生产过程中的决策研究

摘要

本文主要研究了生产过程中如何获得最优决策的问题。生产决策需要企业根据产品信息及工序流程综合评价后做出如何获得低成本高利润的最优决策。该问题的研究对于实际工业生产具有重大意义。

针对问题一，基于抽样检测的要求，采用**统计假设检验原理**构建次品率的单边检验模型，满足信度的同时降低生产风险，得出是否接受零件的最优决策。**计算得满足 $Z\geq1.65$ 时拒收这批零件；$Z\leq-1.26$ 时接收这批零件。**

针对问题二，对题目所给情形概括出 16 中可能情况，分别采用**枚举法**分析是否检测和拆解，计算每种可能情况下的期望购买成本、期望检测成本、期望拆解成本、期望调换成本，最后求和得到生产出每个合格产品的**期望成本和期望利润**，得出最优决策，并对问题二模型进行敏感度分析。采用最优决策计算得到期望利润为 **18.53 元(情况一)；9.75 元(情况二)；16.24 元(情况三)；14.75 元(情况四)；14.98 元(情况五)；21.68 元(情况六)**。

针对问题三，在问题二的基础上建立**贪心算法**模型，在对于 m 道工序，n 个零件的生产过程中，给出将生产过程分成多个类似于问题二的子模型，分别对每个子模型进行遍历求解，从而减少题目复杂度。由于贪心算法模型的局限性考虑引入**期望风险成本**，对上一个子问题中不同决策可能会对后面工序所产生的损失进行衡量，并用前一个模块的期望制造成本与期望风险成本求和来寻找最优决策，最终将所有子问题的局部最优解加以整合，得出问题的一种全局最优决策。最优方案下的**期望成本为 140.10 元，期望利润为 59.9 元**。

针对在问题四的情景下问题二和问题三的共同点，在问题二和问题三的基础上进一步深化，采用**贝叶斯推理**对次品率构建模型，以化解抽样检测的偶然性，通过多次计算次品率的**后验概率**，不断更新次品率，使得实际收益逐渐接近预期收益，得出真实次品率下的最优决策。

关键词：假设检验；枚举法；贪心算法；贝叶斯推理。

一．问题重述

某企业生产某种电子产品，需要购买两种零配件。在企业将两个零配件装配成成品的过程中，成品是否合格具有严格的判断标准。拆解过程不会对零配件造成损坏。对于不合格成品，企业可以选择报废或拆解但需要花费拆解费用。

针对本问题背景，共需要解决四个问题。

问题一主要分析企业在从供应商处接收零部件时，若采用抽样检测的方法，并假定标称值为 10%，检测费用由企业自行承担，在两种特定的给定信度的场景下，如何获得检测次数最少的检测方案以判断是否拒收或接收这批零部件。

问题二主要对企业生产过程的各个阶段作出最优决策，根据表 1 中所给数据，在给定零部件以及成品的次品率的情况下，通过分析各个生产过程之间的关系，分析六种不同情况下相应的衡量指标，以得出具体的决策方案，并给出决策的依据及相应的指标结果。

问题三是在问题二的基础上，讨论更具普遍性的决策方案，考虑到生产实际情况提出了半成品的概念。在 m 道工序、n 个零配件的前提下，根据工序与零配件之间的关系以及零配件、半成品和成品的次品率等参数，分析相应指标，得出决策方案。并在给出的具体情况下给出具体的决策方案。

问题四在问题二和问题三的基础上进一步推广，限定了抽样检测的条件，并分析在新条件下分别对应的最优决策方案。

二．问题分析

2.1 对问题一的分析

该问题要求在次品率不超过 10%的标称值的情况下设计检验次数尽可能少的抽样检测方案。信度是检验结果一致性、稳定性及可靠性的重要指标。本问题主要基于**统计假设检验模型**，对次品率 p 进行**单边假设检验**，控制次品率的同时降低生产过程中的风险，确保在不同置信水平下对零配件是否接收做出准确的**决策**，从而给出最优的抽样检测方案。

2.2 对问题二的分析

该问题要求在已知零配件和成品次品率的条件下，对企业生产过程的各个阶段做出决策，以实现**利润的最大化**。对题目中所给生产阶段分析可以得到对于整个阶段的生产决策的总数为可列有限个的结论结合，采用**枚举法分析**，计算出每种情况对应的生产单

个成品的**期望成本和期望利润**，以得出相对于题中所给的任何一种情况的最优生产决策方法。

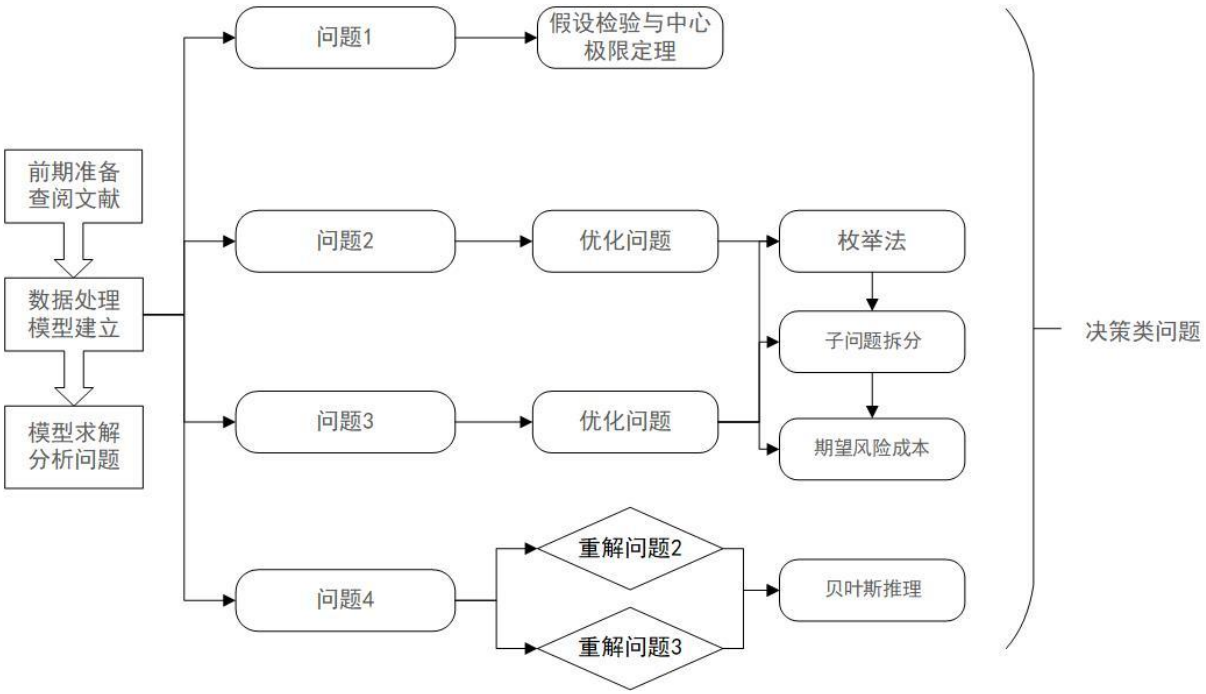
2.3 对问题三的分析

该问题在问题二的基础上进一步贴近生产实际，引入半成品的概念，提出在 m 道工序，n 个零配件的情况下进行最优决策。由于工序与问题二是相同的过程，故在本问题中采用**贪心算法**，将该问题分成若干个子问题，通过对子问题进行类似于问题二的遍历算法，得到每个子问题的**局部最优解**，最后合成为原来问题的一个解。由于贪心算法的局部性，特引入**期望风险成本**这一概念，用于评估各种决策下对后续生产过程可能产生的损失风险成本。

2.4 对问题四的分析

该问题在问题二和问题三的基础上要求采用抽样检测的方式得到次品率，由于考虑到抽样检测在抽样的过程中，所抽样品的质量具有**偶然性**，实际的次品率可能高于观测的次品率。因此在此处采用**贝叶斯推理**(Bayesian Inference)的方法构建模型，通过计算**先验概率和后验概率**使观测次品率逐步接近实际值，当观测值与实际值差值在可接受范围内时，即给出最优决策方案。

2.5 总体思路图



三．模型假设与符号说明

3.1 模型假设

1.组装过程不会对零件造成损坏；
2.检测结果完全准确，不会出现错检情况；
3.报废不合格零件和成品不会产生额外成本；
4.不考虑各种工序耗时不同带来的效益收益。

3.2 符号说明

符号	定义	单位
p	次品率	/
p_0	标称值	/
C_{test}^{k}	检测成本	元/件
$\mathrm{C}_{\mathrm{disa}}^{\mathrm{k}}$	拆解成本	元/件
$C_{assemble}^{k}$	装配成本	元/件
C_{buy}^{k}	购买零件的成本	元/件
C_{swap}^{k}	调换成本	元/件
C_{risk}^{k}	期望风险成本	元/件
C_{total}	期望总成本	元/件

四．模型的建立与求解

4.1 问题一的模型建立与求解

4.1.1 模型建立

问题一要求采用抽样检测的方法从供应商处接收一批零部件，根据统计假设，已知零部件次品率的标称值以及信度，分析在两种情形下是否接收或拒绝假设，并且使检验次数尽可能少。根据题目要求建立**单边检验模型**如下：

（1）根据实际情形建立原假设 H0 和备择假设 H1。

（2）为了求出合适的统计量 Z 以及拒绝域形式，引用**二项分布与正态分布模型**。

由于抽样检测中每个零件的状态可以看做是二项分布的结果，故构建二项分布的**概率密度函数**：

$$\mathrm{P}(\mathrm{X}=\mathrm{k}) = \binom{\mathrm{n}}{\mathrm{k}} \mathrm{p}^{\mathrm{k}} (1-\mathrm{p})^{\mathrm{n-k}} \tag{1}$$

来描述在 n 次独立的伯努利试验中随机变量 X 取到次品的次数为 k，p 为次品率。

由于检测对象的数量足够大，根据**中心极限定理**可将二项分布近似为期望为np,方差为$np(1-p)$的**正态分布**：

$$\mathrm{X}\sim\mathrm{N}(\mathrm{np},\ np(1-p))$$

根据检验统计量Z的**计算公式**：

$$\mathrm{Z} = \frac{\mathrm{p}-\mathrm{p}_0}{\sqrt{\frac{\mathrm{p0}(1-\mathrm{p0})}{\mathrm{n}}}} \tag{2}$$

可以构造出新的的统计量z_α。

根据单边检验的结论，当次品率与标称值满足某不等关系时有：

$$\mathrm{z} \geq \mathrm{z}_\alpha$$

或

$$\mathrm{z} \leq -\mathrm{z}_\alpha$$

根据具体的情形判断是否拒收或接收这批零件。

其中，z_α可根据**正态分布的累积分布函数(CDF)**求出。

4.1.2 模型求解

1. 情形一

已知标称值为 10%，信度为 95%的条件下，构建假设：

原假设 H0：零配件次品率**不超过**标称值($p \le p0$)。

备择假设 H1：零配件次品率**超过**标称值($p > p0$)。

根据单边检验原理

利用

$$\frac{\overline{p}-\mu}{\sigma/\sqrt{n}} \sim N(0,1)$$

及

$$P\left\{\left|\frac{\overline{p}-\mu}{\sigma/\sqrt{n}}\right| \ge \frac{k-\mu 0}{\sigma/\sqrt{n}}\right\} = \alpha$$

又因为

$$\frac{k-\mu 0}{\sigma/\sqrt{n}} = z_{\alpha} \tag{3}$$

由此可得**拒绝域**为

$$z \ge z_{\alpha} \tag{4}$$

根据信度为 0.95，由正态分布的累积分布函数(CDF)

$$\emptyset(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt \quad (x \ge 0) \tag{5}$$

已知$\emptyset(x)=0.95$，查阅资料可知与其对应的$Z_{0.05} = 1.65$

即满足

$$Z \ge 1.65$$

这个条件时否定原假设H_0，即不能认为零部件的次品率没有超过标称值，而是次品率超过了标称值，拒收该批零部件。

在假设检验中，由于样本的**随机性**，在做出判断时有可能出现错误[1]，于是我们提出**可接受误差*E***，满足：

$$n = \left(\frac{z_{\alpha}\sqrt{p0(1-p0)}}{E}\right)^2 \tag{6}$$

本题中，将可接受误差设为 **5%**，以提高**容错率**，利用上述计算可接受误差的公式可以计算出在满足可接受误差前提下的最小抽样数为 99 份，当检出 15 份及以上次品时，可以在 95%的信度下认为零配件次品率超过标称值，拒收这批零配件。下表(表 1)展示了部分当信度为 95%及以上时，部分抽样数所对应的最少次品数，以验证结果的正确。

表 1 95%信度下考虑误差的实验抽样

95%	
抽样数(次)	最少次品数(个)
99	15
112	17
150	21
187	26
199	27

2. 情形二

已知标称值为 10%，信度为 90%的条件下，构建假设：

原假设 H0：零配件的次品率**超过**标称值($p > p0$)

备择假设 H1：零配件的次品率**不超过**标称值($p \leq p0$)

同理，情形二为单边检验的左边检验问题

利用

$$\frac{\overline{p}-\mu}{\sigma/\sqrt{n}} \sim N(0,1)$$

得到检验统计量：

$$Z = \frac{\overline{p}-\mu}{\sigma/\sqrt{n}} \tag{7}$$

拒绝域为：

$$z \leq -z_{\alpha} \tag{8}$$

根据信度为 0.9，由正态分布的累积分布函数(CDF)

$$\emptyset(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt \quad (x \geq 0) \tag{9}$$

已知$\emptyset(x)=0.9$，查阅资料可得其对应的$Z_{0.1} = -1.27$

即满足

$$Z \leq -1.27$$

这个条件时，可以在犯错误概率不超过 10%的概率下否定原假设，并认为次品率不超过标称值，接受这批零件。

在假设检验中，由于样本的**随机性**，在做出判断时有可能出现错误，于是我们提出**可接受误差*E***，满足：

$$n = \left(\frac{z_{\alpha}\sqrt{p0\ (1-p0)}}{E}\right)^2 \tag{10}$$

本题中，将可接受误差设为**5%**，以提高容错率。由上述计算最小抽样数的公式可得在满足可接受误差前提**5%**下的最小样本数为59，当检出3份及以上次品时，可以在90%的信度下认为零配件的次品率超过了标称值，选择拒收这批零配件。如下表 2，展示了当信度为90%以上时，部分抽样数所对应的最多次品数，以验证结果的正确性。

表 2 90%信度下考虑误差的实验抽样

90%	
抽样数(次)	最多次品数(个)
59	3
78	5
92	6
104	7
155	11

4.2 问题二的模型建立与求解

4.2.1 模型建立

根据题干要求：在装配的成品中，只要其中一个零配件不合格，则成品一定不合格；如果两个零配件均合格，装配出的成品也不一定合格。对于不合格成品，企业可以选择报废，或者对其进行拆解，拆解过程不会对零配件造成损坏，但需要花费拆解费用。

在决策过程中，由于每个零配件都有次品率和检测成本，因此我们必须考虑不检测该零件是否会**对后续生产阶段产生影响**，避免后续生产阶段中出现次品。此外，如果某个生产阶段的次品率较高且检验成本较低，应优先在该阶段进行检测。反之，若某道工序的次品率较低但检验成本高，则直接跳过该工序。

