生产过程中的最优检测与拆解策略

摘要

随着全球化进程的推进和科学技术的不断进步，企业必须不断优化生产过程中的决策来适应市场。本文依据题目所给情形建立相关的数学模型，给出相应的决策方案。

对于问题一，首先我们判断本问题中抽样的过程符合二项分布，然后在考虑样本大小的情况下，采用正态分布对本题中抽样的过程进行拟合。利用正态分布对应的 z 检验，计算出不同信度下误差范围内允许的最小样本大小。并针对题目中所给的两种情形给出具体的结果。

对于问题二，我们利用博弈论的思想对生产流程中的决策进行分析，对 W_q 售出的合格成品的收入 W_q，总检测成本 T_{cd}，T_{cr} 退货成本进行计算，并引入拆解收入 W_r 的概念，求取出售产品获得的总收益。最后对所有决策组合进行遍历，找出最佳的决策方案。

对于问题三，我们对问题二中分析生产流程的思路进行推广，计算零件对总成本的贡献 T_{cc}、半成品对总成本的贡献 T_{hc}、成品对总成本的贡献 T_{fc}，求取生产产品所需的总成本。在决策规模较大的情况下，采用遗传算法对决策过程进行分析，设置合理的阈值，给出相应的决策方案。

对于第四问，我们利用伯努利试验在生成的样本数组中模拟次品率的抽样检测过程，求取通过计算样本的平均值来估计次品率。在此基础上，我们对问题二和问题三的代码进行修改，并得出在通过抽样检测方法得到次品率的情况下的相应方案。

最后，我们对模型进行了优缺点分析和推广。

关键字：遗传算法　博弈论　最优化　生产决策

一、 问题重述

1.1 背景资料

随着全球化进程的推进和科学技术的不断进步，市场竞争不断加剧，企业必须通过对产品生产过程的不断优化来维持竞争优势，满足消费者需求。在产品生产过程中，为了防止次品的出现、企业需要对产品生产中的环节进行检测。如果不进行检测，可能会导致大量次品流入市场，为公司带来调换损失和声誉影响，但如果对生产中的所有环节都进行检测，则可能导致公司支付过高的检测费用。因此，在产品的检测环节进行合理决策，对公司的盈利与发展具有十分重要的意义。

1.2 需要解决的问题

我们运用数理统计等思想建立数学模型，研究生产过程中的如下决策问题

（1）设计次数尽可能少的抽样检测方案，并针对题中所述的两种情形给出具体结果。

（2）已知两种零配件和成品的次品率，研究企业生产中的多个阶段，对各个阶段的检测、拆解等行为进行决策，并给出决策依据和相应的指标结果。

（3）对问题二进行扩充，研究在有 m 道工序、n 个零配件的条件下，已知零配件、半成品和成品的次品率，给出生产过程的决策方案。

（4）在零配件、半成品和成品的次品率均是通过抽样检测方法得到的情况下，重新完成问题 2 和问题 3.

二、 问题分析

2.1 问题一分析

题目要求为企业设计抽样检测方案并针对具体情形给出结果。由于本题考虑的是是否拒收零件，因此我们采用单边检验的方式。同时由于产品的样本量较大，采用正态分布的方式进行拟合，并利用 z 检验在允许的误差范围内推出最小的所需样本量。

2.2 问题二分析

题目要求在已知两种零配件和成品次品率的情况下给出生产过程中的决策。我们利用博弈论的思想，首先列举出所有的策略组合，利用题目中所给条件设计函数对所有策略进行收益计算，并据此输出最优策略。

2.3 问题三分析

题目要求在有 m 道工序、n 个零配件并已知零配件、半成品和成品的次品率的情况下，给出具体的决策方案。我们对问题二中的思想方法进行推广，并利用遗传算法对目标函数进行分析得到合理的决策方案。

2.4 问题四分析

题目要求在零配件、半成品和成品的次品率均是通过抽样检测方法得到的情况下重新完成问题 2 和问题 3。我们通过编写函数来模拟二项分布的抽样情形，对问题二和问题三的算法进行修改，得到在抽样检测条件下的结论。

三、 模型的假设

- 假设厂商拥有足量的零件、不存在零件短缺的情况。
- 假设产品的产量较大且不中止生产。
- 假设所有进入市场的次品都被消费者退回，不存在以次充好的情况。
- 假设成品拆解为半成品的过程和半成品拆解为零配件的过程是独立的。

四、 符号说明

符号	意义
W_q	售出的合格成品的收入
T_{cd}	总检测成本
W_r	拆解收入
T_{cr}	退货成本
T_{cc}	零件对总成本的贡献
T_{hc}	半成品对总成本的贡献
T_{fc}	成品对总成本的贡献

五、 模型的建立与求解

5.1 问题一模型的建立及求解

5.1.1 模型的准备

根据问题描述，本题的目标是根据一批零件的次品率是否超过厂商声称的标称值 N_0 来决定是否接收这批零件，即验证:

$$H_0 : N \leq N_0 \tag{1}$$

因此，我们采用单边检验的方式对零件的次品率进行检验。在 n 次独立重复的伯努利实验中，设每次实验中事件只有两种可能的结果，而且这两种结果是否发生是相互对立的，并且相互独立，则将其称为二项分布。本题中抽样得到的每个零配件，只存在两种情况，即合格与不合格，因此本题的模型符合二项分布。

