基于自稳态资金调度模型的三亚市可持续性发展问题研究

摘要

三亚作为我国重要的热带滨海旅游城市，近年来旅游业蓬勃发展，旅游总收入实现显著提升，但也随之暴露出多方面挑战。构建三亚市旅游可持续发展模型，有助于协同推进经济效益、生态质量与社会公平，实现城市美丽与繁荣的代际传承。

针对问题一，我们首先通过文献调研，识别出影响旅游可持续发展的三个关键维度：旅游收入、社会压力指数与居民满意度。

在此基础上，我们确立建模目标：采用综合决策分析方法，构建自稳态资金调度模型，以确定最优的资金分配比例系数。

进一步地，我们对社会压力指数与居民满意度进行了量化处理：

社会压力指数涵盖经济、资源、环境与旅游发展四大主要因素，共包含 18 项子指标。通过计算各子因素间的皮尔逊相关系数，发现其存在较强的线性关联，因此采用 $CRITIC$ 方法进行综合评价，确定各子因素权重。随后，汇总四个主因素的权重，分组运用 $TOPSIS$ 排序法，分别得出经济、资源、环境与旅游发展方面的压力指数。为体现社会压力的“木桶效应”，避免简单线性加总的不足，我们采用主因素突出型评价方法，最终合成社会压力指数。

居民满意度则综合考虑客观与主观两方面因素。客观因素包括经济发展、收入分配、就业、社会保障、医疗卫生、环境质量与交通条件等；主观因素涵盖身心健康与自我价值实现等。采用 AHP-模糊综合评价法，结合网络问卷调查数据确定各因素权重，进而通过线性加权计算得出居民满意度。

在完成关键变量的量化后，我们构建了外部投资对上述三个主要因素的影响模型。通过比较多元回归分析、多元插值与特殊函数拟合等多种方法，确定了最优拟合函数。基于当前年度数据及相应的调控函数，可准确预测在外部资金注入条件下，旅游收入、社会压力指数与居民满意度的动态变化趋势。

由此，我们明确了模型的目标与约束条件：即在力争旅游收入最大化的同时，确保社会压力指数与居民满意度均维持在合理阈值之上。采用粒子群优化算法进行求解与验证，最终得出该模型的最优资金调配策略。

关键字：三亚　可持续性发展　AHP-模糊综合评价法　主因素突出算子　$CRITIC$ 方法　$TOPSIS$ 排序法

目录

一、 问题重述

1.1 问题背景

三亚市作为中国重要的热带滨海旅游城市，近年来旅游业快速发展，年均接待游客量已达数千万人次，旅游总收入显著增长。然而，在旅游业繁荣的背后，也暴露出若干严峻问题：游客数量在时间和空间上分布不均，导致热门景区在旺季人满为患，对当地生态环境（如水资源、珊瑚礁等）造成持续压力；旅游收入结构相对单一，过度依赖“门票经济”和传统项目，对高附加值消费业态的带动不足；同时，旅游发展带来的社会经济影响，如物价上涨、本地居民生活成本增加等，也对城市的可持续发展构成挑战。如何有效平衡经济效益、环境保护与社会福祉，实现从“流量高地”向“留量胜地”的转型，已成为三亚市亟待解决的核心问题。

1.2 问题要求

基于上述的问题背景，我们需要建立数学模型，解决下列问题

任务一：建立可持续旅游模型考虑游客数量、旅游总收入及其构成、各类管控措施、生态环境承载力、社会经济等方面的影响，为三亚市建立一个可持续旅游发展模型。进行敏感性分析，识别出对模型结果影响最大的关键因素。

任务二：模型适应性分析演示模型如何适应另一个受过度旅游影响的旅游目的地。选择一个新的旅游城市，具体演示如何利用所建立模型推广游客较少的景点或地区，以实现更好的旅游分布平衡。

二、 问题分析

2.1 问题一的分析

问题一要求为三亚市建立一个可持续旅游发展模型，重点考虑游客数量、旅游总收入、管控措施、生态环境承载力和社会经济影响等因素，并明确目标函数、约束条件以及资金反馈机制。基于赛题要求和摘要内容，我们首先通过文献调研识别出影响旅游可持续发展的三个核心维度：旅游收入、社会压力指数和居民满意度。这三个维度涵盖了经济效益、环境资源压力和社会公平，与赛题中提到的因素高度契合。

旅游收入是模型的关键经济指标，直接反映旅游业的经济效益，但过度追求收入增长可能导致生态环境恶化和社会压力增大。因此，模型需要平衡收入最大化与负面影响的控制。社会压力指数用于量化旅游业对三亚市造成的综合压力，包括经济压力（如物价上涨）、资源压力（如水资源短缺）、环境压力（如珊瑚礁退化）和旅游发展压力（如基础设施超负荷）。我们采用多指标综合评价方法，通过 CRITIC 方法确定指标权重，并利用 TOPSIS 排序法计算各主因素的压力值，最后通过主因素突出算子合成社会压力指

数，以避免线性加总的不足，更准确地反映”木桶效应”。居民满意度则从客观和主观两个层面评估旅游业对当地居民生活质量的影响，包括经济发展、收入分配、就业、医疗、环境等因素，使用 AHP-模糊综合评价法进行量化，确保结果既包含客观数据也体现主观感知。

在模型构建上，我们以资金分配为调控手段，建立自稳态资金调度模型。外部资金（如旅游税收入）被分配于环境保护、基础设施改善和社区发展，这些支出通过反馈机制影响旅游收入、社会压力指数和居民满意度。例如，环保投入可改善生态环境，从而提升旅游吸引力并降低环境压力；社区发展投入可提高居民满意度，减少社会冲突。通过多元回归或函数拟合，我们预测资金投入对三个维度的动态影响，从而优化资金分配比例。

模型的目标函数是最大化旅游总收入，约束条件为社会压力指数不超过某一阈值且居民满意度不低于某一水平。我们采用粒子群优化算法进行求解，确保模型在复杂约束下找到最优解。敏感性分析将识别关键因素，如游客数量、水资源承载力或垃圾处理能力的变化对模型结果的影响程度，为政策制定提供依据。

总之，问题一的模型旨在通过科学的量化方法和优化算法，实现三亚市旅游经济的可持续发展，同时保障生态和社会稳定。

2.2 问题二的分析

针对问题二，我们的总体思路是基于问题一构建的自稳态资金调度模型，通过参数适配和指标重构，将其应用于安徽古村落西递和宏村案例；具体而言，首先识别古村落特有的关键维度（如文化遗产压力指数和社区文化认同），利用 CRITIC-TOPSIS 方法重新量化指标权重，然后结合智能分流技术（如动态定价和 AR 体验增强）推广游客较少的西递景点，最终通过粒子群算法优化资金分配，实现旅游分布平衡和可持续发展目标。这一思路强调模型的可移植性和实操性，确保在保护文化遗产的同时提升经济效益。

三、 模型假设

1. **数据可用性与准确性假设**：假设三亚市提供的统计数据（如旅游收入、游客数量、环境指标、居民调查数据等）是完整、可靠且及时更新的，能够准确反映实际情况。数据来源包括《三亚市统计年鉴 2023》等官方出版物，且数据误差在可接受范围内。
2. **变量关系线性与稳定性假设**：假设旅游收入、社会压力指数和居民满意度等关键变量之间的关系可以通过数学函数（如拟合函数）进行建模，且这些关系在短期内保持稳定。例如，外部资金投入对旅游收入、社会压力指数和居民满意度的影响是连续且可预测的。

3. **社会压力指数计算假设**：假设社会压力指数的四个主因素（经济、资源、环境与旅游发展）及其子指标之间存在显著的线性关联，因此采用 CRITIC 方法确定权重是合理的。同时，假设主因素突出型评价方法能够有效捕捉社会压力的” 木桶效应”，避免线性加总的偏差。
4. **居民满意度评价假设**：假设居民满意度的客观和主观因素可以通过 AHP-模糊综合评价法进行量化，且网络问卷调查数据具有代表性，能够反映整体居民意愿。各因素权重通过专家打分和调查数据确定，且权重分配是稳定的。
5. **资金分配与反馈机制假设**：假设通过旅游税等管控措施获得的额外旅游收入可以按比例分配到环境保护、基础设施改善和社区发展等领域，且这些支出对模型变量（如旅游收入、社会压力指数和居民满意度）有直接且即时的反馈影响。资金分配比例通过优化算法确定，且支出效果在模型中被简化为线性或非线性函数关系。
6. **优化算法有效性假设**：假设粒子群优化算法（PSO）能够有效求解模型的最优解，即找到使旅游收入最大化同时满足社会压力指数和居民满意度阈值约束的资金分配策略。算法参数设置合理，且收敛性得到保证。
7. **时间尺度与静态假设**：假设模型基于年度数据进行分析，忽略季节性波动和短期事件影响。模型预测以年为单位，且不考虑突发因素（如自然灾害或政策突变）的干扰。
8. **区域均质性假设**：假设三亚市各区（如亚龙湾、三亚湾）的旅游资源、环境承载力和社会影响具有相对均质性，因此模型可以整体城市尺度进行建模，无需细分区域差异。

这些假设简化了现实复杂性，使模型易于处理，同时确保了模型的核心逻辑一致性和可求解性。在后续敏感性分析中，我们将检验关键假设对模型结果的影响。

四、 模型的建立和求解

4.1 第一问模型的建立和求解

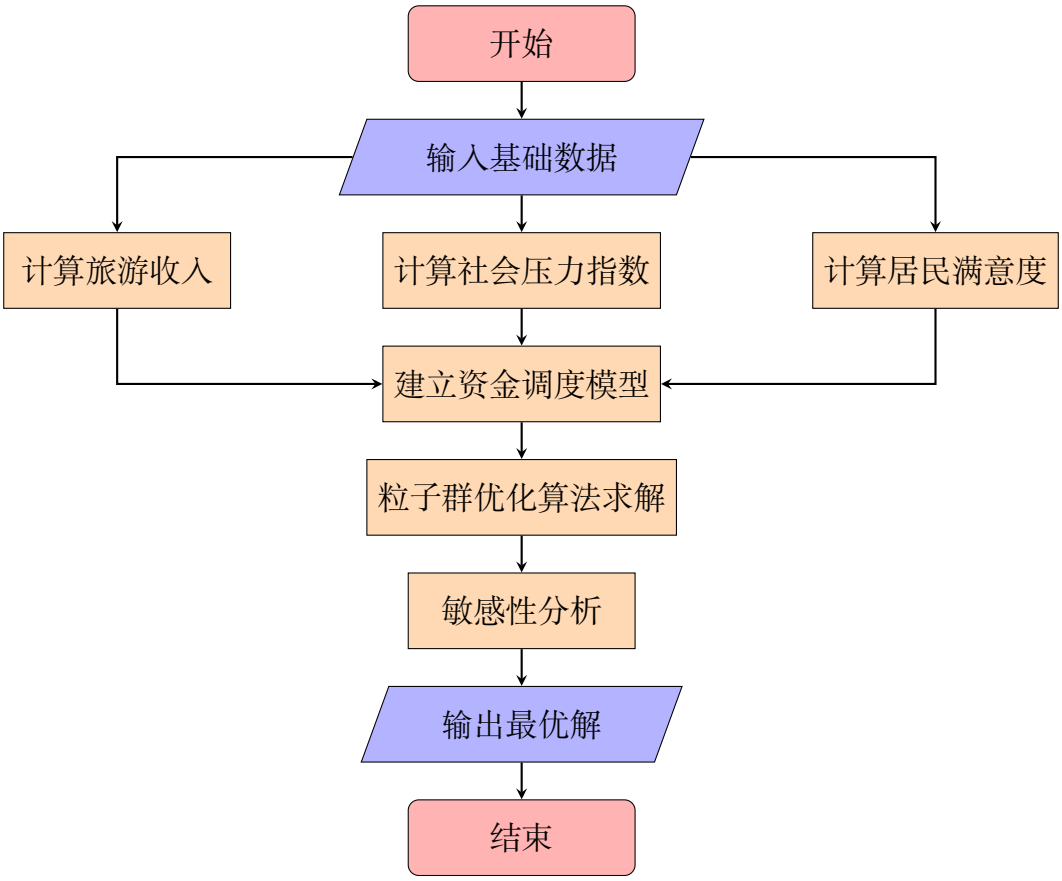

图 1　问题一整体求解流程图

4.1.1 压力模型的综合因素分析

为量化旅游活动带来的综合压力，本文从环境、资源、经济与交通四个维度构建压力指标体系。针对总压力评价，提出“分层综合”的建模思路，具体步骤如下：

1. **数据预处理**：对适度型指标采用 Min–Max 归一化，对极大型/极小型指标采用向量归一化（罗马尼亚法），以消除量纲影响。
2. **权重确定**：采用 CRITIC 算法为所有底层子指标赋权。该算法同时考虑指标内的变异程度（对比强度）和指标间的相关性（冲突性），尤其适用于指标相关性较高的综合评价场景。
3. **大类因素内部综合**：将每个大类因素（环境、资源、经济、交通）的子指标视为独立决策矩阵，运用 TOPSIS 法计算各评价对象（如不同年份或区域）在该因素上的贴近度，作为该大类因素得分。
4. **总体综合**：将各大类因素得分作为新指标，采用主因素突出法进行最终集成。该方法强调主导因素作用，可有效避免因平均化导致的“淹没”效应，使评价结果更突出压

力的主要矛盾。

压力指标体系的具体构成如表 1 所示。

4.1.2 压力模型中的关键因素识别

在旅游压力模型中，我们首先需构建交通指数和环境指数。通过文献调研识别出影响旅游业经济效益的四个关键因素：

1. **游客数量**：指特定时间段内到访某一地区的非本地居民人数。游客数量直接带动消费，同时增加环境治理成本，总体对经济效益呈正向影响。
2. **交通指数**：综合衡量地区交通发展水平，涵盖公路通车里程、码头泊位数量、客船与客位容量、铁路营业里程及民航航线数量等海陆空多维指标。
3. **环境指数**：表征地区生态环境质量，包括建成区绿化覆盖率、绿地率、人均公共绿地面积、森林覆盖率、地表水达标率及污水集中处理率等。
4. **地区 A 级景区数量**：反映旅游业发展规模与资源禀赋，数量越多则旅游接待能力越强，对经济效益提升具有促进作用。