转化为**程序图语言**即为图 1所示。

当零件1与零件2中有一个零件未检测且不合格，则成品不合格，选择拆解后，若此不合格零件继续不检测，则会不停循环，浪费时间与资源，故实际生产中不会作出这种决策。

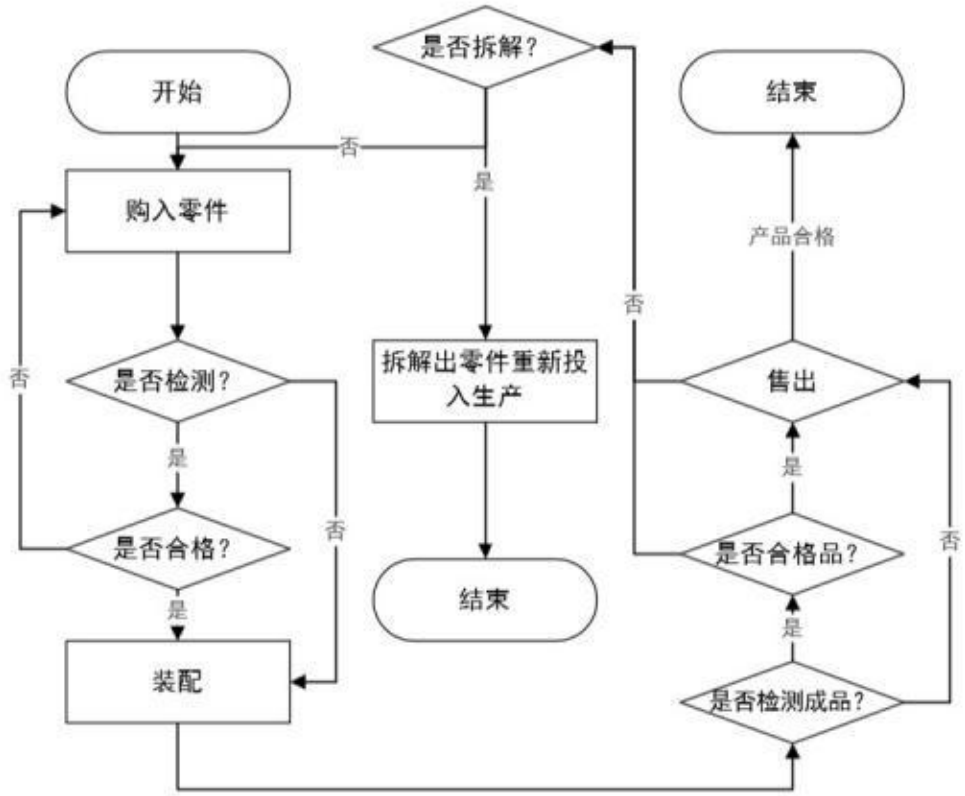


图 1 企业生产决策流程图

因此，根据**实际生产情况**，为模型提出如下**假设**：

1. 当零件未检测且成品不合格时，选择拆解后，此零件必须检测

2. 当零件已检测但成品不合格时，选择拆解后，此零件无需检测；

3. 当事件发生的概率极低时，模型对其进行忽略，对实验结果几乎不产生影响。

如计算某一事件循环往复发生时，用$(p + p^2 + p^3 + p^4 + p^5)$表示事件可能发生的概率，将极低概率事件产生的成本略去。

经分析，每一种情况按照是否检测零件一，是否检测零件二，是否检测成品，是否拆解不合格成品排列组合可总结出 **16 种策略**，由于策略数在可算范围内，故采用**枚举法**，通过计算分析**期望成本和期望利润**，作为判断决策优劣的标准以及衡量指标。

令P^k为各类情况发生的概率，X^k 为各类决策变量(其中 X_{pt}^k 为零件 k 是否检测，X_{ft} 为成品是否被检测，X_{fd} 为成品是否拆解)，若检测，则 X^k 取值为 1；若不检测，X^k 取值为 0.

若不检测零件 1，即 $X_{pt}^{1}=0$，则后续成本次品率

$$P^{k}=1-\left[\left(1-P_{p}^{1}\right)\left(1-P_{f}\right)\right] \tag{11}$$

若零件 1 和零件 2 都不检测，即 $X_{pt}^{1}=X_{pt}^{2}=0$，则后续次品率为

$$P^{k}=1-\left[\left(1-P_{p}^{1}\right)\left(1-P_{p}^{2}\right)\left(1-P_{f}\right)\right] \tag{12}$$

因此可以分别得出

检测期望总成本 C_{test}^{k} 为

$$\sum_{k} P^{k} C_{test}^{k} X^{k} \tag{13}$$

拆解期望总成本 C_{disa}^{k} 为 $\sum_{k} P^{k} C_{test}^{k} X^{k}$

$$\sum_{k} P^{k} C_{disa}^{k} X^{k} \tag{14}$$

装配期望总成本 $C_{assemble}^{k}$ 为注释：

$$\sum_{k} P^{k} C_{assemble}^{k} \tag{15}$$

购买零件期望总成本 C_{buy}^{k} 为

$$\sum_{k} P^{k} C_{buy}^{k} \tag{16}$$

调换期望总成本 C_{swap}^{k} 为

$$\sum_{k} P^{k} C_{swap}^{k} \tag{17}$$

那么，产出单个成品时，**期望总成本** C_{total} 为

$C_{total}=$

$$\sum_{k} P^{k} C_{test}^{k} X^{k}+\sum_{k} P^{k} C_{disa}^{k} X^{k}+\sum_{k} P^{k} C_{assemble}^{k}+\sum_{k} P^{k} C_{swap}^{k}+\sum_{k} P^{k} C_{buy}^{k} \tag{18}$$

而生产单个产品的**期望利润**为单个产品售价与生产单个产品成本之差。

4.2.2 模型求解

针对题目中所给的每一种情况，分别枚举出对应的期望总成本与利润，采用**遍历算法**分别进行计算，用**雷达图**展示最终 16 种策略的期望成本和期望利润。

如图 2 所示。

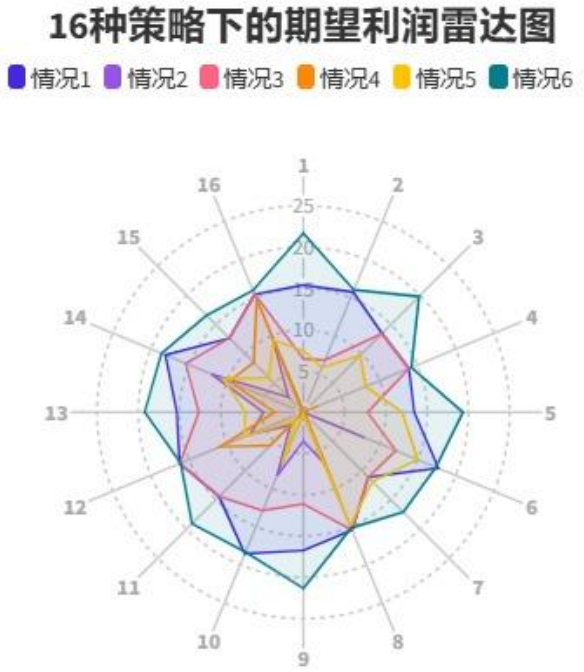


图 2 16 种策略下的期望利润雷达图

结合具体工业生产实际，以**期望成本和期望利润**作为决策的衡量依据，根据雷达图，为获得**低成本，高利润**的方案，得出最终的**决策方案以及指标**如表 3 问题二决策方案及指标所示。

表 3 问题二决策方案及指标

	是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	期望成本(元)	期望利润(元)
情况 1	是	否	否	是	37.47	18.53
情况 2	是	是	是	是	46.25	9.75
情况 3	是	否	是	是	39.76	16.24
情况 4	是	是	是	是	41.25	14.75
情况 5	否	是	否	是	41.02	14.98
情况 6	否	否	否	否	34.32	21.68

4.3 问题三的模型建立与求解

4.3.1 模型建立

该问题在问题二的基础上进一步深化，提出具有 m 道工序，n 个零配件的复杂问题，由于直接分析该问题需要考虑多种因素，不确定性较大，因此在此处采用**贪心算法**[3]构建模型，将问题三分成若干个子问题，对每个子问题进行类似于问题二模型的遍历求出每种情况下的制造费用C_{manu}^{k}。

将上一个子问题计算所得的成品制造费用作为下一个模块的零件购买费用

即

$$\mathrm{C_{buy}^{k+1}} = \mathrm{C_{manu}^{k}} \tag{19}$$

使用问题二模型的遍历算法，得到该子问题的最优决策，最终将每个子问题的最优决策加以整合，得到原问题的一种最优决策。

其中，由于考虑到贪心算法的局部性，我们引入**期望风险成本**这一概念。以评估某个阶段的各种决策对后续生产过程可能产生的损失风险成本。

期望风险成本就是在子问题 k 中，所产生的成品在选择是否要进行检测，都有可能对后续生产过程产生出现次品的风险，从而产生额外成本。

当$X_f^k = 0$，即不检测成品时，有

$$\mathrm{C_{risk}^{k}} = \mathrm{P_{p}^{k}}\left[\mathrm{C_{assemble}^{k+1}} + 0.5\left(\mathrm{C_{test}^{k+1}} + \mathrm{C_{swap}^{k+1}} + \mathrm{C_{disa}^{k+1}} + \mathrm{C_{manu}^{k+1}}\right)\right] + \mathrm{P_{f}^{K+1}C_{manu}^{k}} + \ldots\ldots \tag{20}$$

当X_f^k =1，即检测成品时

$$\mathrm{C_{risk}^{k}} = \mathrm{P_{f}^{k}}\left(\mathrm{C_{test}^{k}} + \mathrm{C_{manu}^{k}}\right) + \ldots\ldots \tag{21}$$

每个子问题中，目标函数均为

$$\min\left[C_{risk}^{k} + C_{manu}^{k}\right] \tag{22}$$

通过逐级求解，最终得到所有子问题的最优决策，即该问题的全局最优决策。

4.3.2 模型求解

根据题目要求，零件 1，2，3 合成半成品 1；零件 4，5，6 合成半成品 2 的过程完全一致，因此只需要求解一次。以此为依据将该问题拆分为子问题，如图 3 所示。

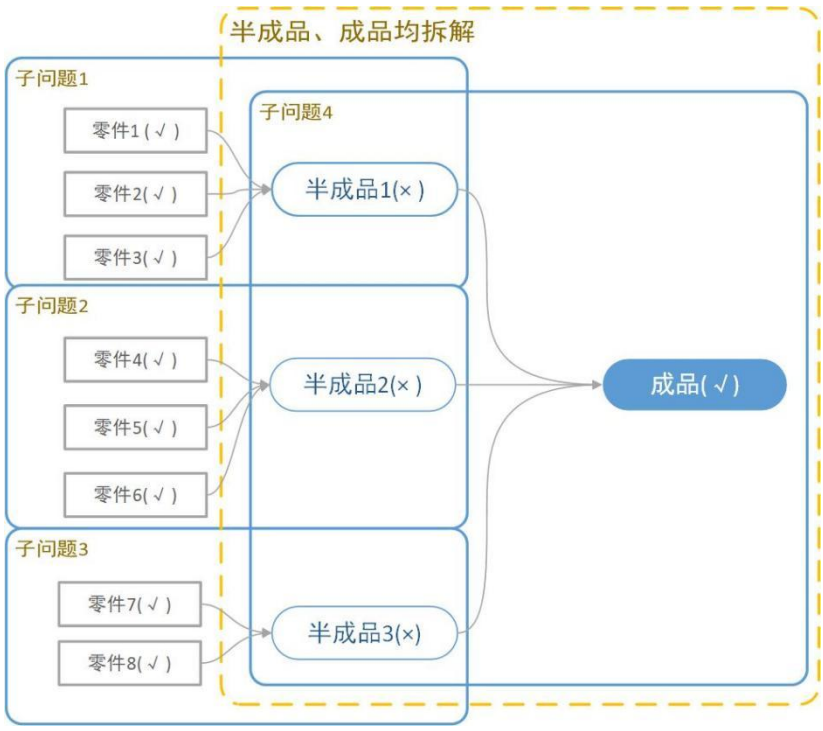


图 3 子问题的拆分与结果展示(检测与否分别用v、×表示)

首先，使用问题二中的模型对子问题 1,2,3 进行求解，得到各自不同决策下的期望制造成本。

接着，计算 3 个子问题不同决策下的期望风险成本，并与相应期望制造成本相加，并进行比较，找出目标函数最低的决策，即为各模块的最优决策。

最后，选择各子问题的最优决策及其制造成本为前置决策及半成品 1,2,3 的购买成本，使用简化后的问题二的模型(即各半成品是否检测的决策变量已知)，以求出模块四的最优决策，与前面模块 1,2,3 的最优决策相拼接，得到全局最优模块。

在此模型下计算可得最优方案如表 4 所示，具体检测过程详见附录

表 4 问题 3 最优方案

检测指标	是否检测
零件 1，4 是否检测	是
零件 2，5 是否检测	是
零件 3，6 是否检测	是
零件 7 是否检测	是
零件 8 是否检测	是
半成品是否检测	否
半成品 1,2,3 是否拆解	是
成品是否检测	是
成品是否拆解	是

在问题三中，考虑到实际生产情况，除关注期望成本和期望利润外，还要关注风险

成本，根据这种方案进行指标计算，得到最终结果如表 5 所示

表 5 问题三的指标及取值

指标	参数(元)
第 1、2 组的制造成本加风险成本	46.65
第 3 组的制造成本加风险成本	28.03
期望成本	140.10
期望利润	59.90

4.4 问题四的模型建立与求解

4.4.1 模型建立

问题四要求在问题二和问题三的基础上，采用抽样检测的方法得到次品率，考虑到抽样检测的**偶然性**，此处采用**贝叶斯推理**[2]的方法进行分析。

贝叶斯推理(Bayesian inference)的主要思想是通过不断更新假设的概率，使次品率的观测值向实际值不断靠近。

首先提出**先验概率** P(H)，即在得到实际次品率前次品率的观测值。

根据贝叶斯公式得到后验概率

$$P(H|E) = \frac{P(E|H)\cdot P（H）}{P（E）} \tag{23}$$

其中，H 代表可能影响次品率可能收到数据影响的任何假设，E 表示未被用于计算鲜先验数据的新数据。P(E)表示为**边际似然函数**，该因子不会影响各个部分之间的相对概率，P(E|H)为**似然函数**，是在假设 H 的前提下观测到 E 的概率。

根据**后验概率**我们即可得到当 H 给予 E 以后，也即在观测到证据 E 以后，更新的概率。

此外，为了提高该模型的可用性，方便具体计算，我们需要对先验分布和似然函数给出明确的建模，我们先假设先验分布是一个 **Beta 分布**，Beta 分布有两个参数，α和β，它们与先验的均值和方差有关。因为 Beta 分布是二项分布参数的共轭先验，后验分布也将是 Beta 分布，这使得计算更加方便。

假设先验是 Beta(α,β)，并且观测到 x 次故障和 n−x 次成功，则后验分布将是 Beta(α+x,β+n−x)。

经过多次计算，使用上述的 Beta 分布更新规则来计算后验分布的参数。后验分布将比先验分布更“集中”在实际值附近。且先验分布越“平坦”(即方差越大)，则后验分布将主要由观测到的数据决定；如果先验分布越“尖锐”(即方差很小)，则后验分布将更多地受到先验分布的影响。

