中国计量测试学会团体标准

《免疫分析 测量结果的不确定度评定》

编制说明

一．任务来源

免疫分析是一种基于抗原与抗体特异性结合反应的生物化学检测技术。其核心原理是利用抗体与抗原之间的高亲和力和特异性，通过标记技术（如酶、荧光、化学发光等）实现对目标物质的定性和定量检测。常见的免疫分析技术包括酶联免疫吸附试验（ELISA）、荧光免疫分析（FIA）和化学发光免疫分析（CLIA）等。免疫分析已经广泛的应用于各个检测领域，在临床诊断中用于检测疾病标志物，如肿瘤标志物、激素和病毒抗原等。在食品安全领域，它用于检测残留农药、毒素和过敏原。在环境监测中，免疫分析用于检测污染物，如重金属和有机污染物。此外，该技术还广泛应用于药物研发、法医学和基础科学研究。标准曲线是免疫分析定量检测的基础，通常通过已知浓度的标准品绘制。通过仪器读取标准溶液的仪器响应信号，如吸光度、荧光强度或化学发光强度，再使用数学模型（如线性回归或四参数逻辑回归）拟合标准曲线。然后，根据样品的信号值，通过标准曲线方程计算目标物浓度。

免疫分析的标准曲线通常呈现S型特征，反映了抗原与抗体结合反应的动态平衡过程。在低浓度区域，信号值随抗原浓度增加而缓慢上

升；在中间浓度区域，信号值变化显著，呈近似线性关系；而在高浓度区域，信号值趋于饱和，变化逐渐平缓。这种非线性关系使得标准曲线的拟合通常需要使用非线性模型，如四参数逻辑回归（4PL）或三次样条插值拟合，以更准确地描述数据分布。此外，免疫分析中使用的标准溶液浓度和仪器响应值均具有不可忽略的不确定度，这意味着数据点的x（浓度）和y（信号值）均存在误差。因此，在进行曲线拟合时，需要采用加权回归等方法，充分考虑x和y方向的不确定度，以提高拟合结果的准确性和可靠性。现有的不确定度评定方法主要针对线性拟合，同时在计算过程中假设了标准溶液标准值的不确定度可以忽略不计，即x的不确定度$u(x)$可以忽略不计，这与免疫分析的实际情况并不相符，因此需要考虑在非线性拟合并需要同时考虑x和y的不确定度时如何评定试样结果的不确定度。

基于上述背景，中国计量科学研究院牵头在2024年向中国计量测试学会申请立项《免疫分析 测量结果的不确定度评定》团体标准，由中国计量测试学会批准立项，文件号量学发【2024】193号。

二．国内外现状

国内外不确定度评定方法主要有三类，主要包括传统的GUM评定方法、蒙特卡罗评定方法以及贝叶斯方法。

2.1 GUM方法

GUM（《测量不确定度表示指南》）是国际通用的不确定度评定标准，由ISO等国际组织于1993年正式发布，其核心是通过数学模型量化测量结果的分散性。GUM法以线性模型和泰勒展开为基础，将不确定度分为A类（统计方法评估）和B类（非统计方法评估），通过合成标准不确定度与扩展不确定度形成完整的评定流程。GUM

不确定度评定方法被广泛的应用于各种测量领域。

然而，GUM法在非线性模型、输入量非正态分布或高相关性场景中存在明显局限。例如，动态检测或复杂系统的不确定度评定常因模型线性假设失效而产生偏差。为此，国际计量委员会（JCGM）于2008年发布GUM-S1补充文件，引入蒙特卡洛方法（MCM）作为扩展，但仍需依赖离线数据处理，难以满足实时性需求。国内研究虽在GUM推广上成效显著（如CNAS-GL10指南的应用），但在智能算法融合和复杂系统评定方面仍落后于国际水平，部分领域（如化学分析中的采样制样不确定度）尚未形成统一标准。

2.2 蒙特卡洛方法（MCM）

蒙特卡洛方法（MCM）通过概率分布随机抽样模拟不确定度传播，突破了GUM法对模型线性和分布类型的限制。其核心步骤包括概率过程建模、随机数生成与统计量估计。国际研究更注重MCM与智能技术的结合。例如，GUM-S1将MCM纳入标准化流程，支持复杂分布和高阶效应的评定。而国内研究多集中于算法优化与工具开发，如开发基于MCM的专用软件，并在GPS测量、动态检测等领域取得进展。然而，MCM的高计算成本限制了其在大规模数据或实时场景中的应用。例如，时间序列数据的在线不确定度评定需依赖数据分解和动态更新策略，传统MCM难以直接适配。此外，国内在MCM的理论创新（如自适应抽样算法）和跨学科应用（如生物医学）方面仍需加强。

2.3 贝叶斯方法

贝叶斯方法通过融合先验知识与观测数据实现不确定度的动态更新，尤其适用于复杂系统和实时场景。其核心是利用后验分布量

化参数不确定性，例如在时间序列分析中，结合STL分解将数据拆分为趋势项和随机项，再通过贝叶斯推断在线更新不确定度。国际上，贝叶斯方法已拓展至多领域：在病原体进化研究中，SeedbankTree框架通过贝叶斯系统发育动力学推断休眠种群的突变率差异，揭示了结核分枝杆菌潜伏感染机制；在癌症基因组学中，贝叶斯多研究非负矩阵分解（NMF）联合分析多数据集，识别结直肠癌的共享突变特征，并关联年龄、性别等协变量效应。

国内研究在贝叶斯理论应用上呈现两大方向：一是改进传统评定流程，如基于模糊贝叶斯的B类不确定度评定，通过模糊集处理主观认知偏差，提升可操作性；二是结合人工智能，如万年峰课题组开发的VB-RobM模型，融合贝叶斯推断与结构方程，提升生态数据路径分析的鲁棒性和计算效率。然而，国内在贝叶斯方法的实际落地（如临床实验室标准化）和跨学科深度整合（如工业机器人终身学习）方面仍存在差距。例如，免疫分析的不确定度评定在国内仍以误差理论为主，而国外已通过贝叶斯优化标记技术（如时间分辨荧光TRFIA）降低试剂批间差异。

GUM、MCM与贝叶斯方法分别代表了不确定度评定的经典、仿真与动态范式。国际研究注重方法创新与实际场景融合（如病原体进化、机器人终身学习），而国内则在算法优化与局部应用（如工程材料、微生物检测）上进展显著。三者互补性强：GUM提供基础框架，MCM解决复杂模型，贝叶斯实现动态更新。未来需进一步推动多方法协同、智能化工具开发及跨学科标准化，以应对高维数据、实时监测等新兴挑战。

免疫分析是一种基于抗体和抗原之间特异性相互作用的生化分

析技术，用于定性和定量检测样本中的特定靶标（如小分子、蛋白质、激素、细胞、病原体等）。该技术利用抗体高度的特异性和结合能力识别并结合到靶标上，通过各种信号转换方法将这种结合事件转化为可测量的信号，从而实现对靶标的定性和定量检测。免疫分析具有高度的特异性、灵敏度和适用性，因此广泛的应用于生命科学的基础研究、医学检验、药物研发、治疗监测、食品安全、生物农业等领域。仅酶联免疫分析方法，我国已经颁布相关检测方法国家标准17项、行业标准64项、地方标准87项、团体标准18项，涵盖医学检验、食品分析、进出口检验检疫、农产品检验、纺织品检验、生物产制品检验、生物学基础研究等领域。免疫分析结果的不确定度评定近年来在干扰因素识别与消除方面取得显著进展。传统不确定度评定方法（如GUM法）在免疫分析中面临非线性模型和动态检测场景的挑战，而蒙特卡洛方法（MCM）和贝叶斯统计的应用为复杂系统提供了新思路。MCM通过概率分布随机抽样模拟不确定度传播，在免疫分析中已用于评估多步骤检测（如样本稀释、酶促反应）的累积误差。贝叶斯方法则通过融合先验知识（如试剂稳定性数据）与实时观测数据，动态更新不确定度估计。人工智能技术的引入进一步推动了不确定度评定的智能化。例如，基于机器学习的空间免疫评分系统（如TIMES模型）通过分析肿瘤微环境中免疫细胞的空间分布特征，量化了肝癌复发预测中的生物学异质性，其不确定度评定不仅涵盖技术误差，还整合了免疫微环境动态变化的复杂性。此外，自动化流水线在免疫检测中的应用减少了人工操作引入的随机误差，但需通过生物安全风险评估和实时质控数据反馈优化不确定度模型。

在化学分析中，最小二乘线性拟合是一种常用的数据处理方法，用于建立标准曲线并计算待测样品的浓度。该方法通常假设x轴（通常是标准溶液的浓度）上的不确定度可以忽略不计，而y轴（通常是测量信号）的不确定度是主要的误差来源。这种假设在许多情况下是合理的，因为标准溶液的制备通常较为精确，不确定度较小。然而，在生物测量领域，特别是涉及大分子（如蛋白质、核酸等）的测量时，标准溶液的不确定度往往较大，无法忽略。因此，假设x轴上的不确定度可以忽略不计，在生物测量中往往与实际情况不符。这可能导致拟合结果的不确定度被低估，进而影响测量结果的准确性和可靠性。

对于非线性拟合的不确定度评定，目前缺乏明确的规范和指导。非线性拟合在化学和生物测量中应用广泛，特别是在涉及复杂反应动力学或剂量-响应关系的情况下。然而，与线性拟合相比，非线性拟合的不确定度评定更为复杂，因为它不仅涉及测量误差的传播，还涉及模型选择和参数估计的不确定性。在实际应用中，非线性拟合的不确定度评定通常依赖于蒙特卡罗模拟或Bootstrap方法。这些方法通过重复采样和拟合过程，估计参数的不确定度和置信区间。然而，这些方法的计算成本较高，且对模型的假设和参数的选择较为敏感。因此，如何在不增加计算复杂性的情况下，准确评定非线性拟合的不确定度，仍然是一个亟待解决的问题。此外，非线性拟合的不确定度评定还面临模型选择的问题。不同的非线性模型可能对同一数据集给出不同的拟合结果，进而影响不确定度的评定。因此，如何选择合适的模型，并在模型选择过程中考虑不确定度的影响，也是一个重要的研究方向。

总之，当前化学分析中最小二乘线性拟合不确定度的假设在生物测量中可能不再适用，特别是涉及大分子标准溶液的情况。同时，非线性拟合的不确定度评定缺乏明确的规范和指导，需要进一步的研究和探索。

三．规范制定过程

1. 立项批准：2024 年 8 月，中国计量科学研究院收到中国计量测试学会下发的量学发函〔2024〕193号文件，批准《免疫分析 测量结果的不确定度评定》正式立项。

2. 组建编制组：2024 年 9 月，中国计量科学研究院联合相关单位组建了《免疫分析 测量结果的不确定度评定》编制组，包括深圳市新产业生物医学工程股份有限公司、贝克曼库尔特公司、北京义翘神州科技股份有限公司 (中国)、山东立菲生物产业有限公司等参与起草。

3. 首次会议：2024 年 9 月 23 日，中国计量科学研究院组织召开首次会议，讨论标准包含的内容、主要技术指标等问题， 商定标准起草的主要思路和起草原则。

4. 第二次会议：2025 年 5 月 12 日，中国计量科学研究员组织召开第二次会议，对首次会议后完成的草稿进行讨论，对标准草案逐条进行研读推敲，形成规范征求意见稿。

5. 标准验证：2025年 5 月~2025 年 6 月，编制组依据《免疫分析 测量结果的不确定度评定》，对企业提供的数据进行了实验测试，验证了标准的适用性。

四．编制依据

为了确保全球范围内的测量结果的一致性和可比性，国际上要求校

准和检测实验室评定测量结果的不确定度，并且将该要求写入了ISO/IEC 17025:2017《检测和校准实验室能力的通用要求》、ISO 15189:2022《医学实验室 质量和能力的要求》等国际标准,同时为了支持这些标准的实施和实验室认可的顺利进行，我国等同采用了上述国际标准，形成了GB/T 27025-2019《检测和校准实验室能力的通用要求》、GB/T 22576.1-2018《医学实验室 质量和能力的要求 第1部分：通用要求》等国家标准，以及CNAS-CL01《检测和校准实验室能力认可准则》等，都对测量结果的不确定度评定提出了要求。

在制定《免疫分析 测量结果的不确定度评定》的过程中，除了上述提到的法律法规和管理办法，还重点参考了以下规范和标准：

1. ISO Guide 98-3:2008《测量不确定度的评定与表示》：测量不确定度评定的国际通用标准，详细规定了不确定度评定的基本概念、方法和表示方式。它基于GUM（测量不确定度表示指南）框架，系统描述了不确定度来源的识别、量化、传播（如使用线性传播或蒙特卡罗模拟）以及结果的报告格式。该标准强调不确定度评定的科学性和规范性，适用于各类测量领域。对于制定免疫分析结果不确定度评定标准，ISO Guide 98-3:2008提供了基础理论和方法论支持，特别是在不确定度传播和量化方面。它帮助免疫分析领域建立统一、规范的不确定度评定流程，确保结果的准确性和可比性，同时为免疫分析中复杂问题（如非线性校准、基质效应）的评定提供了科学依据。

2. ILAC G8:09/2019《实验室测量不确定度的评定与报告指南》：国际实验室认可合作组织（ILAC）发布的技术指南，旨在为实验室提供测量不确定度评定和报告的规范化方法。它特别强调不确定度评定应结合具体测量方法的特点，并提供了实际案例以指导实践。

对于制定免疫分析结果不确定度评定标准，ILAC G8:09/2019提供了重要的实践框架和技术支持。

3. GB/T 27418-2017《测量不确定度评定与表示》是中华人民共和国国家标准，主要规定了测量不确定度的评定方法和表示形式。该标准详细阐述了不确定度的来源、评定步骤、数学模型建立、不确定度分量的量化以及合成不确定度和扩展不确定度的计算方法。它为测量结果的可靠性提供了科学依据，确保测量数据的准确性和可比性。在制定免疫分析结果不确定度评定标准时，GB/T 27418-2017提供了理论基础和技术支持，帮助明确不确定度的来源和评定流程，确保免疫分析结果的科学性和可重复性，从而提升检测结果的可靠性和临床应用价值。

4. GB/T 22553-2023《利用重复性、再现性和正确度的估计值评定测量不确定度的指南》提供了利用重复性、再现性和正确度的估计值来评定测量不确定度的方法和步骤。

5. JJF 1059.1-2012《测量不确定度评定与表示》是中国国家计量技术规范，详细规定了测量不确定度的评定方法和表示方式。该标准基于国际指南ISO/IEC Guide 98-3（GUM），涵盖了不确定度的来源分析、评定步骤、合成方法及报告格式，为各类测量活动提供了统一的技术依据。

