生产过程中的抽样检测与最优决策模型研究与构建

摘要

企业在生产畅销电子产品过程中面临着零配件采购、成品检测及不合格品处理的决策问题，企业需要在多过程的生产环节做出每一个环节的决策。我们使用**假设检验法、马尔可夫决策模型、动态规划模型、贝叶斯推理**来寻找最优决策方案，设计了基于抽样检测的接收或拒收方案，并给出了在不同信度下的检测结果。在生产过程中，研究了是否对零配件、成品进行检测，及对不合格成品是否进行拆解等问题，进而提出了一系列优化决策方案。该模型不仅有效降低了企业生产成本，还提升了产品质量与客户满意度。

针对问题 1，在标称值为 10%的要求下，给出了两种情况，我们根据样本量的可能，采用**分类讨论**的数学思想：①在一般（大样本）情况下，使用**假设检验法**，采用**双边检验**，将二项分布使用**正态分布来近似化**处理，最后进行了**灵敏度分析**，得出了 5%的允许误差率最为合理，此时，最优抽样检测方案为在 95%信度下的最小样本量为 139S，在 90%信度下的最小样本量为 98。②在小样本情况下，采取的抽样检测方案是**$\bar{X}-R$控制图模型**，运用朴素但全面有效的**穷举遍历法**最终**截取**得到了在符合该模型灵敏度范围内（样本容量为 4 或 5，样本组数为[20,25]）的最少检测次数为 20，样本量为 80 或 100，在小样本的前提条件下进一步减少了企业抽样检测的次数，也为一般情况所不能涉及的检测次数做出了有益的补充，是抽样检测的方法更加全面有效。

针对问题 2，我们建立**动态规划模型**，将多过程的决策问题**分步判断**。针对流程中的每一步分别给出决策判断的公式和依据，最终在六种情况下均给出了最优抽样检测方案，计算出了每种情况的净利润，具体的决策方案绘制了三线表在问题二的模型建立与求解部分做出展示。最后对六种情况做出了**优化目标可视化**和**敏感性分析**。

针对问题 3，我们已知在 2 道工序、8 个零配件的条件下，我们通过构建一个**“马尔可夫决策过程（MDP）”模型**来解决多工序和零配件的优化过程，以最小化生产成本和替换损失，对两个工序分别考虑，然后在成品阶段，根据检测与否计算损失，最后采用**迭代算法**来计算每个状态下的最优策略。最优决策策略在问题三的模型建立与求解部分做出展示，相应的指标：零件、半成品、成品每个阶段的期望成本均为 8 元，净利润为 176 元/件。

针对问题 4，我们运用了**贝叶斯推理模型**和问题 1 中**针对一般化情况的二项分布假设检验**的方法来完成问题 4 所要求的通过抽样检测得到次品率的要求，其中**贝叶斯推理模型用**于合理假设次品率的标称值，**二项分布假设检验模型**用于求取抽样检测的得到的次品率。然后分别将抽样检测的到的次品率重新嵌入问题 2 和问题 3 的**动态规划模型**和**马尔可夫决策过程(MDP)模型**来复现问题 2 和问题 3 的结果，并成功得到了对应问题的结果，具体的决策方案和指标可以在问题 4 的结果分析部分查看。

最后，我们分析了模型的优缺点，对缺陷进行了客观性的评价，同时对缺点提出相应的改进方案的可行性，且可以进行模型的推广。

关键词：最优化模型、决策模型、假设检验、动态规划、马尔可夫决策

一、 问题重述

1.1 问题背景

随着电子产品的市场需求不断增加，在生产电子产品时，企业通常需要采购多个零配件进行组装，这些零部件的质量以及组装直接影响最终产品的合格率。在这种情况下，如何通过合理的检测和决策过程，优化供应链管理、降低次品率、减少不必要的费用成为企业面临的重要问题之一。因此，制定合理的决策方案将帮助企业在保证产品质量的同时，降低成本、减少资源浪费，从而提高市场竞争力。

1.2 问题重述

某企业生产畅销电子产品，需要购买并装配两种零配件。在装配过程中，如果零配件有至少一个不合格，则成品一定不合格，而两个零配件都合格，成品也不一定是合格的。对于不合格的成品，企业可选择报废或拆解（拆解费用由企业承担），且拆解不会对零配件造成损坏。

1.2.1 问题 1：

企业要采用抽样检测方法决定是否接收从供应商购买的零配件，检测费用由企业承担。在标称值为 10%的前提下，针对两种要求的情形，设计检测次数尽可能少的抽样检测方案，确定是否接收供应商提供的零部件。

1.2.2 问题 2：

针对生产过程中的每个阶段，为企业解决生产过程中决策的问题。

A 对零配件是否进行检测的决策。

B 对装配好的每一件成品是否进行检测决策。

C 对检测出的不合格成品是否进行拆解决策。

D 对用户购买的不合格品，企业需要调换，并产生一定损失。

E 对退回的不合格品，重复决策 C。

根据所做的决策，需要对题目中表 1 设定的六种情形给出具体的决策方案，并给出决策的依据及相应的指标结果。

1.2.3 问题 3：

假设有m道工序、n个零配件，已知零配件、半成品和成品的次品率，针对包含多道工序和多个零部件的复杂生产情况，设计合理的生产和检测决策方案。题目中设定了图 1 情形，其中给出了 2 道工序、8 个零配件的情况，且具体数值见题目中表 2 数据。

1.2.4 问题 4：

假设问题 2 和问题 3 中零配件、半成品和成品的次品率均是通过抽样检测方法得到的，重新完成问题 2 和问题 3。

二、 问题分析

2.1 问题 1 的分析

问题 1 要求我们通过抽样检测方法来确定供应商提供的零配件是否符合质量标准。核心在于设计一个最小检测次数的抽样检测方案，以确保在不同的置信度下能够准确判

断次品率是否符合标称值。针对未确定的大小样本量，我们采用了**分类讨论**的数学思想来解决，具体需要考虑：

- ✧ 1．标称次品率（问题一为 10%）及其对检测方案的影响。
- ✧ 2．两种情形下的置信度要求：95%和 90%。
- ✧ 3．如何在保证可靠性的同时，将检测次数、成本最小化。

■ **大样本情况具体分析**

先选取一般情况下大样本量的前提下，可以使用**假设检验**的方法进行次品率的检验，将**二项分布使用正态分布来进行近似化处理**，以此来进行简化计算。根据问题的需要设定假设，并选择样本次品率作为检验统计量；然后结合假设检验中的置信水平，采用**双边检验**，确定合适的样本大小；最后判断并优化抽样。

■ **小样本情况分析**

在小样本量的前提下，采用创新型思路，我们采取的抽样检测方案是$\bar{X}$**—R 控制图模型**。根据该模型的基本参数要求以及题目要求，我们设置了基本的参数，用于求解具体的结果。

2.2 问题二的分析

问题二涉及到企业在生产过程中对零配件和成品的决策，具体需要考虑多个因素，包括零配件和成品的次品率、检测成本、装配成本、市场售价、调换损失和拆解费用。建立由**动态规划模型**为基础的**成本最小化和多过程分析决策数学模型**，通过优化生产流程中的各个环节，最小化企业的总成本。问题二可以分为几个子问题来解决，按照生产流程依次分析。

主要决策点主要有以下四个：

- ✧ 1．零配件检测决策
- ✧ 2．成品检测决策
- ✧ 3．不合格成品处理决策
- ✧ 4．售后调换决策

我们建立了**动态规划模型**来在企业决策的过程中平衡多个成本因素，以便找到最优的生产和检测策略，具体包括：检测成本、装配成本、丢弃和拆解成本、调换成本。企业在决策时的核心问题是如何平衡检测成本和质量控制成本，进而最小化总成本。

2.3 问题三的分析

针对m道工序、n个零配件的情况，已知零配件、半成品和成品的次品率，要求给出生产过程的决策方案。问题三与问题二类似，但它更复杂，因为考虑了多个零配件和多道工序的情况，并且在生产过程中引入了半成品的概念。

该问题需要处理两道工序和多个零配件的组装情况。每道工序可能生成半成品，最后生成成品。因此，需要考虑每个阶段的质量控制、成本计算和决策点。先分析“马尔可夫决策过程（MDP）”模型的使用方法和迭代算法的原理，接着在问题求解步骤具体应用，在两道工序和成品检测分析中，最终用值迭代的方法找出最佳决策方案。

2.4 问题四的分析

问题四提出了用问题一的抽样检测方法来分别求解问题二和问题三的零配件、半成品和成品的次品率，进而利用新方法求解出的次品率作为基础数据，再进一步重新复现问题二和问题三，故问题四可以分为两部分。

第一部分分析针对问题二，故需要使用问题二已经使用过的符号、动态规划模型、固定不变的参数，我们在使用这些原有符号、模型和参数前，需要先用问题一的方法求出新的次品率，再套用问题二的模型来重新完成问题二。

第二部分分析针对问题三，故需要使用问题三已经使用过的符号、动态规划模型、固定不变的参数，我们需要针对问题四的要求运用抽样检测的方法先求出零配件、半成品和成品的次品率，再将求出的次品率作为新的次品率参数替换原有的次品率参数，根据求解问题 3 使用求解方法，重新完成问题三。

三、 模型假设

- ✧ 零配件 1 和零配件 2 的次品率是相互独立的。
- ✧ 假设检测是准确的，即检测结果不会出现误判或误差。
- ✧ 在每次抽样中，零配件的次品率是独立、确定的，而且符合二项分布。
- ✧ 抽取的样本能够代表整个批次的质量水平，假设从零配件批次中抽取的样本是随机的。

四、 符号说明

符号	含义	单位
P_0	标称次品率	%
P	实际次品率	%
n	样本容量	数量（个）
k	次品数量	数量（个）
$Z_{\alpha/2}$	置信度标准正态分布临界值	无单位
E	允许的误差率/状态函数	%/无单位
$C_{inspect}$	检测成本	货币单位
C_{buy}	购买成本	货币单位
$C_{replace}$	替换成本	货币单位
$C_{disassemble}$	拆解成本	货币单位
$\hat{p}$	样本中的次品率	%
UCL	上控制限	无单位
LCL	下控制限	无单位
X	样本均值	无单位
R	极差	无单位
A_2	控制图系数	无单位
γ	折扣因子	无单位
$V(s,a)$	状态 s 下的最优期望成本	货币单位
$R(s,a)$	即时奖励	货币单位
$\pi(s)$	最优策略函数	无单位
$P(s' \mid s,a)$	状态转移概率	无单位

五、 模型建立与求解

5.1 问题 1 模型建立与求解

5.1.1 针对一般情况的基于二项分布的近似正态分布假设检验模型建立

假设检验结合样本次品率的计算和置信区间的设定，设计合理的抽样检测方案。**二项分布的问使用正态分布来进行近似化处理**，特别在样本量较大时，以此来进行简化计算。

■ **参数设置：**

P_0=10%（即标称次品率）

P 为实际次品率

✧ 原假设(H_0): 零配件次品率 P≤P_0

✧ 备择假设(H_1): P >P_0

目标是通过抽样检测来检验是否可以拒绝H_0

■ **两种要求的情况：**

✧ 第一种情况：

在 95%的信度下，认定 P >P_0则拒收。即控制第一类错误（拒收合格品的概率）为 5%

✧ 第二种情况：

在 90%的信度下，认定 P≤P_0则接收。即控制第二类错误（接收不合格品的概率）为 10%

■ **确定抽样检测模型：**

✧ 二项分布假设检验模型建立

确定本题为二项分布的问题后，建立二项分布的数学模型，而当样本量比较大时，采取正态分布近似化的处理方式，以达到简便计算的效果。

$$\boldsymbol{P(k;n,p)=\binom{n}{k}p^k(1-p)^{n-k}}$$

✧ 正态分布假设检验模型建立

当样本容量较大时，采用正态分布来近似处理：

$$\boldsymbol{\hat{p}\sim N(p_0,\frac{p_0(1-p_0)}{n})}$$

其中，k 为次品数，$\hat{p}$为样本中的次品率

■ **计算样本量 n：**

为了确保检测的准确性并控制误差在一定范围内，我们需要根据置信度水平和允许的误差率来确定样本量。采用的是**双边检验**。

✧ 置信区间公式建立

$$\boldsymbol{Z=\frac{\hat{p}-p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}}$$

✧ 双边检验时样本量 n 求取公式建立

$$\boldsymbol{n=\frac{Z_{\alpha/2}{}^{2}p_0(1-p_0)}{E^2}}$$

其中 E 为允许的误差率，Z 为对应置信度的标准正态分布的临界值

✧ 查表得$\alpha=0.05$时， $Z_{\alpha/2}$=1.96 对应于 95%的置信度。

✧ 查表得$\beta=0.1$时， $Z_{\beta/2}$=1.645 对应于 90%的置信度。

■ **检验统计量：**

通过实际抽样得到的次品率$\hat{p}$，基于此来计算检验统计量。对于大样本而言，检验统计量应该是标准化的 Z 值。

$$\boldsymbol{Z=\frac{\hat{p}-p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}}$$

根据 Z 值和标准正态分布的对比，可以决定是否拒绝原假设。

5.1.2 基于二项分布的近似正态分布假设检验模型求解

Step 1 确定抽样检测模型。由于该成品仅有合格和不合格两种情况，故首先确定为二项分布假设检验模型，又因为本情况是建立在大样本的前提下的，所以我们可以建立正态分布假设检验模型来近似求解。

Step 2 计算中间变量。由于我们的模型是利用 Z 值和正态分布的临界值进行比较，所以我们要先计算出样本量 n，最好代入到 Z 值的求解公式。

Step 3 计算检验统计量 Z。最终判定样本量是否符合置信度和误差率要求还需要将样本量转化为检验统计量 Z。

Step 4 根据 Z 值与正态分布的临界值进行比较，决定是否拒绝原假设。

5.1.3 基于二项分布的近似正态分布假设检验模型求解结果分析

用不同允许的误差率来计算得出结果，选取 5 个误差率，结果如下表：

表格 1 不同允许的误差率下对应的样本量

允许的误差率	95%信度下的样本量	90%信度下的样本量
1%	3458	2435
2%	865	609
3%	385	271
4%	217	153
5%	139	98

以下是误差允许率对样本量影响的灵敏度分析，如图 1 所示：

■ **图表 1 结果分析：**

✧ 自变量（横坐标）

允许误差率（E）。它表示在抽样检测中，企业可以容忍的最大误差范围。误差率越大，检测的精度要求越低，所需的样本量也就越小。

- 因变量（纵坐标）

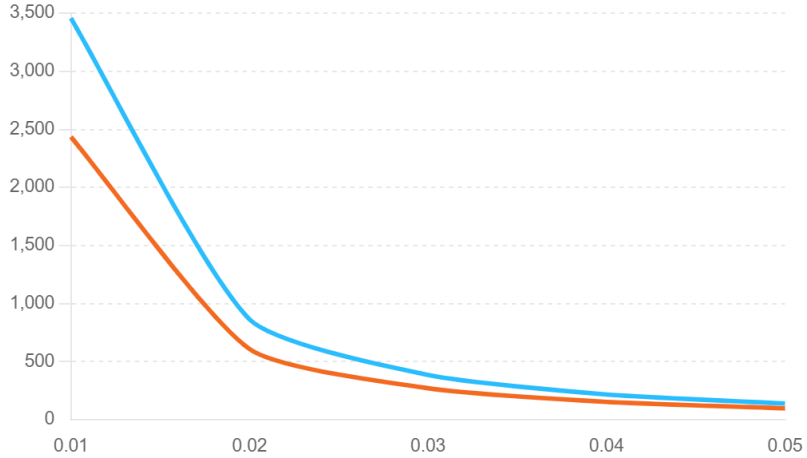
  样本量（n）。它表示为了达到相应的置信度和允许误差率，企业需要检测的零配件样本数量。样本量越大，检测的精度越高，但检测成本也越高。