4.4.2 模型求解

根据上述模型以及题目要求，可以将工业生产的检测流程构建为以下**程序语言**，如图 4 所示。

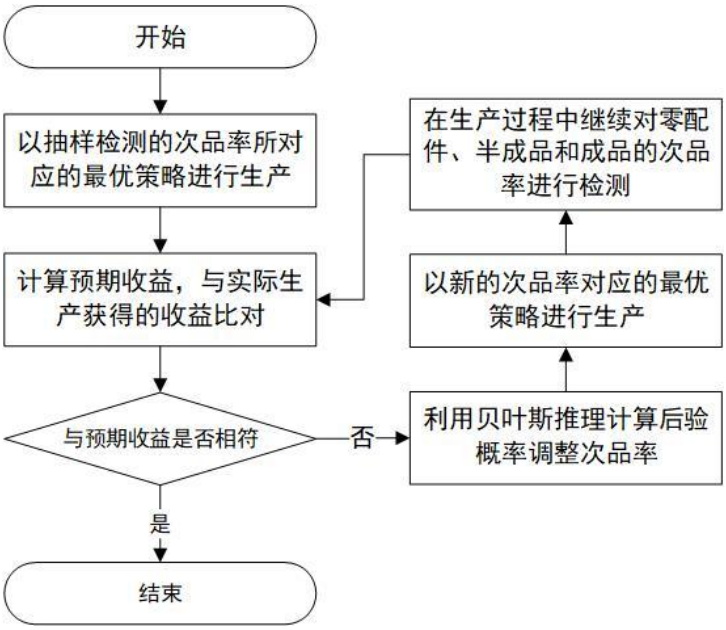


图 4 问题四生产流程图

1. 问题二决策优化

结合题目中所给出的具体情形，考虑到用抽样检测方式获得次品率，对问题二的决策进行更新。

首先按照题目表格中所给出的抽样检测所得到的次品率所对应的最优策略进行生产，计算出预期收益并根据实际生产数据得出实际收益。

经比对，预期收益与实际收益存在误差，因此采用贝叶斯推理模型计算出后验概率，作为新的次品率，以这个次品率所对应的最优策略进行生产，得到新的预期收益和实际收益。

如此循环往复，逐渐能够调整预期值与实际值相匹配。当预期收益与实际收益的差小于10^{-6}时，停止计算，认为此时所得的次品率所对应的生产策略为最优策略。

具体结果见表 6 到表 11，计算过程详见附录

情况一

零件 1，2，成品的抽样次品率分别为 10%，10%，10%；利用贝叶斯原理修正后的次品率分别为 12.5%，12.5%，12.5%。

表 6 情况一决策方案

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	预期收益（元）	实际收益（元）
是	否	否	是	16.7145	18.1112

是	是	否	是	18.5325	18.5325

情况二

零件 1，2，成品的抽样次品率分别为 20%，20%，20%；利用贝叶斯原理修正后的次品率分别为 22.5%，22.5%，22.5%。

表 7 情况二决策方案

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	预期收益（元）	实际收益（元）
是	是	否	是	12.0035	10.2332

情况三

零件 1，2，成品的抽样次品率分别为 10%，10%，10%；利用贝叶斯原理修正后的次品率分别为 12.5%，12.5%，12.5%。

表 8 情况三决策方案

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	预期收益（元）	实际收益（元）
是	否	是	是	16.2361	14.1893

情况四

零件 1，2，成品的抽样次品率分别为 20%，20%，20%；利用贝叶斯原理修正后的次品率分别为 17.5%，17.5%，17.5%。

表 9 情况四决策方案

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	预期收益（元）	实际收益（元）
是	是	是	是	14.753	16.1528

情况五

零件 1，2，成品的抽样次品率分别为 10%，20%，10%；利用贝叶斯原理修正后的次品率分别为 12.5%，22.5%，12.5%。

表 10 情况五决策方案

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	预期收益（元）	实际收益（元）
否	是	否	是	14.9817	12.0004
否	是	是	是	12.4512	12.4512

情况六

零件 1，2，成品的抽样次品率分别为 5%，5%，5%；利用贝叶斯原理修正后的次品率分别为 4%，4%，4%。

表 11 情况六决策方案

是否检测零	是否检测零	是否检测成	是否拆解不	预期收益（元）	实际收益（元）

件 1	件 2	品	合格成品		
否	否	否	否	21.679	23.0493

综上所述，得出问题四情境下问题二的预期利润随预期次品率不断调整的三维图如图 5。调整过后，预期收益与实际收益的差小于10^{-6}。

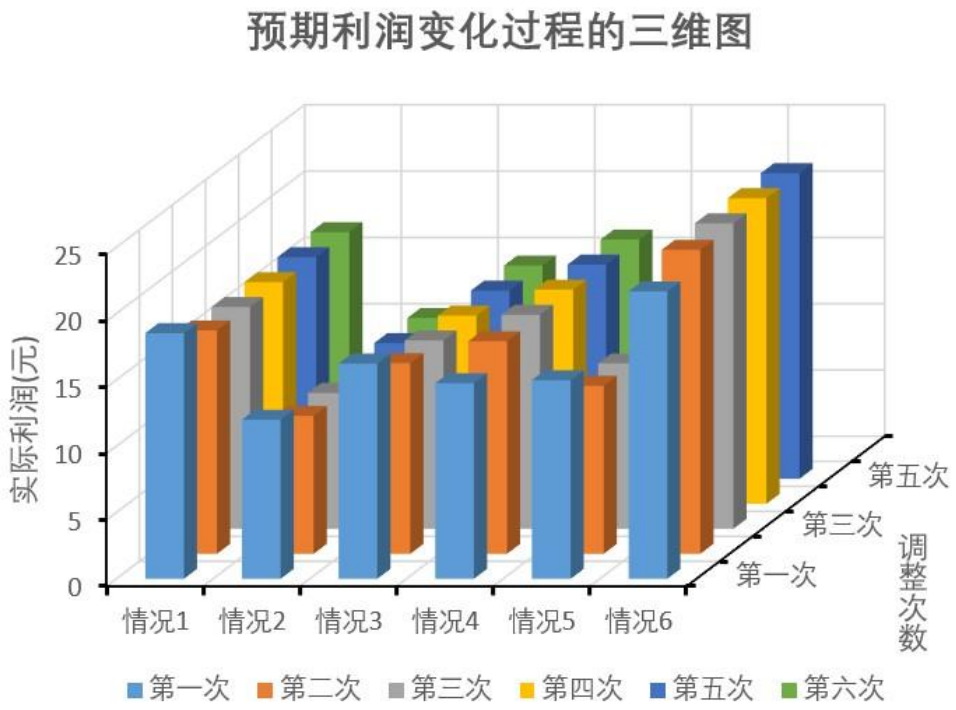


图 5 预期利润变化的三维图

初始状态按照正常生产计划进行生产，同时安排对每个零件与半成品都进行检测，以期在更大一些次数的实验中快速掌握各个零件与半成品的次品率，计算出**更准确**的次品率，对应调整生产策略。

2. 问题三决策优化

问题三解法与问题二相似，在问题二的基础上，结合实际工业生产情况，进一步考虑半成品因素。

首先按照题目表格中所给出的抽样检测所得到的零配件、半成品和成品的次品率所对应的最优策略进行生产，计算出预期收益并根据实际生产数据计算出实际收益。

经比对，预期收益与实际收益存在误差，因此采用贝叶斯推理模型计算出后验概率，作为新的次品率，以这个次品率所对应的最优策略进行生产，得到新的预期收益和实际收益。

如此循环往复，逐渐能够调整预期值与实际值相匹配。当预期收益与实际收益的差小于10^{-6}时，停止计算，认为此时所得的次品率所对应的生产策略为最优策略。

五．模型的评价与推广

1. 模型评价

优点：

1. 基于题目给出的生产过程，枚举所有不同决策下的期望成本并做出决策，避免得到局部最优解；

2. 在对于问题三多个工序的求解中，使用贪心算法降低复杂度；

3. 引入期望风险成本，来得出贪心算法易得出局部最优解的局限性；

4. 问题四利用贝叶斯原理，得出的后验次品率在逼近真实次品率时速度快，调整生产策略的次数少。

缺点：

1. 在对于更多种工序更多种原材料的生产决策问题求解时明显比较发在；

2. 所做假设可能不总符合实际情况。

2.模型推广

本题基于工业化生产背景，运用统计学知识，分析抽样检测在实际工业化生产中的应用，题目逐层深入，从特殊到一般进行逐层推广，以给出最优决策方案，在生产与生活中具有重要意义。

六．模型的检验与敏感性分析

本文主要对问题二模型进行敏感性分析。

在问题二中，由于题目中表格内都是要求固定不变的参数。为了检验模型的稳定性，特对这些参数进行修改，并将更改后所得的决策和期望成本与更改前进行对比分析。具体检验情况如表所示。

	是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	期望成本(元)	期望利润(元)
情况 1	是	否	否	是	37.47	18.53
情况 1 检测	是	否	否	是	38.58	17.42

其中，将零件 1 的检测成本由 2 元/件改为 3 元/件，得出最优决策没有发生变化，期望成本和期望利润变化不大。因此，该模型较稳定。

七. 参考文献

[1] 王雪琴.单边检验在方差检验中的局限性[J].河南科学,2011,(10): 1163-1164

[2] 王丽. 浅析贝叶斯公式及其在概率推理中的应用[J]. 科技创新导报,2010,(24):136.DOI:10.16660/j.cnki.1674-098x.2010.24.143.

[3] 杨冰,魏新莉,胡孙跃,等.基于贪心算法的板式家具订单备料调度研究[J].林产工业,2024,61(04):72-75.DOI:10.19531/j.issn1001-5299.202404012.

八．附录

8.1 问题一相关代码

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

# 假设次品率
p0 = 0.10

# 设定信度
confidence_level_1 = 0.95  # 拒收
confidence_level_2 = 0.90  # 接收

# 样本量范围
n_range = np.arange(10, 200)

# 计算对应的 k 值和拒收/接收的概率
k_values_1 = []
k_values_2 = []
for n in n_range:
    # 拒收情形的 k 值
    k1 = binom.ppf(confidence_level_1, n, p0)
    k_values_1.append(k1)

    # 接收情形的 k 值
    k2 = binom.ppf(1-confidence_level_2, n, p0)
    k_values_2.append(k2)

# 绘制 k 值随样本量变化的图
plt.figure(figsize=(12, 6))
plt.plot(n_range, k_values_1, label='95% confidence level (Refuse)', marker='o')
plt.plot(n_range, k_values_2, label='90% confidence level (Accept)', marker='x')
plt.xlabel('Sample Size (n)')
```

```
plt.ylabel('Critical Value (k)')
plt.title('Critical Value vs. Sample Size for Different Confidence Levels')
plt.legend()
plt.grid(True)
plt.show()
result_array = np.column_stack((n_range, k_values_1))

# 打印结果数组
print(result_array)
```

8.2 问题二的相关代码

代码一

```
import numpy as np
import matplotlib.pyplot as plt

#定义全局变量
p_1 = 0.1    #次品率
p_2 = 0.1
p_3 = 0.1
cost_buy_1 = 4    #零件单价
cost_buy_2 = 18
cost_assemble = 6    #装配成本
cost_test_1 = 2    #检测成本
cost_test_2 = 3
cost_test_3 = 3
cost_swap = 6    #调换损失
cost_disa = 5    #拆解费用
sell_price = 56    #市场售价

def process_1_1(x1): #未检测过检测
        c = 0
        if x1 == 0:
                c = c + cost_buy_1
```

```
        else:
            c = c + cost_buy_1 + cost_test_1 + p_1*(cost_buy_1 + cost_test_1) + \
                p_1**2*(cost_buy_1 + cost_test_1) + p_1**3*(cost_buy_1 + cost_test_1)
+ \
                p_1**4*(cost_buy_1 + cost_test_1) + p_1**5*(cost_buy_1 + cost_test_1)
        return c
    def process_1_1_1(x1): #已检测过调换后重新检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1
        return c
    def process_1_2(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2 + p_2*(cost_buy_2 + cost_test_2) + \
                p_2**2*(cost_buy_2 + cost_test_2) + p_2**3*(cost_buy_2 + cost_test_2)
+ \
                p_2**4*(cost_buy_2 + cost_test_2) + p_2**5*(cost_buy_2 + cost_test_2)
        return c
    def process_1_2_1(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2
        return c
    def process_2(x1,x2,x3,x4):
        c = 0
        if x3 == 0:
            c = c + cost_assemble
            if x4 ==0:
                if x1 == 0 and x2 == 0:
```

```
                p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
            if x1 == 1 and x2 == 0:
                p = p_2 + p_3 - p_2*p_3
            if x1 == 0 and x2 == 1:
                p = p_1 + p_3 - p_1*p_3
            if x1 == 1 and x2 == 1:
                p = p_3
            c = c + p*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                    p**2*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                    p**3*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                    p**4*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                    p**5*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble)
        else: #拆解
            if x1 == 0 and x2 == 0:
                p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                        -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2+0.5*cost_buy_1+0.5*cost_buy_2)
            if x1 == 1 and x2 == 0:
                p = p_2 + p_3 - p_2*p_3
                c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                        -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+ cost_test_2+0.5*cost_buy_2)
            if x1 == 0 and x2 == 1:
                p = p_1 + p_3 - p_1*p_3
                c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                        -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
- cost_test_2 +0.5*cost_buy_1)
```

```
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
- cost_test_2)
            else:
                c = c + cost_assemble + cost_test_3
                if x4 == 0:
                    if x1 == 0 and x2 == 0:
                        p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    if x1 == 1 and x2 == 0:
                        p = p_2 + p_3 - p_2*p_3
                    if x1 == 0 and x2 == 1:
                        p = p_1 + p_3 - p_1*p_3
                    if x1 == 1 and x2 == 1:
                        p = p_3
                    c = c + p*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**2*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**3*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**4*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**5*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3)
                else: #拆解
                    if x1 == 0 and x2 == 0:
                        p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2 + 0.5*cost_buy_1+0.5*cost_buy_2)
                    if x1 == 1 and x2 == 0:
                        p = p_2 + p_3 - p_2*p_3
```

```
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+cost_test_2 +0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
-cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
-cost_test_2 )
        return c


    def main():
        var = [[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],
               [0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],
               [1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],
               [1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]]
        cost = np.zeros(16)
        for n in range(16):
            cost[n] = process_1_1(var[n][0]) + process_1_2(var[n][1]) \
                + process_2(var[n][0],var[n][1],var[n][2],var[n][3])
        print('是否检测零件 1      是否检测零件 2'  '是否检测成品'  '是否拆解次品'  '
成本期望')
        print('否    否   否   否',cost[0])
        print('否    否   否   是',cost[1])
        print('否    否   是   否',cost[2])
        print('否    否   是   是',cost[3])
        print('否    是   否   否',cost[4])
        print('否    是   否   是',cost[5])
        print('否    是   是   否',cost[6])
```

```
        print('否    是   是   是',cost[7])
        print('是    否   否   否',cost[8])
        print('是    否   否   是',cost[9])
        print('是    否   是   否',cost[10])
        print('是    否   是   是',cost[11])
        print('是    是   否   否',cost[12])
        print('是    是   否   是',cost[13])
        print('是    是   是   否',cost[14])
        print('是    是   是   是',cost[15])

