基于状态机动态选择策略的生产流程决策模型

摘要

生产流程决策是提高企业生产效益和实现产品质量保障的关键。本文从优化策略角度出发，面向企业生产的电子产品，对购买零配件次品率进行可信度评估，并针对企业生产遇到的情况而进行生产阶段的决策，以最小化成本和最大化利润，为企业提供了在多生产阶段的不同成本条件下决策制定思路。

针对问题一，首先利用**莱维-林德伯格中心极限定理**，将足够大的满足二项分布的样本近似为正态分布从而建立统计量，然后在给定零配件次品率标称值和置信度条件下，通过**假设检验**确定了最小样本量与允许次品数的不等关系。最后，采用**蒙特卡罗方法**对方案的有效性进行了验证，最终得到在 95%的信度下拒收零配件的最小抽样次数为 **71**；在 90%的信度下接收零配件的最小抽样次数为 **40**。

针对问题二，首先划分出零配件 1、2 的检测、成品的检测与不合格成品的拆解 4 个决策点(设置为进行该决策为 1，否则为 0)，其次通过建立**状态机动态选择策略**，成功将两零配件进行绑定，并通过状态转移机制，得到多个策略间的动态变化关系，接着在解绑零配件后，又给出**状态机补充决策**，综合考虑零配件购买比例，并建立**新的评价指标** ROI，从而得到最优化决策方案。然后，通过引入**遗传算法**对生产流程各阶段进行模拟，将四个决策点纳入群体，得到各种情况下适应性最大的最优化决策。最后，综合两种优化策略，得到针对六种情况下两个零配件和成品生产流程的最佳策略，六种情况的决策分别为 1101、1111、1111、1111、0111、0000。

针对问题三，需要对　道工序、　个零配件生产和销售步骤进行决策，由于所需工序较多，对整体进行考虑需要大量的分类讨论，增加模型的复杂性，所以考虑将整个工序按照问题二的模型进行分解，**将整体的工序分解为几部分**，在问题二模型的基础上进行改进，确定每一部分的目标函数计算表达式，然后根据前后因素间的影响关系对部分进行组合，需要考虑零部件、半成品和成品是否检测对后续成品的次品率的影响。最后**比较全局目标函数**的大小，最终选取目标函数利润值最高的情况作为最终的决策方案，最后得到的最佳决策为 111100，即**对零件 1-8 进行检测，对成品进行检测，对检测出的不合格品和成品不进行拆解**。

针对问题四，我们考虑了次品率在实际生产生活中的**波动性**，通过**蒙特卡罗模拟方法**计算了在**不同置信度下的次品率置信区间**，通过随机模拟在置信区间范围内取出 5 个随机数，分别对问题二和问题三的情形进行计算，将每种决策情况算得的净利润取平均值作为最终的决策指标，并以此为基础，对问题二和问题三的决策方案进行了调整，以更贴近实际生产需求，最终得到针对问题二决策为 1101、1111、1111、1111、0111、0000，针对问题三结果为 111101、111100、101101、101100、110101。

关键词：状态机动态选择策略 遗传算法 蒙特卡罗模拟 莱维-林德伯格中心极限定理 假设检验

一、问题重述

某企业生产某种畅销的电子产品，需要分别购买两种零配件（零配件 1 和零配件 2），在企业将两个零配件装配成成品。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。对于不合格成品，企业可以选择报废，或者对其进行拆解，拆解过程不会对零配件造成损坏，但需要花费拆解费用。

问题一：它设定了零配件 1 和零配件 2 的次品率不会超过标称值，并给出了本题的标称值为 10%。要求我们采用抽样检测方法决定是否购买这批零配件，在给定信度要求的情况下，为企业设计检验次数尽可能少的检测方案。

问题二：它设定了企业生产的各阶段，并且给定了零配件 1/零配件 2 的次品率、购买单价、检测成本，成品的次品率、装配成本、检测成本、市场售价以及不合格成品的调换损失、拆解费用。要求我们针对表 1 的六种情形进行零配件 1 和/或零配件 2 是否检测、装配好的成品是否检测、不合格成品是否拆解四项的决策，给出决策的依据与指标。

问题三：它将问题二中的一道工序，2 个零配件延伸到m 道工序、n 个零配件，并且给定了零配件、半成品和成品的次品率、购买单价、检测成本、市场售价以及不合格成品的调换损失、拆解费用等参数。要求我们针对表 2 进行零配件是否检测、装配好的成品是否检测、不合格（半）成品等决策，给出决策的依据与指标。

问题四：它设定了问题二和问题三中零配件、半成品和成品的次品率均是通过抽样检测方法得到的,需要在此基础上替换问题二问题三的次品率，然后针对表 1 和表 2 的其他数据进行零配件是否检测、装配好的成品是否检测、不合格（半）成品等决策，给出决策的依据与指标。

二、问题分析

本题的研究对象是生产决策方案，研究的内容是企业生产决策最优化方案以及相关依据、指标结果。它实际上是一道决策分析和优化问题，要求我们得出按照信度要求的次品率、在给定生产步骤下，考虑各阶段任务选择及其对下一阶段的影响，最终达到使决策方案利润最大化的目标。

2.1 问题一分析

对于问题一，分析企业选择最优化的抽样检测方案，给定了零配件次品率的标称值 p_0=0.1,并给定了两种情形下的置信度 α，问题的关键是设计一个抽样检测方案，使得检测次数尽可能少，同时能够根据检测结果做出是否接收零配件的决策。首先由于产品总数未知，但是所有抽取样本 x_i 均满足二项分布，且所有的样本独立并具有相同分布，因此假设抽取样本量 n 足够大，根据莱维-林德伯格中心极限定理，可得到服从标准正态分布的统计量。

然后在给定置信度 α 的情况下，采用假设检验的方法，确定原假设和备择假设，通过查阅标准正态分布表，可得两种情况下的分位点 Z_α，题目给出的情况均是单侧比较，因此可得到单边检验的拒绝域，从而推导出抽取最小样本数与允许次品数的不等式关系。

最后，使用蒙特卡罗方法进行验证，设定总样本数 N，通过选取多组随机数，在规定最大容忍次品数和最小样本数的条件下，对抽样检测进行进行 m 次随机模拟，比较抽取样品

中次品数与样品，获得对应情况下的信度水平，从而验证最终结果的正确性。

2.2 问题二分析

对于问题二，分析企业生产的各阶段流程，给定了各产品对应项目的费用，针对企业生产过程对零配件 1 和/或零配件 2 是否检测、装配好的成品是否检测、不合格成品是否拆解这四项问题进行决策，问题的关键在于设计一个全面的评价体系，使得最终的利润与成本投入比最大。针对流程进行分析，发现如果对不合格成品进行回收，可以充分利用零件数，带来可能利益的更大化，但是模型的约束条件难以确定。针对这种各因素互相联系并且存在迭代解的情况，首先确定针对四个决策变量和产品购买比例确定目标函数，采用状态机动态选择策略,为了简化模型，首先将零配件 1 和零配件 2 进行绑定，即同量购买并且同进行检测决策，此时我们只考虑三个决策变量，可以将 8 种状态进行分类分析，其中四种可以直接求解，另外四种可以按照计算概率，进行递归或递推迭代求解，并且 101 会动态转变为 001，111 会动态转变为 011，并且非终止状态最终可能会向终止状态转变。分类后各因素条件相互独立，规定好约束条件后，求解相对容易，得到针对八种状态的选择。

然后，可以聚焦零配件 1 和零配件 2 的次品率，购买单价及检测成本，针对二者的检测方案以及购买比进行决策，最终根据定义的投资回报率得到最优决策的选择。

接着，针对复杂的决策，我们使用了遗传算法进行生产流程各阶段的模拟，定义完整的问题参数，根据生产流程写出适应性函数，设置出遗传算法的参数进行模拟，经过初始化种群，轮盘赌选择，交叉，变异，最终得到各情况针对四个决策变量的最优决策方案。

最后，结合状态动态选择方法和遗传算法，针对六种情况，综合两种方法的决策结果和实际情况进行最优决策的选择。

2.3 问题三分析

对于问题三，需要对　道工序、　个零配件重复问题二中的生产和销售步骤，给出决策方案，由于所需工序较多，对整体进行考虑需要大量的分类讨论，增加模型的复杂性，所以考虑将整个工序按照问题二的模型进行分解，例如，对于给出的两道工序、8 个零配件具体组装情形，可以将其分解为半成品 1 的组装、半成品 2 的组装、半成品 3 的组装、成品的组装、成品卖出后的调换几部分。然后在问题二模型的基础上进行改进，确定每一部分的目标函数计算表达式，然后进行组合，以比较全局目标函数的大小，组合的时候需要考虑零部件、半成品和成品是否检测对后续成品的次品率的影响，最终选取目标函数利润值最高的情况作为最终的决策方案。

2.4 问题四分析

对于问题四，问题四在问题二和问题三基础上提出假设：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。根据问题二和问题三中已经给出的各个零配件、半成品和成品次品率，现在已经知道它们的真值，但在实际的决策中，次品率因为常常有一个波动的范围，为了使次品率更加贴近真实值，需要通过抽样检测的方法进行计算，从而更加符合实际生产生活的需要。在问题一中，我们已经使用蒙特卡罗方法进行验证，我们可以在问题一的基础上，继续使用蒙特卡罗模拟来实现抽样检测，计算出实际在 95%置信度下的次品率范

围，然后利用随机模拟在范围内取出几个数，将这几个次品率下计算出的利润平均值作为目标函数，从而实现问题二和问题三中的决策。

三、模型假设

（1）假设问题一中每次抽样结果相互独立，且都遵循同样的二项分布 B（1，0.1）；
（2）假设问题一中抽取的样本数足够大；
（3）假设问题一中信度水平在满足信度要求后保持上升趋势；
（4）假设问题二中零配件 1 和零配件 2 进行绑定后再解除绑定，过程中对情况 5 分析没有影响；

四、符号约定

符号	说明	单位
N	总样本数	-
n	抽取样本数/最小抽取次数	-
m	蒙特卡洛模拟次数	-
p_0	次品率标称值	-
X	最大容忍次品数	-
R	信度水平	-
E	模拟计算时最大容忍次品数与次品数的大小比较总和	-
α	显著性水平	-
P	信度	-
A	投资成本	-
W	净利润	-
gen	遗传算法遗传代数	-
Q	拆解产品产生的费用	-

五、模型的建立与求解

5.1 问题一的求解

5.1.1 统计量的建立

设 x_i 表示第 i 次抽取一件产品中次品所占的数量，且均为独立具有相同分布的随机变量序列。在本题中，x_i 均服从二项分布 B（1，0.1），具有有限的数学期望与方差，E（x_i）= p_0

=0.1,D（x_i）= p_0（1-p_0）。根据莱维-林德伯格定理，当抽取样本数 n 充分大时，随机变量

$$X=\sum_{i=1}^{n}x_i$$

近似地服从正态分布 N(nμ，nσ²)。
设统计量

$$Z=\frac{\frac{X}{n}-E(x_i)}{\sqrt{D(x_i)}}\sqrt{n}=\frac{\frac{X}{n}-p_0}{\sqrt{p_0(1-p_0)}}\sqrt{n}$$

