这里换成你的论文的标题

摘 要

鉴于乡村地区的独特地理和气候条件，以及有限的耕地资源，发展有机种植产业显得尤为迫切。这不仅能最大限度利用现有耕地资源，而且能根据不同地区的具体情况实施差异化的种植模式，从而对乡村经济的可持续发展产生深远影响。因此，本研究建立种植优化决策模型，来指导乡村地区实现农业生产的可持续增长和经济效益的最大化。

针对问题一，在销售量、成本、产量和价格稳定的前提下，优化农作物种植方案。为此，构建**独立耕地种植协同优化模型**，其中**决策变量为不同作物在不同地块的种植量**，**目标函数为最大化总销售利润**。模型针对水稻以外的单季作物以及水稻与双季作物，分别设计了两个独立的**目标规划模型**。在第一个模型中，针对平旱地、梯田和山坡地，模型考虑了**种植面积、重茬限制、豆类作物种植频率、地块分散度和最小种植面积的关键约束条件**，以确保种植的合理性和可持续性。在第二个模型中，基于子模型一，针对水浇地、普通大棚和智慧大棚，并结合水稻及其他双季作物的具体种植计划，对目标和约束进行了相应的调整和优化以构建新的子模型二。为简化模型的求解过程，在确定相关参数值后，引入**新的0-1决策变量**以**线性化处理**目标函数和约束条件中的非线性部分。最终，通过Python的**OR-Tools工具包**对两个子模型分别进行求解，从而确定最优耕地种植策略。

针对问题二，为应对现实农业生产中农作物属性的不确定性，特别是预期销售量、种植成本、亩产量和销售价格的波动性，建立**随机规划模型**。模型中，引入**随机变量**来描述不同农作物各属性的随机增长率。鉴于缺乏具体的历史数据以确定随机变量的分布，本研究采用**法则**对部分随机变量进行**正态分布**的假设，而对于其他随机变量，则基于其变动范围设定为**均匀分布**。此外，所有**随机变量被假定为相互独立**。在继承并扩展问题一独立耕地种植协同优化模型的基础上，将原模型中的确定性目标函数进行修改，**转化为包含随机变量的期望函数**，从而将确定性优化问题转化为随机优化问题。为更精确地反映随机增长率的不确定性，模型中对相关的不确定变量进行了适当的调整。通过上述方法，成功建立随机规划模型，并最终运用**样本均值近似（SAA）方法**对模型进行求解。

针对问题三，

关键词：关键词1 关键词2 关键词3 关键词4

# 问题重述

为促进乡村经济可持续发展，需在华北山区乡村合理利用耕地资源，发展有机种植。该乡村拥有1201亩耕地，分为34个地块，包括平旱地、梯田、山坡地和水浇地，以及16个普通大棚和4个智慧大棚。平旱地、梯田和山坡地适合种植一季粮食，水浇地可种一季水稻或两季蔬菜。露天耕地和所有大棚均遵循重茬种植限制和豆类作物轮作要求，同时也要考虑方便耕种和田间管理的相关事宜。

根据所给的2023年种植数据，构建数学模型，研究以下问题：

1. 在2024-2030年间，基于2023年数据，探讨在销售量、成本、产量和价格稳定的前提下，如何制定该乡村未来八年农作物的最佳种植策略，以应对不同情况下产量过剩的挑战。
2. 面对农作物销售量、产量、成本和价格的波动，构建2024-2030年该乡村农作物的适应性种植模型，以应对市场变化和种植风险，确保最优种植策略。
3. 在问题2的基础上，考虑农作物间的相互影响和市场相关性，构建并优化2024-2030年乡村农作物的种植策略，通过模拟对比分析，揭示综合因素下的最佳种植方案。

# 问题分析

## 问题一的分析

问题一要求在已知农作物每年的预期销售量、种植成本、亩产量和销售价格的前提下，考虑不同产品的滞销情形，给出最优的农作物种植方案。本文以最大化总销售利润为目标，针对水稻以外的单季作物以及水稻与双季作物，分别构建了两个独立的目标规划模型。通过整合这两个模型，形成独立耕地种植协同优化模型，其结构如图1所示。

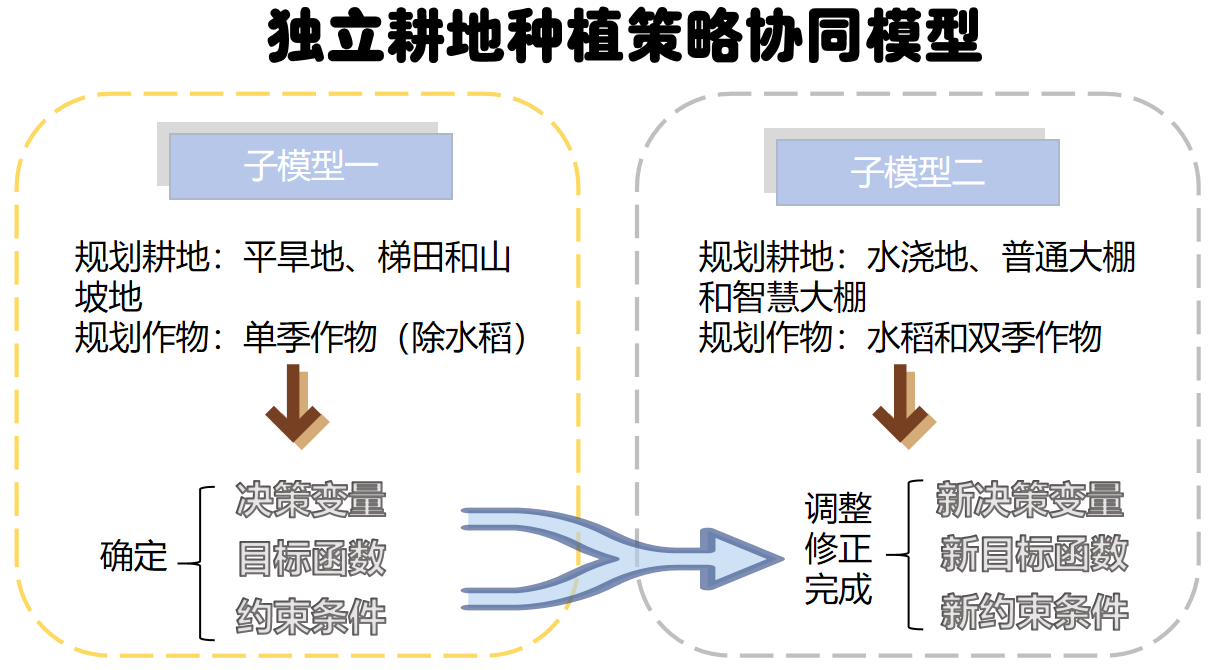


图 1 独立耕地种植协同优化模型的构成

首先，针对平旱地、梯田和山坡地，考虑除水稻外的单季农作物的种植情况。设置决策变量表示不同年份各个地块上不同农作物的种植量，而后分别考虑了以下约束条件：种植地面积约束，不能连续重茬种植约束，三年内至少种植一次豆类作物约束，种植地不会过于分散约束，以及种植面积不宜太小约束。基于此，建立目标规划模型求解最优种植方案。其次，针对水浇地、普通大棚和智慧大棚，结合水稻这一单季作物和其他双季作物在不同季的种植规划，在第一阶段单季作物目标规划模型的基础上，对相应的决策变量、目标函数和约束条件进行调整修正，建立了新的目标规划模型。而后，我们引入新的0-1决策变量表示不同年份各个地块上是否种植了某种农作物，基于此将原规划模型中的非线性部分进行线性转化，便于后续模型的求解。最后，我们采用Python的OR-Tools工具包对转化完的两个目标规划模型分别求解。

## 问题二的分析

在现实农业生产中，农作物属性如预期销售量、种植成本、亩产量和销售价格均存在不同程度的波动性。这些属性不再表现为确定性数值，而是在不同情境下呈现出随机变化。为了在不确定性环境中制定最优的种植策略，本研究构建了一个随机规划模型以求解相关问题。

与问题一中采用的确定性模型相比，本研究首先定义了随机变量，用以表示农作物j的属性s的随机增长率。鉴于缺乏具体的历史数据进行分析，本研究采用3σ法则对部分随机变量进行分布构造，将其设定为符合正态分布；对于其他随机变量，则根据其变动范围设定为均匀分布，并假设所有随机变量之间相互独立。在问题一的独立耕地种植协同优化模型的基础上，本研究对目标函数进行了扩展，以纳入期望的随机优化问题，即原确定性目标函数被修改为包含随机变量的期望函数。同时，为了反映随机增长率的潜在扰动，对相关变量进行了调整。通过这些步骤，本研究完成了随机规划模型的构建。最终，采用样本均值近似（SAA）方法对所建立的模型进行求解。

## 问题三的分析

# 模型假设

1.假设2023年产出的农作物全部售出；

2.假设问题二中的随机变量之间是相互独立的；

# 符号说明

|  |  |  |
| --- | --- | --- |
| **符号** | **说明** | **单位** |
|  | 第t年耕地i种植农作物j的量 | 亩 |
|  | 第t年所规划农作物的总销售额 | 元 |
|  | 第t年所规划农作物的总种植成本 | 元 |
|  | 农作物j的单位面积种植成本 | 元/亩 |
|  | 农作物j的销售单价 | 元/斤 |
|  | 农作物j在耕地i上的单位亩产量 | 斤/亩 |
|  | 农作物j每季的预期销售量 | 斤 |
|  | 超过部分的折扣利润率 | - |
|  | 地块i的总面积 | 亩 |

