**数学建模C题思路**

1. **概述**

c题是一道很典型的很传统的数学建模应用问题，这个题对于大多数没有基础或者第一次参加数学建模比赛的同学非常适合。这个题目整体难度并不是很大，而且在选题的方法上仍然给足了一定的空间，并且题目本身的发挥空间并不大，因此解题过程中可以循规蹈矩，按照正常的方法和思路来做，就可以能够做出结果，并且取得不错的成绩。

整个题目主要有三问，第一问是一道优化问题，主要是对于蔬菜的种植方式能够获得最好的一种优化结果。以便取得最优的收益。第二问是预测问题，要预测未来六年的蔬菜整体的种植情况，以便在波动的价格和其他的影响范围内获得足够的收益。第三问并没有要求我们求出来结果并附上表格。但是第三位要求我们分析各个数值之间的相对数量关系，这一问需要我们对相关因素和相关农作物等等进行充分的关联分析，这也会用到一些数据预处理相关的一些技术。同时第三方给了较大的开放空间，因此同学们可以在这一问中有一些可发挥的余地。

这道题的一个特点是有很多的约束，因此对于每个约束都要进行详细的分析。建议在做题之前首先梳理整道题的全部涉及到的约束，并且梳理完成之后再明确决策变量进行建模求解。约束在梳理的过程中不要遗漏，要反复检查。

题目中给出了2023年种植的情况，这里的情况主要是受到其中一条约束也就是三年内必须种植一次豆类作物，这个影响。

**二、第一问的解决思路**

问题1的解决方案涉及为乡村制定从2024年到2030年的最优农作物种植方案，分别考虑两种情况:（1）超过的作物产量滞销导致浪费；

（2）超过的作物产量按2023年销售价格的 降价出售。以下是详细的解决步骤。

1. **决策变量定义**

定义决策变量如下:

* : 表示在第 年在第 个地块 (或大棚) 上种植第 种作物的面积（单位：亩）。其中：
* ：农作物种类集合（如小麦、玉米、豆类、蔬菜等）。
* 、地块集合（包括平旱地、梯田、山坡地、水浇地、普通大棚和智慧大棚等) 。
* 种植年度集合。

1. **目标函数构建**

建立两个不同情况下的目标函数:  
(1) 超过部分滞销，造成浪费

目标是最大化乡村在2024年至2030年期间的总收益，考虑到超过的作物产量将无法销售，造成浪费:

其中:

* : 第 种作物的销售价格 (单位: 元/亩) 。
* : 第 种作物在第 年的总产量 (单位: 吨), 是第 种作物的亩产量 (吨/亩)。
* ：第 种作物的预期销售量 (吨)。
* ：第 种作物的种植成本 (元/亩)。  
  (2) 超过部分按50%价格出售：目标是最大化乡村在2024年至2030年期间的总收益，考虑到超过部分作物产量可以按 的价格出售:

两个小问唯一的区别就在于目标函数不同，因此不必分开来做。

**约束条件：**

1. 种植面积
2. 连续种植
3. 三年种植一次豆类作物
4. 种植分布限制
5. 特定地块的种植作物类型

使用线性规划求解器（如Gurobi、Lingo、Python的SciPy等）求解，或者采用智能优化算法（粒子群算法、模拟退火算法等）获取最优解。

求解算法汇总：

线性规划（LP）：适用于大规模线性约束优化问题。

整数线性规划（ILP）：当作物种植决策需要整数解时使用。

混合整数规划（MIP）：适用于包含连续和整数决策变量的问题。

动态规划（DP）：适合分阶段决策问题，可以用于简单场景的求解。

遗传算法（GA）：适用于大规模组合优化问题，通过模拟进化过程找到近似最优解。

模拟退火算法（SA）：适用于非线性优化问题，通过模拟物理退火过程找到全局最优解。

粒子群优化算法（PSO）：基于群体智能的优化算法，适用于非线性、多峰值问题。

其他的智能优化算法、元启发式算法、超启发式算法均可用。

* 1. **第二问的解决思路**

和第一问的变化：

* 小麦和玉米的预期销售量年增长率介于5%~10%之间。
* 其他农作物每年预期销售量相对于2023年有±5%的变化。
* 每种作物的亩产量受气候等因素影响，每年有±10%的变化。
* 种植成本每年平均增长5%左右。
* 销售价格变化：
* 粮食类作物的销售价格稳定。
* 蔬菜类作物销售价格年均增长5%左右。
* 食用菌销售价格每年下降1%~5%，羊肚菌每年下降5%。