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

代码二

```
import numpy as np
import matplotlib.pyplot as plt

#定义全局变量
p_1 = 0.2   #次品率
p_2 = 0.2
p_3 = 0.2
cost_buy_1 = 4   #零件单价
cost_buy_2 = 18
cost_assemble = 6   #装配成本
cost_test_1 = 2   #检测成本
cost_test_2 = 3
cost_test_3 = 3
cost_swap = 6   #调换损失
cost_disa = 5   #拆解费用
sell_price = 56   #市场售价
#定义初始量

def process_1_1(x1): #未检测过检测
        c = 0
        if x1 == 0:
```

```
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1 + p_1*(cost_buy_1 + cost_test_1) + \
                p_1**2*(cost_buy_1 + cost_test_1) + p_1**3*(cost_buy_1 + cost_test_1)
+ \
                p_1**4*(cost_buy_1 + cost_test_1) + p_1**5*(cost_buy_1 + cost_test_1)
        return c
    def process_1_1_1(x1): #已检测过调换后重新检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1
        return c
    def process_1_2(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2 + p_2*(cost_buy_2 + cost_test_2) + \
                p_2**2*(cost_buy_2 + cost_test_2) + p_2**3*(cost_buy_2 + cost_test_2)
+ \
                p_2**4*(cost_buy_2 + cost_test_2) + p_2**5*(cost_buy_2 + cost_test_2)
        return c
    def process_1_2_1(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2
        return c
    def process_2(x1,x2,x3,x4):
        c = 0
        if x3 == 0:
            c = c + cost_assemble
            if x4 ==0:
```

```
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**2*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**3*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**4*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**5*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2+0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+ cost_test_2+0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
```

```
- cost_test_2 +0.5*cost_buy_1)
                    if x1 == 1 and x2 == 1:
                        p = p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                 -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
- cost_test_2)
            else:
                c = c + cost_assemble + cost_test_3
                if x4 == 0:
                    if x1 == 0 and x2 == 0:
                        p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    if x1 == 1 and x2 == 0:
                        p = p_2 + p_3 - p_2*p_3
                    if x1 == 0 and x2 == 1:
                        p = p_1 + p_3 - p_1*p_3
                    if x1 == 1 and x2 == 1:
                        p = p_3
                    c = c + p*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**2*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**3*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**4*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**5*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3)
                else: #拆解
                    if x1 == 0 and x2 == 0:
                        p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                                 -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2 + 0.5*cost_buy_1+0.5*cost_buy_2)
                    if x1 == 1 and x2 == 0:
```

```
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+cost_test_2 +0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
-cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
-cost_test_2 )
        return c


    def main():
        var = [[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],
               [0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],
               [1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],
               [1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]]
        cost = np.zeros(16)
        for n in range(16):
            cost[n] = process_1_1(var[n][0]) + process_1_2(var[n][1]) \
                + process_2(var[n][0],var[n][1],var[n][2],var[n][3])
        print('是否检测零件 1    是否检测零件 2'  '是否检测成品'  '是否拆解次品'  '
成本期望')
        print('否   否  否  否',cost[0])
        print('否   否  否  是',cost[1])
        print('否   否  是  否',cost[2])
        print('否   否  是  是',cost[3])
        print('否   是  否  否',cost[4])
        print('否   是  否  是',cost[5])
```

```
        print('否    是   是   否',cost[6])
        print('否    是   是   是',cost[7])
        print('是    否   否   否',cost[8])
        print('是    否   否   是',cost[9])
        print('是    否   是   否',cost[10])
        print('是    否   是   是',cost[11])
        print('是    是   否   否',cost[12])
        print('是    是   否   是',cost[13])
        print('是    是   是   否',cost[14])
        print('是    是   是   是',cost[15])

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

代码三

```
import numpy as np
import matplotlib.pyplot as plt

#定义全局变量
p_1 = 0.1   #次品率
p_2 = 0.1
p_3 = 0.1
cost_buy_1 = 4   #零件单价
cost_buy_2 = 18
cost_assemble = 6   #装配成本
cost_test_1 = 2   #检测成本
cost_test_2 = 3
cost_test_3 = 3
cost_swap = 30   #调换损失
cost_disa = 5   #拆解费用
sell_price = 56   #市场售价
#定义初始量

def process_1_1(x1): #未检测过检测
        c = 0
```

```
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1 + p_1*(cost_buy_1 + cost_test_1) + \
                p_1**2*(cost_buy_1 + cost_test_1) + p_1**3*(cost_buy_1 + cost_test_1)
+ \
                p_1**4*(cost_buy_1 + cost_test_1) + p_1**5*(cost_buy_1 + cost_test_1)
        return c
    def process_1_1_1(x1): #已检测过调换后重新检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1
        return c
    def process_1_2(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2 + p_2*(cost_buy_2 + cost_test_2) + \
                p_2**2*(cost_buy_2 + cost_test_2) + p_2**3*(cost_buy_2 + cost_test_2)
+ \
                p_2**4*(cost_buy_2 + cost_test_2) + p_2**5*(cost_buy_2 + cost_test_2)
        return c
    def process_1_2_1(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2
        return c
    def process_2(x1,x2,x3,x4):
        c = 0
        if x3 == 0:
            c = c + cost_assemble
```

```
            if x4 ==0:
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**2*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**3*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**4*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**5*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2+0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+ cost_test_2+0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
```

```
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
- cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
- cost_test_2)
            else:
                c = c + cost_assemble + cost_test_3
                if x4 == 0:
                    if x1 == 0 and x2 == 0:
                        p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    if x1 == 1 and x2 == 0:
                        p = p_2 + p_3 - p_2*p_3
                    if x1 == 0 and x2 == 1:
                        p = p_1 + p_3 - p_1*p_3
                    if x1 == 1 and x2 == 1:
                        p = p_3
                    c = c + p*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**2*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**3*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**4*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**5*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3)
                else: #拆解
                    if x1 == 0 and x2 == 0:
                        p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2 + 0.5*cost_buy_1+0.5*cost_buy_2)
```

```
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+cost_test_2 +0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
-cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
-cost_test_2 )
        return c

    def main():
        var = [[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],
               [0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],
               [1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],
               [1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]]
        cost = np.zeros(16)
        for n in range(16):
            cost[n] = process_1_1(var[n][0]) + process_1_2(var[n][1]) \
                + process_2(var[n][0],var[n][1],var[n][2],var[n][3])
        print('是否检测零件 1      是否检测零件 2'  '是否检测成品'  '是否拆解次品'  '
成本期望')
        print('否    否   否   否',cost[0])
        print('否    否   否   是',cost[1])
        print('否    否   是   否',cost[2])
        print('否    否   是   是',cost[3])
        print('否    是   否   否',cost[4])
```

```
        print('否    是   否   是',cost[5])
        print('否    是   是   否',cost[6])
        print('否    是   是   是',cost[7])
        print('是    否   否   否',cost[8])
        print('是    否   否   是',cost[9])
        print('是    否   是   否',cost[10])
        print('是    否   是   是',cost[11])
        print('是    是   否   否',cost[12])
        print('是    是   否   是',cost[13])
        print('是    是   是   否',cost[14])
        print('是    是   是   是',cost[15])

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

代码四

```
import numpy as np
import matplotlib.pyplot as plt

#定义全局变量
p_1 = 0.2   #次品率
p_2 = 0.2
p_3 = 0.2
cost_buy_1 = 4   #零件单价
cost_buy_2 = 18
cost_assemble = 6   #装配成本
cost_test_1 = 1   #检测成本
cost_test_2 = 1
cost_test_3 = 2
cost_swap = 30   #调换损失
cost_disa = 5   #拆解费用
sell_price = 56   #市场售价
#定义初始量

def process_1_1(x1): #未检测过检测
```

```
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1 + p_1*(cost_buy_1 + cost_test_1) + \
                p_1**2*(cost_buy_1 + cost_test_1) + p_1**3*(cost_buy_1 + cost_test_1)
+ \
                p_1**4*(cost_buy_1 + cost_test_1) + p_1**5*(cost_buy_1 + cost_test_1)
        return c
    def process_1_1_1(x1): #已检测过调换后重新检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1
        return c
    def process_1_2(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2 + p_2*(cost_buy_2 + cost_test_2) + \
                p_2**2*(cost_buy_2 + cost_test_2) + p_2**3*(cost_buy_2 + cost_test_2)
+ \
                p_2**4*(cost_buy_2 + cost_test_2) + p_2**5*(cost_buy_2 + cost_test_2)
        return c
    def process_1_2_1(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2
        return c
    def process_2(x1,x2,x3,x4):
        c = 0
        if x3 == 0:
```

```
            c = c + cost_assemble
            if x4 ==0:
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**2*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**3*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**4*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**5*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2+0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+ cost_test_2+0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
```

```
process_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
- cost_test_2 +0.5*cost_buy_1)
                    if x1 == 1 and x2 == 1:
                        p = p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
- cost_test_2)
            else:
                c = c + cost_assemble + cost_test_3
                if x4 == 0:
                    if x1 == 0 and x2 == 0:
                        p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    if x1 == 1 and x2 == 0:
                        p = p_2 + p_3 - p_2*p_3
                    if x1 == 0 and x2 == 1:
                        p = p_1 + p_3 - p_1*p_3
                    if x1 == 1 and x2 == 1:
                        p = p_3
                    c = c + p*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**2*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**3*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**4*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                            p**5*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3)
                else: #拆解
                    if x1 == 0 and x2 == 0:
                        p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
```

```
+cost_test_2 + 0.5*cost_buy_1+0.5*cost_buy_2)
                    if x1 == 1 and x2 == 0:
                        p = p_2 + p_3 - p_2*p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+cost_test_2 +0.5*cost_buy_2)
                    if x1 == 0 and x2 == 1:
                        p = p_1 + p_3 - p_1*p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
-cost_test_2 +0.5*cost_buy_1)
                    if x1 == 1 and x2 == 1:
                        p = p_3
                        c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
-cost_test_2 )
            return c


    def main():
        var = [[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],
               [0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],
               [1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],
               [1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]]
        cost = np.zeros(16)
        for n in range(16):
            cost[n] = process_1_1(var[n][0]) + process_1_2(var[n][1]) \
                + process_2(var[n][0],var[n][1],var[n][2],var[n][3])
        print('是否检测零件 1      是否检测零件 2'  '是否检测成品'  '是否拆解次品'  '
成本期望')
        print('否    否   否   否',cost[0])
        print('否    否   否   是',cost[1])
        print('否    否   是   否',cost[2])
        print('否    否   是   是',cost[3])
```

```
        print('否    是   否   否',cost[4])
        print('否    是   否   是',cost[5])
        print('否    是   是   否',cost[6])
        print('否    是   是   是',cost[7])
        print('是    否   否   否',cost[8])
        print('是    否   否   是',cost[9])
        print('是    否   是   否',cost[10])
        print('是    否   是   是',cost[11])
        print('是    是   否   否',cost[12])
        print('是    是   否   是',cost[13])
        print('是    是   是   否',cost[14])
        print('是    是   是   是',cost[15])

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

代码五

```
import numpy as np
import matplotlib.pyplot as plt

#定义全局变量
p_1 = 0.1   #次品率
p_2 = 0.2
p_3 = 0.1
cost_buy_1 = 4   #零件单价
cost_buy_2 = 18
cost_assemble = 6   #装配成本
cost_test_1 = 8   #检测成本
cost_test_2 = 1
cost_test_3 = 2
cost_swap = 10   #调换损失
cost_disa = 5   #拆解费用
sell_price = 56   #市场售价
#定义初始量
```

```
    def process_1_1(x1): #未检测过检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1 + p_1*(cost_buy_1 + cost_test_1) + \
                p_1**2*(cost_buy_1 + cost_test_1) + p_1**3*(cost_buy_1 + cost_test_1)
+ \
                p_1**4*(cost_buy_1 + cost_test_1) + p_1**5*(cost_buy_1 + cost_test_1)
        return c
    def process_1_1_1(x1): #已检测过调换后重新检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1
        return c
    def process_1_2(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2 + p_2*(cost_buy_2 + cost_test_2) + \
                p_2**2*(cost_buy_2 + cost_test_2) + p_2**3*(cost_buy_2 + cost_test_2)
+ \
                p_2**4*(cost_buy_2 + cost_test_2) + p_2**5*(cost_buy_2 + cost_test_2)
        return c
    def process_1_2_1(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2
        return c
    def process_2(x1,x2,x3,x4):
        c = 0
```

```
        if x3 == 0:
            c = c + cost_assemble
            if x4 ==0:
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**2*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**3*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**4*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**5*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2+0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+ cost_test_2+0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
```

```
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
- cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
- cost_test_2)
        else:
            c = c + cost_assemble + cost_test_3
            if x4 == 0:
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**2*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**3*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**4*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**5*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2(x2) \
```

```
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2 + 0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+cost_test_2 +0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
-cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
-cost_test_2 )
        return c


    def main():
        var = [[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],
               [0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],
               [1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],
               [1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]]
        cost = np.zeros(16)
        for n in range(16):
            cost[n] = process_1_1(var[n][0]) + process_1_2(var[n][1]) \
                + process_2(var[n][0],var[n][1],var[n][2],var[n][3])
        print('是否检测零件 1      是否检测零件 2'  '是否检测成品'  '是否拆解次品'  '
成本期望')
        print('否    否   否   否',cost[0])
        print('否    否   否   是',cost[1])
        print('否    否   是   否',cost[2])
```

```
        print('否    否   是   是',cost[3])
        print('否    是   否   否',cost[4])
        print('否    是   否   是',cost[5])
        print('否    是   是   否',cost[6])
        print('否    是   是   是',cost[7])
        print('是    否   否   否',cost[8])
        print('是    否   否   是',cost[9])
        print('是    否   是   否',cost[10])
        print('是    否   是   是',cost[11])
        print('是    是   否   否',cost[12])
        print('是    是   否   是',cost[13])
        print('是    是   是   否',cost[14])
        print('是    是   是   是',cost[15])

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

代码六

```
import numpy as np
import matplotlib.pyplot as plt

#定义全局变量
p_1 = 0.05   #次品率
p_2 = 0.05
p_3 = 0.05
cost_buy_1 = 4   #零件单价
cost_buy_2 = 18
cost_assemble = 6   #装配成本
cost_test_1 = 2   #检测成本
cost_test_2 = 3
cost_test_3 = 3
cost_swap = 10   #调换损失
cost_disa = 40   #拆解费用
sell_price = 56   #市场售价
#定义初始量
```

```
    def process_1_1(x1): #未检测过检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1 + p_1*(cost_buy_1 + cost_test_1) + \
                p_1**2*(cost_buy_1 + cost_test_1) + p_1**3*(cost_buy_1 + cost_test_1)
+ \
                p_1**4*(cost_buy_1 + cost_test_1) + p_1**5*(cost_buy_1 + cost_test_1)
        return c
    def process_1_1_1(x1): #已检测过调换后重新检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1
        return c
    def process_1_2(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2 + p_2*(cost_buy_2 + cost_test_2) + \
                p_2**2*(cost_buy_2 + cost_test_2) + p_2**3*(cost_buy_2 + cost_test_2)
+ \
                p_2**4*(cost_buy_2 + cost_test_2) + p_2**5*(cost_buy_2 + cost_test_2)
        return c
    def process_1_2_1(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2
        return c
    def process_2(x1,x2,x3,x4):
```

```
        c = 0
        if x3 == 0:
            c = c + cost_assemble
            if x4 ==0:
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**2*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**3*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**4*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**5*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2+0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+ cost_test_2+0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
```

```
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
- cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
- cost_test_2)
        else:
            c = c + cost_assemble + cost_test_3
            if x4 == 0:
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**2*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**3*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**4*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**5*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
```

```
process_1_1(x1) + process_1_2(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2 + 0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+cost_test_2 +0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
-cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
-cost_test_2 )
        return c


    def main():
        var = [[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],
               [0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],
               [1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],
               [1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]]
        cost = np.zeros(16)
        for n in range(16):
            cost[n] = process_1_1(var[n][0]) + process_1_2(var[n][1]) \
                + process_2(var[n][0],var[n][1],var[n][2],var[n][3])
        print('是否检测零件 1      是否检测零件 2'  '是否检测成品'  '是否拆解次品'  '
成本期望')
        print('否    否   否   否',cost[0])
        print('否    否   否   是',cost[1])
```

```
        print('否    否   是   否',cost[2])
        print('否    否   是   是',cost[3])
        print('否    是   否   否',cost[4])
        print('否    是   否   是',cost[5])
        print('否    是   是   否',cost[6])
        print('否    是   是   是',cost[7])
        print('是    否   否   否',cost[8])
        print('是    否   否   是',cost[9])
        print('是    否   是   否',cost[10])
        print('是    否   是   是',cost[11])
        print('是    是   否   否',cost[12])
        print('是    是   否   是',cost[13])
        print('是    是   是   否',cost[14])
        print('是    是   是   是',cost[15])

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

代码七

```
import numpy as np
import matplotlib.pyplot as plt

#定义全局变量
p_1 = 0.1   #次品率
p_2 = 0.1
p_3 = 0.1
cost_buy_1 = 4   #零件单价
cost_buy_2 = 18
cost_assemble = 6   #装配成本
cost_test_1 = 3   #检测成本
cost_test_2 = 3
cost_test_3 = 3
cost_swap = 6   #调换损失
cost_disa = 5   #拆解费用
sell_price = 56   #市场售价
```

