基于多阶段决策和序贯分析的产品质量检测与成本控制的研究

摘　要

随着现代制造业和智能化生产的发展，质量控制和生产优化问题成为企业管理者最为关注的核心问题。为了解决生产过程中质量的不确定性以及成本控制的需求，本文结合了抽样检测理论和成本效益分析，通过确定检测成本、次品率、装配成本等指标，以最大化企业利润与最小化生产成本为目标，构建数学模型，并使用优化算法对模型进行求解，制定了检测与处理的最佳决策。

针对问题一，主要是某企业生产电子产品过程中涉及的零配件质量检测问题，我们首先设计了利用传统假设检验方法建立零配件次品率的检测方案，然后尝试通过序贯概率比检验（SPRT）方法进行优化，在保证信度的前提下，减少检测次数，降低成本。通过理论推导与计算，本文给出了 95%信度下拒收零配件以及 90%信度下接收零配件两种情形下的具体解决方案，并比较了这两种方法在样本量和性能上的表现。最后，基于仿真结果对模型进行了评价和检验。

针对问题二，本文通过构建生产全流程的决策模型，对不同情境下零配件和成品次品率、检测成本、拆解费用等数据进行综合分析，比较了不同决策对生产成本的影响，并提供了具体的决策依据及相应的经济指标，构建了一套优化策略，帮助企业在保证产品质量的同时，最小化成本损失。

针对问题三，本文进一步扩展了问题二的模型，考虑到多个零配件与多个生产工序之间的联系，通过对比多个策略组合的成本表现，通过递归算法和遗传算法求解，找出在复杂生产流程中的最优生产控制方案。

针对问题四，本文基于前面问题建立的模型，结合实际生产中的抽样检测数据，对模型进行了调整和优化，同时计算了保守决策与基于期望的决策，帮助企业优化决策。

最后，我们对建立的模型进行全方面的评价和推广，并对模型在解决现实问题的能力上进行论证。

关键词：生产过程决策　假设检验　递推思想　遗传算法　贝叶斯定理　序贯分析

一、 问题重述

1.1 问题背景

某企业计划生产一种畅销的电子产品，产品的生产需要购买两种零配件（零配 1 和零配件 2），并将它们装配成成品。在装配过程中，若其中任何一个零配件不合格，则成品必然不合格；若两个零配件都合格，成品也不一定合格。对于不合格的成品，企业可以选择报废或拆解。拆解过程不会损坏零配件，但会产生拆解费用。

为了优化企业的生产流程，减少成本损失，企业希望通过数学建模来解决以下几个问题：如何通过抽样检测方案确定是否接受供应商提供的零配件，如何在生产的各个阶段做出合理的决策以降低次品率，减少检测和拆解成本，以及如何在多道工序和多个零配件的条件下进行决策优化。

1.2 问题要求

问题一：企业从供应商处采购某种零配件用于产品装配，企业担心零配件的次品率可能超过标称值（10%）。希望通过抽样检测方法来决定是否接收供应商的零配件。如果次品率超过标称值，企业将拒收这批零配件。目标是选取合适的抽样检测方法，以最小样本量在指定信度下做出合理决策，并计算出检测次品率所需的最小样本量。两种具体场景如下：

情形一：在 95%的信度下，认定次品率超过标称值，则拒收；

情形二：在 90%的信度下，认定次品率不超过标称值，则接收。

问题二：企业在生产过程中面临多个决策环节，分别包括：是否对零配件进行检测，是否对装配成品进行检测，是否拆解不合格成品，以及如何处理用户退回的不合格品。需要为企业提供一个决策方案，并根据不同情况下的次品率、检测成本等数据，给出每个环节的具体决策依据和相关指标结果。

问题三：企业的生产过程涉及多道工序和多个零配件。已知各个零配件、半成品、成品的次品率，需要在问题二的基础上对决策过程进行改进，给出针对多道工序的优化方案。

问题四：假设问题二和问题三中的次品率均是通过抽样检测方法得到的，重新完成问题二和问题三的决策优化，并根据检测结果调整生产过程中的各项决策。

二、 问题分析

2.1 问题一

问题一属于生产过程中的质量控制问题，企业需在供应商声称的次品率（标称值）与实际次品率之间进行对比，在一定置信度下通过抽样检测判定零配件是否符合要求，即零配件的次品率是否超过标称值，从而对是否接收零配件进行决策。若实际次品率超过标称值，企业面临的风险是高次品率可能导致大量不合格产品。因此，设计一个有效的抽样方案，既能在较高信度下识别高次品率，又能尽可能减少检测成本。

质量控制中的样本量确定是统计学中的经典问题，涉及置信度、误差范围和总体次品率等因素。传统方法通常基于固定的统计公式，我们可以基于二项分布建立数学模型，结合假设检验和大样本近似来设计抽样检测方案。但是，忽略了生产过程中可能存在的动态变化和先验信息。

通过引入序贯抽样优化，可以利用历史数据和实时反馈，动态调整样本量，实现更高效的检测。此外，不同的抽样方法在样本量需求和实施复杂度上存在差异，选择合适的方法对于优化检测过程至关重要。

2.2 问题二

在本问题中我们需要对生产过程中的多个环节进行决策，优化每个阶段的检测、装配、拆解及不合格品的处理，为企业生产过程提供决策。因此本文以企业利润最大化为目标建立优化模型，将各个决策环节设置为 0-1 变量，由于变量数量较少，对于每种情况仅有 16 种组合方式可以分别计算每种情况下的企业利润，最后我们通过比较分析不同情况下的成本确认最优的决策。

2.3 问题三

本问题是多工序和多个零配件下的决策优化，进一步扩展了生产环节，考虑了更加复杂的生产环境。通过对不同生产工序和多个零配件的综合分析，给出企业生产成本与工序和零配件关系的表达式，考虑在变量取值内的所有指标成果，构建全局优化模型，使用遗传算法和递归算法求解。

2.4 问题四

基于抽样检测得到的次品率，重新对问题二和问题三的决策模型进行了优化。通过结合抽样检测结果，模型进一步改进了生产决策的精确性，调整优化策略，更好地适应实际生产环境，提升决策效率和生产效益。

三、 模型假设

（1）**次品率稳定性假设：**零配件、半成品和成品的次品率在各个生产环节中是稳定的，不会随着时间和批次的变化而波动；

（2）**独立检测假设：**零配件质量与批次无关，所有零配件、半成品和成品的检测过程相互独立，检测过程是随机抽样，样本具有代表性。检测结果不会受到其他零配件或者工序的影响。每一步检测的结果仅取决于被检测对象的质量情况；

（3）**无外部干扰假设：**整个生产过程中不存在外部干扰因素，如市场需求、设备故障等，这些因素不会影响生产成本和次品率；

（4）**成本线性关系假设：**生产中涉及到的成本（包括检测成本、装配成本、拆解成本、市场损失等）是线性增加的，即单位成本不随生产规模或检测数量的变化而变化；

（5）**次品率传递假设：**半成品和成品的次品率受到装配零件的次品率影响。次品率的传递遵循简单的概率叠加原则，不考虑复杂的交互影响；

（6） **检测准确性假设：**检测设备和方法能够准确判定零配件、半成品和成品的合格与否，不存在误判或者漏判的情况；

（7）**拆解回收假设：**拆解过程不会损坏零配件，拆解后的零配件可以被重新使用且质量与原始状态一致。

（8）**检验错误率已知假设：**企业能够控制并且已知检验中的第一类错误率（假阳性）和第二类错误率（假阴性）。

四、 符号说明

符号	说明
p_0	供应商声称的次品率标称值， $p_0 = 0.10$
p_i	零配件i的次品率，$i = 1,2$
α	第一类错误的概率，即拒收合格零配件的概率
β	第二类错误的概率，即接收不合格零配件的概率
n	抽样样本数量

x	样本中的次品数量
$\hat{p}$	样本中次品率的估计值，即实际检测到的次品率
A	SPRT 的上限
B	SPRT 的下限
b_i	零配件i的购买单价，$i = 1,2$
d_i	零配件i的检测成本，$i = 1,2$
p_f	将正品零配件装配后的产品次品率
a_f	成品的装配成本
d_f	成品的检测成本
s_f	成品的市场售价
r_f	不合格成品的调换损失
m_f	不合格成品的拆解费用
x_1	是否对零配件 1 进行检测的 0-1 变量
x_2	是否对零配件 2 进行检测的 0-1 变量
x_3	是否对装配好的成品进行检测的 0-1 变量
x_4	是否对不合格成品进行拆解的 0-1 变量
$p_{i,j}$	第i道工序第j件子配件的次品率
$b_{i,j}$	第i道工序第j件子配件的购买单价
$d_{i,j}$	第i道工序第j件子配件的检测成本
l_i	第i道工序的子配件数
$p_{s,i}$	第i道工序的父配件使用正品子配件装配后的次品率
$a_{s,i}$	第i道工序的父配件的装配成本
$d_{s,i}$	第i道工序的父配件的检测成本
$m_{s,i}$	不合格的第i道工序的父配件的拆解费用
$x_{i,j}$	表示是否对第i道工序第j件子配件进行检测的 0-1 变量
$x_{p,i}$	表示是否对第i道工序的父配件进行检测的 0-1 变量
$y_{p,i}$	表示是否对不合格的第i道工序的父配件进行拆解的 0-1 变量

对于所有的 0-1 变量，0 为否，1 为是

问题一中p_1表示企业可接受的次品率上限

五、 模型建立与求解

5.1 问题一模型建立与求解

5.1.1 模型建立思路

样本量的确定基于统计学中的置信区间理论，目标是在给定的置信度和误差范围下，最小化所需的样本量。

首先，先通过假设检验来验证供应商的声称值是否可靠，通过简单随机抽样的样本量公式进行初步计算。

在此基础上使用序贯概率比检验（SPRT）对样本量进行优化，利用先验信息和实时反馈，动态更新检测数据，调整抽样决策，达到减少检测次数的目的，减少企业检测成本，实现进一步的优化。

5.1.2 模型建立

一、假设检验模型：

1. 设立假设：

零假设H_0：$p \leq p_0$（即零配件实际次品率不大于标称值）

备择假设H_1：$p > p_0$（即零配件实际次品率大于标称值）

p_0指供应商承诺的零件次品率标称值，此处$p_0 = 0.10$；p指该批零件的实际次品率。接下来我们将采用单边假设检验来验证次品率是否大于标称值。

2. 模型建立

假设零配件的次品率 p 是已知的，那么抽样中发现次品的数量服从二项分布$x \sim B(n,p)$，其中 n 是抽样大小。

抽样大小计算：可以使用以下公式来估计抽样大小 n：

$$n = \frac{Z_\alpha^2 \cdot p \cdot (1-p)}{E^2}$$

其中，Zα 是标准正态分布的临界值，即分位数，它取决于所选的信度水平，E 是可接受的误差范围，设定为 2%。

次品数临界值计算：

对于两种情形，根据公式计算最小样本量，并确定样本中的次品数临界值。

二、SPRT 模型：

对于单式抽样方案，选定两个整数 n>c>=0，从该批零部件中随机抽取 n，如果这 n

件中所含的不合格品件数 d>c，则拒收该批产品；若 d<=c，则接收该批产品。该验收方案的样本量（抽样量）是固定的整数 n，但是如果企业在抽样过程中，如果未抽到 n 件就已抽到了 c+1 个不合格品，就没有继续抽样的必要。故而选择事先固定样本量，会造成成本、时间等资源的浪费。

