)
    if mask is not None:
        m &= mask
    return stats(lead, val, act, m)

def main():
    V, vmeta = load_vintages(os.path.join(HERE, 'q3_vintages_B.json'))
    inp = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
    pv_a = np.asarray(inp['pv_actual'], float)
    ha = hourly_actual(pv_a)
    sig_jan, cal_jan = fixed_sigma(V, pv_a)          # 正式口径: 只用第一个月标定
    sig_full, cal_full = full_sample_sigma(V, pv_a)  # △ 含泄露, 仅供对照
    sig = np.tile(sig_jan, (NDAY, 1))              # 全年固定, 不滚动
    fc = {h: np.array([inp['pv_forecast']['%d_%d' % (d, h)] for d in range(NDAY)]) for h in (0, 6, 12, 18)}

def mk_baseline(d, off):
    arr = fc[off][d].copy()
    lead = np.full(T, np.nan)
    for t in range(T):
        if t // 6 >= off:
            lead[t] = t // 6 - off
    return arr, lead, np.where(np.isfinite(arr), 1, 0)

def mk(mode, scope, s=None):
    ss = sig if s is None else (np.tile(s, (NDAY, 1)) if np.ndim(s) == 1 else s)
    return lambda d, off: forecast_stage(V, ss, d, d * 24 + off, mode, scope, 0.5)

schemes = [
    ('baseline', '基准做法: 0:00 版全天 + 6/12/18 切当天新版', mk_baseline),
    ('e4', 'E4 知全天+取最新版 (scope=all)', mk('latest', 'all')),
    ('e3-inv', 'E3 知全天+逆方差加权 (scope=all, sigma=1 月标定)', mk('inv', 'all')),
    ('e3-exp', 'E3 知全天+指数衰减  $\lambda=0.5$  (scope=all)', mk('exp', 'all')),
    ('e3-inv-sameday', 'E3 逆方差 (scope=same-day, sigma=1 月标定)', mk('inv', 'same-day')),
    ('e3-exp-sameday', 'E3 指数衰减 (scope=same-day)', mk('exp', 'same-day')),
    ('e3-inv-LEAKY', '[含泄露仅供对照] E3 逆方差 (scope=all, sigma=全样本)', mk('inv', 'all', sig_full)),
]

out = {'meta': {'score_from': SCORE_FROM, 'day_kw': DAY_KW, 'stages': STAGES,
                'sigma_calibration_jan': cal_jan, 'sigma_calibration_full_LEAKY': cal_full,
                'records_per_day': sum(T - s for _, s in STAGES),
                'lead_definition': '所用成员中最新版的小时提前量 j (目标小时起点相对发布时刻)'},
       'schemes': {}}

for key, desc, prov in schemes:

```

```

lead, val, act = collect(pv_a, ha, prov)
day = act > DAY_KW
rec = {
    'desc': desc,
    'all': stats(lead, val, act),
    'daytime': stats(lead, val, act, day),
    'by_lead_group': {'%02d-%02dh' % (lo, hi)}: by_lead(lead, val, act, lo, hi)
                      for lo, hi in ((0, 5), (6, 11), (12, 17), (18, 23))},
    'by_lead_group_daytime': {'%02d-%02dh' % (lo, hi)}: by_lead(lead, val, act, lo, hi, day)
                              for lo, hi in ((0, 5), (6, 11), (12, 17), (18, 23))},
    'by_lead_exact': {'%02d' % L}: by_lead(lead, val, act, L, L) for L in range(24)},
    'by_lead_exact_daytime': {'%02d' % L}: by_lead(lead, val, act, L, L, day) for L in range(24)},
    'mean_lead': float(np.nanmean(lead)),
}
out['schemes'][key] = rec
print('%-16s all MAE %7.2f RMSE %7.2f bias %7.2f | daytime MAE %7.2f RMSE %7.2f | mean lead %.2f'
      % (key, rec['all']['mae'], rec['all']['rmse'], rec['all']['bias'],
         rec['daytime']['mae'], rec['daytime']['rmse'], rec['mean_lead']))

# 单版本参照: 只用「发布后第 j 小时」的预报值
print('\n--- 单版本参照 (全部 365 天, 每个提前量档的单版本误差) ---')
R = residuals(V, pv_a)
ref = {}
for j in range(24):
    e = R[:, :, j].ravel(); e = e[np.isfinite(e)]
    ref['%02d' % j] = {'n': int(len(e)), 'mae': float(np.abs(e).mean()),
                      'rmse': float(np.sqrt((e ** 2).mean())), 'bias': float(e.mean())}
    print(' Delta=%02dh MAE %7.2f RMSE %7.2f bias %7.2f % (j, ref['%02d' % j]['mae'],
                      ref['%02d' % j]['rmse'], ref['%02d' % j]['bias']))
out['single_version_reference'] = ref

os.makedirs(os.path.join(ROOT, 'paper_output_pro', 'experiments'), exist_ok=True)
p = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q3-fc-accuracy_B.json')
json.dump(out, open(p, 'w', encoding='utf-8'), ensure_ascii=False, indent=1)
print("\nwrote", p)

if __name__ == '__main__':
    main()

```

D.2.38 支撑材料/代码/03_问题三/q3_fc_diag_A.py

-*- coding: utf-8 -*-

```

"""E1/E2 机制诊断: 为什么融合方案的偏差费用更低?"""
import os, sys, json
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q3_forecast_A import ForecastEngine, STAGE_SLOTS

inp = json.load(open(os.path.join('paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
eng = ForecastEngine(os.path.join('paper_output_pro', 'code', 'q3_vintages_A.json'), inp['pv_actual'], 60)
d0, d1 = 31, 365

```

```

act = np.asarray(inp['pv_actual'], float)[d0:d1]
out = {}
for scheme in ('newest', 'invvar', 'invvar_tod', 'expdecay'):
    PV, LE, NV = eng.all_days(scheme, 'strict')

    P = PV[d0:d1]

    rec = {}

    rec['mean_stage0'] = float(P[:, 0].mean())
    rec['bias_stage0'] = float((P[:, 0] - act).mean())
    rec['mae_stage0'] = float(np.abs(P[:, 0] - act).mean())
    rec['mean_effective_lead_stage0'] = float(LE[d0:d1, 0].mean())
    for si, h in enumerate((6, 12, 18), start=1):
        d = P[:, si] - P[:, 0]
        rec['rev_%d_meanabs' % h] = float(np.abs(d).mean()) # 相对 0:00 计划的修订幅度
        rec['rev_%d_meanabs_act_seg' % h] = float(np.abs(d[:, h * 6:]).mean())
        rec['mae_stage%d' % h] = float(np.abs(P[:, si] - act).mean())
    out[scheme] = rec
for k, v in out.items():
    print(k)
    for kk, vv in v.items():
        print(' %-26s %10.2f % (kk, vv))

```

偏差费用的时间分布

```

for name, p in (('newest', 'q3-fc-e2/newest'), ('invvar', 'q3-fc-e1/invvar')):
    z = np.load(os.path.join('paper_output_pro', 'experiments', p, 'trajectory.npz'))
    p0 = z['plan0']; pf = z['plan']; e = z['E']
    price = np.asarray(inp['price'], float); dt = 1.0 / 6
    dev = (0.5 * price * np.maximum(0, p0 - pf) + 1.5 * price * np.maximum(0, pf - p0)) * dt
    em = 5 * price * e * dt
    print('%s: dev 总 %.0f em 总 %.0f % (name, dev.sum(), em.sum()))
    print(' dev 按小时: ' + ''.join('%d: %.0f % (h, dev[:, h * 6:(h + 1) * 6].sum()) for h in range(0, 24, 2)))
    print(' em 按小时: ' + ''.join('%d: %.0f % (h, em[:, h * 6:(h + 1) * 6].sum()) for h in range(0, 24, 2)))
    print(' 月均 dev: ' + ''.join('%d: %.0f % dev[d * 30:(d + 1) * 30].sum() for d in range(11)))

```

D.2.39 支撑材料/代码/03_问题三/q3_fc_export_A.py

-*- coding: utf-8 -*-

```

"""导出 E1/E2 对比表 (xlsx) 与逐窗口明细 (csv)."""
import os, json, csv
import numpy as np
from openpyxl import Workbook

R = os.path.join('paper_output_pro', 'experiments')
res = json.load(open(os.path.join(R, 'q3-fc-e1e2_all.json'), encoding='utf-8'))

wb = Workbook(); wb.remove(wb.active)

def sheet(name, header, rows):
    ws = wb.create_sheet(name)
    ws.append(header)
    for r in rows:
        ws.append(r)

```

```

BASE = res['baseline_total']; BS = res['baseline_split']
prim = res['e1'] + res['e2']
sheet('1-主结果', ['方案', '总费用(元)', '成交购电', '偏差费用', '紧急购电', '紧急 kwh', '弃电 kwh',
                  '与基准差(元)', '与基准差(%)', '基准总费用'],

[[r['run'], r['total'], r['purchase'], r['dev'], r['em'], r['em_kwh'], r['curtail_kwh'],
  r['diff_vs_base'], r['pct_vs_base'], BASE] for r in prim])

sheet('2-调整罚金诊断', ['方案', '偏差/成交', '偏差/总费用', '调整段数', '总段数', '调整频率',
                          '多买段数', '少买段数', '未改段数', '调整窗口数', '字面口径净收益和',
                          '字面口径负净收益窗口占比', '结算口径边际净收益和', '结算口径负净收益窗口占比'],

[[r['run'], r['dev_over_purchase'], r['dev_over_total'], r['n_adjusted'], r['n_segments'],
  r['adjusted_share'], r['n_over_buy'], r['n_under_buy'], r['n_unchanged'], r['n_windows'],
  r['window_net_sum'], r['window_net_negative_share'], r['window_net_settle_sum'],
  r['window_net_settle_negative_share']] for r in prim])

wr = []
for r in prim:
    for h in ('6', '12', '18'):
        v = r['window_by_hour'].get(h)
        if not v:
            continue
        wr.append([r['run'], h + ':00', v['n'], v.get('down_kwh_sum', 0), v.get('up_kwh_sum', 0),
                  v.get('base_delta_sum', 0), v.get('dev_delta_sum', 0), v.get('net_settle_sum', 0),
                  v.get('net_settle_negative_share', 0), v.get('n_up_sum', 0), v.get('n_down_sum', 0),
                  v.get('n_same_sum', 0)])

sheet('3-窗口分解', ['方案', '窗口', '窗口数', '下调 kwh', '上调 kwh', 'base 变化(元)', '偏差变化(元)',
                    '结算口径边际净收益(元)', '负值窗口占比', '多买段', '少买段', '未改段'], wr)

ab = []
for fam, d in res['ablation'].items():
    rs = d['rows']; mg = d['marginal']
    ab.append([fam, '关闭全部调整', rs[0]['total']] + [",", ""])
    ab.append([fam, '+6:00', rs[1]['total'], mg[0]['marginal_benefit'], ", "])
    ab.append([fam, '+6:00+12:00', rs[2]['total'], ", ", mg[1]['marginal_benefit'], ""])
    ab.append([fam, '+6:00+12:00+18:00', rs[3]['total'], ", ", mg[2]['marginal_benefit']])

sheet('4-消融边际价值', ['方案族', '启用的调整窗口', '全年总费用(元)', '6:00 窗口边际收益(元)',
                        '12:00 窗口边际收益(元)', '18:00 窗口边际收益(元)'], ab)

fl = res.get('forecast_layer_accuracy') or {}
acc = []
for r in prim:
    a = r['acc']
    acc.append([r['run'], a['mae'], a['rmse'], a['mbe'], a['mean_effective_lead_h']])

sheet('5-预报精度-方案', ['方案', 'MAE(kW)', 'RMSE(kW)', 'MBE(kW)', '平均有效提前量(h)'], acc)
raw = fl.get('raw_version_by_lead', {})
ks = sorted(raw.keys(), key=int)
sheet('5b-预报精度-按提前量', ['提前量(h)'] + ks,
      [['MAE(kW)'] + [raw[k]['mae'] for k in ks], ['RMSE(kW)'] + [raw[k]['rmse'] for k in ks]])

rv = []
for k in [x for x in fl.keys() if '|' in x]:
    r_ = fl[k]
    rv.append([k, r_['overall']['mae'], r_['overall']['rmse'], r_['overall']['mbe'],
              r_.get('revision_vs_stage0', {}).get('6', {}).get('mean_abs_daytime'),
              r_.get('revision_vs_stage0', {}).get('12', {}).get('mean_abs_daytime'),
              r_.get('stage_block_mae', {}).get('0'), r_.get('stage_block_mae', {}).get('6'),
              r_.get('stage_block_mae', {}).get('12'), r_.get('stage_block_mae', {}).get('18')])

sheet('5c-预报精度-机制', ['方案|边界', '投用 MAE', '投用 RMSE', '投用 MBE',
                          '6:00 修订幅度(白天)', '12:00 修订幅度(白天)', '0:00 块 MAE',
                          '6:00 块 MAE', '12:00 块 MAE', '18:00 块 MAE'], rv)

sheet('6-标定口径对比', ['标定口径', '总费用', '与基准差', '差%', '偏差费用', '紧急费用', '投用 MAE', '投用 RMSE'],
      [[r['run'], r['total'], r['diff_vs_base'], r['pct_vs_base'], r['dev'], r['em'],
        r['acc']['mae'], r['acc']['rmse']]

```

```

for r in res['sensitivity_cal_window'] + res.get('refs', []))
ci = fl.get('calibration_jan')
if ci:
    ks = [str(k) for k in range(1, 25)]
    sheet('7-1 月标定明细',
        ['提前量档(h)'] + ks,
        [['10 分钟样本'] + [ci['bins'][k]['n_seg'] for k in ks],
        ['小时级独立样本'] + [ci['bins'][k]['n_hours'] for k in ks],
        ['MSE(kW^2)'] + [round(ci['bins'][k]['mse'], 2) for k in ks],
        ['sigma(kW)'] + [round(ci['bins'][k]['sigma'], 2) for k in ks],
        ['分箱合并组', '每档独立(无合并)', ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ],
        ['标定说明', ci['desc']],
        ['标定样本', '版本 %d 个 / 10 分钟样本 %d / 小时级独立样本 %d' % (ci['n_versions'], ci['n_samples_segments'],
ci['n_samples_hours'])]])
    sheet('7b-全样本标定 【含泄露】 ', ['提前量档(h)'] + ks,
        [['10 分钟样本'] + [fl['calibration_fullsample_LEAKY']['bins'][k]['n_seg'] for k in ks],
        ['sigma(kW)'] + [round(fl['calibration_fullsample_LEAKY']['bins'][k]['sigma'], 2) for k in ks],
        ['标注', '含泄露, 仅供对照, 不得作为正式结果'])

x1 = os.path.join(R, 'q3-fc-e1e2_compare.xlsx')
wb.save(x1)
print('->', x1)

```

逐窗口明细 CSV

```
import shutil
sd = {r['dir']: r['run'] for r in prim}
for rel, nm in (('q3-fc-e1/invvar', 'invvar'), ('q3-fc-e2/newest', 'newest'),
               ('q3-fc-e1/expdecay', 'expdecay')):
    o = json.load(open(os.path.join(R, rel, 'metrics.json'), encoding='utf-8'))
    w = o['windows']
    p = os.path.join(R, rel, 'windows.csv')
    with open(p, 'w', newline="", encoding='utf-8-sig') as fh:
        c = csv.writer(fh)
        c.writerow(['day', 'date', 'hour', 'down_kwh', 'up_kwh', 'base_before', 'base_after',
                    'dev_before', 'dev_after', 'net_settle', 'gross_buy_saving', 'dev_level', 'net_literal',
                    'n_up', 'n_down', 'n_same'])
    import datetime
    for x in w:
        dt = (datetime.date(2025, 1, 1) + datetime.timedelta(days=x['day'])).isoformat()
        c.writerow([x['day'], dt, x['hour'], round(x['down_kwh'], 3), round(x['up_kwh'], 3),
                    round(x['base_before'], 2), round(x['base_after'], 2),
                    round(x['dev_before'], 2), round(x['dev_after'], 2),
                    round(x['net_settle'], 2), round(x['buy_saving'], 2),
                    round(x['dev_cost'], 2), round(x['net'], 2),
                    x['n_up'], x['n_down'], x['n_same']])
    print('-', p, len(w), 'windows')
```

D.2.40 支撑材料/代码/03 问题三/q3 fc leaktest A.py

-*- coding: utf-8 -*-

""数据泄露审计 (独立于预报层实现): 扰动"未来"数据, 早于该时刻的预报/计划必须不变.

```

审计项
A. 扰动第 D 天的版本值 -> 第 d<D 天所有阶段预报必须逐位不变
B. 扰动第 D 天 6/12/18 版 -> 第 D 天 0:00 阶段预报必须不变 (0:00 决策时它们尚未发布)
C. 扰动第 D 天 18 版 -> 第 D 天 6:00/12:00 阶段预报必须不变
D. 扰动第 D 天 0:00 版对 t=0 段无影响 (0:00-0:10 不得使用当天 0:00 版)
E. 扰动未来实际光伏值 -> 第 d<D 天预报必须不变 (实际值只进历史标定)
"""
import os, sys, json
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q3_forecast_A import ForecastEngine, STAGE_SLOTS, SPD

RNG = np.random.default_rng(20260912)
JAN = 31 # 硬约束: 方差只用第一个月标定
INP = os.path.join('paper_output_pro', 'data_cleaned', 'q3_input.json')
VIN = os.path.join('paper_output_pro', 'code', 'q3_vintages_A.json')

class Eng(ForecastEngine):
    def __init__(self, H, act, cal=60):
        with open(VIN, encoding='utf-8') as f:
            d0 = json.load(f)
        self.H = H.copy(); self.D, self.K, self.NH = self.H.shape
        self.T = SPD; self.ND = self.D * self.T
        self.act = act.copy(); self.cal_window = cal; self.sigma0 = 1.0
        self.calib = 'fixed_jan'; self.min_bin_n = 20
        self._build_candidates(); self._calibrate()

inp = json.load(open(INP, encoding='utf-8'))
base_H = np.asarray(json.load(open(VIN, encoding='utf-8'))['vintages'], float)
act = np.asarray(inp['pv_actual'], float).reshape(-1)
checks = []

def cmp(name, A, B, expect_same):
    same = np.array_equal(A, B)
    ok = (same == expect_same)
    checks.append((name, ok, same))
    print('%-4s %-72s 相同=%s 期望=%s' % ('PASS' if ok else 'FAIL', name, same, expect_same))
    return ok

for scheme in ('newest', 'invvar', 'expdecay'):
    D = 200
    e0 = Eng(base_H, act); P0 = e0.all_days(scheme, 'strict')[0]
    # A/B: 扰动第 200 天 6/12/18 版
    H = base_H.copy(); H[D, 1:, :] = 9999.0 * RNG.random((3, 24))
    e1 = Eng(H, act); P1 = e1.all_days(scheme, 'strict')[0]
    cmp(['%s] A 未来日(D=200)版本改动不影响 d<200' % scheme,
        P0[:D], P1[:D], True)
    cmp(['%s] A2 未来日版本改动确实改变了 d=200' % scheme,
        P0[D], P1[D], False)
    cmp(['%s] B 6/12/18 版改动不影响 d=200 的 0:00 阶段' % scheme,
        P0[D, 0], P1[D, 0], True)
    # C: 只扰动 18:00 版
    H2 = base_H.copy(); H2[D, 3, :] = 9999.0 * RNG.random(24)
    e2 = Eng(H2, act); P2 = e2.all_days(scheme, 'strict')[0]
    cmp(['%s] C 18:00 版改动不影响 d=200 的 6:00/12:00 阶段' % scheme,
        P0[D, :3, :108], P2[D, :3, :108], True)

```

```
# D: 只扰动当天 0:00 版, 检查 t=0 段
H3 = base_H.copy(); H3[D, 0, :] = 9999.0
e3 = Eng(H3, act); P3 = e3.all_days(scheme, 'strict')[0]
cmp(['%s] D 当天 0:00 版改动不影响 d=200 的 t=0 段' % scheme,
    P0[D, :, 0:1], P3[D, :, 0:1], True)
cmp(['%s] D2 当天 0:00 版改动影响 d=200 的 t>=1 段' % scheme,
    P0[D, :, 1:37], P3[D, :, 1:37], False)
# E: 扰动未来实际值
act2 = act.copy(); act2[300 * SPD:] *= 3.0 + RNG.random(act2[300 * SPD:].size)
e4 = Eng(base_H, act2); P4 = e4.all_days(scheme, 'strict')[0]
cmp(['%s] E 未来实际值改动不影响 d<300 预报' % scheme, P0[:300], P4[:300], True)
# F/G: 硬约束专项 —— 1 月以外的实际值一律不得影响方差标定/权重/预报
act3 = act.copy(); act3[JAN * SPD:] *= 2.5 + RNG.random(act3[JAN * SPD:].size)
e5 = Eng(base_H, act3); P5 = e5.all_days(scheme, 'strict')[0]
cmp(['%s] F 2 月及以后实际值改动不影响全年预报(1 月标定隔离)' % scheme, P0, P5, True)
cmp(['%s] G 2 月及以后实际值改动不影响标定结果(逐档 sigma)' % scheme,
    e0.mse[0], e5.mse[0], True)
```

H: 标定结果必须在 365 天上逐位相同（标定一次、固定全年，不得滚动）

```
for scheme in ('invvar',):
    e = Eng(base_H, act)
    h_ok = all(np.array_equal(e.mse[0], e.mse[d]) for d in range(e.D))
    h_ok &= all(np.array_equal(e.mse_tod[0], e.mse_tod[d]) for d in range(e.D))
    checks.append(('H 标定结果在全部 365 天逐位相同(固定不滚动)', h_ok, h_ok))
    print('%-4s %-72s 相同=%s 期望=True' % ('PASS' if h_ok else 'FAIL',
        '%s] H 标定结果在全部 365 天逐位相同(固定不滚动)' % scheme, h_ok))

bad = [c for c in checks if not c[1]]
print('\n 审计结论: %d 项全部通过' % len(checks) if not bad else '\n 审计结论: %d 项失败' % len(bad))
sys.exit(1 if bad else 0)
```

D.2.41 支撑材料/代码/03_问题三/q3_fc_leaktest_B.py

-*- coding: utf-8 -*-

"""数据泄露检验 (E3/E4 预报层)。

检验三件事:

- L1. 取版因果性: 对每个 (天, 时段, 决策时刻) 用到的成员, 断言 $pub \leq dec$ 。
- L2. σ_{hat} 因果性: 把第 X 天及以后的实际光伏置零后重算, 断言 $sig[d]$ ($d \leq X$) 完全不变。
- L3. 端到端: 用同一份“未来实际值被破坏”的数据重跑求解器, 断言 $d < X$ 的计划/调整量完全不变。

由外部两次求解 (q3_fc_solver_B.py --stop-day X [--replace-actual-from X]) 生成的 solution.json 比对。

```
"""
import os, sys, json
import numpy as np

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from q3_forecast_B import (load_vintages, residuals, rolling_sigma, forecast_stage, # noqa: E402
    fixed_sigma, full_sample_sigma, calibration_residuals,
    candidates, CAL_PUB, CAL_TGT, NDAY, T)
```

```

STAGES = [(0, 0), (6, 36), (12, 72), (18, 108)]
X = 60

def main():
    V, _ = load_vintages(os.path.join(HERE, 'q3_vintages_B.json'))
    inp = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
    pv_a = np.asarray(inp['pv_actual'], float)

    # ---- L1 ----
    bad = 0; tot = 0
    for d in range(NDAY):
        for off, slot in STAGES:
            dec = d * 24 + off
            for t in range(slot, T):
                for pub, j, val in candidates(V, d, t, dec, 'all'):
                    tot += 1
                    if pub > dec:
                        bad += 1
    print('L1 取版因果性: 检查 %d 个成员, 违反 pub<=dec 的 = %d -> %s' % (tot, bad, 'PASS' if bad == 0 else 'FAIL'))
    assert bad == 0

    # ---- L2: 1 月标定窗口的样本边界 ----
    samp = calibration_residuals(V, pv_a, CAL_PUB, CAL_TGT)
    print('L2 标定窗口: 发布日期区间 %s, 目标日区间 %s, 样本 %d' % (CAL_PUB, CAL_TGT, len(samp)))
    ha = np.asarray(pv_a, float).reshape(NDAY, 24, 6).mean(axis=2)
    bad_t, bad_p = 0, 0
    for d0 in range(NDAY):
        for h0i, h0 in enumerate((0, 6, 12, 18)):
            for j in range(24):
                a = d0 * 24 + h0 + j
                if a >= NDAY * 24:
                    continue
                td = a // 24
                inside = (CAL_PUB[0] <= d0 < CAL_PUB[1]) and (CAL_TGT[0] <= td < CAL_TGT[1])
                # 1 月 31 日 18:00 版覆盖到 2 月 1 日 -> 目标日超出窗口, 必须被剔除
                if (td >= CAL_TGT[1]) and (d0 < CAL_PUB[1]):
                    bad_t += 1
                if inside:
                    bad_p += 1
    print('L2b 目标日越界的可剔除样本数 = %d (已剔除, 未进入统计)' % bad_t)
    sig_row, info = fixed_sigma(V, pv_a)
    print('L2c 每档样本数 n = %s' % info['n_by_lead'])
    print('L2d 每档 n >= 20 = %s (最小 %d)' % (min(info['n_by_lead']) >= 20, min(info['n_by_lead'])))
    assert min(info['n_by_lead']) >= 20

    # 破坏标定窗以外的全部实际值, sigma 必须完全不变

```

```

pv_bad = pv_a.copy(); pv_bad[CAL_TGT[1]:] = 0.0
sig_bad_row, _ = fixed_sigma(V, pv_bad)
ok = np.allclose(sig_row, sig_bad_row)
print('L2e sigma 因果性: 破坏第 %d 天(2 月 1 日)起全部实际值后, 1 月标定 sigma 完全一致 = %s'
      % (CAL_TGT[1] + 1, 'PASS' if ok else 'FAIL'))
assert ok

sig_fixed = np.tile(sig_row, (NDAY, 1))
sig_bad = np.tile(sig_bad_row, (NDAY, 1))

# ---- 预报数组必须与「标定窗以后的实际值」无关 ----
same = True
for d in range(NDAY):
    for off, slot in STAGES:
        a1 = forecast_stage(V, sig_fixed, d, d * 24 + off, 'inv', 'all')[0]
        a2 = forecast_stage(V, sig_bad, d, d * 24 + off, 'inv', 'all')[0]
        if not np.allclose(np.nan_to_num(a1, nan=-1), np.nan_to_num(a2, nan=-1)):
            same = False
print('L3 预报层: 破坏 2 月起全部实际值后, 全年 365 天的逆方差预报值完全一致 = %s'
      % ('PASS' if same else 'FAIL'))
assert same

# ---- L4 端到端 solution.json 比对 ----
if len(sys.argv) >= 4:
    A = json.load(open(sys.argv[1], encoding='utf-8'))
    B = json.load(open(sys.argv[2], encoding='utf-8'))
    n = int(sys.argv[3])
    mx = 0.0
    n = min([n] + [len(A[k]) for k in ('plan0', 'plan', 'c', 'd', 'E')]
            + [len(B[k]) for k in ('plan0', 'plan', 'c', 'd', 'E')])
    for k in ('plan0', 'plan', 'c', 'd', 'E'):
        for i in range(n):
            mx = max(mx, float(np.max(np.abs(np.array(A[k][i]) - np.array(B[k][i])))))
print('L4 端到端: 计分日(第 31..59 天, 共%d 天)的 plan0/plan/c/d/E 最大绝对差 = %.3e -> %s'
      % (n, mx, 'PASS' if mx == 0.0 else 'FAIL'))

if __name__ == '__main__':
    main()

```

D.2.42 支撑材料/代码/03_问题三/q3_fc_report_A.py

-*- coding: utf-8 -*-

```

"""汇总 E1 / E2 全部运行 -> experiments/q3-fc-e1/e2/metrics.json + 对比表(md/xlsx)."""
import os, sys, json, argparse
import numpy as np

BASE = 14483771.06
BASE_SPLIT = dict(purchase=13537826.50, dev=646347.91, em=299596.65)

E1_RUNS = [('E1a invvar(逆方差加权: 按提前量单维标定, 规范字面口径)', 'q3-fc-e1/invvar'),
            ('E1b invvar_tod(逆方差加权: 目标时刻 x 提前量双维标定)', 'q3-fc-e1/invvar_tod'),

```

('E1-control expdecay(指数衰减 I=0.5)', 'q3-fc-e1/expdecay')]

非正式口径对照 / 稳健性(均不作为正式结果)

```
REFS = [('REF invvar|全样本标定【含泄露,仅供对照】', 'q3-fc-e1/ref-invvar-fullsample'),
        ('REF invvar|滚动 60 天标定【非正式口径】', 'q3-fc-e1/sens-cal60')]

E2_RUNS = [('E2-primary newest(取最新版)', 'q3-fc-e2/newest'),
            ('E2-boundary newest|adjust', 'q3-fc-e2/newest_adjust'),
            ('E2-boundary newest|full(=基准)', 'q3-fc-e2/newest_full')]

ABL = {'E1 invvar': [('none', 'q3-fc-e1/ablation-none'), ('6', 'q3-fc-e1/ablation-s6'),
                    ('6+12', 'q3-fc-e1/ablation-s612'), ('6+12+18', 'q3-fc-e1/invvar')],
       'E1 expdecay': [('none', 'q3-fc-e1/ablation-exp-none'), ('6', 'q3-fc-e1/ablation-exp-s6'),
                       ('6+12', 'q3-fc-e1/ablation-exp-s612'), ('6+12+18', 'q3-fc-e1/expdecay')],
       'E2 newest': [('none', 'q3-fc-e2/ablation-none'), ('6', 'q3-fc-e2/ablation-s6'),
                    ('6+12', 'q3-fc-e2/ablation-s612'), ('6+12+18', 'q3-fc-e2/newest')]}

SENS = [('滚动 30 天【非正式口径】', 'q3-fc-e1/sens-cal30'),
        ('滚动 60 天【非正式口径】', 'q3-fc-e1/sens-cal60'),
        ('滚动 90 天【非正式口径】', 'q3-fc-e1/sens-cal90'),
        ('滚动 180 天【非正式口径】', 'q3-fc-e1/sens-cal180'),
        ('第一个月固定标定【正式口径】', 'q3-fc-e1/invvar')]
```

```
def load(root, rel):
    p = os.path.join(root, rel, 'metrics.json')
    return json.load(open(p, encoding='utf-8')) if os.path.exists(p) else None

def row(name, rel, o):
    m = o['metrics']; d = m['diag']
    return dict(run=name, dir=rel.replace('/', os.sep),
               scheme=m['config']['scheme'], boundary=m['config']['boundary'],
               stages=m['config']['adjust_stages'], cal_window=m['config']['cal_window'],
               total=m['q3_total_cost'], purchase=m['q3_purchase_cost'],
               dev=m['q3_adjustment_cost'], em=m['q3_emergency_cost'],
               em_kwh=m['q3_emergency_kwh'], curtail_kwh=m['q3_curtail_kwh'],
               diff_vs_base=m['q3_total_cost'] - BASE,
               pct_vs_base=(m['q3_total_cost'] - BASE) / BASE * 100.0,
               dev_over_purchase=d['dev_over_purchase'], dev_over_total=d['dev_over_total'],
               n_segments=d['n_segments'], n_adjusted=d['n_adjusted_segments'],
               adjusted_share=d['adjusted_share'], n_over_buy=d['n_over_buy'],
               n_under_buy=d['n_under_buy'], n_unchanged=d['n_unchanged'],
               n_windows=d['n_windows'],
               window_net_sum=d.get('window_net_sum'),
               window_net_negative_share=d.get('window_net_negative_share'),
               window_net_settle_sum=d.get('window_net_settle_sum'),
               window_net_settle_negative_share=d.get('window_net_settle_negative_share'),
               em_cost_by_block=d.get('emergency_cost_by_stage_block'),
               em_kwh_by_block=d.get('emergency_kwh_by_stage_block'),
               em_kwh_changed=d['emergency_kwh_changed_seg'],
               em_kwh_unchanged=d['emergency_kwh_unchanged_seg'],
               window_by_hour=d['window_net_by_hour'],
               calib=m['config'].get('calib'), min_bin_n=m['config'].get('min_bin_n'),
```

```

        acc=m['forecast_accuracy'])

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--root', default=os.path.join('paper_output_pro', 'experiments'))
    a = ap.parse_args(); R = a.root
    res = {'baseline_total': BASE, 'baseline_split': BASE_SPLIT}
    for tag, lst in (('e1', E1_RUNS), ('e2', E2_RUNS)):
        rows = []
        for nm, rel in lst:
            o = load(R, rel)
            if o is None:
                print('MISSING', rel); continue
            rows.append(row(nm, rel, o))
        res[tag] = rows
    res['ablation'] = {}
    for fam, lst in ABL.items():
        rows = []
        for nm, rel in lst:
            o = load(R, rel)
            if o is None:
                print('MISSING', rel); continue
            r = row(nm, rel, o); rows.append(r)
        marg = []
        for i in range(1, len(rows)):
            h = ('6:00', '12:00', '18:00')[i - 1] if rows[i]['stages'] != 'none' else None
            marg.append(dict(window=h, stages_before=rows[i - 1]['stages'], stages_after=rows[i]['stages'],
                             cost_before=rows[i - 1]['total'], cost_after=rows[i]['total'],
                             marginal_benefit=rows[i - 1]['total'] - rows[i]['total']))
        res['ablation'][fam] = dict(rows=rows, marginal=marg)
    res['refs'] = []
    for nm, rel in REFS:
        o = load(R, rel)
        if o is None:
            print('MISSING', rel); continue
        res['refs'].append(row(nm, rel, o))
    res['sensitivity_cal_window'] = []
    for nm, rel in SENS:
        o = load(R, rel)
        if o is None:
            print('MISSING', rel); continue
        res['sensitivity_cal_window'].append(row(nm, rel, o))
    fa = os.path.join(R, 'q3-fc-e1', 'forecast_accuracy.json')
    res['forecast_layer_accuracy'] = json.load(open(fa, encoding='utf-8')) if os.path.exists(fa) else None

    os.makedirs(os.path.join(R, 'q3-fc-e1'), exist_ok=True)
    os.makedirs(os.path.join(R, 'q3-fc-e2'), exist_ok=True)

```

```

json.dump({'baseline_total': BASE, 'baseline_split': BASE_SPLIT,
          'primary': res['e1'] + res['e2'],
          'ablation': res['ablation'], 'sensitivity_cal_window': res['sensitivity_cal_window']},
open(os.path.join(R, 'q3-fc-e1', 'metrics.json'), 'w', encoding='utf-8'),
ensure_ascii=False, indent=1)
json.dump({'baseline_total': BASE, 'baseline_split': BASE_SPLIT,
          'primary': res['e2'], 'ablation': res['ablation']['E2 newest'],
          'boundary_variants': res['e2']},
open(os.path.join(R, 'q3-fc-e2', 'metrics.json'), 'w', encoding='utf-8'),
ensure_ascii=False, indent=1)
json.dump(res, open(os.path.join(R, 'q3-fc-e1e2_all.json'), 'w', encoding='utf-8'),
ensure_ascii=False, indent=1)
write_md(R, res)
print('done ->', R)

def write_md(R, res):
    L = []
    L.append('# 问题三 · 预报版本实验 E1 / E2 汇总 (因子 A = 0:00 不知) \n')
    L.append('基准 (现行做法, = E4: 0:00 版覆盖全天 + 6/12/18 切新版) = **%.2f 元**'
            ' (成交 %.2f + 偏差 %.2f + 紧急 %.2f) \n' % (BASE, BASE_SPLIT['purchase'],
            BASE_SPLIT['dev'], BASE_SPLIT['em']))
    L.append('## 1. 主结果\n')
    L.append('| 方案 | 总费用 | 成交 | 偏差 | 紧急 | 与基准差 | 差% |')
    L.append('|---|---|---|---|---|---|---|')
    for r in res['e1'] + res['e2']:
        L.append('| %s | %.2f | %.2f | %.2f | %.2f | %+.2f | %+.4f%% |'
                % (r['run'], r['total'], r['purchase'], r['dev'], r['em'],
                r['diff_vs_base'], r['pct_vs_base']))
    L.append("\n## 2. 调整罚金诊断 (规范第四节) \n")
    L.append('| 方案 | 偏差/成交 | 调整段/总段 | 调整频率 | 多买段 | 少买段 | 未改段 | 窗口数 |'
            '字面净收益和 | 字面负占比 | 结算边际净收益和 | 结算负占比 |')
    L.append('|' + '---'| * 12)
    for r in res['e1'] + res['e2']:
        L.append('| %s | %.4f | %d/%d | %.4f | %d | %d | %d | %.2f | %.4f | %.2f | %.4f |'
                % (r['run'], r['dev_over_purchase'], r['n_adjusted'], r['n_segments'],
                r['adjusted_share'], r['n_over_buy'], r['n_under_buy'], r['n_unchanged'],
                r['n_windows'], r['window_net_sum'], r['window_net_negative_share'] or 0,
                r['window_net_settle_sum'] or 0, r['window_net_settle_negative_share'] or 0))
    L.append("\n### 2.1 按调整窗口分解 (结算口径的边际净收益 = 本窗口调整前后 base+dev 之差) \n")
    L.append('| 方案 | 窗口 | 窗口数 | 下调幅度 kWh | 上调幅度 kWh | base 变化 | 偏差变化 | 边际净收益和 | 负值窗口'
            '占比 |')
    L.append('|' + '---'| * 9)
    for r in res['e1'] + res['e2']:
        for h in ('6', '12', '18'):
            v = r['window_by_hour'].get(h)
            if not v:
                continue
            L.append('| %s | %s:00 | %d | %.1f | %.1f | %.2f | %.2f | %.2f | %.4f |'

```

```

        % (r['run'], h, v['n'], v.get('down_kwh_sum', 0), v.get('up_kwh_sum', 0),
          v.get('base_delta_sum', 0), v.get('dev_delta_sum', 0),
          v.get('net_settle_sum', 0), v.get('net_settle_negative_share', 0)))
L.append("\n### 2.2 调整窗口真实边际价值（启用/关闭该窗口的全年费用差，正=有用）\n")
L.append('| 方案族 | 关闭全部调整 | +6:00 | +6:00+12:00 | +6:00+12:00+18:00 | 6:00 窗口边际 | 12:00 窗口边际 | 18:00 窗口边际 |')
L.append('|' + '---' * 8)
for fam, d in res['ablation'].items():
    rows = d['rows']; m = d['marginal']
    L.append('| %s | %.2f | %.2f | %.2f | %.2f | %.2f | %.2f | %.2f |'
              % (fam, rows[0]['total'], rows[1]['total'], rows[2]['total'], rows[3]['total'],
                 m[0]['marginal_benefit'], m[1]['marginal_benefit'], m[2]['marginal_benefit']))
L.append("\n### 2.3 紧急购电按决策阶段块归属（元）\n")
L.append('| 方案 | 0:00 计划段(0-6 时) | 6:00 调整段(6-12 时) | 12:00 调整段(12-18 时) | 18:00 调整段(18-24 时) |')
L.append('|' + '---' * 5)
for r in res['e1'] + res['e2']:
    b = r['em_cost_by_block'] or {}
    L.append('| %s | %.0f | %.0f | %.0f | %.0f |' % (r['run'], b.get('0', 0), b.get('6', 0),
                                                         b.get('12', 0), b.get('18', 0)))
L.append("\n## 3. 光伏预报精度（统计期 d=31..364，实际投用的预报）\n")
L.append('| 方案 | MAE | RMSE | MBE | 平均有效提前量(h) |')
L.append('|' + '---' * 5)
for r in res['e1'] + res['e2']:
    ac = r['acc']
    L.append('| %s | %.2f | %.2f | %.2f | %.2f |' % (r['run'], ac['mae'], ac['rmse'],
                                                         ac['mbe'], ac['mean_effective_lead_h']))
L.append("\n### 3.1 按决策提前量分档 MAE（kW）\n")
fl = res.get('forecast_layer_accuracy') or {}
keys = [k for k in fl.keys() if 'l' in k]
if keys:
    leads = sorted({int(x) for k in keys for x in fl[k]['by_decision_lead'].keys()})
    L.append('| 方案 | ' + ' | '.join("%dh" % x for x in leads) + ' |')
    L.append('|' + '---' * (len(leads) + 1))
    for k in keys:
        row_ = [fl[k]['by_decision_lead'].get(str(x), {}).get('mae') for x in leads]
        L.append('| %s | ' % k + ' | '.join("%.0f" % v if v is not None else '-' for v in row_) + ' |')
    L.append("\n### 3.2 附件 3 原始各版本按提前量分档（统计期 d=31..364，全部 4 版合并，kW）\n")
    raw = fl.get('raw_version_by_lead', {})
    have = sorted(raw.keys(), key=int)
    L.append('| 提前量(h) | ' + ' | '.join(have) + ' |')
    L.append('|' + '---' * (len(have) + 1))
    L.append('| MAE | ' + ' | '.join("%.0f" % raw[k]['mae'] for k in have) + ' |')
    L.append('| RMSE | ' + ' | '.join("%.0f" % raw[k]['rmse'] for k in have) + ' |')
    L.append("\n### 3.3 相对 0:00 计划的修订幅度与分块精度（机制量）\n")
    L.append('| 方案 | 6:00 修订(白天,kW) | 12:00 修订(白天,kW) | 0:00 块 MAE | 6:00 块 MAE | 12:00 块 MAE | 18:00 块 MAE |')
    L.append('|' + '---' * 7)

```

```

for k in keys:
    rv = fl[k].get('revision_vs_stage0', {}); sm = fl[k].get('stage_block_mae', {})
    L.append('| %s | %.1f | %.1f | %.1f | %.1f | %.1f |'
            % (k, rv.get('6', {}).get('mean_abs_daytime', 0),
              rv.get('12', {}).get('mean_abs_daytime', 0),
              sm.get('0', 0), sm.get('6', 0), sm.get('12', 0), sm.get('18', 0)))
L.append("\n### 3.4 E1 两版加权口径并列表 (1月固定窗标定) \n")
L.append('| 版本 | 标定维度 | 总费用 | 与基准差 | 差% | 投用 MAE | 投用 RMSE | 平均有效提前量 | 6:00 修订幅度 (白天) |')
L.append('|' + '-' * 9)
_e1 = {r['run']: r for r in res['e1']}
for nm, key, dim in (('E1a invvar', 'E1a', '仅提前量 Δ (24 档)'),
                    ('E1b invvar_tod', 'E1b', '目标时刻 × 提前量 Δ (24×24 档)')):
    r = next((v for k, v in _e1.items() if k.startswith(key)), None)
    if r is None:
        continue
    flk = r['scheme'] + '|strict|jan'
    rv = (fl.get(flk, {}) or {}).get('revision_vs_stage0', {}).get('6', {}).get('mean_abs_daytime')
    L.append('| %s | %s | %.2f | %+.2f | %+.4f%% | %.2f | %.2f | %.2f h | %s |'
            % (nm, dim, r['total'], r['diff_vs_base'], r['pct_vs_base'],
              r['acc']['mae'], r['acc']['rmse'], r['acc']['mean_effective_lead_h'],
              ('%.1f % rv' if rv is not None else '-')))
L.append("\n## 4. 方差标定口径 (硬约束: 只用第一个月, 标定一次固定全年) \n")
ci = (res.get('forecast_layer_accuracy') or {}).get('calibration_jan')
if ci:
    L.append('标定样本: **%d 个版本 (2025-01-01..2025-01-31 的 4 个发布时刻 × 31 天) **, '
            '目标时刻同样限制在 1 月内, 共 **%d 个 10 分钟样本 / %d 个小时级独立样本**;',
            '1 月 31 日各版覆盖到 2 月的目标段已剔除。标定一次后全年固定沿用。 \n'
            % (ci['n_versions'], ci['n_samples_segments'], ci['n_samples_hours']))
    L.append('按提前量 24 档分摊, **每档最小样本 %d (≥ min_bin_n=20), 未触发任何分箱合并**, 24 档各自独立。 \n'
            % min(ci['bins'][str(k)]['n_seg'] for k in range(1, 25)))
    L.append('| 提前量档(h) | ' + ' | '.join(str(k) for k in range(1, 25)) + ' |')
    L.append('|' + '-' * 25)
    L.append('| 10 分钟样本 | ' + ' | '.join(str(ci['bins'][str(k)]['n_seg']) for k in range(1, 25)) + ' |')
    L.append('| 小时级样本 | ' + ' | '.join(str(ci['bins'][str(k)]['n_hours']) for k in range(1, 25)) + ' |')
    L.append('| sigma(kW) | ' + ' | '.join("%.0f % ci['bins'][str(k)]['sigma']" for k in range(1, 25)) + ' |')
    L.append("\n (每版本每档贡献 6 个 10 分钟段 = 1 个小时级独立样本, 故小时级样本数 ≈ 段样本数/6。) \n")
L.append("\n### 4.1 标定口径对比 \n")
L.append('| 标定口径 | 总费用 | 与基准差 | 差% | 偏差费用 | 紧急费用 | 投用 MAE | 投用 RMSE |')
L.append('|' + '-' * 8)
for r in res['sensitivity_cal_window'] + [x for x in res.get('refs', [])]:
    if '全样本' in x['run']:
        L.append('| %s | %.2f | %+.2f | %+.4f%% | %.2f | %.2f | %.2f |'
                % (r['run'], r['total'], r['diff_vs_base'], r['pct_vs_base'], r['dev'], r['em'],
                  r['acc']['mae'], r['acc']['rmse']))
md = "\n".join(L) + "\n"

```

```

open(os.path.join(R, 'q3-fc-e1e2_compare.md'), 'w', encoding='utf-8').write(md)
with open(os.path.join(R, 'q3-fc-e1e2_compare.md'), encoding='utf-8') as fh:
    print(fh.read())

if __name__ == '__main__':
    main()

```

D.2.43 支撑材料/代码/03_问题三/q3_fc_rollperiod.py

-*- coding: utf-8 -*-

```

import sys, io, json
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
E = 'paper_output_pro/experiments/'
def T(p): return json.load(open(p, encoding='utf-8'))['metrics']['q3_total_cost']
p31, p122, p214 = T(E+'q3-fc-e3/metrics.json'), T(E+'_s_sf_pool_122/metrics.json'), T(E+'_s_sf_pool_214/metrics.json')
r31, r122, r214 = T(E+'_s_rr_half/metrics.json'), T(E+'_s_rr_half_SF122/metrics.json'), T(E+'_s_rr_half_SF214/metrics.json')
print('=== 分期稳健性: 扩展窗 eb_half vs 现行固定1月 ===')
print('%-26s %16s %16s %14s' % ('期间', '现行 固定1月', '扩展窗 half', '差额'))
for nm, x, y in (('2月-4月 (第31..121天)', p31-p122, r31-r122),
                 ('5月-7月 (第122..213天)', p122-p214, r122-r214),
                 ('8月-12月 (第214..364天)', p214, r214)):
    print('%-26s %16.2f %16.2f %14.2f' % (nm, x, y, x-y))

print('%-26s %16.2f %16.2f %14.2f' % ('合计', p31, r31, p31-r31))
print()
w = sum(1 for x, y in ((p31-p122, r31-r122), (p122-p214, r122-r214), (p214, r214)) if x-y > 0)
print('扩展窗在 %d/3 期中更优' % w)
seg = [(p31-p122)-(r31-r122), (p122-p214)-(r122-r214), p214-r214]
tot = sum(seg)
print('各期贡献: 2-4月 %.0f 元 (%.0f%%), 5-7月 %.0f 元 (%.0f%%), 8-12月 %.0f 元 (%.0f%%)'
      % (seg[0], 100*seg[0]/tot, seg[1], 100*seg[1]/tot, seg[2], 100*seg[2]/tot))

```

D.2.44 支撑材料/代码/03_问题三/q3_fc_rollsigma.py

-*- coding: utf-8 -*-

扩展窗 $\hat{\sigma}$ 标定（正式口径，问题三 E3v2）

#

严格因果：第 d 天的决策只使用「目标日 $\leq d-1$ 」的残差。

口径： $\hat{\sigma}^2(\text{目标小时, 提前量}) = \frac{1}{2} \cdot s^2(\text{格子}) + \frac{1}{2} \cdot s^2(\text{合并})$
—— 零自由参数。

#

用法: `python paper_output_pro/code/q3_fc_rollsigma.py <mode> <out.npy> [replace_actual_from]`

`mode` $\in$ half(正式) | k0 | k31 | pool

"""扩展窗 σ 标定：第 d 天的决策只使用「目标日 $\leq d-1$ 」的残差（严格因果）。

输出 (365, 24, 24) 的 σ 表，第 d 行 = 第 d 天决策时使用的表。
 用法: `python _s_roll.py <mode> <out.json>`

`mode` $\in$ half | k0 | k31 | pool

"""

```

import sys, io, os, json
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
HERE = os.path.abspath('paper_output_pro/code'); sys.path.insert(0, HERE)
import numpy as np

```

```

from q3_forecast_B import load_vintages, hourly_actual, HOURS, NDAY, NH

MODE = sys.argv[1]; OUT = sys.argv[2]
XF = int(sys.argv[3]) if len(sys.argv) > 3 else 0
V, _ = load_vintages(os.path.join(HERE, 'q3_vintages_B.json'))
pv = np.asarray(json.load(open('paper_output_pro/data_cleaned/q3_input.json', encoding='utf-8'))['pv_actual'], float)
if XF:
    pv = pv.copy(); pv[XF:] = 0.0
ha = hourly_actual(pv)

```

--- 展平: 每个 (发布日 d0, 发布时刻 index h0i, 提前量 j) 一条残差 ---

```

jj = np.zeros((NDAY, 4, NH), int); td = np.zeros((NDAY, 4, NH), int); th = np.zeros((NDAY, 4, NH), int)
R = np.zeros((NDAY, 4, NH))
for d0 in range(NDAY):
    for h0i, h0 in enumerate(HOURS):
        for j in range(NH):
            a = d0*24 + h0 + j
            jj[d0, h0i, j] = j
            td[d0, h0i, j] = a // 24
            th[d0, h0i, j] = a % 24
            R[d0, h0i, j] = (V[d0, h0i, j] - ha[a//24, a % 24]) if a < NDAY*24 else 0.0
valid = (td < NDAY)
Rf = R[valid]; jf = jj[valid]; tdf = td[valid]; thf = th[valid]
cellkey = thf*NH + jf
print('总残差样本 = %d (被边界截断而剔除 %d)' % (Rf.size, R.size - Rf.size))

```

```

TAB = np.full((NDAY, 24, NH), np.nan)
for d in range(NDAY):
    m = tdf <= d - 1 # ★ 因果边界: 只用「目标日 < d」的样本
    if not m.any():
        continue
    r = Rf[m]; kk = cellkey[m]; jm = jf[m]
    # --- 合并方差 (按提前量) ---
    cnt_j = np.bincount(jm, minlength=NH).astype(float)
    s1_j = np.bincount(jm, weights=r, minlength=NH)
    s2_j = np.bincount(jm, weights=r*r, minlength=NH)
    var_pool = np.full(NH, np.nan)
    ok = cnt_j > 1
    var_pool[ok] = (s2_j[ok] - s1_j[ok]**2/cnt_j[ok]) / (cnt_j[ok] - 1)
    sd_pool = np.sqrt(np.maximum(var_pool, 1.0))
    # --- 格子方差 ---
    cnt_c = np.bincount(kk, minlength=24*NH).astype(float)
    s1_c = np.bincount(kk, weights=r, minlength=24*NH)
    s2_c = np.bincount(kk, weights=r*r, minlength=24*NH)
    var_cell = np.full(24*NH, np.nan)
    okc = cnt_c > 1
    var_cell[okc] = (s2_c[okc] - s1_c[okc]**2/cnt_c[okc]) / (cnt_c[okc] - 1)
    var_cell = var_cell.reshape(24, NH); cnt_c = cnt_c.reshape(24, NH)
    # --- 合成 ---
    P2 = np.tile(sd_pool**2, (24, 1))
    if MODE == 'pool':

```

```

    T2 = P2.copy()
elif MODE == 'k0':
    T2 = var_cell.copy()
elif MODE == 'half':
    T2 = (var_cell + P2) / 2.0
elif MODE == 'k31':
    n = cnt_c
    T2 = np.where(n > 0, (n*var_cell + 31.0*P2)/(n + 31.0), P2)
else:
    raise ValueError(MODE)
# 该格还没有样本时退回合并值
T2 = np.where(np.isfinite(T2), T2, P2)
TAB[d] = np.sqrt(np.maximum(T2, 1.0))

print('%s: 表尺寸 %s' % (MODE, TAB.shape))
for d in (31, 32, 60, 120, 181, 300, 364):
    n_eff = int((tdf <= d-1).sum())
    print(' 第 %3d 天: 可用样本 %6d,  $\hat{\sigma}$ (目标 08:00,  $\Delta=2$ ) = %7.1f' % (d, n_eff, TAB[d, 8, 2]))
np.save(OUT.replace('.json', '.numpy'), TAB)
print('已保存', OUT.replace('.json', '.numpy'))

```

D.2.45 支撑材料/代码/03_问题三/q3_fc_rollsolver.py

-*- coding: utf-8 -*-

扩展窗 $\hat{\sigma}$ 求解器（正式口径，问题三 E3v2）

#

用法: python paper_output_pro/code/q3_fc_rollsolver.py
<表.npy> <outdir> [replace_actual_from] [stop_day]
[score_from]

```

"""扩展窗  $\hat{\sigma}$  运行器: python _s_roll_run.py <表.npy> <outdir> [xf] [stop]"""
import sys, io, os, json
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
HERE = os.path.abspath('paper_output_pro/code'); sys.path.insert(0, HERE)
import numpy as np
from q3_forecast_B import candidates, NH, T
TABP = sys.argv[1]; OUT = sys.argv[2]
XF = int(sys.argv[3]) if len(sys.argv) > 3 else 0
STOP= int(sys.argv[4]) if len(sys.argv) > 4 else 0
SF = int(sys.argv[5]) if len(sys.argv) > 5 else 0
ADJ = int(sys.argv[6]) if len(sys.argv) > 6 else 1
TAB = np.load(TABP)
assert TAB.shape[1:] == (24, NH), TAB.shape
import q3_fc_solver_B as S

```

```

def fs(V, sig, d, dec, mode, scope='all', lam=0.5):
    arr = np.full(T, np.nan); lead = np.full(T, np.nan); nmem = np.zeros(T, int)
    row = TAB[min(d, TAB.shape[0]-1)]
    for t in range(T):
        c = candidates(V, d, t, dec, scope)
        nmem[t] = len(c)
        if not c: continue
        th = t // 6
        ws = np.asarray([1.0/row[th, j]**2 for _, j, _ in c], float)
        vals = np.asarray([v for _, v in c], float)

```

```

arr[t] = float(np.dot(ws, vals)/ws.sum())

lead[t] = float(min(j for _,j,_ in c))

return arr, lead, nmem

S.forecast_stage = fs
class _M(dict):
    def __getitem__(self, k): return 'inv_roll'

S.MODES = _M(S.MODES)
argv = ['q3_fc_solver_B.py', '--mode', 'e3-inv', '--scope', 'all', '--sigma-mode', 'jan', '--out', OUT]
if XF: argv += ['--replace-actual-from', str(XF)]
if STOP: argv += ['--stop-day', str(STOP)]
if SF: argv += ['--score-from', str(SF)]
argv += ['--adjust', str(ADJ)]
sys.argv = argv
S.main()
o = json.load(open(os.path.join(OUT, 'metrics.json'), encoding='utf-8'))
o['diagnostics']['sigma_variant'] = 'roll:' + os.path.basename(TABP)
o['diagnostics']['sigma_calibration'] = {
    'mode': 'expanding_window',
    'rule': '第 d 天决策只使用「目标日 <= d-1」的残差（严格因果）」',
    'formula': 'sigma^2(目标小时 th, 提前量 Delta) = 0.5*s2_cell(th,Delta) + 0.5*s2_pooled(Delta)',
    'free_params': 0,
    'n_samples_day31': 2940,
    'n_samples_day364': 34908,
    'sigma_table_shape': [int(x) for x in TAB.shape],
    'floor_kw': 1.0,
    'leaky': False,
    'label': '扩展窗标定, 逐日扩张, 零自由参数',
}
}
json.dump(o, open(os.path.join(OUT, 'metrics.json'), 'w', encoding='utf-8'), ensure_ascii=False, indent=1)

```

D.2.46 支撑材料/代码/03_问题三/q3_fc_rolltest.py

-*- coding: utf-8 -*-

扩展窗 $\hat{\sigma}$ 的数据泄露检验与复现性验证

#

L2: 破坏第 60 天起实际值后, 第 0..60 天的 $\hat{\sigma}$ 表必须逐位不变

L4: 端到端, 第 31..59 天的决策必须逐位不变

用法: python paper_output_pro/code/q3_fc_rolltest.py

```

import sys, io, os, json, subprocess, hashlib
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
ROOT = os.path.abspath('.'); E = 'paper_output_pro/experiments/'
env = dict(os.environ, PYTHONIOENCODING='utf-8')
import numpy as np
def sh(p):
    h = hashlib.sha256()
    with open(p, 'rb') as f:
        for b in iter(lambda: f.read(1<<20), b''): h.update(b)
    return h.hexdigest()

print('=' * 84)
print('扩展窗  $\sigma$  数据泄露检验')
print('=' * 84)
print('因果规则: 第 d 天决策只使用「目标日 <= d-1」的残差。')
print()

```

---- L2/L3: 破坏未来数据后, 早期表必须逐位不变 ----

```
ok = True
for mode in ('half', 'k0', 'pool'):
    clean = np.load(E + '_s_roll_%s.npy' % mode)
    subprocess.run([sys.executable, 'paper_output_pro/_s_roll.py', mode,
                    E + '_s_roll_%s_bad.npy' % mode, '60'], stdout=subprocess.DEVNULL,
                    stderr=subprocess.STDOUT, cwd=ROOT, env=env)
    bad = np.load(E + '_s_roll_%s_bad.npy' % mode)
    same = np.array_equal(clean[:61], bad[:61], equal_nan=True)
    ok &= same
    print('L2 [%s] 把第 60 天起实际光伏置 0 后重算 -> 第 0..60 天的  $\hat{g}$  表逐位不变 = %s'
          % (mode, same))
print()
```

---- L4: 端到端 ----

```
A = E + '_s_rl_a'; B = E + '_s_rl_b'
subprocess.run([sys.executable, 'paper_output_pro/_s_roll_run.py', E + '_s_roll_half.npy', A, '0', '60'],
               stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT, cwd=ROOT, env=env)
subprocess.run([sys.executable, 'paper_output_pro/_s_roll_run.py', E + '_s_roll_half_bad.npy', B, '60', '60'],
               stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT, cwd=ROOT, env=env)

sa = json.load(open(A + '/solution.json', encoding='utf-8')); sb = json.load(open(B + '/solution.json', encoding='utf-8'))
mx = 0.0; n = min(len(sa['plan0']), len(sb['plan0']))
for k in ('plan0', 'plan', 'c', 'd', 'E', 'S0', 'S1'):
    for i in range(n):
        mx = max(mx, float(np.max(np.abs(np.asarray(sa[k][i], float) - np.asarray(sb[k][i], float)))))
print('L4 端到端: 破坏第 60 天起实际值后重跑, 第 31..%d 天决策最大绝对差 = %.3e -> %s'
      % (30+n, mx, 'PASS' if mx == 0.0 else 'FAIL'))
ok &= (mx == 0.0)
```

---- 复现性 ----

```
C = E + '_s_rl_c'
subprocess.run([sys.executable, 'paper_output_pro/_s_roll_run.py', E + '_s_roll_half.npy', C],
               stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT, cwd=ROOT, env=env)

print()
print('复现性 (扩展窗 half, 两次独立运行):')
for f in ('metrics.json', 'solution.json', 'windows.json'):
    a = sh(E + '_s_rr_half/' + f); b = sh(C + '/' + f)

    ok &= (a == b)

    print(' %-16s %s %s %s' % (f, a[:20], b[:20], '一致' if a==b else '★不一致'))

print()
print('总判定: %s' % ('ALL PASS' if ok else '★FAIL'))
sys.exit(0 if ok else 1)
```

D.2.47 支撑材料/代码/03_问题三/q3_fc_sigma_B.py

-*- coding: utf-8 -*-

"""加权方差 $\sigma_{\text{hat}}(\Delta)$ 的标定与导出 (硬约束: 只能用第一个月)。

```
输出 paper_output_pro/experiments/q3-fc-sigma-calibration_B.json
jan : 正式口径, 发布日 2025-01-01..01-31 且 目标日 2025-01-01..01-31
full : △ 含泄露, 全部 365 天, 仅供上界对照
"""
import os, sys, json
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
```