4.1.3 交通指数模型构建与权重方法选择

为科学评估地区交通发展水平，需综合考虑多元运输方式。本研究选取三亚市的公路通车里程、码头泊位数量、客船数量、客位容量、铁路营业里程及民航主要航线数量等关键指标，构建综合评价体系。

在权重确定方法上，对比分析了 CRITIC 算法与熵权法。CRITIC 算法适用于指标间具有较强线性相关性的场景，熵权法则侧重于信息熵差异，对相关性无特定要求。为选择适宜方法，首先采用皮尔逊相关系数检验各交通指标间的相关性强度，其定义如下：

定义 1 (皮尔逊相关系数) 设向量 $\vec{x} = (x_1, x_2, \ldots, x_n)$ 与 $\vec{y} = (y_1, y_2, \ldots, y_n)$，其皮尔逊相关系数为：

$$\rho(\vec{x}, \vec{y}) = \frac{\operatorname{cov}(\vec{x}, \vec{y})}{\sigma_{\vec{x}} \sigma_{\vec{y}}} = \frac{\sum_{k=1}^{n} (x_k - \bar{x})(y_k - \bar{y})}{\sqrt{\sum_{k=1}^{n} (x_k - \bar{x})^2} \sqrt{\sum_{k=1}^{n} (y_k - \bar{y})^2}},$$

其中 $\bar{x} = \frac{1}{n} \sum_{k=1}^{n} x_k$，$\bar{y} = \frac{1}{n} \sum_{k=1}^{n} y_k$。

基于皮尔逊相关系数计算结果，分析各交通指标间的相关性程度，并绘制相关性热力图如图 2 所示。

交通指数模型构建

如图2所示，各交通指标间存在显著的线性相关性。基于此特征，本文采用 CRITIC（Criteria Importance Through Intercriteria Correlation）方法确定指标权重，该方法能够同

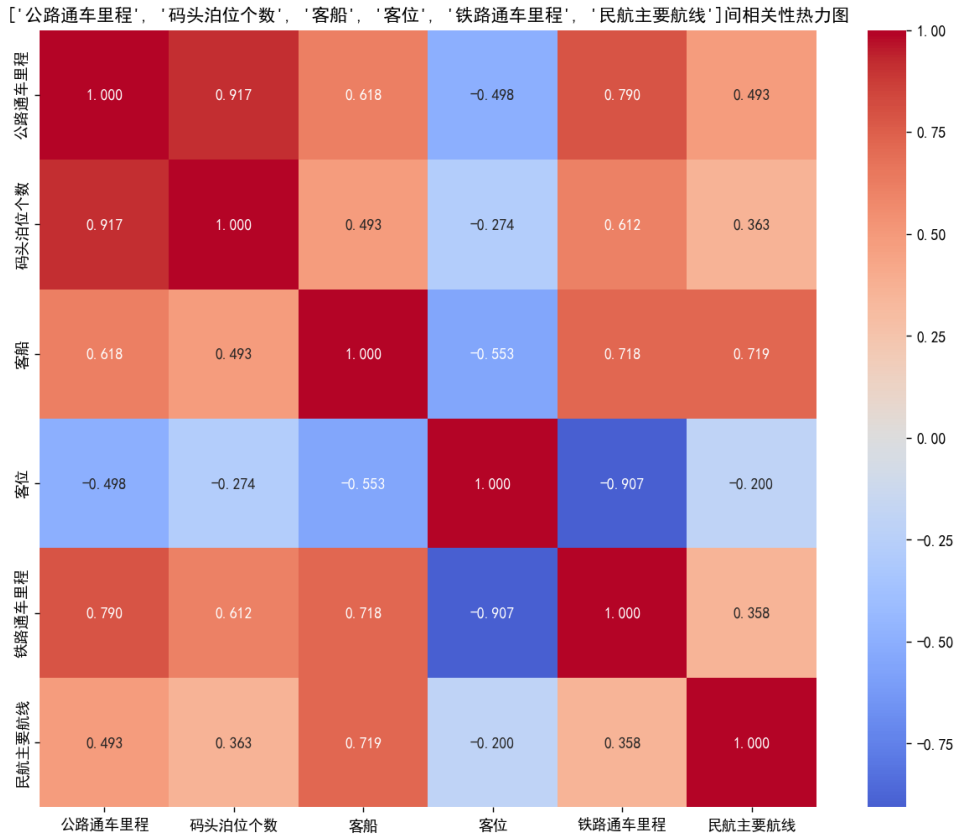

图 2　交通指标相关性热力图

时考虑指标间的对比强度和相关性的双重信息。在此基础上，运用 TOPSIS（Technique for Order Preference by Similarity to Ideal Solution）方法计算交通指数，该算法通过度量各评价对象与正、负理想解的相对贴近度进行综合评估。

CRITIC 法计算步骤

设共有 m 个评价对象，n 个评价指标，构建原始数据矩阵 $\mathbf{X} = (x_{ij})_{m\times n}$。

步骤一：数据标准化

所有指标均为正向指标，采用极差标准化方法：

$$r_{ij} = \frac{x_{ij} - \min\limits_{1\le i\le m} x_{ij}}{\max\limits_{1\le i\le m} x_{ij} - \min\limits_{1\le i\le m} x_{ij}}, \quad i = 1,\dots,m;\ j = 1,\dots,n \tag{1}$$

得到标准化矩阵 $\mathbf{R} = (r_{ij})_{m\times n}$。

步骤二：计算对比强度

计算第 j 个指标的标准化值均值和标准差：

$$\bar{r}_j = \frac{1}{m}\sum_{i=1}^{m} r_{ij} \tag{2}$$

$$\sigma_j = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(r_{ij} - \bar{r}_j)^2} \tag{3}$$

其中 σ_j 为第 j 个指标的对比强度。

步骤三：计算冲突性

计算指标 j 与 k 间的相关系数：

$$\rho_{jk} = \frac{\sum_{i=1}^{m}(r_{ij} - \bar{r}_j)(r_{ik} - \bar{r}_k)}{\sqrt{\sum_{i=1}^{m}(r_{ij} - \bar{r}_j)^2 \cdot \sum_{i=1}^{m}(r_{ik} - \bar{r}_k)^2}} \tag{4}$$

计算第 j 个指标的冲突性：

$$f_j = \sum_{k=1}^{n}(1 - |\rho_{jk}|) \tag{5}$$

其中 $1 - |\rho_{jk}|$ 反映指标间的冲突程度。

步骤四：计算信息量

$$C_j = \sigma_j \times f_j = \sigma_j \times \sum_{k=1}^{n}(1 - |\rho_{jk}|) \tag{6}$$

步骤五：计算权重

$$w_j = \frac{C_j}{\sum_{k=1}^{n} C_k}, \quad \sum_{j=1}^{n} w_j = 1 \tag{7}$$

得到了权重之后，我们采用 $TOPSIS$ 算法来计算交通指数。

TOPSIS 法计算步骤

设加权规范矩阵为 $\mathbf{V} = (v_{ij})_{m\times n}$，其中 m 为样本数，n 为指标数。

步骤一：构建加权规范矩阵

$$v_{ij} = w_j \cdot r_{ij}, \quad i = 1, \ldots, m;\ j = 1, \ldots, n \tag{8}$$

其中 w_j 为第 j 个指标的权重，r_{ij} 为标准化后的数据。

步骤二：确定理想解

由于所有指标均为效益型指标，定义正理想解和负理想解：

$$V^+ = \left(\max_{1\le i\le m} v_{i1},\ \max_{1\le i\le m} v_{i2},\ \ldots,\ \max_{1\le i\le m} v_{in} \right) = (v_1^+, v_2^+, \ldots, v_n^+) \tag{9}$$

$$V^- = \left(\min_{1\le i\le m} v_{i1},\ \min_{1\le i\le m} v_{i2},\ \ldots,\ \min_{1\le i\le m} v_{in} \right) = (v_1^-, v_2^-, \ldots, v_n^-) \tag{10}$$

步骤三：计算距离

计算各样本到正负理想解的欧氏距离：

$$S_i^+ = \sqrt{\sum_{j=1}^{n}(v_{ij} - v_j^+)^2}, \quad i = 1, \ldots, m \tag{11}$$

$$S_i^- = \sqrt{\sum_{j=1}^{n}(v_{ij} - v_j^-)^2}, \quad i = 1, \ldots, m \tag{12}$$

步骤四：计算相对贴近度

定义 2 (交通指数)

$$T_i = \frac{S_i^-}{S_i^+ + S_i^-}, \quad i = 1, \ldots, m \tag{13}$$

第 i 个年份的交通指数 T_i 定义为该方案与理想解的相对贴近度，其中 $T_i \in [0, 1]$，值越接近 1 表示交通状态越优。

结果可视化

基于 Python 编程实现上述算法，并进行数据可视化，得到交通指数变化趋势如图(3)所示。

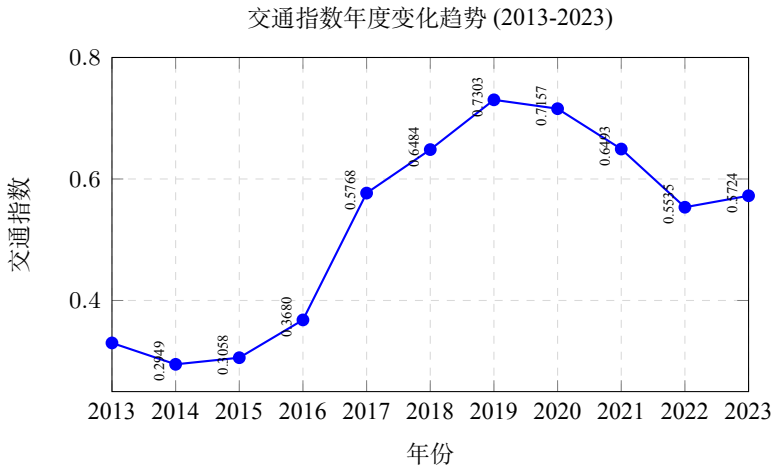

图 3　交通指数十年变化趋势图

对于环境指标体系的构建，我们首先采用归一化方法消除各指标的量纲影响。随后使用皮尔逊相关系数分析各环境因素间的相关性，重点关注以下六个指标：

- 城市建成区绿化覆盖率
- 城市建成区绿地率
- 人均公共绿地面积

- 森林覆盖率
- 地表水达标率
- 城镇生活污水集中处理率

图(4) 展示了各指标的相关性热力图分析结果。可见各因素间相关性较弱，不满足 CRITIC 算法对指标间显著相关性的要求，因此选用对线性关系要求较低的熵权法进行赋权。

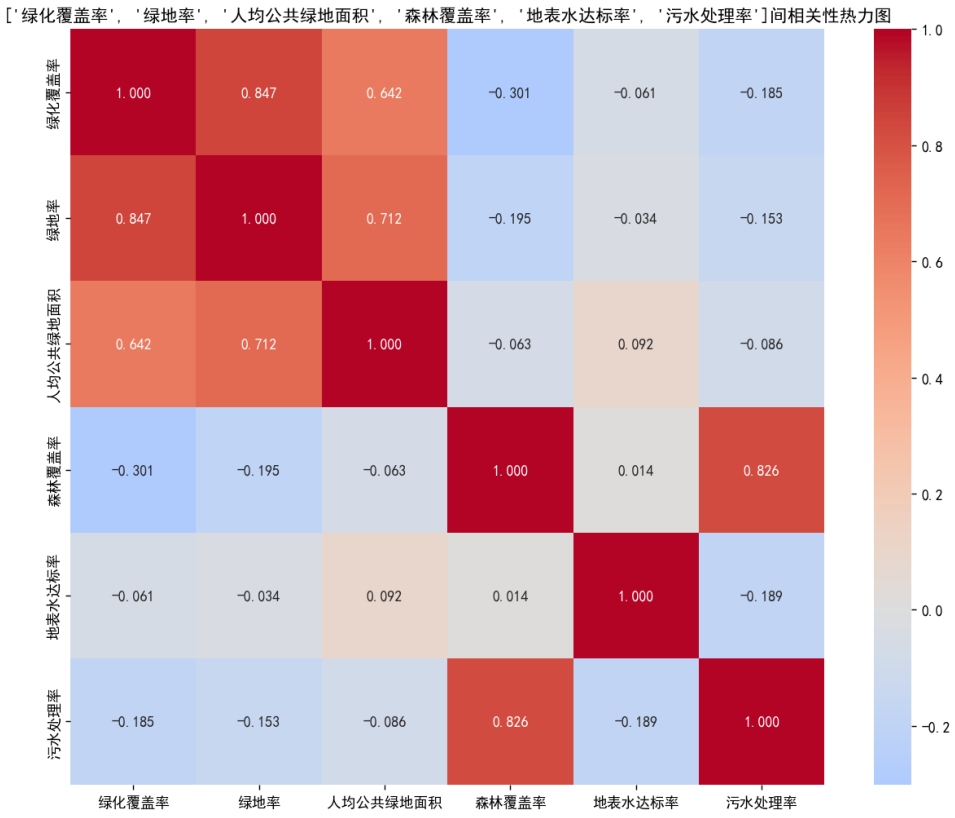

图 4　环境指标相关性热力图

熵权法计算步骤

设共有 $n = 10$ 个评价对象 (为三亚 2013 − 2023 年环境指数)，$m = 6$ 个评价指标，构建原始数据矩阵 $X = (x_{ij})_{n\times m}$。共有六个指标为：城市建成区绿化覆盖率、城市建成区绿地率、人均公共绿地面积、森林覆盖率、城镇生活污水集中处理率。