1. 理论介绍：

序贯概率比检验是一种逐次采样的方法，依次计算次品率的似然比，直到累积的证据足够多来做出决策。其优点是平均样本量较小，能动态终止采样。

设X的概率密度是$f_1(x)$或$f_2(x)$，真实概率分布无法确定是二者中的哪一个，假设H_1：$f_1(x)$是真正的分布密度，对立假设H_2：$f_2(x)$是真正的分布密度。考虑似然统计量：$L_n = \frac{\prod_{i=1}^{n} f_2(X_i)}{\prod_{i=1}^{n} f_1(X_i)}$ $(n \geq 1)$，根据 N-P 引理，找出临界值 A 和 B，当$L_n \geq B$，则停止观测并拒绝H_1；若$L_n \leq A$，则停止观测并接受假设H_1；若$B < L_n < A$，则进行下一次观测，根据观测值计算下一个似然比L_{n+1}；不断迭代，直至停止观测，作出决策。可见 SPRT 分析的关键在于停止法则的具体表达式。

2. 模型建立：

将其运用到该公司对工厂零部件进行检测的实际中，由前述假设检验中的讨论可知，零配件抽样次品数应符合伯努利分布。

对于伯努利分布情形：

$$L = \frac{P(\text{当前观测数据}|H_1)}{P(\text{当前观测数据}|H_0)} = \frac{p_1^x(1-p_1)^{n-x}}{p_0^x(1-p_0)^{n-x}}$$

其中 n 为当前的样本数，x 为检验到的不合格品数量

检验设计：

在每次抽样后，计算当前累计样本的似然比（L），并将其与两个临界值 A 和 B，然后进行比较：根据抽样过程中出现的情况来决定抽样量。

若$L \geq A$，则拒收该批零部件，认定次品率超过 10%；

若$L < B$，则接收该批零部件，认定次品率不超过 10%；

若$B < L < A$，则继续抽样。

下面计算决策边界 A 和 B：

$$A = \frac{1-\beta}{\alpha}$$

$$B = \frac{\beta}{1-\alpha}$$

两个边界是基于 Wald 的假设推导所得。A 为上边界，当似然比超过 A 时，有足够的证据拒绝原假设；B 是下边界，当似然比低于 B 时，没有充足的证据拒绝原假设。

取对数简化计算，得：

$$lnL(x) = xln\left(\frac{p_1}{p_2}\right) + (n-x)\ln\left(\frac{1-p_1}{1-p_0}\right)$$

$$lnA = \ln\left(\frac{1-\beta}{\alpha}\right)$$

$$lnB = \ln\left(\frac{\beta}{1-\alpha}\right)$$

自适应序贯抽样检测的算法流程：

- 从供应商的零配件批次中进行抽样，每次抽样后，计算当前次品率的似然比
- 比较似然比L与决策边界 A 和 B，决定是否继续抽样
- 根据观测到的次品数累计更新检验结果

3. 模型评估：

操作特征函数(OC 函数)：

$$\mathrm{L(p)} \approx \frac{\alpha(1-\beta) - (1-\alpha)\beta\left(\frac{\mathrm{p}}{\mathrm{p_0}}\right)^{\mathrm{h}}}{\left(\frac{\mathrm{p}}{\mathrm{p_1}}\right)^{\mathrm{h}} - \left(\frac{\mathrm{p}}{\mathrm{p_0}}\right)^{\mathrm{h}}}$$

其中，$\mathrm{h} = \frac{\ln\left(\frac{1-\mathrm{p_1}}{1-\mathrm{p_0}}\right)}{\ln\left(\frac{\mathrm{p_1}(1-\mathrm{p_0})}{\mathrm{p_0}(1-\mathrm{p_1})}\right)}$

OC 函数描述了在不同实际次品率 p 下，接受批次的概率(AP)

平均样本量函数(ASN 函数)：

$$\mathrm{E(n|p)} \approx \frac{\mathrm{L(p)lnB} + (1-\mathrm{L(p)})\mathrm{lnA}}{\mathrm{E(z|p)}}$$

其中$\mathrm{E(z|p)} = \mathrm{pln}\left(\frac{\mathrm{p_1}}{\mathrm{p_0}}\right) + (1-\mathrm{p})\ln\left(\frac{1-\mathrm{p_1}}{1-\mathrm{p_0}}\right)$

ASN 函数描述了在不同实际次品率 p 下，平均需要的样本量（ASN）

5.1.3 模型求解与结果分析

一、模型一具体计算（假设检验）：

情况一：95%信度下，认定次品率超过 10%，则拒收零配件

$$Z_{0.05} = 1.645，p = 0.10，E = 0.02$$

$$n_{95} = (\frac{1.645 \times \sqrt{0.10 \times 0.9}}{0.02})^2 \approx 609$$

情况二：90%信度下，认定次品率不超过 10%，则接收零配件

$$Z_{0.1} = 1.282，p = 0.10，E = 0.02$$

$$n_{95} = (\frac{1.282 \times \sqrt{0.10 \times 0.9}}{0.02})^2 \approx 370$$

决策方案设计与次品数临界值确定

次品数的临界值$x_{临界} = p_0 \cdot n + Z_\alpha \cdot \sqrt{n \cdot p_0 \cdot (1 - p_0)}$

$$x_{临界_1} = 0.10 \cdot 98 + 1.645 \cdot \sqrt{609 \cdot 0.10 \cdot (1 - 0.10)} \approx 73.06 \approx 74$$

$$x_{临界_2} = 0.10 \cdot 59 + 1.282 \cdot \sqrt{370 \cdot 0.10 \cdot (1 - 0.10)} \approx 44.40 \approx 45$$

95%的置信度下，样本数量为 609。当检测出的次品数目大于临界值 74 时，拒收这批零配件；90%的置信度下，样本数量为 370。当检测出的次品数目大于临界值 45 时，拒收这批零配件；

二、模型二具体计算（SPRT 方法）：

使用随机数模拟抽样，单次模拟结果如下：

95%的置信度下，样本数量为 200。当检测出的次品数目大于临界值 18 时，拒收这批零配件；90%的置信度下，样本数量为 88。当检测出的次品数目大于临界值 10 时，拒收这批零配件；

我们设置了真实次品率的范围（0.05，0.26），步长设为 0.01，进行循环模拟抽样，下面列举了 7 轮的结果：

		轮次	1	2	3	4	5	6	7	average
95%		n	210	137	88	72	119	331	145	
		$x_{临界}$	21	12	6	4	21	36	13	
	p_0	AP	0.949	0.952	0.953	0.958	0.953	0.952	0.953	0.953
		ASN	187.248	190.834	190.628	197.262	188.479	198.53	180.084	190.438
	p_1_high	AP	0.115	0.099	0.098	0.111	0.098	0.089	0.104	0.102

		ASN	205.361	206.459	198.511	198.49	199.087	200.8	209.498	202.601
90%		n	23	180	150	41	27	82	105	
		$x_{临界}$	6	17	15	7	6	10	12	
	p_0	AP	0.908	0.908	0.895	0.901	0.902	0.899	0.901	0.902
		ASN	127.701	129.324	128.039	125.237	124.759	131.437	128.224	127.817
	p_1_low	AP	0.03	0.04	0.034	0.043	0.041	0.038	0.039	0.038
		ASN	124.459	128.112	125.008	120.337	122.699	123.818	124.322	124.108

上表分别反映了情形 1（95%信度）和情形 2（90%信度）下，单次抽样的抽样数和次品临界数以及每轮次中 OC 和 ASN 曲线在 p0 和 p1_high（或 p1_low）点处的值。

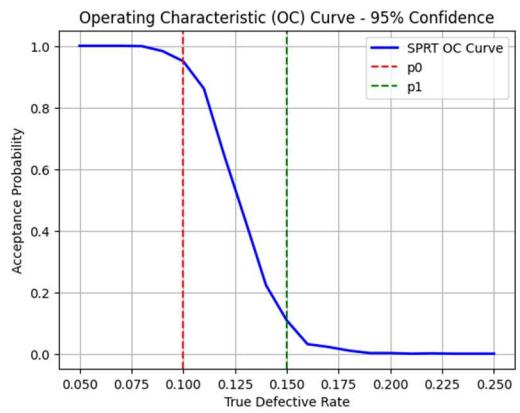
可以看出在 95%信度下，在p_0处接受概率（AP）接近 0.95，在p_1_high处的接受概率接近 0.1；同理 90%信度下在各处的 AP 值也接近理论值，这反映了该模型与实际情况的拟合度高，具有可信度。

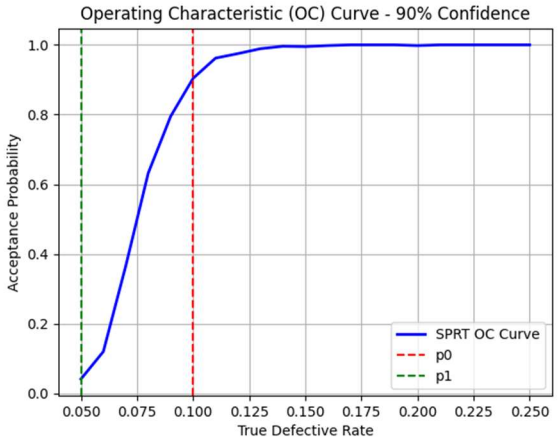
每轮的平均抽样数比较稳定，95%信度下大约在 190 左右，90%信度下大约在 128 左右，相比于使用假设检验方法计算的固定抽样数 609 和 370，使用 SPRT 方法能够非常有效地减小最小抽样数，减小企业的检测成本。

