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，**滞销处理时，年均最大销售利润为573万，半价销售处理时，年均最大销售利润为814万**。

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，**滞销处理时，年均最大销售利润为592万**。

针对问题三，为构建一个综合考量**农作物互补性与可替代性**的最优种植策略规划模型，首先，通过对经济学核心原理的细致剖析，明确互补性和可替代性的基本概念。而后通过广泛的文献调研和资料搜集，对农作物进行系统分类，并**认定同一类别农作物之间存在的可替代性特征**。此外，从**使用配套、营养互补和消费习惯等多个维度，详细阐述农作物间的互补性关系**。为更精确地评估销售价格和种植成本对农作物预期销售量的影响，引入**Rotterdam需求系统模型**，对农作物的**预期销售量进行修正调整**。通过对特定农作物案例的深入分析，探讨它们之间的互补性和替代性动态，并据此计算预期销售量。最终，运用随机规划模型，求解最优种植方案。研究结果表明，该修正模型能够有效地适应市场变化，对农作物销售量的动态变化作出响应。

关键词：线性 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. 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七年农作物的最佳种植策略。
2. 针对农作物销售量、产量、成本和价格的波动，构建适应性种植模型，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，构建并优化农作物种植策略，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

要求在预期销售量、种植成本、亩产量和销售价格不变的前提下，给出最优种植方案。本文以最大化总销售利润为目标，依据地块类型不同，分别构建了两个子目标规划模型形成独立耕地种植协同优化模型。以不同年份各个地块上不同农作物的种植量为决策变量，考虑种植地面积约束，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，种植地不过于分散，以及种植面积不宜太小等约束条件。

通过引入新的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，将原规划模型中的非线性部分进行线性转化。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

考虑现实农业生产中农作物属性的波动性，认为其不再表现为确定性数值，而是呈现出随机变化。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。

鉴于缺乏具体的历史数据进行分析，本研究采用3σ法则为边界的正态分布以及范围设定的均匀分布来拟合随机变量，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。利用样本均值近似（SAA）方法对所建立的模型进行求解。

## 问题三的分析

为了深入探讨农作物商品间的互补性和可替代性，首先依据经济学原理其概念进行详细的解析。在概念界定清晰之后，通过文献调研和资料搜集，对本文所涉及的农作物进行了分类，并且我们认定同一类别的农作物之间存在着一定的可替代性。同时，基于使用配套、营养补充以及消费习惯阐述农作物间的互补性。

为了精确研究农作物之间的关系以及销售价格对预期销售量的影响，本文采用Rotterdam需求系统模型对农作物的预期销售量进行了细致的修正。随后，通过选取特定的农作物案例，分析它们之间的互补性和替代性关系，并计算相应的预期销售量。最终，利用随机规划模型，求解最优种植方案。

# 模型假设

1.假设2023年产出的农作物种植预先合理规划，恰好全部售出；

2.假设题中“当季”特指第一季、第二季，不考虑具体月份差异；

3.参考2023年数据，假设智能大棚的第二季每种农作物预期销售量均为0.3亩；

4.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

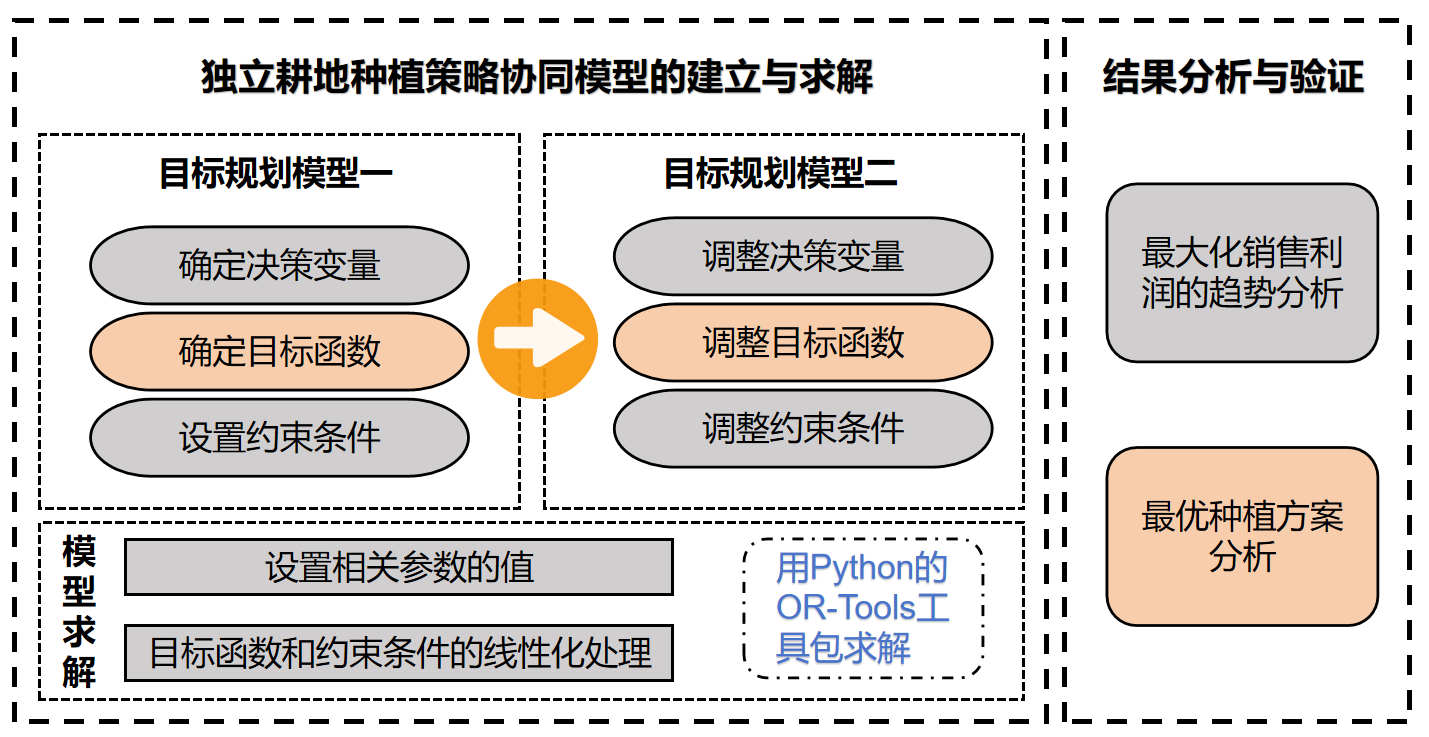


图 1 问题一研究框架图

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。另外，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。基于假设1，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，考虑不同地块类型的种植作物的完全差异，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。

### 目标规划模型一的建立

我们首先考虑在平旱地、梯田和山坡地的种植规划问题。

* **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最好，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浇地、普通大棚和智慧大棚中的种植规划问题。

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。

* **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，无法精确估计单价。进一步考虑实际售价平稳波动性，本文决定采用该区间的平均单价作为模型输入。

|  |  |
| --- | --- |
|  | (15) |

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。

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

* 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：

|  |  |
| --- | --- |
|  | (7) |

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：

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

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：

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

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：

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

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：

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

* 目标函数的线性转化

基于最大值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：

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

* 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。基于假设2，认为其种植面积均为0.3亩。

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2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。通过初步观察种植方案，可以发现：

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。

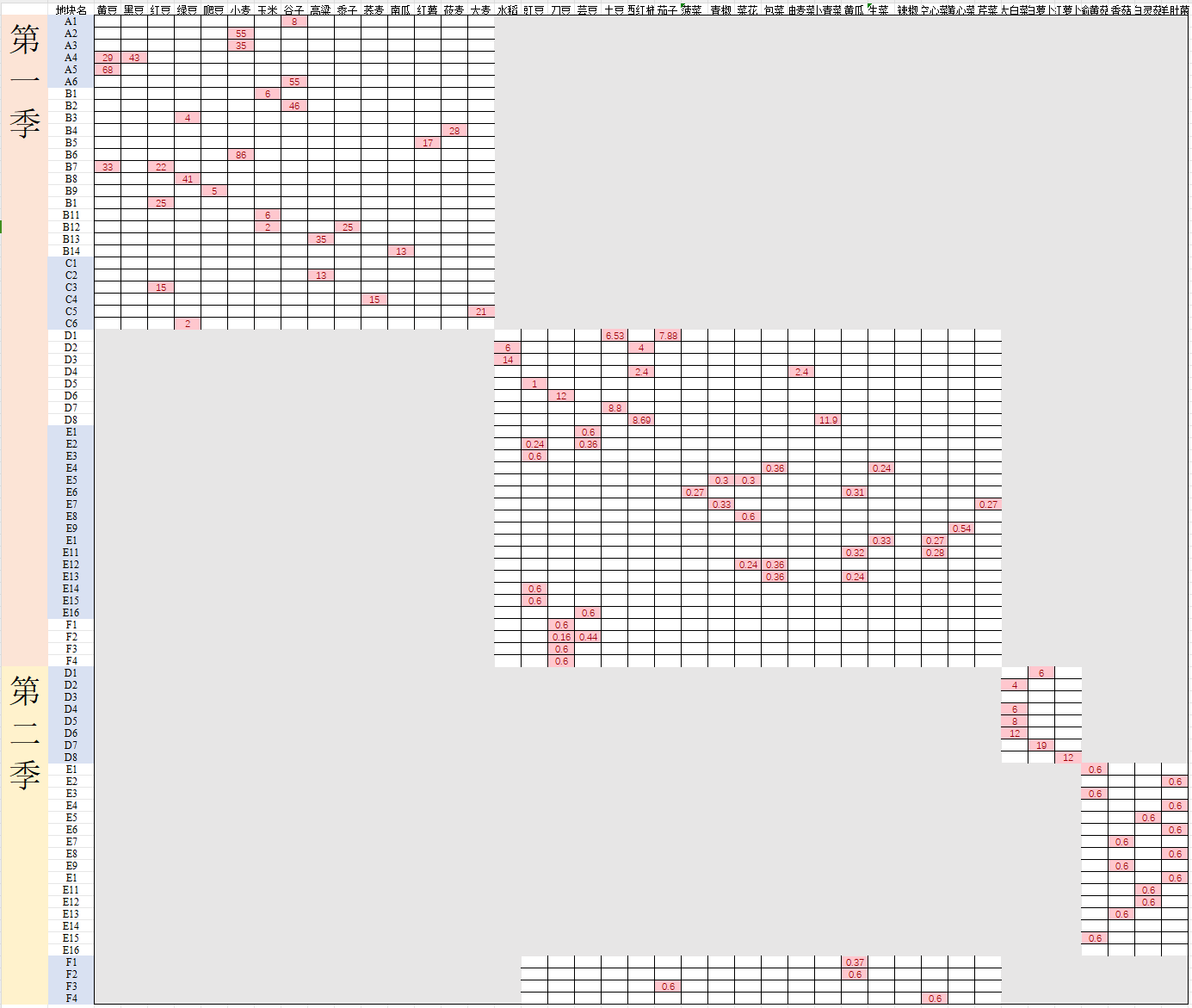
综上所述，本耕地种植协同优化模型在农作物种植安排上表现出卓越的性能。

图 2 最优种植安排(2024年)