之后需要确定【不确定性参数】的分布，例如定义均匀分布，增长率，或者其他分布方法。也可以直接采用蒙特卡洛方法。

大致思路：可以采取多次随机求解选择平均值，也可以计算最坏结果，保证最坏结果的最优值（即各种情况都是最差的基础上，求得最大收益，稳健优化方法）

可以将问题建模为一个随机规划问题，其中模型中考虑了不同场景的权重和概率。

可选算法：

蒙特卡罗模拟 (Monte Carlo Simulation) - 用于处理不确定性因素和随机变量的模拟。

随机规划 (Stochastic Programming) - 处理不确定性参数的优化问题。

稳健优化 (Robust Optimization) - 在不确定性环境下寻找最坏情况下的最优解。

情景分析 (Scenario Analysis) - 分析不同情景下的最优决策。

样本均值近似 (Sample Average Approximation, SAA) - 使用样本均值来近似解决随机规划题。

随机梯度下降 (Stochastic Gradient Descent, SGD) - 适用于大规模随机优化问题。

分布鲁棒优化 (Distributionally Robust Optimization) - 考虑分布不确定性，在不同分布下进行优化。

**四．第三问的解决思路**

1、明确可替代性和互补性的定义：

示例：

**可替代性**：小麦和玉米作为粮食类作物，可能在相同条件下满足同类市场需求，这种情况下，一种作物的种植面积增加可能导致另一种作物的种植面积减少。

**互补性**：豆类作物具有固氮作用，有利于提高后续其他作物的产量。因此，豆类与其他作物之间存在互补性，种植豆类可能会提高其他作物的收益。

**具体可收集相关文献。**

2、收集相关数据进行关联性分析，建立关联性模型。

线性回归、联合概率分布模型、相关系数矩阵

3、调整目标函数、综合优化求解（可以不求出全部解，因为不需要给出结果的表格）

也可以只做一部分，给出具体的说理分析。

可选算法：

混合整数非线性规划 (Mixed-Integer Nonlinear Programming, MINLP) - 处理含有非线性目标函数和约束的混合整数问题。

协方差调整优化 (Covariance Adjusted Optimization) - 通过调整协方差矩阵处理相关性问题。

多目标优化 (Multi-Objective Optimization) - 在多个目标之间进行权衡和优化。

拉格朗日松弛法 (Lagrangian Relaxation) - 用于处理具有复杂约束的优化问题。

多目标遗传算法 (Multi-Objective Genetic Algorithm, MOGA) - 适用于同时优化多个目标的复杂问题。

求解：

假设条件：

假设2023年的产量即为每种作物的预期销售量（因为题目中提到“假定各种农作物未来的预期销售量、种植成本、亩产量和销售价格相对于2023年保持稳定”）

第一问：

**定义决策变量如下:**

* 设 表示在第 年，第 个地块或大棚上，第 季度种植第 种作物的面积 (单位: 亩) 。

其中：

* 表示第 种作物。
* 表示第 个地块或大棚。
* 表示第 季度，取值范围为 1 或 2 ，对应于一年中的两季。
* 表示第 年，取值范围为 。

**目标函数：**根据题意，需要针对两种情况分别给出目标函数。目标函数旨在最大化种植收益。  
**情况1：超过部分滞销，造成浪费**

1. 符号定义:

* : 第 种作物的销售价格（单位：元/亩）。
* : 第 种作物的种植成本 (单位: 元/亩) 。
* ：第 种作物的亩产量（单位：千克/亩）。
* ：第 种作物的预期销售量（单位：千克）。
* : 第 个地块或大棚的总面积 (单位：亩)。
* : 第 种作物在第 年第 季度的总产量（单位：千克）。

1. 目标函数表达:

目标函数表示为最大化总收益，考虑超过部分不能正常销售的情况:

其中:

* 为第 种作物在第 年第 季度的总产量。

**情况2：超过部分按2023年销售价格的50%降价出售**

1. 符号定义:

除了上述符号外，新增定义:

* ：第 种作物超过部分的降价销售价格。

1. 目标函数表达:

目标函数表示为最大化总收益，考虑超过部分以折扣价销售的情况:

其中：

* 表示在第 年第 季度正常销售的部分。
* 表示超过预期销售量的部分。

————————————————————————————————————————

情况1: 超过部分滞销，造成浪费  
目标函数表示为最大化2024年至2030年期间的总收益，考虑超过部分不能正常销售的情况:

其中:

解释:

* 表示作物 在第 年第 季度的实际销售量（考虑预期销售量的上限）。
* 表示作物 的实际销售收入。
* 表示作物 的种植成本。
* 是2024年至2030年期间的总收益，考虑了所有种植季节和种植作物。

情况2：超过部分按2023年销售价格的50%降价出售  
目标函数表示为最大化2024年至2030年期间的总收益，考虑超过部分以折扣价销售的情况:

其中：

解释:

* 表示作物 在第 年第 季度的正常销售量。
* 表示超过预期销售量的部分。
* 是正常销售的部分收入。
* 是按折扣价销售的部分收入。
* 表示种植成本。
* 是2024年至2030年期间的总收益，考虑了所有种植季节和作物销售情况。

——————————————————————————————————————————————

**地块名称 (Plot Name) :**用符号 表示第 个地块，其中 为地块的编号。例如， 表示地块 A1，。

**地块类型 (Plot Type) :**用符号 表示第 个地块的类型。

取值范围: 平旱地, 梯田, 山坡地, 水浇地, 普通大棚, 智慧大棚 。

可以进一步用符号表示地块类型:平旱地: 梯田: 山坡地: 水浇地: 普通大棚: 智慧大棚:

