生产过程中的决策问题

摘要

工厂生产过程成品过程中常涉及到零件和成品合格率检查、次品返厂调换等问题。为了节约成本、使企业利润最大化，需要我们对零件验收过程抽样数、成品生产过程中的检查过程、次品返厂调换过程进行规划。

针对问题一，解决**最小抽样数**问题，为降低抽测成本，要使抽测数量尽可能小。据此，假定一个容许的误差率ε，根据**样本量计算公式**可以解出最少样本量。由公式发现ε与 n 呈现**负相关**的规律，即企业需要在较高的抽测成本和较大的误差间进行抉择，选择出适宜的误差（适度的抽测量)。我们在问题四中利用模拟，提出了一个合理的误差选择方法，来更完善地回答误差的选择。

针对问题二，解决从零件初始获得到次品无限次返厂这一全流程中，如何设计零件和成品检验以及返厂次品拆解决策，能够实现利润最大化问题。我们建立了优化模型，设定产品每次循环从零件购买开始到次品返厂结束，每次循环用上一轮返厂次品中的拆解数即为下一轮初始零件检验数作为连接，以零件 1 和 2 的检验决策、成品的检验决策、出售次品的拆解决策作为决策变量，构造**利润最大化模型**，并利用**遗传算法**求解。此外我们还设计了**仿真程序模拟**生产流程，进行了结果的佐证和比对。

针对问题三，解决复杂的多个零配件、多道工序的生产决策问题。依然按照类似问题二做法采取**利润最大化模型**建立和求解与**仿真模拟**交叉验证的方式考虑决策组合，建立复杂的利润最大化模型。求解结果是：所有零件中仅对零件一，零件二进行检测；半成品和成品中，对半成品二半成品三进行检测；拆解选择上，仅对半成品一进行拆解，这样得到的目标函数(单件期望收益)是 81.83 元。而基于**计算机模拟仿真**结果，应对所有零件做检测、不对半成品和最终成品做检测、仅对半成品一和最终成品做拆解，模拟的单件期望收益是 73.09 元。建模与模拟的底层逻辑不尽相同，于是我们给出如上两套有价值的决策方案。

针对问题四，解决总体次品率未知的生产规划问题。在零件总体的次品率未知的情形下，利用已知的样本次品率，获得总体次品率的分布。充分利用已有的**仿真模型**，在此基础上引入误差，为次品率增加一套**随机机制**。检验原先问题二、三各个情形中表现良好的决策组合，基于**蒙特卡洛模拟**结果来考察它们的性能变化。此外，我们也将记录总体性能在次品率变异性下的变化，以此定义一个创新性的样本量选择方案。我们发现问题二中的各个情形下，引入随机机制没有破坏优势决策组合的选择；但是在问题三中，由于更加复杂的生产条件，微小的误差产生的次品率变异就会导致原先最佳的决策组合暴露出问题，因此我们会选择检测所有的零件，并且至少检测两个半成品(或成品)，同时对至少两个半成品(或成品)的次品进行拆解，以此来应对次品率的**波动性**。

关键词：样本量公式、大数定律、仿真模拟、递归方程、遗传算法、蒙特卡洛模拟

一、问题提出

在企业产品生产过程中，检验零件的合格程度、产品合格程度和调换返厂不合格品等事务非常普遍，除了要完成上述事务，企业还要考虑生产过程中耗费的成本，由此，在生产进行之前所进行的统筹规划就显得格外重要。企业需要在零件、成品、返厂次品各环节的针对性策略，运用数学工具建立各决策变量之间的关系公式与模型，以反映决策问题的实质，将复杂的决策问题简化。

认识到生产过程的决策分配对企业利润的重要性，解决以下问题：

问题一

假设有充足且总数未知的零件总体，从中抽出 n 件进行检测，得出抽测的次品率。再根据已知抽测次品率这个条件，推断是否有大于信度的概率认定真实的总体的次品率大于或小于标称值，进而选择拒收或接受。

问题二

已知零件在初始获得到成品售出的过程中，分别经历两个零件的检验、成品检验和次品拆解等步骤，每一步骤在总体中选择合适的执行比例，作为最佳决策，使得执行后企业在整个生产过程获得最大利润。

问题三

同问题二，复杂化为 8 个零件、3 个半成品、1 个成品，对每一零件、半成品、成品选择是否检验，半成品、成品选择是否拆解。规划求解每个决策适宜的决策比例，使得企业实现生产利润最大化。

问题四

在仅知道样本次品率、总体次品率不明情况下，对问题二和问题三进行再次规划，同样要求找到各个步骤合适的决策，使得企业生产利润最大化。

二、问题分析

2.1 问题一的分析

在问题一中，要求企业对供货商提供的零件进行验收，满足条件即可入库。经分析，题干要求等价于一个**概率不等式**，即要求解一个概率大于等于信度的方程。计划利用**大数定律和 Slutsky 定理**，推导出抽测次品率近似服从的分布，进而得到了计算抽测次品率与总体次品率间误差的概率的方法，并反解得出一个关于 n 的不等式。

2.2 问题二的分析

在问题二中，“零件一是否检测”、“零件二是否检测”、“成品是否检测”、“不合格成品是否拆解”是四个待决策的变量。计划将其表示为四个 0-1 型变量，根据题意的约束条件构造了一个**非线性规划模型**，求解使利润最大化的决策及其利润值。同时额外单独使用**蒙特卡洛模拟**的方法，按照给定的次品率，从**伯努利分布随机**生成零件是否是次品或装配是否出错的事件，重复**模拟一万次**取均值，得出不同策略在模拟下的单位利润，既贴切真实场景，又能与模型构造的非线性规划相互印证结果。

2.3 问题三的分析

在问题三中，多道工序使得流程变得异常复杂，每一道工序都会涉及到“是

否检验”、“是否拆解”的问题，都是问题二的复现。与问题二类似，构建了一个更为复杂的非线性规划模型，该模型唯物每一道工序都构设置了一层循环，用来解决这道工序两端产品的拆解与重新装配等循环往复的问题。此外，我们也额外单独进行了蒙特卡洛模拟，按照给定的概率生成大量零件样本来模拟，以此对所有不同的策略空间进行搜索，给出了另一种自洽的最佳决策。

2.4 问题四的分析

问题四是问题二、三的延伸。在问题二、三中，零件总体的次品率已知，由此建立了利润最大化模型。但在本问中，零件总体的次品率未知，已知的次品率是样本次品率。为了解决问题，我们以一种情形为例，充分利用之前模型，模拟得到多组总体次品率后代入问题二、三模型。我们将检验原先问题二、三各个情形中表现良好(前三名)的决策组合，模拟它们在变异的真实次品率下的性能变化，以此来引入一个有参考价值的决策维度。此外，我们也将记录前三名的总体性能在次品率变异性的变化，以此来给出一个更加具体的样本量选择方案。即我们会利用误差ε扰动进行灵敏度分析，由图像解释对误差ε进行筛选。

三、模型假设

1.假设不考虑产品循环返厂过程中对零件和成品质量的影响。

2.假设企业检测后为良品的成品不存在被用户返回、所有次品均会被返回。

3.假设企业拆解过程中，只会拆解一道工序。

4.假设企业在制作过程中，对于次品，为了让 1、2 两种零件相配，会再次购买新的零件。

5.假设企业每一次对相同零件、半成品、成品的检测和拆解比例不变。

四、符号说明

表 1 本文的符号说明

符号	说明	单位
n	零件抽样个数	个
μ_n	样本中次品个数	个
ε	允许误差	%
p	标称值	%
q	单价	元
C	成本	元
R	收入	元
Z	利润	元
$Score$	均值-方差得分	1
S	成本销售单价	元
$Price_i$	零件i的购买成本	元
p_i	零件i/半成品或成品总体次品率	%
p_i'	零件i的样本次品率	%
E_i	第i大利润向量期望	元
Var_i	第i大利润向量方差	元

五、 模型建立与说明

5.1 问题一：大数定律解决概率问题

在问题一中，要求企业对供货商提供的零件进行验收。采用抽样调查方法，写出题干要求的**概率公式大于等于信度的方程**。利用**大数定律**改写概率公式为符合**标准正态分布**形式，代入信度、设定误差、标称值，即可求出满足信度的最小样本数。

5.1.1 概率公式联合信度方程构建

以第一小问为例，题干要求找到一种抽样方案，抽取尽可能少的零件就能在信度为95%时认定总体次品率超过标称值，则拒收这批零配件。这里没有给出零件总体的数量量，若选用传统的假设检验方式则需要自行定义，且也无法体现出抽测数量“尽可能少”。故从另一角度思考：给定误差，令总体次品率围绕在抽测次品率（通过抽测而已知的数）周围的一个小区间的概率大于信度，那么只要该区间不包含标称值，即可判定总体次品率以大于信度的概率大于或小于等于标称值，进而选择拒收或接受，分别对应问题一的两种情形。以下是详细的公式推导过程。

写出的等价于题意的概率公式：

$$P\left(\left|\frac{\mu_n}{n}-p\right|\leqslant\varepsilon\right)\geqslant\text{信度} \tag{1}$$

其中，ε为一设定值，是误差范围。

由大数定律：

$$\frac{\mu_n-np}{\sqrt{np(1-p)}}\xrightarrow{P}N(0,1) \tag{2}$$

利用 Slutsky 定理变形之后得到：

$$\frac{\mu_n}{n}\xrightarrow{P}N\left(p,\frac{p(1-p)}{n}\right) \tag{3}$$

则原始公式可化为标准正态分布：

$$\begin{aligned}&\therefore P\left(\left|\frac{\mu_n}{n}-p\right|\leq\varepsilon\right)\\&=P\left(-\varepsilon\frac{\sqrt{n}}{\sqrt{p(1-p)}}<\frac{\mu_n-np}{\sqrt{np(1-p)}}<\varepsilon\frac{\sqrt{n}}{\sqrt{p(1-p)}}\right)\\&\approx2\Phi\left(\varepsilon\frac{\sqrt{n}}{\sqrt{p(1-p)}}\right)-1\end{aligned} \tag{4}$$

将（4）式代入（1）式，即求解：

$$2\mathit{\Phi}\left(\varepsilon\frac{\sqrt{n}}{\sqrt{p(1-p)}}\right)-1\geqslant\text{信度} \tag{5}$$

分别代入信度为95%、90%，ε为 1%，p 为标称值 10%, 利用 python 代码得：

$$\text{信度}=95\%，n\approx 3457.31$$

$$\text{信度}=90\%，n\approx 2434.99$$

表 2 两种情形不同信度和允许误差下的最小样本量n

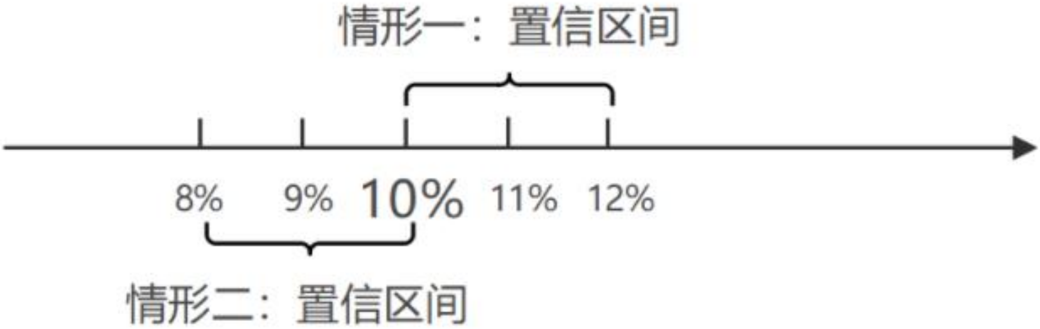
信度\误差	1%	2%	3%	4%	5%
95%	3457.31	864.34	384.15	216.08	138.29
90%	2434.99	608.75	270.55	152.19	97.40

5.1.2 抽样方案得出

将不同的假定ε和两个情形中的信度代入，计算得如上表 2 的最小样本量。进而指导企业进行验收工作：以情况一为例，若取误差为 1%，则至少抽取 3458 个样本，若取误差为 5%，则需抽 139 个样本。即每个误差与对应一个需要抽测的最小样本数，且呈现负相关的关系。

下面针对两种情形下的决策做进一步解释。情形一中，不妨假设误差取 1%，则根据之前的等价概率公式可知“总体次品率 p 在抽测次品率（已知）周围 1%范围的区间内的概率大于等于 95%”，则若抽测出的次品率为 11%，可以画出图一中置信区间。一旦该区间完全落在标称值 10%的右侧，则由“总体次品率落在该区间概率大于等于 95%”可以推得“总体次品率大于标称值的概率大于等于 95%”，即应拒收该批产品。而由于该区间的中心点为抽测次品率，则“区间完全落在标称值右侧”等价于抽测次品率大于 11%。因此，在情形一中，只要抽测的次品率大于 11%，则能以 95%信度认定零件总体的次品率超过标称值并拒收。

图 1 ε为 1%时不同次品率的企业对零件措施



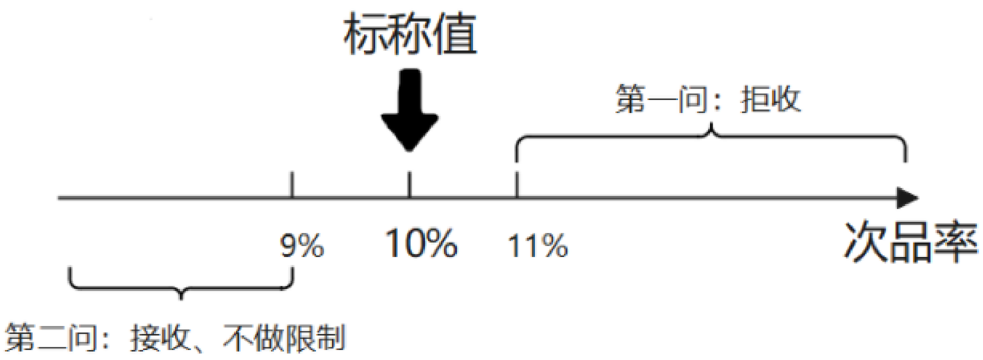
下面分析该如何选定误差。上一段的分析在误差假定为 1%的情况下，需要抽测 3458 个。而若选择误差为 5%，那么只需要抽测 139 个即可，但是，上述推断出的“抽测次品率大于 11%时拒收”应修改为 “抽测次品率大于 15%时才拒收”，显然这需要企业承担更大的风险，尽管所需的抽测成本较小。因此企业需要在风险和较少的抽测成本中进行抉择来选定一个合理的误差范围（即选定合理的抽测量）。**在第四问中，我们回顾了问题一并利用模拟在不同情形下给出了一个我们推荐的误差选择范围，这会在解答问题四时介绍。**

情形二与情形一类似。要使以抽测次品率（已知）为中心的置信区间完全

落在标称值左侧才接受，则当误差取 1%时，需要抽测的次品率小于等于 9%才接受，置信区间如图一所示。同样的，这也与误差的选择高度相关，若选误差为 5%，尽管只需要抽测 98 次就能得出概率大于等于 90%的结论，但需要在抽测次品率小于 5%时才会接受，这无疑会让企业错过好多批次品率较低的配件，即错过这些商机。所以企业仍需要选择一个合理的误差。

以下是选定误差 1%时，两种情形下企业根据抽测次品率做决策的示意。

图 2 ε为 1%时不同情形下的置信区间



5.2 **问题二：**企业生产过程的各个阶段决策

第二问主要解决安排企业零件检测、成品检测和次品拆解过程中的决策问题，要求利润最大化。针对问题，分别拆解产品全流程收入和成本，再进行汇总，求解优化问题。

在本问中，符号C_1表示购买零件的成本，符号$C_2^{(k)}$，$C_3^{(k)}$,$C_4^{(k)}C_5^{(k)}$,$C_6^{(k)}$分别表示第k轮下的零件检测成本、装配成本、成品检验成本、调换成本、拆解成本。符号q_1,q_2,q_3,q_4,q_5,q_6分别表示零件一、零件二的检测单价、装配单价、成品检验单价、调换单价、成品拆解单价。符号p_1,p_2,p_3分别表示零件一、零件二的次品率和成品装配过程中的次品率。符号$Price_1,Price_2$分别表示零件一、零件二的购买单价。符号$N_1^{(k)}$表示第 k 轮目标要搭建出的成品的数量。4 个 0-1 型决策变量为d_1,d_2,d_3,d_4，分别代表零件一、零件二是否检测，、成品是否检验和是否会拆解成品。

5.2.1 企业生产过程决策模型建立

图 3 生产过程示意

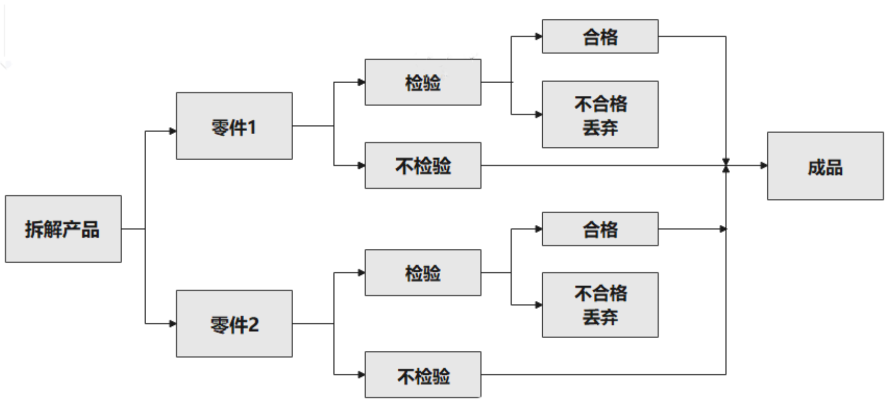
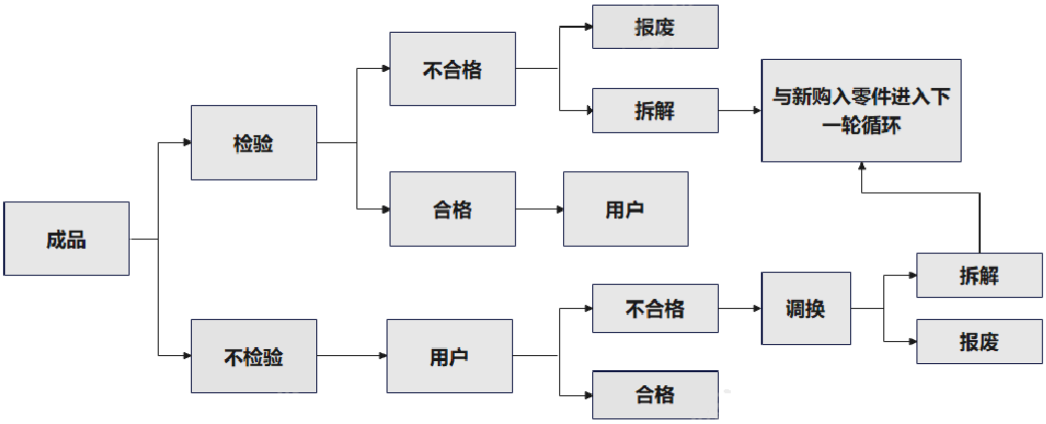


图 4 生产过程示意



有必要解释的是，出售的产品返厂后，如果考虑拆解，又涉及到新一轮的零件检测循环，所以我们一次循环从获得零件开始到次品返厂拆解结束。以下解释，除C_1外都以中途第 k 轮的零件获得开始，符号上角标的（k）代表第 k 轮，适用于任意一轮。如下为产品每轮接受的检测流程，为零件检测流程和成品装配流程、成品检测流程和次品返厂及拆解流程。

■ 成本拆解

成本拆解过程中，我们先考虑一轮循环中的总成本，分别为零件购买成本、零件检测成本、成品装配成本、成品检测成本、次品调换成本以及次品拆解成本，再将成本按照每轮累加。

1）零件购买成本

零件购买为整个企业生产循环的开始，我们从第一次购买两种零件开始进行分析。

第一次购买零件，经过检测后会出现剩下的零件数目与另一零件并不匹配的情况，按照假设，企业会再次购入新的零件以达到匹配目的。这里需要解释一点，模型建立要尽量**贴合实际**，那么为了匹配零件个数而再次购入的零件，在经过检验后还是有部分丢弃，那么需要再次补充数目，由此产生购买零件循环。因此在购买零件开始，已经进入了无休止的循环，即**零件购买成本除第一次购买的数模，还应加上为了达到两种零件数目匹配而不断购买新零件的成本。**设对零件i的检测比例为d_i, 购买N_i对零件，则每次检测有$N_i \cdot d_i \cdot p_i$个次品零件被检测出来被丢弃，为了将良品零件个数补充至N_i并两两零件配对，需要再次购买$N_i \cdot d_i \cdot p_i$个零件。累加之后得到从开始购买N_i对零件开始，一共需要进行购买的零件个数为：

$$N_i(1+d_ip_i+d_i^2p_i^2+\ldots+d_i^np_i^n) \\ =N_i\frac{1}{1-d_ip_i} \tag{6}$$

用每种零件单价乘以购买数目，并进行化简得到如下零件购买公式：

$$C_1=\sum_{i=1}^{2}N_1^{(1)}\cdot\frac{1}{1-d_ip_i}\cdot\mathrm{Price}_i \tag{7}$$

2) 零件检测成本

与零件购买成本相同，零件检测成本也有相同的**费用迭代**现象，即进行过第

一轮检测后还需要进行后续无数轮的补充零件的检测。如（6）式，经由第一次购买零件后，每轮需要进行的零件检测数是初始数目加上补充数目再乘以检测比例d_i。易得，每轮零件 1、2 总共的检测费用为：

$$C_2^{(k)} = \sum_{i=1}^{2} N_1^{(k)} \cdot d_i \cdot \frac{1}{1-d_i p_i} \cdot q_i \tag{8}$$

3）成品装配成本

在考虑装配时，要明确装配时两种零件缺一不可，遵从短板效应，只能按照两个合格零件数目中最少的数目去配对，那么**装配成功的成品数要减去两种零件中的最大不合格数目**。则成品个数为：

$$N_2^{(k)} = N_1^{(k)}[1-\max(d_2 p_2, d_2 p_2)] \tag{9}$$

由此，成品装配成本成品数乘以成品装配单价：

$$C_3^{(k)} = N_2 \cdot q_3 \tag{10}$$

4）成品检测成本

设对成品的检测比例为d_3，容易得出检测成本即为成品数乘以检测比例再乘以成品检测成本：

$$C_4^{(k)} = N_2 \cdot d_3 \cdot q_4 \tag{11}$$

5）次品调换成本$C_5^{(k)}$

解决次品调换问题要明晰两点，一是实际调换成本，而是调换个数。由于调换时既要考虑调换成本，还要考虑到企业需要**无条件予以调换的新货品成本**，按照市价计算重新发出的货品，那么实际调换成本应该为(q_5+S)。而调换个数方面，首先调换总体都是在成品检测阶段未接收检测的部分，即$N_2 \cdot (1-d_3)$，并且必然有问题的成品。而必然有问题的成品有三种成因，成因一和成因二分别是零件 1、2 有问题、没有经历零件检测阶段且非成品装配阶段问题，成因三则为无论零件检测阶段有无问题，成品装配阶段问题。在这里无论零件检测阶段有无检查、零件本身是否良品都算入成因三，**是为了考虑最坏情况，让装配出的次品最多。**故这部分比例为$\{[(1-d_1)p_1+(1-d_2)p_2](1-p_3)+p_3\}$。

故调换成本为：

$$C_5^{(k)} = \mathrm{N}_2^{(k)}(1-\mathrm{d}_3)\{[(1-\mathrm{d}_1)\mathrm{p}_1+(1-\mathrm{d}_2)\mathrm{p}_2](1-\mathrm{p}_3)+\mathrm{p}_3\}(\mathrm{p}_5+\mathrm{S}) \tag{12}$$

