哈尔滨工业大学深圳校区

统计计算课程论文

题　　目 基于统计计算方法

复现与拓展《单位扩展指数分布与应用》

姓　　名 龚仔航 林思涛 钟问重 郝子翔

学　　号 2208101 /29/34/31/12

学　　院 理学院

专　　业 数据科学与大数据技术

授 课 教 师 冯峥晖

提 交 日 期 2024 年 12 月 5 日

组员分工：

龚仔航：负责论文前三章 UEXE 模型相关理论数学推导以及图像绘制，第五章 Bootstrap 重抽样应用。

林思涛：负责复现模拟研究的模型样本生成、对参数的点估计和模型在现实中的应用部分及图像绘制。

钟问重：负责复现原论文中关于区间估计的覆盖率以及平均长度的模拟研究部分及图像绘制。

郝子翔：负责基于原论文复现两数据集模型应用与评估及图像绘制。

摘　要

本文**复现与拓展**了论文《Unit extended exponential distribution with applications》[1] 的核心研究内容，该论文系统介绍了一种双参数寿命分布的单位扩展指数分布，即 **UExE 分布**（Unit Extended Exponential Distribution），并研究了其统计特性及其实际应用价值。

本文首先通过数学理论推导证明了该分布具有闭合形式的累积分布函数以及在不同参数下，UExE 分布的统计量（如矩、均值、方差、偏斜度和峰度）、生存函数和风险函数呈现多样性特征，能够灵活地模拟多种形状的寿命数据。

基于原论文提出的**极大似然估计（MLE）**的参数点估计和利用渐近正态性构建参数的置信区间的方法。我们合理运用**牛顿-拉夫逊方法**于极大似然估计任务中以及 **Fisher 信息矩阵**于置信区间估计任务当中，高效的解决了极大似然估计与置信区间估计任务中效率低、数值不稳定等问题。我们同时创造性的提出了**基于二分法的逆变换抽样方法**，实现了高效的从复杂分布中快速产生样本，并结合**蒙特卡洛模拟方法**，评估研究了上述估计方法的效果并说明了样本量对其影响。结果表明随着样本量的增加，参数估计偏差显著减小，覆盖率逐渐逼近理论值，从而验证了 UExE 模型参数估计方法的有效性和渐近稳定性。

按原论文的研究思路，我们复现了两个实际数据集的应用，并使用 **Akaike 信息准则**等多种评估指标以及 **K-S 检验**等多种拟合优度检验方法，全面地评估了 UExE 分布与其他九种竞争模型的优劣，并结合可视化得出 UExE 分布相较于其他模型表现出优越的拟合效果，展示了其在实际数据建模中的显著优势。我们额外利用该两组小样本数据使用 **Bootstrap 重抽样方法**计算了各自估计量的偏差与方差，指出了极大似然估计在该模型参数估计中存在一定的局限性。

此外，我们额外使用了 2024 年 **“泰迪杯”** 数据分析技能赛中的数据用以评估模型在真实的企业生产大样本数据下的表现。本文在验证数据集上做了相关评估，科学验证了 UEXE 模型在该种数据拟合上仍有较高的鲁棒性、高效性与准确性，从而更加科学地指导企业生产活动更加合理分配资源和决策。

总体而言，我们对原论文的**复现和扩展**不仅验证了 UExE 分布的统计性质和参数估计方法的可靠性，还通过实际数据分析进一步凸显了其在复杂数据建模中的潜力，为扩展指数分布在统计学和应用领域的研究奠定了坚实基础。

关键词：扩展指数分布；Bootstrap 重抽样；逆变换抽样；极大似然估计

目　录

第 1 章　论文背景介绍

1.1　论文背景

生命周期数据分析是统计学中专门研究事物或产品在其生命周期内行为和特征的一种方法，广泛应用于工程、医学、社会科学等领域。例如，在机械设备的故障预测中，我们希望通过数据分析来确定设备的使用寿命及其失效风险；在医学研究中，可以用于分析患者生存时间或药物疗效持续时间。这种分析的核心目标通常是理解事件发生的概率随时间变化的规律，比如 “某设备在某一时间点失效的风险是多少”。

在生命周期数据分析中，风险函数是一个关键的数学工具。它描述了在某一时间已经存活下来的前提下，事物在接下来的瞬间失效的概率。风险函数的形状反映了失效风险是如何随着时间变化的：

(1) 单调递增的风险函数表明，随着时间的推移，失效的风险越来越大（例如老化过程）。

(2) 单调递减的风险函数则意味着，随着时间推移，事物变得更加可靠（例如某些测试过程筛选掉了不可靠的个体）。

(3) 恒定风险函数表明失效的风险不随时间变化（如某些随机失效现象）。

因此，生命周期数据分析通常依赖于具有单调风险函数的模型，如伽马分布。然而，某些模型（如伽马模型）缺乏封闭形式的风险函数(即无法通过简单公式直接计算)，因此需要通过数值积分计算。为了解决这一问题，能够使用封闭形式的函数. Gómez 等（2014）提出了扩展指数分布（ExE），具有两个参数 δ 与 λ。其累积分布函数（CDF）和概率密度函数（PDF）分别如下：

$$\mathbf{H}(\mathbf{y};\delta,\lambda) = \mathbf{1} - \frac{[\delta + \lambda + \delta\lambda\mathbf{y}]}{\delta + \lambda}\mathbf{e}^{-\delta\mathbf{y}}\ ;\ \ where\ \mathbf{y} > \mathbf{0}, \delta, \lambda > \mathbf{0} \tag{1-1}$$

$$\mathbf{h}(\mathbf{y};\delta,\lambda) = \frac{\delta^2\,[\mathbf{1} + \lambda\mathbf{y}]}{\delta + \lambda}\mathbf{e}^{-\delta\mathbf{y}} \tag{1-2}$$

ExE 模型（Exponential-Exponential 模型）是一种非常灵活的寿命分布模型，因为它能够包含两个经典的寿命分布作为特例。当参数 $\delta = 0$ 时，ExE 模型退化为指数分布（E 模型），这是一种常用于描述恒定失效率的简单寿命分布；而当参数 $\lambda = 1$ 时，ExE 模型退化为由 Lindley（1958 年）提出的 Lindley 分布，这是

一种更复杂的寿命分布形式，适用于一些更复杂的失效率情景。通过调整参数 δ 和 λ，ExE 模型能够灵活适应不同的寿命分布需求，体现了其广泛的适用性和建模能力。

近年来，研究者们越来越关注描述那些支持范围有限的分布，主要是因为许多实际问题中随机变量的取值范围有限，如比例数据、浓度或存活率常限制在 [0,1] 区间内。传统分布模型虽然广泛应用，但其定义范围往往过宽，直接使用可能导致不适配实际数据的偏差，或引入复杂的数学处理。而支持范围有限的模型，尤其是定义在单位区间内的模型，能够更自然地捕捉这些数据的特性，避免了额外截断或变换的复杂性。

单位模型研究因此逐渐占据主导地位。这类模型不仅适用于描述单位区间内的数据，还具有高度的灵活性和拓展性，可以通过调整参数适配多种实际分布形状。此外，这些模型弥补了传统分布在比例数据分析、单位区间回归等领域的空白，为研究复杂现象提供了强有力的工具。这些特点使得单位模型成为当前统计学研究的重要方向之一。

鉴于 **ExE** 模型在生命周期数据分析中的灵活性，以及单位区间模型在处理有限支持范围数据中的重要性，将这两者结合起来，有望为数据建模提供新的视角和工具，展现出潜在的研究价值和广泛的应用前景。

1.2 研究内容

为了充分结合 **ExE** 模型的灵活性和单位区间模型在处理有限支持范围数据中的重要性，论文提出了一种新颖的单位分布 (即定义在 (0,1) 区间内) 的扩展指数分布（ExE），该分布被称为 UExE 分布。

作为两种模型的复合模型，我们将在接下来的文章中研究说明其具有以下重要特性:

(I) 它的累积分布函数（CDF）具有闭合形式且表达简单。

(II) 它可以表现出 J 型、U 型以及浴缸型的危险率函数（HRF）。

(III) 与其他常见的单位区间分布相比，它在拟合数据时可能具有更高的准确性。

其次本文将研究模型的参数 $\delta\lambda$ 的估计方法以及置信区间，并在论文提供的真实数据集上使用 **Bootstrap** 重抽样方法，**Anderson-Darling (A)*** 、**Cramer-Von Mises (W)*** 和 **Kolmogorov-Smirnov** 拟合优度检验方法对模型评估效果以及拟合质量做进一步的研究。

同时，针对论文中使用的数据集较小不够贴合实际数据的不足，我们进一步使用 2024 年（第 7 届）“泰迪杯”数据分析技能赛中，A 题–自动化生产线数据分析的某生产企业两条生产线全年加工处理过程中各个工序的生产线数据对 **UEXE** 模型在更大更真实数据进行研究。

1.3 研究方法

在第2章中，我们首先通过数学推导证明以及蒙特卡洛模拟等方法，对UExE模型的性质做了一定的研究，初步说明了其具有前文在1.2中所提到的诸如灵活性、封闭形式的累积分布函数以及多性态的危险率函数等优秀特性。

对于 **UEXE** 这样的双参数模型，模型参数 $\delta\lambda$ 的估计至关重要。因此在第3章中，我们针对模型参数估计这一问题做了着重研究。具体而言，我们使用极大似然估计进行模型参数的点估计，以及利用极大似然估计的渐进正态性对参数的进行置信区间估计。在这一章节中我们对上述两个方法提供了详细的数学理论推导，同时我们也给出了 K-S 拟合优度检验（Kolmogorov-Smirnov Goodness-of-Fit Test）的理论说明。上述理论推导与说明将用于后续进一步的研究。

在第4章模拟研究部分，我们重点对于在第3章中给出的点估计和置信区间估计方法做了评估和说明。我们使用**蒙特卡洛模拟**方法评估了点估计方法的偏差和均方误差以及置信区间估计的覆盖率和置信区间长度。具体而言，我们首相创造性地使用**基于二分法的逆变换抽样方法**生成了大量样本，随后基于这些样本我们使用牛顿迭代法计算极大似然估计的点估计并计算偏差与均方误差，同时使用渐进正态性以及 Fisher 信息矩阵计算置信区间并计算覆盖率以及平均置信区间长度。具体算法与结果可详见第4章。

最后在第5章模型应用部分，我们着重使用多组真实数据对 **UEXE** 模型性能与拟合效果等做了分析。具体而言，我们基于论文中提供的两组小样本原始数据，对 **UEXE** 模型和 **Beta** 分布、**Kumaraswamy** 分布等多种模型，在 **Akaike** 信息准则、**Anderson-Darling**（A*）检验以及 **Kolmogorov-Smirnov**（K-S）检验等多种指标与检验方法上做了评估与对比分析。同时在原论文基础上我们针对小样本使用 Bootstrap 重抽样方法对模型参数估计进行偏差与方差的计算。为了更全面的评估模型性能，我们使用了“泰迪杯”数据分析技能赛中所提供的大样本真实数据对于模型的拟合与预测效果做了更进一步的分析。具体结果与可视化图片见第5章。

第 2 章　UExE 模型的性质研究

2.1　UExE 模型的提出

因此基于章节1的阐述与要求，我们可以利用转换函数 $\mathbf{X}=\mathbf{e}^{-\mathbf{y}}$ 将自变量范围变换到 $(0,1)$ 之间，从而将 **EXE** 概率模型转化为 **UExE** 概率模型，具体而言，结合式1-1以及式1-2变换后的累积分布函数（CDF）和概率密度函数（PDF）分别如下：

$$\mathbf{G}(\mathbf{x};\delta,\lambda)=\left(\mathbf{1}-\frac{\delta\lambda\mathbf{log}(\mathbf{x})}{\delta+\lambda}\right)\mathbf{x}^{\delta};\ \ where\ \mathbf{x}\in(\mathbf{0},\mathbf{1}),\delta,\lambda>\mathbf{0} \tag{2-1}$$

$$\mathbf{g}(\mathbf{x};\delta,\lambda)=\frac{\delta^2[\mathbf{1}-\lambda\mathbf{log}(\mathbf{x})]}{\delta+\lambda}\mathbf{x}^{\delta-1};\ \ where\ \mathbf{x}\in(\mathbf{0},\mathbf{1}),\delta,\lambda>\mathbf{0} \tag{2-2}$$

同时容易证明如下定理：

定理 2.1 UExE 概率模型的 PDF 在 $(\mathbf{0},\mathbf{1})$ 区间内积分值为 $\mathbf{1}$，即 $\int_0^1\mathbf{g}(\mathbf{x};\delta,\lambda)\ \mathbf{dx}=\mathbf{1}$

结合定理2.1可以发现，UExE 模型的累积分布函数（CDF）是一个封闭的表达简单的函数，复合上述我们对于模型的要求。

绘制 UExE 累积分布函数关于不同参数的图片可以发现，UExE 模型的分布随着参数 $\delta\lambda$ 的不同取值，可以有不同的表现，具体可以见图2-1。

为了进一步展现该该模型的优秀性质，本文将进一步从生存函数、风险函

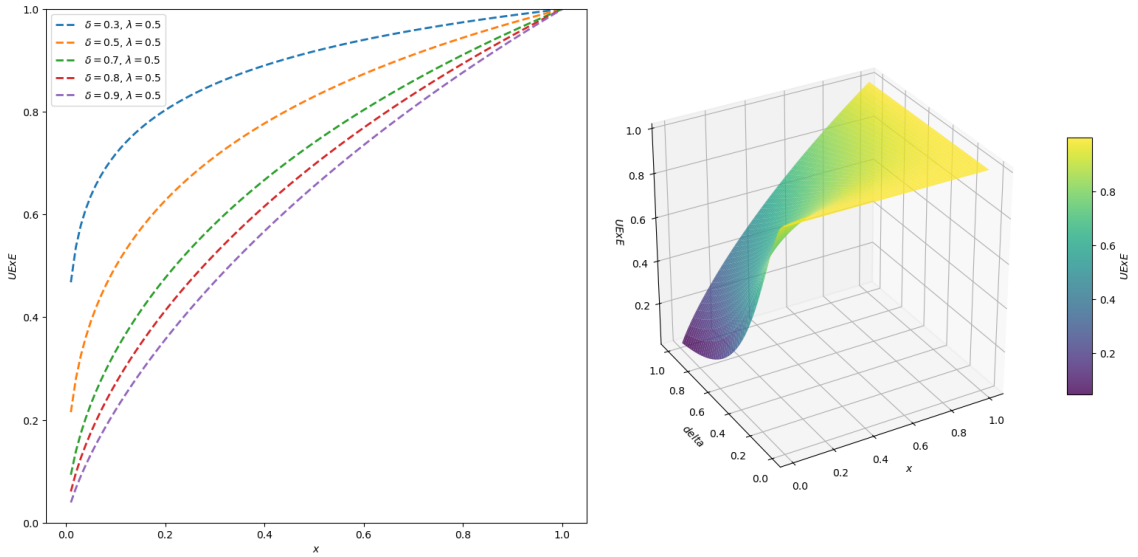


图 2-1　在 λ = 0.5 时 UExE 的累积分布函数的 2D 与 3D 图像

数以及该概率函数的统计特性。

2.1.1 UExE 模型的生存函数 (SF)

生存函数（Survival Function, SF）表示在某个特定时间 **t** 后，系统或物品“存活”的概率。它用于表示“事件发生的相反过程”，例如一个设备持续运行不发生故障的概率，或者疾病的存活概率。因此对于 UExE 模型而言, 其 SF 函数如下：

$$\mathbf{S}(\mathbf{x};\delta,\lambda) = 1-\mathbf{G}(\mathbf{x};\delta,\lambda) = \mathbf{1}-\left(\mathbf{1}-\frac{\delta\lambda\log(\mathbf{x})}{\delta+\lambda}\right)\mathbf{x}^{\delta};\ \ where\ \mathbf{x}\in(\mathbf{0},\mathbf{1}),\delta,\lambda>\mathbf{0} \quad (2\text{-}3)$$

与 UExE 模型的累积分布函数（CDF）同理，我们绘制其生存函数 (SF)，见图2-2, 在不同参数配置下的表现同样可以发现，生存函数在不同的参数设置下，生存函数分布有着非常平稳连续的变化，这也表明该模型在现实意义下有着较为优秀的适应性。

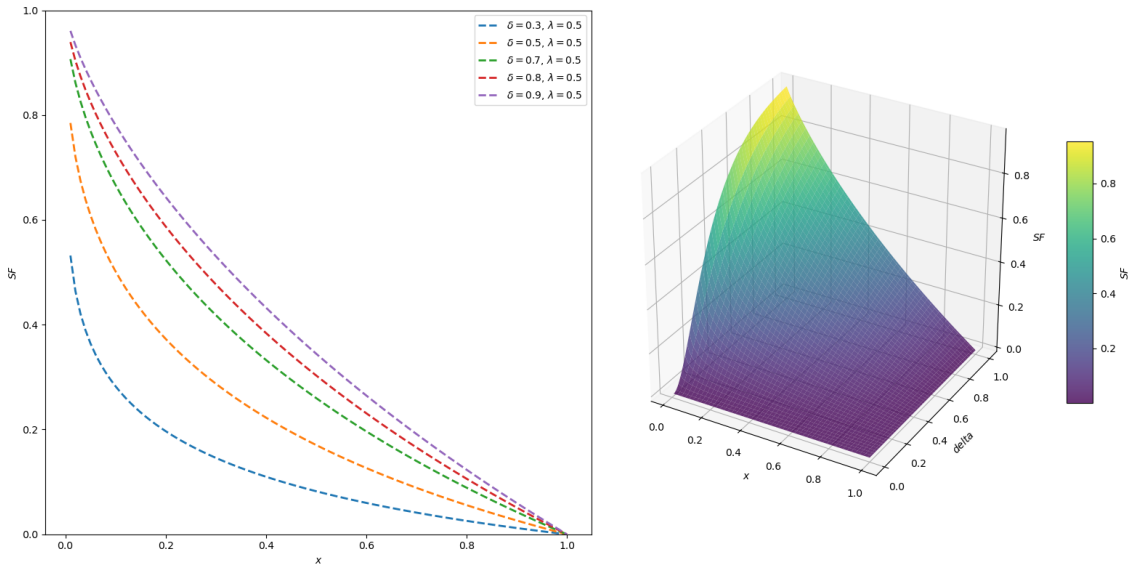


图 2-2　在 λ = 0.5 时 UExE 的生存函数 (SF) 的 2D 与 3D 图像

2.1.2 UExE 模型的风险函数 (HRF)

风险函数（Hazard Rate Function, HRF）也叫瞬时风险函数，它描述了在时间 **t** 时刻发生某个事件的即时发生率，给定此时之前还没有发生事件的条件。换句话说，它表示在特定时刻，事件发生的“强度”或“速率”。它在生存分析中非常重要，尤其是在医疗、工程等领域，能为决策提供关于事件发生时刻的重要信息。对于 UExE 概率模型而言，其表达式如下：

$$\mathbf{r}(\mathbf{x};\delta,\lambda)=\frac{\mathbf{g}(\mathbf{x};\delta,\lambda)}{1-\mathbf{G}(\mathbf{x};\delta,\lambda)}=\frac{\delta^2[\mathbf{1}-\lambda\mathbf{log}(\mathbf{x})]}{(\delta+\lambda)[\mathbf{x}^{1-\delta}-\mathbf{x}]+\delta\lambda\mathbf{log}(\mathbf{x})};\ where\ \mathbf{x}\in(\mathbf{0},\mathbf{1}),\delta,\lambda>\mathbf{0} \tag{2-4}$$

在 UExE 模型中，从图2-3可以观察到，其概率密度函数（PDF）可以呈现下降、左偏、右偏以及单峰的形状，而其危险率函数（HRF）则可能表现为 J 型、U 型或浴缸型。因此该模型具有较强的灵活性以及广泛的适应性。

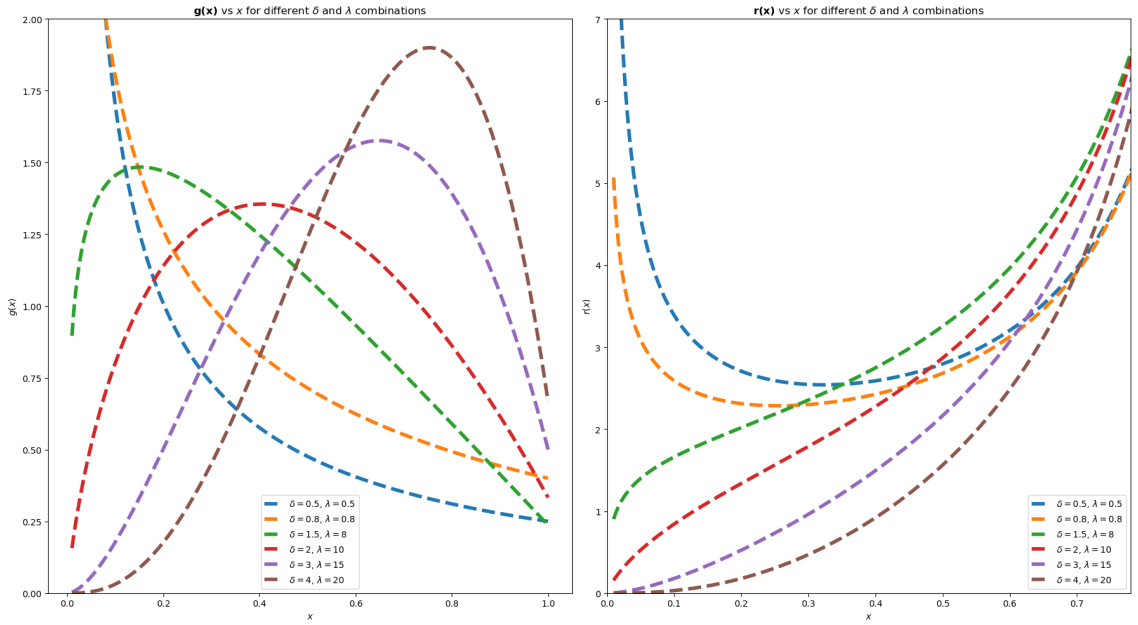


图 2-3　UExE 的的概率密度函数（左）与风险函数 (右)

2.1.3　UExE 概率函数的统计性质

本节深入探讨了 UExE 分布的重要统计特性，包括矩、均值、方差、偏度、峰度、变异系数、离散指数等常见统计量，用以进一步说明 UEXE 分布具有较为丰富的形态与性质。

2.1.3.1　常规矩

本节将详细介绍几个用以描述一个分布的普通统计量，它们是理解和描述统计分布特性的重要工具。这些统计量不仅帮助解释数据分布的核心特征，还为研究人员和统计学家提供了描述分布形状、中心位置以及离散程度的有力手段。在实际应用中，这些统计量被广泛用于分析数据行为，识别分布模式，并支持决策制定。以下是这些统计量的详细定义：

(1) 均值：$\{\mu_x=\mu_1'\}$ 均值表示分布的中心位置。它衡量数据的集中趋势，是

描述分布最基本的特性之一。

(2) 方差：$\{\sigma_x^2 = \mu_2^{'} - (\mu_1^{'})^2\}$ 表示数据的离散程度，用于衡量数据值偏离均值的程度。

(3) 偏度:$\{\gamma_1 = \frac{[\mu_3^{'} - 3\mu_1^{'}\mu_3^{'} + 2(\mu_1^{'})^3]}{\sigma_x^3}\}$ 反映分布的对称性或偏斜程度。偏度值为正表示分布右偏（长尾在右侧），值为负表示左偏（长尾在左侧）。

(4) 峰度:$\{\gamma_2 = \frac{[\mu_4^{'} - 4\mu_1^{'}\mu_3^{'} + 6(\mu_1^{'})^2\mu_2^{'} - 3(\mu_1^{'})^4]}{\sigma_x^4}\}$ 描述分布的峰态或尾部厚度，通常用于区分不同分布的尖峰或尾部行为。一般来说，较高的峰度值表明分布有更尖锐的峰和更厚的尾（重尾分布）。

(5) 变异系数:$\{CV = \frac{\sigma_x}{\mu_1^{'}}\}$ 变异系数通过标准差与均值的比值，提供了一个无量纲的测度，可以用于不同单位、量纲或数量级的数据集之间的比较。较大的变异系数说明数据分布具有较高的相对变异性，数据点围绕均值的分散程度较高；较小的变异系数则表示数据点相对更加集中，数据分布的相对一致性较强。

(6) 离散指数:$\{ID = \frac{\mu_1^{'}}{\sigma_x}\}$ 离散指数是变异性与集中性的另一种比值形式，可以理解为均值在离散程度上的相对表现。

在上述定义中，$\mu_1^{'}, \mu_2^{'}, \mu_3^{'}$, 表示分布的各阶矩。而关于 UExE 概率函数的各阶矩的计算，我们有如下定理：

定理 2.2 UExE 分布的 **r** 阶矩表达式为：

$$\mu_r^{'} = \frac{\delta^2(r + \delta + \lambda)}{(\delta + \lambda)(r + \delta)^2}\ ;\ \ r = 1, 2, 3 \tag{2-5}$$

证明:

$$\mu_r^{'} = \mathbb{E}[X^r] = \int_0^1 X^r \mathbf{g}\left(\mathbf{x}; \delta, \lambda\right)\ dx \tag{2-6}$$

$$= \frac{\delta^2}{\delta + \lambda} \int_0^1 [1 - \lambda \mathbf{log}(x)] x^{r+\delta-1}\ dx \tag{2-7}$$

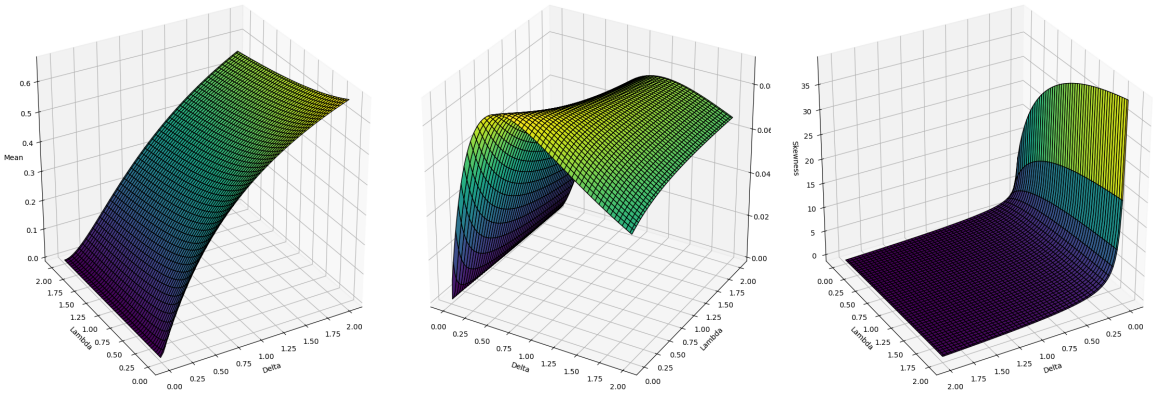
$$= \frac{\delta^2}{\delta + \lambda} [\int_0^1 x^{r+\delta-1}\ dx - \lambda \int_0^1 x^{r+\delta-1} \mathbf{log}(x)\ dx] \tag{2-8}$$

$$= \frac{\delta^2}{\delta + \lambda} [\frac{1}{r + \delta} + \frac{\lambda}{(r + \delta)^2}] \tag{2-9}$$

$$\tag{2-10}$$

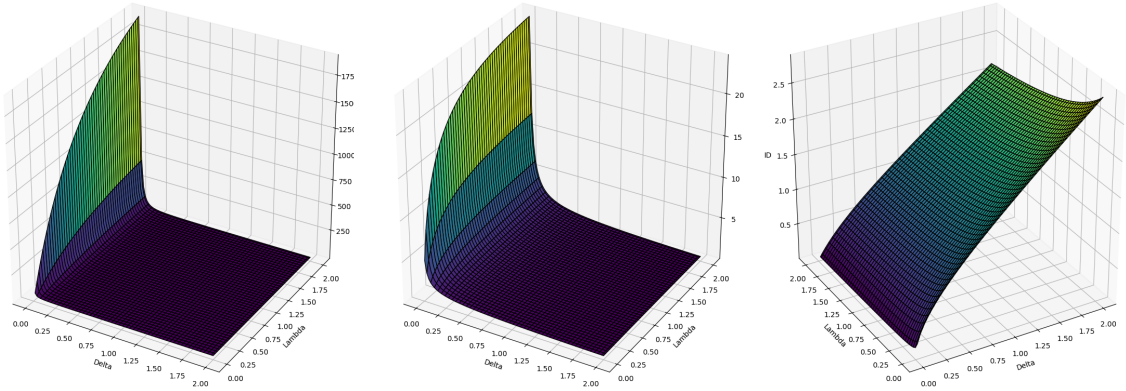
由此，我们证明了定理2.2。

进一步，我们在 $(0, 2)$ 区间内遍历 $\delta\lambda$ 的值，观察参数对应上述统计量的影响，具体结果见图:



(a) 均值　　(b) 方差　　(c) 偏度

图 2-4　均值、方差、偏度 3D 变化图



(a) 峰度　　(b) 变异系数　　(c) 离散指数

图 2-5　峰度、变异系数、离散指数 3D 变化图

从图2-4与2-5 中，可以观察到所选参数值对数据分布特性产生了显著的影响。具体来说，随着参数的变化，均值的减小表明数据的中心趋向于较低的值，而分布的其他统计特性则表现出相反的趋势。偏度、峰度、变异系数（CV）和离散指数（ID）均有所增加，而方差则呈现出反常的变化。

UExE 模型的研究为概率分布的理论发展和实际应用提供了重要的参考。通过对其累积分布函数（CDF）、生存函数（SF）、风险函数（HRF）以及统计特性（如均值、方差、偏度等）的深入分析，我们证明了该模型在不同参数下具有很强的灵活性和适应性。这使得 UExE 模型能够广泛应用于生存分析、风险管理、设备故障预测、疾病分析等领域，为实际决策提供精确的支持。其可调参数使其能够适应各种数据分布特性，从而在多个实际场景中展现出强大的预测能力和广泛的应用潜力。

第 3 章　参数估计与分布拟合

在该部分，我们将对于提出的 **UExE** 概率函数的参数 δ, λ 进行估计。分别从极大似然估计以及置信区间来进行估计。同时对于真实数据的拟合，我们这里介绍 K-S 单样本拟合优度检验来评估模型拟合效果

3.1　点估计：极大似然估计

极大似然估计（Maximum Likelihood Estimation, **MLE**）是一种广泛应用的技术，以其一致性、高效性和渐近正态性而闻名。该方法通过分析观测数据，最大化似然函数，从而确定统计模型的最优参数值，即在假定的分布下，找到最能解释观测数据的参数。

设 X_1, X_2, X_n 是服从参数为 δ 和 λ 的 UExE 概率分布的 **n** 个随机变量，并设 x_1, x_2, x_n 是其所对应的随机观测值。由此极大似然估计函数可以写作如下：

$$L(\delta, \lambda | x_1, x_2, x_n) = \prod_{i=1}^{n} \mathbf{g}(\mathbf{x}_i; \delta, \lambda) = \prod_{i=1}^{n} \frac{\delta^2 [\mathbf{1} - \lambda \mathbf{log}(\mathbf{x}_i)]}{\delta + \lambda} \mathbf{x}_i^{\delta - 1} \tag{3-1}$$

从而进一步可以推出其对数似然函数为：

$$l(\delta, \lambda) = 2n\mathbf{log}(\delta) - n\mathbf{log}(\delta + \lambda) + \sum_{i=1}^{n} \mathbf{log}[1 - \lambda \mathbf{log}(x_i)] + (\delta - 1) \sum_{i=1}^{n} \mathbf{log}(x_i) \tag{3-2}$$

进一步，估计值 $\left(\widehat{\delta}, \widehat{\lambda}\right)$ 即为使得极大对数似然函数3-2的极大值点。具体而言即使得 $l(\delta, \lambda)$ 偏导数为 **0** 的点。

对数极大似然函数的偏导数如下：

$$\frac{\partial l(\delta, \lambda)}{\partial \delta} = \frac{2n}{\delta} - \frac{n}{\delta + \lambda} + \sum_{i=1}^{n} \mathbf{log}(x_i) \tag{3-3}$$

$$\frac{\partial l(\delta, \lambda)}{\partial \lambda} = -\frac{n}{\delta + \lambda} - \sum_{i=1}^{n} \frac{\mathbf{log}(x_i)}{1 - \lambda \mathbf{log}(x_i)} \tag{3-4}$$

上述偏导数方程3-3以及3-4直接计算较为复杂，可以使用数值计算方法来计算得到。具体数值计算的过程和结果我们将在第4章中详细分析。

3.2 置信区间估计

区间估计通常采用置信区间（Confidence Interval, **CI**）的形式，表示参数的估计值落在某一范围内的可信程度。对于该概率函数，我们基于极大似然估计的渐近正态性，来构建渐近置信区间（Asymptotic Confidence Interval）。具体而言，即对于极大似然估计值 $\left(\widehat{\delta},\widehat{\lambda}\right)$，其有渐进正态性假设如下：

$$\widehat{\delta} \sim N(\delta, \mathrm{Var}(\widehat{\delta}))\ ;\ \widehat{\lambda} \sim N(\lambda, \mathrm{Var}(\widehat{\lambda})) \tag{3-5}$$

其中 $\mathrm{Var}(\hat{\delta}), \mathrm{Var}(\hat{\lambda})$ 分别为 $\widehat{\delta},\widehat{\lambda}$ 的渐进方差。因此对于 δ,λ 的一个 $(1-\alpha)\,100\%$ 置信度的置信区间即可写为：

$$\delta' s\ \ (1-\alpha)\,100\%\ CI:\ \left[\widehat{\delta} - Z_{\frac{\alpha}{2}}\sqrt{\mathbf{var}\left(\widehat{\delta}\right)}, \widehat{\delta} + Z_{\frac{\alpha}{2}}\sqrt{\mathbf{var}\left(\widehat{\delta}\right)}\right] \tag{3-6}$$

$$\lambda' s\ \ (1-\alpha)\,100\%\ CI:\ \left[\widehat{\lambda} - Z_{\frac{\alpha}{2}}\sqrt{\mathbf{var}\left(\widehat{\lambda}\right)}, \widehat{\lambda} + Z_{\frac{\alpha}{2}}\sqrt{\mathbf{var}\left(\widehat{\lambda}\right)}\right] \tag{3-7}$$

其中 $Z_{\frac{\alpha}{2}}$ 是标准正态分布的上 $\left(\frac{\alpha}{2}\right)$ 百分位数。

考虑到信息矩阵即为协方差矩阵的逆的性质，因此关于极大似然估计量的渐近方差计算，本文采用 Fisher 信息矩阵的逆来计算。具体过程如下：

首先计算 δ 和 λ 的 Fisher 信息矩阵如下：

$$I_n(\delta,\lambda) = -\begin{pmatrix} I_{\delta\delta} & I_{\delta\lambda} \\ I_{\lambda\delta} & I_{\lambda\lambda} \end{pmatrix}_{(\delta,\lambda)=\left(\hat{\delta},\hat{\lambda}\right)} \tag{3-8}$$

其中矩阵中各元素为：

$$I_{\delta\delta} = \frac{\partial^2 l(\delta,\lambda)}{\partial\delta^2} = -\frac{2n}{\delta^2} + \frac{n}{(\delta+\lambda)^2} \tag{3-9}$$

$$I_{\delta\lambda} = I_{\lambda\delta} = \frac{\partial^2 l(\delta,\lambda)}{\partial\delta\partial\lambda} = \frac{n}{(\delta+\lambda)^2} \tag{3-10}$$

$$I_{\lambda\lambda} = \frac{\partial^2 l(\delta,\lambda)}{\partial\lambda^2} = \frac{n}{(\delta+\lambda)^2} - \sum_{i=1}^{n}\frac{\mathbf{log}(x_i)^2}{[\mathbf{1}-\lambda\mathbf{log}(\mathbf{x}_i)]^2} \tag{3-11}$$

通过上式求得 Fisher 信息矩阵，其逆矩阵的对角元素即为渐近方差，具体如

下式：

$$Diag\left(I_n^{-1}(\delta,\lambda)\right)=\begin{pmatrix}\mathrm{Var}(\hat{\delta}) & \\ & \mathrm{Var}(\hat{\lambda})\end{pmatrix} \tag{3-12}$$

综上求完 $\mathrm{Var}(\hat{\delta})$ 以及 $\mathrm{Var}(\hat{\lambda})$ 后，结合公式3-6以及3-7便得到了参数 δ,λ 的置信区间。具体计算过程和结果将在第4章中展示。

3.3 K-S 拟合优度检验

K-S 拟合优度检验（Kolmogorov-Smirnov Goodness-of-Fit Test）是一种非参数统计检验方法，用于检验样本数据是否来自某个特定的理论分布。该方法通过比较样本的经验累积分布函数（ECDF）与理论分布的累积分布函数（CDF）之间的差异，来评估样本数据与理论分布的拟合程度。

设样本数据为 $\{x_1,x_2,\ldots,x_n\}$，其经验累积分布函数为 $F_n(x)$，即：

$$F_n(x)=\frac{\text{number of observations} \le x}{n}$$

假设理论分布的累积分布函数为 $F(x)$，K-S 检验的原假设为样本数据服从该理论分布，备择假设则是样本数据不服从该理论分布。

K-S 检验的统计量定义为样本经验分布函数与理论分布函数之间的最大绝对差值：

$$D_n=\max_x|F_n(x)-F(x)|$$

根据此统计量，K-S 检验的决定规则为：计算出该统计量 D_n 后，与对应的临界值进行比较。如果 D_n 大于临界值，则拒绝原假设，认为数据不符合该理论分布；如果 D_n 小于等于临界值，则无法拒绝原假设，认为数据与理论分布拟合良好。

3.3.1 K-S 检验的优势

K-S 拟合优度检验有以下几个显著优势：

- **无需假设数据分布形式**：K-S 检验不依赖于数据的具体分布类型，可以用于检验任何已知的理论分布。
- **适用于小样本**：即使样本量较小，K-S 检验仍然有效，且对样本量的要求不高。

第 4 章　模拟研究

本章我们将使用蒙特卡洛模拟结合逆变换抽样法来生成对应概率函数的随机变量，随后基于这些随机变量对第3章中所提出的极大似然估计方法以及置信区间进行模拟计算并做进一步的估计与评价。

