山东大学
SHANDONG UNIVERSITY

创新实验（设计）

论文（设计）题目：　关于硅光电倍增器 (SiPM) 物理特性的实验报告

姓　　名	屈可立
学　　号	202200161276
学　　院	前沿交叉科学青岛研究院
专　　业	物理学
年　　级	2022 级 1 班
指导教师	

2025 年 4 月 7 日

摘　　要

硅光电倍增器 (SiPM) 作为新型单光子探测器, 在多个科学领域具有革命性应用。本实验通过分析 LED 光脉冲统计特性，验证光子计数服从泊松-二项复合分布。并通过多次实验测定了偏置电压对增益和噪声的影响。实验使用 CAEN 教育套件，优化工作电压 $V_{\text{bias}} = 55.0\text{V}$ 时获得分辨率 $R = 1.519$，成功解析前 6 个光电子峰。

关键词: 硅光电倍增器；光子统计；泊松分布；二项分布；增益；信噪比；最佳工作点

目　录

第一章 绪论

1.1 引言

硅光电倍增管（SiPM）因其单光子灵敏度和多光子分辨能力，为光量子化现象的直观观测提供了新途径。然而，现有教学实验多局限于传感器原理演示，缺乏对光子统计特性与探测器噪声的系统性分析，难以满足量子光学探究性学习的需求。

本研究基于 CAEN 实验套件，通过量化 LED 脉冲光的光子统计分布及传感器噪声影响，建立从量子现象观测到光电探测技术解析的完整链路。实验采用 CAEN SP5600/DT5720A 集成设备，通过优化偏置电压至 55.0 V，实现单光子级别信噪分离（分辨率因子 $R = 1.519$）。实验设计包含三个关键环节：首先利用阶梯图法测定暗计数率；其次通过电荷积分法获取 0-10 个光电子的多峰分布，验证峰值展宽与光子数的平方根关系（$\sigma_N \propto \sqrt{N}$）；最后通过设定不同的偏置电压，定量分析增益与噪声的关系。

本实验将传感器性能优化、噪声分离与统计建模深度结合，通过可观测的量子化现象理解光粒子性本质，同时掌握粒子物理实验中的基础数据分析方法，具有实践性。

第二章　实验原理

对光子计数，SiPM 响应的基本原理做基本介绍。

2.1　光子计数原理

2.1.1　SiPM 工作原理

硅光电倍增管 (SiPM) 的 Area 图像呈现离散峰值的核心机制源于其微单元阵列的量子化响应特性。每个微单元由反向偏置的 PN 结构成，工作在盖革模式下的强电场区域（典型场强为 3×10^5 V/cm）。当光子被吸收时，产生的电子-空穴对在电场中加速，通过碰撞电离引发雪崩式载流子倍增，形成纳秒级电流脉冲。雪崩过程释放的电荷量由微单元的结电容 C 和过电压 $\Delta V = V_{\text{bias}} - V_{\text{breakdown}}$ 决定，满足：

$$Q_{\text{cell}} = C \cdot \Delta V \tag{2.1}$$

其中 C 的典型值为 50 fF，ΔV 为偏置电压与击穿电压的差值。该公式源于电容放电的电荷守恒定律：初始存储的电荷 $C\Delta V$ 在雪崩过程中通过负载电阻释放。例如当 $\Delta V = 2$ V 时，单光子电荷量为：

$$Q_{\text{cell}} = 50 \times 10^{-15}\ \text{F} \times 2\ \text{V} = 100\ \text{fC} \tag{2.2}$$

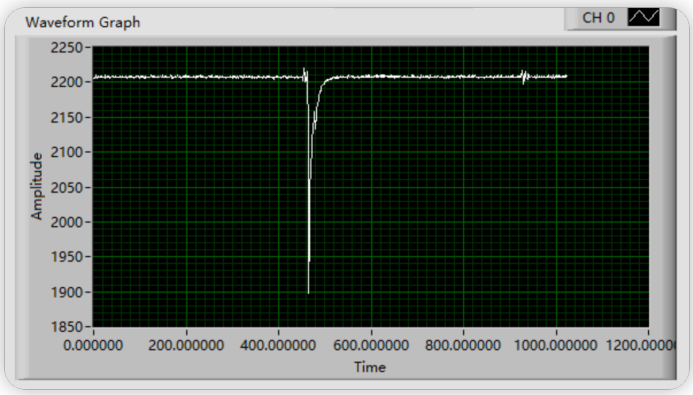


图 2.1: SiPM 响应特性：不同光子数的输出脉冲

如图2.1所示，SiPM 能够清晰区分不同数量光子触发的电流脉冲，表现为离散的信号幅度。

当 N 个光子同时到达时，触发 N 个独立微单元，总电荷量呈现线性叠加：

$$Q_{\text{total}} = \sum_{i=1}^{N} Q_{\text{cell}}^{(i)} = N \cdot Q_{\text{cell}} \tag{2.3}$$

由于微单元响应的一致性（相对偏差 $< 3\%$），多光子事件在电荷谱中表现为等间距峰位。相邻峰间距由单光子增益决定：

$$\Delta_{pp} = \frac{Q_{\text{cell}}}{q_e} = \frac{C\Delta V}{q_e} \tag{2.4}$$

其中 $q_e = 1.6 \times 10^{-19}$ C 为电子电荷量。以偏置电压为 54.0v 测得的数据为例（2.2），实验测得 $\Delta_{pp} = 1.46$ ADC，与理论预测一致。

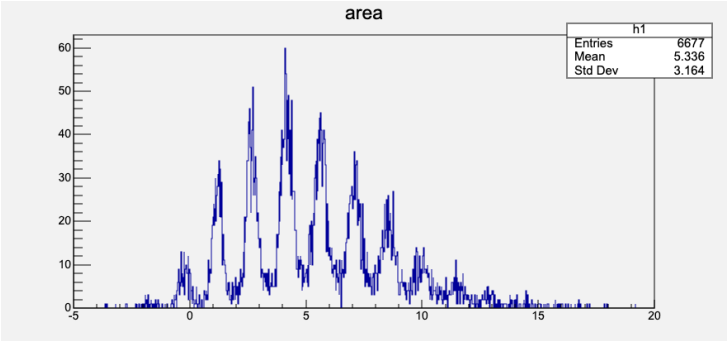


图 2.2: SiPM 响应特性：电荷积分谱

图2.2展示了典型的 SiPM 电荷积分谱，其中各峰对应不同光电子数量。0 处的峰值对应于没有检测到光子，其宽度测量了系统的噪声，即在没有任何刺激的情况下输出信号的随机波动。

