基于混合整数规划与模拟退火算法的农业种植收益最大化研究

摘要

鉴于对可持续农业发展不断增长的需求，整合**需求侧和供给侧**两方面要素优化种植方案以最大化种植收益成为了农业生产的关键问题。本文基于题目提供的信息，考虑了**作物种植要求**、**市场波动**、**作物轮作要求**等多方面因素，建立了一个基于**分布式鲁棒优化**的**结合混合整数规划与模拟退火算法**的数学模型进行分析决策，从而为 2024 年至 2030 年收益最大化提供了全面的解决方案。

首先，对附件表单中高维、冗杂的历史数据特征，我们进行了**数据整合**、**异常值处理**、**数据可视化分析和相关系数矩阵的计算**等步骤。将附件 1 和附件 2 自然连接，并为地块名称、作物类型、种植季次编码，便于后续构造优化模型的约束条件。为保证后续求解的准确性，我们假设季度内销售单价服从正态分布，从而将**连续区间量化**为正态分布均值常量。

针对问题一，为了更好的分析，本文将问题聚焦在构建全面的种植方案约束条件，不仅考虑了常见的**种植面积和产量限制**，还结合了**重茬限制**、**豆类轮作**、**种植地块**等多种复杂约束。为了解决多约束问题达到效益最大化，我们构建了**基于混合整数规划和模拟退火算法的优化模型**，不仅提高了复杂约束下的求解效率，同时通过**动态扰动机制**避免陷入局部最优解，提升了模型的**全局最优性**。基于 2023 年的统计数据，求解出了 2024-2030 年的最优种植方案，在**滞销浪费**的情况下取得的最大收益**约为 3400 万元**，在**降价销售**的情况下取得的最大收益**约为 3420 万元**。

针对问题二，考虑到种植成本、销售单价等多方面的不确定因素和潜在风险，我们创新性地采用**分布式鲁棒优化**方法来**量化处理不确定性因素**，结合问题一中的**混合整数规划模型和模拟退火算法**，引入动态参数，最终求解出一个具有**高鲁棒性**和**可扩展性**的最大化收益的种植方案。该方法适用于多种不同场景，在本问中得到了 **约 4200 万元**的最大收益。

针对问题三，我们通过**相关性分析**和**需求曲线分析**找出作物之间的可替代性和互补性，然后通过 **Ridge 回归分析**来确定预期销售量与销售价格、种植成本之间的相关性，结果表明**生菜与小青菜等作物之间具有互补性，蔬菜的销售量与销售价格**、**种植成本有较强的相关性**。我们制定了如下策略：**互补性作物应保持合理的种植比例，而可替代性作物应避免同时大面积种植**。将我们分析得到的作物之间的可替代性与互补性关系，结合市场需求曲线，作为新的约束条件加入到问题二建立的优化模型中，通过模拟数据求解出种植方案的最大收益**约为 4300 万元**，相较于问题二有所提升。同时，我们采用**可视化对比分析**的方法发现问题三的种植策略更加稳健。

最后本文对所建立的模型进行了讨论和**灵敏度分析**，验证了模型的鲁棒性，综合评价了模型的优缺点。

关键词：混合整数规划　模拟退火算法　分布式鲁棒优化　最大化收益

一、 问题重述

1.1 问题背景

在农业种植过程中，由于不同地块类型适合不同的作物种植且种植资源条件有限，因此乡村需要充分利用有限的耕地资源，优化有机种植策略。[4] 由于市场的波动和气候的变化，乡村需要根据作物的历史销售情况和种植经验优化作物种植策略。另外，每种作物不能连续在同一地块重茬种植，且种植豆类作物对土壤存在有利影响。

因此，需要对市场需求、种植资源以及成本进行合理分析，从而优化作物的种植策略。**从需求侧来看，**作物的市场销售量和价格会受到气候因素和整体经济环境的影响。**从供给侧来看，**除了要考虑作物的成本和种植环境，还要考虑轮作要求，避免重茬种植导致产量下降。**从成本侧来看，**不同作物的亩产量和种植成本也会影响最终收益。结合需求侧、供给侧以及成本侧，可以分析各类作物的销售情况、种植资源以及成本产量之间的关系，从而制定合理的种植策略。

1.2 问题要求

问题设置是层层递进，并且服务于同一主题——如何通过对农作物的销售量、种植成本、亩产量及销售价格的分析，综合考虑长期经济效益和土壤健康，从而制定合理的种植方案，实现收益最大化。

问题一：假设各种农作物未来七年（2024-2030 年）预期销售量、每亩种植成本、每亩的产量和平均销售价格在 2023 年的基础上保持稳定，每季农作物的总产量超过预期的部分不能留到下一季销售，会造成浪费或者半价销售，分别求解这两种情况下的最优种植策略，以得到最大销售利润。

问题二：玉米和小麦未来七年（2024-2030 年）预期销量平均每年会增加 5% 至 10%，其他农作物未来七年每年的预期销量会在 2023 年的基础上波动 ±5%；农作物每亩的产量每年会波动 ±10%，农作物每亩的种植成本平均每年会增加约 5%；粮食类作物的售价基本保持不变，但是蔬菜类的售价平均每年会增加约 5%，而食用菌的售价每年会下降 1% 至 5%，其中羊毛菌的售价每年下降 5%。综合考虑这些不确定因素和潜在的风险，求解最优的种植策略，以获得最大的种植利润。

问题三：考虑作物之间的替代性和互补性，以及预期销售量与销售价格、种植成本之间的相关性，在问题 2 的基础上，求解最优种植策略（2024 2030 年），并与问题 2 结果对比分析。

二、 问题分析

2.1 总体分析

本题是一个关于农产品种植决策的优化问题。图1展示了我们问题分析的流程。

从分析目的来看，本题旨在利用 2023 年的销售数据和种植数据来实现对农产品种植策略的规划，可以将模型分为**约束和优化**两大部分。因此，本题需要完成三方面的任务——其一，利用 2023 年统计的种植方案、亩产量、种植成本和销售情况，根据经验预测未来的销售价格、亩产量、种植成本和预期销售量，作为优化农产品种植策略的参数。其二，综合考虑重茬种植、土地肥力、种植条件和田间管理等方面，制定一系列合理的约束条件。其三，通过建立合理的优化模型和算法，求解出在多约束条件下的最优种植方案，最大化种植利润。

从数据特性来看，本题给出的种植种类、种植季度、亩产量、销售单价数据表现为高度结构化，标签明确，索引清晰。数据中异常值较少且没有缺失值。其中，销售单价数据为连续的数据区间，因此需要探究数据区间的含义，给出合理的假设，通过数据处理将其转为常量。同时，为了便于分析，我们将附件 1 和附件 2 的数据通过相同的特征进行拼接。

从模型选择来看，农产品种植策略受到田间管理、种植条件、重茬种植和土地肥力等多种复杂的约束条件控制，其中定义的决策变量包括连续型和二元变量，因此我们考虑使用基于混合整数规划模型和模拟退火算法的优化模型来求解最优农产品种植策略。而问题二中需要考虑多种不确定因素和潜在风险，所以我们考虑使用分布式鲁棒优化来求解在各种场景下都具有强鲁棒性的最优种植方案。

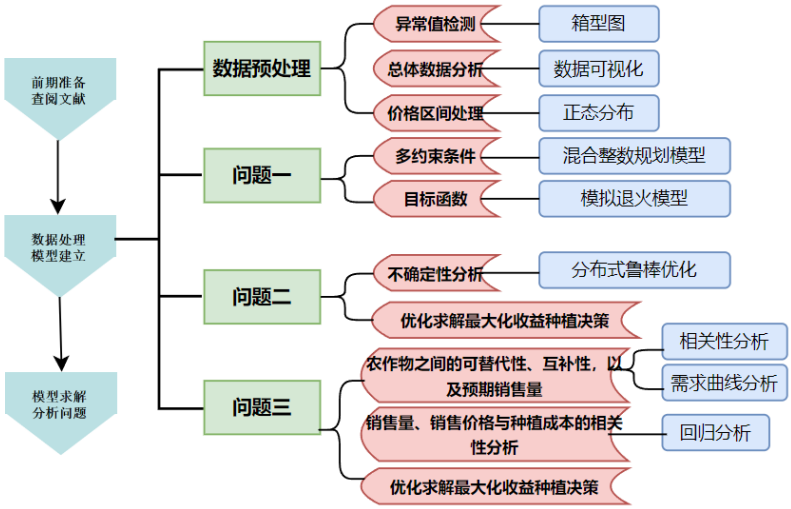


图 1　问题分析

2.2 问题一分析

问题一的核心在于构建**销售利润作为目标函数，以种植作物类型和种植面积为决策变量**的优化模型，对种植策略进行求解。我们首先对附件中的异常数据进行修正。其次，根据保持土壤肥力的原则，同时考虑到田间管理以及耕种作业等的便利性，得到有关作物种植条件、种植面积等约束条件，从而构建所需的优化模型。最后写出以农作物销售总利润为目标变量，种植作物类型和种植面积为自变量的目标函数，并使用启发式算法对其进行求解，得到要求年份的最优种植方案，最大化收益。

2.3 问题二分析

问题二的核心目的在于将未来各类种植作物的预期销售量、种植成本、亩产量和销售价格等因素的不确定性纳入考虑。针对这一问题，我们可以沿用问题一种的模型，通过将**不确定性因素量化**改变第一问模型的参数，最后求解目标函数的最优解，得到最大化收益和最优种植方案。

2.4 问题三分析

问题三的核心目的在于**考虑作物之间的可替代性和互补性，以及预期销售量与销售价格、种植成本之间的相关性**。因此，种植决策不再只影响单一的种植作物，同时可以影响具有可替代性和互补性的相关种植作物的种植收益。求解本问时应充分考虑作物之间的相关性以及市场的联动，得到最大化收益。将该收益和种植策略与问题二的结果进行对比，分析和评估作物之间的替代性和互补性，以及预期销售量与销售价格、种植成本之间的相关性对种植策略的影响。

三、 模型假设

本文提出以下合理假设：

- 假设地块类型、占地等种植资源在预测期内不会发生改变；
- 假设可供种植的作物种类已经全部给出，不考虑新增的作物种类；
- 假设不同作物的种植面积是可调的，可以根据预测出的不同需求调整种植比例；
- 假设豆类作物轮作的限制严格遵守每个地块所有土地三年内至少种植一次规则，并且不考虑其他的轮作作物；
- 假设土壤中的豆类作物根菌不会扩散至周围土壤中；
- 假设同一品种的作物在一个种植季度中的销售单价服从正态分布，模拟气候条件、市场需求、生产成本等多个外部因素会导致农产品价格的季节性波动，使得销售单价基本稳定在均值附近，偶尔出现极端高或低的价格；

四、 符号说明

符号	意义	符号	意义
$D_{j,k}$	地块 j 在第 k 年种植的豆类作物面积	A_j	地块 j 的总面积
$A_{\min}$	单个地块上某种作物的最小种植面积	P_i	作物 i 的销售价格
$Y_{i,j,k,m}$	作物 i 在地块 j 在第 k 年第 m 季的产量	$C_{i,j,k,m}$	作物 i 在地块 j 在在第 k 年第 m 季的种植成本
$A_{i,j,k,m}$	表示地块 j 在年份 k 第 m 季种植作物 i 的面积（亩）	$X_{i,j,k,m}$	二进制变量，表示地块 j 在年份 k 第 m 季是否种植作物 i
S_i	作物 i 的最大销售量		

五、 数据预处理

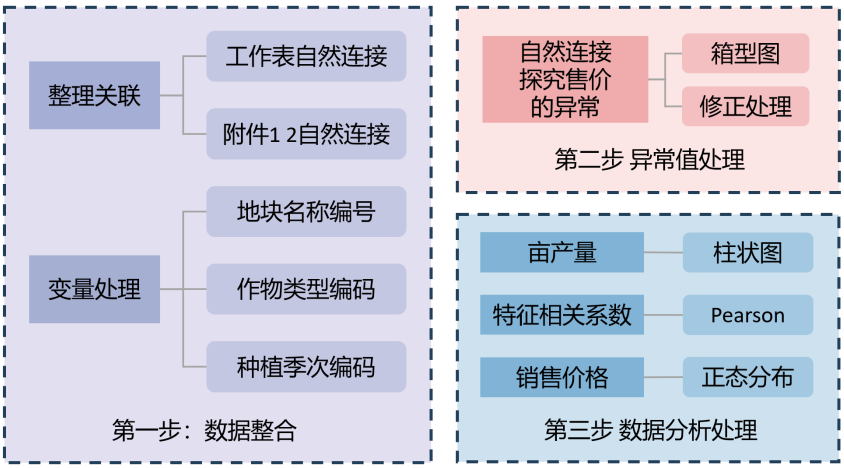
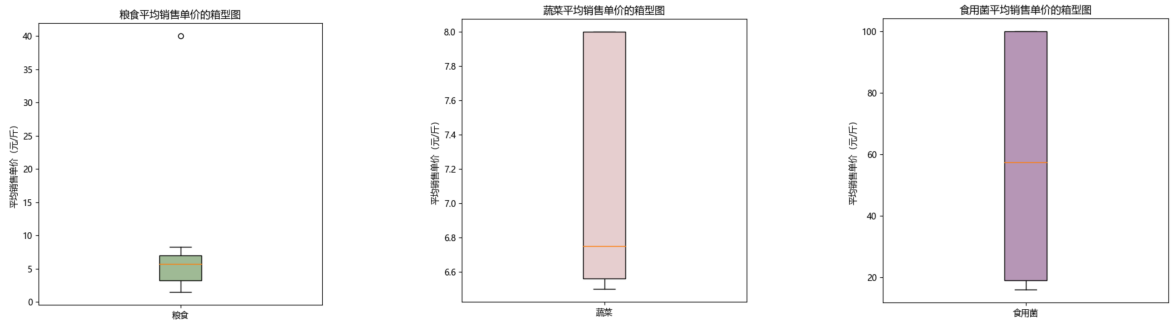


图 2　数据预处理思路

5.1 异常值检测

由图3可知，农作物的平均销售价格与其种类有较大的关联，粮食作物的平均销售价格主要集中在 3-8 元之间，而蔬菜作物的平均销售价格集中在 6-8 元之间。相比之下，食用菌的平均销售价格明显高于粮食作物和蔬菜作物，位于 20-100 元之间，并且分布更加对称。通过箱线图我们可以发现粮食作物的平均销售价格中**存在一个非常显著的异常值**。进一步观察数据，我们发现该异常值为荞麦的平均销售价格（40 元/斤）。通过

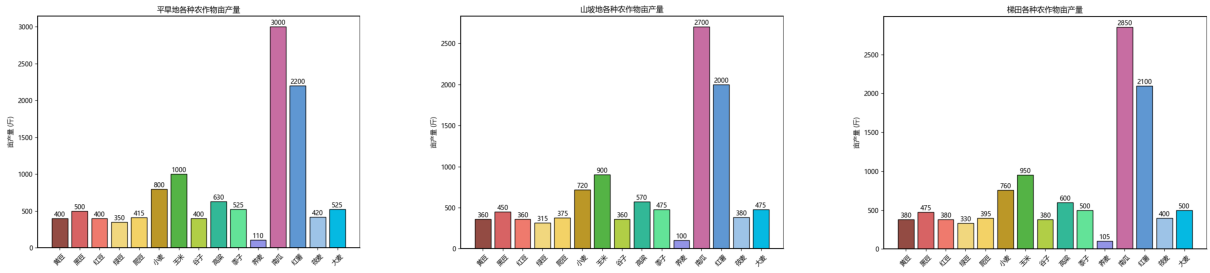


(a) 粮食异常值 **(b) 蔬菜异常值** **(c) 食用菌异常值**

图 3　异常值检测

市场调研，我们发现荞麦近期的价格普遍在 5 元/斤，据此我们将其销售单价修改为 3-5 元/斤，以保证数据的准确性。

5.2 总体数据分析



(a) 平旱地亩产量 **(b) 山坡地亩产量** **(c) 梯田亩产量**

图 4　种植作物亩产量

通过对平旱地、梯田、山坡地上不同农作物的亩产量的比较，如图14，我们发现不同的地块类型上各种农作物亩产量的分布趋势近似相同，亩产量最高的为南瓜，其次为红薯，都远高于其他农作物，并且亩产量最低的均为荞麦，显著低于其他农作物，由此可以认为亩产量受作物种类影响较大，受地块类型影响相对较小。

为了探究销售单价与亩产量、季节、作物种类等特征之间的关系，我们计算了所有特征以及销售单价之间的皮尔逊相关系数。

皮尔逊相关系数公式如下：

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{Y})^2}} \tag{1}$$

其中：

- r 是皮尔逊相关系数；
- x_i 和 y_i 是第 i 个样本的观测值；
- $\bar{X}$ 和 $\bar{Y}$ 分别是 X 和 Y 的均值；
- n 是样本的总数。

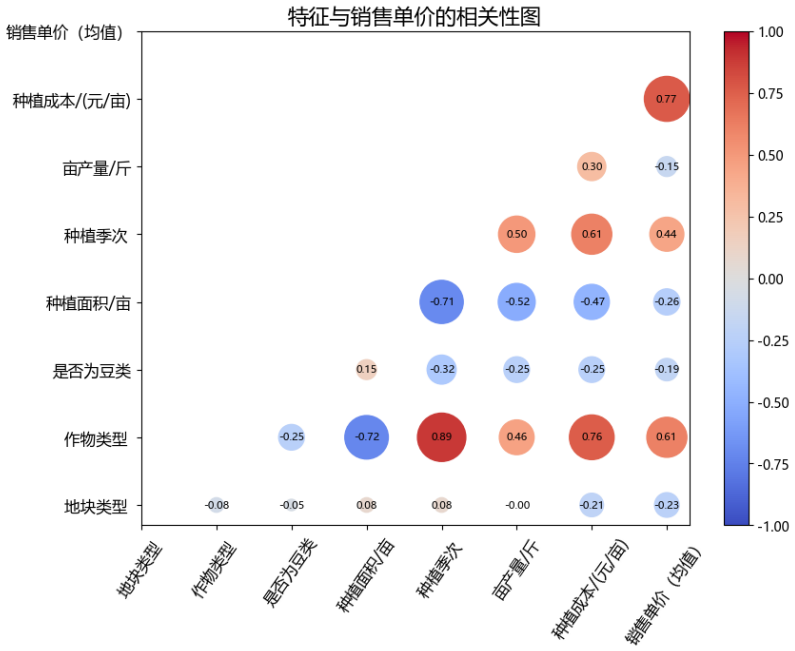


图 5　特征与销售单价的相关性图

我们将得到的所有相关系数可视化为了一张热力图，如图5所示。从图中我们可以看出种植成本与销售单价的相关性最强，相关系数为 0.77，作物类型、亩产量、种植季次与销售单价也有较高的正相关性，而是否为豆类、种植面积、地块类型与与销售单价的相关性较弱。

5.3 价格区间处理

农产品的销售单价往往会受到气候变化、市场供需变化等多种外部因素的影响，因此在一个销售季度中往往会表现为一个价格区间而非单一值。为了简化分析，我们假设农产品的销售单价服从正态分布，模拟价格受到市场需求和季节性变化等多种因素影响导致的波动。根据正态分布的特点，价格集中稳定在区间均值附近，偶尔出现最高价或最低价，因此我们将正态分布的均值作为销售单价的代表值来确保价格的上下限波动对结果的影响相对平衡。

正态分布的概率分布公式如下：

$$f(x|\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2}$$

最终我们得到的销售单价如表1所示。

表 1　作物种类和种植限制

作物	地块类型	种植季次	销售单价	作物	地块类型	种植季次	销售单价
小麦	平旱地	1	3.5	玉米	平旱地	1	3.0
黄豆	平旱地	1	3.25	绿豆	平旱地	1	7.0
谷子	平旱地	1	6.75	小麦	梯田	1	3.5
黑豆	梯田	1	7.5	红豆	梯田	1	8.25
绿豆	梯田	1	7.0	爬豆	梯田	1	6.75
谷子	梯田	1	6.75	小麦	梯田	1	3.5
谷子	梯田	1	6.75	高粱	梯田	1	6.0
黍子	梯田	1	7.5	黄豆	梯田	1	3.25
玉米	梯田	1	3.0	莜麦	梯田	1	5.5
大麦	梯田	1	3.5	荞麦	山坡地	1	40.0
南瓜	山坡地	1	1.5	黄豆	山坡地	1	3.25
红薯	山坡地	1	3.25	小麦	山坡地	1	3.5
红豆	山坡地	1	8.25	白萝卜	水浇地	2	2.5
大白菜	水浇地	2	2.5	白萝卜	水浇地	2	2.5
红萝卜	水浇地	2	3.25	榆黄菇	普通大棚	2	57.5
芹菜	智慧大棚	2	4.8	…	…	…	…

六、 问题一的模型建立与求解

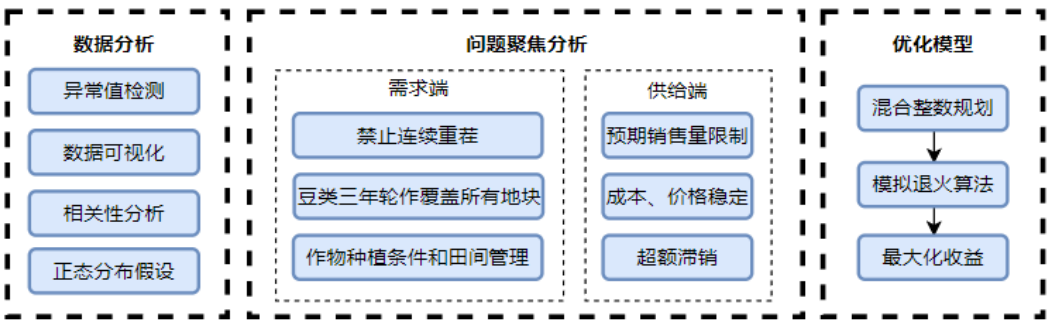


图 6　问题一总体思路

6.1 问题一求解思路

首先，我们对附件 1 和附件 2 进行了**异常值检测、数据可视化和相关性分析**。为了方便求解，我们将附件 2 中的销售单价取正态分布均值来作为我们计算最大收益时的销售单价。从供给端看，**每个地块不得连续重茬种植同一种作物，任意三年内种植豆类作物必须覆盖每个地块的所有土地**，同时要考虑到**作物的种植条件和田间管理**。从需求端看，假定各种农作物的预期销售量、种植成本、亩产量和销售价格**相对于 2023 年保持稳定**，每种作物的预期销售量成为需求端的核心限制条件。考虑到作物产量超过预期销售量的部分将无法正常销售等若干约束条件，本文构建了一个**基于混合整数规划模型和模拟退火算法的优化模型**以最大化作物种植收益。

6.2 问题一的混合整数规划模型建立

在问题一中，基于作物的种植条件和田间管理等约束条件，我们建立了一个混合整数规划模型 [1] 来优化农作物的种植方案。该模型可以在处理作物种类、种植面积等约束条件下最大化可销售部分的作物收益，满足需求端和供给端的全部要求。在混合整数规划模型求解时，我们利用 2023 年统计的种植数据初始化作物的预期销售量、亩产量、成本、销售价格等固定参数，同时将 2023 年数据作为约束条件，要求 2024-2030 年间满足重茬种植同一种作物且任意三年内种植豆类作物必须覆盖每个地块的所有土地。

6.2.1 农作物种植优化问题的抽象与变量定义

为了更好的用数学模型来描述农作物种植优化问题，我们明确了模型中的变量与参数，定义如下：

1. 决策变量定义

- $X_{i,j,k,m}$：二进制变量，表示地块 j 在年份 k 的第 m 季是否种植作物 i（若种植作物 i，则 $X_{i,j,k,m}$ = 1，否则 $X_{i,j,k,m}$=0）。
- $A_{i,j,k,m}$：连续变量，表示地块 j 在年份 k 第 m 季种植作物 i 的面积（亩）。

2. 固定参数定义

- S_i：作物 i 的最大可销售量，当作物产量超过 S_i 时，超过部分将滞销。
- $Y_{i,j,k,m}$：作物 i 在地块 j 在第 k 年第 m 季的亩产量。
- P_i：作物 i 的销售价格。
- $C_{i,j,k,m}$：作物 i 的种植成本（元/亩）。
- A_j：地块 j 的总面积，约束种植面积不会超过可用地块面积。
- A_{min}：某种作物在地块上的最小种植面积。

