基于多目标优化模型对农作物种植策略最优化

摘要

随着经济的发展，建设用地占用农业用地的现象愈发显著。对于乡村来说，如何利用有限的耕地资源创造更多的经济效益成为其农业生产活动中的关键性问题，这意味着劳动者需要合理分配有限的土地、水热、肥力等条件以获取最大化收益。基于上述背景，本文主要研究了农业种植策略的优化方案。，结合优化模型，采用粒子群算法，最后根据多约束条件下的优化算法，给出了在不同条件下农业种植的最优策略。

对于问题一，在假定各种农作物未来的预期销售量、种植成本、亩产量和销售价格相对于 2023 年保持稳定的情况下，我们以不同农作物的种植面积作为决策变量，建立了线性规划模型，目标是最大化总收益。模型的约束条件包含各种土地和大棚的种植作物限制，重茬要求，豆类的种植要求和方便田间作业管理的要求等。通过该模型的求解，得到了在以上条件下农作物的最优种植方案。结果表明，在最优种植方案下，通过合理分配作物后，总收益对比 2023 年提升了，并且农作物分布更加均匀，减少了成本和风险。

对于问题二，为进一步探讨未来市场不确定性因素（如预期销售量、价格、种植成本等）对种植策略的影响，我们运用了粒子群优化算法模型。模型综合考虑了农作物预期销售量、价格、亩产量和种植成本的波动，我们得到了一个最小化风险和最大化收益的种植方案。结果表明，在新增大量复杂情景的条件之后，优化后的方案。结果表明，在新增大量复杂情景的条件之后，相较于传统方案平均收益率下降 17.7%，优化后的方案只下降了 6.4%，展示出较强的抗风险能力。

对于问题三，我们得到了各个农作物之间的互补性与替代性，并计算得出预期销售量和价格、种植成本之间的相关性，构建了多目标优化模型。该模型考虑了作物之间的相互影响，如互补组合之间等协同效应和替代组合之间合种。在最佳种植方案中，组合显著增加了销售量，减少了作物种植成本，同时受益增加了。结果显示，该方案可以有效利用各作物之间的相互作用，提升农业生产率和经济效益。

关键字：线性规划　多目标优化模型　粒子群优化算法　农作物种植策略优化

一、 问题重述

1.1 问题背景

在农业生产过程中，劳动者往往需要根据不同耕地的水热及地形条件选择适宜的农作物，优化种植策略，降低生产成本以获取最大的经济效益。同时，合理的种植策略有利于保持土壤生产力，推动乡村经济的可持续发展。

1.2 问题要求

某乡村位于华北山区，地势条件复杂，有露天耕地 34 个大小不同的地块，合计 1201 亩，分别属于平旱地、梯田、水浇地以及山坡地，另有 16 个普通大棚以及 4 个智慧大棚，每个大棚面积为 0.6 亩。由于不同种耕地下水热及土壤条件不同，同种耕地往往只能耕种特定的作物（可见附件 1）。普通大棚每年可种植一季蔬菜与一季食用菌，智慧大棚每年都可以种植两季蔬菜，且同一地块每季可种植多种作物。在种植过程中，耕作者需注意以下条件: 首先, 同一地块下的同种作物不可连续种植。其次, 从 2023 年起，同一地块下的所有土地每三年需种植一季豆科作物以保持土壤生产力。最后，为了节约生产成本，作物种植地不可过于分散，作物种植面积也不宜太小。

问题 1 给出了 2023 年农作物的种植情况以及种植的相关数据，基于未来农产品预期销售量、种植成本、亩产量和销售价格与 2023 年相比保持稳定以及每季的产出都在当季销售的假设，分别让我们给出当（1）超出需求部分滞销.(2) 超出需求部分以 2023 年售价的 50% 的价格售出. 这两种情况下 2024-2030 年农作物种植的最优方案。

问题 2 要求制定农作物种植优化方案，考虑到小麦和玉米的未来预期销售量将以 5% 至 10% 的年平均增长率增长，而其他农作物的年预期销售量则可能在 2023 年的基础上波动 ±5%。同时，农作物的亩产量会受到气候等因素影响，每年可能波动 ±10%。此外，由于市场条件的变化，农作物的种植成本预计平均每年增长约 5%。在此背景下，需要设计一个综合考虑这些因素的农作物种植优化策略，以最大化经济效益和应对不确定性。

问题 3 引入了替代品与互补品的概念，要求在问题 2 的基础上需要考虑商品之间的相关性，进一步优化问题 2 对农作物种植方案的决策

二、 问题分析

2.1 问题 1 分析

问题 1 基于未来农产品预期销售量、种植成本、亩产量和销售价格与 2023 年相比保持稳定以及每季的产出都在当季销售的假设，要求我们给出在针对超产部分的两种不同处理方案下最优的农作物种植方案。对于耕作者来说，最优的种植方案应能使其利润达到最大化的同时实现可持续发展，即题干以及给出了种植过程的约束条件，即本题实际要求我们利用线性规划模型将问题转化为在“问题要求”中所述三种条件的约束下给出能够最大化其利润函数的方案。

2.2 问题 2 分析

问题 2 在问题 1 的基础上，原本稳定的预期销售量，种植成本，亩产量与销售价格都发生可量化的改变，在此条件下，考虑潜在的种植风险，再给出未来 2024-2030 年农作物种植策略的最优方案，其本质还是在题干的约束条件下最大化利润函数。

2.3 问题 3 分析

问题 3 在前两问的基础上，引入了经济学中互补品与替代品的概念，要求我们在决策过程中还得考虑商品属性之间的关系，变量进一步增多，某商品效率的变动会导致与其相关商品价格的变动，从而进一步影响其销量、种植面积、成本。最终要求我们在问题 2 的基础上加入商品属性的定义，分析其相关性并优化种植方案，与问题 2 的结果进行比较。

三、 模型假设

为简化问题，本文做出以下假设：

- 生产者在决策过程中总是追求利润最大化
- 土地、水源、劳动力和其他生产资源是有限的，并且其可获取性在规划期内是稳定的。
- 不同作物的生长特性是稳定不变的。
- 农业税包含在成本中
- 所有相同种类的农产品都属于替代品，非相同种类的农产品都属于互补品
- 这几年内不会发生自然灾害、人为事故等影响农业生产的公共安全事件

四、 符号说明

符号	说明	单位
p	利润	元
R	总销售额	元
C	成本	元
P	利润矩阵	元
E	预测面积矩阵	亩
B	单亩产出矩阵	元/亩
D	成本矩阵	元
$\vec{s_1}$	地块面积向量	亩
Y	亩产矩阵	kg/亩
$\vec{f_i}$	销售单价	元
$\vec{s}$	预期销售量	kg

五、 问题一的模型的建立和求解

5.1 模型建立

问题一要求我们给出最大化其利润的方案, 设 p 为最终利润，R 为总销售额，C 为成本，有利润函数

$$p = R - C$$

在本题中，利润函数可被表示为

$$P = EB^{\mathrm{T}} - ED^{\mathrm{T}} \tag{1}$$

其中，面积矩阵 E^t 表示 $t(t = 2023, 2024 \cdots, 2030)$ 年不同地块（不同季也算是不同地块）下不同种类作物的种植面积，如 2023 年作物种植的面积矩阵可表示为

$$E^{2023} = \begin{bmatrix} 0 & 0 & 0 & 72 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 68 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

矩阵中的元素 E_{ij} 中的 i 表示地块，j 则表示作物编号，如 E_{i1} 表示第 i 个地块下编号 1 作物（黄豆）的种植面积，$i=1$ 时表示 A_1 地块一季，$i=2$ 时表示 A_2 地块一季, 以此类推。[1]

单亩产出矩阵 B 中的元素 B_{ij} 表示在 i 地块下 j 编号作物的亩产出，成本矩阵 D 中的元素 D_{ij} 表示在 i 地块下 j 编号作物的成本。[2]

则问题一可处理为:

$$P=\sum_{t=2024}^{2030}(E^tB^{\mathrm{T}}-E^tD^{\mathrm{T}})$$

$$\textbf{maxmine}\quad tr(P)$$

$$\textbf{st.}\begin{cases}E_{ij}^t\geq 6\\ \forall j\in\{1,2,\cdots,41\},\sum_{i=1}^{90}E_{ij}^t\geq\vec{s_1}\\ \text{产量小于预期销售量}\\ \text{同一年内同一地块下},E_{ij}^t\neq E_{ij+1}^t\\ \text{相邻一年内，同一季下}E_{ij}^t\neq E_{ij}^{t+1}\end{cases}$$

其中，$P_{ii}(i\in\{1,2,\cdots,41\})$ 表示 i 编号地块的总利润（分一季与二季）,$\vec{s_i}$ 表示 i 编号地块的面积（分一季度与二季度）。

关于约束条件的说明:

- 约束 1：每个地块每种作物的种植面积不宜过小，查阅文献后发现，华北地区农作物单地块单作物种植最低种植面积为 5-10 亩之间，这里取 6 亩。
- 约束 2：对于所有地块来说，每种作物的种植面积需小于该地块总面积。
- 约束 3：产量小于预期销售量
- 约束 4: 作物不能重茬种植

5.2 模型求解

5.2.1 读取与清洗数据

读取有关作物习性，土地类型以及 2023 年农业生产相关数据，使用 pandas 库对数据进行清洗，包括去除空格、填充缺失值、统一大小写等，以确保数据的准确性和一致性，方便后续模型建构与求解。

[1]相同地块的不同季单独视为矩阵中的一列，如 E_{i27} 与 E_{i28} 分别表示 D1 地块第一季与第二季种植 i 编号作物的面积

[2]地块编号 i 的定义与上述相同

5.2.2 数据转换与扩展

将扩展数据转换为 Dataframe 类型，以将原始数据转换为适合模型输入的格式，随后根据不同种类地块的特性进行数据扩展，生成所有可能的种植组合。

5.2.3 定义决策变量

在对数据进行预处理后，需要将预处理后的数据进一步处理以方便计算。为了计算农产品总销售额与成本，先根据现有数据计算单亩产出矩阵 B(附录 A 图4), 再创建成本矩阵 D(附录 A 图5) 与地块面积向量 $\vec{s_1}$, 后建立预测的种植面积矩阵 E，矩阵 E 即为决策变量。

5.2.4 建立目标函数

若要求利润，需要先求得销售量。预测种植面积矩阵 E 与单位亩产矩阵 B^{T}，得到 41×41 方阵 P, 取对角线上所有元素，元素如 P_{11} 即表示小麦 7 年所有土地总收入, 则 $tr(P)$ 代表该乡村七年中种植所有作物的总收入。

需要留意的是，输出结果 P(代码中为 sum_1) 的排列并不是严格地以矩阵乘法的顺序，而是按 x 的下标排列:1，10，100,$\cdots$,2,20,$\cdots$。由于输出的仅仅是一年的数据，因此要用 矩阵串联 $\times 7$。矩阵 E(附录 A 图6)(代码中为 a) 不是简单水平堆叠，因为不能有 7 个 x_1，故更改变量名复制 7 次；B，D 是纯数据，直接垂直堆叠即可。

5.2.5 设置约束条件

- w 中所有非零元素的数量必须大于等于 6
- E 矩阵每列的和小于 s1 中对应的元素
- E 矩阵每行分段和小于 s1 的元素
- 相邻变量下标相同、字母相邻的变量限制
- 特定行列求和后大于 s1