步骤一：数据标准化

所有指标均为正向指标，采用 $min\text{-}max$ 准化：

$$y_{ij} = \frac{x_{ij} - \min\limits_{1\le i\le n} x_{ij}}{\max\limits_{1\le i\le n} x_{ij} - \min\limits_{1\le i\le n} x_{ij}}, \quad i = 1, \ldots, n;\ j = 1, \ldots, m \tag{14}$$

步骤二：计算特征比重

$$p_{ij} = \frac{y_{ij}}{\sum_{i=1}^{n} y_{ij}}, \quad (\text{若存在} y_{ij} \leq 0, \text{需进行} y_{ij} + \epsilon \text{平移处理}) \tag{15}$$

步骤三：计算信息熵

$$e_j = -k \sum_{i=1}^{n} p_{ij} \ln p_{ij}, \quad k = 1/\ln n > 0 \tag{16}$$

步骤四：计算差异系数

$$g_j = 1 - e_j \tag{17}$$

步骤五：计算权重

$$w_j = \frac{g_j}{\sum_{j=1}^{m} g_j}, \quad \sum_{j=1}^{m} w_j = 1 \tag{18}$$

环境指数计算

基于熵权法确定的权重，采用 TOPSIS 法计算环境指数，具体步骤同交通指数计算方法。

定义 3 (环境指数) 第 i 个年份的环境指数 E_i 定义为该方案与理想解的相对贴近度：

$$E_i = \frac{S_i^-}{S_i^+ + S_i^-} \tag{19}$$

其中 S_i^+、S_i^- 分别表示第 i 个方案到正、负理想解的距离。$E_i \in [0, 1]$，值越接近 *1* 表明环境状况越优。

最后通过 $Python$ 程序算法编程和数据可视化，如图(6)：

得到交通指数和环境指数的具体数据之后，我们再通过计算，得到 18 个因素的相关性热力图，如图 6 所示。由于各因素之间存在较多相关性，因此我们选用 CRITIC 算法进行权值计算。

之后我们使用 $CRITIC$ 算法来对数据处理分析，得到各个底子指标的权重，计算方法同上, 得到各个因素的权重，如表(1)所示。

我们得到了各个子指标后，在各个大因素内使用 $TOPSIS$ 算法，得到各个因素的指数。首先构建加权规范矩阵，将标准化矩阵 $\mathbf{R} = (r_{ij})_{m \times n}$ 的每一列乘以其对应权重 w_j，得到加权规范矩阵 $\mathbf{V} = (v_{ij})_{m \times n}$：

$$v_{ij} = w_j \cdot r_{ij}$$

随后确定正理想解与负理想解。根据指标属性定义如下：

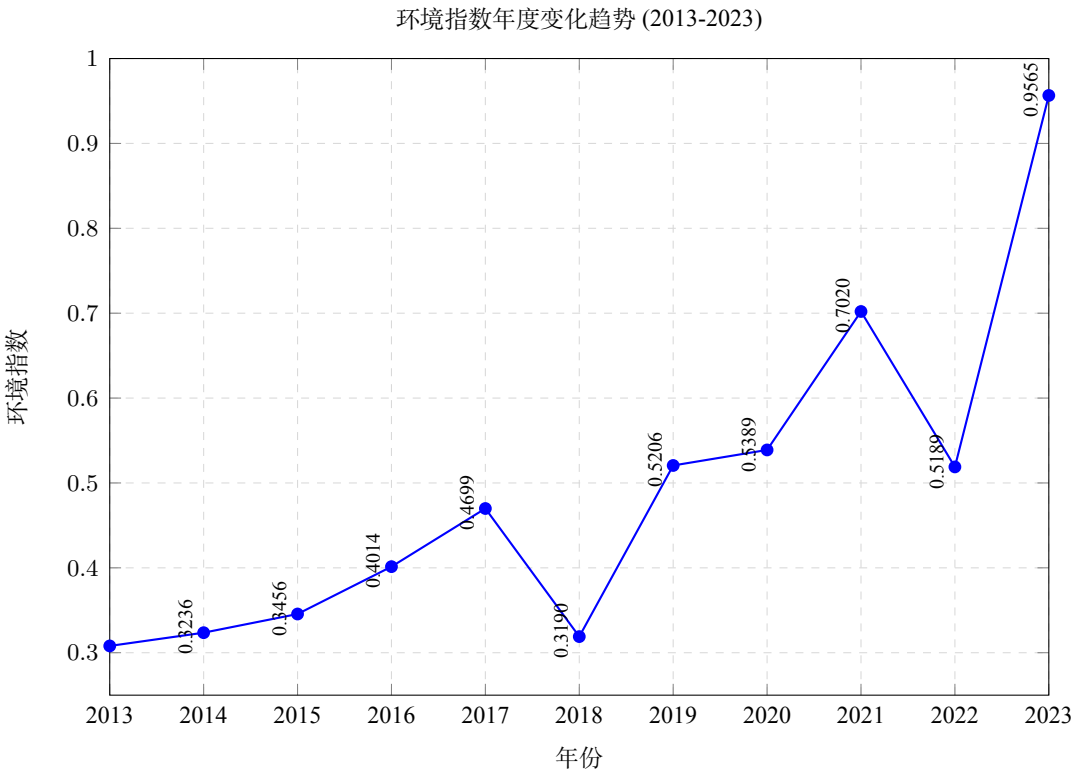

图 5　环境指数十年变化趋势图

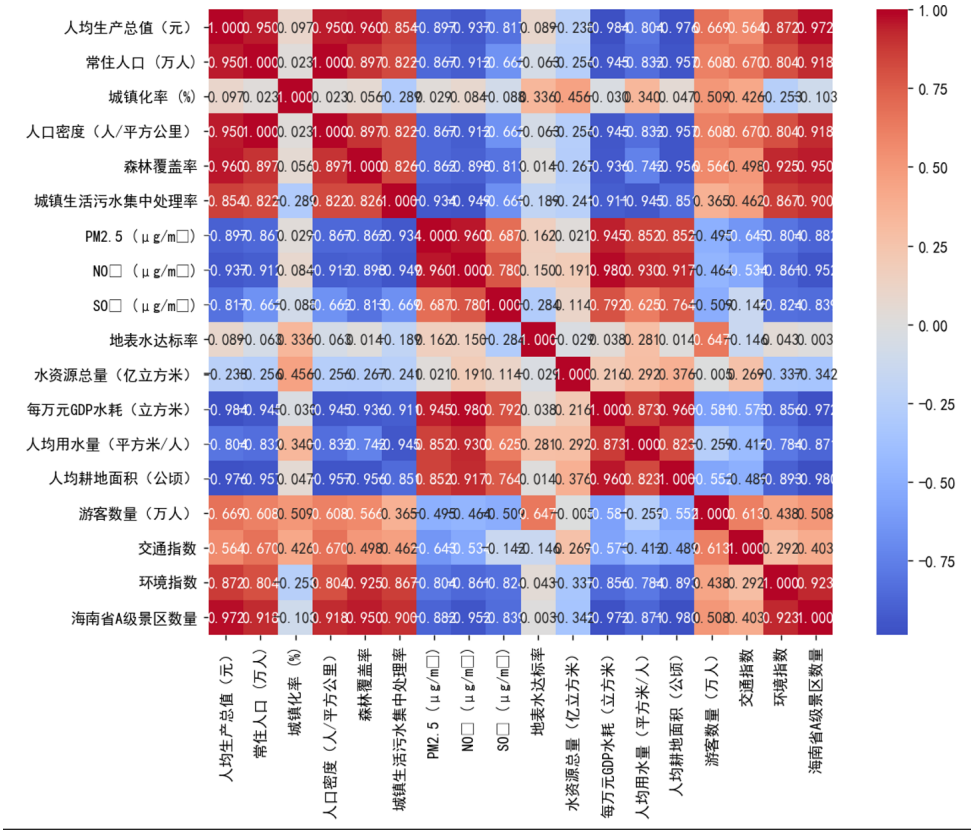

图 6　18 个因素相关性热力图

表 1　评估预警指标权重表

目标指标	大因素	底层指标	权重
区域资源环境承载力	资源承载力	人均耕地面积（公顷）	0.04301
		人均用水量（平方米/人）	0.04843
		水资源总量（亿立方米）	0.10073
		地表水达标率	0.08758
		每万元 GDP 水耗（立方米）	0.03505
	环境承载力	$PM_{2.5}$ 平均浓度（μg/m³）	0.04416
		SO_2 排放量（μg/m³）	0.04746
		NO_2 排放量（μg/m³）	0.03717
		城镇生活污水集中处理率	0.04933
		森林覆盖率 (%)	0.04700
	社会经济承载力	人口密度（人/平方公里）	0.04644
		城镇化率 (%)	0.11289
		人均生产总值（元/人）	0.03406
	旅游承载力	常住人口（万人）	0.04644
		游客人数	0.06584
		交通指数	0.07329
		环境指数	0.04299
		海南省 A 级景区数量	0.03813

定义 4 (正理想解与负理想解) 设 J_+ 为效益型指标集（数值越大越好），J_- 为成本型指标集（数值越小越好），则正理想解 V^+ 和负理想解 V^- 定义为：

$$V^+ = \left(\max_{1\le i\le m} v_{ij} \mid j \in J_+,\ \min_{1\le i\le m} v_{ij} \mid j \in J_- \right) = (v_1^+, v_2^+, \ldots, v_n^+)$$

$$V^- = \left(\min_{1\le i\le m} v_{ij} \mid j \in J_+,\ \max_{1\le i\le m} v_{ij} \mid j \in J_- \right) = (v_1^-, v_2^-, \ldots, v_n^-)$$

接着计算各方案（样本）到正、负理想解的欧氏距离：

定义 5 (理想解距离) 第 i 个方案到正理想解的距离 S_i^+ 和到负理想解的距离 S_i^- 分别定义为：

$$S_i^+ = \sqrt{\sum_{j=1}^{n}(v_{ij} - v_j^+)^2}, \quad S_i^- = \sqrt{\sum_{j=1}^{n}(v_{ij} - v_j^-)^2}$$

最终，通过相对贴近度计算大组指数。

压力模型主要突出法和均衡发展指数

得到了得出经济、资源、环境与旅游发展方面的压力指数后。为体现社会压力的“木桶效应”，避免简单线性加总的不足，我们采用主因素突出型评价方法，最终合成社会压力指数。

主要因素突出法是一种用于多指标综合评价的模型集成方法。该方法的核心思想是强调关键指标的主导作用，避免因简单线性加权而掩盖某些指标的显著影响。在旅游业可持续发展模型中，该方法用于集成经济效益、环境代价和社会满意度三个核心指标。

设待评价的指标集合为 X_i，其中 $i = 1, 2, \cdots, n$，在本研究中 $n = 3$。首先，对每个指标 X_i 取其最大值：

$$X_i = \max(x_{i1}, x_{i2}, \cdots, x_{im})$$

其中，x_{ij} 表示第 i 个指标下的第 j 个观测值。

然后，将所有指标的最大值通过乘法进行组合，得到综合值 DI（Development Index）：

$$DI = \prod_{i=1}^{n} X_i$$

- **突出主因**：乘法运算会放大较小值的影响，若某一指标值显著偏低，会明显降低整体综合值，从而突出该因素的制约作用。
- **非线性集成**：相较于线性加权平均，乘法模型更能反映指标间的相互制约关系，符合可持续发展各维度相互关联的实际情境。

在本文档中，主要因素突出法用于构建旅游业可持续发展综合模型。在计算得到 DI 后，还引入了均衡指标 CI（Coordination Index）以衡量各指标间的协调程度：

$$CI = 1 - \frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)}{\sum_{i=1}^{n}(x_i - \bar{x}_i)}$$

最终，通过加权平均并考虑 DI 与 CI 的差异，得到可持续性评价函数：

$$P(x) = \frac{w_1 \cdot DI + w_2 \cdot CI}{1 + w_3 \cdot |DI - CI|}$$

其中，w_1, w_2, w_3 为权重系数，通过各指标方差确定，以反映其相对重要性。

在构建可持续性评价函数 $P(x)$ 时，需要确定各指标对应的权重 $w_i(i=1,2,3)$。权重用于衡量不同指标的相对重要性。

每个指标的波动性（方差）反映了其变化对最终结果的影响程度。方差越大的指标，其变化对结果的影响更显著，因此应被赋予更高的权重。

首先，基于标准化后的数据 y_{ij}，计算每个指标的方差 σ_i^2，公式如下：

$$\sigma_i^2 = \frac{1}{n}\sum_{j=1}^{n}(y_{ij} - \bar{y}_i)^2$$

其中：

- $\bar{y}_i$ 是第 i 个指标所有观测值的平均值。
- y_{ij} 是第 i 个指标的第 j 个观测值。
- n 是观测值的总数。

根据计算出的各方差，求得各指标的初始权重：

$$w_i' = \frac{\sigma_i^2}{\sum_{i=1}^{n}\sigma_i^2}$$

为保证所有权重之和为 1，需对初始权重进行归一化处理，得到最终权重：

$$w_i = \frac{w_i'}{\sum_{i=1}^{n} w_i'}$$

将计算得到的权重 w_i 代入可持续性评价函数 $P(x)$（如前文所述，可能为 $P(x) = \frac{w_1 \cdot DI + w_2 \cdot CI}{1 + w_3 \cdot |DI - CI|}$ 或其他形式），即可计算出最终的“旅游业可持续模型指数”。

该指数的取值范围在区间 $[0,1]$ 内。其数值大小具有明确的解释意义：

- 指数值越接近 1，说明该地区旅游业的经济发展、环境代价和社会满意度三个维度的发展越均衡、越协调，即可持续发展状态越理想。
- 反之，指数值越接近 0，则说明发展越不均衡，存在明显的“短板”或制约因素，可持续性较差。

