针对某电子产品生产过程最优决策研究

摘要

对于问题 1，本文使用**序贯抽样**，即随着抽样过程的进行，动态调整抽样次数。对于第一小问，取定 $p_0 = 0.1$，$p_1 = 0.12$，对于略大于 0.1 的真实次品率，平均抽样次数约为 1200，并向两侧快速递减至 100 左右；对于第二小问，取定 $p_0 = 0.08, p_1 = 0.1$，对于略小于 0.1 的真实次品率，平均抽样次数约为 900，并向两侧快速递减至 100 左右。此外本文对抽样模型的参数进行了灵敏度分析，模型对 p_0，p_1 的设置较为敏感，对 α，β 敏感度较低。本文还设置了抽样截断阈值为 1500，即当抽样次数达到 1500 仍无法接受或拒绝时，则使用正态分布近似的假设检验进行决策。

对于问题 2，本文首先对问题进行简化，设置零配件检测策略和零配件丢弃策略，并设定最大化利润率为目标。之后计算零配件合格的不合格成品期望收益，将迭代问题转化为树状结构使用**决策树模型**进行求解。同时本文也使用了**蒙特卡洛模型**对所有 16 种可能决策进行随机模拟。六种情况的最大利润率均在 20%-40% 之间，两种模型得出相同的结果，彼此印证，兼顾可解释性与全面性。此外，若将该问题视为**不确定型决策**，“两个零配件均检测、成品不检测、不合格的成品拆解”在绝大多数的决策准则下利润率最高。

对于问题 3，本文首先使用**动态拆解策略**，即一开始所有拆解均拆解到零配件，之后逐渐减少拆解幅度，并扩展问题 2 零配件检测策略，从零件开始逐层检验以确保每层的合格率。接着参考问题 2 中的思路，计算可能出现无限次迭代半成品的期望收益，从而将迭代问题转化为树状结构进行求解。对于给出的样例，由于参数空间较大，计算复杂度较高，本文选择了启发式算法进行优化求解，以**蒙特卡洛模拟**得到的利润率均值作为目标值，使用二进制对参数空间进行编码，最后使用**增强精英保留遗传算法 (SEGA)** 求解，得到最优决策为不检测零件、检测并拆解半成品、不检测但拆解成品，期望利润率为 26.90%。

对于问题 4，本文首先使用**贝叶斯推断**，选择 **Beta 分布**作为次品率 p 的先验分布，之后根据抽样结果，计算似然函数并利用贝叶斯公式更新对 p 的信念形成后验分布。在重做问题 2、问题 3 的过程中，不再将 p 设为固定值，而是在后验分布中随机采样。发现得到利润率分布的标准差显著增大后，参考**夏普比率**，使用超额利润率与其标准差的比值作为指标对各决策方案进行综合评价，发现问题 2 中大多数决策都选择了同时检测两个零件、问题 3 选择对成品也进行检测以对冲次品率波动带来的风险，而其期望利润率在两种情况下并无显著差异。

关键字：**序贯抽样　决策树　蒙特卡洛　增强精英保留遗传算法　贝叶斯推断**

一、 问题重述

1.1 问题背景

某企业生产某款畅销的电子产品。为此，需要分别采购两种零配件（零配件 1 和零配件 2）进行成品装配。在装配过程中，如果两个零配件均合格，成品也可能会出现不合格的情况；但只要其中一个零配件不合格，那么成品必然不符合标准。面对不合格的成品，公司有两种选择：可以将其报废，或者进行拆解。拆解过程不会对零配件造成损伤，但需要支付相应的拆解费用。

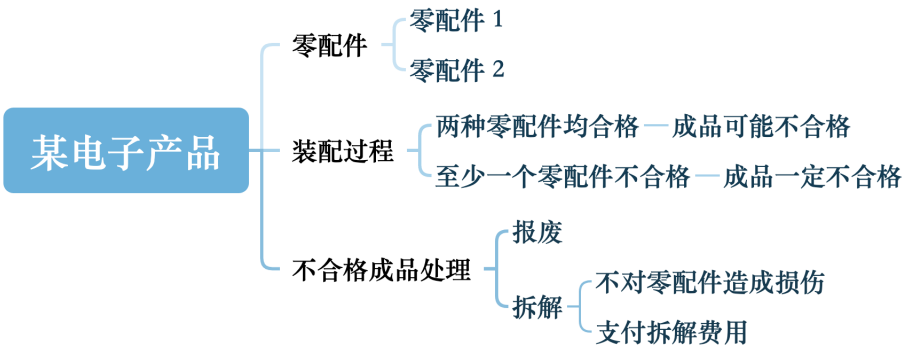


图 1　问题背景分析导图

1.2 问题要求

根据题目背景，需要建立数学模型解决以下问题：

- **问题 1**：为节约检测成本，要设计一种抽样次数尽可能小的检测方案，通过执行该检测方案判定是否接收供应商提供的这批零件。若零配件的理论标称值为 10%，对以下两种情形分别给出结果：在 95% 的信度下，如果次品率超过标称值，应拒绝接收；在 90% 的信度下，如果次品率不超过标称值，应接受该批次。
- **问题 2**：在不同的生产情况下，两种零配件和成品的次品率、购买成本和检测成本不同，不合格产品的拆解成本和调换成本也不同，根据不同的生产情况做出各个生产过程的决策（包括是否检测零配件和成品、是否拆解不合格成品），力图最大化利润率。
- **问题 3**：对于含 m 道工序和 n 个零配件的生产流程，已知各产品（含零配件、各层半成品、成品）的成本、检测费用、次品率等信息，推广问题 2 的分析

方法，制定最优化生产决策。特别地，对于 2 道工序和 8 个零配件的生产流程，已知各参数的具体值，确定各个生产情况下的最优生产决策。

- **问题 4**：各产品的次品率不再是确定的值，而是通过抽样检测获取的，在此条件下再对问题 2、问题 3 进行分析，获取新的最优生产决策。

二、 问题分析

2.1 问题 1 分析

问题 1 本质是一个假设检验在抽样检测中的应用问题，给定置信水平，都过对抽样方式与判断准则的合理设计，可以减少所需抽样次数，降低企业生产成本。设计一个序贯抽样检测方案，不预先设定抽样次数，而是随着抽样的进行动态更新，即可控制抽样检测次数尽可能少。

2.2 问题 2 分析

问题 2 本质是一个决策问题，难点在于其不是传统的树状决策问题，存在无限迭代下去的可能性，直接求解较为困难，故考虑计算期望值的极限，将可能无限迭代下去的问题转为迭代次数有限的问题。由于参数空间较小，可以考虑用模拟的方式求解。当认为问题 2 是一个不确定性决策的时候，可以采用乐观准则、悲观准则等多种决策准则完成对最终决策方案的挑选。

2.3 问题 3 分析

问题 3 与问题 2 本质属于同一类问题，问题 3 是将问题 2 的复杂度进行了横向和纵向的延伸，因此可以尝试使用与问题 2 同样的思想来解决问题 3，即将可能无限次迭代下去的图结构转化为树状结构，用决策树的思想解决问题。但不同于解空间较小的问题 2，问题 3 不再可以使用枚举的方式对不同解的优劣程度进行直观对比，而是需要应用启发式算法完成对最优解的寻找。

2.4 问题 4 分析

问题 4 中次品率由一个确定的数值转变为了通过抽样估计出的值，可以尝试求解次品率的分布，通过抽样的过程不断对其进行更新；此外，通过抽样得到的次品率必然会带来利润率方差的提升，可能需要权衡利润率的期望与方差

三、 模型的假设

1. **检测绝对准确**

   假设采用的所有质量检测手段均达到绝对准确的标准，即检测过程能无偏差地揭示零配件及成品的真实质量状况，有助于消除因检测误差而引入的不确定因素。

2. **市场需求充足**

   假设市场对产品的需求始终保持在高水平，即不存在市场需求不足或销售受阻的情况。这一假设意味着生产出的产品能够迅速且完全地被市场消化，从而在大样本容量的情境中，为模型模拟提供了一个稳定且可靠的基础。

3. **零配件质量相互独立**

   为简化分析框架，我们设定不同批次的零配件 1 与零配件 2 在质量表现上相互独立，即一种零配件的质量状况不对另一种构成直接影响。此设定旨在让评估聚焦于单个零配件的质量特性及其次品率，从而提升分析的精确性和效率。

四、 符号说明

表 1　问题 1 中符号说明

符号	意义
n	抽样次数
p_1	值较高的、可接收限度最低的合格率标准
p_0	值较低的、期望达到的合格标准
d_n	第 n 次序贯抽检后累计出现不合格品的个数
L_n	第 n 次抽样的似然比
A	拒收整批产品的阈值
B	接收整批产品的阈值
α	次品率合格但却判定不接收这批产品的概率
β	次品率超标但却判定可以接收这批产品的概率
H_1	次品率不合格，拒收整批产品
H_0	次品率合格，接收整批产品
$f(p)$	接收这批零配件的概率
ASN	平均所需抽样次数

表 2　问题 2 中符号说明

符号	意义
$d_1d_2d_3d_4$	蒙特卡洛模型中表示 16 种决策方案
p_i	零配件 $i(i=1,2)$ 的次品率
a_j	不确定型决策中表示 16 种决策方案
PR_{jq}	利润率
λ	折中系数 $(0<\lambda<1)$

表 3　问题 4 中符号说明

符号	意义
α	虚拟的成功次数
β	虚拟的失败次数
k	观察到的成功次数
n	试验次数
L	似然函数
R_p	投资组合的预期回报率
R_f	无风险回报率
σ_p	投资组合的回报率标准差

五、 模型的建立与求解

5.1 问题 1 模型的建立及求解

5.1.1 模型建立：序贯概率比检验（SPRT）

序贯概率比检验 (Sequential probability ratio test,SPRT) 是一种不预先确定抽样次数的统计假设检验方法。在这种方法中，抽样次数 n 是一个随机变量，其值取决于数据收集过程中得到的信息，而传统的假设检验通常设定一个固定的抽样次数 N。序贯概率比检验的目标是在尽可能小的抽样次数下快速做出决策，以提高检验的效率。

(1) 似然比检验

在运用序贯抽样检验方法时，其核心策略是逐次从待检批次中选取单一单位产品进行质量评估。此过程依据已累积的检验结果，即不合格品数量 d_i，动态地分析并计算两种假设下的概率：一是假设不合格品率为 p_1（通常代表一个值较高的、可接收限度最低的合格率标准）时，出现 d_i 个不合格品的概率；二是假设不合格品率为 p_0（通常代表一个值较低的、期望达到的合格标准）时，同样条件下出现 d_i 个不合格品的概率。[1]

通过比较这两个概率值，检验流程得以推进：

1. **优质批判断**：若基于 p_0 的概率显著大于基于 p_1 的概率，这表明实际不合格品率更接近于 p_0，即该批次很可能是优质批，因此应判定该批次为合格。
2. **劣质批判断**：若基于 p_1 的概率显著超出基于 p_0 的概率，这表明实际不合格品率可能远高于 p_0，指向该批次为劣质批，故应判定该批次为不合格。
3. **持续检验**：在两种概率相差无几，无法明确判定批次质量优劣的情境下，检验过程不会立即结束。此时，将继续从批次中抽取下一个单位产品进行检验，并更新 d_i 的值，重复上述的概率比较步骤，直至累积的数据足以支持做出明确的合格或不合格判定。序贯概率比抽样经过有限次抽样后终止的概率是 1[2]，因此不需要考虑无限抽样的可能性。

似然比 (Likelihood Ratio) L_n 是构建序贯概率比检验的核心统计量，用于在每次抽样之后连续更新当前观测数据的信息以评估假设。在本题中似然比定义为：

$$L_n = \frac{C_n^{d_n} p_1^{d_n} (1-p_1)^{n-d_n}}{C_n^{d_n} p_0^{d_n} (1-p_0)^{n-d_n}} = \frac{p_1^{d_n} (1-p_1)^{n-d_n}}{p_0^{d_n} (1-p_0)^{n-d_n}} \tag{1}$$

设 A 为拒收整批产品的阈值，B 为接收整批产品的阈值。若 $L_n \leq B$，判批合格而接收；若 $L_n \geq A$，判批不合格而拒收；若 $B < L_n < A$，则继续抽取一个单位产品进行检验。

在进行序贯概率比检验时，关键的操作是持续更新数据集并重新计算似然比 L_n，以确定接下来的操作。每次新增数据后，立即重新计算似然比，使得决策过程非常依赖于每次抽样的结果。令 H_0 为原假设（即次品率合格，接收整批产品），H_1 为备择假设（即次品率不合格，拒收整批产品），如果似然比指向 H_1 的概率显著高于 H_0，可能很快就会停止数据收集，反之亦然。这种实时更新和决策的过程大大减少了总体平均抽样数，因为在许多情况下，通过少量本就足以达到决策的置信度。

（2）决策规则和阈值

决策规则围绕设定的两个阈值 A 和 B（通常 $A > 1 > B > 0$）展开，α 表示次品率合格但却判定不接收这批产品的概率，β 表示次品率超标但却判定可以接收这批产品的概率。似然比的取值在 $(-\infty, B]$、(B, A)、$[A, +\infty)$ 三个不同区间内会得到不同的运行决策，但最终必然归于 $(-\infty, B]$ 或 $[A, +\infty)$ 两个区间内，即接收或拒收。

那么，在实际次品率达标的情况下：

- P_{12} 表示做出拒收判断的概率，已知 $P_{12} = \alpha$
- P_{11} 表示做出接收判断的概率，得 $P_{11} = 1 - \alpha$

在实际次品率不达标的情况下：

- P_{21} 表示做出接收判断的概率，已知 $P_{21} = \beta$
- P_{22} 表示做出拒收判断的概率，得 $P_{22} = 1 - \beta$

根据上文的似然比判断法：

$$\frac{P_{22}}{P_{12}} \geqslant A \tag{2}$$

$$\frac{P_{21}}{P_{11}} \leqslant B \tag{3}$$

可以推导出：

$$L_1 : \beta \leqslant 1 - \alpha A \tag{4}$$

$$L_2 : \beta \leqslant (1 - \alpha)B \tag{5}$$

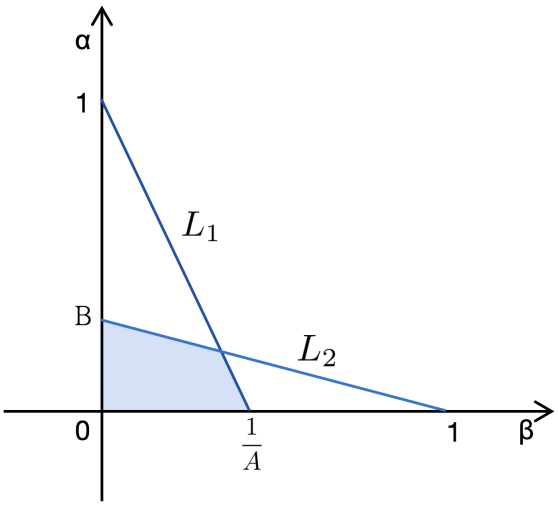


图 2　可行域图示

图2中阴影所示的部分即为满足约束条件的可行域。

由于 A 是上阀值，取其可以取到的最大值作为最终的判断标准可以确保 A 能满足每一次判定需要，B 与之同理。其中，因 A 取最大值、B 取最小值所造成的 α 和 β 的误差可以小到忽略不计 [2]，最终选取：

$$A = \frac{1-\beta}{\alpha} \tag{6}$$

$$B = \frac{\beta}{1-\alpha} \tag{7}$$

（3）序贯抽样方案的图示法

由前可知，继续抽样判定规则为：

$$B < \frac{p_1^{d_n}(1-p_1)^{n-d_n}}{p_0^{d_n}(1-p_0)^{n-d_n}} < A \tag{8}$$

令

$$s = \frac{ln\frac{1-p_0}{1-p_1}}{ln\frac{p_1}{p_0} - ln\frac{1-p_1}{1-p_0}} \tag{9}$$

$$h_1 = \frac{lnA}{ln\frac{p_1}{p_0} - ln\frac{1-p_1}{1-p_0}} \tag{10}$$

$$h_2 = \frac{lnB}{ln\frac{p_1}{p_0} - ln\frac{1-p_1}{1-p_0}} \tag{11}$$

为简化计算，将不等式全部取对数后计算得到的继续抽查域为：

$$h_1 + n \cdot s < d_n < h_2 + n \cdot s \tag{12}$$

设定一个坐标系，其中横轴代表累计的抽检产品数，记作 n，纵轴对应这些产品中累计发现的不合格品数量，标记为 d_n。基于这一设定绘制两条直线：$d_n = h_1 + n \cdot s$ 与 $d_n = h_2 + n \cdot s$，这两条直线共同构成了序贯抽样方案的核心图形框架，如图3所示。