蒙特卡洛模拟是一种基于随机采样的计算方法，用于模拟复杂的数学问题。它通过从概率分布中随机选取数据点，来估计该分布的参数。当解析解过于复杂或难以直接获得时，这种技术尤为实用。蒙特卡洛模拟不仅提供了一种高效的近似方法，还为解决实际问题提供了灵活的手段，展现了数学与计算科学的深度融合之美。

4.1　样本生成

为了后续的模拟和统计分析，本节我们将阐明如何产生服从参数 $\{\delta\,\lambda\}$ 的 UExE 模型的样本。

首先介绍逆变换抽样法：

定理 4.1（**逆变换抽样法**）设连续型随机变量 ϵ 的分布函数 $F(x)$ 是连续且严格单调上升的分布函数，其反函数存在且记为 $F^{-1}(x)$, 则有：

(1) 随机变量 $F(\epsilon)$ 服从 $(0,1)$ 上的均匀分布，即 $F(\epsilon) \sim U(0,1)$;

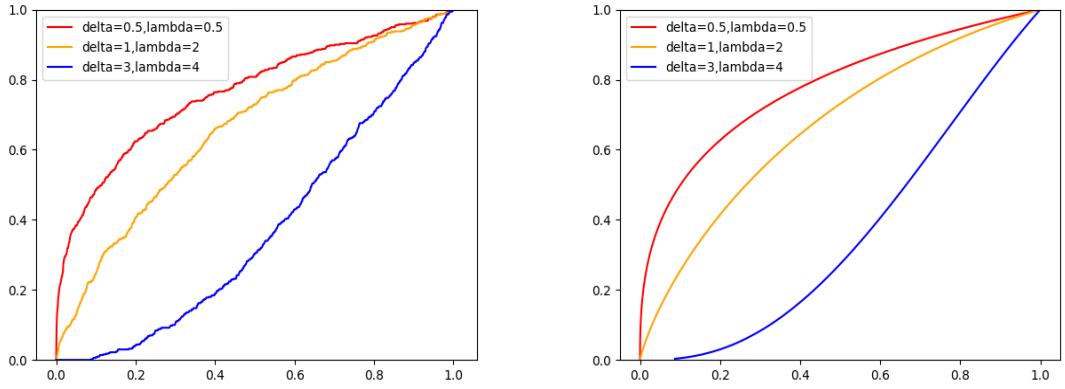
(2) 对于随机变量 $U \sim U(0,1)$, $F^{-1}(U)$ 的分布函数为 $F(x)$。

我们为了利用逆变换抽样法获得满足 UExE 模型的分布函数2-1的随机样本，首先需要求得分布函数2-1的反函数。然而，这个函数的反函数的解析表达式我们无法求得，于是我们使用数值方法来模拟这个分布函数的反函数。具体地，记 $G(x;\delta,\lambda)$ 为 UExE 的分布表达式，我们首先得到 $y_i \in Y \sim U(0,1)$ 作为逆变换抽样法中的满足均匀分布的随机变量，此时我们要求的就是满足 $y_i = G(x;\delta,\lambda)$ 的 $x = x_i$，即此时 $x \in X \sim UExE(\delta,\lambda)$。我们提出用二分法求解使得非线性方程 $y_i = G(x;\delta,\lambda)$ 成立的 $x = x_i$，因为注意到分布函数 $G(x;\delta,\lambda)$ 一定是一个**单调递增函数**，所以这个方法一**定是收敛的**，并且具有**较低的计算复杂度** $O(nlogn)$，具体算法过程见算法1。

Algorithm 1 样本生成算法

Input:
None
Output:
Samples: $x_i \in X \sim UExE(\delta, \lambda)$
Derive $y_i \in Y \sim U(0,1)$
Initialize $low \leftarrow 0$, $high \leftarrow 1$ and threshold: tol
while $high - low > tol$ **do**
  $mid \leftarrow (low + high)/2$
  $y_{mid} \leftarrow \left(1 - \frac{\delta \cdot \lambda \cdot \log(mid)}{\delta + \lambda}\right) \cdot mid^{\delta}$
  if $|y_{mid} - y| < tol$ **then**
    return $x_i = mid$
  end if
  if $y_{mid} < y$ **then**
    $low \leftarrow mid$
  else
    $high \leftarrow mid$
  end if
end while
return $x_i = (low + high)/2.0$

我们参照原文的做法，按照我们提出的上述抽样方法对 CaseI:(δ=0.5, λ=0.5)、CaseII:(δ=1, λ=2) 和 CaseIII:(δ=3, λ=4) 抽取了样本量为 100、150、200、300、400、500 的样本各 1000 份。获得样本后，我们如图4-1 (a)对样本的分布进行可视化，同时与图4-1 (b)的理论分布进行对比，验证了我们抽取的样本的正确性和准确性。



(a) 模拟生成的样本分布图　　(b) 真实分布图

图 4-1　模拟生成的样本分布和真实分布对比

至此，我们完成了用于后续工作的样本生成。

4.2 参数 δ 和 λ 的估计

本节我们将完成对 UExE 分布的参数 δ 和 λ 的估计，主要包括点估计中的极大似然估计并计算偏差、MSE 等指标和进行区间估计并计算平均长度和覆盖率等指标。

4.2.1 点估计：极大似然估计

我们的目标分布 UExE 的对数极大似然函数的偏导数已经由式3-3和3-4给出，我们目标是令二者均为 0，解出由样本作为自变量的参数 δ 和 λ 的表达式，由此得到参数的 δ 和 λ 的极大似然估计值。正如第3章所述，完成上述工作是困难的，所以我们按照原论文的思路，采用牛顿-拉夫逊方法（即牛顿迭代法）对上述两个方程进行数值求解。

具体地，我们首先给出牛顿迭代数值解法：

定理 4.2（牛顿迭代法） 设目标函数 $f(x)$ 是一个实值函数，定义在 $\mathcal{R}^n$ 中，且 $f(x)$ 二阶可微。假设当前迭代点为 x_k，则对下一个迭代点 x_k+1 地更新公式为：

$$x_{k+1} = x_k - H_f(x_k)^{-1}\nabla f(x_k) \tag{4-1}$$

其中，$H_f(x_k)$ 是 $f(x)$ 在 x_k 处的 Hessian 矩阵，$\nabla f(x_k)$ 是 $f(x)$ 在 x_k 处的梯度。

对于我们的问题，δ 和 λ 即我们要用此迭代方法求的两个解。我们设置停止条件，当 $||\nabla f(x_k)|| < \epsilon$ 时，即梯度小于提前设置的阈值 ϵ 时，停止迭代得到最终的 δ 和 λ 值，即为我们极大似然估计的结果。其中需要求 Hessian 矩阵的逆矩阵，当遇到不可逆的矩阵时，我们选择加入正则化项进行处理，即 $H_f(x_k)+\mu I$，具体过程如算法2所示。

Algorithm 2 牛顿迭代求极大似然估计算法

Input:
Samples derived from Algorithm 1.
Output:
MLE values of δ and λ for each input sample.
Set hyperparameters: ϵ and $max\_iteration$.
Initialize $\delta = \delta_0$ and $\lambda = \lambda_0$
while $k < max\_iteration$ **do**
  Derive gradient $\nabla f(\delta_k, \lambda_k)$ by Equation3-3 and 3-4
  if $||\nabla f(\delta_k, \lambda_k)|| < \epsilon$ **then**
    return $\delta = \delta_k$ and $\lambda = \lambda_k$
  end if
  Derive Hessian matrix $H_f(\delta_k, \lambda_k)$
  Derive inverse matrix of the Hessian matrix: $H_f(\delta_k, \lambda_k)^{-1}$
  Update $(\delta_{k+1}, \lambda_{K+1}) = (\delta_k, \lambda_k) - H_f(\delta_k, \lambda_k)^{-1}\nabla f(\delta_k, \lambda_k)$
end while
return $\delta = \delta_K$ and $\lambda = \lambda_K$

我们记统计量 (δ,λ) 为 ω，则样本容量为 n 的一个样本（这里我们除了估计了 n=100,150,200,300,400,500 与原论文一致的量外，还取了更大的 n 值，如 1000,2000）的估计统计量记为 ω_i，我们取了 1000 份这样的样本（即 $i = 1,2...1000$），则有：

$$\hat{\omega} = \frac{1}{1000}\sum_{i}^{1000}\omega_i \tag{4-2}$$

$$Bias(\hat{\omega}) = \sum_{i}^{1000}(\omega_i - \omega) = \hat{\omega} - \omega \tag{4-3}$$

$$MSE(\hat{\omega}) = \sum_{i}^{1000}(\omega_i - \omega)^2 \tag{4-4}$$

估计结果 $\hat{\omega}$、估计偏差 $Bias(\hat{\omega})$ 和估计均方误差 $MSE(\hat{\omega})$ 如下表4.2.1、4.2.1和4.2.1所示，我们可见，用我们的方法得到的估计和原论文中的结果在误差的范围内，是正确的。

表 4-1　Case1($\delta = 0.5,\lambda = 0.5$) 参数估计结果

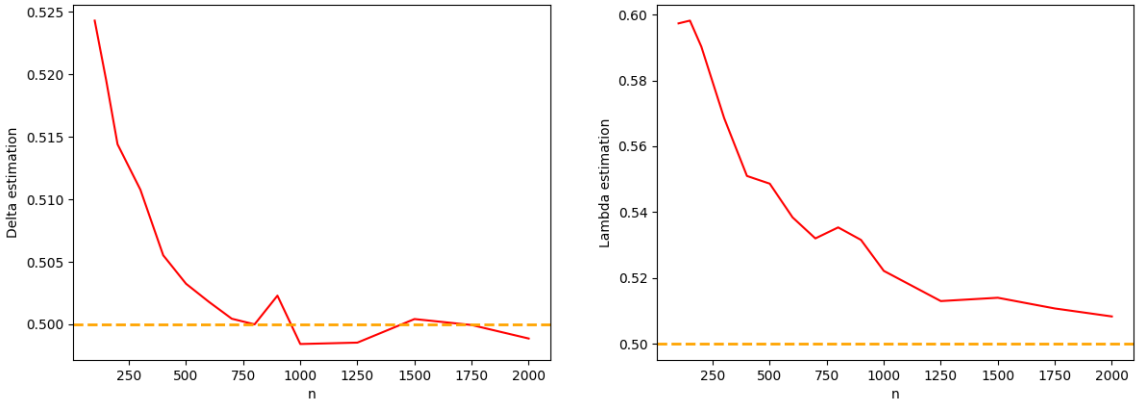
$\delta = 0.5, \lambda = 0.5$	n=100	n=150	n=200	n=300	n=400	n=500
$\hat{\delta}$	0.5240	0.5188	0.5149	0.5107	0.5062	0.5039
$Bias$	0.0240	0.0188	0.0149	0.0107	0.0062	0.0039
MSE	0.0111	0.0077	0.0064	0.0054	0.0043	0.0032
$\hat{\lambda}$	0.5958	0.5992	0.5907	0.5712	0.5522	0.5495
$Bias$	0.0958	0.0992	0.0907	0.0712	0.0522	0.0495
MSE	0.0852	0.0750	0.0728	0.0580	0.0446	0.0395

表 4-2　Case2($\delta = 1.0,\lambda = 2.0$) 参数估计结果

$\delta = 1.0, \lambda = 2.0$	n=100	n=150	n=200	n=300	n=400	n=500
$\hat{\delta}$	1.0617	1.0464	1.0421	1.0278	1.0180	1.0146
$Bias$	0.0617	0.0464	0.0421	0.0278	0.0180	0.0146
MSE	0.0324	0.0252	0.0224	0.0148	0.0141	0.0102
$\hat{\lambda}$	2.5456	2.4029	2.3642	2.2746	2.2220	2.2055
$Bias$	0.5456	0.4029	0.3642	0.2746	0.2220	0.2055
MSE	1.6750	1.3890	1.1755	0.9127	0.7370	0.6435

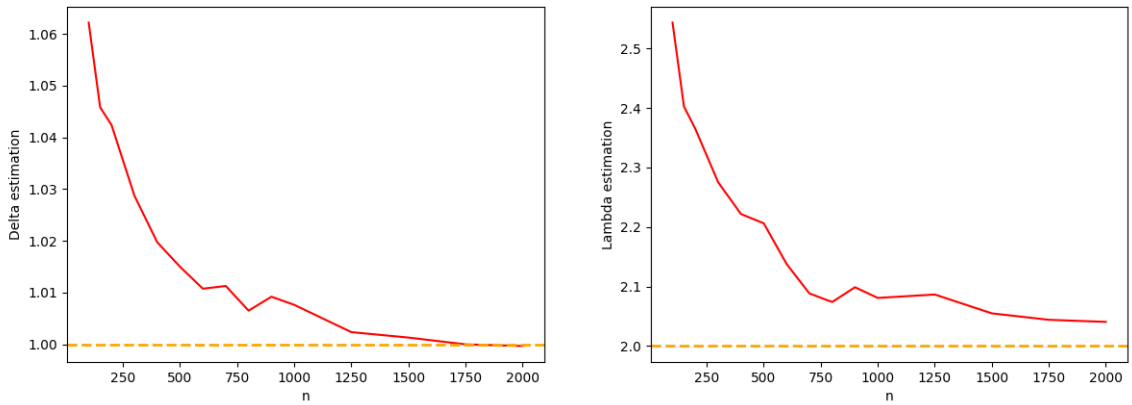
表 4-3　Case3($\delta = 3.0, \lambda = 4.0$) 参数估计结果

$\delta = 3.0, \lambda = 4.0$	n=100	n=150	n=200	n=300	n=400	n=500
$\hat{\delta}$	3.2131	3.1361	3.0932	3.0665	3.0539	3.0456
$Bias$	0.2131	0.1361	0.0932	0.0665	0.0539	0.0456
MSE	0.3857	0.2556	0.2000	0.1735	0.1518	0.1313
$\hat{\lambda}$	4.9039	4.7202	4.5982	4.4382	4.3928	4.3484
$Bias$	0.9039	0.7202	0.5982	0.4382	0.3928	0.3484
MSE	5.8795	4.9286	3.9873	3.2807	2.7493	2.4668



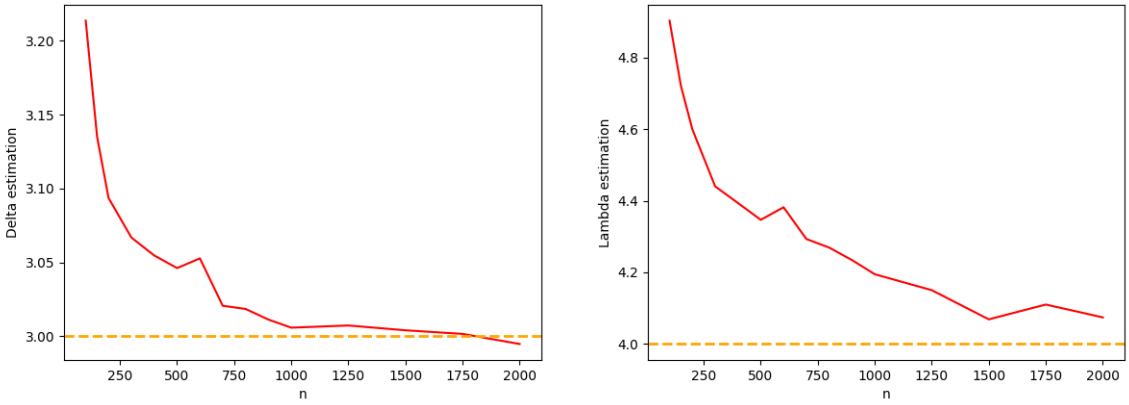
(a) $\delta = 0.5$ 参数估计结果　　(b) $\lambda = 0.5$ 参数估计结果

图 4-2　Case1($\delta = 0.5, \lambda = 0.5$) 参数估计结果



(a) $\delta = 1.0$ 参数估计结果　　(b) $\lambda = 2.0$ 参数估计结果

图 4-3　Case2($\delta = 1.0, \lambda = 2.0$) 参数估计结果



(a) $\delta = 3.0$ 参数估计结果　　(b) $\lambda = 4.0$ 参数估计结果

图 4-4　Case3($\delta = 3.0$,$\lambda = 4.0$) 参数估计结果

4.2.2　区间估计

我们的信息矩阵的元素即目标分布 UExE 的对数极大似然函数的二阶偏导数已经由式 3-9，3-10 和 3-11 和给出，我们的目标是求出信息矩阵的逆，由此得到参数的 δ 和 λ 的渐近方差。由于信息矩阵是一个二阶方阵，我们直接采用伴随矩阵来求解矩阵的逆。

首先，我们给出使用伴随矩阵求解矩阵的逆的方法：

定理 4.3（**矩阵的逆**）设 A 为 n 阶方阵，$\det(A) \neq 0$, 若 $n \geq 2$, 则:

$$A^{-1} = \frac{A^*}{\det(A)} \tag{4-5}$$

求出信息矩阵的逆后，对角线元素即为参数 δ 和 λ 的渐近方差。

Algorithm 3 置信区间计算算法

Input:

Parameters $\hat{\delta}$ and $\hat{\lambda}$ for each sample, sample_sizes and corresponding data sample.

Output:

95% confidence intervals for $\hat{\delta}$ and $\hat{\lambda}$ for each sample.

1: Initialize $\delta = \hat{\delta}$ and $\lambda = \hat{\lambda}$
2: Construct the information matrix I by Eq.3-8:
3: Compute the inverse of I to get variances $\text{var}(\delta)$ and $\text{var}(\lambda)$ by using theory4.3
4: Compute standard errors:

$$\text{SE}_\delta = \sqrt{\text{var}(\delta)}, \quad \text{SE}_\lambda = \sqrt{\text{var}(\lambda)}.$$

5: Compute the critical Z-value for a 95% confidence interval: $Z_{0.975} = 1.96$.
6: Compute confidence intervals:

$$\text{CI}_\delta = (\delta - Z_{0.975} \cdot \text{SE}_\delta, \delta + Z_{0.975} \cdot \text{SE}_\delta),$$

$$\text{CI}_\lambda = (\lambda - Z_{0.975} \cdot \text{SE}_\lambda, \lambda + Z_{0.975} \cdot \text{SE}_\lambda).$$

7: **return** CI_δ , CI_λ.

根据式3-6和3-7，我们求得每组数据的 δ 和 λ 的 $(1-\alpha)100\%$ 置信度的置信区间。算法 3 描述了求得置信区间得具体过程。求出参数的置信区间后我们计算了不同样本长度下的平均区间长度 AL 和区间的覆盖率 CP。具体公式（以 δ 为例）如下：

$$LCI\left(\hat{\delta}\right) = \hat{\delta} - Z_{\phi/2} \cdot \sqrt{\text{var}\left(\hat{\delta}\right)} \tag{4-6}$$

$$UCI\left(\hat{\delta}\right) = \hat{\delta} + Z_{\phi/2} \cdot \sqrt{\text{var}\left(\hat{\delta}\right)} \tag{4-7}$$

$$AL\left(\hat{\delta}\right) = UCI\left(\hat{\delta}\right) - LCI\left(\hat{\delta}\right) \tag{4-8}$$

$$CP\left(\hat{\delta}\right) = I_{true\,\delta}/1000 \tag{4-9}$$

其中 $I_{true\,\delta}$ 等于 1 如果 δ 的真实值在区间里，否则为 0。

置信区间的覆盖率和平均长度如下表所示，我们可以得出结论，随着样本量的增加，95% 置信区间变得更加紧凑。置信区间的覆盖率仍接近预期的 95% 水平，这表明即使在更大的范围内，MLE 也是准确的。此外，除了原论文中的提到的样本容量的结果，我们还对 n=600,700,800....2000 的样本容量 95% 置信区间的平均长度的进行了估计，如图4-5、4-6和**??**所示。

表 4-4　Case1($\delta = 0.5$) 区间估计结果

$\delta = 0.5$	n=100	n=150	n=200	n=300	n=400	n=500
$CP_{95\%}$	0.893	0.905	0.906	0.910	0.929	0.921
$AL_{95\%}$	0.29043	0.28978	0.20313	0.17533	0.15055	0.13509

表 4-5　Case1($\lambda = 0.5$) 区间估计结果

$\lambda = 0.5$	n=100	n=150	n=200	n=300	n=400	n=500
$CP_{95\%}$	0.944	0.947	0.927	0.941	0.942	0.952
$AL_{95\%}$	1.92271	1.90152	1.34598	1.13333	0.96426	0.86674

表 4-6　Case2($\delta = 1.0$) 区间估计结果

$\delta = 1.0$	n=100	n=150	n=200	n=300	n=400	n=500
$CP_{95\%}$	0.946	0.931	0.922	0.903	0.909	0.918
$AL_{95\%}$	0.46074	0.38350	0.33396	0.27152	0.23392	0.20897

表 4-7　Case2($\lambda = 2.0$) 区间估计结果

$\lambda = 2.0$	n=100	n=150	n=200	n=300	n=400	n=500
$CP_{95\%}$	0.9	0.929	0.933	0.923	0.93	0.949
$AL_{95\%}$	7.46980	6.33110	5.477932	4.21908	3.53470	3.12073

表 4-8　Case3($\delta = 3.0$) 区间估计结果

$\delta = 3.0$	n=100	n=150	n=200	n=300	n=400	n=500
$CP_{95\%}$	0.941	0.926	0.927	0.925	0.94	0.939
$AL_{95\%}$	1.64564	1.37259	1.187062	0.99931	0.84387	0.79329

表 4-9　Case3($\lambda = 4.0$) 区间估计结果

$\lambda = 4.0$	n=100	n=150	n=200	n=300	n=400	n=500
$CP_{95\%}$	0.950	0.930	0.939	0.938	0.951	0.967
$AL_{95\%}$	14.1734	11.4154	9.9990	8.4954	7.2459	6.8499

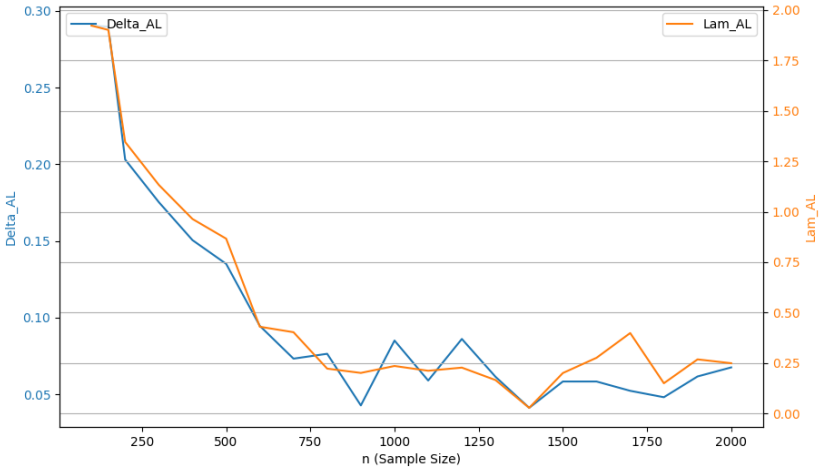
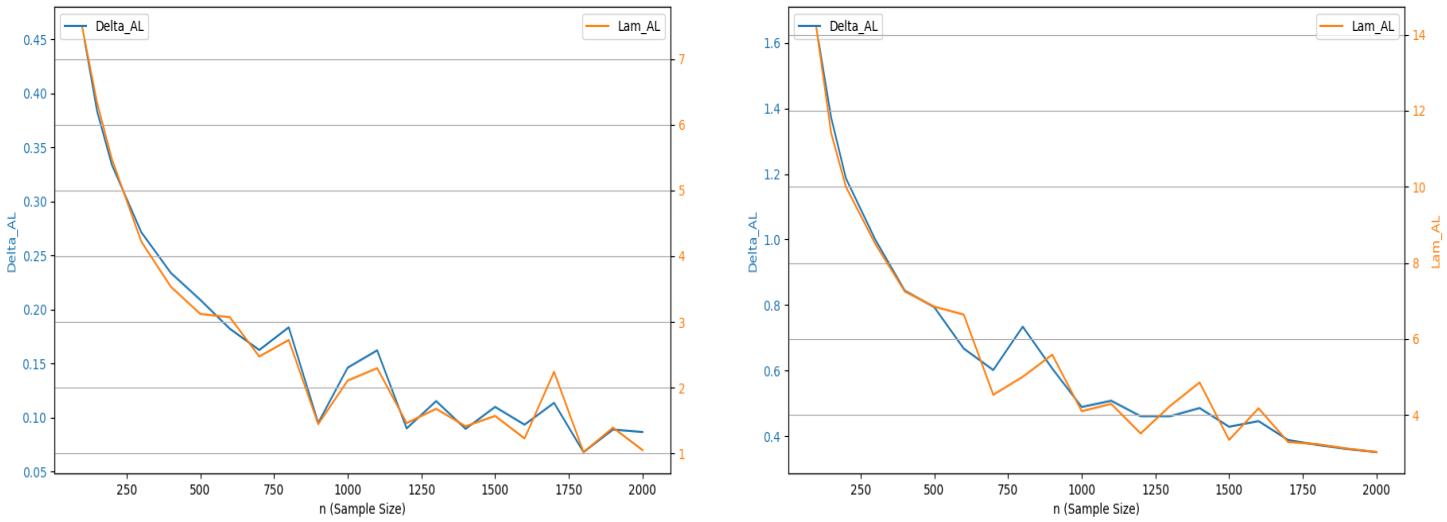


图 4-5　Case1($\delta = 0.5$,$\lambda = 0.5$)95% 置信区间平均长度估计结果

通过图4-5、4-6我们可以看出，无论是 δ 还是 λ，其平均长度都会随着样本量的增加而减少，表明样本量的增加对估计结果的精确性具有显著提升作用。不同参数的收敛速度存在差异，其中 λ 的收敛速度普遍慢于 δ，尤其在小样本时表现更为显著。δ 的置信区间平均长度在各个案例中相对较小，表明其估计结果更加稳定。λ 的置信区间平均长度较大，特别是在小样本情况下，显示出更高的不确定性，但随着样本量的增加，其收敛趋势仍较为明显。



(a) (b)

图 4-6 Case2(δ = 1.0,λ = 2.0)95% 和 Case3(δ = 3.0,λ = 4.0)95% 置信区间平均长度估计结果

4.3 总结

本章通过蒙特卡洛模拟结合逆变换抽样法，系统地生成了 UExE 模型的随机变量样本，并在此基础上使用极大似然估计验证了参数估计方法的统计有效性和理论可靠性并且估计了参数的置信区间及其平均长度和覆盖率。

针对模型参数 δ 和 λ 的点估计结果表明，随着样本量的增加，参数估计的偏差（Bias）和均方误差（MSE）均显著下降，估计值逐步趋近于真实值。这一结果验证了极大似然估计方法的渐近一致性特性，说明该方法在大样本情形下具有良好的估计精度和稳健性。

在置信区间的构建方面，基于参数的渐近正态性，构造了参数 δ 和 λ 的 95% 置信区间。研究结果显示，随着样本量的增加，置信区间的平均长度（AL）逐渐缩短，区间覆盖率（CP）逐渐接近理论值 95%，进一步验证了所构建置信区间的可靠性和方法。同时不难发现 δ 的收敛速度相较于 λ 略慢。此外，样本量的增加显著提升了参数估计的精度，这表明在实际应用中应根据样本量的需求合理选择统计方法。

综上所述，本章的模拟研究全面验证了 UExE 模型参数估计方法的有效性和稳健性，充分展现了该模型和方法的统计学优势，为进一步的实际数据分析提供了坚实的理论基础。

第 5 章　模型应用

5.1　模型基本应用

本节按照原论文的思路，通过分析两个基本的实际数据集，将 **UExE** 模型与 Beta 分布模型、Kumaraswamy 分布模型等 9 种竞争模型进行了比较。**UExE** 模型为拟合单位区间数据提供了一种优异的分布选择，其在对数似然方程（Log_L）、Akaike 信息准则（AIC）、Kolmogorov-Smirnov（K-S）检验等指标上的表现均优于传统模型。实验结果表明，**UExE** 模型能够为数据提供更好的拟合效果，展示了 **UExE** 模型在实际场景中的应用表现，并且通过图像分析进一步验证了这一结论。

5.1.1　数据集

第一个数据集包括 30 次聚酯纤维的拉伸强度观察[2]：{0.023, 0.032, 0.054, 0.069, 0.081, 0.094, 0.105, 0.127, 0.148, 0.169, 0.188, 0.216, 0.255, 0.277, 0.311, 0.361, 0.376, 0.395, 0.432, 0.463, 0.481, 0.519, 0.529, 0.567, 0.642, 0.674, 0.752, 0.823, 0.887, 0.926}。

第二个数据集包含 P3 算法的 22 条计算时间记录[3]：{0.853, 0.759, 0.874, 0.800, 0.716, 0.557, 0.503, 0.399, 0.334, 0.207, 0.118, 0.097, 0.078, 0.067, 0.056, 0.044, 0.036, 0.026, 0.019, 0.014, 0.010, 0.118}。

5.1.2　竞争模型

首先再次回顾 UExE 模型的概率密度函数:

$$g(x;\delta,\lambda)=\frac{\delta^2(1-\lambda\log(x))x^{\delta-1}}{\delta+\lambda},\quad 0<x<1,\ \delta,\lambda>0.$$

UExE 模型为分析单位区间数据提供了一种有效的工具。本节将介绍 UExE 模型相较于 Beta 分布、Kumaraswamy 分布等传统模型的优势。原论文选取了九种分布模型与 UExE 模型进行比较：

- Beta 分布[4]

$$g(x)=\frac{(1-x)^{\kappa-1}x^{\gamma-1}}{\mathrm{Beta}(\kappa,\gamma)};\quad 0<x<1,\ \kappa,\gamma>0$$

- Kumaraswamy 分布[5]

$$g(x) = abx^{a-1}(1-x^a)^{b-1}; \quad 0 < x < 1,\ a, b > 0$$

- 单位 Gamma 分布[6]

$$g(x) = \frac{\theta x^{\theta-1}(-\theta\log(x))^{\rho-1}}{\Gamma(\rho)}; \quad 0 < x < 1,\ \theta, \rho > 0$$

- Power Topp-Leone 分布[7]

$$g(x) = 2\tau\nu x^{\tau\nu-1}(1-x^\tau)(2-x^\tau)^{\nu-1}; \quad 0 < x < 1,\ \nu, \tau > 0$$

- 有界加权指数分布[8]

$$g(x) = \frac{(\sigma+1)\eta x^{\eta-1}(1-x^{\sigma\eta})}{\sigma}; \quad 0 < x < 1,\ \sigma, \eta > 0$$

- 单位 Burr XII 分布[9]

$$g(x) = \frac{\mu\phi\left[1+(-\log(x))^\phi\right]^{-1-\mu}}{[-\log(x)]^{-\phi+1}x}; \quad 0 < x < 1,\ \mu, \phi > 0$$

- 单位 Birnbaum-Saunders 分布[10]

$$g(x) = \frac{\left(\sqrt{\frac{-\beta}{\log(x)}} + \left(\frac{-\beta}{\log(x)}\right)^{3/2}\right)\exp\left(\frac{\frac{\beta}{\log(x)} + \frac{\log(x)}{\beta} + 2}{2\alpha^2}\right)}{2\sqrt{2\pi}\alpha\beta x}; \quad 0 < x < 1,\ \alpha, \beta > 0$$

- 单位逆高斯分布[11]

$$g(x) = \frac{\zeta\exp\left[\frac{\zeta(\log(x)+\omega)^2}{2\omega^2\log(x)}\right]}{\sqrt{2\pi}x\log^2(x)\sqrt{\frac{-\zeta}{\log(x)}}}; \quad 0 < x < 1,\ \zeta, \omega > 0$$

- 单位 Mirra 分布[12]

$$g(x) = \frac{\psi^3(\Theta + (\Theta+2)x^2 - 2\Theta x)e^{\psi - \frac{\psi}{x}}}{2x^4(\Theta+\psi^2)}; \quad 0 < x < 1,\ \psi, \Theta > 0$$

5.1.3 评估指标

通过以下指标对模型拟合效果进行评估:

- 对数似然函数（Log_L ）

模型参数在给定数据下的对数似然值的总和，反映模型解释数据的能力。该值越大，说明模型在当前参数下对数据的拟合程度越高。

- Akaike 信息准则（AIC）

用于评估模型复杂性和拟合优良度的平衡。计算公式为:

$$\text{AIC} = -2\text{Log\_L} + 2k \qquad \text{(k 是模型中估计参数的个数)}$$

AIC 值越小，说明模型在平衡复杂性和拟合数据方面表现更优。

- 一致 AIC（CAIC）

AIC 的改进版本，在惩罚项中引入了样本大小 n，更加倾向于惩罚复杂模型。计算公式为：

$$\text{CAIC} = \text{AIC} + 2k(k+1)/(n-k-1) \qquad \text{(k 是模型中估计参数的个数)}$$

CAIC 值越小，模型的拟合优良性和简单性平衡越好。

- Hannon-Quinn 信息准则（HQIC）

介于 AIC 和 CAIC 之间的一种准则，对于模型复杂性较高的情况比 AIC 更严格。计算公式为：

$$\text{HQIC} = -2\text{Log\_L} + 2k\log(\log(n)) \qquad \text{(k 是模型中估计参数的个数)}$$

HQIC 值越小，模型的拟合表现更优。

- Anderson-Darling（A*）

基于样本数据的累积分布函数与理论分布函数之间差异的统计量。计算公式为：

$$A^2 = -n - \frac{1}{n}\sum_{i=1}^{n}\left[(2i-1)(\ln(F(X_i)) + \ln(1-F(X_i)))\right]$$

A* 值越小，说明样本分布与理论分布的拟合越好。

- Cramer-Von Mises（W*）

用于度量经验分布函数和理论分布函数的总体差异。计算公式为：

$$W^2 = n\int_{-\infty}^{\infty}\left[F_n(x) - F(x)\right]^2 dF(x)$$

W* 值越小，说明样本分布与理论分布的整体拟合程度更高。

- Kolmogorov-Smirnov（K-S）检验及其 p 值

K-S 统计量通过计算样本分布函数和理论分布函数的最大差值，来检验二者是否一致。p 值表示检验的显著性。K-S 统计量越小，说明样本分布更接近理论分布。若 p 值大于显著性水平（如 0.05），说明无法拒绝样本分布与理论分布一致的假设，即拟合效果较好。

5.1.4 数据集 1 的分析

5.1.4.1 点估计

表5-1展示了不同模型下参数的 MLE 和标准差。不难看出，使用 UExE 模型

时，估计 λ 时求出的标准差很小，说明 λ 的估计较为精确。随后，利用表5-1的数据，得到评估指标的值，如表5-2、表5-3所示。由图表可知，使用 UExE 模型时，Log_L 的值最大，AIC、CAIC、HQIC 的值最小，反映出 UExE 更强的拟合效果。再分析表5-1中的数据，可以发现 UExE 模型下的 **A*** 值和 **W*** 值最小，意味着拟合效果最佳。此外，UExE 的 K-S 统计量也很小，p 值却很大，说明从理论上说，同其他九个模型相比，UExE 模型的估计效果是最好的。

表 5-1　第一个数据集下的 MLE 和标准差（括号内）

Model	**MLE estimates**	
UExE (λ, δ)	$\hat{\lambda}$ = 1.310338 (0.248247)	$\hat{\delta}$ = 5.452465 (5.919716)
Beta (κ, γ)	$\hat{\kappa}$ = 1.620463 (0.434655)	$\hat{\gamma}$ = 0.966648 (0.223703)
Kum (a, b)	$\hat{a}$ = 1.608369 (0.481730)	$\hat{b}$ = 0.962711 (0.237157)
UG (θ, ρ)	$\hat{\theta}$ = 1.588786 (0.366692)	$\hat{\rho}$ = 1.152583 (0.316150)
PTL (ν, τ)	$\hat{\nu}$ = 1.895015 (2.380473)	$\hat{\tau}$ = 0.489427 (0.758221)
BWE (σ, η)	$\hat{\sigma}$ = 2.224477 (3.996305)	$\hat{\eta}$ = 0.950432 (0.624523)
UBXII (μ, ϕ)	$\hat{\mu}$ = 1.033098 (0.206176)	$\hat{\phi}$ = 1.846464 (0.395060)
UBS (α, β)	$\hat{\alpha}$ = 1.077204 (0.142473)	$\hat{\beta}$ = 0.848295 (0.198977)
UIG (ζ, ω)	$\hat{\zeta}$ = −0.922059 (0.238218)	$\hat{\omega}$ = 1.378456 (0.311083)
UM (ψ, Θ)	$\hat{\psi}$ = 0.005133 (1.000000)	$\hat{\Theta}$ = 0.283976 (1.000000)

表 5-2　第一个数据集的 Log_L、AIC、CAIC、HQIC

Model	**Log_L**	**AIC**	**CAIC**	**HQIC**
UExE (λ, δ)	3.469139	-2.938279	-2.493834	-2.041768
Beta (κ, γ)	3.305064	-2.610127	-2.165683	-1.713617
Kum (a, b)	3.311035	-2.622069	-2.177625	-1.725559
UG (θ, ρ)	3.424438	-2.848876	-2.404432	-1.952366
PTL (ν, τ)	3.048852	-2.097703	-1.653259	-1.201193
BWE (σ, η)	2.957385	-1.914769	-1.470325	-1.018259
UBXII (μ, ϕ)	1.038896	1.922029	2.366473	2.818539
UBS (α, β)	0.505026	2.989948	3.434393	3.886459
UIG (ζ, ω)	-1.278312	6.556623	7.001068	7.453133
UM (ψ, Θ)	-0.065821	4.131642	4.576086	5.028152

表 5-3　第一个数据集的 A*, W*, K-S 和 K-S 检验的 p 值

Model	A*	W*	K-S	p
UExE (λ, δ)	0.119773	0.020139	0.035563	1
Beta (κ, γ)	0.170288	0.025173	0.045079	0.999999
Kum (a, b)	0.163295	0.024498	0.043714	0.999999
UG (θ, ρ)	0.153004	0.022474	0.041211	1
PTL (ν, τ)	0.223818	0.026462	0.049972	0.999999
BWE (σ, η)	0.250319	0.025764	0.051457	0.999998
UBXII (μ, ϕ)	0.503722	0.051779	0.081076	0.989169
UBS (α, β)	1.111359	0.086296	0.153286	0.481285
UIG (ζ, ω)	1.472928	0.098022	0.166934	0.373246
UM (ψ, Θ)	2.312309	0.115427	0.212323	0.133718