接下来我们将基于这些变量构建混合整数规划模型并进行优化分析。

6.2.2 构建目标函数

题目中给出了两种针对滞销产品的情况：(1) 超过部分滞销，造成浪费；(2) 超过部分按 2023 年销售价格的 50% 降价出售。

因此，针对不同情况我们构建了不同的目标函数：

针对情况一，我们的目标函数为：

$$Z = \sum_{i,j,k,m} (\min(A_{i,j,k,m} \cdot Y_{i,j,k,m}, S_i) \cdot P_i) - \sum_{i,j,k,m} A_{i,j,k,m} \cdot C_{i,j,k,m} \tag{3}$$

其中，第一项 $\min(A_{i,j,k,m} \cdot Y_{i,j,k,m}, S_i) \cdot P_i$ 表示每种作物的收益，仅考虑不超过预期销售量 S_i 的部分，超过部分视为滞销浪费；第二项 $A_{i,j,k,m} \cdot C_{i,j,k,m}$ 为种植成本。

针对情况二，我们的目标函数为：

$$\begin{aligned} Z = \sum_{i,j,k,m} & (\min(A_{i,j,k,m} \cdot Y_{i,j,k,m}, S_i) \cdot P_i \\ & + \max(0, A_{i,j,k,m} \cdot Y_{i,j,k,m} - S_i) \cdot 0.5 \cdot P_i) \\ & - \sum_{i,j,k,m} A_{i,j,k,m} \cdot C_{i,j,k,m} \end{aligned} \tag{4}$$

其中，第一项 $\min(A_{i,j,k,m} \cdot Y_{i,j,k,m}, S_i) \cdot P_i$ 是正常销售部分的收益，第二项 $\max(0, A_{i,j,k,m} \cdot Y_{i,j,k,m} - S_i) \cdot 0.5 \cdot P_i$ 表示超过销售量的部分按 50% 降价出售的收益，最后减去总的种植成本。

6.2.3 构建约束公式

为了优化农作物种植方案，除了最大化收益的目标函数以外，我们还需要构建一系列的约束条件。综合题目中给定的农作物种植规则，我们构建出了以下约束公式：

1. 每个地块所有土地三年内至少种植一次豆类作物

由于豆类作物可以增加土壤肥力，因此要求每个地块的所有土地在任意三年内必须种植过豆类作物，所以我们要求任意三年内每个地块的豆类作物种植面积要覆盖地块面积，构建约束如下：

$$\sum_{k=t}^{t+2} D_{j,k} \geq A_j \quad \forall j, t = 1, \ldots, T-2$$

这里 $D_{j,k}$ 表示地块 j 在第 k 年种植的豆类作物面积，A_j 为地块 j 的总面积。该约束确保豆类作物在三年内至少覆盖地块的所有土地。

2. 每种作物每季的种植地不能太分散

为了减少管理成本和方便田间管理，优化后的种植方案需要避免作物种植地块过于分散，因此模型需要限制每种作物在每个季节的种植地块数量不能超过阈值 $N_{\max}$ ，保证作物的集中种植，构建约束如下：

$$\sum_{j} \mathbb{I}(X_{i,j,k,m} = 1) \leq N_{\max} \quad \forall i, k, m$$

3. 每种作物在单个地块种植地面积不宜太小

为了避免管理不便，要求地块种植面积不宜过小，因此模型要求每个地块的作物种植面积不小于一个最小值 $A_{\min}$，构建约束如下：

$$A_{i,j,k,m} \geq A_{\min} \quad \forall i, j, k, m$$

4. 地块种类和地块面积固定

由于地块的总面积是固定的，种植的总面积不能超过可用面积，构建约束如下：

$$\sum_{i,k,m} A_{i,j,k,m} \leq A_j \quad \forall j$$

其中，A_j 是该地块的实际面积。

5.2023 年的种植情况需要考虑

2023 年的种植情况作为模型第一年的输入，不允许忽略，因此为其构建约束，保证 2023 年种植情况纳入规划的约束条件中，构建约束如下：

$$X_{i,j,2023,m} = X_{i,j,m}^{\text{已知}} \quad \forall i, j, m$$

6. 种植条件和重茬限制

综合题干信息和附件 1 中信息，我们发现农作物的种植受到地块类型、作物种类及季节性要求的严格限制。因此，我们需要考虑同时种植限制和避免重茬，进行分类讨论：

（1）平旱地、梯田和山坡地每年只适宜种植一季粮食类作物，且不能种植水稻，所以同一地块相邻年份不能种植相同的粮食作物，构建约束如下：

$$X_{i,j,k,1} + X_{i,j,k+1,1} \leq 1 \quad \forall i, j, k$$

（2）水浇地每年可以选择种植一季水稻或两季蔬菜。如果选择种植水稻，次年该地块不允许再种水稻；如果选择种植两季蔬菜，第一季可以种植多种蔬菜（大白菜、白萝卜和红萝卜除外），第二季只能种大白菜、白萝卜或红萝卜中的一种。此外，在同一年及跨年度的相邻季节同一地块不能连续种植相同作物，构建约束如下：

$$X_{\text{水稻},j,k,1} + X_{\text{水稻},j,k+1,1} \leq 1 \quad \forall j, k$$

$$X_{i,j,k,1} + X_{i,j,k,2} \leq 1 \quad \forall i,j,k$$

$$X_{i,j,k,2} + X_{i,j,k+1,1} \leq 1 \quad \forall i,j,k$$

$$X_{\text{大白菜/白萝卜/红萝卜},j,k,1} = 0 \quad \forall j,k$$

（3）智慧大棚每年适宜种植两季蔬菜，种植的蔬菜必须不同（且不包括大白菜、白萝卜和红萝卜），蔬菜在同一年及跨年相邻季节不能重茬，构建约束如下：

$$X_{i,\text{智慧大棚},k,1} + X_{i,\text{智慧大棚},k,2} \leq 1 \quad \forall i,k$$

$$X_{i,\text{智慧大棚},k,2} + X_{i,\text{智慧大棚},k+1,1} \leq 1 \quad \forall i,k$$

（4）特定作物如大白菜、白萝卜和红萝卜只能在水浇地的第二季种植。作物的种植要求如表2所示。

表 2　作物种类和种植限制

作物名称	作物类型	种植耕地
黄豆	粮食（豆类）	平旱地、梯田、山坡地
黑豆	粮食（豆类）	平旱地、梯田、山坡地
红豆	粮食（豆类）	平旱地、梯田、山坡地
绿豆	粮食（豆类）	平旱地、梯田、山坡地
爬豆	粮食（豆类）	平旱地、梯田、山坡地
小麦	粮食	平旱地、梯田、山坡地
玉米	粮食	平旱地、梯田、山坡地
谷子	粮食	平旱地、梯田、山坡地
高粱	粮食	平旱地、梯田、山坡地
黍子	粮食	平旱地、梯田、山坡地
荞麦	粮食	平旱地、梯田、山坡地
南瓜	粮食	平旱地、梯田、山坡地
红薯	粮食	平旱地、梯田、山坡地
莜麦	粮食	平旱地、梯田、山坡地
大麦	粮食	平旱地、梯田、山坡地
水稻	粮食	水浇地
豇豆	蔬菜（豆类）	水浇地（1）、普通大棚（1）、智慧大棚
刀豆	蔬菜（豆类）	水浇地（1）、普通大棚（1）、智慧大棚

作物名称	作物类型	种植耕地
芸豆	蔬菜（豆类）	水浇地（1）、普通大棚（1）、智慧大棚
土豆	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
西红柿	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
茄子	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
菠菜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
青椒	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
菜花	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
包菜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
油麦菜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
小青菜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
黄瓜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
生菜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
辣椒	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
空心菜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
黄心菜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
芹菜	蔬菜	水浇地（1）、普通大棚（1）、智慧大棚
大白菜	蔬菜	水浇地（2）
白萝卜	蔬菜	水浇地（2）
红萝卜	蔬菜	水浇地（2）
榆黄菇	食用菌	普通大棚（2）
香菇	食用菌	普通大棚（2）
白灵菇	食用菌	普通大棚（2）
羊肚菌	食用菌	普通大棚（2）

根据上表我们得到约束公式如下：

a) 粮食作物（豆类）约束条件

- **黄豆、黑豆、红豆、绿豆、爬豆**只能种植在平旱地、梯田和山坡地，每年只能种一季：

$$X_{i,j,k,m} = 0 \quad \forall j \notin \{\text{平旱地}, \text{梯田}, \text{山坡地}\}, \forall k, m, \forall i \in \{\text{黄豆}, \text{黑豆}, \text{红豆}, \text{绿豆}, \text{爬豆}\}$$

这些作物在相邻年份不能连续种植：

$$X_{i,j,k,1} + X_{i,j,k+1,1} \leq 1 \quad \forall j, k, \forall i \in \{\text{黄豆}, \text{黑豆}, \text{红豆}, \text{绿豆}, \text{爬豆}\}$$

b) 粮食作物（非豆类）约束条件

- **小麦、玉米、谷子、高粱、黍子、荞麦、南瓜、红薯、莜麦、大麦**只能种植在平旱地、梯田和山坡地，每年只能种一季：

$$X_{i,j,k,m} = 0 \quad \forall j \notin \{\text{平旱地}, \text{梯田}, \text{山坡地}\}, \forall k, m, \}$$

$$\forall i \in \{\text{小麦}, \text{玉米}, \text{谷子}, \text{高粱}, \text{黍子}, \text{荞麦}, \text{南瓜}, \text{红薯}, \text{莜麦}, \text{大麦}\}$$

相邻年份不能连续种植：

$$X_{i,j,k,1}+X_{i,j,k+1,1} \leq 1 \quad \forall j, k, \forall i \in \{\text{小麦}, \text{玉米}, \text{谷子}, \text{高粱}, \text{黍子}, \text{荞麦}, \text{南瓜}, \text{红薯}, \text{莜麦}, \text{大麦}\}$$

c) 水稻种植约束条件

- **水稻**只能在水浇地种植，每年只能种一季：

$$X_{\text{水稻},j,k,1} = 0 \quad \forall j \notin \{\text{水浇地}\}, \forall k$$

水稻不能在相邻年份连续种植：

$$X_{\text{水稻},j,k,1} + X_{\text{水稻},j,k+1,1} \leq 1 \quad \forall j, k$$

d) 豆类蔬菜约束条件

- **豇豆、刀豆、芸豆**可在水浇地、普通大棚和智慧大棚种植，但需要满足以下条件：
  - **豇豆**在水浇地只能在第一季种植：

$$X_{\text{豇豆},j,k,2} = 0 \quad \forall j \in \{\text{水浇地}\}, \forall k$$

  - **刀豆、芸豆**的种植同样要遵循相应的季节性要求：

$$X_{\text{刀豆/芸豆},j,k,2} = 0 \quad \forall j \in \{\text{水浇地}\}, \forall k$$

e) 普通大棚和智慧大棚的蔬菜种植约束条件

- **蔬菜作物（如土豆、西红柿、茄子、菠菜、青椒、菜花等）**在普通大棚和智慧大棚的种植限制如下：
  - **普通大棚**每年第一季可种植蔬菜，第二季只能种植食用菌：

$$X_{i,\text{普通大棚},k,2} = 0 \quad \forall i \in \{\text{蔬菜类作物}\}, \forall k$$

  - **智慧大棚**可每年种植两季蔬菜，且相邻季节不能种植相同作物：

$$X_{i,\text{智慧大棚},k,1} + X_{i,\text{智慧大棚},k,2} \leq 1 \quad \forall i, k$$

$$X_{i,\text{智慧大棚},k,2} + X_{i,\text{智慧大棚},k+1,1} \leq 1 \quad \forall i, k$$

f) 特定蔬菜（大白菜、白萝卜、红萝卜）的种植约束条件

- **大白菜、白萝卜、红萝卜**只能在水浇地的第二季种植：

$$X_{\text{大白菜/白萝卜/红萝卜},j,k,1} = 0 \quad \forall j \in \{\text{水浇地}\}, \forall k$$

$$X_{\text{大白菜/白萝卜/红萝卜},j,k,m} = 0 \quad \forall j \notin \{\text{水浇地}\}, \forall k, m$$

g) 食用菌的种植约束条件

- **榆黄菇、香菇、白灵菇、羊肚菌**只能在普通大棚的第二季种植：

$$X_{\text{榆黄菇/香菇/白灵菇/羊肚菌,普通大棚},k,1} = 0 \quad \forall k$$

食用菌不能与其他作物在同一季节种植：

$$X_{\text{榆黄菇/香菇/白灵菇/羊肚菌,普通大棚},k,2} = 1$$

综合上面所有内容，我们汇总除种植限制和重茬限制以外的约束条件如下：

$$\begin{cases} \sum_{k=t}^{t+2} D_{j,k} \geq A_j \quad \forall j, t = 1, \ldots, T-2 & (1)\ \text{豆类作物三年种植覆盖} \\ \sum_j \mathbb{I}(X_{i,j,k,m} = 1) \leq N_{\max} \quad \forall i, k, m & (2)\ \text{每季种植地块不分散} \\ A_{i,j,k,m} \geq A_{\min} \quad \forall i, j, k, m & (3)\ \text{种植面积不小于最小面积} \\ \sum_{i,k,m} A_{i,j,k,m} \leq A_j \quad \forall j & (4)\ \text{地块种类和面积固定} \\ X_{i,j,2023,m} = X_{i,j,m}^{\text{已知}} \quad \forall i, j, m & (5)\ 2023\ \text{年种植限制} \end{cases} \tag{5}$$

6.3 问题一的混合整数规划模型的求解和分析

混合整数规划模型在多约束条件下找到了一个可行解如表3所示：

表 3　混合整数规划模型 2024-2030 年种植方案初始解（超过部分按 2023 年销售价格的 50% 降价出售）

年份	作物名称	地块名称	种植季次	种植面积/亩	年份	作物名称	地块名称	种植季次	种植面积/亩
2024	黄豆	A1	0	59.05	2024	黄豆	A3	0	35.0
2024	黄豆	B13	0	35.0	2024	黄豆	B14	0	16.0
2024	黑豆	A1	0	43.7	2024	红豆	B6	0	68.8
2024	红豆	B10	0	25.0	2024	绿豆	C1	0	15.0
2024	绿豆	C2	0	13.0	2024	绿豆	C4	0	18.0
2024	绿豆	C5	0	27.0	2024	爬豆	B8	0	44.0
2024	小麦	A2	0	55.0	2024	小麦	A3	0	35.0

年份	作物名称	地块名称	种植季次	种植面积/亩	年份	作物名称	地块名称	种植季次	种植面积/亩
2024	小麦	A5	0	68.0	2024	小麦	A6	0	55.0
2024	玉米	A1	0	75.15	2024	玉米	A4	0	57.6
2024	谷子	B12	0	36.0	2024	高粱	B1	0	50.0
2024	黍子	B5	0	25.0	2024	荞麦	C3	0	15.0
2024	南瓜	B4	0	22.4	2024	莜麦	B2	0	36.8
2024	大麦	C6	0	20.0	2024	水稻	D2	0	8.0
2024	水稻	D3	0	11.6	2024	水稻	D4	0	4.8
2024	水稻	D5	0	8.0	2024	水稻	D6	0	9.6
2024	豇豆	D8	0	20.0	2024	豇豆	E6	0	0.6
2024	豇豆	E13	0	0.6	2024	刀豆	D1	0	15.0
2024	刀豆	E2	0	0.6	2024	刀豆	E14	0	0.6
2024	芸豆	F1	0	0.6	2024	土豆	E7	0	0.6
2024	土豆	E11	0	0.6	2024	土豆	F4	0	0.6
2024	西红柿	D7	0	11.75	2024	西红柿	E1	0	0.6
2024	西红柿	E3	0	0.6	2024	西红柿	E4	0	0.6
2024	西红柿	E9	0	0.6	2024	茄子	E1	0	0.6
2024	茄子	E6	0	0.6	2024	茄子	E9	0	0.6
2024	茄子	E12	0	0.6	2024	茄子	F4	0	0.6
2024	青椒	E1	0	0.6	2024	菜花	F3	0	0.6
2024	包菜	E13	0	0.6	2024	油麦菜	E1	0	0.6
2024	小青菜	D7	0	11.0	2024	小青菜	E10	0	0.3
2024	小青菜	F1	0	0.3	2024	黄瓜	E16	0	0.6
2024	生菜	E1	0	0.3	2024	辣椒	E4	0	0.6
2024	空心菜	E5	0	0.48	2024	榆黄菇	E1	2	0.6
2024	榆黄菇	E11	2	0.6	2024	榆黄菇	E14	2	0.6
2024	香菇	E3	2	0.6	2024	香菇	E9	2	0.6
2024	香菇	E15	2	0.6	2024	白灵菇	E6	2	0.6
2024	白灵菇	E9	2	0.6	2024	白灵菇	E15	2	0.6
2024	羊肚菌	E2	2	0.6	2024	羊肚菌	E3	2	0.6
2024	羊肚菌	E8	2	0.6	2024	羊肚菌	E10	2	0.6
…	…	…	…	…	…	…	…	…	…

此时，两个初始解方案得到的最大化收益如下：

- 超过部分滞销，造成浪费:30306272.75 元。
- 超过部分按 2023 年销售价格的 50% 降价出售:33322990.98 元。

6.4 问题一基于混合整数规划模型和模拟退火算法的优化模型建立

针对第一题，我们已经用混合整数规划模型生成的一个初始解，接下来应用模拟退火算法 [3] 进行进一步的优化，通过引入随机扰动来探索解空间，避免陷入局部最优解，最终收敛到接近全局最优的种植规划。

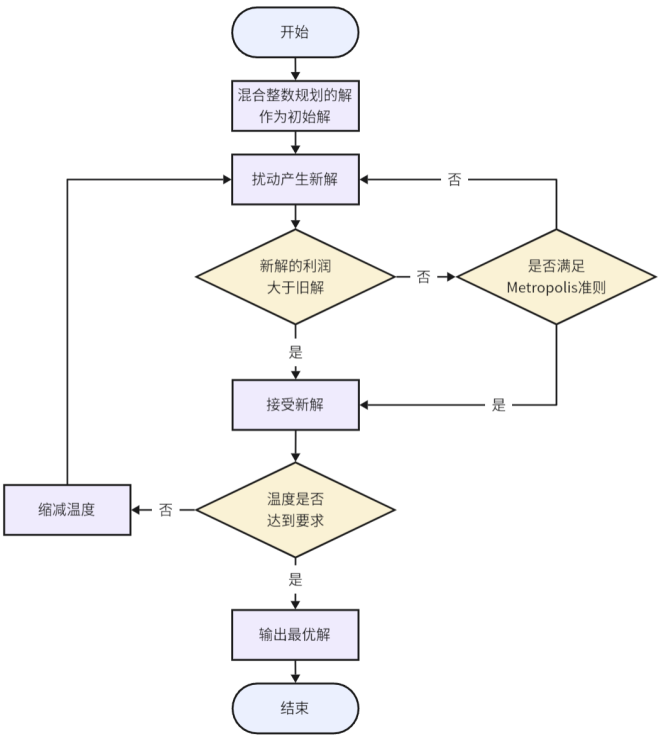


图 7　模拟退火算法流程图

下面是我们建立模拟退火算法过程：

Step1. 解的随机扰动

为了对当前的农作物种植方案进行一定程度的调整，模拟退火需要对当前解进行随机扰动，生成一个新的解，我们将从下面几个角度进行随机扰动：

- **作物种类的变化：**随机选择一个地块，调整其种植的作物。譬如，某个地块原本种植黄豆，可以随机变更为玉米或高粱等符合该地块条件的作物。这个扰动的目的是探索其他作物组合对收益的影响。
- **种植面积的调整：**随机选择一个地块，调整其作物的种植面积，确保该地块的总种植面积不超过地块的可用面积。譬如，某个地块种植了 50% 的玉米和 50% 的小麦，可以随机调整为 60% 的玉米和 40% 的小麦，或引入新的作物。

- **季次数种植顺序的变换：**随机选择一个地块，调整其作物的种植季次的顺序，确保该地块避免重茬等问题。譬如，某个地块前一季种植了玉米，后一季种植了小麦，可以随机调整为前一季种植小麦，后一季种植玉米，或引入新的作物。

Step2. 接受新解的策略

每次扰动后产生一个新的解，模拟退火算法需要决定是否接受该新解。基于新解与当前解的目标函数值差异，我们制定下面策略：

- **接受更优的解**：如果新解的收益比当前解高，直接接受新解。
- **接受次优解的概率**：如果新解的收益较低，根据以下概率函数接受次优解：

$$P = \exp\left(\frac{f_{\text{new}} - f_{\text{current}}}{T}\right)$$

其中，f_{new} 和 f_{current} 分别表示新解和当前解的收益，T 为当前温度。温度较高时接受次优解的概率较大，随着温度降低，次优解的接受概率逐渐减小。

Step3. 逐步降温

温度的控制决定了解的探索深度和收敛速度。逐步降温的过程如下：

- **初始温度**：为了保证探索更多可能的种植方案，设置较高的初始温度 T_0，探索更多可能的种植方案。
- **温度更新策略**：温度 T 随迭代次数逐渐降低，通常采用指数衰减策略：

$$T_{\text{new}} = \alpha \cdot T_{\text{current}}, \quad 0 < \alpha < 1$$

其中，α 为降温因子，通常取值在 0.8 到 0.99 之间。α 越大，降温过程越慢。

- **温度终止条件**：当温度降至阈值 T_{min} 时，算法终止，进入收敛阶段。

逐步降温使得算法从初期的全局探索逐步转向局部优化，并最终收敛到较优解，找到最终的最优种植策略。

Step4. 终止条件

构建出的模拟退火算法通过以下条件之一终止：

- **最大迭代次数**：设置最大迭代次数，当达到该次数时，算法自动停止。
- **温度降至阈值**：当温度降至设定的最低温度 T_{min} 时，算法终止，此时系统几乎不再接受次优解。

6.5 问题一模型求解与分析

通过模拟退火算法，我们得到更新后的最优种植方案，如表4和表5所示：

表 4　模拟退火辅助的混合整数规划模型 2024-2030 年种植方案优化解（情况 1）

年份	作物名称	地块名称	种植季次	种植面积/亩	年份	作物名称	地块名称	种植季次	种植面积/亩
2024	黄豆	A1	0	63.61	2024	黄豆	A3	0	35.0
2024	黄豆	B4	0	11.2	2024	黄豆	B13	0	35.0
2024	黑豆	A1	0	32.0	2024	黑豆	B12	0	36.0
2024	红豆	B2	0	18.4	2024	红豆	B6	0	48.894737
2024	绿豆	C1	0	15.0	2024	绿豆	C2	0	13.0
2024	绿豆	C4	0	18.0	2024	绿豆	C5	0	27.0
2024	绿豆	C6	0	20.0	2024	爬豆	B8	0	44.0
2024	小麦	A2	0	44.0	2024	小麦	A3	0	24.87
2024	小麦	A5	0	68.0	2024	小麦	A6	0	44.0
2024	小麦	B6	0	34.4	2024	玉米	A1	0	32.0
2024	玉米	A4	0	61.97	2024	玉米	B5	0	20.0
…	…	…	…	…	…	…	…	…	…

表 5　模拟退火辅助的混合整数规划模型 2024-2030 年种植方案优化解（情况 2）

年份	作物名称	地块名称	种植季次	种植面积/亩	年份	作物名称	地块名称	种植季次	种植面积/亩
2024	黄豆	A1	0	80.0	2024	黄豆	A3	0	35.0
2024	黄豆	B13	0	35.0	2024	黄豆	B14	0	20.0
2024	黑豆	B6	0	68.8	2024	红豆	B2	0	36.8
2024	红豆	B10	0	25.0	2024	绿豆	C1	0	15.0
2024	绿豆	C2	0	13.0	2024	绿豆	C4	0	18.0
2024	绿豆	C5	0	27.0	2024	爬豆	B8	0	44.0
2024	小麦	A2	0	55.0	2024	小麦	A3	0	35.0
2024	小麦	A5	0	68.0	2024	小麦	A6	0	55.0
2024	玉米	A1	0	75.15	2024	玉米	A4	0	57.6
2024	谷子	B3	0	32.0	2024	谷子	B7	0	44.0
2024	谷子	B9	0	40.0	2024	谷子	B11	0	48.0
2024	谷子	B12	0	36.0	2024	高粱	B1	0	50.0
…	…	…	…	…	…	…	…	…	…