由统计理论可知，当二项分布的实验次数 n 非常大时，二项分布的形状近似于正态分布。为简化计算，我们采用正态分布的方式对零件的抽样过程进行拟合。对正态分布 $N\left(\mu, \sigma^2\right)$, 其分布函数为:

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{x} e^{-\frac{(t-a)^2}{2\sigma^2}} \, \mathrm{d}i \tag{2}$$

5.1.2 检验方案的确定

确定本题是用正态分布的方式进行拟合后，我们利用 Z 检验方法对样品进行检验。Z 检验是基于标准正态分布的理论的，一般用于较大样本平均值差异性检验的方法。设误差范围 $d = p_i - N_0$（其中 p_i 是实际的次品率，N_0 是标称次品率），则检验统计量 z 的计算公式如下:

$$z = \frac{p' - N_0}{\sqrt{\frac{N_0(1-N_0)}{n}}} \tag{3}$$

对这一公式进行变换，可得:

$$n = \frac{z^2 N_0(1 - N_0)}{d^2} \tag{4}$$

则 n 即为我们所要求取的最小样本量。

5.1.3 针对两种情形的具体结果

针对题目中给出的两种情形，我们通过上文中所得出的结论进行分析，记原假设为:$H_0 : N \leq N_0$，备择假设为：$H_1 : N > N_0$。下图是在 90% 的信度下，抽取不同样本

数量对应的接受原假设的概率：

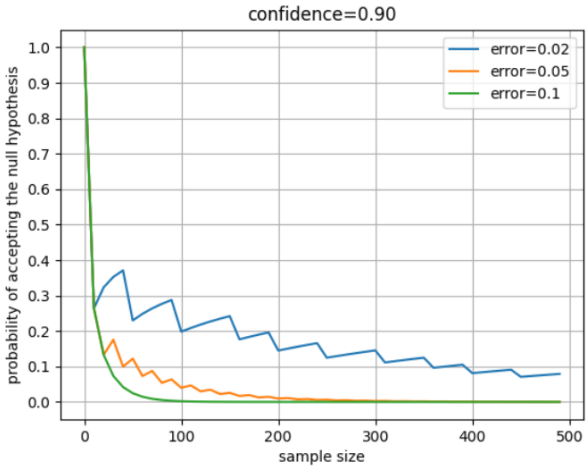

图 1　90% 的信度下不同样本量接受原假设的概率

在显著性水平 α=0.05 且允许误差范围为 0.05 的情况下，第一种情形的具体结果为 98，第二种情形的具体结果为 60。

5.2 问题二模型的建立及求解

5.2.1 模型的准备

对于问题二，我们可以采用博弈论的思想对问题进行分析。在博弈论中，每个决策（如检测、拆解等）都可以视为一个策略选择，并且企业在不同阶段作出的检测相互影响。通过博弈论，我们可以找到最优的策略组合。

我们可以将这个问题分解为多个子博弈，每个子博弈对应一个决策阶段：

1. 零配件检测的子博弈

策略：

- 对零配件 1 和零配件 2 全部或部分进行检测。
- 对零配件 1 和零配件 2 都不进行检测，直接进入装配环节。

收益和损失：

- 检测带来了检测成本，但可以减少进入装配环节的次品数量，进而减少成品次品率。
- 不检测节省了检测成本，但成品的次品率会提升，可能导致企业需要支付额外的调换损失。

2. 成品检测的子博弈

策略：

- 检测成品。
- 不检测成品，将未检测的成品直接投入市场。

收益和损失：

- 检测成品增加了检测成本，但可以避免次品进入市场，减免了调换损失。
- 不检测成品节省了检测成本，但次品会流入市场，企业需要为退回的产品支付额外的调换损失。

3. 不合格成品拆解的子博弈

策略：

- 对检测出来的不合格成品或用户退回的不合格成品进行拆解，回收零配件。
- 不进行拆解，直接丢弃不合格成品。

收益和损失：

- 拆解不合格成品需要支付拆解成本，但可以回收零配件，进而节省额外购买零配件的成本。
- 直接丢弃不合格成品无需支付拆解成本，但会损失零配件的回收价值。

综合上文的分析过程，我们设置了四个决策变量，分别是零配件 1 检测决策变量 x_1（为 1 时则进行检测，不为 1 时则不进行检测），零配件 2 检测决策变量 x_2，成品检测决策变量 y，成品拆解变量 z。

5.2.2 产品总收益的影响因素

1. 售出的合格成品的收入 W_q

要计算售出的合格成品的收入，即计算售出的合格成品的数量乘以合格成品的售价。首先，我们要由如下公式计算检测零配件 1 和零配件 2 对成品次品率的影响 F_1 和 F_2：

$$F_1 = p_1 \times (1 - x_1) \tag{5}$$

$$F_2 = p_2 \times (1 - x_2) \tag{6}$$

其中 p_1 和 p_2 分别是零配件 1 和零配件 2 的次品率。

对于成品的最终次品率，我们要考虑到成品组装过程中的次品率和来自零配件的次品率。产品的最终次品率即为 1-产品的最终成品率。依据 F_1 和 F_2，我们可以由如下公式计算出成品的最终次品率 F：

$$F = 1 - (1 - P)(1 - F_1)(1 - F_2) \tag{7}$$

其中 P 为成品组装过程中的的次品率

记进入市场的次品占所有进入市场的成品的比例为 P'。则：

$$P' = F \times (1 - y) \tag{8}$$

据此可以计算出 W_q:

$$W_q = V_p \times (1 - F) \tag{9}$$

2. 总检测成本 T_{cd}

对零配件或成品进行检测的过程中，都需要花费额外的检测成本。这一成本受对应的检测决策变量的影响，计算公式如下:

$$T_{cd} = C_{d1} \times x_1 + C_{d2} \times x_2 + C_{d3} \times x_3 \tag{10}$$

式中，C_{di} 表示零配件 i 的检测成本。

3. 拆解收入 W_r

对于检测出的不合格成品以及用户退回的不合格成品，企业可以在拆解和丢弃中进行决策。由于拆解不会对零配件造成损坏，在拆解的过程中企业可能会成功回收一部分零配件。因此，我们引入拆解收入的概念，即企业在回收零配件的过程中所节省的采购成本。其计算公式如下:

$$W_r = (\max(x_1, 1 - p_1)\, m_1 + \max(x_2, 1 - p_2)\, m_2 - C_s) \times z \times F \tag{11}$$

式中，m_i 表示零配件 i 的价格，C_s 表示拆解成本。

4. 退货成本 T_{cr}

企业在调换用户退回的不合格成品时，需要支付一定的退货成本。

$$T_{cr} = C_r \times P' \tag{12}$$

式中，C_r 表示成本。

5.2.3 最优策略模型

1. 目标函数的确定

由上文确定的影响因素，我们可以确定目标函数。

$$T_p = W_q - T_{cd} - W_r - T_{cr} - (m_1 + m_2) - C_m \tag{13}$$

式中 C_m 表示组装成本

2. 模型的求解

要求解本模型，我们对 16 种策略组合的总收益 T_p 分别进行计算并比较，找出总收益的最大值与其对应的策略组合。

对本题中所给的具体情形，我们的运行结果如下表:

情况	是否检测零件一	是否检测零件二	是否检测成品	不合格成品是否拆解	最终收益
1	是	是	否	否	13.8
2	是	是	否	是	18.5
3	是	是	否	是	14.0
4	是	是	否	是	16.1
5	否	是	否	是	17.6
6	是	是	否	否	19.7

表 1　运行结果

经检验，我们的模型能够在次品率，检测成本，拆解费用等变换的情况下适时地给出最佳决策方案。

5.3 问题三模型的建立及求解

5.3.1 模型的准备

本题是对问题二的进一步推广。问题二中，零配件被直接组装为成品。而问题三引入了多道工序，零件会被先组装为半成品，再被组装为成品，其决策的规模相比问题二大了许多，难以再用列举策略组合的方式进行求解。但是，问题三中分析产品成本的整体思路与问题二是类似的，我们在此基础上引入遗传算法的思想对模型进行优化，进而推导出最优的决策方案。

5.3.2 产品总成本的影响因素

1. 零件对总成本的贡献 T_{cc}

产品生产中的成本很大一部分来自于零件总成本，零件总成本的值即所有零件的购买价格加上检测成本，公式如下：

$$T_{cc} = \sum_{i=1}^{8} (m_i + x_i C_{di}) \tag{14}$$

式中 m_i 表示零配件 i 的购买单价，x_i 表示零配件 i 是否检测的决策变量（为 0 表示不检测，为 1 表示检测），C_{di} 表示零配件 i 的检测成本。

2. 半成品对总成本的贡献 T_{hc}

半成品的成本也会对成品的总成本造成影响。半成品总成本受半成品次品率的影响，而在多道工序的情况下，半成品的最终次品率又会受到零件次品率的影响，其计算公式如下：

$$F_{hi} = 1 - (1 - p_{hi}) \prod (1 - p_j) \tag{15}$$

式中，F_{hi} 表示半成品 i 在实际生产中的最终次品率，p_{hi} 表示半成品 i 在组装过程中的次品率，p_j 表示零件 j 的次品率。在本题的条件下，半成品 1 对应零件 1、2、3，半成品 2 对应零件 4、5、6，半成品 3 对应零件 7、8。

由于对半成品进行检测会减少进入下一环节的次品数量，因此进入下一环节的次品比例 $F_{hi}{}'$ 如下：

$$F_{hi}{}' = F_{hi} \times (1 - x_{hi}) \tag{16}$$

式中 x_{hi} 表示半成品 i 是否进行检测的决策变量。

考虑到半成品的拆解过程会获取拆解收益，因此在计算半成品成品的过程中也要加上相应的拆解收益，计算公式如下：

$$W_{hi} = \sum \max(x_{hj}, 1 - p_j) \times F_{hi}{}' \tag{17}$$

其中 W_{hi} 是半成品 i 的拆解收益，x_{hj} 是半成品 j 是否检测的决策变量，p_j 是零配件 j 的次品率。

在考虑拆解收益和检测成本等影响因素后，可推知半成品总成本如下：

$$T_{hc} = \sum_{i=1}^{3} ((C_{hmi} + x_{hi} \times C_{hdi} + z_{hi} \times C_{hsi}) \times F_{hi}{}' - W_{hi}) \tag{18}$$

其中 C_{hmi} 是半成品 i 的组装成本，x_{hi} 是是否检测半成品 i 的决策变量，C_{hdi} 是半成品 i 的检测成本，z_{hi} 是是否拆解半成品 i 的决策变量，C_{hsi} 是半成品 i 的拆解成本。