**地块面积 (Plot Area) :**用符号 表示第 个地块的面积（单位：亩）。例如， 表示地块 A 1 的面积。

**作物编号 (Crop ID) :**用符号 表示第 种作物的编号。

**作物名称（Crop Name）：**用 表示第 种作物的名称。例如: 黄豆

**作物类型（Crop Type）：**用 表示第 种作物的类型。

取值范围: 粮食, 粮食 (豆类), 蔬菜, 蔬菜 (豆类), 食用菌

可以进一步用符号表示作物类型：

粮食: 粮食 (豆类) : 蔬菜: 蔬菜 (豆类) 食用菌：

约束条件：

1. 每个地块 (或大棚) 每年的种植面积不能超过总面积：对于每个地块 ，每季度种植面积总和不能超过地块总面积
2. 地块类型与作物类型的适应性约束:

定义三元决策变量 :

* : 表示作物 可以在季度 种植在地块 上。
* 表示作物 在季度 不能种植在地块 上。

新的适应性约束可以表示为:

解释

* 地块类型 ：代表地块 的类型（如平旱地、梯田、山坡地、水浇地、普通大棚、智慧大棚) 。
* 作物类型 ：代表作物 的类型（如粮食、粮食（豆类）、蔬菜、蔬菜（豆类）、食用菌）。
* 季度 : 表示种植季度，取值为 1 或2，分别对应一年中的两季。

1. 季节性种植限制:

* 对于"普通大棚"（ ，每年只能种植—季蔬菜和一季食用菌：
* 对于"智慧大棚" ，每年可以种植两季蔬菜:

1. 禁止重茬种植约束：对于每个地块 ，同一种作物 不能在相邻的两年或两季连续种植：
2. 豆类作物种植频率限制：每个地块（或大棚）在三年内至少需要种植一次豆类作物（粮食豆类 或蔬菜豆类
3. 作物种植集中度约束：我们将最小种植面积设置为 0.1 亩，因此种植面积 必须大于或等于 0.1 亩。

其中， 是一个二元变量，当 时， ；否则， 。

9.种植面积为 的倍数的约束: 为了确保 是 0.1 的倍数，我们可以将 表示为 0.1 的整数倍形式:

其中， 是一个非负整数，表示种植面积 为 0.1 畾的整数倍。

1. 平旱地、梯田和山坡地每年都只能种植一季作物:
2. 水浇地每年可以种植一季也可以种植两季作物:
3. 大棚能够在一定程度上起保温作用，每年都可以种植两季作物：
4. 平旱地、梯田和山坡地每年适宜单季种植粮食类作物 (水稻除外) :
5. 水浇地每年可以单季种植水槄或两季种植蓅菜作物:

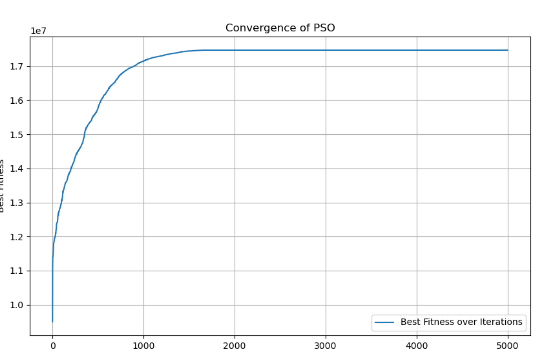
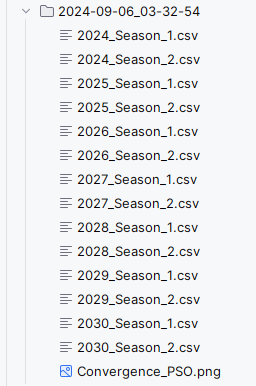
其中， 表示地块 在第 年是否种植两季蔬菜。  
7. 若在某块水浇地种植两季蓅菜，第一季可种植多种蔬菜（大白菜、白夢卜和红贾卜除外）；第二季只能种植大白菜、白夢卜和红要卜中的一种（便于管理）：

1. 根据季节性要求，大白菜、白夢卜和红贾卜只能在水浇地的第二季种植:
2. 普通大棚每年种植两季作物，第一季可种植多种蔬菜（大白菜、白夢卜和红萝卜除外），第二季只能种植食用菌：
3. 因食用菌类适应在较低且适宜的温度和湿度环境中生长，所以只能在秋冬季的普通大棚里种植：
4. 智慧大棚每年都可种植两季蔬菜 (大白菜、白萝卜和红夢卜除外) :