6. JJF 1059.2-2012《用蒙特卡洛法评定测量不确定度》详细介绍了蒙特卡洛法在测量不确定度评定中的应用。该标准通过随机模拟方法，利用概率分布模型生成大量随机样本，计算测量结果的分布特征，从而评估测量不确定度。其优势在于能够处理复杂的非线性模型和分布不对称的情况，提供更精确的不确定度估计。对于免疫分析结果的不确定度评定，该标准提供了重要的方法论支撑，尤其

适用于免疫分析中复杂的测量模型和影响因素。通过蒙特卡洛法，可以更准确地量化免疫分析结果的不确定度，提升结果的可靠性和可比性，为制定相关标准提供科学依据。

7. 其他相关技术报告：总体最小二乘法（Total Least Squares，TLS）、马尔可夫链蒙特卡洛（Markov Chain Monte Carlo，MCMC）、贝叶斯方法相关技术文献。TLS是一种用于回归分析的技术，适用于x和y方向均存在不确定度的情况。与普通最小二乘法（OLS）仅考虑y方向误差不同，TLS同时考虑x和y方向的误差，通过最小化点到拟合直线的垂直距离来优化模型。TLS在处理非线性问题时，可以通过迭代或线性化方法扩展其应用范围。MCMC是一种基于概率分布的随机采样方法，适用于复杂非线性模型和高维不确定度分析。通过构建马尔可夫链，MCMC可以从后验分布中生成样本，用于估计参数及其不确定度。在处理x和y方向均具有不确定度的问题时，MCMC可以灵活地整合多维不确定度信息，提供更全面的分析结果。贝叶斯方法结合先验信息和观测数据，通过后验分布估计参数及其不确定度。在处理非线性问题时，贝叶斯方法可以结合MCMC等技术，有效处理多维不确定度，尤其适用于免疫分析等复杂系统的建模。

通过以上法律法规和标准的编制依据，确保了《免疫分析 测量结果的不确定度评定》的科学性、实用性和权威性， 为科学合理评定免疫分析结果的不确定度提供计量支撑。

五．主要技术内容

1. 适用范围

本文件规定了采用马尔可夫链蒙特卡罗法确定线性拟合、四参数模型拟合以及三次样条插值拟合确定免疫分析结果及其不确定度

的评定方法。本文件适用于标准溶液和仪器响应均存在不可忽略的不确定度时线性拟合、四参数模型拟合以及三次样条插值拟合确定免疫分析结果及其不确定度。本文件不适用于医学检验实验室免疫分析结果的不确定度评定。

2. 主要技术内容

(1) 马尔可夫链蒙特卡罗法（MCMC）的基本原理与应用

本文件详细介绍了马尔可夫链蒙特卡罗法（MCMC）的基本原理及其在免疫分析结果不确定度评定中的应用。MCMC通过构建马尔可夫链，从后验分布中生成随机样本，用于估计模型参数及其不确定度。该方法特别适用于处理非线性模型和多维不确定度问题，能够有效整合标准溶液和仪器响应的不确定度信息。通过MCMC，可以更准确地评估线性拟合、四参数模型拟合和三次样条插值拟合中的不确定度，为免疫分析结果提供更可靠的评定依据。

(2) 线性拟合的不确定度评定方法

本文件规定了采用MCMC法评定线性拟合中免疫分析结果及其不确定度的具体步骤。线性拟合广泛应用于免疫分析中，但当标准溶液和仪器响应均存在不可忽略的不确定度时，传统的最小二乘法可能无法准确评估不确定度。MCMC法通过随机采样和概率分布模型，能够同时考虑x和y方向的不确定度，提供更全面的不确定度评定结果。

(3) 四参数模型拟合的不确定度评定方法

四参数模型是免疫分析中常用的非线性拟合模型，用于描述标准曲线与仪器响应之间的关系。本文件详细说明了如何利用MCMC法评定四参数模型拟合中的不确定度。通过生成大量随机样本，MCMC能够模拟模型参数的分布特征，从而准确估计拟合结果的不确定度。该

方法特别适用于处理复杂的非线性关系和多维不确定度问题，为免疫分析结果的准确性和可靠性提供重要支持。

（4） 三次样条插值拟合的不确定度评定方法

三次样条插值拟合是一种常用的数据平滑和拟合技术，适用于免疫分析中的标准曲线拟合。本文件规定了采用MCMC法评定三次样条插值拟合中不确定度的具体方法。通过随机采样和概率分布模型，MCMC能够量化插值结果的不确定度，同时考虑标准溶液和仪器响应的不确定度。该方法能够有效处理非线性插值问题，为免疫分析结果的评定提供更精确的不确定度估计。

六．标准内容要点说明

《免疫分析 测量结果的不确定度评定》的主要内容要点说明如下：

1. 适用范围：采用马尔可夫链蒙特卡罗法确定线性拟合、四参数模型拟合以及三次样条插值拟合确定免疫分析结果及其不确定度的评定方法。适用于标准溶液和仪器响应均存在不可忽略的不确定度时线性拟合、四参数模型拟合以及三次样条插值拟合确定免疫分析结果及其不确定度。不适用于医学检验实验室免疫分析结果的不确定度评定。

2. 引用文件：参考引用了测量不确定度评定和表示（GB/T 27418-2017）、测量不确定度评定和表示 补充文件1: 基于蒙特卡洛方法的分布传播（GB/T 27419-2018）、医学实验室 测量不确定度评定指南（GB/Z 43280-2023）。

3. 术语和定义：主要包括免疫分析、抗体、抗原、测量不确定度、标准不确定度、合成标准不确定度、扩展不确定度、蒙特卡洛

方法、贝叶斯定理、马尔可夫链蒙特卡罗法等。

4. 概述：免疫分析基于抗体-抗原特异性结合，通过信号转换定量检测靶标（如蛋白质、激素等）。测量过程中，标准溶液浓度和仪器响应均存在不确定度，需通过数学模型（如线性拟合、四参数拟合等）计算试样浓度及其不确定度。

5. 不确定度评定步骤：标准溶液不确定度计算（$u(x_i)$）、仪器响应不确定度计算（$u(y_i)$和$u(y)$）、试样浓度不确定度计算（$u(x)$）、扩展不确定度计算(U)。

6. 附录示例：线性回归拟合示例，四参数拟合示例，三次样条插值拟合示例。

七． 主要试验（或验证）情况

1. Cry3A试样酶联免疫分析结果不确定度的评定示例

1.1 实验方法

采用商品化Cry3A转基因产品酶联免疫检测试剂盒高低两个水平的转基因试样进行检测，使用试剂盒自带的6个水平的标准溶液绘制标准曲线，每个浓度的标准溶液重复检测2次，每个试样重复检测6次。

1.2 实验结果

酶联免疫分析实验结果如表7.1所示。

表7.1 Cry3A试样酶联免疫分析试剂盒检测结果

样品名称	浓度 ng/mL	吸光度			吸光度平均值
空白	0	0.162	0.178	/	0.1700
标准 1	0.25	0.235	0.235	/	0.2350
标准 2	0.50	0.296	0.285	/	0.2905
标准 3	1.00	0.405	0.441	/	0.4230
标准 4	2.00	0.624	0.623	/	0.6235
标准 5	4.00	0.891	0.974	/	0.9325
试样 1	/	0.267 0.259	0.277 0.259	0.276 0.253	0.2650
试样 2	/	0.449	0.434	0.463	0.4480

0.445	0.450	0.446

根据$u(x_i)$,$u(y_i)$和$u(y)$的计算方式，标准溶液和试样的$u(x_i)$,$u(y_i)$和$u(y)$计算结果见表A.2。

表7.2 标准溶液和试样的$u(x_i)$,$u(y_i)$和$u(y)$计算结果

样品名称	x_i	$u(x_i)$	y_i	$u(y_i)$	$\overline{y}$	$u(\overline{y})$
空白	0	0	0.1700	0.01003	/	/
标准 1	0.25	0.02512	0.2350	0	/	/
标准 2	0.50	0.05017	0.2905	0.006896	/	/
标准 3	1.00	0.1002	0.4230	0.02257	/	/
标准 4	2.00	0.2002	0.6235	0.0006269	/	/
标准 5	4.00	0.4000	0.9325	0.05203	/	/
样品 1	/	/	/	/	0.2650	0.004020
样品 2	/	/	/	/	0.4480	0.003825

1.2.1 线性拟合$u(x)$和U的计算结果

通过Python编程计算$u(x)$，代码如下：

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import emcee
from scipy.stats import norm
# 数据
x_data = np.array([0, 0.25, 0.50, 1.00, 2.00, 4.00])
y_data = np.array([0.17, 0.235, 0.2905, 0.423, 0.6235, 0.9325])
u_x_data = np.array([1e-6, 0.02512, 0.05017, 0.1002, 0.2002, 0.4000])
u_y_data = np.array([0.010029883, 1e-6, 0.006895545, 0.022567238, 0.000626868, 0.05203002])
y0 = 0.2650
u_y0 = 0.004020
# 定义线性模型
def linear_model(x, a, b):
    return a * x + b
# 定义对数似然函数 (考虑 x 和 y 不确定度)
def log_likelihood(theta, x, y, yerr, xerr):
```

```
    a, b, log_f = theta
    model = linear_model(x, a, b)
    sigma2 = yerr**2 + (a * xerr)**2 + model**2 * np.exp(2 * log_f)  # 包含 x 不确定度
    return -0.5 * np.sum((y - model)**2 / sigma2 + np.log(sigma2))
# 定义先验分布
def log_prior(theta):
    a, b, log_f = theta
    if -5.0 < a < 0.5 and -0.5 < b < 1.0 and -10 < log_f < 1.0:
        return 0.0
    return -np.inf
# 定义后验分布
def log_probability(theta, x, y, yerr, xerr):
    lp = log_prior(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + log_likelihood(theta, x, y, yerr, xerr)
# MCMC 设置
nwalkers = 32
ndim = 3  # a, b, log_f
nsteps = 5000
# 初始位置
initial = np.array([0.1, 0.2, -3])  # a, b, log_f 的初始值
pos = initial + 1e-4 * np.random.randn(nwalkers, ndim)
# 创建采样器
sampler = emcee.EnsembleSampler(nwalkers, ndim, log_probability, args=(x_data, y_data,
u_y_data, u_x_data))
# 运行 MCMC
sampler.run_mcmc(pos, nsteps, progress=True)
# 获取样本
samples = sampler.get_chain(flat=True)
# 提取 a 和 b 的样本
a_samples = samples[:, 0]
b_samples = samples[:, 1]
# 考虑 u_y0 的影响：对 y0 进行抽样
```

```
n_y0_samples = 1000
y0_samples = np.random.normal(y0, u_y0, n_y0_samples)
# 初始化 x0 样本
x0_samples = np.array([])

# 对每个 y0 样本，计算 x0 的分布
for y0_sample in y0_samples:
    x0_samples = np.concatenate([x0_samples, (y0_sample - b_samples) / a_samples])
# 计算 x0 的不确定度
x0_mean = np.mean(x0_samples)
x0_std = np.std(x0_samples)
print(f"x0 = {x0_mean:.4f} +/- {x0_std:.4f}")
# 可视化
# 1. MCMC 采样轨迹
fig, axes = plt.subplots(ndim, figsize=(10, 7), sharex=True)
labels = ["a", "b", "log_f"]
for i in range(ndim):
    ax = axes[i]
    ax.plot(sampler.get_chain()[:, :, i], "k", alpha=0.3)
    ax.set_xlim(0, len(sampler.get_chain()))
    ax.set_ylabel(labels[i])
axes[-1].set_xlabel("step number")
plt.suptitle("MCMC Sampling Trace")
plt.tight_layout(rect=[0, 0.05, 1, 0.95]) # 调整子图，避免标题重叠
plt.show()
# 2. 后验分布
fig, axes = plt.subplots(ndim, figsize=(10, 7))
labels = ["a", "b", "log_f"]
for i in range(ndim):
    ax = axes[i]
    ax.hist(samples[:, i], bins=50, density=True, color="steelblue", edgecolor="black")
    ax.set_xlabel(labels[i])
    ax.set_ylabel("Probability Density")
plt.suptitle("Posterior Distributions")
```

```
plt.tight_layout(rect=[0, 0.05, 1, 0.95])

plt.show()

# 3. x0 的后验分布

plt.hist(x0_samples, bins=50, density=True, color="steelblue", edgecolor="black")

plt.xlabel("x0")

plt.ylabel("Probability Density")

plt.title("Posterior Distribution of x0")

plt.axvline(x=x0_mean, color='red', linestyle='--', label=f'Mean = {x0_mean:.4f}')

plt.axvline(x=x0_mean + x0_std, color='green', linestyle='--', label=f'Mean + Std = {x0_mean + x0_std:.4f}')

plt.axvline(x=x0_mean - x0_std, color='green', linestyle='--', label=f'Mean - Std = {x0_mean - x0_std:.4f}')

plt.legend()

plt.show()

# 4. 拟合图

plt.figure(figsize=(8, 6))

plt.errorbar(x_data, y_data, xerr=u_x_data, yerr=u_y_data, fmt="o", label="Data")

# 绘制样本的拟合线

nsamples_plot = 100

inds = np.random.randint(len(samples), size=nsamples_plot)

for ind in inds:

    a, b, log_f = samples[ind]

    y_fit = linear_model(x_data, a, b)

    plt.plot(x_data, y_fit, "C1", alpha=0.1)

# 绘制最佳拟合线

a_best, b_best, log_f_best = np.mean(a_samples), np.mean(b_samples), np.mean(samples[:,2])  # 使用平均值作为最佳估计

x_fit = np.linspace(np.min(x_data), np.max(x_data), 100)

y_fit = linear_model(x_fit, a_best, b_best)

plt.plot(x_fit, y_fit, "r-", linewidth=2, label="Best Fit")

# 标注 x0 位置

plt.axvline(x=x0_mean, color="k", linestyle="--", label=f"x0 = {x0_mean:.4f} +/- {x0_std:.4f}")

# 绘制 y0 的不确定区域

plt.axhspan(y0 - u_y0, y0 + u_y0, color='gray', alpha=0.3, label='y0 Uncertainty')

# 绘制 y0 的中心线
```

```
plt.axhline(y=y0, color="k", linestyle=":")
# 添加十字线
plt.plot(x0_mean, y0, marker='+', color='k', markersize=10)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Fit with MCMC")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