```
    def process_1_1(x1): #未检测过检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1 + p_1*(cost_buy_1 + cost_test_1) + \
                p_1**2*(cost_buy_1 + cost_test_1) + p_1**3*(cost_buy_1 + cost_test_1)
+ \
                p_1**4*(cost_buy_1 + cost_test_1) + p_1**5*(cost_buy_1 + cost_test_1)
        return c
    def process_1_1_1(x1): #已检测过调换后重新检测
        c = 0
        if x1 == 0:
            c = c + cost_buy_1
        else:
            c = c + cost_buy_1 + cost_test_1
        return c
    def process_1_2(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2 + p_2*(cost_buy_2 + cost_test_2) + \
                p_2**2*(cost_buy_2 + cost_test_2) + p_2**3*(cost_buy_2 + cost_test_2)
+ \
                p_2**4*(cost_buy_2 + cost_test_2) + p_2**5*(cost_buy_2 + cost_test_2)
        return c
    def process_1_2_1(x2):
        c = 0
        if x2 == 0:
            c = c + cost_buy_2
        else:
            c = c + cost_buy_2 + cost_test_2
        return c
    def process_2(x1,x2,x3,x4):
```

```
        c = 0
        if x3 == 0:
            c = c + cost_assemble
            if x4 ==0:
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**2*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**3*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**4*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble) + \
                        p**5*(cost_swap + process_1_1(x1) + process_1_2(x2) +
cost_assemble)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2+0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+ cost_test_2+0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
```

```
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
- cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                            -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
- cost_test_2)
        else:
            c = c + cost_assemble + cost_test_3
            if x4 == 0:
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                if x1 == 1 and x2 == 1:
                    p = p_3
                c = c + p*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**2*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**3*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**4*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3) + \
                        p**5*(process_1_1(x1) + process_1_2(x2) + cost_assemble +
cost_test_3)
            else: #拆解
                if x1 == 0 and x2 == 0:
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
```

```
process_1_1(x1) + process_1_2(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
+cost_test_2 + 0.5*cost_buy_1+0.5*cost_buy_2)
                if x1 == 1 and x2 == 0:
                    p = p_2 + p_3 - p_2*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
+cost_test_2 +0.5*cost_buy_2)
                if x1 == 0 and x2 == 1:
                    p = p_1 + p_3 - p_1*p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble + cost_test_1
-cost_test_2 +0.5*cost_buy_1)
                if x1 == 1 and x2 == 1:
                    p = p_3
                    c = c + (p+p**2+p**3+p**4+p**5)*(cost_test_3 + cost_disa +
process_1_1_1(x1) + process_1_2_1(x2) \
                                    -cost_buy_1 - cost_buy_2 + cost_assemble - cost_test_1
-cost_test_2 )
        return c


    def main():
        var = [[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],
               [0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],
               [1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],
               [1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]]
        cost = np.zeros(16)
        for n in range(16):
            cost[n] = process_1_1(var[n][0]) + process_1_2(var[n][1]) \
                + process_2(var[n][0],var[n][1],var[n][2],var[n][3])
        print('是否检测零件 1      是否检测零件 2'  '是否检测成品'  '是否拆解次品'  '
成本期望')
        print('否    否   否   否',cost[0])
        print('否    否   否   是',cost[1])
```

```
        print('否    否   是   否',cost[2])
        print('否    否   是   是',cost[3])
        print('否    是   否   否',cost[4])
        print('否    是   否   是',cost[5])
        print('否    是   是   否',cost[6])
        print('否    是   是   是',cost[7])
        print('是    否   否   否',cost[8])
        print('是    否   否   是',cost[9])
        print('是    否   是   否',cost[10])
        print('是    否   是   是',cost[11])
        print('是    是   否   否',cost[12])
        print('是    是   否   是',cost[13])
        print('是    是   是   否',cost[14])
        print('是    是   是   是',cost[15])

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

8.3 问题三的相关代码

代码一

```
import numpy as np
import matplotlib.pyplot as plt
import itertools

#定义全局变量
p_part = np.zeros(8)
p_h = np.zeros(3)
cost_part = np.zeros(8)
cost_h = np.zeros(3)
cost_test_part = np.zeros(8)
cost_test_h = np.zeros(3)
cost_disa_h = np.zeros(3)
p_part = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]    #次品率
p_h = [0.1,0.1,0.1]
```

```
p_f = 0.1
cost_part = [2,8,12,2,8,12,8,12]   #零件单价
cost_h = [8,8,8]
cost_f = 8
cost_test_part = [1,1,2,1,1,2,1,2]
cost_test_h = [4,4,4]
cost_test_f = 6
cost_disa_h = [6,6,6]
cost_swap = 40   #调换损失
cost_disa_f = 10   #拆解费用
sell_price = 200   #市场售价
# x 表示存放零件是否检测的数组
# x_h 表示存放半成品是否检测的数组
# x_f 表示是否检测成品
# x_h_disa 表示存放是否拆解不合格半成品的数组
# x_f_disa 表示是否拆解成品
def process_part_1(n,x): #未检测过检测
    c = 0
    if x[n] == 0:
        c = c + cost_part[n]
    else:
        c = c + cost_part[n] + cost_test_part[n] +
(p_part[n]+p_part[n]**2+p_part[n]**3+p_part[n]**4+p_part[n]**5)*\
            (cost_part[n] + cost_test_part[n])
    return c
def process_part_2(n,x): #已检测过重新检测
    c = 0
    if x[n] == 0:
        c = c + cost_part[n]
    else:
        c = c + cost_part[n] + cost_test_part[n]
    return c

def process_h_1_1(x,x_h,x_h_disa):
    c = 0
    if x_h == 0:
```

```
            c = c + cost_h[0]
        else:
            if x_h_disa == 0:
                if x[0] == 0 and x[1] == 0 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1])*(1-p_part[2]))
                if x[0] == 0 and x[1] == 0 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1]))
                if x[0] == 0 and x[1] == 1 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[2]))
                if x[0] == 0 and x[1] == 1 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0]))
                if x[0] == 1 and x[1] == 0 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[2])*(1-p_part[1]))
                if x[0] == 1 and x[1] == 0 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[1]))
                if x[0] == 1 and x[1] == 1 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[2]))
                if x[0] == 1 and x[1] == 1 and x[2] == 1:
                    p = p_h[0]
                c = c + cost_h[0] + cost_test_h[0] + (p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + \
                    process_part_1(1,x) +process_part_1(2,x)+cost_h[0] + cost_test_h[0] )
            else: #拆解
                if x[0] == 0 and x[1] == 0 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1])*(1-p_part[2]))
                    c = c + cost_h[0] + cost_test_h[0] + (p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + process_part_1(1,x) +\
                    process_part_1(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0] - cost_part[0] - cost_part[1] - cost_part[2]+\

cost_test_part[0]+cost_test_part[1]+cost_test_part[2]+0.5*cost_part[0]+0.5*cost_part[1]+0.5*cost_part[2])
                if x[0] == 0 and x[1] == 0 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1]))
                    c = c + cost_h[0] + cost_test_h[0] +
```

```
(p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + process_part_1(1,x) +\
                        process_part_2(2,x)+cost_h[0]    +    cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]+\

cost_test_part[0]+cost_test_part[1]-cost_test_part[2]+0.5*cost_part[0]+0.5*cost_part[1])
                    if x[0] == 0 and x[1] == 1 and x[2] == 0:
                        p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[2]))
                        c       =       c       +       cost_h[0]       +       cost_test_h[0]       +
(p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + process_part_2(1,x) +\
                        process_part_1(2,x)+cost_h[0]    +    cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]+\

cost_test_part[0]-cost_test_part[1]+cost_test_part[2]+0.5*cost_part[0]+0.5*cost_part[2])
                    if x[0] == 0 and x[1] == 1 and x[2] == 1:
                        p = 1-((1-p_h[0])*(1-p_part[0]))
                        c       =       c       +       cost_h[0]       +       cost_test_h[0]       +
(p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + process_part_2(1,x) +\
                        process_part_2(2,x)+cost_h[0]    +    cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]+\

cost_test_part[0]-cost_test_part[1]-cost_test_part[2]+0.5*cost_part[0])
                    if x[0] == 1 and x[1] == 0 and x[2] == 0:
                        p = 1-((1-p_h[0])*(1-p_part[1])*(1-p_part[2]))
                        c       =       c       +       cost_h[0]       +       cost_test_h[0]       +
(p+p**2+p**3+p**4+p**5)*(process_part_2(0,x) + process_part_1(1,x) +\
                        process_part_1(2,x)+cost_h[0]    +    cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]-\

cost_test_part[0]+cost_test_part[1]+cost_test_part[2]+0.5*cost_part[2]+0.5*cost_part[1])
                    if x[0] == 1 and x[1] == 0 and x[2] == 1:
                        p = 1-((1-p_h[0])*(1-p_part[1]))
                        c       =       c       +       cost_h[0]       +       cost_test_h[0]       +
(p+p**2+p**3+p**4+p**5)*(process_part_2(0,x) + process_part_1(1,x) +\
                        process_part_2(2,x)+cost_h[0]    +    cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]-\
```

```
cost_test_part[0]+cost_test_part[1]-cost_test_part[2]+0.5*cost_part[1])
                    if x[0] == 1 and x[1] == 1 and x[2] == 0:
                        p = 1-((1-p_h[0])*(1-p_part[2]))
                        c = c + cost_h[0] + cost_test_h[0] + (p+p**2+p**3+p**4+p**5)*(process_part_2(0,x) + process_part_2(1,x) +\
                        process_part_1(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[0] - cost_part[1] - cost_part[2]-\

cost_test_part[0]-cost_test_part[1]+cost_test_part[2]+0.5*cost_part[2])
                    if x[0] == 1 and x[1] == 1 and x[2] == 1:
                        p = p_h[0]
                        c = c + cost_h[0] + cost_test_h[0] + (p+p**2+p**3+p**4+p**5)*(process_part_2(0,x) + process_part_2(1,x) +\
                        process_part_2(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[0] - cost_part[1] - cost_part[2]-\
                        cost_test_part[0]-cost_test_part[1]-cost_test_part[2])
            return c

    def main():
        var = list(itertools.product([0, 1], repeat=5))
        cost = np.zeros(32)
        for n in range(32):
            cost[n] = process_part_1(0,var[n][:]) + process_part_1(1,var[n][:])+ process_part_1(2,var[n][:]) \
                    + process_h_1_1(var[n][:],var[n][4],var[n][3])
            print(var[n],cost[n])

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

代码二

```
import numpy as np
import matplotlib.pyplot as plt
import itertools
```

```
#定义全局变量
p_part = np.zeros(8)
p_h = np.zeros(3)
cost_part = np.zeros(8)
cost_h = np.zeros(3)
cost_test_part = np.zeros(8)
cost_test_h = np.zeros(3)
cost_disa_h = np.zeros(3)
p_part = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]  #次品率
p_h = [0.1,0.1,0.1]
p_f = 0.1
cost_part = [2,8,12,2,8,12,8,12]  #零件单价
cost_h = [8,8,8]
cost_f = 8
cost_test_part = [1,1,2,1,1,2,1,2]
cost_test_h = [4,4,4]
cost_test_f = 6
cost_disa_h = [6,6,6]
cost_swap = 40  #调换损失
cost_disa_f = 10  #拆解费用
sell_price = 200  #市场售价
# x 表示存放零件是否检测的数组
# x_h 表示存放半成品是否检测的数组
# x_f 表示是否检测成品
# x_h_disa 表示存放是否拆解不合格半成品的数组
# x_f_disa 表示是否拆解成品
def process_part_1(n,x): #未检测过检测
    c = 0
    if x[n] == 0:
        c = c + cost_part[n]
    else:
        c = c + cost_part[n] + cost_test_part[n] + p_part[n]*(cost_part[n] + cost_test_part[n]) + \
        p_part[n]**2*(cost_part[n] + cost_test_part[n]) + p_part[n]**3*(cost_part[n] + cost_test_part[n]) + \
        p_part[n]**4*(cost_part[n] + cost_test_part[n]) + p_part[n]**5*(cost_part[n] +
```

```
cost_test_part[n])
            return c
        def process_part_2(n,x): #已检测过重新检测
            c = 0
            if x[n] == 0:
                c = c + cost_part[n]
            else:
                c = c + cost_part[n] + cost_test_part[n]
            return c


        def process_h_1_1(x,x_h,x_h_disa):
            c = 0
            if x_h == 0:
                if x[0] == 0 and x[1] == 0 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1])*(1-p_part[2]))
                    c = c + cost_h[0] + (p+p**2+p**3+p**4+p**5)*(cost_f + 0.5*30 + 0.5\
                    *cost_test_f + 0.5*cost_swap+0.5*cost_disa_f) +
(p_f+p_f**2+p_f**3+p_f**4+p_f**5)*30
                if x[0] == 0 and x[1] == 0 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1]))
                    c = c + cost_h[0] + (p+p**2+p**3+p**4+p**5)*(cost_f + 0.5*30 + 0.5\
                    *cost_test_f + 0.5*cost_swap+0.5*cost_disa_f) +
(p_f+p_f**2+p_f**3+p_f**4+p_f**5)*33.55554
                if x[0] == 0 and x[1] == 1 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[2]))
                    c = c + cost_h[0] + (p+p**2+p**3+p**4+p**5)*(cost_f + 0.5*30 + 0.5\
                    *cost_test_f + 0.5*cost_swap+0.5*cost_disa_f) +
(p_f+p_f**2+p_f**3+p_f**4+p_f**5)*31.99999
                if x[0] == 0 and x[1] == 1 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0]))
                    c = c + cost_h[0] + (p+p**2+p**3+p**4+p**5)*(cost_f + 0.5*30 + 0.5\
                    *cost_test_f + 0.5*cost_swap+0.5*cost_disa_f) +
(p_f+p_f**2+p_f**3+p_f**4+p_f**5)*35.55553
                if x[0] == 1 and x[1] == 0 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[2])*(1-p_part[1]))
                    c = c + cost_h[0] + (p+p**2+p**3+p**4+p**5)*(cost_f + 0.5*30 + 0.5\
```

```
                *cost_test_f + 0.5*cost_swap+0.5*cost_disa_f) +
(p_f+p_f**2+p_f**3+p_f**4+p_f**5)*31.33333
            if x[0] == 1 and x[1] == 0 and x[2] == 1:
                p = 1-((1-p_h[0])*(1-p_part[1]))
                c = c + cost_h[0] + (p+p**2+p**3+p**4+p**5)*(cost_f + 0.5*30 + 0.5\
                *cost_test_f + 0.5*cost_swap+0.5*cost_disa_f) +
(p_f+p_f**2+p_f**3+p_f**4+p_f**5)*34.88887
            if x[0] == 1 and x[1] == 1 and x[2] == 0:
                p = 1-((1-p_h[0])*(1-p_part[2]))
                c = c + cost_h[0] + (p+p**2+p**3+p**4+p**5)*(cost_f + 0.5*30 + 0.5\
                *cost_test_f + 0.5*cost_swap+0.5*cost_disa_f) +
(p_f+p_f**2+p_f**3+p_f**4+p_f**5)*33.33332
            if x[0] == 1 and x[1] == 1 and x[2] == 1:
                p = p_h[0]
                c = c + cost_h[0] + (p+p**2+p**3+p**4+p**5)*(cost_f + 0.5*30 + 0.5\
                *cost_test_f + 0.5*cost_swap+0.5*cost_disa_f) +
(p_f+p_f**2+p_f**3+p_f**4+p_f**5)*36.88886
        else:
            if x_h_disa == 0:
                if x[0] == 0 and x[1] == 0 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1])*(1-p_part[2]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
51.7356447911881)
                if x[0] == 0 and x[1] == 0 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
51.4961095708501)
                if x[0] == 0 and x[1] == 1 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[2]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
49.3631413525011)
                if x[0] == 0 and x[1] == 1 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
48.831690216349)
                if x[0] == 1 and x[1] == 0 and x[2] == 0:
```

