多模型融合与高龄段经验外推驱动的死亡率预测及 MGMA 产品主导的养老年金长寿风险动态管控研究

团队成员 1：郑皓月

华东师范大学

团队成员 2：余乐扬

重庆大学

团队成员 3：范洪玮

华东师范大学

指导老师：钱林义

华东师范大学

摘　要

在全球人口老龄化持续加剧、平均寿命不断延长的背景下，长寿风险已成为养老金制度与年金产品面临的核心挑战之一。本研究基于2010-2021年保险人口死亡率数据，融合国际权威人类死亡率数据库（Human Mortality Database, HMD）所涵盖的多国人口死亡率信息，构建了系统化的死亡率建模与预测体系以及养老年金长寿风险评估框架，并在此基础上提出了具有结构创新性与管理前瞻性的风险应对策略。

在**死亡率预测**方面，研究采用 **Lee-Carter 模型**与 **Gompertz-Makeham 模型**相结合的**混合建模**策略，通过**年龄分层**与**动态加权**，有效整合两种模型在年龄趋势刻画与衰老机制建模方面的优势，提升了死亡率预测的稳健性与精度。在模型预测结果基础上，使用 **Whittaker-Henderson 等平滑算法**进行修匀处理，保证了死亡率曲线的平滑性与生物学合理性。针对超高龄段样本稀疏导致的外推误差问题，进一步引入 HMD 中多个死亡率结构相似国家的**高质量生命表数据作为外部锚定**，对 80 岁以上**高龄段死亡率**进行**修正外推**，显著提升了模型在极高龄区间的预测合理性与稳定性，构建出一条平滑、连续、符合生物学逻辑的 0-120 岁保险人口死亡率预测曲线。

在**长期趋势建模**中，综合考虑**人口统计学队列效应**、**医疗进步边际收益递减**与**寿命渐进延长假设**，延展性地构建了 2025-2125 年保险人口死亡率演化路径。结果显示，保险人口整体死亡率将稳步下降，但改善幅度逐步减缓；女性寿命持续领先于男性，长寿风险在未来一百年内呈现稳中升高的长期趋势。

在**长寿风险量化**方面，使用**死亡率改善场景下的年金现值差法**构建了**长寿风险评估模型**，通过对不同性别、贴现率及投保年龄情景下的风险敞口与相对风险率进行测算，发现当前静态定价假设存在系统性低估趋势，特别是在男性、低贴现率及低龄投保人群中表现尤为显著。

在**应对策略**方面，提出**最低保证死亡率年金（MGMA）**作为核心创新产品结构，通过引入**与死亡率挂钩**的浮动因子、设置**最低年金支付保障**，在控制尾部责任增长的同时保障客户基本利益。实证显示，该结构在寿命持续改善情景下具有显著的**风险缓释效果**，是对传统固定年金产品的重要改进。此外，研究还系统构建了**多层次风险管理框架**：从投资端引入资产久期匹配与负债驱动投资策略，增强资产配置对未来现金流波动的适应能力；从风险转移维度提出寿命再保险机制、长寿衍生品工具与死亡率互助结构设计；从制度视角提出监管协调与 ESG 责任导向的资本配置建议，实现产品—精算—投资—监管的闭环联动。

综上，本文构建了从经验数据出发，贯穿建模、预测、量化到全维度风险应对路径设计的完整研究闭环，在死亡率外推机制、模型融合策略及结构性养老年金设计等方面具有一定创新性与实际应用价值，可为保险公司在长寿风险日益凸显的背景下实现稳健经营与产品优化提供理论支持与技术路径。

关键词：长寿风险；死亡率预测；Lee-Carter 与 Gompertz-Makeham 动态混合模型；MGMA 产品；风险量化；人口统计学；衰老生物学

目 录

图目录

表目录

一、引言

（一）研究背景与现实动因

21 世纪以来，人口老龄化已成为全球结构性挑战之一。伴随着生育率持续走低和医疗卫生条件显著改善，人类平均寿命呈现持续增长态势。根据联合国《2024 年世界人口展望：结果摘要》报告，预计到 2054 年，全球人均预期寿命将达到 77.4 岁；在 2080 年前，65 岁及以上年龄段人口数量将超过 18 岁及以下年龄段人口数量。中国作为全球人口最多的国家之一，也正加速迈入“深度老龄化”社会。民政部数据显示，截至 2024 年年底，我国 60 岁及以上老年人口达到 3.1 亿，占总人口的 22%。根据预测，到 2035 年左右，我国 60 岁及以上老年人口将突破 4 亿，在总人口中的占比将超过 30%，进入重度老龄化阶段。随着寿命延长成为不可逆的趋势，传统的社会保障体系、养老制度与商业保险产品均面临深远影响。

在此背景下，长寿风险（Longevity Risk）这一概念受到社会各界前所未有的关注。长寿风险是指由于人群实际寿命超过预期寿命而导致的财务责任增加所产生的风险，表现为养老金、年金等长期支付责任的持续期被显著拉长，进而对个人财务、保险公司偿付能力及国家财政可持续性形成系统性冲击。特别是对于年金保险类产品而言，投保人存活时间的每一年延长，都会叠加形成责任现值的非线性增加，从而放大保险公司的尾部风险暴露。

（二）研究价值与核心问题

尽管长寿趋势已成为不可忽视的现实，但死亡率预测作为寿险产品定价与风险控制的基础工具，其建模仍面临诸多挑战。一方面，人口死亡率在不同性别、年龄、时间和人群类型之间存在显著异质性，需精细建模才能精准捕捉其内在结构；另一方面，超高龄人群（尤其是 90 岁以上）样本稀缺、数据波动性强，使得死亡率的外推及长期趋势判断存在高度不确定性。此外，极端事件（如新冠疫情）、医疗技术突破（如肿瘤免疫疗法）和健康行为变化（如吸烟率下降）等因素的交织影响，也对死亡率的动态预测构成挑战。

因此，准确识别死亡率的演化规律与群体差异，构建合理、稳健且具有可解释性的预测模型，并据此量化养老年金的长寿风险敞口，对于完善保险公司精算定价体系、优化产品结构、制定投资策略及推动养老金融创新具有重要的理论意义与现实价值。

（三）研究思路与内容安排

本研究围绕长寿风险的识别、建模、预测与应对展开，基于大赛提供的 2010-2021 年保

险业经验死亡率数据，辅以人类死亡率数据库（Human Mortality Database, HMD）中多国人口死亡率历史数据，在建模逻辑、预测策略与应用场景方面进行了系统性设计。主要研究思路如下：

1. 死亡率水平与趋势差异分析：研究保险人口与一般人口在不同性别、年龄段、时间维度上的死亡率水平与变化趋势，识别保险人群的特征性差异，并结合社会环境、医疗资源、生活方式等因素解释形成机制；

2. 2025 年保险人口死亡率预测与曲线外推：构建以 Lee-Carter 模型与 Gompertz-Makeham 模型为核心的混合建模体系，对保险人口 2025 年分性别、分年龄的死亡率进行预测，采用平滑与生物逻辑约束方法修匀死亡率曲线，并外推至 120 岁极高龄区间；

3. 未来百年保险人口死亡率趋势展望：在前期预测结果的基础上，结合死亡率改善路径、生物学假设、人口队列效应与医疗变革等因素，构建 2025-2125 年的长期死亡率预测模型，评估寿命延长趋势及其节奏特征；

4. 养老年金长寿风险管控建议：通过年金现值对比法量化静态定价与动态预测之间的责任敞口，识别不同性别、贴现率与投保年龄组合下的长寿风险程度，创新设计最低保证死亡率年金产品，并在此基础上综合投资管理、风险转移、监管协调等方面提出风险应对策略。

（四）预期贡献与应用前景

本研究拟在理论层面完善长寿风险建模方法，在实践层面为养老年金产品的精算设计与风险控制提供数据支撑与决策依据。具体贡献包括以下四个方面：

第一，构建结合静态保险数据与动态人口数据的死亡率预测框架，增强预测模型的适应性与稳定性；第二，系统评估保险人口的长期死亡率趋势与长寿风险暴露，为保险公司制定准备金政策与资本评估提供量化依据；第三，提出具有操作性的产品与风险综合管理优化策略，有助于增强养老年金产品及保险公司的抗风险能力与市场竞争力；第四，为保险监管机构与养老金政策制定者提供参考，促进养老金融产品的可持续发展。

当前，死亡率变化趋势与长寿风险管理已构成养老保障体系建设的关键议题。本研究旨在通过数据驱动的建模与实证分析，深入理解保险人口的生命规律与风险暴露特征，推动寿险行业向更加精准、稳健与可持续的方向发展。

二、不同人群死亡率差异分析

（一）数据来源与建模方法

本研究的数据来源主要包括两部分：一是由大赛组委会提供的保险行业死亡率数据，二是研究组从人类死亡数据库（Human Mortality Database, HMD）中检索的国际人口死亡率数据。针对大赛提供的数据，本文采用了 Gompertz-Makeham 模型进行拟合与预测；而对于 HMD 数据，分别应用 Gompertz-Makeham 模型与 Lee-Carter 模型进行建模和比较分析，以增强模型预测的稳健性与可靠性。

（二）保险业死亡率数据特征与趋势分析

基于大赛提供的 2010-2021 年期间的保险业死亡率数据进行可视化分析（如图 1-图 4 所示），研究发现如下主要特征和变化趋势。

1. 年龄结构特征

死亡率整体随年龄增长而上升，呈现指数型增长趋势。其中，老年群体的死亡风险显著高于中青年群体，符合人口生命周期规律。

2. 性别差异特征

男性在大多数年龄段的死亡率均高于女性，尤其在中老年阶段差异更为明显。此外，男性死亡率的年际波动幅度相较女性更大，表明男性群体的死亡率受外部风险因素影响较强。

3. 时间演化特征

从时间维度来看，2010 年至 2016 年，死亡率呈现缓慢上升趋势；2016 年后整体趋于平稳，有轻微波动但变化不大，说明死亡率水平在近年已进入相对稳定期。

4. 分年龄段特征

按照年龄段划分，婴幼儿群体（0-10 岁）的死亡率相对较高，可能受先天性疾病或基础卫生条件影响；青少年群体（10-20 岁）的死亡率维持在较低且平稳的水平；中年群体（21-60 岁）的死亡率呈轻微上升趋势；老年群体（61 岁及以上）的死亡率则显著上升，是整体死亡风险的主要贡献来源。

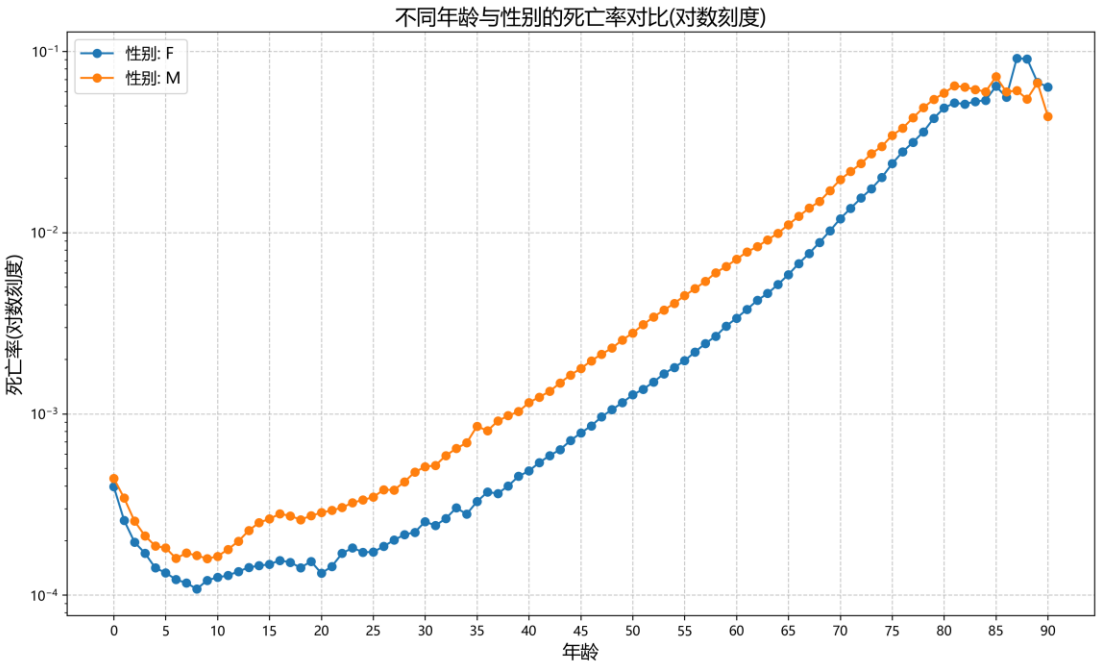

图 1　对数刻度下不同年龄与性别保险人群死亡率对比图

（M 性：男性；F 性：女性）

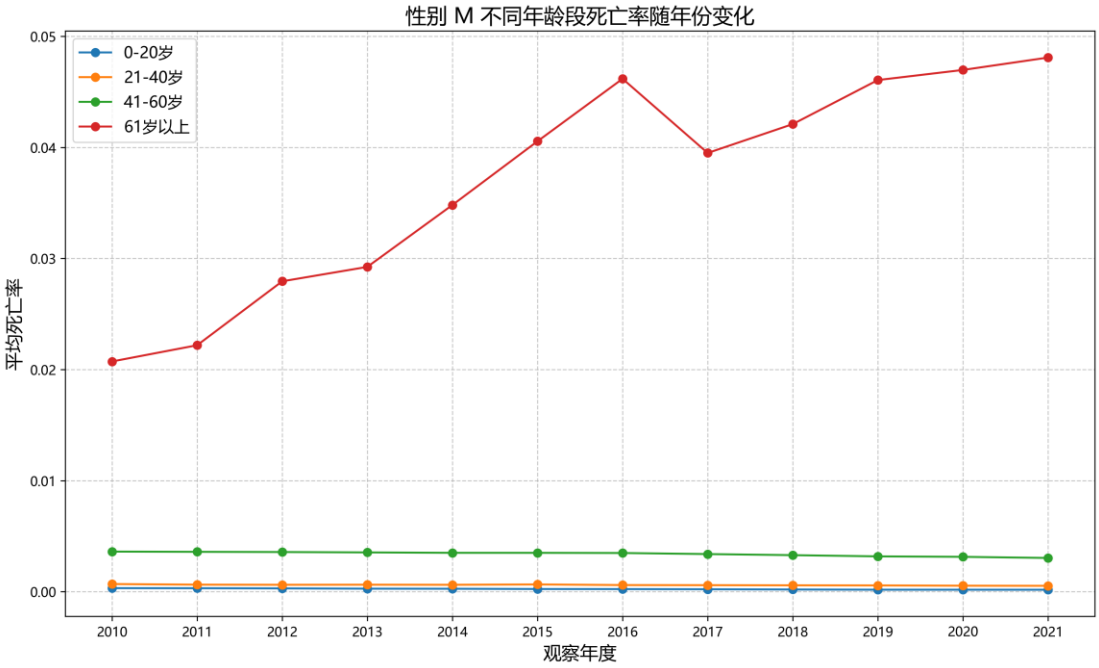

图 2　男性保险群体不同年龄段死亡率随时间变化趋势图

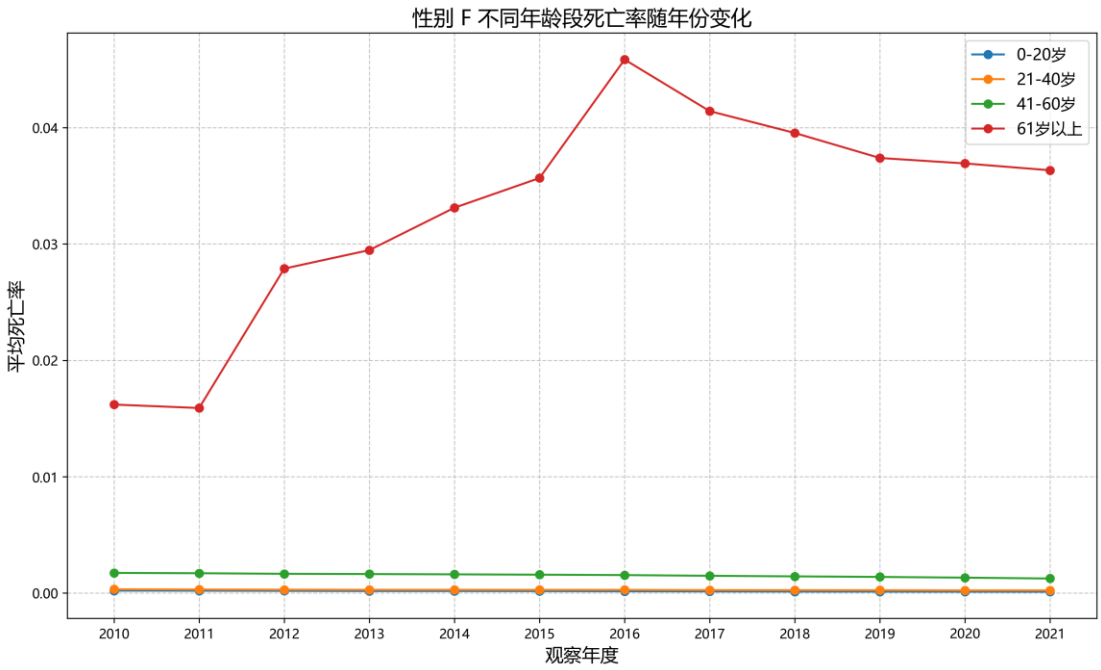

图 3　女性保险群体不同年龄段死亡率随时间变化趋势图

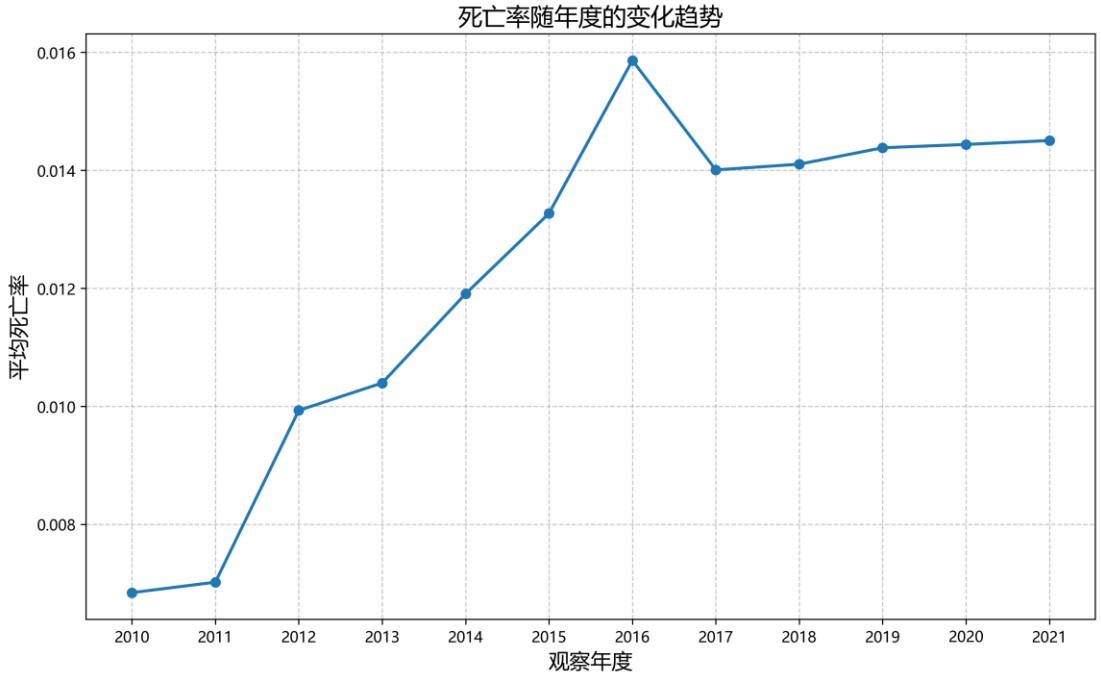

图 4　保险业死亡率随时间变化趋势图

（三）死亡率变化的驱动因素分析

对上述死亡率趋势及性别、年龄差异的形成机制进行归因分析，可能受到多种生理、行为、社会及环境因素的共同驱动，主要包括以下几个方面。

1. 生理差异与生活方式的性别影响

男性与女性在生理结构与基因表达上存在本质差异，女性在免疫系统功能、激素水平等方面通常表现出更强的生存优势，这可能部分解释其整体死亡率较低、寿命更长的原因。

此外，不同性别在生活方式与风险行为上也存在一定差异。男性可能更倾向于吸烟、饮酒、高脂饮食以及从事高风险职业，且在交通事故、暴力事件、职业伤害等中的暴露程度普遍高于女性。

这些因素共同作用，使得男性在中老年阶段积累了更高的健康风险，反映为死亡率持续高于女性。

2. 医疗进步与老龄化趋势的交叉影响

在婴幼儿阶段（0-10 岁），尽管死亡率仍相对较高，但随着医疗技术进步、疫苗普及、产前产后保健和新生儿急救能力等的不断提升，该年龄段死亡率已大幅下降，并在 10 岁后趋于稳定。青少年阶段受慢性病侵袭较少，死亡率维持在较低水平。

相较之下，老年群体的死亡率则受到人口结构性变化的显著影响。随着老龄化程度加剧，老年人口在总人口中的比重不断上升，尤其是慢性非传染性疾病（如心血管疾病、癌症、糖尿病）成为主要死亡原因，加剧了高龄群体的死亡风险累积，导致死亡率在高龄段呈陡峭上升趋势，对整体人口死亡率水平构成上行压力。

3. 外部环境与突发事件的干扰作用

2014-2016 年间死亡率的明显上升，可能源于多个外部扰动因素的叠加。一方面，随着老年人口的快速增长，本身已推高该年龄段的死亡率水平；另一方面，可能受到特定年份内公共卫生事件、环境污染、自然灾害等不可控因素的影响。

同时，社会节奏加快、职场压力上升、居民心理健康问题加剧等，也可能通过不良生活方式（如熬夜、酗酒、抑郁等）间接推高死亡风险。这些外部环境变化虽非长期趋势，但会在特定阶段内造成数据的异常波动。

（四）基于 HMD 数据的对比与趋势验证

为增强实证分析的广度与稳定性，研究组还使用 HMD 中提供的跨国历史人口死亡率数据进行建模与验证。分析结果在性别、年龄和时间维度上与大赛数据基本一致，进一步印证了前述趋势判断。鉴于 HMD 数据覆盖国家众多，此处仅选取在人口结构、医疗体系与养老制度方面具有代表性的四个国家——日本、澳大利亚、美国和德国的死亡率水平及演化趋势

进行展示，如图 5-图 7 所示。

1. 性别与年龄结构特征

HMD 数据同样表明，在性别方面，男性死亡率显著高于女性；按年龄分层分析，婴幼儿阶段的死亡率偏高，青少年死亡率处于最低水平，而老年阶段的死亡率呈现快速上升趋势。

2. 时间序列演化趋势

在研究所覆盖的时间区间内，死亡率整体呈现缓慢下降趋势，反映出医疗卫生水平的持续提升。但各年龄组下降速度存在一定差异，婴幼儿与青少年群体的死亡率下降幅度更加显著，老年阶段的下降速度相对较慢，且近年趋于平稳甚至轻微回升，表明老龄化风险已成为制约死亡率进一步下降的关键因素。

此外，近年死亡率下降速度有所放缓，可能与慢性疾病高发、医养负担增加及养老保障体系滞后等结构性问题相关。该现象从长期看可能对养老年金类产品带来较大支付压力。

3. 模型应用与比较分析

由于各个国家或地区在经济发展、医疗水平、风俗习惯等方面不尽相同，导致其死亡率可能存在差异。本研究基于 HMD 数据分别构建了 Gompertz-Makeham 模型与 Lee-Carter 模型，对不同性别和年龄人群的死亡率进行拟合与预测。对比发现，在多数国家与时间序列样本中，Lee-Carter 模型在中长期预测精度上优于 Gompertz-Makeham 模型。该结论为本研究后续长期死亡率的预测提供了一定的建模依据。

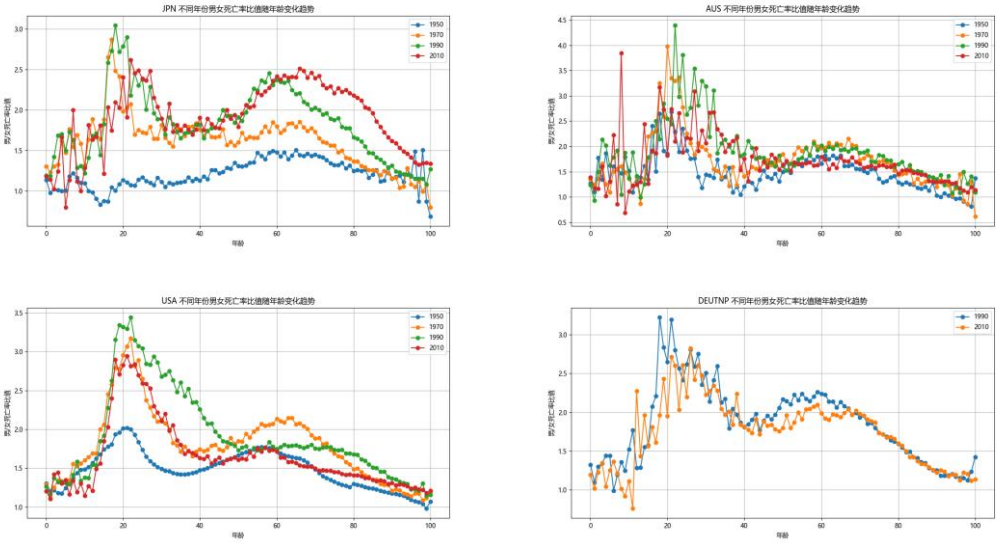

图 5　日、澳、美、德四国不同年份男女死亡率比值随年龄变化趋势图

（左上：日本；右上：澳大利亚；左下：美国；右下：德国）

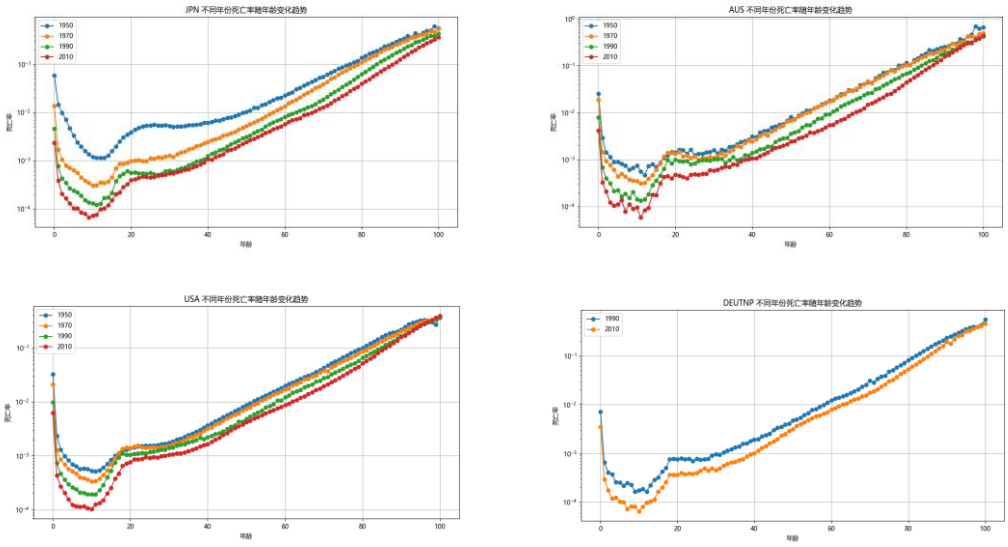

图 6　日、澳、美、德四国不同年份死亡率随年龄变化趋势图

（左上：日本；右上：澳大利亚；左下：美国；右下：德国）

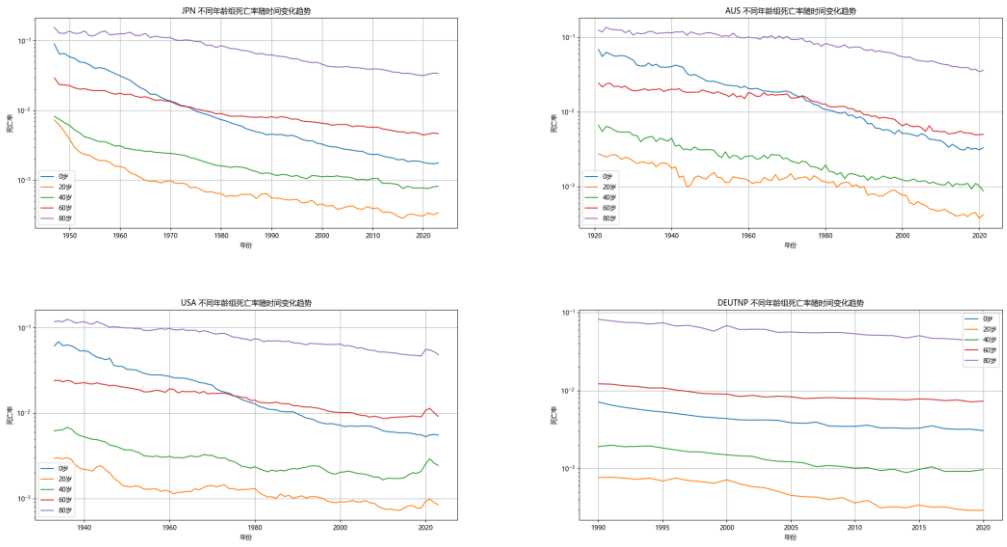

图 7　日、澳、美、德四国不同年龄组死亡率随时间变化趋势图

（左上：日本；右上：澳大利亚；左下：美国；右下：德国）

三、保险人口 2025 年死亡率预测分析

为评估未来养老年金产品中高龄人群的死亡率水平及其对责任现值和长寿风险的影响，本章基于大赛提供的 2010-2021 年保险人口死亡率数据，对 2025 年不同性别、各年龄点的死亡率展开预测分析。

（一）初步模型构建与评估

1. 初步建模思路

本节目标为预测 2025 年保险人口的死亡率，分别按性别与年龄进行建模，预测年龄范围为 0 岁至 120 岁。数据来源为大赛提供的 2010-2021 年间某保险产品的死亡率数据，共计 2184 条记录（男性 1092 条，女性 1092 条），涵盖 0 至 90 岁年龄段。

Lee-Carter 模型与 Gompertz-Makeham 模型是当前学术界和业界常用的死亡率模型。

Lee-Carter 模型形式为：

$$\ln m(x,t) = a(x) + b(x)k(t) \quad (1)$$

用于捕捉时间序列中的系统性趋势变化，是国际标准方法，具有较为成熟的理论。

Gompertz-Makeham 模型形式为：

$$\mu(x) = A + B \cdot \exp(Cx) \quad (2)$$

适用于建模成年人死亡率的年龄模式，具有生物学合理性和良好的外推能力。

为充分捕捉死亡率的年龄结构特征与时间演化趋势，在初步建模中，研究组选择同时使用上述两个模型（而非简单使用一种模型），采用 **Lee-Carter 模型与 Gompertz-Makeham 模型的年龄加权组合策略**，以综合发挥两种模型的预测潜力，并在此基础上进行死亡率曲线修匀。

（1）年龄分层动态加权策略

为充分发挥 Lee-Carter 模型与 Gompertz-Makeham 模型在不同年龄段的预测优势，研究组在建模中采用以下动态权重分配策略：

- **青年段（≤25 岁）**

考虑到 Lee-Carter 模型对年轻群体死亡率模式具有更优的拟合能力，在青年段以该模型作为主导，设置权重为 90%。

- **中年过渡段（26-65 岁）**

采用平滑过渡权重策略：在 26-40 岁，设置 Gompertz-Makeham 权重从 10%线性递增至 40%；在 41-65 岁，设置 Gompertz-Makeham 模型权重从 40%线性递增至 70%。

- **老年段（≥65 岁）**

考虑到 Gompertz-Makeham 模型的理论框架更符合衰老生物学机制，在老年段以该模型作为主导，设置权重为 70%。

这种“年龄分层+动态权重”的混合策略机制的优势在于通过 Lee-Carter 模型捕捉人口

特异性模式、通过 Gompertz-Makeham 模型描述生物衰老规律，在权重分配下实现了**经验数据与理论法则的最佳平衡**，有效提升了拟合稳定性与预测精度。

（2）后处理优化

在将两种死亡率模型融合后，研究组通过 **Whittaker-Henderson 平滑算法**对曲线进行修匀，消除了组合过程可能产生的伪影，**确保死亡率曲线的连续性和单调性**。

2. 模型参数与预测结果

（1）模型参数

依据上述方法，使用具体数据进行初步建模预测，得到模型主要参数如表 1、表 2 所示。图 8 展示了 2010-2021 年保险业死亡率数据随时间变化的趋势，以及 Lee-Carter 模型$b(x)$参数随年龄x的变化情况。

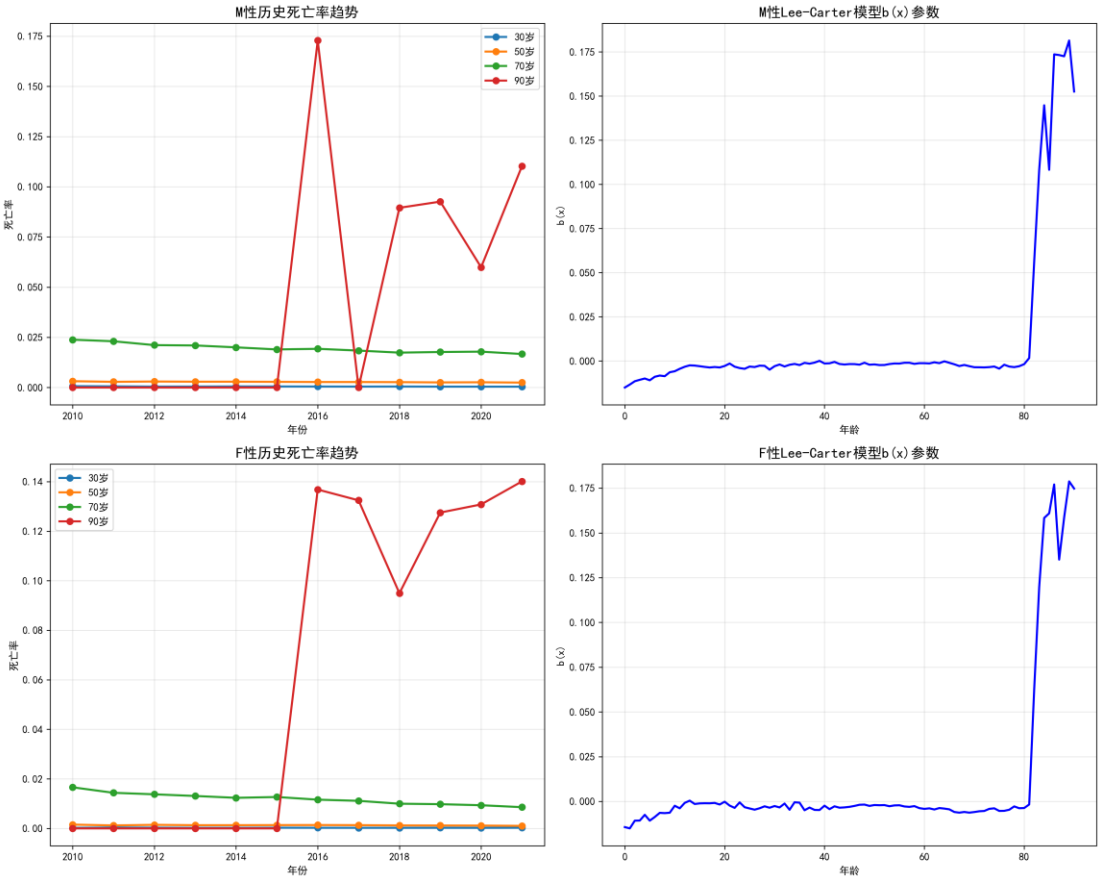

图 8　分性别历史死亡率趋势与初步模型中 Lee-Carter 模型$b(x)$参数图

（M 性：男性；F 性：女性）

表 1　2025 年保险人口死亡率初步预测模型中 Lee-Carter 模型主要参数

性别	$k(t)$趋势斜率	预测$k(2025)$
男性	3.467872	43.543343
女性	3.341246	43.434078

表 2　2025 年保险人口死亡率初步预测模型中 Gompertz-Makeham 模型参数

性别	参数A（基础死亡率）	参数B（年龄相关基数）	参数C（衰老速率）	模型R^2
男性	0.000000	0.000017	0.100841	0.9638
女性	0.000021	0.000001	0.131474	0.9981

