基于线性规划和蒙特卡洛模拟的农作物种植决策

摘要

在科学发展观的指导下，优化农业经济结构是经济战略调整的核心内容，也是农村经济工作的重中之重。近年来，农业生产面临的各种风险和挑战持续制约着农业经济的发展。为此，本文结合实际农业生产情况，基于线性规划模型、蒙特卡洛模拟算法，构建优化模型，以提升农业生产力，促进农村经济的持续发展。

针对问题一，首先进行数据预处理，要得到 2024 到 2030 年的最优作物种植策略，即以收益最大化为目标，因此在本问中可以采取**线性规划模型**。以决策的总收益作为**目标函数**，作物在不同地块上不同年份不同季次的种植面积作为**决策变量**来建立优化模型。为满足文中种植要求，还需要设立**约束条件**来进行规划求解，对于问题一中的两种不同情形，即作物产物大于预期销售量时滞销或贬价，通过设置超产量来对目标函数进行约束。对题目信息中所给的各种种植条件，本文将通过详细给出数学表达式并以此限制模型。通过该模型，最终得到 2024 到 2030 年的最优种植策略。

针对问题二，由于影响农业生产的各种因素，如预期销售量、亩产量、作物价格等在实际中是会变化的。为了增加模型在实际应用中的健硕，本问假设相关因素受到的随机扰动符合**正态分布**，并借助**蒙特卡洛模拟**的思想，产生这些随机变量的多组样本，并以此来模拟 2024-2030 年多种情形下的农业生产因素的变化。通过**线性规划模型**解得在不同因素下的最优作物种植策略。最终，为了充分应对农业生产中的各种风险挑战，本问采取鲁棒规划思想，选取在多次模拟下总利益最小的种植决策作为最优的种植方案。

针对问题三，本问在问题二的基础上，增加了不同作物之间的互补性因素及同种作物不同指标的相关性因素，为了得到更接近实际的数据，本问以第二问的多次预测数据为基础，通过计算不同作物间的**交叉价格弹性**与相关性矩阵，在决策模型中假如互补性、可替代性以及相关性因素的条件约束，进而模拟出更接近实际的种植方案。

关键词：　线性规划　蒙特卡洛模拟　交叉价格弹性　循环迭代

一、问题背景与重述

1.1 问题背景

基于农村的实际情况，乡村经济的可持续发展正面临耕地资源有限、市场价格波动和需求不确定等多重挑战。因此，制定合理的作物种植策略，以有效应对这些风险并最大化利润，是推动乡村经济持续健康发展的关键。

1.2 题目信息

1. 耕地信息：某乡村共有 34 个大小不同的露天耕地以及 20 个大棚，耕地分为平旱地、梯田、山坡地和水浇地 4 种类型，大棚分为 16 个普通大棚和 4 个智慧大棚。
2. 耕地种植信息：平旱地、梯田和山坡地适宜每年种植一季粮食类作物；水浇地适宜每年种植一季水稻或两季蔬菜。
3. 普通大棚适宜每年种植一季蔬菜和一季食用菌，智慧大棚适宜每年种植两季蔬菜。同一地块（含大棚）每季可以合种不同的作物。
4. 每种作物不能在同一地块连续重茬种植；所有土地 (含大棚) 每三年至少种植一次大豆；为方便管理，每种作物在单个地块 (含大棚) 种植的面积不宜过小。
5. 在附件一中给出了每块耕地的面积以及不同作物适宜耕种土地的详细信息；附件二给出了 2023 年的农作物种植情况以及不同作物的亩产量、耕种成本以及销售单价。

1.3 问题重述

问题一：若某种作物每季的总产量超过其预期销售量，超过部分无法正常销售。在两种情形下分别给出 2024 到 2030 年农作物的最有种植方案。(情形一：超过部分滞销，造成浪费。情形二：超过部分按 2023 年销售价格的 50% 出售)

问题二：综合考虑各种农作物的预期销售量、亩产量、种植成本和销售价格的不确定性以及潜在的种植风险，给出该乡村 2024 2030 年农作物的最优种植方案。

问题三：以问题二为基础，综合考虑农作物之间的替代性和互补性，预期销售量与销售价格、种植成本之间的相关性等因素，给出该乡村 2024 2030 年农作物的最优种植策略，通过模拟数据进行求解，并与问题 2 的结果作比较分析。

二、模型假设

1. 假设不考虑作物的储存和运输成本，同时作物在储存、运输和销售过程中不存在损耗。

2. 假设 2023 年生产的所有作物全部售出，不存在滞销。

3. 假设作物的历史销售量能够代表未来的预期销售量，即通过作物在 2023 年的销售量代表未来的预期销售量。

4. 假设作物的销售价格、成本、亩产量、销售量等因素的变化服从正态分布。

三、符号说明

符号	说明
Q	农作物一年的销售总利润
S_{ijys}	第 i 块地上在第 y 年第 s 季种植作物 j 的面积
M_{ijs}	第 i 块地上在第 s 季作物 j 的亩产量
C_{ijs}	第 i 块地在第 s 季作物 j 的种植成本
P_{js}	作物 j 在第 s 季的销售单价
d_j	作物 j 的预期销售量
r_j	作物 j 的季节限制
a_i	地块 i 的土地面积
p_j	2023 年作物 j 的总产量
m	作物的最小种植面积
e_{1y}	小麦和玉米的预期销售量在第 y 年的随机扰动
e_{2y}	其余作物的预期销售量在第 y 年的随机扰动
e_{3y}	作物第 y 年的亩产量受到的随机扰动
e_{4y}	食用菌第 y 年的销售价格受到的随机扰动
$E_{j_1j_2y}$	作物 j_1与作物j_2在第 y 年的交叉价格弹性

四、数据预处理

观察题设所给附件信息，数据信息过于分散，为了方便后续问题的分析与求解，因此对所给信息进行数据关联与整合从而建立一个规范合理的数据结构。

指标选取：

以 i 为指标代指不同的地块，以 j 为指标代指不同的作物，以 y 为指标代指不同的年份，以 s 为指标代指不同的季度。

数据关联：

附件一的“乡村现有耕地”表中描述了耕地信息，附件二的“2023 统计的相关数据”表中描述作物在不同类型的耕地中的种植信息，因此首先通过编程，剔除无用信息后以“地块类型”为关键字关联两张表格，从而得到了不同的地块上的种植信息。

数据整合：

- **季节限制**:

  在作物种植信息中发现，一些能够两季种植的作物在不同的季度，亩产量、种植成本和销售单价均不相同。因此为区别季节产生的差异，以变量 r_j 代表作物的季节限制。若作物 j 只能单季种植，则 r_j=1；若作物能够两季种植，则 r_j=2。

- **预期销量**:

  题设假定预期销售量相对于 2023 年保持稳定，但并未给出 2023 年作物的具体销售状况。因此在假设 2023 年生产的作物全部售出的前提下，本文以 2023 年作物的产量替代预期销售量

$$d_j = p_j$$

  其中 d_j 表示作物 j 的预期销售量，p_j 表示 2023 年作物 j 的总产量，p_j 可以如下公式计算

$$p_j = \sum_{s=1}^{2} \sum_{i=1}^{54} S_{ijs} * M_{ijs}$$

  其中 S_{ijs} 表示第 i 块地上作物 j 在第 s 季的种植面积，M_{ijs} 表示第 i 块地上作物 j 在第 s 季的亩产量。

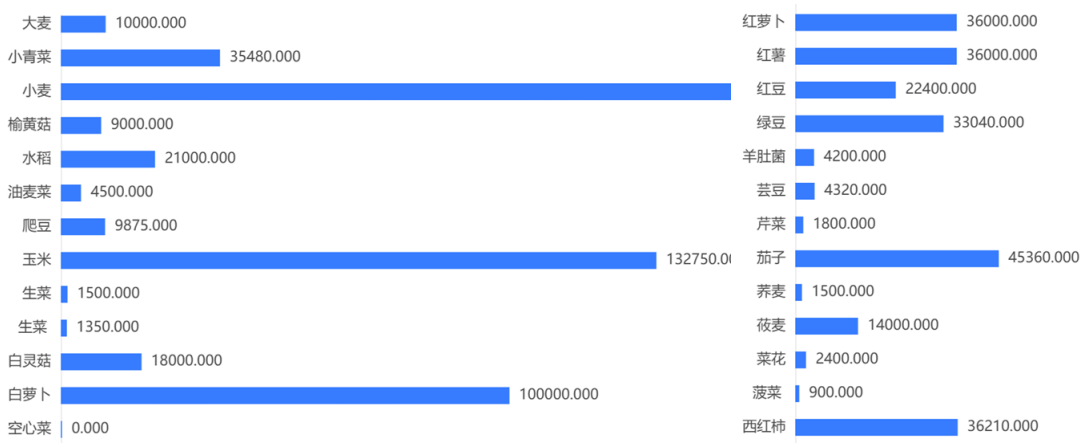

图 1 2023 年部分作物总销量

- **价格选定**

根据实际生活与历史经验，可以假定作物的单价服从以价格区间的中点为均值的正态分布，因此本文选取作物在 2023 年的价格区间的中值作为销售价格。

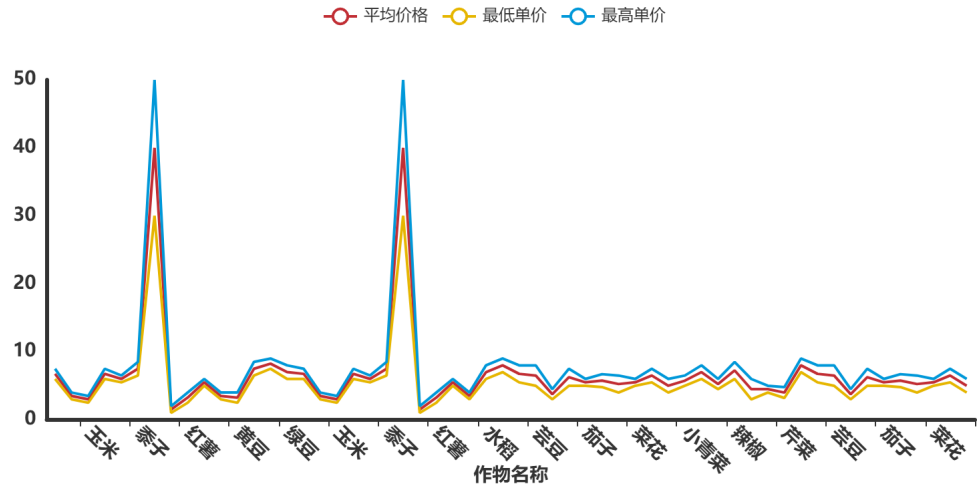

图 2 2023 年部分作物的平均价格

- **数据的可视化处理**

为了对数据有更直观的认识，以方便后续对模型的建立，本文也对其余的数据，如不同作物的亩产量以及不同地块的土地面积等进行了可视化。

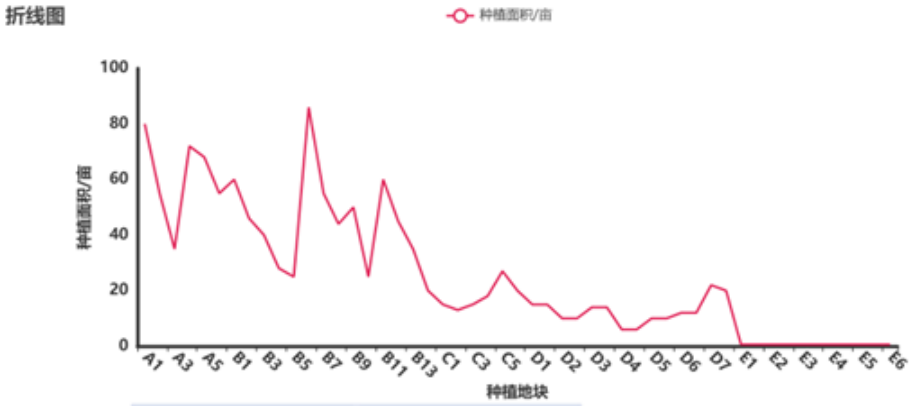

图 3 不同地块面积

从图中可以看出该村耕地的大部分都主要集中在 A 类和 B 类土地，即平旱地与梯田，小部分为 C 类与 D 类，即山坡地与水浇地。大棚的面积与露天耕地相比相对较小，但其种植周期较短，管理较为方便，因此其也具有较高的经济价值。从部分作物总销量的图中可以看出，以黄豆、爬豆、黑豆和红豆为首的豆类作物销售量普遍较低，而以油麦菜、黄瓜、红萝卜等为首的蔬菜类作物普遍较高。由此说明在建立模型时蔬菜类经济效益较高的作物可以优先考虑，豆类作物虽然销

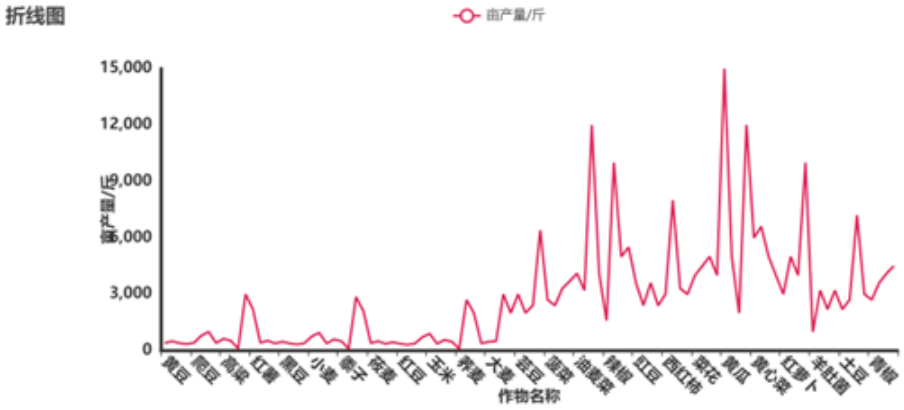

图 4 2023 年部分作物总销量

售量较低，但因含有豆类作物根菌的土壤能够促进其它作物的生长，因此实际种植中也应该考虑该作物。

五、问题分析与模型求解

5.1 问题一

5.1.1 问题一分析

问题一要求制定 2024 至 2030 年最优的作物种植方案使销售利润最大化，可以通过构建目标利益最大化决策模型来对该问题进行求解。为了以最佳的方式分配有限的土地资源，并充分地发挥资源的效能最终获取最佳经济效益，对于本问将采取线性规划模型。[1]