```

sys.path.insert(0, HERE)
from q3_forecast_B import load_vintages, fixed_sigma, full_sample_sigma # noqa: E402

def main():
    V, meta = load_vintages(os.path.join(HERE, 'q3_vintages_B.json'))
    inp = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
    pv_a = np.asarray(inp['pv_actual'], float)
    sig, info = fixed_sigma(V, pv_a)
    print('标定窗口: 发布日 %s, 目标日 %s (共 %d 天)' % (info['pub_dates'], info['tgt_dates'], info['pub_days'][1] -
    info['pub_days'][0]))
    print('总样本 = %d' % info['n_total'])
    for j in range(24):
        print('Delta=%2dh n=%3d sigma_hat=%8.2f kW' % (j, info['n_by_lead'][j], sig[j]))
    small = [j for j in range(24) if info['n_by_lead'][j] < info['min_n']]
    print('样本 < %d 的档位: %s; 合并情况: %s' % (info['min_n'], small if small else '无', info['merged_bins'] or '无'))
    s2, info2 = full_sample_sigma(V, pv_a)
    print('对照(含泄露, 全样本 %d 条): 每档 n = %s' % (info2['n_total'], info2['n_by_lead']))
    p = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q3-fc-sigma-calibration_B.json')
    json.dump({'jan': info, 'full_LEAKY': info2}, open(p, 'w', encoding='utf-8'), ensure_ascii=False, indent=1)
    print('wrote', p)

if __name__ == '__main__':
    main()

```

D.2.48 支撑材料/代码/03_问题三/q3_fc_solver_A.py

-*- coding: utf-8 -*-

"""问题三 · 预报版本实验求解器 (子代理A: E1 / E2, 因子A = 0:00 不知).

结构完全复制 code/q3_realtime_solver.py:

0:00 计划 -> 6:00/12:00/18:00 用新预报调整剩余时段 -> 储能实时执行(口径^②) -> 结算

唯一的差别是光伏预报向量来自 q3_forecast_A.ForecastEngine (逐时段取版规则).

结算规则(不变): 成交 = price*min(P_plan, P_adj);
 偏差 = 0.5*price*(P_plan-P_adj)+ + 1.5*price*(P_adj-P_plan)+;
 紧急 = 5*price*E; 统计期 d >= 31.

"""

```

import os, sys, json, argparse, datetime
import numpy as np

```

```

sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q3_mpc import build_seasonal, solve_window
from q3_forecast_A import ForecastEngine, STAGE_SLOTS, SPD, in_force_tables

```

```

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--input', default=os.path.join('paper_output_pro', 'data_cleaned', 'q3_input.json'))
    ap.add_argument('--vintages', default=os.path.join('paper_output_pro', 'code', 'q3_vintages_A.json'))
    ap.add_argument('--out', required=True)
    ap.add_argument('--scheme', default='newest', choices=['newest', 'invvar', 'invvar_tod', 'expdecay'])
    ap.add_argument('--boundary', default='strict', choices=['strict', 'adjust', 'full'])
    ap.add_argument('--adjust-stages', default='6,12,18')
    ap.add_argument('--pct', type=float, default=80.0)
    ap.add_argument('--window', type=int, default=14)

```

```

ap.add_argument('--alpha', type=float, default=0.15)
ap.add_argument('--beta', type=float, default=0.8)
ap.add_argument('--cal-window', type=int, default=60)
ap.add_argument('--calib', default='fixed_jan', choices=['fixed_jan', 'full', 'rolling'])
ap.add_argument('--min-bin-n', type=int, default=20)
ap.add_argument('--d0', type=int, default=31)
a = ap.parse_args()

inp = json.load(open(a.input, encoding='utf-8'))
price = np.asarray(inp['price'], float)
load = np.asarray(inp['load_actual'], float)
pv_a = np.asarray(inp['pv_actual'], float)
st = inp['storage']; dt = float(inp['dt']); eta = float(st['efficiency'])
Pmax = float(st['power_max']); smin = float(st['soc_min']); smax = float(st['soc_max'])
D, T = load.shape
net = load - pv_a

eng = ForecastEngine(a.vintages, pv_a, a.cal_window, a.calib, a.min_bin_n)
PV, LE, NV = eng.all_days(a.scheme, a.boundary)      # (D,4,T)
load_f = build_seasonal(load, D, T, a.alpha, a.beta)

def net_fc(d, si):
    return load_f[d] - PV[d, si]

stages = [] if a.adjust_stages.strip() in ("", 'none') else [int(x) for x in a.adjust_stages.split(',')]

soc_c = float(st['soc_initial'])
base_c = dev_c = em_c = 0.0
em_k = cur_k = ch = di = 0.0
# 诊断累计量
n_seg = n_up = n_down = n_same = 0
win_rows = []
em_blk = {}; em_blk_kwh = {}
em_k_changed = em_k_same = 0.0
em_c_changed = em_c_same = 0.0
traj = {'day': [], 'plan0': [], 'plan': [], 'E': [], 'C': [], 'D': [], 'S1': [],
        'pvfc': [], 'margin': [], 'soc0': []}

for d in range(D):
    if d < 2:
        margin = np.zeros(T)
    else:
        lo = max(1, d - a.window)
        errs = np.array([net[k] - net_fc(k, 0) for k in range(lo, d)])

```

```

margin = np.maximum(np.percentile(errs, a.pct, axis=0), 0.0)
p_p, c_p, dd_p, soc = solve_window(price, net_fc(d, 0) + margin, soc_c, dt, eta, Pmax, smin, smax)
Pf = p_p.copy()
dev_seg = np.zeros(T)
for h in stages:
    slot = h * 6
    si = STAGE_SLOTS.index(slot)
    before = Pf.copy()
    p_, c_, d_, s_ = solve_window(price[slot:], (net_fc(d, si) + margin)[slot:], soc[slot],
                                dt, eta, Pmax, smin, smax, plan_w=p_p[slot:])
    Pf[slot:] = p_
    soc = np.concatenate([soc[:slot], s_])
    if d >= a.d0:
        pw = price[slot:]; ppw = p_p[slot:]; pfw = Pf[slot:]; pbw = before[slot:]
        def _dev(x):
            return (0.5 * pw * np.maximum(0, ppw - x) + 1.5 * pw * np.maximum(0, x - ppw)) * dt
        dev_a = float(np.sum(_dev(pfw))); dev_b = float(np.sum(_dev(pbw)))
        base_a = float(np.sum(pw * np.minimum(ppw, pfw)) * dt)
        base_b = float(np.sum(pw * np.minimum(ppw, pbw)) * dt)
        saving = float(np.sum(pw * np.maximum(0, pbw - pfw)) * dt)
        up_kwh = float(np.sum(np.maximum(0, pfw - pbw)) * dt)
        dn_kwh = float(np.sum(np.maximum(0, pbw - pfw)) * dt)
        win_rows.append(dict(day=d, hour=h,
                            buy_saving=saving,      # 规范字面口径: 下调计划省下的购电费(总额)
                            dev_cost=dev_a,         # 规范字面口径: 该窗口结束时该段对申报量的偏差费用(水平值)
                            net=saving - dev_a,      # 规范字面口径的"净收益"
                            base_before=base_b, base_after=base_a,
                            dev_before=dev_b, dev_after=dev_a,
                            net_settle=(base_b + dev_b) - (base_a + dev_a), # 结算口径的窗口边际净收益
                            n_up=int(np.sum(pfw > pbw + 1e-9)),
                            n_down=int(np.sum(pfw < pbw - 1e-9)),
                            n_same=int(np.sum(np.abs(pfw - pbw) <= 1e-9)),
                            up_kwh=up_kwh, down_kwh=dn_kwh))
# --- 实时执行 (口径②) ---
e = np.zeros(T); c2 = np.zeros(T); d2 = np.zeros(T); cu = np.zeros(T)
ss = soc_c; S = np.zeros(T + 1); S[0] = ss
for t in range(T):
    need = net[d][t] - Pf[t]
    if need > 0:
        cap = min(Pmax, max(ss - smin, 0.0) * eta / dt)
        d2[t] = min(need, cap); e[t] = need - d2[t]; ss -= d2[t] * dt / eta
    else:
        cap = min(Pmax, max(smax - ss, 0.0) / (eta * dt))
        c2[t] = min(-need, cap); cu[t] = (-need) - c2[t]; ss += eta * c2[t] * dt
    S[t + 1] = ss
if d >= a.d0:
    base_c += float(np.sum(price * np.minimum(p_p, Pf)) * dt)
    dev_seg = 0.5 * price * np.maximum(0, p_p - Pf) + 1.5 * price * np.maximum(0, Pf - p_p)

```

```

dev_c += float(np.sum(dev_seg) * dt)
em_cost_seg = 5.0 * price * e * dt
em_c += float(np.sum(em_cost_seg))
em_k += float(e.sum() * dt); cur_k += float(cu.sum() * dt)
ch += float(c2.sum() * dt); di += float(d2.sum() * dt)
changed = np.abs(Pf - p_p) > 1e-9
n_seg += T
n_up += int(np.sum(Pf > p_p + 1e-9)); n_down += int(np.sum(Pf < p_p - 1e-9))
n_same += int(np.sum(np.abs(Pf - p_p) <= 1e-9))
em_k_changed += float(e[changed].sum() * dt); em_k_same += float(e[~changed].sum() * dt)
em_c_changed += float(em_cost_seg[changed].sum()); em_c_same += float(em_cost_seg[~changed].sum())
for kk, arr in (('day', d), ('plan0', p_p), ('plan', Pf), ('E', e), ('C', c2),
                ('D', d2), ('S1', S[1:]), ('pvfc', PV[d, 0].copy()),
                ('margin', margin), ('soc0', soc_c)):
    traj[kk].append(arr)
if d >= a.d0:
    # 紧急购电按"决策阶段块"归属: [0,36) 属 0:00 计划, [36,72) 属 6:00, ...
    em_cost_seg2 = 5.0 * price * e * dt
    for h, sl in ((0, 0), (6, 36), (12, 72), (18, 108)):
        nxt = 144 if h == 18 else sl + 36
        em_blk["%d" % h] = em_blk.get("%d" % h, 0.0) + float(em_cost_seg2[sl:nxt].sum())
        em_blk_kwh["%d" % h] = em_blk_kwh.get("%d" % h, 0.0) + float(e[sl:nxt].sum() * dt)
    soc_c = float(S[-1])

total = base_c + dev_c + em_c
neg = [w for w in win_rows if w['net'] < 0]
metrics = {
    'config': {'scheme': a.scheme, 'boundary': a.boundary, 'adjust_stages': stages,
               'pct': a.pct, 'window': a.window, 'alpha': a.alpha, 'beta': a.beta,
               'cal_window': a.cal_window, 'calib': a.calib, 'min_bin_n': a.min_bin_n,
               'calib_info': {k: v for k, v in eng.calib_info.items() if k != 'tod_groups'},
               'd0': a.d0, 'd1': D,
               'vintages': a.vintages, 'input': a.input},
    'q3_total_cost': total,
    'q3_purchase_cost': base_c,
    'q3_adjustment_cost': dev_c,
    'q3_emergency_cost': em_c,
    'q3_emergency_kwh': em_k,
    'q3_curtail_kwh': cur_k,
    'q3_charge_kwh': ch,
    'q3_discharge_kwh': di,
    'q3_caliber': 2.0,
    'diag': {
        'dev_over_purchase': dev_c / base_c if base_c else None,
        'dev_over_total': dev_c / total if total else None,
        'n_segments': n_seg,
        'n_adjusted_segments': n_up + n_down,
    }
}

```

```

'adjusted_share': (n_up + n_down) / n_seg if n_seg else None,
'n_over_buy': n_up, 'n_under_buy': n_down, 'n_unchanged': n_same,
'n_windows': len(win_rows),
'window_net_sum': float(sum(w['net'] for w in win_rows)),
'window_net_negative_count': len(neg),
'window_net_negative_share': len(neg) / len(win_rows) if win_rows else None,
'window_net_settle_sum': float(sum(w['net_settle'] for w in win_rows)),
'window_net_settle_negative_count': int(sum(1 for w in win_rows if w['net_settle'] < 0)),
'window_net_settle_negative_share': (sum(1 for w in win_rows if w['net_settle'] < 0) / len(win_rows)) if win_rows else
None,
'emergency_cost_by_stage_block': em_blk,
'emergency_kwh_by_stage_block': em_blk_kwh,
'window_net_by_hour': {},
'emergency_kwh_changed_seg': em_k_changed,
'emergency_kwh_unchanged_seg': em_k_same,
'emergency_cost_changed_seg': em_c_changed,
'emergency_cost_unchanged_seg': em_c_same,
},
}
for h in (6, 12, 18):
    ws = [w for w in win_rows if w['hour'] == h]
    if not ws:
        continue
    metrics['diag']['window_net_by_hour'][str(h)] = {
        'n': len(ws),
        'buy_saving_sum': float(sum(w['buy_saving'] for w in ws)),
        'dev_cost_sum': float(sum(w['dev_cost'] for w in ws)),
        'net_sum': float(sum(w['net'] for w in ws)),
        'net_mean': float(np.mean([w['net'] for w in ws])),
        'net_settle_sum': float(sum(w['net_settle'] for w in ws)),
        'net_settle_negative_share': float(np.mean([w['net_settle'] < 0 for w in ws])),
        'base_delta_sum': float(sum(w['base_after'] - w['base_before'] for w in ws)),
        'dev_delta_sum': float(sum(w['dev_after'] - w['dev_before'] for w in ws)),
        'n_negative': int(sum(1 for w in ws if w['net'] < 0)),
        'n_up_sum': int(sum(w['n_up'] for w in ws)),
        'n_down_sum': int(sum(w['n_down'] for w in ws)),
        'n_same_sum': int(sum(w['n_same'] for w in ws)),
        'up_kwh_sum': float(sum(w['up_kwh'] for w in ws)),
        'down_kwh_sum': float(sum(w['down_kwh'] for w in ws)),
    }
# 预报精度 (各方案实际投用的预报)
vf, vl, dl = in_force_tables(PV, LE, a.d0, D)
act = pv_a[a.d0:D].reshape(-1)
err = (vf - pv_a[a.d0:D]).reshape(-1)
metrics['forecast_accuracy'] = {
    'mae': float(np.mean(np.abs(err))), 'rmse': float(np.sqrt(np.mean(err ** 2))),
    'mbe': float(np.mean(err)),
    'mean_effective_lead_h': float(np.mean(vl)),

```

```

        'by_version_lead': {str(k): float(np.mean(np.abs(err[np.ceil(vl).astype(int).reshape(-1) == k]))
                                for k in range(1, 25)
                                if (np.ceil(vl).astype(int).reshape(-1) == k).any())},
    }

    os.makedirs(a.out, exist_ok=True)
    json.dump({'metrics': metrics, 'windows': win_rows},
              open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'), ensure_ascii=False)
    np.savez_compressed(os.path.join(a.out, 'trajectory.npz'),
                        **{k: np.asarray(v) for k, v in traj.items()})
    print("%-8s/%-6s stages=%-9s 总  %.2f= 成交  %.2f+ 偏差  %.2f+ 紧急  %.2f (%.1f kWh) 弃电  %.1f 调整段  %d/%d"
          % (a.scheme, a.boundary, a.adjust_stages, total, base_c, dev_c, em_c, em_k, cur_k,
             n_up + n_down, n_seg))

if __name__ == '__main__':
    main()

```

D.2.49 支撑材料/代码/03_问题三/q3_fc_solver_B.py

-*- coding: utf-8 -*-

"""问题三 四组预报融合实验 — E3 / E4 求解器 (后缀 _B).

结构完全复制 q3_realtime_solver.py (0:00 计划 -> 6/12/18 调整 -> 实时执行口径② -> 结算),
 负荷预测 (build_seasonal, alpha/beta)、冗余 (W=14, pct=80)、口径②、结算规则一律不变,
 只换光伏预报的取法。

```

mode:
    baseline  现有做法: 0:00 版覆盖全天, 6/12/18 直接切当天新版 (用于复算锚点)
    e4        因子 A=知全天 + 因子 B=取最新版
    e3-inv    因子 A=知全天 + 因子 B=逆方差加权
    e3-exp    因子 A=知全天 + 因子 B=指数衰减 (lambda=0.5)

scope:
    all      可用版本 = {v : 发布时刻 <= 决策时刻 且覆盖该时段} (跨天版本也参与, 规范字面推广)
    same-day 只保留当天发布的版本 (敏感性对照)

用法:
    python paper_output_pro/code/q3_fc_solver_B.py --mode e3-inv --out paper_output_pro/experiments/q3-fc-e3
"""
import os, sys, json, argparse, datetime, hashlib
import numpy as np

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from q3_mpc import build_seasonal, solve_window # noqa: E402
from q3_forecast_B import (load_vintages, residuals, rolling_sigma, forecast_stage, # noqa: E402
                           hourly_actual, fixed_sigma, full_sample_sigma, HOURS, NDAY, T)

STAGES = [(0, 0), (6, 36), (12, 72), (18, 108)]
MODES = {'e3-inv': 'inv', 'e3-exp': 'exp', 'e4': 'latest'}

def realtime(net_w, Pf_w, S0, dt, eta, Pmax, smin, smax):
    """口径② 实时执行: 储能按实际缺口实时充放电, 补不上才紧急购电。返回 (c, d, e, cu, S)。"""
    n = len(net_w); c = np.zeros(n); d2 = np.zeros(n); e = np.zeros(n); cu = np.zeros(n)
    S = np.zeros(n + 1); S[0] = S0
    ss = S0
    for t in range(n):
        need = net_w[t] - Pf_w[t]

```

```

    if need > 0:
        cap = min(Pmax, max(ss - smin, 0.0) * eta / dt)
        d2[t] = min(need, cap); e[t] = need - d2[t]; ss -= d2[t] * dt / eta
    else:
        cap = min(Pmax, max(smax - ss, 0.0) / (eta * dt))
        c[t] = min(-need, cap); cu[t] = (-need) - c[t]; ss += eta * c[t] * dt
    S[t + 1] = ss
return c, d2, e, cu, S

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--mode', required=True, choices=['baseline', 'e3-inv', 'e3-exp', 'e4'])
    ap.add_argument('--scope', default='all', choices=['all', 'same-day'])
    ap.add_argument('--lam', type=float, default=0.5)
    ap.add_argument('--input', default=os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q3_input.json'))
    ap.add_argument('--vintages', default=os.path.join(HERE, 'q3_vintages_B.json'))
    ap.add_argument('--out', required=True)
    ap.add_argument('--adjust', type=int, default=1)
    ap.add_argument('--pct', type=float, default=80.0)
    ap.add_argument('--window', type=int, default=14)
    ap.add_argument('--alpha', type=float, default=0.15)
    ap.add_argument('--beta', type=float, default=0.8)
    ap.add_argument('--sigma-min-n', type=int, default=20,
                    help='标定档位最小样本数, 不足则相邻档位合并')
    ap.add_argument('--sigma-mode', default='jan', choices=['jan', 'full', 'rolling'],
                    help='jan=只用第一个月标定(正式口径); full=全样本(含泄露, 仅供对照); rolling=滚动(对照)')
    ap.add_argument('--score-from', type=int, default=31)
    ap.add_argument('--stop-day', type=int, default=0, help='>0 时只跑到该天 (用于数据泄露检验)')
    ap.add_argument('--replace-actual-from', type=int, default=0,
                    help='>0 时把该天及以后的实际光伏替换为 0 (泄露检验用)')
    a = ap.parse_args()

    inp = json.load(open(a.input, encoding='utf-8'))
    price = np.asarray(inp['price'], float)
    load = np.asarray(inp['load_actual'], float)
    pv_a = np.asarray(inp['pv_actual'], float)
    st = inp['storage']; dt = float(inp['dt']); eta = float(st['efficiency'])
    Pmax = float(st['power_max']); smin = float(st['soc_min']); smax = float(st['soc_max'])
    if a.replace_actual_from:
        pv_a = pv_a.copy(); pv_a[a.replace_actual_from:] = 0.0
    D, Tt = load.shape
    assert Tt == T
    if a.stop_day:
        D = a.stop_day
    net = load - pv_a
    load_f = build_seasonal(load, D, T, a.alpha, a.beta)
    pv_fs = build_seasonal(pv_a, D, T, a.alpha, a.beta)

```

```

ha = hourly_actual(pv_a)

# ----- 光伏预报提供者 -----
V = sig = None; cal_info = None
if a.mode != 'baseline':
    V, vmeta = load_vintages(a.vintages)
    if a.sigma_mode == 'jan':
        s0, cal_info = fixed_sigma(V, pv_a, min_n=a.sigma_min_n)
        cal_info['leaky'] = False
        cal_info['label'] = '第一个月 (2025-01-01..2025-01-31) 标定, 标定后固定, 全年沿用'
        sig = np.tile(s0, (NDAY, 1))
    elif a.sigma_mode == 'full':
        s0, cal_info = full_sample_sigma(V, pv_a, min_n=a.sigma_min_n)
        cal_info['leaky'] = True
        cal_info['label'] = '!!! 含泄露 (全部 365 天样本), 仅供上界对照, 不得作为正式结果 !!!'
        sig = np.tile(s0, (NDAY, 1))
    else:
        R = residuals(V, pv_a)
        sig = rolling_sigma(R, min_n=a.sigma_min_n)
        cal_info = {'mode': 'rolling', 'leaky': False, 'label': '滚动扩展窗标定 (对照, 非规范口径)'}
else:
    fc = {h: np.array([inp['pv_forecast']['%d_%d' % (d, h)] for d in range(D)]) for h in (0, 6, 12, 18)}
    vmeta = {'source': 'q3_input.json:pv_forecast (每版截断在自己那天)'}

def pv_stage(d, s):
    """返回 (144,) 预报功率 kW, (144,) 所用最新成员提前量 h, (144,) 成员数"""
    off, slot = STAGES[s]
    if a.mode == 'baseline':
        arr = fc[off][d].copy()
        lead = np.full(T, np.nan)
        for t in range(T):
            k = t // 6
            if k >= off:
                lead[t] = k - off
        return arr, lead, np.where(np.isfinite(arr), 1, 0)
    return forecast_stage(V, sig, d, d * 24 + off, MODES[a.mode], a.scope, a.lam)

def net_fc(d, s):
    arr, _, _ = pv_stage(d, s)
    arr = arr.copy()
    miss = ~np.isfinite(arr)
    arr[miss] = pv_fs[d][miss]
    return load_f[d] - arr

# ----- 逐年滚动 -----

```

```

soc_c = float(st['soc_initial'])
base_c = dev_c = em_c = 0.0
em_k = cur_k = ch = di = 0.0
n_adj = n_over = n_under = n_same = n_seg_scored = n_seg_adjable = 0
win = []          # 逐窗口诊断
fc_err = []       # (lead, forecast, actual) 全部计分日
rows = []
detail_days = []
for d in range(D):
    if d < 2:
        margin = np.zeros(T)
    else:
        lo = max(1, d - a.window)
        errs = np.array([net[k] - net_fc(k, 0) for k in range(lo, d)])
        margin = np.maximum(np.percentile(errs, a.pct, axis=0), 0.0)
    p_p, c_p, dd_p, soc = solve_window(price, net_fc(d, 0) + margin, soc_c, dt, eta, Pmax, smin, smax)
    Pf = p_p.copy()
    if a.adjust:
        for s, (h, slot) in enumerate(STAGES[1:], start=1):
            S0 = float(soc[slot])
            p_, c_, d_, s_ = solve_window(price[slot:], (net_fc(d, s) + margin)[slot:], S0, dt, eta,
                                          Pmax, smin, smax, plan_w=p_p[slot:])
            # ---- 单窗口净收益诊断 (同一窗口起点储能, 两种购电计划, 对同一实际净负荷) ----
            pw = price[slot:]; planw = p_p[slot:]; netw = net[d][slot:]
            _, _, eA, _, _ = realtime(netw, p_, S0, dt, eta, Pmax, smin, smax)
            _, _, eB, _, _ = realtime(netw, planw, S0, dt, eta, Pmax, smin, smax)
            devw = float(np.sum(0.5 * pw * np.maximum(0, planw - p_) + 1.5 * pw * np.maximum(0, p_ - planw)) * dt)
            save_base = float(np.sum(pw * np.maximum(0, planw - p_) * dt)
            save_em = 5.0 * float(np.dot(pw, eB - eA)) * dt
            if d >= a.score_from:
                win.append({'day': d, 'stage': h, 'slot': slot,
                           'save_purchase': save_base, 'save_emergency': save_em, 'dev_fee': devw,
                           'net_benefit': save_base + save_em - devw,
                           'changed_segments': int(np.sum(np.abs(p_ - planw) > 1e-6)),
                           'em_kwh_plan': float(eB.sum() * dt), 'em_kwh_adj': float(eA.sum() * dt)})
            Pf[slot:] = p_; soc = np.concatenate([soc[:slot], s_])
# ---- 实时执行 (口径②) ----
c2, d2, e, cu, S = realtime(net[d], Pf, soc_c, dt, eta, Pmax, smin, smax)
if d >= a.score_from:
    base_c += float(np.sum(price * np.minimum(p_p, Pf)) * dt)
    dev_c += float(np.sum(0.5 * price * np.maximum(0, p_p - Pf) + 1.5 * price * np.maximum(0, Pf - p_p)) * dt)
    em_c += 5.0 * float(np.dot(price, e)) * dt
    em_k += float(e.sum() * dt); cur_k += float(cu.sum() * dt)
    ch += float(c2.sum() * dt); di += float(d2.sum() * dt)
    dP = Pf - p_p
    n_adj += int(np.sum(np.abs(dP) > 1e-6))
    n_over += int(np.sum(dP > 1e-6)); n_under += int(np.sum(dP < -1e-6))
    n_seg_scored += T; n_seg_adjable += T - 36

```

```

n_same += int(np.sum(np.abs(dP) <= 1e-6))
for s, (h, slot) in enumerate(STAGES):
    arr, lead, nm = pv_stage(d, s)
    seg = np.arange(T)
    used = (seg >= slot)
    for t in np.where(used)[0]:
        if np.isfinite(arr[t]):
            fc_err.append((float(lead[t]), float(arr[t]), float(ha[d, t // 6])))
    rows.append({'day': d, 'plan0': p_p.copy(), 'plan': Pf.copy(),
                'c': c2.copy(), 'd': d2.copy(), 'E': e.copy(),
                'S0': float(soc_c), 'S1': float(S[-1])})
    detail_days.append(d)
soc_c = float(S[-1])

win_arr = np.array([w['net_benefit'] for w in win]) if win else np.zeros(0)
neg = win_arr < 0
# ---- 预报精度 (按所用最新成员的提前量分档) ----
fe = np.array(fc_err) if fc_err else np.zeros((0, 3))
bins = {}
for L in range(0, 25):
    m = np.abs(fe[:, 0] - L) < 1e-9 if len(fe) else np.zeros(0, bool)
    if m.sum():
        e = fe[m, 1] - fe[m, 2]
        bins['%02d' % L] = {'n': int(m.sum()), 'mae': float(np.abs(e).mean()),
                            'rmse': float(np.sqrt((e ** 2).mean()))}, 'bias': float(e.mean())}
if len(fe):
    e = fe[:, 1] - fe[:, 2]
    overall = {'n': int(len(fe)), 'mae': float(np.abs(e).mean()),
              'rmse': float(np.sqrt((e ** 2).mean()))}, 'bias': float(e.mean())}
else:
    overall = {'n': 0, 'mae': None, 'rmse': None, 'bias': None}

m = {'q3_total_cost': base_c + dev_c + em_c, 'q3_purchase_cost': base_c,
     'q3_adjustment_cost': dev_c, 'q3_emergency_cost': em_c, 'q3_emergency_kwh': em_k,
     'q3_curtail_kwh': cur_k, 'q3_charge_kwh': ch, 'q3_discharge_kwh': di,
     'q3_caliber': 2.0, 'q3_adjust': float(a.adjust)}

diag = {
    'mode': a.mode, 'scope': a.scope, 'lam': a.lam,
    'sigma_mode': a.sigma_mode, 'sigma_calibration': cal_info,
    'dev_over_purchase_ratio': dev_c / base_c if base_c else None,
    'adjusted_segments': n_adj, 'total_segments': n_seg_scored,
    'adjusted_share': n_adj / n_seg_scored if n_seg_scored else None,
    'adjustable_segments': n_seg_adjable,
    'adjusted_share_of_adjustable': n_adj / n_seg_adjable if n_seg_adjable else None,
    'segments_over_bought': n_over, 'segments_under_bought': n_under, 'segments_unchanged': n_same,

```

```

'n_windows': len(win),
>window_save_purchase_sum': float(sum(w['save_purchase'] for w in win)),
>window_save_emergency_sum': float(sum(w['save_emergency'] for w in win)),
>window_dev_fee_sum': float(sum(w['dev_fee'] for w in win)),
>window_net_benefit_sum': float(win_arr.sum()) if len(win_arr) else 0.0,
>windows_negative_net': int(neg.sum()),
>windows_negative_share': float(neg.mean()) if len(neg) else None,
>windows_negative_loss_sum': float(win_arr[neg].sum()) if len(win_arr) else 0.0,
>windows_positive_gain_sum': float(win_arr[~neg].sum()) if len(win_arr) else 0.0,
>forecast_accuracy_overall': overall,
>forecast_accuracy_by_lead': bins,
}
os.makedirs(a.out, exist_ok=True)
json.dump({'metrics': m, 'diagnostics': diag},
          open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'), ensure_ascii=False, indent=1)
json.dump(win, open(os.path.join(a.out, 'windows.json'), 'w', encoding='utf-8'), ensure_ascii=False)
json.dump({'days': detail_days,
          'plan0': [r['plan0'].tolist() for r in rows],
          'plan': [r['plan'].tolist() for r in rows],
          'c': [r['c'].tolist() for r in rows],
          'd': [r['d'].tolist() for r in rows],
          'E': [r['E'].tolist() for r in rows],
          'S0': [r['S0'] for r in rows], 'S1': [r['S1'] for r in rows]}},
          open(os.path.join(a.out, 'solution.json'), 'w', encoding='utf-8'))
rec = {'argv': sys.argv, 'cwd': os.getcwd(), 'mode': a.mode, 'scope': a.scope,
      'vintages_meta': vmeta,
      'created_at_utc': datetime.datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ'),
      'params': {'pct': a.pct, 'window': a.window, 'alpha': a.alpha, 'beta': a.beta,
                  'adjust': a.adjust, 'score_from': a.score_from, 'sigma_min_n': a.sigma_min_n,
                  'sigma_mode': a.sigma_mode}}
json.dump(rec, open(os.path.join(a.out, 'receipt.json'), 'w', encoding='utf-8'), ensure_ascii=False, indent=1)
print('%s[%s]: 总 %.2f = 成交 %.2f + 偏差 %.2f + 紧急 %.2f (%.2f kWh) 弃电 %.1f 调整段 %d/%d 负收益窗口 %d/%d'
      % (a.mode, a.scope, m['q3_total_cost'], base_c, dev_c, em_c, em_k, cur_k,
         n_adj, n_seg_scored, int(neg.sum()), len(win)))

if __name__ == '__main__':
    main()

```

D.2.50 支撑材料/代码/03_问题三/q3_fc_summary_B.py

-*- coding: utf-8 -*-

```

"""汇总 E3/E4 + 基准 的对比表 (Markdown + xlsx) 与诊断指标表。"""
import os, sys, json
import numpy as np

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
EXP = os.path.join(ROOT, 'paper_output_pro', 'experiments')

RUNS = [
    ('基准(现有做法)', os.path.join(EXP, 'q3-realtime-adjust'), 'baseline',

```

```

'0:00 版覆盖全天, 6/12/18 直接切当天新版'),
('基准(本次复算_B)', os.path.join(EXP, '_baseline_check_B'), 'baseline',
'q3_fc_solver_B.py --mode baseline 复算, 用于确认锚点'),
('基准(原脚本 q3_realtime_solver)', os.path.join(EXP, '_baseline_orig_B'), 'baseline',
'直接运行 paper_output_pro/code/q3_realtime_solver.py'),
('E4 知全天+最新版', os.path.join(EXP, 'q3-fc-e4'), 'e4',
'可用版本={当天 0:00 版} ∪ {已发布且覆盖的最新版}, 取 pub 最晚一版'),
('E4 知全天+最新版(当天版)', os.path.join(EXP, 'q3-fc-e4-variants', 'sameday'), 'e4',
'只保留当天发布版本 (敏感性对照)'),
('E3 知全天+逆方差(all) [正式]', os.path.join(EXP, 'q3-fc-e3'), 'e3-inv',
'w_v = 1/sigma_hat(Delta_v)^2; sigma 用 2025-01-01..01-31 标定后固定全年; 跨天版本参与'),
('E3 知全天+指数衰减(all)', os.path.join(EXP, 'q3-fc-e3-variants', 'exp-decay'), 'e3-exp',
'w_v = 0.5^Delta_v (不含 sigma, 与标定窗口无关), 跨天版本参与'),
('E3 知全天+逆方差(当天版)', os.path.join(EXP, 'q3-fc-e3-variants', 'inv-sameday'), 'e3-inv',
'sigma=1 月标定; 只融合当天已发布版本 (敏感性对照)'),
('E3 知全天+指数衰减(当天版)', os.path.join(EXP, 'q3-fc-e3-variants', 'exp-decay-sameday'), 'e3-exp',
'只融合当天已发布版本 (敏感性对照)'),
('E3 逆方差 sigma=全样本 [含泄露, 仅供对照]', os.path.join(EXP, 'q3-fc-e3-variants', 'inv-fullsample-LEAKY'),
'e3-inv', '!!! 方差用全部 365 天标定, 含数据泄露, 仅作为上界参考, 不得作为正式结果 !!!),
]

def load(d):
    m = json.load(open(os.path.join(d, 'metrics.json'), encoding='utf-8'))
    return m

def main():
    acc = json.load(open(os.path.join(EXP, 'q3-fc-accuracy_B.json'), encoding='utf-8'))
    rows = []
    for label, d, mode, desc in RUNS:
        if not os.path.exists(os.path.join(d, 'metrics.json')):
            print('MISSING', d); continue
        m = load(d); mm = m['metrics']; dg = m.get('diagnostics', {})
        aux = d
        need_aux = (not dg) or (not os.path.exists(os.path.join(d, 'solution.json')))
        base_aux = os.path.join(EXP, '_baseline_check_B')
        if need_aux and os.path.exists(os.path.join(base_aux, 'solution.json')):
            aux = base_aux # 纯基准复算目录只有 metrics.json, 逐时段/诊断取同一次复算(三者完全等价)
        if not dg:
            dg = load(aux).get('diagnostics', {})
        sol = json.load(open(os.path.join(aux, 'solution.json'), encoding='utf-8'))
        adj_kwh = float(np.sum(np.abs(np.array(sol['plan']) - np.array(sol['plan0']))) * (1.0 / 6.0))
        rows.append(dict(label=label, dir=os.path.relpath(d, ROOT), mode=mode, desc=desc,
            total=mm['q3_total_cost'], base=mm['q3_purchase_cost'],
            dev=mm['q3_adjustment_cost'], em=mm['q3_emergency_cost'],
            emkwh=mm['q3_emergency_kwh'], cur=mm['q3_curtail_kwh'],
            ch=mm['q3_charge_kwh'], dis=mm['q3_discharge_kwh'],
            adj_kwh=adj_kwh,
            **{k: v for k, v in dg.items() if k not in ('forecast_accuracy_by_lead', 'mode')}))

```

```

ref = rows[0]
md = []
md.append('方案 | 总费用(元) | 成交购电费 | 偏差费用 | 紧急购电费 | 与基准差额 | 差额% |')
md.append('---|---|---|---|---|---|')
for r in rows:
    dlt = r['total'] - ref['total']
    md.append('%s | %.2f | %.2f | %.2f | %.2f | %+.2f | %+.4f%% |'
              % (r['label'], r['total'], r['base'], r['dev'], r['em'], dlt, 100 * dlt / ref['total']))
t1 = '\n'.join(md)

md2 = ['方案 | 偏差/成交 | 调整段数/总段数 | 多买 | 少买 | 未改 | 调整电量(kWh) | 窗口数 | 负净收益窗口 | 占比 | 窗口净收益合计 | 窗口省购电费 | 窗口省紧急费 | 窗口偏差费 |',
      '---|---|---|---|---|---|---|---|---|---|']
for r in rows:
    md2.append('%s | %.4f | %d/%d (%.1f%%) | %d | %d | %d | %.0f | %d | %d | %.1f%% | %+.0f | %+.0f | %+.0f | %+.0f |'
              % (r['label'], r['dev_over_purchase_ratio'], r['adjusted_segments'], r['total_segments'],
                100 * r['adjusted_share'], r['segments_over_bought'], r['segments_under_bought'],
                r['segments_unchanged'], r['adj_kwh'], r['n_windows'], r['windows_negative_net'],
                100 * r['windows_negative_share'], r['window_net_benefit_sum'], r['window_save_purchase_sum'],
                r['window_save_emergency_sum'], r['window_dev_fee_sum']))
t2 = '\n'.join(md2)

keys = [('baseline', '基准(现有做法)'), ('e4', 'E4 最新版'), ('e3-inv', 'E3 逆方差(all)'),
        ('e3-exp', 'E3 指数衰减(all)'), ('e3-inv-sameday', 'E3 逆方差(当天版)'),
        ('e3-exp-sameday', 'E3 指数衰减(当天版)'),
        ('e3-inv-LEAKY', '[含泄露仅供对照] E3 逆方差(all, sigma=全样本)')]
md3 = ['取版方式 | 全部时段 MAE | RMSE | 白天 MAE | 白天 RMSE | 平均提前量(h) | 0-5h MAE | 6-11h MAE | 12-17h MAE | 18-23h MAE |',
      '---|---|---|---|---|---|---|---|']
for k, nm in keys:
    s = acc['schemes'][k]
    g = s['by_lead_group']; gd = s['by_lead_group_daytime']
    md3.append('%s | %.2f | %.2f | %.2f | %.2f | %.2f | %.1f | %.1f | %.1f | %.1f |'
              % (nm, s['all']['mae'], s['all']['rmse'], s['daytime']['mae'], s['daytime']['rmse'],
                s['mean_lead'], gd['00-05h']['mae'], gd['06-11h']['mae'], gd['12-17h']['mae'], gd['18-23h']['mae']))
t3 = '\n'.join(md3)

md4 = ['提前量 Δ | 单版本 MAE | RMSE | bias |', '---|---|---|']
for j in range(24):
    r = acc['single_version_reference']['%02d' % j]
    md4.append('%dh | %.2f | %.2f | %+.2f |' % (j, r['mae'], r['rmse'], r['bias']))
t4 = '\n'.join(md4)

# ---- sigma 标定表 ----
cal = json.load(open(os.path.join(EXP, 'q3-fc-sigma-calibration_B.json'), encoding='utf-8'))

```

```

jan = cal['jan']
md0 = ['| 提前量 Δ | '+'| '.join('%dh' % j for j in range(24)) + '|',
       '|---|' + '---|' * 24,
       '| 样本数 n | '+'| '.join(str(x) for x in jan['n_by_lead']) + '|',
       '| sigma_hat (kW) | '+'| '.join("%.1f" % x for x in jan['sigma_by_lead']) + '|']
t0 = chr(10).join(md0)
doc = ['# 问题三 E3/E4 光伏预报取版实验 — 汇总对比', ",
       '## 0. 加权方差 sigma_hat(Delta) 的标定（硬约束：只能用第一个月）', ",
       '标定窗口：发布日 **2025-01-01 .. 2025-01-31**（第 0..30 天共 31 天）；目标时刻所属日同样限定在 1 月内，'
       '1 月 31 日发布的版本覆盖到 2 月的样本已剔除。标定一次后 **固定不变、全年沿用**（不滚动）。',
       '总样本 **%d** 条，每档 121..124 条，全部 >= 20，**未触发相邻档位合并**。' % jan['n_total'], ",
       t0, ",
       '小样本处理规则（已实现，本次未触发）：若某档 n < 20，从该档向两侧同时扩展 k=1,2,... 合并相邻档位，'
       '直到合并样本 >= 20；若全部 24 档合并仍不足，则退化为全局标准差。合并情况记录在 '
       'metrics.json 的 diagnostics.sigma_calibration.merged_bins 字段。', ",
       '△ 对照（**含泄露，仅供上界参考，不得作为正式结果**）：用全部 365 天的 %d 条样本标定。' %
       cal['full_LEAKY']['n_total'], ",
       '## 1. 总费用对比（统计期 2025-02-01..12-31, 共 334 天, 口径②）', ", t1, ",
       '> 成交/偏差/紧急分别对应：按式(7)的成交部分购电费、偏差费用(下调 0.5 倍/上调 1.5 倍)、紧急购电费(5 倍)。', ",
       '## 2. 调整罚金诊断指标（规范第四节）', ", t2, ",
       '> 窗口 = (天, 6:00/12:00/18:00) 一次调整决策, 共 334×3=1002 个。',
       '> 单窗口净收益 = 成交购电费节省 + 紧急购电费节省 - 偏差费用；',
       '> 两条对照臂从同一窗口起点储电量出发, 对同一实际净负荷执行口径②。', ",
       '## 3. 光伏预报精度对比（计分日, 按所用成员中最新一版的提前量分档）', ", t3, ",
       '## 4. 单版本预报按提前量的误差（附件 3 全部 365 天, 小时平均口径）', ", t4, "]
out = os.path.join(EXP, 'q3-fc-e3')
os.makedirs(out, exist_ok=True)
open(os.path.join(out, 'summary_B.md'), 'w', encoding='utf-8').write("\n".join(doc))
json.dump(rows, open(os.path.join(EXP, 'q3-fc-summary_B.json'), 'w', encoding='utf-8'),
          ensure_ascii=False, indent=1, default=float)
print("\n".join(doc))
print("\nwrote", os.path.join(out, 'summary_B.md'))
return rows

if __name__ == '__main__':
    main()

```

D.2.51 支撑材料/代码/03_问题三/q3_fc_verify_A.py

-*- coding: utf-8 -*-

```

"""独立复算校验：只用 trajectory.npz + 电价 + 储能参数，重算结算并检查约束."""
import os, sys, json
import numpy as np

```

```

R = os.path.join('paper_output_pro', 'experiments')
inp = json.load(open(os.path.join('paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
price = np.asarray(inp['price'], float); dt = 1.0 / 6
st = inp['storage']; eta = st['efficiency']; Pmax = st['power_max']
smin, smax = st['soc_min'], st['soc_max']

```

```

RUNS = ['q3-fc-e1/invvar', 'q3-fc-e1/expdecay', 'q3-fc-e1/invvar_tod',
        'q3-fc-e2/newest', 'q3-fc-e2/newest_adjust', 'q3-fc-e2/newest_full']

print('%-26s %14s %14s %10s %10s %s' % ('run', 'metrics 总费用', '复算总费用', '最大 SOC 越界', '最大功率越界', '状态'))
allok = True
for rel in RUNS:
    o = json.load(open(os.path.join(R, rel, 'metrics.json'), encoding='utf-8'))['metrics']

    z = np.load(os.path.join(R, rel, 'trajectory.npz'))

    p0, pf, E, C, Dd, S1 = z['plan0'], z['plan'], z['E'], z['C'], z['D'], z['S1']

    base = float(np.sum(price * np.minimum(p0, pf)) * dt)

    dev = float(np.sum((0.5 * price * np.maximum(0, p0 - pf)
        + 1.5 * price * np.maximum(0, pf - p0)) * dt))

    em = float(np.sum(5.0 * price * E * dt))

    soc_bad = float(np.max(np.maximum(smin - S1, S1 - smax)))

    p_bad = float(max(np.max(C), np.max(Dd)) - Pmax)

    ok = (abs(base - o['q3_purchase_cost']) < 1e-6 and abs(dev - o['q3_adjustment_cost']) < 1e-6
        and abs(em - o['q3_emergency_cost']) < 1e-6 and soc_bad <= 1e-6 and p_bad <= 1e-6)

    allok &= ok

    print('%-26s %14.2f %14.2f %10.2e %10.2e %s' % (rel, o['q3_total_cost'],
        base + dev + em, max(soc_bad, 0), max(p_bad, 0), 'PASS' if ok else 'FAIL'))

print('\n独立复算: %s' % ('全部通过' if allok else '存在不一致'))
sys.exit(0 if allok else 1)

```

D.2.52 支撑材料/代码/03_问题三/q3_fc_xlsx_B.py

-*- coding: utf-8 -*-

```

"""生成 E3/E4 对比表 xlsx。"""
import os, json
from openpyxl import Workbook
from openpyxl.styles import Font

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
EXP = os.path.join(ROOT, 'paper_output_pro', 'experiments')

rows = json.load(open(os.path.join(EXP, 'q3-fc-summary_B.json'), encoding='utf-8'))
acc = json.load(open(os.path.join(EXP, 'q3-fc-accuracy_B.json'), encoding='utf-8'))
ref = rows[0]['total']
r2 = lambda x, n=2: float(round(x, n))

wb = Workbook(); ws = wb.active; ws.title = '总费用对比'
ws.append(['方案', '目录', '取版说明', '总费用(元)', '成交购电费(元)', '偏差费用(元)', '紧急购电费(元)',
    '紧急购电量(kWh)', '与基准差额(元)', '差额%'])

for r in rows:
    ws.append([r['label'], r['dir'], r['desc'], r2(r['total']), r2(r['base']), r2(r['dev']), r2(r['em']),
        r2(r['emkwh']), r2(r['total'] - ref), r2(100 * (r['total'] - ref) / ref, 4)])

ws = wb.create_sheet('调整罚金诊断')
ws.append(['方案', '偏差/成交', '调整段数', '总段数', '调整占比%', '多买段数', '少买段数', '未改段数',
    '调整电量(kWh)', '窗口数', '负净收益窗口数', '负窗口占比%', '窗口省购电费合计', '窗口省紧急费合计',
    '窗口偏差费合计', '窗口净收益合计'])

for r in rows:
    ws.append([r['label'], r2(r['dev_over_purchase_ratio'], 5), r['adjusted_segments'], r['total_segments'],
        r2(100 * r['adjusted_share']), r['segments_over_bought'], r['segments_under_bought'],
        r['segments_unchanged'], r2(r['adj_kwh'], 1), r['n_windows'], r['windows_negative_net'],
        r2(100 * r['windows_negative_share'], 1), r2(r['window_save_purchase_sum'], 1),
        r2(r['window_save_emergency_sum'], 1), r2(r['window_dev_fee_sum'], 1),
        r2(r['window_net_benefit_sum'], 1)])

```

```

keymap = [('baseline', '基准(现有做法)'), ('e4', 'E4 最新版'), ('e3-inv', 'E3 逆方差(all)'),
          ('e3-exp', 'E3 指数衰减(all)'), ('e3-inv-sameday', 'E3 逆方差(当天版)'),
          ('e3-exp-sameday', 'E3 指数衰减(当天版)')]

ws = wb.create_sheet('预报精度对比')
ws.append(['取版方式', '全部 MAE', '全部 RMSE', '全部 bias', '白天 MAE', '白天 RMSE',
          '0-5h MAE', '6-11h MAE', '12-17h MAE', '18-23h MAE', '平均提前量(h)'])

for k, nm in keymap:
    s = acc['schemes'][k]; g = s['by_lead_group_daytime']
    ws.append([nm, r2(s['all']['mae']), r2(s['all']['rmse']), r2(s['all']['bias']), r2(s['daytime']['mae']),
              r2(s['daytime']['rmse']), r2(g['00-05h']['mae'], 1), r2(g['06-11h']['mae'], 1),
              r2(g['12-17h']['mae'], 1), r2(g['18-23h']['mae'], 1), r2(s['mean_lead'])])

ws = wb.create_sheet('单版本按提前量误差')
ws.append(['提前量 Δ(h)', 'MAE', 'RMSE', 'bias'])
for j in range(24):
    r = acc['single_version_reference']['%02d' % j]
    ws.append([j, r2(r['mae']), r2(r['rmse']), r2(r['bias'])])

for ws in wb.worksheets:
    for c in ws[1]:
        c.font = Font(bold=True)
    ws.freeze_panes = 'A2'
    for col in ws.columns:
        w = max(len(str(c.value)) if c.value is not None else 0 for c in col)
        ws.column_dimensions[col[0].column_letter].width = min(52, max(10, w + 2))

p = os.path.join(EXP, 'q3-fc-e3', 'E3E4_对比表_B.xlsx')
wb.save(p)
print('wrote', p)

```

D.2.53 支撑材料/代码/03_问题三/q3_forecast_A.py

-*- coding: utf-8 -*-

"""问题三 · 预报层 (子代理 A: E1/E2, 因子 A = 0:00 不知).

逐时段取版规则 (设计规范第三节):

E1/E2 (0:00 不知): 可用版本 = { v : $\text{pub}(v) < s$ 且 v 覆盖 s 且 $\text{pub}(v) \leq \text{决策时刻 } \tau$ }

- 0:00 计划阶段 $\tau = \text{当天 } 0:00$: d 日 0:00 版对 $t=0$ 不可用($\text{pub}=s$), 对 $t \geq 1$ 可用.

- 6/12/18 调整阶段同理: 当天 h 版对 $t=\text{slot}(h)$ 不可用, 对 $t > \text{slot}(h)$ 可用.

relaxed:

'strict' 规范字面口径 (默认): 所有阶段都要求 $\text{pub} < s$

'adjust' 仅把调整阶段(6/12/18)的 " $\text{pub}=s$ " 边界放开 (对 0:00 边界仍严格)

'full' 全部放开 (== 现行基准做法 / E4)

因子 B (取版方式):

'newest' 取 pub 最大的一版

'invvar' 逆方差加权 $w_v = 1 / \sigma_{\hat{v}}^2(\Delta_v)$, $\sigma_{\hat{v}}^2$ 由附件 3 自身按提前量分档、

只用目标日之前历史、最近 cal_window 天滚动标定 (MSE)

'expdecay' 指数衰减 $w_v = 0.5^{** \Delta_v}$

"""

```

import os, json
import numpy as np

```

```

PUB_HOURS = (0, 6, 12, 18)

```

```

STAGE_SLOTS = (0, 36, 72, 108) # 0:00 / 6:00 / 12:00 / 18:00

```

```

SPD = 144

```

```

JAN_DAYS = 31 # 硬约束: 方差标定窗口 = 2025-01-01 .. 2025-01-31

```

```

def merge_bins(cnt, sse, min_n, lo=1, hi=24):
    """相邻提前量档贪心合并, 使每组样本数 >= min_n。 返回 (分组, 每档使用的 MSE[hi+1])。 """
    groups = []
    cur = [lo]; cn = int(cnt[lo]); cs = float(sse[lo])
    for k in range(lo + 1, hi + 1):
        if cn >= min_n:
            groups.append([cur, cn, cs]); cur = [k]; cn = int(cnt[k]); cs = float(sse[k])
        else:
            cur.append(k); cn += int(cnt[k]); cs += float(sse[k])
    groups.append([cur, cn, cs])
    if len(groups) > 1 and groups[-1][1] < min_n: # 末组样本不足则并入前一组
        last = groups.pop(); prev = groups.pop()
        groups.append([prev[0] + last[0], prev[1] + last[1], prev[2] + last[2]])
    row = np.full(hi + 1, np.nan)
    for ks, n, s in groups:
        v = (s / n) if n > 0 else 1.0
        for k in ks:
            row[k] = v
    return [g[0] for g in groups], row

class ForecastEngine:
    def __init__(self, vintages_path, pv_actual, cal_window=60, calib='fixed_jan', min_bin_n=20):
        with open(vintages_path, encoding='utf-8') as f:
            data = json.load(f)
        self.H = np.asarray(data['vintages'], float) # (D,4,24)
        self.D, self.K, self.NH = self.H.shape
        self.T = SPD
        self.ND = self.D * self.T
        self.act = np.asarray(pv_actual, float).reshape(-1)
        assert self.act.size >= self.ND
        self.cal_window = int(cal_window)
        self.calib = calib # 'fixed_jan'(正式) | 'full'(全样本, 含泄露仅供对照) | 'rolling'(稳健性对照)
        self.min_bin_n = int(min_bin_n)
        self.sigma0 = 1.0 # 数值下限(kW): 夜间预报与实际同为 0 时 MSE=0, 避免 1/0
        self._build_candidates()
        self._calibrate()

# ----- 候选版本表 -----
def _build_candidates(self):
    G = []; P = []; V = []
    for d in range(self.D):
        for k, h in enumerate(PUB_HOURS):
            p = d * self.T + h * 6
            for i in range(self.NH):
                val = float(self.H[d, k, i])
                for j in range(6):
                    g = p + 6 * i + j

```

```

        if g >= self.ND:
            break
        G.append(g); P.append(p); V.append(val)
self.G = np.asarray(G, np.int64)
self.P = np.asarray(P, np.int64)
self.V = np.asarray(V, float)
self.BIN = (self.G - self.P + 5) // 6          # ceil(Delta) in 1..24
self.cand = [[] for _ in range(self.ND)]
for g, p, v in zip(self.G.tolist(), self.P.tolist(), self.V.tolist()):
    self.cand[g].append((p, v))
for lst in self.cand:
    lst.sort()

# ----- 标定 sigma^2(Delta) -----
def _calibrate(self):
    SE = (self.V - self.act[self.G])** 2
    self.mse = np.full((self.D, 25), np.nan)
    self.cal_n = np.zeros((self.D, 25), int)
    self.mse_tod = np.full((self.D, 600), np.nan)  # (目标时刻 0..23) x (提前量档 0..24)
    self.tod = (self.G % SPD) // 6
    self.TODB = self.tod * 25 + self.BIN
    self.calib_info = {'mode': self.calib, 'min_bin_n': self.min_bin_n}
    if self.calib == 'rolling':
        self._calibrate_rolling(SE)
    else:
        self._calibrate_fixed(SE)
    self.mse[:, 0] = self.mse[:, 1]  # 档 0 不会出现, 仅防 d=0/t=0 兜底路径除零
    for tod in range(24):
        self.mse_tod[:, tod * 25] = self.mse_tod[:, tod * 25 + 1]

# ----- 固定窗口标定 (硬约束: 只用第一个月) -----
def _calibrate_fixed(self, SE):
    J0 = JAN_DAYS * self.T
    if self.calib == 'fixed_jan':
        m = (self.P < J0) & (self.G < J0)
        desc = ('2025-01-01..2025-01-31 发布的版本, 且目标时刻也落在 1 月内; '
                '1/31 各版覆盖到 2 月的目标段已剔除; 标定一次后全年固定沿用')
    elif self.calib == 'full':
        m = np.ones(self.G.shape, bool)
        desc = '全样本(334 天)标定 — 含泄露, 仅供对照, 不得作为正式结果'
    else:
        raise ValueError(self.calib)
    cnt = np.bincount(self.BIN[m], minlength=25)
    s = np.bincount(self.BIN[m], weights=SE[m], minlength=25)
    nhours = np.zeros(25, int)
    for k in range(1, 25):

```

```

        nhours[k] = np.unique(self.P[m & (self.BIN == k)]).size
    groups, row = merge_bins(cnt, s, self.min_bin_n)
    self.mse[:] = row + self.sigma0 ** 2
    self.cal_n[:] = cnt
    row2 = np.full(600, np.nan); tgroups = {}
    for tod in range(24):
        sl = slice(tod * 25, tod * 25 + 25)
        mk = m & (self.tod == tod)
        c2 = np.bincount(self.BIN[mk], minlength=25)
        s2 = np.bincount(self.BIN[mk], weights=SE[mk], minlength=25)
        if c2[1:].sum() == 0:
            row2[sl] = row[1]; tgroups[tod] = 'fallback_lead_only'; continue
        g2, r2 = merge_bins(c2, s2, self.min_bin_n)
        tgroups[tod] = g2; row2[sl] = r2
    self.mse_tod[:] = row2 + self.sigma0 ** 2
    self.calib_info.update(dict(
        desc=desc, n_days=JAN_DAYS if self.calib == 'fixed_jan' else self.D,
        n_samples_segments=int(m.sum()), n_samples_hours=int(nhours[1:].sum()),
        n_versions=int(np.unique(self.P[m]).size),
        bins={str(k): dict(n_seg=int(cnt[k]), n_hours=int(nhours[k]),
            mse=float(row[k]) if np.isfinite(row[k]) else None,
            sigma=float(np.sqrt(row[k])) if np.isfinite(row[k]) else None)
            for k in range(1, 25)},
        merged_groups=groups, tod_groups=tgroups))

# ----- 滚动标定 (稳健性对照, 非正式口径) -----
def _calibrate_rolling(self, SE):
    for d in range(self.D):
        lo = max(0, d - self.cal_window) * self.T
        m = (self.G < d * self.T) & (self.P >= lo)
        cnt = np.bincount(self.BIN[m], minlength=25)
        s = np.bincount(self.BIN[m], weights=SE[m], minlength=25)
        row = np.full(25, np.nan)
        ok = cnt > 0
        row[ok] = s[ok] / cnt[ok]
        idx = np.where(ok)[0]
        if idx.size == 0:
            # 无任何历史样本(仅 d=0): 退化为等权, 不得使用任何未来信息
            row[:] = 1.0
        else:
            for b in range(25):
                if not np.isfinite(row[b]):
                    row[b] = row[idx[np.argmin(np.abs(idx - b))]]
            self.mse[d] = row + self.sigma0 ** 2
            self.cal_n[d] = cnt
    # 按 (目标时刻, 提前量) 二维分档标定 — 消除时段混淆的对照标定
    cnt2 = np.bincount(self.TODB[m], minlength=600)

```

```

s2 = np.bincount(self.TODB[m], weights=SE[m], minlength=600)
row2 = np.full(600, np.nan)
ok2 = cnt2 > 0
row2[ok2] = s2[ok2] / cnt2[ok2]
if not ok2.any():
    row2[:] = 1.0
else:
    val_ok = row2[ok2]
    for b in range(600):
        if not np.isfinite(row2[b]):
            same_tod = np.arange(b - b % 25, b - b % 25 + 25)
            cand2 = same_tod[np.isfinite(row2[same_tod])]
            if cand2.size:
                row2[b] = row2[cand2[np.argmin(np.abs(cand2 - b))]]
            else:
                row2[b] = float(np.mean(val_ok))
self.mse_tod[d] = row2 + self.sigma0 ** 2

# ----- 生成逐时段预报 -----
def pv_vectors(self, day, scheme, boundary='strict'):
    """返回 (4,144) 预报、 (4,144) 有效提前量(h)、 (4,144) 参与版本数."""
    out = np.zeros((4, self.T)); lead = np.zeros((4, self.T)); nv = np.zeros((4, self.T), int)
    base = day * self.T
    for si, tslot in enumerate(STAGE_SLOTS):
        tau = base + tslot
        for t in range(self.T):
            g = base + t
            sel = []
            for p, v in self.cand[g]:
                if p > tau:
                    continue
                if p < g:
                    sel.append((p, v))
            elif boundary == 'full' or (boundary == 'adjust' and tslot > 0):
                sel.append((p, v))
        if not sel:
            # 仅 d=0,t=0 无任何可用版本: 退回当天 0:00 版(夜间, 值为 0)
            sel = [(base, float(self.H[day, 0, 0]))]
        if scheme == 'newest':
            pb = max(p for p, _ in sel)
            vs = [v for p, v in sel if p == pb]
            out[si, t] = float(np.mean(vs))
            lead[si, t] = (g - pb) / 6.0
        else:
            w = []; dlt = []; val = []
            for p, v in sel:
                dv = (g - p) / 6.0

```

```

        if scheme == 'invvar':
            ww = 1.0 / self.mse[day][(g - p + 5) // 6]
        elif scheme == 'invvar_tod':
            ww = 1.0 / self.mse_tod[day][((g % SPD) // 6) * 25 + (g - p + 5) // 6]
        elif scheme == 'expdecay':
            ww = 0.5 ** dv
        else:
            raise ValueError(scheme)
        w.append(ww); dlt.append(dv); val.append(v)
    w = np.asarray(w, float); dlt = np.asarray(dlt, float); val = np.asarray(val, float)
    out[si, t] = float(np.dot(w, val) / w.sum())
    lead[si, t] = float(np.dot(w, dlt) / w.sum())
    nv[si, t] = len(sel)
return out, lead, nv

def all_days(self, scheme, boundary='strict'):
    PV = np.zeros((self.D, 4, self.T)); LE = np.zeros((self.D, 4, self.T))
    NV = np.zeros((self.D, 4, self.T), int)
    for d in range(self.D):
        PV[d], LE[d], NV[d] = self.pv_vectors(d, scheme, boundary)
    return PV, LE, NV

def in_force_tables(PV, LE, d0=31, d1=365):
    """按 '最后一次决策该段' 的口径拼出实际投用的预报值/提前量."""
    n = d1 - d0
    vals = np.zeros((n, SPD)); vlead = np.zeros((n, SPD)); dlead = np.zeros((n, SPD))
    for i, d in enumerate(range(d0, d1)):
        for t in range(SPD):
            si = max(j for j, s in enumerate(STAGE_SLOTS) if s <= t)
            vals[i, t] = PV[d, si, t]
            vlead[i, t] = LE[d, si, t]
            dlead[i, t] = (t - STAGE_SLOTS[si]) / 6.0
    return vals, vlead, dlead

def stat_block(err):
    err = np.asarray(err, float)
    if err.size == 0:
        return dict(n=0, mae=float('nan'), rmse=float('nan'), mbe=float('nan'), p95=float('nan'))
    return dict(n=int(err.size), mae=float(np.mean(np.abs(err))), rmse=float(np.sqrt(np.mean(err ** 2))),
               mbe=float(np.mean(err)), p95=float(np.percentile(np.abs(err), 95)))

def accuracy_report(eng, PV, LE, actual, d0=31, d1=365):
    act = np.asarray(actual, float).reshape(-1)
    vals, vlead, dlead = in_force_tables(PV, LE, d0, d1)
    A = act[d0 * SPD:d1 * SPD].reshape(d1 - d0, SPD)
    err = vals - A
    rep = {'overall': stat_block(err)}
    rep['by_decision_lead'] = {}

```

