

基于多阶段模拟仿真的生产决策问题

摘 要

企业生产过程中需要考虑次品抽样、生产流程质量控制中的决策等问题。这些是关乎企业效益的极具经济价值的关键难题，而其本质上是一个多阶段策略优化问题。本文综合假设检验、收益期望、贝叶斯理论等数理统计方法，建立多阶段优化模型，运用模拟仿真、线性模拟退火算法完成了模型求解，得到了不同情况下的最优决策。

对于问题一（以及之后的所有问题），在计算机上进行模拟仿真。根据实际现象规律，生成一组正态分布数据，并根据标准正态分布、给定的次品率得到次品的数据范围，再比较不同的抽样结果得出优劣。结果表明实际应用过程中经常使用的 (n, c) 抽样法较优（判断 n 件产品中次品数是否超过 c ，若超过则不接受，若不超过则接受），该法同时具有国家标准，具有较好的可信度。本文采用两种计算方法得出了一系列 (n, c) 值，列举一部分值如下：(1)[78, 12],[85, 13];(2)[68, 10],[65, 3],[78, 4]。

对于问题二，分析生产全流程，利用迭代思想将流程决策问题转化为多阶段问题，降低仿真难度。利用全概率公式考虑零件组合成为正品的概率，将拆解决策转换为不同阶段参数。把其余所有决策转化为概率事件，从而将决策对应的价值期望转化为净利润的比较，建立以最大盈利值为目标的单目标决策函数优化模型。考虑成品中次品检测与退换的等价性，改进决策函数，确定 6 个决策变量：购买获得的零配件 1、2 的检测率，成品检测率、拆解率，拆解获得的零配件 1、2 的检测率。并且通过线性模拟退火寻找最优的决策变量，得到不同情形下最优决策，发现所有决策结果均为 0、1 事件。且当调换费用远大于成品检查费用时，期望进行成品检查，当次品率较大时，期望检查购买所得零件。列出情形一的答案如下：只检验零配件 1，不检验零配件 2，不检验成品，对所有不合格成品进行拆解，不检验拆解后得到的零配件 1，检验拆解后得到的零配件 2。

对于问题三，生产流程添加半成品环节，通过对半成品进行分类，修改问题二的多阶段问题，从而解决对半成品的处理问题。通过新增不同阶段参数，按照问题二所述方法，将所有新增决策转化为净利润的比较，对应优化单目标决策函数优化模型。考虑模型实际意义，改进决策函数，确定 16 个决策变量，将其按顺序排成新的决策矩阵。沿用问题二结论，所有决策为 0、1 事件，比较所有可能决策方案，得到两个最优的决策矩阵（1 代表检验或拆解，0 代表不检验或不拆解）和净盈利值（仿真初始配件数为各 10000 件）：[1111111100011111],448518; [1111111111100001],435188。

对于问题四，对于扰动问题，本文使用贝叶斯统计公式，结合先验信息（可能的次品率分布）和抽样数据（似然概率分布，在本题中为二项分布）来计算后验概率分布，即零件真实的次品率分布。得到最大可能的置信区间，对置信区间内的最大和最小次品率重新代入二三问的模型，给出新决策，从而得出引入干扰后的决策范围。结果表明在 0.95 的置信区间内，二三题的决策不变，利润值产生微小改变，如第三题的两种方案利润值分别变为：448492、436620（0.95 置信区间左侧值）；425154、421235（0.95 置信区间的右侧值）。

关键词：线性模拟退火 多阶段决策 模拟仿真 贝叶斯统计

1 问题重述

1.1 问题背景

企业生产产品需要考虑许多重要因素，首先需要确保生产的零件符合质量标准，这涉及到生产过程的监控、检验和测试，以确保产品的质量可靠。在保持质量的同时，努力降低生产成本，提高生产效率。此外还需要建立健全供应链体系，确保原材料、零部件和其他必要物资的供应及时并具有稳定性。且在生产过程中要考虑生产对环境的影响，遵守相关的环境法规，采取可持续的生产方式，减少对环境的负面影响。某企业需要分别购买两种或多种零配件以装配成成品，对于不合格成品，可以选择拆解或报废。请建立数学模型优化其生产过程中的决策问题。

1.2 问题提出

1.2.1 问题一

公司零部件供应商所提供的零部件不合格率不得高于某一定值，为了检查零部件的合格率，企业采取了一种抽样测试的方式来确定是否从供应商处购买此类零部件。因为测试会产生测试成本，故应以最小测试次数的方式进行测试。特别是在下面的两个例子中，假设额定值是 10%，提供一个方案：

- (1) 在 95% 的信度下认定零配件次品率超过标称值，则拒收这批零配件；
- (2) 在 90% 的信度下认定零配件次品率不超过标称值，则接收这批零配件。

1.2.2 问题二

在这两种零配件和成品的次品率已知的情形下，对企业生产过程的四个阶段作出决策：

- (1) 对零配件（零配件 1 和/或零配件 2）是否进行检测，如果对某种零配件不检测，这种零配件将直接进入到装配环节；否则将检测出的不合格零配件丢弃；（即总共四种情况）
- (2) 对装配好的每一件成品是否进行检测，如果不检测，装配后的成品直接进入到市场；否则只有检测合格的成品进入到市场；（对每件成品的决策）
- (3) 对检测出的不合格成品是否进行拆解，如果不拆解，直接将不合格成品丢弃；否则对拆解后的零配件，重复步骤（1）和步骤（2）；（对每件不合格成品的决策）
- (4) 对用户购买的不合格品，企业将无条件予以调换，并产生一定的调换损失（包括了物流成本、企业信誉等）。对退回的不合格品，重复步骤（3）。（该决策包含在步骤三中，即对每件退回的不合格品是否拆解或丢弃）

其中零配件具有次品率、购买单价、检测成本；成品具有次品率、装配成本、检测成本、市场售价；不合格成品具有调换损失和拆解费用。题目给出六种情况，请对这些情形给出具体方案。

1.2.3 问题三

对 m 道工序、 n 个零配件，已知零配件、半成品和成品的次品率，仿照问题二，给出决策方案。并在图三给出的一种具体情况下（ $m=2, n=8$ ）给出具体决策方案、决策依据、相应

指标。

1.2.4 问题四

若第 2、3 问各工序中产品的次品率均由抽样检测方法得到，请重新完成问题 2、3。

2 问题分析

2.1 问题一的分析

抽样整体差异性较小，为等概率抽样。对于等概率抽样，简单随机抽样具有较好的代表性，故直接的思路是将简单随机抽样应用到合理选择的抽样样本上。对于大规模抽样可以采用整群抽样，即以箱为单位划分产品，对部分箱子运用简单随机抽样或全部进行检验（取决于箱子大小）。也可以采用多重抽样以降低样本数。

由于题目并未给出实际的产品样本，故需要在程序上进行模拟仿真。对于以万为单位的产品数（假设），可以考虑生成一组正态分布数据，并根据标准正态分布、给定的次品率得到次品的数据范围，再比较不同抽样的仿真结果得出优劣。

简单随机抽样具有较明显的局限性，对于理想的抽样，抽样样本数应当控制在数百以内，而且不随整体产品总量的增多而增大。当然简单随机抽样的结果可以作为检测仿真数据准确性的有力判据。

由此可以考虑采用实际应用过程中经常使用的 (n, c) 抽样法，该法同时具有国家标准，具有较好的可信度。在理想情况下，对具有一定次品率的整批产品，选取抽样样本为 n ，则数学上相当于 n 重伯努利分布。对于某次品率 p 、样本判据 c （即 n 件产品中次品数 v 是否小于等于 c ）、标称值 p_0 （在本题中等于 0.1）、人为给定的次品率 p_1 ，以及最关键的判断接受与否的置信度 α 、 β ，可以得到包括以上变量的不等式（即约束条件），由此可以求出合适的 (n, c) 数对作为较为可行的抽样方案，其最明显的优点便是：有较为严谨的推导公式、有相应的国家标准可以直接查阅合适的 n 、 c 值。除去直接查阅国家标准，本题考虑采用两种不同方法进行直接计算，可以更有效率地得到期望值。

2.2 问题二的分析

分析决策图、将决策转换为决策参数后，可以将企业生产流程从零件的检查与装配开始分析到拆解环节。本题中不存在多目标规划，或者说存在一个最重要的指标：直接盈利值。除此之外，分析题干知，还有环保性指标（即丢弃产品的价值）、调换费用（包括信誉等较难量化的指标）。

但是决策设计拆解环节必定要求对产品进行二次利用，于是有了对循环生产的要求。故如果有拆解得到的配件，便在程序上设计循环直至直接盈利值饱和，由此可以得到某种决策的重要性不同的三个衡量指标。

则题目目标转化为优化决策参数矩阵使衡量指标达到最优，最后人为筛选最优的几个解并翻译成文字决策，即决策方案。

2.3 问题三的分析

显然问题三是对问题二的推广，对于每种零配件、每种半成品、成品都存在一个决策参数，即是否检验，对于三种半成品、成品存在是否拆解的决策参数。第一次循环拆解成品后得到的半成品、组装成成品前剩下的半成品（已检验或未检验）、组装成半成品前剩下的零件（已检验或未检验）则进入第二个循环，得到第二个盈利值，最后得到次品并重新进入循环。设立决策变量矩阵后，再仿照问题二的思路进行模拟退火，得到最优的决策矩阵。

2.4 问题四的分析

直接的思路便是利用贝叶斯公式从先验概率、似然概率求出后验概率。先验概率一般选择为 Beta 分布，且其与二项分布（即似然概率函数）有一定相似性。在仿真中设定抽样整体值 n ，随机抽样次品率 θ ，若出现 c 个次品，则后验概率密度函数即为

$$P(\theta|c) = \frac{L(\theta|c)p(\theta)}{\int L(\theta|c)p(\theta)d\theta}$$

根据后验概率密度函数，可以得到最大可能的概率区间，对区间内的次品率重新代入问题二、三模型，从而给出新的决策。

若某批零件有一定的次品率，即不能 100% 确定它是正品还是次品。如果由某零件生产的产品是次品，那么也无法确定该产品拆解得到的零件是否为次品。进一步讲即使检测了一部分，但生产时又将检测出来的正品零件和未知质量的零件混合，即使拆解也无法确定哪些是正品，哪些是次品了。故不能省去关键的检测步骤。（与问题三的第二个小循环的情形存在差异）

3 模型假设

- (1) 假设零件的次品和正品随机分布，以避免次品的集中扎堆情形。
- (2) 假设半成品和产品的制作过程中，由于工艺出现次品的概率是随机的。
- (3) 检测产品半成品和零件时，若部分检测，则随机地选择部分零件检测。
- (4) 假设若不能回收零件、半成品和产品的次品，则直接丢弃不会造成环境污染等回收费用。
- (5) 假设生产环境中的多个环节都会影响半成品、成品的次品率，但均抽象成一个次品率。
- (6) 假设生产规模较大，本文仿真起始均使用 10000 套零件。

4 符号说明与名词解释

符号	含义
$L(p, n, c)$	次品率为 p 的整批产品, 选取抽样数为 n , 则其中次品件数 $v \leq c$ 的概率
M_i	零配件 i 的购买单价
T_i	零配件 i 的检测成本
P_i	零配件 i 的次品率
n_i	零配件 i 的购买个数
M_0	成品的售价
T_0	成品的检测成本
P_0	成品的次品率
C_0	成品的调换损失
D_0	成品的拆解费用
H_0	成品的装配成本
x_i	检验参数或拆解参数
m_i	衡量指标

5 模型建立与求解

5.1 问题一的求解

企业的产品检测一般不存在非等概率现象, 对于分类较少的大样本等概率检测, 采用三种可行思路: 简单随机抽样、整群抽样、二重抽样。由于题目考虑理想情况且未给出数据, 故自行生成产品数据进行模拟检验。

5.1.1 简单随机抽样

定义及操作: 从 N 个单元的总体中抽取 n 个单元组成样本, 如果抽样是不放回的, 则所有可能的样本有 C_N^n 个, 每个样本被抽中的概率为 $\frac{1}{C_N^n}$, 该方法精度中等, 但对总体的代表性并不差。

5.1.2 整群抽样

对于大样本, 直接采用简单随机抽样成本过大且在实际情况中操作较为麻烦。考虑到企业生产的产品多以箱的形式构成群体, 故可以采用整群抽样。

定义及操作: 对于某产品总体, 先将其分成很多不重合的子总体或群, 然后以群为抽样单元, 随机抽取若干个群, 对群内所有单元全部进行调查。实际意义为随机抽取若干个产品箱, 对这些箱内的产品全部进行检验。

5.1.3 二重抽样

同整群抽样的目标, 即降低抽样样本规模。

定义及操作：从总体中抽取一个样本量较大的样本，即第一重样本，在对第一重样本进行简单调查或分析，并从第一重样本中抽样出第二重样本。最后对第二重样本采用其他随机抽样方法。

5.1.4 模拟仿真与求解

采用生成正态分布数据模拟供应商生产的零配件质量，设正态分布数据的均值为 0、标准差为 1，并初步规定样本总数、次品率。

首先假设样本总数为 100000，次品率为 0.11。如果某抽样方法理想，得出的结果应为：在 95% 的可信度下认定零件次品率超过 10%；没有足够证据认为在 90% 的信度下零配件次品率不超过 10%。对于正态分布数据，采用 z 检验和零检验假设，并设定 α 值作为判据，且在程序上可以设定随机种子的值来模拟多次的随机抽样。以下给出某次抽样的结果。

对于简单随机抽样，设抽样比例为 0.1，即抽取 10000 个配件，其中次品个数为 1085；对于整群抽样，设定每个群的大小为 10，整体抽样比例仍为 10000，其中次品个数为 1086；对于二重抽样，两次抽样的比例分别为 0.5 和 0.2，总抽样比例为 0.1，抽样方法为简单随机抽样，其中次品个数为 1088。这三种抽样方法在抽样比例均为 0.1 的情况下，得出的结果均为：有 95% 的把握认为次品率大于 10%，没有足够证据认为在 90% 的信度下零配件次品率不超过 10%。

同理，假设次品率为 0.09，如果某抽样方法理想，得出的结果应为：没有足够证据认为在 95% 的可信度下零件次品率超过 10%；在 90% 的信度下认定零配件次品率不超过 10%。对于三种抽样方法，抽样比仍为 0.1，所得次品数分别为 872、899、908，z 检验下所得结果为：有 90% 的把握认为次品率低于 10%，没有足够证据认为在 95% 的可信度下零件次品率超过 10%。

从以上数据来看，在抽样比例为 0.1 的情况下，三种抽样方法均有较大的可信度，次品概率也相当准确。

5.1.5 (p,c) 抽样法

在实际生活中，根据产品的生产和使用需要，一般定出两个正数 p_0, p_1 作为检测指标，其中 p_1 略大于 p_0 。在本题中 $p_0 = 0.1$ ，即标称值。显然，若产品的次品率 $p \leq p_0$ ，则接受这批产品出厂；若 $p \geq p_1$ ，则拒绝这批产品出厂。将以上两个事件分别记作 H_0 和 H_1 。

但在判断为 H_0 事件还是 H_1 事件时，存在判断错误的可能，即置信度。故除了给定 n 与 c ，还需要给定两个数 α, β ，其含义分别为：对于事件 H_0 ，将其判断为 H_1 的概率不大于 α ；对于事件 H_1 ，将其判断为 H_0 的概率不大于 β 。在此题 (1) 中， α 的值未知， β 的值已确定为 0.05；在此题 (2) 中， α 的值已确定为 0.1， β 的值未知。

接下来进行模拟抽样。显然某个配件不是正品就是次品，故符合二项分布。对次品率为 p 的整批产品，选取抽样数为 n ，则其中次品件数 $v \leq c$ 的概率为

$$L(p, n, c) = \sum_{k=0}^c \binom{n}{k} p^k (1-p)^{n-k}$$

在给定 n 、 c 的情况下，将 $L(p, n, c)$ 简记为 $L(p)$ 。由此可以将问题一转化为以下数学问题：

从整批产品中随机抽取 n 件产品进行检验（即检验次数为 n ），其中次品的件数记为 v 。若 $v \leq c$ 时，接受 H_0 事件；若 $v \geq c$ 时，拒绝 H_0 事件，即接受 H_1 事件。若这批产品的次品率 $p \geq p_1$ ，则理想情况下应当出现 H_1 事件，而 $L(p)$ 的值为判断成 H_0 事件的概率，亦即判断错误的概率。由题知，该概率应当小于等于 β ，即

$$L(p_1) \leq L(p) \leq \beta$$

同理，若这批产品的次品率 $p \leq p_0$ ，则理想情况下应当出现 H_0 事件，而 $L(p)$ 的值为判断成 H_0 事件的概率，则错判为 H_1 事件的概率为 $1 - L(p)$ 。由题知，该概率应当小于等于 α ，即

$$1 - L(p_0) \leq 1 - L(p) \leq \alpha$$

即对给定的 p_0 、 p_1 、 α 、 β ，给出检验次数 n 、检验判据 c ，使之满足以上两个约束条件。因为 L 的表达式较复杂，精确求解以上两式较为困难，当 n 较大时，可以利用

$$T = \frac{v - np}{\sqrt{np(1-p)}}$$

渐近于 $N(0, 1)$ 标准正态分布作近似处理。因为在实际应用中，当样本容量较大时（通常大于 30）， t 分布的 t 值会逼近于标准正态分布的 z 值，因此可以使用正态分布表中的值来近似 t 分布的计算。由于 n 较大的情况下，二项分布 $B(n, p)$ 也近似于正态分布，则所给约束条件可进行如下转化

$$L(p_1) \leq \beta \Leftrightarrow T(n, c, p_1) \leq Z(1 - \beta)$$

$$L(p_0) \geq 1 - \alpha \Leftrightarrow T(n, c, p_0) \geq Z(\alpha)$$

解得

$$np_1 + Z(1 - \beta)\sqrt{np_1(1 - p_1)} \geq c \geq np_0 + Z(\alpha)\sqrt{np_0(1 - p_0)}$$

$$\sqrt{n} \geq \frac{Z(\alpha)\sqrt{p_0(1 - p_0)} - Z(1 - \beta)\sqrt{p_1(1 - p_1)}}{p_1 - p_0}$$

5.1.6 问题 1 (1) 的求解

未知数为 p_1 和 α 。

方法一：

通过遍历 p_1 和 α 得到 n 的最小值。

因为

$$\sqrt{n} \geq \frac{Z(\alpha)\sqrt{p_0(1 - p_0)} - Z(1 - \beta)\sqrt{p_1(1 - p_1)}}{p_1 - p_0}$$

故

$$n_{min} = \frac{[Z(\alpha)\sqrt{p_0(1 - p_0)} - Z(1 - \beta)\sqrt{p_1(1 - p_1)}]^2}{(p_1 - p_0)^2}$$

为便于分析，以 p_1 和 α 为横、纵坐标， n 为纵坐标，作出二维函数图。

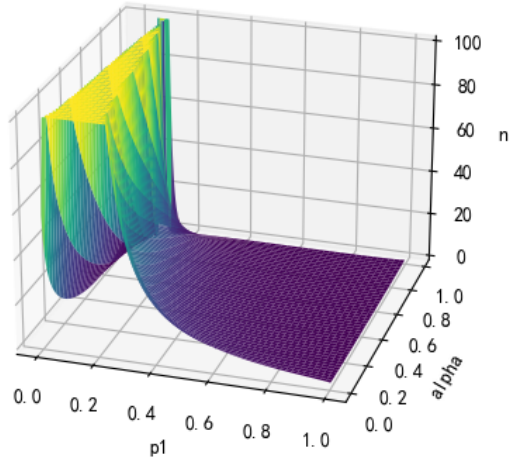


图 1

理想情况下， p_1 应当控制在 0.13-0.15 左右，而 n 应当小于 100。求得 (n, c) 数对如下：

$$[65, 4], [42, 2], [28, 1], [91, 7]$$

方法二：

若 $p = 0.1 \leq p_1$ ，则错判为拒收的概率应当小于 5%，即

$$1 - L(p, n, c) \leq \beta$$

亦即

$$L(p, n, c) = P(X \leq c) = \sum_{k=0}^c \binom{n}{k} p^k (1-p)^{n-k} \geq 1 - \beta = 0.95$$

那么对于某个 n ， c 应当有一个极限值 c_0 ，使上式取等，由于 n 重伯努利分布对应的横坐标为整数，故 c_0 也为整数，合格的 c_0 所对应的抽样概率 $L(p, n, c_0)$ 应当尽可能接近 0.95，人为规定误差为 1%，那么可以列出

$$L(p, n, c) \geq 0.95 \dots (c \geq c_0)$$

$$\frac{L(p, n, c_0) - 0.95}{0.95} \leq 0.01$$

由此遍历 n 值（一般小于 100 较为合适），若存在 c_0 满足以上不等式，则对应的 (n, c_0) 数对即为所得抽样方案。所有可行的 (n, c_0) 数对如下：

$$[14, 3], [20, 4], [27, 5], [33, 6], [34, 6], [40, 7], [41, 7], [47, 8], [48, 8], [55, 9], [56, 9], [62, 10], [63, 10], [70, 11], [71, 11], [77, 12], [78, 12], [79, 12], [85, 13], [86, 13], [93, 14], [94, 14]$$