同时题设规定作物超过预期销售量的部分无法正常销售，要求在两种不同的情形下分别给中最优的种植方案。

在情形一中，作物超过预期销售量的部分滞销，造成浪费。因此在情形一下，作物的总种植面积不能超过其与销售量，否则必然会造成亏损。

在情形二中，作物超过预期销量的部分以 2023 年单价的 50% 出售。假设在某种方案下销售的总利润为 Q, 当某种作物耕种面积增产 Δs 时，由于耕地的总面积保持不变，必然伴随着另一种作物的减产 Δs。在此情形下，销售的总利润变为

$$Q^{'} = Q + \Delta s * (0.5u_1 - c_1) - \Delta s * (u_2 - c_2)$$

其中 u_1,c_1 分别代表增产作物每亩的收益和成本，u_2，c_2 分别代表减产作物每亩的收益和成本，因此虽然增产作物每亩的利润降低，但只要满足

$$0.5u_1 - c_1 > u_2 - c_2$$

利润 $Q^{'}$ 相较于 Q 仍然有所上涨。因此情形二与情形一不同的是，作物的总产量允许超过预期销售量。

5.1.2 问题一模型建立与求解

情形一:

- **目标函数**:

  选定利润 Q 作为目标函数，以第 s 季地块 i 上作物 j 的种植面积 S_{ijs} 作为决策变量，此时 Q 的计算式为

$$Q=\sum_{y=1}^{6}\sum_{s=1}^{2}\sum_{i=1}^{54}\sum_{j=1}^{40}S_{ijys}*(M_{ijs}*P_{js}-C_{ijs})$$

  式中 S_{ijys} 代表第 i 块地上在第 y 年第 s 季种植作物 j 的面积，M_{ijs} 代表第 i 块地上在第 s 季作物 j 的亩产量,P_{js} 代表作物 j 在第 s 季的销售单价,C_{ijs} 代表第 i 块地在第 s 季作物 j 的种植成本。所以该模型的目标函数为

$$max\quad Q=\sum_{y=1}^{6}\sum_{s=1}^{2}\sum_{i=1}^{54}\sum_{j=1}^{40}S_{ijs}*(M_{ijs}*P_{js}-C_{ijs})$$

- **约束条件**:

  1. 在地块 i 种植作物的面积必然小于等于地块 i 的面积，因此第一个约束条件为

$$0\leq\sum_{j=1}^{40}S_{ijys}\leq a_i\quad s=1,2\quad j=1,2,\cdots,54\quad y=1,2\cdots,6$$

     其中 a_i 表示地块 i 的面积。

  2. 为方便管理，作物要集中种植，即每块地上作物 j 的种植面积不能过小，由于每个地块的大小不尽相同，例如平旱地与大棚的面积差异非常大，因此我们可以假设每块地上作物的种植面积不能小于该地块的 5% 该约束条件为

$$a_i*5\%\leq S_{ijys}\quad s=1,2\quad j=1,2,\cdots,54\quad i=1,2\cdots,40\quad y=1,2\cdots,6$$

  3. 在情形一下，由前面的分析作物的总产量不能超过其预期销售量，因此该约束条件为

$$d_j\geq\sum_{s=1}^{2}\sum_{j=1}^{54}M_{ijs}*S_{ijys}\quad i=1,2\cdots,40\quad y=1,2\cdots,6$$

  4. 在同一块地上禁止两年或两季内重茬种植，即连续两年内作物 j 在地块 i 上的总种植面积不能超过地块 i 的面积

$$S_{ijys}+S_{ijy+1s}\leq a_i\quad s=1,2\quad j=1,2,\cdots,54\quad i=1,2,\cdots,40\quad y=1,2,\cdots,5$$

$$S_{ijy1}+S_{ijy2}\leq a_i\quad j=1,2,\cdots,54\quad i=1,2,\cdots,40\quad y=1,2,\cdots,5$$

5. 豆类作物根菌的土壤有利于其他作物生长, 因此每块地所有土地三年内至少种植一次豆类作物

$$\sum_{s=1}^{2} S_{ijys}+S_{ijy+1s}+S_{ijy+2s} \geq a_i \quad y=1,2,3,4 \quad i=1,2\cdots,54 \quad \text{作物 j 为豆类}$$

6. 为方便说明，定义新的作物种植面积为 S'_{ijys}。由附件所给信息，地块上的作物种植条件大致可分为，在平旱地、梯田和山坡地上，只能单季种植非水稻的粮食类作物；在水浇地上，可单季种植水稻或两季种植蔬菜；在普通大棚可在第一季种植蔬菜，第二季种植大白菜、白萝卜、红萝卜和食用菌；在智慧大棚可两季种植除大白菜、白萝卜、红萝卜外的蔬菜，因此 S'_{ijys} 可按如下定义

$$S'_{ijys}=\begin{cases} S_{ijys} & \text{地块 i 为平旱地、梯田或山坡地} \\ & \text{作物 j 为除水稻外的粮食类作物} \quad s=1 \\ S_{ijys} & \text{地块 i 为水浇地，作物 j 为水稻和除大白} \\ & \text{菜、白萝卜、红萝卜外的蔬菜，s=1} \\ S_{ijys} & \text{地块 i 为水浇地，作物 j 为大} \\ & \text{白菜、白萝卜、红萝卜,s=2} \\ S_{ijys} & \text{地块 i 为普通大棚，作物 j 为水稻和除大白} \\ & \text{菜、白萝卜、红萝卜外的蔬菜，s=1} \\ S_{ijys} & \text{地块 i 为普通大棚，作物 j 为菌类，s=2} \\ S_{ijys} & \text{地块 i 为智慧大棚，作物 j 为水稻和除大白} \\ & \text{菜、白萝卜、红萝卜外的蔬菜，s=1,2} \\ 0 & \text{其他} \end{cases}$$

- **模型**:

$$\max \quad Q=\sum_{y=1}^{6}\sum_{s=1}^{2}\sum_{i=1}^{54}\sum_{j=1}^{40} S'_{ijys}*(M_{ijs}*P_{js}-C_{ijs})$$

$$\text{s.t.} \begin{cases} 0 \leq \sum_{j=1}^{40} S_{ijys} \leq a_i & s=1,2 \quad j=1,2,\cdots,54 \\ & y=1,2\cdots,6 \\ m \leq S_{ijys} & s=1,2 \quad j=1,2,\cdots,54 \quad i=1,2\cdots,40 \\ & y=1,2\cdots,6 \\ d_j \geq \sum_{s=1}^{2}\sum_{j=1}^{54} M_{ijys}*S_{ijs} & i=1,2\cdots,40 \quad y=1,2\cdots,6 \\ S_{ijys}+S_{ijy+1s} \leq a_i & s=1,2 \quad j=1,2,\cdots,54 \quad i=1,2,\cdots,40 \\ & y=1,2,\cdots,5 \\ S_{ijy1}+S_{ijy2} \leq a_i & j=1,2,\cdots,54 \quad i=1,2,\cdots,40 \\ & y=1,2,\cdots,5 \end{cases} \tag{1}$$

- **模型简化**：经分析可发现，在情形一下当作物 j 的总产量超过其预期销售量时，超出的部分必然会造成亏损，假设此时的总利润为 Q。若适当减少其生产面积，

使该种作物的总产量恰好等于其预期销售量，设作物 j 在地块 i 上减少的面积为 ΔS_{ij}，此时的总利润 $Q^{'}$ 可表示为

$$Q^{'} = = Q + \sum_{i=1}^{54} \Delta S_{ij} * C_{ijs}$$

式中 C_{ijs} 表示作物 j 在地块 i 上的种植成本。因此对产量超过预期销售量的方案必然不是最优的，因为总能找到利润更大一种方案。所以可以去掉限制条件

$$d_j \geq \sum_{s=1}^{2} \sum_{j=1}^{54} M_{ijs} * S_{ijys} \quad i = 1, 2 \cdots, 40 \quad y = 1, 2 \cdots, 6$$

模型简化为

$$\max \quad Q = \sum_{y=1}^{6} \sum_{s=1}^{2} \sum_{i=1}^{54} \sum_{j=1}^{40} S'_{ijys} * (M_{ijs} * P_{js} - C_{ijs})$$

$$\text{s.t.} \begin{cases} 0 \leq \sum_{j=1}^{40} S_{ijys} \leq a_i & s = 1, 2 \quad j = 1, 2, \cdots, 54 \\ & y = 1, 2 \cdots, 6 \\ a_i * 5\% \leq S_{ijys} & s = 1, 2 \quad j = 1, 2, \cdots, 54 \quad i = 1, 2 \cdots, 40 \\ & y = 1, 2 \cdots, 6 \\ S_{ijys} + S_{ijy+1s} \leq a_i & s = 1, 2 \quad j = 1, 2, \cdots, 54 \quad i = 1, 2, \cdots, 40 \\ & y = 1, 2, \cdots, 5 \\ S_{ijy1} + S_{ijy2} \leq a_i & j = 1, 2, \cdots, 54 \quad i = 1, 2, \cdots, 40 \\ & y = 1, 2, \cdots, 5 \end{cases} \tag{2}$$

- **模型求解**：通过解上述模型，得到最优种植策略。

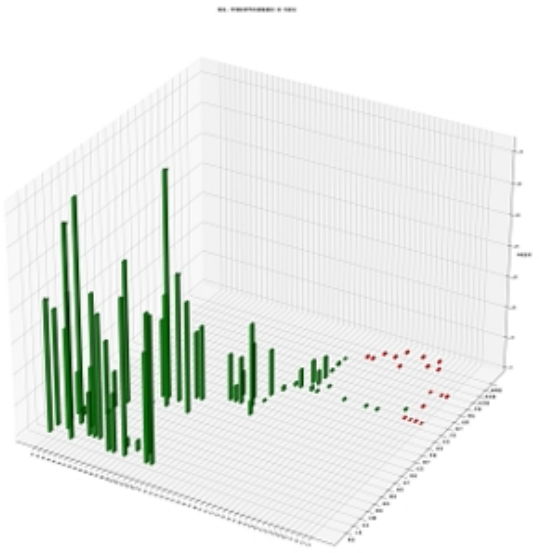

图 5 情形一种植策略（以 2024 年为例）

情形二:

- **目标函数**:

  在情形二下，作物超出预期销售量的部分以 2023 年价格的 50% 出售，因此定义

$$w_{iy} = \sum_{s=1}^{2}\sum_{j=1}^{54} S'_{ijys} * M_{ijs} - d_j$$

  此外定义函数

$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{3}$$

  此时目标函数可以写为

$$\max \quad Q = \sum_{y=1}^{6}\sum_{s=1}^{2}\sum_{i=1}^{54}\sum_{j=1}^{40} S'_{ijys} * (M_{ijs} * P_{js} - C_{ijs}) - f(w_{iy}) * 0.5 * P_{js}$$

- **约束条件**:

  由模型简化可知情形一同样不需要考虑作物的总产量小于其预期销售量，因此与情形一的约束条件相同。

- **模型**:

$$\max \quad Q = \sum_{y=1}^{6}\sum_{s=1}^{2}\sum_{i=1}^{54}\sum_{j=1}^{40} S'_{ijys} * (M_{ijs} * P_{js} - C_{ijs}) - f(w_{iy}) * 0.5 * P_{js}$$

$$\text{s.t.} \begin{cases} 0 \leq \sum_{j=1}^{40} S_{ijys} \leq a_i & s = 1,2 \quad j = 1,2,\cdots,54 \\ & y = 1,2\cdots,6 \\ a_i * 5\% \leq S_{ijys} & s = 1,2 \quad j = 1,2,\cdots,54 \quad i = 1,2\cdots,40 \\ & y = 1,2\cdots,6 \\ S_{ijys} + S_{ijy+1s} \leq a_i & s = 1,2 \quad j = 1,2,\cdots,54 \quad i = 1,2,\cdots,40 \\ & y = 1,2,\cdots,5 \\ S_{ijy1} + S_{ijy2} \leq a_i & j = 1,2,\cdots,54 \quad i = 1,2,\cdots,40 \\ & y = 1,2,\cdots,5 \end{cases} \tag{4}$$

- **模型求解**:

  情形二中的模型求解如下

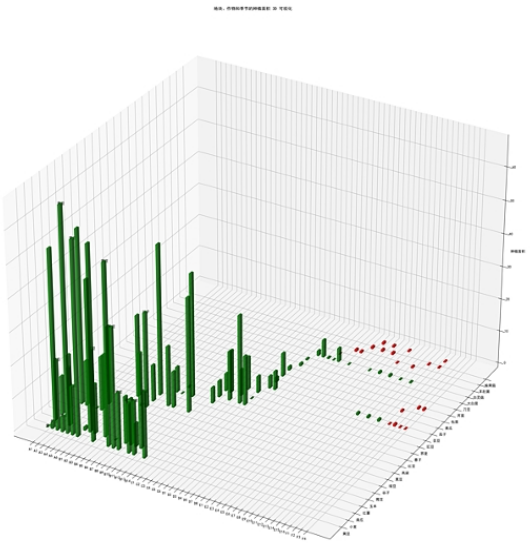

图 6 情形二种植策略（以 2024 年为例）

5.2 问题二

5.2.1 问题二分析

在实际生活中，由于受到市场波动、气候变化等因素的影响，农作物的预期销售量、销售价格、种植成本以及亩产量都受到了不同程度的随机扰动，这些市场的不确定性以及种植风险也给农业生产带来了更多的挑战。因此为了充分应对这些不确定性，本文采取蒙特卡洛模拟的思想以选取最优的种植方案。