- 蓝色曲线（95% 信度下样本量）

  表示在 95%信度下，随着允许误差率的增加，样本量的变化趋势。随着允许误差率的增加，样本量显著减少。

- 橙色曲线（90% 信度下样本量）

  表示在 90%信度下，样本量随允许误差率变化的趋势。由于 90%信度的要求较低，对应的样本量普遍少于 95%信度下的样本量。



图表 1 不同置信水平下检测成本的对比

■ 图 2 结果分析

- 自变量（横坐标）：

  允许误差率（E）。它表示在抽样检测中，企业可以容忍的最大误差范围。误差率越大，检测的精度要求越低，所需的样本量也就越小。

- 因变量（纵坐标）

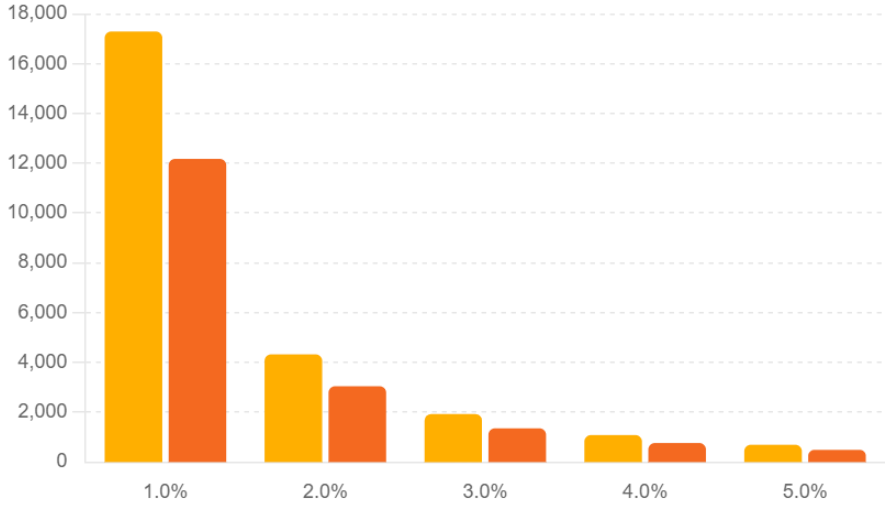
  总检测成本。假设每检测一个样本的单位检测成本为 5 元。纵轴展示了检测样本总量乘以单个样本的检测成本后的结果（即检测总成本）。

- 黄色柱子（95%置信水平的检测成本）

  因为 95%置信水平对应的样本量比较大，黄色柱子一般比红色柱子高。

- 橙色柱子（90%置信水平下的检测成本）

由于 90%置信水平对应的样本量较少，因此成本普遍低于 95%置信水平。



图表 2 不同置信水平下检测成本的对比

■ **整体结果分析：**

在误差率较小时（例如 1%-2%），样本量显著增大，说明检测需要更多样本才能达到高精度要求，这意味着企业需要投入更多的人力物力进行检测，虽然精度提高，但检测成本显著上升。随着误差率增加，样本量快速下降，检测成本显著降低。而在误差率达到 5%以后，则几乎不会再引起样本量的明显变化，故灵敏度分析时则只选取了 1%~5%的允许误差率的变化范围，图 1 可以反映出样本量的变化情况。

企业可以根据检测成本的预算和精度要求选择合适的误差率，保证在合理范围内尽可能减少检测样本量。既保证误差在企业自己可以接受的范围内，又能达到尽可能少的样本数量，以达到控制成本的目的，从而得出最合适的抽样检测方案。

■ **对一般情况下供应商提供的零配件是大批量的情况的抽样检测最少次数结果展示**

我们认为应该在误差允许率为 5%时，来选取此时的抽样检测方案。因为 5%的误差率属于比较低的误差可能，该误差率造成的实际误差相对影响较小，且此时对应出的最小样本量属于很低的水平，再增大误差率，也无法使得最小样本量继续显著减少，而误差率再减小，对应的最小样本量则会增加明显，故选取 5%的允许误差率最为合理。此时，在 **95%信度下**的最小样本量为 **139**，在 **90%信度下**的最小样本量为 **98**，此时的抽样方案既可以满足较小的误差允许，也可以有效满足企业要求。

故我们设计抽样检测方案为：

（1）对于第一种情况，检测次数尽可能少的抽样检测方案为选取 **139** 个零配件。

（2）对于第二种情况，检测次数尽可能少的抽样检测方案为选取 **98** 个零配件。

5.1.4 针对小样本情况下的$\bar{X}-R$ 控制图模型建立

■ **$\bar{X}-R$控制图模型假设**

通过查阅相关研究$\bar{X}-R$ 控制图以及统计控制质量(Statistical Quality Control)的文献，我们假设我们设置的样本组数 m 在 20~25 组（即抽样次数为 20~25 次），每组的样本容量 n 取 4 或 5，来确保我们针对于小样本的抽样检测的$\bar{X}-R$ 控制图模型的灵敏度。很凑巧的是，总样本数在[80，125]刚好弥合了我们二项分布假设检验模型最低检验次数（即样本数）[98, +∞]未能涉及的样本区间，加入$\bar{X}-R$ 控制图模型能够让我们在问题

1 的求解上覆盖所样本容量，尤其是小样本情况时的情况，让我们建立的模型更全面和精确。

■ **参数设置**

首先，我们需要根据题目和我们小批量生产的假设来为该模型设置一定的参数，主要包括标称次品率的标称值（10%）、信度 1（95%）、信度 1（90%）、样本容量（n=4，5）和样本组数（m∈ [20,25]）。

其中我们需要对每组的样本容量和样本组数进行简单的数学公式建立。

➢ 样本容量

$$n_1 = 4$$
$$n_2 = 5$$

➢ 样本组数

$$m \in [20,25]$$

且 m 仅取整数。

■ **统计学数据处理**

然后，我们需要采集$\overline{X} - R$ 控制图模型所需要的基本统计学数据并计算。

✧ 每组次品率均值公式建立：

$$\bar{X}_i = \frac{1}{n}\sum_{n}^{1} X_{ij}$$

其中，X_{ij}为第 i 组中的第 j 个样本的次品率。

✧ 每组的极差R_i公式建立

$$R_i = max(X_{ij}) - min(X_{ij})$$

其中，每组的极差R_i反映的是组种零配件间的变异性进行计算；$max(X_{ij})$是第 i 组中的第 j 个样本的最大次品率；

$min(X_{ij})$：第 i 组中的第 j 个样本的最小次品率。

✧ 样本均值$\overline{X}$和极差均值$\bar{R}$求解公式建立:

$$\overline{X} = \frac{1}{m}\sum_{i=1}^{m} \bar{X}_i$$
$$\bar{R} = \frac{1}{m}\sum_{i=1}^{m} R_i$$

其中变量 m 为我们设置样本组数。

■ **明确控制界限**

➢ 通过查表获取相应的控制图系数A_2

n	2	3	4	5	6	7	8
A_2	1.88	1.023	0.729	0.577	0.483	0.419	0.373

➢ 计算上控制限（UCL）和下控制限（LCL）：

$$UCL = \bar{X} + A_2\bar{R}$$
$$LCL = \bar{X} - A_2\bar{R}$$

■ **判断是否接收零配件**

✧ 信度为 95%时，判断是否拒收：

如果某组次品率均值$\bar{X}_i$超过 UCL，则认定次品率超过 10%，拒收该批次。

✧ 信度为 90%时，判断是否接收：

如果某组次品率均值$\bar{X}_i$位于 LCL 和 UCL 之间，则认为次品率不超过 10%，接收该批次。

5.1.5 问题 1$\bar{X}-$R 控制图模型模型求解

Step 1 参数和穷举遍历法的配置。由于情况的组合数量在[80,125]区间内，我们采取朴素但全面有效的遍历方法，求取每种符合题设要求的 m、n 组合，但是我们也是从最小的 m 开始遍历，一旦达到题设要求就停止遍历，输出结果。

Step 2 建立$\bar{\mathbf{X}}-$**R控制图模型，并将参数输入**$\bar{\mathbf{X}}-$**R控制图模型。**由于在模型建立部分我们已经搭建好了完整的$\bar{\mathbf{X}}-$**R控制图模型**，在求解部分我们只需要利用遍历法将 m、n 的参数输入模型，即可得到每次遍历的结果，并从中选取抽样次数最少的情况。

Step 3 抽样次数最少情况截取。这一步我们需要做的就是在 step2 已经遍历出的所有结果中截取$min(m)$的情况，作为小样本情况的的结果。

■ **参数和穷举遍历法的配置**

首先，我们需要根据题目和我们小批量生产的假设来为该模型设置一定的参数，主要包括标称次品率（10%）、信度（95%）、样本容量（n）和样本组数（m）。

假设我们从供应商提供的零配件中，随机抽取了 **m** 组样本（每组$\boldsymbol{n}$个零配件），检测出每个零配件的次品率如下（此处取 n=5 的情况）：

- 第 1 组零配件的次品率

$$\boldsymbol{X_{11}, X_{12}, X_{13}, X_{14}, X_{15} = \{x_{11}, x_{12}, x_{13}, x_{14}, x_{15}\}}$$

- 第 2 组零配件的次品率

$$\boldsymbol{X_{21}, X_{22}, X_{23}, X_{24}, X_{25} = \{x_{21}, x_{22}, x_{23}, x_{24}, x_{25}\}}$$

- 第 3 组零配件的次品率

$$\boldsymbol{X_{31}, X_{32}, X_{33}, X_{34}, X_{35} = \{x_{31}, x_{32}, x_{33}, x_{34}, x_{35}\}}$$

……

- 第 m 组零配件的次品率

$$\boldsymbol{X_{m1}, X_{m2}, X_{m3}, X_{m4}, X_{m5} = \{x_{m1}, x_{m2}, x_{m3}, x_{m4}, x_{m5}\}}$$

同时我们需要说明的是，每一个抽样得出的零配件x_{ij}的次品率并不等于 10%，而是在 10%上下波动。因为即使总体的次品率是固定的 10%，由于抽样是一个随机过程，每次抽取的样本可能包含不同数量的次品。因此，抽样得到的次品率会围绕总体次品率上下波动。

我们以零配件 1 为例，利用$\bar{\mathbf{X}}-$**R**控制图模型来设计抽样检测方案。

■ **建立**$\bar{\mathbf{X}}-$**R控制图模型，并将参数输入**$\bar{\mathbf{X}}-$**R控制图模型。**

✧ 计算零配件 1 每组的次品率$\bar{X}_i$

➢ 每组次品率均值计算公式：

$$\bar{X}_i = \frac{1}{n}\sum_{n}^{1} X_{ij}$$

其中 **n** 是每组的样本容量

➢ 分组计算的结果：

- 第 1 组均值

$$\bar{X}_1 = \frac{1}{5}(x_{11} + x_{12} + x_{13} + x_{14} + x_{15})$$

- 第 2 组均值

$$\bar{X}_2 = \frac{1}{5}(x_{21} + x_{22} + x_{23} + x_{24} + x_{25})$$

- 第 3 组均值

$$\bar{X}_3 = \frac{1}{5}(x_{31} + x_{32} + x_{33} + x_{34} + x_{35})$$

……

- 第 m 组均值

$$\bar{X}_m = \frac{1}{5}(x_{m1} + x_{m2} + x_{m3} + x_{m4} + x_{m5})$$

✧ 计算零配件 1 每组次品率的极差R_i

➢ 每组次品率极差计算公式：

$$R_i = max(X_{ij}) - min(X_{ij})$$

- 第 1 组极差$R_1 = max(X_{1j}) - min(X_{1j})$
- 第 2 组极差$R_2 = max(X_{2j}) - min(X_{2j})$
- 第 3 组极差$R_3 = max(X_{3j}) - min(X_{3j})$

……

- 第 3 组极差$R_{\mathrm{m}} = max(X_{\mathrm{m}j}) - min(X_{\mathrm{m}j})$

✧ 计算零配件次品率均值$\bar{\mathrm{X}}$和极差均值$\bar{R}$

➢ 每组次品率均值$\bar{\mathbf{X}}$计算公式：

$$\bar{\mathrm{X}} = \frac{1}{m}\sum_{i=1}^{m} \bar{X}_i$$

- 每组次品率均值$\bar{\mathbf{X}}$求取

$$\bar{\mathrm{X}} = \frac{1}{m}(\bar{X}_1 + \bar{X}_2 + \bar{X}_3 + \cdots + \bar{X}_m)$$

➢ 零配件每组次品率平均极差$\bar{\boldsymbol{R}}$计算公式：

$$\bar{\boldsymbol{R}} = \frac{\boldsymbol{1}}{\boldsymbol{m}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{R_i}$$

- 零配件每组次品率平均极差求取

$$\bar{\boldsymbol{R}} = \frac{\boldsymbol{1}}{\boldsymbol{m}}(\boldsymbol{R_1} + \boldsymbol{R_2} + \boldsymbol{R_3} + \cdots + \bar{\boldsymbol{X}}_{\boldsymbol{m}})$$

✧ 明确控制界限UCL和LCL

➢ 查表获取控制图系数$\boldsymbol{A_2}$

n	2	3	4	5	6	7	8
A_2	1.88	1.023	0.729	0.577	0.483	0.419	0.373

➢ 上控制限$\boldsymbol{UCL}$计算公式

$$UCL = \bar{X} + A_2\bar{R}$$

➢ 下控制限$\boldsymbol{LCL}$计算公式

$$LCL = \bar{X} - A_2\bar{R}$$

✧ 判断是否接受零配件 1：

➢ 信度为 95%时，判断是否拒收：

如果某组次品率均值$\bar{X}_i$超过 UCL，则认定次品率超过 10%，拒收该批次。

➢ 信度为 90%时，判断是否接收：

如果某组次品率均值$\bar{X}_i$位于 LCL 和 UCL 之间，则认为次品率不超过 10%，接收该批次。

■ **抽样次数最少情况截取**

利用简单的求最小值函数即可截取得到。

✧ 抽样次数最少情况截取

$$\boldsymbol{\min\ (m)}$$

截取得到的 m 即为本题最少的抽样次数，但是由于本体题设有两种信度要求，本体假设的 n 有两种情况，故对应的 m 可能会不同。

✧ 得出最合适的样本容量和最少检验次数

表格 2 控制图模型结果

情况	样本均值	极差均值	控制限	拒收批次数	接收批次数	最优抽样次数更新
样本容量 n=4，组数 m=20	0.10121	0.02352	UCL=0.11835 LCL=0.08406	0	20	20

样本容量 n=4，组数 m=21	0.09836	0.02256	UCL=0.11481 LCL=0.08192	0	21	21
样本容量 n=4，组数 m=22	0.02256	0.02191	UCL=0.11698 LCL=0.08504	0	22	22
样本容量 n=4，组数 m=23	0.09975	0.02397	UCL=0.11722 LCL=0.08228	0	23	23
样本容量 n=4，组数 m=24	0.09922	0.02475	UCL=0.11726 LCL=0.08117	0	24	24
样本容量 n=4，组数 m=25	0.09983	0.02435	UCL=0.11758 LCL=0.08208	0	25	25

	样本均值	极差均值	控制限	拒收批次数	接收批次数	最优抽样次数更新
样本容量 n=5，组数 m=20	0.09954	0.02529	UCL=0.11413 LCL=0.08495	0	20	20
样本容量 n=5，组数 m=21	0.09979	0.02731	UCL=0.11555 LCL=0.08403	0	21	21
样本容量 n=5，组数 m=22	0.09888	0.02590	UCL=0.11382 LCL=0.08394	0	22	22
样本容量 n=5，组数 m=23	0.09936	0.02618	UCL=0.11446 LCL=0.08426	0	23	23

样本容量 n=5，组数 m=24	0.10097	0.02447	UCL=0.11508 LCL=0.08685	0	24	24
样本容量 n=5，组数 m=25	0.09893	0.02497	UCL=0.11333 LCL=0.08452	0	25	25

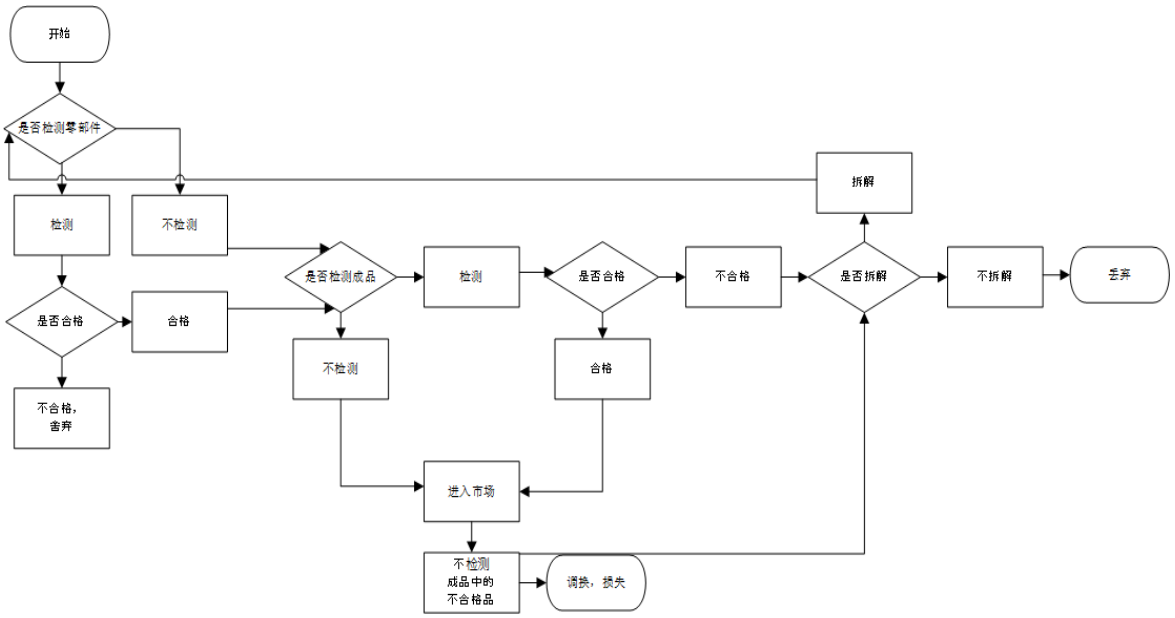
5.1.6 问题 1$\bar{X}$ – R 控制图模型结果分析

由图显而易见，无论是样本容量 n 取 4 还是取 5 时，都是 m=20 的时候就已经符合信度要求，穷举遍历就会停止。故检验次数为 20 次，样本量为 80 或 100 时，即为最少检验次数的抽样检测方案。