5.3 求解结果

在超过部分滞销的情况下，输出结果为

类型_种植	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
长城单个	1170	3375	2970	2266	2531.25	2520	2700	2430	3420	3862.5	4000	4050	6300	2090	1662.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
产马单个	1300	3750	3300	2450	2981.25	2800	3000	2700	3780	3937.5	4400	4500	7150	2310	1837.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
巨人棚边	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28800	16200	23400	9000	18750	44000	18975	15750	22000	29250	25000	23000	105000	26250	14500	54000	27000	26400	0	0	0	0	0	0	0
巨人棚顶	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	287300	76000	160000	100000	0
巨人棚顶	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30720	17820	24960	9900	20250	47520	20700	16360	23760	31900	27000	24840	113400	28350	15660	59400	29160	28800	0	0	0	0	0	0	0
棉田单季	1235	3562.5	3135	2310	2666.25	2660	2850	2565	3600	3750	4200	4275	6825	2200	1750	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
长防单个	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
设施豆	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24000	13500	19500	7500	15000	33000	15525	12600	18150	24050	20500	18400	84000	21525	11600	45000	22500	22000	0	0	0	0	0	0	0
设施豆	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12500	10000	9750	0	0	0	0
巨人棚顶	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28800	16200	23400	9000	18750	44000	18975	15750	22000	29250	25000	23000	105000	26250	14500	54000	27000	26400	0	0	0	0	0	0	0

图 1　滞销情况下的生产策略

其中，列为农作物编号，行为不同种地块类型种植某作物的面积 (亩)

六、 问题二的模型的建立和求解

6.1 模型建立

问题 2 在问题 1 的基础上，各个稳定的量都有一定波动，其中各个变动的量如图所示:

初始量	说明	变动后
$\vec{s_i}(i=6,7)$	小麦，玉米销售量	$\vec{s_i^t}(1+a\%)^{(2030-t)}$
$\vec{s_i}(i\neq 6,7)$	除小麦，玉米的销售量	$\vec{s_i^t}(1+b\%)^{(2030-t)}$
Y_{ij}	i 地块下 j 作物的亩产	$\mathrm{Y}_{ij}^t = Y_{ij}^{t-1}(1\pm 10\%)$
D	成本矩阵	$\mathrm{D}_{ij}^t = D_{ij}^{t-1}(1+5\%)$
$\vec{f_i}(i\in[38,40])$	食用菌销价	$\vec{f_i^{t+1}} = \vec{f_i^t}(1-c\%)$
$\vec{f_i}(i\in[17,37])$	蔬菜销价	$\vec{f_i^{t+1}} = \vec{f_i^t}(1+5\%)$
$\vec{f_{41}}$	羊肚菌销价	$\vec{f_i^{t+1}} = \vec{f_i^t}(1+5\%)$

其中，$a\in[5,10], b\in[-5,5].c\in[1,5]$. 将变量表示出来后，即可创建目标函数

$$P=\sum_{t=2024}^{2030}(E^tB^{\mathrm{T}}-E^tD^{\mathrm{T}})$$

$$\textbf{maxmine}\quad tr(P)$$

其中,$B=Y\vec{f}$，表示波动状态下的单位亩产量。

6.2 模型求解

6.2.1 数据预处理

通过 pandas 库读取了两个 Excel 文件中的多个工作表，这些工作表包含了乡村的现有耕地信息、乡村种植的农作物信息、2023 年的农作物种植情况以及相关统计数据。移除了乡村种植的农作物中的特定行（41-44 行）。从 2023 年统计的相关数据中移除了几行（108-110 行）并进行了一系列的数据清洗，包括去除字符串中的空白字符、换行符等。清洗地块类型列，确保每个作物的种植耕地信息正确，并创建作物编号和种植耕地的对应关系。

为 2024 年至 2030 年生成了随机波动范围内的数据，包括小麦和玉米销售量增长率（5%-10%）、其他作物增长率为 ±5%、亩产量波动为 ±10%、种植成本每年增加 5%、粮食类价格稳定、蔬菜类价格年增长 5%、食用菌价格年下降 1%-5%。

6.2.2 定义决策变量

使用 model.addVars() 方法定义决策变量 $x[i,j,t]$，其中 i 代表地块编号，j 代表作物编号，t 代表季度（假设每年有两个种植季次）。这个变量表示地块 i 在季度 t 种植作物 j 的面积。

6.2.3 约束条件设立

- 对于每个地块和每个季度，种植的总面积不能超过该地块的可用面积。代码通过遍历所有地块和季节，并使用 model.addConstr() 方法添加约束来实现。
- 每三年种植一次豆类作物，以确保土地的轮作。这是通过检查年份和作物类型，并添加相应的约束来实现的。
- 同一地块不能连续季节种植相同作物，以减少病虫害等问题。
- 每块地种植每种作物的面积不得小于设定的最小面积（如 5 亩）。
- 不同的地块类型适合种植的作物类型不同，如平旱地、梯田、山坡地主要适合种植粮食类作物，水浇地适合水稻和蔬菜，普通大棚适合蔬菜和食用菌，智慧大棚主要适合蔬菜。
- 对于每种作物，所有地块和季度的总产量不能超过其预期销售量。

6.2.4 定义目标函数

目标函数是最大化总收入减去总成本，代码部分为 gp.quicksum(P[j] * Y[j] * x[i, j, t] - C[j] * x[i, j, t])，其中 P[j] 是作物 j 的销售单价，Y[j] 是作物 j 的亩产量，C[j] 是作物 j 的种植成本。

6.2.5 求解过程

由于数据量可能较大，代码采用了分批处理的方式，每次只处理一部分地块、年份和作物，以提高求解效率。使用 Gurobi 优化器创建模型，定义变量、约束条件和目标函数。调用 model.optimize() 方法求解模型。

如果模型求解成功（model.status == GRB.OPTIMAL），则将结果保存到 Excel 文件中。如果模型求解不可行（model.status == GRB.INFEASIBLE），则输出不可行的约束集到文件，以便分析原因。求解成功时，将每种作物在每块地每个季度的种植面积输出到 Excel 文件（result2.xlsx）。

6.3 求解结果

西红柿	茄子	菠菜	青椒	菜花	包菜	油麦菜	小青菜	黄瓜	生菜	辣椒	空心菜	黄心菜	芹菜	大白菜	白萝卜	红萝卜	榆黄菇	香菇	灵	肚	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3.00519435	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	6.295657054	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10.13724576	0	0	0	0	0	0	11.86275424	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.330831675	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.1	0	0.5	0	0	0	0	0	0	0	0	0	0	0
0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.237070681	0.362929319	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.429929064	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.35931309	0	0	0	0	0.24068691	0	0	0	0	0	0	0	0	0	0	0	0
0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.230831675	0.369168325	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.458047121	0	0	0	0	0	0.141952879	0	0	0	0	0	0	0	0	0	0	0
0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.125858637	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.145569108	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.32097644	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.32097644	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	13.16094242	0	0.839057582	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	18.83670154	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	16.74803663	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.572826333	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.566464738	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.32097644	0	0	0	0	0	0	0	0
0	0	0.32097644	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 2　波动下的最优化方案

七、 问题三的模型的建立和求解

7.1 模型建立

对于第三题，我们先给出互补品和替代品在经济学上的定义

定义: 互补品是指两种商品的消费是相互依赖的。当一种商品的消费增加时，另一种商品的消费也会增加。例如，咖啡和糖是互补品。

- **需求函数**: 假设商品 X 和商品 Y 是互补品，需求函数可以表示为:

$$Q_x = f(P_x, P_y, I)$$

  其中 Q_x 是商品 X 的需求量，P_x 和 P_y 分别是商品 X 和商品 Y 的价格，I 是消费者的收入。

- **交叉价格弹性**: 对于互补品，交叉价格弹性是负值:

$$E_{xy} = \frac{\partial Q_x}{\partial P_y} \cdot \frac{P_y}{Q_x} < 0$$

这意味着当商品 Y 的价格上升时，商品 X 的需求量下降。

定义: 替代品是指两种商品可以互相替代，消费者在选择时可以用一种商品替代另一种商品。例如，咖啡和茶是替代品。

- **需求函数**: 假设商品 A 和商品 B 是替代品，需求函数可以表示为:

$$Q_a = g(P_a, P_b, I)$$

其中 Q_a 是商品 A 的需求量，P_a 和 P_b 分别是商品 A 和商品 B 的价格，I 是消费者的收入。

- **交叉价格弹性**: 对于替代品，交叉价格弹性是正值:

$$E_{ab} = \frac{\partial Q_a}{\partial P_b} \cdot \frac{P_b}{Q_a} > 0$$

这意味着当商品 B 的价格上升时，商品 A 的需求量上升。

在本题中，若作物 j 与作物 k 是替代品的关系，则有

$$\forall \vec{f_j^t} < \vec{f_j^{t+1}},$$

$$E_{ik}^{t+1} = E_{ik}^t(1+a)(a \in \{5\% \sim 10\%\}),$$

$$D_{ik}^{t-1} = D_{ik}^t(1+5\%)$$

对于该乡村种植的农产品，所有相同种类的农产品都属于替代品。

根据互补性在经济学上的定义，若作物 j 与作物 k 存在互补关系，则有

$$\forall \vec{f_j^t} < \vec{f_j^{t+1}},$$

$$E_{ik}^{t+1} = E_{ik}^t(1-a)(a \in \{5\% \sim 10\%\}),$$

$$D_{ik}^{t+1} = D_{ik}^t(1-5\%)$$

$\vec{f_j}$ 表示 j 作物的销售单价。对于该乡村种植的农产品，所有不同种类的农产品都属于互补品。

对于预期销售量与销售价格、种植成本之间的相关性，应求 $E_{ij}^t, \vec{f_i}^t, D_{ij}^t$ 之间的协方差，得出协方差矩阵

$$\begin{pmatrix} Var(E_{ij}^t) & Cov(E_{ij}^t, \vec{f_i}^t) & Cov(E_{ij}^t, D_{ij}^t) \\ Cov(\vec{f_i}^t, E_{ij}^t) & Var(\vec{f_i}^t) & Cov(\vec{f_i}^t, D_{ij}^t) \\ Cov(D_{ij}^t, E_{ij}^t) & Cov(D_{ij}^t, \vec{f_i}^t) & Var(D_{ij}^t) \end{pmatrix}$$

之后计算三个变量之间的回归系数. 再根据问题二的模型求得最优种植策略。

7.2 模型求解

7.2.1 数据清洗和整合

使用 pandas 库读取 Excel 文件中的数据。清洗数据，包括删除不必要的行、处理缺失值、去除多余的空格和换行符等。创建作物编号和种植耕地的对应关系，并更新 df_sheet_2 中的种植耕地信息。

7.2.2 整合数据

将 df_sheet_3 中的作物信息拆分成多行,每行代表一个单独的作物信息。将 df_sheet_3 和 df_sheet_4 中的数据通过作物编号、种植季次和地块类型进行合并，以获取完整的作物种植和销售信息。

7.2.3 模拟未来农作物产量和销售

为不同的作物类型（如小麦、玉米、蔬菜、食用菌等）设定产量、成本和价格的随机波动范围。创建一个新的 DataFrame df_randomized，用于存储模拟后的数据。对每个年份，根据设定的波动参数，计算新的亩产量、种植成本和销售单价。根据作物编号和年份，计算每种作物的总产量和销售量。

7.2.4 优化作物种植布局

设定迭代次数、年份范围等参数。初始化总收益和最佳解决方案。在每次迭代中，对每个地块和年份，随机分配作物种植面积，确保满足地块类型适应性、作物类型多样性等约束条件。计算每种作物的总产量和总收益。根据作物之间的互补和替代关系，调整相关作物的种植面积和成本。

7.2.5 评估并更新最佳解决方案

如果当前迭代的总收益高于之前的最佳收益，则更新最佳解决方案。重复迭代过程，直到达到设定的迭代次数或满足其他停止条件。

7.3 求解结果

生成的协方差矩阵与回归系数如下：协方差矩阵：

$$[[8.86487022e+06 \quad -2.67707422e+03 \quad 3.50717897e+06]$$

$$[-2.67707422e+03 \quad 1.40754462e+02 \quad 9.03760527e+03]$$

$$[3.50717897e+06 \quad 9.03760527e+03 \quad 3.66334999e+06]]$$

回归系数:

$$[[2.26057313e-07 \quad 2.16201976e-05 \quad -2.69758082e-07]$$

$$[2.16201976e-05 \quad 1.05095457e-02 \quad -4.66259103e-05]$$

$$[-2.69758082e-07 \quad -4.66259103e-05 \quad 6.46259968e-07]]$$

- 粮食类作物的总种植面积增加;
- 粮食类、蔬菜类和食用菌类之中各农作物错季种植的比例显著增加;
- 总收益率增加了 25.6%。
- 有效增强该农村的抗风险能力，提高了农业生产率，推动农业可持续发展，具有相较于问题二更广阔的实际意义和指导意义

