基于模拟实验的生产过程中的决策问题研究

摘　要

科学的决策方案能够有效地控制产品质量，降低生产成本。通过合理设计抽样检测方案和检测策略，企业可以及时发现和排除不合格零配件，避免将次品流入市场，减少返工和拆解的成本，从而实现成本效益最大化。

针对问题一，为了在不同的信度下检测样本的次品率与供应商声称的标称值进行临界判断，并设计一个尽可能少的抽样检测方案，首先基于二项分布的正态近似进行假设检验。随后，利用序贯概率比检验（SPRT），根据不同的置信度设置相应的上界和下界，以标称值作为可接受水平（AQL），企业质量要求设置拒绝质量水平（RQL）。经模拟结果表明，在95%置信度下，最少需要75个样本进行检测；在90%置信度下，最少需要48个样本。

针对问题二，为了给出题中表1中六种情形的具体决策方案，首先需对题目条件进行可视化理解。然后，基于决策树分析进行模拟实验。定义四个主要决策分支：即“零件1和零件2均检测”、“零件1检测但零件2不检测”、“零件1不检测但零件2检测”，以及“零件1和零件2均不检测”。在这些主要分支下，进一步细分为六种情况的分支。每种情况又细分为四种具体方案：即“成品检测-不合格品拆解”、“成品检测-不合格品不拆解”、“成品不检测-不合格品拆解”和“成品不检测-不合格品不拆解”。

假设初始零配件数量均为100个，对共计96个方案进行了模拟，结果如表3所示。表中结果不仅展示了不同方案的效益，还间接反映出“次品率”、“检测成本”、“调换损失”和“拆解费用”对方案的影响。

针对问题三，采用蒙特卡洛进行模拟实验与遗传算法进行决策优化，通过蒙特卡洛模拟，可以系统地评估不同次品率下的决策方案表现，从而识别潜在的风险和问题。结合遗传算法的优化能力，可以在广泛的决策空间中寻找最优方案，提升决策的整体效果和收益。

针对问题四，在重新完成问题2和问题3时，需要将抽样误差和次品率的不确定性纳入考虑，重新评估原有的决策方案，使用鲁棒性分析模型评估在存在不确定性和扰动的情况下，决策方案的稳定性和可靠性。

关键词：假设检验 SPRT 模拟实验 决策树分析 蒙特卡洛模拟 遗传算法

一、 问题重述

1.1 问题背景

某企业生产某种畅销的电子产品，需要分别购买两种零配件（零配件 1 和零配件 2）。在组装成品时，只要其中一个零配件不合格，成品必定不合格。

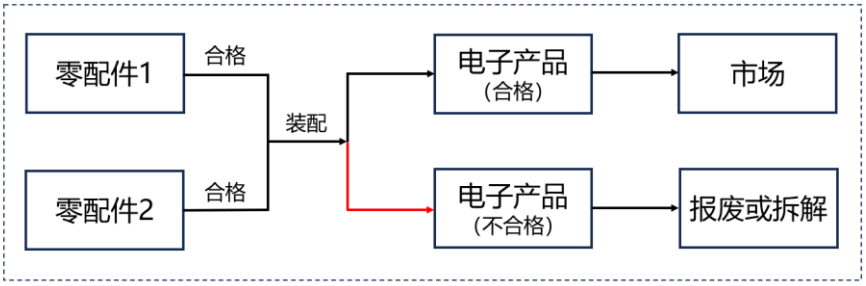

图 1 生产流程图

如果两个零配件都合格，成品仍可能不合格。对于不合格的成品，企业可以选择报废或拆解该成品，拆解过程中不会损坏零配件，但需要支付拆解费用。

故在生产过程中，制定决策有利于最大化利润。权衡生产和拆解成本，与不合格成品的报废成本进行比较。通过优化零配件的质量检验、控制拆解和报废的决策，可以减少整体成本，提高生产效率，从而达到经济上的最终利益。

1.2 问题提出

基于上述背景，利用数学模型对下列问题进行研究并求解。

问题 1：供应商声明某批零配件的次品率不会超过 10%。企业需设计一个抽样检测方案，以尽可能减少检测次数，为以下两种情形设计抽样检测方案：

1. 拒收条件：在 95%的置信水平下，如果检测结果显示次品率超过 10%，则拒收这批零配件。
2. 接收条件：在 90%的置信水平下，如果检测结果显示次品率不超过 10%，则接收这批零配件。

问题 2：已知两种零配件和成品的次品率，为企业生产过程的各个阶段制定决策：

1. 对零配件（零配件 1 和/或零配件 2）是否进行检测。如果不检测，这些零配件将直接进入装配环节；如果检测，丢弃检测出的不合格零配件；
2. 对每一件装配好的成品是否进行检测。如果不检测，装配后的成品直接进入市场；否则，只有检测合格的成品进入市场；
3. 对检测出的不合格成品是否进行拆解。如果不拆解，直接丢弃不合格成品；否则，对拆解后的零配件重复步骤 1 和步骤 2；

4. 对用户购买的不合格品，企业将无条件调换，并产生调换损失（如物流成本、企业信誉等）。对退回的不合格品，重复步骤 3。

根据决策方案和相应表中的具体情形给出具体方案及其依据和指标结果。

问题 3：根据图 1 所展示的两道工序、8 个零配件的组装情况，以及相应表中企业在生产中遇到的具体情况，同时考虑已知的零配件、半成品和成品的次品率，为具有 m 道工序和 n 个零配件的生产过程设计具体的决策方案。分析决策依据和相关指标。

问题 4：假设问题 2 和问题 3 中所提及的零配件、半成品和成品的次品率数据均是问题 1 中使用的方法获得的。基于这一假设，重新考虑并完成问题 2 和问题 3。

二、 模型假设

- 假设零配件次品率服从正态分布；
- 假设零配件数据处理为向上取整；
- 假设零配件估计误差选取值合理；
- 假设零配件不检测的次品数依次品率计算；
- 假设零配件生产过程中多余零件暂时存储；
- 假设零配件生产过程后多余零件进行存储；
- 假设零配件第二次装配后必定为合格品；
- 假设企业有一个仓库用以调换不合格品。

三、 符号说明

符号	说明
n	抽样的零配件数量
p	零配件次品率（实际值）
p_0	零配件次品率（标称值）
X	检测到的次品数量
E	估计误差
$parts1$	零配件 1 的数量
$parts2$	零配件 2 的数量
p_i	零配件 1、2 和成品次品率
W	单个方案的收益
D	拆解费用
I	装配费用
L	调换损失

四、 模型的建立与求解

4.1 问题一模型的建立与求解

4.1.1 问题分析

企业在设计抽样检测方案时需要满足两个主要条件。首先，对于拒收条件，目标是在 95%的置信水平下，如果检测结果显示次品率超过 10%，则拒收该批零配件。这要求样本量足够，以在高置信水平下准确识别次品率过高的情况。其次，对于接收条件，目标是在 90%的置信水平下，如果检测结果显示次品率不超过 10%，则接收该批零配件。这要求样本量适当，以在较低置信水平下确认次品率符合标准。

为实现这些目标，可以基于二项分布的正态近似进行假设检验，并使用序贯概率比检验（SPRT）来模拟所需的最小样本量，从而优化检测的准确性和有效性。

4.1.2 假设检验估计

二项分布

二项分布描述了在一系列相同的试验中，每次试验只有两种结果（例如成功或失败），并且每次试验成功的概率是相同的情况下，成功的次数可能是多少。比如，如果抛一枚硬币多次，每次抛硬币的结果只有“正面”或“反面”两种可能，那么硬币抛出的正面次数就是一个二项分布的例子。

由于次品率是一个概率事件，假设从该批零配件中随机抽取样本，样本大小为 n，次品率（标称值）为 p，则次品的数量服从二项分布，即：

$$X \sim Binomial\left(n, p\right) \tag{1}$$

正态分布近似

正态分布近似是指在样本量足够大的情况下，许多类型的随机变量的分布会接近正态分布。比如，根据中心极限定理，当独立随机变量的样本量增加时，它们的平均值的分布趋向于正态分布。这意味着即使原始数据分布不是正态的，样本均值的分布也会趋向于正态分布。

$$X \sim N\left(np, \sqrt{np\left(1-p\right)}\right) \tag{2}$$

采用正态分布近似来估算样本大小，样本容量足够大时，二项分布可以近似为正态分布。

随后进行假设检验：

情形 1：在 95%的信度下认定零配件次品率超过标称值则拒收这批零配件

1. 建立假设：

- 零假设（H_0）：次品率 $p \le 0.1$
- 备择假设（H_1）：次品率 $p > 0.1$

2. 检验统计量

$$Z = \frac{p - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} \sim N(0,1) \tag{3}$$

3. 确定临界值

根据置信水平（95%对应的显著性水平 $a = 0.05$），查找标准正态分布表中的临界值。

$$Z_{0.05} = 1.65 \tag{4}$$

4. 做出决策

如果 $p > \frac{0.4935}{\sqrt{n}} - 0.1$，则拒绝零假设，认为次品率显著高于 10%，拒收这批零配件。

如果 $p \le \frac{0.4935}{\sqrt{n}} - 0.1$，则不能拒绝零假设，接受这批零配件。

情形 2：在 90%的信度下认定零配件次品率不超过标称值则接收这批零配件

假设检验同**情形 1**，决策为：

如果 $p > \frac{0.384}{\sqrt{n}} - 0.1$，则拒绝零假设，认为次品率显著高于 10%，拒收这批零配件。

如果 $p \le \frac{0.384}{\sqrt{n}} - 0.1$，则不能拒绝零假设，接受这批零配件。

样本量估计

假设估计误差 $E = 0.05$，则估算样本量公式如下：

$$n=\frac{\left(Z_{a/2}\right)^2 p_0\left(1-p_0\right)}{E^2} \tag{5}$$

代入 $E=0.05$ 和 $p_0=0.1$ 并根据不同置信度水平下求得的样本数（向上取整）为：

$$\begin{cases} n=139, a=0.05 \\ n=99, a=0.1 \end{cases} \tag{6}$$

即在 95%置信度下，检测样本量至少为 139 个；在 90%置信度下，检测样本量至少为 99 个。

4.1.3 SPRT 模型估计

SPRT 模型

SPRT（Sequential Probability Ratio Test，序贯概率比检验）模型是一种用于统计假设检验的序贯检验方法，主要用于在逐步获得数据的过程中决定是否接受或拒绝零假设。其核心思想是通过实时计算数据的概率比值来进行决策，而不是在所有数据收集完成后再进行检验。

