

基于规划模型的农作物种植策略优化

摘要

农业作为乡村经济的核心产业，其种植策略的优化不仅影响着农民的收益，也对生态环境和资源的合理利用起着关键作用。本研究针对华北山区某乡村的农作物种植策略优化问题，旨在制定 2024-2030 年期间的最优种植方案，以最大化经济效益，并应对市场不确定性、政策变化及环境保护需求。在农业发展面临气候变化、市场波动和资源限制的背景下，优化种植策略对于提高农业生产效率、增加农民收入以及促进乡村经济的可持续发展具有重要意义。

问题一中，我们解决了在稳定市场条件下的农作物种植策略优化问题，应用线性规划模型，通过有效分配作物提高了总收入。

问题二中，我们处理了销售量、价格和气候的不确定性，采用鲁棒优化和蒙特卡洛模拟，得到了在收益和风险之间平衡的策略。

问题三中，我们考虑了作物替代性和互补性，以及经济模型中内含的各种价格、成本等之间的关系，并将这些关系整合到模型中以进一步优化种植策略。本文强调了适应性策略在农业规划中的重要性，为可持续农村发展提供了见解。

通过本研究，我们不仅为华北山区的农作物种植策略提供了科学的优化方案，还为应对农业领域的复杂不确定性提供了新的视角。我们的模型强调了适应性策略在农业规划中的重要性，能够在多变的市场和环境条件下实现经济效益的最大化。该研究的亮点在于结合了线性规划、蒙特卡洛模拟，鲁棒优化和多目标优化等多种方法，为可持续农村发展提供了有力的理论支持和实践指导。未来，我们可以将遗传算法和模拟退火算法等整合到模型中，以求得进一步对模型的改进。

关键字： 农作物种植优化, 线性规划, 蒙特卡洛模拟, 鲁棒优化, 多目标优化, 不确定性分析

一、问题的提出与重述

1.1 问题的提出

随着全球对可持续发展的日益重视，有机农业和高效农业管理已逐渐成为研究的热点领域。已有研究显示，合理规划农作物的种植方案可以提高产量、降低种植风险，并促进乡村经济的可持续发展 [1]。同时，农作物的生长周期、气候条件以及市场需求等因素在农业管理中的重要性也日益凸显。在优化种植方案时，不仅需要考虑单一作物的经济收益，还应平衡多种作物的合理组合、市场需求和土地的利用效率。本文以华北山区某乡村为背景，乡村常年温度偏低，耕地类型多样。如何在有限的耕地资源上制定合理的农作物种植策略，以最大化经济效益并减少种植风险，成为亟待解决的问题。因此，本研究聚焦于华北山区某乡村的农作物种植策略优化问题，旨在通过科学的模型和方法提升乡村的经济效益。研究的核心问题是如何在未来七年（2024-2030 年）内，在多种市场和环境条件下，优化农作物的种植方案，以实现经济收益最大化。

1.2 背景描述

某华北乡村共有 34 块露天耕地，分为平旱地、梯田、山坡地和水浇地 4 种类型，不同地块适宜种植不同作物。此外，该乡村还有 16 个普通大棚和 4 个智慧大棚，适合种植不同类型的蔬菜和食用菌。此外，农作物不能连续重茬种植，每块地必须每三年种植一次豆类作物以改良土壤条件。

1.3 问题的重述

问题涉及农作物销售量、成本、产量等因素，要求制定 2024-2030 年的农作物种植方案。方案需要在多样化耕地类型和销售条件下，找到收益最大、风险最小的种植策略。

问题 1：要求制定 2024-2030 年间的种植方案，假定各类作物的销售量、成本和销售价格与 2023 年保持一致。在此基础上，分别针对两种情况设计最优方案：第一种情况是某种作物的产量超过预期销售量时，超过部分将被浪费；第二种情况是超过部分以 2023 年价格的 50% 进行降价销售。因此需要根据不同作物的特性和地块的适宜性，平衡每季的产量与销售量，避免浪费或损失。

问题 2：小麦和玉米的预期销售量将以 5%-10% 的速度逐年增长，其他作物的销售量变化范围在 $\pm 5\%$ 。此外，作物的亩产量将因气候波动出现 $\pm 10\%$ 的浮动，种植成本每年预计增长 5%，部分蔬菜价格也有一定上升趋势。在这些不确定性因素下，需对未来 7 年的种植方案进行优化，找到在收益和风险之间的平衡点。

问题 3: 在现实中, 农作物之间可能存在替代性和互补性, 销售量、种植成本和价格之间也存在一定的相关性。基于问题 2 的模型, 需要综合考虑这些复杂的关联性, 找到一套更加合理的种植方案, 并与问题 2 的结果进行对比分析。通过模拟数据, 我们可以进一步探索农作物之间的动态关系, 优化整个乡村的种植策略。

具体来说, 本研究将问题划分为以下几个部分:

最优种植方案的制定: 在假设未来销售量、种植成本、亩产量和销售价格稳定的条件下, 确定各类土地和大棚上最优的作物种植组合, 目标是最大化总收益。

应对不确定性条件的优化: 在考虑未来销售量、价格、气候变化和种植成本等不确定因素的影响下, 优化种植策略以确保收益的稳定性和风险的最小化。

综合考虑作物间关系的优化: 进一步分析作物之间的替代性和互补性, 以及销售量、价格和成本之间的相关性, 制定一个综合效益更高的种植方案。

本研究将通过建立线性规划模型和多目标优化模型, 对上述问题进行系统分析和优化。

二、问题分析

问题一的分析在假设未来几年农作物的销售量、种植成本、亩产量和销售价格保持相对稳定的前提下, 问题的目标是为该乡村在 2024-2030 年期间制定一个最优的农作物种植方案, 以最大化其经济效益。

2.1 问题一的分析

在假设未来几年农作物的销售量、种植成本、亩产量和销售价格保持相对稳定的前提下, 问题一的目标是为该乡村在 2024-2030 年期间制定最大化其经济效益的农作物种植方案。因此, 该方案需要综合考虑多种因素, 包括不同土地类型的种植条件、作物的分布均匀性以及种植过程中所涉及的各种限制条件。

为实现本目标, 我们采用线性规划模型, 将作物种类对应的种植面积以及是否种植该作物作为决策变量, 并将模型的目标函数设定为总收益, 即所有作物的销售收入减去相应的种植成本, 目标函数可以表示为:

$$\zeta = \sum_{c=1}^n \sum_{r=1}^m (P_{c,s} \cdot Y_{c,r} \cdot A_{c,r,y,s} - C_{c,r} \cdot A_{c,r,y,s}) \quad (1)$$

该模型的约束条件包括多方面的要求: 首先, 不同类型的土地(如平旱地、梯田、山坡地和水浇地)和大棚具有各自适宜种植的作物类型, 因此需限制作物的种植地; 其次, 需遵循重茬限制, 确保同一地块内不允许连续种植相同的作物, 以避免减产风险; 此外, 还要满足豆类作物的种植要求, 即在 2024-2030 年期间, 每个地块或大棚三年内至少种植一次豆类作物; 其次, 考虑到市场销售情况, 每种作物的总产量不能超过其预期销售

量，避免因产量过剩而导致的滞销或降价处理的经济损失；同时，方案还需考虑作物分布的均匀性，确保种植过程便于管理，避免过于分散或种植面积过小的情况。

在模型构建中，需要假设未来几年内各类作物的销售量和价格保持稳定，每种作物的亩产量和种植成本不变。这意味着可以使用 2023 年的数据作为模型输入，包括各类作物的亩产量、种植成本、销售价格和销售量，以及各个地块和大棚的类型、面积和种植历史数据，这些数据将用来设定模型的参数和约束条件。

总而言之，我们需要构建一个以最大化总收益为目标的目标函数，同时建立相应的涵盖种植类型、重茬限制、豆类种植要求等多个方面的约束条件。通过使用线性规划求解算法来求解该模型，并验证结果是否符合所有的约束条件。最终，通过调整模型参数和约束条件，可以进一步优化方案，以确保其在实际操作中具备可行性和最大化经济效益的潜力。通过这样的分析方式，可以得出符合 2024-2030 年经济效益最大化目标的最优种植方案。

2.2 问题二的分析

问题二的设定大体上延续了问题 1 的设定，而进一步提出了农作物的预期销售量、种植成本、亩产量、销售价格的变化趋势。因此我们可以直接沿用问题 1 的模型，在其中增加或更改代码以适用于问题 2 的目标。问题 2 的目标是在考虑未来销售量、价格、气候变化和种植成本等不确定因素的影响下，优化种植策略以确保收益的稳定性和风险的最小化。因此，我们需要在问题 1 的基础上，引入一些新的变量和约束条件，以适应问题 2 的要求。其中，题干中提到一些量的年平均变化率在一个区间内波动，对于这一随机波动的刻画是问题 2 的基础。

问题二中核心问题是将各种农作物的种植风险量化为可编程可计算的代码。在实际农业生产中，自然灾害等风险因素会因地区、作物种类和种植条件的不同而有很大差异。但是我们可以通过查阅文献和综合研判各种风险等方式粗略给出了各种农作物的种植风险，并给出一个固定的数值作为农作物发生减产的概率。

三、模型假设

3.1 问题一的假设

由于在题目给出的条件中不含有预期销售量的相关数据，我们将 2023 年各类作物在两季的总产量假设为 2024 年两季的预期销售量。对于题干中提及的“不能连续重茬种植，否则会减产”，我们认定同一年的第一季和第二季是连续的，上一年的第二季和下一年的第一季也是连续的。对于“每种作物每季的种植地不能太分散”的要求，我们

假设每种作物每个季节种植的地块数不得多于 8 块。对于“每种作物在单个地块（含大棚）种植的面积不宜太小”的要求，我们假设每种作物的种植面积不得小于所在地块面积的 30%。

3.2 问题二的假设

对于题干中声称“基本稳定”的量，我们在模型中作不变处理；对于题干中声称平均年变化率为定值的量，我们将每年的变化率都取该定值；对于题干中声称平均年变化率在一个区间内波动的量，例如“平均年增长率介于 5%~10% 之间”，我们假设年增长率的概率分布在该区间内是均匀的。

3.3 问题三的假设

在问题三中，我们假设以下几点：

1. 作物之间存在一定的替代性和互补性。具体来说，谷物作物和豆类作物在某些情况下可以互相替代，而某些作物在同一地块种植时会产生互补效应。
2. 预期销售量与销售价格、种植成本之间存在相关性，遵循经济学中的需求定律和边际成本原则。
3. 互补性和互斥性在模型中通过固定的增产或减产比例来表示。

3.4 全局假设

1. 假设农民是理性的有预期的种植，产出的产量总是和市场需求大致相同。
2. 假设每年可供种植的土地面积固定且不变，不会因气候或其他因素减少。
3. 假设在同一地块上种植的同种作物的亩产量是恒定的，不受种植面积影响。
4. 假设每种作物的生长期不变，在同一时间段内种植的作物生长周期不受年份变化影响
5. 假设在整个种植期内，不会发生影响农作物生长的病虫害或自然灾害。

四、符号说明

以下是模型中使用的主要符号：

符号	说明	单位
$P_{c,s}$	指定作物在指定季节的 单位重量销售价格	元/斤
$Y_{c,r}$	指定作物在指定地块的 单位面积产量	斤/亩
$C_{c,r}$	指定作物在指定地块的 单位面积成本	元/亩
$A_{c,r,y,s}$	指定作物在指定地块在指定年份在指定季节的 种植面积 (决策变量)	亩
$E_{c,s}$	指定作物在指定季节的 预期销售量	斤
r	reduction rate	
ζ	总收益	元

五、数据侧写

题目提供了附件 1 和附件 2 等数据文件，虽然数据量不大，但其中涉及的各类关系相对复杂。因此，有必要对这些数据进行一个简要的分析和整理，以理清各变量之间的关联性，从而使模型构建的逻辑更加清晰和合理。此外，通过对数据的初步梳理，可以识别出关键的影响因素，确保后续的分析能够准确捕捉到作物之间的相互关系及其对收益的潜在影响，从而为制定最优种植策略打下扎实的基础。

