SummerChallenge2019

# 量子の波動性と粒子性

教員：山崎祐司, 河野能知
TA：釣希夢, 島田拓也, 角源一郎, 谷口浩平

2019 年 8 月 20 日〜28 日

# 目 次

# 1　目的

光には、波動性と粒子性の 2 重性がある。今回の実験では、光の粒子性または波動性のどちらか一方が顕著に現れる現象ではなく、光が両方の性質を同時にもっていると考えざるを得ないということを検証する。

# 2　演習内容

## 2.1　演習１　光子を見る

まずは干渉縞を観測する前に、そのために用いる実験装置の使い方や仕組みを学ぶために簡単に光子の測定を行う。今回の実験では光源として LED, 検出器として MPPC,EASIROC を用いる。それぞれの詳しい説明は後で述べるためここでは省略するが、演習 1 でそれぞれの使い方を理解することを目的とする。

- LED をパルスジェネレータを用いて光らせ、それがきちんと光っていることを目視で確認する。
- MPPC で光子を観測する。そのためには EASIROC,DELAY,DISCRIMINATOR といったモジュールの使い方を理解して、さらに EASIROC の操作方法を理解する。
- データを解析する。ここで簡単な ROOT の使い方を理解する。

## 2.2　演習２　干渉縞を観測する

今回の実験のメインテーマとなるスリットを用いた干渉縞の観測を行う。この演習では、干渉縞を観測できるセットアップとその結果の解析がメインになると思う。

- レーザーポインタを用いてスリットの干渉縞を目視で確認する。実際の測定は暗箱内で行うため、常にこの干渉縞を観測しているとイメージしながら以降の測定を行ってほしい。
- 2 重スリットを用いた干渉実験。まずは干渉縞が MPPC で観測できるようなセットアップにし、稼働ステージで MPPC を移動させながら測定を行う。1 回の移動で動かす距離は、理論式から明線と暗線の間隔を計算し、そこから決めるとよい。
- ROOT を用いて測定データを解析して、干渉縞のグラフを描く。具体的な流れについては後で説明するためここでは省略する。

## 2.3　演習３　１光子の干渉縞を測定する

演習 2 からの変化として光子数を減らして、1 光子数での干渉縞の観測を目指す。解析手法はほとんど変わらず、光量を抑える工夫をすればよい。基本的には LED への印加電圧を下げれば光量は減少するが、それでも足りなければ各自で工夫してみてください。

# 3　原理

## 3.1　光とは

電子のエネルギー状態が、高エネルギー状態から低エネルギー状態へ変化した時、このエネルギーの差分を原子の外に波動エネルギーとして放出する。この波動エネルギーを電磁波・光と呼ぶ。位置 $\boldsymbol{r}$、時間 t における電磁波は以下の式で表される。

- 電場：$\boldsymbol{E}(\boldsymbol{r},t) = \boldsymbol{E_0}\sin(\omega t - \boldsymbol{k}\cdot\boldsymbol{r})$
- 磁場：$\boldsymbol{B}(\boldsymbol{r},t) = \boldsymbol{B_0}\sin(\omega t - \boldsymbol{k}\cdot\boldsymbol{r})$

$\boldsymbol{E_0}, \boldsymbol{B_0}$：定数、　$\boldsymbol{k}$：波数ベクトル、　$\omega$：角振動数とする。

## 3.2　光の干渉

光の波動性を見る 1 つの実験として、干渉実験がある。以下、光の干渉の原理について述べる。

### 3.2.1　線状光源

直線に並んだ振幅、周波数が互いに等しい N 個の光源を考える。各光源は等しい初期位相角をもっていると仮定する。ここで、j 番目の光源が $r_j$ 離れた点で作る電場 $E_j$ は

$$E_j = E_0\sin(\omega t - kr_j)$$

と書ける。今、$E_0 \propto 1/r_j$ なので、定数 $C_0$ を用いて

$$E_j = \frac{C_0}{r_j}\sin(\omega t - kr_j)$$

さらに、この光源を無限に並べた線状光源を考える。ここで、考えている状況は、各光源は非常に弱く、光源の数 N は極めて多く、光源間の間隔が無視できるほど小さい状況である。ここで、D を線状光源全体の長さとすると、線状光源の微小部分 $\delta y_i$ 個の光源を含んでいる。（ここで、光源は M 個の微小部分に分けられているとする。$1 \leq i \leq M$）

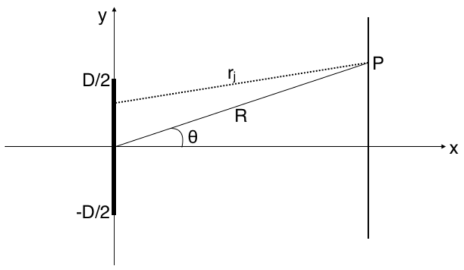



図 1: 線状光源

このとき $\delta y_i$ が P に作る電場は、

$$E_i = \frac{C_0}{r_i}\sin(\omega t - kr_i)\frac{\delta y_i N}{D}$$

となる。ただし、$\delta y_i$ は微小であり、この微小範囲内の各光源から P まで距離は一定であるとする。さらに一定値の $C_L$ を

$$C_L = \frac{1}{D}\lim_{N\to\infty}(C_0 N)$$

と定義できる。M 個の全部分による P での電場は、

$$E = \sum_{i=1}^{M}\frac{C_L}{r_i}\sin(\omega t - kr_i)\delta y_i$$

最後に、$\delta y_i$ は無限小になるはずで、

$$E = C_L\int_{-D/2}^{D/2}\frac{\sin(\omega t - kr)}{r}dy$$

となる。ここで、$r = r(y) = \sqrt{R^2\cos^2\theta + (R\sin\theta - y)^2}$ である。

### 3.2.2 単スリット

まず、単スリットの場合どのように干渉が起こるかを確かめる。振幅、周波数が互いに等しい長さ D の線状光源を考える。線状光源の中心から y だけ離れた光源の微小部分 dy が、中心から xy 平面上の角 $\theta$ の方向に R だけ離れた点 P に作る電場は、