模型特点：

序贯性：检验过程在每一步都可以做出决策，而不需要等到所有样本数据收集完成。这种方法特别适用于需要快速决策的场景。

概率比：计算样本数据下的似然比（或概率比），以此与预设的临界值进行比较。

临界界限：设定两个临界界限，分别对应接受和拒绝零假设的阈值。如果比值超出界限，则做出相应的决策。

效率：通常需要较少的样本量即可做出决策，提升了统计检验的效率。

应用流程：

- 设定假设：设定零假设 H_0 和备择假设 H_1
- 选择显著性水平：确定误差概率（如 a 和 β ）
- 计算似然比：随着样本数据的积累，计算当前数据下的似然比

$$\ln L=\sum_{x=1}^{n}\left[x\cdot\log\left(\frac{p_1}{p_0}\right)+\left(1-x\right)\cdot\log\left(\frac{1-p_1}{1-p_0}\right)\right] \tag{7}$$

➢ 比较界限：将计算的似然比与设定的临界界限比较，决定是否接收或者继续采集更多数据

➢ 做出决策：根据比较结果做出最终的检验决策

置信度为 90%（95%）时，SPRT（顺序概率比检验）的a（显著性水平）和β（检验功效）设置通常为$a=0.1$（$a=0.05$）和$\beta=0.1$（$\beta=0.05$）。这意味着在零假设为真时，有 10%的概率错误地拒绝零假设（第一类错误），而在备择假设为真时，有 10%的概率错误地接受零假设（第二类错误）。

根据模拟实验求解结果如下图所示：

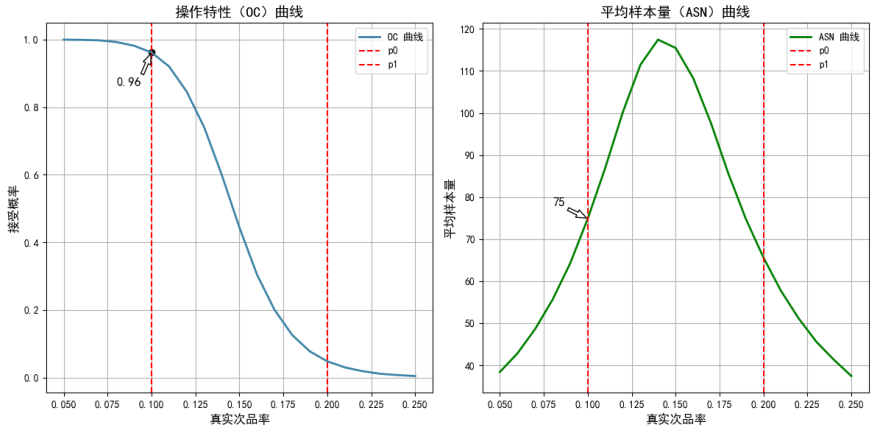

图 2 SPRT 模型模拟结果

假设估计与 SPRT 估计对比如下表所示：

表 1 假设估计与 SPRT 估计结果对比

置信区间	95%	90%
假设估计	139	99
SPRT 估计	**75**	**48**

SPRT（顺序概率比检验）是一种在逐步采集数据的过程中动态决策的抽样检测方案，而不需要在开始时就确定样本量。这种方法根据当前样本数据的累积结果不断更新对假设的信念，从而决定是否继续采样或停止检验。SPRT 的主要优点在于它能够根据实时数据做出灵活的决策，从而可能减少所需的样本量，提高检验效率。

模拟实验结果显示，在 95%置信度下，使用 SPRT 所需的最少样本量为 75 个；而在 90%置信度下，所需的最少样本量为 48 个。这些样本量均低于传统假设检验所需的样本量。

4.2 问题二模型的建立与求解

4.2.1 问题分析

已知两种零配件和成品次品率，为企业生产过程中的各个阶段做出决策，首先将各个阶段决策过程进行可视化如图 3 所示。

由图分析之，在企业生产过程中，可能出现的情况为经过检测后零件 1 的数量与零件 2 的数量不等，即：

$$parts1 \neq parts2 \tag{8}$$

在生产过程中遇到零件数量不匹配时可以存入临时仓库，用于后续循环准备，如果在生产过程之后有多余零件，则假设生产过程中的终止条件为至少一个零配件的数量为 0。

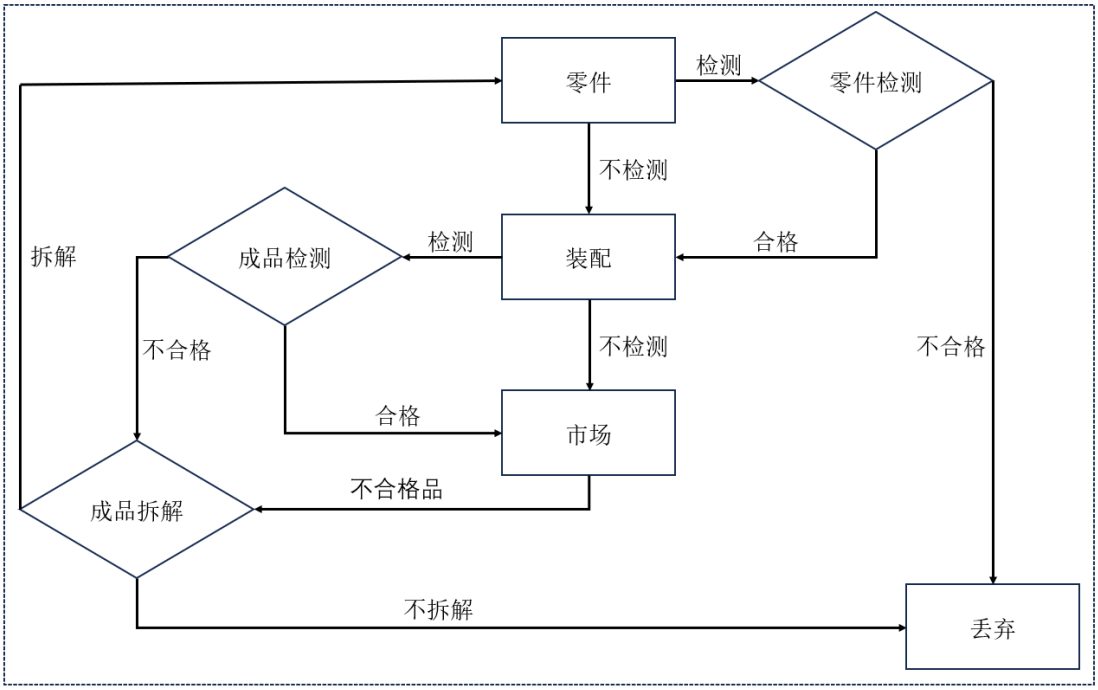

图 3 决策过程可视化

此外，本文假设企业有一个仓库，仓库中存有可以满足调换数量的合格品用以应对调换需求，但按其售价计算其损失。

4.2.2 模型建立

采用基于决策树分析与模拟实验建立模型，首先将整个生产流程进行决策树可视化如下图所示：

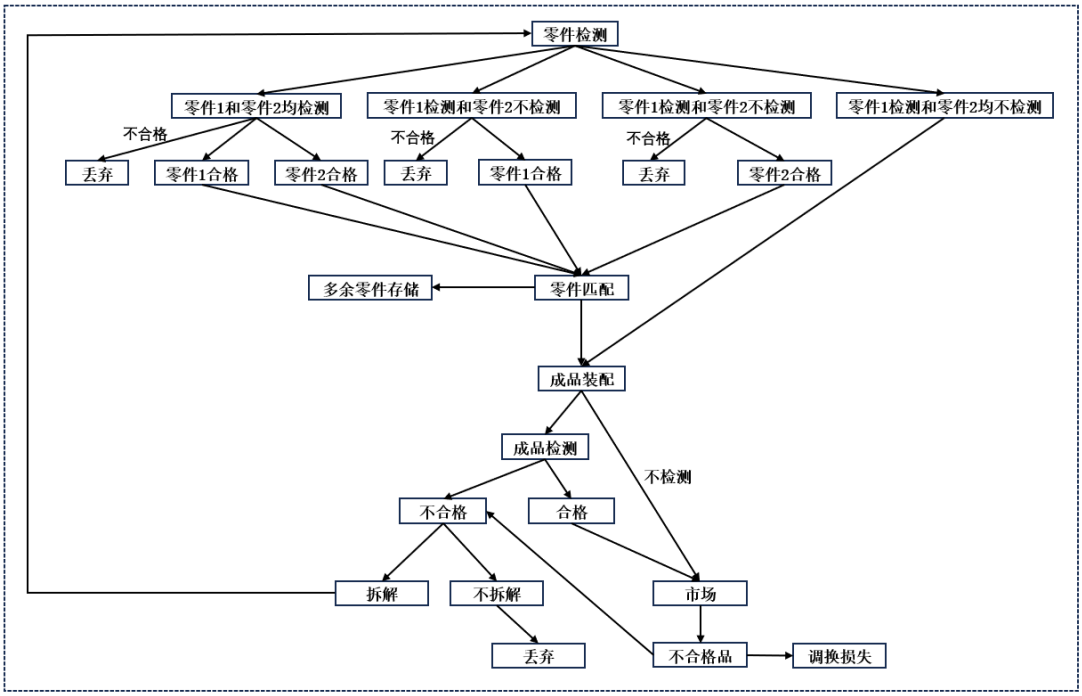

图 4 生产流程可视化

针对问题二，这里简化其流程逻辑，即循环到步骤 1 的过程，例如：

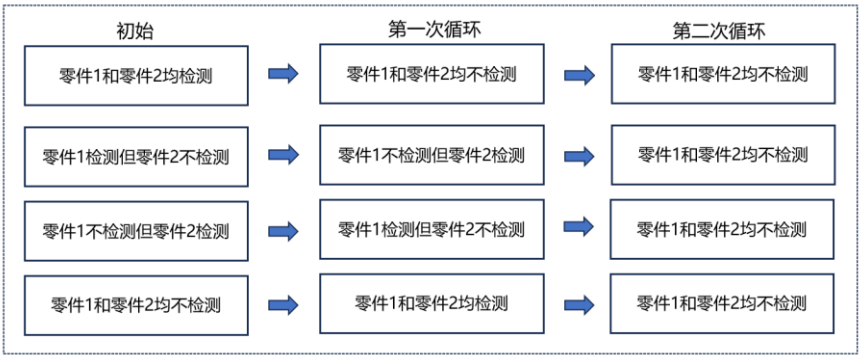

图 5 步骤简化

即对零件进行步骤 1（是否对零件进行检测）时会分为四种情况，但第二次进行步骤一时，再次检测的方案为对步骤 1 的互补，第三次进行步骤 1 时，由于零件 1 与零件 2 均已检测过，故无需进行检测，直接进入到装配环节。