低值试样线性拟合结果如图7.1所示，试样分析结果x=（0.3960 ± 0.0371）ng/mL，即x值的不确定 $u(x)$= 0.0371 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.0371=0.075 ng/mL，试样分析结果可以表示为（0.396 ± 0.075）ng/mL。

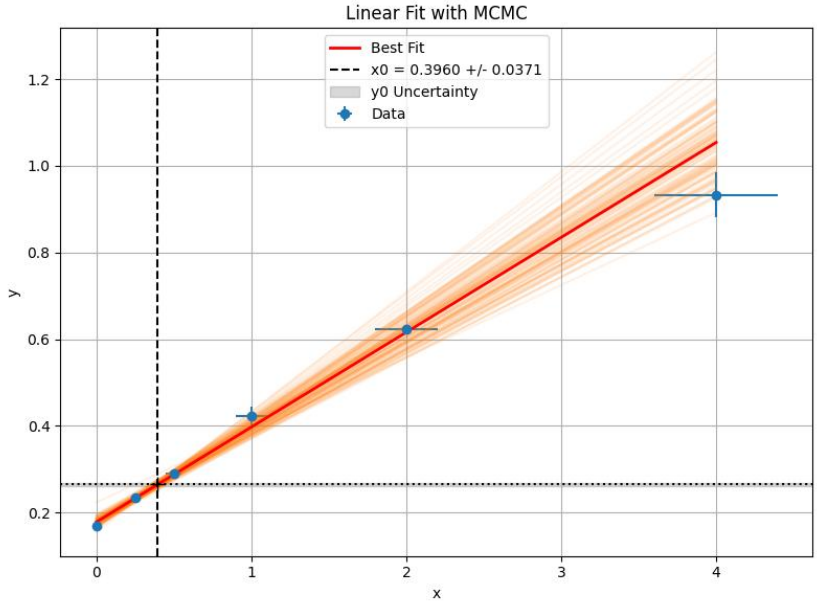

图7.1 低值试样线性拟合计算结果

如果使用试剂盒自带的标准溶液直接测定，这些标准溶液没有提供不确定度也无法通过分析确定不确定度，则忽略掉标准溶液的不确定度，将$u(x_i)$ 均设置为10^{-6} ng/mL，此时低值试样线性拟合结果如图7.2所示，试样分析结果x=（0.3883 ± 0.0346）ng/mL，即x值的不确定 $u(x)$=0.0346 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.0346=

0.070 ng/mL，试样分析结果可以表示为（0.388 ± 0.070）ng/mL。

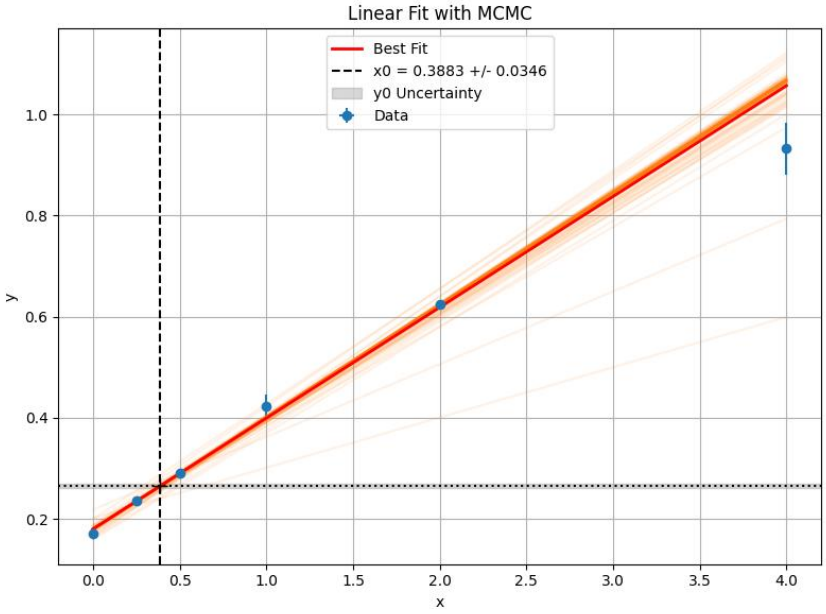

图7.2 低值试样线性拟合计算结果[$u(x_i)$ =0]

高值试样线性拟合结果如图7.3所示，试样分析结果x=（1.2379 ± 0.1314）ng/mL，即x值的不确定 $u(x)$= 0.131 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.131=0.27 ng/mL，试样分析结果可以表示为（1.23 ± 0.27）ng/mL。

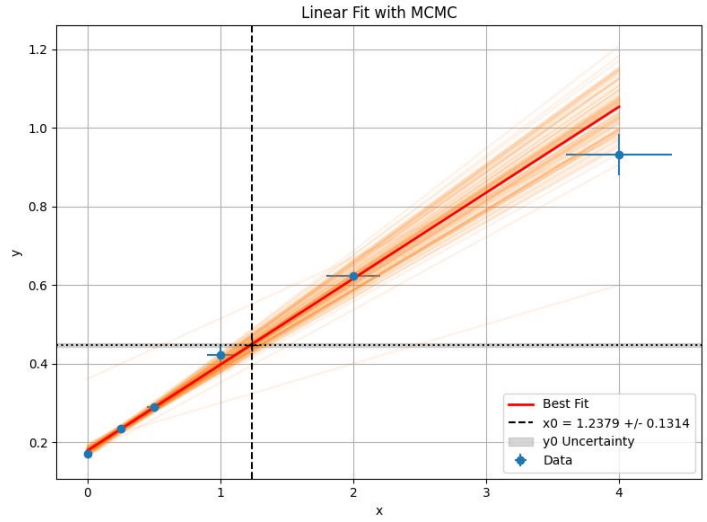

图7.3 高值试样线性拟合计算结果

如果使用试剂盒自带的标准溶液直接测定，这些标准溶液没有提供不确定度也无法通过分析确定不确定度，则忽略掉标准溶液的不确定度，将$u(x_i)$ 均设置为10^{-6} ng/mL，此时高值试样线性拟合结果如图A.4所示，

试样分析结果x=（1.2291 ± 0.1078）ng/mL，即x值的不确定 $u(x)$=0.108 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.108= 0.22 ng/mL，试样分析结果可以表示为（1.22 ± 0.22）ng/mL。

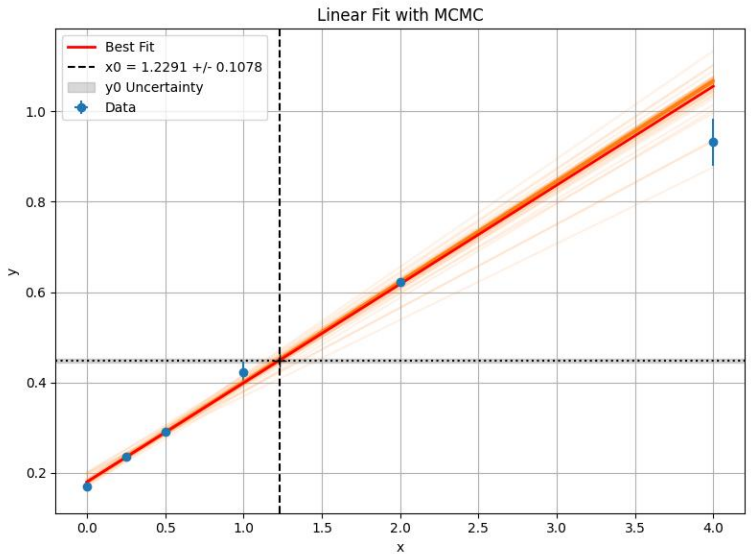

图7.4 高值试样线性拟合计算结果[$u(x_i)$ =0]

1.2.2 四参数模型拟合$u(x)$和U的计算结果

通过Python编程计算$u(x)$，代码如下：

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import emcee
import corner
from scipy.optimize import curve_fit
from scipy.stats import norm

# Define the four-parameter model
def four_param_model(x, A, B, C, D):
    return D + (A - D) / (1 + (x/C)**B)

# Define the log-likelihood function
def log_likelihood(theta, x_data, y_data, u_x_data, u_y_data):
    A, B, C, D = theta
```

```
        model = four_param_model(x_data, A, B, C, D)
        sigma2 = u_y_data**2 + (B * (A - D) * (x_data**B) * u_x_data / (C**B * (1 + x_data/C)**B)**2))**2
        return -0.5 * np.sum((y_data - model)**2 / sigma2 + np.log(2 * np.pi * sigma2))

    # Define the log-prior function
    def log_prior(theta):
        A, B, C, D = theta
        if not all(0 < value < 10 for value in [A, B, C, D]):
            return -np.inf
        return 0.0

    # Define the log-probability function
    def log_probability(theta, x_data, y_data, u_x_data, u_y_data):
        lp = log_prior(theta)
        if not np.isfinite(lp):
            return -np.inf
        return lp + log_likelihood(theta, x_data, y_data, u_x_data, u_y_data)

    # Define the data
    x_data = np.array([0, 0.25, 0.50, 1.00, 2.00, 4.00])
    y_data = np.array([0.17, 0.235, 0.2905, 0.423, 0.6235, 0.9325])
    u_x_data = np.array([1e-6, 0.02512, 0.05017, 0.1002, 0.2002, 0.4000])
    u_y_data = np.array([0.010029883, 1e-6, 0.006895545, 0.022567238, 0.000626868, 0.05203002])
    y0 = 0.2650
    u_y0 = 0.004020

    # Initial guess for the parameters
    initial_guess = [0.169915, 1.064054, 6.244783, 2.156547]  # A, B, C, D

    # Maximum likelihood estimation (MLE) using curve_fit
    try:
        popt, pcov = curve_fit(four_param_model, x_data, y_data, p0=initial_guess, sigma=u_y_data, absolute_sigma=True)
        A_fit, B_fit, C_fit, D_fit = popt
```

```
        print("MLE Fit Parameters (curve_fit):")
        print("A =", A_fit)
        print("B =", B_fit)
        print("C =", C_fit)
        print("D =", D_fit)
    except RuntimeError as e:
        print(f"Error during curve_fit: {e}")
        A_fit, B_fit, C_fit, D_fit = initial_guess  # Fallback to initial guess
        pcov = np.eye(4) * 1e-6  # Dummy covariance matrix

    # MCMC setup
    nwalkers = 32
    ndim = 4
    nsteps = 10000
    burnin = 2000

    # Initialize the walkers
    rng = np.random.default_rng()
    initial = np.array(initial_guess)
    pos = initial + 1e-4 * rng.standard_normal((nwalkers, ndim))

    # Create the sampler
    sampler = emcee.EnsembleSampler(nwalkers, ndim, log_probability, args=(x_data, y_data,
u_x_data, u_y_data))

    # Run the MCMC sampler
    sampler.run_mcmc(pos, nsteps, progress=True)

    # Discard the burn-in samples
    samples = sampler.get_chain(discard=burnin, flat=True)

    # Analyze the MCMC results
    A_mcmc, B_mcmc, C_mcmc, D_mcmc = np.mean(samples, axis=0)
    print("\nMCMC Fit Parameters:")
    print("A =", A_mcmc)
```

```
print("B =", B_mcmc)
print("C =", C_mcmc)
print("D =", D_mcmc)

# Corner plot
corner.corner(samples, labels=["A", "B", "C", "D"], truths=[A_fit, B_fit, C_fit, D_fit])
plt.suptitle("MCMC Parameter Distributions", y=0.98)
plt.show()  # Display the corner plot
plt.close()

# Function to find x0 and its uncertainty, now including u_y0
def find_x0(y0, u_y0, samples, x_data, num_y0_samples=100):
    x0_values = []
    A_samples = samples[:, 0]
    B_samples = samples[:, 1]
    C_samples = samples[:, 2]
    D_samples = samples[:, 3]
    rng = np.random.default_rng()  # Use a random number generator
    for A, B, C, D in zip(A_samples, B_samples, C_samples, D_samples):
        # Sample y0 values from a normal distribution
        y0_samples = norm.rvs(loc=y0, scale=u_y0, size=num_y0_samples, random_state=rng)
        for y0_sampled in y0_samples: #Iterate over the sampled y0 values
            # Use a root-finding method to solve for x0
            def equation(x):
                return four_param_model(x, A, B, C, D) - y0_sampled
            # Providing a bracket around the root can help the solver
            x_lower = min(x_data)
            x_upper = max(x_data)
            try:
                from scipy.optimize import brentq
                x0 = brentq(equation, x_lower, x_upper)
                x0_values.append(x0)
            except ValueError:
                # If brentq fails (e.g., no root within the bracket), return NaN
```

```
                x0_values.append(np.nan)
            except Exception as e:
                print(f"Error during root finding: {e}")
                x0_values.append(np.nan)
    x0_values = np.array(x0_values)
    x0_values = x0_values[~np.isnan(x0_values)]  # Filter out NaN values
    if len(x0_values) == 0:
        return np.nan, np.nan  # Return NaN if no valid x0 values are found
    x0_mean = np.mean(x0_values)
    x0_uncertainty = np.std(x0_values)
    return x0_mean, x0_uncertainty

# Find x0 and its uncertainty
x0, ux0 = find_x0(y0, u_y0, samples, x_data) #Pass u_y0 to find_x0
print("\nx0 =", x0)
print("ux0 =", ux0)

# Plot the data and the fit
plt.figure(figsize=(10, 8))
plt.errorbar(x_data, y_data, xerr=u_x_data, yerr=u_y_data, fmt="o", label="Data")

# Plot the best-fit curve
x_fit = np.linspace(min(x_data), max(x_data), 100)
y_fit = four_param_model(x_fit, A_mcmc, B_mcmc, C_mcmc, D_mcmc)
plt.plot(x_fit, y_fit, label="MCMC Best Fit")

# Plot x0 with crosshairs
if not np.isnan(x0):
    plt.axvline(x=x0, color='red', linestyle='--', label=f'x0 = {x0:.3f}')
    plt.axhline(y=y0, color='green', linestyle='--', label=f'y0 = {y0:.3f}')
    plt.errorbar(x0, y0, xerr=ux0, yerr=u_y0, fmt='+', color='purple', markersize=12, label='x0
Uncertainty')
plt.xlabel("x")
plt.ylabel("y")
plt.title("Four-Parameter Fit with x0")
```

```
plt.legend()
plt.grid(True)
plt.show()  # Display the fit plot
plt.close()