$$dE = \epsilon\frac{\sin(\omega t - kr)}{r}dy$$

$r(y)$ を $y$ でテイラー展開すると、

$$\begin{aligned} r &= r(0) + \frac{\partial r(0)}{\partial y}y + \frac{1}{2!}\frac{\partial^2 r(0)}{\partial y^2}y^2 + \cdots \\ &= R - y\sin\theta + \frac{y^2}{2R}\cos^2\theta \end{aligned}$$

となる。いま、$R \gg y$ なので、第 3 項以降は無視できる。

$$dE = C_0\frac{\sin(\omega t - k(R - y\sin\theta))}{R - y\sin\theta}dy$$

これの分母は $R \gg y$ より $R$ としてよいので、

$$dE = C_0\frac{\sin(\omega t - k(R - y\sin\theta))}{R}dy$$

これは、R が十分大きいとき、$\theta$ の全ての値に対して正しい。

$$\begin{aligned} E &= C_L\int_{-D/2}^{D/2}\frac{\sin(\omega t - k(R - y\sin\theta))}{R}dy \\ &= \frac{C_L}{R(kD/2)\sin\theta}\sin\left(\frac{kD}{2}\sin\theta\right)\sin(\omega t - kR) \\ &= \frac{C_L D}{R\beta}\sin\beta\sin(\omega t - kR) \end{aligned}$$

$\beta = \frac{kD}{2}\sin\theta$ とすると、以上のようになり、線状光源が作る電場が導かれた。このとき、強度は $I(\theta) = \langle E^2 \rangle_T$ で求められるので、

$$\begin{aligned} I(\theta) &= \langle E^2 \rangle_T \\ &= \langle \sin^2(\omega t - kR) \rangle_T \left( \frac{C_L D}{E} \right)^2 \left( \frac{\sin\beta}{\beta} \right)^2 \end{aligned}$$

$\langle E^2 \rangle_T$ は $E^2$ の十分に長い時間発展で、周期 $2\pi/\omega$ とすると、

$$\begin{aligned} \langle \sin^2(\omega t - kR) \rangle_T &= \frac{\omega}{2\pi} \int_0^{2\pi/\omega} \sin^2(\omega t - kR) dt \\ &= \frac{1}{2} \end{aligned}$$

より、

$$I(\theta) = \frac{1}{2} \left( \frac{C_L D}{R} \right)^2 \left( \frac{sin\beta}{\beta} \right)^2$$

$I(0)$ を求めると

$$I(0) = \frac{1}{2} \left( \frac{C_L D}{R} \right)^2$$

なので、

$$I(\theta) = I(\theta) \left( \frac{\sin\beta}{\beta} \right)^2$$

と求まる。光の強度の変化と観測位置の関係は以下のように表される。

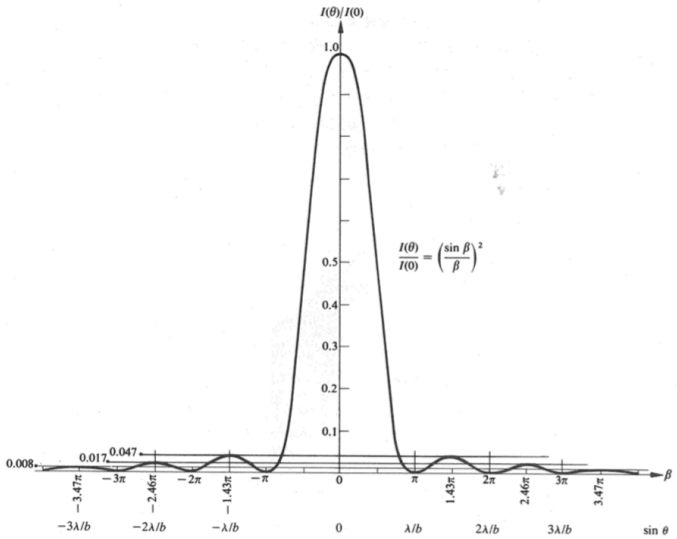



図 2: 単スリットでの光の強度と角度の関係

### 3.2.3　2 重スリット

図のような、幅 $b$、中心間隔 $a$ の 2 本の長いスリットがあるとする。スクリーン上のある点の光波に対する式を得るには、2 つの電場の和になるので、

$$\begin{aligned} E &= \frac{C_L}{R}\int_{-b/2}^{b/2} F(z)dz + \frac{C_L}{R'}\int_{a-b/2}^{a+b/2} F(z)dz \\ &\simeq \frac{C_L}{R}\int_{-b/2}^{b/2} F(z)dz + \frac{C_L}{R}\int_{a-b/2}^{a+b/2} F(z)dz \end{aligned}$$

ここで、

$$\begin{aligned} F(z) &= \sin(\omega t - k(R - z\sin\theta)) \\ \alpha &= \frac{ka}{2}\sin\theta \\ \beta &= \frac{kb}{2}\sin\theta \end{aligned}$$