蒙特卡洛模拟是一种基于概率的统计方法，可以直接处理风险因素的不确定性：将不确定性以概率分布的形式表示，建立风险决策的随机模型，对随机变量抽样试验，模拟结果分析上，不仅能得出决策目标输出的期望值等多种统计量，也可给出概率分布。[2] 因此为解决该问题，可采用蒙特卡洛模拟的思想，生成多组随机数，用以模拟 2024 2030 年预期销售量、销售单价以及种植成本等因素的变化过程，再通过建立最优化模型解得各组的最优种植方案，采取鲁棒规划思想，选取在多次模拟下总利益最小的种植决策作为最优的种植方案。

同时从种植者的角度出发，我们也必须要有“兜底”的思想，即考虑在最坏情况下如何改进自己的种植方案，以此来做出应对。最坏情况下最稳健方案（Worst-Case Robust Solution）的求解在决策分析和优化问题中非常重要。这个概念通常用于应对不确定性和极端情况，确保方案在最不利条件下依然具有良好的表现。在问题二中我们认为最坏情况就是指市场和种植条件（如天气、土壤等因素）的负面影响达到边界条件的最大值，我们在算法中把它具体表现为预期销售量、亩产量等超参数手动调整为最坏情况。

5.2.2 问题二模型建立与求解

蒙特卡洛模拟:

1. 随机扰动

   (a) 预期销售量的随机扰动:

   小麦和玉米的增长率介于 5% 和 10% 之间，因此可假设其第 y 年的增长率为符合 Normal(0.075,0.0125) 的随机变量 e_{1y}, 且 e_{1y} 的取值介于 5% 和 10% 之间。

   其他作物每年大约有 ±5% 的变化率，因此可假设其第 y 年的增长率为符合 Normal(0,0.025) 的随机变量 e_{2y}, 取值介于 ±5% 之间。

   (b) 亩产量的随机扰动:

   作物的亩产量每年大约有 ±10% 的变化率，因此可假设其第 y 年的增长率为符合 Normal(0,0.05) 的随机变量 e_{3y}, 取值介于 ±10% 之间。

   (c) 种植成本地扰动:

   农作物的种植成本平均每年增长 5%。

   (d) 销售单价的随机扰动:

   粮食类作物基本保持稳定，蔬菜类作物的价格平均每年增长 5%，羊肚菌的价格每年下降的幅度约为 5%。其他菌类的价格则以 1% 至 5% 的趋势逐年递减，因此可假设其他菌类的变化率为符合 Normal(0.03,0.01) 的随机变量 e_{4y}，且 e_{4y} 的取值介于 1% 至 5% 之间。

2. 参数更新

   由于作物的预期销售量、亩产量、种植成本和销售单价每一年都受到了不同程度的扰动，因此重新定义 d_{jy} 表示作物 j 在第 y 年的预期销售量、M_{ijys} 表示地块 i 上作物 j 在第 y 年第 s 季的亩产量、C_{ijys} 表示地块 i 上作物 j 在第 y 年第 s 季的种植成本、P_{jys} 表示作物 j 在第 y 年第 s 季的销售单价。

   (a) 预期销售量可通过下式计算

   $$d_{jy} = \begin{cases} d_{jy-1} * (1 + e_{1y}) & \text{作物 j 为小麦或玉米} \\ d_{jy-1} * (1 + e_{2y}) & \text{其他作物} \end{cases} \tag{5}$$

   d_{j0} 表示 2023 年的预期销售量。

   (b) 亩产量的更新

   $$M_{ijys} = M_{ijy-1s} * (1 + e_{3y}) \tag{6}$$

   M_{ij0s} 表示 2023 年的亩产量。

   (c) 种植成本的更新

   $$C_{ijys} = C_{ijy-1s} * 1.05 \tag{7}$$

   C_{ij0s} 表示 2023 年的亩产量。

(d) 销售单价的更新

$$P_{jys}=\begin{cases} P_{jy-1s} & \text{作物 j 为粮食类} \\ P_{jy-1s}*1.05 & \text{作物 j 为蔬菜类} \\ P_{jy-1s}*0.95 & \text{作物 j 为羊肚菌} \\ P_{jy-1s}*(1+e_{4y}) & \text{其他菌类} \end{cases} \tag{8}$$

P_{j0s} 表示 2023 年的销售单价。

3. 模型建立

在问题一的基础上以贬值 50% 处理滞销作物的基础上。本文决策模型为

$$\max \quad Q=\sum_{y=1}^{6}\sum_{s=1}^{2}\sum_{i=1}^{54}\sum_{j=1}^{40}S'_{ijys}*(M_{ijys}*P_{jys}-C_{ijys})-f(w_{iy})*0.5*P_{jys}$$

$$\text{s.t.}\quad \begin{cases} 0\le\sum_{j=1}^{40}S_{ijys}\le a_i & s=1,2 \quad j=1,2,\cdots,54 \\ & y=1,2\cdots,6 \\ a_i*5\%\le S_{ijys} & s=1,2 \quad j=1,2,\cdots,54 \quad i=1,2\cdots,40 \\ & y=1,2\cdots,6 \\ d_j\ge\sum_{s=1}^{2}\sum_{j=1}^{54}M_{ijys}*S_{ijs} & i=1,2\cdots,40 \quad y=1,2\cdots,6 \\ S_{ijys}+S_{ijy+1s}\le a_i & s=1,2 \quad j=1,2,\cdots,54 \quad i=1,2,\cdots,40 \\ & y=1,2,\cdots,5 \\ S_{ijy1}+S_{ijy2}\le a_i & j=1,2,\cdots,54 \quad i=1,2,\cdots,40 \\ & y=1,2,\cdots,5 \end{cases} \tag{9}$$

Worst-Case Robust Solution：

1. 定义不确定性集：问题二涉及不确定性为作物的预期销售量、亩产量、销售价格，以及这些不确定因素的扰动范围大致已知。

2. 构建鲁棒优化模型（Robust Optimization）：将不确定性作为约束条件纳入优化模型中，目的是优化在所有这些约束条件下的最坏情况，为了模拟最坏情况，我们这里直接确定各种超参数的值为它们变化范围的最小值，例如小麦和玉米的预期销售量调整为固定的 5

3. 求解模型，并将该模型得到的最稳健种植方案与上文的蒙特卡洛模拟方法迭代得到的种植方案比较，判断蒙特卡洛模拟方法得到的种植方案的最终收益相较于最稳健方案的最终收益是否有明显提高。

模型求解

通过蒙特卡洛模拟生成多种情形下的预期销售量、亩产量、亩成本，最终采取鲁棒规划的思想，选取最稳健的种植方案作为决策：

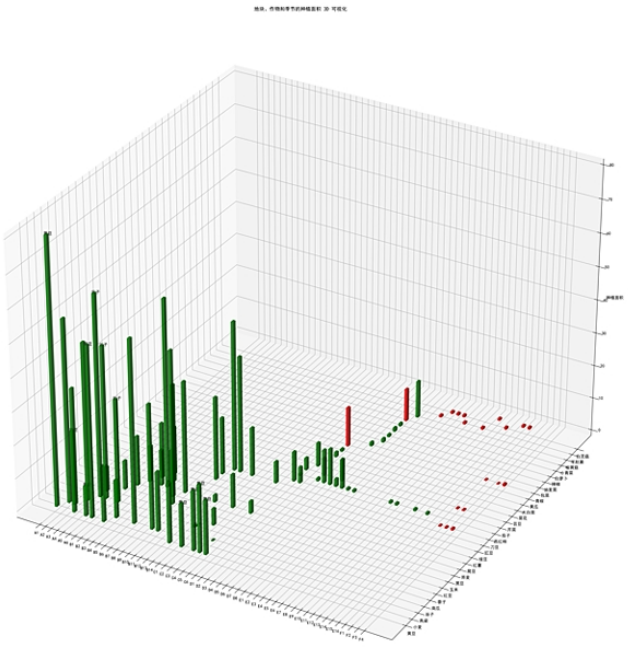

图 7 问题二模型求解结果 (以 2024 年为例)

5.3 问题三

5.3.1 问题三分析

在实际情形中，作物的预期销售量、销售单价、种植成本的并不是随机变化的，例如不同作物之间存在替代性与互补性，同种作物的预期销售量、销售单价、种植成本之间则存在相关性。