（2）预测结果

初步模型下 2025 年保险人口死亡率预测结果如图 9、图 10 所示。表 3 汇总了在初步模型预测下 2025 年不同性别保险人口在关键年龄点的死亡率预测值。可见，预测值随年龄增长呈显著上升趋势，老年群体的死亡风险远高于中青年群体。整体来看，得益于混合模型策略的使用，预测结果在 80 岁前表现优秀；但在 80 岁以后的超高龄段，死亡率出现快速升高，预测效果较差。因此，需要对初步构建的预测模型进行评估，并在此基础上进一步优化和改进。

表 3　初步模型下 2025 年保险人口关键年龄点死亡率预测值

	30 岁	50 岁	65 岁	80 岁	100 岁	120 岁
男性	0.000417	0.002543	0.011077	0.075763	1.002729	1.107911
女性	0.000182	0.000922	0.004918	0.059897	1.432932	1.583240

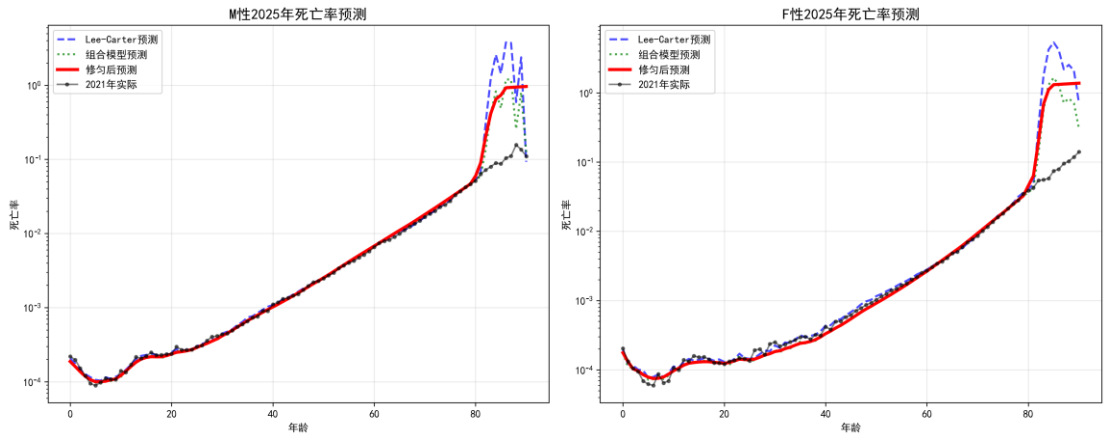

图 9　初步模型下 2025 年保险人口死亡率预测结果图

（M 性：男性；F 性：女性）

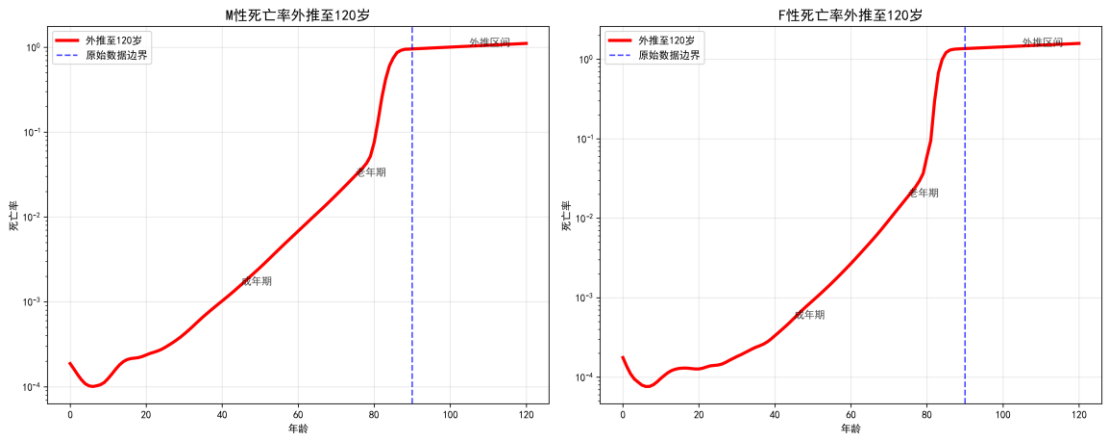

图 10　初步模型下 2025 年保险人口死亡率外推至 120 岁预测结果图

（M 性：男性；F 性：女性）

3. 模型评估与合理性验证

（1）验证集效果

在验证集上对所构建的初步预测模型进行性能评估，结果如图 11、图 12 所示。具体来说，混合模型中，Lee-Carter 模型的拟合优度R^2约为 0.6，表明其在拟合历史死亡率数据时表现一般，存在一定的局限性；Gompertz-Makeham 模型的拟合优度R^2接近 1，显示出极强的拟合能力和良好的拟合效果，能够较好地反映死亡率随年龄增长的趋势。总体来看，整体模型在预测死亡率方面表现较好，但由于 Lee-Carter 模型的拟合效果一般，提示**模型在某些年龄段或时间维度上仍有改进空间**。

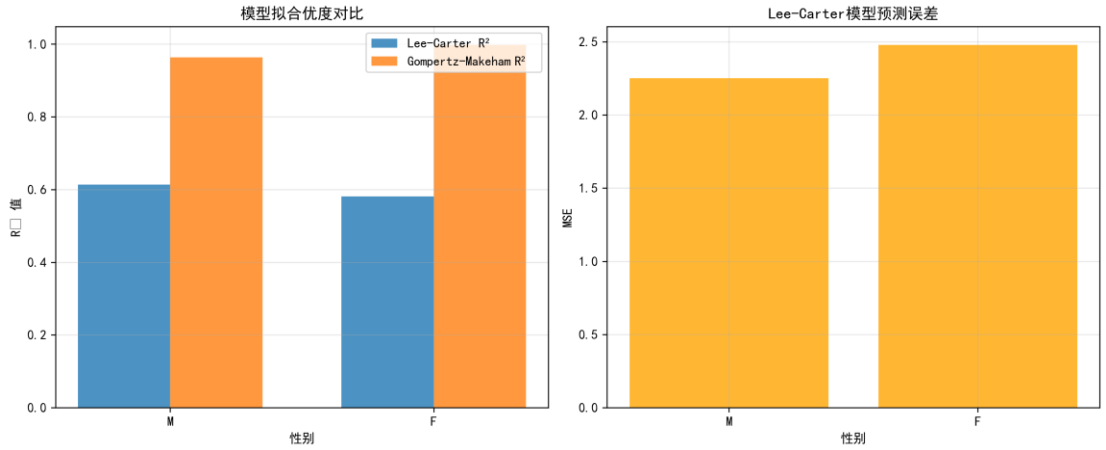

图 11　2025 年保险人口死亡率初步预测模型拟合优度与测试误差分性别对比图

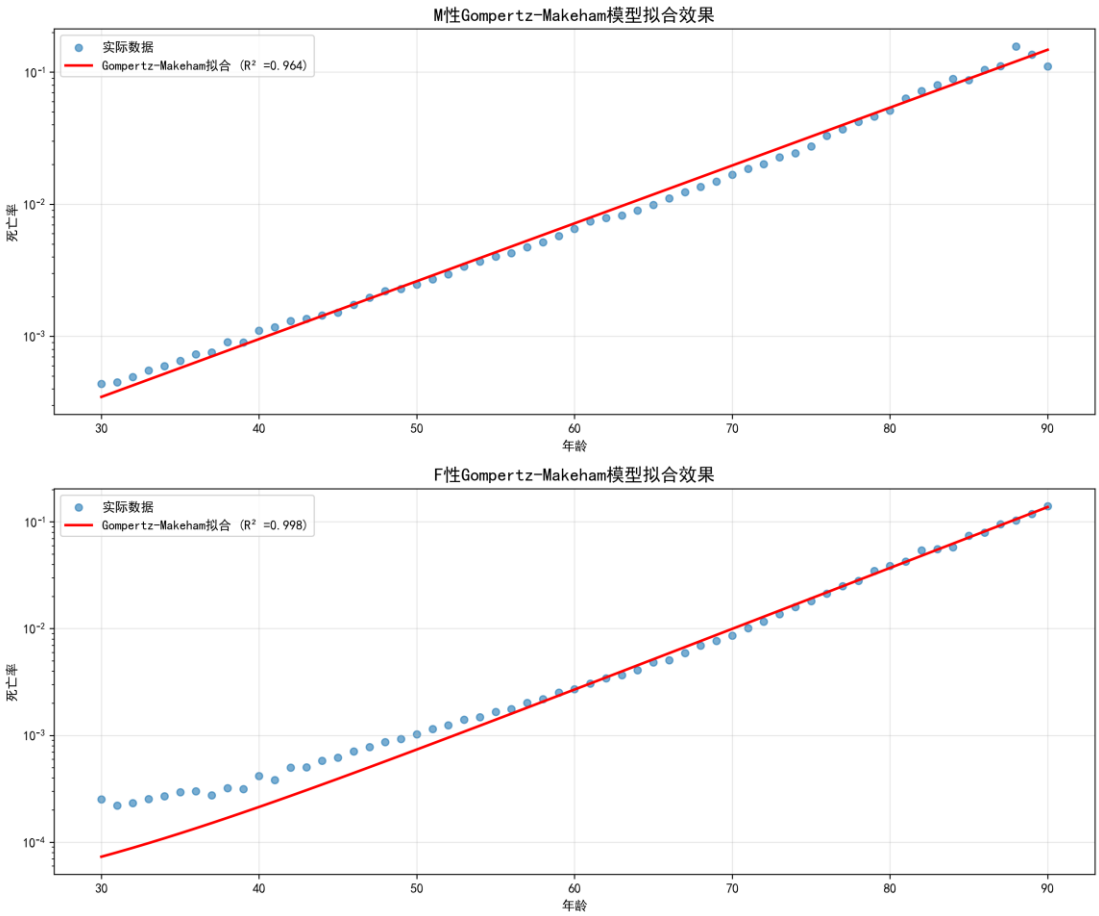

图 12　2025 年保险人口死亡率初步预测模型中 Gompertz-Makeham 模型拟合效果图

（2）生物学合理性检查

对初步模型预测结果的生物学合理性进行评估发现，男性和女性的死亡率预测曲线在 30 岁以上年龄均表现出 100%的单调性，高于良好阈值 95%，表明模型能够较好地捕捉人群随年龄增长而死亡率逐步上升的自然衰老趋势，具备较强的生物学解释力。然而，**模型预测的最大死亡率值均超过 1**，且出现在 120 岁这一极高年龄，男性最大死亡率预测值为 1.107911，女性最大死亡率预测值为 1.583240，明显违背了死亡率作为概率变量不应超过 1 的基本逻辑。这种异常可能源于模型在高龄阶段外推预测时出现的过度拟合或指数增长速度过快，尤其是在数据稀缺的高龄区间，模型缺乏足够约束，导致死亡率异常膨胀。因此，**需要在模型外推过程中加强合理性约束**，以提高预测结果的生物学有效性和现实可接受性。

（3）国际经验对比

通过与国际死亡率参考数据进行对比分析（图 13），同样发现所构建初步模型在高龄阶段的死亡率预测存在一定偏差，尤其在 85 岁及以上年龄段表现明显。如表 4 所示，65 岁时

男性预测死亡率与国际死亡率参考数据的比值为 0.74，表明拟合效果较为理想，女性则相对偏离，为 0.49。至 85 岁，男性预测死亡率达到国际参考数据的 9.3 倍，女性更为突出，达 20.21 倍。至 100 岁时，男性和女性的预测死亡率分别为国际参考的 3.34 倍和 5.73 倍，显著超出合理范围。此结果表明，初步模型虽然能够较好地反映中青年年龄段的死亡率趋势，但**在超高龄阶段存在系统性高估的风险**，可能系因极端年龄段样本数据不足或模型参数对该年龄段的外推存在局限性所致。为提高预测的准确性和科学性，**需要对超高龄区间进行专门调整或引入外部修正机制**。

表 4　初步模型下 2025 年保险人口死亡率预测值与国际参考数据比值

年龄	男性预测值与国际参考之比	女性预测值与国际参考之比
65 岁	0.74（合理）	0.49（需关注）
85 岁	9.30（需关注）	20.21（需关注）
100 岁	3.34（需关注）	5.73（需关注）

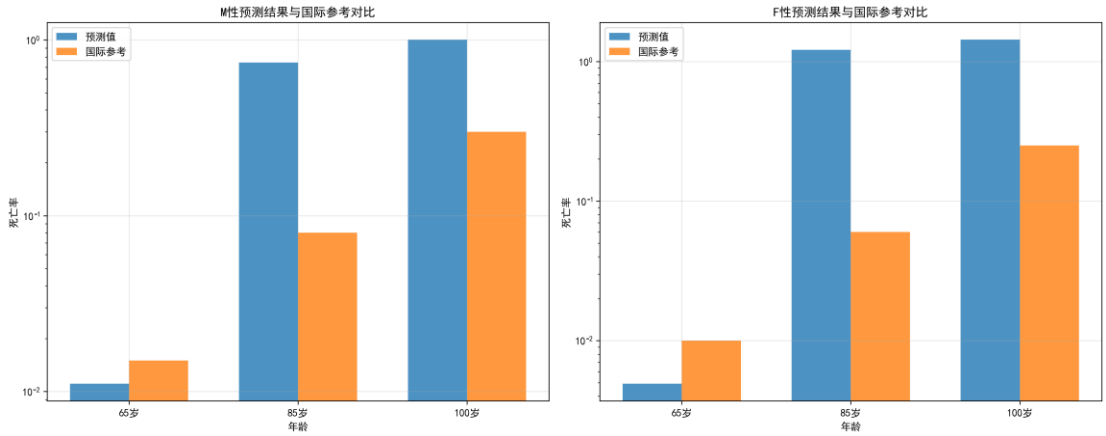

图 13　2025 年保险人口死亡率初步预测模型预测结果与国际参考数据对比图

（4）残差分析与正态性检验

对初步预测模型进行残差分析与正态性检验，结果如图 14、表 5 所示。容易发现，模型残差分布明显偏离正态性，尤其表现为**左偏重尾分布**，这通常意味着预测值系统性高估，且误差在个别年龄段（特别是高龄阶段）出现极端波动。

上述统计特征与此前高龄段死亡率预测显著偏高的结果相一致，进一步印证了模型在极端年龄区间存在外推误差或拟合不足的问题，可以考虑在保留现有模型结构基础上，结合高龄段的预测误差特征进行二次拟合修正，以提高残差分布的正态性和整体预测的稳定性。

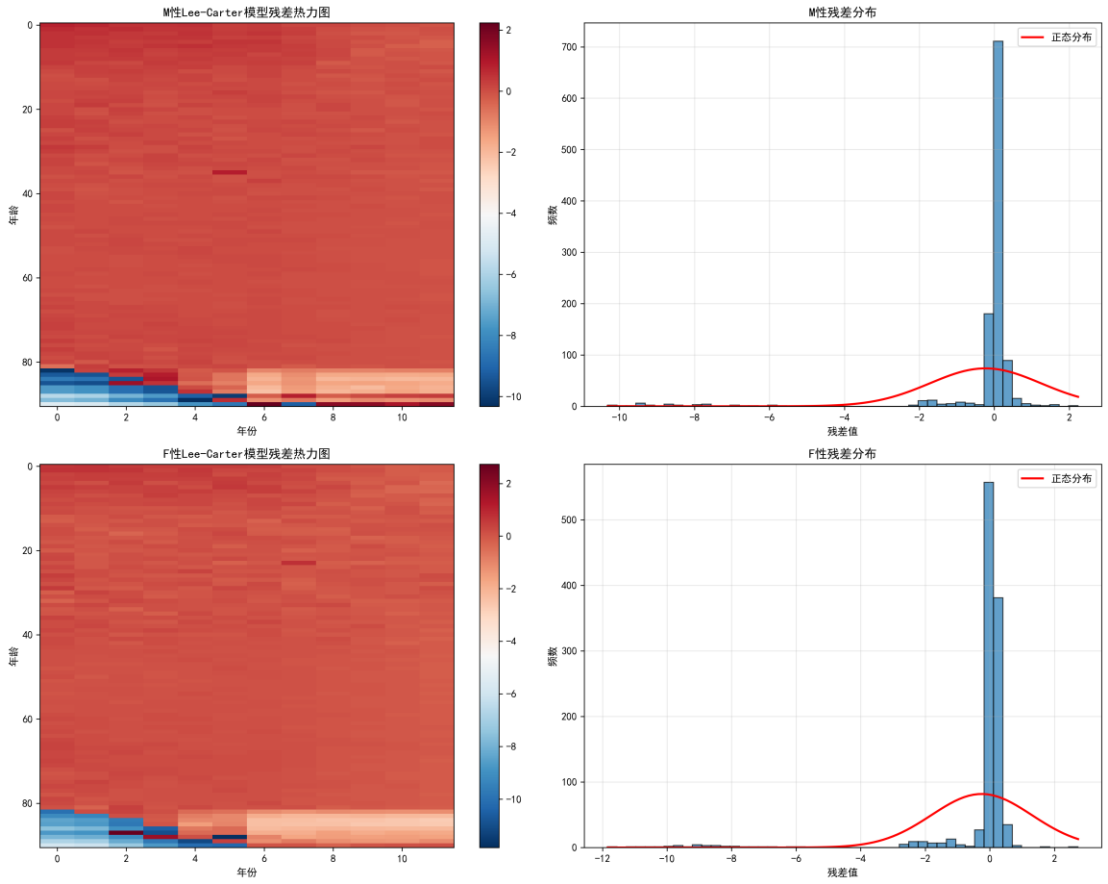

图 14　2025 年保险人口死亡率初步预测模型残差分析图

表 5　2025 年保险人口死亡率初步预测模型残差统计结果

性别	均值	标准差	偏度	峰度	正态性检验 P 值
男性	-0.236294	1.481890	-5.1524	26.8038	0.0000（非正态）
女性	-0.253157	1.554209	-5.1901	27.6397	0.0000（非正态）

（5）模型质量总结

从整体预测效果来看，初步构建的死亡率预测模型在 80 岁以下人群中表现出良好的拟合精度与趋势一致性，能够有效捕捉中青年阶段的死亡率变化规律。然而，在 80 岁及以上的高龄阶段，模型预测结果出现明显偏差，表现为系统性高估，这一问题在生物学合理性检查和国际经验对比中尤为突出。此外，残差分析与正态性检验进一步揭示出模型在高龄区间的拟合不足，残差分布偏离正态，存在较强的左偏和尖峰现象。

综上所述，**当前模型在中青年年龄段具有较强的实用性，但在超高龄阶段尚需优化调整**。整体模型质量评价为：需改进。

4. 模型问题与改进方向

（1）风险提示

当前死亡率预测模型的应用尚面临一系列潜在风险与挑战，在对模型进行改进时需予以重视：

- 保险人口选择效应可能导致样本群体与真实风险群体之间存在偏差，从而影响预测的代表性与准确性；
- 医疗技术进步及健康水平提升具有持续性，可能长期拉低死亡率趋势，进而影响历史数据对未来的预测力；
- 将死亡率外推至 120 岁具有高度不确定性，尤其在缺乏足够实证数据支撑的前提下，可能导致模型外推偏差扩大；
- 极端事件（如重大疫情、自然灾害等黑天鹅事件）可能在短期内显著扰动死亡率水平，造成预测误差。

（2）模型表现问题及原因分析

从图 9 及前文模型评估分析中可以观察到，初步构建的预测模型在 80 岁以下的年龄段拟合效果良好，显示出较高的可靠性；而在 80 岁及以上的老年阶段，预测表现显著下降，死亡率出现非正常的跳跃式上升，甚至存在高于 1 的异常值，表明模型在高龄段存在系统性偏高问题。

出现这一问题的根本原因可能在于：80 岁以上的投保人口数据存在样本量稀疏、数据噪声较大或质量不高等情况，导致 Lee-Carter 模型或组合模型在高龄区间预测稳定性较差，难以捕捉真实的衰老趋势，无法维持合理的死亡率增长路径。

（3）模型改进方向

为修正上述高龄阶段预测偏差的问题，研究组将结合日本（JPN）、中国香港（HKG）、中国台湾（TWN）和韩国（KOR）等生命表质量较高且在文化和人口统计学上与中国大陆最为接近的国家与地区的数据，对 80 岁及以上老年人群的死亡率进行趋势锚定。

观察这些国际参考数据可发现，所有国家与地区 80 岁以后的死亡率普遍呈现稳定的指数型上升，未出现模型当前预测中所表现出的异常“峰值”或“跳跃”。因此，亟需借助额外的高质量死亡率数据对模型 80 岁后的高龄段进行结构性修正，以提升整体预测能力。

（二）模型改进与评估

1. 模型改进方法

原先模型对于 80 岁以下年龄段的死亡率预测表现稳健且准确，主要问题集中在 80 岁以上高龄阶段的预测失真。基于对该模型预测结果的深入分析和验证，研究组决定**采用外部高质量数据对 80 岁以上人口的死亡率进行替代性预测**。该方法不仅能够有效规避模型在极端高龄区间的预测误差，还便于实现对最高年龄至 120 岁的死亡率合理外推。

具体而言，本研究在初步模型的基础上构建了**“数据收集—国家筛选—模式适配—平滑衔接”**四步改进策略，并在实证中验证有效性。

（1）引入国际高质量生命表数据

从人类死亡率数据库（Human Mortality Database, HMD）数据库中收集全球 49 个国家和地区覆盖 0 至 110 岁完整年龄段的死亡率与生命表数据（具体国家和地区见附录 II 表 12），确保数据来源广泛、标准一致、质量可靠，并对原始数据进行标准化处理。

（2）基于死亡率模式相似度筛选参考国家

以 40 至 80 岁为基准年龄段，通过对数相似度度量，为每个性别分别筛选出 5 个与大赛提供的保险业死亡率数据模式最为相似的国家：

- **男性：**意大利、荷兰、日本、以色列、丹麦；
- **女性：**瑞士、瑞典、西班牙、意大利、挪威。

该筛选策略确保所选国家具有与我国保险人口接近的死亡率结构，增强高龄段预测的本土适应性。

（3）构建国际模式驱动的高龄死亡率修正曲线

提取所选国家 80 岁以上的年龄段死亡率曲线，计算其加权平均值；设定平滑连接点于 80 岁，保持与本研究预测曲线在该点连续；对连接处实施平滑处理，防止突变，维持曲线单调性；生成从 80 岁至 120 岁递增、平稳、符合生物规律的死亡率外推路径。

（4）形成最终全年龄段死亡率修正曲线

维持 80 岁以下年龄段的初步预测死亡率不变，应用基于国际数据的 80 岁以上预测死亡率，将两段曲线连接，并进行适当平滑处理，最终形成 0-120 岁全年龄段死亡率修正曲线。

2. 改进效果评估与对比分析

（1）误差显著降低，验证精度大幅提升

模型验证结果显示，经改进后，模型的预测性能显著提升。如表 6 所示，男性死亡率验

证的平均绝对误差由原先的 64.9%降至 8.4%，女性死亡率验证的平均绝对误差由 129.2%降至 13.1%。此外，男性最大误差从 788.8%大幅缩减至 94.2%，女性最大误差亦从 1816.1%显著下降至 48.7%。以上数据充分表明，改进措施在提升高龄段死亡率预测的准确性方面取得了显著成效。

表 6　2025 年保险人口死亡率预测模型改进前后验证平均绝对误差对比

性别	改进前平均绝对误差	改进后平均绝对误差	降幅
男性	64.9%	8.4%	56.5%
女性	129.2%	13.1%	116.1%

（2）消除异常突增峰值，曲线结构更具生物学合理性

观察改进模型后的预测曲线（图 15），可以发现模型改进成功消除了先前 80 岁以上死亡率预测中出现的异常突增峰值问题。现在，80 岁至 120 岁的死亡率呈现出更加平滑且符合自然衰老规律的递进趋势。此趋势反映出模型对高龄段死亡风险的合理刻画，不仅符合生物学常识，也提升了模型在高龄群体寿命预测中的应用价值，增强了其在保险定价和风险管理中的科学性和可靠性。

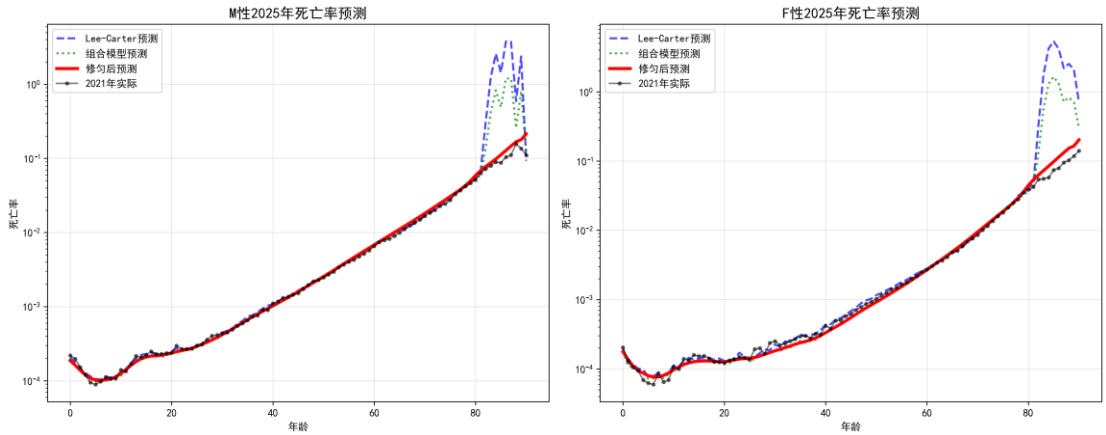

图 15　改进模型下 2025 年保险人口死亡率预测结果图

（M 性：男性；F 性：女性）

（3）本土化调整与国际经验有效融合

通过筛选与大赛提供的 80 岁以下保险业死亡率数据模式相似的国家作为参考，改进后的模型不仅引入了国际公认的高龄死亡率趋势，同时确保了预测结果与保险数据特定人口的特征高度契合，兼顾了本土化需求与国际经验的平衡。

如图 16 所示，改进后的模型在高龄阶段的死亡率预测结果与国际参考死亡率数据的差异较小，与改进前相比显著缩小，实现了更为精准的高龄段死亡率估计。

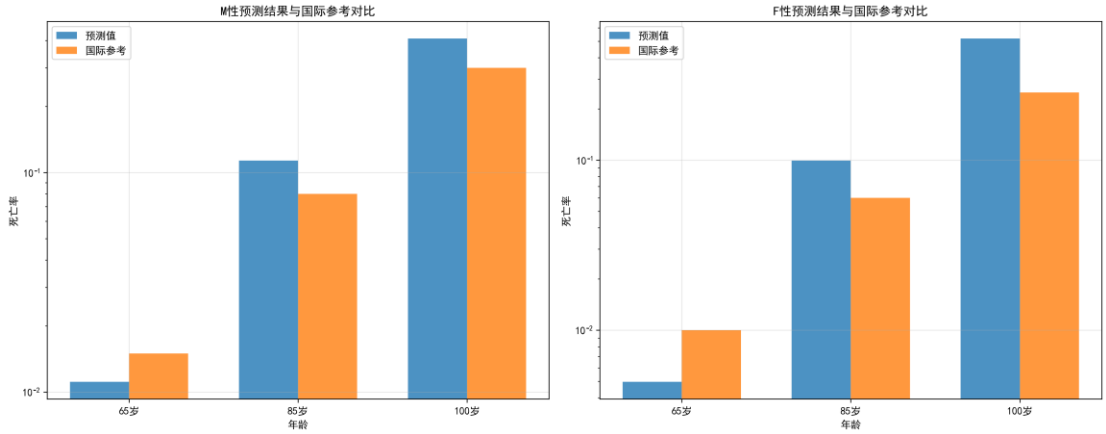

图 16　2025 年保险人口死亡率改进预测模型预测结果与国际参考数据对比图

3. 模型改进方法优势总结

本次模型改进方法的核心优势在于**引入了多国高质量、真实的生命表数据，借助成熟且经过充分验证的死亡率模式，显著提升了 80 岁以上高龄段的预测准确性**；同时**充分保留了原模型对 80 岁以下年龄段死亡率的稳健准确预测，确保已有数据的有效利用**。在该方法下获得的最终预测结果既符合生物学规律，又兼顾本土人口的特性，为长寿风险管理与控制提供了更加可靠的数据基础。

（三）保险人口 2025 年死亡率预测结果

基于上述改进后的混合模型，研究组对 2025 年 0-120 岁保险人口的死亡率进行了系统预测。表 7 详细列出了不同年龄男性与女性的预测死亡率数据，图 17 展示了预测死亡率随年龄变化的曲线，未出现异常波动，显示模型在捕捉年龄与死亡风险关系方面的良好表现。

本预测结果有助于为保险公司提供更加精准和稳健的风险评估依据，尤其在定价、准备金计提及风险管理等环节，能够有效降低因高龄人群死亡率预测不准确带来的潜在风险，增强保险产品的竞争力和可持续发展能力。

表 7　2025 年保险人口死亡率预测结果

年龄	男性死亡率	女性死亡率	年龄	男性死亡率	女性死亡率	年龄	男性死亡率	女性死亡率
0	0.000188	0.000176	41	0.001115	0.000366	81	0.134262	0.094501
1	0.000161	0.000137	42	0.001215	0.000403	82	0.253299	0.296672
2	0.000138	0.000110	43	0.001326	0.000444	83	0.418816	0.672054
3	0.000120	0.000095	44	0.001451	0.000493	84	0.601830	0.971259
4	0.000108	0.000086	45	0.001590	0.000549	85	0.743709	0.994827
5	0.000102	0.000079	46	0.001747	0.000613	86	0.863532	0.997293
6	0.000101	0.000076	47	0.001918	0.000683	87	0.953698	0.997755
7	0.000103	0.000076	48	0.002107	0.000757	88	0.960330	0.997909
8	0.000106	0.000080	49	0.002313	0.000835	89	0.961437	0.997979
9	0.000112	0.000087	50	0.002543	0.000922	90	0.962537	0.998046
10	0.000125	0.000097	51	0.002802	0.001017	91	0.963628	0.998108
11	0.000141	0.000106	52	0.003094	0.001124	92	0.964710	0.998167
12	0.000161	0.000115	53	0.003422	0.001244	93	0.965782	0.998222
13	0.000181	0.000122	54	0.003784	0.001381	94	0.966842	0.998273
14	0.000198	0.000126	55	0.004183	0.001536	95	0.967889	0.998322
15	0.000209	0.000129	56	0.004619	0.001712	96	0.968923	0.998367
16	0.000215	0.000130	57	0.005095	0.001912	97	0.969942	0.998410
17	0.000217	0.000130	58	0.005616	0.002138	98	0.970946	0.998450
18	0.000221	0.000128	59	0.006188	0.002395	99	0.971933	0.998487
19	0.000227	0.000127	60	0.006819	0.002688	100	0.972903	0.998522
20	0.000237	0.000127	61	0.007514	0.003026	101	0.973855	0.998555
21	0.000246	0.000130	62	0.008284	0.003413	102	0.974788	0.998586
22	0.000255	0.000134	63	0.009131	0.003854	103	0.975702	0.998615
23	0.000263	0.000139	64	0.010065	0.004352	104	0.976597	0.998642
24	0.000275	0.000140	65	0.011077	0.004918	105	0.977471	0.998667
25	0.000291	0.000142	66	0.012189	0.005567	106	0.978324	0.998690
26	0.000310	0.000146	67	0.013427	0.006327	107	0.979157	0.998712
27	0.000331	0.000153	68	0.014828	0.007220	108	0.979967	0.998733
28	0.000355	0.000163	69	0.016406	0.008264	109	0.980757	0.998752
29	0.000383	0.000172	70	0.018171	0.009477	110	0.981524	0.998770
30	0.000417	0.000182	71	0.020136	0.010874	111	0.982270	0.998787
31	0.000456	0.000191	72	0.022329	0.012478	112	0.982993	0.998802
32	0.000500	0.000202	73	0.024782	0.014308	113	0.983695	0.998817
33	0.000551	0.000213	74	0.027541	0.016395	114	0.984374	0.998830
34	0.000607	0.000226	75	0.030632	0.018786	115	0.985032	0.998843
35	0.000667	0.000238	76	0.034107	0.021576	116	0.985667	0.998854
36	0.000729	0.000248	77	0.037971	0.024814	117	0.986282	0.998865
37	0.000796	0.000260	78	0.043047	0.029383	118	0.986874	0.998875
38	0.000867	0.000278	79	0.051797	0.036484	119	0.987446	0.998885
39	0.000943	0.000302	80	0.075763	0.059897	120	0.987997	0.998893
40	0.001025	0.000332						

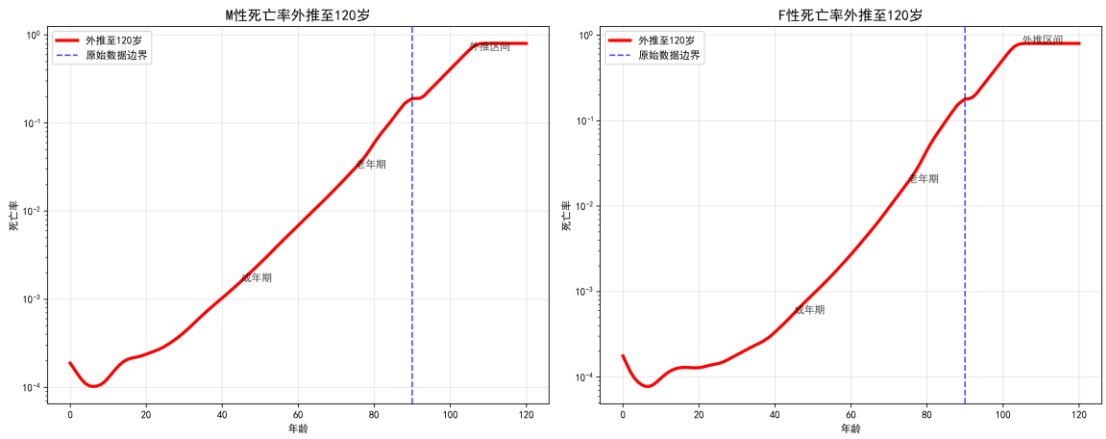

图 17　2025 年保险人口死亡率外推至 120 岁最终预测结果图

（M 性：男性；F 性：女性）

四、保险人口长期死亡率发展趋势预测分析

根据现有趋势，预计未来一个世纪内，保险人口的死亡率将**持续稳步改善**，但**改善速度有望逐渐放缓**。寿险行业普遍指出，死亡率的持续改善正在加剧长寿风险，因此准确理解这一趋势的方向及其影响具有重要意义。历史数据表明，女性的预期寿命普遍高于男性，且近年来各国数据均验证了女性在生存率方面的优势。鉴于保险人口的特征与一般人口相似，女性的生存率预计将持续高于男性。

参考瑞士再保险关于长期预测应综合历史趋势及医疗保健与生活方式进步的建议，本研究对死亡率的长期趋势预测将基于 2010-2021 年的保险业经验数据，结合已拟合至 2025 年的短期模型（Lee-Carter 与 Gompertz-Makeham 混合模型），采用一种**整体性方法**向前延伸——该方法将外推法与专家对未来医疗进步及社会变革的合理判断相结合，确保预测结果兼具数据驱动与前瞻性。

在此基础上，本研究假设男女两性生存率将持续提升，预期寿命将进一步延长，但提升过程可能呈现**波动性特征**。自 20 世纪 50 年代以来，心血管疾病及吸烟相关死亡率的下降体现了死亡率改善的阶段性突进；而未来的改善则更有可能依托于癌症治疗、精准医疗技术以及老年医学等领域的持续进步。本章的核心关注点是死亡率的长期下降趋势及相应的长寿风险增加，而非逐年或单年龄段的精确数值预测。