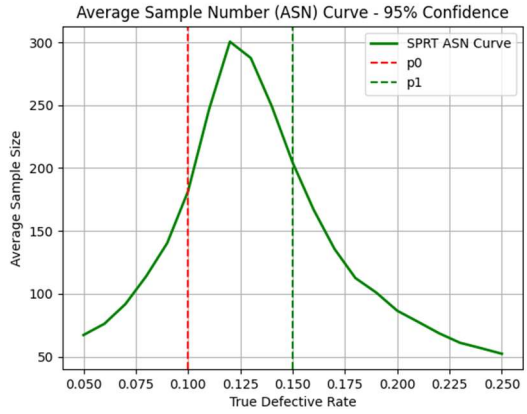
5.1.4 模型评估

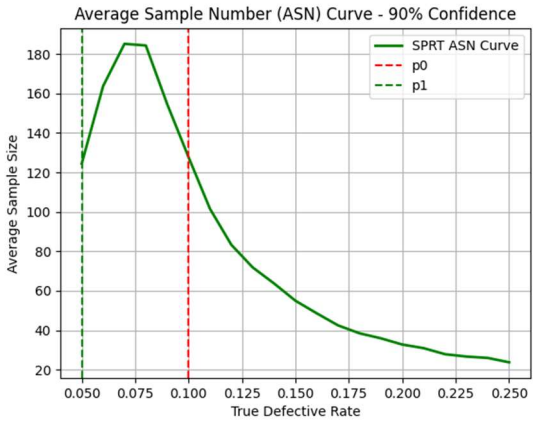
SPRT 模型性能验证可视化结果：

绘制某轮次模拟抽样时 OC 曲线和 ASN 曲线，展示在不同真实次品率下的模型表现：









由 OC 曲线可知，当该真实次品率小于 p0 接近或小于时，接收的概率很高；当真实次品率接近或者大于 p1（0.15）时，接收率很低。这说明 SPRT 模型能够有效地区分批次的质量。

由 ASN 曲线可知，当真实次品率接近 p0 或 p1 时，平均样本量较小；当真实次品率介于二者之间时，平均样本量较大，即当质量水平接近决策边界时，需要较多的样本才能够做出准确度更高的决策。

通过仿真实验验证了模型在不同次品率下的性能表现，绘制了操作特征（OC）曲线和平均样本量（ASN）曲线。结果表明，SPRT 方法在不同次品率下表现良好，能有效减少样本量并保持较高的决策准确性。

综上，假设检验提供了确定性检验的方案，而 SPRT 能动态调整，通常情况下更具优势，即最佳检测方案采用序贯概率比检验（SPRT）方法，其具体方案为：
设置允许误差（本文采用 2%），从供应商的零配件批次中进行抽样，每次抽样后，计算当前次品率的似然比， 比较似然比 L 与决策边界 A 和 B，决定是否继续抽样，根据观测到的次品数累计更新检验结果，直至达到上界或下界。

本实验中 95%信度下最小样本数约为 190，90%信度下最小样本数约为 128。

5.2 问题二模型建立与求解

5.2.1 模型建立思路

企业在生产过程中面临三大决策：

（1） 是否检测零配件：检测零配件后，不合格的零配件将被丢弃，剩余零配件将

用于装配成品。检测零配件将产生检测成本，但可降低成品的次品率；

（2） 是否检测成品：检测成品可将不合格品剔除，减少市场上的次品风险，降低企业因产品不合格带来的调换损失，但也会因此增加检测成本；

（3） 是否拆解不合格的产品：企业可以选择拆解不合格的成品，让零件再次使用，或者选择直接将不合格的成品丢弃。拆解会增加成本，但有可能减少总体的损失，需要具体分析。

5.2.2 模型建立

为便于计算，采用装配一件产品的期望利润作为决策的指标。以下计算均为期望值。

当对零配件 i 进行检测时，由于检测出的不合格零配件需要丢弃，所以购入次品率为 0 的零配件 i 的单价为：

$$\frac{b_i + d_i}{1 - p_i}$$

因此购买零配件的总成本为：

$$C_b = \sum_{i=1}^{2} \frac{b_i + d_i}{1 - p_i} x_i + \sum_{i=1}^{2} b_i (1 - x_i)$$

装配成本为：

$$C_a = a_f$$

装配完成后产品次品率为：

$$p = 1 - (1 - p_f) \prod_{i=1}^{2} (1 - p_i)^{1 - x_i}$$

对成品进行检测的成本为：

$$C_d = d_f x_3$$

如果对成品进行了检测，则不存在需要调换的情况，因此调换的成本为：

$$C_r = r_f p (1 - x_3)$$

对不合格成品进行拆解对成本为：

$$C_m = m_f p x_4$$

由于不合格品需要调换，因此销售产品的收入为：

$$I_s = s_f (1 - p)$$

对于拆解获得的零配件，当装配之前已对零配件进行检测时，这些零配件的价格可视为获得等量的次品率为 0 的相应配件。当装配之前未对零配件进行检测时，若不对其

进行检测而直接用于生产，将造成不合格的零配件不断积累，因此先需要对其进行检测。我们将这些零配件抵扣下一次生产的成本视为本次生产中拆解不合格成品的收入，为：

$$\mathrm{I_m}=\sum_{i=1}^{2}\frac{b_i}{1-p_i}px_ix_4+\sum_{i=1}^{2}\left[\frac{b_i}{1-p_i}(p-p_i)-d_ip\right](1-x_i)x_4$$

因此，目标函数为：

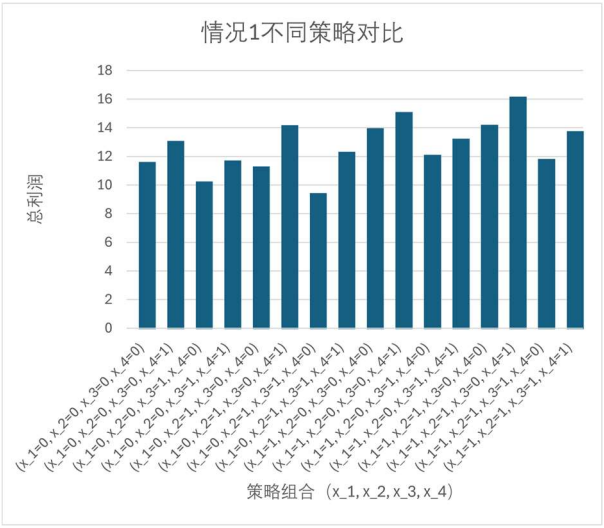
$$\max z=\mathrm{I_s}+\mathrm{I_m}-\mathrm{C_b}-\mathrm{C_d}-\mathrm{C_r}-\mathrm{C_m}$$

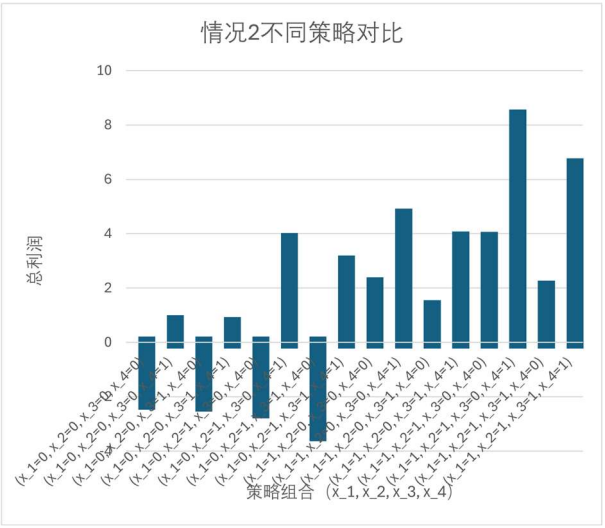
约束条件为：

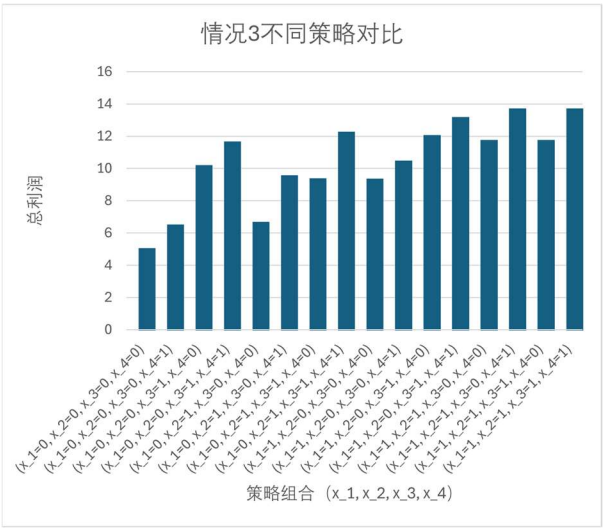
$$x_j=0\text{ 或 }1,\ j=1,2,3,4$$

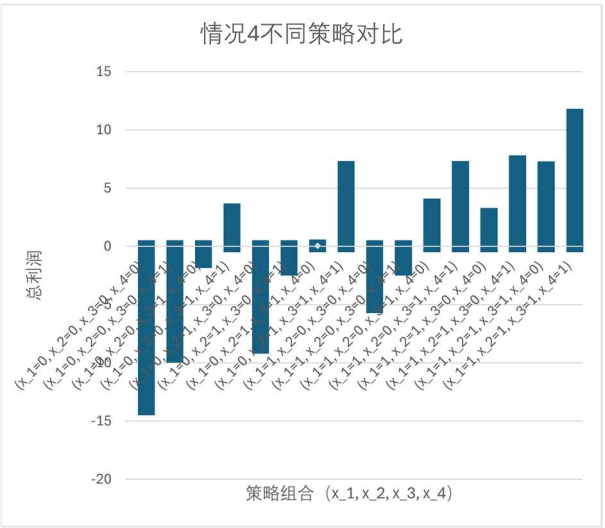
5.2.3 模型求解

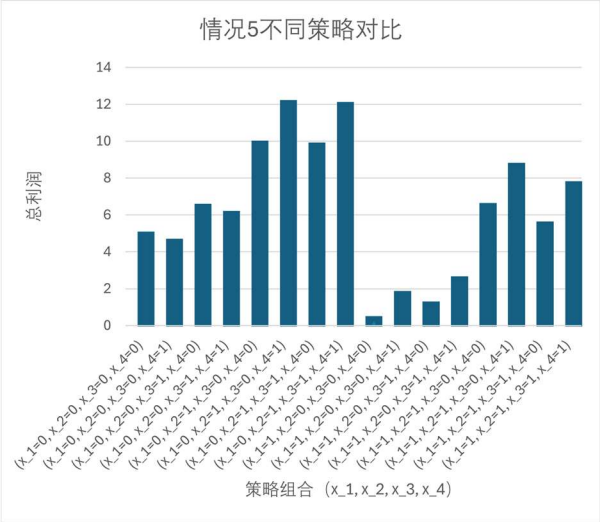
由于变量数目较少，我们通过穷举所有可能的组合，使用代码计算每种策略组合下的总利润，选取总利润最高的策略组合作为最优解，并绘制图表。

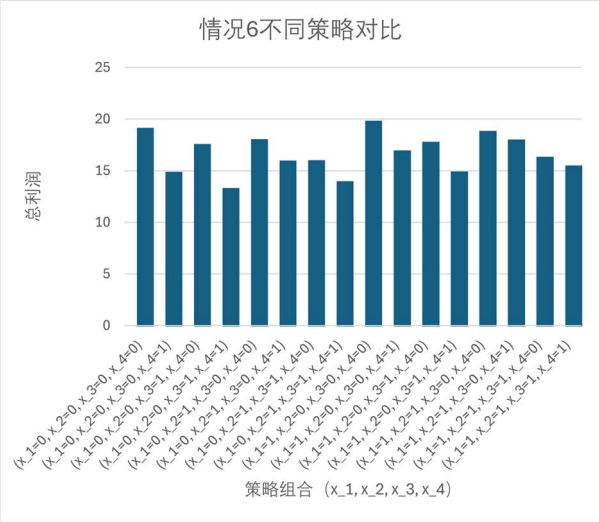












通过比较分析我们得到了不同情形下的最优解:

情况	x_1	x_2	x_3	x_4	总利润 z
1	1	1	0	1	15.744
2	1	1	0	1	8.350
3	1	1	0	1	13.344
4	1	1	1	1	11.300
5	0	1	0	1	11.915
6	1	0	0	0	19.249

5.2.4 结果分析

1.在多数情况下，同时检测零配件 1 和零配件 2 是最优选择，特别是在次品率较高或检测成本较低时，零配件的检测可以有效提高总利润；

2.检测成品在大多数情况下是非必要的，对零配件的检测可以降低后续成品的次品率，只有在次品率较高且检测成本较低时（如情况 4），检测成品可以得到较高的利润。

3.当拆解费用较低时，拆解不合格品是最佳选择。但在拆解费用较高时（如情况 6），不拆解能够得到更高的经济效益。

5.3 问题三模型建立与求解

5.3.1 模型建立思路

零配件具有三个参数：次品率、购买单价和检测成本，相关决策包括是否在装配前

对其进行检测。半成品则有五个参数：零配件数量、正品子配件装配后的次品率、装配成本、检测成本和拆解费用，相关决策包括是否在装配前对其进行检测以及是否对不合格品进行拆解。零配件始终作为子配件存在，而半成品既是父配件的同时，也是下一道工序的子配件。为了使父配件能够作为子配件参与下一道工序的计算，需要在上一道工序的计算中将其参数转换为与最初的子配件（即零配件）相似的形式，其中，父配件无需购买，其购买单价等同生产成本。

5.3.2 模型建立

定义第 i 道工序中第 j 件子配件参数和决策变量的集合为 $C_{i,j}$，$C_{i,j}$对于 j 的集合为 C_i，父配件参数和决策变量的集合为 P_j，则：

$$C_{i,j} = \{p_{i,j}, b_{i,j}, d_{i,j}, x_{i,j}\}, \qquad i = 1,2,\cdots,m$$

$$P_i = \{l_i, p_{s,i}, a_{p,i}, d_{p,i}, m_{p,i}, x_{p,i}, y_{p,i}\}, \qquad i = 1,2,\cdots,m$$

对于第 i 道工序，根据子配件的次品率和检测与否，可计算得父配件次品率为：

$$p_{p,i} = F_1(C_i) = 1 - (1 - p_{s,i}) \prod_{j=1}^{l_i} (1 - p_{i,j})^{1 - x_{i,j}}$$

父配件装配需要的购买子配件成本：

$$\mathrm{Cost}_{\mathrm{b,i}} = \sum_{j=1}^{l_i} \frac{b_{i,j} + d_{i,j}}{1 - p_{i,j}} x_{i,j} + \sum_{j=1}^{l_i} b_{i,j}(1 - x_{i,j})$$

父配件装配成本：

$$\mathrm{Cost}_{\mathrm{a,i}} = a_{p,i}$$

父配件拆解不合格品的成本：

$$\mathrm{Cost}_{\mathrm{m,i}} = m_{p,i} p_{p,i} y_{p,i}$$

父配件拆解不合格品产生的收益：

$$\mathrm{Profit}_{\mathrm{m,i}} = \sum_{j=1}^{l_i} \frac{b_{i,j}}{1 - p_{i,j}} p_{p,i}\, x_{i,j} y_{p,i} + \sum_{j=1}^{l_i} \left[\frac{b_{i,j}}{1 - p_{i,j}} (p_{p,i} - p_{i,j}) - d_{i,j} p_{p,i} \right] (1 - x_{i,j}) y_{p,i}$$

综上，父配件购买单价（生产成本）为：

$$b_{p,i} = F_2(C_i) = \mathrm{Cost}_{\mathrm{b,i}} + \mathrm{Cost}_{\mathrm{a,i}} + \mathrm{Cost}_{\mathrm{m,i}} - \mathrm{Profit}_{\mathrm{m,i}}$$

父配件的检测成本和是否在装配前对其进行检测的决策变量原本已有，假设该父配件在下一道工序中为第 k 件子配件，则：

$$C_{i+1,k} = \{\mathrm{F}_1(C_i), \mathrm{F}_2(C_i), d_{p,i}, x_{p,i}\}, \qquad i = 0,1,2,\cdots,m-1$$

j 与 k 之间的关系应有具体的生产流程确定。

经过 m 道工序后，可得到 $C_{m,1}=\{p_{m,1},b_{m,1},d_{m,1},x_{m,1}\}$，其中$p_{m,0}$为成品次品率，$b_{m,1}$为成品除去检测成本和调换损失后的生产成本。所以，成品的单件利润为：

$$\mathrm{Profit_f}=s_f(1-p_{m,1})-b_{m,1}-d_f x_{p,m}-r_f p_{m,1}(1-x_{p,m})$$

最后，模型的目标函数为：

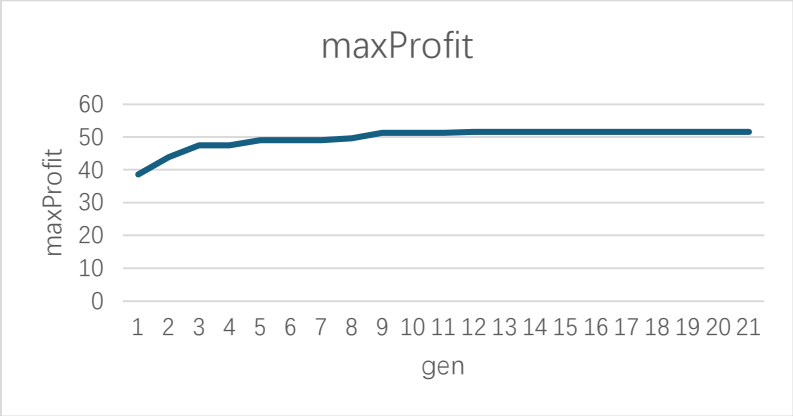
$$\max \mathrm{Profit_f}$$

模型的约束条件为：

$$\begin{cases} x_{p,i}=0 \text{ 或 } 1,\ i=1,2,\cdots,m \\ x_{0,j}=0 \text{ 或 } 1,\ j=1,2,\cdots,n \\ y_{p,i}=0 \text{ 或 } 1,\ i=1,2,\cdots,m \end{cases}$$

5.3.3 模型求解

针对题目中图 1 和表 2 中的情形，利用递归算法和遗传算法进行编程求解



得到的利润最大化的决策方案为：对所有零配件、半成品进行装配前的检测，对所有不合格的半成品、成品进行拆解，对成品不做检测。此时的成品单件利润为 51.644 元

5.4 问题四模型建立与求解

5.4.1 模型建立思路

当次品率是通过抽样检测方法得到的时候，我们可以使用贝叶斯公式计算真实次品率的后验分布，并通过鲁棒优化求解相对保守的策略，和通过随机规划以及蒙特卡洛模拟求解最大化目标函数的期望的策略。

5.4.2 模型建立

假设我们从一批真实次品率为 p 的零配件（或半成品、成品）中抽取了 n 个样本，并检测到其中有 k 个不合格品。那么 k 的分布可以建模为一个二项分布：

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

我们采用 Beta 分布作为 p 的先验分布。由于没有强先验信息。因此 p 的先验分布可以取为均匀分布，即:

$$p\sim \mathrm{Beta}(1,1)$$

根据贝叶斯公式，定义 p 的先验分布为 $P(p)$，则 p 的后验分布为：

$$P(p\mid k,n)\propto P(k\mid p,n)\cdot P(p)$$

将具体的形式代入：

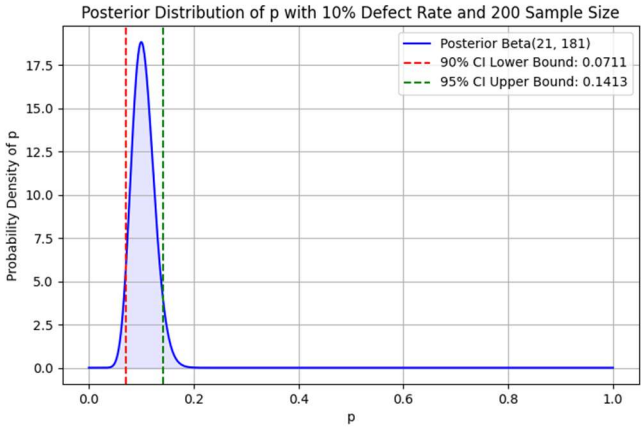
$$P(p\mid k,n)\propto\left[\binom{n}{k}p^k(1-p)^{n-k}\right]\cdot\left[\frac{p^{1-1}(1-p)^{1-1}}{B(1,1)}\right]$$

由于$\binom{n}{k}$ 和 $B(1,1)$ 是常数，因此：

$$P(p\mid k,n)\propto p^{(k)}(1-p)^{(n-k)}$$

p 的后验分布依然是 Beta 分布，为：

$$p\mid k,n\sim \mathrm{Beta}(k+1,n-k+1)$$



5.4.3 模型求解

保守决策

当使用鲁棒优化求解时，由于次品率越高对目标函数最大化越不利，因此可以选取置信区间上界作为决策参数即可。在这里我们对所有次品率的抽样检测方法均为抽取了 200 个样本，使用 90%置信度计算置信区间上界，然后代入问题 2 的模型进行求解，得到的各种情况的最优解如下：

情况	x_1	x_2	x_3	x_4	z
1	1	1	0	1	12.710

2	1	1	0	1	4.505
3	1	1	1	1	10.558
4	1	1	1	1	8.016
5	0	1	1	1	7.387
6	1	1	0	0	15.085

对于问题 3，使用相同的方法，代入问题 3 的模型进行求解，得到问题 3 的保守决策为：对所有零配件、半成品进行装配前的检测，对所有不合格的半成品、成品进行拆解，拆解后的零配件（半成品）不做检测，对成品不做检测。此时的成品单件期望利润为 39.347 元。

基于期望值决策

由于目标函数比较复杂，难以直接分析计算其期望值，因此使用蒙特卡洛模拟来进行估计。对于问题 2，从 p 的后验分布中生成十万个样本，对于每个样本计算相应的目标函数值，然后通过这些样本的平均值来近似估计期望值。最后求得的各种情况下的最优解为：

情况	x_1	x_2	x_3	x_4	z
1	1	1	0	1	15.435
2	1	1	0	1	8.093
3	1	1	1	1	13.062
4	1	1	1	1	11.084
5	0	1	0	1	11.436
6	1	0	0	0	18.657

对于问题 3，由于模型计算更加复杂，因此蒙特卡洛模拟中的样本量调整为一万个样本。最后的决策为：对所有零配件、半成品进行装配前的检测，对所有不合格的半成品、成品进行拆解，拆解后的零配件（半成品）不做检测，对成品不做检测。此时的成品单件期望利润为 50.447 元。