5.1 地块与面积分布

不同地块的面积分布可提供对各地块可用种植面积的直观了解，有助于确定后续种植策略中的面积分配，具体如下图所示：

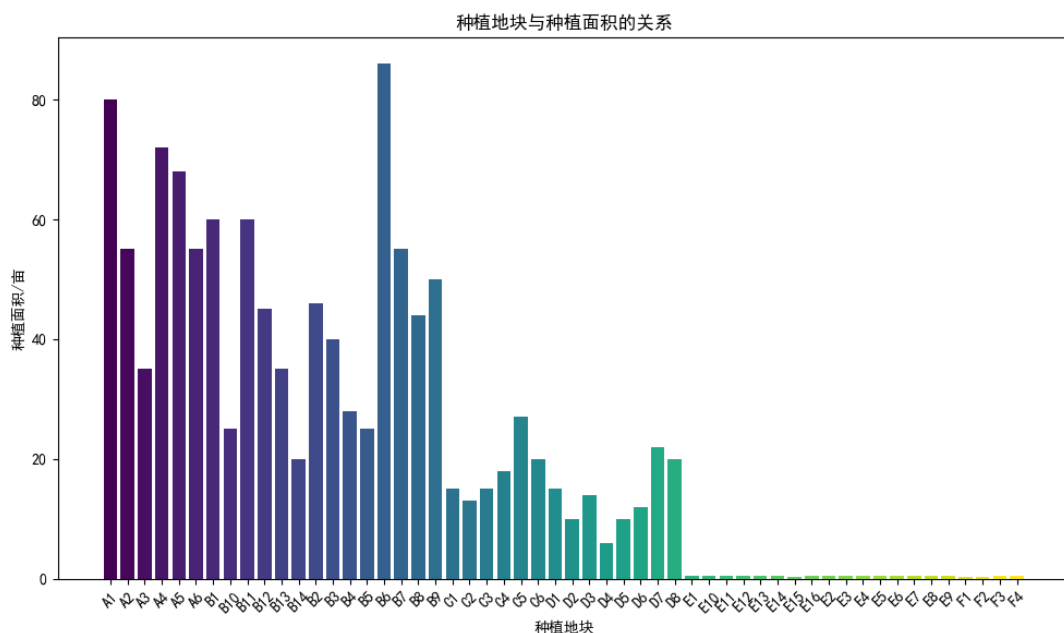


图 1 地块与面积分布图

其中地块 A、B、C、D 分别为干旱地、梯田、山坡地与水浇地。干旱地面积较大，因为地形平坦、开阔，适合大规模种植；梯田面积中等，因地形限制形成的阶梯状结构，适合中规模耕种；山坡地由于坡度大，地形复杂，耕地面积较小；水浇地面积较小，因为水资源的分布有限，适合小规模种植。

总体来看，地块面积的分布反映了自然条件对农业生产的制约和不同地块的适种性。

5.2 各类型作物平均亩产量分析

各类型作物平均亩产量的分析评估了不同作物的单位面积生产效益，以此识别高产作物，提升耕地资源的利用效率，优化作物选择和种植结构，从而提高整体农业产量。分析结果将为制定科学合理的种植规划提供重要的依据，具体如下图所示：

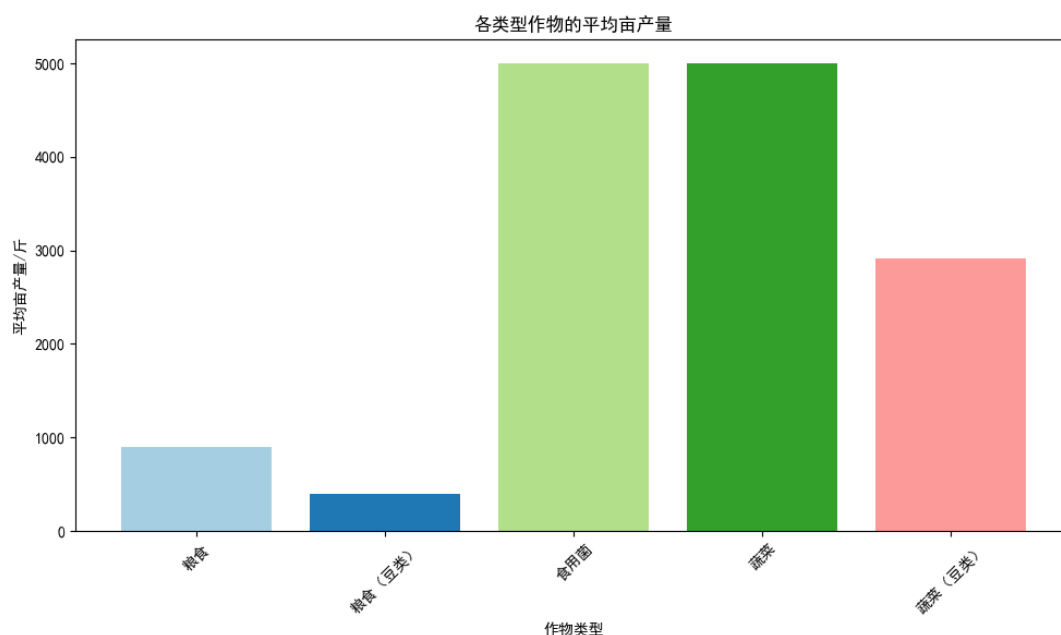


图 2 各类型作物平均亩产量

由图可见，食用菌与蔬菜的平均亩产量最高，其原因一方面在于生长周期较短、种植密度较高，另一方面在于具有大棚等理想种植条件实现多次收获。与之相对地，豆类和粮食作物由于生长周期长，种植密度低，且豆类作物的主要功能在于改善土壤结构，因而其平均亩产量相对较低。

六、准备工作/预处理

6.1 异常检验

本文对映射关系和数据范围进行了检验。首先，验证了附件 1 中各地块的面积是否大于附件 2 中 2023 年各地块的农作物种植面积；其次，检查了作物编号、作物名称、地块名称、地块类型之对应的成本与销售单价是否唯一。经检验，未发现异常数据。

6.2 .csv 文件的生成

我们发现即使 `panda` 库中有 `read_excel()` 函数可以直接读取附件中的 `.xlsx` 文件，但是其有一系列问题，例如 `.xlsx` 中合并的单元格会导致数据读取错误，只有第一个单元格的数据能被读取，并且读取的速度较慢等，因此我们选择将数据转换为 `.csv` 格式。`.csv` 作为文本格式，相较于 `.xlsx` 格式更加方便处理。因此，我们将原始数据转换为 `.csv` 格式，以便于后续的数据处理和模型构建。`src/utls/xlsx to csv/main.py` 中详细记录了生成 `.csv` 文件的源代码。

6.3 full_table.csv 的生成

在仔细阅读附件中的数据后，我们发现附件中的数据分为两部分，一部分是各类农作物的种植成本、亩产量、销售价格和预期销售量等数据，另一部分是各地块和大棚的类型、面积和种植历史数据。有一个很大的问题是为了方便后续的数据处理和模型构建，我们将这两部分数据合并为一个完整的数据表，即 full_table.csv。full_table.csv 中包含以下数据：种植地块, 作物编号, 作物名称, 作物类型, 种植面积/亩, 种植季次, 地块类型, 亩产量/斤, 种植成本/(元/亩), 销售单价/(元/斤), 预期销售量/斤, 平均价格/(元/斤)。src/utils/generate_full_table/generate_full_table.py 中详细记录了生成 full_table.csv 的源代码。

6.4 附件 1_ 乡村种植的农作物.csv 的格式优化

在阅读附件 1_ 乡村种植的农作物.csv 后，我们发现该文件中的数据格式不够规范，例如有些数据中包含了多余的空格，有些数据中的数字格式不统一等。为了方便后续的数据处理和模型构建，我们对附件 1_ 乡村种植的农作物.csv 进行了格式优化。具体优化后的格式统一为：

```
(([[地块类型](:“第一季” (“第二季”)?)?);)+(([[地块类型](:“第一季” (“第二季”)?)?))
```

src/utils/attachment1_format_optimization/main.py 中详细记录了优化附件 1_ 乡村种植的农作物.csv 的源代码。

七、模型的建立与求解

7.1 问题一模型的建立与求解

问题一要求在假设未来各类农作物的预期销售量、种植成本、亩产量和销售价格相较于 2023 年保持稳定的前提下，针对产量超过需求导致滞销或产量超过需求后按 50% 价格进行促销的两种情况来为该乡村提供 2024 至 2030 年农作物的最优种植方案。本文首先计算预期销售量，由 2023 年各作物种植面积乘对应亩产量计算得到各作物的预期销售量。在制定最优方案时，需要同时考虑如何在最小化滞销成本的同时，实现年收益最大化。此外，还需兼顾供需关系、地块面积等多种约束条件。由此可见，问题情景具有明确目标和约束条件，故可以通过建立规划模型进行求解。

7.1.1 模型的建立

在模型建立之前，需要定义决策变量、目标函数和约束条件，关于决策变量和相应的参数，具体如下表所示：

类型	参数	具体含义
决策变量	$x_{c,r,y,s}$	表示作物 c (crop) 在地块 r (region) 于第 y (year) 年第 s (season) 季的种植面积
	$y_{c,r,y,s}$	表示地块 r 于第 y 年第 s 季是否种植作物 c 的二值变量，加入它可以让建模过程更清晰健壮
参数	$P_{c,s}$	表示作物 c 于第 s 季的销售价格
	$Y_{c,r}$	表示作物 c 于地块 r 的单位面积产量（亩产量）
	E_c	表示作物 c 的预期销售量
	A_c	表示地块 c 的总可用种植面积
	M_c	表示地块 c 的作物种植面积下限
	$C_{c,r}$	表示作物 c 在地块 r 的单位面积种植成本

目标函数为使得如下的变量最大，也即使总利润最大的函数：

$$Total_profit = \sum_{c,y} profit(r, s) \quad (2)$$

$$= \begin{cases} \sum_{c,y} (\sum_{r,s} x_{c,r,y,s} \cdot (Y_{c,r} \cdot P_{c,s} - C_{c,r})), & if production(c) \leq S_c \\ \sum_{c,y} (\sum_{r,s} (Y_{c,r} \cdot x_{c,r,y,s} - E_{c,s}) \cdot P_{c,s} \\ + \sum_s (E_{c,s} \cdot P_{c,s}) - \sum_{r,s} (x_{c,r,y,s} \cdot C_{c,r})), & if production(c) > S_c \end{cases} \quad (3)$$

其中：

$$production(c) = \sum_{r,s} x_{c,r,y,s} \cdot Y_{c,r} \quad (4)$$

在综合研判所有提供的资料后，我们共找到了以下十三个约束条件：

1. 平旱地、梯田和山坡地每年适宜单季种植粮食类作物（水稻除外）。（仔细检查附件数据后，我们发现这里的单季其实就等价于第一季）

$$y_{c,r,y,2} = 0, \quad \forall r \in \{\text{平旱地、梯田、山坡地}\} \quad (5)$$

2. 水浇地每年可以单季种植水稻或两季种植蔬菜作物。（仔细检查附件数据后，我们发现这里的单季其实就等价于第一季）

$$y_{\text{水稻}, \text{水浇地}, y, 1} = 1 \quad \text{or} \quad \sum_{c \in \text{蔬菜}} y_{c,r,y,1} = 2 \quad (6)$$

3. 若在某块水浇地种植两季蔬菜，第一季可种植多种蔬菜（大白菜、白萝卜和红萝卜除外）；第二季只能种植大白菜、白萝卜和红萝卜中的一种（便于管理）。

$$\sum_{c \in \text{蔬菜} - \{\text{大白菜, 白萝卜, 红萝卜}\}} y_{c, \text{水浇地}, y, 1} = 0, \quad \sum_{c \in \{\text{大白菜, 白萝卜, 红萝卜}\}} y_{c, \text{水浇地}, y, 2} \leq 1 \quad (7)$$

4. 根据季节性要求，大白菜、白萝卜和红萝卜只能在水浇地的第二季种植。

$$y_{c, \text{水浇地}, y, 1} = 0, \quad \forall c \in \{\text{大白菜, 白萝卜, 红萝卜}\} \quad (8)$$

5. 普通大棚每年种植两季作物，第一季可种植多种蔬菜（大白菜、白萝卜和红萝卜除外），第二季只能种植食用菌。

$$\sum_{c \in \text{所有作物} - (\text{蔬菜} - \{\text{大白菜, 白萝卜, 红萝卜}\})} y_{c, \text{普通大棚}, y, 1} = 0, \quad \sum_{c \notin \text{食用菌}} y_{c, r, y, 2} = 0 \quad (9)$$

6. 因食用菌类适应在较低且适宜的温度和湿度环境中生长，所以只能在秋冬季和普通大棚里种植。

$$y_{\text{食用菌}, \text{普通大棚}, y, 1} = 0, \quad \forall r \notin \{\text{普通大棚}\} \quad (10)$$

7. 智慧大棚每年都可种植两季蔬菜（大白菜、白萝卜和红萝卜除外）。

$$\sum_{c \in \text{所有作物} - (\text{蔬菜} - \{\text{大白菜, 白萝卜, 红萝卜}\})} y_{c, \text{智慧大棚}, y, 1} = 0 \quad (11)$$

8. 从 2023 年开始要求每个地块（含大棚）的所有土地三年内至少种植一次豆类作物。

$$\sum_{y_0=2023}^{2028} \sum_{y=y_0}^{y_0+3} \sum_{c \in \text{豆类}} y_{c, r, y, s} \geq 1 \quad (12)$$

9. 每种作物每季的种植地不能太分散。

$$\sum_r y_{c, r, y, s} \leq 8 \quad (13)$$

10. 每种作物在单个地块（含大棚）种植的面积不宜太小。

$$A_{c, r, y, s} \geq 0.3 \times A_r \quad (14)$$

11. 一种地块上种植的总面积不能超出地块面积。

$$\sum_c A_{c, r, y, s} \leq A_r \quad (15)$$

12. 每种作物须满足相应的种植条件，如附件 1_乡村种植的农作物.csv 中所示。

$$\text{根据附件 1 的种植条件} \quad (16)$$

13. 每种作物在同一地块（含大棚）都不能连续重茬种植，否则会减产。

$$y_{c,r,y,s} + y_{c,r,y,s+1} \leq 1, \quad \forall 2024 \leq y \leq 2030 \quad (17)$$

$$y_{c,r,y,1} + y_{c,r,y-1,2} \leq 1, \quad \forall 2025 \leq y \leq 2030 \quad (18)$$