（一）方法论

对于 0-80 岁年龄段，延续使用 Lee-Carter 与 Gompertz-Makeham 组合模型，在预测时对死亡率指数进行平滑处理，并校准至保险人口经验数据。对于 80 岁以上年龄段，维持现有

的锚定高质量人口生命表（HMD）策略，以缓解尾部预测波动。

实践中，**极端高龄尾部**由类 Gompertz 外推方法（或逻辑“Kannisto”扩展法）控制曲线结构，并引入温和减速项以反映人类寿命的理论极限。现有研究表明，任何明显的高龄平台期在很大程度上是假象：在高质量数据中，死亡率即使在 100 岁之后仍呈指数增长[1-2]。Dang 等未在法国 105 岁以上数据中发现平台期的证据[1]；而 Gavrilov 和 Gavrilova 报告称，高质量数据的提高表明极高龄死亡率仍遵循 Gompertz 型对数线性增长[2]。

因此，本研究预测假设死亡率随年龄持续上升，仅通过模型参数设定反映改善速度的可能减缓，而非对曲线进行截断；同时考虑以下两项关键因素：

- **队列效应**：年轻队列得益于更良好的儿童健康状况与更先进的治疗；
- **选择效应**：年金保险保单持有人的健康状况普遍优于一般人群。

总体而言，研究组对模型进行校准，使其能反映合理的中期增益（多数分析认为每 10 年预期寿命增长约 2-3 年），且在远期阶段逐步收窄增长幅度。

（二）性别维度的死亡率趋势预测

未来一个世纪内，男性和女性的死亡率变化趋势如图 19-图 21 所示。图 18 展示了保险人口从 2025 年到 2125 年出生时预期寿命的预测趋势。整体来看，两条曲线均呈现平稳上升态势，其中女性（红线）预期寿命始终高于男性（蓝线），反映出持续存在的性别差异。曲线斜率在前几十年相对较为陡峭，主要归因于近期医疗技术进步及吸烟率下降带来的显著效益；随后增速逐步放缓，符合易得收益递减的规律。具体预测显示，女性预期寿命将从 2025 年的约 82 岁提升至 2125 年的约 92 岁，男性则由约 76 岁提升至约 86 岁。该终点水平与多项长期精算预测相符（例如，美国至 2100 年女性预期寿命约 86.9 岁，男性约 82.9 岁），亦契合无硬性寿命上限但改善速度放缓的人口统计模型[3]。该图的核心信息在于趋势方向：未来百年中，尽管年度改善幅度逐渐减弱，两个性别的死亡率整体呈现下降趋势，生存率持续提高。

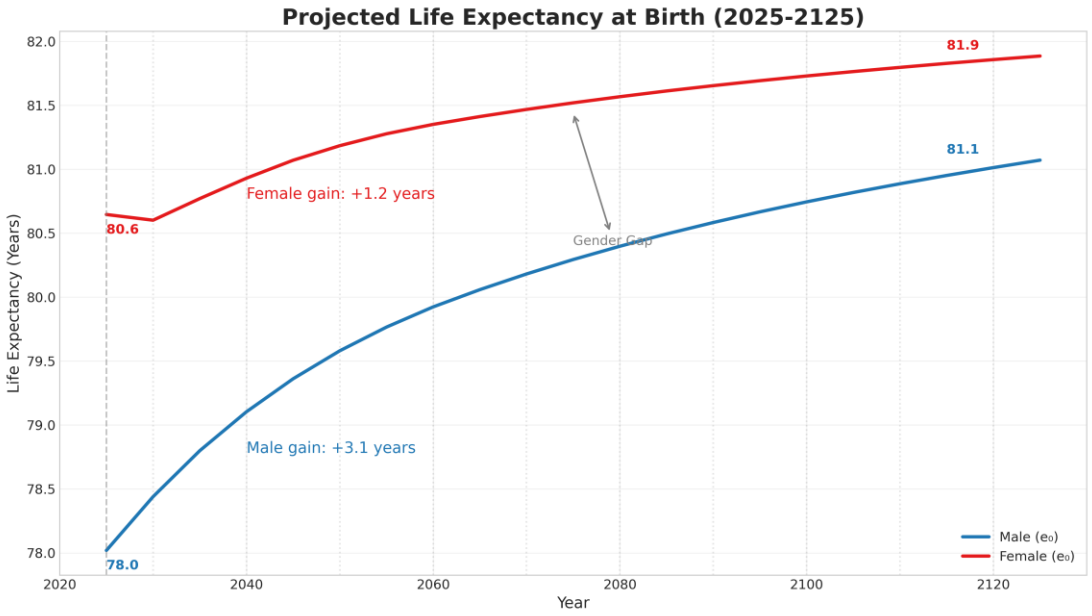

图 18　2025-2125 年保险人口出生时预期寿命变化趋势图

（Male：男性；Female：女性）

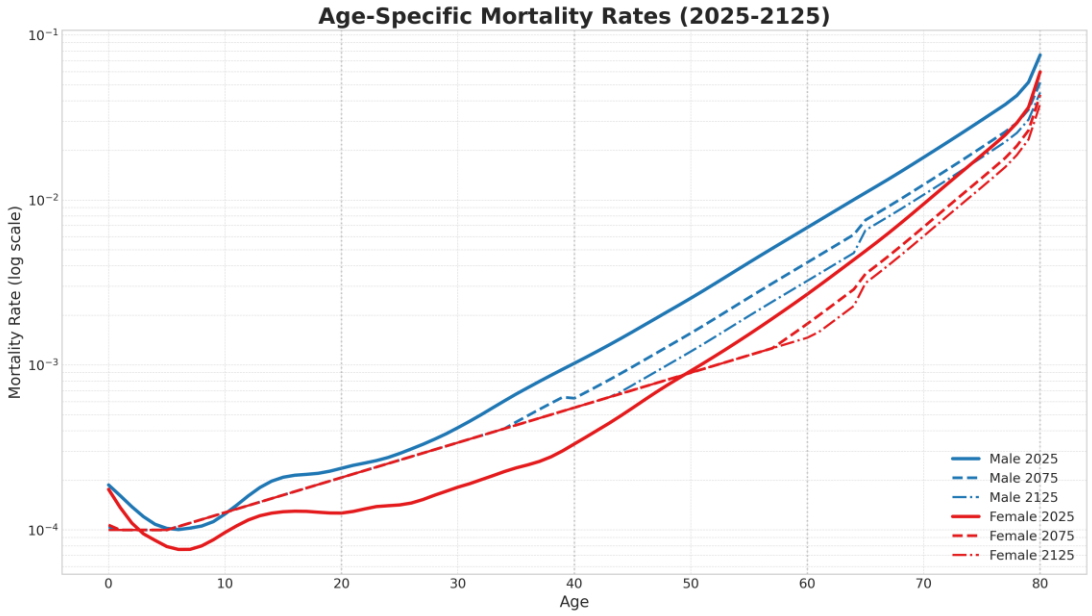

图 19　2025-2125 年保险人口死亡率分年份预测曲线图

（Male：男性；Female：女性）

图 20　2025-2125 年男性保险人口死亡率预测曲面图

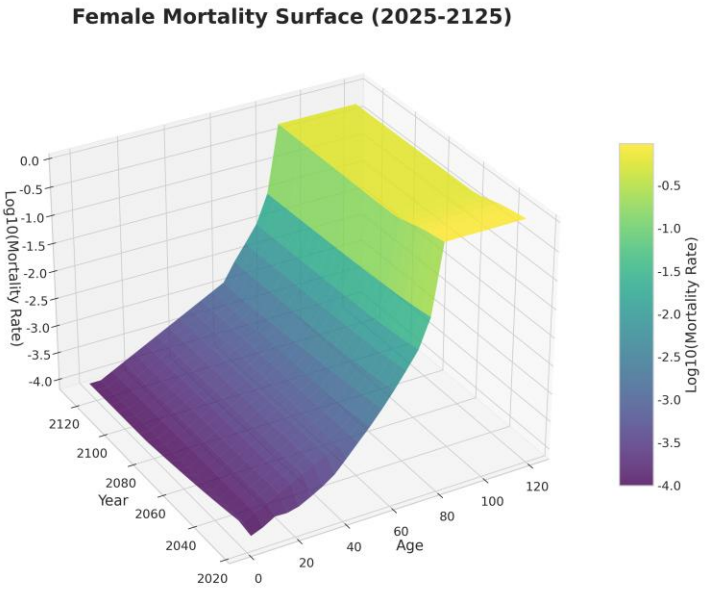

图 21　2025-2125 年女性保险人口死亡率预测曲面图

（三）生物学与人口统计学合理性分析

本文对 2025-2125 年保险人口长期死亡率趋势的预测，严格基于人口学统计研究与相关证据。关于人类寿命是否存在严格的生物学上限，目前学术界尚无明确共识。部分研究对“寿命即将接近极限”的观点提出了质疑。例如，Newman 指出，晚年阶段死亡率曲线中常见的趋缓现象常源于数据误差，而非真实的生理变化，这表明生物衰老过程在整个生命历程中持续稳定进行；尽管其研究也指出寿命可能最终趋于某一生物极限，但这一限制在可预见的未来尚不会显现[4]。

在本研究中，为兼顾理论可行性与实际应用，模型在长期预测中引入了一种递减的增益机制，即随着时间推移，死亡率改善幅度逐步收窄。这一假设基于生物逻辑（如拟合逻辑渐近线）以及现实中的医疗、生活方式等因素的逐步边际收益递减趋势，反映了寿命改善“减速而不终止”的可能路径。整体来看，本预测框架并未预设硬性生物极限，而是认为在未来一个世纪内寿命将持续改善，尽管其速度可能逐步趋缓。改善最终会受到抑制，但不会在近 100 年内发生。

在建模过程中，关键的人口统计学考量包括以下几个方面。

1. 死亡率减速与 Gompertz 假设

近期多项实证研究表明，高龄死亡率减速大多为数据偏差的产物[1-3]。因此，本研究假设在极高龄阶段死亡率遵循对数线性增长（Gompertz 型），而非出现“平台期”，具有充分的理论依据。

2. 平台期与寿命极限问题

虽然部分研究提出人类寿命存在理论上的上限[4]，但本研究并未在当前的寿命水平附近发现任何趋势性拐点的证据。历史经验表明，预期寿命通常以“波浪”形式推进，即通过阶段性的技术突破和健康行为改善逐步推动，而非稳定、线性地延伸。因此，本文假设未来寿命的改善以类似方式推进，而非在某个固定年龄突然停止，但增长幅度将逐渐收敛，符合逻辑渐近式的长期规律。

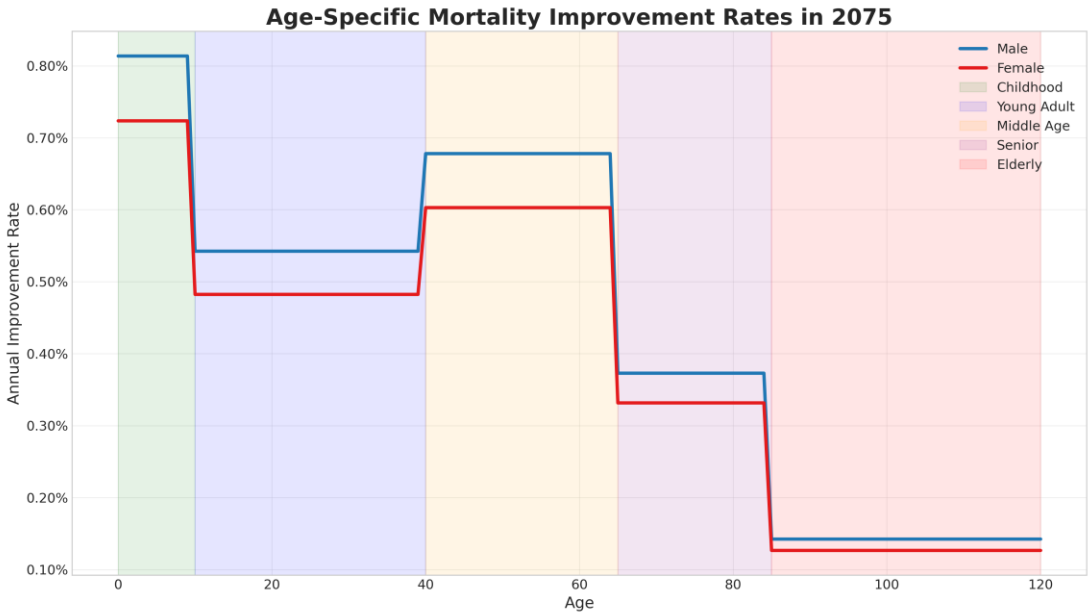

图 22　2075 年保险人口预测死亡率改善率图

（Male：男性；Female：女性）

3. 极高龄数据的稀疏性与外推方法

由于百岁以上人群数量极少，死亡率估计受样本波动和测量误差影响显著。为减轻该问题对模型预测的干扰，本研究将尾部锚定在人口基准（HMD）上，并使用平滑外推法进行处理。该策略强调趋势的合理性而非极端年龄点上的精确数值，如 110 岁的精确死亡率水平不如 100 岁以下的整体死亡率下降趋势重要。

4. 生活方式与风险因素变动

吸烟、肥胖、新发疾病及其干预手段等行为与环境变量是影响未来死亡率的重要因素。瑞士再保险指出，肥胖、糖尿病等慢性病的扩散可能会抵消医疗进步所带来的正向效果，减缓预期寿命的增长。本研究在建模中综合考虑这些因素，假设新兴治疗（如减肥药、先进癌症疗法）会带来适度净收益，但也承认不良健康行为可能抑制未来的改善。

5. 大流行病与外部冲击的短期效应

COVID-19 等公共卫生事件在短期内显著扰动了死亡率水平，但研究表明，其长期影响将逐步减弱，至 2030 年基本消退。保险公司不应将这种冲击过度外推至未来。因此，本研究在 2030 年后基本恢复了 2020 年前的死亡率趋势路径，并仅对可能存在残留效应的部分参数作出调整，避免因短期异常影响长期判断。

6. 社会经济差异与队列效应

不同群体的死亡率存在异质性。例如，瑞士再保险指出，美国不同社会经济地位的死亡率存在差异。考虑到保险人口（即养老年金保单持有人）在健康状况与风险行为方面往往优于一般人口，其死亡率改善趋势应更接近高收入群体或教育水平较高群体的路径。因此，本研究预期保险人口的死亡率改善趋势将与高收入群体保持一致。队列效应（儿童期条件更好的年轻队列）已通过年龄—时期维度的建模结构隐含地捕捉。

综上所述，本文对长期死亡率的预测在生物学和人口统计层面具有高度合理性，符合当前主流研究对人类寿命改善趋势的理解：人类死亡率将持续下降但改善幅度逐步放缓，女性领先于男性，且在 2125 年前不会出现死亡率戏剧性“触底”。图 23 是示意性的，但其形态与美国社会保障精算师和联合国等机构的预测相符[3]，说明本文预测模型具备较强的科学支撑与应用可行性。

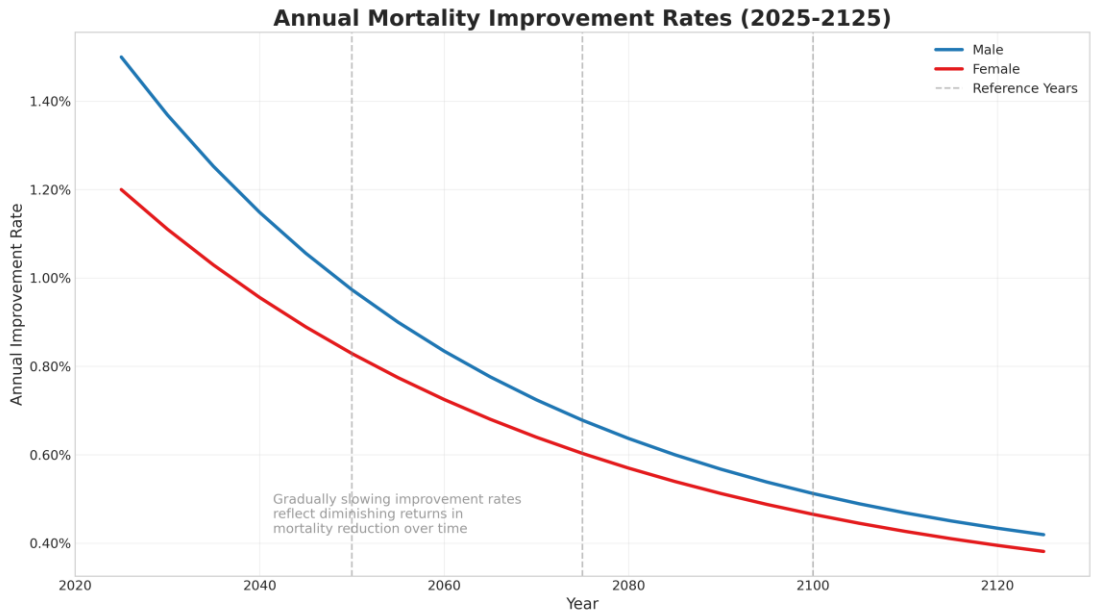

图 23　2025-2125 年保险人口预测死亡率改善率变化趋势

（四）结论与前景判断

本研究对 2025-2125 年死亡率趋势的预测结果显示，**男性和女性保险人口的死亡率均将逐渐下降。女性预期寿命仍将保持领先，两性寿命水平整体呈上升态势，但其增益幅度预计将随时间推移逐渐收窄。**这一趋势与人口统计学和精算学的最新研究一致：预计改善不会突然停止，但潜在因素（生物学极限、生活方式变化、数据不确定性）表明，随着年龄增长，

寿命增长将放缓。

总体而言，保险人口的整体死亡率趋势在未来一个世纪指向下行，体现出医疗技术进步、健康管理改善与社会结构演变等长期因素的累积影响。同时，该趋势也反映了人类寿命增长所面临的自然限制之间的平衡。

五、养老年金风险管控

（一）长寿风险识别与量化

随着我国人口老龄化程度不断加剧和预期寿命持续提升，死亡率改善已成为长期趋势。对于养老年金类保险产品而言，投保人寿命显著超过精算假设下的预期，导致保险公司面临着年金给付期限延长、支付总额增加、准备金不足和资本压力提升等严峻问题。

为了有效地进行年金产品优化、投资策略制定与风险转移设计，首先需要对长寿风险进行清晰的识别与合理的量化。本研究采用**死亡率改善情景下的年金现值差法**（ΔPV法），构建了长寿风险量化的核心模型。通过比较动态预测情景与静态定价情景下的年金期望现值（Expected Present Value, EPV），得到长寿风险敞口ΔPV，以衡量由于寿命延长造成的赔付现值超出原始定价所估现值的幅度。

1. 长寿风险敞口量化模型

假设投保人在x岁时投保终身养老年金，在z岁时开始领取，每年可领取固定金额A元，领取至身故或达到终止年龄 $T = 120$ 岁。

在定价情境下，使用静态死亡率表进行精算定价得到的年金现值为

$$EPV_{Pricing} = {}_{z-x}p_x \sum_{t=0}^{120-z} \frac{{}_tp_z \cdot A}{(1+r)^{z-x+t}} \tag{3}$$

其中，${}_{z-x}p_x$表示投保人从x岁存活到z岁的概率，${}_tp_z$表示投保人从z岁存活到$z+t$岁的概率，${}_0p_z = 1$，r为贴现率。${}_tp_z$与${}_{z-x}p_x$均可由静态死亡率计算得出，计算公式如下：

$${}_tp_z = \prod_{s=0}^{t-1} p_{z+s} = \prod_{s=0}^{t-1}(1-q_{z+s}) \tag{4}$$

$${}_{z-x}p_x = \prod_{s=0}^{z-x-1} p_{x+s} = \prod_{s=0}^{z-x-1}(1-q_{x+s}) \tag{5}$$

其中，q_{x+s}表示产品定价时所使用的投保人在$x+s$岁死亡的概率。

在预测情境下，使用随时间变化的预测死亡率进行精算定价得到的年金现值为

$$EPV_{pred} = {}_{z-x}\hat{p}_{x,k} \sum_{t=0}^{120-z} \frac{{}_{t}\hat{p}_{z,k+z-x} \cdot A}{(1+r)^{z-x+t}} \tag{6}$$

其中，${}_{z-x}\hat{p}_{x,k}$表示在年份k年龄为x岁的投保人能够存活到z岁的预测概率，${}_{t}\hat{p}_{z,k+z-x}$表示在年份$k+z-x$年龄为z岁的投保人能够存活到$z+t$岁的预测概率，${}_{0}\hat{p}_{z,k+z-x}=1$，$r$为贴现率。${}_{t}\hat{p}_{z,k+z-x}$与${}_{z-x}\hat{p}_{x,k}$均可由预测死亡率计算得出，计算公式如下：

$${}_{t}\hat{p}_{z,k+z-x} = \prod_{s=0}^{t-1} \hat{p}_{z+s,k+z-x+s} = \prod_{s=0}^{t-1} \left(1-\hat{q}_{z+s,k+z-x+s}\right) \tag{7}$$

$${}_{z-x}\hat{p}_{x,k} = \prod_{s=0}^{z-x-1} \hat{p}_{x+s,k+s} = \prod_{s=0}^{z-x-1} \left(1-\hat{q}_{x+s,k+s}\right) \tag{8}$$

其中，$\hat{q}_{x+s,k+s}$表示在年份$k+s$年龄为$x+s$岁的投保人死亡的预测概率。

在此基础上，定义长寿风险敞口ΔPV为

$$\Delta PV = EPV_{pred} - EPV_{Pricing} \tag{9}$$

若$\Delta PV > 0$，说明预测场景下投保人寿命长于定价假设，保险公司面临额外负担。

2. 长寿风险标准化指标

为使不同投保人、产品、性别之间具有可比性，本研究引入长寿风险率LRR这一指标，用于评估风险的相对严重程度。计算公式如下：

$$LRR = \frac{\Delta PV}{EAV_{Pricing}} \tag{10}$$

长寿风险率越高，表明相对风险越大。若长寿风险率为 10%，则说明保险公司可能需要增加约 10%的准备金才能覆盖预测场景下的养老年金支付需求。

3. 长寿风险量化结果

假设投保人在 2025 年购买年固定给付额为 10,000 元的终身养老年金产品，且男性投保人的养老年金领取年龄为 65 岁，女性投保人的养老年金领取年龄为 60 岁。为考察贴现率对长寿风险量化结果的影响，本文设置了**三个典型贴现率情景**进行对比分析：**1.5%、2.0% 和 2.5%**。其中，2.0%对应 2025 年中国 30 年期国债收益率，作为基准贴现率，具有较强的现实基础；1.5%为审慎贴现率情景，模拟低利率或极端市场下的风险敞口扩大效应；2.5%为行业常见定价假设，反映利率上行或长期资产匹配理想情形。三组贴现率共同构成了本研究中用于评估养老年金长寿风险的贴现参数体系。

分别使用 2025 年的死亡率、2025-2125 年的预测死亡率作为定价情景和预测情景下的年金现值计算依据，计算得到不同性别、不同投保年龄投保人在不同贴现率下对应的长寿风

险敞口和长寿风险率如表 8 所示。

表 8　长寿风险量化结果

贴现率	性别	投保年龄	领取年龄	风险敞口ΔPV（元）	长寿风险率LRR
1.50%	男	30	65	7,937.80	10.55%
		35	65	7,889.45	9.71%
	女	30	60	4,589.48	3.98%
		35	60	4,751.39	3.82%
2.00%	男	30	65	6,260.99	10.27%
		35	65	6,372.64	9.44%
	女	30	60	3,584.32	3.78%
		35	60	3,804.91	3.63%
2.50%	男	30	65	4,948.41	9.99%
		35	65	5,157.20	9.18%
	女	30	60	2,804.65	3.59%
		35	60	3,052.52	3.45%

整体来看，养老年金产品在预测死亡率持续改善的背景下，普遍存在长寿风险暴露，且该风险受到贴现率水平、性别差异与投保年龄的多重影响。

首先，从**贴现率变化的影响**来看，随着贴现率从 1.50%上升至 2.50%，长寿风险敞口和风险率整体呈下降趋势。较高的贴现率降低了未来现金流的现值，从而缓解了寿命延长带来的财务冲击。这一趋势在各类人群中均表现稳定，说明**贴现率是影响风险敏感度的重要参数**。

其次，从**性别差异**来看，男性投保人的长寿风险敞口ΔPV和长寿风险率LRR整体显著高于女性。在相同投保年龄与贴现率下，男性因寿命延长而导致的未来年金支付现值增幅更大，反映出定价假设相较于实际寿命的偏差较为显著。这表明，尽管女性具有更高的预期寿命，但男性在定价模型中对寿命改善的预估往往更为保守，所使用的静态死亡率普遍偏高，而实际预测死亡率下降幅度更大，导致未来实际寿命延长带来的风险暴露更为明显，从而增加了保险公司的支付责任。

再次，从**投保年龄维度**分析，投保年龄越小，长寿风险越高。对于男性和女性而言，相同贴现率下，30 岁投保者的ΔPV与LRR普遍高于 35 岁投保者。其原因在于较早投保意味着从保单购买到领取年金之间有更长的时间跨度，死亡率预测的不确定性更高，累积误差效应更明显，进而带来更大的偏离与风险敞口。值得注意的是，较早投保虽略有增加ΔPV，但对LRR影响有限。这表明，在生命周期更长的情况下，长寿风险在绝对值上有所增加，但相对风险水平仍处于可控范围内。换言之，早期投保在长寿风险管理方面并不导致风险比例的显

著恶化。

综上，不同贴现率场景下的量化结果揭示了养老年金产品的长寿风险特征，为后续的产品设计优化、投资管理策略和风险转移机制提供了坚实的量化基础。男性投保人、低贴现率及较早投保人群构成风险敞口较大的重点群体。保险公司在进行产品定价及风险管理策略制定时，应重点关注此类群体的寿命改善趋势，适当修正定价假设，并考虑通过资本准备、再保险或金融对冲工具应对潜在的风险暴露。

（二）产品设计优化方案

在长寿风险持续上升的大背景下，养老年金产品面临“生存期延长、支付期限拉长、准备金不足、资本压力增大”等多重挑战。传统固定年金产品（Fixed Life Annuity, FLA）在设计上对死亡率变化缺乏响应机制，导致当实际死亡率持续改善时，保险公司将承担不断增长的尾部支付责任，准备金压力与偿付能力风险急剧上升。

根据前文的量化结果，在贴现率为 2.0%的情况下，当前 30 岁男性投保人的长寿风险敞口ΔPV达 6,260.99 元，风险率IRR为 10.27%。在低利率情景（1.5%）下，ΔPV和IRR更高。这充分说明传统产品结构已难以适应未来人口寿命提升的趋势。因此，有必要从产品端对传统年金结构进行重构与优化，以增强其对死亡率不确定性的适应能力，在保障投保人基本养老收入的同时，提升可持续性与风险控制能力。

1. 产品设计优化方向

针对当前传统养老年金产品存在的问题，本文从给付形态、领取时间安排、支付期限控制和风险共担机制等方面出发，提出以下五类养老年金产品优化设计方向。

（1）领取额递增/递减年金产品

递增年金是随着投保人年龄增长，每期年金领取金额逐步上升的年金产品；递减年金则是在早期领取较高金额，后期领取额逐步下降的年金产品。两者均为“非平稳”给付结构，相对于传统年金的固定给付更贴合投保人真实的生命周期消费需求。

递减型年金适用于退休初期支出较大、但高龄阶段消费下降的投保人，如计划旅行、改善居住条件的主动退休人群。递增型年金能够帮助投保人应对在高龄阶段可能上升的医疗和照护开支，更适合对未来健康风险有高度关注的群体。

此类产品的核心优势在于优化年金现金流分布，通过前后期的支付差异缓解尾部风险集中的问题，同时提高投保人对年金的个性化认同感。对于保险公司而言，这种结构能够在不增加整体责任的前提下，对现金流进行内部再分配，从而实现一定程度的风险调节。

（2）年龄封顶型年金产品

年龄封顶型年金在合同条款中明确设定保险公司对养老金支付的年龄责任上限（如 105 岁或 110 岁），当投保人达到该年龄后，无论其是否在世，年金支付将自动终止，或通过一次性结清方式完成责任转移。

该设计的核心逻辑在于以“封顶责任”的方式对抗极端长寿情形下可能造成的尾部风险无限扩张问题。在人口平均寿命持续延长但极端长寿人群仍为小概率事件的现实背景下，封顶机制既能降低保险公司对超高龄阶段现金流的不确定性暴露，又能提升精算可控性和资本效率。

（3）死亡率互助型年金产品

死亡率互助型年金是一种基于参保人群内部“死亡率再分配”机制设计的养老产品，其核心理念是：将同一风险池内个体的死亡率偏差通过动态分摊机制在存活者之间共享，从而缓解保险公司面临的整体长寿风险压力。

在产品设计之初，保险公司按照预定死亡率假设计算各参保人年金支付额，并建立一个死亡率“浮动调整账户”。当实际死亡率低于预期（即寿命延长）时，意味着年金责任期被拉长，该账户面临支出压力，系统可自动启动“调整机制”：对尚存活者的未来领取额进行适度下调；反之，若某期实际死亡率高于预期，导致资金池出现盈余，则可将其按比例分摊给当期存活者，提升其后续年金水平。在这一过程中，保险公司作为管理者不承诺固定给付水平，而是在设定最低保障基础上，通过规则化、可预测的再分配机制完成支付调整，从而将系统性长寿风险转移至客户端进行“集体共担”。

该产品适用于团体客户或社群式参保对象，如企业年金计划、行业协会、社区型养老组织等，便于建立可控的风险池。其结构优势在于能够形成规模经济与长寿分摊双重作用，降低保险公司对极端尾部风险的单边暴露，同时避免定价过于保守所带来的市场竞争劣势。

（4）与死亡率挂钩的浮动型年金产品

浮动型年金产品将年金领取金额与死亡率挂钩，根据实际死亡率的变化进行动态调整。例如，在实际死亡率低于预期时适当下调给付，反之上调给付。浮动机制的引入使保险公司可将系统性死亡率偏差的财务影响转移至客户端，适用于风险承受能力较高、对养老金规划具备长期视野的投保人。

（5）最低保证死亡率年金产品

最低保证死亡率年金融合了死亡率浮动机制与最低保障机制，旨在兼顾长寿风险控制与投保人权益保障。该产品的年金支付金额由与死亡率改善相关的浮动因子决定，但设定一个

最低给付限额，确保即使在人口寿命显著改善、支付期延长的情形下，投保人仍可获得不低于合同约定标准的基础养老金收入。

这种结构一方面使保险公司可依据死亡率趋势动态调整责任，缓解尾部风险敞口；另一方面，又通过最低保障机制维护投保人对年金收入稳定性的预期，是一种在**“保障—灵活”**之间取得平衡的理想结构，适合具有一定风险承受能力、希望获得可预期养老金收入的主流群体；同时也为引入寿命再保险、寿命衍生品等风险转移工具提供了良好的结构化基础。

综合来看，最低保证死亡率年金产品在监管兼容性、投保沟通及风险转移对接方面具备较高的实用性，**适合在当前及未来长期趋势中作为核心优化方向进行推广**。下文将对其进行重点分析。

2. 最低保证死亡率年金

最低保证死亡率年金（Minimum Guaranteed Mortality Annuities, MGMA）是一种**年金给付额与死亡率动态挂钩**、但**设有最低给付保障**的终身年金产品。其年支付结构如下：

$$B_t = \max(B_0 \cdot f_t, B_{min}) = \max(B_0 \cdot f_t, (1-\xi)B_0) \quad (11)$$

其中，B_t为领取期内第t年的年金支付金额；B_0为起始时设定的基础年金金额；f_t为浮动因子，是反映死亡率改善程度的函数；$B_{min} = (1-\xi)B_0$，为最低年金保障金额，ξ为最大下调比例。

（1）浮动因子设定方法及最优选择

在最低保证死亡率年金（MGMA）产品中，浮动因子f_t是控制年金给付随时间调整的重要变量，其设定将直接影响产品对死亡率改善的响应能力与长寿风险敞口的大小，决定了产品应对长寿风险的能力、投保人的养老金稳定性体验，以及精算责任的可预测性。合理设计f_t的机制，不仅能够有效缓释死亡率持续改善所带来的尾部责任增长，还能维持产品的市场吸引力和投保人信任度。

合理的浮动因子f_t应具备以下属性：

- **反映死亡率趋势：**随着死亡率下降，年金逐步减少；
- **可控、平滑：**避免剧烈波动，保障投保人稳定领取预期；
- **结构可解释：**易于投保人理解、可纳入条款披露；
- **模型可计算：**便于精算现值计算、支持敏感性测试；
- **政策可兼容：**支持再保险、政策对接、风险管理工具嵌入。

为更好地设定f_t的具体形式，本研究将对死亡率比例法、固定递减路径法、寿命折算法、

死亡率改善率转化法等方法展开综合对比，并在此基础上选择最佳的设定方法。

死亡率比例法以预测死亡率与定价死亡率的比值决定每年的浮动因子，形式为

$$f_t = \min\left(1, \frac{q_{x+t}^{Pred}}{q_{x+t}^{Pricing}}\right) \tag{12}$$

其中，$q_{x+t}^{Pricing}$为产品定价时使用的$x+t$岁个体的死亡率；q_{x+t}^{Pred}为未来预测情境下$x+t$岁个体的死亡率。

尽管此方法直观、结构简单，便于快速量化死亡率改善对养老金支付的影响，但存在明显局限。首先，该方法本质上是一种“水平差异”式判断，对每年死亡率值极为敏感，若预测值微小波动，即可能导致年金金额跳变，**缺乏平滑性与可控性**。其次，该设定结构**刚性较强**，没有灵敏度调节参数，无法在精算建模中灵活调节对死亡率改善的响应强度。再次，由于该方法**直接与死亡率挂钩，缺乏投保人熟悉的解释路径**，在实际销售与条款披露中难以让投保人理解其背后的调整逻辑。此外，当预测死亡率意外偏高时，为保护投保人权益通常不允许年金“逆向上浮”，该方法会陷入上限约束，调整机制失效。

固定递减路径法是一种不依赖死亡率预测、直接预设年金逐年下降比例的方式，其浮动因子的形式为

$$f_t = (1-g)^t \tag{13}$$

或

$$f_t = f_0 - kt \tag{14}$$

其中，g是每期下调比例；或从f_0开始以分段方式逐期下降k。

这种设定计算简单、易于产品条款表达，便于投保人理解“年金每年下降一定比例”的直观概念。然而，其最大问题在于**脱离实际死亡率趋势变化**，无法动态响应人口寿命改善的速度或幅度。如果死亡率改善幅度超过预设下调路径，年金责任可能无法有效削减；而若未来死亡率稳定甚至回升，仍执行下降路径将造成**年金下调过度，损害投保人利益**。此外，由于该方法是**静态参数化设定**，在长期运营中缺乏灵活性与调整能力，不具备与精算模型、风险转移机制等对接的技术基础，**难以满足养老金的长期可持续性需求**。

寿命折算法基于预期寿命的变化进行统一折算，反映人口长寿趋势对年金支付责任的影响，其浮动因子的形式为

$$f_t = \frac{e_x^{Pricing}}{e_x^{Pred}} \tag{15}$$

其中，$e_x^{Pricing}$表示按定价假设计算的x岁个体的剩余预期寿命，e_x^{Pred}表示按预测死亡率计算

的x岁个体的剩余预期寿命。

该方法可提供一个**简洁、可量化的整体调整系数**，适用于总体责任水平调整，但也存在明显不足。首先，它是一种**静态的整体调节方式，无法用于逐年给付的动态调整**，不适合商业年金的年度计提和现金流精算。其次，该方法依赖于寿命期望的“平均化特征”，当死亡率仅在特定高龄段显著改善时（如 90 岁以上），寿命期望增幅可能较小，但尾部年金责任却已大幅增加，该方法**无法捕捉尾部责任膨胀所带来的精算压力**。同时，将“寿命”直接用于年金金额调整，在条款解释和投保人理解上存在**抽象性和歧义性**，较难转化为销售与合约表达。总体而言，该方法更适用于国家养老金或大规模制度性产品的风险评估，而不适用于面向个人市场的商业养老年金定价机制。

