第十届湖南省研究生数学建模竞赛承诺书

我们仔细阅读了湖南省高校研究生数学建模竞赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们完全清楚，在竞赛中必须合法合规地使用文献资料、软件工具和 AI 工具，不能有任何侵犯知识产权的行为。否则我们将失去评奖资格，并可能受到严肃处理。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们授权湖南省研究生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号是（从组委会提供的赛题中选择一项填写）：A

我们的参赛编号（请填写完整参赛编号）：202518001005

所属学校（请填写完整的全名）：国防科技大学

参赛队员（打印后签名）：1. 黎焕文

2. 赵丰硕

3. 叶昕岛

指导教师或指导教师组负责人（打印后签名）：

日期：2025 年 8 月 27 日

（请勿改动此页内容和格式。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。）

第十届湖南省研究生数学建模竞赛

题　目：基于 TOPSIS 与概率仿真的大型装置多阶段测试流程优化

摘要：随着高端制造产业对产品可靠性要求的提升，大型复杂装置多阶段测试的效率与准确性成为企业产能保障的关键瓶颈。本文针对大型装置测试流程中的资源调度、时间优化及判罚准确性问题，基于概率论、TOPSIS 多指标决策法与仿真模拟思想，通过确定任务完成时间、合格装置数量、漏判/误判概率、资源利用率等核心指标，构建多场景下的测试调度优化模型，并结合 Python 代码对模型进行仿真求解。

针对问题 1：聚焦测试流程中的概率参数计算，基于全概率公式与贝叶斯定理，忽略测手误判的微小干扰，推导各子系统漏判比$\lambda_1 \sim \lambda_4$及综合测试问题检出概率的数学表达式。代入已知子系统问题概率（A:2.5%、B:3%等）与测手差错概率（A:3%、B:4%等），计算得$\lambda_1 = 0.2235. \lambda_2 = 0.3567. \lambda_3 = 0.1195. \lambda_4 = 0.3003$，综合测试问题检出概率为 0.01326。

针对问题 2：以 100 个装置单班 12 小时测试为场景，考虑设备故障、重测规则、转运时间等约束，设计 ABC 子系统 6 种测试顺序方案。通过 Python 代码仿真获取各方案的平均天数（T）、合格数（S）等指标，再用 TOPSIS 法对 “平均天数-漏判概率”双目标排序，确定最优测试顺序为 C-A-B-E。该方案下，任务完成平均天数 T=51.02 天，通过测试装置平均数目 S=92.47，总漏判概率$P_L = 0.0001$，总误判概率P_W=0.0137，A~E 组有效工作时间比 YXB 分别为 0.4164、0.3275、0.4213、0.4618。

针对问题 3：以双分队倒班（$K \in [9,12]$小时，步长 0.5 小时）加速任务为场景，沿用 C-A-B-E 测试顺序，仿真不同 K 值下的调度效果。结果显示 K<11 时因测试流程无法单班完成导致时间激增，K≥11 时效率趋稳；结合“平均天数-资源利用率”平衡，确定最优 K=11.5 小时。此方案下，T=26.03 天，S=92.92，P_L=6.3525E-05，P_W=0.0119，YXB 分别为 0.4300、0.3379、0.4333、0.4770。

针对问题 4：通过敏感性分析明确班次时间 K、设备更换策略、设备故障率是影响任务时间的核心因素。据此提出改进建议：优先采用 11 小时双倒班模式，测试设备不提前更换，同时优化测手培训以降低差错率。

本文模型充分贴合测试流程实际约束，兼顾时间效率与判罚准确性，可推广至半导体、轨道交通等类似多阶段检测场景，为企业资源调度提供科学依据。

关键词：大型装置测试；TOPSIS 法；概率计算；双分队倒班；仿真优化

目录

1 问题综述

1.1 问题背景

随着高端制造产业对产品可靠性要求的不断提高，大型复杂装置的多阶段测试已成为质检流程中最耗时的瓶颈环节[1]。例如，近年来我国新能源汽车、半导体、轨道交通等行业因测试环节拖期导致的交付延期事件不胜枚举，直接造成了大量的经济损失。本题中的测试大厅仅配备两个测试台和四组专业测试工位，而单班 12 小时工作制又进一步限制了产能。与此同时，测试设备老化、人员操作失误、子系统偶发缺陷等不确定性因素，使得测试流程频繁中断，造成“人等机、机等人”的资源闲置现象，严重影响企业的产能提升与市场响应速度。

目前，针对这个问题行业内已经有诸多的研究，也提出了很多解决问题的方案。虽然这些研究对当下大型装置的检测起到很大的推进作用，但在很多时候，检测的准确性和效率依然主要取决于测试设备和人员。不过，通过更为科学合理的安排，我们可以有效避免资源闲置现象，从而提高工作效率，这也促使我们选择更加合理的算法、探索更加合理的工作方式。

1.2 问题提出

我们需要根据企业的实际情况制定一套测试工作计划，使得该工作计划下的总用时和资源闲置现象尽量少、可靠度尽量高。为此，我们将首先讨论大型装置的检测流程、分工方法以及在检测中可能会出现的意外问题，并根据已知数据计算所需参数，以便后续的模型搭建。我们需要满足题中所给的下列约束：

(1) B、C 子系统分别单独测试。

(2) 测试大厅有两个测试台，可同时放置并测试 2 个大型装置。

(3) 若某道工序测出有问题，需进行重测，连续两次未通过测试，则该装置将退出测试。

(4) 子系统已通过测试，但可能有漏判。

我们需要解决下列四个问题:

(1) 问题 1:写出计算各比例参数 $\lambda_i, i=1,2,3,4$ 的数学表达式，以及综合测试测出系统有问题的概率表达式，并代入具体数值计算这些概率值与比例参数值。

(2) 问题 2：一个测试分队受领一项测试任务，测试 100 个大型装置。每日一个班次，每个班次工作时间不超过 12 小时。规定若某个工序测试因故发生中断时该工序测试不得接续进行，需重新开始测试。请根据测试任务的主要目标制定测试工作计划。计算在该工作计划下，任务完成平均天数（T）、通过测试的装置的平均数目（S）、总漏判概率（P_L）、总误判概率（P_W）、各个专业测试组的有效工作时间比（即：小组每班次平均测试时间/12，记作 YXB）等各项统计指标。

(3) 问题 3：由于任务紧急，为了加快测试进度，在第二批 100 个装置开始测试时安排两个分队接续倒班，每个班次工作 K（9≤K≤12）小时，同工序两个班次使用同一套测试装备。请确定最优的 K 值（以半小时为最小单位），并制定测试工作计划，计算在该工作计划下，任务完成平均天数、通过测试的装置的平均数目、总漏判概率、总误判概率以及各个专业测试组的有效工作时间比（即：小组每次平均测试时间/k，记作 YXB）等各项统计指标。

(4) 问题 4：分析讨论问题 3 中各个因素对测试任务平均完成时间的影响，并据此向主管部门提出对测试工作的改进建议。

2 模型假设与符号说明

2.1 模型基本假设

(1) 假设每个大型设备在同一时刻只能进行其中一项的检测。
(2) 假设每个小组在同一时刻只能对其中一台大型设备进行检测。
(3) 假设任务起始时刻待测试装置已到位。
(4) 假设一台设备在运入或运出过程中不会影响另一台设备。
(5) 假设大型设备的 ABC 三个子系统的检测中没有先后的次序。

2.2 符号说明

本文定义了如下 9 个使用次数较多的符号, 其余符号在使用时注明.

表1 符号说明

符号	含义	单位
λ_1	问题指向子系统 A 的比例	无
λ_2	问题指向子系统 B 的比例	无
λ_3	问题指向子系统 C 的比例	无
λ_4	问题指向子系统 D 的比例	无
T	任务完成平均天数	天
S	通过测试的装置的平均数目	个
P_L	总漏判概率	无
P_W	总误判概率	无
YXB	有效工作时间比	无

3 问题 1 分析与求解

3.1 背景与假设

题目 1 以大型装置的多阶段可靠性测试为背景，介绍了在可靠性测试的流程以及可能出现的误判和漏判的情形。问题一要求我们求解在综合测试发现问题时各子系统漏判所占比例 $\lambda_i, i=1,2,3,4$ 的数学表达式和值及综合测试测出系统有问题的概率表达式。

基于题中所给出的工作流程，我们先讨论 E 测试小组的测手发生差错的问题。假设 E 测试小组测手发生差错（2%概率）且为漏判（2%*50%=1%），则综合测试无法测出问题，无法体现各子系统所占比例；若为误判（概率同样为 1%），由于综合测试测出的问题具有指向性，根据题目所给条件无法判断此时测出问题该指向哪个子系统。除此之外，当综合测试测出问题时属于误判的概率较小，可以忽略该情形。因此，我们假定表 1 综合测试测出有问题时各部分所占比例是在 E 测试小组测手未发生差错（98%概率）时得到的，即综合测试指向性只取决于真实问题。

由于题目中提到“4 个小组各有一个专用工位，可分别独立进行测试”，我们据此可以假设子系统 A、B、C、D 的问题发生是独立的。另外，多个问题同时发生的概率很小，我们假设在综合测试时，最多只有一个子系统有问题，即忽略问题同时发生的概率。

3.2 核心公式推导

基于以上假设，我们可以根据全概率公式及贝叶斯公式得到所有问题概率之和、各比例参数 λ_i、某子系统出现问题的概率及该系统出现漏判从而导致综合测试发生错误的概率四者之间的关系。

令所有问题概率之和为 P_{total}、各比例参数为 $\lambda_i(i=1,2,3,4)$、子系统 A 出现问题的概率为 P_1（同理 B 对应 2、C 对应 3、D 对应 4），子系统 A 出现问题却漏判从而导致综合测试发生错误的概率为 P_{1_miss}（同理 B 对应 2、C 对应 3）有

$$P_{total}=\sum_{i=1}^{3}P_i\cdot P_{i\_miss}+P_4\ （全概率公式）$$

$$\lambda_i=\frac{P_i\cdot P_{i\_miss}}{\sum_{i=1}^{3}P_i\cdot P_{i\_miss}+P_4}=\frac{P_i\cdot P_{i\_miss}}{P_{total}}\ （贝叶斯公式）$$

其中，$P_i(i=1,2,3,4)$ 已知。根据题目条件“若某道工序测出有问题，需进行重测，连续两次未通过测试，则该装置将退出测试”，我们可以推断出子系统出现问题却漏判从而导致综合测试发生错误的概率由两部分组成，即第一次检测便发生漏判与第一次检测检出问题重测而第二次检测漏判。设 A、B、C 三测试小组测手差错概率分别为 P_{1_false}、P_{2_false}、P_{3_false}，则

$$\begin{aligned}P_{i\_miss}&=P_{i\_false}\times 50\%+(1-P_{i\_false}\times 50\%)\times P_{i\_false}\times 50\%\\&=(2-P_{i\_false}\times 50\%)\times P_{i\_false}\times 50\%\end{aligned}$$

下面计算综合测试测出系统有问题的概率：

综合测试系统实际有问题的概率即为所有问题概率之和 P_{total}，设 E 测试小组测手差错概率为 P_{E_false}、综合测试测出系统有问题的概率为 P_{detected}，有

$$P_{\mathrm{detected}}=P_{total}\times(1-P_{E\_false})+P_{E\_false}\times 50\%$$

现整理题目已知数据如下表：

表2 已知参数表

参数符号	参数含义	值
P_1	子系统 A 出现问题的概率	2.5%
P_2	子系统 B 出现问题的概率	3%
P_3	子系统 C 出现问题的概率	2%
P_4	子系统 D 出现问题的概率	0.1%
$P_{1\ false}$	A 测试小组的测手发生差错的概率	3%
$P_{2\ false}$	B 测试小组的测手发生差错的概率	4%
$P_{3\ false}$	C 测试小组的测手发生差错的概率	2%
$P_{E\ false}$	E 测试小组的测手发生差错的概率	2%

3.3 程序搭建及计算结果

通过以上数学公式，我们使用 python 搭建出了一个用于计算题目所要求参数的程序（附录 1），并通过运行程序计算得到题目 1 所要求的参数值如下（保留四位有效数字）：

λ_1=0.2235，λ_2=0.3567，λ_3=0.1195，λ_4=0.3003，P_{detected}=0.01326

4 问题 2 的分析与模型搭建

4.1 背景与假设

问题 2 以实际生产中的测试任务为背景，在大型装置可靠性测试体系中，测试任务的科学规划是确保装置质量与测试效率的核心环节。题中所给的某大型装置由 A、B、C 三个子系统构成，测试流程要遵循子系统单独测试通过后再进行综合测试的逻辑，即待测大型设备需先通过 A、B、C 子系统的独立测试，方可进入综合测试 E，且仅当 4 项测试全部通过时，装置才算通过整体测试；若任一工序测试出现问题需重测，连续两次未通过则装置退出测试。

测试环节需要依托专用的硬件和人力资源，其中测试大厅配备 2 个可同时运行的测试台，对应 4 个专业测试小组（A、B、C、E），各小组拥有独立工位且测试过程互不干扰。装置运出测试大厅与运入待测试装置各需 0.5 小时，且转运操作不影响大厅内另一装置的测试。

同时在测试过程存在多类不确定性因素，需纳入计划考量：

1.设备故障风险：A、B、C、E 测试设备的故障概率随使用时间分段变化——使用 120 小时内，累积故障概率分别为 3%、4%、2%、3%；使用 120-240 小时区间内，累积故障概率升至 5%、7%、6%、5%。设备使用满 240 小时或发生故障时必须更换，未故障时提前更换需满足“至少工作 120 小时”的条件。

2.子系统固有问题：A、B、C 子系统自身存在问题的概率分别为 2.5%、3%、2%，检出问题后需重测。

3.测手操作差错：A、B、C、E 小组测手发生差错的概率分别为 3%、4%、2%、2%，差错分为两类——误判（系统无问题却判定为有问题，占差错的 50%，需重测）与漏判（系统有问题却未检出，占差错的 50%，工序被判为正常）。