- **相关性分析**

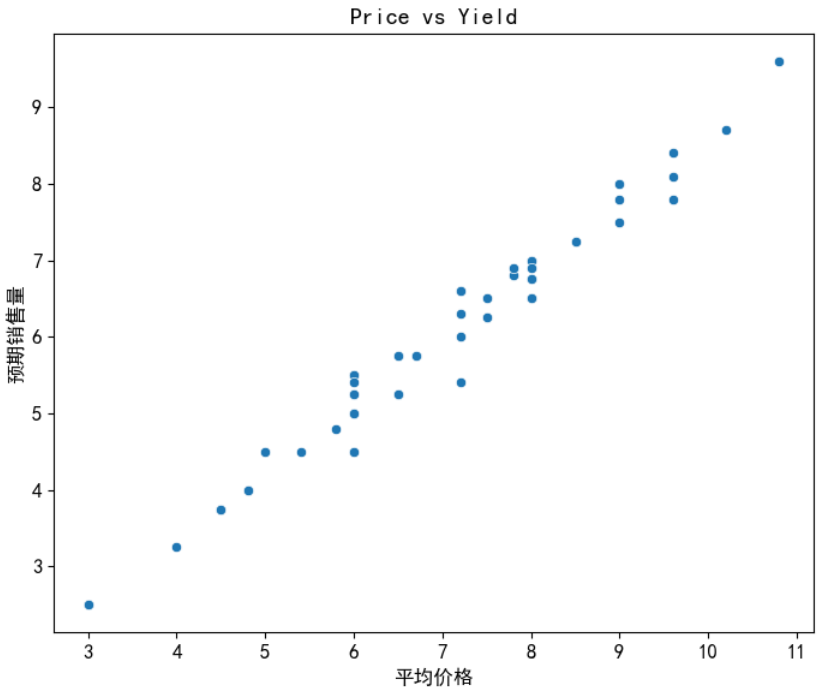

图 8 蔬菜的预期销售量与平均价格的散点图

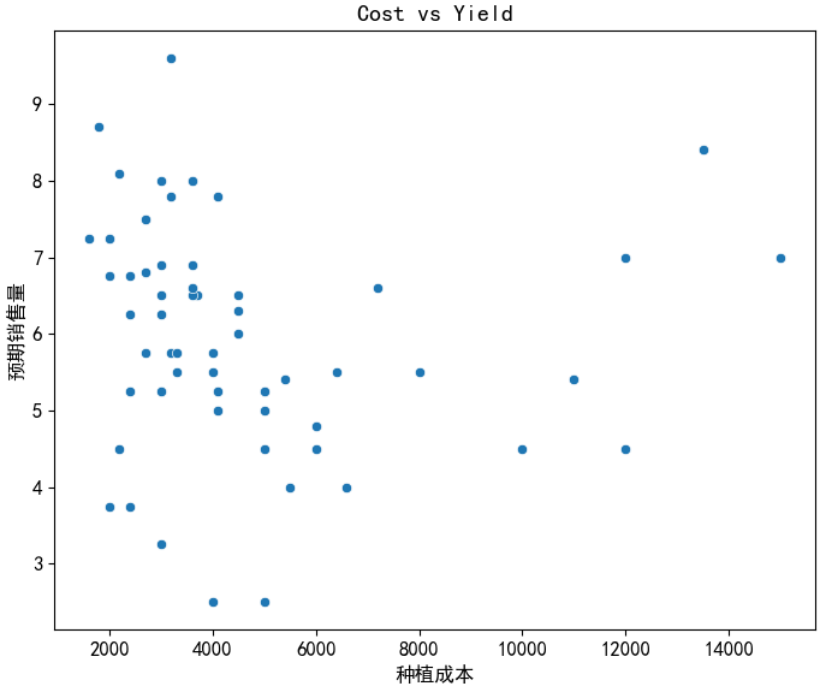

图 9 蔬菜的预期销售量与种植成本的散点图

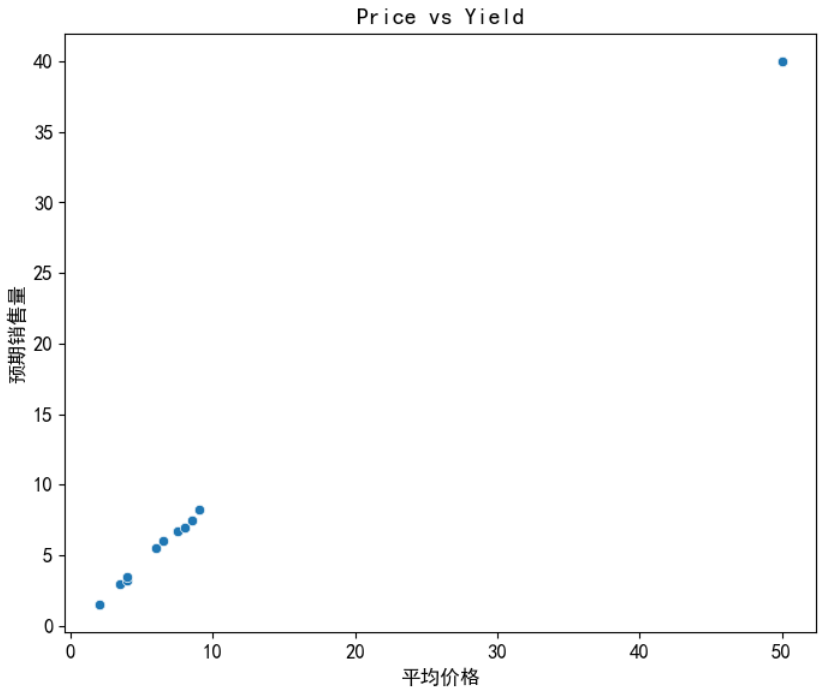

图 10 粮食的预期销售量与平均价格的散点图

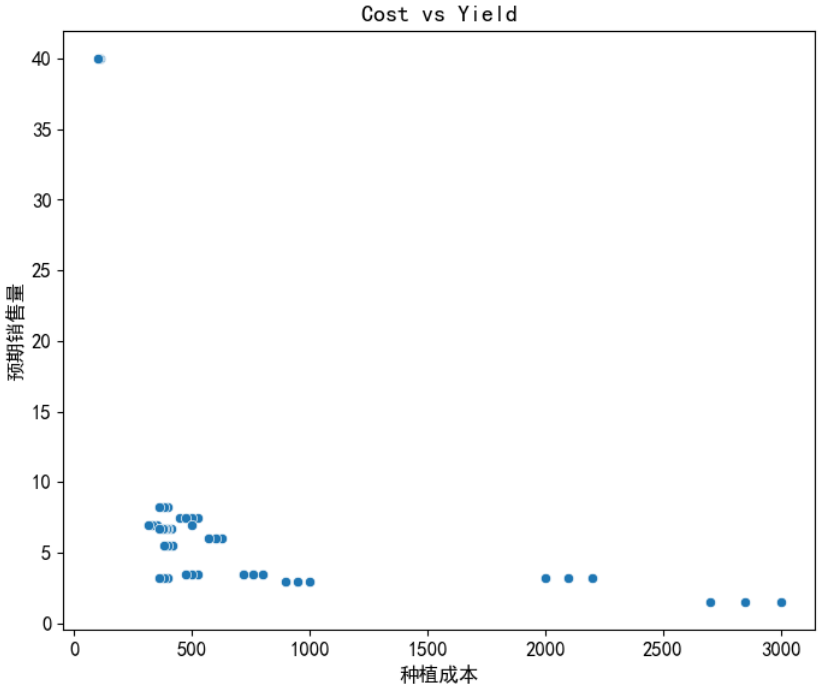

图 11 粮食的预期销售量与种植成本的散点图

以 2023 年数据为例，绘制作物的预期销售量、种植成本和平均价格的相关关系图。从图中可以看出对平均价格越高的作物，其预期销售量夜越高，而种植成本

越低的作物，其预期销售量则越高，即价格与预期销售量正相关，种植成本与预期销售量负相关。这似乎与常识有所违背，这是由模型假设导致的。由于本文假设 2023 年的作物不存在滞销，并把 2023 年的销售量作为预期销售量，因此实际上为价格与产量呈正相关，成本与产量呈负相关。

对于预期销售量、种植成本、单价三者间的相关性，由于题目只给了作物年在 2023 年的数据，因此为了得到具体作物预期销售量、种植与价格之间的相关性，可以在第二问的基础上通过蒙特卡洛模拟生成作物从 2024 年到 2030 年的预期销售量、种植成本与单价，在以此作为数据计算预期销售量、种植成本与单价三者之间的相关系数。

为了将相关性添加到约束条件中，我们可以通过引入价格-销售量、成本-销售量的相关性来控制农作物的需求上限，确保需求不会因为价格波动而出现不合理的增加或减少，最终保证销售量和价格之间的关系不会出现过度依赖。设作物 j 的价格-销售量相关系数为 ρ_{i1}，成本-销售量的相关系数为 ρ_{i2}，并引入价格调整因子 γ，与成本调整因子 β，则作物的总产量可做如下限制

$$\sum_{s=1}^{2}\sum_{i=1}^{54} M_{ijys} * S_{ijys} \leq d_{jy} - \gamma * P_{jys} * \rho_{i1}$$

$$\sum_{s=1}^{2}\sum_{i=1}^{54} M_{ijys} * S_{ijys} \leq d_{jy} - \beta * C_{ijys} * \rho_{i2}$$

- **替代与互补性分析**

替代性与互补性可以理解为两种商品之间的竞争与促进关系。对于不同作物之间的替代性与互补性，在经济学中通常用交叉价格弹性来衡量，交叉价格弹性 E_{ij} 表示商品 i 的需求量 Q_i 对商品 j 的价格 P_j 的变动的敏感程度。可用以下公式计算 [3]

$$E_{ij} = \frac{\Delta Q_i / Q_i}{\Delta P_j / P_j} \tag{10}$$

式中 Q_i 表示商品 i 的需求量，P_j 表示商品 j 的销售价格。当 E_{ij}>0 时，表示当商品 j 的价格上升时，商品 i 的需求量增加。也就是说，这两个商品是替代品. 并且 E_{ij} 越接近 1，表示这两个商品的替代性越强。当 E_{ij}<0 时，表示商品 j 的价格上升会导致商品 i 的需求量减少。这两个商品是互补品。例如，汽油和汽车就是互补品，汽油价格上升，汽车的需求可能会减少。

作物 j_1 与作物 j_2 的在第 y 年的交叉价格弹性 $E_{j_1 j_2 y}$ 同样可以用蒙特卡洛模拟的方法来计算，通过随机生成作物 i_1 与作物 I_2 各年的预期销售量 $d_{j_1 y}$、$d_{j_2 y}$ 与销售单价 $P_{j_1 y}$、$P_{j_2 y}$，再通过以下公式计算

$$E_{j_1 j_2 y} = \frac{\Delta d_{j_1 y} / d_{j_1 y}}{\Delta P_{j_2 y} / P_{j_2 y}} \tag{11}$$

式中 $\Delta d_{j_1y}=d_{j_1y}-d_{j_1y-1}$,$\Delta P_{j_2y}=P_{j_2y}-P_{j_2y-1}$, 为了将替代与互补性添加到限制条件中，可借助以下两个事实，即两种相互竞争的作物二者的种植总面积不能过大，而对于两种相互促进的作物要使二者的总面积尽量大。因此可添加以下两个限制条件

$$\begin{cases} Sij_1ys+S_{ij_2ys}\leq a_i & 若E_{j_1j_2y}\geq 0 \\ Sij_1ys\leq S_{ij_2ys} & 若E_{j_1j_2y}\leq 0 \end{cases} \tag{12}$$

5.3.2 模型建立与求解

由于题设只给了 2023 年的数据，因此首先可通过解问题二的模型，将该解所在的情形作为历史数据从而计算预期销售量-种植成本，预期销售量-作物价格的相关系数矩阵以及不同作物之间的交叉价格弹性系数，并以此作为依据进一步限制问题二中的模型从而进行求解。

- **模型建立**

设问题二中的模型为 $\mathscr{M}_0$, 以此得到的解为 $\mathscr{X}_0$, 由该解得到的预期销售量-种植成本相关系数矩阵、预期销售量-作物价格相关系数矩阵、交叉价格弹性系数矩阵分别为 $\mathscr{A}_0$、$\mathscr{B}_0$、$\mathscr{E}_0$, 并以此限制 $\mathscr{M}_0$，得到 $\mathscr{M}_1=\mathscr{M}(\mathscr{A}_0,\mathscr{B}_0\ \mathscr{E}_0)$, 模型 $\mathscr{M}_1$ 的数学表达式为

$$\max \quad Q=\sum_{y=1}^{6}\sum_{s=1}^{2}\sum_{i=1}^{54}\sum_{j=1}^{40}S'_{ijys}*(M_{ijys}*P_{jys}-C_{ijys})-f(w_{iy})*0.5*P_{jys}$$

$$\text{s.t.}\ \begin{cases} 0\leq\sum_{j=1}^{40}S_{ijys}\leq a_i & s=1,2 \quad j=1,2,\cdots,54 \\ & y=1,2\cdots,6 \\ a_i*5\%\leq S_{ijys} & s=1,2 \quad j=1,2,\cdots,54 \\ & i=1,2\cdots,40 \\ & y=1,2\cdots,6 \\ d_j\geq\sum_{s=1}^{2}\sum_{j=1}^{54}M_{ijys}*S_{ijs} & i=1,2\cdots,40 \quad y=1,2\cdots,6 \\ S_{ijys}+S_{ijy+1s}\leq a_i & s=1,2 \quad j=1,2,\cdots,54 \\ & i=1,2,\cdots,40 \\ & y=1,2,\cdots,5 \\ S_{ijy1}+S_{ijy2}\leq a_i & j=1,2,\cdots,54 \\ & i=1,2,\cdots,40 \\ & y=1,2,\cdots,5 \\ Sij_1ys+S_{ij_2ys}\leq a_i & 若E_{j_1j_2y}\geq 0 \\ Sij_1ys\leq S_{ij_2ys} & 若E_{j_1j_2y}\leq 0 \\ \sum_{s=1}^{2}\sum_{i=1}^{54}M_{ijys}*S_{ijys}\leq d_{jy}-\gamma*P_{jys}*\rho_{i1} & j=1,2,\cdots,40 \quad y=1,2,\cdots,6 \\ \sum_{s=1}^{2}\sum_{i=1}^{54}M_{ijys}*S_{ijys}\leq d_{jy}-\beta*C_{ijys}*\rho_{i2} & j=1,2,\cdots,40 \quad y=1,2,\cdots,6 \end{cases} \tag{13}$$

并由模型 $\mathscr{M}_1$ 解得 $\mathscr{X}_1$。

- **通过迭代优化模型**

  为了保证模型的稳健性与最优性，可通过每次得到的解所在的情形更新预期销售量-种植成本，预期销售量-作物价格的相关系数矩阵以及不同作物之间的交叉价格弹性系数，并再次求解。不断迭代该过程，直到种植方案的利润收敛，从而得到最终解。

  由模型 $\mathscr{M}_i$ 解得 $\mathscr{X}_i$ 以及 $\mathscr{A}_i$、$\mathscr{B}_i$、$\mathscr{E}_i$，并由此得到模型 $\mathscr{M}_{i+1} = \mathscr{M}(\mathscr{A}_i, \mathscr{B}_i\, \mathscr{E}_i)$，设该模型的解为 $\mathscr{X}_{i+1}$。假设第 i 次迭代得到的总利润为 Q_i，当 $|Q_i - Q_{i+1}| \leq \varepsilon$ 时停止迭代，并取 $\mathscr{X}_{i+1}$ 为最终的解。但由于无法证明 Q_i 是否收敛，因此对于该方案只提出思路，并不具体求解。

5.3.3 问题二与问题三的结果对比分析

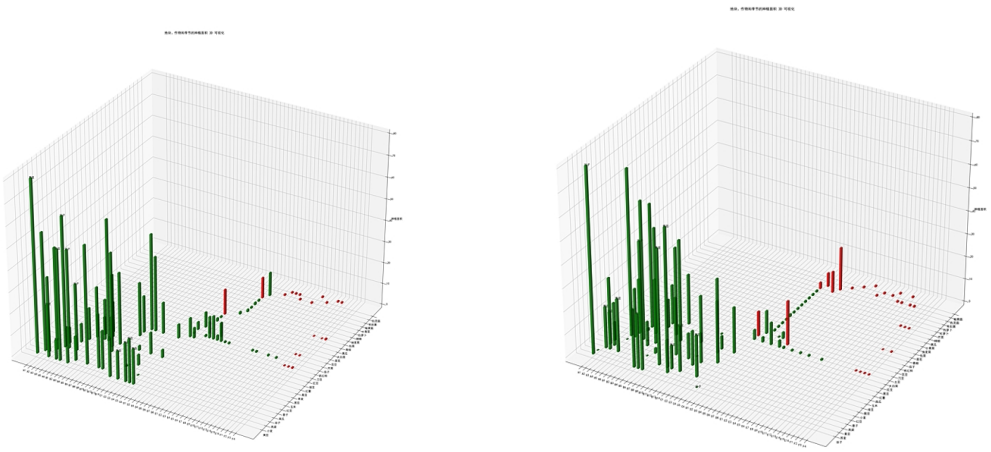

图 12 Q1 与 Q2 结果对比，左图为 Q1，右图为 Q2（2024 年）

通过对比两种情形下 2024 年的种植方案来看，在加入替代性、互补性和相关性等因素后，模型的预测结果略有不同，种植方案中更接近于实际生活。

六、模型的评价

6.1 模型优点

1. 在问题一中对数据信息中的现实问题以及自然规律进行了详细的分析，能够较为准确的反映现实，为实际的生产提供助力。

2. 在问题二中则将模型更一步的带入现实，采用蒙特卡洛模拟的思想，通过概率统计知识模拟实际的农业生产中出现的各种不确定性因素，进一步提高了方案的准

确性与稳健性。

3. 在问题三中通过动态规划和多目标规划思想，对问题二中的模型进一步扩展，使模型的准确性与稳健性都得到了进步的提高。

6.2 模型缺点

1. 由于限制条件过多，模型求解过于困难。

2. 问题二中直接假设预期销售量、亩产量、种植成本和单价的变化率符合正态分布仍有待商榷。这里只是因为正态分布是生活中最常见的分布，但实际情况下亩产量、销售价格等因素还受到不同的因素的制约。

3. 结合具体实际，我们在这个农作物种植领域上还提出一些影响决策的因素，例如作物的密集种植虽然一定程度上方便了管理，但是会提高作物疾病传染的风险，在一定程度上会降低作物的亩产量，于是考虑不同作物混合种植，降低感病组织的空间密度，在一定面积上减少感病组织量 [5]。

七、进一步的计划

1. 基于我们模型不够精准，在综合考虑实际情况，理清各种风险和收益方面做得仍然有所欠缺，并且时间比较仓促的情况下。我们认为后续可以展开进一步的讨论和模型的改良，我们考虑使用粒子群算法去模拟决策产生的过程。

2. 粒子群优化算法与蚁群算法（Ant Colony Optimization,ACO）类似，也是一种基于群体智能的优化算法，即模拟鸟群觅食的过程，而其功能与遗传算法（Genetic Algorithm,GA）非常相似 [4]。我们考虑，在这个算法中，我们把一种作物自己的种植方案视为一个初始粒子，而沿用相对简单的第一问中的决策方案得到初始粒子群，求解每个粒子群的个体极值（对应单个作物的最优种植方案）和粒子群的全局极值（对应全体种植方案的最优解）。

3. 而在迭代过程中，将实现单个作物或者全体利益最大化视为“鸟群的食物”，外界的约束条件则视为风力对鸟群移动的限制，不同粒子之间的相互作用，则反应不同作物之间替代、互补和相关性。在此基础上，给出每个粒子一个初始速度（即种植方案的大致优化方向），粒子群本身的位置变动即视为全局种植方案的变动。最后，当鸟群某一刻的位置覆盖的“食物范围”达到输入的预期收益时停止迭代。

八、参考文献

[1] 朱春江, 唐德善. 基于线性规划模型的农业种植业结构优化研究 [J]. 安徽农业科学,2006,(12):2623-2624.

