基于动态规划的多工序质量检验策略问题

摘　要

本问针对多工序质量检测策略问题，建立了一次计数抽样检测优化模型，解决了企业抽样检测方案制定问题；建立了动态规划模型，解决了企业检测策略问题；建立了不确定次品率的动态规划模型，解决了抽样检测后次品率不确定时企业的检测策略问题。

针对问题一,建立了一次计数抽样检测优化模型,解决了企业抽样检测方案制定问题。建立一次计数抽样检测优化模型解决抽样检测的效度问题,以生产方风险为 10%、使用方风险 5%、合格质量水平 5%和不合格质量水平 15%为初始值，抽样特性曲线（简称 OC 曲线）贴近理想曲线为优化目标,再通过约束函数值约束曲线图像大致形状。通过遍历法可以计算出最优方案为（*a,b*），*a* 代表样本容量，*b* 代表合格判定数，当批量为 300 时最优方案为（72，6），接着通过蒙特卡罗模拟，计算出当次品率为 0.06 时，方案可以在 90%信度下认定零配件次品率不超过标称值,可以接收这批零件。次品率高于 0.15 时,方案可以在 95%信度下认定零配件次品率超过标称值，拒收这批零件。最后多次改变批量数求方案，可以发现批量数越大,方案逐渐会收敛,当批量数大于 1000 时，方案会收敛至（92，8）。

针对问题二,建立了动态规划模型,解决了企业检测策略问题。首先令状态变量为合格成品和不合格成品数量。然后建立企业利润为阶段指标函数，接着通过是否检测零件 1 或 2,是否检测成品,是否拆解不合格成品四个决策变量进行遍历计算。对于不合格零件之间装配的概率问题，本问考虑最坏的情况，力求最坏情况下也能使企业盈利。最后计算出第 6 种情况在 100 个零件下什么都不检测，也不拆解不合格品,最大盈利为 1810 元（其余答案见表 1）。然后利用蒙特卡罗进行大量随机决策,第六种情况利润最大值也是 1810 元，说明遍历算法并无遗漏最大值的情况。最后利用粗糙集理论对结果分析可以发现当次品率高时不进行充分检测，会导致利润下降，当次品率低时检测过多也会时利润下降。当检测等级与次品率等级相近的时候利润最大。其中次品率高但检测过少对利润的负面影响最大。

针对问题三，建立了动态规划模型，解决了复杂企业检测策略问题。首先令合格品的半成品和成品数量以及不合格的半成品和成品数量作为状态变量，写出状态转移方程，以利润作为阶段指标函数，以优化利润最大为目标，将是否检测零件、是否检测半成品、是否检测成品、是否拆解不合格半成品、是否拆解不合格成品作为决策变量。在零件数量均为 100 情况下，估算计算复杂度后利用蒙特卡罗模拟法进行十万次计算，最终最大利润值收敛于 5920 元。对半成品和成品的次品率进行灵敏度分析，可以发现零配件次品率对利润的影响较小，半成品的次品率对利润的影响比零配件的次品率更大。

针对问题四，建立了不确定次品率的动态规划模型，解决了抽样检测后次品率不确定时企业的检测策略问题。将第一问抽检情况进行模拟，利用不同次品率的分布情况计算次品率概率。代入问题二和问题三的模型，最后计算出问题 2 情况 6 利润最大值为 2258 元。

关键词：动态规划；蒙特卡罗模拟；OC 曲线；粗糙集理论

一、问题重述

1.1 问题背景

在当今竞争激烈的市场环境中，企业生产过程中的工序质量检测点设置的决策至关重要[1]。以某企业生产畅销电子产品为例，该产品需要两种零配件（零配件 1 和零配件 2），经装配后成为成品。然而，产品质量的把控是一个关键问题，因为在装配的成品中，只要有一个零配件不合格，成品就必定不合格；即使两个零配件均合格，成品也未必合格。对于不合格的成品，企业需要做出决策，是选择报废，还是进行拆解，而拆解过程虽然不会损坏零配件，但会产生一定的拆解费用。

1.2 需要解决的问题

（1）供应商对零配件次品率的标称值可能存在不确定性，企业需要通过检测次数尽可能少的抽样检测方案来做出准确判断。

（2）企业生产过程的多个关键阶段决策，这些决策直接关乎企业的成本控制和效益提升。请按照四个决策：1.两个零件是否分别检测。2.成品是否检测。3.不合格成品是否拆解，如果拆解则重复前两个步骤。4.对于用户购买的次品需要调换，然后对不合格品重复第三个决策。

（3）将问题的复杂性进一步提升，当涉及到 m 道工序和 n 个零配件时，决策的难度大幅增加。在这种情况下，如何制定最优的生产过程决策方案。

（4）将第一问中的抽样检测方法运用到零配件、半成品、成品的次品率计算中，然后对于第 2 问和第 3 问的情况重新计算。

二、问题分析

问题一要求设计在一定信度下设计最经济的抽样检测方案，问题二要求在不同的六种情况中计算最大利润，问题三在问题二的基础上多加了 1 道工序和 6 个零配件，属于问题二的更复杂情况。问题四则要求将问题二和问题三中次品率更换为问题一中抽样检测的结果，因此需要将问题一的结果和问题二以及问题三结合起来。

2.1 问题一分析

问题一属于抽样检测方案设计问题，一般有简单随机抽样，分层抽样，整群抽样等等。本问需要检测零件的次品率，零件生产后一般在检测前无特殊分类，因此也就不适用于分层抽样，系统抽样等按照特性分类的抽样方法。并且简单随机抽样操作简单，易于计算，抽样方法较为有效，从经济的角度考虑，简单随机抽样的成本也是最低的。在本文中使用简单随机抽样，就是在一批零件中选出一个小样本，然后根据小样本中次品个数来估算整体的次品率。这个过程涉及两方面问题，信度和效度。效度指的是测量时的准确性和真实性，而信度指的是检测方案的可靠性，一致性和稳定性，一般通过多次测量来进行分析。对于效度，本文引入抽样特性函数和抽样特性曲线进行分析，通过对曲线突变部分倾斜度增大的优化，进行方案的效度分析和最优方案的寻找。最后利用蒙特卡罗算法对计算

出来的优良方案的信度进行模拟计算，寻找符合题目要求的抽检方案。由于题目未给出零件批次总量，因此可以多次改变零件总量，分析不同总量对抽检方案的影响。

2.2 问题二分析

问题二属于决策类问题，通常可以使用目标规划、整数规划、动态规划等方式进行求解。对于本问中四个决策，是否检测零件会对成品的次品率有影响，进而影响成品是否需要检测的决策。是否拆解不合格品会重新进行检测，装配，拆解的决策，这是一个不同决策间前后影响，且可能存在多轮决策的问题。动态规划这个方法在每段决策时，不仅考虑本阶段最优和不同阶段之间的影响，还能同时会考虑到每个阶段的决策对总目标的影响，十分适合解决多轮重复决策问题。所以本问在此选择动态规划，既能利用状态转移方程表示多轮决策之间的影响，还可以求解时避免陷入局部最优解而错过全局最优解。本问是一个典型的多阶段加工问题[1]。将状态变量设置为合格成品数和不合格成品数，写出状态变量的状态转移方程，然后利用状态变量可以写出企业利润作为阶段指标函数，通过是否拆解这个决策变量，能够将加工转变为多轮情况，通过计算不同轮数的利润，可以得到企业决策的最优方案。接着利用蒙特卡罗算法将不同决策进行检验，可以比较出计算结果是否是最优解。最后可以通过粗糙集理论，对决策和零件、成品性质进行分析，总结出不同价格、质量的零件更加倾向于什么样的决策。

2.3 问题三分析

问题三属于复杂决策类问题，通常可以使用，目标规划，整数规划，动态规划，蒙特卡罗模拟等算法进行求解。根据问题二的结果，我们只考虑最多拆解一次的情况。对于本问中 20 个决策，零配件，半成品，成品每轮是否检验。半成品，成品检验后是否进行拆解。半成品，成品检验后是拆解后是否还需要对组装的成品进行检验进行模拟。蒙特卡罗模拟能处理复杂问题，提供随机样本的近似解，适用于不确定性分析，本题相对于第二问明显复杂许多，如果继续采用动态规划，通过状态转移方程进行全过程的连接，以及遍历所有的决策计算量过大以及在第二问的检验分析中我们发现蒙特卡罗的效果很好，所以这题我们打算应用蒙特卡罗算法给出相应的决策方案，以企业利润的最大为目标，不断迭代最优值，并保留利润最大时的最优决策。

2.4 问题四分析

问题四属于前三问结合的问题，利用第一问计算出的抽检方案。本文可以通过前两问模拟供应商供应的零件的次品率大概为 5%~20%，通过随机取数，再利用抽样方案进行检验，查看能接收的零件次品率分布情况，根据分布情况将次品率分别代入到第二问和第三问中的模型，计算出方案，根据不同次品率的出现的频数，可以计算出某个方案成为最优方案的概率，找出概率最大的方案作为企业的在这种抽检方案下的最优生产策略，就能尽可能让企业盈利。

三、基本假设

1.假设零件的检验是无损检验，不会对后续生产有影响。

2.假设生产的合格产品都能卖出。
3.假设企业检测时不存在错检和漏检。

四、符号说明

符号	意义
a	样本容量
b	设定的合格边界数
c	实际检验不合格零件个数
N	本批零件总个数
m	不合格率
α	供应商风险
β	使用方风险

其余公式符号在公式前后均有说明。

五、模型建立与求解

5.1 问题一的建模与求解

进货检验是企业质量控制的关键步骤。因为企业需要承担供应商零件的检测费，所以尽可能少的检测次数和抽样的样本量可以为企业减少成本，但抽样的样本量过低会导致抽样方案信度过低，企业接收或拒绝零件批次的风险较大，可能会影响生产质量问题。为了尽可能减小企业成本又能保证抽样检验方案具有一定的可信度，本文选择使用一次计数检验通过寻找最经济的方案，使得企业在不同信度下均可以设计出最小样本量的抽样检测方案。针对抽样检验方案效度问题，引入抽样特性函数和抽样特性曲线进行方案合理性分析。对于抽样检测方案信度问题，使用蒙特卡罗模拟，计算方案错误的概率，从而得出方案的信度。

5.1.1 模型建立

一次计数检验方案就是在一批数量为N的零件 1 或零件 2 中，随机抽取一个样本量为a的样本进行检验。通过样本中抽出的不合格零件个数，来推测总体零件是否符合企业所声称的标称值。对于抽样检测方案的制定就是确定样本容量a和合格边界数b，然后进行实际抽样，得出实际不合格零件个数c。如果$c \leqslant b$，意味着企业可以认同标称值并接收这批零件，反之，$c \geqslant b$，意味着企业需要提出质量问题并拒收零件。

建立一次计数抽样检测优化模型解决抽样检测的效度问题

一次抽样检测，对于批量为N 的零件来说，其抽样特性函数可以利用超几何分布进行计算[2]

$$L(m)=\sum_{c=0}^{b}\frac{\binom{N\cdot m}{c}\binom{N(1-m)}{a-c}}{\binom{N}{a}} \tag{1}$$

式子实际含义可以解释为企业接收这批零件的概率[3]，其中m表示这批零件中实际次品率。如果将令m从 0%取值到 100%，就可以画出公式（1）的函数图像，也就是抽样特性曲线（简称 OC 曲线）。

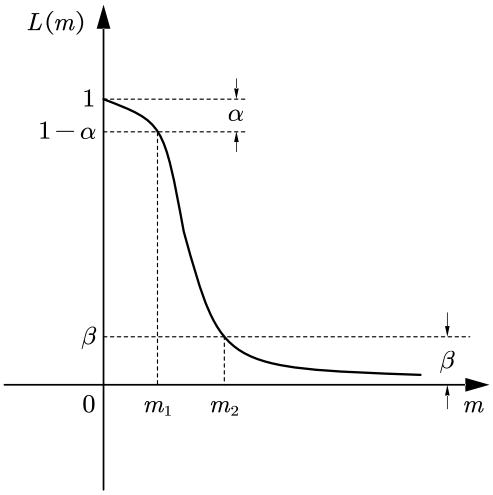


图 1 OC 曲线图像

图 1 中可以看出当供应商提供零件的次品率不断增大时，企业接收概率也不断在降低，这是符合实际情况的。图中m_1表示合格质量水平，即企业较为能接受的次品率的最大值，也就是说在次品率小于m_1的部分企业都是大概率会接收且乐于接收的。m_2表示不合格质量水平，即企业不能接收的次品率的最小值，也就是说在m_2右侧的部分企业完全不能接受如此高的次品率是尽可能不接收这批零件。而在两者之间，企业虽然不希望接收较高次品率的零件，但是可能因为检测方案的原因导致接收了这批零件。因此通过上述的叙述可以看出，如果检测方案优良，那么m_1和m_2之间的距离应该尽可能短，这样次品率一旦高于设定值，就能迅速拒绝这批零件。那么图中α表示此时次品率企业应当接收这批零件而拒绝了这样造成的损失是供应商会卖不出这批零件，所以称为供应商风险。图中β是企业本应该因为次品率过高拒绝这批零件，但是错误的接收了零件，这样企业支付了正常的价格却接收了一批次品率高的零件，企业会有较高损失，因此可以称为使用方风险。

从图 1 中可以看出供应商风险和使用方风险计算公式为

$$\begin{cases}\alpha=1-L(m_1)=1-\sum_{c=0}^{b}\dfrac{\binom{N\cdot m_1}{c}\binom{N(1-m_1)}{a-c}}{\binom{N}{a}}\\ \beta=L(m_2)=\sum_{c=0}^{b}\dfrac{\binom{N\cdot m_2}{c}\binom{N(1-m_2)}{a-c}}{\binom{N}{a}}\end{cases} \tag{2}$$

可以看出上述方程十分复杂，而且α，β，m_1，m_2四个值相互之间有一定的关联，求解十分困难。而且针对不同的方案（a，b），其 OC 曲线的倾斜程度不一样，导致方案优度也不一样。根据以上提出的问题，如果利用公式（2）简单的通过遍历方案进行求解然后进行方案优度比较，显然情况复杂，不利于求解。因此通过查阅文献，本文选择使用单目标优化，设定约束条件进行最佳方案求解。

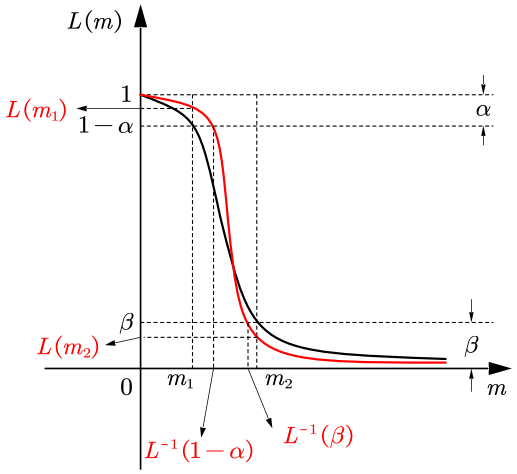


图 2 优化趋势示意图

如图 2 所示，红色曲线是本文希望的优化出的理想曲线的趋势，即中间突变情况函数斜率更加陡峭，因此可以得出优化函数为

$$\min\ L^{-1}(\beta)-L^{-1}(1-\alpha) \tag{3}$$

其中$L^{-1}(m)$是抽样特性函数$L(m)$的反函数。但是仅凭优化目标进行优化，可能会导致间距可能减小，但是斜率不怎么变化，因此可以约束m_1，m_2处的函数值使得函数尽量维持 Z 字型

$$\begin{cases} L(m_1)\geq 1-\alpha \\ L(m_2)\leq \beta \end{cases} \tag{4}$$

综上所述，一次抽样检验优化模型为

$$s.t.\begin{cases} L(m_1)\geq 1-\alpha \\ L(m_2)\leq \beta \end{cases} \quad \min\ L^{-1}(\beta)-L^{-1}(1-\alpha) \tag{5}$$

针对公式（5）进行方案遍历求解，就可以得出不同情况下的最优方案。

蒙特卡罗检验计算优化方案的信度

上述优化模型，只能解决抽检方案的效度问题，使得抽检的样本量尽可能少且效度尽可能高。而信度一般指测量数据的一致性，稳定性和可靠性，也就是说这个方案在使用时，可信度高不高，会不会出现使用 10 次中有 8 次都是存在一定的错误导致误判。而针对信度的计算一般是通过重复测量，检查误判次数衡量信度，因此本文选择使用蒙特卡罗检验，模拟抽样检测过程，进行抽样检测，计算误判率，从而得到信度。

5.1.2 模型求解

求解抽样检测方案，需要先确定α，β，m_1，m_2四个值，其中α，β是供应商

风险和使用方风险。供应商风险是指企业应该接收却误判，然后拒收，这对供应商是巨大损失，但同时对企业来说，花费了检测费用但却拒收了合格的零件，可能会耽误生产等，对企业来说也是一种损失，因此α值不能过大。而β是使用方损失，是企业不应该接收而误判，将不合格零件批次接收，对于企业的生产损失相对于供应商是巨大的，因为企业会比之前的情况多支付零件的费用。因此需要尽可能减少β值，对于α值可以稍微放宽要求，避免造成没有解的情况。通过查阅文献可以得到α，β一般取值为 10%和 5%[3]，因此本问取α为 10%，β为 5%。

对于m_1，m_2的取值，供应商提供的标称值为 10%，意味着企业要求的很有可能就是 10%的次品率，那么当零件次品率为 5%时，应该尽可能接收零件，当零件次品率为 15%的时候尽可能拒收零件。通过问题二中的情况可以看出，企业能够接受 20%次品率的零件，所以本问将m_1设置为 5%，将m_2设置为 15%，将理想曲线的突变区域尽可能优化到次品率为 10%的情况，这样使得次品率小于 10%能够选择接收零件，大于 10%就能更快拒绝零件。

遍历计算时确定一个b值，然后对容量值a遍历，可以得到不同方案的特征曲线，判断是否满足约束条件，接着计算目标函数，即可比较出方案的优劣。

5.1.3 问题结论

将样本量假设为 300，将不同遍历情况的α值和β值绘制出来。

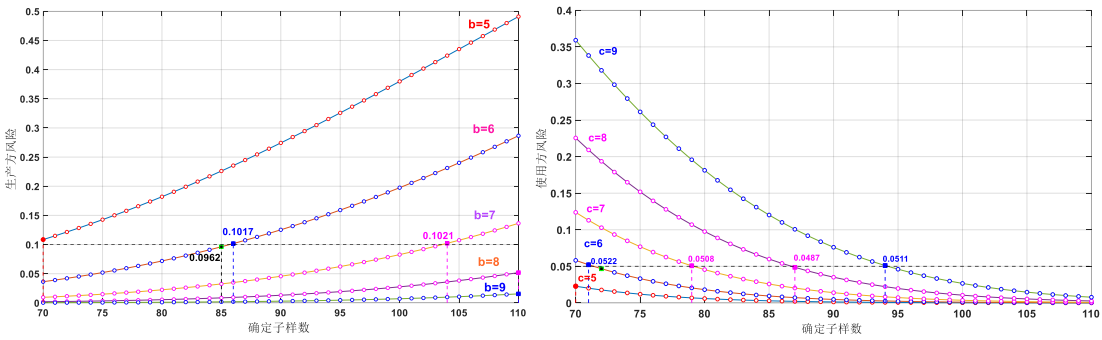


图 3 不同方案下的α值（左图）和β值（右图）

可以从图 3 中看出，当b固定时，样本量越小，α值越小，β值反而增大。随着b不断增大，α值曲线的斜率不断降低，上升的速度逐渐降低；β值曲线的斜率不断增大，下降速度变快。

