基于抽样检测与利润最大化的生产过程决策模型研究

摘要

本文针对生产过程中的决策问题，立足于收益最大化这一目标，建立数学模型。提出了蒙特卡洛模拟最优决策方法、成本在决策单元间的动态转移函数模型以及由贝叶斯推断实时更新次品率的方法，给企业的决策提供了数据支持。

对于问题一，本文使用假设检验的方法来确定是否接收零配件。考虑到样本量的大小，分别采用 t 检验和 z 检验，分别在 95% 和 90% 的置信度下求解所需的最小样本量，并得出相应的检验统计量，得到显著性检验结果。

对于问题二，需要企业在生产过程中做出由零部件到成品的决策优化。这涉及四个主要决策：是否检测零件 1、零件 2、成品，以及是否拆解不合格成品。利用 Python 程序建立蒙特卡洛模拟模型，根据次品率生成随机的零件质量状态。按照随机生成的零件状态和 16 种不同的决策方案，更新各阶段所需成本，并在模拟结束后计算出不同决策方案下的平均成本。得出平均成本最低的决策方案，以最大化产品的平均利润。

对于问题三，需要企业优化多阶段决策以降低生产成本，获得最大利润。由于问题三需要考虑零配件到半成品再到成品的复杂流程，若采用问题二的蒙特卡洛随机模型求解，运算速度较慢且输出结果具有波动性。本文采用概率动态转移函数，将每一个零件组合部分抽象为决策单元，通过计算期望成本和对应次品率，在决策单元依据装配顺序进行正向传播。将方程和具体参数代入代码计算不同决策路径下的期望成本，确定最优决策途径以实现最低平均成本，从而最大化利润。

对于问题四，已知次品率信息可以通过有效抽样检测机制获得，企业需要根据已有的次品率信息，合理转变决策方案。本文采用贝叶斯推断，通过似然函数以及先验概率，得到更新后的后验概率即次品率估计。根据更新后的次品率，结合前文所建立的决策模型，得以调整决策方案，最大化利润。

文章最后对模型的优缺点进行分析和评价，并阐述了模型的推广价值。

关键字：生产过程决策　假设检验　蒙特卡洛模拟　动态转移函数　贝叶斯推断

一、 问题重述

1.1 问题背景

在探讨某企业针对其畅销电子产品生产流程的优化策略时，企业需要聚焦于其装配过程中所需零配件的质量控制，及其对最终成品合格率的影响。

该生产系统呈现出一种复杂的逻辑依赖关系，即成品的质量状态受制于零配件的个体质量：任一零配件的不合格直接导致成品不合格；同时，即便所有零配件均达到质量标准，成品仍有不合格的可能，体现了生产过程中的不确定。对于检测出的不合格成品，企业面临两种主要处理策略：一是直接报废，该策略能清除不合格品，但可能伴随较高的直接经济损失；二是进行拆解再利用，此策略能回收部分零配件价值，但拆解过程本身需投入额外拆解成本。

面对这一挑战，企业需构建一套高效且经济的检测决策机制，旨在最小化生产成本与不合格品带来的损失。具体而言，企业需考虑在零配件采购入库、装配前检测、成品质量检测、成品拆解利用等多个环节进行合理的决策，以最大化企业既得利润。

因此，企业需科学决策何时采取何种措施，在保障产品质量的同时，实现生产成本的最优化控制。部分单一品种产品的生产决策采用盈亏平衡，多品种产品的生产优化决策模型采用的比较多的就是线性规划、目标规划与整数规划，分支定界法和本量利分析[1]。智能算法作为主要研究手段被广泛应用于工序规划的研究中，比如遗传算法，蚁群算法，模拟退火以及分层多算子遗传算法 [2]。这一过程不仅考验企业的技术实力，更对其数据分析与决策能力提出了更高要求。

1.2 问题要求

问题 1 供应商声称一批零配件的次品率不会超过某个标称值。企业需要采用抽样检测方法决定是否接收从供应商购买的这批零配件。该问题需要在标称值为 10%，信度为 95% 和 90% 两种情况下，计算对应的最少抽样检测次数及相应检验统计量。

问题 2 面对不同生产情境，企业需构建一套综合决策框架，以决定不同情况下是否对零配件及成品进行检测，是否拆解不合格产品。题目给出六种不同情况，对应不同的零件次品率、成品次品率、调换费用、检测费用和拆解费用，需要综合考量次品率、各阶段成本及产品调换损失等多维度因素，得出最优决策方案，并给出决策的依据及相应的指标结果。

问题 3 在多道工序、多零配件的复杂生产环境中，在考虑各工序次品率的基础上，动态调整生产流程与资源配置，以达到整体生产成本的降低与产品质量的提升。题目要

求在已知次品率和各种成本的情况下，设计零配件-半成品-成品生产线成本最低的最优决策流程，说明决策的依据及相应指标。

问题 4 在假设次品率信息已通过有效抽样检测机制获得的前提下，需重新审视并优化问题二与问题三中的决策过程。本问题强调了对已有检测数据的充分利用，通过更精准的次品率估计，进一步细化生产与拆解决策，以及多工序多零配件生产流程的优化。此过程旨在减少因信息不足或误判而导致的额外成本，提升生产决策的科学性与有效性。

二、 问题分析

2.1 问题总分析

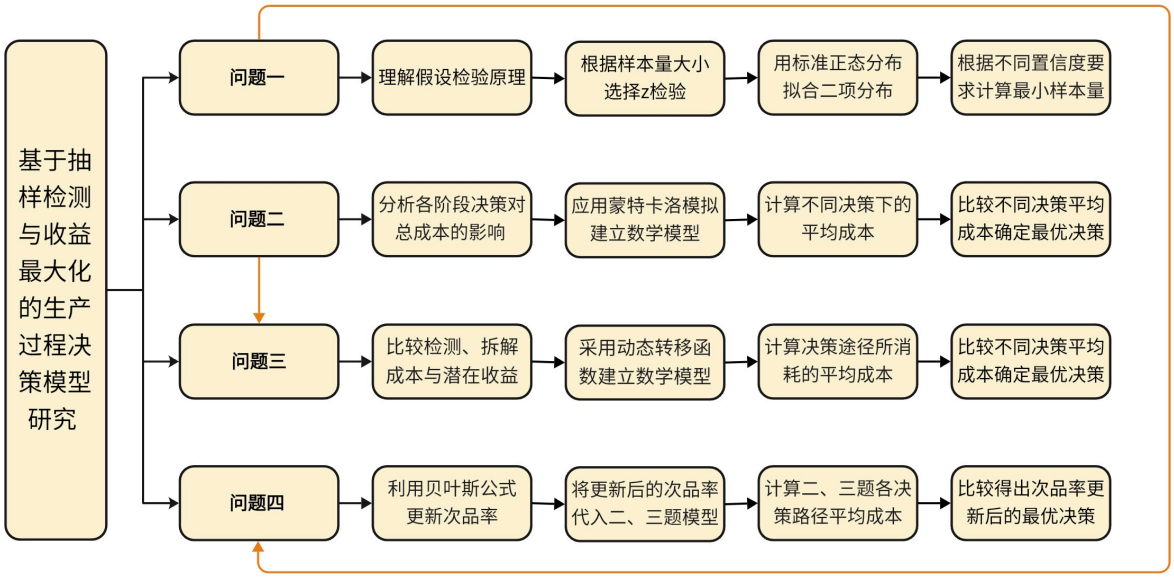

图 1　问题总分析流程图

本文的题目总体分析流程图如图 1，根据假设检验原理，使用 t 检验、z 检验，拟合二项分布并求得对应的最小样本量，并计算对应检验统计量；问题二根据零件加工流程，使用蒙特卡洛模拟建立数学模型，遍历所有决策得出最优解；问题三是问题二的复杂形式，零件加工流程更加复杂。将整体决策流程拆分为决策单元，使用动态转移函数建立数学模型，求解最优决策；问题四要求结合第一问，使用贝叶斯推断更新次品率，重新计算第二三问。

2.2 问题一分析

针对问题一，我们可以从统计学的角度入手，利用假设检验的方法来确定是否接收或拒收零配件。根据假设检验的原理，首先需要设定一个标称次品率作为基准，并通过

抽样检测计算实际的次品率，再根据抽样结果判断实际次品率是否与标称次品率存在显著差异。通常情况下，样本量较小时，数据的波动性较大，因此在样本量小于 30 的情况下，本文采用 t 检验来判断次品率是否显著高于或低于标称值。当样本量较大时，由于样本均值的分布趋于正态分布，采用 z 检验进行假设检验。对于两个不同置信度下的决策问题，即 95% 置信度下的拒收决策和 90% 置信度下的接收决策，分别通过计算得出每种情况下所需的最小样本量，以确保在既定的置信度下能够做出合理的决策。

2.3 问题二分析

对于问题二，企业生产过程中涉及多阶段的决策，目的是找到最优策略，降低生产成本并减少次品对市场和用户造成的影响。在整个流程中，企业需要针对零配件、装配环节以及成品的各个阶段作出一系列检测与处理的决策。可以初步简单推断，如果零件 1 或零件 2 的次品率较高且检测成本不高，则优先检测，以确保装配的零件质量，从而降低成品次品率。对于检测成本较高且用户退货损失较大的产品，可以考虑进行成品检测，减少不合格品进入市场的几率。如果拆解后零件的再利用价值较高，可以选择拆解不合格成品，否则，直接丢弃可能是更经济的方案。

本题中，需要做出的决策分别为是否检测零件 1，是否检测零件 2，是否检测成品，是否拆解成品。本文使用 Python 编程，应用蒙特卡洛模拟建立数学模型，在循环中遍历不同决策下，得出不同成本与次品率对应的产品平均利润大小，比较得出最优决策。

2.4 问题三分析

问题三的分析涉及企业在生产过程中不同阶段的是从半成品到成品的决策，主要分为两个步骤：第一步是从半成品到成品的决策，企业需要决定是否对半成品进行检测，成品是否检测，以及不合格成品是否进行拆解。每一个决策都会影响最终的生产成本和产品质量，因此需要根据次品率、检测成本、拆解成本和回收得失来决策。第二步是从配件到半成品的决策，企业需要决定是否对每个配件进行检测，确保半成品的质量，此外还需考虑不合格半成品是否拆解。在这个过程中，检测与拆解的成本需要与潜在的收益相比较，以最大化生产收益。

问题三是问题二的复杂化，需要考虑 8 个零配件各自合成 3 个半成品，半成品再装配成成品的复杂流程。采用问题二蒙特卡洛模拟与循环代码尝试求解，同时寻求更优化的数学模型，采用动态转移函数建立数学模型，划分决策单元，比较一个合格产品对应一条决策途径所消耗的平均成本，得出最优决策途径与对应的最低平均成本。