これより、この式を簡単にすると、

$$E = 2\frac{C_L b}{R}\frac{\sin\beta}{\beta}\cos\alpha\sin(\omega t - kR + \alpha)$$

であり、単スリットの時と同様に強度 $I(\theta) = \langle E^2\rangle_T$ を求めると、$I(\theta) = \frac{1}{2}(\frac{C_L b}{R})^2$ より

$$I(\theta) = 4I_0\left(\frac{\sin\beta}{\beta}\right)^2\cos^2\alpha$$

となる。

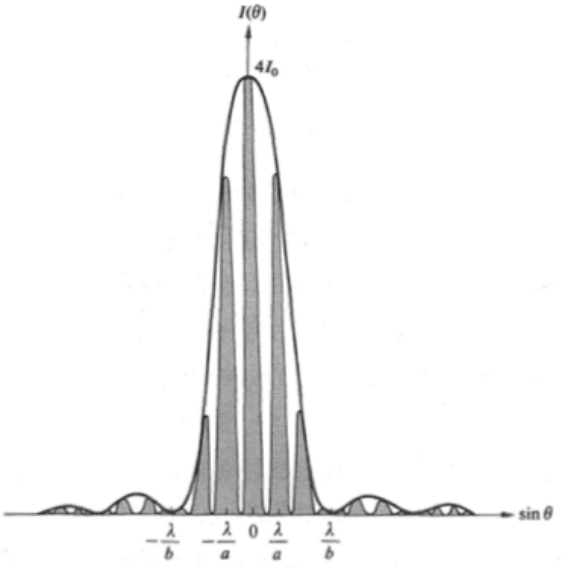



図 3: 2 重スリットでの光の強度と角度の関係

### 3.2.4　参考文献

- 「フーリエ光学 (第 3 版)」森北博巳（森北出版株式会社）2012 年初版

### 3.3 光子を数える

光量が極端に少なくなると、光は光子として離散的になり、その数を数えることができるようになる。「光子数を数える」ということで、光の粒子性を観測することができる。どのような機器で測定するかは、次章にて述べる。

## 4 実験装置

用いる実験装置は、光検出デバイス MPPC、MPPC 読み出しモジュール EASIROC、暗箱、光学機器、スリット、しぼり、LED。光学機器を図のように並べ、光の干渉を起こす。各デバイスの位置や距離によって干渉の見え方が変わるが、これは実際に物を置いていろいろ試してみるとよい。以下では、光検出器とその読み出しについて説明する。

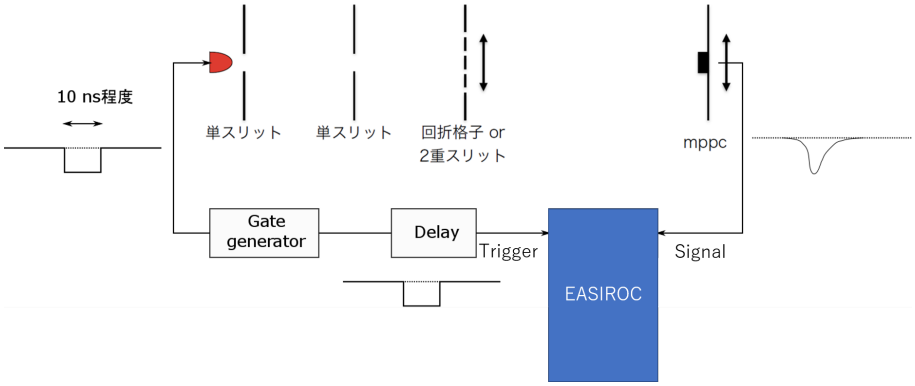



図 4: セットアップ

### 4.1 MPPC

MPPC(Micro Pixel Photon Counter) は Avalamche Photo Diode(APD) が多数配列された光検出器である。APD は半導体でできている。P 型と N 型、異なるドープ型の半導体にある向きで電圧をかけると、キャリアの少ない空乏層ができる。ここに光子が入射すると、光電効果で電子をはじき出す。電子は印加電圧によってエネルギーを持ち、さらに周囲の原子殻内電子をはじき出し、雪崩 (Avalanche) 的に電子を倍増する。印加電圧によって様々な倍増モードがあるが、MPPC 内の APD では、十分な印加電圧をかけることで入射光子数によらない出力電流を得る (ガイガーモード)。このガイガーモード APD が多数配列することにより、我々は光子が入射した APD のチャンネル数だけを数えることで、(pile up はあるものの) 光子数をデジタルに数えることができる。浜松ホトニクスのハンドブックに、詳細な MPPC の挙動が説明されている。[1]

### 4.2 EASIROC

EASIROC モジュールは、書き換え不可能な集積回路 (ASIC) である EASIROC チップと、EASIROC チップ制御用の書き換え可能な集積回路 (FPGA) である Artix7 が搭載された、MPPC の (多チャンネネル) 読み出しモジュールである。今回は 1 チャンネルしか使わないし、FPGA の firmware の書き換えも (おそらく) ないので、ここでは、EASIROC チップの回路の概要と、アナログな出力波形について見ることにす

図 5: MPPC

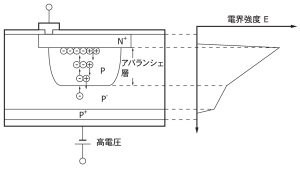



図 6: APD

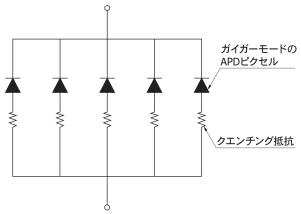



図 7: MPPC は APD を並列に並べたもの

る。

EASIROC チップは、Pre-Amp,Shaper,Discriminator,Capacitor からなる。Pre-Amp,Slow Shaper を通った信号電圧を Capacitor に保存し、Trigger 信号が入力されたときに電圧を hold し、ADC(Analog Digital Converter) でデジタル情報に変換する。各デバイスを通った信号の出力波形は、EASIROC モジュールの