八、 模型的评价及改进空间

8.1 模型的优点

• 明确的目标函数: 模型以最大化利润为目标，通过利润函数清晰地表达了优化目标，便于理解和求解。

• 详细的数据表示: 引入了面积矩阵 E、单亩产出矩阵 B 和成本矩阵 D，这些矩阵详细地表示了不同年份、不同地块、不同作物的种植面积、产出和成本，为模型的精确计算提供了基础。

• 多年规划: 模型考虑了从 2024 年到 2030 年的多年规划，这有助于制定长期战略，平衡不同年份之间的资源分配，提高整体效益。

• 多样化的约束条件: 模型包含了一系列多样化的约束条件，如最小种植面积、地块总面积限制、产量与预期销售量的关系、作物不能重茬种植等，这些约束条件贴近实际农业生产情况，提高了模型的实用性和准确性。

• 地块和作物的细化处理: 模型将地块和作物进行了细化处理，不仅考虑了不同地块之间的差异，还考虑了同一地块不同季节之间的差异，这有助于更精确地模拟实际农业生产情况。

8.2 模型的缺点

• 参数敏感性: 模型的优化结果对参数（如单亩产出、成本等）的敏感性较高，如果参数估计不准确或发生较大变化，可能会导致优化结果的不稳定或失效。

• 非线性和复杂性: 虽然模型在形式上是线性的（基于矩阵运算），但考虑到实际农业生产中的诸多非线性因素和复杂性（如天气、病虫害、市场波动等），模型可能难以完全捕捉这些因素对利润的影响。

- **未考虑动态变化：**模型假设了每年之间的决策是独立的，但实际上农业生产中的许多因素（如土壤质量、作物轮作等）可能具有动态变化的特性，这些在模型中未得到充分考虑。
- **约束条件的简化：**虽然模型包含了一系列约束条件，但某些约束条件可能过于简化或不够精确（如最小种植面积的设定、产量与预期销售量的关系等），这可能导致模型结果与实际情况存在一定偏差。
- **求解难度：**由于模型涉及多个年份、多个地块和多种作物，且包含多个约束条件，因此求解过程可能较为复杂和耗时。

8.3 模型的改进及推广空间

8.3.1 模型的改进空间

尽管代码中提到了基于产量和销售单价的利润计算思路，但具体实现并不完整。可以进一步详细实现利润计算的模型，以提供更全面的经济评估。

目前的模型主要关注农作物自身的属性和种植条件，但未涉及市场供需关系。可以引入市场数据，如历史销售数据、消费者偏好变化等，以更准确地预测未来销售和价格。且代码中仅提到了亩产量波动为 ±10%，但未详细处理这种波动对模型的影响。可以建立更复杂的产量预测模型，考虑天气、病虫害等不确定性因素，并进行蒙特卡洛模拟等风险评估. 同样，可以引入市场风险分析，如价格波动风险、市场需求变化风险等，以提高模型的鲁棒性和实用性。

当前模型可能依赖于有限的数据源, 可以通过整合更多数据源（如政府统计数据、行业报告、市场调研等）来提高模型的准确性和适用性。考虑将模型与实时数据系统对接，实现数据的实时更新和模型的动态调整，以更好地应对市场变化。

代码中可以考虑引入更高效的优化算法（如遗传算法等）来提高优化效果。在某些情况下，可能需要同时优化多个目标（如利润最大化、风险最小化等）。可以引入多目标优化算法来求解这类问题。

8.3.2 模型的推广空间

当前模型针对华北地区该乡村的农作物种植情况。可以通过调整模型参数和数据源，将其推广至不同地域的农作物种植场景。该模型的基本框架和思路也可以应用于其他行业，如畜牧业、渔业等。通过调整相关参数和数据源，可以建立针对这些行业的经济评估模型。

除此之外，还可以在模型的基础上开发图形用户界面（GUI），使非专业用户也能轻松使用模型进行农作物种植的经济评估。自动生成详细的评估报告，包括图表、数据分析结果和决策建议等，便于用户理解和应用模型结果。在模型应用于实际场景时，收

集实时数据对模型进行反馈和调整，以不断优化模型性能。

参考文献

[1] 李光兵. 国外两种农户经济行为理论及其启示[J]. 农村经济与社会, 1992(06):52-57.
[2] 夏凡捷, 夏新念. 农产品价格波动时的数理微分模型分析 [J]. 统计与决策,2013,(14):58-61.DOI:10.13546/j.cnki.tjyjc.2013.14.003.

附录 A　生成数据

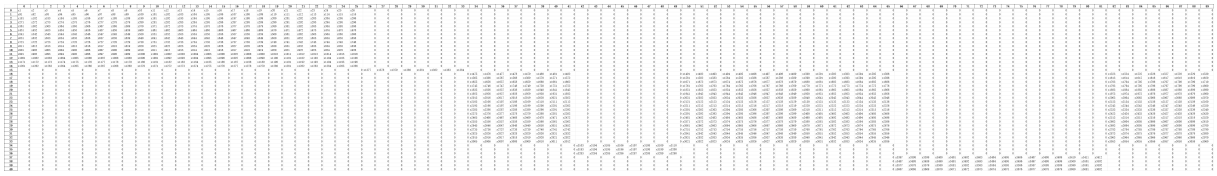

图 3　符号变量矩阵 A

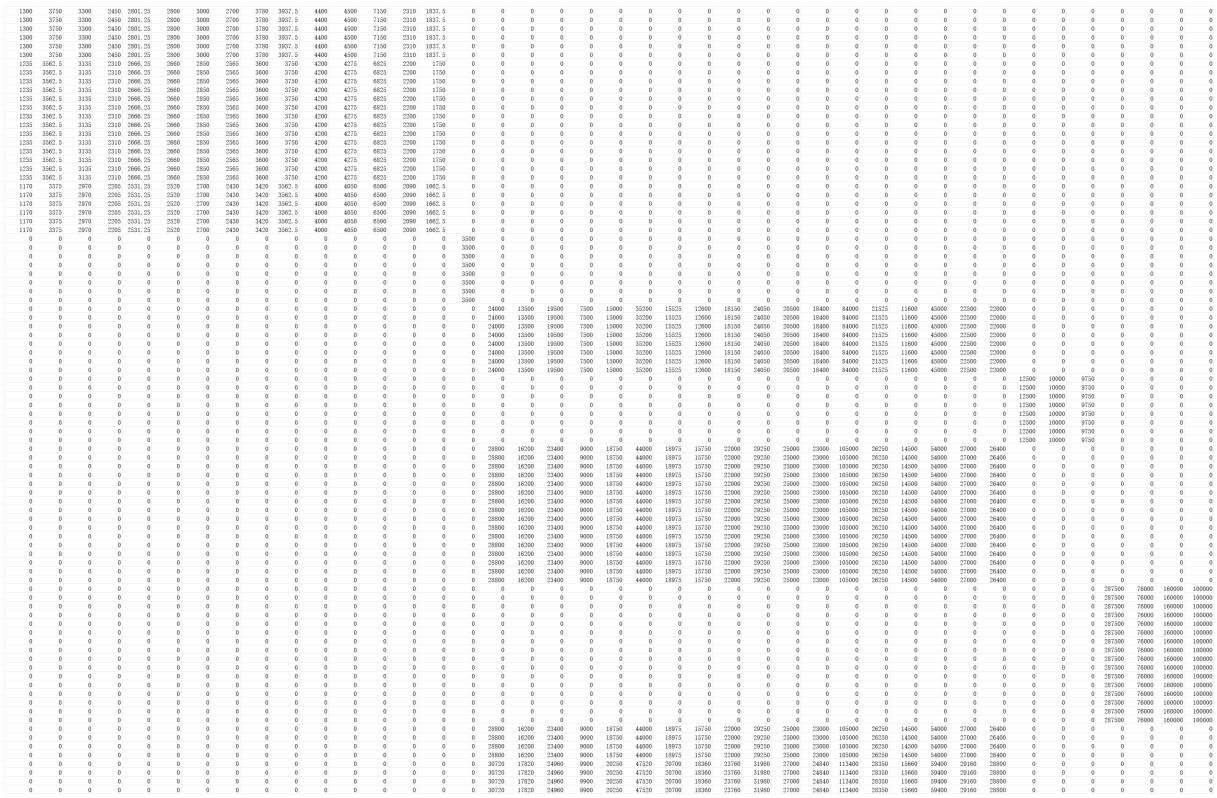

图 4　单亩产出矩阵 B

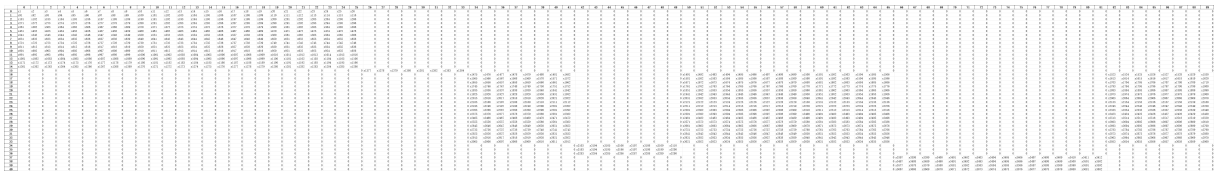

图 6　面积矩阵 E

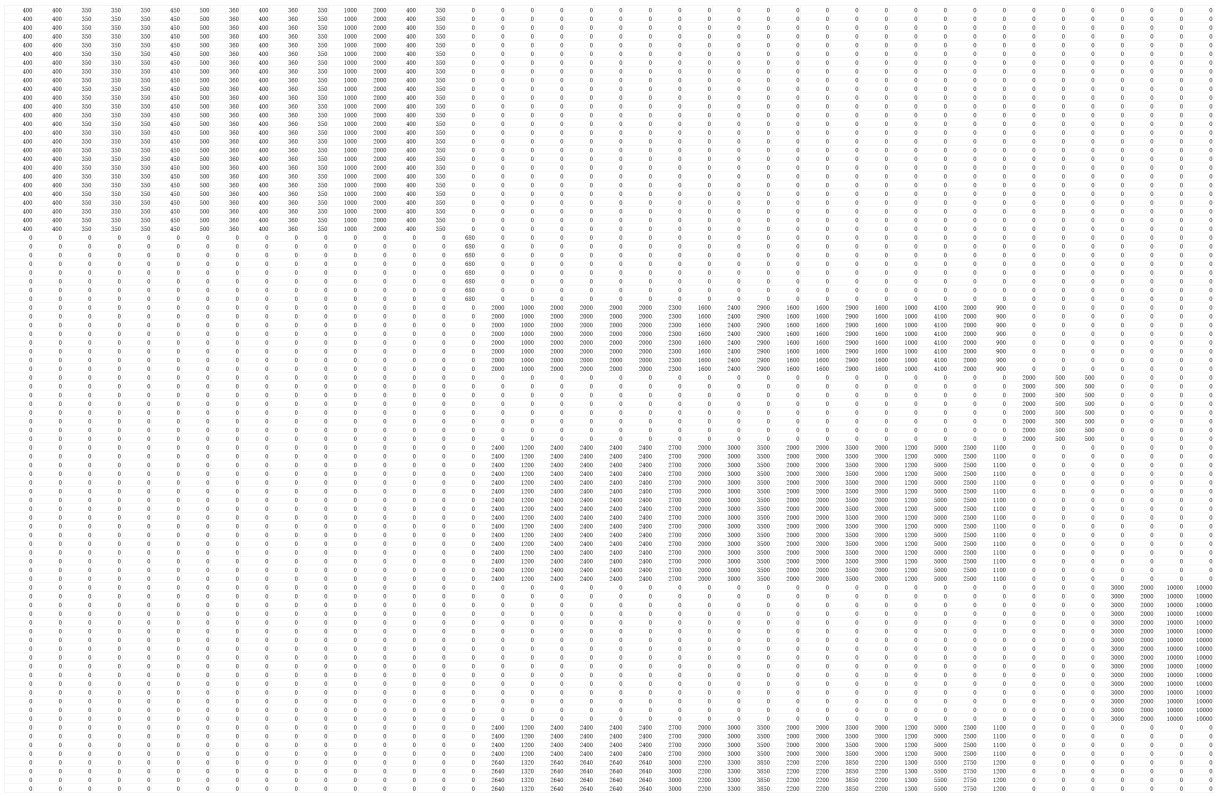

图 5　成本矩阵 *D*

附录 B　文件列表

文件名	功能描述
1.py	问题一程序代码
q2.py	问题二程序代码
q3.c	问题三程序代码
q4.cpp	问题四程序代码

附录 C　代码

问题 1 代码部分 (python)