2.5 问题四分析

对于问题四，在重新审视问题二和问题三时，我们认识到次品率的定义需要依据抽样检测数据来估计，而非固定常量。这一变化引入了统计不确定性，影响了企业在生产过程中对次品率的理解与应对策略。抽样检测只能提供有限的信息，可能导致整体次品率的误判，因此次品率的估计需考虑样本的代表性及抽样误差，这对后续决策至关重要。使用贝叶斯推断方法可以结合先验知识与抽样数据，动态更新对次品率的估计，从而获取更为准确的次品率分布，减少决策的不确定性。此外，随着更多数据的收集与分析，企业应建立动态调整机制，有助于企业在快速变化的环境中保持竞争优势。

三、 模型假设

为简化问题，本文做出以下假设:

- 假设 1：半成品、成品的次品率是将正品零配件（或者半成品）装配后的产品次品率。
- 假设 2：不合格成品中的调换损失是指除调换次品之外的损失（如：物流成本、企业信誉等）。
- 假设 3：零件容量充足，可以满足大样本的抽样。

四、 符号说明

符号	说明	单位
p	次品率	%
p_0	次品率标称值	/
n	样本量	/
$\hat{p}$	实际观测的次品率	%
C	花费成本变量	元
N	决策数	/
$c^{(k)}$	当前件装配成本	元
$d^{(k)}$	当前件检测成本	元
$q^{(k)}$	当前件次品率	%
$t^{(k)}$	当前件拆除成本	元
θ	抽样检测的观测数据	/
k	检测到的不合格品数量	/

五、 问题一的模型的建立和求解

5.1 模型建立

二项分布 (Binomial Distribution) 是概率论中常见的离散概率分布，描述了在一系列独立重复的随机试验中，成功事件发生的次数的概率分布。在抽样检测中，二项分布被广泛应用于描述二元结果的概率分布情况。二项分布的概率质量函数可以表示为:

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

其中，n 表示试验的次数，k 表示成功事件发生的次数，p 表示单次试验成功的概率。

在抽样检测中，样本量的大小和置信水平影响着检测的准确性。通过二项分布，我们可以计算在给定置信水平下，需要的最小样本量，以确保检测结果的可靠性。

5.1.1 问题的统计模型

对于每批零部件，我们将其次品量视为一个随机变量，假设次品率为 p。供应商声称次品率不会超过某个标称值 p_0。因此，我们的任务是通过抽样检验来确定实际的次品率是否显著高于或低于 p_0。

5.1.2 假设检验的建立

对于以下两种情况分别进行假设检验:

- 情况一：在 95% 的置信度下，认定零配件次品率超过标称值 p_0，即拒收这批零配件。

$$\text{原假设 } H_0\text{：} p \leq p_0$$
$$\text{备择假设 } H_1\text{：} p > p_0$$

我们需要通过抽样来检测是否可以拒绝 H_0。

- 情况二：在 90% 的置信度下，认定零配件次品率不超过标称值 p_0，即接受这批零配件。

$$\text{原假设 } H_0\text{：} p \geq p_0$$
$$\text{备择假设 } H_1\text{：} p < p_0$$

我们需要通过抽样来检测是否可以拒绝 H_0。

在假设检验中，可能犯的错误有下面两种类型:

- 第一类错误率 (Type I error)：当 H_0 是真相时, 我们拒绝了 H_0。

- 第二类错误率 (Type II error)：当 H_1 是真相时, 我们没能拒绝 H_0。

由于假设检验中通常把 H_0 置于被保护的地位，显著性检验的思想诞生，旨在在能够控制第 I 类错误率低于一定水平的前提下，尽量减小第 II 类错误率。

5.1.3 样本量的确定

根据样本量大小的不同，我们需要采用不同的检验方法来确定最小样本量。

根据现实情况假设，在零部件的抽样检测中，若样本容量小于 30，我们采用 t 检验。t 检验 (t-test) 主要用于小样本量的假设检验，一般适用于样本量小于 30 且总体标准差未知的情况。在这种情况下，我们使用 t 分布来替代标准正态分布能够更好地处理小样本量带来的不确定性。

在样本量较小的 t 检验下，我们可以得到计算最小样本量，如下公式：

$$n = \frac{t^2 \cdot p_0 \left(1 - p_0\right)}{\left(\hat{p} - p_0\right)^2} \tag{1}$$

其中，t 是置信度下的 t 分布临界值，p_0 是标称值，$\hat{p}$ 是实际观测的次品率。

根据现实情况假设，在零部件的抽样检测中，若样本容量无限大，我们采用 z 检验。棣莫弗一拉普拉斯局部极限定理 (De Moivre-Laplace local limit theorem) 是概率论历史上第一个中心极限定理，针对二项分布的正态近似。由此我们可以知道，当样本趋近于无穷大时，标准正态分布可以拟合二项分布。

为了确保检测的准确性并控制误差，我们需要根据置信度水平和允许的误差率来确定所需的样本量。样本量的计算可以依据标准样本量计算公式进行：

$$n = \frac{Z^2 \cdot p_0 \cdot (1 - p_0)}{E^2} \tag{2}$$

其中，Z 是对应置信度的标准正态分布的临界值，E 是允许的误差率，可以根据实际要求而定，p_0 是标称值。

根据工厂零部件具体的产品类型、行业标准和质量要求，允许误差 E 有着不同的数值。在一般情况下，允许的误差常常设定为 0.05，这是一个常见的选择，能够综合考虑精度和实际情况。本题取 $E = 0.05$。

通过计算样本量 n，我们可以确定在不同样本量大小下，需要抽样检测的零配件数量。

5.1.4 检验统计量的计算

根据样本量大小的不同，我们需要采用不同的检验方法来计算检验统计量。

根据现实情况假设，在零部件的抽样检测中，若样本容量小于 30，我们采用 t 检验。单样本 t 检验的公式为：

$$t_0 = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} \tag{3}$$

其中，n 为总样本量，p_0 是标称值，$\hat{p}$ 是实际观测的次品率。根据 t_0 与正态分布下 t_α 对比，可以决定是否拒绝原假设。

正态分布 $N(\mu, \sigma^2)$ 的概率质量函数为：

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

任何正态分布都可以通过标准化，化简为标准正态分布。即，若 $X \sim N(\mu, \sigma^2)$，那么 $Z = \frac{X-\mu}{\sigma} \sim N(0, 1)$。

对于样本容量无限大的情况，二项分布拟合正态分布后，我们通过二项分布的均值以及方差可以得到：

$$E[x] = np \tag{4}$$

$$Var[x] = np(1-p) \tag{5}$$

根据以上式子拟合正态分布可知，检验统计量可以化为标准正态分布的 Z_0 值：

$$Z_0 = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} \tag{6}$$

其中，n 为总样本量，p_0 是标称值，$\hat{p}$ 是实际观测的次品率。根据 Z_0 与正态分布下 Z_α 对比，可以决定是否拒绝原假设。

5.1.5 根据抽样结果进行决策

对于情况 (1)，如果检验统计量超过相应的临界值，则拒绝原假设，认为次品率超过标称值。

对于情况 (2)，如果检验统计量值小于相应的临界值，则拒绝原假设，认为次品率不超过标称值。

5.2 模型求解

Step1：根据5.1.2 所述，我们可以得到两种情况下的零假设和备择假设。

Step2：查表获得 t 分布和 z 分布下的对应的临界值，其中 t 检验临界值需要根据实际样本量 n 求得 $n-1$ 自由度下的临界值。对于 z 分布可以得到，$Z_{0.05} = 1.96$、$Z_{0.1} = 1.64$。根据公式1、公式2，可以获得两种情况下对应的标准样本量。

Step3：根据5.1.5可知，通过公式3、公式6算出检验统计量，与对应临界值作比较，得出最终的决策结果。即是否接受该批次零部件。

5.3 求解结果

由于 t 检验下的样本量需要根据实际次品率带入1而定，因此求解结果需要视真实情况而定。以下展示 z 检验下的求得的最小样本量。

1) 对于置信度为 95% 下的样本量：

$$n_{0.05} = \frac{(1.96)^2 \cdot 0.1 \cdot (1-0.1)}{0.05^2} = 138.2976 \approx 139$$

2) 对于置信度为 90% 下的样本量：

$$n_{0.1} = \frac{(1.64)^2 \cdot 0.1 \cdot (1-0.1)}{0.05^2} = 96.8256 \approx 97$$

3) 对于情况 1，如果求得的检验统计量超过相应的临界值，则拒绝原假设，认为次品率超过标称值。

4) 对于情况 2，如果求得的检验统计量小于相应的临界值，则拒绝原假设，认为次品率不超过标称值。

六、 问题二的模型的建立和求解

6.1 模型建立

6.1.1 绘制流程图

根据题目绘制零件 1 和零件 2 加工过程的流程图如图 2。图中蓝绿色框表示生产流程中需要做出的主要决策。

6.1.2 定义关键参数

零件 1 次品率，零件 2 次品率，零件 1 检测成本，零件 2 检测成本，成品检测成本，成品拆解成本，成品调换成本。

6.1.3 编辑代码

建立函数，使用嵌套的循环和多维数组枚举所有可能的组合，通过 for 循环遍历四种布尔变量（零件 1 是否检测、零件 2 是否检测、成品是否检测、不合格品是否拆解），总共 4*4=16 种可能，根据流程图设计步骤，每经过一个步骤就根据 if 语句更新产品生成花费成本变量 C（比如拆解成品后，需要加上成品拆解流程）。

$$C_{\text{总成本}} = C_{\text{购买}} + C_{\text{检测}} + C_{\text{装配}} + C_{\text{拆解}} + C_{\text{调换}}$$