在逐一检查产品并累积至第 n 个时，我们根据当前的 (n, dn) 坐标点位置来判断整批产品的接受状态。具体而言：

- 若该坐标点位于预设的“接收域”内，则表明当前抽检结果符合既定的质量标准，因此可以作出接收整批产品的决定。
- 若坐标点落入了“拒收域”，则意味着不合格品数量已超出接收范围，必须拒收整批产品。
- 若坐标点恰好位于“继抽域”，即上述两区域之外的区域，则表明当前信息不足以作出明确判断，需要继续抽取第 $n+1$ 个产品进行检测，并重复此过程，直至能够明确地将整批产品归类为接收或拒收。

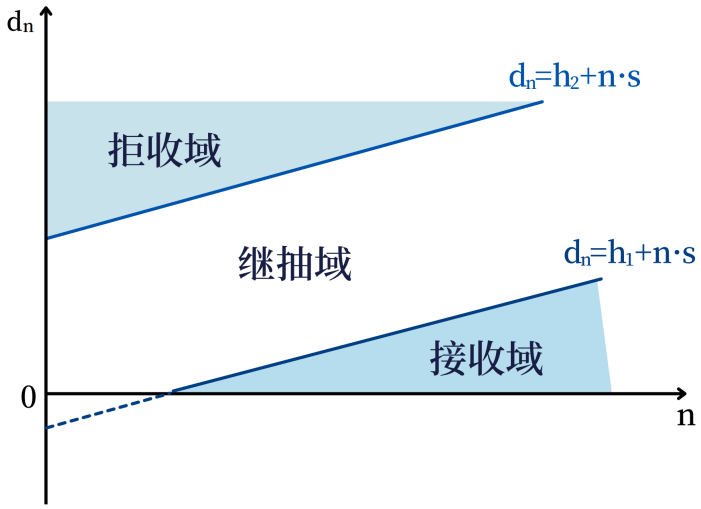


图 3　序贯抽样方案图示

5.1.2 模型求解

(1) 在 95% 的信度下认定这批零配件次品率合格则接收，此时的 α 值为 0.05

p0 等于供应商声称的标称值 10%。这批零配件的实际次品率 p 是一个在 $[0,1]$ 区间上的随机数。在不同的 p 值下，通过对序贯抽样操作进行蒙特卡洛模拟，可以得到接收这批零配件的概率 $f(p)$ 和平均所需抽样次数 ASN。

改变 β 值与 p_0 值，可得不同情况下的 $f(p)$ 与 ASN。

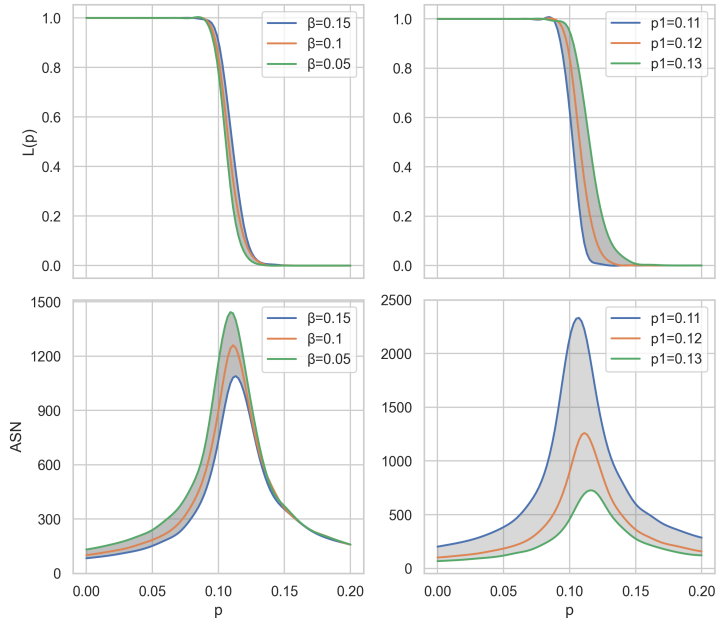


图 4　不同 β 值与 p_1 值下的 $f(p)$、ASN 关于实际次品率 p 的关系曲线
(在实际作图过程中使用了三次样条插值法对曲线进行平滑处理)

据图4可知，当实际次品率高于 10% 的时候，β 值越小则判断效果越好。与此同时，适当放宽 β 的限制可以使得所需抽检次数变小。为兼顾检测效果与抽样次数，取 β 为 0.1。

同理，p_1 的值越大则检测效果越好，但所需检测次数也显著增加，无法通过调节 p_1 的值来同时获得判断精确度的增加与检测次数的减少。为满足尽量减少检测次数的需求，p_1 值不宜设置得过大。据图5所示，当 p_1 值大于 0.12 时，随着 p_1 的增大，所需平均检测次数减少幅度变得更加明显。因此，在兼顾检测效果与检测次数的情况下，可以设定 p_1 为 0.12。

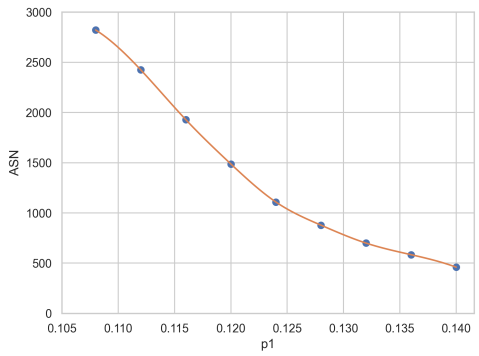


图 5　ASN 关于 p_1 的关系曲线

在 β 值为 0.1、p_1 值为 0.12 的情况下，即图4中的橙色折线，序贯抽样所需的平均抽检次数在 1500 次以下。

(2) 在 90% 的信度下认定这批零件次品率不合格则拒收，此时的 β 值为 0.1，p_1'

为 10%

与（1）同理，可以绘制不同的 α 值与 p_0' 值下的 $f(p)$ 和 ASN 函数图像。

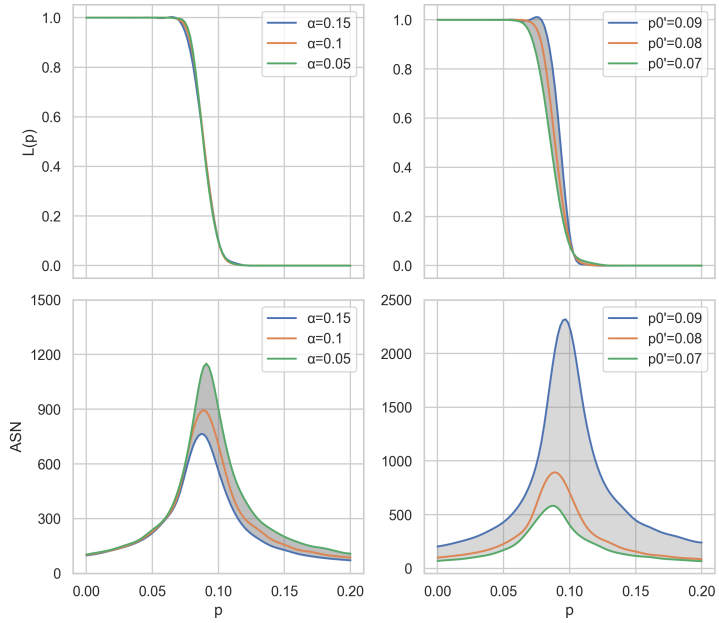


图 6　不同 α 值与 p_0' 值下的 $f(p)$、ASN 关于实际次品率 p 的关系曲线
(在实际作图过程中使用了三次样条插值法对曲线进行平滑处理)

对参数 α 进行模型的灵敏度分析，可得在一定范围内 α 的变动并不会引起判断准确性的显著差异，即此情况下序贯抽样检测法的稳定性较强。

p_0' 的值越大则检测效果越好，同时所需检测次数越多，如图7所示。在兼顾检测效果与检测次数的情况下，可以设定 p_0' 为 0.08。

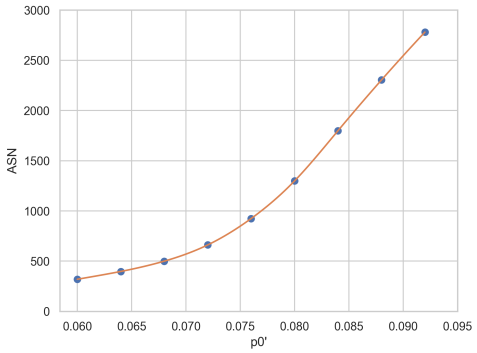


图 7　ASN 关于 p_0' 的关系曲线

在 p_0' 的值为 0.08 的情况下，即图6中的橙色折线，序贯抽样所需的平均检测次数在 1000 次以下。

在实际应用过程中，由于序贯抽样法不提前设定抽取次数，而是根据已有抽取情况对判断做出实时更新，所以有做出最终判断所需更新次数极多的可能，

这就违背了序贯抽样法减少抽样次数的初衷。因此，我们应当规定一个阈值，当抽样次数达到该阈值却仍未做出最终判断时，停止抽取新样本，根据已抽取的样本情况进行基础的假设检验进而做出最终判断。

根据（1）（2）两问中模拟出的 ASN 曲线，选取 1500 为终止序贯抽样的阈值较为合适。当抽样次数达到阈值后，决策过程如下所示：

设 $X_1, X_2, \cdots, X_{1500}$ 是抽样得到的来自总体 $B(1,p)$ 的随机样本，统计量 $T = \sum_{i=1}^{1500} X_i \sim B(1500, p)$。

由于 $n = 1500$ 足够大，由中心极限定理：

$$T \sim N(1500p, 1500p(1-p)) \tag{13}$$

记 $U = \frac{T - 1500p_0}{\sqrt{1500p_0(1-p_0)}}$，其中 $p_0 = 0.1$。

- 对（1），$H_0 : p \leqslant p_0$ v.s. $H_1 : p > p_0$

  当 $U \geqslant u_{o.95} = 1.64$ 时拒绝原假设，拒收这批零配件

- 对（2），$H_0 : p \geqslant p_0$ v.s. $H_1 : p < p_0$

  当 $U \leqslant u_{o.1} = -1.28$ 时拒绝原假设，接收这批零配件

5.2 问题 2 模型的建立及求解

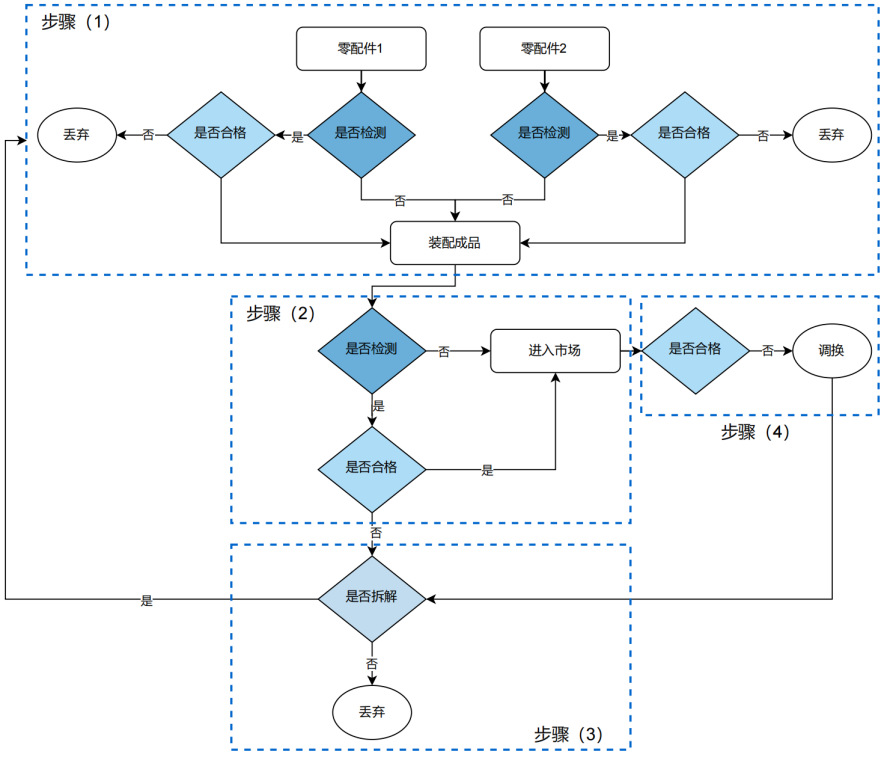


图 8　问题 2 流程图

问题 2 的符号预设:

- p_1, p_2, p_3 分别表示零配件 1、2 和成品的次品率
- c_1, c_2, c_3 分别表示零配件 1、2 购买成本和成品的装配成本
- t_1, t_2, t_3 分别表示零配件 1、2 和成品的检测成本
- r 表示成品的市场售价
- e 表示不合格成品的调换损失
- d 表示不合格成品的拆解费用
- R 表示零配件都合格的不合格成品的期望收益

5.2.1 问题简化

- **零配件检测的决策模型**：如果第一次检测了某零配件，那么检测完毕后该零配件不会再出现次品，后续不再检测该零配件；但如果第一次不检测，则下一次被拆解返回时，必须检测该零配件，同时后续不再检测该零配件。
  必须检测的理由如下：若始终不检测该零配件，每次被拆解返回的零配件次品率 p_1, p_2 将满足以下公式:

$$p_1 = \frac{p_{10} + p_{11}}{p_{00} + p_{01} + p_{10} + p_{11}} \tag{14}$$

$$p_2 = \frac{p_{01} + p_{11}}{p_{00} + p_{01} + p_{10} + p_{11}} \tag{15}$$

$$\begin{cases} p_{00} = p_3 \cdot (1 - p_1) \cdot (1 - p_2) \\ p_{01} = (1 - p_1) \cdot p_2 \\ p_{10} = p_1 \cdot (1 - p_2) \\ p_{11} = p_1 \cdot p_2 \end{cases} \tag{16}$$

  注：$p_{x_1x_2}$ 表示某种类型次品出现的概率，x_1x_2 分别表示零配件 1、2 是否合格，0 表示合格，1 表示不合格，即 p_{00} 表示零配件 1、2 都合格的次品出现的概率。
  根据以上公式计算得出当零配件 1 始终不检测时，经过四次迭代其自身次品率以及由其合成的成品次品率的变化趋势图9。可以发现，无论零件 2 是否检测，第一次迭代后成品次品率都将达到 60%。因此，为避免过高的次品率带来的高昂成本和零配件陷入死循环的情况发生，我们规定对于任意零配件，若第一次不检测，则下一次被拆解返回时，必须检测。

