数学建模在英语听说考试分数调整中的应用

一、摘要

研究背景与目的

某地市举行了一场英语听说考试，计划在一天内完成，上午和下午的试题不同，两份试题满分值都是 60 分。为了确保考试公平，主办方设定了一个规则：如果两次考试的均分差距超过 2 分，将对均分较低的一次考试的分数进行调整，以使均分差距不超过 2 分。实际考试结果显示，上午共有约 9000 人参加，均分为 42 分，最高分是 58 分，最低分是 0 分。下午共有约 700 人参加，均分为 28 分，最高分是 53 分，最低分是 0 分。两次考试成绩的标准差均为 10 分。为了完成赋分的公平性，本研究应用了数学建模技术，以解决该问题。

方法概述

我们利用正态分布和偏态分布生成模拟数据，确保数据量和范围符合要求。接着，应用分位数调整方法，确保两次考试在不同分位点上的成绩分布相似。最后，采用高斯混合模型精确拟合和调整下午考试的分布，使得调整后的成绩更加符合实际分布特征。

主要发现与结论

通过这些方法，我们成功将下午考试的成绩分布调整到与上午考试成绩分布相似，并将两次考试的均分差异控制在 2 分以内。研究结果表明，数学建模技术可以有效解决考试成绩调整的问题，确保考试的公平性和科学性。这为类似考试的分数调整提供了理论依据和实践方法。

二、引言

问题背景

英语听说考试的重要性和挑战

英语听说考试在评估学生的语言能力方面具有重要地位。与传统的笔试相比，听说考试能够更全面地考察学生的实际语言应用能力。因此，中考，高考都加入了听说考试。然而，在大规模考试中，如何确保考试的公平性和有效性是一项巨大的挑战。考试的时间安排、试题难度的均衡、考生的临场发挥等因素都可能影响考试结果，导致不同场次的考试成绩出现显著差异。

在某地市举行的一次英语听说考试中，考试被安排在一天内完成，分为上午和下午两场，分别有约 9000 人和 700 人参加。为保证考试的公平性，主办方设定了一个规则：如果两次考试的均分差距超过 2 分，将对均分较低的一次考试的分数进行调整。然而，实际考试结果显示，上午考试的均分为 42 分，下午考试的均分为 28 分，这一显著差异引发了如何科学合理地进行成绩调整的问题。

研究目的

通过数学建模调整考试分数，使不同场次考试成绩的均衡性。

本研究旨在通过数学建模的方法，解决这一成绩调整问题。我们拟定了一系列步骤，包括生成模拟数据、进行数据平滑处理、应用分位数调整方法以及使用高斯混合模型进行精确拟合和分布调整。

具体来说，研究目标是将下午考试的成绩分布调整到与上午考试相似，并将两次考试的均分差异控制在 2 分以内，从而确保考试结果的公平性和可比性。

研究意义

保障考试的公平性和有效性。

通过本研究，我们希望为大规模考试中的成绩调整提供一种科学有效的方法，确保考试的公平性和有效性。这不仅有助于教育机构在组织考试时更好地评估学生的真实水平，还可以减少因考试安排不当或试题难度差异引起的不公平现象。更广泛地，数学建模技术在考试成绩调整中的应用，能够为其他类似场景提供理论基础和实践参考，推动教育评估方法的科学化和标准化。

总之，本研究通过数学建模技术，旨在通过赋分，实现不同场次考试成绩的均衡性，保障考试的公平性，具有重要的理论意义和实际应用价值。

三、模型假设

1.假如题目所给数据真实有效。
2.假如学生考试成绩集中在平均值附近，极端高分和低分的学生较少。
3.假如上午和下午学生的总体实力差距不大。

四、研究方法

数据描述

上午考试数据：9000 名考生，均分 42 分，最高 58 分，最低 0 分，标准差 10 分
下午考试数据：700 名考生，均分 28 分，最高 53 分，最低 0 分，标准差 10 分