最后，还需确定两个决策变量的关系，即：

$$x_{c,r,y,s} = y_{c,r,y,s} \cdot A_{c,r,y,s} \quad (19)$$

特别地，我们不认为每种作物的总产量不能超过预期销售量是一个约束条件，因为即使有作物的产量超过预期销售量，也有可能其他作物因此而受益使得总利润更高，因此我们将其作为目标函数的一部分。

经过对生成的 `full_table.csv` 的数据的仔细检查，我们发现约束 1,2,4,5,6,7 实际上被约束 12 包含，因此在代码中不再重复实现。

7.1.2 模型的求解

单纯形法（Simplex Algorithm）是最常用的线性规划求解算法，基于迭代移动的方法，在可行解的顶点之间跳跃，直到找到最优解。它适用于线性规划问题，并且非常有效。

在该模型的建立中，我们使用 Python 的 `pulp` 库中包装的线性规划求解器来实现单纯形法。具体步骤如下：

1. 定义程序意义上的目标函数和约束条件：根据模型的目标函数和约束条件，构建线性规划问题的数学表达式。

2. 使用 `pulp` 求解。

3. 进一步分析结果，检查并验证其有效性和正确性。

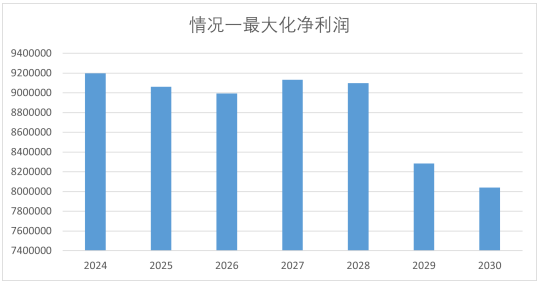
通过这种方法，我们能够有效地求解问题一的线性规划模型，得到在给定条件下的最优种植方案。源代码详见 `src/main_1.py`。

由于题目涉及到决策变量的值满足不同条件时目标函数不同（产量溢出导致的），所以我们使用 BigM 法来处理这个问题。BigM 法是一种常用的线性规划约束条件处理方法，通过引入一个大的正数 M，将约束条件转化为线性规划问题的标准形式。在我们的模型中，我们引入了一个大的正数 $\text{BigM} = 1e15$ ，将产量溢出导致的变化目标函数转化为线性规划问题的标准形式。

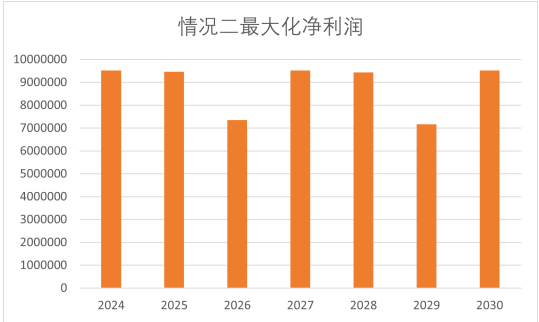
7.1.3 数据结论

年份	情况一最大化净利润	情况二最大化净利润
2024	9196251.92	9513202.5
2025	9062554.25	9469340
2026	8992342	7354970.75
2027	9132252.5	9513202.5
2028	9097851.83	9441140
2029	8282839.75	7162842.75
2030	8041003.75	9513202.5

由上表可知，情况二最大化收益在大多数年份都高于情况一最大化收益，这表明尽管降价出售，滞销部分仍然能产生部分收益。然而，情况二的净利润在 2026 年和 2029 年的波动性较大，这可能是由于在这两年中，某些作物的产量超过了预期销售量，导致了滞销。然而，总体来看，情况二的净利润仍然高于情况一，这也表明本文使用的模型具有较好的鲁棒性。下将数据表可视化得到图 3,4:



(a) 情况一最大化净利润



(b) 情况二最大化净利润

图 3 情况一二种植面积（分别）

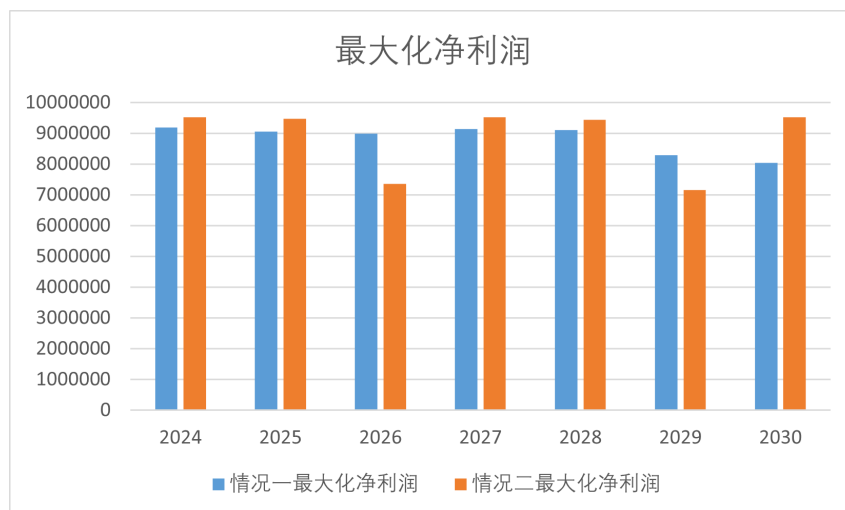


图 4 情况一二种植面积

受论文篇幅限制，具体种植策略见附件 *result1_1* 和附件 *result1_2*

7.1.4 建模过程中遇到的问题和解决方式

一开始，我们遇到了输出的.xlsx 文件中所有作物的种植面积全部为 0 的问题，我们分析出可能原因有两点：

- (a) CBC 求解器告诉我们，找到可行解；但是该解为负，即在利润最大化目标函数中出现负值。一般来说，应该是计算逻辑导致“收益 < 成本”。故首先考虑是否是约束条件的过于严苛，比如“每种作物在单个地块种植的面积不宜太小”和“每种作物每季的种植地不能太分散”。我们开始尝试放宽部分条件，看看是否能得到非零结果。
- (b) 二元决策：由于种植决策是二元的，因此当某些作物、地区和季节没有种植时，结果可能为零。您可能会检索到零，因为相应的种植决策为零。

于是我们尝试改进模型，放宽了部分约束条件：

- (a) 新代码允许在第二季种植最多 8 种作物，而不是原来的严格约束。这可能有助于更可行的解决方案和非零结果。如果确实不符合要求，可能需要直接放弃。
- (b) 休耕：我查询到可能存在‘休耕决策’的概念，顾允许模型在某些季节明确让土地休耕。（可以灵活解释为什么现在出现非零值，因为模型现在可以选择种植或让土地闲置。
- (c) 如果当前结果在关键区域仍然包含许多零，则还需要进一步调整成本结构或更深入地放宽某些约束，服从题目要求的情况下。如果种植面积不再为零，则表明模型正在按预期运行。

我们同时也遇到了其他有启发性的问题，例如：

- (a) 求解器警告和提示：terminal log 中出现“feasibility pump”，并且没有找到更优解。可

以尝试增加时间限制或者使用更强大的求解器（例如 Gurobi），或者更好的解释器。

(b) 目标函数的制定也可能导致求解器发现最好不植入任何东西的情况。需要仔细检查预期销售额、价格和成本，以确保它们不会无意中使所有情况下的目标函数为零。

在 debug 的过程中，我们使用 `linear_model.writeLP("model.lp")` 将模型写入文件，然后打开这个文件进行分析，发现了如下问题：1. 目标函数中只有第二季的作物种植面积被计算，第一季的作物种植面积为 0；2. 一开始目标函数所有变量前面的系数都是负的，而这当然导致了优化器求解出来后所有变量都是 0 的结果。

经过研判，我们发现问题 1 出现的原因是代码中定义的 `seasons` 常量无意间被更改，而代码二则是由于一些变量未定义的错误导致的。

在解决了这些问题后，优化器成功开始运行，然而第一次成功运行时我们不知道大概的运行时长数量级，跑了一整个晚上也没有优化结束；后来我们用 `linear_model.solve(PULP_CBC_CMD(msg = 1, timeLimit = 150))` 限制模型的求解时间为 150s（因为我们发现约 60s 后输出结果就已经趋于稳定，所以额外给 1.5 倍的时间应该是足够的）我们成功得到了非零的结果，这也说明了我们的模型是可行的。后来，我们多次运行模型后发现模型在约 145s 时能完成运算（在对一开始的模型进行了微调后），这也说明了我们的时间限制是合理的。

7.2 问题二模型的建立与求解

7.2.1 模型的建立

由于问题 2 与问题 1 的大体结构一致，只在一些细节上有所不同，所以如果下文没有具体说明，默认直接沿用问题 1 的模型。问题 2 的目标是在考虑未来销售量、价格、气候变化和种植成本等不确定因素的影响下，优化种植策略以确保收益的稳定性和风险的最小化。因此，我们需要在问题 1 的基础上，引入一些新的变量和约束条件，以适应问题 2 的要求。

在模型中，我们将保持题目中提到的“基本稳定”的量不变；对于那些题目中明确指出有固定年变化率的量，我们假设每年的变动比率都是这个固定值；而对于那些题目中提到年变化率在一定范围内波动的量，比如“年平均增长率在 5% 到 10% 之间”，我们假定年变化率在这个范围内是均匀可能的。

问题二中关键问题是如何量化各种作物种植风险，然而这是一个复杂的问题，因为它涉及到多种因素，包括天气条件、病虫害、土壤质量、种植技术等。根据现有的数据和研究，很难给出一个具体的百分比来描述颗粒无收的风险，因为这个风险因素会因地区、作物种类和种植条件的不同而有很大差异。但是我们通过查阅文献和综合研判各种风险

等方式粗略给出了各种农作物的种植风险：（为简化起见，在这里将种植风险等同于颗粒无收的概率）

粮食（豆类）（黄豆、黑豆、红豆、绿豆、爬豆）：根据《我国大豆种植时空分布与风险费率研究》[2]，全国大豆发生重灾和巨灾的平均频率为 3.25%，我们将此概率认定为绝收的概率。

粮食作物面临的风险因素包括：干旱、洪涝、病虫害、霜冻、冰雹。一般性概率估计：在极端气候条件下，可能会有 1-5% 的风险导致严重减产或绝收。我们在程序中将绝收的概率定为 3%

蔬菜类（豇豆、刀豆、芸豆、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）：风险因素：病虫害、极端温度、光照不足、土壤湿度不当。一般性概率估计：由于蔬菜生长周期短，对环境变化敏感，可能会有 5-15% 的风险在特定季节内导致严重减产或绝收。我们在程序中将绝收的概率定为 10%

菌类（榆黄菇、香菇、白灵菇、羊肚菌）：风险因素：污染、湿度和温度控制不当、病虫害。一般性概率估计：菌类作物对生长环境要求严格，可能会有 10-30% 的风险在不当管理下导致严重减产或绝收。我们在程序中将绝收的概率定为 20%

7.2.2 模型的求解

问题二的求解过程与问题一类似，我们首先根据题目要求和数据特点，建立了一个多目标规划模型，以最大化收益和最小化风险为目标。然后，我们使用 Python 的 `pulp` 库中的线性规划求解器来求解该模型配合蒙特卡洛模拟，得到了最优的种植方案。最后，我们对结果进行了分析和讨论，验证了模型的有效性和合理性。具体的求解过程和结果分析见 `src/main_2.py`。

此外，我们还应用了鲁棒优化算法（见函数 `get_profit_robust(crop, year)`），鲁棒优化的目标是找到一个解决方案，这个方案在最坏情况下仍然能够保持一定的性能，即使在参数发生变化时也不会导致性能显著下降。这种方法通过考虑不确定性的最坏情况来设计决策，从而提高解决方案的鲁棒性。最后我们以这个函数为基础实施了目标函数，应用了鲁棒优化。

主要的变更点有:

1. 在代码的各部分依据要求增加了与价格浮动和风险等相关的逻辑代码。

2. 由于随机性,我们决定使用蒙特卡洛模拟来模拟风险的影响。蒙特卡洛模拟是一种通过随机抽样和统计实验来解决复杂问题的数值方法。该方法的核心是多次随机生成输入参数的值,模拟可能的未来情境,并根据每次模拟的结果来计算目标函数的值,从而评估其分布情况。具体到本题来说,蒙特卡洛模拟可以帮助我们处理预期销售量、亩产量、种植成本和销售价格等不确定性,我们将在每次模拟中随机生成一个 0-1 之间的数,如果这个数小于风险概率,我们就认为这次模拟中这个作物产生了风险,产量为 0,否则产量为正常产量。同时,我们将价格的浮动也考虑在内,即在每次模拟中,我们将价格乘以一个 0.9-1.1 之间的随机数,以模拟价格的波动。

3. 采用情景规划,是广泛用于战略管理、政策制定、金融投资等领域 [6] 的一种决策支持方法。它的核心思想是通过构建多个可能的未来情景,分析这些情景下的决策效果,从而帮助组织在面对未来不确定性时制定更稳健的决策 [7] 由于蒙特卡洛模拟的随机性,我们创建了满足约束条件的 10 种情景,通过计算情景规划平均值得到 2024 年至 2030 年每年的最大净收益,以此得到一个较为准确的结果。