则 Z 近似服从标准正态分布 N（0，1）。

5.1.2 假设检验分析与不等关系建立

5.1.2.1 假设检验分析

在建立检验统计量后，针对企业设定的两种情形，提出原假设 H_0 和备择假设 H_1：
H_0（原假设）：在α(0<α<1)的显著性水平下，x<=10%，即接受这批零部件；
H_1（备择假设）：在α的显著性水平下，x> 10%，即拒收这批零部件。
在给定α的基础上，由标准正态分布表得到临界值，该单边检验的拒绝域为

$$Z=\frac{\frac{X}{n}-p_0}{\sqrt{p_0(1-p_0)}}\sqrt{n}\geq Z_\alpha$$

5.1.2.2 不等关系建立

由此可得抽取总样本量 n 时最大容忍次品数 X，即抽取最小样本数与最大容忍次品数的不等式关系：

$$X\geq[\frac{z_\alpha\sqrt{p_0(1-p_0)}}{\sqrt{n}}+p_0]n$$

X 应向上取到整数。
在已有的不等关系基础上，我们计算在给定α条件下，最大容忍次品数 X=1,2,...20 时，对应的最小抽检次数。

表 1 在给定α下最大容忍次品数与最小抽检次数的关系系数表

X	1	2	3	4	5	6	7	8	9	10
α =0.05	2	6	11	17	23	29	36	42	49	56
α =0.1	3	7	13	19	26	33	40	47	54	62

X	11	12	13	14	15	16	17	18	19	20
α =0.05	64	71	79	86	94	101	109	117	125	133

α =0.1	70	77	85	93	101	109	117	125	134	142

表 1 具体描述了最小抽检次数的对应关系，当显著性水平α=0.05 时，当最大容忍次品数 X=10 时，可以计算出最小抽检次数 n=56，也就是当抽检内容中出现了 10 件次品，只有当抽检次数>=56,接收此批零件，而抽检次数<56,拒绝接收此批零件。同理，针对显著性水平α=0.1 也是如此。

5.1.3 蒙特卡罗验证

蒙特卡罗方法又称统计模拟法，是一种以概率和统计理论方法为基础的一种随机模拟方法，通过使用随机数（或更常见的伪随机数）来解决计算问题。将所求解的问题同一定的概率模型相联系，用计算机实现统计模拟或抽样，以获得问题的近似解，针对问题一，根据得到的可容忍次品数与最小抽样次数的不等关系，可使用蒙特卡罗方法。

5.1.3.1 参数规定

通过可容忍次品数与最小抽样次数按照蒙特卡洛法对抽取到的次品数进行模拟计算，

首先选取多组随机数，选择总样本数为 N=10000，规定次品数的具体序号。在规定最大容忍次品数和最小样本数的条件下，在此处我们模拟最大容忍次品数为 X=1,2,…,20，对抽样检测进行进行 m=1000 次随机模拟，每次进行简单抽样检测，随机抽取最小样本数 n 个产品，最终得到该次模拟的次品数，比较抽取样品中次品数与最大容忍样品数，如果满足，则进行评价记录。在完成全部抽取与评价记录后，获得对应情况下的信度水平，从而验证最终结果的正确性。

5.1.3.2 模拟计算

规定信度水平 R, 用来衡量信度的可靠性。已知模拟次数 m 和每次比较结果 E(模拟计算时最大容忍次品数与次品数的大小比较总和),那么

$$R=\frac{E}{m}$$

假设次品数小于最大容忍次品数时，记为成功一次，则 E 大小加 1.

计算步骤为：

Step.1 确定可容忍次品数与最小抽样次数，确定具体目标为记录单次模拟中抽取到的总次品数，评价指标为多次模拟后的信度水平；

Step.2 设定随机抽样的分布和参数，选定次品的具体编号；

Step.3 进行大量的随机抽样，生成符合设定分布的随机数据样本；

Step.4 根据模拟流程得到模拟结果，从而确定规定最小样本数和最大容忍次品数下的信度水平；

重复进行蒙特卡洛模拟计算多次，从而得到更稳定和可靠的结果，可以通过统计分析方法对结果进行汇总和分析

5.1.3.3 信度水平分析

根据模拟计算的结果，得到规定最小抽检次数下，显著性水平为 0.05 和 0.1 的最大容忍次品数与信度水平的关系。根据图 2 可得，整体的信度水平呈上升趋势，逐渐满足信度水平的要求。

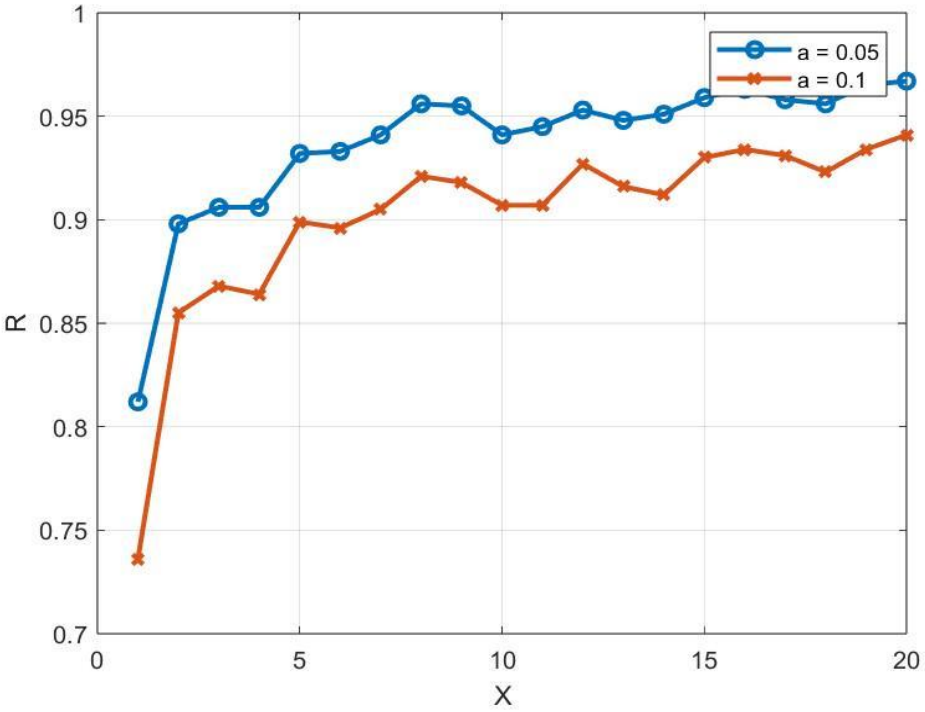

图 1 规定最小抽检次数下的最大容忍次品数与信度水平关系图

在最大容忍次品数 X 较小时，其对应的置信水平较低，不满足 95%和 90%的置信度要求，由于其对应的最小抽检次数<30，抽检样品总体不满足正态分布，不满足统计量 Z 建立的前提条件，因而信度水平脱离信度水平的要求。

表 2 规定最小抽检次数下的最大容忍次品数与信度水平系数表

X	1	2	3	4	5	6	7	8	9	10
α =0.05	0.812	0.898	0.906	0.906	0.932	0.933	0.941	0.956	0.955	0.941
α =0.1	0.736	0.855	0.868	0.864	0.899	0.896	0.905	0.921	0.918	0.907

X	11	12	13	14	15	16	17	18	19	20
α =0.05	0.945	0.953	0.948	0.951	0.959	0.963	0.958	0.956	0.965	0.967
α =0.1	0.907	0.927	0.916	0.912	0.93	0.934	0.931	0.923	0.934	0.941

由表 2 内容，判断显著性水平为 α =0.05 时，当最大容忍次品数达到 12 后，信度水平逐渐稳定在信度要求内，显著性水平为 α =0.1 时，当最大容忍次品数达到 7 后，信度水平逐渐稳定在信度要求内。

综合表 1 内容，得到结论：在α=0.05 时，只有最小抽样次数达到 71，才能保证在最大容忍次品数 X=12 时，满足信度要求；在α=0.1 时，只有最小抽检次数达到 40，才能保证在最大容忍次品数 X=7 时，满足信度要求。

5.2 问题二的求解

5.2.1 状态机动态选择策略

针对流程进行分析，发现如果对不合格成品进行回收，可以充分利用零件数，带来可能利益的更大化，但是模型的约束条件难以确定。针对这种各因素互相联系并且存在迭代解的情况，首先确定针对四个决策变量和产品购买比例确定目标函数，采用状态动态选择策略，为了简化模型，首先将零配件 1 和零配件 2 进行绑定，即同量购买并且同进行检测决策。

5.2.1.1 状态模式选择

在零配件 1 和零配件 2 绑定的基础上，此时我们只考虑三个决策变量，可以将 8 种状态进行分类分析，假设绑定零配件检测 g_1，成品检测 g_2，成品拆解 g_3，其中 1 表示执行，0 表示不执行。

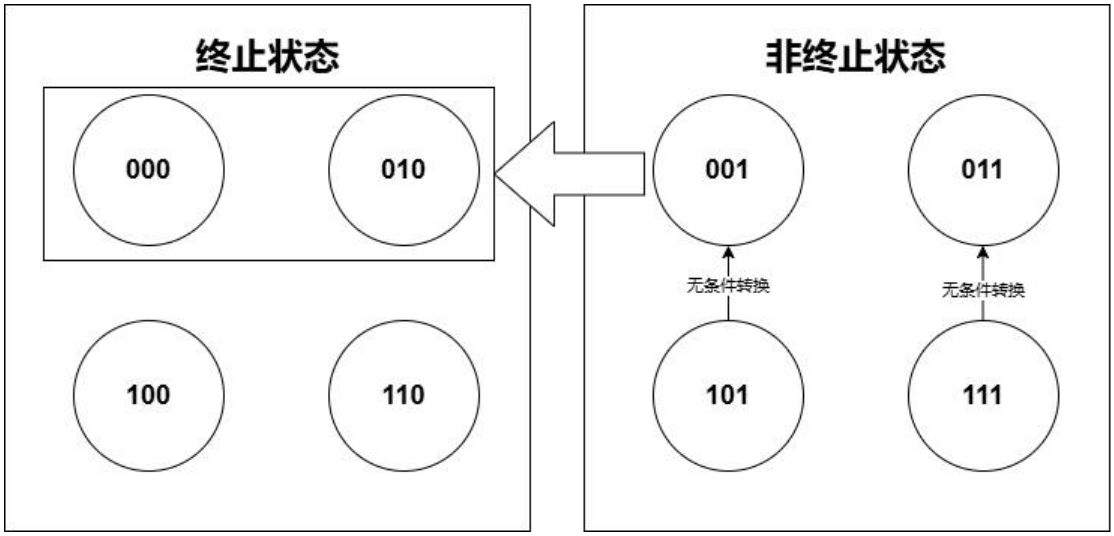

图 2 状态模式分类

其中 000、010、100、110 四种状态，为终止状态，由于不需要对不合格成品进行拆解，则在第一轮售卖后就结束，可以直接求解。另外 001、011、101、111 四种状态是非终止状态，由于选择对不合格成品进行拆解，会发生状态的自旋和状态的移动，因此需要递归或递推迭代求解。其中 101 会无条件动态转变为 001，111 会无条件动态转变为 011，非终止状态在进行本轮状态到下一状态的转变时（可能是自旋也可能是不同状态的转变），我们会主动预测下一状态是否有必要进行。状态分类后各因素条件相互独立，规定好约束条件后，求解相对容易，得到针对八种状态的选择的利润计算。

5.2.1.2 状态动态计算

针对八种情况进行分类分析，根据状态转移关系，逐个求解出其最优决策方案下的利润。其中非终止状态 101 会无条件动态转变为 001，111 会无条件动态转变为 011，同时非终止状态 001 和 011 出现自旋或者分别向 000 和 010 转换。

定义零件 1 的次品率记为 α_1，零件 2 的次品率记为 α_2,成品的次品率记为 α_3，零件 1、零件 2 的购买单价，成品的装配成本分别记为 r_1、r_2、r_3,零件 1、零件 2 和成品的检测成本分别记为 k_1、k_2、k_3,市场售价为 w,调换成本为 e，拆解成本为 f。并假设各购买了 b 个零件 1 和零件 2。