# 模型的建立与求解

## 问题一独立耕地种植协同优化模型的建立与求解

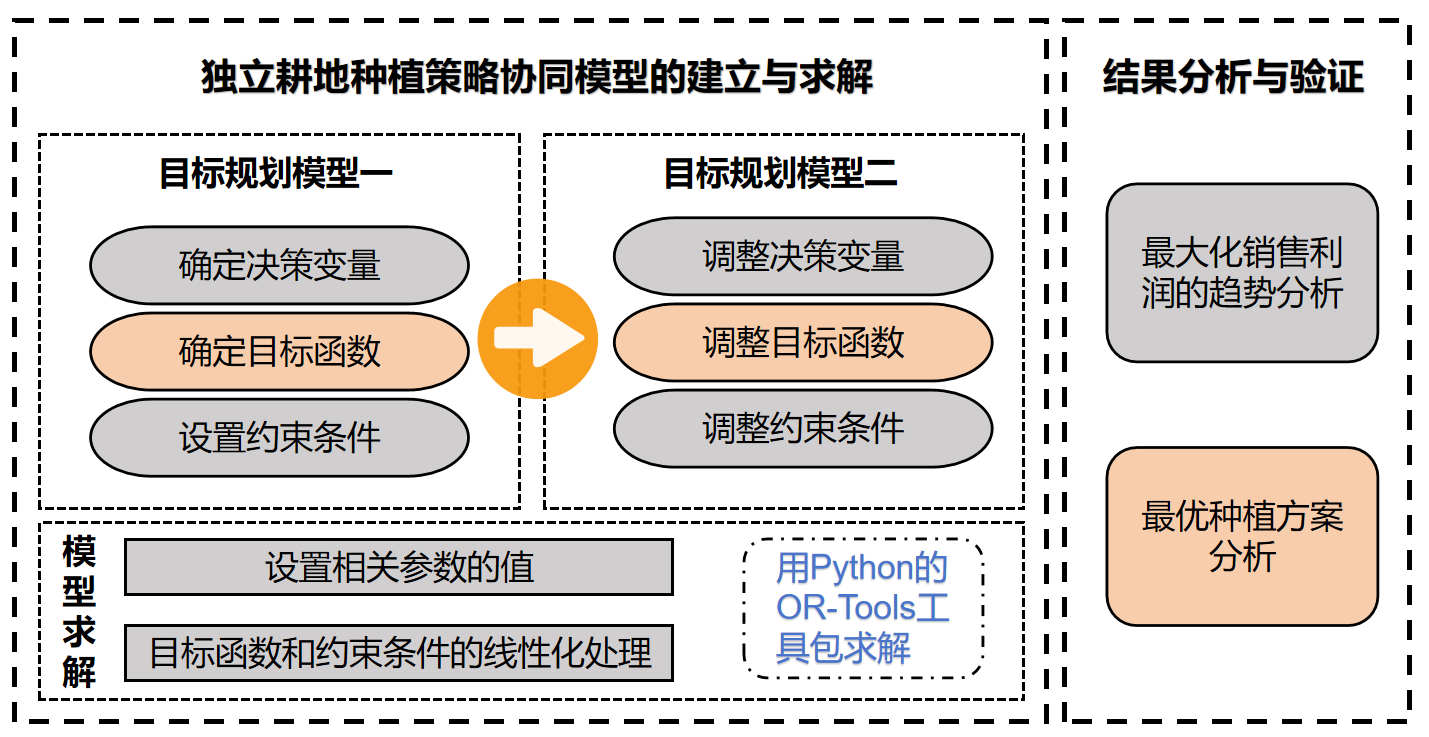


图 2 问题一思路框架图

根据题目要求，各种农作物未来的预期销售量、种植成本、亩产量和销售价格与2023年农作物种植的相关数据一致。由于所给的数据中并未直接给出2023年的农作物销售量数据，鉴于销售数据、市场供需状况、销售渠道的畅通性、价格稳定性、政府政策支持以及农民的经济动机的考虑，本研究假设2023年该村庄生产的农作物将全部售出。这一假设基于对农业市场环境的综合评估，旨在简化模型构建过程，同时确保研究结果的合理性和实用性。因此，将2023年各个农作物的总产量视为未来的预期销售量。

在深入分析数据的基础上，为确保种植结构的科学性安排，本研究针对水稻以外的单季作物以及水稻与双季作物，分别构建了两个独立的目标规划模型。通过整合这两个模型，形成了一个独立的耕地种植协同优化模型。该模型旨在在确保销售量、成本控制、产量稳定以及价格波动可控的前提下，为该乡村未来八年的农作物种植制定最优策略。

### 目标规划模型一的建立

我们首先考虑单季作物的种植规划问题。

单季作物中，除了水稻，其余都适合种植在平旱地、梯田和山坡地。考虑到水稻在水浇地的种植决策可能会影响水浇地其他双季作物的种植结构，我们首先仅仅考虑除水稻外的其他单季农作物的种植优化问题。

* **确定决策变量**

由题意可知，本问中的决策变量为每年各个耕地上每种农作物的种植量，因此我们引入决策变量表示第t年耕地i种植农作物j的量。

其中，，我们以2023年为起始时间，此时；，分别表示属于平旱地、梯田和山坡地类型的26个地块；，分别表示平旱地、梯田和山坡地可种植的15种农作物。

* **确定优化目标**

本问中需要制定种植方案使得农作物销售情况好，故本文将最大化销售利润作为优化的目标：

|  |  |
| --- | --- |
|  | () |

其中，表示第t年所规划农作物的总销售额，表示第t年所规划农作物的总种植成本。

针对种植成本，其计算如公式(2)所示：

|  |  |
| --- | --- |
|  | () |

其中，表示农作物j的单位面积种植成本。

针对销售额，包括预期销售量的销售和超过部分的折扣处理，分析可知，若总产量为达到预期销售量，则直接将所产农作物全部售出；若总产量超出预期销售量，则为超过部分按照预期销售量正常销售，超出部分作折扣处理。据此，其计算如公式(3)所示：

|  |  |
| --- | --- |
|  | () |

其中，表示农作物j的销售单价，表示农作物j在耕地i上的单位亩产量，表示农作物j每季的预期销售量，表示超过部分的折扣利润率：

|  |  |
| --- | --- |
|  | () |

* **确定约束条件**

1. 种植面积约束：地块i上所有种植物的总种植面积不得超过其总面积

|  |  |
| --- | --- |
|  | () |

1. 不能连续种植约束：作物j在同一地块i不能连续重茬种植

|  |  |
| --- | --- |
|  | () |

1. 豆类作物种植约束：为确保地块i在三年内至少有部分土地种植豆类作物j，允许在调整种植结构时，同一地块内不同区域种植不同的作物。关键在于，通过合理规划豆类作物j的种植区域，确保三年内豆类作物的总种植面积不小于地块i的总面积，从而实现所有土地在该周期内至少种植一次豆类作物的目标

|  |  |
| --- | --- |
|  | () |

1. 种植不过于分散的约束：为确保作物种植的集中性，避免过于分散，对于每种作物j，其每季的种植地点数量被限制在一个较小的范围内，具体设定为不超过一个常数N

|  |  |
| --- | --- |
|  | () |

1. 单个地块种植面积不过于小的约束：鉴于作物j在单个地块i上的种植面积不宜过小，且考虑到种植分散性的限制，分析表明该面积应相对于地块i的总面积而言。因此，本研究设定作物j在地块i上的种植面积应占地块总面积的至少一个比例，该比例被定义为常数M

|  |  |
| --- | --- |
|  | () |

综上，除水稻外的单季作物目标规划模型为：





### 目标规划模型二的建立

其次我们需要考虑水稻和双季作物的种植规划问题。

在农业种植规划中，鉴于水稻作为单季作物与部分双季作物在适宜种植条件上的相似性，尤其是在水浇地第一季的种植中，两者存在相互影响的关系，因此，有必要将水稻与双季作物纳入同一规划框架进行考量。

进一步地，通过对第一季与第二季种植时间的区分，我们发现水浇地和普通大棚在两个季节内的适宜种植作物类型存在显著差异，表现出相对独立性。然而，智慧大棚在两个季节中的可种植作物类型则高度一致。为了更有效地满足作物不能连续种植的约束条件，本研究选择将第一季和第二季的农作物种植规划合并为一个统一的决策过程。

此外，考虑到不同类型耕地（如水浇地、普通大棚和智慧大棚）对农作物种植的影响，以及计算每种农作物在每个季节预期销售量的便利性，本研究决定将各类耕地的农作物种植规划整合到一个模型中进行统一决策。

因此，本研究将水稻与双季作物在不同类型耕地上不同季节的种植规划问题综合起来，作为一个整体进行深入分析与决策，以实现更科学、高效的农业种植管理。

* **确定决策变量**

由题意可知，本问中的决策变量为每年每季各个耕地上每种农作物的种植量。

对于双季农作物，我们引入决策变量表示第t季耕地i种植农作物j的量：

|  |  |
| --- | --- |
|  | (0) |

其中，，我们以2023年的第一季为起始时间，此时；，分别表示属于水浇地、普通大棚和智慧大棚类型的28个地块；，表示j属于地块i所对应的不同类型耕地不同季的可种植作物类型集合。

对于水稻，由于其单季作物的性质，我们只需要考虑其每年在不同地块的种植量，相当于同一年中第一季和第二季的水稻种植情况完全一样，因此，设定决策变量：

|  |  |
| --- | --- |
|  | (11) |

该决策变量表示第T年水稻在地块i的种植量，其中，，我们仍然以2023年的为起始时间，此时。

* **确定优化目标**

本模型中的优化目标仍然是最大化销售利润，其定义和计算方法与第一阶段单季作物目标规划模型相同，此处不再赘述，具体见公式(1)(2)(3)(4)。

* **确定约束条件**

1. 种植面积约束：第t个季时在地块i上水稻和双季作物的总种植面积不大于该地块的总面积

|  |  |
| --- | --- |
|  | (12) |

1. 不能连续种植约束：对于水稻来说，即相邻年份间不能都种植水稻；对于其他双季作物来说，即相邻种植季之间不能种植同一种作物

|  |  |
| --- | --- |
|  | (13) |

1. 豆类作物种植约束：在本模型中，种植季被选定为时间的基本单位。为了确保地块i上的所有土地在三年内至少种植一次豆类作物，只需保证在六个种植季内豆类作物的总种植面积至少与地块i的总面积相匹配即可

|  |  |
| --- | --- |
|  | (14) |

1. 种植不宜过于分散的约束和单块种植面积不宜过小的约束与单季作物目标规划模型一致，具体见公式(8)(9)