死亡率改善率转化法基于预测死亡率相较于定价死亡率的改善幅度，设计浮动因子为

$$f_t = 1 - \theta \cdot \Delta q_{x+t} = 1 - \theta \cdot \frac{q_{x+t}^{Pricing} - q_{x+t}^{Pred}}{q_{x+t}^{Pricing}} \quad (16)$$

其中，Δq_{x+t}为$x+t$岁个体的相对死亡率改善率；θ为调整敏感度系数，控制给付调整幅度；$q_{x+t}^{Pricing}$为产品定价时使用的$x+t$岁个体的死亡率；q_{x+t}^{Pred}为未来预测情境下$x+t$岁个体的死亡率。

该方法直接将预测死亡率与定价假设之间的相对偏差转化为浮动因子，从本质上建立了“责任差异→给付调整”的函数映射，有效将定价偏差转化为量化的年金下调路径，**在精算逻辑上具备直接对应性**，有利于风险识别与成本控制；同时，**依赖逐年改善率而非绝对值差距**，在死亡率连续变化的情形下可形成平滑的年金浮动曲线，**避免跳变现象和剧烈调整，**投保人在合同履行期间可获得连续、缓慢、可预测的年金调整体验，有利于信任维护和产品稳定运营；引入参数θ作为给付调整的“放大/缩小因子”，使保险公司可根据风险偏好、资本成本、再保险安排等因素设定不同的响应强度，实现从“温和调整”到“快速修正”的多层次设计；基于“若未来死亡率比预期更低，则养老金适度下调”的简单逻辑，**便于条款表达、客户沟通及监管备案**。

综上，死亡率改善率转化法兼具结构灵活性、响应敏感性、模型可控性与客户友好性，是综合优势最高的 MGMA 浮动因子设定方案。因此，本文选取该种浮动因子设定方式，并在此基础上构建 MGMA 模型。

（2）最低保证死亡率年金产品模型

为保护投保人的基本养老金权益，最低保证死亡率年金产品（MGMA）设定最低年金保障比例$1-\xi$，即投保人在任何情景下获得的年金不得低于基础给付金额B_0的$1-\xi$倍。使用

死亡率改善率转化法下的浮动因子f_t（如式（16）所示），得到 MGMA 产品领取期内第t年的年金领取额为

$$B_t = \max(B_0 \cdot f_t, (1-\xi)B_0) = B_0 \cdot \max\left(1 - \theta \cdot \frac{q_{x+t}^{Pricing} - q_{x+t}^{Pred}}{q_{x+t}^{Pricing}}, 1-\xi\right) \quad (17)$$

该结构在数值上体现为：若寿命延长显著，即预测死亡率明显低于定价，f_t下降；当f_t小于最低保障比例$1-\xi$时，年金金额触底，不再继续下调；若死亡率变化不明显，年金金额保持稳定。

（3）最低保证死亡率年金产品风险敞口计算

为量化 MGMA 产品对未来死亡率改善所带来精算责任膨胀的应对能力，本文采用ΔPV模型对其长寿风险敞口进行评估。

假设投保人在x岁时投保 MGMA 产品，在z岁时开始领取，每年领取浮动金额B_t元，领取至身故或达到终止年龄 $T = 120$ 岁。

在定价情境下，使用静态死亡率表计算得到的年金现值为

$$EPV_{MGMA}^{pricing} = {}_{z-x}p_x \sum_{t=0}^{120-z} \frac{{}_tp_z \cdot B_t}{(1+r)^{z-x+t}} \quad (18)$$

其中，${}_{z-x}p_x$表示投保人从x岁存活到z岁的概率，${}_tp_z$表示投保人从z岁存活到$z+t$岁的概率，${}_0p_z = 1$，r为贴现率。${}_tp_z$与${}_{z-x}p_x$可分别由式（4）、（5）计算得出。

需要特别注意的是，在定价情景中，由于使用的是静态死亡率，即死亡率没有改善，故有$f_t = 1$，因此$B_t = B_0 \cdot \max(f_t, 1-\xi) = B_0$。

在预测情境下，使用随时间变化的预测死亡率计算得到的年金现值为

$$\begin{aligned} EPV_{MGMA}^{pred} &= {}_{z-x}\hat{p}_{x,k} \sum_{t=0}^{120-z} \frac{{}_t\hat{p}_{z,k+z-x} \cdot B_t}{(1+r)^{z-x+t}} \\ &= {}_{z-x}\hat{p}_{x,k} \sum_{t=0}^{120-z} \frac{{}_t\hat{p}_{z,k+z-x} \cdot B_0 \cdot \max(f_t, 1-\xi)}{(1+r)^{z-x+t}} \end{aligned} \quad (19)$$

其中，${}_{z-x}\hat{p}_{x,k}$表示在年份k年龄为x岁的投保人能够存活到z岁的预测概率，${}_t\hat{p}_{z,k+z-x}$表示在年份$k+z-x$年龄为z岁的投保人能够存活到$z+t$岁的预测概率，${}_0\hat{p}_{z,k+z-x} = 1$，$r$为贴现率，$\theta$为死亡率调整敏感度系数，$\xi$为支付额最大下调比例。${}_t\hat{p}_{z,k+z-x}$与${}_{z-x}\hat{p}_{x,k}$可分别由式（7）、（8）计算得出。浮动因子$f_t$的计算公式为

$$f_t = 1 - \theta \cdot \frac{q_{z+t} - \hat{q}_{z+t,k+z-x+t}}{q_{z+t}}$$

最低保证死亡率年金（MGMA）的长寿风险敞口ΔPV_{MGMA}为

$$\Delta PV_{MGMA} = EPV_{MGMA}^{pred} - EPV_{MGMA}^{pricing} \quad (20)$$

该值越大，说明在预测死亡率下，被保险人的寿命延长越显著，且产品年金浮动部分对保险公司的责任增加越明显。

（4）数值模拟与对比分析

假设投保人在 2025 年购买年基础给付金额B_0为 10,000 元的 MGMA 产品，且男性投保人的养老年金领取年龄为 65 岁，女性投保人的养老年金领取年龄为 60 岁。

分别使用 2025 年的死亡率、2025-2125 年的预测死亡率作为定价情景和预测情景下的年金现值计算依据，在**贴现率 2.0%**的情景下，计算得到 35 岁男性和女性投保人在不同死亡率调整敏感度系数θ、支付额最大下调比例ξ下对应的长寿风险敞口ΔPV_{MGMA}和长寿风险率LRR_{MGMA}如表 9、表 10 所示。

表 9　MGMA 产品的长寿风险敞口与长寿风险率（男性，35 岁投保，65 岁领取）

θ \ ξ	0	0.025	0.050	0.075	0.100	0.125	0.150
0	6,372.64	6,372.64	6,372.64	6,372.64	6,372.64	6,372.64	6,372.64
	(9.44%)	(9.44%)	(9.44%)	(9.44%)	(9.44%)	(9.44%)	(9.44%)
0.05	6,372.64	5,327.97	5,327.97	5,327.97	5,327.97	5,327.97	5,327.97
	(9.44%)	(7.89%)	(7.89%)	(7.89%)	(7.89%)	(7.89%)	(7.89%)
0.10	6,372.64	4,525.84	4,283.31	4,283.31	4,283.31	4,283.31	4,283.31
	(9.44%)	(6.71%)	(6.35%)	(6.35%)	(6.35%)	(6.35%)	(6.35%)
0.15	6,372.64	4,525.84	3,240.25	3,238.65	3,238.65	3,238.65	3,238.65
	(9.44%)	(6.71%)	(4.80%)	(4.80%)	(4.80%)	(4.80%)	(4.80%)
0.20	6,372.64	4,525.84	2,679.04	2,193.99	2,193.99	2,193.99	2,193.99
	(9.44%)	(6.71%)	(3.97%)	(3.25%)	(3.25%)	(3.25%)	(3.25%)
0.25	6,372.64	4,525.84	2,679.04	1,180.73	1,149.32	1,149.32	1,149.32
	(9.44%)	(6.71%)	(3.97%)	(1.75%)	(1.70%)	(1.70%)	(1.70%)
0.30	6,372.64	4,525.84	2,679.04	832.24	107.86	104.66	104.66
	(9.44%)	(6.71%)	(3.97%)	(1.23%)	(0.16%)	(0.16%)	(0.16%)
0.35	6,372.64	4,525.84	2,679.04	832.24	-780.74	-939.57	-940.00
	(9.44%)	(6.71%)	(3.97%)	(1.23%)	(-1.16%)	(-1.39%)	(-1.39%)
0.40	6,372.64	4,525.84	2,679.04	832.24	-1,014.56	-1,974.56	-1,984.66
	(9.44%)	(6.71%)	(3.97%)	(1.23%)	(-1.50%)	(-2.93%)	(-2.94%)
0.45	6,372.64	4,525.84	2,679.04	832.24	-1,014.56	-2,692.85	-3,024.53
	(9.44%)	(6.71%)	(3.97%)	(1.23%)	(-1.50%)	(-3.99%)	(-4.48%)
0.50	6,372.64	4,525.84	2,679.04	832.24	-1,014.56	-2,861.36	-4,011.18
	(9.44%)	(6.71%)	(3.97%)	(1.23%)	(-1.50%)	(-4.24%)	(-5.94%)

注：贴现率为 2%，年基础给付金额为 10,000 元。括号外数据为 MGMA 产品的长寿风险敞口（元），括号内数据为 MGMA 产品的长寿风险率。

表 10　MGMA 产品的长寿风险敞口与长寿风险率（女性，35 岁投保，60 岁领取）

θ \ ξ	0	0.025	0.050	0.075	0.100	0.125	0.150
0	3,804.91	3,804.91	3,804.91	3,804.91	3,804.91	3,804.91	3,804.91
	(3.63%)	(3.63%)	(3.63%)	(3.63%)	(3.63%)	(3.63%)	(3.63%)
0.05	3,804.91	2,493.45	2,493.45	2,493.45	2,493.45	2,493.45	2,493.45
	(3.63%)	(2.38%)	(2.38%)	(2.38%)	(2.38%)	(2.38%)	(2.38%)
0.10	3,804.91	1,244.20	1,181.99	1,181.99	1,181.99	1,181.99	1,181.99
	(3.63%)	(1.19%)	(1.13%)	(1.13%)	(1.13%)	(1.13%)	(1.13%)
0.15	3,804.91	1,090.45	-106.46	-129.47	-129.47	-129.47	-129.47
	(3.63%)	(1.04%)	(-0.10%)	(-0.12%)	(-0.12%)	(-0.12%)	(-0.12%)
0.20	3,804.91	1,090.45	-1,316.50	-1,426.26	-1,440.93	-1,440.93	-1,440.93
	(3.63%)	(1.04%)	(-1.26%)	(-1.36%)	(-1.38%)	(-1.38%)	(-1.38%)
0.25	3,804.91	1,090.45	-1,624.01	-2,698.01	-2,743.56	-2,752.39	-2,752.39
	(3.63%)	(1.04%)	(-1.55%)	(-2.58%)	(-2.62%)	(-2.63%)	(-2.63%)
0.30	3,804.91	1,090.45	-1,624.01	-3,877.21	-4,017.82	-4,058.87	-4,063.85
	(3.63%)	(1.04%)	(-1.55%)	(-3.70%)	(-3.83%)	(-3.87%)	(-3.88%)
0.35	3,804.91	1,090.45	-1,624.01	-4,338.47	-5,284.95	-5,337.62	-5,374.18
	(3.63%)	(1.04%)	(-1.55%)	(-4.14%)	(-5.04%)	(-5.09%)	(-5.13%)
0.40	3,804.91	1,090.45	-1,624.01	-4,338.47	-6,437.92	-6,609.37	-6,657.43
	(3.63%)	(1.04%)	(-1.55%)	(-4.14%)	(-6.14%)	(-6.31%)	(-6.35%)
0.45	3,804.91	1,090.45	-1,624.01	-4,338.47	-7,015.42	-7,867.38	-7,929.18
	(3.63%)	(1.04%)	(-1.55%)	(-4.14%)	(-6.70%)	(-7.51%)	(-7.57%)
0.50	3,804.91	1,090.45	-1,624.01	-4,338.47	-7,052.93	-8,998.62	-9,200.93
	(3.63%)	(1.04%)	(-1.55%)	(-4.14%)	(-6.73%)	(-8.59%)	(-8.78%)

注：贴现率为 2%，年基础给付金额为 10,000 元。括号外数据为 MGMA 产品的长寿风险敞口（元），括号内数据为 MGMA 产品的长寿风险率。

数值模拟结果显示，MGMA 产品通过引入支付金额调整机制及风险敏感度参数θ和支付最大下调比例ξ，显著缩减了长寿风险敞口和长寿风险率。当θ和ξ均取 0 时，MGMA 产品退化为传统固定年金，此时的风险敞口和风险率反映了传统年金产品面临的长寿风险水平。结果表明，传统年金在面对寿命延长时，风险敞口和风险率较高，保险公司所需承担的长寿风险较大，财务压力显著。

相较于传统年金，MGMA 产品通过动态调整年金支付金额，有效对冲了死亡率预测偏差带来的风险，降低了保险公司的潜在损失。**随着参数θ和ξ的逐步增大，风险敞口和风险率均呈现稳步下降趋势**，部分参数组合下甚至出现风险敞口为负的情况，说明预测死亡率情景下的年金现值低于定价情景，保险公司可能获得经验收益。这种现象表明，在未来死亡率上升的假设下，通过灵活调整支付策略，保险公司能够有效降低长寿风险负担，从而实现风险转移与收益优化，体现了风险管理工具的经济价值和应用潜力。

参数θ作为风险敏感度系数，直接反映了年金支付对死亡率变化的响应强度。较低的θ值意味着支付调整对死亡率预测偏差的敏感性较弱，长寿风险敞口和风险率维持较高水平；而随着θ的增加，支付调整机制更积极响应实际死亡率的变化，风险缓释效果显著提升。模拟结果中，θ值从 0 逐渐增加至 0.5 时，风险敞口出现明显下降，说明**适当提升风险敏感度系数有助于增强产品对长寿风险的防御能力**。

支付额最大下调比例ξ设定了年金给付金额的下限，是保障投保人利益的关键参数。ξ值越大，意味着支付金额的下调空间越大，保险公司可以在更大范围内降低支付压力，从而进一步压缩风险敞口和风险率。但ξ过大可能影响投保人的稳定收入预期，降低产品吸引力。数值模拟表明，**合理设定ξ可以在控制风险和保障投保人权益之间取得平衡**，尤其在θ较高时，较大的ξ值仍能实现较好的风险缓释，体现出θ和ξ参数间的协同作用。

此外，男性和女性投保人在长寿风险表现上存在显著差异。总体来看，男性的风险敞口和风险率普遍高于女性，反映了男性寿命预测中更大的不确定性和更高的长寿风险。女性风险敞口最高约为 3,805 元，风险率为 3.63%，均显著低于男性。而且，随着θ和ξ的提升，女性的风险敞口降幅更大，显示出风险管理措施对女性投保人的风险缓释效果更为明显。

（5）最低保证死亡率年金产品设计与风险管理建议

MGMA 产品不仅为保险公司提供了有效的长寿风险缓释工具，也推动了寿险产品设计向更加灵活和智能化的方向发展。

在实际运营中，保险公司应注重**合理设定并动态优化θ和ξ参数**，以实现风险管理与客户服务的平衡。精准的死亡率预测和模型校准是保证调整机制有效性的关键，建议强化对死亡率数据的持续收集和分析，提高预测的准确性和及时性。同时，应构建完善的风险监测和动态管理体系，及时发现和应对风险变化，确保产品在不同市场环境下的稳健运行。

此外，针对不同性别、年龄段及健康状况的投保人群体，保险公司可**设计差异化的 MGMA 方案**，增强产品的个性化和市场适应性。在推广过程中，加强教育和沟通，明确解释支付调整机制的必要性和运作方式，有助于提升投保人对产品的理解和接受度，增强客户满意度和忠诚度，促进业务的可持续发展。

（三）投资管理策略

养老年金产品作为典型的长周期、稳定负债类险种，其资产端管理策略必须紧密围绕负债特征展开。前文量化分析结果表明，年金现值对贴现率及寿命改善趋势敏感，通过引入 MGMA 产品结构，能够有效降低长寿风险敞口，特别是在合理设定死亡率调整敏感度系数

θ和支付额最大下调比例ξ两个参数下，ΔPV_{MGMA}可以大幅降低，接近为 0，甚至出现风险过度转移的负值。

然而，这种产品结构性缓释在多数情况下只能部分解决问题，保险公司仍需在资产端采取全面、细致、动态的投资管理策略，才能实现“精算责任稳定+资本消耗可控+客户权益保障”的三重目标。

1. 资产久期与现金流匹配：建立“中长期+尾部锁定”结构

从 MGMA 结构给付特征来看，年金支付呈现**初期稳定、后期缓降、底线保障**的三段结构。结合 MGMA 产品模型输出的现值与浮动路径结果，可提取不同年龄组、性别和参数组合下的年金支出现金流序列。保险公司可将该责任流作为核心输入，构建现金流匹配模型，用于指导以下资产配置：

（1）短中期债券（5-10 年）：匹配初期高额年金支出，保障产品现金流稳定；

（2）中长期利率债券（15-30 年）：匹配保障责任底线区间，为贴现率压力提供底层对冲；

（3）超长期债权计划（30 年+）或储备式资产池（如不动产债权、基础设施项目类 ABS）：锁定高龄责任现金流峰值，有助于对冲贴现率变化对尾部现值膨胀的敏感性。

建议在贴现率 2.0%下建立现金流匹配模型，以 MGMA 在不同θ, ξ参数下的给付路径作为目标序列，指导资产现金流配置比例。并通过 1.5%-2.5%贴现率情景进行压力测试，以确定久期敏感性和再投资节点。

2. 利差稳定与利率风险管理：控制贴现率敏感敞口

传统年金产品责任现值对贴现率变化极度敏感，MGMA 结构产品虽缓释部分波动，但保障底线仍为刚性负债，尤其在低利率周期，保险公司可能陷入“利差挤压”或“利差反转”困境。为此，建议其构建“底线—浮动分层型投资组合”：

（1）基于保障底线建立“利差底账”：用固定利率类资产（如国开债、政策性金融债、超长期国债）锁定此部分现金流；

（2）对于浮动年金区间，采用浮息债、MBS 等利率弹性较高的工具作为浮动调节缓冲带，对冲再投资风险；

（3）引入利率掉期、利率上限期权等衍生品进行利差保护，尤其在低利率或倒挂利差周期中，防止责任现值急剧扩张。

同时，还可建立贴现率敏感性矩阵，将不同参数设定下的ΔPV曲线与收益曲线波动进行

映射，为再投资时点选择提供风险参考。

3. 动态应对机制与策略联动：建立“产品—精算—投资”闭环

MGMA 产品的参数（θ, ξ）决定了未来年金给付的波动性，从而影响负债现值和流动性需求。因此，保险公司在资产配置策略中，不能“孤立看资产”，而应嵌入以下机制：

（1）每季度或半年重估ΔPV变化曲线，更新资产配置比例；

（2）根据浮动参数敏感性，动态划分“刚性”与“弹性”资产池比例；

（3）结合投保人年龄结构变化与产品销售结构，通过寿命预测更新和市场利率动态调整资产久期与波动敞口；

（4）在年金责任逐步下降过程中，逐步转移权益资产比重，增强资本释放效率；

（5）在浮动比例较大的产品中，设立“责任压力缓冲基金”，作为资产流动性保障工具。

4. 长期配置方向建议：增加现金流导向与人群匹配型资产

面对未来人口老龄化带来的结构性挑战，长远来看，主销养老年金的保险公司重点发展下列资产方向：

（1）基础设施 REITs 与能源项目债：为尾部责任提供稳定现金流支持；

（2）养老金目标基金（TDF）与养老目标日期类产品：与客户生命周期匹配，可嵌入产品责任转移机制；

（3）寿命联动资产（如长寿指数债券、长寿风险转移资产）：作为衔接风险转移机制的准备资产基础；

（4）适度引入分红/参与型权益资产：提升浮动区间回报空间，增强长期吸引力。

5. 可持续性要求与责任特征对接：引入 ESG 与抗通胀资产配置机制

考虑到养老年金产品通常为长期稳定型责任，对抗通胀和社会责任投资（ESG）将成为关键。因此，建议保险公司：

（1）引入抗通胀类资产：如与 CPI 挂钩的城投债、农地租赁债券；

（2）将 ESG 纳入策略评估：如绿色 REITs、低碳基础设施计划，适用于保险资金长期约束；

（3）推动“养老金+ESG”并行机制，实现“稳健投资+责任管理”双目标。

6. 数据驱动投资演进路径：构建责任联动型投资决策系统

当前阶段，数据基础仍以死亡率和责任现值为主，尚未具备完整的市场—责任动态联动

模型，建议保险公司在中长期建设以下技术能力：

（1）构建“ΔPV—利率—资产收益率”三维敏感性分析系统；

（2）搭建“精算—责任—资产”三端联动的数据平台；

（3）以 MGMA 责任路径为主线，开发“责任驱动型资产筛选工具”，实现策略推荐自动化。

（四）风险转移创新

长寿风险的本质是“未来死亡率改善的不确定性”。MGMA 产品通过结构性年金浮动机制初步建立了防线；责任匹配、利差管理和长期资产配置等手段也能够帮助保险公司部分吸收长寿风险责任。但长期来看，单一依靠资产端缓冲难以完全覆盖未来死亡率持续改善的不确定性，特别是在**贴现率长期走低、人口老龄化加剧、客户生存期远超预期**的极端情境下，保险公司仍面临巨大尾部责任暴露。

因此，需在产品设计与资产配置之外，构建长寿风险的外部风险转移机制，通过再保险、资本市场工具、互助结构等创新路径，将部分或全部尾部风险转出，实现**“风险分层+风险转出”协同机制**，从而建立完整的长寿风险管控闭环。

1. 再保险机制扩展：建立 MGMA 尾部责任再保通道

传统再保险主要用于应对重大疾病、自然灾害或退保高峰等突发性风险，而长寿风险作为一种缓慢累积、波动较低但长期放大的责任风险，目前尚缺乏成熟的再保产品覆盖路径。

针对 MGMA 产品，建议保险公司与再保险公司协商设计以下结构化再保险方案：

（1）尾部超额赔付型再保险：设定责任现值超过某预期值（如ΔPV超出定价阈值的 10%）后由再保险公司覆盖；

（2）高龄触发型比例再保险：如被保险人活至 90 岁后，责任现值中超过基础预期的部分由保险公司和再保险公司按比例分摊；

（3）人群层级打包再保险：将同一出生队列 MGMA 客户的高龄尾部责任集合打包，通过人群池化形成再保险谈判基础。

该机制有助于将不可预见的极端尾部风险通过精算模型参数化转移，降低保险公司资本金压力，增强 MGMA 产品推广的可持续性。

2. 资本市场工具创新：探索长寿债券与衍生品挂钩机制

资本市场对可交易性强、结构清晰的风险敞口具备较高吸收能力。随着海外“寿命证券

化”工具的发展，中国养老保险行业也可尝试探索适合本土的长寿风险证券化路径，包括：

（1）长寿债券：挂钩人群生存率或死亡率指数，发行方向为政府、保险集团、养老金池，由市场机构承接部分超额给付风险；

（2）死亡率联动衍生品：如死亡率上限互换、死亡率改善指数期权，用于对冲ΔPV突增带来的精算责任扩张；

（3）MGMA 结构联动指数产品：将 MGMA 浮动参数路径（如年金下降幅度）构建为衍生品基础值，在资产管理产品中嵌入，与保险责任动态对冲。

这类工具在我国目前仍属于相对前沿的尝试，可借鉴海外经验并结合国内死亡率模型发展，实现责任资本市场分摊。

3. 内部互助机制尝试：设计死亡率互助型年金结构

在 MGMA 产品的浮动设计基础上，可以进一步嵌入人群互助特征，通过生存人群对已故人群的隐性给付分摊，在公司内部建立“长寿内部共济机制”，形成死亡率互助型年金，其要点包括：

（1）参与者之间资金共享：若某年度整体死亡率低于预测，节省的责任现值部分用于补贴长寿客户的后续年金；

（2）基于队列的年金修正机制：同一出生队列的 MGMA 客户，以当期生存人数为基准，重新测算后续年金支付；

（3）激励与保障兼顾：通过分摊机制减少公司尾部责任，同时增强客户参与意识和契约稳定性。

这类结构无需依赖外部再保或资本市场，可作为风险转移与客户行为管理结合的微创新工具，也更容易实现初步落地试点。

4. 建立全生命周期风险分层机制：从起领前到高龄阶段分段转移

长寿风险并非在退休后才形成，而是在从投保到领取的整个生命周期中逐步积累。因此，建议保险公司在 MGMA 产品中建立生命周期阶段性风险转移机制：

（1）起领前（30-60 岁）：预测不确定性高、责任滞后，可使用死亡率互助机制缓冲责任波动；

（2）领取初期（60-80 岁）：年金高峰期，责任负担重，可使用责任锁定再保险或利率锁期工具；

（3）高龄尾部（80 岁以上）：生存概率低但支付敏感性高，可启动尾部再保险、长寿债

承接机制。

该风险转移机制可结合 MGMA 浮动路径，构建分阶段风险图谱，并作为责任准备金评估与资本安排依据，推动保险公司从“静态再保”向“动态转移”模式升级。

（五）分阶段实施计划

MGMA 等养老年金产品的推广及其配套的投资管理和风险转移机制，涉及产品开发、系统建设、责任精算、资产配置、监管适配等多个环节，不能一蹴而就。保险公司应采取**“分阶段、分模块、分层次”**的稳健推进策略。

结合前文研究成果，本节提出三阶段实施计划，从产品初步试点到制度性部署，再至系统性集成，逐步构建起应对长寿风险的完整解决体系。

1. 阶段一：产品构建与模型落地（第 0-1 年）

初始阶段的核心在于完成 MGMA 等产品的原型设计和责任评估模型的搭建，建立基础的风险识别和缓释能力。

建议保险公司优先完成 MGMA 等产品的核心参数设定（如浮动因子f_t、死亡率调整敏感度系数θ、支付额最大下调比例ξ），并将其纳入条款语言和精算责任测算流程；落地ΔPV长寿风险量化模型，在定价假设、预测死亡率路径和贴现率变动下，完成责任现值压力测试，并据此制定产品初期给付区间。同时，构建年金责任现金流数据库，并将其作为投资配置的输入基准，启动初步的资产责任匹配。建议配置以中长期固定利率债为主的资产组合，锁定基础利差空间。此外，公司内部应组建产品—精算—投资联合小组，推进精算系统和责任模型的数字化改造，为 MGMA 等产品的大规模推广打下基础。

2. 阶段二：机制联动与风险对接（第 1-3 年）

中期推进的核心是实现 MGMA 等年金产品结构与资产配置、风险转移机制之间的动态联动，提升整体系统的抗压能力与运营效率。

在产品方面，应逐步推进 MGMA 等产品的系列化开发，针对不同人群设计浮动路径分层方案，引入生命周期配置建议、激励型浮动策略等差异化创新。投资端应全面构建“底线保障+浮动弹性”的资产结构，使用久期匹配工具（长期债、浮息产品、不动产债权等）应对不同年金阶段的现金流。同时，启动与再保险机构的初步谈判，建立“高龄责任比例再保”或“尾部超额给付再保”机制，将极端长寿情境下的风险部分外部化。此外，保险公司可在选定 MGMA 人群中试点死亡率互助型机制，实现参与者之间的责任内部调节，以提高风险

缓释能力并增强客户黏性。

3. 阶段三：制度集成与生态构建（第 3-5 年及以后）

长期目标应聚焦于 MGMA 等产品责任与公司战略、资本计划、监管制度之间的深度融合，实现制度化、标准化、平台化运营。

在精算管理方面，可将ΔPV作为偿付能力二代补充指标纳入压力测试体系，形成“死亡率动态识别—责任敞口定量评估—投资策略联动配置”的管理闭环。在资本市场合作方面，应逐步推动 MGMA 等产品责任池的证券化尝试，探索与寿命指数挂钩的金融工具开发，并与再保险机制共同构建“可交易的长寿风险”转移渠道。内部系统应实现责任估算、现金流预测、投资响应、风险敞口识别的全流程数字化，通过可视化模型实时监控责任变化与资本需求。同时，建议联合行业协会推动死亡率预测模型、浮动路径参数和责任评估标准的统一，推动构建全国范围内的长寿风险评估与数据共享平台，形成面向寿命延长趋势的行业公共基础设施。

（六）政策与监管建议

随着人口老龄化的加剧和寿命持续改善，长寿风险已经成为影响保险业财务稳健和产品可持续性的重要挑战。前文通过ΔPV风险量化模型和 MGMA 结构型年金产品的优化方案，初步构建了企业层面应对长寿风险的路径。然而，要真正推动这类产品在市场中落地并长期运营，还需要在宏观政策和行业监管层面建立完善的制度保障体系。为此，本文基于当前国内市场实际情况与已有改革动向，提出以下综合性政策与监管建议，旨在为保险公司开展长寿风险管理和 MGMA 等养老年金产品创新营造更加明确、稳定、包容的制度环境。

首先，应明确 MGMA 产品在现行监管体系下的分类与条款监管口径。由于该类产品引入死亡率驱动的年金浮动机制，其责任计提、客户披露与精算准备金制度均与传统年金存在差异。建议监管机构出台针对结构性浮动年金产品的条款审查与责任评估指南，明确浮动给付范围、最低保障比例的合理区间、以及客户年金变动的年度披露机制。在偿付能力监管中，也应设定保守估值基准，避免浮动机制在极端情景下引发责任低估。

在资产负债管理方面，MGMA 产品责任期长、浮动程度高，建议监管引导建立长周期资产匹配监管指标，例如基于未来现金流贴现后的ΔPV敞口指标，作为风险监测参考。同时，应对部分久期较长、现金流稳定的资产类别（如养老 REITs、基础设施 ABS 等）给予适度资本优惠，引导保险公司配置与年金责任匹配度更高的长期资产，提升责任履约能力。

在风险转移机制方面，建议政策层面支持保险公司与再保险机构合作开发高龄尾部责任

比例再保险或触发型再保条款，用以缓释极端长寿情景下的责任扩张。同时，监管可推动发展与死亡率指数挂钩的金融工具，引导资本市场逐步承接部分人口风险责任。在政策允许范围内，也可探索MGMA责任池的资产证券化试点，为长寿风险的市场化分摊提供制度基础。

此外，应高度重视MGMA产品的客户保护机制。年金浮动结构对多数消费者而言相对陌生，若缺乏足够解释和示例披露，可能引发认知偏差和信任流失。建议对浮动型年金产品实施分级销售适当性管理，并要求产品说明书中列明年金调整路径的示例与不利情景说明，同时建立年度年金调整披露机制，提升产品透明度与客户接受度。鼓励保险公司探索参与式MGMA设计，将浮动机制与客户账户收益挂钩，增强客户的收益体验和风险共担意愿。

最后，应通过监管科技手段推动长寿风险治理能力的系统升级。建议建立监管层级的长寿风险监测与预警系统，将ΔPV、责任现值波动率等指标纳入行业风险评价体系，并通过“监管沙盒”机制支持结构性年金、长寿再保、死亡率互助产品等制度试点。同时，鼓励行业协会牵头构建长寿风险评估实验平台，开展人口预测研究、产品共建与数据开放合作，为制度层面的持续优化提供理论和数据支撑。

综上所述，MGMA产品作为应对长寿风险的重要工具，其大规模推行不仅需要公司内部模型能力与资产配置能力的提升，也依赖于宏观政策层对数据、产品、资产、风险和客户的全方位监管协同。建议监管机构在推动养老保险体系多层次发展过程中，将长寿风险纳入监管视野，构建责任驱动型产品监管与资本制度，并以数据标准、市场机制和客户权益为基础，构建科学、稳健、可持续的长寿风险治理生态。

六、结论与展望

（一）研究结论

本研究聚焦于人口老龄化背景下养老年金产品所面临的长寿风险挑战，基于保险行业真实经验数据与国际权威人口死亡率数据库，系统开展了死亡率建模与预测框架，并围绕养老年金产品的风险敞口评估与综合应对策略展开深入研究，主要结论如下：

第一，保险人群死亡率在性别、年龄和时间维度上呈现显著差异。男性死亡率普遍高于女性，尤其在中老年阶段差异明显；死亡率整体随年龄增长呈指数型上升；随着医疗技术进步和社会保障水平提升，近年死亡率改善趋缓但仍保持稳定。

第二，死亡率预测模型融合与高龄外推策略有效提升了建模精度与生物合理性。基于Lee-Carter与Gompertz-Makeham模型的混合建模策略在中青年阶段预测效果良好，但在外

推至 80 岁以上的高龄阶段时需借助国际高质量死亡率数据进行修正。通过引入国外生命表锚定策略，能够成功解决高龄段死亡率预测偏高的问题，确保全年龄段死亡率曲线的平滑性、生物学合理性及现实适用性。

第三，长期死亡率趋势呈持续下降态势，但改善幅度逐步收敛。预测显示，在未来百年内，保险人口出生时预期寿命稳步上升，男性与女性预期寿命差距持续存在。死亡率改善速度受生物极限、生活方式、医疗进步边际效应等多重因素影响，整体呈“减速而不停止”的长期趋势。

第四，养老年金类产品在静态定价假设下面临显著的长寿风险暴露，且该风险具有性别差异、贴现率敏感性及投保年龄依赖性。长寿风险敞口和风险率在男性、低贴现率及低龄投保人群中表现更为突出，需引起保险公司在产品定价与精算假设设定中的高度关注。

第五，多元化的风险应对策略有助于提升养老年金产品的可持续性与市场竞争力，加强保险公司的长寿风险适应能力。最低保证死亡率年金（MGMA）产品能够通过结合死亡率浮动因子与最低给付保障机制，实现年金给付与死亡率改善之间的动态联动，兼顾精算平衡与客户保障，有效缓释长寿趋势带来的财务冲击。在产品的创新基础上，可进一步构建产品—投资—风险转移—监管协同的综合风险管理框架，推动形成闭环联动的风险管控体系。

（二）研究展望

尽管本研究在死亡率预测、长寿风险量化与管控方面取得了较为系统的成果，但仍存在一定限制与进一步研究空间：

第一，模型进一步深化与参数不确定性建模。当前所建立模型在结构合理性与拟合性能方面表现良好，但对参数不确定性、极端事件扰动、医疗突破等因素的响应能力仍较为有限。未来可考虑引入贝叶斯方法或情景模拟技术，对死亡率路径的不确定区间进行量化。

第二，引入健康状态转移模型与多状态年金产品建模。仅依赖死亡率作为单一指标难以全面刻画老年人群的生命历程。未来可引入多状态 Markov 模型，结合“健康—亚健康—失能—死亡”状态转移机制，更加真实地估计养老支付压力，并探索护理型、分段给付型等年金产品的长寿风险适配性。

第三，长寿风险对冲机制建设。当前寿命衍生工具市场尚不成熟，限制了保险公司在资本市场进行有效的风险转移。未来研究可聚焦于寿命证券化产品设计、区域再保险池建设及政策性长寿保险保障制度的构建，推动长寿风险的金融化管理。

综上所述，长寿风险作为保险业与养老金体系共同面临的挑战，需要多维度、跨学科的

系统性研究与管理手段。本研究的相关成果可以为保险机构优化年金产品结构、强化资产负债管理、制定可持续发展战略提供支持，同时也为推动我国养老保障制度在新时代背景下的高质量发展提供了有益探索与理论基础。

参考文献

[1] Dang L H K, Camarda C G, Ouellette N, et al. The question of the human mortality plateau[J]. Demographic Research, 2023, 48: 321-338.

[2] Gavrilov L A, Gavrilova N S. New trend in old-age mortality: Gompertzialization of mortality trajectory[J]. Gerontology, 2019, 65(5): 451-457.

[3] Bell F C, Miller M L. Life tables for the United States social security area, 1900-2100[M]. Social Security Administration, Office of the Chief Actuary, 2005.

[4] Newman S J. Errors as a primary cause of late-life mortality deceleration and plateaus[J]. PLoS Biology, 2018, 16(12): e2006776.