数学模型

1. 生成模拟数据

实际问题背景：

考试成绩通常会受多种因素影响，如学生的学习能力、考试难度、临场发挥等。通常情况下，考试成绩会呈现出一定的分布特征。

正态分布：

使用原因：考试成绩往往集中在平均值附近，极端高分和低分的学生较少，这符合正态分布的特征。

实际应用：例如，若大部分学生在上午考试中得分在 42 分左右，而少数学生得分极高或极低，正态分布能较好地模拟这种情况。

偏态分布：

使用原因：考试成绩可能因考试时间或难度差异而偏斜，使用偏态分布可以更贴近这种实际情况。

实际应用：下午考试相对较难，导致大部分学生成绩偏低，偏态分布能更准确地模拟这种情况。

环境噪声模拟：

使用原因：考试成绩可能因考试时间或难度差异而偏斜，使用偏态分布可以更贴近这种实际情况。

实际应用：下午考试相对较难，导致大部分学生成绩偏低，偏态分布能更准确地模拟这种情况。

重采样：

使用原因：初次生成的数据可能不完全符合指定范围或数量要求，重采样可以保证数据量和数据范围符合要求。

实际应用：重采样确保我们生成的 9000 个上午成绩和 700 个下午成绩都在合理范围内，并符合预

期的分布特征，提供足够的数据用于后续分析和调整。

2. 分位数调整

实际问题背景：

考试分数均值的差异可能由多种因素引起，为了确保考试的公平性，需要调整分数使其具有可比性。

分位数调整：

原因：通过调整不同分位数上的成绩，可以确保两次考试在不同分位点上的成绩分布具有相似性，从而减少均值差异。

实际应用：例如，若上午考试的 50 百分位成绩是 42 分，下午考试是 28 分，通过调整后使两次考试的 50 百分位成绩相近，确保两次考试的成绩具有可比性，并将均值差异控制在 2 分以内。

3. 高斯混合模型（GMM）

实际问题背景：

考试成绩分布可能较为复杂，单一的分布模型无法完全拟合，需要更灵活的模型来准确描述数据分布。

高斯混合模型（GMM）：

原因：GMM 能通过多个高斯分布的加权和来拟合复杂数据分布，适应数据的多模态特性。

实际应用：GMM 可以拟合上午考试的复杂分布，并从中采样生成与下午考试相似的分布，保证调整后的下午成绩与上午成绩的分布特征一致。

4. 高斯混合模型和分位数调整的优势

高斯混合模型的优势：

精确拟合：能准确拟合复杂分布，处理数据的多模态特性。

实际应用：比如，上午考试的成绩可能呈现多个高峰，通过 GMM 模型可以准确拟合这些特征，从而更好地调整下午考试的成绩分布。

分位数调整的优势

分布均衡：确保不同场次考试在各分位点上的成绩相似，减少均值差异。

简单有效：实现简单，能有效控制分数差距。

实际应用：通过分位数调整，可以直观地调整下午成绩，使其各分位点的成绩与上午成绩相似，最终保证均值差异不超过 2 分。

5.对使用数学方法的合理性的反思

数据分布假设：假设考试成绩符合正态或偏态分布可能不完全准确，实际分布可能更加复杂。

模型复杂性：使用 GMM 和分位数调整虽然精确，但增加了模型的复杂性，可能导致过拟合。

算法实现

程序由 python 编写，下面将解释如何通过算法实现

1. 数据生成：generate_data 函数的实现与调整

```
    def generate_data(a, loc, scale, size, lower, upper):
        # 使用偏态分布生成初始数据
        data = skewnorm.rvs(a=a, loc=loc, scale=scale, size=size)
        # 过滤数据，确保在指定范围内
        data = data[(data >= lower) & (data <= upper)]

        # 如果数据点不足，继续生成直到达到所需数量
        while len(data) < size:
            # 生成额外数据
            additional_data = skewnorm.rvs(a=a, loc=loc, scale=scale,
    size=(size - len(data)))
            # 过滤额外数据，确保在指定范围内
            additional_data = additional_data[(additional_data >=
    lower) & (additional_data <= upper)]
            # 将额外数据与已有数据合并
            data = np.concatenate((data, additional_data))

        # 返回最终数据，确保数据量为 size
        return data[:size]
```

generate_data 函数利用偏态分布（Skewed Normal Distribution）生成数据，并通过重取样算法确保生成的数据在指定的范围内（lower 到 upper 之间）。

```
# 生成数据
a_random_1 = round(ran.uniform(-0.1, 0.1), 3)  # a 值随机
a_random_2 = round(ran.uniform(-0.1, 0.1), 3)
#生成早上和中午的数据
morning_data = generate_data(a=a_random_1, loc=42, scale=10,
size=9000, lower=0, upper=58)
afternoon_data = generate_data(a=a_random_2, loc=28, scale=10,
size=700, lower=0, upper=53)
```