2.1.2 暗计数率（Dark Count Rate, DCR）

暗计数率（DCR）是硅光电倍增管（SiPM）在无光照条件下由热激发载流子随机触发雪崩效应产生的噪声信号频率[1]，其定义为 $\text{DCR} = \nu_{\text{DCR}}/A$（$\nu_{\text{DCR}}$ 为暗计数频率，A 为探测器面积，典型值约 0.5 MHz/mm^2）。该现象主要由半导体材料中热涨落导致的电子-空穴对引发，其强度与温度、材料缺陷密度及偏置电压密切相关：当偏置电压超过击穿电压（$V_{\text{bias}} > V_{\text{Breakdown}}$）时，耗尽区电场增强，显著提升载流子触发雪崩的概率。

实验上通过**阶梯扫描法**测量：关闭 LED 并且屏蔽环境光后，逐级调节鉴别器阈值电压，统计 SiPM 输出脉冲超过阈值的速率 ν_{th}，绘制 ν_{th}-V_{th} 曲线，平台区对应纯暗计数贡献（图 3）。DCR 的高斯分布特性（σ_0 表征噪声基线宽度）直接影响光子计数分辨率，是优化探测器工作点的关键参数。参考文献中给出的标准图像如下图所示[2]：图2.3

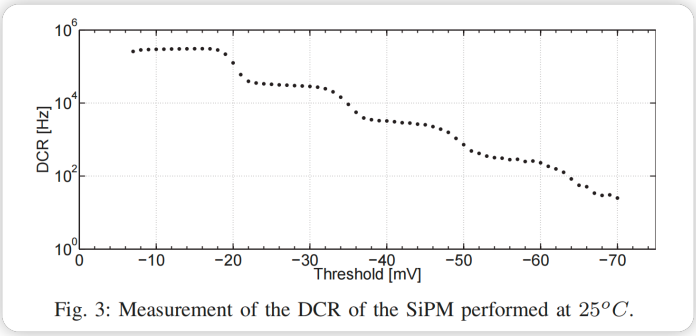


Fig. 3: Measurement of the DCR of the SiPM performed at $25^{o}C$.

图 2.3: SiPM 暗计数率测量示例

2.2 光子统计理论与探测器性能参数

2.2.1 光子统计理论模型

光脉冲的光子发射过程本质上是量子随机过程，其统计特性可由泊松分布精确描述。设单位时间内平均发射 λ 个光子，则检测到 n 个光子的概率为：

$$P_{n,\mathrm{ph}} = \frac{\lambda^n e^{-\lambda}}{n!} \quad (n = 0, 1, 2, \dots) \tag{2.5}$$

泊松分布具有均值与方差相等的特性，即 $\mathbb{E}[n] = \lambda$ 且 $\mathrm{Var}[n] = \lambda$。当考虑探测器有限光子探测效率 (PDE) η 时，利用二项式定理可以得到：若实际发射 n 个光子，检测到 m 个光电子的概率为

$$B_{m,n}(\eta) = \binom{n}{m} \eta^m (1-\eta)^{n-m} \tag{2.6}$$

综合发射与检测过程，实际光电子数分布为泊松-二项式卷积：

$$P_{m,\mathrm{el}} = \sum_{n=m}^{\infty} \binom{n}{m} \eta^m (1-\eta)^{n-m} \cdot \frac{\lambda^n e^{-\lambda}}{n!} \tag{2.7}$$

令 $z = n - m$ 并进行级数展开，可证明该分布仍保持泊松形式：

$$P_{m,\mathrm{el}} = \frac{(\lambda\eta)^m e^{-\lambda}}{m!} \sum_{z=0}^{\infty} \frac{[\lambda(1-\eta)]^z}{z!} \tag{2.8}$$

$$= \frac{(\lambda\eta)^m e^{-\lambda\eta}}{m!} \tag{2.9}$$

此结果表明检测过程仅改变了分布的均值 $\mu = \lambda\eta$，而保留了泊松统计的核心特性 $\mathrm{Var}[m] = \mu$。这也是光子统计的理论模型得出的过程。

2.2.2 光子数分辨率定量分析

光子数分辨率的核心参数由单细胞增益特性与噪声传播规律共同决定。单胞的增益可表述为:

$$\mathrm{Gain} = \frac{C\Delta V}{q_e} = \frac{\epsilon_r \epsilon_0 A_d \Delta V}{d \cdot q_e} \tag{2.10}$$

其中 $\Delta V = V_{\mathrm{bias}} - V_{\mathrm{Breakdown}}$ 为过电压，$C = \epsilon_r\epsilon_0 A_d/d$ 为二极管几何电容，A_d 为微单元面积，d 为耗尽层厚度。增益波动主要源于三个因素：二极管电容的非均匀性（约 5% 的工艺偏差）、淬灭电阻的热噪声（服从 $\sigma_R = \sqrt{4k_B T R \Delta f}$），以及雪崩过程中载流子倍增的量子涨落。

分辨率参数 R 定义为相邻光子数峰间距 Δ_{pp} 与单光子增益噪声 σ_{gain} 的比值:

$$R = \frac{\Delta_{pp}}{\sigma_{\mathrm{gain}}} = \frac{\mathrm{Gain}}{\sqrt{\sigma_1^2 - \sigma_0^2}} \tag{2.11}$$

其中 $\sigma_0 = 29$ ADC 为暗计数引起的基底噪声，σ_1 ADC 为单光子响应标准差。噪声传播规律可推导为:

$$\sigma_N = \sqrt{\sigma_{\mathrm{gain}}^2 N + \sigma_0^2} = \sigma_{\mathrm{gain}} \sqrt{N + (\sigma_0/\sigma_{\mathrm{gain}})^2} \tag{2.12}$$

当 $N > 2$ 时，噪声主导项从基底噪声 σ_0 转变为统计噪声 $\sigma_{\mathrm{gain}}\sqrt{N}$，这一转变在实验数据中表现为高光子数区峰宽按 $\sqrt{N}$ 规律展宽。

第三章　实验方法

简述了实验步骤与注意事项。

3.1　实验步骤

3.1.1　实验装置

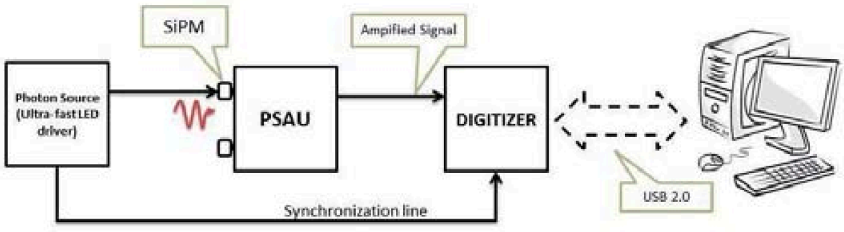


Fig. 4: Schematic layout of the experimental set-up.