4.综合测试风险：综合测试需覆盖前期子系统漏判问题及装置整体联接后的新问题（联接系统记为子系统 D，其存在问题的概率为 0.1%）。综合测试检出问题时具有指向性，指向 A、B、C、D 子系统的比例分别为 λ_1、λ_2、λ_3、λ_4（近似满足 $\lambda_1+\lambda_2+\lambda_3+\lambda_4=1$），检出问题后需重测。

此外，测试流程存在明确约束：测试任务从“待测试装置到位且所有设备调试校对完毕”时刻起算（设备首次调试时间：A 组 30 分钟、B 组 20 分钟、C 组 20 分钟、E 组 40 分钟，调试含安装拆卸时间）；若某工序测试因故中断（如设备故障、需重测等），该工序不得接续进行，需重新从零开始测试；每日仅安排 1 个班次，每个班次工作时间不超过 12 小时；任务结束时刻为最后 1 个装置完成测试（通过或退出）的时刻。

问题二的核心矛盾在于“单一指标最优≠整体性能最优”：例如某测试顺序可能总时间最短，但漏判概率或误判概率过高，导致合格装置产出不稳定；另一顺序可能资源利用率高，但总周期过长。因此需通过多指标综合评价方法平衡时间、产出、判罚准确性与资源利用效率。为了能够综合考量这些指标而得出最优的方案，我们选择 TOPSIS

法（理想解相似度排序法），该方法能有效处理“不同方向、不同量纲”的多指标体系，同时避免主观偏好导致的决策偏差[2]。

4.2 TOPSIS 法

TOPSIS 法（Technique for Order Preference by Similarity to an Ideal Solution,理想解相似度排序法）[3]是一种多指标决策分析方法。它的基本思想是：优选方案应当距离正理想解（理想状态）最近，距离负理想解（最差状态）最远。通过计算每个方案与正、负理想解的距离，确定各个方案的相对接近度，从而对方案进行排序和选择。

4.2.1 核心概念

理想解：设想的最优方案，其各个属性值都达到各备选方案中的最好值。例如在选择供应商时，若评价指标有产品质量、价格、交货期等，理想解就是产品质量最优、价格最低、交货期最短的供应商（假设这些指标的理想状态如此）[4]。

负理想解：设想的最劣方案，其各个属性值都达到各备选方案中的最坏值 。如上述供应商选择中，负理想解可能是产品质量最差、价格最高、交货期最长的供应商。

相对接近度：通过计算各方案与理想解和负理想解的距离，得出相对接近度。与理想解距离越近，同时与负理想解距离越远，则该方案越优。相对接近度越大，方案越接近正理想解，在备选方案中的排名越高[5]。

4.2.2 计算步骤

构建决策矩阵：由 m 个方案和 n 个指标组成，矩阵的每一行代表一个方案，每一列代表一个评价指标。

标准化决策矩阵：为消除量纲影响，常用向量归一化方法进行标准化处理。

构建加权标准化决策矩阵：根据指标权重对标准化后的矩阵进行加权，以反映各指标的重要性。

确定正理想解和负理想解：正理想解各指标取最大值（效益型指标）或最小值（成本型指标）；负理想解各指标取最小值（效益型指标）或最大值（成本型指标）。

计算与正、负理想解的距离及相对接近度：使用欧几里得距离公式计算各方案与正、负理想解的距离，进而得出相对接近度，依此对方案排序。

4.3 问题分析与建模

基于问题 2 中的检测流程以及约束条件，我们认为其检测可以通过调整 ABC 的检测顺序来寻找一个最为合适的方案。其核心思想为先让其中一台设备使用方案顺序进行检测，待第一台设备的第一个环节通过后第二台设备跟上，以此循环检测。为了找出最为合适的方案，ABC 三个环节共有$A_3^2 = 6$种排序。我们通过程序按照检测流程进行建模，并对每一种可能的方案进行仿真。程序如附录 2 所示，通过仿真模拟我们得到如下结果。

表3 不同测试顺序仿真结果

测试顺序	平均天数(T)	通过数(S)	漏判概率(P_L)	误判概率(P_W)	YXB1	YXB2	YXB3	YXB4
ABCE	50.85	92.44	6.3851E-05	0.0139	0.4267	0.3355	0.3991	0.4634
ACBE	50.98	92.37	6.3622E-05	0.0140	0.4260	0.3278	0.4108	0.4620
BACE	50.77	92.36	6.3997E-05	0.0141	0.4153	0.3444	0.4002	0.4639

续表

测试顺序	平均天数(T)	通过数(S)	漏判概率(P_L)	误判概率(P_W)	YXB1	YXB2	YXB3	YXB4
BCAE	50.79	92.39	6.3902E-05	0.0138	0.4063	0.3442	0.4108	0.4636
CABE	51.02	92.47	6.3525E-05	0.0137	0.4164	0.3275	0.4213	0.4618
CBAE	50.96	92.51	6.3668E-05	0.0141	0.4049	0.3361	0.4216	0.4630

其中漏判概率(P_L)为理论计算值，理论推导如下：

若某一大型装置发生漏判，则是在 A、B、C 三道子系统测试程序中至少发生一次漏判，然后在综合测试中发生漏判，其计算程序见附件 2。

通过上述各项数据，我们可以看出所需要核心考虑的指标主要为平均天数和总漏判概率，针对问题 2 上述过程求得的 6 种不同测试顺序的任务完成平均天数和总漏判概率结果，为了能够综合考量这些指标而得出最优的方案，我将采用 TOPSIS 法进行排序评价。具体程序见附录 3，其具体数学过程如下：

4.3.2 数据正向化处理

首先要对数据进行正向化处理，但是在由于平均天数和漏判概率都是越小越好的指标，两者“方向相同”，所以无需进行正向化处理。

4.3.3 数据标准化处理

接着要对数据进行数据标准化处理，在这里我们将采用向量归一化法进行标准化，消除量纲影响：

$$z_{ij}=\frac{x_{ij}}{\sqrt{\sum_{i=1}^{n}x_{ij}^2}}$$

其中，x_{ij}为原始数据，z_{ij}为标准化后的数据，$n=6$为方案数。

经过处理后可以得到标准化平均天数(ST)和标准化漏判概率(SP_L)：

表4 标准化处理结果

测试顺序	标准化平均天数(ST)	标准化漏判概率(SP_L)
ABCE	0.4079	0.4088
ACBE	0.4089	0.4074
BACE	0.4072	0.4098
BCAE	0.4074	0.4091
CABE	0.4093	0.4067
CBAE	0.4088	0.4077

4.3.4 确定正理想解和负理想解

对于越小越好的指标来说，正理想解为各指标最小值，负理想解为各指标最大值。设标准化后的矩阵为 Z，则：

$$Z^+ = \left(\min z_{i1}, \min z_{i2}\right) = \left(z_1^+, z_2^+\right)$$
$$Z^- = \left(\max z_{i1}, \max z_{i2}\right) = \left(z_1^-, z_2^-\right)$$

4.3.5 计算各方案与正负理想解的距离

计算各方案与正理想解的距离：

$$D_I^+ = \sqrt{\sum_{j=1}^{m}\left(z_{ij} - z_j^+\right)^2}$$

计算各方案与负理想解的距离：

$$D_I^- = \sqrt{\sum_{j=1}^{m}\left(z_{ij} - z_j^-\right)^2}$$

其中，$m = 2$ 为指标数。

4.3.6 计算相对距离并排序

我们将根据如下公式计算正负理想解的相对距离，并进行排序:

$$C_i = \frac{D_i^-}{D_i^+ + D_i^-}$$

其中，C_i 值越大，表示方案越接近正理想解，性能越好。

排序结果为：

表5 不同测试顺序排序结果

测试顺序	平均天数	漏判概率	相对接近度	排序
CABE	51.02	6.35E-05	0.601513	1
ACBE	50.98	6.36E-05	0.573913	2
CBAE	50.96	6.37E-05	0.548918	3
BCAE	50.79	6.39E-05	0.444967	4
ABCE	50.85	6.39E-05	0.430444	5
BACE	50.77	6.4E-05	0.398487	6

可见当我们使用 TOPSIS 法进行处理后，得到最优的排序方式为 CABE。

贴合首要目标：时间可控。CABE 的 T=51.02 天，虽比最短的 BACE（50.77 天）多 0.25 天，但差距仅 0.5%，且远低于“任务可接受周期”（按 12 小时/班，51 天约为单分队满负荷工作的合理范围），未显著违背“尽快完成”的首要目标。满足次要目标：漏判低+误判最低

CABE 的 P_L=6.35E-05，且 P_W=0.0137（6 种方案中最低）——误判最低意味着 “重测次数最少”，间接减少了时间浪费，也降低了装置因重测失败退出的概率，保障了 S=92.47（6 种方案中第 2 高）的合格产出。

资源利用率均衡：核心小组高效 CABE 的 YXB3=0.4213（6 种方案中最高），YXB4=0.4618（处于较高水平）——C 小组利用率高，避免了“C 测试等待前序”的闲

置；E 小组作为综合测试的关键环节，高利用率确保了“前序通过后能快速衔接 E 测试”，无流程断点。对比其他方案的劣势规避

规避 BACE 的缺陷：BACE 虽 T 最短，但 P_W =0.0141（最高），导致重测多，实际操作中易出现“班次内测试中断”，反而延长实际周期；

规避 BCAE 的缺陷：BCAE 虽 P_L =6.39E-05，但 YXB1=0.4063（最低），A 小组闲置率高，资源浪费严重；

规避 CBAE 的缺陷：CBAE 虽 S=92.51（最高），但 P_W =0.0141（最高），且 YXB1=0.4049（最低），资源与判罚稳定性不足。

4.4 结论

根据分析结果，我们最终对所有大型装置采用 C-A-B-E 的测试顺序，该方案下具体统计指标如下表：

表6 问题 2 结果统计指标

T	S	P_L	P_W	YXB1	YXB2	YXB3	YXB4
51.02	92.47	0.0001	0.0137	0.4164	0.3275	0.4213	0.4618

5 问题 3 的分析与模型搭建

5.1 问题背景与假设

面对当下大型设备检测任务时间紧急的问题，问题 3 聚焦于加快检测的进度，在现有的检测硬件设备的前提下，通过安排两个分队接续倒班，以达到能够尽快完成任务的目标。同工序共享一套测试装备，若工序中断需重新开始，班次时长 K∈[9,12]小时（最小步长 0.5 小时）。核心目标仍为“任务尽快完成、总漏判概率尽量低”，补充次级目标“减小 K 以提升各测试组有效工作时间比（YXB=小组每班次平均测试时间/K），减少人员浪费”。

测试顺序依旧沿用问题 2 验证的最优顺序“C-A-B-E”。设备更换策略为仅当设备“使用满 240 小时”或“发生故障”时更换，不提前更换（避免增加调试时间导致工序中断，影响任务进度）。流程时间与仿真控制为单个装置完整流程（运入+测试+运出）时长固定为 11 小时（0.5+2.5+2.5+2+3+0.5）；仿真轮次设为 100 轮，消除随机误差，各异常事件概率严格遵循题中给定值（如 Y1 中 120 小时内 A 设备故障概率 3%、Y2 中 A 子系统问题概率 2.5%等）。倒班衔接规则为两分队仅交接设备累计使用时间，未完成工序由下一分队重新开始。

5.2 问题分析与建模

相对于问题 2 来说，问题 3 新增了条件“两个分队接续倒班，每个班次工作 K（9≤K≤12）小时”，由于“规定若某个工序测试因故发生中断时，该工序测试不得接续进行，需重新开始测试”，因此两分队间不能对测试任务进行接力。我们先假设 K=12，此时两个分队接续倒班的工作模式和问题 2 仅有一个分队的工作模式是非常相近的，可以简单看作问题 2 的工作天数除以 2 并向上取整。当 K 取其他值时，可以将代码中的工作时间参数调整为其他时间值便可进行仿真模拟。

在问题 3 的背景中，安排两个分队接续倒班的原因是测试任务紧急、需要加快测试进度，因此问题 3 的目标应该为在保证任务完成平均天数没有显著增加的前提下尽量减小每个班次工作时间 K，从而使得各个专业测试组的有效工作时间比取较大值，以减少人员浪费。

由于测试任务安排主要考虑的目标“任务尽快完成，总的漏判概率尽量低”不变，我们仍沿用问题 2 的工作计划，即子系统测试顺序与测试设备更换逻辑。此时，唯一的自变量为每个班次工作时间 K，下面利用改进的程序仿真模拟 K=9-12、步长 0.5 共 7 个 K 值的测试工作，并记录任务完成平均天数（T）、通过测试的装置的平均数目（S）、总漏判概率（P_L）、总误判概率（P_W）、各个专业测试组的有效工作时间比 YXB。

我们在问题 2 分析出结果：当工作时长为 12 小时（对应问题 3 每个班次工作时间 K）的情况下，任务完成平均天数、总的漏判概率仅与测试顺序有关。但在问题 3 中，当每个班次工作时间 K 减小时，一班次内除去大型装置完整测试时间 10 小时、装置运入运出测试大厅时间 1 小时后没有多余时间用于更换并调试校对测试设备。此时测试设备的更换策略变得紧要，因为每次意料之外的由测试故障引起的设备更换都可能导致该班次的剩余工作时间不足以支撑剩余测试任务的进行，从而引起大量的工作时间浪费。

下面进行仿真数据分析：

我们设定测试顺序为问题 2 结论中的最优顺序 C-A-B-E，模拟轮次为 100 轮，记录不同 K 值下不同设备更换策略的任务完成平均天数、通过测试的装置的平均数目、总漏判概率、总误判概率以及各个专业测试组的有效工作时间比，取任务完成平均天数最小的策略为最优策略，记录该策略的输出结果于表格。