5.1.4.2　方差-协方差矩阵与置信区间

UExE 模型在第一个数据集上的极大似然估计（MLEs）的方差-协方差矩阵为:

$$\begin{bmatrix} 0.06162658 & 0.91386527 \\ 0.91386527 & 35.04303335 \end{bmatrix}.$$

参数 λ 和 δ 的 95% 置信区间为:

$$\lambda \in [1.79690205, 17.0551073], \quad \delta \in [0.82377375, -6.15017804].$$

从该方差-协方差矩阵, 我们可以进一步比较 UExE 的两个参数的估计效果。分析主对角线元素，λ 的估计值精确度要优于 δ，相关系数 ρ 大概取值为 0.62。

5.1.4.3　可视化

除了计算参数的 MLE、各种评估指标、方差-协方差矩阵和置信区间，图5-1提供了基于数据集 1 的各个模型拟合的概率密度函数以及概率分布函数的可视化结果。由图5-1可知，UExE 模型在数据集 1 的拟合度和其他九种模型相比更能够反应数据的真实变化。同时对比原论文，概率密度函数与概率分布函数与原论文基本一致，可见前文提出的估计方法较为准确与高效。

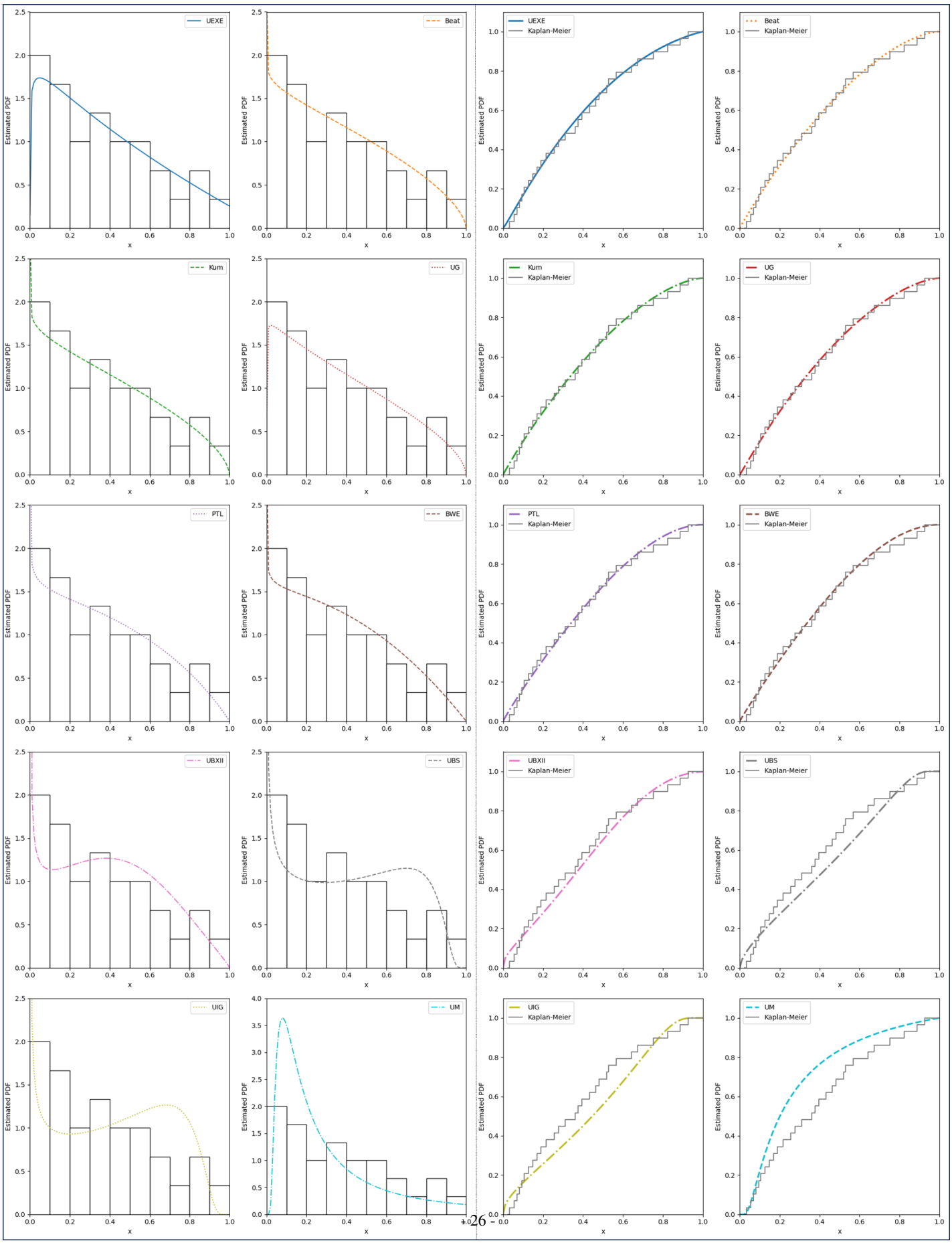


图 5-1　数据集一与拟合模型的概率密度函数与概率分布函数

5.1.5 数据集 2 的分析

5.1.5.1 点估计

对于数据集 2，依旧采用分析数据集 1 时评价模型拟合效果好坏的方法。表5-4展示了不同模型下参数的 MLE 和标准差。使用 UExE 模型时，估计 λ 时求出的标准差依旧很小，说明 λ 的估计依旧较为精确。随后，利用表5-4的数据，得到评估指标的值，如表5-5、表5-6所示。

表 5-4 第二个数据集的 MLE 估计和标准差（括号内）

Model	MLE estimates	
UExE (λ, δ)	$\hat{\lambda}$ = 0.779734 (0.199214)	$\hat{\delta}$ = 0.944176 (1.340949)
Beta (κ, γ)	$\hat{\kappa}$ = 1.219779 (0.375774)	$\hat{\gamma}$ = 0.553954 (0.142315)
Kum (a, b)	$\hat{a}$ = 1.230547 (0.352244)	$\hat{b}$ = 0.571800 (0.150000)
UG (θ, ρ)	$\hat{\theta}$ = 1.249895 (0.340925)	$\hat{\rho}$ = 0.629702 (0.215507)
PTL (ν, τ)	$\hat{\nu}$ = 11.580963 (17.036527)	$\hat{\tau}$ = 0.044831 (0.069197)
BWE (σ, η)	$\hat{\sigma}$ = 30.402870 (11.905059)	$\hat{\eta}$ = 0.519848 (0.235491)
UBXII (μ, ϕ)	$\hat{\mu}$ = 0.837525 (0.205573)	$\hat{\phi}$ = 1.521768 (0.297165)
UBS (α, β)	$\hat{\alpha}$ = 1.241258 (0.189752)	$\hat{\beta}$ = 1.079841 (0.235725)
UIG (ζ, ω)	$\hat{\zeta}$ = −0.932061 (0.279264)	$\hat{\omega}$ = 1.984899 (0.612988)
UM (ψ, Θ)	$\hat{\psi}$ = 0.001042 (0.045199)	$\hat{\Theta}$ = 0.075105 (0.133228)

表 5-5 第二个数据集的 Log_L、AIC、CAIC、HQIC

Model	Log_L	AIC	CAIC	HQIC
UExE (λ, δ)	7.148433	-10.296866	-9.665288	-9.782833
Beta (κ, γ)	6.781925	-9.563850	-8.932271	-9.049816
Kum (a, b)	6.843617	-9.687234	-9.055655	-9.173201
UG (θ, ρ)	6.905199	-9.810399	-9.178820	-9.296365
PTL (ν, τ)	6.959497	-9.918994	-9.287415	-9.404960
BWE (σ, η)	7.027168	-10.054336	-9.422757	-9.540302
UBXII (μ, ϕ)	3.433134	-2.866268	-2.234689	-2.352234
UBS (α, β)	5.765115	-7.530231	-6.898652	-7.016197
UIG (ζ, ω)	3.924543	-3.849085	-3.217506	-3.335052
UM (ψ, Θ)	3.250433	-2.500866	-1.869287	-1.986833

表 5-6　第二个数据集的 A*, W*, K-S and K-S 检验的 p 值

Model	A*	W*	K-S	HQIC
UExE (λ, δ)	0.625465	0.066750	0.162441	0.607145
Beta (κ, γ)	0.724122	0.073536	0.178529	0.484699
Kum (a, b)	0.717327	0.072175	0.174624	0.513471
UG (θ, ρ)	0.728349	0.071917	0.173167	0.524388
PTL (ν, τ)	0.722908	0.075571	0.183693	0.447874
BWE (σ, η)	0.726861	0.075590	0.183734	0.447590
UBXII (μ, ϕ)	1.318834	0.107738	0.239988	0.158580
UBS (α, β)	1.314328	0.107841	0.236304	0.171287
UIG (ζ, ω)	1.800986	0.124780	0.263812	0.093554
UM (ψ, Θ)	4.117071	0.135983	0.263478	0.094281

由图表可知，使用 UEXE 模型时，Log_L 的值最大，AIC、CAIC、HQIC 的值最小，反映出 UExE 在数据集 2 下也具有更强的拟合效果。再分析表5-4中的数据，可以发现 UExE 模型下的 **A*** 值和 **W*** 值最小，即拟合效果理应最佳。此外，UExE 在第二个数据集下的 K-S 统计量也很小，而 p 值却很大。因此即便是在数据量更少的数据集 2 下，UExE 模型的估计效果依旧是非常好的。

5.1.5.2　方差-协方差矩阵与置信区间

UExE 模型在数据集 2 下的极大似然估计（MLEs）的方差-协方差矩阵为

$$\begin{bmatrix} 0.03968613 & 0.20741785 \\ 0.20741785 & 1.79814382 \end{bmatrix}.$$

参数 λ 和 δ 的 95% 置信区间为：

$$\lambda \in [1.17019341, 3.57243589], \quad \delta \in [0.38927535, -1.6840836].$$

使用数据集 2 时，λ 的估计值精确度同样优于 δ。因此，两个数据集均是 δ 的估计值更精确，相关系数 ρ 取值大概为 0.78。

5.1.5.3　可视化

图5-2展示了基于数据集 2 的各个模型拟合的概率密度函数以及概率分布函数的可视化结果。由图5-2可知，UExE 模型在数据集 2 上的拟合度相比于其他九种模型，更能够反应数据的真实变化。同时对比原论文，概率密度函数与概率分布函数与原论文基本一致，可见前文提出的估计方法较为准确与高效。

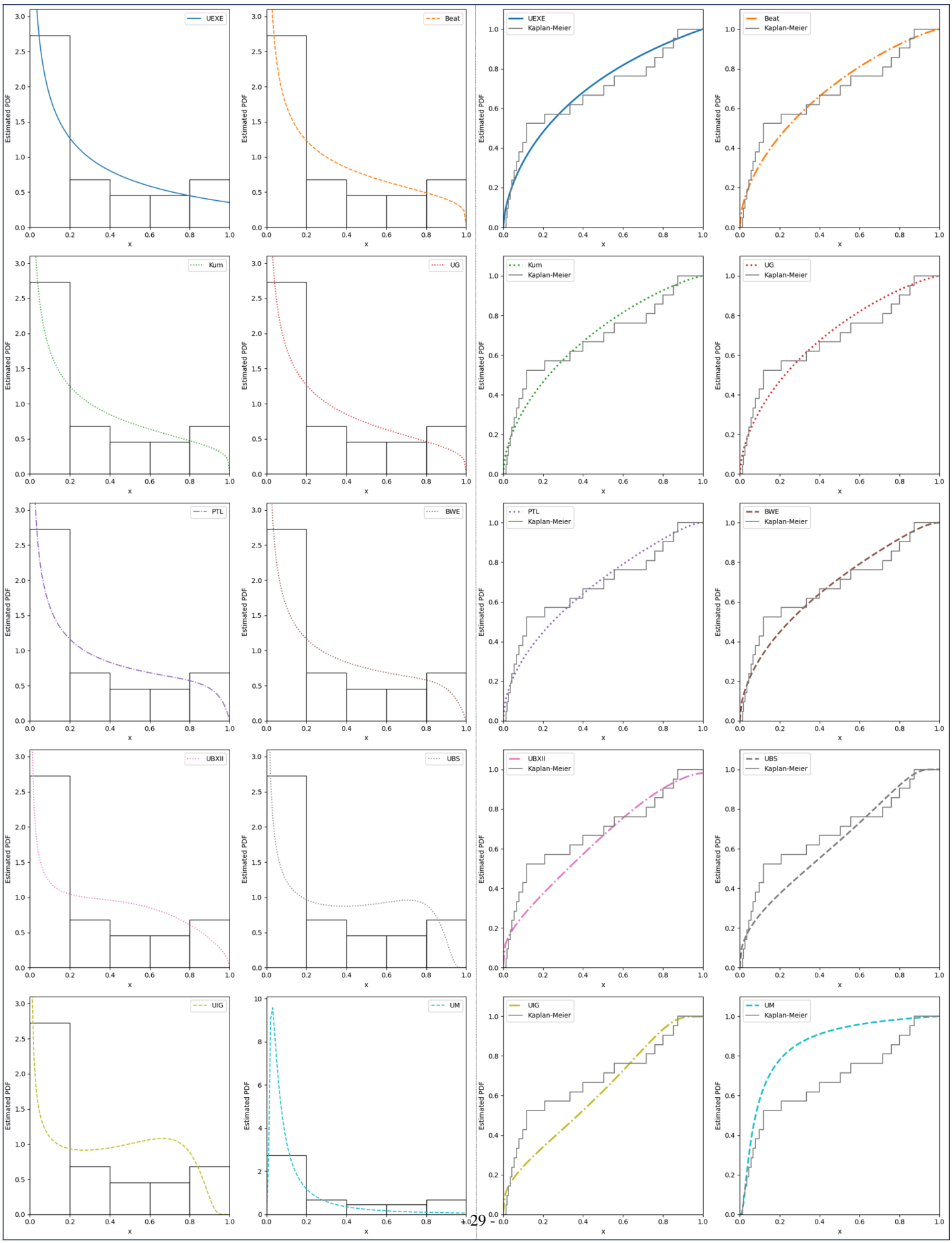


图 5-2　数据集二与拟合模型的概率密度函数与概率分布函数

5.1.6 用 Bootstrap 方法估计参数估计效果

由于原论文中提供的两组真实数据的样本量较小，因此我们进一步通过 **Bootstrap** 重抽样方法计算对两个样本的极大似然估计的参数估计量的偏差与方差。具体结果见表5-7。

表 5-7 用 Bootstrap 方法对两数据集参数估计量计算偏差和方差

Model	Parameters	Bias(data1)	Var(data1)	Bias(data2)	Var(data2)
UExE	λ	-1.46834E-07	3.93898E-13	-7.20281E-08	2.39841E-15
	δ	-0.000004	8.85095E-10	-5.90377E-07	1.60066E-13
Beta	κ	-3.61754E-08	1.64679E-14	1.14725E-07	6.88796E-15
	γ	-1.16305E-08	4.19494E-15	4.61550E-08	5.65239E-16
Kum	a	-2.07969E-08	4.64956E-16	4.76384E-09	4.17140E-16
	b	-9.64206E-09	8.55726E-17	1.03251E-09	4.77524E-17
UG	θ	7.54413E-08	3.74441E-15	-4.357829E-09	1.76732E-15
	ρ	5.28274E-08	2.25134E-15	-3.25715E-09	4.93891E-16
PTL	ν,	1.24499E-06	5.90130E-12	7.27E-5	2.40815E-08
	τ	-3.91231E-07	5.90660E-13	-2.93436E-07	4.17429E-13
BWE	σ	-7.84231E-06	2.06868E-10	-6.5779E-4	9.90219E-07
	η	5.75564E-07	1.05639E-12	-6.5779E-4	1.15449E-12
UBXII	μ	-9.083399E-09	9.83887E-17	-1.49109E-08	2.13993E-16
	ϕ	1.18058E-09	4.32326E-16	1.33381E-08	4.09009E-16
UBS	α	-7.97684E-09	2.24423E-17	-1.292038E-08	1.33272E-16
	β	5.77174E-09	1.93763E-17	-1.96131E-08	3.20589E-16
UIG	ζ	6.41040E-09	5.90380E-17	2.041587E-08	1.063544E-16
	ω	1.81185E-08	2.24511E-16	2.94752E-08	3.02292E-15
UM	ψ	-1.07314E-10	1.76438E-19	-4.04571E-11	2.84285E-21
	Θ	-1.07314E-10	1.21956E-17	-6.93795E-10	7.90342E-19

Bootstrap 重抽样具体流程如下（以 UExE 模型估计为例）：

(I) 从原始小样本 $x_1, x_2, , x_n$ 中通过极大似然估计估计参数 $\hat{\delta}, \hat{\lambda}$

(II) 从原始小样本 $x_1, x_2, , x_n$ 中又放回的重抽样 n 次得到一组新样本

$x_1^*, x_2^*, , x_n^*$ ，并基于此样本计算参数估计量 $\hat{\delta}_1^*, \hat{\lambda}_1^*$

(III) 重复步骤 (II) $M = 1000$ 次，得到估计量 $\hat{\delta}_1^*, \hat{\delta}_2^*, , \hat{\delta}_M^*$ 以及 $\hat{\lambda}_1^*, \hat{\lambda}_2^*, , \hat{\lambda}_M^*$ 并计算估计量均值：$\overline{\hat{\delta}^*} = \frac{1}{M}\sum_{i=1}^{M}\hat{\delta}_i^*$ 以及 $\overline{\hat{\lambda}^*} = \frac{1}{M}\sum_{i=1}^{M}\hat{\lambda}_i^*$。

(IV) 计算 **Bootstrap** 重抽样偏差 $\widehat{Bias_B}(\hat{\delta}) = \frac{1}{M}\sum_{i=1}^{M}(\hat{\delta}_i^* - \hat{\delta})$ 以及 $\widehat{Bias_B}(\hat{\lambda}) = \frac{1}{M}\sum_{i=1}^{M}(\hat{\lambda}_i^* - \hat{\lambda})$

(V) 计算 **Bootstrap** 重抽样方差 $\widehat{Var_B}(\hat{\delta}) = \frac{1}{M}\sum_{i=1}^{M}\left(\hat{\delta}_i^* - \overline{\hat{\delta}^*}\right)^2$ 以及 $\widehat{Var_B}(\hat{\lambda}) = \frac{1}{M}\sum_{i=1}^{M}\left(\hat{\lambda}_i^* - \overline{\hat{\lambda}^*}\right)^2$

5.1.7 局限性

本章的前 6 张表展示了不同模型下不同评估指标的取值。尽管 UExE 模型在绝大多数评估指标数值上的反映占优，UExE 模型仍有其局限性。其一，对于 UExE 模型的的两个参数 λ 和 δ，λ 的估计准确性的确相对较高，然而该模型的另一个参数 δ 的估计准确性和其他九个模型相比占有很大的劣势。拿 Power Topp-Leone 分布模型举例，就单一参数评估准确性而言，UExE 模型远不如 Power Topp-Leone 分布模型。其二，分析在使用 Bootstrap 方法估计参数估计效果时，无论是数据集 1 还是数据集 2，UExE 模型的两个参数估计量的偏差和方差均不是最小的，即 UExE 两参数估计量在偏差与方差上并不是特别的出色。因此，虽然 UExE 模型在之前数据拟合方面表现较为优异，通过极大似然估计对两参数的进行估计的方法可能存在一定的局限性。但考虑偏差与方差的数量级较小，在实际估计过程中，极大似然估计方法总体上还是可以接受的。

5.2 模型现实应用

我们所复现的论文提出的模型 UExE 相比其他竞争的模型，能够为数据提供更好的拟合效果。经过前文对原论文提出的基本数据进行复现应用后，我们得到的效果确实如此。同时，模型 UExE 是基于 UExE 模型结合有限范围数据的特性而建立的，这使得此模型对在一定范围内的生命周期数据的分析具有灵活性且在拟合数据是时相比其他的模型有更高的准确性。UExE 模型具有的以上性质使得其在现实世界生产和预测中可以得到广泛的应用。所以在本小节中，我们尝试利用我们寻找的真实的企业的生产数据将此模型扩展应用到现实生产的估计和预测中。

5.2.1 生产数据集介绍

我们选取了 2024 年第七届泰迪杯数据分析技能赛 A 题-**自动化生产线数据分析的数据**①。该数据记录了某生产企业两条生产线全年加工处理过程中各个工序的生产线数据，其中包括了各个工序的状态和生产数量，具体的数据形式和数据见附录2和附件。我们可以根据原始数据中的每天各个工序的状态得到每天生产故障时长和工作时长，以此作为生产的生命周期的数据，应用 UExE 模型来拟合并对企业的生产做预测。

5.2.2 利用 UExE 模型拟合生产线每日正常工作时长

对于生产线每日正常工作时长数据，我们沿用前文使用的极大似然估计法即算法2对 UExE 模型的两个参数 δ、λ 进行估计。具体地，我们分别对两条生产线划分一年 365 天中的 250 天的生产线状态数据存活时间作为 UExE 的拟合数据，剩余的数据进行验证模型拟合的优劣。我们对两条生产线建立了拟合好的模型对企业后续的生产决策起到关键作用。

5.2.3 模拟结果和应用

我们对参数 δ、λ 进行估计后的结果如表5-8所示。结果表明在此样本量较大的数据集中对两个生产线的拟合效果 W^* 的值均较小，甚至小于前文对小样本量的拟合应用，所以我们认为利用 UExE 模型拟合该企业的生产状态的数据是合理的并且较为准确的。

表 5-8　两条生产线数据拟合结果

生产线	$\hat{\delta}$	$\hat{\lambda}$	A^*	W^*	**K-S**	p
M101	2.1778	79.3240	1.5468	0.0467	0.1227	0.0628
M102	2.5029	64.1217	1.6440	0.0464	0.0954	0.2460

数据的频数柱状图和拟合的概率密度函数如图5-3所示，其中左图为生产线 M101 的存活时间的概率密度曲线，右图为生产线 M102 的存活时间的概率密度曲线。我们利用 UExE 模型对其准确的建模，为生产企业提供了更加准确的生产决策：比如从图中可见生产线 M102 整体存活时间较高，而 M101 生产线较低，企业可以根据我们的对存活时间的准确建模结合生产合格率、生产速度等其他因素，综合对企业生产做出决策。

① 数据下载地址 https://www.tipdm.org:10010/#/competition/1825410816212639744/question

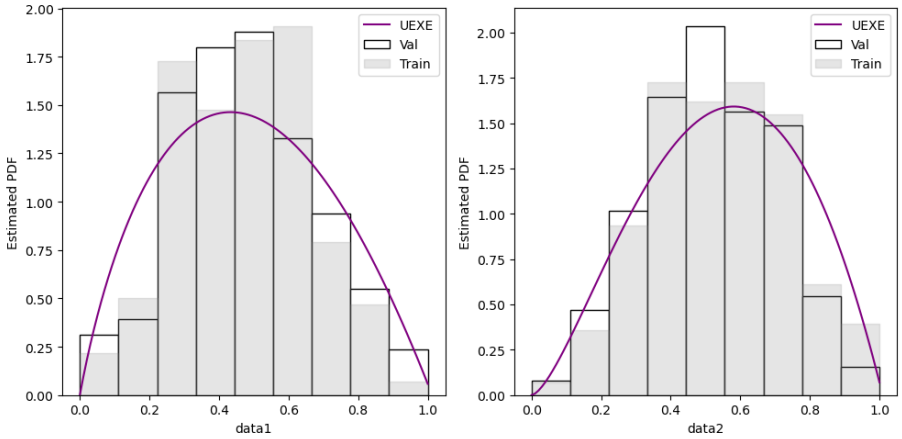


图 5-3　拟合概率密度函数与频数柱状图

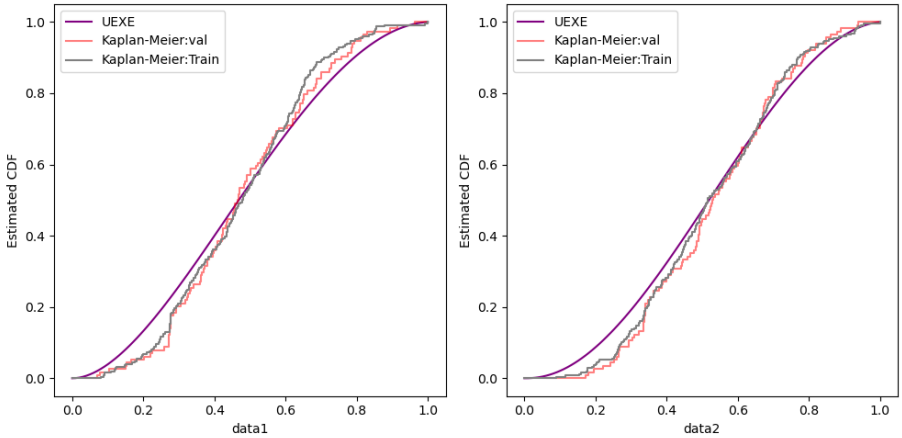


图 5-4　拟合概率分布函数与经验分布函数图

图5-4展示了两条生产线生存时间的分布，可见在较大的实际生产数据下，UExE 仍然能够表现出较为精确的拟合效果。这进一步论证了原论文所声明的 UExE 模型可以灵活地拟合分析生命周期数据和其建模的准确性。

5.3　总结分析

本章我们首先复现了原论文在两个小样本数据集下的应用，通过不同的评估指标，验证了其拟合能力的优越性。同时额外对这两个小样本数据集进行 Bootstrap 重抽样，对其做了更全面的估计量的偏差和方差分析，指出虽然模型拟合能力好，但是使用极大似然方法进行估计还是存在一定的局限性。随后我们在真实的企业生产大样本数据集上对 UExE 模型进行了进一步验证与应用，验证了其在大样本条件下仍然具有好的鲁棒性和准确性，该模型对数据的拟合具有的这样的性质为企业的生产决策提供了有力的帮助。

结　论

本文通过复现论文《Unit extended exponential distribution with applications》研究了一种新颖的单位分布 (即定义在 (0,1) 区间内) 的扩展指数分布，即 **UExE** 分布。

本文首先通过数学理论推导证明了该分布具有闭合形式的累积分布函数以及多种形态的危险率函数等丰富优秀性质。同时本文给出了 **UEXE** 模型两个参数 δ, λ 的极大似然估计的点估计，以及利用渐进正态性以及 Fisher 信息矩阵实现置信区间估计的理论推导。并利用基于**二分法的逆变换抽样**方法以及**蒙特卡洛方法**，通过模拟计算进一步评估说明了原论文所使用的点估计与置信区间估计方法具有高效率、低偏差、高精度的效果。

在复现原论文的真实数据集的拟合评估中，本文使用了 Akaike 信息准则、Hannon-Quinn 信息准则等多种估计量评估指标以及 Anderson-Darling（A*）、Kolmogorov-Smirnov（K-S）检验等多种**拟合优度检验**方法，实现了较为全面、多维度地评估模型的优劣。同时本文进一步将 UEXE 模型与 9 种传统的单位区间模型诸如 Beta 分布、Kumaraswamy 分布等进行对比，结合可视化结果验证说明了 UEXE 在多个指标与拟合优度检验方面更优于现存的单位分布竞争模型，具有较为优越的拟合预测性能。本文额外对两组真实小样本数据使用 **Bootstrap 重抽样方法**做了更全面的分析，计算了两组数据上极大似然估计的偏差与方差，指出了极大似然估计在该模型参数点估计中存在一定的局限性。

为了评估模型在更真实的使用场景以及大样本量数据下的表现，本文额外使用了 2024 年第七届泰迪杯数据分析技能赛 A 题-自动化生产线数据分析的数据，将其划分为拟合数据与验证数据，并在验证数据集上做了全面的评估，较为科学地验证了该 **UEXE** 模型在真实企业生产大样本数据拟合上依旧有较高的鲁棒性、高效性与准确性，从而更加科学地指导企业生产活动更加合理分配资源和决策。

本文**创造性工作**如下：

（1）创造性的实现了**基于二分法的逆变换抽样方法**，实现了系统高效的从复杂分布模型中快速产生样本。

（2）合理运用**牛顿-拉夫逊方法**（即牛顿迭代法）于极大似然估计任务中，

稳定高效地解决了极大似然估计任务中效率低、数值不稳定等问题。

（3）基于参数的渐近正态性，结合 **Fisher 信息矩阵**高效的计算了参数的置信区间，并结合**蒙特卡洛模拟**评估置信区间覆盖率与区间长度等指标。

（4）进一步探究了样本量对于点估计偏差与均方误差以及置信区间估计的覆盖率与区间长度的影响，**证明了**大样本量对于提升 **UEXE** 模型参数估计效果与精度的重要性。

（5）在小样本真实数据评估中，额外使用了 **Bootstrap** 重抽样方法**证明了**极大似然估计对于 **UEXE** 模型的参数估计存在一定的局限性。

（6）通过数据预处理，本文提供了两组真实的样本量更大的企业生产线生命周期数据，进一步评估了模型在真实场景下的表现与性能。

当然基于前文分析结果，本文的**局限性**如下：

(1) 基于第5章中多种评估指标分析以及 **Bootstrap** 重抽样方法的估计量偏差与方差评估，我们发现对 UExE 模型的两个参数 δ, λ 使用极大似然估计有一定的局限性，其估计偏差与方差相较于其他模型并不是特别出色。尤其是对于参数 δ 的估计更是具有非常大的 MLE 标准差，这也正说明使用极大似然估计具有比较大的不稳定性。后续研究可以着重寻找其他更为优秀稳定的、方差更小的参数估计方法。

（2）基于第4 中置信区间估计的图表分析，我们发现由于 **UEXE** 模型的置信区间估计中需要使用到渐进正态性这一结论，因此关于置信区间的估计在小样本量的时候有较大的不稳定性，其区间覆盖率以及区间长度相比于大样本表现的均较差。在后续研究中，可以着重于小样本的置信区间估计效果提升以及寻找其他更优的置信区间估计方法。

（3）基于模拟部分样本使用方式，本文的局限性在于模拟过程中使用的是完整分布的样本数据，并用其估算分布的未知参数。但在实际的生命周期数据中，完整分布的样本数据较难获得，因此今后的研究可以聚焦于使用不同的样本删减方案估算分布的参数，并评估其相应的效果。

参考文献

[1] RAGAB I E, ALSADAT N, BALOGUN O S, et al. Unit extended exponential distribution with applications [J/OL]. Journal of Radiation Research and Applied Sciences, 2024, 17 (4): 101118. https://www.sciencedirect.com/science/article/pii/S1687850724003029.

[2] QUESENBERRY C, HALES C. Concentration bands for uniformity plots [J/OL]. Journal of Statistical Computation and Simulation, 1980, 11 (1): 41-53. http://dx.doi.org/10.1080/00949658008810388.

[3] CARAMANIS M, STREMEL J, FLECK W, et al. Probabilistic production costing: An investigation of alternative algorithms [J/OL]. International Journal of Electrical Power Energy Systems, 1983, 5 (2): 75-86. https://www.sciencedirect.com/science/article/pii/014206158390011X.

[4] JOHNSON N L, KOTZ S, BALAKRISHNAN N. Continuous univariate distributions, volume 2: Vol 289 [M]. [S.l.]: John wiley & sons, 1995.

[5] KUMARASWAMY P. A generalized probability density function for double-bounded random processes [J/OL]. Journal of Hydrology, 1980, 46 (1): 79-88. https://www.sciencedirect.com/science/article/pii/0022169480900360.

[6] GRASSIA A. ON A FAMILY OF DISTRIBUTIONS WITH ARGUMENT BETWEEN 0 AND 1 OBTAINED BY TRANSFORMATION OF THE GAMMA AND DERIVED COMPOUND DISTRIBUTIONS [J/OL]. Australian Journal of Statistics, 1977, 19 (2): 108–114. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84990518249&doi=10.1111%2fj.1467-842X.1977.tb01277.x&partnerID=40&md5=a766f811188459ea5b0d90736dc17a37.

[7] ELGARHY M, SOLIMAN A, NAGY H. Parameter Estimation Methods and Applications of the Power Topp-Leone Distribution [J/OL]. Gazi University Journal of Science, 2022, 35 (2): 731–746. http://dx.doi.org/10.35378/gujs.776277.

[8] MALLICK A, GHOSH I, DEY S, et al. Bounded Weighted Exponential Distribution with Applications [J/OL]. American Journal of Mathematical and Management Sciences, 2020, 40 (1): 68–87. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85094126508&doi=10.1080%2f01966324.2020.1834893&partnerID=40&md5=8a6cd048600fcf3d80696ecde2697ea8.

[9] ZAYED M A，HASSAN A S，ALMETWALLY E M，et al. A Compound Class of Unit Burr XII Model: Theory, Estimation, Fuzzy, and Application [J/OL]. Scientific Programming，2023，2023. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85158889202&doi=10.1155%2f2023%2f4509889&partnerID=40&md5=37c16d30e680df50ed328c4e77b349f3.

[10] MAZUCHELI J，MENEZES A F，DEY S. The unit-Birnbaum-Saunders distribution with applications [J]. Chilean Journal of Statistics，2018，9（1）：47-57.

[11] J Mazucheli M. E. GHITANY A F B M，ALQALLAF F. The unit-inverse Gaussian distribution: A new alternative to two-parameter distributions on the unit interval [J/OL]. Communications in Statistics - Theory and Methods，2019，48（14）：3423-3438. http://dx.doi.org/10.1080/03610926.2018.1476717.

[12] AL-OMARI A I，ALANZI A R，ALSHQAQ S S. The unit two parameters Mirra distribution: Reliability analysis, properties, estimation and applications [J/OL]. Alexandria Engineering Journal，2024，92：238-253. https://www.sciencedirect.com/science/article/pii/S1110016824002023.

附录 1　论文原文

Unit extended exponential distribution with applications

Ibrahim E. Ragab [a], Najwan Alsadat [b], Oluwafemi Samson Balogun [c], Mohammed Elgarhy [d,e,*]

[a] Department of Basic Sciences, Egyptian Institute of Alexandria Academy for Management and Accounting, EIA, Alexandria, Egypt
[b] Department of Quantitative Analysis, College of Business Administration, King Saud University, P.O. Box 71115, Riyadh, 11587, Saudi Arabia
[c] Department of Computing, University of Eastern Finland, FI-70211, Finland
[d] Mathematics and Computer Science Department, Faculty of Science, Beni-Suef University, Beni-Suef, 62521, Egypt
[e] Department of Basic Sciences, Higher Institute for Administrative Sciences, Belbeis, AlSharkia, Egypt

ARTICLE INFO

Keywords:
Unit distributions
Extended exponential distribution
Entropy
Monte-Carlo simulation
Real data applications

ABSTRACT

In this paper, the unit extended exponential distribution is presented. The extended exponential distribution is a two-parameter lifespan distribution that has been demonstrated to be very adaptable. Our goal is to convert this flexibility between the unit interval and the extended exponential distribution. Several different types of probability density functions, including decreasing, left-skewed, right-skewed, and unimodal, have been shown. However, the hazard rate function may take on various forms, including J-shaped, U-shaped, and bathtub shapes. A few statistical properties, such as moments, mean, variance, skewness, kurtosis, expectation-weighted moments, incomplete moments, conditional moments, mean deviation, Lorenz and Bonferroni curves, mean residual life, mean inactivity time, and various entropy measures, round out the theoretical section. The maximum likelihood technique is utilized to estimate model parameters using unit data. This study provides simulation data to evaluate the proposed strategy. Two applications utilizing actual data sets emphasize the significance of the new model compared to existing unit models.

1. Introduction

Lifetime data analysis often prefers models with monotone risk functions, such as the gamma distribution. Several models, like the Gamma model, lack closed-form risk functions; therefore, their computation may necessitate numerical integration. Modified extensions of the exponential distributions have been presented in contemporary statistical literature to outline such challenges. For instance, the generalized exponential distribution was presented by (Gupta & Kundu, 1999) as an extension of the exponential distribution. Ref. (Nadarajah & Haghighi, 2011) proposed another extension of the exponential (E) model called the Nadarajah-Haghighi E distribution.

Ref. (Gómez et al., 2014) introduced the extended exponential distribution (ExE) with parameters δ and λ. Assuming Y is a non-negative random variable with the ExE, its cumulative distribution function (CDF) is given by:

$$H(y;\delta,\lambda)=1-\frac{[\delta+\lambda+\delta\lambda y]}{\delta+\lambda}e^{-\delta y}\ ;\ y>0,\delta,\lambda>0, \tag{1}$$

The relevant probability density function (PDF) is given as follows:

$$h(y;\delta,\lambda)=\frac{\delta^2\ [1+\lambda y]}{\delta+\lambda}e^{-\delta y}\ . \tag{2}$$

The ExE model is very flexible model because it has two well-known lifetime models. The ExE model reduces to the E model at $\delta=0$, also reduces to the Lindley model (Lindley, 1958) at $\lambda=1$.

Multiple authors in the statistical literature have concentrated on building new and more flexible statistical distributions by applying appropriate transformation techniques (see, for example (Alzaatreh et al., 2013; Jones, 2004). Most of the obtained distributions address unboundedly supported continuous random variables. In recent years, there has been an emphasis on better characterizing limited support distributions and actual events with confined ranges by bridging their gaps. Several others are more recent ideas. Research on unit models has dominated during the last ten years. Often generated by modifying random variables, these models interpret data with support in unit intervals. Among them are the unit-Burr III model (Modi & Gill, 2019; Singh et al., 2022), unit- Lindley model (Mazucheli, Menezes, & Chakraborty, 2019), unit-Gompertz model (Mazucheli, Menezes, & Dey, 2019), unit-Burr XII model (Korkmaz & Chesneau, 2021), unit-inverse

* Corresponding author. Mathematics and Computer Science Department, Faculty of Science, Beni-Suef University, Beni-Suef, 62521, Egypt.
E-mail addresses: ibrahim.ragab@eia.edu.eg (I.E. Ragab), nalsadat@ksu.edu.sa (N. Alsadat), samsb@student.uef.fi (O.S. Balogun), dr.moelgarhy@gmail.com (M. Elgarhy).

https://doi.org/10.1016/j.jrras.2024.101118
Received 22 July 2024; Received in revised form 10 September 2024; Accepted 15 September 2024
Available online 26 September 2024

Gaussian model (Ghitany et al., 2018), logit slash model (Korkmaz, 2019), generalized exponentiated unit Gompertz model (Sindhu et al., 2023), unit Gumbel Type-II model (Shafiq, Sindhu, Hussain, et al., 2023), Shafiq, Sindhu, Dey, et al., 2023 and half-logistic unit-Gompertz Type-I model (Shafiq, Sindhu, Dey, et al., 2023).