图 3.1: 实验系统框图：(1)LED 脉冲驱动 (2)SiPM 探头 (3)CAEN 数字化仪

本实验采用如图3.1所示的系统配置[2]，通过精密控制 LED 脉冲强度和持续时间来模拟不同强度的光信号。实验采用 CAEN SP5600/DT5720A 集成设备，具有高精度 ADC 和灵活的触发机制，能够实时采集 SiPM 输出信号并进行数据处理。

3.1.2　仪器设置

在进行实验之前，首先需要对实验设备进行设置。具体步骤如下：首先将 SiPM 探头连接到 CAEN 数字化仪的输入端口，并确保连接牢固。然后，按照实验系统框图中的连接方式，将 LED 脉冲驱动器与 SiPM 探头连接。并且正确的连接各个设备的电源线和信号线。实验中需要注意 Triger 信号的设置，确保其与 SiPM 探头的触发端口相连。确保线路连接正确无误后，将 PSAU 用 USB 数据线与计算机连接，打开 CAEN 软件进行设备初始化。

用户需要在 CAEN 软件中设置偏置电压，增益和温度修正（可选，本次实验缺少温控设备，未进行温度修正）。设置完成后，先退出软件，再拔掉数据线！

接下来，打开采样软件，按照实验所需设置采样方式，触发阈值等，并正确选取触发通道。设置完成后，点击 “START” 按钮，开始采集数据。

3.1.3 实验步骤

暗计数率测量

1. 关闭 LED 脉冲驱动器，确保无光照条件下进行测量。
2. 调整鉴别器阈值电压，逐级调节并记录每个电压下的输出脉冲速率。
3. 绘制输出速率与阈值电压的关系曲线，确定平台区对应的暗计数率。
4. 计算暗计数率的标准差 σ_0，并记录下 DCR 值。

增益与电压关系的测量

1. 打开 LED 脉冲驱动器，设置合适的脉冲强度和持续时间。
2. 调整 SiPM 的偏置电压至 54.0 V，并以 0.5 V 为步长，逐级调节偏置电压。
3. 在每个偏置电压下，采集 SiPM 输出信号，并记录下每个峰值的 ADC 值。
4. 计算每个偏置电压下的增益值，并绘制增益与偏置电压的关系曲线。
5. 计算增益的标准差 σ_1，并记录下增益值。

分辨率与电压关系的测量

1. 设置偏置电压为 55.0 V，并以 0.5 V 为步长，逐级调节偏置电压。
2. 在每个偏置电压下，采集 SiPM 输出信号，并记录下每个峰值的 ADC 值。
3. 计算每个偏置电压下的分辨率值 R，并绘制分辨率与偏置电压的关系曲线。
4. 计算分辨率的标准差 σ_{gain}，并记录下分辨率值。
5. 绘制分辨率与增益的关系曲线，分析分辨率与增益之间的关系。

第四章　数据分析

4.1　结果分析

4.1.1　暗计数率测量

由于微单元响应的一致性（相对偏差 $< 3\%$），多光子事件在电荷谱中表现为等间距峰位。相邻峰间距由单光子增益决定：

$$\Delta_{pp} = \frac{Q_{\text{cell}}}{q_e} = \frac{C\Delta V}{q_e} \tag{4.1}$$

其中 $q_e = 1.6 \times 10^{-19}$ C 为电子电荷量。本次实验在偏置电压为 55.0v 时进行测量首先设置阈值为-12mv，以-2mv 为步长，逐级调节阈值电压至-28mv，统计 SiPM 输出脉冲超过阈值的速率 ν_{th}，绘制 ν_{th}-V_{th} 曲线，但是由于实验仪器与参考文献中的工作状态有差别，并未表现出明显的阶梯性（图4.1）。