1）状态 000

本状态不进行零配件和成品的检测，且不进行不合格成品的拆解，本状态下的净利润表达式 W 为：

$$W=-b\times(r_1+r_2)-b\times r_3+a\times w-b\times \alpha_4\times e$$

其中：

$$\alpha_4=1-(1-\alpha_1)(1-\alpha_2)(1-\alpha_3)$$

2）状态 010

本状态不进行零配件的检测，但是进行成品的检测，不进行不合格成品的拆解，，本状态下的净利润表达式 W 为：

$$W=-b\times(r_1+r_2)-b\times(r_3+k_3)+b\times(1-\alpha_4)\times w$$

其中：

$$\alpha_4=1-(1-\alpha_1)(1-\alpha_2)(1-\alpha_3)$$

3）状态 100

本状态进行零配件的检测，不进行成品的检测，且不进行不合格成品的拆解，本状态下的净利润表达式 W 为：

$$W=-b\times(r_1+r_2+k_1+k_2)-b\times(1-\alpha_1)\times r_3+b\times(1-\alpha_1)\times w-b\times(1-\alpha_1)\times \alpha_3\times e$$

4）状态 110

本状态进行零配件测检测，且进行成品的检测，但不进行不合格成品的拆解，本状态下的净利润表达式 W 为：

$$W=-b\times(r_1+r_2+k_1+k_2)-b\times(1-\alpha_1)\times(r_3+k_3)+b\times(1-\alpha_1)\times(1-\alpha_3)\times w$$

5）状态 001

本状态不进行零配件和成品的检测，但进行不合格成品的拆解，，本状态下的净利润表达式 W 为：

$$W=-b\times(r_1+r_2)-b\times r_3+b\times w-Q$$

其中：

$$\alpha_4=1-(1-\alpha_1)(1-\alpha_2)(1-\alpha_3)$$

$$Q=\sum_{i=1}^{\infty}[b\times\alpha_4^{\,i}\times(f+e+r_3)-b\times\alpha_4^{\,i}\times w]$$

6）状态 011

本状态不进行零配件的检测，但进行成品的检测和不合格成品的拆解，，本状态下的净利润表达式 W 为：

$$W=-b\times(r_1+r_2)-b\times(r_3+k_3)+b\times(1-\alpha_4)\times w-Q$$

其中：

$$\alpha_4=1-(1-\alpha_1)(1-\alpha_2)(1-\alpha_3)$$

$$Q=\sum_{i=1}^{\infty}[b\times\alpha_4^{\,i}\times(f+k_3+r_3)-b\times\alpha_4^{\,i}\times(1-\alpha_4)\times w]$$

7）状态 101

本状态进行零配件的检测，但不进行成品的检测，进行不合格成品的拆解。随着过程的进行，状态 101 可能会变为 001，因为对零配件的检验将直接筛去所有的次品，让留下的零件全为正品，不需要再花费此检验费用。

本状态下的净利润表达式 W 为：

$$W=-b\times(r_1+r_2+k_1+k_2)-b\times(1-\alpha_1)\times r_3+b\times(1-\alpha_1)\times w-Q$$

其中：

$$Q=\sum_{i=1}^{\infty}\{b\times(1-\alpha_1)\times\alpha_3^{\,i}\times[f+e+k_3+r_3-(1-\alpha_3)\times w]\}$$

8）状态 111

本状态进行零配件的检测，且进行成品的检测和拆解。随着过程的进行，状态 111 可能会变为 011，因为对零配件的检验将直接筛去所有的次品，让留下的零件全为正品，不需要再花费此检验费用。

本状态下的净利润表达式 W 为：

$$W=-100\times(r_1+r_2+k_1+k_2)-100\times(1-\alpha_1)\times(r_3+k_3)+100\times(1-\alpha_1)\times(1-\alpha_3)\times w-Q$$

其中：

$$Q=\sum_{i=1}^{\infty}\{b\times(1-\alpha_1)\times\alpha_3^{\,i}\times[f+k_3+r_3-(1-\alpha_3)\times w]\}$$

根据八种状态的计算，最终得到每个状态下的利润，根据利润，我们可以进行绑定零配件是否检测、装配好的成品是否检测、不合格成品是否拆解这三个决策变量的判定。

5.2.1.3 状态机补充决策与评价指标

在绑定条件下，可以稳定求解出各决策变量的选择情况，然后我们选择解除绑定，单独考虑零配件 1 和零配件 2 的检测情况，并且结合零配件的次品率和购买单价成本，综合评定

出其他决策信息。并且给定评价指标投资回报率（ROI），ROI 是衡量投资效益的一个重要指标，它表示准确衡量利润与投资成本之间的比例关系，便于企业选择更加的决策方案。

$$ROI=\frac{W}{A}$$

我们结合状态机选择策略结合零配件 1、2 的相关情况给出约定。

首先，针对每一个零件，我们将其检测成本与购买单价的比值作为指标$1，将次品率作为指标$2,将购买单价作为指标$3。针对两个零件，选择零件 1 购买单价与零件 2 购买单价作为指标$4。

给出以下附加决策法则：

1.如果$4<0.5 或$4>2，为了充分利用单价较高零件，选择对单价低的零配件多进行购买。

2.如果$1>1.5,选择不检测该零件。并且根据$2 和$3 考虑是否增加零配件的量。

这些决策方案建立在状态动态计算的基础上，即如果未选择对零配件进行检测，综合考虑 4 项指标，在充分服从状态动态计算的基础上考虑附加决策。

如果两个零配件选择同样数量，同样决策，则以净利润作为最终决策依据；弱国两个零配件存在数量和决策上的差异，则以 ROI 作为最终决策依据。

5.2.1.4 决策方案选择

取总零件数 N=100,由于情况 5 两个零件的次品率不同，无法使用该模型，我们默认使用平均值进行计算，并且假设这种操作不影响最终模型的求解。得到绑定下的状态机动态选择策略方案选择，但是对于零配件 1，零配件 2 的具体情况没有进行决策。

表 3 状态机动态选择策略方案选择

情况		选择决策
1	1586.70	101
2	1192.16	111
3	1569.10	111
4	1591.36	111
5	1147.65	111
6	1858.68	000

表 4 状态机动态选择策略具体利润计算及排名

情况 1		情况 2		情况 3	
101	1586.70	111	1192.16	111	1569.1
111	1569.10	101	930.72	101	1349.1
100	1462.00	100	748	100	1246
001	1261.18	110	604	110	1246
110	1246.00	011	-154.546885	011	1146.81786
011	1146.82	001	-212.162623	010	982.4
000	1119.80	000	-225.6	000	469.4

010	982.40	010	-232.8	001	469.4

情况 4		情况 5		情况 6	
111	1591.36	111	1147.65	000	1858.675
110	984.00	100	919	001	1858.675
101	754.56	101	911.15	100	1846.5
100	664.00	110	834	101	1758.825
011	-5.75	011	761.696594	010	1701.3
010	-132.80	010	641.4	011	1701.3
000	-1396.80	001	491.65	111	1619.45
001	-1396.80	000	491.65	110	1609

接着对零配件 1、2 进行解绑，根据状态机补充决策下的两条法则进行判断，最终得到结论：

情况 1：二者绑定为检测，满足附加决策 1，最终选择零配件 1 检测，零配件 2 检测，适当增加零配件 1 的个数。

情况 2：二者绑定为检测，满足附加决策 1，最终选择零配件 1 检测，零配件 2 检测，适当增加零配件 1 的个数。

情况 3：二者绑定为检测，满足附加决策 1，最终选择零配件 1 检测，零配件 2 检测，适当增加零配件 1 的个数。

情况 4：二者绑定为检测，满足附加决策 1，最终选择零配件 1 检测，零配件 2 检测，适当增加零配件 1 的个数。

情况 5：二者绑定为检测，$1>2,满足附加决策 1，2，选择不检测该零件，最终选择零配件 1 不检测，零配件 2 检测，适当增加零配件 1 的个数。

情况 6：二者绑定为检测，满足附加决策 1，最终选择零配件 1 检测，零配件 2 检测，适当增加零配件 1 的个数。

具体增加零配件 1 个数应结合状态机动态计算模型，通过 ROI 作为决策目标进行具体选择。

5.2.2 遗传算法

5.2.2.1 遗传算法原理

遗传算法是一种基于自然选择原理和自然遗传机制的优化决策算法,通过模拟自然界中的生命进化机制,在人工系统中实现特定目标的优化。通过对群体的个体进行设置,根据适者生存的原则逐代进化,最终得到最优解或准最优解。它需要经历:初始群体的产生、每一个体的适应度的评估、根据适者生存的原则选择优良个体、被选出的优良个体两两配对,通过随机交叉其染色体的基因并随机变异某些染色体的基因生成下一代群体,按此方法使群体逐代进化,直到满足进化终止条件。

针对问题 2，遗传算法具体步骤如下：

初始化：首先确定群体规模，随机生成初始种群，种群相当于一组问题的决策，通过随机选择来初始化。

适应度函数评估：对每次迭代中种群中的个体进行评估。适应度函数根据生产流程进行设定，为每个个体设定数值得分，该得分反映了在迭代中个体作为决策方案的质量，适应度得分更高的个体代表更好的解，其更有可能被选择繁殖并且其性状会在下一代得到表现。我们选择了进行 gen=100，随着遗传算法的进行，解的质量会提高，适应度会增加。

选择：根据个体的适应度，选择较优的个体作为父代。选择过程针对问题采用轮盘赌选择，保留多样性，避免早熟收敛，有利于区分利润相差较小个体之间的差异。

交叉：为了创建一对新个体，通常将从当前代中选择双亲样本的部分染色体互换（交叉），以创建代表后代的两个新染色体。

变异：变异操作的目的是定期随机更新种群，将新模式引入染色体，并鼓励在解空间的未知区域进行搜索。突变可能表现为基因的随机变化，变异是通过随机改变一个或多个染色体实现的，此处设定为随机变异策略。

终止：遗传算法会重复执行上述步骤，直到迭代到 10000 代，保证解的质量满足要求或适应度不再显著提高。

5.2.2.2 遗传算法求解

（1）根据问题二中的四个决策变量（零配件 1、零配件 2 是否检测，成品是否检测，不合格成品是否拆解）,确定一种编码方法,用四位数值串表示可行解域的每一解，每一数值代表决策方案中的一种选择；

（2）设定适应度函数评估种群中个体的适应度，在本题中，使用目标函数，即净利润表达式作为适应度函数；

（3）确定进化参数群体规模 M、交叉概率 p、变异概率 P、进化终止条件。种群规模通常远大于决策变量可能组合的数量。这是因为遗传算法依赖于种群的多样性来探索解空间，并找到优化问题的解。本题将种群规模 M 设置为 50，交叉概率 p 设置为 0.5，变异概率 P 设置为 0.1，进化终止条件设为运行 10000 代之后结束。

5.2.2.3 决策方案选择

经过对群体进行 10000 次的迭代，并进行了 100 次遗传算法的频率统计，确保群体的最终结果的稳定性，最终得到的遗传算法方案选择与状态机动态选择相同。

表 5 遗传算法方案选择表

情况	最大适应度	选择决策
1	2212	1101
2	1625	1111
3	1863	1111
4	291	1111
5	1798	0111
6	2382	0000

5.2.3 综合决策评价

结合两种策略进行最终方案的选择，得到针对问题 2 的结论：

表 6 综合决策选择表

情况	选择决策
1	1101
2	1111
3	1111
4	1111
5	0111
6	0000

此外，根据状态机动态选择补充策略的两条规则，应适当提高零配件 1 的量，以减小零配件 2 的损耗，充分利用完整的零配件 2。如果两零配件数量相等，使用利润作为最优评价指标；如果两配件数量不等，使用 ROI 作为最优评价指标。