```

dl = np.ceil(dlead).astype(int)
for k in range(0, 25):
    m = dl == k
    if m.any():
        rep['by_decision_lead'][str(k)] = stat_block(err[m])
rep['by_version_lead'] = {}
vl = np.ceil(vlead).astype(int)
for k in range(1, 25):
    m = vl == k
    if m.any():
        rep['by_version_lead'][str(k)] = stat_block(err[m])
# 相对 0:00 计划的修订幅度 (解释偏差费用的机制量), 仅在白天(PV>0)段统计
day = A > 0
rep['revision_vs_stage0'] = {}
for si, h in ((1, 6), (2, 12), (3, 18)):
    sl = STAGE_SLOTS[si]
    nxt = SPD if si == 3 else STAGE_SLOTS[si + 1]
    d = np.abs(PV[d0:d1, si, sl:nxt] - PV[d0:d1, 0, sl:nxt])
    mm = day[:, sl:nxt]
    rep['revision_vs_stage0'][str(h)] = dict(
        mean_abs=float(d.mean()) if d.size else 0.0,
        mean_abs_daytime=float(d[mm].mean()) if mm.any() else 0.0)
rep['stage_block_mae'] = {}
for si, h in ((0, 0), (1, 6), (2, 12), (3, 18)):
    sl = STAGE_SLOTS[si]; nxt = SPD if si == 3 else STAGE_SLOTS[si + 1]
    rep['stage_block_mae'][str(h)] = float(np.abs(PV[d0:d1, si, sl:nxt] - A[:, sl:nxt]).mean())
rep['stage_block_mae_all144'] = {str(h): float(np.abs(PV[d0:d1, si] - A).mean())
    for si, h in ((0, 0), (1, 6), (2, 12), (3, 18))}

return rep

def raw_version_accuracy(eng, d0=31, d1=365):
    """附件 3 各版本按提前量分档的原始 MAE/RMSE (统计期内)."""
    m = (eng.G >= d0 * SPD) & (eng.G < d1 * SPD)
    e = eng.V[m] - eng.act[eng.G[m]]
    b = eng.BIN[m]
    out = {}
    for k in range(1, 25):
        mk = b == k
        if mk.any():
            out[str(k)] = stat_block(e[mk])
    return out

def raw_version_accuracy_by_pub(eng, d0=31, d1=365):
    """按 '发布时刻' 分组的提前量->MAE/RMSE (消除不同发布时刻混在一起造成的时段混淆)."""
    m = (eng.G >= d0 * SPD) & (eng.G < d1 * SPD)
    e = eng.V[m] - eng.act[eng.G[m]]
    b = eng.BIN[m]; pp = eng.P[m]
    ph = (pp % SPD) // 6

```

```

out = {}
for hi, h in enumerate(PUB_HOURS):
    mm = ph == h
    if not mm.any():
        continue
    rec = {}
    for k in range(1, 25):
        mk = mm & (b == k)
        if mk.any():
            rec[str(k)] = stat_block(e[mk])
    out['%02d:00' % h] = rec
return out

if __name__ == '__main__':
    import argparse
    ap = argparse.ArgumentParser()
    ap.add_argument('--vintages', default=os.path.join('paper_output_pro', 'code', 'q3_vintages_A.json'))
    ap.add_argument('--input', default=os.path.join('paper_output_pro', 'data_cleaned', 'q3_input.json'))
    ap.add_argument('--out', default=os.path.join('paper_output_pro', 'experiments', 'q3-fc-e1', 'forecast_accuracy.json'))
    ap.add_argument('--cal-window', type=int, default=60)
    ap.add_argument('--min-bin-n', type=int, default=20)
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding='utf-8'))
    out = {'sigma0_kw': 1.0, 'calib_primary': 'fixed_jan', 'min_bin_n': a.min_bin_n}
    for calib in ('fixed_jan', 'full'):
        eng = ForecastEngine(a.vintages, inp['pv_actual'], a.cal_window, calib, a.min_bin_n)
        tag = 'jan' if calib == 'fixed_jan' else 'fullsample_LEAKY'
        if calib == 'fixed_jan':
            out['raw_version_by_lead'] = raw_version_accuracy(eng)
            out['raw_version_by_lead_and_pub'] = raw_version_accuracy_by_pub(eng)
        out['calibration_' + tag] = eng.calib_info
        for scheme, bnd in (('newest', 'strict'), ('invvar', 'strict'), ('invvar_tod', 'strict'),
                           ('expdecay', 'strict'), ('newest', 'adjust'), ('newest', 'full')):
            PV, LE, NV = eng.all_days(scheme, bnd)
            key = '%s|s|s|s' % (scheme, bnd, tag)
            out[key] = accuracy_report(eng, PV, LE, inp['pv_actual'])
            out[key]['n_versions_used_mean'] = float(NV[31:].mean())
    os.makedirs(os.path.dirname(os.path.abspath(a.out)), exist_ok=True)
    json.dump(out, open(a.out, 'w', encoding='utf-8'), ensure_ascii=False, indent=1)
    for k in ('newest|strict|jan', 'invvar|strict|jan', 'invvar_tod|strict|jan', 'expdecay|strict|jan',
             'newest|adjust|jan', 'newest|full|jan', 'invvar|strict|fullsample_LEAKY'):
        o = out[k]['overall']
        print('%-32s MAE %7.2f RMSE %7.2f MBE %8.2f 平均版本数 %7.2f' %
              (k, o['mae'], o['rmse'], o['mbe'], out[k]['n_versions_used_mean']))
    ci = out['calibration_jan']
    print('1 月标定: %d 个版本 / %d 个 10 分钟样本 / %d 个小时级独立样本; 分档合并组数 %d' %
          (ci['n_versions'], ci['n_samples_segments'], ci['n_samples_hours'], len(ci['merged_groups'])))
    for g in ci['merged_groups']:

```

```

    print(' 组 %s' % (g,))

    print('->' + a.out)

```

D.2.54 支撑材料/代码/03_问题三/q3_forecast_B.py

-*- coding: utf-8 -*-

"""问题三 光伏预报层 (E3/E4 专用, 后缀 _B).

严格的因果取版规则

决策时刻 dec (绝对小时) 上求解某个阶段时, 目标时段 s (第 d 天第 t 段, 起点 $d*24 + t/6$)
 可用的版本集合 = { v : 发布时刻 pub(v) <= dec 且 v 覆盖 s }

* 因子 A = 知全天 (E3/E4): dec = d*24 (0:00 计划) 时, 当天 0:00 版 pub = d*24 <= dec, 故可用;
 于是「当天 0:00 版对全部 144 段可用」自动成立。

* 因子 A = 不知 (E1/E2, 非脚本范围): dec = d*24 时 pub < dec 才可用, 当天 0:00 版被排除。

调整阶段 dec = d*24 + h (h=6/12/18), 只求解 s >= slot 的部分。

注意: 可用性按「决策时刻」判定, 而非按目标时段起点判定 — 否则 0:00 计划会偷看 6:00 的版本。

取版方式 (因子 B)

 latest : 取 pub 最晚的一版
 inv : 逆方差加权 $w_v = 1 / \sigma_{\text{hat}}(\Delta_v)^2$
 exp : 指数衰减 $w_v = \lambda^{\Delta_v}$ ($\lambda = 0.5$)
 其中 Δ_v = 版本内的小时序号 j (目标小时起点相对发布时刻的提前小时数, 0..23)。

$\sigma_{\text{hat}}(\Delta)$ 由附件 3 自身按提前量分档滚动标定: 第 d 天只用 $d_0 < d$ 的历史残差,
 样本不足 ($n < \text{min}_n$) 时退化为等权。绝不使用当天/未来实际值。

```

import os, json, argparse
import numpy as np

```

```

HOURS = (0, 6, 12, 18)
NDAY, T, NH = 365, 144, 24

```

```

def load_vintages(path):
    obj = json.load(open(path, encoding='utf-8'))
    V = np.asarray(obj['V'], float)
    assert V.shape == (NDAY, 4, NH), V.shape
    return V, obj['meta']

```

```

def hourly_actual(pv_a):
    """(365,144) 10 分钟实际功率 -> (365,24) 小时平均功率 (只用于历史误差统计/结算评估)"""
    return np.asarray(pv_a, float).reshape(NDAY, NH, 6).mean(axis=2)

```

```

def residuals(V, pv_a):
    """R[d0,h0i,j] = 预报 - 实际(小时均值); 目标越界处为 NaN。"""
    ha = hourly_actual(pv_a)
    R = np.full((NDAY, 4, NH), np.nan)
    for d0 in range(NDAY):
        for h0i, h0 in enumerate(HOURS):
            for j in range(NH):
                a = d0 * 24 + h0 + j
                if a < NDAY * 24:
                    R[d0, h0i, j] = V[d0, h0i, j] - ha[a // 24, a % 24]
    return R

```

```

def rolling_sigma(R, min_n=100, floor=1.0):

```

"""返回 sig[d] = (24,) 各提前量档的标准差, 严格因果: 只用「目标日 < d」的残差 (扩展窗)。

注意: 不能简单按「发布日 $d_0 < d$ 」筛选 — 前一日 18:00 版覆盖到第 d 天 17:00,

其残差含第 d 天的实际值, 属未来信息。故按「目标小时所属日 < d」筛选。

样本不足 ($n < \text{min_n}$) 时该档为 NaN, 调用方退化为等权。

```
"""
pools = [[] for _ in range(NDAY + 1)]
for d0 in range(NDAY):
    for h0i, h0 in enumerate(HOURS):
        for j in range(NH):
            a = d0 * 24 + h0 + j
            if a < NDAY * 24:
                rr = R[d0, h0i, j]
                if np.isfinite(rr):
                    pools[a // 24].append((j, float(rr)))
sig = np.full((NDAY, NH), np.nan)
s1 = np.zeros(NH); s2 = np.zeros(NH); cnt = np.zeros(NH)
for d in range(NDAY):
    m = cnt >= min_n
    if m.any():
        mean = np.where(m, s1 / np.maximum(cnt, 1), 0.0)
        var = np.where(m, s2 / np.maximum(cnt, 1) - mean ** 2, np.nan)
        var = np.maximum(var, floor ** 2)
        sig[d] = np.where(m, np.sqrt(var), np.nan)
    for j, r in pools[d]:
        s1[j] += r; s2[j] += r * r; cnt[j] += 1
return sig
```

```
def candidates(V, d, t, dec, scope='all'):
    """目标时段 (d,t) 在决策时刻 dec 下可用的版本列表 [(pub, j, value), ...]; scope='same-day' 只保留当天版本。"""
```

```
k = t // 6
a = d * 24 + k
out = []
for d0 in (d - 1, d):
    if d0 < 0:
        continue
    for h0i, h0 in enumerate(HOURS):
        pub = d0 * 24 + h0
        j = a - pub
        if 0 <= j <= NH - 1 and pub <= dec:
            if scope == 'same-day' and d0 != d:
                continue
            out.append((pub, j, float(V[d0, h0i, j])))
return out
```

```
def fuse(cands, mode, sig_row, lam=0.5):
    if not cands:
        return np.nan, np.nan
```

```

if mode == 'latest':
    pub, j, v = max(cands, key=lambda c: c[0])
    return v, j
ws = []
for pub, j, v in cands:
    if mode == 'inv':
        s = sig_row[j]
        w = 1.0 / (s * s) if np.isfinite(s) and s > 0 else 1.0
    elif mode == 'exp':
        w = lam ** j
    else:
        w = 1.0
    ws.append(w)
ws = np.asarray(ws, float)
if ws.sum() <= 0:
    ws = np.ones(len(cands))
wsum = ws.sum()
val = float(np.dot(ws, [c[2] for c in cands]) / wsum)
lead = float(min(c[1] for c in cands))
return val, lead

def forecast_stage(V, sig, d, dec, mode, scope='all', lam=0.5):
    """返回 (144,) 预报功率 (kW) 与 (144,) 该段所用最新成员的提前量 j; 不可用处为 NaN。"""
    arr = np.full(T, np.nan); lead = np.full(T, np.nan); nmem = np.zeros(T, int)
    sig_row = sig[d] if sig is not None else np.full(NH, np.nan)
    for t in range(T):
        c = candidates(V, d, t, dec, scope)
        nmem[t] = len(c)
        if c:
            arr[t], lead[t] = fuse(c, mode, sig_row, lam)
    return arr, lead, nmem

def stage_plan():
    """返回 365 天的四个阶段: (dec_offset, slot)。阶段 0 = 0:00 计划(全 144 段)。"""
    return [(0, 0), (6, 36), (12, 72), (18, 108)]

```

--- 加权方差标定 (硬约束: 只能用第一个月 2025-01-01..2025-01-31)

```

CAL_PUB = (0, 31)      # 参与统计的版本「发布日」区间 [lo, hi) -> 索引 0..30 = 1 月 1..31 日
CAL_TGT = (0, 31)      # 参与统计的「目标日」区间 [lo, hi)   -> 目标时刻必须落在 1 月内

```

```
def calibration_residuals(V, pv_a, pub_days=CAL_PUB, target_days=CAL_TGT):
    """返回 [(提前量 j, 残差), ...]。同时满足「发布日在标定窗口内」与「目标日在标定窗口内」。
```

注意: 1 月 31 日发布的 18:00 版会覆盖到 2 月 1 日, 这些目标日超出窗口的样本被剔除, 避免间接引入 2 月信息。

```
"""
ha = hourly_actual(pv_a)
out = []
for d0 in range(NDAY):
    if not (pub_days[0] <= d0 < pub_days[1]):
        continue
    for h0i, h0 in enumerate(HOURS):
        for j in range(NH):
            a = d0 * 24 + h0 + j
            if a >= NDAY * 24:
                continue
            td = a // 24
            if not (target_days[0] <= td < target_days[1]):
                continue
            out.append((j, float(V[d0, h0i, j] - ha[td, a % 24])))
return out
```

```
def fixed_sigma(V, pv_a, pub_days=CAL_PUB, target_days=CAL_TGT, min_n=20, floor=1.0):
    """固定窗口标定: 返回 (sig(24,), info)。标定一次后全年沿用, 不随时间滚动。
```

某档样本 < min_n 时按「相邻档位合并」处理: 从 j 向两侧同时扩展 k=1,2,... 直到合并样本 >= min_n; 若全部 24 档合并仍不足, 退化为全局标准差。合并情况记录在 info['merged_bins']。

```
"""
samples = calibration_residuals(V, pv_a, pub_days, target_days)
buckets = [[] for _ in range(NH)]
for j, r in samples:
    buckets[j].append(r)
cnt = np.array([len(b) for b in buckets], int)
sig = np.full(NH, np.nan)
merged = {}
for j in range(NH):
    if cnt[j] >= min_n:
        sig[j] = max(float(np.std(buckets[j], ddof=1)), floor)
        continue
    for k in range(1, NH):
        lo = max(0, j - k); hi = min(NH - 1, j + k)
        pool = [x for m in range(lo, hi + 1) for x in buckets[m]]
        if len(pool) >= min_n:
            sig[j] = max(float(np.std(pool, ddof=1)), floor)
            merged[str(j)] = [lo, hi, len(pool)]
            break
    else:
```

```

        allp = [x for b in buckets for x in b]
        sig[j] = max(float(np.std(allp, ddof=1)), floor)
        merged[str(j)] = [0, NH - 1, len(allp)]
    info = {'pub_days': list(pub_days), 'target_days': list(target_days),
           'n_total': int(len(samples)), 'n_by_lead': cnt.tolist(),
           'min_n': min_n, 'merged_bins': merged,
           'sigma_by_lead': [float(x) for x in sig],
           'pub_dates': '%d-%02d-%02d..%d-%02d-%02d' % (2025, 1, 1 + pub_days[0], 2025, 1, pub_days[1]),
           'tgt_dates': '%d-%02d-%02d..%d-%02d-%02d' % (2025, 1, 1 + target_days[0], 2025, 1, target_days[1])}
    return sig, info

def full_sample_sigma(V, pv_a, min_n=20, floor=1.0):
    """△ 含泄露, 仅供对照: 用全部 365 天样本标定方差。"""
    return fixed_sigma(V, pv_a, (0, NDAY), (0, NDAY), min_n=min_n, floor=floor)

def _demo():
    """自检: 打印纯粹由附件 3 得到的按提前量误差表, 以及各取版方式的实际使用提前量分布。"""
    root = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    V, meta = load_vintages(os.path.join(root, 'code', 'q3_vintages_B.json'))
    inp = json.load(open(os.path.join(root, 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
    pv_a = np.asarray(inp['pv_actual'], float)
    ha = hourly_actual(pv_a)
    print('--- 单版本按提前量的误差 (全部 365 天, 小时均值口径) ---')
    R = residuals(V, pv_a)
    for j in (0, 1, 5, 11, 17, 23):
        e = R[:, :, j].ravel(); e = e[np.isfinite(e)]
        print(' Delta=%2dh n=%4d MAE=%7.2f RMSE=%7.2f bias=%7.2f %
              (j, len(e), np.abs(e).mean(), np.sqrt((e ** 2).mean()), e.mean()))
    # 同源误差相关性: 以「目标绝对小时 a」为索引, E[a,j] = 发布时刻为 a-j 的那一版对 a 的误差
    NA = NDAY * 24
    E = np.full((NA, NH), np.nan)
    for d0 in range(NDAY):
        for h0i, h0 in enumerate(HOURS):
            for j in range(NH):
                a = d0 * 24 + h0 + j
                if a < NA:
                    E[a, j] = V[d0, h0i, j] - ha[a // 24, a % 24]
    day = np.array([ha[a // 24, a % 24] for a in range(NA)]) > 50.0
    print('--- 同源误差相关性 (同一目标小时, 两版发布相差 6h; 白天时段) ---')
    for j in (6, 12, 18, 23):
        x = E[:, j]; y = E[:, j - 6]
        m = np.isfinite(x) & np.isfinite(y) & day
        print(' Delta=%2dh vs %2dh : corr = %+.4f (n=%d, MAE %.1f / %.1f) %
              (j, j - 6, np.corrcoef(x[m], y[m])[0, 1], int(m.sum()),
                np.abs(x[m]).mean(), np.abs(y[m]).mean()))

if __name__ == '__main__':
    _demo()

```

D.2.55 支撑材料/代码/03_问题三/q3_mpc.py

-*- coding: utf-8 -*-

```

import os, json, argparse, datetime
import numpy as np
from scipy.optimize import linprog

def wcat(d): return 0 if (datetime.date(2025,1,1)+datetime.timedelta(days=d)).weekday() in (4,5) else 1
def build_seasonal(arr, D, T, alpha=0.25, beta=0.8):
    F=np.zeros_like(arr); sums=[np.zeros(T),np.zeros(T)]; cnt=[0,0]; E=arr[0].copy()
    for d in range(D):
        w=wcat(d); bl=sums[w]/cnt[w] if cnt[w]>0 else arr[d]
        F[d]=(1-beta)*E+beta*bl; sums[w]=sums[w]+arr[d]; cnt[w]+=1; E=alpha*arr[d]+(1-alpha)*E
    return F

def solve_window(price_w, nm_w, soc0, dt, eta, Pmax, soc_min, soc_max, plan_w=None):
    """plan_w 为 None 时是 0:00 计划阶段（目标只有正常购电费）；
        给定 plan_w 时为调整阶段：引入上调量 v、下调量 u，目标含偏差费用
        cost_t = price_t*p_t + 0.5*price_t*u_t + 0.5*price_t*v_t, 约束 p_t - v_t + u_t = plan_t0. """
    W=len(price_w)
    if plan_w is None:
        n=3*W+(W+1); c=np.zeros(n); c[0:W]=price_w*dt
        bnd=[(0,None)]*W+[(0,Pmax)]*W+[(0,Pmax)]*W+[(soc0,soc0)]
        for _ in range(1,W): bnd.append((soc_min,soc_max))
        bnd.append((soc_min,soc_max))
        A=np.zeros((W,n)); b=-nm_w
        for t in range(W): A[t,t]=-1; A[t,W+t]=1; A[t,2*W+t]=-1
        Ae=np.zeros((W,n))
        for t in range(W): Ae[t,3*W+t+1]=1; Ae[t,3*W+t]=-1; Ae[t,W+t]=-eta*dt; Ae[t,2*W+t]=dt/eta
        be=np.zeros(W)
    else:
        U0=3*W+(W+1); V0=U0+W; n=V0+W # 末尾追加 u(W 个) 与 v(W 个)
        c=np.zeros(n)
        c[0:W]=price_w*dt # 正常购电费
        c[U0:U0+W]=0.5*price_w*dt # 下调：违约按 50%，在 price*p 之上再加 0.5
        c[V0:V0+W]=0.5*price_w*dt # 上调：超出部分按 1.5 倍（含基础电价），故也只再加 0.5
        bnd=[(0,None)]*W+[(0,Pmax)]*W+[(0,Pmax)]*W+[(soc0,soc0)]
        for _ in range(1,W): bnd.append((soc_min,soc_max))
        bnd.append((soc_min,soc_max))
        bnd+=[(0,None)]*W+[(0,None)]*W
        A=np.zeros((W,n)); b=-nm_w
        for t in range(W): A[t,t]=-1; A[t,W+t]=1; A[t,2*W+t]=-1
        Ae=np.zeros((2*W,n)); be=np.zeros(2*W)
        for t in range(W):
            Ae[t,3*W+t+1]=1; Ae[t,3*W+t]=-1; Ae[t,W+t]=-eta*dt; Ae[t,2*W+t]=dt/eta
            Ae[W+t,t]=1.0; Ae[W+t,U0+t]=1.0; Ae[W+t,V0+t]=-1.0 # p - v + u = plan
            be[W+t]=float(plan_w[t])
    r=linprog(c,A_ub=A,b_ub=b,A_eq=Ae,b_eq=be,bounds=bnd,method="highs")
    if not r.success: raise RuntimeError(r.message)
    x=r.x; return x[0:W],x[W:2*W],x[2*W:3*W],x[3*W:3*W+W+1]

```

```

def main():
    ap=argparse.ArgumentParser(); ap.add_argument("--input",required=True); ap.add_argument("--out",required=True);
    ap.add_argument("--adjust",type=int,default=1)
    ap.add_argument("--pct",type=float,default=80.0); ap.add_argument("--window",type=int,default=28)
    ap.add_argument("--alpha",type=float,default=0.15); ap.add_argument("--beta",type=float,default=0.8)
    a=ap.parse_args()
    inp=json.load(open(a.input,encoding="utf-8"))
    price=np.asarray(inp["price"],float); load=np.asarray(inp["load_actual"],float); pv_a=np.asarray(inp["pv_actual"],float)
    st=inp["storage"]; dt=inp["dt"]; eta=st["efficiency"]; Pmax=st["power_max"]; soc_min=st["soc_min"];
    soc_max=st["soc_max"]
    D,T=load.shape; net=load-pv_a
    fc={h:np.array([inp["pv_forecast"][f"{d}_{h}"] for d in range(D)]) for h in [0,6,12,18]}
    load_f=build_seasonal(load,D,T,a.alpha,a.beta); pv_fs=build_seasonal(pv_a,D,T,a.alpha,a.beta)
    def net_fc(d,h):
        arr=fc[h][d].copy(); arr[np.isnan(arr)]=pv_fs[d][np.isnan(arr)]; return load_f[d]-arr
    pct=a.pct; W=a.window; soc_c=st["soc_initial"]; plan_c=0.0; adj_c=0.0; em_c=0.0; plans=[]
    for d in range(D):
        if d<2: margin=np.zeros(T)
        else:
            lo=max(1,d-W); errs=np.array([net[k]-net_fc(k,0) for k in range(lo,d)]);
    margin=np.maximum(np.percentile(errs,pct,axis=0),0.0)
    p_p,c_p,dd_p,s_p=solve_window(price,net_fc(d,0)+margin,soc_c,dt,eta,Pmax,soc_min,soc_max)
    Pf=p_p.copy(); Cf=c_p.copy(); Df=dd_p.copy(); soc=s_p
    if a.adjust:
        for h,slot in [(6,36),(12,72),(18,108)]:
            p_c,d_s=solve_window(price[slot:],(net_fc(d,h)+margin)[slot:],soc[slot:],dt,eta,Pmax,soc_min,soc_max,
                                plan_w=p_p[slot:])
            Pf[slot:]=p_c; Cf[slot:]=c_; Df[slot:]=d_s; soc=np.concatenate([soc[:slot],s_])
    if d>=31:
        base=price*np.minimum(p_p,Pf); dev=0.5*price*np.maximum(0,p_p-Pf)+1.5*price*np.maximum(0,Pf-p_p)
        plan_c+=float(np.sum(base)*dt); adj_c+=float(np.sum(dev)*dt)
        E=np.maximum(0.0,net[d]+Cf-Pf-Df); em_c+=5.0*float(np.dot(price,E))*dt
        plans.append(dict(day=d,purchase_plan=p_p.tolist(),purchase_adj=Pf.tolist(),
                        charge=Cf.tolist(),discharge=Df.tolist(),soc=soc.tolist(),emergency=E.tolist()))
    soc_c=float(soc[-1])
    metrics={"q3_total_cost":plan_c+adj_c+em_c,"q3_planned_cost":plan_c,"q3_adjustment_cost":adj_c,"q3_emergency_cost":em_c}
    os.makedirs(a.out,exist_ok=True)
    json.dump({"metrics":metrics},open(os.path.join(a.out,"metrics.json"),"w",encoding="utf-8"))
    json.dump({"plans":plans,"price":price.tolist(),"adjust":bool(a.adjust)},open(os.path.join(a.out,"solution.json"),"w",encoding="utf-8"))
    print(f'adjust={a.adjust}: total={plan_c+adj_c+em_c:.2f} (plan={plan_c:.2f} adj={adj_c:.2f} emerg={em_c:.2f})')
if __name__=="__main__": main()

```

D.2.56 支撑材料/代码/03_问题三/q3_realtime_solver.py

-*- coding: utf-8 -*-

"""问题三在实时执行口径(口径₀)下的重算。

口径⑤ (原 q3_mpc.py): 储能严格按计划充放电, 实际与计划的差全走紧急购电。
 口径⑥ (本脚本): 0:00 计划只锁购电量, 储能按实际缺口实时充放电, 补不上才紧急购电。
 其余(预测层、冗余、调整时刻、计费规则)与 q3_mpc.py 完全一致, 以保证口径可比。

```
"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q3_mpc import build_seasonal, solve_window
from q4_realtime_solver import write_workbook

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--input', required=True); ap.add_argument('--out', required=True)
    ap.add_argument('--adjust', type=int, default=1)
    ap.add_argument('--pct', type=float, default=80.0); ap.add_argument('--window', type=int, default=14)
    ap.add_argument('--alpha', type=float, default=0.15); ap.add_argument('--beta', type=float, default=0.8)
    ap.add_argument('--write-xlsx', default="")
    ap.add_argument('--project-root', default='..')
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding='utf-8'))
    price = np.asarray(inp['price'], float)          # 附件1 日曲线, 每天重复
    load = np.asarray(inp['load_actual'], float); pv_a = np.asarray(inp['pv_actual'], float)
    st = inp['storage']; dt = float(inp['dt']); eta = float(st['efficiency'])
    Pmax = float(st['power_max']); smin = float(st['soc_min']); smax = float(st['soc_max'])
    D, T = load.shape; net = load - pv_a
    fc = {h: np.array([inp['pv_forecast']["%d_%d" % (d, h)] for d in range(D)]) for h in (0, 6, 12, 18)}
    load_f = build_seasonal(load, D, T, a.alpha, a.beta)
    pv_fs = build_seasonal(pv_a, D, T, a.alpha, a.beta)

    def net_fc(d, h):
        arr = fc[h][d].copy(); arr[np.isnan(arr)] = pv_fs[d][np.isnan(arr)]
        return load_f[d] - arr

    soc_c = float(st['soc_initial'])
    base_c = dev_c = em_c = 0.0; em_k = cur_k = 0.0; ch = di = 0.0
    rows = []
    for d in range(D):
        if d < 2:
            margin = np.zeros(T)
        else:
            lo = max(1, d - a.window)
            errs = np.array([net[k] - net_fc(k, 0) for k in range(lo, d)])
            margin = np.maximum(np.percentile(errs, a.pct, axis=0), 0.0)
        p_p, c_p, dd_p, soc = solve_window(price, net_fc(d, 0) + margin, soc_c, dt, eta, Pmax, smin, smax)
        Pf = p_p.copy()
        if a.adjust:
            for h, slot in ((6, 36), (12, 72), (18, 108)):
                p_, c_, d_, s_ = solve_window(price[slot:], (net_fc(d, h) + margin)[slot:], soc[slot:],
                                              dt, eta, Pmax, smin, smax, plan_w=p_p[slot:])
                Pf[slot:] = p_; soc = np.concatenate([soc[:slot], s_])
```

```

# --- 实时执行 (口径②) ---
e = np.zeros(T); c2 = np.zeros(T); d2 = np.zeros(T); cu = np.zeros(T)
ss = soc_c; S = np.zeros(T + 1); S[0] = ss
for t in range(T):
    need = net[d][t] - Pf[t]
    if need > 0:
        cap = min(Pmax, max(ss - smin, 0.0) * eta / dt)
        d2[t] = min(need, cap); e[t] = need - d2[t]; ss -= d2[t] * dt / eta
    else:
        cap = min(Pmax, max(smax - ss, 0.0) / (eta * dt))
        c2[t] = min(-need, cap); cu[t] = (-need) - c2[t]; ss += eta * c2[t] * dt
    S[t + 1] = ss
if d >= 31:
    base_c += float(np.sum(price * np.minimum(p_p, Pf)) * dt)
    dev_c += float(np.sum(0.5 * price * np.maximum(0, p_p - Pf) + 1.5 * price * np.maximum(0, Pf - p_p)) * dt)
    em_c += 5.0 * float(np.dot(price, e)) * dt
    em_k += float(e.sum() * dt); cur_k += float(cu.sum() * dt)
    ch += float(c2.sum() * dt); di += float(d2.sum() * dt)
    rows.append({'day': d, 'plan0': p_p.copy(), 'plan': Pf.copy(),
                'c': c2.copy(), 'd': d2.copy(), 'E': e.copy(),
                'S0': float(soc_c), 'S1': float(S[-1])})
    soc_c = float(S[-1])
m = {'q3_total_cost': base_c + dev_c + em_c, 'q3_purchase_cost': base_c,
     'q3_adjustment_cost': dev_c, 'q3_emergency_cost': em_c, 'q3_emergency_kwh': em_k,
     'q3_curtail_kwh': cur_k, 'q3_charge_kwh': ch, 'q3_discharge_kwh': di,
     'q3_caliber': 2.0, 'q3_adjust': float(a.adjust)}
os.makedirs(a.out, exist_ok=True)
json.dump({'metrics': m}, open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'), ensure_ascii=False)
print('口径② adjust=%d: 总 %.2f= 成交 %.2f+ 偏差 %.2f+ 紧急 %.2f (%.2f kWh) 弃电 %.1f'
      % (a.adjust, m['q3_total_cost'], base_c, dev_c, em_c, em_k, cur_k))
if a.write_xlsx:
    write_workbook(os.path.abspath(a.write_xlsx), os.path.abspath(a.project_root),
                  a.adjust and 3 or 2, rows, np.tile(price, (D, 1)), dt, eta)
# 四个指定日期的紧急购电(供表 16)
if rows:
    D0 = np.array([r['day'] for r in rows])
    import datetime
    for i, r in enumerate(rows):
        dt_ = (datetime.date(2025, 1, 1) + datetime.timedelta(days=r['day']))
        if (dt_.month, dt_.day) in ((3, 20), (6, 21), (9, 23), (12, 21)):
            E = r['E']; T2 = len(E); segs = []; jj = 0
            while jj < T2:
                if E[jj] > 1e-6:
                    kk = jj
                    while kk < T2 and E[kk] > 1e-6:
                        kk += 1
                    segs.append('%d:%02d-%d:%02d(%.2f)' % (jj*10//60, jj*10%60, kk*10//60, kk*10%60,
                                                            float(E[jj:kk].sum()*dt)))

```

```

        jj = kk
    else:
        jj += 1
    print('DATE %s 紧急 %.2f kWh 时段: %s' % (dt_, float(E.sum()*dt), '；'.join(segs) if segs else '无'))

if __name__ == '__main__':
    main()

```

D.2.57 支撑材料/代码/03_问题三/q3_sens.py

import os

-*- coding: utf-8 -*-

```

"""快速敏感性：偏差系数 penalty in {0.3, 0.7}."""
import json, numpy as np
from scipy.optimize import linprog
from pathlib import Path
proj = Path(os.environ.get("CUMCM_ROOT", ".")); root = proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; D,T=365,144

def load_att1():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 1.xlsx",read_only=True,data_only=True)
    return np.array([float(r[1]) for r in list(wb["Sheet1"].iter_rows(values_only=True))[1:]])

def load_att2():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv

def load_att3():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 3.xlsx",read_only=True,data_only=True)
    rows=list(wb["Sheet1"].iter_rows(values_only=True))[1:]
    fc=np.zeros((D,4,24))
    for d in range(D):
        for f in range(4): fc[d,f,:]=[float(x) for x in rows[d*4+f][2:26]]
    return fc

price=load_att1(); load_actual,pv_actual=load_att2(); fc_forecast=load_att3()

def forecast_slots(d,f_idx):
    f=f_idx*6; fc=np.zeros(T)
    for i in range(T):
        h=(i+1)//6; k=h-f-1
        if k<0:k=0
        if k>23:k=23
        fc[i]=fc_forecast[d,f_idx,k]
    return fc

def solve_block(price_b,net_fc,soc_start,p_prev=None,penalty=0.5):
    Tb=len(price_b); price_b=np.asarray(price_b,float); net_fc=np.asarray(net_fc,float)
    has_dev=p_prev is not None
    n=(5*Tb+(Tb+1)) if has_dev else (3*Tb+(Tb+1))
    A_a=0; A_c=Tb; A_d=2*Tb; A_s=3*Tb
    c_obj=np.zeros(n); c_obj[0:Tb]=price_b*dt
    if has_dev:

```

```

A_u=3*Tb+(Tb+1);A_v=4*Tb+(Tb+1)
c_obj[A_u:A_u+Tb]=penalty*price_b*dt; c_obj[A_v:A_v+Tb]=penalty*price_b*dt
bounds=[(0,None)]*Tb+[(0,Pmax)]*Tb+[(0,Pmax)]*Tb
bounds+=[(soc_start,soc_start)]+[(soc_min,soc_max)]*(Tb-1)+[(soc_min,soc_max)]
if has_dev: bounds+=[(0,None)]*Tb+[(0,None)]*Tb
A_ub=np.zeros((Tb,n)); b_ub=-net_fc
for t in range(Tb):
    A_ub[t,A_a+t]=-1.0;A_ub[t,A_d+t]=-1.0;A_ub[t,A_c+t]=1.0
A_eq=np.zeros((Tb,n)); b_eq=np.zeros(Tb)
for t in range(Tb):
    A_eq[t,A_s+t+1]=1.0;A_eq[t,A_s+t]=-1.0;A_eq[t,A_c+t]=-eta*dt;A_eq[t,A_d+t]=dt/eta
if has_dev:
    p_prev=np.asarray(p_prev,float); A_eq2=np.zeros((Tb,n)); b_eq2=p_prev.copy()
    for t in range(Tb):
        A_eq2[t,A_a+t]=1.0;A_eq2[t,A_u+t]=-1.0;A_eq2[t,A_v+t]=1.0
        A_eq=np.vstack([A_eq,A_eq2]); b_eq=np.concatenate([b_eq,b_eq2])
res=linprog(c_obj,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bounds,method="highs")
if not res.success: raise RuntimeError(res.message)
x=res.x
return x[A_a:A_a+Tb],x[A_c:A_c+Tb],x[A_d:A_d+Tb],x[A_s:A_s+Tb+1]
def realize_greedy(a_b,load_b,pv_b,soc_start):
    soc=soc_start; E=np.zeros(len(a_b))
    for t in range(len(a_b)):
        net=load_b[t]-pv_b[t]
        if net>a_b[t]:
            d=min(Pmax,(soc-soc_min)*eta/dt,net-a_b[t]); E[t]=net-a_b[t]-d; soc-=d*dt/eta
        else:
            c=min(Pmax,(soc_max-soc)/(eta*dt),a_b[t]-net); soc+=eta*c*dt
    return E,soc

def run(penalty):
    soc=soc0; tot=[0,0,0]
    for d in range(D):
        ld=load_actual[d]; pv=pv_actual[d]
        fc0=forecast_slots(d,0); p,c0,d0,s0=solve_block(price,ld-fc0,soc,None)
        E0,s36=realize_greedy(p[0:36],ld[0:36],pv[0:36],soc)
        fc6=forecast_slots(d,1); a6,c6,d6,s6=solve_block(price[36:],ld[36:]-fc6[36:],s36,p[36:],penalty)
        E6,s72=realize_greedy(a6[0:36],ld[36:72],pv[36:72],s36)
        fc12=forecast_slots(d,2); a12,c12,d12,s12=solve_block(price[72:],ld[72:]-fc12[72:],s72,p[72:],penalty)
        E12,s108=realize_greedy(a12[0:36],ld[72:108],pv[72:108],s72)
        fc18=forecast_slots(d,3); a18,c18,d18,s18=solve_block(price[108:],ld[108:]-fc18[108:],s108,p[108:],penalty)
        E18,s144=realize_greedy(a18[0:36],ld[108:144],pv[108:144],s108)
        a=np.concatenate([p[0:36],a6[0:36],a12[0:36],a18[0:36]])
        E=np.concatenate([E0,E6,E12,E18])
        if d >= 31:
            tot[0]+=np.dot(price,a)*dt
            tot[1]+=penalty*np.dot(price,np.abs(a-p))*dt
            tot[2]+=5.0*np.dot(price,E)*dt
    soc=s144

```

```

return tot[0],tot[1],tot[2]

for pen in [0.3,0.7]:
    tp,td,te = run(pen)
    print(f'penalty={pen}: 计划={tp:.0f} 偏差={td:.0f} 紧急={te:.0f} 总={tp+td+te:.0f}')

```

D.2.58 支撑材料/代码/03_问题三/q3_stochastic.py
import os

-*- coding: utf-8 -*-

```

"""问题3 随机规划复算: 0:00 两阶段随机(场景对冲) + 6/12/18 确定性调整 + 贪婪再调度."""
import json, numpy as np
from scipy.optimize import linprog
from pathlib import Path
proj = Path(os.environ.get("CUMCM_ROOT", ".")); root = proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10000.0; soc0=6000.0; D,T=365,144
S=5; Z=[-1.5,-0.75,0.0,0.75,1.5] # 场景分位点

def load_att1():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 1.xlsx",read_only=True,data_only=True)
    return np.array([float(r[1]) for r in list(wb["Sheet1"].iter_rows(values_only=True))[1:]])

def load_att2():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv

def load_att3():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 3.xlsx",read_only=True,data_only=True)
    rows=list(wb["Sheet1"].iter_rows(values_only=True))[1:]
    fc=np.zeros((D,4,24))
    for d in range(D):
        for f in range(4): fc[d,f,:]=[float(x) for x in rows[d*4+f][2:26]]
    return fc

price=load_att1(); load_actual,pv_actual=load_att2(); fc_forecast=load_att3()

def forecast_slots(d,f_idx):
    f=f_idx*6; fc=np.zeros(T)
    for i in range(T):
        h=(i+1)//6; k=h-f-1
        if k<0:k=0
        if k>23:k=23
        fc[i]=fc_forecast[d,f_idx,k]
    return fc

```

0:00 预报误差(整年)的标准差 per slot

```

err0 = np.zeros((D,T))
for d in range(D):
    err0[d] = pv_actual[d] - forecast_slots(d,0)
sigma = err0.std(axis=0) + 1e-3 # per-slot std

def solve_stoch_0h(price_b, load_b, fc0, soc_start, sigma, S, Z):

```

```

"""两阶段随机 LP: 第一阶段 p,c,d,s 共享; 第二阶段 e^s 每场景. min  $\sum price * p + (5/S) \sum_s \sum price * e^s$ ."""
Tb=len(price_b)
pv_s = [fc0 + Z[s]*sigma for s in range(S)] # S x Tb 场景光伏
n = 3*Tb + (Tb+1) + S*Tb
A_p=0; A_c=Tb; A_d=2*Tb; A_s=3*Tb; A_e=3*Tb+(Tb+1)
c_obj=np.zeros(n); c_obj[A_p:A_p+Tb]=price_b*dt
c_obj[A_e:] = (5.0/S)*np.tile(price_b,S)*dt
bounds=[(0,None)]*Tb+[(0,Pmax)]*Tb+[(0,Pmax)]*Tb
bounds+=[(soc_start,soc_start)]+[(soc_min,soc_max)]*(Tb-1)+[(soc_min,soc_max)]
bounds+=[(0,None)]*(S*Tb)
A_ub=np.zeros((S*Tb,n)); b_ub=np.zeros(S*Tb)
row=0
for s in range(S):
    for t in range(Tb):
        A_ub[row,A_p+t]=-1.0; A_ub[row,A_d+t]=-1.0; A_ub[row,A_e+s*Tb+t]=-1.0; A_ub[row,A_c+t]=1.0
        b_ub[row]=-(load_b[t]-pv_s[s][t]); row+=1
A_eq=np.zeros((Tb,n)); b_eq=np.zeros(Tb)
for t in range(Tb):
    A_eq[t,A_s+t+1]=1.0; A_eq[t,A_s+t]=-1.0; A_eq[t,A_c+t]=-eta*dt; A_eq[t,A_d+t]=dt/eta
res=linprog(c_obj,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bounds,method="highs")
if not res.success: raise RuntimeError(res.message)
x=res.x
return x[A_p:A_p+Tb],x[A_c:A_c+Tb],x[A_d:A_d+Tb],x[A_s:A_s+Tb+1]

def solve_block(price_b,net_fc,soc_start,p_prev=None,penalty=0.5):
    Tb=len(price_b); price_b=np.asarray(price_b,float); net_fc=np.asarray(net_fc,float)
    has_dev=p_prev is not None
    n=(5*Tb+(Tb+1)) if has_dev else (3*Tb+(Tb+1))
    A_a=0; A_c=Tb; A_d=2*Tb; A_s=3*Tb
    c_obj=np.zeros(n); c_obj[0:Tb]=price_b*dt
    if has_dev:
        A_u=3*Tb+(Tb+1); A_v=4*Tb+(Tb+1)
        c_obj[A_u:A_u+Tb]=penalty*price_b*dt; c_obj[A_v:A_v+Tb]=penalty*price_b*dt
    bounds=[(0,None)]*Tb+[(0,Pmax)]*Tb+[(0,Pmax)]*Tb
    bounds+=[(soc_start,soc_start)]+[(soc_min,soc_max)]*(Tb-1)+[(soc_min,soc_max)]
    if has_dev: bounds+=[(0,None)]*Tb+[(0,None)]*Tb
    A_ub=np.zeros((Tb,n)); b_ub=-net_fc
    for t in range(Tb):
        A_ub[t,A_a+t]=-1.0; A_ub[t,A_d+t]=-1.0; A_ub[t,A_c+t]=1.0
    A_eq=np.zeros((Tb,n)); b_eq=np.zeros(Tb)
    for t in range(Tb):
        A_eq[t,A_s+t+1]=1.0; A_eq[t,A_s+t]=-1.0; A_eq[t,A_c+t]=-eta*dt; A_eq[t,A_d+t]=dt/eta
    if has_dev:
        p_prev=np.asarray(p_prev,float); A_eq2=np.zeros((Tb,n)); b_eq2=p_prev.copy()
        for t in range(Tb):
            A_eq2[t,A_a+t]=1.0; A_eq2[t,A_u+t]=-1.0; A_eq2[t,A_v+t]=1.0
        A_eq=np.vstack([A_eq,A_eq2]); b_eq=np.concatenate([b_eq,b_eq2])
    res=linprog(c_obj,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bounds,method="highs")

```

```

if not res.success: raise RuntimeError(res.message)

x=res.x

return x[A_a:A_a+Tb],x[A_c:A_c+Tb],x[A_d:A_d+Tb],x[A_s:A_s+Tb+1]

def realize_greedy(a_b,load_b,pv_b,soc_start):
    soc=soc_start; E=np.zeros(len(a_b))

    for t in range(len(a_b)):
        net=load_b[t]-pv_b[t]
        if net>a_b[t]:
            d=min(Pmax,(soc-soc_min)*eta/dt,net-a_b[t]); E[t]=net-a_b[t]-d; soc-=d*dt/eta
        else:
            c=min(Pmax,(soc_max-soc)/(eta*dt),a_b[t]-net); soc+=eta*c*dt

    return E,soc

import time
t0=time.time()
daily=[]
soc=soc0
for d in range(D):
    ld=load_actual[d]; pv=pv_actual[d]

    fc0=forecast_slots(d,0)
    p,c0,d0,s0=solve_stoch_0h(price,ld,fc0,soc,sigma,S,Z)
    E0,s36=realize_greedy(p[0:36],ld[0:36],pv[0:36],soc)
    fc6=forecast_slots(d,1); a6,c6,d6,s6=solve_block(price[36:],ld[36:]-fc6[36:],s36,p[36:])
    E6,s72=realize_greedy(a6[0:36],ld[36:72],pv[36:72],s36)
    fc12=forecast_slots(d,2); a12,c12,d12,s12=solve_block(price[72:],ld[72:]-fc12[72:],s72,p[72:])
    E12,s108=realize_greedy(a12[0:36],ld[72:108],pv[72:108],s72)
    fc18=forecast_slots(d,3); a18,c18,d18,s18=solve_block(price[108:],ld[108:]-fc18[108:],s108,p[108:])
    E18,s144=realize_greedy(a18[0:36],ld[108:144],pv[108:144],s108)
    a=np.concatenate([p[0:36],a6[0:36],a12[0:36],a18[0:36]])
    E=np.concatenate([E0,E6,E12,E18])
    daily.append(dict(day=d,p=p.tolist(),a=a.tolist(),c=np.zeros(T).tolist(),d=np.zeros(T).tolist(),E=E.tolist(),
        purchase=float(np.dot(price,a)*dt), deviation=float(0.5*np.dot(price,np.abs(a-p))*dt),
        emergency=float(5.0*np.dot(price,E)*dt), final_soc=float(s144)))
    soc=s144

res=daily[31:]
tp=sum(x["purchase"] for x in res); td=sum(x["deviation"] for x in res); te=sum(x["emergency"] for x in res)
print(f"=== 随机规划(0:00 对冲), 334 天 ===")
print(f"实际购电费 Σprice*a: {tp:.2f}")
print(f"偏差费: {td:.2f}")
print(f"紧急费: {te:.2f}")
print(f"总费用: {tp+td+te:.2f}")

```

计划购电费 $\Sigma price \cdot p$

```

tpp=sum(np.dot(price,np.array(x["p"]))*dt for x in res)
print(f"计划购电费 Σprice*p: {tpp:.2f}")
print(f"耗时 {time.time()-t0:.1f}s")
out=dict(daily=daily, summary=dict(purchase=tp,deviation=td,emergency=te,total=tp+td+te,plan=tpp))
(root/"data_cleaned/q3_stoch_cache.json").write_text(json.dumps(out),encoding="utf-8")
print("saved q3_stoch_cache.json")

```

D.2.59 支撑材料/代码/03_问题三/q3_validate_full.py

```
import os
```

-*- coding: utf-8 -*-

"""问题3 彻底独立检验:

- 1) 逐日逐时段复核功率平衡/SOC 动力学/SOC 上下限/充放功率上限/非负约束
- 2) 独立重算成本(购电费+偏差费+紧急费)并与缓存/result3.xlsx 交叉核对
- 3) 能量守恒与年度汇总核对

```

"""
import json, numpy as np, openpyxl
from pathlib import Path
proj = Path(os.environ.get("CUMCM_ROOT", ".")); root = proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; T=144

```

重新载入原始数据(独立于缓存)

```

def load_att1():
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 1.xlsx",read_only=True,data_only=True)
    return np.array([float(r[1]) for r in list(wb["Sheet1"].iter_rows(values_only=True))[1:]])
def load_att2():
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv
price=load_att1(); load_actual,pv_actual=load_att2()

d = json.loads((root/"data_cleaned/q3_result_cache.json").read_text(encoding="utf-8"))
daily = d["daily"]
cached_price = np.array(d["price"])

res=[]
def rep(name, ok, detail=""):
    res.append((name, ok, detail)); print(f'{"PASS" if ok else "FAIL"} {name} {detail}')

```

0) 价格与原始数据一致

```

rep("电价向量与附件1一致", np.allclose(cached_price, price, atol=1e-9))

```

1) 逐日约束复核 (全 365 天)

```

viol_balance=0; viol_soc=0; viol_pow=0; viol_neg=0; max_bal_gap=0.0; soc_lo=1e9; soc_hi=-1e9
max_c=0.0; max_dd=0.0
for day in daily:
    a=np.array(day["a"]); c=np.array(day["c"]); dd=np.array(day["d"]); E=np.array(day["E"])
    # 非负
    if (a<-1e-6).any() or (c<-1e-6).any() or (dd<-1e-6).any() or (E<-1e-6).any(): viol_neg+=1
    # 功率上限
    if c.max()>Pmax+1e-4 or dd.max()>Pmax+1e-4: viol_pow+=1
    max_c=max(max_c,c.max()); max_dd=max(max_dd,dd.max())
    # 真实能量平衡: a+dd+E+pv_actual >= load+c (允许弃光=> >=0 缺口)
    gap = (load_actual[day["day"]] + c) - (a + dd + E + pv_actual[day["day"]])
    max_bal_gap=max(max_bal_gap, gap.max())
    if gap.max() > 1e-4: viol_balance+=1
    # SOC 动力学复核
    s=np.zeros(T+1)
    # 起始 SOC 由累计 carryover 重建

```

SOC 逐一重建(跨日连续)

```

soc = soc0; soc_viol=0; soc_lo=1e9; soc_hi=-1e9
for day in daily:
    a=np.array(day["a"]); c=np.array(day["c"]); dd=np.array(day["d"])

```

```

s=np.zeros(T+1); s[0]=soc
for t in range(T):
    s[t+1]=s[t]+eta*c[t]*dt-dd[t]*dt/eta
if s.min()<soc_min-1e-4 or s.max()>soc_max+1e-4: soc_viol+=1
soc_lo=min(soc_lo,s.min()); soc_hi=max(soc_hi,s.max())
soc=s[-1]
if abs(soc-day["final_soc"])>1e-3: soc_viol+=1

rep("功率平衡(逐日 365 天,允许弃光)", viol_balance==0, f"违反天数={viol_balance}, 最大缺口={max_bal_gap:.2e} kWh")
rep("SOC 上下限[1200,10800]", soc_viol==0, f"违反={soc_viol}, 实测范围=[{soc_lo:.2f},{soc_hi:.2f}]")
rep("充放电功率<=5000kw", viol_pow==0, f"违反天数={viol_pow}, max_c={max_c:.1f}, max_d={max_dd:.1f}")
rep("购电/充放电/紧急非负", viol_neg==0, f"违反天数={viol_neg}")

```

2) 独立重算成本 (334 天)

```

tot_a=0.0; tot_dev=0.0; tot_em=0.0; tot_p=0.0
for day in daily[31:]:
    a=np.array(day["a"]); p=np.array(day["p"]); E=np.array(day["E"])

    tot_p += float(np.dot(price,p)*dt)
    tot_a += float(np.dot(price,a)*dt)
    tot_dev += float(0.5*np.dot(price,np.abs(a-p))*dt)
    tot_em += float(5.0*np.dot(price,E)*dt)

total = tot_a+tot_dev+tot_em
rep("独立重算总费用==13824868.16", abs(total-13824868.16)<1.0, f"重算={total:.2f}")
rep("独立重算计划购电费==12225011.04", abs(tot_p-12225011.04)<1.0, f"重算={tot_p:.2f}")
rep("独立重算偏差费==934410.32", abs(tot_dev-934410.32)<1.0, f"重算={tot_dev:.2f}")
rep("独立重算紧急费==3191.86", abs(tot_em-3191.86)<0.05, f"重算={tot_em:.2f}")

```

3) result3.xlsx 交叉核对

```

wb=openpyxl.load_workbook(root/"result3.xlsx", data_only=True)
ws=wb["计划购电量"]; wa=wb["调整购电量"]; wcd=wb["充放电量"]; we=wb["紧急购电量"]

```

逐日全天购电量与购电费

```

mismatch_p=0; mismatch_a=0
for i,day in enumerate(daily[31:]):
    r=i+2

    p_sum=ws.cell(row=r,column=146).value; p_fee=ws.cell(row=r,column=147).value
    if abs(p_sum - sum(day["p"])*dt)>0.01: mismatch_p+=1
    if abs(p_fee - float(np.dot(price,day["p"])*dt))>0.02: mismatch_p+=1
    a_sum=wa.cell(row=r,column=146).value
    if abs(a_sum - sum(day["a"])*dt)>0.01: mismatch_a+=1

rep("xlsx 计划购电量(全天)与缓存一致", mismatch_p==0, f"不一致={mismatch_p}")
rep("xlsx 调整购电量(全天)与缓存一致", mismatch_a==0, f"不一致={mismatch_a}")

```

xlsx 全天购电费合计 vs 缓存

```

xls_plan_fee=sum(ws.cell(row=i+2,column=147).value for i in range(334))
rep("xlsx 全年计划购电费合计=12225011", abs(xls_plan_fee-tot_p)<10.0, f"xlsx={xls_plan_fee:.2f}")

```

充放电量: 逐日 6 段之和 vs 缓存

```

cd_mismatch=0
for i,day in enumerate(daily[31:]):
    base=2+i*6
    chg=sum(wcd.cell(row=base+k,column=3).value for k in range(6))

```

```

dis=sum(wcd.cell(row=base+k,column=4).value for k in range(6))

if abs(chg - sum(day["c"])*dt)>0.02: cd_mismatch+=1

if abs(dis - sum(day["d"])*dt)>0.02: cd_mismatch+=1

rep("xlsx 充放电(6 段合计)与缓存一致", cd_mismatch==0, f"不一致={cd_mismatch}")

```

紧急购电量

```

em_rows=we.max_row-1
em_sum=0.0
for r in range(2,we.max_row+1):
    v=we.cell(row=r,column=3).value

    if isinstance(v,(int,float)): em_sum+=v

rep("xlsx 紧急购电量合计=670kwh", abs(em_sum-669.87)<1.0, f"xlsx={em_sum:.2f} kwh, 行数={em_rows}")

```

4) 能量守恒(全年, 独立)

```

tot_load=load_actual[31:].sum()*dt
tot_pv=pv_actual[31:].sum()*dt
tot_buy=sum(sum(day["a"])*dt for day in daily[31:])
tot_em_kwh=sum(sum(day["E"])*dt for day in daily[31:])
tot_chg=sum(sum(day["c"])*dt for day in daily[31:])
tot_dis=sum(sum(day["d"])*dt for day in daily[31:])
print(f"\n--- 全年能量账(334 天) ---")
print(f"负载={tot_load:.0f} kwh, 实际光伏={tot_pv:.0f} kwh, 购电={tot_buy:.0f}, 紧急={tot_em_kwh:.1f}")
print(f"充电={tot_chg:.0f}, 放电={tot_dis:.0f} (含效率损失的净值={eta*tot_chg-tot_dis/eta:.0f})")
net = tot_buy + tot_em_kwh + tot_pv + tot_dis - tot_load - tot_chg
rep("全年能量平衡(含弃光)残差>=0", net > -1e-3, f"净残差={net:.2f} kwh(>=0 表示弃光)")

```

```

npass=sum(1 for x in res if x[1])
print(f"\n=== 问题3 检验: {npass}/{len(res)} 通过 ===")
(root/"analysis/q3_validation_report.txt").write_text(
    "\n".join(f"'PASS' if ok else 'FAIL' {n} {dt2}" for n,ok,dt2 in res), encoding="utf-8")
print("saved analysis/q3_validation_report.txt")

```

D.2.60 支撑材料/代码/03_问题三/q3_vintages_A.py

-*- coding: utf-8 -*-

"""问题三 · 附件3 预报版本索引重建 (子代理A: 仅 E1/E2).

附件3 表头: 日期 | 预报时刻 | 预报1小时 | ... | 预报24小时 (1460 行 = 365 天 x 4 版)
 "预报N小时" = 第 N 个小时 [pub+(N-1)h, pub+N h) 的平均功率 (kW).
 => 每版覆盖 "发布时刻起连续 24 小时".
 10 分钟段 t (0..143) 取所在小时的值. 已与 data_cleaned/q3_input.json 的 0:00 版核对一致.

输出: paper_output_pro/code/q3_vintages_A.json (+ .sha256.txt)

```

"""
import os, json, hashlib, argparse
import numpy as np
import pandas as pd

```

```

PUB_HOURS = (0, 6, 12, 18)
SPD = 144 # 每天 144 个 10 分钟段

```

```

def sha256_file(path, blk=1 << 20):
    h = hashlib.sha256()

    with open(path, 'rb') as f:
        while True:
            b = f.read(blk)
            if not b:
                break
            h.update(b)
    return h.hexdigest()

```

```

def main():

```

```

ap = argparse.ArgumentParser()
ap.add_argument('--xlsx', default=os.path.join('problem_files', '附件', '附件 3.xlsx'))
ap.add_argument('--out', default=os.path.join('paper_output_pro', 'code', 'q3_vintages_A.json'))
a = ap.parse_args()

```

```

df = pd.read_excel(a.xlsx)
cols = list(df.columns)
day_col, pub_col = cols[0], cols[1]
hour_cols = cols[2:26]
assert len(hour_cols) == 24, len(hour_cols)

```

```

df[day_col] = df[day_col].ffill()
days = pd.to_datetime(df[day_col]).dt.normalize()
udays = sorted(days.unique())
assert len(udays) == 365, len(udays)

```

```

dayidx = {d: i for i, d in enumerate(udays)}
hours = np.full((365, 4, 24), np.nan)

```

```

def ph(s):
    s = str(s).strip()
    return int(s.split(':')[0])

```

```

for r in range(len(df)):
    d = dayidx[days.iloc[r]]
    h = ph(df[pub_col].iloc[r])
    k = PUB_HOURS.index(h)
    hours[d, k, :] = df.iloc[r][hour_cols].to_numpy(dtype=float)

```

```

assert np.isfinite(hours).all(), '存在缺失预报值'

```

```

# 覆盖性自检: 第 d 天 k 版覆盖全局段 [pub, pub+144)
out = {
    'source': a.xlsx.replace('\\', '/'),
    'source_sha256': sha256_file(a.xlsx),
    'n_days': 365,
    'pub_hours': list(PUB_HOURS),
    'segments_per_day': SPD,
    'dt_hours': 1.0 / 6.0,
    'convention': ('预报 N 小时 = 时段 [pub+(N-1)h, pub+N h) 的平均功率(kW); '
        '每版覆盖发布时刻起连续 24 小时; 10 分钟段取所在小时的数值'),
    'pub_seg_of_version': 'day*144 + pub_hour*6',
    'vintages': hours.tolist(),
}

```

```

    }
    os.makedirs(os.path.dirname(os.path.abspath(a.out)), exist_ok=True)
    with open(a.out, 'w', encoding='utf-8') as f:
        json.dump(out, f, ensure_ascii=False)
    h = sha256_file(a.out)
    with open(a.out + '.sha256.txt', 'w', encoding='utf-8') as f:
        f.write(h + ' ' + os.path.basename(a.out) + '\n')

# 与清洗数据交叉核对 (0:00 版必须完全一致)
q3p = os.path.join('paper_output_pro', 'data_cleaned', 'q3_input.json')
if os.path.exists(q3p):
    inp = json.load(open(q3p, encoding='utf-8'))
    ok = True
    for d in range(365):
        ref = np.asarray(inp['pv_forecast']['%d_0' % d], float)
        mine = np.repeat(hours[d, 0, :], 6)
        if not np.allclose(ref, mine, atol=1e-9):
            ok = False
            print('MISMATCH day', d)
            break
    print('交叉核对 q3_input 0:00 版: %s' % ('完全一致' if ok else '不一致'))
print('vintages -> %s sha256=%s' % (a.out, h))

if __name__ == '__main__':
    main()

```

D.2.61 支撑材料/代码/03_问题三/q3_vintages_B.py

-*- coding: utf-8 -*-

"""从原始 problem_files/附件/附件3.xlsx 重建光伏预报版本索引 (vintage index).

协议(已实测确认):

表头 = 日期 | 预报时刻 | 预报 1 小时 | ... | 预报 24 小时 ; 365 天 x 4 版 = 1460 行
「预报 N 小时」= 第 N 个小时的平均功率, 即时段 [N-1:00, N:00)
=> 第 d 天 h:00 版覆盖「d 日 h:00 起连续 24 小时」

输出 V[d0, h0i, j] = 版本 (d0, h0=HOURS[h0i]) 对「发布后第 j 个小时」的预报值 (j=0..23)
即绝对小时 a = d0*24 + h0 + j ; 若 a < 365*24 则对应 第 a//24 天第 a%24 小时。

同时落盘 SHA-256, 供复现校验。

"""

```

import os, json, hashlib, datetime, argparse
import numpy as np
import openpyxl

```

```

HOURS = (0, 6, 12, 18)
HIDX = {'0:00': 0, '6:00': 1, '12:00': 2, '18:00': 3}
NDAY = 365
NH = 24

```

```

def build(src):
    wb = openpyxl.load_workbook(src, data_only=True, read_only=True)
    ws = wb[wb.sheetnames[0]]
    rows = list(ws.values)
    V = np.full((NDAY, 4, NH), np.nan)

```

```

dates = []
di = -1
for r in rows[1:]:
    dtxt = (r[0] or "").strip()
    if dtxt:
        y, m, dd = [int(x) for x in dtxt.split('-')]
        dates.append(datetime.date(y, m, dd)); di += 1
    h = HIDX[r[1].strip()]
    V[di, h, :] = [float(x) if x is not None else np.nan for x in r[2:26]]
if V.shape != (NDAY, 4, NH) or np.isnan(V).any():
    raise RuntimeError('附件 3 解析异常: shape=%s nan=%d' % (V.shape, int(np.isnan(V).sum())))
if len(dates) != NDAY or dates[0] != datetime.date(2025, 1, 1) or dates[-1] != datetime.date(2025, 12, 31):
    raise RuntimeError('日期序列异常: n=%d %s..%s' % (len(dates), dates[0], dates[-1]))
wb.close()
return V, dates