6）次品拆解成本$C_6^{(k)}$

考虑次品拆解率为d_4，面临拆解的成品其总数为成品检测阶段检测出的次品加上未经历成品检测阶段的次品。这部分次品的个数计算与$C_5^{(k)}$较为相似，**考虑最坏情况**让装配出的次品最多。组合拆解个数、拆解率、拆解成本，得到次品拆解成本为：

$$C_6^{(k)} = N_2^{(k)}\{[(1-d_1)p_1+(1-d_2)p_2](1-p_3)+p_3\}d_4 q_6 \tag{13}$$

■ 收入分析

不同于成本拆解，每轮产品的收入构成较为简单，仅有售出成品收入。考虑收入时，按照成品销售个数乘以成品销售单价即可求得。其中求得成品销售个数须详细介绍。

利用朴素的理解，销售出去的成品个数即为所有装配成功的成品个数减去在成品检测阶段筛除的次品个数。**被成品检测筛去的成品是未经历零件检测但零件存在问题的成品与装配过程出现问题的并集**，为符合实际，这里不同于上面的采用最坏情况做法，我们使用**期望思想**，认为两种情况出现概率相同，从而得到被筛去的成品个数的表达式：

$$\frac{1}{2}N_2\{[(1-d_1)p_1+(1-d_2)p_2]+Max[(1-d_1)p_1,(1-d_2)p_2]\} \tag{14}$$

由此，每轮的收入方程为：

$$R^{(k)}=S\left\{N_2^{(k)}-\frac{1}{2}N_2^{(k)}\{[(1-d_1)p_1+(1-d_2)p_2]+Max[(1-d_1)p_1+(1-d_2)p_2]\}\right\} \tag{15}$$

■ 递归公式

拆解了每轮循环中所有收入和成本，下面来介绍循环之间的迭代过程。

我们知道，客户在收到次品后会将成品返厂，商家除了调换商品还面临一个决策，即是否对次品进行拆解问题。不仅返厂商品面临拆解，成品检测结束后也面临着这样的决策，且由题意成品拆解后将返回一、二流程，即**上一次循环结束后所有拆解成品所得的零件将进入下一循环的零件检测过程**，这就是递归来源。

$$N_2^{(k)}=N_2^{(k-1)}\cdot d_4\cdot\{[(1-d_1)p_1+(1-d_2)p_2](1-p_3)+p_3\} \tag{16}$$

■ 利润最大化模型建立

以上阐述，选取某一循环过程具体分析了影响利润的收入和成本，还得到了连接每一循环的递归公式。

综上，我们可以得到利润最大化模型：

$$\mathrm{Max}\,Z=\sum_{k=1}^{\infty}R^{(k)}-\left[C_1+\sum_{k=1}^{\infty}\sum_{j=2}^{6}C_j^{(k)}\right] \tag{17}$$

求解合适的d_1、d_2、d_3、d_4，使得以上函数实现最大化。

5.2.2 利用遗传算法实现最大化模型求解

遗传算法是一种通过模拟自然进化过程搜索最优解的方法，通过大量备选解的变换、迭代和变异，在解空间中并行动态地进行全局搜索。

表 3 问题二算法求解结果

情况	d_1	d_2	d_3	d_4	$MaxZ$(平均)
一	1	1	0	1	17.49
二	1	1	0	1	10.44

三	1	1	1	1	14.83
四	1	1	1	1	14.13
五	1	1	01	1	9.46
六	1	0	0	0	19.25

5.2.3 算法与模拟结果对比验证及差异分析

以上利用遗传算法对问题进行优化求解，得到了四种情形下对应的决策以及利润最大值。为了验证其准确性，我们还采用 python 代码，代入算法求解的决策d_1、d_2、d_3、d_4，传入题干的设定的概率，对每种情形进行生产全流程的模拟。用该模拟结果得到的总利润与分别上述 5.2.2 中遗传算法求解的利润最大值进行比较，作为交叉验证。

该模拟流程，与图 3、图 4 流程相同，即从购买零件为开始到遇到两个终止条件——正常卖出或丢弃结束，其中正常卖出会产生收益。要注意的是在四个决策点，即是否检测零件 1、2，是否检测成品，是否拆解成品中，在是否检测阶段，我们设定的模拟机制与模型搭建时相同，即会在购买阶段进行循环检测和购买，如果合格零件数量不匹配，那么就一直停留在购买阶段，以保证零件两两配对。

故而，应当注意，我们的模拟流程对决策组合有着一定的要求，即：如果决定对次品成品进行拆解，那么必然要求对两个零件都进行检测，否则将会出现次品的零件反复进入装配阶段的**死循环**局面。

考虑我们一次模拟的可能结果：第一，正常卖出，尽管过程未必一帆风顺，可能面临多次检测发现次品而重新购买零件增加成本的情况，但是最终我们会得到一个良品的成品，正常卖出，产生正的利润；第二，丢弃，由于选择不拆解次品成品，所以我们会选择直接丢弃，虽然减少了拆解的费用，但最终一定是产生纯粹的损失。

对于一套给定的决策参数，我们会进行充分多的模拟，求其均值，我们认为这个均值能够反映这套决策参数在当前情形下的性能(期望收益)。将其作为一个评价指标，控制情形相同，我们就能比较不同决策参数的优劣，并获得一个大致的收益期望。下图展示了在情形一下，不同决策参数能够获得的期望收益，直观地呈现我们如何通过数值模拟来印证我们的建模结果：

表 4 问题二模拟结果

情况	d_1	d_2	d_3	d_4	$MaxZ$(平均)
一	1	1	0	1	20.84
二	1	1	0	1	10.84
三	1	1	1	1	18.34
四	1	1	1	1	10.28
五	1	1	0	1	17.47
六	1	1	0	0	23.69

5.3 **问题三** 2 道工序、8 个零配件生产决策

问题三是问题二的深入，将装配过程复杂化，多出半成品装配流程。实际解决核心思路与问题二有较大相关性，但是在模型建立上比问题二更加深入和复杂，需要对公式进行大量阐述。问题三的分析从规划模型建立、算法介入求解模型最大值、蒙特卡洛模拟和评估三个步骤进行。

在本问中符号$C_1^{(m)},C_2^{(m)}$，$C_3^{(m)},C_4^{(m)}C_5^{(m)}$分别代表第一道工序中第 m 轮零件的购买成本、零件检测成本、半成品装配成本、半成品检验成本、半成品拆解成本。符号$C_6^{(k)}C_7^{(k)}C_8^{(k)}C_9^{(k)}$分别代表第二道工序中第 k 轮成品的装配成本、成品的检验成本、成品的拆解成本、成品的调换成本。符号q_i，$i=1,\ldots,8$表示八个零件的检测单价，符号q_i，$i=9,10,11$表示三个半成品的装配单价，符号q_i，$i=12,13,14$表示三个半成品的检验单价，符号q_i，$i=15,16,17$分别表示三个半成品的拆解单价，符号q_{18}、q_{19}、q_{20}、q_{21}分别表示成品的装配单价、成品的检验单价、成品的拆解单价和成品的调换单价。符号p_i，$i=1,\ldots,8$表示八个零件的次品率，符号$p_i^{(1)}$，$i=12,13,14$表示三个半成品的装配次品率，符号p_{15}表示成品的装配次品率。符号$Price_i,i=1,\ldots,8$表示阿哥零件的购买单价。

5.3.1 复杂的利润最大化模型建立

还是以一个循环为例，从购买八种零配件开始，到次品返厂拆解结束，详细分析题意。具体的模型分析步骤从四个方面进行，分别是全局公式分析、成本拆解、递归阐述、综合模型建立，由此逐步得到复杂的利润最大模型建立。

- **全局公式分析**

首先假设我们计划生产$N^{(1)}$件半成品，则显然初始应当 8 种零配件各准备$N_i^{(1)}=N^{(1)}$个。公式如下：

$$N^{(1)}=N_i^{(1)}=指定起始数值 \tag{18}$$

注意，$N^{(m)}=N_i^{(m)}$表示零件与半成品交互过程第 m 论开始时的所有种类的零件数量。

此外，在定义的变量中，$p_{12}^{(1)},p_{13}^{(1)},p_{14}^{(1)},p_{15}^{(1)}$分别表示三个半成品和成品在装配过程中的次品率，由于一个半成品是次品的原因不止于装配过程，还有可能是未检测出的次品零件的被装配了。所以半成品一自身的次品率（不是装配过程中的次品率）并不是$p_{12}^{(1)}$,而是由如下公式计算所得：

$$p_{12}^{(2)}=\left(p_{12}^{(1)}+\left(1-p_{12}^{(1)}\right)\sum_{i=1}^{3}(1-d_i)p_i\right) \tag{19}$$

其中$p_{12}^{(1)}$代表装配过程导致次品的概率，在装配过程没有问题的另外$\left(1-p_{12}^{(1)}\right)$比例中，可能因为零件存在检测的次品导致损坏。每个零件剩下次品进入装配过程的比例为$(1-d_i)p_i$，即未检测的比例乘以次品率。

我们采纳了考虑最坏情况的思想，因为这里只是把三个零件的进入次品的比率简单相加，而事实上应该对他们取并集来考虑到搭配的三个零件中有不止一个是坏的情况，这样的次品率较前面的公式而言会减少。一方面出于便于计算，另一方面是为了考虑最坏情况，我们做出如上公式。

同理，另外两个半成品和最终成品公式也是如此：

$$p_{13}^{(2)} = \left(p_{13}^{(1)} + \left(1 - p_{13}^{(1)}\right) \sum_{i=4}^{6} \left(1 - d_i\right) p_i \right) \tag{20}$$

$$p_{14}^{(2)} = \left(p_{14}^{(1)} + \left(1 - p_{14}^{(1)}\right) \sum_{i=7}^{8} \left(1 - d_i\right) p_i \right) \tag{21}$$

$$p_{15}^{(2)} = \left(p_{15}^{(1)} + \left(1 - p_{15}^{(1)}\right) \sum_{i=12}^{14} \left(1 - d_i\right) p_i^2 \right) \tag{22}$$

■ **成本拆解**（第一道工序）

以下按照生产过程中**从前往后的顺序**来拆解**第一道工序**中可能产生的成本。

1）零件初始购买成本

为准备每种零件$N_i^{(1)}$个，我们需要按照它的市价购买，则会产生$\sum_{i=1}^{8} N_i^{(1)} \cdot Price_i$的成本。但如同问题二一样，我们认为检测出零件是次品后，还需要重新购买零件以实现搭配出$N^{(1)}$件半成品的目标。检测率为d_i, 次品率为p_i, 所以会有$d_i \cdot p_i$比例的零件被检测出次品并丢弃，这些被丢弃的数量需要被购买回来，所以目前的总购买量为$\sum_{i=1}^{8} N_i^{(1)} \cdot (1 + d_i \cdot p_i) Price_i$。然而购买的第二波中仍会检测出需要丢弃的次品，于是在第二波$d_i \cdot p_i$的比例中又会丢弃$d_i \cdot p_i$比例的零件，即占初始总体的$(d_i \cdot p_i)^2$。循环往复，最终购买零件的成本会是$C_1 = \sum_{i=1}^{8} N_i^{(1)} \cdot Price_i \cdot (1 + d_i p_i + (d_i p_i)^2 + \cdots)$。将无穷级数化简，得到如下的零件购买成本。

$$C_1^{(m)} = \sum_{i=1}^{8} N^{(m)} \cdot \mathrm{Price}_i \cdot \frac{1}{1 - d_i p_i} \tag{23}$$

2）零件检测成本

N、d、q 分别表示零件的综述，检测的比率和检测的单价，显然他们相乘再带上表示零件的角标 i 就表示了一个零件的检测成本。但与购买成本一样，第一次检测完丢掉后会要第二次购买，这些购买的也需要检测，同样继续丢继续买一直循环下去也会出现无穷级数的形式，所以每个零件的检测成本还需要乘上$(1 + d_i p_i + (d_i p_i)^2 + \cdots)$。对各零件的检测成本求和并化简后得到如下零件检测总成本：

$$C_2^{(\mathrm{m})} = \sum_{i=1}^{8} N^{(m)} \cdot q_i d_i \cdot \frac{1}{1 - d_i p_i} \tag{24}$$

3）半成品装配成本

尽管零件会检测出次品都丢掉，但又会购入新的零件填充，不断重复下去，就会保证最终剩下能够进入装配的每种零件还是有$N_i^{(m)}$个，且q_9, q_{10}, q_{11}分别代表三个半成品的装配单价，故半成品装配成本如下：

$$C_3^{(\mathrm{m})} = q_9 \cdot N_1^{(\mathrm{m})} + q_{10} \cdot N_4^{(\mathrm{m})} + q_{11} \cdot N_7^{(\mathrm{m})} \tag{25}$$

4）半成品检验成本

类似，q_{12}, d_{12}分别代表半成品检验的单价和检验的比例，则半成品检验成本如下：

$$C_4^{(m)} = q_{12} d_{12} N_1^{(m)} + q_{13} d_{13} N_4^{(m)} + q_{14} d_{14} N_7^{(m)} \tag{26}$$

5) 半成品拆解成本

拆解的半成品一定是要被检验成次品的，即一定在检验处的比例d_i中，$p_i^{(2)}$表示半成品自身的次品率，其具体形式已在全局公式分析中指明。这两者相乘得出半成品检测出的次品站所有半成品的比例，再乘以d_{i-3}表示的拆解百分比，即得出总共需要拆解的占比，再乘以拆解的单价q_{i+3}和总数$N^{(m)}$，即得某一半成品的拆解费用。则半成品拆解总费用如下：

$$C_5^{(m)} = \sum_{i=12}^{14} N^{(m)} \cdot d_i \cdot P_i^{(2)} \cdot d_{i-3} \cdot q_{i+3} \tag{27}$$

■ **递归阐述**（第一道工序）

对于零件到半成品的这第一道工序而言，假设一开始每种零件都有$N^{(1)}$个，尽管检测零件这一步需要丢掉零件，但也会通过不断买入新的零件，保证最终走入装配阶段的零件数仍是 $N^{(1)}$个，即三种半成品都会搭配出$N^{(1)}$个。然而半成品也会检验，检验出的次品按一定比例d_9, d_{10}, d_{11}拆解, 拆解出的零件则又会进入重新组装成半成品的环节，即重新走一遍上一步的流程。唯一的区别在于，一开始的目标是要每种半成品都装出$N^{(1)}$个，现在加入三类半成品拆下来的零件数量不同，从最终产生的半成品的数量要能配对的角度出发，我们把计划生产的目标半成品数量定位$N^{(m)} \cdot Min\left(d_{12}p_{12}^{(2)}, d_{13}p_{13}^{(2)}, d_{14}p_{14}^{(2)}\right)$。所以目标半成品数量 N 的迭代公式如下：

$$N^{(m+1)} = N^{(m)} \cdot Min\left(d_{12}p_{12}^{(2)}d_9, d_{13}p_{13}^{(2)}d_{10}, d_{14}p_{14}^{(2)}d_{11}\right) \tag{28}$$

此外，记$M^{(1)}$为进入下一道工序时已有半成品的对数。由于初始计划装配$N^{(1)}$对半成品，尽管在零件层面和装配层面都会因为出现损失，但这很快被下一轮的装配补足了（迭代无穷轮的目的正是确保最终能出来$N^{(1)}$对半成品）。所以我们有：

$$M^{(1)} = N^{(1)} \tag{29}$$

■ **成本拆解**（第二道工序）

6）成品装配成本

公式显然如下，为半成品对数乘以成品装配成本：

$$C_6^{(k)} = M^{(k)} q_{18} \tag{30}$$

7）成品检验成本：

公式显然如下，为半成品对数乘以检验率和检验单价：

$$C_7^{(k)} = M^{(k)} \cdot d_{15} \cdot q_{19} \tag{31}$$

8）成品拆解成本

公式显然如下，为装配对数乘以成品次品率乘以成品拆解率乘以成品拆解单价：

$$C_8^{(k)} = M^{(k)} \cdot d_{16} \cdot q_{20} \cdot p_{15}^{(2)} \tag{32}$$

9）成品调换费用

半成品对数乘以不检测比例，这是因为只有不检测的部分才会被发往客户。再乘以成品的次品率确认发给客户的次品成品数量，由于我们假设每一个发给客户的次品都会被调换，因此再乘以单位调换成本即可。公示如下：

$$C_9^{(k)} = M^{(k)} \cdot (1 - d_{15}) \cdot q_{21} \cdot p_{15}^{(2)} \tag{33}$$

■ **递归阐述**（第二道工序）

对于第二道工序装出来的成品，一旦拆解成半成品后仍旧会再走一遍之前的步骤并装出来新的成品。所以每一轮执行完后，更新$M^{(k)}$以进行下一轮。具体更新公式如下：

$$M^{(k+1)} = M^{(k)} p_{15}^{(2)} d_{16} \tag{34}$$

这是显然的，因为这三项的乘积结果是再成品拆解成本中提到的拆解的数量，之前已在成品拆解成本的公式中推导过。

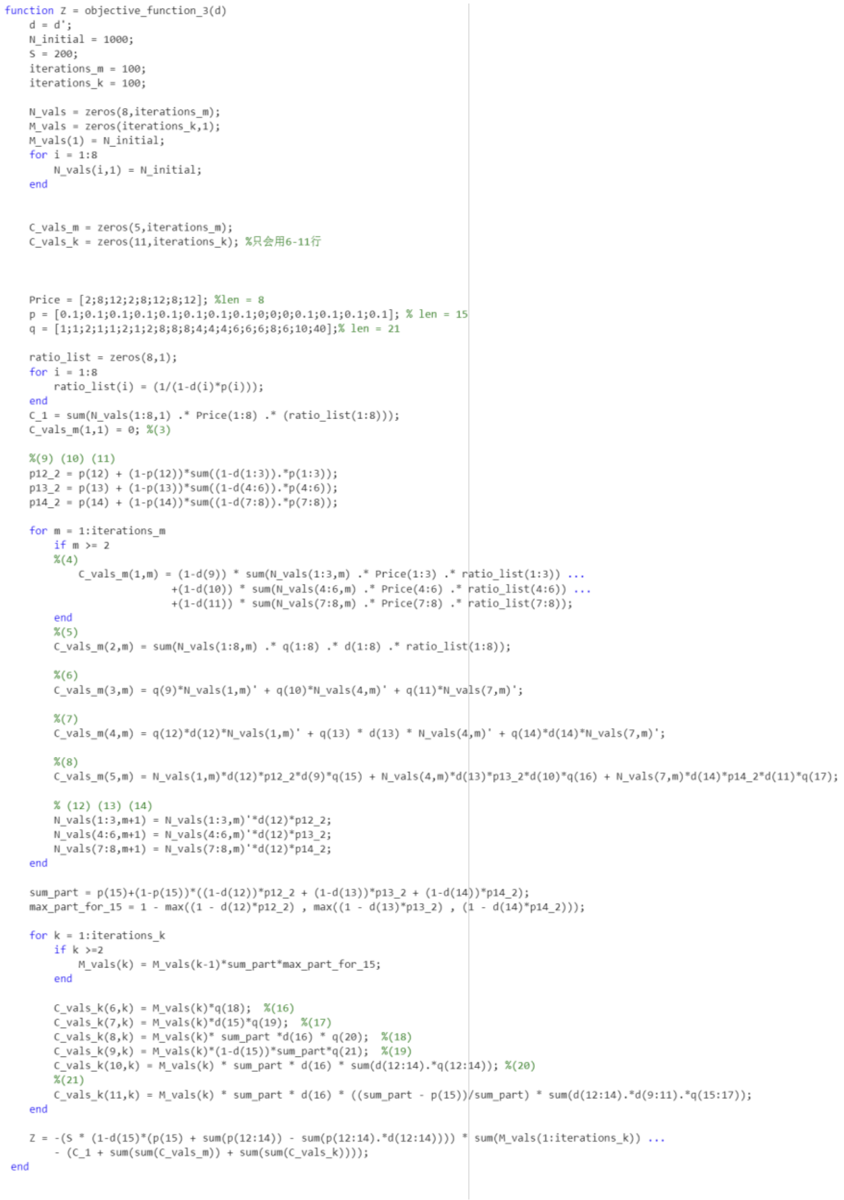
■ **复杂的利润最大化模型建立**

$$\max Z = S \cdot \sum_{k=1}^{\infty} M^{(k)} \left(1 - d_{15} \cdot p_{15}^{(2)}\right) - \left(\sum_{m=1}^{\infty} \sum_{j=1}^{5} C_j^{(m)} + \sum_{k=1}^{\infty} \sum_{j=6}^{9} C_j^{(k)} \right) \tag{35}$$

S 代表每个成品的售价，$M^{(k)} \cdot \left[1 - d_{15} p_{15}^{(2)}\right]$代表每一轮的产生的成品中没有被检验或检验了但是是良品的数量，因此减号前的部分表示了总收入。而减号后的部分则是各轮下各类型费用的总和，是总成本。两者相减即可得到需要最大化的总利润。

5.3.2 算法解决复杂的利润最大化模型

利用 MATLAB 代码，使用遗传算法，建立目标函数。



```
function Z = objective_function_3(d)
    d = d';
    N_initial = 1000;
    S = 200;
    iterations_m = 100;
    iterations_k = 100;

    N_vals = zeros(8,iterations_m);
    M_vals = zeros(iterations_k,1);
    M_vals(1) = N_initial;
    for i = 1:8
        N_vals(i,1) = N_initial;
    end


    C_vals_m = zeros(5,iterations_m);
    C_vals_k = zeros(11,iterations_k); %只会用6-11行



    Price = [2;8;12;2;8;12;8;12]; %len = 8
    p = [0.1;0.1;0.1;0.1;0.1;0.1;0.1;0.1;0;0;0;0.1;0.1;0.1;0.1]; % len = 15
    q = [1;1;2;1;1;2;1;2;8;8;8;4;4;4;6;6;6;8;6;10;40];% len = 21

    ratio_list = zeros(8,1);
    for i = 1:8
        ratio_list(i) = (1/(1-d(i)*p(i)));
    end
    C_1 = sum(N_vals(1:8,1) .* Price(1:8) .* (ratio_list(1:8)));
    C_vals_m(1,1) = 0; %(3)

    %(9) (10) (11)
    p12_2 = p(12) + (1-p(12))*sum((1-d(1:3)).*p(1:3));
    p13_2 = p(13) + (1-p(13))*sum((1-d(4:6)).*p(4:6));
    p14_2 = p(14) + (1-p(14))*sum((1-d(7:8)).*p(7:8));

    for m = 1:iterations_m
        if m >= 2
        %(4)
            C_vals_m(1,m) = (1-d(9)) * sum(N_vals(1:3,m) .* Price(1:3) .* ratio_list(1:3)) ...
                           +(1-d(10)) * sum(N_vals(4:6,m) .* Price(4:6) .* ratio_list(4:6)) ...
                           +(1-d(11)) * sum(N_vals(7:8,m) .* Price(7:8) .* ratio_list(7:8));
        end
        %(5)
        C_vals_m(2,m) = sum(N_vals(1:8,m) .* q(1:8) .* d(1:8) .* ratio_list(1:8));

        %(6)
        C_vals_m(3,m) = q(9)*N_vals(1,m)' + q(10)*N_vals(4,m)' + q(11)*N_vals(7,m)';

        %(7)
        C_vals_m(4,m) = q(12)*d(12)*N_vals(1,m)' + q(13) * d(13) * N_vals(4,m)' + q(14)*d(14)*N_vals(7,m)';

        %(8)
        C_vals_m(5,m) = N_vals(1,m)*d(12)*p12_2*d(9)*q(15) + N_vals(4,m)*d(13)*p13_2*d(10)*q(16) + N_vals(7,m)*d(14)*p14_2*d(11)*q(17);

        % (12) (13) (14)
        N_vals(1:3,m+1) = N_vals(1:3,m)'*d(12)*p12_2;
        N_vals(4:6,m+1) = N_vals(4:6,m)'*d(12)*p13_2;
        N_vals(7:8,m+1) = N_vals(7:8,m)'*d(12)*p14_2;
    end

    sum_part = p(15)+(1-p(15))*((1-d(12))*p12_2 + (1-d(13))*p13_2 + (1-d(14))*p14_2);
    max_part_for_15 = 1 - max((1 - d(12)*p12_2) , max((1 - d(13)*p13_2) , (1 - d(14)*p14_2)));

    for k = 1:iterations_k
        if k >=2
            M_vals(k) = M_vals(k-1)*sum_part*max_part_for_15;
        end

        C_vals_k(6,k) = M_vals(k)*q(18);  %(16)
        C_vals_k(7,k) = M_vals(k)*d(15)*q(19);  %(17)
        C_vals_k(8,k) = M_vals(k)* sum_part *d(16) * q(20);  %(18)
        C_vals_k(9,k) = M_vals(k)*(1-d(15))*sum_part*q(21);  %(19)
        C_vals_k(10,k) = M_vals(k) * sum_part * d(16) * sum(d(12:14).*q(12:14)); %(20)
        %(21)
        C_vals_k(11,k) = M_vals(k) * sum_part * d(16) * ((sum_part - p(15))/sum_part) * sum(d(12:14).*d(9:11).*q(15:17));
    end

    Z = -(S * (1-d(15)*(p(15) + sum(p(12:14)) - sum(p(12:14).*d(12:14)))) * sum(M_vals(1:iterations_k)) ...
        - (C_1 + sum(sum(C_vals_m)) + sum(sum(C_vals_k))));
end
```

```
diary on;
% 设置变量的上下界，假设d1到d4的取值范围为[0,1]
lb = zeros(16,1); % 下界
ub = ones(16,1); % 上界
% 定义整数约束，表示d1, d2, d3, d4都是整数变量
IntCon = [1, 2, 3, 4,5,6,7,8,9,10,11,12,13,14,15,16];
% 使用遗传算法求解，并设置整数约束
options = optimoptions('ga', 'Display', 'iter', 'PopulationSize', 200, 'MaxGenerations', 300);
% 调用遗传算法，指定16个整数变量
[x, fval] = ga(@objective function 3, 16, [], [], [], [], lb, ub, [],IntCon,options);
```