通过生成随机噪声种子 a1、a2，通过引入随机微调，避免数据调整过于机械化，使结果更自然。并通过 generate_data 函数生成贴合题目要求的早上和中午两组数据。

```
def adjust_data(data, a, loc, scale, size, lower, upper):
    data = data[(data >= lower) & (data <= upper)]  # 确保数据在范
围内
    # 如果数据点不足，继续生成直到达到所需数量
    while len(data) < size:
        additional_data = skewnorm.rvs(a=a, loc=loc, scale=scale,
size=(size - len(data)))
        additional_data = additional_data[(additional_data >=
lower) & (additional_data <= upper)]
        data = np.concatenate((data, additional_data))
    return data[:size]
```

abjust_data 函数用于过滤出在指定范围内的数据。如果数据点不足，继续生成数据并合并，直到数据点数量达到要求。以确保生成的数据在指定范围内，并且数据点数量达到要求。

```
#计算上午和下午数据的均值。
morning_mean = np.mean(morning_data)
afternoon_mean = np.mean(afternoon_data)
#计算当前均值与目标均值的差异。
morning_mean_diff = 42 - morning_mean
afternoon_mean_diff = 28 - afternoon_mean

#如果差异大于 0.2，调整数据并重新生成符合范围和数量要求的数据，迭代到当前均值与目标均值的差异小于 0.2
while (abs(morning_mean_diff) > 0.2 or abs(afternoon_mean_diff) > 0.2):
    if abs(morning_mean_diff) > 0.2:
        morning_data += morning_mean_diff
        morning_data = adjust_data(data=morning_data, a=a_random_1, loc=42, scale=10, size=9000, lower=0, upper=58)

    if abs(afternoon_mean_diff) > 0.2:
        afternoon_data += afternoon_mean_diff
        afternoon_data = adjust_data(data=afternoon_data, a=a_random_2, loc=28, scale=10, size=700, lower=0, upper=53)

    morning_mean = np.mean(morning_data)
    afternoon_mean = np.mean(afternoon_data)
    morning_mean_diff = 42 - morning_mean
    afternoon_mean_diff = 28 - afternoon_mean
```

通过使用 adjust_data 函数，迭代调整上午和下午数据的均值，使其接近预设的目标值（上午 42 分，下午 28 分）。

2.分位数调整与均值修正

```
    morning_sorted = np.sort(morning_data)
    afternoon_sorted = np.sort(afternoon_data)

    #进行分位数调整
    for percentile in np.linspace(0, 100, 101):
        morning_value = scoreatpercentile(morning_sorted, percentile)
        afternoon_value = scoreatpercentile(afternoon_sorted,
    percentile)
        adjustment = morning_value - afternoon_value
        afternoon_sorted[np.isclose(afternoon_sorted,
    afternoon_value, atol=1e-6)] += adjustment
```

对上午和下午数据进行排序。对每个百分位计算对应的值，并调整下午数据，使其与上午数据在每个百分位上的值相同。确保了两场考试在各个百分位上的成绩分布相似。

```
    morning_mean = np.mean(morning_sorted)
    afternoon_mean = np.mean(afternoon_sorted)
    mean_diff = morning_mean - afternoon_mean

    #均值调整调整
    if abs(mean_diff) > 2:
        adjustment = mean_diff - 2 if mean_diff > 0 else mean_diff +
    2
        addition_random = round(ran.uniform(0, 4), 2)
        afternoon_sorted += adjustment + addition_random

    # 确保调整后的数据在正确范围内
    afternoon_mean = np.mean(afternoon_sorted)
    afternoon_sorted = adjust_data(data=afternoon_sorted,
    a=a_random_2, loc=afternoon_mean, scale=10, size=700, lower=0,
    upper=60)
```

计算调整后的均值差异。如果差异超过 2 分，进行适当的调整。确保调整后的数据在 0 到 60 分范围内，并重新生成满足要求的数据。进一步调整下午数据的均值，使其与上午数据的均值相差不超过 2 分，并确保调整后的数据在正确范围内。