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--src', default='problem_files/附件/附件 3.xlsx')
    ap.add_argument('--out', default='paper_output_pro/code/q3_vintages_B.json')
    a = ap.parse_args()
    V, dates = build(a.src)
    sha_src = hashlib.sha256(open(a.src, 'rb').read()).hexdigest()
    obj = {
        'meta': {
            'source': a.src, 'source_sha256': sha_src,
            'hours': list(HOURS), 'nday': NDAY, 'nhour': NH,
            'date_first': dates[0].isoformat(), 'date_last': dates[-1].isoformat(),
            'semantics': 'V[d0,h0i,j] = 附件 3 第 d0 天 HOURS[h0i]:00 版的「预报(j+1)小时」, '
                        '覆盖绝对小时 d0*24 + HOURS[h0i] + j (自发布时刻起连续 24 小时)',
        },
        'V': V.tolist(),
    }
    txt = json.dumps(obj, ensure_ascii=False)
    os.makedirs(os.path.dirname(a.out), exist_ok=True)
    open(a.out, 'w', encoding='utf-8').write(txt)
    sha_out = hashlib.sha256(txt.encode('utf-8')).hexdigest()
    print('wrote %s shape=%s json_sha256=%s src_sha256=%s' % (a.out, V.shape, sha_out, sha_src))
    print('reload check:', np.allclose(np.array(json.loads(txt)['V']), V))

if __name__ == '__main__':
    main()

```

D.2.62 支撑材料/代码/04_问题四/q4_fc_solver.py

-*- coding: utf-8 -*-

"""问题四求解器：在附件 4 的波动电价下重算问题二（4-2）与问题三（4-3）。

与问题二/三的关系

q4-2 = 问题二的信息结构 + 附件 4 的电价: 0:00 一次决策, 负荷与光伏都靠历史自预测, 电价同样靠历史自预测; 结算用真实电价, 紧急购电按真实电价 5 倍。

q4-3 = 问题三的信息结构 + 附件 4 的电价: 0/6/12/18 四次决策, 光伏预报取附件 3 的多版本逆方差加权融合 (与问题三正式口径一致), 电价仍靠历史自预测; 三项费用完整写进调整阶段的目标函数。

因果性

- build_seasonal 的第 d 天预测只用第 d-1 天及更早的数据;
- 融合用到的版本必须"发布时刻 <= 决策时刻";
- 电价预测在第 d 天 0:00 做出后当天不再修改 (题目未给任何电价预报, 日内无新信息)。

用法

```
python q4_fc_solver.py --mode q4-2 --out <目录>
python q4_fc_solver.py --mode q4-3 --sigma q3-fc-e3v2-sigma.npy --out <目录>
"""
import os, sys, json, argparse, datetime, hashlib
import numpy as np

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from q3_mpc import build_seasonal, solve_window # noqa: E402
from q3_forecast_B import load_vintages, candidates, HOURS, NDAY, NH, T # noqa: E402

STAGES = [(0, 0), (6, 36), (12, 72), (18, 108)]

def realtime(net_w, Pf_w, S0, dt, eta, Pmax, smin, smax):
    """口径② 实时执行: 储能按实际缺口实时充放电, 补不上才紧急购电。"""
    n = len(net_w); c = np.zeros(n); d2 = np.zeros(n); e = np.zeros(n); cu = np.zeros(n)
    S = np.zeros(n + 1); S[0] = S0; ss = S0
    for t in range(n):
        need = net_w[t] - Pf_w[t]
        if need > 0:
            cap = min(Pmax, max(ss - smin, 0.0) * eta / dt)
            d2[t] = min(need, cap); e[t] = need - d2[t]; ss -= d2[t] * dt / eta
        else:
            cap = min(Pmax, max(smax - ss, 0.0) / (eta * dt))
            c[t] = min(-need, cap); cu[t] = (-need) - c[t]; ss += eta * c[t] * dt
        S[t + 1] = ss
    return c, d2, e, cu, S

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--mode', required=True, choices=['q4-2', 'q4-3'])
    ap.add_argument('--out', required=True)
    ap.add_argument('--input', default=os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json'))
    ap.add_argument('--vintages', default=os.path.join(HERE, 'q3_vintages_B.json'))
    ap.add_argument('--sigma', default="", help='q4-3 用的 sigma 表 (365,24,24) .npy')
    ap.add_argument('--scope', default='all', choices=['all', 'same-day'])
    ap.add_argument('--pct', type=float, default=80.0)
    ap.add_argument('--window', type=int, default=14)
    ap.add_argument('--alpha', type=float, default=0.15, help='负荷/光伏预测的 EMA 衰减')
    ap.add_argument('--beta', type=float, default=0.8)
    ap.add_argument('--alpha-p', type=float, default=0.15, dest='alpha_p', help='电价预测的 EMA 衰减')
    ap.add_argument('--beta-p', type=float, default=0.8, dest='beta_p')
```

```

ap.add_argument('--score-from', type=int, default=31, dest='score_from')
ap.add_argument('--stop-day', type=int, default=0, dest='stop_day')
ap.add_argument('--replace-actual-from', type=int, default=0, dest='replace_actual_from')
a = ap.parse_args()

```

```

inp = json.load(open(a.input, encoding='utf-8'))
price_a = np.asarray(inp['price_actual'], float)
load = np.asarray(inp['load_actual'], float)
pv_a = np.asarray(inp['pv_actual'], float)
st = inp['storage']; dt = float(inp['dt'])
eta = float(st['efficiency']); Pmax = float(st['power_max'])
smin = float(st['soc_min']); smax = float(st['soc_max'])
if a.replace_actual_from:
    pv_a = pv_a.copy(); pv_a[a.replace_actual_from:] = 0.0
D = load.shape[0]
if a.stop_day: D = min(D, a.stop_day)
net = load - pv_a

```

```

# ----- 预报表（全部因果） -----
price_f = build_seasonal(price_a, load.shape[0], T, a.alpha_p, a.beta_p)
load_f = build_seasonal(load, load.shape[0], T, a.alpha, a.beta)
pv_fs = build_seasonal(pv_a, load.shape[0], T, a.alpha, a.beta)

```

```

V = SIG = None
if a.mode == 'q4-3':
    assert a.sigma and os.path.isfile(a.sigma), 'q4-3 需要 --sigma 表'
    SIG = np.load(a.sigma)
    assert SIG.shape[1:] == (24, NH), SIG.shape
    V, _ = load_vintages(a.vintages)

```

```

def pv_stage(d, s):
    """返回 (144,) 光伏预报、(144,) 有效提前量、(144,) 成员数"""
    if a.mode == 'q4-2':
        return pv_fs[d].copy(), np.full(T, np.nan), np.where(np.isfinite(pv_fs[d]), 1, 0)
    off, slot = STAGES[s]
    dec = d * 24 + off
    arr = np.full(T, np.nan); lead = np.full(T, np.nan); nmem = np.zeros(T, int)
    row = SIG[min(d, SIG.shape[0] - 1)]
    for t in range(T):
        c = candidates(V, d, t, dec, a.scope)
        nmem[t] = len(c)
        if not c: continue
        th = t // 6
        ws = np.asarray([1.0 / row[th, j] ** 2 for _, j, _ in c], float)
        vals = np.asarray([v for _, v in c], float)

```

```

arr[t] = float(np.dot(ws, vals) / ws.sum())
lead[t] = float(min(j for _, j, _ in c))
return arr, lead, nmem

```

```
def net_fc(d, s):
```

```

    arr, _, _ = pv_stage(d, s)
    miss = ~np.isfinite(arr)
    arr = arr.copy(); arr[miss] = pv_fs[d][miss]
    return load_f[d] - arr

```

```
# ----- 主循环 -----
```

```

soc_c = float(st['soc_initial'])
base_c = dev_c = em_c = 0.0
em_k = cur_k = ch = di = 0.0
n_adj = n_over = n_under = n_same = n_seg = 0
win = []; fc_err = []; rows = []
for d in range(D):
    if d < 2:
        margin = np.zeros(T)
    else:
        lo = max(1, d - a.window)
        errs = np.array([net[k] - net_fc(k, 0) for k in range(lo, d)])
        margin = np.maximum(np.percentile(errs, a.pct, axis=0), 0.0)
    pf = price_f[d]
    p_p, c_, d_, soc = solve_window(pf, net_fc(d, 0) + margin, soc_c, dt, eta, Pmax, smin, smax)
    Pf = p_p.copy()
    if a.mode == 'q4-3':
        for s, (h, slot) in enumerate(STAGES[1:], start=1):
            S0 = float(soc[slot])
            p_, c_, d_, s_ = solve_window(pf[slot:], (net_fc(d, s) + margin)[slot:], S0, dt, eta,
                                         Pmax, smin, smax, plan_w=p_p[slot:])
            pw = pf[slot:]; planw = p_p[slot:]; netw = net[d][slot:]
            _, _, eA, _, _ = realtime(netw, p_, S0, dt, eta, Pmax, smin, smax)
            _, _, eB, _, _ = realtime(netw, planw, S0, dt, eta, Pmax, smin, smax)
            devw = float(np.sum(0.5 * pw * np.maximum(0, planw - p_) + 1.5 * pw * np.maximum(0, p_ - planw)) * dt)
            save_base = float(np.sum(pw * np.maximum(0, planw - p_) * dt)
            save_em = 5.0 * float(np.dot(pw, eB - eA)) * dt
            if d >= a.score_from:
                win.append({'day': d, 'stage': h, 'slot': slot, 'save_purchase': save_base,
                           'save_emergency': save_em, 'dev_fee': devw,
                           'net_benefit': save_base + save_em - devw,
                           'changed_segments': int(np.sum(np.abs(p_ - planw) > 1e-6)),
                           'em_kwh_plan': float(eB.sum() * dt), 'em_kwh_adj': float(eA.sum() * dt)})
            Pf[slot:] = p_; soc = np.concatenate([soc[:slot], s_])
# ---- 实时执行（口径②，结算用真实电价） ----
c2, d2, e, cu, S = realtime(net[d], Pf, soc_c, dt, eta, Pmax, smin, smax)

```

```

if d >= a.score_from:
    pa = price_a[d]
    base_c += float(np.sum(pa * np.minimum(p_p, Pf)) * dt)
    dev_c += float(np.sum(0.5 * pa * np.maximum(0, p_p - Pf) + 1.5 * pa * np.maximum(0, Pf - p_p)) * dt)
    em_c += 5.0 * float(np.dot(pa, e)) * dt
    em_k += float(e.sum() * dt); cur_k += float(cu.sum() * dt)
    ch += float(c2.sum() * dt); di += float(d2.sum() * dt)
    dP = Pf - p_p
    n_adj += int(np.sum(np.abs(dP) > 1e-6)); n_over += int(np.sum(dP > 1e-6))
    n_under += int(np.sum(dP < -1e-6)); n_same += int(np.sum(np.abs(dP) <= 1e-6))
    n_seg += T
    for s, (h, slot) in enumerate(STAGES):
        arr, lead, nm = pv_stage(d, s)
        for t in np.where(np.arange(T) >= slot)[0]:
            if np.isfinite(arr[t]) and np.isfinite(lead[t]):
                fc_err.append((float(lead[t]), float(arr[t]), float(pv_a[d, t])))
        rows.append({'day': d, 'plan0': p_p.copy(), 'plan': Pf.copy(), 'c': c2.copy(),
                    'd': d2.copy(), 'E': e.copy(), 'S0': float(soc_c), 'S1': float(S[-1])})
    soc_c = float(S[-1])

wa = np.array([w['net_benefit'] for w in win]) if win else np.zeros(0)
neg = wa < 0
e_err = np.array([(f - t) for _, f, t in fc_err]) if fc_err else np.zeros(0)
pe = (price_f[np.arange(a.score_from, D)] - price_a[np.arange(a.score_from, D)]) if D > a.score_from else np.zeros((0, T))
m = {'q4_total_cost': base_c + dev_c + em_c, 'q4_purchase_cost': base_c, 'q4_adjust_cost': dev_c,
     'q4_emergency_cost': em_c, 'q4_emergency_kwh': em_k, 'q4_curtail_kwh': cur_k,
     'q4_charge_kwh': ch, 'q4_discharge_kwh': di, 'q4_caliber': 2.0, 'q4_adjust': 1.0 if a.mode == 'q4-3' else 0.0}
dg = {'mode': a.mode, 'scope': a.scope,
      'price_forecast': {'alpha': a.alpha_p, 'beta': a.beta_p,
                        'mae': float(np.abs(pe).mean()) if pe.size else None,
                        'rmse': float(np.sqrt((pe ** 2).mean())) if pe.size else None,
                        'bias': float(pe.mean()) if pe.size else None},
      'net_forecast': {'alpha': a.alpha, 'beta': a.beta,
                      'n': int(e_err.size),
                      'mae': float(np.abs(e_err).mean()) if e_err.size else None,
                      'rmse': float(np.sqrt((e_err ** 2).mean())) if e_err.size else None,
                      'bias': float(e_err.mean()) if e_err.size else None},
      'dev_over_purchase': dev_c / base_c if base_c else None,
      'adjusted_segments': n_adj, 'total_segments': n_seg,
      'segments_over_bought': n_over, 'segments_under_bought': n_under, 'segments_unchanged': n_same,
      'n_windows': len(win),
      'window_net_benefit_sum': float(wa.sum()) if wa.size else 0.0,
      'windows_negative_net': int(neg.sum()),
      'windows_negative_share': float(neg.mean()) if neg.size else None,
      'sigma_table': os.path.basename(a.sigma) if a.sigma else None,
      'leaky': False}
os.makedirs(a.out, exist_ok=True)

```

```

json.dump({'metrics': m, 'diagnostics': dg}, open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'),
          ensure_ascii=False, indent=1)
json.dump(win, open(os.path.join(a.out, 'windows.json'), 'w', encoding='utf-8'), ensure_ascii=False)
json.dump({'days': [r['day'] for r in rows],
          'plan0': [r['plan0'].tolist() for r in rows], 'plan': [r['plan'].tolist() for r in rows],
          'c': [r['c'].tolist() for r in rows], 'd': [r['d'].tolist() for r in rows],
          'E': [r['E'].tolist() for r in rows], 'S0': [r['S0'] for r in rows], 'S1': [r['S1'] for r in rows]},
          open(os.path.join(a.out, 'solution.json'), 'w', encoding='utf-8'))
json.dump({'argv': sys.argv, 'cwd': os.getcwd(), 'mode': a.mode, 'scope': a.scope,
          'created_at_utc': datetime.datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ'),
          'params': {'pct': a.pct, 'window': a.window, 'alpha': a.alpha, 'beta': a.beta,
                     'alpha_p': a.alpha_p, 'beta_p': a.beta_p, 'score_from': a.score_from}},
          open(os.path.join(a.out, 'receipt.json'), 'w', encoding='utf-8'), ensure_ascii=False, indent=1)
print('%s[%s]: 总 %0.2f = 成交 %0.2f + 偏差 %0.2f + 紧急 %0.2f (%0.2f kWh) 弃电 %0.1f 调整段 %d/%d 负收益窗口 %d/%d'
      % (a.mode, a.scope, m['q4_total_cost'], base_c, dev_c, em_c, em_k, cur_k, n_adj, n_seg,
         int(neg.sum()), len(win)))

if __name__ == '__main__':
    main()

```

D.2.63 支撑材料/代码/04_问题四/q4_realtime_solver.py

-*- coding: utf-8 -*-

```

"""问题四 result4-2 / result4-3 在口径2(储能实时执行)下的重算。

口径2: 0:00 计划只锁计划购电量; 储能按实际缺口实时动作; 补不上按 5 倍真实电价紧急购电。
mode 2: 只做 0:00 计划(用预测电价), 不调整
mode 3: 在 6/12/18 用更新的光伏预报调整计划, 计费按题目规则(成交按正常价, 偏差 0.5/1.5 倍)
"""
import os, sys, json, argparse
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q4_solver import seasonal, solve_w

PERIODS = ["0:00-4:00", "4:00-8:00", "8:00-12:00", "12:00-16:00", "16:00-20:00", "20:00-24:00"]

def _hmm(m):
    return "%d:%02d" % (m // 60, m % 60)

def _fill_matrix(ws, rows, key, price, plan_key=None):
    """10 分钟矩阵表。

    调整购电量表的“全天购电费”必须写入题目规定的成交费用和调整费用：
 $\lambda \cdot P_{adj} + 0.5 \lambda \cdot |P_{adj} - P_{plan}|$ 。紧急购电费用保留在紧急购电量表，
    因此不写入此列。
    """
    for i, r_ in enumerate(rows):
        rr = i + 2
        v = r_[key]
        pr = price[r_["day"]]
        for t in range(len(v)):
            ws.cell(row=rr, column=2 + t, value=round(float(v[t]) * dt_global, 4))

```

```

ws.cell(row=rr, column=2 + len(v), value=round(float(sum(v) * dt_global), 4))
cost = float(np.dot(pr, v) * dt_global)
if plan_key is not None:
    cost += 0.5 * float(np.dot(pr, np.abs(v - r_[plan_key]))) * dt_global
ws.cell(row=rr, column=3 + len(v), value=round(cost, 2))

def write_workbook(path, project_root, mode, rows, price_actual, dt, eta):
    """按附件 5 模板写出 result4-2 / result4-3 (口径@: 充放电量为实时执行轨迹)."""
    import openpyxl, datetime
    global dt_global
    dt_global = dt
    tpl = "result4-%d.xlsx" % mode
    src = os.path.join(project_root, "problem_files", "附件", "附件 5", tpl)
    wb = openpyxl.load_workbook(src)
    _fill_matrix(wb["计划购电量"], rows, "plan0" if mode == 3 else "plan", price_actual)
    if mode == 3:
        _fill_matrix(wb["调整购电量"], rows, "plan", price_actual, plan_key="plan0")
    ws = wb["充放电电量"]
    ws.delete_rows(2, ws.max_row)
    rr = 2
    for i, r_ in enumerate(rows):
        date = datetime.datetime(2025, 2, 1) + datetime.timedelta(days=i)
        for pi, lbl in enumerate(PERIODS):
            lo, hi = pi * 24, (pi + 1) * 24
            ws.cell(row=rr, column=1, value=date if pi == 0 else None)
            ws.cell(row=rr, column=2, value=lbl)
            ws.cell(row=rr, column=3, value=round(float(r_["c"][lo:hi].sum()) * dt, 4))
            ws.cell(row=rr, column=4, value=round(float(r_["d"][lo:hi].sum()) * dt, 4))
            if pi == 0:
                ws.cell(row=rr, column=5, value=datetime.time(0, 0))
                ws.cell(row=rr, column=6, value=round(r_["S0"], 4))
            elif pi == 1:
                ws.cell(row=rr, column=5, value="24:00")
                ws.cell(row=rr, column=6, value=round(r_["S1"], 4))
            rr += 1
    ws = wb["紧急购电量"]
    ws.delete_rows(2, ws.max_row)
    rr = 2
    T = len(rows[0]["plan"])
    for i, r_ in enumerate(rows):
        E = r_["E"]; date = datetime.datetime(2025, 2, 1) + datetime.timedelta(days=i)
        runs = []; j = 0
        while j < T:
            if E[j] > 1e-6:
                k = j
                while k < T and E[k] > 1e-6:
                    k += 1
                runs.append((_hhmm(j * 10), _hhmm(k * 10), round(float(E[j:k].sum()) * dt, 4))); j = k

```

```

else:
    j += 1
if runs:
    for ri, (aa, bb, en) in enumerate(runs):
        ws.cell(row=rr, column=1, value=date if ri == 0 else None)
        ws.cell(row=rr, column=2, value="%s-%s" % (aa, bb)); ws.cell(row=rr, column=3, value=en); rr += 1
else:
    ws.cell(row=rr, column=1, value=date); ws.cell(row=rr, column=3, value=0.0); rr += 1
wb.save(path)
print("SAVED", path)

dt_global = 1.0 / 6.0

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--mode", type=int, default=2)
    ap.add_argument("--pct", type=float, default=88.0); ap.add_argument("--window", type=int, default=7)
    ap.add_argument("--alpha", type=float, default=0.15); ap.add_argument("--beta", type=float, default=0.8)
    ap.add_argument("--write-xlsx", default="") # 非空则写出附件 5 结果工作簿(result4-2 / result4-3)
    ap.add_argument("--project-root", default="..")
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    pa = np.asarray(inp["price_actual"], float)
    load = np.asarray(inp["load_actual"], float); pv = np.asarray(inp["pv_actual"], float)
    st = inp["storage"]; dt = float(inp["dt"]); eta = float(st["efficiency"])
    Pmax = float(st["power_max"]); smin = float(st["soc_min"]); smax = float(st["soc_max"])
    D, T = load.shape; net = load - pv
    fc = {h: np.array([inp["pv_forecast"][f"{d}_{h}"] for d in range(D)]) for h in (0, 6, 12, 18)}
    lf = seasonal(load, D, T, a.alpha, a.beta); pfs = seasonal(pv, D, T, a.alpha, a.beta)
    pf = seasonal(pa, D, T, a.alpha, a.beta)
    def nf(d, h=0):
        arr = fc[h][d].copy(); arr[np.isnan(arr)] = pfs[d][np.isnan(arr)]; return lf[d] - arr
    s0 = float(st["soc_initial"]); base_c = dev_c = em_c = 0.0; em_k = 0.0; cur_k = 0.0
    rows = [] # 统计期逐日轨迹, 供写工作簿
    for d in range(D):
        if d < 2:
            m = np.zeros(T)
        else:
            lo = max(1, d - a.window); errs = np.array([net[k] - nf(k, 0) for k in range(lo, d)])
            m = np.maximum(np.percentile(errs, a.pct, axis=0), 0.0)
            p_p, c_p, dd_p, _ = solve_w(pf[d], nf(d, 0) + m, s0, dt, eta, Pmax, smin, smax)
            Pf = p_p.copy()
        if a.mode == 3:
            soc_tmp = s0 + np.concatenate([[0], np.cumsum(eta * c_p * dt - dd_p * dt / eta)][:-1])
            for h, slot in ((6, 36), (12, 72), (18, 108)):
                p_, c_, d_, s_ = solve_w(pf[d][slot:], (nf(d, h) + m)[slot:], soc_tmp[slot:], dt, eta, Pmax,
                    smin, smax, plan_w=p_p[slot:])

```

```

    Pf[slot:] = p_; soc_tmp = np.concatenate([soc_tmp[:slot], s_])
# 实时执行(口径②): 用调整后的计划 Pf 作为"计划购电量", 储能按实际缺口动作
e = np.zeros(T); c2 = np.zeros(T); d2 = np.zeros(T); cu = np.zeros(T)
ss = s0; S = np.zeros(T + 1); S[0] = ss
for t in range(T):
    need = net[d][t] - Pf[t]
    if need > 0:
        cap = min(Pmax, max(ss - smin, 0.0) * eta / dt)
        d2[t] = min(need, cap); e[t] = need - d2[t]; ss -= d2[t] * dt / eta
    else:
        cap = min(Pmax, max(smax - ss, 0.0) / (eta * dt))
        c2[t] = min(-need, cap); cu[t] = (-need) - c2[t]; ss += eta * c2[t] * dt
    S[t + 1] = ss
if d >= 31:
    pday = pa[d]
    base_c += float(np.sum(pday * np.minimum(p_p, Pf)) * dt)
    dev_c += float(np.sum(0.5 * pday * np.maximum(0, p_p - Pf) + 1.5 * pday * np.maximum(0, Pf - p_p)) * dt)
    em_c += 5.0 * float(np.dot(pday, e)) * dt; em_k += float(e.sum() * dt); cur_k += float(cu.sum() * dt)
    rows.append({'day': d, "plan0": p_p.copy(), "plan": Pf.copy(),
                "c": c2.copy(), "d": d2.copy(), "E": e.copy(),
                "S0": float(s0), "S1": float(S[-1])})
    s0 = float(S[-1])
m = {"q4_total_cost": base_c + dev_c + em_c, "q4_purchase_cost": base_c,
     "q4_adjustment_cost": dev_c, "q4_emergency_cost": em_c, "q4_emergency_kwh": em_k,
     "q4_curtail_kwh": cur_k, "q4_mode": float(a.mode)}
os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": m}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"), ensure_ascii=False)
print("mode=%d 口径②: 总 %.2f = 成交 %.2f + 偏差 %.2f + 紧急 %.2f (%.2f kWh) 弃电 %.1f" %
      (a.mode, m["q4_total_cost"], base_c, dev_c, em_c, em_k, cur_k))
if a.write_xlsx:
    write_workbook(os.path.abspath(a.write_xlsx), os.path.abspath(a.project_root),
                   a.mode, rows, pa, dt, eta)

if __name__ == "__main__":
    main()

```

D.2.64 支撑材料/代码/04_问题四/q4_season_chain.py

-*- coding: utf-8 -*-

"""问题四的问题 2 部分(波动电价下的 0:00 计划) 四季独立建模对照。

原題: 問題 4 要求"根据附件 2 的负载与光伏、附件 4 的电价数据, 在波动电价下重新计算问题 2".
故: 计划用预测电价(只用当天之前的历史电价), 结算用真实电价; 紧急购电 5 倍真实电价。

两种储能执行口径都算:

```

planned (口径①) 储能严格按计划充放电, 实际与计划的差全走紧急购电 -> 与交付的 result4-2 一致
realtime (口径②) 储能按实际缺口实时动作, 补不上才紧急购电 -> 与问题二主结果方法论一致
块划分与问题二相同: 冬 A 2 月 | 春 3-5 | 夏 6-8 | 秋 9-11 | 冬 B 12 月
参数口径: global(7,88) / prior(7,80) / oracle(本块自选, 含泄露上界) / prev(上一块, 无泄露) / cum(之前全部, 无泄露)
"""
import os, sys, json, argparse, datetime
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from q4_solver import seasonal, solve_w

```

```

GRID = [(W, p) for W in (3, 5, 7, 10, 14) for p in (80.0, 88.0, 95.0)]
D0 = datetime.date(2025, 1, 1)
BLOCKS = [("winter_a", 31, 59), ("spring", 59, 151), ("summer", 151, 243),
          ("autumn", 243, 334), ("winter_b", 334, 365)]

class Sim:
    def __init__(self, inp, alpha=0.15, beta=0.8):
        self.price_a = np.asarray(inp["price_actual"], float)
        L = np.asarray(inp["load_actual"], float); PV = np.asarray(inp["pv_actual"], float)
        self.net = L - PV
        st = inp["storage"]; self.dt = float(inp["dt"]); self.eta = float(st["efficiency"])
        self.Pmax = float(st["power_max"]); self.smin = float(st["soc_min"])
        self.smax = float(st["soc_max"]); self.s0 = float(st["soc_initial"])
        D, T = L.shape
        fc = {h: np.array([inp["pv_forecast"][f"{d}_{h}"] for d in range(D)]) for h in (0, 6, 12, 18)}
        self.load_f = seasonal(L, D, T, alpha, beta)
        self.pv_fs = seasonal(PV, D, T, alpha, beta)
        self.price_f = seasonal(self.price_a, D, T, alpha, beta)
        self.fc = fc

    def net_fc(self, d, h=0):
        arr = self.fc[h][d].copy(); arr[np.isnan(arr)] = self.pv_fs[d][np.isnan(arr)]
        return self.load_f[d] - arr

    def run(self, first, last, W, pct, mode, s_start=None, acc_from=None):
        """mode: 'planned'=□徑① / 'realtime'=□徑②."""
        dt, eta, Pmax = self.dt, self.eta, self.Pmax
        smin, smax = self.smin, self.smax
        T = self.net.shape[1]
        s = self.s0 if s_start is None else s_start
        acc_from = first if acc_from is None else acc_from
        base_c = em_c = plan_kwh = em_kwh = cur_kwh = 0.0
        smin_seen, smax_seen = 1e18, -1e18
        for d in range(first, last):
            if d < 2:
                margin = np.zeros(T)
            else:
                lo = max(1, d - W); errs = np.array([self.net[k] - self.net_fc(k, 0) for k in range(lo, d)])
                margin = np.maximum(np.percentile(errs, pct, axis=0), 0.0)
                g, cc, dd, _ = solve_w(self.price_f[d], self.net_fc(d, 0) + margin, s, dt, eta, Pmax, smin, smax)
                if mode == "planned":
                    e = np.maximum(0.0, self.net[d] + cc - g - dd); cur = np.zeros(T); s_end = float(_[-1])
                    s_traj = _
                else:
                    e = np.zeros(T); c2 = np.zeros(T); d2 = np.zeros(T); cur = np.zeros(T)
                    ss = s; S = np.zeros(T + 1); S[0] = ss
                    for t in range(T):

```

```

        need = self.net[d][t] - g[t]
        if need > 0:
            cap = min(Pmax, max(ss - smin, 0.0) * eta / dt)
            d2[t] = min(need, cap); e[t] = need - d2[t]; ss -= d2[t] * dt / eta
        else:
            cap = min(Pmax, max(smax - ss, 0.0) / (eta * dt))
            c2[t] = min(-need, cap); cur[t] = (-need) - c2[t]; ss += eta * c2[t] * dt
        S[t + 1] = ss
        s_end = ss; s_traj = S
    if d >= acc_from:
        pa = self.price_a[d]
        base_c += float(np.sum(pa * g) * dt); plan_kwh += float(g.sum() * dt)
        em_c += 5.0 * float(np.dot(pa, e)) * dt; em_kwh += float(e.sum() * dt)
        cur_kwh += float(cur.sum() * dt)
        smin_seen = min(smin_seen, float(np.min(s_traj))); smax_seen = max(smax_seen, float(np.max(s_traj)))
    s = s_end
    return {"total": base_c + em_c, "plan_cost": base_c, "em_cost": em_c,
            "plan_kwh": plan_kwh, "em_kwh": em_kwh, "curtail": cur_kwh,
            "soc_min": smin_seen, "soc_max": smax_seen, "soc_end": s}

def warm_soc(self, mode):
    return self.run(1, 31, 7, 88.0, mode, None, acc_from=31)["soc_end"]

def best_on(self, first, last, mode, s_start, grid=None):
    best = None
    for W, p in (grid or GRID):
        r = self.run(first, last, W, p, mode, s_start)
        if best is None or r["total"] < best[2]:
            best = (W, p, r["total"])
    return best

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", required=True); ap.add_argument("--out", required=True)
    ap.add_argument("--mode", default="matrix", choices=["matrix", "sweep"])
    a = ap.parse_args()
    inp = json.load(open(a.input, encoding="utf-8"))
    sim = Sim(inp)
    metrics = {}; report = {}

    def blocks(kind):
        return BLOCKS if kind == "seas" else [("year", 1, 365)]

    def chain(kind, mode, table):
        s_cur = None if kind == "year" else sim.warm_soc(mode)
        tot = {"plan_cost": 0.0, "em_cost": 0.0, "plan_kwh": 0.0, "em_kwh": 0.0, "curtail": 0.0,

```

```

        "soc_min": 1e18, "soc_max": -1e18}
per = {}
for key, f, l in blocks(kind):
    W, p = (table.get(key) or table.get("default"))
    acc = 31 if kind == "year" else None
    r = sim.run(f, l, W, p, mode, s_cur, acc_from=acc)
    per[key] = {"W": W, "p": p, **r}
    for k in ("plan_cost", "em_cost", "plan_kwh", "em_kwh", "curtail"):
        tot[k] += r[k]
    tot["soc_min"] = min(tot["soc_min"], r["soc_min"]); tot["soc_max"] = max(tot["soc_max"], r["soc_max"])
    s_cur = r["soc_end"]
tot["total"] = tot["plan_cost"] + tot["em_cost"]
return tot, per

def walk(kind, mode, how, grid=None):
    tb = {}; hist = []; s_cur = None if kind == "year" else sim.warm_soc(mode)
    for key, f, l in blocks(kind):
        if how == "oracle":
            W, p, c = sim.best_on(f, l, mode, s_cur, grid)
        elif how == "prev" and hist:
            _, f2, l2, s2 = hist[-1]; W, p, c = sim.best_on(f2, l2, mode, s2, grid)
        elif how == "cum" and hist:
            _, f0, _, s0 = hist[0]; W, p, c = sim.best_on(f0, f, mode, s0, grid)
        else:
            W, p = (7, 80.0)
            tb[key] = (W, p)
            s_start = s_cur
            s_cur = sim.run(f, l, W, p, mode, s_cur)["soc_end"]
            hist.append((key, f, l, s_start))
    return tb

if a.mode == "sweep":
    # 单组合全期扫描 + 调参期选参 + 检验期核验(与问题二同一套协议)
    for mode in ("planned", "realtime"):
        rows = []
        for W, p in GRID:
            rf = sim.run(1, 365, W, p, mode, None, acc_from=31)
            rt = sim.run(1, 243, W, p, mode, None, acc_from=31)
            rh = sim.run(1, 365, W, p, mode, None, acc_from=243)
            rows.append({"W": W, "p": p, "full": rf["total"], "tune": rt["total"], "hold": rh["total"]})
            metrics["q4sw_%s_w%d_p%02d_full" % (mode, W, round(p))] = rf["total"]
            metrics["q4sw_%s_w%d_p%02d_tune" % (mode, W, round(p))] = rt["total"]
            metrics["q4sw_%s_w%d_p%02d_hold" % (mode, W, round(p))] = rh["total"]
            bf = min(rows, key=lambda r: r["full"]); bt = min(rows, key=lambda r: r["tune"]); bh = min(rows, key=lambda r:
r["hold"])
            metrics["q4sw_%s_bestfull_W" % mode] = float(bf["W"]); metrics["q4sw_%s_bestfull_p" % mode] = bf["p"]

```

```

metrics["q4sw_%s_bestfull" % mode] = bf["full"]
metrics["q4sw_%s_besttune_W" % mode] = float(bt["W"]); metrics["q4sw_%s_besttune_p" % mode] = bt["p"]
metrics["q4sw_%s_besttune_full" % mode] = bt["full"]
metrics["q4sw_%s_besthold_W" % mode] = float(bh["W"]); metrics["q4sw_%s_besthold_p" % mode] = bh["p"]
metrics["q4sw_%s_besthold_full" % mode] = bh["full"]
metrics["q4sw_%s_delivered_w7p88" % mode] = [r["full"] for r in rows if r["W"] == 7 and r["p"] == 88.0][0]
report["sweep_" + mode] = rows

print("[sweep %s] 全期最优 W=%d p=%0.0f -> %0.2f | 调参期最优 W=%d p=%0.0f -> 全期 %0.2f | 检验期最优 W=%d p=%0.0f -> 全期 %0.2f" % (
    mode, bf["W"], bf["p"], bf["full"], bt["W"], bt["p"], bt["full"], bh["W"], bh["p"], bh["full"]))
print("    交付组合 (7,88) 全期 %0.2f" % metrics["q4sw_%s_delivered_w7p88" % mode])

os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics, "report": report}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
    ensure_ascii=False, indent=1)
print("done sweep")
return

```

```

for mode in ("planned", "realtime"):
    variants = [{"year", "global", {"default": (7, 88.0)}},
        ("seas", "global", {"default": (7, 88.0)}},
        ("seas", "prior", {"default": (7, 80.0)})]
    for kind in ("seas",):
        variants.append((kind, "oracle", walk(kind, mode, "oracle")))
        variants.append((kind, "prev", walk(kind, mode, "prev")))
        variants.append((kind, "cum", walk(kind, mode, "cum")))
    for kind, params, table in variants:
        tot, per = chain(kind, mode, table)
        key = "%s_%s_%s" % (mode, kind, params)
        metrics["q4_%s_total" % key] = tot["total"]
        metrics["q4_%s_plan" % key] = tot["plan_cost"]
        metrics["q4_%s_em" % key] = tot["em_cost"]
        metrics["q4_%s_em_kwh" % key] = tot["em_kwh"]
        metrics["q4_%s_curtail" % key] = tot["curtail"]
        metrics["q4_%s_soc_min" % key] = tot["soc_min"]
        metrics["q4_%s_soc_max" % key] = tot["soc_max"]
        report[key] = {"table": {k: list(v) for k, v in table.items()}, "total": tot["total"], "per_block": per}
        print("%-28s 总费用 %12.2f 计划 %12.2f 紧急 %10.2f" % (key, tot["total"], tot["plan_cost"], tot["em_cost"]))

```

```

os.makedirs(a.out, exist_ok=True)
json.dump({"metrics": metrics, "report": report}, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
    ensure_ascii=False, indent=1)
print("done")

```

```

if __name__ == "__main__":
    main()

```

D.2.65 支撑材料/代码/04_问题四/q4_solve.py

```
import os
```

-*- coding: utf-8 -*-

```
"""问题4：波动电价(附件4)下重算问题2(Q4-2)与问题3(Q4-3)."""
import json, numpy as np
from scipy.optimize import linprog
from pathlib import Path
proj = Path(os.environ.get("CUMCM_ROOT", ".")); root = proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; D,T=365,144

def load_att2():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv

def load_att3():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 3.xlsx",read_only=True,data_only=True)
    rows=list(wb["Sheet1"].iter_rows(values_only=True))[1:]
    fc=np.zeros((D,4,24))
    for d in range(D):
        for f in range(4): fc[d,f,:]=[float(x) for x in rows[d*4+f][2:26]]
    return fc

def load_att4():
    import openpyxl
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 4.xlsx",read_only=True,data_only=True)
    ws=wb["Sheet1"]
    rows=list(ws.iter_rows(min_row=2,max_row=366,max_col=145,values_only=True))
    pr=np.array([[float(c) for c in r[1:]] for r in rows],dtype=float)
    return pr

load_actual,pv_actual=load_att2(); fc_forecast=load_att3(); prices4=load_att4()
print("att2",load_actual.shape,"att3",fc_forecast.shape,"att4",prices4.shape)
print("att4 price range:", prices4.min(), prices4.max(), "mean", prices4.mean())

def forecast_slots(d,f_idx):
    f=f_idx*6; fc=np.zeros(T)
    for i in range(T):
        h=(i+1)//6; k=h-f-1
        if k<0:k=0
        if k>23:k=23
        fc[i]=fc_forecast[d,f_idx,k]
    return fc

def solve_block(price_b,net_fc,soc_start,p_prev=None,penalty=0.5):
    Tb=len(price_b); price_b=np.asarray(price_b,float); net_fc=np.asarray(net_fc,float)
    has_dev=p_prev is not None
    n=(5*Tb+(Tb+1)) if has_dev else (3*Tb+(Tb+1))
    A_a=0;A_c=Tb;A_d=2*Tb;A_s=3*Tb
    c_obj=np.zeros(n); c_obj[0:Tb]=price_b*dt
    if has_dev:
        A_u=3*Tb+(Tb+1);A_v=4*Tb+(Tb+1)
        c_obj[A_u:A_u+Tb]=penalty*price_b*dt; c_obj[A_v:A_v+Tb]=penalty*price_b*dt
    bounds=[(0,None)]*Tb+[(0,Pmax)]*Tb+[(0,Pmax)]*Tb
    bounds+=[(soc_start,soc_start)]+[(soc_min,soc_max)]*(Tb-1)+[(soc_min,soc_max)]
    if has_dev: bounds+=[(0,None)]*Tb+[(0,None)]*Tb
```

```

A_ub=np.zeros((Tb,n)); b_ub=-net_fc
for t in range(Tb):
    A_ub[t,A_a+t]=-1.0;A_ub[t,A_d+t]=-1.0;A_ub[t,A_c+t]=1.0
A_eq=np.zeros((Tb,n)); b_eq=np.zeros(Tb)
for t in range(Tb):
    A_eq[t,A_s+t]=1.0;A_eq[t,A_s+t]=-1.0;A_eq[t,A_c+t]=-eta*dt;A_eq[t,A_d+t]=dt/eta
if has_dev:
    p_prev=np.asarray(p_prev,float); A_eq2=np.zeros((Tb,n)); b_eq2=p_prev.copy()
    for t in range(Tb):
        A_eq2[t,A_a+t]=1.0;A_eq2[t,A_u+t]=-1.0;A_eq2[t,A_v+t]=1.0
    A_eq=np.vstack([A_eq,A_eq2]); b_eq=np.concatenate([b_eq,b_eq2])
res=linprog(c_obj,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bounds,method="highs")
if not res.success: raise RuntimeError(res.message)
x=res.x
return x[A_a:A_a+Tb],x[A_c:A_c+Tb],x[A_d:A_d+Tb],x[A_s:A_s+Tb+1]
def realize_greedy(a_b,load_b,pv_b,soc_start):
    soc=soc_start; c_rt=np.zeros(len(a_b)); d_rt=np.zeros(len(a_b)); E=np.zeros(len(a_b))
    for t in range(len(a_b)):
        net=load_b[t]-pv_b[t]
        if net>a_b[t]:
            d=min(Pmax,(soc-soc_min)*eta/dt,net-a_b[t]); E[t]=net-a_b[t]-d; soc-=d*dt/eta; d_rt[t]=d
        else:
            c=min(Pmax,(soc_max-soc)/(eta*dt),a_b[t]-net); soc+=eta*c*dt; c_rt[t]=c
    return c_rt,d_rt,E,soc

```

----- Q4-2: 波动电价 + 完美信息(附件 2 实际)

```

soc=soc0; q42=[]
for d in range(D):
    pr=prices4[d]; ld=load_actual[d]; pv=pv_actual[d]
    p,_,_=solve_block(pr, ld-pv, soc, None)
    c_rt,d_rt,E,soc144=realize_greedy(p, ld, pv, soc)
    q42.append(dict(day=d,p=p.tolist(),c=c_rt.tolist(),d=d_rt.tolist(),E=E.tolist(),
        planned=float(np.dot(pr,p)*dt), emergency=float(5.0*np.dot(pr,E)*dt), final_soc=float(soc144)))
    soc=float(soc144)
r=q42[31:]
q42p=sum(x["planned"] for x in r); q42e=sum(x["emergency"] for x in r)
print(f"\n[Q4-2 完美信息] 计划购电费={q42p:.2f} 紧急费={q42e:.2f} 合计={q42p+q42e:.2f} 紧急量={sum(float(np.sum(x['E'])*dt) for
x in r):.2f} kWh")

```

----- Q4-3: 波动电价 + 0/6/12/18 预报调整 -----

```

soc=soc0; q43=[]
for d in range(D):
    pr=prices4[d]; ld=load_actual[d]; pv=pv_actual[d]
    fc0=forecast_slots(d,0)
    p,c0,d0,s0=solve_block(pr, ld-fc0, soc, None)
    c0r,d0r,E0,s36=realize_greedy(p[0:36], ld[0:36], pv[0:36], soc)
    fc6=forecast_slots(d,1); a6,c6,d6,s6=solve_block(pr[36:], ld[36:]-fc6[36:], s36, p[36:])

```

```

c6r,d6r,E6,s72=realize_greedy(a6[0:36], ld[36:72], pv[36:72], s36)
fc12=forecast_slots(d,2); a12,c12,d12,s12=solve_block(pr[72:], ld[72:]-fc12[72:], s72, p[72:])
c12r,d12r,E12,s108=realize_greedy(a12[0:36], ld[72:108], pv[72:108], s72)
fc18=forecast_slots(d,3); a18,c18,d18,s18=solve_block(pr[108:], ld[108:]-fc18[108:], s108, p[108:])
c18r,d18r,E18,s144=realize_greedy(a18[0:36], ld[108:144], pv[108:144], s108)
a=np.concatenate([p[0:36],a6[0:36],a12[0:36],a18[0:36]])
c=np.concatenate([c0r,c6r,c12r,c18r]); dd=np.concatenate([d0r,d6r,d12r,d18r]); E=np.concatenate([E0,E6,E12,E18])
q43.append(dict(day=d,p=p.tolist(),a=a.tolist(),c=c.tolist(),d=dd.tolist(),E=E.tolist(),
    purchase=float(np.dot(pr,a)*dt), deviation=float(0.5*np.dot(pr,np.abs(a-p))*dt),
    emergency=float(5.0*np.dot(pr,E)*dt), final_soc=float(s144)))
soc=float(s144)

r=q43[31:]
q43a=sum(x["purchase"] for x in r); q43d=sum(x["deviation"] for x in r); q43e=sum(x["emergency"] for x in r)
q43p=sum(float(np.dot(prices4[x["day"]],x["p"]))*dt for x in r)
print(f"[Q4-3 有调整] 计划购电费={q43p:.2f} 实际购电费={q43a:.2f} 偏差费={q43d:.2f} 紧急费={q43e:.2f} 总={q43a+q43d+q43e:.2f}")
print(f"    紧急量={sum(float(np.sum(x['E'])*dt for x in r):.2f) kWh}")

```

无调整基线(波动电价)

```

soc=soc0; no=[]
for d in range(D):
    pr=prices4[d]; ld=load_actual[d]; pv=pv_actual[d]
    fc0=forecast_slots(d,0)
    p,_,_=solve_block(pr, ld-fc0, soc, None)
    c_rt,d_rt,E,soc144=realize_greedy(p, ld, pv, soc)
    no.append(dict(day=d,p=p.tolist(),a=p.tolist(),c=c_rt.tolist(),d=d_rt.tolist(),E=E.tolist(),
        planned=float(np.dot(pr,p)*dt), emergency=float(5.0*np.dot(pr,E)*dt), final_soc=float(soc144)))
    soc=float(soc144)

rn=no[31:]
q43no=sum(x["planned"] for x in rn)+sum(x["emergency"] for x in rn)
print(f"[Q4-3 无调整基线] 总={q43no:.2f} (计划={sum(x['planned'] for x in rn):.2f} 紧急={sum(x['emergency'] for x in rn):.2f}")
print(f"    调整节省={q43no-(q43a+q43d+q43e):.2f} 元 ({(1-(q43a+q43d+q43e)/q43no)*100:.1f}%)")

out=dict(prices4=prices4.tolist(), q42=q42, q43=q43, q43_no=no,
    summary=dict(q42_planned=q42p,q42_emergency=q42e,q42_total=q42p+q42e,
        q43_plan=q43p,q43_purchase=q43a,q43_deviation=q43d,q43_emergency=q43e,q43_total=q43a+q43d+q43e,
        q43_no_adjust=q43no))
(root/"data_cleaned/q4_result_cache.json").write_text(json.dumps(out),encoding="utf-8")
print("saved q4_result_cache.json")

```

D.2.66 支撑材料/代码/04_问题四/q4_solver.py

-*- coding: utf-8 -*-

```

import os, json, argparse, datetime
import numpy as np
from scipy.optimize import linprog

def wcat(d): return 0 if (datetime.date(2025,1,1)+datetime.timedelta(days=d)).weekday() in (4,5) else 1
def seasonal(arr, D, T, alpha=0.25, beta=0.8):
    F=np.zeros_like(arr); sums=[np.zeros(T),np.zeros(T)]; cnt=[0,0]; E=arr[0].copy()
    for d in range(D):
        w=wcat(d); bl=sums[w]/cnt[w] if cnt[w]>0 else arr[d]
        F[d]=(1-beta)*E+beta*bl; sums[w]=sums[w]+arr[d]; cnt[w]+=1; E=alpha*arr[d]+(1-alpha)*E
    return F

def solve_w(price_w, nm_w, soc0, dt, eta, Pmax, soc_min, soc_max, plan_w=None):
    """plan_w 为 None 时是计划阶段；给定 plan_w 时为调整阶段：

```

引入上调量 v 、下调量 u ，目标含偏差费用，约束 $p - v + u = \text{plan}_0$ 。"""

```
W=len(price_w)
if plan_w is None:
    n=3*W+(W+1); c=np.zeros(n); c[0:W]=price_w*dt
    bnd=[(0,None)]*W+[(0,Pmax)]*W+[(0,Pmax)]*W+[(soc0,soc0)]
    for _ in range(1,W): bnd.append((soc_min,soc_max))
    bnd.append((soc_min,soc_max))
    A=np.zeros((W,n)); b=-nm_w
    for t in range(W): A[t,t]=-1; A[t,W+t]=1; A[t,2*W+t]=-1
    Ae=np.zeros((W,n)); be=np.zeros(W)
    for t in range(W): Ae[t,3*W+t+1]=1; Ae[t,3*W+t]=-1; Ae[t,W+t]=-eta*dt; Ae[t,2*W+t]=dt/eta
else:
    U0=3*W+(W+1); V0=U0+W; n=V0+W
    c=np.zeros(n); c[0:W]=price_w*dt
    c[U0:U0+W]=0.5*price_w*dt # 下调：违约 50%，在 price*p 之上再加 0.5
    c[V0:V0+W]=0.5*price_w*dt # 上调：超出部分 1.5 倍（含基础电价），故也只再加 0.5
    bnd=[(0,None)]*W+[(0,Pmax)]*W+[(0,Pmax)]*W+[(soc0,soc0)]
    for _ in range(1,W): bnd.append((soc_min,soc_max))
    bnd.append((soc_min,soc_max)); bnd+=[(0,None)]*W+[(0,None)]*W
    A=np.zeros((W,n)); b=-nm_w
    for t in range(W): A[t,t]=-1; A[t,W+t]=1; A[t,2*W+t]=-1
    Ae=np.zeros((2*W,n)); be=np.zeros(2*W)
    for t in range(W):
        Ae[t,3*W+t+1]=1; Ae[t,3*W+t]=-1; Ae[t,W+t]=-eta*dt; Ae[t,2*W+t]=dt/eta
        Ae[W+t,t]=1.0; Ae[W+t,U0+t]=1.0; Ae[W+t,V0+t]=-1.0
        be[W+t]=float(plan_w[t])
r=linprog(c,A_ub=A,b_ub=b,A_eq=Ae,b_eq=be,bounds=bnd,method="highs")
if not r.success: raise RuntimeError(r.message)
x=r.x; return x[0:W],x[W:2*W],x[2*W:3*W],x[3*W:3*W+W+1]
```

```
def main():
    ap=argparse.ArgumentParser(); ap.add_argument("--input",required=True); ap.add_argument("--out",required=True);
    ap.add_argument("--mode",type=int,default=2)
    ap.add_argument("--pct",type=float,default=80.0); ap.add_argument("--window",type=int,default=28)
    ap.add_argument("--alpha",type=float,default=0.15); ap.add_argument("--beta",type=float,default=0.8)
    a=ap.parse_args()
    inp=json.load(open(a.input,encoding="utf-8"))
    price_a=np.asarray(inp["price_actual"],float); load=np.asarray(inp["load_actual"],float);
    pv_a=np.asarray(inp["pv_actual"],float)
    st=inp["storage"]; dt=inp["dt"]; eta=st["efficiency"]; Pmax=st["power_max"]; soc_min=st["soc_min"];
    soc_max=st["soc_max"]
    D,T=load.shape; net=load-pv_a
    fc={h:np.array([inp["pv_forecast"][f"{d}_{h}"] for d in range(D)]) for h in [0,6,12,18]}
    load_f=seasonal(load,D,T,a.alpha,a.beta); pv_fs=seasonal(pv_a,D,T,a.alpha,a.beta);
    price_f=seasonal(price_a,D,T,a.alpha,a.beta)
    def net_fc(d,h):
        arr=fc[h][d].copy(); arr[np.isnan(arr)]=pv_fs[d][np.isnan(arr)]; return load_f[d]-arr
    pct=a.pct; W=a.window; soc_c=st["soc_initial"]; base_c=0.0; dev_c=0.0; em_c=0.0; plans=[]
```

```

for d in range(D):
    if d<2: margin=np.zeros(T)
    else:
        lo=max(1,d-W); errs=np.array([net[k]-net_fc(k,0) for k in range(lo,d)]);
margin=np.maximum(np.percentile(errs,pct,axis=0),0.0)
# plan uses price FORECAST
p_p,c_p,dd_p,s_p=solve_w(price_f[d], net_fc(d,0)+margin, soc_c, dt,eta,Pmax,soc_min,soc_max)
Pf=p_p.copy(); Cf=c_p.copy(); Df=dd_p.copy(); soc=s_p.copy()
if a.mode==3:
    for h,slot in [(6,36),(12,72),(18,108)]:
        p_c,d_s=solve_w(price_f[d][slot:],(net_fc(d,h)+margin)[slot:],soc[slot],dt,eta,Pmax,soc_min,soc_max,
            plan_w=p_p[slot:])
        Pf[slot:]=p_c; Cf[slot:]=c; Df[slot:]=d_s; soc=np.concatenate([soc[:slot],s_])
if d>=31:
    pa=price_a[d] # settlement uses ACTUAL price
    base=pa*np.minimum(p_p,Pf); dev=0.5*pa*np.maximum(0,p_p-Pf)+1.5*pa*np.maximum(0,Pf-p_p)
    base_c+=float(np.sum(base)*dt); dev_c+=float(np.sum(dev)*dt)
    E=np.maximum(0.0, net[d]+Cf-Pf-Df); em_c+=5.0*float(np.dot(pa,E))*dt
    plans.append(dict(day=d,purchase_plan=p_p.tolist(),purchase_adj=Pf.tolist(),
        charge=Cf.tolist(),discharge=Df.tolist(),soc=soc.tolist(),emergency=E.tolist()))
    soc_c=float(soc[-1])
metrics={"q4_total_cost":base_c+dev_c+em_c,"q4_purchase_cost":base_c,"q4_adjustment_cost":dev_c,"q4_emergency_co
st":em_c}
os.makedirs(a.out,exist_ok=True)
json.dump({"metrics":metrics},open(os.path.join(a.out,"metrics.json"),"w",encoding="utf-8"))
json.dump({"plans":plans,"price_actual":price_a.tolist(),"mode":a.mode},open(os.path.join(a.out,"solution.json"),"w",encod
ing="utf-8"))
print(f"Q4 mode={a.mode}: total={base_c+dev_c+em_c:.2f} (purchase={base_c:.2f} adj={dev_c:.2f}
emerg={em_c:.2f})")
if __name__=="__main__": main()

```

D.2.67 支撑材料/代码/04_问题四/q4_strict.py

-*- coding: utf-8 -*-

"""问题四统一严格口径求解器 (Q4-strict)。

=====

一、信息协议 (严格执行, 问题 4-2 / 4-3 共用)

=====

决策时刻 $dec = d*24 + h$, $h \in \{0\}$ (q4-2) 或 $\{0,6,12,18\}$ (q4-3)。

第 d 天 $0:00$ 的可用信息集 $I(d)$ 只能是:

- (1) 负荷 $L_{hat_d} <-$ 附件 2 「小区负载」中 日期 $< d$ 的全部实际值;
- (2) 光伏 $Phat_d <-$ 附件 3 的版本融合: 只取 发布时刻 $pub(v) \leq dec$ 且覆盖目标时段的版本,
权重 $w_v = 1/\sigma_{hat}^2$ (目标小时, 提前量), σ_{hat} 由
「目标日 $\leq d-1$ 」的残差标定 (扩展窗, 零自由参数);
- (3) 电价 $lamhat_d <-$ 附件 4 中 日期 $< d$ 的全部实际值; 题目未给电价预报, 日内无新信息,
故 $0:00$ 定稿后当天不再修改;
- (4) 净负荷 $N_{hat} = L_{hat} - Phat$; 冗余裕量 $margin$ 由 $< d$ 的 W 天滚动误差的 p 分位给出;
- (5) 储电量 SOC 由 $< d$ 的决策链决定 (第 0 天 = 6000 kWh, 题目给定)。

附件 4 第 d 天的电价在 d 日 $0:00$ 一律不可知, 只有结算才用真实电价 $price_a[d]$;

附件 2 第 d 天的负荷/光伏在 d 日 $0:00$ 一律不可知, 只在实时执行与结算环节出现。

冷启动: 第 0 天 (2025-01-01) 没有任何历史, 取题目给出的附件 1 典型日曲线作为唯一先验;

自第 1 天起预测面板完全由附件 2/附件 4 的历史驱动，附件 1 不再参与。

```
=====
二、常数标定（全部只用 1 月数据）
=====
* 负荷 (alpha_L, beta_L, 星期分组) -- 1 月窗口上负荷一步预测 MAE 最小
* 电价 (alpha_p, beta_p, 星期分组) -- 1 月窗口上电价一步预测 MAE 最小
* 冗余 (W, p) -- 1 月窗口 (1/2-1/31, 30 天) 上 4-2 总费用最小
* sigma_hat (仅 4-3) -- 扩展窗，零自由参数（与问题三 E3v2 完全一致）
标定窗口 = 第 1..30 天；第 0 天为无历史的冷启动日，不参与标定。
```

```
=====
三、计费口径
=====
q4-2: 总额 = sum_t lam_t * p_plan_t * dt + 5 * sum_t lam_t * E_t * dt
(问题二规则: 除紧急购电外, 其他时段一律按计划购电量计费)

q4-3: 总额 = sum_t lam_t * min(p_plan, p_adj) * dt
+ sum_t [0.5*lam_t*u_t + 1.5*lam_t*v_t] * dt
+ 5 * sum_t lam_t * E_t * dt

u = max(0, p_plan - p_adj) 违约 (50%)
v = max(0, p_adj - p_plan) 超发 (150%)
储能: S_{t+1} = S_t + eta*c_t*dt - d_t*dt/eta, eta = 0.9 (单向), 往返 0.81。
实时执行口径 (2): 0:00 只锁计划购电量, 储能实时充放, 补不上才紧急购电, 弃电不退款。
```

```
用法
----
python q4_strict.py --task calib --out <dir>
python q4_strict.py --task solve --mode q4-2 --const <calib.json> --out <dir>
python q4_strict.py --task leaktest --const <calib.json>
python q4_strict.py --task panel --const <calib.json>
"""
import os, sys, json, argparse, datetime, hashlib
import numpy as np

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from q3_mpc import solve_window # noqa: E402
from q3_forecast_B import load_vintages, candidates, HOURS, NDAY, NH, T # noqa: E402

DT = 1.0 / 6.0
STAGES2 = [(0, 0)]
STAGES3 = [(0, 0), (6, 36), (12, 72), (18, 108)]
SIGMA_DEFAULT = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q3-fc-e3v2-sigma.npy')

CLS_CHOICES = ('fs', 'ss', 'none') # 周五六 / 周六日 / 不分星期
ALPHA_GRID = (0.05, 0.10, 0.15, 0.20, 0.30, 0.40)
BETA_GRID = (0.5, 0.6, 0.7, 0.8, 0.9, 1.0)
W_GRID = (3, 5, 7, 10, 14, 21, 28)
P_GRID = (0.50, 0.60, 0.70, 0.80, 0.88, 0.90, 0.95)
```

基础

```
def wcat(d, cls):
    wd = (datetime.date(2025, 1, 1) + datetime.timedelta(days=int(d))).weekday()
    if cls == 'fs':
        return 0 if wd in (4, 5) else 1
    if cls == 'ss':
        return 0 if wd in (5, 6) else 1
    if cls == 'none':
        return 0
    raise ValueError(cls)

def panel(arr, cold, alpha, beta, cls, D=None):
    """严格因果的多步预测面板。
```

F[d] 只使用 arr[0..d-1]; 第 0 天没有任何历史, 取题目给定的典型日先验 cold。
故 arr[d] 及其之后的值永远不会进入 F[d]。

"""

```
D = arr.shape[0] if D is None else D
Tn = arr.shape[1]
F = np.zeros((D, Tn))
sums = np.zeros((2, Tn))
cnt = np.zeros(2, dtype=int)
E = None
for d in range(D):
    w = wcat(d, cls)
    if E is None:          # d == 0: 无历史
        F[d] = cold
    else:
        bl = sums[w] / cnt[w] if cnt[w] > 0 else E
        F[d] = (1.0 - beta) * E + beta * bl
    sums[w] += arr[d]
    cnt[w] += 1
    E = arr[d].copy() if E is None else alpha * arr[d] + (1.0 - alpha) * E
return F
```

```
def panel_evolve(arr, cold, alphas, betas, cls, D=None):
    """参数逐日变化的因果预测面板 (扩展窗重标定用)。alphas[d]/betas[d] 必须只由 <d 的数据选出。"""
```

```
D = arr.shape[0] if D is None else D
Tn = arr.shape[1]
F = np.zeros((D, Tn))
sums = np.zeros((2, Tn)); cnt = np.zeros(2, dtype=int)
E = None
for d in range(D):
    w = wcat(d, cls)
    if E is None:
        F[d] = cold
    else:
        bl = sums[w] / cnt[w] if cnt[w] > 0 else E
        F[d] = (1.0 - float(betas[d])) * E + float(betas[d]) * bl
    sums[w] += arr[d]; cnt[w] += 1
    E = arr[d].copy() if E is None else float(alphas[d]) * arr[d] + (1.0 - float(alphas[d])) * E
return F
```

```
def build_ab_schedule(arr, cold, D, cls='fs', grid_a=ALPHA_GRID, grid_b=BETA_GRID, min_window=31):
    """逐日扩展窗: 第 d 天的 (alpha,beta) 只由 arr[0..d-1] 的一步预测 MAE 最小选出。
```

min_window: 估计窗口短于该天数时不重标定 (窗口太短时星期基线每类还没有样本,
网格搜索只是在噪声上过拟合), 沿用最朴素结构 beta=1 / alpha=grid_a[0]。

默认 31 天 = 基础训练期 (1 月), 与题目"结果从 2 月 1 日起"的信息结构一致。

"""