3. 成品对总成本的贡献 T_{fc}

要计算成品对总成本的贡献，首先要根据半成品进入市场的次品比例计算出成品次品率 F：

$$F = 1 - (1 - P) \prod (1 - F_{hi}{}') \tag{19}$$

由此可以计算出进入市场的次品比例 F'

$$F' = F \times (1 - y) \tag{20}$$

使用下列公式计算产品的总成本

$$T_{fc} = C_{fm} + y \times C_{fd} + z \times C_{fs} + (C_r - W_f) \times F' \tag{21}$$

其中，C_{fm} 为成品的装配成本，C_{fd} 为成品的检测成本，C_{fs} 为成品的拆解成本，C_r 为成品的退换成本，W_f 为成品拆解为半成品的拆解收益，可由与计算半成品拆解为零配件的拆解收益相同的算法得到。

5.3.3 多阶段策略模型

由上文可知，模型的总成本即为零配件对总成本的贡献、半成品对总成本的贡献、成品对总成本的贡献之和，这便是我们要求取的目标函数：

$$T = T_{cc} + T_{hc} + T_{fc} \tag{22}$$

问题三中的决策变量达到 16 个，其样本空间的大小为 2^{16}，难以继续采用第二问的枚举法进行求解。因此，考虑到本题的目标是求取最优的决策。我们利用遗传算法对问题进行求解。

遗传算法是一种自适应全局优化搜索算法，使用二进制遗传编码表示等位基因和个体空间，其繁殖分为交叉与变异两个独立的步骤进行。在算法执行的过程中实现个体评价、种群进化、母体和子代的选择等。这一算法利用了生物遗传进化的思想，具有自组织、自适应和智能性且基本思想易于理解。

利用遗传算法对目标函数进行分析，将决策变量的范围 bounds 设置为 0-1，求取的结果如下：

- 零配件 1 是否检测：否
- 零配件 2 是否检测：否
- 零配件 3 是否检测：否
- 零配件 4 是否检测：否
- 零配件 5 是否检测：否
- 零配件 6 是否检测：否
- 零配件 7 是否检测：否
- 零配件 8 是否检测：否
- 半成品 1 是否检测：是
- 半成品 1 是否检测：是
- 半成品 1 是否检测：是
- 半成品 1 是否拆解：否
- 半成品 2 是否拆解：否
- 半成品 3 是否拆解：是
- 成品是否检测：是
- 成品是否拆解：否

遗传算法所取的决策变量为 0-1 之间的浮点数，我们对决策的判定设置的阈值为 0.5，即超过 0.5 则将决策变量设置为 1，小于 0.5 则将决策变量设置为 0。

5.4 问题四模型的建立及求解

5.4.1 模型的准备

问题二和问题三中零配件、半成品和成品的次品率均是题目中所给的确定值，而本题中的次品率是由抽样检测的方法得到的。因此我们使用伯努利试验来模拟次品率的抽样检测过程，模拟过程可以总结为以下几步：

1. 生成样本

假设某个零配件或成品的真实次品率为 p。我们从生产线中抽取一定数量的样本（例如 100 个）来检测这些产品的次品情况。每个样本是一个伯努利试验的结果，可能为合格品或者次品。在代码中，使用 python 的 binomial 函数生成一个二项分布的样本数组。

2. 估计次品率

从生成的样本中，我们可以计算出次品率的估计值。通过计算样本中缺陷产品的比例，得到估计次品率 e：

$$e = \frac{n}{N} \tag{23}$$

其中：n 代表样本中缺陷产品的数量，N 代表样本总数。

我们通过这一步模拟了在现实中通过抽样检测得到次品率估计的过程。通过计算样本的平均值来估计次品率。在样本中，次品用 1 表示，非次品用 0 表示，因此平均值即为次品率的估计值。在模型中，我们通过大量抽样来使得次品率估计更加准确。

3. 构建收益函数

利用问题二和问题三中的目标函数对本题中的收益函数进行构建。

4. 策略枚举与求解

采用与问题二三中相同的方法对问题进行求解。

5.4.2 次品率的模拟与优化

1. 控制误差

为了使估计的次品率接近真实次品率，我们在每次模拟中设置了一个最大误差范围 ϵ，以确保抽样结果的合理性。如果某次模拟中的估计次品率与真实次品率的差距超过设定的误差范围，我们将重新进行抽样，直到误差符合要求为止。这样可以保证模拟出的次品率不会偏离真实情况太远。

2. 大量模拟

由于每次抽样检测的结果具有随机性，为了减小偶然因素对收益计算的影响，我们对每种策略组合进行了大量（1000 次以上）模拟。通过多次抽样和收益计算，得到每种策略的平均收益，以此来评估策略的长期收益表现。

5.4.3 运行结果

对问题二进行重做的运行结果如下：

情况	是否检测零件一	是否检测零件二	是否检测成品	不合格成品是否拆解	最终收益
1	否	否	否	是	23.3
2	否	否	否	是	18.0
3	否	否	否	是	21.0
4	否	否	是	是	17.2
5	否	否	否	是	22.7
6	否	否	否	是	24.1