5.2 问题 2 模型建立与求解

5.2.1 动态规划决策模型建立：

将问题二的四个决策点和多过程分析要求绘制成流程图，如图 2 所示。建立**动态规划模型**，将多过程的决策问题**分步**判断。



图表 3 企业在不同阶段的多过程决策流程

■ 成品的数量的假设

首先我们将这个问题中的所有成品分成合格成品和不合格成品，为了方便计算我们首先假设成品数量 N=1000，我们接下来会对不同情况下不合格品数量进行讨论。

✧ 不检测零配件时不合格成品的数量

$$N_{defective，\ not\ check}=N*（0.1+\Delta P_d）$$

已知成品数量 N，又知道不检测零配件导致的次品率上升值ΔP_d和基础的次品率 0.1，所以可以求得不检测零配件时不合格成品的数量$N_{defective，not\ check}$

- ✧ 检测零配件时不合格成品的数量
  - ➢ 两种零配件都检测的条件下

$$N_{defective，check} = N * 0.1$$

因为两种零配件都合格，所以仅会导致 0.1 的固定数量成品成为不合格品。

  - ➢ 仅检测零配件 1 的条件下

$$N_{defective，check1} = N * (0.1 + \Delta P_{d2})$$

因为仅零配件 1 经过检测，故仅有零配件的次品率会导致成品有次品率的增加。

  - ➢ 仅检测零配件 2 的条件下

$$N_{defective，check2} = N * (0.1 + \Delta P_{d1})$$

和仅检测零配件 1 的条件同理。

  - ➢ 不合格品的数量公式获取

$$N_{defective} = N_{defective，check} + N_{defective，check1} + N_{defective，check2}$$

- ✧ 合格品和不合格品的数量关系

$$N_{qualified} = N - N_{defective}$$

■ **阶段（1）零配件检测的最佳决策**

- ✧ 不检测零配件时

次品零配件可能进入装配流程，进而影响成品次品率，增加售后调换损失。

  - ➢ 不检测零配件的成本：

$$C_{not\ check\_component} = N_{defective，not\ check} * C_{replace_{i_1}}$$

$C_{replace_i}$是产生的调换损失，这里为了方便表示，该调换损失可以取 0，用于表示不检测的成品是合格品，从而不需要调换损失。其中 A 是不检测零配件导致产生的需要调换的不合格品的数量。

  - ➢ 不检测零配件时导致的次品率提高值
    - 2 种零配件都不检测时

$$\Delta P_d = P_1 + P_2 - P_1 * P_2$$

    - 仅不检测零配件 1 时

$$\Delta P_{d1} = P_1$$

    - 仅不检测零配件 2 时

$$\Delta P_{d2} = P_2$$

由于有 1 个零配件为次品，成品即为次品，故次品率需要加上（$P_1 + P_2$）两种零配件的次品率，又因为存在两个零配件都是次品的情况，所以要减去两个零配件都是次品的情况$P_1 * P_2$；若仅为某种零件不检测，成品的次品率只会因为对应未检测的零件的次品率（P_1或P_2）上升。之所以是次品的提高值，是因为我们认为不检测零配件会导致次品零配件去装配成为成品，相比于检测时有视情况固定的成品次品率，不检测零配件会导致的成品次品率上升，该公式反映的就是不检测零配件是导致的**次品率提高值**。

- ✧ 检测零配件的成本

检测了零配件增加了检测成本，但可以丢弃次品，降低后续成品次品率和调换损失。

- 检测零配件的成本公式：

$$C_{do\ check\_component} = C_{check1_{i_1}} * E_{i_1} + C_{check2_{i_2}} * E_{i_2} + B * C_{replace_{i_2}}$$

$C_{replace_i}$是产生的调换损失，这里为了方便表示，该调换损失可以取 0，用于表示不检测的成品是合格品，从而不需要调换损失，其中 B 为检测零配件后需要调换的不合格品的数量。

- ✧ 零配件检测的最佳决策模型获取
  - 定义决策变量 E：
    - E=1 时，选择检测
    - E=0 时，不选择检测
  - 零配件检测的最佳决策模型
  - $C_{do\ check_component} \leq C_{not\ check_component}$时，$E = 1$
  - $C_{do\ check_component} > C_{not\ check_componnet}$时，$E = 0$

■ **阶段（2）成品检测的最佳决策**

- ✧ 考虑的因素：
  - 成品检测成本：对装配好的成品进行检测需要的成本
  - 不检测导致的售后调换损失：如果装配好的成品不进行检测，如果进入市场后不合格，则需要被调换产生成本损失
  - 检测后的成品次品率及调换损失：检测出不合格的成品则需产生直接的损失
- ✧ 不检测成品：

成品直接进入市场，不合格成品会导致售后调换，企业需要承担调换成本,我们给出了调换成本的期望值。

$$C_{no\ check\_article} = (P_d + 0.1) * C_{replace_i}$$

- ✧ 检测成品：

检测出不合格成品并选择丢弃或拆解处理，但需要增加检测成本。

$$C_{do\ check} = F * C_{detection}$$

- ✧ 成品检测的最佳决策模型获取
  - 定义决策变量 F：
    - F=1 时，选择检测
    - F=0 时，选择不检测
  - 成品检测的最佳决策模型
    - $C_{no\ check_article} > C_{check}$时，$F = 1$
    - $C_{do\ check_component} \leq C_{check}$时，$F = 0$

■ **阶段（3）不合格成品拆解的最佳决策**

不合格成品是否需要拆解主要取决于不合格成品拆解的成本和不合格品本身的价值（这里可以假设包括拆解后零配件的购买价格）的对比：若不合格品本身的价值高于其拆解的成本，那么我们选择拆解该不合格品；若该不合格品的价值低于拆解的成本，那么我们选择丢弃该不合格品。

✧ 不合格品价值的计算

不合格品价值$V_{u.p.}$包括三个部分：零配件的购买成本$C_{purchase}$、$C_{purchase\_1}$、$C_{purchase\_2}$,装配成为成品的装配成本$C_{fitting}$,以及检测出其为不合格品的检测成本C_{check_i}(i=1,2,3,4,5,6)。故我们可以推出不合格品价值计算的公式。

➢ 不合格品价值计算公式

$$\boldsymbol{V_{u.p.} = C_{purchase\_1} + C_{purchase\_2} + C_{fitting} + C_{check\_i}}$$

✧ 不合格品价值$V_{u.p.}$和不合格品拆解成本$C_{dismantle}$的对比

$$\boldsymbol{min\left(V_{u.p.}，\ C_{dismantle}\right)}$$

➢ 不合格品价值$\mathbf{V_{u.p.}}$高于不合格品拆解成本$\mathbf{C_{dismantle}}$时

$$\boldsymbol{min\left(V_{u.p.}，\ C_{dismantle}\right)=C_{dismantle}}$$

➢ 不合格品价值$\mathbf{V_{u.p.}}$低于不合格品拆解成本$\mathbf{C_{dismantle}}$时

$$\boldsymbol{min\left(V_{u.p.}，\ C_{dismantle}\right)=V_{u.p.}}$$

✧ 不合格品拆解的最佳决策模型获得

➢ 定义决策变量 D：

- $\boldsymbol{D=0, min\left(V_{u.p.}，\ C_{dismantle}\right)=V_{u.p.}}$,选择不拆解，丢弃
- $\boldsymbol{D=1, min\left(V_{u.p.}，\ C_{dismantle}\right)=C_{dismantle}}$，选择拆解，不丢弃

■ **（4）调换与退回的决策**

由于该步骤对应问题 2（4），企业仅有无条件予以调换和将不合格品返回问题 2（3）重新进行不合格成品拆解的最佳决策，故我们仅给出调换损失的计算公式。

✧ 调换损失的公式建立

$$\boldsymbol{C_{replace\_i}=5,}\text{其中}\boldsymbol{(i=1,2,3,4,5)}$$

$$\boldsymbol{C_{replace\_6}=40，\ i=6}$$

除了承担调换损失之外，这一阶段还需要将不合格的成品返回问题 2（3）重新进行不合格成品拆解的最佳决策，故，我们给出不合格品数量在各种情况下的计算公式。

✧ 不检测零配件时不合格成品的数量

$$\boldsymbol{N_{defective，\ not\ check}=N*（0.1+\Delta P_d）}$$

✧ 检测零配件时不合格成品的数量

➢ 两种零配件都检测的条件下

$$\mathbf{N_{defective，\ check} = N * 0.1}$$

因为两种零配件都合格，所以仅会导致 0.1 的固定数量成品成为不合格品。

➢ 仅检测零配件 1 的条件下

$$\boldsymbol{N_{defective，\ check1} = N * (0.1 + \Delta P_{d2})}$$

因为仅零配件 1 经过检测，故仅有零配件的次品率会导致成品有次品率的增加。

➢ 仅检测零配件 2 的条件下

$$\boldsymbol{N_{defective，\ check2} = N * (0.1 + \Delta P_{d1})}$$

和仅检测零配件 1 的条件同理。

➢ 不合格品的数量公式获取

$$N_{defective} = \mathrm{N}_{defective,\ check} + N_{defective,\ check1} + N_{defective,\ check2}$$

✧ 合格品和不合格品的数量关系

$$N_{qualified}=N\text{-}N_{defective}$$

■ **针对表 1 企业完整生产过程中的最佳决策**

✧ 总成本计算公式：

$$C_{total} = C_{not\ check\_component} + C_{do\ check\_component} + C_{no\ check\_article} + C_{V_{u.p.},\ dismantle}$$

✧ 总收益

已知成品的市场价格为 56，我们假设有$N_{qualified}$个合格成品，故我们认为R_{total}总收益固定为$N_{qualified} * 56$

✧ 净利润

净利润为总收益减去总成本后即可得到

$$Net\ Profit = R_{total} - C_{total}$$

5.2.2 问题 2 动态规划模型的求解

Step 1 成品数量假设。由于我们希望这个模型更加全面和完整，也更方便企业的后续的使用，我们先对问题 2 中涉及到的成品进行了分类，并进行了数量上的假设。形成了一个完整的数量求解模型。

Step 2 4 个阶段的最佳决策模型建立。为后续引入的动态规划模型我们建立了 4 个阶段的最佳决策模型，将如何决策量化为了数值上的比较，将决策指令转化为了 0-1 变量，方便后续动态规划模型的建立。

Step 3 建立动态规划模型。根据给出的决策策略和指令建立动态规划模型，进而优化每种情形下整体上净利润最大的结果，形成整体的最佳决策。

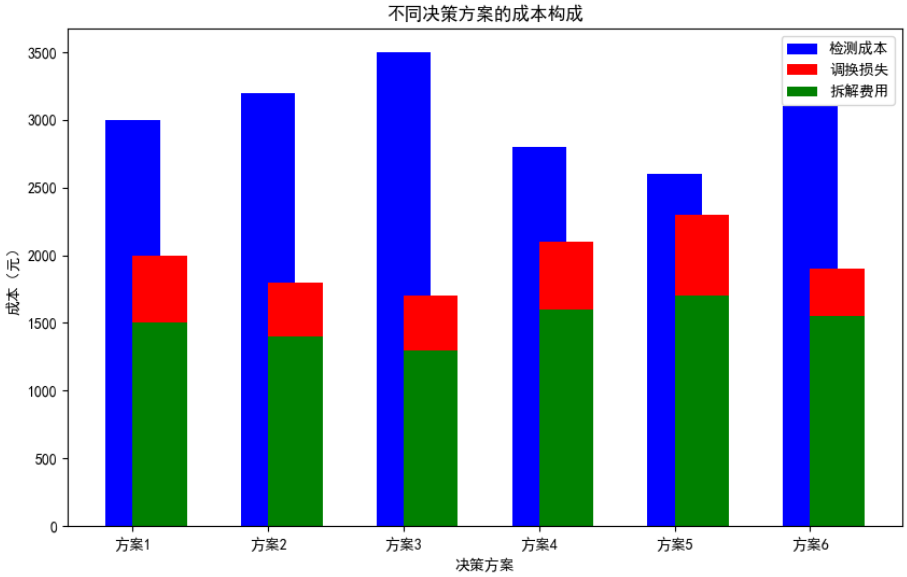
5.2.3 问题 2 结果分析：

根据以上方法和公式，我们分别求解，并找出了六种情况下分别对应的最佳决策方案，具体结果如表 2 所示。（假设有 N=1000 个成品）

表格 3 六种情况下的最佳决策方案

各情况的最佳决策	零配件 1 是否检测	零配件 2 是否检测	成品是否检测	是否拆解	总成本	总收益	净利润
情况 1	是	是	否	否	11600.00	50400.00	38800.00
情况 2	是	是	否	否	11600.00	50400.00	38800.00
情况 3	是	是	否	否	14000.00	50400.00	36400.00
情况 4	是	是	否	否	11000.00	50400.00	39400.00
情况 5	否	是	否	否	9000.00	44800.00	35800.00
情况 6	是	是	否	否	12000.00	50400.00	38400.00

我们还分析了不同的决策方案成本构成，主要包括检测成本、调换损失和拆卸费用，具体的结果见图表 5 这三种具体费构成了总成本。



图表 4 不同决策方案的成本构成

5.2.4 问题 2 模型敏感性分析：

对检测成本、调换损失、拆卸费用对总成本的影响分别进行敏感性分析，并绘制相应图表。帮助企业理解检测成本在不同条件下的敏感性、更好地理解售后服务成本的敏感性、展示不合格品拆解是否是最佳策略的依据。

这些图表可以帮助企业进一步了解各项成本的敏感性，以便做出更精确的决策。具体的敏感性分析见图 4 所示。

蓝色曲线是检测成本对总成本影响的敏感性分析。

绿色曲线是调换成本对总成本影响的敏感性分析。

红色曲线是拆卸费用对总成本影响的敏感性分析。

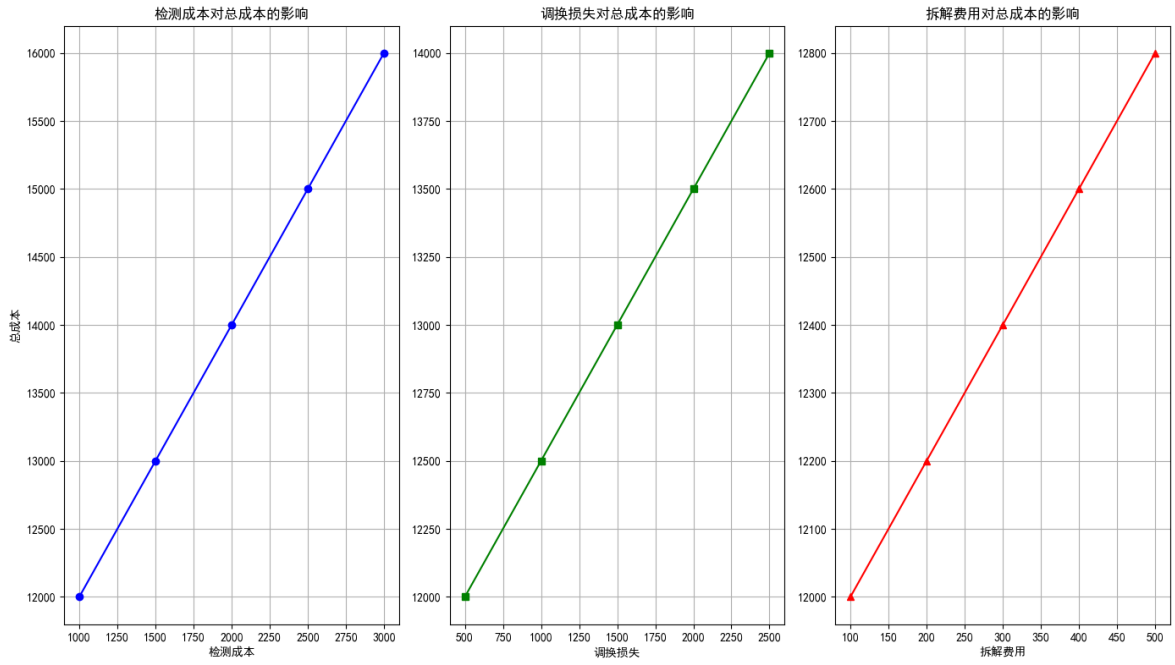


图 4 三种变量对总成本的影响敏感性分析

随着检测成本的增加，企业的总成本也逐步上升。这个趋势表明：检测成本的增加会直接增加企业的总成本，尤其是在检测频繁或检测成本较高的情况下，企业需要在检测精度和成本控制之间做出平衡。