计算多次模拟结果的平均成本 C_{avg}, 最终得出每一种决策方案下的平均值。

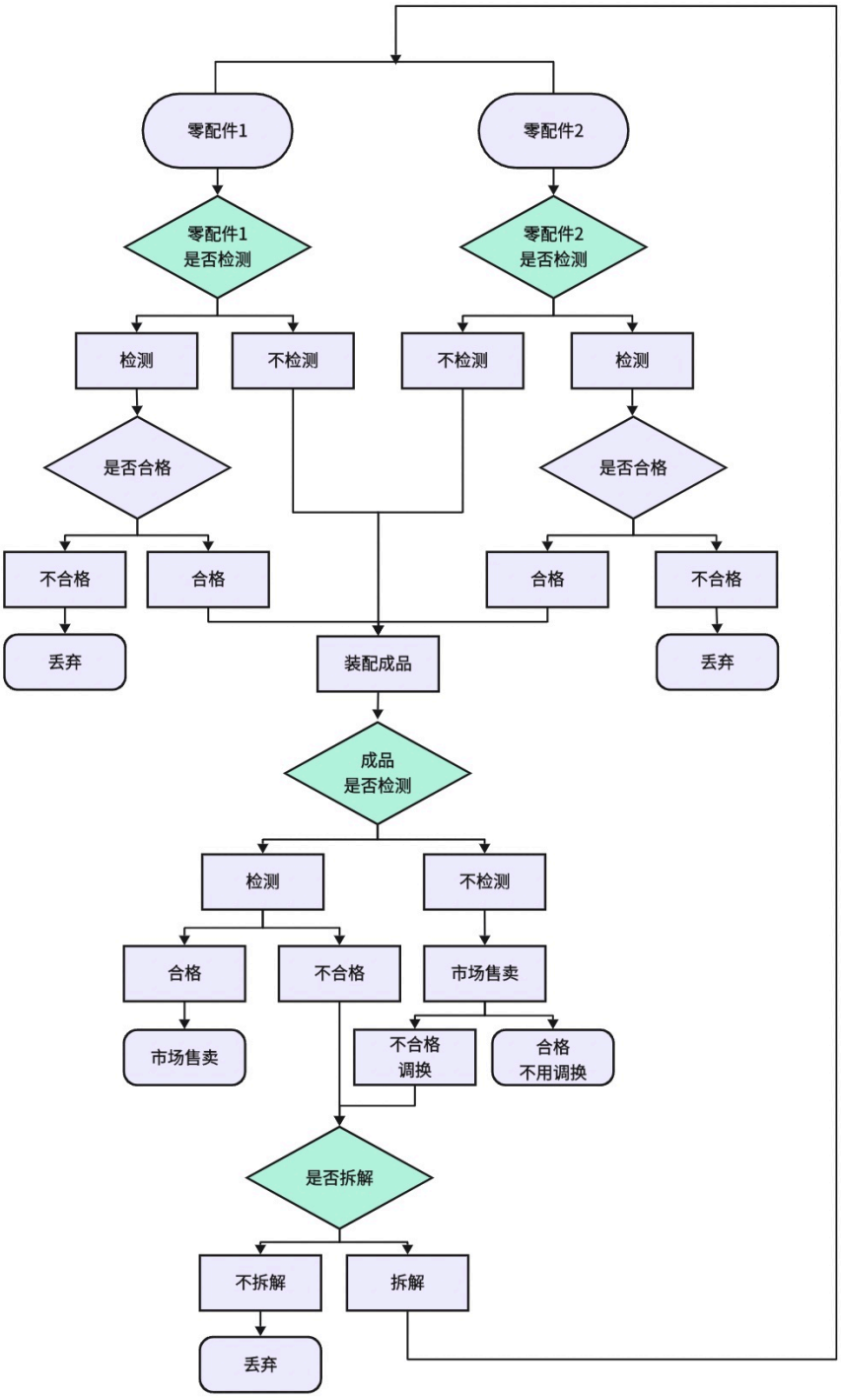

图 2　问题二流程图

$$C_{\text{avg}} < C_{\text{best}}$$

在每次计算后，通过上述语句判断该决策流程，是否是目前最优的决策组合，将得到的平均最低成本。通过净利润公式计算平均最优利润，并记录最优的决策组合。净利润公式为：

$$W_{\text{净利润}} = W_{\text{售出价格}} - C_{\text{总成本}}$$

6.1.4 蒙特卡洛模拟

蒙特卡洛模拟是以概率和统计理论方法为基础的一种计算方法，通过使用随机数（或伪随机数），将所求解的问题同一定的概率模型相联系，用电子计算机实现统计模拟或抽样，以获得问题的近似解。为了更好地模拟不同策略下的零件加工与产品生产的真实表现，本题的 Python 代码使用蒙特卡洛模拟多次运算（100000 次以上），生成随机的零件质量状态（即零件是否次品），并计算不同检测策略下的平均总成本。每次模拟中，代码根据随机生成的零件状态和决策方案，更新各阶段的成本，并在模拟结束时计算出不同检测策略下的平均收益。

6.1.5 明确产品次品率

已知在装配的成品中，只要其中一个零配件不合格，则成品一定不合格；如果两个零配件均合格，装配出的成品也不一定合格。题目表格中给出的成品次品率，是在两个零件均合格的情况下成品有 m 的次品率，那么零件是否检测会影响最终成品的实际次品率。分类讨论如下：

令零件 1 次品率为 a，零件 2 次品率为 b，两个零件均合格的情况下成品次品率为 m，成品实际次品率为 p_1。

1) 如果零件 1 检测，零件 2 不检测，$p_1 = b + m * (1 - b)$ ；

2) 如果零件 1 不检测，零件 2 检测，$p_1 = a + m * (1 - a)$；

3) 如果零件 1 检测，零件 2 检测，$p_1 = m$；

4) 如果零件 1 不检测，零件 2 不检测，$p_1 = 1 - (1 - a)(1 - b)(1 - m)$；

6.2 模型求解

以情况 4 为例，求解该情况下最优决策：

Step1：带入零件 1 次品率为 0.2，零件 2 次品率为 0.2，零件 1 检测成本为 1 元，零件 2 检测成本为 1 元，成品检测成本为 2 元，成品拆解成本 5 元，成品调换成本 30 元的参数值。应用函数进行 100000 次蒙特卡洛模拟。

Step2：从输出结果可以直观得出第二种决策对应的平均成本最低，即最优平均利润为 10.42，最优决策组合为：检测零件 1 为 True；检测零件 2 为 True；检测成品为 True；拆解不合格成品为 True（true 为 1，false 为 0）。增大蒙特卡洛模拟次数，输出结果不变。说明为了降低生产流程的总成本，应该检测零件 1 和零件 2，检测成品，拆解成品。这与如果零件 1 或零件 2 的次品率较高且检测成本不高，则优先检测，以确保装配的零件质量，从而降低成品次品率以及对于检测成本较高且用户退货损失较大的产品，可以考虑进行成品检测的预期结果相符合。

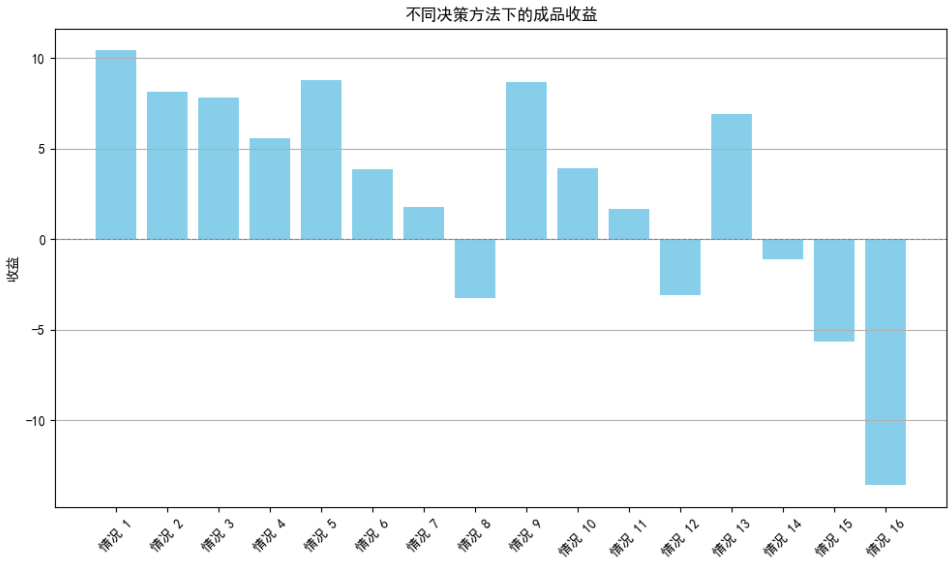

图 3　情况 4 中 16 种决策对应计算结果对比图

6.3 求解结果

重复上述步骤，得出情况 1-6 对应的最优决策：

情况 1：最优平均利润为 15.69 元。最优决策组合为不检测零件 1，不检测零件 2，不检测成品，拆解不合格成品。

情况 2：最优平均利润为 8.97 元。最优决策组合为检测零件 1，检测零件 2，不检测成品，拆解不合格成品。

情况 3：最优平均利润为 14.45 元。最优决策组合为不检测零件 1，不检测零件 2，检测成品，拆解不合格成品。

情况 4：最优平均利润为 10.42 元。最优决策组合为检测零件 1，检测零件 2，检测成品，拆解不合格成品。

情况 5：最优平均利润为 14.13 元。最优决策组合为不检测零件 1，检测零件 2，检测成品，拆解不合格成品。

情况 6：最优平均利润为 18.62 元。最优决策组合为检测零件 1，不检测零件 2，不检测成品，不拆解不合格成品。在计算过程中，该决策与不检测零件 1，不检测零件 2，不检测成品，拆解不合格成品这一决策产生的平均成本较为接近，差距极小，因此可以得出是否拆解不合格成品对该情况生产平均成本影响不大的结论。

该结果符合问题分析中预期，其中情况 4 对应的生产成本更低，这可能与检测成本较低有关；情况 2 相对于情况 1 次品率较高，因此最优决策选择检测零件 1 与成品；情况 3 相对与情况 1 调换成本增高，因此需要检测成品，减少用户退货损失；情况 5 中零件 1 检测成本过高，因此选择不检测零件 1；情况 6 拆解成本较高，因此不选择拆解不合格成品。

情况	最优平均利润	检测零件 1	检测零件 2	检测成品	拆解不合格成品
情况 1	15.69	否	否	否	是
情况 2	8.97	是	是	否	是
情况 3	14.45	否	否	是	是
情况 4	10.42	是	是	是	是
情况 5	14.13	否	是	是	是
情况 6	18.62	是	否	否	否

表 1　问题二各情况的最优决策组合与最低平均成本

七、 问题三的模型的建立和求解

7.1 模型建立

7.1.1 明确流程

问题三的情况相对复杂，新增零件 1-8 的检测成本和次品率，半成品和成品的次品率和检测成本，半成品的装配成本，成品的装配成本，不合格成品调换成本多个变量。对于每一个零件，需要决策是否需要检测，装配成为半成品后，需要决策是否检测半成品 1-3，如果检测，是否拆解不合格的半成品，需要决策是否检测成品，以及调换产生的不合格半成品是否拆解。令总共需要做的决策数为 N，各个决策之间有 M 种排列组合的可能，即有 M 种决策途径。

$$N = 8 + 3 + 1 + 1 + 3 = 16$$

(8 个零件检测 + 3 个半成品检测 + 成品检测 + 成品拆解 + 半成品拆解)

$$M = 2^{16}$$

7.1.2 蒙特卡洛模拟尝试

基于问题二的基础，使用蒙特卡洛方法模拟不同的决策组合，计算其对应的总成本 C，并尝试找到最优决策组合。建立嵌套循环和数组，遍历 M 种决策途径，带入参数求解最低成本对应的决策途径。