We intend to present a novel unit distribution with a (0,1) interval to generate a flexible density function with different shapes and skewed forms inside the unit interval. The distribution of UExE is essential because.

(I) It provides closed-form and simple-form formulations for the CDF.
(II) It can give J-shaped, U-shaped, and bathtub hazard rate function (HRF).
(III) It may yield more accurate fits than other well-known statistical distributions constructed on the unit interval.

Furthermore, we concentrate on a few specific motivations:

(1) To construct distributions with diverse density-shaped models.
(2) To investigate mathematical properties such as moments, mean, variance, skewness, kurtosis, moment-generating functions, probability-weighted moments, incomplete moments, conditional moments, mean deviation, Lorenz and Bonferroni curves, mean residual life, mean inactivity time and different measures of entropy.
(3) To estimate the UExE distribution parameters via the technique of maximum likelihood and analyze the results through simulation studies.
(4) To demonstrate the adaptability of the UExE model.
(5) To provide a model that is more suitable than other models.

The subsequent sections of the paper are structured as follows: Section 2 outlines the development of the UExE model. Section 3 contains numerous statistical characteristics of the UExE model. The parameter estimates of the UExE distribution using the maximum likelihood approach are presented in section 4. The behavior of estimated parameters using a Monte-Carlo simulation is investigated in Section 5. Section 6 focuses on demonstrating the results using actual data. Section 7 contains a set of final remarks.

2. The UExE model

Utilizing the transformation $X = e^{-Y}$, a novel unit model on the (0, 1) interval results, which is known as the unit extended exponential distribution (UExE). Therefore, the CDF of the UExE distribution is described as:

$$G(x;\delta,\lambda) = \left(1 - \frac{\delta\ \lambda\ \log(x)}{\delta+\lambda}\right)\ x^{\delta}\ ;\ \ x \in (0,1), \delta, \lambda > 0. \tag{3}$$

The accompanying PDF is presented as follows:

$$g(x;\delta,\lambda) = \frac{\delta^2[1-\lambda\ \log(x)]}{\delta+\lambda}\ x^{\delta-1};\ \ x \in (0,1), \delta, \lambda > 0. \tag{4}$$

Theorem 1. *The integration of the UExE PDF over the interval (0,1) equals one;* $\int_0^1 g(x;\delta,\lambda)\ dx = 1$.

Proof: Utilizing the PDF given in Equation (4), *we have*:

$$\int_0^1 g(x;\delta,\lambda)\ dx = \frac{\delta^2}{\delta+\lambda}\int_0^1 [1-\lambda\ \log(x)]\ x^{\delta-1}\ dx$$

$$= \frac{\delta^2}{\delta+\lambda}\left[\int_0^1 x^{\delta-1}\ dx - \lambda\int_0^1 x^{\delta-1}\log(x)\ dx\right]$$

$$= \frac{\delta^2}{\delta+\lambda}\left[\frac{1}{\delta} - \lambda\int_0^1 x^{\delta-1}\log(x)\ dx\right].$$

Now, letting $u = \log(x) \Rightarrow du = dx/x$ while $dv = x^{\delta-1}\ dx \Rightarrow v = x^{\delta}/\delta$, hence

$$\lambda\int_0^1 x^{\delta-1}\log(x)\ dx = \lambda\left(\frac{x^{\delta}}{\delta}\log(x)\right)\Big|_0^1 - \lambda\int_0^1 \frac{x^{\delta-1}}{\delta}\ dx = -\frac{\lambda}{\delta^2}$$

then,

$$\int_0^1 g(x;\delta,\lambda)\ dx = \frac{\delta^2}{\delta+\lambda}\left[\frac{1}{\delta}+\frac{\lambda}{\delta^2}\right] = 1\ .$$

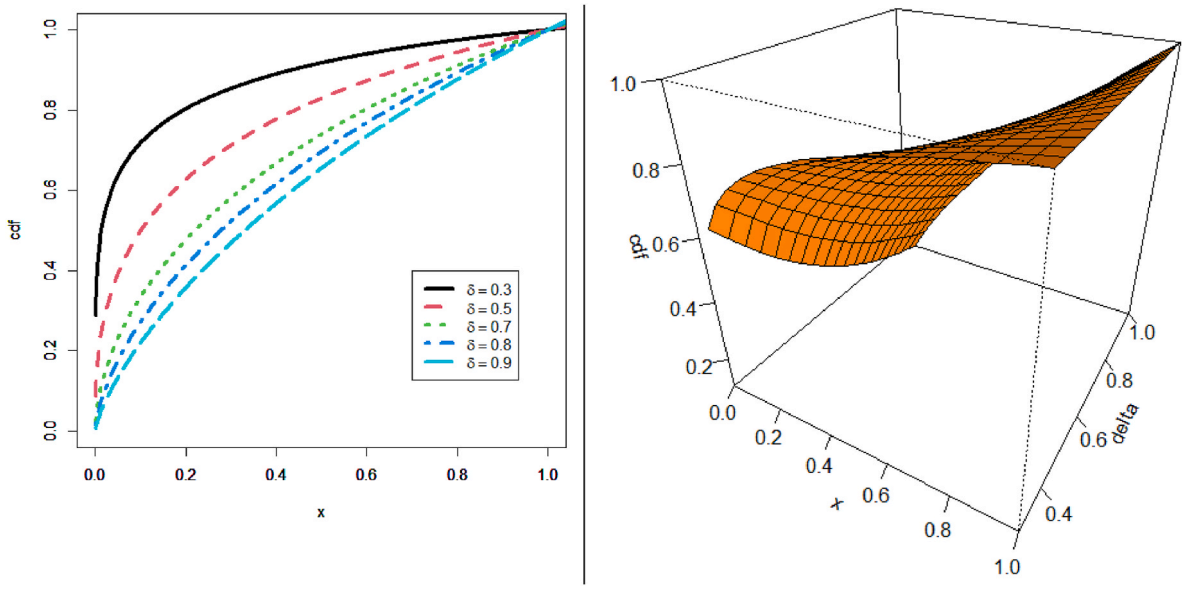


Fig. 1. 2D and 3D plots of CDF for the UExE distribution at $\lambda = 0.5$.

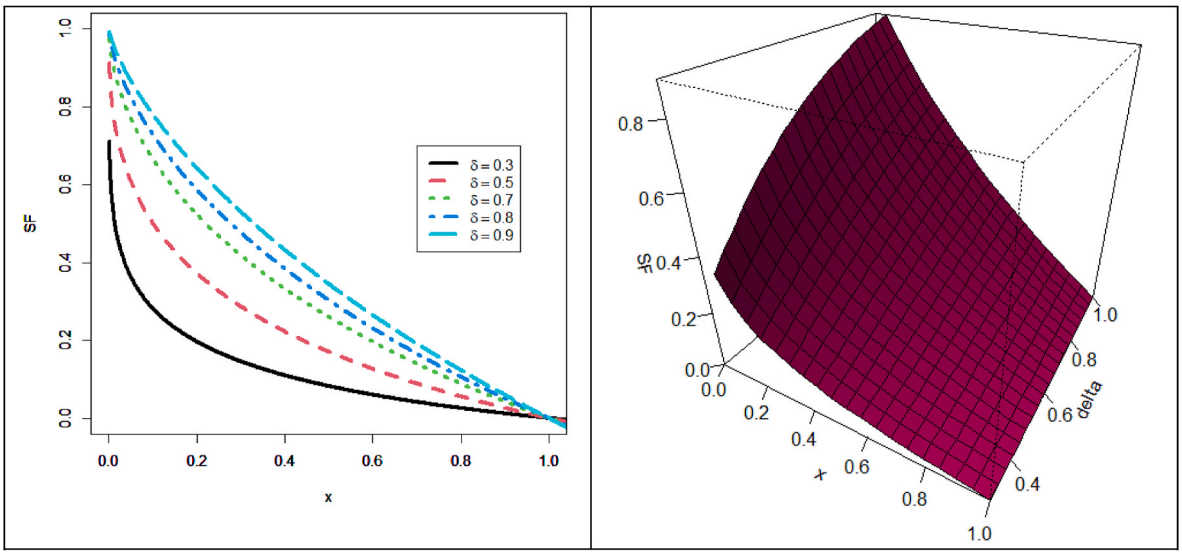


Fig. 2. 2D and 3D plots of SF for the UExE distribution at $\lambda = 0.5$.

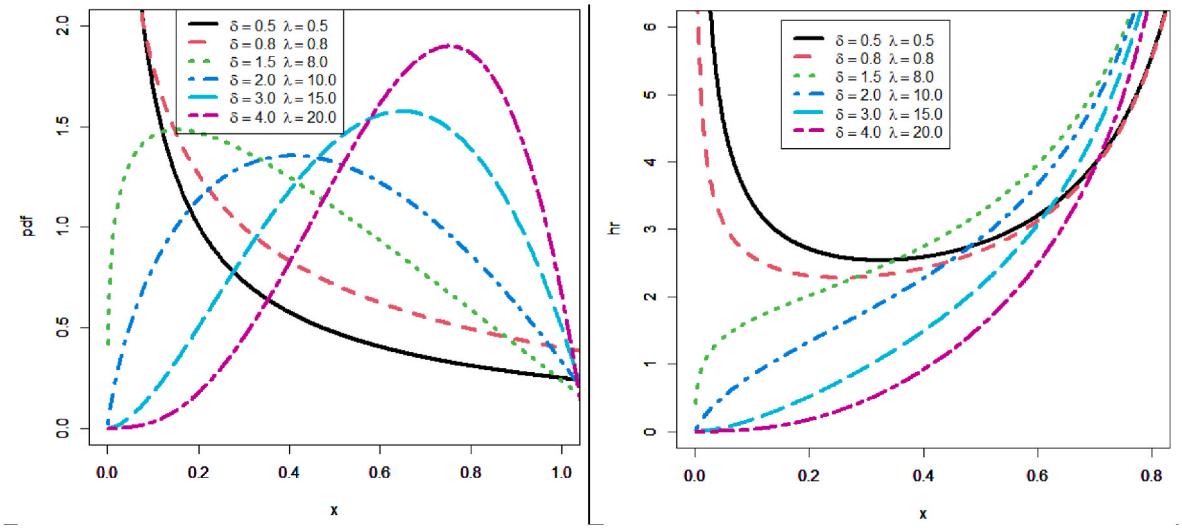


Fig. 3. 2D plots of PDF (left panel) and HRF (right panel) for the UExE distribution.

Now we completed the proof of Theorem 1.

The survival function (SF) indicates the probability that a particular system or item will survive beyond a specific time (t). The SF has significance in modeling and investigating time-to-event data, including failure times, disease onset, and a wide variety of other significant events that occur over time. The SF for the UExE distribution is derived as:

$$S(x;\delta,\lambda) = 1 - \left(1 - \frac{\delta\ \lambda\ \log(x)}{\delta+\lambda}\right)\ x^{\delta};\ \ x \in (0,1), \delta, \lambda > 0. \tag{5}$$

The HRF is a powerful tool that provides real-time information about the probability of a particular event occurring, making it an essential asset in fields such as medicine and engineering. This function offers valuable and ever-changing insights that can be applied in many practical applications. The HRF of the UExE distribution is specified by:

$$r(x;\delta,\lambda) = \frac{\delta^2[1-\lambda\log(x)]}{(\delta+\lambda)[x^{1-\delta}-x] + \delta\,\lambda\,x\log(x)};\ \ x \in (0,1), \delta, \lambda > 0. \tag{6}$$

Figs. 1 and 2: show 2D and 3D plots of CDF and SF for the UExE distribution at $\lambda = 0.5$. Fig. 3: shows 2D plots of the PDF and HRF for UExE model. It is observed from Fig. 3 that the PDF can be decreased, left skewed, right skewed and unimodal but the HRF can be J-shaped, U-shaped and bathtub.

3. Statistical properties

This section delves into vital statistical properties of the UExE distribution, including moments, mean, variance, skewness, kurtosis, moment-generating functions, incomplete moments, conditional moments, mean deviation, Lorenz and Bonferroni curves, mean residual life, mean inactivity time and different measures of entropy.

3.1. Ordinary moments

In this section, the first four ordinary moments, which are important in explaining the features of distributions are presented. These moments provide significant details about the shape, center, and spread of data distributions. They are frequently used by investigators and statisticians to comprehend and describe data behavior. The initial four moments are defined as:

Table 1
Numerical representation of the first four ordinary moments, σ_x^2, γ_1, γ_2 and the CV.

$\lambda\uparrow$	$\delta\uparrow$	$\mu_1'\uparrow$	$\mu_2'\uparrow$	$\mu_3'\uparrow$	$\mu_4'\uparrow$	$\sigma_x^2\downarrow$	$\gamma_1\downarrow$	$\gamma_2\uparrow$	$CV\downarrow$
1	1.5	0.504000	0.330612	0.244444	0.193388	0.076596	0.028627	1.858000	0.549128
	2	0.592593	0.416667	0.320000	0.259259	0.065501	−0.271064	2.028520	0.431884
	2.5	0.655977	0.485009	0.383707	0.316991	0.054703	−0.485805	2.306480	0.356549
	3	0.703125	0.540000	0.437500	0.367347	0.045615	−0.650142	2.612490	0.303754
	3.5	0.739369	0.584940	0.483235	0.411358	0.038274	−0.781132	2.917510	0.264600
	4	0.768000	0.622222	0.522449	0.450000	0.032398	−0.888509	3.209930	0.234369
1.5	1.5	0.480000	0.306122	0.222222	0.173554	0.075722	0.124293	1.886720	0.573286
	2	0.571429	0.392857	0.297143	0.238095	0.066327	−0.184331	1.983120	0.450694
	2.5	0.637755	0.462963	0.361570	0.295858	0.056231	−0.405757	2.214790	0.371822
	3	0.687500	0.520000	0.416667	0.346939	0.047344	−0.575749	2.489990	0.316489
	3.5	0.725926	0.566942	0.463905	0.392000	0.039974	−0.711773	2.774210	0.275420
	4	0.756364	0.606061	0.504638	0.431818	0.033975	−0.823721	3.052760	0.243695
2	1.5	0.462857	0.28863	0.206349	0.159386	0.074393	0.191624	1.92683	0.589276
	2	0.555556	0.375000	0.280000	0.222222	0.066358	−0.120773	1.967770	0.463681
	2.5	0.623583	0.445816	0.344353	0.279421	0.056961	−0.345001	2.162000	0.382731
	3	0.675000	0.504000	0.400000	0.330612	0.048375	−0.517517	2.410040	0.325842
	3.5	0.714927	0.552216	0.448090	0.376162	0.041096	−0.655976	2.674080	0.283554
	4	0.746667	0.592593	0.489796	0.416667	0.035082	−0.770314	2.937600	0.250849
2.5	1.5	0.45000	0.275510	0.194444	0.14876	0.0730102	0.241066	1.96726	0.600453
	2	0.54321	0.361111	0.266667	0.209877	0.066034	−0.072677	1.966480	0.473060
	2.5	0.612245	0.432099	0.330579	0.266272	0.057255	−0.297800	2.131030	0.390824
	3	0.664773	0.490909	0.386364	0.317254	0.048986	−0.471208	2.356320	0.332939
	3.5	0.705761	0.539945	0.434911	0.362963	0.041846	−0.610667	2.602460	0.289847
	4	0.738462	0.581197	0.477237	0.403846	0.035871	−0.726122	2.851800	0.256475
3	1.5	0.44000	0.265306	0.185185	0.140496	0.0717061	0.278578	2.00450	0.608591
	2	0.533333	0.350000	0.256000	0.200000	0.065556	−0.035306	1.971850	0.480072
	2.5	0.602968	0.420875	0.319309	0.255514	0.057305	−0.260347	2.112840	0.397008
	3	0.656250	0.480000	0.375000	0.306122	0.049336	−0.433771	2.319340	0.338464
	3.5	0.698006	0.529561	0.423760	0.351795	0.042349	−0.573421	2.550060	0.294825
	4	0.731429	0.571429	0.466472	0.392857	0.036441	−0.689238	2.786660	0.260989
3.5	1.5	0.432000	0.257143	0.177778	0.133884	0.0705189	0.307786	2.037620	0.614708
	2	0.525253	0.340909	0.247273	0.191919	0.065019	−0.005627	1.980240	0.485458
	2.5	0.595238	0.411523	0.309917	0.246548	0.057214	−0.230084	2.102380	0.401848
	3	0.649038	0.470769	0.365385	0.296703	0.049518	−0.403052	2.293350	0.342857
	3.5	0.691358	0.520661	0.414201	0.342222	0.042685	−0.542429	2.510940	0.298838
	4	0.725333	0.562963	0.457143	0.383333	0.036855	−0.658157	2.736340	0.264672

- Mean $\{\mu_x = \mu_1'\}$ also known as the expected value, represents the central tendency of a dataset.
- Variance $\{\sigma_x^2 = \mu_2' - (\mu_1')^2\}$ refers to the spread or dispersion of data points around the average value. It determines how much individual data points differ from the mean.
- Skewness $\{\gamma_1 = [\mu_3' - 3\,\mu_1'\,\mu_2' + 2\,(\mu_1')^3]\,/\sigma_x^3\}$ represents the model's asymmetry. Positive skewness suggests that the tail is longer on the right (right-skewed), whereas negative skewness shows a larger tail on the left.
- Kurtosis $\{\gamma_2 = [\mu_4' - 4\,\mu_1'\,\mu_3' + 6\,(\mu_1')^2\,\mu_2 - 3(\mu_1')^4]\,/\sigma_x^4\}$ is a measure of how a model's tails compare to those of the normal distribution. It determines whether the distribution is slightly or heavily peaked (leptokurtic) or flatter (platykurtic) than the normal distribution.

In addition, the coefficient of variation $\{CV = \sigma_x/\,\mu_1'\}$ and the index of dispersion $\{ID = \mu_1'/\sigma_x\}$ give a standardized measure of dispersion, which allowing for comparisons between datasets. A larger CV reflects higher relative variability (spread) among the data points. A lower CV indicates less relative variability and more consistency around the mean.

Theorem 2. *The r_{th} moment of the UExE distribution is given by*

$$\mu_r' = \frac{\delta^2\ (r+\delta+\lambda)}{(\delta+\lambda)\ (r+\delta)^2}\ ;\ r = 1,2,3,\ldots \tag{7}$$

Proof:

$$\mu_r' = \mathbb{E}[X^r] = \int_0^1 x^r g(x;\delta,\lambda)\ dx$$

$$= \frac{\delta^2}{\delta+\lambda}\int_0^1 [1-\lambda\ \log(x)]\,x^{r+\delta-1}\ dx$$

$$= \frac{\delta^2}{\delta+\lambda}\left[\int_0^1 x^{r+\delta-1}\,dx - \lambda\int_0^1 x^{r+\delta-1}\log(x)\,dx\right]$$

$$= \frac{\delta^2}{\delta+\lambda}\left[\frac{1}{r+\delta}+\frac{\lambda}{(r+\delta)^2}\right].$$

Now we completed the proof of Theorem 2.

In order to determine the origin's moments, insert $r = 1, 2, 3,$ or 4. The first four moments around the origin can be identified as:

$$\mu_1' = \frac{\delta^2\ (1+\delta+\lambda)}{(\delta+\lambda)\ (1+\delta)^2},\quad \mu_2' = \frac{\delta^2\ (2+\delta+\lambda)}{(\delta+\lambda)\ (2+\delta)^2},$$
$$\mu_3' = \frac{\delta^2\ (3+\delta+\lambda)}{(\delta+\lambda)\ (3+\delta)^2},\quad \mu_4' = \frac{\delta^2\ (4+\delta+\lambda)}{(\delta+\lambda)\ (4+\delta)^2}\ . \tag{8}$$

Applying Equation (8), the variance of UExE distribution can be determined as:

$$\sigma_x^2 = \frac{\delta^2}{(\delta+\lambda)^2}\left[\frac{(\delta+\lambda)(2+\delta+\lambda)}{(2+\delta)^2} - \frac{\delta^2(1+\delta+\lambda)^2}{(1+\delta)^4}\right]. \tag{9}$$

Table 1: indicates that ordinary moments can vary significantly depending on parameter values. This variation results for specific parameter λ values as well as different parameter δ choices. Increasing these parameters raises the first four moments while decreasing the

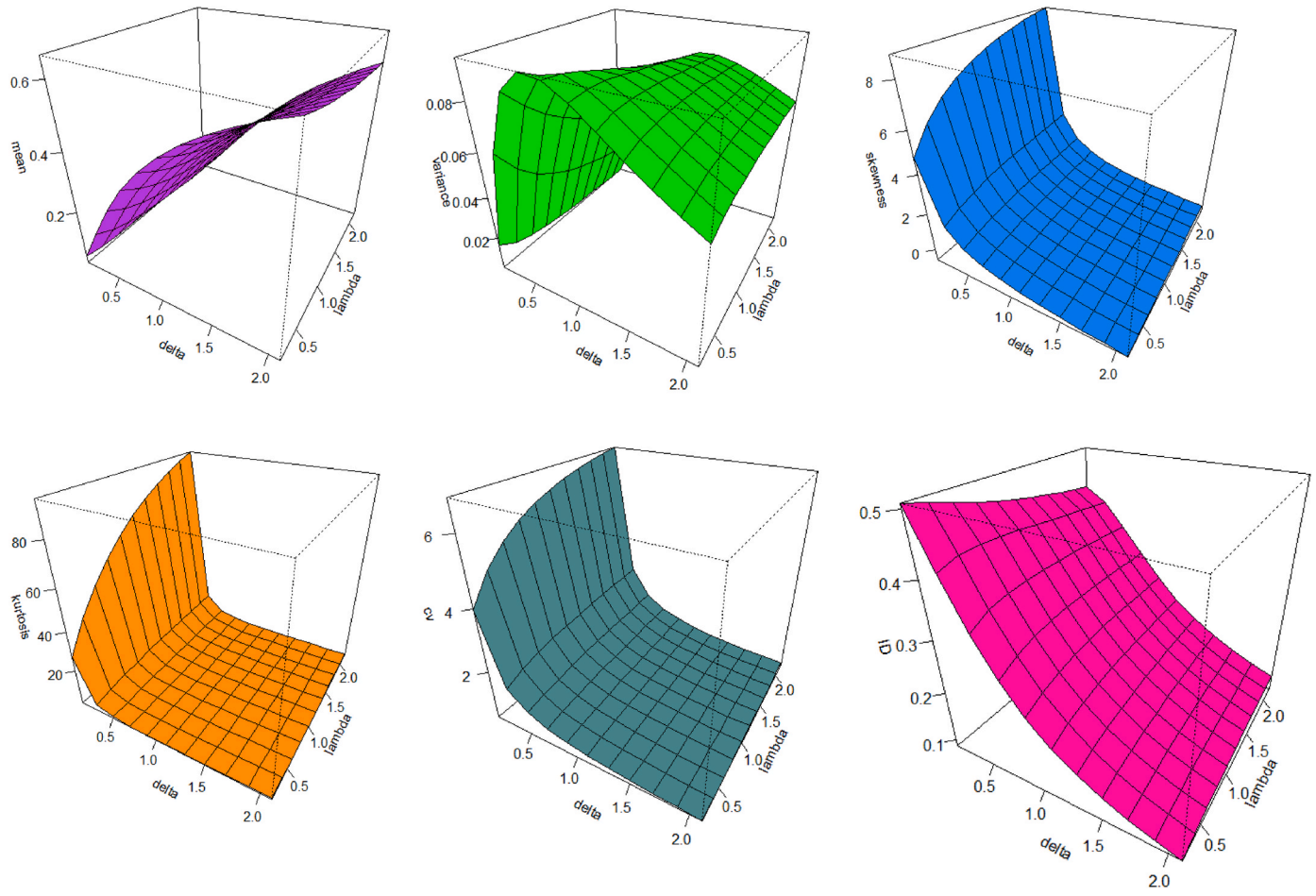


Fig. 4. 3D plots of the moments for the UExE distribution.

coefficient of variation. Fig. 4: depicts these findings graphically. Fig. 4 indicates the mean is decreased but, the skewness, kurtosis, CV and index of dispersion (ID) are increased for the selected values of parameters. Also, the variance is up-side-down.

3.2. Moment generating function

The moment-generating function (MGF) helps associate moments and probability distributions. The MGF of a random variable is a mathematical function that characterizes the probability distribution. It gives a systematic approach for calculating the moments of a random variable by differentiating the MGF at the point of origin. In particular, the n_{th} derivative of the MGF at zero indicates the n_{th} moment about the point of origin.

Theorem 3. *The MGF for UExE distribution is given by*:

$$\underset{t}{M}(x) = \frac{\delta^2\ (-t)^{-\delta}\ \gamma(\delta,-t) + \lambda_2 F_2(\delta,\delta;\delta+1,\delta+1;\ t)}{\delta+\lambda}\ . \tag{10}$$

Proof:

$$\underset{t}{M}(x) = \int_0^1 e^{t\,x} g(x;\delta,\lambda)\ d\,x$$

$$= \frac{\delta^2}{\delta+\lambda}\left[\int_0^1 x^{\delta-1}\ e^{tx}\ dx - \lambda\int_0^1 x^{\delta-1}\ e^{tx}\log(x)\ dx\right]$$

$$= \frac{\delta^2}{\delta+\lambda}\left[\frac{\gamma(\delta,-t)}{(-t)^{\delta}} + \frac{\lambda_2 F_2(\delta,\delta;\delta+1,\delta+1;\ t)}{\delta^2}\right]$$

$$= \frac{\delta^2\ (-t)^{-\delta}\ \gamma(\delta,-t) + \lambda_2 F_2(\delta,\delta;\delta+1,\delta+1;\ t)}{\delta+\lambda}\ .$$

Now we completed the proof of Theorem 3. Where, $\gamma(s,\varepsilon) = \int_0^{\varepsilon} z^{s-1}\ \exp[-z]\ dz$ identifies the lower incomplete gamma function.

The term: ${}_2F_2[\delta,\delta;\delta+1,\delta+1;\ t] = \delta\int_0^1 {}_1F_1[\delta;\delta+1;t\,y]\ dy = \delta^2\int_0^1\int_0^1 e^{t\,u\,y}(u\,y)^{\delta-1}du\,dy$ represents the generalized hypergeometric function ((Progri, 2020)).

3.3. Incomplete moments

Incomplete moments are statistical metrics that focus on certain subsets of data to provide additional insight on its distribution. They facilitate comparisons between different groups or subsets within a dataset. The s_{th} incomplete moments of the UExE distribution are represented as follows:

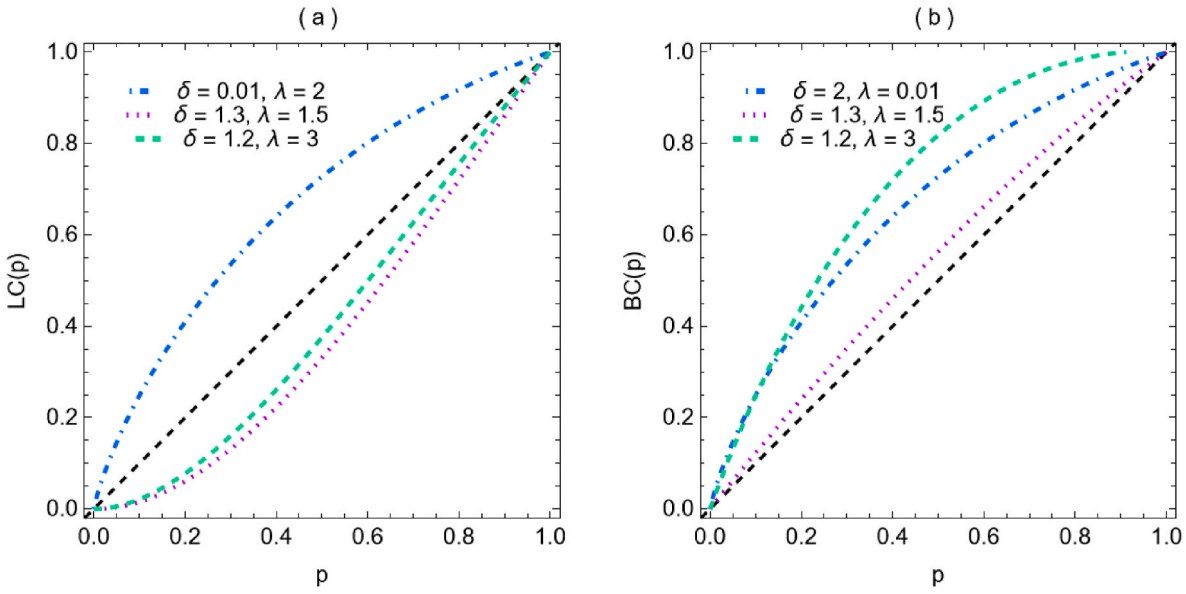


Fig. 5. LC(p) and BC(p) curves for different values of the UExE(δ, λ) distribution.

$$\eta_s(x) = \int_0^t x^s g(x;\delta,\lambda)\ dx$$
$$= \frac{\delta^2}{\delta+\lambda}\left[\int_0^t x^{s+\delta-1}\,dx - \lambda\int_0^t x^{s+\delta-1}\log(x)\,dx\right] \quad (11)$$
$$= \frac{\delta^2}{\delta+\lambda}\left[\frac{t^{s+\delta}}{s+\delta} + \lambda\frac{t^{s+\delta}}{(s+\delta)^2} - \lambda\frac{t^{s+\delta}}{s+\delta}\log t\right]$$
$$= \frac{\delta^2\ t^{s+\delta}\left[s+\delta+\lambda-(s+\delta)\,\lambda\log(t)\right]}{(\delta+\lambda)\ (s+\delta)^2}\ ;\ s=1,2,3,\ldots.$$

Applying Equation (11) for ($s = 1$), the mean deviation around the mean (μ_x) or the median (M_0) has been determined by the following forms:

$\Delta_1(x) = \mathbb{E}[|\ X-\mu_x\ |] = 2\,\mu_x\,G(\mu_x) - 2\eta_1(\mu_x)$, or $\Delta_2(x) = \mathbb{E}[|\ X-M_0\ |] = \mu_x - 2\,\eta_1(M_0)$, where $G(\mu_x)$ is calculated from Equation (3).

Incomplete moments can improve empirical explanations of distributions and inequality measures, including the widely used Lorenz {LC(p)} and Bonferroni {BC(p)} curves for assessing income and wealth inequality. The LC(p) shows the proportion of total income controlled by individuals with incomes less than or equal to a specific level. Whereas, the Bi(p) is a scaled conditional mean curve that reflects the ratio of group mean income within the population. The LC(p) and BC(p) can be computed by using the first incomplete moment ($s = 1$) given in Equation (11) and insert it into the following formulas:

$$LC(p) = \frac{1}{\mu_1}\int_0^q x\,g(x;\delta,\lambda)\ dx = \frac{\eta_1(q)}{\mu_1}$$
$$= \frac{q^{1+\delta}[1+\delta+\lambda-(1+\delta)\,\lambda\log(q)]}{1+\delta+\lambda},$$

and

$$BC(p) = \frac{1}{p\times\mu_1}\int_0^q x\,g(x;\delta,\lambda)\ dx = \frac{\eta_1(q)}{p\times\mu_1}$$
$$= \frac{q\ (\alpha+\delta)[1+\alpha+\delta-\alpha\,(\delta+1)\log(q)]}{(1+\alpha+\delta)[\alpha+\delta-\alpha\,\delta\log(q)]};\ p = G(q).$$

Fig. 5: depicts the LC(p) and BC(p) curves for some parameter values. In economics, if $p = G(q)$ represents the proportion of units earning less than or equal to q, then LC(p) indicates the proportion of total income held by those units. Similarly, the BC(p) measures the mean income of this group relative to the population's mean income.

3.4. Conditional moments

Conditional moments are the moments of a random variable under specific criteria or constraints. These moments provide helpful information about the random variable's behavior under specific conditions, allowing for a deeper understanding of the relationship between variables and the underlying data distribution. The conditional moments of the UExE distribution can be derived as:

$$\mathbb{E}(X^s|X>t) = \frac{\nu_s(t)}{S(x;\delta,\lambda)} \quad (12)$$

Let us derive the term $\nu_s(t)$ as follows:

Table 2
The MRL and MIT of the UExE distribution at $t = 0.7$.

$\lambda\uparrow$	$\delta\uparrow$	$MRL\uparrow$	$MIT\downarrow$	$\lambda\uparrow$	$\delta\uparrow$	$MRL\uparrow$	$MIT\downarrow$
2	0.5	0.132326	0.575573	4	0.5	0.124509	0.586018
	1	0.136703	0.444255		1	0.128746	0.458921
	1.5	0.141103	0.353523		1.5	0.133027	0.367958
	2	0.145512	0.290663		2	0.137341	0.303614
	2.5	0.149916	0.245471		2.5	0.141675	0.256764
	3	0.154302	0.211764		3	0.146015	0.221542
2.5	0.5	0.130017	0.579524	6	0.5	0.119346	0.589960
	1	0.134361	0.44962		1	0.123449	0.464879
	1.5	0.138734	0.358688		1.5	0.127610	0.374117
	2	0.143123	0.295225		2	0.131816	0.309337
	2.5	0.147515	0.249403		2.5	0.136056	0.261887
	3	0.151896	0.215137		3	0.140319	0.226070
3	0.5	0.127972	0.582321	7.5	0.5	0.116495	0.591610
	1	0.132280	0.453550		1	0.120510	0.467449
	1.5	0.136624	0.362555		1.5	0.124588	0.376829
	2	0.140990	0.298692		2	0.128718	0.311896
	2.5	0.145365	0.252424		2.5	0.132890	0.264205
	3	0.149736	0.217752		3	0.137092	0.228137

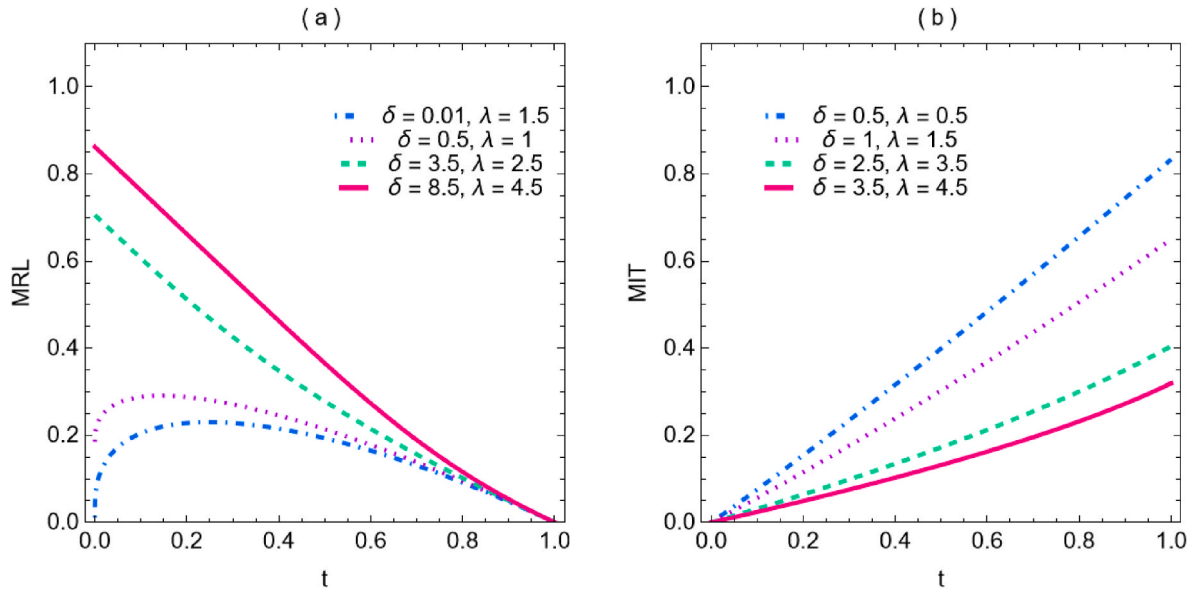


Fig. 6. The MRL (left panel) and MIT (right panel) for the UExE distribution.

$$\nu_s(t) = \int_t^1 x^s g(x;\delta,\lambda)dx = \frac{\delta^2}{\delta+\lambda}\left[\int_t^1 x^{s+\delta-1}dx - \lambda\int_t^1 x^{s+\delta-1}log(x)dx\right]$$
$$= \frac{\delta^2}{\delta+\lambda}\left[\frac{1-t^{\delta+s}}{s+\delta} - \frac{\lambda(\{1-(s+\delta)\ log(t)\}t^{s+\delta}-1)}{(s+\delta)^2}\right]$$
$$= \frac{\delta^2[(s+\delta)\ t^{\delta+s}\ \lambda\ log(t)\ -(s+\delta+\lambda)(t^{s+\delta}-1)]}{(\delta+\lambda)(s+\delta)^2}\ ;\ s=1,2,3,\ldots \tag{13}$$

3.5. Mean residual life and mean inactivity time

Mean Residual Life (MRL), commonly defined as the expected remaining lifetime, is vital in several domains, including survival analysis and reliability theory. It provides essential information on the average remaining lifetime of individuals or systems alive at a particular time. This information can be used to estimate future survival and mortality rates in various sectors, including medical treatment, actuarial studies, and the social sciences. The MRL function for the UExE distribution is given by:

$$MRL(t) = \mathbb{E}[X-t|\ X>t] = \frac{\nu_1(t)}{S(t;\delta,\lambda)} - t,$$

By using Equation (13) to determine the first conditional moment ($s = 1$):

$$\nu_1(t) = \frac{\delta^2[(\delta+1)\ t^{\delta+1}\ \lambda\ \log(t)\ -(\delta+\lambda+1)(t^{\delta+1}-1)]}{(\delta+\lambda)(\delta+1)^2}\ ; \tag{14}$$

Utilizing Equation (5 &14), yields the MRL function as follows:

$$MRL(t) = \frac{\delta^2[(1+\delta)\lambda t^{1+\delta}\log(t) + (1+\delta+\lambda)(1-t^{1+\delta})]}{(1+\delta)^2[\delta\lambda t^{\delta}\log(t) + (\delta+\lambda)(1-t^{\delta})]} - t\,.$$

Mean Inactivity Time (MIT) is a significant statistic in several disciplines, including reliability theory and survival analysis. It provides insights into the average duration of system or item idleness, which improves comprehension and system or item reliability and performance. The MIT function for UExE distribution is defined as:

$$MIT(t) = \mathbb{E}[t-X\ |\ X<t] = t - \frac{\eta_1(t)}{G(t;\delta,\lambda)}$$

By substituting $s = 1$ into Equation (11) and applying Equation (3), the MIT function simplifies to the following form:

$$MIT(t) = \frac{t}{(1+\delta)^2}\left[1+\delta+\frac{\delta\lambda}{\delta+\lambda-\delta\lambda\log(t)}\right]\ ;\ 0<t<1\,. \tag{15}$$

Table 2: displays the MRL and MIT at the point ($t = 0.7$) for the UExE distribution depending on different parameter choices. It has been observed that when parameter values grow, the MRL increases while the MIT decreases.