表7 不同 K 值下对应的最佳设备更换策略及其仿真结果

K	策略	平均天数	通过数	漏判概率	误判概率	YXB1	YXB2	YXB3	YXB4
9	A0B0C1E1	34.83	92.63	6.35e-05	0.0135	0.4112	0.3221	0.4143	0.4550
9.5	A0B0C0E0	34.05	92.12	6.35e-05	0.0134	0.3973	0.3118	0.4014	0.4370
10	A1B0C0E0	32.08	92.53	6.35e-05	0.0132	0.4009	0.3149	0.4034	0.4433
10.5	A0B1C0E0	30.88	92.81	6.35e-05	0.0132	0.3973	0.3120	0.4005	0.4408
11	A0B0C0E0	26.42	92.79	6.35e-05	0.0137	0.4418	0.3472	0.4482	0.4916
11.5	A0B0C0E0	26.03	92.92	6.35e-05	0.0119	0.4300	0.3379	0.4333	0.4770
12	A0B0C0E0	25.91	92.54	6.35e-05	0.0143	0.4151	0.3279	0.4186	0.4602

我们可以观察发现，当 K=11、11.5、12 时，平均天数的变化值不大；但当 K=10.5 时，相较于 K=11，其平均天数发生明显的上升，这是因为对于某一大型装置完成完整的“运入大厅-C 测试-A 测试-B 测试-E 测试-运出大厅”过程需要 11 小时，当 K<11 时，一定会导致大型装置的完整测试流程无法在一个班次内完成，从而造成时间浪费。除此之外，当 K=11 时，大型装置测试台完全没有冗余时间，如果发生测试设备更换，则一定会将本可以在一班次内完成的测试工作额外增加 1 工作班次；同理，K=11.5 时无法承受 E 设备更换的额外调试校对时间，但能承受 A、B、C 设备的更换。

5.3 结论

因此，根据模型仿真结果，问题 3 最优 K 值为 11.5，其任务完成平均天数 26.03 天与 K=12 时的 25.91 天非常接近，能利用 0.5 小时的冗余时间承受 A、B、C 设备的突发更换，各测试小组的有效工作时间比也有较大提升。由于测试任务安排主要考虑的目标“任务尽快完成，总的漏判概率尽量低”并没有改变，所以测试顺序依旧为问题 2 得到的 C-A-B-E。设备更换策略为仅当设备使用达到 240 小时或发生故障时才进行更换。

表8 问题 3 结果统计指标

T	S	P_L	P_W	YXB1	YXB2	YXB3	YXB4
26.03	92.92	0.0001	0.0119	0.4300	0.3379	0.4333	0.4770

6 问题 4 的分析与模型搭建

基于问题 3 的仿真结果和结论，我们分析了影响测试任务平均完成时间（以下简称“平均时间”）的关键因素，并据此提出改进建议。问题 3 中，我们通过仿真模拟了不同班次工作时间 K（9≤K≤12）下的测试过程，考虑了设备故障、子系统问题、测手差错、设备更换策略等随机事件，最终确定 K=11.5 小时为最优值，平均时间为 26.03 天。以下分析主要基于问题 3 的仿真数据和相关模型。

6.1 因素分析

影响平均时间的主要因素包括班次工作时间 K、设备更换策略、测试顺序、设备故障率、子系统问题概率、测手差错概率、运输时间和调试时间等。这些因素通过影响测试中断、重测次数、设备更换频率等间接影响平均时间。具体分析如下：

1. 班次工作时间 K：K 值直接决定每个班次的有效测试时间。从问题 3 的仿真结果看，当 K<11 时，平均时间显著增加（如 K=9 时平均 34.83 天，K=10.5 时 30.88 天），因为一个装置的完整测试流程（包括运输和测试）需要 11 小时，K<11 导致测试无法在一个班次内完成，造成班次切换时的中断和时间浪费。当 K≥11 时，平均时间明显降低（K=11 时 26.42 天，K=11.5 时 26.03 天，K=12 时 25.91 天），但 K=11.5 在平均时间接近 K=12 的情况下，有效工作时间比（YXB）更高，人员利用率更好。因此，K 值是影响平均时间的最关键因素之一。
2. 设备更换策略：设备更换策略能影响设备故障导致的测试中断时间。问题 3 中，对于不同 K 值，我们采用了不同的策略（如 A0B0C1E1 表示对 C 和 E 设备提前更换）。仿真显示，当 K 较小时（如 K=9），提前更换策略（如提前更换 C 和 E 设备）有助于减少故障中断，从而优化平均时间。但对于 K≥11.5，策略 A0B0C0E0（仅当设备使用达 240 小时或故障时更换）已能较好平衡平均时间和更换成本。策略选择对平均时间的影响在 K 值较小时更为显著。
3. 测试顺序：测试顺序影响测试资源的利用率和并行性，从而影响总时间，同时也会影响总漏判概率，相关计算方法与结论已在问题 2 中进行充分叙述。为了满足题目条件“测试任务安排主要考虑的目标是：任务尽快完成，总的漏判概率尽量低”，问题 3 继续沿用问题 2 的最优顺序 C-A-B-E。
4. 设备故障率：设备故障导致测试中断和设备更换会增加调试时间，从而延长平均时间。问题中设备故障率分为两段（0-120 小时和 120-240 小时），故障率较高时（如

B 设备在 120-240 小时故障率 7%），平均时间可能增加。通过提前更换设备（使用满 120 小时即更换）可以降低故障风险，但会增加更换次数。

5. 子系统问题概率和测手差错概率：子系统问题（A:2.5%, B:3%, C:2%, D:0.1%）和测手差错（误判和漏判各占 50%）导致重测，增加测试时间。最优测试顺序 C-A-B-E 漏判概率为 6.35e-05，但误判概率（P_W）在 0.0119-0.0143 之间波动，误判导致的重测会轻微增加平均时间。
6. 运输时间和调试时间：运输时间（运入和运出各 0.5 小时）和调试时间（A:30min, B:20min, C:20min, E:40min）是固定开销，但班次切换时如果测试中断，这些时间会被浪费。K 值较小时，这些时间占比增加，影响更明显。

在以上因素中，可以调整的有班次工作时间 K、设备更换策略和测试顺序。其中测试顺序已固定为 C-A-B-E，而对于每一个 K 值，我们都可以通过排序找到最优的设备更换策略。因此本题的主要目标是找到最优的 K 值，使得任务完成平均天数和各个专业测试组的有效工作时间比都变得让人能接受。

6.2 最优 K 值的寻找

下面使用 TOPSIS 法分析最优 K 值：

6.2.1 数据正向化处理

由于平均天数是越小越好的指标，而平均有效工作时间比是越大越好的指标，两者“方向不同”，需要进行正向化处理。对于越大越好的指标，常用的正向化方法是取倒数：

$$x_{ij}' = \frac{1}{x_{ij}}$$

其中，x_{ij} 为原始数据，x_{ij}' 为正向化后的数据。

数据正向化处理后结果为：

表9 正向化处理结果

K 值	平均天数	正向化 YXBave
9	34.83	2.496
9.5	34.05	2.585
10	32.08	2.560
10.5	30.88	2.580
11	26.42	2.314
11.5	26.03	2.384
12	25.91	2.466

6.2.2 数据标准化处理

采用向量归一化法进行标准化，消除量纲影响：

$$z_{ij}=\frac{x_{ij}'}{\sqrt{\sum_{i=1}^{n}\left(x_{ij}'\right)^2}}$$

其中，x_{ij}'为原始数据，z_{ij}为标准化后的数据，$n=7$为方案数。

数据标准化处理后结果为：

表10 标准化处理结果

K 值	标准化平均天数	标准化 YXBave
9	0.4353	0.3796
9.5	0.4256	0.3931
10	0.4009	0.3893
10.5	0.3859	0.3923
11	0.3302	0.3519
11.5	0.3253	0.3625
12	0.3238	0.3751

6.2.3 确定正理想解和负理想解

现在所有指标均已转化为越小越好。对于越小越好的指标来说，正理想解为各指标最小值，负理想解为各指标最大值。设标准化后的矩阵为 Z，则：

$$Z^{+}=\left(\min z_{i1},\min z_{i2}\right)=\left(z_1^{+},z_2^{+}\right)$$

$$Z^{-}=\left(\max z_{i1},\max z_{i2}\right)=\left(z_1^{-},z_2^{-}\right)$$

6.2.4 计算各方案与正负理想解的距离

计算各方案与正理想解的距离：

$$D_I^{+}=\sqrt{\sum_{j=1}^{m}\left(z_{ij}-z_j^{+}\right)^2}$$

计算各方案与负理想解的距离：

$$D_I^{-}=\sqrt{\sum_{j=1}^{m}\left(z_{ij}-z_j^{-}\right)^2}$$

其中，$m=2$为指标数。

6.2.5 计算相对距离并排序

$$C_i=\frac{D_i^{-}}{D_i^{+}+D_i^{-}}$$

其中，C_i值越大，表示方案越接近正理想解，性能越好。

排序结果为：

表11 不同 K 值双目标排序结果

K 值	平均天数	YXBave	相对接近度	排序
11	26.42	0.4322	0.946562	1
11.5	26.03	0.41955	0.914197	2
12	25.91	0.40545	0.829477	3
10.5	30.88	0.38765	0.39981	4
10	32.08	0.390625	0.28741	5
9	34.83	0.40065	0.105269	6
9.5	34.05	0.386875	0.081565	7

6.3 结论

每班次工作时间 K 为 11 时，“平均天数-有效比”双目标可以达到相对最优；同时，采用 C-A-B-E 的测试顺序能有效减少漏测概率。

参考文献

[1] 张国跃. 汽车测试中如何运用通用测试仪器解决难题——横河产品在汽车测试中的经典应用 [J]. 国外电子测量技术, 2014, 33(05): 10-3.

[2] 司守奎, 孙兆亮主编；孙玺菁, 周刚, et al. 数学建模算法与应用 [M]. 2015.

[3] 虞晓芬 傅. 多指标综合评价方法综述 [J]. 统计与决策, 2004, (11): 119-21.

[4] 黄会群. 基于熵权-双基点的供应商选择优化模型研究 [J]. 科学技术与工程, 2008, (09): 2411-3+6.

[5] 宋宗耘, 肖鑫利. 基于两阶段决策目标的增量配电网投资决策评价研究 [J]. 项目管理评论, 2021, (04): 56-9.

附　录

代码环境	**操作系统: Windows 11 家庭中文版 (Version 24H2)** 编程语言: Python 3.9.23 (conda 23.5.2) 编辑器: PyCharm 2023.1.6 (Community Edition) 代码详见: /python_work

附录 A: 问题 1 代码

q1.py

```
P_Asys_ques = 0.025
P_Bsys_ques = 0.03
P_Csys_ques = 0.02
P_Dsys_ques = 0.001

# 漏判概率（Y32）
P_Lou_Aer = 0.03 * 0.5 * (2-0.015)
P_Lou_Ber = 0.04 * 0.5 * (2-0.02)
P_Lou_Cer = 0.02 * 0.5 * (2-0.01)
P_error_Eer = 0.02

P_Lou_Asys = P_Asys_ques * P_Lou_Aer
P_Lou_Bsys = P_Bsys_ques * P_Lou_Ber
P_Lou_Csys = P_Csys_ques * P_Lou_Cer
P_Lou_Dsys = P_Dsys_ques

P_Lou_sum = P_Lou_Asys + P_Lou_Bsys + P_Lou_Csys + P_Lou_Dsys
lam_Asys = P_Lou_Asys / P_Lou_sum
lam_Bsys = P_Lou_Bsys / P_Lou_sum
lam_Csys = P_Lou_Csys / P_Lou_sum
lam_Dsys = P_Lou_Dsys / P_Lou_sum
P_JianCe = P_Lou_sum * (1-P_error_Eer) + P_error_Eer * 0.5
print('lam_Asys=', lam_Asys, 'lam_Bsys=', lam_Bsys, 'lam_Csys=', lam_Csys,
'lam_Dsys=', lam_Dsys, 'P_Lou_sum', P_Lou_sum, 'P_JianCe=', P_JianCe)
```