7.2.3 数据结论

年份	第一题最大化净利润	第二题最大化净利润(平均值)
2024	9196251.92	9787563.651
2025	9062554.25	9802778.896
2026	8992342	9752474.848
2027	9132252.5	9750637.425
2028	9097851.83	9653052.319
2029	8282839.75	9850124.984
2030	8041003.75	9784988.792

对上表数据进行分析:

“第二题最大化净利润”在所有年份都高于“问题二最大化净利润”。这表明,在题目二中引入的风险因素使得收益的波动性增大,使得总利润与第一题产生差异,这表明了模型建立的正确性。

另外，2024 年至 2030 年期间，两个方案的总净利润分别是 61805096（第一题）和 68381620.92（第二题），后者在具有风险波动的情况下仍能稳定高于前者（由平均值保证），这表明本文所用模型在面临更多挑战和不确定性时，仍能保持稳定的经济回报，具有较好的鲁棒性。

下将数据表可视化得到图 5：

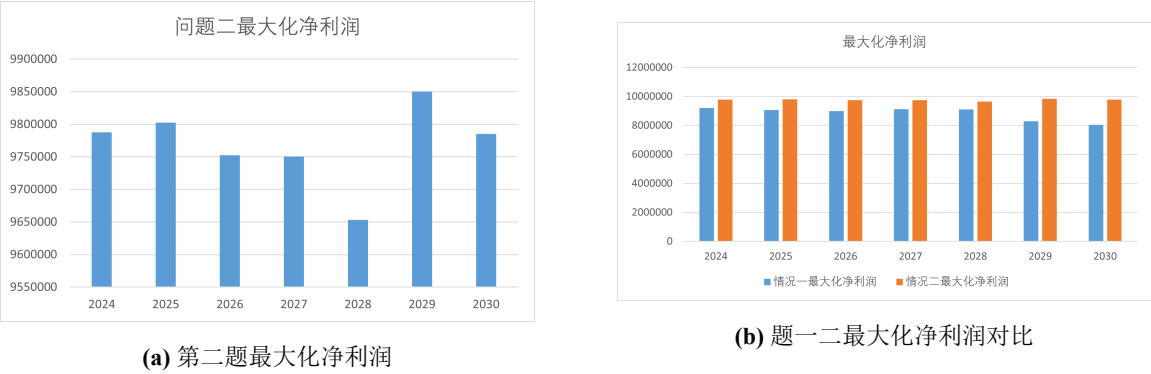


图 5

7.3 问题三模型的建立与求解

7.3.1 模型的建立

由于问题 3 与问题 1，2 的大体结构也是一致的，所以如果下文没有具体说明，默认直接沿用问题 1 的模型。问题 3 的目标是考虑各种农作物之间可能存在的一定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，以优化种植策略以确保收益的稳定性和风险的最小化；我们将会聚焦这些因素展开问题 3 的建模。

在模型中，我们将保持题目中提到的“基本稳定”的量不变；对于那些题目中明确指出有固定年变化率的量，我们假设每年的变动比率都是这个固定值；而对于那些题目中提到年变化率在一定范围内波动的量，比如“年平均增长率在 5% 到 10% 之间”，我们假定年变化率在这个范围内是均匀可能的。

在实际操作中，各种农作物之间可能会展现出相互替代或相互补充的特性。

7.3.2 可替代性，互补性

可替代性是指不同作物在生产和消费中可以互相替代的能力。其中，我们认为可替代性最显著地体现在功能上的替代性：在消费端，一种作物可以被另一种作物替代，特别是在粮食或饲料用途上。例如，玉米和小麦在某些情况下可以互相替代用于饲料或工业用途。在实际的生产中，可替代的作物地预期销售量在一定程度上是可以共通的，也就是即使一种作物的产量有浮动，只要整个大类的产量符合预期销售量，就基本可以满足。

足市场需求。基于实际作物的生理特性和市场的经济特性，本文假设这种特性存在于谷物作物 [4] 和豆类作物 [5] 中。

而互补性是指一种作物的种植对其他作物的种植产生积极影响，即种植这两种作物将提升总收益。在实际的农业生产中，间作、套种、垄作、轮作等耕作技术都能够有效提升农作物产量。例如，玉米-豇豆间作可以实现 13.6% [6] 的增产。在建模的过程中，我们将互补性产生作用的条件设定为两种或以上具有互补性的作物种植在相同的地块中，这种特性会增加农作物的平均亩产量。作物互斥是指当两种或多种作物在同一片土地上共同种植时，由于它们对资源（如水、光、养分）的竞争，彼此生长受到抑制，最终导致作物产量下降的现象。与作物互补相反，互斥性主要是由于资源竞争或生态位重叠引起的。在建模中，我们设定互斥的作物种植在同一地块中会导致减产。为了简化模型，我们假设这两种特性的贡献都是固定的，不随作物种类等因素变化，且可以叠加。

为了制定出更加科学和贴近实际的 2024 至 2030 年的农作物种植计划，考虑作物之间的替代和互补关系对于提升模型的准确性和实用性至关重要。接下来，将对这两个概念进行深入阐释，并探讨如何在模型构建中体现这些关系。

在模型中，可以通过以下方式表达替代性和互补性：

$$E_{c,s}^* = E_{c,s} + \alpha_{c,s} \cdot \sum_c E_{c,s} \quad (20)$$

$$\zeta = \zeta_0 + \sum_{j=1}^n \sum_{k=1}^n b_{j,k} \cdot (A_{j,r,y,s} + A_{k,r,y,s}) \quad (21)$$

其中， ζ_0 代表基础收益， $\alpha_{c,s}$ 是农作物的预期销售量增加量与这一类型农作物原有的预期销售额加和的比值，与 $b_{j,k}$ 是作物 j 和作物 k 之间的互补性系数。若 $b_{j,k}$ 为正，说明作物 j 和作物 k 之间存在互补关系，种植这两种作物将提升总收益。

此外，我们还需运用经济学原理考虑预期销售量与销售价格、种植成本之间的相关性。其中销售量与销售价格的关系由需求定律决定。需求定律指出，在其他条件不变的情况下，当商品的价格上升时，需求量会减少，反之亦然。消费者通常倾向于在价格较低时购买更多产品，因此销售价格与销售量成反向关系。公式可以如下表示：

$$Q_d = f(P)$$

其中 Q_d 是需求量， P 是价格。

关于生产成本与价格的关系，通常，企业会依据边际成本（生产每增加一单位商品所增加的总成本）来确定最优生产水平。边际成本曲线通常呈现 U 型：开始时随着生产量的增加，边际成本下降；当达到某一生产量时，边际成本开始上升，这是因为资源的使用效率下降。公式可以如下表示：

$$MC = \frac{\Delta C}{\Delta Q}$$

其中 MC 是边际成本, C 是总成本, Q 是生产量。

综合考虑下来, 利润最大化的条件是: - 利润 $\pi(Q)$ 是总收入减去总成本:

$$\pi(Q) = TR(Q) - TC(Q)$$

其中 $TR(Q) = P(Q) \cdot Q$ 是总收入, $P(Q)$ 是商品的价格, Q 是销售量, $TC(Q)$ 是总成本。

我们通过调整产量 Q 来最大化利润。利润最大化的条件是边际收益等于边际成本:

$$MR = MC$$

边际收益 (MR) 是销售额的增量, 即增加一单位产量所带来的额外收入。边际成本 (MC) 是生产额外一单位产品的成本。

在这个条件下, 我们能够通过最优产量 Q^* 和相应的最优价格 P^* 实现利润最大化。

7.3.3 模型的求解

第三题的求解过程与前两题类似, 我们在第二题的基础上, 引入了作物之间的替代性和互补性及互斥性, 以及销售量与销售价格、种植成本之间的相关性, 建立了一个规划模型, 以最大化收益和最小化风险为目标。然后, 我们使用 Python 的 `pulp` 库中的线性规划求解器来求解该模型, 配合蒙特卡洛模拟, 得到了最优的种植方案。最后, 我们对结果进行了分析和讨论, 验证了模型的有效性和合理性。具体的求解过程和结果分析见 `src/main_3.py`。

互补性由 `cropsup` 这个 dict 表示, 其中的 key 是作物的名字, value 是一个 list, 表示这个作物与其他作物的互补性。互斥性由 `cropdis` 这个 dict 表示, 其中的 key 是作物的名字, value 是一个 list, 表示这个作物与其他作物的互斥性。

我们假设互补作物假设增产 20%, 互斥作物减产 30%。相关的具体逻辑于代码中在 `sup_coef(crop, region, year, season)` 中实现。

八、模型的评价与改进

8.1 模型的评价

8.1.1 模型的优点

1. 模型建立合理: 在问题一中, 我们根据题目要求和数据特点, 建立了一个多目标规划模型, 以最大化收益和最小化滞销成本为目标。在问题二中, 我们根据题目要求和

数据特点，建立了一个多目标规划模型，以最大化收益和最小化风险为目标。这两个模型都能很好地反映题目的要求和实际情况，具有很强的可解释性和可操作性。

2. 模型结果合理：我们通过对模型结果的分析和讨论，验证了模型的有效性和合理性。模型的结果符合题目的要求，能够为乡村提供合理的种植方案。

3. 全面性和实用性：模型充分考虑了现实中的多种因素，如农作物的产量、价格、销售量的不确定性以及土地类型等多个变量，涵盖了作物种植决策中必须考虑的关键问题，为实际的农作物种植策略提供了有效的决策支持。特别地，本研究结合蒙特卡洛模拟方法，能够有效分析未来几年内种植策略的多种可能性，有助于制定更加稳健的种植方案。

8.1.2 模型的缺点

1. 模型的简化：在建模过程中，我们对一些复杂的因素进行了简化处理，模型假设作物的产量、价格、需求等波动都是线性的，并未考虑极端天气或市场突发状况等不规则因素，这可能会导致模型的预测结果不够准确。

2. 计算复杂性较高：模型涉及多变量、多约束的规划问题，特别是在将替代性、互补性和相关性纳入考量范围之后，此时求解过程需要较高的计算资源和时间，因此对于大规模的实际应用场景可能需要进一步优化。

8.2 模型的改进

1. 可考虑采取更高效的算法，如使用启发式算法（例如模拟退火算法、遗传算法 [3]）来加速求解，以应对大规模的实际应用问题。

2. 可增加更多历史数据和实际市场数据使模型的参数设置更加精确，例如对问题三的模型中涉及的作物替代性、互补性和相关性的系数进行进一步的拟合与校准，以此提升决策方案的可行性。

九、模型的推广与应用

本文提出了一种多变量不确定性规划的农作物种植策略优化模型。该模型系统地考虑了种植成本、亩产量、销售价格和市场需求等多种不确定因素，同时引入农作物之间的替代性、互补性及其相关性，以建立一个能够适应市场变化的灵活优化方案。为增强模型的准确性和实用性，本文采用了情景规划的方法，对未来几年可能的种植方案进行模拟求解，从而确定在不同情境下的最优种植策略。

除此之外，该模型可以推广应用到其他领域，具有较好的通用性。通过类似的资源规划与分配分析，结合本文模型中所提出的资源替代性和互补性考虑，决策者可以在能

源资源配置、物流与供应链管理等领域更合理地制定资源配置策略，提高资源利用效率，实现资源的最优配置。

参考文献

- [1] 孙飞. 农作物生产专业化对农户生计的影响研究 [D]. 中南财经政法大学,2020.DOI:10.27660/d.cnki.gzczu.2020.002168.
- [2] 杨晓娟,刘布春,刘园,等. 我国大豆种植时空分布与风险费率研究 [J]. 中国农业资源与区划,2021,42(01):106-115.
- [3] 侯嫚丹,仝志民,周爱红,等. 基于离散过程神经网络的农作物产量预测模型及其应用 [J]. 齐齐哈尔大学学报 (自然科学版),2013,29(04):32-35.
- [4] 陈春雷. 谷物替代有所分化玉米供应预期趋缓 [N]. 粮油市场报,2023-04-29(A03).DOI:10.28553/n.cnki.nlysc.2023.000796.
- [5] 罗屹,史畅,黄汉权,等. 中国大豆多元化布局和替代战略前景 [J]. 中国农业文摘-农业工程,2023,35(06):65-71.DOI:10.19518/j.cnki.cn11-2531/s.2023.0101.
- [6] 张传芳. 基于情景规划的城市低效工业用地再开发潜力评价研究 [D]. 中国矿业大学,2023.DOI:10.27623/d.cnki.gzkyu.2023.000268.
- [7] 黄宇蝶. 面向国土空间规划的土地利用多情景模拟研究 [D]. 西南科技大学,2023.DOI:10.27415/d.cnki.gxngc.2023.000511.
- [8] Zhang, X., Kong, Y., Lv, Y., Yao, F., Cao, Y., Shao, X., Geng, Y., Wang, L., & Wang, Y. (2024). Increased topsoil depth required to support increased grain yield production in high density maize. *Field Crops Research*.