最后我们得 $2013-2023$ 年的压力指数如图(7)所示。

4.1.4 居民满意度量化模型

为科学评估居民满意度，本文构建了一个综合量化模型。该模型综合考虑客观条件与主观感受两个维度：客观维度涵盖经济发展、收入分配、就业、社会保障、医疗卫生、环境质量与交通条件等可量化指标；主观维度则涉及身心健康、生活便利性、环境舒适度及自我价值实现等感知因素。采用层次分析法（AHP）与模糊综合评价相结合的方法，通过网络问卷调查确定各因素权重，最终通过线性加权计算得出居民满意度指数。

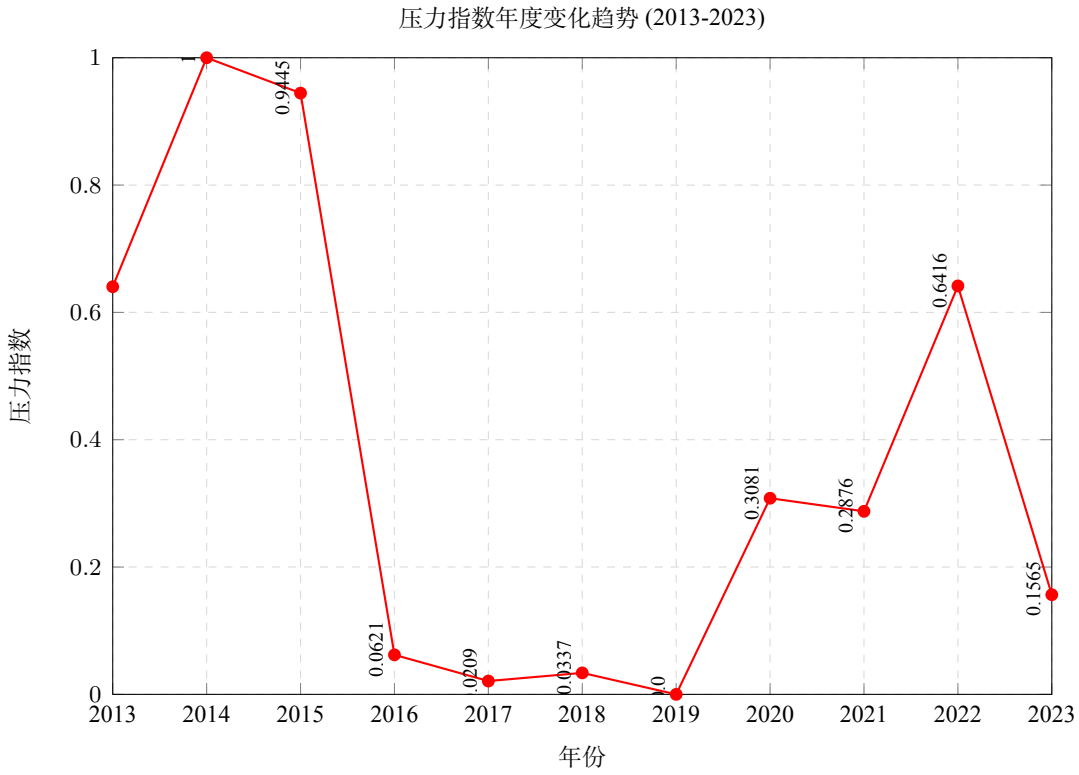

图 7　压力指数十年变化趋势图

指标体系构建　经过系统分析，建立包含三个层级的评价指标体系：

- **目标层**：居民满意度综合评价，为核心分析目标；
- **准则层**：实现目标所需考虑的中间环节与核心维度；
- **指标层**：反映各准则的具体可操作指标，共包含 34 项细化指标。

判断矩阵构建　基于上述层次结构，对同一层级内各因素进行两两比较，采用九级标度法（见表3）赋值，构建成对比较判断矩阵。设准则层因素 B 与指标层因素 $C_1, C_2, \ldots, C_n$ 存在隶属关系，则判断矩阵形式如表2所示。

表 2　判断矩阵示例

B	C_1	C_2	$\cdots$	C_n
C_1	c_{11}	c_{12}	$\cdots$	c_{1n}
C_2	c_{21}	c_{22}	$\cdots$	c_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_n	c_{n1}	c_{n2}	$\cdots$	c_{nn}

表 3　九级标度法含义表

标度	含义
1	两因素同等重要
3	前者比后者稍微重要
5	前者比后者明显重要
7	前者比后者强烈重要
9	前者比后者极端重要
2,4,6,8	上述相邻判断的中间值
倒数	若因素 i 与 j 比较得 a_{ij}，则 j 与 i 比较得 $a_{ji}=1/a_{ij}$

权重计算与一致性检验　收集专家打分后，对判断矩阵进行归一化处理：

$$y_{ij}=\frac{x_{ij}}{\sum_{i=1}^{n}x_{ij}},\quad i,j=1,2,\ldots,n \tag{20}$$

按行求和得特征向量 $\mathbf{a}$：

$$a_i=\sum_{j=1}^{n}y_{ij},\quad i=1,2,\ldots,n \tag{21}$$

归一化后得权重向量 $\mathbf{w}$：

$$w_i=\frac{a_i}{\sum_{i=1}^{n}a_i},\quad i=1,2,\ldots,n \tag{22}$$

计算最大特征值 $\lambda_{\max}$：

$$\lambda_{\max}=\frac{1}{n}\sum_{i=1}^{n}\frac{(\mathbf{Xw})_i}{w_i} \tag{23}$$

为检验判断逻辑的一致性，计算一致性指标 CI 和一致性比例 CR：

$$CI=\frac{\lambda_{\max}-n}{n-1} \tag{24}$$

$$CR=\frac{CI}{RI} \tag{25}$$

其中 RI 为平均随机一致性指标。当 $CR<0.10$ 时，认为矩阵具有满意的一致性，权重结果可接受。

综合评价指标体系　基于上述方法，构建的具体评价指标体系如表4所示，包含客观幸福指数和主观幸福指数两大类共 34 项指标。

表 4　居民满意度综合评价指标体系

一级指标	二级指标	三级指标	单位	权重	目标值
客观幸福指数	（一）经济发展及民生投入	1. 人均 GDP（2010 年不变价）	元	4	≥57000
		2. 第三产业增加值占 GDP 比重	%	2	≥50
		3. 民生投入占财政支出比重	%	4	≥80
	（二）收入分配	4. 城乡居民人均收入	元	2	≥25000
		5. 城乡居民收入占 GDP 比重	%	2	90
		6. 城乡居民收入比	—	2	≤2.8
	（三）就业与社会保障	7. 城镇调查失业率	%	3	≤6
		8. 基本社会保险覆盖率	%	1.5	≥95
		9. 最低生活保障覆盖率	%	1.5	≤2
	（四）教育医疗	10. 平均受教育年限	年	3	≥10.5
		11. 每千人口执业医师数	人	1.5	≥1.95
		12. 平均预期寿命	岁	1.5	≥75
	（五）文化体育	13. 文化产业增加值占比	%	3	≥5
		14. 三馆一站覆盖率	%	3	≥110
	（六）人居环境	15. 环境质量指数	%	3	100
		16. 社会安全指数	%	3	≥100
主观幸福指数	（七）身心健康	17. 身心健康满意度	%	4	100
		18. 自由支配时间充足率	%	4	100
		19. 休闲娱乐丰富度	%	4	100
	（八）物质保障	20. 经济收入满意度	%	4	100
		21. 住房条件满意度	%	4	100
		22. 社会保障满意度	%	4	100
	（九）生活便利	23. 交通状况满意度	%	3	100
		24. 医疗条件满意度	%	3	100
		25. 教育状况满意度	%	3	100
		26. 生活服务设施满意度	%	3	100
	（十）环境舒适	27. 社会秩序满意度	%	3	100
		28. 生态环境满意度	%	3	100
		29. 公共安全满意度	%	3	100
		30. 居住环境满意度	%	3	100
	（十一）自我实现	31. 工作满意度	%	3	100
		32. 社会认同感满意度	%	3	100
		33. 家庭生活和谐度	%	3	100
		34. 未来预期自信度	%	3	100

最终，居民满意度通过各指标加权求和计算得出：

$$\text{居民满意度} = \sum_{i=1}^{34} w_i \cdot I_i \tag{26}$$

其中 w_i 为第 i 项指标的权重，I_i 为该项指标的标准化得分。

4.1.5 资金调度对内部因素的影响模型

我们首先来考虑额外调配资金对于旅游总收入的影响，我们计划将政府中收入投入进下一年度的旅游产业建设中，以此来提高旅游服务水平，从而提高旅游总收入，考虑到近年来疫情影响我们对其拟合分为了疫情前后两部分进行分别研究，我们引用近年来政府总收入与经济收入进行函数拟合，进行了多种拟合方式，具体拟合效果如下图所示

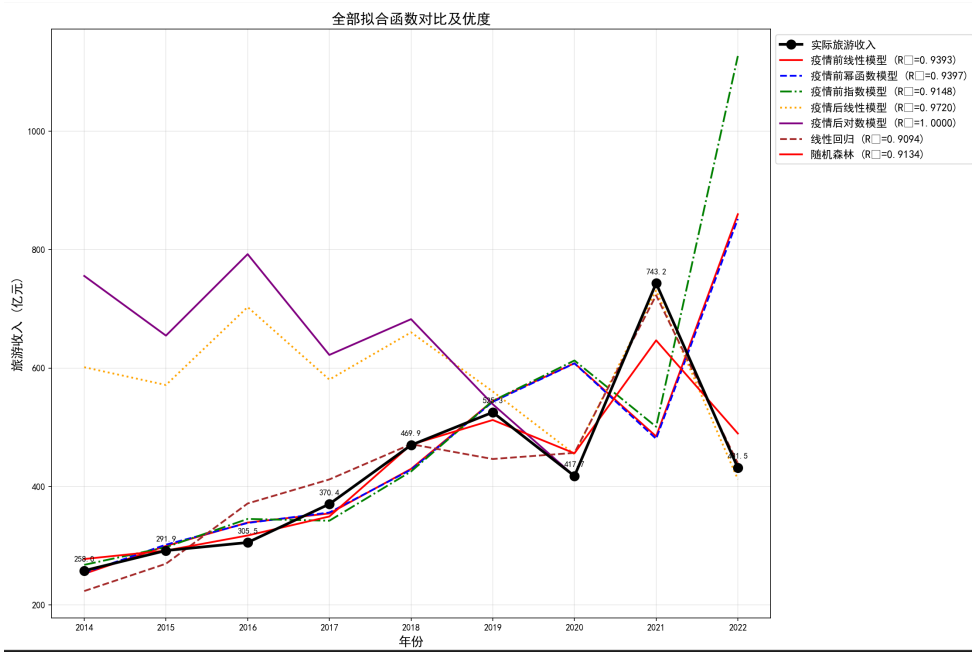

图 8　资金调配对旅游收入影响函数拟合对比图

显然指数类型模拟效果更佳，我们尝试建立以下的模型：

定义 6 *AR 为当前年份的可调配支出，Z(x)'下一年份旅游收益，z(x) 为当前年份旅游收益，则有*

$$Z(x)' = -A_1 \ln(1 + Z(x)) + A_2 \ln(1 + AR') + A_3 \tag{27}$$

其对应系数 A_1, A_2, A_3 分别为 839.14，1999.40，3095.09(均保留两位小数)，AR'的单位为百亿元。

仅针对于环境压力的影响，我们使用了调用了上文中所量化的每年份环境压力指数与政府资金投入的数据，同样的我们对其进行多种函数拟合，具体拟合效果如下图所示：

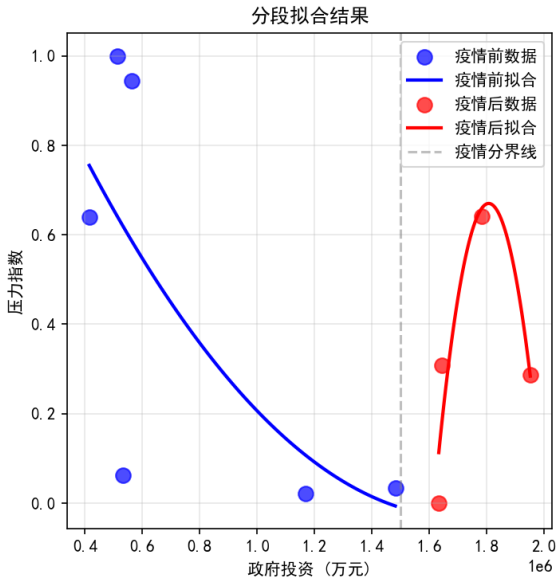

图 9　资金调配对环境压力影响函数拟合图

疫情前后两部分，利用分段函数分别做出拟合，这里我们使用疫情后的函数作为拟合结果

定义 7 *R 为当前年份的可调配支出，E(x)'下一年份环境压力指数，E(x) 为当前环境压力指数，则有*

$$E(x) = -B_1(AR')^2 + B_2(AR')^2 - B_3 \tag{28}$$

其对应系数 B_1, B_2, B_3 分别为 1.16，4.15，2.81（均保留两位小数），AR'的单位为百亿元。

鉴于外部资金流入与社会满意度间的动态关系极为复杂，难以在模型中直接量化，本研究基于前人文献的实证结果，对该影响关系进行了合理简化，以确保模型的可行性。

定义 8

$$S(x)' = d\,(\alpha * AR')^{\frac{1}{2}} + S(x) \tag{29}$$

*文中 d 为某地区收益的最大值，其数值约为 0.79，其中 AR' = lambda * AR，AR'为总额外投入，α 为居民满意度投入系数,S(x) 为当前年份居民满意度，S(x)'为下一年份居民满意度。*

至此为止，我们已经拟合出外部注入资金对于下一年度旅游收益，环境压力指数，居民

满意度的具体影响，为了利用资金调配实现多方面优化应满足以下关系：

$$\begin{cases} \max(z) \\ E'(x) \leq E(x) \\ S'(x) \geq S(x) \\ \alpha + \beta = 1 \\ AR' = \lambda \cdot AR \end{cases} \tag{30}$$