Fig. 6: depicts various shapes of the MRL (left) and MIT (right) depending on distribution parameter values.

3.6. Entropy

Entropy measures the uncertainty or randomness associated with a random variable or probability distribution. It specifies how much information is required to describe an observation selected from the distribution. High entropy indicates greater unpredictability or uncertainty in the data, whereas low entropy illustrates greater certainty or less unpredictability. This section will briefly explore the several types of entropy:

- Rényi entropy (RE) (Renner & Wolf, 2004), described as a parametric relaxation of Shannon's entropy (Shannon, 1948), provides a flexible framework for assessing uncertainty and information content. The RE of a random variable X following the UExE distribution can be computed as:

$$RE(\rho) = \frac{1}{1-\rho}\log\left[\int_0^1 g^{\rho}(x;\delta,\lambda)\,dx\right]\ ;\ \rho>0, \rho\neq 1$$

The integral of $g^{\rho}(x;\delta,\lambda)$ can be obtained as:

$$\int_0^1\left(\frac{\delta^2 x^{\delta-1}[1-\lambda\log(x)]}{\delta+\lambda}\right)^{\rho}dx = \frac{\delta^{2\rho}e^{\frac{(\delta-1)\rho+1}{\lambda}}}{\lambda(\delta+\lambda)^{\rho}}\mathbb{E}_{-\rho}\left(\frac{(\delta-1)\ \rho+1}{\lambda}\right);\ \rho < 1+\delta\rho.$$

where, $\mathbb{E}_i(y) = \int_{-z}^{\infty}[e^{-t}/t]dt$ is the exponential integral function.

Hence, the RE is given by:

$$RE(\rho)=\frac{1}{1-\rho}\log\left[\frac{\delta^{2\rho}\ e^{\frac{(\delta-1)\rho+1}{\lambda}}}{\lambda(\delta+\lambda)^{\rho}}\mathbb{E}_{-\rho}\left(\frac{(\delta-1)\ \rho+1}{\lambda}\right)\right].$$

- Tsallis entropy (TE) (Tsallis, 2019) has been widely employed in several disciplines due to its adaptability to a wide range of systems. TE has proven to be a useful tool for evaluating complex systems in a variety of fields, including physics, image and signal processing, brain activity evaluation, and picture segmentation. For UExE distribution, the TE is defined as:

$$TS(\rho)=\frac{1}{\rho-1}\left[1-\int_0^1 g^{\rho}(x;\delta,\lambda)\ dx\right]\ ;\rho>0,\rho\neq 1$$

$$=\frac{1}{\rho-1}\left[1-\frac{\delta^{2\rho}\ e^{\frac{(\delta-1)\rho+1}{\lambda}}}{\lambda(\delta+\lambda)^{\rho}}\mathbb{E}_{-\rho}\left(\frac{(\delta-1)\ \rho+1}{\lambda}\right)\right].$$

- Arimoto entropy (AE) (Arimoto, 1971) is utilized in image fusion and pattern recognition. It measures the information present in a fused image created from two input images. The AE of UExE distribution is derived as:

$$AR(\rho)=\frac{\rho}{1-\rho}\left[\left(\int_0^1 g^{\rho}(x;\delta,\lambda)\ dx\right)^{\frac{1}{\rho}}-1\right]\ ;\rho>0,\rho\neq 1$$

$$=\frac{\rho}{1-\rho}\left[\left(\frac{\delta^{2\rho}\ e^{\frac{(\delta-1)\rho+1}{\lambda}}}{\lambda(\delta+\lambda)^{\rho}}\mathbb{E}_{-\rho}\left(\frac{(\delta-1)\ \rho+1}{\lambda}\right)\right)^{\frac{1}{\rho}}-1\right].$$

- Havrda and Charvát (Havrda & Charvát, 1967) suggested a parameterized cross-entropy (HCE) to enhance the traditional Shannon-based cross-entropy. In contrast to Shannon entropy with a set logarithm base (usually 2), the Havrda-Charvát entropy offers greater flexibility. The HCE of UExE model is described as:

$$HCE(\rho)=\frac{1}{2^{1-\rho}-1}\left[\int_0^{\infty} g^{\rho}(x;\delta,\lambda)\ dx-1\right]\ ;\ \rho>0,\rho\neq 1$$

$$=\frac{1}{2^{1-\rho}-1}\left[\frac{\delta^{2\rho}\ e^{\frac{(\delta-1)\rho+1}{\lambda}}}{\lambda(\delta+\lambda)^{\rho}}\mathbb{E}_{-\rho}\left(\frac{(\delta-1)\ \rho+1}{\lambda}\right)-1\right].$$

3.7. Order statistics

Order statistics are values in a sample presented in ascending order that reveal information about data distribution and sample characteristics. They include minimum and maximum values and percentiles, such as median and quartiles, illustrating data distribution and central tendency. The pdf of the r_{th} order statistic is described as:

$$g_{r:n}(x)=C_r^n[G(x;\delta,\lambda)]^{r-1}[1-G(x;\delta,\lambda)]^{n-r}g(x;\delta,\lambda)\ ;\ C_r^n\frac{n!}{(r-1)!(n-r)!}.$$

Utilizing Equation (3&4), the PDF of the r_{th} order statistic of the UExE distribution can be calculated as:

$$g_{r:n}(x)=C_r^n\frac{\delta^2\ x^{\delta-1}[1-\lambda\log(x)]}{\delta+\lambda}\left[\left(1-\frac{\delta\ \lambda\ \log(x)}{\delta+\lambda}\right)\ x^{\delta}\right]^{r-1}\left[1-\left(1-\frac{\delta\ \lambda\ \log(x)}{\delta+\lambda}\right)\ x^{\delta}\right]^{n-r}.$$

Furthermore, the CDF of the r_{th} order statistic is described as:

$$G_{r:n}(x)=\sum_{k=r}^{n}\binom{n}{k}\ [G(x;\delta,\lambda)]^k[1-G(x;\delta,\lambda)]^{n-k}\ .$$

Using Equation (3), the CDF of the r_{th} order statistic for UExE distribution is given by:

$$G_{r:n}(x)=\sum_{k=r}^{n}\binom{n}{k}\left[\left(1-\frac{\delta\ \lambda\ \log(x)}{\delta+\lambda}\right)\ x^{\delta}\right]^k\left[1-\left(1-\frac{\delta\ \lambda\ \log(x)}{\delta+\lambda}\right)\ x^{\delta}\right]^{n-k}.$$

4. Parameter estimation

This section discusses estimating parameters for the UExE distribution, along with asymptotic confidence intervals for these parameters.

4.1. Maximum likelihood estimation

Maximum likelihood estimation (MLE) is a widespread technique recognized for its consistency, efficiency, and asymptotic normality. It analyzes observed data to identify the optimal parameter values for a statistical model by maximizing the likelihood function, thereby determining the parameters that best explain the observed data under the assumed distribution.

Let $X_1, X_2, \ldots, X_n$ be a random sample of size n and $x_1, x_2, \ldots, x_n$ be its observed values selected from the UExE distribution with parameters λ and δ. The likelihood function is described as:

$$L(\delta,\lambda|x_1,x_2,\ldots,x_n)=\prod_{i=1}^{n}g(x_i;\delta,\lambda)=\prod_{i=1}^{n}\frac{\delta^2[1-\lambda\ \log(x_i)]}{\delta+\lambda}\ x_i^{\delta-1},$$

The corresponding log-likelihood function is provided by

$$l(\delta,\lambda)=2n\log(\delta)-n\log(\delta+\lambda)+\sum_{i=1}^{n}\log[1-\lambda\log(x_i)]+(\delta-1)\sum_{i=1}^{n}\log(x_i). \tag{16}$$

The maximum likelihood estimates of δ and λ, represented as $\widehat{\delta}$ and $\widehat{\lambda}$, that maximize $l(\delta,\lambda)$. The partial derivatives of $l(\delta,\lambda)$ relative to each parameter are set to zero as follows:

$$\frac{\partial l(\delta,\lambda)}{\partial\delta}=\frac{2n}{\delta}-\frac{n}{\delta+\lambda}+\sum_{i=1}^{n}\log(x_i), \tag{17}$$

$$\frac{\partial l(\delta,\lambda)}{\partial\lambda}=-\frac{n}{\delta+\lambda}-\sum_{i=1}^{n}\frac{\log(x_i)}{1-\lambda\log(x_i)}, \tag{18}$$

Equation 17 and 18 can be solved using optimization techniques like the Newton-Raphson method.

4.2. Interval estimation

To estimate the intervals of (δ,λ), it's necessary to use the information matrix. The observed information matrix $I_n(\delta,\lambda)$ is defined as:

$$I_n(\delta,\lambda)=-\begin{pmatrix}I_{\delta\delta}\ I_{\delta\lambda}\\ I_{\lambda\delta}\ I_{\lambda\lambda}\end{pmatrix}_{(\delta,\lambda)=(\widehat{\delta},\widehat{\lambda})},$$

where the elements of the matrix $I_n(\delta,\lambda)$ are provided by:

$$I_{\delta\delta}=\frac{\partial^2 l(\delta,\lambda)}{\partial\delta^2}=-\frac{2n}{\delta^2}+\frac{n}{(\delta+\lambda)^2},$$

$$I_{\delta\lambda}=I_{\lambda\delta}=\frac{\partial^2 l(\delta,\lambda)}{\partial\delta\partial\lambda}=\frac{n}{(\delta+\lambda)^2},$$

and

$$I_{\lambda\lambda}=\frac{\partial^2 l(\delta,\lambda)}{\partial\lambda^2}=\frac{n}{(\delta+\lambda)^2}-\sum_{i=1}^{n}\frac{\log(x_i^2)}{(1-\lambda\log(x_i))^2}.$$

Thus, the variance-covariance matrix will be $I_n^{-1}(\delta,\lambda)$. where $I_n^{-1}(\delta,\lambda)$ represents the inverse of the observed information matrix. The approximate $(1-\varphi)100\%$ confidence intervals for the parameters δ and λ are: $\widehat{\delta}\pm Z_{\varphi/2}\sqrt{\mathrm{var}(\widehat{\delta})}$, and $\widehat{\lambda}\pm Z_{\varphi/2}\sqrt{\mathrm{var}(\widehat{\lambda})}$, where $\mathrm{var}(\widehat{\delta})$ and $\mathrm{var}(\widehat{\lambda})$ are asymptotic variances were obtained from the diagonal elements of $I_n^{-1}(\delta,\lambda)$. The term $Z_{\varphi/2}$ is the upper $(\varphi/2)$ percentile of the standard normal distribution.

5. Simulation study

Monte Carlo simulation is a computing approach that utilizes random sampling to simulate complex mathematical problems. It involves selecting random data points from a probability distribution in order to estimate the distribution's parameters. This technique can be beneficial when obtaining an exact analytical solution when it is complex. We employed Wolfram Mathematica (13) to run the simulation and analyze the results. The parameter estimators $(\widehat{\delta},\widehat{\lambda})$ of the proposed distribution were evaluated by simulating the following cases:

Case I: ($\delta=0.5,\lambda=0.5$), Case II: ($\delta=1,\delta=2$) and Case III: ($\delta=3$, $\lambda=4$).

A simulation study is conducted using the following steps:

1 Generate 1000 random samples of size (100, 150, 200, 300, 400, and 500) from the UExE distribution by applying the inverse CDF to uniformly distributed random numbers, using Equation (5).
2 Determine the average of estimates for 1000 random samples using this relation:

$$Mean(\widehat{\varpi})=\frac{1}{1000}\sum_{i=1}^{1000}\widehat{\varpi}_i\,.$$

3 Compute the average bias and MSE of the estimates from the following formulas:

$$Bias(\widehat{\varpi})=\frac{1}{1000}\sum_{i=1}^{1000}(\widehat{\varpi}_i-\widehat{\varpi}),$$

$$MSE(\widehat{\varpi})=\frac{1}{1000}\sum_{i=1}^{1000}(\widehat{\varpi}_i-\varpi)^2,\ \varpi=(\delta,\lambda).$$

4 Calculate the average length (AL) of the simulated confidence interval using the lower (LCI) and upper (UCI) limits, as follows:

$$LCI(\widehat{\varpi})=\widehat{\varpi}-Z_{\varphi/2}\sqrt{\mathrm{var}(\widehat{\varpi})},$$
$$UCI(\widehat{\varpi})=\widehat{\varpi}+Z_{\varphi/2}\sqrt{\mathrm{var}(\widehat{\varpi})},$$
$$AL(\widehat{\varpi})=UCI(\widehat{\varpi})-LCI(\widehat{\varpi}).$$

5 Calculate the coverage probability (CP), which is the proportion of simulated confidence intervals that include the parameter ϖ.

Tables (3–5) demonstrate the simulation results, which led us to the following conclusion:

- The bias and MSE of the MLE estimates approached zero with larger sample size.
- As sample sizes increased, the 95% confidence intervals became more compact.
- The estimate of δ and λ are said to be an overestimate.
- The CP of the confidence interval remains near to the expected 95% level, demonstrating the ML E's accuracy even on larger scales.

Table 3
The results of the simulation study (Case I).

$\delta=0.5$	$n=100$	$n=150$	$n=200$	$n=300$	$n=400$	$n=500$
$\widehat{\delta}$	0.50533	0.50457	0.50372	0.50352	0.50342	0.50324
Bias	0.00533	0.00457	0.00372	0.00352	0.00342	0.00324
MSE	0.00573	0.00388	0.00300	0.00187	0.00137	0.00114
$CPs_{95\%}$	0.938	0.946	0.941	0.943	0.955	0.954
$AL_{95\%}$	0.29604	0.24350	0.21424	0.16837	0.14467	0.13185
$\lambda=0.5$	$n=100$	$n=150$	$n=200$	$n=300$	$n=400$	$n=500$
$\widehat{\lambda}$	0.71512	0.64576	0.61942	0.58994	0.56014	0.55537
Bias	0.21512	0.14576	0.11942	0.08994	0.06014	0.05537
MSE	0.34807	0.21187	0.16201	0.10120	0.06641	0.05295
$CPs_{95\%}$	0.904	0.921	0.927	0.934	0.940	0.946
$AL_{95\%}$	2.15457	1.71236	1.50750	1.19673	0.98278	0.87592

Table 4
The results of the simulation study (Case II).

$\delta=1$	$n=100$	$n=150$	$n=200$	$n=300$	$n=400$	$n=500$
$\widehat{\delta}$	1.00619	1.00612	1.00539	1.00521	1.00367	1.00284
Bias	0.00619	0.00612	0.00539	0.00521	0.00367	0.00284
MSE	0.01440	0.00947	0.00704	0.00458	0.00342	0.00287
$CPs_{95\%}$	0.946	0.952	0.947	0.945	0.954	0.959
$AL_{95\%}$	0.47011	0.38098	0.32832	0.26448	0.22895	0.20995
$\lambda=1$	$n=100$	$n=150$	$n=200$	$n=300$	$n=400$	$n=500$
$\widehat{\lambda}$	2.69492	2.52529	2.43662	2.32934	2.21273	2.18983
Bias	0.69492	0.52529	0.43662	0.32934	0.21274	0.18983
MSE	3.59165	2.51646	1.99147	1.28001	0.84280	0.67790
$CPs_{95\%}$	0.921	0.901	0.912	0.925	0.933	0.948
$AL_{95\%}$	6.91504	5.87055	5.26309	4.24504	3.50252	3.14214

Table 5
The results of the simulation study (Case III).

$\delta=1$	$n=100$	$n=150$	$n=200$	$n=300$	$n=400$	$n=500$
$\widehat{\delta}$	3.00961	3.00784	3.00611	3.00362	3.00234	3.00137
Bias	0.00961	0.00784	0.00611	0.00362	0.00234	0.00137
MSE	0.10092	0.10083	0.07902	0.04913	0.03789	0.03203
$CPs_{95\%}$	0.946	0.955	0.944	0.945	0.959	0.958
$AL_{95\%}$	1.51704	1.24529	1.10241	0.86761	0.76285	0.70139
$\lambda=1$	$n=100$	$n=150$	$n=200$	$n=300$	$n=400$	$n=500$
$\widehat{\lambda}$	4.67562	4.57861	4.52806	4.46784	4.32303	4.31182
Bias	0.67562	0.57861	0.52806	0.46784	0.32303	0.31182
MSE	7.08393	5.45917	4.82844	3.57887	2.73383	2.30280
$CPs_{95\%}$	1.000	1.000	1.000	0.945	0.951	0.956
$AL_{95\%}$	10.09664	8.87818	8.36546	7.18907	6.35972	5.82456

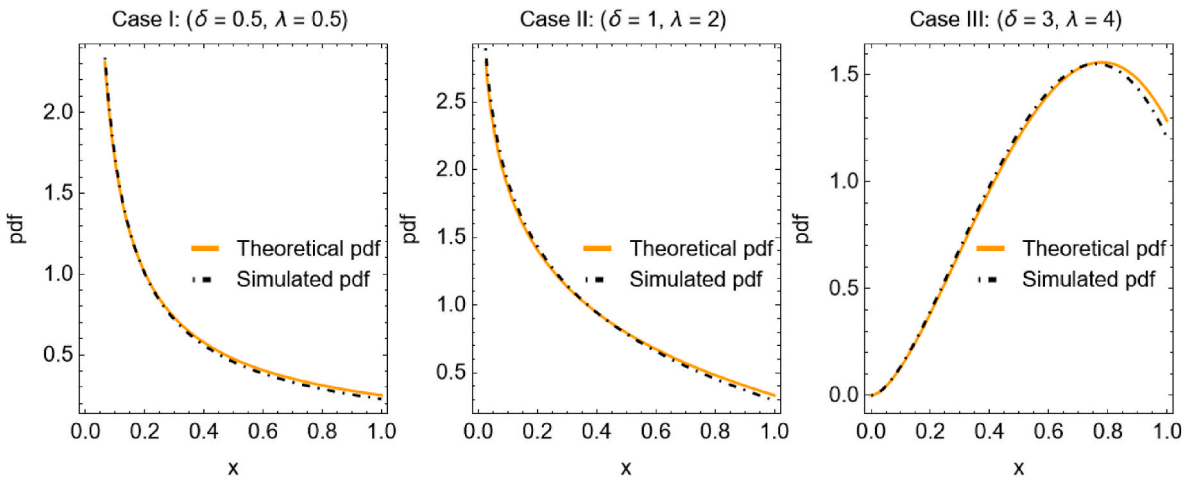


Fig. 7. Plots of theoretical and simulated PDFs of UExE distribution.

- The average length of the simulated confidence intervals decreases with increasing *n*.

Fig. 7: depicts the theoretical and simulated PDFs of the UExE distribution for the selected parameters. The reliability of estimators is determined by their biases and mean squared error (MSE). In conclusion, the findings demonstrate that MLE is an effective tool for accurate forecasting.

6. Applications

This section demonstrates the versatility of the UExE model using practical datasets, comparing it to several competing models. The UExE model offers an alternative distribution for fitting unit interval data, surpassing earlier models in the literature. The proposed model's fit was compared to nine competing two-parameter models listed in Table 6.

Two real data sets are analyzed by fitting the UExE and competitor models, using the following metrics: log likelihood function (Log-L),

Table 6
Probability density function of the competitor models.

Model	Abbreviation	Probability Density Function	Ref.
Beta Distribution	Beta (κ,γ)	$g(x) = \frac{(1-x)^{\kappa-1}x^{\gamma-1}}{\text{Beta}(\kappa,\gamma)}$; $0<x<1,\kappa,\gamma>0$	Johnson et al. (1955)
Kumaraswamy Distribution	Kum (a,b)	$g(x) = a\,b\,x^{a-1}(1-x^a)^{b-1}$; $0<x<1,a,b>0$	Kumaraswamy (1980)
Unit Gamma Distribution	UG (θ,ρ)	$g(x) = \frac{\theta x^{\theta-1}(-\theta\log(x))^{\rho-1}}{\Gamma(\rho)}$; $0<x<1,\theta,\rho>0$	Grassia (1977)
Power Topp-Leone Distribution	PTL (ν,τ)	$g(x) = 2\tau\nu x^{\tau\nu-1}(1-x^{\tau})(2-x^{\tau})^{\nu-1}$; $0<x<1,\nu,\tau>0$	Elgarhy et al. (2022)
Bounded Weighted Exponential Distribution	BWE (σ,η)	$g(x) = \frac{(\sigma+1)\,\eta\,x^{\eta-1}(1-x^{\sigma\,\eta})}{\sigma}$; $0<x<1,\sigma,\eta>0$	Mallick et al. (2020)
Unit Burr XII Distribution	UBXII (μ,ϕ)	$g(x) = \frac{\mu\,\phi\left[1+(-\log(x))^{\phi}\right]^{-1-\mu}}{[-\log(x)]^{-\phi+1}x}$; $0<x<1,\mu,\phi>0$	Zayed et al. (2023)
Unit Birnbaum-Saunders Distribution	UBS(α,β)	$g(x) = \frac{\left(\sqrt{-\frac{\beta}{\log(x)}}+\left(-\frac{\beta}{\log(x)}\right)^{3/2}\right)\exp\left(\frac{\frac{\beta}{\log(x)}+\frac{\log(x)}{\beta}+2}{2\alpha^2}\right)}{2\sqrt{2\pi}\alpha\beta x}$; $0<x<1,\alpha,\beta>0$	Mazucheli et al. (2018)
Unit Inverse Gaussian Distribution	UIG (ζ,ω)	$g(x) = \frac{\zeta\,\exp\left[\frac{\zeta(\log(x)+\omega)^2}{2\omega^2\log(x)}\right]}{\sqrt{2\pi}\,x\log^2(x)\sqrt{-\frac{\zeta}{\log(x)}}}$; $0<x<1,\zeta,\omega>0$	Ghitany et al. (2019)
Unit Mirra Distribution	UM (ψ,Θ)	$g(x) = \frac{\psi^3(\Theta+(\Theta+2)x^2-2\,\Theta\,x)e^{\psi-\frac{\psi}{x}}}{2x^4(\Theta+\psi^2)}$; $0<x<1,\psi,\Theta>0$	Al-Omari et al. (2024)

Table 7
MLE estimates and their SEs are in parentheses for the first data set.

Model	MLE estimates	
UExE (λ,δ)	$\widehat{\lambda}=1.310340$ (0.237419)	$\widehat{\delta}=5.452480$ (8.621930)
Beta (κ,γ)	$\widehat{\kappa}=1.620460$ (0.410655)	$\widehat{\gamma}=0.966648$ (0.223783)
Kum (a,b)	$\widehat{a}=1.608370$ (0.413672)	$\widehat{b}=0.962711$ (0.201704)
UG (θ,ρ)	$\widehat{\theta}=1.588780$ (0.375044)	$\rho=1.152580$ (0.319212)
PTL (ν,τ)	$\widehat{\nu}=1.895020$ (2.382460)	$\widehat{\tau}=0.489427$ (0.756056)
BWE (σ,η)	$\widehat{\sigma}=2.224500$ (6.588940)	$\widehat{\eta}=0.950430$ (0.487940)
UBXII (μ,ϕ)	$\widehat{\mu}=1.033100$ (0.205956)	$\widehat{\phi}=1.846460$ (0.305385)
UBS(α,β)	$\widehat{\alpha}=1.077200$ (0.139246)	$\widehat{\beta}=0.848295$ (0.144393)
UIG (ζ,ω)	$\widehat{\zeta}=0.922059$ (0.238075)	$\widehat{\omega}=1.378460$ (0.307716)
UM (ψ,Θ)	$\widehat{\psi}=0.005133$ (0.006992)	$\widehat{\Theta}=0.203977$ (0.044707)

Table 8
Values of Log-L, AIC, BIC, and HQIC statistics for the first data set.

Model	Log-L	AIC	CAIC	HQIC
UExE (λ,δ)	3.46914	−2.93828	−2.49383	−2.04177
Beta (κ,γ)	3.30506	−2.61013	−2.16568	−1.71362
Kum (a,b)	3.31103	−2.62207	−2.17762	−1.72556
UG (θ,ρ)	3.42444	−2.84888	−2.40443	−1.95237
PTL (ν,τ)	3.04885	−2.09770	−1.65326	−1.20119
BWE (σ,η)	2.95738	−1.91477	−1.47032	−1.01826
UBXII (μ,ϕ)	1.03899	1.92203	2.36647	2.81854
UBS(α,β)	0.50503	2.98995	3.43439	3.88646
UIG (ζ,ω)	−1.27831	6.55662	7.00107	7.45313
UM (ψ,Θ)	−0.06582	4.13164	4.57609	5.02815

Table 9
Values of A*, W*, K-S and *p*-value of K-S statistic for the first data.

Model	A*	W*	K-S	*p*-value (K-S)
UExE (λ,δ)	0.123067	0.015976	0.057138	0.999973
Beta (κ,γ)	0.174971	0.022450	0.066919	0.999297
Kum (a,b)	0.167786	0.021062	0.064987	0.999584
UG (θ,ρ)	0.157212	0.019329	0.063050	0.999767
PTL (ν,τ)	0.229973	0.026585	0.071811	0.997807
BWE (σ,η)	0.257201	0.027838	0.073296	0.997042
UBXII (μ,ϕ)	0.513601	0.077203	0.099253	0.929069
UBS(α,β)	1.141920	0.215563	0.161331	0.415699
UIG (ζ,ω)	1.516040	0.281479	0.174979	0.317283
UM (ψ,Θ)	2.375900	0.419638	0.229565	0.084676

Table 10
MLE estimates and their SEs are in parentheses for the second data set.

Model	MLE estimates	
UExE (λ,δ)	$\widehat{\lambda}=0.944177$ (1.33634)	$\widehat{\delta}=0.779734$ (0.196414)
Beta (κ,γ)	$\widehat{\kappa}=1.219780$ (0.375781)	$\widehat{\gamma}=0.553954$ (0.142337)
Kum (a,b)	$\widehat{a}=1.230550$ (0.348306)	$\widehat{b}=0.571800$ (0.147779)
UG (θ,ρ)	$\widehat{\theta}=1.249890$ (0.338200)	$\widehat{\rho}=0.629702$ (0.208451)
PTL (ν,τ)	$\widehat{\nu}=11.58090$ (17.2772)	$\widehat{\tau}=0.044832$ (0.069510)
BWE (σ,η)	$\widehat{\sigma}=30.40410$ (38.2133)	$\widehat{\eta}=0.519846$ (0.112434)
UBXII (μ,ϕ)	$\widehat{\mu}=0.837525$ (0.205620)	$\widehat{\phi}=1.521770$ (0.299432)
UBS(α,β)	$\widehat{\alpha}=1.241260$ (0.187506)	$\widehat{\beta}=1.079840$ (0.235726)
UIG (ζ,ω)	$\widehat{\zeta}=0.932061$ (0.281027)	$\widehat{\omega}=1.984900$ (0.617553)
UM (ψ,Θ)	$\widehat{\psi}=0.001042$ (0.001449)	$\widehat{\Theta}=0.075106$ (0.019052)

Table 11
Values of Log-L, AIC, CAIC and HQIC statistics for the second data.

Model	Log-L	AIC	CAIC	HQIC
UExE (λ,δ)	7.14843	−10.2969	−9.66529	−9.78283
Beta (κ,γ)	6.78192	−9.56385	−8.93227	−9.04982
Kum (a,b)	6.84362	−9.68723	−9.05566	−9.17320
UG (θ,ρ)	6.90520	−9.8104	−9.17882	−9.29637
PTL (ν,τ)	6.95950	−9.91899	−9.28741	−9.40496
BWE (σ,η)	7.02717	−10.0543	−9.42276	−9.54030
UBXII (μ,ϕ)	3.43313	−2.86627	−2.23469	−2.35223
UBS(α,β)	5.76512	−7.53023	−6.89865	−7.01620
UIG (ζ,ω)	3.92454	−3.84909	−3.21751	−3.33505
UM (ψ,Θ)	3.25043	−2.50087	−1.86929	−1.98683

Table 12
Values of A*, W*, K-S and *p*-value of K-S statistics for the second data.

Model	A*	W*	K-S	*p*-value (K-S)
UExE (λ,δ)	0.649696	0.108743	0.184086	0.445131
Beta (κ,γ)	0.752175	0.129277	0.200174	0.341309
Kum (a,b)	0.745117	0.125826	0.196269	0.364947
UG (θ,ρ)	1.021000	0.188215	0.231408	0.189402
PTL (ν,τ)	0.750913	0.132587	0.205338	0.311644
BWE (σ,η)	0.755023	0.133081	0.205379	0.311417
UBXII (μ,ϕ)	1.211900	0.218930	0.243403	0.147483
UBS(α,β)	1.364610	0.262513	0.257948	0.107031
UIG (ζ,ω)	1.851910	0.347420	0.283360	0.058440
UM (ψ,Θ)	4.276560	0.449783	0.291617	0.047423

Akaike information criterion (AIC), consistent AIC (CAIC) and Hannon-Quinn's information criterion (HQIC), Anderson-Darling (A*) and Cramer-Von Mises (W*). Additionally, the Kolmogorov–Smirnov (K-S) test is employed, with the p-value of the K-S statistic used to compare models; this statistic assesses the fit between the empirical and fitted distribution functions, making it a popular choice for evaluating how well a random sample's distribution aligns with a theoretical distribution. Generally, the model with the lowest scores of these metrics and the highest p-value for K-S statistic is considered the most effective.

The first dataset includes 30 tensile strength observations of polyester fibers obtained by (Quesenberry & Hales, 1980). The data are described as: "0.023, 0.032, 0.054, 0.069, 0.081, 0.094, 0.105, 0.127, 0.148, 0.169, 0.188, 0.216, 0.255, 0.277, 0.311, 0.361, 0.376, 0.395, 0.432, 0.463, 0.481, 0.519, 0.529, 0.567, 0.642, 0.674, 0.752, 0.823, 0.887, 0.926". The data results are included in Tables (7–9), as well as Figs. (8), (9) and (12).

The second data set contains 22 records of the computing time of P3 algorithms, as reported by (Caramanis et al., 1983). The data are listed as: "0.853, 0.759, 0.874, 0.800, 0.716, 0.557, 0.503, 0.399, 0.334, 0.207, 0.118, 0.097, 0.078, 0.067, 0.056, 0.044, 0.036, 0.026, 0.019, 0.014, 0.010, 0.118". The data results are presented in Tables (10–12), and Figures (10), (11) and (13).

Tables 7–12 summarize the estimated MLEs and goodness-of-fit information criteria for both data sets across competitive models. The mentioned measures indicate that the UExE distribution is a better competitor to other models, offering the best fit among them.

The following graphical illustration validates these findings:

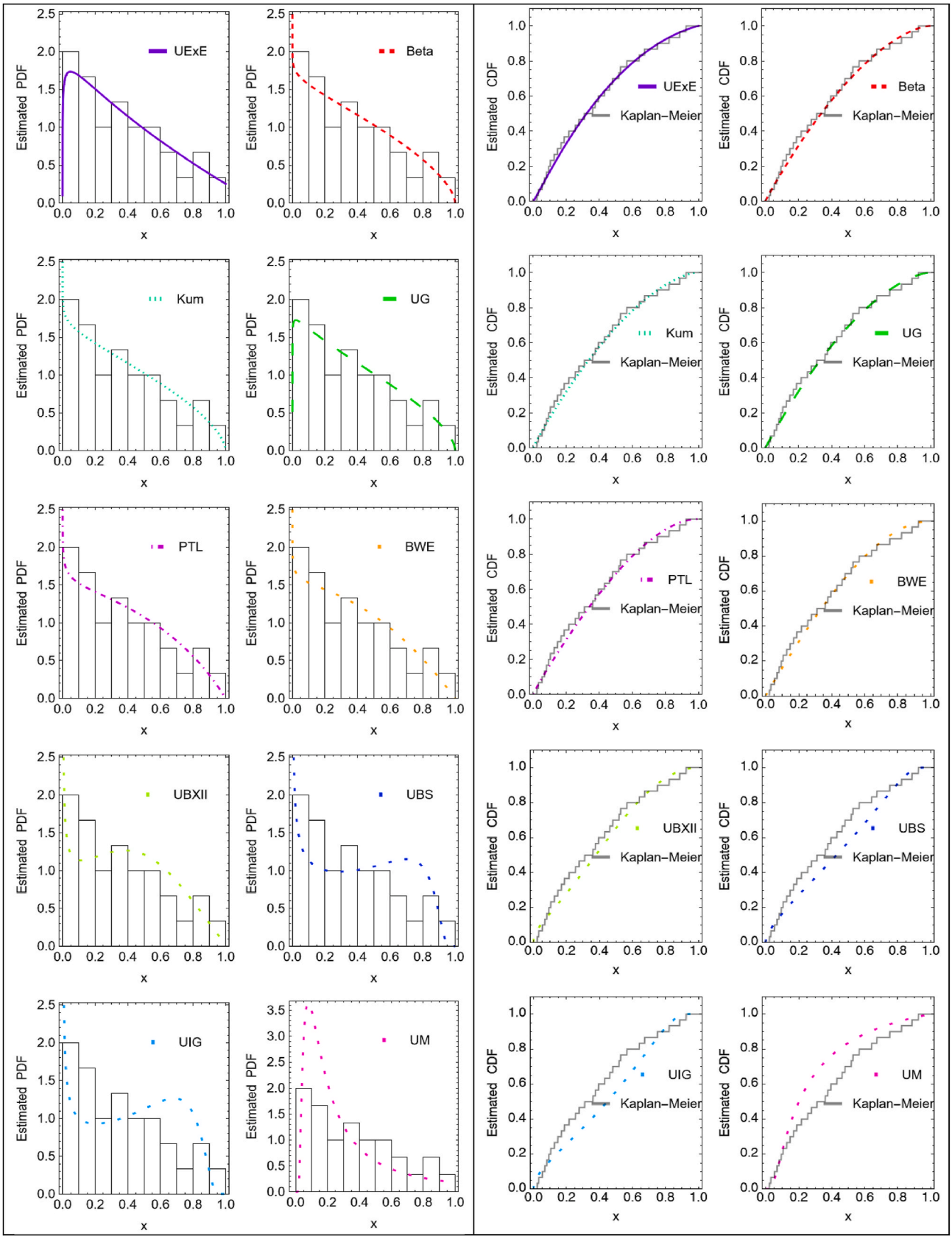


Fig. 8. Fitted PDFs and CDFs for the first data set.

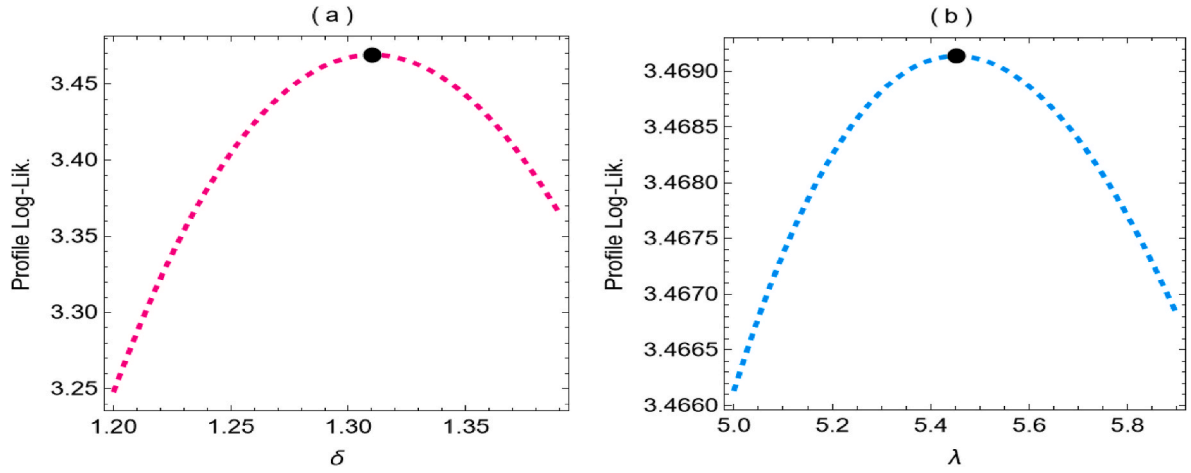


Fig. 9. Profile likelihood functions of parameters for the first data set.

- Figs. 8 and 10 display the histograms of the data sets with the fitted PDF, along with the empirical and estimated CDF plots, to assess the UExE model's suitability.
- Figs. 9 and 11 illustrate the profile likelihood functions of parameters of the proposed model for both datasets, demonstrating the uniqueness of the maximum likelihood estimates of δ and λ.
- Figs. 12 and 13 provide an additional insight by demonstrating the models' adequacy using Q-Q plot for both data sets, the scatter plot's perfect alignment with the Q-Q line indicates that the UExE distribution matches better than others.

The variance–covariance matrix of the MLEs of the UExE distribution for the first data set is:

$$I^{-1}(\widehat{\varpi}) = \begin{pmatrix} 0.05636771 & 1.42206543 \\ 1.42206543 & 74.33765931 \end{pmatrix}_{2\times 2}$$

The approximate 95 % CIs of the parameters δ and λ are: $[0.844997, 1.77568]$ and $(0, 22.3515]$, respectively.

The variance–covariance matrix of the MLEs of the UExE distribution for the second data set is:

$$I^{-1}(\widehat{\varpi}) = \begin{pmatrix} 0.03857842 & 0.20348369 \\ 0.20348369 & 1.78579653 \end{pmatrix}_{2\times 2}$$

The approximate 95 % CIs of the parameters δ and λ are: $[0.39476, 1.16471]$ and $(0, 3.56339]$, respectively.