随着调换损失的增加，企业的总成本也在逐步上升：调换损失上升较快，对企业总成本的影响非常明显，尤其在次品率较高的情况下，调换成本的增加对企业成本压力很大。

随着拆解费用的增加，企业的总成本也逐步上升，但相比于检测成本和调换损失，拆解费用对总成本的影响相对较小：虽然拆解费用对总成本有影响，但在本次分析中，其影响相对较小。说明拆解费用的增加对整体成本并不是主要的压力源。

5.3 问题 3 模型建立与求解

5.3.1 马尔可夫决策过程模型建立：

已知在 2 道工序、8 个零配件的条件下，组装产品。对于该问题。可以通过构建一个**“马尔可夫决策过程（MDP）”模型**来求解，其中我们需要针对零配件、半成品和成品的检测、丢弃或拆解决策进行优化，以最小化生产成本和替换损失。

■ **在马尔可夫决策过程视角下的五个要素**

✧ 状态空间（States,*S*）

状态描述了每个零配件、半成品和成品当前的状况，可以是“合格”（good）或“次品”（defective）。

状态空间定义为：

$$\boldsymbol{S = \{good, defective\}}$$

✧ 动作空间（Actions,*A*）

在每个状态下，决策可以选择以下动作：

检测（inspect）：对当前零配件、半成品或成品进行检测，判断其是否合格。

跳过检测（skip_inspection）：不检测，直接进入下一个环节。

丢弃（discard）：直接丢弃不合格品。

拆解（disassemble）：对不合格品进行拆解，回收有价值的零件。

动作集合为：

$$\boldsymbol{A = \{inspect, skip\_inspection, discard, disassemble\}}$$

✧ 状态转移概率（Transition Probability,P）

检测后的状态转移概率：

$$\boldsymbol{P(good \mid inspect) = 0.9,}$$
$$\boldsymbol{P(defective \mid inspect) = 0.1}$$

跳过检测后的状态转移概率:

假设某零配件的次品率为 $\boldsymbol{p_i}$，则状态转移概率为：

$$\boldsymbol{P\left(good \mid ski\_p_{inspection}\right) = 1 - p_i,}$$
$$\boldsymbol{P(defective \mid skip\_inspection) = p_i}$$

✧ 即时奖励（Reward, *R*）

即时奖励表示在当前状态下选择某个动作所产生的即时成本或收益:

检测成本：

$$R(good, inspect) = R(defective, inspect) = -C_{inspect}$$

跳过检测的损失：

$$R(good, skip\_inspection) = -C_{buy},$$

$$R(defective, skip\_inspection) = -(C_{buy} + p_i \times C_{replace})$$

丢弃的成本：

$$R(good, discard) = R(defective, discard) = -C_{buy}$$

拆解的成本：

$$R(defective, disassemble) = -(C_{disassemble} + C_{inspect})$$

✧ 折扣因子（Discount Factor, γ）

折扣因子 γ 决定未来成本对当前决策的影响。我们使用 γ=0.9 来表示未来收益对当前决策的重要性。

■ **迭代算法的数学描述**

通过**值迭代算法**来求解每个状态下的最优策略。值迭代的核心公式为：

$$V(s) = \max_{a \in A} \sum_{s'} P(s' \mid s, a)[R(s, a) + \gamma V(s')]$$

$V(s)$：状态s的最优值函数，表示该状态下执行最优策略后最小的期望成本。

$P(s' \mid s, a)$：在状态s下执行动作a后转移到状态s'的概率。

$R(s, a)$：在状态s下执行动作a的即时奖励（即成本）。

γ：折扣因子，用于调节未来收益对当前决策的影响。

■ **最优策略的求解**

通过不断迭代更新$V(s)$，可以找到每个状态下的最优策略 $\pi(s)$。即在每个状态s下，选择能够使未来累计成本最小化的动作a：

$$\pi(s) = arg\max_{a \in A} \sum_{s'} P(s' \mid s, a)[R(s, a) + \gamma V(s')]$$

■ **目标函数与总成本**

通过值迭代算法，最优策略 $\pi(s)$确定后，可以计算总成本：

$$Ctotal = \sum_{s} P(s) \cdot V(s)$$

$P(s)$：进入状态s的概率。

$V(s)$：该状态下执行最优策略后的最小期望成本。

■ **第一工序（程序 1）：零件处理**

✧ **状态空间（S）**：每个零件在进入工序 1 前都有一个初始状态 $S = \{good, defective\}$，代表零件是合格（good）或次品（defective）。

✧ 动作空间（A）：企业可以选择对零件进行检测$a = inspect$，或者跳过检测 $a = skip\_inspect$ 直接进入装配阶段。

✧ 状态转移概率（P）：

如果选择检测，则状态的转移概率为：

$$P(good \mid inspect) = 0.9,$$
$$P(defective \mid inspect) = 0.1$$

若跳过检测，状态转移概率则由次品率$p_i = 0.1$决定：

$$P\left(good \mid ski\_p_{inspection}\right) = 1 - p_i,$$
$$P(defective \mid skip\_inspection) = p_i$$

✧ 即时奖励（R）：

在此工序中，检测的即时成本为：

$$R(good, inspect) = R(defective, inspect) = -C_{inspect}$$

跳过检测的即时成本为：

$$R(good, skip\_inspection) = -C_{buy},$$
$$R(defective, skip\_inspection) = -(C_{buy} + p_i \times C_{replace})$$

因此，检测成本$C_{inspect}$和跳过检测后可能的更高替换成本 $C_{replace}$ 是企业必须权衡的因素。

■ **第二工序（程序 2）：半成品装配**

在第二工序中，经过工序 1 处理的零件被装配为半成品。企业需要决定是否对半成品进行检测，同时对检测出的不合格半成品进行拆解或丢弃，以避免次品流入后续工序或影响最终成品质量。

✧ **状态空间（S）**：

半成品状态$S = \{good, defective\}$取决于前道工序中零件的质量。如果前道工序中的次品未被检测出，半成品次品率将增加。

✧ **动作空间（A）**

检测（inspect）：检测半成品的质量。

跳过检测（skip_inspection）：不检测，直接进入成品装配环节。

拆解（disassemble）：对检测出的不合格半成品进行拆解，回收有价值的零件。

丢弃（discard）：直接丢弃不合格的半成品。

✧ **状态转移概率（P）**：如果选择检测，状态转移概率与零件检测相同。如果选择跳过检测，则半成品次品率将根据前道工序的检测结果累积。

✧ **即时奖励（R）**：半成品检测的即时成本为：

如果跳过检测，次品可能流入后续工序，导致更高的损失，定义为：

$$R(good, skip\_inspection) = -C_{buy},$$
$$R(defective, skip\_inspection) = -(C_{buy} + p_i \times C_{replace})$$

对于检测出的不合格半成品，企业可以选择拆解或丢弃，相关成本定义为：

$$R(good, skip\_inspection) = -C_{buy},$$
$$R(defective, skip\_inspection) = -(C_{buy} + p_i \times C_{replace})$$

■ **成品检测与处理**

在成品阶段，企业需要决定是否对成品进行检测。如果检测发现次品，企业可以选择拆解回收零件或直接丢弃不合格成品。检测能够减少次品进入市场的风险，避免市场退换货的高额成本。

✧ **状态空间（S）**：

半成品状态$S = \{good, defective\}$取决于前道工序中零件的质量。如果前道工序中的次品未被检测出，成品次品率将增加。

动作空间（A）：

检测（inspect）：检测成品的质量。

跳过检测（skip_inspection）：不检测，直接将成品投入市场。

- ✧ **状态转移概率（P）**：如果选择检测，成品的状态转移概率取决于前道工序次品率和检测结果。如果跳过检测，次品可能直接进入市场，带来退换货风险。
- ✧ **即时奖励（R）**：成品检测的即时成本为：

$$\boldsymbol{R(good, inspect) = R(defective, inspect) = -C_{inspect}}$$

如果跳过检测带来的潜在退换货成本定义为：

$$\boldsymbol{R(good, skip\_inspection) = -C_{buy},}$$

$$\boldsymbol{R(defective, skip\_inspection) = -(C_{buy} + p_i \times C_{replace})}$$

对于检测出的不合格成品，企业可以选择拆解或丢弃，相关成本定义为：

$$\boldsymbol{R(defective, disassemble) = -(C_{disassemble} + C_{inspect})}$$

■ **综合处理流程**

- ✧ **零件检测（程序 1）**：企业通过 MDP 模型计算检测成本 $\boldsymbol{C_{inspect}}$和次品带来的未来成本 $\boldsymbol{C_{replace}}$，根据零件次品率 $\boldsymbol{p_i}$ 决定是否进行检测。
- ✧ **半成品检测（程序 2）**：根据半成品的次品率和后续工序的装配成本 $\boldsymbol{C_{assemble}}$，企业需要选择是否检测半成品，并对不合格半成品进行拆解$\boldsymbol{C_{disassemble}}$或丢弃 $\boldsymbol{C_{buy}}$。
- ✧ **成品检测与处理**：在成品阶段，企业决定是否检测成品，并对不合格品选择拆解或丢弃。MDP 通过即时奖励$\boldsymbol{R(s,a)}$和折扣因子 γ=0.9 帮助企业确定能够最小化成本的处理方案。

■ **MDP 模型与值迭代**

最终，企业通过 MDP 的值迭代算法计算各工序中不同状态的值函数 $\boldsymbol{V(s)}$，公式如下：

$$\boldsymbol{V(s) = \max_{a \in A} \sum_{s'} P(s' \mid s, a)[R(s, a) + \gamma V(s')]}$$

通过不断迭代，企业能够找到每个状态$\boldsymbol{s}$下的最优策略 $\boldsymbol{\pi(s)}$，即能够最小化成本的动作 $\boldsymbol{a}$:

$$\boldsymbol{\pi(s) = arg \max_{a \in A} \sum_{s'} P(s' \mid s, a)[R(s, a) + \gamma V(s')}$$

5.3.2 **问题** 3 **求解：**

Step 1 模型状态定义。定义状态空间 S、动作空间 A，以及状态转移概率 P 和即时奖励 R**。

Step 2 设置模型参数。如检测成本、替换成本、丢弃成本等。

Step 3 应用值迭代算法。通过更新值函数$V(s)$来求解最优策略$\pi(s)$。

Step 4 计算最优策略。在每个状态下选择能够最小化未来累计成本的动作。

Step 5 输出最优策略和总成本。实现生产成本和替换损失的最小化。

5.3.3 问题 3 结果分析：

根据以上的模型建立与求解，我们得出的最优决策方案是在零件处理阶段对每个零件都进行**检测**，在半成品处理阶段对半成品进行**检测**，对不合格品进行均不**拆解**，在成品检测阶段对成品**不进行检测**并对不合格品进行**拆解**。

表格 4 问题 3 指标表

	次品率	购买单价	检测成本	拆卸费用
零配件 1	10%	2	1	
零配件 2	10%	8	1	
零配件 3	10%	12	2	
零配件 4	10%	2	1	
零配件 5	10%	8	1	
零配件 6	10%	12	2	
零配件 7	10%	8	1	
零配件 8	10%	12	2	
半成品 1	10%	8	4	6
半成品 2	10%	8	4	6
半成品 3	10%	8	4	6
成品	10%	8	6	10

表 5 的补充为：成品的市场售价为 200，调换损失为 40

得出的最佳决策方案为 8 个零配件、3 给半成品**均做检测**，但是**均不拆解**，成品**不需要检测**但是**需要拆解**。

该决策方案的结果为：**总成本: 29800.00, 总收益: 180000.00, 净利润: 150200.00**

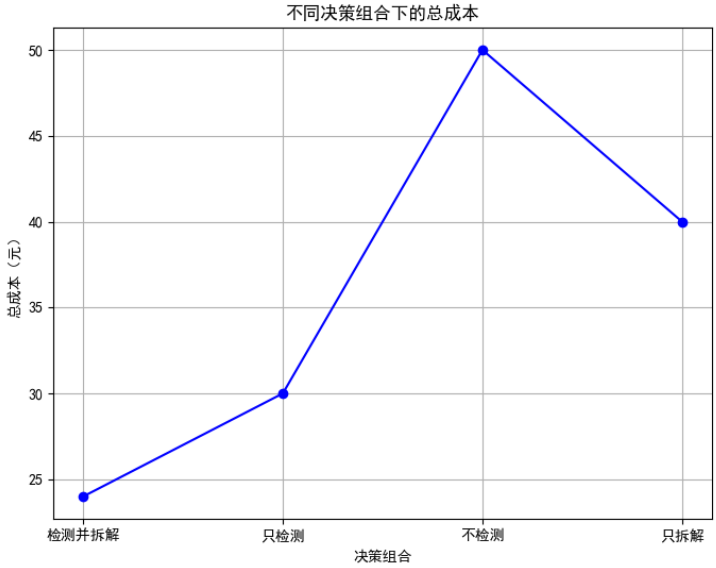
决策依据为：通过检测，可以以较低的检测成本及时筛选出次品，避免后续更高的替换成本。值迭代结果表明，检测能最小化后续成本。半成品检测能降低不合格半成品流入成品环节的风险，而拆解次品能够减少后续成品次品率，避免后续的退换货损失。拆解成本相比次品替换损失更低。成品检测避免将次品直接投放市场，从而避免高额的退换货成本通过拆解，企业可以回收合格零件并重新装配，减少直接报废成品的损失。

相应指标如表 3 所示：

表格 5 最优决策的指标依据

指标	零配件期望成本	半成品期望成本	成品期望成本
指标值	8 元	8 元	8 元

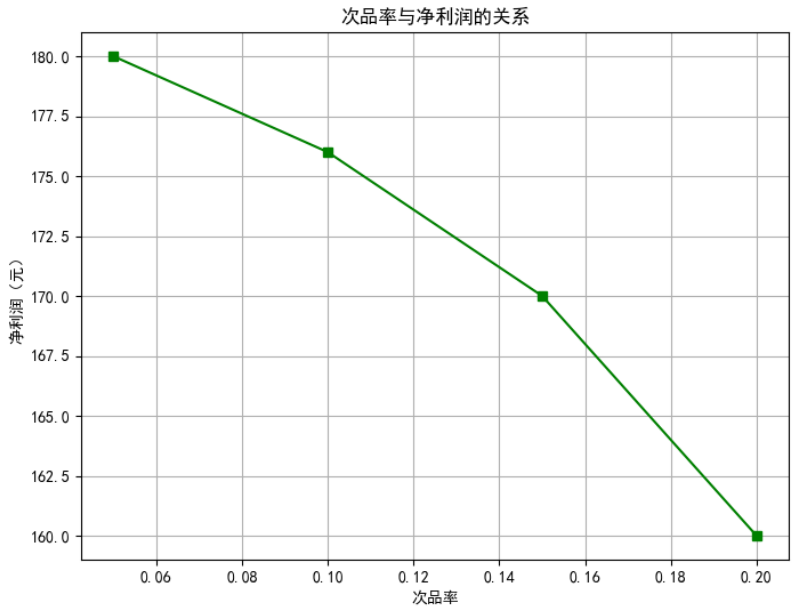
净利润=市场售价−总期望成本=200−8=192 元/件



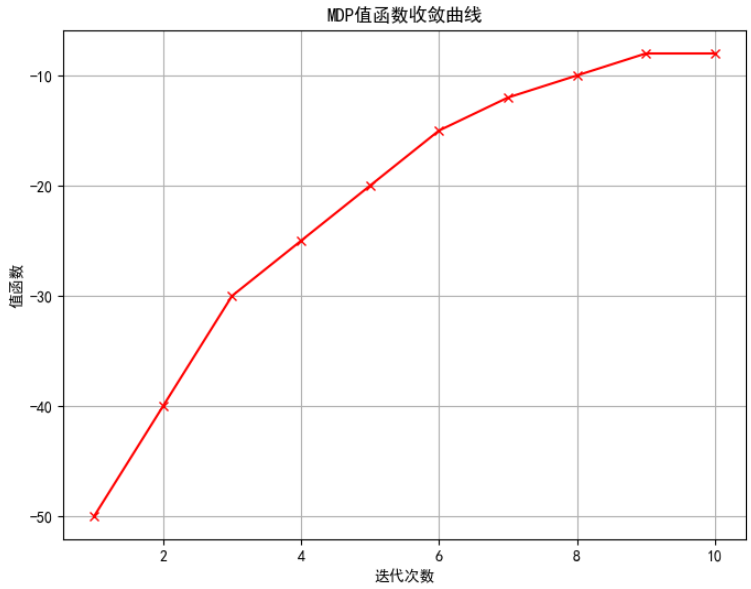
图表 5 不同决策组合下的总成本

图 6 展示了在不同决策组合下的总成本，可以更直观地反映出最优决策依据。

5.3.4 问题 3 灵敏度分析：



图表 6 次品率与净利润的关系



图表 7 MDP 值函数收敛曲线

5.4 问题 4 模型建立与求解

问题分析中已经给出了初步分析，并给出了初步的判断，根据问题四的要求可以将问题四拆分为两个部分来分别解决。第一部分分析针对问题二，第二分分析针对问题三。

✧ 问题 4 模型建立与求解的假设

- 信度统一为 95%
- 假设问题 4 涉及的模型对信度不敏感

由于涉及到了假设检验模型，我们对置信度做出了沿袭问题 1 的统一的假设，即信度为 95%，且对信度不敏感。

5.4.1 问题 4 第一部分

5.5 问题 4 第一部分的模型建立与求解

首先我们需要确定我们需要用到的模型仍然是二项分布假设检验模型以及动态规划模型，用问题 1 建立的**二项分布假设检验模型对零配件 1、零配件 2 和成品的次品率**，再将新的次品率参数嵌入问题 2 动态规划模型。