5.3.3 模拟交叉验证

与问题二相同，除了利用算法对复杂的利润最大化模型进行求解，我们还利用 python 代码全流程模拟了算法求解的决策变量，再将之求得结果与算法求解结果进行对照。

下面我们将简单介绍一下模拟的流程。模拟的起始与问题二中类似，我们将根据给定次品率随机生成八个零件。说明将分成四个部分，即组装半成品一的部

分、组装半成品二的部分、组装半成品三的部分和组装最终成品的部分。

组装半成品一时涉及到五个决策：[是否检测零件一]、[是否检测零件二]、[是否检测零件三]、[是否检测半成品一]、[是否拆解半成品一]。

组装半成品二时涉及到五个决策：[是否检测零件四]、[是否检测零件五]、[是否检测零件六]、[是否检测半成品二]、[是否拆解半成品二]。

组装半成品三时涉及到四个决策：[是否检测零件七]、[是否检测零件八]、[是否检测半成品三]、[是否拆解半成品三]。

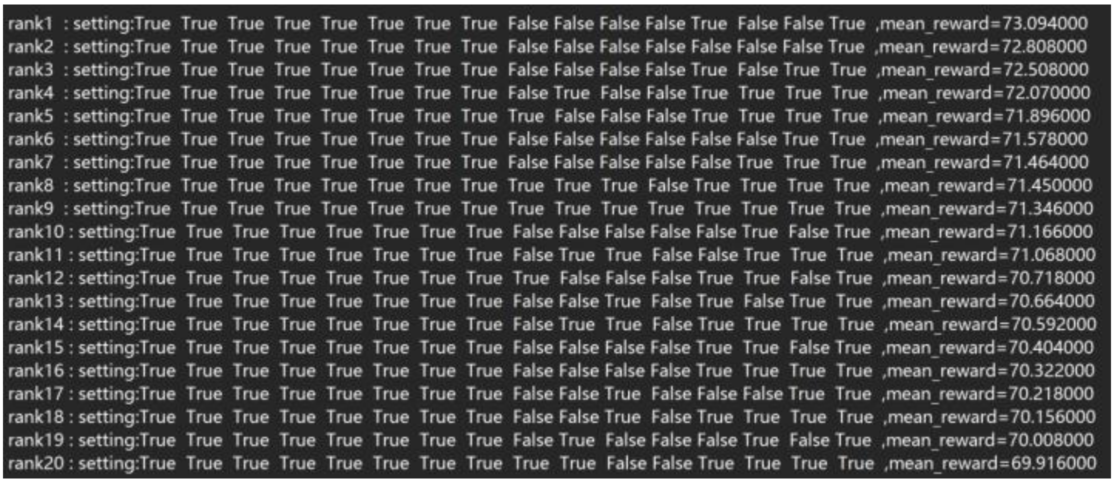
组装最终成品时涉及到两个决策：[是否检测成品]、[是否拆解成品]。

我们的一次模拟将以两种结果作为模拟终止结算的条件：产品卖出、产品废弃。

考虑到我们的模拟逻辑，我们将主动排除一些决策设置。例如，如果决定检测并拆解半成品一，那么就一定要对三个零件都做检测，否则会出现次品零件不断在装配环节循环，导致成本趋于无穷大的死循环情况。

再做一些其他的说明。如果我们决定检测零件，发现次品零件以后我们的设定是按照零件成本重新购入对应零件，保证生产环节的流畅。这样做的结果就会导致在这一轮生产中以更高的成本，更低的净利润卖出一件产品。我们认为这样模拟是符合逻辑的，因为检测将会导致更高的生产成本；如果我们在最终成品阶段检测出了次品，我们的拆解不会跨越一个阶段以上，即只会将其拆解为半成品，不会考虑进一步拆解为零件，这个假设一方面是模拟程序复杂度的考虑，另一方面是认为这种做法现实意义不大；如果次品的最终成品到了客户的手上进行返厂，我们也对情形做了简化考虑，倘若决策中会对成品进行拆解，那么我们将在总成本中重新加入拆解费用、成品的装配费用和调换损失，但如果决策中不对成品进行拆解，我们将以最高的成本重新生产一个成品来为用户进行替换。

这里附上模拟结果的部分排名，布尔值的次序分别代表了八个零件的检测决策、四个半成品/成品的检测决策和四个半成品/成品的拆解决策。



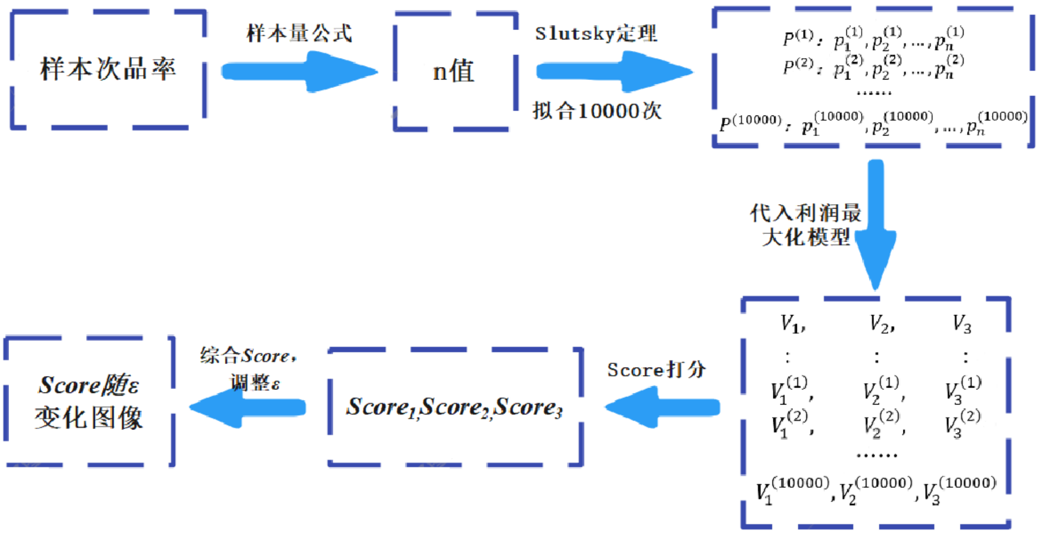
```
rank1  : setting:True  True  True  True  True  True  True  True  False False False False True  False False True  ,mean_reward=73.094000
rank2  : setting:True  True  True  True  True  True  True  True  False False False False False False False True  ,mean_reward=72.808000
rank3  : setting:True  True  True  True  True  True  True  True  False False False False True  False True  True  ,mean_reward=72.508000
rank4  : setting:True  True  True  True  True  True  True  True  False True  False False True  True  True  True  ,mean_reward=72.070000
rank5  : setting:True  True  True  True  True  True  True  True  True  False False False True  True  True  True  ,mean_reward=71.896000
rank6  : setting:True  True  True  True  True  True  True  True  False False False False False False True  True  ,mean_reward=71.578000
rank7  : setting:True  True  True  True  True  True  True  True  False False False False False True  True  True  ,mean_reward=71.464000
rank8  : setting:True  True  True  True  True  True  True  True  True  True  True  False True  True  True  True  ,mean_reward=71.450000
rank9  : setting:True  True  True  True  True  True  True  True  True  True  True  True  True  True  True  True  ,mean_reward=71.346000
rank10 : setting:True  True  True  True  True  True  True  True  False False False False False True  False True  ,mean_reward=71.166000
rank11 : setting:True  True  True  True  True  True  True  True  False True  True  False False True  True  True  ,mean_reward=71.068000
rank12 : setting:True  True  True  True  True  True  True  True  True  False False False True  True  False True  ,mean_reward=70.718000
rank13 : setting:True  True  True  True  True  True  True  True  False False True  False True  False True  True  ,mean_reward=70.664000
rank14 : setting:True  True  True  True  True  True  True  True  False True  True  False True  True  True  True  ,mean_reward=70.592000
rank15 : setting:True  True  True  True  True  True  True  True  False False False False True  True  False True  ,mean_reward=70.404000
rank16 : setting:True  True  True  True  True  True  True  True  False False False False True  True  True  True  ,mean_reward=70.322000
rank17 : setting:True  True  True  True  True  True  True  True  False False True  False False False True  True  ,mean_reward=70.218000
rank18 : setting:True  True  True  True  True  True  True  True  False False True  False True  True  True  True  ,mean_reward=70.156000
rank19 : setting:True  True  True  True  True  True  True  True  False True  False False False True  False True  ,mean_reward=70.008000
rank20 : setting:True  True  True  True  True  True  True  True  True  True  False False True  True  True  True  ,mean_reward=69.916000
```

5.4 **问题四：**总体次品率未知的利润最大化模型检测

问题四是问题二、三的延伸。在问题二、三中，零件总体的次品率已知，由此建立了利润最大化模型。但在本问中，零件总体的次品率未知，已知的次品率是样本次品率。为了解决问题，我们以一种情形为例，充分利用之前模型，模拟得到多组总体次品率后代入问题二、三模型。我们将检验原先问题二、三各个情形中表现良好(前三名)的决策组合，模拟它们在变异的真实次品率下的性能变化，以此来引入一个有参考价值的决策维度。此外，我们也将记录前三名的总体性能在次品率变异性的变化，以此来给出一个更加具体的样本量选择方案。即我们会

利用误差ε扰动进行灵敏度分析，由图像解释对误差ε进行筛选。

5.4.1 模型搭建


样本次品率
样本量公式
n值
Slutsky定理
拟合10000次
$P^{(1)}$: $p_1^{(1)}, p_2^{(1)}, \dots, p_n^{(1)}$
$P^{(2)}$: $p_1^{(2)}, p_2^{(2)}, \dots, p_n^{(2)}$
……
$P^{(10000)}$: $p_1^{(10000)}, p_2^{(10000)}, \dots, p_n^{(10000)}$
代入利润最大化模型
V_1, V_2, V_3
$V_1^{(1)}$, $V_2^{(1)}$, $V_3^{(1)}$
$V_1^{(2)}$, $V_2^{(2)}$, $V_3^{(2)}$
……
$V_1^{(10000)}, V_2^{(10000)}, V_3^{(10000)}$
Score打分
$Score_1, Score_2, Score_3$
综合Score，调整ε
Score随ε变化图像

Step 1：次品率随机性的引入

只考虑一组样本次品率。问题四已知样本次品率，为了契合问题二、三模型需要由此得到为总体次品率。利用 **Slutsky 定理**，可以得到总体次品率与样本次品率渐进公式：

$$p_i \sim N\left(p_i', \frac{p_i'(1-p_i')}{n_i}\right) \tag{36}$$

根据**样本量公式**，设置ε可以解出满足抽样要求的最小样本量n，代入上述正态分布，我们将获得给定误差ε下，次品率的变异机制。以问题二为例，给定情形一，我们将零件一、零件二、成品(装配过程)的次品率根据这一随机机制生成，实现未知真实次品率的模拟。固定决策组合与情形，对于一套给定的随机次品率，我们可以通过大量重复模拟，获得这个决策组合对于这套次品率的期望收益。再重复多次随机次品率的生成，我们便能获得这个决策组合在给定情形下，面对ε带来的次品率偏差，所能获得的期望收益。

具体实现上，每次模拟求出一组p_i，故可根据一种情形中的一组估计次品率p_i'，随机产生出 10000 组真实次品率p_i。记第一组$p_i^{(1)}$为向量$P^{(1)}$，第k组$p_i^{(k)}$为向量$P^{(k)}$，……以此类推，最后一组为向量$P^{(10000)}$。

Step 2:引入随机次品率后对决策组合重新进行蒙特卡洛模拟

现利用每组$P^{(k)}$，代入第二问的模型中求得的表现比较优秀的决策，可分别求出它们各自的期望利润。把第 k 组随机次品率向量$P^{(k)}$情形下，原先前三优秀的决策组合获得的利润记为$V_1^{(k)}$、$V_2^{(k)}$、$V_3^{(k)}$，利用不同的$P^{(k)}$可以获得原先前三优秀的决策组合在随机次品率的情形下的三个收益向量。

Step 3:构造全新的决策评价维度

针对以上生成的三个收益向量，按照其中各含有 10000 个元素，进行期望 E 和方差 Var 的计算。

$$E(X)=\sum_{i}x_{i}\cdot P(X=x_{i}) \tag{37}$$

$$\mathrm{Var}(X)=E[(X-\mu)^{2}]=\sum_{i}(x_{i}-\mu)^{2}\cdot P(X=x_{i}) \tag{38}$$

能够得到相应E_i、$Var_i(i=1,2,3)$。

Step 4:均值-方差公式对前三大利润向量打分并总分

为了重新考量每种决策组合的优劣度，我们利用**均值-方差公式**对每种决策组合进行打分。该公式考量每一向量中的元素的两方面特征：期望与方差。其中期望表示这一决策组合在全局设置与给定随机机制下的平均收益能力，故对利润评分起正向作用故为正，方差表示了这一决策组合收到随机机制的影响而产生的波动幅度，故给其加上负系数来影响得分函数。构建该公式如下：

$$Score_{i}=E_{i}-\lambda{\sigma_{i}}^{2} \tag{39}$$

由此可计算出前三大利润分别对应得分$Score_i$。并且可以获得一个总得分：

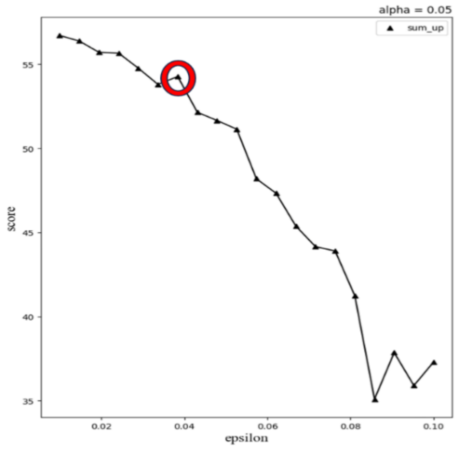
$$Score=Score_{1}+Score_{2}+Score_{3} \tag{40}$$

Step 5:灵敏度分析——使用不同的误差ε重复 Step1-4

由 Step1 Slutsky 定理及样本量公式可知，企业的零件抽样数应当是误差ε的函数，且具负相关性。设定 1%-10%的允许误差ε的范围，记为ε_1、ε_2、…、ε_{20}。设定针对一组样本次品率$P'^{(k)}$计算$P^{(k)}$时，使用的允许误差ε相同，即每次灵敏度分析中，变换误差要针对一组样本次品率$P'^{(k)}$数据。重复 Step1-4 进行 20 次运算。

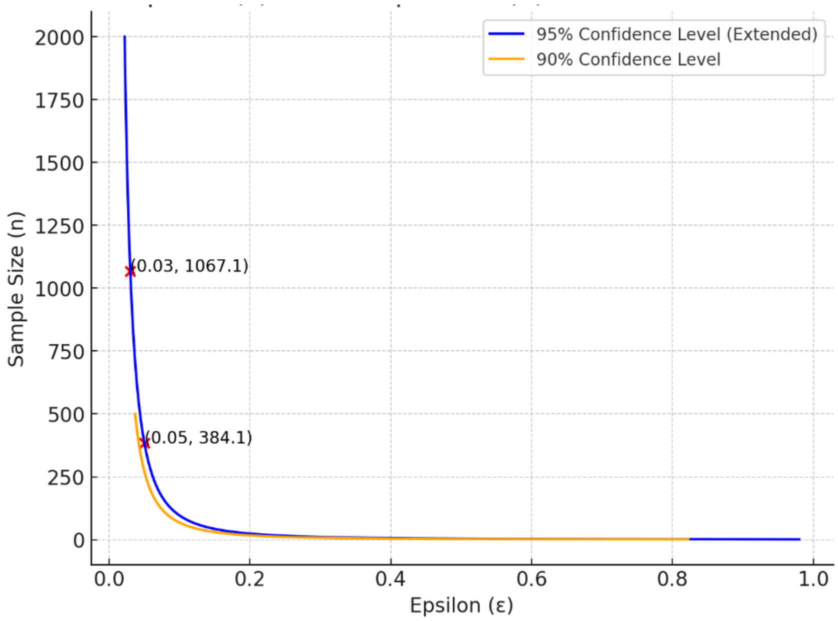
由此，针对一组$P'^{(k)}$每次变换一个允许的误差ε，我们可以得到一个得分$Score$值，进行 20 次误差变换即可以得到 20 个打分值。可以绘制出 20 个得分$Score$与 20 个允许误差ε的函数关系图像：

5.4.2 图像分析



该图描述了不同的ε设置下，模拟而得的三个收益向量的总和的变化趋势。随着ε的增大，所有决策组合的总体性能随之下降，但是考虑到我们使用的抽样方法，越大的ε将要求越少的样本量，所以我们希望找到ε扩大的极限，只要我们还能接受总体决策组合因其性能衰减的幅度。我们的依据是寻找“拐点”，即下降速度突然(开始)加快的位置，这将能通过图像直观定位到。上图是一个例子，由于这种决策方式涉及主观判断，所以我们建议截取一段ε的变化范围，来得到对应的样本量范围。

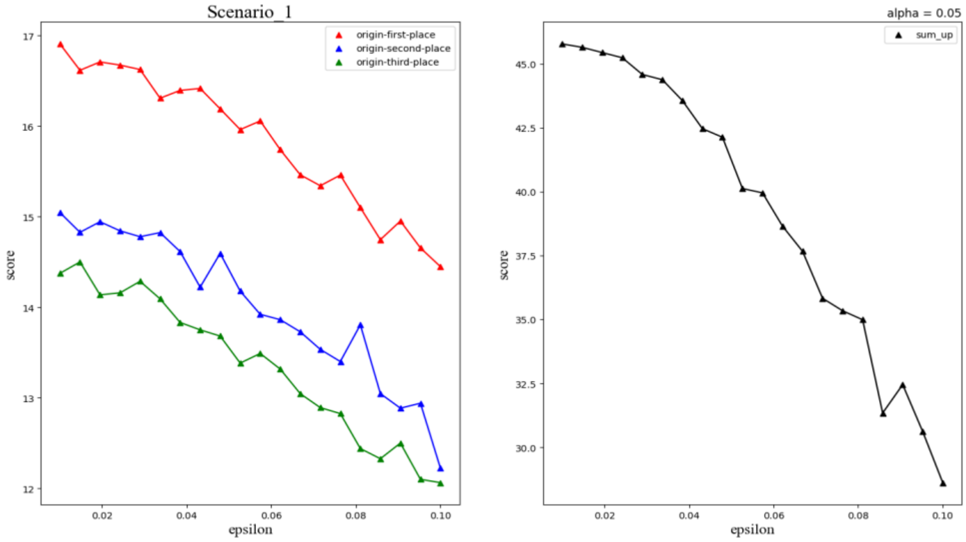
由图表可明确看到，在信度为95%时，ε在0.03-0.05时，n范围为384.1-1067.1之间。



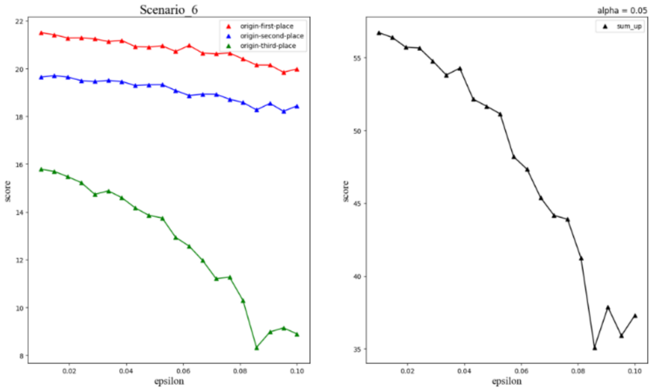
为了进一步说明，我们将对问题二中的情形一、情形六，以及问题三中的情形分别做分析。

1）问题二情形一：

根据上图，可以看到，在原始固定次品率情形下表现最佳的三个决策组合，随着ε扰动的加剧，它们的得分发生了类似的变化趋势，这可能说明它们应对未知次品率的能力没有明显差异，故而在这个情形下，我们不会改变我们的决策，依然会采用整体性能最好的决策方案。



2）问题二情形六：

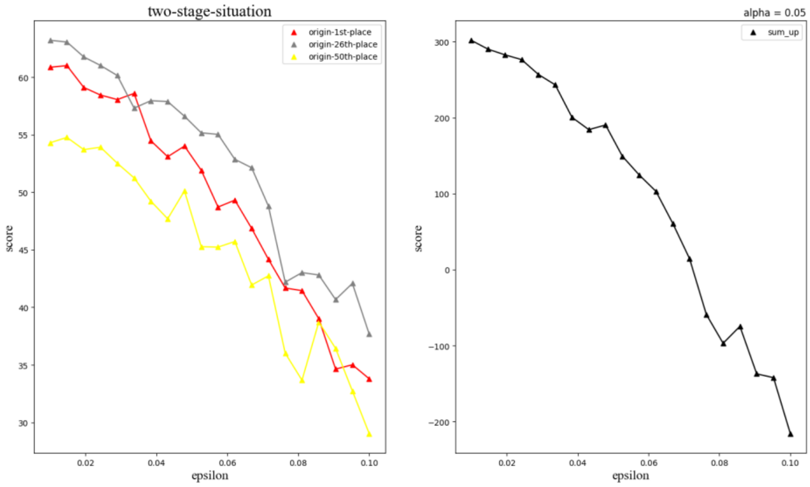


根据上图，可以看到原先表现最佳的三个决策组合中，前两名依然保持了优势，但是第三名却随着ε扰动的加剧，性能发生了骤降，我们可以在问题二的输出中找到这个决策：

```
decisions setting:
part_1_test, part_2_test,   product_test,   bad_product_disassemble
True                  True                  True                     True
mean_return= 20.2924
True                  True                 True                     False
mean_return= 20.9606
True                   True                  False                    True
mean_return= 22.6741(rank 2)
True                  True                  False                    False
mean_return= 23.6904(rank 1)
True                  False                True                     False
mean_return= 19.1128
True                   False                 False                    False
mean_return= 21.1158(rank 3)
False                True                 True                     False
mean_return= 18.883
False                 True                  False                    False
mean_return= 20.467
False                 False                True                     False
mean_return= 17.1824
False                 False                 False                    False
mean_return= 18.8656
```