因为要求α小于 0.1，β要小于 0.05，满足这两个条件的方案中，目标函数值最小的方案为（72，6），即抽取一个样本容量为 72 的小样本，如果其中次品个数小于等于 6 个，就可以接收零件，反之若次品数大于 6 个，则拒绝这批零件。

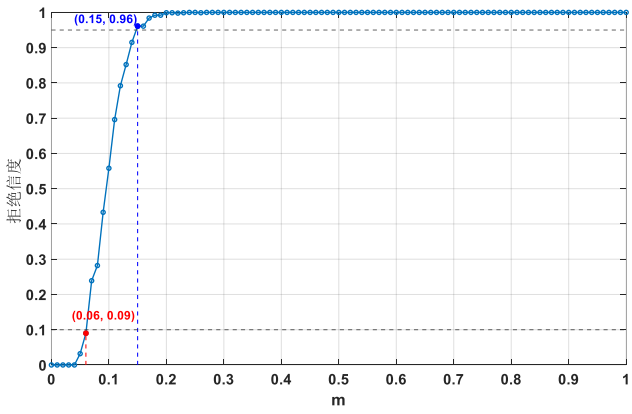


图 4 蒙特卡罗模拟图

将方案代入蒙特卡洛模拟，假设供应商不同次品率的零件每种提供了 1000 次，计算企业拒绝接收货物的频率。当企业提供次品率为 0.06 的货物时，企业接收的概率为 0.91，此时可以在 90%信度下认定零配件次品率不超过标称值，可以接收这批零件。同理，当零配件实际次品率高于 0.15 时，企业拒绝概率为 96%，因此可以认为在 95%信度下认定零配件次品率超过标称值，拒收这批零件。

可以从图 4 中看出，本文提供的方案，当供应商提供的零件次品率高于 10%时，企业接收的可能性低于 50%。当供应商提供的零件次品率低于 0.06 时，有 90%以上的概率企业会接收零件。当供应商提供的零件次品率高于 15%时，企业拒绝这批零件的概率至少为 95%。从上述内容可以发现方案在次品率为 10%附近突变，说明方案对于 10%次品率附近十分敏感。从 6%次品率到 15%次品率，拒绝率上升了 87%，证明方案能够十分合理有效。

5.1.4 检验分析

通过将批量值a进行上下波动，可以得出最优方案波动为

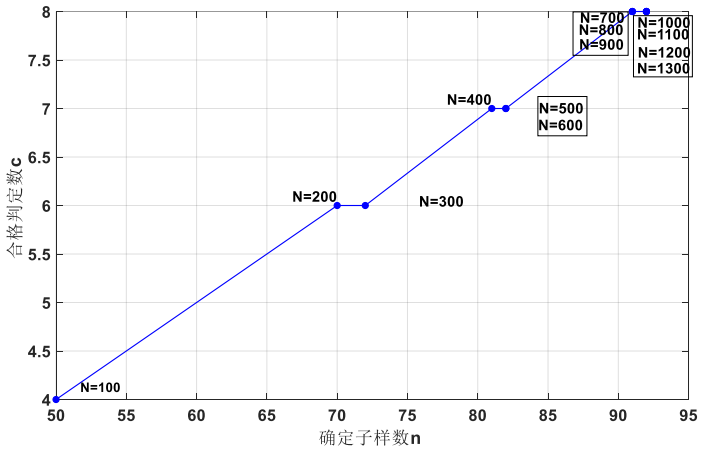


图 5 问题一灵敏度分析

可以看出当 N 变大后，计算出的最优方案逐渐和 N 无关，即方案逐渐收敛至（92，8）的抽检方案。这是可以解释的，因为经过查阅文献可以知道当N逐渐变大后，可能会满足$N>10a$的关系，而此时批次量N对抽样特性函数几乎没有影响，可以把抽样检验看成返回抽检，抽样特性函数可以近似为二项分布计算[2]，即

$$L(m)=\sum_{c=0}^{b}\frac{\binom{N\cdot m}{c}\binom{N(1-m)}{a-c}}{\binom{N}{a}}\approx\sum_{c=0}^{b}\binom{a}{c}m^{c}(1-m)^{a-c}\qquad N>10a$$

从图中可以看出当批次量超过 1000 后，抽样方案就固定在（92，8）处。

5.1.5 小结

本问利用计数抽样检测优化模型求出批次量在 300 时的最优方案为(72,6)，然后可以通过蒙特卡罗模拟计算出当次品率小于 6%时，企业可以在 90%信度下认定零配件次品率不超过标称值，可以接收这批零件。当次品率大于 15%时，企业可以在 95%信度下认定零配件次品率超过标称值，拒收这批零件。通过灵敏度

分析，不断增大 N 值，可以发现方案逐渐收敛至固定值，尤其当 N 大于 1000 后，方案将收敛于（92，8）处。这是可以解释的，因为当 N 增大到 10 倍的样本容量后，抽样检测可以近似看成放回抽样，抽样特性函数可以近似用二项分布进行计算。

5.2 问题二的建模与求解

本问是决策问题，具有是否检测两个零件，是否检测成品，是否拆解不合格成品四个决策步骤。决策相互之间有一定的影响，且可能存在多轮生产过程，因此本问考虑使用动态规划模型，以合格成品数量和不合格成品数量作为动态变量建立动态转移方程，以利润作为阶段指标函数，以四个决策作为决策变量，优化利润最大作为目标。通过遍历的方法进行计算，最终得出答案。利用蒙特卡罗检验遍历是否有遗漏的更大利润的情况，并使用粗糙集理论进行结果分析。

5.2.1 模型建立

在开始建模之前，需要针对第二问中不确定性的东西进行讨论。题干中提到即使是两个合格零件装配，也不一定能拼出合格的成品，意味着零件拼装成成品是存在失败概率的。假设现在是题干中的第一种情况，零件 1 的次品率是 10%，零件 2 的次品率是 10%，成品的次品率是 10%。如果两个零件都不检测，那么理论上因为次品零件造成成品为次品的件数应该占 10%~20%（波动是因为零件次品配合格品，合格品配合格品，次品配次品的不同情况，10%的情况是所有次品零件都恰好配次品零件）。但是这样情况下不满足成品次品率为 10%，即使所有次品零件都相互配对，那么会造成成品中所有的次品都来自于次品零件，进而可以推出本次合格品零件都装配成合格的成品，与题目条件“两个零配件均合格，装配出的成品也不一定合格”相矛盾。因此本文将题目中成品的次品率理解为两个合格零件装配成成品是次品的概率，这样就不会存在上述的矛盾。

对于如果第一步和第二步不检测零件，那么在装配成品的时候会造成次品和合格品零件配对的排列组合问题。本文认为企业的主要目的是盈利，那么就要考虑最坏情况对企业的影响。而本问中最坏的情况是次品都配合格品零件的情况。

根据题干所述，需要企业做出的三项决策分别为 1.是否检测零件 1 或者零件 2 的合格情况。2.是否检测成品的合格情况，如果不检测则会使用户退回不合格产品产生额外损失。3.是否拆解不合格产品。

对于零件 1 和零件 2 是否检测合格情况设置决策变量为

$$ck_1=\begin{cases}0 & \text{不检测零件}1\\ 1 & \text{检测零件}1\end{cases}$$
$$ck_2=\begin{cases}0 & \text{不检测零件}2\\ 1 & \text{检测零件}2\end{cases} \tag{6}$$

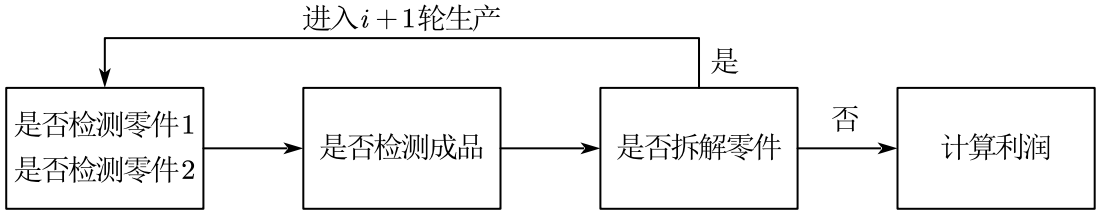
对于产品是否合格的检测，设置决策变量为

$$k_1=\begin{cases}0 & \text{不检测}\\ 1 & \text{检测}\end{cases} \tag{7}$$

针对是否拆解不合格产品，设置决策变量为

$$k_2=\begin{cases}0 & \text{不拆解}\\ 1 & \text{拆解}\end{cases} \tag{8}$$

可以看出本问的加工工序仅有一道，但是拆解零件再加工成成品，就会变成多工序问题。



第i轮生产流程

图 6 第i轮生产流程设计图

如图 5 所示，本文将流程设计为i轮，如果将零件进行拆解，将进入下一轮生产，而生产的原材料仅有上一轮拆卸的零件。如果不拆解零件，则不存在下一轮生产。

根据是否检测成品合格情况和是否拆解不合格成品可以将情况分为 4 类，分别为检测不拆解，检测并拆解，不检测不拆解和不检测并拆解。

如果令H_i表示第i轮生产的合格品数量，B_i为第i轮生产的不合格品数量，则可以将四类的状态转移方程和阶段指标函数$V_i(k_1,k_2,H_i,B_i)$表示出来，其中阶段指标函数表示在不同的k_1,k_2,H_i,B_i情况下，企业能获得的利润。

- 检测成品合格情况，不拆解不合格品时

状态转移方程即第i轮生产的结果进入第$i+1$轮后所处的状态的数学描述。对于合格品数量和不合格品数量来说，检测成品是否合格对数量无影响。而不拆解的操作，使得上一轮到下一轮的成品和不合格品数量无变化，因此可以得到状态转移方程为

$$\begin{cases}H_{i+1}=H_i\\ B_{i+1}=B_i\end{cases}\qquad k_1=1,k_2=0 \tag{9}$$

对于阶段指标函数来说，检测成品合格情况直接影响了是否产生调换的损失。假设有 100 件成品，其中有 90 件合格品，如果进行成品检测，那么就只会有 90 件合格品的利润，次品不产生利润。如果不进行成品检测，那么在同样产生 90 件合格品的利润的情况下，还需要承担调换货物的损失（包括一些企业信誉损失）。但是成品检测是有成本的，还要将检测成品的费用加入到阶段指标函数中。而不合格品是否拆解，也影响着企业是否需要支付拆解的成本。

阶段指标函数代表着不同状态和决策下，企业的利润，那么可以得到公式为

$$V_i(k_1,k_2,H_i,B_i)=$$

$$H_i p-(H_i+B_i)(cp_3+p_3)-n_1p_1-n_2p_2-\frac{H_i}{1-\eta_1}ck_1cp_1-\frac{H_i}{1-\eta_2}ck_2cp_2 \tag{10}$$

其中p（price）代表着成品的价格，而p_1，p_2，p_3分别代表着零件 1 的价格，零件 2 的价格，成品的装配价格。cp（check price）代表着检查价格，cp_1，cp_2，cp_3则分别代表着零件 1 检测的价格，零件 2 检测的价格，成品的检测价格。n代表着本轮生产中的零件数量（number），n_1代表零件 1 的数量，n_2代表零件 2 的数量。

ck（check）代表是否检查零件，是一个决策变量，其值为

$$ck_j=\begin{cases}0 & \text{不检查零件}j\\ 1 & \text{检查零件}j\end{cases} \tag{11}$$

η_i'则表示在本轮中，成品中的次品占所有成品的比例。在本题中，零件的装配是随机的，按照本问开头所提到的，考虑企业最坏的情况，那么就是零件 1 和零件 2 的次品都与合格品配对，在此基础上还产生了两个合格品配成的次品。那么计算公式为

$$\eta_1'=\eta_0+(1-ch_1)\eta_1+(1-ch_2)\eta_2 \tag{12}$$

其中ch表示是否检查，η_0，η_1，η_2分别表示两个合格品装配成次品的概率，零件 1 的次品率，零件 2 的次品率。

- 不检测成品合格情况，不拆解不合格品时

对于检测成品，依然对本轮成品数量H_i和本轮次品成品数量B_i无影响。而不拆解也不改变这两个变量的值。因此这种情况下的状态转移方程为

$$\begin{cases}H_{i+1}=H_i\\ B_{i+1}=B_i\end{cases}\qquad k_1=0,k_2=0 \tag{13}$$

对于阶段指标函数，在现在这种情况下，没有进行成品检测，那么就不需要加入成品检测的费用，但是要加入因为次品造成的调换费用。根据以上分析可以写出阶段指标函数为

$V_i(k_1,k_2,H_i,B_i)=$

$$H_ip-(H_i+B_i)p_3-n_1p_1-n_2p_2-\frac{H_i}{1-\eta_1}ck_1cp_1-\frac{H_i}{1-\eta_2}ck_2cp_2-B_ich \tag{14}$$

其中ch（change）表示调换一个次品的概率。

- 不检测成品合格情况，拆解不合格品时

拆解不合格品会造成下一轮的合格品和不合格品数量的变化，如果拆解就会进入下一轮生产，对于已经检验过的零件就不需要再进行检验，未检验的零件可以选择是否检验，首先对于状态转移方程，在这种情况下变为

$$\begin{cases}H_{i+1}=H_i+(1-\eta_i')B_i\\ B_{i+1}=\eta_i'B_i\end{cases}\qquad k_1=0,k_2=1 \tag{15}$$

公式中的η_i'表示第i轮的次品成品占所有成品的比例，在公式（12）中有计算第一轮η_1'的值，后续每轮的比例可以运用如下公式计算

$$\eta_{i+1}'=\frac{(1-ch_1)\eta_1+(1-ch_2)\eta_2}{\eta_i'}+\eta_0 \tag{16}$$

其中ch表示是否检查，是决策变量，η_0，η_1，η_2分别表示两个合格品装配成次品的概率，零件 1 的次品率，零件 2 的次品率。

对于阶段指标函数，本轮不检测成品，要加入调换费用。因为拆解了不合格成品，所以要加入拆解的费用。那么阶段指标函数为

$V_i(k_1,k_2,H_i,B_i)=$

$$H_ip-(H_i+B_i)p_3-n_1p_1-n_2p_2-\frac{H_i}{1-\eta_1}ck_1cp_1-\frac{H_i}{1-\eta_2}ck_2cp_2-B_ich+B_idp \tag{17}$$

其中dp（disassemble）表示拆解 1 个零件的费用。

- 检测成品合格情况，拆解不合格品时

拆解依然会造成合格品和不合格品数量变化，所以这种情况下状态转移方程为

$$\begin{cases} H_{i+1}=H_i+(1-\eta_i')B_i \\ B_{i+1}=\eta_i'B_i \end{cases} \qquad k_1=1,k_2=1 \tag{18}$$

检测会有检测成本，而不存在调换货物的成本，拆解货物有拆解的成本，将这些考虑进阶段指标函数可得

$V_i(k_1,k_2,H_i,B_i)=$

$$H_ip-(H_i+B_i)(cp_3+p_3)-n_1p_1-n_2p_2-\frac{H_i}{1-\eta_1}ck_1cp_1-\frac{H_i}{1-\eta_2}ck_2cp_2+B_idp \tag{19}$$

综上所述

上述所有情况可以总结出动态规划模型的决策变量为

$$\begin{cases} ch_1 & \text{是否检测零件}1 \\ ch_2 & \text{是否检测零件}2 \\ k_1 & \text{是否检测成品} \\ k_2 & \text{是否拆解不合格成品} \end{cases} \tag{20}$$

状态转移方程为

$$\begin{cases} H_{i+1}=H_i \\ B_{i+1}=B_i \end{cases} \qquad k_1=1\text{或}0,k_2=0$$

$$\begin{cases} H_{i+1}=H_i+(1-\eta_i')B_i \\ B_{i+1}=\eta_i'B_i \end{cases} \qquad k_1=1\text{或}0,k_2=1 \tag{21}$$

阶段指标函数为

$V_i(k_1,k_2,H_i,B_i)=$

$$\begin{cases} H_ip-(H_i+B_i)(cp_3+p_3)-n_1p_1-n_2p_2-\frac{H_i}{1-\eta_1}ck_1cp_1-\frac{H_i}{1-\eta_2}ck_2cp_2 \\ H_ip-(H_i+B_i)p_3-n_1p_1-n_2p_2-\frac{H_i}{1-\eta_1}ck_1cp_1-\frac{H_i}{1-\eta_2}ck_2cp_2-B_ich \\ H_ip-(H_i+B_i)p_3-n_1p_1-n_2p_2-\frac{H_i}{1-\eta_1}ck_1cp_1-\frac{H_i}{1-\eta_2}ck_2cp_2-B_ich+B_idp \\ H_ip-(H_i+B_i)(cp_3+p_3)-n_1p_1-n_2p_2-\frac{H_i}{1-\eta_1}ck_1cp_1-\frac{H_i}{1-\eta_2}ck_2cp_2+B_idp \end{cases} \tag{22}$$

目标函数为利润最大

$$f(i)=\max\{V_i+f(i-1)\} \tag{23}$$

通过遍历的方法可以进行模型的求解。

5.2.2 模型求解

Step1:参数初始化和变量设置

在 main()函数的开始，代码初始化了一系列参数和变量。这包括零件和成品

的次品率(eta1,eta2,eta0)、各种成本(p1,p2,cp1,cp2,cp3,price3,change_cost,break_cost)、初始零件数量(n1,n2)等。同时，它还初始化了用于追踪最优解的变量，如 f 列表(用于存储每步的最优值)、max_v(最大价值)和 path(最优决策路径)，cheeked1,2 标记变量，保证零件 1,2 最多被检查一次。这一步为整个决策过程奠定了基础，确保后续的计算都基于正确的初始条件。

Step2:决策循环的开始

代码使用一个 while 循环来模拟决策过程，循环次数由 step 变量控制。在每次循环中，代码考虑所有可能的决策组合。这包括是否检查零件 1 和零件 2(check1,check2)，以及是否进行成品检查和处理(k1,k2)。通过多层嵌套的 for 循环，代码系统地探索了所有可能的决策组合。这种穷举方法虽然计算量大，但保证了不会遗漏任何可能的最优解。