5.3 问题三的求解

5.3.1 复杂工序的分解和情况组合

5.3.2.1 组装情况分解

问题二实质上是一道工序、2 个零配件的组装情况，是问题三所需的 m 道工序、n 个零配件的最简单情况。随着 m 和 n 数量的增加，如果直接从全局考虑，需要讨论的情况数会急剧增加，增加模型的复杂度。因此，我们考虑化繁为简，将 m 道工序、n 个零配件的组装情况分解为几个一道工序、2-3 个零配件的组装情况，然后利用问题 2 的模型，算出每一部分的目标函数，然后在组合的时候考虑零配件是否检测对进而组合在一起组成最后的决策方案。

对于问题三给出的两道工序、8 个零配件的组装情况，我们考虑将其分为四部分：半成品 1 的组装（一道工序、3 个零配件）、半成品 2 的组装（一道工序、3 个零配件）、半成品 3 的组装（一道工序、2 个零配件）和成品的组装（一道工序、3 个零配件）。

5.3.2.2 各部分状态动态计算

对于每一部分采用状态动态选择策略,为了简化模型，将每部分的零配件或半成品进行绑定，即同量购买并且同进行检测决策。

5.3.2.2.1 参数设置

对每一具体决策用 0-1 变量进行表示，若执行，则该变量为 1；若不执行，则该变量为 0。

（1）零配件是否进行检测记为 a，a 为 0-1 变量；

（2）对装配好的半成品是否进行检测记为 b，b 为 0-1 变量；

（3）对检测出的不合格半成品是否进行拆解记为 c，c 为 0-1 变量；

（4）装配好的每一件成品是否进行检测记为 d，d 为 0-1 变量；

（5）检测出的不合格成品是否进行拆解记为 e，e 为 0-1 变量；

（6）用户购买的不合格品是否进行拆解记为 f，f 为 0-1 变量；

（7）设对于每一个具体情形，分别买了 100 个零件 1-零件 8；

（8）零件 1-零件 8 的次品率记为 α （α=10%）；半成品 1-3，成品的次品率记为 α_1-α_4；

（9）零件 1-零件 8 的购买单价为 r_1-r_8，半成品 1-3 的装配成本记为 r_9, 成品的装配成本记为 r_{10}；

（10）零件 1-零件 8 的检测成本分别记为 k_1-k_8；半成品的检测成本为 k_9；成品的检测成本为 k_{10}；

（11）成品的市场售价记为 w；

（12）成品的调换损失记为 g；

（13）半成品和成品的拆解费用记为 h_1, h_2。

5.3.2.2.2 各部分利润计算

组装半成品 1 的利润 W

1. 当 a_1=1、b=1 时，净利润表达式 W 为：

$$W=-100\times(r_1+r_2+r_3+k_1+k_2+k_3)-100\times(1-\alpha)\times(r_9+k_9)-Q\times c$$

其中：

$$Q=\sum_{i=1}^{\infty}[(r_9+k_9+h_1)\times 100\times(1-\alpha)\times\alpha^i]$$

2. 当 a_1=1、b=0 时，净利润表达式 W 为：

$$W=-100\times(r_1+r_2+r_3+k_1+k_2+k_3)-100\times(1-\alpha)\times r_9$$

3. 当 a_1=0、b=0 时，净利润表达式 W 为：

$$W=-100\times(r_1+r_2+r_3+r_9)$$

4. 当 a_1=0、b=1 时，净利润表达式 W 为：

$$W=-100\times(r_1+r_2+r_3)-100\times(r_9+k_9)-Q\times c$$

其中：

$$\alpha_6=1-(1-\alpha)(1-\alpha)(1-\alpha)(1-\alpha)$$

$$Q=\sum_{i=1}^{\infty}[(h_1+r_9+k_9)\times 100\times \alpha_6^{\,i}]$$

组装半成品 2 的利润 W

1. 当 a_2=1、b=1 时，净利润表达式 W 为：

$$W=-100\times(r_4+r_5+r_6+k_4+k_5+k_6)-100\times(1-\alpha)\times(r_9+k_9)-Q\times c$$

其中：

$$Q=\sum_{i=1}^{\infty}[(r_9+k_9+h_1)\times 100\times(1-\alpha)\times\alpha^{i}]$$

2. 当 a_2=1、b=0 时，净利润表达式 W 为：

$$W=-100\times(r_4+r_5+r_6+k_4+k_5+k_6)-100\times(1-\alpha)\times r_9$$

3. 当 a_2=0、b=0 时，净利润表达式 W 为：

$$W=-100\times(r_4+r_5+r_6+r_9)$$

4. 当 a_2=0、b=1 时，净利润表达式 W 为：

$$W=-100\times(r_4+r_5+r_6)-100\times(r_9+k_9)-Q\times c$$

其中：

$$\alpha_6=1-(1-\alpha)(1-\alpha)(1-\alpha)(1-\alpha)$$

$$Q=\sum_{i=1}^{\infty}[(h_1+r_9+k_9)\times 100\times \alpha_6^{\,i}]$$

组装半成品 3 的利润 W

1. 当 a_3=1、b=1 时，净利润表达式 W 为：

$$W=-100\times(r_7+r_8+k_7+k_8)-100\times(1-\alpha)\times(r_9+k_9)-Q\times c$$

其中：

$$Q=\sum_{i=1}^{\infty}[(r_9+k_9+h_1)\times 100\times(1-\alpha)\times\alpha^{i}]$$

2. 当 a_3=1、b=0 时，净利润表达式 W 为：

$$W=-100\times(r_7+r_8+k_7+k_8)-100\times(1-\alpha)\times r_9$$

3. 当 a_3=0、b=0 时，净利润表达式 W 为：

$$W=-100\times(r_7+r_8)-100\times r_9$$

4. 当 a_3=0、b=1 时，净利润表达式 W 为：

$$W=-100\times(r_7+r_8)-100\times(r_9+k_9)-Q\times c$$

其中：

$$\alpha_7=1-(1-\alpha)(1-\alpha)(1-\alpha)$$

$$Q=\sum_{i=1}^{\infty}[(h_1+r_9+k_9)\times 100\times\alpha_7^{\,i}]$$

组装成品的利润 W

1. 当 b=1、d=1 时，净利润表达式 W 为：

$$W=-100\times 3\times k_9-100\times(1-\alpha_4)\times(r_{10}+k_{10})-Q\times e$$

其中：

$$Q=\sum_{i=1}^{\infty}[(r_{10}+k_{10}+h_2)\times 100\times(1-\alpha_4)^{i}\times\alpha_4^{\,i}]$$

2. 当 b=1、d=0 时，净利润表达式 W 为：

$$W=-100\times 3\times k_9-100\times(1-\alpha_4)\times r_{10}$$

3. 当 b=0、d=0 时，净利润表达式 W 为：

$$W=-100\times r_{10}$$

4. 当 b=0、d=1 时，净利润表达式 W 为：

$$W=-100\times(r_{10}+k_{10})-Q\times e$$

其中：

$$Q=\sum_{i=1}^{\infty}[(h_2+r_{10}+k_{10})\times 100\times\alpha_4^{\,i}]$$

成本卖出后产生的利润

净利润表达式 W 为：

$$W=100\times w-100\times\alpha_4\times g-Q\times f$$

其中：

$$Q=\sum_{i=1}^{\infty}[(r_{10}+k_{10}+h_2)\times b\times\alpha_4^{\,i}-b\times(1-\alpha_4)^{i}\times\alpha_4^{\,i}]$$

5.3.2.3 各部分组合利润计算

由于零配件 1-8 和半成品 1-3 的是否检测都会对组装出的成品的次品率造成影响，因此，在组合综合决策的时候，目标函数利润不是简单的直接加和，需要考虑到前后因素的相互影响，讨论如下：

（1）a1=1，α_1=10%

（2）a1=0，α_1=1-（1-10%）^3

（3）a2=1，α_2=10%

（4）a2=1，α_2=1-（1-10%）^3

（5）a3=1，α_3=10%

（6）a3=0，α_3=1-（1-10%）^2

组合之后，成品的次品率α_4=1-（1-α_1）（1-α_2）（1-α_3）（1-10%）

5.3.2.4 综合决策结果分析

本文通过已列出的不同情况下的目标函数表达式，以及前后因素的相互影响关系，计算出了不同情况下的总利润，进行排序比较不同决策方案的利润，选择利润最大的决策情况作为最终的决策方案。

利润最大的前五种决策方案，如下表：

表 7 利润最大决策方案及其净利润表

决策情况	净利润
111100	8572.34
111101	8456.00
011100	8221.96
011101	8126.20
101100	8105.72

从计算结果可以看出，最终选择的利润最大的方案为 111100，即对零件 1-8 进行检测，对成品进行检测，对检测出的不合格品和成品不进行拆解。

5.4 问题四的求解

5.4.1 蒙特卡罗模拟实际次品率

为了通过抽样检测的方法获取实际生产生活中次品率的波动范围，考虑使用蒙特卡罗模拟，设置随机数 x_i，10%的概率为次品，90%的概率为合格品，通过 10000 次随机模拟，得到次品率的概率密度函数，然后通过模拟出的概率密度函数，找出次品率在 95%置信度下的置信区间。

10000 次随机模拟 10%次品率真值下的概率密度函数图像如下：

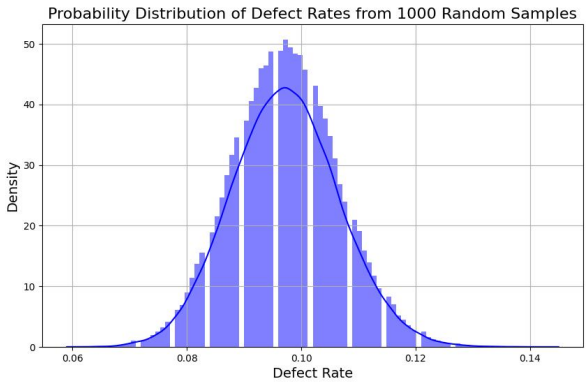

图 3 蒙特卡洛模拟 10%次品率真值下的概率密度函数

在 95%的置信度下，算得 10%次品率的置信区间为[0.0918,0.1035]

在这个置信区间内，随机抽取 5 个次品率数值，将这个数值带入到问题二和问题三的模型中，取这 5 个次品率计算所得最终目标函数利润的平均值作为决策指标，从而求出新的最优决策情况。

5.4.2 随机模拟次品率下的问题二求解

本文计算了问题二模型的基础上，每种决策情况下使用五个随机从置信区间内取出的次品率所得利润的平均值，使用该利润平均值进行新的决策方案比较。

算得每一种情况利润最大的决策方案，如下表：

表 8 随机模拟次品率下综合决策选择表

情况	决策选择	净利润
1	101	1586.99
2	111	1190.15
3	111	1569.24
4	111	1589.23
5	111	1140.98
6	100	1832.46

5.4.3 随机模拟次品率下的问题三求解

本文计算了问题三模型的基础上，每种决策情况下使用五个随机从置信区间内取出的次品率所得利润的平均值，使用该利润平均值进行新的决策方案比较。

算得利润最大的前五种决策方案，如下表：

表 9 随机模拟次品率下利润最大决策方案及其净利润表

决策情况	净利润
111101	8572.34
111100	8456.00
101101	8221.96
101100	8126.20
110101	8105.72