* 最大销售利润对比分析

对超出预期销售量的部分作滞销处理时的最优种植策略下的最大利润如图3所示：

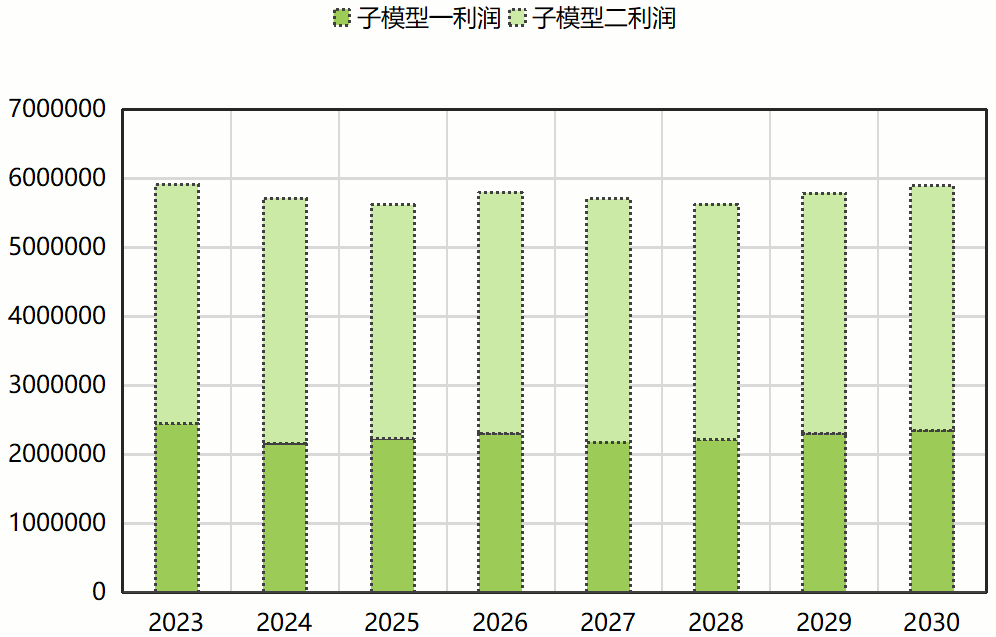


图 3 最大化利润值（一）

其中，子模型一利润即在平旱地、梯田和山坡地上种植农作物所获得的利润，子模型二利润即在水浇地、普通大棚和智慧大棚上种植农作物所获得的利润。从求解出的最大利润统计图可以得出以下结论：

1. 在对比模型一和模型二所求得的最优种植策略时，我们发现其**最大利润水平与2023年的实际利润数据大致吻合**，且随时间推移呈现出微小的波动。这一发现表明，所构建的**优化模型能够较好地反映该地区农作物的实际种植与销售状况**。具体而言，以下两点进一步支持了这一结论：一方面，2023年的种植方案实现了所有地块的充分利用，且所有农产品均实现了销售，这表明在保持预计销售量不变的前提下，2023年的利润已达到历史最高水平，因此可以认为2023年的利润代表了在当前市场条件下的最大潜在利润；另一方面，由于模型一和模型二求得的最大利润与2023年的实际利润水平相近，且波动幅度较小，这为验证我们的优化模型求解结果提供了强有力的证据。由此可以看出所构建的**独立耕地种植协同优化模型不仅能够求解出接近实际的最优解，而且能够为种植者提供最优的种植方案**。
2. 在分析2024至2030年的总种植利润时，观察到**这些年份的利润总额均略低于2023年的水平**。这一现象可以归因于以下两个主要因素：第一，鉴于2023年实现了对所有种植地块的充分利用，并且所有农产品均成功销售，这一年的利润达到了一个较高的基准，在随后的年份中，为了防止连续重茬种植，种植结构必然会发生调整。这种调整可能导致某些地块的种植面积减少，从而**影响了整体的种植密度**；第二，为了满足种植集中度的约束条件，部分地块可能需要预留出较小的面积以适应特定的种植要求，这种预留面积的增加虽然不会显著改变总种植面积，但会导致**地块的整体利用率相对2023年有所下降**。因此，考虑到这些策略性的调整和种植效率的微小降低，2024至2030年总种植利润略低于2023年的结果**在逻辑上是合理的，并且与农业生产实践中常见的土地管理和种植策略相符合**。

对超出预期销售量的部分作降价出售处理时的最优种植策略下的最大利润如图4所示：

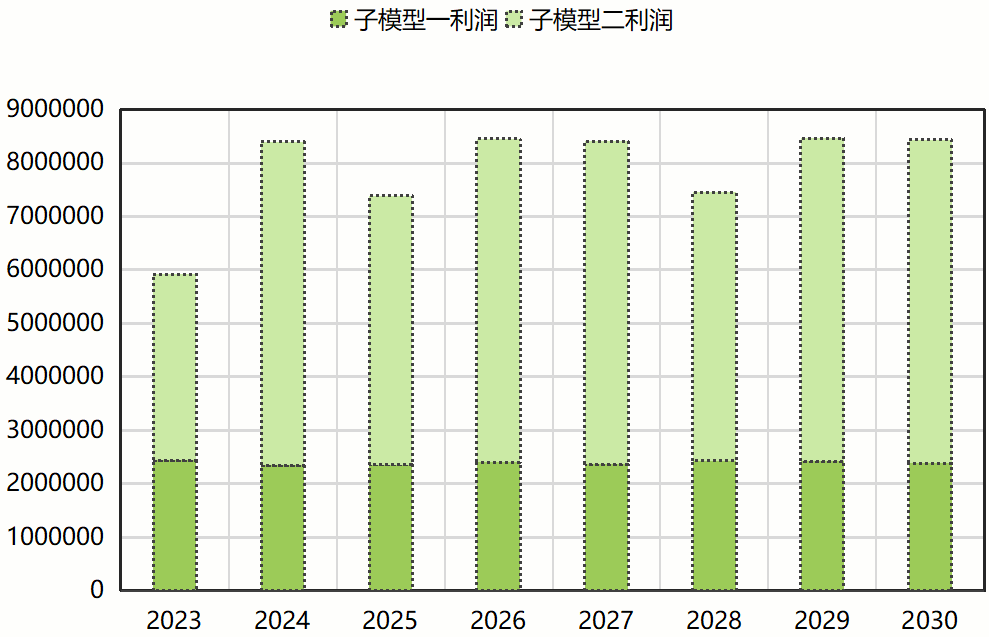


图 4 最大化利润值（二）

从求解出的最大利润统计图可以得出以下结论：

1. 分析结果显示，即便对超出预期销售量的产品实施半价销售策略，**平旱地、梯田和山坡地的最大利润依然与2023年的水平保持接近**。通过深入审查最优种植配置和作物收益数据，我们发现这些地块适宜种植的**作物单位面积利润普遍较低**，大致介于1000至3000元/亩。即便我们倾向于增加单位利润较高的作物种植比例，由于半价销售导致的单位利润下降，其数值已接近原种植方案中单位利润最低的作物。这一发现表明，对于模型一中的作物组合而言，**实施半价销售策略极其不利**。因此，**在重新规划种植策略时，并未出现显著倾向于改变原种植方案的趋势**，且所获得的最大利润与2023年的水平相近，这一结果在逻辑上是合理的，并反映了原种植方案的可持续性和经济性。
2. 在对2024至2030年的水浇地、普通大棚和智慧大棚实施超出预期销售量产品半价销售策略的情况下，**最优种植策略下的最大利润显著超过了2023年的水平，增幅最高可达约250万元**。通过对此时的种植方案及作物收益数据的深入分析，我们发现**黄瓜的种植量相较于采用滞销策略时大幅增加**，其预期销售量从26100斤上升至实际种植安排中的最高85万斤。这部分黄瓜的半价销售贡献了约250万元的利润。这一种植安排的合理性体现在以下几个方面：第一，**黄瓜的单位利润相较于其他作物较高**，最高可达101500元/亩，最低为81100元/亩，即便在半价销售的情况下，其单位利润仍远超多数农作物；第二，**黄瓜在第一季和第二季的适宜种植面积较大**，这为增加黄瓜的种植量提供了空间。因此，**种植方案的调整倾向于增加黄瓜的种植量**，且增加黄瓜的种植量是合理的，这也解释了为何在实施半价销售策略后，最大利润能够较大超过2023年的水平。
3. 在2024至2030年的种植方案中，**观察到2025年和2028年的最大利润与前一年相比显著下降了近100万元**。分析可知，这种利润下降是**为了遵守三年内至少种植一次豆类的土地轮作约束条件**。具体而言，在这两年中，原本计划用于种植黄瓜的较大面积土地被调整为种植豆类，这一调整旨在维持土壤的营养平衡，并促进农业种植的可持续发展，与实际情况相符。