[2] 杨衡. 蒙特卡罗模拟优化与风险决策分析的应用研究 [D]. 天津大学,2004.

[3] 胡睿乐, 赵梦涛. 基于需求交叉价格弹性的天然气市场地位分析 [J]. 煤气与热力,2020,40(07):37-39+46.DOI:10.13608/j.cnki.1000-4416.2020.07.020.

[4] 崔长彩, 李兵, 张认成. 粒子群优化算法 [J]. 华侨大学学报: 自然科学版, 2006, 27(4): 343-347.

[5] 徐学荣, 李宏宇, 林奇英, 等. 作物混合种植布局模型研究 [J]. 农业技术经济, 2002 (2): 8-10.

[6] Filippi C, Mansini R, Stevanato E. Mixed integer linear programming models for optimal crop selection[J]. Computers Operations Research, 2017, 81: 26-39.

[7] Dury J, Schaller N, Garcia F, et al. Models to support cropping plan and crop rotation decisions. A review[J]. Agronomy for sustainable development, 2012, 32: 567-580.

[8] 吕振肃, 侯志荣. 自适应变异的粒子群优化算法 [J]. 电子学报, 2004, 32(3): 416-420.

[9] Bhatia M, Rana A. A mathematical approach to optimize crop allocation–A linear programming model[J]. Int. J. Des. Nat. Ecodynamics, 2020, 15(2): 245-252.

[10] 杨丽英, 吴成东, 韩建达, 等. 多目标追逐问题的一种混合整数线性规划解 [J]. 机械工程学报, 2008, 44(10): 51-59.

[11] Fowler K R, Jenkins E W, Ostrove C, et al. A decision making framework with MODFLOW-FMP2 via optimization: Determining trade-offs in crop selection[J]. Environmental Modelling Software, 2015, 69: 280-291.

附录

附录 1：支撑材料的文件列表

文件名	说明
prob1.ipynb	问题一模型建立与求解代码
prob2.ipynb	问题二模型求解求解代码
prob2.ipynb	问题三模型求解求解代码
visual.ipynb	结果可视化
write_to_excel.ipynb	将结果写入 excel 表格
crop_planting_pro1_1_2024.svg	情形一的结果图
crop_planting_pro12_2024.svg	情形二的结果图
crop_planting_pro2_2024.svg	问题二的结果图
crop_planting_pro3_2024.svg	问题三的结果图
que3_analesis.py	计算作物间的替代与互补性
总相关性分析.py	计算预期销售量与单价、成本的相关性
main.py	绘制预期销售量的变化
crop_commd.xlsx	2023 年各作物的需求量
elasticity_matrix.xlsx	交叉价格弹性矩阵
result1_1.xlsx	问题一情形一的最终结果
result1_2.xlsx	问题一情形二的最终结果
result2.xlsx	问题二的最终结果
参考文献 (文件夹)	