附录 B: 问题 2 代码

q2_data.py

```
P_Asys_ques = 0.025
P_Bsys_ques = 0.03
P_Csys_ques = 0.02
P_Dsys_ques = 0.001

P_GuZhang_Asys = 0.03
P_GuZhang_Bsys = 0.04
P_GuZhang_Csys = 0.02
P_GuZhang_Esys = 0.02
# 漏判概率（Y32）
P_Lou_Aer = P_GuZhang_Asys * 0.5 * ( 2 - 0.5 * P_GuZhang_Asys )
P_Lou_Ber = P_GuZhang_Bsys * 0.5 * ( 2 - 0.5 * P_GuZhang_Bsys )
P_Lou_Cer = P_GuZhang_Csys * 0.5 * ( 2 - 0.5 * P_GuZhang_Csys )
P_Lou_Eer = P_GuZhang_Esys * 0.5 * ( 2 - 0.5 * P_GuZhang_Esys )

# 误判概率（Y31）
P_Wu_Aer = ( 1 - P_Asys_ques ) * ( 0.5 * P_GuZhang_Asys ) ** 2
P_Wu_Ber = ( 1 - P_Bsys_ques ) * ( 0.5 * P_GuZhang_Bsys ) ** 2
P_Wu_Cer = ( 1 - P_Csys_ques ) * ( 0.5 * P_GuZhang_Csys ) ** 2
```

```
P_Wu_Eer = ( 1 - P_Asys_ques ) * ( 1 - P_Bsys_ques ) * ( 1 - P_Csys_ques )
* ( 1 - P_Dsys_ques ) * ( 0.5 * P_GuZhang_Esys ) ** 2

# 通过并进行下一工序概率
P_Guo_Asys = 1 - (P_Asys_ques * (1 - 0.5 * P_GuZhang_Asys) ** 2 + ( 1 -
P_Asys_ques ) * ( 0.5 * P_GuZhang_Asys) ** 2)
P_Guo_Bsys = 1 - (P_Bsys_ques * (1 - 0.5 * P_GuZhang_Bsys) ** 2 + ( 1 -
P_Bsys_ques ) * ( 0.5 * P_GuZhang_Bsys) ** 2)
P_Guo_Csys = 1 - (P_Csys_ques * (1 - 0.5 * P_GuZhang_Csys) ** 2 + ( 1 -
P_Csys_ques ) * ( 0.5 * P_GuZhang_Csys) ** 2)

P_Lou_ABCsys = ( P_Lou_Aer * P_Asys_ques + P_Guo_Asys * P_Lou_Ber *
P_Bsys_ques + P_Guo_Asys * P_Guo_Bsys * P_Lou_Cer * P_Csys_ques +
P_Guo_Asys * P_Guo_Bsys * P_Guo_Csys * P_Dsys_ques ) * P_Lou_Eer
P_Lou_ACBsys = ( P_Lou_Aer * P_Asys_ques + P_Guo_Asys * P_Lou_Cer *
P_Csys_ques + P_Guo_Asys * P_Guo_Csys * P_Lou_Ber * P_Bsys_ques +
P_Guo_Asys * P_Guo_Bsys * P_Guo_Csys * P_Dsys_ques ) * P_Lou_Eer
P_Lou_BACsys = ( P_Lou_Ber * P_Bsys_ques + P_Guo_Bsys * P_Lou_Aer *
P_Asys_ques + P_Guo_Bsys * P_Guo_Asys * P_Lou_Cer * P_Csys_ques +
P_Guo_Asys * P_Guo_Bsys * P_Guo_Csys * P_Dsys_ques ) * P_Lou_Eer
P_Lou_BCAsys = ( P_Lou_Ber * P_Bsys_ques + P_Guo_Bsys * P_Lou_Cer *
P_Csys_ques + P_Guo_Bsys * P_Guo_Csys * P_Lou_Aer * P_Asys_ques +
P_Guo_Asys * P_Guo_Bsys * P_Guo_Csys * P_Dsys_ques ) * P_Lou_Eer
P_Lou_CABsys = ( P_Lou_Cer * P_Csys_ques + P_Guo_Csys * P_Lou_Aer *
P_Asys_ques + P_Guo_Csys * P_Guo_Asys * P_Lou_Ber * P_Bsys_ques +
P_Guo_Asys * P_Guo_Bsys * P_Guo_Csys * P_Dsys_ques ) * P_Lou_Eer
P_Lou_CBAsys = ( P_Lou_Cer * P_Csys_ques + P_Guo_Csys * P_Lou_Ber *
P_Bsys_ques + P_Guo_Csys * P_Guo_Bsys * P_Lou_Aer * P_Asys_ques +
P_Guo_Asys * P_Guo_Bsys * P_Guo_Csys * P_Dsys_ques ) * P_Lou_Eer

P_false_ABC = P_Wu_Aer + (1 - P_Asys_ques) * P_Wu_Ber + (1 - P_Asys_ques) *
(1 - P_Bsys_ques) * P_Wu_Cer + P_Wu_Eer
P_Wu_ACBsys = P_Wu_Aer + (1 - P_Asys_ques) * P_Wu_Cer + (1 - P_Asys_ques) *
(1 - P_Csys_ques) * P_Wu_Ber + P_Wu_Eer
P_Wu_BACsys = P_Wu_Ber + (1 - P_Bsys_ques) * P_Wu_Aer + (1 - P_Bsys_ques) *
(1 - P_Asys_ques) * P_Wu_Cer + P_Wu_Eer
P_Wu_BCAsys = P_Wu_Ber + (1 - P_Bsys_ques) * P_Wu_Cer + (1 - P_Bsys_ques) *
(1 - P_Csys_ques) * P_Wu_Aer + P_Wu_Eer
P_Wu_CABsys = P_Wu_Cer + (1 - P_Csys_ques) * P_Wu_Aer + (1 - P_Csys_ques) *
(1 - P_Asys_ques) * P_Wu_Ber + P_Wu_Eer
P_Wu_CBAsys = P_Wu_Cer + (1 - P_Csys_ques) * P_Wu_Ber + (1 - P_Csys_ques) *
(1 - P_Bsys_ques) * P_Wu_Aer + P_Wu_Eer

print('P_Lou_ABCsys=', P_Lou_ABCsys)
print('P_Lou_ACBsys=', P_Lou_ACBsys)
print('P_Lou_BACsys=', P_Lou_BACsys)
print('P_Lou_BCAsys=', P_Lou_BCAsys)
print('P_Lou_CABsys=', P_Lou_CABsys)
print('P_Lou_CBAsys=', P_Lou_CBAsys)

print('P_false_ABC=', P_false_ABC)
print('P_Wu_ACBsys=', P_Wu_ACBsys)
print('P_Wu_BACsys=', P_Wu_BACsys)
print('P_Wu_BCAsys=', P_Wu_BCAsys)
print('P_Wu_CABsys=', P_Wu_CABsys)
print('P_Wu_CBAsys=', P_Wu_CBAsys)

P_Lou_Asys = P_Asys_ques * P_Lou_Aer
P_Lou_Bsys = P_Bsys_ques * P_Lou_Ber
P_Lou_Csys = P_Csys_ques * P_Lou_Cer
P_Lou_Dsys = P_Dsys_ques
```

```
P_Lou_sum = P_Lou_Asys + P_Lou_Bsys + P_Lou_Csys + P_Lou_Dsys
lam_Asys = P_Lou_Asys / P_Lou_sum
lam_Bsys = P_Lou_Bsys / P_Lou_sum
lam_Csys = P_Lou_Csys / P_Lou_sum
lam_Dsys = P_Lou_Dsys / P_Lou_sum
P_JianCe = P_Lou_sum * (1 - P_GuZhang_Esys) + P_GuZhang_Esys * 0.5
print('lam_Asys=', lam_Asys, 'lam_Bsys=', lam_Bsys, 'lam_Csys=', lam_Csys,
'lam_Dsys=', lam_Dsys, 'P_Lou_sum=', P_Lou_sum, 'P_JianCe=', P_JianCe)
```

q2order.py

```
import numpy as np
import simpy
import random
from collections import defaultdict
import time

# 设置六种测试顺序
TEST_SHUNXU = {
    'ABCE': ['A', 'B', 'C', 'E'],
    'ACBE': ['A', 'C', 'B', 'E'],
    'BACE': ['B', 'A', 'C', 'E'],
    'BCAE': ['B', 'C', 'A', 'E'],
    'CABE': ['C', 'A', 'B', 'E'],
    'CBAE': ['C', 'B', 'A', 'E']
}


def Ques2_FangZhen(test_ShunXu):
    # 参数设置
    num_unit = 100
    YunShu_time = 1
    CeShi_times = {'A': 2.5, 'B': 2.0, 'C': 2.5, 'E': 3.0}
    TiaoShi_times = {'A': 0.5, 'B': 1 / 3, 'C': 1 / 3, 'E': 2 / 3}

    # 设备故障概率（累积概率）
    P_GuZhang_Short = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.03}
    P_GuZhang_Long = {'A': 0.05, 'B': 0.07, 'C': 0.06, 'E': 0.05}

    P_Ques_Sys = {'A': 0.025, 'B': 0.03, 'C': 0.02, 'D': 0.001}
    P_error_er = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.02}
    work_time = 12

    # 初始化仿真环境
    env = simpy.Environment()
    test_bigtables = simpy.Resource(env, capacity=2)
    work_smalltable = {
       'A': simpy.Resource(env, capacity=1),
       'B': simpy.Resource(env, capacity=1),
       'C': simpy.Resource(env, capacity=1),
       'E': simpy.Resource(env, capacity=1)
    }

    # 定义全局统计变量
    statis_var = {
       'start_time': 0,
       'end_time': 0,
       'Guo_count': 0,
       'sum_Lou_count': 0,
       'sum_Wu_count': 0,
       'sum_CeShi_count': 0,
```

```
        'ZuBie_times': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'TiaoShi_times': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'YunShu_times': 0,
        'unit_use_time': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'unit_over': 0,
        'over_times': [],
        'units_changes': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'num_Lou_unit': 0
    }

    # 计算设备故障概率的函数
    def JiSuan_GuZhangprob_Sys(test_name, test_time):
        use_time = statis_var['unit_use_time'][test_name]
        delt_t = test_time

        if use_time < 120:
            # 使用时间在 0-120 小时范围内
            H_ut = P_GuZhang_Short[test_name] * (use_time / 120)
            H_ut_delt_t = P_GuZhang_Short[test_name] * ((use_time + delt_t) /
120)
        elif use_time < 240:
            # 使用时间在 120-240 小时范围内
            H_ut = P_GuZhang_Short[test_name] + (P_GuZhang_Long[test_name] -
P_GuZhang_Short[test_name]) * (
                    (use_time - 120) / 120)
            H_ut_delt_t = P_GuZhang_Short[test_name] +
(P_GuZhang_Long[test_name] - P_GuZhang_Short[test_name]) * (
                    (use_time + delt_t - 120) / 120)
        else:
            return 1.0  # 使用时间达到 240 小时，必须更换设备

        if H_ut >= 1.0:
            return 1.0

        # 故障概率 = [F(use_time+delt_t) - F(use_time)] / [1 - F(use_time)]
        P_GuZhang = (H_ut_delt_t - H_ut) / (1 - H_ut)
        return min(P_GuZhang, 1.0)

    # 设备更换和调试过程
    def change_and_tiaoshi_device(env, test_name, statis_var):
        statis_var['unit_use_time'][test_name] = 0

        TiaoShi_time = TiaoShi_times[test_name]
        yield env.timeout(TiaoShi_time)
        statis_var['TiaoShi_times'][test_name] += TiaoShi_time

    # 装置的测试过程
    def test_process(env, device_id, test_bigtables, work_smalltable, sta-
tis_var):
        ZhenShi_CuoWu = {
            'A': np.random.rand() < P_Ques_Sys['A'],
            'B': np.random.rand() < P_Ques_Sys['B'],
            'C': np.random.rand() < P_Ques_Sys['C'],
            'D': np.random.rand() < P_Ques_Sys['D']
        }

        yield env.timeout(YunShu_time)
        statis_var['YunShu_times'] += YunShu_time

        with test_bigtables.request() as bench_request:
```

```
        yield bench_request
        ceshis = test_ShunXu.copy()
        test_try = {ceshi: 0 for ceshi in ceshis}

        test_JieGuo = {}
        unit_change = False

        while ceshis and not unit_change:
           test_name = ceshis.pop(0)
           try_count = test_try[test_name]

           if try_count >= 2:
              test_JieGuo[test_name] = False
              unit_change = True
              continue

           with work_smalltable[test_name].request() as worktable_Xuqiu:
              yield worktable_Xuqiu

              # 获取当前班次剩余工作时间
              DangQian_time = env.now
              ZhuanHuan_start = (DangQian_time // work_time) * work_time
              ZhuanHuan_end = ZhuanHuan_start + work_time
              ShenYu_time = ZhuanHuan_end - DangQian_time

              if ShenYu_time < CeShi_times[test_name]:
                 yield env.timeout(ShenYu_time)
                 ceshis.insert(0, test_name)
                 continue

              # 检查设备故障概率
              P_GuZhang = JiSuan_GuZhangprob_Sys(test_name, Ce-
Shi_times[test_name])
              if np.random.rand() < P_GuZhang:
                 statis_var['units_changes'][test_name] += 1
                 yield from change_and_tiaoshi_device(env, test_name, sta-
tis_var)
                 ceshis.insert(0, test_name)
                 continue

              # 没有故障，进行测试
              yield env.timeout(CeShi_times[test_name])
              statis_var['unit_use_time'][test_name] += Ce-
Shi_times[test_name]
              statis_var['ZuBie_times'][test_name] += CeShi_times[test_name]
              statis_var['sum_CeShi_count'] += 1

              # 检查测手差错
              CuoWu_er = np.random.rand() < P_error_er[test_name]
              if CuoWu_er:
                 CuoWu_LeiXing = np.random.rand()
                 if CuoWu_LeiXing < 0.5:
                    # 误判
                    statis_var['sum_Wu_count'] += 1
                    test_JieGuo[test_name] = False
                    test_try[test_name] += 1
                    ceshis.insert(0, test_name)
                 else:
                    # 漏判
                    statis_var['sum_Lou_count'] += 1
```

```
                    test_JieGuo[test_name] = True
                else:
                    # 无测手差错，检查真实问题
                    if test_name in ['A', 'B', 'C']:
                        if ZhenShi_CuoWu[test_name]:
                            test_JieGuo[test_name] = False
                            test_try[test_name] += 1
                            ceshis.insert(0, test_name)
                        else:
                            test_JieGuo[test_name] = True
                    else:
                        # 检查是否有任何问题（A、B、C 漏检或 D 有问题）
                        RenYi_ques = ZhenShi_CuoWu['A'] or ZhenShi_CuoWu['B'] or
ZhenShi_CuoWu['C'] or ZhenShi_CuoWu['D']

                        if RenYi_ques:
                            test_JieGuo[test_name] = False
                            test_try[test_name] += 1
                            ceshis.insert(0, test_name)
                        else:
                            test_JieGuo[test_name] = True

            # 检查装置是否通过所有测试
            Guo_all = all(test_JieGuo.values()) if test_JieGuo else False

            if Guo_all:
                statis_var['Guo_count'] += 1

                if ZhenShi_CuoWu['A'] or ZhenShi_CuoWu['B'] or
ZhenShi_CuoWu['C'] or ZhenShi_CuoWu['D']:
                    statis_var['num_Lou_unit'] += 1

            over_time = env.now
            statis_var['over_times'].append(over_time)

            statis_var['unit_over'] += 1
            if statis_var['unit_over'] < num_unit:
                yield env.timeout(YunShu_time)
                statis_var['YunShu_times'] += YunShu_time

    # 创建装置测试过程
    for i in range(2):
        env.process(test_process(env, i, test_bigtables, work_smalltable,
statis_var))

    def yanchi_initiate(env, device_id, delay):
        yield env.timeout(delay)
        env.process(test_process(env, device_id, test_bigtables,
work_smalltable, statis_var))

    for i in range(2, num_unit):
        delay = YunShu_time * (i - 1)
        env.process(yanchi_initiate(env, i, delay))

    env.run()
    statis_var['end_time'] = max(statis_var['over_times']) if sta-
tis_var['over_times'] else 0
    sum_days = np.ceil(statis_var['end_time'] / work_time)
    Guo_count = statis_var['Guo_count']

    # 漏判概率获取
```

```
    P_Lou = statis_var['num_Lou_unit'] / num_unit if num_unit > 0 else 0

    # 误判概率获取
    P_Wu = statis_var['sum_Wu_count'] / statis_var['sum_CeShi_count'] if
statis_var['sum_CeShi_count'] > 0 else 0

    # 获取有效工作时间比（YXB）
    sum_shift = np.ceil(statis_var['end_time'] / work_time)
    yxb = {}
    for ceshi in ['A', 'B', 'C', 'E']:
        sum_work_hour = statis_var['ZuBie_times'][ceshi] + sta-
tis_var['TiaoShi_times'][ceshi]
        yxb[ceshi] = (sum_work_hour / sum_shift) / work_time

    return {
        'sum_days': sum_days,
        'Guo_count': Guo_count,
        'P_Lou': P_Lou,
        'P_Wu': P_Wu,
        'yxb': yxb,
        'over_times': statis_var['over_times'],
        'units_changes': statis_var['units_changes']
    }


# 通过多次仿真来求取平均
def DuoCi_FangZhenPingJun(test_ShunXu, n_simulations=10):
    results = []
    for i in range(n_simulations):
        print(f"正在进行第 {i + 1} 次仿真...")
        result = Ques2_FangZhen(test_ShunXu)
        results.append(result)

    # 求取平均值
    PJ_sum_day = np.mean([r['sum_days'] for r in results])
    PJ_Guo_count = np.mean([r['Guo_count'] for r in results])
    PJ_P_Lou = np.mean([r['P_Lou'] for r in results])
    PJ_P_Wu = np.mean([r['P_Wu'] for r in results])

    # 求取 YXB 的平均值
    PJ_yxb = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    for r in results:
        for ceshi in ['A', 'B', 'C', 'E']:
            PJ_yxb[ceshi] += r['yxb'][ceshi]
    for ceshi in ['A', 'B', 'C', 'E']:
        PJ_yxb[ceshi] /= n_simulations

    # 求取设备更换次数的平均值
    PJ_change_count = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    for r in results:
        for ceshi in ['A', 'B', 'C', 'E']:
            PJ_change_count[ceshi] += r['units_changes'][ceshi]
    for ceshi in ['A', 'B', 'C', 'E']:
        PJ_change_count[ceshi] /= n_simulations

    return {
        'PJ_sum_day': PJ_sum_day,
        'PJ_Guo_count': PJ_Guo_count,
        'PJ_P_Lou': PJ_P_Lou,
        'PJ_P_Wu': PJ_P_Wu,
```

```
        'PJ_yxb': PJ_yxb,
        'PJ_change_count': PJ_change_count
    }


def main():
    n_simulations = 100  # 每种顺序的仿真次数
    all_results = {}

    # 对每种测试顺序进行仿真
    for ShunXu_name, ShunXu in TEST_SHUNXU.items():
        print(f"\n开始仿真测试顺序: {ShunXu_name}")
        start_time = time.time()

        results = DuoCi_FangZhenPingJun(ShunXu, n_simulations)
        all_results[ShunXu_name] = results

        end_time = time.time()
        print(f"测试顺序 {ShunXu_name} 完成，耗时: {end_time - start_time:.2f} 秒
")

    print("\n" + "=" * 80)
    print("六种测试顺序的仿真结果比较")
    print("=" * 80)

    print(
        f"{'测试顺序':<8} {'平均天数(T)':<12} {'通过数(S)':<10} {'漏判概率
(PL)':<14} {'误判概率(PW)':<14} {'YXB_A':<8} {'YXB_B':<8} {'YXB_C':<8}
{'YXB_E':<8}")
    print("-" * 80)

    for ShunXu_name, results in all_results.items():
        print(f"{ShunXu_name:<8} {results['PJ_sum_day']:<12.2f} {re-
sults['PJ_Guo_count']:<10.2f} "
              f"{results['PJ_P_Lou']:<14.4f} {results['PJ_P_Wu']:<14.4f} "
              f"{results['PJ_yxb']['A']:<8.4f} {results['PJ_yxb']['B']:<8.4f} "
              f"{results['PJ_yxb']['C']:<8.4f} {results['PJ_yxb']['E']:<8.4f}")

    best_ShunXu = min(all_results.items(), key=lambda x: x[1]['PJ_sum_day'])
    print(f"\n最优测试顺序: {best_ShunXu[0]} (平均完成天数:
{best_ShunXu[1]['PJ_sum_day']:.2f} 天)")

    best_Lou_ShunXu = min(all_results.items(), key=lambda x:
x[1]['PJ_P_Lou'])
    print(f"漏判概率最低的顺序: {best_Lou_ShunXu[0]} (漏判概率:
{best_Lou_ShunXu[1]['PJ_P_Lou']:.4f})")

    print("\n设备平均更换次数:")
    for ShunXu_name, results in all_results.items():
        print(f"{ShunXu_name}: A={results['PJ_change_count']['A']:.2f},
B={results['PJ_change_count']['B']:.2f}, "
              f"C={results['PJ_change_count']['C']:.2f}, E={re-
sults['PJ_change_count']['E']:.2f}")


# 运行主函数
if __name__ == "__main__":
    main()
```