表 2　运行结果

对问题三进行重做的运行结果如下：

- 零配件 1 是否检测：否
- 零配件 2 是否检测：否
- 零配件 3 是否检测：否
- 零配件 4 是否检测：否
- 零配件 5 是否检测：否
- 零配件 6 是否检测：否
- 零配件 7 是否检测：否
- 零配件 8 是否检测：否
- 半成品 1 是否检测：是
- 半成品 1 是否检测：是
- 半成品 1 是否检测：是
- 半成品 1 是否拆解：是
- 半成品 2 是否拆解：否
- 半成品 3 是否拆解：是
- 成品是否检测：是
- 成品是否拆解：否

六、 模型的评价

6.1 模型的优点

1. 本文提出拆解收益公式来解决次品的零件再利用问题，将次品的零件再利用减少的亏损整合至所有成品的零件采购成本，是本文模型的创新与独到之处。

2. 本文采用的模型较为简单，提出的假设适当简化了问题，使得模型易于理解和实现，同时也提高了计算效率

6.2 模型的缺点

1. 本文对问题一的求解中，对范第一类与第二类错误的情况考虑尚不周全，可能导致结果的准确性受到影响。

2. 现实情况中，企业可能根据产品的收益相对调整该产品项目的预算，从而导致采购计划有变，样本零件的数量减少，此时结果可能存在误差。

七、 模型的推广

1. 本文提出的模型不仅适用于制造业中的次品零件再利用问题，还可以推广到其他行业，如电子产品、医疗器械等领域，解决类似的资源优化问题。

2. 模型可以应用于供应链管理中，帮助企业优化采购计划，减少库存成本，提高资源利用效率。

参考文献

[1] 葛继科, 邱玉辉, 吴春明, 等. 遗传算法研究综述 [J]. 计算机应用研究,2008,(10):2911-2916.

[2] 陈益飞, 杜建丽, 魏卓亚, 等. 基于遗传算法的生产企业原材料订购方案 [J]. 信息记录材料,2024,25(02):151-153+157.DOI:10.16009/j.cnki.cn13-1295/tq.2024.02.032.

[3] 张建英. 博弈论的发展及其在现实中的应用 [J]. 理论探索,2005,(02):36-37.

[4] 张宏伟. 数学建模中的动态规划问题 [D]. 东北师范大学,2008.

附录 A　问题一

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

standard_rate = 0.1
reject_confidence = 0.95
accept_confidence = 0.90
error = 0.05

alpha_reject = 1 - reject_confidence
z_reject = stats.norm.ppf(alpha_reject)
alpha_accept = 1 - accept_confidence
z_accept = stats.norm.ppf(1 - alpha_accept)

def required_sample_size(standard_rate, delta, z):
p = 1 - standard_rate
return z**2*standard_rate*p/(delta**2)


required_sample_size_reject = required_sample_size(standard_rate, error, z_reject)
required_sample_size_accept = required_sample_size(standard_rate, error, z_accept)
print(required_sample_size_reject, required_sample_size_accept, z_reject, z_accept)

simple_sizes = np.arange(0, 500, 10)
errors = [0.02, 0.05, 0.1]


for error in errors:
probability = [1 - binom.cdf(np.ceil((standard_rate + error)*simple_size)-1, simple_size,
    standard_rate) for simple_size in simple_sizes]
plt.plot(simple_sizes, probability, label=f"error={error}")
plt.title(f"confidence={accept_confidence:.2f}")
plt.xlabel("sample size")
plt.ylabel("probability of accepting the null hypothesis")
yticks = np.arange(0, 1.1, 0.1)
plt.yticks(yticks)
plt.grid(True)
plt.legend()
plt.show()

# errors = np.arange(0.01, 1.1, 0.05)
# plt.plot(errors, [required_sample_size(standard_rate, error, z_reject) for error in errors],
    label="Reject Null Hypothesis")
```

```
# plt.show()
```

附录 B　问题二

```
import math

import numpy as np
import itertools

# 参数设置
V_p = 56 # 成品的市场售价
check_1_cost = 2 # 每次检测的成本
check_2_cost = 3
check_cost = 3
buy_cost1 = 4 # 购买配件的成本
buy_cost2 = 18
C_s = 40 # 拆解成本
C_r = 10 # 退货和调换成本

# 零配件和成品的次品率
p1 = 0.05 # 零配件1的次品率
p2 = 0.05 # 零配件2的次品率
p_c = 0.05 # 成品的次品率

# 生成不同的策略组合
# x1, x2 分别代表是否检测零配件1和2;  y 代表是否检测成品;  z 代表是否拆解不合格成品
strategies = list(itertools.product([0, 1], repeat=4)) # 共有 2^4 = 16 种策略组合



# 收益计算函数
def calculate_profit(strategy, V_p, C_d1, C_d2,C_d3,C_s, C_r, p1, p2, p_c):
x1, x2, y, z = strategy # x1检测零配件1, x2检测零配件2, y检测成品, z拆解成品

# 检测零配件后的成品次品率 (如果检测, 则次品率降低)
final_p1 = p1 * (1 - x1)
final_p2 = p2 * (1 - x2)

# 最终成品的次品率 = 成品的自然次品率 + 来自零配件的次品率
final_p_c = 1 - (1 - p_c) * (1 - final_p1) * (1 - final_p2)

# 若检测成品, 则减少次品进入市场
market_p_c = final_p_c * (1 - y)

# 检测带来的成本, 拆解带来的收益
detect_cost = C_d1 * x1 + C_d2*x2 + C_d3*y
```

```
rework_cost = (max(x1, 1-p1)*buy_cost1 + max(x2, 1-p2)*buy_cost2 - C_s) * z * final_p_c