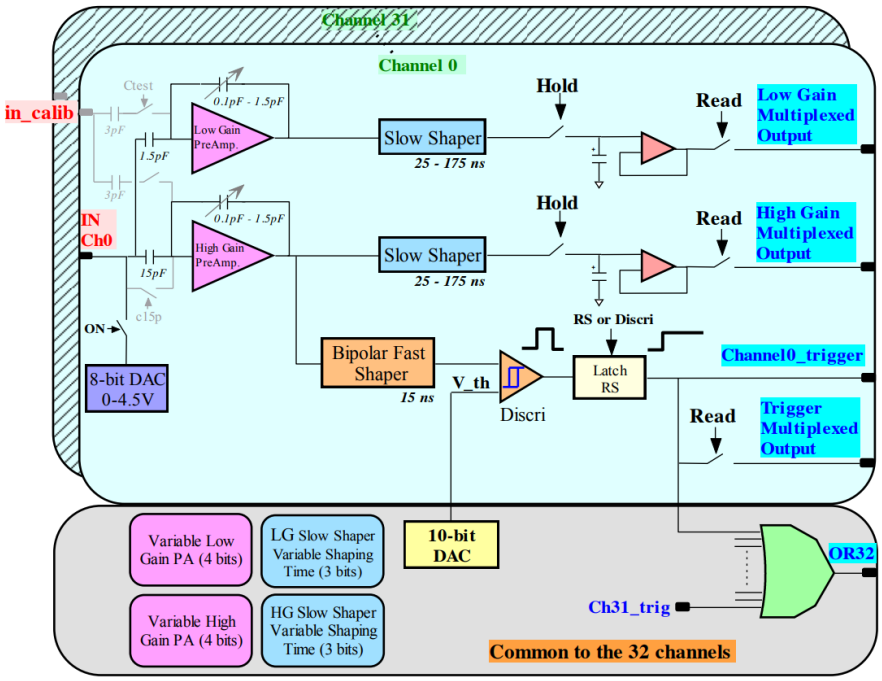



図 8: EASIROC 回路概略

Probe 出力を観察するとわかる。Fast Shaper は Self Trigger 用なので、今回は (おそらく) 使わない。Trigger 信号に合わせて Shaper の波高の一番高いところが読み出されているのがわかる。Trigger 信号と Shaper の立ち上がりがきちんと合うかどうかはケーブルの長さや Delay モジュールによって操作できる。

## 4.3 EASIROC を用いたデータ収集の手順

2 つの実行ファイルを並行して操作する必要があるため、2 つのウィンドウを立ち上げる。これらを今後、ウィンドウ 1, ウィンドウ 2 と区別する。

1. EASIROC モジュールにアクセスする (ウィンドウ 1)

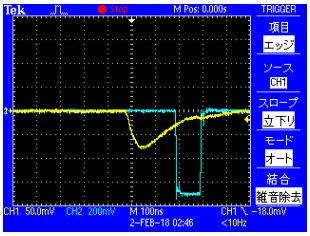


図 9: Pre-Amp

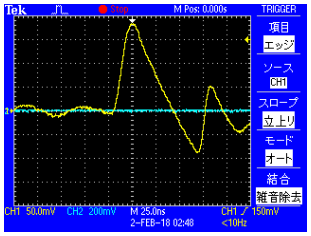


図 10: Fast Shaper

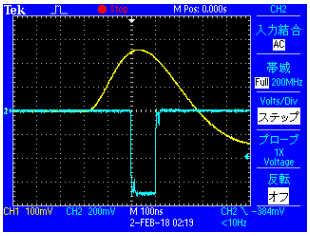


図 11: Slow Shaper

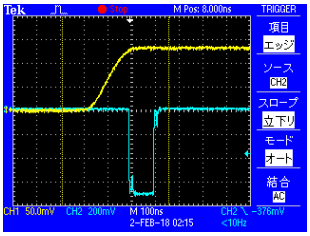


図 12: hold した Slow Shaper

```
\$ cd EASIROC/easiroc
\$ ./easiroc 192.168.10.102 3

\ I'm : 192.168.10.102
\ Please select
\ 1. Transmit SC
\ 2. Tranmit Read SC
\ 3. ASIC Initialize
\ 4. Transmit Probe
\ 5. Connection Close
\ 6. Start DAQ
\ 7. Debug
\ input # === >
```

”Please select” 以降の文字が表示されていれば OK。以降でこれら 7 つのコマンドを使い分け、EASIROC を操作する。主な各コマンドの操作内容は次のようになっている。

- 1.EASIROC 測定後でもオシロスコープで波形を見ることができる
- 2. どのチャンネルが出力されるか確認 (今回は ch.12 から出力される信号のみを見るため特に出番はない)
- 3. 接続を切る
- 4. 測定を開始する

2. udp102 を立ち上げる (ウィンドウ 2)。

```
\$ cd EASIROC/easiroc/easiroc_UDPcontrol
\$ ./udp102
```

ここで用いるコマンド番号は以下の様になっている。

- 1 HighVoltage の設定この時打った電圧が全チャンネルに反映される.）
- 2 HiVol と電流のステータス (電流は正常な値で数 uA 程度である。10 倍以上の電流が流れるようであればすぐに HiVol を 0 にすること)
- 5 exit