在此基础上简化其生产流程，首先定义四个第一分支，即初始是否对零件进行检测，分为“零件 1 和零件 2 均检测”、“零件 1 检测但零件 2 不检测”、“零件 1 不检测但零件 2 检测”和“零件 1 和零件 2 均不检测”。对第一分支下定义四个第二分支，即“成品检测-不合格品拆解”、“成品检测-不合格品不拆解”、“成

品不检测-不合格品拆解”和“成品不检测-不合格品不拆解”，用以简化流程，最后定义 6 种情况对应的第三分支。

共有 96 种方案：

$$第一分支4\times第二分支4\times第三分支6=96 \tag{9}$$

4.2.3 模型建立

观察题目中的表 1，发现其变动的因素只有次品率、检测成本、调换损失和拆解费用，故定义单个方案的收益函数如下：

$$W=f\left(p,D,I,L\right) \tag{10}$$

其中 p 为零配件次品率，D 为拆解费用，I 为装配费用，L 为调换损失。

设定初始零件数为 100，通过模拟并对比 96 种方案的最大收益来确定 6 种情况下的最佳方案。

$$情况_{i}=MAX\left(W_1,W_2...W_{16}\right) \tag{11}$$

4.2.4 问题求解

对于第一分支，检测零件 1 和零件 2 时，由于情况 1、2、3、4、6 中零件 1 和零件 2 的次品率一致，因此在开始阶段即可计算成本，包括购买成本、检测成本和装配成本。由于零件 1 和零件 2 已经完成检测，因此在后续流程中无需重复检测这些零件，只需关注是否需要拆解和再次安装的问题。

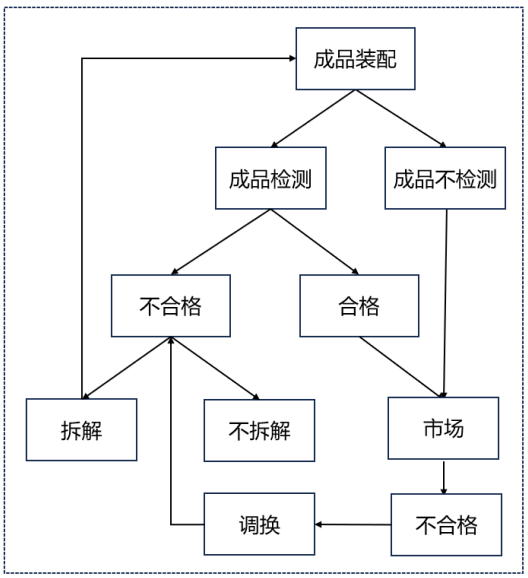

图 6 第一分支流程图

在此假设下，合格的零配件第一次组装时定位不合格品的原因主要是安装失误。由于第二次安装时安装失误的概率极小，因此可以简化流程如下图 6 所示。

在第一分支流程中，共分为“成品检测-不合格品拆解”、“成品检测-不合格品不拆解”、“成品不检测-不合格品拆解”和“成品不检测-不合格品不拆解”四种方案，其中由于流程中不合格品的零配件均为合格品，所以假设第二次装配时出现错误的概率较小，故简化为第二次装配必定为合格品。

对于情况 5，其在第一分支中要与其它情况进行区分，由于零配件 1 与零配件 2 的次品率不同，则必然会造成零件匹配不对等的做法，本文采取的方案是，将零件匹配时多余零件进行暂时存储，等到再次需要零件时进行填补（适用于第一分支零件 1 检测但零件 2 不检测和零件 1 不检测但零件 2 检测的情况），在生产流程之后，多余的零件存储在仓库中。

基于此，对 96 种方案进行模拟，结果如下（仅展示第一分支零件 1 和零件 2 均检测的情况）：

表 2 零件 1 和零件 2 均检测模拟实验结果

方案	情况	收益（元）	情况	收益（元）
检测-拆解	情况 1	1431 元	情况 4	1264 元
检测-不拆解		1026 元		544 元
不检测-拆解		1647 元		944 元
不检测-不拆解		1242 元		224 元
检测-拆解	情况 2	884 元	情况 5	652 元
检测-不拆解		164 元		292 元
不检测-拆解		1028 元		732 元
不检测-不拆解		308 元		372 元
检测-拆解	情况 3	1431 元	情况 6	1546 元
检测-不拆解		1026 元		1499 元
不检测-拆解		1431 元		1784 元
不检测-不拆解		1026 元		1720 元

在对 96 种方案进行模拟并对比各种情形的最大效益及方案结果如下表所示：

表 3 6 种情况最佳方案

情况	最佳方案	收益（元）
情况 1	**检测**零件 1 但**不检测**零件 2 成品**不检测**-不合格品**拆解**	1649 元
情况 2	对零件 1 和零件 2 **均进行检测** 成品进行**检测**-不合格品**拆解**	1028 元
情况 3	对零件 1 和零件 2 **均不检测** 成品进行**检测**-不合格品**拆解**或**不拆解**	1444 元
情况 4	对零件 1 和零件 2 均进行**检测** 成品进行**检测**-不合格品**拆解**	1264 元
情况 5	**不检测**零件 1 但**检测**零件 2 成品**不检测**-不合格品**拆解**	1440 元
情况 6	对零件 1 和零件 2 **均不检测** 成品进行**检测**-不合格品**拆解**	2016 元

从表 3 可以看出，影响方案的变量包括次品率、检测成本、调换损失和拆解费用。通过分析 6 种情况的具体决策方案，可以发现，对不合格品进行拆解后可重新销售，而且拆解与装配的费用低于其售价，即使在情况 6 中拆解费用达到 40，但拆解与装配费用总和仍低于售价，因此拆解后再次组装能提升收益。而在情况 3 中，拆解与不拆解的最终收益一致。

其中，在情形 5 中，由于零配件 1 的检测成本显著高于其他情况，因此为了控制成本，决定不对零配件 1 进行检测，而是将检测资源集中于零配件 2。这样可以降低总体检测费用，同时保持对关键零配件的质量控制，从而优化成本效益。

此外，从表中得出的结论是，次品率越高，对检测的需求也越大。次品率越高意味着生产过程中出现的缺陷产品增加，这需要更频繁的检测来确保质量，减少不合格品的流入市场。检测可以帮助及时发现问题，降低潜在的调换损失和售后服务成本，从而提升整体质量控制和盈利能力。

4.3 问题三模型的建立与求解

4.3.1 问题分析

针对有 m 道工序和 n 个零配件的生产过程，制定决策方案需考虑零配件、半成品和成品的次品率及其检测成本。首先，分析每道工序对零配件的处理效果，评估工序 1 和工序 2 的次品率影响。接着，依据每个零配件的次品率和检测成本，决定是否进行检测。次品率高的零配件应优先检测，半成品如果次品率较高，则需在工序 2 前进行检测。最终，评估检测成本与因次品导致的调换损失，结合拆解和重组的收益，优化检测方案。例如，如果某零配件的次品率较高且检测成本合理，应进行检测，而次品率低的零配件则可以减少检测频率。通过这种综合分析，可以在确保生产效率和产品质量的同时，控制成本并提高收益。

4.3.2 模型建立

蒙特卡洛模拟

蒙特卡洛模拟是一种通过随机采样来估计复杂系统行为的计算方法。其核心在于通过大量随机样本来模拟和分析系统的性能，评估可能的结果和风险。这种方法得名于摩纳哥的蒙特卡洛赌场，因为它涉及大量的随机性和概率。其基本步骤包括定义问题、构建数学模型、生成随机样本、运行模型并汇总结果。通过这些步骤，可以估计输出变量的概率分布、均值和方差等特性。蒙特卡洛模拟广泛应用于金融工程、项目管理、工程与制造、供应链管理以及科学研究等领域，因其灵活性和处理复杂性的能力而受到重视。然而，它也有计算成本高和结果不确定性的缺点，需要足够的计算资源和多次模拟以获得稳定的结果。

遗传算法

遗传算法是一种模拟自然选择和遗传机制的优化方法，用于解决复杂的优化问题。其基本过程包括初始化种群、评估适应度、选择、交叉和变异。算法从一组候选解（种群）开始，通过选择适应度较高的个体进行繁殖，生成新的解（后代）。交叉操作模拟基因重组，而变异则引入随机性以增加多样性。经过多次迭代，遗传算法能逐步找到最优或接近最优的解。该算法广泛应用于工程设计、机器学习、路径规划等领域。

```
初始化种群
种群大小 = N
个体长度 = 16  // 决策变量 X 的长度
// 生成初始种群
对于每个个体 i 从 1 到 N:
    个体[i] = 随机生成一个 16 位的二进制向量
定义适应度函数（计算总成本和总利润）
函数 计算适应度(个体):
    总成本 = 0
    总利润 = 0
    // 模拟过程，根据个体的二进制值进行决策
    // 1. 抽取零部件，检测和处理
    // 2. 合成半成品，检测和拆解
    // 3. 处理成品，检测和拆解
    返回 总利润 - 总成本
迭代直到满足终止条件:
    评估每个个体的适应度
    选择操作:
        选择适应度较高的个体作为父代
    交叉操作:
        对选择的父代个体进行交叉，生成新个体
    变异操作:
        对新个体进行随机变异
    替换操作:
        用新个体替换旧个体
    更新种群
输出最佳个体
```

图 7 算法流程

基于蒙特卡洛进行模拟实验与遗传算法进行决策优化的流程如图所示。

决策变量 X 是一个由 16 个二进制位组成的向量，具体结构如下：前 8 位用于表示是否对 8 个零件进行检测（1 表示检测，0 表示不检测）；接下来的 6 位中，前 3 位表示是否对半成品 1、2、3 进行检测，后 3 位表示是否对不合格的半成品 1、2、3 进行拆解；最后的 2 位分别表示是否对成品进行检测和拆解。

在问题三中，按照每个零部件的次品率随机生成指定数量的零部件，并记录每个零部件的状态（合格或不合格）以及检测状态（检测或不检测）。利用蒙特卡洛模拟，从零部件中抽取样本，根据决策变量 X 对应的二进制位进行决策。检测时，通过动态记录当前检测成本，处理不合格的产品并计入成本，检测合格的产品则更新其状态。

在生产决策过程中，根据决策变量 X 的二进制位对半成品进行检测和拆解，将拆解后的半成品放回零件库，形成回退机制。同样，成品的处理也遵循相同的决策过程。

为求解这个优化问题，将其抽象为二进制优化问题，通过遗传算法进行求解。

遗传算法在处理离散问题上相较于粒子群优化等连续优化方法表现更佳。具体而言，遗传算法将每个可行解 X 视作一个个体，n 个个体组成一个种群。算法通过模拟自然界的进化过程，运用交叉、变异和替换操作来更新个体。通过适应度函数评估个体，经过多次迭代，最终找到一个最优解，实现总成本最小化和总利润最大化的目标。