从每年的最大化利润和具体的种植安排来看，该独立耕地种植协同优化模型能够安排符合实际情况的最优种植策略，模型性能较好。

## 问题二随机规划模型的建立与求解

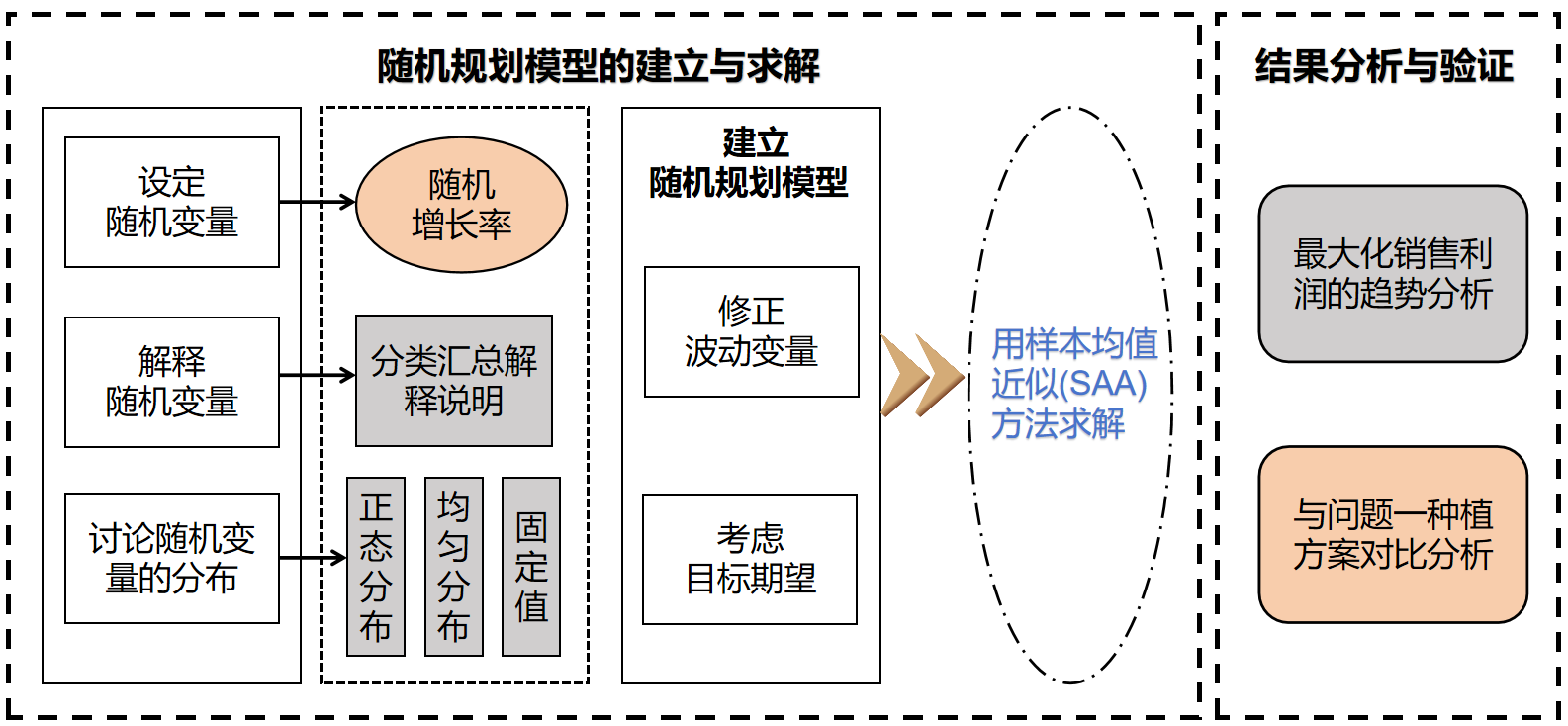


图 5 问题二研究框架图

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%。

### 随机变量的分布确定

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：

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

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：

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

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：

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

### 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：

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

其中，表示第n次的采样值。

此时相应的目标函数转化为：

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

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，在运用SAA方法求解最优种植策略下获得的最大利润如表2所示，具体种植安排见附件“result2.xlsx”。

表 2 问题二最大利润

|  |  |  |  |
| --- | --- | --- | --- |
|  | 子模型一利润 | 子模型二利润 | 总利润 |
| 2023 | 2446601 | 3472472 | 5919073 |
| 2024 | 2362917 | 3567132 | 5930049 |
| 2025 | 2148294 | 3532182 | 5680477 |
| 2026 | 1956957 | 3729521 | 5686478 |
| 2027 | 2132998 | 3816690 | 5949688 |
| 2028 | 1990045 | 3817647 | 5807692 |
| 2029 | 2008765 | 4058974 | 6067740 |
| 2030 | 2145468 | 4180648 | 6326116 |

## 问题三模型的建立与求解

图 6 问题三研究框架图

### 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：

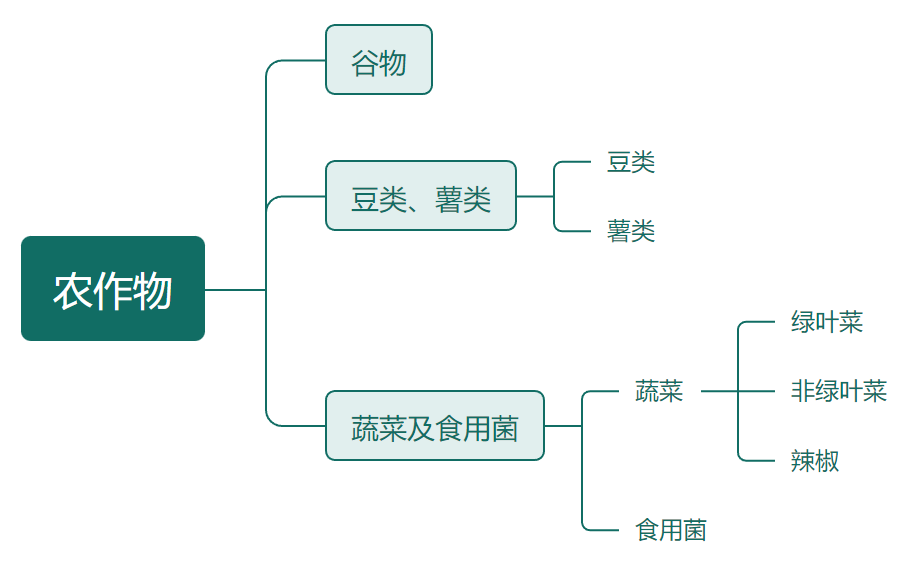


图 7 农作物分类

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：

|  |  |
| --- | --- |
|  | (28) |

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，即在不考虑农作物价格的修正下的初始值；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：

|  |  |
| --- | --- |
|  | (29) |

### 随即规划模型的修正与求解

由于本问缺少农作物的历史种植和销售数据，且没有合理的依据以供我们进行数据模拟，因此我们难以真正去拟合Rotterdam需求系统模型的参数数据。在此，我们为了展示该模型的求解过程，考虑农作物之间的关系，并查阅相关文献，自主对参数进行较为合理的设定。

基于问题二的随机规划模型，本问中的随机规划对预期销售量的计算运用Rotterdam需求系统模型进行了修正。其具体计算公式如下所示：

|  |  |
| --- | --- |
|  | (30) |

在此，根据需求系统模型原理，我们可以知道在不受农作物的价格影响时，

|  |  |
| --- | --- |
|  | (31) |

为了方便模型的求解，在此我们将预期销售量的增长率固定为7.5%。进一步联立公式(30)(31)，预期销售量的计算公式化简为：

|  |  |
| --- | --- |
|  | (32) |

在分析农作物的预期销售量时，我们选取小麦、水稻、玉米和黄瓜作为代表，并重点考虑这些作物之间的互补性或替代性关系。基于实际情况，我们可以得出以下结论：小麦、玉米和水稻均属于粮食作物，它们在功能上具有相似性，均能作为主食来源，因此，这些作物之间可以被视为互为替代品。另一方面，黄瓜作为一种蔬菜，其与粮食作物在营养和功能上存在互补性，因此，黄瓜可以分别与小麦、水稻、玉米定义为互补作物。基于此，并参考文献[]，设定交叉价格弹性参数的值如表3所示：

表 3 交叉价格弹性系数设定

|  | 小麦 | 玉米 | 蔬菜 | 水稻 |
| --- | --- | --- | --- | --- |
| 小麦 | -0.15 | 0.1 | -0.05 | 0.1 |
| 玉米 | 0.1 | -0.14 | -0.06 | 0.1 |
| 蔬菜 | 0.02 | 0.1 | -0.15 | 0.03 |
| 水稻 | 0.1 | 0.1 | -0.07 | -0.13 |

最后基于问题二随机规划模型进行求解。问题二、问题三计算所得最优策略下小麦、水稻、玉米和黄瓜每年的销售量如下图8所示：

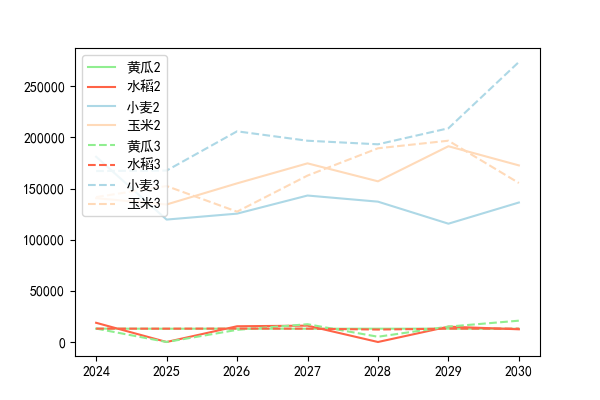


图 8 问题二、三求解结果对比

通过对问题二和问题三最优种植方案中农作物销售量变化趋势的比较分析，我们可以观察到以下现象：在问题三的方案中，互为替代品的小麦和玉米的销售量表现出更为显著的相反变化趋势。以小麦为例，当小麦价格上涨时，其预期销售量呈现出下降的趋势，相应地，其替代品玉米的销量则有所增加。另一方面，互为互补品的黄瓜和小麦的销售量变化趋势一致，即小麦价格上涨导致其销售量下降，黄瓜的销售量也随之降低。然而，可能由于黄瓜和小麦之间的互补性相对较弱，这种趋势变化并不显著。

这一分析结果表明，基于预期销售量修正的随机规划模型能够更有效地考虑农作物之间潜在的互补性或替代性关系对销售量的影响。该模型的应用有助于更准确地预测和评估市场变化对农作物销售量的动态响应。