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Sep  6 16:42:26 2024

@author: Lucius
"""

import pandas as pd
import numpy as np
import sympy as sp
from scipy.optimize import linprog,minimize
import re




attach_1=r"C:\Users\Lucius\Desktop\CUMCM2024Problems\C题\附件1.xlsx"
attach_2=r"C:\Users\Lucius\Desktop\CUMCM2024Problems\C题\附件2.xlsx"
sheet_1="乡村的现有耕地"
sheet_2="乡村种植的农作物"
sheet_3="2023年的农作物种植情况"
sheet_4="2023年统计的相关数据"
df_1 = pd.read_excel(attach_1, sheet_name=sheet_1)
df_2 = pd.read_excel(attach_1, sheet_name=sheet_2)
df_3 = pd.read_excel(attach_2, sheet_name=sheet_3)
df_4 = pd.read_excel(attach_2, sheet_name=sheet_4)
df_4=df_4.drop(index=107).drop(index=108).drop(index=109)


# 统一列名，去除列名中的空格
df_1.columns = df_1.columns.str.strip()
df_2.columns = df_2.columns.str.strip()
```

```
df_3.columns = df_3.columns.str.strip()
df_4.columns = df_4.columns.str.strip()

# 去除数据中的空格
df_1 = df_1.applymap(lambda x: x.strip() if isinstance(x, str)
    else x)
df_2 = df_2.applymap(lambda x: x.strip() if isinstance(x, str)
    else x)
df_3 = df_3.applymap(lambda x: x.strip() if isinstance(x, str)
    else x)
df_4 = df_4.applymap(lambda x: x.strip() if isinstance(x, str)
    else x)









#求预期销售量
# 定义地块类型映射字典
mapping = {
    'A': '平旱地',
    'B': '梯田',
    'C': '山坡地',
    'D': '水浇地',
    'E': '普通大棚',
    'F': '智慧大棚'
}

# 填充缺失的“种植地块”信息
df_3['种植地块'] = df_3['种植地块'].fillna(method='ffill')

# 扩展数据的函数
def expand_row(row):
```

```
    # 拆解作物信息
    crop_ids = str(row['作物编号']).split(',')
    crop_names = str(row['作物名称']).split(',')
    crop_types = str(row['作物类型']).split(',')
    areas = str(row['种植面积/亩']).split(',')

    # 确保长度一致
    length = max(len(crop_ids), len(crop_names), len(crop_types), len(areas))
    crop_ids += [None] * (length - len(crop_ids))
    crop_names += [None] * (length - len(crop_names))
    crop_types += [None] * (length - len(crop_types))
    areas += [None] * (length - len(areas))

    return [
        {
            '种植地块': row['种植地块'],
            '作物编号': crop_id,
            '作物名称': crop_name,
            '作物类型': crop_type,
            '种植面积/亩': area,
            '种植季次': row['种植季次']
        }
        for crop_id, crop_name, crop_type, area in zip(crop_ids, crop_names, crop_types, areas)
    ]

# 扩展所有行
expanded_data = [item for sublist in df_3.apply(expand_row, axis=1) for item in sublist]

# 将扩展后的数据转换为 DataFrame
df_com = pd.DataFrame(expanded_data)

# 添加地块类型列
```

```python
df_com['地块类型'] = df_com['种植地块'].str.extract(r'([A-Z])')[0].map(mapping)

# 保存调整后的数据
path ='附件二去除合并单元格.xlsx'
df_com.to_excel(path, index=False)

# 处理 df_4 中的空格
df_4['地块类型'] = df_4['地块类型'].str.strip()
df_4['种植季次'] = df_4['种植季次'].str.strip()

# 处理 df_com 中的空格
df_com['地块类型'] = df_com['地块类型'].str.strip()
df_com['种植季次'] = df_com['种植季次'].str.strip()

# 统一大小写
df_4['地块类型'] = df_4['地块类型'].str.upper()
df_com['地块类型'] = df_com['地块类型'].str.upper()
df_4['种植季次'] = df_4['种植季次'].str.upper()
df_com['种植季次'] = df_com['种植季次'].str.upper()

# 确保数据类型一致
df_com['作物编号'] = df_com['作物编号'].astype(str)
df_4['作物编号'] = df_4['作物编号'].astype(str)

# 提取用于合并的列
df_com_filtered = df_com[['作物编号', '种植季次', '地块类型', '种植面积/亩']]
df_4_filtered = df_4[['作物编号', '种植季次', '地块类型', '亩产量/斤']]

df_merged = pd.merge(df_com_filtered, df_4_filtered, on=['作物编号', '种植季次', '地块类型'], how='left')


```

```
#智慧大棚第一季缺失
df_merged.at[77,'亩产量/斤'] =4000
df_merged.at[78,'亩产量/斤'] =4500
df_merged.at[81,'亩产量/斤'] =3600
df_merged.at[84,'亩产量/斤'] =3600
df_merged.at[73,'亩产量/斤'] =5500
df_merged.at[74,'亩产量/斤'] =6000


df_merged['种植面积/亩'] = pd.to_numeric(df_merged['种植面积/亩'], errors='coerce')
df_merged['亩产量/斤'] = pd.to_numeric(df_merged['亩产量/斤'], errors='coerce')

df_merged['总产量/斤'] = df_merged['种植面积/亩'] * df_merged['亩产量/斤']
df_merged['作物编号'] = df_merged['作物编号'].astype(int)


df_sales = df_merged.groupby('作物编号')['总产量/斤'].sum().reset_index()
df_sales = df_sales.sort_values(by='作物编号')
output_path = "sales.xlsx"
df_sales.to_excel(output_path, index=True)


sales = df_sales["总产量/斤"].tolist()
#print (len(sales))









df_3['作物编号'] = df_3['作物编号'].astype(int)
df_4['作物编号'] = df_4['作物编号'].astype(int)
```

```
158
159
160
161
162
163
164 #计算单亩产出
165 df_c = df_4[['销售单价/(元/斤)', '亩产量/斤', '作物编号', '地
       块类型','种植季次']]
166 # 确保销售单价是字符串类型
167 df_c['销售单价/(元/斤)'] = df_c['销售单价/(元/斤)'].astype(str
       )
168
169 # 函数：从区间计算中点
170 def calculate_midpoint(price_range):
171     if isinstance(price_range, str):
172         try:
173             low, high = map(float, price_range.split('-'))
174             return (low + high) / 2
175         except ValueError:
176             # 如果分割失败或格式不正确，返回NaN
177             return np.nan
178     else:
179         # 如果不是字符串类型，返回NaN
180         return np.nan
181
182 # 计算每一行的单亩产出均值
183 df_c['中点销售单价'] = df_c['销售单价/(元/斤)'].apply(
       calculate_midpoint)
184 df_c['单亩产出均值'] = df_c['中点销售单价'] * df_c['亩产量/斤'
       ]
185
186 # 合并地块类型和种植季次列为一个新的列
187 df_c['地块类型_种植季次'] = df_c['地块类型'] + df_c['种植季次'
       ]
```

```
188
189
190 # 创建dataframe：行索引为“作物编号”，列索引为“地块类型”，
    值为“单亩产出均值”
191 pivot_table = df_c.pivot_table(
192     index=['地块类型_种植季次'],
193     columns='作物编号',
194     values='单亩产出均值',
195     aggfunc='mean'
196 )
197 data_copy = pivot_table.loc ["普通大棚第一季",:]
198 pivot_table.loc["智慧大棚第一季"] = data_copy
199 pivot_table.fillna(0, inplace=True)
200
201
202 #print(pivot_table)
203 output_path = 'output_pivot_table.xlsx'
204 pivot_table.to_excel(output_path)
205
206
207
208
209
210
211
212
213
214 # 创建可以矩阵相乘的单亩产出矩阵
215 column1 = list(range(1, 42))
216 B = pd.DataFrame(columns=column1)
217
218 # 定义函数以重复数据并将其添加到 DataFrame 中
219 def add_repeated_rows(data_copy, repeat_count):
220     data_repeated = pd.DataFrame([data_copy] * repeat_count)
221     return pd.concat([B, data_repeated], ignore_index=True)
```

```
222
223 data_copy = pivot_table.loc["平旱地单季", :]
224 B = add_repeated_rows(data_copy, 6)
225
226 data_copy = pivot_table.loc["梯田单季", :]
227 B = add_repeated_rows(data_copy, 14)
228
229 data_copy = pivot_table.loc["山坡地单季", :]
230 B = add_repeated_rows(data_copy, 6)
231
232 data_copy = pivot_table.loc["水浇地单季", :]
233 B = add_repeated_rows(data_copy, 8)
234
235 data_copy = pivot_table.loc["水浇地第一季", :]
236 B = add_repeated_rows(data_copy, 8)
237
238 data_copy = pivot_table.loc["水浇地第二季", :]
239 B = add_repeated_rows(data_copy, 8)
240
241 data_copy = pivot_table.loc["普通大棚第一季", :]
242 B = add_repeated_rows(data_copy, 16)
243
244 data_copy = pivot_table.loc["普通大棚第二季", :]
245 B = add_repeated_rows(data_copy, 16)
246
247 data_copy = pivot_table.loc["智慧大棚第一季", :]
248 B = add_repeated_rows(data_copy, 4)
249
250 data_copy = pivot_table.loc["智慧大棚第二季", :]
251 B = add_repeated_rows(data_copy, 4)
252
253 #print(B)
254 output_path = 'B.xlsx'
255 B.to_excel(output_path, index=True)
256
```

```
257
258
259
260
261
262
263
264
265 # 创建符号变量矩阵
266 rows, cols = 41, 90
267 variables = [sp.Symbol(f'x{i+1}') for i in range(rows * cols)]
268 A = pd.DataFrame([variables[i*cols:(i+1)*cols] for i in range(
       rows)])
269
270 # 先将整个 DataFrame 赋值为0
271 A[:] = 0
272 # 确保区域数据的形状一致
273 def set_data(start_row, end_row, start_col, end_col):
274     data = [[variables[i*cols + j] for j in range(start_col,
       end_col)] for i in range(start_row, end_row)]
275     if np.array(data).shape == A.loc[start_row:end_row-1,
       start_col:end_col-1].shape:
276         A.loc[start_row:end_row-1, start_col:end_col-1] = data
277     else:
278         print(f"Shape mismatch for region ({start_row}:{
       end_row}, {start_col}:{end_col}):", np.array(data).shape, A.
       loc[start_row:end_row-1, start_col:end_col-1].shape)
279
280 # 设置不同区域的数据
281 set_data(0, 15, 0, 26)        # 0-14行交0-25列
282 set_data(15, 16, 26, 34)      # 15行交26-33列
283 set_data(16, 34, 34, 42)      # 16-33行交34-41列
284 set_data(34, 37, 42, 50)      # 34-36行交42-49列
285 set_data(16, 34, 50, 66)      # 16-33行交50-64列
286 set_data(37, 41, 66, 82)      # 37-40行交66-80列
```