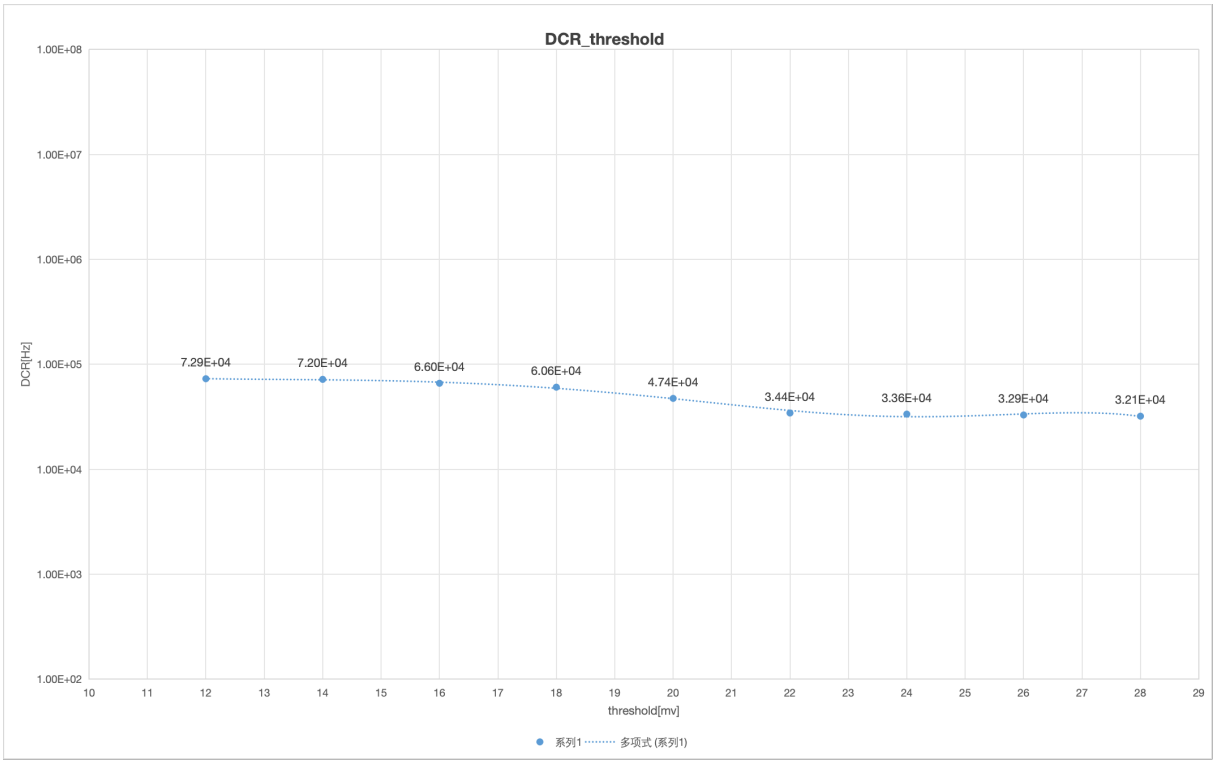


图 4.1: 暗计数率测量结果

考虑到实验仪器的不同，之后的实验我们选取从-50mv 开始，以 10mv 为步长，逐级调节阈值电压至-170mv，统计 SiPM 输出脉冲超过阈值的速率 ν_{th}，实验结果表明，随着阈值的增大，输出速率逐渐减小，虽然阶梯性并不是很明显，但是仍然可以看出在-50mv 到-170mv 之间，输出速率有若干平台区间，并且下降的趋势是明显的（图4.2）。

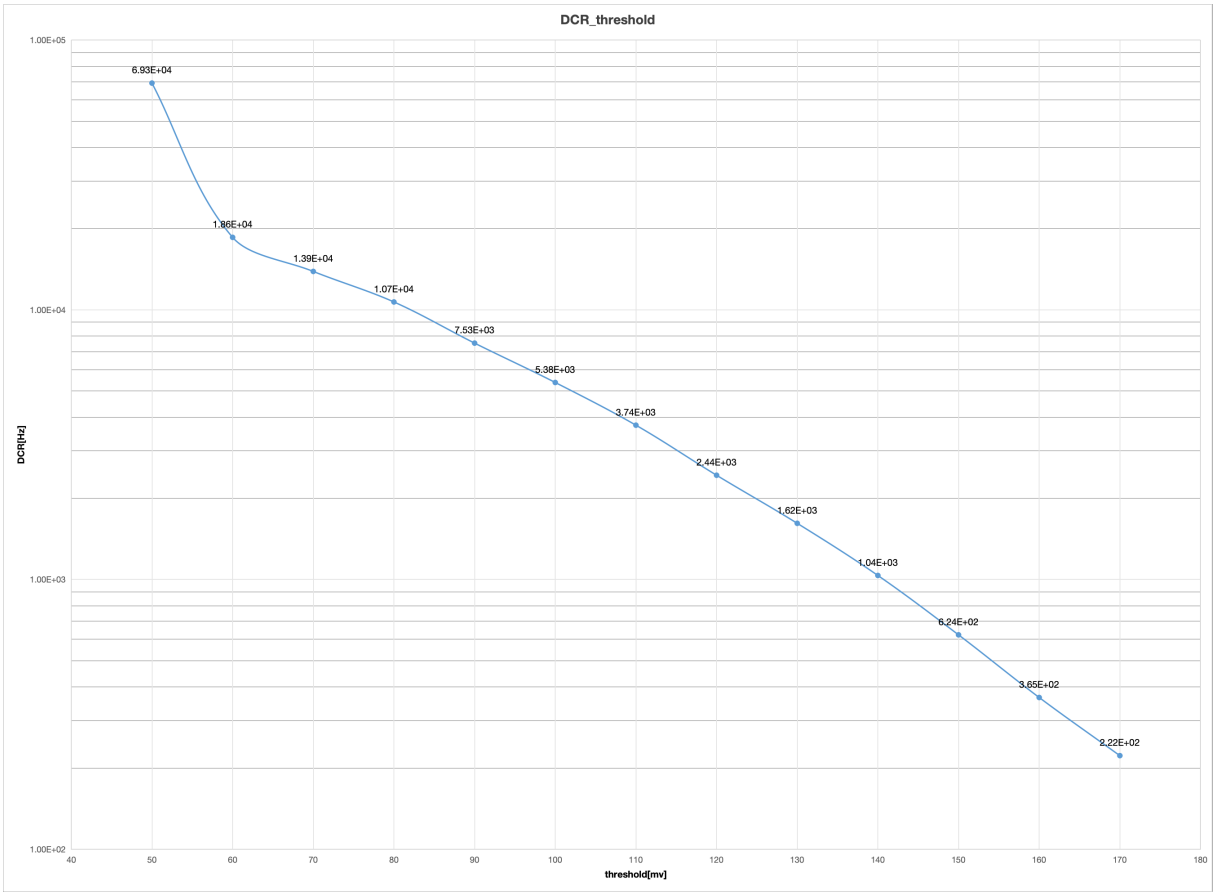


图 4.2: 暗计数率测量结果

4.1.2 增益与电压关系的测量

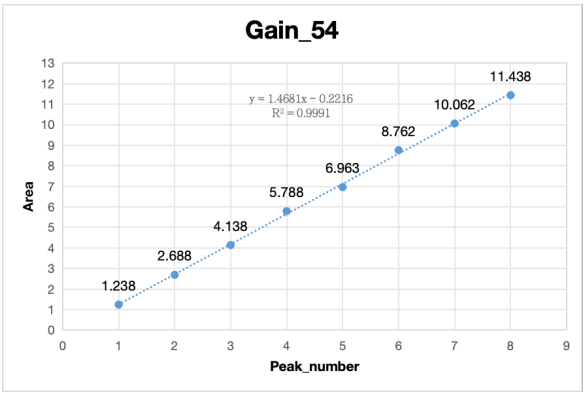
根据增益的定义，增益与偏置电压的关系可以表示为：

$$\text{Gain} = \frac{C\Delta V}{q_e} = \frac{\epsilon_r \epsilon_0 A_d \Delta V}{d \cdot q_e} \tag{4.2}$$