六、模型评价与推广

6.1 模型优点

1.贴近实际生产场景：本文的模型基于生产流程中的实际问题，通过对零配件、成品的次品率、检测成本、拆解费用等指标进行全面分析，提供了一个具有现实应用价值的生产优化方案。模型中的抽样检测和利润分析能够准确反映并满足企业在生产过程中的决策需求；

2.可量化的成本和收益：通过数学模型对不同决策下的生产成本和利润进行量化，使企业能够清晰地了解各项决策对成本以及利润的影响。使得模型具有良好的指导意义，能够为企业的生产决策提供数据支持；

3.灵活性高：模型可以适应不同的生产环境和参数设置，例如不同的零件次品率、成品次品率、拆解费用、检测费用等。模型在多个场景下，例如，检测与不检测、拆解与不拆解以及从少量零件到多零件、多工序等提供了灵活的决策组合。企业可以根据实际生产情况选择最优的处理策略，从而平衡成本与质量；

4.系统性和层次性强：该模型结构清晰，能够系统地描述从零配件购买、装配、检测到成品市场销售的整个生产流程。它考虑了零件、半成品、成品的次品率及其对成本和利润的影响，层次分明、逻辑性强。

5.适用广泛：假设检验法适合在明确样本量的情况下使用，能够为企业提供一个简单可行的检测方案，尤其是当企业希望确保严格的信度时。SPRT 方法动态调整抽样量，减少不必要的抽样，可以在前期抽样少量样本时快速拒收或接收，极大地减少了平均样本量，适用于企业在降低检测成本的同时保证决策准确性的情境。

6.2 模型缺点

1.忽略了动态因素：模型假设在生产过程中次品率、检测成本、市场损失等因素是固定的，忽略了在实际生产过程中这些因素可能会随着时间、市场需求等变化而波动，影响了模型的适用性；

2.缺乏对时间因素的考量：模型未考虑生产过程中的时间限制，比如生产、检测、拆解所需要的时间成本，可能无法有效解决实际问题；

3.简化成本计算：模型中对成本计算进行了简化处理。在实际的生产过程中，检测成本、拆解成本和调换成本往往更加复杂，例如，拆解过程中的零件损坏以及调换过程

中的损坏，这会导致实际生产过程中的成本估算与模型结果存在差异。

6.3 模型推广

1.在生产管理和利润优化方面，可以将模型中的次品率等参数替换为动态的市场参数，从而解决更广泛的决策优化问题；

2.本文提出的模型不仅适用于电子产品生产流程的质量控制与利润优化，也可以应用到其他制造业领域。对于依赖多个零配件和工序协同生产的行业，如汽车制造、机械制造等，该模型同样适用。

七、参考文献

[1]杨风萍.带有检测风险的抽样方案设计及其应用[D].南京信息工程大学,2023.DOI:10.27248/d.cnki.gnjqc.2023.001003.

[2]陈家鼎,孙山泽,李东风,刘力平.数理统计学讲义[M].北京：高等教育出版社,2006：325-350.

[3]张士军.计数序贯抽样检验方案的OC函数和ASN[J].军械工程学院学报,2013,25(3):70-73

[4]薛峰.产品质量检验抽样方法研究[J].建筑工程技术与设计,2018,(19):4989.

[5]Sivanandam, S. N., et al. "Genetic algorithm optimization problems." Introduction to genetic algorithms (2008): 165-209.

[6]亓四华.基于遗传算法的制造质量控制多目标的优化[J].农业机械学报,2006,(6):110-112,116.

[7]江海峰.单总体下假设检验的另类区间估计检验及其MCS研究[J].安徽工业大学学报（自然科学版）,2009,(4):422-427.

附录

问题一：

Python 代码

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

# ================================
# 假设检验法
# ================================

# 参数设置
p0 = 0.10  # 标称次品率
alpha_95 = 0.05  # 第一类错误概率（95%信度）
alpha_90 = 0.10  # 第一类错误概率（90%信度）
beta_10 = 0.10   # 第二类错误概率（95%信度）
beta_05 = 0.05   # 第二类错误概率（90%信度）
margin = 0.02    # 允许误差（2%）

# 计算 z 值
z_alpha_95 = norm.ppf(1 - alpha_95)  # 对应 95%信度
z_alpha_90 = norm.ppf(1 - alpha_90)  # 对应 90%信度

# 样本量计算函数
def sample_size(p, z_alpha, margin):
    q = 1 - p  # 合格品率
    n = math.ceil((z_alpha ** 2) * p * q / (margin ** 2))
    return n

# 次品数临界值计算函数
def critical_value(n, p, z_alpha):
    return math.ceil(n * p + z_alpha * math.sqrt(n * p * (1 - p)))

# 计算在 95%信度下的样本量和次品数临界值（情形 1）
n_alpha_95 = sample_size(p0, z_alpha_95, margin)
critical_value_95 = critical_value(n_alpha_95, p0, z_alpha_95)
```

```
# 计算在 90%信度下的样本量和次品数临界值（情形 2）
n_alpha_90 = sample_size(p0, z_alpha_90, margin)
critical_value_90 = critical_value(n_alpha_90, p0, z_alpha_90)

print("假设检验法结果：")
print(f"情形 1（95%信度）最小样本量：{n_alpha_95},次品数临界值：{critical_value_95}")
print(f"情形 2（90%信度）最小样本量：{n_alpha_90},次品数临界值：{critical_value_90}")
print("\n")

# ================================
# SPRT 方法
# ================================

# 参数设置
p1_high = 0.15  # 假设检验中的备择假设次品率（超出标称值）
p1_low = 0.05   # 假设检验中的备择假设次品率（低于标称值）

# 计算决策边界
lnA_high = np.log((1 - beta_10) / alpha_95)
lnB_high = np.log(beta_10 / (1 - alpha_95))
lnA_low = np.log((1 - beta_05) / alpha_90)
lnB_low = np.log(beta_05 / (1 - alpha_90))

# SPRT 模拟函数，返回最小样本量和次品数临界值
def sprt_min_sample_size(p0, p1, lnA, lnB, true_p):
    n = 0
    sum_x = 0
    while True:
        n += 1
        x = np.random.rand() < true_p  # 模拟抽样
        sum_x += x
        lnL = sum_x * np.log(p1 / p0) + (n - sum_x) * np.log((1 - p1) / (1 -
p0))
        # 决策判断
        if lnL >= lnA:
            return n, sum_x  # 拒绝
        elif lnL <= lnB:
            return n, sum_x  # 接受
```

```
# 情形 1：95%信度下的 SPRT（拒收）
min_sample_size_95, critical_defective_95 = sprt_min_sample_size(p0, p1_high,
lnA_high, lnB_high, p0)

# 情形 2：90%信度下的 SPRT（接收）
min_sample_size_90, critical_defective_90 = sprt_min_sample_size(p0, p1_low,
lnA_low, lnB_low, p0)

print("SPRT 方法结果：")
print(f"情形 1（95%信度）最小样本量：{min_sample_size_95}，次品数临界值：
{critical_defective_95}")
print(f"情形 2（90%信度）最小样本量：{min_sample_size_90}，次品数临界值：
{critical_defective_90}")
print("\n")

# ================================
# 比较两种方法的性能
# ================================

print("性能比较：")
print(f"情形 1（95%信度）：")
print(f"- 假设检验法最小样本量：{n_alpha_95}，次品数临界值：{critical_value_95}")
print(f"- SPRT 方法最小样本量：{min_sample_size_95}，次品数临界值：
{critical_defective_95}")

print(f"\n 情形 2（90%信度）：")
print(f"- 假设检验法最小样本量：{n_alpha_90}，次品数临界值：{critical_value_90}")
print(f"- SPRT 方法最小样本量：{min_sample_size_90}，次品数临界值：
{critical_defective_90}")
print("\n")

# ================================
# 可视化比较
# ================================

# 为了更全面地比较两种方法的性能，绘制 OC 曲线和 ASN 曲线

# 定义 SPRT 模拟函数，用于绘制曲线
```

```
def sprt_simulation(p0, p1, lnA, lnB, true_p_range, num_simulations=1000):
    oc_curve = []
    asn_curve = []
    for true_p in true_p_range:
        decisions = []
        sample_sizes = []
        for _ in range(num_simulations):
            n = 0
            sum_x = 0
            while True:
                n += 1
                x = np.random.rand() < true_p  # 模拟抽样
                sum_x += x
                lnL = sum_x * np.log(p1 / p0) + (n - sum_x) * np.log((1 - p1)
/ (1 - p0))
                if lnL >= lnA:
                    decisions.append('reject')
                    sample_sizes.append(n)
                    break
                elif lnL <= lnB:
                    decisions.append('accept')
                    sample_sizes.append(n)
                    break
        oc_curve.append(decisions.count('accept') / num_simulations)
        asn_curve.append(np.mean(sample_sizes))
    return np.array(oc_curve), np.array(asn_curve)

# 定义真实次品率范围
true_p_range = np.arange(0.05, 0.26, 0.01)

# 情形 1：95%信度下的 SPRT 性能曲线
oc_curve_high, asn_curve_high = sprt_simulation(p0, p1_high, lnA_high,
lnB_high, true_p_range)

# 情形 2：90%信度下的 SPRT 性能曲线
oc_curve_low, asn_curve_low = sprt_simulation(p0, p1_low, lnA_low, lnB_low,
true_p_range)

# 绘制 OC 曲线（情形 1）
```

```
plt.figure()
plt.plot(true_p_range, oc_curve_high, 'b-', linewidth=2, label='SPRT OC Curve')
plt.axvline(x=p0, color='r', linestyle='--', linewidth=1.5, label='p0')
plt.axvline(x=p1_high, color='g', linestyle='--', linewidth=1.5, label='p1')
plt.title('Operating Characteristic (OC) Curve - 95% Confidence')
plt.xlabel('True Defective Rate')
plt.ylabel('Acceptance Probability')
plt.legend(loc='best')
plt.grid(True)
plt.show()

# 绘制 ASN 曲线（情形 1）
plt.figure()
plt.plot(true_p_range, asn_curve_high, 'g-', linewidth=2, label='SPRT ASN Curve')
plt.axvline(x=p0, color='r', linestyle='--', linewidth=1.5, label='p0')
plt.axvline(x=p1_high, color='g', linestyle='--', linewidth=1.5, label='p1')
plt.title('Average Sample Number (ASN) Curve - 95% Confidence')
plt.xlabel('True Defective Rate')
plt.ylabel('Average Sample Size')
plt.legend(loc='best')
plt.grid(True)
plt.show()

# 绘制 OC 曲线（情形 2）
plt.figure()
plt.plot(true_p_range, oc_curve_low, 'b-', linewidth=2, label='SPRT OC Curve')
plt.axvline(x=p0, color='r', linestyle='--', linewidth=1.5, label='p0')
plt.axvline(x=p1_low, color='g', linestyle='--', linewidth=1.5, label='p1')
plt.title('Operating Characteristic (OC) Curve - 90% Confidence')
plt.xlabel('True Defective Rate')
plt.ylabel('Acceptance Probability')
plt.legend(loc='best')
plt.grid(True)
plt.show()