# 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。

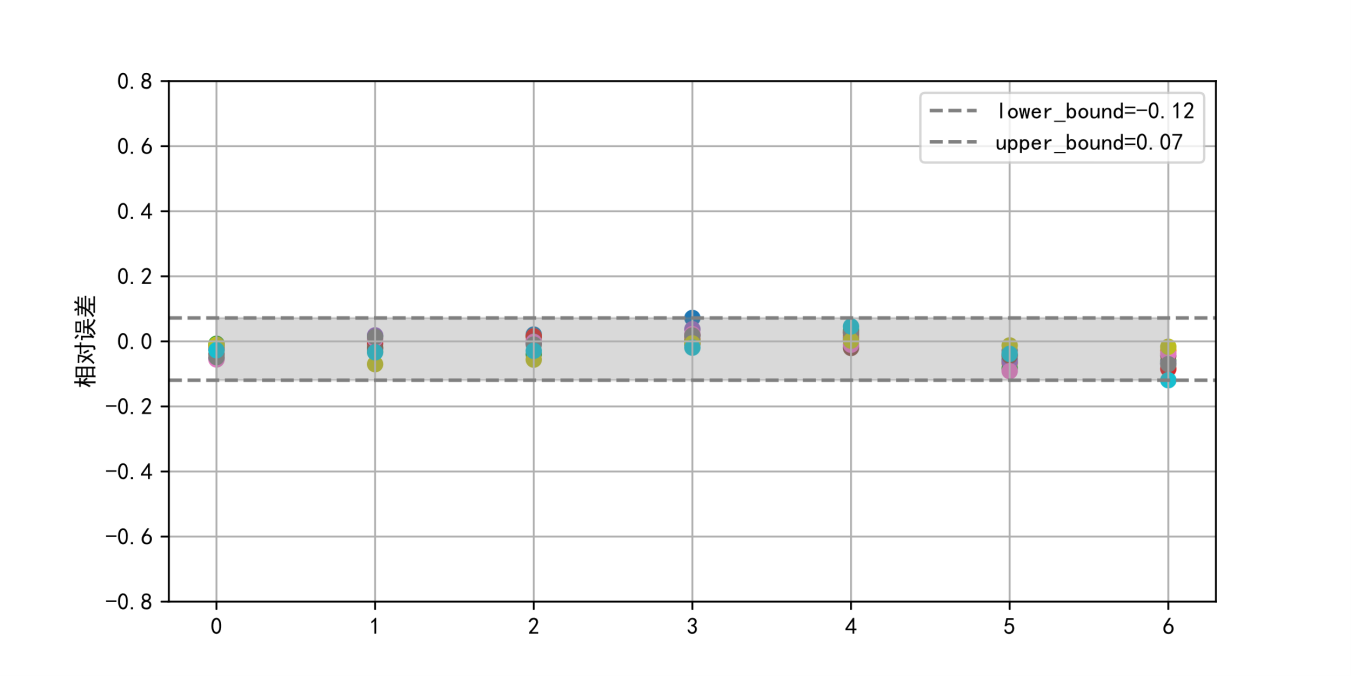


图 9 利润波动范围图

图9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 9. data2=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\附件2.xlsx",sheet\_name=0) 10. data2 = data2.fillna(method='ffill') 11. 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'} 12. data2['地块类型'] = data2['种植地块'].apply(**lambda** x: type\_mapping[x[0]]) 13. crop=data2[['种植地块','作物编号','作物名称','作物类型']] 15. crop['是否豆类'] = crop['作物类型'].apply(**lambda** x: 1 **if** '豆类' **in** x **else** 0) 17. crop\_class=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\附件1.xlsx",sheet\_name=1) 18. crop\_class=crop\_class.drop(columns=['说明','种植耕地']) 20. crop\_class['是否豆类'] = crop\_class['作物类型'].apply(**lambda** x: 1 **if** '豆类' **in** x **else** 0) 21. crop\_class 22. 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) 24. muchan=market[['作物编号','地块类型','亩产量/斤']] 25. crop1=data2[data2['地块类型'].isin(['平旱地', '梯田','山坡地'])] 26. 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) 27. y0=x0.copy() 28. y0[y0>0]=int(1) 29. one\_area=area\_class[area\_class['地块类型'].isin(['平旱地', '梯田','山坡地'])] 30. # one\_area.set\_index('种植地块',inplace=True) 31. # 创建一个作物编号列的表，将作物编号作为列名 32. 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() 33. pro= pd.merge(one\_area['地块类型'],one\_muchan, on='地块类型', how='left').drop(columns=['地块类型']) 34. b=market[market['地块类型']=='平旱地'][['种植成本/(元/亩)']] 35. s=market[market['地块类型']=='平旱地'][[ '种植成本/(元/亩)']] 36. Exp=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\exp.xlsx") 37. Exp=Exp.drop(columns=['作物编号']) 38. sc=pd.read\_excel(r"D:\adocument\数学建模\24年国赛\CUMCM2024Problems\s.xlsx") 39. bean\_class\_indices=crop\_class['是否豆类'].iloc 40. bean\_class\_indices = [i **for** i, value **in** enumerate(bean\_class\_indices[:15]) **if** value == 1] 42. bc=b['种植成本/(元/亩)'] 43. sc=sc['平均单价'].iloc[:15] 44. pro.iloc[0,0] 46. area=one\_area['地块面积/亩'].iloc[:] 47. Exp.iloc[0,0] 48. **from** ortools.linear\_solver **import** pywraplp 49. ni=26 50. nj=15 51. nt=7 52. solver = pywraplp.Solver.CreateSolver('CP-SAT') 53. # solver.SetSolverSpecificParametersAsString('log\_level=2')  # 2 代表详细日志级别 54. 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)] 55. 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)] 57. theta=0 58. # b = [solver.NumVar(0, solver.infinity(), f'b\_{j}') for j in range(nj)] 59. B = [solver.NumVar(0, solver.infinity(), f'B\_{t}') **for** t **in** range(nt)] 60. S = [solver.NumVar(0, solver.infinity(), f'S\_{t}') **for** t **in** range(nt)] 61. **for** t **in** range(nt): 62. B[t] = solver.Sum(bc.iloc[j] \* x[t][i][j] **for** i **in** range(ni) **for** j **in** range(nj)) 63. output =[[solver.NumVar(0, solver.infinity(), f'output\_{t}\_{j}') **for** j **in** range(nj)] **for** t **in** range(nt)] 64. remain =[[solver.NumVar(0, solver.infinity(), f'remain\_{t}\_{j}') **for** j **in** range(nj)] **for** t **in** range(nt)] 65. z1=[[solver.IntVar(0,1 , f'output\_{t}\_{j})') **for** j **in** range(nj)] **for** t **in** range(nt)] 66. z2=[[solver.IntVar(0,1 , f'output\_{t}\_{j})') **for** j **in** range(nj)] **for** t **in** range(nt)] 68. **for** t **in** range(nt): 69. sum\_output = solver.Sum([sc[j] \* output[t][j] **for** j **in** range(nj)]) 70. sum\_remain = theta\*solver.Sum([sc[j] \* remain[t][j] **for** j **in** range(nj)]) 71. **for** j **in** range(nj): 72. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 73. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] **for** i **in** range(ni)) 74. solver.Add(output[t][j] <= sum\_pro) 75. # b=sum\_pro-Exp.iloc[j,0] 76. # M=1000000 77. # solver.Add(remain[t][j] >= 0) 78. # # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 79. # solver.Add(remain[t][j] >= sum\_pro - Exp.iloc[j,0]) 80. # solver.Add(b>=-M\*(1-z1[t][j])) 81. # solver.Add(remain[t][j]<=M\*z1[t][j]) 82. solver.Add(output[t][j] <= Exp.iloc[j,0]) 83. solver.Add(remain[t][j]<=sum\_pro+1000\*z1[t][j]) 84. solver.Add(remain[t][j]<= 0+1000\*z2[t][j]) 85. solver.Add(remain[t][j]>=sum\_pro-Exp.iloc[j,0]) 86. solver.Add(remain[t][j]>=0) 87. solver.Add(z1[t][j]+z2[t][j]==1) 88. S[t]=sum\_output+sum\_remain 89. # for i, idx in enumerate(x0.index): 90. #     for j, col in enumerate(x0.columns): 91. #         x[0][i][j] = x0.loc[idx, col] 92. # for i, idx in enumerate(x0.index): 93. #     for j, col in enumerate(x0.columns): 94. #         y[0][i][j] = y0.loc[idx, col] 95. **for** i **in** range(ni): 96. **for** t **in** range(nt): 97. constraint\_expr = solver.Sum(x[t][i][j] **for** j **in** range(nj)) 98. solver.Add(constraint\_expr <= area[i]) 99. **for** t **in** range(nt): 100. **if** t==0: 101. **for** i **in** range(ni): 102. **for** j **in** range(nj): 103. solver.Add(y[t][i][j] + y0.iloc[i,j] <= 1) 104. **else**: 105. **for** i **in** range(ni): 106. **for** j **in** range(nj): 107. solver.Add(y[t][i][j] + y[t -1][i][j] <= 1) 108. big\_M = 10000 109. **for** t **in** range(nt): 110. **for** i **in** range(ni): 111. **for** j **in** range(nj): 112. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 113. **for** i **in** range(ni): 114. **for** t **in** range(1, nt): 115. **if**(t==1): 116. # 创建约束 117. constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x0.iloc[i,j] **for** j **in** bean\_class\_indices) 118. solver.Add(constraint\_expr >= area[i]) 119. **else**: 120. # 创建约束 121. constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x[t-2][i][j] **for** j **in** bean\_class\_indices) 122. solver.Add(constraint\_expr >= area[i]) 124. N = 3 125. **for** j **in** range(nj): 126. **for** t **in** range(nt): 127. # 创建约束 128. constraint\_expr = solver.Sum(y[t][i][j] **for** i **in** range(ni)) 129. solver.Add(constraint\_expr <= N) 130. M = 0.3 132. **for** t **in** range(nt): 133. **for** j **in** range(nj): 134. **for** i **in** range(ni): 135. # 创建约束 136. constraint\_expr = x[t][i][j] >= M \* y[t][i][j] \* area[i] 137. solver.Add(constraint\_expr) 138. solver.SetTimeLimit(120000) 139. solver.EnableOutput() 140. objective\_expr = solver.Sum(S[t] **for** t **in** range(nt)) - solver.Sum(B[t] **for** t **in** range(nt)) 141. solver.Maximize(objective\_expr) 142. status = solver.Solve() 143. status 144. # 输出结果 145. **import** numpy as np 146. **if** status == pywraplp.Solver.OPTIMAL: 147. **print**('Optimal solution found.') 148. **print**('Solution:') 149. **for** t **in** range(nt): 150. **print**(S[t].solution\_value(),B[t].solution\_value(),S[t].solution\_value()-B[t].solution\_value()) 151. 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)]) 152. 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)]) 153. # x\_np= np.round(x\_np) 154. **elif** status == pywraplp.Solver.INFEASIBLE: 155. **print**('Problem is infeasible.') 156. **elif** status == pywraplp.Solver.UNBOUNDED: 157. **print**('Problem is unbounded.') 158. **elif** status == pywraplp.Solver.ABNORMAL: 159. **print**('Solver encountered an abnormal termination.') 160. **else**: 161. **print**(status) 162. **for** t **in** range(nt): 163. **print**(S[t].solution\_value(),B[t].solution\_value(),S[t].solution\_value()-B[t].solution\_value()) 164. # print(f'B\_{t} = {B[t].solution\_value()}') 165. # print(f'S\_{t}-B\_{t} = {S[t].solution\_value()-B[t].solution\_value()}') 166. **print**('x values:') 167. 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)]) 168. 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)]) 169. file\_path = r'D:\adocument\数学建模\24年国赛\CUMCM2024Problems\x10\_5.xlsx' 170. with pd.ExcelWriter(file\_path, engine='xlsxwriter') as writer: 171. **for** i **in** range(nt): 172. 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)]) 173. 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. plant\_data 5. 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") 6. Statistics\_data = Statistics\_data.iloc[:107] 7. Statistics\_data 8. # 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 9. 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'} 10. plant\_data['地块类型'] = plant\_data['种植地块'].apply(**lambda** x: type\_mapping[x[0]]) 12. # 找到对应的行，并计算产量 13. merged\_data = pd.merge(plant\_data, Statistics\_data, on=['地块类型', '作物编号']) 14. merged\_data['总产量'] = merged\_data['种植面积/亩'] \* merged\_data['亩产量/斤'] 16. 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']] 17. first\_season\_data = Statistics\_data[Statistics\_data['种植季次']=='第一季'] 18. first\_season\_price = {} 19. **for** \_, row **in** first\_season\_data.iterrows(): 20. **if** row['作物编号'] **in** first\_season\_price: 21. **if** row['平均单价'] != first\_season\_price[row['作物编号']-16]: 22. **raise** ValueError(f"作物编号 {row['作物编号']} 对应的平均单价不一致") 23. **else**: 24. # 如果作物编号不在字典中，则添加到字典中 25. first\_season\_price[row['作物编号']-16] = row['平均单价'] 27. first\_season\_price[0]=7 28. first\_season\_price 29. second\_season\_price={} 30. second\_season\_data = Statistics\_data[Statistics\_data['种植季次']=='第二季'] 31. **for** \_, row **in** second\_season\_data.iterrows(): 32. price\_key = row['作物编号'] - 16 33. **if** second\_season\_price.get(price\_key) **is** **not** None: 34. **if** row['平均单价'] != second\_season\_price[price\_key]: 35. **raise** ValueError(f"作物编号 {row['作物编号']} 对应的平均单价不一致") 36. **else**: 37. # 如果作物编号不在字典中，则添加到字典中 38. second\_season\_price[price\_key] = row['平均单价'] 39. second\_season\_price[0]=0 40. array\_size = 26 41. first\_season= np.zeros((array\_size)) 42. **for** key, value **in** first\_season\_price.items(): 43. **if** key <= array\_size:  # 确保索引在数组范围内 44. first\_season[key ] = value  # 键-1作为索引 45. second\_season= np.zeros((array\_size)) 46. **for** key, value **in** second\_season\_price.items(): 47. **if** key <= array\_size:  # 确保索引在数组范围内 48. second\_season[key ] = value  # 键-1作为索引 50. sc={} 51. sc[0]=first\_season 52. sc[1]=second\_season 53. sc[1] 54. **from** ortools.linear\_solver **import** pywraplp 55. ni=28 56. nj=41-15 57. nt=14 58. solver = pywraplp.Solver.CreateSolver('SCIP') 59. # solver.SetSolverSpecificParametersAsString('log\_level=2')  # 2 代表详细日志级别 60. 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)] 61. 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)] 63. theta=0.5 65. S\_T = [solver.NumVar(0, solver.infinity(), f'S\_T[{T}]') **for** T **in** range(nt//2)] 66. B\_T = [solver.NumVar(0, solver.infinity(), f'B\_T[{T}]') **for** T **in** range(nt // 2)] 67. output = [[solver.NumVar(0, solver.infinity(), f'output[{T}]') **for** j **in** range(nj)] **for** T **in** range(nt//2 )] 68. remain = [[solver.NumVar(0, solver.infinity(), f'remain[{j}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)] 69. z1 = [[solver.IntVar(0, 1, f'output[{T}]') **for** j **in** range(nj)] **for** T **in** range(nt//2 )] 70. z2 = [[solver.IntVar(0, 1, f'remain[{j}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)] 71. **for** T **in** range(nt // 2): 72. B\_T[T] = solver.Sum( 73. solver.Sum(bc.iloc[i,j] \* x[T][i][j] **for** i **in** non\_smart) + 74. solver.Sum(bc.iloc[i,j]  \* x[2\*T] [i][j] **for** i **in** smart) 75. **for** j **in** range(nj) 76. ) 77. **for** T **in** range(nt//2): 78. sum\_output = solver.Sum([sc[0][j] \* output[T][j] **for** j **in** range(nj)]) 79. sum\_remain = theta\*solver.Sum([sc[0][j] \* remain[T][j] **for** j **in** range(nj)]) 80. **for** j **in** range(nj): 81. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 82. output\_constraint\_1 = solver.Sum(x[T][i][j] \* pro1[0].iloc[i,j] **for** i **in** non\_smart) 83. output\_constraint\_2 = solver.Sum(x[2\*T] [i][j]\* pro1[0].iloc[i,j] **for** i **in** smart) 84. solver.Add(output[T][j] <= output\_constraint\_1+output\_constraint\_2) 85. # 约束3: output\_{t}\_{j} <= exp\_{j} 86. solver.Add(output[T][j] <= Exp[j]) 87. m=1000000 88. b=output\_constraint\_1+output\_constraint\_2- Exp[j] 89. c=0 90. solver.Add(remain[T][j]<=b+m\*z1[T][j]) 91. solver.Add(remain[T][j]<=c+m\*z2[T][j]) 92. solver.Add(remain[T][j]>=b) 93. solver.Add(remain[T][j]>=c) 94. solver.Add(z1[T][j]+z2[T][j]==1) 95. # # 约束4: remain\_{t}\_{j} >= 0 96. # solver.Add(remain[T][j] >= 0) 97. # # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 98. # solver.Add(remain[T][j] >= output\_constraint\_1+output\_constraint\_2- Exp[j]) 99. S\_T[T]=sum\_output+sum\_remain 100. output1 = [[solver.NumVar(0, solver.infinity(), f'output[{T}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)] 101. remain1 = [[solver.NumVar(0, solver.infinity(), f'remain[{j}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)] 102. B = [solver.NumVar(0, solver.infinity(), f'B\_{t}') **for** t **in** range(nt//2)] 103. S = [solver.NumVar(0, solver.infinity(), f'S\_{t}') **for** t **in** range(nt//2)] 104. z11 = [[solver.NumVar(0, 1 , f'output[{T}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)] 105. z22 = [[solver.NumVar(0, 1, f'remain[{j}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)] 107. **for** t **in** range(nt//2): 108. sum\_output = solver.Sum([sc[1][j] \* output1[t][j] **for** j **in** range(nj)]) 109. sum\_remain = theta\*solver.Sum([sc[1][j] \* remain1[t][j] **for** j **in** range(nj)]) 110. **for** j **in** range(nj): 111. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 112. output\_constraint\_2 = solver.Sum(x[2\*t+1][i][j]\* pro1[1].iloc[i,j] **for** i **in** smart) 113. solver.Add(output1[t][j] <= output\_constraint\_2) 114. # 约束3: output\_{t}\_{j} <= exp\_{j} 115. solver.Add(output1[t][j] <= Exp[j]) 116. b=output\_constraint\_2- Exp[j] 117. c=0 118. m=1000000 119. solver.Add(remain1[t][j]<=b+m\*z11[t][j]) 120. solver.Add(remain1[t][j]<=c+m\*z22[t][j]) 121. solver.Add(remain1[t][j]>=b) 122. solver.Add(remain1[t][j]>=c) 123. solver.Add(z11[t][j]+z22[t][j]==1) 124. # # 约束4: remain\_{t}\_{j} >= 0 125. # solver.Add(remain1[t][j] >= 0) 126. # # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 127. # solver.Add(remain1[t][j] >= output\_constraint\_2- Exp[j]) 128. S[t] =sum\_output+sum\_remain 129. **for** t **in** range(nt//2): 130. B[t] = solver.Sum(bc.iloc[i,j] \* x[2\*t+1][i][j] **for** i **in** smart **for** j **in** range(nj)) 131. **for** T **in** range(nt//2): 132. **for** i **in** non\_smart: 133. non\_smart\_sum = solver.Sum(x[T][i][j] **for** j **in** range(nj)) 134. # 添加约束 135. solver.Add(non\_smart\_sum <= area.iloc[i]) 136. **for** t **in** range(nt):  # t = 3, 5, ..., 15 137. **for** i **in** smart: 138. smart\_sum\_t = solver.Sum(x[t][i][j] **for** j **in** range(nj)) 139. solver.Add(smart\_sum\_t <= area.iloc[i]) 140. # for i in range(ni): 141. #     for t in range(2, nt): 142. #         # 创建约束 143. #         constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x[t-2][i][j] for j in bean\_class\_indices) 144. #         solver.Add(constraint\_expr >= area[i]) 146. # # 三年种豆 147. **for** t **in** range(4, nt): 148. **if** t==4: 149. **for** i **in** smart: 150. 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] 151. **for** j **in** bean\_class\_indices) 152. solver.Add(bean\_sum >= area.iloc[i]) 153. **else**: 154. **for** i **in** smart: 155. 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] 156. **for** j **in** bean\_class\_indices) 157. solver.Add(bean\_sum >= area.iloc[i]) 158. **for** T **in** range(1, nt//2): 159. **if** T==1: 160. **for** i **in** non\_smart: 161. 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) 162. solver.Add(bean\_sum\_non\_smart >= area.iloc[i]) 163. **else**: 164. **for** i **in** non\_smart: 165. 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) 166. solver.Add(bean\_sum\_non\_smart >= area.iloc[i]) 167. # 相邻两季不同 168. **for** T **in** range(nt // 2 ): 169. **if** T==0: 170. **for** i **in** non\_smart: 171. **for** j **in** [0]: 172. solver.Add(y[T][i][j] + y0.iloc[i,j] <= 1) 173. **else**: 174. **for** i **in** non\_smart: 175. **for** j **in** [0]: 176. solver.Add(y[T][i][j] + y[T-1][i][j] <= 1) 177. **for** t **in** range(nt): 178. **if** t==0: 179. **for** i **in** smart: 180. **for** j **in** range(nj): 181. solver.Add(y[t][i][j] + y0.iloc[i,j]<= 1) 182. **else**: 183. **for** i **in** smart: 184. **for** j **in** range(nj): 185. solver.Add(y[t][i][j] + y[t-1][i][j] <= 1) 187. # x,y的约束 188. big\_M = 10000 189. **for** t **in** range(nt): 190. **for** i **in** range(ni): 191. **for** j **in** range(nj): 192. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 193. # 不能太分散 194. N=5 195. **for** T **in** range(nt//2):  # T = 1, 2, 3, ... 196. **for** j **in** range(nj): 197. non\_smart\_sum\_y = solver.Sum(y[T][i][j] **for** i **in** non\_smart) 198. smart\_sum\_y = solver.Sum(y[2\*T] [i][j]**for** i **in** smart) 199. solver.Add(non\_smart\_sum\_y + smart\_sum\_y <= N) 200. **for** t **in** range(1, nt, 2):  # t = 1, 3, 5, ... 201. **for** j **in** range(nj): 202. smart\_sum\_y\_t = solver.Sum(y[t][i][j] **for** i **in** smart) 203. solver.Add(smart\_sum\_y\_t <= N) 204. M=0.4 205. **for** T **in** range(nt // 2):  # T = 1, 2, 3, ... 206. **for** j **in** range(nj): 207. **for** i **in** non\_smart: 208. solver.Add(x[T][i][j] >=area.iloc[i] \*M \* y[T][i][j]) 210. solver.SetTimeLimit(120000) 211. solver.EnableOutput() 212. 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)) 214. solver.Maximize(objective\_expr) 215. status = solver.Solve() 216. status 217. remain\_values = [[remain[T][j].solution\_value() **for** j **in** range(nj)] **for** T **in** range(nt//2)] 218. output\_values=[[output[T][j].solution\_value() **for** j **in** range(nj)] **for** T **in** range(nt//2)] 219. remain1\_values = [[remain1[T][j].solution\_value() **for** j **in** range(nj)] **for** T **in** range(nt//2)] 220. output1\_values=[[output1[T][j].solution\_value() **for** j **in** range(nj)] **for** T **in** range(nt//2)] 222. **for** T **in** range(nt//2): 223. sum\_output = sum([sc[0][j] \* output\_values[T][j] **for** j **in** range(nj)]) 224. sum\_remain = theta\*sum([sc[0][j] \* remain\_values[T][j] **for** j **in** range(nj)]) 225. S\_T[T]=sum\_output+sum\_remain 226. **for** T **in** range(nt // 2): 227. B\_T[T] = sum( 228. sum(bc.iloc[i,j] \* x\_np[T][i][j] **for** i **in** non\_smart) + 229. sum(bc.iloc[i,j]  \* x\_np[2\*T] [i][j] **for** i **in** smart) 230. **for** j **in** range(nj) 231. ) 232. **for** t **in** range(nt//2): 234. **for** j **in** range(nj): 235. output\_constraint\_2 = sum(x\_np[2\*t+1][i][j]\* pro1[1].iloc[i,j] **for** i **in** smart) 236. **if** output\_constraint\_2- Exp[j]>0: 237. **print**(f'{t}\_{j}    {output\_constraint\_2- Exp[j]}') 238. remain1\_values[t][j]=max((output\_constraint\_2- Exp[j]),0) 239. sum\_output = sum([sc[1][j] \* output1\_values[t][j] **for** j **in** range(nj)]) 240. sum\_remain = theta\*sum([sc[1][j] \* remain1\_values[t][j] **for** j **in** range(nj)]) 241. S[t] =sum\_output+sum\_remain 242. **for** t **in** range(nt//2): 243. B[t] =sum(bc.iloc[i,j] \* x\_np[2\*t+1][i][j] **for** i **in** smart **for** j **in** range(nj)) 244. **for** t **in** range(nt//2): 245. **print**(S\_T[t]-B\_T[t]+S[t]-B[t]) 246. cucumber=np.zeros((7, 2)) 247. **for** t **in** range(nt//2): 248. cucumber[t][0]=output\_values[t][13]+output1\_values[t][13] 249. cucumber[t][1]=remain\_values[t][13]+remain1\_values[t][13] 250. cucumber 251. **from** ortools.linear\_solver **import** pywraplp 252. ni=24 253. nj=41-15 254. nt=7 255. solver = pywraplp.Solver.CreateSolver('SCIP') 256. # solver.SetSolverSpecificParametersAsString('log\_level=2')  # 2 代表详细日志级别 257. 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)] 258. 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)] 260. theta=0 261. # b = [solver.NumVar(0, solver.infinity(), f'b\_{j}') for j in range(nj)] 262. B = [solver.NumVar(0, solver.infinity(), f'B\_{t}') **for** t **in** range(nt)] 263. S = [solver.NumVar(0, solver.infinity(), f'S\_{t}') **for** t **in** range(nt)] 264. **for** t **in** range(nt): 265. B[t] = solver.Sum(bc.iloc[i,j] \* x[t][i][j] **for** i **in** range(ni) **for** j **in** range(nj)) 266. output =[[solver.NumVar(0, solver.infinity(), f'output\_{t}\_{j}') **for** j **in** range(nj)] **for** t **in** range(nt)] 267. remain =[[solver.NumVar(0, solver.infinity(), f'remain\_{t}\_{j}') **for** j **in** range(nj)] **for** t **in** range(nt)] 268. **for** t **in** range(nt): 269. sum\_output = solver.Sum([sc[j] \* output[t][j] **for** j **in** range(nj)]) 270. sum\_remain = theta\*solver.Sum([sc[j] \* remain[t][j] **for** j **in** range(nj)]) 271. **for** j **in** range(nj): 272. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 273. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] **for** i **in** range(ni)) 274. solver.Add(output[t][j] <= sum\_pro) 275. # 约束3: output\_{t}\_{j} <= exp\_{j} 276. solver.Add(output[t][j] <= Exp[j]) 277. # 约束4: remain\_{t}\_{j} >= 0 278. solver.Add(remain[t][j] >= 0) 279. # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 280. solver.Add(remain[t][j] >= sum\_pro - Exp[j]) 281. S[t]=sum\_output+sum\_remain 283. **for** i **in** range(ni): 284. **for** t **in** range(nt): 285. constraint\_expr = solver.Sum(x[t][i][j] **for** j **in** range(nj)) 286. solver.Add(constraint\_expr <= area[t][i]) 288. big\_M = 10000 289. **for** t **in** range(nt): 290. **for** i **in** range(ni): 291. **for** j **in** range(nj): 292. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 293. N = 5 294. **for** j **in** range(nj): 295. **for** t **in** range(nt): 296. # 创建约束 297. constraint\_expr = solver.Sum(y[t][i][j] **for** i **in** range(ni)) 298. solver.Add(constraint\_expr <= N) 299. M = 0.4 301. **for** t **in** range(nt): 302. **for** j **in** range(nj): 303. **for** i **in** range(ni): 304. # 创建约束 305. constraint\_expr = x[t][i][j] >= M \* y[t][i][j] \* area[t][i] 306. solver.Add(constraint\_expr) 307. solver.SetTimeLimit(120000) 308. solver.EnableOutput() 309. objective\_expr = solver.Sum(S[t] **for** t **in** range(nt)) - solver.Sum(B[t] **for** t **in** range(nt)) 310. solver.Maximize(objective\_expr) 311. status = solver.Solve() 312. status |