# Plot MCMC samples
plt.figure(figsize=(12, 8))
for i in range(ndim):
    plt.subplot(ndim, 1, i + 1)
    plt.plot(sampler.get_chain()[:, :, i], color="k", alpha=0.3)
    plt.ylabel(f"Parameter {['A', 'B', 'C', 'D'][i]}")
plt.xlabel("Step Number")
plt.suptitle("MCMC Sample Traces", y=0.98)
plt.tight_layout()
plt.show() # Display MCMC traces
plt.close()
# Plot posterior distributions
plt.figure(figsize=(12, 8))
for i in range(ndim):
    plt.subplot(ndim, 1, i + 1)
    plt.hist(samples[:, i], bins=50, density=True, alpha=0.6, color="skyblue")
    plt.xlabel(f"Parameter {['A', 'B', 'C', 'D'][i]}")
    plt.ylabel("Density")
    plt.title(f"Posterior Distribution of Parameter {['A', 'B', 'C', 'D'][i]}")
plt.suptitle("Posterior Distributions of Parameters", y=0.98)
plt.tight_layout()
plt.show() # Display posterior distributions
plt.close()
```

低值试样四参数拟合结果如图7.5所示，试样分析结果x=（0.373 ± 0.0210) ng/mL，即x值的不确定 $u(x)$ = 0.0210 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.021=0.042 ng/mL，试样分析结果可以表示为（0.373 ± 0.042）ng/mL。

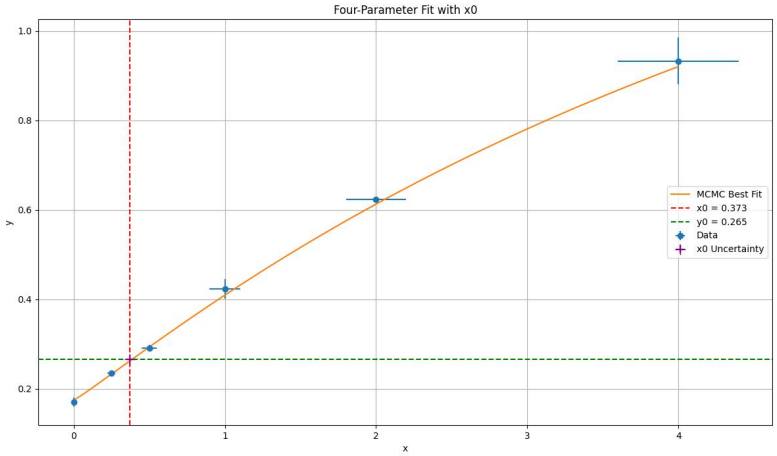

图7.5 低值试样四参数拟合计算结果

如果使用试剂盒自带的标准溶液直接测定，这些标准溶液没有提供不确定度也无法通过分析确定不确定度，则忽略掉标准溶液的不确定度，将$u(x_i)$均设置为10^{-6} ng/mL，此时低值试样四参数拟合结果如图7.6所示，试样分析结果x=（0.369 ± 0.017）ng/mL，即x值的不确定 $u(x)$= 0.017 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.017=0.034 ng/mL，试样分析结果可以表示为（0.369 ± 0.034）ng/mL。

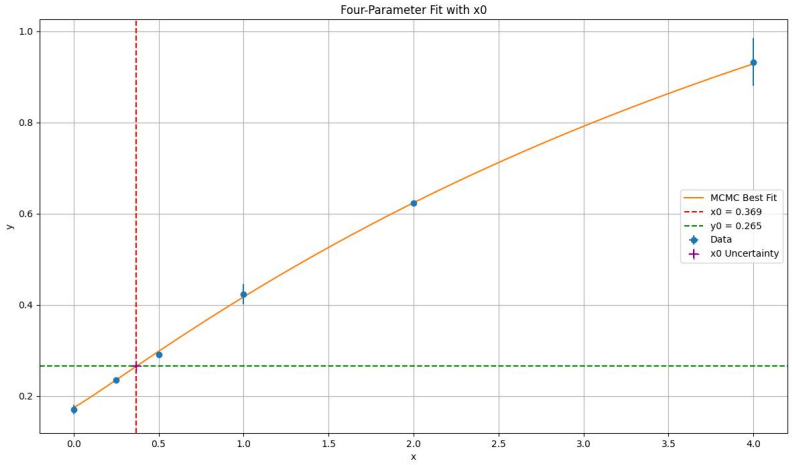

图7.6 低值试样四参数拟合计算结果[$u(x_i)$=0]

高值试样四参数拟合结果如图7.7所示，试样分析结果x=

（1.164 ± 0.095）ng/mL，即x值的不确定 $u(x)$ = 0.095 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.095 = 0.19 ng/mL，试样分析结果可以表示为（1.16 ± 0.19）ng/mL。

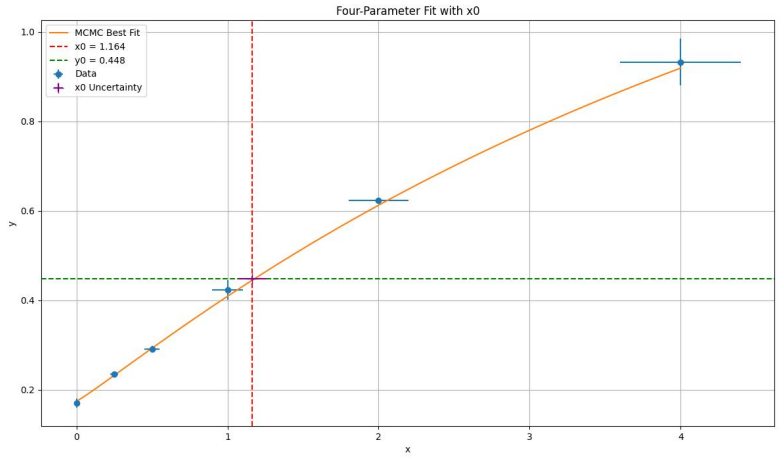

图7.7 高值试样四参数拟合计算结果

如果使用试剂盒自带的标准溶液直接测定，这些标准溶液没有提供不确定度也无法通过分析确定不确定度，则忽略掉标准溶液的不确定度，将$u(x_i)$ 均设置为10^{-6} ng/mL，此时高值试样四参数拟合结果如图7.8所示，试样分析结果x=（1.134 ± 0.020）ng/mL，即x值的不确定 $u(x)$ =0.020 ng/mL，取扩展因子k=2，则扩展不确定度U= 2×0.020 = 0.040 ng/mL，试样分析结果可以表示为（1.134 ± 0.040）ng/mL。

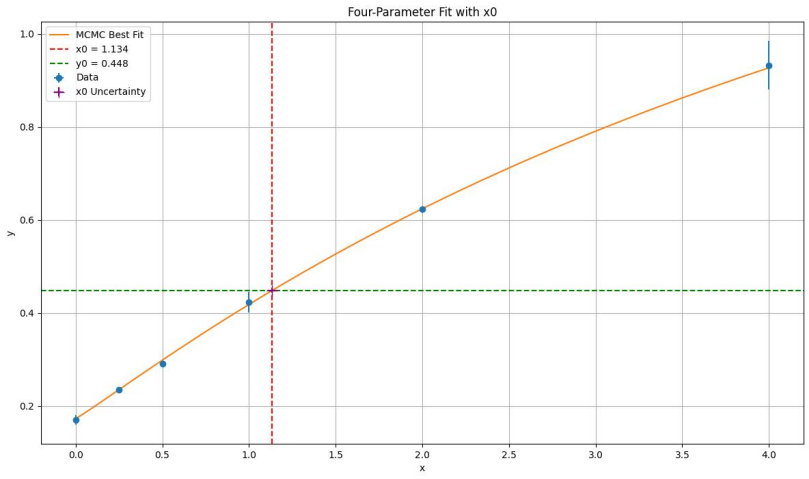

图7.8 高值试样四参数拟合计算结果[$u(X_i)$ =0]

1.2.3 三次样条插值拟合$u(x)$和U的计算结果

通过Python编程计算$u(x)$，代码如下：

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline
from scipy.optimize import brentq
import emcee
import corner
import seaborn as sns

# 设置 seaborn 样式
sns.set(style="whitegrid")

# 定义数据
x_data = np.array([0, 0.25, 0.50, 1.00, 2.00, 4.00])
y_data = np.array([0.17, 0.235, 0.2905, 0.423, 0.6235, 0.9325])
u_x_data = np.array([1e-6, 0.02512, 0.05017, 0.1002, 0.2002, 0.4000])
u_y_data = np.array([0.010029883, 1e-6, 0.006895545, 0.022567238, 0.000626868, 0.05203002])
y0 = 0.2650
u_y0 = 0.004020

n_params = 13  # 6 x_real, 6 y_real, 1 y0_true

def log_prior(theta):
    x_real = theta[:6]
    y_real = theta[6:12]
    y0_true = theta[12]

    # 检查x_real是否递增
    if not np.all(np.diff(x_real) > 0):
        return -np.inf
```

```
        # 计算先验概率
        log_p = 0.0
        # x_real的先验
        log_p += np.sum(-0.5 * ((x_real - x_data) ** 2 / (u_x_data ** 2)) - np.log(u_x_data * np.sqrt(2 * np.pi)))
        # y_real的先验
        log_p += np.sum(-0.5 * ((y_real - y_data) ** 2 / (u_y_data ** 2)) - np.log(u_y_data * np.sqrt(2 * np.pi)))
        # y0_true的先验
        log_p += -0.5 * ((y0_true - y0) ** 2 / (u_y0 ** 2)) - np.log(u_y0 * np.sqrt(2 * np.pi))

        return log_p

    def log_probability(theta):
        lp = log_prior(theta)
        if not np.isfinite(lp):
            return -np.inf

        x_real = theta[:6]
        y_real = theta[6:12]
        y0_true = theta[12]

        # 构造三次样条插值
        try:
            cs = CubicSpline(x_real, y_real, extrapolate=False)
        except:
            return -np.inf

        # 定义求解x0的函数
        def func(x):
            return cs(x) - y0_true

        x_min = x_real.min()
        x_max = x_real.max()
```

```
    # 检查端点处的符号
    try:
        f_min = func(x_min)
        f_max = func(x_max)
    except:
        return -np.inf

    if np.sign(f_min) == np.sign(f_max):
        return -np.inf  # 没有根

    # 寻找根
    try:
        x0_val = brentq(func, x_min, x_max)
    except:
        return -np.inf

    return lp

# 生成初始位置
n_walkers = 50

def generate_initial_positions(n_walkers):
    initial = np.zeros((n_walkers, n_params))
    for i in range(n_walkers):
        valid = False
        while not valid:
            x_real = x_data + np.random.normal(0, u_x_data / 10)
            if np.all(np.diff(x_real) > 0):
                valid = True
        y_real = y_data + np.random.normal(0, u_y_data / 10)
        y0_true = y0 + np.random.normal(0, u_y0 / 10)
        initial[i] = np.concatenate([x_real, y_real, [y0_true]])
    return initial
```

```
initial_positions = generate_initial_positions(n_walkers)

# 运行MCMC
sampler = emcee.EnsembleSampler(n_walkers, n_params, log_probability)

# Burn-in
print("Running burn-in...")
n_burn = 1000
state = sampler.run_mcmc(initial_positions, n_burn, progress=True)
sampler.reset()

# 主采样
n_steps = 5000
print("Running production...")
sampler.run_mcmc(state, n_steps, progress=True)

# 获取样本
samples = sampler.get_chain(flat=True)

# 计算x0的后验样本
x0_samples = []
for theta in samples:
    x_real = theta[:6]
    y_real = theta[6:12]
    y0_true = theta[12]

    try:
        cs = CubicSpline(x_real, y_real, extrapolate=False)
    except:
        continue

    def func(x):
        return cs(x) - y0_true
```

```
    x_min = x_real.min()
    x_max = x_real.max()

    try:
        f_min = func(x_min)
        f_max = func(x_max)
    except:
        continue

    if np.sign(f_min) == np.sign(f_max):
        continue

    try:
        x0 = brentq(func, x_min, x_max)
        x0_samples.append(x0)
    except:
        continue

x0_samples = np.array(x0_samples)

# 检查x0_samples是否为空
if len(x0_samples) == 0:
    raise ValueError("No valid x0 samples found.")

# 计算x0的中位数和不确定度
x0_median = np.median(x0_samples)
ux0 = np.std(x0_samples)

# 输出x0的值
print(f"Median x0: {x0_median:.4f}")
print(f"Estimated uncertainty u_x0: {ux0:.4f}")

# 可视化x0的后验分布
```

```
    plt.figure(figsize=(10, 6))
    plt.hist(x0_samples, bins=50, density=True, alpha=0.6, color='blue')
    plt.title(f'Posterior Distribution of x0\n $u_{{{x0}}}$ = {ux0:.4f}', fontsize=16)
    plt.xlabel('x0', fontsize=14)
    plt.ylabel('Probability Density', fontsize=14)
    plt.show()

    # 绘制拟合曲线
    plt.figure(figsize=(10, 6))
    plt.errorbar(x_data, y_data, xerr=u_x_data, yerr=u_y_data, fmt='o', color='k', label='Data with Errors')

    # 绘制部分后验样本的曲线
    for i in np.random.choice(len(samples), 100):
        theta = samples[i]
        x_real = theta[:6]
        y_real = theta[6:12]
        try:
            # 构造三次样条插值
            cs = CubicSpline(x_real, y_real, extrapolate=False)
            x_plot = np.linspace(x_real.min(), x_real.max(), 100)
            y_plot = cs(x_plot)
            plt.plot(x_plot, y_plot, color='gray', alpha=0.1, label='Posterior Samples' if i == 0 else None)
        except:
            continue  # 如果插值失败，跳过该样本

    # 绘制中位数的x0
    plt.axvline(x0_median, color='r', linestyle='--', label=f'Median x0 = {x0_median:.3f}')
    plt.axhline(y0, color='b', linestyle='--', label=f'y0 = {y0}')
    plt.plot(x0_median, y0, 'r+', markersize=15, markeredgewidth=2, label='x0 Position')

    # 添加标题和标签
    plt.xlabel('x', fontsize=14)
    plt.ylabel('y', fontsize=14)
```

```
plt.title('Spline Fits with MCMC Uncertainty', fontsize=16)

# 添加图例
plt.legend(loc='upper left', fontsize=12)

# 调整布局
plt.tight_layout()

# 显示图形
plt.show()

# 使用corner绘制参数的后验分布
corner_labels = [f'x_real_{i+1}' for i in range(6)] + [f'y_real_{i+1}' for i in range(6)] + ['y0_true']
corner.corner(samples, labels=corner_labels, truths=np.median(samples, axis=0))
plt.show()
```