# 绘制 ASN 曲线（情形 2）
plt.figure()
```

```
plt.plot(true_p_range, asn_curve_low, 'g-', linewidth=2, label='SPRT ASN Curve')
plt.axvline(x=p0, color='r', linestyle='--', linewidth=1.5, label='p0')
plt.axvline(x=p1_low, color='g', linestyle='--', linewidth=1.5, label='p1')
plt.title('Average Sample Number (ASN) Curve - 90% Confidence')
plt.xlabel('True Defective Rate')
plt.ylabel('Average Sample Size')
plt.legend(loc='best')
plt.grid(True)
plt.show()

# 输出 OC 和 ASN 曲线在两种情况下 p0 和 p1 点处的值
def get_curve_values(true_p_range, oc_curve, asn_curve, p0, p1):
    p0_index = np.abs(true_p_range - p0).argmin()
    p1_index = np.abs(true_p_range - p1).argmin()
    p0_oc_value = oc_curve[p0_index]
    p1_oc_value = oc_curve[p1_index]
    p0_asn_value = asn_curve[p0_index]
    p1_asn_value = asn_curve[p1_index]
    return p0_oc_value, p1_oc_value, p0_asn_value, p1_asn_value

# 情形 1（95%信度）OC 和 ASN 曲线在 p0 和 p1_high 点处的值
p0_oc_value_high, p1_oc_value_high, p0_asn_value_high, p1_asn_value_high = get_curve_values(true_p_range, oc_curve_high, asn_curve_high, p0, p1_high)
print(f"情形 1（95%信度）OC 和 ASN 曲线在 p0 和 p1_high 点处的值:")
print(f"- p0 = {p0}: OC = {p0_oc_value_high}, ASN = {p0_asn_value_high}")
print(f"- p1_high = {p1_high}: OC = {p1_oc_value_high}, ASN = {p1_asn_value_high}")

# 情形 2（90%信度）OC 和 ASN 曲线在 p0 和 p1_low 点处的值
p0_oc_value_low, p1_oc_value_low, p0_asn_value_low, p1_asn_value_low = get_curve_values(true_p_range, oc_curve_low, asn_curve_low, p0, p1_low)
print(f"情形 2（90%信度）OC 和 ASN 曲线在 p0 和 p1_low 点处的值:")
print(f"- p0 = {p0}: OC = {p0_oc_value_low}, ASN = {p0_asn_value_low}")
print(f"- p1_low = {p1_low}: OC = {p1_oc_value_low}, ASN = {p1_asn_value_low}")
```

问题二：

Python 代码

```
from itertools import product
import pandas as pd
import numpy as np

def z(status, x_1, x_2, x_3, x_4):
    p_1 = status[1]  # 零配件 1 次品率
    p_2 = status[4]  # 零配件 2 次品率
    p_f = status[7]  # 成品次品率
    b_1 = status[2]  # 零配件 1 单价
    b_2 = status[5]  # 零配件 2 单价
    a_f = status[8]  # 装配成本
    d_1 = status[3]  # 零配件 1 检测成本
    d_2 = status[6]  # 零配件 2 检测成本
    d_f = status[9]  # 成品检测成本
    s_f = status[10]  # 市场售价
    r_f = status[11]  # 调换损失
    m_f = status[12]  # 拆解费用
    C_b = (b_1 + d_1) / (1 - p_1) * x_1 + (b_2 + d_2) / (1 - p_2) * x_2 + b_1
* (1 - x_1) + b_2 * (1 - x_2)  # 购买零配件成本
    C_a = a_f  # 装配成本
    p = 1 - (1 - p_f) * (1 - p_1) ** (1 - x_1) * (1 - p_2) ** (1 - x_2)  # 装
配完成的产品次品率
    C_d = d_f * x_3  # 成品检测成本
    C_r = r_f * p * (1 - x_3)  # 调换成本
    C_m = m_f * p * x_4  # 拆解成本
    I_s = s_f * (1 - p)  # 销售收入
    I_m = b_1 / (1 - p_1) * p * x_1 * x_4 + b_2 / (1 - p_2) * p * x_2 * x_4
+ (b_1 / (1 - p_1) * (p - p_1) - d_1 * p) * (1 - x_1) * x_4 + (b_2 / (1 -
p_2) * (p - p_2) - d_2 * p) * (1 - x_2) * x_4  # 拆解收入
    return I_s + I_m - C_b - C_d - C_r - C_m


status = np.array(pd.read_csv("status.csv"))  #该文件为题目表 1（除去标题）
for statu in status:
    max_z = None
    optimal_solution = None
    for x_1, x_2, x_3, x_4 in product([0, 1], repeat=4):
```

```
        z_ = z(statu, x_1, x_2, x_3, x_4)
        if not max_z or (max_z and max_z < z_):
            max_z = z_
            optimal_solution = [x_1, x_2, x_3, x_4]
    print(f"情况：{int(statu[0])} 最优解：{optimal_solution} max_z = {max_z}")
```

问题二计算结果：

情况	x_1	x_2	x_3	x_4	总利润 z
1	0	0	0	0	11.198
1	0	0	0	1	12.668
1	0	0	1	0	9.824
1	0	0	1	1	11.294
1	0	1	0	0	10.886667
1	0	1	0	1	13.756667
1	0	1	1	0	9.026667
1	0	1	1	1	11.896667
1	1	0	0	0	13.553333
1	1	0	0	1	14.677778
1	1	0	1	0	11.693333
1	1	0	1	1	12.817778
1	1	1	0	0	13.8
1	1	1	0	1	15.744444
1	1	1	1	0	11.4
1	1	1	1	1	13.344444
2	0	0	0	0	-2.256
2	0	0	0	1	0.784
2	0	0	1	0	-2.328
2	0	0	1	1	0.712
2	0	1	0	0	-2.57
2	0	1	0	1	3.81

2	0	1	1	0	-3.41
2	0	1	1	1	2.97
2	1	0	0	0	2.18
2	1	0	0	1	4.7
2	1	0	1	0	1.34
2	1	0	1	1	3.86
2	1	1	0	0	3.85
2	1	1	0	1	8.35
2	1	1	1	0	2.05
2	1	1	1	1	6.55
3	0	0	0	0	4.694
3	0	0	0	1	6.164
3	0	0	1	0	9.824
3	0	0	1	1	11.294
3	0	1	0	0	6.326667
3	0	1	0	1	9.196667
3	0	1	1	0	9.026667
3	0	1	1	1	11.896667
3	1	0	0	0	8.993333
3	1	0	0	1	10.117778
3	1	0	1	0	11.693333
3	1	0	1	1	12.817778
3	1	1	0	0	11.4
3	1	1	0	1	13.344444
3	1	1	1	0	11.4
3	1	1	1	1	13.344444
4	0	0	0	0	-13.968
4	0	0	0	1	-9.464
4	0	0	1	0	-1.328

4	0	0	1	1	3.176
4	0	1	0	0	-8.71
4	0	1	0	1	-1.97
4	0	1	1	0	0.09
4	0	1	1	1	6.83
4	1	0	0	0	-5.21
4	1	0	0	1	-1.97
4	1	0	1	0	3.59
4	1	0	1	1	6.83
4	1	1	0	0	2.8
4	1	1	0	1	7.3
4	1	1	1	0	6.8
4	1	1	1	1	11.3
5	0	0	0	0	4.768
5	0	0	0	1	4.38
5	0	0	1	0	6.288
5	0	0	1	1	5.9
5	0	1	0	0	9.71
5	0	1	0	1	11.915
5	0	1	1	0	9.61
5	0	1	1	1	11.815
5	1	0	0	0	0.186667
5	1	0	0	1	1.551111
5	1	0	1	0	0.986667
5	1	0	1	1	2.351111
5	1	1	0	0	6.316667
5	1	1	0	1	8.511111
5	1	1	1	0	5.316667
5	1	1	1	1	7.511111

6	0	0	0	0	18.58675
6	0	0	0	1	14.313625
6	0	0	1	0	17.013
6	0	0	1	1	12.739875
6	0	1	0	0	17.459737
6	0	1	0	1	15.412105
6	0	1	1	0	15.434737
6	0	1	1	1	13.387105
6	1	0	0	0	19.249211
6	1	0	0	1	16.367237
6	1	0	1	0	17.224211
6	1	0	1	1	14.342237
6	1	1	0	0	18.278947
6	1	1	0	1	17.436842
6	1	1	1	0	15.778947
6	1	1	1	1	14.936842

问题二计算得到最优结果：

情况	x_1	x_2	x_3	x_4	总利润 z
1	1	1	0	1	15.744
2	1	1	0	1	8.350
3	1	1	0	1	13.344
4	1	1	1	1	11.300
5	0	1	0	1	11.915
6	1	0	0	0	19.249

问题三：

Python 代码

```
from deap import base, creator, tools, algorithms
```

```
import random

def to_child_arg(child_args, parent_arg, decision_args):
    true_child_args = []
    if isinstance(decision_args[0], list):
        decision_arg = decision_args[0]
        for i, child_arg in enumerate(child_args):
            if isinstance(child_arg, dict):
                 true_child_args.append(to_child_arg(child_arg["child_args"],
child_arg["parent_arg"], decision_args[i + 1]))
            else:
                true_child_args.append(child_arg)
    else:
        decision_arg = decision_args
        true_child_args = child_args

    # 装配成本
    C_a = parent_arg[1]
    # 计算父配件生产次品率
    p = 1 - parent_arg[0]
    for i, x_i in enumerate(decision_arg[1:]):
        p = p * (1 - true_child_args[i][0]) ** (1 - x_i)
    p = 1 - p
    # 拆解成本
    C_m = parent_arg[3] * p * decision_arg[0]
    # 购买子配件成本和回收收益
    C_b = I_m = 0
    for i, child_arg in enumerate(true_child_args):
         C_b = C_b + (child_arg[1] + child_arg[2]) / (1 - child_arg[0]) *
decision_arg[i + 1] + child_arg[1] * (1 - decision_arg[i + 1])
        I_m = I_m + child_arg[1] / (1 - child_arg[0]) * p * decision_arg[i +
1] * decision_arg[0] + (child_arg[1] / (1 - child_arg[0]) * (p - child_arg[0])
- child_arg[2] * p) * (1 - decision_arg[i + 1]) * decision_arg[0]
    cost = C_b + C_a + C_m - I_m
    return [p, cost, parent_arg[2]]

def z(decision_args):
    # parent_arg: [次品率, 装配成本, 检测成本, 拆解费用]
    # true_child_arg: [次品率, 购买单价, 检测成本]
```