q2order01.py

```
import numpy as np
import simpy
import random
from collections import defaultdict
import time
import itertools

# 设置六种测试顺序
TEST_SHUNXU = {
    'ABCE': ['A', 'B', 'C', 'E'],
    'ACBE': ['A', 'C', 'B', 'E'],
    'BACE': ['B', 'A', 'C', 'E'],
    'BCAE': ['B', 'C', 'A', 'E'],
    'CABE': ['C', 'A', 'B', 'E'],
    'CBAE': ['C', 'B', 'A', 'E']
}

UNIT = ['A', 'B', 'C', 'E']


def Ques2_FangZhen(test_ShunXu, replace_strategy):
    # 参数设置
    num_unit = 100
    YunShu_time = 1
    CeShi_times = {'A': 2.5, 'B': 2.0, 'C': 2.5, 'E': 3.0}
    TiaoShi_times = {'A': 0.5, 'B': 1 / 3, 'C': 1 / 3, 'E': 2 / 3}

    # 设备故障概率（累积概率）
    P_GuZhang_Short = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.03}
    P_GuZhang_Long = {'A': 0.05, 'B': 0.07, 'C': 0.06, 'E': 0.05}

    P_Ques_Sys = {'A': 0.025, 'B': 0.03, 'C': 0.02, 'D': 0.001}
    P_error_er = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.02}
    work_time = 12

    # 初始化仿真环境
    env = simpy.Environment()
    test_bigtables = simpy.Resource(env, capacity=2)
    work_smalltable = {
       'A': simpy.Resource(env, capacity=1),
       'B': simpy.Resource(env, capacity=1),
       'C': simpy.Resource(env, capacity=1),
       'E': simpy.Resource(env, capacity=1)
    }

    # 定义全局统计变量
    statis_var = {
       'start_time': 0,
       'end_time': 0,
       'Guo_count': 0,
       'sum_Lou_count': 0,
       'sum_Wu_count': 0,
       'sum_CeShi_count': 0,
       'ZuBie_times': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
       'TiaoShi_times': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
       'YunShu_times': 0,
       'unit_use_time': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
       'unit_over': 0,
       'over_times': [],
```

```
        'units_changes': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'num_Lou_unit': 0,
        'preventive_replacements': {'A': 0, 'B': 0, 'C': 0, 'E': 0}  # 预防性更换次数
    }

    # 计算设备故障概率的函数
    def JiSuan_GuZhangprob_Sys(test_name, test_time):
        use_time = statis_var['unit_use_time'][test_name]
        delt_t = test_time

        if use_time < 120:
            # 使用时间在 0-120 小时范围内
            H_ut = P_GuZhang_Short[test_name] * (use_time / 120)
            H_ut_delt_t = P_GuZhang_Short[test_name] * ((use_time + delt_t) / 120)
        elif use_time < 240:
            # 使用时间在 120-240 小时范围内
            H_ut = P_GuZhang_Short[test_name] + (P_GuZhang_Long[test_name] - P_GuZhang_Short[test_name]) * (
                    (use_time - 120) / 120)
            H_ut_delt_t = P_GuZhang_Short[test_name] + (P_GuZhang_Long[test_name] - P_GuZhang_Short[test_name]) * (
                    (use_time + delt_t - 120) / 120)
        else:
            return 1.0  # 使用时间达到 240 小时，必须更换设备

        if H_ut >= 1.0:
            return 1.0

        # 故障概率 = [F(use_time+delt_t) - F(use_time)] / [1 - F(use_time)]
        P_GuZhang = (H_ut_delt_t - H_ut) / (1 - H_ut)
        return min(P_GuZhang, 1.0)

    # 检查设备是否需要更换
    def JianCha_unit_change(test_name):
        usage = statis_var['unit_use_time'][test_name]
        strategy = replace_strategy[test_name]

        if strategy == 0:
            return usage >= 240
        else:
            return usage >= 120

    # 设备更换和调试过程
    def change_and_tiaoshi_device(env, test_name, statis_var, is_preventive=False):
        if is_preventive:
            statis_var['preventive_replacements'][test_name] += 1
        else:
            statis_var['units_changes'][test_name] += 1

        statis_var['unit_use_time'][test_name] = 0

        TiaoShi_time = TiaoShi_times[test_name]
        yield env.timeout(TiaoShi_time)
        statis_var['TiaoShi_times'][test_name] += TiaoShi_time

    # 装置的测试过程
```

```
    def test_process(env, device_id, test_bigtables, work_smalltable, sta-
tis_var):
      ZhenShi_CuoWu = {
        'A': np.random.rand() < P_Ques_Sys['A'],
        'B': np.random.rand() < P_Ques_Sys['B'],
        'C': np.random.rand() < P_Ques_Sys['C'],
        'D': np.random.rand() < P_Ques_Sys['D']
      }

      yield env.timeout(YunShu_time)
      statis_var['YunShu_times'] += YunShu_time

      with test_bigtables.request() as bench_request:
        yield bench_request
        ceshis = test_ShunXu.copy()
        test_try = {ceshi: 0 for ceshi in ceshis}

        test_JieGuo = {}
        unit_change = False

        while ceshis and not unit_change:
          test_name = ceshis.pop(0)
          try_count = test_try[test_name]

          if try_count >= 2:
            # 已尝试 2 次仍未通过，装置退出
            test_JieGuo[test_name] = False
            unit_change = True
            continue

          with work_smalltable[test_name].request() as worktable_Xuqiu:
            yield worktable_Xuqiu

            if JianCha_unit_change(test_name):
              yield from change_and_tiaoshi_device(env, test_name, sta-
tis_var, is_preventive=True)
              ceshis.insert(0, test_name)
              continue

            # 获取当前班次剩余工作时间
            DangQian_time = env.now
            ZhuanHuan_start = (DangQian_time // work_time) * work_time
            ZhuanHuan_end = ZhuanHuan_start + work_time
            ShenYu_time = ZhuanHuan_end - DangQian_time

            if ShenYu_time < CeShi_times[test_name]:
              yield env.timeout(ShenYu_time)
              ceshis.insert(0, test_name)
              continue

            # 检查设备故障概率
            P_GuZhang = JiSuan_GuZhangprob_Sys(test_name, Ce-
Shi_times[test_name])
            if np.random.rand() < P_GuZhang:
              yield from change_and_tiaoshi_device(env, test_name, sta-
tis_var, is_preventive=False)
              ceshis.insert(0, test_name)
              continue

            # 没有故障，进行测试
            yield env.timeout(CeShi_times[test_name])
```

```
                statis_var['unit_use_time'][test_name] += Ce-
Shi_times[test_name]
                statis_var['ZuBie_times'][test_name] += CeShi_times[test_name]
                statis_var['sum_CeShi_count'] += 1

                # 检查测手差错
                CuoWu_er = np.random.rand() < P_error_er[test_name]
                if CuoWu_er:
                    CuoWu_LeiXing = np.random.rand()
                    if CuoWu_LeiXing < 0.5:
                        # 误判
                        statis_var['sum_Wu_count'] += 1
                        test_JieGuo[test_name] = False
                        test_try[test_name] += 1
                        ceshis.insert(0, test_name)
                    else:
                        # 漏判
                        statis_var['sum_Lou_count'] += 1
                        test_JieGuo[test_name] = True
                else:
                    # 无测手差错，检查真实问题
                    if test_name in ['A', 'B', 'C']:
                        if ZhenShi_CuoWu[test_name]:
                            test_JieGuo[test_name] = False
                            test_try[test_name] += 1
                            ceshis.insert(0, test_name)
                        else:
                            test_JieGuo[test_name] = True
                    else:
                        # 检查是否有任何问题（A、B、C 漏检或 D 有问题）
                        RenYi_ques = ZhenShi_CuoWu['A'] or ZhenShi_CuoWu['B'] or
ZhenShi_CuoWu['C'] or ZhenShi_CuoWu['D']

                        if RenYi_ques:
                            test_JieGuo[test_name] = False
                            test_try[test_name] += 1
                            ceshis.insert(0, test_name)
                        else:
                            test_JieGuo[test_name] = True

            # 检查装置是否通过所有测试
            Guo_all = all(test_JieGuo.values()) if test_JieGuo else False

            if Guo_all:
                statis_var['Guo_count'] += 1

                if ZhenShi_CuoWu['A'] or ZhenShi_CuoWu['B'] or
ZhenShi_CuoWu['C'] or ZhenShi_CuoWu['D']:
                    statis_var['num_Lou_unit'] += 1

            over_time = env.now
            statis_var['over_times'].append(over_time)

            statis_var['unit_over'] += 1
            if statis_var['unit_over'] < num_unit:
                yield env.timeout(YunShu_time)
                statis_var['YunShu_times'] += YunShu_time

    # 创建装置测试过程
    for i in range(2):
```

```
        env.process(test_process(env, i, test_bigtables, work_smalltable,
statis_var))

    def yanchi_initiate(env, device_id, delay):
        yield env.timeout(delay)
        env.process(test_process(env, device_id, test_bigtables,
work_smalltable, statis_var))

    for i in range(2, num_unit):
        delay = YunShu_time * (i - 1)
        env.process(yanchi_initiate(env, i, delay))

    env.run()
    statis_var['end_time'] = max(statis_var['over_times']) if sta-
tis_var['over_times'] else 0
    sum_days = np.ceil(statis_var['end_time'] / work_time)
    Guo_count = statis_var['Guo_count']

    # 漏判概率获取
    P_Lou = statis_var['num_Lou_unit'] / num_unit if num_unit > 0 else 0

    # 误判概率获取
    P_Wu = statis_var['sum_Wu_count'] / statis_var['sum_CeShi_count'] if
statis_var['sum_CeShi_count'] > 0 else 0

    # 获取有效工作时间比（YXB）
    sum_shift = np.ceil(statis_var['end_time'] / work_time)
    yxb = {}
    for ceshi in ['A', 'B', 'C', 'E']:
        sum_work_hour = statis_var['ZuBie_times'][ceshi] + sta-
tis_var['TiaoShi_times'][ceshi]
        yxb[ceshi] = (sum_work_hour / sum_shift) / work_time

    return {
        'sum_days': sum_days,
        'Guo_count': Guo_count,
        'P_Lou': P_Lou,
        'P_Wu': P_Wu,
        'yxb': yxb,
        'units_changes': statis_var['units_changes'],
        'preventive_replacements': statis_var['preventive_replacements'],
        'total_replacements': {
            ceshi: statis_var['units_changes'][ceshi] + statis_var['preven-
tive_replacements'][ceshi]
            for ceshi in UNIT
        }
    }


# 通过多次仿真来求取平均
def DuoCi_FangZhenPingJun(test_ShunXu, replace_strategy, n_simulations=5):
    results = []
    for i in range(n_simulations):
        result = Ques2_FangZhen(test_ShunXu, replace_strategy)
        results.append(result)

    # 求取平均值
    PJ_sum_day = np.mean([r['sum_days'] for r in results])
    PJ_Guo_count = np.mean([r['Guo_count'] for r in results])
    PJ_P_Lou = np.mean([r['P_Lou'] for r in results])
    PJ_P_Wu = np.mean([r['P_Wu'] for r in results])
```

```
    # 求取 YXB 的平均值
    PJ_yxb = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    for r in results:
        for ceshi in ['A', 'B', 'C', 'E']:
            PJ_yxb[ceshi] += r['yxb'][ceshi]
    for ceshi in ['A', 'B', 'C', 'E']:
        PJ_yxb[ceshi] /= n_simulations

    # 求取设备更换次数的平均值
    PJ_change_count = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    avg_preventive_replacements = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    avg_total_replacements = {'A': 0, 'B': 0, 'C': 0, 'E': 0}

    for r in results:
        for ceshi in UNIT:
            PJ_change_count[ceshi] += r['units_changes'][ceshi]
            avg_preventive_replacements[ceshi] += r['preventive_replace-
ments'][ceshi]
            avg_total_replacements[ceshi] += r['total_replacements'][ceshi]

    for ceshi in UNIT:
        PJ_change_count[ceshi] /= n_simulations
        avg_preventive_replacements[ceshi] /= n_simulations
        avg_total_replacements[ceshi] /= n_simulations

    return {
        'PJ_sum_day': PJ_sum_day,
        'PJ_Guo_count': PJ_Guo_count,
        'PJ_P_Lou': PJ_P_Lou,
        'PJ_P_Wu': PJ_P_Wu,
        'PJ_yxb': PJ_yxb,
        'PJ_change_count': PJ_change_count,
        'avg_preventive_replacements': avg_preventive_replacements,
        'avg_total_replacements': avg_total_replacements,
        'replace_strategy': replace_strategy
    }


# 生成所有可能的设备更换策略组合
def ShengCheng_all_strategies():
    strategies = []
    for strategy in itertools.product([0, 1], repeat=4):
        strategy_dict = {
            'A': strategy[0],
            'B': strategy[1],
            'C': strategy[2],
            'E': strategy[3]
        }
        strategies.append(strategy_dict)
    return strategies


def main():
    n_simulations = 100  # 每种组合的仿真次数
    test_order_name = 'CABE'  # 选择一种测试顺序进行设备策略比较
    all_strategies = ShengCheng_all_strategies()
    all_results = {}

    print(f"开始仿真测试顺序: {test_order_name}")
```

```
    print(f"共有 {len(all_strategies)} 种设备更换策略组合")

    # 对每种设备更换策略进行仿真
    for i, strategy in enumerate(all_strategies):
        print(f"正在仿真策略组合 {i + 1}/{len(all_strategies)}: {strategy}")
        start_time = time.time()

        results = DuoCi_FangZhenPingJun(TEST_SHUNXU[test_order_name], strat-
egy, n_simulations)
        strategy_key = f"A{strategy['A']}B{strategy['B']}C{strat-
egy['C']}E{strategy['E']}"
        all_results[strategy_key] = results

        end_time = time.time()
        print(f"策略组合 {strategy_key} 完成，耗时: {end_time - start_time:.2f}
秒")

    print("\n" + "=" * 120)
    print(f"测试顺序 {test_order_name} 的 16 种设备更换策略仿真结果比较")
    print("=" * 120)

    header = f"{'策略':<8} {'平均天数':<8} {'通过数':<8} {'漏判概率':<10} {'误判
概率':<10} "
    header += " ".join([f"{'YXB_' + d:<8}" for d in UNIT])

    print(header)
    print("-" * 120)

    for strategy_key, results in all_results.items():
        line = f"{strategy_key:<8} {results['PJ_sum_day']:<8.2f} {re-
sults['PJ_Guo_count']:<8.2f} "
        line += f"{results['PJ_P_Lou']:<10.4f} {results['PJ_P_Wu']:<10.4f} "

        for device in UNIT:
            line += f"{results['PJ_yxb'][device]:<8.4f} "

        print(line)

    print("\n" + "=" * 80)
    print("设备更换次数统计")
    print("=" * 80)

    header2 = f"{'策略':<8} "
    header2 += " ".join([f"{'R_' + d:<6}" for d in UNIT])
    header2 += " ".join([f"{'P_' + d:<6}" for d in UNIT])
    header2 += " ".join([f"{'T_' + d:<6}" for d in UNIT])

    print(header2)
    print("-" * 80)

    for strategy_key, results in all_results.items():
        line = f"{strategy_key:<8} "

        for device in UNIT:
            line += f"{results['PJ_change_count'][device]:<6.1f} "

        for device in UNIT:
            line += f"{results['avg_preventive_replacements'][device]:<6.1f} "

        for device in UNIT:
```

```
        line += f"{results['avg_total_replacements'][device]:<6.1f} "

    print(line)

best_strategy = min(all_results.items(), key=lambda x:
x[1]['PJ_sum_day'])
print(f"\n最优设备更换策略: {best_strategy[0]} (平均完成天数: {best_strat-
egy[1]['PJ_sum_day']:.2f} 天)")

best_leak_strategy = min(all_results.items(), key=lambda x:
x[1]['PJ_P_Lou'])
print(f"漏判概率最低的策略: {best_leak_strategy[0]} (漏判概率:
{best_leak_strategy[1]['PJ_P_Lou']:.4f})")

best_yxb_strategy = max(all_results.items(), key=lambda x:
sum(x[1]['PJ_yxb'].values()))
PJ_yxb = sum(best_yxb_strategy[1]['PJ_yxb'].values()) / 4
print(f"设备利用率最高的策略: {best_yxb_strategy[0]} (平均 YXB:
{PJ_yxb:.4f})")


# 运行主函数
if __name__ == "__main__":
    main()
```