Step3:状态更新和初始成本计算

在考虑每种决策组合时，代码首先更新系统状态。它计算经过零件检查后的合格品数量(hk)，并初始化总成本(total_cost)，包括生产成本和原料成本。这一步还处理了零件检查的逻辑，更新了总的次品率。其中第一轮的次品率计算如公式（12）所示，之后每轮的次品率如公式（16）所示这个步骤实现了状态转移方程的一部分，为后续的价值计算做准备。

Step4:调用函数进行详细计算

根据公式（9），公式（12），公式（15），公式（13）更新合格品和不合格品的数量。根据公式（10），公式（14），公式（17），公式（19）计算当前决策的经济价值，考虑了销售收入、各种成本(检验、购买、装配、调换、拆解等)。这两个函数共同实现了数学模型中的状态转移方程和目标函数，是整个优化过程的核心。

Step5:更新最优解

在计算出当前决策组合的价值后，代码将其与当前的最大值 max_v 进行比较。如果新的价值更大，就更新 max_v 并记录对应的决策路径。这个过程保证了我们始终保持着到目前为止的最优解。通过这种方式，代码实现了动态规划的核心思想，即通过解决和记录子问题的最优解来构建全局最优解。

Step6:输出结果

循环结束后，代码输出最大的 f(k)值(即最大利润)和对应的决策路径。f 列表中的最后一个元素就是整个决策过程的最大价值，而 path 数组则记录了达到这个最大价值的决策序列。这个步骤为决策者提供了优化生产过程的具体指导，展示了整个优化过程的最终结果。

5.2.3 问题结论

以两种零件各 100 个为例，通过模型求解可以计算出最大利润和此时的决策为

表 1 第二问结果

情况	第一轮生产				第二轮生产				最大利润
	零件一检验	零件二检验	成品检验	不合格品拆解	零件一检验	零件二检验	成品检验	不合格品拆解	
1	1	1	0	1	0	0	0	0	1650
2	1	1	0	1	0	0	0	0	941
3	1	1	0	1	0	0	0	0	1432

4	1	1	1	1	0	0	0	0	1085
5	0	1	0	1	0	0	0	0	865
6	0	0	0	0	0	0	0	0	1810

从表中可以看出，当零件拆解之后，几乎不做过多的检测和第二次拆解，也就是生产轮数不多，这是符合情况的，因为生产轮数越多，产生的费用就越多，每次装配成品都是有一定失败率的，当零件少了之后，再加上像第六种情况，零件并没有经过检测，选择不拆解、不生产能及时止损，避免死循环和成本上升。

在第二轮生产中，几乎没有任何操作。在前四轮可以解释为之前检验过两个零件，且成品检验成本高于调换成本。因为如果两个零件都检测了，那么拆解的零件都是合格品，第二轮制作的成品大部分都是合格的，不合格的占比很小，可以支付一部分调换费用，避免大量的检测成本。而第 5 种情况，零件 1 的检测价格昂贵且次品率低，因此检测成本高且收益低，而零件 2 的检测成本极低且次品率高，进行检测的收益极高，所以应当零件一不检验零件二检验，这个方案是符合逻辑的。对于第六种情况来说，次品率极低极低，因此不需要进行检测，同时拆解费用极高，不进行拆解，所以第六种情况是什么检测和拆解都不做就能得到极高的利润。同时也可以看出来，在次品率低的情况下利润大概率会比较高。

5.2.4 检验分析

蒙特卡罗模拟法可以实现计算的某个指标或者总体参数的随机试验法，本问通过对决策变量和生产轮数进行随机模拟，可以得到如下图像

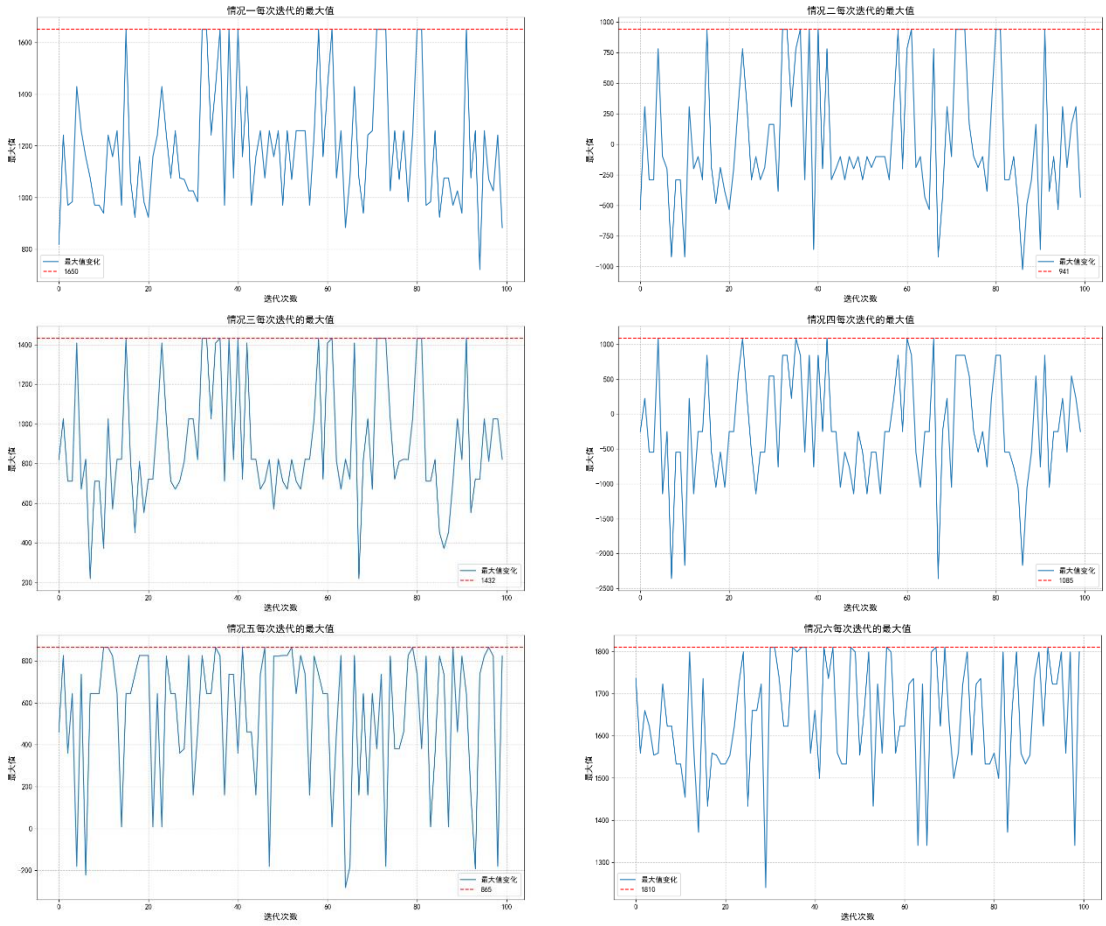


图 7 六种不同情况的蒙特卡罗检验

如图 7 所示，六种情况下进行蒙特卡罗检验所能得到的最大值，都与本问求

解情况相同，可以一定程度上说明模型计算时遍历结果较为合理，没有遗漏最大利润的情况，是全局最优解。

粗糙集理论是一种处理不确定性和不精确性的数学工具，其核心思想是通过不可分辨关系来近似描述数据集中的概念和分类。

将零件平均次品率，平均购买单价，平均检测成本，检测情况作为条件属性。将最终利润情况作为决策属性可以得到决策表为

表 2 粗糙集决策表

情况	平均次品率	平均购买单价	平均检测成本	检测情况	拆解情况	最终利润情况
1	2	1	2	2	2	3
2	3	1	2	2	2	1
3	1	1	2	2	2	2
4	2	1	1	3	2	2
5	2	1	3	1	2	1
6	1	1	2	1	1	3

计算出每种情况的各项条件属性数值，然后将其离散化能得到表 2 中的数据。经过检测与修改，决策表是协调的，然后对其进行依赖度计算，删除与决策属性不相关或不重要的属性，对属性进行约简后可得最小约简集为[“平均次品率”，“检测情况”]。然后可以生成 if-then 规则为：

IF 平均次品率 ＝2 ，检测情况 ＝2 THEN 最终利润情况 ＝3
IF 平均次品率 ＝3 ，检测情况 ＝2 THEN 最终利润情况 ＝1
IF 平均次品率 ＝1 ，检测情况 ＝2 THEN 最终利润情况 ＝2
IF 平均次品率 ＝2 ，检测情况 ＝3 THEN 最终利润情况 ＝2
IF 平均次品率 ＝2 ，检测情况 ＝1 THEN 最终利润情况 ＝1
IF 平均次品率 ＝1 ，检测情况 ＝1 THEN 最终利润情况 ＝3

从上述规则可以看出，如果平均次品率的等级与检测情况等级接近，利润往往偏高；如果检测情况大于平均次品率，那么利润是中等的；如果平均次品率等级高于检测情况，那么利润会偏低。是否检测的决策不单单受次品率影响，还会受到检测成本的制约，但是在进行粗糙集分析时，其对依赖度影响较低，并未考虑。

5.2.5 小结

本问使用动态规划将不同决策情况下每个阶段的利润表示出来，通过遍历法进行求解，能够得到最大利润的情况。然后利用蒙特卡罗模拟，检验遍历时是否遗漏了最优情况。通过粗糙集理论对结果进行分析可以发现，当次品率高时不进行充分检测，会导致利润下降。当次品率低时检测过多也会时利润下降。当检测等级与次品率等级相近的时候利润最大。其中次品率高但检测过少对利润的影响最大。

5.3 问题三的建模与求解

5.3.1 模型建立

比较问题二和问题三，问题三基于问题二的基础上不仅增加了配件，也增加了工序。问题二中每一轮有2^4种情况，问题三的中不考虑拆解后进入下一轮的情况依然有2^{12}种情况，通过第二问的检验分析，发现蒙特卡罗模拟与遍历求解动

态规划问题具有相似的效果。由于本问情况十分复杂，遍历求解困难极大，因此决定使用蒙特卡罗模拟的方法求解出最优解。

根据题干所述，需要企业做出的决策主要包含两大类：1.各道工序是否需要对工件进行检验。2.从第二道工序开始是否需要对工件进行拆解。

对于各道工序的工件是否进行检验设置决策变量为

$$ck_{mn}=\begin{cases}0 & \text{不检验第}m\text{道工序第}n\text{个零件}\\ 1 & \text{检验第}m\text{道工序第}n\text{个零件}\end{cases} \tag{24}$$

对于从第二道工序开始的工件是否需要拆解

$$kt_{mn}\begin{cases}0 & \text{不拆解}m\text{道工序的}n\text{工件}\\ 1 & \text{拆解}m\text{道工序的}n\text{工件}\end{cases}\quad m\geqslant 2 \tag{25}$$

如果令H_{mn}表示 m 道工序第 n 件工件中合格产品的数量，B_{mn}表示 m 道工序第 n 件工件中不合格品的数量。令$V_{mn}(H_{mn},B_{mn})$为各个阶段的指标函数，表示在不同H_{mn},B_{mn}企业可以获得的利润。

首先对于各个情况下H_{mn},B_{mn}，的数量进行讨论，便于后续带入$V_{mn}(H_{mn},B_{mn})$求解各个阶段的最优策略。令q_{mn}为 m 道工序下 n 工件的总数量，r_{mn}为 m 道工序下 n 工件的次品率。q_{1n}为各个零部件的开始购买量。

零部件购买阶段(m=1)：本问需要计算各个零部件中合格品的合格品和不合格品数量，便于后续的成本以及利润的统计。

$$\begin{cases}H_{1n}=q_{1n}(1-r_{1n})\\ B_{1n}=q_{1n}r_{1n}\end{cases} \tag{26}$$

在这里，为了表示个零部件用来组装办成品，本文引入$G_{mi}(1<=i<=n)$存储 m-1 工序中传入到 m 阶段工序中的工件数量。

如：第一道工序中的零配件 1,2,3 都是用来合成第二道工序中的半成品 1

$$G_{21}=\{(1-ck_{11})B_{11}+H_{11},(1-ck_{12})B_{12}+H_{12},(1-ck_{13})B_{13}+H_{13}\} \tag{27}$$

半成品组装以及成品组装阶段：在这里需要利用上一轮中，组成半成品零配件的

$$\begin{cases}q_{mn}=min\{G_{\mathrm{mi}}\}\\ H_{\mathrm{mn}}=q_{mn}(1-r_{mn})\ 2\leqslant m\\ B_{mn}=q_{mn}r_{mn}\end{cases} \tag{28}$$

拆解过后再重组会使合格品的数量增加。由于拆解有可能无限循环下去，根据第一问的 6 种情况，可以发现成品最多被拆解一次，即可得到最大的利润。在这里为了简化运算，只考虑成品或者半成品最多拆解一次的情况。

$$\begin{cases}H_{mn}=q_{mn}(1-r_{mn})+q_{mn}r_{mn}(1-r_{mn})\cdot kt_{mn}\\ B_{mn}=q_{mn}r_{mn}-q_{mn}r_{mn}(1-r_{mn})\cdot kt_{mn}\end{cases} \tag{29}$$

通过阶段指标函数，可以计算出各个阶段状况，通过状态转移求解出各个方向的，从而由阶段的最优值求解出整体的最优值。

$$V_{mn}(H_{mn},B_{mn})=H_{mn}\cdot p-total\_cost \tag{30}$$

其中$total_cost$由初始购买价格，各轮检测费用，各轮中拆解造成的费用，调换货物造成的损失几部分组成。

$$total\_cost = \sum_{i=2}^{m}\sum_{j=1}^{n} p_{ij}(H_{ij} + B_{ij}) + (H_{1n} + B_{1n})P_{1n} + \sum_{i=2}^{m}\sum_{j=1}^{n} dp_{ij} \cdot kt_{ij} \cdot B_{ij} + B_{mn} \cdot change\_cost \tag{31}$$

最终计算

$$f = max\,(f, V(H_{mn}, B_{mn})) \tag{32}$$

5.3.2 模型求解

对于半成品，（不考虑组装好不检查，不由成品返回直接拆解的情况）可不检查直接组装，半成品的拆解是在成品有问题退回来后，决定拆解了，拆成半成品了再决定是否拆解。

Step 1: 初始化生产线模型

在这个步骤中，我们定义了生产线的核心元素：Component（零部件）、SemiProduct（半成品）和 FinalProduct（成品）类。每个类都包含了关键属性，如不合格率、成本和价格等。Component 类包括购买单价和检测成本，SemiProduct 类添加了装配和拆卸成本，而 FinalProduct 类还考虑了市场售价和更换成本。这些类的设计直接影响后续的生产决策和利润计算，为整个模拟提供了基础框架，确保模型能够准确反映实际生产环境的复杂性。

Step2 :蒙特卡罗初始化

创建空列表 profits 和 decisions，用于存储每次模拟的结果。设置模拟次数 1000 次。

Step3: 零部件检查决策

对每个零部件决定是否进行检查。半成品检查决策：对每个半成品决定是否进行检查。半成品拆解决策：对每个半成品决定是否拆解不合格品。半成品拆解后检查决策：对拆解后的零部件决定是否再次检查。成品检查决策：决定是否对最终产品进行检查。成品拆解决策：决定是否拆解不合格的成品。成品拆解后检查决策：对拆解后的半成品决定是否再次检查。随机生成了 2 轮内的各种决策。

Step4: 执行生产流程

首先，调用公式（26）对零部件进行处理，计算出合格品和不合格品的含量。接着调用公式（28）计算出进程中半成品加工过程中合格品和不合格品的价格变化，再利用公式（29）计算拆解随机模拟的拆解决策后的合格品和不合格品的数量。再根据利用公式（31）利用随机的对半成品和成品，已经半成品拆解后是否检查的决策计算出过程的总耗费（total_cost），最后根据公式（30），计算出该次的利润，并根据公式（32），与之前的利润最大值进行对比，取两者最大值。

Step 5: 运行主程序

首先，初始化生产线参数，创建零部件、半成品和成品对象，并生成 ProductionLine 实例。随后，运行蒙特卡洛模拟生成数据，并进行可视化分析。最后，找出最高利润及对应的决策组合，打印关键信息。

5.3.3 问题结论

对模型的复杂度可以估算大概在 2 的 15 次方左右，可以使用蒙特卡罗模拟十万次进行检验，就能大概覆盖所有情况。

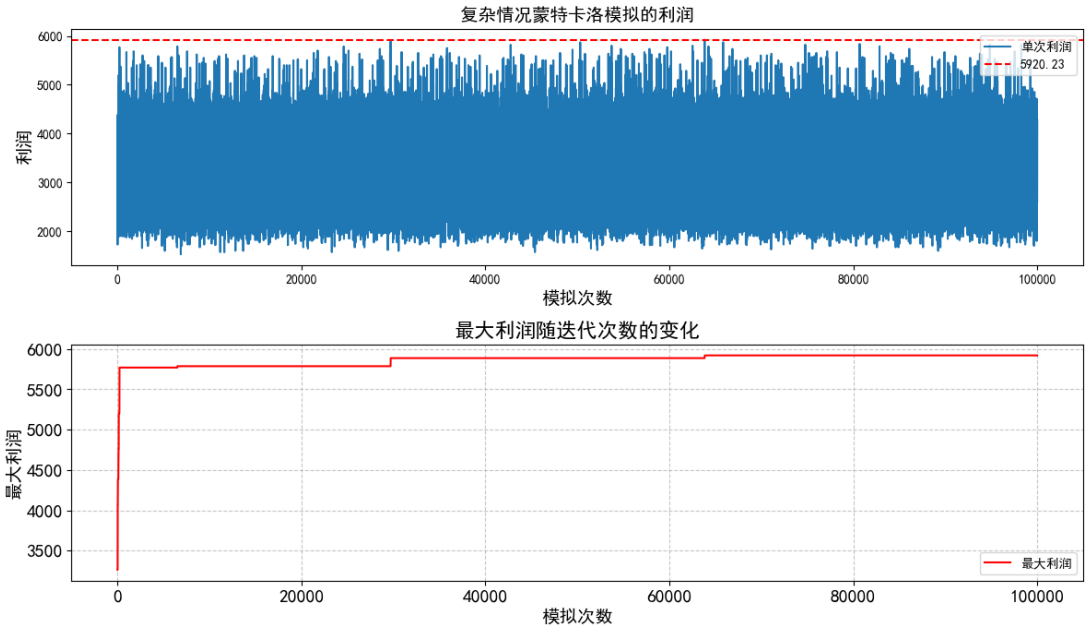


图 8 第三问蒙特卡罗模拟法结果

通过蒙特卡罗模拟的结果看出随着模拟次数增加，最大利润已经收敛，收敛至 5920，也就是说企业可以获利 5920 元。

5.3.4 检验分析

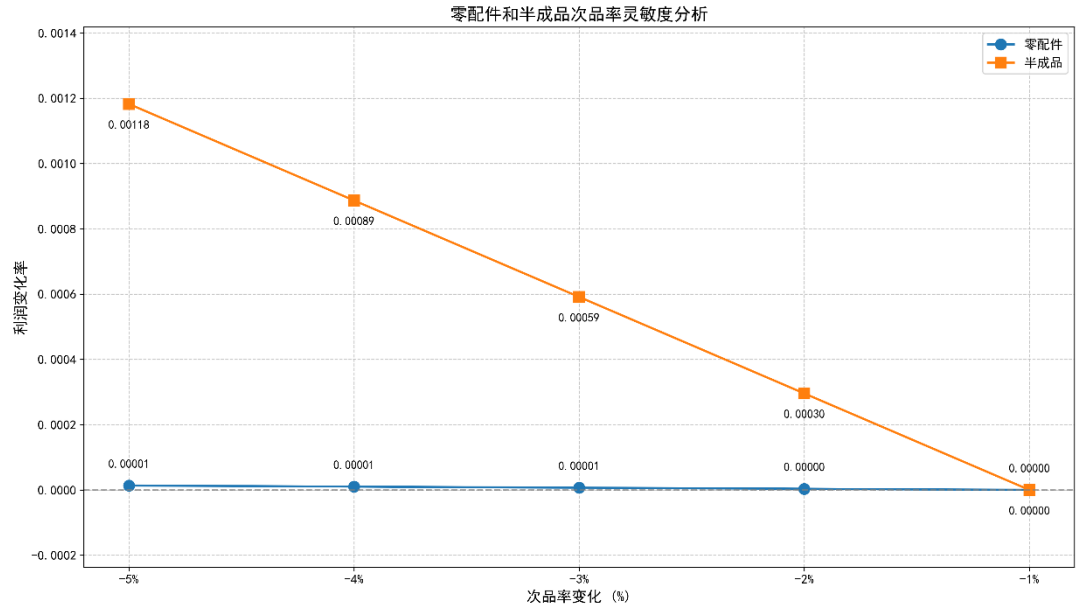


图 9 第三问灵敏度分析

通过对零配件和半成品次品率进行灵敏度分析，可以看出零配件的次品率对利润的影响十分小，而半成品的次品率对利润的相较于零配件的次品率更加灵敏。但总体对利润影响都不是特别大。