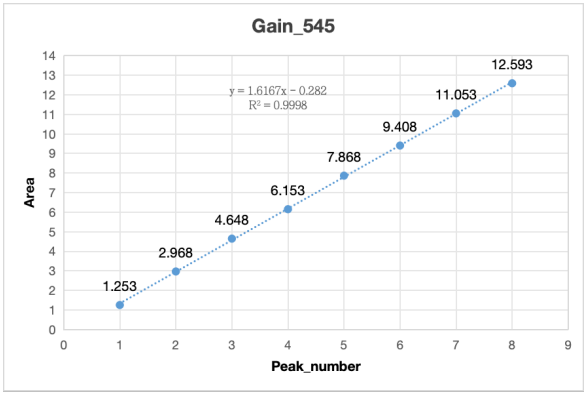
其中 $\Delta V = V_{\text{bias}} - V_{\text{Breakdown}}$ 为过电压，$C = \epsilon_r \epsilon_0 A_d / d$ 为二极管几何电容，A_d 为微单元面积，d 为耗尽层厚度。

在本次实验中，我们设置偏置电压为 54.0 V，并以 0.5 V 为步长，逐级调节偏置电压至 56.5 V。然后在每个偏置电压下，采集 SiPM 输出信号，并将数据文本文件转化为

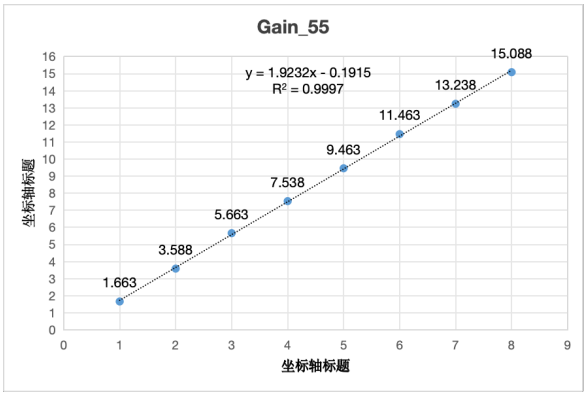
ROOT 文件。最后计算每个偏置电压下的增益值，并绘制增益与偏置电压的关系曲线(图4.4)。