最终题目条件以及我们的得到的**种植方案最大化收益**如下：

- 超过部分滞销，造成浪费:**34064884.54 元**。
- 超过部分按 2023 年销售价格的 50% 降价出售 **34168726.34 元**。

七、 问题二的模型建立与求解

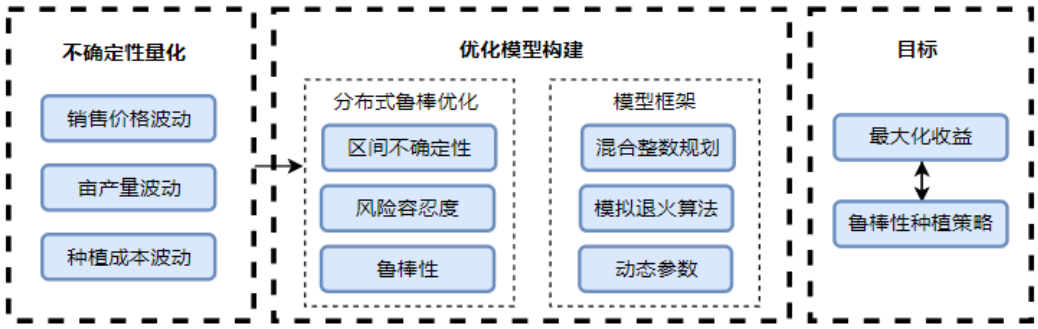


图 8　问题二总体思路

7.1 问题二求解思路

问题二基于问题一的模型框架，结合未来（2024-2030 年）各类作物的**不确定性因素进行量化处理**，构建出新的优化模型。考虑到种植成本、销售单价等多方面的不确定因素和潜在风险，我们选择使用**分布式鲁棒优化**来作为量化不确定因素的方法。首先我们将玉米小麦 5% 至 10% 的预期销售增长率、±5% 的其他农作物预期销量波动等不确定因素量化为区间变量，作为模型的**动态参数**。基于此，我们采用分布式鲁棒优化方法来在模型中引入区间不确定性和风险容忍度，以确保在所有可能场景下求解出的最优种植策略具有**鲁棒性**。在模型框架上仍沿用问题一的混合整数规划模型和模拟退火算法，引入动态参数，并添加未来波动的上下限作为新的约束，最终求解出最大化收益的最优种植方案。

7.2 问题二模型构建

7.2.1 不确定性因素建模

在农产品种植方案决策时，种植方案不仅受到当前市场和资源条件的影响，同时会受到地理环境和气候变化等方面的制约，因此，针对这些不确定性因素，我们需要优化种植方案来降低生成风险，提高种植收益。根据题目信息，我们对不确定性因素进行了进一步的解释和数学化。

（1）预期销售量的不确定性

a）小麦和玉米

小麦和玉米的销售量预计在未来几年会有 5%-10% 的增长：

$$S_{\text{小麦},k} = S_{\text{小麦},2023} \times (1 + r_{\text{小麦},k}) \tag{6}$$

$$S_{\text{玉米},k} = S_{\text{玉米},2023} \times (1 + r_{\text{玉米},k}) \tag{7}$$

其中，$r_{\text{小麦},k} \in [0.05, 0.10]$ 和 $r_{\text{玉米},k} \in [0.05, 0.10]$。

b）其他农作物

其他农作物未来每年的销售量相对较为稳定，预计每年在 ±5% 范围内波动：

$$S_{i,k} = S_{i,2023} \times (1 + \delta_{i,k}) \tag{8}$$

其中，$\delta_{i,k} \in [-0.05, 0.05]$。

（2）亩产量的不确定性

农作物的亩产量产生影响，预计每年有 ±10% 的波动：

$$Y_{i,j,k,m} = Y_{i,j,2023,m} \times (1 + \epsilon_{i,k}) \tag{9}$$

其中，$\epsilon_{i,k} \in [-0.10, 0.10]$。

（3）种植成本的不确定性

受到市场条件等外部因素影响，种植成本每年将以 5% 的速度增长：

$$C_{i,j,k,m} = C_{i,j,2023,m} \times (1 + 0.05)^{k-2023} \tag{10}$$

（4）销售价格的不确定性

a）粮食类作物

粮食类作物的销售价格在未来预计保持稳定：

$$P_{i,k,m} = P_{i,2023,m} \tag{11}$$

b）蔬菜类作物

蔬菜类作物的销售价格预计每年增长 5%：

$$P_{i,k,m} = P_{i,2023,m} \times (1 + 0.05)^{k-2023} \tag{12}$$

c）食用菌类作物

食用菌类作物的价格则呈下降趋势，特别是羊肚菌，价格预计每年下降 5%：

$$P_{i,k,m} = P_{i,2023,m} \times (1 - d_{i,k})^{k-2023} \tag{13}$$

其中，$d_{i,k} \in [0.01, 0.05]$。

对于羊肚菌：

$$P_{\text{羊肚菌},k,m} = P_{\text{羊肚菌},2023,m} \times (1 - 0.05)^{k-2023} \tag{14}$$

7.2.2 潜在的种植风险

（1）气候和自然灾害风险

气候条件和自然灾害是制约农作物产量和质量的关键因素，气候的变化和病虫害给农业种植带来了巨大的风险，不适宜的气候条件或骤变的气候条件会导致作物减产或者绝收。温度波动和降雨不确定性困难会影响作物养分吸收，甚至导致生长困难或死亡。

（2）市场价格波动风险

市场价格波动是农作物种植策略面临的重要风险之一，受到供需关系、国际市场、政策调整等多方面因素，可能导致产品过剩或市场需求的突然下降，价格大幅度下跌，从而给农户收入带来风险。

（3）种植技术和管理风险

使用不当的农作物管理技术或者违背作物的生长规律，可能导致病虫害和亩产减少等问题。例如过度施肥和滥用农药等不当种植方式不仅会降低作物产量，同时可能导致土壤肥力下降，有害微生物繁殖。

（4）政策风险

政府的农业政策、补贴和国际贸易政策的变动会给农业生产带来风险，补贴的调整可能会改变农户种植作物的倾向，而政策的突然变化可能导致种植作物的滞销。

7.2.3 分布式鲁棒优化构建

Step 1. 构建鲁棒优化模型

构建一个最大化收益的优化模型：

- 外层优化：决策种植策略。
- 内层优化：在不确定性集合中寻找最不利的场景。

Step 2. 公式化模型

设目标函数 Z 为最大化收益：

$$\max_{\mathbf{x}} \min_{\mathbf{u} \in \mathcal{U}} Z(\mathbf{x}, \mathbf{u})$$

其中 $\mathbf{x}$ 为决策变量，$\mathbf{u}$ 为不确定性变量，$\mathcal{U}$ 为不确定性集合。

Step 3. 求解模型

使用模拟退火算法，交替求解外层的最优决策和内层的不利场景。

7.3 问题二模型的求解和分析

基于分布式鲁棒优化，我们在问题一中混合整数规划和模拟退火算法的优化模型框架上通过处理众多不确定性因素，得到了下面如表6所示的最优方案，该方案具有很强的鲁棒性：

表 6　问题二模型的 2024-2030 年最优种植方案

年份	作物名称	地块名称	种植季次	种植面积/亩	年份	作物名称	地块名称	种植季次	种植面积/亩
2024	小麦	A1	0	70.8	2024	玉米	A2	0	85.2
2024	黄豆	A4	0	76.3	2024	绿豆	A5	0	64.3
2024	谷子	A6	0	58.1	2024	小麦	B1	0	121.8
2024	黍子	B10	0	22.7	2024	黄豆	B11	0	62.3
2024	玉米	B12	0	49.7	2024	莜麦	B13	0	37.5
2024	大麦	B14	0	22.7	2024	黑豆	B2	0	43.2
2024	红豆	B3	0	42.7	2024	绿豆	B4	0	30.7
2024	爬豆	B5	0	27.3	2024	谷子	B6	0	133.0
2024	高粱	B9	0	49.9	2024	荞麦	C1	0	13.4
2024	南瓜	C2	0	13.0	2024	黄豆	C3	0	14.6
…	…	…	…	…	…	…	…	…	…
2025	小麦	A1	0	74.1	2025	玉米	A2	0	84.5
2025	黄豆	A4	0	76.5	2025	绿豆	A5	0	71.1
2025	谷子	A6	0	53.8	2025	黍子	B10	0	23.7
2025	黄豆	B11	0	62.8	2025	玉米	B12	0	49.6
2025	莜麦	B13	0	34.5	2025	大麦	B14	0	20.5
2025	黑豆	B2	0	48.1	2025	红豆	B3	0	42.7
2025	绿豆	B4	0	28.7	2025	爬豆	B5	0	26.9
2025	小麦	B7	0	120.4	2025	谷子	B8	0	129.3
2025	高粱	B9	0	54.1	2025	荞麦	C1	0	13.7
…	…	…	…	…	…	…	…	…	…

最终，考虑到种植成本、销售单价等多方面的不确定因素和潜在风险，我们的模型给出**最优方案的最大化利润为 42784489.60 元**。

八、 问题三的模型建立与求解

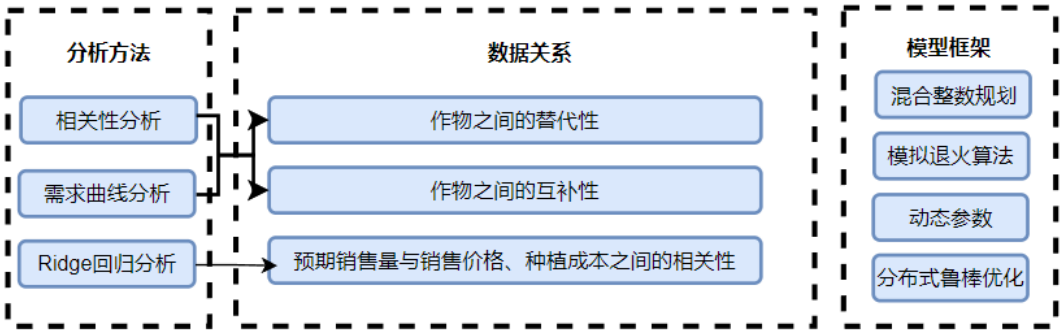


图 9　问题三总体思路

8.1 问题三求解思路

首先，我们通过**相关性分析和需求曲线分析**找出来作物之间的**替代性和互补性**，然后通过 **Ridge 回归分析**来确定预期销售量与销售价格、种植成本之间的**相关性**。将我们找出的关系作为新的约束条件加入到问题二中建立的优化模型里，求解出最大收益的种植方案，并与问题二结果进行对比。此处，我们使用问题二中跑出的 2024-2030 年数据来进行相关性分析。

8.2 作物可替代性和互补性

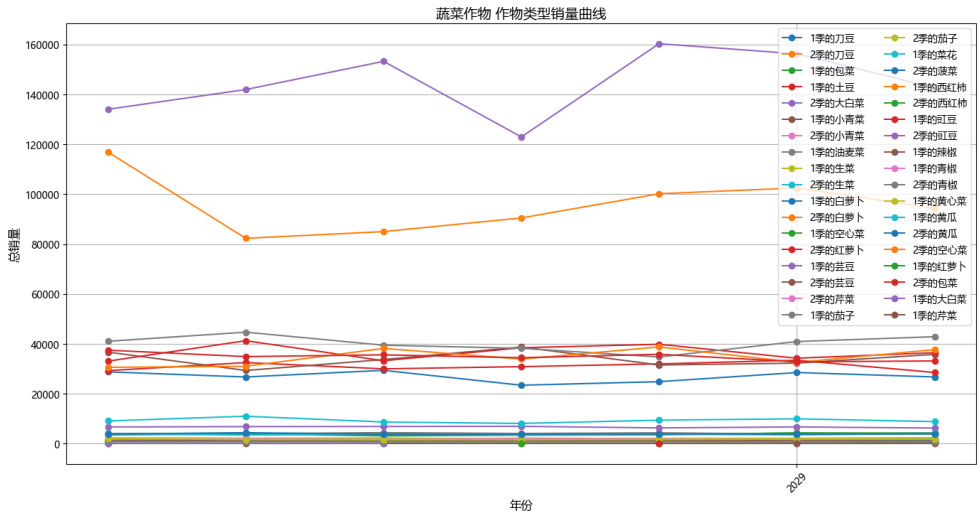


图 10　蔬菜需求曲线

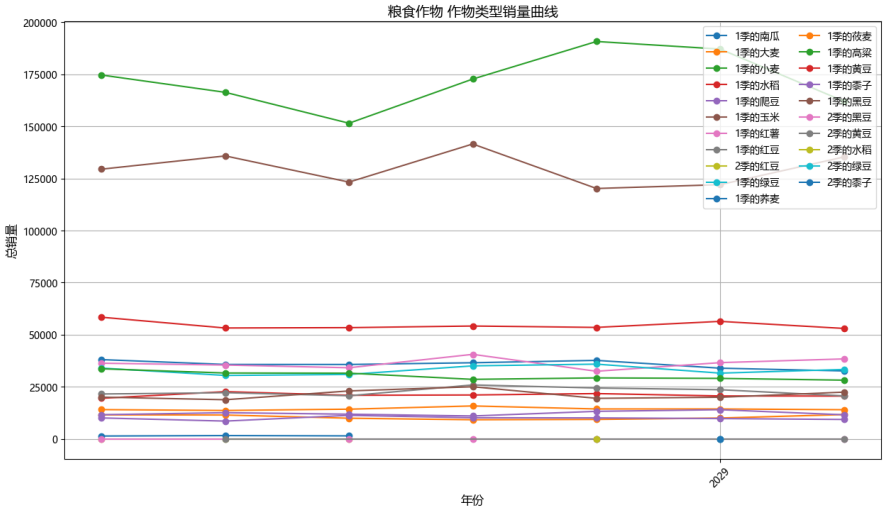


图 11　粮食需求曲线

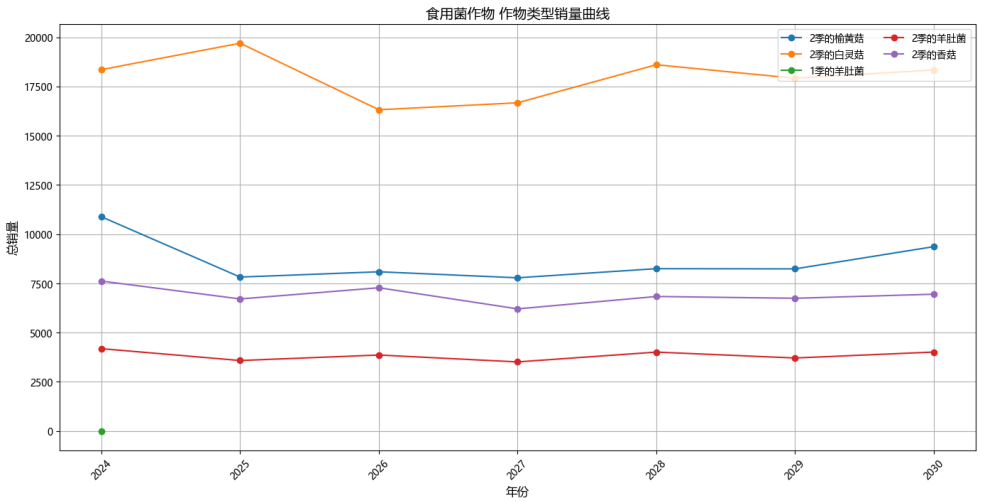
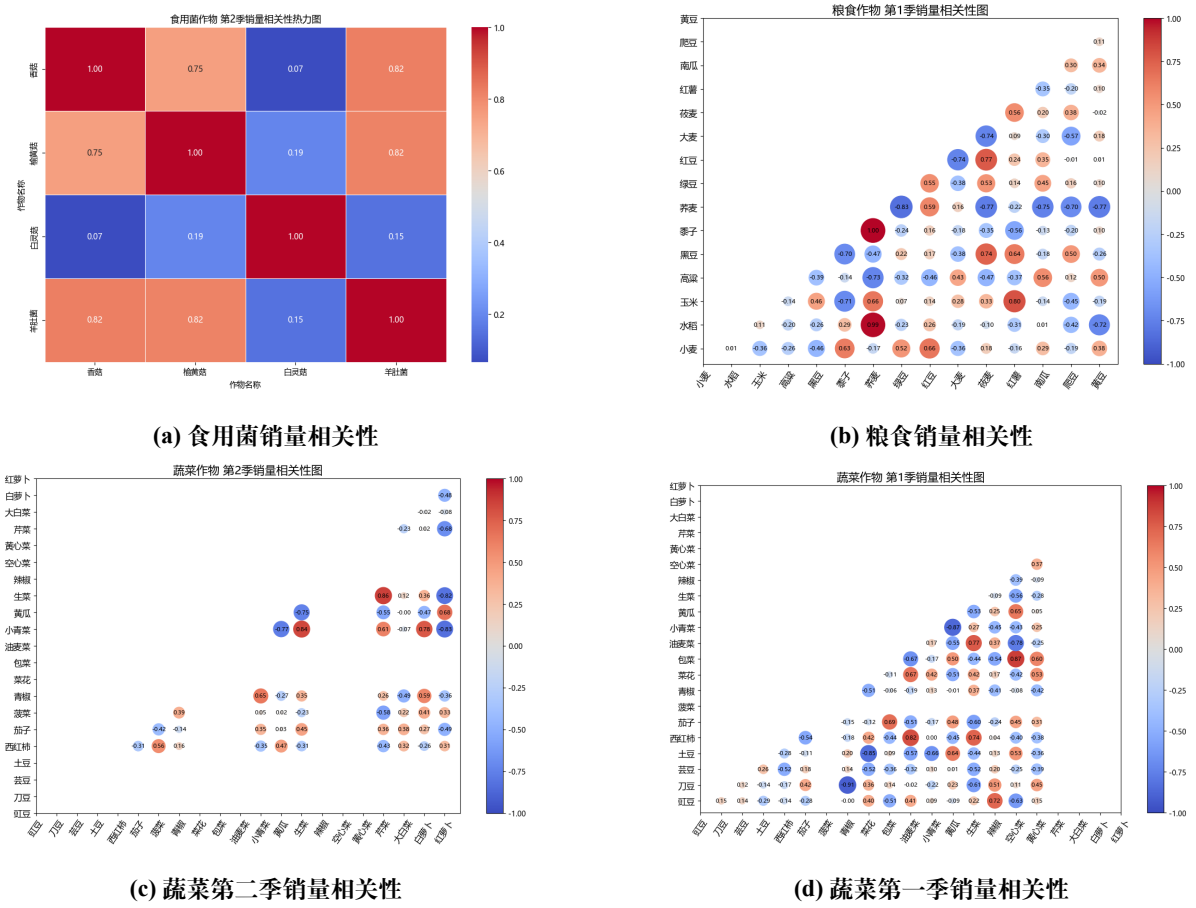


图 12　食用菌需求曲线

从图10、图11和图12中可以看出，在粮食作物的需求曲线中，大麦和玉米的需求总体上升，且走势类似，表明它们是互补性作物，适合共同种植。而水稻和豆类作物不随其他粮食作物的需求波动，说明它们可能是独立种植作物，不与其他作物互补。而榆黄菇和香菇的需求变化趋势接近，在部分年份，两者呈现出相似的下降和上升趋势，说明它们有一定的互补性，适合在同一时期的市场中销售。此外，大白菜和胡萝卜、西红柿和茄子需求变化趋势相似，表明它们是互补作物，适合交替种植。黄瓜和青椒的销量波动具有反向趋势，当黄瓜的销量下降时，青椒的销量上升，表明它们可能是潜在的替代作物。



(a) 食用菌销量相关性

(b) 粮食销量相关性

(c) 蔬菜第二季销量相关性

(d) 蔬菜第一季销量相关性

图 13　不同作物销量相关性图

如图13所示，生菜与小青菜（0.84）、大白菜与红萝卜（0.86）、黍子与荞麦（0.80）、谷子与高粱（0.64）、香菇与榆黄菇（0.75）以及香菇与羊肚菌（0.82）具有较强的正相关性，表明它们应该是互补关系。而小麦与水稻（-0.70）、小麦与黍子（-0.77）、小麦与荞麦（-0.72）、油麦菜与生菜（-0.77）表现出显著的负相关性，说明当一种作物的销量上升时，另一种作物的销量下降，表明它们之间有替代性。

综合来看，通过相关性分析和需求曲线分析，可以帮助种植者根据作物之间的互补性和替代性调整种植策略。

8.3 预期销售量与销售价格、种植成本的相关性

为了探索预期销售量与种植成本、销售价格之间的相关性，我们选择通过回归分析来定量研究三者。综合历史数据和问题二中预测的成本、销售价格、销售量等数据，为了获得更加符合实际的回归系数，我们使用 Ridge 回归模型来分析这些相关性，得到的结果如表7所示。

从表中可以看出，粮食作物受的销售单价到种植成本和销量影响小，而食用菌作物和蔬菜作物的自变量对销售单价有较强的解释能力，因此种植成本和总销量对价格的影

表 7　不同作物类型的 Ridge 回归系数与截距

作物类型	自变量	回归系数	截距
粮食作物	成本、销量	$[3.5926 \times 10^{-3}, -4.3772 \times 10^{-6}]$	0.7580
蔬菜作物	销售单价、销量	$[1.2804 \times 10^{2}, -2.7001 \times 10^{-3}]$	841.6876
食用菌作物	销售单价、成本	$[-44.7668, -0.7748]$	16607.0482

响较为显著。

8.4 种植策略的制定

我们基于以下规则制定种植策略：

- **互补性作物：**这些作物共同种植能带来协同效应，提高总产出和市场适应性。
- **替代性作物：**这些作物之间具有竞争关系，适合在需求波动时交替种植，以降低市场风险。
- **成本与销售价格的相关性：**根据 Ridge 回归模型，种植策略应确保在满足市场需求的同时最大化经济收益。

替代性是指当某一种作物需求或销量上升时，另一种作物需求或销量下降，这类作物具有相互制约性，因此当作物 A 和作物 B 具有替代性关系，可以建立以下约束条件：

$$X_{A,j,k,m} + X_{B,j,k,m} \leq 1 \quad \forall j, k, m$$

这个条件确保了避免在一块地上同时种植替代性作物。

互补性体现在这些作物可以共同种植以产生协同效应，因此，我们建立以下约束条件：

$$X_{C,j,k,m} = X_{D,j,k,m} \quad \forall j, k, m$$

此约束条件确保作物 C 和 D 在同一块地、同一时间段中共同种植。

而针对 Ridge 回归模型，我们可以得到方程：

$$P_{i,k,m} = \alpha_i C_{i,k,m} + \beta_i$$

其中：

- $P_{i,k,m}$ 表示作物 i 在年份 k，季节 m 的销售价格；
- $C_{i,k,m}$ 表示作物 i 在年份 k，季节 m 的种植成本；

- α_i 是 Ridge 回归得到的回归系数;
- β_i 是 Ridge 回归得到的截距。

约束条件构建如下:

$$P_{i,k,m} = \alpha_i C_{i,k,m} + \beta_i \quad \forall i, k, m$$

8.5 问题三综合模型建立

- **Step1.** 综合上一问中相关性分析、需求曲线分析、回归分析得到的结果，引入新的相关性约束。
- **Step2.** 基于回归分析的作物种植成本、销量和单价的动态模型，融入到原有模型约束中。
- **Step3.** 基于混合整数规划和模拟退火算法的组合方法，在多种情况下找到全局最优解。
- **Step4.** 找到最终的种植策略，分析种植作物可替代性和互补性对种植策略的影响。

8.6 问题三模型求解与分析

通过一系列分析，我们制定如下具体的种植策略:

通过需求曲线和相关性分析，互补性作物如大麦与玉米、香菇与榆黄菇应保持合理的种植比例，而替代性作物如黄瓜与青椒应避免同时大量种植，以分散市场风险。而基于 Ridge 回归的分析结果，我们需要进一步优化种植策略，确保在经济效益和市场供需之间找到最佳平衡。因此，我们给出如下的简单约束公式:

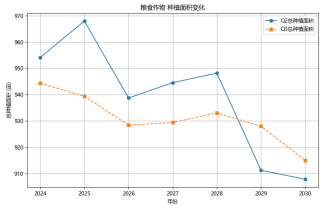
$$\begin{aligned} A_{\text{大麦}} + A_{\text{玉米}} &\geq 0.3 \times A_{\text{总面积}} \\ A_{\text{黄瓜}} + A_{\text{青椒}} &\leq 0.5 \times A_{\text{总面积}} \\ A_{\text{生菜}} + A_{\text{小青菜}} &\geq 0.2 \times A_{\text{总面积}} \\ A_{\text{小麦}} + A_{\text{水稻}} &= \text{常数} \end{aligned} \tag{15}$$

大麦和玉米作为互补作物，其需求变化趋势相似，因此适合共同种植。而黄瓜和青椒的销量呈现替代性，限制两者的总种植面积可以分散市场风险，避免因市场需求波动导致的经济损失。考虑到生菜和小青菜的销量具有较强的正相关性，设定较高的种植面积下限可以保证需求较高时不会出现供应不足的情况。小麦和水稻的替代性关系要求两者的总种植面积保持稳定。通过 Ridge 回归分析得出的销售单价、种植成本与销售量之间的关系，在确保蔬菜作物种植方案在经济效益上最大化的同时，满足市场需求。

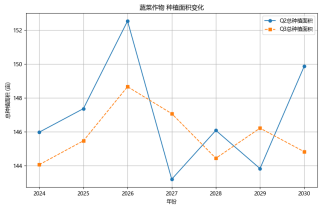
最终我们得到了考虑作物之间的替代性和互补性和预期销售量与销售价格、种植成本之间的相关性的情况下,最大收益为 **42889653.65 元**,相较于问题二的收益 42784489.60

元有所提升。通过分析作物的替代性和互补性，可以看出这些因素对种植策略有显著影响，不仅优化了资源配置，还提升了整体收益。

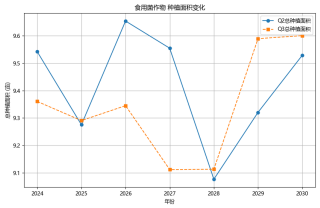
8.7 问题三与问题二结果对比



(a) 问题二三粮食种植面积对比



(b) 问题二三蔬菜种植面积对比

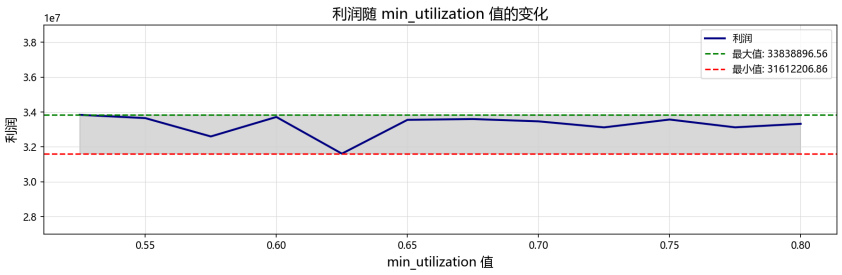


(c) 问题二三食用菌种植面积对比

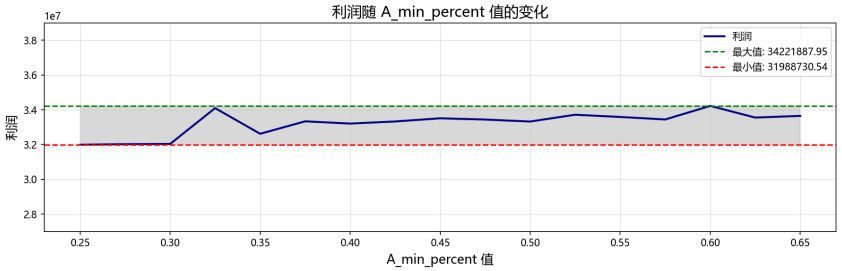
图 14　问题二三对比

从图14中可以看出，粮食作物种植面积上，问题二策略在 2025 年种植面积大幅度上升，而问题三策略保持平稳，反映了问题二策略更注重短期粮食的高产，而问题三策略更加关注持续稳定的种植。而食用菌的数据显示，问题二对市场波动敏感，尤其是 2025 年到 2028 年，而问题三的策略则表现出更强的稳定性。综合对比来看，问题三的策略比问题二更加的稳健，往往选择长期稳定的规划来减少波动保持市场供应的平衡。

九、 模型灵敏度分析



(a) 每个地块种植作物总面积最少占地块面积的百分比灵敏度分析



(b) 某种作物在地块上的最小种植面积占地块面积的百分比灵敏度分析

图 15　灵敏度分析

灵敏度分析是一种研究和分析模型对周围条件及输入参数改变的敏感程度的有效方法。对于最优化问题而言，灵敏度分析可以很好地研究评估当原始数据无法准确确定或其发生变化时，模型求得最优解的稳定性。

由图15a可知，当某种作物在地块上的最小种植面积占地块面积 A_min_pre 保持为 50% 时，我们给每个地块种植作物面积最少占地块面积 min_utilization 加入一些的扰动，输出的结果保持在 3161 万至 3384 万之间，波动幅度小于 7%；同样，由图15b可知，保持每个地块种植作物面积最少占地块面积 min_utilization 为 80% 时，给某种作物在地块上的最小种植面积占地块面积 A_min_pre 加入一些扰动，输出的结果保持在 3199 万至 3422 万之间，波动幅度小于 7%。据此，可认为**本模型普适性较强**，受干扰后变化并不明显，可以应对多种不同的情况。

十、 模型评价

10.1 模型的优点

- **灵活性与拓展性.** 通过将混合整数规划和模拟退火算法结合，本模型具有较高的灵活性和可拓展性，能够处理多个不同类型的约束条件（如轮作要求、种植条件、田间管理等方面），并且可以通过改变参数来模拟在不同的不同的场景和种植规划的情况中的最大化收益。
- **准确率与效率.** 由于问题约束条件众多，求解最大化收益的时候往往会陷入效率的困境。本模型创新性地将模拟退火算法和混合整数规划结合起来，在较大规模和复杂的决策空间中有效探索，查找全局最优解。通过实验验证，该模型在多约束条件的优化问题中取得了良好的准确率和性能。
- **鲁棒性与全面性.** 本模型采用分布式鲁棒优化方法来在模型中引入区间不确定性和风险容忍度，确保了在所有可能的情况下求解出的最优种植策略具有鲁棒性。灵敏度分析实验证明，本模型克服了传统模拟退火模型对参数的敏感性，考虑到不确定性因素和潜在风险，模型在鲁棒性大大提升的同时增强了模型的全面性。

10.2 模型的缺点

- **参数依赖性.** 模拟退火的使用导致模型依赖于算法参数的设置，如初始温度、冷却速率等。如果参数设置不当，可能导致陷入局部最优解等问题。

参考文献

[1] 买买提 · 海力力. 基于混合整数规划的生鲜农产品分销网络优化研究[J]. 价值工程, 2024, (09), 51-53.

[2] 高晓松, 李更丰, 肖遥, 别朝红. 基于分布鲁棒优化的电–气–热综合能源系统日前经济调度[J]. 电网技术, 2020, (06), 2245-2254. 10.13335/j.1000-3673.pst.2019.2356.

[3] Bertsimas, D., & Tsitsiklis, J. Simulated annealing[J]. Statistical science, 1993, 8(1), 10-15.

[4] 陈美, 丁恒平. 我国农业种植结构调整存在的问题及优化策略[J]. 乡村科技, 2017, (33), 17-18. 10.19345/j.cnki.1674-7909.2017.33.002.

附录 A　支撑材料列表

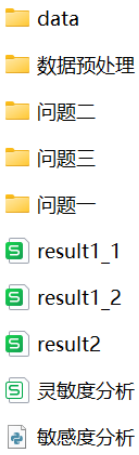


图 16　主文件夹

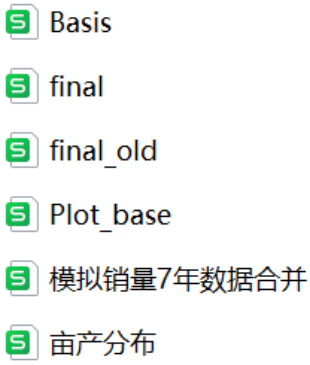


图 17　data 文件夹

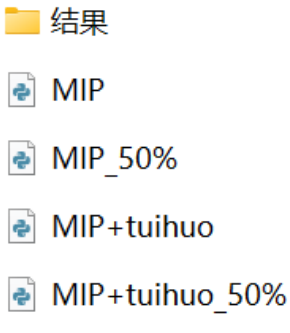


图 18　问题一文件夹

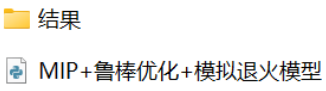


图 19　问题二文件夹

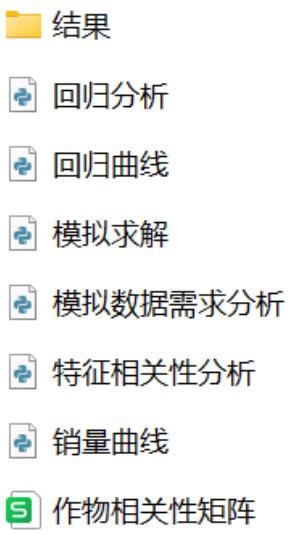


图 20　问题三文件夹

附录 B　数据预处理

2.1　附件拼接形成可用数据文件

```python
import pandas as pd

# 加载两个Excel文件的相关表格
file_path_1 = 'data/附件1.xlsx'
file_path_2 = 'data/新_附件2.xlsx'
```

```
# 读取两个文件中相关的工作表
df1 = pd.read_excel(file_path_1, sheet_name='乡村的现有耕地')
df2 = pd.read_excel(file_path_2, sheet_name='2023年的农作物种植情况')

# 去除可能存在的多余空格
df1['地块名称'] = df1['地块名称'].str.strip()
df1['地块类型'] = df1['地块类型'].str.strip()

df2['种植地块'] = df2['种植地块'].str.strip()
df2['作物名称'] = df2['作物名称'].str.strip()

# 处理附件2中的合并单元格问题，将种植地块的缺失值填充为前一个有效值
df2['种植地块'].fillna(method='ffill', inplace=True)

# 根据“地块名称”和“种植地块”进行匹配，并添加“地块类型”和“地块面积/亩”列
df_corrected_merge = pd.merge(df2, df1[['地块名称', '地块类型', '地块面积/亩']],
    left_on='种植地块', right_on='地块名称', how='left')

# 删除多余的“地块名称”列
df_corrected_merge.drop(columns=['地块名称'], inplace=True)

# 加载附加的表格 "2023年统计的相关数据"
df_additional = pd.read_excel(file_path_2, sheet_name='2023年统计的相关数据')

# 去除多余的空格
df_additional['作物名称'] = df_additional['作物名称'].str.strip()
df_additional['地块类型'] = df_additional['地块类型'].str.strip()

# 选择需要的列用于合并
df_additional_subset = df_additional[['作物名称', '地块类型', '种植季次', '亩产量/斤',
    '种植成本/(元/亩)', '销售单价/(元/斤)']]

# 合并附加数据，根据“作物名称”、“地块类型”和“种植季次”进行匹配
df_final_merge = pd.merge(df_corrected_merge, df_additional_subset, on=['作物名称', '地块类型',
    '种植季次'], how='left')

# 计算“平均销售单价”
df_final_merge[['销售单价最小值', '销售单价最大值']] =
    df_final_merge['销售单价/(元/斤)'].str.split('-', expand=True)

# 将分割出的最小值和最大值转换为浮点数
df_final_merge['销售单价最小值'] = pd.to_numeric(df_final_merge['销售单价最小值'],
    errors='coerce')
df_final_merge['销售单价最大值'] = pd.to_numeric(df_final_merge['销售单价最大值'],
    errors='coerce')

# 计算平均销售单价
```

```
df_final_merge['平均销售单价'] = (df_final_merge['销售单价最小值'] +
    df_final_merge['销售单价最大值']) / 2

# 为每个地块添加编号
df1['地块编号'] = range(1, len(df1) + 1)

# 将地块编号合并到数据中
df_final_merge_with_plot_number = pd.merge(df_final_merge, df1[['地块名称', '地块编号']],
    left_on='种植地块', right_on='地块名称', how='left')
df_final_merge_with_plot_number.drop(columns=['地块名称'], inplace=True)

# 添加“是否为豆类”列
df_final_merge_with_plot_number['是否为豆类'] =
    df_final_merge_with_plot_number['作物类型'].apply(lambda x: 1 if '豆类' in str(x) else 0)

# 调整列顺序，将“地块编号”移到第一列，并将“是否为豆类”列放在“作物类型”之后
cols = df_final_merge_with_plot_number.columns.tolist()
豆类_index = cols.index('作物类型') + 1
cols.insert(豆类_index, cols.pop(cols.index('是否为豆类')))
cols = ['地块编号'] + [col for col in cols if col != '地块编号'] # 将“地块编号”放到第一列
df_final_merge_with_plot_number = df_final_merge_with_plot_number[cols]

# 保存最终结果到Excel文件
output_file_path = 'data/附件拼合.xlsx'
df_final_merge_with_plot_number.to_excel(output_file_path, index=False)

print(f"文件已保存至 {output_file_path}")
```

2.2 特征相关性分析

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import LabelEncoder

# 设置字体为“Microsoft YaHei”
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei'] # 设置中文字体
plt.rcParams['axes.unicode_minus'] = False # 解决负号显示问题

# 从 Excel 文件中加载数据
data = pd.read_excel('./../data/final.xlsx') # 请确保文件路径正确

# 目标变量（销售单价）
target = '销售单价（均值）'
```

```
# 检查并转换所有非数值型列为数值型
# 通过 LabelEncoder 处理 "地块类型" 和 "作物类型"
le = LabelEncoder()
data['地块类型 '] = le.fit_transform(data['地块类型'])
data['作物类型 '] = le.fit_transform(data['作物类型'])

# 假设“种植季次”是一个字符串类型的列（例如“单季”），我们需要对其编码
data['种植季次 '] = le.fit_transform(data['种植季次'])

# 选择需要进行相关性分析的特征（已编码后的列）
features_encoded = ['地块类型 ', '作物类型 ', '是否为豆类', '种植面积/亩', '种植季次 ',
    '亩产量/斤', '种植成本/(元/亩)']
data_encoded = data[features_encoded]

# 添加目标变量（销售单价）
data_encoded[target] = data[target]

# 确保没有缺失值或非数值内容
data_encoded = data_encoded.apply(pd.to_numeric, errors='coerce')

# 计算相关性矩阵
correlation_matrix = data_encoded.corr()

# 设置图形大小
plt.figure(figsize=(12, 10))

# 生成一个掩码，用于只显示相关矩阵的下三角部分
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# 使用掩码遮挡上三角部分
correlation_matrix = correlation_matrix.mask(mask)

# 绘制相关性矩阵图，使用圆圈大小来表示相关性强度
correlation_matrix = correlation_matrix.round(2)
corr = correlation_matrix.values

# 生成网格
x = np.arange(corr.shape[0])
y = np.arange(corr.shape[1])

# 创建网格图，scatter绘制每个点
plt.scatter(x=np.repeat(x, len(y)), y=np.tile(y, len(x)),
          s=np.abs(corr.flatten()) * 1500, # 圆圈大小根据相关性强度
          c=corr.flatten(), cmap='coolwarm', vmin=-1, vmax=1) # 圆圈颜色根据相关性值

# 在每个圆圈上显示对应的相关性数值，跳过 NaN 值
for i in range(corr.shape[0]):
```

```
    for j in range(i+1): # 只显示下三角部分
        if not np.isnan(corr[i, j]): # 跳过 NaN
            plt.text(i, j, f'{corr[i, j]:.2f}', ha='center', va='center', color='black',
                fontsize=8) # 数字字号减少为 8

# 设置刻度和标签
plt.xticks(np.arange(len(correlation_matrix.columns)), correlation_matrix.columns,
    rotation=55, fontsize=12)
plt.yticks(np.arange(len(correlation_matrix.columns)), correlation_matrix.columns, fontsize=12)

# 设置标题
plt.title('特征与销售单价的相关性图', fontsize=16)

# 显示颜色条
plt.colorbar()

# 调整布局
plt.tight_layout()

# 显示图形
plt.show()
```

2.3 农作物亩产量可视化分析

```
import pandas as pd
import matplotlib.pyplot as plt

# 加载数据
file_path = 'data/亩产分布.xlsx'
data = pd.read_excel(file_path, sheet_name='2023年统计的相关数据')

# 过滤数据
ping_data = data[data['地块类型'] == '平旱地']
tian_data = data[data['地块类型'] == '梯田']
shan_data = data[data['地块类型'] == '山坡地']

plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
# 自定义颜色列表
custom_colors = ['#934B43', '#D76364', '#EF7A6D', '#F1D77E', '#F3D266','#BB9727',
    '#54B345','#B1CE46','#32B897','#63E398','#9394E7','#C76DA2','#5F97D2','#9DC3E7','#05B9E2']
    # 调整颜色

# 绘制自定义的柱状图
def plot_refined_yield_bar_chart(data, title):
    plt.figure(figsize=(8, 6))
```

```
    # 如果数据量大于颜色数量，重复颜色
    colors = custom_colors * (len(data['作物名称']) // len(custom_colors) + 1)

    # 绘制柱状图并为每个柱子加边框
    bars = plt.bar(data['作物名称'], data['亩产量/斤'], color=colors[:len(data['作物名称'])],
         edgecolor='black')

    # 在每个柱子上方添加数值标签
    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), ha='center', va='bottom')

    # 设置标题和标签
    plt.title(f'{title}各种农作物亩产量')
    plt.xticks(rotation=45)
    plt.ylabel('亩产量（斤）')

    # 隐藏顶部和右侧的坐标轴线，只保留左侧（数字轴）
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)

    # 添加外框
    plt.gca().add_patch(plt.Rectangle((0, 0), 1, 1, fill=False, edgecolor='black', lw=2,
         transform=plt.gca().transAxes))

    # 调整布局并显示图表
    plt.tight_layout()
    plt.show()

# 分别绘制平旱地、梯田、山坡地的图表
plot_refined_yield_bar_chart(ping_data, '平旱地')
plot_refined_yield_bar_chart(tian_data, '梯田')
plot_refined_yield_bar_chart(shan_data, '山坡地')
```

2.4 作物价格的异常值检测

```
import pandas as pd
import matplotlib.pyplot as plt

# 加载数据集
file_path = 'data/附件拼合.xlsx'
df = pd.read_excel(file_path, sheet_name='Sheet1')

# 映射作物类型到分类
```

```
crop_types_mapping = {
    '粮食': '粮食',
    '粮食（豆类）': '粮食',
    '蔬菜': '蔬菜',
    '蔬菜（豆类）': '蔬菜',
    '食用菌': '食用菌'
}
df['作物类型分类'] = df['作物类型'].map(crop_types_mapping)

# 筛选出有效的作物类型数据
df_filtered = df.dropna(subset=['作物类型分类'])

# 设置字体为微软雅黑
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']

# 绘制粮食的箱型图
plt.figure(figsize=(6, 6))
df_grain = df_filtered[df_filtered['作物类型分类'] == '粮食']
plt.boxplot(df_grain['平均销售单价'], labels=['粮食'], patch_artist=True,
    boxprops=dict(facecolor='#9FBA95'))
plt.title('粮食平均销售单价的箱型图')
plt.ylabel('平均销售单价（元/斤）')
plt.show()

# 绘制蔬菜的箱型图
plt.figure(figsize=(6, 6))
df_vegetable = df_filtered[df_filtered['作物类型分类'] == '蔬菜']
plt.boxplot(df_vegetable['平均销售单价'], labels=['蔬菜'], patch_artist=True,
    boxprops=dict(facecolor='#E6CECF'))
plt.title('蔬菜平均销售单价的箱型图')
plt.ylabel('平均销售单价（元/斤）')
plt.show()

# 绘制食用菌的箱型图
plt.figure(figsize=(6, 6))
df_mushroom = df_filtered[df_filtered['作物类型分类'] == '食用菌']
plt.boxplot(df_mushroom['平均销售单价'], labels=['食用菌'], patch_artist=True,
    boxprops=dict(facecolor='#B696B6'))
plt.title('食用菌平均销售单价的箱型图')
plt.ylabel('平均销售单价（元/斤）')
plt.show()
```

附录 C　问题一的模型建立与求解

3.1　MIP 模型

```
import pandas as pd
import pulp as pl
import numpy as np

# 加载数据
final_data = pd.read_excel('./data/final.xlsx') # 使用你上传的文件
Basic_data = pd.read_excel('./data/Basis.xlsx')
plot_areas_data = pd.read_excel('./data/Plot_base.xlsx')

# 提取相关数据用于模型构建
crop_ids = Basic_data['作物编号'].unique() # 作物编号
plot_ids = plot_areas_data['地块编号'].unique() # 地块编号
season_ids = Basic_data['种植季次编号'].unique() # 种植季次编号
num_years = 7 + 1 # 我们计划的年限是7年（从2024到2030）

# 地块类型分类
plot_types = {
    '平旱地': list(range(1, 7)),
    '梯田': list(range(7, 21)),
    '山坡地': list(range(21, 27)),
    '水浇地': list(range(27, 35)),
    '普通大棚': list(range(35, 51)),
    '智慧大棚': list(range(51, 55))
}
# 地块类型编号
basic_plot_types = {
    '平旱地': 1,
    '梯田': 2,
    '山坡地': 3,
    '水浇地': 4,
    '普通大棚': 5,
    '智慧大棚': 6
}


# 使用映射来关联地块编号和地块类型编号
def get_plot_type(j):
    for plot_type, plot_list in plot_types.items():
        if j in plot_list:
            return basic_plot_types[plot_type]
    return None
```

```
# 作物类型分类
crop_types = {
    '粮食作物': list(range(1, 17)), # 1-16 为粮食作物
    '蔬菜作物': list(range(17, 38)), # 17-37 为蔬菜作物
    '食用菌作物': list(range(38, 42)) # 38-41 为食用菌作物
}

# 打印映射结果，检查映射是否正确
# print(plot_type_to_plots)

# 构建作物数据的字典
crop_names = dict(zip(Basic_data['作物编号'], Basic_data['作物名称'])) # 作物编号和名称的映射
plot_names = dict(zip(plot_areas_data['地块编号'], plot_areas_data['地块名称'])) #
    地块编号和名称的映射

# 首先对相同作物编号、地块类型、种植季次编号的对应销量求和
grouped_data = final_data.groupby(['作物编号', '种植季次编号'])['对应销量'].sum().reset_index()

# 生成字典
crop_sales_limits = dict(zip(zip(grouped_data['作物编号'], grouped_data['种植季次编号']),
    grouped_data['对应销量']))
plot_areas = dict(zip(plot_areas_data['地块编号'], plot_areas_data['地块面积/亩'])) # 地块面积

# 使用映射来关联地块编号和地块类型编号
crop_yields = dict(zip(
    zip(Basic_data['作物编号'], Basic_data['地块类型编号'], Basic_data['种植季次编号']),
    Basic_data['亩产量/斤']
))

crop_costs = dict(zip(
    zip(Basic_data['作物编号'], Basic_data['地块类型编号'], Basic_data['种植季次编号']),
    Basic_data['种植成本/(元/亩)']
))

crop_prices = dict(zip(
    zip(Basic_data['作物编号'], Basic_data['种植季次编号']),
    Basic_data['销售单价(均值)']
))

# 初始化优化模型 (最大化利润)
model = pl.LpProblem("Crop_Planning_Optimization", pl.LpMaximize)

# 决策变量
X = pl.LpVariable.dicts("X", (crop_ids, plot_ids, range(num_years), season_ids), cat='Binary')
    # 是否种植的二元变量
A = pl.LpVariable.dicts("A", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0) #
```

```
    种植面积的连续变量
Z = pl.LpVariable.dicts("Z", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0)