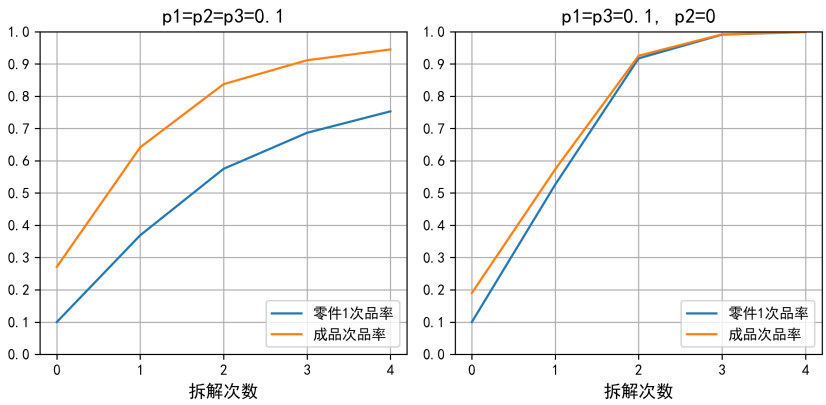


图 9　次品率变化趋势图

- **零配件丢弃的决策模型**：经过零件检测和对次品零件的丢弃后，取零配件 1 剩余数量和零配件 2 剩余数量中的较小值，丢弃多余的零配件以保证两个零件都能两两配对，简化问题。

5.2.2 决策树模型

决策树是用于分析决策路径的一种图形结构，通过树状结构来描述决策过程，其中不同的节点代表不同的含义。

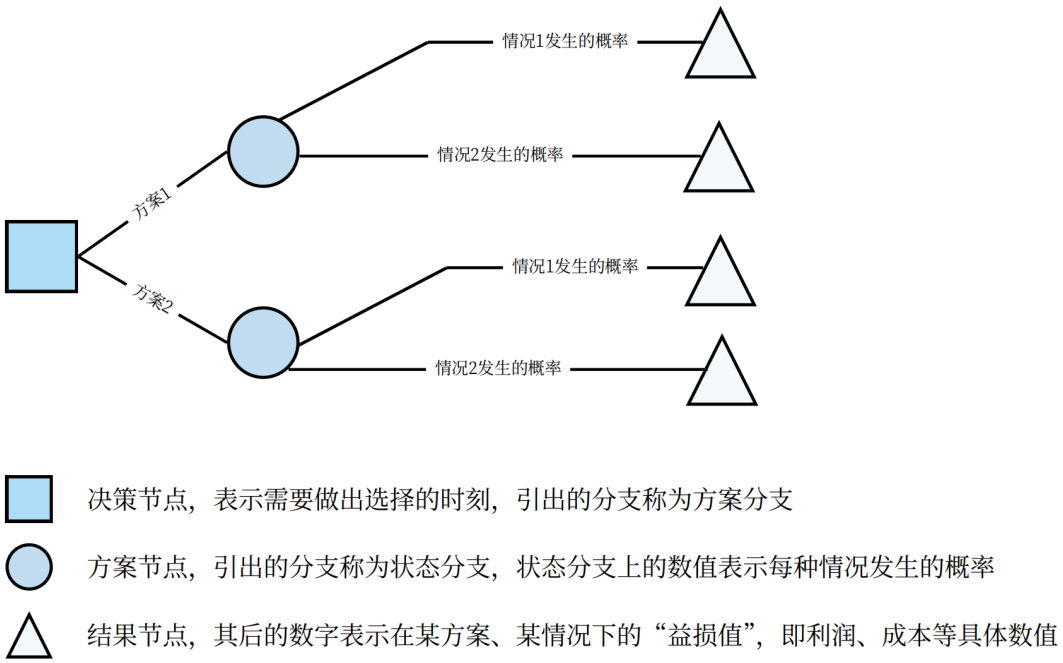


图 10　决策树说明图

在完成对决策树的绘制后，从结果节点开始由右向左逆序推导，通过对不同情况发生的概率及其对应的益损值进行计算即可获得不同方案的数学期望值。

由先前检测零件的策略可知，第一次迭代之后所有零件均已查验过一次，即仅有两个零件都合格的不合格成品在不断迭代，因此只要能求解出此类成品的期望收益，即可消除迭代的影响，将此问题转化为真正的树状结构，应用决策树模型进行求解。

考察该成品的小“决策树”，如图11所示：

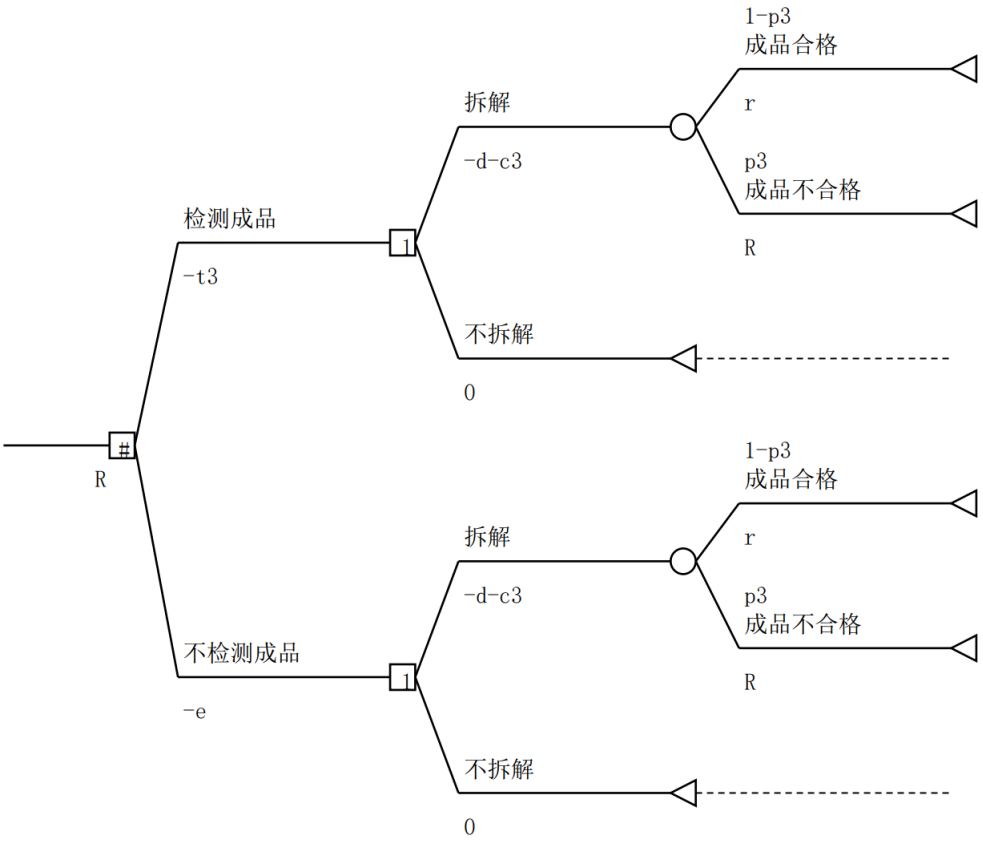


图 11　零件合格的不合格成品的小“决策树”

假设 R 为其期望收益（d_3d_4 不同，R 不同），当 $d_3=1$，$d_4=1$ 或 $d_3=0$，$d_4=1$ 时可列出以下方程，求解 R 即可得到以上两种情况下的期望收益，而另两种情况无需列方程求解。

$$R=-t_3-d-c_3+(1-p_3)\cdot r+p_3\cdot R \tag{17}$$

$$R=-e-d-c_3+(1-p_3)\cdot r+p_3\cdot R \tag{18}$$

综上，d_3d_4 取不同值时两个零件都合格的不合格成品的期望收益如下所示：

$$R=\begin{cases} r-\frac{t_3+d+c_3}{1-p_3} & d_3=1,d_4=1 \\ r-\frac{e+d+c_3}{1-p_3} & d_3=0,d_4=1 \\ -t_3 & d_3=1,d_4=0 \\ -e & d_3=0,d_4=0 \end{cases} \tag{19}$$

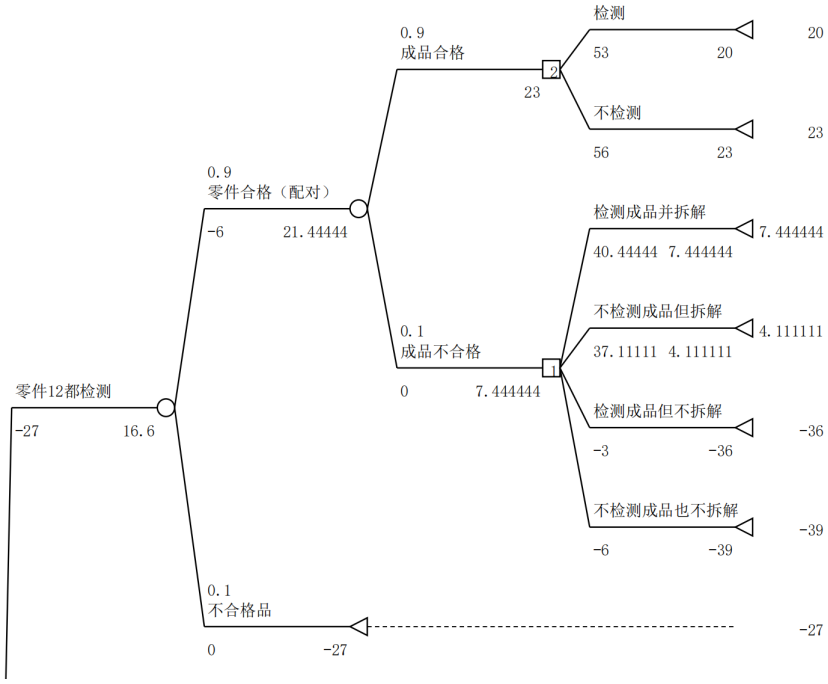


图 12　消除迭代影响后的决策树（仅展示部分，详见支撑材料）

注：以上决策树按事件发生顺序绘制，旨在清晰展示决策流程。但实际求解过程中，考虑到各个子决策之间存在相互影响（如若选择检测成品，那么在任何子决策中都应检测成品，而不是分开进行决策），因此需要将 d_3d_4 的决策阶段前置，再进行求解，该决策树未在文中画出。

5.2.3 蒙特卡洛模型

除了使用基于计算期望的决策树模型，对于复杂问题也可以使用随机模拟的方式求解，并与决策树模型的结果相互验证。

蒙特卡洛模型（Monte Carlo Model）是一种基于概率和统计理论方法的随机模拟方法。用蒙特卡洛模型解决复杂问题时，需要构建概率模型或随机过程，利用随机数进行多次模拟试验，计算关键参数的统计特征，以这些统计量的估计值作为问题的近似解。

（1）模型初始化

- **二进制表示选择方案**：16 种不同的检测方案用 4 位二进制数 $d_1d_2d_3d_4$ 表示 (0000 到 1111)，其中每一位代表一个特定的决策点（1 为检测，0 为不检测），

从左至右分别为“是否检测零配件 1”“是否检测零配件 2”“是否检测成品”和“是否拆解”。例如，$d_1d_2d_3d_4 = 1110$ ，表示在 4 个检测点中，检测零配件 1、零配件 2 和成品，不拆解。

- **生成零配件质量数据**：利用 Python 随机生成零件 1 和零件 2 的质量状况。记零配件 1 和零配件 2 的质量状况分别为 X_1 和 X_2，p_i 是零配件 $i(i = 1, 2)$ 的次品率。令 $x = 1$ 表示次品，$x = 0$ 表示合格品。

$$p(X_i = x) = \begin{cases} p_i & (x = 1) \\ 1 - p_i & (x = 0) \end{cases}$$

  X_i 服从二项分布，即 $X_i \backsim B(1, p_i)$。
- **进货量初始化**：设两种零配件的进货量原始值均为 10000。

(2) 优化策略

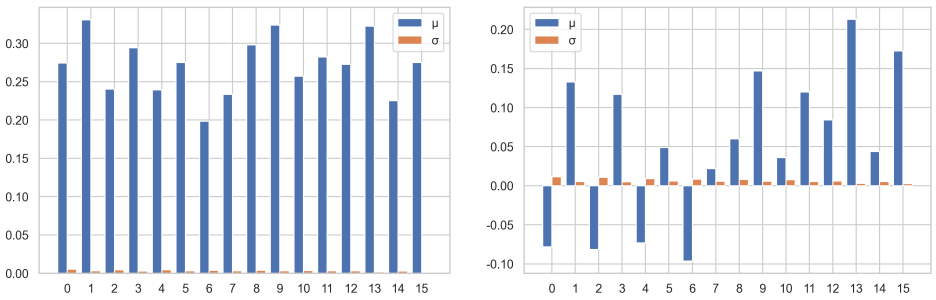
考虑到简化策略中的零配件丢弃模型可能会浪费一部分合格零件，我们尝试在其基础上进行优化，尽可能少地丢弃正品零件。

根据零件检测模型，从第二轮开始，拆解返回的零件都为正品且已两两配对，因此我们只对购买零件时的规则进行优化。规则如下：设进货量基准值为 10000，若方案选择不检测零配件 $i(i = 1, 2)$，则零配件 i 的的进货量取基准值 10000；若方案选择检测零配件 i，则零配件 i 的进货量设为 10000 ÷ (1−p_i)，p_i 为零配件 i 的次品率。这样的策略保证了初始丢弃的零件几乎都是次品，减少浪费。

经验证，是否选用该策略对后续六种情况下的最优决策及该决策下的利润率无显著影响，但能够使得 16 种可能决策的平均利润率上升约 2%。

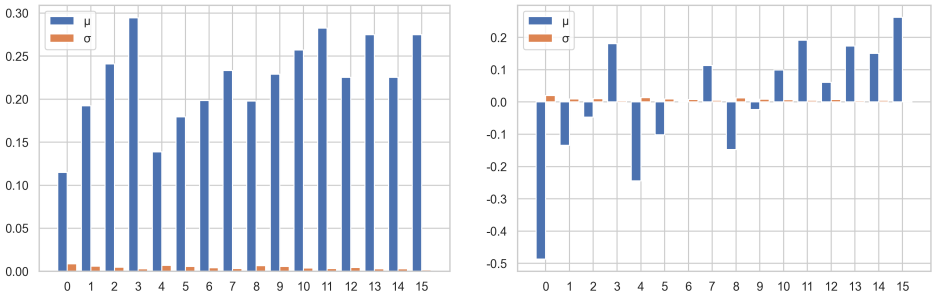
(3) 模型结论

考虑到问题的复杂性，我们直接选取利润率（利润率 =（收入 - 成本）/ 收入）作为选取决策的判断依据。由附录B可知，蒙特卡洛模拟得到的利润率分布呈近似正态，因此可以使用其均值来衡量期望利润率、标准差来衡量其波动程度。



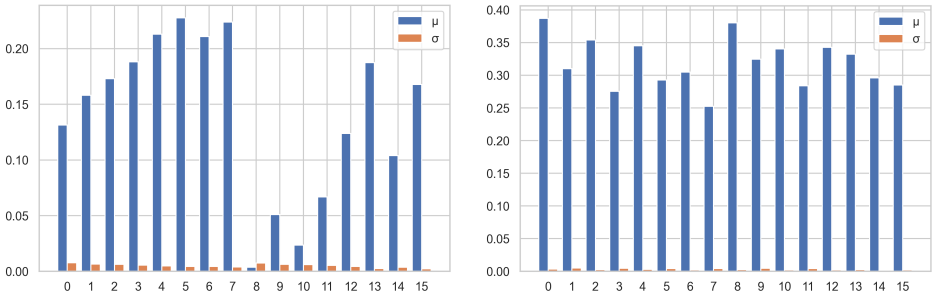
(a) 情况 1

(b) 情况 2



(c) 情况 3

(d) 情况 4



(e) 情况 5

(f) 情况 6

图 13　6 种情况下不同决策方案的利润率的平均值和标准差

由图13可知，以上六种情况中各决策的标准差极小，因此忽略标准差，仅通过模拟利润率的均值作为唯一评价标准，最终得到如下结论，与决策树模型得到的结论一致，彼此印证。

表 4　问题 2 用蒙特卡洛模型得到的结论

情景	零配件 1 检测	零配件 2 检测	成品检测	不合格成品拆解	利润率
1	否	否	否	是	33.08%
2	是	是	否	是	21.29%
3	否	否	是	是	29.44%
4	是	是	是	是	26.20%
5	否	是	否	是	22.78%
6	否	否	否	否	38.72%