由于决策途径数量较多，对任意途径需要模拟的零件数不能太小，蒙特卡洛模拟方法的计算量较大，计算时间较久。模拟次数为 500 时最优决策途径最低平均成本为 32.3 元，输出结果如表 2。

决策	是否检测	决策	是否检测
零件 1	是	零件 5	是
零件 2	是	零件 6	是
零件 3	是	零件 7	是
零件 4	是	零件 8	是
半成品 1	否	成品是否检测	是
半成品 2	否	半成品 1 是否拆解	否
半成品 3	否	半成品 2 是否拆解	是
成品是否拆解	否	半成品 3 是否拆解	是

表 2　蒙特卡洛模拟法零件、半成品及成品的检测和拆解结果

7.1.3 基于次品率的期望成本

由于问题三的决策途径数量级较大，无法通过蒙特卡洛模拟来近似得出最终的结果。已知每个零部件、半成品、成品的次品率，可以通过事件的相互独立性得到每个决策后的实际次品率。通过次品率可计算后续期望成本。

通过不同的决策路径，获得对应的次品率，由此获得对应期望成本。比较不同路径下的期望成本，获得最终的最小期望成本，即最优决策。

7.1.4 概率动态转移函数模型

本文将零件到半成品，半成品到成品的部分分为四个决策单元，如图4左图所示。

对于单个决策单元，本文将其抽象为一个类似于单层感知机的元件。由多个决策单元组合为整体的决策流程，零件的组装过程看做零件的成本动态转移向成品过程。对于每一个决策单元，由上一层每个部件的期望成本以及实际次品率，获得当前部件的期望成本以及综合次品率。按照题意，假设有 m 道工序、n 个零配件。

决策单元的动态转移函数如下推导：

函数输入值为 n 个 $C_i^{(k-1)}$ 和 $P_i^{(k-1)}$，其中 $C_i^{(k-1)}$ 代表上一层第 i 件的期望成本，$P_i^{(k-1)}$ 代表上一层第 i 件的次品率。当前部件位于第 k 层，上一层即为 $k-1$ 层，内部包含了 n 个部件。如图4右图所示。

令 $c^{(k)}$ 为当前件的装配成本，$d^{(k)}$ 为当前件的检测成本，$q^{(k)}$ 为当前件次品率，$t^{(k)}$ 为当前件拆除成本。

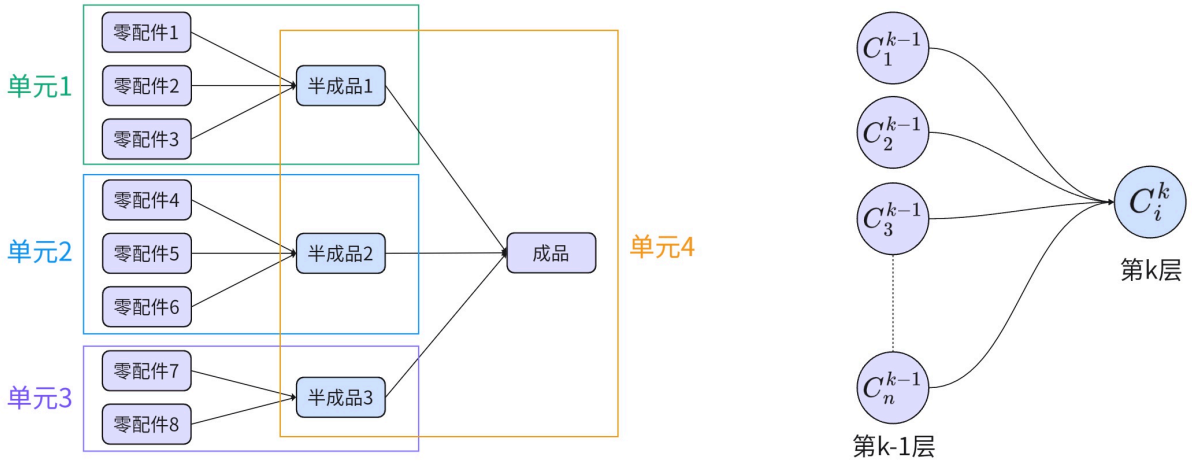

图 4　决策单元模型的建立

当前件的次品率为：

$$p^{(k)} = 1 - (1-q)\prod^{n}(1 - P_i^{(k-1)}) \tag{7}$$

情况一：不检验当前件是否合格，成本为：

$$c^{(k)} + \sum_{i=1}^{n} C_i^{(k-1)} \tag{8}$$

情况二：检验当前件是否合格，令最终得到好件的期望成本为 E，成本如下讨论：

- 当前件以 $1-p^{(k)}$ 的概率检测通过，成本为：

$$c^{(k)} + d^{(k)} + \sum_{i=1}^{n} C_i^{(k-1)} \tag{9}$$

- 当前件 $p^{(k)}$ 的概率检测不通过，成本为：

$$C^k = \begin{cases} c^{(k)} + d^{(k)} + \sum_{i=1}^{n} C_i^{(k-1)} + E & \text{若不拆解} \\ c^{(k)} + d^{(k)} + E + t^{(k)} & \text{若拆解} \end{cases} \tag{10}$$

我们计算检测产品下的期望成本，分为拆解与不拆解两种情况：

$$E_{\text{不拆解}} = (1-p^{(k)}) \times (c^{(k)} + d^{(k)} + \sum_{i=1}^{n} C_i^{(k-1)}) + p^{(k)} \times (c^{(k)} + d^{(k)} + \sum_{i=1}^{n} C_i^{(k-1)} + E_{\text{不拆解}}) \tag{11}$$

$$E_{\text{拆解}} = (1-p^{(k)}) \times (c^{(k)} + d^{(k)} + \sum_{i=1}^{n} C_i^{(k-1)}) + p^{(k)} \times (c^{(k)} + d^{(k)} + E_{\text{拆解}} + t^{(k)}) \tag{12}$$

对于公式11、公式12进行解方程，可得：

$$E_{\text{不拆解}} = \frac{c^{(k)} + d^{(k)} + \sum_{i=1}^{n} C_i^{(k-1)}}{1 - p^{(k)}} \tag{13}$$

$$E_{\text{拆解}} = \frac{c^{(k)} + d^{(k)} + (1-p^{(k)})\sum_{i=1}^{n} C_i^{(k-1)}}{1-p^{(k)}} = \frac{c^{(k)} + d^{(k)} + t^{(k)} \times p^{(k)}}{1-p^{(k)}} + \sum_{i=1}^{n} C_i^{(k-1)} \quad (14)$$

由以上式子可得，当前件成本的动态转移方程为：

$$C^{(k)} = \begin{cases} \frac{c^{(k)}+d^{(k)}+t^{(k)}\times p^{(k)}}{1-p^{(k)}} + \sum_{i=1}^{n} C_i^{(k-1)} & \text{若检测且拆解} \\ \frac{c^{(k)}+d^{(k)}+\sum_{i=1}^{n} C_i^{(k-1)}}{1-p^{(k)}} & \text{若检测不拆解} \\ c^{(k)} + \sum_{i=1}^{n} C_i^{(k-1)} & \text{若不检测} \end{cases} \quad (15)$$

显然可得，当前件最终次品率的动态转移方程为：

$$P^{(k)} = \begin{cases} 0 & \text{若检测} \\ p^{(k)} & \text{若不检测} \end{cases} \quad (16)$$

得到当前件期望成本动态转移方程以及当前件最终次品率动态转移方程，即获得当前决策单元的输出，此输出可以跟随当前层流入下一个决策单元。

对于成品，迭代得到成品 $C^{(m)}$ 和 $P^{(m)}$。由于成品存在不合格成品中的调换损失 r（如：物流成本、企业信誉等），因此最终的成品期望成本为：

$$E^{(m)} = ((1-P^{(m)}) \times (C^{(m)}) + (P^{(m)}) \times (C^{(m)} + r + E^{(m)})) \quad (17)$$

解方程得：

$$E^{(m)} = \frac{r \times P^{(m)} + C^{(m)}}{(1-P^{(m)})} \quad (18)$$

可得最终利润为：

$$E_{\text{Profit}} = \text{Price} - E^{(m)} \quad (19)$$

7.2 模型求解

Step1：带入参数，题目三参数如图5所示。参数数量较多，将具体的参数大小带入代码中的数组。

Step2：运行代码，使用动态转移方程来计算零件、半成品和成品的预期成本。遍历 M 种可能的决策组合，分别计算每个决策组合的总成本。更新当前的最优决策组合和对应的最低平均成本。

Step3：输出结果，Python 程序将比较所有途径下的平均成本大小输出最低平均成本和最优决策途径。在代码执行完毕后，最终的输出将是最优决策途径以及对应的最低平均成本。

零配件	次品率	购买单价	检测成本	半成品	次品率	装配成本	检测成本	拆解费用
1	10%	2	1	1	10%	8	4	6
2	10%	8	1	2	10%	8	4	6
3	10%	12	2	3	10%	8	4	6
4	10%	2	1					
5	10%	8	1	成品	10%	8	6	10
6	10%	12	2					
7	10%	8	1		市场售价		调换损失	
8	10%	12	2	成品	200		40	

图 5　题目三参数图

7.3 求解结果

本题建模后的求解结果，取最优的 12 个决策集，给出决策结果、各半成品期望成本及最终成品售出预期收益。详见附录。决策集分析及可视化见问题四。

八、 问题四的模型的建立和求解

8.1 模型建立

8.1.1 明确具体问题

针对问题 4，需要重新审视前面问题 2 和问题 3 中次品率定义。假设所有的次品率（包括零配件、半成品和成品的次品率）均通过抽样检测方法得出。这种情况下必须考虑到抽样检测方法本身所带来的不确定性。由于抽样检测只能提供部分信息，因此其结果并不能完全代表整体情况。为了更准确地估计次品率，本文采用贝叶斯推断方法，从而得到更新后的次品率。

贝叶斯推断是一种统计方法，它将抽样结果与已有的先验信息结合起来，从而形成更准确的次品率估计。即后验概率（考虑相关证据或数据后，某一事件的条件概率）作为先验概率（考虑相关证据或数据前，某一事件不确定性的概率）和似然函数（由观测数据的统计模型（概率模型）推导而得）这两个前因导出的结果。通过这种方式，可以不断更新次品率的概率分布，使其更加接近真实情况。这些次品率的更新将直接影响后续的生产决策，从而提高生产效率和产品质量。贝叶斯推断提供了一种灵活、动态的方式来应对不确定性，使决策更加科学和合理。