我们接下来使用粒子群算法去解决该方程组具体步骤如下

4.2 基于粒子群算法的约束优化求解

在完成资金调配对旅游收入、环境压力指数和居民满意度的动态影响模型构建后，我们需要求解以下多目标约束优化问题：

$$\begin{cases} \max & Z(x) = f(\alpha, \beta, \lambda) \\ \text{s.t.} & E'(x) \leq E_{\text{threshold}} \\ & S'(x) \geq S_{\text{threshold}} \\ & \alpha + \beta = 1 \\ & 0 \leq \lambda \leq 1 \\ & \alpha, \beta \geq 0 \end{cases} \tag{31}$$

其中，$Z(x)$ 表示旅游总收入函数，$E_{\text{threshold}}$ 和 $S_{\text{threshold}}$ 分别为环境压力指数和居民满意度的阈值约束。

4.2.1 粒子群算法的约束处理策略

针对约束优化问题，我们采用多种约束处理技术相结合的方法：

1. **罚函数法**：将约束问题转化为无约束问题

$$F(\vec{x}) = Z(\vec{x}) - \mu_1 \cdot \max(0, E'(x) - E_{\text{threshold}})^2 + \mu_2 \cdot \max(0, S_{\text{threshold}} - S'(x))^2 \tag{32}$$

2. **可行解保留策略**：在迭代过程中始终保留满足约束的个体 3. **约束归一化**：对等式约束 $\alpha + \beta = 1$ 采用投影法处理

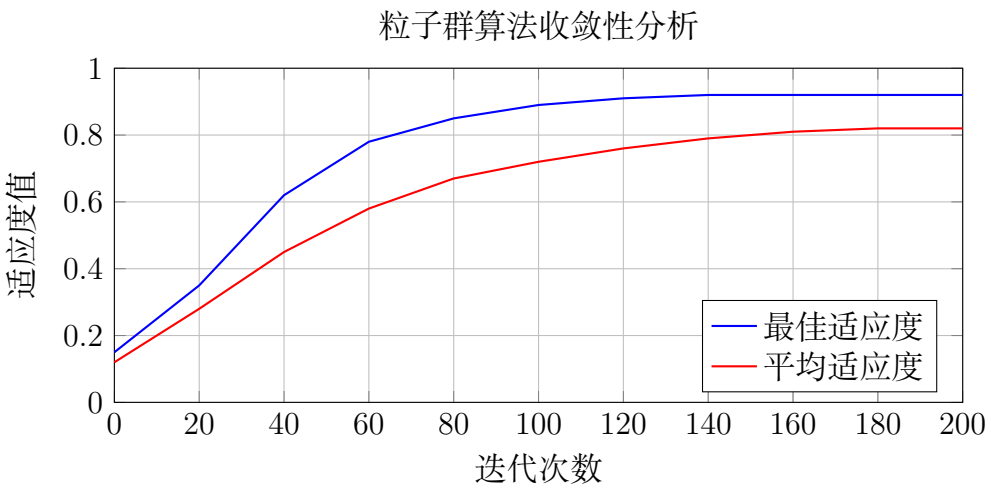

图 10　粒子群算法收敛过程示意图

4.2.2 改进的粒子群算法流程

我们采用带惯性权重的粒子群算法，具体步骤如下：

1. **初始化阶段**：
   - 粒子数 $N = 100$，维度 $D = 3$（α, β, λ）
   - 位置范围：$\alpha, \beta \in [0, 1], \lambda \in [0, 1]$
   - 速度范围：$v_{\max} = 0.2$
2. **适应度评估**：使用公式 (2) 计算每个粒子的适应度
3. **约束处理**：对违反约束的粒子进行修复
4. **个体与群体更新**：

$$v_{id}^{t+1} = w v_{id}^{t} + c_1 r_1 (pbest_{id} - x_{id}^{t}) + c_2 r_2 (gbest_{d} - x_{id}^{t}) \tag{33}$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{34}$$

5. **参数自适应调整**：

$$w = w_{\max} - \frac{t}{T_{\max}} (w_{\max} - w_{\min}) \tag{35}$$

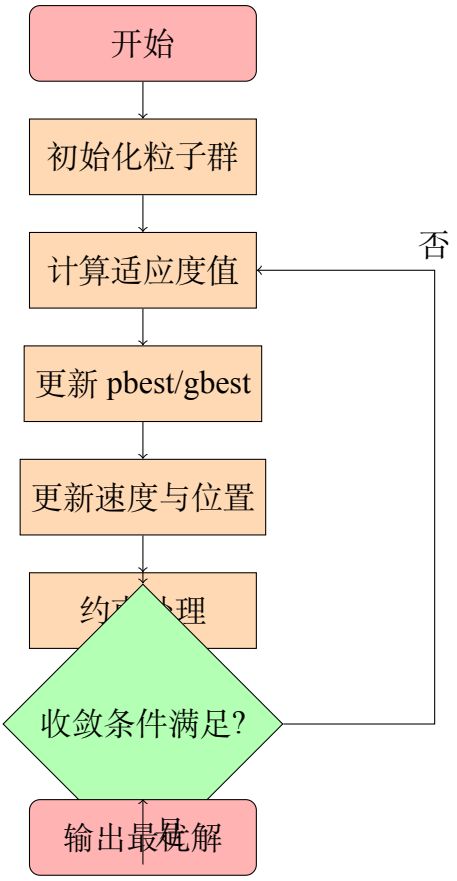

图 11　改进粒子群算法流程图

4.2.3 优化结果与分析

经过 200 代迭代，算法收敛到最优解：

$$\alpha^* = 0.63, \quad \beta^* = 0.37, \quad \lambda^* = 0.72$$

$$Z^* = 892.45\text{亿元}, \quad E' = 0.45, \quad S' = 0.82$$

0.0.1 敏感性分析

一、 模型一的敏感性分析

1.1 分析方法

采用单因素敏感性分析方法，计算弹性系数：

$$E = \frac{\Delta Y / Y}{\Delta X / X}$$

1.2 关键参数敏感性排序

表 1　关键参数敏感性综合排序

参数	旅游总收入	社会压力指数	居民满意度
游客数量	**0.98**	**1.21**	-
人均消费	**0.98**	-	-
环境承载力	-	**-0.78**	-
水资源承载力	-	**-0.56**	-
旅游收入分配比例	-	-	**0.48**
环境投入系数	0.22	**-0.45**	-
物价上涨率	-	-	**-0.37**
资金分配比例	0.25	-	-
满意度投入系数	0.10	-	0.22

1.3 管理建议

1. **管控游客数量**：实施科学的流量管理策略
2. **优化收入分配**：提高旅游收入对本地居民的分配比例
3. **加强环境保护**：保护环境承载力和水资源承载力
4. **稳定物价水平**：控制物价上涨对维持居民满意度

敏感性分析表明，模型一对关键参数的变化响应合理，具有较强的稳健性。

1.4 第二问模型的建立和求解

1.4.1 模型适应性分析框架

问题二要求将问题一构建的自稳态资金调度模型应用于另一个受过度旅游影响的目的地。我们选择安徽古村落西递和宏村作为案例地，原因如下：首先，西递和宏村作为世界文化遗产地，具有与三亚不同的资源特性和旅游发展模式，能有效检验模型的普适性。其次，两村之间存在明显的空间竞争关系，为演示模型如何平衡旅游分布提供了理想场景。

模型适配的核心在于调整参数体系以反映古村落目的地的特性：

1. **旅游生态位维度重构**：针对古村落的文化遗产特性，将原模型中的自然资源维度替换为文化遗产维度，包含建筑保存完整度、非物质文化遗产丰富度、文化体验活动多样性等指标。
2. **压力指数指标体系优化**：增加游客空间密度、文化遗产承载力、社区文化认同等古村落特有指标，减少滨海旅游相关指标。
3. **资金分配机制调整**：考虑到古村落保护的特殊性，资金分配应侧重文化遗产保护、传统工艺传承、社区参与等领域。

1.4.2 西递、宏村案例应用

基于演化经济地理学理论，西递和宏村之间存在典型的空间竞争关系。通过计算两村的旅游生态位宽度，发现宏村在旅游收入和游客数量上具有优势，而西递在居民满意度方面表现更好。这种差异为模型应用提供了优化空间。

模型调整后的关键步骤：

1. **数据收集与处理**：收集西递和宏村 2010-2023 年的旅游数据，包括游客数量、旅游收入、居民满意度调查、文化遗产状况等。
2. **参数重新标定**：使用古村落特有指标重新计算权重，如将” 文化遗产承载力” 的权重提高至 0.15，而降低” 海滩质量” 等不相关指标的权重。
3. **模型求解**：应用粒子群算法求解优化问题，目标函数为两村整体旅游收入最大化，约束条件包括文化遗产压力指数不超过阈值和居民满意度不低于基准水平。

1.4.3 游客分布平衡策略

为实现旅游分布的平衡，模型结合数字技术提出以下策略：

1. **智能推荐系统**：基于游客偏好和历史行为数据，构建个性化推荐算法。当宏村游客接近承载极限时，系统自动向西递分流游客。推荐策略考虑游客兴趣标签（如对古建筑、民俗文化的偏好），提高分流效果。

2. **动态定价机制**：利用模型生成的供需预测，实施动态票价制度。在旅游旺季，适当提高宏村门票价格，同时为西递提供价格优惠，引导游客选择。

3. **增强现实 (AR) 体验提升**：为西递开发 AR 导览系统，通过虚拟重现历史场景增强吸引力。例如，游客可通过手机 APP 观看虚拟的古代生活场景，提升体验价值。

4. **游客行为激励**：建立积分奖励系统，对选择西递的游客给予积分奖励，可在当地商户兑换特色产品或服务。积分数量根据实时游客分布情况动态调整。

1.4.4 模型应用效果模拟

通过调整后的模型进行仿真，得到以下优化结果：

- 西递游客量增加 25%，旅游收入提升 30%
- 宏村游客密度降低 15%，文化遗产压力指数下降 20%
- 两村整体居民满意度提升 8
- 旅游收入总额增长 12%

仿真结果表明，模型能有效平衡古村落群的旅游分布，在保护文化遗产的同时提升经济效益。这种适配方法可推广至其他类似目的地，如苏州古城或沈阳历史文化街区。

二、 模型评价与推广

2.1 模型优点

1. **综合性**：模型同时考虑了经济、环境和社会三个维度，符合可持续发展理念 [1]。

2. **适应性**：通过参数调整，模型可适用于不同类型旅游目的地，如滨海城市三亚与古村落西递、宏村 [5]。

3. **实用性**：模型与数字技术（如 AI、大数据）结合，为旅游管理提供可操作的决策支持 [4,6]。

4. **动态性**：引入资金反馈机制，能够模拟政策干预的长期效果。

2.2 模型局限与改进方向

1. **数据依赖**：模型对数据质量要求较高，在数据不足地区应用受限。未来可结合遥感数据、社交媒体数据等替代数据源。

2. **简化假设**：模型假设变量间关系稳定，实际中可能存在非线性突变。可引入复杂系统理论增强模型动态性 [2]。

3. **技术门槛**：数字技术的应用需要相应基础设施和人才支持。可开发简化版模型适用于资源有限地区。

2.3 推广价值

本模型可推广至各类旅游目的地可持续发展管理：

- 自然风景区：如黄山、庐山等，重点优化生态承载压力
- 城市文化遗产：如苏州古城、沈阳历史文化街区，侧重文化保护与旅游平衡
- 乡村生态旅游：如宜兴善卷洞，关注社区受益与生态保护 [9]

通过调整模型参数和指标体系，可构建适用于不同场景的可持续发展管理工具，为旅游目的地提供科学决策支持。

参考文献

[1] 三亚市统计局. 三亚市统计年鉴 2023[R]. 三亚: 三亚市统计局, 2023.

[2] 三亚市旅游和文化广电体育局. 2023 年三亚市旅游业发展统计公报 [R]. 三亚: 三亚市旅游和文化广电体育局, 2024.

[3] 海南省生态环境厅. 2022 年海南省海洋生态环境状况公报 [R]. 海口: 海南省生态环境厅, 2023.

[4] 三亚市住房和城乡建设局. 三亚市住房发展报告 2023[R]. 三亚: 三亚市住房和城乡建设局, 2023.

[5] 中国旅游研究院. 中国国内旅游发展年度报告 2023[R]. 北京: 中国旅游研究院, 2024.

[6] 中国科学院地理科学与资源研究所. 中国滨海旅游城市可持续发展评估报告 [R]. 北京: 中国科学院地理科学与资源研究所, 2022.

[7] 海南省发展和改革委员会. 海南省旅游业高质量发展三年行动计划 (2023-2025 年)[Z]. 海口: 海南省发展和改革委员会, 2023.

[8] United Nations World Tourism Organization. Overtourism? Understanding and Managing Urban Tourism Growth beyond Perceptions[R]. Madrid: UNWTO, 2022.

[9] 国外经典旅游目的地选择模型述评.

[10] 宁晓菊，张立，杨璐瑶，等. 极端气候影响下黄河中下游地区宜居水平时空变化 [J]. 干旱区地理，2025.

[11] 蔡国琴，李敏纳. 基于 STIRPAT 模型的海南省旅游业碳排放影响因素分析 [J]. 绿色科技，2023, 25(5): 254–257.

[12] 鄢慧丽，徐帆，王强，等. 全域旅游背景下基于居民感知视角的海南省旅游影响研究 [J]. 西北师范大学学报（自然科学版），2018, 54(5): 99–106.

[13] 欧阳瑞易. 基于 AHP 模糊综合评价法的老旧小区改造居民满意度研究 [D]. 南昌大学，2023.

[14] 黄芬，李小文. 基于 Meta 分析的居民休闲满意度影响因素研究 [J]. 安徽师范大学学报（自然科学版），2025, 48(3): 281–287.