5.3.5 小结

本问通过动态规划计算出了企业最大获利为 5920 元。

5.4 问题四的建模与求解

5.4.1 模型建立

对于不同批次的零件，本问可以使用第一问的抽检方法进行接收和拒收的判断。但是供应商提供的零件可能质量层次不齐，结合第二问和第三问的概率，可以模拟供应商供应的零件的次品率为 5%~20%，随机取数，然后利用抽样方案进行检验，查看能接收的零件次品率分布情况，根据分布情况将次品率分别代入到第二问和第三问中的模型，计算出方案，根据不同次品率的出现的频数，可以计算出某个方案成为最优方案的概率，找出概率最大的方案作为企业的在这种抽检方案下的最优生产策略，就能尽可能让企业盈利。

关于半成品的次品率和成品的次品率，可以观察问题二和问题三，半成品和成品的次品率总是和零件的次品率的最小值一致，因此本问也延续问题二和问题三的特点，抽样检测后，半成品和成品的次品率和零配件次品率的最小值持平。

5.4.2 模型求解

首先利用假设一批零件为 300 件，通过随机数模拟可以得出不同次品率的频率分布直方图为

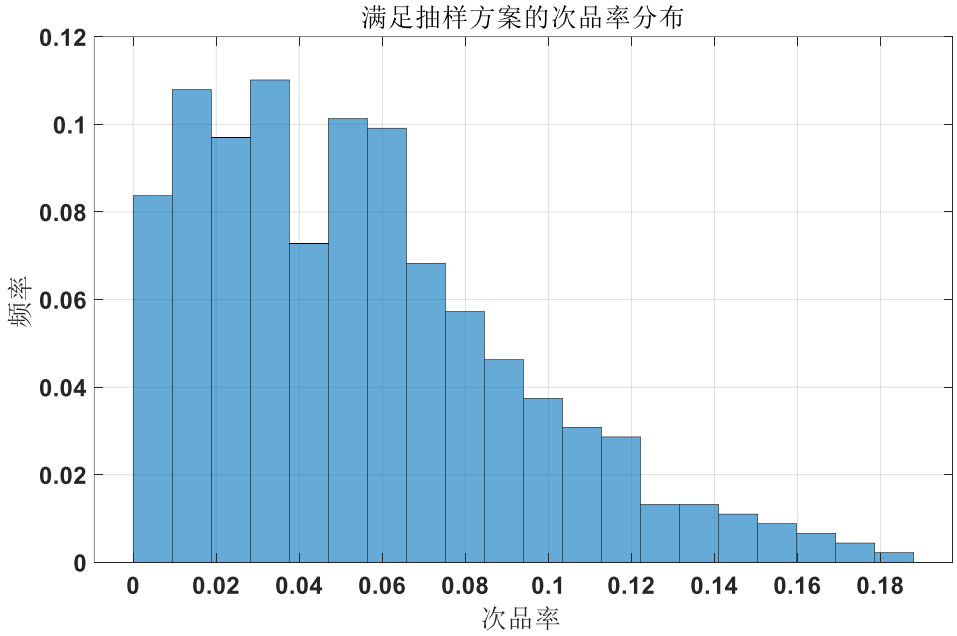


图 10 不同次品率直方图

然后可以不同次品率在总体中出现的概率，作为加工中随机的概率情况。通过将

5.4.3 问题结论

问题二的决策为

表 3 问题四结果

编号	概率	最佳路径
1	29%	[0, 0, 0, 1, 0, 0, 0, 0]
2	39%	[0, 0, 0, 1, 0, 0, 0, 0]
3	34%	[1, 1, 0, 1, 0, 0, 0, 0]
4	59%	[1, 1, 0, 1, 0, 0, 0, 0]
5	55.00%	[0, 1, 0, 1, 0, 0, 0, 0]
6	72.00%	[0, 0, 0, 0, 0, 0, 0, 0]

其中第 1 种情况下的利润为

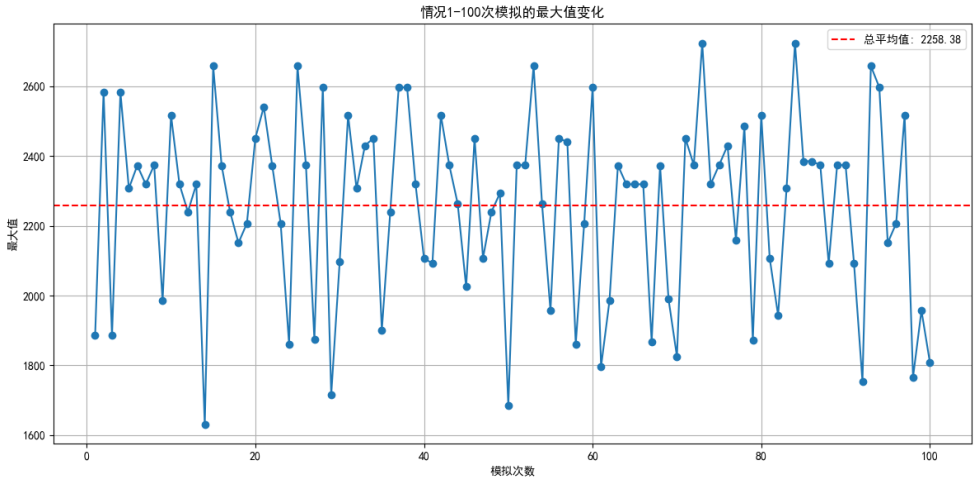


图 11 问题二情况 1 的利润变化图

从图中可以看出利润平均值大概在 2258 元。

问题三的策略：

零部件检查决策: (False, False, False, False, False, False, False, False)

半成品检查决策: (True, False, True)

半成品拆解决策: (False, False, False)

半成品拆解后检查决策: (False, False, False)

成品检查决策: True

成品拆解决策: False

成品拆解后检查决策: False

上述策略是最常见的决策策略出现 3 次，概率为 3.00%

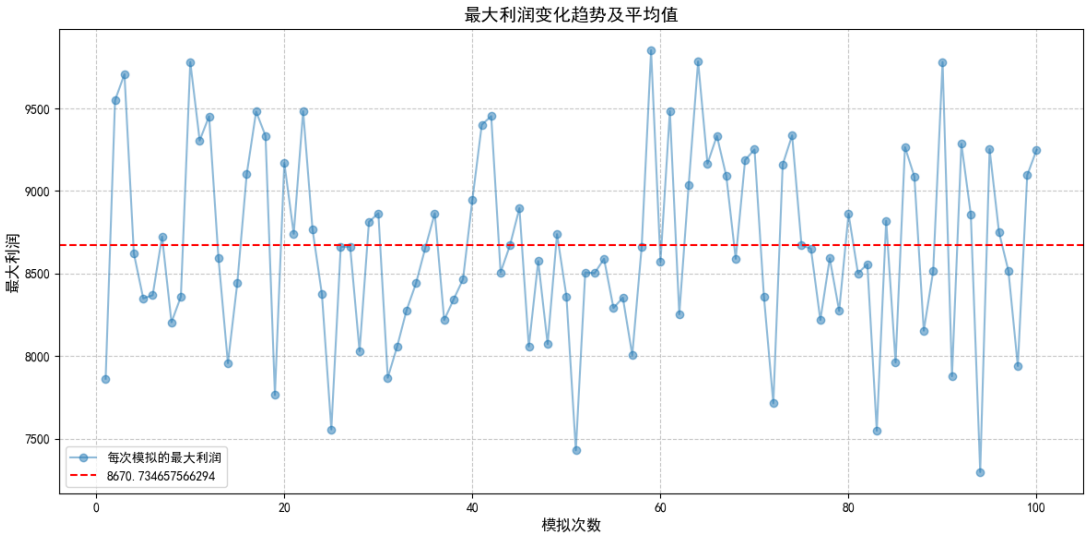


图 12 问题三的情况

这种情况下的利润为 8670 元。

5.4.4 小结

本文通过结合问题一和问题二问题三的模型，最终算出在抽样检测的次品率下，前两问模型利润的最大值。

六、模型评价与推广

模型评价

适当的抽样方法可以减少数据集的大小，节省计算资源和时间，同时提高模型的泛化能力，避免过拟合。优化抽样方案模型尽量满足不同情况下的样品总量进行优化，模型适用性较广泛。抽样方法的选择依赖于数据的特性和任务的需求。求解多种情况的决策方案计算量较大，但求解较准确。用蒙特卡洛进行大量仿真测试，检验结果较准确，模型实用性较强。

模型推广

该模型进行抽样的样品总量可以切换成更多情况，并可根据不同情况下生成不同的决策方案，模型的可推广到不同场景的抽样方案最优化选取，如样本总量为大规模数据集、医学研究、机器学习模型训练，具有较多的可用空间。

参考文献

[1] 丁鼎,余忠华,邓军.基于质量成本模型的工序检验策略优化研究[J].现代制造工程,2005,(06):1-3.

[2] 宋保维,毛昭勇,鲍亚东,等.基于质量标准的计数抽样检验优化方法[J].计算机工程,2008,(03):256-257.

[3] 孙志刚.浅谈抽检特性曲线(OC 曲线)在抽样检验中的应用[J].林产工业,2007,(04):33-35+46.

[4] 翟敬梅,蒋梁中,谢存禧,等.粗集理论在生产过程质量诊断中的应用[J].机床与液压,2002,(04):109-111+100.

[5] 张赤斌,王海燕.多工序质量检验计划的多目标优化蚁群算法研究[J].中国机械工程,2006,(11):1166-1169.

[6] 温德成,安玉红,阮金祥.单质量特性进货检验方案的优化研究[J].标准科学,2010,(03):4-8.

[7] 施佳琛,陈向宇,高歌,等.蒙特卡罗模拟对多分类敏感问题随机应答技术模型下整群抽样信度与效度的评价[J].南京医科大学学报(自然科学版),2013,33(07):1007-1011.

[8] Yong S ,Feifei L ,Zheng W , et al.Uncertainty Quantification of Data-driven Quality Prediction Model For Realizing the Active Sampling Inspection of Mechanical Properties in Steel Production[J].International Journal of Computational Intelligence Systems,2024,17(1):

[9] 刘鸿飞.考虑两类检验错误的工序质检方案优化研究[D].山东大学,2019.

[10] 银路,刘卫,陈运.多工序质量检验点设置的经济分析[J].电子科技大学学报,1992,(01):104-111.