5.1.7 问题 1 (2) 的求解

未知数为 p_1 和 β 。

方法一：

通过遍历 p_1 和 β 得到 n 的最小值。同理

$$n_{min} = \frac{[Z(\alpha)\sqrt{p_0(1-p_0)} - Z(1-\beta)\sqrt{p_1(1-p_1)}]^2}{(p_1-p_0)^2}$$

同样，以 p_1 和 β 为横、纵坐标， n 为纵坐标，作出二维函数图。

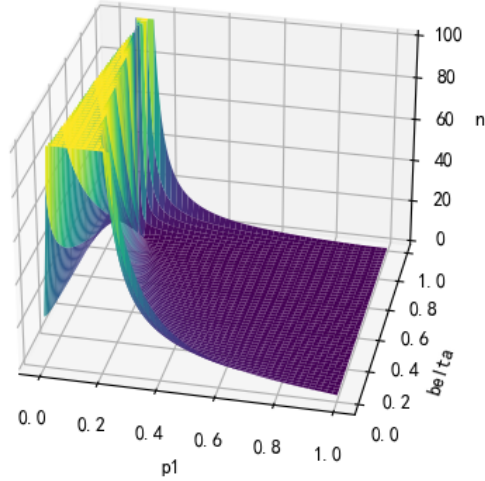


图 2

同理，理想情况下， p_1 应当控制在 0.15 左右，而 n 应当小于 100。求得 (n, c) 数对如下：

$$[68, 10], [52, 8], [55, 13]$$

方法二：

同问题一 (1)，若 $p = 0.1 \geq p_0$ ，则错判为接受的概率应当小于 10%，即

$$L(p, n, c) \leq \alpha$$

亦即

$$L(p, n, c) = P(X \leq c) = \sum_{k=0}^c \binom{n}{k} p^k (1-p)^{n-k} \leq \alpha = 0.1$$

同样的，对于某个 n 值，应当存在一个 c_0 值，满足下述不等式

$$L(p, n, c) \leq 0.1 \dots (c \leq c_0)$$

$$\frac{0.1 - L(p, n, c_0)}{0.1} < 0.01$$

遍历 n ，求得 (n, c) 数对如下：

$$[65, 3], [78, 4]$$

5.1.8 小结

结果表明实际应用过程中经常使用的 (n, c) 抽样法较简单随机抽样优 ((n, c) 数对的意思为判断 n 件产品中次品数是否超过 c ，若超过则不接受，若不超过则接受)。

5.2 问题二的求解

5.2.1 未知数假设

对于决策 1，有四种情况，即两种零配件都检验、都不检验、只检验零配件 1、只检验零配件 2。考虑到从决策 2 开始，都只考虑成品，决策过程相似，故将决策 1 的四种情况合并，并对每项决策都设计检验参数来模拟是否检验产品。

对于零配件 1，其检验参数设为 x_1 ，其值为 1 则对零配件 1 检验，其值为 0 则不对零配件 1 检验；

对于零配件 2，其检验参数设为 x_2 ，意义同理；

成品的检验参数设为 x_3 ，由于决策 2 考虑的是每件成品，故该参数应为介于 0 到 1 中的某个值，其实际意义为对占比为 x_3 的成品进行检验。

5.2.2 模型建立

Step1.

首先，对于两种零配件，企业花费的购买价格为

$$M_b = n_1 M_1 + n_2 M_2$$

两种零配件的检验个数为

$$n'_1 = n_1 x_1, n'_2 = n_2 x_2$$

两种零配件的检验成本为

$$M_{t_1} = n_1 x_1 T_1 + n_2 x_2 T_2$$

最终进入成品装配的两种零配件的个数为

$$n''_1 = n_1(1 - x_1) + n_1 x_1(1 - P_1)$$

$$n''_2 = n_2(1 - x_2) + n_2 x_2(1 - P_2)$$

两种零配件丢弃的个数为

$$n_1 = n_1 x_1 P_1, n_2 = n_2 x_2 P_2$$

零件更新流程图如下。

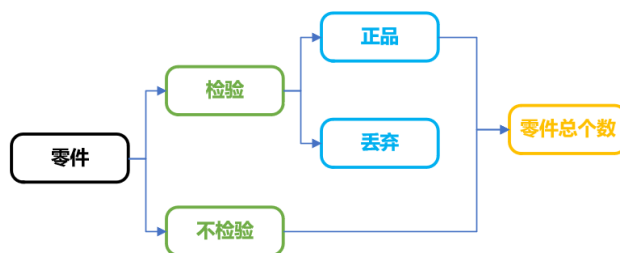


图 3

其中两种零配件的合格数为

$$n_1^* = n_1(1 - P_1), n_2^* = n_2(1 - P_2)$$

则两种零配件的新次品率为

$$P_1^* = \frac{P_1(1 - x_1)}{1 - x_1 P_1}, P_2^* = \frac{P_2(1 - x_2)}{1 - x_2 P_2}$$

成品个数为

$$n = \min(n_1'', n_2'')$$

因为现在零配件 1 的次品率为 P_1^* , 零配件 2 的次品率为 P_2^* , 两个正品零配件装配成功的概率为 $1 - P_3$, 则这些成品的正品率为

$$(1 - P_1^*)(1 - P_2^*)(1 - P_3)$$

这些成品的次品率为

$$P_3^* = 1 - (1 - P_1^*)(1 - P_2^*)(1 - P_3)$$

Step2.

检测成品的个数为

$$n' = nx_3$$

成品的检测成本为

$$M_{t_2} = nx_3 T_0$$

则可以直接售卖并盈利的成品个数为

$$n_p = nx_3(1 - P_3^*) + n(1 - x_3)(1 - P_3^*)$$

盈利的收入为

$$M_p = [nx_3(1 - P_3^*) + n(1 - x_3)(1 - P_3^*)]M_0$$

检测过的成品中, 不合格数为

$$n'' = nx_3 P_3^*$$

Step3.

设其中需要拆解的比例为 x_4 , 则拆解的数目为

$$n_{dm_1} = nx_3 P_3^* x_4$$

拆解花费为

$$M_{dm_1} = nx_3 P_3^* x_4 D_0$$

直接丢弃的数目为

$$n_{dc_1} = nx_3 P_3^* (1 - x_4)$$

Step4.

未检测过的成品个数为

$$n_t = n(1 - x_3)$$

其中不合格成品数为

$$n_{t1} = n(1 - x_3)P_3^*$$

这些不合格产品产生的调换费用为

$$M_e = n(1 - x_3)P_3^*C_0$$

调换回企业后，设这些不合格产品决定拆解的比例为 x_5 ，则拆解费用为

$$M_{dm_2} = n(1 - x_3)P_3^*x_5H_0$$

直接丢弃的数目为

$$n_{dc_2} = n(1 - x_3)P_3^*(1 - x_5)$$

5.2.3 成品决策的流程图

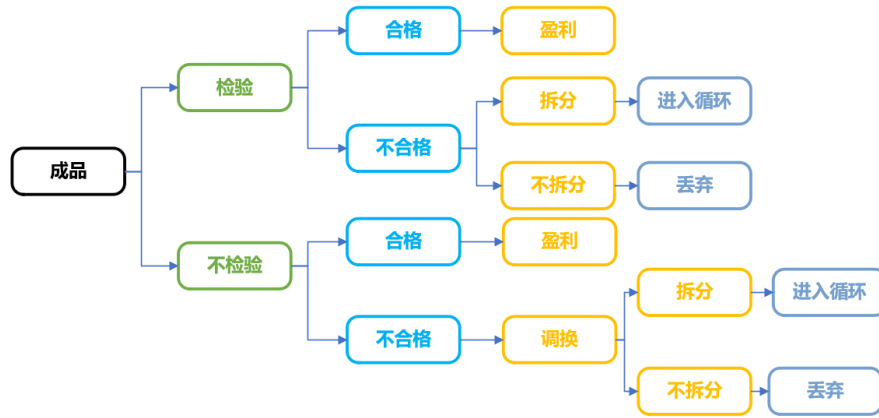


图 4

5.2.4 生产过程中检验与否的讨论

若某批零件有一定的次品率，即不能 100% 确定它是正品还是次品。如果由某零件生产的产品是次品，那么也无法确定该产品拆解得到的零件是否为次品。进一步讲即使检测了一部分，但生产时又将检测出来的正品零件和未知质量的零件混合，即使拆解也无法确定哪些是正品，哪些是次品了。故不能省去关键的检测步骤。（与问题三的第二个小循环的情形存在差异）

5.2.5 决策指标计算

在这个过程中，销售收入（即盈利收入）为

$$M_p = n(1 - P_3^*)M_0$$

检测费用为

$$M_T = n_1x_1T_1 + n_2x_2T_2 + n_3x_3T_3$$

装配成本为

$$M_a = nH_0$$

拆解成本为

$$M_{dm} = nx_3P_3^*x_4D_0 + n(1-x_3)P_3^*x_5D_0$$

考虑三个决策指标:直接盈利值(销售收入-检测成本-产品购买成本-拆解成本-装配成本)、不合格成品调换费用(包括了物流成本、企业信誉等)、环保性(丢弃产品的价值)。

其中调换费用虽然直接包含了有形的金钱损失,但也包括了无形的企业信誉,所以单独作为一个指标衡量。分别设为 m_1 、 m_2 、 m_3 。则

$$m_1 = n(1-P_3^*)M_0 - (n_1x_1T_1 + n_2x_2T_2 + n_3x_3T_3) - (n_1M_1 + n_2M_2) - [nx_3P_3^*x_4D_0 + n(1-x_3)P_3^*x_5D_0] - nH_0$$

$$m_2 = n(1-x_3)P_3^*C_0$$

$$m_3 = n_1x_1P_1M_1 + n_2x_2P_2M_2 + [nx_3P_3^*(1-x_4) + n(1-x_3)P_3^*(1-x_5)](M_1 + M_2 + H_0)$$

5.2.6 整体流程图

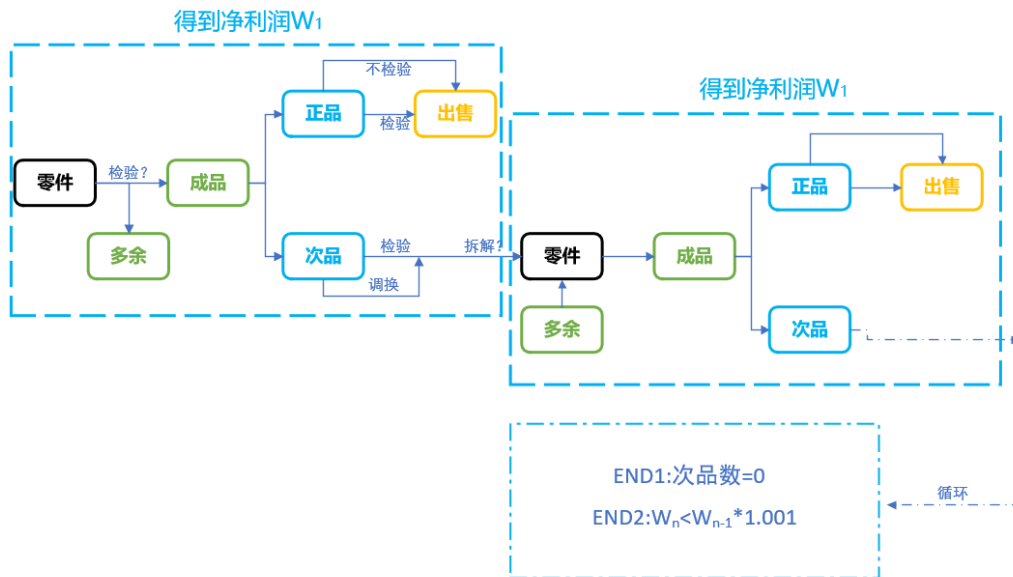


图 5

5.2.7 拆解后的处理

前述的讨论可以视作首次循环,为了最大化利用拆解后的产品,可以把拆解后的产品重新投入使用。对此又产生了新的决策分支:1. 对这些拆解后的产品全部进行检测,若检测合

格则重新装配出售；2. 不检测，直接重新投入使用，但这些拆解后的产品次品率必然较大，故通过购买更多零配件以降下次品率，即进入第二层次循环。

对于决策 1，可以对首次循环的检测情况作出优化：如果对零配件 1、零配件 2 都检测，则进入参与成品生产的零配件均为正品，拆解后不需要进行检测，直接重新装配出售，节省了二次检测成本；对于其他三种情况，无法确定拆解后的零配件是否为次品，需要进行二次检测。

以上为理论最优情形，但在仿真过程中，考虑到现实处理的便利与变量分析的统一，对所有不合格产品（包括直接检测的和用户退回的）设定一个统一的拆解比例 $x = x_4 = x_5$ ，再对拆解后得到的零件 1 和零件 2 分别设置一个额外的检测比例 x_6, x_7 ，并且从程序可行性角度出发，不严格设定 x_1, x_2 的值为 0 或 1，而是等同于其他检测比例，可以在 0 ~ 1 间变动。

5.2.8 进一步的循环与退火算法

由于拆解这一步骤影响下一个循环，所以将第 n 轮循环最后的拆解步骤并入第 $n + 1$ 轮循环的起始步骤以便分析与处理。第 $n + 1$ 轮循环起始所具有的零配件应当为第 n 轮循环起始所余下的零配件和第 n 轮循环末尾拆解所得到的零配件，在该模型中，并不接续加入新零配件，即不考虑决策 2。那么循环应当迅速收敛，因为每一轮循环后配件数会大大减少，三个指标值也会迅速减少。该猜想符合实际的仿真过程。

具体算法上，现在有三个决策指标 m_1, m_2, m_3 ，需要进行优化的变量可以写成如下数组形式

$$X = [x_1, x_2, x_3, x_4, x_5, x_6, x_7]$$

其中

x_1 = 是否对零件 1 进行检验

x_2 = 是否对零件 2 进行检验

x_3 = 是否对成品进行检验

x_4 = 是否对检验过的成品中的次品进行拆解

x_5 = 是否对退回的成品进行拆解

x_6 = 是否对拆解后得到的零件 1 进行检验

x_7 = 是否对拆解后得到的零件 2 进行检验

且一般假定 $x_4 = x_5$ 。

对于多变量决策问题，可以采用模拟退火算法来优化变量矩阵 X 。对于某个 X ，每一轮循环都会得到三个指标值，即直接盈利值、调换费用、环保性，将这三个值累加到最终决策指标 m_1, m_2, m_3 上。这三个指标中， m_1 的重要性最大，而且数值比重也最大，故将中止循环的条件设为

$$m_{n+1} \leq 1.001m_n$$

(意为盈利达到饱和，继续循环的意义不大，如若继续循环，甚至会亏本)

或

$$\text{第 } n \text{ 次循环拆解数} = 0$$

故对于某个 X ，在进行若干轮循环后，盈利值达到饱和（实际 3 ~ 8 轮即接近饱和），同时对应一个调换费用和一个环保性（这两个指标越低越好），仿真的目标即为找到一个 X ，使盈利值尽可能大，调换费用和环保性尽可能小。

5.2.9 仿真结果分析

仿真结果非常理想，而且存在以下特点：

1. 尽管初始设定 x_1 、 x_2 的值可变，但结果值总取在 0 或 1，即决策 1 的四种情况。
2. x_3 、 x_4 （或 x_5 ）的结果值同样取在 0 或 1，即要么都检测要么都不检测。
3. x_6 、 x_7 的值对最终结果影响非常小，大约在 2 ~ 3%，同时题干也并未要求对此进行讨论，可以视作过程量弱化其讨论结果，忽略不计。

5.2.10 最终结果与决策

仿真初始的两种零配件数量级设为 10000，所得最优的 6 个 X 与对应的企业净盈利值如下表。（将 x_4 、 x_5 合并后 X 的长度由 7 变为 6）

表 1

情况	1	2	3	4	5	6
X	100101	110100	001111	111100	011110	000011
净盈利值	168993.2	95995	147894	117979	186483	185867

该表的对应决策如下：

- （1）只检验零配件 1，不检验零配件 2，不检验成品，对所有不合格成品进行拆解。不检验拆解后得到的零配件 1，检验拆解后得到的零配件 2。
- （2）对零配件 1、2 都检验，不检验成品，对所有不合格成品进行拆解。不检验拆解后得到的零配件。
- （3）对零配件 1、2 都不检验，检验全部成品，对所有不合格成品进行拆解。检验所有拆解后得到的两种零配件。
- （4）对零配件 1、2 都检验，检验全部成品，对所有不合格成品进行拆解。不检验拆解后得到的零配件。
- （5）不检验零配件 1，只检验零配件 2，检验全部成品，对所有不合格成品进行拆解。检验拆解后得到的零配件 1，不检验拆解后得到的零配件 2。
- （6）对零配件 1、2 都不检验，不检验成品，不拆解所有不合格成品。

5.2.11 小结

从结果中可见，当调换费用远大于成品检查费用时，期望进行成品检查，当次品率较大时，期望检查购买所得零件。

5.2.12 算法合理性解释

模拟退火的某次优化过程如下图。

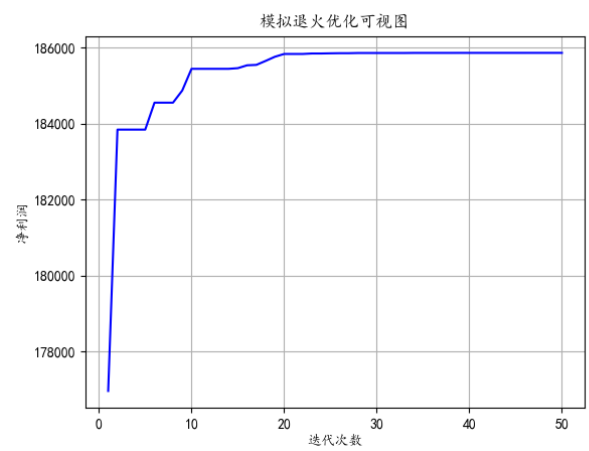


图 6

从图中可见，只有当 x_i 取到极值（0 或 1）时，才会达到最优的收敛值。

5.3 问题三的求解

5.3.1 决策模型的建立

分析过程同问题二，区别在于决策变量 x 增多，令决策矩阵

$$X = [x_1, \dots, x_{16}]$$

这些决策变量分别代表是否对 8 个零配件进行检验、是否对 3 个半成品进行检验、是否对成品进行检验、是否对三个半成品进行拆解、是否对成品进行拆解。同理，优化后的决策变量很可能只取 0 或 1，但为了完成模拟退火的优化过程，允许其值在 0 ~ 1 之间浮动。
具体决策过程如下。

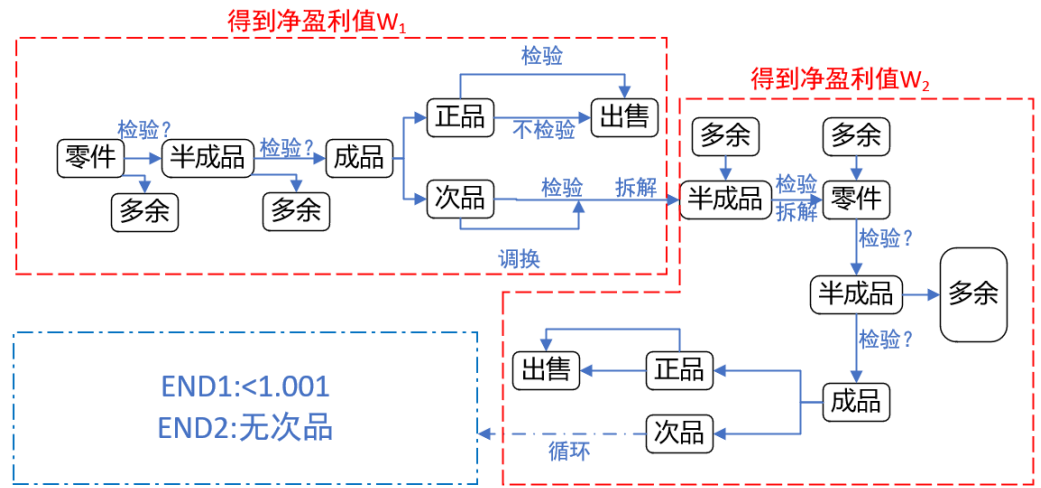


图 7

5.3.2 第一次小循环

首先在零件组成半成品时，会有多余的零件；在半成品组成成品时，会有多余的半成品；成品同样分为正品和次品（或者检验部分和不检验部分）。每一步都包含是否检验的决策参数。对于成品中的正品，可以直接出售盈利，并视作第一次循环停止，算出该循环内的直接盈利值 W_1 。

5.3.3 第二次小循环

对于成品中的次品，可以决定是否拆解，拆解得到的半成品进入第二次循环，在这些半成品中加入第一次循环得到的多余半成品，并决定是否对这些半成品进行检验和拆解，对拆解出来的零件（并加入第一次循环多余的零件）重新组成半成品，再组成成品，最后正品出售盈利，次品重新进入循环。在第二次循环中同样可以得到一个直接盈利值 W_2 ，这两个盈利值之和为一个循环得到的总直接盈利值 W_3 ，判断是否进入下一个循环的条件类似题二，即 $W_{3_{n+1}}$ 是否小于 $1.001W_{3_n}$ 或者是否产生了次品。