参考代码：

|  |
| --- |
| import numpy as np import os import pandas as pd from datetime import datetime import matplotlib.pyplot as plt  *# 粒子群算法的参数* w = 0.5 *# 惯性权重* c1 = 1.5 *# 个体学习因子* c2 = 1.5 *# 社会学习因子* num\_particles = 30 *# 粒子数量* max\_iter = 1000 *# 最大迭代次数  # 问题参数（以你的符号定义）* num\_years = 7 *# 从2024到2030年* num\_crops = 41 *# 作物种类数* num\_plots = 34 *# 地块数量* num\_seasons = 2 *# 季节数量  # 读取附件2.xlsx中的数据* file\_path = '附件2.xlsx'  *# 读取销售价格（p）、种植成本（c）、亩产量（q）等数据* data\_stats = pd.read\_excel(file\_path, sheet\_name='2023年统计的相关数据')  *# 将数据转换为字符串，以防止非字符串类型导致错误* data\_stats['销售单价/(元/斤)'] = data\_stats['销售单价/(元/斤)'].astype(str)  *# 获取作物的销售价格（p），使用区间的平均值* p = data\_stats['销售单价/(元/斤)'].apply(  lambda x: (float(x.split('-')[0]) + float(x.split('-')[1])) / 2 if '-' in x else float(x) ).values  *# 获取作物的种植成本（c）* c = data\_stats['种植成本/(元/亩)'].values  *# 获取作物的亩产量（q），将产量从‘斤’转换为‘千克’* q = (data\_stats['亩产量/斤'].values / 2).astype(float) *# 1斤 = 0.5千克  # 读取2023年农作物种植情况，假设其为预期销售量（D）* data\_crop\_situation = pd.read\_excel(file\_path, sheet\_name='2023年的农作物种植情况')  *# 计算预期销售量 D：使用种植面积/亩 乘以 对应作物的亩产量（q）* D = (data\_crop\_situation['种植面积/亩'].values \* q[data\_crop\_situation['作物编号'].values - 1])  *# 输出目标函数2定义和参数读取部分* print("目标函数2已经定义，销售价格、种植成本、亩产量、预期销售量已从附件中读取。")  *# 初始化* particles = np.random.rand(num\_particles, num\_crops, num\_plots, num\_seasons, num\_years) *# 粒子的位置* velocities = np.random.rand(num\_particles, num\_crops, num\_plots, num\_seasons, num\_years) *# 粒子的速度* p\_best = np.copy(particles) *# 每个粒子的最佳位置* g\_best = np.copy(particles[0]) *# 全局最佳位置* best\_fitness\_over\_time = []  *# 确保x是0.1的倍数* def ensure\_tenth\_multiples(x):  return np.round(x \* 10) / 10  *# 目标函数1* def objective\_function1(x):  Z1 = 0  for t in range(num\_years):  for k in range(num\_seasons):  for i in range(num\_crops):  y\_ikt = np.sum(x[i, :, k, t]) \* q[i]  Z1 += p[i] \* min(y\_ikt, D[i]) - c[i] \* np.sum(x[i, :, k, t])  return Z1 *# 由于PSO算法是求最大化问题，直接返回Z1  # 目标函数2的定义* def objective\_function2(x):  Z2 = 0  for t in range(num\_years):  for k in range(num\_seasons):  for i in range(num\_crops):  y\_ikt = np.sum(x[i, :, k, t]) \* q[i]  *# 计算收益，考虑滞销部分按折扣价出售* Z2 += p[i] \* min(y\_ikt, D[i]) + (0.5 \* p[i]) \* max(y\_ikt - D[i], 0) - c[i] \* np.sum(x[i, :, k, t])  return Z2 *# 由于PSO算法是求最大化问题，直接返回Z2  # 粒子群算法主函数* def pso():  global g\_best, p\_best, best\_fitness\_over\_time  for iter in range(max\_iter):  if iter % 100 == 0 and iter>1:  *# 绘制核心图表* plt.figure(figsize=(10, 6))  plt.plot(range(max\_iter), best\_fitness\_over\_time, label='Best Fitness over Iterations')  plt.xlabel('Iterations')  plt.ylabel('Best Fitness')  plt.title('Convergence of PSO')  plt.legend()  plt.grid()  plt.savefig(f"{folder\_name}/Convergence\_PSO.png")  plt.show()   for n in range(num\_particles):  fitness = objective\_function1(particles[n])  p\_best\_fitness = objective\_function1(p\_best[n])  g\_best\_fitness = objective\_function1(g\_best)  *# 更新个体最佳位置* if fitness > p\_best\_fitness: *# 改为最大化* p\_best[n] = particles[n]  *# 更新全局最佳位置* if fitness > g\_best\_fitness: *# 改为最大化* g\_best = particles[n]  *# 更新粒子的速度和位置* velocities[n] = w \* velocities[n] + c1 \* np.random.rand() \* (p\_best[n] - particles[n]) + c2 \* np.random.rand() \* (g\_best - particles[n])  particles[n] += velocities[n]  *# 非负性约束：强制所有位置为非负* particles[n] = np.maximum(particles[n], 0)  *# 强制x为0.1的倍数* particles[n] = ensure\_tenth\_multiples(particles[n])  *# 记录每次迭代后的全局最佳适应度* best\_fitness\_over\_time.append(g\_best\_fitness)  print("Iteration " + str(iter) + str(-g\_best\_fitness))   return g\_best  *# 运行粒子群算法* best\_solution = pso() best\_value = objective\_function1(best\_solution) *# 计算最优解对应的目标函数值  # 创建文件夹，命名为当前日期+时间* folder\_name = datetime.now().strftime('%Y-%m-%d\_%H-%M-%S') os.makedirs(folder\_name, exist\_ok=True)  *# 输出x解空间矩阵到多个csv文件* for t in range(num\_years):  for k in range(num\_seasons):  df = pd.DataFrame(best\_solution[:, :, k, t], columns=[f"P\_{j+1}" for j in range(num\_plots)], index=[f"C\_{i+1}" for i in range(num\_crops)])  file\_name = f"{folder\_name}/{2024 + t}\_Season\_{k + 1}.csv"  df.to\_csv(file\_name)  *# 绘制核心图表* plt.figure(figsize=(10, 6)) plt.plot(range(max\_iter), best\_fitness\_over\_time, label='Best Fitness over Iterations') plt.xlabel('Iterations') plt.ylabel('Best Fitness') plt.title('Convergence of PSO') plt.legend() plt.grid() plt.savefig(f"{folder\_name}/Convergence\_PSO.png") plt.show()  print("最优解对应的总收益:", best\_value) print(f"解空间矩阵已输出到文件夹: {folder\_name}") |