# 为Z添加约束条件，按照地块编号 j 确定地块类型编号
for i in crop_ids:
    for j in plot_ids:
        plot_type_id = get_plot_type(j) # 获取地块类型编号
        for k in range(1, num_years): # 从k=1开始，忽略k=0
            for m in season_ids:
                # 使用地块类型编号来访问 crop_yields 中的对应值
                model += Z[i][j][k][m] <= A[i][j][k][m] * crop_yields.get((i, get_plot_type(j),
                    m), 0) # Z <= A * 单产量

# 1. 引入新的变量 Z_total 用于表示作物的总销售量
Z_total = pl.LpVariable.dicts("Z_total", (crop_ids, range(num_years), season_ids), lowBound=0)

# 2. 对每个作物的总销售量添加约束，确保其不超过预测销量
for i in crop_ids:
    for k in range(1, num_years):
        for m in season_ids:
            model += Z_total[i][k][m] == pl.lpSum([Z[i][j][k][m] for j in plot_ids]) # Z_total
                是所有地块销量的总和
            model += Z_total[i][k][m] <= crop_sales_limits.get((i, m), 0) # Z_total 不超过销量上限

# 3. 修改目标函数：使用 Z_total 计算总利润
model += pl.lpSum([
    Z_total[i][k][m] * crop_prices.get((i, m), 0) # 使用 Z_total 计算利润
    - pl.lpSum([A[i][j][k][m] * crop_costs.get((i, get_plot_type(j), m), 0) for j in plot_ids])
        # 计算总成本
    for i in crop_ids
    for k in range(1, num_years) # 从k=1到k=7
    for m in season_ids
])

# 记录已经在final_data中指定的2023年种植组合
specified_combinations = set()

# 首先处理已指定的种植计划 (k=0)
for _, row in final_data.iterrows():
    i = row['作物编号']
    j = row['地块编号']
    m = row['种植季次编号']
    # 设置X为1，表示种植
    model += X[i][j][0][m] == 1
    # 设置种植面积
    model += A[i][j][0][m] == row['种植面积/亩']
    # 记录该组合
```

```
    specified_combinations.add((i, j, 0, m))

# 对于未指定的组合，将X和A置为0 (k=0)
for i in crop_ids:
    for j in plot_ids:
        for m in season_ids:
            if (i, j, 0, m) not in specified_combinations:
                model += X[i][j][0][m] == 0
                model += A[i][j][0][m] == 0

# 添加约束：如果 X 为 0，则 A 必须为 0
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years):
            for m in season_ids:
                model += A[i][j][k][m] <= plot_areas[j] * X[i][j][k][m]

# 约束条件
# 1. 每块地在3年内应至少种植豆类作物
for j in plot_ids:
    for t in range(1, num_years - 1): # 从k=1开始
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for k in range(t - 1, t + 2) for m in
            season_ids if
                      final_data.loc[final_data['作物编号'] == i, '是否为豆类'].values[0] == 1)
                          >= plot_areas[j]

# 2. 限制每个季节种植某种作物的地块数量 (k=1到k=7)
N_max = 5
for i in crop_ids:
    for k in range(1, num_years):
        for m in season_ids:
            model += pl.lpSum(X[i][j][k][m] for j in plot_ids) <= N_max

# 3. 每块地的最小种植面积 (k=1到k=7)
A_min_percent = 0.5 # 假设每个地块至少种植50%的面积
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years):
            for m in season_ids:
                model += A[i][j][k][m] >= A_min_percent * plot_areas[j] * X[i][j][k][m]

# 4. 总种植面积不能超过地块的可用面积 (k=1到k=7)
min_utilization = 0.8 # 每个地块至少使用80%的面积
for j in plot_ids:
    for k in range(1, num_years):
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for m in season_ids) >=
            min_utilization * plot_areas[j]
```

```
# 5. 各种地块类型的种植条件限制 (k=1到k=7)
# 平旱地、梯田和山坡地只能种植粮食作物，且相邻年份不能种植相同的作物，且只能种单季 m=0
for plot_category in ['平旱地', '梯田', '山坡地']:
    for j in plot_types[plot_category]: # 遍历平旱地、梯田和山坡地
        # 限制只能种单季 m=0
        for k in range(1, num_years): # 遍历每一年
            for m in [1, 2]: # 限制双季种植 (m=1 和 m=2)
                # 不能在双季种植任何作物
                for i in crop_ids:
                    model += X[i][j][k][m] == 0

            # 只能种粮食作物
            for i in crop_types['蔬菜作物'] + crop_types['食用菌作物']: #
                禁止蔬菜和食用菌作物在这些地块种植
                model += X[i][j][k][0] == 0 # 禁止在单季种植非粮食作物

        # 对粮食作物的约束
        for i in crop_types['粮食作物']: # 允许种植粮食作物
            for k in range(1, num_years): # 遍历每一年
                # 限制不能种水稻 (编号16)
                if i == 16: # 水稻的编号是16
                    model += X[i][j][k][0] == 0 # 禁止水稻在这些地块种植
            for k in range(num_years - 1):
                # 限制相邻年份同一地块不能连续种植同一种粮食作物
                model += X[i][j][k][0] + X[i][j][k + 1][0] <= 1

# 水浇地可以一年种植一季水稻，且相邻年份水稻不能重茬
# 限制双季 (m=1 和 m=2) 不允许种粮食作物、食用菌作物
# 水浇地可以第一季种植蔬菜 (大白菜、白萝卜和红萝卜除外)，第二季只能种大白菜、白萝卜和红萝卜其中一种
for j in plot_types['水浇地']: # 遍历所有水浇地
    for k in range(1, num_years):
        # 限制只能在单季种植水稻 (编号16)，且相邻年份水稻不能连续种植
        model += X[16][j][k][1] == 0 # 不能按两季种
        model += X[16][j][k][2] == 0

        # 其他粮食作物 (除水稻外) 不能在水浇地种植
        for i in crop_types['粮食作物']:
            if i != 16: # 除水稻外的粮食作物
                for m in [0, 1, 2]: # 限制全年不种植
                    model += X[i][j][k][m] == 0

        # 第一季可以种蔬菜，但不允许种大白菜 (35)、白萝卜 (36)、红萝卜 (37)
        for i in [35, 36, 37]: # 限制特定蔬菜
            model += X[i][j][k][1] == 0 # 第一季不允许种植大白菜 (35)、白萝卜 (36)、红萝卜 (37)
            model += X[i][j][k][0] == 0
```

```
        for i in crop_types['蔬菜作物']: # 第二季只能种植大白菜（35）、白萝卜（36）、红萝卜（37）
            if i != [35, 36, 37]: # 仅允许这三种蔬菜在第二季种植
                model += X[i][j][k][2] == 0 # 第二季不能种植其他蔬菜
                model += X[i][j][k][0] == 0
        model += X[35][j][k][2] + X[36][j][k][2] + X[37][j][k][2] <= 1

        # 食用菌作物不能在水浇地种植
        for i in crop_types['食用菌作物']:
            for m in [0, 1, 2]: # 全年不允许食用菌作物
                model += X[i][j][k][m] == 0

    # 相邻年份不能连续种植水稻
    for k in range(num_years - 1):
        model += X[16][j][k][0] + X[16][j][k + 1][0] <= 1

#
    普通大棚每年适宜种植一季蔬菜(35,36,37除外)和一季食用菌且蔬菜只能在第一季种植、食用菌只能在第二季种植
for j in plot_types['普通大棚']: # 遍历普通大棚
    for k in range(1, num_years):
        for i in crop_types['蔬菜作物']: # 蔬菜作物
            if i in [35, 36, 37]:
                model += X[i][j][k][1] == 0
            # 限制蔬菜只能在第一季种植
            model += X[i][j][k][2] == 0
            model += X[i][j][k][0] == 0
        for i in crop_types['粮食作物']: # 粮食作物
            # 限制粮食不能种植
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            # 限制食用菌只能在第二季种植
            model += X[i][j][k][1] == 0
            model += X[i][j][k][0] == 0

# 智慧大棚每年种植两季蔬菜(35,36,37除外)，且相邻季节不能种植相同的作物
for j in plot_types['智慧大棚']: # 遍历智慧大棚
    for i in crop_types['蔬菜作物']: # 蔬菜作物
        for k in range(num_years - 1):
            # 相邻年份不能重茬
            model += X[i][j][k][2] + X[i][j][k + 1][1] <= 1
        for k in range(num_years):
            # 同年不能种植相同的蔬菜
            model += X[i][j][k][1] + X[i][j][k][2] <= 1

    for k in range(1, num_years):
        for i in crop_types['蔬菜作物']: # 蔬菜作物不能种单季
            model += X[i][j][k][0] == 0
```

```
            if i in [35, 36, 37]: # 蔬菜中的大白菜（35）、白萝卜（36）、红萝卜（37）不能种
                model += X[i][j][k][1] == 0
                model += X[i][j][k][2] == 0
        for i in crop_types['粮食作物']: # 粮食作物
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0

# 求解模型
solver = pl.PULP_CBC_CMD(msg=True, threads=16,timeLimit=6)
model.solve(solver)

# 检查模型状态
status = pl.LpStatus[model.status]
print(f"模型求解状态: {status}")

# 输出目标函数的值（利润）
if status == "Optimal":
    best_profit = pl.value(model.objective)
    print(f"\n最优利润: {best_profit}")
else:
    print(f"\n未找到最优解。状态: {status}")

# 循环每一年，保存2024到2030年的结果为单独的csv文件
if status == "Optimal":
    start_year = 2023 # 从2024年开始
    for k in range(1, num_years):
        year_solution = []

        for i in crop_ids:
            for j in plot_ids:
                for m in season_ids:
                    # 收集每一年的决策变量值
                    x_value = pl.value(X[i][j][k][m])
                    a_value = pl.value(A[i][j][k][m])

                    # 如果种植面积大于 0，提取对应的单价、亩产量和成本
                    if a_value is not None and a_value > 0:
                        unit_price = crop_prices.get((i, m), 0) # 单价
                        yield_per_mu = crop_yields.get((i, get_plot_type(j), m), 0) # 亩产量
                        cost_per_mu = crop_costs.get((i, get_plot_type(j), m), 0) # 种植成本
                        # 将信息添加到year_solution
                        year_solution.append(
                            [crop_names[i], plot_names[j], m, x_value, a_value, unit_price,
                                yield_per_mu, cost_per_mu])
```

```
    # 将结果转换为DataFrame
    year_solution_df = pd.DataFrame(year_solution,
                                    columns=['作物名称', '地块名称', '种植季次', '种每种',
                                        '种植面积/亩',
                                            '单价（元/斤）', '亩产量（斤/亩）',
                                                '种植成本（元/亩）'])

    # 保存当前年份的解决方案为CSV文件
    year_solution_df.to_csv(f"问题一/结果/crop_plan_{start_year + k}.csv", index=False,
        encoding='utf-8-sig')
    print(f"{start_year + k}年种植计划已保存为CSV文件")
```

3.2 MIP+ 模拟退火模型

```
import pandas as pd
import pulp as pl
import numpy as np
import random
import math

# 加载数据
final_data = pd.read_excel('./data/final.xlsx') # 使用你上传的文件
Basic_data = pd.read_excel('./data/Basis.xlsx')
plot_areas_data = pd.read_excel('./data/Plot_base.xlsx')

# 提取相关数据用于模型构建
crop_ids = Basic_data['作物编号'].unique() # 作物编号
plot_ids = plot_areas_data['地块编号'].unique() # 地块编号
season_ids = Basic_data['种植季次编号'].unique() # 种植季次编号
num_years = 7 + 1 # 我们计划的年限是7年（从2024到2030）

# 地块类型分类
plot_types = {
    '平旱地': list(range(1, 7)),
    '梯田': list(range(7, 21)),
    '山坡地': list(range(21, 27)),
    '水浇地': list(range(27, 35)),
    '普通大棚': list(range(35, 51)),
    '智慧大棚': list(range(51, 55))
}
# 地块类型编号
basic_plot_types = {
    '平旱地': 1,
    '梯田': 2,
```

```
    '山坡地': 3,
    '水浇地': 4,
    '普通大棚': 5,
    '智慧大棚': 6
}


# 使用映射来关联地块编号和地块类型编号
def get_plot_type(j):
    for plot_type, plot_list in plot_types.items():
        if j in plot_list:
            return basic_plot_types[plot_type]
    return None


# 作物类型分类
crop_types = {
    '粮食作物': list(range(1, 17)), # 1-16 为粮食作物
    '蔬菜作物': list(range(17, 38)), # 17-37 为蔬菜作物
    '食用菌作物': list(range(38, 42)) # 38-41 为食用菌作物
}

# 打印映射结果，检查映射是否正确
# print(plot_type_to_plots)

# 构建作物数据的字典
crop_names = dict(zip(Basic_data['作物编号'], Basic_data['作物名称'])) # 作物编号和名称的映射
plot_names = dict(zip(plot_areas_data['地块编号'], plot_areas_data['地块名称'])) #
    地块编号和名称的映射
# crop_sales_limits = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
    final_data['种植季次编号']), final_data['对应销量'])) # 作物编号和销量限制的映射
# crop_sales_limits = dict(zip(zip(final_data['作物编号'],
    get_plot_type(final_data['地块类型']),final_data['种植季次编号']), final_data['对应销量']))

# 首先对相同作物编号、地块类型、种植季次编号的对应销量求和
grouped_data = final_data.groupby(['作物编号', '种植季次编号'])['对应销量'].sum().reset_index()

# 生成字典
crop_sales_limits = dict(zip(zip(grouped_data['作物编号'], grouped_data['种植季次编号']),
    grouped_data['对应销量']))
plot_areas = dict(zip(plot_areas_data['地块编号'], plot_areas_data['地块面积/亩'])) # 地块面积

# 使用映射来关联地块编号和地块类型编号
crop_yields = dict(zip(
    zip(Basic_data['作物编号'], Basic_data['地块类型编号'], Basic_data['种植季次编号']),
    Basic_data['亩产量/斤']
))
```

```
crop_costs = dict(zip(
    zip(Basic_data['作物编号'], Basic_data['地块类型编号'], Basic_data['种植季次编号']),
    Basic_data['种植成本/(元/亩)']
))

crop_prices = dict(zip(
    zip(Basic_data['作物编号'], Basic_data['种植季次编号']),
    Basic_data['销售单价(均值)']
))

# 初始化优化模型（最大化利润）
model = pl.LpProblem("Crop_Planning_Optimization", pl.LpMaximize)

# 决策变量
X = pl.LpVariable.dicts("X", (crop_ids, plot_ids, range(num_years), season_ids), cat='Binary')
    # 是否种植的二元变量
A = pl.LpVariable.dicts("A", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0) #
    种植面积的连续变量
Z = pl.LpVariable.dicts("Z", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0)

# 为Z添加约束条件，按照地块编号 j 确定地块类型编号
for i in crop_ids:
    for j in plot_ids:
        plot_type_id = get_plot_type(j) # 获取地块类型编号
        for k in range(1, num_years): # 从k=1开始，忽略k=0
            for m in season_ids:
                # 使用地块类型编号来访问 crop_yields 中的对应值
                model += Z[i][j][k][m] <= A[i][j][k][m] * crop_yields.get((i, get_plot_type(j),
                    m), 0) # Z <= A * 单产量

# 1. 引入新的变量 Z_total 用于表示作物的总销售量
Z_total = pl.LpVariable.dicts("Z_total", (crop_ids, range(num_years), season_ids), lowBound=0)

# 2. 对每个作物的总销售量添加约束，确保其不超过预测销量
for i in crop_ids:
    for k in range(1, num_years):
        for m in season_ids:
            model += Z_total[i][k][m] == pl.lpSum([Z[i][j][k][m] for j in plot_ids]) # Z_total
                是所有地块销量的总和
            model += Z_total[i][k][m] <= crop_sales_limits.get((i, m), 0) # Z_total 不超过销量上限

# 3. 修改目标函数：使用 Z_total 计算总利润
model += pl.lpSum([
    Z_total[i][k][m] * crop_prices.get((i, m), 0) # 使用 Z_total 计算利润
    - pl.lpSum([A[i][j][k][m] * crop_costs.get((i, get_plot_type(j), m), 0) for j in plot_ids])
        # 计算总成本
```

```
    for i in crop_ids
    for k in range(1, num_years) # 从k=1到k=7
    for m in season_ids
])

# 记录已经在final_data中指定的2023年种植组合
specified_combinations = set()

# 首先处理已指定的种植计划 (k=0)
for _, row in final_data.iterrows():
    i = row['作物编号']
    j = row['地块编号']
    m = row['种植季次编号']
    # 设置X为1，表示种植
    model += X[i][j][0][m] == 1
    # 设置种植面积
    model += A[i][j][0][m] == row['种植面积/亩']
    # 记录该组合
    specified_combinations.add((i, j, 0, m))

# 对于未指定的组合，将X和A置为0 (k=0)
for i in crop_ids:
    for j in plot_ids:
        for m in season_ids:
            if (i, j, 0, m) not in specified_combinations:
                model += X[i][j][0][m] == 0
                model += A[i][j][0][m] == 0

# 添加约束：如果 X 为 0，则 A 必须为 0
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years):
            for m in season_ids:
                model += A[i][j][k][m] <= plot_areas[j] * X[i][j][k][m]

# 约束条件
# 1. 每块地在3年内应至少种植豆类作物
for j in plot_ids:
    for t in range(1, num_years - 1): # 从k=1开始
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for k in range(t - 1, t + 2) for m in
            season_ids if
                        final_data.loc[final_data['作物编号'] == i, '是否为豆类'].values[0] == 1)
                            >= plot_areas[j]

# 2. 限制每个季节种植某种作物的地块数量 (k=1到k=7)
N_max = 5
for i in crop_ids:
```

```
    for k in range(1, num_years):
        for m in season_ids:
            model += pl.lpSum(X[i][j][k][m] for j in plot_ids) <= N_max

# 3. 每块地的最小种植面积 (k=1到k=7)
A_min_percent = 0.4 # 假设每个地块至少种植50%的面积
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years):
            for m in season_ids:
                model += A[i][j][k][m] >= A_min_percent * plot_areas[j] * X[i][j][k][m]

# 4. 总种植面积不能超过地块的可用面积 (k=1到k=7)
min_utilization = 0.8 # 每个地块至少使用80%的面积
for j in plot_ids:
    for k in range(1, num_years):
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for m in season_ids) >=
            min_utilization * plot_areas[j]

# 5. 各种地块类型的种植条件限制 (k=1到k=7)
# 平旱地、梯田和山坡地只能种植粮食作物，且相邻年份不能种植相同的作物，且只能种单季 m=0
for plot_category in ['平旱地', '梯田', '山坡地']:
    for j in plot_types[plot_category]: # 遍历平旱地、梯田和山坡地
        # 限制只能种单季 m=0
        for k in range(1, num_years): # 遍历每一年
            for m in [1, 2]: # 限制双季种植 (m=1 和 m=2)
                # 不能在双季种植任何作物
                for i in crop_ids:
                    model += X[i][j][k][m] == 0

            # 只能种粮食作物
            for i in crop_types['蔬菜作物'] + crop_types['食用菌作物']: #
                禁止蔬菜和食用菌作物在这些地块种植
                model += X[i][j][k][0] == 0 # 禁止在单季种植非粮食作物

        # 对粮食作物的约束
        for i in crop_types['粮食作物']: # 允许种植粮食作物
            for k in range(1, num_years): # 遍历每一年
                # 限制不能种水稻 (编号16)
                if i == 16: # 水稻的编号是16
                    model += X[i][j][k][0] == 0 # 禁止水稻在这些地块种植
            for k in range(num_years - 1):
                # 限制相邻年份同一地块不能连续种植同一种粮食作物
                model += X[i][j][k][0] + X[i][j][k + 1][0] <= 1

# 水浇地可以一年种植一季水稻，且相邻年份水稻不能重茬
# 限制双季 (m=1 和 m=2) 不允许种粮食作物、食用菌作物
```

```
# 水浇地可以第一季种植蔬菜（大白菜、白萝卜和红萝卜除外），第二季只能种大白菜、白萝卜和红萝卜其中一种
for j in plot_types['水浇地']: # 遍历所有水浇地
    for k in range(1, num_years):
        # 限制只能在单季种植水稻（编号16），且相邻年份水稻不能连续种植
        model += X[16][j][k][1] == 0 # 不能按两季种
        model += X[16][j][k][2] == 0

        # 其他粮食作物（除水稻外）不能在水浇地种植
        for i in crop_types['粮食作物']:
            if i != 16: # 除水稻外的粮食作物
                for m in [0, 1, 2]: # 限制全年不种植
                    model += X[i][j][k][m] == 0

        # 第一季可以种蔬菜，但不允许种大白菜（35）、白萝卜（36）、红萝卜（37）
        for i in [35, 36, 37]: # 限制特定蔬菜
            model += X[i][j][k][1] == 0 # 第一季不允许种植大白菜（35）、白萝卜（36）、红萝卜（37）
            model += X[i][j][k][0] == 0
        for i in crop_types['蔬菜作物']: # 第二季只能种植大白菜（35）、白萝卜（36）、红萝卜（37）
            if i != [35, 36, 37]: # 仅允许这三种蔬菜在第二季种植
                model += X[i][j][k][2] == 0 # 第二季不能种植其他蔬菜
                model += X[i][j][k][0] == 0
        model += X[35][j][k][2] + X[36][j][k][2] + X[37][j][k][2] <= 1

        # 食用菌作物不能在水浇地种植
        for i in crop_types['食用菌作物']:
            for m in [0, 1, 2]: # 全年不允许食用菌作物
                model += X[i][j][k][m] == 0

    # 相邻年份不能连续种植水稻
    for k in range(num_years - 1):
        model += X[16][j][k][0] + X[16][j][k + 1][0] <= 1

#
    普通大棚每年适宜种植一季蔬菜(35,36,37除外)和一季食用菌且蔬菜只能在第一季种植、食用菌只能在第二季种植
for j in plot_types['普通大棚']: # 遍历普通大棚
    for k in range(1, num_years):
        for i in crop_types['蔬菜作物']: # 蔬菜作物
            if i in [35, 36, 37]:
                model += X[i][j][k][1] == 0
            # 限制蔬菜只能在第一季种植
            model += X[i][j][k][2] == 0
            model += X[i][j][k][0] == 0
        for i in crop_types['粮食作物']: # 粮食作物
            # 限制粮食不能种植
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
```

```
            # 限制食用菌只能在第二季种植
            model += X[i][j][k][1] == 0
            model += X[i][j][k][0] == 0

# 智慧大棚每年种植两季蔬菜(35,36,37除外)，且相邻季节不能种植相同的作物
for j in plot_types['智慧大棚']: # 遍历智慧大棚
    for i in crop_types['蔬菜作物']: # 蔬菜作物
        for k in range(num_years - 1):
            # 相邻年份不能重茬
            model += X[i][j][k][2] + X[i][j][k + 1][1] <= 1
        for k in range(num_years):
            # 同年不能种植相同的蔬菜
            model += X[i][j][k][1] + X[i][j][k][2] <= 1

    for k in range(1, num_years):
        for i in crop_types['蔬菜作物']: # 蔬菜作物不能种单季
            model += X[i][j][k][0] == 0
            if i in [35, 36, 37]: # 蔬菜中的大白菜（35）、白萝卜（36）、红萝卜（37）不能种
                model += X[i][j][k][1] == 0
                model += X[i][j][k][2] == 0
        for i in crop_types['粮食作物']: # 粮食作物
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0


def simulated_annealing(model, initial_solution, max_iter=500, initial_temp=10,
    cooling_rate=0.99, tolerance=1e-5):
    """
    模拟退火算法,用于优化PuLP模型的解。

    参数:
    - model: PuLP 模型
    - initial_solution: 初始解
    - max_iter: 最大迭代次数
    - initial_temp: 初始温度
    - cooling_rate: 温度递减率
    - tolerance: 目标函数差值的容忍度，改进小于该值时提前停止

    返回:
    - 最优解和最优目标函数值
    """
    current_solution = initial_solution
    current_value = pl.value(model.objective)
```

```
    best_solution = current_solution
    best_value = current_value

    temperature = initial_temp

    for iteration in range(max_iter):
        new_solution = generate_neighbor_solution(current_solution)
        apply_solution_to_model(model, new_solution)

        # 重新求解模型
        model.solve()
        new_value = pl.value(model.objective)

        # 计算目标函数差值
        delta = new_value - current_value

        # 根据退火准则决定是否接受新解
        if delta > 0 or random.uniform(0, 1) < math.exp(delta / temperature):
            current_solution = new_solution
            current_value = new_value