```
                    p = 1-((1-p_h[0])*(1-p_part[2])*(1-p_part[1]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
48.4490179926318)
                if x[0] == 1 and x[1] == 0 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[1]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
48.0086918998145)
                if x[0] == 1 and x[1] == 1 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[2]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
46.088350149469)
                if x[0] == 1 and x[1] == 1 and x[2] == 1:
                    p = p_h[0]
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
45.4320212346)
                c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(0,x)\
                     + process_part_1(1,x) +process_part_1(2,x)+cost_h[0] +
cost_test_h[0])
            else: #拆解
                if x[0] == 0 and x[1] == 0 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1])*(1-p_part[2]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + process_part_1(1,x) +\
                    process_part_1(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0] -
cost_part[0] - cost_part[1] - cost_part[2]+\

cost_test_part[0]+cost_test_part[1]+cost_test_part[2]+0.5*cost_part[0]+0.5*cost_part[1]+0.5
*cost_part[2])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
51.2140081796826)
                if x[0] == 0 and x[1] == 0 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[1]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + process_part_1(1,x) +\
                    process_part_2(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-
```

```
cost_part[0] - cost_part[1] - cost_part[2]+\

cost_test_part[0]+cost_test_part[1]-cost_test_part[2]+0.5*cost_part[0]+0.5*cost_part[1])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
46.8355087947837)
                if x[0] == 0 and x[1] == 1 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[0])*(1-p_part[2]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + process_part_2(1,x) +\
                    process_part_1(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]+\

cost_test_part[0]-cost_test_part[1]+cost_test_part[2]+0.5*cost_part[0]+0.5*cost_part[2])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
46.3935550501578)
                if x[0] == 0 and x[1] == 1 and x[2] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[0]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(0,x) + process_part_2(1,x) +\
                    process_part_2(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]+\

cost_test_part[0]-cost_test_part[1]-cost_test_part[2]+0.5*cost_part[0])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
44.245726398)
                if x[0] == 1 and x[1] == 0 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[1])*(1-p_part[2]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_2(0,x) + process_part_1(1,x) +\
                    process_part_1(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]-\

cost_test_part[0]+cost_test_part[1]+cost_test_part[2]+0.5*cost_part[2]+0.5*cost_part[1])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
46.8404913055318)
                if x[0] == 1 and x[1] == 0 and x[2] == 1:
```

```
                    p = 1-((1-p_h[0])*(1-p_part[1]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_2(0,x) + process_part_1(1,x) +\
                    process_part_2(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]-\

cost_test_part[0]+cost_test_part[1]-cost_test_part[2]+0.5*cost_part[1])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
44.2825958577)
                if x[0] == 1 and x[1] == 1 and x[2] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[2]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_2(0,x) + process_part_2(1,x) +\
                    process_part_1(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]-\

cost_test_part[0]-cost_test_part[1]+cost_test_part[2]+0.5*cost_part[2])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
43.4305753174)
                if x[0] == 1 and x[1] == 1 and x[2] == 1:
                    p = p_h[0]
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_2(0,x) + process_part_2(1,x) +\
                    process_part_2(2,x)+cost_h[0] + cost_test_h[0]+cost_disa_h[0]-
cost_part[0] - cost_part[1] - cost_part[2]-\
                    cost_test_part[0]-cost_test_part[1]-cost_test_part[2])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
42.88884)
            return c


        def main():
            var = list(itertools.product([0, 1], repeat=5))
            cost = np.zeros(32)
            for n in range(32):
                cost[n] = process_part_1(0,var[n][:]) + process_part_1(1,var[n][:])+
process_part_1(2,var[n][:]) \
```

```
                    + process_h_1_1(var[n][:],var[n][4],var[n][3])
            print(var[n],cost[n])

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

代码三

```
import numpy as np
import matplotlib.pyplot as plt
import itertools

#定义全局变量
p_part = np.zeros(8)
p_h = np.zeros(3)
cost_part = np.zeros(8)
cost_h = np.zeros(3)
cost_test_part = np.zeros(8)
cost_test_h = np.zeros(3)
cost_disa_h = np.zeros(3)
p_part = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]   #次品率
p_h = [0.1,0.1,0.1]
p_f = 0.1
cost_part = [2,8,12,2,8,12,8,12]   #零件单价
cost_h = [8,8,8]
cost_f = 8
cost_test_part = [1,1,2,1,1,2,1,2]
cost_test_h = [4,4,4]
cost_test_f = 6
cost_disa_h = [6,6,6]
cost_swap = 40   #调换损失
cost_disa_f = 10   #拆解费用
sell_price = 200   #市场售价
# x 表示存放零件是否检测的数组
# x_h 表示存放半成品是否检测的数组
# x_f 表示是否检测成品
```

```
    # x_h_disa 表示存放是否拆解不合格半成品的数组
    # x_f_disa 表示是否拆解成品
    def process_part_1(n,x): #未检测过检测
        c = 0
        if x[n] == 0:
            c = c + cost_part[n]
        else:
            c = c + cost_part[n] + cost_test_part[n] + p_part[n]*(cost_part[n] + cost_test_part[n]) + \
            p_part[n]**2*(cost_part[n] + cost_test_part[n]) + p_part[n]**3*(cost_part[n] + cost_test_part[n]) + \
            p_part[n]**4*(cost_part[n] + cost_test_part[n]) + p_part[n]**5*(cost_part[n] + cost_test_part[n])
        return c
    def process_part_2(n,x): #已检测过重新检测
        c = 0
        if x[n] == 0:
            c = c + cost_part[n]
        else:
            c = c + cost_part[n] + cost_test_part[n]
        return c

    def process_h_1_1(x,x_h,x_h_disa):
        c = 0
        if x_h == 0:
            c = c + cost_h[0]
        else:
            if x_h_disa == 0:
                if x[0] == 0 and x[1] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[1])*(1-p_part[2]))
                if x[0] == 0 and x[1] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[1]))
                if x[0] == 1 and x[1] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[2]))
                if x[0] == 1 and x[1] == 1:
                    p = p_h[0]
```

```
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(1,x) +\
                                        process_part_1(2,x)+cost_h[0] + cost_test_h[0])
                else: #拆解
                    if x[0] == 0 and x[1] == 0:
                        p = 1-((1-p_h[0])*(1-p_part[1])*(1-p_part[2]))
                        c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(1,x) + process_part_1(2,x) +\
                        cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[1]-cost_part[2]\

+cost_test_part[1]+cost_test_part[2]+0.5*cost_part[1]+0.5*cost_part[2])
                    if x[0] == 0 and x[1] == 1:
                        p = 1-((1-p_h[0])*(1-p_part[1]))
                        c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(1,x) + process_part_2(2,x) +\
                        cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[1]-cost_part[2]\
                            +cost_test_part[1]-cost_test_part[2]+0.5*cost_part[1])
                    if x[0] == 1 and x[1] == 0:
                        p = 1-((1-p_h[0])*(1-p_part[2]))
                        c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_2(1,x) + process_part_1(2,x) +\
                        cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[1]-cost_part[2]\
                            -cost_test_part[1]+cost_test_part[2]+0.5*cost_part[1])
                    if x[0] == 1 and x[1] == 1:
                        p = p_h[0]
                        c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_2(1,x) + process_part_2(2,x) +\
                        cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[1]-cost_part[2]\
                            -cost_test_part[1]-cost_test_part[2])
            return c

        def main():
            var = list(itertools.product([0, 1], repeat=4))
            cost = np.zeros(16)
            for n in range(16):
                cost[n] = process_part_1(0,var[n][:]) + process_part_1(1,var[n][:])\
```

```
                + process_h_1_1(var[n][:],var[n][3],var[n][2])
        print(var[n],cost[n])

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

代码四

```
import numpy as np
import matplotlib.pyplot as plt
import itertools

#定义全局变量
p_part = np.zeros(8)
p_h = np.zeros(3)
cost_part = np.zeros(8)
cost_h = np.zeros(3)
cost_test_part = np.zeros(8)
cost_test_h = np.zeros(3)
cost_disa_h = np.zeros(3)
p_part = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]   #次品率
p_h = [0.1,0.1,0.1]
p_f = 0.1
cost_part = [2,8,12,2,8,12,8,12]   #零件单价
cost_h = [8,8,8]
cost_f = 8
cost_test_part = [1,1,2,1,1,2,1,2]
cost_test_h = [4,4,4]
cost_test_f = 6
cost_disa_h = [6,6,6]
cost_swap = 40   #调换损失
cost_disa_f = 10   #拆解费用
sell_price = 200   #市场售价
# x 表示存放零件是否检测的数组
# x_h 表示存放半成品是否检测的数组
# x_f 表示是否检测成品
```

```
    # x_h_disa 表示存放是否拆解不合格半成品的数组
    # x_f_disa 表示是否拆解成品
    def process_part_1(n,x): #未检测过检测
        c = 0
        if x[n] == 0:
            c = c + cost_part[n]
        else:
            c = c + cost_part[n] + cost_test_part[n] + p_part[n]*(cost_part[n] + cost_test_part[n]) + \
            p_part[n]**2*(cost_part[n] + cost_test_part[n]) + p_part[n]**3*(cost_part[n] + cost_test_part[n]) + \
            p_part[n]**4*(cost_part[n] + cost_test_part[n]) + p_part[n]**5*(cost_part[n] + cost_test_part[n])
        return c
    def process_part_2(n,x): #已检测过重新检测
        c = 0
        if x[n] == 0:
            c = c + cost_part[n]
        else:
            c = c + cost_part[n] + cost_test_part[n]
        return c


    def process_h_1_1(x,x_h,x_h_disa):
        c = 0
        if x_h == 0:
            if x[0] == 0 and x[1] == 0:
                p = 1-((1-p_h[0])*(1-p_part[1])*(1-p_part[2]))
                c = c + cost_h[0] + p*(cost_f + cost_h[0] + 18 + 0.5*cost_test_f + 0.5*cost_swap) + \
                + p**2*(cost_f + cost_h[0] + 18 + 0.5*cost_test_f + 0.5*cost_swap) +\
                + p**3*(cost_f + cost_h[0] + 18 + 0.5*cost_test_f + 0.5*cost_swap)
            if x[0] == 0 and x[1] == 1:
                p = 1-((1-p_h[0])*(1-p_part[1]))
                c = c + cost_h[0] + p*(cost_f + cost_h[0] + 20 + 0.5*cost_test_f + 0.5*cost_swap) + \
                + p**2*(cost_f + cost_h[0] + 20 + 0.5*cost_test_f + 0.5*cost_swap) +\
```

```
                + p**3*(cost_f + cost_h[0] + 20 + 0.5*cost_test_f + 0.5*cost_swap)
            if x[0] == 1 and x[1] == 0:
                p = 1-((1-p_h[0])*(1-p_part[2]))
                c = c + cost_h[0] + p*(cost_f + cost_h[0] + 19.33333 + 0.5*cost_test_f +
0.5*cost_swap) + \
                + p**2*(cost_f + cost_h[0] + 19.33333 + 0.5*cost_test_f +
0.5*cost_swap) +\
                + p**3*(cost_f + cost_h[0] + 19.33333 + 0.5*cost_test_f +
0.5*cost_swap)
            if x[0] == 1 and x[1] == 1:
                p = p_h[0]
                c = c + cost_h[0] + p*(cost_f + cost_h[0] + 21.33332 + 0.5*cost_test_f +
0.5*cost_swap) + \
                + p**2*(cost_f + cost_h[0] + 21.33332 + 0.5*cost_test_f +
0.5*cost_swap) +\
                + p**3*(cost_f + cost_h[0] + 21.33332 + 0.5*cost_test_f +
0.5*cost_swap)
        else:
            if x_h_disa == 0:
                if x[0] == 0 and x[1] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[1])*(1-p_part[2]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
33.8783600573232)
                if x[0] == 0 and x[1] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[1]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
31.9733215315018)
                if x[0] == 1 and x[1] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[2]))
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
30.8376442368)
                if x[0] == 1 and x[1] == 1:
                    p = p_h[0]
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
29.1110588889)
                c = c + cost_h[0] + cost_test_h[0] + p*(process_part_1(1,x) +
```

```
process_part_1(2,x) +\
                    cost_h[0] + cost_test_h[0]) +p**2*(process_part_1(1,x) +
process_part_1(2,x) +\
                    cost_h[0] + cost_test_h[0]) +p**3*(process_part_1(1,x) +
process_part_1(2,x) +\
                    cost_h[0] + cost_test_h[0])
            else: #拆解
                if x[0] == 0 and x[1] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[1])*(1-p_part[2]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(1,x) + process_part_1(2,x) +\
                    cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[1]-cost_part[2]\

+cost_test_part[1]+cost_test_part[2]+0.5*cost_part[1]+0.5*cost_part[2])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
34.826973315476)
                if x[0] == 0 and x[1] == 1:
                    p = 1-((1-p_h[0])*(1-p_part[1]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_1(1,x) + process_part_2(2,x) +\
                    cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[1]-cost_part[2]\
                        +cost_test_part[1]-cost_test_part[2]+0.5*cost_part[1])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
29.8627331524018)
                if x[0] == 1 and x[1] == 0:
                    p = 1-((1-p_h[0])*(1-p_part[2]))
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_2(1,x) + process_part_1(2,x) +\
                    cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[1]-cost_part[2]\
                        -cost_test_part[1]+cost_test_part[2]+0.5*cost_part[1])
                    c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
29.5608649027472)
                if x[0] == 1 and x[1] == 1:
                    p = p_h[0]
                    c = c + cost_h[0] + cost_test_h[0] +
(p+p**2+p**3+p**4+p**5)*(process_part_2(1,x) + process_part_2(2,x) +\
```

```
                cost_h[0] + cost_test_h[0]+cost_disa_h[0]-cost_part[1]-cost_part[2]\
                    -cost_test_part[1]-cost_test_part[2])
                c = c + (p_f+p_f**2+p_f**3+p_f**4+p_f**5)*(cost_test_h[0] +
27.3333)
        return c

    def main():
        var = list(itertools.product([0, 1], repeat=4))
        cost = np.zeros(16)
        for n in range(16):
            cost[n] = process_part_1(0,var[n][:]) + process_part_1(1,var[n][:])\
                + process_h_1_1(var[n][:],var[n][3],var[n][2])
            print(var[n],cost[n])