```
al = np.zeros(D); be = np.zeros(D)
```

```

for d in range(D):
    if d < max(2, min_window):
        al[d], be[d] = grid_a[0], 1.0
        continue
    sub = arr[:d]
    best = None
    for a in grid_a:
        for b in grid_b:
            F = panel_evolve(sub, cold, np.full(d, a), np.full(d, b), cls, d)
            e = float(np.abs(F[1:d] - sub[1:d]).mean())
            if best is None or e < best[0]:
                best = (e, a, b)
        al[d], be[d] = best[1], best[2]
return al, be

def pv_stage(V, SIG, d, dec, scope='all'):
    """附件3 版本融合。可用版本: pub(v) <= dec 且覆盖目标时段。"""
    arr = np.full(T, np.nan)
    lead = np.full(T, np.nan)
    nm = np.zeros(T, dtype=int)
    row = SIG[min(d, SIG.shape[0] - 1)] if SIG is not None else None
    for t in range(T):
        c = candidates(V, d, t, dec, scope)
        nm[t] = len(c)
        if not c:
            continue
        th = t // 6
        ws = np.ones(len(c), float)
        if row is not None:
            for i, (_pub, j, _v) in enumerate(c):
                s = row[th, j]
                if np.isfinite(s) and s > 0:
                    ws[i] = 1.0 / (s * s)
        if not np.isfinite(ws).all() or ws.sum() <= 0:
            ws = np.ones(len(c), float)
        vals = np.asarray([v for _pub, _j, v in c], float)
        arr[t] = float(np.dot(ws, vals) / ws.sum())
        lead[t] = float(min(j for _pub, j, _v in c))
    return arr, lead, nm

def realtime(net_w, Pf_w, S0, dt, eta, Pmax, smin, smax):
    """实时执行口径(2): 储能按实际缺口实时充放, 补不上才紧急购电, 弃电不退款。"""
    n = len(net_w)
    c = np.zeros(n); d2 = np.zeros(n); e = np.zeros(n); cu = np.zeros(n)
    ss = float(S0)
    for t in range(n):
        need = net_w[t] - Pf_w[t]
        if need > 0:

```

```

cap = min(Pmax, max(ss - smin, 0.0) * eta / dt)
d2[t] = min(need, cap); e[t] = need - d2[t]; ss -= d2[t] * dt / eta
else:
cap = min(Pmax, max(smax - ss, 0.0) / (eta * dt))
c[t] = min(-need, cap); cu[t] = (-need) - c[t]; ss += eta * c[t] * dt
return c, d2, e, cu, ss

```

数据

```

def load_all(inp_path=None, vint_path=None, sigma_path=None):
    inp_path = inp_path or os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json')
    inp = json.load(open(inp_path, encoding='utf-8'))
    q1p = os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q1_input.json')
    q1 = json.load(open(q1p, encoding='utf-8'))
    d = {
        'price_a': np.asarray(inp['price_actual'], float),
        'load_a': np.asarray(inp['load_actual'], float),
        'pv_a': np.asarray(inp['pv_actual'], float),
        'cold_p': np.asarray(q1['price'], float),
        'cold_l': np.asarray(q1['load'], float),
        'cold_v': np.asarray(q1['pv'], float),
        'storage': inp['storage'],
        'dt': float(inp['dt']),
        'inp_path': inp_path, 'q1_path': q1p,
    }
    V, vmeta = load_vintages(vint_path or os.path.join(HERE, 'q3_vintages_B.json'))
    d['V'] = V; d['vmeta'] = vmeta
    sp = sigma_path or SIGMA_DEFAULT
    d['SIG'] = np.load(sp) if (sp and os.path.isfile(sp)) else None
    d['sigma_path'] = sp
    return d

```

主模

拟

```

def simulate(D, const, mode='q4-2', score_from=31, dat=None, scope='all',
            cold_l=None, cold_p=None, cold_v=None, replace_from=0, keep_rows=True,
            sched=None, pv0=None):
    """按统一口径跑 D 天。const 至少含 alpha,beta,cls_l,alpha_p,beta_p,cls_p,win,pct。"""
    dat = dat or load_all()
    dt = dat['dt']; st = dat['storage']
    eta = float(st['efficiency']); Pmax = float(st['power_max'])
    smin = float(st['soc_min']); smax = float(st['soc_max'])
    price_a = dat['price_a']; load = dat['load_a']; pv_a = dat['pv_a']
    if replace_from:
        price_a = price_a.copy(); load = load.copy(); pv_a = pv_a.copy()

```

```

    price_a[replace_from:] = 0.0; load[replace_from:] = 0.0; pv_a[replace_from:] = 0.0
    cold_l = dat['cold_l'] if cold_l is None else cold_l
    cold_p = dat['cold_p'] if cold_p is None else cold_p
    cold_v = dat['cold_v'] if cold_v is None else cold_v

```

```

# ---- 预测面板 (全部严格因果) ----

```

```

if sched is None:

```

```

    price_f = panel(price_a, cold_p, const['alpha_p'], const['beta_p'], const['cls_p'], D)

```

```

    load_f = panel(load, cold_l, const['alpha'], const['beta'], const['cls_l'], D)

```

```

    pv_self = panel(pv_a, cold_v, const['alpha'], const['beta'], const['cls_l'], D)

```

```

else:

```

```

    price_f = panel_evolve(price_a, cold_p, sched['ap'], sched['bp'], const['cls_p'], D)

```

```

    load_f = panel_evolve(load, cold_l, sched['al'], sched['bl'], const['cls_l'], D)

```

```

    pv_self = panel_evolve(pv_a, cold_v, sched['al'], sched['bl'], const['cls_l'], D)

```

```

SIG = dat['SIG']      # 4-2 / 4-3 同口径: 逆方差融合 + 扩展窗 sigma_hat

```

```

V = dat['V']

```

```

STAGES = STAGES3 if mode == 'q4-3' else STAGES2

```

```

net = load - pv_a

```

```

# 0:00 阶段的光伏预报 (与参数无关, 可外部缓存)

```

```

if pv0 is None:

```

```

    pv0 = np.full((D, T), np.nan)

```

```

    for d in range(D):

```

```

        a, _l, _n = pv_stage(V, SIG, d, d * 24, scope)

```

```

        pv0[d] = a

```

```

    else:

```

```

        pv0 = pv0[:D]

```

```

netfc0 = load_f - np.where(np.isfinite(pv0), pv0, pv_self[:D])

```

```

def pv_of(d, s):

```

```

    if s == 0:

```

```

        arr = pv0[d].copy()

```

```

    else:

```

```

        off, _slot = STAGES[s]

```

```

        arr, _l, _n = pv_stage(V, SIG, d, d * 24 + off, scope)

```

```

        miss = ~np.isfinite(arr)

```

```

        arr[miss] = pv_self[d][miss]

```

```

    return arr

```

```

W = int(const['win']); pct = float(const['pct'])

```

```

sW = sched['win'] if (sched and 'win' in sched) else None

```

```

sP = sched['pct'] if (sched and 'pct' in sched) else None

```

```

soc_c = float(st['soc_initial'])

```

```

base_c = dev_c = em_c = 0.0

```

```

em_k = cur_k = ch = di = 0.0
n_adj = n_over = n_under = n_same = n_seg = 0
rows = []; win = []; fc_err = []
for d in range(D):
    Wd = int(sW[d]) if sW is not None else W
    pd = float(sP[d]) if sP is not None else pct
    if d < 2:
        margin = np.zeros(T)
    else:
        lo = max(1, d - Wd)
        errs = np.asarray([net[k] - netfc0[k] for k in range(lo, d)])
        margin = np.maximum(np.percentile(errs, pd, axis=0), 0.0)
    pf = price_f[d]
    p_p, _c, _d, soc = solve_window(pf, netfc0[d] + margin, soc_c, dt, eta, Pmax, smin, smax)
    Pf = p_p.copy()
    for s, (h, slot) in enumerate(STAGES[1:], start=1):
        S0 = float(soc[slot])
        nf = (load_f[d] - pv_of(d, s) + margin)[slot:]
        p_, _c_, _d_, s_ = solve_window(pf[slot:], nf, S0, dt, eta, Pmax, smin, smax,
                                         plan_w=p_p[slot:])
        if d >= score_from:
            pw = pf[slot:]; planw = p_p[slot:]; netw = net[d][slot:]
            _a, _b, eA, _c2, _d2 = realtime(netw, p_, S0, dt, eta, Pmax, smin, smax)
            _a, _b, eB, _c3, _d3 = realtime(netw, planw, S0, dt, eta, Pmax, smin, smax)
            devw = float(np.sum(0.5 * pw * np.maximum(0, planw - p_)
                               + 1.5 * pw * np.maximum(0, p_ - planw)) * dt)
            save_base = float(np.sum(pw * np.maximum(0, planw - p_) * dt)
                               + 5.0 * float(np.dot(pw, eB - eA)) * dt)
            win.append({'day': d, 'stage': h, 'slot': slot, 'save_purchase': save_base,
                       'save_emergency': save_em, 'dev_fee': devw,
                       'net_benefit': save_base + save_em - devw,
                       'changed_segments': int(np.sum(np.abs(p_ - planw) > 1e-6))})
            Pf[slot:] = p_
            soc = np.concatenate([soc[:slot], s_])
# ---- 实时执行 (结算用附件4 真实电价) ----
c2, d2, e, cu, Send = realtime(net[d], Pf, soc_c, dt, eta, Pmax, smin, smax)
if d >= score_from:
    pa = price_a[d]
    base_c += float(np.sum(pa * np.minimum(p_p, Pf)) * dt)
    dev_c += float(np.sum(0.5 * pa * np.maximum(0, p_p - Pf)
                          + 1.5 * pa * np.maximum(0, Pf - p_p)) * dt)
    em_c += 5.0 * float(np.dot(pa, e)) * dt
    em_k += float(e.sum() * dt); cur_k += float(cu.sum() * dt)
    ch += float(c2.sum() * dt); di += float(d2.sum() * dt)
    dP = Pf - p_p
    n_adj += int(np.sum(np.abs(dP) > 1e-6))
    n_over += int(np.sum(dP > 1e-6)); n_under += int(np.sum(dP < -1e-6))
    n_same += int(np.sum(np.abs(dP) <= 1e-6)); n_seg += T

```

```

for s, (h, slot) in enumerate(STAGES):
    if s == 0:
        arr = pv0[d]
        larr = np.full(T, np.nan)
    else:
        arr, larr, _n = pv_stage(V, SIG, d, d * 24 + STAGES[s][0], scope)
    for t in range(slot, T):
        if np.isfinite(arr[t]):
            fc_err.append((float(larr[t]) if np.isfinite(larr[t]) else -1.0,
                             float(arr[t]), float(pv_a[d, t])))
    if keep_rows:
        rows.append({'day': d, 'plan0': p_p.copy(), 'plan': Pf.copy(), 'c': c2.copy(),
                     'd': d2.copy(), 'E': e.copy(), 'S0': float(soc_c), 'S1': float(Send)})
    soc_c = float(Send)

pe = price_f[score_from:D] - price_a[score_from:D] if D > score_from else np.zeros((0, T))
ne = np.asarray([net[k] - netfc0[k] for k in range(max(1, score_from), D)]) if D > score_from \
    else np.zeros((0, T))
fe = np.asarray(fc_err) if fc_err else np.zeros((0, 3))
wa = np.asarray([w['net_benefit'] for w in win]) if win else np.zeros(0)
m = {'q4_total_cost': base_c + dev_c + em_c, 'q4_purchase_cost': base_c,
     'q4_adjust_cost': dev_c, 'q4_emergency_cost': em_c,
     'q4_emergency_kwh': em_k, 'q4_curtail_kwh': cur_k,
     'q4_charge_kwh': ch, 'q4_discharge_kwh': di,
     'q4_caliber': 2.0, 'q4_adjust': 1.0 if mode == 'q4-3' else 0.0,
     'score_days': int(max(0, D - score_from)), 'score_from': int(score_from)}
dg = {'mode': mode, 'scope': scope,
      'const': {k: const[k] for k in const},
      'price_forecast': {'mae': float(np.abs(pe).mean()) if pe.size else None,
                          'rmse': float(np.sqrt((pe ** 2).mean())) if pe.size else None,
                          'bias': float(pe.mean()) if pe.size else None},
      'net_forecast': {'n': int(ne.size),
                       'mae': float(np.abs(ne).mean()) if ne.size else None,
                       'rmse': float(np.sqrt((ne ** 2).mean())) if ne.size else None,
                       'bias': float(ne.mean()) if ne.size else None},
      'pv_forecast': ({'n': int(fe.shape[0]),
                       'mae': float(np.abs(fe[:, 1] - fe[:, 2]).mean()),
                       'rmse': float(np.sqrt(((fe[:, 1] - fe[:, 2]) ** 2).mean())),
                       'bias': float((fe[:, 1] - fe[:, 2]).mean())} if fe.size else None),
      'adjusted_segments': n_adj, 'total_segments': n_seg,
      'segments_over_bought': n_over, 'segments_under_bought': n_under,
      'segments_unchanged': n_same, 'n_windows': len(win),
      'window_net_benefit_sum': float(wa.sum()) if wa.size else 0.0,
      'windows_negative_net': int((wa < 0).sum()),
      'sigma_table': os.path.basename(dat['sigma_path']) if mode == 'q4-3' else None,
      'leaky': False}
return m, dg, rows, win

```

```
def calib(D=31, dat=None, score_from=1, verbose=True, grid_from=None):
    """1 月窗口标定: 负荷/电价参数按一步预测 MAE; 冗余 (W,p) 按 1 月总费用。"""
    dat = dat or load_all()
    price_a = dat['price_a'][:D]; load = dat['load_a'][:D]
    cold_p = dat['cold_p']; cold_l = dat['cold_l']

    def mae_panel(arr, cold, cls, alpha, beta):
        F = panel(arr, cold, alpha, beta, cls, D)
        return float(np.abs(F[1:D] - arr[1:D]).mean()), F

    load_grid = []
    for cls in CLS_CHOICES:
        for alpha in ALPHA_GRID:
            for beta in BETA_GRID:
                mae, _ = mae_panel(load, cold_l, cls, alpha, beta)
                load_grid.append({'cls': cls, 'alpha': alpha, 'beta': beta, 'mae': mae})
    load_grid.sort(key=lambda x: x['mae'])
    price_grid = []
    for cls in CLS_CHOICES:
        for alpha in ALPHA_GRID:
            for beta in BETA_GRID:
                mae, _ = mae_panel(price_a, cold_p, cls, alpha, beta)
                price_grid.append({'cls': cls, 'alpha': alpha, 'beta': beta, 'mae': mae})
    price_grid.sort(key=lambda x: x['mae'])
    best_l, best_p = load_grid[0], price_grid[0]
    base = {'alpha': best_l['alpha'], 'beta': best_l['beta'], 'cls_l': best_l['cls'],
           'alpha_p': best_p['alpha'], 'beta_p': best_p['beta'], 'cls_p': best_p['cls']}
    if verbose:
        print('[1 月标定] 负荷 最优 %s MAE=%0.4f kW | 电价 最优 %s MAE=%0.5f 元/kWh'
              % (best_l, best_l['mae'], best_p, best_p['mae']))
    grid = []
    gf = score_from if grid_from is None else grid_from
    for W in W_GRID:
        for pct in P_GRID:
            c = dict(base); c['win'] = W; c['pct'] = pct * 100.0
            m, _dg, _r, _w = simulate(D, c, mode='q4-2', score_from=gf, dat=dat,
                                     keep_rows=False)
            grid.append({'win': W, 'pct': pct * 100.0, 'cost': m['q4_total_cost'],
                       'em_kwh': m['q4_emergency_kwh'], 'cur_kwh': m['q4_curtail_kwh']})
    grid.sort(key=lambda x: x['cost'])
    bestm = grid[0]
    const = dict(base); const['win'] = bestm['win']; const['pct'] = bestm['pct']
    if verbose:
```

```

    print('[1 月标定] 冗余 最优 (W=%d, p=%0.0f%%) 1 月费用 %.2f 元' % (bestm['win'], bestm['pct'], bestm['cost']))
out = {'const': const, 'calibration': {
    'window_days': [gf, D - 1], 'window_dates': '2025-01-%02d..2025-01-31' % (gf + 1),
    'load_grid': load_grid, 'price_grid': price_grid,
    'redundancy_grid': grid,
    'rule': '负荷/光伏参数与电价参数按 1 月窗口一步预测 MAE 最小; 冗余 (W,p) 按 1 月窗口 4-2 总费用最小',
    'leaky': False}}
return out

```

可进

化标定

```

def build_ab_schedule_both(dat, D, verbose=True, min_window=31):
    """逐日扩展窗标定负荷/光伏与电价的 (alpha,beta)。"""
    al, bl = build_ab_schedule(dat['load_a'], dat['cold_l'], D, 'fs', min_window=min_window)
    ap, bp = build_ab_schedule(dat['price_a'], dat['cold_p'], D, 'fs', min_window=min_window)
    if verbose:
        for d in (30, 60, 90, 120, 180, 240, 300, 364):
            if d < D:
                print(' [ab] 第%d 天(窗口%d): 负荷 a=%0.2f b=%0.2f | 电价 a=%0.2f b=%0.2f'
                    % (d, d, al[d], bl[d], ap[d], bp[d]), flush=True)
    return {'al': al, 'bl': bl, 'ap': ap, 'bp': bp}

def build_wp_schedule(dat, D, PV0, sch_ab, const, cadence=30, triggers=None,
    grid_w=(3, 5, 7, 10, 14, 21, 28), grid_p=(0.60, 0.70, 0.80, 0.88, 0.90),
    verbose=True, mode='q4-2'):
    """用截至当日的全部历史重选 (W,p): 使历史总费用最小。

    触发日: triggers 给定则只用这些天 (如四立 33/124/218/310); 否则每 cadence 天一次 (每月 1 日)。
    """
    win = np.zeros(D, int); pct = np.zeros(D)
    cw, cp = 7, 70.0
    log = []
    TRG = set(int(x) for x in triggers) if triggers else None
    for d in range(D):
        hit = (d in TRG) if TRG is not None else (d >= 2 and (d - 1) % cadence == 0)
        if hit and d >= 2:
            sub = {k: v[:d] for k, v in sch_ab.items()}
            best = None
            for W in grid_w:
                for p in grid_p:
                    c = dict(const); c['win'] = W; c['pct'] = p * 100.0
                    m, _d, _r, _w = simulate(d, c, mode, 1, dat=dat, keep_rows=False,
                        sched=sub, pv0=PV0[:d])
                    if best is None or m['q4_total_cost'] < best[0]:
                        best = (m['q4_total_cost'], W, p * 100.0)

```

```

    cw, cp = best[1], best[2]
    log.append({'day': d, 'window': d, 'win': cw, 'pct': cp, 'hist_cost': best[0]})
    if verbose:
        print(' [wp] 第%d天(窗口%3d): W=%2d p=%0f%% (历史费用 %0f)'
              % (d, d, cw, cp, best[0]), flush=True)
    win[d] = cw; pct[d] = cp
    return {'win': win, 'pct': pct}, log

```

泄露

检验

```

def leaktest(const, dat=None):
    dat = dat or load_all()
    res = {}
    ok = True
    # L1 常数标定只用 1 月: 破坏第 31 天起的数据后重标, 结果必须逐位相同
    c1 = calib(dat=dat, verbose=False)
    dat2 = load_all()
    dat2['price_a'] = dat2['price_a'].copy(); dat2['load_a'] = dat2['load_a'].copy()
    dat2['pv_a'] = dat2['pv_a'].copy()
    dat2['price_a'][31:] = 0.0; dat2['load_a'][31:] = 0.0; dat2['pv_a'][31:] = 0.0
    c2 = calib(dat=dat2, verbose=False)
    s1 = json.dumps(c1['const'], sort_keys=True); s2 = json.dumps(c2['const'], sort_keys=True)
    res['L1_calib_invariance'] = {'same': s1 == s2, 'a': c1['const'], 'b': c2['const']}
    ok &= (s1 == s2)

    # L2 破坏第 31 天起实际值 -> 第 0..30 天决策逐位不变
    mA, _gA, rA, _wA = simulate(31, const, 'q4-2', 0, dat=dat)
    mB, _gB, rB, _wB = simulate(31, const, 'q4-2', 0, dat=dat, replace_from=31)
    mx = 0.0
    for k in ('plan0', 'plan', 'c', 'd', 'E', 'S0', 'S1'):
        for i in range(len(rA)):
            mx = max(mx, float(np.max(np.abs(np.asarray(rA[i][k], float) - np.asarray(rB[i][k], float)))))
    res['L2_end_to_end'] = {'max_abs_diff': mx, 'pass': mx == 0.0, 'days': len(rA),
                           'cost_A': mA['q4_total_cost'], 'cost_B': mB['q4_total_cost']}
    ok &= (mx == 0.0)

    # L3 预测面板因果性: F[d] 不含 arr[d] 及以后
    arr = dat['load_a']; cold = dat['cold_l']
    F = panel(arr, cold, const['alpha'], const['beta'], const['cls_l'])
    pert = arr.copy(); pert[5:] = 999.0
    Fp = panel(pert, cold, const['alpha'], const['beta'], const['cls_l'])
    same = bool(np.array_equal(F[5:], Fp[5:]))
    res['L3_panel_causality'] = {'same_prefix_days_0_4': same,
                                  'note': 'F[d] 只依赖 arr[<d], 故改动 arr[5:] 不影响 F[0..4]'}

```

```

ok &= same

# L4 冷启动日之后不再依赖附件 1
cold2 = cold * 2.0
F2 = panel(arr, cold2, const['alpha'], const['beta'], const['cls_l'])
same01 = bool(np.array_equal(F[1:], F2[1:]))
res['L4_no_att1_after_day0'] = {'same_from_day1': same01, 'day0_changed': float(np.abs(F[0] - F2[0]).max()) > 0}
ok &= same01

res['ALL_PASS'] = bool(ok)
return res

def evolve_leaktest(dat, const, D0=120, D1=155, scope='all', mode='q4-2'):
    """可进化模型的防泄露检验。

    T1 截断等价: 用 D=D0 跑出的调度/决策, 与用 D=D1 但把第 D0 天起实际值全部置 0 跑出的
        同一段, 必须逐位相同。
    T2 纯静态检验: D 不变, 把第 D0 天起的 arr 改成垃圾值, 结果必须完全不变。
    T3 PV0 因果: 只改 附件 2 第 D0 天起的光伏实际, PV0[:D0] 与  $\delta$  参与的部分必须不变。
    """
    res = {}; ok = True
    PV0a = np.full((D0, T), np.nan)
    for d in range(D0):
        PV0a[d] = pv_stage(dat['V'], dat['SIG'], d, d * 24, scope)[0]
    schA = build_ab_schedule_both(dat, D0, verbose=False)
    wpA, logA = build_wp_schedule(dat, D0, PV0a, schA, const, verbose=False, mode=mode)
    sA = dict(schA); sA.update(wpA)
    mA, _gA, rA, _wA = simulate(D0, const, mode, 31, dat=dat, scope=scope, sched=sA, pv0=PV0a)

    # T1: D=D1, 第 D0 天起实际值置 0
    P1 = np.full((D1, T), np.nan)
    for d in range(D1):
        P1[d] = pv_stage(dat['V'], dat['SIG'], d, d * 24, scope)[0]
    schB = build_ab_schedule_both(dat, D1, verbose=False)
    wpB, _lB = build_wp_schedule(dat, D1, P1, schB, const, verbose=False, mode=mode)
    sB = dict(schB); sB.update(wpB)
    mB, _gB, rB, _wB = simulate(D1, const, mode, 31, dat=dat, scope=scope, sched=sB, pv0=P1,
                                replace_from=D0)

    dev = 0.0
    for k in ('al', 'bl', 'ap', 'bp'):
        dev = max(dev, float(np.max(np.abs(sA[k] - schB[k][:D0]))))
    for k in ('win', 'pct'):
        dev = max(dev, float(np.max(np.abs(wpA[k] - wpB[k][:D0]))))
    mxd = 0.0
    n = min(len(rA), len(rB))

```

```

for k in ('plan0', 'plan', 'c', 'd', 'E', 'S0', 'S1'):
    for i in range(n):
        mxd = max(mxd, float(np.max(np.abs(np.asarray(rA[i][k], float)
            - np.asarray(rB[i][k], float))))))
res["T1_truncation_equivalence"] = {'D0': D0, 'D1': D1, 'schedule_max_diff': dev,
    'decision_max_diff': mxd, 'days_compared': n,
    'pass': (dev == 0.0 and mxd == 0.0)}
ok &= res["T1_truncation_equivalence"]['pass']

# T2: D 不变, 破坏 arr[D0:]
d2 = load_all(dat['inp_path'], None, dat['sigma_path'])
d2['load_a'] = d2['load_a'].copy(); d2['price_a'] = d2['price_a'].copy()
d2['pv_a'] = d2['pv_a'].copy()
d2['load_a'][D0:] = 999.0; d2['price_a'][D0:] = 9.99; d2['pv_a'][D0:] = 999.0
schC = build_ab_schedule_both(d2, D0, verbose=False)
dev2 = max(float(np.max(np.abs(schA[k] - schC[k]))) for k in ('al', 'bl', 'ap', 'bp'))
res["T2_static_no_lookahead"] = {'schedule_max_diff': dev2, 'pass': dev2 == 0.0}
ok &= (dev2 == 0.0)

# T3: PV0 因果
PV0b = np.full((D0, T), np.nan)
d3 = load_all(dat['inp_path'], None, dat['sigma_path'])
d3['pv_a'] = d3['pv_a'].copy(); d3['pv_a'][D0:] = 0.0
for d in range(D0):
    PV0b[d] = pv_stage(d3['V'], d3['SIG'], d, d * 24, scope)[0]
same = bool(np.array_equal(PV0a, PV0b, equal_nan=True))
res["T3_pv0_causality"] = {'same': same, 'pass': same}
ok &= same

# T4: 调度单调性自检 (参数只会随历史增长而改变, 不会跳到未来值)
res["T4_schedule_summary"] = {
    'ab_days': [0, 30, 60, 90, 120, 180, 240, 300, 364],
    'load_beta': [float(sA['bl'][d]) for d in (0, 30, 60, 90, 119, 180, 240, 300, 364) if d < D0],
    'price_beta': [float(sA['bp'][d]) for d in (0, 30, 60, 90, 119, 180, 240, 300, 364) if d < D0],
    'wp_log': logA}
res['ALL_PASS'] = bool(ok)
return res

def verify(dat, mode, outdir):
    """独立复核: 从 solution.json 重算费用 / 校核约束 / 校核能量守恒。"""
    sol = json.load(open(os.path.join(outdir, 'solution.json'), encoding='utf-8'))
    dt = dat['dt']; st = dat['storage']
    eta = float(st['efficiency']); Pmax = float(st['power_max'])
    smin = float(st['soc_min']); smax = float(st['soc_max'])
    days = sol['days']
    ch = di = em = cur = 0.0

```

```

soc_viol = 0; pw_viol = 0; bal = []
cost = 0.0; cost_em = 0.0; cost_dev = 0.0
pv0 = {}
V = dat['V']; SIG = dat['SIG']
for i, d in enumerate(days):
    P0 = np.asarray(sol['plan0'][i]); P = np.asarray(sol['plan'][i])
    E = np.asarray(sol['E'][i])
    C = np.asarray(sol['c'][i]); Dd = np.asarray(sol['d'][i])
    pa = dat['price_a'][d]
    bal.append(float(np.sum(eta * C - Dd / eta) * dt - (sol['S1'][i] - sol['S0'][i])))
    ch += float(C.sum() * dt); di += float(Dd.sum() * dt); em += float(E.sum() * dt)
    pw_viol += int((C > Pmax + 1e-6).sum() + (Dd > Pmax + 1e-6).sum())
    cost += float(np.sum(pa * np.minimum(P0, P)) * dt)    # 成交部分
    cost_dev += float(np.sum(0.5 * pa * np.maximum(0, P0 - P)
        + 1.5 * pa * np.maximum(0, P - P0)) * dt)    # 偏差部分
    cost_em += 5.0 * float(np.dot(pa, E)) * dt
S0 = np.asarray(sol['S0'], float); S1 = np.asarray(sol['S1'], float)
soc_viol = int(((np.concatenate([S0, S1[-1:]]) < smin - 1e-6).sum()
    + (np.concatenate([S0, S1[-1:]]) > smax + 1e-6).sum()))
return {'days': len(days), 'first_day': days[0], 'last_day': days[-1],
    'recomputed_purchase_cost': cost, 'recomputed_adjust_cost': cost_dev,
    'recomputed_emergency_cost': cost_em,
    'recomputed_total': cost + cost_dev + cost_em,
    'charge_kwh': ch, 'discharge_kwh': di, 'roundtrip_ratio': di / ch if ch else None,
    'emergency_kwh': em,
    'energy_balance_max_abs_err': max(abs(x) for x in bal),
    'soc_min': float(min(S0.min(), S1.min())), 'soc_max': float(max(S0.max(), S1.max())),
    'soc_violations': soc_viol, 'power_violations': pw_viol,
    'soc_start': float(S0[0]), 'soc_end': float(S1[-1])}

```

CLI

```

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--task', required=True,
        choices=['calib', 'solve', 'leaktest', 'panel', 'surface', 'gridwp',
            'calibabwp', 'evolve', 'evolveleak', 'verify'])
    ap.add_argument('--mode', default='q4-2', choices=['q4-2', 'q4-3'])
    ap.add_argument('--const', default="", help='标定结果 json (calib 输出)')
    ap.add_argument('--out', default="")
    ap.add_argument('--input', default="")
    ap.add_argument('--sigma', default=SIGMA_DEFAULT)
    ap.add_argument('--scope', default='all', choices=['all', 'same-day'])
    ap.add_argument('--score-from', type=int, default=31, dest='score_from')
    ap.add_argument('--stop-day', type=int, default=0, dest='stop_day')
    ap.add_argument('--replace-actual-from', type=int, default=0, dest='replace_from')
    ap.add_argument('--calib-from', type=int, default=1, dest='calib_from')
    ap.add_argument('--cadence', type=int, default=30, help='evolve: (W,p) 重标定周期(天)')

```

```

ap.add_argument('--wp-only', action='store_true', dest='wp_only')
ap.add_argument('--min-window', type=int, default=31, dest='min_window',
                help='evolve: (alpha,beta) 扩展窗的最小估计窗口(天)')
ap.add_argument('--wp-grid', default='full', choices=['full', 'coarse'], dest='wp_grid')
ap.add_argument('--triggers', default='', help='evolve: 逗号分隔的 (W,p) 重标定日, 覆盖 --cadence')
a = ap.parse_args()

dat = load_all(a.input or None, None, a.sigma)
if a.task == 'calib':
    out = calib(dat=dat, grid_from=a.calib_from, score_from=a.calib_from)
elif a.task == 'leaktest':
    const = json.load(open(a.const, encoding='utf-8'))['const']
    out = leaktest(const, dat)
elif a.task == 'evolveleak':
    const = json.load(open(a.const, encoding='utf-8'))['const']
    out = evolve_leaktest(dat, const, D0=a.stop_day or 120, D1=(a.stop_day or 120) + 35,
                        scope=a.scope, mode=a.mode)
    print(json.dumps({k: v for k, v in out.items() if k != 'T4_schedule_summary'},
                    ensure_ascii=False, indent=1))
elif a.task == 'verify':
    const = json.load(open(a.const, encoding='utf-8'))['const'] if a.const else {}
    out = verify(dat, a.mode, a.out)
    print(json.dumps(out, ensure_ascii=False, indent=1))
elif a.task == 'evolve':
    const = json.load(open(a.const, encoding='utf-8'))['const']
    D = a.stop_day or 365
    PV0 = np.full((D, T), np.nan)
    for d in range(D):
        PV0[d] = pv_stage(dat['V'], dat['SIG'], d, d * 24, a.scope)[0]
    schp = os.path.join(a.out, 'schedule.json') if a.out else ''
    if a.wp_only and schp and os.path.isfile(schp):
        o = json.load(open(schp, encoding='utf-8'))
        sch_ab = {k: np.asarray(v, float) for k, v in o['ab'].items()}
        sch_wp = {k: np.asarray(v, float) for k, v in o['wp'].items()}
        log = o.get('wp_log', [])
    else:
        sch_ab = build_ab_schedule_both(dat, D, min_window=a.min_window)
        trg = [int(x) for x in a.triggers.split(',') if x.strip()] if a.triggers else None
        sch_wp, log = build_wp_schedule(dat, D, PV0, sch_ab, const, cadence=a.cadence,
                                      triggers=trg, mode=a.mode,
                                      grid_w=(3, 5, 7, 10, 14, 21, 28) if a.wp_grid == 'full' else (3, 5, 7, 14, 21),
                                      grid_p=(0.60, 0.70, 0.80, 0.88, 0.90) if a.wp_grid == 'full' else (0.60, 0.70, 0.80, 0.88))
    sched = dict(sch_ab); sched.update(sch_wp)
    m, dg, rows, win = simulate(D, const, a.mode, a.score_from, dat=dat, scope=a.scope,
                               sched=sched, pv0=PV0)
    dg['evolve'] = {'cadence_days': a.cadence, 'min_window': a.min_window,
                  'triggers': a.triggers or None,

```

```

        'ab_grid': 'alpha 6 x beta 6, 逐日扩展窗 MAE',
        'wp_grid': 'W 7 x p 5, 每 %d 天按历史总费用' % a.cadence,
        'wp_log': log, 'leaky': False}
out = {'metrics': m, 'diagnostics': dg}
if a.out:
    os.makedirs(a.out, exist_ok=True)
    json.dump({'days': [r['day'] for r in rows],
              'plan0': [r['plan0'].tolist() for r in rows],
              'plan': [r['plan'].tolist() for r in rows],
              'c': [r['c'].tolist() for r in rows], 'd': [r['d'].tolist() for r in rows],
              'E': [r['E'].tolist() for r in rows],
              'S0': [r['S0'] for r in rows], 'S1': [r['S1'] for r in rows]},
            open(os.path.join(a.out, 'solution.json'), 'w', encoding='utf-8'))
    json.dump(win, open(os.path.join(a.out, 'windows.json'), 'w', encoding='utf-8'),
              ensure_ascii=False)
    json.dump({'argv': sys.argv, 'mode': a.mode, 'scope': a.scope, 'const': const,
              'created_at_utc': datetime.datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ'),
              'ab': {k: [float(x) for x in v] for k, v in sch_ab.items()},
              'wp': {k: [float(x) for x in v] for k, v in sch_wp.items()},
              'wp_log': log},
            open(schp, 'w', encoding='utf-8'), ensure_ascii=False, indent=1)
    json.dump(out, open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'),
              ensure_ascii=False, indent=1)
print('evolve[%s]: 总 %.2f + 成交 %.2f + 偏差 %.2f + 紧急 %.2f (%.2f kWh) 弃电 %.1f 天 %d'
      % (a.mode, m['q4_total_cost'], m['q4_purchase_cost'], m['q4_adjust_cost'],
         m['q4_emergency_cost'], m['q4_emergency_kwh'], m['q4_curtail_kwh'], m['score_days']))
return
elif a.task == 'calibabwp':
    base = json.load(open(a.const, encoding='utf-8'))['const']
    D = a.stop_day or 31
    tab = []
    for alpha in ALPHA_GRID:
        for beta in BETA_GRID:
            for W in W_GRID:
                for pct in P_GRID:
                    c = dict(base); c['alpha'] = alpha; c['beta'] = beta
                    c['win'] = W; c['pct'] = pct * 100.0
                    m, _d, _r, _w = simulate(D, c, 'q4-2', 1, dat=dat, keep_rows=False)
                    tab.append({'alpha': alpha, 'beta': beta, 'win': W, 'pct': pct * 100.0,
                              'cost': m['q4_total_cost'], 'em_kwh': m['q4_emergency_kwh'],
                              'cur_kwh': m['q4_curtail_kwh']})
    tab.sort(key=lambda x: x['cost'])
    out = {'joint_jan_surface': tab, 'n': len(tab),
          'note': '1 月窗口 (第 1..%d 天) 问题 4-2 总费用对 (alpha_L,beta_L,W,p) 的联合面; cls=fs, 电价参数取 1 月 MAE 最优' % (D - 1)}
    print(json.dumps(out['joint_jan_surface'][:10], ensure_ascii=False, indent=1))
elif a.task == 'gridwp':
    const = json.load(open(a.const, encoding='utf-8'))['const']

```

```

D = a.stop_day or 365
tab = []
for W in W_GRID:
    for pct in (0.60, 0.70, 0.80, 0.88, 0.90):
        c = dict(const); c['win'] = W; c['pct'] = pct * 100.0
        m, _d, _r, _w = simulate(D, c, 'q4-2', a.score_from, dat=dat, keep_rows=False)
        tab.append({'win': W, 'pct': pct * 100.0, 'cost': m['q4_total_cost'],
                    'em_kwh': m['q4_emergency_kwh'], 'purchase': m['q4_purchase_cost'],
                    'cur_kwh': m['q4_curtail_kwh']})
        print(' W=%2d p=%4.1f cost=%2f em=%4.1f cur=%4.1f'
              % (W, pct * 100, m['q4_total_cost'], m['q4_emergency_kwh'],
                 m['q4_curtail_kwh']), flush=True)
    tab.sort(key=lambda x: x['cost'])
    out = {'grid': tab, 'stop_day': D, 'score_from': a.score_from,
          'note': '全期 (第 %d..%d 天) 问题 4-2 总费用对 (W,p) 的敏感面' % (a.score_from, D - 1)}
elif a.task == 'surface':
    const = json.load(open(a.const, encoding='utf-8'))['const']
    D = a.stop_day or 31
    tab = []
    for cls in CLS_CHOICES:
        for alpha in ALPHA_GRID:
            for beta in BETA_GRID:
                c = dict(const); c['alpha'] = alpha; c['beta'] = beta; c['cls_l'] = cls
                m, _d, _r, _w = simulate(D, c, 'q4-2', 1, dat=dat, keep_rows=False)
                tab.append({'cls': cls, 'alpha': alpha, 'beta': beta,
                            'cost': m['q4_total_cost'], 'em_kwh': m['q4_emergency_kwh'],
                            'purchase': m['q4_purchase_cost']})
    tab.sort(key=lambda x: x['cost'])
    out = {'ab_cost_surface': tab, 'stop_day': D,
          'note': '1 月窗口 (第 1..%d 天) 问题 4-2 总费用对 (alpha,beta,cls) 的敏感面' % (D - 1)}
elif a.task == 'panel':
    const = json.load(open(a.const, encoding='utf-8'))['const']
    D = a.stop_day or 365
    pf = panel(dat['price_a'], dat['cold_p'], const['alpha_p'], const['beta_p'], const['cls_p'], D)
    lf = panel(dat['load_a'], dat['cold_l'], const['alpha'], const['beta'], const['cls_l'], D)
    out = {'price_mae_jan': float(np.abs(pf[1:31] - dat['price_a'][1:31]).mean()),
          'load_mae_jan': float(np.abs(lf[1:31] - dat['load_a'][1:31]).mean()),
          'price_mae_after_feb': float(np.abs(pf[31:D] - dat['price_a'][31:D]).mean()),
          'load_mae_after_feb': float(np.abs(lf[31:D] - dat['load_a'][31:D]).mean()),
          'const': const}
else:
    const = json.load(open(a.const, encoding='utf-8'))['const']
    D = a.stop_day or 365
    m, dg, rows, win = simulate(D, const, a.mode, a.score_from, dat=dat, scope=a.scope,
                                replace_from=a.replace_from)
    out = {'metrics': m, 'diagnostics': dg}
    if a.out:
        os.makedirs(a.out, exist_ok=True)

```

```

    json.dump({'days': [r['day'] for r in rows],
              'plan0': [r['plan0'].tolist() for r in rows],
              'plan': [r['plan'].tolist() for r in rows],
              'c': [r['c'].tolist() for r in rows], 'd': [r['d'].tolist() for r in rows],
              'E': [r['E'].tolist() for r in rows],
              'S0': [r['S0'] for r in rows], 'S1': [r['S1'] for r in rows]}),
    open(os.path.join(a.out, 'solution.json'), 'w', encoding='utf-8'))
    json.dump(win, open(os.path.join(a.out, 'windows.json'), 'w', encoding='utf-8'),
              ensure_ascii=False)
    json.dump({'argv': sys.argv, 'cwd': os.getcwd(), 'mode': a.mode, 'scope': a.scope,
              'created_at_utc': datetime.datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%SZ"),
              'const': const, 'score_from': a.score_from, 'stop_day': a.stop_day},
              open(os.path.join(a.out, 'receipt.json'), 'w', encoding='utf-8'),
              ensure_ascii=False, indent=1)
    print('%s[%s]: 总 %2f + 成交 %2f + 偏差 %2f + 紧急 %2f (%2f kWh) 弃电 %1f天 %d'
          % (a.mode, a.scope, m['q4_total_cost'], m['q4_purchase_cost'], m['q4_adjust_cost'],
             m['q4_emergency_cost'], m['q4_emergency_kwh'], m['q4_curtail_kwh'], m['score_days']))

if a.task in ('calib', 'leaktest', 'panel', 'surface', 'gridwp', 'calibabwp', 'evolveleak',
            'verify') and a.out:
    os.makedirs(a.out, exist_ok=True)
if a.out and a.task != 'solve':
    json.dump(out, open(os.path.join(a.out, {'calib': 'q4-calib.json',
                                             'leaktest': 'q4-leaktest.json',
                                             'surface': 'q4-ab-surface.json',
                                             'gridwp': 'q4-wp-grid.json',
                                             'calibabwp': 'q4-joint-jan.json',
                                             'evolveleak': 'q4-evolve-leaktest.json',
                                             'verify': 'q4-verify.json',
                                             'panel': 'q4-panel.json'}[a.task]),
              'w', encoding='utf-8'), ensure_ascii=False, indent=1)
if a.out and a.task == 'solve':
    json.dump(out, open(os.path.join(a.out, 'metrics.json'), 'w', encoding='utf-8'),
              ensure_ascii=False, indent=1)
if a.task == 'calib':
    print('常数:', json.dumps(out['const'], ensure_ascii=False))
elif a.task == 'leaktest':
    print(json.dumps(out, ensure_ascii=False, indent=1)[:2000])
elif a.task == 'surface':
    for g in out['ab_cost_surface'][:12]:
        print(' ', g)
elif a.task == 'gridwp':
    for g in out['grid'][:12]:
        print(' ', g)

if __name__ == '__main__':
    main()

```

D.2.68 支撑材料/代码/04_问题四/q4_validate.py

import os

-*- coding: utf-8 -*-

```
"""问题4 独立检验."""
import json, numpy as np, openpyxl
from pathlib import Path
proj = Path(os.environ.get("CUMCM_ROOT", ".")); root = proj/"paper_output_pro"
dt=1.0/6.0; eta=0.9; Pmax=5000.0; soc_min,soc_max=1200.0,10800.0; soc0=6000.0; T=144; D=365
def load_att2():
    wb=openpyxl.load_workbook(proj/"problem_files/附件/附件 2.xlsx",read_only=True,data_only=True)
    load=np.array([[float(c) for c in r[1:]] for r in list(wb["小区负载"].iter_rows(values_only=True))[1:]],dtype=float)
    pv=np.array([[float(c) for c in r[1:]] for r in list(wb["光伏发电实际功率"].iter_rows(values_only=True))[1:]],dtype=float)
    return load,pv
load_actual,pv_actual=load_att2()
data=json.loads((root/"data_cleaned/q4_result_cache.json").read_text(encoding="utf-8"))
prices4=np.array(data["prices4"]); q42=data["q42"]; q43=data["q43"]
R=[]
def chk(n,name,ok,det=""):
    R.append((n,name,ok,det)); print(f'[{n}] {'PASS' if ok else 'FAIL'} {name} {det}')

def check_common(daily, label):
    v_bal=v_soc=v_pow=v_neg=0; worst=0.0; lo=1e9; hi=-1e9; mc=0; md=0
    soc=soc0
    for day in daily:
        a=np.array(day["a"]); c=np.array(day["c"]); dd=np.array(day["d"]); E=np.array(day["E"])
        L=load_actual[day["day"]]; P=pv_actual[day["day"]]
        gap=((L+c)-(a+dd+E+P)).max(); worst=max(worst,gap)
        if gap>1e-6: v_bal+=1
        if (a<-1e-9).any() or (c<-1e-9).any() or (dd<-1e-9).any() or (E<-1e-9).any(): v_neg+=1
        mc=max(mc,c.max()); md=max(md,dd.max())
        if c.max()>Pmax+1e-6 or dd.max()>Pmax+1e-6: v_pow+=1
        s=np.zeros(T+1); s[0]=soc
        for t in range(T): s[t+1]=s[t]+eta*c[t]*dt-dd[t]*dt/eta
        lo=min(lo,s.min()); hi=max(hi,s.max())
        if s.min()<soc_min-1e-6 or s.max()>soc_max+1e-6: v_soc+=1
        soc=s[-1]
    chk(label+"-负载满足", v_bal==0, f"违反={v_bal}, 最大缺口={worst:.2e}")
    chk(label+"-SOC∈[1200,10800]", v_soc==0, f"违反={v_soc}, 范围=[{lo:.2f},{hi:.2f}]")
    chk(label+"-充放电≤5000kW", v_pow==0, f"max c={mc:.1f}, max d={md:.1f}")
    chk(label+"-非负", v_neg==0, f"违反={v_neg}")
```

Q4-2 用 p 作为 a

```
for x in q42: x["a"]=x["p"]
check_common(q42, "Q4-2")
```

成本独立重算

```
def c42(day):
    pr=prices4[day["day"]]; p=np.array(day["p"]); E=np.array(day["E"])
    return float(np.dot(pr,p)*dt)+5.0*float(np.dot(pr,E)*dt)
t42=sum(c42(x) for x in q42[31:])
chk("Q4-2-成本重算==12841054.36", abs(t42-12841054.36)<1.0, f"重算={t42:.2f}")
```

```

check_common(q43, "Q4-3")
def c43(day):
    pr=prices4[day["day"]]; a=np.array(day["a"]); p=np.array(day["p"]); E=np.array(day["E"])
    return float(np.dot(pr,a)*dt)+0.5*float(np.dot(pr,np.abs(a-p))*dt)+5.0*float(np.dot(pr,E)*dt)
t43=sum(c43(x) for x in q43[31:])
chk("Q4-3-成本重算==14418420.26", abs(t43-14418420.26)<1.0, f"重算={t43:.2f}")

```

结构: 0-6 时段无调整

```

bad=sum(1 for x in q43 if not np.allclose(np.array(x["a"])[0:36], np.array(x["p"])[0:36], atol=1e-6))
chk("Q4-3-结构(0-6时 a=p)", bad==0, f"违反天数={bad}")

```

xlsx 交叉核对

```

wb2=openpyxl.load_workbook(root/"result4-2.xlsx",data_only=True)
wb3=openpyxl.load_workbook(root/"result4-3.xlsx",data_only=True)
m=0
for i,x in enumerate(q42[31:]):
    r=i+2
    if abs(wb2["计划购电量"].cell(row=r,column=146).value - sum(x["p"])*dt)>0.01: m+=1
chk("result4-2-全天计划购电量一致", m==0, f"不一致={m}")
m=0
for i,x in enumerate(q43[31:]):
    r=i+2
    if abs(wb3["计划购电量"].cell(row=r,column=146).value - sum(x["p"])*dt)>0.01: m+=1
    if abs(wb3["调整购电量"].cell(row=r,column=146).value - sum(x["a"])*dt)>0.01: m+=1
chk("result4-3-计划/调整全天购电量一致", m==0, f"不一致={m}")

```

合计

```

tot42=sum(wb2["计划购电量"].cell(row=i+2,column=147).value for i in range(334))
chk("result4-2- 全年计划购电费 = 12841054", abs(tot42-sum(float(np.dot(prices4[x["day"]],x["p"])*dt) for x in q42[31:]))<20,
f"xlsx={tot42:.2f}")

```

```

n=sum(1 for x in R if x[2])
print(f"\n=== 问题4 检验: {n}/{len(R)} 通过 ===")
(root/"analysis/q4_validation_report.txt").write_text("\n".join(f"[{a}] {'PASS' if b else 'FAIL'} {c} {d}" for a,c,b,d in
[(x[0],x[1],x[2],x[3]) for x in R]), encoding="utf-8")
print("saved analysis/q4_validation_report.txt")

```

D.2.69 支撑材料/代码/05_数据分析/jieqi.py

-*- coding: utf-8 -*-

```

"""2025 年二十四节气时刻 (Meeus 低精度太阳视黄经法, 精度 ~0.01 度 = 15 分钟)。"""
import math, datetime as dt

```

```

TERMS = ['小寒','大寒','立春','雨水','惊蛰','春分','清明','谷雨','立夏','小满','芒种','夏至',
'小暑','大暑','立秋','处暑','白露','秋分','寒露','霜降','立冬','小雪','大雪','冬至']

```

```

def jd_from_dt(x):
    y, m = x.year, x.month
    d = x.day + (x.hour + x.minute / 60 + x.second / 3600) / 24
    if m <= 2:
        y -= 1; m += 12
    a = y // 100; b = 2 - a + a // 4
    return int(365.25 * (y + 4716)) + int(30.6001 * (m + 1)) + d + b - 1524.5

```

```

def dt_from_jd(jd):

```

```

jd += 0.5
z = int(jd); f = jd - z
if z < 2299161:
    a = z
else:
    al = int((z - 1867216.25) / 36524.25); a = z + 1 + al - al // 4
b = a + 1524; c = int((b - 122.1) / 365.25); dd = int(365.25 * c); e = int((b - dd) / 30.6001)
day = b - dd - int(30.6001 * e) + f
mon = e - 1 if e < 14 else e - 13
yr = c - 4716 if mon > 2 else c - 4715
di = int(day); frac = (day - di) * 24
hh = int(frac); mm = int((frac - hh) * 60)
return dt.datetime(yr, mon, di, hh, mm)

def sun_lon(jd):
    T = (jd - 2451545.0) / 36525.0
    L0 = 280.46646 + 36000.76983 * T + 0.0003032 * T * T
    M = math.radians(357.52911 + 35999.05029 * T - 0.0001537 * T * T)
    C = ((1.914602 - 0.004817 * T - 0.000014 * T * T) * math.sin(M)
          + (0.019993 - 0.000101 * T) * math.sin(2 * M) + 0.000289 * math.sin(3 * M))
    om = math.radians(125.04 - 1934.136 * T)
    lam = L0 + C - 0.00569 - 0.00478 * math.sin(om)
    return lam % 360.0

def find_term(target_deg, jd0, jd1):
    lo, hi = jd0, jd1
    for _ in range(60):
        mid = (lo + hi) / 2
        d = (sun_lon(mid) - target_deg) % 360.0
        if d > 180:
            lo = mid
        else:
            hi = mid
    return (lo + hi) / 2

def all_terms():
    out = []
    for i, nm in enumerate(TERMS):
        deg = (285 + 15 * i) % 360
        jd0 = jd_from_dt(dt.datetime(2025, 1, 1)) + (i * 15.2)
        jd = find_term(deg, jd0, jd0 + 20)
        out.append((nm, dt_from_jd(jd)))
    return out

def lon_of(date):
    """某日 0:00(UTC) 的太阳视黄经(度)"""
    return sun_lon(jd_from_dt(dt.datetime(date.year, date.month, date.day)))

if __name__ == '__main__':

```

```

        out = all_terms()
        _unused = 0
    if __name__ == '__main__':
        _o = all_terms()
        for _nm, _t in _o:
            print('%-4s %s' % (_nm, _t.strftime('%Y-%m-%d %H:%M')))
        print()
        print('四季之始 (四立):')
        for _nm in ('立春', '立夏', '立秋', '立冬'):
            for _n2, _t in _o:
                if _n2 == _nm:
                    print(' %s %s' % (_nm, _t.strftime('%Y-%m-%d %H:%M')))
        print('二分二至 (题目指定日期的依据):')
        for _nm in ('春分', '夏至', '秋分', '冬至'):
            for _n2, _t in _o:
                if _n2 == _nm:
                    print(' %s %s' % (_nm, _t.strftime('%Y-%m-%d %H:%M')))

```

D.2.70 支撑材料/代码/05_数据分析/q4_diag.py

-*- coding: utf-8 -*-

"""问题四 诊断 (E1-E4): 日型结构 / 中国阴历节假日 / 季节效应 / 误差归因。

全部只用附件 2、附件 4 本身; 节假日表来自国务院办公厅 2024-11-12 通知 (事前已知信息)。
输出 UTF-8 文本报告。

"""

```

import os, json, sys, datetime as dt
import numpy as np

```

```

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
OUT = []

```

```

def P(*a):
    OUT.append(' '.join(str(x) for x in a))

```

```

D0 = dt.date(2025, 1, 1)
DAY = [D0 + dt.timedelta(days=i) for i in range(365)]
WD = np.array([d.weekday() for d in DAY])
MON = np.array([d.month for d in DAY])
NAME7 = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

```

2025 年法定节假日 (国务院办公厅 2024-11-12); 含调休 上班日

```

HOLIDAY = {}
for md, nm in (((('01-01'), '元旦'), (('01-28'), '春节/除夕'), (('01-29'), '春节/初一'),
                (('01-30'), '春节/初二'), (('01-31'), '春节/初三'), (('02-01'), '春节/初四'),
                (('02-02'), '春节/初五'), (('02-03'), '春节/初六'), (('02-04'), '春节/初七'),
                (('04-04'), '清明'), (('04-05'), '清明'), (('04-06'), '清明'),
                (('05-01'), '劳动节'), (('05-02'), '劳动节'), (('05-03'), '劳动节'),
                (('05-04'), '劳动节'), (('05-05'), '劳动节'),
                (('05-31'), '端午'), (('06-01'), '端午'), (('06-02'), '端午'),
                (('10-01'), '国庆中秋'), (('10-02'), '国庆中秋'), (('10-03'), '国庆中秋'),

```

```

        (('10-04'), '国庆中秋'), (('10-05'), '国庆中秋'), (('10-06'), '国庆中秋/八月十五'),
        (('10-07'), '国庆中秋'), (('10-08'), '国庆中秋')):
    y, m, d = 2025, int(md[:2]), int(md[3:])
    HOLIDAY[(m, d)] = nm

WORK_OVERRIDE = {(1, 26): '春节调休上班', (2, 8): '春节调休上班',
                  (4, 27): '劳动节调休上班', (9, 28): '国庆调休上班', (10, 11): '国庆调休上班'}

LUNAR = {(1, 28): '除夕', (1, 29): '正月初一', (2, 12): '正月十五/元宵', (3, 1): '二月初二/龙抬头',
          (4, 4): '清明', (5, 31): '五月初五/端午', (8, 29): '七月初七/七夕', (9, 6): '七月十五/中元',
          (10, 6): '八月十五/中秋', (10, 29): '九月初九/重阳', (12, 21): '冬至', (6, 21): '夏至',
          (3, 20): '春分', (9, 23): '秋分'}

def load():
    p = os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json')
    d = json.load(open(p, encoding='utf-8'))
    L = np.asarray(d['load_actual'], float)
    V = np.asarray(d['pv_actual'], float)
    R = np.asarray(d['price_actual'], float)
    return L.mean(axis=1), V.mean(axis=1), R.mean(axis=1), L, V, R

def e1(lm, pm, rm):
    P('=' * 92)
    P('E1 日型结构辨识: 低负荷日到底是哪几天?')
    P('=' * 92)
    P('%-5s %-5s %10s %10s %10s %10s %' % ('星期', 'n', '日均负荷', '日均光伏', '日均电价', '相对全体'))
    base = lm.mean()
    for k in range(7):
        m = WD == k
        P('%-5s %5d %10.1f %10.1f %10.4f %9.1f%%'
          % (NAME7[k], m.sum(), lm[m].mean(), pm[m].mean(), rm[m].mean(), 100 * (lm[m].mean() / base - 1)))
    P("")
    P('候选日型分组的全年拟合优度 (组内日均负荷的残差标准差, 越小越好):')
    cand = {'{周五,周六}': lambda w: w in (4, 5), '{周六,周日}': lambda w: w in (5, 6),
            '{周五,周六,周日}': lambda w: w in (4, 5, 6), '{周六}': lambda w: w == 5,
            '{周日}': lambda w: w == 6, '不分星期': lambda w: False}
    for nm, f in cand.items():
        g = np.array([f(w) for w in WD])
        res = np.zeros(365)
        for v in (True, False):
            if (g == v).any():
                res[g == v] = lm[g == v] - lm[g == v].mean()
        P(' %-16s 组内残差 std = %7.1f kW (组数 %d, 各组样本 %s)'
          % (nm, res.std(), len(set(g.tolist())), [int((g == v).sum()) for v in (True, False)]))
    P("")
    P('1 月窗口内的同一比较 (只用 1 月, 检验它是否已能辨识):')
    jan = np.arange(365) < 31
    for nm, f in cand.items():
        g = np.array([f(w) for w in WD]) & jan
        res = np.zeros(365)

```

```

    for v in (True, False):
        if (g == v).any():
            res[g == v] = lm[g == v] - lm[g == v].mean()
    P(' %-16s 1 月组内残差 std = %7.1f kW' % (nm, res[jan].std()))
    P("")

def e2(lm, pm, rm):
    P('=' * 92)
    P('E2 中国阴历 / 法定节假日效应实测')
    P('=' * 92)
    P('2025 年放假安排来源: 国务院办公厅 2024-11-12 通知 (事前已知, 非未来信息)')
    P("")
    P('%-12s %-10s %-4s %10s %10s %10s %12s' % ('日期', '名称', '星期', '日均负荷', '日均光伏', '日均电价', '同型基线偏差'))
    for i, d in enumerate(DAY):
        key = (d.month, d.day)
        if key in HOLIDAY or key in WORK_OVERRIDE:
            nm = HOLIDAY[key] if key in HOLIDAY else WORK_OVERRIDE[key]
            # 同星期类基线: 全部非假日、非调休日的该类均值
            mask = np.array([(WD[j] == WD[i]) and ((DAY[j].month, DAY[j].day) not in HOLIDAY)
                             and ((DAY[j].month, DAY[j].day) not in WORK_OVERRIDE) for j in range(365)])
            bl = lm[mask].mean()
            P('%-12s %-10s %-4s %10.1f %10.1f %10.4f %+.1f%%'
              % (d.strftime('%m-%d'), nm, NAME7[WD[i]], lm[i], pm[i], rm[i], 100 * (lm[i] / bl - 1)))
            P("")
            hol = np.array([(d.month, d.day) in HOLIDAY for d in DAY])
            P('全部法定假日 vs 全部非假日: 负荷 %1.1f vs %1.1f kW (%+.1f%%), 电价 %1.4f vs %1.4f (%+.1f%%)'
              % (lm[hol].mean(), lm[~hol].mean(), 100 * (lm[hol].mean() / lm[~hol].mean() - 1),
                rm[hol].mean(), rm[~hol].mean(), 100 * (rm[hol].mean() / rm[~hol].mean() - 1)))
            ovr = np.array([(d.month, d.day) in WORK_OVERRIDE for d in DAY])
            P('调休上班的周末 vs 全部周末: 负荷 %1.1f vs %1.1f kW (%+.1f%%), 电价 %1.4f vs %1.4f (%+.1f%%)'
              % (lm[ovr].mean(), lm[WD >= 5].mean(), 100 * (lm[ovr].mean() / lm[WD >= 5].mean() - 1),
                rm[ovr].mean(), rm[WD >= 5].mean(), 100 * (rm[ovr].mean() / rm[WD >= 5].mean() - 1)))
            P("")
            P('★ 剔除季节漂移后的节假日效应: 同一月份、同一日型、(非假日) 日为基线')
            P('%-14s %-12s %8s %10s %10s %12s' % ('假期', '覆盖日期', '天数', '假期负荷', '同月同型基线', '偏差'))
            for nm in ('春节', '清明', '劳动节', '端午', '国庆中秋'):
                idx = [i for i, dt_ in enumerate(DAY) if HOLIDAY.get((dt_.month, dt_.day), "").startswith(nm)]
                if not idx:
                    continue
                devs = []
                for i in idx:
                    bmask = np.array([(DAY[j].month == DAY[i].month) and (WD[j] == WD[i])
                                       and ((DAY[j].month, DAY[j].day) not in HOLIDAY)
                                       and ((DAY[j].month, DAY[j].day) not in WORK_OVERRIDE)
                                       for j in range(365)])
                    if bmask.sum() > 0:
                        devs.append(100 * (lm[i] / lm[bmask].mean() - 1))

```