附录 I：附图[1]

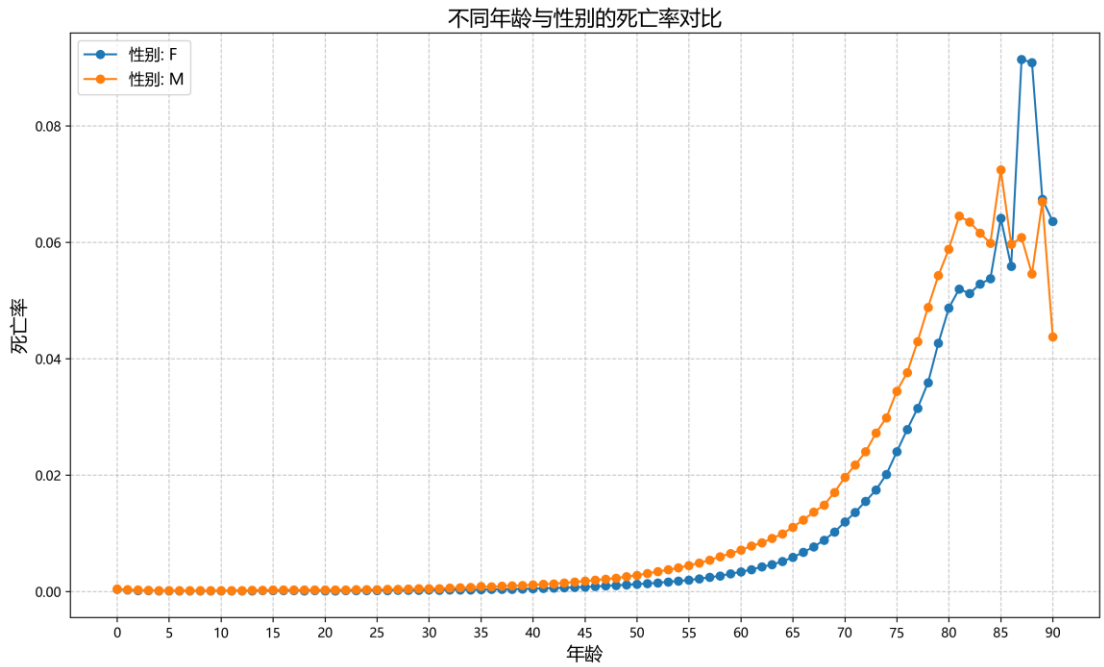

图 24　不同年龄与性别保险人群死亡率对比图

（M 性：男性；F 性：女性）

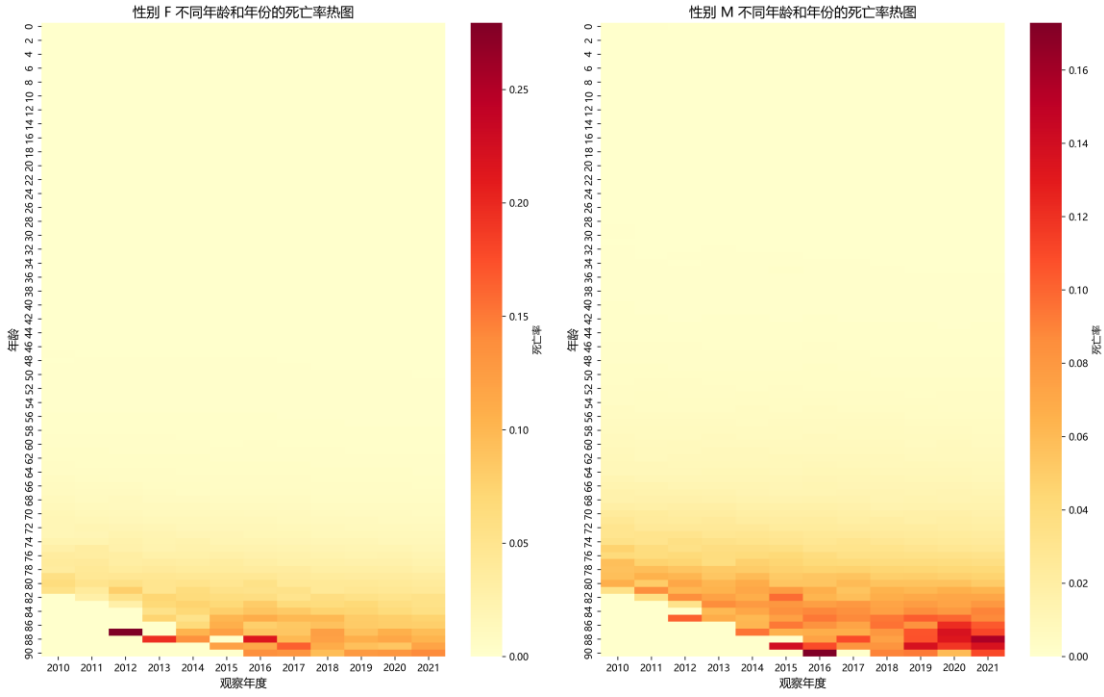

图 25　2010-2021 年不同年龄保险人口死亡率热力图

（M 性：男性；F 性：女性）

[1] 本研究在死亡率建模与预测分析过程中生成了大量图片。为控制报告整体篇幅，正文与附录仅展示关键图片，完整图片及相关资料可通过以下链接查阅：https://arthur-stat.github.io/docs/longevity_risk.zip。

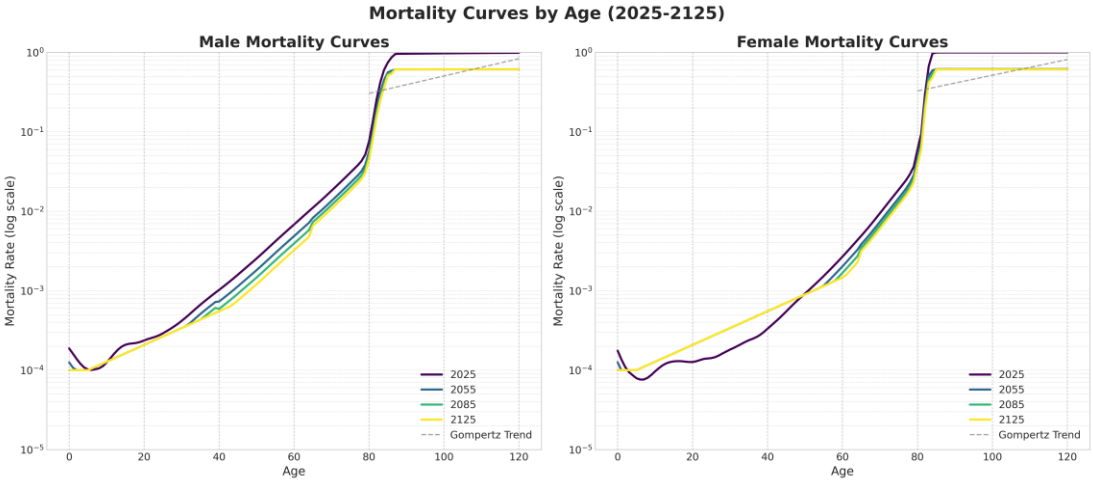

图 26　2025-2125 年全年龄段保险人口死亡率分年份预测曲线图

（左图：男性；右图：女性）

附录 II：附表

表 11　2025 年保险人口死亡率初步预测模型评价指标

性别	Lee-Carter 模型R^2	Lee-Carter 模型 MSE	Gompertz-Makeham 模型R^2
男性	0.6138	2.251832	0.9638
女性	0.5807	2.479653	0.9981

表 12　所使用 HMD 死亡率数据涵盖的国家/地区

序号	国家/地区	代码
1	澳大利亚	AUS
2	奥地利	AUT
3	比利时	BEL
4	保加利亚	BGR
5	白俄罗斯	BLR
6	加拿大	CAN
7	瑞士	CHE
8	智利	CHL
9	捷克	CZE
10	德国东部	DEUTE
11	德国全国	DEUTNP
12	德国西部	DEUTW
13	丹麦	DNK
14	西班牙	ESP
15	爱沙尼亚	EST
16	芬兰	FIN
17	法国海外省	FRACNP
18	法国全国	FRATNP
19	英格兰和威尔士	GBRCENW/GBRTENW
20	英国北爱尔兰	GBR_NIR
21	英国全国	GBR_NP
22	英国苏格兰	GBR_SCO
23	希腊	GRC
24	中国香港	HKG
25	克罗地亚	HRV
26	匈牙利	HUN
27	爱尔兰	IRL
28	冰岛	ISL
29	以色列	ISR
30	意大利	ITA
31	日本	JPN
32	韩国	KOR
33	立陶宛	LTU

34	卢森堡	LUX
35	拉脱维亚	LVA
36	荷兰	NLD
37	挪威	NOR
38	新西兰毛利人	NZL_MA
39	新西兰非毛利人	NZL_NM
40	新西兰全国	NZL_NP
41	波兰	POL
42	葡萄牙	PRT
43	俄罗斯	RUS
44	斯洛伐克	SVK
45	斯洛文尼亚	SVN
46	瑞典	SWE
47	中国台湾	TWN
48	乌克兰	UKR
49	美国	USA

附录 III：代码

本研究所涉及的主要建模与分析工作均采用 Python 编程语言实现。鉴于代码量较大，为控制报告整体篇幅，附录中仅择要展示部分核心代码片段。完整代码可通过作者提供的在线地址查看[2]。

1. 2010-2021 年保险人口死亡率数据分析

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline
from sklearn.linear_model import LinearRegression
import seaborn as sns
from matplotlib.font_manager import FontProperties
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from scipy.optimize import curve_fit


plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Source Han Sans SC', 'Noto Sans CJK SC', 'SimHei']
plt.rcParams['axes.unicode_minus'] = True


# plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
# plt.rcParams['axes.unicode_minus'] = False

plt.rcParams['text.usetex'] = False

# 加载数据
file_path = "【选题 1 数据】某保险产品死亡率.xlsx"
df = pd.read_excel(file_path, sheet_name='data')

# 显示数据基本信息
print("数据基本信息:")
print(df.info())
print("\n 数据前 5 行:")
print(df.head())
print("\n 数据统计摘要:")
print(df.describe())
```

[2] 完整代码访问链接：https://arthur-stat.github.io/docs/longevity_risk.zip。

```
34.
35. # 计算死亡率（理赔数/暴露数）
36. df['死亡率'] = df['理赔数'] / df['暴露数']
37.
38. # 数据概览
39. print(f"\n总数据条目: {len(df)}")
40. print(f"年份范围: {df['观察年度'].min()} - {df['观察年度'].max()}")
41. print(f"年龄范围: {df['到达年龄'].min()} - {df['到达年龄'].max()}")
42. print(f"性别分布: {df['性别'].value_counts().to_dict()}")
43.
44. # 按性别和年龄分组查看死亡率
45. gender_age_mortality = df.groupby(['性别', '到达年龄'])['死亡率
    '].mean().reset_index()
46. print("\n按性别和年龄的平均死亡率(前10行):")
47. print(gender_age_mortality.head(10))
48.
49. # 按观察年度查看死亡率趋势
50. year_mortality = df.groupby(['观察年度'])['死亡率'].mean().reset_index()
51. print("\n按观察年度的平均死亡率趋势:")
52. print(year_mortality)
53.
54. # 按性别、年龄和年份分析死亡率
55. gender_age_year_mortality = df.groupby(['性别', '到达年龄', '观察年度'])['死亡率
    '].mean().reset_index()
56.
57. # 可视化：性别和年龄对死亡率的影响
58. plt.figure(figsize=(14, 8))
59. for gender in df['性别'].unique():
60.     gender_data = gender_age_mortality[gender_age_mortality['性别'] == gender]
61.     plt.plot(gender_data['到达年龄'], gender_data['死亡率
    '], marker='o', label=f"性别: {gender}")
62.
63. plt.title('不同年龄与性别的死亡率对比', fontsize=16)
64. plt.xlabel('年龄', fontsize=14)
65. plt.ylabel('死亡率', fontsize=14)
66. plt.grid(True, linestyle='--', alpha=0.7)
67. plt.legend(fontsize=12)
68. plt.xticks(np.arange(0, df['到达年龄'].max()+1, 5))
69. plt.savefig('gender_age_mortality.png', dpi=300, bbox_inches='tight')
70. plt.tight_layout()
71. plt.close()
72.
73. # 使用对数刻度以便更好地观察
74. plt.figure(figsize=(14, 8))
```

```
for gender in df['性别'].unique():
    gender_data = gender_age_mortality[gender_age_mortality['性别'] == gender]
    plt.semilogy(gender_data['到达年龄'], gender_data['死亡率
'], marker='o', label=f"性别: {gender}")

plt.title('不同年龄与性别的死亡率对比(对数刻度)', fontsize=16)
plt.xlabel('年龄', fontsize=14)
plt.ylabel('死亡率(对数刻度)', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(fontsize=12)
plt.xticks(np.arange(0, df['到达年龄'].max()+1, 5))
plt.savefig('gender_age_mortality_log.png', dpi=300, bbox_inches='tight')
plt.tight_layout()
plt.close()

# 可视化：死亡率随年度的变化
plt.figure(figsize=(12, 7))
plt.plot(year_mortality['观察年度'], year_mortality['死亡率
'], marker='o', linewidth=2)
plt.title('死亡率随年度的变化趋势', fontsize=16)
plt.xlabel('观察年度', fontsize=14)
plt.ylabel('平均死亡率', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(year_mortality['观察年度'])
plt.savefig('year_mortality_trend.png', dpi=300, bbox_inches='tight')
plt.tight_layout()
plt.close()

  # 可视化：热图展示年龄、年份、性别对死亡率的影响
  plt.figure(figsize=(16, 10))
  for i, gender in enumerate(['F', 'M']):
      plt.subplot(1, 2, i+1)
      gender_data = df[df['性别'] == gender].pivot_table(
          index='到达年龄', columns='观察年度', values='死亡率'
      )
      sns.heatmap(gender_data, annot=False, cmap='YlOrRd', cbar_kws={'label':
'死亡率'})
      plt.title(f'性别 {gender} 不同年龄和年份的死亡率热图', fontsize=14)
      plt.xlabel('观察年度', fontsize=12)
      plt.ylabel('年龄', fontsize=12)
  plt.tight_layout()
  plt.savefig('mortality_heatmap.png', dpi=300, bbox_inches='tight')
  plt.close()

```

```
    # 针对研究目标1：不同人群死亡率比较
    # 年龄段死亡率
    age_groups = {
        '0-20岁': (0, 20),
        '21-40岁': (21, 40),
        '41-60岁': (41, 60),
        '61岁以上': (61, 100)
    }

    for gender in ['F', 'M']:
        plt.figure(figsize=(14, 8))
        for name, (start, end) in age_groups.items():
            group_data = df[(df['性别'] == gender) & (df['到达年龄'] >= start) & (df['到达年龄'] <= end)]
            yearly_rate = group_data.groupby('观察年度')['死亡率'].mean()
            plt.plot(yearly_rate.index, yearly_rate.values, marker='o', label=name)

        plt.title(f'性别 {gender} 不同年龄段死亡率随年份变化', fontsize=16)
        plt.xlabel('观察年度', fontsize=14)
        plt.ylabel('平均死亡率', fontsize=14)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.legend(fontsize=12)
        plt.xticks(df['观察年度'].unique())
        plt.savefig(f'age_group_trends_{gender}.png', dpi=300, bbox_inches='tight')
        plt.close()
```

2. HMD 数据分析

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import os
from datetime import datetime
import logging
import re

# 配置日志
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')

```

```
# # 配置中文显示
# plt.rcParams['font.sans-serif'] = ['SimHei']
# plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

class MortalityDataAnalyzer:
    def __init__(self, hmd_dir):
        self.hmd_dir = hmd_dir
        self.data = {}
        self.output_dir = 'international_mortality_analysis/eda_output'
        os.makedirs(self.output_dir, exist_ok=True)
        logging.info(f"初始化分析器，数据目录：{hmd_dir}")

    def load_data(self):
        """加载标准化后的 HMD 死亡率数据（Mx_1x1_std）"""
        logging.info("开始加载标准化数据...")

        if not os.path.exists(self.hmd_dir):
            logging.error(f"数据目录不存在：{self.hmd_dir}")
            return

        for file in os.listdir(self.hmd_dir):
            if file.endswith('.Mx_1x1.txt'):
                match = re.match(r'([A-Z]{3,}(_[A-Z]+)?)(\.Mx_1x1)?\.txt', file)
                if match:
                    country = match.group(1)
                    file_path = os.path.join(self.hmd_dir, file)
                    try:
                        logging.info(f"正在加载 {country} 数据: {file_path}")
                        df = pd.read_csv(file_path, sep='\t', skiprows=2)
                        df.columns = df.columns.str.strip()
                        required_cols = ['Year', 'Age', 'Female', 'Male', 'Total']
                        if all(col in df.columns for col in required_cols):
                            df['Year'] = pd.to_numeric(df['Year'], errors='coerce')
                            df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
                            for col in ['Female', 'Male', 'Total']:
                                df[col] = pd.to_numeric(df[col], errors='coerce')
                            df = df.dropna()
```

```
                            df = df[df['Age'] <= 100]
                            self.data[country] = df
                            logging.info(f"成功加载 {country} 数据，形状: {df.shape}")
                        else:
                            missing_cols = [col for col in required_cols if col not in df.columns]
                            logging.warning(f"{country} 数据缺少必要的列: {missing_cols}")
                    except Exception as e:
                        logging.error(f"加载 {country} 数据时出错: {e}")

        if not self.data:
            logging.error("没有成功加载任何数据！")
        else:
            logging.info(f"数据加载完成！共加载 {len(self.data)} 个国家的数据")

    def calculate_statistics(self):
        """计算基本统计指标"""
        logging.info("开始计算统计指标...")

        if not self.data:
            logging.error("没有数据可供分析！")
            return {}

        stats_results = {}

        for country, df in self.data.items():
            logging.info(f"正在计算 {country} 的统计指标...")

            country_stats = {
                '总体统计': {},
                '年龄组统计': {},
                '时间趋势': {}
            }

            # 1. 总体统计
            for col in ['Female', 'Male', 'Total']:
                rates = df[col].replace(0, np.nan)
                if rates.notna().any():
                    country_stats['总体统计'][col] = {
                        '均值': rates.mean(),
                        '中位数': rates.median(),
                        '标准差': rates.std(),
```

```
                        '最小值': rates.min(),
                        '最大值': rates.max(),
                        '变异系数
    ': rates.std() / rates.mean() if rates.mean() != 0 else np.nan
                    }

                # 2. 年龄组统计
                age_groups = [0, 20, 40, 60, 80]
                for year in df['Year'].unique():
                    year_data = df[df['Year'] == year]
                    year_stats = {}
                    for age in age_groups:
                        age_data = year_data[year_data['Age'] == age]
                        if not age_data.empty:
                            year_stats[age] = {
                                'Female': age_data['Female'].iloc[0],
                                'Male': age_data['Male'].iloc[0],
                                'Total': age_data['Total'].iloc[0]
                            }
                    if year_stats:
                        country_stats['年龄组统计'][year] = year_stats

                # 3. 时间趋势
                for col in ['Female', 'Male', 'Total']:
                    rates = df[col].replace(0, np.nan)
                    if rates.notna().any():
                        # 计算年度变化率
                        changes = rates.pct_change()
                        country_stats['时间趋势'][col] = {
                            '平均年变化率': changes.mean(),
                            '变化率标准差': changes.std(),
                            '最大年增长': changes.max(),
                            '最大年下降': changes.min()
                        }

                stats_results[country] = country_stats
                logging.info(f"{country} 统计指标计算完成")

            return stats_results

        def visualize_trends(self, stats_results):
            """可视化死亡率趋势"""
            logging.info("开始生成可视化图表...")

```

```
138.         if not stats_results:
139.             logging.error("没有统计结果可供可视化！")
140.             return
141.
142.         for country, stats in stats_results.items():
143.             if country not in self.data:
144.                 continue
145.
146.             logging.info(f"正在为 {country} 生成可视化图表...")
147.             df = self.data[country]
148.
149.             # 1. 死亡率随年龄变化趋势
150.             plt.figure(figsize=(12, 6))
151.             years = [1950, 1970, 1990, 2010]
152.             for year in years:
153.                 if year in df['Year'].values:
154.                     year_data = df[df['Year'] == year]
155.                     plt.plot(year_data['Age'], year_data['Total'], marker='o
    ', label=str(year))
156.
157.             plt.title(f'{country} 不同年份死亡率随年龄变化趋势')
158.             plt.xlabel('年龄')
159.             plt.ylabel('死亡率')
160.             plt.legend()
161.             plt.grid(True)
162.             plt.yscale('log')  # 使用对数刻度更好地显示差异
163.             plt.savefig(os.path.join(self.output_dir, f'{country}_age_trends
    .png'))
164.             plt.close()
165.
166.             # 2. 死亡率随时间变化趋势
167.             plt.figure(figsize=(12, 6))
168.             ages = [0, 20, 40, 60, 80]
169.             for age in ages:
170.                 age_data = df[df['Age'] == age]
171.                 if not age_data.empty:
172.                     plt.plot(age_data['Year'], age_data['Total'], label=f'{a
    ge}岁')
173.
174.             plt.title(f'{country} 不同年龄组死亡率随时间变化趋势')
175.             plt.xlabel('年份')
176.             plt.ylabel('死亡率')
177.             plt.legend()
178.             plt.grid(True)
```

```
179.                 plt.yscale('log')
180.                 plt.savefig(os.path.join(self.output_dir, f'{country}_time_trend
    s.png'))
181.                 plt.close()
182.
183.                 # 3. 性别差异对比
184.                 plt.figure(figsize=(12, 6))
185.                 years = [1950, 1970, 1990, 2010]
186.                 for year in years:
187.                     if year in df['Year'].values:
188.                         year_data = df[df['Year'] == year]
189.                         plt.plot(year_data['Age'], year_data['Male'] / year_data
    ['Female'],
190.                                  marker='o', label=str(year))
191.
192.                 plt.title(f'{country} 不同年份男女死亡率比值随年龄变化趋势')
193.                 plt.xlabel('年龄')
194.                 plt.ylabel('男/女死亡率比值')
195.                 plt.legend()
196.                 plt.grid(True)
197.                 plt.savefig(os.path.join(self.output_dir, f'{country}_gender_rat
    io.png'))
198.                 plt.close()
199.
200.                 logging.info(f"{country} 可视化图表生成完成")
201.
202.         def generate_report(self, stats_results):
203.             """生成分析报告"""
204.             logging.info("开始生成分析报告...")
205.
206.             if not stats_results:
207.                 logging.error("没有统计结果可供生成报告！")
208.                 return
209.
210.             report = """# 国际死亡率数据探索性分析报告
211.
212.     ## 1. 数据概览
213.
214.     本报告基于Human Mortality Database (HMD)的数据，对多个国家的死亡率数据进行了探索
    性分析。分析内容包括基本统计指标、时间趋势和年龄特征等。
215.
216.     ## 2. 统计指标分析
217.
218.     """
```

```
219.
220.         for country, stats in stats_results.items():
221.             report += f"\n### {country}\n\n"
222.
223.             # 总体统计
224.             report += "#### 2.1 总体统计\n\n"
225.             report += "| 性别 | 均值 | 中位数 | 标准差 | 最小值 | 最大值 | 变异系数 |\n"
226.             report += "|------|------|--------|--------|--------|--------|--------|\n"
227.
228.             for gender, metrics in stats['总体统计'].items():
229.                 report += f"| {gender} | {metrics['均值']:.6f} | {metrics['中位数']:.6f} | "
230.                 report += f"{metrics['标准差']:.6f} | {metrics['最小值']:.6f} | "
231.                 report += f"{metrics['最大值']:.6f} | {metrics['变异系数']:.6f} |\n"
232.
233.             # 时间趋势
234.             report += "\n#### 2.2 时间趋势\n\n"
235.             report += "| 性别 | 平均年变化率 | 变化率标准差 | 最大年增长 | 最大年下降 |\n"
236.             report += "|------|--------------|--------------|------------|------------|\n"
237.
238.             for gender, metrics in stats['时间趋势'].items():
239.                 report += f"| {gender} | {metrics['平均年变化率']:.6f} | "
240.                 report += f"{metrics['变化率标准差']:.6f} | {metrics['最大年增长']:.6f} | "
241.                 report += f"{metrics['最大年下降']:.6f} |\n"
242.
243.         report += """
244.   ## 3. 可视化分析
245.
246.   通过生成的可视化图表，我们可以观察到：
247.
248.   1. 死亡率随年龄变化趋势：
249.      - 婴幼儿期死亡率较高
250.      - 青少年期死亡率最低
251.      - 老年期死亡率呈指数增长
252.
253.   2. 死亡率随时间变化趋势：
254.      - 整体呈下降趋势
```

```
    - 不同年龄组下降速度不同
    - 近年来下降速度有所减缓

3. 性别差异：
    - 男性死亡率普遍高于女性
    - 性别差异在不同年龄组表现不同
    - 性别差异随时间有所变化

## 4. 主要发现

1. 年龄特征：
    - 死亡率随年龄增长呈指数上升
    - 不同年龄组死亡率变化特征不同
    - 老年人口死亡率变化最为显著

2. 时间特征：
    - 长期来看死亡率呈下降趋势
    - 下降速度在不同时期有所不同
    - 近年来下降速度有所减缓

3. 性别特征：
    - 男性死亡率普遍高于女性
    - 性别差异在不同年龄组表现不同
    - 性别差异随时间有所变化

## 5. 建议

1. 数据收集：
    - 建议收集更多国家的数据
    - 增加更多相关变量
    - 提高数据质量

2. 分析方法：
    - 考虑更多影响因素
    - 使用更复杂的统计模型
    - 加强预测能力

3. 政策建议：
    - 关注老年人口健康
    - 加强医疗体系建设
    - 改善环境质量
"""

```

```
            with open(os.path.join(self.output_dir, 'eda_report.md'), 'w', encoding='utf-8') as f:
                f.write(report)

            logging.info("分析报告生成完成！")

    def main():
        # 初始化分析器，切换到标准化目录
        analyzer = MortalityDataAnalyzer('HMD/death_rates/Mx_1x1_std')
        analyzer.load_data()
        stats_results = analyzer.calculate_statistics()
        analyzer.visualize_trends(stats_results)
        analyzer.generate_report(stats_results)

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

3. 基于 HMD 数据的 Gompertz-Makeham 模型与 Lee-Carter 模型对比分析

```
import pandas as pd
import numpy as np
from scipy.optimize import minimize
from scipy.linalg import svd
import matplotlib.pyplot as plt
import os
import logging
from datetime import datetime
import matplotlib as mpl

# # 配置中文显示
# plt.rcParams['font.sans-serif'] = ['SimHei']
# plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

# 配置日志
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')

class MortalityModelComparison:
    def __init__(self, hmd_dir):
        self.hmd_dir = hmd_dir
        self.data = {}
```

```
25.         self.output_dir = 'international_mortality_analysis/model_comparison_ou
    tput'
26.         os.makedirs(self.output_dir, exist_ok=True)
27.
28.     def load_data(self):
29.         """加载 HMD 死亡率数据"""
30.         logging.info("开始加载数据...")
31.
32.         if not os.path.exists(self.hmd_dir):
33.             logging.error(f"数据目录不存在：{self.hmd_dir}")
34.             return
35.
36.         for file in os.listdir(self.hmd_dir):
37.             if file.endswith('.Mx_1x1.txt'):
38.                 country = file.split('.')[0]
39.                 file_path = os.path.join(self.hmd_dir, file)
40.                 try:
41.                     df = pd.read_csv(file_path, sep='\t', skiprows=2)
42.                     df.columns = df.columns.str.strip()
43.                     required_cols = ['Year', 'Age', 'Female', 'Male', 'Total']
44.                     if all(col in df.columns for col in required_cols):
45.                         df['Year'] = pd.to_numeric(df['Year'], errors='coerce')
46.                         df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
47.                         for col in ['Female', 'Male', 'Total']:
48.                             df[col] = pd.to_numeric(df[col], errors='coerce')
49.                         df = df.dropna()
50.                         df = df[df['Age'] <= 100]
51.                         self.data[country] = df
52.                         logging.info(f"成功加载 {country} 数据")
53.                 except Exception as e:
54.                     logging.error(f"加载 {country} 数据时出错: {e}")
55.
56.     def fit_gompertz_makeham(self, mortality_rates, ages, years):
57.         """拟合 Gompertz-Makeham 模型"""
58.         try:
59.             def objective(params):
60.                 a, b, c, d = params
61.                 # 为每个年龄组计算预测值
62.                 predicted = np.zeros_like(mortality_rates)
63.                 for i, age in enumerate(ages):
64.                     predicted[i] = c + a * np.exp(b * age) + d * np.mean(years)
65.                 return np.sum((mortality_rates - predicted) ** 2)
66.
67.             # 初始参数猜测
```

```
            initial_params = [0.0001, 0.1, 0.0001, 0.0001]
            bounds = [(0, None), (0, None), (0, None), (-0.1, 0.1)]

            result = minimize(objective, initial_params, bounds=bounds, method=
    'L-BFGS-B')
            return result.x
        except Exception as e:
            logging.error(f"Gompertz-Makeham 模型拟合失败: {e}")
            return None

    def fit_lee_carter(self, mortality_matrix):
        """拟合 Lee-Carter 模型"""
        try:
            # 添加小常数避免取对数时出现零值
            mortality_matrix = mortality_matrix + 1e-10

            # 计算年龄和时间效应
            log_mx = np.log(mortality_matrix)
            ax = np.mean(log_mx, axis=1)
            centered_log_mx = log_mx - ax[:, np.newaxis]

            # SVD 分解
            U, S, V = svd(centered_log_mx, full_matrices=False)

            # 提取参数
            bx = U[:, 0]
            kt = S[0] * V[0, :]

            return ax, bx, kt
        except Exception as e:
            logging.error(f"Lee-Carter 模型拟合失败: {e}")
            return None, None, None

    def predict_lee_carter(self, ax, bx, kt, years):
        """使用 Lee-Carter 模型预测死亡率"""
        if ax is None or bx is None or kt is None:
            return None

        try:
            # 预测 kt
            kt_mean = np.mean(kt)
            kt_std = np.std(kt)
            predicted_kt = kt_mean + (years - years[0]) * kt_std

```

```
111.                 # 计算预测死亡率
112.                 predicted_log_mx = ax[:, np.newaxis] + np.outer(bx, predicted_kt
    )
113.                 predicted_mx = np.exp(predicted_log_mx)
114.
115.                 return predicted_mx
116.             except Exception as e:
117.                 logging.error(f"Lee-Carter 模型预测失败: {e}")
118.                 return None
119.
120.         def predict_gompertz_makeham(self, params, ages, years):
121.             """使用 Gompertz-Makeham 模型预测死亡率"""
122.             if params is None:
123.                 return None
124.             try:
125.                 a, b, c, d = params
126.                 predicted = np.zeros((len(ages), len(years)))
127.                 for i, age in enumerate(ages):
128.                     for j, year in enumerate(years):
129.                         predicted[i, j] = c + a * np.exp(b * age) + d * year
130.                 return predicted
131.             except Exception as e:
132.                 logging.error(f"Gompertz-Makeham 模型预测失败: {e}")
133.                 return None
134.
135.         def evaluate_models(self, country, gender='Total'):
136.             """评估两个模型的表现"""
137.             logging.info(f"开始评估 {country} 的模型表现...")
138.
139.             if country not in self.data:
140.                 logging.error(f"找不到 {country} 的数据")
141.                 return None
142.
143.             df = self.data[country]
144.
145.             # 准备数据
146.             years = df['Year'].unique()
147.             ages = df['Age'].unique()
148.
149.             # 创建死亡率矩阵
150.             mortality_matrix = df.pivot(index='Age', columns='Year', values=gend
    er).values
151.
152.             # 分割训练集和验证集
```

```
153.            train_years = years[:-5]  # 使用除最后5年外的所有数据作为训练集
154.            val_years = years[-5:]    # 最后5年作为验证集
155.
156.            train_matrix = mortality_matrix[:, :len(train_years)]
157.            val_matrix = mortality_matrix[:, len(train_years):]
158.
159.            # 拟合Gompertz-Makeham模型
160.            # 使用训练集的平均死亡率
161.            train_mean_rates = np.mean(train_matrix, axis=1)
162.            gompertz_params = self.fit_gompertz_makeham(train_mean_rates, ages,
    train_years)
163.
164.            # 拟合Lee-Carter模型
165.            ax, bx, kt = self.fit_lee_carter(train_matrix)
166.
167.            # 预测
168.            gompertz_pred = self.predict_gompertz_makeham(gompertz_params, ages,
     val_years)
169.            lee_carter_pred = self.predict_lee_carter(ax, bx, kt, val_years)
170.
171.            # 计算评估指标
172.            results = {}
173.
174.            if gompertz_pred is not None:
175.                gompertz_mse = np.mean((val_matrix - gompertz_pred) ** 2)
176.                gompertz_mae = np.mean(np.abs(val_matrix - gompertz_pred))
177.                gompertz_r2 = 1 - np.sum((val_matrix - gompertz_pred) ** 2) / np
    .sum((val_matrix - np.mean(val_matrix)) ** 2)
178.                results['Gompertz-Makeham'] = {
179.                    'MSE': gompertz_mse,
180.                    'MAE': gompertz_mae,
181.                    'R2': gompertz_r2
182.                }
183.
184.            if lee_carter_pred is not None:
185.                lee_carter_mse = np.mean((val_matrix - lee_carter_pred) ** 2)
186.                lee_carter_mae = np.mean(np.abs(val_matrix - lee_carter_pred))
187.                lee_carter_r2 = 1 - np.sum((val_matrix - lee_carter_pred) ** 2)
    / np.sum((val_matrix - np.mean(val_matrix)) ** 2)
188.                results['Lee-Carter'] = {
189.                    'MSE': lee_carter_mse,
190.                    'MAE': lee_carter_mae,
191.                    'R2': lee_carter_r2
192.                }
```

```
193.
194.             # 绘制比较图
195.             self.plot_comparison(country, gender, val_matrix, gompertz_pred, lee
    _carter_pred, val_years, ages)
196.
197.             return results
198.
199.         def plot_comparison(self, country, gender, actual, gompertz_pred, lee_ca
    rter_pred, years, ages):
200.             """绘制模型比较图"""
201.             plt.figure(figsize=(15, 10))
202.
203.             # 选择几个代表性年龄组进行展示
204.             age_groups = [0, 20, 40, 60, 80]
205.             age_indices = [np.where(ages == age)[0][0] for age in age_groups]
206.
207.             for i, age_idx in enumerate(age_indices):
208.                 plt.subplot(len(age_groups), 1, i+1)
209.                 plt.plot(years, actual[age_idx, :], 'k-', label='实际值')
210.
211.                 if gompertz_pred is not None:
212.                     plt.plot(years, gompertz_pred[age_idx, :], 'r--
    ', label='Gompertz-Makeham 预测')
213.                 if lee_carter_pred is not None:
214.                     plt.plot(years, lee_carter_pred[age_idx, :], 'b--
    ', label='Lee-Carter 预测')
215.
216.                 plt.title(f'{country} - {gender} - {ages[age_idx]}岁')
217.                 plt.legend()
218.                 plt.grid(True)
219.
220.             plt.tight_layout()
221.             plt.savefig(os.path.join(self.output_dir, f'{country}_{gender}_model
    _compare.png'))
222.             plt.close()
223.
224.         def generate_report(self, results):
225.             """生成比较报告"""
226.             logging.info("开始生成比较报告...")
227.
228.             timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
229.             report_file = os.path.join(self.output_dir, f'model_comparison_repor
    t_{timestamp}.md')
230.
```