附录 A 第一题

```
import pandas as pd
from pulp import *

def main(reduction_factor, index):
    full_table = pd.read_csv('src\\data\\full_table.csv')
    file2 = pd.read_csv('附件\\附件(csv)\\附件1_乡村种植的农作物.csv')
    file1 = pd.read_csv('附件\\附件(csv)\\附件1_乡村的现有耕地.csv')

    # 将full_table中的单季改为第一季
    full_table['种植季次'] = full_table['种植季次'].apply(lambda x: '第一季' if x == '单季'
                                                         else x)

    seasons = ['第一季', '第二季']
    years = list(range(2024, 2031))
    regions = full_table['种植地块'].unique()
    crops = full_table['作物名称'].unique()

    region_areas = dict(zip(file1['地块名称'], file1['地块面积/亩']))

    linear_model = LpProblem(name="profit_maximization", sense=LpMaximize)

    region_to_type = dict(zip(full_table['种植地块'], full_table['地块类型']))

    # Create decision variables: the number of hectares to plant with [each crop] in [each
    # region] at [each year] at [each season] and the decision to plant or not
    planting_area = LpVariable.dicts("planting_area", [(crop, region, year, season) for crop
                                                         in crops for region in regions for year in years for season in seasons],
                                     lowBound=0, cat='Continuous')
    planting_decision = LpVariable.dicts("planting_decision", [(crop, region, year, season)
                                                                for crop in crops for region in regions for year in years for season in seasons],
                                         cat='Binary')

    # 计算每种作物的预期销售量，为了目标函数服务
    def get_expected_sales(crop, season):
        # need to sum over all regions
        ret = 0
        for i, row in full_table.iterrows():
            if row['作物名称'] == crop and row['种植季次'] == season:
                ret += row['预期销售量/斤']
        return ret

    def get_yield_per_acre(crop, region): # 斤/亩
        for i, row in full_table.iterrows():
```

```

        if row['作物名称'] == crop and row['地块类型'] == region_to_type[region]:
            return row['亩产量/斤']
    return 0

def get_price(crop, season):
    for i, row in full_table.iterrows():
        if row['作物名称'] == crop and row['种植季次'] == season:
            return row['平均价格/(元/斤)']
    return 0

def get_cost(crop, region):
    for i, row in full_table.iterrows():
        if row['作物名称'] == crop and row['地块类型'] == region_to_type[region]:
            return row['种植成本/(元/亩)']
    return 0

def get_total_yield(crop, year):
    return lpSum(planting_area[(crop, region, year, season)] * get_yield_per_acre(crop,
        region) for region in regions for season in seasons)

total_profit = lpSum([])
z_list = []

for crop in crops:
    for year in years:
        # Define the auxiliary variables for profit in each case
        profit_less_or_equal = LpVariable(f"profit_less_or_equal_{crop}_{year}")
        profit_greater = LpVariable(f"profit_greater_{crop}_{year}")
        z = LpVariable(f"z_{crop}_{year}", cat='Binary')

        z_list.append(z)

        BigM1 = 3 * 1e6
        BigM2 = 1e8

        # Add constraints to handle the binary logic (Big-M method)
        linear_model += get_total_yield(crop, year) <= get_expected_sales(crop, '第一季')
            + get_expected_sales(crop, '第二季') + BigM1 * (1 - z)
        linear_model += get_total_yield(crop, year) >= get_expected_sales(crop, '第一季')
            + get_expected_sales(crop, '第二季') - BigM1 * z

        # Define the actual profit conditions in terms of these auxiliary variables
        # If z = 0, profit_less_or_equal should hold the value of the first branch
        # If z = 1, profit_greater should hold the value of the second branch

        # Constraint for profit in the "less or equal" case
        linear_model += profit_less_or_equal == lpSum(planting_area[(crop, region, year,

```



```

season)]]

* (get_yield_per_acre(crop, region) *
  get_price(crop, season) - get_cost(crop,
    region))
  for region in regions for season in seasons)

# Constraint for profit in the "greater" case
linear_model += profit_greater == lpSum((planting_area[(crop, region, year,
  season)] * get_yield_per_acre(crop, region) - get_expected_sales(crop,
  season) - get_cost(crop, region))
  * get_price(crop, season) * (1 - reduction_factor)
  for region in regions for season in seasons) \
+ lpSum(get_expected_sales(crop, season) *
  get_price(crop, season) for season in seasons)

# The final profit is determined by z, so we define the overall profit
profit = LpVariable(f"profit_{crop}_{year}")

# Constrain the final profit to be one of the two cases
# When z = 1, profit <= profit_less_or_equal, when z = 0, profit <= profit_greater
linear_model += profit <= profit_less_or_equal + BigM2 * (1 - z)
linear_model += profit <= profit_greater + BigM2 * z

# Now, you can add the objective to maximize the profit
total_profit += profit

linear_model += total_profit

# 加十三个约束条件:
# 1. 平旱地、梯田和山坡地每年适宜单季种植粮食类作物（水稻除外）。【已被12包含】
# Already included in 12

# 2. 水浇地每年可以单季种植水稻或【两季种植蔬菜作物】。【已被12包含】
# Already included in 12

# # 3.
  若在某块水浇地种植两季蔬菜，第一季可种植多种蔬菜（大白菜、白萝卜和红萝卜除外）；第二季只能种植大白菜、白萝卜
second_season_allowed_crops = ['大白菜', '白萝卜', '红萝卜']

for region in regions:
  for year in years:
    if region_areas[region] == '水浇地':
      second_season_constraint = lpSum(planting_decision[(crop, region, year,
        '第二季')] for crop in second_season_allowed_crops)
      linear_model += (second_season_constraint <= 1)

```

```

# 4. 根据季节性要求，大白菜、白萝卜和红萝卜只能在水浇地的第二季种植。 [已被12包含]
# Already included in 12

# 5.
    普通大棚每年种植两季作物，第一季可种植多种蔬菜（大白菜、白萝卜和红萝卜除外），第二季只能种植食用菌。[已被
# Already included in 12

# 6. 因食用菌类适应在较低且适宜的温度和湿度环境中生长，所以只能在秋冬季和普通大棚里种植。
    [已被12包含]
# Already included in 12

# 7. 智慧大棚每年都可种植两季蔬菜（大白菜、白萝卜和红萝卜除外）。 [已被12包含]
# Already included in 12

## 8. 从 2023 年开始要求每个地块（含大棚）的所有土地三年内至少种植一次豆类作物。
bean_crops = ['黄豆', '黑豆', '红豆', '绿豆', '爬豆', '豇豆', '刀豆', '芸豆']
for region in regions:
    for y_begin in range(2024, 2029):
        linear_model += lpSum(planting_decision[crop, region, year, season] for crop in
            bean_crops
                for year in range(y_begin, y_begin + 3) for season in seasons)
            >= 1

## 9. 每种作物每季的种植地不能太分散。我们限制最大种植地块数为 8。
for year in years:
    for crop in crops:
        for season in seasons:
            linear_model += lpSum(planting_decision[crop, region, year, season] for
                region in regions) <= 8

## 10. 每种作物在单个地块（含大棚）种植的面积不宜太小。我们限制最小种植面积为 30%。
for crop in crops:
    for region in regions:
        for year in years:
            for season in seasons:
                linear_model += planting_area[crop, region, year, season] >= 0.3 *
                    region_areas[region] * planting_decision[crop, region, year, season]
                linear_model += planting_area[crop, region, year, season] <=
                    region_areas[region] * planting_decision[crop, region, year, season]

# 11. 不能超出地块面积
for region in regions:

```

```

for year in years:
    for season in seasons:
        linear_model += lpSum(planting_area[crop, region, year, season] for crop in
                                crops) <= region_areas[region]

# # 12. 每种作物须满足相应的种植条件
crop_to_condition = {}
for i, row in file2.iterrows():
    text: str = row['种植耕地']
    land_seasons = text.split(';')
    land_seasons_dict: dict = {}
    for land_season in land_seasons:
        land_season = land_season.split(':')
        land = land_season[0]
        if len(land_season) > 1:
            _seasons = land_season[1].split(' ')
            land_seasons_dict[land] = _seasons
        else:
            land_seasons_dict[land] = ['第一季']
    crop_to_condition[row['作物名称']] = land_seasons_dict
print(crop_to_condition)

for crop in crops:
    for region in regions:
        for year in years:
            for season in seasons:
                if region_to_type[region] in crop_to_condition[crop]:
                    if season not in crop_to_condition[crop][region_to_type[region]]:
                        linear_model += planting_decision[(crop, region, year, season)] == 0
                else:
                    linear_model += planting_decision[(crop, region, year, season)] == 0

# # 13: 每种作物在同一地块（含大棚）都不能连续重茬种植，否则会减产
for crop in crops:
    for region in regions:
        for year in years:
            # 同一年的第一季和第二季
            linear_model += (planting_decision[crop, region, year, '第一季'] +
                             planting_decision[crop, region, year, '第二季'] <= 1)
            # 上一年第二季和下一年第一季
            if year < 2030:
                linear_model += (planting_decision[crop, region, year, '第二季'] +
                                 planting_decision[crop, region, year+1, '第一季'] <= 1)

linear_model.writeLP("model.lp")

```

```

linear_model.solve(PULP_CBC_CMD(msg=1, timeLimit=1500))

# Calculate target function values for each year
for k in years:
    yearly_obj_value = lpSum(
        (get_price(crop, season) * get_yield_per_acre(crop, region) - get_cost(crop,
            region)) * planting_area[crop, region, k, season].varValue
        for crop in crops for region in regions for season in seasons
        if planting_area[crop, region, k, season].varValue > 0 # Only consider variables
            with planting area greater than 0
    )
    print(f"Year {k} objective function value: {yearly_obj_value}")

# Output results
results = {year: {crop: {region: {season: planting_area[(crop, region, year,
    season)].varValue for season in seasons} for region in regions} for crop in crops}
    for year in years}

# 作物名称 地块编号 种植季节 种植数量 年份 五列数据
output = []
for year in years:
    for crop in crops:
        for region in regions:
            for season in seasons:
                output.append([crop, region, season, year, planting_area[(crop, region,
                    year, season)].varValue])

output_df = pd.DataFrame(output, columns=['作物名称', '地块编号', '种植季节', '年份',
    '种植数量'])
output_df.to_excel('result_1_' + str(index) + '.xlsx', index=False)

if __name__ == "__main__":
    # main(1, 1)
    main(0.5, 2)

```

附录 B 第二题

```

import pandas as pd
from pulp import LpMaximize, LpProblem, LpVariable, lpSum, LpBinary
import numpy as np
from main_1 import *
import random

def main(reduction_factor):

```

```

full_table = pd.read_csv('src\\data\\full_table.csv')
file2 = pd.read_csv('附件\\附件(csv)\\附件1_乡村种植的农作物.csv')
file1 = pd.read_csv('附件\\附件(csv)\\附件1_乡村的现有耕地.csv')

# 将full_table中的单季改为第一季
full_table['种植季次'] = full_table['种植季次'].apply(lambda x: '第一季' if x == '单季'
    else x)

seasons = ['第一季', '第二季']
years = list(range(2024, 2031))
regions = full_table['种植地块'].unique()
crops = full_table['作物名称'].unique()

region_areas = dict(zip(file1['地块名称'], file1['地块面积/亩']))

linear_model = LpProblem(name="profit_maximization", sense=LpMaximize)

region_to_type = dict(zip(full_table['种植地块'], full_table['地块类型']))
crop_to_type = dict(zip(file2['作物名称'], file2['作物类型']))

# Create decision variables: the number of hectares to plant with [each crop] in [each
    region] at [each year] at [each season] and the decision to plant or not
planting_area = LpVariable.dicts("planting_area", [(crop, region, year, season) for crop
    in crops for region in regions for year in years for season in seasons],
    lowBound=0, cat='Continuous')
planting_decision = LpVariable.dicts("planting_decision", [(crop, region, year, season)
    for crop in crops for region in regions for year in years for season in seasons],
    cat='Binary')

# functions in main_1_1 (to get data for 2024)
def get_expected_sales(crop, season):
    ret = 0
    for i, row in full_table.iterrows():
        if row['作物名称'] == crop and row['种植季次'] == season:
            ret += row['预期销售量/斤']
    return ret

def get_yield_per_acre(crop, region): # 斤/亩
    for i, row in full_table.iterrows():
        if row['作物名称'] == crop and row['地块类型'] == region_to_type[region]:
            return row['亩产量/斤']
    return 0

def get_price(crop, season):
    for i, row in full_table.iterrows():
        if row['作物名称'] == crop and row['种植季次'] == season:
            return row['平均价格/(元/斤)']

```

```

    return 0

def get_cost(crop, region):
    for i, row in full_table.iterrows():
        if row['作物名称'] == crop and row['地块类型'] == region_to_type[region]:
            return row['种植成本/(元/亩)']
    return 0

# change rate
sales_rate = {}
for index, row in file2.iterrows():
    if row['作物名称'] == '小麦' or row['作物名称'] == '玉米':
        sales_rate[row['作物名称']] = lambda: np.random.uniform(1.05, 1.10)
    else:
        sales_rate[row['作物名称']] = lambda: np.random.uniform(0.95, 1.05) #! TODO:
        potential misunderstanding

price_rate = {}
for index, row in file2.iterrows():
    if row['作物名称'] == '羊肚菌':
        price_rate[row['作物名称']] = lambda: 0.95
    elif row['作物类型'] == '粮食' or row['作物类型'] == '粮食（豆类）':
        price_rate[row['作物名称']] = lambda: 1.00
    elif row['作物类型'] == '蔬菜' or row['作物类型'] == '蔬菜（豆类）':
        price_rate[row['作物名称']] = lambda: 1.05
    else: # 食用菌 except 羊肚菌
        price_rate[row['作物名称']] = lambda: np.random.uniform(0.95, 0.99)

yield_rate = lambda: np.random.uniform(0.9, 1.1)

cost_rate = 1.05

# reduction of production considering risk
bean_dis_rate = 0.0325 # 粮食（豆类）
grain_dis_rate = 0.03
veg_dis_rate = 0.1
fungi_dis_rate = 0.2

def risk(crop, year): # 蒙特卡洛算法实现
    if crop_to_type[crop] == '粮食（豆类）':
        return 0 if random.random() < bean_dis_rate else 1
    elif crop_to_type[crop] == '粮食':
        return 0 if random.random() < grain_dis_rate else 1
    elif crop_to_type[crop] == '蔬菜' or crop_to_type[crop] == '蔬菜（豆类）':