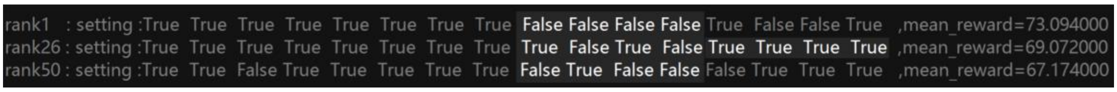
第三名对应的决策是零件一检测 True、零件二检测 False、成品检测 False、成品拆解 False。而前两名至少都对两个零件做了检测，确实符合我们的直觉，当零件的次品率提升，如果不对零件做检测，将很大概率提高后续费用(替换费用)出现的概率。因此，尽管我们的最终决策不会改变，但是我们却发现了原先表现良好的决策确实可能因为次品率的随机异动导致性能的急剧下降。

3）问题三情形：



由于问题三的情形存在大量的组合可能，故而我们挑选了在问题三中表现最好的、第二十六好的和第五十好的决策组合。

观察上图，我们发现第二十六好的决策组合在大多数ε的设定下，都能反超原先的第一名。分析可能原因如下：



```
rank1  : setting :True  True  True  True  True  True  True  True  False False False False True  False False True  ,mean_reward=73.094000
rank26 : setting :True  True  True  True  True  True  True  True  True  False True  False True  True  True  True  ,mean_reward=69.072000
rank50 : setting :True  True  False True  True  True  True  True  False True  False False False True  True  True  ,mean_reward=67.174000
```

图中左侧高亮部分分别代表“对半成品一进行检测”、“对半成品二进行检测”、“对半成品三进行检测”、“对最终成品进行检测”。

图中右侧高亮区域分别代表“对半成品一进行拆解”、“对半成品二进行拆解”、“对半成品三进行拆解”、“对最终成品进行拆解”。

第二十六好的决策显然会更多地对半成品进行检测，并且会对每一个次品做拆解。这种决策面对问题三中固定的、较低的次品率而言可能会徒增费用，导致期望收益较低。但是在加入次品率的随机扰动以后，这些决策就可能会显著提高生产的良品率，避免后期的替换损失，实现更好的效果。

所以在这样的条件下，我们认为，原先第二十六好的决策组合将会成为更好的决策组合。

5.4.3 综合描述

综上，我们做出如下判断，对于问题二的情形，次品率随机性的影响并不显著，我们仍然可以采取原来最佳的决策组合。但是对于问题三，更加复杂的生产过程导致随机性可能会对生产过程产生巨大的负面影响，这种时候对半成品和成品的检测就显得十分关键，所以我们认为应当在做好所有零件的检测的基础上，显著增加半成品和成品的检测比例(比如至少检测其中两项)，而且最好增加拆解的比例以应对更频繁出现的次品。

六、 模型评价

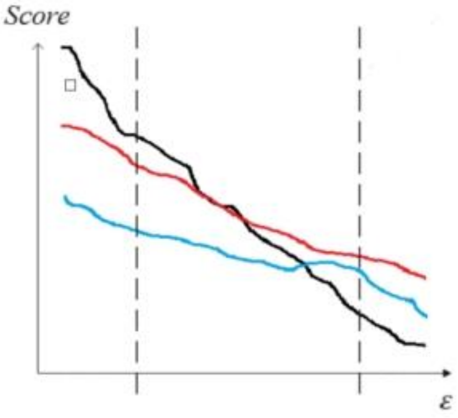
针对问题二、问题三中的优化模型，我们尽可能完善地考虑了生产中的影响因素，我们希望将它们悉数呈现目标函数中，但是这也导致我们的建模变得十分复杂，这影响了我们对结果的可解释性。但是考虑到多工序、多零件问题的模拟成本是指数型增长的，我们建立的优化模型依然提供了一个可推广的建模思路。

对于问题四，显然未知的次品率更符合实际生产中的真实情况，我们建立了一种基于蒙特卡洛模拟的评价方法，将不同决策组合随着误差扰动的变化趋势可视化，提出了选择理想误差的创新评价维度。

基于问题四的结果，我们有如下启发性思考。根据我们设计的这套评价维度，以及根据它对问题二和问题三中的决策组合进行了重新的审视，我们可以作出如下总结：

（1）由抽样检查得到的次品率所反映的真实次品率的随机性会影响决策组合的性能，这可以成为样本量选择的一个考虑维度；

（2）不同的决策组合受到真实次品率随机性的影响幅度不同，基于这一点，这可以成为评价决策组合的一个具有统计意义的参考标准；如果在某个全局设置下，若干个决策组合的得分随着扰动ε的变化呈现出类似如下的趋势：



我们将有理由说，相比较黑线代表的决策组合，红线代表的决策组合会是该情形下更优秀的决策组合。

七、 参考文献

[1] 姜启源，谢金星，叶俊.数学模型[M].北京：高等教育出版社，2018.

[2] 王夏阳,傅 科,梁桂添.原材料价格波动下的库存与生产联合决策[J].管理科学学报,2016,19(12):47~58

[3]李佳欣.大数据技术在制造型企业战略决策和生产经营中的应用探讨[J].信息系统工程,2024,(07):37-39.

[4]薛凤,靖富营.生产能力约束下易逝品动态批量决策研究[J].价值工程,2024,43(18):159-162.

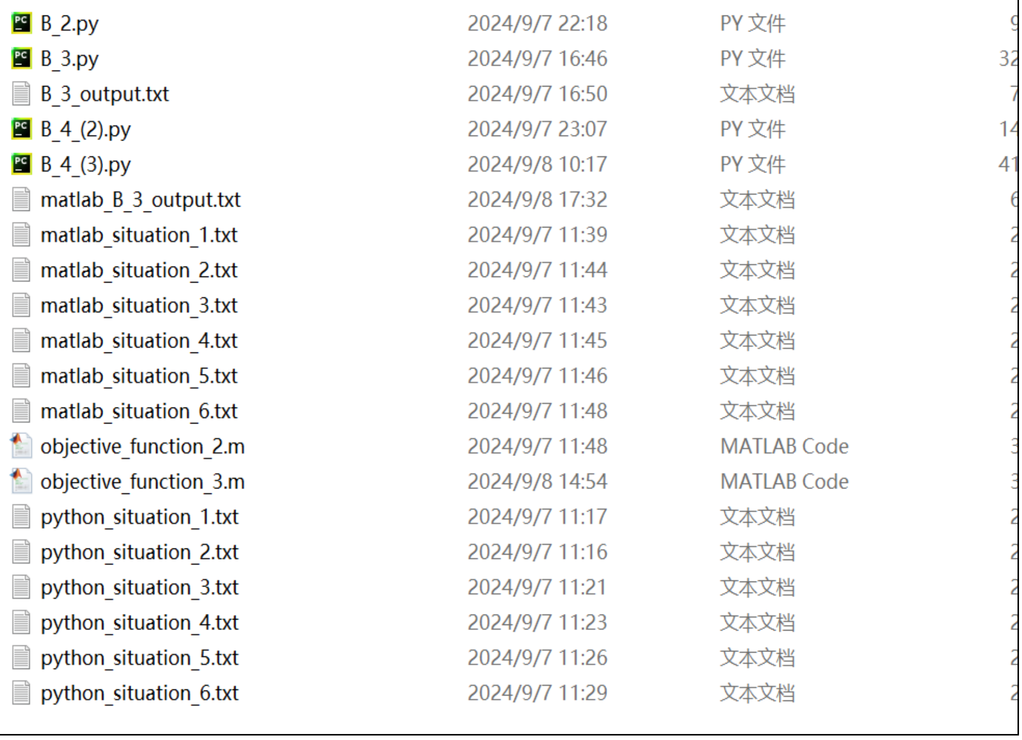
[6]Winston, W. L. (2004). Operations research: Applications and algorithms (4th ed.). Brooks/Cole.

[7]王夏阳, 傅科, 梁桂添. (2016). 原材料价格波动下的库存与生产联合决策. 管理科学学报, 19(12), 47–58.

附录

附录 1

介绍：支撑材料的文件列表



B_2.py	2024/9/7 22:18	PY 文件	9
B_3.py	2024/9/7 16:46	PY 文件	32
B_3_output.txt	2024/9/7 16:50	文本文档	7
B_4_(2).py	2024/9/7 23:07	PY 文件	14
B_4_(3).py	2024/9/8 10:17	PY 文件	41
matlab_B_3_output.txt	2024/9/8 17:32	文本文档	6
matlab_situation_1.txt	2024/9/7 11:39	文本文档	2
matlab_situation_2.txt	2024/9/7 11:44	文本文档	2
matlab_situation_3.txt	2024/9/7 11:43	文本文档	2
matlab_situation_4.txt	2024/9/7 11:45	文本文档	2
matlab_situation_5.txt	2024/9/7 11:46	文本文档	2
matlab_situation_6.txt	2024/9/7 11:48	文本文档	2
objective_function_2.m	2024/9/7 11:48	MATLAB Code	3
objective_function_3.m	2024/9/8 14:54	MATLAB Code	3
python_situation_1.txt	2024/9/7 11:17	文本文档	2
python_situation_2.txt	2024/9/7 11:16	文本文档	2
python_situation_3.txt	2024/9/7 11:21	文本文档	2
python_situation_4.txt	2024/9/7 11:23	文本文档	2
python_situation_5.txt	2024/9/7 11:26	文本文档	2
python_situation_6.txt	2024/9/7 11:29	文本文档	2

附录 2

Python 模拟代码

B_2.py(对问题二中的生产流程进行模拟)

```
import numpy as np
class one_circle_in_manufacturing:
    def __init__(self, part_1_test = None, part_2_test = None, product_test  = None,
bad_product_disassemble = None ,part_1=None,part_2=None,reassemble_time = 0):
        # 参数定义
        self.part_1_test = part_1_test
        self.part_2_test = part_2_test
        self.product_test = product_test
        self.bad_product_disassemble = bad_product_disassemble
        self.total_cost = 0
        self.part_1_cost = 4
        self.part_2_cost = 18
        self.assemble_cost = 6
        self.part_1_test_cost = 2
        self.part_2_test_cost = 3
        self.product_test_cost = 3
        self.exchange_cost = 6
        self.disassemble_cost = 40
        self.product_return = 56
        self.part_1_bad_ratio = 0.2
        self.part_2_bad_ratio = 0.2
        self.product_bad_ratio = 0.2

    # 成本首先是两个零件的购买单价
        self.total_cost = self.part_1_cost+self.part_2_cost
        self.part_1 = part_1
        self.part_2 = part_2
        self.product = None
        self.reassemble_time = reassemble_time
    # question.2_step.1
    def given_two_parts_get_one_product(self,total_cost,part_1 = None,part_2 = None):
        if part_1 == None:
            part_1 = np.random.binomial(1, (1 - self.part_1_bad_ratio), 1)[0]
            self.part_1 = part_1
        if part_2 == None:
            part_2 = np.random.binomial(1, (1 - self.part_2_bad_ratio), 1)[0]
            self.part_2 = part_2
        # 如果对零件一做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现次品
        if self.part_1_test == True:
            #做检测的话将确保 part_1 是良品
            while part_1 != 1:
                total_cost += self.part_1_cost
                part_1 = np.random.binomial(1, (1 - self.part_1_bad_ratio), 1)[0]
                self.part_1 = part_1

        # 如果对零件二做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现次品
        if self.part_2_test == True:
            # 做检测的话将确保 part_2 是良品
            while part_2 != 1:
                total_cost += self.part_2_cost
                part_2 = np.random.binomial(1, (1 - self.part_2_bad_ratio), 1)[0]
                self.part_2 = part_2

        # 只有零件一和零件二都是良品，才能保证成品是良品
        if self.part_1 == 1 & self.part_2 == 1:
            self.product = np.random.binomial(1, (1-self.product_bad_ratio), 1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product[0]]
            return return_list
        elif part_1 == 0 or part_2 == 0:
            self.product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product]
            return return_list

    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.product_test == True:
            total_cost += self.product_test_cost
            if product == 0:
                return ['defective',total_cost]
            else:
                return ['good',total_cost]
        else:
            return ['unknown',total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return ['disassembled',total_cost]
        else:
            return ['abandoned',total_cost]

result = None
end_cost = 0
# while (result == 'sold' or result == 'abandoned'):
def start(part_1_test = None, part_2_test = None, product_test  = None,
bad_product_disassemble = None,part_1 = None,part_2 = None,reassemble_time = 0):
    environment = one_circle_in_manufacturing(part_1_test, part_2_test, product_test,
bad_product_disassemble,part_1,part_2,reassemble_time)
    total_cost = environment.total_cost

    #进行零件检查到组装过程
    after_step_assemble =
environment.given_two_parts_get_one_product(total_cost,part_1,part_2)
    # if (environment.part_1 == 0 or environment.part_2 == 0 or environment.product ==
0):
    #     print("捕捉到异常")
    total_cost = after_step_assemble[0]
    curr_product = after_step_assemble[1]

    #对产品装配进行检查
    after_step_product_test = environment.product_test_procedure(total_cost,curr_product)
    product_test_result = after_step_product_test[0]
    total_cost = after_step_product_test[1]

    if product_test_result == ″good″:
        result = ’sold’
        end_cost = total_cost - environment.product_return
        return end_cost, result
    elif product_test_result ==  ’defective’:
        after_defective_procedure = environment.defective_procedure(total_cost)
        dealing = after_defective_procedure[0]
        total_cost = after_defective_procedure[1]
        if dealing == ’disassembled’:
            end_cost,result =
start(environment.part_1,environment.part_2,environment.reassemble_time)
            end_cost = total_cost + end_cost
            return end_cost,result

        elif dealing == ’abandoned’:
            end_cost = total_cost - 0
            result = ″abandoned″
            return end_cost,result
    elif product_test_result == ’unknown’:
        if environment.product == 0:
            total_cost = total_cost + environment.exchange_cost
            re_defective_procedure = environment.defective_procedure(total_cost)
            dealing = re_defective_procedure[0]
            total_cost = re_defective_procedure[1]

            if dealing == ’disassembled’:
                result = None
                while result != ’sold’:
                    end_cost,result = start(environment.part_1,
environment.part_2,environment.reassemble_time)
                    total_cost += end_cost
                if environment.reassemble_time == 0:
                    end_cost = total_cost
                else:
                    end_cost = total_cost



                return end_cost,result

            elif dealing == ’abandoned’:
                end_cost = total_cost - 0
                result = 'abandoned'
                return end_cost,result
        elif environment.product == 1:
            end_cost = total_cost - environment.product_return
            result = 'sold'
            return end_cost,result

# trial_list = []
# for i in range(0,10000):
#     one_trial = start()[0]
#     trial_list.append(one_trial)
# print(trial_list,"\nmean=",sum(trial_list)/10000)

#setting = part_1_test , part_2_test , product_test , bad_product_disassemble
#if bad_product_disassemble = True,both part_1_test and part_2_test should be True

print("with global settings:(situation 6)")
print('total_cost = 0 \n',
      'part_1_cost = 4 \n',
      'part_2_cost = 18 \n',
      'assemble_cost = 6 \n',
      'part_1_test_cost = 2 \n',
      'part_2_test_cost = 3 \n',
      'product_test_cost = 3 \n',
      'exchange_cost = 10 \n',
      'disassemble_cost = 40 \n',
      'product_return = 56 \n',
      'part_1_bad_ratio = 0.05 \n',
      'part_2_bad_ratio = 0.05 \n',
      'product_bad_ratio = 0.05 \n')
print('decisions setting:')
print('part_1_test,part_2_test,product_test,bad_product_disassemble\n')
for decision_1 in [True,False]:
    for decision_2 in [True,False]:
        for decision_3 in [True,False]:
            for decision_4 in [True,False]:
                if (decision_4 == True) and ((decision_1 and decision_2) == False):
                    continue
                trial_list = []

                for i in range(0, 10000):
                    one_trial = start(part_1_test = decision_1, part_2_test =
decision_2 , product_test = decision_3, bad_product_disassemble = decision_4)[0]
```

```
                    trial_list.append(one_trial)
                print("%0-11s %0-11s %0-12s %0-
23s"%(decision_1,decision_2,decision_3,decision_4),"mean_return=", -sum(trial_list) /
10000)
```