4.3.3 问题求解

实验结果（部分）如下表所示：

表 4 问题三实验结果

序号	**方案**	收益（元）
1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0]	8760 元
2	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]	8760 元
3	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0]	9040 元
4	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]	9020 元
6	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]	8800 元

方案列表中的前 8 位表示零件是否需要检测，1 代表需要检测，0 代表不需要检测。第 9 至 11 位表示半成品是否需要检测，第 12 至 14 位用于标识拆解不合格半成品的检测情况，第 15 位表示成品是否需要检测，第 16 位表示拆解成品的检测情况。

其中最佳方案为对不合格半成品均进行拆解，在问题 2 中实验结果表明，对不合格品进行拆解可使效益达到较高，也验证了这一结果。

4.4 问题四模型的建立与求解

4.4.1 问题分析

在重新完成问题 2 和问题 3 时，需要考虑抽样检测结果的准确性和误差。首先，计算每个检测结果的置信区间，以评估实际次品率的可能范围。对抽样检测误差进行分析，并调整次品率数据以减少偏差。对于零配件和半成品，优先检测次品率高的项，即使存在误差，也要确保检测频率足够高以维持准确性。增加样本量可以提高检测结果的准确性。最终，根据调整后的次品率和成本效益分析，优化检测方案并重新评估检测成本和风险，以平衡生产效率和产品质量。

4.4.2 方法介绍

在生产过程中，决策方案涉及选择不同的质量控制措施。通过鲁棒性分析模型，可以：

- **模拟次品率的波动：**确定不同质量控制措施对次品率变化的响应。
- **应用敏感性分析：**评估次品率波动、成本变化等因素对整体生产成本和产品质量的影响。
- **优化决策：**在保证成本最小化的同时，确保产品质量在各种不确定性条件下都能达到预期标准。

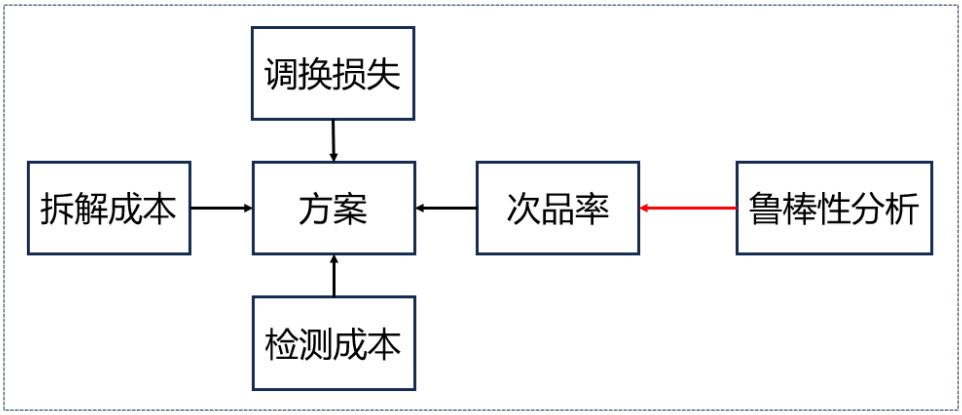

图 8 分析方法

4.4.3 问题求解

以表 3 中的情况 4 为例进行次品率的鲁棒性分析。

情况 4	对零件 1 和零件 2 均进行**检测** 成品进行**检测**-不合格品**拆解**	1264 元

设置次品率的波动范围为 0.05-0.25 之间，间隔为 0.05，检验结果如下表所示：

表 5 次品率波动表

次品率 / 方案	0.05	0.1	0.15	0.2	0.25
检测-拆解	2108	1821	1540	1264	994
检测-不拆解	1894	1416	966	544	150
不检测-拆解	2155	1731	1327	944	581
不检测-不拆解	1920	1326	732	224	-284

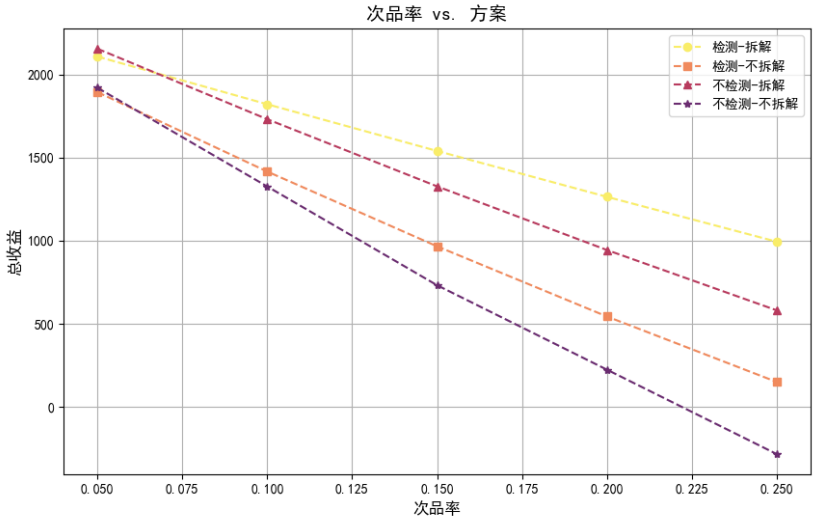

图 9 生产成本与次品率可视化

从图中可以分析出，在检测成本、调换损失和拆解费用保持不变的情况下，总收益随着次品率的增加而减少。同时，数据也显示出，不进行检测和不进行拆解会导致不同的收益变化趋势。

方案三通过模拟不同次品率进行实验，旨在获取在各种次品率条件下的收益率数据。通过这些数据，我们可以深入分析次品率的波动性。具体步骤包括设定一系列不同的次品率，并基于这些次品率进行模拟实验。每次实验都严格按照方案三的步骤，以确保结果的精确性和可信度。通过重复实验，我们积累了各种次品率下的收益率数据，并通过统计分析这些数据，从而更准确地理解和控制次品率对整体收益率的影响。

五、 模型的评价、改进与推广

5.1 模型的优点

问题一中所使用的序贯概率比检验（SPRT）可以通过逐步检验数据来决定是否继续收集更多数据或做出最终决策，从而减少了所需的样本量和测试时间。此外，SPRT 提供了最优的决策规则，通过控制误判的概率（如假阳性和假阴性）来优化决策的准确性。其计算过程相对简单，使得这一方法易于在实际生产和质量控制中实现和应用。

问题二中采用了决策树分析与模拟实验相结合的方法。通过决策树分析，系统地识别了所有可能的决策路径及其潜在结果，并对每个路径进行详细评估。随后，通过模拟实验，生成各种可能的情况，全面评估了每种情况下的效益和风险。

最终，综合决策树分析和模拟实验的结果，选取了效益最大的方案。这一方法不仅确保了决策的全面性，还优化了预期的经济效益和风险控制。

问题三中遗传算法（GA）在处理二进制优化问题时具有显著的优点。首先，GA 展现出强大的全局优化能力，能够在广泛的解空间中进行搜索，从而减少陷入局部最优解的风险。此外，它适用于处理复杂的目标函数和约束条件，并且具有较强的自适应性，通过交叉和变异操作动态调整搜索策略。GA 还支持并行处理，这在计算资源充足时能显著提高计算效率。

5.2 模型的缺点

问题一中序贯概率比检验（SPRT）虽然在减少样本量和测试时间方面具有明显优势，但也存在一些缺点。首先，SPRT 对模型参数的选择和假设较为敏感，需要准确设定误判的允许概率，否则可能会导致不准确的决策。此外，当样本数据的分布特性不完全符合假设时，SPRT 的效果可能会受到影响，从而影响决策的可靠性。最后，尽管 SPRT 的计算过程相对简单，但在复杂的生产环境中实现时，可能需要额外的计算和调整，增加了实际应用的难度。

问题二中采用决策树分析与模拟实验的方法也有其不足之处。首先，决策树分析虽然能系统地评估决策路径，但在面对复杂的决策问题时，可能会生成非常庞大的树结构，这使得分析和计算变得更加复杂和耗时。其次，模拟实验依赖于准确的模型假设和输入数据，如果输入数据不准确或假设不合理，模拟结果可能会失真，影响最终决策的有效性。此外，模拟实验可能需要大量的计算资源和时间，尤其在面对多个变量和复杂情境时，可能导致计算成本的增加。

问题三中 GA 也存在一些缺点。计算开销较大，因为算法涉及种群生成、适应度评估和交叉变异等操作，尤其在问题规模较大时，计算成本显著。GA 的收敛速度通常较慢，可能需要较多的迭代才能找到较优解，特别是在复杂的解空间中。此外，算法的参数调优较为复杂，需通过经验或试验确定合适的参数设置。由于 GA 的随机性，通常需要多次运行以获得稳定的结果，这进一步增加了计算成本。因此，尽管遗传算法在二进制优化问题中提供了有效的全局搜索策略，但计算开销和收敛速度等挑战需要通过合理的调整和实验来克服。

5.3 模型的改进

对于序贯概率比检验（SPRT），可以改进其对模型参数的敏感性，通过使用自适应的策略来动态调整允许的误判概率。此外，引入更多的数据分布模型以提高其在不同数据环境下的适用性，也可以增强 SPRT 的可靠性。在实际应用中，可以结合其他统计方法，以减少计算和调整的复杂性。

对于决策树分析与模拟实验，可以通过采用剪枝技术来简化决策树结构，从而减少计算复杂性。同时，改进模拟实验的模型假设与输入数据的准确性，使用

先进的数据验证技术，可以提升模拟结果的可靠性。引入并行计算技术也可以提高模拟实验的效率。

遗传算法（GA）方面，优化算法的收敛速度可以通过引入更高效的选择和变异策略来实现。例如，使用自适应的变异率和选择机制，能够更好地调整搜索策略。此外，结合局部搜索算法（如爬山算法）与 GA 的全局搜索能力，可以提高求解速度和精度。减小计算开销可以通过减少种群规模和优化算法参数进行。

5.4 模型的推广

SPRT 的推广可以集中在更多的领域，如医疗检测和金融风险评估等，这些领域对样本量和测试时间有严格要求。在这些领域，通过定制 SPRT 的误判控制策略，可以提高决策的准确性和效率。

决策树分析与模拟实验的方法可以推广到复杂项目管理和风险评估中，尤其是在处理多变量和复杂决策环境时。结合实际情况进行模型调整和优化，能够更好地适应不同行业的需求。

遗传算法（GA）的推广可以应用于更多的优化问题和复杂系统设计中，例如供应链管理和自动化设计等领域。通过与其他优化技术结合，GA 可以在更广泛的应用场景中发挥其全局搜索能力，提升系统性能和设计效率。

六、 参考文献

[1] 钟伦珑,刘炅坡,刘永玉,等.基于改进SPRT的机载欺骗式干扰检测方法[J].计算机工程与设计,2023,44(08):2352-2359.