5.3.4 模型的优化

但该模型存在需要优化的地方，经过仔细分析后，将模型优化如下。

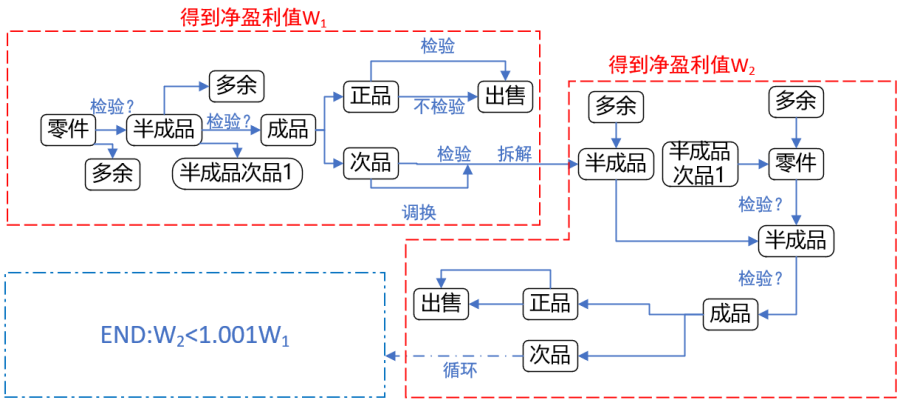


图 8

在优化前的模型中，进入第二个小循环的半成品拆解再组装，是一个重复步骤，故去掉该决策，而是只对已经检验出来的半成品次品进行拆解，在第二次组成半成品时加入这些半成品，再组装成成品进行第二次出售。

5.3.5 决策变量的优化

显然以上步骤并不止 16 个决策变量，而是 27 个（需要加上第二次循环中，是否对 8 个零件进行检验、是否对 3 个半成品进行检验）。但通过分析可以发现，若第一次循环已经对产品进行检验，在第二次循环中不必进行再次检验，反之亦然。所以实际上只需考虑 16 个决策变量。

在求解过程中发现，这些变量中，起到决定性作用的变量只有前 12 个，即是否对零配件、半成品、成品进行检验，而是否拆解的决策重要性并不强。

5.3.6 线性退火算法求解

求解过程依旧采用退火算法（设置初始零件为 10000 套），求得最优结果如下。

表 2

答案	1	2
X	1111111100011111	11111111111100001
净盈利值	448518	435188

转化为文字决策，即

(1)8 种零件都检测，3 种半成品都不检测，检测全部成品。次品（包括 3 种半成品以及成品）全部拆解。

(2)8 种零件都检测，3 种半成品都检测，不检测成品。对于次品，不拆解半成品，拆解全部成品。

5.3.7 小结

以上给出的两种最优方案有细微差别。最终盈利值较小（即第二种）的方案收敛较快，可以视作短期决策。最终盈利值较大（即第一种）的方案收敛较慢，可以视作长期决策。两种决策的利润曲线如下。

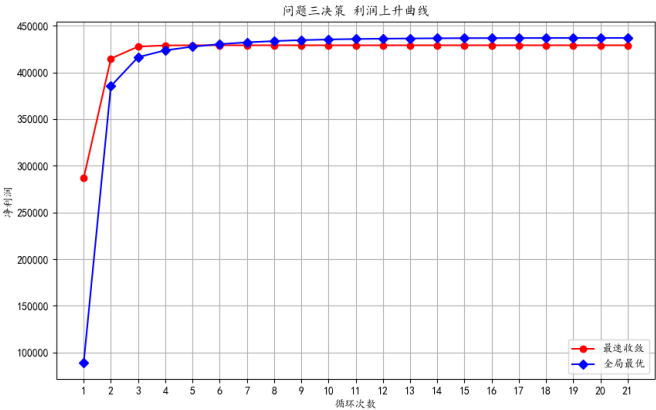


图 9

从图中可见，方案二在第三轮循环即达到收敛，方案一在第十轮循环后达到收敛。即企业若想达到最大利润，不考虑循环的时间成本，方案一较优。

5.4 问题四的求解

假设若根据简单随机抽样得到某次品率，可以使用贝叶斯统计推断来估计零件的真实次品率。该方法结合先验信息（可能的次品率分布）和抽样数据的似然函数来（二项分布）计算后验概率分布，从而得到零件真实的次品率。

5.4.1 先验分布

首先假设真实的次品率服从 Beta 分布，即先验分布。（Beta 分布具有较好的性质，适于表示在 $[0, 1]$ 区间上各种形状的概率分布。因为它具有良好的形状灵活性，能够很好地描述对参数的先验认识，在贝叶斯统计中，Beta 分布常常被用作先验分布。）Beta 函数的定义为

$$B(\alpha, \beta) = \int_0^1 u^{\alpha-1}(1-u)^{\beta-1} du$$

Beta 分布的均值和方差分别为

$$\mu = \frac{\alpha}{\alpha + \beta}$$
$$\sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}$$

归一化后得到的次品概率密度分布函数

$$p(\theta) = f(\theta; \alpha, \beta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha, \beta)}$$

5.4.2 似然函数

抽样数据服从二项分布，若次品率为 θ ，抽样 n 次，则出现 c 次次品的概率可以表示为

$$L(\theta, c) = \binom{n}{c} \theta^c (1-\theta)^{n-c}$$

5.4.3 后验概率

最后利用贝叶斯公式计算后验分布，并通过后验均值（即后验分布的最可能值）来估计真实的次品率，从而得到可能的真实次品率以及对应的置信度。积分形式下贝叶斯公式为

$$P(\theta|c) = \frac{L(\theta|c)p(\theta)}{\int L(\theta|c)p(\theta)d\theta}$$

对于连续的参数变量 θ ，其后验概率密度即为 $P(\theta|c)$ 。

为计算以上公式，需要设定 α 、 β 的初始值。其取值大小取决于对参数的信心程度。取值较小时，即对参数的先验知识不强，从而准许更多可能性；取值较大时，即对参数的先验性较强，对概率分布情况更有信心。参数的选择可以显著影响最终的后验分布，从而影响对真实参数的估计结果。

在贝叶斯推断中，通常会根据先验信息和信念程度来选择合适的先验分布参数 α 和 β 。根据似然原则，令先验函数与似然函数成比例，则

$$\alpha - 1 = c$$

$$\beta - 1 = n - c$$

带入贝叶斯公式，得到二项分布数据的后验概率密度函数

$$P(\theta|c) = \frac{\theta^{\bar{\alpha}-1}(1-\theta)^{\bar{\beta}-1}}{B(\bar{\alpha}, \bar{\beta})}$$

其中

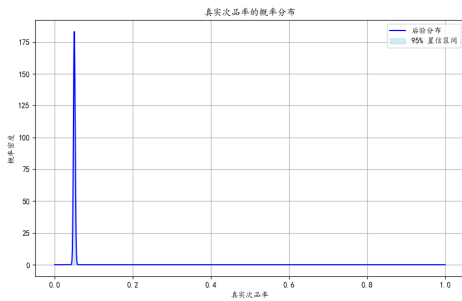
$$\bar{\alpha} = \alpha + c$$

$$\bar{\beta} = \beta + n - c$$

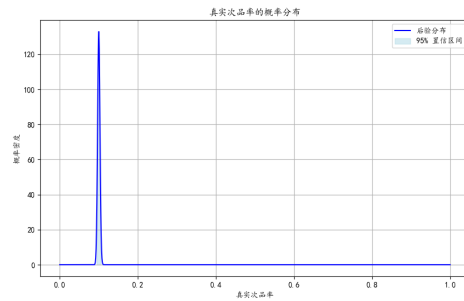
5.4.4 模拟仿真求解不同次品率的后验概率分布

例如，令 $n = 10000, c = 500$ ，则抽样得到的次品率为 5.0%，可能的真实次品率均值为 0.05008980243464378，置信度为 0.95 可能的真实次品率置信区间为 $[0.04590519, 0.05444447]$ 。同样若抽样得到的次品率为 20.0%，可能的真实次品率均值为 0.19976052684094991，置信度为 0.95 的可能的真实次品率置信区间为 $[0.19199026, 0.20764429]$ 。若抽样得到的次品率为 10.0%，可能的真实次品率均值为 0.09998004390341249，置信度为 0.95 的可能的真实次品率置信区间为 $[0.09418352, 0.10592778]$ 。

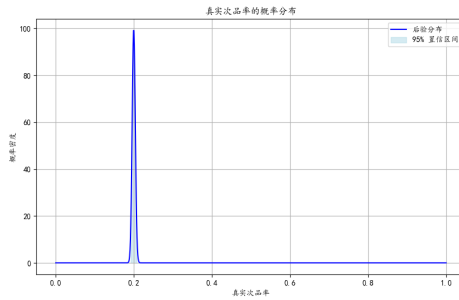
抽样后次品率为 5.0%、10% 和 20% 的后验概率分布图如下。



(a) 次品率为 0.05



(b) 次品率为 0.1



(c) 次品率为 0.2

图 10

5.4.5 重新求解问题二、三

最终的结果相当于在原有确定概率上加入了微扰。将扰动后的 0.95 概率置信区间的两个端点值作为新的次品率，重新进行问题二、问题三的操作，所得结果如下：

(1) 对于问题二，在 0.95 置信区间下，所得决策不变，即 5.2.10 的六种决策方案。

(2) 对于问题三，在 0.95 置信区间下，所得决策亦不变，即 5.3.6 的两种决策方案。决策矩阵分别为 $[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1]$ 、 $[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1]$ 。若带入置

信区间的左侧值，两种决策方案所得利润值分别为：448491.9298575758、436619.5888。若带入置信区间的右侧值，两种决策方案所得利润值分别为：425153.86094869545、421234.6082。

(3) 对于问题三，画出两种不同扰动的利润上升曲线如下，可见特征依旧符合原来问题三的结论。

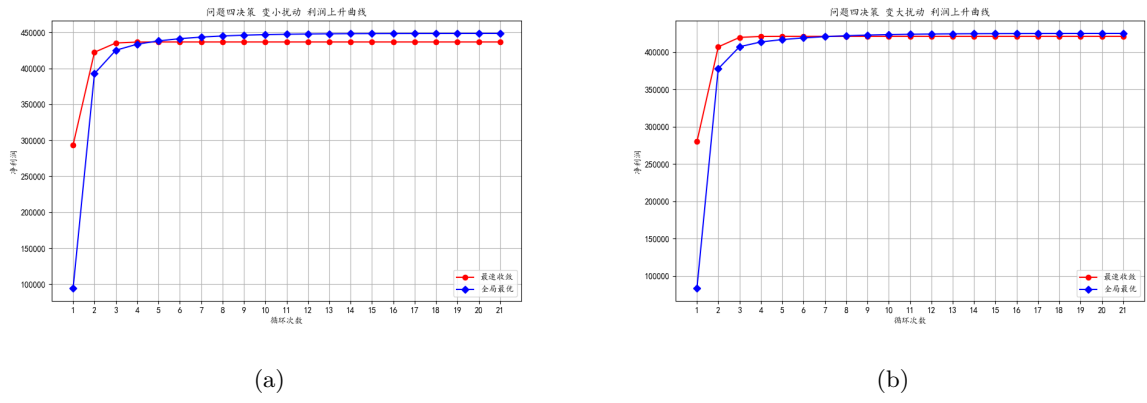


图 11

5.4.6 小结

在 0.95 置信区间下，决策并未发生显著改变，说明本题的随机抽样不会对决策结果、净利润有较大影响。

6 模型改进

(1) 问题三决策变量较多，传统的模拟退火、禁忌搜索蚁群算法、遗传算法等，效果无法达到预期。

(2) 问题三更改了模拟退火算法中一些关键函数，从而可以使用优化后的遗传算法，即类似于最速粒子群算法和飞蛾扑火算法，但即使这两个算法在特定情况下结果比较好，情况依然非常多。

(3) 预期在这两个算法的基础上，使用最速粒子群算法得到一个更好的快速迭代方法。

7 模型优缺点评价

7.1 模型优点

(1) 模型的稳定性：第二问和第三问分成两个模块，多个步骤，不需要考虑扰动的问题，本文的模型给出了期望值，并且十分准确。第四问加入扰动，但决策结果不变，证明模型至少有 95% 以上的抗扰动能力。且模型随着加工的零件和产品数目上升，越来越稳定。

(2) 模型的灵敏性：问题二使用模拟退火算法可以收敛到局部最优解。通过多次使用模拟退火可以筛选出一两个备用的决策，发现在几个情况下最优的决策比次优的决策仅仅高出 0.012 的利润。

(3) 算法：本文模型是一个线性进程，有多个步骤，每个步骤都是一个循环。于是可以无限循环迭代。而且使用了非传统的线性模拟退火。筛选出最优解以及几个备用决策，把备用的决策再验算，便得出正确的解。

7.2 模型缺点

(1) 迭代过程中有部分决策过程不合适，但是最终结果正确。

(2) 通过模型，整个仿真过程转化成不断迭代的线性进程。但该模型忽视了一种情况，一般在生产中会优先使用检测出来的正品，但模型中存在检测到的是正品，又对其拆解的情况。但最后迭代得出的结果表明该现象为算法区域机制，对于最终结果不存在任何影响。

(3) 生产规模很小的情况下，决策不一定最好，因为模型建立在 10000 套的基础上，更适用于常规规模。

8 参考文献

参考文献

- [1] 孙小素, 尚书钰. 计数标准型一次抽样检验方案设计方法探讨——兼议 GB/T 13262—2008 改进问题 [J]. 山东工商学院学报, 2024, 38(01): 69-76.

9 附录

9.1 程序结果汇总

9.2 代码汇总

```
1 from scipy.stats import norm
2 import math
3 import random
4 import matplotlib.pyplot as plt
5 import numpy as np
6
7 plt.rcParams['font.sans-serif'] = ['KaiTi']
8 plt.rcParams['axes.unicode_minus'] = False
9
10 def quantile_value(p):
11     # 计算标准正态分布的第p分位数值
12     value = norm.ppf(p)
13     return value
14
15 def get_n(p0, p1, alpha, belta):
16     swap_n = quantile_value(1 - alpha) * math.sqrt(p0 * (1 - p0)) - quantile_value(belta)
17     * math.sqrt(p1 * (1 - p1))
18     swap_n /= p1 - p0
19     return swap_n ** 2 if swap_n ** 2 < 100 else 100
```

```

20 def get_c(p0, p1, alpha, belta, n):
21     # print(p0, p1, alpha, belta, n)
22     l_value = n * p1 + quantile_value(belta) * math.sqrt(n * p1 * (1 - p1))
23     r_value = n * p0 + quantile_value(1 - alpha) * math.sqrt(n * p0 * (1 - p0))
24     return l_value, r_value
25
26 def get_ans(p1, belta):
27     n = get_n(p0, p1, alpha, belta)
28     c_l, c_r = get_c(p0, p1, alpha, belta, n)
29     # print(n, c_l, c_r)
30     return n, c_l
31
32 def get_ans_n(p1, belta):
33     n = get_n(p0, p1, alpha, belta)
34     c_l, c_r = get_c(p0, p1, alpha, belta, n)
35     # print(n, c_l, c_r)
36     return n
37
38 # todo: 问题一 , 法一
39 print("问题二 , 法一")
40 f1 = open("问题二, 法一.txt", mode="w", encoding="utf-8")
41 p0 = 0.1
42 belta = 0
43 p1 = 0
44 alpha = 0.1
45 for p1 in [0.001 * _ for _ in range(1, 1000) if 0.001 * _ != p0]:
46     for belta in [0.01 * _ for _ in range(1, 100)]:
47         n, c = get_ans(p1, belta)
48         if c < 0:
49             continue
50         f1.write(f"p1 = {p1}, belta = {belta},,此时n为: {n}, c为: {c}\n")
51         # print(f"解为: p1 = {p1}, alpha = {alpha}")
52         # print(f"此时n为: {n}, c为: {c}")
53 x = np.array([0.001 * _ for _ in range(1, 1000) if 0.001 * _ != p0])
54 y = np.array([0.01 * _ for _ in range(1, 100)])
55 x, y = np.meshgrid(x, y)
56 custom_function_vectorized = np.vectorize(get_ans_n)
57 z = custom_function_vectorized(x, y)
58 # 创建三维图形对象
59 fig = plt.figure()
60 ax = fig.add_subplot(111, projection='3d')
61
62 # 绘制三维图形
63 ax.plot_surface(x, y, z, cmap='viridis')
64
65 # 设置坐标轴标签
66 ax.set_xlabel('p1')
67 ax.set_ylabel('belta')
68 ax.set_zlabel('n')
69 ax.set_title('问题二 方法一')
70 plt.show()

```

```

71
72 from scipy.stats import norm
73 import math
74 import random
75 import matplotlib.pyplot as plt
76 import numpy as np
77
78 plt.rcParams['font.sans-serif'] = ['KaiTi']
79 plt.rcParams['axes.unicode_minus'] = False
80
81 def quantile_value(p):
82     # 计算标准正态分布的第p分位数值
83     value = norm.ppf(p)
84     return value
85
86 def get_n(p0, p1, alpha, belta):
87     swap_n = quantile_value(1 - alpha) * math.sqrt(p0 * (1 - p0)) - quantile_value(belta)
88         * math.sqrt(p1 * (1 - p1))
89     swap_n /= p1 - p0
90     return swap_n ** 2 if swap_n ** 2 < 100 else 100
91
92 def get_c(p0, p1, alpha, belta, n):
93     l_value = n * p1 + quantile_value(belta) * math.sqrt(n * p1 * (1 - p1))
94     r_value = n * p0 + quantile_value(1 - alpha) * math.sqrt(n * p0 * (1 - p0))
95     return l_value, r_value
96
97 def get_new_ans(p1, alpha, t):
98     p1 += 2 * random.random() * math.exp(t - 1000) - math.exp(t - 1000)
99     alpha += 2 * random.random() * math.exp(t - 1000) - math.exp(t - 1000)
100     return p1, alpha
101
102 def get_ans(p1, alpha):
103     n = get_n(p0, p1, alpha, belta)
104     c_l, c_r = get_c(p0, p1, alpha, belta, n)
105     # print(n, c_l, c_r)
106     return n, c_l
107
108 def get_ans_n(p1, alpha):
109     n = get_n(p0, p1, alpha, belta)
110     c_l, c_r = get_c(p0, p1, alpha, belta, n)
111     # print(n, c_l, c_r)
112     return n
113
114 def is_ans(p1, alpha):
115     if p1 < 0.13 or p1 > 0.19 or p1 <= p0:
116         return False
117     if alpha < 0 or alpha > 1:
118         return False
119     return True
120

```

```

121 def get_p(delta, t):
122     # 返回一个无限小的正数, 范围为0到1
123     # 随着退火次数增加而越来越小
124     return math.exp(-delta/t)
125
126 ## todo: 问题一, 法一
127 # print("问题一, 法一")
128 # f1 = open("问题一, 法一.txt", mode="w", encoding="utf-8")
129 # p0 = 0.1
130 # belta = 0.05
131 # p1 = 0.15
132 # alpha = 0.1
133 # for p1 in [0.001 * _ for _ in range(1, 1000) if 0.001 * _ != p0]:
134 #     for alpha in [0.01 * _ for _ in range(1, 100)]:
135 #         n, c = get_ans(p1, alpha)
136 #         if c < 0:
137 #             continue
138 #         f1.write(f"p1 = {p1}, alpha = {alpha},,此时n为: {n}, c为: {c}\n")
139 #         # print(f"解为: p1 = {p1}, alpha = {alpha}")
140 #         # print(f"此时n为: {n}, c为: {c}")
141 # x = np.array([0.001 * _ for _ in range(1, 1000) if 0.001 * _ != p0])
142 # y = np.array([0.01 * _ for _ in range(1, 100)])
143 # x, y = np.meshgrid(x, y)
144 # custom_function_vectorized = np.vectorize(get_ans_n)
145 # z = custom_function_vectorized(x, y)
146 ## 创建三维图形对象
147 # fig = plt.figure()
148 # ax = fig.add_subplot(111, projection='3d')
149 #
150 ## 绘制三维图形
151 # ax.plot_surface(x, y, z, cmap='viridis')
152 #
153 ## 设置坐标轴标签
154 # ax.set_xlabel('p1')
155 # ax.set_ylabel('alpha')
156 # ax.set_zlabel('n')
157 # ax.set_title('问题一 方法一')
158 # # ax.set_zlim(0, 10)
159 # plt.show()
160 #
161 ## t = 1000          # 初始化温度值
162 ## t_min = 980      # 设置温度下限
163 ## k = 1000         # 设置迭代次数 (有用吗)
164 ##
165 ## while (t > t_min):
166 ##     for i in range(500):
167 ##         n, c = get_ans(p1, alpha)
168 ##         p1_, alpha_ = get_new_ans(p1, alpha, t)
169 ##         if is_ans(p1_, alpha_):
170 ##             n_, c_ = get_ans(p1_, alpha_)
171 ##             delta = n_ - n