3. ウィンドウ 2 にてコマンド”2”を入力し印加電圧を設定する。印加電圧は 53.5 から 54V 程度が好ましく、それ以上に設定すると MPPC に負担がかかるため極力上げないこと。
4. ウィンドウ 1 でコマンド”1”→”2”を入力し、オシロスコープにて波形を確認する。
5. ウィンドウ 1 でコマンド”6”を入力し、測定を開始する。

コマンド”6”を入力した後、保存したいファイルの名前と測定するイベント数を入力。

```
Input output data file name : ファイル名.dat
Input total # of events ===== > 欲しいイベントの数
```

# 5　データ解析

## 5.1　MPPC で取得されるデータの解析

MPPC で取得されるデータは EASIROC によって tree 形式で保存される。今回はこのデータを ROOT を用いて解析を行う。ここから解析に必要な ROOT の知識を簡単に説明するが、すべてを説明することはできないためわからない部分は各自調べるか TA に質問したりするようにしてください。

## 5.2　データ解析用 PC にログインする

ここでは、自分の PC から Linux にリモートでログインする方法を説明します。

### 5.2.1　Windows Subsystem for Linux (WSL) を用いて Linux を使う

Windows10 以降で使用可能になった Windows Subsystem for Linux(WSL) は、 Windows10 から Linux の実行環境を実現するサブシステムです。簡単にいうと、 Windows のアプリケーションとして Linux を実行できます。

#### WSL の有効化

まず、Windows 側で Linux Subsystem を有効化します。

1. スタートボタンを右クリックし、「アプリと機能」をクリックします。
2. 右上の「プログラムと機能」をクリックします。
3. 左サイドバーの「Windows の機能の有効化または無効化」をクリックします。
4. 「Windows Subsystem for Linux」を探して、チェックを入れます。
5. 「OK」を押すと、インストールが始まります。インストール後、再起動します。

#### Ubuntu のインストール

1. Microsoft Store で「WSL」と検索します。もしくは「Ubuntu」。
2. アプリの中から「Ubuntu」を探し、インストールします。
3. インストール作業が終わったら、スタートに追加される「Ubuntu」をを起動します。
4. ユーザー名とパスワード を要求されるため、プロンプトに従って入力していきます。

Listing 1: 表示されるスクリプト例

```
1 Installing, this may take a few minutes...
2 Please create a default UNIX user account. The username does not need to match your
      Windows username.
3 For more information visit: https://aka.ms/wslusers
4 Enter new UNIX username: [user name]
5 Enter new UNIX password: [password]
```

```
Retype new UNIX password: [password]
passwd: password updated succsessfully
Installation successful!
To run a command as administrator (user ‘‘root’’), use ‘‘sudo’’.
See ‘‘man sudo_root’’ for details.
```

5. パッケージのアップデート
   パッケージのアップデートは日々行ったほうが良いです。

```
sudo apt update
sudo apt upgrade
```

## X window のインストール

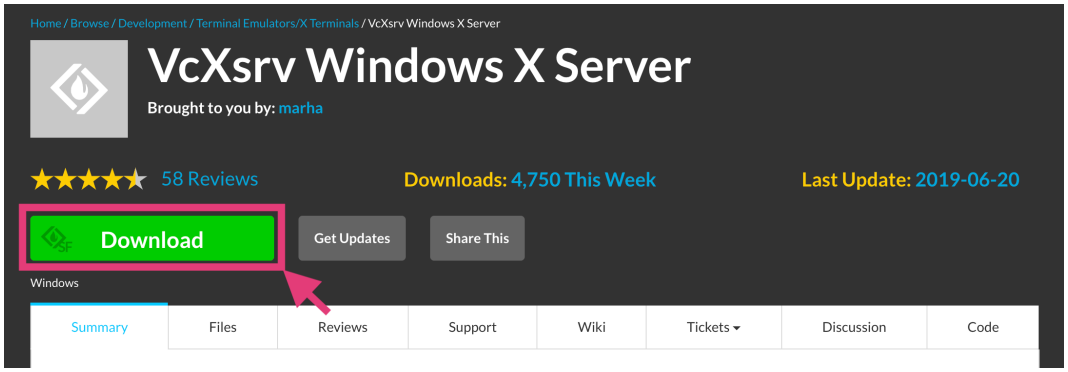



図 13: VcXsrv の HP。Download をクリックします。

Windows OS で X window (画像表示用・root 用) を使用するため、「VcXsrv」をインストールします。

1. `https://sourceforge.net/projects/vcxsrv/` にアクセスし、「Download」をクリックします。
2. インストーラを起動し、オプションやインストール先などを指定してインストールします。
   - Installation Option：全てにチェックする。 (Full)
   - Installation Folder：C:￥Program File￥VcXsrv
3. インストール作業が終わったら、スタートに追加される「VcXsrv」をを起動します。
4. 起動するとウィザード画面が表示される。それぞれ選択し、設定を完了させます。
   - Display settings：好きなもので
   - Client startup：次へ (start no client)
   - Extra settings：次へ (Clipboard - Primary Selection, Native opengl
   - Save configuration：「Save configuration」をクリック
5. システムトレイに VcXsrv のアイコンが表示されていれば OK です。