```
    # fake_child_arg: {parent_arg，child_args}
    args = {
        "parent_arg": [0.1, 8, 6, 10],  # 成品
        "child_args": [
            {
                "parent_arg": [0.1, 8, 4, 6],  # 半成品 1
                "child_args": [[0.1, 2, 1], [0.1, 8, 1], [0.1, 12, 2]],  # 零
配件 1，零配件 2，零配件 3
            },
            {
                "parent_arg": [0.1, 8, 4, 6],  # 半成品 2
                "child_args": [[0.1, 2, 1], [0.1, 8, 1], [0.1, 12, 2]],  # 零
配件 4，零配件 5，零配件 6
            },
            {
                "parent_arg": [0.1, 8, 4, 6],  # 半成品 3
                "child_args": [[0.1, 8, 1], [0.1, 12, 2]],  # 零配件 7，零配件 8
            },
        ],
    }
    x_f = decision_args[0]
    s_f = 200  # 市场售价
    r_f = 40  # 调换损失
          decision_args    =    [decision_args[1:5],    decision_args[5:9],
decision_args[9:13], decision_args[13:16]]
     p,  cost,  d_f  =  to_child_arg(args["child_args"],  args["parent_arg"],
decision_args)
    C_d = d_f * x_f  # 成品检测成本
    C_r = r_f * p * (1 - x_f)  # 调换成本
    I_s = s_f * (1 - p)
    return (I_s - C_d - C_r - cost,)

def evaluate(individual):
    return z(individual)

def main():
    # 目标为最大化
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMax)
```

```
    toolbox = base.Toolbox()
    toolbox.register("attr_int", random.randint, 0, 1)
    # 共16 个变量，依次为[成品检测，成品拆解，半成品 1 检测，半成品 2 检测，半成品 3 检测，半成品 1 拆解，零配件 1 检测，零配件 2 检测，零配件 3 检测，半成品 2 拆解，零配件 4 检测，零配件 5 检测，零配件 6 检测，半成品 3 拆解，零配件 7 检测，零配件 8 检测]
    toolbox.register("individual",   tools.initRepeat,   creator.Individual, toolbox.attr_int, n=16)
    toolbox.register("population",          tools.initRepeat,          list, toolbox.individual)
    toolbox.register("evaluate", evaluate)
    toolbox.register("mate", tools.cxTwoPoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    population = toolbox.population(n=50)
    # 使用 logbook 记录每一代的最适应值
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("max", max)
    _, logbook = algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2, ngen=50, stats=stats, verbose=False)

    print(logbook)
    # 输出最优解
    words = ["成品检测", "成品拆解", "半成品 1 检测", "半成品 2 检测", "半成品 3 检测", "半成品 1 拆解", "零配件 1 检测", "零配件 2 检测", "零配件 3 检测", "半成品 2 拆解", "零配件 4 检测", "零配件 5 检测", "零配件 6 检测", "半成品 3 拆解", "零配件 7 检测", "零配件 8 检测"]
    best_individual = tools.selBest(population, 1)[0]
    print("最优决策为: ")
    for i, x_i in enumerate(best_individual):
        print(f'{words[i]}: {"是" if x_i else "否"}')
    print("最优目标函数值: ", best_individual.fitness.values)

if __name__ == "__main__":
    main()
```

问题三计算结果：

gen	nevals	max

0	50	38.61139
1	33	43.88272
2	30	47.49438
3	28	47.49438
4	31	49.01413
5	29	49.01413
6	30	49.01413
7	29	49.6284
8	32	51.2716
9	31	51.2716
10	33	51.2716
11	41	51.64444
12	33	51.64444
13	27	51.64444
14	30	51.64444
15	37	51.64444
16	29	51.64444
17	32	51.64444
18	31	51.64444
19	39	51.64444
20	29	51.64444

最优决策为:

成品检测：否

成品拆解：是

半成品 1 检测：是

半成品 2 检测：是

半成品 3 检测：是

半成品 1 拆解：是

零配件 1 检测：是

零配件 2 检测：是

零配件 3 检测：是

半成品 2 拆解：是

零配件 4 检测：是

零配件 5 检测：是

零配件 6 检测：是

半成品 3 拆解：是

零配件 7 检测：是

零配件 8 检测：是

最优目标函数值： (51.64444)

问题四：

问题 2 保守决策 Python 代码

```
from itertools import product
import pandas as pd
import numpy as np
from scipy.stats import beta


def calc_upper_bound(defective_percentage):
    n = 200  # 总样本数
    n_d = n * defective_percentage  # 根据抽样次品率计算抽样次品样本数
    # 后验分布参数
    alpha_post = n_d + 1
    beta_post = n - n_d + 1
    alpha = 0.1  # 90% 置信度
    # 计算上限
    return beta.ppf(1 - alpha / 2, alpha_post, beta_post)


def z(status, x_1, x_2, x_3, x_4):
    p_1 = calc_upper_bound(status[1])  # 零配件 1 次品率
    p_2 = calc_upper_bound(status[4])  # 零配件 2 次品率
    p_f = calc_upper_bound(status[7])  # 成品次品率
    b_1 = status[2]  # 零配件 1 单价
```

```
    b_2 = status[5]  # 零配件 2 单价
    a_f = status[8]  # 装配成本
    d_1 = status[3]  # 零配件 1 检测成本
    d_2 = status[6]  # 零配件 2 检测成本
    d_f = status[9]  # 成品检测成本
    s_f = status[10]  # 市场售价
    r_f = status[11]  # 调换损失
    m_f = status[12]  # 拆解费用
    C_b = (b_1 + d_1) / (1 - p_1) * x_1 + (b_2 + d_2) / (1 - p_2) * x_2 + b_1 * (1 - x_1) + b_2 * (1 - x_2)  # 购买零配件成本
    C_a = a_f  # 装配成本
    p = 1 - (1 - p_f) * (1 - p_1) ** (1 - x_1) * (1 - p_2) ** (1 - x_2)  # 装配完成的产品次品率
    C_d = d_f * x_3  # 成品检测成本
    C_r = r_f * p * (1 - x_3)  # 调换成本
    C_m = m_f * p * x_4  # 拆解成本
    I_s = s_f * (1 - p)  # 销售收入
    I_m = b_1 / (1 - p_1) * p * x_1 * x_4 + b_2 / (1 - p_2) * p * x_2 * x_4 + (b_1 / (1 - p_1) * (p - p_1) - d_1 * p) * (1 - x_1) * x_4 + (b_2 / (1 - p_2) * (p - p_2) - d_2 * p) * (1 - x_2) * x_4  # 拆解收入
    return I_s + I_m - C_b - C_d - C_r - C_m - C_a


status = np.array(pd.read_csv("status.csv"))  # 该文件为题目表 1（除去标题）
for statu in status:
    max_z = None
    optimal_solution = None
    for x_1, x_2, x_3, x_4 in product([0, 1], repeat=4):
        z_ = z(statu, x_1, x_2, x_3, x_4)
        if not max_z or (max_z and max_z < z_):
            max_z = z_
            optimal_solution = [x_1, x_2, x_3, x_4]
    print(f"情况：{int(statu[0])} 最优解：{optimal_solution} max_z = {max_z}")
```

计算结果

情况	x_1	x_2	x_3	x_4	z
1	1	1	0	1	12.710
2	1	1	0	1	4.505

3	1	1	1	1	10.558
4	1	1	1	1	8.016
5	0	1	1	1	7.387
6	1	1	0	0	15.085

问题 2 期望决策 Python 代码

```
from itertools import product
import pandas as pd
import numpy as np
from scipy.stats import beta

def calc_distribution(defective_percentage):
    n = 200  # 总样本数
    n_d = n * defective_percentage  # 根据抽样次品率计算抽样次品样本数
    # 后验分布参数
    alpha_post = n_d + 1
    beta_post = n - n_d + 1
    return alpha_post, beta_post

def z(status, x_1, x_2, x_3, x_4):
    p_1 = status[1]  # 零配件 1 次品率
    p_2 = status[4]  # 零配件 2 次品率
    p_f = status[7]  # 成品次品率
    b_1 = status[2]  # 零配件 1 单价
    b_2 = status[5]  # 零配件 2 单价
    a_f = status[8]  # 装配成本
    d_1 = status[3]  # 零配件 1 检测成本
    d_2 = status[6]  # 零配件 2 检测成本
    d_f = status[9]  # 成品检测成本
    s_f = status[10]  # 市场售价
    r_f = status[11]  # 调换损失
    m_f = status[12]  # 拆解费用
    C_b = (b_1 + d_1) / (1 - p_1) * x_1 + (b_2 + d_2) / (1 - p_2) * x_2 + b_1
* (1 - x_1) + b_2 * (1 - x_2)  # 购买零配件成本
    C_a = a_f  # 装配成本
    p = 1 - (1 - p_f) * (1 - p_1) ** (1 - x_1) * (1 - p_2) ** (1 - x_2)  # 装
配完成的产品次品率
```

```
    C_d = d_f * x_3  # 成品检测成本
    C_r = r_f * p * (1 - x_3)  # 调换成本
    C_m = m_f * p * x_4  # 拆解成本
    I_s = s_f * (1 - p)  # 销售收入
    I_m = b_1 / (1 - p_1) * p * x_1 * x_4 + b_2 / (1 - p_2) * p * x_2 * x_4
+ (b_1 / (1 - p_1) * (p - p_1) - d_1 * p) * (1 - x_1) * x_4 + (b_2 / (1 -
p_2) * (p - p_2) - d_2 * p) * (1 - x_2) * x_4  # 拆解收入
    return I_s + I_m - C_b - C_d - C_r - C_m - C_a

def expected_z(status, x_1, x_2, x_3, x_4):
    N = 100000
    p_samples = [[], [], []]
    z_samples = []
    for i, j in enumerate([1, 4, 7]):
        alpha, beta = calc_distribution(status[j])
        p_samples[i] = np.random.beta(alpha, beta, N)
    for i in range(N):
        status_sample = status.copy()
        for j, k in enumerate([1, 4, 7]):
            status_sample[k] = p_samples[j][i]
        z_samples.append(z(status_sample, x_1, x_2, x_3, x_4))
    return np.mean(np.array(z_samples))

status = np.array(pd.read_csv("status.csv"))  # 该文件为题目表1（除去标题）
for statu in status:
    max_z = None
    optimal_solution = None
    for x_1, x_2, x_3, x_4 in product([0, 1], repeat=4):
        z_ = expected_z(statu, x_1, x_2, x_3, x_4)
        if not max_z or (max_z and max_z < z_):
            max_z = z_
            optimal_solution = [x_1, x_2, x_3, x_4]
    print(f"情况：{int(statu[0])} 最优解：{optimal_solution} max_z = {max_z}")
```

计算结果

情况	x_1	x_2	x_3	x_4	z
1	1	1	0	1	15.435
2	1	1	0	1	8.093

3	1	1	1	1	13.062
4	1	1	1	1	11.084
5	0	1	0	1	11.436
6	1	0	0	0	18.657

问题 3 保守决策 Python 代码