8.1.2 次品率变换导致决策变换的可视化推导

假设次品率是通过抽样方法得出，使用贝叶斯推断更新产品的次品率。本文通过控制其他决策环节的次品率不变，单纯变化一个决策环节的次品率，观察决策结果是否会

发生相应的转变。

以问题 2 的情况六为例，控制零部件 2，成品的次品率不变，零部件 1 的次品率在 1% 到 20% 之间浮动。使用问题 2 的蒙特卡洛模型进行计算可以得出一下可视化散点图。

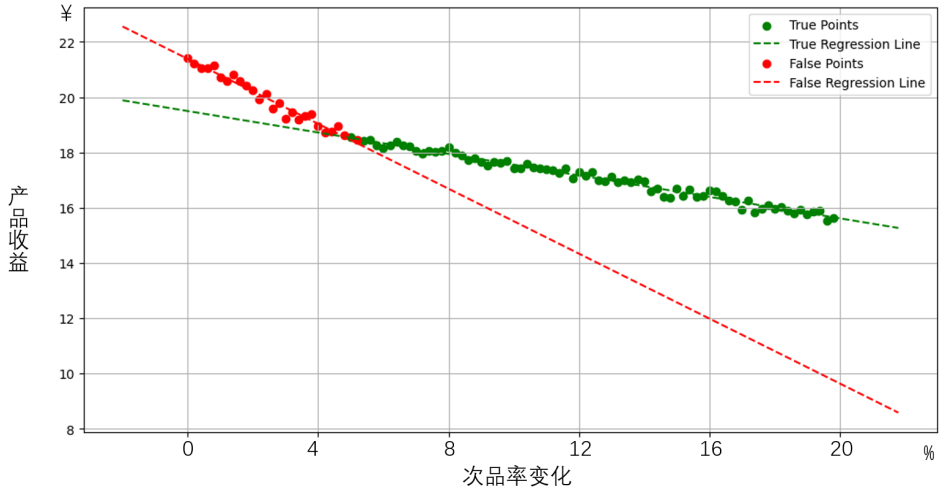

图 6　次品率变化导致决策变换可视化

如图6所示，由此可以得到，在零件 1 次品率为 4.2% 左右时，要使得成本最小化，决策由零件 1 不检测转变为检测。

8.1.3 贝叶斯推断模型

贝叶斯推断的核心公式为：

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \tag{20}$$

其中：

- θ 表示待估计的次品率；
- D 表示抽样检测的观测数据；
- $P(\theta|D)$ 是后验概率，即给定检测数据 D 后的次品率的概率分布；
- $P(D|\theta)$ 是似然函数，表示在次品率为 θ 的情况下观测到数据 D 的概率；
- $P(\theta)$ 是先验概率，表示在检测之前的次品率分布。

通过贝叶斯推断，可以将检测结果与企业之前的经验数据（先验概率）结合，更新得到新的次品率估计。

8.1.4 贝叶斯推断的具体计算

已知零配件的次品率服从二项分布，即 $X \sim \mathrm{B}(n,\theta)$，其中 n 为检测样本量，θ 为次品率。抽样过程中检测到 k 个不合格品，则似然函数为：

$$P(X=k \mid \theta)=\binom{n}{k}\theta^{k}(1-\theta)^{n-k} \tag{21}$$

由于先验分布为次品率分布，其取值在 (0,1) 之间。由于假设中检测样本量足够大，根据中心极限定理，当样本量 n 较大时，后验分布会趋近于正态分布。但是，对于次品概率的取值，可以使用更加贴合的 Beta 分布。

Beta 分布是指一组定义在 (0,1) 区间的连续概率分布。Beta 分布的概率密度函数是:

$$f(x;\alpha,\beta)=\frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}du}=\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1} \tag{22}$$

其中 $\Gamma(z)$ 是 Γ 函数。如果 n 为正整数，则有 $\Gamma(n)=(n-1)!$。随机变量 X 服从参数为 α,β 的 B 分布通常写作 $X \sim \mathrm{Beta}(\alpha,\beta)$

共轭性质指的是在贝叶斯推断中，如果先验分布和似然函数的形式相同，那么后验分布也将具有相同的形式。在二项分布和 Beta 分布的情况下：

假设 θ 的先验分布为 $\mathrm{Beta}(\alpha,\beta)$。似然函数 $P(D|\theta)$ 为二项分布：

$$P(X=k|n,\theta)=\binom{n}{k}\theta^{k}(1-\theta)^{n-k} \tag{23}$$

根据贝叶斯定理，后验概率 $P(\theta|D)$ 的分布可以表示为：

$$P(\theta|X=k) \propto P(X=k|\theta)P(\theta) \tag{24}$$

将先验概率 $P(\theta)$ 的分布和似然函数 $P(D|\theta)$ 代入，我们有：根据贝叶斯定理，后验概率 $P(\theta|D)$ 的分布可以表示为：

$$P(\theta|X=k) \propto P(X=k|\theta)\cdot P(\theta) \propto \theta^{k}(1-\theta)^{n-k}\cdot\theta^{\alpha-1}(1-\theta)^{\beta-1} \tag{25}$$

合并后：

$$P(\theta|X=k) \propto \theta^{k+\alpha-1}(1-\theta)^{(n-k)+\beta-1} \tag{26}$$

后验概率 $P(\theta|D)$ 的分布可以看作是一个新的 Beta 分布：

$$\theta|X=k \sim \mathrm{Beta}(\alpha+k,\beta+n-k) \tag{27}$$

其中：

- α 和 β 是先验 Beta 分布的参数；
- k 是检测到的不合格品数量；

- n 是检测样本量。

通过贝叶斯推断更新后的 θ 代表新的次品率估计。通过抽样检测，企业可以灵活地获得更准确的次品率数据。这些数据会对之前问题 2 和问题 3 中所涉及的决策过程产生显著影响。因此，企业需要根据这些新数据重新评估和调整生产、检测和拆解策略，以确保运营的高效性和经济性。

8.2 模型求解

由于实际的抽样检测结果未知，我们可以根据

Step1：对于问题二，以情况四为例，通过贝叶斯推断，基于已有的检测数据，选取合适的 n 值和 k 值，完成了对次品率的后验分布计算，使用更新后的 Beta 分布参数进行蒙特卡洛模拟，分析不同决策路径下的成本。

对于先验 Beta 分布参数，Beta 分布的形状可以通过其两个参数（α 和 β）进行调节，这使得它能够表示各种不同的先验信念。例如，当 α > β 时，分布偏向于 1（表示高成功率），而当 α < β 时，分布偏向于 0（表示低成功率）。通过调整 Beta 分布的参数 α 和 β 来实现次品率最接近问题二表格中的数值。Beta 分布的均值公式为：

$$\mu = \frac{\alpha}{\alpha + \beta} \tag{28}$$

想要让均值接近 0.2，需要调整 α 和 β，使得：

$$\frac{\alpha}{\alpha + \beta} = 0.2$$

假设 $\alpha + \beta = N$，则：

$$\alpha = 0.2 \times N$$

$$\beta = (1 - 0.2) \times N = 0.8 \times N$$

Beta 分布的方差可以通过以下公式计算：

$$\text{Var} = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)} \tag{29}$$

如果希望减少次品率的波动性，可以增大 N 的值，从而让分布更加集中。零件 1 的后验概率模型：

$$\theta_1 \mid X = k_1 \sim \text{Beta}(\alpha_1 + k_1, \beta_1 + n_1 - k_1)$$

零件 2 的后验概率模型：

$$\theta_2 \mid X = k_2 \sim \text{Beta}(\alpha_2 + k_2, \beta_2 + n_2 - k_2)$$

根据零件 1 和零件 2 更新后的次品率，结合成品次品率的后验概率模型，应用问题二中分析的动态成品次品率公式，计算更新后的成品次品率。

更新问题二 Python 代码，使用蒙特卡洛模拟遍历 16 种决策途径，输出结果，绘制柱状图如图7。

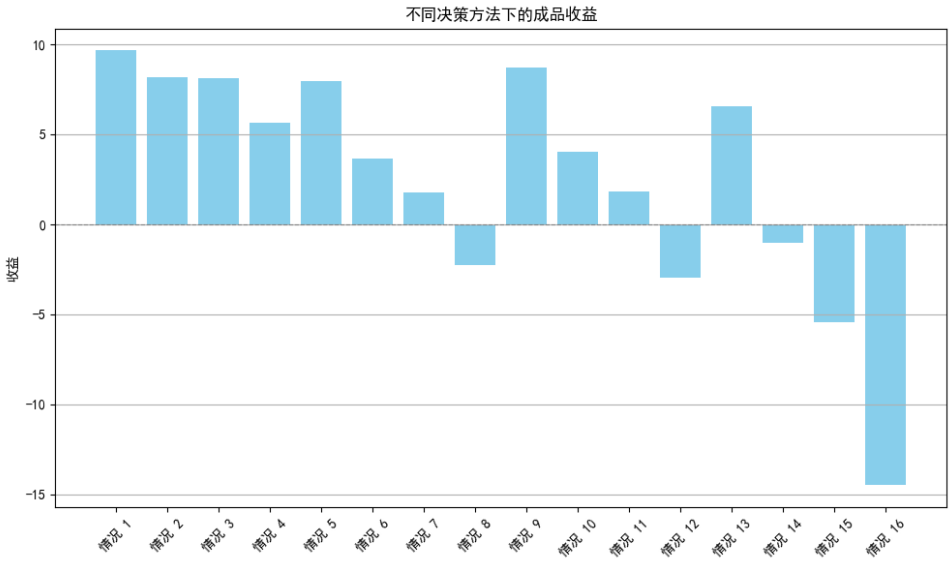

图 7　问题 4 中重算问题二情况 4 结果柱状图

Step2：探究情况四下成品检测的不合格品数量 k 的变化对 16 种决策途径各自最低平均成本的大小影响，控制其它参数不变，改变成品检测 k 值（1000-3000），绘制三维图如图8所示。可以判断各个决策的平均最低成本差异，随着 k 增大，各决策消耗的平均成本均随之增大。