低值试样三次样条插值拟合结果如图7.9所示，试样分析结果x=（0.387 ± 0.056）ng/mL，即x值的不确定$u(x)$ = 0.056 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.056=0.12 ng/mL，试样分析结果可以表示为（0.39 ± 0.12) ng/mL。

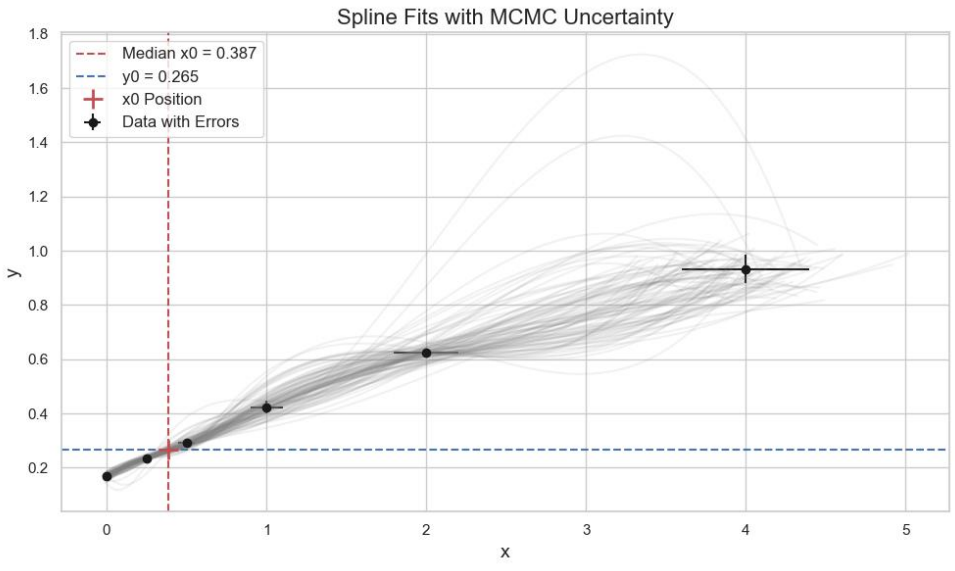

图7.9 低值试样三次样条插值拟合计算结果

如果使用试剂盒自带的标准溶液直接测定，这些标准溶液没有

提供不确定度也无法通过分析确定不确定度，则忽略掉标准溶液的不确定度，将$u(x_i)$均设置为10^{-6} ng/mL，此时低值试样三次样条插值拟合结果如图7.10所示，试样分析结果x=（0.387 ± 0.028）ng/mL，即x值的不确定 $u(x)$= 0.028 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.028 = 0.056 ng/mL，试样分析结果可以表示为（0.387 ± 0.056）ng/mL。

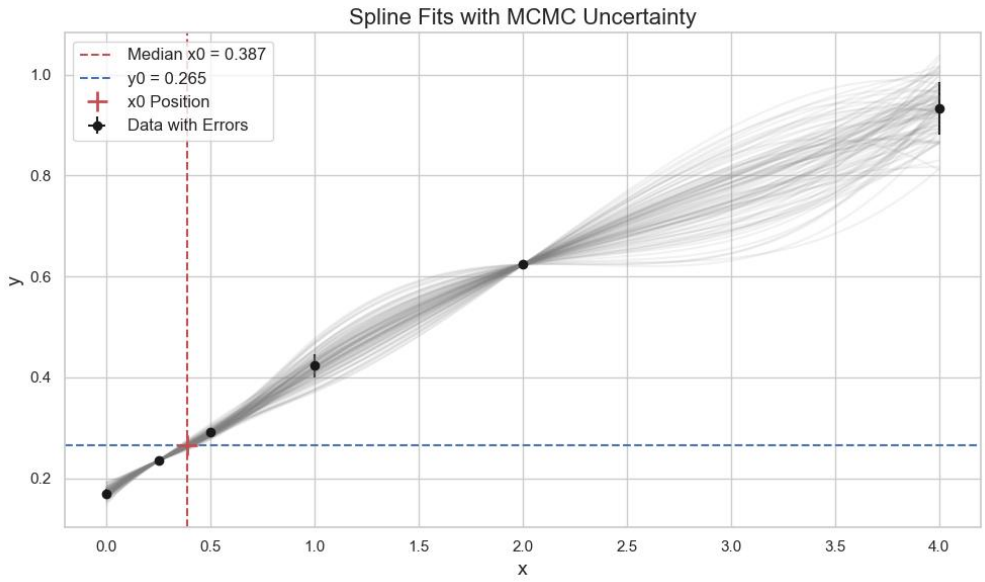

图7.10 低值试样三次样条插值拟合计算结果[$u(x_i)$ =0]

高值试样三次样条插值拟合结果如图7.11所示，试样分析结果x=（1.09 ± 0.15）ng/mL，即x值的不确定$u(x)$ = 0.15 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.15=0.30 ng/mL，试样分析结果可以表示为（1.09 ± 0.30）ng/mL。

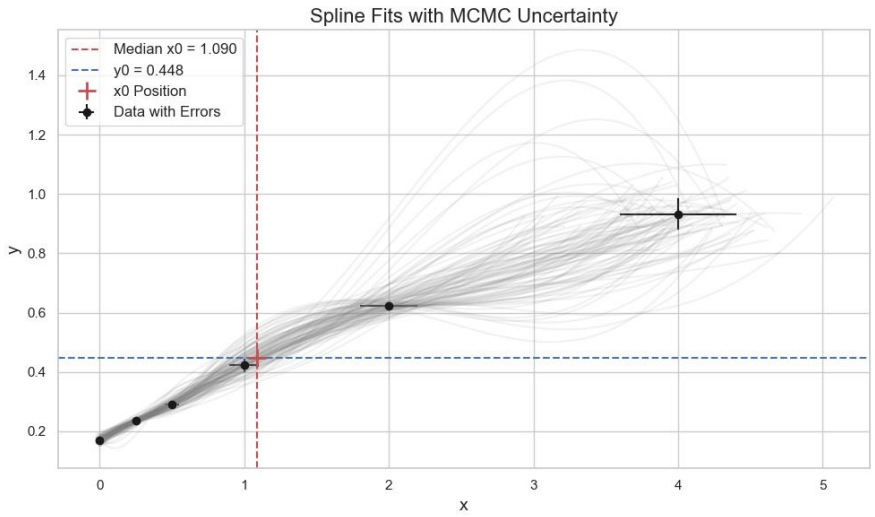

图7.11 高值试样三次样条插值拟合计算结果

如果使用试剂盒自带的标准溶液直接测定，这些标准溶液没有提供不确定度也无法通过分析确定不确定度，则忽略掉标准溶液的不确定度，将$u(x_i)$均设置为10^{-6} ng/mL，此时高值试样三次样条插值拟合结果如图7.12所示，试样分析结果x=（1.10 ± 0.11）ng/mL，即x值的不确定 $u(x)$= 0.11 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.11= 0.22 ng.mL，试样分析结果可以表示为（1.10 ± 0.22）ng/mL。

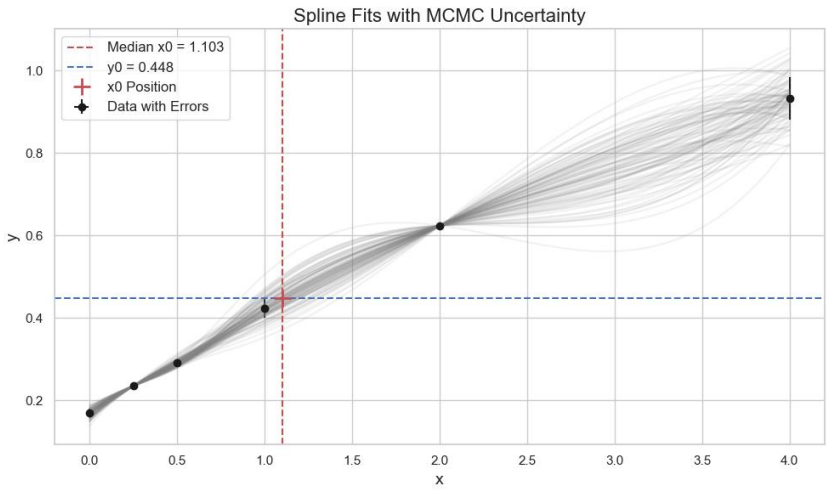

图 7.12 高值试样三次样条插值拟合计算结果[$u(x_i)$ =0]

2. 游离三碘甲状腺原氨酸试样化学发光法结果不确定的评定示例

2.1 实验方法

采用游离三碘甲状腺原氨酸测定试剂盒高低两个水平的试样进行检测，使用试剂盒自带的5个水平的标准溶液绘制标准曲线，其中空白样品的标准溶液重复检测3次，其余每个浓度的标准溶液重复检测2次，每个试样重复检测2次。

2.2 实验结果

化学发光法实验结果如表7.2所示。

表7.2 游离三碘甲状腺原氨酸测定试剂盒检测结果

样品名称	浓度pg/mL		光量值(*RLU*)		光量值平均值
空白(s0)	0	2078480	2049750	2208790	2112340
标准1(s1)	2.18	1309880	1290680	/	1300280
标准2(s2)	7.02	506580	473042	/	489811
标准3(s3)	17.97	95879	97814	/	96847
标准4(s4)	24.94	23870	23877	/	23874
试样1	/	1006450	1005530	/	1005990
试样2	/	683371	674321	/	678846

根据$u(x_i)$,$u(y_i)$和$u(y)$的计算方式，标准溶液和试样的$u(x_i)$,$u(y_i)$和$u(y)$计算结果见表7.3。

表7.3 标准溶液和试样的$u(x_i)$,$u(y_i)$和$u(y)$计算结果

样品名称	x_i	$u(x_i)$	y_i	$u(y_i)$	$\overline{y}$	$u(\overline{y})$
空白(s0)	0	0	2112340	54255.36919	/	/
标准1(s1)	2.18	0.253079379	1300280	12028.39568	/	/
标准2(s2)	7.02	0.541241421	489811	21010.85074	/	/
标准3(s3)	17.97	0.804611057	96846.5	1212.236752	/	/
标准4(s4)	24.94	0.31410032	23873.5	4.38535259	/	/
样品1	/	/	/	/	1005990	576.3606261
样品2	/	/	/	/	678846	5669.63442

2.2.1 四参数模型拟合$u(x)$和U的计算结果

通过Python编程计算$u(x)$，代码如下：

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import emcee
import corner
from scipy.optimize import curve_fit, brentq
from scipy.stats import norm


# Define the four-parameter model
def four_param_model(x, A, B, C, D):
    return D + (A - D) / (1 + (x / C) ** B)


# Define the log-likelihood function with improved numerical stability
def log_likelihood(theta, x_data, y_data, u_x_data, u_y_data):
    A, B, C, D = theta
    model = four_param_model(x_data, A, B, C, D)
    term = (x_data / C) ** B
    denominator = (1 + term) ** 2
    numerator = B * (A - D) * (x_data ** B) * u_x_data
    sigma2 = u_y_data ** 2 + (numerator / (C ** B * denominator)) ** 2
    sigma2 = np.clip(sigma2, 1e-10, None)  # Prevent division by zero
    return -0.5 * np.sum((y_data - model) ** 2 / sigma2 + np.log(2 * np.pi * sigma2))


# Define the log-prior function
def log_prior(theta):
    A, B, C, D = theta
    if not all(0 < value < 10 for value in [A, B, C, D]):
        return -np.inf
    return 0.0


# Define the log-probability function
def log_probability(theta, x_data, y_data, u_x_data, u_y_data):
    lp = log_prior(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + log_likelihood(theta, x_data, y_data, u_x_data, u_y_data)


# Define the data
x_data = np.array([0.00000001, 2.18, 7.02, 17.97, 24.94])
y_data = np.array([2112340, 1300280, 489811, 96846.5, 23873.5])
u_x_data = np.array([0.00000001, 0.25308, 0.54124, 0.80461, 0.3141])
u_y_data = np.array([54255.4, 12028.4, 21010.9,1212.24,4.38535])
y0 = 678846
u_y0 = 5669.63442

# Initial guess for the parameters
initial_guess = [2112341.33478, 1.30913, 3.35404,-127128.81488]  # A, B, C, D
```

```
# Maximum likelihood estimation (MLE) using curve_fit with better error handling
try:
    popt, pcov = curve_fit(four_param_model, x_data, y_data,
                           p0=initial_guess,
                           sigma=u_y_data,
                           absolute_sigma=True,
                           maxfev=10000)
    A_fit, B_fit, C_fit, D_fit = popt
    print("MLE Fit Parameters (curve_fit):")
    print(f"A = {A_fit:.6f}")
    print(f"B = {B_fit:.6f}")
    print(f"C = {C_fit:.6f}")
    print(f"D = {D_fit:.6f}")
except Exception as e:
    print(f"Error during curve_fit: {e}")
    A_fit, B_fit, C_fit, D_fit = initial_guess
    pcov = np.eye(4) * 1e-6

# MCMC setup with fixed random state for reproducibility
rng = np.random.RandomState(42)
nwalkers = 32
ndim = 4
nsteps = 10000
burnin = 2000

# Initialize the walkers
initial = np.array(initial_guess)
pos = initial + 1e-4 * rng.randn(nwalkers, ndim)

# Create and run the sampler
sampler = emcee.EnsembleSampler(nwalkers, ndim, log_probability,
                                args=(x_data, y_data, u_x_data, u_y_data))
sampler.run_mcmc(pos, nsteps, progress=True)

# Process samples
samples = sampler.get_chain(discard=burnin, flat=True)
A_mcmc, B_mcmc, C_mcmc, D_mcmc = np.median(samples, axis=0)  # Using median for robustness
print("\nMCMC Fit Parameters (median):")
print(f"A = {A_mcmc:.6f}")
print(f"B = {B_mcmc:.6f}")
print(f"C = {C_mcmc:.6f}")
print(f"D = {D_mcmc:.6f}")

# Corner plot with better formatting
fig = corner.corner(samples,
                    labels=["A", "B", "C", "D"],
                    truths=[A_fit, B_fit, C_fit, D_fit],
                    quantiles=[0.16, 0.5, 0.84],
                    show_titles=True,
                    title_kwargs={"fontsize": 12})
plt.suptitle("MCMC Parameter Distributions", y=1.02, fontsize=14)
plt.tight_layout()
```

```
plt.show()
plt.close()