从计算结果可以看出，最终选择的利润最大的方案为 111101，即对零件 1-8 进行检测，对成品进行检测，对检测出的不合格品不进行拆解,对成品进行拆解。

六、模型的优缺点及改进方向

6.1 模型的优点

1.在求解最小抽检次数问题时采用蒙特卡洛模拟方法进行最小抽检次数的模拟计算，证明了求解模型的正确性。

2.在求解两种零配件和成品生产流程最优策略时采用状态机动态选择方法，将零配件 1 和零配件 2 进行绑定计算，由此简化模型，并探究状态机变化机制，从而模拟出决策之间的动态变化，得到最优解。并且综合使用遗传算法，来模拟实际生产决策的最优个体，从而得到印证状态机动态选择策略的正确性。此外，还考虑了零配件 1 和零配件 2 的关系和相关四项指标，制定出状态机动态选择补充策略，给出了零配件购买的比例关系以及不同比例下模型的评价指标。

6.2 模型的缺点

1.在求解最小抽检次数时假设抽检样品足够大，因此在实际求解时模型不能很好解释小样本量的最小抽检样品。

2.在求解两种零配件和成品生产流程最优策略时采用状态机动态选择方法时，默认第五种情况使用平均值作为两零配件误差率，结合最终补充策略，不会影响最终结果的正确性，但实际存在一定误差。

6.3 模型的推广

本文采用了状态机动态选择策略，这种来源于电子计算机底层的思维方式，可以应用到此题目的生产流程策略的动态转移关系，从而给不同策略间关系清晰的区分，可以推广到更多的流程决策问题；对于问题本身，采用了蒙特卡罗模拟方法，为该方法在生产流程中的应用，提供了新的参考。

七、参考文献

[1]吴张傲.基于遗传算法的多无人机塔杆巡检任务分配算法研究[J].电力与能源,2024,45(04):474-476+528.
[2]胡韬,周咪.软件工程视角下遗传算法与项目调度优化研究[J].黑龙江科学,2024,15(16):123-125.
[3] 田明华.基于事件驱动的审核系统任务状态流转设计[A]天津市电子工业协会 2024 年年会论文集[C].天津市电子工业协会, 天津市电子工业协会,2024: 4.
[4]韩红梅.基于统计模拟算法的工程造价预测模型[J].中国建筑金属结构,2024,23(08):38-40.
[5] 刘树东,吴昊,丛佳,顾播宇.双碳目标下基于改进型 NSGA-II 算法的港口作业调度优化算法[J].计算机应用,1-11.

八、附录

程序一 问题一代码
程序二 问题二代码
程序三 问题三代码
程序四 问题四代码

程序一 问题一代码

```
%% 正态分布，式子转换后

clc;
clear;

% 定义常数
a1 = 0.05; % 第一组置信水平
a2 = 0.1; % 第二组置信水平
p0 = 0.1; % 假设比例

% 计算正态分布的分位数
z1 = norminv(1 - a1 / 2); % 对于 a = 0.05
z2 = norminv(1 - a2 / 2); % 对于 a = 0.1
```

```
% 定义 X 的范围
X_values = 1:1:10; % 自变量 X 从 0 到 10 的范围

% 初始化 n 的存储空间
n_values_a1 = zeros(size(X_values)); % 置信水平为 0.05 时的 n
n_values_a2 = zeros(size(X_values)); % 置信水平为 0.1 时的 n

% 遍历每一个 X 值，找到相应的 n
for i = 1:length(X_values)
X = X_values(i); % 当前的 X 值
% 置信水平 a = 0.05
fun1 = @(n) (X - p0 * n)^2 - (z1^2 * p0 * (1 - p0) * n);
n_solution_a1 = fsolve(fun1, 10); % 使用初始猜测值 10
n_values_a1(i) = n_solution_a1; % 存储求解结果
% 置信水平 a = 0.1
fun2 = @(n) (X - p0 * n)^2 - (z2^2 * p0 * (1 - p0) * n);
n_solution_a2 = fsolve(fun2, 10); % 使用初始猜测值 10
n_values_a2(i) = n_solution_a2; % 存储求解结果
end

% 显示结果
disp('X values and corresponding n values for a = 0.05 and a = 0.1:');
disp(table(X_values', n_values_a1', n_values_a2', 'VariableNames', {'X',
'n_a_0_05', 'n_a_0_1'}));

% 绘制 X 与 n 的关系图
figure;
plot(X_values, n_values_a1, '-o', 'LineWidth', 2, 'DisplayName', 'a =
0.05');
hold on;
plot(X_values, n_values_a2, '-x', 'LineWidth', 2, 'DisplayName', 'a =
0.1');
xlabel('X');
ylabel('n');
title('X 与 n 的关系图（不同置信水平）');
legend('show');
grid on;

%% 正态分布，未转换式子
clc;
clear;

% 定义常数
```

```
a1 = 0.05; % 第一组置信水平
a2 = 0.1; % 第二组置信水平
p0 = 0.1; % 假设比例

% 计算正态分布的分位数
z1 = norminv(1 - a1 / 2); % 对于 a = 0.05
z2 = norminv(1 - a2 / 2); % 对于 a = 0.1

% 定义 X 的范围
X_values = 1:1:10; % 自变量 X 从 0 到 10 的范围

% 初始化 n 的存储空间
n_values_a1 = zeros(size(X_values)); % 置信水平为 0.05 时的 n
n_values_a2 = zeros(size(X_values)); % 置信水平为 0.1 时的 n

% 遍历每一个 X 值，找到相应的 n
for i = 1:length(X_values)
X = X_values(i); % 当前的 X 值
% 置信水平 a = 0.05
fun1 = @(n) (z1 * sqrt(p0 * (1 - p0)) / sqrt(n) + p0) * n - X;
n_solution_a1 = fsolve(fun1, 10); % 使用初始猜测值 10
n_values_a1(i) = n_solution_a1; % 存储求解结果
% 置信水平 a = 0.1
fun2 = @(n) (z2 * sqrt(p0 * (1 - p0)) / sqrt(n) + p0) * n - X;
n_solution_a2 = fsolve(fun2, 10); % 使用初始猜测值 10
n_values_a2(i) = n_solution_a2; % 存储求解结果
end

% 显示结果
disp('X values and corresponding n values for a = 0.05 and a = 0.1:');
disp(table(X_values', n_values_a1', n_values_a2', 'VariableNames', {'X',
'n_a_0_05', 'n_a_0_1'}));

% 绘制 X 与 n 的关系图
figure;
plot(X_values, n_values_a1, '-o', 'LineWidth', 2, 'DisplayName', 'a =
0.05');
hold on;
plot(X_values, n_values_a2, '-x', 'LineWidth', 2, 'DisplayName', 'a =
0.1');
xlabel('X');
ylabel('n');
title('X 与 n 的关系图（不同置信水平）');
legend('show');
```

```
grid on;

%% 正态分布，未转换式子取整
clc;
clear;

% 定义常数
a1 = 0.05; % 第一组置信水平
a2 = 0.1; % 第二组置信水平
p0 = 0.1; % 假设比例

% 计算正态分布的分位数
z1 = norminv(1 - a1 / 2); % 对于 a = 0.05
z2 = norminv(1 - a2 / 2); % 对于 a = 0.1

% 定义 X 的范围
X_values = 1:1:20; % 自变量 X 从 0 到 10 的范围

% 初始化 n 的存储空间
n_values_a1 = zeros(size(X_values)); % 置信水平为 0.05 时的 n
n_values_a2 = zeros(size(X_values)); % 置信水平为 0.1 时的 n

% 遍历每一个 X 值，找到相应的 n
for i = 1:length(X_values)
X = X_values(i); % 当前的 X 值
% 置信水平 a = 0.05
fun1 = @(n) (z1 * sqrt(p0 * (1 - p0)) / sqrt(n) + p0) * n - X;
n_solution_a1 = fsolve(fun1, 10); % 使用初始猜测值 10
n_values_a1(i) = ceil(n_solution_a1); % 向上取整并存储求解结果
% 置信水平 a = 0.1
fun2 = @(n) (z2 * sqrt(p0 * (1 - p0)) / sqrt(n) + p0) * n - X;
n_solution_a2 = fsolve(fun2, 10); % 使用初始猜测值 10
n_values_a2(i) = ceil(n_solution_a2); % 向上取整并存储求解结果
end

% 显示结果
disp('X values and corresponding n values for a = 0.05 and a = 0.1 (rounded
up):');
disp(table(X_values', n_values_a1', n_values_a2', 'VariableNames', {'X',
'n_a_0_05', 'n_a_0_1'}));

% 绘制 X 与 n 的关系图
figure;
```

```
plot(X_values, n_values_a1, '-o', 'LineWidth', 2, 'DisplayName', 'a =
0.05');
hold on;
plot(X_values, n_values_a2, '-x', 'LineWidth', 2, 'DisplayName', 'a =
0.1');
xlabel('X');
ylabel('n');
title('X 与 n 的关系图（不同置信水平，结果向上取整）');
legend('show');
grid on;
%% 验证
% clc,clear;

% 参数设置
total_count = 10000; % 总体数量
defective_rate = 0.10; % 次品比例
simulation_count = 1000; % 模拟次数

% 定义 n_values
% n_values_a1 = [2, 6, 11, 17, 23, 29, 36, 42, 49, 56];
% n_values_a2 = [3, 7, 13, 19, 26, 33, 40, 47, 54, 62];
X_values = 1:20; % 对应的 X 值

% 初始化结果存储
probability_a1 = zeros(size(X_values));
probability_a2 = zeros(size(X_values));

% 生成总体
total_parts = [ones(1, round(total_count * defective_rate)), zeros(1,
round(total_count * (1 - defective_rate)))];
total_parts = total_parts(randperm(total_count)); % 随机打乱总体

% 模拟过程
for i = 1:length(X_values)
X = X_values(i);
% 置信水平 a = 0.05
n_a1 = n_values_a1(i);
count_a1 = 0;
% 置信水平 a = 0.1
n_a2 = n_values_a2(i);
count_a2 = 0;
for j = 1:simulation_count
% 随机抽取
sample_a1 = total_parts(1:n_a1);
```

```
sample_a2 = total_parts(1:n_a2);
% 计算次品数
defective_count_a1 = sum(sample_a1);
defective_count_a2 = sum(sample_a2);
% 判断是否小于对应的 X
if defective_count_a1 < X
count_a1 = count_a1 + 1;
end
if defective_count_a2 < X
count_a2 = count_a2 + 1;
end
% 打乱总体以供下次模拟
total_parts = [ones(1, round(total_count * defective_rate)), zeros(1,
round(total_count * (1 - defective_rate)))];
total_parts = total_parts(randperm(total_count));
end
% 计算概率
probability_a1(i) = count_a1 / simulation_count;
probability_a2(i) = count_a2 / simulation_count;
end

% 显示结果
disp('Probability of defective count being less than X for a = 0.05 and
a = 0.1:');
disp(table(X_values', probability_a1', probability_a2',
'VariableNames', {'X', 'P_a_0_05', 'P_a_0_1'}));

% 绘制结果图
figure;
plot(X_values, probability_a1, '-o', 'LineWidth', 2, 'DisplayName', 'a
= 0.05');
hold on;
plot(X_values, probability_a2, '-x', 'LineWidth', 2, 'DisplayName', 'a
= 0.1');
xlabel('X');
ylabel('R');
title('最大容忍次数 X 与 R 的关系图');
legend('show');
grid on;

%%
% 参数设置
N = 10000; % 样本总数
n_simulations = 100000; % 模拟次数
```

```
sample_size = 10000; % 每次抽样的样本数
defect_rate = 0.05; % 次品率

% 生成 10000 个样品，次品率为 10%
sample = rand(N, 1) < defect_rate;

% 模拟抽样 100000 次，每次抽样 10000 个样本并计算次品率
pro = zeros(n_simulations, 1);
for i = 1:n_simulations
cnt = sum(sample(randi([1 N], sample_size, 1))); % 随机抽样
pro(i) = cnt / sample_size; % 计算次品率
end

% 对次品率数据进行排序
pro = sort(pro);

% 计算 95%置信区间
lower_bound = pro(round(n_simulations * 0.025));
upper_bound = pro(round(n_simulations * 0.975));

fprintf('95%% 置信区间的下分位点: %.4f\n', lower_bound);
fprintf('95%% 置信区间的上分位点: %.4f\n', upper_bound);

% 绘制次品率的概率分布
figure;
histogram(pro, 100, 'Normalization', 'pdf');
hold on;
% 绘制核密度估计
[f, xi] = ksdensity(pro);
plot(xi, f, 'LineWidth', 2);
title('10000 个随机样本次品率的概率分布', 'FontSize', 16);
xlabel('次品率', 'FontSize', 14);
ylabel('概率密度', 'FontSize', 14);
grid on;
hold off;
```