Step3：对于问题三同理，根据公式 20-29 推导，更新零件次品率，更新成品次品率，更新问题三的 Python 代码。详细结果见附录 B 问题 4 结果。

8.3 求解结果

对于问题二的重算，求解结果如下表。

对于问题三，取最优的 12 个决策集，给出决策结果、各半成品期望成本及最终成品售出预期收益。详见附录。

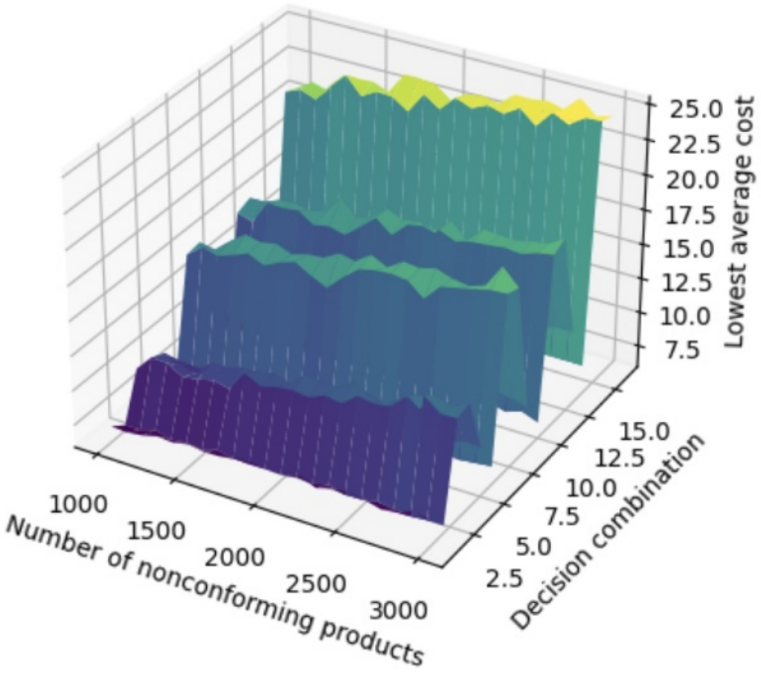

图 8　问题四贝叶斯法成品 k 值对各决策平均最低成本影响图

情况	最优平均利润	检测零件 1	检测零件 2	检测成品	拆解不合格成品
情况 1	15.74	否	否	否	是
情况 2	9.06	是	是	否	是
情况 3	14.56	否	否	是	否
情况 4	9.67	是	是	是	否
情况 5	13.98	否	是	是	是
情况 6	17.83	是	否	否	否

表 3　问题四中重算问题二各情况的最优决策组合与最低平均成本

九、 模型的误差分析

蒙特卡洛方法依赖于大量的随机样本来估计期望值或概率分布。如果样本数量不足，估计结果可能具有较大波动。如果随机数生成器质量不佳，采样结果可能带有系统性偏差，影响估计精度。伪随机数生成器的周期性和相关性可能会导致偏差。在进行数值计算时，由于计算机浮点数表示的精度有限，可能会引入舍入误差，尤其在多次迭代或复杂运算中，舍入误差可能积累。

贝叶斯模型依赖先验分布，如果选择的先验不准确或偏离实际情况，则会引入先验误差。如果观测数据质量差或者样本量少，似然函数估计可能不准确，导致后验分布的偏差。

通过增加样本量、改进采样方法、使用适当的先验分布以及进行敏感性分析等方法，可以有效地控制和减少误差。

十、 模型的评价

10.1 模型的优点

- 优点 1：模型通过大量模拟和采样，可以提供逼近最优解的近似解。
- 优点 2：蒙特卡洛模拟适合并行处理，因此可以在多核系统中并行运行，速度提升显著。
- 优点 3：动态转移方法通过系统化的方式明确问题的分解、状态、决策和状态转移，使得问题更有条理性。通过构建状态转移方程，能保证找到最优解，尤其适用于问题有最优子结构的情况

10.2 模型的缺点

- 缺点 1：由于蒙特卡洛模拟存在随机性，因此模拟结果存在随机波动，最低平均成本精确值存在误差，可能出现微小变动。模型假设零件总数足够多，不需考虑零件次品率可能因为成品拆解，不合格零件重返流程产生动态变化，但是在实际生产过程中需要加以考虑。虽然通过增加采样次数可以降低误差，但误差不容易预测，尤其是在某些复杂的多峰分布或高维问题中。
- 缺点 2：没有充分考虑企业可能面对的不确定性问题，比如市场需求波动、原材料成本变化等外部因素。尽管通过模拟可以估计次品率和成本，但如果外部环境变化较大，可能会影响策略的有效性。
- 缺点 3：动态转移模型和蒙特卡洛模拟模型在状态空间或决策空间非常大的问题中（如上亿种状态可能需要过长的计算时间），可能需要付出巨大的计算成本。

10.3 模型的推广

该决策模型具备广泛的应用价值，该模型考虑了检测和拆解的决策过程，模型能够帮助企业合理安排检测流程，减少不必要的检测环节，优化检测成本，在多行业、多场景的生产优化、质量控制、成本管理等方面具有潜力。随着智能制造、自动化生产和绿色生产的不断发展，该模型还可以与先进技术相结合，有效减少资源浪费，降低生产过程中不必要的能耗和材料消耗，成为企业实现精益生产和可持续发展的重要工具。模型

使用的动态规划方法可以解决一系列具有阶段性决策问题的场景，例如供应链优化、设备维护、项目管理等，具备很强的通用性。随着智能制造和工业 4.0 的兴起，企业可以利用该模型进一步自动化决策过程，结合传感器数据、机器学习和实时反馈，通过数据驱动的方式实时优化检测和生产策略，从而提高生产线的智能化和柔性化水平。

参考文献

[1] 孙红平. 基于本量利分析法的 W 化工公司生产决策研究[D]. [出版地不详]: 东南大学, 2016.

[2] 郝博, 傅士栗, 王建新, 等. 基于改进遗传算法零件加工工序优化研究[J]. 机床与液压, 2022, 50(22):18-25.

附录 A　文件列表

文件名	功能描述
q2.py	问题二程序代码
q31.m	问题三程序代码蒙特卡洛模型
q32.py	问题三程序代码动态转移模型
q41.py	问题四重算问题二程序代码
q42.py	问题四重算问题三程序代码
q43.py	问题四探究可视化代码

附录 B　问题三、四结果

2.1　问题三决策表

p1	p2	p3	p4	p5	p6	p7	p8	s1d	s1t	s2d	s2t	s3d	s3t	fd	ft	**profit**	s1_p	s2_p	s3_p
1	1	1	1	1	1	1	1	0	/	0	/	0	/	1	1	**66.09**	33.33	36.89	33.56
1	1	0	1	1	1	1	1	0	/	0	/	0	/	1	1	**65.58**	33.33	36.89	33.56
1	1	1	1	1	0	1	1	0	/	0	/	0	/	1	1	**65.58**	36.89	33.33	33.56
1	1	1	1	1	1	1	0	0	/	0	/	0	/	1	1	**65.58**	36.89	36.89	30.00
1	1	1	1	1	1	1	0	0	/	0	/	1	1	1	1	**65.08**	36.89	36.89	38.22
1	1	0	1	1	1	1	1	1	1	0	/	0	/	1	1	**65.08**	41.56	36.89	33.56
1	1	1	1	1	0	1	1	0	/	1	1	0	/	1	1	**65.08**	36.89	41.56	33.56
1	1	0	1	1	0	1	1	1	1	0	/	0	/	1	1	**64.98**	41.56	33.33	33.56
1	1	0	1	1	1	1	0	0	/	0	/	1	1	1	1	**64.98**	33.33	36.89	38.22
1	1	0	1	1	1	1	0	1	1	0	/	0	/	1	1	**64.98**	41.56	36.89	30.00
1	1	1	1	1	0	1	0	0	0	0	/	1	1	1	1	**64.98**	36.89	33.33	38.22
1	1	1	1	1	0	1	0	0	0	1	1	0	/	1	1	**64.98**	36.89	41.56	30.00

在本表格中，p1 到 p8 代表零部件 1 至 8 是否被检测，s1d-s3d 代表半成品 1 至 3 是否被检测，s1t-s3t 代表半成品是否被拆解，fd 代表成品是否被检测，ft 代表成品是否被拆解。s1p-s3p 代表半成品 1 至 3 的预期成本。profit 代表该决策集的最终预期收益。

2.2 问题四决策表（重算问题三）

本决策表是在加入贝叶斯 balabala 后的表。验算出的概率对照正文。表格标题项与问题三相同。

p1	p2	p3	p4	p5	p6	p7	p8	s1d	s1t	s2d	s2t	s3d	s3t	fd	ft	**profit**	s1_p	s2_p	s3_p
1	1	0	1	1	1	1	1	1	1	0	/	0	/	1	1	**61.22**	43.96	37.45	34.44
1	1	1	1	1	1	1	1	0	0	0	/	0	/	1	1	**60.92**	37.03	37.45	34.44
1	1	0	1	1	0	1	1	1	1	0	/	0	/	1	1	**60.88**	43.96	33.72	34.44
1	0	0	1	1	1	1	1	1	1	0	/	0	/	1	1	**60.67**	44.51	37.45	34.44
1	1	0	1	1	1	1	0	1	1	0	/	0	/	1	1	**60.39**	43.96	37.45	30.34
1	0	0	1	1	0	1	1	1	1	0	/	0	/	1	1	**60.33**	44.51	33.72	34.44
1	1	1	1	1	1	1	0	0	/	0	/	1	1	1	1	**60.32**	37.03	37.45	39.86
1	1	1	1	1	1	1	1	1	1	0	/	0	/	1	1	**60.20**	44.98	37.45	34.44
1	1	0	1	1	0	1	0	1	1	0	/	1	1	1	1	**59.90**	43.96	33.72	39.86
1	1	1	1	1	0	1	1	/	0	/	/	0	/	1	1	**59.88**	37.03	41.70	34.44
1	0	0	1	1	1	1	0	1	1	0	/	0	/	1	1	**59.84**	44.51	37.45	30.34
1	1	0	1	1	1	1	0	1	1	0	/	1	1	1	1	**59.75**	43.96	37.45	39.86