            # 更新最优解
            if current_value > best_value:
                best_solution = current_solution
                best_value = current_value

        # 打印当前迭代状态
        print(
            f"迭代 {iteration}: 当前解的目标函数值 = {current_value}, 最优解的目标函数值 =
                {best_value}, 温度 = {temperature}")

        # 如果改进的幅度很小，则提前停止
        if abs(delta) < tolerance:
            print(f"优化幅度低于 {tolerance}, 提前停止.")
            break

        # 降低温度
        temperature *= cooling_rate

        # 如果温度过低，则停止
        if temperature < 1e-3:
            print("温度过低，停止优化.")
            break

    return best_solution, best_value
```

```python
def generate_neighbor_solution(current_solution):
    """
    生成当前解的邻域解。

    参数:
    - current_solution: 当前解

    返回:
    - 邻域解
    """
    new_solution = current_solution.copy()

    # 随机选择一个作物和地块进行微调
    i = random.choice(list(crop_ids))
    j = random.choice(list(plot_ids))
    k = random.randint(1, num_years - 1)
    m = random.choice(list(season_ids))

    # 随机调整种植面积（例如上下波动 5%）
    adjustment = random.uniform(-0.05, 0.05) * plot_areas[j]
    new_solution[(i, j, k, m)] = max(0, current_solution[(i, j, k, m)] + adjustment)

    return new_solution


def apply_solution_to_model(model, solution):
    """
    将解应用到模型的决策变量中。

    参数:
    - model: PuLP 模型
    - solution: 新的解
    """
    for i in crop_ids:
        for j in plot_ids:
            for k in range(1, num_years):
                for m in season_ids:
                    # 获取解中的值，如果不存在，则默认为 0
                    value = solution.get((i, j, k, m), 0)
                    if value is not None: # 确保值不为 None
                        A[i][j][k][m].setInitialValue(value)


# 求解模型，获取初始解
solver = pl.PULP_CBC_CMD(msg=True, threads=8, timeLimit=10)
model.solve(solver)
```

```
# 初始解，确保所有变量都有值，默认值为 0
initial_solution = {(i, j, k, m): pl.value(A[i][j][k][m]) if pl.value(A[i][j][k][m]) is not
    None else 0
                for i in crop_ids for j in plot_ids for k in range(1, num_years) for m in
                    season_ids}

# 应用模拟退火算法进行优化
best_solution, best_value = simulated_annealing(model, initial_solution)

# 打印退火算法优化后的最优解
print(f"退火算法优化后的最优利润: {best_value}")

# 输出最终的最优解并保存
if best_solution:
    start_year = 2023 # 从2024年开始
    for k in range(1, num_years):
        year_solution = []

        for i in crop_ids:
            for j in plot_ids:
                for m in season_ids:
                    # 收集每一年的决策变量值
                    x_value = pl.value(X[i][j][k][m])
                    a_value = best_solution.get((i, j, k, m), 0)

                    # 如果种植面积大于 0，提取对应的单价、亩产量和成本
                    if a_value > 0:
                        unit_price = crop_prices.get((i, m), 0) # 单价
                        yield_per_mu = crop_yields.get((i, get_plot_type(j), m), 0) # 亩产量
                        cost_per_mu = crop_costs.get((i, get_plot_type(j), m), 0) # 种植成本
                        # 将信息添加到year_solution
                        year_solution.append(
                            [crop_names[i], plot_names[j], m, x_value, a_value, unit_price,
                                yield_per_mu, cost_per_mu])

        # 将结果转换为DataFrame
        year_solution_df = pd.DataFrame(year_solution,
                                        columns=['作物名称', '地块名称', '种植季次', '种每种',
                                            '种植面积/亩',
                                                '单价（元/斤）', '亩产量（斤/亩）',
                                                    '种植成本（元/亩）'])

        # 保存当前年份的解决方案为CSV文件
        year_solution_df.to_csv(f"问题一/结果/优化_crop_plan_{start_year + k}.csv", index=False,
            encoding='utf-8-sig')
        print(f"{start_year + k}年种植计划已保存为CSV文件")
```

附录 D　问题二的模型建立与求解

4.1　MIP+ 鲁棒优化 + 模拟退火模型

```
import pandas as pd
import pulp as pl
import random
import numpy as np

# 加载数据
final_data = pd.read_excel('./data/final.xlsx')

# 提取相关数据用于模型构建
crop_ids = final_data['作物编号'].unique()
plot_ids = final_data['地块编号'].unique()
season_ids = final_data['种植季次编号'].unique()
num_years = 7+1 # 我们计划的年限是7年（从2024到2030）

# 作物类型分类
crop_types = {
    '粮食作物': list(range(1, 17)), # 1-16 为粮食作物
    '蔬菜作物': list(range(17, 38)), # 17-37 为蔬菜作物
    '食用菌作物': list(range(38, 42)) # 38-41 为食用菌作物
}

# 地块类型分类
plot_types = {
    '平旱地': list(range(1, 7)),
    '梯田': list(range(7, 21)),
    '山坡地': list(range(21, 27)),
    '水浇地': list(range(27, 35)),
    '普通大棚': list(range(35, 51)),
    '智慧大棚': list(range(51, 55))
}

# 构建作物数据的字典
crop_names = dict(zip(final_data['作物编号'], final_data['作物名称']))
plot_names = dict(zip(final_data['地块编号'], final_data['地块名称'])) # 物块编号和名称的映射

crop_sales_limits = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
    final_data['种植季次编号']), final_data['对应销量']))
crop_yields = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
    final_data['种植季次编号']), final_data['亩产量/斤']))
crop_costs = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
    final_data['种植季次编号']), final_data['种植成本/(元/亩)']))
crop_prices = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
```

```
    final_data['种植季次编号']), final_data['销售单价（均值）']))
plot_areas = dict(zip(final_data['地块编号'], final_data['地块面积/亩'])) # 地块面积

# 初始化优化模型（最大化利润）
model = pl.LpProblem("Crop_Planning_Optimization", pl.LpMaximize)

# 决策变量
X = pl.LpVariable.dicts("X", (crop_ids, plot_ids, range(num_years), season_ids), cat='Binary')
    # 是否种植的二元变量
A = pl.LpVariable.dicts("A", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0) #
    种植面积的连续变量
Z = pl.LpVariable.dicts("Z", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0)

# 参数：小麦、玉米的不确定性增长率范围
wheat_growth_range = [0.05, 0.10] # 小麦未来几年的增长率范围 5%-10%
corn_growth_range = [0.05, 0.10] # 玉米未来几年的增长率范围 5%-10%

# 参数：其他作物的销售量波动范围和亩产量的不确定性
sales_variation_range = [-0.05, 0.05] # 其他作物每年销售量波动范围 ±5%
yield_variation_range = [-0.10, 0.10] # 其他作物亩产量的不确定性范围 ±10%

# 参数：蔬菜类和食用菌类作物的价格增长或下降范围
vegetable_price_growth = 0.05 # 蔬菜类作物价格每年增长 5%
fungi_price_decrease = [0.01, 0.05] # 食用菌类作物价格每年下降 1%-5%

# 对所有作物的亩产量进行不确定性调整
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years): # k 表示年份，从2024到2030年
            for m in season_ids:
                # 考虑亩产量的 ±10% 波动
                epsilon = random.uniform(yield_variation_range[0], yield_variation_range[1]) #
                    随机生成亩产量波动
                yield_2023 = crop_yields.get((i, j, m), 0) # 获取2023年亩产量
                yield_adjusted = yield_2023 * (1 + epsilon) # 调整后的亩产量
                model += Z[i][j][k][m] <= A[i][j][k][m] * yield_adjusted #
                    约束：实际销售量不能超过产量

# 添加不确定性因素的销量限制
for i in crop_ids:
    for m in season_ids:
        for t in range(1, num_years): # k 表示年份，从2024到2030年
            for plot_category in plot_types:
                # 销量限制根据作物类型处理
                if i == "小麦": # 小麦的销售量每年增长 5%-10%
                    wheat_growth = random.uniform(wheat_growth_range[0], wheat_growth_range[1])
                    sales_limit_adjusted = pl.lpSum(crop_sales_limits.get((i, j, m), 0) * (1 +
```

```
                    wheat_growth) ** (t - 1) for j in plot_types[plot_category])
            elif i == "玉米": # 玉米的销售量每年增长 5%-10%
                corn_growth = random.uniform(corn_growth_range[0], corn_growth_range[1])
                sales_limit_adjusted = pl.lpSum(crop_sales_limits.get((i, j, m), 0) * (1 +
                    corn_growth) ** (t - 1) for j in plot_types[plot_category])
            else: # 其他农作物的销售量波动范围 ±5%
                delta = random.uniform(sales_variation_range[0], sales_variation_range[1])
                sales_limit_adjusted = pl.lpSum(crop_sales_limits.get((i, j, m), 0) * (1 +
                    delta) for j in plot_types[plot_category])
            # 添加销量限制
            model += pl.lpSum(Z[i][j][t][m] for j in plot_types[plot_category]) <=
                sales_limit_adjusted

# 创建字典来保存求解时的价格、成本和亩产量
price_cost_yield_records = {}

# 添加不确定性因素的目标函数
objective_terms = [] # 初始化目标函数项的列表

# 遍历作物、地块、年份和季次，计算每个部分的目标函数值，并将它们收集到 objective_terms 中
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years): # k 从 1 开始表示 2024 到 2030 年
            for m in season_ids:
                # 如果是蔬菜类作物，价格每年上涨 5%
                if i in crop_types['蔬菜作物']:
                    price_adjusted = crop_prices.get((i, j, m), 0) * (1 + vegetable_price_growth)
                        ** k

                # 如果是食用菌类作物，进一步区分羊肚菌和其他食用菌
                elif i in crop_types['食用菌作物']:
                    if crop_names[i] == "羊肚菌": # 羊肚菌的价格每年固定下降 5%
                        price_adjusted = crop_prices.get((i, j, m), 0) * (1 - 0.05) ** k
                    else: # 其他食用菌价格每年随机下降 1%-5%
                        fungi_decrease = random.uniform(fungi_price_decrease[0],
                            fungi_price_decrease[1])
                        price_adjusted = crop_prices.get((i, j, m), 0) * (1 - fungi_decrease) ** k

                # 粮食类作物价格不变
                else:
                    price_adjusted = crop_prices.get((i, j, m), 0) # 粮食价格保持不变

                # 调整亩产量
                epsilon = random.uniform(yield_variation_range[0], yield_variation_range[1]) #
                    随机生成亩产量波动
                yield_adjusted = crop_yields.get((i, j, m), 0) * (1 + epsilon) # 调整后的亩产量
```

```
                # 添加目标函数项（收入 - 成本）
                revenue = Z[i][j][k][m] * price_adjusted # 收入 = 实际销售量 * 调整后的销售价格
                cost = A[i][j][k][m] * crop_costs.get((i, j, m), 0) * (1.05 ** (k - 1)) # 成本 =
                    种植面积 * 每年增长5%的种植成本
                objective_terms.append(revenue - cost) # 收集目标函数项：收入 - 成本

                # 记录价格、成本和亩产量数据
                price_cost_yield_records[(i, j, k, m)] = {
                    '价格': price_adjusted,
                    '成本': crop_costs.get((i, j, m), 0) * (1.05 ** (k - 1)),
                    '亩产量': yield_adjusted # 记录调整后的亩产量
                }

# 设置唯一的目标函数为所有目标项的总和
model += pl.lpSum(objective_terms) # 最终的目标函数是所有目标项的总和


# 记录已经在final_data中指定的2023年种植组合
specified_combinations = set()

# 首先处理已指定的种植计划（k=0）
for _, row in final_data.iterrows():
    i = row['作物编号']
    j = row['地块编号']
    m = row['种植季次编号']
    # 设置X为1，表示种植
    model += X[i][j][0][m] == 1
    # 设置种植面积
    model += A[i][j][0][m] == row['种植面积/亩']
    # 记录该组合
    specified_combinations.add((i, j, 0, m))

# 对于未指定的组合，将X和A置为0（k=0）
for i in crop_ids:
    for j in plot_ids:
        for m in season_ids:
            if (i, j, 0, m) not in specified_combinations:
                model += X[i][j][0][m] == 0
                model += A[i][j][0][m] == 0

# 约束条件

# 1. 每块地在3年内应至少种植豆类作物
for j in plot_ids:
    for t in range(1, num_years - 1): # 从k=1开始
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for k in range(t-1, t+2) for m in
            season_ids if final_data.loc[final_data['作物编号'] == i, '是否为豆类'].values[0] ==
```

```
            1) >= plot_areas[j]

# 2. 限制每个季节种植某种作物的地块数量 (k=1到k=7)
N_max = 5
for i in crop_ids:
    for k in range(1, num_years):
        for m in season_ids:
            model += pl.lpSum(X[i][j][k][m] for j in plot_ids) <= N_max

# 3. 每块地的最小种植面积 (k=1到k=7)
A_min_percent = 0.5 # 假设每个地块至少种植50%的面积
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years):
            for m in season_ids:
                model += A[i][j][k][m] >= A_min_percent * plot_areas[j] * X[i][j][k][m]

# 4. 总种植面积不能超过地块的可用面积 (k=1到k=7)
min_utilization = 0.8 # 每个地块至少使用80%的面积
for j in plot_ids:
    for k in range(1, num_years):
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for m in season_ids) >=
            min_utilization * plot_areas[j]

# 5. 各种地块类型的种植条件限制 (k=1到k=7)
# 平旱地、梯田和山坡地只能种植粮食作物，且相邻年份不能种植相同的作物
for plot_category in ['平旱地', '梯田', '山坡地']:
    for j in plot_types[plot_category]: # 遍历平旱地、梯田和山坡地
        # 添加对非粮食作物的限制：蔬菜和食用菌不能种植在这些地块
        for i in crop_types['蔬菜作物']: # 禁止蔬菜作物
            for k in range(num_years): # 遍历每一年
                model += X[i][j][k][1] == 0 # 禁止蔬菜在这些地块的第1季种植

        for i in crop_types['食用菌作物']: # 禁止食用菌作物
            for k in range(num_years): # 遍历每一年
                model += X[i][j][k][1] == 0 # 禁止食用菌在这些地块的第1季种植

        # 对粮食作物的约束
        for i in crop_types['粮食作物']: # 允许种植粮食作物
            for k in range(num_years): # 遍历每一年
                # 限制不能种水稻 (编号16)
                if i == 16: # 水稻的编号是16
                    model += X[i][j][k][1] == 0 # 禁止在这些地块种植水稻
            for k in range(num_years - 1):
                # 限制相邻年份同一地块不能连续种植同一种粮食作物
                model += X[i][j][k][1] + X[i][j][k+1][1] <= 1
```

```
# 水浇地每年只能种植一季水稻或两季蔬菜，且相邻年份水稻不能重茬
for j in plot_types['水浇地']: # 遍历水浇地
    for k in range(num_years - 1):
        # 水稻编号为16，限制水稻相邻年份不能重茬
        model += X[16][j][k][1] + X[16][j][k+1][1] <= 1
    for i in crop_types['蔬菜作物']: # 蔬菜作物
        for k in range(num_years):
            # 连续不能种植两季蔬菜
            model += X[i][j][k][1] + X[i][j][k][2] <= 1
        for k in range(num_years-1):
            model += X[i][j][k][2] + X[i][j][k+1][1] <= 1
        # 限制大白菜（35）、白萝卜（36）、红萝卜（37）不能在第一季种植
        model += X[35][j][k][1] == 0 # 大白菜
        model += X[36][j][k][1] == 0 # 白萝卜
        model += X[37][j][k][1] == 0 # 红萝卜
    for i in crop_types['粮食作物']: # 粮食作物
        for k in range(num_years):
            if i != 16:
                # 限制粮食（除了水稻）不能在大棚种植
                model += X[i][j][k][1] == 0
                model += X[i][j][k][2] == 0
    for i in crop_types['食用菌作物']: # 食用菌作物
        for k in range(num_years):
            # 限制粮食不能在大棚种植
            model += X[i][j][k][1] == 0
            model += X[i][j][k][2] == 0

# 普通大棚每年适宜种植一季蔬菜和一季食用菌且蔬菜只能在第一季种植、食用菌只能在第二季种植
for j in plot_types['普通大棚']: # 遍历普通大棚
    for k in range(num_years):
        for i in crop_types['蔬菜作物']: # 蔬菜作物
            # 限制蔬菜只能在第一季种植
            model += X[i][j][k][2] == 0
        for i in crop_types['粮食作物']: # 粮食作物
            # 限制粮食不能种植
            model += X[i][j][k][2] == 0
            model += X[i][j][k][1] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            # 限制食用菌只能在第二季种植
            model += X[i][j][k][1] == 0

# 智慧大棚每年种植两季蔬菜，且相邻季节不能种植相同的作物
for j in plot_types['智慧大棚']: # 遍历智慧大棚
    for i in crop_types['蔬菜作物']: # 蔬菜作物
        for k in range(num_years - 1):
            # 同年不能种植相同的蔬菜
            model += X[i][j][k][1] + X[i][j][k][2] <= 1
```

```
            # 相邻年份不能重茬
            model += X[i][j][k][2] + X[i][j][k+1][1] <= 1
    for k in range(num_years):
        for i in crop_types['粮食作物']: # 粮食作物
            model += X[i][j][k][2] == 0
            model += X[i][j][k][1] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            model += X[i][j][k][2] == 0
            model += X[i][j][k][1] == 0

# 定作物的种植约束（大白菜、白萝卜、红萝卜只能在水浇地的第二季种植）
for plot_category in plot_types: # 遍历所有地块类型
    if plot_category != '水浇地': # 如果不是水浇地
        for j in plot_types[plot_category]:
            for k in range(num_years):
                # 大白菜（35）、白萝卜（36）、红萝卜（37）不能在非水浇地种植
                model += X[35][j][k][1] == 0 # 大白菜
                model += X[36][j][k][1] == 0 # 白萝卜
                model += X[37][j][k][1] == 0 # 红萝卜
                # 水稻（16）不能在非水浇地种植
                model += X[16][j][k][1] == 0

# 求解模型
model.solve()

# 初始解状态
initial_profit = pl.value(model.objective)
print(f"初始最优利润: {initial_profit}")

# 退火算法优化部分
def simulated_annealing(solution, temperature, cooling_rate=0.99):
    best_solution = solution.copy()
    current_solution = solution.copy()
    current_profit = initial_profit
    best_profit = current_profit

    while temperature > 1:
        # 随机改变解
        new_solution = perturb_solution(current_solution)
        new_profit = evaluate_solution(new_solution)

        # 如果新解更优或以一定概率接受较差解
        if new_profit > current_profit or random.uniform(0, 1) < np.exp((new_profit -
            current_profit) / temperature):
            current_solution = new_solution
            current_profit = new_profit
            if new_profit > best_profit:
```

```
                best_solution = new_solution
                best_profit = new_profit

        # 降低温度
        temperature *= cooling_rate

    return best_solution, best_profit

def evaluate_solution(solution):
    # 计算解的利润
    profit = 0
    for i in crop_ids:
        for j in plot_ids:
            for k in range(1, num_years):
                for m in season_ids:
                    area = solution.get((i, j, k, m), 0)
                    yield_per_acre = crop_yields.get((i, j, m), 0)
                    cost_per_acre = crop_costs.get((i, j, m), 0) * (1.05 ** (k - 1)) #
                        每年成本增长5%

                    # 价格调整
                    if i in crop_types['蔬菜作物']:
                        # 蔬菜价格每年上涨5%
                        price_adjusted = crop_prices.get((i, j, m), 0) * (1 +
                            vegetable_price_growth) ** k
                    elif i in crop_types['食用菌作物']:
                        if crop_names[i] == "羊肚菌":
                            # 羊肚菌价格每年下降5%
                            price_adjusted = crop_prices.get((i, j, m), 0) * (1 - 0.05) ** k
                        else:
                            # 其他食用菌价格随机下降1%-5%
                            fungi_decrease = random.uniform(fungi_price_decrease[0],
                                fungi_price_decrease[1])
                            price_adjusted = crop_prices.get((i, j, m), 0) * (1 - fungi_decrease)
                                ** k
                    else:
                        # 其他作物价格保持不变
                        price_adjusted = crop_prices.get((i, j, m), 0)

                    # 计算利润 = 实际面积的收入 - 成本
                    revenue = area * yield_per_acre * price_adjusted
                    cost = area * cost_per_acre
                    profit += revenue - cost

    print(profit) # 输出利润以便调试
    return profit
```

```
def perturb_solution(solution):
    new_solution = solution.copy()
    i = random.choice(crop_ids)
    j = random.choice(plot_ids)
    k = random.randint(1, num_years - 1)
    m = random.choice(season_ids)

    # 确保作物只能在允许的季节种植
    if i in crop_types['蔬菜作物'] and m == 2: # 蔬菜只能在第一季种植
        return solution
    if i in crop_types['食用菌作物'] and m == 1: # 食用菌只能在第二季种植
        return solution

    # 确保种植面积不会超过地块可用面积
    max_area = plot_areas[j]
    used_area = sum(new_solution.get((i, j, k, m), 0) for i in crop_ids)
    available_area = max_area - used_area

    # 控制调整幅度
    adjustment_factor = 0.1 # 调整幅度为10%

    if available_area > 0:
        new_solution[(i, j, k, m)] = new_solution.get((i, j, k, m), 0) * (1 +
            random.uniform(-adjustment_factor, adjustment_factor))

    return new_solution



# 将初始解提取为退火算法的输入
initial_solution = {(i, j, k, m): pl.value(A[i][j][k][m]) for i in crop_ids for j in plot_ids
    for k in range(1, num_years) for m in season_ids}

# 运行退火算法
final_solution, final_profit = simulated_annealing(initial_solution, temperature=1000)

print(f"退火优化后的利润: {final_profit}")

# 保存退火优化后的结果，并添加价格、成本、亩产量信息
start_year = 2023
for k in range(1, num_years): # 遍历每年
    year_solution = []
    for i in crop_ids:
        for j in plot_ids:
            for m in season_ids:
                a_value = final_solution.get((i, j, k, m), 0)
```

```
            if a_value > 0:
                # 从求解时的记录中获取价格、成本和亩产量
                record = price_cost_yield_records[(i, j, k, m)]
                price_adjusted = record['价格']
                cost_adjusted = record['成本']
                yield_adjusted = record['亩产量']

                # 将结果存储为列表
                year_solution.append([
                    crop_names[i], # 作物名称
                    plot_names[j], # 地块名称
                    m, # 种植季次
                    a_value, # 种植面积
                    price_adjusted, # 调整后的销售单价
                    cost_adjusted, # 调整后的种植成本
                    yield_adjusted # 调整后的亩产量
                ])

    # 转换为DataFrame
    year_solution_df = pd.DataFrame(
        year_solution,
        columns=['作物名称', '地块名称', '种植季次', '种植面积/亩', '调整后的销售单价',
            '调整后的种植成本', '调整后的亩产量']
    )

    # 保存结果为CSV文件
    year_solution_df.to_csv(f"问题二/结果/Q2鲁棒MIP退火结果_{start_year + k}_带单价成本产量.csv",
        index=False, encoding='utf-8-sig')
    print(f"{start_year + k}年优化后的种植计划（包含单价、成本和亩产量）已保存为CSV文件")
```

附录 E　问题三的模型建立与求解

5.1　需求曲线

```
import pandas as pd
import matplotlib.pyplot as plt

# 设置字体为微软雅黑
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei'] # 设置字体为微软雅黑
plt.rcParams['axes.unicode_minus'] = False # 解决负号显示问题

# 加载合并的总销量表，假设数据中包含年份、季次、作物名称、作物编号、总销量列
file_path = './data/模拟销量7年数据合并.xlsx'
sales_data = pd.read_excel(file_path)
```

```python
# 定义作物类型分类，不再使用作物编号，而是使用作物名称
crop_types = {
    '粮食作物': ['小麦', '水稻', '玉米', '高粱', '黑豆', '黍子', '荞麦', '绿豆', '红豆', '大麦',
        '莜麦', '红薯', '南瓜', '爬豆', '黄豆'],
    '蔬菜作物': ['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'],
    '食用菌作物': ['香菇', '榆黄菇', '白灵菇', '羊肚菌']
}