程序二 问题二代码

```
import random
from math import ceil

parameters = [
    [0.1, 4, 2, 0.1, 18, 3, 0.1, 6, 3, 56, 6, 5],
```

```
        [0.2, 4, 2, 0.2, 18, 3, 0.2, 6, 3, 56, 6, 5],
        [0.1, 4, 2, 0.1, 18, 3, 0.1, 6, 3, 56, 30, 5],
        [0.2, 4, 1, 0.2, 18, 1, 0.2, 6, 2, 56, 30, 5],
        [0.15, 4, 8, 0.15, 18, 1, 0.1, 6, 2, 56, 10, 5],
        [0.05, 4, 2, 0.05, 18, 3, 0.05, 6, 3, 56, 10, 40]
]
# a1 = 0.2
# r1 = 4
# k1 = 1
# a2 = 0.2
# r2 = 18
# k2 = 1
# a3 = 0.2
# r3 = 6
# k3 = 2
# price = 56
# change = 30
# dismantle = 5

def l000(A):
        cost, profit = 0, 0
        cost += A*r3    # 装配成本
        hege = A*(1-a1)**2*(1-a3)      # 合格
        buhege = A - hege        # 不合格
        profit += hege*price        # 获利
        cost += buhege*change      # 退换损失
        return profit - cost - A*(r1+r2)


def l010(A):
        cost, profit = 0, 0
        cost += A*r3    # 装配成本
        cost += A*k3    # 检验成本
        hege = A*(1-a1)**2*(1-a3)      # 合格
        buhege = A - hege        # 不合格
        profit += hege*price        # 获利
        return profit - cost - A*(r1+r2)


def l100(A):
        cost, profit = 0, 0
        cost += A*(k1+k2)      # 检测零件成本
        A = A*(1-a1)
        cost += A*r3    # 装配成本
```

```
    hege = A*(1-a3)        # 合格
    buhege = A - hege
    profit = hege*price
    cost += buhege*change
    return profit - cost - A*(r1+r2)


def l110(A):
    cost, profit = 0, 0
    cost += A*(k1+k2)   # 检验成本
    A = A-A*a1
    cost += A*r3   # 装配成本
    cost += A*k3   # 检验成本
    hege = A*(1-a3)    # 合格
    buhege = A - hege       # 不合格
    profit += hege*price       # 获利
    return profit - cost - A*(r1+r2)

def l011(A):   # 0 1 1
    global a1, a2, a3
    Cost = 0
    Profit = 0
    cost = 0
    profit = 0
    error = A * a1   # 本身存在的不合格数
    Acount = A
    while 1:
        cost += A * (r3 + k3)   # 装配成本 r+检测成本 k
        Cost += cost
        hege = A * (1 - a1) * (1 - a2) * (1 - a3)   # 合格品数量
        buhege = A - hege   # 不合格数量
        profit += hege * price   # 获利
        Profit += profit
        # 对下一轮推测，是否进行下一循环
        a1 = error / buhege   # 换概率
        a2 = a1
        lasthege = buhege * (1 - a1) * (1 - a2) * (1 - a3)   # 下一轮合格品数量
        lastprofit = lasthege * price   # 下一轮预计获利
        lastcost = buhege * dismantle + buhege * r3 + buhege * k3
        if lastprofit < lastcost:
            return Profit - Cost - (r1 + r2) * Acount   # 最后的利润
        Cost += buhege * dismantle
        cost, profit = 0, 0
        A = buhege   # 用于下一轮
```

```
            if buhege <= 1:
                return Profit - Cost - (r1 + r2) * Acount    # 最后的利润


# print(l011(100))


def l001(A):    # 0 0 1
    global a1, a2, a3
    Cost = 0
    Profit = 0
    cost = 0
    profit = 0
    error = A * a1    # 本身存在的不合格数
    Acount = A
    while 1:
        cost += A * r3    # 装配成本
        Cost += cost    # 更新成本
        hege = A * (1 - a1) * (1 - a2) * (1 - a3)    # 合格品数量
        buhege = A - hege    # 不合格数量
        profit += hege * price    # 获利
        Profit += profit    # 更新获利
        cost += buhege * change    # 不合格品的调换价
        # 对下一轮预测
        a1 = error / buhege
        a2 = a1
        lasthege = buhege * (1 - a1) * (1 - a2) * (1 - a3)    # 合格品数量
        lastbuhege = buhege - lasthege
        if lasthege * price < buhege * dismantle + lastbuhege * change + buhege * r3:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润
        Cost += buhege * dismantle
        cost, profit = 0, 0
        if buhege <= 1:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润
        A = buhege    # 用于下一轮


# print(l001(100))


def l101(A):    # 1 0 1
    global a1, a2, a3
    Cost = 0
    Profit = 0
```

```
    cost = 0
    profit = 0
    error = A * a1    # 本身存在的不合格数
    cost += A * (k1 + k2)    # 加上检测费用
    Acount = A
    A = A - error
    while 1:
        cost += A * r3    # 装配成本
        hege = A * (1 - a3)
        buhe = A - hege
        profit += hege * price    # 获利
        cost += buhe * change    # 不合格品的调换价
        Cost += cost
        Profit += profit
        # 对下一轮进行预测
        lasthege = buhe * (1 - a3)
        lastbuhege = buhe - lasthege
        lastprofit = lasthege * price
        lastcost = lastbuhege * change + buhe * dismantle + buhe * r3
        if lastprofit < lastcost:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润
        Cost += buhe * dismantle
        cost, profit = 0, 0
        A = buhe    # 用于下一轮
        if buhe <= 1:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润


# print(l101(100))


def l111(A):    # 1 1 1

    global a1, a2, a3
    Cost = 0
    Profit = 0
    cost = 0
    profit = 0
    error = A * a1    # 本身存在的不合格数
    cost += A * (k1 + k2)    # 加上检测费用
    A = A - error
    Acount = A
    while 1:
        cost += A * r3    # 装配成本
```

```
            cost += A * k3    # 检验成本 k
            hege = A * (1 - a3)
            buhe = A - hege
            profit += hege * price    # 获利
            Profit, Cost = profit, cost
            # 预测，判断是佛拆卸
            lasthege = buhe * (1 - a3)
            lastbuhe = buhe - lasthege
            lastprofit = lasthege * price
            lastcost = buhe * dismantle + buhe * k3 + buhe * r3
            if lastprofit < lastcost:
                return Profit - Cost - (r1 + r2) * Acount    # 最后的利润
            Cost += buhe * dismantle
            cost, profit = 0, 0
            A = buhe    # 用于下一轮
            if buhe <= 1:
                return Profit - Cost - (r1 + r2) * Acount    # 最后的利润


# print(l111(100))
# score = []
# score.append([str(0) + str(0) + str(0), front(100, 0, 0, 0)])
# score.append([str(0) + str(1) + str(0), front(100, 0, 1, 0)])
# score.append([str(1) + str(0) + str(0), front(100, 1, 0, 0)])
# score.append([str(1) + str(1) + str(0), front(100, 1, 1, 0)])
#
# score.append([str(0) + str(0) + str(1), front(100, 0, 0, 1)])
# score.append([str(0) + str(1) + str(1), front(100, 0, 1, 1)])
# score.append([str(1) + str(0) + str(1), front(100, 1, 0, 1)])
# score.append([str(1) + str(1) + str(1), front(100, 1, 1, 1)])
# 对结果进行排序
# sorted_data = sorted(score, key=lambda x: x[1], reverse=True)
# print(sorted_data)

result = []
for i in range(6):
    para = parameters[i]
    a1 = para[0]
    r1 = para[1]
    k1 = para[2]
    a2 = para[3]
    r2 = para[4]
    k2 = para[5]
    a3 = para[6]
```

```
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score = []
score.append(['000', l000(100)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
r2 = para[4]
k2 = para[5]
a3 = para[6]
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score.append(['010', l010(100)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
r2 = para[4]
k2 = para[5]
a3 = para[6]
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score.append(['100', l100(100)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
r2 = para[4]
```

```
k2 = para[5]
a3 = para[6]
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score.append(['110', l110(100)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
r2 = para[4]
k2 = para[5]
a3 = para[6]
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score.append(['001', l001(100)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
r2 = para[4]
k2 = para[5]
a3 = para[6]
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score.append(['011', l011(100)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
```

```
        r2 = para[4]
        k2 = para[5]
        a3 = para[6]
        r3 = para[7]
        k3 = para[8]
        price = para[9]
        change = para[10]
        dismantle = para[11]
        score.append(['101', l101(100)])

        para = parameters[i]
        a1 = para[0]
        r1 = para[1]
        k1 = para[2]
        a2 = para[3]
        r2 = para[4]
        k2 = para[5]
        a3 = para[6]
        r3 = para[7]
        k3 = para[8]
        price = para[9]
        change = para[10]
        dismantle = para[11]
        score.append(['111', l111(100)])
        sorted_data = sorted(score, key=lambda x: x[1], reverse=True)
        # print(sorted_data)
        result.append(sorted_data[0])
print(result)
```