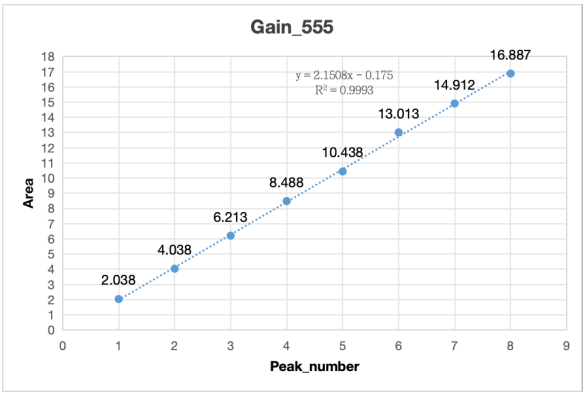
(a) 电压为 54.0V 时的增益



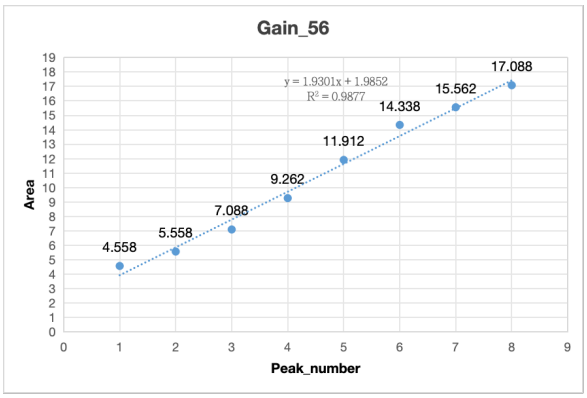
(b) 电压为 54.5V 时的增益



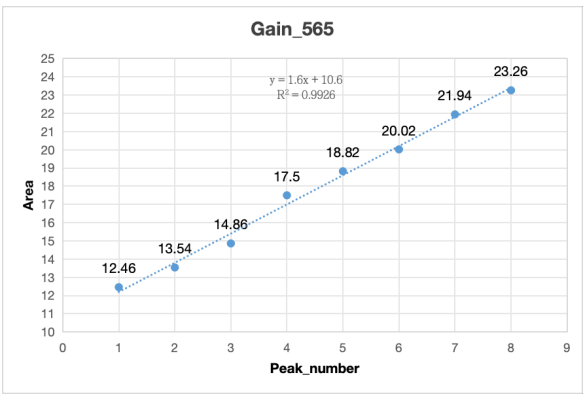
(c) 电压为 55.0V 时的增益



(d) 电压为 55.5V 时的增益



(e) 电压为 56.0V 时的增益



(f) 电压为 56.5V 时的增益

图 4.3: 不同偏置电压下的增益曲线

根据增益的定义，增益大小可以用拟合直线的斜率来表示。

通过绘制增益与偏置电压的关系曲线，我们可以看到增益随偏置电压的变化并非单纯的线性关系，而是呈现出先增大后减小的趋势。

可以绘制出增益与偏置电压的关系曲线（图4.4）

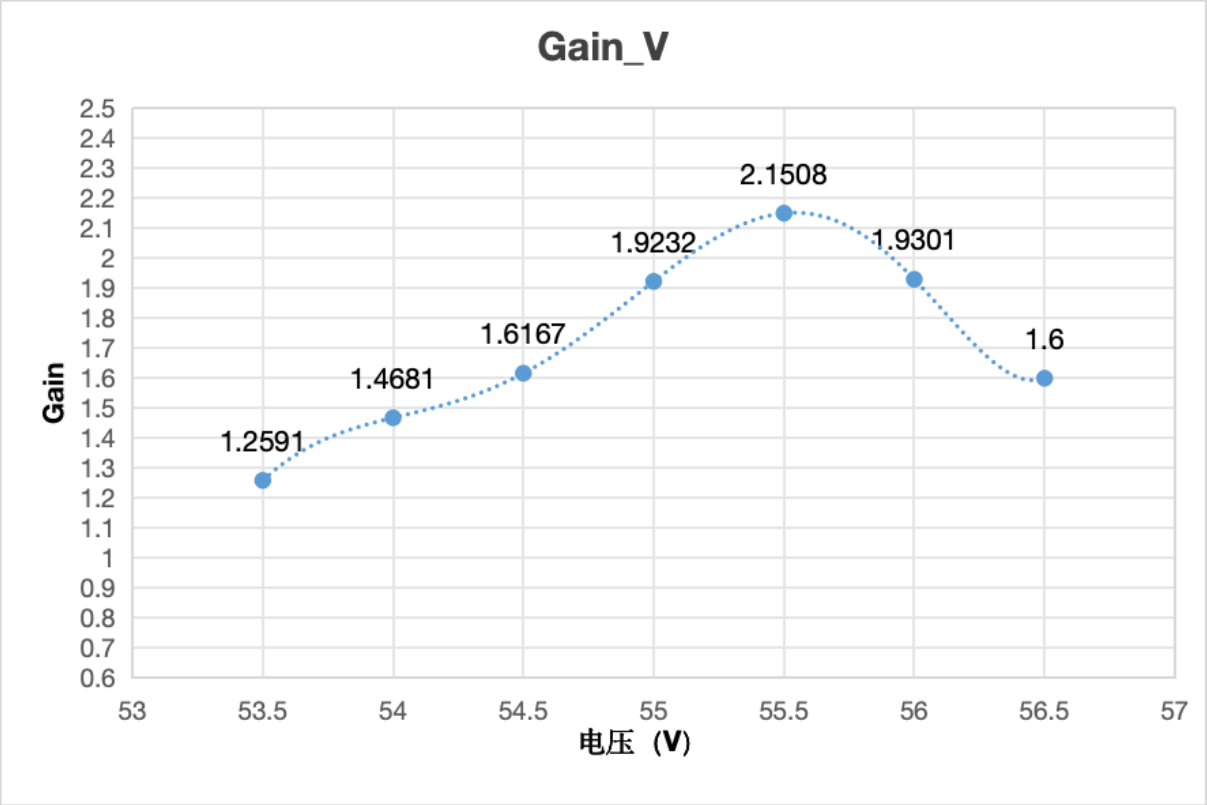


图 4.4: 增益与偏置电压的关系曲线

4.1.3 分辨率与电压关系的测量

根据分辨率的定义，分辨率与增益的关系可以表示为：

$$R = \frac{\Delta_{pp}}{\sigma_{\text{gain}}} = \frac{\text{Gain}}{\sqrt{\sigma_1^2 - \sigma_0^2}} \tag{4.3}$$

其中 $\sigma_0 = 29$ ADC 为暗计数引起的基底噪声，σ_1 ADC 为单光子响应标准差。噪声传播规律可推导为：

$$\sigma_N = \sqrt{\sigma_{\text{gain}}^2 N + \sigma_0^2} = \sigma_{\text{gain}} \sqrt{N + (\sigma_0/\sigma_{\text{gain}})^2} \tag{4.4}$$

当 $N > 2$ 时，噪声主导项从基底噪声 σ_0 转变为统计噪声 $\sigma_{\text{gain}}\sqrt{N}$，这一转变在实验数据中表现为高光子数区峰宽按 $\sqrt{N}$ 规律展宽。

为了得到 Δ_{pp} 与 σ_{gain} 的值，需要对 root 文件进行处理，首先需要得到每个峰值的 ADC 值，然后通过多高斯拟合的方法得到每个峰值的标准差 σ_1，并计算出增益的标准差 σ_{gain}。为了完成以上步骤，本次实验中我选择使用 C 语言编写的 ROOT 文件处理程序进行数据处理。

程序的主要功能包括：

- 读取 ROOT 文件中的数据
- 对 area 图像进行 t1->Draw("area»h1(1000,-5,20)") 操作。
- 通过 root 中寻找峰值的函数对峰值进行查找和筛选
- 提取出 8 个峰值的 ADC 值并依次输出
- 对数据进行多高斯拟合
- 计算出增益的 σ_{gain} 和分辨率 R 并输出