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

代码五

```
import numpy as np
import matplotlib.pyplot as plt
import itertools

#定义全局变量
cost_h = np.zeros(3)
p_h = 0.1
p_f = 0.1
cost_h = [36.88886,36.88886,21.33332]
cost_f = 8
cost_assemble_h = 8
cost_test_f = 6
cost_test_h = 4
cost_swap = 40   #调换损失
cost_disa_f = 10   #拆解费用
cost_disa_h = 6
sell_price = 200   #市场售价
```

```
#前面步骤所得到的最后决策为 1 1 1 1 1 1 1 1 0 0 0
# x 表示存放决策变量的数组 ：成品检测，成品拆解
# x_h 表示存放半成品是否拆解
def process_f(x_f_1,x_f_2,x_h_1,x_h_2,x_h_3):
    c = 0
    p = 1-((1-p_h)*(1-p_h)*(1-p_h)*(1-p_f))
    if x_f_1 == 0:
        if x_f_2 == 0:
            c = c + cost_f + cost_h[0] + cost_h[1] + cost_h[2]
            c = c + (p+p**2+p**3+p**4+p**5)*(cost_swap + cost_f + cost_h[0] +
cost_h[1] + cost_h[2])
        else: #拆解
            c = c + cost_f + cost_h[0] + cost_h[1] + cost_h[2]
            c = c + (p+p**2+p**3+p**4+p**5)*(cost_disa_f + cost_swap + cost_f +
cost_test_h*3)
            c = c + (p_h/p)*((cost_disa_h+cost_assemble_h)*x_h_1+(1-x_h_1)*(cost_h[0]))+\

(p_h/p)*((cost_disa_h+cost_assemble_h)*x_h_2+(1-x_h_2)*(cost_h[1]))+\

(p_h/p)*((cost_disa_h+cost_assemble_h)*x_h_3+(1-x_h_3)*(cost_h[2]))
    else: #检测
        if x_f_2 == 0:
            c = c + cost_f + cost_test_f + cost_h[0] + cost_h[1] + cost_h[2]
            c = c + (p+p**2+p**3+p**4+p**5)*(cost_h[0]+cost_h[1]+cost_h[2] +
cost_f + cost_test_f)
        else: #拆解
            c = c + cost_f + cost_test_f + cost_h[0] + cost_h[1] + cost_h[2]
            c = c + (p+p**2+p**3+p**4+p**5)*(cost_disa_f + cost_f + cost_test_f +
cost_test_h*3)
            c = c + (p_h/p)*((cost_disa_h+cost_assemble_h)*x_h_1+(1-x_h_1)*(cost_h[0]))+\

(p_h/p)*((cost_disa_h+cost_assemble_h)*x_h_2+(1-x_h_2)*(cost_h[1]))+\

(p_h/p)*((cost_disa_h+cost_assemble_h)*x_h_3+(1-x_h_3)*(cost_h[2]))
```

```
    return c
def main():
    var = list(itertools.product([0, 1], repeat=5))
    cost = np.zeros(32)
    for n in range(32):
        cost[n] = process_f(var[n][0],var[n][1],var[n][2],var[n][3],var[n][4])
        print(var[n],cost[n])

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

8.4 问题四相关代码

代码一

```
clear;clc
% 定义全局变量
global p_1;
global p_2;
global p_3;
global cost_buy_2;
global cost_buy_1;
global cost_assemble;
global cost_test_1;
global cost_test_2;
global cost_test_3;
global cost_swap;
global cost_disa;
p_1 = 0.05;  % 统计方法认为的次品率
p_2 = 0.05;  %题目标准率
p_3 = 0.05;
cost_buy_1 = 4;  % 零件单价
cost_buy_2 = 18;
cost_assemble = 6;  % 装配成本
cost_test_1 = 2;  % 检测成本
cost_test_2 = 3;
```

```
cost_test_3 = 3;  % 注意：虽然定义了 cost_test_3，但在提供的函数中未使用
cost_swap = 10;  % 调换损失
cost_disa = 40;  % 拆解费用，注意：在提供的函数中未使用
sell_price = 56;  % 市场售价
   predict_p1=p_1;
   predict_p2=p_2;
   predict_p3=p_3;
% 定义函数


    var = [0 0 0 0; 0 0 0 1; 0 0 1 0; 0 0 1 1;
           0 1 0 0; 0 1 0 1; 0 1 1 0; 0 1 1 1;
           1 0 0 0; 1 0 0 1; 1 0 1 0; 1 0 1 1;
           1 1 0 0; 1 1 0 1; 1 1 1 0; 1 1 1 1];
    cost = zeros(16, 1);

    for n = 1:16
        cost(n) = process_1_1(var(n, 1)) + process_1_2(var(n, 2)) + ...
                  process_2(var(n, 1), var(n, 2), var(n, 3), var(n, 4));
    end


    % 打印结果  （认为是 15%）
    fprintf('是否检测零件 1    是否检测零件 2    是否检测成品    是否拆解次品    成本期望\n');
    for n = 1:16
        fprintf('%d    %d    %d    %d    %.2f\n', ...
                var(n, 1), var(n, 2), var(n, 3), var(n, 4), cost(n));
    end


    %%
    [min_cost,position]=min(cost);%认为的最小成本

    % 打印结果
    best_method=var(position,:);%15%下最好的方法
```

```
X=['预期的最好方法与收入',num2str(var(position,:)),'
',num2str(sell_price-min_cost)];
 disp(X);
 %%以实际数据中越各有 20%的次品执行这种方法
p_1 = 0.04;  % 实际设定次品率
p_2 = 0.04;
p_3 = 0.04;

new_cost=process_1_1(var(position, 1)) + process_1_2(var(position, 2)) + ...
                process_2(var(position, 1), var(position, 2), var(position, 3),
var(position, 4));
X=['此种方法下的实际收入',num2str(sell_price-new_cost)];
disp(X);%实际收入与预期收入不符
  %%重新执行，估计新的方法
  % 使用贝叶斯更新来计算先验分布的参数
  p_1 = 0.05;  % 题目次品率
 p_2 = 0.05;
 p_3 = 0.05;
  prior_alpha_1 = predict_p1*10;
  prior_beta_1 = 10-prior_alpha_1;
  prior_alpha_2 = predict_p2*10;
  prior_beta_2 = 10-prior_alpha_2;
  prior_alpha_3 = predict_p3*10;  %先验分布，为 20%,设置 beta(alpha,beta)
  prior_beta_3 = 10-prior_alpha_3;
     mean_posterior_1 = predict_p1;
   mean_posterior_2 = predict_p2;
     mean_posterior_3 = predict_p3;%调整值
  p_1 = 0.04;  % 实际初始设置
  p_2 = 0.04;
  p_3 = 0.04;

  while
abs(mean_posterior_1-p_1)>1e-6&&abs(mean_posterior_2-p_2)>1e-6&&abs(mean_posterior
_3-p_3)>1e-6%当后验分布于实际生产过程中的分布一致时，生产方式调整停止
        posterior_alpha_1 = prior_alpha_1 + p_1*100;
        posterior_beta_1 = prior_beta_1 + 100*(1-p_1);
```

```
%p_1
% 计算后验分布的均值
mean_posterior_1 = posterior_alpha_1 / (posterior_alpha_1 + posterior_beta_1);
prior_alpha_1=mean_posterior_1*10;
prior_beta_1=10-prior_alpha_1;%产生新的 p
p_1=mean_posterior_1;%生产策略中的预期被调整

posterior_alpha_2 = prior_alpha_2 + p_2*100;
posterior_beta_2 = prior_beta_2 + 100*(1-p_2);
%p_2
% 计算后验分布的均值
mean_posterior_2 = posterior_alpha_2 / (posterior_alpha_2 + posterior_beta_2);
prior_alpha_2=mean_posterior_2*10;
prior_beta_2=10-prior_alpha_2;%产生新的 p
p_2=mean_posterior_2;

posterior_alpha_3 = prior_alpha_3 + p_3*100;
posterior_beta_3 = prior_beta_3 + 100*(1-p_3);
%p_3
% 计算后验分布的均值
mean_posterior_3 = posterior_alpha_3 / (posterior_alpha_3 + posterior_beta_3);
prior_alpha_3=mean_posterior_3*10;
prior_beta_3=10-prior_alpha_3;%产生新的 p
p_3=mean_posterior_3;
    for n=1:16
      cost(n) = process_1_1(var(n, 1)) + process_1_2(var(n, 2)) + ...
                process_2(var(n, 1), var(n, 2), var(n, 3), var(n, 4)); %以新的概率设置重新计算花费
    end

[min_cost,position]=min(cost);%最小花销的最小值与方案
X=['预期的最好方法与收入',num2str(var(position,:)),'',num2str(sell_price-min_cost)];
```

```
                disp(X)
                %%以实际上为 20%的统计量执行这种方法
                p_1 = 0.04;   % 实际设置次品率
                p_2 = 0.04;
                p_3 = 0.04;

                new_cost=process_1_1(var(position, 1)) + process_1_2(var(position, 2))
+ ...
                                process_2(var(position, 1), var(position, 2), var(position,
3), var(position, 4));
                X=['此种方法下的实际收入',num2str(sell_price-new_cost)];
                        disp(X)
            end
            X=[' 最 终 调 整 至 方 法 为 ',num2str(var(position,:)),'  最 大 利 润 为
',num2str(sell_price-min_cost)]
            disp(X)
```

代码二

```
function c = process_2(x1, x2, x3, x4)
    global p_1;
global p_2;
global cost_buy_2;
global cost_buy_1;
global cost_assemble;
global cost_test_1;
global cost_test_2;
global cost_swap;
global cost_disa;
global p_3;
global cost_test_3;
c = 0;
    if x3 == 0   %成品不检测
        c = c + cost_assemble;
        if x4 == 0   %不合格品不拆解
            % 计算 p 的值（这里使用了 Matlab 的向量化计算来简化代码）
```

```
                p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3;
                if x1 == 1 && x2 == 0
                    p = p_2 + p_3 - p_2*p_3;
                elseif x1 == 0 && x2 == 1
                    p = p_1 + p_3 - p_1*p_3;
                elseif x1 == 1 && x2 == 1
                    p = p_3;
                end
                % 注意：这里使用了 Matlab 的向量化来计算 p 的幂次方和对应的成本
                %调换损失+
                c = c + sum(p.^(1:5) .* (cost_swap + process_1_1(x1) + process_1_2(x2) + cost_assemble));
            else %不合格品拆解
                if x1 == 0&&x2 == 0
                    p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 + p_1*p_2*p_3 ;
    %
                    c=c+sum(p.^(1:5).*(cost_swap + cost_disa + process_1_1(x1) +...
                    process_1_2(x2)-cost_buy_1 - cost_buy_2 + cost_assemble+...
                    cost_assemble + cost_test_1 +cost_test_2+0.5*cost_buy_1+0.5*cost_buy_2)) ;


                elseif x1 == 1&&x2 == 0%1 经过检验
                    p = p_2 + p_3 - p_2*p_3;
    %
                        c=c+sum(p.^(1:5).*(cost_swap + cost_disa + process_1_1_1(x1) +...
                    process_1_2(x2)-cost_buy_1 - cost_buy_2 + cost_assemble-...
                    cost_test_1 + cost_test_2+0.5*cost_buy_2)) ;
                elseif x1 == 0&&x2 == 1
                    p = p_1 + p_3 - p_1*p_3;
    %
                        c=c+sum(p.^(1:5).*(cost_swap + cost_disa + process_1_1(x1) +...
```

```
                process_1_2_1(x2)-cost_buy_1 - cost_buy_2 + cost_assemble...
                + cost_test_1 - cost_test_2 +0.5*cost_buy_1)) ;
            elseif x1 == 1&&x2 == 1%x1 与 x2 均经过检验
                p = p_3;
                c=c+sum(p.^(1:5).*(cost_swap + cost_disa + process_1_1_1(x1) +...
                process_1_2_1(x2)-cost_buy_1 - cost_buy_2 + cost_assemble-
cost_test_1 - cost_test_2));
            end
        end
    else%成品检测
        c = c + cost_assemble + cost_test_3;
        if x4 == 0
            if x1 == 0&&x2 == 0
                p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 +
2*p_1*p_2*p_3 ;
            elseif x1 == 1&&x2 == 0
                p = p_2 + p_3 - p_2*p_3;
            elseif x1 == 0&&x2 == 1
                p = p_1 + p_3 - p_1*p_3;
            elseif x1 == 1&&x2 == 1
                p = p_3;
            end
              c=c+sum(p.^(1:5).*(process_1_1(x1) + process_1_2(x2) +
cost_assemble + cost_test_3));
        else
            if x1 == 0&&x2 == 0
                p = p_1 + p_2 + p_3 - p_1*p_2 - p_2*p_3 - p_1*p_3 +
2*p_1*p_2*p_3 ;
              c=c+sum(p.^(1:5).*(cost_test_3+cost_disa+process_1_1(x1) +
process_1_2(x2) -...
                  cost_buy_1-cost_buy_2+ cost_assemble +...
                  cost_test_1 +cost_test_2 + 0.5*cost_buy_1+0.5*cost_buy_2 ));
            elseif x1 == 1&&x2 == 0
                p = p_2 + p_3 - p_2*p_3;

c=c+sum(p.^(1:5).*(cost_test_3+cost_disa+process_1_1_1(x1) + process_1_2(x2) -...
```

```
                                        cost_buy_1-cost_buy_2+ cost_assemble    -...
                                        cost_test_1 +cost_test_2 +0.5*cost_buy_2));
                elseif x1 == 0&&x2 == 1
                    p = p_1 + p_3 - p_1*p_3;

c=c+sum(p.^(1:5).*(cost_test_3+cost_disa+process_1_1(x1) + process_1_2_1(x2) -...
                                        cost_buy_1-cost_buy_2+ cost_assemble +...
                                        cost_test_1 -cost_test_2 +0.5*cost_buy_1 ));
                elseif x1 == 1&&x2 == 1
                    p = p_3;

c=c+sum(p.^(1:5).*(cost_test_3+cost_disa+process_1_1_1(x1) + process_1_2_1(x2)- ...
                                        cost_buy_1   -   cost_buy_2   +cost_assemble   -
cost_test_1 -cost_test_2));

                end

            end
        end
```

代码三

```
function c = process_1_2(x2)
%含有不检 2 与检 2 的成本
global p_1;
global p_2;
global cost_buy_2;
global cost_buy_1;
global cost_assemble;
global cost_test_1;
global cost_test_2;
global cost_swap;
global cost_disa;
global p_3;
global cost_test_3;
    c = 0;
```

```
        if x2 == 0
            c = c + cost_buy_2;    %
        else
            c = c + cost_buy_2 + cost_test_2 + sum(p_2.^(1:5) .* (cost_buy_2 +
cost_test_2));
        end
    end

    function c = process_1_2_1(x2) %调换后更换部件 2 的成本
    global p_1;
    global p_2;
    global cost_buy_2;
    global cost_buy_1;
    global cost_assemble;
    global cost_test_1;
    global cost_test_2;
    global cost_swap;
    global p_3;
    global cost_test_3;
    global cost_disa;
        c = 0;
        if x2 == 0
            c = c + cost_buy_2;    %
        else
            c = c + cost_buy_2 + cost_test_2;
        end
    end

    function c = process_1_1(x1)   %未通过检测，含有不检 1 与检 1 的成本
        global p_1;
    global p_2;
    global cost_buy_2;
    global cost_buy_1;
    global cost_assemble;
    global cost_test_1;
    global cost_test_2;
```