B_3.py(对问题三中的生产流程进行模拟)

```
import numpy as np
import sys
import os
#定义 B_3_1 过程的模拟
class one_circle_in_B_3_1:
    def __init__(self, part_1_test = None, part_2_test = None, part_3_test =
None,half_product_1_test  = None, bad_product_disassemble =
None ,part_1=None,part_2=None,part_3 = None,reassemble_time = 0):
        # 参数定义
        self.part_1_test = part_1_test
        self.part_2_test = part_2_test
        self.part_3_test = part_3_test
        self.product_test = half_product_1_test
        self.bad_product_disassemble = bad_product_disassemble

        self.total_cost = 0
        self.part_1_cost = 2
        self.part_2_cost = 8
        self.part_3_cost = 12
        self.assemble_cost = 8
        self.part_1_test_cost = 1
        self.part_2_test_cost = 1
        self.part_3_test_cost = 2
        self.product_test_cost = 4
        self.disassemble_cost = 6

        self.part_1_bad_ratio = 0.1
        self.part_2_bad_ratio = 0.1
        self.part_3_bad_ratio = 0.1
        self.product_bad_ratio = 0.1

        self.part_1 = part_1
        self.part_2 = part_2
        self.part_3 = part_3
        self.product = None
        self.reassemble_time = reassemble_time

    # question.2_step.1
    def given_three_parts_get_one_product(self,total_cost,part_1 = None,part_2 = None,part_3 = None):
        if part_1 == None:
            part_1 = np.random.binomial(1, (1 - self.part_1_bad_ratio), 1)[0]
            self.part_1 = part_1
            self.total_cost += self.part_1_cost
        if part_2 == None:
            part_2 = np.random.binomial(1, (1 - self.part_2_bad_ratio), 1)[0]
            self.part_2 = part_2
            self.total_cost += self.part_2_cost
        if part_3 == None:
            part_3 = np.random.binomial(1, (1 - self.part_3_bad_ratio), 1)[0]
            self.part_3 = part_3
            self.total_cost += self.part_3_cost
        # 如果对零件一做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现次品
        if self.part_1_test == True:
            #做检测的话将确保 part_1 是良品
            while part_1 != 1:
                total_cost += self.part_1_cost
                part_1 = np.random.binomial(1, (1 - self.part_1_bad_ratio), 1)[0]
                self.part_1 = part_1

        # 如果对零件二做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现次品
        if self.part_2_test == True:
            # 做检测的话将确保 part_2 是良品
            while part_2 != 1:
                total_cost += self.part_2_cost
                part_2 = np.random.binomial(1, (1 - self.part_2_bad_ratio), 1)[0]
                self.part_2 = part_2

        # 如果对零件三做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现次品
        if self.part_3_test == True:
            # 做检测的话将确保 part_1 是良品
            while part_3 != 1:
                total_cost += self.part_3_cost
                part_3 = np.random.binomial(1, (1 - self.part_3_bad_ratio), 1)[0]
                self.part_3 = part_3
        total_cost += self.total_cost
        # 只有零件一和零件二都是良品，才能保证成品是良品
        if self.part_1 == 1 & self.part_2 == 1 & self.part_3 == 1:
            self.product = np.random.binomial(1, (1-self.product_bad_ratio), 1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product[0]]
            return return_list
        elif part_1 == 0 or part_2 == 0 or part_3 == 0:
            self.product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product]
            return return_list

    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.product_test == True:
            total_cost += self.product_test_cost
            if product == 0:
                return ['defective',total_cost]
            else:
                return ['good',total_cost]
        else:
            return ['unknown',total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return ['disassembled',total_cost]
        else:
            return ['abandoned',total_cost]

#定义 B_3_2 过程的模拟
class one_circle_in_B_3_2:
    def __init__(self, part_4_test = None, part_5_test = None, part_6_test =
None,half_product_2_test  = None, bad_product_disassemble =
None ,part_4=None,part_5=None,part_6 = None,reassemble_time = 0):
        # 参数定义
        self.part_4_test = part_4_test
        self.part_5_test = part_5_test
        self.part_6_test = part_6_test
        self.product_test = half_product_2_test
        self.bad_product_disassemble = bad_product_disassemble
        self.total_cost = 0
        self.part_4_cost = 2
        self.part_5_cost = 8
        self.part_6_cost = 12
        self.assemble_cost = 8
        self.part_4_test_cost = 1
        self.part_5_test_cost = 1
        self.part_6_test_cost = 2
        self.product_test_cost = 4
        self.disassemble_cost = 6
        self.part_4_bad_ratio = 0.1
        self.part_5_bad_ratio = 0.1
        self.part_6_bad_ratio = 0.1
        self.product_bad_ratio = 0.1
        self.part_4 = part_4
        self.part_5 = part_5
        self.part_6 = part_6
        self.product = None
        self.reassemble_time = reassemble_time
    # question.2_step.1
    def given_three_parts_get_one_product(self,total_cost,part_4 = None,part_5 =
None,part_6 = None):
        if part_4 == None:
            part_4 = np.random.binomial(1, (1 - self.part_4_bad_ratio), 1)[0]
            self.part_4 = part_4
            self.total_cost += self.part_4_cost
        if part_5 == None:
            part_5 = np.random.binomial(1, (1 - self.part_5_bad_ratio), 1)[0]
            self.part_5 = part_5
            self.total_cost += self.part_5_cost
        if part_6 == None:
            part_6 = np.random.binomial(1, (1 - self.part_6_bad_ratio), 1)[0]
            self.part_6 = part_6
            self.total_cost += self.part_6_cost
        # 如果对零件一做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现
次品
        if self.part_4_test == True:
            #做检测的话将确保 part_1 是良品
            while part_4 != 1:
                total_cost += self.part_4_cost
                part_4 = np.random.binomial(1, (1 - self.part_4_bad_ratio),
1)[0]
                self.part_4 = part_4
        # 如果对零件二做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现
次品
        if self.part_5_test == True:
            # 做检测的话将确保 part_2 是良品
            while part_5 != 1:
                total_cost += self.part_5_cost
                part_5 = np.random.binomial(1, (1 - self.part_5_bad_ratio),
1)[0]
                self.part_5 = part_5

        # 如果对零件三做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现
次品
        if self.part_6_test == True:
            # 做检测的话将确保 part_1 是良品
            while part_6 != 1:
                total_cost += self.part_6_cost
                part_6 = np.random.binomial(1, (1 - self.part_6_bad_ratio),
1)[0]
                self.part_6 = part_6

        total_cost += self.total_cost
        # 只有零件一和零件二都是良品，才能保证成品是良品
        if self.part_4 == 1 & self.part_5 == 1 & self.part_6 == 1:
            self.product = np.random.binomial(1, (1-self.product_bad_ratio), 1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product[0]]
            return return_list
        elif part_4 == 0 or part_5 == 0 or part_6 == 0:
            self.product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product]
            return return_list



    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.product_test == True:
            total_cost += self.product_test_cost
            if product == 0:
                return ['defective',total_cost]
            else:
                return ['good',total_cost]
        else:
            return [’unknown’,total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return [’disassembled’,total_cost]
        else:
            return [’abandoned’,total_cost]

#定义 B_3_3 过程的模拟
class one_circle_in_B_3_3:
    def __init__(self, part_7_test = None, part_8_test =
None,half_product_3_test  = None, bad_product_disassemble =
None ,part_7=None,part_8=None,reassemble_time = 0):
        # 参数定义
        self.part_7_test = part_7_test
        self.part_8_test = part_8_test

        self.product_test = half_product_3_test
        self.bad_product_disassemble = bad_product_disassemble

        self.total_cost = 0
        self.part_7_cost = 8
        self.part_8_cost = 12

        self.assemble_cost = 8
        self.part_7_test_cost = 1
        self.part_8_test_cost = 2

        self.product_test_cost = 4
        self.disassemble_cost = 6

        self.part_7_bad_ratio = 0.1
        self.part_8_bad_ratio = 0.1

        self.product_bad_ratio = 0.1

    # 成本首先是两个零件的购买单价
        self.total_cost = 0

        self.part_7 = part_7
        self.part_8 = part_8

        self.product = None
        self.reassemble_time = reassemble_time
    # question.2_step.1
    def given_three_parts_get_one_product(self,total_cost,part_7 = None,part_8 =
None):
        if part_7 == None:
            part_7 = np.random.binomial(1, (1 - self.part_7_bad_ratio), 1)[0]
            self.part_7 = part_7
            self.total_cost += self.part_7_cost
        if part_8 == None:
            part_8 = np.random.binomial(1, (1 - self.part_8_bad_ratio), 1)[0]
            self.part_8 = part_8
            self.total_cost += self.part_8_cost

        # 如果对零件七做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现
次品
        if self.part_7_test == True:
            #做检测的话将确保 part_1 是良品
            while part_7 != 1:
                total_cost += self.part_7_cost
                part_7 = np.random.binomial(1, (1 - self.part_7_bad_ratio),
1)[0]
                self.part_7 = part_7

        # 如果对零件二做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率出现
次品
        if self.part_8_test == True:
            # 做检测的话将确保 part_2 是良品
            while part_8 != 1:
                total_cost += self.part_8_cost
                part_8 = np.random.binomial(1, (1 - self.part_8_bad_ratio),
1)[0]
                self.part_8 = part_8

        total_cost += self.total_cost
        # 只有零件一和零件二都是良品，才能保证成品是良品
        if self.part_7 == 1 & self.part_8 == 1:
            self.product = np.random.binomial(1, (1-self.product_bad_ratio), 1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product[0]]
            return return_list
        elif part_7 == 0 or part_8 == 0:
            self.product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product]
            return return_list



    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.product_test == True:
            total_cost += self.product_test_cost
            if product == 0:
                return [’defective’,total_cost]
            else:
                return [’good’,total_cost]
        else:
            return [’unknown’,total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return [’disassembled’,total_cost]
        else:
            return [’abandoned’,total_cost]


#定义 B_3_4 过程(已经去掉半成品的生成和检测过程)
class one_circle_in_B_3_4:
    def __init__(self, half_product_1 = None,half_product_2 =
None,half_product_3 = None,final_product_test  = None, bad_product_disassemble =
None ,reassemble_time = 0):
        # 参数定义
        self.half_product_1 = half_product_1
        self.half_product_2 = half_product_2
        self.half_product_3 = half_product_3
        self.final_product_test = final_product_test
        self.bad_product_disassemble = bad_product_disassemble
        self.total_cost = 0
        self.assemble_cost = 8
        self.final_product_test_cost = 6
        self.disassemble_cost = 10
        self.final_product_bad_ratio = 0.1
        self.final_product_reward = 200
        self.exchange_cost = 40
    # 成本首先是两个零件的购买单价
        self.final_product = None
        self.reassemble_time = reassemble_time
    # question.2_step.1
    def given_three_parts_get_one_product(self,total_cost):
        # 只有半成品一二三都是良品，才能保证成品是良品
        if self.half_product_1 == 1 & self.half_product_2 == 1 &
self.half_product_3 == 1:
            self.final_product = np.random.binomial(1, (1-
self.final_product_bad_ratio), 1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.final_product[0]]
            return return_list
        elif self.half_product_1 == 0 or self.half_product_2 == 0 or
self.half_product_3 == 0:
            self.final_product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.final_product]
            return return_list



    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.final_product_test == True:
            total_cost += self.final_product_test_cost
            if product == 0:
                return [’defective’,total_cost]
            else:
                return [’good’,total_cost]
        else:
            return [’unknown’,total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return [’disassembled’,total_cost]
        else:
            return [’abandoned’,total_cost]
```

```
def start_B_3_1(part_1_test = True, part_2_test = True,part_3_test =
True,product_test  = True, bad_product_disassemble = True,part_1 = None,part_2 =
None,part_3 = None,reassemble_time = 0):
    environment = one_circle_in_B_3_1(part_1_test, part_2_test,
part_3_test,product_test,
bad_product_disassemble,part_1,part_2,part_3,reassemble_time)
    total_cost = environment.total_cost

    #进行零件检查到组装过程
    after_step_assemble =
environment.given_three_parts_get_one_product(total_cost,part_1,part_2,part_3)
    # if (environment.part_1 == 0 or environment.part_2 == 0 or
environment.part_3 == 0 or environment.product == 0):
    #     print(″捕捉到异常″)
    total_cost = after_step_assemble[0]
    curr_half_product = after_step_assemble[1]

    #对产品装配进行检查
    after_step_product_test =
environment.product_test_procedure(total_cost,curr_half_product)
    product_test_result = after_step_product_test[0]
    total_cost = after_step_product_test[1]

    if product_test_result == ″good″:
        result = 'half_product_1_OK’
        end_cost = total_cost
        return end_cost,
result,curr_half_product,environment.part_1,environment.part_2,environment.part_
3
    elif product_test_result ==  'defective':
        after_defective_procedure = environment.defective_procedure(total_cost)
        dealing = after_defective_procedure[0]
        total_cost = after_defective_procedure[1]
        if dealing == 'disassembled':
            end_cost,result,curr_half_product,_,_,_ =
start_B_3_1(part_1=environment.part_1,part_2=environment.part_2,part_3=environme
nt.part_3,reassemble_time = environment.reassemble_time)
            end_cost = total_cost + end_cost
            return
end_cost,result,curr_half_product,environment.part_1,environment.part_2,environm
ent.part_3

        elif dealing == 'abandoned':
            end_cost = total_cost - 0
            result = "half_product_1_abandoned"
            return
end_cost,result,None,environment.part_1,environment.part_2,environment.part_3
    elif product_test_result == 'unknown':
        end_cost = total_cost
        result = "half_product_1_unknown"
        return
end_cost,result,curr_half_product,environment.part_1,environment.part_2,environm
ent.part_3



def start_B_3_2(part_4_test = True, part_5_test = True,part_6_test =
True,product_test  = True, bad_product_disassemble = True,part_4 = None,part_5 =
None,part_6 = None,reassemble_time = 0):
    environment = one_circle_in_B_3_2(part_4_test, part_5_test,
part_6_test,product_test,
bad_product_disassemble,part_4,part_5,part_6,reassemble_time)
    total_cost = environment.total_cost

    #进行零件检查到组装过程
    after_step_assemble =
environment.given_three_parts_get_one_product(total_cost,part_4,part_5,part_6)
    # if (environment.part_4 == 0 or environment.part_5 == 0 or
environment.part_6 == 0 or environment.product == 0):
    #     print("捕捉到异常")
    total_cost = after_step_assemble[0]
    curr_half_product = after_step_assemble[1]

    #对产品装配进行检查
    after_step_product_test =
environment.product_test_procedure(total_cost,curr_half_product)
    product_test_result = after_step_product_test[0]
    total_cost = after_step_product_test[1]

    if product_test_result == "good":
        result = 'half_product_2_OK'
        end_cost = total_cost
        return end_cost,
result,curr_half_product,environment.part_4,environment.part_5,environment.part_
6
    elif product_test_result ==  'defective':
        after_defective_procedure = environment.defective_procedure(total_cost)
        dealing = after_defective_procedure[0]
        total_cost = after_defective_procedure[1]
        if dealing == 'disassembled':
            end_cost,result,curr_half_product,_,_,_ = start_B_3_2(part_4 =
environment.part_4, part_5 = environment.part_5, part_6 = environment.part_6,
reassemble_time = environment.reassemble_time)
            end_cost = total_cost + end_cost
            return
end_cost,result,curr_half_product,environment.part_4,environment.part_5,environm
ent.part_6

        elif dealing == 'abandoned':
            end_cost = total_cost - 0
            result = "half_product_2_abandoned"
            return
end_cost,result,None,environment.part_4,environment.part_5,environment.part_6
    elif product_test_result == 'unknown':
        end_cost = total_cost
        result = "half_product_2_unknown"
        return
end_cost,result,curr_half_product,environment.part_4,environment.part_5,environm
ent.part_6

def start_B_3_3(part_7_test = True, part_8_test = True,product_test  = True,
bad_product_disassemble = True,part_7 = None,part_8 = None,reassemble_time = 0):
    environment = one_circle_in_B_3_3(part_7_test, part_8_test,product_test,
bad_product_disassemble,part_7,part_8,reassemble_time)
    total_cost = environment.total_cost

    #进行零件检查到组装过程
    after_step_assemble =
environment.given_three_parts_get_one_product(total_cost,part_7,part_8)
    # if (environment.part_7 == 0 or environment.part_8 == 0 or
environment.product == 0):
    #     print("捕捉到异常")
    total_cost = after_step_assemble[0]
    curr_half_product = after_step_assemble[1]

    #对产品装配进行检查
    after_step_product_test =
environment.product_test_procedure(total_cost,curr_half_product)
    product_test_result = after_step_product_test[0]
    total_cost = after_step_product_test[1]

    if product_test_result == "good":
        result = 'half_product_3_OK'
        end_cost = total_cost
        return end_cost,
result,curr_half_product,environment.part_7,environment.part_8
    elif product_test_result == 'defective':
        after_defective_procedure = environment.defective_procedure(total_cost)
        dealing = after_defective_procedure[0]
        total_cost = after_defective_procedure[1]
        if dealing == 'disassembled':
            end_cost,result,curr_half_product,_,_ = start_B_3_3(part_7 =
environment.part_7, part_8 = environment.part_8, reassemble_time =
environment.reassemble_time)
            end_cost = total_cost + end_cost
            return
end_cost,result,curr_half_product,environment.part_7,environment.part_8

        elif dealing == 'abandoned':
            end_cost = total_cost - 0
            result = "half_product_3_abandoned"
            return end_cost,result,None,environment.part_7,environment.part_8
    elif product_test_result == 'unknown':
        end_cost = total_cost
        result = "half_product_3_unknown"
        return
end_cost,result,curr_half_product,environment.part_7,environment.part_8




def
startB_3_4(part_1_test=True,part_2_test=True,part_3_test=True,part_4_test=True,p
art_5_test=True,part_6_test=True,part_7_test=True,part_8_test=True,

half_product_1_test=True,half_product_2_test=True,half_product_3_test=True,half_
product_1_disassemble=True,half_product_2_disassemble=True,
               half_product_3_disassemble=True,final_product_test=True,
bad_product_disassemble = True ,reassemble_time = 0,part_1 = None,
               part_2 = None,part_3 = None,part_4 = None,part_5 = None,part_6 =
None,part_7 = None
               ,part_8 = None,part_9 = None):
    end_cost_1,result_1,half_product_1,part_1,part_2,part_3 =
start_B_3_1(part_1_test = part_1_test, part_2_test = part_2_test,part_3_test =
part_3_test,product_test  = half_product_1_test, bad_product_disassemble =
half_product_1_disassemble,
                                                                          part_1
```

```
= part_1,part_2=part_2,part_3=part_3)
    end_cost_2, result_2, half_product_2,part_4,part_5,part_6 =
start_B_3_2(part_4_test = part_4_test, part_5_test = part_5_test,part_6_test =
part_6_test,product_test  = half_product_2_test, bad_product_disassemble =
half_product_2_disassemble,

part_4=part_4,part_5=part_5,part_6=part_6)
    end_cost_3, result_3, half_product_3,part_7,part_8 = start_B_3_3(part_7_test
= part_7_test, part_8_test = part_8_test,product_test  = half_product_3_test,
bad_product_disassemble = half_product_3_disassemble,

part_7=part_7,part_8=part_8)

    if half_product_1 == None or half_product_2 == None or half_product_3 ==
None:
        return (end_cost_1+end_cost_2+end_cost_3), 'failed'
    environment_4 =
one_circle_in_B_3_4(half_product_1,half_product_2,half_product_3,final_product_t
est,bad_product_disassemble,reassemble_time)
    #装配
    assemble_result =
environment_4.given_three_parts_get_one_product(environment_4.total_cost)
    curr_total_cost = assemble_result[0] + end_cost_1 + end_cost_2 + end_cost_3
    curr_product = assemble_result[1]

    # if (curr_product == 0):
    #     print("出现次品成品！")

    #最终成品检验
    final_product_test =
environment_4.product_test_procedure(curr_total_cost,curr_product)
    test_result = final_product_test[0]
    curr_total_cost = final_product_test[1]

    if test_result == 'good':
        end_cost = curr_total_cost - environment_4.final_product_reward
        result = 'sold'
        return end_cost,result
    elif test_result == 'defective':
        defective_dealing = environment_4.defective_procedure(curr_total_cost)
        dealing = defective_dealing[0]
        curr_total_cost = defective_dealing[1]
        if dealing == 'disassembled':
            if (half_product_1 == 0)or(half_product_2 == 0)or(half_product_3
==0):
                return (curr_total_cost +4+4+4),'abandoned’
            else:
                another_assemble = np.random.binomial(1,0.9,1)[0]
                curr_total_cost += 8 #重新装配成本
                while( another_assemble != 1):
                    curr_total_cost += 10 #再次拆解
                    curr_total_cost += 8 #再次重新装配
                    another_assemble = np.random.binomial(1, 0.9, 1)[0]
                return (curr_total_cost-
environment_4.final_product_reward),'sold’

        elif dealing == 'abandoned':
            return curr_total_cost,'abandoned’



    elif test_result == 'unknown':
        #如果出现了次品到客户手上的情况，那么我们需要无偿为客户替换件，我们将以
最高成本的方式准备一件良品
        if environment_4.final_product == 0:
            curr_total_cost += environment_4.exchange_cost
            re_defective_procedure =
environment_4.defective_procedure(curr_total_cost)
            dealing = re_defective_procedure[0]
            total_cost = re_defective_procedure[1]

            if dealing == 'disassembled':
                return (total_cost - environment_4.final_product_reward + 12 + 8
+ 6),'problem’
            elif dealing == 'abandoned':
                return (total_cost -environment_4.final_product_reward + 64 + 11
+ 24 + 12 + 8 + 6),'problem’

        elif environment_4.final_product == 1:
            end_cost = curr_total_cost - environment_4.final_product_reward
            result = 'sold’
            return end_cost, result




product_1_test = [True,False]
product_2_test = [True,False]
product_3_test = [True,False]
product_4_test = [True,False]
```

```
product_5_test = [True,False]
product_6_test = [True,False]
product_7_test = [True,False]
product_8_test = [True,False]

half_product_1_test = [True,False]
half_product_2_test = [True,False]
half_product_3_test = [True,False]
final_product_test = [True,False]

half_product_1_disassemble = [True,False]
half_product_2_disassemble = [True,False]
half_product_3_disassemble = [True,False]
final_product_disassemble = [True,False]

count = 0
output_dic = dict()
for decision_1 in product_1_test:
    for decision_2 in product_2_test:
        for decision_3 in product_3_test:
            for decision_4 in product_4_test:
                for decision_5 in product_5_test:
                    for decision_6 in product_6_test:
                        for decision_7 in product_7_test:
                            for decision_8 in product_8_test:
                                for decision_9 in half_product_1_test:
                                    for decision_10 in half_product_2_test:
                                        for decision_11 in half_product_3_test:
                                            for decision_12 in
final_product_test:
                                                for decision_13 in
half_product_1_disassemble:
                                                    for decision_14 in
half_product_2_disassemble:
                                                        for decision_15 in
half_product_3_disassemble:
                                                            for decision_16 in
final_product_disassemble:
                                                                count += 1


print("setting:",count)
                                                                if (decision_13
== True) and ((decision_1 and decision_2 and decision_3) == False):
                                                                    continue
                                                                if (decision_14
== True) and ((decision_4 and decision_5 and decision_6) == False):
                                                                    continue
                                                                if (decision_15
== True) and ((decision_7 and decision_8) == False):
                                                                    continue
                                                                trial_list = []
                                                                for i in
range(0, 1000):
                                                                    end_cost,
result =
startB_3_4(part_1_test=decision_1,part_2_test=decision_2,part_3_test=decision_3,
part_4_test=decision_4,

part_5_test=decision_5,part_6_test=decision_6,part_7_test=decision_7,part_8_test
=decision_8,

half_product_1_test=decision_9,half_product_2_test=decision_10,half_product_3_te
st=decision_11,

final_product_test=decision_12,half_product_1_disassemble=decision_13,half_produ
ct_2_disassemble=decision_14,

half_product_3_disassemble=decision_15,bad_product_disassemble=decision_16)

trial_list.append(end_cost)
                                                                setting_label =
(’%0-6s’%(decision_1) +’%0-6s’%(decision_2) +’%0-6s’%(decision_3) +’%0-
6s’%(decision_4)

+’%0-6s’%(decision_5) +’%0-6s’%(decision_6) +’%0-6s’%(decision_7) +’%0-
6s’%(decision_8)

+’%0-6s’%(decision_9) +’%0-6s’%(decision_10) +’%0-6s’%(decision_11) +’%0-
6s’%(decision_12)

+’%0-6s’%(decision_13) +’%0-6s’%(decision_14) +’%0-6s’%(decision_15) +’%0-
6s’%(decision_16))

output_dic[setting_label] = -sum(trial_list) / 1000



#print(″product_1_test=%s,product_2_test=%s,product_3_test=%s,half_product_1_tes
t=%s,half_product_1_disassemble=%s″%(decision_1,decision_2,decision_3,decision_9
,decision_13))

#寻找前 n 大的值
n = 50

values = sorted(output_dic.items(), key=lambda item: item[1], reverse=True)

values = values[:n]

# print(values)

good_settings_and_benefits = {}
for value in values:
    good_settings_and_benefits[value[0]] = value[1]

# print(good_settings_and_benefits)

cnt = 0

fw = open(″B_3_output.txt″, ’w’)
for key,value in good_settings_and_benefits.items():
    cnt+=1
    print(″rank%0-2d : setting:%s,mean_reward=%f″%(cnt,key,value))
    fw.write(″rank%0-2d : setting:%s,mean_reward=%f″%(cnt,key,value))
    fw.write(″\n″)
fw.write(″\n″)
fw.close()
```

B_4_(2).py(在加入了次品率的扰动后重新对问题二中的生产流程进行模拟)

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

class one_circle_in_manufacturing:
    def __init__(self, part_1_test = None, part_2_test = None, product_test
= None, bad_product_disassemble = None ,
                 part_1=None,part_2=None,reassemble_time =
0,part_1_bad_ratio = 0.1,part_2_bad_ratio=0.1,product_bad_ratio = 0.1):
        # 参数定义
        self.part_1_test = part_1_test
        self.part_2_test = part_2_test
        self.product_test = product_test
        self.bad_product_disassemble = bad_product_disassemble
        self.total_cost = 0
        self.part_1_cost = 4
        self.part_2_cost = 18
        self.assemble_cost = 6
        self.part_1_test_cost = 2
        self.part_2_test_cost = 3
        self.product_test_cost = 3
        self.exchange_cost = 6
        self.disassemble_cost = 5
        self.product_return = 56
        self.part_1_bad_ratio = part_1_bad_ratio
        self.part_2_bad_ratio =  part_2_bad_ratio
        self.product_bad_ratio =  product_bad_ratio
    # 成本首先是两个零件的购买单价
        self.total_cost = self.part_1_cost+self.part_2_cost
        self.part_1 = part_1
        self.part_2 = part_2
        self.product = None
        self.reassemble_time = reassemble_time

    # question.2_step.1
    def given_two_parts_get_one_product(self,total_cost,part_1 = None,part_2
= None):
        if part_1 == None:
            part_1 = np.random.binomial(1, (1 - self.part_1_bad_ratio),
1)[0]
            self.part_1 = part_1
        if part_2 == None:
            part_2 = np.random.binomial(1, (1 - self.part_2_bad_ratio),
1)[0]
            self.part_2 = part_2
        # 如果对零件一做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率
出现次品
        if self.part_1_test == True:
            #做检测的话将确保 part_1 是良品
            while part_1 != 1:
                total_cost += self.part_1_cost
                part_1 = np.random.binomial(1, (1 - self.part_1_bad_ratio),
1)[0]
                self.part_1 = part_1

        # 如果对零件二做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率
出现次品
        if self.part_2_test == True:
            # 做检测的话将确保 part_2 是良品
            while part_2 != 1:
                total_cost += self.part_2_cost
                part_2 = np.random.binomial(1, (1 - self.part_2_bad_ratio),
1)[0]
                self.part_2 = part_2

        # 只有零件一和零件二都是良品，才能保证成品是良品
        if self.part_1 == 1 & self.part_2 == 1:
            self.product = np.random.binomial(1, (1-self.product_bad_ratio),
1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product[0]]
            return return_list
        elif part_1 == 0 or part_2 == 0:
            self.product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product]
            return return_list



    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.product_test == True:
            total_cost += self.product_test_cost
            if product == 0:
                return [’defective’,total_cost]
            else:
                return [’good’,total_cost]
        else:
            return [’unknown’,total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return [’disassembled’,total_cost]
        else:
            return [’abandoned’,total_cost]

result = None
end_cost = 0
```

```
# while (result == ’sold’ or result == ’abandoned’):
def start(part_1_test = None, part_2_test = None, product_test  = None,
bad_product_disassemble = None,part_1 = None,part_2 = None,reassemble_time =
0,
          part_1_bad_ratio = 0.1,part_2_bad_ratio = 0.1,product_bad_ratio =
0.1):
    environment = one_circle_in_manufacturing(part_1_test, part_2_test,
product_test, bad_product_disassemble,

part_1,part_2,reassemble_time,part_1_bad_ratio,part_2_bad_ratio,product_bad_
ratio)
    total_cost = environment.total_cost

    #进行零件检查到组装过程
    after_step_assemble =
environment.given_two_parts_get_one_product(total_cost,part_1,part_2)
    # if (environment.part_1 == 0 or environment.part_2 == 0 or
environment.product == 0):
    #     print(″捕捉到异常″)
    total_cost = after_step_assemble[0]
    curr_product = after_step_assemble[1]

    #对产品装配进行检查
    after_step_product_test =
environment.product_test_procedure(total_cost,curr_product)
    product_test_result = after_step_product_test[0]
    total_cost = after_step_product_test[1]

    if product_test_result == ″good″:
        result = ’sold’
        end_cost = total_cost - environment.product_return
        return end_cost, result
    elif product_test_result ==  ’defective’:
        after_defective_procedure =
environment.defective_procedure(total_cost)
        dealing = after_defective_procedure[0]
        total_cost = after_defective_procedure[1]
        if dealing == ’disassembled’:
            end_cost,result =
start(environment.part_1,environment.part_2,environment.reassemble_time)
            end_cost = total_cost + end_cost
            return end_cost,result

        elif dealing == ’abandoned’:
            end_cost = total_cost - 0
            result = ″abandoned″
            return end_cost,result
    elif product_test_result == ’unknown’:
        if environment.product == 0:
            total_cost = total_cost + environment.exchange_cost
            re_defective_procedure =
environment.defective_procedure(total_cost)
            dealing = re_defective_procedure[0]
            total_cost = re_defective_procedure[1]

            if dealing == ’disassembled’:
                result = None
                while result != ’sold’:
                    end_cost,result = start(environment.part_1,
environment.part_2,environment.reassemble_time)
                    total_cost += end_cost
                if environment.reassemble_time == 0:
                    end_cost = total_cost
                else:
                    end_cost = total_cost
                return end_cost,result
            elif dealing == ’abandoned’:
                end_cost = total_cost - 0
                result = ’abandoned’
                return end_cost,result
        elif environment.product == 1:
            end_cost = total_cost - environment.product_return
            result = ’sold’
            return end_cost,result

# trial_list = []
# for i in range(0,10000):
#     one_trial = start()[0]
#     trial_list.append(one_trial)
# print(trial_list,″\nmean=″,sum(trial_list)/10000)