综上，水稻和双季作物目标规划模型为：





### 目标规划模型的求解

为了便于模型的求解，在其之前，我们首先需要设定相关参数值，线性化非线性部分。以下我们将以子模型一为例阐述具体的处理过程。

* 相关参数值的确定

1. 单价

鉴于所提供的数据仅包含农作物的单价区间（即销售期内的最低和最高售价），而具体销售情况，如销售时长等细节信息未知，进一步考虑到销售期内农作物的实际售价可能围绕某一平均值波动，且为了简化计算并确保模型分析的一致性，我们决定采用该区间的平均单价作为模型输入。此方法基于对市场波动的一般性假设，即农作物的售价在销售期内将均匀分布在该区间内，从而在缺乏具体销售数据的情况下，提供一个合理的近似值，以支持模型的有效性和预测准确性。

1. 预期销售量

通过模型中关于预期销售量的分析，可知我们将2023年每个农作物每年的总产量视为未来的预期销售量，经过计算可得的部分作物的预期销售量如表1所示。

表 1 部分作物的预期销售量

|  |  |  |  |
| --- | --- | --- | --- |
| 作物编号 | 作物名称 | 作物类型 | 预期销售量/斤 |
| 1 | 黄豆 | 粮食（豆类） | 57000 |
| 2 | 黑豆 | 粮食（豆类） | 21850 |
| 3 | 红豆 | 粮食（豆类） | 22400 |
| 4 | 绿豆 | 粮食（豆类） | 33040 |
| 5 | 爬豆 | 粮食（豆类） | 9875 |
| 6 | 小麦 | 粮食 | 170840 |
| 7 | 玉米 | 粮食 | 132750 |
| 8 | 谷子 | 粮食 | 71400 |
| 9 | 高粱 | 粮食 | 30000 |
| 10 | 黍子 | 粮食 | 12500 |

1. 常数N和M

对于一种农作物在每一年最多可种植地块数目N，我们基于提高生产效率、增强市场竞争力、保护生态环境、降低风险以及符合农业规划等多重因素的综合考虑，选取3个种植地作为每种作物每季的最大种植地个数。

对于一种农作物最小的单块面积种植比例M，我们基于经济效益、土壤健康、市场接受度、技术管理、环境因素和政策法规等多方面因素的综合考量，最终选取0.4作为单块耕地种植某种作物面积的最小比例，具有一定科学性和合理性。

* 约束条件的线性转化

1. 约束条件(2)

我们发现，设置好的约束条件(2)是一个具有较强约束性的非线性等式约束。这种约束条件可能会限制优化算法的搜索空间，使得算法难以找到最优解。因此，我们首先引入变量表示第t年在地块i上农作物j是否耕种：

|  |  |
| --- | --- |
|  | (5) |

为了便于优化算法的求解，我们用如下公式(14)所示的线性约束条件组表示决策变量和的关系：

|  |  |
| --- | --- |
|  | (16) |

其中，MM是一个足够大的常数，确保当时，能够无限接近0。

在此基础上，我们可以将约束条件(2)修改为与决策变量有关的线性不等式约束，如公式(15)所示：

|  |  |
| --- | --- |
|  | (17) |

1. 约束条件(5)

约束条件(5)是一个分段不等式约束，我们可以再次利用新引入的决策变量对其进行线性化处理，如公式(16)所示：

|  |  |
| --- | --- |
|  | (18) |

1. 约束条件(4)

相应的，约束条件(4)中农作物j每年的种植地个数可以表示为：

|  |  |
| --- | --- |
|  | (19) |

* 目标函数的线性转化

基于线性规划中的等价原理（的线性转化同理，具体证明过程见附录？），我们首先将原目标函数中的非线性部分化简表示为：

|  |  |
| --- | --- |
|  | (20) |

* 求解方法的选定

基于以上参数值的选定、目标函数与条件的线性化处理，我们使用Python的OR-Tools工具包对模型进行求解，计算所得平旱地、梯田和山坡地的每年的最大利润。

在处理和求解子模型二时，我们采用了与先前模型相同的策略。然而，在求解过程中，我们遇到了一个特定情况，即当参数N取值为3时，模型无法找到可行解。针对这一情况，我们对水浇地、普通大棚和智慧大棚的耕地面积以及农作物的预期销售量进行了重新评估，并将参数N调整为5。此外，鉴于智慧大棚的种植面积限制，我们注意到在第二季中，仅有8种蔬菜有预期销售量。考虑到实际种植条件，我们推断其他蔬菜的预期销售量并非零。进一步分析智慧大棚第二季其他蔬菜的种植数据，我们发现其种植面积普遍约为0.3亩。基于此，我们将那些未明确预期销售量的蔬菜的可能种植量统一设定为0.3亩以计算其对应的预期销售量，以反映实际种植的均衡分布。

分别求解两个子模型，将所得的最优种植安排合并即可获得整个最优耕地种植安排。

### 结果分析

* 最优种植方案分析

以2024年的种植方案为具体案例，其详细的种植布局如图3所示。图中，灰色区域标示了不宜种植特定作物的地块，而浅红色区域则指示了已安排种植农作物的区域。通过观察，我们可以确认以下几点：

1. 每个地块都根据其特性合理分配了适宜的农作物，确保了种植的适宜性；农作物种植分布均匀，没有太分散，这有助于简化田间管理流程；
2. 充分利用了每一块耕地，有利于实现收益的最大化；

此外，通过对比分析不同年份的农作物种植安排，我们发现：

1. 并未出现连续重茬种植的情况，这一做法符合农作物的种植规律，有助于维持土壤的健康和肥力；
2. 豆类植物的合理安排，进一步确保了耕地营养性成分的持续供应。

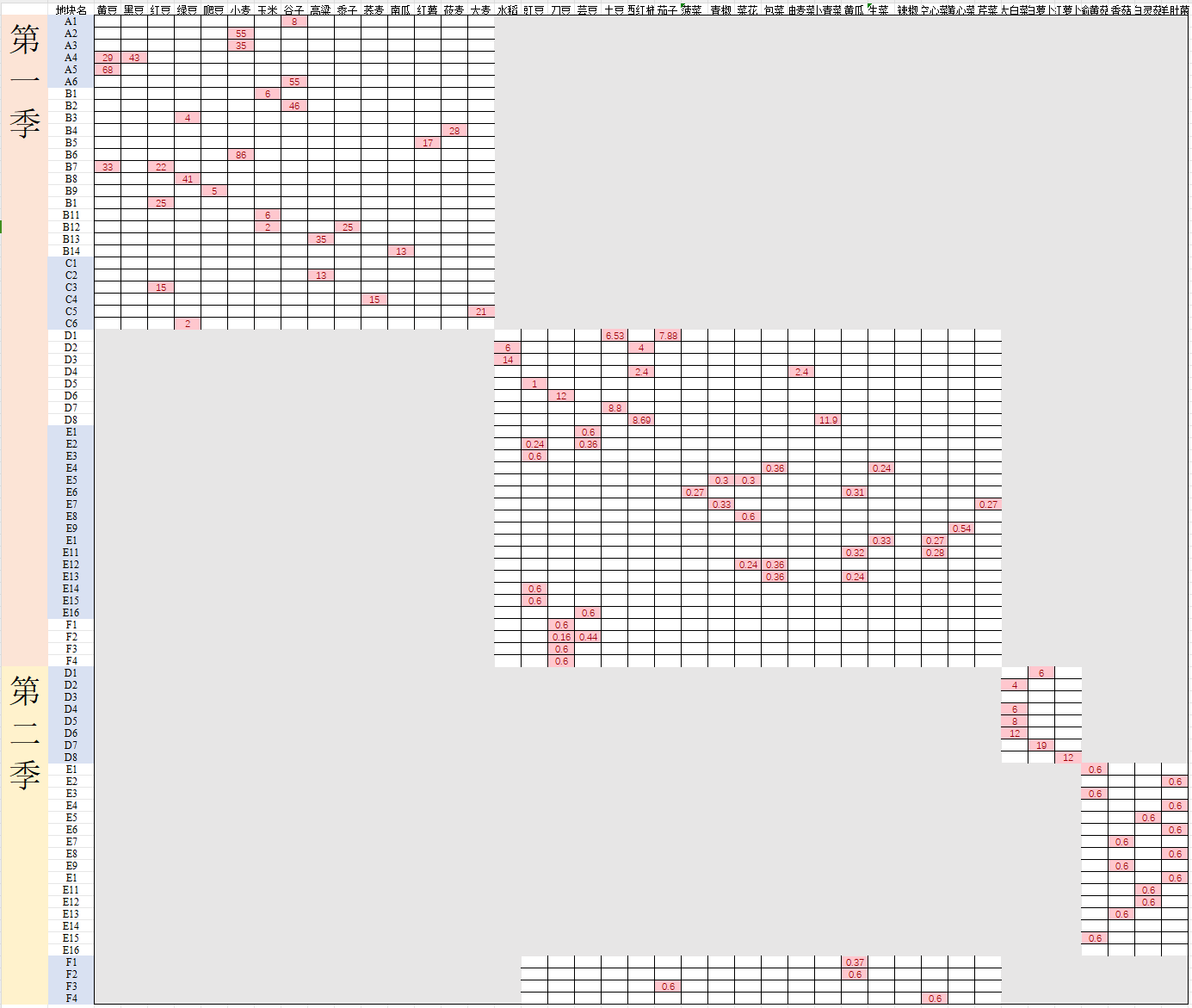
综上所述，本耕地种植协同优化模型在农作物种植安排上表现出卓越的性能。

图 3 最优种植安排(2024年)