程序三 问题三代码

```
Num = 100
alpha = 0.1
r1, r2, r3, r4, r5, r6, r7, r8, r9, r10 = 2, 8, 12, 2, 8, 12, 8, 12, 8, 8
k1, k2, k3, k4, k5, k6, k7, k8, k9, k10 = 1, 1, 2, 1, 1, 2, 1, 2, 4, 6
w = 200
g = 40
h1, h2 = 6, 10

xuanze = ['000', '001(0)', '001(1)', '110', '111(0)', '111(1)']
```

```
# 组装半成品 1 的利润 W
def Get_half1_W00():
    return -100 * (r1 + r2 + r3 + r9)


def Get_half1_W01(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha2 = 1 - (1 - alpha) ** 4
    while n:
        if (r9 + k9 + h1) * 100 * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * alpha2 ** n
            n += 1
    return -100 * (r1 + r2 + r3) - 100 * (r9 + k9) - Q * c


def Get_half1_W10():
    return -100 * (r1 + r2 + r3 + k1 + k2 + k3) - 100 * (1 - alpha) * r9


def Get_half1_W11(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    while n:
        # print(1, Q)
        if (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n
            n += 1
    return -100 * (r1 + r2 + r3 + k1 + k2 + k3) - 100 * (1 - alpha) * (r9 + k9) - Q * c


result = [Get_half1_W00(), Get_half1_W01(0), Get_half1_W01(1), Get_half1_W10(),
Get_half1_W11(0),
          Get_half1_W11(1)]
print(result)
G1 = [max(Get_half1_W00(), Get_half1_W01(0), Get_half1_W01(1)), max(Get_half1_W10(),
Get_half1_W11(0),
```

```
Get_half1_W11(1))]


# 组装半成品 2 的利润 W
def Get_half2_W11(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    while n:
        # print(2, Q)
        if (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n
            n += 1
    return -100 * (r4 + r5 + r6 + k4 + k5 + k6) - 100 * (1 - alpha) * (r9 + k9) - Q * c


def Get_half2_W01(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha2 = 1 - (1 - alpha) ** 4
    while n:
        if (r9 + k9 + h1) * 100 * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * alpha2 ** n
            n += 1
    return -100 * (r4 + r5 + r6) - 100 * (r9 + k9) - Q * c


def Get_half2_W10():
    return -100 * (r4 + r5 + r6 + k4 + k5 + k6) - 100 * (1 - alpha) * r9


def Get_half2_W00():
    return -100 * (r4 + r5 + r6 + r9)


result = [Get_half2_W00(), Get_half2_W01(0), Get_half2_W01(1), Get_half2_W10(),
Get_half2_W11(0),
          Get_half2_W11(1)]
```

```
print(result)
G2 = [max(Get_half2_W00(), Get_half2_W01(0), Get_half2_W01(1)), max(Get_half2_W10(),
Get_half2_W11(0),

Get_half2_W11(1))]


# 组装半成品 3 的利润 W
def Get_half3_W11(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    while n:
        # print(2, Q)
        if (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n
            n += 1
    return -100 * (r7 + r8 + k7 + k8) - 100 * (1 - alpha) * (r9 + k9) - Q * c


def Get_half3_W01(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha2 = 1 - (1 - alpha) ** 3
    while n:
        if (r9 + k9 + h1) * 100 * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * alpha2 ** n
            n += 1
    return -100 * (r7 + r8) - 100 * (r9 + k9) - Q * c


def Get_half3_W10():
    return -100 * (r7 + r8 + k7 + k8) - 100 * (1 - alpha) * r9


def Get_half3_W00():
    return -100 * (r7 + r8) - 100 * r9
```

```
result = [Get_half3_W00(), Get_half3_W01(0), Get_half3_W01(1), Get_half3_W10(), Get_half3_W11(0),
          Get_half3_W11(1)]
print(result)
G3 = [max(Get_half3_W00(), Get_half3_W01(0), Get_half3_W01(1)), max(Get_half3_W10(), Get_half3_W11(0),

Get_half3_W11(1))]

# 组装成品的利润 W
def Down11(a, b, c, e):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha = 0.1
    alpha1 = 1 - (1 - alpha) ** 3
    alpha2 = 1 - (1 - (alpha if a else alpha1)) * (1 - (alpha if b else alpha1)) * (1 - (alpha if c else alpha1))
    while n:
        if (r10 + k10 + h2) * 100 * (1 - alpha2) ** n * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r10 + k10 + h2) * 100 * (1 - alpha2) ** n * alpha2 ** n
            n += 1
    return -100 * 3 * k9- 100 * (1-alpha2) * (r10 + k10) - Q * e


def Down10(a, b, c):
    alpha = 0.1
    alpha1 = 1 - (1 - alpha) ** 3
    alpha2 = 1 - (1 - (alpha if a else alpha1)) * (1 - (alpha if b else alpha1)) * (1 - (alpha if c else alpha1))
    return -100*3*k9 - 100*(1-alpha2)*r10


def Down00():
    return -100*r10


def Down01(a, b, c, e):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha = 0.1
```

```
    alpha1 = 1 - (1 - alpha) ** 3
    alpha2 = 1 - (1 - (alpha if a else alpha1)) * (1 - (alpha if b else alpha1)) * (1 - (alpha if c else alpha1))
    while n:
        if (r10 + k10 + h2) * 100 * (1 - alpha2) ** n < 0.000001:
            break
        else:
            Q += (r10 + k10 + h2) * 100 * (1 - alpha2) ** n
            n += 1
    return -100 * (r10 + k10) - Q * e



# print(G1)
# print(G2)
# print(G3)


# 成本卖出后产生的利润
def Get_G4(a, b, c, f):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha = 0.1
    alpha1 = 1 - (1 - alpha) ** 3
    alpha2 = 1 - (1 - (alpha if a else alpha1)) * (1 - (alpha if b else alpha1)) * (1 - (alpha if c else alpha1))
    while n:
        if (r10 + k10 + h2) * 100 * alpha2 ** n - 100 * (1 - alpha2) ** n * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r10 + k10 + h2) * 100 * alpha2 ** n - 100 * (1 - alpha2) ** n * alpha2 ** n
            n += 1
    return 100 * w - 100 * alpha2 * g - Q * f


result = []

for i in range(2):
    for j in range(2):
        for k in range(2):
            temp = G1[i] + G2[j] + G3[k]
            for o in range(2):
                for e in range(2):
```

```
                    t = max(Down11(i, j, k, e), Down00(), Down01(i, j, k, e), Down10(i, j, k))
                    result.append([f'{i}{j}{k}{o}{e}', G1[i], G2[j], G3[k], Get_G4(i, j, k, o), t,
temp + Get_G4(i, j, k, o)+ t])



for i in range(len(result)):
    print(i+1, result[i])
for i in sorted(result,    key=lambda x: x[6], reverse=True):
    print(i)
```

程序四 问题四代码

求置信度在 95%内的次品数

```
import math
import random
import matplotlib.pyplot as plt
import seaborn as sns

N = 10000
n_defect = 1000

sample = []

# 生成 10000 个样品，次品率为 10%
for i in range(N):
    if random.random() < 0.2:
        sample.append(1)
    else:
        sample.append(0)

pro = []
# 模拟抽样 100000 次，每次抽 10000 个样本并计算次品率
for i in range(100000):
    cnt = 0
    for j in range(1000):
        index = random.randint(0, N-1)
        if sample[index]:
            cnt += 1
    pro.append(cnt / 1000)
```

```
# 排序次品率列表
pro = sorted(pro)

# 输出样本中的第 250 个和第 99750 个次品率值
# print(pro[250], pro[100000-250])
# 计算 95%置信区间的上下限
lower_bound = pro[int(100000 * 0.025)]
upper_bound = pro[int(100000 * 0.975)]

print(f"95% 置信区间的下分位点: {lower_bound}")
print(f"95% 置信区间的上分位点: {upper_bound}")
# 使用 Seaborn 的 kdeplot 绘制平滑核密度估计图，同时美化图形
plt.figure(figsize=(10, 6))
sns.histplot(pro, bins=100, kde=True, color='b', stat="density", linewidth=0)
plt.title("10000 个随机样本次品率的概率分布", fontsize=16)
plt.xlabel("次品率", fontsize=14)
plt.ylabel("概率", fontsize=14)
plt.grid(True)
plt.show()
```

问题三再求解

```
import random

Num = 100
alpha = 0.1
r1, r2, r3, r4, r5, r6, r7, r8, r9, r10 = 2, 8, 12, 2, 8, 12, 8, 12, 8, 8
k1, k2, k3, k4, k5, k6, k7, k8, k9, k10 = 1, 1, 2, 1, 1, 2, 1, 2, 4, 6
w = 200
g = 40
h1, h2 = 6, 10

xuanze = ['000', '001(0)', '001(1)', '110', '111(0)', '111(1)']


# 组装半成品 1 的利润 W
def Get_half1_W00():
    return -100 * (r1 + r2 + r3 + r9)


def Get_half1_W01(c):
    n = 1
```

```
    Q = 0
    # 求 Q 的极限
    alpha2 = 1 - (1 - alpha) ** 4
    while n:
        if (r9 + k9 + h1) * 100 * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * alpha2 ** n
            n += 1
    return -100 * (r1 + r2 + r3) - 100 * (r9 + k9) - Q * c


def Get_half1_W10():
    return -100 * (r1 + r2 + r3 + k1 + k2 + k3) - 100 * (1 - alpha) * r9


def Get_half1_W11(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    while n:
        # print(1, Q)
        if (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n
            n += 1
    return -100 * (r1 + r2 + r3 + k1 + k2 + k3) - 100 * (1 - alpha) * (r9 + k9) - Q * c



def Get_G1():
    return [max(Get_half1_W00(), Get_half1_W01(0), Get_half1_W01(1)),
max(Get_half1_W10(), Get_half1_W11(0),

Get_half1_W11(1))]


# 组装半成品 2 的利润 W
def Get_half2_W11(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    while n:
```

```
        # print(2, Q)
        if (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n
            n += 1
    return -100 * (r4 + r5 + r6 + k4 + k5 + k6) - 100 * (1 - alpha) * (r9 + k9) - Q * c


def Get_half2_W01(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha2 = 1 - (1 - alpha) ** 4
    while n:
        if (r9 + k9 + h1) * 100 * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * alpha2 ** n
            n += 1
    return -100 * (r4 + r5 + r6) - 100 * (r9 + k9) - Q * c


def Get_half2_W10():
    return -100 * (r4 + r5 + r6 + k4 + k5 + k6) - 100 * (1 - alpha) * r9


def Get_half2_W00():
    return -100 * (r4 + r5 + r6 + r9)




def Get_G2():
    return [max(Get_half2_W00(), Get_half2_W01(0), Get_half2_W01(1)),
max(Get_half2_W10(), Get_half2_W11(0),

Get_half2_W11(1))]


# 组装半成品 3 的利润 W
def Get_half3_W11(c):
    n = 1
    Q = 0
```

```
    # 求 Q 的极限
    while n:
        # print(2, Q)
        if (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * (1 - alpha) * alpha ** n
            n += 1
    return -100 * (r7 + r8 + k7 + k8) - 100 * (1 - alpha) * (r9 + k9) - Q * c


def Get_half3_W01(c):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha2 = 1 - (1 - alpha) ** 3
    while n:
        if (r9 + k9 + h1) * 100 * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r9 + k9 + h1) * 100 * alpha2 ** n
            n += 1
    return -100 * (r7 + r8) - 100 * (r9 + k9) - Q * c


def Get_half3_W10():
    return -100 * (r7 + r8 + k7 + k8) - 100 * (1 - alpha) * r9


def Get_half3_W00():
    return -100 * (r7 + r8) - 100 * r9



def Get_G3():
    return [max(Get_half3_W00(), Get_half3_W01(0), Get_half3_W01(1)),
max(Get_half3_W10(), Get_half3_W11(0),

Get_half3_W11(1))]

# 组装成品的利润 W
def Down11(a, b, c, e):
    n = 1
    Q = 0
```

```
    # 求 Q 的极限
    alpha = 0.1
    alpha1 = 1 - (1 - alpha) ** 3
    alpha2 = 1 - (1 - (alpha if a else alpha1)) * (1 - (alpha if b else alpha1)) * (1 - (alpha if c else alpha1))
    while n:
        if (r10 + k10 + h2) * 100 * (1 - alpha2) ** n * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r10 + k10 + h2) * 100 * (1 - alpha2) ** n * alpha2 ** n
            n += 1
    return -100 * 3 * k9- 100 * (1-alpha2) * (r10 + k10) - Q * e


def Down10(a, b, c):
    alpha = 0.1
    alpha1 = 1 - (1 - alpha) ** 3
    alpha2 = 1 - (1 - (alpha if a else alpha1)) * (1 - (alpha if b else alpha1)) * (1 - (alpha if c else alpha1))
    return -100*3*k9 - 100*(1-alpha2)*r10


def Down00():
    return -100*r10


def Down01(a, b, c, e):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha = 0.1
    alpha1 = 1 - (1 - alpha) ** 3
    alpha2 = 1 - (1 - (alpha if a else alpha1)) * (1 - (alpha if b else alpha1)) * (1 - (alpha if c else alpha1))
    while n:
        if (r10 + k10 + h2) * 100 * (1 - alpha2) ** n < 0.000001:
            break
        else:
            Q += (r10 + k10 + h2) * 100 * (1 - alpha2) ** n
            n += 1
    return -100 * (r10 + k10) - Q * e
```

```
# print(G1)
# print(G2)
# print(G3)


# 成本卖出后产生的利润
def Get_G4(a, b, c, f):
    n = 1
    Q = 0
    # 求 Q 的极限
    alpha = 0.1
    alpha1 = 1 - (1 - alpha) ** 3
    alpha2 = 1 - (1 - (alpha if a else alpha1)) * (1 - (alpha if b else alpha1)) * (1 - (alpha if c else
alpha1))
    while n:
        if (r10 + k10 + h2) * 100 * alpha2 ** n - 100 * (1 - alpha2) ** n * alpha2 ** n < 0.000001:
            break
        else:
            Q += (r10 + k10 + h2) * 100 * alpha2 ** n - 100 * (1 - alpha2) ** n * alpha2 ** n
            n += 1
    return 100 * w - 100 * alpha2 * g - Q * f


up = 0.1035
down = 0.0918
R = None
flag = 0
for x in range(5):
    alpha = random.uniform(down, up)
    G1 = Get_G1()
    G2 = Get_G2()
    G3 = Get_G3()
    result = []
    for i in range(2):
        for j in range(2):
            for k in range(2):
                temp = G1[i] + G2[j] + G3[k]
                for o in range(2):
                    for e in range(2):
                        t = max(Down11(i, j, k, e), Down00(), Down01(i, j, k, e), Down10(i, j,
k))
                        result.append([f'{i}{j}{k}{o}{e}', temp + Get_G4(i, j, k, o) + t])
    # for i in result:
    #     print(i[1])
```

```
        if flag == 0:
            R = result
            flag = 1
        else:
            for i in range(len(result)):
                R[i].append(result[i][1])
        for i in R:
            print(i)
        print()
Avera = []
for i in range(len(R)):
        Avera.append([R[i][0], sum(R[i][1:])/len(R[i][1:])])
        print(Avera[i])
```