7. Conclusion and summary

This study introduces the unit extended exponential distribution. The extended exponential distribution is a versatile lifetime distribution with two parameters. Our objective is to extend this flexibility for adjustment from the elongated exponential distribution to the unit interval. Various probability density functions have been shown, such as declining, left-skewed, right-skewed, and unimodal. Nevertheless, the hazard rate function may exhibit several shapes, such as J-shaped, U-shaped, and bathtub shapes. The theoretical part covers various statistical properties such as moments, mean, variance, skewness, kurtosis, moment-generating functions, probability-weighted moments, incomplete moments, conditional moments, mean deviation, Lorenz and

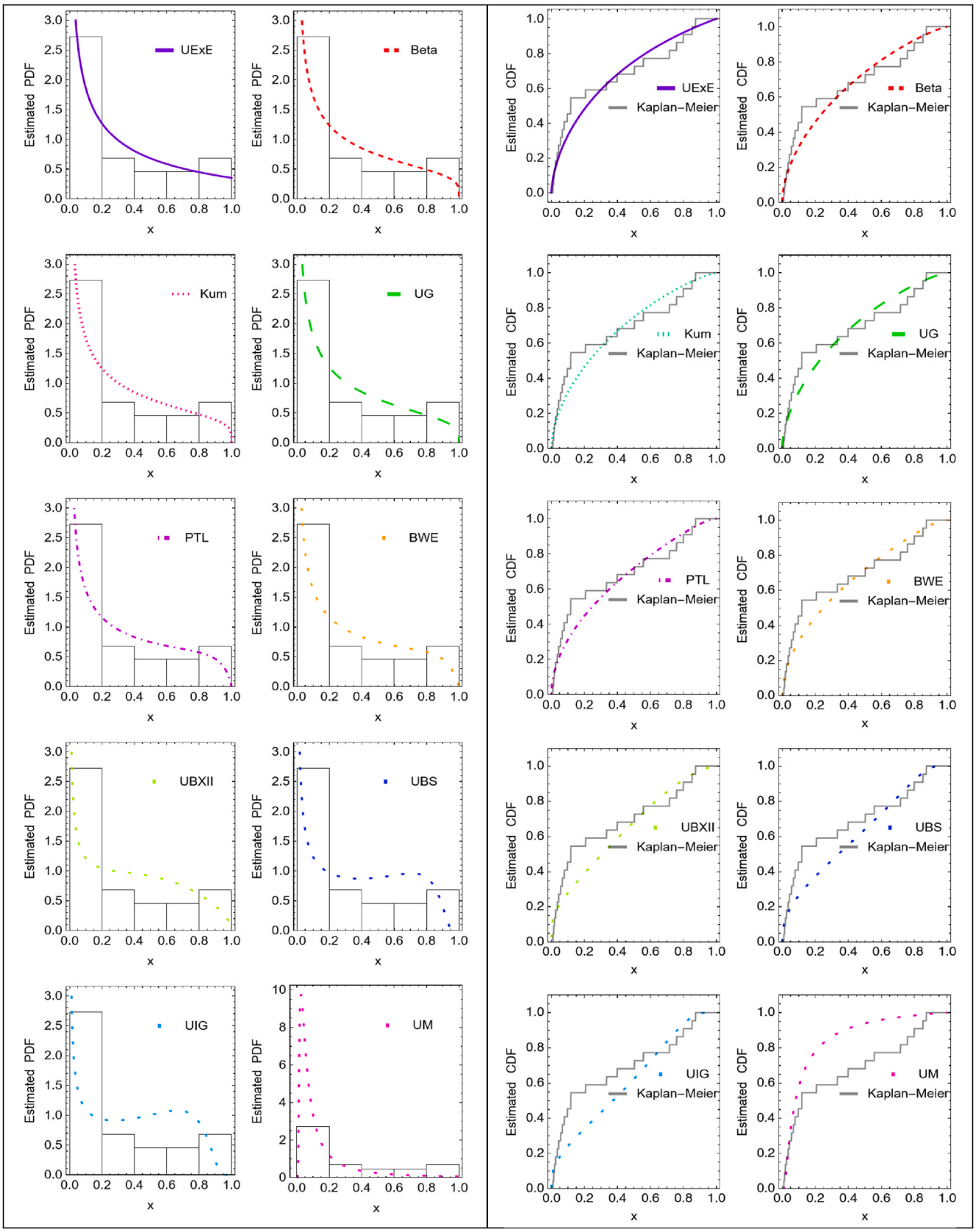


Fig. 10. Fitted PDFs and CDFs of all fitted models for the second data set.

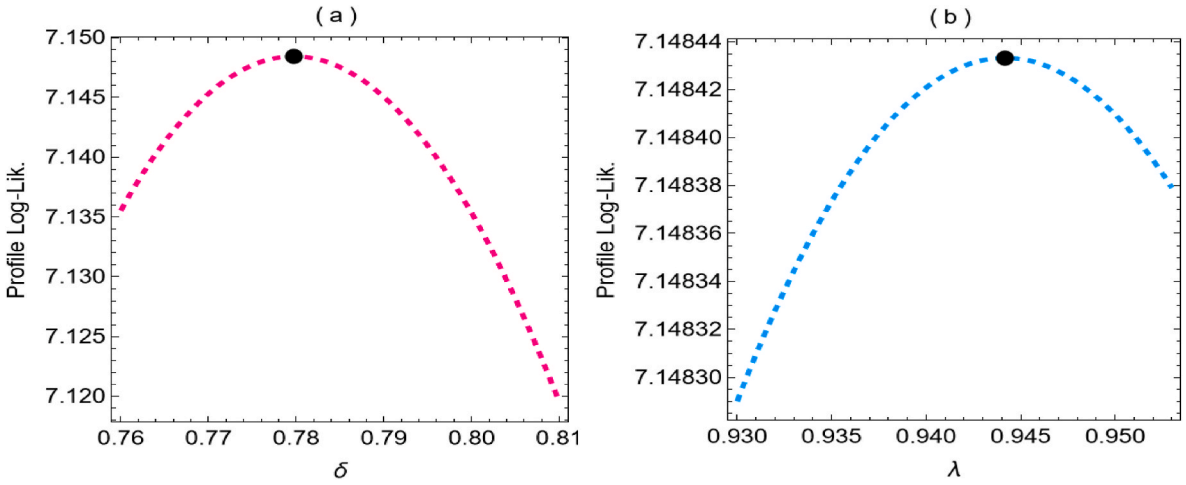


Fig. 11. Profile likelihood functions of parameters for the second data.

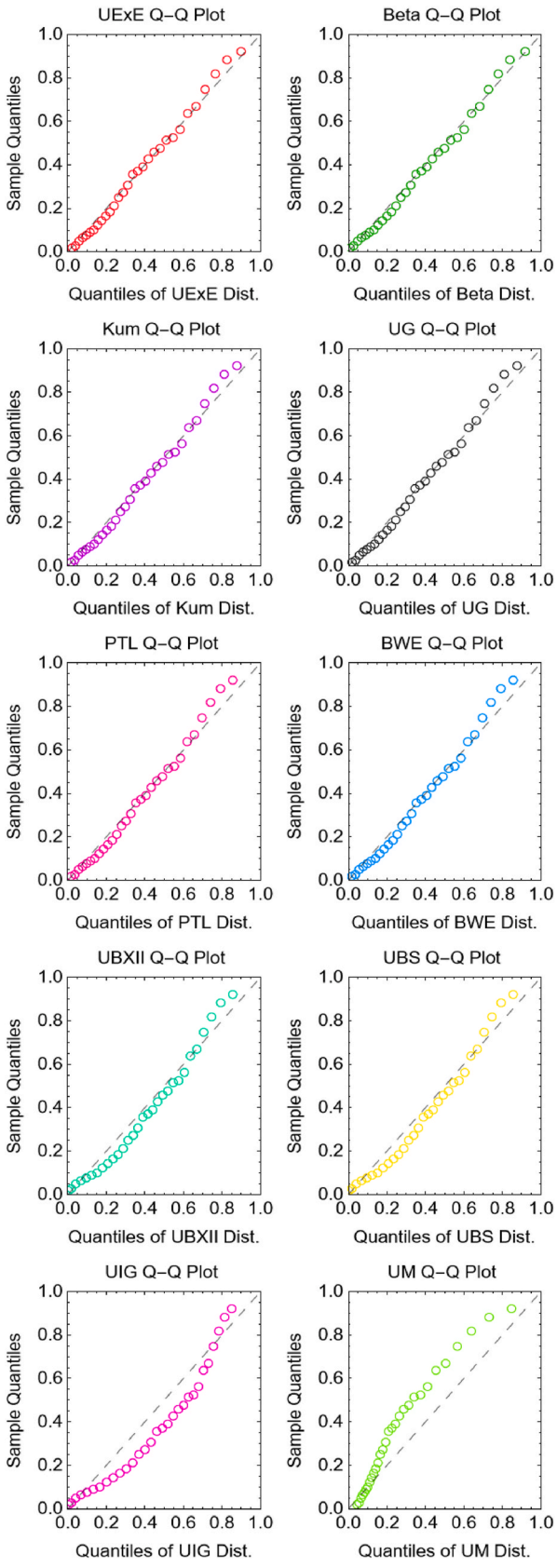


Fig. 12. The Q-Q plots of the first data set.

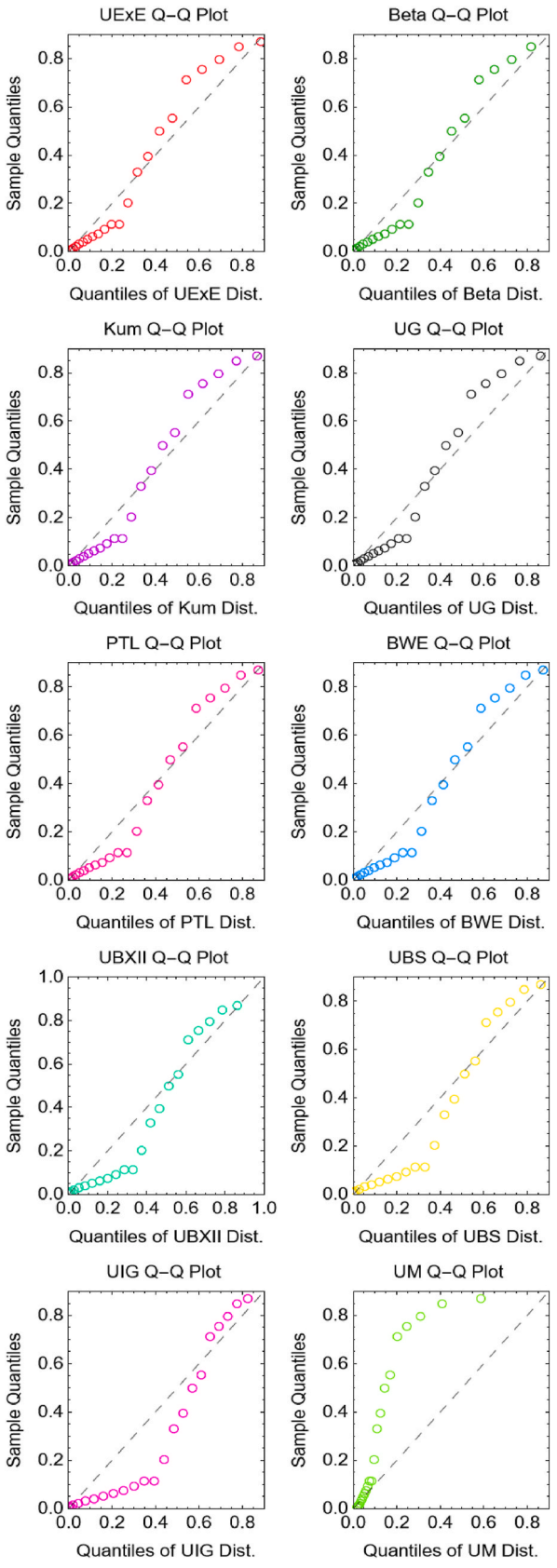


Fig. 13. The Q-Q plots of the second data set.

Bonferroni curves, mean residual life, mean inactivity time, and different measures of entropy. The maximum likelihood method is used to estimate model parameters using individual data. Simulated data is provided to facilitate the evaluation of this method. Two examples that use real data sets highlight the importance of the new model in comparison to conventional unit models. The limitation of this paper is that the simulation part is complete data that is only used to estimate the unknown parameters for the suggested model. This reason opens the door for researchers to work in the future to estimate the parameters of the proposed distribution using different censored schemes.

Funding statement

This research is supported by researchers Supporting Project number (RSPD2024R548), King Saud University, Riyadh, Saudi Arabia.

Data availability

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

CRediT authorship contribution statement

Ibrahim E. Ragab: Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Conceptualization. **Najwan Alsadat:** Writing – review & editing, Writing – original draft, Formal analysis, Conceptualization. **Oluwafemi Samson Balogun:** Writing – review & editing, Writing – original draft, Data curation, Conceptualization. **Mohammed Elgarhy:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no conflicts of interest to report regarding the present study.

Acknowledgement

This research is supported by researchers Supporting Project number (RSPD2024R548), King Saud University, Riyadh, Saudi Arabia.

References

Al-Omari, A. I., Alanzi, A. R., & Alshqaq, S. S. (2024). The unit two parameters mirra distribution: Reliability analysis, properties, estimation and applications. *Alexandria Engineering Journal, 92*, 238–253.

Alzaatreh, A., Famoye, F., & Lee, C. (2013). A new method for generating families of continuous distributions. *Metron, 71*, 63–79.

Arimoto, S. (1971). Information-theoretical considerations on estimation problems. *Information and Control, 19*(3), 181–194.

Caramanis, M., Stremel, J., Fleck, W., & Daniel, S. (1983). Probabilistic production costing: An investigation of alternative algorithms. *International Journal of Electrical Power & Energy Systems, 5*(2), 75–86.

Elgarhy, M., Soliman, A., & Heba, N. A. G. Y. (2022). Parameter estimation methods and applications of the power Topp-Leone distribution. *Gazi University Journal of Science, 35*(2), 731–746.

Ghitany, M., Mazucheli, J., Menezes, A., & Alqallaf, F. (2018). The unit-inverse Gaussian distribution: A new alternative to two-parameter distributions on the unit interval. *Communications in Statistics - Theory and Methods, 48*, 3423–3438.

Ghitany, M. E., Mazucheli, J., Menezes, A. F. B., & Alqallaf, F. (2019). The unit-inverse Gaussian distribution: A new alternative to two-parameter distributions on the unit interval. *Communications in Statistics - Theory and Methods, 48*(14), 3423–3438.

Gómez, Y. M., Bolfarine, H., & Gómez, H. W. (2014). A new extension of the exponential distribution. *Revista Colombiana de Estadística, 37*(1), 25–34.

Grassia, A. (1977). On a family of distributions with argument between 0 and 1 obtained by transformation of the gamma and derived compound distributions. *Australian Journal of Statistics, 19*(2), 108–114.

Gupta, R. D., & Kundu, D. (1999). Generalized exponential distributions. *Australian & New Zealand Journal of Statistics, 41*(2), 173–188.

Havrda, J., & Charvát, F. (1967). Quantification method of classification processes. Concept of structural $ a $-entropy. *Kybernetika, 3*(1), 30–35.

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1955). Continuous univariate distributions. In*Chapter 25: Beta distributions* (2nd ed., Vol. 2). Hoboken, NJ, USA: Wiley.

Jones, M. (2004). Families of distributions arising from the distributions of order statistics. *Test, 13*, 1–43.

Korkmaz, M. (2019). A new heavy-tailed distribution defined on the bounded interval: The logit slash distribution and its applications. *Journal of Applied Statistics, 473*, 2097–2119.

Korkmaz, M., & Chesneau, C. (2021). On the unit Burr-XII distribution with the quantile regression modeling and applications. *Computational and Applied Mathematics, 40*, 29.

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology, 46*(1–2), 79–88.

Lindley, D. V. (1958). Fiducial distributions and Bayes' theorem. *Journal of the Royal Statistical Society: Series B, 20*, 102–107.

Mallick, A., Ghosh, I., Dey, S., & Kumar, D. (2020). Bounded weighted exponential distribution with applications. *American Journal of Mathematical and Management Sciences, 40*(1), 68–87.

Mazucheli, J., Menezes, A., & Chakraborty, S. (2019). On the one parameter unit-Lindley distribution and its associated regression model for proportion data. *Journal of Applied Statistics, 46*, 700–714.

Mazucheli, J., Menezes, A. F., & Dey, S. (2018). The unit-Birnbaum-Saunders distribution with applications. *Chilean Journal of Statistics, 9*(1), 47–57.

Mazucheli, J., Menezes, A., & Dey, S. (2019). Unit-Gompertz distribution with applications. *Statistica, 79*, 26–43.

Modi, K., & Gill, V. (2019). Unit Burr III distribution with application. *Journal of Statistics & Management Systems, 23*, 579–592.

Nadarajah, S., & Haghighi, F. (2011). An extension of the exponential distribution. *Statistics: A Journal of Theoretical and Applied Statistics, 45*(6), 543–558.

Progri, I. F. (2020). The computation of a 2F2 hypergeometric function. *J. Geol. Geoinfo. Geointel, 2020*.

Quesenberry, C. P., & Hales, C. (1980). Concentration bands for uniformity plots. *Journal of Statistical Computation and Simulation, 11*(1), 41–53.

Renner, R., & Wolf, S. (2004). Smooth Rényi entropy and applications. In *International symposium on information theory, 2004. ISIT 2004. Proceedings* (p. 233). IEEE.

Shafiq, A., Sindhu, T. N., Dey, S., Lone, S. A., & Abushal, T. A. (2023). Statistical features and estimation methods for half-logistic unit-gompertz type-I model. *Mathematics, 11*(4), 1007.

Shafiq, A., Sindhu, T. N., Hussain, Z., Mazucheli, J., & Alves, B. (2023). A flexible probability model for proportion data: Unit Gumbel type-II distribution, development, properties, different method of estimations and applications. *Austrian Journal of Statistics, 52*(2), 116–140.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal, 27*(3), 379–423.

Sindhu, T. N., Shafiq, A., & Huassian, Z. (2023). Generalized exponentiated unit Gompertz distribution for modeling arthritic pain relief times data: Classical approach to statistical inference. *Journal of Biopharmaceutical Statistics, 34*(3), 323–348.

Singh, D. P., Jha, M., Tripathi, Y., & Wang, L. (2022). Reliability estimation in a multicomponent stress-strength model for unit Burr III distribution under progressive censoring. *Quality Technology & Quantitative Management, 19*, 605–632.

Tsallis, C. (2019). Beyond Boltzmann–Gibbs–Shannon in physics and elsewhere. *Entropy, 21*(7), 696.

Zayed, M. A., Hassan, A. S., Almetwally, E. M., Aboalkhair, A. M., Al-Nefaie, A. H., & Almongy, H. M. (2023). A compound class of unit Burr XII model: Theory, estimation, fuzzy, and application. *Scientific Programming, 2023*(1), Article 4509889.

附录 2　额外数据说明和附件说明

泰迪杯额外数据形式：

M101.csv 和 M102.csv 分别是两条生产线生产状态的原始数据，数据的形式见数据字段说明.xlsx

泰迪杯数据.xlsx 是我们对原始数据进行预处理后的数据。

以上文件均位于附件数据目录下。

附录 3　代码

Listing 3.1　绘制函数图像 python 代码

```
import matplotlib.pyplot as plt
import numpy as np
import math
def G_fun(delta, lam, x):
    x = np.asarray(x)
    return (1 - delta*lam*np.log(x)/(delta+lam+1e-7))*x**delta
def g_fun(delta, lam, x):
    x = np.asarray(x)
    return (delta**2*(1-lam*np.log(x)))/(delta+lam)*x**(delta-1)
def S_fun(delta, lam, x):
    x = np.asarray(x)
    return 1 - ((1 - delta*lam*np.log(x)/(delta+lam+1e-7))*x**delta)
def r_fun(delta, lam ,x):
    x = np.asarray(x)
    return (delta**2*(1-lam*np.log(x))) / ((delta+lam)*(x**(1-delta) - x)
        + delta*lam*x*np.log(x) +1e-7)

# 设置 delta 和 lam 的不同值组合
delta_lam_combinations = [(0.3, 0.5), (0.5, 0.5), (0.7, 0.5), (0.8, 0.5),
    (0.9, 0.5)]
# 生成 x 在 (0, 1) 范围内的值
x_values = np.linspace(0.01, 1, 100)  # 使用 0.01 避免 log(0) 问题
# 绘图
plt.figure(figsize=(10, 10))
# 绘制每个 delta 和 lam 组合的 S_fun(x)
for delta, lam in delta_lam_combinations:
    S_values = S_fun(delta, lam, x_values)
    # 为每个组合选择一个颜色，使用不同颜色标识
```

```
        plt.plot(x_values, S_values, label=f'$\delta={delta}$,␣$\lambda={lam}
            $', linestyle='--', linewidth=4)
# 图标题和标签
# plt.title(r'SF for the UExE distribution at λ = 0.5')
plt.xlabel('$x$')
plt.ylabel('$SF\\␣CDF$')
plt.grid(False)
plt.legend()  # 显示图例
plt.ylim(0, 1)
# 显示图形
plt.tight_layout()
plt.show()

# 设置 lam 值
lam = 0.5
# 定义 x 和 delta 的范围
x_values = np.linspace(0.01, 1, 100)  # 避免 log(0) 问题
delta_values = np.linspace(0, 1, 100)
# 创建网格
X, Delta = np.meshgrid(x_values, delta_values)
# 计算 S_fun 的值
Z = S_fun(Delta, lam, X)
# 创建 3D 图
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
# 绘制曲面图
surf = ax.plot_surface(X, Delta, Z, cmap='viridis', edgecolor='none',
    alpha=0.8)
# 添加颜色条
fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10, label="$SF$")
# 设置轴标签
ax.set_xlabel('$x$')
```

```
ax. set_ylabel ('$\delta$')
ax. set_zlabel ('$SF$')
# 设置标题
# ax. set_title (r'3D Surface Plot of $S(x, \delta)$ for $\lambda = 0.5$')
# 显示图形
plt.show()

# 设置 delta 和 lam 的不同值组合
delta_lam_combinations = [(0.3, 0.5), (0.5, 0.5), (0.7, 0.5), (0.8, 0.5),
    (0.9, 0.5)]
# 生成 x 在 (0, 1) 范围内的值
x_values = np.linspace(0.01, 1, 100)  # 避免 log(0) 问题
# 设置 lam 值
lam = 0.5
# 定义 delta 的范围
delta_values = np.linspace(0, 1, 100)
# 创建网格
X, Delta = np.meshgrid(x_values, delta_values)
Z = S_fun(Delta, lam, X)
# 创建合并图形
fig = plt.figure(figsize=(16, 8))
# 子图 1: 2D 图
ax1 = fig.add_subplot(121)
for delta, lam in delta_lam_combinations:
    S_values = S_fun(delta, lam, x_values)
    ax1.plot(x_values, S_values, label=f'$\delta={delta}$, $\lambda={lam}
        $', linestyle='--', linewidth=2)
ax1. set_xlabel ('$x$')
ax1. set_ylabel ('$SF$')
ax1.set_ylim(0, 1)
ax1.legend()
ax1.grid(False)
```

```
# ax1. set_title ('2D SF for different $\delta$ values')
# 子图 2: 3D 图
ax2 = fig.add_subplot(122, projection='3d')
surf = ax2.plot_surface(X, Delta, Z, cmap='viridis', edgecolor='none',
    alpha=0.8)
fig.colorbar(surf, ax=ax2, shrink=0.5, aspect=10, label="$SF$")
ax2.set_xlabel('$x$')
ax2.set_ylabel('$delta$')
ax2.set_zlabel('$SF$')
# ax2. set_title ('3D Surface Plot for $S(x, \delta)$ with $\lambda = 0.5$')
ax2.set_box_aspect([4, 4, 4])
# 调整布局并显示
plt.tight_layout()
plt.show()

# 设置 delta 和 lam 的不同值组合
delta_lam_combinations = [(0.3, 0.5), (0.5, 0.5), (0.7, 0.5), (0.8, 0.5),
    (0.9, 0.5)]
# 生成 x 在 (0, 1) 范围内的值
x_values = np.linspace(0.01, 1, 100) # 避免 log(0) 问题
# 设置 lam 值
lam = 0.5
# 定义 delta 的范围
delta_values = np.linspace(0, 1, 100)
# 创建网格
X, Delta = np.meshgrid(x_values, delta_values)
Z = G_fun(Delta, lam, X)
# 创建合并图形
fig = plt.figure(figsize=(16, 8))
# 子图 1: 2D 图
ax1 = fig.add_subplot(121)
for delta, lam in delta_lam_combinations:
```

```
        S_values = G_fun(delta, lam, x_values)
        ax1.plot(x_values, S_values, label=f'$\delta={delta}$,␣$\lambda={lam}
            $', linestyle='--', linewidth=2)
ax1.set_xlabel('$x$')
ax1.set_ylabel('$UExE$')
ax1.set_ylim(0, 1)
ax1.legend()
ax1.grid(False)
# ax1.set_title('2D UExE for different $\delta$ values')
# 子图 2: 3D 图
ax2 = fig.add_subplot(122, projection='3d')
surf = ax2.plot_surface(X, Delta, Z, cmap='viridis', edgecolor='none',
    alpha=0.8)
fig.colorbar(surf, ax=ax2, shrink=0.5, aspect=10, label="$UExE$")
ax2.set_xlabel('$x$')
ax2.set_ylabel('$delta$')
ax2.set_zlabel('$UExE$')
# ax2.set_title('3D Surface Plot for $S(x, \delta)$ with $\lambda = 0.5$')
ax2.set_box_aspect([4, 4, 4])
ax2.view_init(elev=30, azim=240)
# 调整布局并显示
plt.tight_layout(pad= 2)
plt.show()

# 设置 delta 和 lam 的不同值组合
delta_lam_combinations = [(0.5, 0.5), (0.8, 0.8), (1.5, 8), (2, 10), (3,
    15), (4, 20)]
# 生成 x 在 (0, 1) 范围内的值
x_values = np.linspace(0.01, 1, 1000) # 使用 0.01 避免 log(0) 问题
# 创建一个 1x2 的子图，左边绘制 g_fun，右边绘制 r_fun
fig, axs = plt.subplots(1, 2, figsize=(18, 10))
# 绘制每个 delta 和 lam 组合的 g_fun(x)（左图）
```

```
for delta, lam in delta_lam_combinations:
    S_values = g_fun(delta, lam, x_values)
    axs[0].plot(x_values, S_values, label=f'$\delta={delta}$, $\lambda={
        lam}$', linestyle='--', linewidth=4)
# 左图的标题和标签
axs[0].set_title(r'$\mathbf{g(x)}$ vs $x$ for different $\delta$ and $\
    lambda$ combinations')
axs[0].set_xlabel('$x$')
axs[0].set_ylabel('$g(x)$')
axs[0].grid(False)
axs[0].legend()
# 绘制每个 delta 和 lam 组合的 r_fun(x)（右图）
for delta, lam in delta_lam_combinations:
    r_values = r_fun(delta, lam, x_values)
    axs[1].plot(x_values, r_values, label=f'$\delta={delta}$, $\lambda={
        lam}$', linestyle='--', linewidth=4)
# 右图的标题和标签
axs[1].set_title(r'$\mathbf{r(x)}$ vs $x$ for different $\delta$ and $\
    lambda$ combinations')
axs[1].set_xlabel('$x$')
axs[1].set_ylabel('$r(x)$')
axs[1].grid(False)
axs[1].legend()
# 设置 y 轴范围为 (0, 2)
axs[0].set_ylim(0, 2)
axs[1].set_ylim(0, 7)
axs[1].set_xlim(0, 0.78)
# 调整布局，避免标签重叠
plt.tight_layout()
# 显示图形
plt.show()
```

Listing 3.2　矩估计 Python 代码

```
import numpy as np
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
def G_fun(delta, lam, x):
    x = np.asarray(x)
    return (1 - delta*lam*np.log(x)/(delta+lam+1e-7))*x**delta
def g_fun(delta, lam, x):
    x = np.asarray(x)
    return (delta**2*(1-lam*np.log(x)))/(delta+lam)*x**(delta-1)
def S_fun(delta, lam, x):
    x = np.asarray(x)
    return 1 - ((1 - delta*lam*np.log(x)/(delta+lam+1e-7))*x**delta)
def r_fun(delta, lam ,x):
    x = np.asarray(x)
    return (delta**2*(1-lam*np.log(x))) / ((delta+lam)*(x**(1-delta) - x)
        + delta*lam*x*np.log(x) +1e-7)
def get_mu(delta, lam, r):
    return delta**2*(r+delta+lam)/(delta+lam)/(r+delta)**2
def get_mean(delta, lam):
    return get_mu(delta=delta,lam=lam,r=1)
def get_var(delta, lam):
    mu1 = get_mu(delta=delta,lam=lam,r=1)
    mu2 = get_mu(delta=delta,lam=lam,r=2)
    return mu2 - mu1**2
def get_ske(delta, lam):
    mu1 = get_mu(delta=delta,lam=lam,r=1)
    mu2 = get_mu(delta=delta,lam=lam,r=2)
    mu3 = get_mu(delta=delta,lam=lam,r=3)
    sigma = np.sqrt(mu2 - mu1**2)
    return (mu3 - 3*mu1*mu2 + 2*mu1**3)/sigma**3
```

```
def get_kurt(delta, lam):
    mu1 = get_mu(delta=delta,lam=lam,r=1)
    mu2 = get_mu(delta=delta,lam=lam,r=2)
    mu3 = get_mu(delta=delta,lam=lam,r=3)
    mu4 = get_mu(delta=delta,lam=lam,r=4)
    sigma_2 = mu2 - mu1**2
    return (mu4-4*mu1*mu3 + 6*mu1**2*mu2-3*mu1**4)/sigma_2**2
def get_CV(delta, lam):
    mu1 = get_mu(delta=delta,lam=lam,r=1)
    mu2 = get_mu(delta=delta,lam=lam,r=2)
    sigma = np.sqrt(mu2 - mu1**2)
    return sigma/mu1
def get_ID(delta, lam):
    mu1 = get_mu(delta=delta,lam=lam,r=1)
    mu2 = get_mu(delta=delta,lam=lam,r=2)
    sigma = np.sqrt(mu2 - mu1**2)
    return mu1/sigma
# 生成 delta 和 lam 的网格
delta = np.linspace(0, 2, 100)  # delta 在 [0, 2] 范围内取值
lam = np.linspace(0, 2, 100)    # lam 在 [0, 2] 范围内取值
delta_grid, lam_grid = np.meshgrid(delta, lam)
# 计算 get_mean 的值
mean_values = get_mean(delta_grid, lam_grid)
# 创建三维图像
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
# 绘制曲面
surf = ax.plot_surface(delta_grid, lam_grid, mean_values, cmap='viridis',
    edgecolor='k')
# 添加轴标签
ax.set_xlabel('Delta')
ax.set_ylabel('Lambda')
```

```
ax. set_zlabel ('Mean')
ax. set_box_aspect ([4, 4, 4])
ax. view_init (elev=30, azim=240)
# 添加颜色条
fig . colorbar ( surf , ax=ax, shrink =0.5, aspect=10).remove()
# 显示图像
# plt . title ('3D Plot of get_mean(delta, lam)')
plt .show()
# 生成 delta 和 lam 的网格
delta = np. linspace (0, 2, 100) # delta 在 [0, 2] 范围内取值
lam = np. linspace (0, 2, 100) # lam 在 [0, 2] 范围内取值
delta_grid , lam_grid = np.meshgrid( delta , lam)
# 计算 get_mean 的值
mean_values = get_var ( delta_grid , lam_grid)
# 创建三维图像
fig = plt . figure ( figsize =(10, 10))
ax = fig .add_subplot(111, projection ='3d')
# 绘制曲面
surf = ax. plot_surface ( delta_grid , lam_grid, mean_values, cmap='viridis',
    edgecolor='k')
# 添加轴标签
ax. set_xlabel ('Delta')
ax. set_ylabel ('Lambda')
ax. set_zlabel ('Var')
ax. set_box_aspect ([4, 4, 4])
# ax. view_init ( elev=30, azim=240)
# 添加颜色条
fig . colorbar ( surf , ax=ax, shrink =0.5, aspect=10).remove()
# 显示图像
# plt . title ('3D Plot of get_mean(delta, lam)')
plt .show()
```

```
# 生成 delta 和 lam 的网格
delta = np.linspace(0, 2, 100)  # delta 在 [0, 2] 范围内取值
lam = np.linspace(0, 2, 100)    # lam 在 [0, 2] 范围内取值
delta_grid, lam_grid = np.meshgrid(delta, lam)
# 计算 get_mean 的值
mean_values = get_ske(delta_grid, lam_grid)
# 创建三维图像
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
# 绘制曲面
surf = ax.plot_surface(delta_grid, lam_grid, mean_values, cmap='viridis',
    edgecolor='k')
# 添加轴标签
ax.set_xlabel('Delta')
ax.set_ylabel('Lambda')
ax.set_zlabel('Skewness')
ax.set_box_aspect([4, 4, 4])
ax.view_init(elev=30, azim=60)
# 添加颜色条
fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10).remove()
# 显示图像
# plt.title('3D Plot of get_mean(delta, lam)')
plt.show()

# 生成 delta 和 lam 的网格
delta = np.linspace(0, 2, 100)  # delta 在 [0, 2] 范围内取值
lam = np.linspace(0, 2, 100)    # lam 在 [0, 2] 范围内取值
delta_grid, lam_grid = np.meshgrid(delta, lam)
# 计算 get_mean 的值
mean_values = get_kurt(delta_grid, lam_grid)
# 创建三维图像
fig = plt.figure(figsize=(10, 10))
```

```
ax = fig.add_subplot(111, projection='3d')
# 绘制曲面
surf = ax.plot_surface(delta_grid, lam_grid, mean_values, cmap='viridis',
    edgecolor='k')
# 添加轴标签
ax.set_xlabel('Delta')
ax.set_ylabel('Lambda')
ax.set_zlabel('Kurtosis')
ax.set_box_aspect([4, 4, 4])
ax.view_init(elev=30, azim=300)
# 添加颜色条
fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10).remove()
# 显示图像
# plt.title('3D Plot of get_mean(delta, lam)')
plt.show()

# 生成 delta 和 lam 的网格
delta = np.linspace(0, 2, 100)  # delta 在 [0, 2] 范围内取值
lam = np.linspace(0, 2, 100)    # lam 在 [0, 2] 范围内取值
delta_grid, lam_grid = np.meshgrid(delta, lam)
# 计算 get_mean 的值
mean_values = get_CV(delta_grid, lam_grid)
# 创建三维图像
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
# 绘制曲面
surf = ax.plot_surface(delta_grid, lam_grid, mean_values, cmap='viridis',
    edgecolor='k')
# 添加轴标签
ax.set_xlabel('Delta')
ax.set_ylabel('Lambda')
ax.set_zlabel('CV')
```

```
ax.set_box_aspect ([4, 4, 4])
# ax.view_init (elev=30, azim=0)
# 添加颜色条
fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10).remove()
# 显示图像
# plt.title('3D Plot of get_mean(delta, lam)')
plt.show()

# 生成 delta 和 lam 的网格
delta = np.linspace(0, 2, 100)  # delta 在 [0, 2] 范围内取值
lam = np.linspace(0, 2, 100)    # lam 在 [0, 2] 范围内取值
delta_grid, lam_grid = np.meshgrid(delta, lam)
# 计算 get_mean 的值
mean_values = get_ID(delta_grid, lam_grid)
# 创建三维图像
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
# 绘制曲面
surf = ax.plot_surface(delta_grid, lam_grid, mean_values, cmap='viridis',
    edgecolor='k')
# 添加轴标签
ax.set_xlabel('Delta')
ax.set_ylabel('Lambda')
ax.set_zlabel('ID')
ax.set_box_aspect ([4, 4, 4])
ax.view_init(elev=30, azim=240)
# 添加颜色条
fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10).remove()
# 显示图像
# plt.title('3D Plot of get_mean(delta, lam)')
plt.show()
```

Listing 3.3　基于二分的逆变换抽样法 python 代码

```
'''
@File: DataGenerator.py
Desc: class DataGenerator genetate data for simulation
'''
from config import parse_args
import math as m
import numpy as np
import random
class DataGenerator:
    def __init__(self, opt):
        self.seed = opt.seed
        self.fun_name = opt.func_name
        np.random.seed(self.seed)
        random.seed(self.seed)
    def __call__(self, K, n_list, delta, lambdaa):
        return self.forward(K, n_list, delta, lambdaa)
    def UExE_func(self, x, delta, lambdaa):
        return 1 - (1 - (delta * lambdaa * m.log(x)) / (delta + lambdaa))
            * x**delta
    def find_x(self, y, func_name, delta, lam, low=0, high=1, tol=1e-10):
        if func_name == 'UExE':
            while high - low > tol:
                mid = (low + high) / 2.0
                V_mid = self.UExE_func(mid, delta, lam)
                if abs(V_mid - y) < tol:
                    return mid
                elif V_mid > y:
                    low = mid
                else:
                    high = mid
            return (low + high) / 2.0
```

```
            else:
                return 0
        def inverse_func(self, y_ls, func_name, delta=0, lambdaa=1):
            x_ls = []
            for y in y_ls:
                inverse = self.find_x(y, func_name, delta=delta,lam=lambdaa)
                x_ls.append(inverse)
            return x_ls
        def forward(self, K, n_list, delta=0, lambdaa=1):# eg: K 1000 n_list
            :[100, 150, 200, 300, 400, 500]
            '''
            @Params:
            K: The number of the samples
            n_list: The number of each sample
            func_name: The function that we would like to sample
            delta, lambdaa: The parameters of the function
            @Returns:
            samples: dic{K:{n:[sample list ]}}:
            '''
            samples = {}
            for i in range(K):
                if i not in samples:
                    samples[i] = {}
                for n in n_list:
                    random_y = np.random.uniform(0, 1, n)
                    random_x = self.inverse_func(random_y, self.fun_name,
                        delta, lambdaa)
                    if n not in samples[i]:
                        samples[i][n] = random_x
            return samples
    if __name__ == '__main__':
        opt = parse_args()
```

```
datagenerator = DataGenerator(opt)
K = 2
n_list = [100, 150, 200, 300, 400, 500]
samples = datagenerator(K, n_list, delta=0.5, lambdaa=0.5)
print(samples[1][500])
```