* 最大销售利润对比分析

最优种植策略下的最大利润如图4所示。我们可以发现2024-2030年的最大利润与2023年接近，可以说明模型结果是符合该村庄农作物销售的实际规律的

图 4 最大化利润值

从每年的最大化利润和具体的种植安排来看，该模型求解出的解较为合理，与所给数据的实际情况相符。具体安排见支撑材料“result1\_1.xlsx”和“result1\_2.xlsx”。

## 问题二随机规划模型的建立与求解

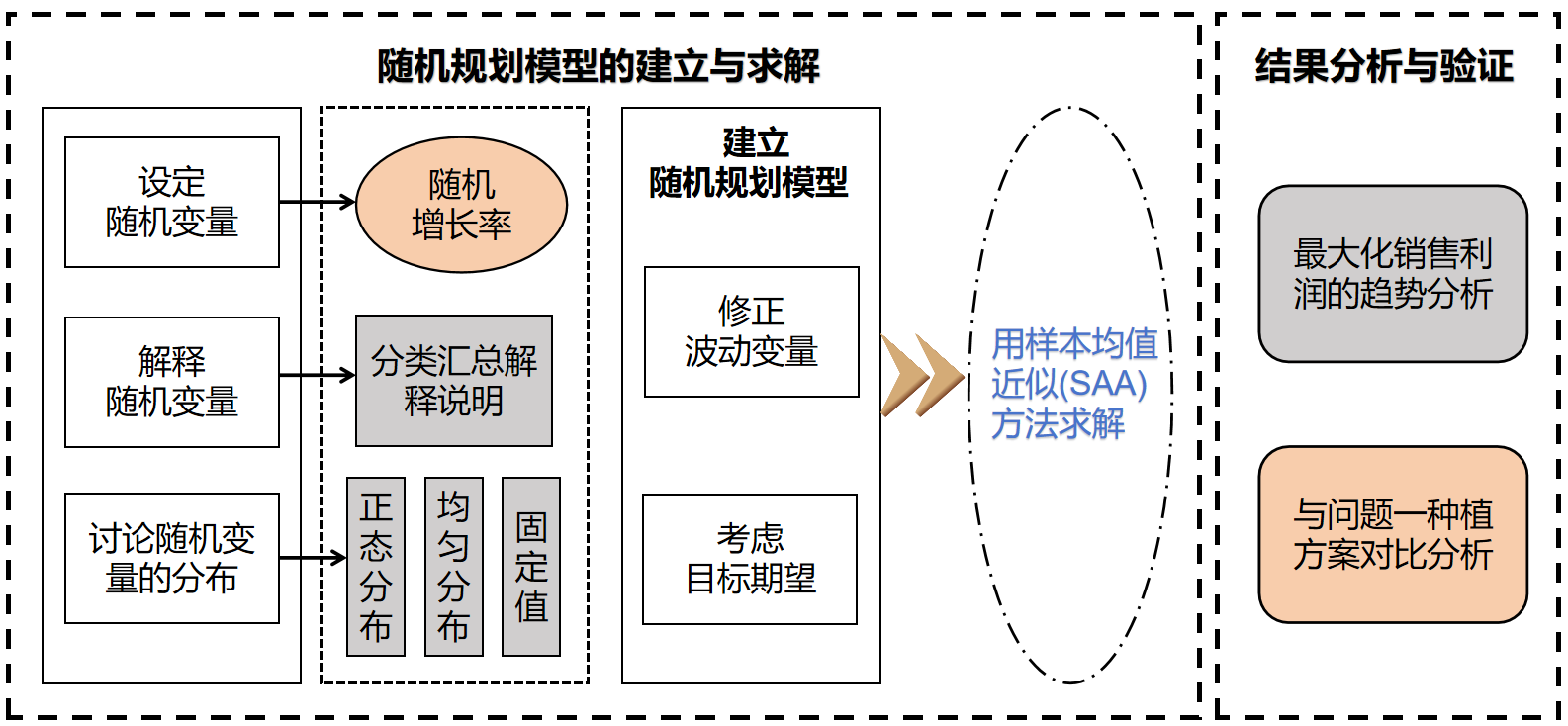


图 5 问题二思路框架图

相较于问题一中的种植规划模型，本问中将约束条件中的系数和目标函数中的部分变量设置为与随机变量相关的变量。为了研究该具有不确定性的随机问题，本文建立了随机规划模型进行求解。

### 随机变量的解释

相关随机变量及其具体的变化情况如下：

1. 小麦和玉米未来的预期销售量有增长的趋势，平均年增长率介于5%~10%之间；
2. 除玉米和小麦的其他农作物未来每年的预期销售量相对于2023年大约有±5%的变化；
3. 农作物的亩产量往往会受气候等因素的影响，每年会有±10%的变化；
4. 因受市场条件影响，农作物的种植成本平均每年增长5%左右；
5. 粮食类作物的销售价格基本稳定，在此我们认为其销售价格不变；
6. 蔬菜类作物的销售价格有增长的趋势，平均每年增长5%左右；
7. 食用菌的销售价格稳中有降，大约每年可下降1%~5%，特别是羊肚菌的销售价格每年下降幅度为5%。

在建立随机规划模型前，我们首先要对随机变量的分布情况进行讨论，从而为决策提供基于统计学的稳健性和灵活性，确保模型能够适应现实世界中的不确定性。在此我们假设这些随机变量之间是相互独立的。由于本题中并没有相应的历史数据供我们分析，因此，我们将仅仅根据随机变量的变动范围，分析设定合理的分布情况，具体设定如下：

1. 小麦和玉米未来的预期销售量会增长，其幅度在5%~10%之间，考虑到这类农作物可能代表市场需求的微小变化或生产成本的轻微调整。在农业种植结构决策中，这类变化通常被视为相对稳定且均匀分布的，因为它们可能受到多种不可预测的小因素影响，如消费者偏好、竞争对手行为等，我们可以认为这些因素在5%到10%的范围内均匀分布。因此我们将**小麦和玉米未来的预期销售量的增长率视为均匀分布**；
2. 除玉米和小麦的其他农作物未来每年的预期销售量有增长或减少5%的变化，在实际农业生产中，许多因素如天气条件、市场波动等都会导致产量或成本出现波动。正态分布能够模拟这些因素引起的随机波动，因为它们往往集中在平均值附近，且两侧的波动是对称的。例如，农作物的产量可能会因为天气异常（如干旱或洪涝）而上下波动，这种波动符合正态分布的特性。因此我们将**除玉米和小麦的其他农作物未来每年的预期销售量的增长率**根据原则，将所给的设为边界构造**正态分布**；
3. **农作物的亩产量增长率**与除玉米和小麦的其他农作物未来每年的预期销售量增长率同理，同样根据原则，将所给的设为边界构造**正态分布**；
4. **农作物的种植成本的增长率**平均每年5%左右，基于对实际增长趋势的合理预期和对潜在波动风险的谨慎评估，我们仍然运用原则构造**正态分布**。通过分别设定5%这一增长率的**0.9倍和1.1倍作为分布边界**，我们能够预测大多数情况下增长率将集中在5%附近，同时识别出超出这一范围的概率较低；
5. **粮食类作物的销售价格我们视为固定值**，始终与2023年数据一致；
6. **蔬菜类作物的销售价格的增长率**与种植成本增长率同理，构造**正态分布**；
7. **食用菌销售价格的下降率**与小麦和玉米未来的预期销售量的增长率同理，构造**均匀分布**；
8. **羊肚菌的销售价格下降幅度每年固定为5%**。

### 随机规划模型的建立

根据题目要求，此处的随机变量即每种作物的预期销售量、种植成本、亩产量和产品价格的增长率（若有数据有所下降则将增长率视作负值即可），将随机变量设为，表示农作物的属性对应的随机增长率，不同农作物的变动属性分别设为与随机变量相关的形式：。

将原确定性目标规划模型中的目标函数设为，其中，X=[]，Y=[]。

对于随机规划模型，预期销售量、种植成本、亩产量和产品价格的增长率都是不确定的随机变量，在此，我们需要考虑包含期望的随机优化问题，因此我们将其目标函数设为如下形式：

|  |  |
| --- | --- |
|  | (21) |

其中，每个随机变量的概率分布对应的服从。我们设是随机变量的样本，是在样本下的一次实现。

根据相关变量的变动要求，我们需要将一些确定性因素改为与随机变量相关的变化因素，具体如下所示：

1. 小麦和玉米的未来预期销售量、农作物的种植成本、蔬菜类作物的销售价格、食用菌的销售价格和羊肚菌的销售价格都是每年相对前一年会有一定的变动程度，对此我们将其表示为：

|  |  |
| --- | --- |
|  | (22) |

1. 除玉米和小麦的其他农作物未来每年的预期销售量会在2023年的亩产量数据的基础上有一定程度的变动，对于农作物的亩产量，考虑到气候等因素的影响在前后种植年份之间没有必然的联系，在此我们认为亩产量也是相对于2023年的亩产量会有一定的上下波动，为此我们将其表示为：

|  |  |
| --- | --- |
|  | (23) |

### 模型求解及结果分析

对于该随机规划模型，通过转换我们可以将其转化为对应的随机线性规划模型，对此采用样本均值近似(SAA)方法进行求解。

根据SAA原理，定义目标函数的样本均值近似为：

|  |  |
| --- | --- |
|  | (24) |

其中，表示第n次的采样值。

此时相应的目标函数转化为：

|  |  |
| --- | --- |
|  | (25) |

最终需要求解的决策变量在每次抽样中始终不变，这样我们便将随机规划问题转化为了确定性优化问题进而求解。

在此次求解时，我们将农作物超过预期销售量的部分作滞销处理，即令.设采样次数，运用SAA方法求解的结果：

## 问题三模型的建立与求解

### 替代性和互补性探究

我们首先需要定义农作物作为销售的商品时，其之间的可替代性和互补性的概念。在此我们用经济学中的相关概念对其进行解释。

商品之间的互补性是指两种或多种商品在消费过程中存在相互增强效用或满足消费者需求的特性，它们之间的使用通常伴随着正的相互依赖关系。具体而言，当一种商品的需求增加时，另一种商品的需求也随之增加，两者在消费体验上相互补充，共同构成消费者满足特定需求的整体。