问题二再求解

```
import random

# 0.1 的置信区间
# 95% 置信区间的下分位点: 0.0935
# 95% 置信区间的上分位点: 0.1053

average10 = 0
for i in range(5):
        average10 += random.uniform(0.0935,0.1053)
average10 /= 5
print(average10)

# 0.2 的置信区间
# 95% 置信区间的下分位点: 0.1908
# 95% 置信区间的上分位点: 0.2066

average20 = 0
for i in range(5):
        average20 += random.uniform(0.1908,0.2066)
average20 /= 5
print(average20)


# 0.15 的置信区间
# 95% 置信区间的下分位点: 0.1491
```

```
# 95%  置信区间的上分位点: 0.1633
average15 = 0
for i in range(5):
    average15 += random.uniform(0.1491,0.1633)
average15 /= 5
print(average15)

# 0.05 的置信区间
# 95%  置信区间的下分位点: 0.0492
# 95%  置信区间的上分位点: 0.0581
average05 = 0
for i in range(5):
    average05 += random.uniform(0.0492,0.0581)
average05 /= 5
print(average05)


import random
from math import ceil

parameters = [
    [average10, 4, 2, 0.1, 18, 3, 0.1, 6, 3, 56, 6, 5],
    [average20, 4, 2, 0.2, 18, 3, 0.2, 6, 3, 56, 6, 5],
    [average10, 4, 2, 0.1, 18, 3, 0.1, 6, 3, 56, 30, 5],
    [average20, 4, 1, 0.2, 18, 1, 0.2, 6, 2, 56, 30, 5],
    [average15, 4, 8, 0.15, 18, 1, 0.1, 6, 2, 56, 10, 5],
    [average05, 4, 2, 0.05, 18, 3, 0.05, 6, 3, 56, 10, 40]
]
N = 100


# a1 = 0.2
# r1 = 4
# k1 = 1
# a2 = 0.2
# r2 = 18
# k2 = 1
# a3 = 0.2
# r3 = 6
# k3 = 2
# price = 56
# change = 30
# dismantle = 5
```

```
def l000(A):
    cost, profit = 0, 0
    cost += A * r3    # 装配成本
    hege = A * (1 - a1) ** 2 * (1 - a3)    # 合格
    buhege = A - hege    # 不合格
    profit += hege * price    # 获利
    cost += buhege * change    # 退换损失
    return profit - cost - A * (r1 + r2)


def l010(A):
    cost, profit = 0, 0
    cost += A * r3    # 装配成本
    cost += A * k3    # 检验成本
    hege = A * (1 - a1) ** 2 * (1 - a3)    # 合格
    buhege = A - hege    # 不合格
    profit += hege * price    # 获利
    return profit - cost - A * (r1 + r2)


def l100(A):
    cost, profit = 0, 0
    cost += A * (k1 + k2)    # 检测零件成本
    A = A * (1 - a1)
    cost += A * r3    # 装配成本
    hege = A * (1 - a3)    # 合格
    buhege = A - hege
    profit = hege * price
    cost += buhege * change
    return profit - cost - A * (r1 + r2)


def l110(A):
    cost, profit = 0, 0
    cost += A * (k1 + k2)    # 检验成本
    A = A - A * a1
    cost += A * r3    # 装配成本
    cost += A * k3    # 检验成本
    hege = A * (1 - a3)    # 合格
    buhege = A - hege    # 不合格
    profit += hege * price    # 获利
    return profit - cost - A * (r1 + r2)
```

```
def l011(A):    # 0 1 1
    global a1, a2, a3
    Cost = 0
    Profit = 0
    cost = 0
    profit = 0
    error = A * a1    # 本身存在的不合格数
    Acount = A
    while 1:
        cost += A * (r3 + k3)    # 装配成本 r+检测成本 k
        Cost += cost
        hege = A * (1 - a1) * (1 - a2) * (1 - a3)    # 合格品数量
        buhege = A - hege    # 不合格数量
        profit += hege * price    # 获利
        Profit += profit
        # 对下一轮推测，是否进行下一循环
        a1 = error / buhege    # 换概率
        a2 = a1
        lasthege = buhege * (1 - a1) * (1 - a2) * (1 - a3)    # 下一轮合格品数量
        lastprofit = lasthege * price    # 下一轮预计获利
        lastcost = buhege * dismantle + buhege * r3 + buhege * k3
        if lastprofit < lastcost:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润
        Cost += buhege * dismantle
        cost, profit = 0, 0
        A = buhege    # 用于下一轮
        if buhege <= 1:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润


# print(l011(100))


def l001(A):    # 0 0 1
    global a1, a2, a3
    Cost = 0
    Profit = 0
    cost = 0
    profit = 0
    error = A * a1    # 本身存在的不合格数
    Acount = A
    while 1:
        cost += A * r3    # 装配成本
```

```
            hege = A * (1 - a1) * (1 - a2) * (1 - a3)  # 合格品数量
            buhege = A - hege  # 不合格数量
            profit += hege * price  # 获利
            Profit += profit  # 更新获利
            cost += buhege * change  # 不合格品的调换价
            Cost += cost  # 更新成本
            # 对下一轮预测
            a1 = error / buhege
            a2 = a1
            lasthege = buhege * (1 - a1) * (1 - a2) * (1 - a3)  # 合格品数量
            lastbuhege = buhege - lasthege
            if lasthege * price < buhege * dismantle + lastbuhege * change + buhege * r3:
                return Profit - Cost - (r1 + r2) * Acount  # 最后的利润
            Cost += buhege * dismantle
            cost, profit = 0, 0
            if buhege <= 1:
                return Profit - Cost - (r1 + r2) * Acount  # 最后的利润
            A = buhege  # 用于下一轮


# print(l001(100))


def l101(A):  # 1 0 1
    global a1, a2, a3
    Cost = 0
    Profit = 0
    cost = 0
    profit = 0
    error = A * a1  # 本身存在的不合格数
    cost += A * (k1 + k2)  # 加上检测费用
    Acount = A
    A = A - error
    while 1:
        cost += A * r3  # 装配成本
        hege = A * (1 - a3)
        buhe = A - hege
        profit += hege * price  # 获利
        cost += buhe * change  # 不合格品的调换价
        Cost += cost
        Profit += profit
        # 对下一轮进行预测
        lasthege = buhe * (1 - a3)
        lastbuhege = buhe - lasthege
```

```
        lastprofit = lasthege * price
        lastcost = lastbuhege * change + buhe * dismantle + buhe * r3
        if lastprofit < lastcost:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润
        Cost += buhe * dismantle
        cost, profit = 0, 0
        A = buhe    # 用于下一轮
        if buhe <= 1:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润


# print(l101(100))


def l111(A):    # 1 1 1

    global a1, a2, a3
    Cost = 0
    Profit = 0
    cost = 0
    profit = 0
    error = A * a1    # 本身存在的不合格数
    cost += A * (k1 + k2)    # 加上检测费用
    A = A - error
    Acount = A
    while 1:
        cost += A * r3    # 装配成本
        cost += A * k3    # 检验成本 k
        hege = A * (1 - a3)
        buhe = A - hege
        profit += hege * price    # 获利
        Profit += profit
        Cost += cost
        # 预测，判断是佛拆卸
        lasthege = buhe * (1 - a3)
        lastbuhe = buhe - lasthege
        lastprofit = lasthege * price
        lastcost = buhe * dismantle + buhe * k3 + buhe * r3
        if lastprofit < lastcost:
            return Profit - Cost - (r1 + r2) * Acount    # 最后的利润
        Cost += buhe * dismantle
        cost, profit = 0, 0
        A = buhe    # 用于下一轮
        if buhe <= 1:
```

```
                return Profit - Cost - (r1 + r2) * Acount    # 最后的利润


result = []
for i in range(6):
        para = parameters[i]
        a1 = para[0]
        r1 = para[1]
        k1 = para[2]
        a2 = para[3]
        r2 = para[4]
        k2 = para[5]
        a3 = para[6]
        r3 = para[7]
        k3 = para[8]
        price = para[9]
        change = para[10]
        dismantle = para[11]
        score = []
        score.append(['000', l000(N)])

        para = parameters[i]
        a1 = para[0]
        r1 = para[1]
        k1 = para[2]
        a2 = para[3]
        r2 = para[4]
        k2 = para[5]
        a3 = para[6]
        r3 = para[7]
        k3 = para[8]
        price = para[9]
        change = para[10]
        dismantle = para[11]
        score.append(['010', l010(N)])

        para = parameters[i]
        a1 = para[0]
        r1 = para[1]
        k1 = para[2]
        a2 = para[3]
        r2 = para[4]
        k2 = para[5]
        a3 = para[6]
```

```
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score.append(['100', l100(N)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
r2 = para[4]
k2 = para[5]
a3 = para[6]
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score.append(['110', l110(N)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
r2 = para[4]
k2 = para[5]
a3 = para[6]
r3 = para[7]
k3 = para[8]
price = para[9]
change = para[10]
dismantle = para[11]
score.append(['001', l001(N)])

para = parameters[i]
a1 = para[0]
r1 = para[1]
k1 = para[2]
a2 = para[3]
r2 = para[4]
k2 = para[5]
```

```
        a3 = para[6]
        r3 = para[7]
        k3 = para[8]
        price = para[9]
        change = para[10]
        dismantle = para[11]
        score.append(['011', l011(N)])

        para = parameters[i]
        a1 = para[0]
        r1 = para[1]
        k1 = para[2]
        a2 = para[3]
        r2 = para[4]
        k2 = para[5]
        a3 = para[6]
        r3 = para[7]
        k3 = para[8]
        price = para[9]
        change = para[10]
        dismantle = para[11]
        score.append(['101', l101(N)])

        para = parameters[i]
        a1 = para[0]
        r1 = para[1]
        k1 = para[2]
        a2 = para[3]
        r2 = para[4]
        k2 = para[5]
        a3 = para[6]
        r3 = para[7]
        k3 = para[8]
        price = para[9]
        change = para[10]
        dismantle = para[11]
        score.append(['111', l111(N)])
        sorted_data = sorted(score, key=lambda x: x[1], reverse=True)
        # print()
        # print(sorted_data)
        result.append(sorted_data[0])
print(result)
```