```

```

172 # # # n最小
173 # # if c_ > 0:
174 # # if delta < 0:
175 # # p1, alpha = p1_, alpha_
176 # # else:
177 # # p = get_p(delta, t)
178 # # if (p > random.random()):
179 # # p1, alpha = p1_, alpha_
180 # #
181 # # print(f"当前最优解为: p1 = {p1}, alpha = {alpha}")
182 # # print(f"此时n为: {n}, c为: {c}")
183 # # # t = t * alpha # 温度下降
184 # # t -= 0.01
185 # #
186 # # n, c = get_ans(p1, alpha)
187 # # print(f"最优解为: p1 = {p1}, alpha = {alpha}")
188 # # print(f"此时n为: {n}, c为: {c}")
189 # 问题一
190 import scipy.stats as stats
191
192 p = 0.1
193 n_values = range(10, 101)
194 valid_n_values = []
195 for n in n_values:
196     binom_dist = stats.binom(n, p)
197     c0 = binom_dist.ppf(0.95)
198     P_X_leq_c0 = binom_dist.cdf(c0)
199     diff = (P_X_leq_c0 - 0.95) / 0.95
200     if diff < 0.01:
201         valid_n_values.append([n, c0])
202 print(f"符合条件的所有n为: {valid_n_values}")
203
204 # 问题二
205 valid_n_values = []
206 for n in n_values:
207     binom_dist = stats.binom(n, p)
208     c0 = binom_dist.ppf(0.1) - 1
209     P_X_leq_c0 = binom_dist.cdf(c0)
210     # print(P_X_leq_c0)
211     diff = (0.1 - P_X_leq_c0) / 0.1
212     if diff < 0.01:
213         valid_n_values.append([n, c0])
214 print(f"符合条件的所有n为: {valid_n_values}")
215 import numpy as np
216 import pandas as pd
217 import random
218 from statsmodels.stats.proportion import proportions_ztest
219
220
221 def ztest_func(data_get):
222     """

```

```

223     传入一维列表，表示样本
224     :param data_get:
225     :return:
226     """
227     # todo: 对抽样数据进行检验 检验
228     data_get = np.array(data_get, dtype="int32")
229     # 进行z检验
230     wanna_p0 = 0.1
231     no_num = int((data_get == 0).sum())
232     all_num = len(data_get)
233     print(f"次品个数: {no_num}, 整体数目: {all_num}")
234
235     # 检验一
236     print("检验一")
237     # 'smaller'代表低于alpha时成立,
238     z_stat, p_val = proportions_ztest(no_num, all_num, wanna_p0, alternative='smaller')
239     print(f"Z statistic: {z_stat}, P-value:{p_val}")
240
241     # 判断原假设
242     alpha = 0.10
243     if p_val < alpha:
244         print("拒绝零假设: 即认为有90%的把握认为次品率低于10%")
245     else:
246         print("接受零假设: 即认为没有90%的把握认为次品率低于10%")
247
248     # 检验二
249     print("检验二")
250     # 'larger'代表高于alpha时成立,
251     z_stat, p_val = proportions_ztest(no_num, all_num, wanna_p0, alternative='larger')
252     print(f"Z statistic: {z_stat}, P-value:{p_val}")
253
254     # 判断原假设
255     alpha = 0.05
256     if p_val < alpha:
257         print("拒绝零假设: 即认为有95%的把握认为次品率大于10%")
258     else:
259         print("接受零假设: 即认为没有95%的把握认为次品率大于10%")
260
261
262 np.random.seed(12845425)
263 random.seed(10524) # 设置随机数种子, 以确保结果可重复
264
265 # todo: 生成合理的样本
266 # 样本总数确定
267 data_num = 100000
268 cipin_lv = 0.11
269
270 ## 样本生成 (应当是二项分布)
271 # data_0_num = int(data_num * cipin_lv)
272 # data_1_num = int(data_num - data_0_num)
273 # data_get = np.concatenate((np.zeros(data_0_num), np.ones(data_1_num)))

```

```

274 # np.random.shuffle(data_get)
275 # # print(f"生成数据: {data_get}")
276
277 # 样本生成 (正态分布, 较大的90%为正品, 其余为次品)
278 data_get = np.random.normal(loc=0, scale=1, size=data_num) # 均值为0, 标准差为1
279 threshold = np.percentile(data_get, cipin_lv * 100)
280 data_get[data_get >= threshold] = 1
281 data_get[data_get != 1] = 0
282 # print(f"生成数据: {data_get}")
283
284 # 样本保存
285 df = pd.DataFrame(data_get, columns=['样本 (0为次品, 1为合格)'])
286 f1 = '第一问/问题一数据.xlsx'
287 df.to_excel(f1, index=False)
288 print(f"样本已成功生成并保存到 {f1} 文件中。")
289 no_num = int((data_get == 0).sum())
290 all_num = len(data_get)
291 print(f"次品个数: {no_num}, 整体数目: {all_num}")
292
293 # todo: 尝试不同的抽样方法
294 # 简单随机抽样, 整群抽样, 二重抽样
295
296 # 载入数据并建检验数据
297 df = pd.read_excel('第一问/问题一数据.xlsx', sheet_name='Sheet1')
298 data_get = df['样本 (0为次品, 1为合格)']
299 print(f"首先进行数据检验, 此批产品中合格率为: {np.sum(data_get)/len(data_get)}")
300 print(f"此批产品中次品率为: {(1 - np.sum(data_get)/len(data_get)) * 100}%")
301
302
303 # todo: 简单随机抽样
304 # 确定抽样比例
305 data_get = df['样本 (0为次品, 1为合格)']
306 no_num = int((data_get == 0).sum())
307 all_num = len(data_get)
308 sample_num = all_num * 0.1
309 # 进行抽样
310 random_indices = random.sample(range(all_num), int(sample_num))
311 # print(random_indices)
312 sample = [data_get[i] for i in random_indices]
313 print("随机抽样的样本为: ", sample)
314 ztest_func(sample)
315
316 # todo: 整群抽样
317 # 依据排序进行分组, 10个位一组
318 data_get = df['样本 (0为次品, 1为合格)']
319 all_num = len(data_get)
320 data_get = [[data_get[__ + __ * 10] for __ in range(10)] for _ in range(int(all_num / 10))]
321 # 创建划分好的产品群体
322 all_num = len(data_get)
323 sample_num = all_num * 0.1

```

```

324 sample_indices = random.sample(range(all_num), int(sample_num))
325
326 # 抽取整个群的产品作为样本
327 sample = []
328 for indices in sample_indices:
329     sample += data_get[indices]
330
331 # 打印抽样结果
332 print("整群抽样的样本为: ", sample)
333 ztest_func(sample)
334
335 # todo: 二重抽样
336 # 第一阶段抽样
337 data_get = df['样本 (0为次品, 1为合格)']
338 all_num = len(data_get)
339 sample_num = all_num * 0.5
340 random_indices = random.sample(range(all_num), int(sample_num))
341 data_get = [data_get[i] for i in random_indices]
342 # print("一重抽样的样本为: ", data_get)
343 # 第二阶段抽样
344 data_get = np.array(data_get)
345 all_num = len(data_get)
346 sample_num = all_num * 0.2
347 random_indices = random.sample(range(all_num), int(sample_num))
348 sample = [data_get[i] for i in random_indices]
349 print("二重抽样的最终样本为: ", sample)
350 ztest_func(sample)
351
352 # todo: 尝试不同的抽样方法
353 # 简单随机抽样, 整群抽样, 二重抽样
354
355 import numpy as np
356 import pandas as pd
357 from statsmodels.stats.proportion import proportions_ztest
358 import random
359
360 # 载入数据并建检验数据
361 df = pd.read_excel('第一问/问题一数据.xlsx', sheet_name='Sheet1')
362 data_get = df['样本 (0为次品, 1为合格)']
363 print(f"首先进行数据检验, 此批产品中合格率为: {np.sum(data_get)/len(data_get)}")
364
365 random.seed(1024) # 设置随机数种子, 以确保结果可重复
366
367 # todo: 简单随机抽样
368 # 确定抽样比例
369 data_get = df['样本 (0为次品, 1为合格)']
370 no_num = int((data_get == 0).sum())
371 all_num = len(data_get)
372 sample_num = all_num * 0.1
373 # 进行抽样
374 random_indices = random.sample(range(all_num), int(sample_num))

```

```

375 # print(random_indices)
376 sample = [data_get[i] for i in random_indices]
377 print("随机抽样的样本为：", sample)
378
379
380 # todo: 整群抽样
381 # 依据排序进行分组, 10个位一组
382 data_get = df['样本 (0为次品, 1为合格)']
383 all_num = len(data_get)
384 data_get = [[data_get[__ + _ * 10] for __ in range(10)] for _ in range(int(all_num / 10))]
385
386 # 创建划分好的产品群体
387 all_num = len(data_get)
388 sample_num = all_num * 0.1
389 sample_indices = random.sample(range(all_num), int(sample_num))
390
391 # 抽取整个群的产品作为样本
392 sample = []
393 for indices in sample_indices:
394     sample += data_get[indices]
395
396 # 打印抽样结果
397 print("整群抽样的样本为：", sample)
398
399 # todo: 二重抽样
400 # 第一阶段抽样
401 data_get = df['样本 (0为次品, 1为合格)']
402 all_num = len(data_get)
403 sample_num = all_num * 0.5
404 random_indices = random.sample(range(all_num), int(sample_num))
405 data_get = [data_get[i] for i in random_indices]
406 print("一重抽样的样本为：", data_get)
407 # 第二阶段抽样
408 data_get = np.array(data_get)
409 all_num = len(data_get)
410 sample_num = all_num * 0.2
411 random_indices = random.sample(range(all_num), int(sample_num))
412 sample = [data_get[i] for i in random_indices]
413 print("二重抽样的最终样本为：", sample)
414
415
416
417
418
419
420 ##
421 # data_get = np.array(sample)
422 # # 进行z检验
423 # wanna_p0 = 0.1
424 # no_num = int((data_get == 0).sum())

```

```

425 # all_num = len(data_get)
426 # print(f"次品个数: {no_num}, 整体数目: {all_num}")
427 # ##
428 # # 检验一
429 # print("检验一")
430 # # 'smaller'代表低于alpha时成立,
431 # z_stat, p_val = proportions_ztest(no_num, all_num, wanna_p0, alternative='smaller')
432 # print(f"Z statistic: {z_stat}, P-value:{p_val}")
433 #
434 # # 判断原假设
435 # alpha = 0.05
436 # if p_val < alpha:
437 #     print("拒绝零假设: 即认为有95%的把握认为次品率低于10%")
438 # else:
439 #     print("接受零假设: 即没有足够的证据认为次品率低于10%")
440 # ##
441 # # 检验二
442 # print("检验二")
443 # # 'larger'代表高于alpha时成立,
444 # z_stat, p_val = proportions_ztest(no_num, all_num, wanna_p0, alternative='larger')
445 # print(f"Z statistic: {z_stat}, P-value:{p_val}")
446 #
447 # # 判断原假设
448 # alpha = 0.10
449 # if p_val < alpha:
450 #     print("拒绝零假设: 即认为有90%的把握认为次品率大于10%")
451 # else:
452 #     print("接受零假设: 即没有足够的证据认为次品率大于10%")
453
454 # todo: 生成合理的样本
455
456 import numpy as np
457 import pandas as pd
458
459 # 样本总数确定
460 data_num = 1000
461 data_0_num = int(data_num * 0.12)
462 data_1_num = int(data_num - data_0_num)
463 # 样本生成
464 np.random.seed(1024)
465 data_get = np.concatenate((np.zeros(data_0_num), np.ones(data_1_num)))
466 np.random.shuffle(data_get)
467 # 样本保存
468 df = pd.DataFrame(data_get, columns=['样本 (0为次品, 1为合格)'])
469 f1 = '第一问/问题一数据.xlsx'
470 df.to_excel(f1, index=False)
471 print(f"样本已成功生成并保存到 {f1} 文件中。")
472
473

```

题一代码

```

1  import numpy as np
2  import pandas as pd
3  import random
4  import math
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8  plt.rcParams['font.sans-serif'] = ['KaiTi']
9  plt.rcParams['axes.unicode_minus'] = False
10
11
12
13 def get_w(data_list):
14     # 此处进行仿真，为时间角度考虑，仅循环两次
15     ans_list = []
16     # todo: 过程函数
17     def get_new_part(n_part_ori, p_part_make, p_part_test):
18         # 零件检测函数，输入零件数、次品率、检测率，得到新零件数、新次品率、检测数
19         n_part_good = n_part_ori * (1 - p_part_make)
20         n_part_bad = n_part_ori * p_part_make
21         n_part_bad_test = n_part_bad * p_part_test
22         n_part_new = n_part_ori - n_part_bad_test
23         p_part_new = 1 - n_part_good / n_part_new
24         return n_part_new, p_part_new, n_part_ori * p_part_test
25
26     def get_product(n_parts, p_parts, p_product_make):
27         # 输入每种零件的个数，得到产品生产数、次品数、正品数
28         p_good = 1 - p_product_make
29         for enum in p_parts:
30             p_good *= 1 - enum
31         n_parts = np.array(n_parts)
32         n_product = np.min(n_parts)
33         n_product_good = n_product * p_good
34         n_product_bad = n_product - n_product_good
35         return n_product, n_product_good, n_product_bad
36
37     def deal_product_bad2(n_product_bad, p_parts):
38         # 输入次品数、每个零件的次品率，得到每种新的零件个数，每种新的零件的次品率（一个
39         # 成品由两个零件构成）
40         n_part_1 = n_product_bad
41         n_part_2 = n_product_bad
42         p_swap = p_parts[0] + p_parts[1] - p_parts[0] * p_parts[1] + (1 - p_parts[0]) * (
43             1 - p_parts[1]) * p_product_make
44         p_part_1 = p_parts[0] / p_swap
45         p_part_2 = p_parts[1] / p_swap
46         return n_part_1, n_part_2, p_part_1, p_part_2
47
48     # 循环仿真直至不能盈利
49     # 此处使用法一（全程固定）进行循环处理
50     # print("此处使用法一（全程固定）进行循环处理")

```

```

50 # todo: 决策变量
51 # 总盈利值、丢弃成本, 调换成本
52 w_1 = 0
53 w_2 = 0
54 w_3 = 0
55 # todo: 决策方案变量
56 # 检测率、拆解率
57 p_part1_test = data_list[0]
58 p_part2_test = data_list[1]
59 p_product_test = data_list[2]
60 p_product_dismantle = data_list[3]
61
62 p_part1_test0 = data_list[4]
63 p_part2_test0 = data_list[5]
64 # todo: 环境变量
65 # 次品率
66 p_part1_make = 0.05
67 p_part2_make = 0.05
68 p_product_make = 0.05
69 # 购买或者制作单价
70 m_part1_buy = 4
71 m_part2_buy = 18
72 m_product_buy = 6 # 制作单价
73 # 测试单价
74 m_part1_test = 2
75 m_part2_test = 3
76 m_product_test = 3
77 # 售价、拆解价格、调换价格
78 m_product_sale = 56
79 m_product_dismantle = 40
80 m_product_exchange = 10
81 # 零件数 (总是希望利用率最大)
82 n_part1_ori = 10000
83 n_part2_ori = 10000
84 # todo: 仿真交换变量
85 n_part1_swap = 0
86 p_part1_swap = 0
87 n_part2_swap = 0
88 p_part2_swap = 0
89 # todo: 仿真过程 (每次仿真得到的新的决策变量应当在前一次的基础上改变)
90 count = 0
91 while True:
92     # print("asd")
93     # 继承上次循环决策变量
94     w_1_new, w_2_new, w_3_new = w_1, w_2, w_3
95     count += 1
96     # print(f"当前是第{count}次循环")
97     # todo: 零件检测
98     if count == 1:
99         # 首次进行时, 需计算零件成本, 计算零件检测成本
100         w_1_new -= n_part1_ori * m_part1_buy + n_part2_ori * m_part2_buy

```

```

101         # 零件检测函数，输入零件数、次品率、检测率，得到新零件数、新次品率、检测数
102         n_part1_new, p_part1_new, n_part1_test = get_new_part(n_part1_ori,
p_part1_make, p_part1_test)
103         n_part2_new, p_part2_new, n_part2_test = get_new_part(n_part2_ori,
p_part2_make, p_part2_test)
104         w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
105         w_2_new += (n_part1_ori - n_part1_new) * m_part1_buy + (n_part2_ori -
n_part2_new) * m_part2_buy
106         # print("asd")
107         # print(n_part1_new, p_part1_new, n_part1_test)
108         # print(n_part2_new, p_part2_new, n_part2_test)
109     else:
110         # 其他情况，产品零件在最后一步进行处理，此处进行继承
111         n_part1_new, p_part1_new = n_part1_swap, p_part1_swap
112         n_part2_new, p_part2_new = n_part2_swap, p_part2_swap
113     # todo: 生成产品
114     n_product, n_product_good, n_product_bad = get_product([n_part1_new, n_part2_new
], [p_part1_new, p_part2_new],
115                                                         p_product_make)
116     n_part1_ex = n_part1_new - n_product
117     n_part2_ex = n_part2_new - n_product
118     p_part1_ex = p_part1_new
119     p_part2_ex = p_part2_new
120     # print(n_part1_ex, n_part2_ex)
121     # print(p_part1_ex, p_part2_ex)
122     # print(n_product, n_product_good, n_product_bad)
123     w_1_new -= n_product * m_product_buy
124     # todo: 计算决策变量（次品默认丢弃），区分产品正品与次品
125     w_1_new += n_product_good * m_product_sale - n_product_good * p_product_test *
m_product_test
126     w_1_new -= n_product_bad * p_product_test * m_product_test + n_product_bad * (
127         1 - p_product_test) * m_product_exchange
128     # print(n_product_bad * p_product_test)
129     # print(n_product_bad * (
130     #     1 - p_product_test))
131     w_2_new += n_product_bad * (m_part1_buy + m_part2_buy + m_product_buy)
132     w_3_new += n_product_bad * m_product_exchange
133     # w_3_new += n_product_bad * (1 - p_product_test) * m_product_exchange
134     # print(f"当前得到的净利润: {w_1_new}, 环境成本: {w_2_new}, 信用成本: {w_3_new}")
135     ans_list.append([w_1_new, w_2_new, w_3_new])
136     # print(f"上一轮得到的净利润: {w_1}, 环境成本: {w_2}, 信用成本: {w_3}")
137     # todo: 判断循环是否结束
138     if w_1_new - w_1 <= w_1 * 0.001 or p_product_dismantle == 0:
139         break
140     # todo: 处理产品次品
141     n_part1_new1, n_part2_new1, p_part1_new1, p_part2_new1 = deal_product_bad2(
n_product_bad * p_product_dismantle,
142
143     p_part1_new, p_part2_new])
144     # print(n_part1_new1, n_part2_new1, p_part1_new1, p_part2_new1)
145     # print(n_part1_new1 + n_part1_ex)

```

```

145     # print(n_part2_new1 + n_part2_ex)
146     p_part1_new1 = (n_part1_new1 * p_part1_new1 + n_part1_ex * p_part1_ex) / (
n_part1_new1 + n_part1_ex)
147     n_part1_new1 = n_part1_new1 + n_part1_ex
148     p_part2_new1 = (n_part2_new1 * p_part2_new1 + n_part2_ex * p_part2_ex) / (
n_part2_new1 + n_part2_ex)
149     n_part2_new1 = n_part2_new1 + n_part2_ex
150     # print(n_part1_new1, n_part2_new1, p_part1_new1, p_part2_new1)
151     w_1_new -= (n_product_bad * p_product_dismantle) * m_product_dismantle
152     w_2_new -= (n_product_bad * p_product_dismantle) * (m_part1_buy + m_part2_buy +
m_product_buy)
153     # todo: 对于拆解得到的零件, 此处讨论是否进行检验
154     # 零件检测函数, 输入零件数、次品率、检测率, 得到新零件数、新次品率、检测数
155     n_part1_new, p_part1_new, n_part1_test = get_new_part(n_part1_new1, p_part1_new1,
p_part1_test0)
156     n_part2_new, p_part2_new, n_part2_test = get_new_part(n_part2_new1, p_part2_new1,
p_part2_test0)
157     # print(n_part1_new, p_part1_new, n_part1_test)
158     # print(n_part2_new, p_part2_new, n_part2_test)
159
160     w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
161     w_2_new += (n_part1_new1 - n_part1_new) * m_part1_buy + (n_part2_new1 -
n_part2_new) * m_part2_buy
162     # todo: 保留迭代值
163     w_1, w_2, w_3 = w_1_new, w_2_new, w_3_new
164     n_part1_swap, p_part1_swap, n_part2_swap, p_part2_swap = n_part1_new, p_part1_new
, n_part2_new, p_part2_new
165
166     # print(f"最终得到的净利润: {w_1_new}, 环境成本: {w_2_new}, 信用成本: {w_3_new}")
167     if len(ans_list) == 1:
168         return ans_list[0]
169     if ans_list[-2][0] > ans_list[-1][0]:
170         return ans_list[-2]
171     return ans_list[-1]
172
173 def get_p(delta, t):
174     return math.exp(delta/t)
175
176 def get_new_ans_list(ans_list, t):
177     swap_list = []
178     for i in range(len(ans_list)):
179         # print(+ 2 * random.random() * math.exp(t - 1000) - math.exp(t - 1000))
180         swap_list.append(ans_list[i] + 2 * random.random() * math.exp(t - 1000) - math.
exp(t - 1000))
181     # print(swap_list)
182     # swap_list = [enum + 2 * random.random() * math.exp(t - 1000) - math.exp(t - 1000)
for enum in ans_list]
183     return swap_list
184
185 def is_ans_list(ans_list_new):
186     for enum in ans_list_new:

```

```

187         if enum < 0 or enum > 1:
188             return False
189         return True
190 # todo: 设定初始值
191 t = 1000          # 初始化温度值
192 t_min = 100      # 设置温度下限
193 # alpha = 0.99    # 设置温度下降率
194 k = 10000        # 设置迭代次数
195 # todo: 给定一个初始解,
196 ans_list = [0.5, 0.2, 0.1, 0.1, 0.1, 0.1]
197 ans_w = 0
198 ans_all = []
199 # todo: 开始退火过程
200 # while (t > t_min):
201 for i in range(50):
202     for i in range(k):
203         ans_w = get_w(ans_list)
204         ans_list_new = get_new_ans_list(ans_list, t)
205         if is_ans_list(ans_list_new):
206             # print("ok")
207             ans_w_new = get_w(ans_list_new)
208             delta = ans_w_new[0] - ans_w[0]
209             if delta > 0:
210                 ans_list = ans_list_new
211             # else:
212             #     p = get_p(delta, t)
213             #     # print(p)
214             #     if (p < random.random()):
215             #         ans_list = ans_list_new
216         # t = t * alpha
217         t -= 0.2
218         print(f"得到的净利润: {ans_w[0]}, 环境成本: {ans_w[1]}, 信用成本: {ans_w[2]}")
219         print(f"最优解为: {ans_list[0]}, {ans_list[1]}, {ans_list[2]}, {ans_list[3]}, {ans_list[4]}, {ans_list[5]}")
220         # print(f"最优解为: {ans_list[0]}, {ans_list[1]}, {ans_list[2]}, {ans_list[3]}")
221         ans_all.append(ans_w[0])
222     print(f"最终得到的净利润: {ans_w[0]}, 环境成本: {ans_w[1]}, 信用成本: {ans_w[2]}")
223
224
225 iteration = list(range(1, len(ans_all) + 1))
226 plt.figure()
227 plt.plot(iteration, ans_all, marker='', color='b', linestyle='--')
228 plt.xlabel('迭代次数')
229 plt.ylabel('净利润')
230 plt.title('模拟退火优化可视图')
231 plt.grid(True)
232 plt.show()
233
234 import numpy as np
235 import pandas as pd
236 import random

```

```

237 import matplotlib.pyplot as plt
238 import numpy as np
239
240 plt.rcParams['font.sans-serif'] = ['KaiTi']
241 plt.rcParams['axes.unicode_minus'] = False
242
243
244 # todo: 过程函数
245 def get_new_part(n_part_ori, p_part_make, p_part_test):
246     # 零件检测函数, 输入零件数、次品率、检测率, 得到新零件数、新次品率、检测数
247     n_part_good = n_part_ori * (1 - p_part_make)
248     n_part_bad = n_part_ori * p_part_make
249     n_part_bad_test = n_part_bad * p_part_test
250     n_part_new = n_part_ori - n_part_bad_test
251     p_part_new = 1 - n_part_good / n_part_new
252     return n_part_new, p_part_new, n_part_ori * p_part_test
253
254
255 def get_product(n_parts, p_parts, p_product_make):
256     # 输入每种零件的个数, 得到产品生产数、次品数、正品数
257     p_good = 1 - p_product_make
258     for enum in p_parts:
259         p_good *= 1 - enum
260     n_parts = np.array(n_parts)
261     n_product = np.min(n_parts)
262     n_product_good = n_product * p_good
263     n_product_bad = n_product - n_product_good
264     return n_product, n_product_good, n_product_bad
265
266
267 def deal_product_bad2(n_product_bad, p_parts):
268     # 输入次品数、每个零件的次品率, 得到每种新的零件个数, 每种新的零件的次品率 (一个成品
    # 由两个零件构成)
269     n_part_1 = n_product_bad
270     n_part_2 = n_product_bad
271     p_swap = p_parts[0] + p_parts[1] - p_parts[0] * p_parts[1] + (1 - p_parts[0]) * (1 -
    p_parts[1]) * p_product_make
272     p_part_1 = p_parts[0] / p_swap
273     p_part_2 = p_parts[1] / p_swap
274     return n_part_1, n_part_2, p_part_1, p_part_2
275
276
277 # 循环仿真直至不能盈利
278 # 此处使用法一 (全程固定) 进行循环处理
279 print("此处使用法一 (全程固定) 进行循环处理")
280 # todo: 决策变量
281 # 总盈利值、丢弃成本, 调换成本
282 w_1 = 0
283 w_2 = 0
284 w_3 = 0
285 # todo: 决策方案变量

```

```

286 # 检测率、拆解率
287 p_part1_test = 0.1
288 p_part2_test = 0.1
289 p_product_test = 0.1
290 p_product_dismantle = 0.1
291
292 p_part1_test0 = 0.1
293 p_part2_test0 = 0.1
294 # todo: 环境变量
295 # 次品率
296
297 p_part1_make = 0.1
298 p_part2_make = 0.1
299 p_product_make = 0.1
300 # 购买或者制作单价
301 m_part1_buy = 4
302 m_part2_buy = 18
303 m_product_buy = 6    # 制作单价
304 # 测试单价
305 m_part1_test = 2
306 m_part2_test = 3
307 m_product_test = 3
308 # 售价、拆解价格、调换价格
309 m_product_sale = 56
310 m_product_dismantle = 5
311 m_product_exchange = 6
312 # 零件数（总是希望利用率最大）
313 n_part1_ori = 10000
314 n_part2_ori = 10000
315 # todo: 仿真交换变量
316 n_part1_swap = 0
317 p_part1_swap = 0
318 n_part2_swap = 0
319 p_part2_swap = 0
320 # todo: 仿真过程(每次仿真得到的新的决策变量应当在前一次的基础上改变)
321 count = 0
322 ans_all = []
323 while True:
324     # 继承上次循环决策变量
325     w_1_new, w_2_new, w_3_new = w_1, w_2, w_3
326     count += 1
327     print(f"当前是第{count}次循环")
328     # todo: 零件检测
329     if count == 1:
330         # 首次进行时，需计算零件成本，计算零件检测成本
331         w_1_new -= n_part1_ori * m_part1_buy + n_part2_ori * m_part2_buy
332         # 零件检测函数，输入零件数、次品率、检测率，得到新零件数、新次品率、检测数
333         n_part1_new, p_part1_new, n_part1_test = get_new_part(n_part1_ori, p_part1_make,
334             p_part1_test)
334         n_part2_new, p_part2_new, n_part2_test = get_new_part(n_part2_ori, p_part2_make,
335             p_part2_test)

```

```

335     w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
336     w_2_new += (n_part1_ori - n_part1_new) * m_part1_buy + (n_part2_ori - n_part2_new
) * m_part2_buy
337     print(n_part1_new, p_part1_new, n_part1_test)
338     print(n_part2_new, p_part2_new, n_part2_test)
339 else:
340     # 其他情况，产品零件在最后一步进行处理，此处进行继承
341     n_part1_new, p_part1_new = n_part1_swap, p_part1_swap
342     n_part2_new, p_part2_new = n_part2_swap, p_part2_swap
343 # todo: 生成产品
344 n_product, n_product_good, n_product_bad = get_product([n_part1_new, n_part2_new], [
p_part1_new, p_part2_new],
345                                                         p_product_make)
346 print(n_product, n_product_good, n_product_bad)
347 w_1_new -= n_product * m_product_buy
348 # todo: 计算决策变量（次品默认丢弃），区分产品正品与次品
349 w_1_new += n_product_good * m_product_sale - n_product_good * p_product_test *
m_product_test
350 w_1_new -= n_product_bad * p_product_test * m_product_test + n_product_bad * (1 -
p_product_test) * m_product_exchange
351 w_2_new += n_product_bad * (m_part1_buy + m_part2_buy + m_product_buy)
352 w_3_new += n_product_bad * (1 - p_product_test) * m_product_exchange
353 print(f"当前得到的净利润: {w_1_new}, 环境成本: {w_2_new}, 信用成本: {w_3_new}")
354 ans_all.append(w_1_new)
355 # print(f"上一轮得到的净利润: {w_1}, 环境成本: {w_2}, 信用成本: {w_3}")
356 # todo: 判断循环是否结束
357 # if w_1_new - w_1 <= w_1 * 0.05 or p_product_dismantle == 0:
358 #
359 #     break
360 if count > 5:
361     break
362 # todo: 处理产品次品
363 n_part1_new1, n_part2_new1, p_part1_new1, p_part2_new1 = deal_product_bad2(
n_product_bad * p_product_dismantle,
364
[
p_part1_new, p_part2_new])
365 # print(n_part1_new1, n_part2_new1, p_part1_new1, p_part2_new1)
366 w_1_new -= (n_product_bad * p_product_dismantle) * m_product_dismantle
367 w_2_new -= (n_product_bad * p_product_dismantle) * (m_part1_buy + m_part2_buy +
m_product_buy)
368 # todo: 对于拆解得到的零件，此处讨论是否进行检验
369 # 零件检测函数，输入零件数、次品率、检测率，得到新零件数、新次品率、检测数
370 n_part1_new, p_part1_new, n_part1_test = get_new_part(n_part1_new1, p_part1_new1,
p_part1_test0)
371 n_part2_new, p_part2_new, n_part2_test = get_new_part(n_part2_new1, p_part2_new1,
p_part2_test0)
372 # print(n_part1_new, p_part1_new, n_part1_test)
373 # print(n_part2_new, p_part2_new, n_part2_test)
374
375 w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
376 w_2_new += (n_part1_new1 - n_part1_new) * m_part1_buy + (n_part2_new1 - n_part2_new)

```

```

377     * m_part2_buy
378     # todo: 保留迭代值
379     w_1, w_2, w_3 = w_1_new, w_2_new, w_3_new
380     n_part1_swap, p_part1_swap, n_part2_swap, p_part2_swap = n_part1_new, p_part1_new,
381     n_part2_new, p_part2_new
382
383     print(f"最终得到的净利润: {w_1_new}, 环境成本: {w_2_new}, 信用成本: {w_3_new}")
384     iteration = list(range(1, len(ans_all) + 1))
385     plt.figure()
386     plt.plot(iteration, ans_all, marker='', color='b', linestyle='--')
387     plt.xlabel('循环次数')
388     plt.ylabel('净利润')
389     plt.title('仿真循环可视图')
390     plt.grid(True)
391     plt.show()
392
393
394
395
396

```

题二代码

```

1     import matplotlib.pyplot as plt
2     import numpy as np
3     from itertools import product
4
5     plt.rcParams['font.sans-serif'] = ['KaiTi']
6     plt.rcParams['axes.unicode_minus'] = False
7
8
9     def get_ans(x_list):
10         # todo: 过程函数
11         def get_new_part(n_part_ori, p_part_make, p_part_test):
12             # 零件检测函数, 输入零件数、次品率、检测率, 得到新零件数、新次品率、检测数
13             n_part_good = n_part_ori * (1 - p_part_make)
14             n_part_bad = n_part_ori * p_part_make
15             n_part_bad_test = n_part_bad * p_part_test
16             n_part_new = n_part_ori - n_part_bad_test
17             p_part_new = 1 - n_part_good / n_part_new
18             return n_part_new, p_part_new, n_part_ori * p_part_test
19
20         def get_product(n_parts, p_parts, p_product_make):
21             # 输入每种零件的个数, 得到产品生产数、次品数、正品数
22             p_good = 1 - p_product_make
23             for enum in p_parts:
24                 p_good *= 1 - enum
25             n_parts = np.array(n_parts)
26             n_product = np.min(n_parts)

```

```

27     n_product_good = n_product * p_good
28     n_product_bad = n_product - n_product_good
29     return n_product, n_product_good, n_product_bad
30
31 def deal_product_bad2(n_product_bad, p_parts, p_product_make):
32     # 输入次品数、每个零件的次品率，制作工艺，得到每种新的零件个数，每种新的零件的次
33     # 品率（一个成品由两个零件构成）
34     n_part_1 = n_product_bad
35     n_part_2 = n_product_bad
36     p_swap = p_parts[0] + p_parts[1] - p_parts[0] * p_parts[1] + (1 - p_parts[0]) * (
37         1 - p_parts[1]) * p_product_make
38     p_part_1 = p_parts[0] / p_swap
39     p_part_2 = p_parts[1] / p_swap
40     return n_part_1, n_part_2, p_part_1, p_part_2
41
42 def deal_product_bad3(n_product_bad, p_parts, p_product_make):
43     # 输入次品数、每个零件的次品率，得到每种新的零件个数，每种新的零件的次品率（一个
44     # 成品由三个零件构成）
45     n_part_1 = n_product_bad
46     n_part_2 = n_product_bad
47     n_part_3 = n_product_bad
48
49     p_good = 1 - p_product_make
50     for enum in p_parts:
51         p_good *= 1 - enum
52     p_swap = 1 - p_good
53     p_part_1 = p_parts[0] / p_swap
54     p_part_2 = p_parts[1] / p_swap
55     p_part_3 = p_parts[2] / p_swap
56     return n_part_1, n_part_2, n_part_3, p_part_1, p_part_2, p_part_3
57
58 # todo: 决策变量
59 # 总盈利值
60 w_1 = 0
61 # todo: 决策方案变量
62 # 检测率、拆解率
63 p_part1_test = x_list[0]
64 p_part2_test = x_list[1]
65 p_part3_test = x_list[2]
66 p_part4_test = x_list[3]
67 p_part5_test = x_list[4]
68 p_part6_test = x_list[5]
69 p_part7_test = x_list[6]
70 p_part8_test = x_list[7]
71 p_half_product1_test = x_list[8]
72 p_half_product2_test = x_list[9]
73 p_half_product3_test = x_list[10]
74 p_product_test = x_list[11]
75 # p_half_product1_dismantle = 0
76 # p_half_product2_dismantle = 0
77 # p_half_product3_dismantle = 0

```

```

76 # p_product_dismantle = 0
77 p_half_product1_dismantle = x_list[12]
78 p_half_product2_dismantle = x_list[13]
79 p_half_product3_dismantle = x_list[14]
80 p_product_dismantle = x_list[15]
81 # todo: 新的参数, 用于第二次及以后的迭代
82 p_part1_test0 = 1 if x_list[0] == 0 else 0
83 p_part2_test0 = 1 if x_list[1] == 0 else 0
84 p_part3_test0 = 1 if x_list[2] == 0 else 0
85 p_part4_test0 = 1 if x_list[3] == 0 else 0
86 p_part5_test0 = 1 if x_list[4] == 0 else 0
87 p_part6_test0 = 1 if x_list[5] == 0 else 0
88 p_part7_test0 = 1 if x_list[6] == 0 else 0
89 p_part8_test0 = 1 if x_list[7] == 0 else 0
90 # p_half_product1_test0 = x_list[16]
91 # p_half_product2_test0 = x_list[17]
92 # p_half_product3_test0 = x_list[18]
93 p_half_product1_test0 = 1 if x_list[8] == 0 else 0
94 p_half_product2_test0 = 1 if x_list[9] == 0 else 0
95 p_half_product3_test0 = 1 if x_list[10] == 0 else 0
96 # p_part1_test0 = x_list[16]
97 # p_part2_test0 = x_list[17]
98 # p_part3_test0 = x_list[18]
99 # p_part4_test0 = x_list[19]
100 # p_part5_test0 = x_list[20]
101 # p_part6_test0 = x_list[21]
102 # p_part7_test0 = x_list[22]
103 # p_part8_test0 = x_list[23]
104 # p_half_product1_test0 = x_list[24]
105 # p_half_product2_test0 = x_list[25]
106 # p_half_product3_test0 = x_list[26]
107 # todo: 环境变量
108 # 次品率
109 p_part1_make = 0.1
110 p_part2_make = 0.1
111 p_part3_make = 0.1
112 p_part4_make = 0.1
113 p_part5_make = 0.1
114 p_part6_make = 0.1
115 p_part7_make = 0.1
116 p_part8_make = 0.1
117 p_half_product1_make = 0.1
118 p_half_product2_make = 0.1
119 p_half_product3_make = 0.1
120 p_product_make = 0.1
121 # 购买或者制作单价
122 m_part1_buy = 2
123 m_part2_buy = 8
124 m_part3_buy = 12
125 m_part4_buy = 2
126 m_part5_buy = 8

```

```

127 m_part6_buy = 12
128 m_part7_buy = 8
129 m_part8_buy = 12
130 m_half_product1_buy = 8
131 m_half_product2_buy = 8
132 m_half_product3_buy = 8
133 m_product_buy = 8
134 # 测试单价
135 m_part1_test = 1
136 m_part2_test = 1
137 m_part3_test = 2
138 m_part4_test = 1
139 m_part5_test = 1
140 m_part6_test = 2
141 m_part7_test = 1
142 m_part8_test = 2
143 m_half_product1_test = 4
144 m_half_product2_test = 4
145 m_half_product3_test = 4
146 m_product_test = 6
147 # 售价、拆解价格、调换价格
148 m_product_sale = 200
149 m_product_exchange = 40
150 m_half_product1_dismantle = 6
151 m_half_product2_dismantle = 6
152 m_half_product3_dismantle = 6
153 m_product_dismantle = 10
154 # todo: 初始零件数
155 n_part1_ori = 10000
156 n_part2_ori = 10000
157 n_part3_ori = 10000
158 n_part4_ori = 10000
159 n_part5_ori = 10000
160 n_part6_ori = 10000
161 n_part7_ori = 10000
162 n_part8_ori = 10000
163 # todo: 循环过程变量（用于维护迭代）
164 n_part1_swap = 0
165 n_part2_swap = 0
166 n_part3_swap = 0
167 n_part4_swap = 0
168 n_part5_swap = 0
169 n_part6_swap = 0
170 n_part7_swap = 0
171 n_part8_swap = 0
172 n_half_product1_swap = 0
173 n_half_product2_swap = 0
174 n_half_product3_swap = 0
175 p_part1_swap = 0
176 p_part2_swap = 0
177 p_part3_swap = 0

```

```

178     p_part4_swap = 0
179     p_part5_swap = 0
180     p_part6_swap = 0
181     p_part7_swap = 0
182     p_part8_swap = 0
183     p_half_product1_swap = 0
184     p_half_product2_swap = 0
185     p_half_product3_swap = 0
186     # todo: 开始循环
187     ans_list = []
188     count = 0
189     while True:
190         # todo: 继承循环
191         w_1_new = w_1
192         count += 1
193         # print(f"当前是第{count}次循环")
194         # todo: 零件检测
195         if count == 1:
196             # todo: 计算零件成本
197             w_1_new -= n_part1_ori * m_part1_buy + n_part2_ori * m_part2_buy
198             w_1_new -= n_part3_ori * m_part3_buy + n_part4_ori * m_part4_buy
199             w_1_new -= n_part5_ori * m_part5_buy + n_part6_ori * m_part6_buy
200             w_1_new -= n_part7_ori * m_part7_buy + n_part8_ori * m_part8_buy
201             # print(w_1_new)
202             # todo: 零件进行检测, 更新数据
203             n_part1_new, p_part1_make_new, n_part1_test = get_new_part(n_part1_ori,
204 p_part1_make, p_part1_test)
205             n_part2_new, p_part2_make_new, n_part2_test = get_new_part(n_part2_ori,
206 p_part2_make, p_part2_test)
207             n_part3_new, p_part3_make_new, n_part3_test = get_new_part(n_part3_ori,
208 p_part3_make, p_part3_test)
209             n_part4_new, p_part4_make_new, n_part4_test = get_new_part(n_part4_ori,
210 p_part4_make, p_part4_test)
211             n_part5_new, p_part5_make_new, n_part5_test = get_new_part(n_part5_ori,
212 p_part5_make, p_part5_test)
213             n_part6_new, p_part6_make_new, n_part6_test = get_new_part(n_part6_ori,
214 p_part6_make, p_part6_test)
215             n_part7_new, p_part7_make_new, n_part7_test = get_new_part(n_part7_ori,
216 p_part7_make, p_part7_test)
217             n_part8_new, p_part8_make_new, n_part8_test = get_new_part(n_part8_ori,
218 p_part8_make, p_part8_test)
219             n_half_product1_new, p_half_product1_new = 0, 0
220             n_half_product2_new, p_half_product2_new = 0, 0
221             n_half_product3_new, p_half_product3_new = 0, 0
222             # todo: 计算检测费
223             w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
224             w_1_new -= n_part3_test * m_part3_test + n_part4_test * m_part4_test
225             w_1_new -= n_part5_test * m_part5_test + n_part6_test * m_part6_test
226             w_1_new -= n_part7_test * m_part7_test + n_part8_test * m_part8_test
227             # print(w_1_new)
228         else:

```

```

221         n_part1_new, p_part1_new = n_part1_swap, p_part1_swap
222         n_part2_new, p_part2_new = n_part2_swap, p_part2_swap
223         n_part3_new, p_part3_new = n_part3_swap, p_part3_swap
224         n_part4_new, p_part4_new = n_part4_swap, p_part4_swap
225         n_part5_new, p_part5_new = n_part5_swap, p_part5_swap
226         n_part6_new, p_part6_new = n_part6_swap, p_part6_swap
227         n_part7_new, p_part7_new = n_part7_swap, p_part7_swap
228         n_part8_new, p_part8_new = n_part8_swap, p_part8_swap
229         n_half_product1_new, p_half_product1_new = n_half_product1_swap,
p_half_product1_swap
230         n_half_product2_new, p_half_product2_new = n_half_product2_swap,
p_half_product2_swap
231         n_half_product3_new, p_half_product3_new = n_half_product3_swap,
p_half_product3_swap
232         # todo: 生产半成品
233         # 输入每种零件的个数，每种零件的次品率，工艺的次品率， 返回得到的产品总数、次品
数、正品数
234         # print(f"零件数: {n_part1_new}")
235         n_half_product1, n_half_product1_good, n_half_product1_bad = \
236         get_product([n_part1_new, n_part2_new, n_part3_new], [p_part1_make_new,
p_part2_make_new, p_part3_make_new],
237                     p_half_product1_make)
238         n_half_product2, n_half_product2_good, n_half_product2_bad = \
239         get_product([n_part4_new, n_part5_new, n_part6_new], [p_part4_make_new,
p_part5_make_new, p_part6_make_new],
240                     p_half_product2_make)
241         n_half_product3, n_half_product3_good, n_half_product3_bad = \
242         get_product([n_part7_new, n_part8_new], [p_part7_make_new, p_part8_make_new],
p_half_product3_make)
243         # print(n_half_product1, n_half_product2, n_half_product3)
244         # todo: 计算制作费用
245         w_1_new -= n_half_product1 * m_half_product1_buy
246         w_1_new -= n_half_product2 * m_half_product2_buy
247         w_1_new -= n_half_product3 * m_half_product3_buy
248         # print(w_1_new)
249         # todo: 得到剩余零件数，得到剩余零件数次品率
250         n_part1_ex1 = n_part1_new - n_half_product1
251         n_part2_ex1 = n_part2_new - n_half_product1
252         n_part3_ex1 = n_part3_new - n_half_product1
253         n_part4_ex1 = n_part4_new - n_half_product2
254         n_part5_ex1 = n_part5_new - n_half_product2
255         n_part6_ex1 = n_part6_new - n_half_product2
256         n_part7_ex1 = n_part7_new - n_half_product3
257         n_part8_ex1 = n_part8_new - n_half_product3
258         p_part1_ex = p_part1_make_new
259         p_part2_ex = p_part2_make_new
260         p_part3_ex = p_part3_make_new
261         p_part4_ex = p_part4_make_new
262         p_part5_ex = p_part5_make_new
263         p_part6_ex = p_part6_make_new
264         p_part7_ex = p_part7_make_new

```

```

265     p_part8_ex = p_part8_make_new
266     # todo: 得到当前半成品数, 得到当前半成品次品率
267     p_half_product1_make_new = (n_half_product1_bad + n_half_product1_new *
p_half_product1_new) / (
268         n_half_product1 + n_half_product1_new)
269     n_half_product1_new = n_half_product1 + n_half_product1_new
270     p_half_product2_make_new = (n_half_product2_bad + n_half_product2_new *
p_half_product2_new) / (
271         n_half_product2 + n_half_product2_new)
272     n_half_product2_new = n_half_product2 + n_half_product2_new
273     p_half_product3_make_new = (n_half_product3_bad + n_half_product3_new *
p_half_product3_new) / (
274         n_half_product3 + n_half_product3_new)
275     n_half_product3_new = n_half_product3 + n_half_product3_new
276     # print(f"{n_half_product1_new}, {n_half_product2_new}, {n_half_product3_new}")
277     if count == 1:
278         # todo: 保留半成品次品
279         n_half_product1_ex2 = p_half_product1_make_new * n_half_product1_new *
p_half_product1_test
280         n_half_product2_ex2 = p_half_product2_make_new * n_half_product2_new *
p_half_product2_test
281         n_half_product3_ex2 = p_half_product3_make_new * n_half_product3_new *
p_half_product3_test
282         # todo: 检测半成品
283         n_half_product1_new, p_half_product1_make_new, n_half_product1_test =
get_new_part(n_half_product1_new,
284
p_half_product1_make_new,
285
p_half_product1_test)
286         n_half_product2_new, p_half_product2_make_new, n_half_product2_test =
get_new_part(n_half_product2_new,
287
p_half_product2_make_new,
288
p_half_product2_test)
289         n_half_product3_new, p_half_product3_make_new, n_half_product3_test =
get_new_part(n_half_product3_new,
290
p_half_product3_make_new,
291
p_half_product3_test)
292     else:
293         # todo: 保留半成品次品
294         n_half_product1_ex2 = p_half_product1_make_new * n_half_product1_new *
p_half_product1_test0
295         n_half_product2_ex2 = p_half_product2_make_new * n_half_product2_new *
p_half_product2_test0
296         n_half_product3_ex2 = p_half_product3_make_new * n_half_product3_new *
p_half_product3_test0
297         # todo: 检测半成品

```

```

298         n_half_product1_new, p_half_product1_make_new, n_half_product1_test =
get_new_part(n_half_product1_new,
299
300         p_half_product1_make_new,
301
302         p_half_product1_test0)
303         n_half_product2_new, p_half_product2_make_new, n_half_product2_test =
get_new_part(n_half_product2_new,
304
305         p_half_product2_make_new,
306
307         p_half_product2_test0)
308         n_half_product3_new, p_half_product3_make_new, n_half_product3_test =
get_new_part(n_half_product3_new,
309
310         p_half_product3_make_new,
311
312         p_half_product3_test0)
313     # todo: 计算检测费
314     w_1_new -= n_half_product1_test * m_half_product1_test + n_half_product2_test *
m_half_product2_test + n_half_product3_test * m_half_product3_test
315     # print(w_1_new)
316
317     # todo: 生产产品
318     # 输入每种零件的个数，每种零件的次品率，工艺的次品率， 返回得到的产品总数、次品
数、正品数
319     n_product, n_product_good, n_product_bad = \
320         get_product([n_half_product1_new, n_half_product2_new, n_half_product3_new],
321                     [p_half_product1_make_new, p_half_product2_make_new,
322                     p_half_product3_make_new], p_product_make)
323     # todo: 得到剩余半成品数， 得到剩余半成品次品率
324     n_half_product1_ex = n_half_product1_new - n_product
325     n_half_product2_ex = n_half_product2_new - n_product
326     n_half_product3_ex = n_half_product3_new - n_product
327     p_half_product1_ex = p_half_product1_make_new
328     p_half_product2_ex = p_half_product2_make_new
329     p_half_product3_ex = p_half_product3_make_new
330     # todo: 计算制作费用
331     w_1_new -= n_product * m_product_buy
332     # todo: 计算决策变量
333     w_1_new += n_product_good * m_product_sale - n_product_good * p_product_test *
m_product_test
334     w_1_new -= n_product_bad * p_product_test * m_product_test + n_product_bad * (
335         1 - p_product_test) * m_product_exchange
336     ans_list.append(w_1_new)
337     # print(f"当前第{count}轮得到的净利润: {w_1_new}")
338     # todo: 判断循环是否结束
339     # if w_1_new - w_1 <= w_1 * 0.001 or p_product_dismantle == 0:
340     # if p_product_dismantle == 0:
341     # if count > 1 and w_1_new < w_1 * 1.001:
342     #     # print("循环结束")

```

```

336         # break
337         if count > 20:
338             break
339         # todo: 成品次品拆成半成品
340         n_half_product1_new1, n_half_product2_new1, n_half_product3_new1,
p_half_product1_new1, p_half_product2_new1, p_half_product3_new1 = \
341             deal_product_bad3(n_product_bad * p_product_dismantle,
342                             [p_half_product1_make_new, p_half_product2_make_new,
p_half_product3_make_new],
343                             p_product_make)
344         # todo: 计算拆解费
345         w_1_new -= n_half_product1_new1 * m_product_dismantle
346         ## todo: 处理半成品次品(前方遗留半成品有检测不合格和多余, 此处检测不合格应当不
进行检测, 数目多余与拆除之后的进行检测进行检测)
347         # todo: 得到当前未检验半成品数目, (多余与拆除之后)
348         if n_half_product1_new1 + n_half_product1_ex == 0:
349             p_half_product1_new1 = 0
350             n_half_product1_new1 = 0
351         else:
352             p_half_product1_new1 = (n_half_product1_new1 * p_half_product1_new1 +
n_half_product1_ex * p_half_product1_ex) / (n_half_product1_new1 + n_half_product1_ex
)
353             n_half_product1_new1 = n_half_product1_new1 + n_half_product1_ex
354         if n_half_product2_new1 + n_half_product2_ex == 0:
355             p_half_product2_new1 = 0
356             n_half_product2_new1 = 0
357         else:
358             p_half_product2_new1 = (n_half_product2_new1 * p_half_product2_new1 +
n_half_product2_ex * p_half_product2_ex) / (n_half_product2_new1 + n_half_product2_ex
)
359             n_half_product2_new1 = n_half_product2_new1 + n_half_product2_ex
360         if n_half_product3_new1 + n_half_product3_ex == 0:
361             p_half_product3_new1 = 0
362             n_half_product3_new1 = 0
363         else:
364             p_half_product3_new1 = (n_half_product3_new1 * p_half_product3_new1 +
n_half_product3_ex * p_half_product3_ex) / (n_half_product3_new1 + n_half_product3_ex
)
365             n_half_product3_new1 = n_half_product3_new1 + n_half_product3_ex
366         # todo: 半成品拆成零件(仅拆前方留下半成品次品)
367         n_part1_new1, n_part2_new1, n_part3_new1, p_part1_new1, p_part2_new1,
p_part3_new1 = \
368             deal_product_bad3(n_half_product1_ex2 * p_half_product1_dismantle,
369                             [p_part1_make, p_part1_make, p_part1_make],
p_half_product1_make)
370         n_part4_new1, n_part5_new1, n_part6_new1, p_part4_new1, p_part5_new1,
p_part6_new1 = \
371             deal_product_bad3(n_half_product2_ex2 * p_half_product2_dismantle,
372                             [p_part4_make, p_part5_make, p_part6_make],
p_half_product2_make)
373         n_part7_new1, n_part8_new1, p_part7_new1, p_part8_new1 = \

```

```

374         deal_product_bad2(n_half_product3_ex2 * p_half_product3_dismantle, [
p_part7_make, p_part8_make],
375                             p_half_product3_make)
376     # todo: 计算半成品拆解费
377     w_l_new -= n_part1_new1 * m_half_product1_dismantle + n_part4_new1 *
m_half_product2_dismantle + n_part7_new1 * m_half_product3_dismantle
378     # todo: 得到当前零件数, 得到当前零件次品率
379     if n_part1_new1 + n_part1_ex1 == 0:
380         p_part1_new1 = 0
381         n_part1_new1 = 0
382     else:
383         p_part1_new1 = (n_part1_new1 * p_part1_new1 + n_part1_ex1 * p_part1_ex) / (
n_part1_new1 + n_part1_ex1)
384         n_part1_new1 = n_part1_new1 + n_part1_ex1
385     if n_part2_new1 + n_part2_ex1 == 0:
386         p_part2_new1 = 0
387         n_part2_new1 = 0
388     else:
389         p_part2_new1 = (n_part2_new1 * p_part2_new1 + n_part2_ex1 * p_part2_ex) / (
n_part2_new1 + n_part2_ex1)
390         n_part2_new1 = n_part2_new1 + n_part2_ex1
391     if n_part3_new1 + n_part3_ex1 == 0:
392         p_part3_new1 = 0
393         n_part3_new1 = 0
394     else:
395         p_part3_new1 = (n_part3_new1 * p_part3_new1 + n_part3_ex1 * p_part3_ex) / (
n_part3_new1 + n_part3_ex1)
396         n_part3_new1 = n_part3_new1 + n_part3_ex1
397     if n_part4_new1 + n_part4_ex1 == 0:
398         p_part4_new1 = 0
399         n_part4_new1 = 0
400     else:
401         p_part4_new1 = (n_part4_new1 * p_part4_new1 + n_part4_ex1 * p_part4_ex) / (
n_part4_new1 + n_part4_ex1)
402         n_part4_new1 = n_part4_new1 + n_part4_ex1
403     if n_part5_new1 + n_part5_ex1 == 0:
404         p_part5_new1 = 0
405         n_part5_new1 = 0
406     else:
407         p_part5_new1 = (n_part5_new1 * p_part5_new1 + n_part5_ex1 * p_part5_ex) / (
n_part5_new1 + n_part5_ex1)
408         n_part5_new1 = n_part5_new1 + n_part5_ex1
409     if n_part6_new1 + n_part6_ex1 == 0:
410         p_part6_new1 = 0
411         n_part6_new1 = 0
412     else:
413         p_part6_new1 = (n_part6_new1 * p_part6_new1 + n_part6_ex1 * p_part6_ex) / (
n_part6_new1 + n_part6_ex1)
414         n_part6_new1 = n_part6_new1 + n_part6_ex1
415     if n_part7_new1 + n_part7_ex1 == 0:
416         p_part7_new1 = 0

```

```

417         n_part7_new1 = 0
418     else:
419         p_part7_new1 = (n_part7_new1 * p_part7_new1 + n_part7_ex1 * p_part7_ex) / (
n_part7_new1 + n_part7_ex1)
420         n_part7_new1 = n_part7_new1 + n_part7_ex1
421         if n_part8_new1 + n_part8_ex1 == 0:
422             p_part8_new1 = 0
423             n_part8_new1 = 0
424         else:
425             p_part8_new1 = (n_part8_new1 * p_part8_new1 + n_part8_ex1 * p_part8_ex) / (
n_part8_new1 + n_part8_ex1)
426             n_part8_new1 = n_part8_new1 + n_part8_ex1
427         if n_part1_new + n_part2_new + n_part3_new + n_part4_new + n_part5_new +
n_part6_new + n_part7_new + n_part8_new == 0 and n_half_product1_swap +
n_half_product2_swap + n_half_product3_swap == 0:
428             break
429         # print(n_part1_new1)
430         # todo: 处理零件
431         if n_part1_new1 > 0:
432             n_part1_new, p_part1_make_new, n_part1_test = get_new_part(n_part1_new1,
p_part1_new1, p_part1_test0)
433         else:
434             n_part1_new, p_part1_make_new, n_part1_test = 0, 0, 0
435         if n_part2_new1 > 0:n_part2_new, p_part2_make_new, n_part2_test = get_new_part(
n_part2_new1, p_part2_new1, p_part2_test0)
436         else:
437             n_part2_new, p_part2_make_new, n_part2_test = 0, 0, 0
438         if n_part3_new1 > 0:n_part3_new, p_part3_make_new, n_part3_test = get_new_part(
n_part3_new1, p_part3_new1, p_part3_test0)
439         else:
440             n_part3_new, p_part3_make_new, n_part3_test = 0, 0, 0
441         if n_part4_new1 > 0:n_part4_new, p_part4_make_new, n_part4_test = get_new_part(
n_part4_new1, p_part4_new1, p_part4_test0)
442         else:
443             n_part4_new, p_part4_make_new, n_part4_test = 0, 0, 0
444         if n_part5_new1 > 0:n_part5_new, p_part5_make_new, n_part5_test = get_new_part(
n_part5_new1, p_part5_new1, p_part5_test0)
445         else:
446             n_part5_new, p_part5_make_new, n_part5_test = 0, 0, 0
447         if n_part6_new1 > 0:n_part6_new, p_part6_make_new, n_part6_test = get_new_part(
n_part6_new1, p_part6_new1, p_part6_test0)
448         else:
449             n_part6_new, p_part6_make_new, n_part6_test = 0, 0, 0
450         if n_part7_new1 > 0:n_part7_new, p_part7_make_new, n_part7_test = get_new_part(
n_part7_new1, p_part7_new1, p_part7_test0)
451         else:
452             n_part7_new, p_part7_make_new, n_part7_test = 0, 0, 0
453         if n_part8_new1 > 0:n_part8_new, p_part8_make_new, n_part8_test = get_new_part(
n_part8_new1, p_part8_new1, p_part8_test0)
454         else:
455             n_part8_new, p_part8_make_new, n_part8_test = 0, 0, 0

```

```

456 # todo: 计算检测费
457 # print(n_part1_test * m_part1_test)
458 w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
459 w_1_new -= n_part3_test * m_part3_test + n_part4_test * m_part4_test
460 w_1_new -= n_part5_test * m_part5_test + n_part6_test * m_part6_test
461 w_1_new -= n_part7_test * m_part7_test + n_part8_test * m_part8_test
462
463 # todo: 处理迭代
464 # if n_part1_new + n_part2_new + n_part3_new + n_part4_new + n_part5_new +
n_part6_new + n_part7_new + n_part8_new == 0:
465     # break
466 # if n_half_product1_swap + n_half_product2_swap + n_half_product3_swap == 0:
467     # break
468 # print(n_part1_new)
469 n_part1_swap = n_part1_new
470 n_part2_swap = n_part2_new
471 n_part3_swap = n_part3_new
472 n_part4_swap = n_part4_new
473 n_part5_swap = n_part5_new
474 n_part6_swap = n_part6_new
475 n_part7_swap = n_part7_new
476 n_part8_swap = n_part8_new
477 n_half_product1_swap = n_half_product1_new1
478 n_half_product2_swap = n_half_product2_new1
479 n_half_product3_swap = n_half_product3_new1
480 p_part1_swap = p_part1_make_new
481 p_part2_swap = p_part2_make_new
482 p_part3_swap = p_part3_make_new
483 p_part4_swap = p_part4_make_new
484 p_part5_swap = p_part5_make_new
485 p_part6_swap = p_part6_make_new
486 p_part7_swap = p_part7_make_new
487 p_part8_swap = p_part8_make_new
488 p_half_product1_swap = p_half_product1_new1
489 p_half_product2_swap = p_half_product2_new1
490 p_half_product3_swap = p_half_product3_new1
491 w_1 = w_1_new
492 # print(f"最终得到的净利润: {w_1_new}")
493 print(f"总循环数: {count}")
494 # print(ans_list)
495 # if len(ans_list) == 1:
496 #     return ans_list[0]
497 # if ans_list[-2] > ans_list[-1]:
498 #     return ans_list[-2]
499 # return ans_list[-1]
500 return ans_list
501
502
503 print(get_ans([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1]))
504 print([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1])
505 # print(get_ans([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1]))

```

```

506 # ans_maybe_list = list(product([0, 1], repeat=16))
507 # ans_list = []
508 # count = 0
509 # for maybe_list in ans_maybe_list:
510 #     count += 1
511 #     print(count)
512 #     ans_list.append([maybe_list, get_ans(maybe_list)])
513 # # print(sorted(ans_list, key=lambda x: x[1]))
514 # ans_list = [enum for enum in ans_list if enum[1] > 0]
515 # ans_list = sorted(ans_list, key=lambda x: x[1])
516 # for enum in ans_list:
517 #     # if enum[1] < 2000000:
518 #         print(enum)
519 print(f"最速收敛: {[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1]}")
520 print(f"全局最优: {[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1]}")
521
522 x = [_ + 1 for _ in range(21)]
523 y1 = get_ans([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1]) # 最速收敛
524 y2 = get_ans([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1]) # 全局最优
525 plt.figure(figsize=(10, 6))
526 plt.subplot(111)
527 plt.plot(x, y1, 'red', label='最速收敛', marker='o')
528 plt.plot(x, y2, color='blue', label='全局最优', marker='D')
529 plt.legend()
530 plt.title('问题三决策 利润上升曲线')
531 plt.xlabel('循环次数')
532 plt.ylabel('净利润')
533 plt.xticks(x)
534 plt.grid(True)
535 plt.show()
536

```

题三代码

```

1  import random
2  import math
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6  plt.rcParams['font.sans-serif'] = ['KaiTi']
7  plt.rcParams['axes.unicode_minus'] = False
8
9
10
11 def get_w(data_list):
12     # 此处进行仿真，为时间角度考虑，仅循环两次
13     ans_list = []
14     # todo: 过程函数
15     def get_new_part(n_part_ori, p_part_make, p_part_test):
16         # 零件检测函数，输入零件数、次品率、检测率，得到新零件数、新次品率、检测数
17         n_part_good = n_part_ori * (1 - p_part_make)