商品之间的可替代性则是指两种或多种商品在满足消费者同一种需求时，能够相互替换，彼此之间具有功能上的相似性。这种替代性基于消费者对商品属性的感知，当一种商品的价格、质量或可获得性发生变化时，消费者可能会转向另一种商品，以维持其需求满足的连续性。在可替代商品之间，存在一种负的相互依赖关系，即一种商品的需求增加可能导致另一种商品的需求减少。

* **可替代性**

通过分析某一农作物商品价格变动时，其他农作物商品的预期销售量变化即可选定该农作物的互补商品或替代商品。考虑到本题中缺少农作物销售的历史数据，且没有任何可知支撑去进行数据合理模拟的依据，基于《国民经济行业分类》 (GB/T4754—2011) 和《统计用产品分类目录》，本文的农作物可以分为分成以下几类：

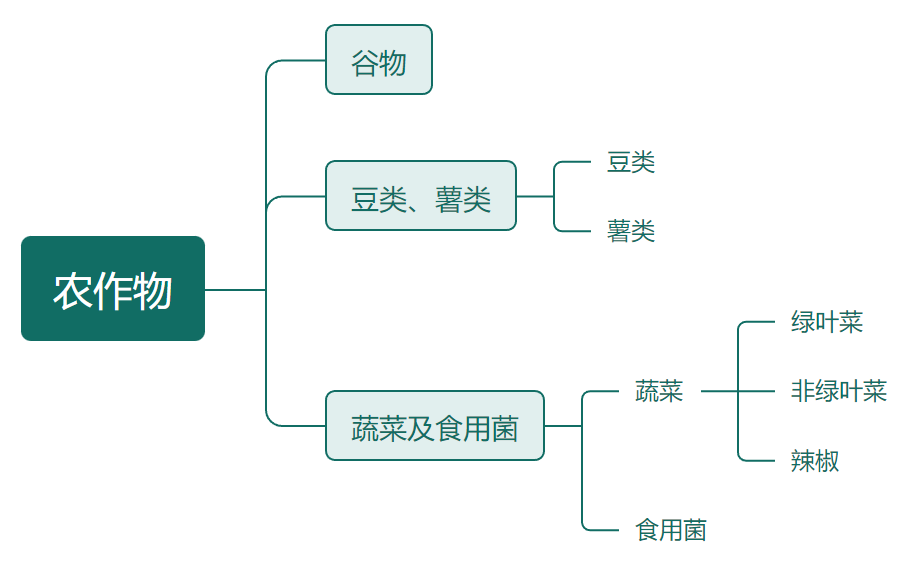


图 6 农作物分类

参考文献[2]，我们得知基于分类的标准，可以认为同属一类的农作物之间有可替代性，且分类越往内层，同属一类的农作物之间的可替代性越强。

* **互补性**

农作物产品作为销售商品时，其之间的的互补性可以表现为以下几种情况：

1. 使用配套：某些农作物产品在使用时需要配套其他产品。例如，购买小麦时，消费者可能也会购买面粉，因为小麦需要磨成面粉才能制作成面包或面条。
2. 营养补充：不同的农作物产品可能提供不同的营养成分，消费者可能会同时购买多种农作物产品以获得全面的营养。例如，购买黄豆时，消费者可能也会购买富含不同维生素和矿物质的蔬菜。
3. 消费习惯：根据消费者的口味和习惯，可能在购买某种蔬菜时同时购入想与其搭配一同吃的其他蔬菜。

我们可以说农作物产品的互补性是指它们在需求、生产、市场和营养等方面相互依存和相互促进的关系。以本题中种植的农作物为例，可以视为具有互补性的农产品有：蔬菜与谷物，消费者在购买蔬菜的同时，可能会购买谷物作为主食，以实现营养均衡。

诸如此类的具有互补性的农作物还可以列举很多，需要注意的是，互补性并不是绝对的，它取决于具体的使用场景和消费者的偏好。

### 预期销售量的修正

为了深入探究农作物预期销售量与农作物间的互补性及可替代性、销售价格以及种植成本之间的相互作用，我们经过广泛文献调研，最终决定采用Rotterdam需求系统模型对农作物的预期销售量进行精细化修正。该模型的优势在于其能够同时捕捉销售量自身动态变化的特点，并充分考虑农作物间的相互关系以及销售价格对销售量的影响，其基本模型结构表述如下：

|  |  |
| --- | --- |
|  | (26) |

其中，是指商品i的消费数量，是指商品k的销售价格，是指所有商品总的消费金额，即。对于参数，根据经济学理论，是商品i的自主需求参数，即 ；商品i与j的交叉价格弹性系数，该参数反应了商品之间的可替代关系和互补关系，在该模型中，当时，表明商品k的价格上涨对商品i的需求量有正向影响，故其之间存在可替代性，当时，表明商品k的价格上涨对商品i的需求量有负向影响，表明商品i与k之间存在互补性。

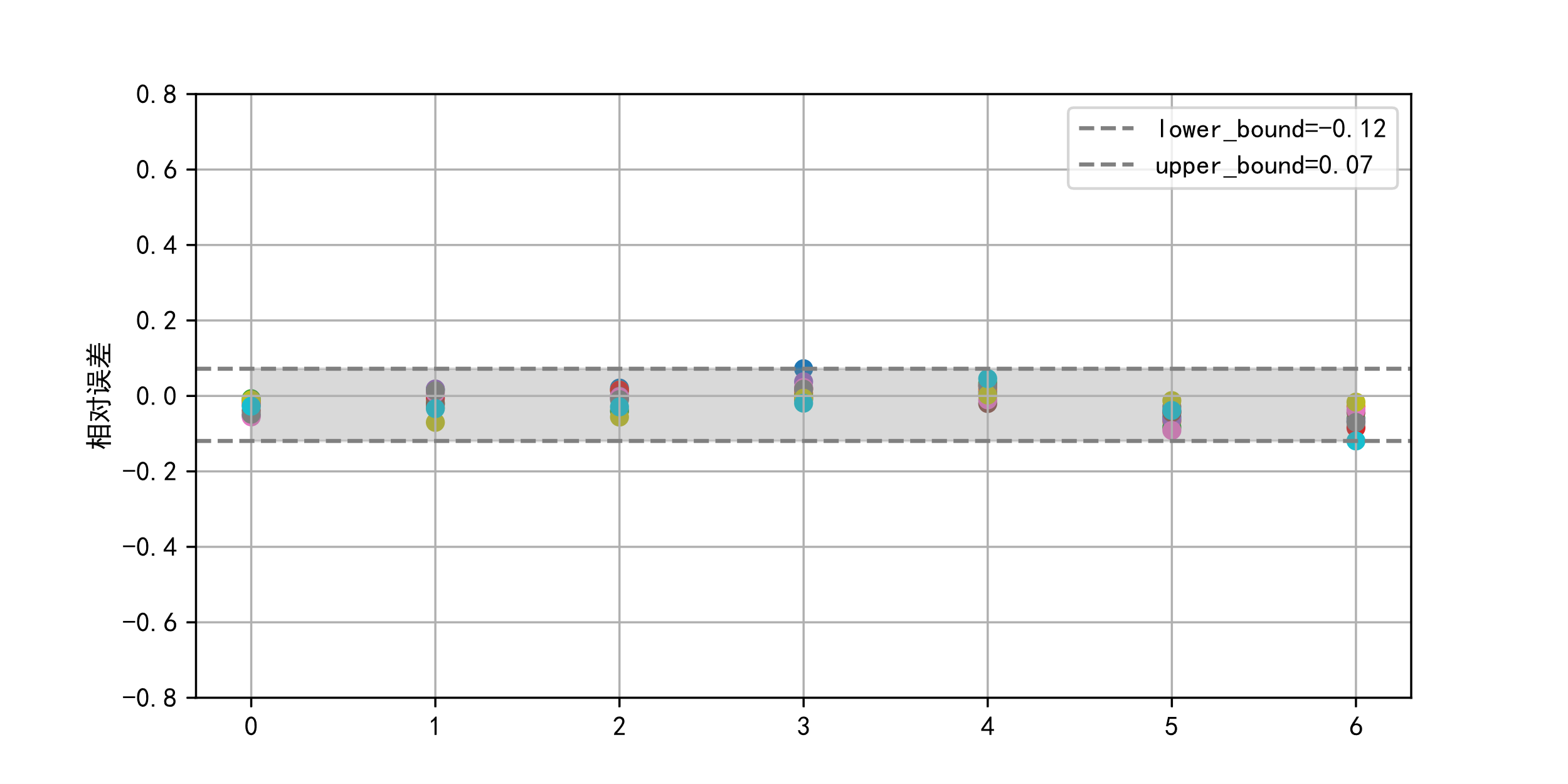
为使模型满足齐次性要求，模型参数应分别满足以下约束：

|  |  |
| --- | --- |
|  | (27) |

### 随即规划模型的修正与求解

# 模型的分析与检验

在本研究中，为了对模型的稳健性和敏感度进行全面分析，我们对以下关键参数进行了调整：各作物的种植面积、出售价格、预期销量、单位亩产值在原始数据基础上波动范围为 0.95-1.05倍。基于上述调整，模型中所有作物的输入参数在合理波动范围内变化，以此评估模型在不同情况下的最大利润的变化情况。