```
    global cost_swap;
    global cost_disa;
    global p_3;
    global cost_test_3;
    c = 0;
        if x1 == 0
            c = c + cost_buy_1;
        else
            c = c + cost_buy_1 + cost_test_1 + sum(p_1.^(1:5) .* (cost_buy_1 +
cost_test_1));
        end
    end

    function c = process_1_1_1(x1)   %已通过检测调换后更换部件 1 的成本
    global p_1;
    global p_2;
    global cost_buy_2;
    global cost_buy_1;
    global cost_assemble;
    global cost_test_1;
    global cost_test_2;
    global cost_swap;
    global cost_disa;
    global p_3;
    global cost_test_3;
        c = 0;
        if x1 == 0
            c = c + cost_buy_1;
        else
            c = c + cost_buy_1 + cost_test_1;
        end   %拆解，仅含有购 1
    end
```

8.5 问题一相关数据

95%		90%	
抽样数（次）	最少次品数（个）	抽样数（次）	最多次品数（个）
10	3	10	0
11	3	11	0
12	3	12	0
13	3	13	0
14	3	14	0
15	4	15	0
16	4	16	0
17	4	17	0
18	4	18	0
19	4	19	0
20	4	20	0
21	5	21	0
22	5	22	1
23	5	23	1
24	5	24	1
25	5	25	1
26	5	26	1
27	5	27	1
28	6	28	1
29	6	29	1
30	6	30	1
31	6	31	1
32	6	32	1
33	6	33	1
34	6	34	1
35	7	35	1
36	7	36	1

37	7	37	1
38	7	38	2
39	7	39	2
40	7	40	2
41	7	41	2
42	8	42	2
43	8	43	2
44	8	44	2
187	26	187	14
188	26	188	14
189	26	189	14
190	26	190	14
191	26	191	14
192	26	192	14
193	26	193	14
194	26	194	14
195	27	195	14
196	27	196	14
197	27	197	14
198	27	198	14

8.6 问题二相关数据

情况一

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	期望成本	期望利润
否	否	否	否	40.62075756	15.37924244
否	否	否	是	40.24955881	15.75044119
否	否	是	否	42.50716131	13.49283869
否	否	是	是	42.13596255	13.86403745
否	是	否	否	42.55735744	13.44264256
否	是	否	是	38.25801622	17.74198378
否	是	是	否	44.85382798	11.14617202
否	是	是	是	40.55448676	15.44551324

是	否	否	否	39.26535183	16.73464817
是	否	否	是	37.46744478	18.53255522
是	否	是	否	41.56182237	14.43817763
是	否	是	是	39.76391532	16.23608468
是	是	否	否	40.66658667	15.33341333
是	是	否	是	37.88884	18.11116
是	是	是	否	43.33325667	12.66674333
是	是	是	是	40.55551	15.44449

情况二

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	期望成本	期望利润
否	否	否	否	59.50938332	-3.509383323
否	否	否	是	58.58263675	-2.582636755
否	否	是	否	59.72914362	-3.729143618
否	否	是	是	58.80239705	-2.80239705
否	是	否	否	59.86930407	-3.869304068
否	是	否	是	47.98939433	8.01060567
否	是	是	否	61.19200773	-5.192007735
否	是	是	是	49.312098	6.687902003
是	否	否	否	52.46545579	3.534544207
是	否	否	是	47.71338455	8.28661545
是	否	是	否	53.78815946	2.21184054
是	否	是	是	49.03608822	6.963911782
是	是	否	否	51.18114017	4.818859827
是	是	否	是	43.99648	12.00352
是	是	是	否	53.43138017	2.568619827
是	是	是	是	46.24672	9.75328

情况三

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	期望成本	期望利润
否	否	否	否	49.5295276	6.470472396
否	否	否	是	49.15832885	6.841671148
否	否	是	否	42.50716131	13.49283869
否	否	是	是	42.13596255	13.86403745
否	是	否	否	48.18559312	7.814406878

否	是	否	是	43.8862519	12.1137481
否	是	是	否	44.85382798	11.14617202
否	是	是	是	40.55448676	15.44551324
是	否	否	否	44.89358751	11.10641249
是	否	否	是	43.09568045	12.90431955
是	否	是	否	41.56182237	14.43817763
是	否	是	是	39.76391532	16.23608468
是	是	否	否	43.33322667	12.66677333
是	是	否	是	40.55548	15.44452
是	是	是	否	43.33325667	12.66674333
是	是	是	是	40.55551	15.44449

情况四

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	期望成本	期望利润
否	否	否	否	81.75130096	-25.75130096
否	否	否	是	78.04431469	-22.04431469
否	否	是	否	57.80239705	-1.80239705
否	否	是	是	54.09541078	1.904589223
否	是	否	否	69.39017724	-13.39017724
否	是	否	是	58.34882621	-2.348826214
否	是	是	否	55.73541147	0.264588531
否	是	是	是	44.69406044	11.30593956
是	否	否	否	63.93507771	-7.935077711
是	否	否	是	58.76363766	-2.763637658
是	否	是	否	50.28031194	5.719688062
是	否	是	是	45.10887188	10.89112812
是	是	否	否	52.49232015	3.507679846
是	是	否	是	46.2448	9.7552
是	是	是	否	47.49456015	8.505439846
是	是	是	是	41.24704	14.75296

情况五

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	期望成本	期望利润
否	否	否	否	48.53042659	7.469573406
否	否	否	是	50.15124975	5.848750254

否	否	是	否	46.20823152	9.791768479
否	否	是	是	47.82905467	8.170945327
否	是	否	否	44.00792817	11.99207183
否	是	否	是	41.01828442	14.98171558
否	是	是	否	44.13184961	11.86815039
否	是	是	是	41.14220586	14.85779414
是	否	否	否	55.70904241	0.290957594
是	否	否	是	49.3681275	6.631872499
是	否	是	否	54.60328563	1.396714369
是	否	是	是	48.26237073	7.737629274
是	是	否	否	48.9797188	7.020281202
是	是	否	是	45.41511	10.58489
是	是	是	否	50.0908388	5.909161202
是	是	是	是	46.52623	9.47377

情况六

是否检测零件 1	是否检测零件 2	是否检测成品	是否拆解不合格成品	期望成本	期望利润
否	否	否	否	34.32095657	21.67904343
否	否	否	是	39.9765493	16.0234507
否	否	是	否	36.15656984	19.84343016
否	否	是	是	41.81216256	14.18783744
否	是	否	否	36.65399074	19.34600926
否	是	否	是	38.58720016	17.41279984
否	是	是	否	38.89776471	17.10223529
否	是	是	是	40.83097414	15.16902586
是	否	否	否	34.67119639	21.32880361
是	否	否	是	37.66198504	18.33801496
是	否	是	否	36.91497037	19.08502963
是	否	是	是	39.90575901	16.09424099
是	是	否	否	36.75900157	19.24099843
是	是	否	是	37.36841969	18.63158031
是	是	是	否	39.39058063	16.60941937
是	是	是	是	39.99999875	16.00000125

灵敏度检测（情况一）

是否检	是否检	是否检	是否拆	期望成本	期望利润

测零件 1	测零件 2	测成品	解不合格成品		
否	否	否	否	40.62075756	15.37924244
否	否	否	是	40.62075756	15.37924244
否	否	是	否	42.50716131	13.49283869
否	否	是	是	42.50716131	13.49283869
否	是	否	否	42.55735744	13.44264256
否	是	否	是	38.49252604	17.50747396
否	是	是	否	44.85382798	11.14617202
否	是	是	是	40.78899658	15.21100342
是	否	否	否	40.63702804	15.36297196
是	否	否	是	38.57855478	17.42144522
是	否	是	否	42.93349858	13.06650142
是	否	是	是	40.87502532	15.12497468
是	是	否	否	41.9011521	14.0988479
是	是	否	是	38.99995	17.00005
是	是	是	否	44.5678221	11.4321779
是	是	是	是	41.66662	14.33338

8.7 问题三相关数据

表格一

零件 1，4 是否检测	零件 2，5 是否检测	零件 3，6 是否检测	是否拆解	半成品 1，2 是否检测	制造成本期望
(0,	0,	0,	0,	0)	30
(0,	0,	0,	0,	1)	51.73564479
(0,	0,	0,	1,	0)	30
(0,	0,	0,	1,	1)	51.21400818
(0,	0,	1,	0,	0)	33.55554
(0,	0,	1,	0,	1)	51.49610957
(0,	0,	1,	1,	0)	33.55554
(0,	0,	1,	1,	1)	46.83550879
(0,	1,	0,	0,	0)	31.99999

(0,	1,	0,	0,	1)	49.36314135
(0,	1,	0,	1,	0)	31.99999
(0,	1,	0,	1,	1)	46.39355505
(0,	1,	1,	0,	0)	35.55553
(0,	1,	1,	0,	1)	48.83169022
(0,	1,	1,	1,	0)	35.55553
(0,	1,	1,	1,	1)	44.2457264
(1,	0,	0,	0,	0)	31.33333
(1,	0,	0,	0,	1)	48.44901799
(1,	0,	0,	1,	0)	31.33333
(1,	0,	0,	1,	1)	46.84049131
(1,	0,	1,	0,	0)	34.88887
(1,	0,	1,	0,	1)	48.0086919
(1,	0,	1,	1,	0)	34.88887
(1,	0,	1,	1,	1)	44.28259586
(1,	1,	0,	0,	0)	33.33332
(1,	1,	0,	0,	1)	46.08835015
(1,	1,	0,	1,	0)	33.33332
(1,	1,	0,	1,	1)	43.43057532
(1,	1,	1,	0,	0)	36.88886
(1,	1,	1,	0,	1)	45.43202123
(1,	1,	1,	1,	0)	36.88886
(1,	1,	1,	1,	1)	42.88884

表格二

零件 1，4 是否检测	零件 2，5 是否检测	零件 3，6 是否检测	是否拆解	半成品 1，2 是否检测	制造成品加风险成品
(0,	0,	0,	0,	0)	59.93676719
(0,	0,	0,	0,	1)	57.92843228
(0,	0,	0,	1,	0)	59.93676719
(0,	0,	0,	1,	1)	57.34883663
(0,	0,	1,	0,	0)	56.21503239
(0,	0,	1,	0,	1)	57.66228231
(0,	0,	1,	1,	0)	56.21503239
(0,	0,	1,	1,	1)	52.48384218

(0,	1,	0,	0,	0)	54.48664523
(0,	1,	0,	0,	1)	55.29231999
(0,	1,	0,	1,	0)	54.48664523
(0,	1,	0,	1,	1)	51.99278295
(0,	1,	1,	0,	0)	51.46610575
(0,	1,	1,	0,	1)	54.70181932
(0,	1,	1,	1,	0)	51.46610575
(0,	1,	1,	1,	1)	49.60630906
(1,	0,	0,	0,	0)	53.74591264
(1,	0,	0,	0,	1)	54.27662838
(1,	0,	0,	1,	0)	53.74591264
(1,	0,	0,	1,	1)	52.48937829
(1,	0,	1,	0,	0)	50.72537316
(1,	0,	1,	0,	1)	53.78737766
(1,	0,	1,	1,	0)	50.72537316
(1,	0,	1,	1,	1)	49.64727508
(1,	1,	0,	0,	0)	48.996986
(1,	1,	0,	0,	1)	51.65366673
(1,	1,	0,	1,	0)	48.996986
(1,	1,	0,	1,	1)	48.70058654
(1,	1,	1,	0,	0)	46.65419123
(1,	1,	1,	0,	1)	50.92441311
(1,	1,	1,	1,	0)	46.65419123
(1,	1,	1,	1,	1)	48.09865901

表格三

零件 7 是否检测	零件 8 是否检测	是否拆解	半成品 3 是否检测	制造成本期望
(0,	0,	0,	0)	18
(0,	0,	0,	1)	33.87836006
(0,	0,	1,	0)	18
(0,	0,	1,	1)	34.82697332
(0,	1,	0,	0)	19.99999
(0,	1,	0,	1)	31.97332153
(0,	1,	1,	0)	19.99999
(0,	1,	1,	1)	29.86273315

(1,	0,	0,	0)	19.33333
(1,	0,	0,	1)	30.83764424
(1,	0,	1,	0)	19.33333
(1,	0,	1,	1)	29.5608649
(1,	1,	0,	0)	21.33332
(1,	1,	0,	1)	29.11105889

表格四

零件 7 是否检测	零件 8 是否检测	是否拆解	半成品 3 是否检测	制造成品加风险成品
(0,	0,	0,	0)	38.76758013
(0,	0,	0,	1)	37.86765694
(0,	0,	1,	0)	38.76758013
(0,	0,	1,	1)	39.14103832
(0,	1,	0,	0)	33.744571
(0,	1,	0,	1)	35.91758943
(0,	1,	1,	0)	33.744571
(0,	1,	1,	1)	33.62522143
(1,	0,	0,	0)	32.92260422
(1,	0,	0,	1)	34.65882865
(1,	0,	1,	0)	32.92260422
(1,	0,	1,	1)	33.2898126
(1,	1,	0,	0)	28.03031852
(1,	1,	0,	1)	32.78628864

表格五

成品是否检测	成品是否拆解	半成品 1 是否拆解	半成品 2 是否拆解	半成品 3 是否拆解	期望成本
(0,	0,	0,	0,	0)	177.762998
(0,	0,	0,	0,	1)	177.762998
(0,	0,	0,	1,	0)	177.762998
(0,	0,	0,	1,	1)	177.762998
(0,	0,	1,	0,	0)	177.762998
(0,	0,	1,	0,	1)	177.762998
(0,	0,	1,	1,	0)	177.762998
(0,	0,	1,	1,	1)	177.762998
(0,	1,	0,	0,	0)	167.2822007

(0,	1,	0,	0,	1)	165.1498017
(0,	1,	0,	1,	0)	160.6265275
(0,	1,	0,	1,	1)	158.4941285
(0,	1,	1,	0,	0)	160.6265275
(0,	1,	1,	0,	1)	158.4941285
(0,	1,	1,	1,	0)	153.9708543
(0,	1,	1,	1,	1)	151.8384554
(1,	0,	0,	0,	0)	166.0273532
(1,	0,	0,	0,	1)	166.0273532
(1,	0,	0,	1,	0)	166.0273532
(1,	0,	0,	1,	1)	166.0273532
(1,	0,	1,	0,	0)	166.0273532
(1,	0,	1,	0,	1)	166.0273532
(1,	0,	1,	1,	0)	166.0273532
(1,	0,	1,	1,	1)	166.0273532
(1,	1,	0,	0,	0)	155.5465559
(1,	1,	0,	0,	1)	153.4141569
(1,	1,	0,	1,	0)	148.8908827
(1,	1,	0,	1,	1)	146.7584837
(1,	1,	1,	0,	0)	148.8908827
(1,	1,	1,	0,	1)	146.7584837
(1,	1,	1,	1,	0)	142.2352095
(1,	1,	1,	1,	1)	140.1028106

8.8 问题四相关数据

零件 1 是否检测	零件 2 是否检测	成品是否检测	成品是否拆解	预期收入	实际收入	迭代次数
是	否	否	是	18.5326	16.2716	1
是	是	否	是	16.8447	16.7145	2
是	是	否	是	16.7263	16.7145	3
是	是	否	是	16.7156	16.7145	4
是	是	否	是	16.7146	16.7145	5
是	是	否	是	16.7145	16.7145	6

零件 1 是否检测	零件 2 是否检测	成品 是否检测	成品 是否拆解	预期收入	实际收入	迭代次数
是	是	否	是	12.0035	10.2332	1
是	是	否	是	10.3987	10.2332	2
是	是	否	是	10.2483	10.2332	3
是	是	否	是	10.2345	10.2332	4
是	是	否	是	10.2333	10.2332	5
是	是	否	是	10.2332	10.2332	6

零件 1 是否检测	零件 2 是否检测	成品 是否检测	成品 是否拆解	预期收入	实际收入	迭代次数
是	否	是	是	16.2361	14.1893	1
是	否	是	是	14.3822	14.1893	2
是	否	是	是	14.2069	14.1893	3
是	否	是	是	14.1909	14.1893	4
是	否	是	是	14.1895	14.1893	5
是	否	是	是	14.1893	14.1893	6

零件 1 是否检测	零件 2 是否检测	成品 是否检测	成品 是否拆解	预期收入	实际收入	迭代次数
是	是	是	是	14.753	16.1528	1
是	是	是	是	16.029	16.1528	2
是	是	是	是	16.1416	16.1528	3
是	是	是	是	16.1518	16.1528	4
是	是	是	是	16.1527	16.1528	5
是	是	是	是	16.1528	16.1528	6

零件 1 是否检测	零件 2 是否检测	成品 是否检测	成品 是否拆解	预期收入	实际收入	迭代次数
否	是	否	是	14.9817	12.004	1
否	是	是	是	12.6776	12.4512	2
否	是	是	是	12.4719	12.4512	3
否	是	是	是	12.4531	12.4512	4
否	是	是	是	12.4514	12.4512	5

否	是	是	是	12.4512	12.4512	6

零件 1 是否检测	零件 2 是否检测	成品是否检测	成品是否拆解	预期收入	实际收入	迭代次数
否	否	否	否	21.679	23.0494	1
否	否	否	否	22.9272	23.0494	2
否	否	否	否	23.0383	23.0494	3
否	否	否	否	23.0484	23.0494	4
否	否	否	否	23.0493	23.0494	5