迭代5000次的图：

 3-1-F1

Iteration 1653-17468469.0 求出的净利润最大。（无约束代码，需要对应的添加相关约束）

|  |
| --- |
| # 施加约束的函数  def apply\_constraints(particles):  for n in range(num\_particles):  for j in range(num\_plots):  # 地块种植面积不超过总面积  for k in range(num\_seasons):  for t in range(num\_years):  if np.sum(particles[n][:, j, k, t]) > A\_j[j]:  particles[n][:, j, k, t] \*= A\_j[j] / np.sum(particles[n][:, j, k, t])  # 地块类型与作物类型的适应性约束  for i in range(num\_crops):  for k in range(num\_seasons):  for t in range(num\_years):  if plot\_types[j] in [T\_5] and k == 2 and i in [35, 36, 37]: # 普通大棚限制  particles[n][i, j, k, t] = 0  if plot\_types[j] in [T\_6] and i in [35, 36, 37]: # 智慧大棚限制  particles[n][i, j, k, t] = 0  # 禁止重茬种植约束  for i in range(num\_crops):  for k in range(num\_seasons):  for t in range(num\_years - 1):  if particles[n][i, j, k, t] > 0:  particles[n][i, j, k, t + 1] = 0  # 豆类作物种植频率限制  for t in range(num\_years - 2):  if np.sum(particles[n][i, j, :, t:t+3]) < A\_j[j]:  particles[n][i, j, :, t:t+3] = A\_j[j] / 3  # 作物种植集中度约束  for i in range(num\_crops):  for k in range(num\_seasons):  for t in range(num\_years):  if particles[n][i, j, k, t] < 0.1 and particles[n][i, j, k, t] > 0:  particles[n][i, j, k, t] = 0.1  # 种植面积为0.1的倍数的约束  for i in range(num\_crops):  for k in range(num\_seasons):  for t in range(num\_years):  particles[n][i, j, k, t] = ensure\_tenth\_multiples(particles[n][i, j, k, t])  # 粒子群算法主函数 |

第2问：

**销售量变化**

实际情况：农作物的销售量受市场需求、供应量、天气条件、运输条件、国际贸易政策等多种因素影响。一般来说，这些因素导致销售量的变化具有正态分布（正态分布能够反映市场需求的波动性），但在极端天气或政策变动下，也可能出现较大的偏差（例如，右偏或左偏的分布)。

建议分布：正态分布或偏态分布。例如，小麦和玉米每年的增长率应为正态分布 ， ），其他作物的变动在均值 和标准差 的范围内（假设在 范围内）。

**亩产量变化**

实际情况：农作物的亩产量主要受天气（如降雨量、温度）、病虫害、土壌条件等自然因素影响。常常是正态分布（天气的年际变化导致产量的波动），但会有尾部风险（如极端天气带来的灾害性减产）。

建议分布：正态分布。例如，亩产量变化可以为正态分布 ，在 的范围内。

**种植成本变化**

实际情况：种植成本受到人工成本、农资（种子、化肥、农药等）价格、燃油费等成本的变化影响，通常随着时间推移而逐渐增加。历史数据表明，这些变化也大致服从正态分布，但随着时间推移可能存在一个正的趋势（通货膨胀或市场波动）。

建议分布：正态分布。例如，种植成本的年增长率为正态分布N(5%,1%)。

**销售价格变化**

实际情况：农作物的销售价格受市场需求、竞争、进出口政策、货币政策、供应量等多重因素影响。价格变化通常具有一定的趋势性（如稳定、上升或下降），而且波动性较大。根据作物类型，这种变化可能偏向不同方向。

建议分布：

粮食类作物：稳定，保持不变。

蔬菜类作物：正态分布N( )。

食用菌类作物：下降，正态分布 。羊肚菌价格下降为固定 。

**修改后的数学模型：**引入随机变量和不确定性范围:

**销售量变化:**

小麦 (如 ) 和玉米（如 ）的预期销售量每年按 增长，定义:

其他作物的预期销售量变化范围在 :

**亩产量变化:**

农作物的亩产量每年有 的变化:

**种植成本变化:**

种植成本平均每年增长 :

**销售价格变化:**

粮食类作物的销售价格基本稳定，不变化:

蔬菜类作物的销售价格每年增长 :

食用菌类作物的销售价格每年下降 :

特别是羊肚菌 ，其价格每年下降5%:

**修改后的目标函数:**

考虑不确定性后，目标函数应当是期望总收益的最大化:

**情况1：超过部分滞销，造成浪费**

**情况2：超过部分按2023年销售价格的50%降价出售**

其中：

**参考论文：**

**1. 农作物销售量的研究**

**研究方向**：预测主要农作物的销售量，探讨了政策、气候、国际市场波动等因素对国内农产品供需关系的影响。

**曹旭光，贾海茹.（2015）**. *基于支持向量机的农产品需求预测模型及其应用*. 农业经济与管理.

**刘洋，杨敏.（2018）**. *中国主要粮食作物的供需动态及市场风险分析*. 农业经济问题.

**2. 农作物产量的研究**

**李树峰，王小俊.（2017）**. *气候变化对中国粮食作物产量的影响研究进展*. 气候变化研究进展.

**张永红，张斌.（2020）**. *基于遥感和机器学习的作物产量预测研究进展*. 地理学报.

**3. 农作物种植成本的研究**

**王建强，李少龙.（2019）**. *中国农业生产成本及收益的时间序列分析*. 农业技术经济.

**赵晨阳，张雅娟.（2020）**. *基于成本效益分析的精准农业应用研究*. 农业工程学报.

**4. 农作物销售价格的研究**

**陈超，李琳.（2016）**. *中国农产品价格波动及其影响因素分析*. 中国农村经济.

**杨丽娟，赵刚.（2019）**. *基于ARIMA模型的农产品价格预测研究*. 农村经济与科技.

**5. 综合性研究**

**黄俊，李斌.（2017）**. *基于蒙特卡罗模拟的农作物生产风险管理研究*. 农业经济问题.

**王晓红，张军.（2021）**. *中国农业经济模型的发展与应用*. 中国农业经济学报.

|  |
| --- |
| import numpy as np import os import pandas as pd from datetime import datetime import matplotlib.pyplot as plt  *# 粒子群算法的参数* w = 0.5 *# 惯性权重* c1 = 1.5 *# 个体学习因子* c2 = 1.5 *# 社会学习因子* num\_particles = 30 *# 粒子数量* max\_iter = 5000 *# 最大迭代次数* num\_simulations = 100 *# 蒙特卡罗模拟次数  # 问题参数（以你的符号定义）* num\_years = 7 *# 从2024到2030年* num\_crops = 41 *# 作物种类数* num\_plots = 34 *# 地块数量* num\_seasons = 2 *# 季节数量  # 读取附件2.xlsx中的数据* file\_path = '附件2.xlsx'  *# 读取销售价格（p）、种植成本（c）、亩产量（q）等数据* data\_stats = pd.read\_excel(file\_path, sheet\_name='2023年统计的相关数据')  *# 将数据转换为字符串，以防止非字符串类型导致错误* data\_stats['销售单价/(元/斤)'] = data\_stats['销售单价/(元/斤)'].astype(str)  *# 获取作物的销售价格（p），使用区间的平均值* p\_base = data\_stats['销售单价/(元/斤)'].apply(  lambda x: (float(x.split('-')[0]) + float(x.split('-')[1])) / 2 if '-' in x else float(x) ).values  *# 获取作物的种植成本（c）* c\_base = data\_stats['种植成本/(元/亩)'].values  *# 获取作物的亩产量（q），将产量从‘斤’转换为‘千克’* q\_base = (data\_stats['亩产量/斤'].values / 2).astype(float) *# 1斤 = 0.5千克  # 读取2023年农作物种植情况，假设其为预期销售量（D）* data\_crop\_situation = pd.read\_excel(file\_path, sheet\_name='2023年的农作物种植情况')  *# 计算预期销售量 D：使用种植面积/亩 乘以 对应作物的亩产量（q）* D\_base = (data\_crop\_situation['种植面积/亩'].values \* q\_base[data\_crop\_situation['作物编号'].values - 1])  *# 输出目标函数2定义和参数读取部分* print("目标函数2已经定义，销售价格、种植成本、亩产量、预期销售量已从附件中读取。")  *# 初始化* particles = np.random.rand(num\_particles, num\_crops, num\_plots, num\_seasons, num\_years) *# 粒子的位置* velocities = np.random.rand(num\_particles, num\_crops, num\_plots, num\_seasons, num\_years) *# 粒子的速度* p\_best = np.copy(particles) *# 每个粒子的最佳位置* g\_best = np.copy(particles[0]) *# 全局最佳位置* best\_fitness\_over\_time = [] iterations = [] *# 用于保存对应的迭代次数   # 确保x是0.1的倍数* def ensure\_tenth\_multiples(x):  return np.round(x \* 10) / 10   *# 模拟参数的生成函数* def simulate\_parameters():  p\_sim = np.copy(p\_base)  c\_sim = np.copy(c\_base)  q\_sim = np.copy(q\_base)  D\_sim = np.copy(D\_base)  *# 小麦和玉米预期销售量的增长* for t in range(1, num\_years + 1):  for i in range(num\_crops):  if i in [5, 6]: *# 小麦和玉米* D\_sim[i] \*= (1 + np.random.normal(0.075, 0.015)) *# 正态分布 N(7.5%, 1.5%)* else:  D\_sim[i] \*= (1 + np.random.normal(0, 0.03)) *# 正态分布 N(0%, 3%)  # 亩产量变化* q\_sim[i] \*= (1 + np.random.normal(0, 0.05)) *# 正态分布 N(0, 5%)  # 种植成本增长* c\_sim[i] \*= (1 + np.random.normal(0.05, 