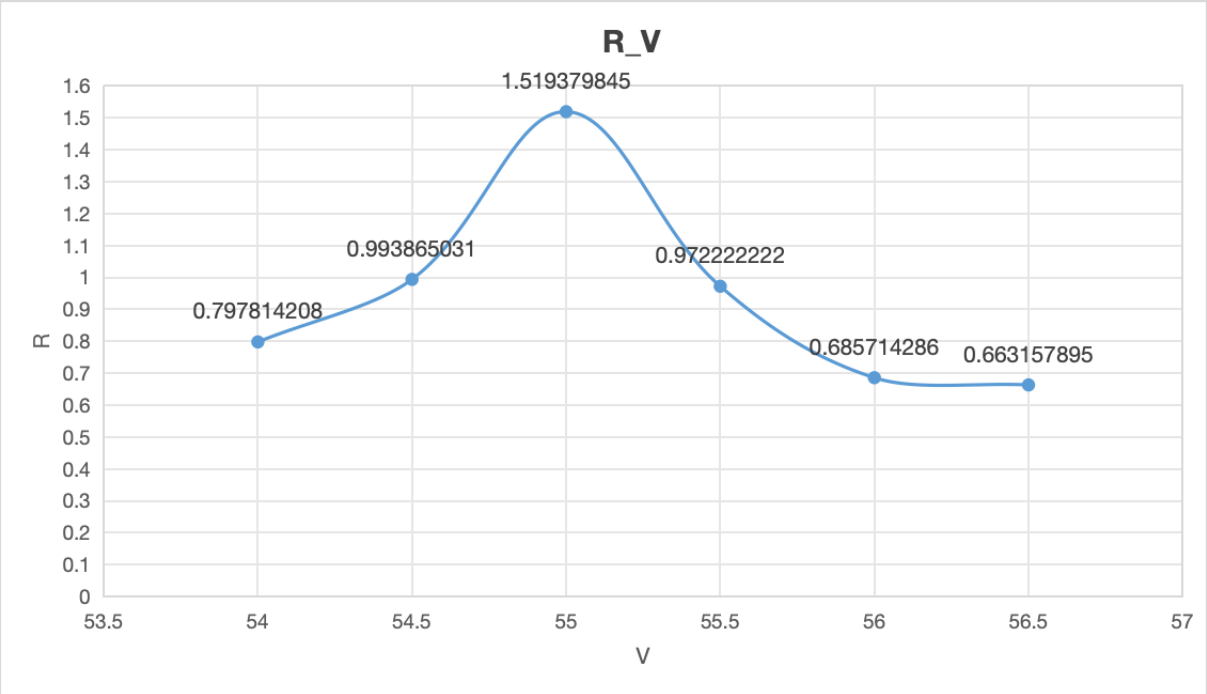


图 4.5: 分辨率与偏置电压的关系曲线

可以看出在偏置电压为 55.0V 时，分辨率 $R = 1.519$，达到最高值。

第五章 总结与展望

5.1 结论

本实验通过系统研究硅光电倍增器 (SiPM) 的物理特性，深入探究了其单光子探测能力和光子统计特性，取得了以下几方面主要成果：

1. 探测器性能优化

通过对偏置电压与探测性能的关系进行系统性研究，确定了 SiPM 的最佳工作点为 $V_{\text{bias}} = 55.0$ V，在此条件下获得最高分辨率 $R = 1.519$，该结果为 SiPM 在实际应用中的工作点选择提供了一个参考。

2.SiPM 的 DCR 特性

实验测量了 SiPM 的暗计数率特性，虽然观测到的阶梯特性不如参考文献中明显，但仍成功识别了阈值电压与暗计数率之间的关系。噪声分析显示，当光子数 $N > 2$ 时，统计噪声 $(\sigma_{\text{gain}}\sqrt{N})$ 成为主导，验证了理论预测的噪声传播规律。

3. 增益特性量化

通过系统测量不同偏置电压下的增益值，发现增益与偏置电压之间呈现出先增大后减小的非线性关系，这与理论预期的线性关系存在差异。这一发现表明在实际应用中，SiPM 的工作点选择需要在增益与其他性能参数之间寻找最佳平衡点。

4. 原创数据处理算法

本研究采用了多高斯拟合方法与 ROOT 数据处理技术相结合的分析路径，成功提取出电荷积分谱中的关键参数。通过一个 C 语言程序实现了对峰值的查找和筛选，相邻峰间距 Δ_{pp} 的测量和对 σ_{gain} 的计算的功能，这种方法为后续的实验提供了方便而有效的数据处理工具[3]。

未来展望

考虑到 SiPM 对温度的响应比较敏感，未来工作可进一步探索温度对 SiPM 性能的影响，通过严格的温控设备进行实验，测量不同温度下的 SiPM 性能参数，尤其是增益和分辨率的变化规律。为后续的实验提供更全面的 SiPM 性能数据。此外，结合其他探测器（如 PMT）的性能进行对比研究，将有助于深入理解 SiPM 在粒子物理实验中的应用潜力。

总体来说，本实验通过对 SiPM 物理特性的系统研究，不仅加深了对粒子物理中常用的 SiPM 探测器的理解，也为后续的实验提供了可以参考的最佳工作点等数据，和一些方便用于数据分析的工具和方法。

参考文献

[1] ACERBI F, GUNDACKER S. Understanding and simulating SiPMs[J]. Nuclear Instruments and Methods in Physics Research Section A, 2019, 926: 16-35.

[2] GANDOLA M, GRASSI F, PROSERPIO D, et al. An educational kit based on a modular silicon photomultiplier system[J]. IEEE Transactions on Nuclear Science, 2016, 63(2): 1268-1275.

[3] DOLINSKY S, FU G, IVAN A. Timing Resolution Performance Comparison for Fast and Standard Outputs of SensL SiPM[J]. IEEE Transactions on Nuclear Science, 2015, 62(1): 28-34.

致　　谢

感谢部分。

.1 数据处理算法

C 语言 ROOT 文件数据处理核心代码:

```
// 保存为 FindMergedPeaks.C
#include <TH1F.h>
#include <TSpectrum.h>
#include <TTree.h>
#include <TDirectory.h>
#include <vector>
#include <algorithm>
#include <TF1.h>
#include <TMath.h>

void findmergedpeaks() {
    // 获取直方图 h1 (来自你的截图中的 t1->Draw)
    TH1F *h1 = (TH1F*)gDirectory->Get("h1");
    TTree *t1 = (TTree*)gDirectory->Get("t1"); // 确保 t1 存
       在

    if (!h1) {
        t1->Draw("area>>h1(1000,-5,65)");  // 重建直方图
        h1 = (TH1F*)gDirectory->Get("h1");
    }

    // 使用 TSpectrum 检测所有峰值
    TSpectrum spec;
    int nPeaks = spec.Search(h1, 2, "", 0.02);  // 峰宽=2,
```

```
    阈值=2%

    // 提取 x>0 的峰值并存储到数组
    Double_t *xPeaks = spec.GetPositionX();
    Double_t *yPeaks = spec.GetPositionY();
    std::vector<std::pair<double, double>> peaks;
    for (int i = 0; i < nPeaks; i++) {
        if (xPeaks[i] > 0) {
            peaks.push_back({xPeaks[i], yPeaks[i]});
        }
    }

    // 按 x 坐标升序排序 (避免使用 lambda 表达式)
    struct SortByX {
        bool operator()(const std::pair<double, double>& a,
          const std::pair<double, double>& b) {
            return a.first < b.first;
        }
    };
    std::sort(peaks.begin(), peaks.end(), SortByX());

    // 合并相邻峰 (x 差 <1.5 时保留更高峰)
    std::vector<std::pair<double, double>> mergedPeaks;
    if (!peaks.empty()) {
        mergedPeaks.push_back(peaks[0]);
        for (size_t i = 1; i < peaks.size(); i++) {
            if (peaks[i].first - mergedPeaks.back().first <
               1.0) {
                if (peaks[i].second > mergedPeaks.back().
                   second) {
                    mergedPeaks.back() = peaks[i];
                }
            } else {
```

```
                mergedPeaks.push_back(peaks[i]);
            }
        }
    }

    // 检测峰分析 - 若有2个以上峰值执行增强分析
    if (nPeaks >= 2) {
        // 获取峰位置并排序
        Double_t *xpeaks = spec.GetPositionX();
        std::sort(xpeaks, xpeaks + nPeaks);

        // 确定要分析的峰数量 (最多6个)
        int numPeaksToAnalyze = TMath::Min(6, nPeaks);

        // 存储每个峰的参数
        std::vector<Double_t> mu_values(numPeaksToAnalyze);
        std::vector<Double_t> sigma_values(numPeaksToAnalyze
            );
        std::vector<Double_t> chi2_values(numPeaksToAnalyze)
            ;

        // 拟合每个峰
        for (int i = 0; i < numPeaksToAnalyze; i++) {
            TF1 *f = new TF1(Form("f%d",i),
                "gaus", xpeaks[i]-5, xpeaks[i]+5);
            h1->Fit(f, "RQ0+");
            mu_values[i] = f->GetParameter(1);
            sigma_values[i] = f->GetParameter(2);
            chi2_values[i] = f->GetChisquare() / f->GetNDF()
                ;
            delete f;
        }