5.2.4 不确定型决策

在实际生产过程中，决策者可能无法预知自己将面临的情况，在生产开始前只能做出一种决策，因此这一决策要兼顾不同种情况。由于不知道六种情况发生的概率，所以该决策过程属于**不确定型决策** [3]，其中常用的决策准则有：乐观准则、悲观准则、折中准则、等可能准则、最小后悔值准则。

记 16 种决策方案为 $a_j(j=1,2,\cdots,16)$，分别对应用二进制表示的 0000 方案至 1111 方案。

根据上文的计算方法可得不同决策方案与不同情况下的利润率如表5所示，其中利润率保留三位小数。

表 5　各情况各决策利润率

	情况 1	情况 2	情况 3	情况 4	情况 5	情况 6
a_1	0.274	-0.078	0.115	-0.487	0.131	0.387
a_2	0.331	0.133	0.192	-0.135	0.158	0.31
a_3	0.24	-0.081	0.241	-0.047	0.173	0.354
a_4	0.294	0.117	0.294	0.18	0.188	0.276
a_5	0.239	-0.073	0.139	-0.244	0.213	0.345
a_6	0.275	0.049	0.179	-0.102	0.228	0.293
a_7	0.198	-0.097	0.198	0.001	0.211	0.305
a_8	0.233	0.022	0.233	0.113	0.224	0.252
a_9	0.298	0.06	0.198	-0.147	0.004	0.38
a_{10}	0.324	0.147	0.229	-0.023	0.051	0.325
a_{11}	0.257	0.036	0.257	0.099	0.023	0.34
a_{12}	0.282	0.12	0.282	0.191	0.067	0.284
a_{13}	0.273	0.084	0.225	0.061	0.124	0.343
a_{14}	0.322	0.213	0.275	0.173	0.188	0.332
a_{15}	0.225	0.044	0.225	0.15	0.104	0.296
a_{16}	0.275	0.173	0.275	0.262	0.168	0.285

记第 j 个方案的在情况 q 下 $(q=1,2,3,4,5,6)$ 的利润率为 PR_{jq}(Profit Rate)。

(1) 乐观准则

始终保持最乐观的态度，使用每种方案在最有利的情况下的利润率进行决策，并选择最优利润率最高的方案作为最终选择。

$$PR^* = \max_j \left(\max_q PR_{jq}\right) \tag{20}$$

不同方案下的最大利润率如下表6所示：

表 6　不同方案的最大利润率

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
0.387	0.331	0.354	0.294	0.345	0.293	0.305	0.252
a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
0.38	0.325	0.34	0.284	0.343	0.332	0.296	0.285

其中的最大值为 a_1 方案下的 0.387，即乐观准则下选择方案 a_1。

（2）悲观准则

在决策过程中持悲观态度，使用每种方案在最不利的情况下的利润率进行决策，并选择最差利润率最高的方案作为最终选择。

$$PR^* = \max_j (\min_q PR_{jq}) \tag{21}$$

与乐观准则同理，选择方案 a_{14}。

（3）折中准则

既不过分乐观，也不过分悲观，设定一个折中系数 $\lambda(0 < \lambda < 1)$。

$$PR^* = \max_j \left\{ \lambda \max_q PR_{jq} + (1-\lambda) \min_q PR_{jq} \right\} \tag{22}$$

不同折中系数的情况下对方案的选择情况如下表7所示:

表 7　不同折中系数下方案的选择

折中系数	选择的方案	折中系数	选择的方案	折中系数	选择的方案
0.00	a_{14}	0.35	a_{14}	0.70	a_{14}
0.05	a_{14}	0.40	a_{14}	0.75	a_{14}
0.10	a_{14}	0.45	a_{14}	0.80	a_{14}
0.15	a_{14}	0.50	a_{14}	0.85	a_{14}
0.20	a_{14}	0.55	a_{14}	0.90	a_9
0.25	a_{14}	0.60	a_{14}	0.95	a_9
0.30	a_{14}	0.65	a_{14}	1.00	a_1

据表可知，只有在相当乐观的情况下才会选择除 a_{14} 外的其他方案（如 a_9），大部分情况下选择 a_{14}。

（4）等可能准则

认为各种情况发生的概率相等，在此基础上计算不同方案的利润率并进行比较。

$$PR^* = \max_j \left(\frac{1}{6}\sum_{q=1}^{6} PR_{jq}\right) \tag{23}$$

通过 MATLAB 计算可知，在等可能准则下选择 a_{14}。

（5）最小后悔值准则

后悔值是每种情况下最高的利润率与其他利润率之差，选取最大后悔值最小的方案，可使选定方案并实施后产生最少的后悔。

$$PR^* = \min_q \left\{ \max_j \left(\max_q PR_{jq} - PR_{jq} \right) \right\} \tag{24}$$

通过 MATLAB 计算，可知此时仍然选择 a_{14}。

总结：

综上所述，**除了乐观准则外，在其他决策准则下 a_{14} 都是最优选择，其二进制表示方法为 1101，即两个零配件均检测、成品不检测、不合格的成品拆解。**

在 a_{14} 的决策下，将确保进入装配环节的零配件 1、零配件 2 全为合格产品，从源头上控制住了外界风险，不让其过多地进入生产系统中，成品的次品率也将大大降低；考虑是否检测成品本质上是比较成品检测成本与成品次品率 × 调换损失孰低，当拥有较低的成品次品率时，在大多数情况下后者更小，因此不需要再对成品进行检测，以降低成本；由于可以确保所有被客户退回的成品中的零配件都是合格的，所以要对退回产品进行拆解，所有拆解后的零件都可以直接被二次利用，不会造成二次检测成本。整个生产过程的无效成本浪费较少，因此 a_{14} 方案下大部分情况的利润率较高。

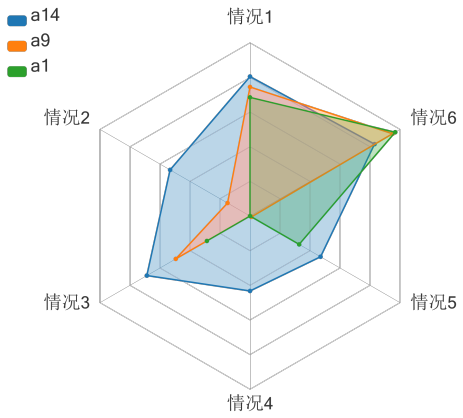


图 14　不同方案下整体利润率对比图

5.3 问题 3 模型的建立及求解

5.3.1 m 道工序、n 个零配件情况

当问题扩展到 m 道工序、n 个零配件时，回顾问题 2，能够将一个迭代问题转化为树状结构进行求解的关键在于求出了在概率上可能无限迭代的产品（问题 2 中是两个零配件都合格的次成品）确定的期望收益，从而可以直接使用这个期望收益而回避了迭代问题；另一方面，问题二的零配件检测模型确保了两次迭代后零配件全部合格，在概率上可能无限迭代的产品唯一，极大简化了求解过程。

在 $m \times n$ 的问题中，参考问题 2 的思路，要求所有产品的拆解都直接拆到零件，同样规定若初始时不检测零件，则当第一次零件返回时必检测，保证在第二次迭代后还存在的零件全部合格，此时可以保证第一道工序中需要求解期望收益的产品为所有零件都合格的不合格半成品。但仿照问题 2 画出“决策树”，发现要想求解期望，还需要这道工序中合格半成品的期望以及下道工序中所有零件都合格的不合格半成品期望。基于该需求，我们决定扩展问题 2 的零配件检测的决策模型，即当零件全部完成检测后，后续所有产品的拆解将拆解到第一层半成品，以此类推，直到经过有限次迭代后，最后一层半成品全部完成检测，此时回到第二问的情况，只需要求出不合格成品的期望收益与最后一层合格半成品的期望收益。这两个量确定后，可以进一步求出最后一层不合格半成品的和倒数第二层合格半成品的期望收益，以此类推直到求出第一层不合格半成品的收益。至此所有在概率上可能无限迭代的产品的期望收益均已求出，问题退化为树状结构，可以直接求解。

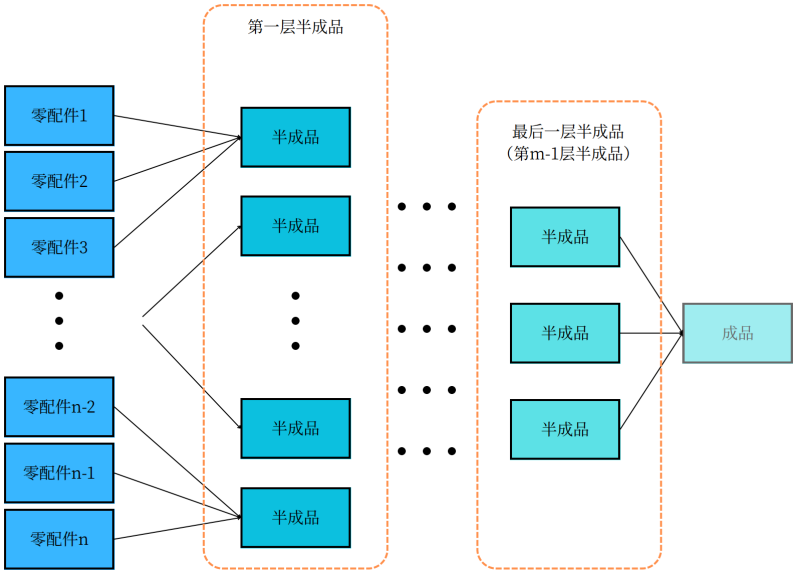


图 15　m 道工序、n 个零配件的组装情况

虽然基于此种方式可以较为准确地求解出各种决策的理论期望值，但整体过程依然过于繁琐，对于 m，n 较大的问题，可以参照以上的拆解策略进行随机模拟并配合启发式搜索算法寻找满意解。

5.3.2 2 道工序、8 个零配件情况

(1) 问题特性

- 问题三的参数空间是 2^{16}，因此这是一个参数空间较大的组合优化问题。
- 通过在参数空间中随机采样，得到图16，由图可知利润率分布范围较广。

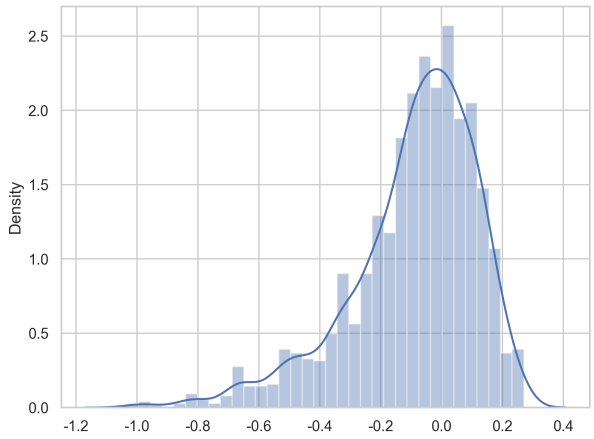


图 16　随机决策利润率分布图

因此在较大空间且目标函数取值范围较大的情况下，搜索难度较大，需要对传统遗传算法模型进行优化，以加速模型收敛。

(2) 目标函数

结合蒙特卡洛模型模拟出的平均利润率作为优化模型的目标值。

(3) 决策变量

BG 编码 (Binary-Gray Encoding) 指的是二进制/格雷码编码，是染色体的一种种基础编码方式，常用于遗传算法。这种编码结合了二进制和格雷码的特点，旨在减少相邻解之间的汉明距离，从而可能改善遗传算法的搜索性能。

(4) 模型优化

- **引入先验信息**：观察问题三的参数，发现检测成本偏低，通过蒙特卡洛模拟发现当所有决策变量均取 1 时即能取得 26.5% 的利润率，较随机初始化的利润率有显著提升，因而并未选择随机初始化种群，而是将所有变量均为 1 作为初始种群，加速算法收敛速度。
- **引入增强精英保留 (SE)**：**增强精英保留遗传算法 (Set-based Elite Genetic Algorithm,SEGA)** 是一种基于自然选择和遗传机制的启发式搜索算法，它旨在

通过保留一部分最优个体（精英）来提高算法的收敛速度和解的质量。这种算法通过维护一个精英集合，确保这些高适应度个体能够被遗传到下一代，从而避免它们被算法的随机性所淘汰，以期找到问题的最优或近似最优解。这些策略不仅确保了高质量解的延续，还通过增加多样性、灵活调整操作参数和引入并行计算能力，有效避免算法早熟，使 SEGA 在复杂多峰值优化问题上展现出优于传统遗传算法的性能。

(5) 模型求解

运用 Python 的 Geatpy 库进行增强精英保留遗传算法的求解，其中规定若循环十次后仍未得到更优解，则跳出局部最优解，以防止算法陷入局部最优。最后得出决策：**零件 1-8 都不检测，半成品 1-3 都检测并拆解，成品不检测但拆解，期望利润率：26.90%**。

5.4 问题 4 模型的建立及求解

5.4.1 概率模型：贝叶斯推断

贝叶斯推断通过结合先验信息和新的观测数据，利用贝叶斯公式来更新对未知参数的信念，形成后验分布。这种方法不仅考虑了数据本身的影响，而且整合了先验知识，使得整个推断过程更为全面和客观。

(1) 先验分布选择

对于二项分布，常选用 Beta 分布作为先验，因为它是一个二项分布的共轭先验，这意味着似然函数与先验分布属于同一族，使得后验分布保持相同的形式，因此在本题中选择 Beta 分布作为先验。

(2) 计算似然函数

似然函数是在已知参数 p 的条件下，得到观测数据的概率性描述。对于二项分布，似然函数为：

$$L(p \mid k, n) = p^k (1-p)^{n-k} \tag{25}$$

其中 k 是观察到的成功次数，n 是试验次数。

(3) 更新后验分布

后验分布是根据贝叶斯定理将先验分布与似然函数结合，更新而得。对于 Beta 先验和二项似然，后验分布同样是 Beta 分布，参数更新为 $\alpha+k$ 和 $\beta+(n-k)$，即 Beta$(\alpha+k, \beta+n-k)$。

在本题中，可以根据历史次品率分布设置先验，并根据本次抽样结果动态调整，形成次品率的后验分布。之后使用该后验分布随机生成次品率值，重新完成问题 2 和问题 3 的蒙特卡洛模拟。具体来说，由于在本题中未给出足够的先验

信息，先验分布设置较为随机，我们设定先验分布中 α=1, β=9，使得均值为 0.1。为便于与第二三问比较，我们假设抽样 1000 次，并使得后验分布的均值接近于问题 2 和问题 3 中的次品率。

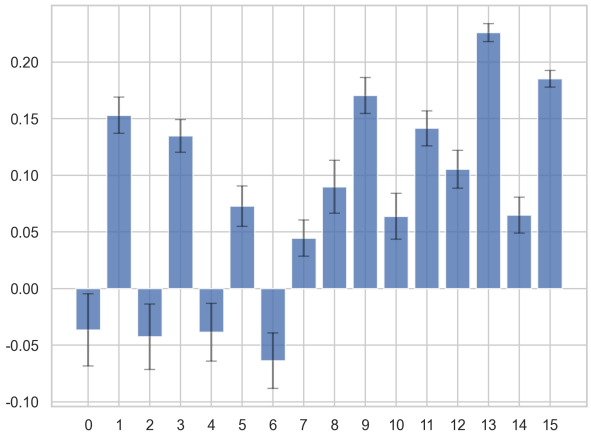


图 17　情况 2 利润率

问题 2 重新模拟所得的利润率如上图17所示，误差线范围为均值 ± 三倍标准差，可以发现标准差显著高于问题 2，这也就意味着我们需要对利润率的期望和波动情况进行权衡。

5.4.2 评价模型：夏普比率

本题最终方案的选择由利润率和利润率的波动情况（即标准差）共同决定，在一定的利润率波动情况下，利润率越高则方案越好。投资学中的**夏普比率**可以综合反映回报与风险的关系，可以利用夏普比率的思想完成本题的评价。