```

```

18     n_part_bad = n_part_ori * p_part_make
19     n_part_bad_test = n_part_bad * p_part_test
20     n_part_new = n_part_ori - n_part_bad_test
21     p_part_new = 1 - n_part_good / n_part_new
22     return n_part_new, p_part_new, n_part_ori * p_part_test
23
24 def get_product(n_parts, p_parts, p_product_make):
25     # 输入每种零件的个数，得到产品生产数、次品数、正品数
26     p_good = 1 - p_product_make
27     for enum in p_parts:
28         p_good *= 1 - enum
29     n_parts = np.array(n_parts)
30     n_product = np.min(n_parts)
31     n_product_good = n_product * p_good
32     n_product_bad = n_product - n_product_good
33     return n_product, n_product_good, n_product_bad
34
35 def deal_product_bad2(n_product_bad, p_parts):
36     # 输入次品数、每个零件的次品率，得到每种新的零件个数，每种新的零件的次品率（一个
    成品由两个零件构成）
37     n_part_1 = n_product_bad
38     n_part_2 = n_product_bad
39     p_swap = p_parts[0] + p_parts[1] - p_parts[0] * p_parts[1] + (1 - p_parts[0]) * (
40         1 - p_parts[1]) * p_product_make
41     p_part_1 = p_parts[0] / p_swap
42     p_part_2 = p_parts[1] / p_swap
43     return n_part_1, n_part_2, p_part_1, p_part_2
44
45 # 循环仿真直至不能盈利
46 # 此处使用法一（全程固定）进行循环处理
47 # print("此处使用法一（全程固定）进行循环处理")
48 # todo: 决策变量
49 # 总盈利值、丢弃成本，调换成本
50 w_1 = 0
51 w_2 = 0
52 w_3 = 0
53 # todo: 决策方案变量
54 # 检测率、拆解率
55 p_part1_test = data_list[0]
56 p_part2_test = data_list[1]
57 p_product_test = data_list[2]
58 p_product_dismantle = data_list[3]
59
60 p_part1_test0 = data_list[4]
61 p_part2_test0 = data_list[5]
62 # todo: 环境变量
63 # 次品率
64 p_part1_make = 0.10592778
65 p_part2_make = 0.10592778
66 p_product_make = 0.1
67 # 购买或者制作单价

```

```

68     m_part1_buy = 4
69     m_part2_buy = 18
70     m_product_buy = 6 # 制作单价
71     # 测试单价
72     m_part1_test = 2
73     m_part2_test = 3
74     m_product_test = 3
75     # 售价、拆解价格、调换价格
76     m_product_sale = 56
77     m_product_dismantle = 5
78     m_product_exchange = 6
79     # 零件数（总是希望利用率最大）
80     n_part1_ori = 10000
81     n_part2_ori = 10000
82     # todo: 仿真交换变量
83     n_part1_swap = 0
84     p_part1_swap = 0
85     n_part2_swap = 0
86     p_part2_swap = 0
87     # todo: 仿真过程(每次仿真得到的新的决策变量应当在前一次的基础上改变)
88     count = 0
89     while True:
90         # print("asd")
91         # 继承上次循环决策变量
92         w_1_new, w_2_new, w_3_new = w_1, w_2, w_3
93         count += 1
94         # print(f"当前是第{count}次循环")
95         # todo: 零件检测
96         if count == 1:
97             # 首次进行时, 需计算零件成本, 计算零件检测成本
98             w_1_new -= n_part1_ori * m_part1_buy + n_part2_ori * m_part2_buy
99             # 零件检测函数, 输入零件数、次品率、检测率, 得到新零件数、新次品率、检测数
100             n_part1_new, p_part1_new, n_part1_test = get_new_part(n_part1_ori,
101 p_part1_make, p_part1_test)
102             n_part2_new, p_part2_new, n_part2_test = get_new_part(n_part2_ori,
103 p_part2_make, p_part2_test)
104             w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
105             w_2_new += (n_part1_ori - n_part1_new) * m_part1_buy + (n_part2_ori -
106 n_part2_new) * m_part2_buy
107             # print("asd")
108             # print(n_part1_new, p_part1_new, n_part1_test)
109             # print(n_part2_new, p_part2_new, n_part2_test)
110         else:
111             # 其他情况, 产品零件在最后一步进行处理, 此处进行继承
112             n_part1_new, p_part1_new = n_part1_swap, p_part1_swap
113             n_part2_new, p_part2_new = n_part2_swap, p_part2_swap
114             # todo: 生成产品
115             n_product, n_product_good, n_product_bad = get_product([n_part1_new, n_part2_new
116 ], [p_part1_new, p_part2_new],
117                                     p_product_make)
118             n_part1_ex = n_part1_new - n_product

```

```

115     n_part2_ex = n_part2_new - n_product
116     p_part1_ex = p_part1_new
117     p_part2_ex = p_part2_new
118     # print(n_part1_ex, n_part2_ex)
119     # print(p_part1_ex, p_part2_ex)
120     # print(n_product, n_product_good, n_product_bad)
121     w_1_new -= n_product * m_product_buy
122     # todo: 计算决策变量（次品默认丢弃），区分产品正品与次品
123     w_1_new += n_product_good * m_product_sale - n_product_good * p_product_test *
m_product_test
124     w_1_new -= n_product_bad * p_product_test * m_product_test + n_product_bad * (
125         1 - p_product_test) * m_product_exchange
126     # print(n_product_bad * p_product_test)
127     # print(n_product_bad * (
128         1 - p_product_test))
129     w_2_new += n_product_bad * (m_part1_buy + m_part2_buy + m_product_buy)
130     w_3_new += n_product_bad * m_product_exchange
131     # w_3_new += n_product_bad * (1 - p_product_test) * m_product_exchange
132     # print(f"当前得到的净利润: {w_1_new}, 环境成本: {w_2_new}, 信用成本: {w_3_new}")
133     ans_list.append([w_1_new, w_2_new, w_3_new])
134     # print(f"上一轮得到的净利润: {w_1}, 环境成本: {w_2}, 信用成本: {w_3}")
135     # todo: 判断循环是否结束
136     if w_1_new - w_1 <= w_1 * 0.001 or p_product_dismantle == 0:
137         break
138     # todo: 处理产品次品
139     n_part1_new1, n_part2_new1, p_part1_new1, p_part2_new1 = deal_product_bad2(
n_product_bad * p_product_dismantle,
140
141     p_part1_new, p_part2_new])
142     # print(n_part1_new1, n_part2_new1, p_part1_new1, p_part2_new1)
143     # print(n_part1_new1 + n_part1_ex)
144     # print(n_part2_new1 + n_part2_ex)
145     p_part1_new1 = (n_part1_new1 * p_part1_new1 + n_part1_ex * p_part1_ex) / (
n_part1_new1 + n_part1_ex)
146     n_part1_new1 = n_part1_new1 + n_part1_ex
147     p_part2_new1 = (n_part2_new1 * p_part2_new1 + n_part2_ex * p_part2_ex) / (
n_part2_new1 + n_part2_ex)
148     n_part2_new1 = n_part2_new1 + n_part2_ex
149     # print(n_part1_new1, n_part2_new1, p_part1_new1, p_part2_new1)
150     w_1_new -= (n_product_bad * p_product_dismantle) * m_product_dismantle
151     w_2_new -= (n_product_bad * p_product_dismantle) * (m_part1_buy + m_part2_buy +
m_product_buy)
152     # todo: 对于拆解得到的零件，此处讨论是否进行检验
153     # 零件检测函数，输入零件数、次品率、检测率，得到新零件数、新次品率、检测数
154     n_part1_new, p_part1_new, n_part1_test = get_new_part(n_part1_new1, p_part1_new1,
p_part1_test0)
155     n_part2_new, p_part2_new, n_part2_test = get_new_part(n_part2_new1, p_part2_new1,
p_part2_test0)
156     # print(n_part1_new, p_part1_new, n_part1_test)
157     # print(n_part2_new, p_part2_new, n_part2_test)

```

```

158     w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
159     w_2_new += (n_part1_new1 - n_part1_new) * m_part1_buy + (n_part2_new1 -
n_part2_new) * m_part2_buy
160     # todo: 保留迭代值
161     w_1, w_2, w_3 = w_1_new, w_2_new, w_3_new
162     n_part1_swap, p_part1_swap, n_part2_swap, p_part2_swap = n_part1_new, p_part1_new
, n_part2_new, p_part2_new
163
164     # print(f"最终得到的净利润: {w_1_new}, 环境成本: {w_2_new}, 信用成本: {w_3_new}")
165     if len(ans_list) == 1:
166         return ans_list[0]
167     if ans_list[-2][0] > ans_list[-1][0]:
168         return ans_list[-2]
169     return ans_list[-1]
170
171 def get_p(delta, t):
172     return math.exp(delta/t)
173
174 def get_new_ans_list(ans_list, t):
175     swap_list = []
176     for i in range(len(ans_list)):
177         # print(+ 2 * random.random() * math.exp(t - 1000) - math.exp(t - 1000))
178         swap_list.append(ans_list[i] + 2 * random.random() * math.exp(t - 1000) - math.
exp(t - 1000))
179     # print(swap_list)
180     # swap_list = [enum + 2 * random.random() * math.exp(t - 1000) - math.exp(t - 1000)
for enum in ans_list]
181     return swap_list
182
183 def is_ans_list(ans_list_new):
184     for enum in ans_list_new:
185         if enum < 0 or enum > 1:
186             return False
187     return True
188 # todo: 设定初始值
189 t = 1000          # 初始化温度值
190 t_min = 100      # 设置温度下限
191 # alpha = 0.99    # 设置温度下降率
192 k = 10000        # 设置迭代次数
193 # todo: 给定一个初始解,
194 ans_list = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
195 ans_w = 0
196 ans_all = []
197 # todo: 开始退火过程
198 # while (t > t_min):
199 for i in range(50):
200     for i in range(k):
201         ans_w = get_w(ans_list)
202         ans_list_new = get_new_ans_list(ans_list, t)
203         if is_ans_list(ans_list_new):
204             # print("ok")

```

```

205         ans_w_new = get_w(ans_list_new)
206         delta = ans_w_new[0] - ans_w[0]
207         if delta > 0:
208             ans_list = ans_list_new
209         else:
210             p = get_p(delta, t)
211             # print(p)
212             if (p < random.random()):
213                 ans_list = ans_list_new
214
215         t -= 0.2
216         print(f"得到的净利润: {ans_w[0]}, 环境成本: {ans_w[1]}, 信用成本: {ans_w[2]}")
217         print(f"最优解为: {ans_list[0]}, {ans_list[1]}, {ans_list[2]}, {ans_list[3]}, {ans_list[4]}, {ans_list[5]}")
218         # print(f"最优解为: {ans_list[0]}, {ans_list[1]}, {ans_list[2]}, {ans_list[3]}")
219         ans_all.append(ans_w[0])
220     print(f"最终得到的净利润: {ans_w[0]}, 环境成本: {ans_w[1]}, 信用成本: {ans_w[2]}")
221
222     # iteration = list(range(1, len(ans_all) + 1))
223     # plt.figure()
224     # plt.plot(iteration, ans_all, marker='', color='b', linestyle='--')
225     # plt.xlabel('迭代次数')
226     # plt.ylabel('净利润')
227     # plt.title('模拟退火优化可视图')
228     # plt.grid(True)
229     # plt.show()
230
231     # 我们抽样得到的次品率10.0%
232     # 可能的真实次品率均值: 0.09998004390341249
233     # 可能的真实次品率置信区间: [0.09418352 0.10592778]
234     # 我们抽样得到的次品率5.0%
235     # 可能的真实次品率均值: 0.05008980243464378
236     # 可能的真实次品率置信区间: [0.04590519 0.05444447]
237     # 我们抽样得到的次品率20.0%
238     # 可能的真实次品率均值: 0.19976052684094991
239     # 可能的真实次品率置信区间: [0.19199026 0.20764429]
240
241     # 情况一:
242     # 偏小扰动: [1, 0, 0, 1, 0, 1]
243     # 偏大扰动: [1, 0, 0, 1, 0, 1]
244
245     import matplotlib.pyplot as plt
246     import numpy as np
247     from itertools import product
248
249     plt.rcParams['font.sans-serif'] = ['KaiTi']
250     plt.rcParams['axes.unicode_minus'] = False
251
252
253     def get_ans(x_list):
254         # todo: 过程函数

```

```

255 def get_new_part(n_part_ori, p_part_make, p_part_test):
256     # 零件检测函数，输入零件数、次品率、检测率，得到新零件数、新次品率、检测数
257     n_part_good = n_part_ori * (1 - p_part_make)
258     n_part_bad = n_part_ori * p_part_make
259     n_part_bad_test = n_part_bad * p_part_test
260     n_part_new = n_part_ori - n_part_bad_test
261     p_part_new = 1 - n_part_good / n_part_new
262     return n_part_new, p_part_new, n_part_ori * p_part_test
263
264 def get_product(n_parts, p_parts, p_product_make):
265     # 输入每种零件的个数，得到产品生产数、次品数、正品数
266     p_good = 1 - p_product_make
267     for enum in p_parts:
268         p_good *= 1 - enum
269     n_parts = np.array(n_parts)
270     n_product = np.min(n_parts)
271     n_product_good = n_product * p_good
272     n_product_bad = n_product - n_product_good
273     return n_product, n_product_good, n_product_bad
274
275 def deal_product_bad2(n_product_bad, p_parts, p_product_make):
276     # 输入次品数、每个零件的次品率，制作工艺，得到每种新的零件个数，每种新的零件的次
277     # 品率（一个成品由两个零件构成）
278     n_part_1 = n_product_bad
279     n_part_2 = n_product_bad
280     p_swap = p_parts[0] + p_parts[1] - p_parts[0] * p_parts[1] + (1 - p_parts[0]) * (
281         1 - p_parts[1]) * p_product_make
282     p_part_1 = p_parts[0] / p_swap
283     p_part_2 = p_parts[1] / p_swap
284     return n_part_1, n_part_2, p_part_1, p_part_2
285
286 def deal_product_bad3(n_product_bad, p_parts, p_product_make):
287     # 输入次品数、每个零件的次品率，得到每种新的零件个数，每种新的零件的次品率（一个
288     # 成品由三个零件构成）
289     n_part_1 = n_product_bad
290     n_part_2 = n_product_bad
291     n_part_3 = n_product_bad
292
293     p_good = 1 - p_product_make
294     for enum in p_parts:
295         p_good *= 1 - enum
296     p_swap = 1 - p_good
297     p_part_1 = p_parts[0] / p_swap
298     p_part_2 = p_parts[1] / p_swap
299     p_part_3 = p_parts[2] / p_swap
300     return n_part_1, n_part_2, n_part_3, p_part_1, p_part_2, p_part_3
301
302 # todo: 决策变量
303 # 总盈利值
304 w_1 = 0
305 # todo: 决策方案变量

```

```

304 # 检测率、拆解率
305 p_part1_test = x_list[0]
306 p_part2_test = x_list[1]
307 p_part3_test = x_list[2]
308 p_part4_test = x_list[3]
309 p_part5_test = x_list[4]
310 p_part6_test = x_list[5]
311 p_part7_test = x_list[6]
312 p_part8_test = x_list[7]
313 p_half_product1_test = x_list[8]
314 p_half_product2_test = x_list[9]
315 p_half_product3_test = x_list[10]
316 p_product_test = x_list[11]
317 # p_half_product1_dismantle = 0
318 # p_half_product2_dismantle = 0
319 # p_half_product3_dismantle = 0
320 # p_product_dismantle = 0
321 p_half_product1_dismantle = x_list[12]
322 p_half_product2_dismantle = x_list[13]
323 p_half_product3_dismantle = x_list[14]
324 p_product_dismantle = x_list[15]
325 # todo: 新的参数, 用于第二次及以后的迭代
326 p_part1_test0 = 1 if x_list[0] == 0 else 0
327 p_part2_test0 = 1 if x_list[1] == 0 else 0
328 p_part3_test0 = 1 if x_list[2] == 0 else 0
329 p_part4_test0 = 1 if x_list[3] == 0 else 0
330 p_part5_test0 = 1 if x_list[4] == 0 else 0
331 p_part6_test0 = 1 if x_list[5] == 0 else 0
332 p_part7_test0 = 1 if x_list[6] == 0 else 0
333 p_part8_test0 = 1 if x_list[7] == 0 else 0
334 # p_half_product1_test0 = x_list[16]
335 # p_half_product2_test0 = x_list[17]
336 # p_half_product3_test0 = x_list[18]
337 p_half_product1_test0 = 1 if x_list[8] == 0 else 0
338 p_half_product2_test0 = 1 if x_list[9] == 0 else 0
339 p_half_product3_test0 = 1 if x_list[10] == 0 else 0
340 # p_part1_test0 = x_list[16]
341 # p_part2_test0 = x_list[17]
342 # p_part3_test0 = x_list[18]
343 # p_part4_test0 = x_list[19]
344 # p_part5_test0 = x_list[20]
345 # p_part6_test0 = x_list[21]
346 # p_part7_test0 = x_list[22]
347 # p_part8_test0 = x_list[23]
348 # p_half_product1_test0 = x_list[24]
349 # p_half_product2_test0 = x_list[25]
350 # p_half_product3_test0 = x_list[26]
351 # todo: 环境变量
352 # 次品率
353 p_part1_make = 0.10592778
354 p_part2_make = 0.10592778

```

```

355 p_part3_make = 0.10592778
356 p_part4_make = 0.10592778
357 p_part5_make = 0.10592778
358 p_part6_make = 0.10592778
359 p_part7_make = 0.10592778
360 p_part8_make = 0.10592778
361 p_half_product1_make = 0.1
362 p_half_product2_make = 0.1
363 p_half_product3_make = 0.1
364 p_product_make = 0.1
365 # 购买或者制作单价
366 m_part1_buy = 2
367 m_part2_buy = 8
368 m_part3_buy = 12
369 m_part4_buy = 2
370 m_part5_buy = 8
371 m_part6_buy = 12
372 m_part7_buy = 8
373 m_part8_buy = 12
374 m_half_product1_buy = 8
375 m_half_product2_buy = 8
376 m_half_product3_buy = 8
377 m_product_buy = 8
378 # 测试单价
379 m_part1_test = 1
380 m_part2_test = 1
381 m_part3_test = 2
382 m_part4_test = 1
383 m_part5_test = 1
384 m_part6_test = 2
385 m_part7_test = 1
386 m_part8_test = 2
387 m_half_product1_test = 4
388 m_half_product2_test = 4
389 m_half_product3_test = 4
390 m_product_test = 6
391 # 售价、拆解价格、调换价格
392 m_product_sale = 200
393 m_product_exchange = 40
394 m_half_product1_dismantle = 6
395 m_half_product2_dismantle = 6
396 m_half_product3_dismantle = 6
397 m_product_dismantle = 10
398 # todo: 初始零件数
399 n_part1_ori = 10000
400 n_part2_ori = 10000
401 n_part3_ori = 10000
402 n_part4_ori = 10000
403 n_part5_ori = 10000
404 n_part6_ori = 10000
405 n_part7_ori = 10000

```

```

406     n_part8_ori = 10000
407     # todo: 循环过程变量（用于维护迭代）
408     n_part1_swap = 0
409     n_part2_swap = 0
410     n_part3_swap = 0
411     n_part4_swap = 0
412     n_part5_swap = 0
413     n_part6_swap = 0
414     n_part7_swap = 0
415     n_part8_swap = 0
416     n_half_product1_swap = 0
417     n_half_product2_swap = 0
418     n_half_product3_swap = 0
419     p_part1_swap = 0
420     p_part2_swap = 0
421     p_part3_swap = 0
422     p_part4_swap = 0
423     p_part5_swap = 0
424     p_part6_swap = 0
425     p_part7_swap = 0
426     p_part8_swap = 0
427     p_half_product1_swap = 0
428     p_half_product2_swap = 0
429     p_half_product3_swap = 0
430     # todo: 开始循环
431     ans_list = []
432     count = 0
433     while True:
434         # todo: 继承循环
435         w_1_new = w_1
436         count += 1
437         # print(f"当前是第{count}次循环")
438         # todo: 零件检测
439         if count == 1:
440             # todo: 计算零件成本
441             w_1_new -= n_part1_ori * m_part1_buy + n_part2_ori * m_part2_buy
442             w_1_new -= n_part3_ori * m_part3_buy + n_part4_ori * m_part4_buy
443             w_1_new -= n_part5_ori * m_part5_buy + n_part6_ori * m_part6_buy
444             w_1_new -= n_part7_ori * m_part7_buy + n_part8_ori * m_part8_buy
445             # print(w_1_new)
446             # todo: 零件进行检测，更新数据
447             n_part1_new, p_part1_make_new, n_part1_test = get_new_part(n_part1_ori,
p_part1_make, p_part1_test)
448             n_part2_new, p_part2_make_new, n_part2_test = get_new_part(n_part2_ori,
p_part2_make, p_part2_test)
449             n_part3_new, p_part3_make_new, n_part3_test = get_new_part(n_part3_ori,
p_part3_make, p_part3_test)
450             n_part4_new, p_part4_make_new, n_part4_test = get_new_part(n_part4_ori,
p_part4_make, p_part4_test)
451             n_part5_new, p_part5_make_new, n_part5_test = get_new_part(n_part5_ori,
p_part5_make, p_part5_test)