3.GMM 模型的拟合与分布调整

```
    # 创建高斯混合模型并进行数据拟合
    def fit_gmm(data, n_components):
        # n_components 表示处理精度，covariance_type='full'表示使用全协方差矩阵，random_state=42 设置随机种子（确保结果的可重复性）
        gmm = GaussianMixture(n_components=n_components, covariance_type='full', random_state=42)
        gmm.fit(data.reshape(-1, 1))  # 高斯混合模型需要二维输入，将数据调整为二维数组
        return gmm
```

fit_gum 函数根据输入数据创建并拟合高斯混合模型（GMM）。

```
    def adjust_data_distribution(morning_data, afternoon_data, n_components):
        morning_gmm = fit_gmm(morning_data, n_components)
        afternoon_gmm = fit_gmm(afternoon_data, n_components)

        morning_sampled, _ = morning_gmm.sample(len(afternoon_data))  # 从早上模型中采样，生成与下午数据长度相同的相似分布
        afternoon_adjusted = morning_sampled.flatten()  # 将采样的数据调整为一维数组

        afternoon_adjusted = np.clip(afternoon_adjusted, np.min(afternoon_data),np.max(afternoon_data))  # 将调整后的数据限制在下午数据的最小值和最大值之间
        return morning_gmm, afternoon_gmm, afternoon_adjusted

    #返回结果
    morning_gmm, afternoon_gmm, afternoon_adjusted = adjust_data_distribution(morning_sorted, afternoon_sorted,n_components=5)
```