```
set_data(16, 34, 82, 86)      # 16-33行交82-84列
set_data(16, 34, 86, 90)      # 16-33行交86-88列

# 输出到 Excel 文件
output_path = 'A.xlsx'
A.to_excel(output_path, index=True)
#print(A)









#创建成本矩阵
df_d=df_4[['作物编号','地块类型','种植季次','种植成本/(元/亩)']]
df_d['地块类型_种植季次'] = df_d['地块类型'] + df_d['种植季次']
cost = df_d.pivot_table(
    index='地块类型_种植季次',
    columns='作物编号',
    values='种植成本/(元/亩)')
cost.fillna(0, inplace=True)
data_copy = cost.loc ["普通大棚第一季",:]
cost.loc["智慧大棚第一季"] = data_copy
#print(cost)

column1 = list(range(1, 42))
D = pd.DataFrame(columns=column1)

# 定义函数以重复数据并将其添加到 DataFrame 中
def add_repeated_rows(data_copy, repeat_count):
```

```
    data_repeated = pd.DataFrame([data_copy] * repeat_count)
    return pd.concat([D, data_repeated], ignore_index=True)

data_copy = cost.loc["平旱地单季", :]
D = add_repeated_rows(data_copy, 6)

data_copy = cost.loc["梯田单季", :]
D = add_repeated_rows(data_copy, 14)

data_copy = cost.loc["山坡地单季", :]
D = add_repeated_rows(data_copy, 6)

data_copy = cost.loc["水浇地单季", :]
D = add_repeated_rows(data_copy, 8)

data_copy =cost.loc["水浇地第一季", :]
D = add_repeated_rows(data_copy, 8)

data_copy = cost.loc["水浇地第二季", :]
D = add_repeated_rows(data_copy, 8)

data_copy = cost.loc["普通大棚第一季", :]
D = add_repeated_rows(data_copy, 16)

data_copy =cost.loc["普通大棚第二季", :]
D = add_repeated_rows(data_copy, 16)

data_copy =cost.loc["智慧大棚第一季", :]
D = add_repeated_rows(data_copy, 4)

data_copy = cost.loc["智慧大棚第二季", :]
D = add_repeated_rows(data_copy, 4)

#print(D)
output_path = 'D.xlsx'
```

```
D.to_excel(output_path, index=False)
```

```
#创建地块面积行向量
df_e = df_1[["地块面积/亩"]]
# 转换为列表
list1 = df_e['地块面积/亩'].values.flatten().tolist()

list2 = list1[:34]
list2.append(list1[26:34])
list2.append(list1[26:34])
list2.append(list1[34:50])
list2.append(list1[34:50])
list2.append(list1[50:])
list2.append(list1[50:])

s1=[item for sublist in list2 for item in (sublist if
   isinstance(sublist, list) else [sublist])]

#print (s1)
```

```
389
390 #求利润
391 #先求销售量
392 #在滞销的情况下，不允许产量超过预期销售量
393 #41*90与90*41相乘，得到41*41方阵
394 #取对角线上所有元素，每个元素，如sum1的（1，1），即表示小麦，7
    年，所有土地，总收入
395 #需要留意的是，输出结果sum1的排列并不是严格地以矩阵乘法的顺
    序，而是按x的下标排列，1，10，100……2,20……
396 #仅仅是一年的数据，要用矩阵串联*7
397 #因为仅仅需要乘积矩阵的对角线，故尝试使用np.einsum()函数减轻运
    算量，但是结果有误可能是理解不到位吧，故放弃
398 #矩阵a不是简单水平堆叠，因为不能有7个x1，故更改变量名复制7次；
    b，d是纯数据，直接垂直堆叠
399
400 # 复制并替换符号变量的前缀
401 prefixes = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
402 matrices = []
403
404 for prefix in prefixes:
405     # 替换 'x' 为新的前缀
406     new_matrix = A.applymap(lambda val: sp.Symbol(str(val).
    replace('x', prefix)) if isinstance(val, sp.Symbol) else val
    )
407     matrices.append(new_matrix)
408
409 # 合并7个矩阵，横向堆叠
410 E = pd.concat(matrices, axis=1, ignore_index=True)
411
412 output_path = 'E.xlsx'
413 E.to_excel(output_path, index=True)
414
415
416
417 a=E.to_numpy()
```

```
b=B.to_numpy()
d=D.to_numpy()

b = np.vstack([b] * 7)
d = np.vstack([d] * 7)

c = a @ b
sum1 = np.diag(c)
sum1 = np.sum(sum1)

check1 = open ("check.txt","w")
check1.write(f"{sum1}")
check1.close()

#print (sum1)

e = a @ d
sum2 = np.diag(e)
sum2 = np.sum(sum2)

check2 = open ("check2.txt","w")
check2.write(f"{sum2}")
check2.close()

w = sum1 - sum2
check3 = open("check3.txt","w")
check3.write(f"{w}")
check3.close()


'''
#校验
a=pd.DataFrame(b)
path = "exam1.xlsx"
a.to_excel(path,index=True)
```

```
print(a)
'''
```

```
#设置限制条件
'''
# 解析 w 表达式
def parse_w_expression(w_expression):
    # 确保 w_expression 是字符串
    if not isinstance(w_expression, str):
        w_expression = str(w_expression)

    terms = w_expression.split('+')
    w_dict = {}
    for term in terms:
        coeff, var = term.split('*')
        coeff = float(coeff.strip())
        var = var.strip()
        w_dict[var] = coeff
    return w_dict


# 将 w 表达式解析为字典
w_dict = parse_w_expression(w)

# 提取 w 中的所有变量
variables = list(w_dict.keys())
'''

# 初始解
w0 = np.ones(E.shape[1]) * 6  # 初始化为所有变量等于6
```

```
488
489 # 定义目标函数
490 def objective(w):
491     return -np.sum(w)  # 假设目标是最大化 w 的和，因此取负号最小化
492
493 # 定义约束条件
494 def constraints(w):
495     constraints_list = []
496
497     # 约束1：w 中所有非零元素的数量必须大于等于6
498     constraints_list.append({'type': 'ineq', 'fun': lambda w: np.count_nonzero(w) - 6})
499
500     # 约束2：E 矩阵每列的和小于 s1 中对应的元素
501     for j in range(E.shape[1]):
502         s1_index = j % 90  # 循环使用 s1 的元素
503         constraints_list.append({'type': 'ineq', 'fun': lambda w, j=j, s1_index=s1_index: s1[s1_index] - np.sum(E.iloc[:, j] * w)})
504
505     # 约束3：E 矩阵每行分段和小于 sales 的元素
506     for i in range(E.shape[0]):
507         for seg in range(0, 630, 90):
508             sales_index = i % 90
509             constraints_list.append({'type': 'ineq', 'fun': lambda w, i=i, seg=seg, sales_index=sales_index: sales[sales_index] - np.sum(E.iloc[i, seg:seg + 90] * w)})
510
511     # 约束4：相邻变量下标相同、字母相邻的变量限制
512     for subscript in range(1, 3682):  # 根据实际变量的数量调整范围
513         for letter in 'abcdefg':  # 遍历相邻的字母对 a-b, b-c, ..., f-g
514             if letter != 'g':  # 因为 g 没有相邻的字母
```

```
                var1 = f"{letter}{subscript}"
                var2 = f"{chr(ord(letter) + 1)}{subscript}"
                # 添加约束：相邻的两个变量中至少有一个为 0
                constraints_list.append({'type': 'ineq', 'fun': lambda w, var1=var1, var2=var2: 1 - w[variables.index(var1)] - w[variables.index(var2)]})
'''
    # 约束5：特定行列求和后大于 s1
    for j in range(E.shape[1]):
        s1_index = j % 90
        E_subset = E.iloc[:, j]  # 假设 E 具有适当的维度
        constraint_value = np.sum(E_subset * w)
        constraints_list.append({'type': 'ineq', 'fun': lambda w, s1_index=s1_index, constraint_value=constraint_value: s1[s1_index] - constraint_value})
'''
# 优化求解
res = minimize(objective, w0, method='SLSQP', constraints=constraints(w0), options={'disp': True})

result = res.x
resul_profit = res.fun
print("最大化后的 w：", -res.fun)
print(result)
file = open("result1.1","w")
file.writestr((result))



#生成完整的参数名列表（7个block，每个block从1到3682）
parameter_names = []
for block_prefix in ['a', 'b', 'c', 'd', 'e', 'f', 'g']:
    for i in range(1, 3683):  # 生成 a1 到 g3682
        parameter_names.append(f'{block_prefix}{i}')
```

```
545
546 #对参数名按照数字自然顺序进行排序
547 def natural_key(s):
548     return [int(text) if text.isdigit() else text for text in
    re.split(r'(\d+)', s)]
549
550 parameter_names_sorted = sorted(parameter_names, key=
    natural_key)
551
552 #初始化参数值列表，将不存在的变量值设置为0
553 param_values = [0] * len(parameter_names_sorted)
554
555 #将 result 按顺序填入 param_values 中，覆盖默认的 0
556 for i in range(len(result)):
557     param_values[i] = result[i]
558
559 df_result = pd.DataFrame({
560     'Parameter': parameter_names_sorted,
561     'Value': param_values
562 })
563
564
565 output_file = 'optimal_params_natural_sorted.xlsx'
566 df_result.to_excel(output_file, index=False)
567
568 print(f"优化结果已保存到 {output_file}")
569
570
571
572
573 '''
574 # 输出优化后的 w 值
575 w_optimized = res.x
576 print("最大化后的 w：", -res.fun)
577
```