Listing 3.4 极大似然估计实现 python 代码

```
'''
@File: MLE_Estimate.py
Desc: class Maximum Likelihood Estimation for delta and lambda
'''
import numpy as np
import math as m
class MLE_Estimate:
    def __init__(self, opt):
        self.delta_init = opt.delta_init
        self.lambda_init = opt.lambda_init
    def __call__(self, x, tol=1e-7, max_iter=1000):
        return self.forward(x, tol, max_iter)
    def gradient(self, delta, lambdaa, x):
        n = len(x)
        grad_delta = (2 * n / delta) - (n / (delta + lambdaa)) + np.sum(
            np.log(x))
        grad_lambd = -(n / (delta + lambdaa)) - np.sum(np.log(x) / (1 -
            lambdaa * np.log(x)))
        return np.array([grad_delta, grad_lambd])
    def hessian(self, delta, lambdaa, x):
        n = len(x)
        h11 = -2 * n / (delta ** 2) + n / ((delta + lambdaa) ** 2)
        h22 = n / ((delta + lambdaa) ** 2) - np.sum((np.log(x) ** 2) / ((1
            - lambdaa * np.log(x)) ** 2))
        h12 = n / ((delta + lambdaa) ** 2)
        H = np.array([[h11, h12], [h12, h22]])
```

```
        return H
    def forward(self, x, tol=1e-7, max_iter=1000):
        delta, lambdaa = self.delta_init, self.lambda_init
        for i in range(max_iter):
            grad = self.gradient(delta, lambdaa, x)
            H = self.hessian(delta, lambdaa, x)
            try:
                H_inv = np.linalg.inv(H)
            except np.linalg.LinAlgError:
                H_inv = np.linalg.inv(H + 1e-6*np.eye(2))
            update = H_inv @ grad
            delta, lambdaa = np.array([delta, lambdaa]) - update
            if m.isnan(delta) or m.isnan(lambdaa):
                return self.delta_init, self.lambda_init
            if np.linalg.norm(grad) < tol:
                break
        delta = delta.clip(0.25*self.delta_init + np.random.normal(0, 0.05)
            , 2*self.delta_init + np.random.normal(0, 0.05))
        lambdaa = lambdaa.clip(0.25*self.lambda_init + np.random.normal(0,
            0.05), 2*self.lambda_init + np.random.normal(0, 0.05))
        return delta, lambdaa
```

Listing 3.5　极大似然估计可视化 Python 代码

```
'''
@File: Visualize.py
Desc: Codes for visualization
'''
import math as m
import numpy as np
import matplotlib.pyplot as plt
class Visual:
    def __init__(self, opt):
        self.opt = opt
```

```
def __call__(self, bias_n, mse_n, mean_esti, real):
    return self.forward(bias_n, mse_n, mean_esti, real)
def uexe_pdf(self, x, delta, lambdaa):
    return ((delta**2 * (1 - lambdaa * m.log(x))) / (delta + lambdaa))
        * x**(delta-1)
def uexe_cdf(self, x, delta, lambdaa):
    return (1 - (delta * lambdaa * m.log(x)) / (delta + lambdaa)) * x
        **delta
    ls_x1 = []
    ls_y1 = []
    for x1 in x_c1[0][500]:
        ls_x1.append(x1)
    ls_x1 = sorted(ls_x1)
    for x1 in ls_x1:
        ls_y1.append(self.uexe_pdf(x1, delta, lambdaa))
    ls_x2 = []
    ls_y2 = []
    for x1 in x_c2[0][500]:
        ls_x2.append(x1)
    ls_x2 = sorted(ls_x2)
    for x1 in ls_x2:
        ls_y2.append(self.uexe_pdf(x1, delta=1, lambdaa=2))
    ls_x3 = []
    ls_y3 = []
    for x1 in x_c3[0][500]:
        ls_x3.append(x1)
    ls_x3 = sorted(ls_x3)
    for x1 in ls_x3:
        ls_y3.append(self.uexe_pdf(x1, delta=3, lambdaa=4))
    plt.figure()
    plt.ylim(0, 2)
    plt.plot(ls_x1, ls_y1, color='red', label='delta=0.5,lambda=0.5')
```

```
        plt.plot(ls_x2, ls_y2, color='orange', label='delta=1,lambda=2')
        plt.plot(ls_x3, ls_y3, color='blue', label='delta=3,lambda=4')
        plt.legend()
        plt.show()
    def plot_cdf(self,x_c1, x_c2, x_c3, delta=0.5, lambdaa=0.5):
        ls_x1 = []
        ls_y1 = []
        for x1 in x_c1[0][500]:
            ls_x1.append(x1)
        ls_x1 = sorted(ls_x1)
        for x1 in ls_x1:
            ls_y1.append(self.uexe_cdf(x1, delta, lambdaa))
        ls_x2 = []
        ls_y2 = []
        for x1 in x_c2[0][500]:
            ls_x2.append(x1)
        ls_x2 = sorted(ls_x2)
        for x1 in ls_x2:
            ls_y2.append(self.uexe_cdf(x1, delta=1, lambdaa=2))
        ls_x3 = []
        ls_y3 = []
        for x1 in x_c3[0][500]:
            ls_x3.append(x1)
        ls_x3 = sorted(ls_x3)
        for x1 in ls_x3:
            ls_y3.append(self.uexe_cdf(x1, delta=3, lambdaa=4))
        index = np.arange(0, 1.01, 0.0001).tolist()
        ls_f1 = []
        for i in index:
            count = 0
            for x1 in ls_x1:
                if x1 < i:
```

```
            count += 1
        else:
            break
    ls_f1.append(count/500)
ls_f2 = []
for i in index:
    count = 0
    for x1 in ls_x2:
        if x1 < i:
            count += 1
        else:
            break
    ls_f2.append(count/500)
ls_f3 = []
for i in index:
    count = 0
    for x1 in ls_x3:
        if x1 < i:
            count += 1
        else:
            break
    ls_f3.append(count/500)
plt.figure()
plt.ylim(0, 1)
plt.plot(index, ls_f1, color='red', label='delta=0.5,lambda=0.5')
plt.plot(index, ls_f2, color='orange', label='delta=1,lambda=2')
plt.plot(index, ls_f3, color='blue', label='delta=3,lambda=4')
plt.legend()
plt.show()
plt.figure()
plt.ylim(0, 1)
plt.plot(ls_x1, ls_y1, color='red', label='delta=0.5,lambda=0.5')
```

```
        plt.plot(ls_x2, ls_y2, color='orange', label='delta=1,lambda=2')
        plt.plot(ls_x3, ls_y3, color='blue', label='delta=3,lambda=4')
        plt.legend()
        plt.show()
    def esti_plot(self, mean_esti, real):
        ls_d = []
        ls_l = []
        for n in mean_esti:
            ls_d.append(mean_esti[n][0])
            ls_l.append(mean_esti[n][1])
        n_list = self.opt.n_list
        plt.plot(n_list, ls_d, color='red')
        plt.axhline(y=real[0], color='orange', linestyle='--', linewidth
            =2)
        plt.xlabel('n')
        plt.ylabel('Delta estimation')
        plt.show()
        plt.plot(n_list, ls_l, color='red')
        plt.axhline(y=real[1], color='orange', linestyle='--', linewidth
            =2)
        plt.xlabel('n')
        plt.ylabel('Lambda estimation')
        plt.show()
    def bias_plot(self, bias_n):
        ls_d = []
        ls_l = []
        for n in bias_n:
            ls_d.append(bias_n[n][0])
            ls_l.append(bias_n[n][1])
        n_list = self.opt.n_list
        plt.plot(n_list, ls_d, color='orange')
        plt.xlabel('n')
```

```
        plt.ylabel('Delta estimation bias')
        plt.show()
        plt.plot(n_list, ls_l, color='orange')
        plt.xlabel('n')
        plt.ylabel('Lambda estimation bias')
        plt.show()
    def mse_plot(self, mse_n):
        ls_d = []
        ls_l = []
        for n in mse_n:
            ls_d.append(mse_n[n][0])
            ls_l.append(mse_n[n][1])
        n_list = self.opt.n_list
        plt.plot(n_list, ls_d, color='red')
        plt.xlabel('n')
        plt.ylabel('Delta estimation mse')
        plt.show()
        plt.plot(n_list, ls_l, color='red')
        plt.xlabel('n')
        plt.ylabel('Lambda estimation mse')
        plt.show()
    def forward(self, bias_n, mse_n, mean_esti, real):
        self.esti_plot(mean_esti, real)
        self.bias_plot(bias_n)
        self.mse_plot(mse_n)
```

Listing 3.6　极大似然估计评估指标 Python 代码

```
'''
@File: metrics.py
Desc: Main for simulation
'''
import numpy as np
class Metrics:
```

```
def __init__(self, opt):
    self.opt = opt
def __call__(self, real, mean_esti, ls_esti):
    return self.forward(real, mean_esti, ls_esti)
def bias(self, real, mean_esti):
    '''
    @Params:
    real: Real values list[real delta, real lambda]
    mean_esti: Estimated statistics

    @Return:
    bias_n: dic{n:[bias_delta, bias_lambda]}, where e.g., n:[100, 150,
        200, 300, 400, 500]
    '''
    bias_n = {}
    for n in mean_esti:
        if n not in bias_n:
            bias_n[n] = [mean_esti[n][0]-real[0], mean_esti[n][1]-real
                [1]]
    return bias_n
def mse(self, real, ls_esti):
    '''
    @Params:
    real: Real values list[real delta, real lambda]
    ls_esti: dic{n:[[delta list], [lambda list]]}, where e.g., n:[100,
        150, 200, 300, 400, 500],
            there are n estimated delta and lambda in each delta list
                and lambda list.
    @Return:
    mse_n: dic{n:[mse_delta, mse_lambda]}, where e.g., n:[100, 150,
        200, 300, 400, 500]
    '''
```

```
        mse_n = {}
        for n in self.opt.n_list:
            biasd2 = []
            biasl2 = []
            for delta in ls_esti[n][0]:
                biasd2.append((delta - real[0])**2)
            for lambdaa in ls_esti[n][1]:
                biasl2.append((lambdaa - real[1])**2)
            if n not in mse_n:
                mse_n[n] = [np.mean(biasd2), np.mean(biasl2)]
        return mse_n
    def forward(self, real, mean_esti, ls_esti):
        '''
        @Params:
        real: Real values list [real delta, real lambda]
        mean_esti: Estimated statistics
        ls_esti: dic{n:[[delta list], [lambda list]]}, where e.g., n:[100,
            150, 200, 300, 400, 500],
                there are n estimated delta and lambda in each delta list
                    and lambda list.
        @Return:
        bias_n: dic{n:[bias_delta, bias_lambda]}, where e.g., n:[100, 150,
            200, 300, 400, 500]
        mse_n: dic{n:[mse_delta, mse_lambda]}, where e.g., n:[100, 150,
            200, 300, 400, 500]
        '''
        return self.bias(real, mean_esti), self.mse(real, ls_esti)
```

Listing 3.7　极大似然估计配置参数 Python 代码

```
'''
@File: config.py
Desc: Parameters for Experiments
'''
```

```
import argparse
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--seed', type=int, default=2024, help='Seed')
    parser.add_argument('--K', type=int, default=1000, help='Number of
        samples')
    parser.add_argument('--n_list', type=str, default="[100, 150, 200,
        300, 400, 500]", help='Number of items in each sample')
    parser.add_argument('--func_name', type=str, default='UExE', help='pdf
        to be simulated')
    parser.add_argument('-- delta_init', type=float, default=0.5, help='
        initial value of delta')
    parser.add_argument('--lambda_init', type=float, default=0.5, help='
        initial value of lambda')
    return parser.parse_args()
```

Listing 3.8　极大似然估计模拟 python 代码

```
'''
@File: main.py
Desc: Main for simulation
'''
from DataGenerator import DataGenerator
from MLE_Estimate import MLE_Estimate
from metrics import Metrics
from Visualize import Visual
from config import parse_args
import numpy as np
import pickle
import logging
import os
logging.basicConfig(level=logging.INFO)
def data_generation(opt):
    K = opt.K
```

```
    n_list = opt.n_list
    logging.info("----------Generating Data----------")
    datagenerator = DataGenerator(opt)
    data1 = datagenerator(K, n_list, delta=0.5, lambdaa=0.5)
    data2 = datagenerator(K, n_list, delta=1, lambdaa=2)
    data3 = datagenerator(K, n_list, delta=3, lambdaa=4)
    logging.info("----------Saving Data----------")
    pickle.dump(data1, open('SimulationData_Case1.pkl', 'wb'))
    pickle.dump(data2, open('SimulationData_Case2.pkl', 'wb'))
    pickle.dump(data3, open('SimulationData_Case3.pkl', 'wb'))
    logging.info("----------Data Saved----------")
    return data1, data2, data3
def mle_estimate(opt, data):
    '''
    @Params:
    opt: configuration
    data: dic{K:{n:[sample list]}}, where length of [sample list] is n
    @Returns:
    ls_kmean: dic{n:[mean(delta list), mean(lambda list)]}, where n:[100,
        150, 200, 300, 400, 500]
    ls_k: dic{n:[[delta list], [lambda list]]}, where n:[100, 150, 200,
        300, 400, 500]
    '''
    mle = MLE_Estimate(opt)
    ls_kmean = {}
    ls_k = {}
    for n in opt.n_list:
        ls_d = []
        ls_l = []
        for k in data:
            data_ls = np.array(data[k][n])
            delta, lambdaa = mle(data_ls)
```

```
                ls_d.append(delta)
                ls_l.append(lambdaa)
            if n not in ls_kmean:
                ls_kmean[n] = [np.mean(ls_d), np.mean(ls_l)]
                ls_k[n] = [ls_d, ls_l]
    return ls_kmean, ls_k
def main():
    opt = parse_args()
    opt.n_list = eval(opt.n_list)
    metrics = Metrics(opt)
    visual = Visual(opt)
    if os.path.exists('SimulationData_Case1.pkl') \
    and os.path.exists('SimulationData_Case2.pkl') \
    and os.path.exists('SimulationData_Case3.pkl'):
        data1 = pickle.load(open('SimulationData_Case1.pkl', 'rb'))
        data2 = pickle.load(open('SimulationData_Case2.pkl', 'rb'))
        data3 = pickle.load(open('SimulationData_Case3.pkl', 'rb'))
    else:
        data1, data2, data3 = data_generation(opt)
    visual.plot_pdf(data1, data2, data3)
    visual.plot_cdf(data1, data2, data3)
    case1_mean, case1_ls = mle_estimate(opt, data1)
    bias_n, mse_n = metrics([0.5, 0.5], case1_mean, case1_ls)
    logging.info('Case1:\nEstimate␣Result:{}\nBias:{}\nMSE:{}'.format(
        case1_mean, bias_n, mse_n))
    visual(bias_n, mse_n, case1_mean, [0.5, 0.5])
    opt.delta_init = 1
    opt.lambda_init = 2
    case2_mean, case2_ls = mle_estimate(opt, data2)
    bias_n, mse_n = metrics([1, 2], case2_mean, case2_ls)
    logging.info('Case2:\nEstimate␣Result:{}\nBias:{}\nMSE:{}'.format(
        case2_mean, bias_n, mse_n))
```

```
        visual(bias_n, mse_n, case2_mean, [1.0, 2.0])
        opt.delta_init = 3
        opt.lambda_init = 4
        case3_mean, case3_ls = mle_estimate(opt, data3)
        bias_n, mse_n = metrics([3, 4], case3_mean, case3_ls)
        logging.info('Case3:\nEstimate Result:{}\nBias:{}\nMSE:{}'.format(
            case3_mean, bias_n, mse_n))
        visual(bias_n, mse_n, case3_mean, [3.0, 4.0])
    if __name__ == '__main__':
        main()
```

Listing 3.9 置信区间估计 Python 代码

```
import numpy as np
import scipy.stats as stats
import pickle
from main import main
from config import parse_args
from DataGenerator import DataGenerator
from MLE_Estimate import MLE_Estimate
from config import parse_args
import numpy as np
import pickle
import logging
import os
logging.basicConfig(level=logging.INFO)
file_path = 'SimulationData_Case3.pkl'
with open(file_path, 'rb') as file:
    sample = pickle.load(file)# sample 的形式{}
opt = parse_args()
# 假设给定的对数似然函数的二阶导数，构造信息矩阵的元素
def second_derivatives(params, x):
    delta, lam = params
    n = len(x)
```

```
    x = np.array(x)
    # 计算对数似然函数的二阶偏导数（示例计算）
    d2ldelta2 = (2 * n / delta**2) - (n / (delta + lam)**2)
    d2ldelta_lambda = -n / (delta + lam)**2 # 示例表达式
    d2lambd2 = -n / (delta + lam)**2 + np.sum(np.log(x)**2 / (1 - lam * np.log
        (x))**2) # 示例表达式
    # 返回信息矩阵的元素
    return d2ldelta2, d2lambd2, d2ldelta_lambda
# 计算信息矩阵和其逆矩阵
def compute_information_matrix(params, x):
    d2ldelta2, d2lambd2, d2ldelta_lambda = second_derivatives(params, x)
    # 构造信息矩阵
    info_matrix = np.array([[d2ldelta2, d2ldelta_lambda],
                            [d2ldelta_lambda, d2lambd2]])
    # 计算信息矩阵的逆
    def inverse_2x2(matrix):
        # 提取元素
        a, b = matrix[0, 0], matrix[0, 1]
        c, d = matrix[1, 0], matrix[1, 1]
        # 计算行列式
        determinant = a * d - b * c
        if determinant == 0:
            raise ValueError("矩阵不可逆")
        # 计算逆矩阵
        inverse = (1 / determinant) * np.array([[d, -b],
                                                [-c, a]])
        return inverse
    # info_matrix_inv = np.linalg.inv(info_matrix)
    info_matrix_inv = inverse_2x2(info_matrix)
    return info_matrix_inv
def calculate_intertval_case1():
    case1_params, case2_params, case3_params= main()
```

```
    n_size = [600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600,
        1700, 1800, 1900, 2000]
    intervals = {}
    for size in n_size:
        delta_list , lam_list = case3_params[size]
        intervals [size] = {}
        i=0
        for j in range(len( delta_list )):
            params = [ delta_list [j], lam_list [j]]
            data = sample[j][ size]
            I_inv = compute_information_matrix(params, data)
            # 获取方差
            var_delta = I_inv [0, 0]  # delta的方差
            var_lambda = I_inv [1, 1]  # lambda的方差
            # 计算标准误差
            se_delta = np. sqrt (abs( var_delta ))
            se_lambda = np. sqrt (abs(var_lambda))
            # 计算95%置信区间
            Z_critical = stats .norm.ppf(0.975)  # Z值, 对于95%置信区间,
            # 置信区间
            conf_interval_delta = ( delta_list [j] - Z_critical * se_delta ,
                delta_list [j] + Z_critical * se_delta )
            conf_interval_lambda = ( lam_list [j] - Z_critical * se_lambda,
                lam_list [j] + Z_critical * se_lambda)
            intervals [size][i] = [ conf_interval_delta , conf_interval_lambda ]
            i += 1
    return intervals
interval = calculate_intertval_case1 ()
for j in [600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700,
    1800, 1900, 2000]:
    l_delta = 0
    r_delta = 0
```

```
    l_lam, r_lam =0 ,0
    delta_len , lam_len = 0,0
    for k,v in interval [j ].items():
        l_delta += v[0][0]
        r_delta += v[0][1]
        l_lam += v[1][0]
        r_lam += v[1][1]
        delta_len += v[0][1] - v[0][0]
        lam_len += v[1][1]-v[1][0]
    # 计算覆盖率
    print(j ,"AL:", delta_len /len( interval [j ]), lam_len/len( interval [j ]))
def calculate_coverage ( true_delta , true_lambda):
    coverage = {}
    n_size = [600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600,
        1700, 1800, 1900, 2000]
    for size in n_size:
        coverage_count_delta = 0
        coverage_count_lambda = 0
        interval = calculate_intertval_case1 ()[ size ]
        coverage[ size ] = {}
        for i in range(len( interval )):
            conf_interval_delta , conf_interval_lam = interval [i]
            # 检查置信区间是否覆盖真实的delta和lambda
            if conf_interval_delta [0] <= true_delta and true_delta <=
                conf_interval_delta [1]:
                coverage_count_delta += 1
            if conf_interval_lam [0] <= true_lambda and true_lambda <=
                conf_interval_lam [1]:  # 示例的lambda覆盖
                coverage_count_lambda += 1
        # 计算覆盖率
        delta_coverage_rate = coverage_count_delta / len( interval )
        lambda_coverage_rate = coverage_count_lambda / len( interval )
```

```
        coverage[ size ][ ' delta ' ] = delta_coverage_rate
        coverage[ size ][ 'lam' ] = lambda_coverage_rate
    return coverage
coverage = calculate_coverage ( true_delta =3.0, true_lambda=4.0)
print(coverage)
```

Listing 3.10　置信区间可视化 Python 代码

```
import matplotlib . pyplot as plt
n_list = [100, 150, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200,
    1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000]
case1_delta_AL = [0.29043270,0.28978012,0.20312707, 0.17532852, 0.15055318,
    0.13509339, 0.094636, 0.0732876, 0.076457, 0.0427945, 0.0850899,
    0.0590222, 0.0861327, 0.06145025, 0.0412185, 0.0583866, 0.0583797,
    0.0523153, 0.04813564, 0.06167466 ,0.0675837]
case1_lam_AL = [1.922718,1.90152,1.34598,1.133325,0.96425,0.86674, 0.4301853,
     0.4030909, 0.2225876, 0.2012340, 0.2356864, 0.211993, 0.227620, 0.1658872,
     0.02826149, 0.2007125, 0.276359, 0.398948, 0.150096, 0.268439, 0.249352]
case2_delta_AL = [0.460749,0.383508,0.333960, 0.271511 ,0.233923,0.208965,
    0.18223195, 0.16245528, 0.18348282, 0.0949992, 0.14612473, 0.1621539,
    0.08988222, 0.11517323, 0.08949564,0.1097848, 0.09337907, 0.11354658,
    0.06791873, 0.08883055, 0.08656186]
case2_lam_AL = [7.46980,6.33110, 5.47793,4.21907,3.534703,3.12072, 3.0713798,
    2.47275597, 2.72431536, 1.44550419, 2.1071645, 2.29537453,1.4604006,
    1.6780735, 1.41213434, 1.57005232, 1.22714673, 2.2399844, 1.01894643,
    1.39100494,1.04994690 ]
case3_delta_AL = [1.64563 ,1.37259 ,1.18706,0.99931,0.84387, 0.79329,
    0.66703122, 0.60156048,0.73393645,0.606311, 0.48891021,0.5079298
    ,0.460773034,0.4607738, 0.48562878, 0.42862701, 0.44584717,
    0.388128155,0.37298252 , 0.3605024,0.35109241]
case3_lam_AL = [14.173358,11.41539,9.998974,8.495354,
    7.245865,6.8498572,6.6404721, 4.5324278, 5.0043682, 5.585836, 4.1014163,
    4.2939529, 3.51580569, 4.2402973, 4.8538850, 3.3467173, 4.1769967,
    3.29161592,3.23196465, 3.11875439, 3.02488545]
```

```
# 创建图形和轴
fig, ax1 = plt.subplots(figsize=(10, 6))
# 绘制 Delta_AL 在主Y轴
color = 'tab:blue'
ax1.set_xlabel('n (Sample Size)')
ax1.set_ylabel('Delta_AL', color=color)
ax1.plot(n_list, case3_delta_AL, color=color, label='Delta_AL')
ax1.tick_params(axis='y', labelcolor=color)
ax1.legend(loc="upper left")
# 创建副Y轴
ax2 = ax1.twinx()
color = 'tab:orange'
ax2.set_ylabel('Lam_AL', color=color)
ax2.plot(n_list, case3_lam_AL, color=color, label='Lam_AL')
ax2.tick_params(axis='y', labelcolor=color)
ax2.legend(loc="upper right")
# 添加标题和网格
plt.grid()
# 显示图形
plt.show()
```

Listing 3.11　真实数据拟合 Python 代码

```
import numpy as np
import scipy.optimize as optimize
from scipy.special import gamma
from scipy import stats
from statsmodels.distributions.empirical_distribution import ECDF
# 数据集
data1 = np.array([0.023, 0.032, 0.054, 0.069, 0.081, 0.094, 0.105, 0.127,
    0.148,
                  0.169, 0.188, 0.216, 0.255, 0.277, 0.311, 0.361, 0.376,
                      0.395,
                  0.432, 0.463, 0.481, 0.519, 0.529, 0.567, 0.642, 0.674,
```

```
                0.752,
               0.823,  0.887,  0.926])
data2 = np.array([0.853,  0.759,  0.874,  0.800,  0.716,  0.557,  0.503,  0.399,
    0.334,
                  0.207,  0.118,  0.097,  0.078,  0.067,  0.056,  0.044,  0.036,
                    0.026,
                  0.019,  0.014,  0.010,  0.118])
def uexe_log_likelihood(params, data):
    delta, lambd = params
    # 检查参数范围
    if delta <= 0 or lambd <= 0:
        return np.inf  # 参数或数据越界时，返回正无穷作为惩罚
    # 计算负对数似然值
    log_pdf = (
        2 * np.log(delta)
        + np.log(1 - lambd * np.log(data))
        - np.log(delta + lambd)
        + (delta - 1) * np.log(data)
    )
    return -np.sum(log_pdf)
def beta_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(stats.beta.logpdf(data, b, a))
def kumaraswamy_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(b) + np.log(a) + (b - 1) * np.log(data) + (a - 1) *
        np.log(1 - data**b))
def unit_gamma_log_likelihood(params, data):
```

```
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    log_pdf = (
        np.log(b)
        + (b-1) * np.log(data)
        + (a-1) * np.log((-b) * np.log(data))
        - np.log(gamma(a))
    )
    return -np.sum(log_pdf)
def power_topp_leone_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(2) + np.log(a) + np.log(b) + (a * b - 1) * np.log(
        data) + np.log(1 - data**a) + (b-1) * np.log(2 - data**a))
def bounded_weighted_exponential_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(a+1) + np.log(b) - np.log(a) + (b-1) * np.log(data)
        + np.log(1 - data**(a*b)))
def unit_burr_xii_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(a) + np.log(b) - (1+a) * np.log(1 + (-np.log(data))
        **b) - np.log(data) - (1-b) * np.log(-np.log(data)))
def unit_birnbaum_saunders_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
```

```
    first  = np.sqrt(-b/np.log(data)) + np.sqrt((-b/np.log(data))**3)
    second1 = b/np.log(data)
    second2 = np.log(data)/b
    second = (second1 + second2 + 2)/(a**2)
    second = second/2
    third  = 2 * a * b * data * np.sqrt(2*np.pi)
    log_pdf = (
            np.log( first )
            + second
            - np.log( third )
    )
    return -np.sum(log_pdf)
def unit_inverse_gaussian_log_likelihood (params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    second_1 = a * ((np.log(data) + b)**2)
    second_2 = 2 * np.log(data) * (b**2)
    second = second_1 / second_2
    c = (np.log(data))**2
    log_pdf = (
            np.log(a)
            + second
            - 0.5 * np.log(2 * np.pi)
            - np.log(data)
            - np.log(c)
            - 0.5 * np.log(-a/np.log(data))
    )
    return -np.sum(log_pdf)
def unit_mirra_log_likelihood (params, data):
    a, b = params
    if a <= 0 or b <= 0:
```

```
        return np.inf
    log_pdf = (
            3 * np.log(b)
            + np.log(a + (a+2) * (data**2) - 2 * a * data)
            + b
            - b/data
            - np.log(2 * (data**4))
            - np.log(a + (b**2))
    )
    return -np.sum(log_pdf)
# 最大似然估计函数
def mle_estimation(log_likelihood_func, initial_guess, data):
    result = optimize.minimize(log_likelihood_func, initial_guess, args=(data
        ,), method='L-BFGS-B',
                               bounds=[(1e-6, None), (1e-6, None)])
    params = result.x
    hessian_inv = result.hess_inv.todense()  # 提取逆Hessian矩阵用于估计标准
        误差
    se = np.sqrt(np.diag(hessian_inv)) if result.success else [np.nan] * len(
        params)
    return params, se
# 定义初始猜测值和分布
initial_guesses = {
    'UEXE': [1, 1],
    'Beta': [1, 1],
    'Kumaraswamy': [1, 1],
    'Unit Gamma': [1, 1],
    'Power Topp_Leone': [1, 1],
    'Bounded Weighted Exponential': [1, 1],
    'Unit Burr XII': [1, 1],
    'Unit Birnbaum_Saunders': [1, 1],
    'Unit Inverse Gaussian': [1, 1],
```

```
    'Unit␣Mirra': [1, 1],
}
distributions = {
    'UEXE': uexe_log_likelihood,
    'Beta': beta_log_likelihood,
    'Kumaraswamy': kumaraswamy_log_likelihood,
    'Unit␣Gamma': unit_gamma_log_likelihood,
    'Power␣Topp_Leone': power_topp_leone_log_likelihood,
    'Bounded␣Weighted␣Exponential':
        bounded_weighted_exponential_log_likelihood,
    'Unit␣Burr␣XII': unit_burr_xii_log_likelihood,
    'Unit␣Birnbaum_Saunders': unit_birnbaum_saunders_log_likelihood,
    'Unit␣Inverse␣Gaussian': unit_inverse_gaussian_log_likelihood,
    'Unit␣Mirra': unit_mirra_log_likelihood,
}
# 计算每种分布的参数估计和标准误差（对于数据集1）
results_data1 = {}
for dist_name, log_likelihood_func in distributions.items():
    initial_guess = initial_guesses[dist_name]
    params, se = mle_estimation(log_likelihood_func, initial_guess, data1)
    results_data1[dist_name] = {'params': params, 'se': se}
# 计算每种分布的参数估计和标准误差（对于数据集2）
results_data2 = {}
for dist_name, log_likelihood_func in distributions.items():
    initial_guess = initial_guesses[dist_name]
    params, se = mle_estimation(log_likelihood_func, initial_guess, data2)
    results_data2[dist_name] = {'params': params, 'se': se}
# 打印结果的函数
def format_results(results):
    print("MLE␣estimates␣and␣their␣SEs␣are␣in␣parentheses␣for␣the␣dataset.")
    print(f"{'Model':<15}{'MLE␣Estimates':<50}")
    for dist_name, result in results.items():
```

```
params = result['params']
ses = result['se']
if dist_name == "UEXE":
    print(f"{dist_name}(λ,δ)".ljust(15) +
          f"^λ={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
          f"^δ={params[1]:.6f}({ses[1]:.6f})")
elif dist_name == "Beta":
    print(f"{dist_name}(κ,γ)".ljust(15) +
          f"^κ={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
          f"^γ={params[1]:.6f}({ses[1]:.6f})")
elif dist_name == "Kumaraswamy":
    print(f"Kum(a,b)".ljust(15) +
          f"^a={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
          f"^b={params[1]:.6f}({ses[1]:.6f})")
elif dist_name == "Unit Gamma":
    print(f"UG(θ,ρ)".ljust(15) +
          f"^θ={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
          f"ρ={params[1]:.6f}({ses[1]:.6f})")
elif dist_name == "Power Topp_Leone":
    print(f"PTL(ν,τ)".ljust(15) +
          f"^ν={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
          f"^τ={params[1]:.6f}({ses[1]:.6f})")
elif dist_name == "Bounded Weighted Exponential":
    print(f"BWE(σ,η)".ljust(15) +
          f"^σ={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
          f"^η={params[1]:.6f}({ses[1]:.6f})")
elif dist_name == "Unit Burr XII":
    print(f"UBXII(μ,ϕ)".ljust(15) +
          f"^μ={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
          f"^ϕ={params[1]:.6f}({ses[1]:.6f})")
elif dist_name == "Unit Birnbaum_Saunders":
    print(f"UBS(α,β)".ljust(15) +
```

```
                f"α={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
                f"β={params[1]:.6f}({ses[1]:.6f})")
        elif dist_name == "Unit Inverse Gaussian":
            print(f"UIG(ζ,ω)".ljust(15) +
                f"ζ={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
                f"ω={params[1]:.6f}({ses[1]:.6f})")
        elif dist_name == "Unit Mirra":
            print(f"UM(ψ,Θ)".ljust(15) +
                f"ψ={params[0]:.6f}({ses[0]:.6f})".ljust(30) +
                f"Θ={params[1]:.6f}({ses[1]:.6f})")
# 打印数据集1结果
format_results(results_data1)
print()
# 打印数据集2结果
format_results(results_data2)
print()
# 计算信息准则：AIC、CAIC、HQIC
def calculate_info_criteria(log_likelihood, k, n):
    aic = 2 * k - 2 * log_likelihood
    caic = aic + 2 * k * (k + 1) / (n - k - 1)
    hqic = 2 * k * np.log(np.log(n)) - 2 * log_likelihood
    return aic, caic, hqic
def format_results_with_criteria(results, data):
    print("MLE estimates, SEs, and information criteria for the dataset.")
    print(f"{'Model':<15}{'Log_L':<15}{'AIC':<15}{'CAIC':<15}{'HQIC':<15}"
        )
    n = len(data)  # 数据点数量
    for dist_name, result in results.items():
        params = result['params']
        ses = result['se']
        # 计算对数似然值
        log_likelihood_func = distributions[dist_name]
```

```
log_likelihood = -log_likelihood_func(params, data)  # 负对数似然
# 计算AIC、CAIC、HQIC
k = len(params)  # 参数个数
aic, caic, hqic = calculate_info_criteria(log_likelihood, k, n)
if dist_name == "UEXE":
    print(f"{dist_name}(λ,δ)".ljust(15) +
          f"{log_likelihood:.6f}".ljust(15) +
          f"{aic:.6f}".ljust(15) +
          f"{caic:.6f}".ljust(15) +
          f"{hqic:.6f}".ljust(15))
elif dist_name == "Beta":
    print(f"{dist_name}(κ,γ)".ljust(15) +
          f"{log_likelihood:.6f}".ljust(15) +
          f"{aic:.6f}".ljust(15) +
          f"{caic:.6f}".ljust(15) +
          f"{hqic:.6f}".ljust(15))
elif dist_name == "Kumaraswamy":
    print(f"Kum(a,b)".ljust(15) +
          f"{log_likelihood:.6f}".ljust(15) +
          f"{aic:.6f}".ljust(15) +
          f"{caic:.6f}".ljust(15) +
          f"{hqic:.6f}".ljust(15))
elif dist_name == "Unit Gamma":
    print(f"UG(θ,ρ)".ljust(15) +
          f"{log_likelihood:.6f}".ljust(15) +
          f"{aic:.6f}".ljust(15) +
          f"{caic:.6f}".ljust(15) +
          f"{hqic:.6f}".ljust(15))
elif dist_name == "Power Topp_Leone":
    print(f"PTL(ν,τ)".ljust(15) +
          f"{log_likelihood:.6f}".ljust(15) +
          f"{aic:.6f}".ljust(15) +
```

```
            f"{caic:.6f}".ljust(15) +
            f"{hqic:.6f}".ljust(15))
    elif dist_name == "Bounded Weighted Exponential":
        print(f"BWE(σ,η)".ljust(15) +
            f"{log_likelihood:.6f}".ljust(15) +
            f"{aic:.6f}".ljust(15) +
            f"{caic:.6f}".ljust(15) +
            f"{hqic:.6f}".ljust(15))
    elif dist_name == "Unit Burr XII":
        print(f"UBXII(μ,ϕ)".ljust(15) +
            f"{log_likelihood:.6f}".ljust(15) +
            f"{aic:.6f}".ljust(15) +
            f"{caic:.6f}".ljust(15) +
            f"{hqic:.6f}".ljust(15))
    elif dist_name == "Unit Birnbaum_Saunders":
        print(f"UBS(α,β)".ljust(15) +
            f"{log_likelihood:.6f}".ljust(15) +
            f"{aic:.6f}".ljust(15) +
            f"{caic:.6f}".ljust(15) +
            f"{hqic:.6f}".ljust(15))
    elif dist_name == "Unit Inverse Gaussian":
        print(f"UIG(ζ,ω)".ljust(15) +
            f"{log_likelihood:.6f}".ljust(15) +
            f"{aic:.6f}".ljust(15) +
            f"{caic:.6f}".ljust(15) +
            f"{hqic:.6f}".ljust(15))
    elif dist_name == "Unit Mirra":
        print(f"UM(ψ,Θ)".ljust(15) +
            f"{log_likelihood:.6f}".ljust(15) +
            f"{aic:.6f}".ljust(15) +
            f"{caic:.6f}".ljust(15) +
            f"{hqic:.6f}".ljust(15))
```

```
# 打印数据集1结果，包含信息准则
  format_results_with_criteria (results_data1, data1)
print()
# 打印数据集2结果，包含信息准则
  format_results_with_criteria (results_data2, data2)
```