|  |
| --- |
| **附录4** |
| 介绍：利用Python语言编写并求解问题二子模型1 |
| 1. **import** numpy as np 3. iterations\_number=50 4. total\_number=7\*iterations\_number 6. mu=1 7. sigma = (1.1 - mu) / 3 8. data\_yield = np.random.normal(mu, sigma, total\_number) 10. mu=1 11. sigma = (1.05 - mu) / 3 12. data\_pre\_sell = np.random.normal(mu, sigma, total\_number) 14. morel\_price = 0.95 16. mushrooms\_price = 1 - np.random.uniform(0.01, 0.05, total\_number) 18. Wheat\_price = 1 + np.random.uniform(0.05, 0.1, total\_number) 20. mu = 5 21. sigma = 0.5/3 22. sell\_price = np.random.normal(1+mu/100, sigma/100, total\_number) 23. plant\_price = np.random.normal(1+mu/100, sigma/100, total\_number) 24. **from** ortools.linear\_solver **import** pywraplp 25. ni=26 26. nj=15 27. nt=7 28. solver = pywraplp.Solver.CreateSolver('SCIP') 29. # solver.SetSolverSpecificParametersAsString('log\_level=2')  # 2 代表详细日志级别 30. 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)] 31. 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)] 33. theta=0 34. # b = [solver.NumVar(0, solver.infinity(), f'b\_{j}') for j in range(nj)] 35. B = [[solver.NumVar(0, solver.infinity(), f'B\_{t}') **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 36. S = [[solver.NumVar(0, solver.infinity(), f'S\_{t}') **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 37. **for** t **in** range(nt): 38. **for** interation **in** range(iterations\_number): 39. plantprice=np.prod(plant\_price[interation\*7:t+1+interation\*7]) 40. B[interation][t] = solver.Sum(bc.iloc[j] \* x[t][i][j] **for** i **in** range(ni) **for** j **in** range(nj))\*plantprice 41. output =[[[solver.NumVar(0, solver.infinity(), f'output\_{t}\_{j}\_{interation}') **for** j **in** range(nj)] **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 42. remain =[[[solver.NumVar(0, solver.infinity(), f'remain\_{t}\_{j}\_{interation}') **for** j **in** range(nj)] **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 43. **for** t **in** range(nt): 44. **for** interation **in** range(iterations\_number): 45. datayield=data\_yield[t+interation\*7] 46. datapresell=data\_pre\_sell[t+interation\*7] 47. Wheatprice=np.prod(Wheat\_price[interation\*7:t+1+interation\*7]) 48. sum\_output = solver.Sum([sc[j]  \* output[interation][t][j] **for** j **in** range(nj)]) 49. sum\_remain = theta\*solver.Sum([sc[j] \* remain[interation][t][j] **for** j **in** range(nj)]) 50. **for** j **in** [jj **for** jj **in** range(nj) **if** jj **not** **in** (5, 6)]: 51. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 52. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] **for** i **in** range(ni)) 53. solver.Add(output[interation][t][j] <= sum\_pro\*datayield) 54. # 约束3: output\_{t}\_{j} <= exp\_{j} 55. solver.Add(output[interation][t][j] <= Exp.iloc[j,0]\*datapresell) 56. # 约束4: remain\_{t}\_{j} >= 0 57. solver.Add(remain[interation][t][j] >= 0) 58. # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 59. solver.Add(remain[interation][t][j] >= sum\_pro\*datayield - Exp.iloc[j,0]\*datapresell) 60. **for** j **in** (5, 6): 61. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 62. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] **for** i **in** range(ni)) 63. solver.Add(output[interation][t][j] <= sum\_pro\*datayield) 64. # 约束3: output\_{t}\_{j} <= exp\_{j} 65. solver.Add(output[interation][t][j] <= Exp.iloc[j,0]\*Wheatprice) 66. # 约束4: remain\_{t}\_{j} >= 0 67. solver.Add(remain[interation][t][j] >= 0) 68. # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 69. solver.Add(remain[interation][t][j] >= sum\_pro\*datayield - Exp.iloc[j,0]\*Wheatprice) 70. S[interation][t]=(sum\_output+sum\_remain) 72. **for** i **in** range(ni): 73. **for** t **in** range(nt): 74. constraint\_expr = solver.Sum(x[t][i][j] **for** j **in** range(nj)) 75. solver.Add(constraint\_expr <= area[i]) 76. **for** t **in** range(nt): 77. **if** t==0: 78. solver.Add(y[t][i][j] + y0.iloc[i,j] <= 1) 79. **else**: 80. **for** i **in** range(ni): 81. **for** j **in** range(nj): 82. solver.Add(y[t][i][j] + y[t -1][i][j] <= 1) 83. big\_M = 10000 84. **for** t **in** range(nt): 85. **for** i **in** range(ni): 86. **for** j **in** range(nj): 87. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 88. **for** i **in** range(ni): 89. **for** t **in** range(1, nt): 90. **if**(t==0): 91. # 创建约束 92. constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x0.iloc[i,j] **for** j **in** bean\_class\_indices) 93. solver.Add(constraint\_expr >= area[i]) 94. **else**: 95. # 创建约束 96. constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x[t-2][i][j] **for** j **in** bean\_class\_indices) 97. 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 = 0.4 106. **for** t **in** range(nt): 107. **for** j **in** range(nj): 108. **for** i **in** range(ni): 109. # 创建约束 110. constraint\_expr = x[t][i][j] >= M \* y[t][i][j] \* area[i] 111. solver.Add(constraint\_expr) 112. solver.SetTimeLimit(600000) 113. solver.EnableOutput() 114. objective\_expr = solver.Sum( 115. S[interation][t] - B[interation][t] 116. **for** interation **in** range(iterations\_number) 117. **for** t **in** range(nt) 118. ) 119. solver.Maximize(objective\_expr) 120. status = solver.Solve() |