附录 2：初始化代码和数据处理代码

```
#数据预处理
import pandas as pd
# 读取 Excel 文件
file1_path = r"附件1.xlsx"
file2_path = r"附件2.xlsx"

# 读取所有sheet
land_data = pd.read_excel(file1_path, sheet_name='乡村的现有耕地') # 附件
    1中包含地块信息
land_crop_data = pd.read_excel(file1_path, sheet_name='乡村种植的农作物'
    ) # 附件1中种植信息

crop_planting_data = pd.read_excel(file2_path, sheet_name='2023年的农作物
    种植情况') # 附件2中2023年作物种植情况
```

```
crop_stats_data = pd.read_excel(file2_path, sheet_name='2023年统计的相关数据') # 附件2中2023年的相关数据

# 假设地块名称的前缀（如A、B）代表地块类型
# 定义地块前缀对应的地块类型映射（假设A代表干旱地，B代表水浇地）
land_type_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'
    # 如果有其他类型，继续在这里添加
}

# 从地块名称中提取前缀并映射到地块类型
land_data = land_data.drop('说明␣', axis=1)
crop_stats_data = crop_stats_data.drop(['序号', '作物编号'], axis=1)
land_data['地块前缀'] = land_data['地块名称'].str[0] # 提取第一个字符作为前缀
land_data['地块类型'] = land_data['地块前缀'].map(land_type_mapping) # 映射到地块类型

# 合并地块信息和种植信息（基于地块类型）
# 首先，地块类型决定作物种类，因此我们根据地块类型合并
merged_data = land_data.merge(crop_stats_data, on='地块类型', how='left')

# 根据种植季次生成季节限制
def generate_season_restriction(row):
    # 如果种植季次是"单季"，季节限制设为1
    if row['种植季次'] == '单季':
        row['季节限制'] = 1
        row['第一季单价(元)'] = row['销售单价/(元/斤)']
        row['第二季单价(元)'] = 0
    # 如果种植季次是"第一季"或"第二季"，季节限制设为2（表示可种两季）
    elif row['种植季次'] in ['第一季', '第二季']:
        row['季节限制'] = 2
```

```
        if row['种植季次'] == '第一季':
            row['第一季单价(元)'] = row['销售单价/(元/斤)']
        else:
            row['第二季单价(元)'] = row['销售单价/(元/斤)']

    return row

# 应用生成季节限制的逻辑
merged_data = merged_data.apply(generate_season_restriction, axis=1)
merged_data =merged_data.fillna(0)
df=merged_data
df['第一季最小单价(元)']=df['第一季单价(元)'].str.split('-', expand=True)[0].
    astype(float, errors='ignore')
df['第一季最大单价(元)']=df['第一季单价(元)'].str.split('-', expand=True)[1].
    astype(float, errors='ignore')
df['第二季最小单价(元)']=df['第二季单价(元)'].str.split('-', expand=True)[0].
    astype(float, errors='ignore')
df['第二季最大单价(元)']=df['第二季单价(元)'].str.split('-', expand=True)[1].
    astype(float, errors='ignore')
df=df.fillna(0)

# 选择我们感兴趣的列
#final_data = merged_data[['地块名称', '作物名称', '地块类型', '亩产量(斤)', '
    种植成本/(元/亩)', '第一季单价(元)', '第二季单价(元)', '季节限制']]

crop_recom=pd.read_excel(r"..\crop_commd.xlsx")

import pulp as lp

# 将DataFrame转化为优化模型需要的字典格式
data = {}
for idx, row in df.iterrows():
    data[(row['地块名称'], row['作物名称'])] = (
        row['地块类型'],
        row['亩产量/斤'], # 转化为吨
        row['种植成本/(元/亩)'],
        (row['第一季最小单价(元)']+row['第一季最大单价(元)'])/2,
```

```
            (row['第二季最小单价(元)']+row['第二季最大单价(元)'])/2,
            row['季节限制']
        )

# 地块和作物名称
land_plots = df['地块名称'].unique().tolist()
crops = df['作物名称'].unique().tolist()
land_area = df.set_index('地块名称')['地块面积/亩'].to_dict()
# 定义豆类作物的列表
legumes = ['黄豆', '黑豆','红豆','绿豆','爬豆','豇豆','刀豆','芸豆',] # 豆类作物
years = [2024, 2025, 2026, 2027, 2028, 2029, 2030] # 年份
#surplus = lp.LpVariable.dicts("surplus", ((j, y) for j in crops for y in years),
    lowBound=0, cat="Continuous")

# 假设一些市场需求（可以从文件中读取或另行定义）
crop_demand = {}
for idx, row in crop_recom.iterrows():
    crop_demand[row['作物名称']] = row['总产量']

# 辅助函数：判断是否为有效的地块和作物的季节种植组合
def is_valid_plot(i, j, s):
    # 检查地块类型和作物季节限制
    plot_type = data[(i, j)][0] # 地块类型
    if plot_type in ['平旱地', '梯田', '山坡地'] and s == 2:
        return False # 平旱地、梯田和山坡地只能种一季
    if plot_type == '水浇地' and s == 1 and j == '蔬菜':
        return False # 水浇地第一季不能种蔬菜
    if plot_type == '普通大棚' and j == '水稻':
        return False # 普通大棚不能种水稻
    return True

#问题一：情形一

import pulp as lp
# 决策变量：x[i][j][s][y] 表示在第 i 块地上种植第 j 种作物的面积，s 表示季节
    (1 或 2)，y 表示年份
x = lp.LpVariable.dicts(
```

```
    "x",
    ((i, j, s, y) for i in land_plots for j in crops for s in [1, 2] for y in years
     if (i, j) in data and (s == 1 or data[(i, j)][5] == 2) and data[(i, j)][3 if s
         == 1 else 4] != 0),
    lowBound=0, cat="Continuous"
)

#创建问题
prob = lp.LpProblem("Crop_Planting_Optimization", lp.LpMaximize)

# 对于每个三年时间窗口，确保豆类作物的种植面积达到总地块面积的 100%

#第一阶段目标优化
# 确保每块地在3年内的所有面积都至少种植一次豆类
for i in land_plots:
    for y in range(2023, max(years) - 4): # 遍历每个连续的三年窗口
        prob += lp.lpSum(
            x[(i, j, s, y+k)] for j in legumes for s in [1, 2] for k in range(3)
            if (i, j, s, y+k) in x and (s == 1 or data[(i, j)][5] == 2) # 只遍历允
                许的季节
        ) >= land_area[i], f"Legume_Rotation_{i}_{y}" # 设置为总面积

# 定义 waste 变量，表示超过产量上限的部分
waste = lp.LpVariable.dicts("waste", ((j, y) for j in crops for y in years),
    lowBound=0, cat="Continuous")
min_area_thresholdd=0

# 去除作物名称中的空格
for j in crops:
    j_clean = j.strip() # 去掉作物名称前后的空格
    for y in years:
        for i in land_plots:
            for s in [1, 2]: # 添加季节 s 来确保约束名称唯一
                # 添加条件到 for 循环中，确保只处理符合条件的组合
                prob += lp.lpSum(
                    x.get((i, j_clean, s, y), 0) * data.get((i, j_clean), [0, 0])[1]
                    for i in land_plots if (i, j_clean) in data and (s == 1 or
```

```
                    data[(i, j_clean)][5] == 2)
                ) <= crop_demand.get(j_clean, 0) + waste.get((j_clean, y), 0)
                    , \
                f"Crop_Yield_Limit_With_Waste_{i}_{j_clean}_{y}_S{s}"
                    # 加上季节 s，确保唯一


#第二阶段目标优化
# 目标函数，扣除滞销部分
prob += lp.lpSum(
    (x[(i, j, s, y)] * data[(i, j)][1] * data[(i, j)][3 if s == 1 else 4] # 正常销售的收
        益
     - data[(i, j)][2] * x[(i, j, s, y)] # 减去种植成本
     - waste[j, y] * data[(i, j)][3 if s == 1 else 4]) # 滞销部分不产生收益
    for i in land_plots for j in crops for s in [1, 2] for y in years
    if (i, j) in data
    and (s == 1 or data[(i, j)][5] == 2)
    and data[(i, j)][3 if s == 1 else 4] != 0
    and is_valid_plot(i, j, s)
), "Maximize_Total_Profit_With_Waste"


# 约束1：每块地的总种植面积不能超过可用面积
for i in land_plots:
    for y in years: # 遍历每个年份
        prob += lp.lpSum(x[(i, j, s, y)] for j in crops for s in [1, 2]
                         if (i, j) in data and (s == 1 or data[(i, j)][5] == 2)
                         and (i, j, s, y) in x) <= land_area[i], f"
                             Land_Area_Constraint_{i}_{y}"


#约束条件2：种植不能太分散,方便管理
for j in crops:
    for i in land_plots:
        for y in years:
            # 根据地块面积动态调整最小种植面积
            if land_area[i] > 10: # 面积大于10亩的地块
```

```
                min_area_threshold = 0.1 * land_area[i] # 地块总面积的10%
            else: # 面积较小的地块
                min_area_threshold = 0.05 * land_area[i] # 地块总面积的5%

            # 确保每个约束的名称唯一，添加季节(s)变量到约束名中
            for s in [1, 2]:
                prob += lp.lpSum(x[(i, j, s, y)] for s in [1, 2] if (i, j, s, y) in x)
                    >= min_area_thresholdd, \
                        f"Min_Concentrated_Area_Constraint_{i}_{j}_{y}
                            _S{s}"

#约束3：禁止同一地块同一年在两季内种植同一作物
for i in land_plots:
    for j in crops:
        for y in years: # 遍历每一年
            if (i, j, 1, y) in x and (i, j, 2, y) in x: # 检查变量是否存在
                prob += x[(i, j, 1, y)] + x[(i, j, 2, y)] <= land_area[i], f"
                    No_Same_Crop_Two_Seasons_{i}_{j}_Year_{y}"

#约束4：禁止连续两年种植同一种作物（防止重茬）
for i in land_plots:
    for j in crops:
        for y in years[:-1]: # 遍历年份，检查连续两年的种植情况
            # 第一季重茬限制
            if (i, j, 1, y) in x and (i, j, 1, y+1) in x:
                prob += x[(i, j, 1, y)] + x[(i, j, 1, y+1)] <= land_area[i], f"
                    No_Continuous_Planting_{i}_{j}_{y}_S1"
            # 第二季重茬限制
            if (i, j, 2, y) in x and (i, j, 2, y+1) in x:
                prob += x[(i, j, 2, y)] + x[(i, j, 2, y+1)] <= land_area[i], f"
                    No_Continuous_Planting_{i}_{j}_{y}_S2"


# 求解问题
prob.solve()

# 输出求解状态
```

```
print(f"状态:␣{lp.LpStatus[prob.status]}")

# 按年份输出种植决策和当年利润
for y in years:
    print(f"\n======情形一：{y}␣年的种植决策␣=====")

    yearly_profit = 0 # 初始化当年的利润

    for i in land_plots:
        for j in crops:
            for s in [1, 2]:
                if (i, j, s, y) in x: # 确保变量存在
                    planted_area = x[(i, j, s, y)].varValue
                    if planted_area > 0: # 只显示种植面积大于0的作物
                        print(f"地块␣{i}␣第{s}季␣种植␣{j}␣{planted_area:.2f}
                            ␣亩")
                        # 计算该作物的利润并累加
                        yearly_profit += (
                            planted_area * data[(i, j)][1] * data[(i, j)][3 if s ==
                                1 else 4] # 种植面积 * 产量 * 单价
                            - data[(i, j)][2] * planted_area # 减去种植成本
                        )

    # 输出当年的利润
    print(f"\n{y}␣年的利润:␣{yearly_profit:.2f}␣元")


#问题一：情形二
import pulp as lp
# 决策变量：x[i][j][s][y] 表示在第 i 块地上种植第 j 种作物的面积，s 表示季节
    (1 或 2)，y 表示年份
x = lp.LpVariable.dicts(
    "x",
    ((i, j, s, y) for i in land_plots for j in crops for s in [1, 2] for y in years
     if (i, j) in data and (s == 1 or data[(i, j)][5] == 2) and data[(i, j)][3 if s
         == 1 else 4] != 0),
    lowBound=0, cat="Continuous"
```

```
)

#创建问题
prob = lp.LpProblem("Crop_Planting_Optimization", lp.LpMaximize)
min_area_thresholdd=0



#第一阶段目标优化
# 确保每块地在3年内的所有面积都至少种植一次豆类
for i in land_plots:
    for y in range(2023, max(years) - 4): # 遍历每个连续的三年窗口
        prob += lp.lpSum(
            x[(i, j, s, y+k)] for j in legumes for s in [1, 2] for k in range(3)
            if (i, j, s, y+k) in x and (s == 1 or data[(i, j)][5] == 2) # 只遍历允
                许的季节
        ) >= land_area[i], f"Legume_Rotation_{i}_{y}" # 设置为总面积

for i in land_plots:
    for y in years: # 遍历每个年份
        prob += lp.lpSum(x[(i, j, s, y)] for j in crops for s in [1, 2]
                         if (i, j) in data and (s == 1 or data[(i, j)][5] == 2)
                         and (i, j, s, y) in x) <= land_area[i], f"
                             Land_Area_Constraint_{i}_{y}"



# 定义 waste 变量，去掉作物名称的空格
waste = lp.LpVariable.dicts("waste", ((j.strip(), y) for j in crops for y in years),
    lowBound=0, cat="Continuous")
# 去除作物名称中的空格
for j in crops:
    j_clean = j.strip() # 去掉作物名称前后的空格
    for y in years:
        for i in land_plots:
            for s in [1, 2]: # 添加季节 s 来确保约束名称唯一
                # 添加条件到 for 循环中，确保只处理符合条件的组合
                prob += lp.lpSum(
                    x.get((i, j_clean, s, y), 0) * data.get((i, j_clean), [0, 0])[1]
```

```
                    for i in land_plots if (i, j_clean) in data and (s == 1 or
                        data[(i, j_clean)][5] == 2)
                ) <= crop_demand.get(j_clean, 0) + waste.get((j_clean, y), 0)
                    , \
                f"Crop_Yield_Limit_With_Waste_{i}_{j_clean}_{y}_S{s}"
                    # 加上季节 s，确保唯一
#第二阶段目标优化
prob += lp.lpSum(
    (x[(i, j, s, y)] * data[(i, j)][1] * data[(i, j)][3 if s == 1 else 4] # 正常销售的收
        益
     - data[(i, j)][2] * x[(i, j, s, y)] # 减去种植成本
     - waste.get((j, y), 0) * 0.5 * data[(i, j)][3 if s == 1 else 4]) # 超过部分按
         50%价格出售
    for i in land_plots for j in crops for s in [1, 2] for y in years
    if (i, j) in data
    and (s == 1 or data[(i, j)][5] == 2)
    and data[(i, j)][3 if s == 1 else 4] != 0
    and is_valid_plot(i, j, s)
), "Maximize_Total_Profit_With_Discounted_Sales"

# 约束1：每块地的总种植面积不能超过可用面积
# 修改 Land_Area_Constraint 名称，添加季节 s
for i in land_plots:
    for y in years: # 遍历每个年份
        for s in [1, 2]: # 添加季节
            prob += lp.lpSum(x[(i, j, s, y)] for j in crops # 获取种植面积
                             if (i, j) in data and (s == 1 or data[(i, j)][5] ==
                                 2)
                             and (i, j, s, y) in x) <= land_area[i], \
                             f"Land_Area_Constraint_{i}_{y}_S{s}"

#约束条件2：种植不能太分散
for j in crops:
    for i in land_plots:
        for y in years:
            # 根据地块面积动态调整最小种植面积
            if land_area[i] > 10: # 面积大于10亩的地块
```

```
                min_area_threshold = 0.1 * land_area[i] # 地块总面积的10%
            else: # 面积较小的地块
                min_area_threshold = 0.05 * land_area[i] # 地块总面积的5%

            # 确保每个约束的名称唯一，添加季节(s)变量到约束名中
            for s in [1, 2]:
                prob += lp.lpSum(x[(i, j, s, y)] for s in [1, 2] if (i, j, s, y) in x)
                    >= min_area_thresholdd, \
                        f"Min_Concentrated_Area_Constraint_{i}_{j}_{y}
                            _S{s}"

#约束3：禁止同一年内同一地块种植同一作物在两季
for i in land_plots:
    for j in crops:
        for y in years: # 遍历每一年
            if (i, j, 1, y) in x and (i, j, 2, y) in x: # 检查变量是否存在
                prob += x[(i, j, 1, y)] + x[(i, j, 2, y)] <= land_area[i], f"
                    No_Same_Crop_Two_Seasons_{i}_{j}_Year_{y}"

#约束4：禁止连续两年种植同一种作物（防止重茬）
for i in land_plots:
    for j in crops:
        for y in years[:-1]: # 遍历年份，检查连续两年的种植情况
            # 第一季重茬限制
            if (i, j, 1, y) in x and (i, j, 1, y+1) in x:
                prob += x[(i, j, 1, y)] + x[(i, j, 1, y+1)] <= land_area[i], f"
                    No_Continuous_Planting_{i}_{j}_{y}_S1"
            # 第二季重茬限制
            if (i, j, 2, y) in x and (i, j, 2, y+1) in x:
                prob += x[(i, j, 2, y)] + x[(i, j, 2, y+1)] <= land_area[i], f"
                    No_Continuous_Planting_{i}_{j}_{y}_S2"


# 求解问题
prob.solve()

# 输出求解状态
```

```
print(f"状态:␣{lp.LpStatus[prob.status]}")

# 按年份输出种植决策和当年利润
for y in years:
    print(f"\n=====␣情形二: {y}␣年的种植决策␣=====")

    yearly_profit = 0 # 初始化当年的利润

    for i in land_plots:
        for j in crops:
            for s in [1, 2]:
                if (i, j, s, y) in x: # 确保变量存在
                    planted_area = x[(i, j, s, y)].varValue
                    if planted_area > 0: # 只显示种植面积大于0的作物
                        print(f"地块␣{i}␣第{s}季␣种植␣{j}␣{planted_area:.2f}
                            ␣亩")
                        # 计算该作物的利润并累加
                        yearly_profit += (
                            planted_area * data[(i, j)][1] * data[(i, j)][3 if s ==
                                1 else 4] # 种植面积 * 产量 * 单价
                            - data[(i, j)][2] * planted_area # 减去种植成本
                        )

    # 输出当年的利润
    print(f"\n{y}␣年的利润:␣{yearly_profit:.2f}␣元")

#将结果写入文件中
# 打开一个记事本文件进行写入（如果文件不存在会自动创建）
with open("result_21.txt", "w", encoding="utf-8") as file:
    for y in years:
        # 写入年份的标题
        file.write(f"\n=====␣{y}␣年的种植决策␣=====\n")

        yearly_profit = 0 # 初始化当年的利润

        for i in land_plots:
            for j in crops:
```

```
                for s in [1, 2]:
                    if (i, j, s, y) in x: # 确保变量存在
                        planted_area = x[(i, j, s, y)].varValue
                        if planted_area > 0: # 只记录种植面积大于0的作物
                            # 写入每个地块、作物的种植情况
                            file.write(f"地块 {i} 第{s}季 种植 {j} {
                                planted_area:.2f} 亩\n")
                            # 计算该作物的利润并累加
                            yearly_profit += (
                                planted_area * data[(i, j)][1] * data[(i, j)][3 if s
                                    == 1 else 4] # 种植面积 * 产量 * 单价
                                - data[(i, j)][2] * planted_area # 减去种植成