使用 fit_gmm 函数分别拟合上午和下午的 GMM 模型。调整下午考试数据的分布，使其与上午考试数据的分布相匹配。返回拟合后的结果。

五、数据处理与结果

1. 数据生成与模拟

根据题目给出数据，通过算法生成的原始分数数据如图 1，图 2：

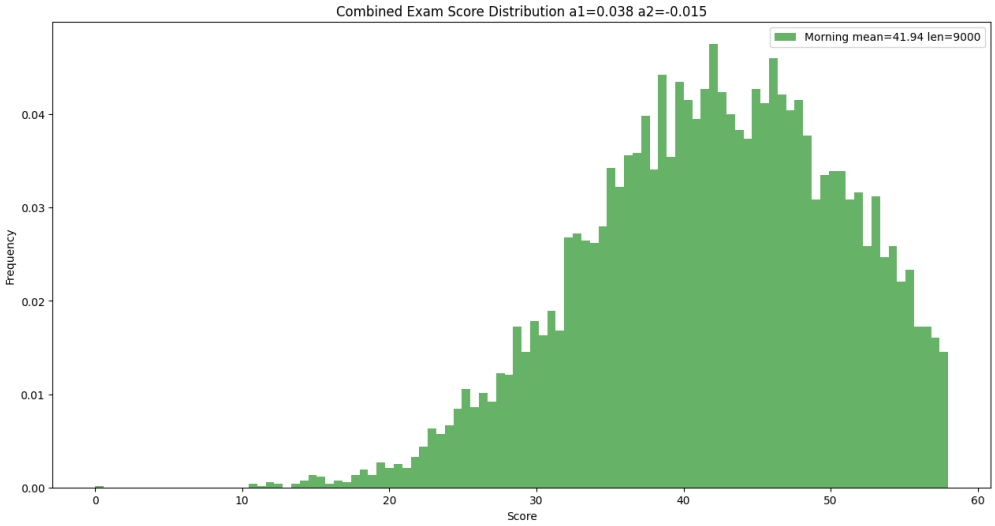

图 1　早上生成成绩直方图

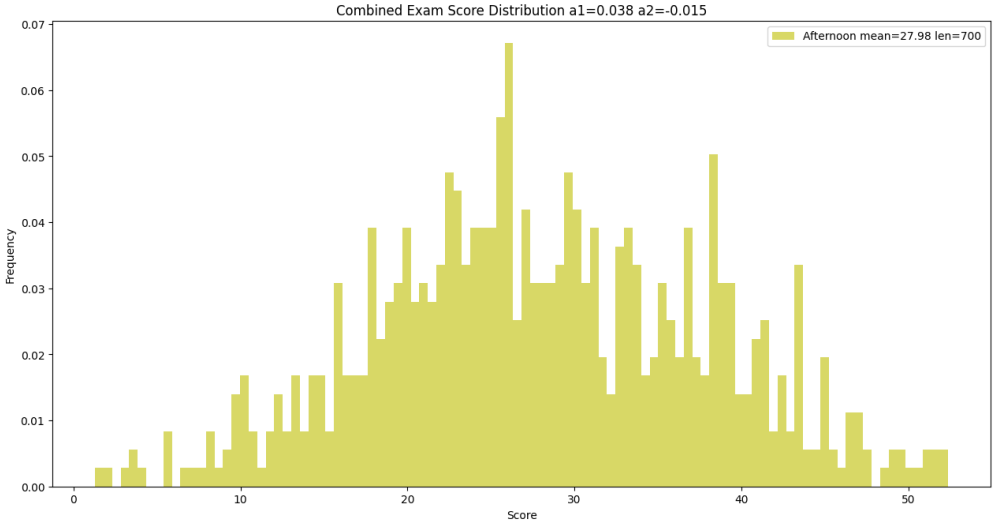

图 2　下午生成成绩直方图

2. 分位数调整及 GMM 分布调整

使用分位数调整及 GMM 模型调整后的分布情况（图 3）：

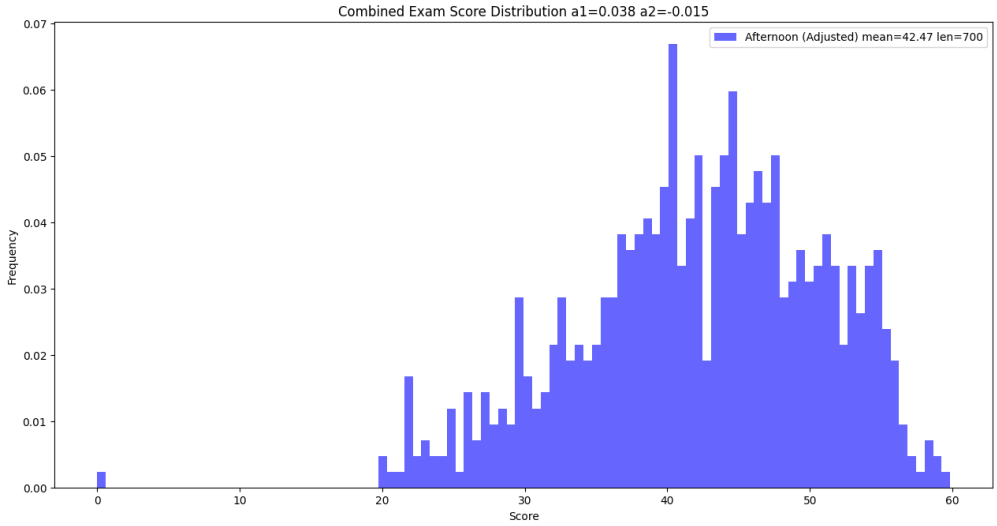

图 3　下午成绩调整后成绩直方图

调整前后的分布对比（图 4）：

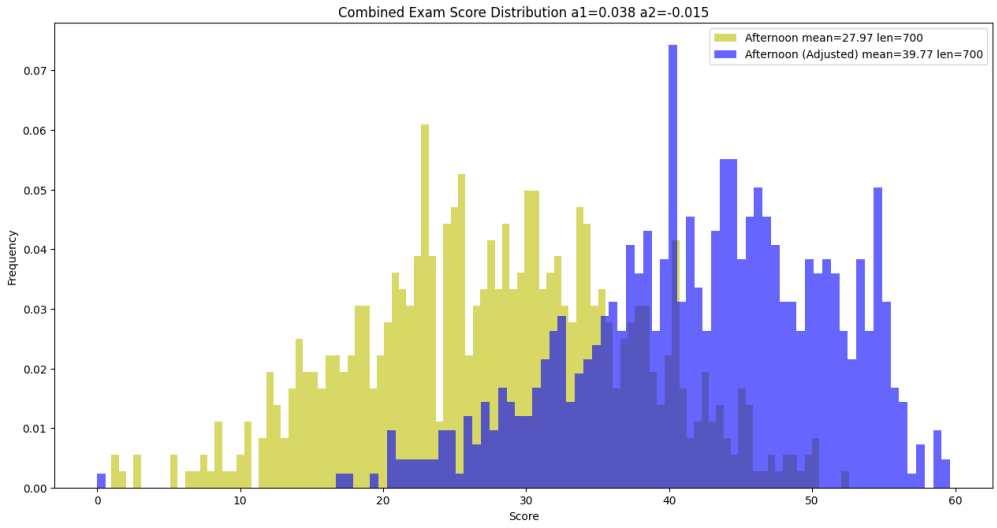

图 4　下午调整前后成绩直方图

3. 结果展示

调整后早上下午数据对比的直方图（图 5）：

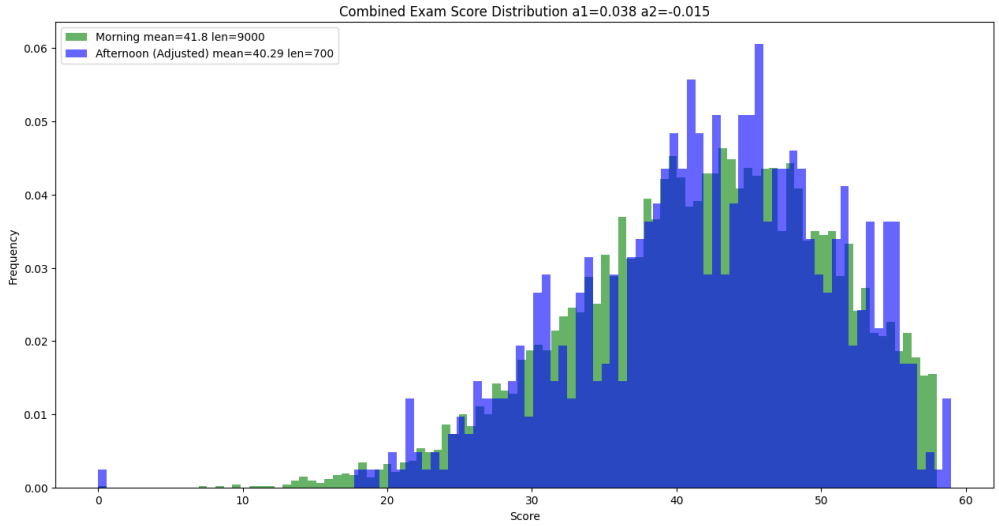

图 4　下午调整前后成绩直方图

可见该方案调整后的数据符合预期分布。

六、讨论

1. 模型评价

模型的优点

精确调整均值差距：本研究中的分位数调整和高斯混合模型有效地调整了考试成绩的均值差距，确保了公平性。

保持数据分布特征：通过使用高斯混合模型，调整后的数据仍然反映考生的真实表现，避免了数据失真。