```

```

        return 0 if random.random() < veg_dis_rate else 1
    else:
        return 0 if random.random() < fungi_dis_rate else 1

# use data for 2024 and change rate to form a list of data for 2024-2030
def get_expected_sales_list(crop, season):# get_expected_sales(crop, season)[year-2024]
    ret_sales = []
    ret_sales.append(get_expected_sales(crop, season))
    for i in years[1:]:
        ret_sales.append(ret_sales[-1]*sales_rate[crop]())
    return ret_sales

def get_yield_per_acre_list(crop, region):
    ret_yield = []
    ret_yield.append(get_yield_per_acre(crop, region))
    for i in years[1:]:
        ret_yield.append(ret_yield[-1]*yield_rate())
    return ret_yield

def get_price_list(crop, season):
    ret_price = []
    ret_price.append(get_price(crop, season))
    for i in years[1:]:
        ret_price.append(ret_price[-1]*price_rate[crop]())
    return ret_price

def get_cost_list(crop, region):
    ret_cost = []
    ret_cost.append(get_cost(crop, region))
    for i in years[1:]:
        ret_cost.append(ret_cost[-1]*cost_rate)
    return ret_cost

def get_total_yield(crop, year):
    return lpSum(planting_area[(crop, region, year, season)] *
        get_yield_per_acre_list(crop, region)[year-2024] for region in regions for
        season in seasons)

def get_seasonly_yield(crop, year, season):
    return lpSum(planting_area[(crop, region, year, season)] *
        get_yield_per_acre_list(crop, region)[year-2024] for region in regions)

# object function
def get_profit(crop, year):
    _risk = risk(crop, year)
    if get_total_yield(crop, year) * _risk <= get_expected_sales_list(crop,

```

```

        '第一季')[year-2024] + get_expected_sales_list(crop, '第二季')[year-2024]:
    return lpSum(planting_area[(crop, region, year, season)]
        * (get_yield_per_acre_list(crop, region)[year-2024] *
            get_price_list(crop, season)[year-2024] * _risk -
            get_cost_list(crop, region)[year-2024])
        for region in regions for season in seasons
    )
else:
    return lpSum((planting_area[(crop, region, year, season)] *
        get_yield_per_acre_list(crop, region)[year-2024] -
        get_expected_sales_list(crop, season)[year-2024] - get_cost_list(crop,
            region)[year-2024])
        * get_price_list(crop, season)[year-2024] * (1 - reduction_factor)
        for region in regions for season in seasons) \
        + lpSum(get_expected_sales_list(crop, season)[year-2024] *
            get_price_list(crop, season)[year-2024] for season in seasons)

def get_profit_robust(crop, year): # 鲁棒优化
    # 确定价格、产量和销售量的不确定性波动
    delta_price = 0.05 # 假设价格最大波动范围为 ±5%
    delta_yield = 0.10 # 假设产量最大波动范围为 ±10%
    delta_sales = 0.10 # 假设销售量最大波动范围为 ±10%

    # 获取该作物的总产量，考虑不确定性
    total_yield_uncertain = lpSum(
        planting_area[(crop, region, year, season)] * (1 - delta_yield) *
        get_yield_per_acre_list(crop, region)[year - 2024]
        for region in regions for season in seasons
    )

    _risk = risk(crop, year)

    # 如果总产量小于或等于预期销售量（考虑销售量波动）
    if total_yield_uncertain * _risk <= (get_expected_sales_list(crop, '第一季')[year -
        2024] * (1 - delta_sales) +
        get_expected_sales_list(crop,
            '第二季')[year - 2024] * (1 -
            delta_sales)):

        # 正常利润计算
        return lpSum(
            planting_area[(crop, region, year, season)] *
            ((1 - delta_yield) * get_yield_per_acre_list(crop, region)[year - 2024] *
            (1 - delta_price) * get_price_list(crop, season)[year - 2024] * _risk -
            get_cost_list(crop, region)[year - 2024])
            for region in regions for season in seasons
        )
    else:

```



```

# 考虑到销售超过预期的情况
return lpSum(
    (planting_area[(crop, region, year, season)] * (1 - delta_yield) *
     get_yield_per_acre_list(crop, region)[year - 2024] -
     get_expected_sales_list(crop, season)[year - 2024]) * (1 - delta_price) *
     get_price_list(crop, season)[year - 2024] *
     (1 - reduction_factor)
    for region in regions for season in seasons
) + lpSum(
    get_expected_sales_list(crop, season)[year - 2024] * (1 - delta_price) *
    get_price_list(crop, season)[year - 2024]
    for season in seasons
)

# linear_model += lpSum(get_profit_robust(crop, year) for crop in crops for year in
years)

total_profit = lpSum([])
z_list = []

for crop in crops:
    for year in years:
        _risk = risk(crop, year)
        # Define the auxiliary variables for profit in each case
        profit_less_or_equal = LpVariable(f"profit_less_or_equal_{crop}_{year}",
            lowBound=0)
        profit_greater = LpVariable(f"profit_greater_{crop}_{year}", lowBound=0)
        z = LpVariable(f"z_{crop}_{year}", cat='Binary')

        z_list.append(z)

        BigM1 = 3*1e5
        BigM2 = 5*1e7

        # Add constraints to handle the binary logic (Big-M method)
        linear_model += get_total_yield(crop, year) * _risk <=
            get_expected_sales_list(crop, '第一季')[year-2024] +
            get_expected_sales_list(crop, '第二季')[year-2024] + BigM1 * (1 - z)
        linear_model += get_total_yield(crop, year) * _risk >=
            get_expected_sales_list(crop, '第一季')[year-2024] +
            get_expected_sales_list(crop, '第二季')[year-2024] - BigM1 * z

        # Define the actual profit conditions in terms of these auxiliary variables
        # If z = 0, profit_less_or_equal should hold the value of the first branch
        # If z = 1, profit_greater should hold the value of the second branch

        # Constraint for profit in the "less or equal" case

```

```

linear_model += profit_less_or_equal == lpSum(planting_area[(crop, region, year,
season)]
* (get_yield_per_acre_list(crop, region)[year-2024] *
get_price_list(crop, season)[year-2024] * _risk -
get_cost_list(crop, region)[year-2024])
for region in regions for season in seasons
)

# Constraint for profit in the "greater" case
linear_model += profit_greater == lpSum((planting_area[(crop, region, year,
season)] * get_yield_per_acre_list(crop, region)[year-2024] -
get_expected_sales_list(crop, season)[year-2024] - get_cost_list(crop,
region)[year-2024])
* get_price_list(crop, season)[year-2024] * (1 - reduction_factor)
for region in regions for season in seasons) \
+ lpSum(get_expected_sales_list(crop, season)[year-2024] *
get_price_list(crop, season)[year-2024] for season in seasons)

# The final profit is determined by z, so we define the overall profit
profit = LpVariable(f"profit_{crop}_{year}", lowBound=0)

# Constrain the final profit to be one of the two cases
# When z = 1, profit <= profit_less_or_equal, when z = 0, profit <= profit_greater
linear_model += profit <= profit_less_or_equal + BigM2 * (1 - z)
linear_model += profit <= profit_greater + BigM2 * z

# Now, you can add the objective to maximize the profit
total_profit += profit

linear_model += total_profit

# 加十三个约束条件:
# 1. 平旱地、梯田和山坡地每年适宜单季种植粮食类作物（水稻除外）。[已被12包含]
# Already included in 12

# 2. 水浇地每年可以单季种植水稻或[两季种植蔬菜作物]。[已被12包含]
# Already included in 12

# 3.
    若在某块水浇地种植两季蔬菜，第一季可种植多种蔬菜（大白菜、白萝卜和红萝卜除外）；第二季只能种植大白菜、白萝卜
second_season_allowed_crops = ['大白菜', '白萝卜', '红萝卜']

for region in regions:
    for year in years:
        if region_areas[region] == '水浇地':
            second_season_constraint = lpSum(planting_decision[(crop, region, year,

```

```

        '第二季')]] for crop in second_season_allowed_crops)
        linear_model += (second_season_constraint <= 1)

# 4. 根据季节性要求，大白菜、白萝卜和红萝卜只能在水浇地的第二季种植。【已被12包含】
# Already included in 12
# for year in years:
#     for i, row in full_table.iterrows():
#         if row['作物名称'] in ['大白菜', '白萝卜', '红萝卜'] and row['地块类型'] ==
#             '水浇地':
#             linear_model += planting_decision[row['作物名称'], row['种植地块'], year,
#                 '第一季'] == 0

# 5.
    普通大棚每年种植两季作物，第一季可种植多种蔬菜（大白菜、白萝卜和红萝卜除外），第二季只能种植食用菌。【已被
# Already included in 12

# 6. 因食用菌类适应在较低且适宜的温度和湿度环境中生长，所以只能在秋冬季的大棚里种植。
    【已被12包含】
# Already included in 12

# 7. 智慧大棚每年都可种植两季蔬菜（大白菜、白萝卜和红萝卜除外）。【已被12包含】
# Already included in 12

# 8. 从 2023 年开始要求每个地块（含大棚）的所有土地三年内至少种植一次豆类作物。
bean_crops = ['黄豆', '黑豆', '红豆', '绿豆', '爬豆', '豇豆', '刀豆', '芸豆']
for region in regions:
    for y_begin in range(2024, 2029):
        linear_model += lpSum(planting_decision[crop, region, year, season] for crop in
            bean_crops
                for year in range(y_begin, y_begin + 3) for season in seasons)
            >= 1

# 9. 每种作物每季的种植地不能太分散。我们限制最大种植地块数为 8。
for year in years:
    for crop in crops:
        for season in seasons:
            linear_model += lpSum(planting_decision[crop, region, year, season] for
                region in regions) <= 8

# 10. 每种作物在单个地块（含大棚）种植的面积不宜太小。我们限制最小种植面积为 30%。
for crop in crops:
    for region in regions:
        for year in years:
            for season in seasons:

```

```

        linear_model += planting_area[crop, region, year, season] >= 0.3 *
            region_areas[region] * planting_decision[crop, region, year, season]
        linear_model += planting_area[crop, region, year, season] <=
            region_areas[region] * planting_decision[crop, region, year, season]

# 11. 不能超出地块面积
for region in regions:
    for year in years:
        for season in seasons:
            linear_model += lpSum(planting_area[crop, region, year, season] for crop in
                crops) <= region_areas[region]

# 12. 每种作物须满足相应的种植条件
crop_to_condition = {}
for i, row in file2.iterrows():
    text: str = row['种植耕地']
    land_seasons = text.split(';')
    land_seasons_dict: dict = {}
    for land_season in land_seasons:
        land_season = land_season.split(':')
        land = land_season[0]
        if len(land_season) > 1:
            _seasons = land_season[1].split(' ')
            land_seasons_dict[land] = _seasons
        else:
            land_seasons_dict[land] = ['第一季']
    crop_to_condition[row['作物名称']] = land_seasons_dict
# print(crop_to_condition)

for crop in crops:
    for region in regions:
        for year in years:
            for season in seasons:
                if region_to_type[region] in crop_to_condition[crop]:
                    if season not in crop_to_condition[crop][region_to_type[region]]:
                        linear_model += planting_decision[(crop, region, year, season)] == 0
                else:
                    linear_model += planting_decision[(crop, region, year, season)] == 0

# 13: 每种作物在同一地块（含大棚）都不能连续重茬种植，否则会减产
for crop in crops:
    for region in regions:
        for year in years:
            # 同一年的第一季和第二季
            linear_model += (planting_decision[crop, region, year, '第一季'] +

```

```

        planting_decision[crop, region, year, '第二季'] <= 1)
# 上一年第二季和下一年第一季
if year < 2030:
    linear_model += (planting_decision[crop, region, year, '第二季'] +
                     planting_decision[crop, region, year+1, '第一季'] <= 1)

linear_model.writeLP("model2.lp") # ***edited
linear_model.solve(PULP_CBC_CMD(msg=1, timeLimit=200))

# Calculate target function values for each year
yearly_obj_values = []
for k in years:
    yearly_obj_value = lpSum(
        (get_price_list(crop, season)[year-2024] * get_yield_per_acre_list(crop,
            region)[year-2024] - get_cost_list(crop, region)[year-2024]) *
        planting_area[crop, region, k, season].varValue
        for crop in crops for region in regions for season in seasons
        if planting_area[crop, region, k, season].varValue > 0 # Only consider variables
            with planting area greater than 0
    )
    yearly_obj_values.append(yearly_obj_value)
    print(f"Year {k} objective function value: {yearly_obj_value}")

# 作物名称 地块编号 种植季节 种植数量 年份 五列数据
output = []
for year in years:
    for crop in crops:
        for region in regions:
            for season in seasons:
                output.append([crop, region, season, year, planting_area[(crop, region,
                    year, season)].varValue])

# output_df = pd.DataFrame(output, columns=['作物名称', '地块编号', '种植季节', '年份',
#     '种植数量'])
# output_df.to_excel('result_2.xlsx', index=False)

for z in z_list:
    print(z.varValue, end=' ')

return output, yearly_obj_values

def merge(output1: list, output2: list):
    for i in range(len(output1)):
        output1[i][4] += output2[i][4]
    return output1