(1) 夏普比率介绍

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p} \tag{26}$$

其中：

- R_p 代表投资组合的预期回报率
- R_f 代表无风险回报率
- σ_p 是投资组合的回报率标准差，表示投资风险

高夏普比率意味着投资组合在每单位风险上获取了较高的超额回报，即投资效率较高。

在本题中：

- R_p 对应预期利润率 μ

- R_f 对应同一情况下不同方案中的最低利润率 μ_{min}，当最低利润率是负值时，取 μ_{min} 为 0
- σ_p 对应利润率的波动情况 σ

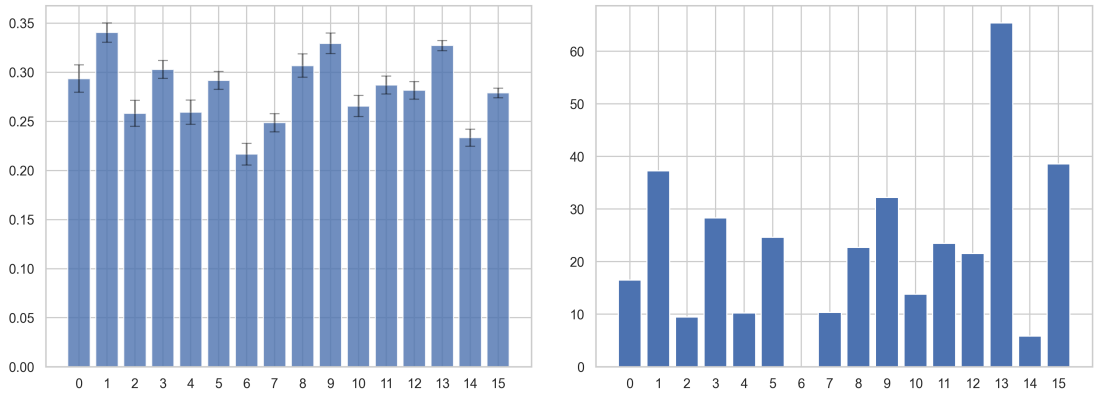
可得评价公式：

$$score = \frac{\mu - \mu_{min}}{\sigma} \tag{27}$$

（2）利用（1）中的评价公式完成对不同方案的评价

以问题 2 为例完成对不同方案的评价，暂不考虑决策数量较多的问题 3。

以情况 1 为例，图18中图 (a) 展示了 16 种方案下的预期利润率，误差线为 $\mu \pm 3\sigma$。



(a) 情况 1 利润率　　**(b) 情况 1 综合得分**

图 18　情况 1 利润率和综合得分

经计算，16 种方案的得分如图18中图 (b) 所示。

情况 1 发生时，选择方案 1101，其他 5 种情况的评价模型可视化见支撑材料，所得问题 2 在次品率通过抽样检测方法求得的条件下的评价结果汇总展示如下表8：

表 8　在次品率通过抽样检测方法求得的条件下，问题 2 的结论

情况	零配件 1 检测	零配件 2 检测	成品检测	成品拆解	利润率
1	是	是	否	是	32.24% (±0.54%)
2	是	是	否	是	21.26% (±0.80%)
3	是	是	是	是	28.53% (±0.52%)
4	是	是	是	是	26.18% (±0.73%)
5	否	是	否	是	23.40% (±1.35%)
6	是	否	否	否	38.04% (±0.90%)

特别地，情况 5 发生时，零配件 1 的检测成本显著高于其他情况，故不考虑方案 1000-方案 1111。

同理可得在次品率通过抽样检测方法求得的条件下，问题 3 的结论：

决策：零件 1-8 都不检测，半成品 1-3 都检测并拆解，成品也检测并拆解

期望利润率：24.78% (±1.90%)

六、 模型的评价与改进

6.1 模型的优点

1. **序贯概率比检验具有成本效益的优越性**：在序贯抽样的过程中，样本依据逐步累积的数据信息，动态地调整抽样计划，确保了抽样过程的灵活性，同时也能够显著减少不必要的样本采集，从而有效降低检测成本。
2. **决策树和蒙特卡洛模型互相验证，兼顾全面性和可解释性**：决策树具有较强的可解释性，清晰呈现决策路径与结果关联；蒙特卡洛模型则更具有全面性。两者互补，使模型更为完善与准确。两种模型独立计算得到的结果相互验证，当两种方法结果一致时，显著增强了结论的可信度。
3. **SEGA 算法具有更快收敛速度**：SEGA 算法通过引入增强精英保留，使得 SEGA 在运行时，相较于传统的遗传算法，能够更快地达到收敛状态，从而提高了算法的执行效率和结果质量。
4. **夏普比率期望贴合实际**：在模型检验时从实际应用角度出发，运用夏普比率综合反映利润率和利润率波动幅度之间的关系，更全面地检验模型能力。

6.2 模型的缺点与改进

序贯概率比检验的缺点与改进

- **缺点**：逐个抽取并检测样本的方式可能会在一定程度上降低工厂的检测效率，因为这种方法每抽取一次就需要重新进行一遍检测，时间成本较高。
- **改进**：采用逐组序贯抽样检验法，每次从待检产品中抽取 n 个样本作为一个检验组，对这 n 个样本同时进行统一检测，并根据检测结果，决定是否继续抽取下一组样本或停止抽样。通过每轮检测多个样本，可以减少检测的总轮数，从而提高检测效率。虽然单次检测成本可能因样本量增加而略有上升，但如果考虑工厂的时间成本，综合检测成本可能会降低。

参考文献

[1] 王庆仁, 康学政, 信海红. 抽样检验技术 [J]. 北京: 中国计量出版社, 1998:64-72.

[2] Wald, A. “Sequential Tests of Statistical Hypotheses.”The Annals of Mathematical Statistics 16, no. 2 (1945): 117−186.

[3] 运筹学教程 [M]. 清华大学出版社有限公司, 2003.

附录 A　支撑材料列表

（一）源程序列表

问题 1：

- 1.1：问题 1（1）序贯抽样随机模拟
- 1.2：问题 1（2）序贯抽样随机模拟

问题 2：

- 2.1：绘制问题 2 中的次品率迭代图
- 2.2：问题 2 的随机模拟

问题 3：

- 3.1：绘制问题 3 的随机采样示意图
- 3.2：问题 3 增强精英遗传算法

问题 4：

- 4.1：问题 4 条件下重新完成问题 2
- 4.2：问题 4 条件下重新完成问题 3

（二）图像列表

问题 2：图 1

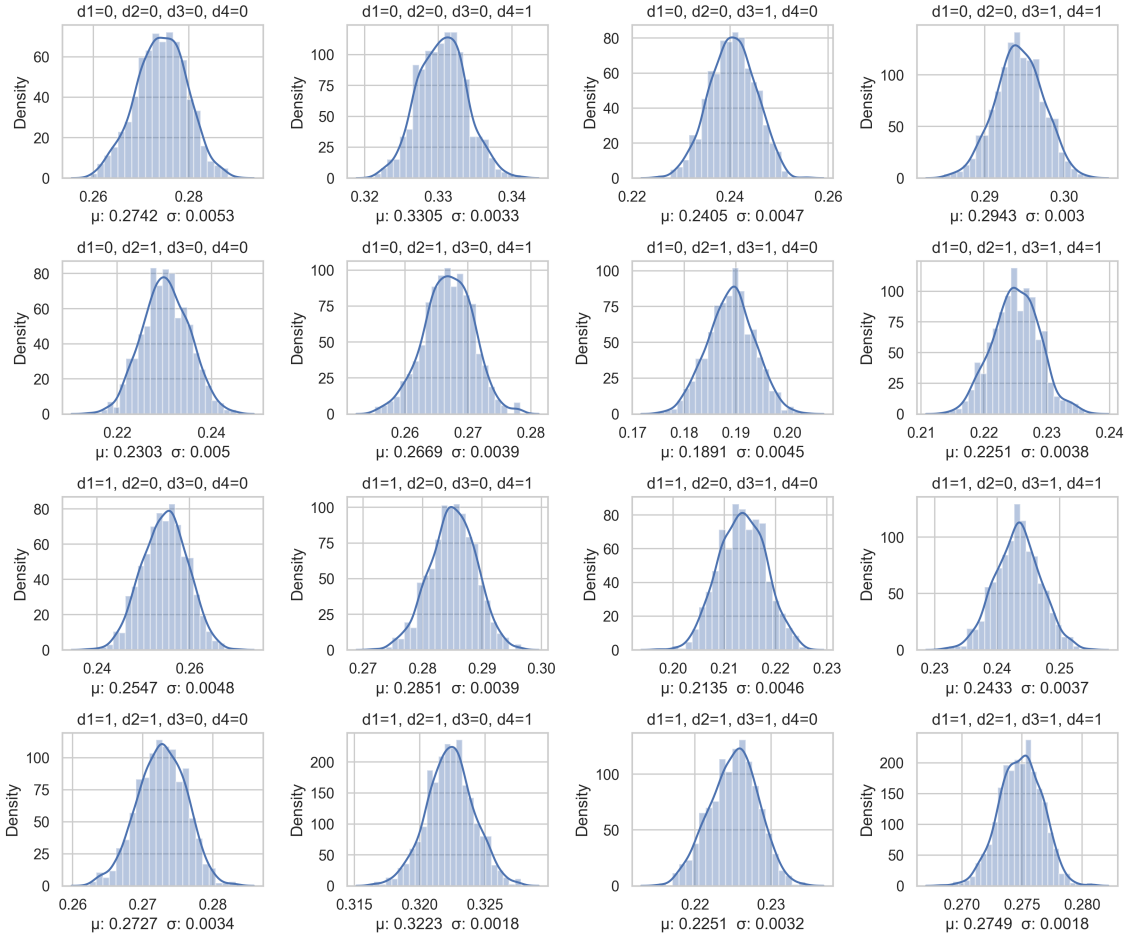
问题 4：图 2、图 3、图 4、图 5、图 6、图 7、图 8、图 9、图 10、图 11、图 12、图 13

（三）文件列表

问题 2：

- 完整决策树：问题 2 决策流程的完整决策树

附录 B　情况一 16 种决策利润率分布图



附录 C　问题 1：（1）序贯抽样随机模拟

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = 'SimHei'

import seaborn as sns
sns.set_theme(style='whitegrid')

import warnings
warnings.filterwarnings('ignore')

from tqdm.notebook import tqdm
p0, p1 = 0.1, 0.12
 ,   = 0.05, 0.1
A =   / (1 -  )
B = (1 -  ) /
res = [] # 每次循环是否被接受
num = [] # 抽样个数
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=2000)
        for i in range(1, 2001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d)) #
                计算似然比
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
            else:
                pass
        else:
            r.append(0)
            n.append(2000)
```

```
    res.append(r)
    num.append(n)
res = np.array(res)
num = np.array(num)

p0, p1 = 0.1, 0.12
 ,   = 0.05, 0.05
A =   / (1 -  )
B = (1 -  ) /
res_ _05 = []
num_ _05 = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=2000)
        for i in range(1, 2001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
            else:
                pass
        else:
            r.append(0)
            n.append(2000)
    res_ _05.append(r)
    num_ _05.append(n)
res_ _05 = np.array(res_ _05)
num_ _05 = np.array(num_ _05)

p0, p1 = 0.1, 0.12
 ,   = 0.05, 0.15
A =   / (1 -  )
B = (1 -  ) /
res_ _15 = []
num_ _15 = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
```

```
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=2000)
        for i in range(1, 2001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
            else:
                pass
        else:
            r.append(0)
            n.append(2000)
    res_ _15.append(r)
    num_ _15.append(n)
res_ _15 = np.array(res_ _15)
num_ _15 = np.array(num_ _15)

p0, p1 = 0.1, 0.13
 ,   = 0.05, 0.1
A =   / (1 -  )
B = (1 -  ) /
res_p1_13 = []
num_p1_13 = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=2000)
        for i in range(1, 2001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
```

```
            else:
                pass
        else:
            r.append(0)
            n.append(2000)
    res_p1_13.append(r)
    num_p1_13.append(n)
res_p1_13 = np.array(res_p1_13)
num_p1_13 = np.array(num_p1_13)

p0, p1 = 0.1, 0.11
 ,   = 0.05, 0.1
A =   / (1 -  )
B = (1 -  ) /
res_p1_11 = []
num_p1_11 = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=3000)
        for i in range(1, 3001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
            else:
                pass
        else:
            r.append(0)
            n.append(3000)
    res_p1_11.append(r)
    num_p1_11.append(n)
res_p1_11 = np.array(res_p1_11)
num_p1_11 = np.array(num_p1_11)

from scipy.interpolate import CubicSpline # 三次样条插值平滑曲线

x = range_
```

```
y_res = res.sum(axis=1) / res.shape[1]
cs = CubicSpline(x, y_res)
y_res = cs(np.linspace(x[0], x[-1], 100))

y_res_ _05 = res_ _05.sum(axis=1) / res_ _05.shape[1]
cs = CubicSpline(x, y_res_ _05)
y_res_ _05 = cs(np.linspace(x[0], x[-1], 100))

y_res_ _15 = res_ _15.sum(axis=1) / res_ _15.shape[1]
cs = CubicSpline(x, y_res_ _15)
y_res_ _15 = cs(np.linspace(x[0], x[-1], 100))

y_res_p1_13 = res_p1_13.sum(axis=1) / res_p1_13.shape[1]
cs = CubicSpline(x, y_res_p1_13)
y_res_p1_13 = cs(np.linspace(x[0], x[-1], 100))

y_res_p1_11 = res_p1_11.sum(axis=1) / res_p1_11.shape[1]
cs = CubicSpline(x, y_res_p1_11)
y_res_p1_11 = cs(np.linspace(x[0], x[-1], 100))

y_num = num.mean(axis=1)
cs = CubicSpline(x, y_num)
y_num = cs(np.linspace(x[0], x[-1], 100))

y_num_ _05 = num_ _05.mean(axis=1)
cs = CubicSpline(x, y_num_ _05)
y_num_ _05 = cs(np.linspace(x[0], x[-1], 100))

y_num_ _15 = num_ _15.mean(axis=1)
cs = CubicSpline(x, y_num_ _15)
y_num_ _15 = cs(np.linspace(x[0], x[-1], 100))

y_num_p1_13 = num_p1_13.mean(axis=1)
cs = CubicSpline(x, y_num_p1_13)
y_num_p1_13 = cs(np.linspace(x[0], x[-1], 100))

y_num_p1_11 = num_p1_11.mean(axis=1)
cs = CubicSpline(x, y_num_p1_11)
y_num_p1_11 = cs(np.linspace(x[0], x[-1], 100))

x = np.linspace(x[0], x[-1], 100)

from pathlib import Path

IMAGES_PATH = Path() / 'images'
IMAGES_PATH.mkdir(parents=True, exist_ok=True)
```

```
def save_fig(fig_id, fig_extension='png', tight_layout=True, resolution=300):
    path = IMAGES_PATH / f'{fig_id}.{fig_extension}'
    if tight_layout:
        plt.tight_layout()
plt.savefig(path, format=fig_extension, dpi=resolution)

fig, ax = plt.subplots(2, 2, sharex=True)
fig.set_figwidth(8)
fig.set_figheight(7)
ax1, ax2 = ax
ax11, ax12 = ax1
ax21, ax22 = ax2

ax11.plot(x, y_res_ _15, label=' =0.15')
ax11.plot(x, y_res, label=' =0.1')
ax11.plot(x, y_res_ _05, label=' =0.05')
ax11.fill_between(x, y_res_ _05, y_res_ _15, color='gray', alpha=0.5)
ax11.set_ylabel('L(p)')
ax11.legend()

ax12.plot(x, y_res_p1_11, label='p1=0.11')
ax12.plot(x, y_res, label='p1=0.12')
ax12.plot(x, y_res_p1_13, label='p1=0.13')
ax12.fill_between(x, y_res_p1_11, y_res_p1_13, color='gray', alpha=0.5)
ax12.legend()

ax21.plot(x, y_num_ _15, label=' =0.15')
ax21.plot(x, y_num, label=' =0.1')
ax21.plot(x, y_num_ _05, label=' =0.05')
ax21.fill_between(x, y_num_ _15, y_num_ _05, color='gray', alpha=0.5)
ax21.set_xticks(np.arange(0, 0.25, 0.05))
ax21.set_xlabel('p')
ax21.set_yticks(np.arange(0, 1800, 300))
ax21.set_ylabel('ASN')
ax21.legend()

ax22.plot(x, y_num_p1_11, label='p1=0.11')
ax22.plot(x, y_num, label='p1=0.12')
ax22.plot(x, y_num_p1_13, label='p1=0.13')
ax22.fill_between(x, y_num_p1_11, y_num_p1_13, color='gray', alpha=0.3)
ax22.set_xticks(np.arange(0, 0.25, 0.05))
ax22.set_xlabel('p')
ax22.set_yticks(np.arange(0, 3000, 500))
ax22.legend()

fig.tight_layout()
save_fig('序贯抽样_1')
```

```
plt.show()