模型的局限性

对极端数据的处理：模型在处理极端分数（非常高或非常低）时存在一定局限性，可能影响整体数据的均衡性。

数据生成的随机性影响：使用随机数和重采样技术会引入一定的随机性，可能导致结果的波动。

2. 方法改进

引入其他调整方法的可能性

机器学习和深度学习技术：使用回归模型或神经网络模型可以更精确地调整分数，提高调整的准确性。

贝叶斯统计方法：通过贝叶斯统计处理不确定性和极端数据，提高模型的稳定性。

提高数据生成和调整的精度

更精细的数据模拟：引入更多参数和更复杂的分布模型，生成更接近实际的模拟数据。
分位数调整的改进：结合其他分布调整方法，如平滑加权分布调整，提高调整的精度。
数据验证和优化：通过交叉验证方法验证调整后的数据分布和均值，进一步优化调整算法。

七、结论

研究的主要发现

通过本研究的数学建模方法，我们成功地调整了某地市英语听说考试的成绩，使得上午和下午两场考试的均分差距控制在合理范围内。这一过程包括以下关键步骤：

1. **数据生成与优化**：利用正态分布和偏态分布生成模拟数据，通过重采样确保数据的合理性和完整性。
2. **数据平滑处理**：应用核密度估计对数据进行平滑处理，使得生成的数据更接近真实分布。
3. **分布调整**：使用分位数调整方法，使下午考试的分布与上午考试的分布更为一致。
4. **高斯混合模型**：通过高斯混合模型进一步调整数据分布，确保调整后的数据在统计学上的合理性和一致性。