附录 C　代码

q2.py

```python
import numpy as np
import random


def monte_carlo_decision(simulations,p1_defect,p2_defect,
   product_defect):
    # 参数定义
    case = "case 6"
    price_1 = 4
    price_2 = 18
    sell_price = 56
    c1_detect = 2  # 零件1检测成本
    c2_detect = 3  # 零件2检测成本
```

```
    c_assembly = 6  # 装配成本
    c_product_detect = 3  # 成品检测成本
    c_replacement = 10  # 不合格品调换成本
    c_disassemble = 40  # 拆解不合格成品成本
    best_cost = float('inf')  # 初始化最小成本为正无穷大
    best_decision = None  # 用于存储最优决策组合

    # 四个决策变量的遍历（2^4 = 16 种组合）
    for detect_part1 in [True, False]:
        for detect_part2 in [True, False]:
            for detect_product in [True, False]:
                for disassemble_defective in [True, False]:
                    # 进行蒙特卡洛模拟，计算每种组合下的平均成本
                    total_cost = np.zeros(simulations)  # 存储每次模拟的总成本
                    for i in range(simulations):
                        cost = 0
                        # if detect_part1 and detect_part2:
                        #     p_product_defect = product_defect
                        # elif detect_part1==True and detect_part2==False:
                        #     p_product_defect = p1_defect + product_defect - product_defect * p1_defect
                        # elif detect_part1==False and detect_part2==True:
                        #     p_product_defect = p2_defect + product_defect - product_defect * p2_defect
                        # else:
                        #     p_product_defect = 1 - (1 - p1_defect) * (1 - p2_defect) * (1 - product_defect)

                        p1_is_defective = False
                        p2_is_defective = False
```

```
                        # 零件1的检测
                        if detect_part1:
                            cost += c1_detect
                            if random.random() < p1_defect:
                                cost += price_1
                                continue
                        else:
                            if random.random() < p1_defect:
                                p1_is_defective = True

                                # 零件2的检测
                        if detect_part2:
                            cost += c2_detect
                            if random.random() < p2_defect:
                                cost += price_2
                                continue
                        else:
                            if random.random() < p2_defect:
                                p2_is_defective = True

                                # 零部件装配

                        cost += price_1 + price_2

                                # 装配
                        cost += c_assembly

                        # 装配后的成品检测
                        if detect_product:
                            cost += c_product_detect
                            if random.random() <
    product_defect or p1_is_defective or p2_is_defective:
                                if disassemble_defective:
                                    cost += c_disassemble
```

```
                                cost -= price_1 + price_2
                            else:
                                pass
                        else:
                            cost -= sell_price
                    else:
                        if random.random() < product_defect or p1_is_defective or p2_is_defective:
                            cost += c_replacement
                            if disassemble_defective:
                                cost += c_disassemble
                                cost -= price_1 + price_2
                            else:
                                pass
                        else:
                            cost -= sell_price


                        # 累计总成本
                    total_cost[i] = cost

                    # 计算该决策组合下的平均成本
                avg_cost = np.mean(total_cost)
                print(
                    f'平均收益为：{-avg_cost:.2f} | 检测零件1: {detect_part1} | 检测零件2: {detect_part2} | 检测成品: {detect_product} | 拆解不合格成品: {disassemble_defective}')

                # 判断是否是最优方案
                if avg_cost < best_cost:
                    best_cost = avg_cost
                    best_decision = (detect_part1, detect_part2, detect_product, disassemble_defective)

                    # 输出最优决策结果
```

```
    print(f'此情况为：{case}')
    print(f'最优平均收益为：{-best_cost:.2f}')
    print(
        f'最优决策组合为：检测零件1：{best_decision[0]} | 检测零件2：{best_decision[1]} | 检测成品：{best_decision[2]} | 拆解不合格成品：{best_decision[3]}')
    return -best_cost, best_decision[0]


monte_carlo_decision(100000,0.05,0.05,0.05)
```

q31.m

```
monte_carlo_assembly(800)
%%
function monte_carlo_assembly(num_simulations)

    p_part_defects = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1];
    c_part_detects = [1, 1, 2, 1, 1, 2, 1, 2];
    p_semi_defects = [0.1, 0.1, 0.1];
    c_semi_detect = 4;
    p_final_defect = 0.1;
    c_final_detect = 6;
    c_assembly = [8, 8, 8];
    c_final_assembly = 8;
    c_replacement = 40;
    c_disassemble = 10;

    best_cost = inf;
    best_decision = [];

    for detect_part = dec2bin(0:(2^8 - 1))' == '1'
        for detect_semi = dec2bin(0:(2^3 - 1))' == '1'
            for detect_final = [true, false]
                for disassemble_semi = dec2bin(0:(2^3 - 1))'
   == '1'
```

```
23                  for disassemble_final = [true, false]
24
25
26                      total_cost = zeros(1, num_simulations)
    ;
27
28
29                      for i = 1:num_simulations
30                          cost = 0;
31                          part_defective = rand(1, 8) <
    p_part_defects;
32
33
34                          for j = 1:8
35                              if detect_part(j)
36                                  cost = cost +
    c_part_detects(j);
37                                  if part_defective(j)
38                                      continue;
39                                  end
40                              end
41                          end
42
43
44                          semi_defective = zeros(1, 3);
45                          semi_defective(1) = any(
    part_defective(1:3)) || rand < p_semi_defects(1);
46                          semi_defective(2) = any(
    part_defective(4:6)) || rand < p_semi_defects(2);
47                          semi_defective(3) = any(
    part_defective(7:8)) || rand < p_semi_defects(3);
48
49                          for j = 1:3
50                              cost = cost + c_assembly(j);
51                              if detect_semi(j)
```

```
                                    cost = cost +
    c_semi_detect;
                                    if semi_defective(j)
                                        if disassemble_semi(j)
                                            cost = cost +
    c_disassemble;
                                        end
                                        continue;
                                    end
                                end
                            end

                            cost = cost + c_final_assembly;
                            if detect_final
                                cost = cost + c_final_detect;
                                if rand < p_final_defect
                                    if disassemble_final
                                        cost = cost +
    c_disassemble;
                                    end
                                    continue;
                                end
                            else
                                if rand < p_final_defect
                                    cost = cost +
    c_replacement;
                                end
                            end

                            total_cost(i) = cost;
                        end

                        avg_cost = mean(total_cost);

                        if avg_cost < best_cost
```

```
                                best_cost = avg_cost;
                                best_decision = [detect_part',
    detect_semi', detect_final, disassemble_semi',
    disassemble_final];
                            end
                        end
                    end
                end
            end
     end

     disp(['×îÓÅ³É±¾: ', num2str(best_cost)]);
     disp(['×îÓÅ¼ö²ß: ', mat2str(best_decision)]);
end
```

q32.py

```
def calculation(in1_c, in1_p, in2_c, in2_p, in3_c, in3_p, c, d
    , q, t, is_dismantle, is_detect):
     small_p = 1 - (1 - q) * (1 - in1_p) * (1 - in2_p) * (1 -
    in3_p)
     E_undismantle = (c + d + in1_c + in2_c + in3_c) / (1 -
    small_p)
     E_dismantle = ((c + d + t * small_p) / (1 - small_p)) +
    in1_c + in2_c + in3_c
     E_undetect = c + in1_c + in2_c + in3_c

     if is_detect:
         if is_dismantle:
             return E_dismantle, 0
         else:
             return E_undismantle, 0
```

```
    else:
        return E_undetect, small_p


t1_c, t1_p = calculation(0,0,0,0,0,0,2,1,0.1,100,0,t1)
t2_c, t2_p = calculation(0,0,0,0,0,0,8,1,0.1,100,0,t2)
t3_c, t3_p = calculation(0,0,0,0,0,0,12,2,0.1,100,0,t3)
t4_c, t4_p = calculation(0,0,0,0,0,0,2,1,0.1,100,0,t4)
t5_c, t5_p = calculation(0,0,0,0,0,0,8,1,0.1,100,0,t5)
t6_c, t6_p = calculation(0,0,0,0,0,0,12,2,0.1,100,0,t6)
t7_c, t7_p = calculation(0,0,0,0,0,0,8,1,0.1,100,0,t7)
t8_c, t8_p = calculation(0,0,0,0,0,0,12,2,0.1,100,0,t8)
t9_c, t9_p = calculation(t1_c, t1_p, t2_c, t2_p, t3_c, t3_p,
   8, 4, 0.1, 6, t9_t, t9_d)
t10_c, t10_p = calculation(t4_c, t4_p, t5_c, t5_p, t6_c, t6_p,
    8, 4, 0.1, 6, t10_t, t10_d)
t11_c, t11_p = calculation(t7_c, t7_p, t8_c, t8_p, 0, 0, 8, 4,
    0.1, 6, t11_t, t11_d)
t12_c, t12_p = calculation(t9_c, t9_p, t10_c, t10_p, t11_c,
   t11_p, 8, 6, 0.1, 10, t12_t, t12_d)
e_final = (40* t12_p + t12_c) / (1 - t12_p)
profit = 200 - e_final
print(t1, t2, t3, t4, t5, t6, t7, t8, t9_d, t9_t, t10_d, t10_t
   , t11_d, t11_t, t12_d, t12_t, profit, t9_c,t10_c,t11_c)
```

q41.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta


def monte_carlo_decision(simulations):
    # 参数定义
    alpha_1, beta_1 = 2000, 8000  # 零件1的先验Beta分布参数
    alpha_2, beta_2 = 2000, 8000  # 零件2的先验Beta分布参数
```

```
    n_1, k_1 = 10000, 2000  # 零件1的检测样本量和不合格品数量
    n_2, k_2 = 10000, 2000  # 零件2的检测样本量和不合格品数量

    # 贝叶斯更新后的次品率分布参数
    alpha_1_updated = alpha_1 + k_1
    beta_1_updated = beta_1 + n_1 - k_1
    alpha_2_updated = alpha_2 + k_2
    beta_2_updated = beta_2 + n_2 - k_2

    # 抽样估计更新后的次品率
    p1_defect = beta.rvs(alpha_1_updated, beta_1_updated, size=simulations)
    p2_defect = beta.rvs(alpha_2_updated, beta_2_updated, size=simulations)

    # 成品次品率的初始贝塔分布参数
    alpha_m, beta_m = 2000, 8000  # 成品次品率先验分布参数
    n_m, k_m = 10000, 2000  # 成品检测样本量和不合格品数量

    # 贝叶斯更新后的成品次品率分布参数
    alpha_m_updated = alpha_m + k_m
    beta_m_updated = beta_m + n_m - k_m

    # 抽样估计更新后的成品次品率
    p_product_defect_bayes = beta.rvs(alpha_m_updated, beta_m_updated, size=simulations)

    # 成本
    c1_detect, c2_detect = 1, 1  # 零件1和零件2的检测成本
    c_assembly, c_product_detect = 6, 2  # 装配成本和成品检测成本
    c_replacement, c_disassemble = 30, 5  # 调换成本和拆解成本

    best_cost = float('inf')  # 初始化最小成本为无穷大
    best_decision = None  # 用于存储最优决策组合
```

```
    results = []

    # 四个决策变量的遍历（2^4 = 16 种组合）
    for detect_part1 in [True, False]:
        for detect_part2 in [True, False]:
            for detect_product in [True, False]:
                for disassemble_defective in [True, False]:
                    total_cost = np.zeros(simulations)  # 存储每次模拟的总成本

                    for i in range(simulations):
                        # 动态次品率
                        p1_def = p1_defect[i]
                        p2_def = p2_defect[i]
                        p_product_def_bayes_sample = p_product_defect_bayes[i]