q2topsis.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import font_manager

try:
    plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
    plt.rcParams['axes.unicode_minus'] = False
except:
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False

# 从 Excel 文件读取数据
filename = 'A2.xlsx'
sheet_name = 'Sheet1'

data = pd.read_excel(filename, sheet_name=sheet_name)
test_order = data['测试顺序']
ShuZhi_data = data[['平均天数(T)', '漏判概率(PL)']].values


n, m = ShuZhi_data.shape
stand_data = ShuZhi_data / np.sqrt(np.sum(ShuZhi_data**2, axis=0))

# 获取正理想解和负理想解
Zheng_solution = np.min(stand_data, axis=0)
Fu_solution = np.max(stand_data, axis=0)

# 求得各方案与正负理想解的距离
dist_Zheng = np.sqrt(np.sum((stand_data - Zheng_solution)**2, axis=1))
dist_Fu = np.sqrt(np.sum((stand_data - Fu_solution)**2, axis=1))

rela_prox = dist_Fu / (dist_Zheng + dist_Fu)
ShunXu_index = np.argsort(-rela_prox)
```

```
JieGuo_dataf = pd.DataFrame({
    '测试顺序': test_order,
    '平均天数': ShuZhi_data[:, 0],
    '漏判概率': ShuZhi_data[:, 1],
    '相对接近度': rela_prox
})

JieGuo_dataf['排名'] = 0
JieGuo_dataf.loc[ShunXu_index, '排名'] = range(1, n+1)
JieGuo_dataf = JieGuo_dataf.sort_values('排名')

print("TOPSIS 分析结果:")
print(JieGuo_dataf.to_string(index=False))
```

附录 C：问题 3 代码

q3order.py

```
import numpy as np
import simpy
import random
from collections import defaultdict
import time

# 设置六种测试顺序
TEST_SHUNXU = {
    'ABCE': ['A', 'B', 'C', 'E'],
    'ACBE': ['A', 'C', 'B', 'E'],
    'BACE': ['B', 'A', 'C', 'E'],
    'BCAE': ['B', 'C', 'A', 'E'],
    'CABE': ['C', 'A', 'B', 'E'],
    'CBAE': ['C', 'B', 'A', 'E']
}


def ques3_FangZhen(test_ShunXu, K=12):
    # 参数设置
    num_unit = 100
    YunShu_time = 1
    CeShi_times = {'A': 2.5, 'B': 2.0, 'C': 2.5, 'E': 3.0}
    TiaoShi_times = {'A': 0.5, 'B': 1 / 3, 'C': 1 / 3, 'E': 2 / 3}

    # 设备故障概率（累积概率）
    P_GuZhang_Short = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.03}
    P_GuZhang_Long = {'A': 0.05, 'B': 0.07, 'C': 0.06, 'E': 0.05}

    P_Ques_Sys = {'A': 0.025, 'B': 0.03, 'C': 0.02, 'D': 0.001}
    P_error_er = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.02}
    work_time = K  # 每班次的工作时间（问题 3 中 K=12）

    # 初始化仿真环境
    env = simpy.Environment()
    test_bigtables = simpy.Resource(env, capacity=2)
    work_smalltable = {
       'A': simpy.Resource(env, capacity=1),
       'B': simpy.Resource(env, capacity=1),
       'C': simpy.Resource(env, capacity=1),
       'E': simpy.Resource(env, capacity=1)
    }
```

```
    # 定义全局统计变量
    statis_var = {
        'start_time': 0,
        'end_time': 0,
        'Guo_count': 0,
        'sum_Lou_count': 0,
        'sum_Wu_count': 0,
        'sum_CeShi_count': 0,
        'ZuBie_times': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'TiaoShi_times': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'YunShu_times': 0,
        'unit_use_time': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'unit_over': 0,
        'over_times': [],
        'units_changes': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'num_Lou_unit': 0
    }

    # 计算设备故障概率的函数
    def JiSuan_GuZhangprob_Sys(test_name, test_time):
        use_time = statis_var['unit_use_time'][test_name]
        delt_t = test_time

        if use_time < 120:
            # 使用时间在 0-120 小时范围内
            H_ut = P_GuZhang_Short[test_name] * (use_time / 120)
            H_ut_delt_t = P_GuZhang_Short[test_name] * ((use_time + delt_t) /
120)
        elif use_time < 240:
            # 使用时间在 120-240 小时范围内
            H_ut = P_GuZhang_Short[test_name] + (P_GuZhang_Long[test_name] -
P_GuZhang_Short[test_name]) * (
                    (use_time - 120) / 120)
            H_ut_delt_t = P_GuZhang_Short[test_name] +
(P_GuZhang_Long[test_name] - P_GuZhang_Short[test_name]) * (
                    (use_time + delt_t - 120) / 120)
        else:
            return 1.0  # 使用时间达到 240 小时，必须更换设备

        if H_ut >= 1.0:
            return 1.0

        # 故障概率 = [F(use_time+delt_t) - F(use_time)] / [1 - F(use_time)]
        P_GuZhang = (H_ut_delt_t - H_ut) / (1 - H_ut)
        return min(P_GuZhang, 1.0)

    # 设备更换和调试过程
    def change_and_tiaoshi_device(env, test_name, statis_var):
        statis_var['unit_use_time'][test_name] = 0

        TiaoShi_time = TiaoShi_times[test_name]
        yield env.timeout(TiaoShi_time)
        statis_var['TiaoShi_times'][test_name] += TiaoShi_time

    # 装置测试过程
    def test_process(env, device_id, test_bigtables, work_smalltable, sta-
tis_var):
        ZhenShi_CuoWu = {
            'A': np.random.rand() < P_Ques_Sys['A'],
```

```
            'B': np.random.rand() < P_Ques_Sys['B'],
            'C': np.random.rand() < P_Ques_Sys['C'],
            'D': np.random.rand() < P_Ques_Sys['D']
        }

        yield env.timeout(YunShu_time)
        statis_var['YunShu_times'] += YunShu_time

        with test_bigtables.request() as bench_request:
            yield bench_request
            ceshis = test_ShunXu.copy()
            test_try = {ceshi: 0 for ceshi in ceshis}

            test_JieGuo = {}
            unit_change = False

            while ceshis and not unit_change:
                test_name = ceshis.pop(0)
                try_count = test_try[test_name]

                if try_count >= 2:
                    test_JieGuo[test_name] = False
                    unit_change = True
                    continue

                with work_smalltable[test_name].request() as worktable_Xuqiu:
                    yield worktable_Xuqiu

                    # 获取当前班次剩余工作时间
                    DangQian_time = env.now
                    ZhuanHuan_start = (DangQian_time // work_time) * work_time
                    ZhuanHuan_end = ZhuanHuan_start + work_time
                    ShenYu_time = ZhuanHuan_end - DangQian_time

                    if ShenYu_time < CeShi_times[test_name]:
                        yield env.timeout(ShenYu_time)
                        ceshis.insert(0, test_name)
                        continue