macOS の場合は、「XQuartz」をインストールします。

1. `https://www.xquartz.org/` にアクセスし、 dmg ファイルをダウンロードします。

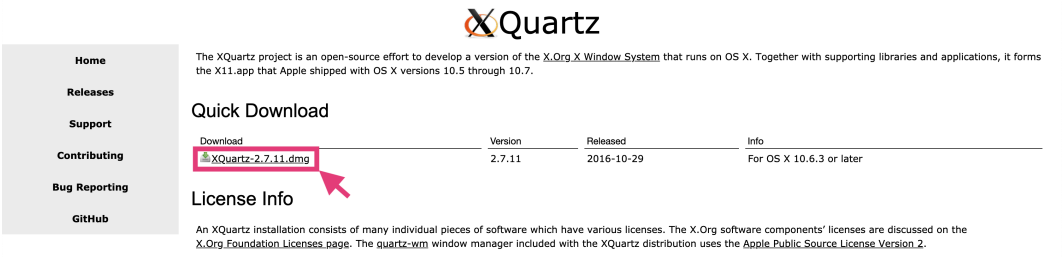



図 14: XQuartz の HP。dmg ファイルをクリックします。

2. dmg ファイルを展開し、XQuartz.pkg をクリックして、起動します。
3. インストーラに従って、XQuartz をインストールします。
4. インストールが完了すると、/アプリケーション/ユーティリティ/ に XQuartz.app が作成されていますので、正常に起動するか確認します。

### 5.2.2 SSH 接続でリモートログインする

```
1 \$ ssh -Y [user name]@[host.IP.Address]
```

これで、Linux サーバーにログインできます。

## 5.3 ROOT のインストール方法

ROOT とは CERN が開発したデータ解析のためのフレームワークです。特によく使われる目的としてはヒストグラム・グラフを描く、任意の関数でフィットする、大量のデータを処理するといったものがあります。物理学実験では大量のデータを扱うため、例えばエクセルのようなものではそれらのデータをプロットしたり複雑な解析を行うことは難しくなります。そのため、プログラミングによる解析を行えるようになる必要となります。言語としては主に C++で動作し、python で書くこともできます。

### 5.3.1 ROOT ビルドの準備

## Windows 版

パッケージの準備

ROOT を Ubuntu 上で動かすにはあらかじめいくつかのパッケージをインストールしておく必要があります。次の URL にある Ubuntu のところのパッケージをインストールしてください。（すでにされているものもあると思います。）
`https://root.cern.ch/build-prerequisites`
最低限下記のものはインストールしてください。

```
sudo apt-get install git dpkg-dev cmake g++ gcc binutils libx11-dev libxpm-dev libxft-dev
     libxext-dev
```

ROOT のソースファイルをダウンロード

Ubuntu 用にコンパイルしたファイル を CERN が用意してくれています。
ROOT ソースファイルのダウンロードは以下のページから。
https://root.cern.ch/downloading-root
「Latest ROOT Releases Pro」をクリックすると、ダウンロード可能なファイルの一覧が表示されます。
ダウンロードするディレクトリを作成します。

```
mkdir local
```

次のコマンドで最新のバージョンの root がダウンロードされます。

```
cd local
git clone http://github.com/root-project/root.git
ls
-root
```

ROOT を build するディレクトリを作成します。

```
mkdir root_build
ls
-root  root_build
cd root_build
```

cmake を行い、ROOT の構築に必要なファイルを生成する。

```
cmake ../root
```

cmake の実行が終了し、問題なければ build を開始する。

```
cmake --build . [-- <options to the native tool>]
cmake --build . -- -jN
```

オプションは下記のサイトで確認。2 つ目のコマンドですると、build をスピードアップできる。N は利用可能なコア数により 2 や 4 などの数字を打つ。
参考 URL(https://root.cern.ch/building-root)
下記のコマンドを打って ROOT が起動すれば build に成功しています。

```
source bin/thisroot.sh
root
```

パスを通す

Ubuntu の起動時に毎回 thisroot.sh が実行されるようにします。

```
vim ~/.bashrc
```

のコマンドでエディタを開いたあと以下を追記

```
#ROOT
. ~/local/root_build/bin/thisroot.sh
```

Ubuntu の起動時に.bashrc が読み込まれるようにします。

```
vim ~/.bash_profile
```

のコマンドでエディタを開いたあと以下を追記

```
if [ -f ~/.bashrc ]; then
        . ~/bashrc
fi
```

一度 Ubuntu を閉じてもう一度起動し、下のコマンドを入力して ROOT が起動すれば成功です。

```
root
```

## Mac 版

### Homebrew のインストール

Homebrew は macOS 用のパッケージマネージャーです。Homebrew で ROOT ビルドに必要な CMake をインストールします。

Homebrew を使うには、Xcode の Command Line Tools が必要になります。ターミナルで次のコマンドを実行し、Xcode をインストールします。

```
\$ xcode-select --install
```

インストーラーの指示にしたがって、インストールを完了させます。正常にインストールされていることを確認するため、

```
\$ xcode-select -v
```

を実行します。バージョンが表示されていればインストール完了です。

次に、Homebrew をインストールします。`https://brew.sh/index_ja.html` 上記の URL にあるコマンドをターミナルに入力します。


図 15: Homebrew の HP 画面。画面下のコマンドをコピーしてターミナル上で実行する。

Listing 2: Homebrew インストール時のスクリプト

```
...
==> Installation successful!
==> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
https://docs.brew.sh/Analytics
==> Homebrew is run entirely by unpaid volunteers. Please consider donating:
https://github.com/Homebrew/brew#donations
==> Next steps:
- Run `brew help` to get started
```

```
- Further documentation:
https://docs.brew.sh
```