图中展示了模型在不同参数设置下的相对误差，从图中可以观察到相对误差变化幅度较小，不同作物种植面积和预期销量的波动对最大收益没有显著的影响。

随着参数变化，粮食作物最大利润的相对误差保持在[-0.12,0.07]。这表明这些参数的变化对模型最大利润的影响较为稳定，模型在这些参数波动下具有较好的鲁棒性。

# 模型的评价、改进与推广

## 模型的优点

**保证全局最优性**。在充分的时间内，线性规划可以通过精确的算法（如单纯性法，内点法）保证全局最优性，尤其是在非凸问题中，可以避免非线性规划的局部最优陷阱。

**保证求解的稳定性**。通过约束线性化，非线性规划问题转换为线性模型，模型求解过程中更加稳定，避免非线性求解中的数值不稳定。

**模型可扩展性强**。转换为线性模型优化，模型具有更多的成熟且高效的求解器处理线性问题，拥有更多的接口和优化选型对模型效果进一步提升，并且可以获得比非线性求解器更大的使用范围，使模型易于扩展。

## 模型的缺点

求解器求解约束增多。在约束线性化过程中，约束数量显著增加，模型的运行时间随之加长，由于时间原因，求解器并未求出最优解，本文选择在可接受的求解时长内，获得最优的可行解。

## 模型的改进

简化冗余的约束，降低了计算复杂度，减少因线性化求解约束增多造成的时间影响，取得全局最优解。

提高线性化约束的精度，采用拟合度更高的分段线性化或高阶近似，提高非线性函数转换的精度。

选择处理大量约束的求解器。本文使用开源的OR-Tools作为线性求解器，预算允许情况下，建议选择标准求解器Gurobi 和 CPLEX**。**

## 模型的推广

引入多目标优化。除了最大化利润或最小化成本外，考虑引入其他目标，如可持续发展、资源使用效率或客户满意度。。

考虑模型的可扩展性，模型可以扩展至其他类型的土地。考虑南北方地区显著的气候差异对农作物生长和生产的影响。在模型中引入季节性变化因素，例如温度、降水量、光照等，以优化各种植物在其他类型土地上大棚和露天种植的决策。

# 参考文献

附录

|  |
| --- |
| 附录1 |
| 介绍：支撑材料的文件列表 |
|  |

|  |
| --- |
| 附录2 |
| 问题一：使用OR-Tools求解模型一线性规划 |
| 1. import pandas as pd 2. area\_class=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\附件1.xlsx") 3. area\_class=area\_class.drop(columns=['说明 ']) 4. area\_class.rename(columns={'地块名称': '种植地块'}, inplace=True) 5. area\_class 6. import pandas as pd 7. from openpyxl import load\_workbook 8. data2=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\附件2.xlsx",sheet\_name=0) 9. data2 = data2.fillna(method='ffill') 10. type\_mapping = {'A': '平旱地', 'B': '梯田', 'C': '山坡地', 'D': '水浇地', 'E': '普通大棚', 'F': '智慧大棚'} 11. data2['地块类型'] = data2['种植地块'].apply(lambda x: type\_mapping[x[0]]) 12. crop=data2[['种植地块','作物编号','作物名称','作物类型']] 13. crop['是否豆类'] = crop['作物类型'].apply(lambda x: 1 if '豆类' in x else 0) 14. crop\_class=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\附件1.xlsx",sheet\_name=1) 15. crop\_class=crop\_class.drop(columns=['说明','种植耕地']) 16. crop\_class['是否豆类'] = crop\_class['作物类型'].apply(lambda x: 1 if '豆类' in x else 0) 17. market=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\附件2.xlsx",sheet\_name=1,index\_col=0) 18. muchan=market[['作物编号','地块类型','亩产量/斤']] 19. crop1=data2[data2['地块类型'].isin(['平旱地', '梯田','山坡地'])] 20. x0=crop1[['种植地块','作物编号','种植面积/亩']].pivot\_table(index='种植地块', columns='作物编号', values='种植面积/亩', aggfunc='sum', fill\_value=0) 21. y0=x0.copy() 22. y0[y0>0]=int(1) 23. one\_area=area\_class[area\_class['地块类型'].isin(['平旱地', '梯田','山坡地'])] 24. *# one\_area.set\_index('种植地块',inplace=True)* 25. *# 创建一个作物编号列的表，将作物编号作为列名* 26. one\_muchan=muchan[muchan['地块类型'].isin(['平旱地', '梯田','山坡地'])].pivot\_table(index='地块类型', columns='作物编号', values='亩产量/斤').reset\_index() 27. pro= pd.merge(one\_area['地块类型'],one\_muchan, on='地块类型', how='left').drop(columns=['地块类型']) 28. b=market[market['地块类型']=='平旱地'][['种植成本/(元/亩)']] 29. s=market[market['地块类型']=='平旱地'][[ '种植成本/(元/亩)']] 30. Exp=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\exp.xlsx") 31. Exp=Exp.drop(columns=['作物编号']) 32. sc=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\s.xlsx") 33. bean\_class\_indices=crop\_class['是否豆类'].iloc 34. bean\_class\_indices = [i for i, value in enumerate(bean\_class\_indices[:15]) if value == 1] 35. bc=b['种植成本/(元/亩)'] 36. sc=sc['平均单价'].iloc[:15] 37. pro.iloc[0,0] 38. area=one\_area['地块面积/亩'].iloc[:] 39. Exp.iloc[0,0] 40. from ortools.linear\_solver import pywraplp 41. ni=26 42. nj=15 43. nt=7 44. solver = pywraplp.Solver.CreateSolver('CP-SAT') 45. y = [[[solver.IntVar(0, 1, f'y\_{t}\_{i}\_{j}') for j in range(nj)] for i in range(ni)] for t in range(nt)] 46. x = [[[solver.IntVar(0, 86, f'x\_{t}\_{i}\_{j}') for j in range(nj)] for i in range(ni)] for t in range(nt)] 47. B = [solver.NumVar(0, solver.infinity(), f'B\_{t}') for t in range(nt)] 48. S = [solver.NumVar(0, solver.infinity(), f'S\_{t}') for t in range(nt)] 49. for t in range(nt): 50. B[t] = solver.Sum(bc.iloc[j] \* x[t][i][j] for i in range(ni) for j in range(nj)) 51. output =[[solver.NumVar(0, solver.infinity(), f'output\_{t}\_{j}') for j in range(nj)] for t in range(nt)] 52. remain =[[solver.NumVar(0, solver.infinity(), f'remain\_{t}\_{j}') for j in range(nj)] for t in range(nt)] 53. z1=[[solver.IntVar(0,1 , f'output\_{t}\_{j})') for j in range(nj)] for t in range(nt)] 54. z2=[[solver.IntVar(0,1 , f'output\_{t}\_{j})') for j in range(nj)] for t in range(nt)] 55. for t in range(nt): 56. sum\_output = solver.Sum([sc[j] \* output[t][j] for j in range(nj)]) 57. sum\_remain = theta\*solver.Sum([sc[j] \* remain[t][j] for j in range(nj)]) 58. for j in range(nj): 59. *# 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij}* 60. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] for i in range(ni)) 61. solver.Add(output[t][j] <= sum\_pro) 62. solver.Add(output[t][j] <= Exp.iloc[j,0]) 63. solver.Add(remain[t][j]<=sum\_pro+1000\*z1[t][j]) 64. solver.Add(remain[t][j]<= 0+1000\*z2[t][j]) 65. solver.Add(remain[t][j]>=sum\_pro-Exp.iloc[j,0]) 66. solver.Add(remain[t][j]>=0) 67. solver.Add(z1[t][j]+z2[t][j]==1) 68. S[t]=sum\_output+sum\_remain 69. for i in range(ni): 70. for t in range(nt): 71. constraint\_expr = solver.Sum(x[t][i][j] for j in range(nj)) 72. solver.Add(constraint\_expr <= area[i]) 73. for t in range(nt): 74. if t==0: 75. for i in range(ni): 76. for j in range(nj): 77. solver.Add(y[t][i][j] + y0.iloc[i,j] <= 1) 78. else: 79. for i in range(ni): 80. for j in range(nj): 81. solver.Add(y[t][i][j] + y[t -1][i][j] <= 1) 82. big\_M = 10000 83. for t in range(nt): 84. for i in range(ni): 85. for j in range(nj): 86. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 87. for i in range(ni): 88. for t in range(1, nt): 89. if(t==1): 90. *# 创建约束* 91. constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x0.iloc[i,j] for j in bean\_class\_indices) 92. solver.Add(constraint\_expr >= area[i]) 93. else: 94. *# 创建约束* 95. constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x[t-2][i][j] for j in bean\_class\_indices) 96. solver.Add(constraint\_expr >= area[i]) 98. N = 3 99. for j in range(nj): 100. for t in range(nt): 101. *# 创建约束* 102. constraint\_expr = solver.Sum(y[t][i][j] for i in range(ni)) 103. solver.Add(constraint\_expr <= N) 104. M = 0.3 105. for t in range(nt): 106. for j in range(nj): 107. for i in range(ni): 108. *# 创建约束* 109. constraint\_expr = x[t][i][j] >= M \* y[t][i][j] \* area[i] 110. solver.Add(constraint\_expr) 111. solver.SetTimeLimit(120000) 112. solver.EnableOutput() 113. objective\_expr = solver.Sum(S[t] for t in range(nt)) - solver.Sum(B[t] for t in range(nt)) 114. solver.Maximize(objective\_expr) 115. status = solver.Solve() 116. *# 输出结果* 117. import numpy as np 118. if status == pywraplp.Solver.OPTIMAL: 119. print('Optimal solution found.') 120. print('Solution:') 121. for t in range(nt): 122. print(S[t].solution\_value(),B[t].solution\_value(),S[t].solution\_value()-B[t].solution\_value()) 123. x\_np = np.array([[[x[t][i][j].solution\_value() for j in range(nj)] for i in range(ni)] for t in range(nt)]) 124. y\_np = np.array([[[y[t][i][j].solution\_value() for j in range(nj)] for i in range(ni)] for t in range(nt)]) 125. *# x\_np= np.round(x\_np)* 126. elif status == pywraplp.Solver.INFEASIBLE: 127. print('Problem is infeasible.') 128. elif status == pywraplp.Solver.UNBOUNDED: 129. print('Problem is unbounded.') 130. elif status == pywraplp.Solver.ABNORMAL: 131. print('Solver encountered an abnormal termination.') 132. else: 133. print(status) 134. for t in range(nt): 135. print(S[t].solution\_value(),B[t].solution\_value(),S[t].solution\_value()-B[t].solution\_value()) 136. print('x values:') 137. x\_np = np.array([[[x[t][i][j].solution\_value() for j in range(nj)] for i in range(ni)] for t in range(nt)]) 138. y\_np = np.array([[[y[t][i][j].solution\_value() for j in range(nj)] for i in range(ni)] for t in range(nt)]) 139. file\_path = r'D:\adocument\数学建模\24年国赛\CUMCM2024Problems\x1.xlsx' 140. with pd.ExcelWriter(file\_path, engine='xlsxwriter') as writer: 141. for i in range(nt): 142. x\_df = pd.DataFrame(x\_np[i], columns=[f'j\_{j}' for j in range(nj)], index=[f'i\_{i}' for i in range(ni)]) 143. x\_df.to\_excel(writer, sheet\_name=str(2023 + i)) |