```

```
84          // 计算相邻峰间距的平均值
85          Double_t avg_delta_pp = 0;
86          for (int i = 1; i < numPeaksToAnalyze; i++) {
87              avg_delta_pp += (xpeaks[i] - xpeaks[i-1]);
88          }
89          avg_delta_pp /= (numPeaksToAnalyze - 1);
90
91          // 计算每个峰的分辨率和平均分辨率
92          std::vector<Double_t> resolution_values(
               numPeaksToAnalyze);
93          Double_t avg_resolution = 0;
94
95          for (int i = 0; i < numPeaksToAnalyze; i++) {
96              // 使用标准定义: R =  /FWHM, 其中FWHM = 2.35*
97              Double_t fwhm = 2.35 * sigma_values[i];
98              resolution_values[i] = mu_values[i] / fwhm;
99              avg_resolution += resolution_values[i];
100         }
101         avg_resolution /= numPeaksToAnalyze;
102
103         // 保留原始sigma_gain计算用于比较
104         Double_t avg_sigma_gain = 0;
105         int valid_sigma_pairs = 0;
106
107         for (int i = 1; i < numPeaksToAnalyze; i++) {
108             Double_t sigma_curr = sigma_values[i];
109             Double_t sigma_prev = sigma_values[i-1];
110
111             if (sigma_curr*sigma_curr > sigma_prev*sigma_
                   prev) {
112                 avg_sigma_gain += TMath::Sqrt
113                   (sigma_curr*sigma_curr - sigma_prev*sigma_
                        prev);
```

```
114                 valid_sigma_pairs++;
115             }
116         }
117 
118         // 避免除以零
119         if (valid_sigma_pairs > 0) {
120             avg_sigma_gain /= valid_sigma_pairs;
121         } else {
122             // 如果没有有效的sigma对, 使用替代方法
123             for (int i = 1; i < numPeaksToAnalyze; i++) {
124                 Double_t sigma_curr = sigma_values[i];
125                 Double_t sigma_prev = sigma_values[i-1];
126                 avg_sigma_gain += TMath::Sqrt(TMath::Abs
127                   (sigma_curr*sigma_curr - sigma_prev*sigma_
                        prev));
128             }
129             avg_sigma_gain /= (numPeaksToAnalyze - 1);
130             printf("警告: 没有有效的sigma对, 使用替代方法计
                    算avg_sigma_gain\n");
131         }
132 
133         Double_t R = avg_delta_pp/avg_sigma_gain;
134         Double_t R_standard = avg_resolution; // 标准定义的
                分辨率
135 
136         // 添加诊断信息
137         printf("\n=== 诊断信息 ===\n");
138         printf("分析了 %d 个峰\n", numPeaksToAnalyze);
139         for (int i = 0; i < numPeaksToAnalyze; i++) {
140             printf("峰 %d: 位置=%.2f, sigma=%.4f, FWHM=%.4f,
                    分辨率R=%.2f,
141               chi²/ndf=%.2f\n",
142                    i, mu_values[i], sigma_values[i], 2.35*
```

```
                        sigma_values[i],
143                   resolution_values[i], chi2_values[i]);
144         }
145
146         // 增强输出
147         printf("\n=== 增强分析结果 (匹配直方图area特性)
               ===\n");
148         printf("原始峰位:");
149         for (int i = 0; i < numPeaksToAnalyze; i++) {
150             printf(" %.2f", xpeaks[i]);
151             if (i < numPeaksToAnalyze-1) printf(",");
152         }
153         printf("\n");
154
155         printf("精修峰位:");
156         for (int i = 0; i < numPeaksToAnalyze; i++) {
157             printf(" %.2f", mu_values[i]);
158             if (i < numPeaksToAnalyze-1) printf(",");
159         }
160         printf("\n");
161
162         printf("平均峰间距Δ_pp: %.2f (理论值~%.2f)\n", avg_
               delta_pp,
163           h1->GetStdDev()*0.8);
164         printf("平均增益波动 _gain: %.2f\n", avg_sigma_gain)
               ;
165         printf("原始计算分辨率R: %.2f\n", R);
166         printf("标准定义分辨率R: %.2f (应在5-15范围内)\n", R
               _standard);
167         printf
               ("==========================================\n
               ");
168     }
```

```
169
170         // 按峰值高度降序排序（避免使用 lambda 表达式）
171         struct SortByY {
172             bool operator()(const std::pair<double, double>& a,
173                 const std::pair<double, double>& b) {
174                 return a.second > b.second;
175             }
176         };
177         std::sort(mergedPeaks.begin(), mergedPeaks.end(),
                SortByY());
178
179         // 取前8个最高峰，按 x 升序输出
180         int nOutput = std::min(8, (int)mergedPeaks.size());
181         std::vector<double> sortedX;
182         for (int i = 0; i < nOutput; i++) {
183             sortedX.push_back(mergedPeaks[i].first);
184         }
185         std::sort(sortedX.begin(), sortedX.end());
186
187         // 输出结果
188         printf("原始检测到 %d 个峰，合并后剩余 %d 个峰，
189           输出最高的 %d 个（按 x 升序）:\n",
190                 (int)peaks.size(), (int)mergedPeaks.size(),
                      nOutput);
191         for (int i = 0; i < nOutput; i++) {
192             printf("峰 %d: x = %.3f\n", i+1, sortedX[i]);
193         }
194     }
```