#setting = part_1_test , part_2_test , product_test ,
bad_product_disassemble
#if bad_product_disassemble = True,both part_1_test and part_2_test should
be True

print(″with global settings:(situation 6)″)
print(’total_cost = 0 \n’,
      'part_1_cost = 4 \n',
      'part_2_cost = 18 \n',
      'assemble_cost = 6 \n',
      'part_1_test_cost = 2 \n',
      'part_2_test_cost = 3 \n',
      'product_test_cost = 3 \n',
      'exchange_cost = 10 \n',
      'disassemble_cost = 40 \n',
      'product_return = 56 \n',
      'part_1_bad_ratio = 0.05 \n',
      'part_2_bad_ratio = 0.05 \n',
      'product_bad_ratio = 0.05 \n')
print('decisions setting:')
print('part_1_test,part_2_test,product_test,bad_product_disassemble\n')



# cnt = 0
# for decision_1 in [True,False]:
#     for decision_2 in [True,False]:
#         for decision_3 in [True,False]:
#             for decision_4 in [True,False]:
#                 if (decision_4 == True) and ((decision_1 and decision_2)
== False):
#                     continue
#                 cnt += 1
#                 setting_list.append("setting_%s"%(cnt))
#                 trial_list = []
#
#                 # 给定随机的次品率
#                 part_1_bad_ratio = np.random.uniform(0.05,0.15,1)[0]
#                 part_2_bad_ratio = np.random.uniform(0.05, 0.15, 1)[0]
#                 product_bad_ratio = np.random.uniform(0.05, 0.15, 1)[0]
#                 for i in range(0, 10000):
#                     one_trial = start(part_1_test = decision_1,
part_2_test = decision_2 , product_test = decision_3,
bad_product_disassemble = decision_4,
#                                       part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio = part_2_bad_ratio,product_bad_ratio =
product_bad_ratio)[0]
#                     trial_list.append(one_trial)
#
#                 print("%0-11s %0-11s %0-12s %0-
23s"%(decision_1,decision_2,decision_3,decision_4),"mean_return=", -
np.mean(trial_list))
```

```
#                 mean_list.append(-np.mean(trial_list))
#                 var_list.append((np.var(trial_list,ddof=1)))
# plt.barh(setting_list,mean_list)
# plt.show()

epsilon_list = []
score_list_1 = []
score_list_2 = []
score_list_3 = []
score_list_total = []
for epsilon in np.linspace(0.01,0.1,20):
    mean_list_1 = []
    mean_list_2 = []
    mean_list_3 = []
    mean_list_total=[]
    standard_bad_ratio = 0.1
    #控制信度
    alpha = 0.05
    Z = norm.isf(q = alpha/2)



    #指定300次不同的次品率组合
    for i in range(0,500):
        trial_list_1 = []
        trial_list_2 = []
        trial_list_3 = []
        total_trial_list = []
        part_1_bad_ratio =
max(standard_bad_ratio,np.random.normal(standard_bad_ratio,epsilon/Z,1)[0])
        part_2_bad_ratio =
max(2*standard_bad_ratio,np.random.normal(2*standard_bad_ratio,epsilon/Z,
1)[0])
        product_bad_ratio =
max(standard_bad_ratio,np.random.uniform(standard_bad_ratio, epsilon/Z,
1)[0])

        # 进行重复模拟
        for i in range(0, 500):
            one_trial = start(part_1_test = True, part_2_test = True ,
product_test = False, bad_product_disassemble = True,
                              part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio = part_2_bad_ratio,product_bad_ratio =
product_bad_ratio)[0]
            second_trial = start(part_1_test = True, part_2_test = True ,
product_test = False, bad_product_disassemble = False,
                              part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio = part_2_bad_ratio,product_bad_ratio =
product_bad_ratio)[0]
            third_trial = start(part_1_test = True, part_2_test = True ,
product_test = True, bad_product_disassemble = True,
                              part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio = part_2_bad_ratio,product_bad_ratio =
product_bad_ratio)[0]
            total_trial_list.append((one_trial+second_trial+third_trial))

            trial_list_1.append(one_trial)
            trial_list_2.append(second_trial)
            trial_list_3.append(third_trial)
        #将期望与方差存入两个列表，以表示特定策略在不匹配的次品率下的表现
        mean_list_1.append(-np.mean(trial_list_1))
        mean_list_2.append(-np.mean(trial_list_2))
        mean_list_3.append(-np.mean(trial_list_3))
        mean_list_total.append(-np.mean(total_trial_list))


    #print("True True False True 策略下面对未知次品率组合时的收益期望与方差：
")
    randomized_expected_reward_1 = np.mean(mean_list_1)
    randomized_sample_variance_1 = np.var(mean_list_1)
    randomized_expected_reward_2 = np.mean(mean_list_2)
    randomized_sample_variance_2 = np.var(mean_list_2)
    randomized_expected_reward_3 = np.mean(mean_list_3)
    randomized_sample_variance_3 = np.var(mean_list_3)
    randomized_expected_reward_total = np.mean(mean_list_total)
    randomized_sample_variance_total = np.var(mean_list_total)


    #print(randomized_expected_reward)
    #print(randomized_sample_variance)

    #均值方差得分系数
    Lambda = 2.0
    #print("均值-方差得分：",randomized_expected_reward -
Lambda*randomized_sample_variance)
    epsilon_list.append(epsilon)
    score_list_1.append(randomized_expected_reward_1 - Lambda *
randomized_sample_variance_1)
    score_list_2.append(randomized_expected_reward_2 - Lambda *
randomized_sample_variance_2)
    score_list_3.append(randomized_expected_reward_3 - Lambda *
randomized_sample_variance_3)
    score_list_total.append(randomized_expected_reward_total - Lambda *
randomized_sample_variance_total)

plt.figure()

plt.subplot(1,2,1)
plt.title(″Scenario_1″,fontdict={’family’ : ’Times New Roman’, ’size’   :
20})
plt.scatter(x = epsilon_list,y = score_list_1,marker= ’ˆ’,color =
'red’,label = 'origin-first-place’)
plt.plot(epsilon_list,score_list_1,color = 'red’)
plt.scatter(x = epsilon_list,y = score_list_2,marker= ’ˆ’,color =
'blue’,label = 'origin-second-place’)
plt.plot(epsilon_list,score_list_2,color = 'blue’)
plt.scatter(x = epsilon_list,y = score_list_3,marker= ’ˆ’,color =
'green’,label = 'origin-third-place’)
plt.plot(epsilon_list,score_list_3,color = 'green’)
plt.xlabel(″epsilon″,fontdict={’family’ : ’Times New Roman’, ’size’   : 16})
plt.ylabel(″score″,fontdict={’family’ : ’Times New Roman’, ’size’   : 16})
plt.legend()