# 成品售出后的退货成本
return_cost = C_r * market_p_c

# 总收益 = 售出的合格成品的收入 - 检测成本 + 拆解收益 - 退货损失 - 购买零件成本 - 组装成品成本
profit = V_p * (1 - final_p_c) - detect_cost + rework_cost - return_cost - buy_cost1 -
    buy_cost2 - 6

return profit
# 对所有策略进行收益计算
profits = [calculate_profit(strategy, V_p, check_1_cost, check_2_cost, check_cost, C_s, C_r,
    p1, p2, p_c) for strategy in strategies]

# 输出最优策略
optimal_strategy_idx = np.argmax(profits)
optimal_strategy = strategies[optimal_strategy_idx]
optimal_profit = profits[optimal_strategy_idx]

strategy_labels = ["检测零配件1", "检测零配件2", "检测成品", "拆解成品"]

print("最优策略组合: ")
for i, label in enumerate(strategy_labels):
print(f"{label}: {'是' if optimal_strategy[i] == 1 else '否'}")

print(f"\n最优策略对应的收益: {optimal_profit}")
```

附录 C　问题三

```
import numpy as np
from scipy.optimize import differential_evolution

# 零配件信息
component_info = {
1: {'defect_rate': 0.1, 'price': 2, 'detection_cost': 1},
2: {'defect_rate': 0.1, 'price': 8, 'detection_cost': 1},
3: {'defect_rate': 0.1, 'price': 12, 'detection_cost': 2},
4: {'defect_rate': 0.1, 'price': 2, 'detection_cost': 1},
5: {'defect_rate': 0.1, 'price': 8, 'detection_cost': 1},
6: {'defect_rate': 0.1, 'price': 12, 'detection_cost': 2},
7: {'defect_rate': 0.1, 'price': 8, 'detection_cost': 1},
8: {'defect_rate': 0.1, 'price': 12, 'detection_cost': 2},
}

# 半成品和成品信息
```

```
intermediate_info = {
1: {'defect_rate': 0.1, 'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
2: {'defect_rate': 0.1, 'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
3: {'defect_rate': 0.1, 'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
}

final_product_info = {'defect_rate': 0.1, 'assembly_cost': 8, 'detection_cost': 4,
    'disassembly_cost': 10, 'market_price': 200, 'return_cost': 40}

# 决策变量数量
n_components = 8
n_intermediates = 3
n_final_products = 1 # 需要检测和拆解决策

# 总决策变量数量
decision_vars = n_components + n_intermediates + n_intermediates + n_final_products +
    n_final_products

# 目标函数
def objective(x):
if len(x) != decision_vars:
raise ValueError(f"决策变量数量错误，期望：{decision_vars}，实际：{len(x)}")

# 解码决策变量
x_d = x[:n_components] # 零配件检测决策
x_d_f = x[n_components:n_components+n_intermediates] # 半成品检测决策
x_s = x[n_components+n_intermediates:2*n_components+n_intermediates] # 半成品拆解决策
x_d_final = x[n_components+2*n_intermediates] # 成品检测决策
x_s_final = x[n_components+2*n_intermediates+1] # 成品拆解决策

# 计算零配件的总成本
component_cost = sum(
(component_info[i+1]['price'] + x_d[i] * component_info[i+1]['detection_cost'])
for i in range(n_components)
)

# 计算半成品的次品率
p_s = 1 - np.prod([(1 - component_info[i+1]['defect_rate']) for i in [0, 1,
    2]])*(1-intermediate_info[1]['defect_rate'])
p_s2 = 1 - np.prod([(1 - component_info[i+1]['defect_rate']) for i in [3, 4,
    5]])*(1-intermediate_info[2]['defect_rate'])
p_s3 = 1 - np.prod([(1 - component_info[i+1]['defect_rate']) for i in [6,
    7]])*(1-intermediate_info[3]['defect_rate'])

# 如果检测半成品，则减少进入市场的次品比例
p_s_d = p_s * (1 - x_d_f[0])
p_s2_d = p_s2 * (1 - x_d_f[1])
```

```
p_s3_d = p_s3 * (1 - x_d_f[2])
list = [(max(x_d[0], 1-component_info[1]['defect_rate'])+max(x_d[1],
    1-component_info[2]['defect_rate'])+max(x_d[2],
    1-component_info[3]['defect_rate'])-intermediate_info[1]['disassembly_cost'])*p_s_d,
(max(x_d[3], 1-component_info[4]['defect_rate'])+max(x_d[4],
    1-component_info[5]['defect_rate'])+max(x_d[5],
    1-component_info[6]['defect_rate'])-intermediate_info[2]['disassembly_cost'])*p_s2_d,
(max(x_d[6],
    1-component_info[7]['defect_rate'])-intermediate_info[3]['disassembly_cost'])*p_s3_d]

# 计算半成品的总成本
intermediate_cost = [
(intermediate_info[1]['assembly_cost'] + x_d_f[0] * intermediate_info[1]['detection_cost'] +
    x_s[0] * intermediate_info[1]['disassembly_cost']) * p_s_d,
(intermediate_info[2]['assembly_cost'] + x_d_f[1] * intermediate_info[2]['detection_cost'] +
    x_s[1] * intermediate_info[2]['disassembly_cost']) * p_s2_d,
(intermediate_info[3]['assembly_cost'] + x_d_f[2] * intermediate_info[3]['detection_cost'] +
    x_s[2] * intermediate_info[3]['disassembly_cost']) * p_s3_d
]
intermediate_cost_real = [x-y for x, y in zip(intermediate_cost, list)]

intermediate_cost_sum = sum(intermediate_cost_real)
# 计算成品的次品率
p_f_d = 1 - (1-p_s_d)*(1-p_s2_d)*(1-p_s3_d)*(1-final_product_info['defect_rate']) #
    成品次品率基于所有半成品的次品率