```
231.          with open(report_file, 'w', encoding='utf-8') as f:
232.              f.write("# 死亡率模型比较报告\n\n")
233.
234.              for country, country_results in results.items():
235.                  f.write(f"## {country}\n\n")
236.
237.                  if not country_results:
238.                      f.write("模型评估失败\n\n")
239.                      continue
240.
241.                  f.write("### 评估指标\n\n")
242.                  f.write("| 模型 | MSE | MAE | R² |\n")
243.                  f.write("|------|-----|-----|-----|\n")
244.
245.                  for model_name, metrics in country_results.items():
246.                      f.write(f"| {model_name} | {metrics['MSE']:.6f} | {metri
    cs['MAE']:.6f} | {metrics['R2']:.6f} |\n")
247.
248.                  f.write("\n### 结论\n\n")
249.
250.                  # 比较两个模型的性能
251.                  if 'Gompertz-Makeham' in country_results and 'Lee-
    Carter' in country_results:
252.                      gompertz_mse = country_results['Gompertz-
    Makeham']['MSE']
253.                      lee_carter_mse = country_results['Lee-Carter']['MSE']
254.
255.                      if gompertz_mse < lee_carter_mse:
256.                          f.write(f"在{country}的数据上，Gompertz-Makeham 模型表
    现更好，MSE 降低了{(lee_carter_mse-gompertz_mse)/lee_carter_mse*100:.2f}%\n\n")
257.                      else:
258.                          f.write(f"在{country}的数据上，Lee-Carter 模型表现更
    好，MSE 降低了{(gompertz_mse-lee_carter_mse)/gompertz_mse*100:.2f}%\n\n")
259.
260.                  f.write("\n---\n\n")
261.
262.          logging.info(f"比较报告已生成：{report_file}")
263.          return report_file
264.
265.  def main():
266.      # 初始化比较器
267.      comparator = MortalityModelComparison('HMD/death_rates/Mx_1x1_std')
268.
269.      # 加载数据
```

```
        comparator.load_data()

        # 评估每个国家的模型
        results = {}
        for country in comparator.data.keys():
            results[country] = comparator.evaluate_models(country)

        # 生成报告
        report_file = comparator.generate_report(results)
        logging.info(f"模型比较完成，报告已生成: {report_file}")

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

4. 2025 年保险人口死亡率预测分析（改进前模型）

```
"""
2025 年保险人口死亡率预测系统（改进前）
实现 Lee-Carter 模型 + Gompertz-Makeham 模型 + 曲线修匀
外推至 120 岁
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
from scipy.optimize import curve_fit
from scipy.interpolate import UnivariateSpline, splrep, splev
from scipy.ndimage import gaussian_filter1d
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
import os
warnings.filterwarnings('ignore')

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

class MortalityPredictor:
    def __init__(self, data_path):
        """
        初始化死亡率预测器

        Parameters:
```

```
        data_path: str, Excel 文件路径
        """
        self.data_path = data_path
        self.df = None
        self.mortality_matrix = {}
        self.lee_carter_params = {}
        self.gompertz_params = {}
        self.predictions_2025 = {}
        self.extrapolated_120 = {}

    def load_and_prepare_data(self):
        """加载并预处理数据"""
        print("正在加载和预处理数据...")

        # 读取数据
        self.df = pd.read_excel(self.data_path, sheet_name='data')

        # 计算死亡率
        self.df['死亡率'] = self.df['理赔数'] / self.df['暴露数']

        # 处理异常值：死亡率>0.5 视为异常
        self.df.loc[self.df['死亡率'] > 0.5, '死亡率'] = np.nan

        # 基本统计信息
        print(f"数据形状: {self.df.shape}")
        print(f"年龄范围: {self.df['到达年龄'].min()} - {self.df['到达年龄'].max()}")
        print(f"观察年度: {self.df['观察年度'].min()} - {self.df['观察年度'].max()}")
        print(f"死亡率范围: {self.df['死亡率'].min():.6f} - {self.df['死亡率'].max():.6f}")

        # 为每个性别创建死亡率矩阵
        for gender in ['M', 'F']:
            gender_data = self.df[self.df['性别'] == gender]
            matrix = gender_data.pivot(index='到达年龄', columns='观察年度', values='死亡率')

            # 处理缺失值：线性插值
            matrix = matrix.interpolate(method='linear', axis=1)
            matrix = matrix.interpolate(method='linear', axis=0)

            # 确保没有零值（对数变换需要）
            matrix = matrix.fillna(method='ffill').fillna(method='bfill')
```

```
            matrix[matrix <= 0] = 1e-6

            self.mortality_matrix[gender] = matrix

        print("数据预处理完成！")

    def lee_carter_fit(self, gender):
        """
        Lee-Carter 模型拟合

        Parameters:
        gender: str, 'M' 或 'F'

        Returns:
        dict: 包含 a(x), b(x), k(t)参数
        """
        matrix = self.mortality_matrix[gender].values
        ages = self.mortality_matrix[gender].index.values
        years = self.mortality_matrix[gender].columns.values

        # Step 1: 计算 a(x) - 平均对数死亡率
        log_mort = np.log(matrix)
        ax = np.mean(log_mort, axis=1)

        # Step 2: 中心化矩阵
        centered_matrix = log_mort - ax.reshape(-1, 1)

        # Step 3: SVD 分解
        U, s, Vt = np.linalg.svd(centered_matrix, full_matrices=False)

        # Step 4: 提取第一主成分
           bx = U[:, 0]
           kt = s[0] * Vt[0, :]

           # 标准化以确保可识别性
           # 约束: sum(b(x)) = 1, k(t0) = 0
           bx = bx / np.sum(bx)
           kt = kt - kt[0]

           return {
               'ax': ax,
               'bx': bx,
               'kt': kt,
               'ages': ages,
```

```
            'years': years
        }

    def predict_kt_2025(self, kt_historical, years):
        """
        预测 2025 年的 k(t)值
        使用更保守的方法，限制外推幅度
        """
        # 简单线性趋势拟合
        coeffs = np.polyfit(years, kt_historical, 1)
        slope, intercept = coeffs

        # 预测 2025 年，但限制变化幅度
        kt_2025_raw = slope * 2025 + intercept

        # 保守限制：相比最近 3 年平均值，变化不超过 30%
        recent_kt_mean = np.mean(kt_historical[-3:])
        max_change = recent_kt_mean * 0.3

        if kt_2025_raw > recent_kt_mean + max_change:
            kt_2025 = recent_kt_mean + max_change
            print(f"    k(t)预测值被限制: 原始{kt_2025_raw:.3f} -> 调整
{kt_2025:.3f}")
        elif kt_2025_raw < recent_kt_mean - max_change:
            kt_2025 = recent_kt_mean - max_change
            print(f"    k(t)预测值被限制: 原始{kt_2025_raw:.3f} -> 调整
{kt_2025:.3f}")
        else:
            kt_2025 = kt_2025_raw

        return kt_2025, slope

    def gompertz_makeham(self, x, A, B, C):
        """Gompertz-Makeham 模型函数"""
        return A + B * np.exp(C * x)

    def fit_gompertz_makeham(self, ages, mortality_rates, age_range=(30, 90)
):
        """
        拟合 Gompertz-Makeham 模型
        只对 30-90 岁年龄段拟合
        """
        # 选择拟合年龄范围
        mask = (ages >= age_range[0]) & (ages <= age_range[1])
```

```
154.             fit_ages = ages[mask]
155.             fit_rates = mortality_rates[mask]
156.
157.             # 去除可能的异常值
158.             valid_mask = (fit_rates > 0) & (fit_rates < 1) & np.isfinite(fit_rat
    es)
159.             fit_ages = fit_ages[valid_mask]
160.             fit_rates = fit_rates[valid_mask]
161.
162.             if len(fit_ages) < 10:  # 确保有足够的数据点
163.                 return None
164.
165.             try:
166.                 # 设置合理的参数边界
167.                 bounds = ([0, 1e-6, 0], [0.1, 1, 0.2])
168.                 popt, pcov = curve_fit(
169.                     self.gompertz_makeham,
170.                     fit_ages,
171.                     fit_rates,
172.                     bounds=bounds,
173.                     maxfev=5000
174.                 )
175.
176.                 # 计算拟合质量
177.                 predicted = self.gompertz_makeham(fit_ages, *popt)
178.                 r_squared = 1 - np.sum((fit_rates - predicted) ** 2) / np.sum((f
    it_rates - np.mean(fit_rates)) ** 2)
179.
180.                 return {
181.                     'params': popt,
182.                     'covariance': pcov,
183.                     'r_squared': r_squared,
184.                     'fit_ages': fit_ages,
185.                     'fit_rates': fit_rates
186.                 }
187.             except:
188.                 return None
189.
190.         def whittaker_henderson_smooth(self, y, lambda_param=1000):
191.             """保守的修匀方法，避免负值和过度平滑"""
192.             n = len(y)
193.             if n < 3:
194.                 return y
195.
```

```
            # 使用移动平均进行温和修匀，保持原有形状
            smoothed = np.copy(y)

            # 对于年龄0-10，使用简单移动平均
            for i in range(1, min(10, n-1)):
                window_size = 3
                start_idx = max(0, i - window_size//2)
                end_idx = min(n, i + window_size//2 + 1)
                smoothed[i] = np.mean(y[start_idx:end_idx])

            # 对于年龄10以上，使用加权移动平均，更多保留原始值
            for i in range(10, n-1):
                # 5点加权移动平均，中心点权重更大
                weights = np.array([0.1, 0.2, 0.4, 0.2, 0.1])
                start_idx = max(0, i - 2)
                end_idx = min(n, i + 3)

                if end_idx - start_idx == 5:
                    smoothed[i] = np.average(y[start_idx:end_idx], weights=weigh
    ts)
                else:
                    # 边界情况，使用简单平均
                    smoothed[i] = np.mean(y[start_idx:end_idx])

            # 确保非负性
            smoothed = np.maximum(smoothed, 1e-6)

            # 确保修匀后的值不会偏离原值太多（最多±50%）
            ratio_bounds = (0.5, 1.5)
            for i in range(n):
                ratio = smoothed[i] / y[i] if y[i] > 0 else 1.0
                if ratio < ratio_bounds[0]:
                    smoothed[i] = y[i] * ratio_bounds[0]
                elif ratio > ratio_bounds[1]:
                    smoothed[i] = y[i] * ratio_bounds[1]

            return smoothed

        def predict_2025_mortality(self):
            """预测2025年死亡率"""
            print("正在预测2025年死亡率...")

            for gender in ['M', 'F']:
                print(f"\n处理{gender}性数据...")
```

```
239.
240.                 # Lee-Carter 模型拟合
241.                 lc_params = self.lee_carter_fit(gender)
242.                 self.lee_carter_params[gender] = lc_params
243.
244.                 print(f"  Lee-Carter 参数: k(t)范
    围 {lc_params['kt'].min():.3f} - {lc_params['kt'].max():.3f}")
245.                 print(f"  年龄范
    围: {lc_params['ages'].min()} - {lc_params['ages'].max()}")
246.
247.                 # 预测 2025 年的 k(t)
248.                 kt_2025, trend_slope = self.predict_kt_2025(lc_params['kt'], lc_
    params['years'])
249.                 print(f"  k(2025)预测值: {kt_2025:.3f}, 趋势斜
    率: {trend_slope:.3f}")
250.
251.                 # 计算 2025 年死亡率（Lee-Carter）
252.                 log_mort_2025_lc = lc_params['ax'] + lc_params['bx'] * kt_2025
253.                 mort_2025_lc = np.exp(log_mort_2025_lc)
254.
255.                 print(f"  Lee-Carter 2025 预测死亡率范
    围: {mort_2025_lc.min():.6f} - {mort_2025_lc.max():.6f}")
256.
257.                 # 获取 2021 年数据用于 Gompertz-Makeham 拟合
258.                 mort_2021 = self.mortality_matrix[gender][2021].values
259.                 ages = lc_params['ages']
260.
261.                 print(f"  实际 2021 年死亡率范
    围: {mort_2021.min():.6f} - {mort_2021.max():.6f}")
262.
263.                 # 显示关键年龄的 2021 实际值 vs Lee-Carter 预测值对比
264.                 key_ages = [30, 50, 70, 80, 90]
265.                 print(f"  关键年龄 2021 实际 vs Lee-Carter 2025 预测对比:")
266.                 for key_age in key_ages:
267.                     if key_age in ages:
268.                         idx = np.where(ages == key_age)[0][0]
269.                         real_2021 = mort_2021[idx]
270.                         pred_lc = mort_2025_lc[idx]
271.                         print(f"    {key_age}岁: 实际
    2021={real_2021:.6f}, LC 2025={pred_lc:.6f}, 比值={pred_lc/real_2021:.3f}")
272.
273.                 # Gompertz-Makeham 模型拟合
274.                 gm_result = self.fit_gompertz_makeham(ages, mort_2021)
275.
```

```
276.                 if gm_result is not None:
277.                     self.gompertz_params[gender] = gm_result
278.                     print(f"  Gompertz-Makeham 拟合成
    功, R²={gm_result['r_squared']:.4f}")
279.                     print(f"  GM 参
    数: A={gm_result['params'][0]:.8f}, B={gm_result['params'][1]:.8f}, C={gm_resul
    t['params'][2]:.6f}")
280.
281.                     # 计算 2025 年死亡率（Gompertz-Makeham）
282.                     # 使用趋势调整而不是直接使用 2021 年参数
283.                     A, B, C = gm_result['params']
284.
285.                     # 简单趋势调整：基于死亡率改善趋势
286.                     # 假设死亡率以每年 1-2%的速度改善
287.                     improvement_rate = 0.015  # 1.5%年改善率
288.                     years_ahead = 2025 - 2021
289.                     trend_factor = (1 - improvement_rate) ** years_ahead
290.
291.                     # 调整参数 A 和 B（保持 C 不变，因为它反映衰老速率）
292.                     A_2025 = A * trend_factor
293.                     B_2025 = B * trend_factor
294.
295.                     print(f"  GM 趋势调整: 改善率={improvement_rate:.1%}/年, 调整因
    子={trend_factor:.3f}")
296.                     print(f"  GM 2025 参
    数: A={A_2025:.8f}, B={B_2025:.8f}, C={C:.6f}")
297.
298.                     mort_2025_gm = self.gompertz_makeham(ages, A_2025, B_2025, C
    )
299.
300.                     print(f"  Gompertz-Makeham 2025 预测死亡率范
    围: {mort_2025_gm.min():.6f} - {mort_2025_gm.max():.6f}")
301.
302.                     print(f"  关键年龄 GM 2025 预测:")
303.                     for key_age in key_ages:
304.                         if key_age in ages:
305.                             idx = np.where(ages == key_age)[0][0]
306.                             pred_gm = mort_2025_gm[idx]
307.                             real_2021 = mort_2021[idx]
308.                             print(f"    {key_age}岁: GM 2025={pred_gm:.6f}, vs 实
    际 2021={real_2021:.6f}, 比值={pred_gm/real_2021:.3f}")
309.
310.                     # 组合两种方法的预测结果，使用更平滑的权重过渡
311.                     weights_gm = np.zeros_like(ages, dtype=float)
```

```
                    for i, age in enumerate(ages):
                        if age <= 25:
                            weights_gm[i] = 0.1  # 低年龄更依赖Lee-Carter
                        elif age <= 40:
                            weights_gm[i] = 0.1 + 0.3 * (age - 25) / 15  # 平滑过渡
                        elif age <= 65:
                            weights_gm[i] = 0.4 + 0.3 * (age - 40) / 25  # 继续平滑过渡
                        else:
                            weights_gm[i] = 0.7  # 高年龄更依赖Gompertz-Makeham

                    weights_lc = 1 - weights_gm

                    # 加权组合
                    mort_2025_combined = weights_lc * mort_2025_lc + weights_gm * mort_2025_gm

                    print(f"  组合模型2025预测死亡率范围: {mort_2025_combined.min():.6f} - {mort_2025_combined.max():.6f}")

                    print(f"  关键年龄组合模型预测 (权重GM/LC):")
                    for key_age in key_ages:
                        if key_age in ages:
                            idx = np.where(ages == key_age)[0][0]
                            pred_combined = mort_2025_combined[idx]
                            real_2021 = mort_2021[idx]
                            w_gm = weights_gm[idx]
                            w_lc = weights_lc[idx]
                            print(f"    {key_age}岁: 组合={pred_combined:.6f}, vs实际2021={real_2021:.6f}, 比值={pred_combined/real_2021:.3f}, 权重GM={w_gm:.1f}/LC={w_lc:.1f}")
                else:
                    # 如果Gompertz-Makeham拟合失败，只使用Lee-Carter
                    print(f"  警告: Gompertz-Makeham拟合失败，只使用Lee-Carter")
                    mort_2025_combined = mort_2025_lc
                    self.gompertz_params[gender] = None

                # 修匀处理前记录
                print(f"  修匀前死亡率范围: {mort_2025_combined.min():.6f} - {mort_2025_combined.max():.6f}")

                # 修匀处理
```

```
348.                 mort_2025_smoothed = self.whittaker_henderson_smooth(mort_2025_c
    ombined, lambda_param=1000)
349.
350.                 print(f"  修匀后死亡率范
    围: {mort_2025_smoothed.min():.6f} - {mort_2025_smoothed.max():.6f}")
351.
352.                 # 检查修匀对关键年龄的影响
353.                 print(f"  修匀对关键年龄的影响:")
354.                 for key_age in key_ages:
355.                     if key_age in ages:
356.                         idx = np.where(ages == key_age)[0][0]
357.                         before = mort_2025_combined[idx]
358.                         after = mort_2025_smoothed[idx]
359.                         real_2021 = mort_2021[idx]
360.                         print(f"    {key_age}岁: 修匀前={before:.6f}, 修匀后
    ={after:.6f}, 变化={((after/before-1)*100):.1f}%, vs 实际 2021={real_2021:.6f}")
361.
362.                 # 确保单调性（对于 30 岁以上）
363.                 monotonic_violations = 0
364.                 for i in range(1, len(mort_2025_smoothed)):
365.                     if ages[i] > 30 and mort_2025_smoothed[i] < mort_2025_smooth
    ed[i-1]:
366.                         mort_2025_smoothed[i] = mort_2025_smoothed[i-1] * 1.01
367.                         monotonic_violations += 1
368.
369.                 if monotonic_violations > 0:
370.                     print(f"  单调性修正: {monotonic_violations}个年龄点被调整")
371.
372.                 print(f"  最终预测死亡率范
    围: {mort_2025_smoothed.min():.6f} - {mort_2025_smoothed.max():.6f}")
373.
374.                 self.predictions_2025[gender] = {
375.                     'ages': ages,
376.                     'mortality_lc': mort_2025_lc,
377.                     'mortality_combined': mort_2025_combined,
378.                     'mortality_smoothed': mort_2025_smoothed,
379.                     'kt_2025': kt_2025,
380.                     'trend_slope': trend_slope
381.                 }
382.
383.             print("\n2025 年死亡率预测完成! ")
384.
385.         def extrapolate_to_120(self):
386.             """外推至 120 岁"""
```

```
387.           print("正在外推至 120 岁...")
388.
389.           for gender in ['M', 'F']:
390.               print(f"处理{gender}性外推...")
391.
392.               current_ages = self.predictions_2025[gender]['ages']
393.               current_mortality = self.predictions_2025[gender]['mortality_smo
    othed']
394.
395.               # 创建 120 岁的年龄序列
396.               extended_ages = np.arange(0, 121)
397.               extended_mortality = np.zeros(121)
398.
399.               # 对于现有年龄范围，使用预测值
400.               for i, age in enumerate(extended_ages):
401.                   if age in current_ages:
402.                       idx = np.where(current_ages == age)[0][0]
403.                       extended_mortality[i] = current_mortality[idx]
404.
405.               # 对于 90 岁以上，使用 Gompertz-Makeham 外推
406.               if self.gompertz_params[gender] is not None:
407.                   gm_params = self.gompertz_params[gender]['params']
408.
409.                   # 外推 90-120 岁
410.                   for age in range(91, 121):
411.                       if age in extended_ages:
412.                           idx = age  # 因为 extended_ages 是 0-120
413.                           # 使用 Gompertz-Makeham 模型外推
414.                           extended_mortality[idx] = self.gompertz_makeham(age,
     *gm_params)
415.
416.                           # 应用上限约束（死亡率不超过 80%）
417.                           extended_mortality[idx] = min(extended_mortality[idx
    ], 0.8)
418.               else:
419.                   # 如果没有 Gompertz-Makeham 参数，使用指数外推
420.                   last_valid_age = int(current_ages[-1])
421.                   last_mortality = current_mortality[-1]
422.
423.                   # 估计增长率
424.                   if len(current_mortality) >= 10:
425.                       growth_rate = np.mean(np.diff(np.log(current_mortality[-
    10:])))
426.                   else:
```

```
                    growth_rate = 0.1  # 默认增长率

                for age in range(last_valid_age + 1, 121):
                    years_ahead = age - last_valid_age
                    extended_mortality[age] = last_mortality * np.exp(growth_rate * years_ahead)
                    extended_mortality[age] = min(extended_mortality[age], 0.8)

            # 对于低年龄段（0-现有最小年龄），使用插值
            min_age = int(current_ages[0])
            if min_age > 0:
                # 简单线性插值至0岁
                for age in range(0, min_age):
                    ratio = age / min_age
                    extended_mortality[age] = extended_mortality[min_age] * ratio

            # 最终修匀
            extended_mortality = self.whittaker_henderson_smooth(extended_mortality, lambda_param=500)

            # 确保单调性约束（30岁以上）
            for i in range(31, 121):
                if extended_mortality[i] < extended_mortality[i-1]:
                    extended_mortality[i] = extended_mortality[i-1] * 1.005

            # 确保非负性
            extended_mortality = np.maximum(extended_mortality, 1e-6)

            self.extrapolated_120[gender] = {
                'ages': extended_ages,
                'mortality': extended_mortality
            }

        print("120岁外推完成！")

    def generate_results_table(self):
        """生成结果表格"""
        results = []

        for gender in ['M', 'F']:
            ages = self.extrapolated_120[gender]['ages']
            mortality = self.extrapolated_120[gender]['mortality']
```

```
467.
468.                 for i, age in enumerate(ages):
469.                     results.append({
470.                         '性别': gender,
471.                         '年龄': age,
472.                         '2025 年预测死亡率': mortality[i]
473.                     })
474.
475.             return pd.DataFrame(results)
476.
477.         def create_visualizations(self, save_path="Task 2"):
478.             """创建可视化图表"""
479.             print("正在生成可视化图表...")
480.
481.             # 创建保存目录
482.             os.makedirs(save_path, exist_ok=True)
483.
484.             # 1. 历史趋势分析图
485.             fig, axes = plt.subplots(2, 2, figsize=(15, 12))
486.
487.             for i, gender in enumerate(['M', 'F']):
488.                 # 历史死亡率趋势
489.                 matrix = self.mortality_matrix[gender]
490.
491.                 # 选择几个关键年龄显示趋势
492.                 key_ages = [30, 50, 70, 90]
493.                 for age in key_ages:
494.                     if age in matrix.index:
495.                         values = matrix.loc[age].values
496.                         years = matrix.columns.values
497.                         axes[i, 0].plot(years, values, marker='o', label=f'{age}
    岁', linewidth=2)
498.
499.                 axes[i, 0].set_title(f'{gender}性历史死亡率趋势', fontsize=14)
500.                 axes[i, 0].set_xlabel('年份')
501.                 axes[i, 0].set_ylabel('死亡率')
502.                 axes[i, 0].legend()
503.                 axes[i, 0].grid(True, alpha=0.3)
504.
505.                 # Lee-Carter 模型参数
506.                 if gender in self.lee_carter_params:
507.                     lc_params = self.lee_carter_params[gender]
508.
509.                     # b(x)参数图
```

```
                axes[i, 1].plot(lc_params['ages'], lc_params['bx'], 'b-
', linewidth=2)
                axes[i, 1].set_title(f'{gender}性 Lee-Carter 模型 b(x)参数
', fontsize=14)
                axes[i, 1].set_xlabel('年龄')
                axes[i, 1].set_ylabel('b(x)')
                axes[i, 1].grid(True, alpha=0.3)

        plt.tight_layout()
        plt.savefig(f'{save_path}/historical_analysis.png', dpi=300, bbox_in
ches='tight')
        plt.close()

        # 2. 2025 年预测结果图
        fig, axes = plt.subplots(1, 2, figsize=(15, 6))

        for i, gender in enumerate(['M', 'F']):
            pred_data = self.predictions_2025[gender]

            axes[i].plot(pred_data['ages'], pred_data['mortality_lc'],
                         'b--', label='Lee-Carter 预测
', linewidth=2, alpha=0.7)
            axes[i].plot(pred_data['ages'], pred_data['mortality_combined'],

                         'g:', label='组合模型预测', linewidth=2, alpha=0.7)
            axes[i].plot(pred_data['ages'], pred_data['mortality_smoothed'],

                         'r-', label='修匀后预测', linewidth=3)

            # 添加历史数据对比
            hist_2021 = self.mortality_matrix[gender][2021]
            axes[i].plot(hist_2021.index, hist_2021.values,
                         'ko-', label='2021 年实际', alpha=0.5, markersize=3)

            axes[i].set_title(f'{gender}性 2025 年死亡率预测', fontsize=14)
            axes[i].set_xlabel('年龄')
            axes[i].set_ylabel('死亡率')
            axes[i].set_yscale('log')
            axes[i].legend()
            axes[i].grid(True, alpha=0.3)

        plt.tight_layout()
        plt.savefig(f'{save_path}/prediction_2025.png', dpi=300, bbox_inches
='tight')
```

```
            plt.close()

            # 3. 120 岁外推结果图
            fig, axes = plt.subplots(1, 2, figsize=(15, 6))

            for i, gender in enumerate(['M', 'F']):
                ext_data = self.extrapolated_120[gender]

                axes[i].plot(ext_data['ages'], ext_data['mortality'],
                             'r-', linewidth=3, label='外推至 120 岁')

                # 标记 90 岁分界线
                axes[i].axvline(x=90, color='blue', linestyle='--
    ', alpha=0.7, label='原始数据边界')

                axes[i].set_title(f'{gender}性死亡率外推至 120 岁', fontsize=14)
                axes[i].set_xlabel('年龄')
                axes[i].set_ylabel('死亡率')
                axes[i].set_yscale('log')
                axes[i].legend()
                axes[i].grid(True, alpha=0.3)

                # 添加年龄段标注
                axes[i].text(45, ext_data['mortality'][45], '成年期
    ', fontsize=10, alpha=0.7)
                axes[i].text(75, ext_data['mortality'][75], '老年期
    ', fontsize=10, alpha=0.7)
                axes[i].text(105, ext_data['mortality'][105], '外推区间
    ', fontsize=10, alpha=0.7)

            plt.tight_layout()
            plt.savefig(f'{save_path}/extrapolation_120.png', dpi=300, bbox_inch
    es='tight')
            plt.close()

            # 4. 模型比较图
            fig, axes = plt.subplots(2, 1, figsize=(12, 10))

            for i, gender in enumerate(['M', 'F']):
                # Gompertz-Makeham 拟合效果
                if self.gompertz_params[gender] is not None:
                    gm_data = self.gompertz_params[gender]
                    fit_ages = gm_data['fit_ages']
                    fit_rates = gm_data['fit_rates']
```

```
                predicted_rates = self.gompertz_makeham(fit_ages, *gm_data['params'])

                axes[i].scatter(fit_ages, fit_rates, alpha=0.6, label='实际数据', s=30)
                axes[i].plot(fit_ages, predicted_rates, 'r-',
                             label=f'Gompertz-Makeham 拟合 (R²={gm_data["r_squared"]:.3f})', linewidth=2)

                axes[i].set_title(f'{gender}性 Gompertz-Makeham 模型拟合效果', fontsize=14)
                axes[i].set_xlabel('年龄')
                axes[i].set_ylabel('死亡率')
                axes[i].set_yscale('log')
                axes[i].legend()
                axes[i].grid(True, alpha=0.3)

        plt.tight_layout()
        plt.savefig(f'{save_path}/model_fitting.png', dpi=300, bbox_inches='tight')
        plt.close()

        print("可视化图表生成完成！")

    def run_complete_analysis(self):
        """运行完整分析流程"""
        print("开始完整的死亡率预测分析...")
        print("="*60)

        # 1. 数据加载和预处理
        self.load_and_prepare_data()

        # 2. 2025 年预测
        self.predict_2025_mortality()

        # 3. 120 岁外推
        self.extrapolate_to_120()

        # 4. 生成结果表格
        results_df = self.generate_results_table()

        # 5. 创建可视化
        self.create_visualizations()

```

```
            # 6. 保存结果
            results_df.to_csv('Task 2/mortality_predictions_2025.csv', index=False, encoding='utf-8-sig')

            print("="*60)
            print("分析完成！结果已保存到 Task 2 文件夹")

            return results_df


    if __name__ == "__main__":
        # 使用示例
        predictor = MortalityPredictor("【选题 1 数据】某保险产品死亡率.xlsx")
        results = predictor.run_complete_analysis()

        # 显示部分结果
        print("\n 预测结果示例:")
        print(results.head(20))
```

5. 2025 年保险人口死亡率预测分析（改进后模型）

```
"""
2025 年保险人口死亡率预测系统（改进后）
实现 Lee-Carter 模型 + Gompertz-Makeham 模型 + 曲线修匀
外推至 120 岁
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
from scipy.optimize import curve_fit
from scipy.interpolate import UnivariateSpline, splrep, splev
from scipy.ndimage import gaussian_filter1d
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
import os
warnings.filterwarnings('ignore')

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

class MortalityPredictor:
```

```
    def __init__(self, data_path):
        """
        初始化死亡率预测器

        Parameters:
        data_path: str, Excel 文件路径
        """
        self.data_path = data_path
        self.df = None
        self.mortality_matrix = {}
        self.lee_carter_params = {}
        self.gompertz_params = {}
        self.predictions_2025 = {}
        self.extrapolated_120 = {}

    def load_and_prepare_data(self):
        """加载并预处理数据"""
        print("正在加载和预处理数据...")

        # 读取数据
        self.df = pd.read_excel(self.data_path, sheet_name='data')

        # 计算死亡率
        self.df['死亡率'] = self.df['理赔数'] / self.df['暴露数']

        # 处理异常值：死亡率>0.5 视为异常
        self.df.loc[self.df['死亡率'] > 0.5, '死亡率'] = np.nan

        # 基本统计信息
        print(f"数据形状: {self.df.shape}")
        print(f"年龄范围: {self.df['到达年龄'].min()} - {self.df['到达年龄
'].max()}")
        print(f"观察年度: {self.df['观察年度'].min()} - {self.df['观察年度
'].max()}")
        print(f"死亡率范围: {self.df['死亡率'].min():.6f} - {self.df['死亡率
'].max():.6f}")

        # 为每个性别创建死亡率矩阵
        for gender in ['M', 'F']:
            gender_data = self.df[self.df['性别'] == gender]
            matrix = gender_data.pivot(index='到达年龄', columns='观察年度
', values='死亡率')

            # 处理缺失值：线性插值
```

```
            matrix = matrix.interpolate(method='linear', axis=1)
            matrix = matrix.interpolate(method='linear', axis=0)

            # 确保没有零值（对数变换需要）
            matrix = matrix.fillna(method='ffill').fillna(method='bfill')
            matrix[matrix <= 0] = 1e-6

            self.mortality_matrix[gender] = matrix

        print("数据预处理完成！")

    def lee_carter_fit(self, gender):
        """
        Lee-Carter 模型拟合

        Parameters:
        gender: str, 'M' 或 'F'

        Returns:
        dict: 包含 a(x), b(x), k(t)参数
        """
        matrix = self.mortality_matrix[gender].values
        ages = self.mortality_matrix[gender].index.values
        years = self.mortality_matrix[gender].columns.values

        # Step 1: 计算a(x) - 平均对数死亡率
        log_mort = np.log(matrix)
        ax = np.mean(log_mort, axis=1)

        # Step 2: 中心化矩阵
        centered_matrix = log_mort - ax.reshape(-1, 1)

        # Step 3: SVD 分解
        U, s, Vt = np.linalg.svd(centered_matrix, full_matrices=False)

        # Step 4: 提取第一主成分
           bx = U[:, 0]
           kt = s[0] * Vt[0, :]

           # 标准化以确保可识别性
           # 约束: sum(b(x)) = 1, k(t0) = 0
           bx = bx / np.sum(bx)
           kt = kt - kt[0]