[2] 丁进良,杨翠娥,陈远东,等.复杂工业过程智能优化决策系统的现状与展望[J].自动化学报,2018,44(11):1931-1943.

[3] 杨善林,刘心报,刘林,等.敏捷型生产过程优化和制造执行系统[J].系统工程理论与实践,2011,31(S1):181-186.

[4] Stoumbos Z G, Reynolds Jr M R. The SPRT control chart for the process mean with samples starting at fixed times[J]. Nonlinear Analysis: Real World Applications, 2001, 2(1): 1-34.

[5] Dudas C, Ng A H C, Pehrsson L, et al. Integration of data mining and multi-objective optimisation for decision support in production systems development[J]. International Journal of Computer Integrated Manufacturing, 2014, 27(9): 824-839.

附录

附录 1

介绍：附录文件列表

```
附件:.
│
├─图片
│        img1.png
│        img2.png
│        img3.png
│        img4.png
│        img5.png
│        img6.png
│        img7.png
│        img8.png
│        img9.png
│
├─问题 1
│        SPRT 模型.py
│
├─问题 2
│    ├─均不检测
│    │        均不检测结果.txt
│    │        情况 123456.py
│    │
│    ├─均检测
│    │        均检测结果.txt
│    │        情形 12346.py
│    │        情形 5.py
│    │
│    ├─零件 1 不检测零件 2 检测
│    │        情况 12346.py
│    │        情形 5.py
│    │        零件 1 不检测零件 2 检测.txt
│    │
│    └─零件 1 检测零件 2 不检测
│             情况 12346.py
│             情况 5.py
│             零件 1 检测零件 2 不检测结果.txt
│
└─问题 3
         基于蒙特卡洛模拟与遗传算法的决策优化.py
         波动率可视化.py
```

附录 2

介绍：问题求解代码

问题一 SPRT.py

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
from numba import jit, prange
import numpy as np
import math
# 设置中文显示和负号正常显示
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 参数设置
p0 = 0.1    # 标称值
p1 = 0.2    # 实际业务假设值
# alpha = 0.1    # 置信度为 90%时的边界
# beta = 0.1    # 置信度为 90%时的边界
alpha = 0.05    # 置信度为 95%时的边界
beta = 0.05    # 置信度为 95%时的边界
# 计算决策边界
A = (1 - beta) / alpha
B = beta / (1 - alpha)
lnA = np.log(A)
lnB = np.log(B)
# SPRT 模拟函数
@jit(nopython=True)
def sprt_simulation(p0, p1, lnA, lnB, true_p):
    n = 0
    sum_x = 0
    while True:
        n += 1
        x = np.random.rand() < true_p
        sum_x += x
        lnL = sum_x * np.log(p1 / p0) + (n - sum_x) * np.log((1 - p1) / (1 - p0))
        if lnL >= lnA:
            return 0, n    # Accept H1 (defective)
        elif lnL <= lnB:
            return 1, n    # Accept H0 (non-defective)
# 模拟不同真实次品率下的 SPRT 性能
true_p_range = np.arange(0.05, 0.26, 0.01)
num_simulations = 100000
oc_curve = np.zeros(len(true_p_range))
```

```
asn_curve = np.zeros(len(true_p_range))
for i in prange(len(true_p_range)):
    true_p = true_p_range[i]
    decisions = np.zeros(num_simulations, dtype=np.int32)
    sample_sizes = np.zeros(num_simulations, dtype=np.int32)
    for j in range(num_simulations):
        decision, n = sprt_simulation(p0, p1, lnA, lnB, true_p)
        decisions[j] = decision
        sample_sizes[j] = n
    oc_curve[i] = np.mean(decisions)   # 计算 OC 曲线
    asn_curve[i] = np.mean(sample_sizes)   # 计算 ASN 曲线
# 绘制 OC 曲线和 ASN 曲线在同一窗口
plt.figure(figsize=(12, 6))
# 绘制 OC 曲线
plt.subplot(1, 2, 1)
plt.plot(true_p_range, oc_curve, 'b-', linewidth=2, c='#3d84a8')
plt.axvline(p0, color='r', linestyle='--', linewidth=1.5)
plt.axvline(p1, color='r', linestyle='--', linewidth=1.5)
plt.title('操作特性（OC）曲线', fontsize=14)
plt.xlabel('真实次品率', fontsize=12)
plt.ylabel('接受概率', fontsize=12)
plt.legend(['OC 曲线', 'p0', 'p1'], loc='best', fontsize=10)
# 找到 OC 曲线与左边界的交点
oc_curve_at_p0 = np.interp(p0, true_p_range, oc_curve)
# 标记交点
plt.annotate(
    f'{oc_curve_at_p0:.2f}',
    xy=(p0, oc_curve_at_p0),
    xytext=(p0 - 0.02, oc_curve_at_p0 + -0.1),
    arrowprops=dict(facecolor='white', shrink=0.05, width=3, headwidth=8,
headlength=10),
    fontsize=12,
    color='black',
)
plt.scatter(p0, oc_curve_at_p0, color='black')   # 显示交点
plt.grid(True)
# 绘制 ASN 曲线
plt.subplot(1, 2, 2)
plt.plot(true_p_range, asn_curve, 'g-', linewidth=2)
plt.axvline(p0, color='r', linestyle='--', linewidth=1.5)
plt.axvline(p1, color='r', linestyle='--', linewidth=1.5)
plt.title('平均样本量（ASN）曲线', fontsize=14)
plt.xlabel('真实次品率', fontsize=12)
```

```
plt.ylabel('平均样本量', fontsize=12)
plt.legend(['ASN 曲线', 'p0', 'p1'], loc='best', fontsize=10)
# 找到 ASN 曲线在 p0 和 p1 处的值
asn_curve_at_p0 = np.interp(p0, true_p_range, asn_curve)
# 标记 p0 处的交点
plt.annotate(
    f'{math.ceil(asn_curve_at_p0)}',
    xy=(p0, asn_curve_at_p0),
    xytext=(p0 - 0.02, asn_curve_at_p0 + 3),
    arrowprops=dict(facecolor='white', shrink=0.05, width=3, headwidth=8,
headlength=10),
    fontsize=12,
    color='black'
)
plt.grid(True)
plt.tight_layout()
plt.show()
# 计算在 p0 和 p1 处的具体性能指标
idx_p0 = np.argmin(np.abs(true_p_range - p0))
idx_p1 = np.argmin(np.abs(true_p_range - p1))
p0_performance = [oc_curve[idx_p0], asn_curve[idx_p0]]
p1_performance = [oc_curve[idx_p1], asn_curve[idx_p1]]
# 输出结果
print('SPRT 模型性能指标：')
print(f'在 p0 ({p0}) 处的接受概率: {p0_performance[0]:.4f}')
print(f'在 p0 ({p0}) 处的平均样本量: {math.ceil(p0_performance[1])}')
print(f'在 p1 ({p1}) 处的接受概率: {p1_performance[0]:.4f}')
print(f'在 p1 ({p1}) 处的平均样本量: {math.ceil(p1_performance[1])}')
# 模拟实际检测过程
num_batches = 1000
batch_decisions = np.zeros(num_batches, dtype=np.float64)
batch_sample_sizes = np.zeros(num_batches, dtype=np.float64)
for i in prange(num_batches):
    true_p = 0.10 + 0.05 * np.random.randn()
    true_p = np.clip(true_p, 0, 1)  # 保证 true_p 在 [0, 1] 之间
    decisions = np.zeros(num_simulations, dtype=np.int32)
    sample_sizes = np.zeros(num_simulations, dtype=np.int32)
    for j in range(num_simulations):
        decision, n = sprt_simulation(p0, p1, lnA, lnB, true_p)
        decisions[j] = decision
        sample_sizes[j] = n
    batch_decisions[i] = np.mean(decisions)
    batch_sample_sizes[i] = np.mean(sample_sizes)
```

```
# 统计结果
reject_rate = round(np.mean(batch_decisions), 2)
avg_sample_size = math.ceil(np.mean(batch_sample_sizes))
# 输出实际检测结果
print()
print('实际检测结果：')
print(f'拒收率: {reject_rate}')
print(f'平均样本量: {avg_sample_size}')
# 绘制样本量分布直方图
# plt.figure()
# plt.hist(batch_sample_sizes, bins=30, density=True, alpha=0.7)
# plt.title('样本量分布', fontsize=14)
# plt.xlabel('样本量', fontsize=12)
# plt.ylabel('频率', fontsize=12)
# plt.grid(False)
# plt.show()
```

问题二 情况 123456.py

```
# -*- coding: utf-8 -*-
import math
parts1 = parts2 = 100   # 样品数量
p1 = 0.05   # 次品率
p2 = 0.05
p3 = 0.05
p4 = 1 - (1 - p1) * (1 - p2)
b1 = 4   # 购买单价
b2 = 18
c1 = 2   # 检测成本
c2 = 3
c3 = 3
w = 6   # 装配成本
m = 10   # 调换损失
d = 40   # 拆解成本
s = 56   # 市场售价
# 情况 1 初始均进行检测
def case(state):
    purchase_cost = parts1 * b1 + parts2 * b2
    if state[0]:
        print("对成品进行检测")
        w_cost = parts1 * w
        qualified_product1 = math.ceil(parts1 * (1 - p4))
```