# 定义绘制销量曲线的函数
def plot_sales_trend(data, crop_type, crop_names):
    # 过滤出对应作物类型的数据，根据作物名称进行过滤
    filtered_data = data[data['作物名称'].isin(crop_names)]

    # 创建唯一标签：种植季次 + 作物名称
    filtered_data['标签'] = filtered_data['种植季次'].astype(str) + '季的' +
        filtered_data['作物名称']

    # 按作物名称和季次分组，绘制每个标签的销量曲线
    for label in filtered_data['标签'].unique():
        label_data = filtered_data[filtered_data['标签'] == label]

        # 确保年份为数值型并按年份排序
        label_data['年份'] = pd.to_numeric(label_data['年份'], errors='coerce')
        label_data = label_data.sort_values(by='年份')

        # 绘制年份 vs. 总销量
        plt.plot(label_data['年份'], label_data['总销量'], marker='o', label=label)

    # 绘制图表
    plt.title(f'{crop_type} 作物类型销量曲线', fontsize=14)
    plt.xlabel('年份', fontsize=12)
    plt.ylabel('总销量', fontsize=12)

    # 强制显示所有年份标签，并旋转45度
    plt.xticks(label_data['年份'].unique(), rotation=45)

    # 设置图例为2列
    plt.legend(loc='upper right', ncol=2)

    plt.grid(True)
    plt.tight_layout()
    plt.show()

# 对每个作物类型绘制销量曲线
```

```
for crop_type, crop_names in crop_types.items():
    print(f'绘制 {crop_type} 的作物销量曲线: ')
    plot_sales_trend(sales_data, crop_type, crop_names)
```

5.2 回归分析

```
import pandas as pd
import numpy as np
import glob
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from matplotlib import rcParams

# 设置字体为微软雅黑，用于显示中文
rcParams['font.sans-serif'] = ['Microsoft YaHei']
rcParams['axes.unicode_minus'] = False

# 定义作物类型
crop_types = {
    '粮食作物': ['小麦', '水稻', '玉米', '高粱', '黑豆', '黍子', '荞麦', '绿豆', '红豆', '大麦',
        '莜麦', '红薯', '南瓜', '爬豆', '黄豆'],
    '蔬菜作物': ['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'],
    '食用菌作物': ['香菇', '榆黄菇', '白灵菇', '羊肚菌']
}

# 创建一个函数来加载多个文件并合并数据
def load_and_merge_data(file_paths):
    data_list = []
    for file in file_paths:
        # 从文件名中提取年份，假设文件名中包含年份
        year = int(file.split('_')[1])
        df = pd.read_csv(file)
        df['年份'] = year # 添加年份列
        data_list.append(df)
    # 合并所有数据
    combined_data = pd.concat(data_list, ignore_index=True)
    return combined_data

# 假设文件保存在特定目录下，匹配所有相关文件
file_paths = glob.glob('./问题二/结果/Q2鲁棒MIP退火结果_*_带单价成本产量.csv')
```

```
# 加载并合并数据
combined_data = load_and_merge_data(file_paths)

# 计算总销量：种植面积 * 亩产量
combined_data['总销量'] = combined_data['种植面积/亩'] * combined_data['调整后的亩产量']

# 计算每单位产量的种植成本
combined_data['种植成本/亩产量'] = combined_data['调整后的种植成本'] /
    combined_data['调整后的亩产量']

# 过滤掉无效或缺失数据
combined_data = combined_data.dropna(subset=['调整后的销售单价', '种植成本/亩产量', '总销量'])

# 对每个作物类型进行回归分析并绘制回归曲线
for crop_type, crop_list in crop_types.items():
    # 过滤指定类型的作物数据
    crop_data = combined_data[combined_data['作物名称'].isin(crop_list)]

    # 检查是否有足够的数据进行回归分析
    if len(crop_data) < 10:
        print(f"{crop_type} 数据量不足，无法进行回归分析。")
        continue

    # 准备自变量和目标变量
    # 自变量：种植成本/亩产量
    X = crop_data[['种植成本/亩产量']]
    # 目标变量：总销量
    y = crop_data['总销量']

    # 将数据拆分为训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # 创建 Ridge 回归模型
    ridge_reg_model = Ridge(alpha=1.0)

    # 拟合模型
    ridge_reg_model.fit(X_train, y_train)

    # 获取回归系数和截距
    ridge_coefficients = ridge_reg_model.coef_
    ridge_intercept = ridge_reg_model.intercept_

    # 输出回归系数和截距
    print(f"\n{crop_type} 的 Ridge 回归结果：")
    print("回归系数:", ridge_coefficients)
    print("截距:", ridge_intercept)
```

```
    # 评估模型性能
    ridge_score_train = ridge_reg_model.score(X_train, y_train)
    ridge_score_test = ridge_reg_model.score(X_test, y_test)

    print("训练集得分:", ridge_score_train)
    print("测试集得分:", ridge_score_test)

    # 绘制回归曲线
    plt.figure(figsize=(10, 6))
    plt.scatter(X_test, y_test, color='blue', label='实际总销量') # 实际值散点图
    plt.plot(X_test, ridge_reg_model.predict(X_test), color='red', label='预测总销量',
        linewidth=2) # 回归曲线
    plt.title(f'{crop_type} 的种植成本/亩产量与总销量的回归曲线', fontsize=16)
    plt.xlabel('种植成本/亩产量', fontsize=12)
    plt.ylabel('总销量', fontsize=12)
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

5.3 问题三模型求解

```
import pandas as pd
import pulp as pl
import random
import numpy as np

# 加载数据
final_data = pd.read_excel('./data/final.xlsx')

# 提取相关数据用于模型构建
crop_ids = final_data['作物编号'].unique()
plot_ids = final_data['地块编号'].unique()
season_ids = final_data['种植季次编号'].unique()
num_years = 7+1 # 我们计划的年限是7年（从2024到2030）

# 作物类型分类
crop_types = {
    '粮食作物': list(range(1, 17)), # 1-16 为粮食作物
    '蔬菜作物': list(range(17, 38)), # 17-37 为蔬菜作物
    '食用菌作物': list(range(38, 42)) # 38-41 为食用菌作物
}

# 地块类型分类
plot_types = {
```

```
    '平旱地': list(range(1, 7)),
    '梯田': list(range(7, 21)),
    '山坡地': list(range(21, 27)),
    '水浇地': list(range(27, 35)),
    '普通大棚': list(range(35, 51)),
    '智慧大棚': list(range(51, 55))
}

# 构建作物数据的字典
crop_names = dict(zip(final_data['作物编号'], final_data['作物名称']))
plot_names = dict(zip(final_data['地块编号'], final_data['地块名称'])) # 物块编号和名称的映射

crop_sales_limits = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
    final_data['种植季次编号']), final_data['对应销量']))
crop_yields = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
    final_data['种植季次编号']), final_data['亩产量/斤']))
crop_costs = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
    final_data['种植季次编号']), final_data['种植成本/(元/亩)']))
crop_prices = dict(zip(zip(final_data['作物编号'], final_data['地块编号'],
    final_data['种植季次编号']), final_data['销售单价（均值）']))
plot_areas = dict(zip(final_data['地块编号'], final_data['地块面积/亩'])) # 地块面积

# 初始化优化模型（最大化利润）
model = pl.LpProblem("Crop_Planning_Optimization", pl.LpMaximize)

# 决策变量
X = pl.LpVariable.dicts("X", (crop_ids, plot_ids, range(num_years), season_ids), cat='Binary')
    # 是否种植的二元变量
A = pl.LpVariable.dicts("A", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0) #
    种植面积的连续变量
Z = pl.LpVariable.dicts("Z", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0)

# 参数：小麦、玉米的不确定性增长率范围
wheat_growth_range = [0.05, 0.10] # 小麦未来几年的增长率范围 5%-10%
corn_growth_range = [0.05, 0.10] # 玉米未来几年的增长率范围 5%-10%

# 参数：其他作物的销售量波动范围和亩产量的不确定性
sales_variation_range = [-0.05, 0.05] # 其他作物每年销售量波动范围 ±5%
yield_variation_range = [-0.10, 0.10] # 其他作物亩产量的不确定性范围 ±10%

# 参数：蔬菜类和食用菌类作物的价格增长或下降范围
vegetable_price_growth = 0.05 # 蔬菜类作物价格每年增长 5%
fungi_price_decrease = [0.01, 0.05] # 食用菌类作物价格每年下降 1%-5%

# 对所有作物的亩产量进行不确定性调整
for i in crop_ids:
    for j in plot_ids:
```

```
        for k in range(1, num_years): # k 表示年份，从2024到2030年
            for m in season_ids:
                # 考虑亩产量的 ±10% 波动
                epsilon = random.uniform(yield_variation_range[0], yield_variation_range[1]) #
                    随机生成亩产量波动
                yield_2023 = crop_yields.get((i, j, m), 0) # 获取2023年亩产量
                yield_adjusted = yield_2023 * (1 + epsilon) # 调整后的亩产量
                model += Z[i][j][k][m] <= A[i][j][k][m] * yield_adjusted #
                    约束：实际销售量不能超过产量

# 添加不确定性因素的销量限制
for i in crop_ids:
    for m in season_ids:
        for t in range(1, num_years): # k 表示年份，从2024到2030年
            for plot_category in plot_types:
                # 销量限制根据作物类型处理
                if i == "小麦": # 小麦的销售量每年增长 5%-10%
                    wheat_growth = random.uniform(wheat_growth_range[0], wheat_growth_range[1])
                    sales_limit_adjusted = pl.lpSum(crop_sales_limits.get((i, j, m), 0) * (1 +
                        wheat_growth) ** (t - 1) for j in plot_types[plot_category])
                elif i == "玉米": # 玉米的销售量每年增长 5%-10%
                    corn_growth = random.uniform(corn_growth_range[0], corn_growth_range[1])
                    sales_limit_adjusted = pl.lpSum(crop_sales_limits.get((i, j, m), 0) * (1 +
                        corn_growth) ** (t - 1) for j in plot_types[plot_category])
                else: # 其他农作物的销售量波动范围 ±5%
                    delta = random.uniform(sales_variation_range[0], sales_variation_range[1])
                    sales_limit_adjusted = pl.lpSum(crop_sales_limits.get((i, j, m), 0) * (1 +
                        delta) for j in plot_types[plot_category])
                # 添加销量限制
                model += pl.lpSum(Z[i][j][t][m] for j in plot_types[plot_category]) <=
                    sales_limit_adjusted

# 创建字典来保存求解时的价格、成本和亩产量
price_cost_yield_records = {}

# 添加不确定性因素的目标函数
objective_terms = [] # 初始化目标函数项的列表

# 遍历作物、地块、年份和季次，计算每个部分的目标函数值，并将它们收集到 objective_terms 中
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years): # k 从 1 开始表示 2024 到 2030 年
            for m in season_ids:
                # 如果是蔬菜类作物，价格每年上涨 5%
                if i in crop_types['蔬菜作物']:
                    price_adjusted = crop_prices.get((i, j, m), 0) * (1 + vegetable_price_growth)
                        ** k
```

```
            # 如果是食用菌类作物，进一步区分羊肚菌和其他食用菌
            elif i in crop_types['食用菌作物']:
                if crop_names[i] == "羊肚菌": # 羊肚菌的价格每年固定下降 5%
                    price_adjusted = crop_prices.get((i, j, m), 0) * (1 - 0.05) ** k
                else: # 其他食用菌价格每年随机下降 1%-5%
                    fungi_decrease = random.uniform(fungi_price_decrease[0],
                        fungi_price_decrease[1])
                    price_adjusted = crop_prices.get((i, j, m), 0) * (1 - fungi_decrease) ** k

            # 粮食类作物价格不变
            else:
                price_adjusted = crop_prices.get((i, j, m), 0) # 粮食价格保持不变

            # 调整亩产量
            epsilon = random.uniform(yield_variation_range[0], yield_variation_range[1]) #
                随机生成亩产量波动
            yield_adjusted = crop_yields.get((i, j, m), 0) * (1 + epsilon) # 调整后的亩产量

            # 添加目标函数项（收入 - 成本）
            revenue = Z[i][j][k][m] * price_adjusted # 收入 = 实际销售量 * 调整后的销售价格
            cost = A[i][j][k][m] * crop_costs.get((i, j, m), 0) * (1.05 ** (k - 1)) # 成本 =
                种植面积 * 每年增长5%的种植成本
            objective_terms.append(revenue - cost) # 收集目标函数项：收入 - 成本

            # 记录价格、成本和亩产量数据
            price_cost_yield_records[(i, j, k, m)] = {
                '价格': price_adjusted,
                '成本': crop_costs.get((i, j, m), 0) * (1.05 ** (k - 1)),
                '亩产量': yield_adjusted # 记录调整后的亩产量
            }

# 设置唯一的目标函数为所有目标项的总和
model += pl.lpSum(objective_terms) # 最终的目标函数是所有目标项的总和


# 记录已经在final_data中指定的2023年种植组合
specified_combinations = set()

# 首先处理已指定的种植计划（k=0）
for _, row in final_data.iterrows():
    i = row['作物编号']
    j = row['地块编号']
    m = row['种植季次编号']
    # 设置X为1，表示种植
    model += X[i][j][0][m] == 1
    # 设置种植面积
```

```
        model += A[i][j][0][m] == row['种植面积/亩']
        # 记录该组合
        specified_combinations.add((i, j, 0, m))

# 对于未指定的组合，将X和A置为0 (k=0)
for i in crop_ids:
    for j in plot_ids:
        for m in season_ids:
            if (i, j, 0, m) not in specified_combinations:
                model += X[i][j][0][m] == 0
                model += A[i][j][0][m] == 0

# 约束条件
# 约束：大麦 (编号为15) 和玉米 (假设编号为7) 的总种植面积  总面积的30%
for k in range(1, num_years):
    for m in season_ids:
        model += pl.lpSum(A[15][j][k][m] + A[7][j][k][m] for j in plot_ids) >= 0.3 *
            pl.lpSum(plot_areas[j] for j in plot_ids)

# 黄瓜编号为29，青椒的编号为24
for k in range(1, num_years):
    for m in season_ids:
        model += pl.lpSum(A[29][j][k][m] + A[24][j][k][m] for j in plot_ids) <= 0.5 *
            pl.lpSum(plot_areas[j] for j in plot_ids), f"Cucumber_Pepper_Max_Area_{k}_{m}"

# 生菜编号为30，小青菜的编号为28
for k in range(1, num_years):
    for m in season_ids:
        model += pl.lpSum(A[30][j][k][m] + A[28][j][k][m] for j in plot_ids) >= 0.2 *
            pl.lpSum(plot_areas[j] for j in plot_ids), f"Lettuce_BokChoy_Min_Area_{k}_{m}"

# 小麦编号为6，水稻编号为16
wheat_rice_area_constant = 1000 # 这里用常数值，您可以根据需要调整
for k in range(1, num_years):
    for m in season_ids:
        model += pl.lpSum(A[6][j][k][m] + A[16][j][k][m] for j in plot_ids) ==
            wheat_rice_area_constant, f"Wheat_Rice_Constant_Area_{k}_{m}"



# 1. 每块地在3年内应至少种植豆类作物
for j in plot_ids:
    for t in range(1, num_years - 1): # 从k=1开始
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for k in range(t-1, t+2) for m in
            season_ids if final_data.loc[final_data['作物编号'] == i, '是否为豆类'].values[0] ==
            1) >= plot_areas[j]
```

```
# 2. 限制每个季节种植某种作物的地块数量 (k=1到k=7)
N_max = 5
for i in crop_ids:
    for k in range(1, num_years):
        for m in season_ids:
            model += pl.lpSum(X[i][j][k][m] for j in plot_ids) <= N_max

# 3. 每块地的最小种植面积 (k=1到k=7)
A_min_percent = 0.5 # 假设每个地块至少种植50%的面积
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years):
            for m in season_ids:
                model += A[i][j][k][m] >= A_min_percent * plot_areas[j] * X[i][j][k][m]

# 4. 总种植面积不能超过地块的可用面积 (k=1到k=7)
min_utilization = 0.8 # 每个地块至少使用80%的面积
for j in plot_ids:
    for k in range(1, num_years):
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for m in season_ids) >=
            min_utilization * plot_areas[j]

# 5. 各种地块类型的种植条件限制 (k=1到k=7)
# 平旱地、梯田和山坡地只能种植粮食作物，且相邻年份不能种植相同的作物
for plot_category in ['平旱地', '梯田', '山坡地']:
    for j in plot_types[plot_category]: # 遍历平旱地、梯田和山坡地
        # 添加对非粮食作物的限制：蔬菜和食用菌不能种植在这些地块
        for i in crop_types['蔬菜作物']: # 禁止蔬菜作物
            for k in range(num_years): # 遍历每一年
                model += X[i][j][k][1] == 0 # 禁止蔬菜在这些地块的第1季种植

        for i in crop_types['食用菌作物']: # 禁止食用菌作物
            for k in range(num_years): # 遍历每一年
                model += X[i][j][k][1] == 0 # 禁止食用菌在这些地块的第1季种植

        # 对粮食作物的约束
        for i in crop_types['粮食作物']: # 允许种植粮食作物
            for k in range(num_years): # 遍历每一年
                # 限制不能种水稻（编号16）
                if i == 16: # 水稻的编号是16
                    model += X[i][j][k][1] == 0 # 禁止在这些地块种植水稻
            for k in range(num_years - 1):
                # 限制相邻年份同一地块不能连续种植同一种粮食作物
                model += X[i][j][k][1] + X[i][j][k+1][1] <= 1

# 水浇地每年只能种植一季水稻或两季蔬菜，且相邻年份水稻不能重茬
for j in plot_types['水浇地']: # 遍历水浇地
```

```
    for k in range(num_years - 1):
        # 水稻编号为16，限制水稻相邻年份不能重茬
        model += X[16][j][k][1] + X[16][j][k+1][1] <= 1
    for i in crop_types['蔬菜作物']: # 蔬菜作物
        for k in range(num_years):
            # 连续不能种植两季蔬菜
            model += X[i][j][k][1] + X[i][j][k][2] <= 1
        for k in range(num_years-1):
            model += X[i][j][k][2] + X[i][j][k+1][1] <= 1
        # 限制大白菜（35）、白萝卜（36）、红萝卜（37）不能在第一季种植
        model += X[35][j][k][1] == 0 # 大白菜
        model += X[36][j][k][1] == 0 # 白萝卜
        model += X[37][j][k][1] == 0 # 红萝卜
    for i in crop_types['粮食作物']: # 粮食作物
        for k in range(num_years):
            if i != 16:
                # 限制粮食（除了水稻）不能在大棚种植
                model += X[i][j][k][1] == 0
                model += X[i][j][k][2] == 0
    for i in crop_types['食用菌作物']: # 食用菌作物
        for k in range(num_years):
            # 限制粮食不能在大棚种植
            model += X[i][j][k][1] == 0
            model += X[i][j][k][2] == 0

# 普通大棚每年适宜种植一季蔬菜和一季食用菌且蔬菜只能在第一季种植、食用菌只能在第二季种植
for j in plot_types['普通大棚']: # 遍历普通大棚
    for k in range(num_years):
        for i in crop_types['蔬菜作物']: # 蔬菜作物
            # 限制蔬菜只能在第一季种植
            model += X[i][j][k][2] == 0
        for i in crop_types['粮食作物']: # 粮食作物
            # 限制粮食不能种植
            model += X[i][j][k][2] == 0
            model += X[i][j][k][1] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            # 限制食用菌只能在第二季种植
            model += X[i][j][k][1] == 0

# 智慧大棚每年种植两季蔬菜，且相邻季节不能种植相同的作物
for j in plot_types['智慧大棚']: # 遍历智慧大棚
    for i in crop_types['蔬菜作物']: # 蔬菜作物
        for k in range(num_years - 1):
            # 同年不能种植相同的蔬菜
            model += X[i][j][k][1] + X[i][j][k][2] <= 1
            # 相邻年份不能重茬
            model += X[i][j][k][2] + X[i][j][k+1][1] <= 1
```

```
    for k in range(num_years):
        for i in crop_types['粮食作物']: # 粮食作物
            model += X[i][j][k][2] == 0
            model += X[i][j][k][1] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            model += X[i][j][k][2] == 0
            model += X[i][j][k][1] == 0

# 定作物的种植约束（大白菜、白萝卜、红萝卜只能在水浇地的第二季种植）
for plot_category in plot_types: # 遍历所有地块类型
    if plot_category != '水浇地': # 如果不是水浇地
        for j in plot_types[plot_category]:
            for k in range(num_years):
                # 大白菜（35）、白萝卜（36）、红萝卜（37）不能在非水浇地种植
                model += X[35][j][k][1] == 0 # 大白菜
                model += X[36][j][k][1] == 0 # 白萝卜
                model += X[37][j][k][1] == 0 # 红萝卜
                # 水稻（16）不能在非水浇地种植
                model += X[16][j][k][1] == 0

# 求解模型
model.solve()

# 初始解状态
initial_profit = pl.value(model.objective)
print(f"初始最优利润: {initial_profit}")

# 退火算法优化部分
def simulated_annealing(solution, temperature, cooling_rate=0.99):
    best_solution = solution.copy()
    current_solution = solution.copy()
    current_profit = initial_profit
    best_profit = current_profit

    while temperature > 1:
        # 随机改变解
        new_solution = perturb_solution(current_solution)
        new_profit = evaluate_solution(new_solution)

        # 如果新解更优或以一定概率接受较差解
        if new_profit > current_profit or random.uniform(0, 1) < np.exp((new_profit -
             current_profit) / temperature):
            current_solution = new_solution
            current_profit = new_profit
            if new_profit > best_profit:
                best_solution = new_solution
                best_profit = new_profit
```

```
        # 降低温度
        temperature *= cooling_rate

    return best_solution, best_profit

def evaluate_solution(solution):
    # 计算解的利润
    profit = 0
    for i in crop_ids:
        for j in plot_ids:
            for k in range(1, num_years):
                for m in season_ids:
                    area = solution.get((i, j, k, m), 0)
                    yield_per_acre = crop_yields.get((i, j, m), 0)
                    cost_per_acre = crop_costs.get((i, j, m), 0) * (1.05 ** (k - 1)) #
                        每年成本增长5%

                    # 价格调整
                    if i in crop_types['蔬菜作物']:
                        # 蔬菜价格每年上涨5%
                        price_adjusted = crop_prices.get((i, j, m), 0) * (1 +
                            vegetable_price_growth) ** k
                    elif i in crop_types['食用菌作物']:
                        if crop_names[i] == "羊肚菌":
                            # 羊肚菌价格每年下降5%
                            price_adjusted = crop_prices.get((i, j, m), 0) * (1 - 0.05) ** k
                        else:
                            # 其他食用菌价格随机下降1%-5%
                            fungi_decrease = random.uniform(fungi_price_decrease[0],
                                fungi_price_decrease[1])
                            price_adjusted = crop_prices.get((i, j, m), 0) * (1 - fungi_decrease)
                                ** k
                    else:
                        # 其他作物价格保持不变
                        price_adjusted = crop_prices.get((i, j, m), 0)

                    # 计算利润 = 实际面积的收入 - 成本
                    revenue = area * yield_per_acre * price_adjusted
                    cost = area * cost_per_acre
                    profit += revenue - cost

    print(profit) # 输出利润以便调试
    return profit


def perturb_solution(solution):
```

```
    new_solution = solution.copy()
    i = random.choice(crop_ids)
    j = random.choice(plot_ids)
    k = random.randint(1, num_years - 1)
    m = random.choice(season_ids)

    # 确保作物只能在允许的季节种植
    if i in crop_types['蔬菜作物'] and m == 2: # 蔬菜只能在第一季种植
        return solution
    if i in crop_types['食用菌作物'] and m == 1: # 食用菌只能在第二季种植
        return solution

    # 确保种植面积不会超过地块可用面积
    max_area = plot_areas[j]
    used_area = sum(new_solution.get((i, j, k, m), 0) for i in crop_ids)
    available_area = max_area - used_area

    # 控制调整幅度
    adjustment_factor = 0.1 # 调整幅度为10%

    if available_area > 0:
        new_solution[(i, j, k, m)] = new_solution.get((i, j, k, m), 0) * (1 +
            random.uniform(-adjustment_factor, adjustment_factor))

    return new_solution



# 将初始解提取为退火算法的输入
initial_solution = {(i, j, k, m): pl.value(A[i][j][k][m]) for i in crop_ids for j in plot_ids
    for k in range(1, num_years) for m in season_ids}

# 运行退火算法
final_solution, final_profit = simulated_annealing(initial_solution, temperature=1000)

print(f"退火优化后的利润: {final_profit}")

# 保存退火优化后的结果，并添加价格、成本、亩产量信息
start_year = 2023
for k in range(1, num_years): # 遍历每年
    year_solution = []
    for i in crop_ids:
        for j in plot_ids:
            for m in season_ids:
                a_value = final_solution.get((i, j, k, m), 0)
                if a_value > 0:
                    # 从求解时的记录中获取价格、成本和亩产量
```

```
                record = price_cost_yield_records[(i, j, k, m)]
                price_adjusted = record['价格']
                cost_adjusted = record['成本']
                yield_adjusted = record['亩产量']

                # 将结果存储为列表
                year_solution.append([
                    crop_names[i], # 作物名称
                    plot_names[j], # 地块名称
                    m, # 种植季次
                    a_value, # 种植面积
                    price_adjusted, # 调整后的销售单价
                    cost_adjusted, # 调整后的种植成本
                    yield_adjusted # 调整后的亩产量
                ])

    # 转换为DataFrame
    year_solution_df = pd.DataFrame(
        year_solution,
        columns=['作物名称', '地块名称', '种植季次', '种植面积/亩', '调整后的销售单价',
            '调整后的种植成本', '调整后的亩产量']
    )

    # 保存结果为CSV文件
    year_solution_df.to_csv(f"问题二/结果/Q2鲁棒MIP退火结果_{start_year + k}_带单价成本产量.csv",
        index=False, encoding='utf-8-sig')
    print(f"{start_year + k}年优化后的种植计划（包含单价、成本和亩产量）已保存为CSV文件")
```