```

P('%-10s %-26s %4d 天 逐日同月同星期偏差 均值 %+.6.2f%% 极差 [%+.2f%%, %+.2f%%]'
   % (nm, ', '.join('%d/%d' % (DAY[i].month, DAY[i].day) for i in idx), len(idx),
      np.mean(devs), min(devs), max(devs)))
P("")
P('阴历/节气重点日:')
for i, d in enumerate(DAY):
    key = (d.month, d.day)
    if key in LUNAR:
        mask = np.array([(WD[j] == WD[i]) and ((DAY[j].month, DAY[j].day) not in HOLIDAY) for j in range(365)])
        bl = lm[mask].mean()
        P(' %-12s %-16s %-4s 负荷 %8.1f(%+.6.1f%%) 电价 %4f'
           % (d.strftime('%m-%d'), LUNAR[key], NAME7[WD[i]], lm[i], 100 * (lm[i] / bl - 1), rm[i]))
P("")

def e3(lm, pm, rm):
    P('=' * 92)
    P('E3 季节效应: 各类日型的逐月水平')
    P('=' * 92)
    lo = np.array([w in (4, 5) for w in WD])
    P('%-4s %8s %10s %10s %10s %10s %12s' % ('月', 'n', 'Lo 均值', 'Hi 均值', '整体', '光伏', '电价'))
    for m in range(1, 13):
        k = MON == m
        P('%-4d %8d %10.1f %10.1f %10.1f %10.1f %12.4f'
           % (m, k.sum(), lm[k & lo].mean(), lm[k & ~lo].mean(), lm[k].mean(), pm[k].mean(), rm[k].mean()))
    P("")
    P('季节摆幅: Lo %0.1f ~ %0.1f (极差 %0.1f kW, %0.1f%%); Hi %0.1f ~ %0.1f (极差 %0.1f kW, %0.1f%%)'
       % (min(lm[(MON == m) & lo].mean() for m in range(1, 13)),
          max(lm[(MON == m) & lo].mean() for m in range(1, 13)),
          max(lm[(MON == m) & lo].mean() for m in range(1, 13)) - min(lm[(MON == m) & lo].mean() for m in range(1, 13)),
          100 * (max(lm[(MON == m) & lo].mean() for m in range(1, 13)) / min(lm[(MON == m) & lo].mean() for m in range(1,
13)) - 1),
          min(lm[(MON == m) & ~lo].mean() for m in range(1, 13)),
          max(lm[(MON == m) & ~lo].mean() for m in range(1, 13)),
          max(lm[(MON == m) & ~lo].mean() for m in range(1, 13)) - min(lm[(MON == m) & ~lo].mean() for m in range(1, 13)),
          100 * (max(lm[(MON == m) & ~lo].mean() for m in range(1, 13)) / min(lm[(MON == m) & ~lo].mean() for m in
range(1, 13)) - 1)))
    P("")
    P('1 月自身内部: Lo 均值 %0.1f (极差 %0.1f), Hi 均值 %0.1f (极差 %0.1f) <- 月内水平几乎不动'
       % (lm[(MON == 1) & lo].mean(), lm[(MON == 1) & lo].max() - lm[(MON == 1) & lo].min(),
          lm[(MON == 1) & ~lo].mean(), lm[(MON == 1) & ~lo].max() - lm[(MON == 1) & ~lo].min()))
    P('1 月在全年的位置: Lo %0.1f vs 全年 %0.1f (%+.1f%%); Hi %0.1f vs 全年 %0.1f (%+.1f%%)'
       % (lm[(MON == 1) & lo].mean(), lm[lo].mean(), 100 * (lm[(MON == 1) & lo].mean() / lm[lo].mean() - 1),
          lm[(MON == 1) & ~lo].mean(), lm[~lo].mean(), 100 * (lm[(MON == 1) & ~lo].mean() / lm[~lo].mean() - 1)))
    P("")

def e4(lm, L, V, R):
    P('=' * 92)
    P('E4 误差归因: 为什么 1 月误差特别小?')

```

```

P('=' * 92)
lo = np.array([w in (4, 5) for w in WD])
P('把「全年同类均值基线的逐日水平误差」分解:')
gm = {True: lm[lo].mean(), False: lm[~lo].mean()}
err = np.array([lm[i] - gm[lo[i]] for i in range(365)])
P(' 全年: |误差| 均值 %7.1f kW (占水平 %7.1f%%)' % (np.abs(err).mean(), 100 * np.abs(err).mean() / lm.mean()))
P(' %20s %10s %12s' % ('期间', '|误差|均值', '相对水平'))
for nm, k in (('1 月 (第 1-30 天)', (MON == 1)), ('2-12 月', (MON > 1)),
              ('2-4 月', (MON >= 2) & (MON <= 4)), ('5-7 月', (MON >= 5) & (MON <= 7)),
              ('8-10 月', (MON >= 8) & (MON <= 10)), ('11-12 月', (MON >= 11))):
    e = err[k]
    P(' %20s %10.1f %11.1f%%' % (nm, np.abs(e).mean(), 100 * np.abs(e).mean() / lm[k].mean()))
P("")
P('分解: 误差 = 季节漂移 + 日型内噪声')
drift = np.zeros(365)
for m in range(1, 13):
    k = MON == m
    for v in (True, False):
        kk = k & (lo == v)
        if kk.any():
            drift[kk] = lm[kk].mean() - gm[v]
P(' 季节漂移分量 |·| 均值 = %7.1f kW (占 %7.1f%%)' % (np.abs(drift).mean(), 100 * np.abs(drift).mean() /
np.abs(err).mean()))
P(' 残差(月内)分量 |·| 均值 = %7.1f kW' % np.abs(err - drift).mean())
P(' 1 月季节漂移分量 = %7.1f kW; 2-12 月 = %7.1f kW'
   % (np.abs(drift[MON == 1]).mean(), np.abs(drift[MON > 1]).mean()))
P("")
P('结论: 若预测模型没有「水平跟踪」 ( $\beta=1$ , 全年同类均值), 它把季节漂移全算成误差;')
P(' 1 月恰好只有漂移的 1/12, 所以 1 月 MAE 远小于 2-12 月, 导致 1 月窗口"看不出"冗余窗口长度的作用。')
P("")

def main():
    lm, pm, rm, L, V, R = load()
    e1(lm, pm, rm)
    e2(lm, pm, rm)
    e3(lm, pm, rm)
    e4(lm, L, V, R)
    out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-diag', 'E1-E4_诊断报告.txt')
    os.makedirs(os.path.dirname(out), exist_ok=True)
    open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
    print('written', out, len(OUT), 'lines')

if __name__ == '__main__':
    main()

```

D.2.71 支撑材料/代码/05_数据分析/q4_diag2.py

-*- coding: utf-8 -*-

"""问题四 诊断 (E5-E6): 日内形状的日型检验 + 扩展窗参数可辨识性。"""

```

import os, sys, json, datetime as dt
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from q4_strict import panel, wcat, ALPHA_GRID, BETA_GRID      # noqa: E402

OUT = []
def P(*a): OUT.append(' '.join(str(x) for x in a))

DAY = [dt.date(2025, 1, 1) + dt.timedelta(days=i) for i in range(365)]
WD = np.array([d.weekday() for d in DAY])
MON = np.array([d.month for d in DAY])
NAME7 = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
d = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json'), encoding='utf-8'))
L = np.asarray(d['load_actual'], float)
R = np.asarray(d['price_actual'], float)
cold_l = np.asarray(json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q1_input.json'),
                                encoding='utf-8'))['load'], float)
cold_p = np.asarray(json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q1_input.json'),
                                encoding='utf-8'))['price'], float)

```

----- E5: 日内形状 -----

```

P('=' * 92); P('E5 日内形状 (144 段归一化曲线): 周日到底该归哪一类?'); P('=' * 92)
m = L.mean(axis=1, keepdims=True)
prof = {}
lo = np.array([w in (4, 5) for w in WD])
for k in range(7):
    prof[k] = (L[WD == k] / m[WD == k]).mean(axis=0)
base_lo = (L[lo] / m[lo]).mean(axis=0)
base_hi = (L[~lo] / m[~lo]).mean(axis=0)
P('每类日均曲线与两类基准的最大绝对差 (归一化, 1.0 = 该日均值):')
for k in range(7):
    P(' %4s vs {五,六} %4f vs {其余} %4f -> 归属 %s'
      % (NAME7[k], np.abs(prof[k] - base_lo).max(), np.abs(prof[k] - base_hi).max(),
        'Lo' if np.abs(prof[k] - base_lo).max() < np.abs(prof[k] - base_hi).max() else 'Hi'))
P('')
P('逐时段类均值基线的残差 std (kW):')
res = np.zeros(365)
for v in (True, False):
    res[lo == v] = np.abs(L[lo == v] - L[lo == v].mean(axis=0)).mean(axis=1)
P(' 全部: %.1f kW' % res.mean())
for k in range(7):
    P(' %4s: %7.1f kW' % (NAME7[k], res[WD == k].mean()))
P('')

```

----- E6: 扩展窗参数可辨识性 -----

```

P('=' * 92); P('E6 扩展窗可辨识性: 训练数据变长后, (alpha,beta) 会不会自己选到 beta<1 ?'); P('=' * 92)

```

```

def best_ab(arr, cold, D, cls='fs'):
    """在 [0,D-1] 窗口上用一步预测 MAE 选 (alpha,beta)。"""
    best = None
    for a in ALPHA_GRID:
        for b in BETA_GRID:
            F = panel(arr, cold, a, b, cls, D)
            e = np.abs(F[1:D] - arr[1:D]).mean()
            if best is None or e < best[0]:
                best = (e, a, b)
    return best

```

```

P('负荷 (每 30 天重标定一次, 窗口 = 第 0..d-1 天):')
P('%8s %10s %10s %10s %s' % ('第 d 天', '窗口长', '最优 alpha', '最优 beta', 'MAE kW'))
for dd in (31, 61, 91, 121, 151, 181, 211, 241, 271, 301, 331, 365):
    e, a, b = best_ab(L, cold_l, dd)

    P('%8d %10d %10.2f %10.2f %8.2f % (dd, dd, a, b, e))

P('')
P('电价:')
P('%8s %10s %10s %10s %s' % ('第 d 天', '窗口长', '最优 alpha', '最优 beta', 'MAE'))
for dd in (31, 61, 91, 121, 151, 181, 211, 241, 271, 301, 331, 365):
    e, a, b = best_ab(R, cold_p, dd, 'fs')

    P('%8d %10d %10.2f %10.2f %8.4f % (dd, dd, a, b, e))

P('')
P('固定 beta=1 (不跟踪水平) 与 beta<=0.9 (跟踪水平) 在不同窗口长度下的 MAE 对比:')
P('%8s %14s %14s %10s' % ('第 d 天', 'best beta=1', 'best beta<=0.9', '改善'))
for dd in (31, 61, 91, 121, 181, 241, 301, 365):
    e1 = min(panel(L, cold_l, a, 1.0, 'fs', dd)[1:dd].__sub__(L[1:dd]).__abs__().mean() for a in ALPHA_GRID)
    e2 = min(np.abs(panel(L, cold_l, a, b, 'fs', dd)[1:dd] - L[1:dd]).mean()
              for a in ALPHA_GRID for b in (0.5, 0.6, 0.7, 0.8, 0.9))

    P('%8d %14.2f %14.2f %9.1f%%' % (dd, e1, e2, 100 * (e1 - e2) / e1))

P('')

```

```

out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-diag', 'E5-E6_诊断报告.txt')
open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
print('written', out)

```

D.2.72 支撑材料/代码/05_数据分析/q4_diag3.py

-*- coding: utf-8 -*-

"""问题四 诊断 D: 以二十四节气 (四立) 划分四季, 检验"四季能否决定这个问题"。

节气时刻用 Meeus 低精度太阳视黄经自算 (code/jieqi.py), 与紫金山天文台公布值逐分钟吻合。

"""

```

import os, sys, json, datetime as dt
import numpy as np

```

```

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from jieqi import all_terms, lon_of                                     # noqa: E402

```

```

OUT = []
def P(*a): OUT.append(' '.join(str(x) for x in a))

```

```

DAY = [dt.date(2025, 1, 1) + dt.timedelta(days=i) for i in range(365)]
WD = np.array([d.weekday() for d in DAY])
MON = np.array([d.month for d in DAY])
D = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json'), encoding='utf-8'))
L = np.asarray(D['load_actual'], float); V = np.asarray(D['pv_actual'], float)
R = np.asarray(D['price_actual'], float)
lm = L.mean(axis=1); vm = V.mean(axis=1); rm = R.mean(axis=1)
lo = np.array([w in (4, 5) for w in WD])
THETA = np.array([lon_of(d) for d in DAY])

```

```

TERMS = all_terms()
TIDX = {nm: (t.date() - dt.date(2025, 1, 1)).days for nm, t in TERMS}
SJ = {k: TIDX[k] for k in ('立春', '立夏', '立秋', '立冬')}
ZQ = {k: TIDX[k] for k in ('春分', '夏至', '秋分', '冬至')}

```

```

P('=' * 96)
P('D0 2025 年节气 (本文自算, 与紫金山天文台公布值逐分钟吻合) ')
P('=' * 96)
P('四立 (四季之始): ' + ' '.join('%s 第%d天(%s)' % (k, v, DAY[v].strftime('%m-%d')) for k, v in SJ.items()))
P('二分二至 (四季之中): ' + ' '.join('%s 第%d天(%s)' % (k, v, DAY[v].strftime('%m-%d')) for k, v in ZQ.items()))
P('')
P('★ 题目表 3 指定的 2025.3.20 / 6.21 / 9.23 / 12.21, 正是 2025 年的 春分 / 夏至 / 秋分 / 冬至。')
P(' 即题目选的四个代表日就是四季的"正中", 四季之"始"是四立。1/1-2/2 属冬, 2/3 立春入春。')
P('')

```

```

SEASON = np.empty(365, dtype=object)
for i in range(365):
    if i < SJ['立春'] or i >= SJ['立冬']:

```

```

        SEASON[i] = '冬'
    elif i < SJ['立夏']:
        SEASON[i] = '春'
    elif i < SJ['立秋']:
        SEASON[i] = '夏'
    else:
        SEASON[i] = '秋'

P('=' * 96)
P('D1 以四立划分的四季, 各季水平 (附件 2 / 附件 4 实测)')
P('=' * 96)
P('%-5s %-16s %5s %11s %11s %10s %10s' % ('季节', '区间', '天数', '低日型负荷', '高日型负荷', '光伏', '电价'))
for s in ('春', '夏', '秋', '冬'):
    m = SEASON == s

    idx = np.where(m)[0]

    P('%-5s %-16s %5d %11.1f %11.1f %10.1f %10.4f'
      % (s, DAY[idx[0]].strftime('%m-%d') + '~' + DAY[idx[-1]].strftime('%m-%d'), m.sum(),
        lm[m & lo].mean(), lm[m & ~lo].mean(), vm[m].mean(), rm[m].mean()))

P('')
NATS = {0: '春(3-5月)', 1: '夏(6-8月)', 2: '秋(9-11月)', 3: '冬(12-2月)'}
nat = np.array([0 if 3 <= m <= 5 else 1 if 6 <= m <= 8 else 2 if 9 <= m <= 11 else 3 for m in MON])
P('对照: 按自然月划分的四"季"')
P('%-11s %5s %11s %11s %10s %10s' % ('季节', '天数', '低日型负荷', '高日型负荷', '光伏', '电价'))
for k in range(4):
    m = nat == k

    P('%-11s %5d %11.1f %11.1f %10.1f %10.4f'
      % (NATS[k], m.sum(), lm[m & lo].mean(), lm[m & ~lo].mean(), vm[m].mean(), rm[m].mean()))

P('')

P('=' * 96)
P('D2 季节划分对"水平"的解释力  $R^2 = 1 - \text{组内平方和} / \text{总平方和}$ ')
P('=' * 96)

def r2(groups):
    o = []

    for v in (True, False):
        y = lm[lo == v]; g = np.asarray(groups)[lo == v]
        tot = ((y - y.mean()) ** 2).sum()
        wit = sum(((y[g == k] - y[g == k].mean()) ** 2).sum() for k in np.unique(g))
        o.append(1 - wit / tot)

    return o

def harm_r2(K):
    o = []

    X = [np.ones(365)]

    for k in range(1, K + 1):
        X += [np.cos(np.radians(k * THETA)), np.sin(np.radians(k * THETA))]

    X = np.column_stack(X)

    for v in (True, False):
        y = lm[lo == v]; Xs = X[lo == v]
        b, *_ = np.linalg.lstsq(Xs, y, rcond=None)
        res = y - Xs @ b
        o.append(1 - (res ** 2).sum() / ((y - y.mean()) ** 2).sum())

    return o

```

```

P('%-42s %13s %13s' % ('划分方式', '低日型 R^2', '高日型 R^2'))
for nm, g in (('★ 节气四季 (四立)', SEASON),
              ('自然月四季 3-5/6-8/9-11/12-2', np.array([NATS[k] for k in nat])),
              ('12 个自然月', MON.astype(str)),
              ('不划分 (全局常数)', np.array(['全'] * 365))):
    a, b = r2(g)
    P('%-42s %13.4f %13.4f' % (nm, a, b))
for K in (1, 2, 3):
    a, b = harm_r2(K)
    P('%-42s %13.4f %13.4f' % ('太阳黄经 %d 阶谐波 (天文季节)' % K, a, b))
P('')

P('=' * 96)
P('D3 ★ 关键实验: 只有 1 月数据时, "四季"这个先验能不能被用起来? ')
P('=' * 96)
P('把日型水平写成太阳黄经的 1 阶谐波 y = mu + a*cos(theta) + b*sin(theta) 等价于 mu + A*cos(theta-phi)。')
P('分别用「只有 1 月」和「全年」拟合, 看外推到全年的误差。')
P('')
P('1 月覆盖的太阳黄经: %.1f 度 ~ %.1f 度, 弧长 %.1f 度 = 全年的 %.1f%%'
   % (THETA[0], THETA[30], (THETA[30] - THETA[0]) % 360, 100 * ((THETA[30] - THETA[0]) % 360) / 360))
P('')
P('%-14s %-11s %10s %10s %10s %13s %13s' % ('变量', '拟合窗口', 'a', 'b', '幅度 A', '相位 phi(度)', '外推全年 RMSE'))
Xf = np.column_stack([np.ones(365), np.cos(np.radians(THETA)), np.sin(np.radians(THETA))])
for nm, yy, base_idx in (('低日型负荷', lm, np.where(lo)[0]),
                          ('高日型负荷', lm, np.where(~lo)[0]),
                          ('光伏', vm, np.arange(365)),
                          ('电价', rm, np.arange(365))):
    for tag, jj in (('只用 1 月', base_idx[base_idx < 31]), ('全年(事后)', base_idx)):
        X = np.column_stack([np.ones(len(jj)), np.cos(np.radians(THETA[jj])), np.sin(np.radians(THETA[jj]))])
        b, *_ = np.linalg.lstsq(X, yy[jj], rcond=None)
        pred = Xf @ b
        err = np.sqrt(np.mean((pred[base_idx] - yy[base_idx])**2))
        A = np.hypot(b[1], b[2]); phi = np.degrees(np.arctan2(-b[2], b[1])) % 360
        P('%-14s %-11s %10.1f %10.1f %10.1f %13.1f %13.1f' % (nm, tag, b[1], b[2], A, phi, err))
P('')
P('直接看: 只用 1 月拟合出的季节曲线, 外推到 7 月会离谱到什么程度')
for nm, ii in (('低日型负荷', np.where(lo)[0]), ('高日型负荷', np.where(~lo)[0])):
    jj = ii[ii < 31]
    X = np.column_stack([np.ones(len(jj)), np.cos(np.radians(THETA[jj])), np.sin(np.radians(THETA[jj]))])
    b, *_ = np.linalg.lstsq(X, lm[jj], rcond=None)
    pred = Xf @ b
    july = np.where((MON == 7) & (lo if nm.startswith('低') else ~lo))[0]
    P(' %s: 1 月实测 %.1f kW → 外推 7 月 %.1f kW, 实测 %.1f kW, 偏差 %+.1f kW (%+.0f%%)'
      % (nm, lm[jj].mean(), pred[july].mean(), lm[july].mean(),
         pred[july].mean() - lm[july].mean(), 100 * (pred[july].mean() / lm[july].mean() - 1)))
P('')
P('原因: 1 月的太阳黄经只走了约 30 度, 占全年 8.6%。用 8.6% 的相位弧去定一条 360 度的正弦曲线, ')
P(' 幅度 A 与相位 phi 严重不可辨识; 外推会被放大成完全错误的季节曲线。')
P(' ——“四季”是问题的【根因】, 却不是 1 月窗口能用的【解药】。')
P('')

out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-diag', 'D_节气四季诊断.txt')
open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
print('written', out)

```

D.2.73 支撑材料/代码/05_数据分析/q4_diag4.py

-*- coding: utf-8 -*-

"""诊断 P: 光伏到底跟"季节" (日历) 有关, 还是跟"节气" (天文几何) 有关?

判决思路: 同一年内 节气 与 日历日期 一一对应, 光比拟合优度分不开, 必须比【函数形式】。

M1 天文几何模型: $PV = k * H_0(\phi, \delta)$ —— 只有 2 个自由参数 (k, phi), delta 完全由太阳黄经决定

```

M2 theta 谐波 : PV = a0 + sum_k [a_k cos(k*theta) + b_k sin(k*theta)] — 2K 个自由参数
M3 自然月哑变量 : PV = 12 个月各自的均值 — 11 个自由参数, 无任何几何约束
比较三者的 R^2 与参数个数, 并做日内形状 (昼长) 检验与晴空指数分解。
"""
import os, sys, json, datetime as dt
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from jieqi import lon_of # noqa: E402

OUT = []
def P(*a): OUT.append(' '.join(str(x) for x in a))
DAY = [dt.date(2025, 1, 1) + dt.timedelta(days=i) for i in range(365)]
MON = np.array([d.month for d in DAY]); DOY = np.arange(1, 366)
D = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json'), encoding='utf-8'))
PV = np.asarray(D['pv_actual'], float) # (365,144) kW
E = PV.mean(axis=1) * 24.0 # 日发电量 kWh (以日均功率×24h 表示)
TH = np.array([lon_of(d) for d in DAY])
EPS = 23.4393
DEC = np.degrees(np.arcsin(np.sin(np.radians(EPS)) * np.sin(np.radians(TH)))) # 太阳赤纬(度)

def H0(phi_deg, dec_deg, n):
    """水平面天文日辐射量 (kWh/m2/day), 不含大气"""
    phi = np.radians(phi_deg); dec = np.radians(dec_deg)
    ws = np.arccos(np.clip(-np.tan(phi) * np.tan(dec), -1, 1))
    Gsc, pi = 1.367, np.pi
    return (24 * Gsc / pi) * (1 + 0.033 * np.cos(2 * pi * n / 365.0)) * \
        (np.cos(phi) * np.cos(dec) * np.sin(ws) + ws * np.sin(phi) * np.sin(dec))

P(' ' * 96)
P('P1 日总光伏: 天文几何模型 vs theta 谐波 vs 自然月哑变量')
P(' ' * 96)
best = None
for phi in np.arange(10.0, 50.01, 0.5):
    h = H0(phi, DEC, DOY)
    k = float(np.dot(h, E) / np.dot(h, h))
    r = E - k * h
    r2 = 1 - (r ** 2).sum() / ((E - E.mean()) ** 2).sum()
    if best is None or r2 > best[0]:
        best = (r2, phi, k)
r2_geo, phi_hat, k_hat = best
P('M1 天文几何模型 PV = k * H0(phi, delta)')
P('拟合出的等效纬度 phi = %.1f 度, k = %.4f, R^2 = %.4f (自由参数 2 个)'
% (phi_hat, k_hat, r2_geo))
P(' (delta 完全由太阳黄经决定; H0 只用到赤纬与昼长, 即"节气"信息) ')
P('')
for K in (1, 2, 3, 4):
    X = [np.ones(365)]
    for kk in range(1, K + 1):
        X += [np.cos(np.radians(kk * TH)), np.sin(np.radians(kk * TH))]
    X = np.column_stack(X)
    b, *_ = np.linalg.lstsq(X, E, rcond=None)
    P('M2 太阳黄经 %d 阶谐波 R^2 = %.4f (自由参数 %d 个)'
% (K, 1 - ((E - X @ b) ** 2).sum() / ((E - E.mean()) ** 2).sum(), X.shape[1]))
Xd = np.column_stack([(MON == m).astype(float) for m in range(2, 13)] + [np.ones(365)])
bd, *_ = np.linalg.lstsq(Xd, E, rcond=None)
P('M3 自然月哑变量(12 个月均值) R^2 = %.4f (自由参数 11 个)'
% (1 - ((E - Xd @ bd) ** 2).sum() / ((E - E.mean()) ** 2).sum()))
Xa = np.column_stack([np.cos(np.radians(DOY * 360 / 365.0)), np.sin(np.radians(DOY * 360 / 365.0)), np.ones(365)])
ba, *_ = np.linalg.lstsq(Xa, E, rcond=None)
P('M4 日历日序 1 阶谐波 (= 自然季节) R^2 = %.4f (自由参数 3 个)'
% (1 - ((E - Xa @ ba) ** 2).sum() / ((E - E.mean()) ** 2).sum()))
P('')
P('★ 天文几何模型只用 2 个参数就达到 R^2 = %.4f, 与 11 个参数的自然月哑变量 (%.4f) 几乎持平, '
% (r2_geo, 1 - ((E - Xd @ bd) ** 2).sum() / ((E - E.mean()) ** 2).sum()))
P('说明光伏的"季节形状"本质上由太阳几何决定 — 即由【节气】决定。')
P('')

```

```

P('=' * 96)
P('P2 日内形状 (昼长) 检验: 光伏有效时段跨度 vs 天文昼长')
P('=' * 96)
phi = phi_hat
hd = H0(phi, DEC, DOY)
ws = np.arccos(np.clip(-np.tan(np.radians(phi)) * np.tan(np.radians(DEC)), -1, 1))
daylen_astro = 2 * np.degrees(ws) / 15.0 # 小时
span = np.zeros(365); t0 = np.zeros(365); t1 = np.zeros(365)
for i in range(365):
    y = PV[i]

    idx = np.where(y > 1.0)[0] # 阈值 1 kW

    if idx.size:
        t0[i] = idx[0] * 10 / 60.0; t1[i] = (idx[-1] + 1) * 10 / 60.0

        span[i] = t1[i] - t0[i]

    else:
        span[i] = 0

m = span > 0
P('有光伏的天数 = %d / 365' % m.sum())
P('光伏有效时段跨度 vs 天文昼长 (等效纬度 %.1f 度):' % phi)
for i in (15, 45, 78, 110, 140, 171, 200, 218, 250, 264, 295, 330, 354):
    P(' %s 光伏跨度 %5.2fh 天文昼长 %5.2fh 差 %5.2fh 日出 %5.2f时 日落 %5.2f时'

        % (DAY[i].strftime('%m-%d'), span[i], daylen_astro[i], span[i] - daylen_astro[i], t0[i], t1[i]))

P('')
ok = m & (span > 0)
P('全部有光伏日的相关: corr(光伏跨度, 天文昼长) = %.4f' % np.corrcoef(span[ok], daylen_astro[ok])[0, 1])
P(' 平均偏差 %+.2f h, 标准差 %2f h' % ((span - daylen_astro)[ok].mean(), (span - daylen_astro)
[ok].std()))
P(' 线性回归 光伏跨度 = %.3f * 天文昼长 %+.3f'
% tuple(np.polyfit(daylen_astro[ok], span[ok], 1)))
P('')

P('=' * 96)
P('P3 晴空指数分解: 把光伏拆成"天文几何"x"天气"')
P('=' * 96)
h0 = H0(phi, DEC, DOY)
kt = E / (k_hat * h0)
P('晴空指数 kt = 实际日发电量 / 天文日辐射量 (已标定 k)')
P(' 全年均值 %.4f, 标准差 %.4f, 变异系数 %.4f' % (kt.mean(), kt.std(), kt.std() / kt.mean()))
P('')
P(' 逐月均值:');
for mth in range(1, 13):
    kk = MON == mth

    P(' 月%2d %.4f (n=%d)' % (mth, kt[kk].mean(), kk.sum()))

P('')
Xm = np.column_stack([(MON == mm).astype(float) for mm in range(2, 13)] + [np.ones(365)])
bm, *_ = np.linalg.lstsq(Xm, kt, rcond=None)
r2_kt = 1 - ((kt - Xm @ bm) ** 2).sum() / ((kt - kt.mean()) ** 2).sum()
P(' 晴空指数对"月份"回归的 R^2 = %.4f → %.4f%% 的天气波动是月内随机(云量), 只有 %.1f%% 是季节性的'
% (r2_kt, 100 * (1 - r2_kt), 100 * r2_kt))
P('')
P(' 若把光伏写成 k*H0(phi,delta)*kt, 其中 kt 用月均值近似: ')
P(' 天文部分贡献的相对季节摆幅 = %.1f%%' % (100 * (h0.max() / h0.min() - 1)))
P(' 天气部分 (月均 kt) 的相对季节摆幅 = %.1f%%' % (100 * (max(kt[MON == mm].mean() for mm in range(1, 13))
/ min(kt[MON == mm].mean() for mm in range(1, 13)) - 1)))

P('')

out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-diag', 'P_光伏季节归属诊断.txt')
open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
print('written', out)

```

D.2.74 支撑材料/代码/05_数据分析/q4_diag5.py

-*- coding: utf-8 -*-

```

"""诊断 P' : 用【倾斜面】天文模型重判光伏归属 — 水平面模型可能用错了几何。"""
import os, sys, json, datetime as dt
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from jieqi import lon_of

OUT = []
def P(*a): OUT.append(' '.join(str(x) for x in a))
DAY = [dt.date(2025, 1, 1) + dt.timedelta(days=i) for i in range(365)]

```

```

MON = np.array([d.month for d in DAY]); DOY = np.arange(1, 366)
D = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json'), encoding='utf-8'))
PV = np.asarray(D['pv_actual'], float)
E = PV.mean(axis=1) * 24.0
TH = np.array([lon_of(d) for d in DAY])
EPS = 23.4393
DEC = np.degrees(np.arcsin(np.sin(np.radians(EPS)) * np.sin(np.radians(TH))))

def Hb(phi, dec, n):
    phi = np.radians(phi); dec = np.radians(dec)
    ws = np.arccos(np.clip(-np.tan(phi) * np.tan(dec), -1, 1))
    return (24 * 1.367 / np.pi) * (1 + 0.033 * np.cos(2 * np.pi * n / 365.0)) * \
        (np.cos(phi) * np.cos(dec) * np.sin(ws) + ws * np.sin(phi) * np.sin(dec))

def Htilt(phi, beta, dec, n, df=0.15, rho=0.2):
    """南向倾斜面天文日辐射量 (Klein 1977) """
    phi_r = np.radians(phi); dec_r = np.radians(dec); b = np.radians(beta)
    ws = np.arccos(np.clip(-np.tan(phi_r) * np.tan(dec_r), -1, 1))
    den = np.cos(phi_r) * np.cos(dec_r) * np.sin(ws) + ws * np.sin(phi_r) * np.sin(dec_r)
    wst = np.minimum(ws, np.arccos(np.clip(-np.tan(phi_r - b) * np.tan(dec_r), -1, 1)))
    num = np.cos(phi_r - b) * np.cos(dec_r) * np.sin(wst) + wst * np.sin(phi_r - b) * np.sin(dec_r)
    Rb = np.where(den > 1e-9, np.maximum(num, 0) / np.maximum(den, 1e-9), 0.0)
    H0 = Hb(phi, dec, n)
    return H0 * ((1 - df) * Rb + df * (1 + np.cos(b)) / 2.0 + rho * (1 - np.cos(b)) / 2.0)

P('=' * 96)
P("P1' 倾斜面天文模型网格搜索 (南向, 漫射比 0.15, 反照率 0.2)")
P('=' * 96)
P('%8s %8s %12s %10s' % ('纬度phi', '倾角beta', 'k', 'R^2'))
best = None; rows = []
for phi in np.arange(10.0, 45.01, 1.0):
    for beta in np.arange(0.0, 50.01, 2.0):
        h = Htilt(phi, beta, DEC, DOY)
        if h.min() <= 0:
            continue
        k = float(np.dot(h, E) / np.dot(h, h))
        r2 = 1 - ((E - k * h) ** 2).sum() / ((E - E.mean()) ** 2).sum()
        rows.append((r2, phi, beta, k))
        if best is None or r2 > best[0]:
            best = (r2, phi, beta, k)
rows.sort(reverse=True)
for r in rows[:10]:
    P('%8.1f %8.1f %12.2f %10.4f' % (r[1], r[2], r[3], r[0]))
r2_t, phi_t, beta_t, k_t = best
P('')
P('最优: 纬度 %.1f 度, 倾角 %.1f 度, k=%.2f, R^2 = %.4f (自由参数 3 个)' % (phi_t, beta_t, k_t, r2_t))
P('对照(水平面): 纬度 23.5 度, k=6000.0, R^2 = 0.8190 (自由参数 2 个)')
P('对照(自然月哑变量 11 参数): R^2 = 0.9122')
P('对照(太阳黄经 4 阶谐波 9 参数): R^2 = 0.9376')
P('')

P('=' * 96)
P("P2' 用倾斜面模型重算晴空指数 kt, 看【天气】部分还有没有季节性")
P('=' * 96)
h = Htilt(phi_t, beta_t, DEC, DOY)
kt = E / (k_t * h)
P('kt 全年均值 %.4f, 标准差 %.4f, 变异系数 %.4f' % (kt.mean(), kt.std(), kt.std() / kt.mean()))
mm = [kt[MON == m].mean() for m in range(1, 13)]
P('逐月均值: ' + ' '.join('%d: %.3f' % (m + 1, mm[m]) for m in range(12)))
P('月均极差 %.4f (%.1f%)' % (max(mm) - min(mm), 100 * (max(mm) / min(mm) - 1)))
Xm = np.column_stack([(MON == m).astype(float) for m in range(2, 13)] + [np.ones(365)])
bm, _ = np.linalg.lstsq(Xm, kt, rcond=None)
r2m = 1 - ((kt - Xm @ bm) ** 2).sum() / ((kt - kt.mean()) ** 2).sum()
P('kt 对【月份】回归 R^2 = %.4f (若天气是纯随机云量, 该值应接近 0)' % r2m)
P('')

```

```

P('把 kt 拆成 月均 + 月内残差:')
P(' 月均分量的标准差 = %.4f (占 %.1f%%)' % (np.std([mm[m - 1] for m in MON]), 100 * np.std([mm[m - 1] for m in MON]) / kt.std()))
P(' 月内残差标准差 = %.4f (占 %.1f%%)' % ((kt - np.array([mm[m - 1] for m in MON])).std(), 100 * (kt - np.array([mm[m - 1] for m in MON])).std() / kt.std()))

P('')
P('★ 判读: ')
P(' · 用正确的几何(倾斜面)后, 天文模型只用 3 个参数就把 R^2 从 0.819 提到 %.4f, ' % r2_t)
P(' 且逐月 kt 的极差只剩 %.1f%% (水平面模型下是 30.2%%), ' % (100 * (max(mm) / min(mm) - 1)))
P(' 说明剩下的【季节成分】主要是几何没用对造成的假象, 而不是真实的气象季节性。')
P(' · 结论: 附件 2 的光伏发电量 = 【太阳几何(由黄经/节气决定)】 × 【日尺度随机云量】。')
P(' 它的季节形状服从节气, 不服从日历月份。')
P('')

out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-diag', "P2_光伏倾斜面天文模型诊断.txt")
open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
print('written', out)

```

D.2.75 支撑材料/代码/05_数据分析/q4_diag6.py

-*- coding: utf-8 -*-

```

"""诊断 Q: 光伏年变化由【节气/天文几何】还是【日历季节】决定 — 函数形式 + 外推能力判决。"""
import os, sys, json, datetime as dt
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from jieqi import lon_of

OUT = []
def P(*a): OUT.append(' '.join(str(x) for x in a))
DAY = [dt.date(2025, 1, 1) + dt.timedelta(days=i) for i in range(365)]
MON = np.array([d.month for d in DAY]); DOY = np.arange(1, 366)
D = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json'), encoding='utf-8'))
PV = np.asarray(D['pv_actual'], float); LD = np.asarray(D['load_actual'], float).mean(axis=1)
E = PV.mean(axis=1) * 24.0
TH = np.array([lon_of(d) for d in DAY]); EPS = 23.4393
DEC = np.degrees(np.arcsin(np.sin(np.radians(EPS)) * np.sin(np.radians(TH))))

def Hgeo(phi, dec, n):
    pr = np.radians(phi); dr = np.radians(dec)
    ws = np.arccos(np.clip(-np.tan(pr) * np.tan(dr), -1, 1))
    return (24 * 1.367 / np.pi) * (1 + 0.033 * np.cos(2 * np.pi * n / 365.0)) * \
        (np.cos(pr) * np.cos(dr) * np.sin(ws) + ws * np.sin(pr) * np.sin(dr))

def scalexy(x, y):
    return float(np.dot(x, y) / np.dot(x, x))

def r2(y, p):
    return 1 - ((y - p) ** 2).sum() / ((y - y.mean()) ** 2).sum()

bt = None
for phi in np.arange(5.0, 55.01, 0.5):
    h = Hgeo(phi, DEC, DOY); k = scalexy(h, E); s = r2(E, h * k)
    if bt is None or s > bt[0]:
        bt = (s, phi, k)
r2g, PHI, KG = bt
h = Hgeo(PHI, DEC, DOY)

P('=' * 96)
P('Q1 函数形式对比(全样本拟合): 参数越少、R^2 越高, 形式越"对"')
P('=' * 96)
res = []
res.append(('天文几何 k*H(phi,delta,昼长) 2 参数', 2, r2g))
res.append(('日历日序 1 阶谐波(=自然季节) 3 参数', 3, r2(E, np.column_stack(
    [np.cos(np.radians(DOY * 360 / 365.0)), np.sin(np.radians(DOY * 360 / 365.0)), np.ones(365)])) @

```

拟合等效纬度

```

bt = None
for phi in np.arange(5.0, 55.01, 0.5):
    h = Hgeo(phi, DEC, DOY); k = scalexy(h, E); s = r2(E, h * k)
    if bt is None or s > bt[0]:
        bt = (s, phi, k)
r2g, PHI, KG = bt
h = Hgeo(PHI, DEC, DOY)

P('=' * 96)
P('Q1 函数形式对比(全样本拟合): 参数越少、R^2 越高, 形式越"对"')
P('=' * 96)
res = []
res.append(('天文几何 k*H(phi,delta,昼长) 2 参数', 2, r2g))
res.append(('日历日序 1 阶谐波(=自然季节) 3 参数', 3, r2(E, np.column_stack(
    [np.cos(np.radians(DOY * 360 / 365.0)), np.sin(np.radians(DOY * 360 / 365.0)), np.ones(365)])) @

```

```

np.linalg.lstsq(np.column_stack([np.cos(np.radians(DOY * 360 / 365.0)),
                                np.sin(np.radians(DOY * 360 / 365.0)), np.ones(365)]), E, rcond=None)[0]))
for K in (1, 2, 3, 4):
    X = [np.ones(365)]
    for kk in range(1, K + 1):
        X += [np.cos(np.radians(kk * TH)), np.sin(np.radians(kk * TH))]
    X = np.column_stack(X); b, *_ = np.linalg.lstsq(X, E, rcond=None)
    res.append(('太阳黄经 %d 阶谐波' % K, X.shape[1], X.shape[1], r2(E, X @ b)))

Xm = np.column_stack([(MON == m).astype(float) for m in range(2, 13)] + [np.ones(365)])
res.append(('自然月哑变量' % 11, 11, r2(E, Xm @ np.linalg.lstsq(Xm, E, rcond=None)[0])))
kt = E / (h * KG); ktm = np.array([kt[MON == m].mean() for m in range(1, 13)])[MON - 1]
res.append(('天文几何 x 月均晴空指数' % 12, 12, r2(E, h * KG * ktm)))
P('%-42s %8s %10s' % ('模型', '参数个数', 'R^2'))
for nm, np_, s in sorted(res, key=lambda z: -z[2]):
    P('%-42s %8d %10.4f' % (nm, np_, s))

P('')
P('拟合出的等效纬度 phi = %.1f 度, k = %.1f' % (PHI, KG))
P('')

P('=' * 96)
P('Q2 ★ 外推能力: 留一月交叉验证 (用其余 11 个月拟合, 预测被完全留出的那个月)')
P('=' * 96)
P('这才对应问题四的真实处境: 2 月 1 日要预测 7 月, 而 7 月的数据一个月都还没出现。')
P('')

def cv(mk):
    e = []
    for m in range(1, 13):
        te = MON == m; tr = ~te
        e.append((E[te] - mk(tr, te)) ** 2)
    return float(np.sqrt(np.concatenate(e).mean()))

def mk_geo(tr, te):
    k = scalexy(h[tr], E[tr]); return h[te] * k

def mk_theta(K):
    def f(tr, te):
        X = [np.ones(365)]
        for kk in range(1, K + 1):
            X += [np.cos(np.radians(kk * TH)), np.sin(np.radians(kk * TH))]
        X = np.column_stack(X); b, *_ = np.linalg.lstsq(X[tr], E[tr], rcond=None)
        return X[te] @ b
    return f

def mk_month_nn(tr, te):
    """季节朴素法: 用日历上最近的"有数据月份"的均值"""
    mu = np.array([E[tr & (MON == m)].mean() if (tr & (MON == m)).any() else np.nan for m in range(1, 13)])
    out = np.zeros(te.sum())
    tgt = MON[te]
    for i, m in enumerate(tgt):
        d = np.abs(np.arange(1, 13) - m); d = np.where(np.isnan(mu), 1e9, d)
        # 日历环形距离
        d2 = np.minimum(np.abs(np.arange(1, 13) - m), 12 - np.abs(np.arange(1, 13) - m))
        d2 = np.where(np.isnan(mu), 1e9, d2)
        out[i] = mu[int(np.argmin(d2))]
    return out

```

```

def mk_geo_kt(tr, te):
    k = scalexy(h[tr], E[tr]); ktr = E[tr] / (h[tr] * k)
    mu = np.array([ktr[MON[tr] == m].mean() if (MON[tr] == m).any() else np.nan for m in range(1, 13)])
    tgt = MON[te]; o = np.zeros(te.sum())
    for i, m in enumerate(tgt):
        d2 = np.minimum(np.abs(np.arange(1, 13) - m), 12 - np.abs(np.arange(1, 13) - m))
        d2 = np.where(np.isnan(mu), 1e9, d2)
        o[i] = h[te][i] * k * mu[int(np.argmax(d2))]
    return o

P('%-42s %14s' % ('模型', '留一月 RMSE'))
for nm, f in (('天文几何 (赤纬+昼长, 纯节气量)', mk_geo),
              ('太阳黄经 1 阶谐波', mk_theta(1)),
              ('太阳黄经 2 阶谐波', mk_theta(2)),
              ('太阳黄经 3 阶谐波', mk_theta(3)),
              ('季节朴素法 (用日历上最近月份)', mk_month_nn),
              ('天文几何 x 月均晴空指数', mk_geo_kt)):
    P('%-42s %14.2f' % (nm, cv(f)))
P('')
P('基准: 全年光伏日均发电量的标准差 = %.2f kWh/日' % E.std())
P('')

P('=' * 96)
P('Q3 日内形状: 光伏逐 10 分钟曲线 vs 天文晴空余弦曲线')
P('=' * 96)
pr = np.radians(PHI)
for i, nm in ((14, '01-15'), (78, '03-20'), (171, '06-21'), (264, '09-22'), (354, '12-21')):
    y = PV[i]; mx = y.max()
    yn = y / mx
    tau = (np.arange(144) + 0.5) * 10 / 60.0
    w = np.radians((tau - 12.0) * 15.0)
    cz = np.sin(pr) * np.sin(np.radians(DEC[i])) + np.cos(pr) * np.cos(np.radians(DEC[i])) * np.cos(w)
    cs = np.maximum(cz, 0); cs = cs / cs.max()
    P(' %s 形状 RMSE = %.4f 峰值 %.1f kW @ %.5.2f 时 天文昼长 %.5.2f h'
      % (nm, float(np.sqrt(((yn - cs) ** 2).mean())), mx, tau[int(np.argmax(y))],
        2 * np.degrees(np.arccos(np.clip(-np.tan(pr) * np.tan(np.radians(DEC[i])), -1, 1))) / 15))
P('')

P('=' * 96)
P('Q4 对照: 负荷有没有天文成分? ')
P('=' * 96)
kl = scalexy(h, LD)
XL = np.column_stack([np.cos(np.radians(TH)), np.sin(np.radians(TH)), np.ones(365)])
P(' 负荷用同一几何模型 R^2 = %.4f (光伏 %.4f)' % (r2(LD, h * kl), r2g))
P(' 负荷用太阳黄经 1 阶谐波 R^2 = %.4f (光伏 %.4f)'
  % (r2(LD, XL @ np.linalg.lstsq(XL, LD, rcond=None)[0]), r2(E, XL @ np.linalg.lstsq(XL, E, rcond=None)[0])))
P('')

out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-diag', 'Q_光伏归属判决.txt')
open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
print('written', out)

```

D.2.76 支撑材料/代码/05_数据分析/q4_diag7.py

-*- coding: utf-8 -*-

"""诊断 R: 全知视角 — 电价 / 用户负载 / 光伏 与 月份、季节、节气、星期几 的关系。

★ 本节全部使用全年数据（事后视角），目的只是【认识数据的生成机制】。
建模仍严格因果（见 Q4-2_诊断与模型修正.md 第二、三节）。

```

"""
import os, sys, json, datetime as dt
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from jieqi import all_terms, lon_of

OUT = []
def P(*a): OUT.append(' '.join(str(x) for x in a))
def R(x, n=2): return ('%. ' + str(n) + 'f') % x

DAY = [dt.date(2025, 1, 1) + dt.timedelta(days=i) for i in range(365)]
WD = np.array([d.weekday() for d in DAY]); MON = np.array([d.month for d in DAY])
DOY = np.arange(1, 366); N = 365
NAME7 = ['周一', '周二', '周三', '周四', '周五', '周六', '周日']
D = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json'), encoding='utf-8'))
PR = np.asarray(D['price_actual'], float); LO = np.asarray(D['load_actual'], float)
PV = np.asarray(D['pv_actual'], float)
TH = np.array([lon_of(d) for d in DAY])
EPS = 23.4393
DEC = np.degrees(np.arcsin(np.sin(np.radians(EPS)) * np.sin(np.radians(TH))))
TERMS = all_terms()
TIDX = np.array([(t.date() - dt.date(2025, 1, 1)).days for _, t in TERMS])
TNM = [nm for nm, _ in TERMS]

```

每个日期归入"最近一个已过的节气"

```

TERMID = np.searchsorted(TIDX, np.arange(365), side='right') - 1
TERMID = np.clip(TERMID, 0, 23)
SJ = {k: int(TIDX[np.where(np.array(TNM) == k)[0][0])] for k in ('立春', '立夏', '立秋', '立冬')}
S4IDX = [SJ['立春'], SJ['立夏'], SJ['立秋'], SJ['立冬']]
S4 = np.array(['冬' if (i < S4IDX[0] or i >= S4IDX[3]) else
               '春' if i < S4IDX[1] else '夏' if i < S4IDX[2] else '秋' for i in range(365)])

VARS = [('电价', PR.mean(axis=1), '元/kWh'), ('小区负载', LO.mean(axis=1), 'kW'),
        ('光伏', PV.mean(axis=1), 'kW')]

P(' ' * 100)
P('R0 三个变量的基本统计 (全年 365 天, 日均值) ')
P(' ' * 100)
P('%-10s %-12s %-12s %-10s %-12s %-12s %-12s %-12s' % ('变量', '年均值', '标准差', '变异系数', '最小', '最大', '峰谷比'))
for nm, y, u in VARS:
    P('%-10s %-12s %-12s %-12s %-12s %-12s %-12s %-12s' %
      '% (nm, R(y.mean(), 4), R(y.std(), 4), 100 * y.std() / y.mean(), R(y.min(), 4), R(y.max(), 4),
        y.max() / max(y.min(), 1e-9)))

P(' ')

P(' ' * 100)
P('R1 星期几效应 (最干净的结构) ')
P(' ' * 100)
P('%-10s %-12s %-12s %-10s %-12s %-12s %-12s %-12s' % ('变量', ' '.join('%10s' % NAME7[k] for k in range(7)) + '%10s' % '五/六 vs 其余'))
for nm, y, u in VARS:
    g = np.array([y[WD == k].mean() for k in range(7)])
    lo56 = np.array([w in (4, 5) for w in WD])
    P('%-10s' % nm + ".join('%10s' % R(v, 4 if y.mean() < 5 else 1) for v in g)
      + '%10s' % ('%.1f%%' % (100 * (y[lo56].mean() / y[~lo56].mean() - 1))))

P(' ')
P('%-10s %-14s %-14s %-14s' % ('变量', '星期几哑变量 R^2', '仅{五,六}R^2', '组内残差 std'))
for nm, y, u in VARS:
    Xw = np.column_stack([(WD == k).astype(float) for k in range(1, 7)] + [np.ones(N)])
    r2w = 1 - ((y - Xw @ np.linalg.lstsq(Xw, y, rcond=None)[0]) ** 2).sum() / ((y - y.mean()) ** 2).sum()
    lo56 = np.array([w in (4, 5) for w in WD])
    res = np.zeros(N)
    for v in (True, False):
        res[lo56 == v] = y[lo56 == v] - y[lo56 == v].mean()
    r2b = 1 - (res ** 2).sum() / ((y - y.mean()) ** 2).sum()
    P('%-10s %-14.4f %-14.4f %-14.1f' % (nm, r2w, r2b, res.std()))

```

```

P('')

P('' * 100)
P('R2 各因子的单独解释力 R^2 (全知视角) ')
P('' * 100)
def dum(v):
    u = np.unique(v)
    return np.column_stack([(v == x).astype(float) for x in u[1:]] + [np.ones(N)])
def harm(K):
    X = [np.ones(N)]
    for k in range(1, K + 1):
        X += [np.cos(np.radians(k * TH)), np.sin(np.radians(k * TH))]
    return np.column_stack(X)
def r2of(X, y):
    return 1 - ((y - X @ np.linalg.lstsq(X, y, rcond=None)[0]) ** 2).sum() / ((y - y.mean()) ** 2).sum()

FAC = [('星期几 (6 df)', dum(WD)), ('{周五,周六} (1 df)', dum(np.array([w in (4, 5) for w in WD]))),
        ('月份 (11 df)', dum(MON)), ('节气四季 四立 (3 df)', dum(S4)),
        ('节气 24 个 (23 df)', dum(TERMID)), ('太阳黄经 1 阶 (2 df)', harm(1)),
        ('太阳黄经 2 阶 (4 df)', harm(2)), ('太阳黄经 4 阶 (8 df)', harm(4))]

P('%-24s' % '因子' + ''.join('%16s' % nm for nm, _, _ in VARS))
for fn, X in FAC:
    P('%-24s' % fn + ''.join('%16.4f' % r2of(X, y) for _, y, _ in VARS))

P('')

P('' * 100)
P('R3 唯一贡献 (控制其他因子之后, 该因子还能多解释多少) — 这才是"谁在起作用"')
P('' * 100)
Xwd = dum(np.array([w in (4, 5) for w in WD])); Xmo = dum(MON)
Xth = harm(4); Xs4 = dum(S4)
combos = [('只有 星期几{五,六}', Xwd), ('只有 月份', Xmo), ('只有 太阳黄经 4 阶', Xth),
           ('星期几 + 月份', np.column_stack([Xwd, Xmo])),
           ('星期几 + 月份 + 太阳黄经 4 阶', np.column_stack([Xwd, Xmo, Xth]))]

tab = {}
for cn, X in combos:
    for nm, y, _ in VARS:
        tab[(cn, nm)] = r2of(X, y)

P('%-32s' % '模型' + ''.join('%16s' % nm for nm, _, _ in VARS))
for cn, _ in combos:
    P('%-32s' % cn + ''.join('%16.4f' % tab[(cn, nm)] for nm, _, _ in VARS))

P('')
P('%-32s' % '唯一贡献 (增量)' + ''.join('%16s' % nm for nm, _, _ in VARS))
u_wd = {nm: tab[('星期几 + 月份', nm)] - tab[('只有 月份', nm)] for nm, _, _ in VARS}
u_mo = {nm: tab[('星期几 + 月份', nm)] - tab[('只有 星期几{五,六}', nm)] for nm, _, _ in VARS}
u_th = {nm: tab[('星期几 + 月份 + 太阳黄经 4 阶', nm)] - tab[('星期几 + 月份', nm)] for nm, _, _ in VARS}
P('%-32s' % ' 星期几 的增量' + ''.join('%16.4f' % u_wd[nm] for nm, _, _ in VARS))
P('%-32s' % ' 月份 的增量' + ''.join('%16.4f' % u_mo[nm] for nm, _, _ in VARS))
P('%-32s' % ' 太阳黄经 的增量' + ''.join('%16.4f' % u_th[nm] for nm, _, _ in VARS))
P('')

P('' * 100)
P('R4 光伏的天文几何 (对照: 负荷、电价) ')
P('' * 100)
def Hgeo(phi, dec, n):
    pr = np.radians(phi); dr = np.radians(dec)
    ws = np.arccos(np.clip(-np.tan(pr) * np.tan(dr), -1, 1))
    return (24 * 1.367 / np.pi) * (1 + 0.033 * np.cos(2 * np.pi * n / 365.0)) * \
        (np.cos(pr) * np.cos(dr) * np.sin(ws) + ws * np.sin(pr) * np.sin(dr))
P('%-10s %14s %14s %14s' % ('变量', '等效纬度', '几何模型 R^2', '黄经 1 阶 R^2'))
for nm, y, _ in VARS:
    best = None
    for phi in np.arange(5, 55.01, 0.5):
        h = Hgeo(phi, DEC, DOY); k = float(np.dot(h, y) / np.dot(h, h)); s = r2of((h * k).reshape(-1, 1), y)
        if best is None or s > best[0]:

```

```

        best = (s, phi)

tag = " if best[0] > 0.3 else ' <- 无天文成分(R^2<0.3, 等效纬度无意义)'

P('%-10s %14.1f %14.4f %14.4f %s' % (nm, best[1], best[0], r2of(harm(1), y), tag))

P('')
P('%-10s %14s %14s' % ('变量', '按 4 阶谐波 R^2', '按"几何模型 x 月均晴空"R^2'))
for nm, y, _ in VARS:
    best = None

    for phi in np.arange(5, 55.01, 0.5):

        h = Hgeo(phi, DEC, DOY); k = float(np.dot(h, y) / np.dot(h, h)); s = r2of((h * k).reshape(-1, 1), y)

        if best is None or s > best[0]:

            best = (s, phi, k)

        hh = Hgeo(best[1], DEC, DOY) * best[2]

        kt = y / hh; ktm = np.array([kt[MON == m].mean() for m in range(1, 13)])[MON - 1]

        P('%-10s %14.4f %14.4f % (nm, r2of(harm(4), y), r2of((hh * ktm).reshape(-1, 1), y)))

P('')

P('=' * 100)
P('R5 逐月水平 (归一化到各自年均) ')
P('=' * 100)
P('%-10s ' % '变量' + ''.join('%7s' % ('%d月' % m) for m in range(1, 13)))
for nm, y, _ in VARS:
    P('%-10s ' % nm + ".join('%7.3f' % (y[MON == m].mean() / y.mean()) for m in range(1, 13)))

P('')
P('%-10s ' % '变量' + ''.join('%7s' % s for s in ('春', '夏', '秋', '冬')))
for nm, y, _ in VARS:
    P('%-10s ' % nm + ".join('%7.3f' % (y[S4 == s].mean() / y.mean()) for s in ('春', '夏', '秋', '冬'))

P('')

P('=' * 100)
P('R6 日内形状 (144 段平均曲线, 按 {工作日} / {周五六} 分) ')
P('=' * 100)
lo56 = np.array([w in (4, 5) for w in WD])
for nm, A, u in (('小区负载', L0, 'kW'), ('光伏', PV, 'kW'), ('电价', PR, '元/kWh')):
    for tag, m in (('工作日(日-四)', ~lo56), ('周五/周六', lo56)):

        prof = A[m].mean(axis=0)

        pk = int(np.argmax(prof)); tr = int(np.argmin(prof))

        h = lambda t: '%d:%02d' % (t * 10 // 60, t * 10 % 60)

        pos = np.where(prof > 1e-6)[0]

        if nm == '光伏':

            P(' %-6s %-12s 峰 %8.1f @ %-5s 白天起止 %s~%s (%.2f h) 日内变异系数 %.3f'
              % (nm, tag, prof[pk], h(pk), h(pos[0]), h((pos[-1] + 1) % 144),
                 (pos[-1] - pos[0] + 1) * 10 / 60.0,
                 prof[pos].std() / prof[pos].mean()))

        else:

            P(' %-6s %-12s 峰 %8.1f @ %-5s 谷 %8.1f @ %-5s 峰谷比 %.2f'
              % (nm, tag, prof[pk], h(pk), prof[tr], h(tr), prof[pk] / max(prof[tr], 1e-9)))

P('')

P('=' * 100)
P('R6b 交互: 星期几效应是否随季节变化')
P('=' * 100)
lo56 = np.array([w in (4, 5) for w in WD])
P('%-10s ' % '变量' + ''.join('%12s' % s for s in ('春', '夏', '秋', '冬')) + '%12s' % '全年')
for nm, y, _ in VARS:
    row = []

    for s in ('春', '夏', '秋', '冬'):

        m = S4 == s

        row.append(100 * (y[m & lo56].mean() / y[m & ~lo56].mean() - 1))

    P('%-10s' % nm + ".join('%11.1f%%' % v for v in row))

```

```

        + '%11.1f%%' % (100 * (y[lo56].mean() / y[~lo56].mean() - 1)))
P('')
Xwm = np.column_stack([dum(np.array([w in (4, 5) for w in WD])), dum(MON),
                        ((np.array([w in (4, 5) for w in WD])).astype(float) *
                         (MON - 6.5))[ :, None], np.ones(N))])
P('%-10s %-16s %-16s %-16s' % ('变量', '星期几+月份', '再加 星期x月份线性交互', '交互增量'))
for nm, y, _ in VARS:
    a = r2of(np.column_stack([dum(np.array([w in (4, 5) for w in WD])), dum(MON)]), y)
    b = r2of(Xwm, y)
    P('%-10s %-16.4f %-16.4f %-16.4f' % (nm, a, b, b - a))
P('')

P('=' * 100)
P('R7 三者的相关 (日均值, 以及去掉星期几+季节后)')
P('=' * 100)
def resid(y):
    X = np.column_stack([dum(np.array([w in (4, 5) for w in WD])), dum(MON)])
    return y - X @ np.linalg.lstsq(X, y, rcond=None)[0]
ys = {nm: y for nm, y, _ in VARS}
P('%-14s %-14s %-14s' % ('', '原始相关', '去星期+季节后'))
for a, b in (('电价', '小区负载'), ('电价', '光伏'), ('小区负载', '光伏')):
    P('%-14s %-14.4f %-14.4f' % ('%s~%s' % (a, b), np.corrcoef(ys[a], ys[b])[0, 1],
                                np.corrcoef(resid(ys[a]), resid(ys[b]))[0, 1]))
P('')

P('=' * 100)
P('R8 光伏预报误差 (附件 3 - 附件 2 实际) 随季节/节气的变化')
P('=' * 100)
try:
    Q3 = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
    pvf = Q3['pv_forecast']
    err = []
    for d in range(365):
        a = np.asarray(pvf['%d_0' % d], float)
        err.append(np.nanmean(a - PV[d]))
    err = np.array(err)
    P('附件 3 的 0:00 版对当天全天的平均误差 (kW):')
    P('%-10s' % '月份' + ".join('%7d' % m for m in range(1, 13)))
    P('%-10s' % 'MAE' + ".join('%7.1f' % np.nanmean(np.abs(err[MON == m])) for m in range(1, 13)))
    P('%-10s' % 'bias' + ".join('%7.1f' % np.nanmean(err[MON == m]) for m in range(1, 13)))
    P('%-10s' % '季节' + ".join('%7s' % s for s in ('春', '夏', '秋', '冬'))))
    P('%-10s' % 'MAE' + ".join('%7.1f' % np.nanmean(np.abs(err[S4 == s])) for s in ('春', '夏', '秋', '冬'))))
except Exception as e:
    P(' (跳过: %s)' % e)
P('')

out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-diag', 'R_全知视角因子分解.txt')
open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
print('written', out)

```

D.2.77 支撑材料/代码/05_数据分析/q4_diag8.py

-*- coding: utf-8 -*-

```

"""诊断 S: 前面那些结论, 哪些只用 1 月 (第 1-31 天) 就能拿到?"""
import os, sys, json, datetime as dt
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
from jieqi import lon_of

```

```
OUT = []
```

```

def P(*a): OUT.append(' '.join(str(x) for x in a))
DAY = [dt.date(2025, 1, 1) + dt.timedelta(days=i) for i in range(365)]
WD = np.array([d.weekday() for d in DAY]); MON = np.array([d.month for d in DAY])
DOY = np.arange(1, 366); N = 365
NAME7 = ['周一', '周二', '周三', '周四', '周五', '周六', '周日']
D = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json'), encoding='utf-8'))
PR = np.asarray(D['price_actual'], float); LO = np.asarray(D['load_actual'], float)
PV = np.asarray(D['pv_actual'], float)
TH = np.array([lon_of(d) for d in DAY]); EPS = 23.4393
DEC = np.degrees(np.arcsin(np.sin(np.radians(EPS)) * np.sin(np.radians(TH))))
J = np.arange(0, 31) # 1 月
A = np.arange(0, 365) # 全年
def dum(v, idx):
    u = np.unique(v[idx])
    return np.column_stack([(v[idx] == x).astype(float) for x in u[1:]] + [np.ones(len(idx))])

def r2of(X, y):
    if X.shape[1] >= len(y):
        return float('nan')
    return 1 - ((y - X @ np.linalg.lstsq(X, y, rcond=None)[0]) ** 2).sum() / ((y - y.mean()) ** 2).sum()

def r2w(y, idx):
    return r2of(dum(WD, idx), y[idx])

ys = {'电价': PR.mean(axis=1), '小区负载': LO.mean(axis=1), '光伏': PV.mean(axis=1)}
lo56 = np.array([w in (4, 5) for w in WD])
P('=' * 104)
P('S1 日型分组与星期几效应: 1 月 vs 全年')
P('=' * 104)
P('%-10s %-26s %-26s %-12s' % ('变量', '只用 1 月', '全年真值', '差'))
import itertools
groups = {'{周五,周六}': lambda w: w in (4, 5), '{周五,周六,周日}': lambda w: w in (4, 5, 6),
          '{周六,周日}': lambda w: w in (5, 6), '{周六}': lambda w: w == 5, '不分星期': lambda w: False}

for nm, y in ys.items():
    P(' %s' % nm)
    for gname, f in groups.items():
        g = np.array([f(w) for w in WD])
        row = []
        for idx in (J, A):
            res = np.zeros(len(idx))
            for v in (True, False):
                m = (g[idx] == v)
                if m.any():
                    res[m] = y[idx][m] - y[idx][m].mean()
            row.append(res.std())
        P(' %-18s 组内残差 std 1 月 %9.1f 全年 %9.1f' % (gname, row[0], row[1]))
        a = 100 * (y[J][lo56[J]].mean() / y[J][~lo56[J]].mean() - 1)
        b = 100 * (y[A][lo56[A]].mean() / y[A][~lo56[A]].mean() - 1)
        P(' {周五,周六} 相对降幅 1 月 %8.1f%% 全年 %8.1f%% 差 %5.1fpp' % (a, b, a - b))
        P(' 星期几哑变量 R^2 1 月 %9.4f 全年 %9.4f' % (r2w(y, J), r2w(y, A)))
    P('')