# Improved x0 calculation with better error handling
def find_x0(y0, u_y0, samples, x_data, num_y0_samples=100):
    x0_values = []
    rng = np.random.RandomState(42)  # Fixed random state
    y0_samples = norm.rvs(loc=y0, scale=u_y0, size=num_y0_samples * len(samples),
random_state=rng)

    for i, (A, B, C, D) in enumerate(samples):
        y0_sampled = y0_samples[i * num_y0_samples:(i + 1) * num_y0_samples]
        for y in y0_sampled:
            try:
                x0 = brentq(lambda x: four_param_model(x, A, B, C, D) - y,
                            min(x_data), max(x_data),
                            rtol=1e-6, maxiter=100)
                x0_values.append(x0)
            except (ValueError, RuntimeError):
                continue

    if not x0_values:
        return np.nan, np.nan

    x0_values = np.array(x0_values)
    return np.median(x0_values), np.std(x0_values)


x0, ux0 = find_x0(y0, u_y0, samples, x_data)
print(f"\nx0 = {x0:.6f} ± {ux0:.6f}")

# Enhanced plotting
plt.figure(figsize=(10, 8))
x_fit = np.linspace(min(x_data), max(x_data), 500)
y_fit = four_param_model(x_fit, A_mcmc, B_mcmc, C_mcmc, D_mcmc)

plt.errorbar(x_data, y_data, xerr=u_x_data, yerr=u_y_data,
             fmt="o", color="k", markersize=8,
             capsize=3, capthick=1, label="Data")

plt.plot(x_fit, y_fit, "r-", lw=2, label="MCMC Best Fit")

if not np.isnan(x0):
    plt.axvline(x=x0, color='b', linestyle='--', alpha=0.7, label=f'x0 = {x0:.3f}')
    plt.axhline(y=y0, color='g', linestyle='--', alpha=0.7, label=f'y0 = {y0:.3f}')
    plt.errorbar(x0, y0, xerr=ux0, yerr=u_y0,
                 fmt='+', color='m', markersize=12,
                 markeredgewidth=2, label='x0 Uncertainty')

plt.xlabel("x", fontsize=12)
plt.ylabel("y", fontsize=12)
plt.title("Four-Parameter Fit with x0", fontsize=14)
```

```
plt.legend(fontsize=10, framealpha=0.9)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
plt.close()

# Improved trace plots
fig, axes = plt.subplots(ndim, 1, figsize=(12, 10), sharex=True)
for i, ax in enumerate(axes):
    ax.plot(sampler.get_chain()[:, :, i].T, color="k", alpha=0.2)
    ax.set_ylabel(["A", "B", "C", "D"][i], fontsize=12)
    ax.yaxis.set_label_coords(-0.1, 0.5)
axes[-1].set_xlabel("Step Number", fontsize=12)
plt.suptitle("MCMC Sample Traces", y=0.99, fontsize=14)
plt.tight_layout()
plt.show()
plt.close()

# Improved posterior plots
fig, axes = plt.subplots(ndim, 1, figsize=(12, 10))
for i, ax in enumerate(axes):
    ax.hist(samples[:, i], bins=50, density=True,
            color="skyblue", edgecolor="navy", alpha=0.7)
    ax.set_xlabel(["A", "B", "C", "D"][i], fontsize=12)
    ax.set_ylabel("Density", fontsize=12)
    ax.grid(True, alpha=0.3)
plt.suptitle("Posterior Distributions of Parameters", y=0.99, fontsize=14)
plt.tight_layout()
plt.show()
plt.close()
```

低值试样四参数拟合结果如图7.13所示，试样分析结果x=（3.222±0.624) ng/mL，即x值的不确定 $u(x)$ = 0.624 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.624=1.248 ng/mL，试样分析结果可以表示为（3.222 ± 1.248）ng/mL。

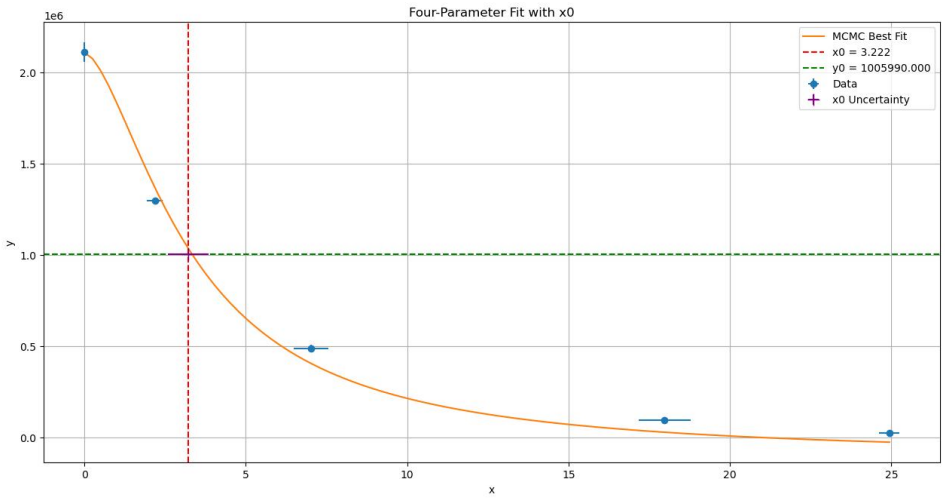

图7.13 低值试样四参数拟合计算结果

高值试样四参数拟合结果如图7.14所示，试样分析结果x=（4.924±1.151) ng/mL，即x值的不确定$u(x)$ = 1.151 ng/mL，取扩展因子k=2，则扩展不确定度U=2×1.151=2.302 ng/mL，试样分析结果可以表示为（4.924± 2.302）ng/mL。

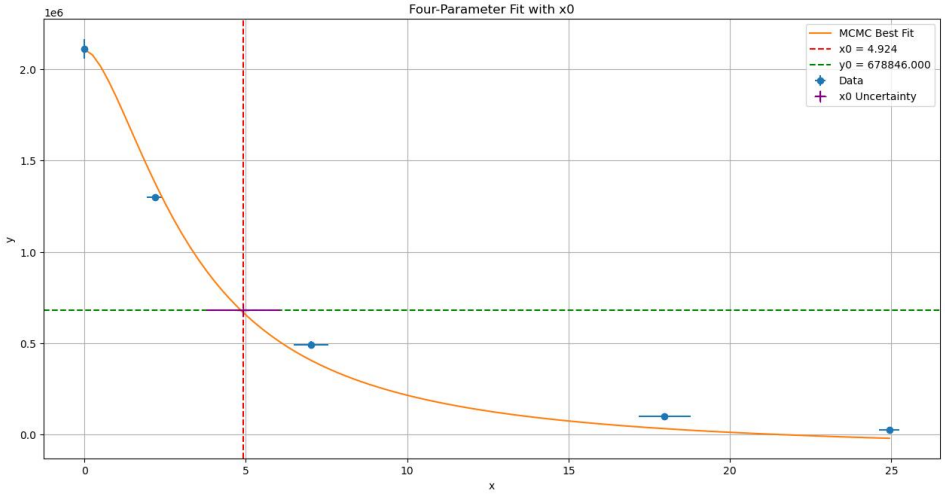

图7.14 高值试样四参数拟合计算结果

2.2.2 三次样条插值拟合$u(x)$和U的计算结果

通过Python编程计算$u(x)$，代码如下：

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline
from scipy.optimize import brentq
```

```
import emcee
import corner
import seaborn as sns

# 设置 seaborn 样式
sns.set(style="whitegrid")

# 定义数据
x_data = np.array([0, 2.1, 4.2, 8.12, 16.12, 24.4])
y_data = np.array([2808275, 1236735, 780648, 455927, 254947, 175650])
u_x_data = np.array([0.0000001, 0.4, 0.68, 1.21, 2.34, 3.66])
u_y_data = np.array([36943.5, 41617, 17394.2, 12054.1, 4439.86, 5061.32])
y0 = 587240
u_y0 = 5704.72

n_params = 13  # 6 x_real, 6 y_real, 1 y0_true

def log_prior(theta):
    x_real = theta[:6]
    y_real = theta[6:12]
    y0_true = theta[12]

    # 检查x_real是否递增
    if not np.all(np.diff(x_real) > 0):
        return -np.inf

    # 计算先验概率
    log_p = 0.0
    # x_real的先验
    log_p += np.sum(-0.5 * ((x_real - x_data) ** 2 / (u_x_data ** 2)) - np.log(u_x_data * np.sqrt(2 *
np.pi)))
    # y_real的先验
    log_p += np.sum(-0.5 * ((y_real - y_data) ** 2 / (u_y_data ** 2)) - np.log(u_y_data * np.sqrt(2 *
np.pi)))
    # y0_true的先验
    log_p += -0.5 * ((y0_true - y0) ** 2 / (u_y0 ** 2)) - np.log(u_y0 * np.sqrt(2 * np.pi))

    return log_p

def log_probability(theta):
    lp = log_prior(theta)
    if not np.isfinite(lp):
        return -np.inf

    x_real = theta[:6]
    y_real = theta[6:12]
    y0_true = theta[12]

    # 构造三次样条插值
    try:
        cs = CubicSpline(x_real, y_real, extrapolate=False)
    except:
        return -np.inf
```

```
    # 定义求解x0的函数
    def func(x):
        return cs(x) - y0_true

    x_min = x_real.min()
    x_max = x_real.max()

    # 检查端点处的符号
    try:
        f_min = func(x_min)
        f_max = func(x_max)
    except:
        return -np.inf

    if np.sign(f_min) == np.sign(f_max):
        return -np.inf  # 没有根

    # 寻找根
    try:
        x0_val = brentq(func, x_min, x_max)
    except:
        return -np.inf

    return lp

# 生成初始位置
n_walkers = 50

def generate_initial_positions(n_walkers):
    initial = np.zeros((n_walkers, n_params))
    for i in range(n_walkers):
        valid = False
        while not valid:
            x_real = x_data + np.random.normal(0, u_x_data / 10)
            if np.all(np.diff(x_real) > 0):
                valid = True
        y_real = y_data + np.random.normal(0, u_y_data / 10)
        y0_true = y0 + np.random.normal(0, u_y0 / 10)
        initial[i] = np.concatenate([x_real, y_real, [y0_true]])
    return initial

initial_positions = generate_initial_positions(n_walkers)

# 运行MCMC
sampler = emcee.EnsembleSampler(n_walkers, n_params, log_probability)

# Burn-in
print("Running burn-in...")
n_burn = 1000
state = sampler.run_mcmc(initial_positions, n_burn, progress=True)
sampler.reset()
```

```
# 主采样
n_steps = 5000
print("Running production...")
sampler.run_mcmc(state, n_steps, progress=True)

# 获取样本
samples = sampler.get_chain(flat=True)

# 计算x0的后验样本
x0_samples = []
for theta in samples:
    x_real = theta[:6]
    y_real = theta[6:12]
    y0_true = theta[12]

    try:
        cs = CubicSpline(x_real, y_real, extrapolate=False)
    except:
        continue

    def func(x):
        return cs(x) - y0_true

    x_min = x_real.min()
    x_max = x_real.max()

    try:
        f_min = func(x_min)
        f_max = func(x_max)
    except:
        continue

    if np.sign(f_min) == np.sign(f_max):
        continue

    try:
        x0 = brentq(func, x_min, x_max)
        x0_samples.append(x0)
    except:
        continue

x0_samples = np.array(x0_samples)

# 检查x0_samples是否为空
if len(x0_samples) == 0:
    raise ValueError("No valid x0 samples found.")

# 计算x0的中位数和不确定度
x0_median = np.median(x0_samples)
ux0 = np.std(x0_samples)

# 输出x0的值
print(f"Median x0: {x0_median:.4f}")
```

```
print(f"Estimated uncertainty u_x0: {ux0:.4f}")

# 可视化x0的后验分布
plt.figure(figsize=(10, 6))
plt.hist(x0_samples, bins=50, density=True, alpha=0.6, color='blue')
plt.title(f'Posterior Distribution of x0\n $u_{{{x0}}}$ = {ux0:.4f}', fontsize=16)
plt.xlabel('x0', fontsize=14)
plt.ylabel('Probability Density', fontsize=14)
plt.show()

# 绘制拟合曲线
plt.figure(figsize=(10, 6))
plt.errorbar(x_data, y_data, xerr=u_x_data, yerr=u_y_data, fmt='o', color='k', label='Data with Errors')

# 绘制部分后验样本的曲线
for i in np.random.choice(len(samples), 100):
    theta = samples[i]
    x_real = theta[:6]
    y_real = theta[6:12]
    try:
        # 构造三次样条插值
        cs = CubicSpline(x_real, y_real, extrapolate=False)
        x_plot = np.linspace(x_real.min(), x_real.max(), 100)
        y_plot = cs(x_plot)
        plt.plot(x_plot, y_plot, color='gray', alpha=0.1, label='Posterior Samples' if i == 0 else None)
    except:
        continue  # 如果插值失败，跳过该样本

# 绘制中位数的x0
plt.axvline(x0_median, color='r', linestyle='--', label=f'Median x0 = {x0_median:.3f}')
plt.axhline(y0, color='b', linestyle='--', label=f'y0 = {y0}')
plt.plot(x0_median, y0, 'r+', markersize=15, markeredgewidth=2, label='x0 Position')

# 添加标题和标签
plt.xlabel('x', fontsize=14)
plt.ylabel('y', fontsize=14)
plt.title('Spline Fits with MCMC Uncertainty', fontsize=16)

# 添加图例
plt.legend(loc='upper left', fontsize=12)

# 调整布局
plt.tight_layout()

# 显示图形
plt.show()

# 使用corner绘制参数的后验分布
corner_labels = [f'x_real_{i+1}' for i in range(6)] + [f'y_real_{i+1}' for i in range(6)] + ['y0_true']
corner.corner(samples, labels=corner_labels, truths=np.median(samples, axis=0))
plt.show()
```

低值试样三次样条插值拟合结果如图7.15所示，试样分析结果x=（3.345 ±0.343）ng/mL，即x值的不确定 $u(x)$ = 0.343 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.343=0.686 ng/mL，试样分析结果可以表示为（3.345 ± 0.686) ng/mL。