附录

说明：由于每个功能代码过长，附录只放了部分核心代码。

附录 1

支撑材料的文件列表

T1-1 模型计算 计算以最大的效度为目标，这批零配件的最优抽样方案。

T1-2 模型计算 以所得最优抽样方案算当为 95%的信度和 90%信度的两种情况下的各自的次品率。

T1 灵敏度分析 改变批产品量 N 的值反映的对比结果，得到模型灵敏度。

T2 模型计算 六种情况通过遍历求解规划，得出每种情况的决策方案。

T2 模型检验 蒙特卡洛随机数检验，验证模型合理性。

T3 模型计算 蒙特卡洛随机数进行计算，求得具体的决策方案，以及决策的依据及相应指标。

T3 灵敏度分析 改变次品率和检测成本，得到模型的灵敏度。

T4-1 模型计算 利用所得最优方案进行抽样检验，求得零配件、半成品、成品的次品率。

T4-2 模型计算 利用所得最优方案进行抽样检验，利用上述求求得求得问题二，问题三中每种情况的决策方案。

附录 2

T1-1

```
%假定 N=300，  画出 L(p)曲线
%定义参数
N = 300;  % 总数
p_values = 0:0.001:0.2;  % p 的遍历范围


% 初始化绘图
figure;
hold on;  % 保持绘图，确保所有曲线都在同一张图上
n=72;
% 可以遍历不同的 n 值，从 45 到 65，步长为 5
for c = 0:floor(n*0.1)

    L_values = zeros(size(p_values));  % 用于保存对应的 L(p) 值

    % 计算似然函数 L(p) 对于当前的 n 和 c
    for i = 1:length(p_values)
        p = p_values(i);
        L = 0;  % 初始化 L(p)

        for d = 0:c
            term1 = binomial_coef(N * p, d);
```

```
            term2 = binomial_coef(N * (1 - p), n - d);
            denominator = binomial_coef(N, n);  % 总的组合数
            term = (term1 * term2) / denominator;
            L = L + term;
        end

        L_values(i) = L;  % 保存每个 p 对应的 L(p)
    end

    % 为每个 n 绘制曲线，区分不同的 n
    plot(p_values, L_values, 'DisplayName', ['n=' num2str(n) ', c=' num2str(c)]);
end

% 在图上画出垂直线 p = 0.05 和 p = 0.15
line([0.05 0.05], [0 1], 'Color', 'g', 'LineStyle', '--', 'DisplayName', 'p=0.05'); % 绿线
line([0.15 0.15], [0 1], 'Color', 'g', 'LineStyle', '--', 'DisplayName', 'p=0.15'); % 绿线

% 在图上画出水平线 L(p) = 0.05 和 L(p) = 0.95
line([0 0.2], [0.05 0.05], 'Color', 'm', 'LineStyle', '--', 'DisplayName', 'L(p)=0.05'); % 紫线
line([0 0.2], [0.95 0.95], 'Color', 'm', 'LineStyle', '--', 'DisplayName', 'L(p)=0.95'); % 紫线

% 设置图形的标签和标题
xlabel('p');
ylabel('L(p)');
title('L(p) for different values of n with c=5');
legend('show');
grid on;
% 定义参数
N = 300;   % 总数
p1 = 0.05; % 概率 p1
c_values = [5, 6, 7, 8, 9];  % c 的取值范围
n_values = 70:110;  % n 的遍历范围

% 开始遍历 c
figure;
hold on;  % 保持绘图

colors = ['r', 'b', 'm', 'g', 'c'];  % 定义颜色，添加青色 'c' 用于 c = 5
```

```
markers = ['o', 's', '^', 'd', '*'];  % 定义标记符号，添加 '*' 用于 c = 5
L_values_all = {};  % 用于存储所有 c 对应的 L(p)

% 遍历 c
for j = 1:length(c_values)
    c = c_values(j);
    L_values = zeros(size(n_values));  % 存储每个 n 对应的 L(p) 值

    % 遍历 n
    for i = 1:length(n_values)
        n = n_values(i);
        L = 0;  % 初始化 L

        % 计算 L(p)
        for d = 0:c
            term1 = binomial_coef3(N * p1, d);        % binom(N*p, d)
            term2 = binomial_coef3(N * (1 - p1), n - d);  % binom(N*(1-p), n-d)
            denominator = binomial_coef3(N, n);       % binom(N, n)

            % 累加每一项
            L = L + (term1 * term2) / denominator;
        end

        % 记录当前 n 对应的 L 值
        L_values(i) = 1-L;
    end

    % 保存每条曲线的数据
    L_values_all{j} = L_values;

    % 绘制当前 c 对应的曲线，带有不同标记符号
    plot(n_values, L_values, [colors(j) '-' markers(j)], 'LineWidth', 2, 'DisplayName', ['c = ' num2str(c)]);
end
% 第一个数据集
%绘制当 c=5,6,7,8,9 时的
x1 = [70:110];
y1 = [0.108516431486950 0.114924238954929 0.121546572728156 0.128382194549302 0.135429492143226 0.142686482316621 0.150150815066441 0.157819778668612 0.165690305713762 0.173758980052357 0.182022044612943
```

```
0.190475410053797 0.199114664208897 0.207935082277918 0.216931637727917 0.226099013850934 0.235431615938825 0.244923584025313 0.254568806148202 0.264360932086133 0.274293387520060 0.284359388573585 0.294551956681262 0.304863933743434 0.315287997512949 0.325816677172577 0.336442369056811 0.347157352469601 0.357953805557012 0.368823821190670 0.379759422818949 0.390752580245358 0.401795225294117 0.412879267325415 0.423996608560837 0.435139159186695 0.446298852198420 0.457467657952889 0.468637598402867 0.479800760976322 0.490949312080481
];
x2 = [70:110];
y2 = [0.0362715433672018 0.0390985172500933 0.0420785674479918 0.0452154623933738 0.0485128218180390 0.0519741051496231 0.0556026002363886 0.0594014124392541 0.0633734541255151 0.0675214345954311 0.0718478504739414 0.0763549765974070 0.0810448574245299 0.0859192989889819 0.0909798614296619 0.0962278521087852 0.101664319344407 0.107290046771314 0.113105548345362 0.119111064006827 0.125306556010115 0.131691705932244 0.138265912361644 0.145028289279289 0.151977665125676 0.159112582559936 0.166431298908789 0.173931787297751 0.181611738463070 0.189468563234522 0.197499395680248 0.205701096901563 0.214070259464801 0.222603212457241 0.231296027146597 0.240144523231796 0.249144275660701 0.258290621994669 0.267578670303636 0.277003307558236 0.286559208504640 ];
% 第二个数据集
x3 = [70:110];
y3 = [0.00949351533825793 0.0104249250087757 0.0114266225542055 0.0125021293927092 0.0136550221856608 0.0148889265974753 0.0162075107238678 0.0176144782063238 0.0191135610486300 0.0207085121506219 0.0224030975786786 0.0242010885926236 0.0261062534514044 0.0281223490099992 0.0302531121430105 0.0325022510055018 0.0348734361627228 0.0373702916088146 0.0399963856993717 0.0427552220256475 0.0456502302523301 0.0486847569481579 0.0518620564306443 0.0551852816587354 0.0586574751901013 0.0622815602359815 0.0660603318388788 0.0699964481946648 0.0740924221495146 0.0783506128945568 0.0827732178831301 0.0873622649950878 0.0921196049711792 0.0970469041428576 0.102145637475368 0.107417081951550 0.112862310312045 0.118482185171487 0.124277353537261 0.130248241735801 0.136395050775466];% 第三个数据集
x4 = [70:110];
y4 = [0.00192842221797229 0.00215866418244914 0.00241134464427051 0.00268812949250696 0.00299076385064223 0.00332107273717353 0.00368096152403252 0.00407241618639298 0.00449750333486776 0.00495837001966026 0.00545724330001796 0.00599642957208713 0.00657831365052974 0.00720535758903729 0.00788009924792332 0.00860515059177336 0.00938319572140611 0.0102169886330843 0.0111093507026573 0.0120631678959638 0.0130813877011128 0.0141670157864617 0.0153231123802265 0.016552788381892
```

```
6 0.0178592011981648 0.0192455503142284 0.0207150726042464 0.02227103
73820169 0.0239167412032003 0.0256555024240486 0.0274906555249338 0.0
294255452075478 0.0314635202753941 0.0336079273103316 0.0358621041520
326 0.0382293731990895 0.0407130345392147 0.0433163589224626 0.046042
5805999434 0.0488948900298867 0.0518764264798218];
106.  x5 = [70:110];
107.  y5 = [0.000299825782355323 0.000342310906708998 0.0003899009487
49355 0.000443096835598511 0.000502436905803871 0.000568498683149676
0.000641900658017724 0.000723304073936615 0.000813414713600635 0.0009
12984676462392 0.00102281414298588 0.00114375311955950 0.001276703159
51700 0.00142261904453744 0.00158251043301083 0.00175744345567419 0.0
0194854225935637 0.00215699048731011 0.00238403268854648 0.0026309756
5122983 0.00289918964861013 0.00319010959327704 0.00350523608669318 0
.00384613636435893 0.00421444511863756 0.00461186519845569 0.00504016
817746156 0.00550119477836109 0.00599685515094284 0.00652912899401670
 0.00710006551423248 0.00771178321463584 0.00836646950582054 0.009066
38013544325 0.00981383842496952 0.0106112343143602 0.0114610232034258
 0.0123657245846239 0.0133279204708372 0.0143502535998079 0.015435425
4248162];
108.  % 创建图形窗口
109.  figure;
110.
111.  % 绘制第一条曲线
112.  plot(x1, y1, '-
o', 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'w', ...
113.       'LineWidth', 1.5, 'MarkerSize', 6);
114.  hold on;
115.
116.  % 绘制第二条曲线
117.  plot(x2, y2, '-
o', 'MarkerEdgeColor', 'b', 'MarkerFaceColor', 'w', ...
118.       'LineWidth', 1.5, 'MarkerSize', 6);
119.
120.  % 绘制第三条曲线（紫色线）
121.  plot(x3, y3, '-
o', 'MarkerEdgeColor', 'm', 'MarkerFaceColor', 'w', ...
122.       'LineWidth', 1.5, 'MarkerSize', 6);
123.  % 绘制第三条曲线（紫色线）
124.  plot(x4, y4, '-
o', 'MarkerEdgeColor', 'm', 'MarkerFaceColor', 'w', ...
125.       'LineWidth', 1.5, 'MarkerSize', 6);
126.  plot(x5, y5, '-
o', 'MarkerEdgeColor', 'b', 'MarkerFaceColor', 'w', ...
127.       'LineWidth', 1.5, 'MarkerSize', 6);
```

```
128.
129.
130.  % 计算并标注距离 0.05 最近的点
131.
132.  % 对于第一条曲线
133.  [~, idx1] = min(abs(y1 - 0.1));
134.  closest_x1 = x1(idx1);
135.  closest_y1 = y1(idx1);
136.
137.  % 对于第二条曲线
138.  below_0_05 = y2 < 0.1;
139.  if any(below_0_05)
140.      [~, idx2] = min(abs(y2(below_0_05) - 0.1));
141.      closest_x2_below = x2(below_0_05);
142.      closest_x2_below = closest_x2_below(idx2);
143.      closest_y2_below = y2(below_0_05);
144.      closest_y2_below = closest_y2_below(idx2);
145.  else
146.      closest_x2_below = NaN;
147.      closest_y2_below = NaN;
148.  end
149.
150.  % 总体距离 0.05 最近的点（包括所有点）
151.  [~, idx2] = min(abs(y2 - 0.1));
152.  closest_x2_all = x2(idx2);
153.  closest_y2_all = y2(idx2);
154.
155.  % 对于第三条曲线
156.  [~, idx3] = min(abs(y3 - 0.1));
157.  closest_x3 = x3(idx3);
158.  closest_y3 = y3(idx3);
159.  % 对于第三条曲线
160.  [~, idx4] = min(abs(y4 - 0.1));
161.  closest_x4 = x4(idx4);
162.  closest_y4 = y4(idx4);
163.  [~, idx5] = min(abs(y5 - 0.1));
164.  closest_x5 = x5(idx5);
165.  closest_y5 = y5(idx5);
166.  % 绘制标记和文本
167.  plot(closest_x1, closest_y1, 'ro', 'MarkerFaceColor', 'r', 'Mar
kerSize', 8);
168.  text(closest_x1, closest_y1, sprintf('%.4f', closest_y1), ...
169.       'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'rig
ht', ...
```

```
        'Color', 'r', 'FontSize', 14, 'FontWeight', 'bold');
    plot([closest_x1, closest_x1], [0, closest_y1], 'r--
','LineWidth', 1.5); % 垂直线

    % 标注第二条曲线中 y 值低于 0.05 的点中距离 0.05 最近的点
    if ~isnan(closest_x2_below) && ~isnan(closest_y2_below)
        plot(closest_x2_below, closest_y2_below, 'gs', 'MarkerFaceC
olor', 'k', 'MarkerSize', 8);
        text(closest_x2_below, closest_y2_below, sprintf('%.4f', cl
osest_y2_below), ...
            'VerticalAlignment', 'top', 'HorizontalAlignment', 'le
ft', ...
            'Color', 'k', 'FontSize', 14, 'FontWeight', 'bold');
        plot([closest_x2_below, closest_x2_below], [0, closest_y2_b
elow], 'k--','LineWidth', 1.5); % 垂直线
    end

    % 标注第二条曲线中所有点的距离 0.05 最近的点
    plot(closest_x2_all, closest_y2_all, 'bs', 'MarkerFaceColor', '
b', 'MarkerSize', 8);
    text(closest_x2_all, closest_y2_all, sprintf('%.4f', closest_y2
_all), ...
        'VerticalAlignment', 'top', 'HorizontalAlignment', 'left',
 ...
        'Color', 'b', 'FontSize', 14, 'FontWeight', 'bold');
    plot([closest_x2_all, closest_x2_all], [0, closest_y2_all], 'b-
-','LineWidth', 1.5); % 垂直线

    % 绘制第三条曲线（紫色线）
    plot(closest_x3, closest_y3, 'ms', 'MarkerFaceColor', 'm', 'Mar
kerSize', 8);
    text(closest_x3, closest_y3, sprintf('%.4f', closest_y3), ...
        'VerticalAlignment', 'top', 'HorizontalAlignment', 'left',
 ...
        'Color', 'm', 'FontSize', 14);
    plot([closest_x3, closest_x3], [0, closest_y3], 'm--
','LineWidth', 1.5); % 垂直线

    plot(closest_x4, closest_y4, 'ms', 'MarkerFaceColor', 'm', 'Mar
kerSize', 8);
    text(closest_x4, closest_y4, sprintf('%.4f', closest_y4), ...
        'VerticalAlignment', 'top', 'HorizontalAlignment', 'left',
 ...
        'Color', 'm', 'FontSize', 14);
```

```
plot([closest_x4, closest_x4], [0, closest_y4], 'm--','LineWidth', 1.5); % 垂直线

plot(closest_x5, closest_y5, 'bs', 'MarkerFaceColor', 'b', 'MarkerSize', 8);
text(closest_x5, closest_y5, sprintf('%.4f', closest_y5), ...
      'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', ...
      'Color', 'b', 'FontSize', 14, 'FontWeight', 'bold');
plot([closest_x5, closest_x5], [0, closest_y5], 'b--','LineWidth', 1.5); % 垂直线

% 标注每条曲线的“c”值
text(x1(end), y1(end), 'b=5', 'Color', 'r', 'FontSize', 22, 'FontWeight', 'bold');
text(x2(end), y2(end), 'b=6', 'Color', 'b', 'FontSize', 22, 'FontWeight', 'bold');
text(x3(end), y3(end), 'b=7', 'Color', 'm', 'FontSize', 22, 'FontWeight', 'bold');
text(x4(end), y4(end), 'b=8', 'Color', 'm', 'FontSize', 22, 'FontWeight', 'bold');
text(x5(end), y5(end), 'b=9', 'Color', 'b', 'FontSize', 22, 'FontWeight', 'bold');
% 添加水平直线
yline(0.1, 'k--', 'LineWidth', 1.5); % 黑色虚线

% 添加图例、标题和轴标签

xlabel('确定子样数', 'FontSize', 24);
ylabel('使用方风险', 'FontSize', 24);

% 显示网格
grid on;

% 释放图形持有
hold off;
```

附录 2

T1-1

```
%假定 N=300，  画出 L(p)曲线
%定义参数
N = 300;  % 总数
p_values = 0:0.001:0.2;  % p 的遍历范围

```

```
232.
233.    % 初始化绘图
234.    figure;
235.    hold on;  % 保持绘图，确保所有曲线都在同一张图上
236.    n=72;
237.    % 可以遍历不同的 n 值，从 45 到 65，步长为 5
238.    for c = 0:floor(n*0.1)
239.
240.    L_values = zeros(size(p_values));  % 用于保存对应的 L(p) 值
241.
242.    % 计算似然函数 L(p) 对于当前的 n 和 c
243.    for i = 1:length(p_values)
244.    p = p_values(i);
245.    L = 0;  % 初始化 L(p)
246.
247.    for d = 0:c
248.    term1 = binomial_coef(N * p, d);
249.    term2 = binomial_coef(N * (1 - p), n - d);
250.    denominator = binomial_coef(N, n);  % 总的组合数
251.    term = (term1 * term2) / denominator;
252.    L = L + term;
253.    end
254.
255.    L_values(i) = L;  % 保存每个 p 对应的 L(p)
256.    end
257.
258.    % 为每个 n 绘制曲线，区分不同的 n
259.    plot(p_values, L_values, 'DisplayName', ['n=' num2str(n) ', c='
    num2str(c)]);
260.    end
261.
262.    % 在图上画出垂直线 p = 0.05 和 p = 0.15
263.    line([0.05 0.05], [0 1], 'Color', 'g', 'LineStyle', '--
   ', 'DisplayName', 'p=0.05'); % 绿线
264.    line([0.15 0.15], [0 1], 'Color', 'g', 'LineStyle', '--
   ', 'DisplayName', 'p=0.15'); % 绿线
265.
266.    % 在图上画出水平线 L(p) = 0.05 和 L(p) = 0.95
267.    line([0 0.2], [0.05 0.05], 'Color', 'm', 'LineStyle', '--
   ', 'DisplayName', 'L(p)=0.05'); % 紫线
268.    line([0 0.2], [0.95 0.95], 'Color', 'm', 'LineStyle', '--
   ', 'DisplayName', 'L(p)=0.95'); % 紫线
269.
270.    % 设置图形的标签和标题
```

```
xlabel('p');
ylabel('L(p)');
title('L(p) for different values of n with c=5');
legend('show');
grid on;
% 定义参数
N = 300;   % 总数
p1 = 0.05; % 概率 p1
c_values = [5, 6, 7, 8, 9];  % c 的取值范围
n_values = 70:110;  % n 的遍历范围

% 开始遍历 c
figure;
hold on;  % 保持绘图

colors = ['r', 'b', 'm', 'g', 'c'];  % 定义颜色，添加青色 'c' 用于 c = 5
markers = ['o', 's', '^', 'd', '*'];  % 定义标记符号，添加 '*' 用于 c = 5
L_values_all = {};  % 用于存储所有 c 对应的 L(p)

% 遍历 c
for j = 1:length(c_values)
c = c_values(j);
L_values = zeros(size(n_values));  % 存储每个 n 对应的 L(p) 值

% 遍历 n
for i = 1:length(n_values)
n = n_values(i);
L = 0;  % 初始化 L

% 计算 L(p)
for d = 0:c
term1 = binomial_coef3(N * p1, d);        % binom(N*p, d)
term2 = binomial_coef3(N * (1 - p1), n - d);  % binom(N*(1-p), n-d)
denominator = binomial_coef3(N, n);       % binom(N, n)

% 累加每一项
L = L + (term1 * term2) / denominator;
end

% 记录当前 n 对应的 L 值
L_values(i) = 1-L;
```

```
    end

    % 保存每条曲线的数据
    L_values_all{j} = L_values;

    % 绘制当前 c 对应的曲线，带有不同标记符号
    plot(n_values, L_values, [colors(j) '-
    ' markers(j)], 'LineWidth', 2, 'DisplayName', ['c = ' num2str(c)]);
    end
    % 第一个数据集
    %绘制当 c=5,6,7,8,9 时的
    x1 = [70:110];
    y1 = [0.108516431486950 0.114924238954929 0.121546572728156 0.1
    28382194549302 0.135429492143226 0.142686482316621 0.15015081506644
    1 0.157819778668612 0.165690305713762 0.173758980052357 0.182022044
    612943 0.190475410053797 0.199114664208897 0.207935082277918 0.2169
    31637727917 0.226099013850934 0.235431615938825 0.244923584025313 0
    .254568806148202 0.264360932086133 0.274293387520060 0.284359388573
    585 0.294551956681262 0.304863933743434 0.315287997512949 0.3258166
    77172577 0.336442369056811 0.347157352469601 0.357953805557012 0.36
    8823821190670 0.379759422818949 0.390752580245358 0.401795225294117
     0.412879267325415 0.423996608560837 0.435139159186695 0.4462988521
    98420 0.457467657952889 0.468637598402867 0.479800760976322 0.49094
    9312080481
    ];
    x2 = [70:110];
    y2 = [0.0362715433672018 0.0390985172500933 0.0420785674479918
    0.0452154623933738 0.0485128218180390 0.0519741051496231 0.05560260
    02363886 0.0594014124392541 0.0633734541255151 0.0675214345954311 0
    .0718478504739414 0.0763549765974070 0.0810448574245299 0.085919298
    9889819 0.0909798614296619 0.0962278521087852 0.101664319344407 0.1
    07290046771314 0.113105548345362 0.119111064006827 0.12530655601011
    5 0.131691705932244 0.138265912361644 0.145028289279289 0.151977665
    125676 0.159112582559936 0.166431298908789 0.173931787297751 0.1816
    11738463070 0.189468563234522 0.197499395680248 0.205701096901563 0
    .214070259464801 0.222603212457241 0.231296027146597 0.240144523231
    796 0.249144275660701 0.258290621994669 0.267578670303636 0.2770033
    07558236 0.286559208504640  ];
    % 第二个数据集
    x3 = [70:110];
    y3 = [0.00949351533825793 0.0104249250087757 0.0114266225542055
     0.0125021293927092 0.0136550221856608 0.0148889265974753 0.0162075
    107238678 0.0176144782063238 0.0191135610486300 0.0207085121506219
    0.0224030975786786 0.0242010885926236 0.0261062534514044 0.02812234
```

```
   90099992 0.0302531121430105 0.0325022510055018 0.0348734361627228 0
   .0373702916088146 0.0399963856993717 0.0427552220256475 0.045650230
   2523301 0.0486847569481579 0.0518620564306443 0.0551852816587354 0.
   0586574751901013 0.0622815602359815 0.0660603318388788 0.0699964481
   946648 0.0740924221495146 0.0783506128945568 0.0827732178831301 0.0
   873622649950878 0.0921196049711792 0.0970469041428576 0.10214563747
   5368 0.107417081951550 0.112862310312045 0.118482185171487 0.124277
   353537261 0.130248241735801 0.136395050775466];% 第三个数据集
   x4 = [70:110];
   y4 = [0.00192842221797229 0.00215866418244914 0.002411344644270
   51 0.00268812949250696 0.00299076385064223 0.00332107273717353 0.00
   368096152403252 0.00407241618639298 0.00449750333486776 0.004958370
   01966026 0.00545724330001796 0.00599642957208713 0.0065783136505297
   4 0.00720535758903729 0.00788009924792332 0.00860515059177336 0.009
   38319572140611 0.0102169886330843 0.0111093507026573 0.012063167895
   9638 0.0130813877011128 0.0141670157864617 0.0153231123802265 0.016
   5527883818926 0.0178592011981648 0.0192455503142284 0.0207150726042
   464 0.0222710373820169 0.0239167412032003 0.0256555024240486 0.0274
   906555249338 0.0294255452075478 0.0314635202753941 0.03360792731033
   16 0.0358621041520326 0.0382293731990895 0.0407130345392147 0.04331
   63589224626 0.0460425805999434 0.0488948900298867 0.051876426479821
   8];
   x5 = [70:110];
   y5 = [0.000299825782355323 0.000342310906708998 0.0003899009487
   49355 0.000443096835598511 0.000502436905803871 0.00056849868314967
   6 0.000641900658017724 0.000723304073936615 0.000813414713600635 0.
   000912984676462392 0.00102281414298588 0.00114375311955950 0.001276
   70315951700 0.00142261904453744 0.00158251043301083 0.0017574434556
   7419 0.00194854225935637 0.00215699048731011 0.00238403268854648 0.
   00263097565122983 0.00289918964861013 0.00319010959327704 0.0035052
   3608669318 0.00384613636435893 0.00421444511863756 0.00461186519845
   569 0.00504016817746156 0.00550119477836109 0.00599685515094284 0.0
   0652912899401670 0.00710006551423248 0.00771178321463584 0.00836646
   950582054 0.00906638013544325 0.00981383842496952 0.010611234314360
   2 0.0114610232034258 0.0123657245846239 0.0133279204708372 0.014350
   2535998079 0.0154354254248162];
   % 创建图形窗口
   figure;

   % 绘制第一条曲线
   plot(x1, y1, '-
   o', 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'w', ...
   'LineWidth', 1.5, 'MarkerSize', 6);
   hold on;
```

```
341.
342.    % 绘制第二条曲线
343.    plot(x2, y2, '-
    o', 'MarkerEdgeColor', 'b', 'MarkerFaceColor', 'w', ...
344.    'LineWidth', 1.5, 'MarkerSize', 6);
345.
346.    % 绘制第三条曲线（紫色线）
347.    plot(x3, y3, '-
    o', 'MarkerEdgeColor', 'm', 'MarkerFaceColor', 'w', ...
348.    'LineWidth', 1.5, 'MarkerSize', 6);
349.    % 绘制第三条曲线（紫色线）
350.    plot(x4, y4, '-
    o', 'MarkerEdgeColor', 'm', 'MarkerFaceColor', 'w', ...
351.    'LineWidth', 1.5, 'MarkerSize', 6);
352.    plot(x5, y5, '-
    o', 'MarkerEdgeColor', 'b', 'MarkerFaceColor', 'w', ...
353.    'LineWidth', 1.5, 'MarkerSize', 6);
354.
355.
356.    % 计算并标注距离 0.05 最近的点
357.
358.    % 对于第一条曲线
359.    [~, idx1] = min(abs(y1 - 0.1));
360.    closest_x1 = x1(idx1);
361.    closest_y1 = y1(idx1);
362.
363.    % 对于第二条曲线
364.    below_0_05 = y2 < 0.1;
365.    if any(below_0_05)
366.    [~, idx2] = min(abs(y2(below_0_05) - 0.1));
367.    closest_x2_below = x2(below_0_05);
368.    closest_x2_below = closest_x2_below(idx2);
369.    closest_y2_below = y2(below_0_05);
370.    closest_y2_below = closest_y2_below(idx2);
371.    else
372.    closest_x2_below = NaN;
373.    closest_y2_below = NaN;
374.    end
375.
376.    % 总体距离 0.05 最近的点（包括所有点）
377.    [~, idx2] = min(abs(y2 - 0.1));
378.    closest_x2_all = x2(idx2);
379.    closest_y2_all = y2(idx2);
380.
```

```
381.   % 对于第三条曲线
382.   [~, idx3] = min(abs(y3 - 0.1));
383.   closest_x3 = x3(idx3);
384.   closest_y3 = y3(idx3);
385.   % 对于第三条曲线
386.   [~, idx4] = min(abs(y4 - 0.1));
387.   closest_x4 = x4(idx4);
388.   closest_y4 = y4(idx4);
389.   [~, idx5] = min(abs(y5 - 0.1));
390.   closest_x5 = x5(idx5);
391.   closest_y5 = y5(idx5);
392.   % 绘制标记和文本
393.   plot(closest_x1, closest_y1, 'ro', 'MarkerFaceColor', 'r', 'Mar
   kerSize', 8);
394.   text(closest_x1, closest_y1, sprintf('%.4f', closest_y1), ...
395.   'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right',
   ...
396.   'Color', 'r', 'FontSize', 14, 'FontWeight', 'bold');
397.   plot([closest_x1, closest_x1], [0, closest_y1], 'r--
   ','LineWidth', 1.5); % 垂直线
398.
399.   % 标注第二条曲线中 y 值低于 0.05 的点中距离 0.05 最近的点
400.   if ~isnan(closest_x2_below) && ~isnan(closest_y2_below)
401.   plot(closest_x2_below, closest_y2_below, 'gs', 'MarkerFaceColor
   ', 'k', 'MarkerSize', 8);
402.   text(closest_x2_below, closest_y2_below, sprintf('%.4f', closes
   t_y2_below), ...
403.   'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', ...
404.   'Color', 'k', 'FontSize', 14, 'FontWeight', 'bold');
405.   plot([closest_x2_below, closest_x2_below], [0, closest_y2_below
   ], 'k--','LineWidth', 1.5); % 垂直线
406.   end
407.
408.   % 标注第二条曲线中所有点的距离 0.05 最近的点
409.   plot(closest_x2_all, closest_y2_all, 'bs', 'MarkerFaceColor', '
   b', 'MarkerSize', 8);
410.   text(closest_x2_all, closest_y2_all, sprintf('%.4f', closest_y2
   _all), ...
411.   'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', ...
412.   'Color', 'b', 'FontSize', 14, 'FontWeight', 'bold');
413.   plot([closest_x2_all, closest_x2_all], [0, closest_y2_all], 'b-
   -','LineWidth', 1.5); % 垂直线
414.
415.   % 绘制第三条曲线（紫色线）
```

```
416.   plot(closest_x3, closest_y3, 'ms', 'MarkerFaceColor', 'm', 'Mar
   kerSize', 8);
417.   text(closest_x3, closest_y3, sprintf('%.4f', closest_y3), ...
418.   'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', ...
419.   'Color', 'm', 'FontSize', 14);
420.   plot([closest_x3, closest_x3], [0, closest_y3], 'm--
   ','LineWidth', 1.5); % 垂直线
421.
422.   plot(closest_x4, closest_y4, 'ms', 'MarkerFaceColor', 'm', 'Mar
   kerSize', 8);
423.   text(closest_x4, closest_y4, sprintf('%.4f', closest_y4), ...
424.   'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', ...
425.   'Color', 'm', 'FontSize', 14);
426.   plot([closest_x4, closest_x4], [0, closest_y4], 'm--
   ','LineWidth', 1.5); % 垂直线
427.
428.   plot(closest_x5, closest_y5, 'bs', 'MarkerFaceColor', 'b', 'Mar
   kerSize', 8);
429.   text(closest_x5, closest_y5, sprintf('%.4f', closest_y5), ...
430.   'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', ...
431.   'Color', 'b', 'FontSize', 14, 'FontWeight', 'bold');
432.   plot([closest_x5, closest_x5], [0, closest_y5], 'b--
   ','LineWidth', 1.5); % 垂直线
433.
434.   % 标注每条曲线的“c”值
435.   text(x1(end), y1(end), 'b=5', 'Color', 'r', 'FontSize', 22, 'Fo
   ntWeight', 'bold');
436.   text(x2(end), y2(end), 'b=6', 'Color', 'b', 'FontSize', 22, 'Fo
   ntWeight', 'bold');
437.   text(x3(end), y3(end), 'b=7', 'Color', 'm', 'FontSize', 22, 'Fo
   ntWeight', 'bold');
438.   text(x4(end), y4(end), 'b=8', 'Color', 'm', 'FontSize', 22, 'Fo
   ntWeight', 'bold');
439.   text(x5(end), y5(end), 'b=9', 'Color', 'b', 'FontSize', 22, 'Fo
   ntWeight', 'bold');
440.   % 添加水平直线
441.   yline(0.1, 'k--', 'LineWidth', 1.5); % 黑色虚线
442.
443.   % 添加图例、标题和轴标签
444.
445.   xlabel('确定子样数', 'FontSize', 24);
446.   ylabel('使用方风险', 'FontSize', 24);
447.
448.   % 显示网格
```

```
449.    grid on;
450.
451.    % 释放图形持有
452.    hold off;
```