```
            c_cost = qualified_product1 * c3
            qualified_product2 = math.ceil(qualified_product1 * (1 - p3))
            unqualified_product = math.ceil(qualified_product1 - qualified_product2)
            if state[1]:
                print('不合格品拆解')
                d_cost = unqualified_product * d
                w_cost_2 = unqualified_product * w
                print(
                    f"总收益: {round(qualified_product2 * s + unqualified_product
* s - d_cost - w_cost_2 - w_cost - c_cost - purchase_cost)}")
                print()
            else:
                print('不合格品不拆解')
                print(f" 总 收 益 : {round(qualified_product2 * s - w_cost -
purchase_cost - c_cost)}")
                print()
        else:
            print('对成品不检测')
            w_cost = parts1 * w
            qualified_product = math.ceil(parts1 * (1 - p4))
            unqualified_product = math.ceil(qualified_product - qualified_product *
(1 - p3))
            # print('产品总数:', qualified_product)
            # print('不合格品:', unqualified_product)
            # print('购买与检测初始成本:', purchase_cost)
            # print('装配成本:', w_cost)
            if state[1]:
                print('不合格品拆解')
                # print('调换成本:', unqualified_product * (s + m))
                # print('拆解与装配:', unqualified_product * (w + d))
                # print('二次产品:', unqualified_product * s)
                # print('销售额:', qualified_product * s)
                print(
                    f" 总 收 益 : {round(qualified_product * s - purchase_cost -
w_cost - unqualified_product * (s + m) - unqualified_product * (w + d) +
unqualified_product * s)}"
                )
                print()
            else:
                print('不合格品不拆解')
                # print('调换成本:', unqualified_product * (s + m))
                # print('销售额:', c_parts1 * s)
                print(
```

```
                    f"总收益: {round(qualified_product * s - unqualified_product *
(s + m) - w_cost - purchase_cost)}")
                print()
case([1, 1])
case([1, 0])
case([0, 1])
case([0, 0])
```

问题二 情形 5.py

```
import math
parts1 = parts2 = 100   # 样品数量
p1 = 0.1   # 次品率
p2 = 0.2
p3 = 0.1
b1 = 4   # 购买单价
b2 = 18
c1 = 8   # 检测成本
c2 = 1
c3 = 2
w = 6   # 装配成本
m = 10   # 调换损失
d = 5   # 拆解成本
s = 56   # 市场售价
# 情况 1 初始均进行检测
def case1(state):
    purchase_cost = parts1 * (c1 + b1) + parts2 * (c2 + b2)
    c_parts2 = parts2 * (1 - p2)
    if state[0]:
        print("对成品进行检测")
        w_cost = c_parts2 * w
        c_cost = c_parts2 * c3
        qualified_product = c_parts2 * (1 - p3)
        if state[1]:
            print('不合格品拆解')
            d_cost = (c_parts2 - qualified_product) * d
            w_cost_2 = (c_parts2 - qualified_product) * w
            qualified_product_2 = c_parts2 - qualified_product
            print(
                f"总收益: {round((qualified_product + qualified_product_2) * s
- w_cost - purchase_cost - c_cost - d_cost - w_cost_2)}")
            print()
```

```
            else:
                print('不合格品不拆解')
                print(f" 总 收 益 : {round(qualified_product * s - w_cost -
purchase_cost - c_cost)}")
                print()
        else:
            print('对成品不检测')
            qualified_product = c_parts2 * (1 - p3)
            w_cost = c_parts2 * w
            if state[1]:
                print('不合格品拆解')
                print(
                    f" 总 收 益 : {round(c_parts2 * s - (math.ceil(c_parts2 -
qualified_product)) * (s + m) - w_cost - purchase_cost - math.ceil(c_parts2 -
qualified_product) * (w + d) + math.ceil(qualified_product * p3) * s)}")
                print()
            else:
                print('不合格品不拆解')
                print(
                    f" 总 收 益 : {round(c_parts2 * s - (math.ceil(c_parts2 -
qualified_product)) * (s + m) - w_cost - purchase_cost)}")
                print()
case1([1, 0])
case1([1, 1])
case1([0, 1])
case1([0, 0])
```

问题二 情形 12346.py

```
# -*- coding: utf-8 -*-

import math
parts1 = parts2 = 100   # 样品数量
p1 = 0.05   # 次品率
p2 = 0.05
p3 = 0.05
b1 = 4   # 购买单价
b2 = 18
c1 = 2   # 检测成本
c2 = 3
c3 = 3
w = 6   # 装配成本
```

```
m = 10   # 调换损失
d = 40   # 拆解成本
s = 56   # 市场售价
# 情况 1 初始均进行检测
def case(state):
    purchase_cost = parts1 * (c1 + b1) + parts2 * (c2 + b2)
    # 假设 c_parts1 = c_parts2
    c_parts1 = parts1 * (1 - p1)
    c_parts2 = parts2 * (1 - p2)
    if state[0]:
        print("对成品进行检测")

        w_cost = c_parts1 * w
        c_cost = c_parts1 * c3
        qualified_product = c_parts2 * (1 - p3)
        if state[1]:
            print('不合格品拆解')
            d_cost = (c_parts1 - qualified_product) * d
            w_cost_2 = (c_parts1 - qualified_product) * w
            qualified_product_2 = c_parts1 - qualified_product
            print(
                f"总收益: {round((qualified_product + qualified_product_2) * s
- w_cost - purchase_cost - c_cost - d_cost - w_cost_2)}")
            print()
        else:
            print('不合格品不拆解')
            print(f"总收益: {round(qualified_product * s - w_cost -
purchase_cost - c_cost)}")
            print()
    else:
        print('对成品不检测')
        qualified_product = c_parts2 * (1 - p3)
        unqualified_product = c_parts1 - qualified_product
        # print('产品总数:', c_parts1)
        # print('合格品:', qualified_product)
        # print('不合格品:', unqualified_product)
        w_cost = c_parts1 * w
        # print('购买与检测初始成本:', purchase_cost)
        # print('装配成本:', w_cost)
        if state[1]:
            # print('不合格品拆解')
            # print('调换成本:', unqualified_product * (s + m))
            # print('拆解与装配:', unqualified_product * (w + d))
```

```
                # print('二次产品:', unqualified_product * s)
                # print('销售额:', c_parts1 * s)
                print(
                    f" 总 收 益 : {round(c_parts1 * s - purchase_cost - w_cost - unqualified_product * (s + m) - unqualified_product * (w + d) + unqualified_product * s)}"
                )
                print()
            else:
                print('不合格品不拆解')
                # print('调换成本:', unqualified_product * (s + m))
                # print('销售额:', c_parts1 * s)
                print(f" 总 收 益 : {round(c_parts1 * s - (math.ceil(c_parts1 - qualified_product)) * (s + m) - w_cost - purchase_cost)}")
                print()
case([1, 1])
case([1, 0])
case([0, 1])
case([0, 0])
```

问题二 情形 12346.py

```
# -*- coding: utf-8 -*-
import math
parts1 = parts2 = 100   # 样品数量
p1 = 0.05   # 次品率
p2 = 0.05
p3 = 0.05
b1 = 4   # 购买单价
b2 = 18
c1 = 2   # 检测成本
c2 = 3
c3 = 3
w = 6   # 装配成本
m = 10   # 调换损失
d = 40   # 拆解成本
s = 56   # 市场售价
def case(state):
    purchase_cost = parts1 * b1 + parts2 * (c1 + b2)
    c_parts2 = parts2 * (1 - p1) - 5
    # 零件 2 多了 10 个
    if state[0]:
```

```
            print("对成品进行检测")
            w_cost = c_parts2 * w
            c_cost = c_parts2 * c3
            qualified_product = math.ceil(c_parts2 * (1 - p3))
            if state[1]:
                print('不合格品拆解')
                d_cost = (c_parts2 - qualified_product) * d
                w_cost_2 = (c_parts2 - qualified_product) * w
                unqualified_product = c_parts2 - qualified_product
                print(
                    f"总收益: {round((qualified_product + unqualified_product) * s
- w_cost - purchase_cost - c_cost - d_cost - w_cost_2 - unqualified_product * c2)}")
                print()
            else:
                print('不合格品不拆解')
                print(f" 总 收 益 : {round(qualified_product * s - w_cost -
purchase_cost - c_cost)}")
                print()
        else:
            print('对成品不检测')
            qualified_product = c_parts2 * (1 - p3)
            unqualified_product = c_parts2 - qualified_product
            # print('产品总数:', c_parts2)
            # print('合格品:', qualified_product)
            # print('不合格品:', unqualified_product)
            w_cost = c_parts2 * w
            # print('购买与检测初始成本:', purchase_cost)
            # print('装配成本:', w_cost)
            if state[1]:
                print('不合格品拆解')
                print(
                    f" 总 收 益 : {round(c_parts2 * s - (math.ceil(c_parts2 -
qualified_product)) * (s + m) - w_cost - purchase_cost - unqualified_product * c2 -
unqualified_product * (w + d)) + unqualified_product * s}")
                print()
            else:
                print('不合格品不拆解')
                # print('调换成本:', unqualified_product * (s + m))
                # print('销售额:', c_parts2 * s)
                print(f" 总 收 益 : {round(c_parts2 * s - (math.ceil(c_parts2 -
qualified_product)) * (s + m) - w_cost - purchase_cost)}")
                print()
case([1, 1])
```

```
case([1, 0])
case([0, 1])
case([0, 0])
```

问题二 情形 5.py

```
# -*- coding: utf-8 -*-
import math
parts1 = parts2 = 100   # 样品数量
p1 = 0.1   # 次品率
p2 = 0.1
p3 = 0.1
b1 = 4   # 购买单价
b2 = 18
c1 = 2   # 检测成本
c2 = 3
c3 = 3
w = 6   # 装配成本
m = 6   # 调换损失
d = 5   # 拆解成本
s = 56   # 市场售价
def case(state):
    purchase_cost = parts1 * b1 + parts2 * (c1 + b2)
    c_parts2 = parts2 * (1 - p1) - 10
    # 零件 2 多了 10 个
    if state[0]:
        print("对成品进行检测")
        w_cost = c_parts2 * w
        c_cost = c_parts2 * c3
        qualified_product = math.ceil(c_parts2 * (1 - p3))
        if state[1]:
            print('不合格品拆解')
            d_cost = (c_parts2 - qualified_product) * d
            w_cost_2 = (c_parts2 - qualified_product) * w
            unqualified_product = c_parts2 - qualified_product
            print(
                f"总收益: {round((qualified_product + unqualified_product) * s
- w_cost - purchase_cost - c_cost - d_cost - w_cost_2 - unqualified_product * c2)}")
            print()
        else:
            print('不合格品不拆解')
            print(f"总收益: {round(qualified_product * s - w_cost -
```

```
purchase_cost - c_cost)}")
                print()
        else:
            print('对成品不检测')
            qualified_product = c_parts2 * (1 - p3)
            unqualified_product = c_parts2 - qualified_product
            # print('产品总数:', c_parts2)
            # print('合格品:', qualified_product)
            # print('不合格品:', unqualified_product)
            w_cost = c_parts2 * w
            # print('购买与检测初始成本:', purchase_cost)
            # print('装配成本:', w_cost)
            if state[1]:
                print('不合格品拆解')
                print(
                    f" 总 收 益 : {round(c_parts2 * s - (math.ceil(c_parts2 -
qualified_product)) * (s + m) - w_cost - purchase_cost - unqualified_product * c2 -
unqualified_product * (w + d)) + unqualified_product * s}")
                print()
            else:
                print('不合格品不拆解')
                # print('调换成本:', unqualified_product * (s + m))
                # print('销售额:', c_parts2 * s)
                print(f" 总 收 益 : {round(c_parts2 * s - (math.ceil(c_parts2 -
qualified_product)) * (s + m) - w_cost - purchase_cost)}")
                print()
case([1, 1])
case([1, 0])
case([0, 1])
case([0, 0])
```

问题二 情况 12346.py