```

```

452         n_part6_new, p_part6_make_new, n_part6_test = get_new_part(n_part6_ori,
p_part6_make, p_part6_test)
453         n_part7_new, p_part7_make_new, n_part7_test = get_new_part(n_part7_ori,
p_part7_make, p_part7_test)
454         n_part8_new, p_part8_make_new, n_part8_test = get_new_part(n_part8_ori,
p_part8_make, p_part8_test)
455         n_half_product1_new, p_half_product1_new = 0, 0
456         n_half_product2_new, p_half_product2_new = 0, 0
457         n_half_product3_new, p_half_product3_new = 0, 0
458         # todo: 计算检测费
459         w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
460         w_1_new -= n_part3_test * m_part3_test + n_part4_test * m_part4_test
461         w_1_new -= n_part5_test * m_part5_test + n_part6_test * m_part6_test
462         w_1_new -= n_part7_test * m_part7_test + n_part8_test * m_part8_test
463         # print(w_1_new)
464     else:
465         n_part1_new, p_part1_new = n_part1_swap, p_part1_swap
466         n_part2_new, p_part2_new = n_part2_swap, p_part2_swap
467         n_part3_new, p_part3_new = n_part3_swap, p_part3_swap
468         n_part4_new, p_part4_new = n_part4_swap, p_part4_swap
469         n_part5_new, p_part5_new = n_part5_swap, p_part5_swap
470         n_part6_new, p_part6_new = n_part6_swap, p_part6_swap
471         n_part7_new, p_part7_new = n_part7_swap, p_part7_swap
472         n_part8_new, p_part8_new = n_part8_swap, p_part8_swap
473         n_half_product1_new, p_half_product1_new = n_half_product1_swap,
p_half_product1_swap
474         n_half_product2_new, p_half_product2_new = n_half_product2_swap,
p_half_product2_swap
475         n_half_product3_new, p_half_product3_new = n_half_product3_swap,
p_half_product3_swap
476         # todo: 生产半成品
477         # 输入每种零件的个数，每种零件的次品率，工艺的次品率， 返回得到的产品总数、次品
数、正品数
478         # print(f"零件数: {n_part1_new}")
479         n_half_product1, n_half_product1_good, n_half_product1_bad = \
480             get_product([n_part1_new, n_part2_new, n_part3_new], [p_part1_make_new,
p_part2_make_new, p_part3_make_new],
481                         p_half_product1_make)
482         n_half_product2, n_half_product2_good, n_half_product2_bad = \
483             get_product([n_part4_new, n_part5_new, n_part6_new], [p_part4_make_new,
p_part5_make_new, p_part6_make_new],
484                         p_half_product2_make)
485         n_half_product3, n_half_product3_good, n_half_product3_bad = \
486             get_product([n_part7_new, n_part8_new], [p_part7_make_new, p_part8_make_new],
p_half_product3_make)
487         # print(n_half_product1, n_half_product2, n_half_product3)
488         # todo: 计算制作费用
489         w_1_new -= n_half_product1 * m_half_product1_buy
490         w_1_new -= n_half_product2 * m_half_product2_buy
491         w_1_new -= n_half_product3 * m_half_product3_buy
492         # print(w_1_new)

```

```

493 # todo: 得到剩余零件数, 得到剩余零件数次品率
494 n_part1_ex1 = n_part1_new - n_half_product1
495 n_part2_ex1 = n_part2_new - n_half_product1
496 n_part3_ex1 = n_part3_new - n_half_product1
497 n_part4_ex1 = n_part4_new - n_half_product2
498 n_part5_ex1 = n_part5_new - n_half_product2
499 n_part6_ex1 = n_part6_new - n_half_product2
500 n_part7_ex1 = n_part7_new - n_half_product3
501 n_part8_ex1 = n_part8_new - n_half_product3
502 p_part1_ex = p_part1_make_new
503 p_part2_ex = p_part2_make_new
504 p_part3_ex = p_part3_make_new
505 p_part4_ex = p_part4_make_new
506 p_part5_ex = p_part5_make_new
507 p_part6_ex = p_part6_make_new
508 p_part7_ex = p_part7_make_new
509 p_part8_ex = p_part8_make_new
510 # todo: 得到当前半成品数, 得到当前半成品次品率
511 p_half_product1_make_new = (n_half_product1_bad + n_half_product1_new *
p_half_product1_new) / (
512     n_half_product1 + n_half_product1_new)
513 n_half_product1_new = n_half_product1 + n_half_product1_new
514 p_half_product2_make_new = (n_half_product2_bad + n_half_product2_new *
p_half_product2_new) / (
515     n_half_product2 + n_half_product2_new)
516 n_half_product2_new = n_half_product2 + n_half_product2_new
517 p_half_product3_make_new = (n_half_product3_bad + n_half_product3_new *
p_half_product3_new) / (
518     n_half_product3 + n_half_product3_new)
519 n_half_product3_new = n_half_product3 + n_half_product3_new
520 # print(f"{n_half_product1_new}, {n_half_product2_new}, {n_half_product3_new}")
521 if count == 1:
522     # todo: 保留半成品次品
523     n_half_product1_ex2 = p_half_product1_make_new * n_half_product1_new *
p_half_product1_test
524     n_half_product2_ex2 = p_half_product2_make_new * n_half_product2_new *
p_half_product2_test
525     n_half_product3_ex2 = p_half_product3_make_new * n_half_product3_new *
p_half_product3_test
526     # todo: 检测半成品
527     n_half_product1_new, p_half_product1_make_new, n_half_product1_test =
get_new_part(n_half_product1_new,
528
p_half_product1_make_new,
529
p_half_product1_test)
530     n_half_product2_new, p_half_product2_make_new, n_half_product2_test =
get_new_part(n_half_product2_new,
531
p_half_product2_make_new,
532

```

```

        p_half_product2_test)
533     n_half_product3_new, p_half_product3_make_new, n_half_product3_test =
get_new_part(n_half_product3_new,
534
        p_half_product3_make_new,
535
        p_half_product3_test)
536 else:
537     # todo: 保留半成品次品
538     n_half_product1_ex2 = p_half_product1_make_new * n_half_product1_new *
p_half_product1_test0
539     n_half_product2_ex2 = p_half_product2_make_new * n_half_product2_new *
p_half_product2_test0
540     n_half_product3_ex2 = p_half_product3_make_new * n_half_product3_new *
p_half_product3_test0
541     # todo: 检测半成品
542     n_half_product1_new, p_half_product1_make_new, n_half_product1_test =
get_new_part(n_half_product1_new,
543
        p_half_product1_make_new,
544
        p_half_product1_test0)
545     n_half_product2_new, p_half_product2_make_new, n_half_product2_test =
get_new_part(n_half_product2_new,
546
        p_half_product2_make_new,
547
        p_half_product2_test0)
548     n_half_product3_new, p_half_product3_make_new, n_half_product3_test =
get_new_part(n_half_product3_new,
549
        p_half_product3_make_new,
550
        p_half_product3_test0)
551     # todo: 计算检测费
552     w_1_new -= n_half_product1_test * m_half_product1_test + n_half_product2_test *
m_half_product2_test + n_half_product3_test * m_half_product3_test
553     # print(w_1_new)
554
555     # todo: 生产产品
556     # 输入每种零件的个数，每种零件的次品率，工艺的次品率， 返回得到的产品总数、次品
数、正品数
557     n_product, n_product_good, n_product_bad = \
558         get_product([n_half_product1_new, n_half_product2_new, n_half_product3_new],
559                     [p_half_product1_make_new, p_half_product2_make_new,
p_half_product3_make_new], p_product_make)
560     # todo: 得到剩余半成品数， 得到剩余半成品次品率
561     n_half_product1_ex = n_half_product1_new - n_product
562     n_half_product2_ex = n_half_product2_new - n_product
563     n_half_product3_ex = n_half_product3_new - n_product
564     p_half_product1_ex = p_half_product1_make_new

```

```

565     p_half_product2_ex = p_half_product2_make_new
566     p_half_product3_ex = p_half_product3_make_new
567     # todo: 计算制作费用
568     w_1_new -= n_product * m_product_buy
569     # todo: 计算决策变量
570     w_1_new += n_product_good * m_product_sale - n_product_good * p_product_test *
m_product_test
571     w_1_new -= n_product_bad * p_product_test * m_product_test + n_product_bad * (
572         1 - p_product_test) * m_product_exchange
573     ans_list.append(w_1_new)
574     # print(f"当前第{count}轮得到的净利润: {w_1_new}")
575     # todo: 判断循环是否结束
576     # if w_1_new - w_1 <= w_1 * 0.001 or p_product_dismantle == 0:
577     # if p_product_dismantle == 0:
578     # if count > 1 and w_1_new < w_1 * 1.001:
579     #     # print("循环结束")
580     #     break
581     if count > 20:
582         break
583     # todo: 成品次品拆成半成品
584     n_half_product1_new1, n_half_product2_new1, n_half_product3_new1,
p_half_product1_new1, p_half_product2_new1, p_half_product3_new1 = \
585         deal_product_bad3(n_product_bad * p_product_dismantle,
586             [p_half_product1_make_new, p_half_product2_make_new,
p_half_product3_make_new],
587             p_product_make)
588     # todo: 计算拆解费
589     w_1_new -= n_half_product1_new1 * m_product_dismantle
590     # # todo: 处理半成品次品(前方遗留半成品有检测不合格和多余, 此处检测不合格应当不进
进行检测, 数目多余与拆除之后的进行检测进行检测)
591     # todo: 得到当前未检验半成品数目, (多余与拆除之后)
592     if n_half_product1_new1 + n_half_product1_ex == 0:
593         p_half_product1_new1 = 0
594         n_half_product1_new1 = 0
595     else:
596         p_half_product1_new1 = (n_half_product1_new1 * p_half_product1_new1 +
n_half_product1_ex * p_half_product1_ex) / (n_half_product1_new1 + n_half_product1_ex
)
597         n_half_product1_new1 = n_half_product1_new1 + n_half_product1_ex
598     if n_half_product2_new1 + n_half_product2_ex == 0:
599         p_half_product2_new1 = 0
600         n_half_product2_new1 = 0
601     else:
602         p_half_product2_new1 = (n_half_product2_new1 * p_half_product2_new1 +
n_half_product2_ex * p_half_product2_ex) / (n_half_product2_new1 + n_half_product2_ex
)
603         n_half_product2_new1 = n_half_product2_new1 + n_half_product2_ex
604     if n_half_product3_new1 + n_half_product3_ex == 0:
605         p_half_product3_new1 = 0
606         n_half_product3_new1 = 0
607     else:

```

```

608         p_half_product3_new1 = (n_half_product3_new1 * p_half_product3_new1 +
n_half_product3_ex * p_half_product3_ex) / (n_half_product3_new1 + n_half_product3_ex
)
609
610         n_half_product3_new1 = n_half_product3_new1 + n_half_product3_ex
# todo: 半成品拆成零件（仅拆前方留下半成品次品）
611         n_part1_new1, n_part2_new1, n_part3_new1, p_part1_new1, p_part2_new1,
p_part3_new1 = \
612             deal_product_bad3(n_half_product1_ex2 * p_half_product1_dismantle,
613                               [p_part1_make, p_part1_make, p_part1_make],
p_half_product1_make)
614         n_part4_new1, n_part5_new1, n_part6_new1, p_part4_new1, p_part5_new1,
p_part6_new1 = \
615             deal_product_bad3(n_half_product2_ex2 * p_half_product2_dismantle,
616                               [p_part4_make, p_part5_make, p_part6_make],
p_half_product2_make)
617         n_part7_new1, n_part8_new1, p_part7_new1, p_part8_new1 = \
618             deal_product_bad2(n_half_product3_ex2 * p_half_product3_dismantle, [
p_part7_make, p_part8_make],
619                               p_half_product3_make)
620         # todo: 计算半成品拆解费
621         w_1_new -= n_part1_new1 * m_half_product1_dismantle + n_part4_new1 *
m_half_product2_dismantle + n_part7_new1 * m_half_product3_dismantle
622         # todo: 得到当前零件数，得到当前零件次品率
623         if n_part1_new1 + n_part1_ex1 == 0:
624             p_part1_new1 = 0
625             n_part1_new1 = 0
626         else:
627             p_part1_new1 = (n_part1_new1 * p_part1_new1 + n_part1_ex1 * p_part1_ex) / (
n_part1_new1 + n_part1_ex1)
628             n_part1_new1 = n_part1_new1 + n_part1_ex1
629         if n_part2_new1 + n_part2_ex1 == 0:
630             p_part2_new1 = 0
631             n_part2_new1 = 0
632         else:
633             p_part2_new1 = (n_part2_new1 * p_part2_new1 + n_part2_ex1 * p_part2_ex) / (
n_part2_new1 + n_part2_ex1)
634             n_part2_new1 = n_part2_new1 + n_part2_ex1
635         if n_part3_new1 + n_part3_ex1 == 0:
636             p_part3_new1 = 0
637             n_part3_new1 = 0
638         else:
639             p_part3_new1 = (n_part3_new1 * p_part3_new1 + n_part3_ex1 * p_part3_ex) / (
n_part3_new1 + n_part3_ex1)
640             n_part3_new1 = n_part3_new1 + n_part3_ex1
641         if n_part4_new1 + n_part4_ex1 == 0:
642             p_part4_new1 = 0
643             n_part4_new1 = 0
644         else:
645             p_part4_new1 = (n_part4_new1 * p_part4_new1 + n_part4_ex1 * p_part4_ex) / (
n_part4_new1 + n_part4_ex1)
646             n_part4_new1 = n_part4_new1 + n_part4_ex1

```

```

647         if n_part5_new1 + n_part5_ex1 == 0:
648             p_part5_new1 = 0
649             n_part5_new1 = 0
650         else:
651             p_part5_new1 = (n_part5_new1 * p_part5_new1 + n_part5_ex1 * p_part5_ex) / (
n_part5_new1 + n_part5_ex1)
652             n_part5_new1 = n_part5_new1 + n_part5_ex1
653         if n_part6_new1 + n_part6_ex1 == 0:
654             p_part6_new1 = 0
655             n_part6_new1 = 0
656         else:
657             p_part6_new1 = (n_part6_new1 * p_part6_new1 + n_part6_ex1 * p_part6_ex) / (
n_part6_new1 + n_part6_ex1)
658             n_part6_new1 = n_part6_new1 + n_part6_ex1
659         if n_part7_new1 + n_part7_ex1 == 0:
660             p_part7_new1 = 0
661             n_part7_new1 = 0
662         else:
663             p_part7_new1 = (n_part7_new1 * p_part7_new1 + n_part7_ex1 * p_part7_ex) / (
n_part7_new1 + n_part7_ex1)
664             n_part7_new1 = n_part7_new1 + n_part7_ex1
665         if n_part8_new1 + n_part8_ex1 == 0:
666             p_part8_new1 = 0
667             n_part8_new1 = 0
668         else:
669             p_part8_new1 = (n_part8_new1 * p_part8_new1 + n_part8_ex1 * p_part8_ex) / (
n_part8_new1 + n_part8_ex1)
670             n_part8_new1 = n_part8_new1 + n_part8_ex1
671         if n_part1_new + n_part2_new + n_part3_new + n_part4_new + n_part5_new +
n_part6_new + n_part7_new + n_part8_new == 0 and n_half_product1_swap +
n_half_product2_swap + n_half_product3_swap == 0:
672             break
673         # print(n_part1_new1)
674         # todo: 处理零件
675         if n_part1_new1 > 0:
676             n_part1_new, p_part1_make_new, n_part1_test = get_new_part(n_part1_new1,
p_part1_new1, p_part1_test0)
677         else:
678             n_part1_new, p_part1_make_new, n_part1_test = 0, 0, 0
679         if n_part2_new1 > 0: n_part2_new, p_part2_make_new, n_part2_test = get_new_part(
n_part2_new1, p_part2_new1, p_part2_test0)
680         else:
681             n_part2_new, p_part2_make_new, n_part2_test = 0, 0, 0
682         if n_part3_new1 > 0: n_part3_new, p_part3_make_new, n_part3_test = get_new_part(
n_part3_new1, p_part3_new1, p_part3_test0)
683         else:
684             n_part3_new, p_part3_make_new, n_part3_test = 0, 0, 0
685         if n_part4_new1 > 0: n_part4_new, p_part4_make_new, n_part4_test = get_new_part(
n_part4_new1, p_part4_new1, p_part4_test0)
686         else:
687             n_part4_new, p_part4_make_new, n_part4_test = 0, 0, 0

```

```

688         if n_part5_new1 > 0:n_part5_new, p_part5_make_new, n_part5_test = get_new_part(
n_part5_new1, p_part5_new1, p_part5_test0)
689     else:
690         n_part5_new, p_part5_make_new, n_part5_test = 0, 0, 0
691         if n_part6_new1 > 0:n_part6_new, p_part6_make_new, n_part6_test = get_new_part(
n_part6_new1, p_part6_new1, p_part6_test0)
692     else:
693         n_part6_new, p_part6_make_new, n_part6_test = 0, 0, 0
694         if n_part7_new1 > 0:n_part7_new, p_part7_make_new, n_part7_test = get_new_part(
n_part7_new1, p_part7_new1, p_part7_test0)
695     else:
696         n_part7_new, p_part7_make_new, n_part7_test = 0, 0, 0
697         if n_part8_new1 > 0:n_part8_new, p_part8_make_new, n_part8_test = get_new_part(
n_part8_new1, p_part8_new1, p_part8_test0)
698     else:
699         n_part8_new, p_part8_make_new, n_part8_test = 0, 0, 0
700     # todo: 计算检测费
701     # print(n_part1_test * m_part1_test)
702     w_1_new -= n_part1_test * m_part1_test + n_part2_test * m_part2_test
703     w_1_new -= n_part3_test * m_part3_test + n_part4_test * m_part4_test
704     w_1_new -= n_part5_test * m_part5_test + n_part6_test * m_part6_test
705     w_1_new -= n_part7_test * m_part7_test + n_part8_test * m_part8_test
706
707     # todo: 处理迭代
708     # if n_part1_new + n_part2_new + n_part3_new + n_part4_new + n_part5_new +
n_part6_new + n_part7_new + n_part8_new == 0:
709         # break
710     # if n_half_product1_swap + n_half_product2_swap + n_half_product3_swap == 0:
711         # break
712     # print(n_part1_new)
713     n_part1_swap = n_part1_new
714     n_part2_swap = n_part2_new
715     n_part3_swap = n_part3_new
716     n_part4_swap = n_part4_new
717     n_part5_swap = n_part5_new
718     n_part6_swap = n_part6_new
719     n_part7_swap = n_part7_new
720     n_part8_swap = n_part8_new
721     n_half_product1_swap = n_half_product1_new1
722     n_half_product2_swap = n_half_product2_new1
723     n_half_product3_swap = n_half_product3_new1
724     p_part1_swap = p_part1_make_new
725     p_part2_swap = p_part2_make_new
726     p_part3_swap = p_part3_make_new
727     p_part4_swap = p_part4_make_new
728     p_part5_swap = p_part5_make_new
729     p_part6_swap = p_part6_make_new
730     p_part7_swap = p_part7_make_new
731     p_part8_swap = p_part8_make_new
732     p_half_product1_swap = p_half_product1_new1
733     p_half_product2_swap = p_half_product2_new1

```

```

734     p_half_product3_swap = p_half_product3_new1
735     w_1 = w_1_new
736     # print(f"最终得到的净利润: {w_1_new}")
737     # print(f"总循环数: {count}")
738     # print(ans_list)
739     # if len(ans_list) == 1:
740     #     return ans_list[0]
741     # if ans_list[-2] > ans_list[-1]:
742     #     return ans_list[-2]
743     # return ans_list[-1]
744     return ans_list
745
746 ## 寻找策略
747 ## print(get_ans([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1]))
748 ## print([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1])
749 ## print(get_ans([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1]))
750 # ans_maybe_list = list(product([0, 1], repeat=16))
751 # ans_list = []
752 # count = 0
753 # for maybe_list in ans_maybe_list:
754 #     count += 1
755 #     print(count)
756 #     ans_list.append([maybe_list, get_ans(maybe_list)])
757 # # print(sorted(ans_list, key=lambda x: x[1]))
758 # ans_list = [enum for enum in ans_list if enum[1] > 0]
759 # ans_list = sorted(ans_list, key=lambda x: x[1])
760 # for enum in ans_list:
761 #     # if enum[1] < 2000000:
762 #     #     print(enum)
763
764 # 展示策略
765 print(f"最速收敛: {[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1]}")
766 print(f"全局最优: {[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1]}")
767 # [(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1), 448518.046585434]
768 # [(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1), 435188.3987616001]
769
770 x = [_ + 1 for _ in range(21)]
771 y1 = get_ans([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1]) # 最速收敛
772 y2 = get_ans([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1]) # 全局最优
773 plt.figure(figsize=(10, 6))
774 plt.subplot(111)
775 plt.plot(x, y1, 'red', label='最速收敛', marker='o')
776 plt.plot(x, y2, color='blue', label='全局最优', marker='D')
777 plt.legend()
778 plt.title('问题四决策 变大扰动 利润上升曲线')
779 plt.xlabel('循环次数')
780 plt.ylabel('净利润')
781 plt.xticks(x)
782 plt.grid(True)
783 plt.show()
784 print(y1)

```

```

785 print(y2)
786
787 # [0.09418352 0.10592778]
788
789 # 对于问题三，三种情况的决策和结果
790 # 正常
791 # [(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1), 448518.046585434]
792 # [(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1), 435188.3987616001]
793 # 偏小
794 # [(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1), 448491.9298575758]
795 # [(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1), 436619.5888]
796 # 偏大
797 # [(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1), 425153.86094869545]
798 # [(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1), 421234.6082]
799
800
801

```

题四代码