附录 3

T1-2

```
1. % 初始化结果数组，用于存储每个 c0 对应的 count_greater_than_c
2. c0_values = 0:0.01:1;  % c0 的取值范围
3. count_greater_than_c_values = zeros(size(c0_values));  % 用于存
   储 count_greater_than_c
4.
5. % 开始遍历每个 c0
6. for idx = 1:length(c0_values)
7.     c0 = c0_values(idx);  % 当前的 c0 值
8.     count_greater_than_c = 0;  % 初始化计数器
9.
10.        % 开始模拟 1000 次
11.        for run = 1:1000
12.
13.            % 初始化 result_matrix (从 1 到 300 的整数)
14.            result_matrix = 1:300;
15.
16.            % 从 result_matrix 随机抽取 n 个数 (n = 87)
17.            sampled_values = randsample(result_matrix, 72);
18.
19.            % 计算小于 300*c0 的个数
20.            count_less_than_30 = sum(sampled_values < 300 * c0);
21.
22.            % 如果小于 30 的个数大于 8，则记录次数
23.            if count_less_than_30 > 6
24.                count_greater_than_c = count_greater_than_c + 1;
25.            end
26.        end
27.
28.        % 记录当前 c0 对应的 count_greater_than_c
29.        count_greater_than_c_values(idx) = count_greater_than_c;
30.    end
31.
32.    % 绘制以 c0 为横坐标、 count_greater_than_c 为纵坐标的图
33.    figure;
34.    plot(c0_values, count_greater_than_c_values, '-o');
35.    xlabel('c0');
36.    ylabel('count\_greater\_than\_c');
```

```
37.    title('Count Greater Than C vs. c0');
38.    grid on;
39.
```

附录 3

T1

```
1. % 根据第一问假定 N=300 时的计算方法，分别算出当 N=100：100：500 时每个最优方案，并画出灵敏度曲线
2. % 数据定义
3. n = [50, 70, 72, 81, 82];  % 横坐标
4. c = [4, 6, 6, 7, 7];  % 纵坐标
5. N_values = [100, 200, 300, 400, 500];  % 对应的 N 值
6.
7. % 绘制图形
8. figure;
9. plot(n, c, 'bo-', 'MarkerFaceColor', 'b', 'MarkerSize', 8, 'LineWidth', 1.5);  % 蓝色带线的点图
10.
11.% 标注每个点的 N 值
12.for i = 1:length(n)
13.    text(n(i), c(i), sprintf('N=%d', N_values(i)), 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'FontSize', 12);
14.end
15.
16.% 添加轴标签和标题
17.xlabel('n', 'FontSize', 14);
18.ylabel('c', 'FontSize', 14);
19.title('n 与 c 关系图', 'FontSize', 16);
20.
21.% 添加网格
22.grid on;
```

附录 4

T2

```
1. # 不同决策下成品合格与不合格的量，第一种情况的决策方案
2. def calculate_hk_bk(h_prev, b_prev, n, k1, k2, tip,nx):
3.     if k1 == 1 and k2 == 0:
4.         bk = b_prev
5.         tip = False
6.         return h_prev, b_prev, bk, tip,
7.     elif k1 == 0 and k2 == 1 and tip == True:
```

```
        bk = b_prev
        return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
    elif k1 == 1 and k2 == 1 and tip == True:
        bk = b_prev
        return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
    else:
        bk = b_prev
        tip = False
        return h_prev, b_prev, bk, tip  # 当 k1 = 0, k2 = 0 时，保持不变

#ji
def calculate_vk(hk, bk, p, cp3, price3, eta1, p1, eta2, p2, check1, cp1, check2, cp2, break_cost, change_cost,
                 checked1, checked2, k1, k2, bk_old, biaoji, total_cost,nx):
    if check1 and not checked1 and biaoji == 0:
        total_cost += (hk + bk) / (1 - eta1) * cp1
        checked1 = True
    if check2 and not checked2 and biaoji == 0:
        total_cost += (hk + bk) / (1 - eta2) * cp2
        checked2 = True

    if k1 == 1 and k2 == 0 and biaoji == 0:
        biaoji += 1
        total_cost += (hk + bk) * cp3
    elif k1 == 0 and k2 == 1:
        total_cost += bk_old * break_cost + bk_old * change_cost+bk*nx*change_cost
    elif k1 == 1 and k2 == 1 and biaoji == 0:
        biaoji += 1
        total_cost += (hk + bk) * cp3 + bk_old * break_cost+bk_old* nx* change_cost
    else:
        total_cost += bk_old * change_cost

    return hk * p - total_cost, checked1, checked2, total_cost


def main():
    eta1 = 0.1      #零件 1 次品率
    p1 = 4          #零件 1 购买单价
    cp1 = 2         #零件 1 检验成本
    eta2 = 0.1      #零件 2 次品率
```

```
    p2 = 18          #零件 2 购买单价
    cp2 = 3          #零件 1 检验成本
    eta0 = 0.1       #成品次品率
    price3 = 6       #装配成本
    cp3 = 3          #成品检测成本
    p = 56           #成品市场销售
    change_cost = 6 #调换损失
    break_cost = 5  #拆解费用
    n = eta0 + eta1 + eta2
    norm = n         #记录总次品率
    n1 = 100         #初始零件 1 个数
    n2 = 100         #初始零件 2 个数


    f = [0]  # 初始化 f(0) = 0

    # biaoji 保证 k2 为 0 即不再拆卸后状态保持不变
    biaoji = 0
    bk = 0
    max_v = float('-inf')
    step=2
    path=[0]*step*4
    k=1
    checked1 = False
    checked2 = False
    while k<=step:
        #便于给 path 标记下标
        m = k
        for check1 in [0, 1]:
            for check2 in [0, 1]:
                s1 = -(check1) * n1 * eta1 + n1
                s2 = -(check2) * n2 * eta2 + n2
                hk = min(s1, s2)
                for k1 in [0, 1]:
                    for k2 in [0, 1]:
                        if k2 == 0:
                            tip = False

                        total_cost = hk * price3 + n1 * p1 + n2 * p
2
                        t1 = 0
                        t2 = 0
                        n = norm
```

```
                        # checked1，checked2 标记保证零件 1,2 分别最多被检查一次

                        tip=True

                        if check1 == 1 and t1 == 0:
                            t1 += 1
                            n = n - eta1
                            m=1
                        if check2 == 1 and t2 == 0:
                            t2 += 1
                            m=1
                                n = n - eta2

                            nx = ((1 - check1) * eta1 + (1 - check2) * eta2) / (
                                    (1 - check1) * eta1 + (1 - check2) * eta2 + eta0) + eta0

                            hk_new, bk_new, bk_old, tip= calculate_hk_bk(hk * (1 - n), hk * n, n, k1, k2, tip,nx)

                            vk, new_checked1, new_checked2, total_cost= calculate_vk(hk_new, \
                        bk_new, p, cp3, price3, eta1, p1,
                        eta2, p2, check1, cp1, check2, cp2, \
                        break_cost, change_cost, checked1,
                        checked2, k1, k2, bk_old, biaoji,
                        total_cost,nx)
                            new_value = vk


                            if new_value > max_v:

                                max_v = round(new_value)
                                path[4*m-4]=check1
                                path[4 * m - 3] = check2
                                path[4 * m - 2] = k1
                                path[4 * m - 1] = k2
```

```
                                bk=bk_new

                            if k2==0:
                                continue

                            f.append(max_v)
                k+=1
        print("最大的 f(k) 值为：", f[-1])
        print(path)


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

附录 5

T2

```
#第一种情况进行蒙特卡洛检验
import random
import matplotlib.pyplot as plt
from matplotlib import font_manager


plt.rcParams['axes.unicode_minus'] = False

# 尝试加载系统中的中文字体
try:
    font_path = font_manager.findfont(font_manager.FontProperties(family='SimHei'))
    prop = font_manager.FontProperties(fname=font_path)
    plt.rcParams['font.family'] = prop.get_name()
except:
    print("无法找到 SimHei 字体，将使用系统默认字体。中文可能无法正确显示。")

def calculate_hk_bk(h_prev, b_prev, n, k1, k2, tip, nx):
    if k1 == 1 and k2 == 0:
        bk = b_prev
        tip = False
        return h_prev, b_prev, bk, tip
    elif k1 == 0 and k2 == 1 and tip == True:
        bk = b_prev
        return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
    elif k1 == 1 and k2 == 1 and tip == True:
```

```
        bk = b_prev
        return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
    else:
        bk = b_prev
        tip = False
        return h_prev, b_prev, bk, tip  # 当 k1 = 0, k2 = 0 时，保持不变

def calculate_vk(hk, bk, p, cp3, price3, eta1, p1, eta2, p2, check1, cp1, check2, cp2, break_cost, change_cost,
                 checked1, checked2, k1, k2, bk_old, biaoji, total_cost, nx):
    if check1 and not checked1 and biaoji == 0:
        total_cost += (hk + bk) / (1 - eta1) * cp1
        checked1 = True
    if check2 and not checked2 and biaoji == 0:
        total_cost += (hk + bk) / (1 - eta2) * cp2
        checked2 = True

    if k1 == 1 and k2 == 0 and biaoji == 0:
        biaoji += 1
        total_cost += (hk + bk) * cp3
    elif k1 == 0 and k2 == 1:
        total_cost += bk_old * break_cost + bk_old * change_cost + bk * nx * change_cost
    elif k1 == 1 and k2 == 1 and biaoji == 0:
        biaoji += 1
        total_cost += (hk + bk) * cp3 + bk_old * break_cost + bk_old * nx * change_cost
    else:
        total_cost += bk_old * change_cost

    return hk * p - total_cost, checked1, checked2, total_cost

def monte_carlo_simulation(iterations, step=2):
    eta1 = 0.1
    p1 = 4
    cp1 = 2
    eta2 = 0.1
    p2 = 18
    cp2 = 3
    eta0 = 0.1
    price3 = 6
    cp3 = 3
```

```
    p = 56
    change_cost = 6
    break_cost = 5
    n = eta0 + eta1 + eta2
    norm = n
    n1 = 100
    n2 = 100

    max_overall_value = float('-inf')
    best_overall_path = None
    max_values_per_iteration = []  # 用来记录每次模拟的最大值

    for _ in range(iterations):
        f = [0]  # 初始化 f(0) = 0
        biaoji = 0
        bk = 0
        max_v = float('-inf')
        path = [0] * step * 4  # 记录随机生成的路径
        k = 1
        while k <= step:
            # 随机生成 check1, check2, k1, k2 的路径
            check1 = random.choice([0, 1])
            check2 = random.choice([0, 1])
            s1 = -(check1) * n1 * eta1 + n1
            s2 = -(check2) * n2 * eta2 + n2
            hk = min(s1, s2)

            k1 = random.choice([0, 1])
            k2 = random.choice([0, 1])

            if k2 == 0:
                tip = False
            total_cost = hk * price3 + n1 * p1 + n2 * p2
            t1 = 0
            t2 = 0
                n = norm
                checked1 = False
                checked2 = False
                tip = True

                if check1 == 1 and t1 == 0:
                    t1 += 1
                    n = n - eta1
                if check2 == 1 and t2 == 0:
```

```
                    t2 += 1
                    n = n - eta2

                nx = ((1 - check1) * eta1 + (1 - check2) * eta2) /
   (
                        (1 - check1) * eta1 + (1 - check2) * eta2 +
    eta0) + eta0

                hk_new, bk_new, bk_old, tip = calculate_hk_bk(hk *
   (1 - n), hk * n, n, k1, k2, tip, nx)

                vk, new_checked1, new_checked2, total_cost = calcul
   ate_vk(hk_new,
          bk_new, p, cp3, price3, eta1, p1,
          eta2, p2, check1, cp1, check2, cp2,
          break_cost, change_cost, checked1,
          checked2, k1, k2, bk_old, biaoji,
          total_cost, nx)

                new_value = vk

                if new_value > max_v:
                    max_v = round(new_value)
                    path[4 * k - 4] = check1
                    path[4 * k - 3] = check2
                    path[4 * k - 2] = k1
                    path[4 * k - 1] = k2
                    bk = bk_new

                f.append(max_v)
                k += 1

            max_values_per_iteration.append(max_v)  # 记录每次迭代的
   最大值

            # 记录整体最大值和最佳路径
            if max_v > max_overall_value:
                max_overall_value = max_v
                best_overall_path = path.copy()
```