|  |
| --- |
| 附录3 |
| 问题一：使用OR-Tools求解模型二线性规划 |
| 1. import numpy as np 2. plant\_data = pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\附件2.xlsx",sheet\_name="2023年的农作物种植情况") 3. plant\_data['种植地块'] = plant\_data['种植地块'].fillna(method='ffill') 4. Statistics\_data = pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\附件2z.xlsx",sheet\_name="2023年统计的相关数据") 5. Statistics\_data = Statistics\_data.iloc[:107] 6. Statistics\_data 7. *# 将plant\_data中的地块编号的第一个字符映射到Statistics\_data的地块类型* 8. type\_mapping = {'A': '平旱地', 'B': '梯田', 'C': '山坡地', 'D': '水浇地', 'E': '普通大棚', 'F': '智慧大棚'} 9. plant\_data['地块类型'] = plant\_data['种植地块'].apply(lambda x: type\_mapping[x[0]]) 10. *# 找到对应的行，并计算产量* 11. merged\_data = pd.merge(plant\_data, Statistics\_data, on=['地块类型', '作物编号']) 12. merged\_data['总产量'] = merged\_data['种植面积/亩'] \* merged\_data['亩产量/斤'] 13. merged\_data[['种植地块', '种植面积/亩', '作物编号', '地块类型', '亩产量/斤', '总产量']] 14. first\_season\_data = Statistics\_data[Statistics\_data['种植季次']=='第一季'] 15. first\_season\_price = {} 16. for \_, row in first\_season\_data.iterrows(): 17. if row['作物编号'] in first\_season\_price: 18. if row['平均单价'] != first\_season\_price[row['作物编号']-16]: 19. raise ValueError(f"作物编号 {row['作物编号']} 对应的平均单价不一致") 20. else: 21. *# 如果作物编号不在字典中，则添加到字典中* 22. first\_season\_price[row['作物编号']-16] = row['平均单价'] 23. first\_season\_price[0]=7 24. first\_season\_price 25. second\_season\_price={} 26. second\_season\_data = Statistics\_data[Statistics\_data['种植季次']=='第二季'] 27. for \_, row in second\_season\_data.iterrows(): 28. price\_key = row['作物编号'] - 16 29. if second\_season\_price.get(price\_key) is not None: 30. if row['平均单价'] != second\_season\_price[price\_key]: 31. raise ValueError(f"作物编号 {row['作物编号']} 对应的平均单价不一致") 32. else: 33. *# 如果作物编号不在字典中，则添加到字典中* 34. second\_season\_price[price\_key] = row['平均单价'] 35. second\_season\_price[0]=0 36. array\_size = 26 37. first\_season= np.zeros((array\_size)) 38. for key, value in first\_season\_price.items(): 39. if key <= array\_size:  *# 确保索引在数组范围内* 40. first\_season[key ] = value  *# 键-1作为索引* 41. second\_season= np.zeros((array\_size)) 42. for key, value in second\_season\_price.items(): 43. if key <= array\_size:  *# 确保索引在数组范围内* 44. second\_season[key ] = value  *# 键-1作为索引* 45. sc={} 46. sc[0]=first\_season 47. sc[1]=second\_season 48. sc[1] 49. Exp=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\exp.xlsx") 50. Exp=Exp.drop(columns=['作物编号']) 51. bean\_class\_indices=[crop\_class['是否豆类'].iloc[15+i] for i in range(41-15)] 52. bean\_class\_indices = [i for i, value in enumerate(bean\_class\_indices) if value == 1] 53. bean\_class\_indices 54. muchan=market[market['地块类型'].isin(['水浇地', '普通大棚', '智慧大棚'])] 55. result = muchan[['作物编号', '地块类型', '种植成本/(元/亩)']].drop\_duplicates().reset\_index() 56. result=result.drop(columns=['序号']) 57. merged\_data = pd.merge(one\_area, result, on='地块类型') 58. pivot\_data = merged\_data.pivot\_table(index='种植地块', columns='作物编号', values='种植成本/(元/亩)', aggfunc='first').reset\_index() 59. bc=pivot\_data.fillna(1000000).drop(columns='种植地块') 60. Exp=[Exp.iloc[i+15,0] for i in range(41-15)] 61. area=one\_area['地块面积/亩'] 62. bc 63. from ortools.linear\_solver import pywraplp 64. ni=28 65. nj=41-15 66. nt=14 67. solver = pywraplp.Solver.CreateSolver('SCIP') 68. *# solver.SetSolverSpecificParametersAsString('log\_level=2')  # 2 代表详细日志级别* 69. y = [[[solver.IntVar(0, 1, f'y\_{t}\_{i}\_{j}') for j in range(nj)] for i in range(ni)] for t in range(nt)] 70. x = [[[solver.NumVar(0, 40, f'x\_{t}\_{i}\_{j}') for j in range(nj)] for i in range(ni)] for t in range(nt)] 71. theta=0.5 72. S\_T = [solver.NumVar(0, solver.infinity(), f'S\_T[{T}]') for T in range(nt//2)] 73. B\_T = [solver.NumVar(0, solver.infinity(), f'B\_T[{T}]') for T in range(nt // 2)] 74. output = [[solver.NumVar(0, solver.infinity(), f'output[{T}]') for j in range(nj)] for T in range(nt//2 )] 75. remain = [[solver.NumVar(0, solver.infinity(), f'remain[{j}]') for j in range(nj)] for T in range(nt//2)] 76. z1 = [[solver.IntVar(0, 1, f'output[{T}]') for j in range(nj)] for T in range(nt//2 )] 77. z2 = [[solver.IntVar(0, 1, f'remain[{j}]') for j in range(nj)] for T in range(nt//2)] 78. for T in range(nt // 2): 79. B\_T[T] = solver.Sum( 80. solver.Sum(bc.iloc[i,j] \* x[T][i][j] for i in non\_smart) + 81. solver.Sum(bc.iloc[i,j]  \* x[2\*T] [i][j] for i in smart) 82. for j in range(nj) 83. ) 84. for T in range(nt//2): 85. sum\_output = solver.Sum([sc[0][j] \* output[T][j] for j in range(nj)]) 86. sum\_remain = theta\*solver.Sum([sc[0][j] \* remain[T][j] for j in range(nj)]) 87. for j in range(nj): 88. *# 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij}* 89. output\_constraint\_1 = solver.Sum(x[T][i][j] \* pro1[0].iloc[i,j] for i in non\_smart) 90. output\_constraint\_2 = solver.Sum(x[2\*T] [i][j]\* pro1[0].iloc[i,j] for i in smart) 91. solver.Add(output[T][j] <= output\_constraint\_1+output\_constraint\_2) 92. *# 约束3: output\_{t}\_{j} <= exp\_{j}* 93. solver.Add(output[T][j] <= Exp[j]) 94. m=1000000 95. b=output\_constraint\_1+output\_constraint\_2- Exp[j] 96. c=0 97. solver.Add(remain[T][j]<=b+m\*z1[T][j]) 98. solver.Add(remain[T][j]<=c+m\*z2[T][j]) 99. solver.Add(remain[T][j]>=b) 100. solver.Add(remain[T][j]>=c) 101. solver.Add(z1[T][j]+z2[T][j]==1) 102. S\_T[T]=sum\_output+sum\_remain 103. output1 = [[solver.NumVar(0, solver.infinity(), f'output[{T}]') for j in range(nj)] for T in range(nt//2)] 104. remain1 = [[solver.NumVar(0, solver.infinity(), f'remain[{j}]') for j in range(nj)] for T in range(nt//2)] 105. B = [solver.NumVar(0, solver.infinity(), f'B\_{t}') for t in range(nt//2)] 106. S = [solver.NumVar(0, solver.infinity(), f'S\_{t}') for t in range(nt//2)] 107. z11 = [[solver.NumVar(0, 1 , f'output[{T}]') for j in range(nj)] for T in range(nt//2)] 108. z22 = [[solver.NumVar(0, 1, f'remain[{j}]') for j in range(nj)] for T in range(nt//2)] 109. for t in range(nt//2): 110. sum\_output = solver.Sum([sc[1][j] \* output1[t][j] for j in range(nj)]) 111. sum\_remain = theta\*solver.Sum([sc[1][j] \* remain1[t][j] for j in range(nj)]) 112. for j in range(nj): 113. *# 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij}* 114. output\_constraint\_2 = solver.Sum(x[2\*t+1][i][j]\* pro1[1].iloc[i,j] for i in smart) 115. solver.Add(output1[t][j] <= output\_constraint\_2) 116. *# 约束3: output\_{t}\_{j} <= exp\_{j}* 117. solver.Add(output1[t][j] <= Exp[j]) 118. b=output\_constraint\_2- Exp[j] 119. c=0 120. m=1000000 121. solver.Add(remain1[t][j]<=b+m\*z11[t][j]) 122. solver.Add(remain1[t][j]<=c+m\*z22[t][j]) 123. solver.Add(remain1[t][j]>=b) 124. solver.Add(remain1[t][j]>=c) 125. solver.Add(z11[t][j]+z22[t][j]==1) 126. S[t] =sum\_output+sum\_remain 127. for t in range(nt//2): 128. B[t] = solver.Sum(bc.iloc[i,j] \* x[2\*t+1][i][j] for i in smart for j in range(nj)) 129. for T in range(nt//2): 130. for i in non\_smart: 131. non\_smart\_sum = solver.Sum(x[T][i][j] for j in range(nj)) 132. *# 添加约束* 133. solver.Add(non\_smart\_sum <= area.iloc[i]) 134. for t in range(nt):  *# t = 3, 5, ..., 15* 135. for i in smart: 136. smart\_sum\_t = solver.Sum(x[t][i][j] for j in range(nj)) 137. solver.Add(smart\_sum\_t <= area.iloc[i]) 138. *# # 三年种豆* 139. for t in range(4, nt): 140. if t==4: 141. for i in smart: 142. bean\_sum = solver.Sum(x[t][i][j] + x[t-1][i][j] + x[t-2][i][j] + x[t-3][i][j] + x[t-4][i][j] + x0.iloc[i,j] 143. for j in bean\_class\_indices) 144. solver.Add(bean\_sum >= area.iloc[i]) 145. else: 146. for i in smart: 147. bean\_sum = solver.Sum(x[t][i][j] + x[t-1][i][j] + x[t-2][i][j] + x[t-3][i][j] + x[t-4][i][j] + x[t-5][i][j] 148. for j in bean\_class\_indices) 149. solver.Add(bean\_sum >= area.iloc[i]) 150. for T in range(1, nt//2): 151. if T==1: 152. for i in non\_smart: 153. bean\_sum\_non\_smart = solver.Sum(x[T][i][j] + x[T-1][i][j] + x0.iloc[i,j] for j in bean\_class\_indices) 154. solver.Add(bean\_sum\_non\_smart >= area.iloc[i]) 155. else: 156. for i in non\_smart: 157. bean\_sum\_non\_smart = solver.Sum(x[T][i][j] + x[T-1][i][j] + x[T-2][i][j] for j in bean\_class\_indices) 158. solver.Add(bean\_sum\_non\_smart >= area.iloc[i]) 159. *# 相邻两季不同* 160. for T in range(nt // 2 ): 161. if T==0: 162. for i in non\_smart: 163. for j in [0]: 164. solver.Add(y[T][i][j] + y0.iloc[i,j] <= 1) 165. else: 166. for i in non\_smart: 167. for j in [0]: 168. solver.Add(y[T][i][j] + y[T-1][i][j] <= 1) 169. for t in range(nt): 170. if t==0: 171. for i in smart: 172. for j in range(nj): 173. solver.Add(y[t][i][j] + y0.iloc[i,j]<= 1) 174. else: 175. for i in smart: 176. for j in range(nj): 177. solver.Add(y[t][i][j] + y[t-1][i][j] <= 1) 178. *# x,y的约束* 179. big\_M = 10000 180. for t in range(nt): 181. for i in range(ni): 182. for j in range(nj): 183. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 184. *# 不能太分散* 185. N=5 186. for T in range(nt//2):  *# T = 1, 2, 3, ...* 187. for j in range(nj): 188. non\_smart\_sum\_y = solver.Sum(y[T][i][j] for i in non\_smart) 189. smart\_sum\_y = solver.Sum(y[2\*T] [i][j]for i in smart) 190. solver.Add(non\_smart\_sum\_y + smart\_sum\_y <= N) 191. for t in range(1, nt, 2):  *# t = 1, 3, 5, ...* 192. for j in range(nj): 193. smart\_sum\_y\_t = solver.Sum(y[t][i][j] for i in smart) 194. solver.Add(smart\_sum\_y\_t <= N) 195. M=0.4 196. for T in range(nt // 2):  *# T = 1, 2, 3, ...* 197. for j in range(nj): 198. for i in non\_smart: 199. solver.Add(x[T][i][j] >=area.iloc[i] \*M \* y[T][i][j]) 200. solver.SetTimeLimit(120000) 201. solver.EnableOutput() 202. objective\_expr =  solver.Sum(S\_T[t]-B\_T[t] for t in range(nt//2))+ solver.Sum(S[t]-B[t] for t in range(nt//2)) 203. solver.Maximize(objective\_expr) 204. status = solver.Solve() 205. status 206. remain\_values = [[remain[T][j].solution\_value() for j in range(nj)] for T in range(nt//2)] 207. output\_values=[[output[T][j].solution\_value() for j in range(nj)] for T in range(nt//2)] 208. remain1\_values = [[remain1[T][j].solution\_value() for j in range(nj)] for T in range(nt//2)] 209. output1\_values=[[output1[T][j].solution\_value() for j in range(nj)] for T in range(nt//2)] 210. for T in range(nt//2): 211. sum\_output = sum([sc[0][j] \* output\_values[T][j] for j in range(nj)]) 212. sum\_remain = theta\*sum([sc[0][j] \* remain\_values[T][j] for j in range(nj)]) 213. S\_T[T]=sum\_output+sum\_remain 214. for T in range(nt // 2): 215. B\_T[T] = sum( 216. sum(bc.iloc[i,j] \* x\_np[T][i][j] for i in non\_smart) + 217. sum(bc.iloc[i,j]  \* x\_np[2\*T] [i][j] for i in smart) 218. for j in range(nj) 219. ) 220. for t in range(nt//2): 222. for j in range(nj): 223. output\_constraint\_2 = sum(x\_np[2\*t+1][i][j]\* pro1[1].iloc[i,j] for i in smart) 224. if output\_constraint\_2- Exp[j]>0: 225. print(f'{t}\_{j}    {output\_constraint\_2- Exp[j]}') 226. remain1\_values[t][j]=max((output\_constraint\_2- Exp[j]),0) 227. sum\_output = sum([sc[1][j] \* output1\_values[t][j] for j in range(nj)]) 228. sum\_remain = theta\*sum([sc[1][j] \* remain1\_values[t][j] for j in range(nj)]) 229. S[t] =sum\_output+sum\_remain 230. for t in range(nt//2): 231. B[t] =sum(bc.iloc[i,j] \* x\_np[2\*t+1][i][j] for i in smart for j in range(nj)) 232. for t in range(nt//2): 233. print(S\_T[t]-B\_T[t]+S[t]-B[t]) 234. cucumber=np.zeros((7, 2)) 235. for t in range(nt//2): 236. cucumber[t][0]=output\_values[t][13]+output1\_values[t][13] 237. cucumber[t][1]=remain\_values[t][13]+remain1\_values[t][13] 238. cucumber |