```

# 创建一个字典来保存优化后的变量值
optimized_values = {var: w_optimized[i] for i, var in
   enumerate(variables)}

# 更新 E 矩阵，将优化后的 w 结果替代进去
for col in E.columns:
    if col in optimized_values:
        E[col] = optimized_values[col]


# 输出矩阵 E 到 Excel 文件
output_path = 'E_optimized.xlsx'
E.to_excel(output_path, index=True)

print(f"已保存优化后的 E 矩阵到: {output_path}")
'''
```

问题 2 代码部分 (python)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Sep  8 15:14:29 2024

@author: Lucius
"""

import pandas as pd
import numpy as np
import gurobipy as gp
from gurobipy import GRB
import itertools



attach_1 = "附件1.xlsx"
```

```
attach_2 = "附件2.xlsx"
sheet_1 = "乡村的现有耕地"
sheet_2 = "乡村种植的农作物"
sheet_3 = "2023年的农作物种植情况"
sheet_4 = "2023年统计的相关数据"

df_sheet_1 = pd.read_excel(attach_1, sheet_name=sheet_1)
df_sheet_2 = pd.read_excel(attach_1, sheet_name=sheet_2)
df_sheet_3 = pd.read_excel(attach_2, sheet_name=sheet_3)
df_sheet_4 = pd.read_excel(attach_2, sheet_name=sheet_4)

df_sheet_2=df_sheet_2.drop(index=41).drop(index=42).drop(index
   =43).drop(index=44)
df_sheet_4_cleaned = df_sheet_4.drop([107, 108, 109], axis=0)


# 清洗 df_sheet_4_cleaned 中的地块类型信息，确保每个作物的种植
   耕地信息正确
df_sheet_4_cleaned['地块类型'] = df_sheet_4_cleaned['地块类型'
   ].str.strip()

# 创建作物编号和种植耕地的对应关系
crop_land_mapping = df_sheet_4_cleaned[['作物编号', '地块类型'
   ]].dropna().drop_duplicates()

# 根据作物编号进行分组，并将多个地块类型进行合并为一个字符串
crop_land_mapping_grouped = crop_land_mapping.groupby('作物编
   号')['地块类型'].apply(lambda x: ','.join(x.unique())).
   reset_index()

# 将作物编号对应的种植耕地替换到 sheet2 的 "种植耕地" 列中
df_sheet_2_updated = df_sheet_2.copy()

# 根据作物编号进行映射
df_sheet_2_updated = df_sheet_2_updated.merge(
```

```
    crop_land_mapping_grouped, on='作物编号', how='left')

# 使用更新后的地块类型信息
df_sheet_2_updated['种植耕地'] = df_sheet_2_updated['地块类型']

# 删除不必要的地块类型列
df_sheet_2_updated = df_sheet_2_updated.drop(columns=['地块类型'])

# 进一步清洗数据，去除多余的空格和换行符，处理种植耕地列中的异常字符
df_sheet_2_updated['种植耕地'] = df_sheet_2_updated['种植耕地'].str.replace(r'\s+', '', regex=True)

# 清洗说明列中的空白符
df_sheet_2_updated['说明'] = df_sheet_2_updated['说明'].str.strip()
df_sheet_2_cleaned = df_sheet_2_updated

df_sheet_1_cleaned = df_sheet_1.copy()
df_sheet_1_cleaned = df_sheet_1_cleaned.applymap(lambda x: x.strip() if isinstance(x, str) else x)

df_sheet_3_cleaned = df_sheet_3.copy()
df_sheet_3_cleaned = df_sheet_3_cleaned.applymap(lambda x: x.strip() if isinstance(x, str) else x)
```

```
#求预期销售量
# 定义地块类型映射字典
mapping = {
    'A': '平旱地',
    'B': '梯田',
    'C': '山坡地',
    'D': '水浇地',
    'E': '普通大棚',
    'F': '智慧大棚'
}

# 填充缺失的“种植地块”信息
df_sheet_3['种植地块'] = df_sheet_3['种植地块'].fillna(method='ffill')

# 扩展数据的函数
def expand_row(row):
    # 拆解作物信息
    crop_ids = str(row['作物编号']).split(',')
    crop_names = str(row['作物名称']).split(',')
    crop_types = str(row['作物类型']).split(',')
    areas = str(row['种植面积/亩']).split(',')

    # 确保长度一致
    length = max(len(crop_ids), len(crop_names), len(crop_types), len(areas))
    crop_ids += [None] * (length - len(crop_ids))
    crop_names += [None] * (length - len(crop_names))
    crop_types += [None] * (length - len(crop_types))
    areas += [None] * (length - len(areas))

    return [
        {
            '种植地块': row['种植地块'],
            '作物编号': crop_id,
```

```
            '作物名称': crop_name,
            '作物类型': crop_type,
            '种植面积/亩': area,
            '种植季次': row['种植季次']
        }
        for crop_id, crop_name, crop_type, area in zip(crop_ids, crop_names, crop_types, areas)
    ]

# 扩展所有行
expanded_data = [item for sublist in df_sheet_3.apply(expand_row, axis=1) for item in sublist]

# 将扩展后的数据转换为 DataFrame
df_com = pd.DataFrame(expanded_data)

# 添加地块类型列
df_com['地块类型'] = df_com['种植地块'].str.extract(r'([A-Z])')[0].map(mapping)

# 保存调整后的数据
path ='附件二去除合并单元格.xlsx'
df_com.to_excel(path, index=False)

# 处理 df_sheet_4 中的空格
df_sheet_4['地块类型'] = df_sheet_4['地块类型'].str.strip()
df_sheet_4['种植季次'] = df_sheet_4['种植季次'].str.strip()

# 处理 df_com 中的空格
df_com['地块类型'] = df_com['地块类型'].str.strip()
df_com['种植季次'] = df_com['种植季次'].str.strip()

# 统一大小写
df_sheet_4['地块类型'] = df_sheet_4['地块类型'].str.upper()
df_com['地块类型'] = df_com['地块类型'].str.upper()
```

```
df_sheet_4['种植季次'] = df_sheet_4['种植季次'].str.upper()
df_com['种植季次'] = df_com['种植季次'].str.upper()

# 确保数据类型一致
df_com['作物编号'] = df_com['作物编号'].astype(str)
df_sheet_4['作物编号'] = df_sheet_4['作物编号'].astype(str)

# 提取用于合并的列
df_com_filtered = df_com[['作物编号', '种植季次', '地块类型',
    '种植面积/亩']]
df_sheet_4_filtered = df_sheet_4[['作物编号', '种植季次', '地
    块类型', '亩产量/斤']]

df_merged = pd.merge(df_com_filtered, df_sheet_4_filtered, on
    =['作物编号', '种植季次', '地块类型'], how='left')


#智慧大棚第一季缺失
df_merged.at[77,'亩产量/斤'] =4000
df_merged.at[78,'亩产量/斤'] =4500
df_merged.at[81,'亩产量/斤'] =3600
df_merged.at[84,'亩产量/斤'] =3600
df_merged.at[73,'亩产量/斤'] =5500
df_merged.at[74,'亩产量/斤'] =6000


df_merged['种植面积/亩'] = pd.to_numeric(df_merged['种植面积/
    亩'], errors='coerce')
df_merged['亩产量/斤'] = pd.to_numeric(df_merged['亩产量/斤'],
     errors='coerce')

df_merged['总产量/斤'] = df_merged['种植面积/亩'] * df_merged[
    '亩产量/斤']
df_merged['作物编号'] = df_merged['作物编号'].astype(int)

```

```
167
168 df_sales = df_merged.groupby('作物编号')['总产量/斤'].sum().
      reset_index()
169 df_sales = df_sales.sort_values(by='作物编号')
170 output_path = "sales.xlsx"
171 df_sales.to_excel(output_path, index=True)
172
173
174 sales = df_sales["总产量/斤"].tolist()
175 #print (len(sales))
176 #print(sales)
177
178
179
180
181
182
183
184
185 # 函数：从区间计算中点
186 def calculate_midpoint(price_range):
187     if isinstance(price_range, str):
188         try:
189             low, high = map(float, price_range.split('-'))
190             return (low + high) / 2
191         except ValueError:
192             # 如果分割失败或格式不正确，返回NaN
193             return np.nan
194     else:
195         # 如果不是字符串类型，返回NaN
196         return np.nan
197 df_sheet_4_cleaned['中点销售单价'] = df_sheet_4_cleaned['销售
      单价/(元/斤)'].apply(calculate_midpoint)
198
199 path='sheet_4.xlsx'
```

```
df_sheet_4_cleaned.to_excel(path)
```

```
sales_a = sales[0:15]
sales_a.append(sales[0:15])
sales_a.append(sales[:34])
sales_a.append(sales[16:])
sales_a.append(sales[16:34])

#展平嵌套列表
sales_a = list(itertools.chain.from_iterable(
    item if isinstance(item, list) else [item] for item in
   sales_a
))

print(len(sales_a))
```

```
df_2024_2030 = df_sheet_4_cleaned[['作物编号', '作物名称', '亩
   产量/斤', '种植成本/(元/亩)', '中点销售单价']].copy()

# 打印合并后的数据检查
print("合并后的数据（初始数据）：")
print(df_2024_2030.head())

# 定义年份和随机波动
years = range(2024, 2031)
```

```
growth_wheat_corn = np.random.uniform(0.05, 0.10, size=len(years))  # 小麦、玉米 5%-10% 增长
growth_other = np.random.uniform(-0.05, 0.05, size=len(years))  # 其他作物 -5%-5%
yield_variation = np.random.uniform(-0.10, 0.10, size=len(years))  # 亩产量波动 ±10%
cost_growth = 1.05 ** np.arange(len(years))  # 种植成本年增长5%
price_growth_vegetable = 1.05 ** np.arange(len(years))  # 蔬菜价格每年增长5%
price_decline_mushroom = np.random.uniform(0.95, 0.99, size=len(years))  # 食用菌价格下降1%-5%

# 初始化存储随机波动后的数据
df_randomized = pd.DataFrame(columns=['作物编号', '作物名称', '年份', '亩产量', '种植成本', '中点销售单价'])

# 对每个年份逐步应用波动生成新的数据
for year, yield_var, cost_grow in zip(years, yield_variation, cost_growth):
    for idx, row in df_2024_2030.iterrows():
        new_row = {
            '作物编号': row['作物编号'],
            '作物名称': row['作物名称'],
            '年份': year,
            '亩产量': row['亩产量/斤'] * (1 + yield_var),  # 亩产量波动
            '种植成本': row['种植成本/(元/亩)'] * cost_grow,  # 种植成本年增长5%
        }

        # 销售单价按不同作物类别进行调整
        if row['作物名称'] in ['豇豆''刀豆''芸豆''土豆''西红柿''茄子''菠菜''青椒''菜花''包菜''油麦菜'
                               '小青菜''黄瓜''生菜辣椒''空心菜''黄
```

```
    心菜''芹菜''大白菜''白萝卜''红萝卜']:
257             new_row['中点销售单价'] = row['中点销售单价'] *
    price_growth_vegetable[year - 2024]  # 蔬菜类价格每年增长5%
258         elif row['作物名称'] in ['食用菌']:
259             new_row['中点销售单价'] = row['中点销售单价'] *
    price_decline_mushroom[year - 2024]  # 食用菌价格下降
260         else:
261             new_row['中点销售单价'] = row['中点销售单价']
262
263         # 将生成的新数据行添加到 df_randomized 中
264         df_randomized = pd.concat([df_randomized, pd.DataFrame
    ([new_row])], ignore_index=True)
265
266 # 打印随机波动后的数据检查
267 print("随机波动后的数据: ")
268 print(df_randomized.head())
269
270 # 生成 sales_a 并确保每年对应 107 行
271 sales_a = sales[:107]  # 每年 107 行
272 sales_a = sales_a * len(years)  # 复制足够的行数
273
274 # 使用 apply 函数来计算 sales 列
275 def calculate_sales(row):
276     year_index = row['年份'] - 2024  # 年份从 2024 开始
277     row_index = row.name % 107  # 使用 mod 107 确保每年 107 行
    数据
278     if row['作物名称'] in ['小麦', '玉米']:
279         return sales_a[row_index] * (1 + growth_wheat_corn[
    year_index])
280     else:
281         return sales_a[row_index] * (1 + growth_other[
    year_index])
282
283 # 应用 calculate_sales 函数
284 df_randomized['sales'] = df_randomized.apply(calculate_sales,
```

```
    axis=1)

# 查看结果
print(df_randomized.head())

# 保存最终结果
df_randomized.to_excel("df_2024_2030_randomized_with_sales.xlsx", index=False)


# 将 df_2024_2030 与 df_sheet_2 基于 '作物编号' 进行合并，补充 '作物类型'
df_2024_2030 = pd.merge(df_2024_2030, df_sheet_2[['作物编号', '作物类型']], on='作物编号', how='left')
df_2024_2030.head()


df_2024_2030.to_excel("df_2024_2030.xlsx")



# 定义问题的基本参数
num_iterations = 1000  # 迭代次数
years = range(2024, 2031)  # 考虑的年份
best_solution = None
best_revenue = -np.inf  # 初始化最大收益

# 豆类作物
legume_crops = df_2024_2030[df_2024_2030['作物类型'].isin(['粮食（豆类）', '蔬菜（豆类）'])]['作物编号'].unique()


situation = 1

# 随机搜索
```

```
for iteration in range(num_iterations):
    total_revenue = 0  # 初始化总收益

    # 存储当前方案的种植面积
    current_solution = {}

    for year in years:
        for land_id in df_sheet_1['地块名称'].unique():
            available_area = df_sheet_1[df_sheet_1['地块名称'] == land_id]['地块面积/亩'].values[0]
            crop_areas = np.zeros(len(df_2024_2030['作物编号'].unique()))

            # 1. 随机分配种植面积，确保不超过可用面积（约束1）
            remaining_area = available_area
            for i, crop_id in enumerate(df_2024_2030['作物编号'].unique()):
                # 确保种植面积不超过地块可用面积
                crop_areas[i] = np.random.uniform(0, remaining_area)
                remaining_area -= crop_areas[i]
                if remaining_area <= 0:
                    break

            # 2. 确保三年内种植一次豆类作物（约束2）
            if year % 3 == 0 and not any(df_2024_2030['作物编号'].isin(legume_crops)):
                legume_idx = np.random.choice(np.where(df_2024_2030['作物编号'].isin(legume_crops))[0])
                crop_areas[legume_idx] = max(5, np.random.uniform(0, available_area))  # 至少种植5亩

            # 3. 确保同一地块不能连续种植相同作物（约束3）
            if year > 2024 and (crop_id, land_id, year-1) in current_solution:
```

```
                crop_areas[i] = 0  # 防止连续种植相同作物

            # 4. 确保每种作物的最小种植面积（5亩）（约束4）
            crop_areas = np.maximum(crop_areas, 5)

            # 5. 根据地块类型适应性分配作物（约束5）
            land_type = df_sheet_1[df_sheet_1['地块名称'] ==
land_id]['地块类型'].values[0]
            for i, crop_id in enumerate(df_2024_2030['作物编号
'].unique()):
                crop_type = df_2024_2030[df_2024_2030['作物编
号'] == crop_id]['作物类型'].values[0]
                if land_type == '平旱地' and crop_type != '粮
食':
                    crop_areas[i] = 0
                elif land_type == '水浇地' and crop_type not
in ['水稻', '蔬菜']:
                    crop_areas[i] = 0
                elif land_type == '普通大棚' and crop_type not
 in ['蔬菜', '食用菌']:
                    crop_areas[i] = 0
                elif land_type == '智慧大棚' and crop_type !=
'蔬菜':
                    crop_areas[i] = 0

            # 计算每个作物的总产量并计算收益
            for crop_id, area in zip(df_2024_2030['作物编号'].
unique(), crop_areas):
                # 计算总产量 = 种植面积 * 亩产量
                yield_per_mu = df_2024_2030[df_2024_2030['作物
编号'] == crop_id]['亩产量/斤'].values[0]
                total_yield = area * yield_per_mu

                # 获取作物的销售量和销售价格
                sales = df_randomized[(df_randomized['作物编号
```

```
    '] == crop_id) & (df_randomized['年份'] == year)]['sales'].
    values[0]
368                 sales_price = df_2024_2030[df_2024_2030['作物
    编号'] == crop_id]['中点销售单价'].values[0]
369
370                 # 6. 产量与销售量的匹配（约束6）
371                 if total_yield <= sales:
372                     total_revenue += total_yield * sales_price
373                 else:
374                     excess = total_yield - sales
375                     if situation == 1:
376                         total_revenue += sales * sales_price
    # 超出部分滞销
377                     elif situation == 2:
378                         total_revenue += sales * sales_price +
     excess * sales_price * 0.5  # 超出部分按50%出售
379
380                 # 存储当前的种植面积
381                 current_solution[(crop_id, land_id, year)] =
    area
382
383
384
385 # 在计算收益时判断情况
386 for crop_id, area in zip(df_2024_2030['作物编号'].unique(),
    crop_areas):
387     # 计算总产量 = 种植面积 * 亩产量
388     yield_per_mu = df_2024_2030[df_2024_2030['作物编号'] ==
    crop_id]['亩产量/斤'].values[0]
389     total_yield = area * yield_per_mu
390
391     # 获取作物的销售量和销售价格
392     sales = df_randomized[(df_randomized['作物编号'] ==
    crop_id) & (df_randomized['年份'] == year)]['sales'].values
    [0]
```

```
    sales_price = df_2024_2030[df_2024_2030['作物编号'] == crop_id]['中点销售单价'].values[0]

    # 判断情况
    if total_yield <= sales:
        total_revenue += total_yield * sales_price
    else:
        excess = total_yield - sales
        if situation == 1:
            total_revenue += sales * sales_price  # 超出部分滞销
        elif situation == 2:
            total_revenue += sales * sales_price + excess * sales_price * 0.5  # 超出部分按50%出售

# 输出最优结果
print("最优方案的总收益: ", best_revenue)
print("最优种植方案: ", best_solution)
```