が表示されれば、インストール完了です。

#### CMake のインストール

Homebrew を用いて、CMake をインストールします。

```
brew install cmake
```

を実行すると、CMake がインストールできます。

### 5.3.2 ROOT ビルド方法

```
\$ cd ~
\$ curl -O https://root.cern.ch/download/root_v6.16.00.source.tar.gz
\$ cd /usr/local
\$ sudo tar zxvf ~/root_v6.16.00.source.tar.gz

\$ cd /usr/local/root-6.16.00
\$ sudo mkdir obj
\$ cd obj
\$ sudo cmake ..

...
-- Configuring done
-- Generating done
-- Build files have been written to: /usr/local/root-6.16.00/obj
```

ここまで出れば CMake 完了です。

```
\$ sudo make -j 8

...
[100%] Generating tutorials/hsimple.root
Processing hsimple.C...
hsimple : Real Time = 0.06 seconds Cpu Time = 0.05 seconds
(TFile *) 0x7f9f7cda9760
[100%] Built target hsimple
```

ここまで出ればビルド完了です[1]。

### 5.3.3 ROOT の環境設定

bash

## 5.4 ROOT を用いたデータ解析

### 5.4.1 ROOT の起動方法

ROOT の起動方法はコマンドラインに以下のコマンドを入力するだけです。

```
root
```

起動の度に図 16 が表示されるため、オプションをつけます。

```
root -l
```

ROOT を終了したいときには、.q と入力すれば ROOT を終了させることができる。

[1] https://root.cern.ch/building-root を参考にしてください。

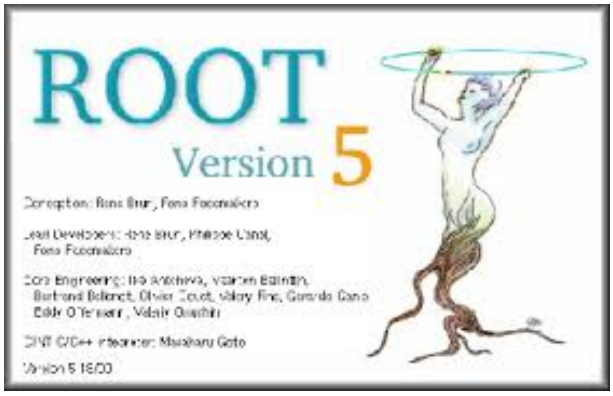



図 16: ROOT のロゴ

表 1: ROOT 起動時のオプションの例

| オプション | 意味 |
|---|---|
| -l: | 起動時にロゴを表示しない |
| -b: | ヒストグラムやグラフのグラフィックを描画しない |
| -q: | スクリプトファイルを実行後自動で ROOT が終了する |

### 5.4.2 ヒストグラムの描き方

root を起動する際に、引数に root ファイルを指定することで root の起動と同時に ROOT ファイルを開くことができる。例えば、今回測定結果を example.root という名前で保存しているとした場合、root example.root とコマンドラインから入力することで測定データに簡単にアクセスすることができる。17

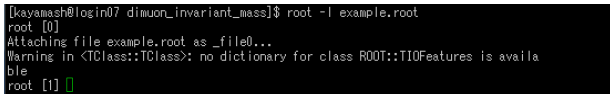



図 17: root ファイルを開く

example.root 内にさらに tree という名前で tree を作っている場合、tree->” コマンド” という書き方をすることができる。それにより、簡単に測定データを確認したりヒストグラムを描くことができる。コマンドとしては例えば、tree->Print() や tree->Scan() があり、Print はデータの数や種類、変数の方を確認でき、Scan は 1 つ 1 つのデータを実際に確認することができる。

今回 tree の中身を Print や Scan で確認すると、いくつかの Branch があると思う。[2]そのうち今回 adc という Branch のヒストグラムを描きたい場合、tree->Draw($''adc''$) と打つだけでヒストグラムを描くことができる。[3]

ヒストグラムを書くときに、ただそのまま書くだけでなく様々な条件をかけることもできる。例えば、tree->Draw(”adc*2”) とすれば、adc の値を 2 倍したヒストグラムを描くことができ、tree->Draw(”adc”,”adc¿900”) とすれば adc が 900 より大きいイベントのみ描くことができます。この条件には、ほかの Branch を用いることもでき tree->Draw(”adc”,”adc_1 > 1000 &adc_1 < 3000”) とすれば 1000<adc_1<3000 のみのヒストグラムを描くことができる。ただし、ある程度複雑な条件になってくるとこの書き方をするのは難しくなっ

[2]Branch とは言葉通り枝のようなもので、1 つの Entry に対して様々なデータを詰め込むことができる。

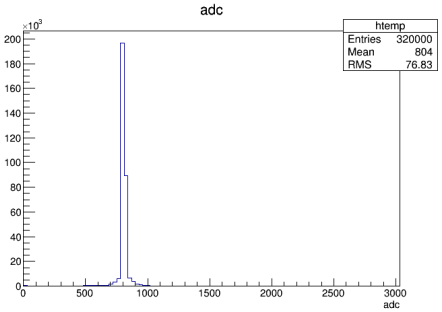
[3]今回、EASIROC では 32ch のデータが配列として adc の Branch に入っており、そのうち MPPC がつながっているのは 30ch のみなので、adc[30] のみを Draw することで測定データのヒストグラムを描ける

```
root [2] tree->Print()
******************************************************************************
*Tree    :tree      : tree                                                   *
*Entries :    10000 : Total =         2568389 bytes  File  Size =     769217 *
*        :          : Tree compression factor =   3.34                       *
******************************************************************************
*Br    0 :adc       : adc[32]/I                                              *
*Entries :    10000 : Total  Size=    1283984 bytes  File Size  =     428790 *
*Baskets :       41 : Basket Size=      32000 bytes  Compression=   2.99     *
*.............................................................................*
*Br    1 :adc_l     : adc_l[32]/I                                            *
*Entries :    10000 : Total  Size=    1284074 bytes  File Size  =     339316 *
*Baskets :       41 : Basket Size=      32000 bytes  Compression=   3.78     *
*.............................................................................*
root [3]
```