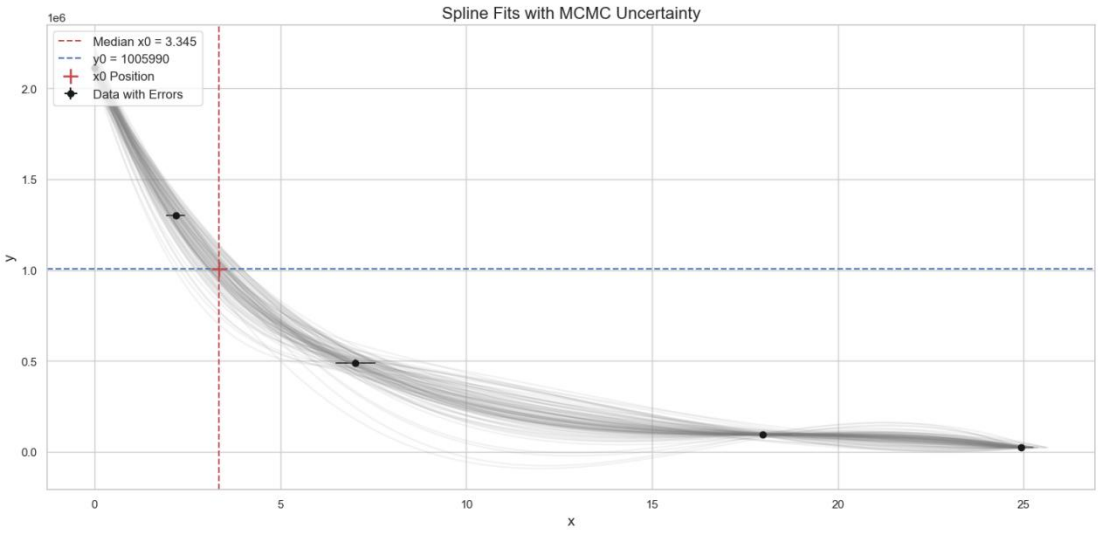

图7.15 低值试样三次样条插值拟合计算结果

低值试样三次样条插值拟合结果如图7.16所示，试样分析结果x=（5.215 ±0.426）ng/mL，即x值的不确定$u(x)$ = 0.426 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.426=0.852 ng/mL，试样分析结果可以表示为（5.215 ± 0.852) ng/mL。

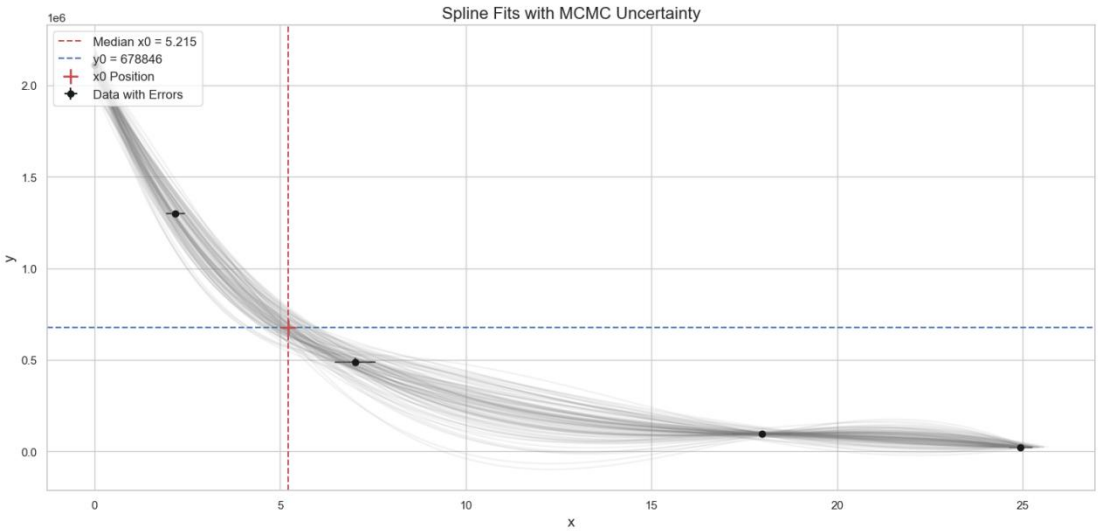

图7.16 高值试样三次样条插值拟合计算结果

3. 高敏C反应蛋白胶乳免疫比浊法结果不确定度的评定示例

3.1 实验方法

采用商品化高敏C反应蛋白测定试剂盒（胶乳免疫比浊法）高低两个水平的试样进行检测，使用试剂盒自带的6个水平的标准溶液绘制标准曲线，每个浓度的标准溶液重复检测2次，每个试样重复检测6次。

3.2 实验结果

胶乳免疫比浊法分析实验结果如表7.4所示。

表7.4高敏C反应蛋白测定试剂盒（胶乳免疫比浊法）检测结果

样品名称	浓度 mg/L	吸光度			吸光度平均值
空白	0	1.005	1.101	/	1.0530
标准 1	1.2875	1.185	1.209	/	1.1970
标准 2	2.575	1.336	1.373	/	1.3545
标准 3	5.15	1.425	1.442	/	1.4335
标准 4	10.30	1.511	1.496	/	1.5035
标准 5	20.60	1.552	1.535	/	1.5435
试样 1	/	1.409 1.398	1.391 1.402	1.387 1.390	1.3962
试样 2	/	1.493 1.489	1.488 1.494	1.491 1.490	1.4908

根据上述计算方式，标准溶液和试样的$u(x_i)$,$u(y_i)$和$u(y)$计算结果见表7.5。

表7.5 标准溶液和试样的$u(x_i)$,$u(y_i)$和$u(y)$计算结果

样品名称	x_i	$u(x_i)$	y_i	$u(y_i)$	$\bar{y}$	$u(\bar{y})$
空白	0	0	1.0530	0.042553	/	/
标准 1	1.2875	0.030657	1.1970	0.010638	/	/
标准 2	2.575	0.030933	1.3545	0.016401	/	/
标准 3	5.15	0.030411	1.4335	0.007535	/	/
标准 4	10.30	0.030586	1.5035	0.006649	/	/
标准 5	20.60	0.030439	1.5435	0.007535	/	/
样品 1	/	/	/	/	1.3962	0.001529
样品 2	/	/	/	/	1.4908	0.000423

3.2.1 四参数模型拟合$u(x)$和U的计算结果

通过Python编程计算$u(x)$，代码如下：

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import emcee
import corner
from scipy.optimize import curve_fit
from scipy.stats import norm

# Define the four-parameter model
def four_param_model(x, A, B, C, D):
    return D + (A - D) / (1 + (x/C)**B)

# Define the log-likelihood function
def log_likelihood(theta, x_data, y_data, u_x_data, u_y_data):
    A, B, C, D = theta
    model = four_param_model(x_data, A, B, C, D)
    sigma2 = u_y_data**2 + (B * (A - D) * (x_data**B) * u_x_data / (C**B * (1 +
(x_data/C)**B)**2))**2
    return -0.5 * np.sum((y_data - model)**2 / sigma2 + np.log(2 * np.pi * sigma2))

# Define the log-prior function
def log_prior(theta):
    A, B, C, D = theta
    if not all(0 < value < 10 for value in [A, B, C, D]):
        return -np.inf
    return 0.0

# Define the log-probability function
def log_probability(theta, x_data, y_data, u_x_data, u_y_data):
    lp = log_prior(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + log_likelihood(theta, x_data, y_data, u_x_data, u_y_data)

# Define the data
x_data = np.array([1e-6,1.2875,2.575,5.15,10.30,20.60])
y_data = np.array([1.0530,1.1970,1.3545,1.4335,1.5035,1.5435])
u_x_data = np.array([1e-6,0.030657,0.030933,0.030411,0.030586,0.030439 ])
u_y_data = np.array([0.042553,0.010638,0.016401,0.007535,0.006649,0.007535])
y0 = 1.3962
u_y0 = 0.001529

# Initial guess for the parameters
initial_guess = [0.169915, 1.064054, 6.244783, 2.156547]  # A, B, C, D

# Maximum likelihood estimation (MLE) using curve_fit
try:
    popt, pcov = curve_fit(four_param_model, x_data, y_data, p0=initial_guess, sigma=u_y_data,
absolute_sigma=True)
    A_fit, B_fit, C_fit, D_fit = popt
    print("MLE Fit Parameters (curve_fit):")
    print("A =", A_fit)
```

```
    print("B =", B_fit)
    print("C =", C_fit)
    print("D =", D_fit)

except RuntimeError as e:
    print(f"Error during curve_fit: {e}")
    A_fit, B_fit, C_fit, D_fit = initial_guess  # Fallback to initial guess
    pcov = np.eye(4) * 1e-6  # Dummy covariance matrix

# MCMC setup
nwalkers = 32
ndim = 4
nsteps = 10000
burnin = 2000

# Initialize the walkers
rng = np.random.default_rng()
initial = np.array(initial_guess)
pos = initial + 1e-4 * rng.standard_normal((nwalkers, ndim))

# Create the sampler
sampler = emcee.EnsembleSampler(nwalkers, ndim, log_probability, args=(x_data, y_data, u_x_data,
u_y_data))

# Run the MCMC sampler
sampler.run_mcmc(pos, nsteps, progress=True)

# Discard the burn-in samples
samples = sampler.get_chain(discard=burnin, flat=True)

# Analyze the MCMC results
A_mcmc, B_mcmc, C_mcmc, D_mcmc = np.mean(samples, axis=0)
print("\nMCMC Fit Parameters:")
print("A =", A_mcmc)
print("B =", B_mcmc)
print("C =", C_mcmc)
print("D =", D_mcmc)

# Corner plot
corner.corner(samples, labels=["A", "B", "C", "D"], truths=[A_fit, B_fit, C_fit, D_fit])
plt.suptitle("MCMC Parameter Distributions", y=0.98)
plt.show()  # Display the corner plot
plt.close()



# Function to find x0 and its uncertainty, now including u_y0
def find_x0(y0, u_y0, samples, x_data, num_y0_samples=100):
    x0_values = []
    A_samples = samples[:, 0]
    B_samples = samples[:, 1]
    C_samples = samples[:, 2]
    D_samples = samples[:, 3]
```

```
    rng = np.random.default_rng()  # Use a random number generator

    for A, B, C, D in zip(A_samples, B_samples, C_samples, D_samples):
        # Sample y0 values from a normal distribution
        y0_samples = norm.rvs(loc=y0, scale=u_y0, size=num_y0_samples, random_state=rng)  #
Sample y0
        for y0_sampled in y0_samples: #Iterate over the sampled y0 values
            # Use a root-finding method to solve for x0
            def equation(x):
                return four_param_model(x, A, B, C, D) - y0_sampled

            # Providing a bracket around the root can help the solver
            x_lower = min(x_data)
            x_upper = max(x_data)

            try:
                from scipy.optimize import brentq
                x0 = brentq(equation, x_lower, x_upper)
                x0_values.append(x0)
            except ValueError:
                # If brentq fails (e.g., no root within the bracket), return NaN
                x0_values.append(np.nan)
            except Exception as e:
                print(f"Error during root finding: {e}")
                x0_values.append(np.nan)

    x0_values = np.array(x0_values)
    x0_values = x0_values[~np.isnan(x0_values)]  # Filter out NaN values

    if len(x0_values) == 0:
        return np.nan, np.nan  # Return NaN if no valid x0 values are found

    x0_mean = np.mean(x0_values)
    x0_uncertainty = np.std(x0_values)
    return x0_mean, x0_uncertainty

# Find x0 and its uncertainty
x0, ux0 = find_x0(y0, u_y0, samples, x_data) #Pass u_y0 to find_x0

print("\nx0 =", x0)
print("ux0 =", ux0)

# Plot the data and the fit
plt.figure(figsize=(10, 8))
plt.errorbar(x_data, y_data, xerr=u_x_data, yerr=u_y_data, fmt="o", label="Data")

# Plot the best-fit curve
x_fit = np.linspace(min(x_data), max(x_data), 100)
y_fit = four_param_model(x_fit, A_mcmc, B_mcmc, C_mcmc, D_mcmc)
plt.plot(x_fit, y_fit, label="MCMC Best Fit")

# Plot x0 with crosshairs
if not np.isnan(x0):
```

```
    plt.axvline(x=x0, color='red', linestyle='--', label=f'x0 = {x0:.3f}')
    plt.axhline(y=y0, color='green', linestyle='--', label=f'y0 = {y0:.3f}')
    plt.errorbar(x0, y0, xerr=ux0, yerr=u_y0, fmt='+', color='purple', markersize=12, label='x0
Uncertainty')

plt.xlabel("x")
plt.ylabel("y")
plt.title("Four-Parameter Fit with x0")
plt.legend()
plt.grid(True)
plt.show()  # Display the fit plot
plt.close()


# Plot MCMC samples
plt.figure(figsize=(12, 8))
for i in range(ndim):
    plt.subplot(ndim, 1, i + 1)
    plt.plot(sampler.get_chain()[:, :, i], color="k", alpha=0.3)
    plt.ylabel(f"Parameter {['A', 'B', 'C', 'D'][i]}")
plt.xlabel("Step Number")
plt.suptitle("MCMC Sample Traces", y=0.98)
plt.tight_layout()
plt.show() # Display MCMC traces
plt.close()