# 准备ASN-P1数据

p0 = 0.1
 ,   = 0.05, 0.1
A =   / (1 -  )
B = (1 -  ) /
number = [] # 每个p1值对应的最大平均抽样个数
for p1 in np.arange(0.108, 0.14, 0.004):
    numb = []
    for p in tqdm(np.arange(0.108, 0.14, 0.004)):
        n = []
        for _ in range(1000):
            m = np.random.binomial(n=1, p=p, size=4000)
            for i in range(1, 4001):
                d = m[:i].sum()
                L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
                if L < A:
                    n.append(i)
                    break
                elif L > B:
                    n.append(i)
                    break
                else:
                    pass
            else:
                n.append(4000)
        numb.append(np.mean(n))
    number.append(max(numb))
number = np.array(number)

from scipy.interpolate import CubicSpline

x = np.arange(0.108, 0.14, 0.004)
y = number

cs = CubicSpline(x, y)

x_new = np.linspace(x[0], x[-1], 100)
y_new = cs(x_new)

plt.plot(x, y, 'o')
plt.plot(x_new, y_new, '-')
plt.xlabel('p1')
plt.xticks(np.arange(0.105, 0.145, 0.005))
plt.ylabel('ASN')
```

```
plt.yticks(np.arange(0, 3500, 500))
plt.ylim([0, 3000])
save_fig('ASN-p1')
plt.show()
```

附录 D　问题 1：（2）序贯抽样随机模拟

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = 'SimHei'

import seaborn as sns
sns.set_theme(style='whitegrid')

import warnings
warnings.filterwarnings('ignore')

from tqdm.notebook import tqdm

p0, p1 = 0.08, 0.1
 ,   = 0.1, 0.1
A =   / (1 -  )
B = (1 -  ) /
res = []
num = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=2000)
        for i in range(1, 2001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
            else:
                pass
        else:
            r.append(0)
            n.append(2000)
```

```
    res.append(r)
    num.append(n)
res = np.array(res)
num = np.array(num)

p0, p1 = 0.08, 0.1
 ,   = 0.05, 0.1
A =   / (1 -  )
B = (1 -  ) /
res_ _05 = []
num_ _05 = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=2000)
        for i in range(1, 2001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
            else:
                pass
        else:
            r.append(0)
            n.append(2000)
    res_ _05.append(r)
    num_ _05.append(n)
res_ _05 = np.array(res_ _05)
num_ _05 = np.array(num_ _05)

p0, p1 = 0.08, 0.1
 ,   = 0.15, 0.1
A =   / (1 -  )
B = (1 -  ) /
res_ _15 = []
num_ _15 = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
```

```
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=2000)
        for i in range(1, 2001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
            else:
                pass
        else:
            r.append(0)
            n.append(2000)
    res_ _15.append(r)
    num_ _15.append(n)
res_ _15 = np.array(res_ _15)
num_ _15 = np.array(num_ _15)

p0, p1 = 0.07, 0.1
 ,   = 0.05, 0.1
A =   / (1 -  )
B = (1 -  ) /
res_p0_07 = []
num_p0_07 = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=2000)
        for i in range(1, 2001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
```

```
            else:
                pass
        else:
            r.append(0)
            n.append(2000)
    res_p0_07.append(r)
    num_p0_07.append(n)
res_p0_07 = np.array(res_p0_07)
num_p0_07 = np.array(num_p0_07)

p0, p1 = 0.09, 0.1
 ,   = 0.05, 0.1
A =   / (1 -  )
B = (1 -  ) /
res_p0_09 = []
num_p0_09 = []
range_ = np.arange(0, 0.21, 0.01)
for p in tqdm(range_):
    r = []
    n = []
    for _ in range(1000):
        m = np.random.binomial(n=1, p=p, size=3000)
        for i in range(1, 3001):
            d = m[:i].sum()
            L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
            if L < A:
                r.append(1) # 接受
                n.append(i)
                break
            elif L > B:
                r.append(0) # 拒绝
                n.append(i)
                break
            else:
                pass
        else:
            r.append(0)
            n.append(3000)
    res_p0_09.append(r)
    num_p0_09.append(n)
res_p0_09 = np.array(res_p0_09)
num_p0_09 = np.array(num_p0_09)

from scipy.interpolate import CubicSpline

x = range_
```

```
y_res = res.sum(axis=1) / res.shape[1]
cs = CubicSpline(x, y_res)
y_res = cs(np.linspace(x[0], x[-1], 100))

y_res_ _05 = res_ _05.sum(axis=1) / res_ _05.shape[1]
cs = CubicSpline(x, y_res_ _05)
y_res_ _05 = cs(np.linspace(x[0], x[-1], 100))

y_res_ _15 = res_ _15.sum(axis=1) / res_ _15.shape[1]
cs = CubicSpline(x, y_res_ _15)
y_res_ _15 = cs(np.linspace(x[0], x[-1], 100))

y_res_p0_07 = res_p0_07.sum(axis=1) / res_p0_07.shape[1]
cs = CubicSpline(x, y_res_p0_07)
y_res_p0_07 = cs(np.linspace(x[0], x[-1], 100))

y_res_p0_09 = res_p0_09.sum(axis=1) / res_p0_09.shape[1]
cs = CubicSpline(x, y_res_p0_09)
y_res_p0_09 = cs(np.linspace(x[0], x[-1], 100))

y_num = num.mean(axis=1)
cs = CubicSpline(x, y_num)
y_num = cs(np.linspace(x[0], x[-1], 100))

y_num_ _05 = num_ _05.mean(axis=1)
cs = CubicSpline(x, y_num_ _05)
y_num_ _05 = cs(np.linspace(x[0], x[-1], 100))

y_num_ _15 = num_ _15.mean(axis=1)
cs = CubicSpline(x, y_num_ _15)
y_num_ _15 = cs(np.linspace(x[0], x[-1], 100))

y_num_p0_07 = num_p0_07.mean(axis=1)
cs = CubicSpline(x, y_num_p0_07)
y_num_p0_07 = cs(np.linspace(x[0], x[-1], 100))

y_num_p0_09 = num_p0_09.mean(axis=1)
cs = CubicSpline(x, y_num_p0_09)
y_num_p0_09 = cs(np.linspace(x[0], x[-1], 100))

x = np.linspace(x[0], x[-1], 100)

from pathlib import Path

IMAGES_PATH = Path() / 'images'
IMAGES_PATH.mkdir(parents=True, exist_ok=True)
```

```
def save_fig(fig_id, fig_extension='png', tight_layout=True, resolution=300):
    path = IMAGES_PATH / f'{fig_id}.{fig_extension}'
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

fig, ax = plt.subplots(2, 2, sharex=True)
fig.set_figwidth(8)
fig.set_figheight(7)
ax1, ax2 = ax
ax11, ax12 = ax1
ax21, ax22 = ax2

ax11.plot(x, y_res_ _15, label=' =0.15')
ax11.plot(x, y_res, label=' =0.1')
ax11.plot(x, y_res_ _05, label=' =0.05')
ax11.fill_between(x, y_res_ _05, y_res_ _15, color='gray', alpha=0.5)
ax11.set_ylabel('L(p)')
ax11.legend()

ax12.plot(x, y_res_p0_09, label="p0'=0.09")
ax12.plot(x, y_res, label="p0'=0.08")
ax12.plot(x, y_res_p0_07, label="p0'=0.07")
ax12.fill_between(x, y_res_p0_09, y_res_p0_07, color='gray', alpha=0.5)
ax12.legend()

ax21.plot(x, y_num_ _15, label=' =0.15')
ax21.plot(x, y_num, label=' =0.1')
ax21.plot(x, y_num_ _05, label=' =0.05')
ax21.fill_between(x, y_num_ _05, y_num_ _15, color='gray', alpha=0.5)
ax21.set_xticks(np.arange(0, 0.25, 0.05))
ax21.set_xlabel('p')
ax21.set_yticks(np.arange(0, 1800, 300))
ax21.set_ylabel('ASN')
ax21.legend()

ax22.plot(x, y_num_p0_09, label="p0'=0.09")
ax22.plot(x, y_num, label="p0'=0.08")
ax22.plot(x, y_num_p0_07, label="p0'=0.07")
ax22.fill_between(x, y_num_p0_09, y_num_p0_07, color='gray', alpha=0.3)
ax22.set_xticks(np.arange(0, 0.25, 0.05))
ax22.set_xlabel('p')
ax22.set_yticks(np.arange(0, 3000, 500))
ax22.legend()

fig.tight_layout()
save_fig('序贯抽样_2')
```

```
plt.show()

p1 = 0.1
 ,   = 0.05, 0.1
A =   / (1 -  )
B = (1 -  ) /
number = []
for p0 in np.arange(0.06, 0.096, 0.004):
    numb = []
    for p in tqdm(np.arange(0.06, 0.096, 0.004)):
        n = []
        for _ in range(1000):
            m = np.random.binomial(n=1, p=p, size=4000)
            for i in range(1, 4001):
                d = m[:i].sum()
                L = (p1 ** d * (1 - p1) ** (i - d)) / (p0 ** d * (1 - p0) ** (i - d))
                if L < A:
                    n.append(i)
                    break
                elif L > B:
                    n.append(i)
                    break
                else:
                    pass
            else:
                n.append(4000)
        numb.append(np.mean(n))
    number.append(max(numb))
number = np.array(number)

from scipy.interpolate import CubicSpline

x = np.arange(0.06, 0.096, 0.004)
y = number

cs = CubicSpline(x, y)

x_new = np.linspace(x[0], x[-1], 100)
y_new = cs(x_new)

plt.plot(x, y, 'o')
plt.plot(x_new, y_new, '-')
plt.xlabel("p0'")
plt.xticks(np.arange(0.06, 0.095, 0.005))
plt.ylabel('ASN')
plt.yticks(np.arange(0, 3500, 500))
plt.ylim([0, 3000])
```

```
save_fig('ASN-p0')
plt.show()

# 若始终不检测零件，零件/成品次品率随迭代次数的变化趋势

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = 'SimHei'

import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

from pathlib import Path

IMAGES_PATH = Path() / 'images'
IMAGES_PATH.mkdir(parents=True, exist_ok=True)

def save_fig(fig_id, fig_extension='png', tight_layout=True, resolution=300):
    # 保存图片
    path = IMAGES_PATH / f'{fig_id}.{fig_extension}'
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

a, b, c = 0.1, 0.1, 0.1
P1 = [] # 零件次品率
F1 = [] # 成品次品率

p1, p2, p3 = a, b, c
for _ in range(5):
    P1.append(p1)
    F1.append(1 - (1 - p3) * (1 - p1) * (1 - p2))
    p00 = p3 * (1 - p1) * (1 - p2)
    p01 = (1 - p1) * p2
    p10 = p1 * (1 - p2)
    p11 = p1 * p2
    p1 = (p10 + p11) / (p00 + p01 + p10 + p11) # 更新p1值
    p2 = (p01 + p11) / (p00 + p01 + p10 + p11) # 更新p2值

a, b, c = 0.1, 0, 0.1
P2 = [] # 零件次品率
```

```
F2 = [] # 成品次品率

p1, p2, p3 = a, b, c
for _ in range(5):
    P2.append(p1)
    F2.append(1 - (1 - p3) * (1 - p1) * (1 - p2))
    p00 = p3 * (1 - p1) * (1 - p2)
    p01 = (1 - p1) * p2
    p10 = p1 * (1 - p2)
    p11 = p1 * p2
    p1 = (p10 + p11) / (p00 + p01 + p10 + p11)
    p2 = (p01 + p11) / (p00 + p01 + p10 + p11)

plt.figure(figsize=(4*2, 4))

ax1 = plt.subplot(1, 2, 1)
ax1.plot(P1, label='零件1次品率')
ax1.plot(F1, label='成品次品率')
ax1.set_xlabel('拆解次数', fontsize=12)
ax1.set_xticks(range(5))
ax1.set_yticks(np.arange(0, 1.1, 0.1))
ax1.set_ylim([0, 1])
ax1.set_title('p1=p2=p3=0.1', fontsize=14)
ax1.legend(loc=4)
ax1.grid()

ax2 = plt.subplot(1, 2, 2)
ax2.plot(P2, label='零件1次品率')
ax2.plot(F2, label='成品次品率')
ax2.set_xlabel('拆解次数', fontsize=12)
ax2.set_xticks(range(5))
ax2.set_yticks(np.arange(0, 1.1, 0.1))
ax2.set_ylim([0, 1])
ax2.set_title('p1=p3=0.1, p2=0', fontsize=14)
ax2.legend(loc=4)
ax2.grid()

save_fig('次品率迭代图')
plt.show()
```

附录 E　问题 2：随机模拟

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = 'SimHei'

import seaborn as sns
sns.set_theme(style='whitegrid')

import warnings
warnings.filterwarnings('ignore')

from tqdm.notebook import tqdm

from pathlib import Path

IMAGES_PATH = Path() / 'images'
IMAGES_PATH.mkdir(parents=True, exist_ok=True)

def save_fig(fig_id, fig_extension='png', tight_layout=True, resolution=300):
    # 保存图片
    path = IMAGES_PATH / f'{fig_id}.{fig_extension}'
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

def simulation(m1, m2, d1, d2, d3, d4):
    # 模拟（通过修改全局变量cost, revenue实现）
    global cost, revenue, times, m1t, m2t
    if len(m1) == 0:
        return
    times += 1
    if d1:
        m1t = 1
        cost += t1 * len(m1)
        m1 = m1[m1 == 0]
    if d2:
        m2t = 1
        cost += t2 * len(m2)
        m2 = m2[m2 == 0]
    num = min(len(m1), len(m2))
    m1, m2 = m1[:num], m2[:num] # 丢弃未配对零件
```

```
    m3 = ((m1 + m2 + np.random.rand(num)) > 1 - p3).astype(int) # 根据m1,
        m2和随机扰动生成成品m3
    cost += c3 * len(m3)
    revenue += s * len(m3[m3 == 0])
    if d3:
        cost += t3 * len(m3)
    else:
        cost += e * len(m3) * m3.mean()
    if d4:
        cost += d * m3.sum()
        simulation(m1[m3 == 1], m2[m3 == 1], 1-m1t, 1-m2t, d3, d4)