                                    本
                            )

        # 写入当年的利润
        file.write(f"\n{y} 年的利润: {yearly_profit:.2f} 元\n")

#问题二：决策函数
#决策函数
def decision():
    import pulp as lp
    import numpy as np

    # 定义模拟的次数
    num_simulations = 10

    # 存储每次模拟的总利润
    profits = {}
    decision={}
    detailed_info={}

    for sim in range(num_simulations):
        print(f"Simulation {sim + 1}/{num_simulations}")

        # 1. 模拟不确定性参数的波动
        # 销售量的变化
```

```
future_crop_demand = {}
for j in crops:
    if j in ['小麦', '玉米']: # 对小麦和玉米，年增长5%~10%
        growth_rate = np.random.uniform(0.05, 0.10)
        for y in years:
            future_crop_demand[j, y] = crop_demand[j] * (1 +
                growth_rate)**(y - 2023)
    else: # 对其他作物 ±5%的变化
        for y in years:

            future_crop_demand[j, y] = crop_demand[j] * np.random.
                uniform(0.95, 1.05)

# 亩产量的变化，每年±10%的变化
future_yield = {}
for i, j in data:
    for y in years:
        future_yield[i, j, y] = data[(i, j)][1] * np.random.uniform(0.9,
            1.1)

# 种植成本年增长5%
future_costs = {}
for i, j in data:
    for y in years:
        future_costs[i, j, y] = data[(i, j)][2] * (1.05)**(y - 2023)

# 销售价格变化
future_prices = {}
for i, j in data:
    for y in years:
        if j in ['小麦', '玉米']: # 粮食价格基本稳定
            future_prices[i, j, 1, y] = data[(i, j)][3]
            future_prices[i, j, 2, y] = data[(i, j)][4]
        elif j in vegetables: # 蔬菜类每年增长5%
            future_prices[i, j, 1, y] = data[(i, j)][3] * (1.05)**(y - 2023)
            future_prices[i, j, 2, y] = data[(i, j)][4] * (1.05)**(y - 2023)
        elif j in fungi: # 食用菌价格每年下降1%−5%
```

```
                decline_rate = np.random.uniform(0.01, 0.05)
                future_prices[i, j, 1, y] = data[(i, j)][3] * (1 - decline_rate)
                    **(y - 2023)
                future_prices[i, j, 2, y] = data[(i, j)][4] * (1 - decline_rate)
                    **(y - 2023)
            else: # 默认保持不变
                future_prices[i, j, 1, y] = data[(i, j)][3]
                future_prices[i, j, 2, y] = data[(i, j)][4]

# 2. 创建新的问题
prob = lp.LpProblem("Crop_Planting_Optimization", lp.LpMaximize)
min_area_thresholdd=0
# 决策变量：x[i][j][s][y] 表示在第 i 块地上种植第 j 种作物的面积，s 表
    示季节 (1 或 2)，y 表示年份
x = lp.LpVariable.dicts(
    "x",
    ((i, j, s, y) for i in land_plots for j in crops for s in [1, 2] for y in
        years
    if (i, j) in data and (s == 1 or data[(i, j)][5] == 2) and data[(i, j)][3
        if s == 1 else 4] != 0),
    lowBound=0, cat="Continuous")
#第一阶段目标优化
# 确保每块地在3年内的所有面积都至少种植一次豆类
for i in land_plots:
    for y in range(2023, max(years) - 4): # 遍历每个连续的三年窗口
        prob += lp.lpSum(
            x[(i, j, s, y+k)] for j in legumes for s in [1, 2] for k in range
                (3)
            if (i, j, s, y+k) in x and (s == 1 or data[(i, j)][5] == 2) #
                只遍历允许的季节
        ) >= land_area[i], f"Legume_Rotation_{i}_{y}" # 设置为总
            面积

for i in land_plots:
    for y in years: # 遍历每个年份
        prob += lp.lpSum(x[(i, j, s, y)] for j in crops for s in [1, 2]
                         if (i, j) in data and (s == 1 or data[(i, j)][5]
```

```
                == 2)
            and (i, j, s, y) in x) <= land_area[i], f"
                Land_Area_Constraint_{i}_{y}"

# 定义 waste 变量，去掉作物名称的空格
waste = lp.LpVariable.dicts("waste", ((j.strip(), y) for j in crops for y in
    years), lowBound=0, cat="Continuous")
# 去除作物名称中的空格
for j in crops:
    j_clean = j.strip() # 去掉作物名称前后的空格
    for y in years:
        for i in land_plots:
            for s in [1, 2]: # 添加季节 s 来确保约束名称唯一
                # 添加条件到 for 循环中，确保只处理符合条件的组合
                prob += lp.lpSum(
                    x.get((i, j_clean, s, y), 0) * data.get((i, j_clean),
                        [0, 0])[1]
                    for i in land_plots if (i, j_clean) in data and (s ==
                        1 or data[(i, j_clean)][5] == 2)
                ) <= crop_demand.get(j_clean, 0) + waste.get((
                    j_clean, y), 0), \
                f"Crop_Yield_Limit_With_Waste_{i}_{j_clean}_{y
                    }_S{s}" # 加上季节 s，确保唯一

# 目标函数：每年收益最大化
prob += lp.lpSum(
    (x.get((i, j, s, y), 0) * future_yield.get((i, j, y), 0) * future_prices.
        get((i, j, s, y), 0) # 销售收益
    - future_costs.get((i, j, y), 0) * x.get((i, j, s, y), 0) # 减去种植成
        本
    - waste.get((j, y), 0) * 0.5 * future_prices.get((i, j, s, y), 0)) # 超
        过部分按50%价格出售
    for i in land_plots for j in crops for s in [1, 2] for y in years
    if (i, j) in data and (s == 1 or data[(i, j)][5] == 2) # 只允许合法种
        植
    and is_valid_plot(i, j, s)
), "Maximize_Total_Profit_With_Discounted_Sales"
```

```
# 约束1：每块地的总种植面积不能超过可用面积
# 修改 Land_Area_Constraint 名称，添加季节 s
for i in land_plots:
    for y in years: # 遍历每个年份
        for s in [1, 2]: # 添加季节
            prob += lp.lpSum(x[(i, j, s, y)] for j in crops # 获取种植面
                积
                            if (i, j) in data and (s == 1 or data[(i, j)
                                ][5] == 2)
                            and (i, j, s, y) in x) <= land_area[i], \
                            f"Land_Area_Constraint_{i}_{y}_S{s}"

#约束条件2：种植不能太分散
for j in crops:
    for i in land_plots:
        for y in years:
            # 根据地块面积动态调整最小种植面积
            if land_area[i] > 10: # 面积大于10亩的地块
                min_area_threshold = 0.1 * land_area[i] # 地块总面
                    积的10%
            else: # 面积较小的地块
                min_area_threshold = 0.05 * land_area[i] # 地块总面
                    积的5%

            # 确保每个约束的名称唯一，添加季节(s)变量到约束名中
            for s in [1, 2]:
                prob += lp.lpSum(x[(i, j, s, y)] for s in [1, 2] if (i, j, s,
                    y) in x) >= min_area_thresholdd, \
                        f"Min_Concentrated_Area_Constraint_{i}_{
                            j}_{y}_S{s}"

#约束3：禁止同一年内同一地块种植同一作物在两季
for i in land_plots:
    for j in crops:
        for y in years: # 遍历每一年
```

```
                if (i, j, 1, y) in x and (i, j, 2, y) in x: # 检查变量是否存在
                    prob += x[(i, j, 1, y)] + x[(i, j, 2, y)] <= land_area[i],
                        f"No_Same_Crop_Two_Seasons_{i}_{j}_Year_{
                        y}"

#约束4：禁止连续两年种植同一种作物（防止重茬）
for i in land_plots:
    for j in crops:
        for y in years[:-1]: # 遍历年份，检查连续两年的种植情况
            # 第一季重茬限制
            if (i, j, 1, y) in x and (i, j, 1, y+1) in x:
                prob += x[(i, j, 1, y)] + x[(i, j, 1, y+1)] <= land_area[
                    i], f"No_Continuous_Planting_{i}_{j}_{y}_S1"
            # 第二季重茬限制
            if (i, j, 2, y) in x and (i, j, 2, y+1) in x:
                prob += x[(i, j, 2, y)] + x[(i, j, 2, y+1)] <= land_area[
                    i], f"No_Continuous_Planting_{i}_{j}_{y}_S2"


# 3. 求解问题
prob.solve()

# 输出求解状态
print(f"状态:␣{lp.LpStatus[prob.status]}")
yearly_profit = []
decision_yearly=[]

# 检查求解状态，并保存模拟的利润和决策
if lp.LpStatus[prob.status] == 'Optimal':
    yearly_profit = []
    decision_yearly = []
    detailed_yearly = [] # 存储详细信息

    for y in years:
        year_profit = 0
        year_detailed = []
```

```
                for i in land_plots:
                    for j in crops:
                        for s in [1, 2]:
                            if (i, j, s, y) in x:
                                planted_area = x[(i, j, s, y)].varValue
                                if planted_area > 0:
                                    # 计算利润
                                    year_profit += planted_area *
                                        future_yield[i, j, y] * future_prices[i, j,
                                         s, y]
                                    # 记录种植决策
                                    decision_yearly.append((i, j, s, y,
                                        planted_area))
                                    # 记录详细信息：成本、销售量、单价、
                                        亩产量、收益等
                                    cost = future_costs[i, j, y]
                                    #revenue = planted_area * future_yield[i,
                                         j, y] * future_prices[i, j, s, y]
                                    demand = future_crop_demand[j, y]
                                    yield_per_acre = future_yield[i, j, y] #
                                        记录亩产量

                                    year_detailed.append({
                                        '地块': i,
                                        '作物': j,
                                        '季节': s,
                                        '年份': y,
                                        '种植面积': round(planted_area, 1), #
                                            保留一位小数
                                        '亩成本': round(cost, 1), # 保留一位
                                            小数
                                        '亩产量': round(yield_per_acre, 1), #
                                            保留一位小数
                                        '销售价格': round(future_prices[i, j, s,
                                            y], 2),
                                        '销售量': round(demand, 1) # 保留一
                                            位小数
```

```
                })

                yearly_profit.append(year_profit)
                detailed_yearly.append(year_detailed)

            # 保存每次模拟的利润和决策
            profits[f"{sim + 1}"] = yearly_profit
            decision[f"{sim + 1}"] = decision_yearly
            detailed_info[f"{sim + 1}"] = detailed_yearly
        else:
            print(f"Simulation {sim + 1} did not find an optimal solution.")

    # 找出最小利润对应的模拟
    worst_simulation = min(profits, key=lambda k: sum(profits[k]))

    # 输出最差情况的种植策略和详细信息
    #print(f"最差利润对应的模拟: {worst_simulation}")
    #print(f"最差利润: {sum(profits[worst_simulation])}")
    #print(f"最差情况下的种植策略: {decision[worst_simulation]}")
    #print(f"最差情况下的详细信息: {detailed_info[worst_simulation]}")

    return detailed_info[worst_simulation],profits[worst_simulation]

#将结果写入csv
#将结果输出到csv
# 展开嵌套的列表字典结构
import pandas as pd
data=decision_info
flattened_data = []
for simulation in data:
    for record in simulation:
        flattened_data.append({
            '年份': record['年份'],
            '地块名': record['地块'],
            '季节': record['季节'],
            '作物名': record['作物'],
            '种植面积': record['种植面积']
```

```
        })

# 将数据转换为 DataFrame
df = pd.DataFrame(flattened_data)
#输出为csv文件
df.to_csv("../re222.csv")

#问题三:
#计算可替代性和互补性
def calculate_elasticity(detailed_data):
    import numpy as np
    import pandas as pd
    # 提取所有的作物名称
    crops = list(set([entry['作物'] for year_data in detailed_data for entry in
        year_data]))
    years = list(set([entry['年份'] for year_data in detailed_data for entry in
        year_data]))

    # 初始化弹性矩阵
    elasticity_matrix = pd.DataFrame(np.nan, index=crops, columns=crops)

    # 遍历作物组合，计算弹性
    for i, crop_i in enumerate(crops):
        for j, crop_j in enumerate(crops):
            if crop_i == crop_j:
                # 自身的替代性为0
                elasticity_matrix.loc[crop_i, crop_j] = 0
            else:
                # 获取每年作物 i 的需求量和作物 j 的价格
                crop_i_demand = {entry['年份']: entry['销售量'] for year_data
                    in detailed_data for entry in year_data if entry['作物'] ==
                    crop_i}
                crop_j_price = {entry['年份']: entry['销售价格'] for year_data
                    in detailed_data for entry in year_data if entry['作物'] ==
                    crop_j}

                # 计算跨年份的需求和价格变化
```

```
                delta_Q_i = []
                delta_P_j = []
                for y in sorted(years)[1:]:
                    if y in crop_i_demand and y-1 in crop_i_demand and y
                        in crop_j_price and y-1 in crop_j_price:
                        try:
                            dq = (crop_i_demand[y] - crop_i_demand[y-1])
                                / crop_i_demand[y-1]
                            dp = (crop_j_price[y] - crop_j_price[y-1]) /
                                crop_j_price[y-1]
                            if dp != 0:
                                delta_Q_i.append(dq)
                                delta_P_j.append(dp)
                        except ZeroDivisionError:
                            pass # 处理分母为0的情况

                # 如果有有效的 delta_Q_i 和 delta_P_j，则计算弹性
                if len(delta_Q_i) > 0 and len(delta_P_j) > 0:
                    elasticity = np.mean([dq/dp for dq, dp in zip(delta_Q_i,
                        delta_P_j)])
                    elasticity_matrix.loc[crop_i, crop_j] = round(elasticity, 2)
                else:
                    elasticity_matrix.loc[crop_i, crop_j] = np.nan # 没有数据
                        则标记为 NaN

    return elasticity_matrix
#计算相关性
def calculate_correlation_matrices(decision_info):
    import pandas as pd

    # 将嵌套列表转换为扁平化的列表
    flat_data = [item for sublist in decision_info for item in sublist]

    # 将数据转换为 DataFrame
    df = pd.DataFrame(flat_data)

    # 计算每种作物的销售量、销售价格和成本之间的相关性
```

```
    correlation_results = {}

    # 按照作物进行分组
    for crop, group in df.groupby('作物'):
        # 计算相关性矩阵
        correlation_matrix = group[['销售量', '销售价格', '亩成本']].corr()

        # 获取相关性值
        sales_price_corr = correlation_matrix.loc['销售量', '销售价格']
        sales_cost_corr = correlation_matrix.loc['销售量', '亩成本']
        price_cost_corr = correlation_matrix.loc['销售价格', '亩成本']

        # 将结果存储到字典中，处理可能的缺失值
        correlation_results[crop] = [
            round(sales_price_corr if not pd.isna(sales_price_corr) else 0, 2),
            round(sales_cost_corr if not pd.isna(sales_cost_corr) else 0, 2),
            round(price_cost_corr if not pd.isna(price_cost_corr) else 0, 2)
        ]

    # 将结果返回
    return correlation_results

#考虑相关性、可替代性等因素决策模型
def decision_upgrade(elasticity_matrix,correlation_results):



    import pulp as lp
    import numpy as np

    # 定义模拟的次数
    num_simulations = 5

    # 存储每次模拟的总利润
    profits = {}
    decision={}
    detailed_info={}
```

```
for sim in range(num_simulations):
    print(f"Simulation {sim + 1}/{num_simulations}")

    # 1. 模拟不确定性参数的波动
    # 销售量的变化
    future_crop_demand = {}
    for j in crops:
        if j in ['小麦', '玉米']: # 对小麦和玉米，年增长5%~10%
            growth_rate = np.random.uniform(0.05, 0.10)
            for y in years:
                future_crop_demand[j, y] = crop_demand[j] * (1 +
                    growth_rate)**(y - 2023)
        else: # 对其他作物 ±5%的变化
            for y in years:

                future_crop_demand[j, y] = crop_demand[j] * np.random.
                    uniform(0.95, 1.05)

    # 亩产量的变化，每年±10%的变化
    future_yield = {}
    for i, j in data:
        for y in years:
            future_yield[i, j, y] = data[(i, j)][1] * np.random.uniform(0.9,