5.5.1 问题 4 第一部分的模型建立

■ **零配件 1、零配件 2 和成品次品率的求取模型**

我们需要中间变量样本量 n、标称值p_0两个中间变量来获得新的次品率，故我们需要建立问题 1 中所提到的二项分布模型来帮助我们将中间变量转换为新的次品率。

假设检验结合样本次品率的计算和置信区间的设定，设计合理的抽样检测方案。我们同样沿袭问题 1 的假设将该**二项分布的问题转化为使用正态分布来进行近似化处理**。

✧ 符号引入

$\boldsymbol{P_{0_{ij}}}$即次品率的标称值,其中 i=1,2,3,4,5,6，j=1,2,3,其中 i 分别代表 6 种情况，j 从 1~3 分别代表零配件 1、零配件 2 和成品，ij 组合形成对应的初始标称值。

$\boldsymbol{\hat{p}}$为我们根据问题 1 的二项分布假设检验法抽样求得的次品率

原假设(H_0): $\boldsymbol{\hat{p} \leq P_{0_{ij}}}$，表示零配件次品率小于标称值

备择假设(H_1) $\boldsymbol{\hat{p} > P_{0_{ij}}}$，表示零配件次品率大于标称值

目标是通过抽样检测来检验是否可以拒绝H_0

✧ 两种信度情况：

➢ 第一种情况：

在 **95%的信度**下，认定 P >P_0则拒收。即控制第一类错误（拒收合格品的概率）为 5%

➢ 第二种情况：

在 **90%的信度**下，认定 P≤P_0则接收。即控制第二类错误（接收不合格品的概率）为 10%

✧ 建立二项分布假设检验抽样模型：

因为仅有次品和合格品的结果，故确定本题为二项分布的问题，建立二项分布的数学模型，而当样本量比较大时，采取正态分布近似化的处理方式，以达到简便计算的效果。

➢ 二项分布公式建立

$$P(k;n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

➢ 当样本容量较大时，采用正态分布来近似处理：

$$\hat{p} \sim N(p_0, \frac{p_0(1-p_0)}{n})$$

注意：k 为次品数，$\hat{p}$为样本中的次品率

✧ 标称值$\boldsymbol{p_0}$的合理假设策略——贝叶斯推理

不同于问题 1 题目直接给出一个确定的标称值，在问题 4 中想要使用二项分布假设检验模型，我们需要自行假设一个合理的标称值$\boldsymbol{p_0}$假设。

我们基于**贝叶斯推理**以及已有数据来假设标称值，这里我们以情况 1 时零配件 1 的次品率 0.1 来作为历史数据来展示我们如何来假设标称值，并推广到每种情况时的每种零配件的次品率标称值假设。

➢ 假设标称值$\boldsymbol{p_{0_{11}}}$

我们根据问题 2 提供的表 1 中情况 1、零配件 1 的次品率假设标称值$p_{0_{11}}$为 0.1，作为历史标称值数据数据。

- **贝叶斯推理更新规则确定**

$$\boldsymbol{\alpha' = \alpha + k}$$
$$\boldsymbol{\beta' = \beta + (n - k)}$$

- **修正后的标称值估计：**

$$\boldsymbol{\hat{\theta} = \frac{\alpha'}{\alpha' + \beta'}}$$

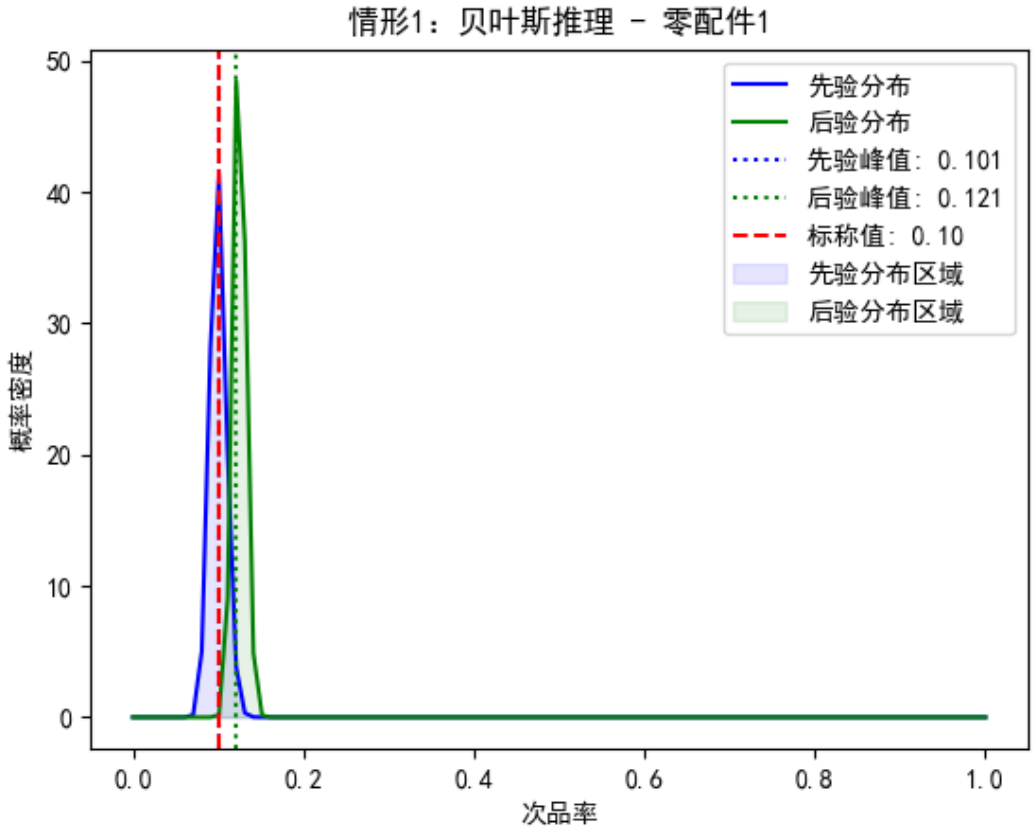
即

$$\widehat{\boldsymbol{\theta}} = \frac{\boldsymbol{\alpha} + \boldsymbol{k}}{\boldsymbol{\alpha} + \boldsymbol{\beta} + \boldsymbol{k}}$$

在这里$\hat{\theta}$是修正后的标称值，用于更新对次品率的合理估计，k 是观测到的次品数，n 为样本的总数。

基于原始标称值（来自表 1），$p_{0_{11}}$为 0.1，我们得到贝叶斯推理处理后的后验分布为 0.121。下图是贝叶斯推理的可视化分析。

图表 8 零配件 1 在情形 1 时的标称值修正过程



➢ 推广至每种情况的零配件以及成品的标称值

表格 6 step1 对标称值的修正值表

贝叶斯修正值 / 情形	零配件 1 标称值	零配件 2 标称值	成品标称值
1	12.50%	13.00%	10.00%
2	20.00%	20.50%	19.50%
3	10.00%	10.50%	9.50%
4	21.00%	21.50%	20.00%
5	14.00%	19.50%	13.50%
6	5.00%	5.50%	4.50%

✧ 抽样检测所得次品率求取

➢ 次品率变量$\hat{p}$公式建立

首先我们确定了检验统计量 Z 的公式

$$Z=\frac{\hat{p}-p_{0_{ij}}}{\sqrt{\frac{p_{0_{ij}}\left(1-p_{0_{ij}}\right)}{n}}}$$

由于 Z 是标准值可以通过查表得出，未知变量仅为抽样得出的零配件次品率$\hat{p}$，因此我们可以将实际抽样的次品率变量$\hat{p}$独立出来，方便求解。

$$\hat{p}=p_0+Z*\sqrt{\frac{p_{0_{ij}}\left(1-p_{0_{ij}}\right)}{n}}$$

- 抽样检测得次品率$\hat{p}$的求取

已知两种信度情况下的样本量、检验统计量的值和次品率的标称值，即可以利用次品率变量$\hat{p}$公式求解出对应情况的下的次品率$\hat{p}$

表格 7 抽样检测求得的次品率

情况	零配件 1 的次品率	零配件 2 的次品率	成品次品率
情形 1	12.5%	13.0%	11.0%
情形 2	22.5%	22.0%	21.5%
情形 3	10.0%	10.5%	9.5%
情形 4	20.0%	19.5%	19.0%
情形 5	14.0%	18.5%	13.0%
情形 6	5.0%	5.5%	4.5%

- **重新获得次品率后问题 2 最佳决策模型的复现**

在重新假设标称值$p_{0_{ij}}$和求取对应次品率后，我们需要仅需要对问题 2 中最佳决策模型（即动态规划模型）的次品率替换为抽样检测得到的新次品率，重复问题 2 中模型的步骤，即可复现出每种情况的最佳决策。

表格 8 最佳决策的记录表

情形	零配件 1 是否检测	零配件 2 是否检测	成品检测	是否拆解	总成本	总收益	总利润
1	是	是	否	是	6271.60	49526.40	43254.80
2	是	是	否	是	7371.60	43926.40	36554.80
3	是	是	否	是	8860.50	49823.20	40962.70
4	是	是	否	是	9728.00	43635.20	33907.20
5	是	是	否	是	11292.00	47443.20	36151.20
6	是	是	否	否	7287.80	52875.20	45587.40

5.5.2 问题 4 第一部分的模型求解

Step1 标称值的合理假设。首先我们根据原问题 2 所提供的表 1 中的每个零配件、成品的次品率作为我们的初步的标称值，然后再根据建立的**贝叶斯推理模型**(Bayesian

inference)对初始标称值进行处理根据后验分布的值得出修正后的标称值，这也就是我们后续在二项分布假设检验模型中所需要用的合理假设的标称值。

Step2 抽样检测得到新的次品率。由于在 step1 我们已经得到了通过贝叶斯推理模型求得的修正后的标称值，故我们可以利用该标称值代入二项分布假设检验模型抽样检测得出我们需要的新次品率。

Step3 次品率替换。这一步我需要做的就是将利用问题 1 中所使用的二项分布假设检验模型所求得的新次品率对应替换到原先问题 2 的表 1 中作为原始数据提供给下一步复现问题 2 的动态规划模型即可。

Step4 问题 2 动态规划模型的复现。因为在问题 2 中已经求解出了每种情况的最佳策略，故我们只需要将新的次品率数据替换原有次品率的数据，然后运行原有代码即可求得利用二项分布假设检验模型、贝叶斯推理所抽样检测得到新次品率所求得的新的最佳决策。

5.5.3 问题 4 第一部分的结果分析

■ **抽样检测结果——标称值的合理假设及新次品率的求取**

根据我们的贝叶斯推理模型，我们将问题 2 表 1 中所有的原始次品率转化为了修正后的标称值，然后以此为基础，重新建立问题 1 的二项分布假设检验模型，求解得到我们问题 2 动态规划模型需要的新的次品率并将他重新嵌入问题 2 表 1 中，完成了抽样检测求取次品率的任务。结果如下。

表格 9 step1 对标称值的修正值表

贝叶斯修正值 / 情形	零配件 1 标称值	零配件 2 标称值	成品标称值
1	12.50%	13.00%	10.00%
2	20.00%	20.50%	19.50%
3	10.00%	10.50%	9.50%
4	21.00%	21.50%	20.00%
5	14.00%	19.50%	13.50%
6	5.00%	5.50%	4.50%

表格 10 利用修正标称值求解得到的新次品率

抽样检测次品率 / 情形	零配件 1 次品率	零配件 2 次品率	成品次品率
1	14.22%	14.75%	11.56%
2	22.08%	22.60%	21.56%
3	11.56%	12.09%	11.03%
4	23.12%	23.64%	22.08%
5	15.08%	21.56%	15.28%
6	6.13%	6.69%	5.58%

■ **动态规划模型复现结果**

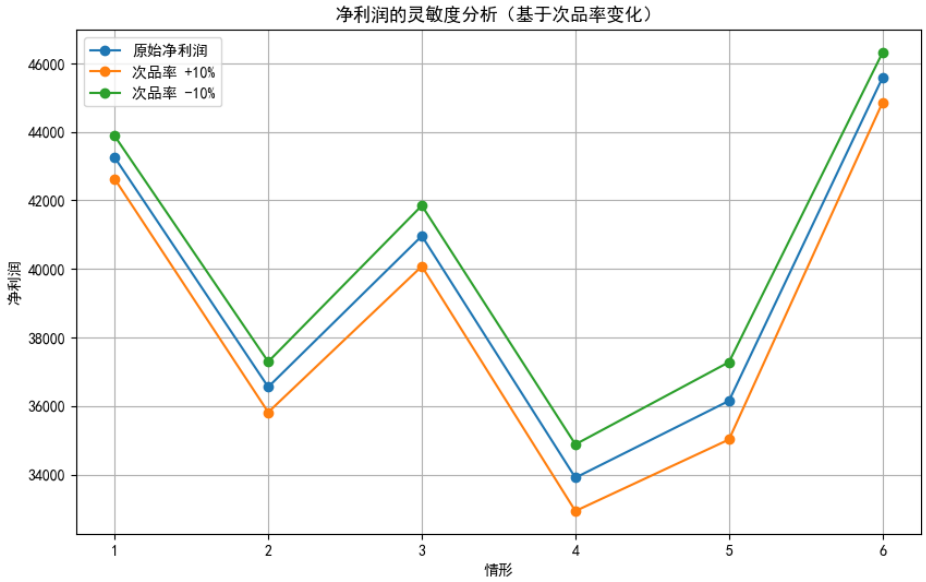
利用抽样检测的方法求取的次品率，根据问题 2 的模型和求解步骤即复现出问题 2 在抽样检测得到的次品率条件下的每种情形的最佳决策。

表格 11 最佳决策的记录表

情形	零配件 1 检测	零配件 2 检测	成品检测	是否拆解	总成本	总收益	总利润
1	是	是	否	是	6271.60	49526.40	43254.80
2	是	是	否	是	7371.60	43926.40	36554.80

3	是	是	否	是	8860.50	49823.20	40962.70
4	是	是	否	是	9728.00	43635.20	33907.20
5	是	是	否	是	11292.00	47443.20	36151.20
6	是	是	否	否	7287.80	52875.20	45587.40

5.5.4 问题 4 第一部分灵敏度分析：



图表 9 基于次品率变化的净利润的灵敏度分析

通过这个灵敏度分析，我们对零配件 1、零配件 2 和成品的次品率进行了±10%的调整，观察其对总成本和净利润的影响。这个过程正是对问题四第一部分中的重新计算次品率后的分析。

5.5.5 问题 4 第二部分的模型建立与求解

思路基本和第一部分相同，先利用现有原始数据利用**二项分布假设检验模型**抽样检测得到新的次品率，然后基于问题 3 的**马尔可夫决策过程模型**和新的次品率，重新复现问题 3 的结果。