|  |
| --- |
| **附录5** |
| 介绍：利用Python语言编写并求解问题二子模型 |
| 1. **from** ortools.linear\_solver **import** pywraplp 2. ni=28 3. nj=41-15 4. nt=14 5. solver = pywraplp.Solver.CreateSolver('SCIP') 6. # solver.SetSolverSpecificParametersAsString('log\_level=2')  # 2 代表详细日志级别 7. 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)] 8. 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)] 10. theta=0 12. S\_T = [[solver.NumVar(0, solver.infinity(), f'S\_T[{T}]') **for** T **in** range(nt//2)]**for** interation **in** range(iterations\_number)] 13. B\_T = [[solver.NumVar(0, solver.infinity(), f'B\_T[{T}]') **for** T **in** range(nt//2)]**for** interation **in** range(iterations\_number)] 14. output = [[[solver.NumVar(0, solver.infinity(), f'output[{T}]') **for** j **in** range(nj)] **for** T **in** range(nt//2 )]**for** interation **in** range(iterations\_number)] 15. remain = [[[solver.NumVar(0, solver.infinity(), f'remain[{j}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)]**for** interation **in** range(iterations\_number)] 16. **for** T **in** range(nt // 2): 17. **for** interation **in** range(iterations\_number): 18. plantprice=np.prod(plant\_price[interation\*14:T+1+interation\*14]) 19. B\_T[interation][T] = solver.Sum( 20. solver.Sum(bc.iloc[i,j] \* x[T][i][j] **for** i **in** non\_smart) + 21. solver.Sum(bc.iloc[i,j] \* x[2\*T] [i][j] **for** i **in** smart) 22. **for** j **in** range(nj) 23. )\*plantprice 24. **for** T **in** range(nt//2): 25. **for** interation **in** range(iterations\_number): 26. datayield=data\_yield[T+interation\*7] 27. datapresell=data\_pre\_sell[T+interation\*7] 28. sellprice=np.prod(sell\_price[interation\*7:T+1+interation\*7]) 29. sum\_output = solver.Sum([sc[0][j] \* output[interation][T][j] **for** j **in** range(1,nj)])\*sellprice+solver.Sum([sc[0][0] \* output[interation][T][0]]) 30. sum\_remain = theta\*solver.Sum([sc[0][j] \* remain[interation][T][j] **for** j **in** range(1,nj)])\*sellprice+theta\*solver.Sum([sc[0][0] \* remain[interation][T][0]]) 31. **for** j **in** range(nj): 32. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 33. output\_constraint\_1 = solver.Sum(x[T][i][j] \* pro1[0].iloc[i,j] **for** i **in** non\_smart)\*datayield 34. output\_constraint\_2 = solver.Sum(x[2\*T][i][j]\* pro1[0].iloc[i,j] **for** i **in** smart)\*datayield 35. solver.Add(output[interation][T][j] <= output\_constraint\_1+output\_constraint\_2) 36. # 约束3: output\_{t}\_{j} <= exp\_{j} 37. solver.Add(output[interation][T][j] <= Exp[j]\*datapresell) 38. # 约束4: remain\_{t}\_{j} >= 0 39. solver.Add(remain[interation][T][j] >= 0) 40. # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 41. solver.Add(remain[interation][T][j] >= output\_constraint\_1+output\_constraint\_2- Exp[j]\*datapresell) 42. S\_T[interation][T]=sum\_output+sum\_remain 43. output1 = [[[solver.NumVar(0, solver.infinity(), f'output[{T}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)]**for** interation **in** range(iterations\_number)] 44. remain1 = [[[solver.NumVar(0, solver.infinity(), f'remain[{j}]') **for** j **in** range(nj)] **for** T **in** range(nt//2)]**for** interation **in** range(iterations\_number)] 45. B = [[solver.NumVar(0, solver.infinity(), f'B\_{t}') **for** t **in** range(nt//2)]**for** interation **in** range(iterations\_number)] 46. S = [[solver.NumVar(0, solver.infinity(), f'S\_{t}') **for** t **in** range(nt//2)]**for** interation **in** range(iterations\_number)] 47. **for** t **in** range(nt//2): 48. **for** interation **in** range(iterations\_number): 49. datayield=data\_yield[t+interation\*7] 50. datapresell=data\_pre\_sell[t+interation\*7] 51. sellprice=np.prod(sell\_price[interation\*7:t+1+interation\*7]) 52. sum\_output = solver.Sum([sc[1][j] \* output1[interation][t][j] **for** j **in** range(1,nj)])\*sellprice+solver.Sum([sc[1][0] \* output1[interation][t][0]]) 53. sum\_remain = theta\*solver.Sum([sc[1][j] \* remain1[interation][t][j] **for** j **in** range(1,nj)])\*sellprice+theta\*solver.Sum([sc[1][0] \* remain[interation][t][0]]) 54. **for** j **in** range(nj): 55. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 56. output\_constraint\_2 = solver.Sum(x[2\*t+1][i][j]\* pro1[1].iloc[i,j] **for** i **in** smart) 57. solver.Add(output1[interation][t][j] <= output\_constraint\_2\*datayield) 58. # 约束3: output\_{t}\_{j} <= exp\_{j} 59. solver.Add(output1[interation][t][j] <= Exp[j]\*datapresell) 60. # 约束4: remain\_{t}\_{j} >= 0 61. solver.Add(remain1[interation][t][j] >= 0) 62. # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 63. solver.Add(remain1[interation][t][j] >= output\_constraint\_2\*datayield- Exp[j]\*datapresell) 64. S[interation][t] =sum\_output+sum\_remain 65. **for** t **in** range(nt//2): 66. plantprice=np.prod(plant\_price[interation\*7:t+1+interation\*7]) 67. B[interation][t] = solver.Sum(bc.iloc[i,j] \* x[2\*t+1][i][j] **for** i **in** smart **for** j **in** range(nj))\*plantprice 69. **for** T **in** range(nt//2): 70. **for** i **in** non\_smart: 71. non\_smart\_sum = solver.Sum(x[T][i][j] **for** j **in** range(nj)) 72. # 添加约束 73. solver.Add(non\_smart\_sum <= area.iloc[i]) 74. **for** t **in** range(nt):  # t = 3, 5, ..., 15 75. **for** i **in** smart: 76. smart\_sum\_t = solver.Sum(x[t][i][j] **for** j **in** range(nj)) 77. solver.Add(smart\_sum\_t <= area.iloc[i]) 79. # 三年种豆 80. **for** t **in** range(4, nt): 81. **if** t==4: 82. **for** i **in** smart: 83. 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] 84. **for** j **in** bean\_class\_indices) 85. solver.Add(bean\_sum >= area.iloc[i]) 86. **else**: 87. **for** i **in** smart: 88. 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] 89. **for** j **in** bean\_class\_indices) 90. solver.Add(bean\_sum >= area.iloc[i]) 91. **for** T **in** range(1, nt//2): 92. **if** T==1: 93. **for** i **in** non\_smart: 94. 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) 95. solver.Add(bean\_sum\_non\_smart >= area.iloc[i]) 96. **else**: 97. **for** i **in** non\_smart: 98. 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) 99. solver.Add(bean\_sum\_non\_smart >= area.iloc[i]) 100. # 相邻两季不同 101. **for** T **in** range(nt // 2 ): 102. **if** T==0: 103. **for** i **in** non\_smart: 104. **for** j **in** [0]: 105. solver.Add(y[T][i][j] + y0.iloc[i,j] <= 1) 106. **else**: 107. **for** i **in** non\_smart: 108. **for** j **in** [0]: 109. solver.Add(y[T][i][j] + y[T-1][i][j] <= 1) 110. **for** t **in** range(nt): 111. **if** t==0: 112. **for** i **in** smart: 113. **for** j **in** range(nj): 114. solver.Add(y[t][i][j] + y0.iloc[i,j]<= 1) 115. **else**: 116. **for** i **in** smart: 117. **for** j **in** range(nj): 118. solver.Add(y[t][i][j] + y[t-1][i][j] <= 1) 120. # x,y的约束 121. big\_M = 10000 122. **for** t **in** range(nt): 123. **for** i **in** range(ni): 124. **for** j **in** range(nj): 125. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 126. # 不能太分散 127. N=5 128. **for** T **in** range(nt//2):  # T = 1, 2, 3, ... 129. **for** j **in** range(nj): 130. non\_smart\_sum\_y = solver.Sum(y[T][i][j] **for** i **in** non\_smart) 131. smart\_sum\_y = solver.Sum(y[2\*T] [i][j]**for** i **in** smart) 132. solver.Add(non\_smart\_sum\_y + smart\_sum\_y <= N) 133. **for** t **in** range(1, nt, 2):  # t = 1, 3, 5, ... 134. **for** j **in** range(nj): 135. smart\_sum\_y\_t = solver.Sum(y[t][i][j] **for** i **in** smart) 136. solver.Add(smart\_sum\_y\_t <= N) 137. M=0.4 138. **for** T **in** range(nt // 2):  # T = 1, 2, 3, ... 139. **for** j **in** range(nj): 140. **for** i **in** non\_smart: 141. solver.Add(x[T][i][j] >=area.iloc[i] \*M \* y[T][i][j]) 143. solver.SetTimeLimit(600000) 144. solver.EnableOutput() 145. objective\_expr =  solver.Sum(S\_T[interation][t]-B\_T[interation][t]+S[interation][t]-B[interation][t] **for** interation **in** range(iterations\_number) **for** t **in** range(nt//2)) 146. solver.Maximize(objective\_expr) 147. status = solver.Solve() 148. status 149. **from** ortools.linear\_solver **import** pywraplp 150. ni=24 151. nj=41-15 152. nt=7 153. solver = pywraplp.Solver.CreateSolver('SCIP') 154. # solver.SetSolverSpecificParametersAsString('log\_level=2')  # 2 代表详细日志级别 155. 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)] 156. 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)] 158. theta=0 159. # b = [solver.NumVar(0, solver.infinity(), f'b\_{j}') for j in range(nj)] 160. B = [[solver.NumVar(0, solver.infinity(), f'B\_{t}') **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 161. S = [[solver.NumVar(0, solver.infinity(), f'S\_{t}') **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 162. **for** t **in** range(nt): 163. **for** interation **in** range(iterations\_number): 164. plantprice=np.prod(plant\_price[interation\*7:t+1+interation\*7]) 165. B[interation][t] = solver.Sum(bc.iloc[i,j] \* x[t][i][j] **for** i **in** range(ni) **for** j **in** range(nj))\*plantprice 166. output =[[[solver.NumVar(0, solver.infinity(), f'output\_{t}\_{j}') **for** j **in** range(nj)] **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 167. remain =[[[solver.NumVar(0, solver.infinity(), f'remain\_{t}\_{j}') **for** j **in** range(nj)] **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 168. **for** t **in** range(nt): 169. **for** interation **in** range(iterations\_number): 170. datayield=data\_yield[t+interation\*7] 171. datapresell=data\_pre\_sell[t+interation\*7] 172. sellprice=np.prod(sell\_price[interation\*7:t+1+interation\*7]) 173. mushroomsprice=np.prod(mushrooms\_price[interation\*7:t+1+interation\*7]) 174. morelprice=morel\_price\*\*(t+1) 175. sum\_output = solver.Sum([sc[j] \* output[interation][t][j] **for** j **in** [jj **for** jj **in** range(nj) **if** jj **not** **in** (22,23,24,25)]])\*sellprice+solver.Sum([sc[j] \* output[interation][t][j] **for** j **in** (22,23,24)])\*mushroomsprice+solver.Sum([sc[25] \* output[interation][t][25]])\*morelprice 176. sum\_remain = theta\*solver.Sum([sc[j] \* remain[interation][t][j] **for** j **in** [jj **for** jj **in** range(nj) **if** jj **not** **in** (22,23,24,25)]])\*sellprice+theta\*solver.Sum([sc[j] \* remain[interation][t][j] **for** j **in** (22,23,24)])\*mushroomsprice+theta\*solver.Sum([sc[25] \* remain[interation][t][25]])\*morelprice 177. **for** j **in** range(nj): 178. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 179. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] **for** i **in** range(ni)) 180. solver.Add(output[interation][t][j] <= sum\_pro\*datayield) 181. # 约束3: output\_{t}\_{j} <= exp\_{j} 182. solver.Add(output[interation][t][j] <= Exp[j]\*datapresell) 183. # 约束4: remain\_{t}\_{j} >= 0 184. solver.Add(remain[interation][t][j] >= 0) 185. # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 186. solver.Add(remain[interation][t][j] >= sum\_pro\*datayield - Exp[j]\*datapresell) 187. S[interation][t]=sum\_output+sum\_remain 189. **for** i **in** range(ni): 190. **for** t **in** range(nt): 191. constraint\_expr = solver.Sum(x[t][i][j] **for** j **in** range(nj)) 192. solver.Add(constraint\_expr <= area[t][i]) 194. big\_M = 10000 195. **for** t **in** range(nt): 196. **for** i **in** range(ni): 197. **for** j **in** range(nj): 198. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 200. N = 5 201. **for** j **in** range(nj): 202. **for** t **in** range(nt): 203. # 创建约束 204. constraint\_expr = solver.Sum(y[t][i][j] **for** i **in** range(ni)) 205. solver.Add(constraint\_expr <= N) 206. M = 0.4 208. **for** t **in** range(nt): 209. **for** j **in** range(nj): 210. **for** i **in** range(ni): 211. # 创建约束 212. constraint\_expr = x[t][i][j] >= M \* y[t][i][j] \* area[t][i] 213. solver.Add(constraint\_expr) 214. solver.SetTimeLimit(120000) 215. solver.EnableOutput() 216. objective\_expr = solver.Sum(S[interation][t] - B[interation][t] **for** interation **in** range(iterations\_number) **for** t **in** range(nt)) 217. solver.Maximize(objective\_expr) 218. status = solver.Solve() 219. status |