                1.1)

    # 种植成本年增长5%
    future_costs = {}
    for i, j in data:
        for y in years:
            future_costs[i, j, y] = data[(i, j)][2] * (1.05)**(y - 2023)

    # 销售价格变化
    future_prices = {}
    for i, j in data:
        for y in years:
            if j in ['小麦', '玉米']: # 粮食价格基本稳定
                future_prices[i, j, 1, y] = data[(i, j)][3]
```

```
                future_prices[i, j, 2, y] = data[(i, j)][4]
            elif j in vegetables: # 蔬菜类每年增长5%
                future_prices[i, j, 1, y] = data[(i, j)][3] * (1.05)**(y - 2023)
                future_prices[i, j, 2, y] = data[(i, j)][4] * (1.05)**(y - 2023)
            elif j in fungi: # 食用菌价格每年下降1%-5%
                decline_rate = np.random.uniform(0.01, 0.05)
                future_prices[i, j, 1, y] = data[(i, j)][3] * (1 - decline_rate)
                    **(y - 2023)
                future_prices[i, j, 2, y] = data[(i, j)][4] * (1 - decline_rate)
                    **(y - 2023)
            else: # 默认保持不变
                future_prices[i, j, 1, y] = data[(i, j)][3]
                future_prices[i, j, 2, y] = data[(i, j)][4]

    # 2. 创建新的问题
    prob = lp.LpProblem("Crop_Planting_Optimization", lp.LpMaximize)
    min_area_thresholdd=0
    # 决策变量：x[i][j][s][y] 表示在第 i 块地上种植第 j 种作物的面积，s 表
        示季节 (1 或 2)，y 表示年份
    x = lp.LpVariable.dicts(
        "x",
        ((i, j, s, y) for i in land_plots for j in crops for s in [1, 2] for y in
            years
        if (i, j) in data and (s == 1 or data[(i, j)][5] == 2) and data[(i, j)][3
            if s == 1 else 4] != 0),
        lowBound=0, cat="Continuous")
    #第一阶段目标优化
    # 确保每块地在3年内的所有面积都至少种植一次豆类
    for i in land_plots:
        for y in range(2023, max(years) - 4): # 遍历每个连续的三年窗口
            prob += lp.lpSum(
                x[(i, j, s, y+k)] for j in legumes for s in [1, 2] for k in range
                    (3)
                if (i, j, s, y+k) in x and (s == 1 or data[(i, j)][5] == 2) #
                    只遍历允许的季节
            ) >= land_area[i], f"Legume_Rotation_{i}_{y}" # 设置为总
                面积
```

```
for i in land_plots:
    for y in years: # 遍历每个年份
        prob += lp.lpSum(x[(i, j, s, y)] for j in crops for s in [1, 2]
                         if (i, j) in data and (s == 1 or data[(i, j)][5]
                             == 2)
                         and (i, j, s, y) in x) <= land_area[i], f"
                             Land_Area_Constraint_{i}_{y}"

# 定义 waste 变量，去掉作物名称的空格
waste = lp.LpVariable.dicts("waste", ((j.strip(), y) for j in crops for y in
    years), lowBound=0, cat="Continuous")
# 去除作物名称中的空格
for j in crops:
    j_clean = j.strip() # 去掉作物名称前后的空格
    for y in years:
        for i in land_plots:
            for s in [1, 2]: # 添加季节 s 来确保约束名称唯一
                # 添加条件到 for 循环中，确保只处理符合条件的组合
                prob += lp.lpSum(
                    x.get((i, j_clean, s, y), 0) * data.get((i, j_clean),
                        [0, 0])[1]
                    for i in land_plots if (i, j_clean) in data and (s ==
                        1 or data[(i, j_clean)][5] == 2)
                ) <= crop_demand.get(j_clean, 0) + waste.get((
                    j_clean, y), 0), \
                f"Crop_Yield_Limit_With_Waste_{i}_{j_clean}_{y
                    }_S{s}" # 加上季节 s，确保唯一

# 目标函数：每年收益最大化
prob += lp.lpSum(
    (x.get((i, j, s, y), 0) * future_yield.get((i, j, y), 0) * future_prices.
        get((i, j, s, y), 0) # 销售收益
    - future_costs.get((i, j, y), 0) * x.get((i, j, s, y), 0) # 减去种植成
        本
    - waste.get((j, y), 0) * 0.5 * future_prices.get((i, j, s, y), 0)) # 超
        过部分按50%价格出售
```

```
    for i in land_plots for j in crops for s in [1, 2] for y in years
    if (i, j) in data and (s == 1 or data[(i, j)][5] == 2) # 只允许合法种
        植
    and is_valid_plot(i, j, s)
), "Maximize_Total_Profit_With_Discounted_Sales"


# 约束1：每块地的总种植面积不能超过可用面积
# 修改 Land_Area_Constraint 名称，添加季节 s
for i in land_plots:
    for y in years: # 遍历每个年份
        for s in [1, 2]: # 添加季节
            prob += lp.lpSum(x[(i, j, s, y)] for j in crops # 获取种植面
                积
                                if (i, j) in data and (s == 1 or data[(i, j)
                                    ][5] == 2)
                                and (i, j, s, y) in x) <= land_area[i], \
                                f"Land_Area_Constraint_{i}_{y}_S{s}"

#约束条件2：种植不能太分散
for j in crops:
    for i in land_plots:
        for y in years:
            # 根据地块面积动态调整最小种植面积
            if land_area[i] > 10: # 面积大于10亩的地块
                min_area_threshold = 0.1 * land_area[i] # 地块总面
                    积的10%
            else: # 面积较小的地块
                min_area_threshold = 0.05 * land_area[i] # 地块总面
                    积的5%

            # 确保每个约束的名称唯一，添加季节(s)变量到约束名中
            for s in [1, 2]:
                prob += lp.lpSum(x[(i, j, s, y)] for s in [1, 2] if (i, j, s,
                    y) in x) >= min_area_thresholdd, \
                        f"Min_Concentrated_Area_Constraint_{i}_{
                            j}_{y}_S{s}"
```

```
#约束3：禁止同一年内同一地块种植同一作物在两季
for i in land_plots:
    for j in crops:
        for y in years: # 遍历每一年
            if (i, j, 1, y) in x and (i, j, 2, y) in x: # 检查变量是否存在
                prob += x[(i, j, 1, y)] + x[(i, j, 2, y)] <= land_area[i],
                    f"No_Same_Crop_Two_Seasons_{i}_{j}_Year_{
                    y}"

#约束4：禁止连续两年种植同一种作物（防止重茬）
for i in land_plots:
    for j in crops:
        for y in years[:-1]: # 遍历年份，检查连续两年的种植情况
            # 第一季重茬限制
            if (i, j, 1, y) in x and (i, j, 1, y+1) in x:
                prob += x[(i, j, 1, y)] + x[(i, j, 1, y+1)] <= land_area[
                    i], f"No_Continuous_Planting_{i}_{j}_{y}_S1"
            # 第二季重茬限制
            if (i, j, 2, y) in x and (i, j, 2, y+1) in x:
                prob += x[(i, j, 2, y)] + x[(i, j, 2, y+1)] <= land_area[
                    i], f"No_Continuous_Planting_{i}_{j}_{y}_S2"

#约束5：互补性和替代性约束
for j1 in crops:
    for j2 in crops:
        if j1 != j2:
            # 替代性：如果两种作物是替代品，限制其种植总量
            elasticity_value = elasticity_matrix.get(j1, {}).get(j2, None
                ) # 避免 KeyError
            if elasticity_value is not None:
                # 替代性约束：当弹性值大于 0 时，表示替代性
                if elasticity_value > 0:
                    for i in land_plots:
                        for y in years:
                            # 检查决策变量是否存在，避免KeyError
                            if (i, j1, 1, y) in x and (i, j2, 1, y) in x:
```

```
                                prob += x[(i, j1, 1, y)] + x[(i, j2, 1, y)
                                    ] <= land_area[i], f"Substitute_{
                                    j1}_{j2}_{i}_{y}"

                    # 互补性约束：当弹性值小于 0 时，表示互补性
                    if elasticity_value < 0:
                        for i in land_plots:
                            for y in years:
                                # 检查决策变量是否存在，避免KeyError
                                if (i, j1, 1, y) in x and (i, j2, 1, y) in x:
                                    prob += x[(i, j1, 1, y)] >= x[(i, j2, 1,
                                        y)], f"Complement_{j1}_{j2}_{i}
                                        _{y}"


# 约束6：限制销售量与价格之间的过度依赖
gamma = 0.01 # 调整因子，根据具体情况设置
for i in land_plots:
    for j in crops:
        for y in years:
            price_j1 = future_prices.get((i, j, 1, y), 0) # 获取价格
            sales_price_corr_j1 = correlation_results.get(j, [0, 0, 0])[0]
                # 获取销售量与价格的相关性
            demand_limit = future_crop_demand.get((j, y), 0) -
                gamma * price_j1 * sales_price_corr_j1

            # 安全访问 x，避免 KeyError
            if (i, j, 1, y) in x:
                prob += x.get((i, j, 1, y), 0) <= lp.LpAffineExpression
                    (demand_limit), \
                        f"Demand_Price_Constraint_{i}_{j}_{y}
                            _S1"

'''
# 约束7：限制成本与价格之间的过度依赖
delta = 0.01 # 调整因子，根据具体情况设置
for i in land_plots:
```

```
        for j in crops:
            for y in years:
                price_j1 = future_prices.get((i, j, 1, y), 0) # 获取价格
                price_cost_corr_j1 = correlation_results.get(j, [0, 0, 0])[2]
                    # 获取价格与成本的相关性
                cost_limit = future_costs.get((i, j, y), 0) - delta *
                    price_j1 * price_cost_corr_j1

                # 安全访问 x，避免 KeyError
                if (i, j, 1, y) in x:
                    prob += future_costs.get((i, j, y), 0) <= lp.
                        LpAffineExpression(cost_limit), \
                            f"Price_Cost_Constraint_{i}_{j}_{y}_S1"

'''
# 约束8：限制销售量与成本之间的过度依赖
beta = 0.01 # 调整因子，根据具体情况设置
for i in land_plots:
    for j in crops:
        for y in years:
            cost_j1 = future_costs.get((i, j, y), 0) # 获取成本
            sales_cost_corr_j1 = correlation_results.get(j, [0, 0, 0])[1]
                # 获取销售量与成本的相关性
            demand_limit = future_crop_demand.get((j, y), 0) - beta
                * cost_j1 * sales_cost_corr_j1

            # 安全访问 x，避免 KeyError
            if (i, j, 1, y) in x:
                prob += x.get((i, j, 1, y), 0) <= lp.LpAffineExpression
                    (demand_limit), \
                        f"Demand_Cost_Constraint_{i}_{j}_{y}_S1
                            "


# 3. 求解问题
prob.solve()
```

```
# 输出求解状态
print(f"状态:␣{lp.LpStatus[prob.status]}")
yearly_profit = []
decision_yearly=[]

# 检查求解状态，并保存模拟的利润和决策
if lp.LpStatus[prob.status] == 'Optimal':
    yearly_profit = []
    decision_yearly = []
    detailed_yearly = [] # 存储详细信息

    for y in years:
        year_profit = 0
        year_detailed = []

        for i in land_plots:
            for j in crops:
                for s in [1, 2]:
                    if (i, j, s, y) in x:
                        planted_area = x[(i, j, s, y)].varValue
                        if planted_area > 0:
                            # 计算利润
                            year_profit += planted_area *
                                future_yield[i, j, y] * future_prices[i, j,
                                 s, y]
                            # 记录种植决策
                            decision_yearly.append((i, j, s, y,
                                planted_area))
                            # 记录详细信息：成本、销售量、单价、
                                亩产量、收益等
                            cost = future_costs[i, j, y]
                            #revenue = planted_area * future_yield[i,
                                 j, y] * future_prices[i, j, s, y]
                            demand = future_crop_demand[j, y]
                            yield_per_acre = future_yield[i, j, y] #
                                记录亩产量
```

```
                year_detailed.append({
                    '地块': i,
                    '作物': j,
                    '季节': s,
                    '年份': y,
                    '种植面积': round(planted_area, 1), # 保留一位小数
                    '亩成本': round(cost, 1), # 保留一位小数
                    '亩产量': round(yield_per_acre, 1), # 保留一位小数
                    '销售价格': round(future_prices[i, j, s, y], 2),
                    '销售量': round(demand, 1) # 保留一位小数
                })

            yearly_profit.append(year_profit)
            detailed_yearly.append(year_detailed)

        # 保存每次模拟的利润和决策
        profits[f"{sim + 1}"] = yearly_profit
        decision[f"{sim + 1}"] = decision_yearly
        detailed_info[f"{sim + 1}"] = detailed_yearly
    else:
        print(f"Simulation {sim + 1} did not find an optimal solution.")

# 找出最小利润对应的模拟
worst_simulation = min(profits, key=lambda k: sum(profits[k]))

# 输出最差情况的种植策略和详细信息
#print(f"最差利润对应的模拟: {worst_simulation}")
#print(f"最差利润: {sum(profits[worst_simulation])}")
#print(f"最差情况下的种植策略: {decision[worst_simulation]}")
#print(f"最差情况下的详细信息: {detailed_info[worst_simulation]}")

return detailed_info[worst_simulation],profits[worst_simulation]
```

```
#迭代更新的代码
# 数据更新：每次根据优化结果更新未来数据
def update_future_data(decision_info):
    # 更新 future_crop_demand, future_yield, future_prices, future_costs 等
    # 可替代性，互补性
    elasticity_matrix = calculate_elasticity(decision_info).fillna(0)

    #相关性
    correlation_results=calculate_correlation_matrices(decision_info)

    return elasticity_matrix,correlation_results

def calculate_matrix_difference(matrix_new, matrix_old):
    """计算互补性和替代性矩阵的前后差距"""
    total_diff = 0
    for crop1 in matrix_new:
        for crop2 in matrix_new[crop1]:
            # 使用 .get() 方法来安全访问矩阵元素，避免 KeyError
            value_new = matrix_new.get(crop1, {}).get(crop2, 0) # 如果不存
                在，默认值为 0
            value_old = matrix_old.get(crop1, {}).get(crop2, 0) # 如果不存
                在，默认值为 0
            total_diff += abs(value_new - value_old)
    return total_diff


def calculate_difference(correlation_results_new, correlation_results_old):
    """计算前后两次相关性字典的差距"""
    total_diff = 0
    for crop in correlation_results_new:
        # 使用 .get() 方法，确保即使某个作物不存在也不会报错
        corr_new = correlation_results_new.get(crop, [0, 0, 0])
        corr_old = correlation_results_old.get(crop, [0, 0, 0])

        # 计算相关性差异
        for i in range(3): # 假设每个作物有3个相关性指标
```

```
            total_diff += abs(corr_new[i] - corr_old[i])
    return total_diff

#迭代函数
def decision_iteration(elasticity_matrix, correlation_results, max_iterations=5,
    tolerance=100000, another_tolerance=10):
    previous_profit = 0
    previous_correlation_results = correlation_results.copy() # 保留前一次的
        相关性结果
    previous_elasticity_matrix = elasticity_matrix.copy() # 保留前一次的互补
        性和替代性矩阵

    for iteration in range(max_iterations):
        print(f"========第 {iteration + 1} 轮迭代=========")
        decision_info, total_profit = decision_up(elasticity_matrix,
            correlation_results)

        # 更新历史数据和矩阵
        elasticity_matrix, correlation_results = update_future_data(
            decision_info)
        # 如果收益收敛，停止迭代
        if abs(total_profit - previous_profit) < tolerance:
            print(f"收益收敛，停止迭代：总利润 {total_profit}")
            return decision_info



        # 计算相关性字典的前后差距
        correlation_diff = calculate_difference(correlation_results,
            previous_correlation_results)

        # 计算互补性和替代性矩阵的前后差距
        elasticity_diff = calculate_matrix_difference(elasticity_matrix,
            previous_elasticity_matrix)

        # 如果相关性和互补性差距均小于设定的阈值，停止迭代
        if correlation_diff < tolerance and elasticity_diff < another_tolerance:
```

```
            print(f"相关性和互补性差距收敛，停止迭代：相关性差距␣{
                correlation_diff}，互补性差距␣{elasticity_diff}")
            return decision_info


        # 更新前一次的相关性字典和互补性矩阵
        previous_correlation_results = correlation_results.copy()
        previous_elasticity_matrix = elasticity_matrix.copy()
        previous_profit = total_profit

    print(f"==========!!!!不收敛!!!==========")
    return decision_info,previous_profit # 如果达到最大迭代次数仍未收敛，
        返回最后的决策信息
```