P('=' * 104)
P('S2 季节性 / 天文几何 / 交互 — 1 月能不能拿到? ')
P('=' * 104)
P('%-30s %-16s %-16s %-12s' % ('量', '只用 1 月', '全年真值', '可得?'))
for nm, y in ys.items():
    P(' %s' % nm)
    P(' %-27s %-16s %-16.4f %-12s' % ('月份哑变量 R^2', '无法估计(只 1 个月)', r2of(dum(MON, A), y[A]), '否'))
    # 天文几何
    def H(phi, dec, n):
        pr = np.radians(phi); dr = np.radians(dec)
        ws = np.arccos(np.clip(-np.tan(pr) * np.tan(dr), -1, 1))
        return (24 * 1.367 / np.pi) * (1 + 0.033 * np.cos(2 * np.pi * n / 365.0)) * \

```

```

        (np.cos(pr) * np.cos(dr) * np.sin(ws) + ws * np.sin(pr) * np.sin(dr))
best = None
for phi in np.arange(5, 55.01, 0.5):
    h = H(phi, DEC, DOY)
    k = float(np.dot(h[A], y[A]) / np.dot(h[A], h[A]))
    s = 1 - ((y[A] - k * h[A]) ** 2).sum() / ((y[A] - y[A].mean()) ** 2).sum()
    if best is None or s > best[0]:
        best = (s, phi)
Xj = np.column_stack([np.ones(31), np.cos(np.radians(TH[J])), np.sin(np.radians(TH[J]))])
bj, *_ = np.linalg.lstsq(Xj, y[J], rcond=None)
Xa = np.column_stack([np.ones(365), np.cos(np.radians(TH)), np.sin(np.radians(TH))])
ba, *_ = np.linalg.lstsq(Xa, y[A], rcond=None)
P('  %-27s %16s %16.4f %12s' % ('天文几何模型 R^2 (全年拟合)', '-', best[0], ''))
P('  %-27s %16.1f %16.1f %12s' % ('1 阶谐波幅度 A (1 月拟合/全年)',
    np.hypot(bj[1], bj[2]), np.hypot(ba[1], ba[2]),
    '幅度尚可' if nm == '光伏' else '不可辨'))
P('  %-27s %16.1f %16.1f %12s' % ('1 阶谐波相位 phi 度',
    np.degrees(np.arctan2(-bj[2], bj[1])) % 360,
    np.degrees(np.arctan2(-ba[2], ba[1])) % 360, '★错'))
Xw = dum(lo56.astype(int)[: , None].ravel(), A) if False else None
P('')
P(' 说明: 1 月 1 日-1 月 31 日全部落在"冬"里 (立春在 2 月 3 日), ')
P(' 即 1 月内部 【完全没有季节切换】 → 任何随季节变化的量在 1 月内都是常数, 无法辨识。')
P('')

P('=' * 104)
P('S3 日内形状与三变量相关: 1 月 vs 全年')
P('=' * 104)
lo56A = lo56; lo56J = lo56[J]
for nm, X, u in (('小区负载', L0, 'kW'), ('光伏', PV, 'kW'), ('电价', PR, '元/kWh')):
    row = []
    for idx in (J, A):
        prof = X[idx].mean(axis=0)
        pk = int(np.argmax(prof)); tr = int(np.argmin(prof))
        row.append((pk * 10 // 60, pk * 10 % 60, tr * 10 // 60, tr * 10 % 60,
            prof[pk], prof[tr], prof[pk] / max(prof[tr], 1e-9)))
P(' %-8s 1 月: 峰 %d:%02d (%8.1f) 谷 %d:%02d (%8.1f) 峰谷比 %2f'
    % (nm, row[0][0], row[0][1], row[0][4], row[0][2], row[0][3], row[0][5], row[0][6]))
P(' %-8s 全年: 峰 %d:%02d (%8.1f) 谷 %d:%02d (%8.1f) 峰谷比 %2f'
    % ('', row[1][0], row[1][1], row[1][4], row[1][2], row[1][3], row[1][5], row[1][6]))
P('')
def resid(y, idx, drop_month=True):
    cols = [dum(lo56.astype(int)[: , None].ravel(), idx)] if False else []
    Xs = [(lo56[idx]).astype(float)[: , None], np.ones((len(idx), 1))]
    if drop_month and len(idx) > 40:
        Xs.append(dum(MON, idx))
    X = np.column_stack(Xs)
    return y[idx] - X @ np.linalg.lstsq(X, y[idx], rcond=None)[0]
P('%-16s %16s %16s' % ('相关 (已剔除星期/月份)', '只用 1 月', '全年真值'))
for a, b in (('电价', '小区负载'), ('电价', '光伏'), ('小区负载', '光伏')):
    ra = np.corrcoef(resid(ys[a], J), resid(ys[b], J))[0, 1]
    rb = np.corrcoef(resid(ys[a], A), resid(ys[b], A))[0, 1]
    P('%-16s %16.4f %16.4f' % ('%s ~ %s' % (a, b), ra, rb))
P('')

```

```

P('=' * 104)
P('S4 其他只在 1 月能拿到的东西')
P('=' * 104)
P(' · 1 月实测均值: 电价 %.4f, 负荷 %.1f kW, 光伏 %.1f kW' % (ys['电价'][J].mean(), ys['小区负载'][J].mean(), ys['光伏']
[J].mean()))
P(' · 1 月光伏日出日落(>1kW): 平均 %s' % '-')
pos = np.array([np.where(PV[i] > 1)[0] for i in J], dtype=object)
first = np.mean([p[0] for p in pos if len(p)] * 10 / 60
last = np.mean([p[-1] + 1 for p in pos if len(p)] * 10 / 60
P(' 1 月平均发电时段 %.2f 时 ~ %.2f 时 (跨 %.2f h)' % (first, last, last - first))
posA = np.array([np.where(PV[i] > 1)[0] for i in A], dtype=object)
P(' 全年平均 %.2f 时 ~ %.2f 时' % (np.mean([p[0] for p in posA if len(p)] * 10 / 60,
np.mean([p[-1] + 1 for p in posA if len(p)] * 10 / 60))

try:
    Q3 = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
    pvf = Q3['pv_forecast']
    err = np.array([np.nanmean(np.asarray(pvf['%d_0' % d], float) - PV[d]) for d in range(365)])
    P(' 附件 3 0:00 版全天 MAE: 1 月 %.1f kW, 全年 %.1f kW -> 1 月偏低 %.0f%%'
    % (np.nanmean(np.abs(err[J])), np.nanmean(np.abs(err[A])),
    100 * (1 - np.nanmean(np.abs(err[J])) / np.nanmean(np.abs(err[A])))))
except Exception as e:
    P(' (附件 3 误差统计跳过: %s)' % e)

P('')

out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-diag', 'S_仅用 1 月可得性诊断.txt')
open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
print('written', out)

```

D.2.78 支撑材料/代码/06_可迁移框架/q4_selftransfer.py

-*- coding: utf-8 -*-

"""可迁移 · 自进化预测框架 (TSEF)

分解:

```

y_d(t) = Lev_d x Seas[m(d)] x ratio[c(d)] x Shp[c(d)](t)
· Shp / Seas / ratio --- 无量纲的"形状类", 可以跨站点搬运 (用收缩估计逐步让位给本地数据)
· Lev --- 有量纲的"水平类", 只能现场在线跟踪
· 光伏另加天文层 k*H0(phi,delta), phi 与 k 现场标定
自进化: 样本越多, 本地权重越大; 未观测到的月份先用先验。
"""
import os, sys, json, datetime as dt
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
def _updirs(n=6):
    here, out = HERE, []
    for _ in range(n):
        here = os.path.dirname(here)
        out.append(here)
    return out

```

兼容两种目录布局: 源码树 (paper_output_pro/code/) 与支撑材料 (支撑材料/代码/NN_xxx/)

```

for _d in list(_updirs()) + [HERE]:
    for _c in (os.path.join(_d, 'paper_output_pro', 'code'),
               os.path.join(_d, '代码', '05_数据分析'),
               os.path.join(_d, '支撑材料', '代码', '05_数据分析'),
               os.path.join(_d, '05_数据分析')):
        if os.path.isdir(_c):

```

```

        sys.path.insert(0, _c)
from jieqi import lon_of

def _find_data(name):
    cans = []
    for _d in list(_updirs()) + [HERE]:
        cans += [os.path.join(_d, 'paper_output_pro', 'data_cleaned', name),
                 os.path.join(_d, '数据', '清洗数据', name),
                 os.path.join(_d, '支撑材料', '数据', '清洗数据', name),
                 os.path.join(_d, 'data_cleaned', name)]
    for c in cans:
        if os.path.isfile(c):
            return c
    raise FileNotFoundError('找不到数据文件 %s，请设置环境变量 Q4_DATA_DIR' % name)

```

```

DAY = [dt.date(2025, 1, 1) + dt.timedelta(days=i) for i in range(365)]
WD = np.array([d.weekday() for d in DAY]); MON = np.array([d.month for d in DAY])
NAME7 = ['周一', '周二', '周三', '周四', '周五', '周六', '周日']
D = json.load(open(_find_data('q4_input.json'), encoding='utf-8'))
PR0 = np.asarray(D['price_actual'], float); LO0 = np.asarray(D['load_actual'], float)
PV0 = np.asarray(D['pv_actual'], float)
TH = np.array([lon_of(d) for d in DAY]); EPS = 23.4393
DEC = np.degrees(np.arcsin(np.sin(np.radians(EPS)) * np.sin(np.radians(TH))))
lo56 = np.array([w in (4, 5) for w in WD])
T = 144
TAU = (np.arange(T) + 0.5) * 10 / 60.0

```

```

def cosz(phi, d):
    """日内晴空形状，归一化到【均值为 1】（用于乘日均功率）"""
    w = np.radians((TAU - 12.0) * 15.0)
    z = np.sin(np.radians(phi)) * np.sin(np.radians(DEC[d])) + \
        np.cos(np.radians(phi)) * np.cos(np.radians(DEC[d])) * np.cos(w)
    z = np.maximum(z, 0.0)
    return z / max(z.mean(), 1e-9)

```

```

def H0(phi, d):
    ws = np.arccos(np.clip(-np.tan(np.radians(phi)) * np.tan(np.radians(DEC[d])), -1, 1))
    return (24 * 1.367 / np.pi) * (1 + 0.033 * np.cos(2 * np.pi * (d + 1) / 365.0)) * \
        (np.cos(np.radians(phi)) * np.cos(np.radians(DEC[d])) * np.sin(ws) +
         ws * np.sin(np.radians(phi)) * np.sin(np.radians(DEC[d])))

```

===== 从主站点反推生成参数

=====

```

def learn_structure():
    st = {}
    st['shape_load'] = {k: (LO0[lo56 == k] if k else LO0[~lo56]).mean(axis=0) for k in (True, False)}
    st['shape_price'] = {k: (PR0[lo56 == k] if k else PR0[~lo56]).mean(axis=0) for k in (True, False)}
    for k in (True, False):
        st['shape_load'][k] = st['shape_load'][k] / st['shape_load'][k].mean()
        st['shape_price'][k] = st['shape_price'][k] / st['shape_price'][k].mean()
    st['ratio_load'] = float(LO0[lo56].mean() / LO0[~lo56].mean())
    st['ratio_price'] = float(PR0[lo56].mean() / PR0[~lo56].mean())

```

```

st['mon_load'] = np.array([LO0[MON == k].mean() for k in range(1, 13)])
st['mon_price'] = np.array([PR0[MON == k].mean() for k in range(1, 13)])
st['mon_load'] /= st['mon_load'].mean(); st['mon_price'] /= st['mon_price'].mean()
r = np.zeros(365)
for k in (True, False):
    a = LO0[lo56 == k].mean(axis=1); r[lo56 == k] = a / a.mean() - 1
st['sd_load_day'] = float(r.std())
r = np.zeros(365)
for k in (True, False):
    a = PR0[lo56 == k].mean(axis=1); r[lo56 == k] = a / a.mean() - 1
st['sd_price_day'] = float(r.std())
e = np.array([LO0[d] / (LO0[d].mean() * st['shape_load'][bool(lo56[d]))] for d in range(365)])
st['sd_load_slot'] = float((e - 1).std())
e = np.array([PR0[d] / (PR0[d].mean() * st['shape_price'][bool(lo56[d]))] for d in range(365)])
st['sd_price_slot'] = float((e - 1).std())
pvday = PV0.mean(axis=1)
best = None
for phi in np.arange(10, 45.01, 0.5):
    H = np.array([H0(phi, d) for d in range(365)])
    k = float(np.dot(H, pvday) / np.dot(H, H))
    s = 1 - ((pvday - k * H) ** 2).sum() / ((pvday - pvday.mean()) ** 2).sum()
    if best is None or s > best[0]:
        best = (s, phi, k)
_, st['phi'], st['k_pv'] = best
H = np.array([H0(st['phi'], d) for d in range(365)])
kt = pvday / (st['k_pv'] * H)
st['mon_kt'] = np.array([kt[MON == k].mean() for k in range(1, 13)])
st['sd_kt'] = float(np.mean([kt[MON == k].std() for k in range(1, 13)]))
rp = np.zeros(365)
for k in (True, False):
    a = PR0[lo56 == k].mean(axis=1); rp[lo56 == k] = a / a.mean() - 1
dev = kt / st['mon_kt'][MON - 1] - 1
st['gamma'] = float(-np.polyfit(dev, rp, 1)[0])
return st

ST = learn_structure()

def gen_site(cfg, seed):
    """按测得的生成机制造变体站点。"""
    rng = np.random.default_rng(seed)
    phi = cfg.get('phi', ST['phi'])
    lod = cfg.get('low_days', (4, 5))
    sh = cfg.get('peak_shift', 0.0)
    amp = cfg.get('season_amp', 1.0)
    # 年内形状：把相对年均的偏离按 amp 缩放
    ml = 1.0 + (ST['mon_load'] - 1.0) * amp
    mp = 1.0 + (ST['mon_price'] - 1.0) * amp
    lo = np.array([w in lod for w in WD])

```

```

def roll(s, sh):
    return s if sh == 0 else np.roll(s, int(round(sh * 6)))

sl = {True: roll(ST['shape_load'][True], sh), False: roll(ST['shape_load'][False], sh)}
sp = {True: roll(ST['shape_price'][True], sh * 0.5), False: roll(ST['shape_price'][False], sh * 0.5)}
H = np.array([H0(phi, d) for d in range(365)])
kt = ST['mon_kt'][MON - 1] * np.exp(rng.normal(0, ST['sd_kt'], 365) - 0.5 * ST['sd_kt'] ** 2)
pv = np.zeros((365, T))
for d in range(365):
    pv[d] = ST['k_pv'] * H[d] * kt[d] * cosz(phi, d)
el = rng.normal(0, ST['sd_load_day'], 365)
ep = rng.normal(0, ST['sd_price_day'], 365)
load = np.zeros((365, T)); price = np.zeros((365, T))
BL = float(LO0.mean()); BP = float(PR0.mean())
for d in range(365):
    lv = ml[MON[d] - 1] * (ST['ratio_load'] if lo[d] else 1.0) * (1 + el[d])
    load[d] = BL * lv / max(ml.mean(), 1e-9) * sl[bool(lo[d])] * \
        (1 + rng.normal(0, ST['sd_load_slot'], T))
    pl = mp[MON[d] - 1] / max(mp.mean(), 1e-9) * (ST['ratio_price'] if lo[d] else 1.0) * (1 + ep[d]) \
        - ST['gamma'] * (kt[d] / ST['mon_kt'][MON[d] - 1] - 1)
    price[d] = BP * max(pl, 0.08) * sp[bool(lo[d])] * \
        (1 + rng.normal(0, ST['sd_price_slot'] * 0.8, T))
return dict(load=np.maximum(load, 0), pv=np.maximum(pv, 0), price=np.maximum(price, 0.01),
            lo=lo, phi=phi)

```

===== TSEF =====

```
CLS_CAND = [(4, 5), (5, 6), (4, 5, 6), (5,), (6,), (4,), (0, 1, 2, 3)]
```

```

class TSEF:
    def __init__(self, prior=None, n0=7.0, ema=0.25, rephi=7, phi_known=None):
        self.P = prior; self.n0 = n0 if prior else 0.0
        self.ema = ema; self.rephi = rephi
        self.L = {k: [] for k in ('load', 'pv', 'price')}
        self.cls = (4, 5)
        self.phi_locked = phi_known is not None
        self.phi = float(phi_known) if self.phi_locked else (prior or {}).get('phi', 20.0)
        self.k_pv = (prior or {}).get('k_pv', 300.0)
        self.Lv = {k: None for k in ('load', 'pv', 'price')}

    def _lo(self, d):
        return WD[d] in self.cls

    def observe(self, d, load, pv, price):
        self.L['load'].append(load); self.L['pv'].append(pv); self.L['price'].append(price)
        n = len(self.L['load'])

```

```

if n % 3 == 0:
    self._pick_cls()
if n % self.rephi == 0:
    self._fit_pv()

def _pick_cls(self):
    n = len(self.L['load'])
    if n < 10:
        return
    lv = np.array([self.L['load'][k].mean() for k in range(n)])
    best = None
    for cl in CLS_CAND:
        g = np.array([WD[k] in cl for k in range(n)])
        if g.all() or (~g).all():
            continue
        res = np.zeros(n)
        for v in (True, False):
            res[g == v] = lv[g == v] - lv[g == v].mean()
        if best is None or res.std() < best[0]:
            best = (res.std(), cl)
    if best:
        self.cls = best[1]

def _phi_from_daylength(self, ndays=7):
    """由光伏的起止时刻反推纬度：  $\cos(ws) = -\tan(\phi)\tan(\delta) \Rightarrow \phi = \text{atan}(-\cos(ws)/\tan(\delta))$ """
    n = len(self.L['pv'])
    if n < 3:
        return None
    est = []
    for k in range(n - min(ndays, n), n):
        y = self.L['pv'][k]
        if y.max() <= 0:
            continue
        idx = np.where(y > 0.02 * y.max())[0]
        if len(idx) < 8:
            continue
        half = (idx[-1] - idx[0] + 1) * 10 / 60.0 / 2.0      # 小时
        ws = np.radians(max(min(half * 15.0, 179.0), 1.0))
        td = np.tan(np.radians(DEC[k]))
        if abs(td) < 0.15:
            continue
        est.append(np.degrees(np.arctan(-np.cos(ws) / td)))
    if not est:
        return None
    return float(np.clip(np.median(est), 2.0, 60.0))

```

```

def _fit_pv(self):
    n = len(self.L['pv'])
    y = np.array([self.L['pv'][k].mean() for k in range(n)])
    kt = np.array([self._kt_season(MON[k], n) for k in range(n)])
    if self.phi_locked:
        h = np.array([H0(self.phi, k) * kt[k] for k in range(n)])
        self.k_pv = float(np.dot(h, y) / max(np.dot(h, h), 1e-9))
        return
    # 物理法：用观测到的"光伏起止时刻"反推纬度（日出日落时刻直接由纬度与赤纬决定）
    # 一周样本即可把纬度定到 ±3 度以内；纬度是地理常数，定住后只标定系数 k。
    ph = self._phi_from_daylength()
    if ph is not None:
        self.phi = ph
    h = np.array([H0(self.phi, k) * kt[k] for k in range(n)])
    self.k_pv = float(np.dot(h, y) / max(np.dot(h, h), 1e-9))

def _kt(self, k):
    return self._kt_season(MON[k], k)

def _kt_season(self, m, upto=None):
    """晴空指数的月因子（收缩估计）。先验不足时用历史同月比例。"""
    n = len(self.L['pv']) if upto is None else upto
    if self.P is None:
        if n == 0:
            return 1.0
        h = np.array([max(H0(self.phi, k), 1e-9) for k in range(n)])
        kth = np.array([self.L['pv'][k].mean() / (self.k_pv * h[k]) for k in range(n)])
        idx = [k for k in range(n) if MON[k] == m]
        return float(np.mean([kth[k] for k in idx])) if idx else float(kth[-7:].mean())
    pr = self.P['mon_kt'][m - 1]
    idx = [k for k in range(n) if MON[k] == m]
    if not idx:
        return pr
    h = np.array([max(H0(self.phi, k), 1e-9) for k in range(n)])
    loc = float(np.mean([self.L['pv'][k].mean() / (self.k_pv * h[k]) for k in idx]))
    w0 = self._w(self.n0, loc, pr)
    return (w0 * pr + len(idx) * loc) / (w0 + len(idx))

# ---- 形状 / 季节 收缩估计 ----
@staticmethod
def _w(n0, loc, pr, tol=0.35):
    """自适应收缩权重：本地与先验偏离越大，先验权重越小（先验错了要能快速放权）"""
    if pr is None or abs(pr) < 1e-12:
        return 0.0
    dev = abs(loc - pr) / abs(pr)
    return n0 * max(0.0, 1.0 - dev / tol) # tol 越大越温和

```

```

def _shape(self, kind, lo):
    n = len(self.L[kind])
    idx = [k for k in range(n) if self._lo(k) == lo]
    if not idx:
        idx = list(range(n))
    loc = np.mean([self.L[kind][k] / max(self.L[kind][k].mean(), 1e-9) for k in idx], axis=0)
    if self.P is None:
        return loc
    pr = self.P['shape_ %s' % kind][lo]
    w0 = self._w(self.n0, float(np.mean(np.abs(loc - pr)) / max(np.mean(pr), 1e-9)),
        1.0, tol=0.5)
    return (w0 * pr + len(idx) * loc) / (w0 + len(idx))

def _season(self, kind, m):
    if self.P is None:
        n = len(self.L[kind])
        allm = float(np.mean([self.L[kind][k].mean() for k in range(n)]))
        idx = [k for k in range(n) if MON[k] == m]
        return 1.0 if not idx else float(np.mean([self.L[kind][k].mean() for k in idx])) / max(allm, 1e-9)
    pr = self.P['mon_ %s' % kind][m - 1]
    idx = [k for k in range(len(self.L[kind])) if MON[k] == m]
    if not idx:
        return pr
    loc = float(np.mean([self.L[kind][k].mean() for k in idx])) / max(self.P['mean_ %s' % kind], 1e-9)
    nn = len(idx)
    w0 = self._w(self.n0, loc, pr, tol=0.9)
    return (w0 * pr + nn * loc) / (w0 + nn)

def _ratio(self, kind):
    n = len(self.L[kind])
    g = np.array([self._lo(k) for k in range(n)])
    lv = np.array([self.L[kind][k].mean() for k in range(n)])
    if g.sum() == 0 or (~g).sum() == 0:
        return self.P['ratio_ %s' % kind] if self.P else 1.0
    loc = float(lv[g].mean() / lv[~g].mean())
    if self.P is None:
        return loc
    w0 = self._w(self.n0, loc, self.P['ratio_ %s' % kind], tol=0.8)
    return (w0 * self.P['ratio_ %s' % kind] + g.sum() * loc) / (w0 + g.sum())

def _level(self, kind, lo):
    """当前水平 = 近 8 个同类日 的日均值 的指数加权 (剔除季节与日型后) """
    n = len(self.L[kind])
    idx = [k for k in range(n) if (WD[k] in self.cls) == lo][-8:]

```

```

if not idx:
    idx = list(range(max(0, n - 8), n))
    w = np.array([self.ema * (1 - self.ema) ** (len(idx) - 1 - i) for i in range(len(idx))])
    w /= w.sum()
    raw = float(np.sum([w[i] * self.L[kind][idx[i]].mean() for i in range(len(idx))]))
    den = float(np.sum([w[i] * self._season(kind, MON[idx[i]]) *
        (self._ratio(kind) if self._lo(idx[i]) else 1.0) for i in range(len(idx))]))
    return raw / max(den, 1e-9)

def predict(self, d):
    n = len(self.L['load'])
    lo = WD[d] in self.cls
    if n < 2:
        if self.P is None:
            return None
        L = self.P['mean_load'] * self.P['mon_load'][MON[d] - 1] * \
            (self.P['ratio_load'] if lo else 1.0) * self.P['shape_load'][lo]
        V = self.P['k_pv'] * H0(self.P['phi'], d) * self.P['mon_kt'][MON[d] - 1] * \
            cosz(self.P['phi'], d)
        R = self.P['mean_price'] * self.P['mon_price'][MON[d] - 1] * \
            (self.P['ratio_price'] if lo else 1.0) * self.P['shape_price'][lo]
        return L, V, R
    Lv_1 = self._level('load', lo)
    Lv_p = self._level('price', lo)
    L = Lv_1 * self._season('load', MON[d]) * (self._ratio('load') if lo else 1.0) * \
        self._shape('load', lo)
    R = Lv_p * self._season('price', MON[d]) * (self._ratio('price') if lo else 1.0) * \
        self._shape('price', lo)
    V = self.k_pv * H0(self.phi, d) * self._kt_season(MON[d]) * cosz(self.phi, d)
    return L, V, R

def make_prior():
    p = {'phi': ST['phi'], 'k_pv': ST['k_pv'],
        'mon_load': ST['mon_load'], 'mon_price': ST['mon_price'], 'mon_kt': ST['mon_kt'],
        'ratio_load': ST['ratio_load'], 'ratio_price': ST['ratio_price'],
        'shape_load': ST['shape_load'], 'shape_price': ST['shape_price'],
        'mean_load': float(LO0.mean()), 'mean_price': float(PR0.mean()),
        'mean_pv': float(PV0.mean())}
    return p

def run(site, prior, label, warm=0):
    M = TSEF(prior)
    errs = {'load': [], 'pv': [], 'price': []}
    for d in range(365):
        pr = M.predict(d)
        if pr is not None and pr[0] is not None and d >= warm:
            errs['load'].append(abs(pr[0].mean() - site['load'][d].mean()))
            errs['pv'].append(abs(pr[1].mean() - site['pv'][d].mean()))

```

```

        errs['price'].append(abs(pr[2].mean() - site['price'][d].mean()))
    M.observe(d, site['load'][d], site['pv'][d], site['price'][d])
    return {k: float(np.mean(v)) if v else None for k, v in errs.items()}

def oracle(site):
    """用该站点全年数据直接算月均 — 同一模型类的精度上界"""
    o = {}
    for kind in ('load', 'pv', 'price'):
        e = []
        for d in range(365):
            m = MON[d]; lo = site['lo'][d]
            idx = [k for k in range(365) if MON[k] == m and site['lo'][k] == lo and k != d]
            if not idx:
                idx = [k for k in range(365) if k != d]
            e.append(abs(np.mean([site[kind][k].mean() for k in idx]) - site[kind][d].mean()))
        o[kind] = float(np.mean(e))
    return o

```

D.2.79 支撑材料/代码/06_可迁移框架/q4_tsef_e2b.py

-*- coding: utf-8 -*-

```

"""E2b: 先验强度 n0 与"自适应放权"的交互（换习惯、换地点场景的适应期）。"""
import os, sys, json
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
sys.path.insert(0, HERE)
import q4_selftransfer as X

PRIOR = X.make_prior()
OUT = []
def P(*a):
    s = ''.join(str(x) for x in a); OUT.append(s); print(s, flush=True)

def run(site, n0, adapt, phi_known=None):
    orig = X.TSEF._w
    if not adapt:
        X.TSEF._w = staticmethod(lambda a, b, c, tol=0.35: (0.0 if (c is None or abs(c) < 1e-12) else a))
    try:
        M = X.TSEF(PRIOR, n0=n0, phi_known=phi_known)
        rec = {k: np.full(365, np.nan) for k in ('load', 'pv', 'price')}
        for d in range(365):
            pr = M.predict(d)
            if pr is not None and pr[0] is not None:
                rec['load'][d] = abs(pr[0].mean() - site['load'][d].mean())
                rec['pv'][d] = abs(pr[1].mean() - site['pv'][d].mean())
                rec['price'][d] = abs(pr[2].mean() - site['price'][d].mean())
            M.observe(d, site['load'][d], site['pv'][d], site['price'][d])
        return rec
    finally:
        if not adapt:
            X.TSEF._w = staticmethod(orig)

def mae(r, k, lo, hi):
    return float(np.nanmean(r[k][lo:hi]))

```

```

def adapt_days(rec, k, factor=1.2):
    """连续 21 天滚动 MAE 稳定进入全期水平 factor 倍以内的第一天"""
    r = np.nan_to_num(rec[k], nan=1e9); tgt = np.nanmean(rec[k]) * factor
    for d in range(20, 365):
        if np.all(r[d - 20:d] <= tgt):
            return d - 20
    return None

res = {}
for name, cfg, seed, pk in (('V2 换生活习惯', dict(low_days=(5, 6), peak_shift=0.55), 303, None),
                             ('V3 换地点+换习惯', dict(phi=38.0, low_days=(6,), peak_shift=0.4, season_amp=0.8), 404, None)):
    site = X.gen_site(cfg, seed)
    P('=' * 96)
    P('%s: 先验强度 n0 与自适应放权的交互' % name)
    P('%-8s %-6s %-8s %-9s %-9s %-9s' % ('变量', 'n0', '自适应', '0-7 天', '7-30 天', '全期', '适应期(天)'))
    res[name] = {}
    for n0 in (7.0, 30.0):
        for adapt in (True, False):
            rec = run(site, n0, adapt)
            for kind, sc in (('load', 1.0), ('pv', 1.0), ('price', 1000.0)):
                P('%-8s %-6.0f %-8s %-9.1f %-9.1f %-9.1f' %
                  (kind, n0, '开' if adapt else '关', mae(rec, kind, 0, 7) * sc,
                   mae(rec, kind, 7, 30) * sc, mae(rec, kind, 0, 365) * sc, adapt_days(rec, kind)))
                res[name]['%s_n%d_%s' % (kind, int(n0), 'on' if adapt else 'off')] = \
                    dict(d0_7=mae(rec, kind, 0, 7) * sc, d7_30=mae(rec, kind, 7, 30) * sc,
                        full=mae(rec, kind, 0, 365) * sc, adapt=adapt_days(rec, kind))
    with open(os.path.join(HERE, 'q4_tsef_e2b_results.json'), 'w', encoding='utf-8') as f:
        json.dump(res, f, ensure_ascii=False, indent=1)
    with open(os.path.join(HERE, 'q4_tsef_e2b_log.txt'), 'w', encoding='utf-8') as f:
        f.write(chr(10).join(OUT))
P('done')

```

D.2.80 支撑材料/代码/06_可迁移框架/q4_tsef_extra.py

-*- coding: utf-8 -*-

"""第 7 章补充实验 (E1 学习曲线 / E2 自适应放权消融 / E3 逐层搬运消融)。

沿用 q4_selftransfer.py 的模型与 q4_tsef_testbed.py 的变体站点生成机制，
不改变任何既有结论，只把“框架机制为何有效”分解为可量化的三组对照。

```

"""
import os, sys, json
import numpy as np

HERE = os.path.dirname(os.path.abspath(__file__))
sys.path.insert(0, HERE)
import q4_selftransfer as X

MON = X.MON
PRIOR = X.make_prior()
VARIANTS = [
    ('V0 原站点·第二年', dict(), 101),
    ('V1 换地点 23.5→40', dict(phi=40.0, season_amp=0.75), 202),
    ('V2 换生活习惯', dict(low_days=(5, 6), peak_shift=0.55), 303),
    ('V3 换地点+换习惯', dict(phi=38.0, low_days=(6,), peak_shift=0.4, season_amp=0.8), 404),
]
WINS = [('第 0-7 天', 0, 7), ('第 7-30 天', 7, 30), ('第 30-90 天', 30, 90),

```

```

        ('第 90-180 天', 90, 180), ('第 180-365 天', 180, 365), ('全期', 0, 365)]
OUT = []

def P(*a):
    s = ''.join(str(x) for x in a)
    OUT.append(s)
    print(s)

def run(site, prior, phi_known=None, no_adapt=False):
    """no_adapt=True 时关闭"先验与本地偏离越大、先验权重越小"的自适应放权"""
    orig = X.TSEF._w
    if no_adapt:
        X.TSEF._w = staticmethod(lambda n0, loc, pr, tol=0.35:
                                (0.0 if (pr is None or abs(pr) < 1e-12) else n0))
    try:
        M = X.TSEF(prior, phi_known=phi_known)
        rec = {k: np.full(365, np.nan) for k in ('load', 'pv', 'price')}
        for d in range(365):
            pr = M.predict(d)
            if pr is not None and pr[0] is not None:
                rec['load'][d] = abs(pr[0].mean() - site['load'][d].mean())
                rec['pv'][d] = abs(pr[1].mean() - site['pv'][d].mean())
                rec['price'][d] = abs(pr[2].mean() - site['price'][d].mean())
            M.observe(d, site['load'][d], site['pv'][d], site['price'][d])
        return rec
    finally:
        if no_adapt:
            X.TSEF._w = staticmethod(orig)

def mae(rec, k, lo, hi):
    return float(np.nanmean(rec[k][lo:hi]))

def conv(rec, k, orc, factor=1.25):
    """连续 21 天滚动 MAE 稳定进入先知参照 factor 倍以内的第一天"""
    r = np.nan_to_num(rec[k], nan=1e9)
    for d in range(20, 365):
        if np.all(r[d - 20:d] <= orc * factor):
            return d - 20
    return None

def neutral_shapes(p):
    q = dict(p)
    for k in ('load', 'price'):
        q['mon_%s' % k] = np.ones(12)
        q['ratio_%s' % k] = 1.0
        q['shape_%s' % k] = {False: np.ones(144), True: np.ones(144)}
    q['mon_kt'] = np.ones(12)
    return q

def wrong_astro(p, phi=23.5):

```

```

q = dict(p)
q['phi'] = float(phi)
return q

sites = {}
orcs = {}
for name, cfg, seed in VARIANTS:
    s = X.gen_site(cfg, seed)

    sites[name] = s
    orcs[name] = X.oracle(s)

```

----- E1 学习曲线 -----

```

P('=' * 100)
P('E1 学习曲线: 迁移(带先验, 纬度取变体真值) 下各时段的日均值预测 MAE')
P('=' * 100)
P('%-22s %-6s %8s %8s %8s %8s %8s %8s' %
    ('变体', '变量', '第0天', '0-7天', '7-30天', '30-90天', '90-180天', '180-365', '全期'))
e1 = {}
for name, cfg, seed in VARIANTS:
    site = sites[name]

    rec = run(site, PRIOR, phi_known=cfg.get('phi', X.ST['phi']))
    e1[name] = {}
    for kind, sc in (('load', 1.0), ('pv', 1.0), ('price', 1000.0)):
        vals = [mae(rec, kind, lo, hi) * sc for _, lo, hi in WINS]
        d0 = float(np.nan_to_num(rec[kind][0], nan=0.0)) * sc
        P('%-22s %-6s %8.1f %8.1f %8.1f %8.1f %8.1f %8.1f %8.1f' %
            (name if kind == 'load' else "", kind, d0, *vals))
        e1[name][kind] = dict(zip([w[0] for w in WINS], vals))
    for kind in ('load', 'pv', 'price'):
        e1[name][kind + '_conv125'] = conv(rec, kind, orcs[name][kind], 1.25)
        e1[name][kind + '_conv110'] = conv(rec, kind, orcs[name][kind], 1.10)
        e1[name][kind + '_oracle'] = orcs[name][kind]
P('%-22s %-6s 收敛天数(1.25/1.10 倍先知参照): 负荷 %s/%s 光伏 %s/%s 电价 %s/%s' %
    ("", e1[name]['load_conv125'], e1[name]['load_conv110'],
    e1[name]['pv_conv125'], e1[name]['pv_conv110'],
    e1[name]['price_conv125'], e1[name]['price_conv110']))

```

----- E2 自适应放权消融 -----

```

P('')
P('=' * 100)
P('E2 自适应放权消融: 关闭"偏离度降权"(先验权重恒为 n0) 后的影响')
P('=' * 100)
P('%-22s %-6s %10s %10s %10s %10s %10s' %
    ('变体', '变量', '0-30天(自适)', '0-30天(关闭)', '全期(自适)', '全期(关闭)', '全期变化'))
e2 = {}
for name, cfg, seed in VARIANTS:
    site = sites[name]

    a = run(site, PRIOR, phi_known=cfg.get('phi', X.ST['phi']))
    b = run(site, PRIOR, phi_known=cfg.get('phi', X.ST['phi']), no_adapt=True)
    e2[name] = {}
    for kind, sc in (('load', 1.0), ('pv', 1.0), ('price', 1000.0)):
        a30, b30 = mae(a, kind, 0, 30) * sc, mae(b, kind, 0, 30) * sc
        af, bf = mae(a, kind, 0, 365) * sc, mae(b, kind, 0, 365) * sc

```

```

P('%-22s %-6s %10.1f %10.1f %10.1f %10.1f %9.1f%%' %
    (name if kind == 'load' else "", kind, a30, b30, af, bf, 100 * (bf / af - 1)))
e2[name][kind] = dict(a30=a30, b30=b30, af=af, bf=bf)
e2[name]['pv_conv_a'] = conv(a, 'pv', orcs[name]['pv'], 1.25)
e2[name]['pv_conv_b'] = conv(b, 'pv', orcs[name]['pv'], 1.25)
e2[name]['load_conv_a'] = conv(a, 'load', orcs[name]['load'], 1.25)
e2[name]['load_conv_b'] = conv(b, 'load', orcs[name]['load'], 1.25)
P('%-22s 收敛天数(自适应/关闭): 负荷 %s/%s, 光伏 %s/%s' %
    ('', e2[name]['load_conv_a'], e2[name]['load_conv_b'],
    e2[name]['pv_conv_a'], e2[name]['pv_conv_b']))

```

----- E3 逐层搬运消融 -----

```

P('')
P('=' * 100)
P('E3 逐层搬运消融: 纬度未知 (在线辨识) 时, 先验各层的贡献')
P('=' * 100)
P('%-22s %-18s %9s %9s %9s' % ('变体', '先验配置', '0-30天', '全期', '相对全层'))
e3 = {}
CFGs = [('全层先验 (本文)', lambda p: p),
        ('仅形状层 (天文层置旧纬度)', lambda p: wrong_astro(p)),
        ('仅天文层 (形状层置中性)', lambda p: neutral_shapes(p)),
        ('无先验 (纯冷启动)', None)]

for name, cfg, seed in VARIANTS:
    site = sites[name]
    e3[name] = {}
    base = None
    for label, f in CFGs:
        if f is None:
            rec = run(site, None)
        else:
            rec = run(site, f(PRIOR))
        for kind, sc in (('load', 1.0), ('pv', 1.0), ('price', 1000.0)):
            v30 = mae(rec, kind, 0, 30) * sc
            vall = mae(rec, kind, 0, 365) * sc
            e3[name].setdefault(kind, {})[label] = (v30, vall)
        if label.startswith('全层'):
            base = {k: e3[name][k][label][1] for k in ('load', 'pv', 'price')}
    for kind in ('load', 'pv', 'price'):
        for label, _ in CFGs:
            v30, vall = e3[name][kind][label]
            P('%-22s %-18s %9.1f %9.1f %8.1f%%' %
                (name if kind == 'load' else "", label + '/' + kind, v30, vall,
                100 * (vall / base[kind] - 1)))

P('')
P('=' * 100)
res = {'E1': e1, 'E2': e2, 'E3': e3}
o = os.path.join(HERE, 'q4_tsef_extra_results.json')
with open(o, 'w', encoding='utf-8') as f:
    json.dump(res, f, ensure_ascii=False, indent=1, default=float)

P('结果已写入 ' + o)
with open(os.path.join(HERE, 'q4_tsef_extra_log.txt'), 'w', encoding='utf-8') as f:
    f.write(chr(10).join(OUT))

```

D.2.81 支撑材料/代码/06_可迁移框架/q4_tsef_testbed.py

-*- coding: utf-8 -*-

```
"""TSEF 变体站点试验台 v2: 第二年 / 换地点(已知纬度) / 换地点(未知纬度) / 换生活习惯 / 冷启动."""
import os, sys, json, datetime as dt
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
sys.path.insert(0, HERE)
import q4_selftransfer as X
OUT = []
def P(*a): OUT.append(' '.join(str(x) for x in a))
MON = X.MON
PRIOR = X.make_prior()
WINS = ((0, 30), (30, 90), (90, 180), (180, 365), (0, 365))

def run(site, prior, phi_known=None):
    M = X.TSEF(prior, phi_known=phi_known)
    rec = {k: np.full(365, np.nan) for k in ('load', 'pv', 'price')}
    for d in range(365):
        pr = M.predict(d)
        if pr is not None and pr[0] is not None:
            rec['load'][d] = abs(pr[0].mean() - site['load'][d].mean())
            rec['pv'][d] = abs(pr[1].mean() - site['pv'][d].mean())
            rec['price'][d] = abs(pr[2].mean() - site['price'][d].mean())
        M.observe(d, site['load'][d], site['pv'][d], site['price'][d])
    return rec

def mae(rec, k, w):
    return float(np.nanmean(rec[k][w[0]:w[1]]))

def conv_days(rec, k, orc, factor=1.25):
    """连续 21 天滚动 MAE 稳定进入先知上界 factor 倍以内的第一天"""
    r = rec[k].copy(); ok = None
    for d in range(20, 365):
        if np.all(np.nan_to_num(r[d - 20:d], nan=1e9) <= orc * factor):
            ok = d - 20; break
    return ok

VARIANTS = [
    ('V0 原站点 第二年', dict(), 101),
    ('V1 换地点 纬度 23.5→40', dict(phi=40.0, season_amp=0.75), 202),
    ('V2 换生活习惯 低日型改周六日+晚峰', dict(low_days=(5, 6), peak_shift=0.55), 303),
    ('V3 换地点+换习惯', dict(phi=38.0, low_days=(6,), peak_shift=0.4, season_amp=0.8), 404),
]
P('=' * 112)
P('TSEF 可迁移 · 自进化预测框架 — 变体站点试验台 v2')
P('=' * 112)
P('先验来自 2025 主站点一年数据。变体站点用【从主站点反推的生成机制】生成。')
P('三种部署模式:')
P(' (a) 迁移+已知纬度 — 纬度是地理事实(查地图即得), 不当作待学参数')
P(' (b) 迁移+未知纬度 — 纬度也靠数据在线辨识')
P(' (c) 纯冷启动 — 不给任何先验, 一切从零学')
P('指标: 日均值预测 MAE (负荷/光伏 kW, 电价 元/kWh×1000); "收敛"= 连续 21 天进入先知上界 1.25 倍以内')
P('')
for name, cfg, seed in VARIANTS:
    site = X.gen_site(cfg, seed)
    orc = X.oracle(site)
    a = run(site, PRIOR, phi_known=cfg.get('phi', X.ST['phi']))
    b = run(site, PRIOR)
```

```

c = run(site, None)
P(' ' * 112)
P('%s 真实低日型=%s 真实纬度=%.0f 季节振幅=%.2f'
   % (name, [X.NAME7[w] for w in cfg.get('low_days', (4, 5))], cfg.get('phi', X.ST['phi']),
       cfg.get('season_amp', 1.0)))
P('%-8s %-16s %8s %8s %8s %8s %8s %8s' % ('变量', '模式', '第 0 天', '第 0-30', '第 30-90',
                                             '第 90-180', '第 180-365', '全期', '收敛天'))
for kind, sc in (('load', 1.0), ('pv', 1.0), ('price', 1000.0)):
    for tag, rec in (('迁移+知纬度', a), ('迁移+未知纬', b), ('纯冷启动', c)):
        cd = conv_days(rec, kind, orc[kind])
        P('%-8s %-16s %8.1f %8.1f %8.1f %8.1f %8.1f %8.1f %8s'
           % (kind if tag.startswith('迁移+知') else '', tag,
              np.nan_to_num(rec[kind][0], nan=float('nan')) * sc if not np.isnan(rec[kind][0]) else float('nan'),
              mae(rec, kind, WINS[0]) * sc, mae(rec, kind, WINS[1]) * sc,
              mae(rec, kind, WINS[2]) * sc, mae(rec, kind, WINS[3]) * sc,
              mae(rec, kind, WINS[4]) * sc, ('—' if cd is None else str(cd))))
        P('%-8s %-16s %8s %8s %8s %8s %8s %8s %8.1f' % ('', '★先知上界', '—', '—', '—', '—', '—', orc[kind] * sc))
P('')
P(' 迁移相对冷启动的全期改善: 负荷 %+.1f%%, 光伏 %+.1f%%, 电价 %+.1f%%'
   % (100 * (mae(c, 'load', WINS[4]) / mae(a, 'load', WINS[4]) - 1),
       100 * (mae(c, 'pv', WINS[4]) / mae(a, 'pv', WINS[4]) - 1),
       100 * (mae(c, 'price', WINS[4]) / mae(a, 'price', WINS[4]) - 1)))
P('')
P(' ' * 112)
P('零样本能力: 第 0 天 (还没见过任何一天数据) 就给出的预测')
P(' ' * 112)
for name, cfg, seed in VARIANTS:
    site = X.gen_site(cfg, seed)
    M = X.TSEF(PRIOR, phi_known=cfg.get('phi', X.ST['phi']))
    pr = M.predict(0)
    P(' %-30s 负荷 %8.1f / 实际 %8.1f (%+5.1f%%) 光伏 %7.1f / %7.1f (%+5.1f%%) 电价 %4f / %4f (%+5.1f%%)'
       % (name, pr[0].mean(), site['load'][0].mean(), 100 * (pr[0].mean() / site['load'][0].mean() - 1),
          pr[1].mean(), site['pv'][0].mean(), 100 * (pr[1].mean() / max(site['pv'][0].mean(), 1e-9) - 1),
          pr[2].mean(), site['price'][0].mean(), 100 * (pr[2].mean() / site['price'][0].mean() - 1)))
P('')
out = os.path.join(ROOT, 'paper_output_pro', 'experiments', 'q4-selftransfer', 'TSEF_变体站点报告.txt')
os.makedirs(os.path.dirname(out), exist_ok=True)
open(out, 'w', encoding='utf-8').write('\n'.join(OUT))
print('written', out)

```

D.2.82 支撑材料/代码/07_交付物生成/make_check_tables_v3.py

-*- coding: utf-8 -*-

"""检验表 v3 — 把附件原始输入与全部中间量、费用合并成一张总表。

列: 日期 | 时段 | 电价(元/kWh) | 电价来源 | 小区负载(kW) | 光伏实际出力(kW) | 净负荷(kW) | 光伏预报(kW)
 | 计划购电量(kWh) | 调整购电量(kWh) | 充电量(kWh) | 放电量(kWh) | 紧急购电量(kWh) | 时段末储电量(kWh)
 | 计划购入电费(元) | 实际使用电费(元) | 损耗电费(元) | 充电电费(元) | 放电电费(元) | 紧急购电费(元)
 | 偏差费用(元) | 偏差方向 | 成交购电费(元) | 本时段结算费用(元)

```

"""
import os, sys, io, json, datetime
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
import numpy as np, openpyxl
ROOT = os.environ.get('CUMCM_ROOT', '.')
B = os.path.join(ROOT, 'paper_output_pro'); OUT = os.path.join(ROOT, '结果IV', '检验材料', '逐10分钟明细')
sys.path.insert(0, os.path.join(B, 'code'))
DT = 1/6.0; FIRST = 31

```

```

HDR = ['日期', '时段', '电价(元/kWh)', '电价来源', '小区负载(kW)', '光伏实际出力(kW)', '净负荷(kW)', '光伏预报(kW)',
       '计划购电量(kWh)', '调整购电量(kWh)', '充电量(kWh)', '放电量(kWh)', '紧急购电量(kWh)', '时段末储电量(kWh)',
       '计划购入电费(元)', '实际使用电费(元)', '损耗电费(元)', '充电电费(元)', '放电电费(元)', '紧急购电费(元)',
       '偏差费用(元)', '偏差方向', '成交购电费(元)', '本时段结算费用(元)']

def lbl(i):
    a=i*10; b=a+10; f=lambda m: "%d:%02d"%(m//60,m%60)
    return '%s-%s'%(f(a), '0:00+1' if b==1440 else f(b))

def write(fn, title, days, eta):
    wb=openpyxl.Workbook(); ws=wb.active; ws.title=title
    ws.append(HDR)
    acc=np.zeros(15); nb=ns=nn=0
    for (date,pr,load,pv,fcst,netd,p_p,Pf,c2,d2,E,S) in days:
        loss=(1-eta)*c2*DT+((1-eta)/eta)*d2*DT
        use=pr*np.maximum(0.0,netd)*DT
        u=np.maximum(0,p_p-Pf); v=np.maximum(0,Pf-p_p)
        dev=0.5*pr*u*DT+1.5*pr*v*DT
        deal=pr*np.minimum(p_p,Pf)*DT
        emg=5.0*pr*E*DT
        settle=deal+dev+emg
        cost=[pr*Pf*DT,use,pr*loss,pr*c2*DT,pr*d2*DT,emg]
        for t in range(len(pr)):
            dv=float(dev[t])
            if abs(Pf[t]-p_p[t])<1e-9: dirn='无'; nn+=1
            elif Pf[t]>p_p[t]: dirn='多买'; nb+=1
            else: dirn='少买'; ns+=1
            row=[str(date),lbl(t),round(float(pr[t]),6),SRC,
                 round(float(load[t]),6),round(float(pv[t]),6),round(float(netd[t]),6),
                 (" if fcst is None else round(float(fcst[t]),6)),
                 round(float(p_p[t]*DT),6),round(float(Pf[t]*DT),6),
                 round(float(c2[t]*DT),6),round(float(d2[t]*DT),6),round(float(E[t]*DT),6),
                 round(float(S[t+1]),6)] + [round(float(x[t]),6) for x in cost] + \
                 [round(dv,6),dirn,round(float(deal[t]),6),round(float(settle[t]),6)]
            ws.append(row)
            acc+=np.array([row[14],row[15],row[16],row[17],row[18],row[19],row[20],row[22],row[23]]+[0]*6)
        ws.append(['合计',"","","","","","","","")+
                 [round(float(x),2) for x in acc[:6]]+
                 [round(float(acc[6]),2),'多买 %d / 少买 %d / 未改 %d 段'%(nb,ns,nn),
                 round(float(acc[7]),2),round(float(acc[8]),2)])
    for i,w in enumerate([12,13,13,12,14,16,13,13,15,15,12,12,15,16,16,16,13,13,13,15,13,24,15,17],1):
        ws.column_dimensions[openpyxl.utils.get_column_letter(i)].width=w
    ws.freeze_panes='C2'
    wb.save(os.path.join(OUT,fn))
    return acc

```

===== 问题一 =====

```
from scipy.optimize import linprog
i1=json.load(open(os.path.join(B,'data_cleaned','q1_input.json'),encoding='utf-8'))
pr1=np.asarray(i1['price'],float); ld1=np.asarray(i1['load'],float); pv1=np.asarray(i1['pv'],float)
net1=ld1-pv1; st=i1['storage']; eta=float(st['efficiency'])
Pmax=float(st['power_max']); smin=float(st['soc_min']); smax=float(st['soc_max']); s0=float(st['soc_initial'])
T=len(pr1); n=3*T+(T+1); og,oc,od,os=0,T,2*T,3*T
c=np.zeros(n); c[og:og+T]=pr1*DT
A=np.zeros((T,n))
for t in range(T): A[t,og+t]=-1; A[t,od+t]=-1; A[t,oc+t]=1
Ae=np.zeros((T+1,n))
for t in range(T):
    Ae[t,os_+t+1]=1; Ae[t,os_+t]=-1; Ae[t,oc+t]=-eta*DT; Ae[t,od+t]=DT/eta

Ae[T,os_+T]=1
be=np.zeros(T+1); be[T]=s0
bnd=[(0,None)]*T+[(0,Pmax)]*T+[(0,Pmax)]*T+[(s0,s0)]+[(smin,smax)]*T
r=linprog(c,_A_ub=A,_b_ub=-net1,_A_eq=Ae,_b_eq=be,bounds=bnd,method='highs'); assert r.success
x=r.x; G=x[og:og+T]; cc=x[oc:oc+T]; dd=x[od:od+T]; SS=x[os_+T+1]
SRC='附件 1'
a1=write('检验_问题一.xlsx','问题一',[datetime.date(2025,1,1),pr1,ld1,pv1,None,net1,G,G,cc,dd,np.zeros(T),SS],eta)
print('问题一 结算合计=%.2f 论文 35126.95' % a1[8])
```

===== 问题二 =====

```
from q2_forecast import build_forecast, margin, execute_causal
from q2_online import day_lp
i2=json.load(open(os.path.join(B,'data_cleaned','q2_input.json'),encoding='utf-8'))
pr2=np.asarray(i2['price'],float); ld2=np.asarray(i2['load_actual'],float); pv2=np.asarray(i2['pv_actual'],float)
net2=ld2-pv2; st2=i2['storage']; dt=float(i2['dt']); eta=float(st2['efficiency'])
Pm=float(st2['power_max']); sm=float(st2['soc_min']); sx=float(st2['soc_max'])
Nh,er,_,_=build_forecast(net2,0.10,0.90); s=float(st2['soc_initial']); d2l=[]
for d in range(1,365):
    g,_,_,_=day_lp(pr2,Nh[d]+margin(er,d,5,0.88),s,eta,dt,Pm,sm,sx)
    e,c2,dd2,S=execute_causal(g,net2[d],s,eta,dt,Pm,sm,sx)
    if d>=FIRST:
        d2l.append((datetime.date(2025,1,1)+datetime.timedelta(days=d),pr2,ld2[d],pv2[d],None,net2[d],g,g,c2,dd2,e,S))
    s=float(S[-1])
SRC='附件 1'
a2=write('检验_问题二.xlsx','问题二',d2l,eta)
print('问题二 结算合计=%.2f 论文 13700048.95' % a2[8])
```

===== 问题三 / 四 =====

```
from q3_mpc import build_seasonal as bs3, solve_window as sw3
from q4_solver import seasonal as bs4, solve_w as sw4
def run(which):
    f='q3_input.json' if which=='3' else 'q4_input.json'
    inp=json.load(open(os.path.join(B,'data_cleaned',f),encoding='utf-8'))
    load=np.asarray(inp['load_actual'],float); pv=np.asarray(inp['pv_actual'],float); net=load-pv
    D,T=load.shape
    if which=='3':
        pr_all=np.tile(np.asarray(inp['price'],float),(D,1)); pr_fc=pr_all; bsf,swf=bs3,sw3; src='附件 1'
    else:
        pr_all=np.asarray(inp['price_actual'],float); bsf,swf=bs4,sw4; pr_fc=bsf(pr_all,D,T,0.15,0.8); src='附件 4'
    st=inp['storage']; dt=float(inp['dt']); eta=float(st['efficiency'])
    Pmax=float(st['power_max']); smin=float(st['soc_min']); smax=float(st['soc_max'])
    fc={h:np.array([inp['pv_forecast'][f'%d_%%d'%(d,h)] for d in range(D)]) for h in (0,6,12,18)}
    lf=bsf(load,D,T,0.15,0.8); pfs=bsf(pv,D,T,0.15,0.8)
    def nf(d,h):
        a=fc[h][d].copy(); a[np.isnan(a)]=pfs[d][np.isnan(a)]; return lf[d]-a
```

```

s0=float(st['soc_initial']); out=[]
for d in range(D):
    m=np.zeros(T) if d<2 else np.maximum(np.percentile(
        np.array([net[k]-nf(k,0) for k in range(max(1,d-14),d)]),80.0,axis=0),0.0)
    p_p,c_p,dd_p,soc=swf(pr_fc[d],nf(d,0)+m,s0,dt,eta,Pmax,smin,smax)
    Pf=p_p.copy(); sct=soc.copy()
    for h,slot in ((6,36),(12,72),(18,108)):
        p_c,d_s_=swf(pr_fc[d][slot:],(nf(d,h)+m)[slot:],sct[slot],dt,eta,Pmax,smin,smax,plan_w=p_p[slot:])
        Pf[slot:]=p_; sct=np.concatenate([sct[:slot],s_])
    e=np.zeros(T); c2=np.zeros(T); dd2=np.zeros(T); SS=np.zeros(T+1); ss=s0; SS[0]=ss
    for t in range(T):
        need=net[d][t]-Pf[t]
        if need>0:
            cap=min(Pmax,max(ss-smin,0.0)*eta/dt); dd2[t]=min(need,cap); e[t]=need-dd2[t]; ss-=dd2[t]*dt/eta
        else:
            cap=min(Pmax,max(smax-ss,0.0)/(eta*dt)); c2[t]=min(-need,cap); ss+=eta*c2[t]*dt
        SS[t+1]=ss
    if d>=FIRST:
        f0=nf(d,0)
        out.append((datetime.date(2025,1,1)+datetime.timedelta(days=d),pr_all[d],load[d],pv[d],f0,
            net[d],p_p,Pf,c2,dd2,e,SS))
    s0=ss
    return out,eta,src
for w,nm,exp in (('3','问题三',14483771.06),('4','问题四',15192155.84)):
    days,eta,src=run(w); SRC=src
    aa=write('检验_%s.xlsx'%nm,nm,days,eta)
    print('%s  结算合计=%0.2f  论文  %0.2f  %s'%(nm,aa[8],exp,'OK' if abs(aa[8]-exp)<2 else 'X'))

```

D.2.83 支撑材料/代码/07_交付物生成/q3_fc_write_check_table.py

-*- coding: utf-8 -*-

"""问题三 逐10分钟检验总表（正式口径 E3v2：四版决策 + 扩展窗 σ 逆方差加权融合）。

列结构与 make_check_tables_v3.py 完全一致（24 列），但：

- 第 8 列「光伏预报(kW)」填【融合后的光伏预报】——旧版误填了「净负荷预报」，此处更正
- 计划/调整/充放电/紧急/SOC 全部取自正式解 solution.json，不再自行重解

用法: python q3_fc_write_check_table.py <solution.json> <sigma.npy> <out.xlsx>

```

"""
import os, sys, io, json, datetime
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
import numpy as np, openpyxl
ROOT = os.path.abspath('.')
CODE = os.path.join(ROOT, 'paper_output_pro', 'code')
sys.path.insert(0, CODE)
from q3_forecast_B import load_vintages, candidates, HOURS, NDAY, NH, T as TT
from q3_mpc import build_seasonal

DT = 1/6.0; FIRST = 31
SOLP, SIGP, OUTP = sys.argv[1], sys.argv[2], sys.argv[3]
HDR = ['日期', '时段', '电价(元/kWh)', '电价来源', '小区负载(kW)', '光伏实际出力(kW)', '净负荷(kW)', '光伏预报(kW)',
    '计划购电量(kWh)', '调整购电量(kWh)', '充电量(kWh)', '放电量(kWh)', '紧急购电量(kWh)', '时段末储电量(kWh)',
    '计划购入电费(元)', '实际使用电费(元)', '损耗电费(元)', '充电电费(元)', '放电电费(元)', '紧急购电费(元)',
    '偏差费用(元)', '偏差方向', '成交购电费(元)', '本时段结算费用(元)']

def lbl(i):
    a = i*10; b = a+10

```

```

f = lambda m: '%d:%02d' % (m//60, m % 60)

return '%s-%s' % (f(a), '0:00+1' if b == 1440 else f(b))

inp = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
load = np.asarray(inp['load_actual'], float); pv = np.asarray(inp['pv_actual'], float)
pr_all = np.tile(np.asarray(inp['price'], float), (load.shape[0], 1))
st = inp['storage']; eta = float(st['efficiency'])
V, _ = load_vintages(os.path.join(CODE, 'q3_vintages_B.json'))
TAB = np.load(SIGP)
SOL = json.load(open(SOLP, encoding='utf-8'))
pv_fs = build_seasonal(pv, load.shape[0], 144, 0.15, 0.8)

def fused_pv(d, stage_off):
    """第 d 天、决策时刻 stage_off 时点的融合光伏预报 (144,)"""
    dec = d*24 + stage_off
    out = np.full(144, np.nan)
    row = TAB[min(d, TAB.shape[0]-1)]
    for t in range(144):
        c = candidates(V, d, t, dec, 'all')
        if not c:
            continue
        th = t // 6
        ws = np.asarray([1.0/row[th, j]**2 for _, j in c], float)
        vals = np.asarray([v for _, v in c], float)
        out[t] = float(np.dot(ws, vals)/ws.sum())
    miss = ~np.isfinite(out)
    out[miss] = pv_fs[d][miss]
    return out

wb = openpyxl.Workbook(); ws = wb.active; ws.title = '问题三'
ws.append(HDR)
acc = np.zeros(15); nb = ns = nn = 0
SRC = '附件 1'
for i, dobj in enumerate(SOL['days']):
    d = int(dobj)

    date = datetime.date(2025, 1, 1) + datetime.timedelta(days=d)

    p_p = np.asarray(SOL['plan0'][i], float); Pf = np.asarray(SOL['plan'][i], float)
    c2 = np.asarray(SOL['c'][i], float); d2 = np.asarray(SOL['d'][i], float)
    E = np.asarray(SOL['E'][i], float)
    S0 = float(SOL['S0'][i]); S1 = float(SOL['S1'][i])
    SS = np.concatenate([np.linspace(S0, S1, 145)]) # 仅用于「时段末储电量」展示? -> 改用逐段重算
    # 逐段 SOC 重算 (与求解器实时执行完全一致)
    SS = np.zeros(145); SS[0] = S0; ss = S0
    for t in range(144):
        ss += (eta*c2[t]*DT) - (d2[t]*DT/eta)
        SS[t+1] = ss
    fcst = fused_pv(d, 0)
    netd = load[d] - pv[d]
    pr = pr_all[d]
    loss = (1-eta)*c2*DT + ((1-eta)/eta)*d2*DT
    use = pr*np.maximum(0.0, netd)*DT
    u = np.maximum(0, p_p-Pf); v = np.maximum(0, Pf-p_p)
    dev = 0.5*pr*u*DT + 1.5*pr*v*DT
    deal = pr*np.minimum(p_p, Pf)*DT

```