首先我们需要确定我们需要用到的模型仍然是二项分布假设检验模型以及马尔可夫过程决策模型，用问题 1 建立的**二项分布假设检验模型**对问题三中的零配件 1-8，半成品 1-3 和成品的次品率，再将新的次品率参数嵌入问题三**马尔可夫过程模型**。

■ **问题四第二部分的模型建立：**

✧ 零配件 1、零配件 2、半成品和成品次品率的求取模型

基本和第一部分的求取模型相同，都是结合了**贝叶斯推理**（求取合理假设的标称值）和**二项分布假设检验模型**（反求次品率），来求取新次品率，如果上文已经认真阅读，此部分可以先略过。

✧ 符号引入

$\boldsymbol{P_{0ij}}$即次品率的标称值,其中当 i=1，2，3 当 i=1 时，j=1,2,3,4,5,6,7,8,当 i=2 时，j=1,2,3,当 i=3 时，j=1。分别代表零配件 1-8，半成品 1-3 及成品。

$\widehat{\boldsymbol{p}}$为我们根据问题 1 的二项分布假设检验法抽样求得的次品率

原假设(H_0): $\widehat{\boldsymbol{p}} \leq \boldsymbol{P_{0_{ij}}}$，表示零配件次品率小于标称值

备择假设(H_1) $\widehat{\boldsymbol{p}} > \boldsymbol{P_{0_{ij}}}$，表示零配件次品率大于标称值

目标是通过抽样检测来检验是否可以拒绝H_0

✧ 两种信度情况：

➢ 第一种情况：

在 **95%的信度**下，认定 P >P_0则拒收。即控制第一类错误（拒收合格品的概率）为 5%

➢ 第二种情况：

在 **90%的信度**下，认定 P≤P_0则接收。即控制第二类错误（接收不合格品的概率）为 10%

✧ 建立二项分布假设检验抽样模型：

因为仅有次品和合格品的结果，故确定本题为二项分布的问题，建立二项分布的数学模型，而当样本量比较大时，采取正态分布近似化的处理方式，以达到简便计算的效果。

➢ 二项分布公式建立

$$\boldsymbol{P(k;n,p) = \binom{n}{k} p^k (1-p)^{n-k}}$$

➢ 当样本容量较大时，采用正态分布来近似处理：

$$\boldsymbol{\hat{p} \sim N(p_0, \frac{p_0(1-p_0)}{n})}$$

其中，k 为次品数，p̂为样本中的次品率

✧ 标称值$\boldsymbol{p_0}$的合理假设策略——贝叶斯推理

不同于问题 1 题目直接给出一个确定的标称值，在问题 4 中想要使用二项分布假设检验模型，我们需要自行假设一个合理的标称值$\boldsymbol{p_0}$假设。

我们基于**贝叶斯推理**以及已有数据来假设标称值，这里我们以情况 1 时零配件 1 的次品率 0.1 来作为历史数据来展示我们如何来假设标称值，并推广到每种情况时的每种零配件的次品率标称值假设。

➢ 假设标称值$\boldsymbol{p_{0_{11}}}$

我们根据问题 3 提供的表 2 中提供的每个零部件以及半成品的称标次品率$p_{0_{11}}$为 0.1，作为历史标称值数据数据。

- **贝叶斯推理更新规则确定**

$$\boldsymbol{\alpha' = \alpha + k}$$

$$\boldsymbol{\beta' = \beta + (n - k)}$$

- **修正后的标称值估计：**

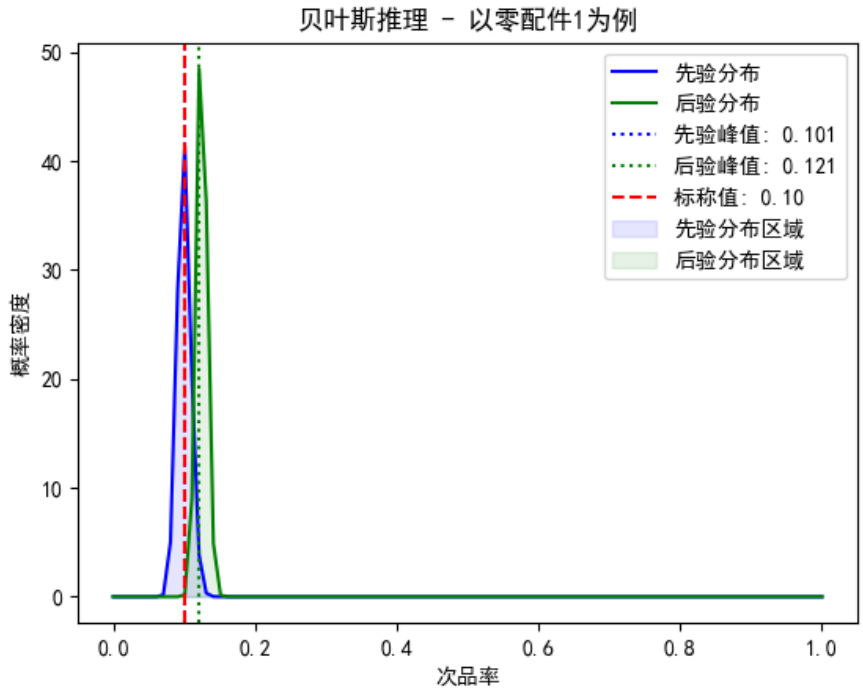
$$\boldsymbol{\hat{\theta} = \frac{\alpha'}{\alpha' + \beta'}}$$

即

$$\hat{\theta} = \frac{\alpha + k}{\alpha + \beta + k}$$

在这里$\hat{\theta}$是修正后的标称值，用于更新对次品率的合理估计，k 是观测到的次品数，n 为样本的总数。

基于原始标称值（来自表 2），$p_{0_{11}}$为 0.1，我们得到贝叶斯推理处理后的后验分布为 0.121。下图是贝叶斯推理的可视化分析。（引用 1.1.1 中的图表）



图表 10 零配件 1 标称值修正过程

表格 12 修正后的合理假设标称值

组件	贝叶斯修正后的新标称值
零配件 1	0.1020
零配件 2	0.1090
零配件 3	0.0900
零配件 4	0.0850
零配件 5	0.1160
零配件 6	0.1140
零配件 7	0.0955
零配件 8	0.1235
半成品 1	0.0970
半成品 2	0.0840
半成品 3	0.0825
成品	0.0750

✧ 抽样检测所得次品率求取

➢ 次品率变量$\hat{p}$公式建立

首先我们确定了检验统计量 Z 的公式

$$Z=\frac{\hat{p}-p_{0_{ij}}}{\sqrt{\frac{p_{0_{ij}}\left(1-p_{0_{ij}}\right)}{n}}}$$

由于 Z 是标准值可以通过查表得出，未知变量仅为抽样得出的零配件次品率$\hat{p}$，因此我们可以将实际抽样的次品率变量$\hat{p}$独立出来，方便求解。

$$\hat{p}=p_0+Z*\sqrt{\frac{p_{0_{ij}}\left(1-p_{0_{ij}}\right)}{n}}$$

- 抽样检测得次品率$\hat{p}$的求取

已知两种信度情况下的样本量、检验统计量的值和次品率的标称值，即可以利用次品率变量$\hat{p}$公式求解出对应情况的下的次品率$\hat{p}$

表格 13 抽样检测求得的次品率

组件	抽样检测次品率
零配件 1	0.1187
零配件 2	0.1242
零配件 3	0.1059
零配件 4	0.0985
零配件 5	0.1227
零配件 6	0.1315
零配件 7	0.1208
零配件 8	0.1506
半成品 1	0.1224
半成品 2	0.0974
半成品 3	0.0958
成品	0.0877

■ 重新获得次品率后问题 3 最佳决策模型的复现

在重新假设标称值$p_{0_{ij}}$和求取对应次品率后，我们需要仅需要对问题 3 中最佳决策模型（即马尔可夫过程模型）的次品率替换为抽样检测得到的新次品率，重复问题 3 中模型的步骤，即可复现出每种情况的最佳决策。

表格 14 最佳决策模型复现后的基础数据表

抽样对象	次品率	购买单价	检测成本		拆卸费用
零配件 1	12.20%	2	1		
零配件 2	13.48%	8	1		
零配件 3	11.93%	12	2		
零配件 4	13.00%	2	1		
零配件 5	12.36%	8	1		
零配件 6	11.93%	12	2		
零配件 7	11.56%	8	1		
零配件 8	11.13%	12	2		
半成品 1	10.76%	8	4		6
半成品 2	13.53%	8	4		6
半成品 3	13.96%	8	4		6
成品	11.51%	8	6		10

表 11 的补充为：成品的市场售价为 200，调换损失为 40

■ **问题 3 马尔可夫过程模型复现结果**

利用抽样检测的方法求取的次品率，根据问题 3 的模型和求解步骤即复现出问题 3 在抽样检测得到的次品率条件下的每种情形的最佳决策：

得出的最佳决策方案为 8 个零配件、3 给半成品**均做检测**，但是**均不拆解**，成品**不需要检测但是需要拆解**。

该决策方案的结果为：**总成本: 30826.80, 总收益: 176980.00, 净利润: 146153.20（假设有 N=1000 个成品）**

5.5.6 问题 4 第二部分马可夫过程模型灵敏度分析：

表格 15 零配件和成品的净利润灵敏度分析

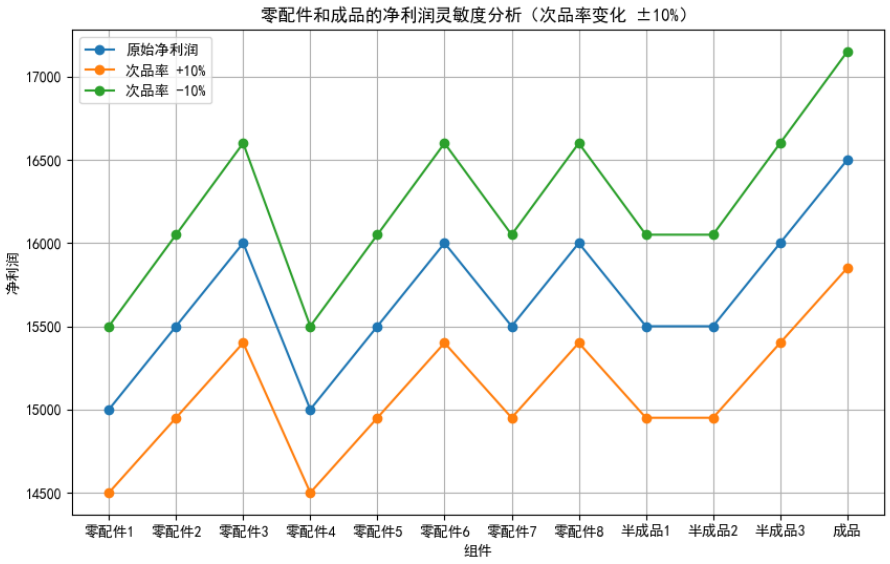


图 12 展示了零配件、半成品和成品的次品率在±10%变化时对净利润的灵敏度分析。可以观察到，次品率的变化对各个组件的净利润影响不同，这表明不同组件的次品率对总体利润的贡献存在差异。

六、 模型的评价与推广

6.1 模型的评价

■ **模型的优点**

- ✧ **全面**。问题 1 中使用的**二项分布假设检验模型**很好的契合题设要求，而我们对其进行了分类讨论，将其分为大样本和小样本的情况，分别选择了**正态分布假设检验模型**以及**$\bar{X}-R$控制图**模型来全面有效的处理各种样本情况。
- ✧ **丰富且契合**。我们选取了**二项分布假设检验模型、正态分布假设检验模型、$\bar{X}-R$控制图、动态规划模型、马尔可夫过程决策模型作为我们的基础模型**。

其中有 1 个常规的假设检验模型，1 个$\bar{X}-R$**控制图**模型作为一个创新点加入，完善了假设检验模型的搭建；有 2 个针对于多过程决策优化的模型，分别针对问题 2 和问题 3 进行求解，执行了具体问题具体分析的方法论。

- ✧ **合理。**主要体现在**贝叶斯推理模型**，因为问题 4 并没有提供标称值，为了搭建假设检验模型，我们必须自行假设一个合理的标称值，贝叶斯推理模型就帮助我们假设了合理的标称值以使用假设检验模型，在假设上是科学的。

■ **模型的缺点**

- ✧ 部分简化。例如，问题 4 中我们为了简化模型，对模型信度和模型对信度的灵敏度进行了固定假设，方便后续计算，实际上信度也是一个会影响结果的指标，但是我们在实际求解过程中没有发现有很大的影响，所以模型有部分的简化。

6.2 模型的改进

- ✧ 可以加入更加丰富且契合的模型来解决生产过程中的最优决策，例如整数规划等优化模型，使模型更加全面，结果更加有效。
- ✧ 加入对不同信度下的模型建立与求解，并对不同信度条件下进行灵敏度检测，可能会得到不同的结果。

6.3 模型的推广

- ✧ 由于我们的模型是基于二项分布的假设检验模型所搭建的，所以可以向任何存在二项分布的产业的抽样检测推广以降低成本。
- ✧ 其次我们也有 2 个模型用于搭建多过程下的总体生产最优决策，可以推广到大部分有多过程的生产中，以达成全局最优。

参考文献

[1] 杨旭,马玉林,杨晓慧.基于小批量生产的统计质量控制[J].计算机集成制造系统-CIMS,2001,(12)62-64.DOI10.13196j.cims.2001.12.63.yangx.013.

[2] Montgomery, Douglas C. (2020). Introduction to Statistical Quality Control 8th Edition

附录

附录 1

基于 python 的问题 1 求解代码

```
mport math
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams

# 设定参数
p0 = 0.10   # 标称次品率
Z_alpha = 1.96   # 95% 信度对应的 Z 值
Z_beta = 1.645   # 90% 信度对应的 Z 值

# 定义样本量计算的函数
def calculate_sample_size(Z_value, p0, E):
    """
    计算给定误差范围内的样本量
    Z_value: Z 值（根据显著性水平）
    p0: 标称次品率
    E: 允许误差范围
    """
    return math.ceil((Z_value ** 2 * p0 * (1 - p0)) / E ** 2)

# 定义不同的允许误差范围
error_rates = [0.01, 0.02, 0.03, 0.04, 0.05]

# 计算不同误差范围下的样本量
sample_sizes_95 = [calculate_sample_size(Z_alpha, p0, E) for E in error_rates]
sample_sizes_90 = [calculate_sample_size(Z_beta, p0, E) for E in error_rates]

# 创建 DataFrame 展示结果
df_sample_sizes = pd.DataFrame({
    '允许误差': error_rates,
    '95% 信度下样本量': sample_sizes_95,
    '90% 信度下样本量': sample_sizes_90
})

# 打印结果
print(df_sample_sizes)
```

```
### 绘制灵敏度分析图
plt.figure(figsize=(10, 6))

# 绘制 95% 信度下样本量的灵敏度分析
plt.plot(error_rates, sample_sizes_95, marker='o', label='95% 信度下样本量')

# 绘制 90% 信度下样本量的灵敏度分析
plt.plot(error_rates, sample_sizes_90, marker='s', label='90% 信度下样本量')

# 设置中文字体
rcParams['font.sans-serif'] = ['SimHei']  # 用于显示中文标签
rcParams['axes.unicode_minus'] = False  # 用于正常显示负号

# 添加图例和标签
plt.title('允许误差率对样本量的灵敏度分析')
plt.xlabel('允许误差率')
plt.ylabel('样本量')
plt.legend()
plt.grid(True)

# 显示图表
plt.show()
```