这些方法综合运用，确保了两场考试的成绩分布在各个百分位上的相似性，最终实现了考试成绩的公平调整。

研究的意义

本研究为解决考试成绩不均衡问题提供了一种科学有效的解决方案，其意义主要体现在以下几个方面：

1. **保障考试的公平性**：通过调整考试成绩，使得不同场次的考试在成绩分布上具有可比性，确保了考试结果的公平性。这对考试组织者和考生来说，都是一种公平公正的体现。
2. **提高考试的可信度**：通过科学的数学建模方法，对考试成绩进行合理调整，提升了考试结果的可信度。这样的调整方法可以应用于其他类似考试场景，具有广泛的适用性。
3. **为教育评估提供参考**：本研究的方法和结果为教育评估提供了参考，通过数学建模和数据分析，可以更科学地评估和调整考试成绩，提升教育评估的科学性和准确性。
4. **技术推广与应用**：本研究中的数学建模方法，特别是高斯混合模型和分位数调整方法，展示了在实际问题中的应用潜力。未来可以在其他领域推广这些技术，解决类似的分布不均衡问题。

八、总结

综上所述，本研究通过科学的数学建模方法，成功实现了对某地市英语听说考试成绩的公平调整，保障了考试的公平性和可信度。研究结果不仅为本次考试提供了有效解决方案，也为未来的教育评估和其他类似场景的应用提供了重要参考。通过推广和应用这些技术方法，可以进一步提升教育和