问题 3 代码部分 (python)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Sep  8 18:52:36 2024

@author: Lucius
"""


import pandas as pd
import numpy as np
import itertools
from joblib import Parallel, delayed


```

```python
attach_1 = "附件1.xlsx"
attach_2 = "附件2.xlsx"
sheet_1 = "乡村的现有耕地"
sheet_2 = "乡村种植的农作物"
sheet_3 = "2023年的农作物种植情况"
sheet_4 = "2023年统计的相关数据"

df_sheet_1 = pd.read_excel(attach_1, sheet_name=sheet_1)
df_sheet_2 = pd.read_excel(attach_1, sheet_name=sheet_2)
df_sheet_3 = pd.read_excel(attach_2, sheet_name=sheet_3)
df_sheet_4 = pd.read_excel(attach_2, sheet_name=sheet_4)

df_sheet_2=df_sheet_2.drop(index=41).drop(index=42).drop(index=43).drop(index=44)
df_sheet_4_cleaned = df_sheet_4.drop([107, 108, 109], axis=0)


# 清洗 df_sheet_4_cleaned 中的地块类型信息，确保每个作物的种植耕地信息正确
df_sheet_4_cleaned['地块类型'] = df_sheet_4_cleaned['地块类型'].str.strip()

# 创建作物编号和种植耕地的对应关系
crop_land_mapping = df_sheet_4_cleaned[['作物编号', '地块类型']].dropna().drop_duplicates()

# 根据作物编号进行分组，并将多个地块类型进行合并为一个字符串
crop_land_mapping_grouped = crop_land_mapping.groupby('作物编号')['地块类型'].apply(lambda x: ','.join(x.unique())).reset_index()

# 将作物编号对应的种植耕地替换到 sheet2 的 "种植耕地" 列中
df_sheet_2_updated = df_sheet_2.copy()

# 根据作物编号进行映射
```

```
df_sheet_2_updated = df_sheet_2_updated.merge(
    crop_land_mapping_grouped, on='作物编号', how='left')

# 使用更新后的地块类型信息
df_sheet_2_updated['种植耕地'] = df_sheet_2_updated['地块类型'
    ]

# 删除不必要的地块类型列
df_sheet_2_updated = df_sheet_2_updated.drop(columns=['地块类
    型'])

# 进一步清洗数据，去除多余的空格和换行符，处理种植耕地列中的异
    常字符
df_sheet_2_updated['种植耕地'] = df_sheet_2_updated['种植耕地'
    ].str.replace(r'\s+', '', regex=True)

# 清洗说明列中的空白符
df_sheet_2_updated['说明'] = df_sheet_2_updated['说明'].str.
    strip()
df_sheet_2_cleaned = df_sheet_2_updated

df_sheet_1_cleaned = df_sheet_1.copy()
df_sheet_1_cleaned = df_sheet_1_cleaned.applymap(lambda x: x.
    strip() if isinstance(x, str) else x)

df_sheet_3_cleaned = df_sheet_3.copy()
df_sheet_3_cleaned = df_sheet_3_cleaned.applymap(lambda x: x.
    strip() if isinstance(x, str) else x)
```

```
72
73  #求预期销售量
74  # 定义地块类型映射字典
75  mapping = {
76      'A': '平旱地',
77      'B': '梯田',
78      'C': '山坡地',
79      'D': '水浇地',
80      'E': '普通大棚',
81      'F': '智慧大棚'
82  }
83
84  # 填充缺失的“种植地块”信息
85  df_sheet_3['种植地块'] = df_sheet_3['种植地块'].fillna(method=
        'ffill')
86
87  # 扩展数据的函数
88  def expand_row(row):
89      # 拆解作物信息
90      crop_ids = str(row['作物编号']).split(',')
91      crop_names = str(row['作物名称']).split(',')
92      crop_types = str(row['作物类型']).split(',')
93      areas = str(row['种植面积/亩']).split(',')
94
95      # 确保长度一致
96      length = max(len(crop_ids), len(crop_names), len(
        crop_types), len(areas))
97      crop_ids += [None] * (length - len(crop_ids))
98      crop_names += [None] * (length - len(crop_names))
99      crop_types += [None] * (length - len(crop_types))
100     areas += [None] * (length - len(areas))
101
102     return [
103         {
104             '种植地块': row['种植地块'],
```

```
            '作物编号': crop_id,
            '作物名称': crop_name,
            '作物类型': crop_type,
            '种植面积/亩': area,
            '种植季次': row['种植季次']
        }
        for crop_id, crop_name, crop_type, area in zip(crop_ids, crop_names, crop_types, areas)
    ]

# 扩展所有行
expanded_data = [item for sublist in df_sheet_3.apply(expand_row, axis=1) for item in sublist]

# 将扩展后的数据转换为 DataFrame
df_com = pd.DataFrame(expanded_data)

# 添加地块类型列
df_com['地块类型'] = df_com['种植地块'].str.extract(r'([A-Z])')[0].map(mapping)

# 保存调整后的数据
path ='附件二去除合并单元格.xlsx'
df_com.to_excel(path, index=False)

# 处理 df_sheet_4 中的空格
df_sheet_4['地块类型'] = df_sheet_4['地块类型'].str.strip()
df_sheet_4['种植季次'] = df_sheet_4['种植季次'].str.strip()

# 处理 df_com 中的空格
df_com['地块类型'] = df_com['地块类型'].str.strip()
df_com['种植季次'] = df_com['种植季次'].str.strip()

# 统一大小写
df_sheet_4['地块类型'] = df_sheet_4['地块类型'].str.upper()
```

```
df_com['地块类型'] = df_com['地块类型'].str.upper()
df_sheet_4['种植季次'] = df_sheet_4['种植季次'].str.upper()
df_com['种植季次'] = df_com['种植季次'].str.upper()

# 确保数据类型一致
df_com['作物编号'] = df_com['作物编号'].astype(str)
df_sheet_4['作物编号'] = df_sheet_4['作物编号'].astype(str)

# 提取用于合并的列
df_com_filtered = df_com[['作物编号', '种植季次', '地块类型',
    '种植面积/亩']]
df_sheet_4_filtered = df_sheet_4[['作物编号', '种植季次', '地
    块类型', '亩产量/斤']]

df_merged = pd.merge(df_com_filtered, df_sheet_4_filtered, on
    =['作物编号', '种植季次', '地块类型'], how='left')


#智慧大棚第一季缺失
df_merged.at[77,'亩产量/斤'] =4000
df_merged.at[78,'亩产量/斤'] =4500
df_merged.at[81,'亩产量/斤'] =3600
df_merged.at[84,'亩产量/斤'] =3600
df_merged.at[73,'亩产量/斤'] =5500
df_merged.at[74,'亩产量/斤'] =6000


df_merged['种植面积/亩'] = pd.to_numeric(df_merged['种植面积/
    亩'], errors='coerce')
df_merged['亩产量/斤'] = pd.to_numeric(df_merged['亩产量/斤'],
     errors='coerce')

df_merged['总产量/斤'] = df_merged['种植面积/亩'] * df_merged[
    '亩产量/斤']
df_merged['作物编号'] = df_merged['作物编号'].astype(int)
```

```
166
167
168 df_sales = df_merged.groupby('作物编号')['总产量/斤'].sum().
       reset_index()
169 df_sales = df_sales.sort_values(by='作物编号')
170 output_path = "sales.xlsx"
171 df_sales.to_excel(output_path, index=True)
172
173
174 sales = df_sales["总产量/斤"].tolist()
175 #print (len(sales))
176 #print(sales)
177
178
179
180
181
182
183
184
185 # 函数：从区间计算中点
186 def calculate_midpoint(price_range):
187     if isinstance(price_range, str):
188         try:
189             low, high = map(float, price_range.split('-'))
190             return (low + high) / 2
191         except ValueError:
192             # 如果分割失败或格式不正确，返回NaN
193             return np.nan
194     else:
195         # 如果不是字符串类型，返回NaN
196         return np.nan
197 df_sheet_4_cleaned['中点销售单价'] = df_sheet_4_cleaned['销售
       单价/(元/斤)'].apply(calculate_midpoint)
198
```