                    # 检查设备故障概率
                    P_GuZhang = JiSuan_GuZhangprob_Sys(test_name, Ce-
Shi_times[test_name])
                    if np.random.rand() < P_GuZhang:
                        statis_var['units_changes'][test_name] += 1
                        yield from change_and_tiaoshi_device(env, test_name, sta-
tis_var)
                        ceshis.insert(0, test_name)
                        continue

                    # 没有故障，进行测试
                    yield env.timeout(CeShi_times[test_name])
                    statis_var['unit_use_time'][test_name] += Ce-
Shi_times[test_name]
                    statis_var['ZuBie_times'][test_name] += CeShi_times[test_name]
                    statis_var['sum_CeShi_count'] += 1

                    # 检查测手差错
                    CuoWu_er = np.random.rand() < P_error_er[test_name]
                    if CuoWu_er:
                        CuoWu_LeiXing = np.random.rand()
                        if CuoWu_LeiXing < 0.5:
```

```
                    # 误判
                    statis_var['sum_Wu_count'] += 1
                    test_JieGuo[test_name] = False
                    test_try[test_name] += 1
                    ceshis.insert(0, test_name)
                else:
                    # 漏判
                    statis_var['sum_Lou_count'] += 1
                    test_JieGuo[test_name] = True
            else:
                # 无测手差错，检查真实问题
                if test_name in ['A', 'B', 'C']:
                    if ZhenShi_CuoWu[test_name]:
                        test_JieGuo[test_name] = False
                        test_try[test_name] += 1
                        ceshis.insert(0, test_name)
                    else:
                        test_JieGuo[test_name] = True
                else:
                    # 检查是否有任何问题（A、B、C 漏检或 D 有问题）
                    RenYi_ques = ZhenShi_CuoWu['A'] or ZhenShi_CuoWu['B'] or
ZhenShi_CuoWu['C'] or ZhenShi_CuoWu['D']

                    if RenYi_ques:
                        test_JieGuo[test_name] = False
                        test_try[test_name] += 1
                        ceshis.insert(0, test_name)
                    else:
                        test_JieGuo[test_name] = True

        # 检查装置是否通过所有测试
        Guo_all = all(test_JieGuo.values()) if test_JieGuo else False

        if Guo_all:
            statis_var['Guo_count'] += 1

            if ZhenShi_CuoWu['A'] or ZhenShi_CuoWu['B'] or
ZhenShi_CuoWu['C'] or ZhenShi_CuoWu['D']:
                statis_var['num_Lou_unit'] += 1

        over_time = env.now
        statis_var['over_times'].append(over_time)

        statis_var['unit_over'] += 1
        if statis_var['unit_over'] < num_unit:
            yield env.timeout(YunShu_time)
            statis_var['YunShu_times'] += YunShu_time

    # 创建装置测试过程
    for i in range(2):
        env.process(test_process(env, i, test_bigtables, work_smalltable,
statis_var))

    def yanchi_initiate(env, device_id, delay):
        yield env.timeout(delay)
        env.process(test_process(env, device_id, test_bigtables,
work_smalltable, statis_var))

    for i in range(2, num_unit):
        delay = YunShu_time * (i - 1)
```

```
        env.process(yanchi_initiate(env, i, delay))

    env.run()
    statis_var['end_time'] = max(statis_var['over_times']) if sta-
tis_var['over_times'] else 0
    sum_days = np.ceil(statis_var['end_time'] / (2 * K))
    Guo_count = statis_var['Guo_count']

    # 漏判概率获取
    P_Lou = statis_var['num_Lou_unit'] / num_unit if num_unit > 0 else 0

    # 误判概率获取
    P_Wu = statis_var['sum_Wu_count'] / statis_var['sum_CeShi_count'] if
statis_var['sum_CeShi_count'] > 0 else 0

    # 获取有效工作时间比（YXB）
    sum_shift = np.ceil(statis_var['end_time'] / K)
    yxb = {}
    for ceshi in ['A', 'B', 'C', 'E']:
        sum_work_hour = statis_var['ZuBie_times'][ceshi] + sta-
tis_var['TiaoShi_times'][ceshi]
        yxb[ceshi] = (sum_work_hour / sum_shift) / K

    return {
        'sum_days': sum_days,
        'Guo_count': Guo_count,
        'P_Lou': P_Lou,
        'P_Wu': P_Wu,
        'yxb': yxb,
        'over_times': statis_var['over_times'],
        'units_changes': statis_var['units_changes']
    }


# 通过多次仿真来求取平均
def DuoCi_FangZhenPingJun(test_ShunXu, n_simulations=10, K=12):
    results = []
    for i in range(n_simulations):
        print(f"正在进行第 {i + 1} 次仿真...")
        result = ques3_FangZhen(test_ShunXu, K)
        results.append(result)

    # 求取平均值
    PJ_sum_day = np.mean([r['sum_days'] for r in results])
    PJ_Guo_count = np.mean([r['Guo_count'] for r in results])
    PJ_P_Lou = np.mean([r['P_Lou'] for r in results])
    PJ_P_Wu = np.mean([r['P_Wu'] for r in results])

    # 求取 YXB 的平均值
    PJ_yxb = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    for r in results:
        for ceshi in ['A', 'B', 'C', 'E']:
            PJ_yxb[ceshi] += r['yxb'][ceshi]
    for ceshi in ['A', 'B', 'C', 'E']:
        PJ_yxb[ceshi] /= n_simulations

    # 求取设备更换次数的平均值
    PJ_change_count = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    for r in results:
        for ceshi in ['A', 'B', 'C', 'E']:
```

```
            PJ_change_count[ceshi] += r['units_changes'][ceshi]
    for ceshi in ['A', 'B', 'C', 'E']:
        PJ_change_count[ceshi] /= n_simulations

    return {
        'PJ_sum_day': PJ_sum_day,
        'PJ_Guo_count': PJ_Guo_count,
        'PJ_P_Lou': PJ_P_Lou,
        'PJ_P_Wu': PJ_P_Wu,
        'PJ_yxb': PJ_yxb,
        'PJ_change_count': PJ_change_count
    }


def main():
    n_simulations = 100  # 每种顺序的仿真次数
    K = 11.5  # 问题 3 中 K=12
    all_results = {}

    # 对每种测试顺序进行仿真
    for ShunXu_name, ShunXu in TEST_SHUNXU.items():
        print(f"\n开始仿真测试顺序: {ShunXu_name}, K={K}")
        start_time = time.time()

        results = DuoCi_FangZhenPingJun(ShunXu, n_simulations, K)
        all_results[ShunXu_name] = results

        end_time = time.time()
        print(f"测试顺序 {ShunXu_name} 完成，耗时: {end_time - start_time:.2f} 秒
")

    print("\n" + "=" * 100)
    print(f"六种测试顺序的仿真结果比较 (K={K})")
    print("=" * 100)

    header = f"{'测试顺序':<8} {'平均天数(T)':<12} {'通过数(S)':<10} {'漏判概率
(PL)':<14} {'误判概率(PW)':<14} "
    header += " ".join([f"{'YXB_' + d:<8}" for d in ['A', 'B', 'C', 'E']])

    print(header)
    print("-" * 100)

    for ShunXu_name, results in all_results.items():
        line = f"{ShunXu_name:<8} {results['PJ_sum_day']:<12.2f} {re-
sults['PJ_Guo_count']:<10.2f} "
        line += f"{results['PJ_P_Lou']:<14.4f} {results['PJ_P_Wu']:<14.4f} "

        for ceshi in ['A', 'B', 'C', 'E']:
            line += f"{results['PJ_yxb'][ceshi]:<8.4f} "

        print(line)

    best_ShunXu = min(all_results.items(), key=lambda x: x[1]['PJ_sum_day'])
    print(f"\n最优测试顺序: {best_ShunXu[0]} (平均完成天数:
{best_ShunXu[1]['PJ_sum_day']:.2f} 天)")

    best_Lou_ShunXu = min(all_results.items(), key=lambda x:
x[1]['PJ_P_Lou'])
    print(f"漏判概率最低的顺序: {best_Lou_ShunXu[0]} (漏判概率:
{best_Lou_ShunXu[1]['PJ_P_Lou']:.4f})")
```

```
    best_yxb_order = max(all_results.items(), key=lambda x:
sum(x[1]['PJ_yxb'].values()))
    PJ_yxb = sum(best_yxb_order[1]['PJ_yxb'].values()) / 4
    print(f"设备利用率最高的顺序: {best_yxb_order[0]} (平均 YXB: {PJ_yxb:.4f})")

    print("\n 设备平均更换次数:")
    for ShunXu_name, results in all_results.items():
        print(f"{ShunXu_name}: A={results['PJ_change_count']['A']:.2f},
B={results['PJ_change_count']['B']:.2f}, "
              f"C={results['PJ_change_count']['C']:.2f}, E={re-
sults['PJ_change_count']['E']:.2f}")


# 运行主函数
if __name__ == "__main__":
    main()
```

q3order01.py

```
import numpy as np
import simpy
import random
from collections import defaultdict
import time
import itertools

# 设置六种测试顺序
TEST_SHUNXU = {
    'ABCE': ['A', 'B', 'C', 'E'],
    'ACBE': ['A', 'C', 'B', 'E'],
    'BACE': ['B', 'A', 'C', 'E'],
    'BCAE': ['B', 'C', 'A', 'E'],
    'CABE': ['C', 'A', 'B', 'E'],
    'CBAE': ['C', 'B', 'A', 'E']
}

UNIT = ['A', 'B', 'C', 'E']


def ques3_FangZhen(test_ShunXu, replace_strategy, K=12):
    # 参数设置
    num_unit = 100
    YunShu_time = 1
    CeShi_times = {'A': 2.5, 'B': 2.0, 'C': 2.5, 'E': 3.0}
    TiaoShi_times = {'A': 0.5, 'B': 1 / 3, 'C': 1 / 3, 'E': 2 / 3}

    # 设备故障概率（累积概率）
    P_GuZhang_Short = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.03}
    P_GuZhang_Long = {'A': 0.05, 'B': 0.07, 'C': 0.06, 'E': 0.05}

    P_Ques_Sys = {'A': 0.025, 'B': 0.03, 'C': 0.02, 'D': 0.001}
    P_error_er = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.02}
    work_time = K  # 每班次的工作时间（问题 3 中 K=12）

    # 初始化仿真环境
    env = simpy.Environment()
    test_bigtables = simpy.Resource(env, capacity=2)
    work_smalltable = {
        'A': simpy.Resource(env, capacity=1),
        'B': simpy.Resource(env, capacity=1),
```

```
        'C': simpy.Resource(env, capacity=1),
        'E': simpy.Resource(env, capacity=1)
    }

    # 定义全局统计变量
    statis_var = {
        'start_time': 0,
        'end_time': 0,
        'Guo_count': 0,
        'sum_Lou_count': 0,
        'sum_Wu_count': 0,
        'sum_CeShi_count': 0,
        'ZuBie_times': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'TiaoShi_times': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'YunShu_times': 0,
        'unit_use_time': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'unit_over': 0,
        'over_times': [],
        'units_changes': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
        'num_Lou_unit': 0,
        'preventive_replacements': {'A': 0, 'B': 0, 'C': 0, 'E': 0},  # 预防性
更换次数
        'shift_changes': 0  # 换班次数
    }

    # 计算设备故障概率的函数
    def JiSuan_GuZhangprob_Sys(test_name, test_time):
        use_time = statis_var['unit_use_time'][test_name]
        delt_t = test_time

        if use_time < 120:
            # 使用时间在 0-120 小时范围内
            H_ut = P_GuZhang_Short[test_name] * (use_time / 120)
            H_ut_delt_t = P_GuZhang_Short[test_name] * ((use_time + delt_t) /
120)
        elif use_time < 240:
            # 使用时间在 120-240 小时范围内
            H_ut = P_GuZhang_Short[test_name] + (P_GuZhang_Long[test_name] -
P_GuZhang_Short[test_name]) * (
                    (use_time - 120) / 120)
            H_ut_delt_t = P_GuZhang_Short[test_name] +
(P_GuZhang_Long[test_name] - P_GuZhang_Short[test_name]) * (
                    (use_time + delt_t - 120) / 120)
        else:
            return 1.0  # 使用时间达到 240 小时，必须更换设备

        if H_ut >= 1.0:
            return 1.0

        # 故障概率 = [F(use_time+delt_t) - F(use_time)] / [1 - F(use_time)]
        P_GuZhang = (H_ut_delt_t - H_ut) / (1 - H_ut)
        return min(P_GuZhang, 1.0)

    # 检查设备是否需要更换
    def JianCha_unit_change(test_name):
        usage = statis_var['unit_use_time'][test_name]
        strategy = replace_strategy[test_name]

        if strategy == 0:
            return usage >= 240
```

```
        else:
            return usage >= 120

    # 设备更换和调试过程
    def change_and_tiaoshi_device(env, test_name, statis_var, is_preven-
tive=False):
        if is_preventive:
            statis_var['preventive_replacements'][test_name] += 1
        else:
            statis_var['units_changes'][test_name] += 1

        statis_var['unit_use_time'][test_name] = 0

        TiaoShi_time = TiaoShi_times[test_name]
        yield env.timeout(TiaoShi_time)
        statis_var['TiaoShi_times'][test_name] += TiaoShi_time

    # 检查是否需要换班
    def JianCha_shift_change(DangQian_time):
        # 计算当前班次的结束时间
        ZhuanHuan_num = int(DangQian_time // work_time)
        ZhuanHuan_end = (ZhuanHuan_num + 1) * work_time
        return ZhuanHuan_end - DangQian_time