```
from deap import base, creator, tools, algorithms
import random
from scipy.stats import beta

def calc_upper_bound(defective_percentage):
    n = 200  # 总样本数
    n_d = n * defective_percentage  # 根据抽样次品率计算抽样次品样本数
    # 后验分布参数
    alpha_post = n_d + 1
    beta_post = n - n_d + 1
    alpha = 0.1  # 90% 置信度
    # 计算上限
    return beta.ppf(1 - alpha / 2, alpha_post, beta_post)

def to_child_arg(child_args, parent_arg, decision_args):
    true_child_args = []
    if isinstance(decision_args[0], list):
        decision_arg = decision_args[0]
        for i, child_arg in enumerate(child_args):
            if isinstance(child_arg, dict):
                true_child_args.append(to_child_arg(child_arg["child_args"],
child_arg["parent_arg"], decision_args[i + 1]))
            else:
                true_child_args.append(child_arg)
    else:
        decision_arg = decision_args
        true_child_args = child_args

    # 装配成本
    C_a = parent_arg[1]
    # 计算父配件生产次品率
```

```
    p = 1 - parent_arg[0]
    for i, x_i in enumerate(decision_arg[1:]):
        p = p * (1 - true_child_args[i][0]) ** (1 - x_i)
    p = 1 - p
    # 拆解成本
    C_m = parent_arg[3] * p * decision_arg[0]
    # 购买子配件成本和回收收益
    C_b = I_m = 0
    for i, child_arg in enumerate(true_child_args):
         C_b = C_b + (child_arg[1] + child_arg[2]) / (1 - child_arg[0]) *
decision_arg[i + 1] + child_arg[1] * (1 - decision_arg[i + 1])
        I_m = I_m + child_arg[1] / (1 - child_arg[0]) * p * decision_arg[i +
1] * decision_arg[0] + (child_arg[1] / (1 - child_arg[0]) * (p - child_arg[0])
- child_arg[2] * p) * (1 - decision_arg[i + 1]) * decision_arg[0]
    cost = C_b + C_a + C_m - I_m
    return [p, cost, parent_arg[2]]

def z(decision_args):
    # parent_arg: [次品率, 装配成本, 检测成本, 拆解费用]
    # true_child_arg: [次品率, 购买单价, 检测成本]
    # fake_child_arg: {parent_arg, child_args}
    p = calc_upper_bound(0.1)
    args = {
        "parent_arg": [p, 8, 6, 10],  # 成品
        "child_args": [
            {
                "parent_arg": [p, 8, 4, 6],  # 半成品 1
                "child_args": [[p, 2, 1], [p, 8, 1], [p, 12, 2]],  # 零配件 1，
零配件 2，零配件 3
            },
            {
                "parent_arg": [p, 8, 4, 6],  # 半成品 2
                "child_args": [[p, 2, 1], [p, 8, 1], [p, 12, 2]],  # 零配件 4，
零配件 5，零配件 6
            },
            {
                "parent_arg": [p, 8, 4, 6],  # 半成品 3
                "child_args": [[p, 8, 1], [p, 12, 2]],  # 零配件 7，零配件 8
            },
```

```
        ],
    }
    x_f = decision_args[0]
    s_f = 200  # 市场售价
    r_f = 40  # 调换损失
          decision_args    =    [decision_args[1:5],    decision_args[5:9],
decision_args[9:13], decision_args[13:16]]
     p,  cost,  d_f  =  to_child_arg(args["child_args"],  args["parent_arg"],
decision_args)
    C_d = d_f * x_f  # 成品检测成本
    C_r = r_f * p * (1 - x_f)  # 调换成本
    I_s = s_f * (1 - p)
    return (I_s - C_d - C_r - cost,)

def evaluate(individual):
    return z(individual)

def main():
    # 目标为最大化
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMax)

    toolbox = base.Toolbox()
    toolbox.register("attr_int", random.randint, 0, 1)
    # 共16 个变量，依次为[成品检测，成品拆解，半成品 1 检测，半成品 2 检测，半成品 3 检测，
半成品 1 拆解，零配件 1 检测，零配件 2 检测，零配件 3 检测，半成品 2 拆解，零配件 4 检测，零配件
5 检测，零配件 6 检测，半成品 3 拆解，零配件 7 检测，零配件 8 检测]
      toolbox.register("individual",  tools.initRepeat,  creator.Individual,
toolbox.attr_int, n=16)
             toolbox.register("population",     tools.initRepeat,     list,
toolbox.individual)
    toolbox.register("evaluate", evaluate)
    toolbox.register("mate", tools.cxTwoPoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    population = toolbox.population(n=50)
    # 使用 logbook 记录每一代的最适应值
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("max", max)
```

```
    _ = algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2, ngen=50,
stats=stats, verbose=True)
    # 输出最优解
    words = ["成品检测", "成品拆解", "半成品 1 检测", "半成品 2 检测", "半成品 3 检测",
"半成品 1 拆解", "零配件 1 检测", "零配件 2 检测", "零配件 3 检测", "半成品 2 拆解", "零配
件 4 检测", "零配件 5 检测", "零配件 6 检测", "半成品 3 拆解", "零配件 7 检测", "零配件 8 检
测"]
    best_individual = tools.selBest(population, 1)[0]
    print("最优决策为: ")
    for i, x_i in enumerate(best_individual):
        print(f'{words[i]}: {"是" if x_i else "否"}')
    print("最优目标函数值: ", best_individual.fitness.values)

if __name__ == "__main__":
    main()
```

问题 3 期望决策 Python 代码

```
from deap import base, creator, tools, algorithms
import random
import numpy as np
from scipy.stats import beta

def calc_distribution(defective_percentage):
    n = 200  # 总样本数
    n_d = n * defective_percentage  # 根据抽样次品率计算抽样次品样本数
    # 后验分布参数
    alpha_post = n_d + 1
    beta_post = n - n_d + 1
    return alpha_post, beta_post

def to_child_arg(child_args, parent_arg, decision_args):
    true_child_args = []
    if isinstance(decision_args[0], list):
        decision_arg = decision_args[0]
        for i, child_arg in enumerate(child_args):
            if isinstance(child_arg, dict):
                true_child_args.append(to_child_arg(child_arg["child_args"],
child_arg["parent_arg"], decision_args[i + 1]))
```

```
            else:
                true_child_args.append(child_arg)
    else:
        decision_arg = decision_args
        true_child_args = child_args

    # 装配成本
    C_a = parent_arg[1]
    # 计算父配件生产次品率
    p = 1 - parent_arg[0]
    for i, x_i in enumerate(decision_arg[1:]):
        p = p * (1 - true_child_args[i][0]) ** (1 - x_i)
    p = 1 - p
    # 拆解成本
    C_m = parent_arg[3] * p * decision_arg[0]
    # 购买子配件成本和回收收益
    C_b = I_m = 0
    for i, child_arg in enumerate(true_child_args):
         C_b = C_b + (child_arg[1] + child_arg[2]) / (1 - child_arg[0]) *
decision_arg[i + 1] + child_arg[1] * (1 - decision_arg[i + 1])
        I_m = I_m + child_arg[1] / (1 - child_arg[0]) * p * decision_arg[i +
1] * decision_arg[0] + (child_arg[1] / (1 - child_arg[0]) * (p - child_arg[0])
- child_arg[2] * p) * (1 - decision_arg[i + 1]) * decision_arg[0]
    cost = C_b + C_a + C_m - I_m
    return [p, cost, parent_arg[2]]

def z(decision_args, p_sample):
    # parent_arg: [次品率, 装配成本, 检测成本, 拆解费用]
    # true_child_arg: [次品率, 购买单价, 检测成本]
    # fake_child_arg: {parent_arg, child_args}
    p = p_sample
    args = {
        "parent_arg": [p[0], 8, 6, 10],  # 成品
        "child_args": [
            {
                "parent_arg": [p[1], 8, 4, 6],  # 半成品 1
                "child_args": [[p[2], 2, 1], [p[3], 8, 1], [p[4], 12, 2]],  #
零配件 1，零配件 2，零配件 3
            },
```

```
            {
                "parent_arg": [p[5], 8, 4, 6],  # 半成品 2
                "child_args": [[p[6], 2, 1], [p[7], 8, 1], [p[8], 12, 2]],  #
零配件 4，零配件 5，零配件 6
            },
            {
                "parent_arg": [p[9], 8, 4, 6],  # 半成品 3
                "child_args": [[p[10], 8, 1], [p[11], 12, 2]],  # 零配件 7，零配
件 8
            },
        ],
    }
    x_f = decision_args[0]
    s_f = 200  # 市场售价
    r_f = 40  # 调换损失
          decision_args    =    [decision_args[1:5],    decision_args[5:9],
decision_args[9:13], decision_args[13:16]]
     p,  cost,  d_f  =  to_child_arg(args["child_args"],  args["parent_arg"],
decision_args)
    C_d = d_f * x_f  # 成品检测成本
    C_r = r_f * p * (1 - x_f)  # 调换成本
    I_s = s_f * (1 - p)
    return (I_s - C_d - C_r - cost,)

def evaluate(individual):
    N = 10000
    p_samples = []
    z_samples = []
    alpha, beta = calc_distribution(0.1)
    for _ in range(12):
        p_samples.append(np.random.beta(alpha, beta, N))
    for i in range(N):
        p_sample = [p_samples[j][i] for j in range(12)]
        z_samples.append(z(individual, p_sample))
    return (np.mean(np.array(z_samples)),)

def main():
    # 目标为最大化
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
```

```
    creator.create("Individual", list, fitness=creator.FitnessMax)

    toolbox = base.Toolbox()
    toolbox.register("attr_int", random.randint, 0, 1)
    # 共16 个变量，依次为[成品检测，成品拆解，半成品 1 检测，半成品 2 检测，半成品 3 检测，半成品 1 拆解，零配件 1 检测，零配件 2 检测，零配件 3 检测，半成品 2 拆解，零配件 4 检测，零配件 5 检测，零配件 6 检测，半成品 3 拆解，零配件 7 检测，零配件 8 检测]
      toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_int, n=16)
             toolbox.register("population", tools.initRepeat, list, toolbox.individual)
    toolbox.register("evaluate", evaluate)
    toolbox.register("mate", tools.cxTwoPoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    population = toolbox.population(n=50)
    # 使用 logbook 记录每一代的最适应值
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("max", max)
    _ = algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2, ngen=30, stats=stats, verbose=True)
    # 输出最优解
    words = ["成品检测", "成品拆解", "半成品 1 检测", "半成品 2 检测", "半成品 3 检测", "半成品 1 拆解", "零配件 1 检测", "零配件 2 检测", "零配件 3 检测", "半成品 2 拆解", "零配件 4 检测", "零配件 5 检测", "零配件 6 检测", "半成品 3 拆解", "零配件 7 检测", "零配件 8 检测"]
    best_individual = tools.selBest(population, 1)[0]
    print("最优决策为: ")
    for i, x_i in enumerate(best_individual):
        print(f'{words[i]}: {"是" if x_i else "否"}')
    print("最优目标函数值: ", best_individual.fitness.values)

if __name__ == "__main__":
    main()
```