附录 2

基于 python 的问题 1 求解代码

```
import numpy as np
import pandas as pd

# 简化的 A2 值查找表（样本量 n 对应的 A2 值）
a2_table = {
    2: 1.880,
    3: 1.023,
    4: 0.729,
    5: 0.577,
    6: 0.483,
    7: 0.419,
    8: 0.373,
    9: 0.337,
    10: 0.308,
    11: 0.285,
    12: 0.266,
    13: 0.249,
    14: 0.235,
    15: 0.2230,
    16: 0.2120,
    17: 0.2030,
    18: 0.1940,
```

```
    19: 0.1870,
    20: 0.1800,
    21: 0.1730,
    22: 0.1670,
    23: 0.1620,
    24: 0.1570,
    25: 0.1530,
}

def get_a2_value(n_over_m):
    """根据 n/m 的近似值查找 A2 值"""
    rounded_value = round(n_over_m)
    return a2_table.get(rounded_value, "A2 value not found for this rounded value")

def generate_valid_n_m():
    """生成满足 n/m <= 25 且 m < 98 的有效 m 和 n 值"""
    while True:
        m = np.random.randint(1, 98)  # 生成 1 到 97 之间的随机整数
        n = np.random.randint(1, 25 * m + 1)  # 生成 1 到 25*m 之间的随机整数
        if m < 98 and n / m <= 25:
            return m, n

def generate_data(m, n, mean=0.1, fluctuation=0.01):
    """生成 m 组，每组 n 个样本的次品率数据"""
    print(f"Generating data with m = {m}, n = {n}")  # 打印 m 和 n 的值
    n_over_m = n / m
    A2 = get_a2_value(n_over_m)
    if isinstance(A2, str):
        raise ValueError(A2)
    print(f"A2 value for n/m rounded to {round(n_over_m)} is {A2}")  # 打印 A2 值
    data = []
    for i in range(m):
        samples = np.random.uniform(mean - fluctuation, mean + fluctuation, n)
        print(f"Group {i+1} samples: {samples}")  # 打印每组的样本数据
        data.append(samples)
    return np.array(data), A2

# 计算每组次品率均值
def calculate_group_means(data):
    return np.mean(data, axis=1)

# 计算每组次品率的极差
def calculate_range(data):
    return np.ptp(data, axis=1)  # ptp = peak-to-peak (最大值 - 最小值)

# 计算控制图统计量
def calculate_control_limits(X_bar, R_bar, A2):
    UCL = X_bar + A2 * R_bar
```

```
    LCL = X_bar - A2 * R_bar
    return UCL, LCL

# 判定是否接受零配件
def check_acceptance(group_means, UCL, LCL):
    results = []
    for mean in group_means:
        if mean > UCL:
            results.append("Reject")
        elif mean < LCL:
            results.append("Reject")
        else:
            results.append("Accept")
    return results

# 穷举不同的 m 和 n 值，生成结果表格
def generate_exhaustive_results():
    results = []
    for _ in range(100):  # 生成 100 组随机的 m 和 n
        m, n = generate_valid_n_m()
        try:
            # 生成数据
            data, A2 = generate_data(m, n)

            # 计算每组次品率均值和极差
            group_means = calculate_group_means(data)
            ranges = calculate_range(data)

            # 计算总体均值和极差均值
            X_bar = np.mean(group_means)
            R_bar = np.mean(ranges)

            # 计算控制限
            UCL, LCL = calculate_control_limits(X_bar, R_bar, A2)

            # 判断每组是否接受
            results_status = check_acceptance(group_means, UCL, LCL)

            # 收集结果
            for i, status in enumerate(results_status, start=1):
                results.append({
                    'm': m,
                    'n': n,
                    'Group': i,
                    'Mean': group_means[i-1],
                    'Range': ranges[i-1],
                    'UCL': UCL,
                    'LCL': LCL,
                    'Status': status
```

```
                    })
            except ValueError as e:
                print(f"Error for m = {m}, n = {n}: {e}")

    # 创建 DataFrame
    df = pd.DataFrame(results)
    return df

# 生成穷举结果并输出为表格
df_results = generate_exhaustive_results()
df_results.to_csv('control_chart_results.csv', index=False)
print("Results have been written to 'control_chart_results.csv'")
```

附录 3

基于 python 的问题 2 求解代码

```
# 定义 6 种情形数据
cases = [
    {'defect_rate_1': 0.1, 'defect_rate_2': 0.1, 'check_cost_1': 2, 'check_cost_2': 3, 'assemble_cost': 6,
     'check_cost': 3, 'market_price': 56, 'replace_cost': 6, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.2, 'defect_rate_2': 0.2, 'check_cost_1': 2, 'check_cost_2': 3, 'assemble_cost': 6,
     'check_cost': 3, 'market_price': 56, 'replace_cost': 6, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.1, 'defect_rate_2': 0.1, 'check_cost_1': 2, 'check_cost_2': 3, 'assemble_cost': 6,
     'check_cost': 3, 'market_price': 56, 'replace_cost': 30, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.2, 'defect_rate_2': 0.2, 'check_cost_1': 1, 'check_cost_2': 1, 'assemble_cost': 6,
     'check_cost': 2, 'market_price': 56, 'replace_cost': 30, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.1, 'defect_rate_2': 0.2, 'check_cost_1': 8, 'check_cost_2': 1, 'assemble_cost': 6,
     'check_cost': 2, 'market_price': 56, 'replace_cost': 10, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.05, 'defect_rate_2': 0.05, 'check_cost_1': 2, 'check_cost_2': 3, 'assemble_cost': 6,
     'check_cost': 3, 'market_price': 56, 'replace_cost': 10, 'dismantle_cost': 40, 'N_total': 1000}
]
# 调整后的计算不合格品数量的逻辑
def calculate_defective_products(case, detect_part_1, detect_part_2):
    N = case["N_total"]
    P1 = case["defect_rate_1"]
    P2 = case["defect_rate_2"]
```

```
    if detect_part_1 == 1 and detect_part_2 == 1:
        return N * 0.1  # 两个零配件都检测，不合格率固定为 10%
    elif detect_part_1 == 1:
        return N * (0.1 + P2)  # 仅检测零配件 1
    elif detect_part_2 == 1:
        return N * (0.1 + P1)  # 仅检测零配件 2
    else:
        delta_Pd = P1 + P2 - P1 * P2  # 不检测时次品率提高值
        return N * (0.1 + delta_Pd)  # 两个零配件都不检测

# 计算退回的不合格品数量
def calculate_return_defective_products(case, detect_part_1, detect_part_2):
    N = case["N_total"]
    P1 = case["defect_rate_1"]
    P2 = case["defect_rate_2"]

    # 根据检测情况计算不合格品数量
    if detect_part_1 == 1 and detect_part_2 == 1:
        return N * 0.1  # 两个零配件都检测
    elif detect_part_1 == 1:
        return N * (0.1 + P2)  # 仅检测零配件 1
    elif detect_part_2 == 1:
        return N * (0.1 + P1)  # 仅检测零配件 2
    else:
        delta_Pd = P1 + P2 - P1 * P2  # 两个零配件都不检测时的次品率
        return N * (0.1 + delta_Pd)

# 计算成本
def calculate_costs(case, detect_part_1, detect_part_2, detect_final_product, dismantle,
return_products=False):
    N = case["N_total"]

    # 根据文档给出的公式，计算不合格品数量
    defective_products = calculate_defective_products(case, detect_part_1, de-
tect_part_2)
    qualified_products = N - defective_products

    # 检测成本
    check_costs = (case["check_cost_1"] * detect_part_1 + case["check_cost_2"] * de-
tect_part_2 + case["check_cost"] * detect_final_product) * N

    # 替换成本（不检测时已包含期望替换成本）
    replace_costs = defective_products * case["replace_cost"]

    # 拆解成本（根据不合格品的拆解决策）
    dismantle_costs = dismantle * defective_products * case["dismantle_cost"]

    # 装配成本
```

```
    assemble_costs = case["assemble_cost"] * N

    # 总成本
    total_cost = check_costs + replace_costs + dismantle_costs + assemble_costs
    return total_cost, qualified_products

# 计算总收益
def calculate_revenue(case, qualified_products):
    return qualified_products * case["market_price"]

# 动态规划逻辑，包括退回的不合格品处理
def dynamic_programming(case, case_index, handle_returns=False):
    detect_part_1, detect_part_2, detect_final_product, dismantle = 0, 0, 0, 0
    max_net_profit = float('-inf')
    best_decision = (0, 0, 0, 0)

    for detect_part_1 in [0, 1]:
        for detect_part_2 in [0, 1]:
            for detect_final_product in [0, 1]:
                for dismantle in [0, 1]:
                    total_cost, qualified_products = calculate_costs(case, detect_part_1, detect_part_2, detect_final_product, dismantle)
                    total_revenue = calculate_revenue(case, qualified_products)
                    net_profit = total_revenue - total_cost

                    # 处理退回的不合格品（这里不会产生新的市场收益，也不产生新的成本）
                    if handle_returns:
                        print(f"情况 {case_index + 1}: 无需额外处理退回品，未产生新的成本和收益")

                    if net_profit > max_net_profit:
                        max_net_profit = net_profit
                        best_decision = (detect_part_1, detect_part_2, detect_final_product, dismantle)
                        print(f"情况 {case_index + 1} - 当前最佳决策: 零配件 1 检测={detect_part_1}, 零配件 2 检测={detect_part_2}, 成品检测={detect_final_product}, 拆解={dismantle}, 净利润={net_profit:.2f}")

    return max_net_profit, best_decision

# 主函数
def main():
    for case_index, case in enumerate(cases):
        max_net_profit, best_decision = dynamic_programming(case, case_index, handle_returns=True)
        initial_cost, qualified_products = calculate_costs(case, *best_decision)
        initial_revenue = calculate_revenue(case, qualified_products)
```

```
            # 退回处理阶段不再产生新成本
            print(f"情况 {case_index + 1}: 退回品已处理，未产生新的退回成本。")

            # 最终净利润 = 初始阶段的净利润（无新的退回处理成本）
            final_net_profit = initial_revenue - initial_cost

            print(f"情况 {case_index + 1} - 最终最佳决策: 零配件 1 检测={best_decision[0]}, 零配件 2 检测={best_decision[1]}, 成品检测={best_decision[2]}, 拆解={best_decision[3]}")
            print(f"初始总成本: {initial_cost:.2f}, 初始总收益: {initial_revenue:.2f}")
            print(f"最终净利润: {final_net_profit:.2f}")
            print("-" * 30)

# 执行程序
main()
```

附录 4

基于 python 的问题 3 求解代码

```
import numpy as np

# 定义状态和动作
states = ['合格', '次品']  # 状态: 合格或次品
actions = ['检测', '跳过检测', '拆解', '跳过拆解']  # 动作: 检测或跳过检测, 拆解或跳过拆解

# 定义奖励函数
def reward_function(state, action, part, case):
    if action == '检测':
        return -case[part]['检测成本']  # 检测成本
    elif action == '跳过检测':
        defective_loss = float(case[part]['次品率'].strip('%')) / 100 * case['成品']['调换损失']
        return -(case[part].get('购买单价', case[part].get('装配成本', 0)) + defective_loss)
    elif action == '拆解':
        return -case[part].get('拆解费用', 0)  # 如果没有'拆解费用'字段，默认值为0
    elif action == '跳过拆解':
        product_value = case[part].get('购买单价', case[part].get('装配成本', 0))
        return min(product_value, case[part].get('拆解费用', 0))  # 默认拆解费用为0
    return 0

# 计算净利润
```

```
def calculate_total_profit(case, detect_parts, dismantle_parts, detect_final_product, dismantle_final_product):
    N_total = 1000  # 成品数量

    # 计算零配件和半成品的联合次品率
    part_defective_rates = [float(case[part]['次品率'].strip('%')) / 100 for part in detect_parts]
    delta_Pd = 1 - np.prod([1 - P for P in part_defective_rates])  # 联合次品率公式

    # 计算半成品的次品率
    semifinished_parts = [part for part in case if part.startswith('半成品')]
    semifinished_defective_rates = [float(case[part]['次品率'].strip('%')) / 100 for part in semifinished_parts]
    delta_semifinished_Pd = 1 - np.prod([1 - P for P in semifinished_defective_rates])

    # 计算成品次品率，合并零配件和半成品的影响
    product_defect_rate = float(case['成品']['次品率'].strip('%')) / 100
    N_defective = N_total * (product_defect_rate + (0 if all(detect_parts.values()) else delta_Pd) + (0 if all([detect_parts[part] for part in semifinished_parts]) else delta_semifinished_Pd))

    N_qualified = N_total - N_defective

    # 检测成本
    check_cost = sum(case[part]['检测成本'] * N_total for part, detect in detect_parts.items() if detect)
    final_check_cost = case['成品']['检测成本'] * N_total if detect_final_product else 0

    # 调换成本
    replace_cost = N_defective * case['成品']['调换损失']

    # 拆解成本
    dismantle_cost = 0
    for part, dismantle in dismantle_parts.items():
        if dismantle:
            dismantle_cost += case[part].get('拆解费用', 0) * N_defective  # 使用get()避免 KeyError
        else:
            product_value = case[part].get('购买单价', case[part].get('装配成本', 0))
            dismantle_cost += N_defective * min(product_value, case[part].get('拆解费用', 0))  # 如果缺少'拆解费用'，则默认为 0

    # 拆解成品
    if dismantle_final_product:
        dismantle_cost += N_defective * case['成品']['拆解费用']
    else:
```

```
            product_value = sum(case[part].get('购买单价', case[part].get('装配成本', 0)) for part in detect_parts) + case['成品']['装配成本']
            dismantle_cost += N_defective * min(product_value, case['成品']['拆解费用'])

        # 总成本
        total_cost = check_cost + final_check_cost + replace_cost + dismantle_cost

        # 总收益
        total_revenue = N_qualified * case['成品']['市场售价']

        # 净利润
        net_profit = total_revenue - total_cost

        return total_cost, total_revenue, net_profit

def calculate_best_decisions(case):
        best_decision = None
        best_profit = float('-inf')

        print(f"\n 计算最佳检测与拆解决策:\n")

        # 零配件部分：只检测，不拆解
        parts = [part for part in case if part.startswith('零配件')]

        # 半成品部分：检测并考虑拆解
        semifinished_parts = [part for part in case if part.startswith('半成品')]

        # 遍历所有组合的检测和拆解策略
        for detect_parts_combination in range(1 << len(parts)):
            detect_parts = {parts[i]: (detect_parts_combination >> i) & 1 for i in range(len(parts))}

            # 保证所有零配件的拆解状态为 False (不拆解)
            dismantle_parts = {part: 0 for part in parts}

            for detect_semifinished_combination in range(1 << len(semifinished_parts)):
                detect_semifinished = {semifinished_parts[i]: (detect_semifinished_combination >> i) & 1 for i in range(len(semifinished_parts))}
                for dismantle_semifinished_combination in range(1 << len(semifinished_parts)):
                    dismantle_semifinished = {semifinished_parts[i]: (dismantle_semifinished_combination >> i) & 1 for i in range(len(semifinished_parts))}

                    # 合并零配件和半成品的检测和拆解策略
                    detect_combined = {**detect_parts, **detect_semifinished}
                    dismantle_combined = {**dismantle_parts, **dismantle_semifinished}
```

```
                    for detect_final_product in [True, False]:
                        for dismantle_final_product in [True, False]:
                            total_cost, total_revenue, net_profit = calculate_total_profit(
                                case,
                                detect_parts=detect_combined,
                                dismantle_parts=dismantle_combined,
                                detect_final_product=detect_final_product,
                                dismantle_final_product=dismantle_final_product
                            )

                            # 将 0 和 1 转换为 T 和 F，输出当前策略和计算结果
                            detect_combined_TF = {part: 'T' if val == 1 else 'F' for part, val in detect_combined.items()}
                            dismantle_combined_TF = {part: 'T' if val == 1 else 'F' for part, val in dismantle_combined.items()}
                            detect_final_product_TF = 'T' if detect_final_product else 'F'
                            dismantle_final_product_TF = 'T' if dismantle_final_product else 'F'

                            print(f"当前策略 - 零配件/半成品检测: {detect_combined_TF}, 零配件/半成品拆解: {dismantle_combined_TF}, "
                                  f"检测成品: {detect_final_product_TF}, 拆解成品: {dismantle_final_product_TF}")
                            print(f"总成本: {total_cost:.2f}, 总收益: {total_revenue:.2f}, 净利润: {net_profit:.2f}\n")

                            # 更新最佳决策
                            if net_profit > best_profit:
                                best_profit = net_profit
                                best_decision = (detect_combined, dismantle_combined, detect_final_product, dismantle_final_product,
                                                 total_cost, total_revenue, net_profit)

    # 输出最佳决策
    if best_decision:
        detect_combined, dismantle_combined, detect_final_product, dismantle_final_product, total_cost, total_revenue, best_profit = best_decision

        detect_combined_TF = {part: 'T' if val == 1 else 'F' for part, val in detect_combined.items()}
        dismantle_combined_TF = {part: 'T' if val == 1 else 'F' for part, val in dismantle_combined.items()}
        detect_final_product_TF = 'T' if detect_final_product else 'F'
        dismantle_final_product_TF = 'T' if dismantle_final_product else 'F'
```

```
            print(f"最佳决策 - 零配件/半成品检测: {detect_combined_TF}, 零配件/半成品拆解: {dismantle_combined_TF}, 检测成品: {detect_final_product_TF}, 拆解成品: {dismantle_final_product_TF}")
            print(f"总成本: {total_cost:.2f}, 总收益: {total_revenue:.2f}, 净利润: {best_profit:.2f}")
        else:
            print("未能找到有效的最佳决策。")

# 使用更新后的 case 数据
cases = {
    "零配件 1": {"次品率": "10%", "购买单价": 2, "检测成本": 1},
    "零配件 2": {"次品率": "10%", "购买单价": 8, "检测成本": 1},
    "零配件 3": {"次品率": "10%", "购买单价": 12, "检测成本": 2},
    "零配件 4": {"次品率": "10%", "购买单价": 2, "检测成本": 1},
    "零配件 5": {"次品率": "10%", "购买单价": 8, "检测成本": 1},
    "零配件 6": {"次品率": "10%", "购买单价": 12, "检测成本": 2},
    "零配件 7": {"次品率": "10%", "购买单价": 8, "检测成本": 1},
    "零配件 8": {"次品率": "10%", "购买单价": 12, "检测成本": 2},
    "半成品 1": {"次品率": "10%", "装配成本": 8, "检测成本": 4, "拆解费用": 6},
    "半成品 2": {"次品率": "10%", "装配成本": 8, "检测成本": 4, "拆解费用": 6},
    "半成品 3": {"次品率": "10%", "装配成本": 8, "检测成本": 4, "拆解费用": 6},
    "成品": {
        "次品率": "10%",
        "装配成本": 8,
        "检测成本": 6,
        "拆解费用": 10,
        "市场售价": 200,
        "调换损失": 40
    }
}

# 运行最佳决策计算
calculate_best_decisions(cases)
```