```
# -*- coding: utf-8 -*-
import math
parts1 = parts2 = 100   # 样品数量
p1 = 0.05   # 次品率
p2 = 0.05
p3 = 0.05
b1 = 4   # 购买单价
b2 = 18
c1 = 2   # 检测成本
```

```
c2 = 3
c3 = 3
w = 6   # 装配成本
m = 10   # 调换损失
d = 40   # 拆解成本
s = 56   # 市场售价
def case(state):
    purchase_cost = parts1 * (c1 + b1) + parts2 * b2
    c_parts1 = parts1 * (1 - p1) - 5
    # 零件 2 多了 10 个
    if state[0]:
        print("对成品进行检测")
        w_cost = c_parts1 * w
        c_cost = c_parts1 * c3
        qualified_product = math.ceil(c_parts1 * (1 - p3))
        if state[1]:
            print('不合格品拆解')
            d_cost = (c_parts1 - qualified_product) * d
            w_cost_2 = (c_parts1 - qualified_product) * w
            unqualified_product = c_parts1 - qualified_product
            print(
                f"总收益: {round((qualified_product + unqualified_product) * s
- w_cost - purchase_cost - c_cost - d_cost - w_cost_2 - unqualified_product * c2)}")
            print()
        else:
            print('不合格品不拆解')
            print(f"总收益: {round(qualified_product * s - w_cost -
purchase_cost - c_cost)}")
            print()
    else:
        print('对成品不检测')
        qualified_product = c_parts1 * (1 - p3)
        unqualified_product = c_parts1 - qualified_product
        # print('产品总数:', c_parts1)
        # print('合格品:', qualified_product)
        # print('不合格品:', unqualified_product)
        w_cost = c_parts1 * w
        # print('购买与检测初始成本:', purchase_cost)
        # print('装配成本:', w_cost)
        if state[1]:
            print('不合格品拆解')
            print(
                f"总收益: {round(c_parts1 * s - (math.ceil(c_parts1 -
```

```
qualified_product)) * (s + m) - w_cost - purchase_cost - unqualified_product * c2 - unqualified_product * (w + d)) + unqualified_product * s}")
                print()
            else:
                print('不合格品不拆解')
                # print('调换成本:', unqualified_product * (s + m))
                # print('销售额:', c_parts1 * s)
                print(f" 总 收 益 : {round(c_parts1 * s - (math.ceil(c_parts1 - qualified_product)) * (s + m) - w_cost - purchase_cost)}")
                print()
case([1, 1])
case([1, 0])
case([0, 1])
case([0, 0])
```

问题二 情况 5.py

```
# -*- coding: utf-8 -*-
import math
parts1 = parts2 = 100   # 样品数量
p1 = 0.05   # 次品率
p2 = 0.05
p3 = 0.05
b1 = 4   # 购买单价
b2 = 18
c1 = 2   # 检测成本
c2 = 3
c3 = 3
w = 6   # 装配成本
m = 10   # 调换损失
d = 40   # 拆解成本
s = 56   # 市场售价
def case(state):
    purchase_cost = parts1 * (c1 + b1) + parts2 * b2
    c_parts1 = parts1 * (1 - p1) - 5
    # 零件 2 多了 10 个
    if state[0]:
        print("对成品进行检测")
        w_cost = c_parts1 * w
        c_cost = c_parts1 * c3
        qualified_product = math.ceil(c_parts1 * (1 - p3))
        if state[1]:
```

```
                print('不合格品拆解')
                d_cost = (c_parts1 - qualified_product) * d
                w_cost_2 = (c_parts1 - qualified_product) * w
                unqualified_product = c_parts1 - qualified_product
                print(
                    f"总收益: {round((qualified_product + unqualified_product) * s
- w_cost - purchase_cost - c_cost - d_cost - w_cost_2 - unqualified_product * c2)}")
                print()
            else:
                print('不合格品不拆解')
                print(f"总收益: {round(qualified_product * s - w_cost -
purchase_cost - c_cost)}")
                print()
        else:
            print('对成品不检测')
            qualified_product = c_parts1 * (1 - p3)
            unqualified_product = c_parts1 - qualified_product
            # print('产品总数:', c_parts1)
            # print('合格品:', qualified_product)
            # print('不合格品:', unqualified_product)
            w_cost = c_parts1 * w
            # print('购买与检测初始成本:', purchase_cost)
            # print('装配成本:', w_cost)
            if state[1]:
                print('不合格品拆解')
                print(
                    f"总收益: {round(c_parts1 * s - (math.ceil(c_parts1 -
qualified_product)) * (s + m) - w_cost - purchase_cost - unqualified_product * c2 -
unqualified_product * (w + d)) + unqualified_product * s}")
                print()
            else:
                print('不合格品不拆解')
                # print('调换成本:', unqualified_product * (s + m))
                # print('销售额:', c_parts1 * s)
                print(f"总收益: {round(c_parts1 * s - (math.ceil(c_parts1 -
qualified_product)) * (s + m) - w_cost - purchase_cost)}")
                print()
case([1, 1])
case([1, 0])
case([0, 1])
case([0, 0])

---------------------------------------------------------------------------------------------------------
```

问题三 基于蒙特卡洛模拟与遗传算法的决策优化.py

```
# 导入必要的库
import functools
import random
import time
def generate_random_number(probability_of_one):
    if random.random() <= probability_of_one:
        return 1
    else:
        return 0
#获取指定类型的数据
def getData(type,name,data):
    if type=="part":
        cost_list=[2,8,12,2,8,12,8,12]
        detect_cost_list=[1,1,2,1,1,2,1,2]
        if data=="defectRate":
            return 0.1
        if data=="cost":
            return cost_list[name-1]
        if data=="detect_cost":
            return detect_cost_list[name-1]
    if type=="sProduct":
        cost_list=[8,8,8]
        detect_cost_list=[4,4,4]
        rework_cost_list=[6,6,6]
        if data=="defectRate":
            return 0.1
        if data=="assemble_cost":
            return cost_list[name-1]
        if data=="detect_cost":
            return detect_cost_list[name-1]
        if data=="rework_cost":
            return rework_cost_list[name-1]
        if data=="compose":
            if name==1:
                return [1,2,3]
            if name==2:
                return [4,5,6]
            if name==3:
                return [7,8]
    if type=="product":
        cost_list=[8]
```

```
            detect_cost_list=[6]
            rework_cost_list=[10]
            price_list=[200]
            swap_cost_list=[40]
            if data=="defectRate":
                  return 0.1
            if data=="assemble_cost":
                  return cost_list[name-1]
            if data=="detect_cost":
                  return detect_cost_list[name-1]
            if data=="rework_cost":
                  return rework_cost_list[name-1]
            if data=="swap_cost":
                  return swap_cost_list[name-1]
            if data=="price":
                  return price_list[name-1]
            if data=="compose":
                  if name==1:
                        return [1,2,3]
#定义零件对象
class Part:
      def __init__(self, name):
            self.type="part"
            cost=getData("part",name,"cost")
            detect_cost = getData("part", name, "detect_cost")
            defect_rate=getData("part", name, "defectRate")
            self.id=time.time()
            self.name = name
            self.cost = cost
            self.detect_cost = detect_cost
            #定义状态 0 合格 1 不合格
            self.state = generate_random_number(defect_rate)
            #定义检测状态 0 未检测 1 检测
            self.detect_state = 0
      def setDetectState(self,detect_state):
            self.detect_state=detect_state
#定义半成品
class sProduct:
      def __init__(self, name,object_list,state=None):
            self.type="sProduct"
            self.state=state
            price=getData("sProduct",name,"price")
            assemble_cost=getData("sProduct",name,"assemble_cost")
```

```
            detect_cost=getData("sProduct",name,"detect_cost")
            defect_rate=getData("sProduct",name,"defectRate")
            swap_loss=getData("sProduct",name,"swap_loss")
            rework_cost=getData("sProduct",name,"rework_cost")
            self.id=time.time()
            self.name = name
            self.price = price
            self.assemble_cost=assemble_cost
            self.detect_cost = detect_cost
            # 定义状态 0 合格 1 不合格
            if self.state is None:
                self.state = generate_random_number(defect_rate)
            # 定义检测状态 0 未检测 1 检测
            self.detect_state = 0
            self.swap_loss=swap_loss
            self.rework_cost= rework_cost
            self.object_list=object_list
    def setDetectState(self,detect_state):
            self.detect_state=detect_state
#定义产品
class Product:
    def __init__(self, name,object_list,state=None):
            self.type="product"
            self.state=state
            price = getData("product", name, "price")
            assemble_cost = getData("product", name, "assemble_cost")
            detect_cost = getData("product", name, "detect_cost")
            defect_rate = getData("product", name, "defectRate")
            swap_loss = getData("product", name, "swap_loss")
            rework_cost = getData("product", name, "rework_cost")
            self.id=time.time()
            self.name = name
            self.price = price
            self.assemble_cost=assemble_cost
            self.detect_cost = detect_cost
            # 定义状态 0 合格 1 不合格
            if self.state is None:
                self.state = generate_random_number(defect_rate)
            # 定义检测状态 0 未检测 1 检测
            self.detect_state = 0
            self.swap_loss=swap_loss
            self.rework_cost= rework_cost
            self.object_list=object_list
```

```
        def setDetectState(self,detect_state):
            self.detect_state=detect_state
        #获取对象列表的所有 name
def getObjectNameList(object_list):
        res=[]
        for object in object_list:
            res.append(object.name)
        res.sort()
        return res
#定义一个仓库
class Factory:
        def __init__(self, part_num_list=None):
            #总收益
            self.total_profit = 0
            #总购买成本
            self.total_cost=0
            self.cost_dict={
                "part":{
                    "detect_cost":0
                },
                "sProduct":{
                    "detect_cost": 0,
                    "assemble_cost":0,
                    "rework_cost":0,
                },
                "product":{
                    "detect_cost":0,
                    "assemble_cost":0,
                    "rework_cost":0,
                    "swap_cost":0,
                }
            }
            #self
            self.object_dict = {
                "part":{
                },
                "sProduct":{
                    "1": [],
                    "2": [],
                    "3": []
                },
                "product":{
                    "1": []
```