# Plot posterior distributions
plt.figure(figsize=(12, 8))
for i in range(ndim):
    plt.subplot(ndim, 1, i + 1)
    plt.hist(samples[:, i], bins=50, density=True, alpha=0.6, color="skyblue")
    plt.xlabel(f"Parameter {['A', 'B', 'C', 'D'][i]}")
    plt.ylabel("Density")
    plt.title(f"Posterior Distribution of Parameter {['A', 'B', 'C', 'D'][i]}")
plt.suptitle("Posterior Distributions of Parameters", y=0.98)
plt.tight_layout()
plt.show() # Display posterior distributions
plt.close()
```

低值试样四参数拟合结果如图7.17所示，试样分析结果x=（3.92 ± 0.20）ng/mL，即x值的不确定 $u(x)$ = 0.20 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.20 = 0.40 ng/mL，试样分析结果可以表示为（3.92 ± 0.40）ng/mL。

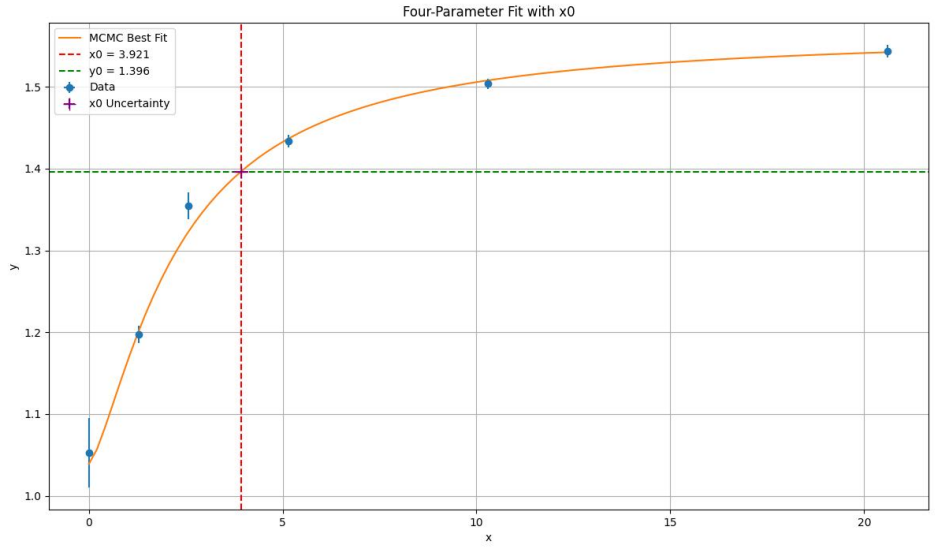

图7.17　低值试样四参数拟合结果

高值试样四参数拟合结果如图7.18所示，试样分析结果x=（8.48 ± 0.51）ng/mL，即x值的不确定 $u(x)$ = 0.51 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.51 = 1.1 ng/mL，试样分析结果可以表示为（8.5 ± 1.1）ng/mL。

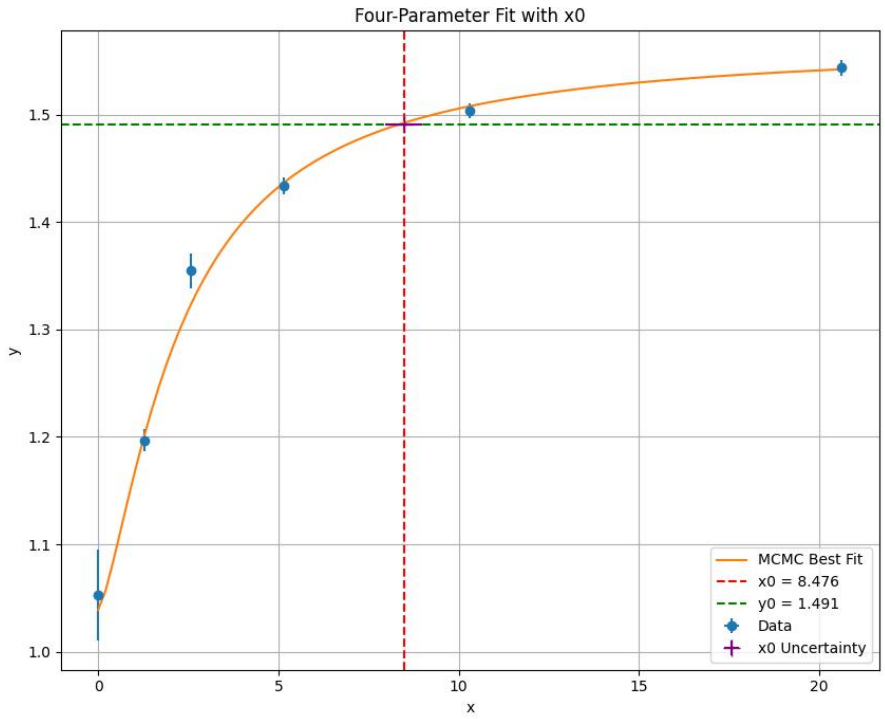

图7.18　高值试样四参数拟合结果

3.2.2 三次样条插值拟合$u(x)$和U的计算结果

通过Python编程计算$u(x)$，代码如下：

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline
from scipy.optimize import brentq
import emcee
import corner
import seaborn as sns

# 设置 seaborn 样式
sns.set(style="whitegrid")

# 定义数据
x_data = np.array([1e-6,1.2875,2.575,5.15,10.30,20.60])
y_data = np.array([1.0530,1.1970,1.3545,1.4335,1.5035,1.5435])
u_x_data = np.array([1e-6,0.030657,0.030933,0.030411,0.030586,0.030439 ])
u_y_data = np.array([0.042553,0.010638,0.016401,0.007535,0.006649,0.007535])
y0 = 1.3962
u_y0 = 0.001529

n_params = 13  # 6 x_real, 6 y_real, 1 y0_true

def log_prior(theta):
    x_real = theta[:6]
    y_real = theta[6:12]
    y0_true = theta[12]

    # 检查x_real是否递增
    if not np.all(np.diff(x_real) > 0):
        return -np.inf

    # 计算先验概率
    log_p = 0.0
    # x_real的先验
    log_p += np.sum(-0.5 * ((x_real - x_data) ** 2 / (u_x_data ** 2)) - np.log(u_x_data * np.sqrt(2 *
np.pi)))
    # y_real的先验
    log_p += np.sum(-0.5 * ((y_real - y_data) ** 2 / (u_y_data ** 2)) - np.log(u_y_data * np.sqrt(2 *
np.pi)))
    # y0_true的先验
    log_p += -0.5 * ((y0_true - y0) ** 2 / (u_y0 ** 2)) - np.log(u_y0 * np.sqrt(2 * np.pi))

    return log_p

def log_probability(theta):
    lp = log_prior(theta)
    if not np.isfinite(lp):
        return -np.inf
```

```
    x_real = theta[:6]
    y_real = theta[6:12]
    y0_true = theta[12]

    # 构造三次样条插值
    try:
        cs = CubicSpline(x_real, y_real, extrapolate=False)
    except:
        return -np.inf

    # 定义求解x0的函数
    def func(x):
        return cs(x) - y0_true

    x_min = x_real.min()
    x_max = x_real.max()

    # 检查端点处的符号
    try:
        f_min = func(x_min)
        f_max = func(x_max)
    except:
        return -np.inf

    if np.sign(f_min) == np.sign(f_max):
        return -np.inf  # 没有根

    # 寻找根
    try:
        x0_val = brentq(func, x_min, x_max)
    except:
        return -np.inf

    return lp

# 生成初始位置
n_walkers = 50

def generate_initial_positions(n_walkers):
    initial = np.zeros((n_walkers, n_params))
    for i in range(n_walkers):
        valid = False
        while not valid:
            x_real = x_data + np.random.normal(0, u_x_data / 10)
            if np.all(np.diff(x_real) > 0):
                valid = True
        y_real = y_data + np.random.normal(0, u_y_data / 10)
        y0_true = y0 + np.random.normal(0, u_y0 / 10)
        initial[i] = np.concatenate([x_real, y_real, [y0_true]])
    return initial

initial_positions = generate_initial_positions(n_walkers)
```

```
# 运行MCMC
sampler = emcee.EnsembleSampler(n_walkers, n_params, log_probability)

# Burn-in
print("Running burn-in...")
n_burn = 1000
state = sampler.run_mcmc(initial_positions, n_burn, progress=True)
sampler.reset()

# 主采样
n_steps = 5000
print("Running production...")
sampler.run_mcmc(state, n_steps, progress=True)

# 获取样本
samples = sampler.get_chain(flat=True)

# 计算x0的后验样本
x0_samples = []
for theta in samples:
    x_real = theta[:6]
    y_real = theta[6:12]
    y0_true = theta[12]

    try:
        cs = CubicSpline(x_real, y_real, extrapolate=False)
    except:
        continue

    def func(x):
        return cs(x) - y0_true

    x_min = x_real.min()
    x_max = x_real.max()

    try:
        f_min = func(x_min)
        f_max = func(x_max)
    except:
        continue

    if np.sign(f_min) == np.sign(f_max):
        continue

    try:
        x0 = brentq(func, x_min, x_max)
        x0_samples.append(x0)
    except:
        continue

x0_samples = np.array(x0_samples)

# 检查x0_samples是否为空
```

```
if len(x0_samples) == 0:
    raise ValueError("No valid x0 samples found.")

# 计算x0的中位数和不确定度
x0_median = np.median(x0_samples)
ux0 = np.std(x0_samples)

# 输出x0的值
print(f"Median x0: {x0_median:.4f}")
print(f"Estimated uncertainty u_x0: {ux0:.4f}")

# 可视化x0的后验分布
plt.figure(figsize=(10, 6))
plt.hist(x0_samples, bins=50, density=True, alpha=0.6, color='blue')
plt.title(f'Posterior Distribution of x0\n $u_{{{x0}}}$ = {ux0:.4f}', fontsize=16)
plt.xlabel('x0', fontsize=14)
plt.ylabel('Probability Density', fontsize=14)
plt.show()

# 绘制拟合曲线
plt.figure(figsize=(10, 6))
plt.errorbar(x_data, y_data, xerr=u_x_data, yerr=u_y_data, fmt='o', color='k', label='Data with Errors')

# 绘制部分后验样本的曲线
for i in np.random.choice(len(samples), 100):
    theta = samples[i]
    x_real = theta[:6]
    y_real = theta[6:12]
    try:
        # 构造三次样条插值
        cs = CubicSpline(x_real, y_real, extrapolate=False)
        x_plot = np.linspace(x_real.min(), x_real.max(), 100)
        y_plot = cs(x_plot)
        plt.plot(x_plot, y_plot, color='gray', alpha=0.1, label='Posterior Samples' if i == 0 else None)
    except:
        continue  # 如果插值失败，跳过该样本

# 绘制中位数的x0
plt.axvline(x0_median, color='r', linestyle='--', label=f'Median x0 = {x0_median:.3f}')
plt.axhline(y0, color='b', linestyle='--', label=f'y0 = {y0}')
plt.plot(x0_median, y0, 'r+', markersize=15, markeredgewidth=2, label='x0 Position')

# 添加标题和标签
plt.xlabel('x', fontsize=14)
plt.ylabel('y', fontsize=14)
plt.title('Spline Fits with MCMC Uncertainty', fontsize=16)

# 添加图例
plt.legend(loc='upper left', fontsize=12)

# 调整布局
plt.tight_layout()
```

```
# 显示图形
plt.show()

# 使用corner绘制参数的后验分布
corner_labels = [f'x_real_{i+1}' for i in range(6)] + [f'y_real_{i+1}' for i in range(6)] + ['y0_true']
corner.corner(samples, labels=corner_labels, truths=np.median(samples, axis=0))
plt.show()
```

低值试样三次样条差值拟合结果如图7.19所示，试样分析结果x=（3.15 ± 0.35）ng/mL，即x值的不确定$u(x)$ = 0.35 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.35 = 0.70 ng/mL，试样分析结果可以表示为（3.15 ± 0.70）ng/mL。

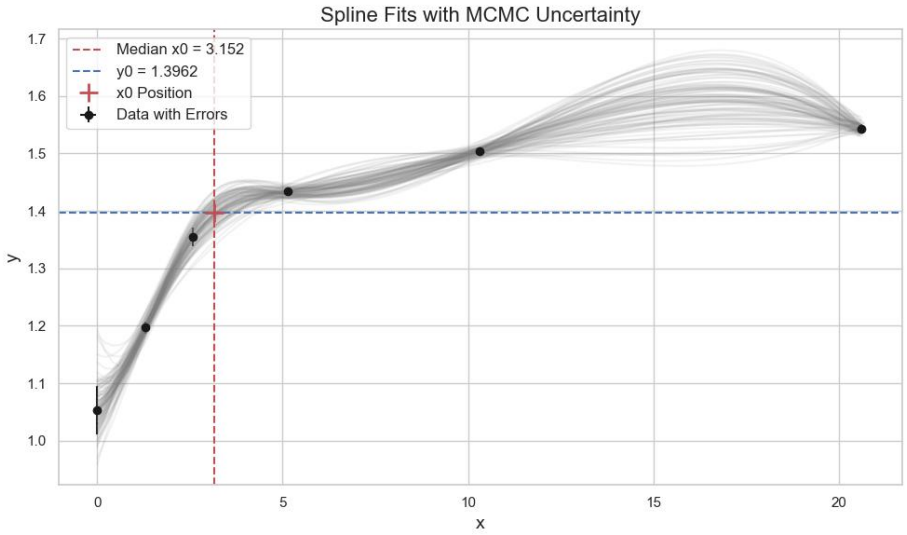

图7.19　低值试样三次样条插值拟合结果

高值试样三次样条差值拟合结果如图7.20所示，试样分析结果x=（9.63 ± 0.96）ng/mL，即x值的不确定$u(x)$ = 0.96 ng/mL，取扩展因子k=2，则扩展不确定度U=2×0.96 = 2.0 ng/mL，试样分析结果可以表示为（9.6 ± 2.0）ng/mL。

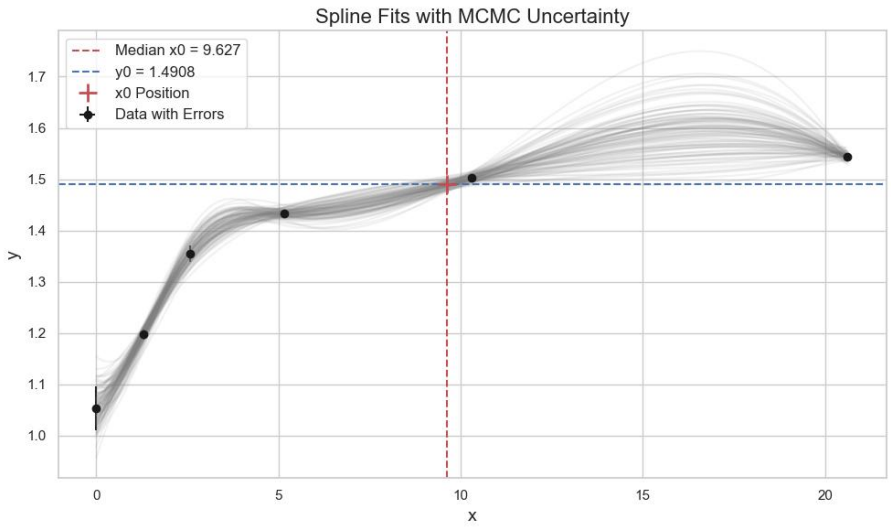

图7.20　高值试样三次样条插值拟合结果

八.与现行相关法律、法规、规章及相关标准，特别是强制性标准的协调性

本规范编制严格遵照 GB/T 1.1-2020 的要求和规定，确保标准的版式、格式、文本结构、表述方式等方面规范和统一。同时在术语及相关条文的表述上将严格与国家现行相关法律、法规、规章及相关标准，特别是强制性标准的协调性。

九．标准性质的建议说明

本标准建议为推荐性标准。

《免疫分析 测量结果的不确定度评定》编制组

2025 年 5 月 31 日