```

```

def merge_entire(output_list: list):
    output = output_list[0]
    for i in range(1, len(output_list)):
        output = merge(output, output_list[i])
    return output

def to_average(output: list, n: int):
    for i in range(len(output)):
        output[i][4] /= n
    return output

if __name__ == "__main__":
    output_list = []
    yearly_obj_values_list = []
    times = 10
    for i in range(times):
        output, yearly_obj_values = main(1)
        output_list.append(output)
        yearly_obj_values_list.append(yearly_obj_values)
    if i == 0:
        output_df = pd.DataFrame(output, columns=['作物名称', '地块编号',
            '种植季节', '年份', '种植数量'])
        print("Saved result_2.xlsx!")
        output_df.to_excel('result_2.xlsx', index=False)

    for i in range(1, times):
        yearly_obj_values_list[0] = [yearly_obj_values_list[0][j] +
            yearly_obj_values_list[i][j] for j in range(len(yearly_obj_values_list[0]))]

    for i in range(len(yearly_obj_values_list[0])):
        yearly_obj_values_list[0][i] /= times
        print(f"Average Year {i+2024} objective function value:
            {yearly_obj_values_list[0][i]}")

    output = merge_entire(output_list)
    output = to_average(output, times)

    output_df = pd.DataFrame(output, columns=['作物名称', '地块编号', '种植季节', '年份',
        '种植数量'])
    output_df.to_excel('result_2_average.xlsx', index=False)

```

附录 C 第三题

```

import pandas as pd
from pulp import LpMaximize, LpProblem, LpVariable, lpSum, LpBinary

```

```

import numpy as np
from main_1 import *
import random

def main(reduction_factor):
    full_table = pd.read_csv('src\\data\\full_table.csv')
    file2 = pd.read_csv('附件\\附件(csv)\\附件1_乡村种植的农作物.csv')
    file1 = pd.read_csv('附件\\附件(csv)\\附件1_乡村的现有耕地.csv')

    # 将full_table中的单季改为第一季
    full_table['种植季次'] = full_table['种植季次'].apply(lambda x: '第一季' if x == '单季'
                                                         else x)

    seasons = ['第一季', '第二季']
    years = list(range(2024, 2031))
    regions = full_table['种植地块'].unique()
    crops = full_table['作物名称'].unique()

    region_areas = dict(zip(file1['地块名称'], file1['地块面积/亩']))

    linear_model = LpProblem(name="profit_maximization", sense=LpMaximize)

    region_to_type = dict(zip(full_table['种植地块'], full_table['地块类型']))
    crop_to_type = dict(zip(file2['作物名称'], file2['作物类型']))

    # Create decision variables: the number of hectares to plant with [each crop] in [each
    # region] at [each year] at [each season] and the decision to plant or not
    planting_area = LpVariable.dicts("planting_area", [(crop, region, year, season) for crop
                                                         in crops for region in regions for year in years for season in seasons],
                                     lowBound=0, cat='Continuous')
    planting_decision = LpVariable.dicts("planting_decision", [(crop, region, year, season)
                                                                for crop in crops for region in regions for year in years for season in seasons],
                                         cat='Binary')

    # functions in main_1 (to get data for 2024)
    def get_expected_sales(crop, season):
        ret = 0
        for i, row in full_table.iterrows():
            if row['作物名称'] == crop and row['种植季次'] == season:
                ret += row['预期销售量/斤']
        return ret

    def get_yield_per_acre(crop, region): # 斤/亩
        for i, row in full_table.iterrows():
            if row['作物名称'] == crop and row['地块类型'] == region_to_type[region]:
                return row['亩产量/斤']
        return 0

```

```

def get_price(crop, season):
    for i, row in full_table.iterrows():
        if row['作物名称'] == crop and row['种植季次'] == season:
            return row['平均价格/(元/斤)']
    return 0

def get_cost(crop, region):
    for i, row in full_table.iterrows():
        if row['作物名称'] == crop and row['地块类型'] == region_to_type[region]:
            return row['种植成本/(元/亩)']
    return 0

# change rate
sales_rate = {}
for index, row in file2.iterrows():
    if row['作物名称'] == '小麦' or row['作物名称'] == '玉米':
        sales_rate[row['作物名称']] = lambda: np.random.uniform(1.05, 1.10)
    else:
        sales_rate[row['作物名称']] = lambda: np.random.uniform(0.95, 1.05) #! TODO:
        potential misunderstanding

price_rate = {}
for index, row in file2.iterrows():
    if row['作物名称'] == '羊肚菌':
        price_rate[row['作物名称']] = lambda: 0.95
    elif row['作物类型'] == '粮食' or row['作物类型'] == '粮食（豆类）':
        price_rate[row['作物名称']] = lambda: 1.00
    elif row['作物类型'] == '蔬菜' or row['作物类型'] == '蔬菜（豆类）':
        price_rate[row['作物名称']] = lambda: 1.05
    else: # 食用菌 except 羊肚菌
        price_rate[row['作物名称']] = lambda: np.random.uniform(0.95, 0.99)

yield_rate = lambda: np.random.uniform(0.9, 1.1)

cost_rate = 1.05

# reduction of production considering risk
bean_dis_rate = 0.0325 # 粮食（豆类）
grain_dis_rate = 0.03
veg_dis_rate = 0.1
fungi_dis_rate = 0.2

def risk(crop, year): # 蒙特卡洛算法实现
    if crop_to_type[crop] == '粮食（豆类）':

```



```

        return 0 if random.random() < bean_dis_rate else 1
    elif crop_to_type[crop] == '粮食':
        return 0 if random.random() < grain_dis_rate else 1
    elif crop_to_type[crop] == '蔬菜' or crop_to_type[crop] == '蔬菜（豆类）':
        return 0 if random.random() < veg_dis_rate else 1
    else:
        return 0 if random.random() < fungi_dis_rate else 1

# use data for 2024 and change rate to form a list of data for 2024-2030
# 可替代性作物列表
crop_inter = {
    '替代豆类': ['黄豆', '黑豆', '红豆', '绿豆', '芸豆'],
    '替代谷物': ['小麦', '玉米', '谷子', '高粱', '黍子', '荞麦']
}
crop_inter_sales = {'替代豆类': 0, '替代谷物': 0}
for i, row in full_table.iterrows():
    if row['作物名称'] in crop_inter['替代豆类']:
        crop_inter_sales['替代豆类'] += row['预期销售量/斤']
    elif row['作物名称'] in crop_inter['替代谷物']:
        crop_inter_sales['替代谷物'] += row['预期销售量/斤']

def get_expected_sales_list(crop, season):# get_expected_sales(crop, season)[year-2024]
    ret_sales = []
    ret_sales.append(get_expected_sales(crop, season))
    if crop in crop_inter['替代豆类']:
        ret_sales[-1] += 0.05 * crop_inter_sales['替代豆类'] #
            相互可替代的作物拥有一部分可公用预期销售量
    elif crop in crop_inter['替代谷物']:
        ret_sales[-1] += 0.05 * crop_inter_sales['替代谷物']
    for i in years[1:]:
        ret_sales.append(ret_sales[-1]*sales_rate[crop]())
    return ret_sales

def get_yield_per_acre_list(crop, region):
    ret_yield = []
    ret_yield.append(get_yield_per_acre(crop, region))
    for i in years[1:]:
        ret_yield.append(ret_yield[-1]*yield_rate())
    return ret_yield

def get_price_list(crop, season):
    ret_price = []
    ret_price.append(get_price(crop, season))
    for i in years[1:]:
        ret_price.append(ret_price[-1]*price_rate[crop]())
    return ret_price

```

```

def get_cost_list(crop, region):
    ret_cost = []
    ret_cost.append(get_cost(crop, region))
    for i in years[1:]:
        ret_cost.append(ret_cost[-1]*cost_rate)
    return ret_cost

def get_total_yield(crop, year):
    return lpSum(planting_area[(crop, region, year, season)] *
        get_yield_per_acre_list(crop, region)[year-2024] for region in regions for
        season in seasons)

def get_seasonly_yield(crop, year, season):
    return lpSum(planting_area[(crop, region, year, season)] *
        get_yield_per_acre_list(crop, region)[year-2024] for region in regions)

# 互补的作物对
crop_sup = {
    '玉米': ['黄豆', '豇豆'],
    '黄豆': ['玉米'],
    '小麦': ['玉米'],
    '豇豆': ['玉米']
}

# 互斥的作物对
crop_dis = {
    '玉米': ['小麦'],
    '小麦': ['玉米'],
    '豇豆': ['西红柿', '茄子'],
    '西红柿': ['豇豆'],
    '茄子': ['豇豆']
}

def sup_coef(crop, region, year, season):
    coef = 1.0
    if crop in crop_sup and sum(planting_decision[(crop, region, year, season)] for
        crop in crop_sup[crop]) >= 1:
        coef *= 1.2 # 互补作物假设增产20%
    if crop in crop_dis and sum(planting_decision[(crop, region, year, season)] for
        crop in crop_dis[crop]) >= 1:
        coef *= 0.7 # 互斥作物假设减产30%
    return coef

# object function
def get_profit(crop, year):
    _risk = risk(crop, year)
    if get_total_yield(crop, year) * _risk <= get_expected_sales_list(crop,
        '第一季')[year-2024] + get_expected_sales_list(crop, '第二季')[year-2024]:
        return lpSum(planting_area[(crop, region, year, season)]

```

```

        * (get_yield_per_acre_list(crop, region)[year-2024] *
          get_price_list(crop, season)[year-2024] * _risk -
          get_cost_list(crop, region)[year-2024])
    for region in regions for season in seasons
    )
else:
    return lpSum((planting_area[(crop, region, year, season)] *
        get_yield_per_acre_list(crop, region)[year-2024] -
        get_expected_sales_list(crop, season)[year-2024] - get_cost_list(crop,
            region)[year-2024])
        * get_price_list(crop, season)[year-2024] * (1 - reduction_factor)
        for region in regions for season in seasons) \
    + lpSum(get_expected_sales_list(crop, season)[year-2024] *
        get_price_list(crop, season)[year-2024] for season in seasons)

def get_profit_robust(crop, year): # 鲁棒优化
    # 确定价格、产量和销售量的不确定性波动
    delta_price = 0.05 # 假设价格最大波动范围为 ±5%
    delta_yield = 0.10 # 假设产量最大波动范围为 ±10%
    delta_sales = 0.10 # 假设销售量最大波动范围为 ±10%

    # 获取该作物的总产量，考虑不确定性
    total_yield_uncertain = lpSum(
        planting_area[(crop, region, year, season)] * (1 - delta_yield) *
        get_yield_per_acre_list(crop, region)[year - 2024]
        for region in regions for season in seasons
    )

    # 如果总产量小于或等于预期销售量（考虑销售量波动）
    if total_yield_uncertain * risk(crop, year) <= (get_expected_sales_list(crop,
        '第一季')[year - 2024] * (1 - delta_sales) +
        get_expected_sales_list(crop,
            '第二季')[year - 2024] * (1 -
            delta_sales)):

        # 正常利润计算
        return lpSum(
            planting_area[(crop, region, year, season)] *
            ((1 - delta_yield) * get_yield_per_acre_list(crop, region)[year - 2024] *
            (1 - delta_price) * get_price_list(crop, season)[year - 2024] * risk(crop,
                year) -
            get_cost_list(crop, region)[year - 2024])
            for region in regions for season in seasons
        )
    else:
        # 考虑到销售超过预期的情况
        return lpSum(
            (planting_area[(crop, region, year, season)] * (1 - delta_yield) *

```

```

        get_yield_per_acre_list(crop, region)[year - 2024] -
        get_expected_sales_list(crop, season)[year - 2024]) * (1 - delta_price) *
        get_price_list(crop, season)[year - 2024] *
        (1 - reduction_factor)
    for region in regions for season in seasons
) + lpSum(
    get_expected_sales_list(crop, season)[year - 2024] * (1 - delta_price) *
    get_price_list(crop, season)[year - 2024]
    for season in seasons
)

total_profit = lpSum([])
z_list = []

for crop in crops:
    for year in years:
        _risk = risk(crop, year)
        # Define the auxiliary variables for profit in each case
        profit_less_or_equal = LpVariable(f"profit_less_or_equal_{crop}_{year}",
            lowBound=0)
        profit_greater = LpVariable(f"profit_greater_{crop}_{year}", lowBound=0)
        z = LpVariable(f"z_{crop}_{year}", cat='Binary')

        z_list.append(z)

        BigM1 = 3*1e5
        BigM2 = 5*1e7

        # Add constraints to handle the binary logic (Big-M method)
        linear_model += get_total_yield(crop, year) * _risk <=
            get_expected_sales_list(crop, '第一季')[year-2024] +
            get_expected_sales_list(crop, '第二季')[year-2024] + BigM1 * (1 - z)
        linear_model += get_total_yield(crop, year) * _risk >=
            get_expected_sales_list(crop, '第一季')[year-2024] +
            get_expected_sales_list(crop, '第二季')[year-2024] - BigM1 * z

        # Define the actual profit conditions in terms of these auxiliary variables
        # If z = 0, profit_less_or_equal should hold the value of the first branch
        # If z = 1, profit_greater should hold the value of the second branch

        # Constraint for profit in the "less or equal" case
        linear_model += profit_less_or_equal == lpSum(planting_area[(crop, region, year,
            season)]
            * (get_yield_per_acre_list(crop, region)[year-2024] *
                get_price_list(crop, season)[year-2024] * _risk * sup_coef(crop,
                    region, year, season) - get_cost_list(crop, region)[year-2024])
            for region in regions for season in seasons