0.01)) *# 正态分布 N(5%, 1%)  # 销售价格变化* if i <= 14: *# 粮食类作物* p\_sim[i] = p\_base[i] *# 稳定不变* elif i <= 36: *# 蔬菜类作物* p\_sim[i] \*= (1 + np.random.normal(0.05, 0.02)) *# 正态分布 N(5%, 2%)* elif i < 40: *# 食用菌类作物* p\_sim[i] \*= (1 - np.random.normal(0.03, 0.01)) *# 正态分布 N(-3%, 1%)* elif i == 40: *# 羊肚菌* p\_sim[i] \*= (1 - 0.05) *# 固定下降5%* return p\_sim, c\_sim, q\_sim, D\_sim   *# 目标函数1* def objective\_function1(x, p\_sim, c\_sim, q\_sim, D\_sim):  Z1 = 0  for t in range(num\_years):  for k in range(num\_seasons):  for i in range(num\_crops):  y\_ikt = np.sum(x[i, :, k, t]) \* q\_sim[i]  Z1 += p\_sim[i] \* min(y\_ikt, D\_sim[i]) - c\_sim[i] \* np.sum(x[i, :, k, t])  return Z1 *# 由于PSO算法是求最大化问题，直接返回Z1   # 粒子群算法主函数* def pso():  global g\_best, p\_best, best\_fitness\_over\_time  for iter in range(max\_iter):  if iter % 100 == 0 and iter > 1:  *# 绘制核心图表* plt.figure(figsize=(10, 6))  plt.plot(iterations, best\_fitness\_over\_time, label='Best Fitness over Iterations')  plt.xlabel('Iterations')  plt.ylabel('Best Fitness')  plt.title('Convergence of PSO')  plt.legend()  plt.grid()  plt.show()   for n in range(num\_particles):  *# 进行多次模拟并求期望* expected\_value = 0  for \_ in range(num\_simulations):  p\_sim, c\_sim, q\_sim, D\_sim = simulate\_parameters()  fitness = objective\_function1(particles[n], p\_sim, c\_sim, q\_sim, D\_sim)  expected\_value += fitness  expected\_value /= num\_simulations   *# 更新个体最佳位置* if expected\_value > objective\_function1(p\_best[n], p\_sim, c\_sim, q\_sim, D\_sim):  p\_best[n] = particles[n]  *# 更新全局最佳位置* if expected\_value > objective\_function1(g\_best, p\_sim, c\_sim, q\_sim, D\_sim):  g\_best = particles[n]  *# 更新粒子的速度和位置* velocities[n] = w \* velocities[n] + c1 \* np.random.rand() \* (  p\_best[n] - particles[n]) + c2 \* np.random.rand() \* (g\_best - particles[n])  particles[n] += velocities[n]  *# 非负性约束：强制所有位置为非负* particles[n] = np.maximum(particles[n], 0)  *# 强制x为0.1的倍数* particles[n] = ensure\_tenth\_multiples(particles[n])   *# 记录每次迭代后的全局最佳适应度* best\_fitness\_over\_time.append(objective\_function1(g\_best, p\_sim, c\_sim, q\_sim, D\_sim))  iterations.append(iter)  print("Iteration " + str(iter) + str(-objective\_function1(g\_best, p\_sim, c\_sim, q\_sim, D\_sim)))   return g\_best   *# 运行粒子群算法* best\_solution = pso() best\_value = objective\_function1(best\_solution, p\_base, c\_base, q\_base, D\_base) *# 计算最优解对应的目标函数值  # 创建文件夹，命名为当前日期+时间* folder\_name = datetime.now().strftime('%Y-%m-%d\_%H-%M-%S') os.makedirs(folder\_name, exist\_ok=True)  *# 输出x解空间矩阵到多个csv文件* for t in range(num\_years):  for k in range(num\_seasons):  df = pd.DataFrame(best\_solution[:, :, k, t], columns=[f"P\_{j + 1}" for j in range(num\_plots)],  index=[f"C\_{i + 1}" for i in range(num\_crops)])  file\_name = f"{folder\_name}/{2024 + t}\_Season\_{k + 1}.csv"  df.to\_csv(file\_name)  *# 绘制核心图表* plt.figure(figsize=(10, 6)) plt.plot(range(max\_iter), best\_fitness\_over\_time, label='Best Fitness over Iterations') plt.xlabel('Iterations') plt.ylabel('Best Fitness') plt.title('Convergence of PSO') plt.legend() plt.grid() plt.savefig(f"{folder\_name}/Convergence\_PSO.png") plt.show()  print("最优解对应的总收益:", best\_value) print(f"解空间矩阵已输出到文件夹: {folder\_name}") |