```
                }
            }
            if part_num_list is None:
                part_num_list = []
            else:
                self.purchasePart(part_num_list)
        #买入零件
        def purchasePart(self,part_num_list):
            for index,num in enumerate(part_num_list):
                if str(index+1) in self.object_dict["part"]:
                    self.object_dict["part"][str(index+1)].extend([Part(index+1)  for
_ in range(num)])
                else:
                    self.object_dict["part"][str(index+1)] = [Part(index+1) for _ in
range(num) ]
                #获取对应的购买成本
                self.total_cost+=getData("part",index+1,"cost")*num
        #根据 ID 丢弃指定元器件，应为买入的时候已经计入成本，不需要再次计
算成本
        def discardObject(self,type,name,object):
            self.object_dict[type][str(name)].remove(object)
        #检查指定元器件,随机抽取指定数量指定类型元件
        def detectObjectList(self,object_list):
            #检测指定元器件列表
            res=[]
            o_list=[]
            for object in object_list:
                if object.detect_state!=1:

index=self.object_dict[object.type][str(object.name)].index(object)
                    detect_cost=getData(object.type,object.name, "detect_cost")
                    self.cost_dict[object.type]["detect_cost"]+=detect_cost
                    object.detect_type=1
                    self.object_dict[object.type][str(object.name)][index]=object
                o_list.append(object)
                res.append(object.state)
            return res,o_list
        #随机抽取元器件
        def choiceObject(self,type,name,num=1):
            res=[]
            if isinstance(name, list):
                for n in name:
                    if len(self.object_dict[type][str(n)]) < 1:
```

```
                    return None
                res.append(random.choice(self.object_dict[type][str(n)]))
            return res
        if num==1:
            if len(self.object_dict[type][str(name)]) < 1:
                return None
            return    random.choice(self.object_dict[type][str(name)])
        else:
            if len(self.object_dict[type][str(name)]) < num:
                return None
            return random.sample(self.object_dict[type][str(name)], num)
    #换掉损失
    def swapProduct(self,name,object):
        swap_cost=getData("product",name, "swap_cost")
        if object.state==0:
            raise Exception("这个产品是合格产品，不允许换掉")
        self.cost_dict["product"]["swap_cost"]+=swap_cost
        index = self.object_dict[object.type][str(object.name)].index(object)
        object.detect_type = 1
        self.object_dict[object.type][str(object.name)][index] = object
    #拆解产品,返回拆解的对象
    def reworkObject(self,type,name,object):
        if type not in ["sProduct","product"]:
            raise    Exception("不支持拆解非半成品、半成品")
        rework_cost=getData(type,name, "rework_cost")
        if object.state==0:
            raise Exception("这个产品是合格产品，不允许拆解")
        self.cost_dict[type]["rework_cost"]+=rework_cost
        #返回拆解对象
        object_list=object.object_list
        # 删除指定对象
        self.object_dict[type][str(name)].remove(object)
        return object_list
    #自动写入对象到指定位置
    def writeAutoObject(self,object_list):
        for object in object_list:
            if isinstance(object, Part):
                self.object_dict["part"][str(object.name)].append(object)
            if isinstance(object,sProduct):
                self.object_dict["sProduct"][str(object.name)].append(object)
            if isinstance(object,Product):
                self.object_dict["product"][str(object.name)].append(object)
    # 自动写入对象到指定位置，对象必须已经存在
```

```
    def deleteAutoObject(self, object_list):
        for object in object_list:
            if isinstance(object, Part):
                self.object_dict["part"][str(object.name)].remove(object)
            if isinstance(object, sProduct):
                self.object_dict["sProduct"][str(object.name)].remove(object)
            if isinstance(object, Product):
                self.object_dict["product"][str(object.name)].remove(object)
    #合成组件，返回合成对象，否则返回 None
    def composeObject(self,type,name,object_list):
        if type not in ["sProduct","product"]:
            raise Exception("合成对象只能是半成品和成品")
        assemble_cost=getData(type,name, "assemble_cost")
        self.cost_dict[type]["assemble_cost"]+=assemble_cost
        if set(getObjectNameList(object_list)) != set(getData(type, name,
"compose")):
            raise Exception(f"组件组成不
对,{getObjectNameList(object_list)},{getData(type, name, 'compose')}")
        #只要子组件有一个不合格成品一定不合格
        state=0
        for object in object_list:
            if object.state:
                state=1
        if type == "sProduct":
            object=sProduct(name=name,object_list=object_list,state=state)
        if type == "product":
            object=Product(name=name,object_list=object_list,state=state)
        self.deleteAutoObject(object_list)
        self.writeAutoObject([object])
        return object
    #获取对象数量
    def countObject(self):
        res={
            "part":[],
            "sProduct":[],
            "product":[],
        }
        for t in ["part","sProduct","product"]:
            if t=="part":
                for n in range(8):
                    res[t].append(len(self.object_dict[t][str(n+1)]))
            if t=="sProduct":
                for n in range(3):
```

```
                        res[t].append(len(self.object_dict[t][str(n+1)]))
            if t=="product":
                for n in range(1):
                    res[t].append(len(self.object_dict[t][str(n+1)]))
        return res
    def countCost(self):
        res=0
        for cost_name,cost in self.cost_dict.items():
            for k,v in cost.items():
                res+=v
        return res + self.total_cost
    #计算利润
    def countProfit(self):
        cost=self.countCost()
        return len(self.object_dict["product"]["1"])*200-cost
    #判断零件是否用完了
    def hasIsObjectZero(self,type_list):
        for t in type_list:
            if t=="part":
                for i in range(8):
                    if len(self.object_dict[t][str(i+1)])==0:
                        return True
            if t=="sProduct":
                for i in range(3):
                    if len(self.object_dict[t][str(i+1)])==0:
                        return True
        return False
#检测零件一 检测零件二 检测成品 拆解成品
def Test(F,X):
    X_part=X[0]
    X_sProduct=X[1]
    X_product=X[2]
    #零件检测已经处理
    object_list=[]
    for index in range(8):
        part=F.choiceObject("part",index+1)
        if X_part[index]:
            object_state_list=F.detectObjectList(object_list=[part])
            if object_state_list[0]:
                F.discardObject("part",index+1,part)
            # print(object_state,F.countObject())
        object_list.append(part)
    part_list_1=F.choiceObject("part", [1,2,3])
```

```
part_list_2 = F.choiceObject("part", [4, 5, 6])
part_list_3 = F.choiceObject("part", [7,8])
if not (part_list_1 and part_list_2 and part_list_3):
    return
sProduct_1=F.composeObject("sProduct",1,part_list_1)
sProduct_2=F.composeObject("sProduct", 2, part_list_2)
sProduct_3=F.composeObject("sProduct", 3, part_list_3)
#处理半成品
for index in range(3):
    sProduct = F.choiceObject("sProduct", index + 1)
    if X_sProduct[index]:
        sProduct_state_list,o_list=F.detectObjectList(object_list=[sProduct])
        sProduct=o_list[0]
    #判断是否已经被检测
    if sProduct.detect_state:
        #拆解半成品
        if X_sProduct[index+3]:
            #返回拆解对象
            part_list=F.reworkObject("sProduct",index+1,sProduct)
            #重新放回工厂
            F.writeAutoObject(part_list)
        #直接丢掉半成品
        else:
            F.discardObject("sProduct",index+1,sProduct)
#组成成品
sProduct_list=F.choiceObject("sProduct", [1, 2, 3])
if sProduct_list is None:
    return
product_1=F.composeObject("product",1,sProduct_list)
#处理成品
product=F.choiceObject("product",1)
if X_product[0]:
    product_state_list,o_list=F.detectObjectList(object_list=[product])
    product=o_list[0]
if product.detect_state:
    #拆解半成品
    if X_product[1]:
        sProduct_list=F.reworkObject("product",1,product)
        F.writeAutoObject(sProduct_list)
    else:
        F.discardObject("product",1,product)
#处理市场里面的产品
product_1 = F.choiceObject("product", 1)
```

```
        #用户获得不合格产品
        if product_1 and product_1.state:
            F.swapProduct(1,product_1)
from scipy.optimize import minimize
#目标函数
def op_func(X):
        if X[11] == 1 and X[8] == 0:
            return -9999,
        if X[12] == 1 and X[9] == 0:
            return -9999,
        if X[13] == 1 and X[10] == 0:
            return -9999,
        X = [
            X[:8],  # 第一行包含前 8 个元素
            X[8:14],  # 第二行包含接下来的 6 个元素
            X[14:]  # 第三行包含剩下的 2 个元素
        ]
        F = Factory([100, 100, 100, 100, 100, 100, 100, 100])
        while not F.hasIsObjectZero(["part"]):
            Test(F,X)
        # print(F.countObject())
        return F.countProfit(),

## 定义生成一个随机二进制串的函数，其中第三位不能是 1
def generate_individual():
        ind = [toolbox.attr_bool() for _ in range(16)]
        return creator.Individual(ind)

from deap import base, creator, tools, algorithms
# 设置遗传算法的参数
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", generate_individual)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", op_func)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# 创建种群并运行遗传算法
population = toolbox.population(n=500)
```

```
NGEN = 100
for gen in range(NGEN):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.1)
    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
        # print(fit)
    population = toolbox.select(offspring, k=len(population))
    print(population)
best_ind = tools.selBest(population, k=1)[0]
print("Best individual:", best_ind)
print("Fitness value:", best_ind.fitness.values[0])
#
# print(op_func([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]))
```

问题三 波动率可视化.py

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
plt.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
# 数据
x = [0.05, 0.1, 0.15, 0.2, 0.25]
y1 = [2108, 1821, 1540, 1264, 994]
y2 = [1894, 1416, 966, 544, 150]
y3 = [2155, 1731, 1327, 944, 581]
y4 = [1920, 1326, 732, 224, -284]
# 绘制折线图
plt.figure(figsize=(10, 6))
plt.plot(x, y1, marker='o', linestyle='--', label='检测-拆解', c='#f9ed69')
plt.plot(x, y2, marker='s', linestyle='--', label='检测-不拆解', c='#f08a5d')
plt.plot(x, y3, marker='^', linestyle='--', label='不检测-拆解', c='#b83b5e')
plt.plot(x, y4, marker='*', linestyle='--', label='不检测-不拆解', c='#6a2c70')
# 添加标题和标签
plt.title('次品率 vs. 方案', fontsize=14)
plt.xlabel('次品率', fontsize=12)
plt.ylabel('总收益', fontsize=12)
plt.legend(fontsize=10)
plt.grid(True)
# 显示图形
plt.show()
```