    # 装置的测试过程
    def test_process(env, device_id, test_bigtables, work_smalltable, sta-
tis_var):
        ZhenShi_CuoWu = {
            'A': np.random.rand() < P_Ques_Sys['A'],
            'B': np.random.rand() < P_Ques_Sys['B'],
            'C': np.random.rand() < P_Ques_Sys['C'],
            'D': np.random.rand() < P_Ques_Sys['D']
        }

        yield env.timeout(YunShu_time)
        statis_var['YunShu_times'] += YunShu_time

        with test_bigtables.request() as bench_request:
            yield bench_request
            ceshis = test_ShunXu.copy()
            test_try = {ceshi: 0 for ceshi in ceshis}

            test_JieGuo = {}
            unit_change = False

            while ceshis and not unit_change:
                test_name = ceshis.pop(0)
                try_count = test_try[test_name]

                if try_count >= 2:
                    # 已尝试 2 次仍未通过，装置退出
                    test_JieGuo[test_name] = False
                    unit_change = True
                    continue

                with work_smalltable[test_name].request() as worktable_Xuqiu:
                    yield worktable_Xuqiu

                    if JianCha_unit_change(test_name):
                        yield from change_and_tiaoshi_device(env, test_name, sta-
tis_var, is_preventive=True)
```

```
                ceshis.insert(0, test_name)
                continue

            # 获取当前班次剩余工作时间
            DangQian_time = env.now
            ShenYu_time = JianCha_shift_change(DangQian_time)

            if ShenYu_time <= 0:
                statis_var['shift_changes'] += 1
                ceshis.insert(0, test_name)
                continue

            if ShenYu_time < CeShi_times[test_name]:
                statis_var['shift_changes'] += 1
                yield env.timeout(ShenYu_time)
                ceshis.insert(0, test_name)
                continue

            # 检查设备故障概率
            P_GuZhang = JiSuan_GuZhangprob_Sys(test_name, Ce-
Shi_times[test_name])
            if np.random.rand() < P_GuZhang:
                yield from change_and_tiaoshi_device(env, test_name, sta-
tis_var, is_preventive=False)
                ceshis.insert(0, test_name)
                continue

            # 没有故障，进行测试
            yield env.timeout(CeShi_times[test_name])
            statis_var['unit_use_time'][test_name] += Ce-
Shi_times[test_name]
            statis_var['ZuBie_times'][test_name] += CeShi_times[test_name]
            statis_var['sum_CeShi_count'] += 1

            # 检查测手差错
            CuoWu_er = np.random.rand() < P_error_er[test_name]
            if CuoWu_er:
                CuoWu_LeiXing = np.random.rand()
                if CuoWu_LeiXing < 0.5:
                    # 误判
                    statis_var['sum_Wu_count'] += 1
                    test_JieGuo[test_name] = False
                    test_try[test_name] += 1
                    ceshis.insert(0, test_name)
                else:
                    # 漏判
                    statis_var['sum_Lou_count'] += 1
                    test_JieGuo[test_name] = True
            else:
                # 无测手差错，检查真实问题
                if test_name in ['A', 'B', 'C']:
                    if ZhenShi_CuoWu[test_name]:
                        test_JieGuo[test_name] = False
                        test_try[test_name] += 1
                        ceshis.insert(0, test_name)
                    else:
                        test_JieGuo[test_name] = True
                else:
                    # 检查是否有任何问题（A、B、C 漏检或 D 有问题）
                    RenYi_ques = ZhenShi_CuoWu['A'] or ZhenShi_CuoWu['B'] or
```

```
ZhenShi_CuoWu['C'] or ZhenShi_CuoWu['D']

                    if RenYi_ques:
                        test_JieGuo[test_name] = False
                        test_try[test_name] += 1
                        ceshis.insert(0, test_name)
                    else:
                        test_JieGuo[test_name] = True

            # 检查装置是否通过所有测试
            Guo_all = all(test_JieGuo.values()) if test_JieGuo else False

            if Guo_all:
                statis_var['Guo_count'] += 1

                if ZhenShi_CuoWu['A'] or ZhenShi_CuoWu['B'] or
ZhenShi_CuoWu['C'] or ZhenShi_CuoWu['D']:
                    statis_var['num_Lou_unit'] += 1

            over_time = env.now
            statis_var['over_times'].append(over_time)

            statis_var['unit_over'] += 1
            if statis_var['unit_over'] < num_unit:
                yield env.timeout(YunShu_time)
                statis_var['YunShu_times'] += YunShu_time

    # 创建装置测试过程
    for i in range(2):
        env.process(test_process(env, i, test_bigtables, work_smalltable,
statis_var))

    def yanchi_initiate(env, device_id, delay):
        yield env.timeout(delay)
        env.process(test_process(env, device_id, test_bigtables,
work_smalltable, statis_var))

    for i in range(2, num_unit):
        delay = YunShu_time * (i - 1)
        env.process(yanchi_initiate(env, i, delay))

    env.run()
    statis_var['end_time'] = max(statis_var['over_times']) if sta-
tis_var['over_times'] else 0

    # 计算任务完成天数（考虑班次和换班）
    # 问题 3 中，两个分队接续倒班，所以实际日历天数为原天数除以 2 并向上取整
    sum_days = np.ceil(statis_var['end_time'] / work_time / 2)

    Guo_count = statis_var['Guo_count']

    # 漏判概率获取
    P_Lou = statis_var['num_Lou_unit'] / num_unit if num_unit > 0 else 0

    # 误判概率获取
    P_Wu = statis_var['sum_Wu_count'] / statis_var['sum_CeShi_count'] if
statis_var['sum_CeShi_count'] > 0 else 0

    # 获取有效工作时间比（YXB）
    # 问题 3 中，YXB = 小组每班次平均测试时间 / K
```

```
    sum_shift = np.ceil(statis_var['end_time'] / work_time)
    yxb = {}
    for ceshi in ['A', 'B', 'C', 'E']:
        sum_work_hour = statis_var['ZuBie_times'][ceshi] + sta-
tis_var['TiaoShi_times'][ceshi]
        yxb[ceshi] = (sum_work_hour / sum_shift) / K

    return {
        'sum_days': sum_days,
        'Guo_count': Guo_count,
        'P_Lou': P_Lou,
        'P_Wu': P_Wu,
        'yxb': yxb,
        'units_changes': statis_var['units_changes'],
        'preventive_replacements': statis_var['preventive_replacements'],
        'total_replacements': {
            ceshi: statis_var['units_changes'][ceshi] + statis_var['preven-
tive_replacements'][ceshi]
            for ceshi in UNIT
        },
        'shift_changes': statis_var['shift_changes']
    }


# 通过多次仿真来求取平均
def DuoCi_FangZhenPingJun(test_ShunXu, replace_strategy, n_simulations=5,
K=12):
    results = []
    for i in range(n_simulations):
        result = ques3_FangZhen(test_ShunXu, replace_strategy, K)
        results.append(result)

    # 求取平均值
    PJ_sum_day = np.mean([r['sum_days'] for r in results])
    PJ_Guo_count = np.mean([r['Guo_count'] for r in results])
    PJ_P_Lou = np.mean([r['P_Lou'] for r in results])
    PJ_P_Wu = np.mean([r['P_Wu'] for r in results])

    # 求取 YXB 的平均值
    PJ_yxb = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    for r in results:
        for ceshi in ['A', 'B', 'C', 'E']:
            PJ_yxb[ceshi] += r['yxb'][ceshi]
    for ceshi in ['A', 'B', 'C', 'E']:
        PJ_yxb[ceshi] /= n_simulations

    # 求取设备更换次数的平均值
    PJ_change_count = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    avg_preventive_replacements = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
    avg_total_replacements = {'A': 0, 'B': 0, 'C': 0, 'E': 0}

    for r in results:
        for ceshi in UNIT:
            PJ_change_count[ceshi] += r['units_changes'][ceshi]
            avg_preventive_replacements[ceshi] += r['preventive_replace-
ments'][ceshi]
            avg_total_replacements[ceshi] += r['total_replacements'][ceshi]

    for ceshi in UNIT:
        PJ_change_count[ceshi] /= n_simulations
        avg_preventive_replacements[ceshi] /= n_simulations
```

```
        avg_total_replacements[ceshi] /= n_simulations

    # 计算平均换班次数
    avg_shift_changes = np.mean([r['shift_changes'] for r in results])

    return {
        'PJ_sum_day': PJ_sum_day,
        'PJ_Guo_count': PJ_Guo_count,
        'PJ_P_Lou': PJ_P_Lou,
        'PJ_P_Wu': PJ_P_Wu,
        'PJ_yxb': PJ_yxb,
        'PJ_change_count': PJ_change_count,
        'avg_preventive_replacements': avg_preventive_replacements,
        'avg_total_replacements': avg_total_replacements,
        'avg_shift_changes': avg_shift_changes,
        'replace_strategy': replace_strategy
    }


# 生成所有可能的设备更换策略组合
def ShengCheng_all_strategies():
    strategies = []
    for strategy in itertools.product([0, 1], repeat=4):
        strategy_dict = {
            'A': strategy[0],
            'B': strategy[1],
            'C': strategy[2],
            'E': strategy[3]
        }
        strategies.append(strategy_dict)
    return strategies


def main():
    n_simulations = 100  # 每种组合的仿真次数
    test_order_name = 'CABE'  # 选择一种测试顺序进行设备策略比较
    K = 12  # 问题3中K=12
    all_strategies = ShengCheng_all_strategies()
    all_results = {}

    print(f"开始仿真测试顺序: {test_order_name}, K={K}")
    print(f"共有 {len(all_strategies)} 种设备更换策略组合")

    # 对每种设备更换策略进行仿真
    for i, strategy in enumerate(all_strategies):
        print(f"正在仿真策略组合 {i + 1}/{len(all_strategies)}: {strategy}")
        start_time = time.time()

        results = DuoCi_FangZhenPingJun(TEST_SHUNXU[test_order_name], strat-
egy, n_simulations, K)
        strategy_key = f"A{strategy['A']}B{strategy['B']}C{strat-
egy['C']}E{strategy['E']}"
        all_results[strategy_key] = results

        end_time = time.time()
        print(f"策略组合 {strategy_key} 完成，耗时: {end_time - start_time:.2f}
秒")

    print("\n" + "=" * 120)
```

```
    print(f"测试顺序 {test_order_name}, K={K} 的 16 种设备更换策略仿真结果比较")
    print("=" * 120)

    header = f"{'策略':<8} {'平均天数':<8} {'通过数':<8} {'漏判概率':<10} {'误判
概率':<10} "
    header += " ".join([f"{'YXB_' + d:<8}" for d in UNIT])

    print(header)
    print("-" * 120)

    for strategy_key, results in all_results.items():
        line = f"{strategy_key:<8} {results['PJ_sum_day']:<8.2f} {re-
sults['PJ_Guo_count']:<8.2f} "
        line += f"{results['PJ_P_Lou']:<10.4f} {results['PJ_P_Wu']:<10.4f} "

        for device in UNIT:
            line += f"{results['PJ_yxb'][device]:<8.4f} "

        print(line)

    print("\n" + "=" * 80)
    print("设备更换次数统计")
    print("=" * 80)

    header2 = f"{'策略':<8} "
    header2 += " ".join([f"{'R_' + d:<6}" for d in UNIT])
    header2 += " ".join([f"{'P_' + d:<6}" for d in UNIT])
    header2 += " ".join([f"{'T_' + d:<6}" for d in UNIT])

    print(header2)
    print("-" * 80)

    for strategy_key, results in all_results.items():
        line = f"{strategy_key:<8} "

        for device in UNIT:
            line += f"{results['PJ_change_count'][device]:<6.1f} "

        for device in UNIT:
            line += f"{results['avg_preventive_replacements'][device]:<6.1f} "

        for device in UNIT:
            line += f"{results['avg_total_replacements'][device]:<6.1f} "

        print(line)

    best_strategy = min(all_results.items(), key=lambda x:
x[1]['PJ_sum_day'])
    print(f"\n最优设备更换策略: {best_strategy[0]} (平均完成天数: {best_strat-
egy[1]['PJ_sum_day']:.2f} 天)")

    best_leak_strategy = min(all_results.items(), key=lambda x:
x[1]['PJ_P_Lou'])
    print(f"漏判概率最低的策略: {best_leak_strategy[0]} (漏判概率:
{best_leak_strategy[1]['PJ_P_Lou']:.4f})")

    best_yxb_strategy = max(all_results.items(), key=lambda x:
sum(x[1]['PJ_yxb'].values()))
    PJ_yxb = sum(best_yxb_strategy[1]['PJ_yxb'].values()) / 4
    print(f"设备利用率最高的策略: {best_yxb_strategy[0]} (平均 YXB:
```

```
{PJ_yxb:.4f})")

# 运行主函数
if __name__ == "__main__":
    main()
```

附录 D: 问题 4 代码

q4topsis.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import font_manager

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 从 Excel 文件读取数据
filename = 'A4.xlsx'
sheet_name = 'Sheet1'

data = pd.read_excel(filename, sheet_name=sheet_name)
k_values = data['K 值']
ShuZhi_data = data[['平均天数', 'YXBave']].values

positive_data = ShuZhi_data.copy()
positive_data[:, 1] = 1 / positive_data[:, 1]

n, m = positive_data.shape
stand_data = positive_data / np.sqrt(np.sum(positive_data**2, axis=0))

# 获取正理想解和负理想解
Zheng_solution = np.min(stand_data, axis=0)
Fu_solution = np.max(stand_data, axis=0)

# 求得各方案与正负理想解的距离
dist_Zheng = np.sqrt(np.sum((stand_data - Zheng_solution)**2, axis=1))
dist_Fu = np.sqrt(np.sum((stand_data - Fu_solution)**2, axis=1))

rela_prox = dist_Fu / (dist_Zheng + dist_Fu)
ShunXu_index = np.argsort(-rela_prox)

JieGuo_dataf = pd.DataFrame({
    'K 值': k_values,
    '平均天数': ShuZhi_data[:, 0],
    'YXBave': ShuZhi_data[:, 1],
    '相对接近度': rela_prox
})

JieGuo_dataf['排名'] = 0
JieGuo_dataf.loc[ShunXu_index, '排名'] = range(1, n+1)
JieGuo_dataf = JieGuo_dataf.sort_values('排名')

print("TOPSIS 分析结果:")
print(JieGuo_dataf.to_string(index=False))
```