plt.subplot(1,2,2)
plt.scatter(x = epsilon_list,y = score_list_total,marker= ’ˆ’,color =
'black’,label = 'sum_up’)
plt.plot(epsilon_list,score_list_total,color = 'black’)
plt.xlabel(″epsilon″,fontdict={’family’ : ’Times New Roman’, ’size’   : 16})
plt.ylabel(″score″,fontdict={’family’ : ’Times New Roman’, ’size’   : 16})
plt.title(″alpha = 0.05″,y=1,loc=’right’)
plt.legend()
plt.show()
```

B_4_(3).py(在加入了次品率的扰动后重新对问题三中的生产流程进行模拟)

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

#定义 B_3_1 过程的模拟
class one_circle_in_B_3_1:
    def __init__(self, part_1_test = None, part_2_test = None, part_3_test = 
None,half_product_1_test  = None, bad_product_disassemble = None ,
                 part_1=None,part_2=None,part_3 = None,reassemble_time = 
0,part_1_bad_ratio = 0.1,part_2_bad_ratio = 0.1,part_3_bad_ratio = 
0.1,half_product_1_bad_ratio = 0.1):
        # 参数定义
        self.part_1_test = part_1_test
        self.part_2_test = part_2_test
        self.part_3_test = part_3_test
        self.product_test = half_product_1_test
        self.bad_product_disassemble = bad_product_disassemble
        self.total_cost = 0
        self.part_1_cost = 2
        self.part_2_cost = 8
        self.part_3_cost = 12
        self.assemble_cost = 8
        self.part_1_test_cost = 1
        self.part_2_test_cost = 1
        self.part_3_test_cost = 2
        self.product_test_cost = 4
        self.disassemble_cost = 6
        self.part_1_bad_ratio = part_1_bad_ratio
        self.part_2_bad_ratio = part_2_bad_ratio
        self.part_3_bad_ratio = part_3_bad_ratio
        self.product_bad_ratio = half_product_1_bad_ratio
        self.part_1 = part_1
        self.part_2 = part_2
        self.part_3 = part_3
        self.product = None
        self.reassemble_time = reassemble_time
    # question.2_step.1
    def given_three_parts_get_one_product(self,total_cost,part_1 = 
None,part_2 = None,part_3 = None):
        if part_1 == None:
            part_1 = np.random.binomial(1, (1 - self.part_1_bad_ratio), 
1)[0]
            self.part_1 = part_1
            self.total_cost += self.part_1_cost
        if part_2 == None:
            part_2 = np.random.binomial(1, (1 - self.part_2_bad_ratio), 
1)[0]
            self.part_2 = part_2
            self.total_cost += self.part_2_cost
        if part_3 == None:
            part_3 = np.random.binomial(1, (1 - self.part_3_bad_ratio),
1)[0]
            self.part_3 = part_3
            self.total_cost += self.part_3_cost
        # 如果对零件一做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率
出现次品
        if self.part_1_test == True:
            #做检测的话将确保 part_1 是良品
            while part_1 != 1:
                total_cost += self.part_1_cost
                part_1 = np.random.binomial(1, (1 - self.part_1_bad_ratio),
1)[0]
                self.part_1 = part_1

        # 如果对零件二做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率
出现次品
        if self.part_2_test == True:
            # 做检测的话将确保 part_2 是良品
            while part_2 != 1:
                total_cost += self.part_2_cost
                part_2 = np.random.binomial(1, (1 - self.part_2_bad_ratio),
1)[0]
                self.part_2 = part_2

        # 如果对零件三做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率
出现次品
        if self.part_3_test == True:
            # 做检测的话将确保 part_1 是良品
            while part_3 != 1:
                total_cost += self.part_3_cost
                part_3 = np.random.binomial(1, (1 - self.part_3_bad_ratio),
1)[0]
                self.part_3 = part_3

        total_cost += self.total_cost
        # 只有零件一和零件二都是良品，才能保证成品是良品
        if self.part_1 == 1 & self.part_2 == 1 & self.part_3 == 1:
            self.product = np.random.binomial(1, (1-self.product_bad_ratio),
1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product[0]]
            return return_list
        elif part_1 == 0 or part_2 == 0 or part_3 == 0:
            self.product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product]
            return return_list

    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.product_test == True:
            total_cost += self.product_test_cost
            if product == 0:
                return [’defective’,total_cost]
            else:
                return [’good’,total_cost]
        else:
            return [’unknown’,total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return [’disassembled’,total_cost]
        else:
            return [’abandoned’,total_cost]

#定义B_3_2过程的模拟
class one_circle_in_B_3_2:
    def __init__(self, part_4_test = None, part_5_test = None, part_6_test = 
None,half_product_2_test  = None, bad_product_disassemble = None ,
                 part_4=None,part_5=None,part_6 = None,reassemble_time = 
0,part_4_bad_ratio = 0.1,part_5_bad_ratio = 0.1,part_6_bad_ratio = 
0.1,half_product_2_bad_ratio = 0.1):
        # 参数定义
        self.part_4_test = part_4_test
        self.part_5_test = part_5_test
        self.part_6_test = part_6_test
        self.product_test = half_product_2_test
        self.bad_product_disassemble = bad_product_disassemble
        self.total_cost = 0
        self.part_4_cost = 2
        self.part_5_cost = 8
        self.part_6_cost = 12
        self.assemble_cost = 8
        self.part_4_test_cost = 1
        self.part_5_test_cost = 1
        self.part_6_test_cost = 2
        self.product_test_cost = 4
        self.disassemble_cost = 6
        self.part_4_bad_ratio = part_4_bad_ratio
        self.part_5_bad_ratio = part_5_bad_ratio
        self.part_6_bad_ratio = part_6_bad_ratio
        self.product_bad_ratio = half_product_2_bad_ratio
        self.part_4 = part_4
        self.part_5 = part_5
        self.part_6 = part_6
        self.product = None
        self.reassemble_time = reassemble_time
    # question.2_step.1
    def given_three_parts_get_one_product(self,total_cost,part_4 =
None,part_5 = None,part_6 = None):
        if part_4 == None:
            part_4 = np.random.binomial(1, (1 - self.part_4_bad_ratio),
1)[0]
            self.part_4 = part_4
            self.total_cost += self.part_4_cost
        if part_5 == None:
            part_5 = np.random.binomial(1, (1 - self.part_5_bad_ratio),
1)[0]
            self.part_5 = part_5
            self.total_cost += self.part_5_cost
        if part_6 == None:
            part_6 = np.random.binomial(1, (1 - self.part_6_bad_ratio),
1)[0]
            self.part_6 = part_6
            self.total_cost += self.part_6_cost
        # 如果对零件一做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率
出现次品
        if self.part_4_test == True:
            #做检测的话将确保 part_1 是良品
            while part_4 != 1:
                total_cost += self.part_4_cost
                part_4 = np.random.binomial(1, (1 - self.part_4_bad_ratio),
1)[0]
                self.part_4 = part_4

        # 如果对零件二做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率
出现次品
        if self.part_5_test == True:
            # 做检测的话将确保 part_2 是良品
            while part_5 != 1:
                total_cost += self.part_5_cost
                part_5 = np.random.binomial(1, (1 - self.part_5_bad_ratio),
1)[0]
                self.part_5 = part_5

        # 如果对零件三做检测，那么能保证良品，但要增加成本，否则以 0.1 的概率
出现次品
        if self.part_6_test == True:
            # 做检测的话将确保 part_1 是良品
            while part_6 != 1:
                total_cost += self.part_6_cost
                part_6 = np.random.binomial(1, (1 - self.part_6_bad_ratio),
1)[0]
                self.part_6 = part_6

        total_cost += self.total_cost
        # 只有零件一和零件二都是良品，才能保证成品是良品
        if self.part_4 == 1 & self.part_5 == 1 & self.part_6 == 1:
            self.product = np.random.binomial(1, (1-self.product_bad_ratio),
1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product[0]]
            return return_list
        elif part_4 == 0 or part_5 == 0 or part_6 == 0:
            self.product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product]
            return return_list

    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.product_test == True:
            total_cost += self.product_test_cost
            if product == 0:
                return ['defective',total_cost]
            else:
                return ['good',total_cost]
        else:
            return ['unknown',total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return [’disassembled’,total_cost]
        else:
            return [’abandoned’,total_cost]



#定义B_3_3过程的模拟
class one_circle_in_B_3_3:
    def __init__(self, part_7_test = None, part_8_test =
None,half_product_3_test  = None, bad_product_disassemble = None ,
                 part_7=None,part_8=None,reassemble_time =
0,part_7_bad_ratio = 0.1,part_8_bad_ratio = 0.1,half_product_3_bad_ratio =
0.1):
        # 参数定义
        self.part_7_test = part_7_test
        self.part_8_test = part_8_test
        self.product_test = half_product_3_test
        self.bad_product_disassemble = bad_product_disassemble
        self.total_cost = 0
        self.part_7_cost = 8
        self.part_8_cost = 12
        self.assemble_cost = 8
        self.part_7_test_cost = 1
        self.part_8_test_cost = 2
        self.product_test_cost = 4
        self.disassemble_cost = 6
        self.part_7_bad_ratio = part_7_bad_ratio
        self.part_8_bad_ratio = part_8_bad_ratio
        self.product_bad_ratio = half_product_3_bad_ratio
    # 成本首先是两个零件的购买单价
        self.total_cost = 0
        self.part_7 = part_7
        self.part_8 = part_8
        self.product = None
        self.reassemble_time = reassemble_time
    # question.2_step.1
    def given_three_parts_get_one_product(self,total_cost,part_7 =
None,part_8 = None):
        if part_7 == None:
            part_7 = np.random.binomial(1, (1 - self.part_7_bad_ratio),
1)[0]
            self.part_7 = part_7
            self.total_cost += self.part_7_cost
        if part_8 == None:
            part_8 = np.random.binomial(1, (1 - self.part_8_bad_ratio),
1)[0]
            self.part_8 = part_8
            self.total_cost += self.part_8_cost

        # 如果对零件七做检测，那么能保证良品，但要增加成本，否则以0.1的概率
出现次品
        if self.part_7_test == True:
            #做检测的话将确保part_1是良品
            while part_7 != 1:
                total_cost += self.part_7_cost
                part_7 = np.random.binomial(1, (1 - self.part_7_bad_ratio),
1)[0]
                self.part_7 = part_7

        # 如果对零件二做检测，那么能保证良品，但要增加成本，否则以0.1的概率
出现次品
        if self.part_8_test == True:
            # 做检测的话将确保part_2是良品
            while part_8 != 1:
                total_cost += self.part_8_cost
                part_8 = np.random.binomial(1, (1 - self.part_8_bad_ratio),
1)[0]
                self.part_8 = part_8

        total_cost += self.total_cost
        # 只有零件一和零件二都是良品，才能保证成品是良品
        if self.part_7 == 1 & self.part_8 == 1:
            self.product = np.random.binomial(1, (1-self.product_bad_ratio),
1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product[0]]
            return return_list
        elif part_7 == 0 or part_8 == 0:
            self.product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.product]
            return return_list

    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.product_test == True:
            total_cost += self.product_test_cost
            if product == 0:
                return [’defective’,total_cost]
            else:
                return [’good’,total_cost]
        else:
            return [’unknown’,total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return [’disassembled’,total_cost]
        else:
            return [’abandoned’,total_cost]


#定义 B_3_4 过程(已经去掉半成品的生成和检测过程)
class one_circle_in_B_3_4:
    def __init__(self, half_product_1 = None,half_product_2 =
None,half_product_3 = None,final_product_test  = None,
bad_product_disassemble = None ,reassemble_time = 0,
                 final_product_bad_ratio = 0.1):
        # 参数定义
        self.half_product_1 = half_product_1
        self.half_product_2 = half_product_2
        self.half_product_3 = half_product_3
        self.final_product_test = final_product_test
        self.bad_product_disassemble = bad_product_disassemble
        self.total_cost = 0
        self.assemble_cost = 8
        self.final_product_test_cost = 6
        self.disassemble_cost = 10
        self.final_product_bad_ratio = final_product_bad_ratio
        self.final_product_reward = 200
        self.exchange_cost = 40

    # 成本首先是两个零件的购买单价

        self.final_product = None
        self.reassemble_time = reassemble_time
    # question.2_step.1
    def given_three_parts_get_one_product(self,total_cost):
        # 只有半成品一二三都是良品，才能保证成品是良品
        if self.half_product_1 == 1 & self.half_product_2 == 1 &
self.half_product_3 == 1:
            self.final_product = np.random.binomial(1, (1-
self.final_product_bad_ratio), 1)
            total_cost += self.assemble_cost
            return_list = [total_cost, self.final_product[0]]
            return return_list
        elif self.half_product_1 == 0 or self.half_product_2 == 0 or
self.half_product_3 == 0:
            self.final_product = 0
            total_cost += self.assemble_cost
            return_list = [total_cost, self.final_product]
            return return_list

    #定义产品检测过程函数，返回后续处理和当前总成本构成的数组
    def product_test_procedure(self,total_cost,product):
        if self.final_product_test == True:
            total_cost += self.final_product_test_cost
            if product == 0:
                return [’defective’,total_cost]
            else:
                return [’good’,total_cost]
        else:
            return [’unknown’,total_cost]

    #定义对不合格品的拆解过程函数
    def defective_procedure(self,total_cost):
        if self.bad_product_disassemble == True:
            total_cost += self.disassemble_cost
            self.reassemble_time += 1
            return [’disassembled’,total_cost]
        else:
            return [’abandoned’,total_cost]



def start_B_3_1(part_1_test = True, part_2_test = True,part_3_test =
True,product_test  = True, bad_product_disassemble = True,part_1 =
None,part_2 = None,part_3 = None,reassemble_time = 0,
                part_1_bad_ratio = 0.1,part_2_bad_ratio =
0.1,part_3_bad_ratio = 0.1,half_product_1_bad_ratio = 0.1):
    environment = one_circle_in_B_3_1(part_1_test, part_2_test,
part_3_test,product_test,
bad_product_disassemble,part_1,part_2,part_3,reassemble_time,
                                      part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio = part_2_bad_ratio,part_3_bad_ratio =
part_3_bad_ratio,half_product_1_bad_ratio = half_product_1_bad_ratio)
    total_cost = environment.total_cost

    #进行零件检查到组装过程
    after_step_assemble =
environment.given_three_parts_get_one_product(total_cost,part_1,part_2,part_
3)
    # if (environment.part_1 == 0 or environment.part_2 == 0 or
environment.part_3 == 0 or environment.product == 0):
    #     print("捕捉到异常")
    total_cost = after_step_assemble[0]
    curr_half_product = after_step_assemble[1]

    #对产品装配进行检查
    after_step_product_test =
environment.product_test_procedure(total_cost,curr_half_product)
    product_test_result = after_step_product_test[0]
    total_cost = after_step_product_test[1]

    if product_test_result == "good":
        result = 'half_product_1_OK'
        end_cost = total_cost
        return end_cost,
result,curr_half_product,environment.part_1,environment.part_2,environment.p
art_3
    elif product_test_result ==  'defective':
        after_defective_procedure =
environment.defective_procedure(total_cost)
        dealing = after_defective_procedure[0]
        total_cost = after_defective_procedure[1]
        if dealing == 'disassembled':
            end_cost,result,curr_half_product,_,_,_ =
start_B_3_1(part_1=environment.part_1,part_2=environment.part_2,part_3=envir
onment.part_3,reassemble_time = environment.reassemble_time)
            end_cost = total_cost + end_cost
            return
end_cost,result,curr_half_product,environment.part_1,environment.part_2,envi
ronment.part_3

        elif dealing == 'abandoned':
            end_cost = total_cost - 0
            result = "half_product_1_abandoned"
            return
```

```
end_cost,result,None,environment.part_1,environment.part_2,environment.part_
3
    elif product_test_result == 'unknown’:
        end_cost = total_cost
        result = "half_product_1_unknown"
        return
end_cost,result,curr_half_product,environment.part_1,environment.part_2,envi
ronment.part_3




def start_B_3_2(part_4_test = True, part_5_test = True,part_6_test =
True,product_test  = True, bad_product_disassemble = True,part_4 =
None,part_5 = None,part_6 = None,reassemble_time = 0,
                part_4_bad_ratio = 0.1,part_5_bad_ratio =
0.1,part_6_bad_ratio = 0.1,half_product_2_bad_ratio = 0.1):
    environment = one_circle_in_B_3_2(part_4_test, part_5_test,
part_6_test,product_test,
bad_product_disassemble,part_4,part_5,part_6,reassemble_time,
                                      part_4_bad_ratio =
part_4_bad_ratio,part_5_bad_ratio = part_5_bad_ratio,part_6_bad_ratio =
part_6_bad_ratio,half_product_2_bad_ratio = half_product_2_bad_ratio)
    total_cost = environment.total_cost

    #进行零件检查到组装过程
    after_step_assemble =
environment.given_three_parts_get_one_product(total_cost,part_4,part_5,part_
6)
    # if (environment.part_4 == 0 or environment.part_5 == 0 or
environment.part_6 == 0 or environment.product == 0):
    #     print("捕捉到异常")
    total_cost = after_step_assemble[0]
    curr_half_product = after_step_assemble[1]

    #对产品装配进行检查
    after_step_product_test =
environment.product_test_procedure(total_cost,curr_half_product)
    product_test_result = after_step_product_test[0]
    total_cost = after_step_product_test[1]

    if product_test_result == "good":
        result = 'half_product_2_OK’
        end_cost = total_cost
        return end_cost,
result,curr_half_product,environment.part_4,environment.part_5,environment.p
art_6
    elif product_test_result ==  ’defective’:
        after_defective_procedure =
environment.defective_procedure(total_cost)
        dealing = after_defective_procedure[0]
        total_cost = after_defective_procedure[1]
        if dealing == ’disassembled’:
            end_cost,result,curr_half_product,_,_,_ = start_B_3_2(part_4 =
environment.part_4, part_5 = environment.part_5, part_6 =
environment.part_6, reassemble_time = environment.reassemble_time)
            end_cost = total_cost + end_cost
            return
end_cost,result,curr_half_product,environment.part_4,environment.part_5,envi
ronment.part_6

        elif dealing == ’abandoned’:
            end_cost = total_cost - 0
            result = ″half_product_2_abandoned″
            return
end_cost,result,None,environment.part_4,environment.part_5,environment.part_
6
    elif product_test_result == ’unknown’:
        end_cost = total_cost
        result = ″half_product_2_unknown″
        return
end_cost,result,curr_half_product,environment.part_4,environment.part_5,envi
ronment.part_6




def start_B_3_3(part_7_test = True, part_8_test = True,product_test  = True,
bad_product_disassemble = True,part_7 = None,part_8 = None,reassemble_time =
0,
                part_7_bad_ratio = 0.1,part_8_bad_ratio =
0.1,half_product_3_bad_ratio = 0.1):
    environment = one_circle_in_B_3_3(part_7_test, part_8_test,product_test,
bad_product_disassemble,part_7,part_8,reassemble_time,
                                      part_7_bad_ratio =
part_7_bad_ratio,part_8_bad_ratio =
part_8_bad_ratio,half_product_3_bad_ratio = half_product_3_bad_ratio)
    total_cost = environment.total_cost

    #进行零件检查到组装过程
    after_step_assemble =
environment.given_three_parts_get_one_product(total_cost,part_7,part_8)
    # if (environment.part_7 == 0 or environment.part_8 == 0 or
environment.product == 0):
    #     print(“捕捉到异常”)
    total_cost = after_step_assemble[0]
    curr_half_product = after_step_assemble[1]

    #对产品装配进行检查
    after_step_product_test =
environment.product_test_procedure(total_cost,curr_half_product)
    product_test_result = after_step_product_test[0]
    total_cost = after_step_product_test[1]

    if product_test_result == “good”:
        result = ‘half_product_3_OK’
        end_cost = total_cost
        return end_cost,
result,curr_half_product,environment.part_7,environment.part_8
    elif product_test_result == ‘defective’:
        after_defective_procedure =
environment.defective_procedure(total_cost)
        dealing = after_defective_procedure[0]
        total_cost = after_defective_procedure[1]
        if dealing == ‘disassembled’:
            end_cost,result,curr_half_product,_,_ = start_B_3_3(part_7 =
environment.part_7, part_8 = environment.part_8, reassemble_time =
environment.reassemble_time)
            end_cost = total_cost + end_cost
            return
end_cost,result,curr_half_product,environment.part_7,environment.part_8

        elif dealing == ‘abandoned’:
            end_cost = total_cost - 0
            result = “half_product_3_abandoned”
            return
end_cost,result,None,environment.part_7,environment.part_8
    elif product_test_result == ‘unknown’:
        end_cost = total_cost
        result = “half_product_3_unknown”
        return
end_cost,result,curr_half_product,environment.part_7,environment.part_8
```

```
def
startB_3_4(part_1_test=True,part_2_test=True,part_3_test=True,part_4_test=Tr
ue,part_5_test=True,part_6_test=True,part_7_test=True,part_8_test=True,

half_product_1_test=True,half_product_2_test=True,half_product_3_test=True,f
inal_product_test=True,half_product_1_disassemble=True,half_product_2_disass
emble=True,
               half_product_3_disassemble=True, bad_product_disassemble =
True ,reassemble_time = 0,part_1 = None,
               part_2 = None,part_3 = None,part_4 = None,part_5 =
None,part_6 = None,part_7 = None,part_8 = None,part_9 = None,
               part_1_bad_ratio = 0.1,part_2_bad_ratio =
0.1,part_3_bad_ratio=0.1,half_product_1_bad_ratio = 0.1,
               part_4_bad_ratio = 0.1,part_5_bad_ratio =
0.1,part_6_bad_ratio=0.1,half_product_2_bad_ratio = 0.1,
               part_7_bad_ratio = 0.1,part_8_bad_ratio =
0.1,half_product_3_bad_ratio = 0.1,final_product_bad_ratio = 0.1):
    end_cost_1,result_1,half_product_1,part_1,part_2,part_3 =
start_B_3_1(part_1_test = part_1_test, part_2_test = part_2_test,part_3_test
= part_3_test,product_test  = half_product_1_test, bad_product_disassemble =
half_product_1_disassemble,

part_1 = part_1,part_2=part_2,part_3=part_3,part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio = part_2_bad_ratio,part_3_bad_ratio =
part_3_bad_ratio,half_product_1_bad_ratio = half_product_1_bad_ratio)
    end_cost_2, result_2, half_product_2,part_4,part_5,part_6 =
start_B_3_2(part_4_test = part_4_test, part_5_test = part_5_test,part_6_test
= part_6_test,product_test  = half_product_2_test, bad_product_disassemble =
half_product_2_disassemble,

part_4=part_4,part_5=part_5,part_6=part_6,part_4_bad_ratio =
part_4_bad_ratio,part_5_bad_ratio = part_5_bad_ratio,part_6_bad_ratio =
part_6_bad_ratio,half_product_2_bad_ratio = half_product_2_bad_ratio)
    end_cost_3, result_3, half_product_3,part_7,part_8 =
start_B_3_3(part_7_test = part_7_test, part_8_test =
part_8_test,product_test  = half_product_3_test, bad_product_disassemble =
half_product_3_disassemble,

part_7=part_7,part_8=part_8,part_7_bad_ratio =
part_7_bad_ratio,part_8_bad_ratio =
part_8_bad_ratio,half_product_3_bad_ratio = half_product_3_bad_ratio)

    if half_product_1 == None or half_product_2 == None or half_product_3 ==
None:
        return (end_cost_1+end_cost_2+end_cost_3), ’failed’
    environment_4 =
one_circle_in_B_3_4(half_product_1,half_product_2,half_product_3,final_produ
ct_test,bad_product_disassemble,reassemble_time,final_product_bad_ratio =
final_product_bad_ratio)
    #装配
    assemble_result =
environment_4.given_three_parts_get_one_product(environment_4.total_cost)
    curr_total_cost = assemble_result[0] + end_cost_1 + end_cost_2 +
end_cost_3
    curr_product = assemble_result[1]

    # if (curr_product == 0):
    #     print(”出现次品成品！”)

    #最终成品检验
    final_product_test =
environment_4.product_test_procedure(curr_total_cost,curr_product)
    test_result = final_product_test[0]
    curr_total_cost = final_product_test[1]

    if test_result == ’good’:
        end_cost = curr_total_cost - environment_4.final_product_reward
        result = ’sold’
        return end_cost,result
    elif test_result == ’defective’:
        defective_dealing =
environment_4.defective_procedure(curr_total_cost)
        dealing = defective_dealing[0]
        curr_total_cost = defective_dealing[1]
        if dealing == ’disassembled’:
            if (half_product_1 == 0)or(half_product_2 == 0)or(half_product_3
==0):
                return (curr_total_cost +4+4+4),’abandoned’
            else:
                another_assemble = np.random.binomial(1,0.9,1)[0]
                curr_total_cost += 8 #重新装配成本
                while( another_assemble != 1):
                    curr_total_cost += 10 #再次拆解
                    curr_total_cost += 8 #再次重新装配
                    another_assemble = np.random.binomial(1, 0.9, 1)[0]
                return (curr_total_cost-
environment_4.final_product_reward),’sold’
```

```
        elif dealing == 'abandoned’:
            return curr_total_cost,'abandoned’



    elif test_result == 'unknown’:
        #如果出现了次品到客户手上的情况，那么我们需要无偿为客户替换件，我们
将以最高成本的方式准备一件良品
        if environment_4.final_product == 0:
            curr_total_cost += environment_4.exchange_cost
            re_defective_procedure =
environment_4.defective_procedure(curr_total_cost)
            dealing = re_defective_procedure[0]
            total_cost = re_defective_procedure[1]

            if dealing == 'disassembled’:
                return (total_cost - environment_4.final_product_reward + 12
+ 8 + 6),'problem’
            elif dealing == 'abandoned’:
                return (total_cost -environment_4.final_product_reward + 64
+ 11 + 24 + 12 + 8 + 6),'problem’

        elif environment_4.final_product == 1:
            end_cost = curr_total_cost - environment_4.final_product_reward
            result = 'sold’
            return end_cost, result




epsilon_list = []
score_list_1 = []
score_list_2 = []
score_list_3 = []
score_list_4 = []
score_list_5 = []
score_list_total = []
for epsilon in np.linspace(0.01,0.1,20):
    mean_list_1 = []
    mean_list_2 = []
    mean_list_3 = []
    mean_list_4 = []
    mean_list_5 = []
    mean_list_total=[]
    standard_bad_ratio = 0.1
    #控制信度
    alpha = 0.05
    Z = norm.isf(q = alpha/2)


    #指定 300 次不同的次品率组合
    for i in range(0,300):
        trial_list_1 = []
        trial_list_2 = []
        trial_list_3 = []
        trial_list_4 = []
        trial_list_5 = []
        total_trial_list = []

        part_1_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        part_2_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        part_3_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        part_4_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        part_5_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        part_6_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        part_7_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        part_8_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        half_product_1_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        half_product_2_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        half_product_3_bad_ratio = max(standard_bad_ratio,
np.random.normal(standard_bad_ratio, epsilon / Z, 1)[0])
        final_product_bad_ratio =
max(standard_bad_ratio,np.random.normal(standard_bad_ratio, epsilon/Z,
1)[0])

        # 进行重复模拟
        for i in range(0, 300):
            one_trial =
startB_3_4(part_1_test=True,part_2_test=True,part_3_test=True,part_4_test=Tr
ue,part_5_test=True,part_6_test=True,part_7_test=True,part_8_test=True,
half_product_1_test=False,half_product_2_test=False,half_product_3_test=Fals
e,final_product_test=False,half_product_1_disassemble=True,

half_product_2_disassemble=False,half_product_3_disassemble=False,
bad_product_disassemble = True ,reassemble_time = 0,part_1 = None,
                                   part_2 = None,part_3 = None,part_4 =
None,part_5 = None,part_6 = None,part_7 = None,part_8 = None,part_9 = None,
                                   part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio =
part_2_bad_ratio,part_3_bad_ratio=part_3_bad_ratio,half_product_1_bad_ratio
= half_product_1_bad_ratio,
                                   part_4_bad_ratio =
part_4_bad_ratio,part_5_bad_ratio =
part_5_bad_ratio,part_6_bad_ratio=part_6_bad_ratio,half_product_2_bad_ratio
= half_product_2_bad_ratio,
                                   part_7_bad_ratio =
part_7_bad_ratio,part_8_bad_ratio =
part_8_bad_ratio,half_product_3_bad_ratio =
half_product_3_bad_ratio,final_product_bad_ratio =
final_product_bad_ratio)[0]

            second_trial =
startB_3_4(part_1_test=True,part_2_test=True,part_3_test=True,part_4_test=Tr
ue,part_5_test=True,part_6_test=True,part_7_test=True,part_8_test=True,

half_product_1_test=False,half_product_2_test=False,half_product_3_test=Fals
e,final_product_test=False,half_product_1_disassemble=False,

half_product_2_disassemble=False,half_product_3_disassemble=False,
bad_product_disassemble = True ,reassemble_time = 0,part_1 = None,
                                   part_2 = None,part_3 = None,part_4 =
None,part_5 = None,part_6 = None,part_7 = None,part_8 = None,part_9 = None,
                                   part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio =
part_2_bad_ratio,part_3_bad_ratio=part_3_bad_ratio,half_product_1_bad_ratio
= half_product_1_bad_ratio,
                                   part_4_bad_ratio =
part_4_bad_ratio,part_5_bad_ratio =
part_5_bad_ratio,part_6_bad_ratio=part_6_bad_ratio,half_product_2_bad_ratio
= half_product_2_bad_ratio,
                                   part_7_bad_ratio =
part_7_bad_ratio,part_8_bad_ratio =
part_8_bad_ratio,half_product_3_bad_ratio =
half_product_3_bad_ratio,final_product_bad_ratio =
final_product_bad_ratio)[0]

            third_trial =
startB_3_4(part_1_test=True,part_2_test=True,part_3_test=True,part_4_test=Tr
ue,part_5_test=True,part_6_test=True,part_7_test=True,part_8_test=True,

half_product_1_test=False,half_product_2_test=False,half_product_3_test=Fals
e,final_product_test=False,half_product_1_disassemble=True,

half_product_2_disassemble=False,half_product_3_disassemble=True,
bad_product_disassemble = True ,reassemble_time = 0,part_1 = None,
                                   part_2 = None,part_3 = None,part_4 =
None,part_5 = None,part_6 = None,part_7 = None,part_8 = None,part_9 = None,
                                   part_1_bad_ratio =
part_1_bad_ratio,part_2_bad_ratio =
part_2_bad_ratio,part_3_bad_ratio=part_3_bad_ratio,half_product_1_bad_ratio
= half_product_1_bad_ratio,
                                   part_4_bad_ratio =
part_4_bad_ratio,part_5_bad_ratio =
part_5_bad_ratio,part_6_bad_ratio=part_6_bad_ratio,half_product_2_bad_ratio
= half_product_2_bad_ratio,
                                   part_7_bad_ratio =
part_7_bad_ratio,part_8_bad_ratio =
part_8_bad_ratio,half_product_3_bad_ratio =
half_product_3_bad_ratio,final_product_bad_ratio =
final_product_bad_ratio)[0]

            fourth_trial = startB_3_4(part_1_test=True, part_2_test=True,
part_3_test=True, part_4_test=True, part_5_test=True,
                       part_6_test=True, part_7_test=True, part_8_test=True,
                       half_product_1_test=True, half_product_2_test=False,
half_product_3_test=True,
                       final_product_test=False,
half_product_1_disassemble=True,
                       half_product_2_disassemble=True,
half_product_3_disassemble=True, bad_product_disassemble=True,
                       reassemble_time=0, part_1=None,
                       part_2=None, part_3=None, part_4=None, part_5=None,
part_6=None, part_7=None, part_8=None,
                       part_9=None,
                       part_1_bad_ratio=part_1_bad_ratio,
part_2_bad_ratio=part_2_bad_ratio,
                       part_3_bad_ratio=part_3_bad_ratio,
half_product_1_bad_ratio=half_product_1_bad_ratio,
                        part_4_bad_ratio=part_4_bad_ratio,
part_5_bad_ratio=part_5_bad_ratio,
                        part_6_bad_ratio=part_6_bad_ratio,
half_product_2_bad_ratio=half_product_2_bad_ratio,
                        part_7_bad_ratio=part_7_bad_ratio,
part_8_bad_ratio=part_8_bad_ratio,
                        half_product_3_bad_ratio=half_product_3_bad_ratio,
                        final_product_bad_ratio=final_product_bad_ratio)[0]

            fifth_trial = startB_3_4(part_1_test=True, part_2_test=True,
part_3_test=False, part_4_test=True, part_5_test=True,
                        part_6_test=True, part_7_test=True, part_8_test=True,
                        half_product_1_test=False, half_product_2_test=True,
half_product_3_test=False,
                        final_product_test=False,
half_product_1_disassemble=False,
                        half_product_2_disassemble=True,
half_product_3_disassemble=True, bad_product_disassemble=True,
                        reassemble_time=0, part_1=None,
                        part_2=None, part_3=None, part_4=None, part_5=None,
part_6=None, part_7=None, part_8=None,
                        part_9=None,
                        part_1_bad_ratio=part_1_bad_ratio,
part_2_bad_ratio=part_2_bad_ratio,
                        part_3_bad_ratio=part_3_bad_ratio,
half_product_1_bad_ratio=half_product_1_bad_ratio,
                        part_4_bad_ratio=part_4_bad_ratio,
part_5_bad_ratio=part_5_bad_ratio,
                        part_6_bad_ratio=part_6_bad_ratio,
half_product_2_bad_ratio=half_product_2_bad_ratio,
                        part_7_bad_ratio=part_7_bad_ratio,
part_8_bad_ratio=part_8_bad_ratio,
                        half_product_3_bad_ratio=half_product_3_bad_ratio,
                        final_product_bad_ratio=final_product_bad_ratio)[0]

total_trial_list.append((one_trial+second_trial+third_trial+fourth_trial+fif
th_trial))

            trial_list_1.append(one_trial)
            trial_list_2.append(second_trial)
            trial_list_3.append(third_trial)
            trial_list_4.append(fourth_trial)
            trial_list_5.append(fifth_trial)
```

```
        #将期望与方差存入两个列表，以表示特定策略在不匹配的次品率下的表现
        mean_list_1.append(-np.mean(trial_list_1))
        mean_list_2.append(-np.mean(trial_list_2))
        mean_list_3.append(-np.mean(trial_list_3))
        mean_list_4.append(-np.mean(trial_list_4))
        mean_list_5.append(-np.mean(trial_list_5))
        mean_list_total.append(-np.mean(total_trial_list))



    #print("True True False True 策略下面对未知次品率组合时的收益期望与方差：
")
    randomized_expected_reward_1 = np.mean(mean_list_1)
    randomized_sample_variance_1 = np.var(mean_list_1)
    randomized_expected_reward_2 = np.mean(mean_list_2)
    randomized_sample_variance_2 = np.var(mean_list_2)
    randomized_expected_reward_3 = np.mean(mean_list_3)
    randomized_sample_variance_3 = np.var(mean_list_3)
    randomized_expected_reward_4 = np.mean(mean_list_4)
    randomized_sample_variance_4 = np.var(mean_list_4)
    randomized_expected_reward_5 = np.mean(mean_list_5)
    randomized_sample_variance_5 = np.var(mean_list_5)
    randomized_expected_reward_total = np.mean(mean_list_total)
    randomized_sample_variance_total = np.var(mean_list_total)


    #print(randomized_expected_reward)
    #print(randomized_sample_variance)

    #均值方差得分系数
    Lambda = 2
    #print("均值-方差得分：",randomized_expected_reward -
Lambda*randomized_sample_variance)
    epsilon_list.append(epsilon)
    score_list_1.append(randomized_expected_reward_1 - Lambda *
randomized_sample_variance_1)
    score_list_2.append(randomized_expected_reward_2 - Lambda *
randomized_sample_variance_2)
    score_list_3.append(randomized_expected_reward_3 - Lambda *
randomized_sample_variance_3)
    score_list_4.append(randomized_expected_reward_4 - Lambda *
randomized_sample_variance_4)
    score_list_5.append(randomized_expected_reward_5 - Lambda *
randomized_sample_variance_5)
```

```
    score_list_total.append(randomized_expected_reward_total - Lambda *
randomized_sample_variance_total)



plt.figure()

plt.subplot(1,2,1)
plt.title(″two-stage-situation″,fontdict={’family’ : ’Times New Roman’,
’size’   : 20})
plt.scatter(x = epsilon_list,y = score_list_1,marker= ’ˆ’,color =
’red’,label = ’origin-1st-place’)
plt.plot(epsilon_list,score_list_1,color = ’red’)
# plt.scatter(x = epsilon_list,y = score_list_2,marker= ’ˆ’,color =
’blue’,label = ’origin-second-place’)
# plt.plot(epsilon_list,score_list_2,color = ’blue’)
# plt.scatter(x = epsilon_list,y = score_list_3,marker= ’ˆ’,color =
’green’,label = ’origin-third-place’)
# plt.plot(epsilon_list,score_list_3,color = ’green’)
plt.scatter(x = epsilon_list,y = score_list_4,marker= ’ˆ’,color =
’grey’,label = ’origin-26th-place’)
plt.plot(epsilon_list,score_list_4,color = ’grey’)
plt.scatter(x = epsilon_list,y = score_list_5,marker= ’ˆ’,color =
’yellow’,label = ’origin-50th-place’)
plt.plot(epsilon_list,score_list_5,color = ’yellow’)
plt.xlabel(″epsilon″,fontdict={’family’ : ’Times New Roman’, ’size’   : 16})
plt.ylabel(″score″,fontdict={’family’ : ’Times New Roman’, ’size’   : 16})
plt.legend()



plt.subplot(1,2,2)
plt.scatter(x = epsilon_list,y = score_list_total,marker= ’ˆ’,color =
’black’,label = ’sum_up’)
plt.plot(epsilon_list,score_list_total,color = ’black’)
plt.xlabel(″epsilon″,fontdict={’family’ : ’Times New Roman’, ’size’   : 16})
plt.ylabel(″score″,fontdict={’family’ : ’Times New Roman’, ’size’   : 16})
plt.title(″alpha = 0.05″,y=1,loc=’right’)
plt.legend()
plt.show()
```

附录 3
Python 模拟结果
Python_situation_1.txt with global settings:(situation 1) total_cost = 0

```
  part_1_cost = 4
  part_2_cost = 18
  assemble_cost = 6
  part_1_test_cost = 2
  part_2_test_cost = 3
  product_test_cost = 3
  exchange_cost = 6
  disassemble_cost = 5
  product_return = 56
  part_1_bad_ratio = 0.1
  part_2_bad_ratio = 0.1
  product_bad_ratio = 0.1

decisions setting:
part_1_test,part_2_test,product_test,bad_product_disassemble

True            True            True            True                            mean_return=
18.4014
True            True            True            False                           mean_return=
17.0026
True            True            False           True                            mean_return=
20.8381
True            True            False           False                           mean_return=
19.546
True            False           True            False                           mean_return=
14.0504
True            False           False           False                           mean_return=
16.0828
False           True            True            False                           mean_return=
12.4014
False           True            False           False                           mean_return=
14.2396
False           False           True            False                     mean_return= 9.7736
False           False           False           False                           mean_return=
10.8694
```

```
Python_situation_2.txt
with global settings:(situation 2)
total_cost = 0
  part_1_cost = 4
  part_2_cost = 18
  assemble_cost = 6
  part_1_test_cost = 2
  part_2_test_cost = 3
  product_test_cost = 3
  exchange_cost = 6
  disassemble_cost = 5
  product_return = 56
  part_1_bad_ratio = 0.2
  part_2_bad_ratio = 0.2
  product_bad_ratio = 0.2

decisions setting:
part_1_test,part_2_test,product_test,bad_product_disassemble

True            True            True             True                                 mean_return=
8.6982
True            True            True             False                                mean_return=
8.5306
True            True            False            True                                 mean_return=
10.8389
True            True            False            False                                mean_return=
9.7106
True            False           True             False                                mean_return=
4.0304
True           False          False          False                            mean_return= 4.3118
False           True            True             False                                mean_return=
0.8442
False          True           False          False                            mean_return= 0.5908
False           False           True             False                              mean_return=  -
2.5464
False           False           False            False                              mean_return=  -
2.4358
```

```
Python_situation_3.txt
with global settings:(situation 3)
total_cost = 0
  part_1_cost = 4
  part_2_cost = 18
  assemble_cost = 6
  part_1_test_cost = 2
  part_2_test_cost = 3
  product_test_cost = 3
  exchange_cost = 30
  disassemble_cost = 5
  product_return = 56
  part_1_bad_ratio = 0.1
  part_2_bad_ratio = 0.1
  product_bad_ratio = 0.1
```

```

decisions setting:
part_1_test,part_2_test,product_test,bad_product_disassemble

True           True           True           True                              mean_return=
18.341
True           True           True           False                             mean_return=
17.1098
True           True           False          True                              mean_return=
18.1364
True           True           False          False                             mean_return=
17.0024
True           False          True           False                             mean_return=
14.0564
True           False          False          False                             mean_return=
11.8664
False         True          True          False                        mean_return= 12.37
False         True          False         False                        mean_return= 9.6484
False         False         True          False                        mean_return= 9.9416
False         False         False         False                        mean_return= 4.9176
```

```
Python_situation_4.txt
with global settings:(situation 4)
total_cost = 0
 part_1_cost = 4
 part_2_cost = 18
 assemble_cost = 6
 part_1_test_cost = 1
 part_2_test_cost = 1
 product_test_cost = 2
 exchange_cost = 30
 disassemble_cost = 5
 product_return = 56
 part_1_bad_ratio = 0.2
 part_2_bad_ratio = 0.2
 product_bad_ratio = 0.2

decisions setting:
part_1_test,part_2_test,product_test,bad_product_disassemble

True           True           True           True                              mean_return=
10.2761
True           True           True           False                             mean_return=
9.349
True           True           False          True                              mean_return=
6.3857
True            True            False           False                           mean_return=
5.5314
True            False           True            False                           mean_return=
4.7252
True            False           False           False                   mean_return= -4.32
False           True            True            False                           mean_return=
0.8368
False           True            False           False                           mean_return=  -
6.9622
False           False           True            False                           mean_return=  -
1.1264
False           False           False           False                           mean_return=  -
13.7444
```

```
Python_situation_5.txt
with global settings:(situation 5)
total_cost = 0
  part_1_cost = 4
  part_2_cost = 18
  assemble_cost = 6
  part_1_test_cost = 8
  part_2_test_cost = 1
  product_test_cost = 2
  exchange_cost = 10
  disassemble_cost = 5
  product_return = 56
  part_1_bad_ratio = 0.1
  part_2_bad_ratio = 0.2
  product_bad_ratio = 0.1

decisions setting:
part_1_test,part_2_test,product_test,bad_product_disassemble

True            True            True            True                            mean_return=
16.5066
True            True            True            False                           mean_return=
15.5346
True            True            False           True                            mean_return=
17.7383
True            True            False           False                           mean_return=
16.6784
True            False           True            False                           mean_return=
10.1964
True            False           False           False                   mean_return= 8.9898
```

```
False         True          True          False                          mean_return=
10.7054
False         True          False         False                          mean_return=
10.4608
False         False         True          False                      mean_return= 6.5064
False         False         False         False                      mean_return= 4.8142
```

```
Python_situation_6.txt
with global settings:(situation 6)
total_cost = 0
 part_1_cost = 4
 part_2_cost = 18
 assemble_cost = 6
 part_1_test_cost = 2
 part_2_test_cost = 3
 product_test_cost = 3
 exchange_cost = 10
 disassemble_cost = 40
 product_return = 56
 part_1_bad_ratio = 0.05
 part_2_bad_ratio = 0.05
 product_bad_ratio = 0.05

decisions setting:
part_1_test,part_2_test,product_test,bad_product_disassemble

True          True          True          True                           mean_return=
20.2924
True          True          True          False                          mean_return=
20.9606
True          True          False         True                           mean_return=
22.6741
True          True          False         False                          mean_return=
23.6904
True          False         True          False                          mean_return=
19.1128
True          False         False         False                          mean_return=
21.1158
False         True          True          False                          mean_return=
18.883
False         True          False         False                      mean_return= 20.467
False         False         True          False                          mean_return=
17.1824
False         False         False         False                          mean_return=
18.8656
```