```
143.
144.         return max_overall_value, best_overall_path, max_values_per
    _iteration
145.
146.     def main():
147.         random.seed(39)  # 设置随机数种子
148.         iterations = 100  # 蒙特卡洛模拟的次数
149.         max_value, best_path, max_values_per_iteration = monte_carl
    o_simulation(iterations)
150.
151.         print("情况四模拟最大值：", max_value)
152.         print("最佳路径：", best_path)
153.
154.         # 使用 matplotlib 绘制最大值的变化趋势
155.         plt.figure(figsize=(12, 7))
156.         plt.plot(range(iterations), max_values_per_iteration, label
    ='最大值变化')
157.
158.
159.         plt.axhline(y=1650, color='r', linestyle='--
    ', label='1650')
160.
161.         # 设置中文标签
162.         plt.xlabel('迭代次数', fontsize=14)
163.         plt.ylabel('最大值', fontsize=14)
164.         plt.title('情况一每次迭代的最大值', fontsize=16)
165.         plt.legend(fontsize=12)
166.         plt.xticks(fontsize=12)
167.         plt.yticks(fontsize=12)
168.         plt.grid(True, linestyle='--', alpha=0.7)
169.         plt.tight_layout()
170.         plt.savefig('1.png')
171.         plt.show()
172.
173.     if __name__ == "__main__":
174.         main()
```

附录 6

T3

```
175.     #第一种情况进行蒙特卡洛检验
176.     import random
177.     import matplotlib.pyplot as plt
178.     from matplotlib import font_manager
179.
180.
```

```
181.   plt.rcParams['axes.unicode_minus'] = False
182.
183.   # 尝试加载系统中的中文字体
184.   try:
185.       font_path = font_manager.findfont(font_manager.FontProperti
   es(family='SimHei'))
186.       prop = font_manager.FontProperties(fname=font_path)
187.       plt.rcParams['font.family'] = prop.get_name()
188.   except:
189.       print("无法找到 SimHei 字体，将使用系统默认字体。中文可能无法正
   确显示。")
190.
191.   def calculate_hk_bk(h_prev, b_prev, n, k1, k2, tip, nx):
192.       if k1 == 1 and k2 == 0:
193.           bk = b_prev
194.           tip = False
195.           return h_prev, b_prev, bk, tip
196.       elif k1 == 0 and k2 == 1 and tip == True:
197.           bk = b_prev
198.           return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
199.       elif k1 == 1 and k2 == 1 and tip == True:
200.           bk = b_prev
201.           return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
202.       else:
203.           bk = b_prev
204.           tip = False
205.           return h_prev, b_prev, bk, tip  # 当 k1 = 0, k2 = 0 时,
   保持不变
206.
207.   def calculate_vk(hk, bk, p, cp3, price3, eta1, p1, eta2, p2, ch
   eck1, cp1, check2, cp2, break_cost, change_cost,
208.                    checked1, checked2, k1, k2, bk_old, biaoji, to
   tal_cost, nx):
209.       if check1 and not checked1 and biaoji == 0:
210.           total_cost += (hk + bk) / (1 - eta1) * cp1
211.           checked1 = True
212.       if check2 and not checked2 and biaoji == 0:
213.           total_cost += (hk + bk) / (1 - eta2) * cp2
214.           checked2 = True
215.
216.       if k1 == 1 and k2 == 0 and biaoji == 0:
217.           biaoji += 1
218.           total_cost += (hk + bk) * cp3
219.       elif k1 == 0 and k2 == 1:
```

```
            total_cost += bk_old * break_cost + bk_old * change_cos
    t + bk * nx * change_cost
        elif k1 == 1 and k2 == 1 and biaoji == 0:
            biaoji += 1
            total_cost += (hk + bk) * cp3 + bk_old * break_cost + b
    k_old * nx * change_cost
        else:
            total_cost += bk_old * change_cost

        return hk * p - total_cost, checked1, checked2, total_cost

    def monte_carlo_simulation(iterations, step=2):
        eta1 = 0.1
        p1 = 4
        cp1 = 2
        eta2 = 0.1
        p2 = 18
        cp2 = 3
        eta0 = 0.1
        price3 = 6
        cp3 = 3
        p = 56
        change_cost = 6
        break_cost = 5
        n = eta0 + eta1 + eta2
        norm = n
        n1 = 100
        n2 = 100

        max_overall_value = float('-inf')
        best_overall_path = None
        max_values_per_iteration = []  # 用来记录每次模拟的最大值

        for _ in range(iterations):
            f = [0]  # 初始化 f(0) = 0
            biaoji = 0
            bk = 0
            max_v = float('-inf')
            path = [0] * step * 4  # 记录随机生成的路径
            k = 1
            while k <= step:
                # 随机生成 check1, check2, k1, k2 的路径
                check1 = random.choice([0, 1])
                check2 = random.choice([0, 1])
```

```
                s1 = -(check1) * n1 * eta1 + n1
                s2 = -(check2) * n2 * eta2 + n2
                hk = min(s1, s2)

                k1 = random.choice([0, 1])
                k2 = random.choice([0, 1])

                if k2 == 0:
                    tip = False
                total_cost = hk * price3 + n1 * p1 + n2 * p2
                t1 = 0
                t2 = 0
                n = norm
                checked1 = False
                checked2 = False
                tip = True

                if check1 == 1 and t1 == 0:
                    t1 += 1
                    n = n - eta1
                if check2 == 1 and t2 == 0:
                    t2 += 1
                    n = n - eta2

                nx = ((1 - check1) * eta1 + (1 - check2) * eta2) /
   (
                        (1 - check1) * eta1 + (1 - check2) * eta2 +
    eta0) + eta0

                hk_new, bk_new, bk_old, tip = calculate_hk_bk(hk *
   (1 - n), hk * n, n, k1, k2, tip, nx)

                vk, new_checked1, new_checked2, total_cost = calcul
   ate_vk(hk_new,
          bk_new, p, cp3, price3, eta1, p1,
          eta2, p2, check1, cp1, check2, cp2,
          break_cost, change_cost, checked1,
          checked2, k1, k2, bk_old, biaoji,
          total_cost, nx)
```

```
297.
298.                 new_value = vk
299.
300.                 if new_value > max_v:
301.                     max_v = round(new_value)
302.                     path[4 * k - 4] = check1
303.                     path[4 * k - 3] = check2
304.                     path[4 * k - 2] = k1
305.                     path[4 * k - 1] = k2
306.                     bk = bk_new
307.
308.                 f.append(max_v)
309.                 k += 1
310.
311.             max_values_per_iteration.append(max_v)  # 记录每次迭代的
    最大值
312.
313.             # 记录整体最大值和最佳路径
314.             if max_v > max_overall_value:
315.                 max_overall_value = max_v
316.                 best_overall_path = path.copy()
317.
318.         return max_overall_value, best_overall_path, max_values_per
    _iteration
319.
320.     def main():
321.         random.seed(39)  # 设置随机数种子
322.         iterations = 100  # 蒙特卡洛模拟的次数
323.         max_value, best_path, max_values_per_iteration = monte_carl
    o_simulation(iterations)
324.
325.         print("情况四模拟最大值：", max_value)
326.         print("最佳路径：", best_path)
327.
328.         # 使用 matplotlib 绘制最大值的变化趋势
329.         plt.figure(figsize=(12, 7))
330.         plt.plot(range(iterations), max_values_per_iteration, label
    ='最大值变化')
331.
332.
333.         plt.axhline(y=1650, color='r', linestyle='--
    ', label='1650')
334.
335.         # 设置中文标签
```

```
        plt.xlabel('迭代次数', fontsize=14)
        plt.ylabel('最大值', fontsize=14)
        plt.title('情况一每次迭代的最大值', fontsize=16)
        plt.legend(fontsize=12)
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.savefig('1.png')
        plt.show()

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

附录 7

T3

```
    #第一种情况进行蒙特卡洛检验
    import random
    import matplotlib.pyplot as plt
    from matplotlib import font_manager


    plt.rcParams['axes.unicode_minus'] = False

    # 尝试加载系统中的中文字体
    try:
        font_path = font_manager.findfont(font_manager.FontProperti
    es(family='SimHei'))
        prop = font_manager.FontProperties(fname=font_path)
        plt.rcParams['font.family'] = prop.get_name()
    except:
        print("无法找到 SimHei 字体，将使用系统默认字体。中文可能无法正
    确显示。")

    def calculate_hk_bk(h_prev, b_prev, n, k1, k2, tip, nx):
        if k1 == 1 and k2 == 0:
            bk = b_prev
            tip = False
            return h_prev, b_prev, bk, tip
        elif k1 == 0 and k2 == 1 and tip == True:
            bk = b_prev
            return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
        elif k1 == 1 and k2 == 1 and tip == True:
            bk = b_prev
```

```
374.             return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
375.         else:
376.             bk = b_prev
377.             tip = False
378.             return h_prev, b_prev, bk, tip  # 当 k1 = 0, k2 = 0 时，保持不变
379.
380.     def calculate_vk(hk, bk, p, cp3, price3, eta1, p1, eta2, p2, check1, cp1, check2, cp2, break_cost, change_cost,
381.                      checked1, checked2, k1, k2, bk_old, biaoji, total_cost, nx):
382.         if check1 and not checked1 and biaoji == 0:
383.             total_cost += (hk + bk) / (1 - eta1) * cp1
384.             checked1 = True
385.         if check2 and not checked2 and biaoji == 0:
386.             total_cost += (hk + bk) / (1 - eta2) * cp2
387.             checked2 = True
388.
389.         if k1 == 1 and k2 == 0 and biaoji == 0:
390.             biaoji += 1
391.             total_cost += (hk + bk) * cp3
392.         elif k1 == 0 and k2 == 1:
393.             total_cost += bk_old * break_cost + bk_old * change_cost + bk * nx * change_cost
394.         elif k1 == 1 and k2 == 1 and biaoji == 0:
395.             biaoji += 1
396.             total_cost += (hk + bk) * cp3 + bk_old * break_cost + bk_old * nx * change_cost
397.         else:
398.             total_cost += bk_old * change_cost
399.
400.         return hk * p - total_cost, checked1, checked2, total_cost
401.
402.     def monte_carlo_simulation(iterations, step=2):
403.         eta1 = 0.1
404.         p1 = 4
405.         cp1 = 2
406.         eta2 = 0.1
407.         p2 = 18
408.         cp2 = 3
409.         eta0 = 0.1
410.         price3 = 6
411.         cp3 = 3
412.         p = 56
```

```
        change_cost = 6
        break_cost = 5
        n = eta0 + eta1 + eta2
        norm = n
        n1 = 100
        n2 = 100

        max_overall_value = float('-inf')
        best_overall_path = None
        max_values_per_iteration = []  # 用来记录每次模拟的最大值

        for _ in range(iterations):
            f = [0]  # 初始化 f(0) = 0
            biaoji = 0
            bk = 0
            max_v = float('-inf')
            path = [0] * step * 4  # 记录随机生成的路径
            k = 1
            while k <= step:
                # 随机生成 check1, check2, k1, k2 的路径
                check1 = random.choice([0, 1])
                check2 = random.choice([0, 1])
                s1 = -(check1) * n1 * eta1 + n1
                s2 = -(check2) * n2 * eta2 + n2
                hk = min(s1, s2)

                k1 = random.choice([0, 1])
                k2 = random.choice([0, 1])

                if k2 == 0:
                    tip = False
                total_cost = hk * price3 + n1 * p1 + n2 * p2
                t1 = 0
                t2 = 0
                n = norm
                checked1 = False
                checked2 = False
                tip = True

                if check1 == 1 and t1 == 0:
                    t1 += 1
                    n = n - eta1
                if check2 == 1 and t2 == 0:
                    t2 += 1
```

```
                    n = n - eta2

                nx = ((1 - check1) * eta1 + (1 - check2) * eta2) /
(
                        (1 - check1) * eta1 + (1 - check2) * eta2 +
 eta0) + eta0

                hk_new, bk_new, bk_old, tip = calculate_hk_bk(hk *
(1 - n), hk * n, n, k1, k2, tip, nx)

                vk, new_checked1, new_checked2, total_cost = calcul
ate_vk(hk_new,
          bk_new, p, cp3, price3, eta1, p1,
          eta2, p2, check1, cp1, check2, cp2,
          break_cost, change_cost, checked1,
          checked2, k1, k2, bk_old, biaoji,
          total_cost, nx)

                new_value = vk

                if new_value > max_v:
                    max_v = round(new_value)
                    path[4 * k - 4] = check1
                    path[4 * k - 3] = check2
                    path[4 * k - 2] = k1
                    path[4 * k - 1] = k2
                    bk = bk_new

                f.append(max_v)
                k += 1

            max_values_per_iteration.append(max_v)  # 记录每次迭代的
最大值

            # 记录整体最大值和最佳路径
            if max_v > max_overall_value:
                max_overall_value = max_v
                best_overall_path = path.copy()

```

```
        return max_overall_value, best_overall_path, max_values_per_iteration

    def main():
        random.seed(39)  # 设置随机数种子
        iterations = 100  # 蒙特卡洛模拟的次数
        max_value, best_path, max_values_per_iteration = monte_carlo_simulation(iterations)

        print("情况四模拟最大值：", max_value)
        print("最佳路径：", best_path)

        # 使用 matplotlib 绘制最大值的变化趋势
        plt.figure(figsize=(12, 7))
        plt.plot(range(iterations), max_values_per_iteration, label='最大值变化')


        plt.axhline(y=1650, color='r', linestyle='--', label='1650')

        # 设置中文标签
        plt.xlabel('迭代次数', fontsize=14)
        plt.ylabel('最大值', fontsize=14)
        plt.title('情况一每次迭代的最大值', fontsize=16)
        plt.legend(fontsize=12)
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.savefig('1.png')
        plt.show()

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

附录 8

T4-1

```
%计算第一问中所得最优抽样方案对零配件次品率的计算
% 参数定义
N = 300;  % 总体数量
n = 72;   % 抽样个数
c = 6;    % 最大不合格品数
num_simulations = 1000;  % 模拟次数
```

```
p_values = zeros(1, num_simulations);  % 用于存储每次模拟的次品率

% 开始模拟 1000 次
for sim = 1:num_simulations
    % 随机生成一个次品率 p，假设在 0 到 0.2 之间
    p = rand() * 0.2;

    % 随机生成 N 个零件，其中次品率为 p
    population = rand(1, N) < p;

    % 从总体中随机抽取 n 个样本
    sample = randsample(population, n);

    % 统计样本中的不合格品数
    num_defects = sum(sample);

    % 如果不合格品数 <= c，则记录该次品率
    if num_defects <= c
        p_values(sim) = p;
    else
        p_values(sim) = NaN;  % 如果不接受，则记为 NaN
    end
end

% 删除 NaN 值，统计所有接受批次的次品率
accepted_p_values = p_values(~isnan(p_values));

% 输出 1000 次模拟中满足抽样方案的次品率
disp('满足抽样方案的次品率:');
disp(accepted_p_values);

% 统计并输出接受率
acceptance_rate = length(accepted_p_values) / num_simulations * 100
;
disp(['接受率为: ', num2str(acceptance_rate), '%']);

% 画出满足抽样方案的次品率分布的直方图
figure;
histogram(accepted_p_values, 20, 'Normalization', 'probability');
xlabel('次品率');
ylabel('频率');
title('满足抽样方案的次品率分布');
grid on;
```

附录 8

T4-2

```
import random
import numpy as np
from collections import Counter


# 不同决策下成品合格与不合格的量
def calculate_hk_bk(h_prev, b_prev, n, k1, k2, tip, nx):
    if k1 == 1 and k2 == 0:
        bk = b_prev
        tip = False
        return h_prev, b_prev, bk, tip
    elif k1 == 0 and k2 == 1 and tip == True:
        bk = b_prev
        return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
    elif k1 == 1 and k2 == 1 and tip == True:
        bk = b_prev
        return h_prev + b_prev * (1 - nx), b_prev * nx, bk, tip
    else:
        bk = b_prev
        tip = False
        return h_prev, b_prev, bk, tip  # 当 k1 = 0, k2 = 0 时，保持不变

def calculate_vk(hk, bk, p, cp3, price3, eta1, p1, eta2, p2, check1, cp1, check2, cp2, break_cost, change_cost,
                 checked1, checked2, k1, k2, bk_old, biaoji, total_cost, nx):
    if check1 and not checked1 and biaoji == 0:
        total_cost += (hk + bk) / (1 - eta1) * cp1
        checked1 = True
    if check2 and not checked2 and biaoji == 0:
        total_cost += (hk + bk) / (1 - eta2) * cp2
        checked2 = True

    if k1 == 1 and k2 == 0 and biaoji == 0:
        biaoji += 1
        total_cost += (hk + bk) * cp3
    elif k1 == 0 and k2 == 1:
```

```
        total_cost += bk_old * break_cost + bk_old * change_cost + bk * nx * change_cost
    elif k1 == 1 and k2 == 1 and biaoji == 0:
        biaoji += 1
        total_cost += (hk + bk) * cp3 + bk_old * break_cost + bk_old * nx * change_cost
    else:
        total_cost += bk_old * change_cost

    return hk * p - total_cost, checked1, checked2, total_cost


def simulate_once(pass_rates, probabilities, p1, cp1, p2, cp2, price3, cp3, p, change_cost, break_cost, n1, n2):
    f = [0]
    biaoji = 0
    bk = 0
    max_v = float('-inf')
    step = 2
    best_path = [0] * step * 4
    k = 1
    checked1 = False
    checked2 = False

    while k <= step:
        eta1 = np.random.choice(pass_rates, p=probabilities)
        eta2 = np.random.choice(pass_rates, p=probabilities)
        eta0 = min(eta1, eta2)
        n = eta0 + eta1 + eta2
        norm = n

        m = k
        for check1 in [0, 1]:
            for check2 in [0, 1]:
                s1 = -(check1) * n1 * eta1 + n1
                s2 = -(check2) * n2 * eta2 + n2
                hk = min(s1, s2)
                for k1 in [0, 1]:
                    for k2 in [0, 1]:
                        if k2 == 0:
                            tip = False
```