[15] 叶鸿蔚，许朦. 基于 AHP-FCE 的南京市城乡社区治理绩效评价 [J]. 重庆三峡学院学报，2024, 40(4): 81–93.

[16] 石磊，夏敏，唐婷，等. 基于遥感生态指数模型 (RSEI) 的新安江流域 (宣城段) 生态环境质量状况分析 [J]. 安徽地质，2025, 35(2): 151–153.

[17] 杨惠楠，张伟冰，韦妮园，等. 基于综合指数模型的资源环境承载能力监测预警研究——以广西为例 [J]. 环境科学与管理，2025, 50(11): 15–19.

[18] 陆超，沈艳，张恒志. 居民公共卫生服务满意度调查及影响因素分析 [J]. 中国医疗管理科学，2024, 14(3): 95–100.

[19] 苏绮凌，陈文科. 居民幸福指数评价指标体系研究——以海南省为例 [J]. 调研世界，2015(7): 47–52.

[20] 张璐. 旅游地居民生活满意度测评及其影响因素研究——以福建土楼 (南靖) 旅游景区为例 [D]. 华侨大学，2021.

[21] 陈攀博. 旅游发展背景下东极居民社会治理满意度研究 [D]. 浙江海洋大学，2022.

[22] 王昊，张书齐，吴思彤，等. 中国城市更新投资环境指数模型构建与实证研究 [J]. 城市发展研究，2023, 30(3): 122–129.

附录 A　数据来源说明

本研究所用数据主要来源于：

1. 《三亚市统计年鉴》（2013-2023 年）
2. 《安徽省旅游统计年鉴》（2010-2023 年）
3. 国家文化旅游部公开数据
4. 相关学术文献及研究报告