# 六种情况的参数
para = [
    [0.1 , 4, 2, 0.1 , 18, 3, 0.1 , 6, 3, 56, 6 , 5 ],
    [0.2 , 4, 2, 0.2 , 18, 3, 0.2 , 6, 3, 56, 6 , 5 ],
    [0.1 , 4, 2, 0.1 , 18, 3, 0.1 , 6, 3, 56, 30, 5 ],
    [0.2 , 4, 1, 0.2 , 18, 1, 0.2 , 6, 2, 56, 30, 5 ],
    [0.1 , 4, 8, 0.2 , 18, 1, 0.1 , 6, 2, 56, 10, 5 ],
    [0.05, 4, 2, 0.05, 18, 3, 0.05, 6, 3, 56, 10, 40],
]

situation = 6 # 模拟第几种情况
name = f'第二问情况{situation}利润率'
p1, c1, t1, p2, c2, t2, p3, c3, t3, s, e, d = para[situation - 1]
dict = {0: (0, 0, 0, 0), 1: (0, 0, 0, 1), 2: (0, 0, 1, 0), 3: (0, 0, 1, 1), 4: (0, 1,
    0, 0), 5: (0, 1, 0, 1), 6: (0, 1, 1, 0), 7: (0, 1, 1, 1),
       8: (1, 0, 0, 0), 9: (1, 0, 0, 1), 10: (1, 0, 1, 0), 11: (1, 0, 1, 1), 12: (1,
           1, 0, 0), 13: (1, 1, 0, 1), 14: (1, 1, 1, 0), 15: (1, 1, 1, 1)}
size = 10000
all_profit_rate = []
all_T = []
for i in tqdm(range(len(dict))):
    d1, d2, d3, d4 = dict[i]
    profit_rate = []
    T = []

    for _ in range(1000):
        m1t, m2t = 0, 0 # 零件12是否被检测过

        if d1:
            m1 = np.random.binomial(n=1, p=p1, size=int(size/(1-p1))) #
                若决定检测，则增加购买量，使得期望合格数等于基准值
        else:
            m1 = np.random.binomial(n=1, p=p1, size=size)

        if d2:
```

```
            m2 = np.random.binomial(n=1, p=p2, size=int(size/(1-p2)))
        else:
            m2 = np.random.binomial(n=1, p=p2, size=size)

        cost = c1 * len(m1) + c2 * len(m2)
        revenue = 0
        times = 0

        simulation(m1, m2, d1, d2, d3, d4)
        profit_rate.append((revenue - cost) / revenue)
        T.append(times)
    all_profit_rate.append(np.array(profit_rate))
    all_T.append(np.array(T))
all_profit_rate = np.array(all_profit_rate)
all_T = np.array(all_T)

print(f'对应决策索引：{(all_profit_rate.mean(axis=1)).argmax()}')

x = np.arange(16)
w = 0.4
plt.figure(figsize=(6, 4))
plt.bar(x - 0.5 * w, all_profit_rate.mean(axis=1), width=w, label=' ')
plt.bar(x + 0.5 * w, all_profit_rate.std(axis=1), width=w, label=' ')
plt.xticks(range(16))
plt.legend()
save_fig(name)
plt.show()

df[f'情况{situation}'] = np.around(all_profit_rate.mean(axis=1), 3)

all_profit_rate = []
for i in tqdm(range(len(dict))):
    d1, d2, d3, d4 = dict[i]
    profit_rate = []

    for _ in range(1000):
        m1t, m2t = 0, 0

        m1 = np.random.binomial(n=1, p=p1, size=size) #
            不使用优化购买的策略，只购买基准值的个数
        m2 = np.random.binomial(n=1, p=p2, size=size)

        cost = c1 * len(m1) + c2 * len(m2)
        revenue = 0

        simulation(m1, m2, d1, d2, d3, d4)
        profit_rate.append((revenue - cost) / revenue)
```

```
    all_profit_rate.append(np.array(profit_rate))
all_profit_rate = np.array(all_profit_rate)

print(f'对应决策索引: {(all_profit_rate.mean(axis=1)).argmax()}')

x = np.arange(16)
w = 0.4
plt.figure(figsize=(6, 4))
plt.bar(x - 0.5 * w, all_profit_rate.mean(axis=1), width=w, label=' ')
plt.bar(x + 0.5 * w, all_profit_rate.std(axis=1), width=w, label=' ')
plt.xticks(range(16))
plt.legend()
plt.show()

fig, ax = plt.subplots(4, 4)
fig.set_figwidth(12)
fig.set_figheight(10)
ax = ax.ravel()
for i, axi in enumerate(ax):
    sns.distplot(all_profit_rate[i], ax=axi)
    axi.set_title(f'd1={dict[i][0]}, d2={dict[i][1]}, d3={dict[i][2]}, d4={dict[i][3]}')
    axi.set_xlabel(f' : {round(all_profit_rate[i].mean(), 4)}  :
        {round(all_profit_rate[i].std(), 4)}')
plt.tight_layout()
save_fig('第二问情况一16种决策利润率分布')
plt.show()
```

附录 F　问题 2：不确定型决策的判断

```
% 乐观准则
table_1 = max(profit_rate,[],2);
table_2 = table_1';
[PR,index] = max(table_2)

% 悲观准则
table_1 = min(profit_rate,[],2);
[PR,index] = max(table_1)

% 折中准则
index_matrix = zeros(21,1);
lamda = linspace(0,1,21);
for i = 1:21
lamda_use = lamda(i);
PR = [];
index = [];
table_1 = lamda_use.*max(profit_rate,[],2)+(1-lamda_use).*min(profit_rate,[],2);
[PR,index] = max(table_1);
index_matrix(i) = index;
end
index_matrix

% 等可能准则
table_1 = sum(profit_rate,2)./6;
[PR,index] = max(table_1)

% 最小后悔值准则
new_profit_rate = max(profit_rate,[],1) - profit_rate;
table_1 = max(new_profit_rate,[],2);
[PR,index] = min(table_1)
```

附录 G　问题 3：随机模拟

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = 'SimHei'

import seaborn as sns
sns.set_theme(style='whitegrid')

import warnings
warnings.filterwarnings('ignore')

para_p = np.array([
    [0.1, 2, 1], [0.1, 8, 1], [0.1, 12, 2],
    [0.1, 2, 1], [0.1, 8, 1], [0.1, 12, 2],
                [0.1, 8, 1], [0.1, 12, 2],
])                                      # 零件参数（次品率、购买单价、检测成本）

para_sf = np.array([[0.1, 8, 4, 6]] * 3) #
    半成品参数（次品率、装配成本、检测成本、拆解费用）

para_f = np.array([0.1, 8, 6, 10, 200, 40]) #
    成品参数（次品率、装配成本、检测成本、拆解费用、市场售价、调换损失）

def make_sf(p_index, sf_index):
    # 合成某一件半成品，p_index为零件索引（列表），sf_index为半成品索引（数值）
    global revenue, cost
    if len(P[p_index[0]]) == 0:
        return
    for i in p_index:
        if D[i]:
            PT[i] = 1
            cost += para_p[i][2] * len(P[i])
            P[i] = P[i][P[i] == 0]
        D[i] = 1 - PT[i]
    num = min([len(P[i]) for i in p_index])
    for i in p_index:
        P[i] = P[i][:num]
    sf_temp = ((sum([P[i] for i in p_index]) + np.random.rand(num)) > 1 -
        para_sf[sf_index][0]).astype(int)
    cost += para_sf[sf_index][1] * num
```

```
    if D[len(para_p) + sf_index * 2]:
        PT[len(para_p) + sf_index] = 1
        cost += para_sf[sf_index][2] * len(sf_temp)
        for i in p_index:
            P[i] = P[i][sf_temp == 1]
        sf_temp = sf_temp[sf_temp == 0]
        SF[sf_index] = np.concatenate([SF[sf_index], sf_temp])
        if D[len(para_p) + sf_index * 2 + 1]:
            cost += para_sf[sf_index][3] * len(P[p_index[0]])
            make_sf(p_index, sf_index)
    else:
        SF[sf_index] = np.concatenate([SF[sf_index], sf_temp])


def make_f():
    # 合成成品
    global revenue, cost
    num = min([len(sf) for sf in SF])
    if num == 0:
        return
    for i in range(len(SF)):
        SF[i] = SF[i][:num]
    F = ((sum([SF[i] for i in range(len(SF))]) + np.random.rand(num)) > 1 -
         para_f[0]).astype(int)
    revenue += para_f[4] * len(F[F == 0])
    cost += para_f[1] * len(F)
    if D[-2]:
        cost += para_f[2] * len(F)
    else:
        cost += para_f[5] * len(F) * F.mean()
    if D[-1]:
        cost += para_f[3] * F.sum()
        for sf_index in range(len(SF)):
            SF[sf_index] = SF[sf_index][F == 1]
            if 1 - PT[len(para_p) + sf_index]:
                PT[len(para_p) + sf_index] = 1
                cost += para_sf[sf_index][2] * len(SF[sf_index])
                SF[sf_index] = SF[sf_index][SF[sf_index] == 0]
        make_f()

decision = np.zeros(len(para_p) + 2 * (len(para_sf) + 1))       #
    决策变量（是否检测零件、是否检测/拆解（半）成品）
n = 10                                                          # 模拟n次
num_of_p = 10000                                                #
    每次模拟每种零件的初始个数
profit_rate = []                                                # 存储每次模拟的利润率
```

```
for _ in range(n):
    D = decision.copy()
    P = []                                                    #
        所有零件正/次品序列（二维数组，次品为1）
    SF = [np.array([])] * len(para_sf)                        # 初始化半成品序列
    for i in range(len(para_p)):
        if D[i]:
            P.append(np.random.binomial(n=1, p=para_p[i][0],
                size=int(num_of_p/(1-para_p[i][0]))))
        else:
            P.append(np.random.binomial(n=1, p=para_p[i][0], size=num_of_p))
    PT = np.zeros((len(para_p) + len(para_sf)))               #
        零件/半成品是否已被检测过
    revenue = 0                                               # 初始化收入
    cost = sum([para_p[i][1] * len(P[i]) for i in range(len(para_p))]) # 购买零件的费用
    make_sf([0, 1, 2], 0)                                     # 生产半成品
    make_sf([3, 4, 5], 1)
    make_sf([6, 7], 2)
    make_f()                                                  # 生产成品
    profit_rate.append((revenue - cost) / revenue)

obj = np.array(profit_rate).mean()                            # 求得模拟的平均利润率
obj

from pathlib import Path

IMAGES_PATH = Path() / 'images'
IMAGES_PATH.mkdir(parents=True, exist_ok=True)

def save_fig(fig_id, fig_extension='png', tight_layout=True, resolution=300):
    # 保存图片
    path = IMAGES_PATH / f'{fig_id}.{fig_extension}'
    if tight_layout:
        plt.tight_layout()
plt.savefig(path, format=fig_extension, dpi=resolution)

profit_rate = []
for _ in range(2000):
    D = np.random.binomial(n=1, p=0.5, size=len(para_p) + 2 * (len(para_sf) + 1))
    n = 10                                                    # 模拟n次
    num_of_p = 10000                                          #
        每次模拟每种零件的初始个数
    revenue = 0                                               # 初始化收入
    cost = para_p[:, 1].sum() * num_of_p                      # 购买零件的费用
    P = []                                                    #
        所有零件正/次品序列（二维数组，次品为1）
    SF = [np.array([])] * len(para_sf)                        # 初始化半成品序列
```

```
    for i in range(len(para_p)):
        if D[i]:
            P.append(np.random.binomial(n=1, p=para_p[i][0],
                size=int(num_of_p/(1-para_p[i][0]))))
        else:
            P.append(np.random.binomial(n=1, p=para_p[i][0], size=num_of_p))
    PT = np.zeros((len(para_p) + len(para_sf)))                #
        零件/半成品是否已被检测过
    revenue = 0                                                # 初始化收入
    cost = sum([para_p[i][1] * len(P[i]) for i in range(len(para_p))]) # 购买零件的费用
    make_sf([0, 1, 2], 0)                                      # 生产半成品
    make_sf([3, 4, 5], 1)
    make_sf([6, 7], 2)
    make_f()
    profit_rate.append((revenue - cost) / revenue)

sns.distplot(profit_rate)
save_fig('第三问随机决策利润率分布图')
plt.show()
```

附录 H　问题 3：遗传算法优化

```
import numpy as np

para_p = np.array([
    [0.1, 2, 1], [0.1, 8, 1], [0.1, 12, 2],
    [0.1, 2, 1], [0.1, 8, 1], [0.1, 12, 2],
                [0.1, 8, 1], [0.1, 12, 2],
])                                      # 零件参数（次品率、购买单价、检测成本）

para_sf = np.array([[0.1, 8, 4, 6]] * 3) #
    半成品参数（次品率、装配成本、检测成本、拆解费用）

para_f = np.array([0.1, 8, 6, 10, 200, 40]) #
    成品参数（次品率、装配成本、检测成本、拆解费用、市场售价、调换损失）

n = 10                                                          # 每个决策模拟n次
num_of_p = 10000                                                #
    每次模拟每种零件的初始个数
ini_data = np.ones(len(para_p) + 2 * (len(para_sf) + 1))        # 先验信息

def obj_function(X):

    def make_sf(p_index, sf_index):
        # 合成某一件半成品，p_index为零件索引（列表），sf_index为半成品索引（数值）
        nonlocal revenue, cost
        if len(P[p_index[0]]) == 0:
            return
        for i in p_index:
            if D[i]:
                PT[i] = 1
                cost += para_p[i][2] * len(P[i])
                P[i] = P[i][P[i] == 0]
            D[i] = 1 - PT[i]
        num = min([len(P[i]) for i in p_index])
        for i in p_index:
            P[i] = P[i][:num]
        sf_temp = ((sum([P[i] for i in p_index]) + np.random.rand(num)) > 1 -
            para_sf[sf_index][0]).astype(int)
        cost += para_sf[sf_index][1] * num
        if D[len(para_p) + sf_index * 2]:
            PT[len(para_p) + sf_index] = 1
            cost += para_sf[sf_index][2] * len(sf_temp)
            for i in p_index:
                P[i] = P[i][sf_temp == 1]
```

```
        sf_temp = sf_temp[sf_temp == 0]
        SF[sf_index] = np.concatenate([SF[sf_index], sf_temp])
        if D[len(para_p) + sf_index * 2 + 1]:
            cost += para_sf[sf_index][3] * len(P[p_index[0]])
            make_sf(p_index, sf_index)
    else:
        SF[sf_index] = np.concatenate([SF[sf_index], sf_temp])

def make_f():
    # 合成成品
    nonlocal revenue, cost
    num = min([len(sf) for sf in SF])
    if num == 0:
        return
    for i in range(len(SF)):
        SF[i] = SF[i][:num]
    F = ((sum([SF[i] for i in range(len(SF))]) + np.random.rand(num)) > 1 -
         para_f[0]).astype(int)
    revenue += para_f[4] * len(F[F == 0])
    cost += para_f[1] * len(F)
    if D[-2]:
        cost += para_f[2] * len(F)
    else:
        cost += para_f[5] * len(F) * F.mean()
    if D[-1]:
        cost += para_f[3] * F.sum()
        for sf_index in range(len(SF)):
            SF[sf_index] = SF[sf_index][F == 1]
            if 1 - PT[len(para_p) + sf_index]:
                PT[len(para_p) + sf_index] = 1
                cost += para_sf[sf_index][2] * len(SF[sf_index])
                SF[sf_index] = SF[sf_index][SF[sf_index] == 0]
        make_f()

objv = []
cv = []
for i in range(len(X)):
    D = X[i]
    profit_rate = []                                                    #
         存储每次模拟的利润
    for _ in range(n):
        revenue = 0                                                     # 初始化收入
        cost = para_p[:, 1].sum() * num_of_p                            # 购买零件的费用
        P = []                                                          #
             所有零件正/次品序列（二维数组，次品为1）
        SF = [np.array([])] * len(para_sf)                              # 初始化半成品序列
        for i in range(len(para_p)):
```

```
            P.append(np.random.binomial(n=1, p=para_p[i][0], size=num_of_p)) #
                正/次品随机初始化（服从二项分布）
        PT = np.zeros((len(para_p) + len(para_sf)))                #
            零件/半成品是否已被检测过
        make_sf([0, 1, 2], 0)                                      # 生产半成品
        make_sf([3, 4, 5], 1)
        make_sf([6, 7], 2)
        make_f()                                                   # 生产成品
        profit_rate.append((revenue - cost) / revenue)
    obj = np.array(profit_rate).mean()                             #
        求得模拟的平均利润
    objv.append([obj])
    cv.append([0])
objv = np.array(objv)
cv = np.array(cv)
return objv, cv

import geatpy as ea

problem = ea.Problem(name='SEGA', M=1, maxormins=[-1], Dim=16, varTypes=[1]*16,
    lb=[0]*16, ub=[1]*16, evalVars=obj_function)
Field = ea.crtfld('BG', problem.varTypes[:], ranges=np.array([problem.lb[:],
    problem.ub[:]]))
population = ea.Population('BG', Field=Field, NIND=25)
# 带有先验信息的SEGA算法（增强精英保留遗传算法）
algorithm = ea.soea_SEGA_templet(problem, population, prophetPop=ini_data, MAXGEN=50,
    maxTrappedCount=10, logTras=10)
res = ea.optimize(algorithm, drawing=1, seed=42)
```