附录 5

基于 python 的问题 4 第一部分求解代码

```
# 定义 6 种情形数据
cases = [
    {'defect_rate_1': 0.1, 'defect_rate_2': 0.1, 'check_cost_1': 2, 'check_cost_2': 3, 'assemble_cost': 6,
     'check_cost': 3, 'market_price': 56, 'replace_cost': 6, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.2, 'defect_rate_2': 0.2, 'check_cost_1': 2, 'check_cost_2': 3, 'assemble_cost': 6,
     'check_cost': 3, 'market_price': 56, 'replace_cost': 6, 'dismantle_cost': 5, 'N_total': 1000},
```

```
    {'defect_rate_1': 0.1, 'defect_rate_2': 0.1, 'check_cost_1': 2, 'check_cost_2': 3, 'assemble_cost': 6,
     'check_cost': 3, 'market_price': 56, 'replace_cost': 30, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.2, 'defect_rate_2': 0.2, 'check_cost_1': 1, 'check_cost_2': 1, 'assemble_cost': 6,
     'check_cost': 2, 'market_price': 56, 'replace_cost': 30, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.1, 'defect_rate_2': 0.2, 'check_cost_1': 8, 'check_cost_2': 1, 'assemble_cost': 6,
     'check_cost': 2, 'market_price': 56, 'replace_cost': 10, 'dismantle_cost': 5, 'N_total': 1000},
    {'defect_rate_1': 0.05, 'defect_rate_2': 0.05, 'check_cost_1': 2, 'check_cost_2': 3, 'assemble_cost': 6,
     'check_cost': 3, 'market_price': 56, 'replace_cost': 10, 'dismantle_cost': 40, 'N_total': 1000}
]
# 调整后的计算不合格品数量的逻辑
def calculate_defective_products(case, detect_part_1, detect_part_2):
    N = case["N_total"]
    P1 = case["defect_rate_1"]
    P2 = case["defect_rate_2"]

    if detect_part_1 == 1 and detect_part_2 == 1:
        return N * 0.1  # 两个零配件都检测，不合格率固定为 10%
    elif detect_part_1 == 1:
        return N * (0.1 + P2)  # 仅检测零配件 1
    elif detect_part_2 == 1:
        return N * (0.1 + P1)  # 仅检测零配件 2
    else:
        delta_Pd = P1 + P2 - P1 * P2  # 不检测时次品率提高值
        return N * (0.1 + delta_Pd)  # 两个零配件都不检测

# 计算退回的不合格品数量
def calculate_return_defective_products(case, detect_part_1, detect_part_2):
    N = case["N_total"]
    P1 = case["defect_rate_1"]
    P2 = case["defect_rate_2"]

    # 根据检测情况计算不合格品数量
    if detect_part_1 == 1 and detect_part_2 == 1:
        return N * 0.1  # 两个零配件都检测
    elif detect_part_1 == 1:
        return N * (0.1 + P2)  # 仅检测零配件 1
    elif detect_part_2 == 1:
        return N * (0.1 + P1)  # 仅检测零配件 2
    else:
        delta_Pd = P1 + P2 - P1 * P2  # 两个零配件都不检测时的次品率
```

```
            return N * (0.1 + delta_Pd)

# 计算成本
def calculate_costs(case, detect_part_1, detect_part_2, detect_final_product, dismantle, return_products=False):
    N = case["N_total"]

    # 根据文档给出的公式，计算不合格品数量
    defective_products = calculate_defective_products(case, detect_part_1, detect_part_2)
    qualified_products = N - defective_products

    # 检测成本
    check_costs = (case["check_cost_1"] * detect_part_1 + case["check_cost_2"] * detect_part_2 + case["check_cost"] * detect_final_product) * N

    # 替换成本（不检测时已包含期望替换成本）
    replace_costs = defective_products * case["replace_cost"]

    # 拆解成本（根据不合格品的拆解决策）
    dismantle_costs = dismantle * defective_products * case["dismantle_cost"]

    # 装配成本
    assemble_costs = case["assemble_cost"] * N

    # 总成本
    total_cost = check_costs + replace_costs + dismantle_costs + assemble_costs
    return total_cost, qualified_products

# 计算总收益
def calculate_revenue(case, qualified_products):
    return qualified_products * case["market_price"]

# 动态规划逻辑，包括退回的不合格品处理
def dynamic_programming(case, case_index, handle_returns=False):
    detect_part_1, detect_part_2, detect_final_product, dismantle = 0, 0, 0, 0
    max_net_profit = float('-inf')
    best_decision = (0, 0, 0, 0)

    for detect_part_1 in [0, 1]:
        for detect_part_2 in [0, 1]:
            for detect_final_product in [0, 1]:
                for dismantle in [0, 1]:
                    total_cost, qualified_products = calculate_costs(case, detect_part_1, detect_part_2, detect_final_product, dismantle)
                    total_revenue = calculate_revenue(case, qualified_products)
                    net_profit = total_revenue - total_cost
```

```
                    # 处理退回的不合格品（这里不会产生新的市场收益，也不产生新的成本）
                    if handle_returns:
                        print(f"情况 {case_index + 1}: 无需额外处理退回品，未产生新的成本和收益")

                    if net_profit > max_net_profit:
                        max_net_profit = net_profit
                        best_decision = (detect_part_1, detect_part_2, detect_final_product, dismantle)
                        print(f"情况 {case_index + 1} - 当前最佳决策: 零配件 1 检测 ={detect_part_1}, 零配件 2 检测 ={detect_part_2}, 成品检测 ={detect_final_product}, 拆解={dismantle}, 净利润={net_profit:.2f}")

    return max_net_profit, best_decision

# 主函数
def main():
    for case_index, case in enumerate(cases):
        max_net_profit, best_decision = dynamic_programming(case, case_index, handle_returns=True)
        initial_cost, qualified_products = calculate_costs(case, *best_decision)
        initial_revenue = calculate_revenue(case, qualified_products)

        # 退回处理阶段不再产生新成本
        print(f"情况 {case_index + 1}: 退回品已处理，未产生新的退回成本。")

        # 最终净利润 = 初始阶段的净利润（无新的退回处理成本）
        final_net_profit = initial_revenue - initial_cost

        print(f"情况 {case_index + 1} - 最终最佳决策: 零配件 1 检测 ={best_decision[0]}, 零配件 2 检测={best_decision[1]}, 成品检测={best_decision[2]}, 拆解={best_decision[3]}")
        print(f"初始总成本: {initial_cost:.2f}, 初始总收益: {initial_revenue:.2f}")
        print(f"最终净利润: {final_net_profit:.2f}")
        print("-" * 30)

# 执行程序
main()
```

、

附录 6

基于 python 的问题 4 第二部分求解代码

```
import numpy as np

# 定义状态和动作
states = ['合格', '次品']  # 状态: 合格或次品
actions = ['检测', '跳过检测', '拆解', '跳过拆解']  # 动作: 检测或跳过检测, 拆解或跳过拆解

# 定义奖励函数
def reward_function(state, action, part, case):
    if action == '检测':
        return -case[part]['检测成本']  # 检测成本
    elif action == '跳过检测':
        defective_loss = float(case[part]['次品率'].strip('%')) / 100 * case['成品']['调换损失']
        return -(case[part].get('购买单价', case[part].get('装配成本', 0)) + defective_loss)
    elif action == '拆解':
        return -case[part].get('拆解费用', 0)  # 如果没有'拆解费用'字段，默认值为0
    elif action == '跳过拆解':
        product_value = case[part].get('购买单价', case[part].get('装配成本', 0))
        return min(product_value, case[part].get('拆解费用', 0))  # 默认拆解费用为0
    return 0

# 计算净利润
def calculate_total_profit(case, detect_parts, dismantle_parts, detect_final_product, dismantle_final_product):
    N_total = 1000  # 成品数量

    # 计算零配件和半成品的联合次品率
    part_defective_rates = [float(case[part]['次品率'].strip('%')) / 100 for part in detect_parts]
    delta_Pd = 1 - np.prod([1 - P for P in part_defective_rates])  # 联合次品率公式

    # 计算半成品的次品率
    semifinished_parts = [part for part in case if part.startswith('半成品')]
    semifinished_defective_rates = [float(case[part]['次品率'].strip('%')) / 100 for part in semifinished_parts]
    delta_semifinished_Pd = 1 - np.prod([1 - P for P in semifinished_defective_rates])

    # 计算成品次品率，合并零配件和半成品的影响
    product_defect_rate = float(case['成品']['次品率'].strip('%')) / 100
```

```
    N_defective = N_total * (product_defect_rate + (0 if all(detect_parts.values()) else delta_Pd) + (0 if all([detect_parts[part] for part in semifinished_parts]) else delta_semifinished_Pd))

    N_qualified = N_total - N_defective

    # 检测成本
    check_cost = sum(case[part]['检测成本'] * N_total for part, detect in detect_parts.items() if detect)
    final_check_cost = case['成品']['检测成本'] * N_total if detect_final_product else 0

    # 调换成本
    replace_cost = N_defective * case['成品']['调换损失']

    # 拆解成本
    dismantle_cost = 0
    for part, dismantle in dismantle_parts.items():
        if dismantle:
            dismantle_cost += case[part].get('拆解费用', 0) * N_defective  # 使用get()避免 KeyError
        else:
            product_value = case[part].get('购买单价', case[part].get('装配成本', 0))
            dismantle_cost += N_defective * min(product_value, case[part].get('拆解费用', 0))  # 如果缺少'拆解费用'，则默认为 0

    # 拆解成品
    if dismantle_final_product:
        dismantle_cost += N_defective * case['成品']['拆解费用']
    else:
        product_value = sum(case[part].get('购买单价', case[part].get('装配成本', 0)) for part in detect_parts) + case['成品']['装配成本']
        dismantle_cost += N_defective * min(product_value, case['成品']['拆解费用'])

    # 总成本
    total_cost = check_cost + final_check_cost + replace_cost + dismantle_cost

    # 总收益
    total_revenue = N_qualified * case['成品']['市场售价']

    # 净利润
    net_profit = total_revenue - total_cost

    return total_cost, total_revenue, net_profit

def calculate_best_decisions(case):
    best_decision = None
    best_profit = float('-inf')
```

```
    print(f"\n 计算最佳检测与拆解决策:\n")

    # 零配件部分：只检测，不拆解
    parts = [part for part in case if part.startswith('零配件')]

    # 半成品部分：检测并考虑拆解
    semifinished_parts = [part for part in case if part.startswith('半成品')]

    # 遍历所有组合的检测和拆解策略
    for detect_parts_combination in range(1 << len(parts)):
        detect_parts = {parts[i]: (detect_parts_combination >> i) & 1 for i in range(len(parts))}

        # 保证所有零配件的拆解状态为 False (不拆解)
        dismantle_parts = {part: 0 for part in parts}

        for detect_semifinished_combination in range(1 << len(semifinished_parts)):
            detect_semifinished = {semifinished_parts[i]: (detect_semifinished_combination >> i) & 1 for i in range(len(semifinished_parts))}
            for dismantle_semifinished_combination in range(1 << len(semifinished_parts)):
                dismantle_semifinished = {semifinished_parts[i]: (dismantle_semifinished_combination >> i) & 1 for i in range(len(semifinished_parts))}

                # 合并零配件和半成品的检测和拆解策略
                detect_combined = {**detect_parts, **detect_semifinished}
                dismantle_combined = {**dismantle_parts, **dismantle_semifinished}

                for detect_final_product in [True, False]:
                    for dismantle_final_product in [True, False]:
                        total_cost, total_revenue, net_profit = calculate_total_profit(
                            case,
                            detect_parts=detect_combined,
                            dismantle_parts=dismantle_combined,
                            detect_final_product=detect_final_product,
                            dismantle_final_product=dismantle_final_product
                        )

                        # 将 0 和 1 转换为 T 和 F，输出当前策略和计算结果
                        detect_combined_TF = {part: 'T' if val == 1 else 'F' for part, val in detect_combined.items()}
                        dismantle_combined_TF = {part: 'T' if val == 1 else 'F' for part, val in dismantle_combined.items()}
                        detect_final_product_TF = 'T' if detect_final_product else 'F'
```

```
                                dismantle_final_product_TF = 'T' if dismantle_final_product else 'F'

                                print(f"当前策略 - 零配件/半成品检测: {detect_combined_TF}, 零配件/半成品拆解: {dismantle_combined_TF}, "
                                      f"检测成品: {detect_final_product_TF}, 拆解成品: {dismantle_final_product_TF}")
                                print(f"总成本: {total_cost:.2f}, 总收益: {total_revenue:.2f}, 净利润: {net_profit:.2f}\n")

                                # 更新最佳决策
                                if net_profit > best_profit:
                                    best_profit = net_profit
                                    best_decision = (detect_combined, dismantle_combined, detect_final_product, dismantle_final_product,
                                                     total_cost, total_revenue, net_profit)

    # 输出最佳决策
    if best_decision:
        detect_combined, dismantle_combined, detect_final_product, dismantle_final_product, total_cost, total_revenue, best_profit = best_decision

        detect_combined_TF = {part: 'T' if val == 1 else 'F' for part, val in detect_combined.items()}
        dismantle_combined_TF = {part: 'T' if val == 1 else 'F' for part, val in dismantle_combined.items()}
        detect_final_product_TF = 'T' if detect_final_product else 'F'
        dismantle_final_product_TF = 'T' if dismantle_final_product else 'F'

        print(f"最佳决策 - 零配件/半成品检测: {detect_combined_TF}, 零配件/半成品拆解: {dismantle_combined_TF}, 检测成品: {detect_final_product_TF}, 拆解成品: {dismantle_final_product_TF}")
        print(f"总成本: {total_cost:.2f}, 总收益: {total_revenue:.2f}, 净利润: {best_profit:.2f}")
    else:
        print("未能找到有效的最佳决策。")

# 使用更新后的 case 数据
cases_updated = {
    "零配件 1": {"次品率": "12.20%", "购买单价": 2, "检测成本": 1},
    "零配件 2": {"次品率": "13.48%", "购买单价": 8, "检测成本": 1},
    "零配件 3": {"次品率": "11.93%", "购买单价": 12, "检测成本": 2},
    "零配件 4": {"次品率": "13.00%", "购买单价": 2, "检测成本": 1},
    "零配件 5": {"次品率": "12.36%", "购买单价": 8, "检测成本": 1},
    "零配件 6": {"次品率": "11.93%", "购买单价": 12, "检测成本": 2},
    "零配件 7": {"次品率": "11.56%", "购买单价": 8, "检测成本": 1},
```

```
    "零配件 8": {"次品率": "11.13%", "购买单价": 12, "检测成本": 2},
    "半成品 1": {"次品率": "10.76%", "装配成本": 8, "检测成本": 4, "拆解费用": 6},
    "半成品 2": {"次品率": "13.53%", "装配成本": 8, "检测成本": 4, "拆解费用": 6},
    "半成品 3": {"次品率": "13.96%", "装配成本": 8, "检测成本": 4, "拆解费用": 6},
    "成品": {
        "次品率": "11.51%",
        "装配成本": 8,
        "检测成本": 6,
        "拆解费用": 10,
        "市场售价": 200,
        "调换损失": 40
    }
}
# 运行最佳决策计算
calculate_best_decisions(cases_updated)
```