```
74.
75.                         total_cost = hk * price3 + n1 *
   p1 + n2 * p2
76.                         t1 = 0
77.                         t2 = 0
78.                         n = norm
79.                         tip = True
80.
81.                         if check1 == 1 and t1 == 0:
82.                             t1 += 1
83.                             n = n - eta1
84.                             m = 1
85.                         if check2 == 1 and t2 == 0:
86.                             t2 += 1
87.                             m = 1
88.                             n = n - eta2
89.
90.                         nx = ((1 - check1) * eta1 + (1 -
    check2) * eta2) / (
91.                                   (1 - check1) * eta1 + (1
    - check2) * eta2 + eta0) + eta0
92.
93.                         hk_new, bk_new, bk_old, tip = ca
   lculate_hk_bk(hk * (1 - n), hk * n, n, k1, k2, tip, nx)
94.
95.                         vk, new_checked1, new_checked2,
   total_cost = calculate_vk(hk_new,
96.
                               bk_new, p, cp3, price3, eta1,
   p1,
97.
                               eta2, p2, check1, cp1, check2,
    cp2,
98.
                               break_cost, change_cost, check
   ed1,
99.
                               checked2, k1, k2, bk_old, biao
   ji,
100.
                                  total_cost, nx)
101.                            new_value = vk
102.
103.                            if new_value > max_v:
```

```
                                max_v = round(new_value)
                                best_path[4 * m - 4] = check1
                                best_path[4 * m - 3] = check2
                                best_path[4 * m - 2] = k1
                                best_path[4 * m - 1] = k2
                                bk = bk_new

                            if k2 == 0:
                                continue

                            f.append(max_v)
            k += 1
        return max_v, best_path

    def main():
        # 设置随机数种子
        seed = 40
        random.seed(seed)
        np.random.seed(seed)

        # 定义参数
        pass_rates = [0.004955483, 0.013928847, 0.02290221, 0.031875574, 0.040848937,
                      0.049822301, 0.058795665, 0.067769028, 0.076742392, 0.085715756,
                      0.094689119, 0.103662483, 0.112635846, 0.12160921, 0.130582574,
                      0.139555937, 0.148529301, 0.157502665, 0.166476028, 0.175449392]
        probabilities = [0.090707909, 0.134955668, 0.095132684, 0.073008806, 0.086283132,
                         0.095132684, 0.081858357, 0.079645969, 0.059734477, 0.030973433,
                         0.030973433, 0.030973433, 0.026548655, 0.033185821, 0.013274329,
                         0.015486715, 0.006637163, 0.008849552, 0.004424777, 0.002212389]
        probabilities = np.array(probabilities) / np.sum(probabilities)
        #其余情况修改对应数据即可
        p1, cp1 = 4, 2
        p2, cp2 = 18, 3
```

```
    price3, cp3 = 6, 3
    p = 56
    change_cost, break_cost = 6, 5
    n1, n2 = 100, 100


    # 运行100次模拟
    paths = []
    for _ in range(100):
        _, path = simulate_once(pass_rates, probabilities, p1, cp1, p2, cp2, price3, cp3, p, change_cost,
                                break_cost, n1, n2)
        paths.append(tuple(path))

    # 路径频率统计
    path_counts = Counter(paths)

    # 获取最常见的路径及其频率
    best_path, count = path_counts.most_common(1)[0]
    probability = count / 100

    print("最佳路径:")
    print(f"路径: {list(best_path)}")
    print(f"出现概率: {probability:.2%}")

    print("\n决策解释:")
    for j in range(0, len(best_path), 4):
        step = j // 4 + 1
        check1, check2, k1, k2 = best_path[j:j + 4]
        print(f"步骤 {step}:")
        print(f"  检查零件1: {'是' if check1 == 1 else '否'}")
        print(f"  检查零件2: {'是' if check2 == 1 else '否'}")
        print(f"  检查成品: {'是' if k1 == 1 else '否'}")
        print(f"  处理不合格品: {'是' if k2 == 1 else '否'}")


if __name__ == "__main__":
    main()
import itertools
import numpy as np
```

```
import matplotlib.pyplot as plt
from typing import List, Tuple
from matplotlib import font_manager
from collections import Counter

plt.rcParams['axes.unicode_minus'] = False

# 尝试加载系统中的中文字体
try:
    font_path = font_manager.findfont(font_manager.FontProperties(family='SimHei'))
    prop = font_manager.FontProperties(fname=font_path)
    plt.rcParams['font.family'] = prop.get_name()
except:
    print("无法找到 SimHei 字体，将使用系统默认字体。中文可能无法正确显示。")

class Component:
    def __init__(self, defect_rate: float, purchase_price: float, inspection_cost: float, initial_quantity: int):
        self.defect_rate = defect_rate
        self.purchase_price = purchase_price
        self.inspection_cost = inspection_cost
        self.initial_quantity = initial_quantity

class SemiProduct:
    def __init__(self, defect_rate: float, assembly_cost: float, inspection_cost: float, disassembly_cost: float):
        self.defect_rate = defect_rate
        self.assembly_cost = assembly_cost
        self.inspection_cost = inspection_cost
        self.disassembly_cost = disassembly_cost

class FinalProduct:
    def __init__(self, defect_rate: float, assembly_cost: float, inspection_cost: float,
                 disassembly_cost: float, market_price: float, replacement_cost: float):
        self.defect_rate = defect_rate
        self.assembly_cost = assembly_cost
        self.inspection_cost = inspection_cost
```

```
211.          self.disassembly_cost = disassembly_cost
212.          self.market_price = market_price
213.          self.replacement_cost = replacement_cost
214.
215.  class ProductionLine:
216.      def __init__(self, components: List[Component], s
    emi_products: List[SemiProduct], final_product: FinalPro
    duct):
217.          self.components = components
218.          self.semi_products = semi_products
219.          self.final_product = final_product
220.
221.      def calculate_components(self, inspect_decisions:
     List[bool]) -> List[Tuple[float, float]]:
222.          results = []
223.          for component, inspect in zip(self.components
    , inspect_decisions):
224.              total = component.initial_quantity
225.              if inspect:
226.                  good = total * (1 - component.defect_
    rate)
227.                  bad = 0
228.              else:
229.                  good = total * (1 - component.defect_
    rate)
230.                  bad = total * component.defect_rate
231.              results.append((good, bad))
232.          return results
233.
234.      def assemble_semi_products(self, component_quanti
    ties: List[Tuple[float, float]],
235.                                 inspect_decisions: Lis
    t[bool]) -> List[Tuple[float, float]]:
236.          results = []
237.          for i, (semi_product, inspect) in enumerate(z
    ip(self.semi_products, inspect_decisions)):
238.              if i < 2:
239.                  components_for_semi = component_quant
    ities[i * 3:(i + 1) * 3]
240.              else:
241.                  components_for_semi = component_quant
    ities[6:]
242.              good_components = min([q[0] for q in comp
    onents_for_semi])
```

```
243.                 assembled = good_components * (1 - semi_p
    roduct.defect_rate)
244.                 defective = good_components * semi_produc
    t.defect_rate
245.                 if inspect:
246.                     results.append((assembled, 0))
247.                 else:
248.                     results.append((assembled, defective)
    )
249.             return results
250.
251.         def assemble_final_product(self, semi_product_qua
    ntities: List[Tuple[float, float]],
252.                                    inspect_decision: bool
    ) -> Tuple[float, float]:
253.             good_semi_products = min([q[0] for q in semi_
    product_quantities])
254.             assembled = good_semi_products * (1 - self.fi
    nal_product.defect_rate)
255.             defective = good_semi_products * self.final_p
    roduct.defect_rate
256.             if inspect_decision:
257.                 return assembled, 0
258.             else:
259.                 return assembled, defective
260.
261.         def disassemble_and_inspect_semi_products(self, s
    emi_product_quantities: List[Tuple[float, float]],
262.                                                   disasse
    mbly_decisions: List[bool],
263.                                                   post_di
    sassembly_inspect_decisions: List[bool]) -> List[Tuple[f
    loat, float]]:
264.             reusable_components = []
265.             for (good, bad), disassemble, inspect, semi_p
    roduct in zip(semi_product_quantities, disassembly_decis
    ions,
266.
                     post_disassembly_inspect_decisions,
267.
                     self.semi_products):
268.                 if disassemble:
269.                     disassembled = bad * (1 - semi_produc
    t.defect_rate)
```

```
                    if inspect:
                        reusable_components.extend([(disassembled, 0)] * 3)
                    else:
                        reusable_components.extend([(disassembled * (1 - self.components[0].defect_rate),
                                                     disassembled * self.components[0].defect_rate)] * 3)
                else:
                    reusable_components.extend([(0, 0)] * 3)
            return reusable_components

        def disassemble_and_inspect_final_product(self, final_product_quantity: Tuple[float, float],
                                                  disassembly_decision: bool,
                                                  post_disassembly_inspect_decision: bool) -> List[Tuple[float, float]]:
            good, bad = final_product_quantity
            if disassembly_decision:
                disassembled = bad * (1 - self.final_product.defect_rate)
                if post_disassembly_inspect_decision:
                    return [(disassembled, 0)] * 3
                else:
                    return [(disassembled * (1 - self.semi_products[0].defect_rate),
                             disassembled * self.semi_products[0].defect_rate)] * 3
            else:
                return [(0, 0)] * 3

        def calculate_costs(self, component_decisions: List[bool], semi_product_decisions: List[bool],
                            semi_product_disassembly: List[bool], semi_product_post_disassembly_inspect: List[bool],
                            final_product_decision: bool, final_product_disassembly: bool,
                            final_product_post_disassembly_inspect: bool, final_product_quantity: float) -> float:
```

```
297.         total_cost = 0
298.
299.         for component, inspect in zip(self.components
    , component_decisions):
300.             total_cost += component.purchase_price *
    component.initial_quantity
301.             if inspect:
302.                 total_cost += component.inspection_co
    st * component.initial_quantity
303.
304.         for semi_product, inspect, disassemble, post_
    inspect in zip(self.semi_products, semi_product_decision
    s,
305.
                 semi_product_disassembly,
306.
                 semi_product_post_disassembly_inspect):
307.             total_cost += semi_product.assembly_cost
    * final_product_quantity
308.             if inspect:
309.                 total_cost += semi_product.inspection
    _cost * final_product_quantity
310.             if disassemble:
311.                 total_cost += semi_product.disassembl
    y_cost * final_product_quantity * semi_product.defect_ra
    te
312.                 if post_inspect:
313.                     total_cost += semi_product.inspec
    tion_cost * final_product_quantity * semi_product.defect
    _rate
314.
315.         total_cost += self.final_product.assembly_cos
    t * final_product_quantity
316.         if final_product_decision:
317.             total_cost += self.final_product.inspecti
    on_cost * final_product_quantity
318.         if final_product_disassembly:
319.             total_cost += self.final_product.disassem
    bly_cost * final_product_quantity * self.final_product.d
    efect_rate
320.             if final_product_post_disassembly_inspect
    :
```

```
                total_cost += self.final_product.inspection_cost * final_product_quantity * self.final_product.defect_rate
            else:
                total_cost += self.final_product.replacement_cost * (final_product_quantity * self.final_product.defect_rate)

        return total_cost

    def calculate_revenue(self, final_product_quantity: float) -> float:
        return final_product_quantity * (1 - self.final_product.defect_rate) * self.final_product.market_price

    def calculate_profit(self, revenue: float, cost: float) -> float:
        return revenue - cost

def monte_carlo_simulation(production_line: ProductionLine, num_simulations: int = 1000, seed: int = 42):
    np.random.seed(seed)
    profits = []
    max_profits = []
    decisions = []
    max_profit_so_far = float('-inf')

    for _ in range(num_simulations):
        component_decisions = [bool(np.random.choice([True, False])) for _ in production_line.components]
        semi_product_inspect_decisions = [bool(np.random.choice([True, False])) for _ in production_line.semi_products]
        semi_product_disassembly = [bool(np.random.choice([True, False])) for _ in production_line.semi_products]
        semi_product_post_disassembly_inspect = [bool(np.random.choice([True, False])) for _ in production_line.semi_products]
        final_product_inspect_decision = bool(np.random.choice([True, False]))
        final_product_disassembly = bool(np.random.choice([True, False]))
```

```
347.         final_product_post_disassembly_inspect = bool
   (np.random.choice([True, False]))
348.
349.         component_quantities = production_line.calcul
   ate_components(component_decisions)
350.         semi_product_quantities = production_line.ass
   emble_semi_products(component_quantities,
351.
                         semi_product_inspect_decisions)
352.
353.         reusable_components = production_line.disasse
   mble_and_inspect_semi_products(
354.             semi_product_quantities, semi_product_dis
   assembly, semi_product_post_disassembly_inspect)
355.         for i in range(len(component_quantities)):
356.             component_quantities[i] = (component_quan
   tities[i][0] + reusable_components[i][0],
357.                                        component_quan
   tities[i][1] + reusable_components[i][1])
358.         semi_product_quantities = production_line.ass
   emble_semi_products(component_quantities,
359.
                         semi_product_inspect_decisions)
360.
361.         final_product_quantity, final_product_defecti
   ve = production_line.assemble_final_product(
362.             semi_product_quantities, final_product_in
   spect_decision)
363.
364.         if final_product_disassembly:
365.             reusable_semi_products = production_line.
   disassemble_and_inspect_final_product(
366.                 (final_product_quantity, final_produc
   t_defective),
367.                 final_product_disassembly,
368.                 final_product_post_disassembly_inspec
   t)
369.             for i in range(len(semi_product_quantitie
   s)):
370.                 semi_product_quantities[i] = (semi_pr
   oduct_quantities[i][0] + reusable_semi_products[i][0],
371.                                               semi_pr
   oduct_quantities[i][1] + reusable_semi_products[i][1])
```

```
372.             final_product_quantity, final_product_def
    ective = production_line.assemble_final_product(
373.                 semi_product_quantities, final_produc
    t_inspect_decision)
374.
375.         cost = production_line.calculate_costs(
376.             component_decisions, semi_product_inspect
    _decisions,
377.             semi_product_disassembly, semi_product_po
    st_disassembly_inspect,
378.             final_product_inspect_decision, final_pro
    duct_disassembly,
379.             final_product_post_disassembly_inspect, f
    inal_product_quantity)
380.         revenue = production_line.calculate_revenue(f
    inal_product_quantity)
381.         profit = production_line.calculate_profit(rev
    enue, cost)
382.
383.         profits.append(profit)
384.         max_profit_so_far = max(max_profit_so_far, pr
    ofit)
385.         max_profits.append(max_profit_so_far)
386.
387.         decisions.append((
388.             tuple(component_decisions),
389.             tuple(semi_product_post_disassembly_inspe
    ct),
390.             tuple(semi_product_disassembly),
391.             tuple(semi_product_inspect_decisions),
392.             final_product_disassembly,
393.             final_product_post_disassembly_inspect,
394.             final_product_inspect_decision
395.
396.         ))
397.
398.     return profits, max_profits, decisions
399.
400. def run_simulation(components, semi_products, final_p
    roduct, num_simulations, seed):
401.     production_line = ProductionLine(components, semi
    _products, final_product)
402.     profits, _, decisions = monte_carlo_simulation(pr
    oduction_line, num_simulations, seed)
```

```
403.      return max(profits), decisions[profits.index(max(
   profits))]
404.
405.
406.  def visualize_max_profits(max_profits):
407.      plt.figure(figsize=(12, 6))
408.
409.      # 绘制每次模拟的最大利润
410.      plt.plot(range(1, len(max_profits) + 1), max_prof
   its, marker='o', alpha=0.5, label='每次模拟的最大利润')
411.
412.      # 计算并绘制平均最大利润线
413.      average_profits = np.mean(max_profits)
414.      plt.axhline(y=average_profits, color='r', linesty
   le='--', label='平均最大利润')
415.
416.      plt.title('最大利润变化趋势及平均值', fontsize=14)
417.      plt.xlabel('模拟次数', fontsize=12)
418.      plt.ylabel('最大利润', fontsize=12)
419.      plt.grid(True, linestyle='--', alpha=0.7)
420.      plt.legend()
421.      plt.tight_layout()
422.      plt.savefig('max_profits_trend_with_average.png')
423.      plt.show()
424.
425.  def main():
426.      pass_rates = [0.004955483, 0.013928847, 0.0229022
   1, 0.031875574, 0.040848937,
427.                    0.049822301, 0.058795665, 0.0677690
   28, 0.076742392, 0.085715756,
428.                    0.094689119, 0.103662483, 0.1126358
   46, 0.12160921, 0.130582574,
429.                    0.139555937, 0.148529301, 0.1575026
   65, 0.166476028, 0.175449392]
430.      probabilities = [0.090707909, 0.134955668, 0.0951
   32684, 0.073008806, 0.086283132,
431.                       0.095132684, 0.081858357, 0.0796
   45969, 0.059734477, 0.030973433,
432.                       0.030973433, 0.030973433, 0.0265
   48655, 0.033185821, 0.013274329,
433.                       0.015486715, 0.006637163, 0.0088
   49552, 0.004424777, 0.002212389]
434.
435.      probabilities = np.array(probabilities)
```

```
436.      probabilities = probabilities / np.sum(probabilit
   ies)
437.
438.      num_simulations = 10000
439.      overall_max_profits = []
440.      all_decisions = []
441.
442.      for i in range(100):
443.          np.random.seed(i)
444.
445.          component_defect_rates = np.random.choice(pas
   s_rates, size=8, p=probabilities)
446.          min_defect_rate = np.min(component_defect_rat
   es)
447.
448.          components = [
449.              Component(rate, 2, 1, 100) if i % 4 == 0
   else
450.              Component(rate, 8, 1, 100) if i % 4 == 1
   else
451.              Component(rate, 12, 2, 100) if i % 4 == 2
    else
452.              Component(rate, 8, 1, 100)
453.              for i, rate in enumerate(component_defect
   _rates)
454.          ]
455.          semi_products = [SemiProduct(min_defect_rate,
    8, 4, 6) for _ in range(3)]
456.          final_product = FinalProduct(min_defect_rate,
    8, 6, 10, 200, 40)
457.
458.          max_profit, best_decision = run_simulation(co
   mponents, semi_products, final_product, num_simulations,
    i)
459.          overall_max_profits.append(max_profit)
460.          all_decisions.append(best_decision)
461.
462.          print(f"重复 {i + 1}/100 完成，最大利
   润: {max_profit:.2f}")
463.
464.      decision_counter = Counter(all_decisions)
465.      most_common_decision, frequency = decision_counte
   r.most_common(1)[0]
466.      probability = frequency / 100
```

```
467.
468.      print(f"\n 最常见的决策策略出现 {frequency} 次，概率为 {probability:.2%}")
469.      print("该策略为:")
470.      print(f"零部件检查决策: {most_common_decision[0]}")
471.      print(f"半成品检查决策: {most_common_decision[1]}")
472.      print(f"半成品拆解决策: {most_common_decision[2]}")
473.      print(f"半成品拆解后检查决策: {most_common_decision[3]}")
474.      print(f"成品检查决策: {most_common_decision[4]}")
475.      print(f"成品拆解决策: {most_common_decision[5]}")
476.      print(f"成品拆解后检查决策: {most_common_decision[6]}")
477.
478.      visualize_max_profits(overall_max_profits)
479.
480.  if __name__ == "__main__":
481.
```