附录 I　问题 4：重做问题 2

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

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = 'SimHei'

import seaborn as sns
sns.set_theme(style='whitegrid')

import warnings
warnings.filterwarnings('ignore')

from tqdm.notebook import tqdm

from pathlib import Path

IMAGES_PATH = Path() / 'images'
IMAGES_PATH.mkdir(parents=True, exist_ok=True)

def save_fig(fig_id, fig_extension='png', tight_layout=True, resolution=300):
    path = IMAGES_PATH / f'{fig_id}.{fig_extension}'
    if tight_layout:
        plt.tight_layout()
plt.savefig(path, format=fig_extension, dpi=resolution)

def simulation(m1, m2, d1, d2, d3, d4):
    global cost, revenue, m1t, m2t
    if len(m1) == 0:
        return
    if d1:
        m1t = 1
        cost += t1 * len(m1)
        m1 = m1[m1 == 0]
    if d2:
        m2t = 1
        cost += t2 * len(m2)
        m2 = m2[m2 == 0]
    num = min(len(m1), len(m2))
    m1, m2 = m1[:num], m2[:num] # 丢弃未配对零件
    m3 = ((m1 + m2 + np.random.rand(num)) > 1 - p3).astype(int)
    cost += c3 * len(m3)
```

```
    revenue += s * len(m3[m3 == 0])
    if d3:
        cost += t3 * len(m3)
    else:
        cost += e * len(m3) * m3.mean()
    if d4:
        cost += d * m3.sum()
        simulation(m1[m3 == 1], m2[m3 == 1], 1-m1t, 1-m2t, d3, d4)

# 题干参数
para = [
    [0.1 , 4, 2, 0.1 , 18, 3, 0.1 , 6, 3, 56, 6 , 5 ],
    [0.2 , 4, 2, 0.2 , 18, 3, 0.2 , 6, 3, 56, 6 , 5 ],
    [0.1 , 4, 2, 0.1 , 18, 3, 0.1 , 6, 3, 56, 30, 5 ],
    [0.2 , 4, 1, 0.2 , 18, 1, 0.2 , 6, 2, 56, 30, 5 ],
    [0.1 , 4, 8, 0.2 , 18, 1, 0.1 , 6, 2, 56, 10, 5 ],
    [0.05, 4, 2, 0.05, 18, 3, 0.05, 6, 3, 56, 10, 40],
]

situation = 1
name = f'第四问情况{situation}利润率'
p1, c1, t1, p2, c2, t2, p3, c3, t3, s, e, d = para[situation - 1]
dict = {0: (0, 0, 0, 0), 1: (0, 0, 0, 1), 2: (0, 0, 1, 0), 3: (0, 0, 1, 1), 4: (0, 1,
    0, 0), 5: (0, 1, 0, 1), 6: (0, 1, 1, 0), 7: (0, 1, 1, 1),
       8: (1, 0, 0, 0), 9: (1, 0, 0, 1), 10: (1, 0, 1, 0), 11: (1, 0, 1, 1), 12: (1,
           1, 0, 0), 13: (1, 1, 0, 1), 14: (1, 1, 1, 0), 15: (1, 1, 1, 1)}
size = 10000
all_profit_rate = []
for i in tqdm(range(len(dict))):
    d1, d2, d3, d4 = dict[i]
    profit_rate = []

    for _ in range(1000):
        m1t, m2t = 0, 0

        # 给p按不同的后验beta分布赋值
        if p1 == 0.1:
            p1 = beta.rvs(100, 900, size=1)[0]
        elif p1 == 0.05:
            p1 = beta.rvs(50, 950, size=1)[0]
        elif p1 == 0.2:
            p1 = beta.rvs(200, 800, size=1)[0]

        if p2 == 0.1:
            p2 = beta.rvs(100, 900, size=1)[0]
        elif p2 == 0.05:
            p2 = beta.rvs(50, 950, size=1)[0]
```

```
        elif p2 == 0.2:
            p2 = beta.rvs(200, 800, size=1)[0]

        if p3 == 0.1:
            p3 = beta.rvs(100, 900, size=1)[0]
        elif p3 == 0.05:
            p3 = beta.rvs(50, 950, size=1)[0]
        elif p3 == 0.2:
            p3 = beta.rvs(200, 800, size=1)[0]

        # 使用购买策略
        if d1:
            m1 = np.random.binomial(n=1, p=p1, size=int(size/(1-p1)))
        else:
            m1 = np.random.binomial(n=1, p=p1, size=size)

        if d2:
            m2 = np.random.binomial(n=1, p=p2, size=int(size/(1-p2)))
        else:
            m2 = np.random.binomial(n=1, p=p2, size=size)

        cost = c1 * len(m1) + c2 * len(m2)
        revenue = 0

        simulation(m1, m2, d1, d2, d3, d4)
        profit_rate.append((revenue - cost) / revenue)
    all_profit_rate.append(np.array(profit_rate))
all_profit_rate = np.array(all_profit_rate)

print(f'对应决策索引：{(all_profit_rate.mean(axis=1)).argmax()}')

# 绘制均值±三倍标准差
mean = all_profit_rate.mean(axis=1)
std = all_profit_rate.std(axis=1)
error = 3 * std
x = np.arange(16)
plt.bar(x, mean, yerr=error, align='center', alpha=0.8, ecolor=(0, 0, 0, 0.4),
    capsize=4)
plt.xticks(range(16))
save_fig(name)
plt.show()

# 计算得分
score = (mean - max(mean.min(), 0)) / std
print(f'最优决策索引：{score.argmax()}')
print(f'对应利润率：{(all_profit_rate.max(axis=1))[score.argmax()]}')
print(f'对应三倍标准差：{3 * (all_profit_rate.std(axis=1))[score.argmax()]}')
```

```
plt.bar(x, score)
plt.xticks(range(16))
save_fig(f'第四问情况{situation}综合得分')
plt.show()
```

附录 J　问题 4: 重做问题 3

```
import numpy as np
from scipy.stats import beta

para_P = np.array([
    [0.1, 2, 1], [0.1, 8, 1], [0.1, 12, 2],
    [0.1, 2, 1], [0.1, 8, 1], [0.1, 12, 2],
                [0.1, 8, 1], [0.1, 12, 2],
])                                      # 零件参数（次品率、购买单价、检测成本）

para_SF = np.array([[0.1, 8, 4, 6]] * 3) #
    半成品参数（次品率、装配成本、检测成本、拆解费用）

para_F = np.array([0.1, 8, 6, 10, 200, 40]) #
    成品参数（次品率、装配成本、检测成本、拆解费用、市场售价、调换损失）

n = 100                                                         # 每个决策模拟n次
num_of_p = 10000                                                #
    每次模拟每种零件的初始个数
ini_data = np.ones(len(para_P) + 2 * (len(para_SF) + 1))        # 先验信息

def obj_function(X):

    def make_sf(p_index, sf_index):
        # 合成某一件半成品，p_index为零件索引（列表），sf_index为半成品索引（数值）
        nonlocal revenue, cost
        if len(P[p_index[0]]) == 0:
            return
        for i in p_index:
            if D[i]:
                PT[i] = 1
                cost += para_p[i][2] * len(P[i])
                P[i] = P[i][P[i] == 0]
            D[i] = 1 - PT[i]
        num = min([len(P[i]) for i in p_index])
        for i in p_index:
            P[i] = P[i][:num]
        sf_temp = ((sum([P[i] for i in p_index]) + np.random.rand(num)) > 1 -
            para_sf[sf_index][0]).astype(int)
        cost += para_sf[sf_index][1] * num
        if D[len(para_p) + sf_index * 2]:
            PT[len(para_p) + sf_index] = 1
            cost += para_sf[sf_index][2] * len(sf_temp)
            for i in p_index:
```

```
            P[i] = P[i][sf_temp == 1]
        sf_temp = sf_temp[sf_temp == 0]
        SF[sf_index] = np.concatenate([SF[sf_index], sf_temp])
        if D[len(para_p) + sf_index * 2 + 1]:
            cost += para_sf[sf_index][3] * len(P[p_index[0]])
            make_sf(p_index, sf_index)
    else:
        SF[sf_index] = np.concatenate([SF[sf_index], sf_temp])

def make_f():
    # 合成成品
    nonlocal revenue, cost
    num = min([len(sf) for sf in SF])
    if num == 0:
        return
    for i in range(len(SF)):
        SF[i] = SF[i][:num]
    F = ((sum([SF[i] for i in range(len(SF))]) + np.random.rand(num)) > 1 -
         para_f[0]).astype(int)
    revenue += para_f[4] * len(F[F == 0])
    cost += para_f[1] * len(F)
    if D[-2]:
        cost += para_f[2] * len(F)
    else:
        cost += para_f[5] * len(F) * F.mean()
    if D[-1]:
        cost += para_f[3] * F.sum()
        for sf_index in range(len(SF)):
            SF[sf_index] = SF[sf_index][F == 1]
            if 1 - PT[len(para_p) + sf_index]:
                PT[len(para_p) + sf_index] = 1
                cost += para_sf[sf_index][2] * len(SF[sf_index])
                SF[sf_index] = SF[sf_index][SF[sf_index] == 0]
        make_f()

objv = []
cv = []
for i in range(len(X)):
    D = X[i]
    profit_rate = []                                                    #
         存储每次模拟的利润
    for _ in range(n):
        para_p = para_P.copy()
        para_sf = para_SF.copy()
        para_f = para_F.copy()
        for i in range(len(para_p)):
            para_p[i][0] = beta.rvs(100, 900, size=1)[0]
```

```
            for i in range(len(para_sf)):
                para_sf[i][0] = beta.rvs(100, 900, size=1)[0]
            para_f[0] = beta.rvs(100, 900, size=1)[0]

            P = []                                                  #
                所有零件正/次品序列（二维数组，次品为1）
            SF = [np.array([])] * len(para_sf)                      # 初始化半成品序列
            for i in range(len(para_p)):
                if D[i]:
                    P.append(np.random.binomial(n=1, p=para_p[i][0],
                        size=int(num_of_p/(1-para_p[i][0]))))
                else:
                    P.append(np.random.binomial(n=1, p=para_p[i][0], size=num_of_p))
            PT = np.zeros((len(para_p) + len(para_sf)))             #
                零件/半成品是否已被检测过
            revenue = 0                                             # 初始化收入
            cost = sum([para_p[i][1] * len(P[i]) for i in range(len(para_p))]) #
                购买零件的费用
            make_sf([0, 1, 2], 0)                                   # 生产半成品
            make_sf([3, 4, 5], 1)
            make_sf([6, 7], 2)
            make_f()                                                # 生产成品
            profit_rate.append((revenue - cost) / revenue)
        obj = np.array(profit_rate).mean() / np.array(profit_rate).std()
        objv.append([obj])
        cv.append([0])
    objv = np.array(objv)
    cv = np.array(cv)
return objv, cv

import geatpy as ea

problem = ea.Problem(name='SEGA', M=1, maxormins=[-1], Dim=16, varTypes=[1]*16,
    lb=[0]*16, ub=[1]*16, evalVars=obj_function)
Field = ea.crtfld('BG', problem.varTypes[:], ranges=np.array([problem.lb[:],
    problem.ub[:]]))
population = ea.Population('BG', Field=Field, NIND=25)
# 带有先验信息的SEGA算法（增强精英保留遗传算法）
algorithm = ea.soea_SEGA_templet(problem, population, prophetPop=ini_data, MAXGEN=50,
    maxTrappedCount=10, logTras=10)
res = ea.optimize(algorithm, drawing=1, seed=42)

def make_sf(p_index, sf_index):
    # 合成某一件半成品，p_index为零件索引（列表），sf_index为半成品索引（数值）
    global revenue, cost
    if len(P[p_index[0]]) == 0:
        return
```

```
    for i in p_index:
        if D[i]:
            PT[i] = 1
            cost += para_p[i][2] * len(P[i])
            P[i] = P[i][P[i] == 0]
        D[i] = 1 - PT[i]
    num = min([len(P[i]) for i in p_index])
    for i in p_index:
        P[i] = P[i][:num]
    sf_temp = ((sum([P[i] for i in p_index]) + np.random.rand(num)) > 1 -
        para_sf[sf_index][0]).astype(int)
    cost += para_sf[sf_index][1] * num
    if D[len(para_p) + sf_index * 2]:
        PT[len(para_p) + sf_index] = 1
        cost += para_sf[sf_index][2] * len(sf_temp)
        for i in p_index:
            P[i] = P[i][sf_temp == 1]
        sf_temp = sf_temp[sf_temp == 0]
        SF[sf_index] = np.concatenate([SF[sf_index], sf_temp])
        if D[len(para_p) + sf_index * 2 + 1]:
            cost += para_sf[sf_index][3] * len(P[p_index[0]])
            make_sf(p_index, sf_index)
    else:
        SF[sf_index] = np.concatenate([SF[sf_index], sf_temp])

def make_f():
    # 合成成品
    global revenue, cost
    num = min([len(sf) for sf in SF])
    if num == 0:
        return
    for i in range(len(SF)):
        SF[i] = SF[i][:num]
    F = ((sum([SF[i] for i in range(len(SF))]) + np.random.rand(num)) > 1 -
        para_f[0]).astype(int)
    revenue += para_f[4] * len(F[F == 0])
    cost += para_f[1] * len(F)
    if D[-2]:
        cost += para_f[2] * len(F)
    else:
        cost += para_f[5] * len(F) * F.mean()
    if D[-1]:
        cost += para_f[3] * F.sum()
        for sf_index in range(len(SF)):
            SF[sf_index] = SF[sf_index][F == 1]
            if 1 - PT[len(para_p) + sf_index]:
                PT[len(para_p) + sf_index] = 1
```

```
            cost += para_sf[sf_index][2] * len(SF[sf_index])
            SF[sf_index] = SF[sf_index][SF[sf_index] == 0]
        make_f()

# 根据给定决策方案，模拟平均利润率和三倍标准差

D = [0] * 8 + [1] * 8
profit_rate = []
for _ in range(1000):
    para_p = para_P.copy()
    para_sf = para_SF.copy()
    para_f = para_F.copy()
    for i in range(len(para_p)):
        para_p[i][0] = beta.rvs(100, 900, size=1)[0]
    for i in range(len(para_sf)):
        para_sf[i][0] = beta.rvs(100, 900, size=1)[0]
    para_f[0] = beta.rvs(100, 900, size=1)[0]

    P = []                                                  #
        所有零件正/次品序列（二维数组，次品为1）
    SF = [np.array([])] * len(para_sf)                      # 初始化半成品序列
    for i in range(len(para_p)):
        if D[i]:
            P.append(np.random.binomial(n=1, p=para_p[i][0],
                size=int(num_of_p/(1-para_p[i][0]))))
        else:
            P.append(np.random.binomial(n=1, p=para_p[i][0], size=num_of_p))
    PT = np.zeros((len(para_p) + len(para_sf)))             #
        零件/半成品是否已被检测过
    revenue = 0                                             # 初始化收入
    cost = sum([para_p[i][1] * len(P[i]) for i in range(len(para_p))]) # 购买零件的费用
    make_sf([0, 1, 2], 0)                                   # 生产半成品
    make_sf([3, 4, 5], 1)
    make_sf([6, 7], 2)
    make_f()                                                # 生产成品
    profit_rate.append((revenue - cost) / revenue)
profit_rate = np.array(profit_rate)
profit_rate.mean(), 3 * profit_rate.std()
```