```

```
            return {
                'ax': ax,
                'bx': bx,
                'kt': kt,
                'ages': ages,
                'years': years
            }

        def predict_kt_2025(self, kt_historical, years):
            """
            预测 2025 年的 k(t)值
            使用更保守的方法，限制外推幅度
            """
            # 简单线性趋势拟合
            coeffs = np.polyfit(years, kt_historical, 1)
            slope, intercept = coeffs

            # 预测 2025 年，但限制变化幅度
            kt_2025_raw = slope * 2025 + intercept

            # 保守限制：相比最近 3 年平均值，变化不超过 30%
            recent_kt_mean = np.mean(kt_historical[-3:])
            max_change = recent_kt_mean * 0.3

            if kt_2025_raw > recent_kt_mean + max_change:
                kt_2025 = recent_kt_mean + max_change
                print(f"    k(t)预测值被限制: 原始{kt_2025_raw:.3f} -> 调整
    {kt_2025:.3f}")
            elif kt_2025_raw < recent_kt_mean - max_change:
                kt_2025 = recent_kt_mean - max_change
                print(f"    k(t)预测值被限制: 原始{kt_2025_raw:.3f} -> 调整
    {kt_2025:.3f}")
            else:
                kt_2025 = kt_2025_raw

            return kt_2025, slope

        def gompertz_makeham(self, x, A, B, C):
            """Gompertz-Makeham 模型函数"""
            return A + B * np.exp(C * x)

        def fit_gompertz_makeham(self, ages, mortality_rates, age_range=(30, 90)
    ):
            """
```

```
149.            拟合 Gompertz-Makeham 模型
150.            只对 30-90 岁年龄段拟合
151.            """
152.            # 选择拟合年龄范围
153.            mask = (ages >= age_range[0]) & (ages <= age_range[1])
154.            fit_ages = ages[mask]
155.            fit_rates = mortality_rates[mask]
156.
157.            # 去除可能的异常值
158.            valid_mask = (fit_rates > 0) & (fit_rates < 1) & np.isfinite(fit_rat
    es)
159.            fit_ages = fit_ages[valid_mask]
160.            fit_rates = fit_rates[valid_mask]
161.
162.            if len(fit_ages) < 10:  # 确保有足够的数据点
163.                return None
164.
165.            try:
166.                # 设置合理的参数边界
167.                bounds = ([0, 1e-6, 0], [0.1, 1, 0.2])
168.                popt, pcov = curve_fit(
169.                    self.gompertz_makeham,
170.                    fit_ages,
171.                    fit_rates,
172.                    bounds=bounds,
173.                    maxfev=5000
174.                )
175.
176.                # 计算拟合质量
177.                predicted = self.gompertz_makeham(fit_ages, *popt)
178.                r_squared = 1 - np.sum((fit_rates - predicted) ** 2) / np.sum((f
    it_rates - np.mean(fit_rates)) ** 2)
179.
180.                return {
181.                    'params': popt,
182.                    'covariance': pcov,
183.                    'r_squared': r_squared,
184.                    'fit_ages': fit_ages,
185.                    'fit_rates': fit_rates
186.                }
187.            except:
188.                return None
189.
190.        def whittaker_henderson_smooth(self, y, lambda_param=1000):
```

```
191.             """保守的修匀方法，避免负值和过度平滑"""
192.             n = len(y)
193.             if n < 3:
194.                 return y
195.
196.             # 使用移动平均进行温和修匀，保持原有形状
197.             smoothed = np.copy(y)
198.
199.             # 对于年龄0-10，使用简单移动平均
200.             for i in range(1, min(10, n-1)):
201.                 window_size = 3
202.                 start_idx = max(0, i - window_size//2)
203.                 end_idx = min(n, i + window_size//2 + 1)
204.                 smoothed[i] = np.mean(y[start_idx:end_idx])
205.
206.             # 对于年龄10以上，使用加权移动平均，更多保留原始值
207.             for i in range(10, n-1):
208.                 # 5点加权移动平均，中心点权重更大
209.                 weights = np.array([0.1, 0.2, 0.4, 0.2, 0.1])
210.                 start_idx = max(0, i - 2)
211.                 end_idx = min(n, i + 3)
212.
213.                 if end_idx - start_idx == 5:
214.                     smoothed[i] = np.average(y[start_idx:end_idx], weights=weigh
    ts)
215.                 else:
216.                     # 边界情况，使用简单平均
217.                     smoothed[i] = np.mean(y[start_idx:end_idx])
218.
219.             # 确保非负性
220.             smoothed = np.maximum(smoothed, 1e-6)
221.
222.             # 确保修匀后的值不会偏离原值太多（最多±50%）
223.             ratio_bounds = (0.5, 1.5)
224.             for i in range(n):
225.                 ratio = smoothed[i] / y[i] if y[i] > 0 else 1.0
226.                 if ratio < ratio_bounds[0]:
227.                     smoothed[i] = y[i] * ratio_bounds[0]
228.                 elif ratio > ratio_bounds[1]:
229.                     smoothed[i] = y[i] * ratio_bounds[1]
230.
231.             return smoothed
232.
233.         def predict_2025_mortality(self):
```

```
            """预测 2025 年死亡率"""
            print("正在预测 2025 年死亡率...")

            for gender in ['M', 'F']:
                print(f"\n 处理{gender}性数据...")

                # Lee-Carter 模型拟合
                lc_params = self.lee_carter_fit(gender)
                self.lee_carter_params[gender] = lc_params

                print(f"  Lee-Carter 参数: k(t)范
    围 {lc_params['kt'].min():.3f} - {lc_params['kt'].max():.3f}")
                print(f"  年龄范
    围: {lc_params['ages'].min()} - {lc_params['ages'].max()}")

                # 预测 2025 年的 k(t)
                kt_2025, trend_slope = self.predict_kt_2025(lc_params['kt'], lc_
    params['years'])
                print(f"  k(2025)预测值: {kt_2025:.3f}, 趋势斜
    率: {trend_slope:.3f}")

                # 计算 2025 年死亡率（Lee-Carter）
                log_mort_2025_lc = lc_params['ax'] + lc_params['bx'] * kt_2025
                mort_2025_lc = np.exp(log_mort_2025_lc)

                print(f"  Lee-Carter 2025 预测死亡率范
    围: {mort_2025_lc.min():.6f} - {mort_2025_lc.max():.6f}")

                # 获取 2021 年数据用于 Gompertz-Makeham 拟合
                mort_2021 = self.mortality_matrix[gender][2021].values
                ages = lc_params['ages']

                print(f"  实际 2021 年死亡率范
    围: {mort_2021.min():.6f} - {mort_2021.max():.6f}")

                # 显示关键年龄的 2021 实际值 vs Lee-Carter 预测值对比
                key_ages = [30, 50, 70, 80, 90]
                print(f"  关键年龄 2021 实际 vs Lee-Carter 2025 预测对比:")
                for key_age in key_ages:
                    if key_age in ages:
                        idx = np.where(ages == key_age)[0][0]
                        real_2021 = mort_2021[idx]
                        pred_lc = mort_2025_lc[idx]
```

```
                    print(f"    {key_age}岁: 实际2021={real_2021:.6f}, LC 2025={pred_lc:.6f}, 比值={pred_lc/real_2021:.3f}")

                # Gompertz-Makeham 模型拟合
                gm_result = self.fit_gompertz_makeham(ages, mort_2021)

                if gm_result is not None:
                    self.gompertz_params[gender] = gm_result
                    print(f"  Gompertz-Makeham 拟合成功, R²={gm_result['r_squared']:.4f}")
                    print(f"  GM 参数: A={gm_result['params'][0]:.8f}, B={gm_result['params'][1]:.8f}, C={gm_result['params'][2]:.6f}")

                    # 计算 2025 年死亡率（Gompertz-Makeham）
                    # 使用趋势调整而不是直接使用 2021 年参数
                    A, B, C = gm_result['params']

                    # 简单趋势调整：基于死亡率改善趋势
                    # 假设死亡率以每年 1-2% 的速度改善
                    improvement_rate = 0.015  # 1.5% 年改善率
                    years_ahead = 2025 - 2021
                    trend_factor = (1 - improvement_rate) ** years_ahead

                    # 调整参数 A 和 B（保持 C 不变，因为它反映衰老速率）
                    A_2025 = A * trend_factor
                    B_2025 = B * trend_factor

                    print(f"  GM 趋势调整: 改善率={improvement_rate:.1%}/年, 调整因子={trend_factor:.3f}")
                    print(f"  GM 2025 参数: A={A_2025:.8f}, B={B_2025:.8f}, C={C:.6f}")

                    mort_2025_gm = self.gompertz_makeham(ages, A_2025, B_2025, C)

                    print(f"  Gompertz-Makeham 2025 预测死亡率范围: {mort_2025_gm.min():.6f} - {mort_2025_gm.max():.6f}")

                    print(f"  关键年龄 GM 2025 预测:")
                    for key_age in key_ages:
                        if key_age in ages:
                            idx = np.where(ages == key_age)[0][0]
                            pred_gm = mort_2025_gm[idx]
```

```
                            real_2021 = mort_2021[idx]
                            print(f"    {key_age}岁: GM 2025={pred_gm:.6f}, vs 实际 2021={real_2021:.6f}, 比值={pred_gm/real_2021:.3f}")

                    # 组合两种方法的预测结果，使用更平滑的权重过渡
                    weights_gm = np.zeros_like(ages, dtype=float)
                    for i, age in enumerate(ages):
                        if age <= 25:
                            weights_gm[i] = 0.1  # 低年龄更依赖Lee-Carter
                        elif age <= 40:
                            weights_gm[i] = 0.1 + 0.3 * (age - 25) / 15  # 平滑过渡
                        elif age <= 65:
                            weights_gm[i] = 0.4 + 0.3 * (age - 40) / 25  # 继续平滑过渡
                        else:
                            weights_gm[i] = 0.7  # 高年龄更依赖Gompertz-Makeham

                    weights_lc = 1 - weights_gm

                    # 加权组合
                    mort_2025_combined = weights_lc * mort_2025_lc + weights_gm * mort_2025_gm

                    print(f"  组合模型 2025 预测死亡率范围: {mort_2025_combined.min():.6f} - {mort_2025_combined.max():.6f}")

                    print(f"  关键年龄组合模型预测 (权重 GM/LC):")
                    for key_age in key_ages:
                        if key_age in ages:
                            idx = np.where(ages == key_age)[0][0]
                            pred_combined = mort_2025_combined[idx]
                            real_2021 = mort_2021[idx]
                            w_gm = weights_gm[idx]
                            w_lc = weights_lc[idx]
                            print(f"    {key_age}岁: 组合={pred_combined:.6f}, vs 实际 2021={real_2021:.6f}, 比值={pred_combined/real_2021:.3f}, 权重 GM={w_gm:.1f}/LC={w_lc:.1f}")
                else:
                    # 如果Gompertz-Makeham拟合失败，只使用Lee-Carter
                    print(f"  警告: Gompertz-Makeham 拟合失败，只使用 Lee-Carter")
                    mort_2025_combined = mort_2025_lc
                    self.gompertz_params[gender] = None

```

```
344.                 # 修匀处理前记录
345.                 print(f"  修匀前死亡率范
    围: {mort_2025_combined.min():.6f} - {mort_2025_combined.max():.6f}")
346.
347.                 # 修匀处理
348.                 mort_2025_smoothed = self.whittaker_henderson_smooth(mort_2025_c
    ombined, lambda_param=1000)
349.
350.                 print(f"  修匀后死亡率范
    围: {mort_2025_smoothed.min():.6f} - {mort_2025_smoothed.max():.6f}")
351.
352.                 # 检查修匀对关键年龄的影响
353.                 print(f"  修匀对关键年龄的影响:")
354.                 for key_age in key_ages:
355.                     if key_age in ages:
356.                         idx = np.where(ages == key_age)[0][0]
357.                         before = mort_2025_combined[idx]
358.                         after = mort_2025_smoothed[idx]
359.                         real_2021 = mort_2021[idx]
360.                         print(f"    {key_age}岁: 修匀前={before:.6f}, 修匀后
    ={after:.6f}, 变化={((after/before-1)*100):.1f}%, vs 实际 2021={real_2021:.6f}")
361.
362.                 # 确保单调性（对于 30 岁以上）
363.                 monotonic_violations = 0
364.                 for i in range(1, len(mort_2025_smoothed)):
365.                     if ages[i] > 30 and mort_2025_smoothed[i] < mort_2025_smooth
    ed[i-1]:
366.                         mort_2025_smoothed[i] = mort_2025_smoothed[i-1] * 1.01
367.                         monotonic_violations += 1
368.
369.                 if monotonic_violations > 0:
370.                     print(f"  单调性修正: {monotonic_violations}个年龄点被调整")
371.
372.                 print(f"  最终预测死亡率范
    围: {mort_2025_smoothed.min():.6f} - {mort_2025_smoothed.max():.6f}")
373.
374.                 self.predictions_2025[gender] = {
375.                     'ages': ages,
376.                     'mortality_lc': mort_2025_lc,
377.                     'mortality_combined': mort_2025_combined,
378.                     'mortality_smoothed': mort_2025_smoothed,
379.                     'kt_2025': kt_2025,
380.                     'trend_slope': trend_slope
381.                 }
```

```
382.
383.             print("\n2025年死亡率预测完成！")
384.
385.         def adjust_80plus_with_international_data(self, intl_data_path='国外数据
    '):
386.             """
387.             使用国际数据调整80岁以上的死亡率预测
388.
389.             基本思路：
390.             1. 读取国际死亡率数据
391.             2. 寻找与我们的数据在80岁之前模式相似的国家
392.             3. 使用这些相似国家80岁以上的死亡率模式来调整我们的预测
393.             """
394.             print("\n使用国际数据调整80岁以上的死亡率...")
395.
396.             # 国际数据目录
397.             death_rates_path = os.path.join(intl_data_path, 'death_rates', 'Mx_1
    x1')
398.             if not os.path.exists(death_rates_path):
399.                 print(f"错误：找不到国际数据目录 {death_rates_path}")
400.                 return False
401.
402.             # 读取所有可用的国家数据
403.             country_files = [f for f in os.listdir(death_rates_path) if f.endswi
    th('.txt')]
404.             print(f"找到{len(country_files)}个国家的死亡率数据")
405.
406.             # 保存国际数据
407.             international_data = {}
408.
409.             # 读取国际死亡率数据
410.             for gender in ['M', 'F']:
411.                 international_data[gender] = {}
412.                 gender_key = 'Male' if gender == 'M' else 'Female'
413.
414.                 # 读取每个国家的数据
415.                 for country_file in country_files:
416.                     country_code = country_file.split('.')[0]
417.                     file_path = os.path.join(death_rates_path, country_file)
418.
419.                     try:
420.                         # 读取文件（跳过前两行）
421.                         with open(file_path, 'r', encoding='utf-8') as f:
422.                             lines = f.readlines()[2:]  # 跳过前两行
```

```
423.
424.                         # 提取列标题
425.                         header = lines[0].strip().split()
426.
427.                         # 确定性别列的索引
428.                         gender_idx = header.index(gender_key) if gender_key in h
    eader else None
429.
430.                         if gender_idx is None:
431.                             continue
432.
433.                         # 提取数据
434.                         country_data = {'ages': [], 'rates': [], 'years': []}
435.                         recent_year_data = {}  # 按年龄存储最新年份的数据
436.
437.                         for line in lines[1:]:
438.                             parts = line.strip().split()
439.                             if len(parts) >= 4:  # 确保有足够的列
440.                                 try:
441.                                     year = int(parts[0])
442.                                     age = int(parts[1])
443.                                     rate = float(parts[gender_idx])
444.
445.                                     # 只保存0-110岁的数据
446.                                     if 0 <= age <= 110 and rate > 0:
447.                                         country_data['ages'].append(age)
448.                                         country_data['rates'].append(rate)
449.                                         country_data['years'].append(year)
450.
451.                                         # 更新最新年份的数据
452.                                         if age not in recent_year_data or year >
     recent_year_data[age]['year']:
453.                                             recent_year_data[age] = {'rate': rat
    e, 'year': year}
454.                                 except:
455.                                     continue
456.
457.                         # 如果数据太少，跳过
458.                         if len(country_data['ages']) < 50:
459.                             continue
460.
461.                         # 提取所有年龄的最新数据
462.                         latest_ages = []
463.                         latest_rates = []
```

```
464.
465.                         for age in range(0, 111):
466.                             if age in recent_year_data:
467.                                 latest_ages.append(age)
468.                                 latest_rates.append(recent_year_data[age]['rate'
    ])
469.
470.                         if len(latest_ages) > 30:  # 确保有足够的数据点
471.                             international_data[gender][country_code] = {
472.                                 'ages': np.array(latest_ages),
473.                                 'rates': np.array(latest_rates)
474.                             }
475.                             print(f"  导入{country_code} {gender_key}数
    据: {len(latest_ages)}个年龄点")
476.                     except Exception as e:
477.                         print(f"  读取{country_file}时出错: {str(e)}")
478.
479.                 print(f"  成功导入{len(international_data[gender])}个国家的
    {gender_key}数据")
480.
481.             # 对每个性别，寻找最相似的国家并调整80岁以上的死亡率
482.             for gender in ['M', 'F']:
483.                 if gender not in self.predictions_2025:
484.                     continue
485.
486.                 pred_ages = self.predictions_2025[gender]['ages']
487.                 pred_mort = self.predictions_2025[gender]['mortality_smoothed']
488.
489.                 print(f"\n{gender}性 80 岁以上数据调整:")
490.
491.                 # 计算每个国家在80岁以前的相似度
492.                 similarity_scores = {}
493.
494.                 for country, data in international_data[gender].items():
495.                     country_ages = data['ages']
496.                     country_rates = data['rates']
497.
498.                     # 计算40-80岁年龄段的相似度（使用对数尺度）
499.                     err_sum = 0
500.                     count = 0
501.
502.                     for age in range(40, 81):
503.                         if age in pred_ages and age in country_ages:
504.                             idx_pred = np.where(pred_ages == age)[0][0]
```

```
                            idx_country = np.where(country_ages == age)[0][0]

                            pred_rate = pred_mort[idx_pred]
                            country_rate = country_rates[idx_country]

                            if pred_rate > 0 and country_rate > 0:
                                # 计算对数空间中的差距
                                log_diff = np.abs(np.log(pred_rate) - np.log(cou
    ntry_rate))
                                err_sum += log_diff
                                count += 1

                    if count > 0:
                        avg_err = err_sum / count
                        similarity_scores[country] = {
                            'score': np.exp(-avg_err),  # 转换为相似度分数（0-1 之
    间）
                            'error': avg_err
                        }

                # 根据相似度排序
                sorted_countries = sorted(
                    similarity_scores.items(),
                    key=lambda x: x[1]['score'],
                    reverse=True
                )

                # 选择前 5 名最相似的国家
                top_countries = [c[0] for c in sorted_countries[:5]]
                print(f"  最相似的 5 个国家: {', '.join(top_countries)}")

                for c in sorted_countries[:5]:
                    print(f"    {c[0]}: 相似度={c[1]['score']:.4f}, 误差
    ={c[1]['error']:.4f}")

                # 使用选定的国家来调整 80 岁以上的死亡率
                if len(top_countries) > 0:
                    # 提取 80 岁以上的数据
                    ages_80plus = list(range(80, 111))  # 80-110 岁
                    avg_rates = np.zeros(len(ages_80plus))
                    count = np.zeros(len(ages_80plus))

                    # 计算加权平均
                    for country in top_countries:
```

```
                        weight = similarity_scores[country]['score']
                        country_data = international_data[gender][country]

                        for i, age in enumerate(ages_80plus):
                            if age in country_data['ages']:
                                idx = np.where(country_data['ages'] == age)[0][0
    ]
                                rate = country_data['rates'][idx]

                                if rate > 0:
                                    avg_rates[i] += rate * weight
                                    count[i] += weight

                    # 计算平均率
                    for i in range(len(ages_80plus)):
                        if count[i] > 0:
                            avg_rates[i] /= count[i]

                    # 调整平均率，使其与我们的80岁死亡率连续
                    idx_80 = np.where(pred_ages == 80)[0]
                    if len(idx_80) > 0:
                        idx_80 = idx_80[0]
                        our_80 = pred_mort[idx_80]
                        intl_80 = avg_rates[0]

                        # 计算调整系数
                        adjustment_factor = our_80 / intl_80 if intl_80 > 0 else
     1.0
                        print(f"  调整系数: {adjustment_factor:.4f} (我们的80岁死
    亡率/国际80岁平均死亡率)")

                        # 应用调整系数
                        avg_rates *= adjustment_factor

                        # 针对70-80岁设置平滑过渡
                        for age in range(70, 80):
                            if age in pred_ages:
                                idx = np.where(pred_ages == age)[0][0]
                                # 保持不变，因为我们只调整80岁以上的数据

                        # 应用调整后的80岁以上死亡率
                        for i, age in enumerate(ages_80plus):
                            if age in pred_ages:
                                idx = np.where(pred_ages == age)[0][0]
```

```
                                if avg_rates[i] > 0:
                                    old_rate = pred_mort[idx]
                                    new_rate = avg_rates[i]
                                    print(f"    {age}岁: {old_rate:.6f} -> {new_rate:.6f} (变化率: {(new_rate/old_rate-1)*100:.1f}%)")
                                    pred_mort[idx] = new_rate

                    # 更新预测结果
                    self.predictions_2025[gender]['mortality_adjusted'] = pred_mort.copy()

                    # 比较调整前后的关键年龄
                    key_ages = [80, 85, 90, 95, 100]
                    print("\n  调整前后的关键年龄对比:")
                    print("  年龄 | 原始预测 | 调整后 | 变化率")
                    print("  --------------------------------")

                    for age in key_ages:
                        if age in pred_ages:
                            idx = np.where(pred_ages == age)[0][0]
                            old_val = self.predictions_2025[gender]['mortality_smoothed'][idx]
                            new_val = pred_mort[idx]
                            change = (new_val / old_val - 1) * 100
                            print(f"  {age:3d} | {old_val:.6f} | {new_val:.6f} | {change:+.1f}%")

                    # 再次进行修匀
                    adjusted_mort_smoothed = self.whittaker_henderson_smooth(pred_mort, lambda_param=1000)
                    self.predictions_2025[gender]['mortality_smoothed'] = adjusted_mort_smoothed

                    # 确保单调性（对于30岁以上）
                    monotonic_violations = 0
                    for i in range(1, len(adjusted_mort_smoothed)):
                        if pred_ages[i] > 30 and adjusted_mort_smoothed[i] < adjusted_mort_smoothed[i-1]:
                            adjusted_mort_smoothed[i] = adjusted_mort_smoothed[i-1] * 1.01
                            monotonic_violations += 1

                    if monotonic_violations > 0:
```

```
                    print(f"  单调性修正: {monotonic_violations}个年龄点被调整")

                print(f"  最终调整后死亡率范围: {adjusted_mort_smoothed.min():.6f} - {adjusted_mort_smoothed.max():.6f}")
            else:
                print("  没有找到足够相似的国家，保持原始预测")

        print("\n80岁以上死亡率调整完成！")
        return True

    def extrapolate_to_120(self):
        """外推至120岁"""
        print("正在外推至120岁...")

        for gender in ['M', 'F']:
            print(f"处理{gender}性外推...")

            current_ages = self.predictions_2025[gender]['ages']
            current_mortality = self.predictions_2025[gender]['mortality_smoothed']

            # 创建120岁的年龄序列
            extended_ages = np.arange(0, 121)
            extended_mortality = np.zeros(121)

            # 对于现有年龄范围，使用预测值
            for i, age in enumerate(extended_ages):
                if age in current_ages:
                    idx = np.where(current_ages == age)[0][0]
                    extended_mortality[i] = current_mortality[idx]

            # 对于90岁以上，使用Gompertz-Makeham外推
            if self.gompertz_params[gender] is not None:
                gm_params = self.gompertz_params[gender]['params']

                # 外推90-120岁
                for age in range(91, 121):
                    if age in extended_ages:
                        idx = age  # 因为extended_ages是0-120
                        # 使用Gompertz-Makeham模型外推
                        extended_mortality[idx] = self.gompertz_makeham(age, *gm_params)

```

```
                            # 应用上限约束（死亡率不超过80%）
                            extended_mortality[idx] = min(extended_mortality[idx], 0.8)
                else:
                    # 如果没有Gompertz-Makeham参数，使用指数外推
                    last_valid_age = int(current_ages[-1])
                    last_mortality = current_mortality[-1]

                    # 估计增长率
                    if len(current_mortality) >= 10:
                        growth_rate = np.mean(np.diff(np.log(current_mortality[-10:])))
                    else:
                        growth_rate = 0.1  # 默认增长率

                    for age in range(last_valid_age + 1, 121):
                        years_ahead = age - last_valid_age
                        extended_mortality[age] = last_mortality * np.exp(growth_rate * years_ahead)
                        extended_mortality[age] = min(extended_mortality[age], 0.8)

                # 对于低年龄段（0-现有最小年龄），使用插值
                min_age = int(current_ages[0])
                if min_age > 0:
                    # 简单线性插值至0岁
                    for age in range(0, min_age):
                        ratio = age / min_age
                        extended_mortality[age] = extended_mortality[min_age] * ratio

                # 最终修匀
                extended_mortality = self.whittaker_henderson_smooth(extended_mortality, lambda_param=500)

                # 确保单调性约束（30岁以上）
                for i in range(31, 121):
                    if extended_mortality[i] < extended_mortality[i-1]:
                        extended_mortality[i] = extended_mortality[i-1] * 1.005

                # 确保非负性
                extended_mortality = np.maximum(extended_mortality, 1e-6)

                self.extrapolated_120[gender] = {
```

```
                    'ages': extended_ages,
                    'mortality': extended_mortality
                }

            print("120岁外推完成！")

        def generate_results_table(self):
            """生成结果表格"""
            results = []

            for gender in ['M', 'F']:
                ages = self.extrapolated_120[gender]['ages']
                mortality = self.extrapolated_120[gender]['mortality']

                for i, age in enumerate(ages):
                    results.append({
                        '性别': gender,
                        '年龄': age,
                        '2025年预测死亡率': mortality[i]
                    })

            return pd.DataFrame(results)

        def create_visualizations(self, save_path="Task 2"):
            """创建可视化图表"""
            print("正在生成可视化图表...")

            # 创建保存目录
            os.makedirs(save_path, exist_ok=True)

            # 1. 历史趋势分析图
            fig, axes = plt.subplots(2, 2, figsize=(15, 12))

            for i, gender in enumerate(['M', 'F']):
                # 历史死亡率趋势
                matrix = self.mortality_matrix[gender]

                # 选择几个关键年龄显示趋势
                key_ages = [30, 50, 70, 90]
                for age in key_ages:
                    if age in matrix.index:
                        values = matrix.loc[age].values
                        years = matrix.columns.values
```

```
                    axes[i, 0].plot(years, values, marker='o', label=f'{age}岁', linewidth=2)

                axes[i, 0].set_title(f'{gender}性历史死亡率趋势', fontsize=14)
                axes[i, 0].set_xlabel('年份')
                axes[i, 0].set_ylabel('死亡率')
                axes[i, 0].legend()
                axes[i, 0].grid(True, alpha=0.3)

                # Lee-Carter 模型参数
                if gender in self.lee_carter_params:
                    lc_params = self.lee_carter_params[gender]

                    # b(x)参数图
                    axes[i, 1].plot(lc_params['ages'], lc_params['bx'], 'b-', linewidth=2)
                    axes[i, 1].set_title(f'{gender}性 Lee-Carter 模型 b(x)参数', fontsize=14)
                    axes[i, 1].set_xlabel('年龄')
                    axes[i, 1].set_ylabel('b(x)')
                    axes[i, 1].grid(True, alpha=0.3)

            plt.tight_layout()
            plt.savefig(f'{save_path}/historical_analysis.png', dpi=300, bbox_inches='tight')
            plt.close()

            # 2. 2025 年预测结果图
            fig, axes = plt.subplots(1, 2, figsize=(15, 6))

            for i, gender in enumerate(['M', 'F']):
                pred_data = self.predictions_2025[gender]

                axes[i].plot(pred_data['ages'], pred_data['mortality_lc'],
                             'b--', label='Lee-Carter 预测', linewidth=2, alpha=0.7)
                axes[i].plot(pred_data['ages'], pred_data['mortality_combined'],
                             'g:', label='组合模型预测', linewidth=2, alpha=0.7)
                axes[i].plot(pred_data['ages'], pred_data['mortality_smoothed'],
                             'r-', label='修匀后预测', linewidth=3)

                # 添加历史数据对比
```

```
780.                 hist_2021 = self.mortality_matrix[gender][2021]
781.                 axes[i].plot(hist_2021.index, hist_2021.values,
782.                              'ko-', label='2021 年实际', alpha=0.5, markersize=3)
783.
784.                 axes[i].set_title(f'{gender}性 2025 年死亡率预测', fontsize=14)
785.                 axes[i].set_xlabel('年龄')
786.                 axes[i].set_ylabel('死亡率')
787.                 axes[i].set_yscale('log')
788.                 axes[i].legend()
789.                 axes[i].grid(True, alpha=0.3)
790.
791.             plt.tight_layout()
792.             plt.savefig(f'{save_path}/prediction_2025.png', dpi=300, bbox_inches
    ='tight')
793.             plt.close()
794.
795.             # 3. 120 岁外推结果图
796.             fig, axes = plt.subplots(1, 2, figsize=(15, 6))
797.
798.             for i, gender in enumerate(['M', 'F']):
799.                 ext_data = self.extrapolated_120[gender]
800.
801.                 axes[i].plot(ext_data['ages'], ext_data['mortality'],
802.                              'r-', linewidth=3, label='外推至 120 岁')
803.
804.                 # 标记 90 岁分界线
805.                 axes[i].axvline(x=90, color='blue', linestyle='--
    ', alpha=0.7, label='原始数据边界')
806.
807.                 axes[i].set_title(f'{gender}性死亡率外推至 120 岁', fontsize=14)
808.                 axes[i].set_xlabel('年龄')
809.                 axes[i].set_ylabel('死亡率')
810.                 axes[i].set_yscale('log')
811.                 axes[i].legend()
812.                 axes[i].grid(True, alpha=0.3)
813.
814.                 # 添加年龄段标注
815.                 axes[i].text(45, ext_data['mortality'][45], '成年期
    ', fontsize=10, alpha=0.7)
816.                 axes[i].text(75, ext_data['mortality'][75], '老年期
    ', fontsize=10, alpha=0.7)
817.                 axes[i].text(105, ext_data['mortality'][105], '外推区间
    ', fontsize=10, alpha=0.7)
818.
```

```
            plt.tight_layout()
            plt.savefig(f'{save_path}/extrapolation_120.png', dpi=300, bbox_inches='tight')
            plt.close()

            # 4. 模型比较图
            fig, axes = plt.subplots(2, 1, figsize=(12, 10))

            for i, gender in enumerate(['M', 'F']):
                # Gompertz-Makeham 拟合效果
                if self.gompertz_params[gender] is not None:
                    gm_data = self.gompertz_params[gender]
                    fit_ages = gm_data['fit_ages']
                    fit_rates = gm_data['fit_rates']
                    predicted_rates = self.gompertz_makeham(fit_ages, *gm_data['params'])

                    axes[i].scatter(fit_ages, fit_rates, alpha=0.6, label='实际数据', s=30)
                    axes[i].plot(fit_ages, predicted_rates, 'r-',
                                 label=f'Gompertz-Makeham 拟合 (R²={gm_data["r_squared"]:.3f})', linewidth=2)

                    axes[i].set_title(f'{gender}性 Gompertz-Makeham 模型拟合效果', fontsize=14)
                    axes[i].set_xlabel('年龄')
                    axes[i].set_ylabel('死亡率')
                    axes[i].set_yscale('log')
                    axes[i].legend()
                    axes[i].grid(True, alpha=0.3)

            plt.tight_layout()
            plt.savefig(f'{save_path}/model_fitting.png', dpi=300, bbox_inches='tight')
            plt.close()

            print("可视化图表生成完成！")

        def run_complete_analysis(self):
            """运行完整分析流程"""
            print("开始完整的死亡率预测分析...")
            print("="*60)

            # 1. 数据加载和预处理
```

```
            self.load_and_prepare_data()

            # 2. 2025 年预测
            self.predict_2025_mortality()

            # 3. 80 岁以上死亡率调整
            self.adjust_80plus_with_international_data()

            # 4. 120 岁外推
            self.extrapolate_to_120()

            # 5. 生成结果表格
            results_df = self.generate_results_table()

            # 6. 创建可视化
            self.create_visualizations()

            # 7. 保存结果
            results_df.to_csv('Task 2/mortality_predictions_2025.csv', index=False, encoding='utf-8-sig')

            print("="*60)
            print("分析完成！结果已保存到 Task 2 文件夹")

            return results_df


    if __name__ == "__main__":
        # 使用示例
        predictor = MortalityPredictor("【选题 1 数据】某保险产品死亡率.xlsx")
        results = predictor.run_complete_analysis()

        # 显示部分结果
        print("\n 预测结果示例:")
        print(results.head(20))
```