                        # 成品的次品率取决于零件检测决策和贝叶斯更新的成品次品率
                        if detect_part1 and not detect_part2:
                            p_product_defect = p2_def + (1 - p2_def) * p_product_def_bayes_sample
                        elif not detect_part1 and detect_part2:
                            p_product_defect = p1_def + (1 - p1_def) * p_product_def_bayes_sample
                        elif detect_part1 and detect_part2:
                            p_product_defect = p_product_def_bayes_sample
                        else:
                            p_product_defect = 1 - (1 - p1_def) * (1 - p2_def) * (1 - p_product_def_bayes_sample)

                        # 初始化成本
                        cost = 0
```

```
68
69                      # 零件1检测
70                      if detect_part1:
71                          if np.random.rand() < p1_def:  # 零件1次品
72                              cost += c1_detect  # 检测成本，丢弃次品
73                              continue  # 零件1不合格，跳过装配
74                          else:
75                              cost += c1_detect  # 检测成本
76
77                      # 零件2检测
78                      if detect_part2:
79                          if np.random.rand() < p2_def:  # 零件2次品
80                              cost += c2_detect  # 检测成本，丢弃次品
81                              continue  # 零件2不合格，跳过装配
82                          else:
83                              cost += c2_detect  # 检测成本
84
85                      # 装配成本
86                      cost += c_assembly
87
88                      # 装配后的成品检测
89                      if detect_product:
90                          if np.random.rand() < p_product_defect:  # 成品不合格
91                              cost += c_product_detect  # 成品检测成本
92                              if disassemble_defective:  # 拆解成品
93                                  cost += c_disassemble  #
```

```
    拆解成本
94                          else:
95                              cost += c_product_detect  # 成
    品检测成本
96                      else:
97                          if np.random.rand() <
    p_product_defect:  # 未检测成品次品
98                              cost += c_replacement  # 调换
    损失
99
100                     # 累计总成本
101                     total_cost[i] = cost
102
103                 # 计算该决策组合下的平均成本
104                 avg_cost = np.mean(total_cost)
105                 results.append(avg_cost)
106
107                 print(f'平均成本为: {avg_cost:.2f} | 检测
    零件1: {detect_part1} | 检测零件2: {detect_part2} '
108                       f'| 检测成品: {detect_product} | 拆
    解不合格成品: {disassemble_defective}')
109
110                 # 判断是否是最优方案
111                 if avg_cost < best_cost:
112                     best_cost = avg_cost
113                     best_decision = [detect_part1,
    detect_part2, detect_product, disassemble_defective]
114
115  # 绘制柱状图
116  plt.bar(range(16), results)
117  plt.ylabel("average cost")
118  plt.title("Average cost comparison of 16 decision
    combinations")
119  plt.show()
120
```

```
    # 输出最优决策结果
    print(f'最优平均成本为：{best_cost:.2f}')
    print(f'最优决策组合为：检测零件1：{best_decision[0]} | 检测零件2：{best_decision[1]} | '
          f'检测成品：{best_decision[2]} | 拆解不合格成品：{best_decision[3]}')


# 调用函数，进行模拟
monte_carlo_decision(10000)
```

q42.py

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

values = np.array(resultbc)
flags = np.array(resultde)



# 获取True和False的索引
true_indices = np.where(flags)[0]
false_indices = np.where(~flags)[0]

# 数据准备
x_true = true_indices.reshape(-1, 1)
y_true = values[true_indices]
x_false = false_indices.reshape(-1, 1)
y_false = values[false_indices]

# 创建线性回归模型
model_true = LinearRegression().fit(x_true, y_true)
model_false = LinearRegression().fit(x_false, y_false)

```

```
# 生成回归线的预测值，扩展到更大的范围
x_min, x_max = -10, len(values) + 10  # 设定回归线的扩展范围
x_range = np.arange(x_min, x_max).reshape(-1, 1)
y_true_pred = model_true.predict(x_range)
y_false_pred = model_false.predict(x_range)

# 创建一个图形和子图
plt.figure(figsize=(12, 6))

# 绘制数值的折线图
# plt.plot(values, marker='o', linestyle='-', color='b', label='Values')

# 绘制True的点和回归线
plt.scatter(true_indices, values[true_indices], color='g', label='True Points')
plt.plot(x_range, model_true.predict(x_range), color='g', linestyle='--', label='True Regression Line')

# 绘制False的点和回归线
plt.scatter(false_indices, values[false_indices], color='r', label='False Points')
plt.plot(x_range, model_false.predict(x_range), color='r', linestyle='--', label='False Regression Line')

# 添加标题和标签
plt.title('数值与布尔值的回归线示例')
plt.xlabel('索引')
plt.ylabel('数值')

# 设置纵坐标范围
# plt.ylim(0, 20)

# 添加图例
plt.legend()
```

```
54
55 # 显示网格线
56 plt.grid(True)
57
58 # 显示图形
59 plt.show()
```

q43.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.stats import beta
4
5  def monte_carlo_decision_3d(simulations):
6      # 参数定义
7      alpha_1, beta_1 = 2000, 8000  # 零件1的先验Beta分布参数
8      alpha_2, beta_2 = 2000, 8000   # 零件2的先验Beta分布参数
9      n_1, k_1 = 10000, 2000  # 零件1的检测样本量和不合格品数量
10     n_2, k_2 = 10000, 2000  # 零件2的检测样本量和不合格品数量
11
12     # 贝叶斯更新后的次品率分布参数
13     alpha_1_updated = alpha_1 + k_1
14     beta_1_updated = beta_1 + n_1 - k_1
15     alpha_2_updated = alpha_2 + k_2
16     beta_2_updated = beta_2 + n_2 - k_2
17
18     # 抽样估计更新后的次品率
19     p1_defect = beta.rvs(alpha_1_updated, beta_1_updated, size
   =simulations)
20     p2_defect = beta.rvs(alpha_2_updated, beta_2_updated, size
   =simulations)
21
22     # 成本
23     c1_detect = 1       # 零件1检测成本
24     c2_detect = 1       # 零件2检测成本
25     c_assembly = 6      # 装配成本
```

```
    c_product_detect = 2  # 成品检测成本
    c_replacement = 30  # 不合格品调换成本
    c_disassemble = 5  # 拆解不合格成品成本

    # 初始化结果存储矩阵
    k_range = np.arange(1000, 3001, 100)  # 成品检测中不合格数量 k_m 的范围
    costs_matrix = np.zeros((len(k_range), 16))  # 用于存储每个 k_m 下16种决策组合的成本

    total_product_samples = 10000  # 总成品检测数量固定为10,000

    # 遍历每个 k_m 值
    for k_index, k_m in enumerate(k_range):
        # 贝叶斯更新后的成品次品率分布参数
        alpha_m_updated = 2000 + k_m
        beta_m_updated = 10000 - k_m  # 固定总成品检测数为10000

        # 抽样估计更新后的成品次品率
        p_product_defect_bayes = np.random.beta(alpha_m_updated, beta_m_updated, [simulations])

        # 遍历决策变量的组合
        decision_index = 0  # 用于记录决策组合索引
        for detect_part1 in [True, False]:
            for detect_part2 in [True, False]:
                for detect_product in [True, False]:
                    for disassemble_defective in [True, False]:
                        decision_index += 1  # 更新决策组合索引
                        total_cost = np.zeros(simulations)  # 存储每次模拟的总成本
```

```
53
54                          # 进行蒙特卡洛模拟，计算每种组合下的平均成本
55                          for i in range(simulations):
56                              # 动态次品率
57                              p1_def = p1_defect[i]
58                              p2_def = p2_defect[i]
59
60                              # 使用贝叶斯更新的成品次品率
61                              p_product_def_bayes_sample = p_product_defect_bayes[i]
62
63                              # 成品的次品率取决于零件检测决策
64                              if detect_part1 and not detect_part2:
65                                  p_product_defect = p2_def + (1 - p2_def) * p_product_def_bayes_sample
66                              elif not detect_part1 and detect_part2:
67                                  p_product_defect = p1_def + (1 - p1_def) * p_product_def_bayes_sample
68                              elif detect_part1 and detect_part2:
69                                  p_product_defect = p_product_def_bayes_sample
70                              else:
71                                  p_product_defect = 1 - (1 - p1_def) * (1 - p2_def) * (1 - p_product_def_bayes_sample)
72
73                              # 初始化成本
74                              cost = 0
75
76                              # 零件1的检测
77                              if detect_part1:
78                                  if np.random.rand() < p1_def:
```

```
                # 如果零件1次品
79                                  cost += c1_detect  # 检测
    成本，丢弃次品
80                                  continue  # 零件1不合格，
    跳过装配
81                              else:
82                                  cost += c1_detect  # 检测
    成本
83
84                          # 零件2的检测
85                          if detect_part2:
86                              if np.random.rand() < p2_def:
                # 如果零件2次品
87                                  cost += c2_detect  # 检测
    成本，丢弃次品
88                                  continue  # 零件2不合格，
    跳过装配
89                              else:
90                                  cost += c2_detect  # 检测
    成本
91
92                          # 装配
93                          cost += c_assembly  # 装配成本
94
95                          # 装配后的成品检测
96                          if detect_product:
97                              if np.random.rand() <
    p_product_defect:  # 如果成品不合格
98                                  cost += c_product_detect
    # 成品检测成本
99                                  if disassemble_defective:
                # 拆解成品
100                                     cost += c_disassemble
                # 拆解成本
101                             else:
```

```
                                    cost += c_product_detect  # 成品检测成本
                            else:
                                if np.random.rand() < p_product_defect:  # 如果不检测，成品进入市场可能不合格
                                    cost += c_replacement  # 产生调换损失

                            # 累计总成本
                            total_cost[i] = cost

                        # 计算该决策组合下的平均成本
                        avg_cost = np.mean(total_cost)

                        # 存储到结果矩阵
                        costs_matrix[k_index, decision_index - 1] = avg_cost

    # 绘制三维图
    K_mesh, D_mesh = np.meshgrid(k_range, np.arange(1, 17))
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(K_mesh, D_mesh, costs_matrix.T, cmap='viridis')

    ax.set_xlabel('Number of nonconforming products')# 不合格产品数量
    ax.set_ylabel('Decision combination')# 决策组合
    ax.set_zlabel('Lowest average cost')# 最低平均成本
    ax.set_title('The lowest average cost change for different product test sample sizes')# 不同成品检测样本量下最低平均成本变化

    plt.show()

```

```
129 # 执行函数
130 monte_carlo_decision_3d(1000)
```