|  |
| --- |
| 附录4 |
| 问题一：使用OR-Tools求解模型三线性规划 |
| 1. from ortools.linear\_solver import pywraplp 2. ni=24 3. nj=41-15 4. nt=7 5. solver = pywraplp.Solver.CreateSolver('SCIP') 6. y = [[[solver.IntVar(0, 1, f'y\_{t}\_{i}\_{j}') for j in range(nj)] for i in range(ni)] for t in range(nt)] 7. x = [[[solver.NumVar(0, 40, f'x\_{t}\_{i}\_{j}') for j in range(nj)] for i in range(ni)] for t in range(nt)] 8. theta=0 9. B = [solver.NumVar(0, solver.infinity(), f'B\_{t}') for t in range(nt)] 10. S = [solver.NumVar(0, solver.infinity(), f'S\_{t}') for t in range(nt)] 11. for t in range(nt): 12. B[t] = solver.Sum(bc.iloc[i,j] \* x[t][i][j] for i in range(ni) for j in range(nj)) 13. output =[[solver.NumVar(0, solver.infinity(), f'output\_{t}\_{j}') for j in range(nj)] for t in range(nt)] 14. remain =[[solver.NumVar(0, solver.infinity(), f'remain\_{t}\_{j}') for j in range(nj)] for t in range(nt)] 15. for t in range(nt): 16. sum\_output = solver.Sum([sc[j] \* output[t][j] for j in range(nj)]) 17. sum\_remain = theta\*solver.Sum([sc[j] \* remain[t][j] for j in range(nj)]) 18. for j in range(nj): 19. *# 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij}* 20. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] for i in range(ni)) 21. solver.Add(output[t][j] <= sum\_pro) 22. *# 约束3: output\_{t}\_{j} <= exp\_{j}* 23. solver.Add(output[t][j] <= Exp[j]) 24. *# 约束4: remain\_{t}\_{j} >= 0* 25. solver.Add(remain[t][j] >= 0) 26. *# 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j}* 27. solver.Add(remain[t][j] >= sum\_pro - Exp[j]) 28. S[t]=sum\_output+sum\_remain 29. for i in range(ni): 30. for t in range(nt): 31. constraint\_expr = solver.Sum(x[t][i][j] for j in range(nj)) 32. solver.Add(constraint\_expr <= area[t][i]) 33. big\_M = 10000 34. for t in range(nt): 35. for i in range(ni): 36. for j in range(nj): 37. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 38. N = 5 39. for j in range(nj): 40. for t in range(nt): 41. *# 创建约束* 42. constraint\_expr = solver.Sum(y[t][i][j] for i in range(ni)) 43. solver.Add(constraint\_expr <= N) 44. M = 0.4 45. for t in range(nt): 46. for j in range(nj): 47. for i in range(ni): 48. *# 创建约束* 49. constraint\_expr = x[t][i][j] >= M \* y[t][i][j] \* area[t][i] 50. solver.Add(constraint\_expr) 51. solver.SetTimeLimit(120000) 52. solver.EnableOutput() 53. objective\_expr = solver.Sum(S[t] for t in range(nt)) - solver.Sum(B[t] for t in range(nt)) 54. solver.Maximize(objective\_expr) 55. status = solver.Solve() 56. status |