6. 模型评估与合理性验证（2025 年保险人口死亡率预测模型）

```
"""
模型验证与合理性分析模块
用于评估死亡率预测模型的质量和结果的合理性
"""

import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import warnings
warnings.filterwarnings('ignore')

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

class ModelValidator:
    def __init__(self, predictor):
        """
        初始化模型验证器

        Parameters:
        predictor: MortalityPredictor 实例
        """
        self.predictor = predictor
        self.validation_results = {}

    def cross_validation_analysis(self):
        """交叉验证分析"""
        print("进行交叉验证分析...")

        validation_results = {}

        for gender in ['M', 'F']:
            print(f"验证{gender}性模型...")

            matrix = self.predictor.mortality_matrix[gender]
            years = matrix.columns.values

            # 使用留一年法进行交叉验证
            errors_lc = []
            errors_gm = []

            for test_year in years[-3:]:  # 测试最后3年
                # 训练集：除test_year外的所有年份
                train_years = [y for y in years if y != test_year]
                train_matrix = matrix[train_years]

                # 拟合Lee-Carter模型
```

```
                log_mort = np.log(train_matrix.values)
                ax = np.mean(log_mort, axis=1)
                centered_matrix = log_mort - ax.reshape(-1, 1)
                U, s, Vt = np.linalg.svd(centered_matrix, full_matrices=False)
                bx = U[:, 0] / np.sum(U[:, 0])
                kt = s[0] * Vt[0, :]
                kt = kt - kt[0]

                # 预测test_year
                # 简单线性外推
                train_years_array = np.array(train_years)
                slope = np.polyfit(train_years_array, kt, 1)[0]
                kt_pred = kt[-1] + slope * (test_year - train_years[-1])

                # 计算预测死亡率
                log_mort_pred = ax + bx * kt_pred
                mort_pred_lc = np.exp(log_mort_pred)

                # 实际值
                mort_actual = matrix[test_year].values

                # 计算误差
                valid_mask = np.isfinite(mort_actual) & np.isfinite(mort_pred_lc)
                if np.sum(valid_mask) > 0:
                    mse_lc = mean_squared_error(mort_actual[valid_mask], mort_pred_lc[valid_mask])
                    errors_lc.append(mse_lc)

                # 对于Gompertz-Makeham，使用更简单的验证
                if len(train_matrix.columns) > 5:  # 确保有足够数据
                    recent_mort = train_matrix.iloc[:, -1].values
                    ages = train_matrix.index.values

                    # 拟合Gompertz-Makeham
                    gm_result = self.predictor.fit_gompertz_makeham(ages, recent_mort)
                    if gm_result is not None:
                        mort_pred_gm = self.predictor.gompertz_makeham(ages, *gm_result['params'])

                        if np.sum(valid_mask) > 0:
                            mse_gm = mean_squared_error(mort_actual[valid_mask], mort_pred_gm[valid_mask])
```

```
                            errors_gm.append(mse_gm)

            validation_results[gender] = {
                'lc_mse': np.mean(errors_lc) if errors_lc else np.nan,
                'gm_mse': np.mean(errors_gm) if errors_gm else np.nan,
                'lc_errors': errors_lc,
                'gm_errors': errors_gm
            }

        self.validation_results['cross_validation'] = validation_results
        print("交叉验证完成！")

    def residual_analysis(self):
        """残差分析"""
        print("进行残差分析...")

        residual_results = {}

        for gender in ['M', 'F']:
            if gender in self.predictor.lee_carter_params:
                lc_params = self.predictor.lee_carter_params[gender]
                matrix = self.predictor.mortality_matrix[gender]

                # 重构死亡率
                reconstructed_log = (lc_params['ax'].reshape(-1, 1) +
                                     np.outer(lc_params['bx'], lc_params['kt']
))
                reconstructed = np.exp(reconstructed_log)

                # 计算残差
                residuals = np.log(matrix.values) - reconstructed_log

                # 残差统计
                residual_stats = {
                    'mean': np.mean(residuals),
                    'std': np.std(residuals),
                    'skewness': stats.skew(residuals.flatten()),
                    'kurtosis': stats.kurtosis(residuals.flatten()),
                    'shapiro_p': stats.shapiro(residuals.flatten())[1] if re
siduals.size < 5000 else np.nan
                }

                residual_results[gender] = {
                    'residuals': residuals,
```

```
                        'stats': residual_stats,
                        'reconstructed': reconstructed
                    }

            self.validation_results['residuals'] = residual_results
            print("残差分析完成！")

        def biological_plausibility_check(self):
            """生物学合理性检查"""
            print("进行生物学合理性检查...")

            plausibility_results = {}

            for gender in ['M', 'F']:
                ext_data = self.predictor.extrapolated_120[gender]
                ages = ext_data['ages']
                mortality = ext_data['mortality']

                checks = {}

                # 1. 单调性检查（30岁以上）
                age_30_idx = np.where(ages >= 30)[0]
                if len(age_30_idx) > 1:
                    mortality_30plus = mortality[age_30_idx[0]:]
                    ages_30plus = ages[age_30_idx[0]:]

                    # 计算违反单调性的比例
                    violations = 0
                    for i in range(1, len(mortality_30plus)):
                        if mortality_30plus[i] < mortality_30plus[i-1]:
                            violations += 1

                    monotonicity_ratio = 1 - violations / (len(mortality_30plus) - 1)
                    checks['monotonicity_30plus'] = monotonicity_ratio

                # 2. 合理的死亡率范围检查
                checks['min_mortality'] = np.min(mortality)
                checks['max_mortality'] = np.max(mortality)
                checks['max_mortality_120'] = mortality[120] if len(mortality) > 120 else mortality[-1]

                # 3. 年龄增长模式检查
                # 计算死亡率加速度（二阶导数）
```

```
174.                 if len(mortality) > 2:
175.                     first_diff = np.diff(mortality)
176.                     second_diff = np.diff(first_diff)
177.                     checks['acceleration_mean'] = np.mean(second_diff)
178.                     checks['acceleration_positive_ratio'] = np.sum(second_diff >
    0) / len(second_diff)
179.
180.                 # 4. 与国际标准对比
181.                 # 使用简化的国际参考值
182.                 international_benchmarks = {
183.                     'M': {'age_65': 0.015, 'age_85': 0.08, 'age_100': 0.3},
184.                     'F': {'age_65': 0.010, 'age_85': 0.06, 'age_100': 0.25}
185.                 }
186.
187.                 benchmark_checks = {}
188.                 for age_label, benchmark_rate in international_benchmarks[gender
    ].items():
189.                     age_num = int(age_label.split('_')[1])
190.                     if age_num < len(mortality):
191.                         predicted_rate = mortality[age_num]
192.                         ratio = predicted_rate / benchmark_rate
193.                         benchmark_checks[age_label] = {
194.                             'predicted': predicted_rate,
195.                             'benchmark': benchmark_rate,
196.                             'ratio': ratio
197.                         }
198.
199.                 checks['international_comparison'] = benchmark_checks
200.
201.                 plausibility_results[gender] = checks
202.
203.             self.validation_results['plausibility'] = plausibility_results
204.             print("生物学合理性检查完成！")
205.
206.         def model_performance_metrics(self):
207.             """模型性能指标"""
208.             print("计算模型性能指标...")
209.
210.             performance_results = {}
211.
212.             for gender in ['M', 'F']:
213.                 if gender in self.predictor.lee_carter_params:
214.                     lc_params = self.predictor.lee_carter_params[gender]
215.                     matrix = self.predictor.mortality_matrix[gender]
```

```
216.
217.                     # Lee-Carter 模型拟合优度
218.                     reconstructed_log = (lc_params['ax'].reshape(-1, 1) +
219.                                          np.outer(lc_params['bx'], lc_params['kt']
    ))
220.
221.                     actual_log = np.log(matrix.values)
222.
223.                     # 计算 R²
224.                     ss_res = np.sum((actual_log - reconstructed_log) ** 2)
225.                     ss_tot = np.sum((actual_log - np.mean(actual_log)) ** 2)
226.                     r_squared_lc = 1 - (ss_res / ss_tot)
227.
228.                     # 计算其他指标
229.                     mse_lc = mean_squared_error(actual_log.flatten(), reconstruc
    ted_log.flatten())
230.                     mae_lc = mean_absolute_error(actual_log.flatten(), reconstru
    cted_log.flatten())
231.
232.                     performance_results[gender] = {
233.                         'lee_carter': {
234.                             'r_squared': r_squared_lc,
235.                             'mse': mse_lc,
236.                             'mae': mae_lc,
237.                             'explained_variance': lc_params['kt'].var() / actual
    _log.var()
238.                         }
239.                     }
240.
241.                     # Gompertz-Makeham 性能
242.                     if self.predictor.gompertz_params[gender] is not None:
243.                         gm_data = self.predictor.gompertz_params[gender]
244.                         performance_results[gender]['gompertz_makeham'] = {
245.                             'r_squared': gm_data['r_squared'],
246.                             'parameters': gm_data['params']
247.                         }
248.
249.             self.validation_results['performance'] = performance_results
250.             print("模型性能指标计算完成！")
251.
252.         def create_validation_visualizations(self, save_path="Task 2"):
253.             """创建验证可视化图表"""
254.             print("生成验证可视化图表...")
255.
```

```
256.             # 1. 残差分析图
257.             if 'residuals' in self.validation_results:
258.                 fig, axes = plt.subplots(2, 2, figsize=(15, 12))
259.
260.                 for i, gender in enumerate(['M', 'F']):
261.                     if gender in self.validation_results['residuals']:
262.                         residuals = self.validation_results['residuals'][gender]
    ['residuals']
263.
264.                         # 残差热力图
265.                         im = axes[i, 0].imshow(residuals, cmap='RdBu_r', aspect=
    'auto')
266.                         axes[i, 0].set_title(f'{gender}性 Lee-Carter 模型残差热力图
    ')
267.                         axes[i, 0].set_xlabel('年份')
268.                         axes[i, 0].set_ylabel('年龄')
269.                         plt.colorbar(im, ax=axes[i, 0])
270.
271.                         # 残差分布直方图
272.                         axes[i, 1].hist(residuals.flatten(), bins=50, alpha=0.7,
     edgecolor='black')
273.                         axes[i, 1].set_title(f'{gender}性残差分布')
274.                         axes[i, 1].set_xlabel('残差值')
275.                         axes[i, 1].set_ylabel('频数')
276.                         axes[i, 1].grid(True, alpha=0.3)
277.
278.                         # 添加正态分布参考线
279.                         x = np.linspace(residuals.min(), residuals.max(), 100)
280.                         y = stats.norm.pdf(x, np.mean(residuals), np.std(residua
    ls))
281.                         y = y * len(residuals.flatten()) * (residuals.max() - re
    siduals.min()) / 50
282.                         axes[i, 1].plot(x, y, 'r-', label='正态分布
    ', linewidth=2)
283.                         axes[i, 1].legend()
284.
285.                 plt.tight_layout()
286.                 plt.savefig(f'{save_path}/residual_analysis.png', dpi=300, bbox_
    inches='tight')
287.                 plt.close()
288.
289.             # 2. 生物学合理性检查图
290.             if 'plausibility' in self.validation_results:
291.                 fig, axes = plt.subplots(1, 2, figsize=(15, 6))
```

```
292.
293.             for i, gender in enumerate(['M', 'F']):
294.                 if gender in self.validation_results['plausibility']:
295.                     # 国际对比图
296.                     int_comp = self.validation_results['plausibility'][gende
    r]['international_comparison']
297.
298.                     ages = []
299.                     predicted = []
300.                     benchmark = []
301.
302.                     for age_label, data in int_comp.items():
303.                         age_num = int(age_label.split('_')[1])
304.                         ages.append(age_num)
305.                         predicted.append(data['predicted'])
306.                         benchmark.append(data['benchmark'])
307.
308.                     if ages:
309.                         x = np.arange(len(ages))
310.                         width = 0.35
311.
312.                         axes[i].bar(x - width/2, predicted, width, label='预
    测值', alpha=0.8)
313.                         axes[i].bar(x + width/2, benchmark, width, label='国
    际参考', alpha=0.8)
314.
315.                         axes[i].set_xlabel('年龄')
316.                         axes[i].set_ylabel('死亡率')
317.                         axes[i].set_title(f'{gender}性预测结果与国际参考对比')
318.                         axes[i].set_xticks(x)
319.                         axes[i].set_xticklabels([f'{age}岁
    ' for age in ages])
320.                         axes[i].legend()
321.                         axes[i].grid(True, alpha=0.3)
322.                         axes[i].set_yscale('log')
323.
324.             plt.tight_layout()
325.             plt.savefig(f'{save_path}/plausibility_check.png', dpi=300, bbox
    _inches='tight')
326.             plt.close()
327.
328.         # 3. 模型性能对比图
329.         if 'performance' in self.validation_results:
330.             fig, axes = plt.subplots(1, 2, figsize=(12, 5))
```

```
331. 
332.                 genders = []
333.                 r2_lc = []
334.                 r2_gm = []
335. 
336.                 for gender in ['M', 'F']:
337.                     if gender in self.validation_results['performance']:
338.                         perf = self.validation_results['performance'][gender]
339.                         genders.append(gender)
340.                         r2_lc.append(perf['lee_carter']['r_squared'])
341.                         if 'gompertz_makeham' in perf:
342.                             r2_gm.append(perf['gompertz_makeham']['r_squared'])
343.                         else:
344.                             r2_gm.append(0)
345. 
346.                 x = np.arange(len(genders))
347.                 width = 0.35
348. 
349.                 axes[0].bar(x - width/2, r2_lc, width, label='Lee-
    Carter R²', alpha=0.8)
350.                 axes[0].bar(x + width/2, r2_gm, width, label='Gompertz-
    Makeham R²', alpha=0.8)
351.                 axes[0].set_xlabel('性别')
352.                 axes[0].set_ylabel('R² 值')
353.                 axes[0].set_title('模型拟合优度对比')
354.                 axes[0].set_xticks(x)
355.                 axes[0].set_xticklabels(genders)
356.                 axes[0].legend()
357.                 axes[0].grid(True, alpha=0.3)
358. 
359.                 # MSE 对比
360.                 mse_lc = []
361.                 for gender in genders:
362.                     if gender in self.validation_results['performance']:
363.                         mse_lc.append(self.validation_results['performance'][gen
    der]['lee_carter']['mse'])
364. 
365.                 axes[1].bar(genders, mse_lc, alpha=0.8, color='orange')
366.                 axes[1].set_xlabel('性别')
367.                 axes[1].set_ylabel('MSE')
368.                 axes[1].set_title('Lee-Carter 模型预测误差')
369.                 axes[1].grid(True, alpha=0.3)
370. 
371.                 plt.tight_layout()
```

```
372.                 plt.savefig(f'{save_path}/model_performance.png', dpi=300, bbox_
    inches='tight')
373.                 plt.close()
374.
375.             print("验证可视化图表生成完成！")
376.
377.         def generate_validation_report(self):
378.             """生成验证报告"""
379.             report = []
380.             report.append("="*60)
381.             report.append("死亡率预测模型验证报告")
382.             report.append("="*60)
383.
384.             # 1. 模型性能总结
385.             report.append("\n1. 模型性能总结")
386.             report.append("-" * 30)
387.
388.             if 'performance' in self.validation_results:
389.                 for gender in ['M', 'F']:
390.                     if gender in self.validation_results['performance']:
391.                         perf = self.validation_results['performance'][gender]
392.                         report.append(f"\n{gender}性:")
393.                         report.append(f"  Lee-Carter 模
    型 R²: {perf['lee_carter']['r_squared']:.4f}")
394.                         report.append(f"  Lee-Carter 模
    型 MSE: {perf['lee_carter']['mse']:.6f}")
395.
396.                         if 'gompertz_makeham' in perf:
397.                             report.append(f"  Gompertz-Makeham 模
    型 R²: {perf['gompertz_makeham']['r_squared']:.4f}")
398.
399.             # 2. 生物学合理性评估
400.             report.append("\n\n2. 生物学合理性评估")
401.             report.append("-" * 30)
402.
403.             if 'plausibility' in self.validation_results:
404.                 for gender in ['M', 'F']:
405.                     if gender in self.validation_results['plausibility']:
406.                         plaus = self.validation_results['plausibility'][gender]
407.                         report.append(f"\n{gender}性:")
408.
409.                         if 'monotonicity_30plus' in plaus:
410.                             mono_ratio = plaus['monotonicity_30plus']
```

```
                            report.append(f"  30岁以上单调性: {mono_ratio:.2%} (良好阈值>95%)")

                        report.append(f"  最小死亡率: {plaus['min_mortality']:.6f}")
                        report.append(f"  最大死亡率: {plaus['max_mortality']:.6f}")

                        if 'max_mortality_120' in plaus:
                            report.append(f"  120岁死亡率: {plaus['max_mortality_120']:.6f}")

                        # 国际对比
                        if 'international_comparison' in plaus:
                            report.append("  国际对比:")
                            for age_label, data in plaus['international_comparison'].items():
                                age = age_label.split('_')[1]
                                ratio = data['ratio']
                                status = "合理" if 0.5 <= ratio <= 2.0 else "需关注"
                                report.append(f"    {age}岁: 预测/参考 = {ratio:.2f} ({status})")

            # 3. 残差分析
            report.append("\n\n3. 残差分析")
            report.append("-" * 30)

            if 'residuals' in self.validation_results:
                for gender in ['M', 'F']:
                    if gender in self.validation_results['residuals']:
                        res_stats = self.validation_results['residuals'][gender]['stats']
                        report.append(f"\n{gender}性残差统计:")
                        report.append(f"  均值: {res_stats['mean']:.6f} (理想值≈0)")
                        report.append(f"  标准差: {res_stats['std']:.6f}")
                        report.append(f"  偏度: {res_stats['skewness']:.4f} (理想值≈0)")
                        report.append(f"  峰度: {res_stats['kurtosis']:.4f} (理想值≈0)")

                        if not np.isnan(res_stats['shapiro_p']):
```

```
                            normality = "正态
    " if res_stats['shapiro_p'] > 0.05 else "非正态"
                            report.append(f"  正态性检验 p
    值: {res_stats['shapiro_p']:.4f} ({normality})")

            # 4. 交叉验证结果
            if 'cross_validation' in self.validation_results:
                report.append("\n\n4. 交叉验证结果")
                report.append("-" * 30)

                for gender in ['M', 'F']:
                    if gender in self.validation_results['cross_validation']:
                        cv_results = self.validation_results['cross_validation']
    [gender]
                        report.append(f"\n{gender}性:")

                        if not np.isnan(cv_results['lc_mse']):
                            report.append(f"  Lee-Carter 平均
    MSE: {cv_results['lc_mse']:.6f}")

                        if not np.isnan(cv_results['gm_mse']):
                            report.append(f"  Gompertz-Makeham 平均
    MSE: {cv_results['gm_mse']:.6f}")

            # 5. 总体评估
            report.append("\n\n5. 总体评估与建议")
            report.append("-" * 30)
            report.append("\n 模型质量评级:")

            # 简单的评级逻辑
            overall_quality = "良好"

            if 'performance' in self.validation_results:
                avg_r2 = np.mean([self.validation_results['performance'][g]['lee
    _carter']['r_squared']
                                  for g in ['M', 'F'] if g in self.validation_res
    ults['performance']])
                if avg_r2 < 0.8:
                    overall_quality = "需改进"
                elif avg_r2 > 0.95:
                    overall_quality = "优秀"

            report.append(f"  整体质量: {overall_quality}")

```

```
            report.append("\n 建议:")
            report.append("  1. 定期更新模型参数以反映最新趋势")
            report.append("  2. 监控预测结果与实际观测的偏差")
            report.append("  3. 考虑外部因素（医疗技术进步、政策变化等）的影响")
            report.append("  4. 建立预测区间以量化不确定性")

            return "\n".join(report)

        def run_full_validation(self):
            """运行完整验证流程"""
            print("开始完整的模型验证...")
            print("="*50)

            # 执行各项验证
            self.cross_validation_analysis()
            self.residual_analysis()
            self.biological_plausibility_check()
            self.model_performance_metrics()

            # 生成可视化
            self.create_validation_visualizations()

            # 生成报告
            report = self.generate_validation_report()

            # 保存报告
            with open('Task 2/validation_report.txt', 'w', encoding='utf-
    8') as f:
                f.write(report)

            print("="*50)
            print("模型验证完成! 报告已保存到 Task 2/validation_report.txt")

            return report


    if __name__ == "__main__":
        # 需要先运行 mortality_predictor.py
        from mortality_predictor import MortalityPredictor

        predictor = MortalityPredictor("【选题 1 数据】某保险产品死亡率.xlsx")
        predictor.run_complete_analysis()

        validator = ModelValidator(predictor)
```

```
    validation_report = validator.run_full_validation()

    print(validation_report)
```

7. 2025-2125 年保险人口长期死亡率趋势预测模型

```
"""
Long-Term Mortality Trend Forecast (2025-2125)
Core module implementing the forecasting methodology described in the report
"""

import numpy as np
import pandas as pd
from scipy.optimize import curve_fit
from scipy.interpolate import splrep, splev
import matplotlib.pyplot as plt
import os
import sys

# Add parent directory to path to import Task 2 modules
sys.path.append('../../问题 2 报告/改进后')
try:
    from mortality_predictor import MortalityPredictor
except ImportError:
    print("Warning: Could not import MortalityPredictor from Task 2")


class LongTermMortalityModel:
    """
    Long-term mortality trend projection model (2025-2125)

    This model extends the short-term projections from Task 2 to create
    a 100-year forecast with gradually slowing improvement rates.
    """

    def __init__(self, data_path=None, task2_predictor=None):
        """
        Initialize the long-term mortality model

        Parameters:
        -----------
        data_path : str
            Path to the mortality data Excel file
        task2_predictor : MortalityPredictor
```

```
            Pre-initialized predictor from Task 2 (optional)
        """
        self.data_path = data_path
        self.predictor = task2_predictor
        self.projections = {}
        self.life_expectancy = {}

        # Key model parameters
        self.projection_start_year = 2025
        self.projection_end_year = 2125
        self.projection_years = list(range(self.projection_start_year,
                                           self.projection_end_year + 1, 5))

        # Model parameters for the long-term improvement curves
        self.improvement_params = {
            'M': {
                'initial_rate': 0.015,  # Initial annual mortality improvement rate
                'asymptotic_rate': 0.003,  # Long-term "floor" improvement rate
                'half_life_years': 30,  # Years until improvement rate halves
            },
            'F': {
                'initial_rate': 0.012,  # Women start with slightly lower improvement rate
                'asymptotic_rate': 0.0025,  # But maintain a steady floor
                'half_life_years': 35,  # Women's improvement declines more slowly
            }
        }

        # Initialize projections dictionary structure
        self._initialize_projections()

    def _initialize_projections(self):
        """Initialize data structures for projections"""
        for gender in ['M', 'F']:
            self.projections[gender] = {
                'mortality_rates': {},  # Year -> age -> rate mapping
                'improvement_rates': {},  # Year -> age -> improvement rate
                'kt_values': {},  # Year -> kt value for Lee-Carter
            }

            self.life_expectancy[gender] = {
                'at_birth': {},  # Year -> e(0)
```

```
                'at_65': {},     # Year -> e(65)
            }

    def initialize_from_task2(self):
        """Initialize the model using Task 2's short-term projection"""
        if self.predictor is None:
            if self.data_path is None:
                raise ValueError("Either data_path or task2_predictor must be p
rovided")

            print("Initializing short-term model from Task 2...")
            self.predictor = MortalityPredictor(self.data_path)
            self.predictor.run_complete_analysis()

        print("Extracting 2025 projections as baseline...")
        for gender in ['M', 'F']:
            if gender in self.predictor.extrapolated_120:
                # Get the full age range (0-120) mortality rates for 2025
                baseline_ages = np.arange(0, 121)
                baseline_rates = self.predictor.extrapolated_120[gender]['morta
lity']

                # Fix any rates that are too high (>0.9) in the base data
                for i, rate in enumerate(baseline_rates):
                    if rate > 0.9:
                        # Apply sigmoid transformation to keep rates below 1
                        baseline_rates[i] = 0.9 + 0.099 * (1 / (1 + np.exp(-
(rate - 0.9) * 10)))

                # Store as the starting point for long-term projection
                self.projections[gender]['mortality_rates'][self.projection_
start_year] = {
                    age: rate for age, rate in zip(baseline_ages, baseline_r
ates)
                }

    def improvement_decay_function(self, time, gender):
        """
        Model the decay of mortality improvement rates over time

        Parameters:
        -----------
        time : int or array
            Years from projection start (0 = 2025)
```

```
        gender : str
            'M' or 'F'

        Returns:
        --------
        float or array
            Mortality improvement rate at the given time
        """
        params = self.improvement_params[gender]
        initial = params['initial_rate']
        asymptotic = params['asymptotic_rate']
        half_life = params['half_life_years']

        # Exponential decay with floor
        decay_factor = np.exp(-np.log(2) * time / half_life)
        return asymptotic + (initial - asymptotic) * decay_factor

    def age_specific_improvement_pattern(self, age, gender, year_offset):
        """
        Create age-specific mortality improvement patterns

        Parameters:
        -----------
        age : int
            Age (0-120)
        gender : str
            'M' or 'F'
        year_offset : int
            Years from projection start (0 = 2025)

        Returns:
        --------
        float
            Age-specific mortality improvement factor
        """
        # Base improvement rate from the decay function
        base_rate = self.improvement_decay_function(year_offset, gender)

        # Age-specific modifiers
        if age < 10:
            # Child mortality improvements will continue strongly
            mod = 1.2
        elif age < 40:
            # Young adult improvements moderate
```

```
163.                 mod = 0.8
164.             elif age < 65:
165.                 # Middle age improvements average
166.                 mod = 1.0
167.             elif age < 85:
168.                 # Senior improvements above average initially
169.                 mod = 1.1 * max(0.5, (1 - year_offset/80))  # Declining effect o
    ver time
170.             else:
171.                 # Very elderly improvements below average and declining over tim
    e
172.                 mod = 0.7 * max(0.3, (1 - year_offset/60))
173.
174.             return base_rate * mod
175.
176.         def _apply_gompertz_correction(self, age, rate):
177.             """
178.             Apply Gompertz correction to ensure mortality rates behave properly
    at high ages
179.
180.             Parameters:
181.             -----------
182.             age : int
183.                 The age
184.             rate : float
185.                 The mortality rate
186.
187.             Returns:
188.             --------
189.             float
190.                 The corrected mortality rate
191.             """
192.             # For ages below 80, no correction needed
193.             if age < 80:
194.                 return rate
195.
196.             # For ages 80+, ensure rates follow reasonable Gompertz pattern
197.             # without exceeding theoretical maximum
198.             if rate > 0.6:
199.                 # Use sigmoid function to asymptotically approach 1
200.                 # as age increases and mortality gets higher
201.                 sigmoid_factor = 1 / (1 + np.exp(-(rate - 0.85) * 15))
202.                 return 0.6 + 0.399 * sigmoid_factor
203.
```

```
204.           return rate
205.
206.       def project_forward(self):
207.           """Project mortality rates forward for 100 years"""
208.           print(f"Projecting mortality rates from {self.projection_start_year}
    to {self.projection_end_year}...")
209.
210.           for gender in ['M', 'F']:
211.               print(f"\nGenerating projections for gender: {gender}")
212.
213.               # Get baseline mortality rates from 2025
214.               baseline_year = self.projection_start_year
215.               if baseline_year not in self.projections[gender]['mortality_rate
    s']:
216.                   raise ValueError(f"No baseline mortality for {gender} in {ba
    seline_year}. "
217.                                    "Run initialize_from_task2() first.")
218.
219.               baseline_mortality = self.projections[gender]['mortality_rates']
    [baseline_year]
220.               ages = sorted(baseline_mortality.keys())
221.
222.               # Project for each 5-year interval
223.               for i, year in enumerate(self.projection_years[1:], 1):
224.                   self.projections[gender]['mortality_rates'][year] = {}
225.                   self.projections[gender]['improvement_rates'][year] = {}
226.
227.                   year_offset = year - self.projection_start_year
228.                   base_improvement = self.improvement_decay_function(year_offs
    et, gender)
229.                   print(f"  Year {year}: Base improvement rate = {base_improve
    ment:.4f}")
230.
231.                   # Apply improvements by age
232.                   for age in ages:
233.                       # Previous timepoint's mortality rate (may be 5 years ag
    o)
234.                       prev_year = self.projection_years[i-1]
235.                       prev_rate = self.projections[gender]['mortality_rates'][
    prev_year][age]
236.
237.                       # Calculate age-specific improvement over 5 years
238.                       age_improvement = self.age_specific_improvement_pattern(
    age, gender, year_offset)
```

```
                    five_year_factor = (1 - age_improvement) ** 5  # Compoun
    d for 5 years

                    # Apply improvement (reduction)
                    new_rate = prev_rate * five_year_factor

                    # Ensure sensible lower bounds
                    if age < 5:
                        new_rate = max(new_rate, 0.0001)
                    else:
                        min_rate = 0.0001 * (1.05 ** (age - 5))
                        new_rate = max(new_rate, min_rate)

                    # Apply Gompertz correction for high ages
                    new_rate = self._apply_gompertz_correction(age, new_rate
    )

                    # Store results
                    self.projections[gender]['mortality_rates'][year][age] =
     new_rate
                    self.projections[gender]['improvement_rates'][year][age]
     = 1 - five_year_factor

                # Calculate life expectancy
                self.calculate_life_expectancy(gender)

        def calculate_life_expectancy(self, gender):
            """Calculate period life expectancy for each projection year"""
            print(f"Calculating life expectancy for {gender}...")

            for year in self.projection_years:
                mortality = self.projections[gender]['mortality_rates'][year]
                ages = sorted(mortality.keys())
                rates = [mortality[age] for age in ages]

                # Create life table
                lx = [100000]  # Radix of 100,000 lives
                for q in rates:
                    lx.append(lx[-
    1] * (1 - min(q, 1.0)))  # Add survivors to next age

                # Calculate expectation of life
                T = sum(lx)  # Total person-years lived by the cohort
```

```
                ex = [sum(lx[i:]) / lx[i] if lx[i] > 0 else 0 for i in range(len(lx)-1)]

                # Store results
                self.life_expectancy[gender]['at_birth'][year] = ex[0]
                if 65 in ages:
                    idx_65 = ages.index(65)
                    self.life_expectancy[gender]['at_65'][year] = ex[idx_65]

            # Print summary
            start_year = self.projection_years[0]
            end_year = self.projection_years[-1]
            print(f"  Life expectancy at birth in {start_year}: {self.life_expectancy[gender]['at_birth'][start_year]:.2f} years")
            print(f"  Life expectancy at birth in {end_year}: {self.life_expectancy[gender]['at_birth'][end_year]:.2f} years")
            print(f"  Gain in e(0) over projection period: {self.life_expectancy[gender]['at_birth'][end_year] - self.life_expectancy[gender]['at_birth'][start_year]:.2f} years")

        def run_projection(self):
            """Run the full projection process"""
            self.initialize_from_task2()
            self.project_forward()
            return self.projections, self.life_expectancy
```

8. 2025-2125 年保险人口长期死亡率趋势预测分析

```
"""
Long-Term Mortality Trend Forecast (2025-2125)
Main execution script to run the long-term mortality projections
"""

import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime

# Add parent directory to path
sys.path.append('../../问题 2 报告/改进后')

# Import local modules
```

```
from long_term_mortality_model import LongTermMortalityModel
from visualization import MortalityVisualizer

# Configure paths
CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_DIR = os.path.join(CURRENT_DIR, '..', '..', '..', '【选题 1 数据】某保险产品死亡率.xlsx')
OUTPUT_DIR = CURRENT_DIR

def format_mortality_table(projections, years=[2025, 2075, 2125], ages=[0, 20, 40, 60, 70, 80]):
    """
    Format mortality data into a presentable table

    Parameters:
    -----------
    projections : dict
        The projection results from the model
    years : list
        Years to include in the table
    ages : list
        Ages to include in the table

    Returns:
    --------
    pandas.DataFrame
        Formatted table of mortality rates
    """
    data = []

    for age in ages:
        row = {'Age': age}

        for gender in ['M', 'F']:
            gender_label = 'Male' if gender == 'M' else 'Female'

            for year in years:
                if year in projections[gender]['mortality_rates']:
                    if age in projections[gender]['mortality_rates'][year]:
                        rate = projections[gender]['mortality_rates'][year][age]
                        row[f'{gender_label} {year}'] = rate

        data.append(row)
```

```
58.
59.     # Create DataFrame and format
60.     df = pd.DataFrame(data)
61.
62.     # Format mortality rates as percentages
63.     for col in df.columns:
64.         if col != 'Age':
65.             df[col] = df[col].map('{:.6f}'.format)
66.
67.     return df
68.
69. def format_life_expectancy_table(life_expectancy, years=None):
70.     """
71.     Format life expectancy data into a presentable table
72.
73.     Parameters:
74.     -----------
75.     life_expectancy : dict
76.         The life expectancy results from the model
77.     years : list
78.         Years to include in the table (default: all available years)
79.
80.     Returns:
81.     --------
82.     pandas.DataFrame
83.         Formatted table of life expectancy values
84.     """
85.     if years is None:
86.         years = sorted(life_expectancy['M']['at_birth'].keys())
87.
88.     # Filter to include only some years if too many
89.     if len(years) > 10:
90.         years = [years[0]] + [y for y in years if y % 25 == 0] + [years[-1]]
91.
92.     data = []
93.
94.     for year in years:
95.         row = {'Year': year}
96.
97.         for gender in ['M', 'F']:
98.             gender_label = 'Male' if gender == 'M' else 'Female'
99.
100.              if year in life_expectancy[gender]['at_birth']:
```

```
                    row[f'{gender_label} e(0)'] = life_expectancy[gender]['at_birth'][year]

                if 'at_65' in life_expectancy[gender] and year in life_expectancy[gender]['at_65']:
                    row[f'{gender_label} e(65)'] = life_expectancy[gender]['at_65'][year]

            # Calculate gender gap
            if 'Male e(0)' in row and 'Female e(0)' in row:
                row['Gender Gap'] = row['Female e(0)'] - row['Male e(0)']

            data.append(row)

        # Create DataFrame and format
        df = pd.DataFrame(data)

        # Format life expectancy values with 2 decimal places
        for col in df.columns:
            if col != 'Year':
                df[col] = df[col].map('{:.2f}'.format)

        return df

    def save_projection_to_csv(projections, life_expectancy, output_dir):
        """
        Save projection results to CSV files

        Parameters:
        -----------
        projections : dict
            The projection results from the model
        life_expectancy : dict
            The life expectancy results from the model
        output_dir : str
            Directory to save the files

        Returns:
        --------
        list
            Paths to the saved files
        """
        os.makedirs(output_dir, exist_ok=True)
        saved_files = []
```

```
142.
143.       # Save mortality rates table
144.       mortality_table = format_mortality_table(projections)
145.       mortality_path = os.path.join(output_dir, 'mortality_projection_table.cs
    v')
146.       mortality_table.to_csv(mortality_path, index=False)
147.       saved_files.append(mortality_path)
148.
149.       # Save life expectancy table
150.       life_exp_table = format_life_expectancy_table(life_expectancy)
151.       life_exp_path = os.path.join(output_dir, 'life_expectancy_projection.csv
    ')
152.       life_exp_table.to_csv(life_exp_path, index=False)
153.       saved_files.append(life_exp_path)
154.
155.       # Save detailed projections by gender
156.       for gender in ['M', 'F']:
157.           gender_label = 'male' if gender == 'M' else 'female'
158.
159.           # Extract all mortality rates by year and age
160.           data = []
161.
162.           for year in sorted(projections[gender]['mortality_rates'].keys()):
163.               mortality = projections[gender]['mortality_rates'][year]
164.
165.               for age in sorted(mortality.keys()):
166.                   data.append({
167.                       'Year': year,
168.                       'Age': age,
169.                       'Mortality_Rate': mortality[age]
170.                   })
171.
172.           # Create and save DataFrame
173.           df = pd.DataFrame(data)
174.           detailed_path = os.path.join(output_dir, f'detailed_{gender_label}_p
    rojection.csv')
175.           df.to_csv(detailed_path, index=False)
176.           saved_files.append(detailed_path)
177.
178.       print(f"Saved {len(saved_files)} projection files:")
179.       for file in saved_files:
180.           print(f"  - {file}")
181.
182.       return saved_files
```

```
183.
184.   def generate_summary_report(model, output_dir):
185.       """
186.       Generate a summary report of the long-term projection
187.
188.       Parameters:
189.       -----------
190.       model : LongTermMortalityModel
191.           The model containing projections
192.       output_dir : str
193.           Directory to save the report
194.
195.       Returns:
196.       --------
197.       str
198.           Path to the saved report
199.       """
200.       report = []
201.       report.append("="*70)
202.       report.append("LONG-TERM MORTALITY TREND FORECAST (2025-2125)")
203.       report.append("="*70)
204.       report.append("")
205.
206.       # 1. Overview
207.       report.append("1. OVERVIEW")
208.       report.append("-"*50)
209.       report.append("This report presents a century-
    long mortality projection for the")
210.       report.append("insured population, based on the 2010-
    2021 experience data and")
211.       report.append("extending the short-term model (Lee-Carter + Gompertz-
    Makeham)")
212.       report.append("through 2125.")
213.       report.append("")
214.
215.       # 2. Methodology
216.       report.append("2. METHODOLOGY")
217.       report.append("-"*50)
218.       report.append("The projection methodology follows these key principles:"
    )
219.       report.append("  - Initial mortality patterns from Task 2's 2025 project
    ion")
220.       report.append("  - Exponential decay of improvement rates over time")
```

```
        report.append("  - Age-specific improvement patterns with higher rates for younger ages")
        report.append("  - Modified Gompertz-Logistic pattern for elderly ages")
        report.append("  - Focus on ages 0-100 where data quality is most reliable")
        report.append("  - Persistent gender gap throughout the projection period")
        report.append("")

        # 3. Key Parameters
        report.append("3. KEY PARAMETERS")
        report.append("-"*50)
        report.append("Long-term improvement parameters:")

        for gender in ['M', 'F']:
            gender_label = 'Male' if gender == 'M' else 'Female'
            params = model.improvement_params[gender]

            report.append(f"  {gender_label}:")
            report.append(f"    - Initial annual improvement rate: {params['initial_rate']*100:.2f}%")
            report.append(f"    - Long-term asymptotic rate: {params['asymptotic_rate']*100:.2f}%")
            report.append(f"    - Half-life of improvement decay: {params['half_life_years']} years")

        report.append("")

        # 4. Key Results
        report.append("4. KEY RESULTS")
        report.append("-"*50)

        # Life expectancy at birth in 2025 and 2125
        start_year = model.projection_years[0]
        end_year = model.projection_years[-1]

        report.append(f"Life expectancy at birth:")
        for gender in ['M', 'F']:
            gender_label = 'Male' if gender == 'M' else 'Female'
            e0_start = model.life_expectancy[gender]['at_birth'][start_year]
            e0_end = model.life_expectancy[gender]['at_birth'][end_year]
            gain = e0_end - e0_start

            report.append(f"  {gender_label}:")
```

```
            report.append(f"    - {start_year}: {e0_start:.2f} years")
            report.append(f"    - {end_year}: {e0_end:.2f} years")
            report.append(f"    - Century gain: +{gain:.2f} years")

        report.append("")
        report.append("Gender gap in life expectancy at birth:")
        gap_start = model.life_expectancy['F']['at_birth'][start_year] - model.life_expectancy['M']['at_birth'][start_year]
        gap_end = model.life_expectancy['F']['at_birth'][end_year] - model.life_expectancy['M']['at_birth'][end_year]
        report.append(f"  - {start_year}: {gap_start:.2f} years")
        report.append(f"  - {end_year}: {gap_end:.2f} years")

        report.append("")
        report.append("Mortality rates at key ages (2125):")
        key_ages = [0, 20, 40, 60, 70, 80]

        for gender in ['M', 'F']:
            gender_label = 'Male' if gender == 'M' else 'Female'
            report.append(f"  {gender_label}:")

            mortality_2125 = model.projections[gender]['mortality_rates'][end_year]
            for age in key_ages:
                rate = mortality_2125[age]
                report.append(f"    - Age {age}: {rate:.6f}")

        report.append("")
        report.append("")

        # 5. Conclusion
        report.append("5. CONCLUSION")
        report.append("-"*50)
        report.append("The century-long projection demonstrates steadily improving")
        report.append("mortality rates for both genders, with females maintaining")
        report.append("their survival advantage. Improvement rates gradually slow")
        report.append("over time, reflecting biological constraints and diminishing")
        report.append("returns from medical advances.")
        report.append("")
        report.append("The results show a plausible path for mortality development,")
```

```
        report.append("with no abrupt changes. The mortality pattern follows a")
        report.append("modified Gompertz-
    Logistic model that ensures rates remain")
        report.append("biologically plausible while preserving the exponential")
        report.append("age-pattern relationship.")
        report.append("")
        report.append("Note: The analysis focuses on ages 0-
    100 where data quality")
        report.append("is most reliable. Projections beyond age 100 should be")
        report.append("interpreted with appropriate caution due to data sparsity
    .")
        report.append("")

        # 6. Generated Files
        report.append("6. GENERATED FILES")
        report.append("-"*50)
        report.append("Data tables:")
        report.append("  - mortality_projection_table.csv")
        report.append("  - life_expectancy_projection.csv")
        report.append("  - detailed_male_projection.csv")
        report.append("  - detailed_female_projection.csv")
        report.append("")
        report.append("Visualizations:")
        report.append("  - life_expectancy_trend.png")
        report.append("  - mortality_curves.png")
        report.append("  - improvement_rates.png")
        report.append("  - age_improvements_2075.png")
        report.append("  - advanced_age_mortality.png")
        report.append("  - mortality_surface_M.png")
        report.append("  - mortality_surface_F.png")
        report.append("")

        # Save the report
        report_path = os.path.join(output_dir, 'long_term_projection_report.txt'
    )
        with open(report_path, 'w', encoding='utf-8') as f:
            f.write('\n'.join(report))

        print(f"Generated summary report: {report_path}")

        return report_path

    def main():
        print("="*70)
```

```
335.       print("LONG-TERM MORTALITY TREND FORECAST (2025-2125)")
336.       print("="*70)
337.
338.       # Record start time
339.       start_time = datetime.now()
340.       print(f"Started at: {start_time.strftime('%Y-%m-%d %H:%M:%S')}")
341.
342.       # Check if the data file exists
343.       if not os.path.exists(DATA_DIR):
344.           print(f"Error: Data file not found at {DATA_DIR}")
345.           print("Please check the path and try again.")
346.           return
347.
348.       try:
349.           # Initialize the model
350.           print("\nInitializing model...")
351.           model = LongTermMortalityModel(data_path=DATA_DIR)
352.
353.           # Run the projection
354.           print("\nRunning 100-year projection...")
355.           projections, life_expectancy = model.run_projection()
356.
357.           # Generate visualizations
358.           print("\nGenerating visualizations...")
359.           visualizer = MortalityVisualizer(model, save_dir=OUTPUT_DIR)
360.           vis_paths = visualizer.generate_all_visualizations()
361.
362.           # Save projection results to CSV
363.           print("\nSaving projection data...")
364.           data_paths = save_projection_to_csv(projections, life_expectancy, OU
    TPUT_DIR)
365.
366.           # Generate summary report
367.           print("\nGenerating summary report...")
368.           report_path = generate_summary_report(model, OUTPUT_DIR)
369.
370.           # Display execution time
371.           end_time = datetime.now()
372.           duration = end_time - start_time
373.           print("\nExecution completed successfully!")
374.           print(f"Duration: {duration}")
375.           print("="*70)
376.
377.       except Exception as e:
```

```
            print(f"\nError during execution: {str(e)}")
            import traceback
            traceback.print_exc()

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