附录 F　灵敏度分析

6.1　灵敏度实验

```
import pandas as pd
import pulp as pl
import numpy as np
import random
import math

# 加载数据
final_data = pd.read_excel('./data/final.xlsx') # 使用你上传的文件
Basic_data = pd.read_excel('./data/Basis.xlsx')
plot_areas_data = pd.read_excel('./data/Plot_base.xlsx')

# 提取相关数据用于模型构建
```

```
crop_ids = Basic_data['作物编号'].unique() # 作物编号
plot_ids = plot_areas_data['地块编号'].unique() # 地块编号
season_ids = Basic_data['种植季次编号'].unique() # 种植季次编号
num_years = 7 + 1 # 我们计划的年限是7年（从2024到2030）

# 地块类型分类
plot_types = {
    '平旱地': list(range(1, 7)),
    '梯田': list(range(7, 21)),
    '山坡地': list(range(21, 27)),
    '水浇地': list(range(27, 35)),
    '普通大棚': list(range(35, 51)),
    '智慧大棚': list(range(51, 55))
}
# 地块类型编号
basic_plot_types = {
    '平旱地': 1,
    '梯田': 2,
    '山坡地': 3,
    '水浇地': 4,
    '普通大棚': 5,
    '智慧大棚': 6
}

# 使用映射来关联地块编号和地块类型编号
def get_plot_type(j):
    for plot_type, plot_list in plot_types.items():
        if j in plot_list:
            return basic_plot_types[plot_type]
    return None

# 作物类型分类
crop_types = {
    '粮食作物': list(range(1, 17)), # 1-16 为粮食作物
    '蔬菜作物': list(range(17, 38)), # 17-37 为蔬菜作物
    '食用菌作物': list(range(38, 42)) # 38-41 为食用菌作物
}

# 打印映射结果，检查映射是否正确
# print(plot_type_to_plots)

# 构建作物数据的字典
crop_names = dict(zip(Basic_data['作物编号'], Basic_data['作物名称'])) # 作物编号和名称的映射
plot_names = dict(zip(plot_areas_data['地块编号'], plot_areas_data['地块名称'])) #
    地块编号和名称的映射

# 首先对相同作物编号、地块类型、种植季次编号的对应销量求和
```

```
grouped_data = final_data.groupby(['作物编号', '种植季次编号'])['对应销量'].sum().reset_index()

# 生成字典
crop_sales_limits = dict(zip(zip(grouped_data['作物编号'], grouped_data['种植季次编号']),
    grouped_data['对应销量']))
plot_areas = dict(zip(plot_areas_data['地块编号'], plot_areas_data['地块面积/亩'])) # 地块面积

# 使用映射来关联地块编号和地块类型编号
crop_yields = dict(zip(
   zip(Basic_data['作物编号'], Basic_data['地块类型编号'], Basic_data['种植季次编号']),
   Basic_data['亩产量/斤']
))

crop_costs = dict(zip(
   zip(Basic_data['作物编号'], Basic_data['地块类型编号'], Basic_data['种植季次编号']),
   Basic_data['种植成本/(元/亩)']
))

crop_prices = dict(zip(
   zip(Basic_data['作物编号'], Basic_data['种植季次编号']),
   Basic_data['销售单价(均值)']
))

# 初始化优化模型 (最大化利润)
model = pl.LpProblem("Crop_Planning_Optimization", pl.LpMaximize)

# 决策变量
X = pl.LpVariable.dicts("X", (crop_ids, plot_ids, range(num_years), season_ids), cat='Binary')
    # 是否种植的二元变量
A = pl.LpVariable.dicts("A", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0) #
    种植面积的连续变量
Z = pl.LpVariable.dicts("Z", (crop_ids, plot_ids, range(num_years), season_ids), lowBound=0)

# 为Z添加约束条件，按照地块编号 j 确定地块类型编号
for i in crop_ids:
   for j in plot_ids:
      plot_type_id = get_plot_type(j) # 获取地块类型编号
      for k in range(1, num_years): # 从k=1开始，忽略k=0
         for m in season_ids:
            # 使用地块类型编号来访问 crop_yields 中的对应值
            model += Z[i][j][k][m] <= A[i][j][k][m] * crop_yields.get((i, get_plot_type(j),
                m), 0) # Z <= A * 单产量

# 1. 引入新的变量 Z_total 用于表示作物的总销售量
Z_total = pl.LpVariable.dicts("Z_total", (crop_ids, range(num_years), season_ids), lowBound=0)

# 2. 对每个作物的总销售量添加约束，确保其不超过预测销量
```

```
for i in crop_ids:
    for k in range(1, num_years):
        for m in season_ids:
            model += Z_total[i][k][m] == pl.lpSum([Z[i][j][k][m] for j in plot_ids]) # Z_total
                是所有地块销量的总和
            model += Z_total[i][k][m] <= crop_sales_limits.get((i, m), 0) # Z_total 不超过销量上限

# 3. 修改目标函数：使用 Z_total 计算总利润
model += pl.lpSum([
    Z_total[i][k][m] * crop_prices.get((i, m), 0) # 使用 Z_total 计算利润
    - pl.lpSum([A[i][j][k][m] * crop_costs.get((i, get_plot_type(j), m), 0) for j in plot_ids])
        # 计算总成本
    for i in crop_ids
    for k in range(1, num_years) # 从k=1到k=7
    for m in season_ids
])

# 记录已经在final_data中指定的2023年种植组合
specified_combinations = set()

# 首先处理已指定的种植计划 (k=0)
for _, row in final_data.iterrows():
    i = row['作物编号']
    j = row['地块编号']
    m = row['种植季次编号']
    # 设置X为1，表示种植
    model += X[i][j][0][m] == 1
    # 设置种植面积
    model += A[i][j][0][m] == row['种植面积/亩']
    # 记录该组合
    specified_combinations.add((i, j, 0, m))

# 对于未指定的组合，将X和A置为0 (k=0)
for i in crop_ids:
    for j in plot_ids:
        for m in season_ids:
            if (i, j, 0, m) not in specified_combinations:
                model += X[i][j][0][m] == 0
                model += A[i][j][0][m] == 0

# 添加约束：如果 X 为 0，则 A 必须为 0
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years):
            for m in season_ids:
                model += A[i][j][k][m] <= plot_areas[j] * X[i][j][k][m]
```

```
# 约束条件
# 1. 每块地在3年内应至少种植豆类作物
for j in plot_ids:
    for t in range(1, num_years - 1): # 从k=1开始
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for k in range(t - 1, t + 2) for m in
            season_ids if
                        final_data.loc[final_data['作物编号'] == i, '是否为豆类'].values[0] == 1)
                            >= plot_areas[j]

# 2. 限制每个季节种植某种作物的地块数量 (k=1到k=7)
N_max = 5
for i in crop_ids:
    for k in range(1, num_years):
        for m in season_ids:
            model += pl.lpSum(X[i][j][k][m] for j in plot_ids) <= N_max

# 3. 每块地的最小种植面积 (k=1到k=7)
A_min_percent = 0.4 # 假设每个地块至少种植50%的面积
for i in crop_ids:
    for j in plot_ids:
        for k in range(1, num_years):
            for m in season_ids:
                model += A[i][j][k][m] >= A_min_percent * plot_areas[j] * X[i][j][k][m]

# 4. 总种植面积不能超过地块的可用面积 (k=1到k=7)
min_utilization = 0.8 # 每个地块至少使用80%的面积
for j in plot_ids:
    for k in range(1, num_years):
        model += pl.lpSum(A[i][j][k][m] for i in crop_ids for m in season_ids) >=
            min_utilization * plot_areas[j]

# 5. 各种地块类型的种植条件限制 (k=1到k=7)
# 平旱地、梯田和山坡地只能种植粮食作物，且相邻年份不能种植相同的作物，且只能种单季 m=0
for plot_category in ['平旱地', '梯田', '山坡地']:
    for j in plot_types[plot_category]: # 遍历平旱地、梯田和山坡地
        # 限制只能种单季 m=0
        for k in range(1, num_years): # 遍历每一年
            for m in [1, 2]: # 限制双季种植 (m=1 和 m=2)
                # 不能在双季种植任何作物
                for i in crop_ids:
                    model += X[i][j][k][m] == 0

            # 只能种粮食作物
            for i in crop_types['蔬菜作物'] + crop_types['食用菌作物']: #
                禁止蔬菜和食用菌作物在这些地块种植
                model += X[i][j][k][0] == 0 # 禁止在单季种植非粮食作物
```

```
        # 对粮食作物的约束
        for i in crop_types['粮食作物']: # 允许种植粮食作物
            for k in range(1, num_years): # 遍历每一年
                # 限制不能种水稻（编号16）
                if i == 16: # 水稻的编号是16
                    model += X[i][j][k][0] == 0 # 禁止水稻在这些地块种植
            for k in range(num_years - 1):
                # 限制相邻年份同一地块不能连续种植同一种粮食作物
                model += X[i][j][k][0] + X[i][j][k + 1][0] <= 1

# 水浇地可以一年种植一季水稻，且相邻年份水稻不能重茬
# 限制双季（m=1 和 m=2）不允许种粮食作物、食用菌作物
# 水浇地可以第一季种植蔬菜（大白菜、白萝卜和红萝卜除外），第二季只能种大白菜、白萝卜和红萝卜其中一种
for j in plot_types['水浇地']: # 遍历所有水浇地
    for k in range(1, num_years):
        # 限制只能在单季种植水稻（编号16），且相邻年份水稻不能连续种植
        model += X[16][j][k][1] == 0 # 不能按两季种
        model += X[16][j][k][2] == 0

        # 其他粮食作物（除水稻外）不能在水浇地种植
        for i in crop_types['粮食作物']:
            if i != 16: # 除水稻外的粮食作物
                for m in [0, 1, 2]: # 限制全年不种植
                    model += X[i][j][k][m] == 0

        # 第一季可以种蔬菜，但不允许种大白菜（35）、白萝卜（36）、红萝卜（37）
        for i in [35, 36, 37]: # 限制特定蔬菜
            model += X[i][j][k][1] == 0 # 第一季不允许种植大白菜（35）、白萝卜（36）、红萝卜（37）
            model += X[i][j][k][0] == 0
        for i in crop_types['蔬菜作物']: # 第二季只能种植大白菜（35）、白萝卜（36）、红萝卜（37）
            if i != [35, 36, 37]: # 仅允许这三种蔬菜在第二季种植
                model += X[i][j][k][2] == 0 # 第二季不能种植其他蔬菜
                model += X[i][j][k][0] == 0
        model += X[35][j][k][2] + X[36][j][k][2] + X[37][j][k][2] <= 1

        # 食用菌作物不能在水浇地种植
        for i in crop_types['食用菌作物']:
            for m in [0, 1, 2]: # 全年不允许食用菌作物
                model += X[i][j][k][m] == 0

    # 相邻年份不能连续种植水稻
    for k in range(num_years - 1):
        model += X[16][j][k][0] + X[16][j][k + 1][0] <= 1

#
    普通大棚每年适宜种植一季蔬菜(35,36,37除外)和一季食用菌且蔬菜只能在第一季种植、食用菌只能在第二季种植
for j in plot_types['普通大棚']: # 遍历普通大棚
```

```
    for k in range(1, num_years):
        for i in crop_types['蔬菜作物']: # 蔬菜作物
            if i in [35, 36, 37]:
                model += X[i][j][k][1] == 0
            # 限制蔬菜只能在第一季种植
            model += X[i][j][k][2] == 0
            model += X[i][j][k][0] == 0
        for i in crop_types['粮食作物']: # 粮食作物
            # 限制粮食不能种植
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            # 限制食用菌只能在第二季种植
            model += X[i][j][k][1] == 0
            model += X[i][j][k][0] == 0

# 智慧大棚每年种植两季蔬菜(35,36,37除外), 且相邻季节不能种植相同的作物
for j in plot_types['智慧大棚']: # 遍历智慧大棚
    for i in crop_types['蔬菜作物']: # 蔬菜作物
        for k in range(num_years - 1):
            # 相邻年份不能重茬
            model += X[i][j][k][2] + X[i][j][k + 1][1] <= 1
        for k in range(num_years):
            # 同年不能种植相同的蔬菜
            model += X[i][j][k][1] + X[i][j][k][2] <= 1

    for k in range(1, num_years):
        for i in crop_types['蔬菜作物']: # 蔬菜作物不能种单季
            model += X[i][j][k][0] == 0
            if i in [35, 36, 37]: # 蔬菜中的大白菜 (35) 、白萝卜 (36) 、红萝卜 (37) 不能种
                model += X[i][j][k][1] == 0
                model += X[i][j][k][2] == 0
        for i in crop_types['粮食作物']: # 粮食作物
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0
        for i in crop_types['食用菌作物']: # 食用菌作物
            for m in [0, 1, 2]:
                model += X[i][j][k][m] == 0


def simulated_annealing(model, initial_solution, max_iter=500, initial_temp=10,
    cooling_rate=0.99, tolerance=1e-5):
    """
    模拟退火算法,用于优化PuLP模型的解。

    参数:
    - model: PuLP 模型
```

```
    - initial_solution: 初始解
    - max_iter: 最大迭代次数
    - initial_temp: 初始温度
    - cooling_rate: 温度递减率
    - tolerance: 目标函数差值的容忍度，改进小于该值时提前停止

    返回:
    - 最优解和最优目标函数值
    """
    current_solution = initial_solution
    current_value = pl.value(model.objective)

    best_solution = current_solution
    best_value = current_value

    temperature = initial_temp

    for iteration in range(max_iter):
        new_solution = generate_neighbor_solution(current_solution)
        apply_solution_to_model(model, new_solution)

        # 重新求解模型
        model.solve()
        new_value = pl.value(model.objective)

        # 计算目标函数差值
        delta = new_value - current_value

        # 根据退火准则决定是否接受新解
        if delta > 0 or random.uniform(0, 1) < math.exp(delta / temperature):
            current_solution = new_solution
            current_value = new_value

            # 更新最优解
            if current_value > best_value:
                best_solution = current_solution
                best_value = current_value

        # 打印当前迭代状态
        print(
            f"迭代 {iteration}: 当前解的目标函数值 = {current_value}, 最优解的目标函数值 =
                {best_value}, 温度 = {temperature}")

        # 如果改进的幅度很小，则提前停止
        if abs(delta) < tolerance:
            print(f"优化幅度低于 {tolerance}, 提前停止.")
            break
```

```
        # 降低温度
        temperature *= cooling_rate

        # 如果温度过低，则停止
        if temperature < 1e-3:
            print("温度过低，停止优化.")
            break

    return best_solution, best_value


def generate_neighbor_solution(current_solution):
    """
    生成当前解的邻域解。

    参数:
    - current_solution: 当前解

    返回:
    - 邻域解
    """
    new_solution = current_solution.copy()

    # 随机选择一个作物和地块进行微调
    i = random.choice(list(crop_ids))
    j = random.choice(list(plot_ids))
    k = random.randint(1, num_years - 1)
    m = random.choice(list(season_ids))

    # 随机调整种植面积（例如上下波动 5%）
    adjustment = random.uniform(-0.05, 0.05) * plot_areas[j]
    new_solution[(i, j, k, m)] = max(0, current_solution[(i, j, k, m)] + adjustment)

    return new_solution


def apply_solution_to_model(model, solution):
    """
    将解应用到模型的决策变量中。

    参数:
    - model: PuLP 模型
    - solution: 新的解
    """
    for i in crop_ids:
        for j in plot_ids:
```

```
            for k in range(1, num_years):
                for m in season_ids:
                    # 获取解中的值，如果不存在，则默认为 0
                    value = solution.get((i, j, k, m), 0)
                    if value is not None: # 确保值不为 None
                        A[i][j][k][m].setInitialValue(value)


# 求解模型，获取初始解
solver = pl.PULP_CBC_CMD(msg=True, threads=8, timeLimit=10)
model.solve(solver)

# 初始解，确保所有变量都有值，默认值为 0
initial_solution = {(i, j, k, m): pl.value(A[i][j][k][m]) if pl.value(A[i][j][k][m]) is not
    None else 0
                    for i in crop_ids for j in plot_ids for k in range(1, num_years) for m in
                        season_ids}

# 应用模拟退火算法进行优化
best_solution, best_value = simulated_annealing(model, initial_solution)

# 打印退火算法优化后的最优解
print(f"退火算法优化后的最优利润: {best_value}")

# 输出最终的最优解并保存
if best_solution:
    start_year = 2023 # 从2024年开始
    for k in range(1, num_years):
        year_solution = []

        for i in crop_ids:
            for j in plot_ids:
                for m in season_ids:
                    # 收集每一年的决策变量值
                    x_value = pl.value(X[i][j][k][m])
                    a_value = best_solution.get((i, j, k, m), 0)

                    # 如果种植面积大于 0，提取对应的单价、亩产量和成本
                    if a_value > 0:
                        unit_price = crop_prices.get((i, m), 0) # 单价
                        yield_per_mu = crop_yields.get((i, get_plot_type(j), m), 0) # 亩产量
                        cost_per_mu = crop_costs.get((i, get_plot_type(j), m), 0) # 种植成本
                        # 将信息添加到year_solution
                        year_solution.append(
                            [crop_names[i], plot_names[j], m, x_value, a_value, unit_price,
                                yield_per_mu, cost_per_mu])
```

```
        # 将结果转换为DataFrame
        year_solution_df = pd.DataFrame(year_solution,
                                        columns=['作物名称', '地块名称', '种植季次', '种每种',
                                            '种植面积/亩',
                                                '单价（元/斤）', '亩产量（斤/亩）',
                                                    '种植成本（元/亩）'])



# 灵敏度分析函数
def sensitivity_analysis_with_annealing(param, values, temperature=1000):
    """
    对给定参数进行灵敏度分析，结合退火算法进行优化
    :param param: 'A_min_percent', 'min_utilization'
    :param values: 参数值的列表
    :param temperature: 初始温度
    :return: 一个DataFrame，显示不同参数值对应的利润
    """
    results = []

    for val in values:
        # 初始化模型副本
        model_copy = model.copy()

        # 修改模型中的相应参数
        if param == 'A_min_percent':
            # 修改 A_min_percent 的约束
            for i in crop_ids:
                for j in plot_ids:
                    for k in range(1, num_years):
                        for m in season_ids:
                            model_copy += A[i][j][k][m] >= val * plot_areas[j] * X[i][j][k][m]

        elif param == 'min_utilization':
            # 修改 min_utilization 的约束
            for j in plot_ids:
                for k in range(1, num_years):
                    model_copy += pl.lpSum(A[i][j][k][m] for i in crop_ids for m in season_ids)
                        >= val * plot_areas[j]

        # 求解修改后的模型
        model_copy.solve()

        # 获取当前解作为退火算法的初始解
        initial_solution = {(i, j, k, m): pl.value(A[i][j][k][m]) for i in crop_ids for j in
            plot_ids for k in
                        range(1, num_years) for m in season_ids}
```

```
            # 运行退火算法进行优化
            final_solution, final_profit = simulated_annealing(initial_solution, temperature)

            # 记录结果
            results.append((param, val, final_profit))

    # 转换为 DataFrame 格式
    results_df = pd.DataFrame(results, columns=['Parameter', 'Value', 'Profit'])
    return results_df



# 定义不同参数的取值范围
A_min_percent_values = np.linspace(0.25,0.65,17)
min_utilization_values = np.linspace(0.525,0.8,12)

# 使用修改后的灵敏度分析函数进行分析
A_min_percent_results = sensitivity_analysis_with_annealing('A_min_percent',
    A_min_percent_values)
min_utilization_results = sensitivity_analysis_with_annealing('min_utilization',
    min_utilization_values)

# 合并所有分析结果
all_results = pd.concat([A_min_percent_results, min_utilization_results])

# 将结果保存为 CSV 文件
all_results.to_csv("sensitivity_analysis_with_annealing_results.csv", index=False)

# 打印结果
print(all_results)
```

6.2 可视化图表

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# 加载数据
file_path = '灵敏度分析.csv'
data = pd.read_csv(file_path)

# 根据不同参数分割数据
a_min_percent_data = data[data['Parameter'] == 'A_min_percent']
```

```
min_utilization_data = data[data['Parameter'] == 'min_utilization']

# 确保 Value 和 Profit 列是数值型
a_min_percent_data['Value'] = pd.to_numeric(a_min_percent_data['Value'], errors='coerce')
a_min_percent_data['Profit'] = pd.to_numeric(a_min_percent_data['Profit'], errors='coerce')
min_utilization_data['Value'] = pd.to_numeric(min_utilization_data['Value'], errors='coerce')
min_utilization_data['Profit'] = pd.to_numeric(min_utilization_data['Profit'], errors='coerce')

plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

# 定义一个函数来绘制每一组数据的折线图
def plot_with_shading(data, title, x_label):
    values = np.array(data['Value'], dtype=float)
    profits = np.array(data['Profit'], dtype=float)
    plt.figure(figsize=(12, 4))
    # 绘制利润曲线（深蓝色并加粗）
    plt.plot(values, profits, color='navy', linewidth=2, label='利润')

    # 获取最大值和最小值，并绘制虚线
    max_profit = profits.max()
    min_profit = profits.min()
    max_profit_line = np.full_like(values, max_profit)
    min_profit_line = np.full_like(values, min_profit)
    plt.axhline(max_profit, color='green', linestyle='--', label=f'最大值: {max_profit:.2f}')
    plt.axhline(min_profit, color='red', linestyle='--', label=f'最小值: {min_profit:.2f}')

    # 在最大值和最小值之间填充灰色阴影
    plt.fill_between(values, min_profit_line, max_profit_line, color='gray', alpha=0.3)

    # 添加标签、标题和网格样式
    plt.xlabel(x_label,fontsize=13)
    plt.ylabel('利润',fontsize=13)
    plt.ylim(2.7e7, 3.9e7)
    plt.title(title)
    plt.title(title, fontsize=15) # 修改标题字体大小

    # 添加浅灰色实线网格
    plt.grid(True, linestyle='-', linewidth=0.5, color='lightgray')
    # 显示图例
    plt.legend()
    # 调整布局
    plt.tight_layout() # 自动调整布局
    # 显示图表
    plt.show()

# 绘制 A_min_percent 图
```

```
plot_with_shading(a_min_percent_data, '利润随 A_min_percent 值的变化', 'A_min_percent 值')
# 绘制 min_utilization 图
plot_with_shading(min_utilization_data, '利润随 min_utilization 值的变化', 'min_utilization 值')
```