```
path='sheet_4.xlsx'
df_sheet_4_cleaned.to_excel(path)
```

```
sales_a = sales[0:15]
sales_a.append(sales[0:15])
sales_a.append(sales[:34])
sales_a.append(sales[16:])
sales_a.append(sales[16:34])

#展平嵌套列表
sales_a = list(itertools.chain.from_iterable(
    item if isinstance(item, list) else [item] for item in
   sales_a
))

print(len(sales_a))
```

```
df_2024_2030 = df_sheet_4_cleaned[['作物编号', '作物名称', '亩
   产量/斤', '种植成本/(元/亩)', '中点销售单价']].copy()

# 打印合并后的数据检查
print("合并后的数据（初始数据）：")
print(df_2024_2030.head())

# 定义年份和随机波动
```

```python
years = range(2024, 2031)
growth_wheat_corn = np.random.uniform(0.05, 0.10, size=len(years))  # 小麦、玉米 5%-10% 增长
growth_other = np.random.uniform(-0.05, 0.05, size=len(years))  # 其他作物 -5%-5%
yield_variation = np.random.uniform(-0.10, 0.10, size=len(years))  # 亩产量波动 ±10%
cost_growth = 1.05 ** np.arange(len(years))  # 种植成本年增长5%
price_growth_vegetable = 1.05 ** np.arange(len(years))  # 蔬菜价格每年增长5%
price_decline_mushroom = np.random.uniform(0.95, 0.99, size=len(years))  # 食用菌价格下降1%-5%

# 初始化存储随机波动后的数据
df_randomized = pd.DataFrame(columns=['作物编号', '作物名称', '年份', '亩产量', '种植成本', '中点销售单价'])

# 对每个年份逐步应用波动生成新的数据
for year, yield_var, cost_grow in zip(years, yield_variation, cost_growth):
    for idx, row in df_2024_2030.iterrows():
        new_row = {
            '作物编号': row['作物编号'],
            '作物名称': row['作物名称'],
            '年份': year,
            '亩产量': row['亩产量/斤'] * (1 + yield_var),  # 亩产量波动
            '种植成本': row['种植成本/(元/亩)'] * cost_grow,  # 种植成本年增长5%
        }

        # 销售单价按不同作物类别进行调整
        if row['作物名称'] in ['豇豆''刀豆''芸豆''土豆''西红柿''茄子''菠菜''青椒''菜花''包菜''油麦菜'
```

```
                                '小青菜''黄瓜''生菜辣椒''空心菜''黄心菜''芹菜''大白菜''白萝卜''红萝卜']:
            new_row['中点销售单价'] = row['中点销售单价'] * price_growth_vegetable[year - 2024]  # 蔬菜类价格每年增长5%
        elif row['作物名称'] in ['食用菌']:
            new_row['中点销售单价'] = row['中点销售单价'] * price_decline_mushroom[year - 2024]  # 食用菌价格下降
        else:
            new_row['中点销售单价'] = row['中点销售单价']

        # 将生成的新数据行添加到 df_randomized 中
        df_randomized = pd.concat([df_randomized, pd.DataFrame([new_row])], ignore_index=True)

# 打印随机波动后的数据检查
print("随机波动后的数据：")
print(df_randomized.head())

# 生成 sales_a 并确保每年对应 107 行
sales_a = sales[:107]  # 每年 107 行
sales_a = sales_a * len(years)  # 复制足够的行数

# 使用 apply 函数来计算 sales 列
def calculate_sales(row):
    year_index = row['年份'] - 2024  # 年份从 2024 开始
    row_index = row.name % 107  # 使用 mod 107 确保每年 107 行数据
    if row['作物名称'] in ['小麦', '玉米']:
        return sales_a[row_index] * (1 + growth_wheat_corn[year_index])
    else:
        return sales_a[row_index] * (1 + growth_other[year_index])

# 应用 calculate_sales 函数
```

```
df_randomized['sales'] = df_randomized.apply(calculate_sales,
    axis=1)

# 查看结果
print(df_randomized.head())

# 保存最终结果
df_randomized.to_excel("df_2024_2030_randomized_with_sales.
    xlsx", index=False)


# 将 df_2024_2030 与 df_sheet_2 基于 '作物编号' 进行合并，补充
    '作物类型'
df_2024_2030 = pd.merge(df_2024_2030, df_sheet_2[['作物编号',
    '作物类型']], on='作物编号', how='left')
df_2024_2030.head()


df_2024_2030.to_excel("df_2024_2030.xlsx")









df_land = pd.read_excel("附件1.xlsx", sheet_name="乡村的现有耕
```

```
    地")
315
316 # 获取作物的种类信息
317 crop_info = df_2024_2030[['作物编号', '作物名称', '作物类型',
    '中点销售单价', '亩产量/斤']]
318
319
320
321 # 获取豆类作物编号，确保三年内至少种植一次豆类作物
322 legume_crops = crop_info[crop_info['作物类型'].isin(['粮食（豆
    类）', '蔬菜（豆类）'])]['作物编号'].unique()
323
324 # 修改豆类作物的分类，确保豆类作物按规定视作相同类型
325 crop_info['作物类型'] = crop_info['作物类型'].replace({
326     '粮食（豆类）': '粮食',
327     '蔬菜（豆类）': '蔬菜'
328 })
329
330
331
332 # 初始化地块种植历史记录
333 land_history = {land_id: {} for land_id in df_land['地块名称'
    ].unique()}
334
335 # 定义互补品和替代品调整规则
336 def adjust_area_cost_for_complement_and_substitute(crop_id,
    sales_price_change, crop_areas, cost_change, crop_info):
337     crop_type_j = crop_info[crop_info['作物编号'] == crop_id][
    '作物类型'].values[0]
338
339     for related_crop_id in crop_info['作物编号'].unique():
340         if related_crop_id == crop_id:
341             continue  # 跳过自身
342
343         crop_type_k = crop_info[crop_info['作物编号'] ==
```

```
related_crop_id]['作物类型'].values[0]
344
345         # 使用loc[]获取作物编号对应的索引，确保索引在有效范围内
346         related_idx = crop_info.loc[crop_info['作物编号'] == related_crop_id].index
347
348         # 检查索引是否超出数组范围
349         if related_idx.empty or related_idx[0] >= len(crop_areas):
350             continue  # 跳过不存在的索引
351
352         # 判断作物之间的关系
353         if crop_type_j == crop_type_k:  # 替代品
354             if sales_price_change > 0:  # 作物j的价格上升
355                 crop_areas[related_idx[0]] -= np.random.uniform(0.05, 0.10) * crop_areas[related_idx[0]]  # 种植面积减少
356                 cost_change[related_idx[0]] += np.random.uniform(0.05, 0.10)  # 成本增加
357             else:  # 作物j的价格下降
358                 crop_areas[related_idx[0]] += np.random.uniform(0.05, 0.10) * crop_areas[related_idx[0]]  # 种植面积增加
359                 cost_change[related_idx[0]] -= np.random.uniform(0.05, 0.10)  # 成本减少
360         else:  # 互补品
361             if sales_price_change > 0:  # 作物j的价格上升
362                 crop_areas[related_idx[0]] += np.random.uniform(0.05, 0.10) * crop_areas[related_idx[0]]  # 种植面积增加
363                 cost_change[related_idx[0]] -= np.random.uniform(0.05, 0.10)  # 成本减少
364             else:  # 作物j的价格下降
365                 crop_areas[related_idx[0]] -= np.random.
```

```
    uniform(0.05, 0.10) * crop_areas[related_idx[0]]  # 种植面积减少
366                 cost_change[related_idx[0]] += np.random.uniform(0.05, 0.10)  # 成本增加
367
368 # 定义每次迭代的函数
369 def single_iteration(years, df_land, df_2024_2030, df_randomized, legume_crops, land_history):
370     total_revenue = 0
371     current_solution = {}
372
373     for year in years:
374         for land_id in df_land['地块名称'].unique():
375             available_area = df_land[df_land['地块名称'] == land_id]['地块面积/亩'].values[0]
376             crop_areas = np.zeros(len(df_2024_2030['作物编号'].unique()))
377             cost_change = np.zeros(len(df_2024_2030['作物编号'].unique()))  # 用于存储作物成本的波动情况
378             remaining_area = available_area
379
380             # 随机分配种植面积，并确保同一地块不能连续种植相同作物
381             for i, crop_id in enumerate(df_2024_2030['作物编号'].unique()):
382                 # 检查是否在上一年种植过该作物，如果是，则跳过
383                 if year > 2024 and land_id in land_history and land_history[land_id].get(year - 1) == crop_id:
384                     continue  # 防止连续种植相同作物
385
386                 crop_areas[i] = np.random.uniform(0, remaining_area)
387                 remaining_area -= crop_areas[i]
388                 if remaining_area <= 0:
389                     break
```

```
390
391             # 确保三年内至少种植一次豆类作物
392             if year % 3 == 0 and not any(df_2024_2030['作物编
    号'].isin(legume_crops)):
393                 legume_idx = np.random.choice(np.where(
    df_2024_2030['作物编号'].isin(legume_crops))[0])
394                 crop_areas[legume_idx] = max(5, np.random.
    uniform(0, available_area))  # 至少种植5亩
395
396             # 确保每种作物的最小种植面积（5亩）
397             crop_areas = np.maximum(crop_areas, 5)
398
399             # 根据地块类型适应性分配作物
400             land_type = df_land[df_land['地块名称'] == land_id
    ]['地块类型'].values[0]
401             for i, crop_id in enumerate(df_2024_2030['作物编号
    '].unique()):
402                 crop_type = crop_info[crop_info['作物编号'] ==
     crop_id]['作物类型'].values[0]
403                 if land_type == '平旱地' and crop_type != '粮
    食':
404                     crop_areas[i] = 0
405                 elif land_type == '水浇地' and crop_type not
    in ['水稻', '蔬菜']:
406                     crop_areas[i] = 0
407                 elif land_type == '普通大棚' and crop_type not
     in ['蔬菜', '食用菌']:
408                     crop_areas[i] = 0
409                 elif land_type == '智慧大棚' and crop_type !=
    '蔬菜':
410                     crop_areas[i] = 0
411
412             # 计算每个作物的总产量并计算收益
413             for crop_id, area in zip(df_2024_2030['作物编号'].
    unique(), crop_areas):
```

```
                yield_per_mu = df_2024_2030[df_2024_2030['作物编号'] == crop_id]['亩产量/斤'].values[0]
                total_yield = area * yield_per_mu
                sales = df_randomized[(df_randomized['作物编号'] == crop_id) & (df_randomized['年份'] == year)]['sales'].values[0]
                sales_price = df_2024_2030[df_2024_2030['作物编号'] == crop_id]['中点销售单价'].values[0]

                # 计算作物j的销售价格波动
                sales_price_change = np.random.uniform(-0.05, 0.05)  # 假设销售价格变化
                adjust_area_cost_for_complement_and_substitute(crop_id, sales_price_change, crop_areas, cost_change, crop_info)

                # 产量与销售量的匹配
                if total_yield <= sales:
                    total_revenue += total_yield * sales_price
                else:
                    excess = total_yield - sales
                    total_revenue += sales * sales_price + excess * sales_price * 0.5  # 超出部分按50%出售

                # 存储当前的种植面积
                current_solution[(crop_id, land_id, year)] = area

            # 记录本年度的作物种植情况，以便下一年检查连续种植约束
            land_history[land_id][year] = np.argmax(crop_areas)  # 存储作物编号

     return total_revenue

```

```
# 迭代次数
num_iterations = 100  # 降低计算负担
years = range(2024, 2031)  # 年份范围

# 并行执行随机搜索
results = Parallel(n_jobs=-1)(delayed(single_iteration)(years,
    df_land, df_2024_2030, df_randomized, legume_crops,
   land_history) for _ in range(num_iterations))

# 选择最优结果
best_revenue = max(results)
print("最优方案的总收益：", best_revenue)
```