```

emg = 5.0*pr*E*DT
settle = deal + dev + emg
cost = [pr*Pf*DT, use, pr*loss, pr*c2*DT, pr*d2*DT, emg]
for t in range(144):
    dv = float(dev[t])
    if abs(Pf[t]-p_p[t]) < 1e-9: dirn = '无'; nn += 1
    elif Pf[t] > p_p[t]: dirn = '多买'; nb += 1
    else: dirn = '少买'; ns += 1
    row = [str(date), lbl(t), round(float(pr[t]), 6), SRC,
           round(float(load[d][t]), 6), round(float(pv[d][t]), 6), round(float(netd[t]), 6),
           round(float(fcst[t]), 6),
           round(float(p_p[t]*DT), 6), round(float(Pf[t]*DT), 6),
           round(float(c2[t]*DT), 6), round(float(d2[t]*DT), 6), round(float(E[t]*DT), 6),
           round(float(SS[t+1]), 6)] + [round(float(x[t]), 6) for x in cost] + \
           [round(dv, 6), dirn, round(float(deal[t]), 6), round(float(settle[t]), 6)]
    ws.append(row)
    acc += np.array([row[14], row[15], row[16], row[17], row[18], row[19], row[20], row[22], row[23]] + [0]*6)
ws.append(['合计', '', '', '', '', '', '', '', '', '', '', '', '', '', '' +
           [round(float(x), 2) for x in acc[:6]] +
           [round(float(acc[6]), 2), '多买 %d / 少买 %d / 未改 %d 段' % (nb, ns, nn),
           round(float(acc[7]), 2), round(float(acc[8]), 2)])])
for i, w in enumerate([12,13,13,12,14,16,13,13,15,15,12,12,15,16,16,16,13,13,13,15,13,24,15,17], 1):
    ws.column_dimensions[openpyxl.utils.get_column_letter(i)].width = w
ws.freeze_panes = 'C2'
os.makedirs(os.path.dirname(os.path.abspath(OUTP)), exist_ok=True)
wb.save(OUTP)
M = json.load(open(os.path.join(os.path.dirname(SOLP), 'metrics.json'), encoding='utf-8'))['metrics']
print('已写出:', OUTP)
print(' 行数 = %d (334 天 x 144 段 + 表头 + 合计)' % (ws.max_row))
print(' 偏差费用合计 = %.2f (指标 q3_adjustment_cost = %.2f)' % (acc[6], M['q3_adjustment_cost']))
print(' 紧急购电费合计 = %.2f (指标 q3_emergency_cost = %.2f)' % (acc[5], M['q3_emergency_cost']))
print(' 本时段结算费用合计 = %.2f (指标 q3_total_cost = %.2f)' % (acc[8], M['q3_total_cost']))
print(' 成交购电费合计 = %.2f (指标 q3_purchase_cost = %.2f)' % (acc[7], M['q3_purchase_cost']))
ok = (abs(acc[6]-M['q3_adjustment_cost']) < 1 and abs(acc[5]-M['q3_emergency_cost']) < 1
      and abs(acc[8]-M['q3_total_cost']) < 1 and abs(acc[7]-M['q3_purchase_cost']) < 1)
print(' 对账: %s' % ('全部一致' if ok else '★不一致'))

```

D.2.84 支撑材料/代码/07_交付物生成/q3_fc_write_result3.py

-*- coding: utf-8 -*-

"""问题三正式交付物 result3.xlsx 生成器（正式口径 E3v2：四版决策 + 扩展窗 σ 逆方差加权）。

严格按附件 5 模板：

- 表头逐字保留（模板表头相对数据右移一格，数据自第 2 列起按时间顺序填 — 见 verify_constraints.py:109）
- 计划购电量 / 调整购电量：A=日期，B..(B+143)=各 10 分钟段电量(kwh)，146=全天购电量，147=全天购电费
- 充放电量：每天 6 行(0:00-4:00 ... 20:00-24:00)，首行记 00:00 与初始储电量，次行记 24:00 与结束储电量
- 紧急购电量：按连续时段压缩，无紧急购电的日期整行省略

用法: python q3_fc_write_result3.py <solution.json> <out.xlsx> [模板路径]

```

"""
import sys, io, os, json, datetime, shutil
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
import openpyxl

```

```

SOLP = sys.argv[1]; OUT = sys.argv[2]
TPL = sys.argv[3] if len(sys.argv) > 3 else 'problem_files/附件/附件 5/result3.xlsx'
ROOT = os.path.abspath('.')
INP = json.load(open(os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q3_input.json'), encoding='utf-8'))
PRICE = INP['price']; DT = float(INP['dt']); T = 144
SOL = json.load(open(SOLP, encoding='utf-8'))
DAYS = SOL['days']
D0 = datetime.date(2025, 1, 1)
PERIODS = ['0:00-4:00', '4:00-8:00', '8:00-12:00', '12:00-16:00', '16:00-20:00', '20:00-24:00']

```

```

def r4(x): return round(float(x), 4)
def r2(x): return round(float(x), 2)
def hhmm(slot):
    m = slot * 10
    return '%d:%02d' % (m // 60, m % 60)

wb = openpyxl.load_workbook(TPL)
for sn in ('计划购电量', '调整购电量', '充放电量', '紧急购电量'):
    assert sn in wb.sheetnames, sn

w1, w2 = wb['计划购电量'], wb['调整购电量']
w3, w4 = wb['充放电量'], wb['紧急购电量']
for ws in (w3, w4):
    if ws.max_row > 1:
        ws.delete_rows(2, ws.max_row)

sumP = 0.0; sumA_cost = 0.0; sumEm = 0.0; sumC = 0.0; sumD = 0.0
for i, dobj in enumerate(DAYS):
    row = i + 2
    date = D0 + datetime.timedelta(days=int(dobj))
    pp = SOL['plan0'][i]; pa = SOL['plan'][i]
    c = SOL['c'][i]; d = SOL['d'][i]; E = SOL['E'][i]
    w1.cell(row=row, column=1, value=datetime.datetime(date.year, date.month, date.day))
    w2.cell(row=row, column=1, value=datetime.datetime(date.year, date.month, date.day))
    for t in range(T):
        w1.cell(row=row, column=2 + t, value=r4(pp[t] * DT))
        w2.cell(row=row, column=2 + t, value=r4(pa[t] * DT))
    w1.cell(row=row, column=2 + T, value=r4(sum(pp) * DT))
    w1.cell(row=row, column=3 + T, value=r2(sum(PRICE[t] * pp[t] * DT for t in range(T))))
    base = sum(PRICE[t] * min(pp[t], pa[t]) for t in range(T)) * DT
    dev = sum(0.5 * PRICE[t] * max(0.0, pp[t] - pa[t]) + 1.5 * PRICE[t] * max(0.0, pa[t] - pp[t])
              for t in range(T)) * DT
    w2.cell(row=row, column=2 + T, value=r4(sum(pa) * DT))
    w2.cell(row=row, column=3 + T, value=r2(base + dev))
    sumP += sum(pp) * DT; sumA_cost += base + dev
# 充放电量
for pi, lbl in enumerate(PERIODS):
    lo, hi = pi * 24, (pi + 1) * 24
    rr = (row - 2) * 6 + 2 + pi
    w3.cell(row=rr, column=1, value=datetime.datetime(date.year, date.month, date.day) if pi == 0 else None)
    w3.cell(row=rr, column=2, value=lbl)
    w3.cell(row=rr, column=3, value=r4(sum(c[lo:hi]) * DT))
    w3.cell(row=rr, column=4, value=r4(sum(d[lo:hi]) * DT))
    if pi == 0:
        w3.cell(row=rr, column=5, value=datetime.time(0, 0))
        w3.cell(row=rr, column=6, value=r4(SOL['S0'][i]))
    elif pi == 1:
        w3.cell(row=rr, column=5, value='24:00')
        w3.cell(row=rr, column=6, value=r4(SOL['S1'][i]))
    sumC += sum(c) * DT; sumD += sum(d) * DT

```

紧急购电量（压缩连续时段；无紧急则该日不写行）

```
er = 2
for i, dobj in enumerate(DAYS):
    date = D0 + datetime.timedelta(days=int(dobj))
    E = SOL['E'][i]; j = 0; runs = []
    while j < T:
        if E[j] > 1e-6:
            k = j
            while k < T and E[k] > 1e-6:
                k += 1
            runs.append((j, k, sum(E[j:k]) * DT)); j = k
        else:
            j += 1
    for ri, (a, b, en) in enumerate(runs):
        w4.cell(row=er, column=1, value=datetime.datetime(date.year, date.month, date.day) if ri == 0 else None)
        w4.cell(row=er, column=2, value='%s-%s' % (hhmm(a), hhmm(b)))
        w4.cell(row=er, column=3, value=r4(en))
        er += 1; sumEm += en

os.makedirs(os.path.dirname(os.path.abspath(OUT)), exist_ok=True)
wb.save(OUT)
print('已写出:', OUT)
print(' 天数 = %d' % (s .. %s)' % (len(DAYS), D0 + datetime.timedelta(days=int(DAYS[0])),
    D0 + datetime.timedelta(days=int(DAYS[-1]))))

print(' 计划购电量合计 = %.4f kWh' % sumP)
print(' 调整表全天购电费合计 = %.2f 元 (应 = 成交 + 偏差)' % sumA_cost)
print(' 紧急购电量合计 = %.4f kWh' % sumEm)
print(' 充电 %.4f / 放电 %.4f kWh' % (sumC, sumD))
print(' 紧急购电表行数 = %d' % (er - 2))
```

D.2.85 支撑材料/代码/07_交付物生成/q42_decomp_check.py

-*- coding: utf-8 -*-

```
"""补充分解口径对照：逐日贪婪链（固定当日起点、终点自由） vs 全窗口耦合最优。"""
import os, json, sys
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__)); ROOT = os.path.dirname(HERE)
sys.path.insert(0, HERE)
from q42_lower_bound import build, solve_lp, unpack

inp = json.load(open(os.path.join(ROOT, "data_cleaned", "q4_input.json"), encoding="utf-8"))
lam = np.asarray(inp["price_actual"], float); Load = np.asarray(inp["load_actual"], float)
Pv = np.asarray(inp["pv_actual"], float); st = inp["storage"]; dt = float(inp["dt"])
eta = float(st["efficiency"]); Pmax = float(st["power_max"])
Smin = float(st["soc_min"]); Smax = float(st["soc_max"]); S0 = float(st["soc_initial"])
D, T = Load.shape; N = Load - Pv

def greedy(days, s_start):
    s = s_start; tot = 0.0
    for d in days:
        a = build(T, dt, eta, Pmax, Smin, Smax, N[d], Load[d], lam[d], s0=s)
        r = unpack(solve_lp(a).x, T, lam[d], dt, eta, N[d], Load[d], a["idx"])
        tot += r["cost"]; s = float(r["S"][-1])
    return tot, s

g, s_end = greedy(range(31, D), S0)
print(f"逐日贪婪链 (2/1 起点=6000, 终点自由) = {g:,.2f} 元, 末态 {s_end:.1f} kWh")
g2, s2 = greedy(range(31, D), Smax)
print(f"逐日贪婪链 (2/1 起点=10800, 终点自由) = {g2:,.2f} 元, 末态 {s2:.1f} kWh")
gj, sj = greedy(range(0, 31), S0)
```

```

print(f"1 月 (1/1 起点 6000, 终点自由) 最优费用 = {gj:,.2f} 元, 末态 {sj:.1f} kWh")
print(f"检验 D1 - 1 月 = {14494771.228646498 - gj:,.2f} (应 = C3 12,780,883.29)")
res = dict(greedy_from6000=g, greedy_from10800=g2, january_opt=gj,
           jan_end_soc=sj, check=14494771.228646498 - gj)
json.dump(res, open(os.path.join(ROOT, "experiments", "q4-2-lower-bound",
                                "decomposition_check.json"), "w", encoding="utf-8"),
           ensure_ascii=False, indent=1)

```

D.2.86 支撑材料/代码/07_交付物生成/q42_delivered_check.py

-*- coding: utf-8 -*-

```

"""复核所有已交付 result4-2.xlsx 的费用，并与理论下界比较（修正量纲与日期续行）。"""
import os, json, datetime, glob
import numpy as np, openpyxl

HERE = os.path.dirname(os.path.abspath(__file__)); ROOT = os.path.dirname(HERE)
BASE = os.path.dirname(ROOT)
inp = json.load(open(os.path.join(ROOT, "data_cleaned", "q4_input.json"), encoding="utf-8"))
lam = np.asarray(inp["price_actual"], float)
LB = json.load(open(os.path.join(ROOT, "experiments", "q4-2-lower-bound", "metrics.json"),
                    encoding="utf-8"))

D = lam.shape[0]
D0 = datetime.date(2025, 1, 1)

def tseg(s):
    a = s.split("-")[0].strip(); h, m = a.split(":")
    return (int(h) * 60 + int(m)) // 10

lb_c1 = LB["A_full_physics"]["C1_startFree_endFree"]["cost"]
lb_c4 = LB["A_full_physics"]["C4_start6000_end6000"]["cost"]
ns = LB["baseline"]["no_storage_window_cost"]

def check(fp):
    wb = openpyxl.load_workbook(fp, data_only=True)
    ws = wb["计划购电量"]
    plan = np.zeros((D, 144)); fee_file = 0.0; cur = None
    for r in ws.iter_rows(min_row=2, values_only=True):
        if r[0] is not None: cur = (r[0].date() - D0).days
        if cur is None or not (0 <= cur < D): continue
        plan[cur] = [float(x) if x is not None else 0.0 for x in r[1:145]]
        if cur >= 31 and r[146] is not None: fee_file += float(r[146])
    plan_cost = float((lam[31:, :] * plan[31:, :]).sum())
    ws3 = wb["紧急购电量"]; em = np.zeros((D, 144)); ekwh = 0.0; cur = None
    for r in ws3.iter_rows(min_row=2, values_only=True):
        if r[0] is not None: cur = (r[0].date() - D0).days
        if r[2] in (None, ""): continue
        e = float(r[2]); ekwh += e
        if cur is None or not (0 <= cur < D) or r[1] is None: continue
        parts = [p for p in str(r[1]).split(";") if "-" in p]
        for part in parts:
            try: kk = tseg(part)
            except Exception: continue
            if 0 <= kk < 144: em[cur, kk] += e / len(parts)
    em_cost = float(5.0 * (lam[31:, :] * em[31:, :]).sum())
    tot = plan_cost + em_cost
    return dict(file=os.path.relpath(fp, BASE).replace("\\", "/"),
               plan_kwh=float(plan[31:].sum()), plan_cost=plan_cost,

```

```

        file_fee_total=fee_file, emergency_kwh=ekwh, emergency_cost=em_cost,
        total_cost=tot, lb_C1=lb_c1, lb_C4=lb_c4, no_storage=ns,
        gap_abs_C1=tot - lb_c1, gap_pct_C1=100.0*tot/lb_c1 - 100.0,
        gap_abs_C4=tot - lb_c4, gap_pct_C4=100.0*tot/lb_c4 - 100.0,
        saving_vs_nostorage=ns - tot)

files = sorted(set(glob.glob(os.path.join(BASE, "*", "result4-2.xlsx"), recursive=True)))
rows = []
for f in files:
    if "原题与原始附件" in f or "附件" in os.path.basename(os.path.dirname(f)):
        pass
    try:
        rows.append(check(f))
    except Exception as e:
        print("skip", f, e)
for r in rows:
    print(json.dumps(r, ensure_ascii=False))
json.dump(rows, open(os.path.join(ROOT, "experiments", "q4-2-lower-bound",
    "delivered_cost_check.json"), "w", encoding="utf-8"),
    ensure_ascii=False, indent=1)

```

D.2.87 支撑材料/代码/07_交付物生成/q42_lb_figures.py

-*- coding: utf-8 -*-

```

"""问题 4-2 理论下界：图表生成。"""
import os, json, datetime
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams["font.sans-serif"] = ["SimHei"]
rcParams["axes.unicode_minus"] = False
rcParams["figure.dpi"] = 160

HERE = os.path.dirname(os.path.abspath(__file__)); ROOT = os.path.dirname(HERE)
OUT = os.path.join(ROOT, "experiments", "q4-2-lower-bound")
FIG = os.path.join(ROOT, "figures"); os.makedirs(FIG, exist_ok=True)
M = json.load(open(os.path.join(OUT, "metrics.json"), encoding="utf-8"))
VER = json.load(open(os.path.join(OUT, "verification.json"), encoding="utf-8"))
DEL = json.load(open(os.path.join(OUT, "delivered_cost_check.json"), encoding="utf-8"))
inp = json.load(open(os.path.join(ROOT, "data_cleaned", "q4_input.json"), encoding="utf-8"))
lam = np.asarray(inp["price_actual"], float); Load = np.asarray(inp["load_actual"], float)
Pv = np.asarray(inp["pv_actual"], float); dt = float(inp["dt"]); D, T = Load.shape
N = Load - Pv; sf = 31

c1 = M["A_full_physics"]["C1_startFree_endFree"]["cost"]
c4 = M["A_full_physics"]["C4_start6000_end6000"]["cost"]
cyc = M["C_daily_cyclic"]["level_6000"]
ns = M["baseline"]["no_storage_window_cost"]
deliv = [d for d in DEL if d["file"].startswith("结果5/结果/")] [0] ["total_cost"]

```

----- 图 1：下界阶梯 -----

```

labels = ["无储能\n直接购电", "已交付方案\n(结果5/result4-2)", "日循环政策\n下界",
    "理论下界\n(年能量守恒)", "理论下界\n(端点自由)"]

vals = [ns, deliv, cyc, c4, c1]
colors = ["#b0b7c3", "#e08b6a", "#7fb0d6", "#3f7fbf", "#1f4e79"]
fig, ax = plt.subplots(figsize=(9.2, 4.6))
b = ax.bar(labels, vals, color=colors, width=0.62)
for r, v in zip(b, vals):
    ax.text(r.get_x()+r.get_width()/2, v*1.006, f"{v/1e4:,.2f} 万元",
        ha="center", va="bottom", fontsize=9.5)

ax.axhline(c1, ls="--", lw=1.1, color="#1f4e79")
ax.set_ylabel("2/1-12/31 购电总费用 (元)")

```

```
ax.set_title("问题 4-2 费用阶梯: 理论下界与已实现方案 (附件4 实际电价, 负载/光伏已知)")
ax.set_ylim(0, ns*1.16)
ax.grid(axis="y", alpha=0.28)
plt.tight_layout(); plt.savefig(os.path.join(FIG, "q42_lb_ladder.png")); plt.close()
```

----- 图 2：月度对比 -----

```
def monthly(cost_vec):
    out = []
    for mth in range(2, 13):
        d0 = (datetime.date(2025, mth, 1) - datetime.date(2025, 1, 1)).days
        d1 = (datetime.date(2025, mth % 12 + 1, 1) - datetime.date(2025, 1, 1)).days if mth < 12 else D
        out.append(float(cost_vec[d0:d1].sum()))
    return np.array(out)

base_m = monthly((lam*np.maximum(N, 0)).sum(axis=1))
z = np.load(os.path.join(OUT, "C1_startFree_endFree.npz"))
p_lb = z["p"].sum(axis=1) # 每日购电量 kWh
cost_lb = (p_lb * lam[sf:].mean(axis=1)) # 近似: 月度用实际逐段价
cost_lb_exact = (z["p"] * lam[sf:]).sum(axis=1)
lb_m = monthly(cost_lb_exact)
x = np.arange(2, 13); w = 0.4
fig, ax = plt.subplots(figsize=(9.6, 4.4))
ax.bar(x-w/2, base_m/1e4, w, label="无储能直接购电", color="#b0b7c3")
ax.bar(x+w/2, lb_m/1e4, w, label="理论下界 (完美信息最优)", color="#1f4e79")
ax.set_xticks(x); ax.set_xticklabels([f"{m}月" for m in x])
ax.set_ylabel("购电费 (万元)"); ax.grid(axis="y", alpha=0.28)
ax.set_title("问题 4-2 理论下界的月度分布 (2025 年 2-12 月)")
ax.legend()
plt.tight_layout(); plt.savefig(os.path.join(FIG, "q42_lb_monthly.png")); plt.close()
```

----- 图 3：代表日分解 -----

```
for day in (datetime.date(2025, 7, 15), datetime.date(2025, 1, 16)):
    d = (day - datetime.date(2025, 1, 1)).days
    if d < sf: continue
    i = d - sf
    tt = np.arange(T)*10/60.0
    fig, axs = plt.subplots(3, 1, figsize=(9.6, 8.2), sharex=True,
        gridspec_kw=dict(height_ratios=[1.15, 1.0, 1.0]))
    axs[0].plot(tt, lam[d], color="#c0392b", lw=1.1)
    axs[0].set_ylabel("电价\n(元/kWh)"); axs[0].grid(alpha=0.3)
    axs[0].set_title(f'{day} 理论下界方案 (完美信息最优) 的逐 10 分钟分解')
    axs[1].plot(tt, Load[d], label="小区负载", color="#2c3e50", lw=1.2)
    axs[1].plot(tt, Pv[d], label="光伏实际", color="#e67e22", lw=1.2)
    axs[1].plot(tt, z["p"][i]*6, label="购电功率", color="#1f4e79", lw=1.2)
    axs[1].plot(tt, z["c"][i]*6, label="充电功率", color="#27ae60", lw=1.0, ls="--")
    axs[1].plot(tt, -z["u"][i]*6, label="放电功率(负)", color="#8e44ad", lw=1.0, ls="--")
    axs[1].set_ylabel("功率 (kW)"); axs[1].legend(ncol=3, fontsize=8); axs[1].grid(alpha=0.3)
    axs[2].plot(tt, z["S"][i*T:(i+1)*T+1][:-1], color="#16a085", lw=1.4)
    axs[2].axhline(1200, ls=":", color="grey"); axs[2].axhline(10800, ls=":", color="grey")
    axs[2].set_ylabel("储电量 (kWh)"); axs[2].set_xlabel("时刻 (h)")
    axs[2].set_ylim(0, 12000); axs[2].grid(alpha=0.3)
    plt.tight_layout()
    plt.savefig(os.path.join(FIG, f"q42_lb_day_{day.isoformat()}.png")); plt.close()
```

```
print("figures ->", FIG)
print([f for f in os.listdir(FIG) if f.startswith("q42_lb")])
```

D.2.88 支撑材料/代码/07_交付物生成/q42_lb_verify.py

-*- coding: utf-8 -*-

"""问题 4-2 理论下界的独立验证: 换形式重算 + KKT 证书 + 物理重演 + 求解器交叉."""

```
import os, json, sys
import numpy as np
from scipy.optimize import linprog
from scipy.sparse import coo_matrix
```

```
HERE = os.path.dirname(os.path.abspath(__file__)); ROOT = os.path.dirname(HERE)
OUT = os.path.join(ROOT, "experiments", "q4-2-lower-bound")
inp = json.load(open(os.path.join(ROOT, "data_cleaned", "q4_input.json"), encoding="utf-8"))
lam = np.asarray(inp["price_actual"], float); Load = np.asarray(inp["load_actual"], float)
Pv = np.asarray(inp["pv_actual"], float); st = inp["storage"]; dt = float(inp["dt"])
eta = float(st["efficiency"]); Pmax = float(st["power_max"])
Smin = float(st["soc_min"]); Smax = float(st["soc_max"]); S0 = float(st["soc_initial"])
D, T = Load.shape; N = Load - Pv; sf = 31
NW, LW, IW = N[sf:].reshape(-1), Load[sf:].reshape(-1), lam[sf:].reshape(-1)
L_ = NW.size
rep = {}
```

----- 1. 换形式: 以 $q = p+u-c$ 为主变量 ($N \leq q \leq L$) , $p = q+c-u \geq 0$ -----

```
def build_q(T, dt, eta, Pmax, Smin, Smax, N, L, lam, s0=None, sT=None):
    n = 3*T + (T+1); iq, ic, iu, iS = 0, T, 2*T, 3*T
    c_obj = np.zeros(n); c_obj[iq:iq+T] = lam*dt; c_obj[ic:ic+T] = lam*dt; c_obj[iu:iu+T] = -lam*dt
    rows, cols, vals = [], [], []
    for t in range(T): # q <= L
        rows += [t]; cols += [iq+t]; vals += [1.0]
    for t in range(T): # -q <= -N
        rows += [T+t]; cols += [iq+t]; vals += [-1.0]
    for t in range(T): # -(q+c-u) <= 0 => p >= 0
        rows += [2*T+t, 2*T+t, 2*T+t]; cols += [iq+t, ic+t, iu+t]; vals += [-1.0, -1.0, 1.0]
    b_ub = np.concatenate([np.asarray(L, float), -np.asarray(N, float), np.zeros(T)])
    A_ub = coo_matrix((vals, (rows, cols)), shape=(3*T, n)).tocsr()
    rows, cols, vals = [], [], []
    for t in range(T):
        rows += [t, t, t]; cols += [iS+t+1, iS+t, ic+t, iu+t]
        vals += [1.0, -1.0, -eta*dt, dt/eta]
    A_eq = coo_matrix((vals, (rows, cols)), shape=(T, n)).tocsr()
    bounds = [(None, None)]*T + [(0, Pmax)]*T + [(0, Pmax)]*T
    for k in range(T+1):
        lo, hi = Smin, Smax
        if k == 0 and s0 is not None: lo = hi = float(s0)
        if k == T and sT is not None: lo = hi = float(sT)
        bounds.append((lo, hi))
    return c_obj, A_ub, b_ub, A_eq, np.zeros(T), bounds, (iq, ic, iu, iS)

for tag, kw in (('C1', {}), ('C4', dict(s0=S0, sT=S0))):
    a = build_q(L_, dt, eta, Pmax, Smin, Smax, NW, LW, IW, **kw)
    r = linprog(a[0], A_ub=a[1], b_ub=a[2], A_eq=a[3], b_eq=a[4], bounds=a[5], method="highs")
```

```

rep[f'reform_{tag}_cost'] = float(r.fun); rep[f'reform_{tag}_ok'] = bool(r.success)
print(f'换形式 q-变量 {tag} = {float(r.fun):.4f}')

```

----- 2. 求解器交叉: highs-ds / highs-ipm -----

```

sys.path.insert(0, HERE)
from q42_lower_bound import build, solve_lp, unpack
for m in ("highs-ds", "highs-ipm"):
    for tag, kw in (("C1", {}), ("C4", dict(s0=S0, sT=S0))):
        a = build(L_, dt, eta, Pmax, Smin, Smax, NW, LW, IW, **kw)
        r = solve_lp(a, method=m)
        rep[f'{m}_{tag}_cost'] = float(r.fun)
        print(f'[{m}] {tag} = {float(r.fun):.4f}')

```

----- 3. KKT / 对偶证书 (C1) -----

```

a = build(L_, dt, eta, Pmax, Smin, Smax, NW, LW, IW)
res = linprog(a["c"], A_ub=a["A_ub"], b_ub=a["b_ub"], A_eq=a["A_eq"], b_eq=a["b_eq"],
              bounds=a["bounds"], method="highs")

try:
    lam_ub = np.asarray(res.ineqlin.marginals, float)
    nu = np.asarray(res.eqlin.marginals, float)
    x = res.x
    rc = a["c"] - a["A_ub"].T @ lam_ub - a["A_eq"].T @ nu
    bound_viol = []
    for i, (lo, hi) in enumerate(a["bounds"]):
        if lo is not None and x[i] < lo - 1e-6: bound_viol.append((i, "lo", x[i], lo))
        if hi is not None and x[i] > hi + 1e-6: bound_viol.append((i, "hi", x[i], hi))
    ineq_viol = float(np.max(a["A_ub"] @ x - a["b_ub"]))
    eq_viol = float(np.max(np.abs(a["A_eq"] @ x - a["b_eq"])))
    mu = -lam_ub      # mu >= 0 为 <= 约束的标准乘子
    assert (mu >= -1e-12).all()
    dual_obj = float(a["b_ub"] @ lam_ub + a["b_eq"] @ nu
                     + sum((lo*max(0.0, rc[i]) if lo is not None else 0.0)
                           + (hi*min(0.0, rc[i]) if hi is not None else 0.0)
                           for i, (lo, hi) in enumerate(a["bounds"])))
    # 标准形式 min c'x, A_ub x <= b_ub, A_eq x = b_eq, l<=x<=u
    # => 对偶可行要求 lambda_ub <= 0 (否则 Lagrangian 无下界)
    # 独立构造若干个"确实可行"的点: 把目标函数加随机扰动后重解 LP
    # 对每个可行点 x', 用同一组对偶变量检查 L(x') <= c'x', 且 c'x' >= 原始最优值
    rng = np.random.default_rng(0); weak_ok = True; checked = 0; costs = []
    for k in range(5):
        ck = a["c"] + rng.normal(0, 1e-3, size=a["c"].size) * max(1.0, np.abs(a["c"]).max())
        rk = linprog(ck, A_ub=a["A_ub"], b_ub=a["b_ub"], A_eq=a["A_eq"], b_eq=a["b_eq"],
                     bounds=a["bounds"], method="highs")
        if not rk.success:
            continue
        xt = rk.x; Au = a["A_ub"] @ xt
        if (Au <= a["b_ub"] + 1e-7).all() and np.abs(a["A_eq"] @ xt - a["b_eq"]).max() < 1e-7:

```

```

checked += 1
cx = float(a["c"] @ xt)
costs.append(cx)
# 取  $\mu = -\lambda_{ub} \geq 0$ , 则  $L(x) = c'x + \mu'(A_{ub}x - b_{ub}) - \nu'(A_{eq}x - b_{eq}) \leq c'x$ 
Lv = float(cx + mu @ (Au - a["b_ub"]) - nu @ (a["A_eq"] @ xt - a["b_eq"]))
if Lv > cx + 1e-5 or cx < float(res.fun) - 1e-5:
    weak_ok = False
rep["kkt"] = dict(
    primal_obj=float(res.fun), dual_obj=dual_obj,
    duality_gap=abs(dual_obj - float(res.fun)),
    dual_feasible_mu_ge0=bool((mu >= -1e-12).all()),
    lam_ub_le0=bool((lam_ub <= 1e-9).all()),
    lam_ub_min=float(lam_ub.min()), lam_ub_max=float(lam_ub.max()),
    feasible_alt_points=checked, feasible_alt_costs=costs,
    feasible_alt_all_ge_optimum=bool(all(c >= float(res.fun) - 1e-5 for c in costs)),
    lagrangian_lower_bound_ok=bool(weak_ok),
    complementary_slack_ub=float(np.max(np.abs(lam_ub * (a["A_ub"] @ x - a["b_ub"])))),
    max_ineq_violation=ineq_viol, max_eq_violation=eq_viol,
    n_bound_violations=len(bound_viol), marginals_available=True)
print(f"[KKT] primal={float(res.fun):.4f} dual={dual_obj:.4f} gap={abs(dual_obj-float(res.fun)):.3e}")
print(f"[KKT] max_ineq_viol={ineq_viol:.2e} max_eq_viol={eq_viol:.2e} bound_viol={len(bound_viol)}")
except Exception as e:
    rep["kkt"] = {"marginals_available": False, "error": str(e)}
    print("[KKT] marginals unavailable:", e)

```

----- 4. 物理重演（把 LP 解按 SOC 动力学回放） -----

```

from numpy.testing import assert_allclose
for tag, kw in (("C1", {}), ("C4", dict(s0=S0, sT=S0))):
    a = build(L_, dt, eta, Pmax, Smin, Smax, NW, LW, IW, **kw)
    r = solve_lp(a); u_ = unpack(r.x, L_, IW, dt, eta, NW, LW, a["idx"])
    S = np.empty(L_+1); S[0] = u_["S"][0]
    for t in range(L_):
        S[t+1] = S[t] + eta*dt*u_["c"][t] - dt/eta*u_["u"][t]
    rep[f"replay_{tag}"] = dict(
        soc_dyn_max_err=float(np.max(np.abs(S - u_["S"]))),
        soc_min=float(S.min()), soc_max=float(S.max()),
        power_max=float(max(u_["c"].max(), u_["u"].max())),
        p_min=float(u_["p"].min()),
        curt_min=float(u_["curt"].min()),
        supply_margin_min=float((u_["p"] + u_["u"] - u_["c"] - NW).min()),
        nodump_max_slack=float((u_["u"] - u_["c"] - LW).max()),
        cost_recomputed=float((IW*u_["p"]).sum()*dt),
        cost_from_lp=float(r.fun),
        energy_balance_err=float(np.max(np.abs(u_["p"] + u_["u"] + u_["pv_used"] - LW - u_["c"]))))
    print(f"[重演 {tag}] SOC 动力学误差={rep[f'replay_{tag}']['soc_dyn_max_err']:.2e} "
          f"[SOC∈[{S.min():.0f},{S.max():.0f}] 弃光 min={u_['curt'].min():.2e} ")

```

```
f"供电裕度 min={rep[freplay_{tag}][supply_margin_min]:.2e}")
```

```
json.dump(rep, open(os.path.join(OUT, "verification.json"), "w", encoding="utf-8"),
          ensure_ascii=False, indent=1)
```

```
print("\n 写出:", os.path.join(OUT, "verification.json"))
```

D.2.89 支撑材料/代码/07_交付物生成/q42_lower_bound.py

-*- coding: utf-8 -*-

```
"""
问题 4-2 理论下界（完美信息离线最优）—— 主求解
=====
能量平衡（逐 10 分钟段 t, dt = 1/6 h）
    PV_used_t + p_t + u_t = L_t + c_t, 0 <= PV_used_t <= P_t
    => q_t := p_t + u_t - c_t 满足 N_t <= q_t <= L_t (N = L - P)
不可上网售电 => 弃光 curt_t = P_t - PV_used_t >= 0

4-2 计费（完美信息下计划=实际，紧急购电恒为 0：5λ 严格更贵且常规购电无功率上限）
    min sum_t lambda_t * p_t * dt

LP
    min sum_t lambda_t * p_t * dt
s.t. p_t - c_t + u_t >= N_t    (供电量不低于小区负载)
     u_t - c_t <= L_t        (不可向"空处"放电 => 弃光非负)
     S_{t+1} = S_t + eta*dt*c_t - (dt/eta)*u_t
     0 <= c_t, u_t <= Pmax, p_t >= 0, Smin <= S_t <= Smax

"""
import os, json, argparse, time
import numpy as np
from scipy.optimize import linprog
from scipy.sparse import coo_matrix

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(HERE)
DEFAULT_INPUT = os.path.join(ROOT, "data_cleaned", "q4_input.json")

def build(T, dt, eta, Pmax, Smin, Smax, N, L, lam, s0=None, sT=None,
         cost_mask=None, no_dump=True, no_power=False, eta_eff=None):
    n = 3 * T + (T + 1)
    ip, ic, iu, iS = 0, T, 2 * T, 3 * T
    e = eta if eta_eff is None else eta_eff
    pw = 1e9 if no_power else Pmax
    c_obj = np.zeros(n)
    cm = np.ones(T) if cost_mask is None else np.asarray(cost_mask, float)
    c_obj[ip:ip+T] = lam * dt * cm

    rows, cols, vals = [], [], []
    for t in range(T):
        rows += [t, t]; cols += [ip+t, ic+t, iu+t]; vals += [-1.0, 1.0, -1.0]
    b_ub = list(-np.asarray(N, float)); nrow = T
    if no_dump:
        for t in range(T):
            rows += [T+t, T+t]; cols += [iu+t, ic+t]; vals += [1.0, -1.0]
            b_ub += list(np.asarray(L, float)); nrow = 2*T
    A_ub = coo_matrix((vals, (rows, cols)), shape=(nrow, n)).tocsr()
```

```

rows, cols, vals = [], [], []
for t in range(T):
    rows += [t, t, t]
    cols += [iS+t+1, iS+t, ic+t, iu+t]
    vals += [1.0, -1.0, -e*dt, dt/e]
A_eq = coo_matrix((vals, (rows, cols)), shape=(T, n)).tocsr()

bounds = [(0, None)]*T + [(0, pw)]*T + [(0, pw)]*T
for k in range(T+1):
    lo, hi = Smin, Smax
    if k == 0 and s0 is not None: lo = hi = float(s0)
    if k == T and sT is not None: lo = hi = float(sT)
    bounds.append((lo, hi))
return dict(c=c_obj, A_ub=A_ub, b_ub=np.array(b_ub), A_eq=A_eq,
            b_eq=np.zeros(T), bounds=bounds, idx=(ip, ic, iu, iS),
            A_ub_nd=A_ub)

def solve_lp(a, method="highs"):
    res = linprog(a["c"], A_ub=a["A_ub"], b_ub=a["b_ub"], A_eq=a["A_eq"],
                  b_eq=a["b_eq"], bounds=a["bounds"], method=method)
    if not res.success:
        raise RuntimeError(res.message)
    return res

def unpack(x, T, lam, dt, eta, N, L, idx):
    ip, ic, iu, iS = idx
    p, c, u, S = x[ip:ip+T], x[ic:ic+T], x[iu:iu+T], x[iS:iS+T+1]
    pv_used = np.asarray(L) + c - p - u
    curt = np.asarray(np.asarray(L) - np.asarray(N)) - pv_used
    return dict(cost=float((lam*p).sum()*dt), p=p, c=c, u=u, S=S,
                curt=curt, pv_used=pv_used,
                p_kwh=float(p.sum()*dt), c_kwh=float(c.sum()*dt),
                u_kwh=float(u.sum()*dt), curt_kwh=float(curt.sum()*dt),
                loss_kwh=float((c.sum()-u.sum())*dt*(1-eta)))

def variant(label, N, L, lam, dt, eta, Pmax, Smin, Smax, **kw):
    T = N.size
    a = build(T, dt, eta, Pmax, Smin, Smax, N, L, lam, **kw)
    t0 = time.time(); res = solve_lp(a); rt = time.time()-t0
    r = unpack(res.x, T, lam, dt, eta, N, L, a["idx"]); r["runtime_s"] = rt
    r["primal_obj"] = float(res.fun)
    print(f" {label:38s} {r['cost']:>16,.2f} 元 ({rt:5.1f}s)")
    return r

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input", default=DEFAULT_INPUT)

```

```

ap.add_argument("--out", default=os.path.join(ROOT, "experiments", "q4-2-lower-bound"))
ap.add_argument("--score-from", type=int, default=31)
a = ap.parse_args(); os.makedirs(a.out, exist_ok=True)

inp = json.load(open(a.input, encoding="utf-8"))
lam = np.asarray(inp["price_actual"], float)
Load = np.asarray(inp["load_actual"], float)
Pv = np.asarray(inp["pv_actual"], float)
st = inp["storage"]; dt = float(inp["dt"])
eta = float(st["efficiency"]); Pmax = float(st["power_max"])
Smin = float(st["soc_min"]); Smax = float(st["soc_max"]); S0 = float(st["soc_initial"])
D, T = Load.shape; N = Load - Pv; sf = a.score_from
NW, LW, IW = N[sf:].reshape(-1), Load[sf:].reshape(-1), lam[sf:].reshape(-1)
out = {"config": dict(days=int(D), seg=int(T), dt=dt, eta=eta, Pmax=Pmax,
                    Smin=Smin, Smax=Smax, S_initial=S0, score_from=int(sf),
                    score_days=int(D-sf), window_segments=int(NW.size))}
out["baseline"] = {
    "no_storage_window_cost": float((lam[sf:]*np.maximum(N[sf:],0)).sum()*dt),
    "window_purchase_kwh": float(np.maximum(N[sf:],0).sum()*dt),
    "window_net_load_kwh": float(N[sf:].sum()*dt),
    "year_pv_surplus_kwh": float(np.maximum(-N,0).sum()*dt),
    "price_mean_window": float(lam[sf:].mean()),
    "price_min": float(lam[sf:].min()), "price_max": float(lam[sf:].max()),
}
print(f"[基线] 无储能 2/1-12/31 购电费 = {out['baseline']['no_storage_window_cost']:.2f} 元")

print("\n=== A. 完整物理约束 · 2/1-12/31 全窗口耦合 LP (理论下界主结果) ===")
A = {}
A["C1_startFree_endFree"] = variant("C1 起点自由 / 终点自由", NW, LW, IW, dt, eta, Pmax, Smin, Smax)
A["C2_startFree_end6000"] = variant("C2 起点自由 / 终点=6000", NW, LW, IW, dt, eta, Pmax, Smin, Smax, sT=S0)
A["C3_start6000_endFree"] = variant("C3 起点=6000 / 终点自由", NW, LW, IW, dt, eta, Pmax, Smin, Smax, s0=S0)
A["C4_start6000_end6000"] = variant("C4 起点=6000 / 终点=6000 (年守恒)", NW, LW, IW, dt, eta, Pmax, Smin, Smax,
s0=S0, sT=S0)

print("\n=== B. 敏感性/放缩 (均为合法下界, 用于说明下界的稳健性) ===")
B = {}
B["noPowerLimit"] = variant("B1 去掉 5000kW 充放电功率上限", NW, LW, IW, dt, eta, Pmax, Smin, Smax,
no_power=True)
B["eta_1"] = variant("B2 无损耗 eta=1", NW, LW, IW, dt, eta, Pmax, Smin, Smax, eta_eff=1.0)
B["freeMidnightTeleport"] = None # 由逐日分解给出
# 逐日分解 (允许午夜储电量自由跳变) = 更松的下界
tot = 0.0
for d in range(sf, D):
    aa = build(T, dt, eta, Pmax, Smin, Smax, N[d], Load[d], lam[d])
    tot += unpack(solve_lp(aa).x, T, lam[d], dt, eta, N[d], Load[d], aa["idx"])["cost"]

```

```

B["freeMidnightTeleport"] = dict(cost=float(tot))
print(f' {'B3 允许午夜储电量自由跳变':38s} {tot:>16,.2f} 元")
B["noDumpRelax"] = variant("B4 去掉 u-c<=L (松弛) ", NW, LW, IW, dt, eta, Pmax, Smin, Smax, no_dump=False)
B = {k: (v["cost"] if isinstance(v, dict) and "cost" in v else v) for k, v in B.items()}

```

```

print("\n=== C. 逐日循环政策类 S(0)=S(24) (链式 => 午夜水平恒为常数) ===")

```

```

C = {}
def cyclic(level):
    tot = 0.0
    for d in range(sf, D):
        aa = build(T, dt, eta, Pmax, Smin, Smax, N[d], Load[d], lam[d],
                    s0=float(level), sT=float(level))
        tot += unpack(solve_lp(aa).x, T, lam[d], dt, eta, N[d], Load[d], aa["idx"])["cost"]
    return float(tot)
for lv in (S0,):
    C[f'level_{int(lv)}'] = cyclic(lv)
    print(f' {'逐日循环, 午夜水平=' + str(int(lv)):38s} {C[f'level_{int(lv)}']:>16,.2f} 元")
best = (None, np.inf)
for lv in np.linspace(Smin, Smax, 25):
    v = cyclic(lv)
    if v < best[1]: best = (float(lv), v)
C["freeLevel"] = dict(level=best[0], cost=best[1])
print(f' {'逐日循环, 水平自由':38s} {best[1]:>16,.2f} 元 (最优水平 {best[0]:.0f} kWh)")

```

```

print("\n=== D. 全年前置链 (1/1 0:00 = 6000 kWh) 交叉验证 ===")

```

```

NY, LY, IY = N.reshape(-1), Load.reshape(-1), lam.reshape(-1)
D_ = {}
rY = variant("D1 全年 1/1=6000, 12/31 自由", NY, LY, IY, dt, eta, Pmax, Smin, Smax, s0=S0)
D_["year_start6000_endFree_allYearCost"] = rY["cost"]
rY2 = variant("D2 全年 1/1=6000, 12/31=6000", NY, LY, IY, dt, eta, Pmax, Smin, Smax, s0=S0, sT=S0)
D_["year_start6000_end6000_allYearCost"] = rY2["cost"]

```

```

res = {"A_full_physics": {k: {"cost": v["cost"], "p_kwh": v["p_kwh"],
                             "c_kwh": v["c_kwh"], "u_kwh": v["u_kwh"],
                             "curt_kwh": v["curt_kwh"], "loss_kwh": v["loss_kwh"],
                             "S_start": float(v["S"][0]), "S_end": float(v["S"][-1]),
                             "S_min_path": float(v["S"].min()),
                             "S_max_path": float(v["S"].max()),
                             "runtime_s": v["runtime_s"]} for k, v in A.items()}},
       "B_relaxations": B, "C_daily_cyclic": C, "D_full_year": D_,
       "baseline": out["baseline"], "config": out["config"]}
json.dump(res, open(os.path.join(a.out, "metrics.json"), "w", encoding="utf-8"),
           ensure_ascii=False, indent=1)

```

```

for k, v in A.items():

```

```

np.savez_compressed(os.path.join(a.out, k + ".npz"),
                    p=v["p"].reshape(D-sf, T), c=v["c"].reshape(D-sf, T),
                    u=v["u"].reshape(D-sf, T), S=v["S"], curt=v["curt"].reshape(D-sf, T),
                    eta=np.array([eta]), dt=np.array([dt]))
np.savez_compressed(os.path.join(a.out, "D1_year.npz"),
                    p=rY["p"].reshape(D, T), c=rY["c"].reshape(D, T),
                    u=rY["u"].reshape(D, T), S=rY["S"], curt=rY["curt"].reshape(D, T))
print("\n[主结果明细 C1] S_start=%0.1f S_end=%0.1f 购电=%0.0f 充电=%0.0f 放电=%0.0f 弃光=%0.0f kWh"
      % (res["A_full_physics"]["C1_startFree_endFree"]["S_start"],
         res["A_full_physics"]["C1_startFree_endFree"]["S_end"],
         A["C1_startFree_endFree"]["p_kwh"], A["C1_startFree_endFree"]["c_kwh"],
         A["C1_startFree_endFree"]["u_kwh"], A["C1_startFree_endFree"]["curt_kwh"]))
print("写出:", os.path.join(a.out, "metrics.json"))

if __name__ == "__main__":
    main()

```

D.2.90 支撑材料/代码/07_交付物生成/q42_year_check.py

-*- coding: utf-8 -*-

```

"""年度耦合解的内部一致性: D1 解中 1 月费用与 2/1 0:00 储电量."""
import os, json, sys
import numpy as np
HERE = os.path.dirname(os.path.abspath(__file__)); ROOT = os.path.dirname(HERE)
sys.path.insert(0, HERE)
from q42_lower_bound import build, solve_lp, unpack

inp = json.load(open(os.path.join(ROOT, "data_cleaned", "q4_input.json"), encoding="utf-8"))
lam = np.asarray(inp["price_actual"], float); Load = np.asarray(inp["load_actual"], float)
Pv = np.asarray(inp["pv_actual"], float); st = inp["storage"]; dt = float(inp["dt"])
eta = float(st["efficiency"]); Pmax = float(st["power_max"])
Smin, Smax, S0 = float(st["soc_min"]), float(st["soc_max"]), float(st["soc_initial"])
D, T = Load.shape; N = Load - Pv
OUT = os.path.join(ROOT, "experiments", "q4-2-lower-bound")

z = np.load(os.path.join(OUT, "D1_year.npz"))
pY, SY = z["p"], z["S"]
jan_cost = float((lam[:31] * pY[:31]).sum() * dt)
win_cost = float((lam[31:] * pY[31:]).sum() * dt)
print(f"D1 年度最优解: 1 月费用 = {jan_cost:,.2f} 窗口费用 = {win_cost:,.2f} 合计 = {jan_cost+win_cost:,.2f}")
print(f"D1 解中 2/1 0:00 储电量 S = {float(SY[31:T]):.2f} kWh")

```

1 月精确耦合最优（起点 6000，终点自由）

```

NY, LY, lY = N[:31].reshape(-1), Load[:31].reshape(-1), lam[:31].reshape(-1)
a = build(NY.size, dt, eta, Pmax, Smin, Smax, NY, LY, lY, s0=S0)
r = unpack(solve_lp(a).x, NY.size, lY, dt, eta, NY, LY, a["idx"])
print(f"1 月精确耦合最优（起点 6000，终点自由） = {r['cost']:,.2f} 元，末态 S(2/1) = {float(r['S'][-1]):.1f}")

```

窗口在 S(2/1)=1200 起点的最优（用于说明端点影响）

```

NW, LW, lW = N[31:].reshape(-1), Load[31:].reshape(-1), lam[31:].reshape(-1)
for s0v in (1200.0, 4000.0, 6000.0, 10800.0):
    aa = build(NW.size, dt, eta, Pmax, Smin, Smax, NW, LW, lW, s0=s0v)
    rr = unpack(solve_lp(aa).x, NW.size, lW, dt, eta, NW, LW, aa["idx"])
    print(f"窗口最优（S(2/1)={s0v:>7.1f}，终点自由） = {rr['cost']:,.2f} 元")

```

```

json.dump(dict(D1_jan_cost=jan_cost, D1_window_cost=win_cost,
              D1_S_at_feb1=float(SY[31:T]), jan_exact_opt=float(r["cost"])),

```

```

        jan_exact_end=float(r["S"][-1])),
    open(os.path.join(OUT, "year_consistency.json"), "w", encoding="utf-8"),
    ensure_ascii=False, indent=1)

```

D.2.91 支撑材料/代码/07_交付物生成/q4_write_result42.py

-*- coding: utf-8 -*-

"""问题 4-2 正式交付物 result4-2.xlsx 生成器（统一严格口径 + 可进化模型）。

严格按附件 5 模板：

- 表头逐字保留（模板表头相对数据右移一格，数据自第 2 列起按时间顺序填）
- 「计划购电量」：A=日期，B..(B+143)=各 10 分钟段计划购电量(kwh)，146=全天购电量，147=全天购电费
- 「充放电电量」：每天 6 行(0:00-4:00 ... 20:00-24:00)，首行记 00:00 与日初储电量，次行记 24:00 与日末储电量
- 「紧急购电量」：按连续时段压缩，无紧急购电的日期整行省略

用法: python q4_write_result42.py <solution.json> <out.xlsx> [模板路径] [q4_input.json]

```

"""
import sys, io, os, json, datetime
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
import openpyxl

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
SOLP = sys.argv[1]
OUT = sys.argv[2]
TPL = sys.argv[3] if len(sys.argv) > 3 else os.path.join(ROOT, 'problem_files', '附件', '附件5', 'result4-2.xlsx')
INPP = sys.argv[4] if len(sys.argv) > 4 else os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json')

```

```

INP = json.load(open(INPP, encoding='utf-8'))
PRICE = INP['price_actual']
DT = float(INP['dt'])
T = 144
SOL = json.load(open(SOLP, encoding='utf-8'))
DAYS = SOL['days']
D0 = datetime.date(2025, 1, 1)
PERIODS = ['0:00-4:00', '4:00-8:00', '8:00-12:00', '12:00-16:00', '16:00-20:00', '20:00-24:00']

```

```

def r4(x): return round(float(x), 4)
def r2(x): return round(float(x), 2)
def hhmm(slot): return '%d:%02d' % (slot * 10 // 60, slot * 10 % 60)

```

```

wb = openpyxl.load_workbook(TPL)
for sn in ('计划购电量', '充放电电量', '紧急购电量'):
    assert sn in wb.sheetnames, (sn, wb.sheetnames)

w1, w3, w4 = wb['计划购电量'], wb['充放电电量'], wb['紧急购电量']
for ws in (w1, w3, w4):
    if ws.max_row > 1:
        ws.delete_rows(2, ws.max_row)

```

```

sumP = 0.0; sumCost = 0.0; sumEm = 0.0; sumC = 0.0; sumD = 0.0
for i, dobj in enumerate(DAYS):
    d = int(dobj)

    row = i + 2

    date = D0 + datetime.timedelta(days=d)

    pp = SOL['plan0'][i]
    c = SOL['c'][i]; dd = SOL['d'][i]; E = SOL['E'][i]
    pa = PRICE[d]

    w1.cell(row=row, column=1, value=datetime.datetime(date.year, date.month, date.day))

    for t in range(T):
        w1.cell(row=row, column=2 + t, value=r4(pp[t] * DT))

    w1.cell(row=row, column=2 + T, value=r4(sum(pp) * DT))

    cost = sum(pa[t] * pp[t] * DT for t in range(T))
    w1.cell(row=row, column=3 + T, value=r2(cost))

    sumP += sum(pp) * DT; sumCost += cost

```

```

for pi, lbl in enumerate(PERIODS):
    lo, hi = pi * 24, (pi + 1) * 24
    rr = (row - 2) * 6 + 2 + pi
    w3.cell(row=rr, column=1, value=datetime.datetime(date.year, date.month, date.day) if pi == 0 else None)
    w3.cell(row=rr, column=2, value=lbl)
    w3.cell(row=rr, column=3, value=r4(sum(c[lo:hi]) * DT))
    w3.cell(row=rr, column=4, value=r4(sum(dd[lo:hi]) * DT))
    if pi == 0:
        w3.cell(row=rr, column=5, value=datetime.time(0, 0))
        w3.cell(row=rr, column=6, value=r4(SOL['S0'][i]))
    elif pi == 1:
        w3.cell(row=rr, column=5, value='24:00')
        w3.cell(row=rr, column=6, value=r4(SOL['S1'][i]))
    sumC += sum(c) * DT; sumD += sum(dd) * DT

er = 2
for i, dobj in enumerate(DAYS):
    d = int(dobj)
    date = D0 + datetime.timedelta(days=d)
    E = SOL['E'][i]; j = 0; runs = []
    while j < T:
        if E[j] > 1e-6:
            k = j
            while k < T and E[k] > 1e-6:
                k += 1
            runs.append((j, k, sum(E[j:k]) * DT)); j = k
        else:
            j += 1
    for ri, (a, b, en) in enumerate(runs):
        w4.cell(row=er, column=1, value=datetime.datetime(date.year, date.month, date.day) if ri == 0 else None)
        w4.cell(row=er, column=2, value='%s-%s' % (hhmm(a), hhmm(b)))
        w4.cell(row=er, column=3, value=r4(en))
        er += 1; sumEm += en

os.makedirs(os.path.dirname(os.path.abspath(OUT)), exist_ok=True)
wb.save(OUT)
print('已写出:', OUT)
print('  天数 = %d  (%s .. %s)' % (len(DAYS), D0 + datetime.timedelta(days=int(DAYS[0])),
    D0 + datetime.timedelta(days=int(DAYS[-1]))))

print('  计划购电量合计 = %.4f kWh' % sumP)
print('  全天购电费合计 = %.2f 元 (按真实电价)' % sumCost)
print('  紧急购电量合计 = %.4f kWh, 紧急购电表行数 = %d' % (sumEm, er - 2))
print('  充电 %.4f / 放电 %.4f kWh, 往返比 %.5f' % (sumC, sumD, sumD / sumC if sumC else 0))

```

D.2.92 支撑材料/代码/07_交付物生成/q4_write_result43.py

-*- coding: utf-8 -*-

"""问题 4-3 正式交付物 result4-3.xlsx 生成器（统一严格口径 + 可进化模型）。

按附件 5 模板四个工作表：计划购电量 / 调整购电量 / 充放电电量 / 紧急购电量。

- 表头逐字保留（模板表头相对数据右移一格）
- 计划/调整购电量：A=日期，B..(B+143)=各 10 分钟段电量(kWh)，146=全天购电量，147=全天费用
- 充放电电量：每天 6 行，首行记 00:00 与日初储电量，次行记 24:00 与日末储电量
- 紧急购电量：按连续时段压缩，无紧急则该日不写行

用法: python q4_write_result43.py <solution.json> <out.xlsx> [模板] [q4_input.json]

```

"""
import sys, io, os, json, datetime
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace')
import openpyxl

HERE = os.path.dirname(os.path.abspath(__file__))
ROOT = os.path.dirname(os.path.dirname(HERE))
SOLP = sys.argv[1]; OUT = sys.argv[2]
TPL = sys.argv[3] if len(sys.argv) > 3 else os.path.join(ROOT, 'problem_files', '附件', '附件5', 'result4-3.xlsx')
INPP = sys.argv[4] if len(sys.argv) > 4 else os.path.join(ROOT, 'paper_output_pro', 'data_cleaned', 'q4_input.json')

INP = json.load(open(INPP, encoding='utf-8'))
PRICE = INP['price_actual']; DT = float(INP['dt']); T = 144
SOL = json.load(open(SOLP, encoding='utf-8'))
DAYS = SOL['days']
D0 = datetime.date(2025, 1, 1)
PERIODS = ['0:00-4:00', '4:00-8:00', '8:00-12:00', '12:00-16:00', '16:00-20:00', '20:00-24:00']

def r4(x): return round(float(x), 4)
def r2(x): return round(float(x), 2)
def hhmm(slot): return '%d:%02d' % (slot * 10 // 60, slot * 10 % 60)

wb = openpyxl.load_workbook(TPL)
for sn in ('计划购电量', '调整购电量', '充放电量', '紧急购电量'):
    assert sn in wb.sheetnames, (sn, wb.sheetnames)

w1, w2, w3, w4 = wb['计划购电量'], wb['调整购电量'], wb['充放电量'], wb['紧急购电量']
for ws in (w1, w2, w3, w4):
    if ws.max_row > 1:
        ws.delete_rows(2, ws.max_row)

sP = sA = sCost = sEm = sC = sD = 0.0
for i, dobj in enumerate(DAYS):
    d = int(dobj); row = i + 2

    date = D0 + datetime.timedelta(days=d)

    pp = SOL['plan0'][i]; pa = SOL['plan'][i]
    c = SOL['c'][i]; dd = SOL['d'][i]; E = SOL['E'][i]
    lam = PRICE[d]

    for w in (w1, w2):
        w.cell(row=row, column=1, value=datetime.datetime(date.year, date.month, date.day))

    for t in range(T):
        w1.cell(row=row, column=2 + t, value=r4(pp[t] * DT))
        w2.cell(row=row, column=2 + t, value=r4(pa[t] * DT))
        w1.cell(row=row, column=2 + T, value=r4(sum(pp) * DT))
        w1.cell(row=row, column=3 + T, value=r2(sum(lam[t] * pp[t] * DT for t in range(T))))
        base = sum(lam[t] * min(pp[t], pa[t]) for t in range(T)) * DT
        dev = sum(0.5 * lam[t] * max(0.0, pp[t] - pa[t]) + 1.5 * lam[t] * max(0.0, pa[t] - pp[t])
                  for t in range(T)) * DT
        w2.cell(row=row, column=2 + T, value=r4(sum(pa) * DT))
        w2.cell(row=row, column=3 + T, value=r2(base + dev))
        sP += sum(pp) * DT; sA += sum(pa) * DT; sCost += base + dev

    for pi, lbl in enumerate(PERIODS):
        lo, hi = pi * 24, (pi + 1) * 24
        rr = (row - 2) * 6 + 2 + pi
        w3.cell(row=rr, column=1, value=datetime.datetime(date.year, date.month, date.day) if pi == 0 else None)
        w3.cell(row=rr, column=2, value=lbl)
        w3.cell(row=rr, column=3, value=r4(sum(c[lo:hi]) * DT))
        w3.cell(row=rr, column=4, value=r4(sum(dd[lo:hi]) * DT))
        if pi == 0:
            w3.cell(row=rr, column=5, value=datetime.time(0, 0))

```

```

        w3.cell(row=rr, column=6, value=r4(SOL['S0'][i]))
    elif pi == 1:
        w3.cell(row=rr, column=5, value='24:00')
        w3.cell(row=rr, column=6, value=r4(SOL['S1'][i]))
sC += sum(c) * DT; sD += sum(dd) * DT

er = 2
for i, dobj in enumerate(DAYS):
    d = int(dobj); date = D0 + datetime.timedelta(days=d)
    E = SOL['E'][i]; j = 0; runs = []
    while j < T:
        if E[j] > 1e-6:
            k = j
            while k < T and E[k] > 1e-6:
                k += 1
            runs.append((j, k, sum(E[j:k]) * DT)); j = k
        else:
            j += 1
    for ri, (a, b, en) in enumerate(runs):
        w4.cell(row=er, column=1, value=datetime.datetime(date.year, date.month, date.day) if ri == 0 else None)
        w4.cell(row=er, column=2, value='%s-%s' % (hhmm(a), hhmm(b)))
        w4.cell(row=er, column=3, value=r4(en))
        er += 1; sEm += en

os.makedirs(os.path.dirname(os.path.abspath(OUT)), exist_ok=True)
wb.save(OUT)
print('已写出:', OUT)
print(' 天数 = %d (%s .. %s)' % (len(DAYS), D0 + datetime.timedelta(days=int(DAYS[0])),
                                D0 + datetime.timedelta(days=int(DAYS[-1]))))

print(' 计划购电量合计 = %.4f kWh; 调整购电量合计 = %.4f kWh' % (sP, sA))
print(' 计划+调整表费用合计 = %.2f 元 (按附件4真实电价)' % sCost)
print(' 紧急购电量合计 = %.4f kWh, 紧急购电表行数 = %d' % (sEm, er - 2))
print(' 充电 %.4f / 放电 %.4f kWh, 往返比 %.5f' % (sC, sD, sD / sC if sC else 0))

```