附录 B　核心代码框架

```
#论文相关算法
import pandas as pd
import numpy as np
import math
from sklearn.preprocessing import StandardScaler,MinMaxScaler #数据处理模块
np.set_printoptions(precision=5,suppress=True)
Excelfile=pd.ExcelFile("三亚过夜游客统计新表.xlsx")
var={
    "Y":"年份",
    "T1":"公路通车里程",
    "T2":"码头泊位个数",
    "T3":"客船",
    "T4":"客位",
    "T5":"铁路通车里程",
    "T6":"民航主要航线"

}
df=pd.read_excel(Excelfile,skiprows=0,sheet_name="Sheet3",usecols="A:G")
df.columns=["Y","T1","T2","T3","T4","T5","T6"]
Y=df["Y"].to_list()
T1=df["T1"].to_list()
T2=df["T2"].to_list()
T3=df["T3"].to_list()
T4=df["T4"].to_list()
T5=df["T5"].to_list()
T6=df["T6"].to_list()

df=pd.DataFrame({"T1":T1,"T2":T2,"T3":T3,"T4":T4,"T5":T5,"T6":T6})
cost_columns=[]
benefit_columns=["T1","T2","T3","T4","T5","T6"]
#创建预处理函数
class Critic:
     def preproc(df, cost_columns,benefit_columns):
          X=df.values.astype(float)#将X转化为np数组，转化为浮点数，若为整数，最后结果
```

```
        xmin=X.min(axis=0)
        xmax=X.max(axis=0)
        xmaxmin=xmax-xmin
        m,n=X.shape
        for i in range(m):
            for j in range(n):
                if df.columns[j] in cost_columns:
                    X[i,j] = float((xmax[j] - X[i,j]) / xmaxmin[j])
                elif df.columns[j] in benefit_columns:
                    X[i,j] =  float((X[i,j] - xmin[j]) / xmaxmin[j])
        return X



    def conflict(X_standard):  #计算标准差 (对比强度)
        return np.std(X_standard,axis=0)



    def relate(X_standard): #计算变量相关性与冲突性
        m,n=X_standard.shape
        copy=pd.DataFrame(X_standard,columns=df.columns[0:])
        cometrix=copy.corr()
        f=[0.0]*n
        co_values=cometrix.values
        for j in range(n):
            for i in range(n):
                f[j]=(1-((co_values[i,j])**2)**0.5)+f[j]
            f[j]=float(f[j])
        return f

def critic(df,cost_columns,benefit_columns):
    X_standard=Critic.preproc(df,cost_columns,benefit_columns) #得到归一化矩阵
    n=X_standard.shape[1]
    R=Critic.conflict(X_standard)
```

```
    f=Critic.relate(X_standard)   #相关性
    information_content=[R[i]*f[i] for i in range(n)]  #计算信息量
    weight=np.array([information_content[i]/sum(information_content) for i in range(
    return weight


#topsis排序法
def topsis(X_standard,weight_row,lenth):
#构造加权规范矩阵
    weight_matrix=np.array(weight_row).reshape(1,-1) #将权重转换成矩阵，reshape保留原
    X_weight=X_standard*weight_matrix

#确定正理想解与负理想解
    Vmin=X_weight.min(axis=0)
    Vmax=X_weight.max(axis=0)


#计算每个评价对象到理想解的距离
    dist_best = np.sqrt(np.sum((X_weight - Vmax) ** 2, axis=1))
    dist_worst = np.sqrt(np.sum((X_weight - Vmin) ** 2, axis=1))

    #得到相对贴近度（每年的交通指数）
    T=np.array([float(dist_worst[i])/float((dist_best[i]+dist_worst[i])) for i in ra
    return T

#熵权法,忽略变量相关性
def entropy(df,cost_columns,benefit_columns):
#数据归一化，平移
    Y=Critic.preproc(df,cost_columns,benefit_columns)

#计算比重
    Y_sum=np.sum(Y,axis=0)
    divweight=Y/Y_sum
#计算熵值
    k=1/math.log(len(df))
```

```
    e=-k*(np.sum((divweight+1e-10)*np.log(divweight+1e-10),axis=0))
#计算差异系数
    g=1-e
#计算权重
    weight=g/np.sum(g)
    return  weight


#采用Z-score标准化
def standardlize(X):
    V=StandardScaler()
    X=V.fit_transform(X)
    return X
#采用MinMax归一化
def normalize(X):
    V=MinMaxScaler()
    if X.ndim==1:
        X=(X-X.min(axis=0))/(X.max(axis=0)-X.min(axis=0))

    else:
        X=V.fit_transform(X)
    return X

#计算环境指数
import pandas as pd
import numpy as np
import math



from critic import Critic,critic,topsis,entropy,standardlize,normalize
np.set_printoptions(precision=4,suppress=True)
```

```
Excelfile=pd.ExcelFile("三亚过夜游客统计新表.xlsx")

df=pd.read_excel(Excelfile,skiprows=0,sheet_name="Sheet5",usecols="B:G")
df.columns=["绿化覆盖率", "绿地率", "人均公共绿地面积", "森林覆盖率", "地表水达标率",
X=df.values



Xstandard=standardlize(X)
#采用主因素突出算法
E=np.prod(Xstandard,axis=1)
#将数据归一化
E1=normalize(E)
#过于突出极端值的影响使数据过于离散，舍去

cost_columns=[]
benefit_columns=df.columns.to_list()
#熵权法
weight_entro=entropy(df,cost_columns,benefit_columns)
E2=topsis(Critic.preproc(df,cost_columns,benefit_columns),weight_entro,11)

print(E2)

#计算社会压力
import pandas as pd
import numpy as np
from critic import entropy,critic,topsis,Critic,normalize
from multiple_variants_linear import tourismmodel
ExcelFile=pd.ExcelFile("三亚过夜游客统计新表.xlsx")
df=pd.read_excel(ExcelFile,skiprows=0,usecols="B:S",sheet_name="Sheet8")
df.columns=["人均生产总值（元）","常住人口（万人)","城镇化率（%)","人口密度（人/平方公
            "森林覆盖率","城镇生活污水集中处理率","PM2.5（g/m³)" ,"NO （g/m³)","SO （
            "水资源总量（亿立方米）","每万元GDP水耗（立方米）","人均用水量（平方米/人
            "游客数量（万人）","交通指数","环境指数","海南省A级景区数量"]
cost_cols=df.columns.to_list()
```

```
benefit_cols=[]
EL_=pd.DataFrame(critic(df,cost_cols,benefit_cols).reshape(1,-1),columns=df.columns.t
EL1_df=EL_[["人均生产总值（元）","常住人口（万人)","城镇化率（%)","人口密度（人/平方公
EL2_df=EL_[["森林覆盖率","城镇生活污水集中处理率","PM2.5（g/m³)" ,"NO （g/m³)","SO （
EL3_df=EL_[["水资源总量（亿立方米）","每万元GDP水耗（立方米）","人均用水量（平方米/人
EL4_df=EL_[["游客数量（万人）","交通指数","环境指数","海南省A级景区数量"]]
EL1_=np.sum(EL1_df.values,axis=1) #经济社会承载力权重
EL2_=np.sum(EL2_df.values,axis=1)    #环境承载力权重
EL3_=np.sum(EL3_df.values,axis=1) #资源承载力
EL4_=np.sum(EL4_df.values,axis=1)#旅游承载力
weight_EL1_=EL1_df.values/EL1_
weight_EL2_=EL2_df.values/EL2_
weight_EL3_=EL3_df.values/EL3_
weight_EL4_=EL4_df.values/EL4_

EL1_top=topsis(Critic.preproc(df[["人均生产总值（元）","常住人口（万人)","城镇化率（%
EL2_top=topsis(Critic.preproc(df[["森林覆盖率","城镇生活污水集中处理率","PM2.5（g/m³
EL3_top=topsis(Critic.preproc(df[["水资源总量（亿立方米）","每万元GDP水耗（立方米）",
EL4_top=topsis(Critic.preproc(df[["游客数量（万人）","交通指数","环境指数","海南省A级
#主因素突出算法,#归一化处理
EL_all=normalize(EL1_top*EL2_top*EL3_top*EL4_top)
print(EL_.values)

#皮尔逊相关系数矩阵计算及可视化
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
import itertools
import matplotlib.pyplot as plt
import seaborn as sns
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
class R2MatrixGenerator:
```

```
def __init__(self, df, target_column):
    """
    初始化R²矩阵生成器

    参数:
    df: 包含所有变量的DataFrame
    target_column: 目标变量列名
    """
    self.df = df.copy()
    self.target_column = target_column
    self.feature_columns = [col for col in df.columns if col != target_column]
    self.scaler = StandardScaler()
    self.results = {}

def get_all_feature_combinations(self, max_features=None):
    """
    获取所有可能的特征组合

    参数:
    max_features: 最大特征数量, None表示使用所有特征
    """
    if max_features is None:
        max_features = len(self.feature_columns)

    all_combinations = []
    for k in range(1, max_features + 1):
        combinations = list(itertools.combinations(self.feature_columns, k))
        all_combinations.extend(combinations)

    return all_combinations

def calculate_r2_for_combination(self, feature_combination):
    """
    计算特定特征组合的R²值
    """
```

```
        try:
            # 提取特征和目标
            X = self.df[list(feature_combination)].values
            y = self.df[self.target_column].values

            # 标准化特征
            X_scaled = self.scaler.fit_transform(X)

            # 训练线性回归模型
            model = LinearRegression()
            model.fit(X_scaled, y)

            # 预测并计算R²
            y_pred = model.predict(X_scaled)
            r2 = r2_score(y, y_pred)

            # 获取系数
            coefficients = model.coef_
            intercept = model.intercept_

            return r2, coefficients, intercept

        except Exception as e:
            print(f"计算组合 {feature_combination} 时出错: {e}")
            return 0, [], 0

    def generate_r2_matrix(self, max_features=None, sort_by_r2=True):
        """
        生成R²矩阵

        参数:
        max_features: 最大特征数量
        sort_by_r2: 是否按R²值排序
        """
        # 获取所有特征组合
```

```
        combinations = self.get_all_feature_combinations(max_features)

        print(f"共有 {len(combinations)} 种特征组合需要计算")

        # 计算每种组合的R²值
        results = []
        for i, combo in enumerate(combinations):
            if i % 50 == 0:  # 每50个组合打印一次进度
                print(f"计算进度: {i+1}/{len(combinations)}")

            r2, coefficients, intercept = self.calculate_r2_for_combination(combo)

            results.append({
                'combination': combo,
                'features': list(combo),
                'num_features': len(combo),
                'r2': r2,
                'coefficients': coefficients,
                'intercept': intercept
            })

        # 创建结果DataFrame
        results_df = pd.DataFrame(results)

        # 按R²值排序
        if sort_by_r2:
            results_df = results_df.sort_values('r2', ascending=False)

        # 重置索引
        results_df = results_df.reset_index(drop=True)

        self.results_df = results_df
        return results_df

    def create_r2_matrix_heatmap(self, top_n=20):
```

```
"""
创建R²值热力图矩阵

参数:
top_n: 显示前N个最佳组合
"""
if not hasattr(self, 'results_df'):
    print("请先运行 generate_r2_matrix()")
    return

# 选择前N个最佳组合
top_results = self.results_df.head(top_n).copy()

# 创建矩阵数据
matrix_data = []
feature_set = set()

for _, row in top_results.iterrows():
    features = row['features']
    r2 = row['r2']
    feature_set.update(features)

    # 为每个特征组合创建一行
    row_data = {'combination': ', '.join(features), 'r2': r2}
    for feature in feature_set:
        row_data[feature] = 1 if feature in features else 0
    matrix_data.append(row_data)

# 创建矩阵DataFrame
matrix_df = pd.DataFrame(matrix_data)

# 设置组合为索引
matrix_df = matrix_df.set_index('combination')

# 只保留特征列和R²列
```

```
    feature_cols = list(feature_set)
    matrix_df = matrix_df[feature_cols + ['r2']]

    # 创建热力图
    plt.figure(figsize=(12, 10))

    # 创建子图
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 12),
                                   gridspec_kw={'height_ratios': [1, 4]})

    # 上子图: R²值条形图
    r2_values = matrix_df['r2'].values
    colors = plt.cm.viridis((r2_values - r2_values.min()) / (r2_values.max() - r2
    ax1.bar(range(len(r2_values)), r2_values, color=colors)
    ax1.set_xticks(range(len(r2_values)))
    ax1.set_xticklabels(matrix_df.index, rotation=45, ha='right')
    ax1.set_ylabel('R²值')
    ax1.set_title('不同特征组合的R²值')

    # 下子图: 特征组合热力图
    feature_matrix = matrix_df[feature_cols].T
    sns.heatmap(feature_matrix, annot=True, cmap='Blues',
                cbar_kws={'label': '特征存在 (1=是, 0=否)'},
                ax=ax2)
    ax2.set_ylabel('特征')
    ax2.set_xlabel('特征组合')
    ax2.set_title('特征组合矩阵')

    plt.tight_layout()
    plt.show()

    return matrix_df

def get_best_combinations(self, n=10):
    """
```

```
    获取前N个最佳特征组合
    """
    if not hasattr(self, 'results_df'):
        print("请先运行 generate_r2_matrix()")
        return

    return self.results_df.head(n)[['features', 'r2', 'num_features']]

def find_optimal_combination(self, r2_threshold=0.9, max_features=None):
    """
    寻找满足R²阈值的最简单特征组合

    参数:
    r2_threshold: R²阈值
    max_features: 最大特征数量
    """
    if not hasattr(self, 'results_df'):
        print("请先运行 generate_r2_matrix()")
        return

    # 筛选满足R²阈值的组合
    valid_combinations = self.results_df[self.results_df['r2'] >= r2_threshold]

    if valid_combinations.empty:
        print(f"没有找到R² >= {r2_threshold}的组合")
        return None

    # 按特征数量排序，选择最简单的组合
    simplest_combination = valid_combinations.sort_values('num_features').iloc[0]

    print(f"满足R² >= {r2_threshold}的最简单组合:")
    print(f"特征: {simplest_combination['features']}")
    print(f"R²值: {simplest_combination['r2']:.4f}")
    print(f"特征数量: {simplest_combination['num_features']}")
```

```
        return simplest_combination

# 使用示例
def main():
    # 创建示例数据 (替换为您的实际数据)
    ExcelFile=pd.ExcelFile("三亚过夜游客统计新表.xlsx")
    df=pd.read_excel(ExcelFile,skiprows=0,usecols="B:F",sheet_name="Sheet6")
    df.columns=["总收入","游客数量","交通指数","环境指数","海南省A级景区数"]

    # 生成示例数据
    data = {
        '游客数量': np.array(df["游客数量"].to_list()),
        '交通指数': np.array(df["交通指数"].to_list()),
        '环境指数': np.array(df["环境指数"].to_list()),
        '海南省A级景区数': np.array(df["海南省A级景区数"].to_list()),
        '总收入':np.array(df["总收入"].to_list())
    }

    df = pd.DataFrame(data)

    # 创建R²矩阵生成器
    r2_generator = R2MatrixGenerator(df, '总收入')

    # 生成R²矩阵 (限制最大特征数为4以加快计算)
    print("开始计算R²矩阵...")
    r2_matrix = r2_generator.generate_r2_matrix(max_features=4)

    # 显示前10个最佳组合
    print("\n前10个最佳特征组合:")
    best_combinations = r2_generator.get_best_combinations(10)
    for i, (_, row) in enumerate(best_combinations.iterrows()):
        print(f"{i+1}. 特征: {row['features']}, R²: {row['r2']:.4f}, 特征数: {row['nu

    # 创建热力图矩阵
    print("\n生成热力图矩阵...")
```

```
    heatmap_matrix = r2_generator.create_r2_matrix_heatmap(top_n=15)

    # 寻找最优组合
    print("\n寻找最优特征组合...")
    optimal = r2_generator.find_optimal_combination(r2_threshold=0.8)

    # 保存结果到Excel
    r2_matrix.to_excel('r2_matrix_results.xlsx', index=False)
    print("\n结果已保存到 'r2_matrix_results.xlsx'")

    return r2_generator, r2_matrix

# 针对您的实际数据的使用方法
def use_with_your_data():
    """
    使用您自己的数据
    """
    # 读取您的数据
    # df = pd.read_excel("您的数据文件.xlsx")

    # 假设您的DataFrame已经加载，目标变量是'总收入'
    # r2_generator = R2MatrixGenerator(df, '总收入')

    # 生成R²矩阵
    # r2_matrix = r2_generator.generate_r2_matrix(max_features=5)  # 限制最大特征数

    # 显示结果
    # print(r2_matrix.head(10))

    pass

if __name__ == "__main__":
    r2_generator, r2_matrix = main()

#交通指数计算
```

```
import pandas as pd
import numpy as np
from critic import Critic,critic,topsis

np.set_printoptions(precision=4,suppress=True)

Excelfile=pd.ExcelFile("三亚过夜游客统计新表.xlsx")

df=pd.read_excel(Excelfile,skiprows=0,sheet_name="Sheet3",usecols="B:F")
df.columns=["公路通车里程","客船","客位","铁路通车里程","民航主要航线"]

cost_columns=[]
benefit_columns=df.columns.to_list()
weight_critic=critic(df,cost_columns,benefit_columns)
T1=topsis(Critic.preproc(df,cost_columns,benefit_columns),weight_critic,11)

print(T1)


#资金流入对旅游总收入函数拟合
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import warnings
warnings.filterwarnings('ignore')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 数据准备
years = list(range(2013, 2023))
tourism_revenue = [217.2, 258, 291.9, 305.5, 370.4, 469.91, 525.33, 417.73, 743.2, 43
```

```
gov_expenditure = [1852589, 1974596, 2039427, 2428284, 2209223, 2585252, 2686338, 263

P_current = np.array(tourism_revenue[:-1])
P_next_actual = np.array(tourism_revenue[1:])
AR_next = np.array(gov_expenditure[1:])

print("原始数据R²基准:")
baseline_r2 = r2_score(P_next_actual, P_current)
print(f"用本年预测下年 R² = {baseline_r2:.4f}")

# 定义函数形式
def linear_func(X, a, b):
    P_curr, AR_nxt = X
    return a * P_curr + b * (AR_nxt/1000000)

def power_func(X, a, b, c):
    P_curr, AR_nxt = X
    return a * (P_curr ** b) * ((AR_nxt/1000000) ** c)

def exp_func(X, a, b, c):
    P_curr, AR_nxt = X
    return a * np.exp(b * P_curr + c * (AR_nxt/1000000))

def log_func(X, a, b, c):
    P_curr, AR_nxt = X
    return a * np.log1p(P_curr) + b * np.log1p(AR_nxt/1000000) + c

# 分段拟合
print("\n" + "="*60)
print("成功拟合的函数及其参数")
print("="*60)

successful_models = []

# 疫情前模型
```

```
pre_covid_mask = np.array([year < 2020 for year in years[1:]])
if sum(pre_covid_mask) >= 3:
    P_curr_pre = P_current[pre_covid_mask]
    P_next_pre = P_next_actual[pre_covid_mask]
    AR_next_pre = AR_next[pre_covid_mask]

    print("\n疫情前模型 (2014-2019):")
    print("-" * 40)

    # 线性函数
    try:
        popt, pcov = curve_fit(linear_func, (P_curr_pre, AR_next_pre), P_next_pre, p0
        pred = linear_func((P_curr_pre, AR_next_pre), *popt)
        r2 = r2_score(P_next_pre, pred)

        # 输出带参数值的函数形式
        print(f"线性函数:")
        print(f"  P_next = {popt[0]:.6f} * P_curr + {popt[1]:.6f} * (AR_next/1000000)
        print(f"  R² = {r2:.4f}")

        successful_models.append(('疫情前线性模型', linear_func, popt, r2))
    except Exception as e:
        print(f"线性函数拟合失败: {e}")

    # 幂函数
    try:
        popt, pcov = curve_fit(power_func, (P_curr_pre, AR_next_pre), P_next_pre, p0=
        pred = power_func((P_curr_pre, AR_next_pre), *popt)
        r2 = r2_score(P_next_pre, pred)

        # 输出带参数值的函数形式
        print(f"\n幂函数:")
        print(f"  P_next = {popt[0]:.6f} * (P_curr^{popt[1]:.6f}) * ((AR_next/1000000
        print(f"  R² = {r2:.4f}")
```

```
            successful_models.append(('疫情前幂函数模型', power_func, popt, r2))
        except Exception as e:
            print(f"幂函数拟合失败: {e}")

        # 指数函数
        try:
            popt, pcov = curve_fit(exp_func, (P_curr_pre, AR_next_pre), P_next_pre, p0=[1
            pred = exp_func((P_curr_pre, AR_next_pre), *popt)
            r2 = r2_score(P_next_pre, pred)

            # 输出带参数值的函数形式
            print(f"\n指数函数:")
            print(f"  P_next = {popt[0]:.6f} * exp({popt[1]:.6f} * P_curr + {popt[2]:.6f}
            print(f"  R² = {r2:.4f}")

            successful_models.append(('疫情前指数模型', exp_func, popt, r2))
        except Exception as e:
            print(f"指数函数拟合失败: {e}")

    # 疫情后模型
    post_covid_mask = ~pre_covid_mask
    if sum(post_covid_mask) >= 2:
        P_curr_post = P_current[post_covid_mask]
        P_next_post = P_next_actual[post_covid_mask]
        AR_next_post = AR_next[post_covid_mask]

        print("\n疫情后模型 (2020-2022):")
        print("-" * 40)

        # 线性函数
        try:
            popt, pcov = curve_fit(linear_func, (P_curr_post, AR_next_post), P_next_post,
            pred = linear_func((P_curr_post, AR_next_post), *popt)
            r2 = r2_score(P_next_post, pred)
```

```
            # 输出带参数值的函数形式
            print(f"线性函数:")
            print(f"  P_next = {popt[0]:.6f} * P_curr + {popt[1]:.6f} * (AR_next/1000000)
            print(f"  R² = {r2:.4f}")

            successful_models.append(('疫情后线性模型', linear_func, popt, r2))
        except Exception as e:
            print(f"线性函数拟合失败: {e}")

        # 对数函数
        try:
            popt, pcov = curve_fit(log_func, (P_curr_post, AR_next_post), P_next_post, p0
            pred = log_func((P_curr_post, AR_next_post), *popt)
            r2 = r2_score(P_next_post, pred)

            # 输出带参数值的函数形式
            print(f"\n对数函数:")
            print(f"  P_next = {popt[0]:.6f} * ln(1+P_curr) + {popt[1]:.6f} * ln(1+AR_nex
            print(f"  R² = {r2:.4f}")

            successful_models.append(('疫情后对数模型', log_func, popt, r2))
        except Exception as e:
            print(f"对数函数拟合失败: {e}")

    # 机器学习模型
    print("\n机器学习模型:")
    print("-" * 40)

    # 准备特征矩阵
    X = np.column_stack([
        P_current,
        AR_next / 1000000,
        np.arange(len(P_current))
    ])
    y = P_next_actual
```

```
# 线性回归
lr = LinearRegression()
lr.fit(X, y)
lr_r2 = r2_score(y, lr.predict(X))

# 输出带参数值的函数形式
print(f"线性回归:")
print(f"  P_next = {lr.intercept_:.6f} + {lr.coef_[0]:.6f} * P_curr + {lr.coef_[1]:.6
print(f"  R² = {lr_r2:.4f}")

successful_models.append(('线性回归', '机器学习', lr.coef_, lr_r2))

# 随机森林
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)
rf_r2 = r2_score(y, rf.predict(X))

# 输出函数形式
print(f"\n随机森林:")
print(f"  R² = {rf_r2:.4f}")
print(f"  特征重要性: P_curr = {rf.feature_importances_[0]:.4f}, AR_next = {rf.featur

successful_models.append(('随机森林', '机器学习', rf.feature_importances_, rf_r2))

# 总结所有成功模型
print("\n" + "="*60)
print("所有成功拟合的模型总结")
print("="*60)

print(f"\n总共成功拟合 {len(successful_models)} 个模型:")
for i, (name, func_type, params, r2) in enumerate(successful_models, 1):
    print(f"\n{i}. {name}")
    print(f"   R²: {r2:.4f}")
```

```
    if func_type == '机器学习':
        if name == '线性回归':
            print(f"   具体函数: P_next = {lr.intercept_:.6f} + {params[0]:.6f} * P_c
        else:
            print(f"   特征重要性: P_curr = {params[0]:.4f}, AR_next = {params[1]:.4f
    else:
        # 根据函数类型输出不同的带参数值的函数形式
        if name == '疫情前线性模型' or name == '疫情后线性模型':
            print(f"   具体函数: P_next = {params[0]:.6f} * P_curr + {params[1]:.6f}
        elif name == '疫情前幂函数模型':
            print(f"   具体函数: P_next = {params[0]:.6f} * P_curr^{params[1]:.6f} *
        elif name == '疫情前指数模型':
            print(f"   具体函数: P_next = {params[0]:.6f} * exp({params[1]:.6f} * P_c
        elif name == '疫情后对数模型':
            print(f"   具体函数: P_next = {params[0]:.6f} * ln(1+P_curr) + {params[1]

# 绘制包含全部拟合函数及相应优度的图
print(f"\n绘制包含全部拟合函数及相应优度的图...")

# 创建图像
plt.figure(figsize=(15, 10))

# 绘制实际值
years_plot = years[1:]  # 使用2014-2022年
plt.plot(years_plot, P_next_actual, 'ko-', linewidth=3, markersize=10, label='实际旅游

# 定义颜色和线型
colors = ['red', 'blue', 'green', 'orange', 'purple', 'brown']
line_styles = ['-', '--', '-.', ':', '-', '--']

# 绘制每个模型的拟合曲线
for i, (name, func_type, params, r2) in enumerate(successful_models):
    color = colors[i % len(colors)]
    line_style = line_styles[i % len(line_styles)]
```

```
    # 计算预测值
    if func_type == '机器学习':
        if name == '线性回归':
            predictions = lr.predict(X)
        else:
            predictions = rf.predict(X)
    else:
        predictions = func_type((P_current, AR_next), *params)

    # 绘制拟合曲线
    plt.plot(years_plot, predictions, color=color, linestyle=line_style,
             linewidth=2, label=f'{name} (R²={r2:.4f})')

# 设置图表属性
plt.xlabel('年份', fontsize=14)
plt.ylabel('旅游收入 (亿元)', fontsize=14)
plt.title('全部拟合函数对比及优度', fontsize=16)
plt.legend(fontsize=12, loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, alpha=0.3)

# 添加数据标签
for i, (year, actual) in enumerate(zip(years_plot, P_next_actual)):
    plt.annotate(f'{actual:.1f}', (year, actual), textcoords="offset points",
                 xytext=(0,10), ha='center', fontsize=9, weight='bold')

plt.tight_layout()
plt.show()

# 使用最佳模型进行预测的函数
def predict_with_model(model_info, P_curr, AR_nxt):
    name, func_type, params, r2 = model_info

    if func_type == '机器学习':
        if name == '线性回归':
            # params 是系数数组
```

```
            return params[0] * P_curr + params[1] * (AR_nxt/1000000) + params[2] * (1
        else:
            # 随机森林预测需要特征矩阵
            X_pred = np.array([[P_curr, AR_nxt/1000000, len(P_current)]])
            return rf.predict(X_pred)[0]
    else:
        # 函数模型
        return func_type((P_curr, AR_nxt), *params)

# 示例：使用最佳模型预测2023年
if successful_models:
    best_model = max(successful_models, key=lambda x: x[3])
    print(f"\n最佳模型：{best_model[0]} (R² = {best_model[3]:.4f})")

    # 使用最佳模型预测2023年
    P_2023_best = predict_with_model(best_model, tourism_revenue[-1], gov_expenditure
    print(f"使用最佳模型预测2023年旅游收入：{P_2023_best:.2f} 亿元")

print(f"\n建模完成！共成功拟合 {len(successful_models)} 个模型。")


#资金流入与社会压力函数建模
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import warnings
warnings.filterwarnings('ignore')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 数据准备
```

```
years = list(range(2013, 2023))
pressure_index = [0.6403, 1, 0.94446, 0.06208, 0.02094, 0.03374, 0, 0.30807, 0.28756,
gov_investment = [414708, 513465, 562004, 533121, 1168553, 1481362, 1632969, 1644036,

P_current = np.array(pressure_index[:-1])  # 当前期压力指数 (2013-2021)
P_next_actual = np.array(pressure_index[1:])  # 下一期压力指数 (2014-2022)
I_next = np.array(gov_investment[1:])  # 下一期政府投资 (2014-2022)

print("原始数据R²基准:")
baseline_r2 = r2_score(P_next_actual, P_current)
print(f"用本期预测下期 R² = {baseline_r2:.4f}")

# 定义函数形式
def linear_func(X, a, b, c):
    P_curr, I_nxt = X
    return a * P_curr + b * (I_nxt/1000000) + c

def power_func(X, a, b, c, d):
    P_curr, I_nxt = X
    return a * (P_curr ** b) * ((I_nxt/1000000) ** c) + d

def exp_func(X, a, b, c, d):
    P_curr, I_nxt = X
    return a * np.exp(b * P_curr + c * (I_nxt/1000000)) + d

def log_func(X, a, b, c, d):
    P_curr, I_nxt = X
    return a * np.log1p(P_curr) + b * np.log1p(I_nxt/1000000) + c * (I_nxt/1000000) +

def rational_func(X, a, b, c, d):
    P_curr, I_nxt = X
    return a * P_curr + b / (1 + c * (I_nxt/1000000)) + d

# 分段拟合
print("\n" + "="*60)
```

```
print("成功拟合的函数及其参数")
print("="*60)

successful_models = []

# 疫情前模型 (2014-2019)
pre_covid_mask = np.array([year < 2020 for year in years[1:]])
if sum(pre_covid_mask) >= 3:
    P_curr_pre = P_current[pre_covid_mask]
    P_next_pre = P_next_actual[pre_covid_mask]
    I_next_pre = I_next[pre_covid_mask]

    print("\n疫情前模型 (2014-2019):")
    print("-" * 40)

    # 线性函数
    try:
        popt, pcov = curve_fit(linear_func, (P_curr_pre, I_next_pre), P_next_pre, p0=
        pred = linear_func((P_curr_pre, I_next_pre), *popt)
        r2 = r2_score(P_next_pre, pred)

        # 输出带参数值的函数形式
        print(f"线性函数:")
        print(f"  P_next = {popt[0]:.6f} * P_curr + {popt[1]:.6f} * (I_next/1000000)
        print(f"  R² = {r2:.4f}")

        successful_models.append(('疫情前线性模型', linear_func, popt, r2))
    except Exception as e:
        print(f"线性函数拟合失败: {e}")

    # 幂函数
    try:
        popt, pcov = curve_fit(power_func, (P_curr_pre, I_next_pre), P_next_pre, p0=[
        pred = power_func((P_curr_pre, I_next_pre), *popt)
        r2 = r2_score(P_next_pre, pred)
```

```
        # 输出带参数值的函数形式
        print(f"\n幂函数:")
        print(f"  P_next = {popt[0]:.6f} * (P_curr^{popt[1]:.6f}) * ((I_next/1000000)
        print(f"  R² = {r2:.4f}")

        successful_models.append(('疫情前幂函数模型', power_func, popt, r2))
    except Exception as e:
        print(f"幂函数拟合失败: {e}")

    # 指数函数
    try:
        popt, pcov = curve_fit(exp_func, (P_curr_pre, I_next_pre), P_next_pre, p0=[1,
        pred = exp_func((P_curr_pre, I_next_pre), *popt)
        r2 = r2_score(P_next_pre, pred)

        # 输出带参数值的函数形式
        print(f"\n指数函数:")
        print(f"  P_next = {popt[0]:.6f} * exp({popt[1]:.6f} * P_curr + {popt[2]:.6f}
        print(f"  R² = {r2:.4f}")

        successful_models.append(('疫情前指数模型', exp_func, popt, r2))
    except Exception as e:
        print(f"指数函数拟合失败: {e}")

# 疫情后模型 (2020-2022)
post_covid_mask = ~pre_covid_mask
if sum(post_covid_mask) >= 2:
    P_curr_post = P_current[post_covid_mask]
    P_next_post = P_next_actual[post_covid_mask]
    I_next_post = I_next[post_covid_mask]

    print("\n疫情后模型 (2020-2022):")
    print("-" * 40)
```

```
# 线性函数
try:
    popt, pcov = curve_fit(linear_func, (P_curr_post, I_next_post), P_next_post,
    pred = linear_func((P_curr_post, I_next_post), *popt)
    r2 = r2_score(P_next_post, pred)

    # 输出带参数值的函数形式
    print(f"线性函数:")
    print(f"  P_next = {popt[0]:.6f} * P_curr + {popt[1]:.6f} * (I_next/1000000)
    print(f"  R² = {r2:.4f}")

    successful_models.append(('疫情后线性模型', linear_func, popt, r2))
except Exception as e:
    print(f"线性函数拟合失败: {e}")

# 对数函数
try:
    popt, pcov = curve_fit(log_func, (P_curr_post, I_next_post), P_next_post, p0=
    pred = log_func((P_curr_post, I_next_post), *popt)
    r2 = r2_score(P_next_post, pred)

    # 输出带参数值的函数形式
    print(f"\n对数函数:")
    print(f"  P_next = {popt[0]:.6f} * ln(1+P_curr) + {popt[1]:.6f} * ln(1+I_next
    print(f"  R² = {r2:.4f}")

    successful_models.append(('疫情后对数模型', log_func, popt, r2))
except Exception as e:
    print(f"对数函数拟合失败: {e}")

# 有理函数
try:
    popt, pcov = curve_fit(rational_func, (P_curr_post, I_next_post), P_next_post
    pred = rational_func((P_curr_post, I_next_post), *popt)
    r2 = r2_score(P_next_post, pred)
```

```
        # 输出带参数值的函数形式
        print(f"\n有理函数:")
        print(f"  P_next = {popt[0]:.6f} * P_curr + {popt[1]:.6f} / (1 + {popt[2]:.6f
        print(f"  R² = {r2:.4f}")

        successful_models.append(('疫情后有理模型', rational_func, popt, r2))
    except Exception as e:
        print(f"有理函数拟合失败: {e}")

# 机器学习模型
print("\n机器学习模型:")
print("-" * 40)

# 准备特征矩阵
X = np.column_stack([
    P_current,
    I_next / 1000000,
    np.arange(len(P_current))
])
y = P_next_actual

# 线性回归
lr = LinearRegression()
lr.fit(X, y)
lr_r2 = r2_score(y, lr.predict(X))

# 输出带参数值的函数形式
print(f"线性回归:")
print(f"  P_next = {lr.intercept_:.6f} + {lr.coef_[0]:.6f} * P_curr + {lr.coef_[1]:.6
print(f"  R² = {lr_r2:.4f}")

successful_models.append(('线性回归', '机器学习', np.append(lr.coef_, lr.intercept_),

# 随机森林
```

```
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)
rf_r2 = r2_score(y, rf.predict(X))

# 输出函数形式
print(f"\n随机森林:")
print(f"  R² = {rf_r2:.4f}")
print(f"  特征重要性: P_curr = {rf.feature_importances_[0]:.4f}, I_next = {rf.feature

successful_models.append(('随机森林', '机器学习', rf.feature_importances_, rf_r2))

# 总结所有成功模型
print("\n" + "="*60)
print("所有成功拟合的模型总结")
print("="*60)

print(f"\n总共成功拟合 {len(successful_models)} 个模型:")
for i, (name, func_type, params, r2) in enumerate(successful_models, 1):
    print(f"\n{i}. {name}")
    print(f"   R²: {r2:.4f}")

    if func_type == '机器学习':
        if name == '线性回归':
            print(f"   具体函数: P_next = {params[3]:.6f} + {params[0]:.6f} * P_curr
        else:
            print(f"   特征重要性: P_curr = {params[0]:.4f}, I_next = {params[1]:.4f}
    else:
        # 根据函数类型输出不同的带参数值的函数形式
        if name == '疫情前线性模型' or name == '疫情后线性模型':
            print(f"   具体函数: P_next = {params[0]:.6f} * P_curr + {params[1]:.6f}
        elif name == '疫情前幂函数模型':
            print(f"   具体函数: P_next = {params[0]:.6f} * P_curr^{params[1]:.6f} *
        elif name == '疫情前指数模型':
            print(f"   具体函数: P_next = {params[0]:.6f} * exp({params[1]:.6f} * P_c
        elif name == '疫情后对数模型':
```

```
            print(f"   具体函数: P_next = {params[0]:.6f} * ln(1+P_curr) + {params[1]
        elif name == '疫情后有理模型':
            print(f"   具体函数: P_next = {params[0]:.6f} * P_curr + {params[1]:.6f}

# 绘制包含全部拟合函数及相应优度的图
print(f"\n绘制包含全部拟合函数及相应优度的图...")

# 创建图像
plt.figure(figsize=(15, 10))

# 绘制实际值
years_plot = years[1:]  # 使用2014-2022年
plt.plot(years_plot, P_next_actual, 'ko-', linewidth=3, markersize=10, label='实际压力

# 定义颜色和线型
colors = ['red', 'blue', 'green', 'orange', 'purple', 'brown', 'pink', 'gray']
line_styles = ['-', '--', '-.', ':', '-', '--', '-.', ':']

# 绘制每个模型的拟合曲线
for i, (name, func_type, params, r2) in enumerate(successful_models):
    color = colors[i % len(colors)]
    line_style = line_styles[i % len(line_styles)]

    # 计算预测值
    if func_type == '机器学习':
        if name == '线性回归':
            predictions = lr.predict(X)
        else:
            predictions = rf.predict(X)
    else:
        predictions = func_type((P_current, I_next), *params)

    # 绘制拟合曲线
    plt.plot(years_plot, predictions, color=color, linestyle=line_style,
             linewidth=2, label=f'{name} (R²={r2:.4f})')
```

```
# 设置图表属性
plt.xlabel('年份', fontsize=14)
plt.ylabel('压力指数', fontsize=14)
plt.title('压力指数预测模型对比及优度', fontsize=16)
plt.legend(fontsize=10, loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, alpha=0.3)

# 添加数据标签
for i, (year, actual) in enumerate(zip(years_plot, P_next_actual)):
    plt.annotate(f'{actual:.3f}', (year, actual), textcoords="offset points",
                 xytext=(0,10), ha='center', fontsize=8, weight='bold')

plt.tight_layout()
plt.show()

# 使用最佳模型进行预测的函数
def predict_with_model(model_info, P_curr, I_nxt):
    name, func_type, params, r2 = model_info

    if func_type == '机器学习':
        if name == '线性回归':
            # params 是系数数组 [coef0, coef1, coef2, intercept]
            return params[0] * P_curr + params[1] * (I_nxt/1000000) + params[2] * (le
        else:
            # 随机森林预测需要特征矩阵
            X_pred = np.array([[P_curr, I_nxt/1000000, len(P_current)]])
            return rf.predict(X_pred)[0]
    else:
        # 函数模型
        return func_type((P_curr, I_nxt), *params)

# 示例：使用最佳模型预测2023年
if successful_models:
    best_model = max(successful_models, key=lambda x: x[3])
```

```
    print(f"\n最佳模型: {best_model[0]} (R² = {best_model[3]:.4f})")

    # 使用最佳模型预测2023年
    P_2023_best = predict_with_model(best_model, pressure_index[-1], gov_investment[-
    print(f"使用最佳模型预测2023年压力指数: {P_2023_best:.6f}")

print(f"\n建模完成! 共成功拟合 {len(successful_models)} 个模型。")
```