# 如果检测成品，则减少进入市场的次品比例
p_f_d = p_f_d * (1 - x_d_final)

# 计算成品的总成本
final_product_cost = (
final_product_info['assembly_cost'] +
x_d_final * final_product_info['detection_cost'] +
x_s_final * final_product_info['disassembly_cost']
) + (
final_product_info['return_cost'] -
(max(x_d_f[0], 1-p_s)*intermediate_cost_real[0]+max(x_d_f[1],
    1-p_s2_d)*intermediate_cost_real[1]+max(x_d_f[2],
    1-p_s3_d)*intermediate_cost_real[2]-final_product_info['assembly_cost'])
) * p_f_d

# 总成本
total_cost = component_cost + intermediate_cost_sum + final_product_cost

return total_cost

# 决策变量的范围：0到1
```

```
bounds = [(0, 1)] * decision_vars

# 使用遗传算法优化
result = differential_evolution(objective, bounds)
optimal_decision = result.x

print(f"最优决策变量: {optimal_decision}")
print(f"最小化总成本: {objective(optimal_decision)}")
```

附录 D　问题四第一部分

```
import numpy as np
from scipy.optimize import differential_evolution

# 模拟抽样检测次品率，允许设定抽样检测误差
def simulate_defect_rate(true_rate, sample_size, max_error=0.05):
while True:
samples = np.random.binomial(1, true_rate, sample_size)
estimated_rate = np.mean(samples)
error = abs(estimated_rate - true_rate)
if error <= max_error:
break
return estimated_rate

# 决策变量数量
n_components = 8
n_intermediates = 3
n_final_products = 1 # 需要检测和拆解决策
decision_vars = n_components + n_intermediates + n_intermediates + n_final_products +
    n_final_products

# 目标函数
def objective(x):
error_margin = 0.08
true_rate = 0.2
sample_size = 100

# 动态抽样零配件、半成品和成品的次品率
component_info = {
1: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
2: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 8,
    'detection_cost': 1},
3: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 12,
    'detection_cost': 2},
```

```
4: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
5: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 8,
    'detection_cost': 1},
6: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
7: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
8: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
}

intermediate_info = {
1: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin),
    'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
2: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin),
    'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
3: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin),
    'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
}

final_product_info = {'defect_rate': simulate_defect_rate(true_rate, sample_size,
    error_margin), 'assembly_cost': 8, 'detection_cost': 6, 'disassembly_cost': 10,
    'return_cost': 40, 'market_price': 200}

# 解码决策变量
x_d = x[:n_components] # 零配件检测决策
x_d_f = x[n_components:n_components + n_intermediates] # 半成品检测决策
x_s = x[n_components + n_intermediates:2 * n_components + n_intermediates] # 半成品拆解决策
x_d_final = x[n_components + 2 * n_intermediates] # 成品检测决策
x_s_final = x[n_components + 2 * n_intermediates + 1] # 成品拆解决策

# 计算零配件的总成本
component_cost = sum(
(component_info[i + 1]['price'] + x_d[i] * component_info[i + 1]['detection_cost'])
for i in range(n_components)
)

# 计算半成品的次品率
p_s = 1 - np.prod([(1 - component_info[i + 1]['defect_rate']) for i in [0, 1, 2]]) * (1 -
    intermediate_info[1]['defect_rate'])
p_s2 = 1 - np.prod([(1 - component_info[i + 1]['defect_rate']) for i in [3, 4, 5]]) * (1 -
    intermediate_info[2]['defect_rate'])
p_s3 = 1 - np.prod([(1 - component_info[i + 1]['defect_rate']) for i in [6, 7]]) * (1 -
    intermediate_info[3]['defect_rate'])

# 如果检测半成品，则减少进入市场的次品比例
```

```
p_s_d = p_s * (1 - x_d_f[0])
p_s2_d = p_s2 * (1 - x_d_f[1])
p_s3_d = p_s3 * (1 - x_d_f[2])

# 计算半成品的总成本
intermediate_cost = [
(intermediate_info[1]['assembly_cost'] + x_d_f[0] * intermediate_info[1]['detection_cost'] +
    x_s[0] * intermediate_info[1]['disassembly_cost']) * p_s_d,
(intermediate_info[2]['assembly_cost'] + x_d_f[1] * intermediate_info[2]['detection_cost'] +
    x_s[1] * intermediate_info[2]['disassembly_cost']) * p_s2_d,
(intermediate_info[3]['assembly_cost'] + x_d_f[2] * intermediate_info[3]['detection_cost'] +
    x_s[2] * intermediate_info[3]['disassembly_cost']) * p_s3_d
]

intermediate_cost_sum = sum(intermediate_cost)

# 计算成品的次品率
p_f_d = 1 - (1 - p_s_d) * (1 - p_s2_d) * (1 - p_s3_d) * final_product_info['defect_rate']

# 如果检测成品，则减少进入市场的次品比例
p_f_d = p_f_d * (1 - x_d_final)

# 计算成品的总成本
final_product_cost = (
final_product_info['assembly_cost'] +
x_d_final * final_product_info['detection_cost'] +
x_s_final * final_product_info['disassembly_cost']
) + final_product_info['return_cost'] * p_f_d