附录 4

Matlab 目标函数

objective_function_2.m(用以求解问题二模型的目标函数建立)

```
    function z = objective_function_2(d)
    % 输入变量 d = [d1, d2, d3, d4]
    d1 = d(1);
    d2 = d(2);
    d3 = d(3);
    d4 = d(4);
    % 定义参数
    Price = [4, 18];
    q = [2, 3, 6, 3, 10, 40];
    p = [0.05, 0.05, 0.05];
    S = 56;
    N1_k = 10000;
    sum_N1 = 0;
    sum_N2 = 0;
    sum_Cj = 0;
    K_max = 100;

    C1 = N1_k * Price(1) * (1/(1-d(1)*p(1)))+ N1_k * Price(2)*(1/(1-
d(2)*p(2)));
    % 循环求解每一个 k
    for k = 1:K_max
    % 计算 C2
    C2_k = N1_k * (q(1) * d1*(1/(1-d(1)*p(1))) + q(2) * d2*(1/(1-
d(2)*p(2))));
    % 计算 N2(k)的更新值
    N2_k = N1_k ;

    % 计算 C3
    C3_k = N2_k * q(3);
    % 计算 C4
    C4_k = N2_k * q(4) * d3;

    coef = (1/2)*((1-d(1))*p(1)+(1-d(2))*p(2)) + (1/2)*max((1-
d(1))*p(1),(1-d(2))*p(2));
    %best_coef = (max(p(3),max((1-p(3))*((1-d(1))*p(1)),(1-p(3))*((1-
d(2))*p(2)))) + ((1-d(1))*p(1)+(1-d(2))*p(2)))/2;
    %coef = max((1-d(1))*p(1),(1-d(2))*p(2));
    %coef = (1-d(1))*p(1) + (1-d(2))*p(2);
    % 计算 C5
    C5_k = (N2_k * p(3) + N2_k * (1 - p(3)) * ((1 - d1) * p(1) + (1 - d2)
* p(2))) * (1 - d3) * (q(5) + S) ;
    %C5_k = (N2_k * p(3) + N2_k * (1 - p(3)) * (coef)) * (1 - d3) * q(5);

    % 计算 C6
    C6_k = N2_k * (p(3) + (1 - p(3)) * ((1 - d1) * p(1) + (1 - d2) *
p(2))) * d4 * q(6);
    %C6_k = N2_k * (p(3) + (1 - p(3)) * (coef)) * d4 * q(6);

    % 更新 N1
    N1_k_next = N2_k * (p(3) + N2_k * (1 - p(3)) * ((1 - d1) * p(1) + (1
- d2) * p(2))) * d4;
    %N1_k_next = N2_k * (p(3) + N2_k * (1 - p(3)) * (coef)) * d4;
    % 累加到总的 N1 和 C_j
    %sum_N1 = sum_N1 + N1_k;

    %sum_N2 = sum_N2 + (1-d(3)*(p(3)+(1-d(1))*p(1) + (1-
d(2))*p(2)))*N2_k;
    sum_N2 = sum_N2 + (1-d(3)*(p(3)+coef))*N2_k;
    sum_Cj = sum_Cj + (C2_k + C3_k + C4_k + C5_k + C6_k);
    % 更新 N1_k 为下一个迭代的 N1_k_next
    N1_k = N1_k_next;
    end
    % 目标函数 Z
    z = S * sum_N2 - (C1 + sum_Cj) ;
    end
```

objective_function_3.m(用以求解问题三模型的目标函数建立)

```
    function Z = objective_function_3(d)
    d = d';
    N_initial = 1000;
    S = 200;
    iterations_m = 100;
    iterations_k = 100;

    N_vals = zeros(8,iterations_m);
    M_vals = zeros(iterations_k,1);
    M_vals(1) = N_initial;
    for i = 1:8
    N_vals(i,1) = N_initial;
    end

    C_vals_m = zeros(5,iterations_m);
    C_vals_k = zeros(11,iterations_k); %只会用 6-11 行
    Price = [2;8;12;2;8;12;8;12]; %len = 8
    p = [0.1;0.1;0.1;0.1;0.1;0.1;0.1;0.1;0;0;0;0.1;0.1;0.1;0.1]; % len =
15
    q = [1;1;2;1;1;2;1;2;8;8;8;4;4;4;6;6;6;8;6;10;40];% len = 21
    ratio_list = zeros(8,1);
    for i = 1:8
    ratio_list(i) = (1/(1-d(i)*p(i)));
    end
    C_1 = sum(N_vals(1:8,1) .* Price(1:8) .* (ratio_list(1:8)));
    C_vals_m(1,1) = 0; %(3)
    %(9) (10) (11)
    p12_2 = p(12) + (1-p(12))*sum((1-d(1:3)).*p(1:3));
    p13_2 = p(13) + (1-p(13))*sum((1-d(4:6)).*p(4:6));
    p14_2 = p(14) + (1-p(14))*sum((1-d(7:8)).*p(7:8));

    for m = 1:iterations_m
    if m >= 2
    %(4)
    C_vals_m(1,m) = (1-d(9)) * sum(N_vals(1:3,m) .* Price(1:3) .*
ratio_list(1:3)) ...
    +(1-d(10)) * sum(N_vals(4:6,m) .* Price(4:6) .* ratio_list(4:6)) ...
    +(1-d(11)) * sum(N_vals(7:8,m) .* Price(7:8) .* ratio_list(7:8));
    end
    %(5)
    C_vals_m(2,m) = sum(N_vals(1:8,m) .* q(1:8) .* d(1:8) .*
ratio_list(1:8));
    %(6)
    C_vals_m(3,m) = q(9)*N_vals(1,m)' + q(10)*N_vals(4,m)' +
q(11)*N_vals(7,m)';

    %(7)
    C_vals_m(4,m) = q(12)*d(12)*N_vals(1,m)' + q(13) * d(13) *
N_vals(4,m)' + q(14)*d(14)*N_vals(7,m)';

    %(8)
    C_vals_m(5,m) = N_vals(1,m)*d(12)*p12_2*d(9)*q(15) +
N_vals(4,m)*d(13)*p13_2*d(10)*q(16) +
N_vals(7,m)*d(14)*p14_2*d(11)*q(17);

    % (12) (13) (14)
    N_vals(1:3,m+1) = N_vals(1:3,m)'*d(12)*p12_2;
    N_vals(4:6,m+1) = N_vals(4:6,m)'*d(12)*p13_2;
    N_vals(7:8,m+1) = N_vals(7:8,m)'*d(12)*p14_2;
    end

    sum_part = p(15)+(1-p(15))*((1-d(12))*p12_2 + (1-d(13))*p13_2 + (1-
d(14))*p14_2);
    max_part_for_15 = 1 - max((1 - d(12)*p12_2) , max((1 - d(13)*p13_2) ,
(1 - d(14)*p14_2)));
    for k = 1:iterations_k
    if k >=2
    M_vals(k) = M_vals(k-1)*sum_part*max_part_for_15;
    end
    C_vals_k(6,k) = M_vals(k)*q(18); %(16)
    C_vals_k(7,k) = M_vals(k)*d(15)*q(19); %(17)
    C_vals_k(8,k) = M_vals(k)* sum_part *d(16) * q(20); %(18)
    C_vals_k(9,k) = M_vals(k)*(1-d(15))*sum_part*q(21); %(19)
    C_vals_k(10,k) = M_vals(k) * sum_part * d(16) *
sum(d(12:14).*q(12:14)); %(20)
    %(21)
    C_vals_k(11,k) = M_vals(k) * sum_part * d(16) * ((sum_part -
p(15))/sum_part) * sum(d(12:14).*d(9:11).*q(15:17));
    end
    Z = -(S * (1-d(15)*(p(15) + sum(p(12:14)) - sum(p(12:14).*d(12:14))))
* sum(M_vals(1:iterations_k)) ...
    - (C_1 + sum(sum(C_vals_m)) + sum(sum(C_vals_k))));
    end
```

附录 5

Matlab 求解代码

```
matlab_situation_1.txt
for d_1 = 0:1
for d_2 = 0:1
for d_3 = 0:1
for d_4 = 0:1
disp("d=");
disp([d_1,d_2,d_3,d_4]);
disp("目标函数值为:");
disp(objective_function_2([d_1,d_2,d_3,d_4]))
end;
end;
end;
end;
d=
     0     0     0     0
```

```
目标函数值为:
      106400

d=
     0     0     0     1

目标函数值为:
   NaN

d=
     0     0     1     0

目标函数值为:
      110000

d=
     0     0     1     1

目标函数值为:
   NaN

d=
     0     1     0     0

目标函数值为:
   1.0887e+05

d=
     0     1     0     1

目标函数值为:
   NaN

d=
     0     1     1     0

目标函数值为:
   8.4667e+04

d=
     0     1     1     1

目标函数值为:
   NaN
```

```
d=
     1     0     0     0

目标函数值为:
   1.3553e+05

d=
     1     0     0     1

目标函数值为:
   NaN

d=
     1     0     1     0

目标函数值为:
   1.1133e+05

d=
     1     0     1     1

目标函数值为:
   NaN

d=
     1     1     0     0

目标函数值为:
      138000

d=
     1     1     0     1

目标函数值为:
   1.7494e+05

d=
     1     1     1     0

目标函数值为:
      114000

d=
         1         1         1         1

目标函数值为:
     1.4827e+05

diary off     %关闭，保存记录
```

```
matlab_situation_2.txt
for d_1 = 0:1
for d_2 = 0:1
for d_3 = 0:1
for d_4 = 0:1
disp("d=");
disp([d_1,d_2,d_3,d_4]);
disp("目标函数值为:");
disp(objective_function_2([d_1,d_2,d_3,d_4]))
end;
end;
end;
end;
d=
         0         0         0         0

目标函数值为:
          -42400

d=
         0         0         0         1

目标函数值为:
     NaN

d=
         0         0         1         0

目标函数值为:
          -30000

d=
         0         0         1         1

目标函数值为:
     NaN

d=
     0     1     0     0

目标函数值为:
      -25700

d=
     0     1     0     1

目标函数值为:
   NaN

d=
     0     1     1     0

目标函数值为:
      -56500

d=
     0     1     1     1

目标函数值为:
   NaN

d=
     1     0     0     0

目标函数值为:
       21800

d=
     1     0     0     1

目标函数值为:
   NaN

d=
     1     0     1     0

目标函数值为:
       -9000

d=
     1     0     1     1
```

```
目标函数值为:
   NaN

d=
     1     1     0     0

目标函数值为:
       38500

d=
     1     1     0     1

目标函数值为:
   1.0438e+05

d=
     1     1     1     0

目标函数值为:
       20500

d=
     1     1     1     1

目标函数值为:
   8.1875e+04

diary off    %关闭，保存记录
```

```
matlab_situation_3.txt
for d_1 = 0:1
for d_2 = 0:1
for d_3 = 0:1
for d_4 = 0:1
disp("d=");
disp([d_1,d_2,d_3,d_4]);
disp("目标函数值为:");
disp(objective_function_2([d_1,d_2,d_3,d_4]))
end;
end;
end;
end;
d=
     0     0     0     0

```

```
目标函数值为:
         39200

d=
     0     0     0     1

目标函数值为:
   NaN

d=
     0     0     1     0

目标函数值为:
        110000

d=
     0     0     1     1

目标函数值为:
   NaN

d=
     0     1     0     0

目标函数值为:
   6.3267e+04

d=
     0     1     0     1

目标函数值为:
   NaN

d=
     0     1     1     0

目标函数值为:
   8.4667e+04

d=
     0     1     1     1

目标函数值为:
   NaN
```

```
d=
     1     0     0     0

目标函数值为:
   8.9933e+04

d=
     1     0     0     1

目标函数值为:
   NaN

d=
     1     0     1     0

目标函数值为:
   1.1133e+05

d=
     1     0     1     1

目标函数值为:
   NaN

d=
     1     1     0     0

目标函数值为:
      114000

d=
     1     1     0     1

目标函数值为:
   1.4827e+05

d=
     1     1     1     0

目标函数值为:
      114000

d=
        1         1         1         1

目标函数值为:
    1.4827e+05

diary off     %关闭，保存记录
```

```
matlab_situation_4.txt
for d_1 = 0:1
for d_2 = 0:1
for d_3 = 0:1
for d_4 = 0:1
disp("d=");
disp([d_1,d_2,d_3,d_4]);
disp("目标函数值为:");
disp(objective_function_2([d_1,d_2,d_3,d_4]))
end;
end;
end;
end;
d=
        0         0         0         0

目标函数值为:
        -167200

d=
        0         0         0         1

目标函数值为:
    NaN

d=
        0         0         1         0

目标函数值为:
         -20000

d=
        0         0         1         1

目标函数值为:
    NaN

d=
     0     1     0     0

目标函数值为:
      -87100

d=
     0     1     0     1

目标函数值为:
   NaN

d=
     0     1     1     0

目标函数值为:
      -21500

d=
     0     1     1     1

目标函数值为:
   NaN

d=
     1     0     0     0

目标函数值为:
      -52100

d=
     1     0     0     1

目标函数值为:
   NaN

d=
     1     0     1     0

目标函数值为:
       13500

d=
     1     0     1     1
```

```
目标函数值为:
   NaN

d=
     1     1     0     0

目标函数值为:
        28000

d=
     1     1     0     1

目标函数值为:
   9.1250e+04

d=
     1     1     1     0

目标函数值为:
        68000

d=
     1     1     1     1

目标函数值为:
   1.4125e+05

diary off     %关闭，保存记录
```

```
matlab_situation_5.txt
for d_1 = 0:1
for d_2 = 0:1
for d_3 = 0:1
for d_4 = 0:1
disp("d=");
disp([d_1,d_2,d_3,d_4]);
disp("目标函数值为:");
disp(objective_function_2([d_1,d_2,d_3,d_4]))
end;
end;
end;
end;
d=
     0     0     0     0
```

```
目标函数值为:
       35800

d=
     0     0     0     1

目标函数值为:
   NaN

d=
     0     0     1     0

目标函数值为:
       64000

d=
     0     0     1     1

目标函数值为:
   NaN

d=
     0     1     0     0

目标函数值为:
       97100

d=
     0     1     0     1

目标函数值为:
   NaN

d=
     0     1     1     0

目标函数值为:
       90500

d=
     0     1     1     1

目标函数值为:
   NaN
```

```
d=
      1     0     0     0

目标函数值为:
   1.8667e+03

d=
      1     0     0     1

目标函数值为:
   NaN

d=
      1     0     1     0

目标函数值为:
  -1.3333e+03

d=
      1     0     1     1

目标函数值为:
   NaN

d=
      1     1     0     0

目标函数值为:
   6.3167e+04

d=
      1     1     0     1

目标函数值为:
   9.4568e+04

d=
      1     1     1     0

目标函数值为:
   5.3167e+04

d=
     1     1     1     1

目标函数值为:
   8.3457e+04
diary off     %关闭，保存记录
```

```
matlab_situation_6.txt
for d_1 = 0:1
for d_2 = 0:1
for d_3 = 0:1
for d_4 = 0:1
disp("d=");
disp([d_1,d_2,d_3,d_4]);
disp("目标函数值为:");
disp(objective_function_2([d_1,d_2,d_3,d_4]))
end;
end;
end;
end;
d=
     0     0     0     0

目标函数值为:
      184300

d=
     0     0     0     1

目标函数值为:
   NaN

d=
     0     0     1     0

目标函数值为:
      180000

d=
     0     0     1     1

目标函数值为:
   NaN

d=
     0     1     0     0
```

```

目标函数值为:
    1.7460e+05

d=
      0       1       0       1

目标函数值为:
    NaN

d=
      0       1       1       0

目标函数值为:
    1.5295e+05

d=
      0       1       1       1

目标函数值为:
    NaN

d=
      1       0       0       0

目标函数值为:
    1.9249e+05

d=
      1       0       0       1

目标函数值为:
    NaN

d=
      1       0       1       0

目标函数值为:
    1.7084e+05

d=
      1       0       1       1

目标函数值为:
    NaN

d=
       1        1        0        0

目标函数值为:
    1.8279e+05

d=
       1        1        0        1

目标函数值为:
    1.8355e+05

d=
       1        1        1        0

目标函数值为:
    1.5779e+05

d=
       1        1        1        1

目标函数值为:
    1.5723e+05
diary off     %关闭，保存记录
```

```
matlab_B_3_output.txt
diary on;
% 设置变量的上下界，假设 d1 到 d4 的取值范围为[0,1]
lb = zeros(16,1); % 下界
ub = ones(16,1); % 上界
% 定义整数约束，表示 d1, d2, d3, d4 都是整数变量
IntCon = [1, 2, 3, 4,5,6,7,8,9,10,11,12,13,14,15,16];
% 使用遗传算法求解，并设置整数约束
options = optimoptions('ga', 'Display', 'iter', 'PopulationSize', 200, 'MaxGenerations', 300);
% 调用遗传算法，指定 16 个整数变量
[x, fval] = ga(@objective_function_3, 16, [], [], [], [], lb, ub, [],IntCon,options);

Single objective optimization:
16 Variable(s)
16 Integer variable(s)

Options:
CreationFcn:          @gacreationuniformint
```

```
CrossoverFcn:           @crossoverlaplace
SelectionFcn:          @selectiontournament
MutationFcn:            @mutationpower

                                     Best              Mean            Stall
Generation          Func-count      Penalty           Penalty       Generations
     1                  400        -7.849e+04        -4.752e+04           0
     2                  595        -7.983e+04        -5.558e+04           0
     3                  790        -7.983e+04        -5.704e+04           1
     4                  985        -8.183e+04        -5.998e+04           0
     5                 1180        -8.183e+04        -6.157e+04           1
     6                 1375        -8.183e+04        -6.001e+04           2
     7                 1570        -8.183e+04        -6.091e+04           3
     8                 1765        -8.183e+04        -6.009e+04           4
     9                 1960        -8.183e+04        -5.858e+04           5
    10                 2155        -8.183e+04        -5.724e+04           6
    11                 2350        -8.183e+04        -5.933e+04           7
    12                 2545        -8.183e+04        -6.072e+04           8
    13                 2740        -8.183e+04        -5.959e+04           9
    14                 2935        -8.183e+04        -5.833e+04          10
    15                 3130        -8.183e+04        -5.994e+04          11
    16                 3325        -8.183e+04        -6.038e+04          12
    17                 3520        -8.183e+04        -5.961e+04          13
    18                 3715        -8.183e+04        -5.995e+04          14
    19                 3910        -8.183e+04        -6.071e+04          15
    20                 4105        -8.183e+04        -5.841e+04          16
    21                 4300        -8.183e+04         -6.06e+04          17
    22                 4495        -8.183e+04        -5.906e+04          18
    23                 4690        -8.183e+04        -5.844e+04          19
    24                 4885        -8.183e+04        -5.907e+04          20
    25                 5080        -8.183e+04        -6.068e+04          21
    26                 5275        -8.183e+04        -5.955e+04          22
    27                 5470        -8.183e+04        -5.879e+04          23
    28                 5665        -8.183e+04        -5.699e+04          24
    29                 5860        -8.183e+04         -5.86e+04          25

                                     Best              Mean            Stall
Generation          Func-count      Penalty           Penalty       Generations
    30                 6055        -8.183e+04        -5.984e+04          26
    31                 6250        -8.183e+04        -6.124e+04          27
    32                 6445        -8.183e+04        -6.002e+04          28
    33                 6640        -8.183e+04        -5.942e+04          29
    34                 6835        -8.183e+04        -5.912e+04          30
    35                 7030        -8.183e+04        -5.913e+04          31
```

```
     36                 7225       -8.183e+04        -6.13e+04        32
     37                 7420       -8.183e+04       -6.141e+04        33
     38                 7615       -8.183e+04       -6.055e+04        34
     39                 7810       -8.183e+04       -6.116e+04        35
     40                 8005       -8.183e+04       -6.099e+04        36
     41                 8200       -8.183e+04       -5.978e+04        37
     42                 8395       -8.183e+04       -5.905e+04        38
     43                 8590       -8.183e+04       -5.739e+04        39
     44                 8785       -8.183e+04       -5.931e+04        40
     45                 8980       -8.183e+04       -5.843e+04        41
     46                 9175       -8.183e+04        -6.04e+04        42
     47                 9370       -8.183e+04       -6.027e+04        43
     48                 9565       -8.183e+04       -5.976e+04        44
     49                 9760       -8.183e+04       -5.963e+04        45
     50                 9955       -8.183e+04       -5.937e+04        46
     51                10150       -8.183e+04       -5.908e+04        47
     52                10345       -8.183e+04       -5.917e+04        48
     53                10540       -8.183e+04       -6.004e+04        49
     54                10735       -8.183e+04       -5.932e+04        50
Optimization terminated: average change in the penalty fitness value less than
options.FunctionTolerance
and constraint violation is less than options.ConstraintTolerance.
% 输出结果
disp('最佳解:');
最佳解:
disp(x);
     1     1     0     0     0     0     0     0     1     0     0     0
1     1     0     0

disp('目标函数值:');
目标函数值:
disp(fval);
   -8.1827e+04

%零件检测
x(1:8)

ans =

     1     1     0     0     0     0     0     0

%半成品与成品检测
x(12:15)
```

```
ans =

     0     1     1     0

%半成品拆解与成品拆解
x(9:11)

ans =

     1     0     0

x(16)

ans =

     0

diary off
```