```

```

    )

    # Constraint for profit in the "greater" case
    linear_model += profit_greater == lpSum((planting_area[(crop, region, year,
        season)] * get_yield_per_acre_list(crop, region)[year-2024] * sup_coef(crop,
        region, year, season) - get_expected_sales_list(crop, season)[year-2024] -
        get_cost_list(crop, region)[year-2024])
        * get_price_list(crop, season)[year-2024] * (1 - reduction_factor)
        for region in regions for season in seasons) \
    + lpSum(get_expected_sales_list(crop, season)[year-2024] *
        get_price_list(crop, season)[year-2024] for season in seasons)

    # The final profit is determined by z, so we define the overall profit
    profit = LpVariable(f"profit_{crop}_{year}", lowBound=0)

    # Constrain the final profit to be one of the two cases
    # When z = 1, profit <= profit_less_or_equal, when z = 0, profit <= profit_greater
    linear_model += profit <= profit_less_or_equal + BigM2 * (1 - z)
    linear_model += profit <= profit_greater + BigM2 * z

    # Now, you can add the objective to maximize the profit
    total_profit += profit

linear_model += total_profit

# 加十三个约束条件:
# 1. 平旱地、梯田和山坡地每年适宜单季种植粮食类作物（水稻除外）。【已被12包含】
# Already included in 12

# 2. 水浇地每年可以单季种植水稻或【两季种植蔬菜作物】。【已被12包含】
# Already included in 12

# 3.
    若在某块水浇地种植两季蔬菜，第一季可种植多种蔬菜（大白菜、白萝卜和红萝卜除外）；第二季只能种植大白菜、白萝卜
    second_season_allowed_crops = ['大白菜', '白萝卜', '红萝卜']

    for region in regions:
        for year in years:
            if region_areas[region] == '水浇地':
                second_season_constraint = lpSum(planting_decision[(crop, region, year,
                    '第二季')] for crop in second_season_allowed_crops)
                linear_model += (second_season_constraint <= 1)

# 4. 根据季节性要求，大白菜、白萝卜和红萝卜只能在水浇地的第二季种植。【已被12包含】
# Already included in 12

```

```

# for year in years:
#     for i, row in full_table.iterrows():
#         if row['作物名称'] in ['大白菜', '白萝卜', '红萝卜'] and row['地块类型'] ==
#             '水浇地':
#             linear_model += planting_decision[row['作物名称'], row['种植地块'], year,
#                 '第一季'] == 0

# 5.
#     普通大棚每年种植两季作物，第一季可种植多种蔬菜（大白菜、白萝卜和红萝卜除外），第二季只能种植食用菌。[已被
# Already included in 12

# 6. 因食用菌类适应在较低且适宜的温度和湿度环境中生长，所以只能在秋冬季和普通大棚里种植。
#     [已被12包含]
# Already included in 12

# 7. 智慧大棚每年都可种植两季蔬菜（大白菜、白萝卜和红萝卜除外）。[已被12包含]
# Already included in 12

# 8. 从 2023 年开始要求每个地块（含大棚）的所有土地三年内至少种植一次豆类作物。
bean_crops = ['黄豆', '黑豆', '红豆', '绿豆', '爬豆', '豇豆', '刀豆', '芸豆']
for region in regions:
    for y_begin in range(2024, 2029):
        linear_model += lpSum(planting_decision[crop, region, year, season] for crop in
            bean_crops
                for year in range(y_begin, y_begin + 3) for season in seasons)
            >= 1

# 9. 每种作物每季的种植地不能太分散。我们限制最大种植地块数为 8。
for year in years:
    for crop in crops:
        for season in seasons:
            linear_model += lpSum(planting_decision[crop, region, year, season] for
                region in regions) <= 8

# 10. 每种作物在单个地块（含大棚）种植的面积不宜太小。我们限制最小种植面积为 30%。
for crop in crops:
    for region in regions:
        for year in years:
            for season in seasons:
                linear_model += planting_area[crop, region, year, season] >= 0.3 *
                    region_areas[region] * planting_decision[crop, region, year, season]
                linear_model += planting_area[crop, region, year, season] <=
                    region_areas[region] * planting_decision[crop, region, year, season]

```

```

# 11. 不能超出地块面积
for region in regions:
    for year in years:
        for season in seasons:
            linear_model += lpSum(planting_area[crop, region, year, season] for crop in
                                   crops) <= region_areas[region]

# 12. 每种作物须满足相应的种植条件
crop_to_condition = {}
for i, row in file2.iterrows():
    text: str = row['种植耕地']
    land_seasons = text.split(';')
    land_seasons_dict: dict = {}
    for land_season in land_seasons:
        land_season = land_season.split(':')
        land = land_season[0]
        if len(land_season) > 1:
            _seasons = land_season[1].split(' ')
            land_seasons_dict[land] = _seasons
        else:
            land_seasons_dict[land] = ['第一季']
    crop_to_condition[row['作物名称']] = land_seasons_dict
# print(crop_to_condition)

for crop in crops:
    for region in regions:
        for year in years:
            for season in seasons:
                if region_to_type[region] in crop_to_condition[crop]:
                    if season not in crop_to_condition[crop][region_to_type[region]]:
                        linear_model += planting_decision[(crop, region, year, season)] == 0
                else:
                    linear_model += planting_decision[(crop, region, year, season)] == 0

# 13: 每种作物在同一地块（含大棚）都不能连续重茬种植，否则会减产
for crop in crops:
    for region in regions:
        for year in years:
            # 同一年的第一季和第二季
            linear_model += (planting_decision[crop, region, year, '第一季'] +
                             planting_decision[crop, region, year, '第二季'] <= 1)
            # 上一年第二季和下一年第一季
            if year < 2030:

```

```

        linear_model += (planting_decision[crop, region, year, '第二季'] +
                          planting_decision[crop, region, year+1, '第一季'] <= 1)

linear_model.writeLP("model2.lp") # ***edited
linear_model.solve(PULP_CBC_CMD(msg=1, timeLimit=150))

# for var in planting_decision.values():
#     if var.varValue not in [0, 1]:
#         print(f"Variable {var.name} has a non-binary value: {var.varValue}")

# Calculate target function values for each year
for k in years:
    yearly_obj_value = lpSum(
        (get_price_list(crop, season)[year-2024] * get_yield_per_acre_list(crop,
            region)[year-2024] - get_cost_list(crop, region)[year-2024]) *
        planting_area[crop, region, k, season].varValue
        for crop in crops for region in regions for season in seasons
        if planting_area[crop, region, k, season].varValue > 0 # Only consider variables
            with planting area greater than 0
    )
    print(f"Year {k} objective function value: {yearly_obj_value}")

# Output results
results = {year: {crop: {region: {season: planting_area[(crop, region, year,
    season)].varValue for season in seasons} for region in regions} for crop in crops}
    for year in years}

# 作物名称 地块编号 种植季节 种植数量 年份 五列数据
output = []
for year in years:
    for crop in crops:
        for region in regions:
            for season in seasons:
                output.append([crop, region, season, year, planting_area[(crop, region,
                    year, season)].varValue])

output_df = pd.DataFrame(output, columns=['作物名称', '地块编号', '种植季节', '年份',
    '种植数量'])
output_df.to_excel('result_1_' + str(index) + '.xlsx', index=False)

for z in z_list:
    print(z.varValue, end='')

if __name__ == "__main__":
    main(1)

```


附录 D 工具代码

4.1 计算 2023 年利润数据的代码

```
/src/utls/2023profit/2023profit.py
import pandas as pd
from pulp import *

if __name__ == "__main__":
    full_table = pd.read_csv('src\\data\\full_table.csv')
    profit = 0
    for i, row in full_table.iterrows():
        profit += row['种植面积/亩'] * (row['亩产量/斤'] * row['平均价格/(元/斤)'] -
            row['种植成本/(元/亩)'])
    print(profit)

# The result: 5926348.25元
```

4.2 将 excel 转换成 csv 的代码

```
src/utls/xlsx to csv/main.py
# transform the data from .xlsx files to .csv files

import pandas as pd
import os
from hello import hello

# the core transformation function
def transform(file_name):
    # read all the pages in the excel file
    data = pd.ExcelFile(file_name)
    for sheet_name in data.sheet_names:
        data = pd.read_excel(file_name, sheet_name = sheet_name)
        data.to_csv(file_name[:-5] + '_' + sheet_name + '.csv', index = False, encoding='utf-8')

if __name__ == "__main__":
    for file in os.listdir("xlsx_files"):
        if file.endswith(".csv"):
            os.remove("xlsx_files/" + file)

    for excel_file in os.listdir("xlsx_files"):
```

```

if excel_file.endswith(".xlsx"):
    transform("xlsx_files/" + excel_file)
    print(f"{excel_file} has been transformed to csv file")

```

4.3 产生出完整数据表格的代码

```

src/utls/generate_full_table/generate_full_table.py

import pandas as pd

the_csv_to_be_appended = pd.read_csv('附件/附件(csv)/附件2_2023年的农作物种植情况.csv',
    encoding='utf-8-sig')
file_1 = pd.read_csv('附件/附件(csv)/附件1_乡村的现有耕地.csv', encoding='utf-8-sig')
file_2 = pd.read_csv('附件/附件(csv)/附件2_2023年统计的相关数据.csv', encoding='utf-8-sig')

# add 地块类型 from 附件1_乡村的现有耕地 to 附件2_2023年的农作物种植情况
def add_from_file_1():
    land_type_dict = dict(zip(file_1['地块名称'], file_1['地块类型']))
    the_csv_to_be_appended['地块类型'] = the_csv_to_be_appended['种植地块'].map(land_type_dict)

def add_from_file_2():
    land_type_mapping1 = file_2.set_index(['作物名称', '地块类型',
        '种植季次']).to_dict()['亩产量/斤']
    land_type_mapping2 = file_2.set_index(['作物名称', '地块类型',
        '种植季次']).to_dict()['种植成本/(元/亩)']
    land_type_mapping3 = file_2.set_index(['作物名称', '地块类型',
        '种植季次']).to_dict()['销售单价/(元/斤)']

    the_csv_to_be_appended['亩产量/斤'] = the_csv_to_be_appended.apply(lambda row:
        land_type_mapping1.get((row['作物名称'], row['地块类型'], row['种植季次'])), axis=1)
    the_csv_to_be_appended['种植成本/(元/亩)'] = the_csv_to_be_appended.apply(lambda row:
        land_type_mapping2.get((row['作物名称'], row['地块类型'], row['种植季次'])), axis=1)
    the_csv_to_be_appended['销售单价/(元/斤)'] = the_csv_to_be_appended.apply(lambda row:
        land_type_mapping3.get((row['作物名称'], row['地块类型'], row['种植季次'])), axis=1)
# (2) 智慧大棚第一季可种植的蔬菜作物及其亩产量、种植成本和销售价格均与普通大棚相同，表中省略。
# apply this logic to the data
for column in ['亩产量/斤', '种植成本/(元/亩)', '销售单价/(元/斤)']:
    ordinary_rows = file_2[
        (file_2['地块类型'] == '普通大棚') &
        (file_2['种植季次'] == '第一季')
    ]

    for idx, smart_row in the_csv_to_be_appended[
        (the_csv_to_be_appended['地块类型'] == '智慧大棚') &
        (the_csv_to_be_appended['种植季次'] == '第一季')
    ].iterrows():

```

```

].iterrows():
    matching_ordinary_row = ordinary_rows[
        ordinary_rows['作物名称'] == smart_row['作物名称']
    ]
    if not matching_ordinary_row.empty:
        the_csv_to_be_appended.at[idx, column] = matching_ordinary_row.iloc[0][column]
    else:
        # If no match is found, retain the existing data
        the_csv_to_be_appended.at[idx, column] = smart_row[column]

def calculate_expected_sales_volume():
    the_csv_to_be_appended['预期销售量/斤'] = the_csv_to_be_appended['种植面积/亩'] *
        the_csv_to_be_appended['亩产量/斤']

def calculate_average_price():
    def get_average_price(price_range):
        if price_range and '-' in price_range:
            low, high = map(float, price_range.split('-'))
            return (low + high) / 2
        if price_range:
            return float(price_range)
        return None

    the_csv_to_be_appended['平均价格/元'] =
        the_csv_to_be_appended['销售单价/(元/斤)'].apply(get_average_price)

if __name__ == '__main__':
    add_from_file_1()
    add_from_file_2()
    calculate_expected_sales_volume()
    calculate_average_price()
    the_csv_to_be_appended.to_csv('full_table.csv', index=False, encoding='utf-8-sig')

```