评估领域的科学性和公平性。

九、参考文献

无

十、附录

1.论文中使用的代码，文件，均已上传至 https://github.com/Linxy330/Modelling

2. 算法实现的源代码：

```
import numpy as np  # 计算
import matplotlib.pyplot as plt  # 绘图
from scipy.stats import skewnorm, scoreatpercentile  # 统计&计算分位数
from sklearn.mixture import GaussianMixture  # 高斯混合模型
import random as ran  # 生成随机数


# 生成数据
def generate_data(a, loc, scale, size, lower, upper):
    data = skewnorm.rvs(a=a, loc=loc, scale=scale, size=size)
    data = data[(data >= lower) & (data <= upper)]  # 确保数据在范围内
    # 如果数据点不足，继续生成直到达到所需数量
    while len(data) < size:
        additional_data = skewnorm.rvs(a=a, loc=loc, scale=scale,
size=(size - len(data)))
        additional_data = additional_data[(additional_data >= lower) &
(additional_data <= upper)]
        data = np.concatenate((data, additional_data))
    return data[:size]


# 生成数据
a_random_1 = round(ran.uniform(-0.1, 0.1), 3)  # a 值随机
a_random_2 = round(ran.uniform(-0.1, 0.1), 3)
morning_data = generate_data(a=a_random_1, loc=42, scale=10, size=9000,
lower=0, upper=58)
afternoon_data = generate_data(a=a_random_2, loc=28, scale=10, size=700,
lower=0, upper=53)
```

```
# 调整上下午平均数
def adjust_data(data, a, loc, scale, size, lower, upper):
    data = data[(data >= lower) & (data <= upper)]  # 确保数据在范围内
    # 如果数据点不足，继续生成直到达到所需数量
    while len(data) < size:
        additional_data = skewnorm.rvs(a=a, loc=loc, scale=scale,
size=(size - len(data)))
        additional_data = additional_data[(additional_data >= lower) &
(additional_data <= upper)]
        data = np.concatenate((data, additional_data))
    return data[:size]


# 计算平均值
morning_mean = np.mean(morning_data)
afternoon_mean = np.mean(afternoon_data)
morning_mean_diff = 42 - morning_mean
afternoon_mean_diff = 28 - afternoon_mean

while (abs(morning_mean_diff) > 0.2 or abs(afternoon_mean_diff) > 0.2):
    if abs(morning_mean_diff) > 0.2:
        morning_data += morning_mean_diff
        morning_data = adjust_data(data=morning_data, a=a_random_1,
loc=42, scale=10, size=9000, lower=0, upper=58)

    if abs(afternoon_mean_diff) > 0.2:
        afternoon_data += afternoon_mean_diff
        afternoon_data = adjust_data(data=afternoon_data, a=a_random_2,
loc=28, scale=10, size=700, lower=0, upper=53)

    morning_mean = np.mean(morning_data)
    afternoon_mean = np.mean(afternoon_data)
    morning_mean_diff = 42 - morning_mean
    afternoon_mean_diff = 28 - afternoon_mean

# 分位数调整
morning_sorted = np.sort(morning_data)
```

```
afternoon_sorted = np.sort(afternoon_data)

for percentile in np.linspace(0, 100, 101):
    morning_value = scoreatpercentile(morning_sorted, percentile)
    afternoon_value = scoreatpercentile(afternoon_sorted, percentile)
    adjustment = morning_value - afternoon_value
    afternoon_sorted[np.isclose(afternoon_sorted, afternoon_value,
atol=1e-6)] += adjustment

# 调整下午平均值，使其与上午平均值相差 2 以内
morning_mean = np.mean(morning_sorted)
afternoon_mean = np.mean(afternoon_sorted)
mean_diff = morning_mean - afternoon_mean

if abs(mean_diff) > 2:
    adjustment = mean_diff - 2 if mean_diff > 0 else mean_diff + 2
    addition_random = round(ran.uniform(0, 4), 2)
    afternoon_sorted += adjustment + addition_random

# 确保调整后的数据在正确范围内
afternoon_mean = np.mean(afternoon_sorted)
afternoon_sorted = adjust_data(data=afternoon_sorted, a=a_random_2,
loc=afternoon_mean, scale=10, size=700, lower=0,
                               upper=60)


# 创建高斯混合模型并进行数据拟合
def fit_gmm(data, n_components):
    # n_components 表示处理精度，covariance_type='full'表示使用全协方差矩阵，
random_state=42 设置随机种子（确保结果的可重复性）
    gmm = GaussianMixture(n_components=n_components,
covariance_type='full', random_state=42)
    gmm.fit(data.reshape(-1, 1))  # 高斯混合模型需要二维输入，将数据调整为二维
数组
    return gmm


# 调整下午数据的分布以匹配上午数据
```

```
def adjust_data_distribution(morning_data, afternoon_data,
n_components):
    morning_gmm = fit_gmm(morning_data, n_components)
    afternoon_gmm = fit_gmm(afternoon_data, n_components)

    morning_sampled, _ = morning_gmm.sample(len(afternoon_data))  # 从早上模型中采样，生成与下午数据长度相同的相似分布
    afternoon_adjusted = morning_sampled.flatten()  # 将采样的数据调整为一维数组

    afternoon_adjusted = np.clip(afternoon_adjusted,
np.min(afternoon_data),
                                 np.max(afternoon_data))  # 将调整后的数据限制在下午数据的最小值和最大值之间

    return morning_gmm, afternoon_gmm, afternoon_adjusted


morning_gmm, afternoon_gmm, afternoon_adjusted =
adjust_data_distribution(morning_sorted, afternoon_sorted,

n_components=5)

# 最低分为0
morning_sorted[0] = 0
afternoon_adjusted[0] = 0

# 合并上午数据和调整后的下午数据
combined_data = np.concatenate((morning_sorted, afternoon_adjusted))

# 统计平均值
morning_mean = round(np.mean(morning_sorted), 2)
afternoon_origin_mean = round(np.mean(afternoon_data), 2)
afternoon_mean = round(np.mean(afternoon_sorted), 2)
combined_mean = round(np.mean(combined_data), 2)
# 统计人数
morning_len = len(morning_sorted)
afternoon_origin_len = len(afternoon_data)
```

```
afternoon_len = len(afternoon_sorted)
combined_len = len(combined_data)

# 绘制
plt.hist(morning_sorted, bins=100, density=True, alpha=0.6, color='g',
         label='Morning mean=' + str(morning_mean) + ' len=' +
str(morning_len))
plt.hist(afternoon_data, bins=100, density=True, alpha=0.6, color='y',
         label='Afternoon mean=' + str(afternoon_origin_mean) + ' len='
+ str(afternoon_origin_len))
plt.hist(afternoon_adjusted, bins=100, density=True, alpha=0.6,
color='b',
         label='Afternoon (Adjusted) mean=' + str(afternoon_mean) + '
len=' + str(afternoon_len))
plt.hist(combined_data, bins=100, density=True, alpha=0.6, color='r',
         label='Combined mean=' + str(combined_mean) + ' len=' +
str(combined_len))
plt.title('Combined Exam Score Distribution a1=' + str(a_random_1) + '
a2=' + str(a_random_2))
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```