Listing 3.12　各个模型概率函数 Python 代码

```
import numpy as np
import scipy.optimize as optimize
from scipy.special import gamma
from scipy import stats
from statsmodels.distributions.empirical_distribution import ECDF
import matplotlib.pyplot as plt
########################################################## log likelihood
    ##########################################
def uexe_log_likelihood(params, data):
    delta, lambd = params
    # 检查参数范围
    if delta <= 0 or lambd <= 0:
        return np.inf  # 参数或数据越界时，返回正无穷作为惩罚
    # 计算负对数似然值
    log_pdf = (
        2 * np.log(delta)
        + np.log(1 - lambd * np.log(data))
        - np.log(delta + lambd)
        + (delta - 1) * np.log(data)
    )
    return -np.sum(log_pdf)
def beta_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(stats.beta.logpdf(data, b, a))
```

```
def kumaraswamy_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(b) + np.log(a) + (b - 1) * np.log(data) + (a - 1) *
        np.log(1 - data**b))
def unit_gamma_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    log_pdf = (
        np.log(b)
        + (b-1) * np.log(data)
        + (a-1) * np.log((-b) * np.log(data))
        - np.log(gamma(a))
    )
    return -np.sum(log_pdf)
def power_topp_leone_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(2) + np.log(a) + np.log(b) + (a * b - 1) * np.log(
        data) + np.log(1 - data**a) + (b-1) * np.log(2 - data**a))
def bounded_weighted_exponential_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(a+1) + np.log(b) - np.log(a) + (b-1) * np.log(data)
        + np.log(1 - data**(a*b)))
def unit_burr_xii_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
```

```
        return np.inf
    return -np.sum(np.log(a) + np.log(b) - (1+a) * np.log(1 + (-np.log(data))
        **b) - np.log(data) - (1-b) * np.log(-np.log(data)))
def unit_birnbaum_saunders_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    first = np.sqrt(-b/np.log(data)) + np.sqrt((-b/np.log(data))**3)
    second1 = b/np.log(data)
    second2 = np.log(data)/b
    second = (second1 + second2 + 2)/(a**2)
    second = second/2
    third = 2 * a * b * data * np.sqrt(2*np.pi)
    log_pdf = (
            np.log( first )
            + second
            - np.log( third )
    )
    return -np.sum(log_pdf)
def unit_inverse_gaussian_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    second_1 = a * ((np.log(data) + b)**2)
    second_2 = 2 * np.log(data) * (b**2)
    second = second_1 / second_2
    c = (np.log(data))**2
    log_pdf = (
            np.log(a)
            + second
            - 0.5 * np.log(2 * np.pi)
            - np.log(data)
```

```
            - np.log(c)
            - 0.5 * np.log(-a/np.log(data))
    )
    return -np.sum(log_pdf)
def unit_mirra_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    log_pdf = (
            3 * np.log(b)
            + np.log(a + (a+2) * (data**2) - 2 * a * data)
            + b
            - b/data
            - np.log(2 * (data**4))
            - np.log(a + (b**2))
    )
    return -np.sum(log_pdf)
############################################## pdf log
    ##########################################
def uexe_log_pdf(params, data):
    delta, lambd = params
    # 检查参数范围
    if delta <= 0 or lambd <= 0:
        return np.inf  # 参数或数据越界时，返回正无穷作为惩罚
    # 计算负对数似然值
    log_pdf = (
        2 * np.log(delta)
        + np.log(1 - lambd * np.log(data))
        - np.log(delta + lambd)
        + (delta - 1) * np.log(data)
    )
    return log_pdf
```

```
def beta_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return stats.beta.logpdf(data, b, a)
def kumaraswamy_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return np.log(b) + np.log(a) + (b - 1) * np.log(data) + (a - 1) * np.log(1
        - data**b)
def unit_gamma_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    log_pdf = (
        np.log(b)
        + (b-1) * np.log(data)
        + (a-1) * np.log((-b) * np.log(data))
        - np.log(gamma(a))
    )
    return log_pdf
def power_topp_leone_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return np.log(2) + np.log(a) + np.log(b) + (a * b - 1) * np.log(data) + np
        .log(1 - data**a) + (b-1) * np.log(2 - data**a)
def bounded_weighted_exponential_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
```

```
    return np.log(a+1) + np.log(b) - np.log(a) + (b-1) * np.log(data) + np.log
        (1 - data**(a*b))
def unit_burr_xii_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return np.log(a) + np.log(b) - (1+a) * np.log(1 + (-np.log(data))**b) - np
        .log(data) - (1-b) * np.log(-np.log(data))
def unit_birnbaum_saunders_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    first = np.sqrt(-b/np.log(data)) + np.sqrt((-b/np.log(data))**3)
    second1 = b/np.log(data)
    second2 = np.log(data)/b
    second = (second1 + second2 + 2)/(a**2)
    second = second/2
    third = 2 * a * b * data * np.sqrt(2*np.pi)
    log_pdf = (
            np.log( first )
            + second
            - np.log( third )
    )
    return log_pdf
def unit_inverse_gaussian_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    second_1 = a * ((np.log(data) + b)**2)
    second_2 = 2 * np.log(data) * (b**2)
    second = second_1 / second_2
    c = (np.log(data))**2
```

```
    log_pdf = (
            np.log(a)
            + second
            - 0.5 * np.log(2 * np.pi)
            - np.log(data)
            - np.log(c)
            - 0.5 * np.log(-a/np.log(data))
    )
    return log_pdf
def unit_mirra_log_pdf(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    log_pdf = (
            3 * np.log(b)
            + np.log(a + (a+2) * (data**2) - 2 * a * data)
            + b
            - b/data
            - np.log(2 * (data**4))
            - np.log(a + (b**2))
    )
    return log_pdf
# 定义分布的累计分布函数(CDF)，需要根据每个分布调整
def uexe_cdf(data, delta, lambd):
    return 1 - np.exp(-lambd * np.log(data) ** delta)
def beta_cdf(data, a, b):
    # 使用自定义CDF实现
    return np.cumsum(data ** (a - 1) * (1 - data) ** (b - 1)) / np.sum(data **
        (a - 1) * (1 - data) ** (b - 1))
def kumaraswamy_cdf(data, a, b):
    return 1 - (1 - data ** a) ** b
def unit_gamma_cdf(data, a, b):
```

```
    return 1 - np.exp(-b * np.log(data) ** a)
def power_topp_leone_cdf(data, a, b):
    return 1 - (1 - data ** a) ** b
def bounded_weighted_exponential_cdf(data, a, b):
    return 1 - (1 - data ** a) ** b
def unit_burr_xii_cdf(data, a, b):
    return 1 - np.exp(-(np.log(data) ** b) ** a)
def unit_birnbaum_saunders_cdf(data, a, b):
    return 1 - np.exp(-a * (data ** 2) / (b ** 2))
def unit_inverse_gaussian_cdf(data, a, b):
    return 1 - np.exp(-((np.log(data) + b) ** 2) / (2 * a))
def unit_mirra_cdf(data, a, b):
    return 1 - np.exp(-b * (data ** a))
```

Listing 3.13　真实数据拟合优度检验 Python 代码

```
import numpy as np
import pandas as pd
import scipy.optimize as optimize
from scipy.special import gamma
from scipy import stats
from statsmodels.distributions.empirical_distribution import ECDF
import matplotlib.pyplot as plt
from model import *
import random
# 数据集
data1 = np.array([0.023, 0.032, 0.054, 0.069, 0.081, 0.094, 0.105, 0.127,
    0.148,
                  0.169, 0.188, 0.216, 0.255, 0.277, 0.311, 0.361, 0.376,
                      0.395,
                  0.432, 0.463, 0.481, 0.519, 0.529, 0.567, 0.642, 0.674,
                      0.752,
                  0.823, 0.887, 0.926])
data2 = np.array([0.853, 0.759, 0.874, 0.800, 0.716, 0.557, 0.503, 0.399,
```

```
    0.334,
                    0.207, 0.118, 0.097, 0.078, 0.067, 0.056, 0.044, 0.036,
                        0.026,
                    0.019, 0.014, 0.010, 0.118])
# 最大似然估计函数
def mle_estimation(log_likelihood_func, initial_guess, data):
    result = optimize.minimize(log_likelihood_func, initial_guess, args=(data
        ,), method='L-BFGS-B',
                                bounds=[(1e-6, None), (1e-6, None)])
    params = result.x
    hessian_inv = result.hess_inv.todense()  # 提取逆Hessian矩阵用于估计标准
        误差
    se = np.sqrt(np.diag(hessian_inv)) if result.success else [np.nan] * len(
        params)
    return params, se
# 定义初始猜测值和分布
initial_guesses = {
    'UEXE': [1, 1],
    'Beta': [1, 1],
    'Kumaraswamy': [1, 1],
    'Unit Gamma': [1, 1],
    'Power Topp-Leone': [1, 1],
    'Bounded Weighted Exponential': [1, 1],
    'Unit Burr XII': [1, 1],
    'Unit Birnbaum-Saunders': [1, 1],
    'Unit Inverse Gaussian': [1, 1],
    'Unit Mirra': [1, 1],
}
distributions = {
    'UEXE': uexe_log_likelihood,
    'Beta': beta_log_likelihood,
    'Kumaraswamy': kumaraswamy_log_likelihood,
```

```
    'Unit␣Gamma': unit_gamma_log_likelihood,
    'Power␣Topp-Leone': power_topp_leone_log_likelihood,
    'Bounded␣Weighted␣Exponential':
        bounded_weighted_exponential_log_likelihood,
    'Unit␣Burr␣XII':  unit_burr_xii_log_likelihood ,
    'Unit␣Birnbaum-Saunders': unit_birnbaum_saunders_log_likelihood,
    'Unit␣Inverse␣Gaussian':  unit_inverse_gaussian_log_likelihood ,
    'Unit␣Mirra':  unit_mirra_log_likelihood ,
}
name = {
    'UEXE':'UEXE',
    'Beta': 'Beat',
    'Kumaraswamy': 'Kum',
    'Unit␣Gamma': 'UG',
    'Power␣Topp-Leone': 'PTL',
    'Bounded␣Weighted␣Exponential': 'BWE',
    'Unit␣Burr␣XII': 'UBXII',
    'Unit␣Birnbaum-Saunders':'UBS',
    'Unit␣Inverse␣Gaussian': 'UIG',
    'Unit␣Mirra': 'UM',
}
# A2 W KS 检验
def cdf_from_log_pdf(params, x, pdf_log_fun, lower_bound=1e-10):
    """
    通过对数PDF积分计算CDF。
    params: 参数列表 [delta, lambd]
    x: 计算CDF的点(可以是标量或 numpy 数组)
    lower_bound: 积分的下界 (通常为分布的起点)
    """
    delta, lambd = params
    # 如果 x 是标量, 将其转化为数组
    x = np.atleast_1d(x)
```

```
    # 用于存储结果的数组
    cdf_values = np.zeros_like(x)
    # 逐一计算每个点的CDF
    for i, xi in enumerate(x):
        if xi < lower_bound:  # 如果 x 小于下界，CDF 为 0
            cdf_values[i] = 0.0
        else:
            # 数值积分
            cdf_values[i], _ = quad(lambda t: np.exp(pdf_log_fun(params, t)),
                lower_bound, xi)
    return cdf_values if len(cdf_values) > 1 else cdf_values[0]
# 经验分布函数 (ECDF)
def ecdf(data):
    sorted_data = np.sort(data)  # 数据排序
    n = len(data)                # 数据点总数
    cdf = np.linspace(0, 1, n)  # 每个点的累计概率
    return sorted_data, cdf
# K-S 检验统计量
def ks_statistic(data, params, pdf_log_fun, lower_bound=1e-10):
    sorted_data, ecdf_values = ecdf(data)
    theory_cdf = cdf_from_log_pdf(params, sorted_data, pdf_log_fun,
        lower_bound)
    ks_stat = np.max(np.abs(ecdf_values - theory_cdf))
    j = np.linspace(1,1000,1000)
    temp = (-1)**(j-1)*np.exp(-2*len(data)*j**2*ks_stat**2)
    p = 2*temp.sum()
    return ks_stat,p
def anderson_darling_statistic(data, params, pdf_log_fun, lower_bound=1e-10):
    sorted_data, _ = ecdf(data)
    n = len(data)
    # 计算理论 CDF
    theory_cdf = cdf_from_log_pdf(params, sorted_data, pdf_log_fun,
```

```
        lower_bound)
    # 计算 A^2
    A2 = -n - np.sum((2 * np.arange(1, n+1) - 1) / n * (
        np.log(theory_cdf) + np.log(1 - np.flip(theory_cdf))
    ))
    return A2
def cramer_von_mises_statistic(data, params, pdf_log_fun, lower_bound=1e-10):
    sorted_data, ecdf_values = ecdf(data)
    n = len(data)
    # 计算理论 CDF
    theory_cdf = cdf_from_log_pdf(params, sorted_data, pdf_log_fun,
        lower_bound)
    # 计算 W^2
    W2 = np.sum((theory_cdf - ecdf_values) ** 2) / n
    return np.sqrt(W2)
result_data1 = {}
result_data2 = {}
for i,(dist_name, log_likelihood_func) in enumerate(distributions.items()):
    initial_guess = initial_guesses[dist_name]
    pdf_fun = globals()[f'{dist_name.lower().replace(" ","_").replace("-","_
        ")}_log_pdf']
    params, se = mle_estimation(log_likelihood_func, initial_guess, data1)
    A2 = anderson_darling_statistic(data1, params, pdf_fun)
    W = cramer_von_mises_statistic(data1, params, pdf_fun)
    ks,p = ks_statistic(data1, params, pdf_fun)
    result_data1[dist_name] = {'A2':A2, 'W*':W, 'ks':ks, 'p':p}
    params, se = mle_estimation(log_likelihood_func, initial_guess, data2)
    A2 = anderson_darling_statistic(data2, params, pdf_fun)
    W = cramer_von_mises_statistic(data2, params, pdf_fun)
    ks,p = ks_statistic(data2, params, pdf_fun)
    result_data2[dist_name] = {'A2':A2, 'W*':W, 'ks':ks, 'p':p}
df = pd.DataFrame.from_dict(result_data1, orient='index')
```

```
# 保存为 Excel 文件
df.to_excel("data1_awks.xlsx", index=True)
df = pd.DataFrame.from_dict(result_data2, orient='index')
# 保存为 Excel 文件
df.to_excel("data2_awks.xlsx", index=True)
```

Listing 3.14　真实数据应用可视化 Python 代码

```
import numpy as np
import scipy.optimize as optimize
from scipy.special import gamma
from scipy import stats
from statsmodels.distributions.empirical_distribution import ECDF
import matplotlib.pyplot as plt
from model import *
from scipy.integrate import quad
import random
# 数据集
data1 = np.array([0.023, 0.032, 0.054, 0.069, 0.081, 0.094, 0.105, 0.127,
    0.148,
                  0.169, 0.188, 0.216, 0.255, 0.277, 0.311, 0.361, 0.376,
                      0.395,
                  0.432, 0.463, 0.481, 0.519, 0.529, 0.567, 0.642, 0.674,
                      0.752,
                  0.823, 0.887, 0.926])
data2 = np.array([0.853, 0.759, 0.874, 0.800, 0.716, 0.557, 0.503, 0.399,
    0.334,
                  0.207, 0.118, 0.097, 0.078, 0.067, 0.056, 0.044, 0.036,
                      0.026,
                  0.019, 0.014, 0.010, 0.118])

# 最大似然估计函数
def mle_estimation(log_likelihood_func, initial_guess, data):
    result = optimize.minimize(log_likelihood_func, initial_guess, args=(data
```

```
        ,) , method='L-BFGS-B',
                                    bounds=[(1e-6, None), (1e-6, None)])
    params = result.x
    hessian_inv = result.hess_inv.todense()  # 提取逆Hessian矩阵用于估计标准
        误差
    se = np.sqrt(np.diag(hessian_inv)) if result.success else [np.nan] * len(
        params)
    return params, se
# 定义初始猜测值和分布
initial_guesses = {
    'UEXE': [1, 1],
    'Beta': [1, 1],
    'Kumaraswamy': [1, 1],
    'Unit Gamma': [1, 1],
    'Power Topp-Leone': [1, 1],
    'Bounded Weighted Exponential': [1, 1],
    'Unit Burr XII': [1, 1],
    'Unit Birnbaum-Saunders': [1, 1],
    'Unit Inverse Gaussian': [1, 1],
    'Unit Mirra': [1, 1],
}
distributions = {
    'UEXE': uexe_log_likelihood,
    'Beta': beta_log_likelihood ,
    'Kumaraswamy': kumaraswamy_log_likelihood,
    'Unit Gamma': unit_gamma_log_likelihood,
    'Power Topp-Leone': power_topp_leone_log_likelihood,
    'Bounded Weighted Exponential':
        bounded_weighted_exponential_log_likelihood,
    'Unit Burr XII': unit_burr_xii_log_likelihood ,
    'Unit Birnbaum-Saunders': unit_birnbaum_saunders_log_likelihood,
    'Unit Inverse Gaussian': unit_inverse_gaussian_log_likelihood ,
```

```
    'Unit␣Mirra':  unit_mirra_log_likelihood ,
}
name = {
    'UEXE':'UEXE',
    'Beta': 'Beat',
    'Kumaraswamy': 'Kum',
    'Unit␣Gamma': 'UG',
    'Power␣Topp-Leone': 'PTL',
    'Bounded␣Weighted␣Exponential': 'BWE',
    'Unit␣Burr␣XII': 'UBXII',
    'Unit␣Birnbaum-Saunders':'UBS',
    'Unit␣Inverse␣Gaussian': 'UIG',
    'Unit␣Mirra': 'UM',
}
# 绘制cdf
def cdf_from_log_pdf(params, x, pdf_log_fun, lower_bound=1e-10):
    """
    通过对数PDF积分计算CDF。
    params: 参数列表 [delta, lambd]
    x: 计算CDF的点(可以是标量或 numpy 数组)
    lower_bound: 积分的下界 (通常为分布的起点)
    """
    delta , lambd = params
    # 如果 x 是标量, 将其转化为数组
    x = np. atleast_1d (x)
    # 用于存储结果的数组
    cdf_values  = np. zeros_like (x)
    # 逐一计算每个点的CDF
    for i, xi in enumerate(x):
        if xi < lower_bound:  # 如果 x 小于下界, CDF 为 0
            cdf_values [i] = 0.0
        else :
```

```
            # 数值积分
            cdf_values[i], _ = quad(lambda t: np.exp(pdf_log_fun(params, t)),
                lower_bound, xi)
    return cdf_values if len(cdf_values) > 1 else cdf_values[0]
x_values = np.linspace(0, 1, 100)
x_values_add = np.linspace(1e-7, 1-1e-7, 100)
# 设置随机颜色和线条样式
line_styles = ['--', '-.', ':']
colors = plt.cm.get_cmap('tab10', len(distributions))
# 创建画布
plt.figure(figsize=(20, 26))
sorted_data = np.sort(data2)  # 数据排序
n = len(data2)                # 数据点总数
ecdf = np.linspace(0, 1, n)  # 每个点的累计概率
sorted_data = np.append(sorted_data, 1.0)  # 添加 1.0 到数据末尾
ecdf = np.append(ecdf, 1.0)
for i,(dist_name, log_likelihood_func) in enumerate(distributions.items()):
    # 随机选择颜色和线条样式
    color = colors(i)
    line_style = random.choice(line_styles)
    initial_guess = initial_guesses[dist_name]
    params, se = mle_estimation(log_likelihood_func, initial_guess, data2)
    pdf_fun = globals()[f'{dist_name.lower().replace(" ","_").replace("-","_")}_log_pdf']
    cdf_values = cdf_from_log_pdf(params, x_values_add,pdf_fun)
    if not isinstance(cdf_values, np.ndarray) or len(cdf_values) != 100:
        raise RuntimeError(
            f'{dist_name.lower().replace(" ","_").replace("-","_")}_log_pdf'
        )
    count= i+1
    if count % 2 == 0:
        plt.subplot(5, 4, 2*count)
```

```
    else:
        plt.subplot(5, 4, 2*count+1)
    if i == 0:
        plt.plot(x_values, cdf_values, label=f'{name[dist_name]}', color=color
            , linestyle='-', lw=2.5)
    else:
        plt.plot(x_values, cdf_values, label=f'{name[dist_name]}', color=color
            , linestyle = line_style, lw=2.5)
    plt.step(sorted_data, ecdf, where='post', color='gray', label='Kaplan-
        Meier')
    plt.xlabel('x')
    plt.ylabel('Estimated␣PDF')
    plt.xticks(np.arange(0, 1.1, 0.2))
    plt.xlim(0,1)
    plt.yticks(np.arange(0, 1.1, 0.2))
    plt.ylim(0,1.1)
    # plt.gca().set_aspect('equal', adjustable='box')
    plt.legend()
for i,(dist_name, log_likelihood_func) in enumerate(distributions.items()):
        # 随机选择颜色和线条样式
        color = colors(i)
        line_style = random.choice(line_styles)
        initial_guess = initial_guesses[dist_name]
        params, se = mle_estimation(log_likelihood_func, initial_guess, data2)
        # print(f'{dist_name.lower().replace(" ", "_").replace("-", "_")}
            _log_pdf')
        pdf_fun = globals()[f'{dist_name.lower().replace("␣","␣"_").replace
            ("-","␣"_")}_log_pdf']
        pdf_values = np.exp(pdf_fun(params, x_values_add))
        if not isinstance(pdf_values, np.ndarray) or len(pdf_values) != 100:
            raise RuntimeError(
                f'{dist_name.lower().replace("␣","␣"_").replace("-","␣"_")}
```

```
                _log_pdf'
            )
        count= i+1
        if count % 2 == 0:
            plt.subplot(5, 4, 2*count-2)
        else:
            plt.subplot(5, 4, 2*count-1)
        if i == 0 :
            plt.plot(x_values, pdf_values, label=f'{name[dist_name]}', color=
                color, linestyle ='-')
        else:
            plt.plot(x_values, pdf_values, label=f'{name[dist_name]}', color=
                color, linestyle = line_style )
        plt.hist(data2, bins=np.linspace(0, 1, 6), density=True, alpha=1,
            color='white', edgecolor='black')
        plt.xlabel('x')
        plt.ylabel('Estimated PDF')
        plt.xticks(np.arange(0, 1.1, 0.2))
        plt.xlim(0,1)
        if i == 9:
            plt.yticks(np.arange(0, 10.2, 2))
            plt.ylim(0,10.1)
        else:
            plt.ylim(0, 3.1)
            plt.yticks(np.arange(0, 3.1, 0.5))
        # plt.gca().set_aspect('equal', adjustable='box')
        plt.legend()
plt.tight_layout()
plt.show()
```

Listing 3.15 Bootstrap 重抽样 Python 代码

```
import numpy as np
import scipy.optimize as optimize
```

```
from scipy.special import gamma
from scipy import stats
import random
from tqdm import tqdm
from statsmodels.distributions.empirical_distribution import ECDF
import pandas as pd
# 数据集
data1 = np.array([0.023, 0.032, 0.054, 0.069, 0.081, 0.094, 0.105, 0.127,
    0.148,
                  0.169, 0.188, 0.216, 0.255, 0.277, 0.311, 0.361, 0.376,
                      0.395,
                  0.432, 0.463, 0.481, 0.519, 0.529, 0.567, 0.642, 0.674,
                      0.752,
                  0.823, 0.887, 0.926])
data2 = np.array([0.853, 0.759, 0.874, 0.800, 0.716, 0.557, 0.503, 0.399,
    0.334,
                  0.207, 0.118, 0.097, 0.078, 0.067, 0.056, 0.044, 0.036,
                      0.026,
                  0.019, 0.014, 0.010, 0.118])
def uexe_log_likelihood(params, data):
    delta, lambd = params
    # 检查参数范围
    if delta <= 0 or lambd <= 0:
        return np.inf  # 参数或数据越界时，返回正无穷作为惩罚
    # 计算负对数似然值
    log_pdf = (
        2 * np.log(delta)
        + np.log(1 - lambd * np.log(data))
        - np.log(delta + lambd)
        + (delta - 1) * np.log(data)
    )
    return -np.sum(log_pdf)
```

```
def beta_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(stats.beta.logpdf(data, b, a))
def kumaraswamy_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(b) + np.log(a) + (b - 1) * np.log(data) + (a - 1) *
        np.log(1 - data**b))
def unit_gamma_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    log_pdf = (
        np.log(b)
        + (b-1) * np.log(data)
        + (a-1) * np.log((-b) * np.log(data))
        - np.log(gamma(a))
    )
    return -np.sum(log_pdf)
def power_topp_leone_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(2) + np.log(a) + np.log(b) + (a * b - 1) * np.log(
        data) + np.log(1 - data**a) + (b-1) * np.log(2 - data**a))
def bounded_weighted_exponential_log_likelihood(params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
```

```
    return -np.sum(np.log(a+1) + np.log(b) - np.log(a) + (b-1) * np.log(data)
        + np.log(1 - data**(a*b)))
def unit_burr_xii_log_likelihood (params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    return -np.sum(np.log(a) + np.log(b) - (1+a) * np.log(1 + (-np.log(data))
        **b) - np.log(data) - (1-b) * np.log(-np.log(data)))
def unit_birnbaum_saunders_log_likelihood (params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    first = np.sqrt(-b/np.log(data)) + np.sqrt((-b/np.log(data))**3)
    second1 = b/np.log(data)
    second2 = np.log(data)/b
    second = (second1 + second2 + 2)/(a**2)
    second = second/2
    third = 2 * a * b * data * np.sqrt(2*np.pi)
    log_pdf = (
            np.log( first )
            + second
            - np.log( third )
    )
    return -np.sum(log_pdf)
def unit_inverse_gaussian_log_likelihood (params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    second_1 = a * ((np.log(data) + b)**2)
    second_2 = 2 * np.log(data) * (b**2)
    second = second_1 / second_2
    c = (np.log(data))**2
```

```
    log_pdf = (
            np.log(a)
            + second
            - 0.5 * np.log(2 * np.pi)
            - np.log(data)
            - np.log(c)
            - 0.5 * np.log(-a/np.log(data))
    )
    return -np.sum(log_pdf)
def unit_mirra_log_likelihood (params, data):
    a, b = params
    if a <= 0 or b <= 0:
        return np.inf
    log_pdf = (
            3 * np.log(b)
            + np.log(a + (a+2) * (data**2) - 2 * a * data)
            + b
            - b/data
            - np.log(2 * (data**4))
            - np.log(a + (b**2))
    )
    return -np.sum(log_pdf)
# 最大似然估计函数
def mle_estimation(log_likelihood_func, initial_guess, data):
    result = optimize.minimize(log_likelihood_func, initial_guess, args=(data
        ,), method='L-BFGS-B',
                               bounds=[(1e-6, None), (1e-6, None)])
    params = result.x
    hessian_inv = result.hess_inv.todense()  # 提取逆Hessian矩阵用于估计标准
        误差
    se = np.sqrt(np.diag(hessian_inv)) if result.success else [np.nan] * len(
        params)
```

```
    return params, se
# 定义初始猜测值和分布
initial_guesses = {
    'UEXE': [1, 1],
    'Beta': [1, 1],
    'Kumaraswamy': [1, 1],
    'Unit Gamma': [1, 1],
    'Power Topp-Leone': [1, 1],
    'Bounded Weighted Exponential': [1, 1],
    'Unit Burr XII': [1, 1],
    'Unit Birnbaum-Saunders': [1, 1],
    'Unit Inverse Gaussian': [1, 1],
    'Unit Mirra': [1, 1],
}
distributions = {
    'UEXE': uexe_log_likelihood,
    'Beta': beta_log_likelihood,
    'Kumaraswamy': kumaraswamy_log_likelihood,
    'Unit Gamma': unit_gamma_log_likelihood,
    'Power Topp-Leone': power_topp_leone_log_likelihood,
    'Bounded Weighted Exponential':
        bounded_weighted_exponential_log_likelihood,
    'Unit Burr XII': unit_burr_xii_log_likelihood,
    'Unit Birnbaum-Saunders': unit_birnbaum_saunders_log_likelihood,
    'Unit Inverse Gaussian': unit_inverse_gaussian_log_likelihood,
    'Unit Mirra': unit_mirra_log_likelihood,
}
if __name__ == "__main__":
    M = 1000
    # boot_strap data1
    results_data1 = {}
    for dist_name, log_likelihood_func in distributions.items():
```

```
    initial_guess = initial_guesses[dist_name]
    ori_params, _ = mle_estimation(log_likelihood_func, initial_guess,
        data1)
    boot_out = []
    for _ in range(M):
        data_sample = np.array(random.sample(list(data1), len(data1)))
        params, se = mle_estimation(log_likelihood_func, initial_guess,
            data_sample)
        boot_out.append(params)
    boot_out = np.array(boot_out)
    first_boot = boot_out[:,0]
    sec_boot = boot_out[:,-1]
    first_boot_bias = (first_boot - ori_params[0]).mean()
    sec_boot_bias = (sec_boot - ori_params[1]).mean()
    first_boot_var = first_boot.var()
    sec_boot_var = sec_boot.var()
    results_data1[dist_name] = {'first_param_bias': first_boot_bias, '
        sec_param_bias': sec_boot_bias,
                                'first_param_var': first_boot_var, '
                                    sec_param_var': sec_boot_var}
# 转化为 DataFrame
df = pd.DataFrame.from_dict(results_data1, orient='index')
# 保存为 Excel 文件
df.to_excel("data1_bootstrap.xlsx", index=True)
# boot_strap data2
results_data2 = {}
for dist_name, log_likelihood_func in distributions.items():
    initial_guess = initial_guesses[dist_name]
    ori_params, _ = mle_estimation(log_likelihood_func, initial_guess,
        data2)
    boot_out = []
    for _ in range(M):
```

```
            data_sample = np.array(random.sample(list(data2), len(data2)))
            params, se = mle_estimation(log_likelihood_func, initial_guess,
                data_sample)
            boot_out.append(params)
        boot_out = np.array(boot_out)
        first_boot = boot_out[:,0]
        sec_boot = boot_out[:,-1]
        first_boot_bias = (first_boot - ori_params[0]).mean()
        sec_boot_bias = (sec_boot - ori_params[1]).mean()
        first_boot_var = first_boot.var()
        sec_boot_var = sec_boot.var()
        results_data2[dist_name] = {'first_param_bias': first_boot_bias, '
            sec_param_bias': sec_boot_bias,
                                    'first_param_var': first_boot_var, '
                                        sec_param_var': sec_boot_var}
    # 转化为 DataFrame
    df = pd.DataFrame.from_dict(results_data2, orient='index')
    # 保存为 Excel 文件
    df.to_excel("data2_bootstrap.xlsx", index=True)
```

Listing 3.16　泰迪杯数据应用 Python 代码

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as optimize
from model import uexe_log_likelihood, uexe_log_pdf
from scipy.integrate import quad
df = pd.read_excel('result1_4.xlsx')
data1 = np.array(df['data1'])
data2 = np.array(df['data2'])
for i in range(len(data1)):
    if data1[i] == 0:
        data1[i] += 1e-8
```

```
    elif data1[i] == 1:
        data1[i] -= 1e-8

    if data2[i] == 0:
        data2[i] += 1e-8
    elif data2[i] == 1:
        data2[i] -= 1e-8
# 经验分布函数(ECDF)
def ecdf(data):
    sorted_data = np.sort(data)  # 数据排序
    n = len(data)                # 数据点总数
    cdf = np.linspace(0, 1, n)  # 每个点的累计概率
    return sorted_data, cdf
# 最大似然估计函数
def mle_estimation(log_likelihood_func, initial_guess, data):
    result = optimize.minimize(log_likelihood_func, initial_guess, args=(data
        ,), method='L-BFGS-B',
                               bounds=[(1e-6, None), (1e-6, None)])
    params = result.x
    hessian_inv = result.hess_inv.todense()  # 提取逆Hessian矩阵用于估计标准
        误差
    se = np.sqrt(np.diag(hessian_inv)) if result.success else [np.nan] * len(
        params)
    return params, se
x_values = np.linspace(0, 1, 100)
x_values_add = np.linspace(1e-7, 1-1e-7, 100)
plt.figure(figsize=(10, 5))
# 打乱数据的索引
np.random.seed(42)
shuffled_indices = np.random.permutation(len(data1))
# 划分训练集和验证集
train_size = 250
```

```
train_indices = shuffled_indices[:train_size]
val_indices = shuffled_indices[train_size:]
train_data1 = data1[train_indices]
val_data1 = data1[val_indices]
train_data2 = data2[train_indices]
val_data2 = data2[val_indices]
# 使用这些索引来获取训练集和验证集
params, se = mle_estimation(uexe_log_likelihood, [1, 1], train_data1)
pdf_values = np.exp(uexe_log_pdf(params, x_values_add))
plt.subplot(1, 2, 1)
plt.plot(x_values, pdf_values, label='UEXE', color='purple', linestyle='-')
plt.hist(val_data1, bins=np.linspace(0, 1, 10), density=True, alpha=1, color='
    white', edgecolor='black', label='Val')
plt.hist(train_data1, bins=np.linspace(0, 1, 10), density=True, alpha=0.2,
    color='gray', edgecolor='gray', label='Train')
plt.xlabel('data1')
plt.ylabel('Estimated␣PDF')
plt.legend()
plt.subplot(1, 2, 2)
# 使用这些索引来获取训练集和验证集
params, se = mle_estimation(uexe_log_likelihood, [1, 1], train_data2)
pdf_values = np.exp(uexe_log_pdf(params, x_values_add))
plt.plot(x_values, pdf_values, label='UEXE', color='purple', linestyle='-')
plt.hist(val_data2, bins=np.linspace(0, 1, 10), density=True, alpha=1, color='
    white', edgecolor='black', label='Val')
plt.hist(train_data2, bins=np.linspace(0, 1, 10), density=True, alpha=0.2,
    color='gray', edgecolor='gray', label='Train')
plt.xlabel('data2')
plt.ylabel('Estimated␣PDF')
plt.legend()
plt.tight_layout()
plt.show()
```

```
def ecdf_add(data):
    sorted_data = np.sort(data)  # 数据排序
    n = len(data)                # 数据点总数
    cdf = np.linspace(0, 1, n)  # 每个点的累计概率
    sorted_data = np.append(sorted_data, 1.0)  # 添加 1.0 到数据末尾
    cdf = np.append(cdf, 1.0)
    return sorted_data, cdf
def ecdf(data):
    sorted_data = np.sort(data)  # 数据排序
    n = len(data)                # 数据点总数
    cdf = np.linspace(0, 1, n)  # 每个点的累计概率
    return sorted_data, cdf
def cdf_from_log_pdf(params, x, pdf_log_fun, lower_bound=1e-10):
    """
    通过对数PDF积分计算CDF。
    params: 参数列表 [delta, lambd]
    x: 计算CDF的点(可以是标量或 numpy 数组)
    lower_bound: 积分的下界（通常为分布的起点）
    """
    delta, lambd = params

    # 如果 x 是标量，将其转化为数组
    x = np.atleast_1d(x)

    # 用于存储结果的数组
    cdf_values = np.zeros_like(x)

    # 逐一计算每个点的CDF
    for i, xi in enumerate(x):
        if xi < lower_bound:  # 如果 x 小于下界，CDF 为 0
            cdf_values[i] = 0.0
```

```
            else:
                # 数值积分
                cdf_values[i], _ = quad(lambda t: np.exp(pdf_log_fun(params, t)),
                    lower_bound, xi)

    return cdf_values if len(cdf_values) > 1 else cdf_values[0]
# K-S 检验统计量
def ks_statistic(data, params, pdf_log_fun, lower_bound=1e-10):
    sorted_data, ecdf_values = ecdf(data)
    theory_cdf = cdf_from_log_pdf(params, sorted_data, pdf_log_fun,
        lower_bound)
    ks_stat = np.max(np.abs(ecdf_values - theory_cdf))
    j = np.linspace(1,1000,1000)
    temp = (-1)**(j-1)*np.exp(-2*len(data)*j**2*ks_stat**2)
    p = 2*temp.sum()
    return ks_stat,p
def anderson_darling_statistic(data, params, pdf_log_fun, lower_bound=1e-10):
    sorted_data, _ = ecdf(data)
    n = len(data)
    # 计算理论 CDF
    theory_cdf = cdf_from_log_pdf(params, sorted_data, pdf_log_fun,
        lower_bound)
    print(len(theory_cdf))
    print(len(sorted_data))
    print(len(data))
    # 计算 A^2
    A2 = -n - np.sum((2 * np.arange(1, n+1) - 1) / n * (
        np.log(theory_cdf) + np.log(1 - np.flip(theory_cdf))
    ))
    return A2
def cramer_von_mises_statistic(data, params, pdf_log_fun, lower_bound=1e-10):
    sorted_data, ecdf_values = ecdf(data)
```

```
    n = len(data)
    # 计算理论 CDF
    theory_cdf = cdf_from_log_pdf(params, sorted_data, pdf_log_fun,
        lower_bound)
    # 计算 W^2
    W2 = np.sum((theory_cdf - ecdf_values) ** 2) / n
    return np.sqrt(W2)
result = {}
# 打乱数据的索引
np.random.seed(42)
shuffled_indices = np.random.permutation(len(data1))
# 划分训练集和验证集
train_size = 250
train_indices = shuffled_indices[:train_size]
val_indices = shuffled_indices[train_size:]
train_data1 = data1[train_indices]
val_data1 = data1[val_indices]
np.random.seed(32)
shuffled_indices = np.random.permutation(len(data1))
# 划分训练集和验证集
train_size = 250
train_indices = shuffled_indices[:train_size]
val_indices = shuffled_indices[train_size:]
train_data2 = data2[train_indices]
val_data2 = data2[val_indices]
x_values = np.linspace(0, 1, 100)
x_values_add = np.linspace(1e-7, 1-1e-7, 100)
plt.figure(figsize=(10, 5))
params, se = mle_estimation(uexe_log_likelihood, [1, 1], train_data1)
cdf_values = cdf_from_log_pdf(params, x_values_add,uexe_log_pdf)
A2 = anderson_darling_statistic(val_data1, params, uexe_log_pdf)
W = cramer_von_mises_statistic(val_data1, params, uexe_log_pdf)
```

```
ks,p = ks_statistic (val_data1, params, uexe_log_pdf)
result['data1'] = {'params1':params[0],'params2':params[1], 'A2':A2, 'W*':W,
    'ks':ks, 'p':p}
plt.subplot(1, 2, 1)
plt.plot(x_values, cdf_values, label='UEXE', color='purple', linestyle='-')
sorted_data, cdf = ecdf_add(val_data1)
plt.step(sorted_data, cdf, where='post', alpha=0.5, color='r', label='Kaplan-
    Meier:val')
sorted_data, cdf = ecdf_add(train_data1)
plt.step(sorted_data, cdf, where='post', color='gray', label='Kaplan-Meier:
    Train')
plt.xlabel('data1')
plt.ylabel('Estimated CDF')
plt.legend()
params, se = mle_estimation(uexe_log_likelihood, [1, 1], train_data2)
cdf_values = cdf_from_log_pdf(params, x_values_add,uexe_log_pdf)
A2 = anderson_darling_statistic (val_data2, params, uexe_log_pdf)
W = cramer_von_mises_statistic(val_data2, params, uexe_log_pdf)
ks,p = ks_statistic (val_data2, params, uexe_log_pdf)
result['data2'] = {'params1':params[0],'params2':params[1], 'A2':A2, 'W*':W,
    'ks':ks, 'p':p}
plt.subplot(1, 2, 2)
plt.plot(x_values, cdf_values, label='UEXE', color='purple', linestyle='-')
sorted_data, cdf = ecdf_add(val_data2)
plt.step(sorted_data, cdf, where='post', alpha=0.5, color='r', label='Kaplan-
    Meier:val')
sorted_data, cdf = ecdf_add(train_data2)
plt.step(sorted_data, cdf, where='post', color='gray', label='Kaplan-Meier:
    Train')
plt.xlabel('data2')
plt.ylabel('Estimated CDF')
plt.legend()
```

```
plt . tight_layout ()
plt .show()
df = pd.DataFrame.from_dict( result , orient ='index' )
# 保存为 Excel 文件
df . to_excel (" real_data . xlsx", index=True)
```