第3问：

定义协方差矩阵来捕捉销售量、销售价格和种植成本之间的线性关系：

其中:

* 表示预期销售量的方差。
* 表示销售价格的方差。
* 表示种植成本的方差。
* 和 分别表示预期销售量与销售价格、预期销售量与种植成本、销售价格与种植成本之间的协方差。

使用多元线性回归来捕捉这些变量之间的关系。例如，我们可以假设预期销售量 是销售价格 和种植成本 的线性函数:

同样，销售价格 和种植成本 也可以分别作为其他两个变量的函数来进行建模。

由于销售量、价格和成本均随时间变化，我们还可以使用时间序列分析模型（如ARIMA、VAR等）来更好地描述它们之间的动态关系。

|  |
| --- |
| import pandas as pd import numpy as np import matplotlib.pyplot as plt import torch import torch.nn as nn from torch.utils.data import DataLoader, TensorDataset from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import mean\_squared\_error, r2\_score  *# 设置随机种子* torch.manual\_seed(42)  *# 读取数据* data = pd.read\_csv('拟合数据.csv')  *# 提取自变量（销售价格和种植成本）和因变量（预期销售量）* X = data[['销售价格', '种植成本']].values Y = data['预期销售量'].values  *# 标准化特征* scaler\_X = StandardScaler() scaler\_Y = StandardScaler()  X = scaler\_X.fit\_transform(X) Y = scaler\_Y.fit\_transform(Y.reshape(-1, 1)).flatten()  *# 拆分数据集为训练集和测试集* X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)  *# 转换为PyTorch张量* X\_train\_tensor = torch.tensor(X\_train, dtype=torch.float32) Y\_train\_tensor = torch.tensor(Y\_train, dtype=torch.float32).view(-1, 1) X\_test\_tensor = torch.tensor(X\_test, dtype=torch.float32) Y\_test\_tensor = torch.tensor(Y\_test, dtype=torch.float32).view(-1, 1)  *# 创建数据加载器* train\_dataset = TensorDataset(X\_train\_tensor, Y\_train\_tensor) train\_loader = DataLoader(train\_dataset, batch\_size=16, shuffle=True)  *# 定义神经网络模型* class MLP(nn.Module):  def \_\_init\_\_(self):  super(MLP, self).\_\_init\_\_()  self.hidden1 = nn.Linear(2, 64)  self.hidden2 = nn.Linear(64, 32)  self.output = nn.Linear(32, 1)   def forward(self, x):  x = torch.relu(self.hidden1(x))  x = torch.relu(self.hidden2(x))  x = self.output(x)  return x  model = MLP()  *# 定义损失函数和优化器* criterion = nn.MSELoss() optimizer = torch.optim.Adam(model.parameters(), lr=0.01)  *# 训练模型* epochs = 500 for epoch in range(epochs):  model.train()  for X\_batch, Y\_batch in train\_loader:  optimizer.zero\_grad()  Y\_pred = model(X\_batch)  loss = criterion(Y\_pred, Y\_batch)  loss.backward()  optimizer.step()   *# 打印训练进度* if (epoch + 1) % 50 == 0:  print(f"Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}")  *# 预测和评估* model.eval() with torch.no\_grad():  Y\_test\_pred = model(X\_test\_tensor).numpy()  *# 反标准化预测值* Y\_test\_pred = scaler\_Y.inverse\_transform(Y\_test\_pred) Y\_test\_actual = scaler\_Y.inverse\_transform(Y\_test\_tensor.numpy())  *# 评估模型* mse = mean\_squared\_error(Y\_test\_actual, Y\_test\_pred) r2 = r2\_score(Y\_test\_actual, Y\_test\_pred)  print(f"均方误差: {mse:.2f}") print(f"R^2值: {r2:.2f}")  *# 绘制实际值与预测值的散点图* plt.figure(figsize=(10, 6)) plt.scatter(Y\_test\_actual, Y\_test\_pred, color='blue', label='预测值 vs 实际值') plt.plot([Y\_test\_actual.min(), Y\_test\_actual.max()], [Y\_test\_actual.min(), Y\_test\_actual.max()], color='red', linestyle='--', label='理想拟合线') plt.xlabel('实际销售量') plt.ylabel('预测销售量') plt.title('实际值与预测值对比图（神经网络）') plt.legend() plt.grid(True) plt.show() |