|  |
| --- |
| **附录6** |
| 介绍：利用Python语言编写并求解问题三子模型 |
| 1. **from** ortools.linear\_solver **import** pywraplp 2. ni=26 3. nj=15 4. nt=7 5. solver = pywraplp.Solver.CreateSolver('SCIP') 6. # solver.SetSolverSpecificParametersAsString('log\_level=2')  # 2 代表详细日志级别 7. 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)] 8. 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)] 10. theta=0 11. # b = [solver.NumVar(0, solver.infinity(), f'b\_{j}') for j in range(nj)] 12. B = [[solver.NumVar(0, solver.infinity(), f'B\_{t}') **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 13. S = [[solver.NumVar(0, solver.infinity(), f'S\_{t}') **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 14. **for** t **in** range(nt): 15. **for** interation **in** range(iterations\_number): 16. plantprice=np.prod(plant\_price[interation\*7:t+1+interation\*7]) 17. B[interation][t] = solver.Sum(bc.iloc[j] \* x[t][i][j] **for** i **in** range(ni) **for** j **in** range(nj))\*plantprice 18. output =[[[solver.NumVar(0, solver.infinity(), f'output\_{t}\_{j}\_{interation}') **for** j **in** range(nj)] **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 19. remain =[[[solver.NumVar(0, solver.infinity(), f'remain\_{t}\_{j}\_{interation}') **for** j **in** range(nj)] **for** t **in** range(nt)]**for** interation **in** range(iterations\_number)] 20. **for** t **in** range(nt): 21. **for** interation **in** range(iterations\_number): 22. datayield=data\_yield[t+interation\*7] 23. datapresell=data\_pre\_sell[t+interation\*7] 24. Wheatprice=np.prod(Wheat\_price[interation\*7:t+1+interation\*7]) 25. sum\_output = solver.Sum([sc[j]  \* output[interation][t][j] **for** j **in** range(nj)]) 26. sum\_remain = theta\*solver.Sum([sc[j] \* remain[interation][t][j] **for** j **in** range(nj)]) 27. **for** j **in** [jj **for** jj **in** range(nj) **if** jj **not** **in** (5, 6)]: 28. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 29. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] **for** i **in** range(ni)) 30. solver.Add(output[interation][t][j] <= sum\_pro\*datayield) 31. # 约束3: output\_{t}\_{j} <= exp\_{j} 32. solver.Add(output[interation][t][j] <= Exp.iloc[j,0]\*datapresell) 33. # 约束4: remain\_{t}\_{j} >= 0 34. solver.Add(remain[interation][t][j] >= 0) 35. # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 36. solver.Add(remain[interation][t][j] >= sum\_pro\*datayield - Exp.iloc[j,0]\*datapresell) 37. **for** j **in** (5, 6): 38. # 约束2: output\_{t}\_{j} <= sum\_{i=1} x\_{ijt} \* pro\_{ij} 39. sum\_pro = solver.Sum(x[t][i][j] \* pro.iloc[i,j] **for** i **in** range(ni)) 40. solver.Add(output[interation][t][j] <= sum\_pro\*1.075\*\*t\*ppprice[0]\*\*gamma[j-5,0]\*ppprice[1]\*\*gamma[j-5,1]\*(ppprice[2]\*1.05\*\*t)\*\*gamma[j-5,2]\*ppprice[3]\*\*gamma[j-5,3]) 41. # 约束3: output\_{t}\_{j} <= exp\_{j} 42. solver.Add(output[interation][t][j] <= Exp.iloc[j,0]\*Wheatprice) 43. # 约束4: remain\_{t}\_{j} >= 0 44. solver.Add(remain[interation][t][j] >= 0) 45. # 约束5: remain\_{t}\_{j} >= sum\_{i=1} x\_{ijt} \* pro\_{ij} - exp\_{j} 46. solver.Add(remain[interation][t][j] >= sum\_pro\*1.075\*\*t\*ppprice[0]\*\*gamma[j-5,0]\*ppprice[1]\*\*gamma[j-5,1]\*(ppprice[2]\*1.05\*\*t)\*\*gamma[j-5,2]\*ppprice[3]\*\*gamma[j-5,3] - Exp.iloc[j,0]\*Wheatprice) 47. S[interation][t]=(sum\_output+sum\_remain) 49. **for** i **in** range(ni): 50. **for** t **in** range(nt): 51. constraint\_expr = solver.Sum(x[t][i][j] **for** j **in** range(nj)) 52. solver.Add(constraint\_expr <= area[i]) 53. **for** t **in** range(nt): 54. **if** t==0: 55. solver.Add(y[t][i][j] + y0.iloc[i,j] <= 1) 56. **else**: 57. **for** i **in** range(ni): 58. **for** j **in** range(nj): 59. solver.Add(y[t][i][j] + y[t -1][i][j] <= 1) 60. big\_M = 10000 61. **for** t **in** range(nt): 62. **for** i **in** range(ni): 63. **for** j **in** range(nj): 64. solver.Add(x[t][i][j] <= big\_M \* y[t][i][j]) 65. **for** i **in** range(ni): 66. **for** t **in** range(1, nt): 67. **if**(t==0): 68. # 创建约束 69. constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x0.iloc[i,j] **for** j **in** bean\_class\_indices) 70. solver.Add(constraint\_expr >= area[i]) 71. **else**: 72. # 创建约束 73. constraint\_expr = solver.Sum(x[t][i][j] + x[t-1][i][j] + x[t-2][i][j] **for** j **in** bean\_class\_indices) 74. solver.Add(constraint\_expr >= area[i]) 75. N = 3 76. **for** j **in** range(nj): 77. **for** t **in** range(nt): 78. # 创建约束 79. constraint\_expr = solver.Sum(y[t][i][j] **for** i **in** range(ni)) 80. solver.Add(constraint\_expr <= N) 81. M = 0.4 83. **for** t **in** range(nt): 84. **for** j **in** range(nj): 85. **for** i **in** range(ni): 86. # 创建约束 87. constraint\_expr = x[t][i][j] >= M \* y[t][i][j] \* area[i] 88. solver.Add(constraint\_expr) 89. solver.SetTimeLimit(600000) 90. solver.EnableOutput() 91. objective\_expr = solver.Sum( 92. S[interation][t] - B[interation][t] 93. **for** interation **in** range(iterations\_number) 94. **for** t **in** range(nt) 95. ) 96. solver.Maximize(objective\_expr) 97. status = solver.Solve() |