# 总成本
total_cost = component_cost + intermediate_cost_sum + final_product_cost

return total_cost

# 决策变量的范围：0到1
bounds = [(0, 1)] * decision_vars

# 使用遗传算法优化
result = differential_evolution(objective, bounds)
optimal_decision = result.x

print(f"最优决策变量：{optimal_decision}")
print(f"最小化总成本：{objective(optimal_decision)}")
```

附录 E　问题四第二部分

```
import numpy as np
from scipy.optimize import differential_evolution

# 模拟抽样检测次品率，允许设定抽样检测误差
def simulate_defect_rate(true_rate, sample_size, max_error=0.05):
while True:
samples = np.random.binomial(1, true_rate, sample_size)
estimated_rate = np.mean(samples)
error = abs(estimated_rate - true_rate)
if error <= max_error:
break
return estimated_rate

# 决策变量数量
n_components = 8
n_intermediates = 3
n_final_products = 1 # 需要检测和拆解决策
decision_vars = n_components + n_intermediates + n_intermediates + n_final_products +
    n_final_products

# 目标函数
def objective(x):
error_margin = 0.08
true_rate = 0.2
sample_size = 100

# 动态抽样零配件、半成品和成品的次品率
component_info = {
1: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
2: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 8,
    'detection_cost': 1},
3: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 12,
    'detection_cost': 2},
4: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
5: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 8,
    'detection_cost': 1},
6: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
7: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
8: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin), 'price': 2,
    'detection_cost': 1},
```

```
}

intermediate_info = {
1: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin),
    'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
2: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin),
    'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
3: {'defect_rate': simulate_defect_rate(true_rate, sample_size, error_margin),
    'assembly_cost': 8, 'detection_cost': 4, 'disassembly_cost': 6},
}

final_product_info = {'defect_rate': simulate_defect_rate(true_rate, sample_size,
    error_margin), 'assembly_cost': 8, 'detection_cost': 6, 'disassembly_cost': 10,
    'return_cost': 40, 'market_price': 200}

# 解码决策变量
x_d = x[:n_components] # 零配件检测决策
x_d_f = x[n_components:n_components + n_intermediates] # 半成品检测决策
x_s = x[n_components + n_intermediates:2 * n_components + n_intermediates] # 半成品拆解决策
x_d_final = x[n_components + 2 * n_intermediates] # 成品检测决策
x_s_final = x[n_components + 2 * n_intermediates + 1] # 成品拆解决策

# 计算零配件的总成本
component_cost = sum(
(component_info[i + 1]['price'] + x_d[i] * component_info[i + 1]['detection_cost'])
for i in range(n_components)
)

# 计算半成品的次品率
p_s = 1 - np.prod([(1 - component_info[i + 1]['defect_rate']) for i in [0, 1, 2]]) * (1 -
    intermediate_info[1]['defect_rate'])
p_s2 = 1 - np.prod([(1 - component_info[i + 1]['defect_rate']) for i in [3, 4, 5]]) * (1 -
    intermediate_info[2]['defect_rate'])
p_s3 = 1 - np.prod([(1 - component_info[i + 1]['defect_rate']) for i in [6, 7]]) * (1 -
    intermediate_info[3]['defect_rate'])

# 如果检测半成品，则减少进入市场的次品比例
p_s_d = p_s * (1 - x_d_f[0])
p_s2_d = p_s2 * (1 - x_d_f[1])
p_s3_d = p_s3 * (1 - x_d_f[2])

# 计算半成品的总成本
intermediate_cost = [
(intermediate_info[1]['assembly_cost'] + x_d_f[0] * intermediate_info[1]['detection_cost'] +
    x_s[0] * intermediate_info[1]['disassembly_cost']) * p_s_d,
(intermediate_info[2]['assembly_cost'] + x_d_f[1] * intermediate_info[2]['detection_cost'] +
    x_s[1] * intermediate_info[2]['disassembly_cost']) * p_s2_d,
```

```
(intermediate_info[3]['assembly_cost'] + x_d_f[2] * intermediate_info[3]['detection_cost'] +
    x_s[2] * intermediate_info[3]['disassembly_cost']) * p_s3_d
]

intermediate_cost_sum = sum(intermediate_cost)

# 计算成品的次品率
p_f_d = 1 - (1 - p_s_d) * (1 - p_s2_d) * (1 - p_s3_d) * final_product_info['defect_rate']

# 如果检测成品，则减少进入市场的次品比例
p_f_d = p_f_d * (1 - x_d_final)

# 计算成品的总成本
final_product_cost = (
final_product_info['assembly_cost'] +
x_d_final * final_product_info['detection_cost'] +
x_s_final * final_product_info['disassembly_cost']
) + final_product_info['return_cost'] * p_f_d

# 总成本
total_cost = component_cost + intermediate_cost_sum + final_product_cost

return total_cost

# 决策变量的范围：0到1
bounds = [(0, 1)] * decision_vars

# 使用遗传算法优化
result = differential_evolution(objective, bounds)
optimal_decision = result.x

print(f"最优决策变量：{optimal_decision}")
print(f"最小化总成本：{objective(optimal_decision)}")
```