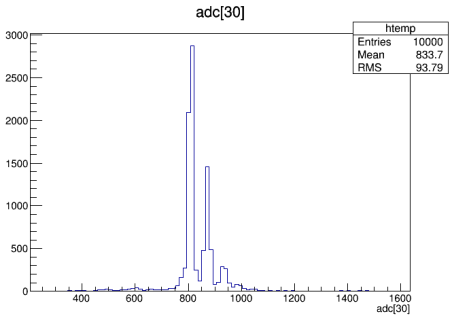
(a)tree->Print() の実行例

```
root [1] tree->Scan()
************************************************
*    Row   * Instance *       adc *     adc_l *
************************************************
*        0 *        0 *       735 *       815 *
*        0 *        1 *       718 *       814 *
*        0 *        2 *       790 *       815 *
*        0 *        3 *       767 *       809 *
*        0 *        4 *       772 *       808 *
*        0 *        5 *       793 *       812 *
*        0 *        6 *       790 *       810 *
*        0 *        7 *       773 *       815 *
*        0 *        8 *       766 *       815 *
*        0 *        9 *       724 *       813 *
*        0 *       10 *       766 *       803 *
*        0 *       11 *       748 *       813 *
*        0 *       12 *       757 *       813 *
*        0 *       13 *       793 *       812 *
*        0 *       14 *       787 *       804 *
*        0 *       15 *       777 *       812 *
*        0 *       16 *       819 *       816 *
```

(b)tree->Scan() の実行例




(a)tree->Draw("adc") の実行例




(b)tree->Draw("adc[30]") の実行例

てくるので、後述するマクロを書いてヒストグラムを描くことをお勧めする。

最後に tree をそのまま draw する以外のヒストグラムの描き方として、Fill という方法がある。簡単な例として、自分でガウス分布に従う乱数を発生させてそれをヒストグラムに詰めるやり方を紹介する。

```
1      TH1D *h1 = new TH1D(''h1'',''h1'',50,-5,5);
2      for(Int_t i = 0;i < 10000;i++){
3       Double_t x = gRandom->Gaus();
4       h1->Fill(x);
5      }
6      h1->Draw();
```

3 行目で gRandom というものを用いて Gauss 分布に従う乱数を発生させ、それを 4 行目で Fill というコマンドで h1 のヒストグラムに詰めている。それを for 文で回して 10000 回詰めている。このヒストグラムを Draw することで Gauss 分布に従うヒストグラムを描くことができる。今回は例として乱数を Fill したが、乱数の代わりに詰めたい値を詰めることで好きなヒストグラムを描くことができる。

### 5.4.3 マクロの書き方

簡単なマクロはコマンドラインに書いてヒストグラムを描くときと同じものを描くだけでよい。ただし、root ファイルを読み込む際には必要な手順があるので例としてそれを示す。

```
1  void macro(){
2    TFile *file = new TFile("example.root");
3    TH1D *h1 = (TH1D*)file->Get("adc");
4    h1->Draw();
5  }
```

というファイルを作り、ターミナル上で root macro.cxx や root を起動した後、.x macro.cxx とコマンドすることでこのマクロを実行できる。データの解析には基本的には ROOT 固有の知識はそれほど必要ではなく、c++の知識があれば充分であるため深くは説明しない。

もう一つ必要な知識として、詳しい解析を行う際には Branch のデータをヒストグラムとしてではなく値として取り出す必要がある。そのためこれも簡単にではあるが、値としての取り出し方を説明する。

```
1  void macro(){
2      TFile *file = new TFile(inputfilename, "read");
3      TTree *tree = (TTree*)file->Get("tree");
4
5      Double_t adc[32]
6      tree->SetBranchAddress("adc", &adc);
7
8      const Int_t N = tin->GetEntries();
9      for (Int_t ientry = 0; ientry < N; ientry++) {
10      tree->GetEntry(ientry);
11      cout<<ientry<<''   ''<<adc[30]<<endl;
12    }
13 }
```

tree->GetEntry(ientry) とすることで、ientry 番目のイベントの adc を読み込んでおり、それを for 文で全イベント回すことで全データを読み込んでいる。今回のマクロでは cout で出力しているだけだが、その値を vector や配列に詰めることで解析に用いることができる。

コマンドラインによる解析の仕方も一応説明しているが、測定直後にデータを簡単に確認したいといった場合を除いてはマクロを用いればよい。また、データを確認する場合でも TBrowser[4]が便利であるため、あまり使用頻度は高くないと思われる。

[4]ROOT を起動した後、TBrowser tb と入力することで root ファイルの中身をヒストグラムとして簡単に見ることができる

### 5.4.4 ヒストグラムのフィット

ROOT にはあらかじめガウス分布やポアソン分布などの様々な関数が用意されており、それらをヒストグラムのフィットを行う際に利用できる。例えばガウス分布でフィットする際には、htemp->Fit(”gaus”) とすればよい。[5]コマンドラインで行った場合にはコマンドラインに fit の結果が出力される。

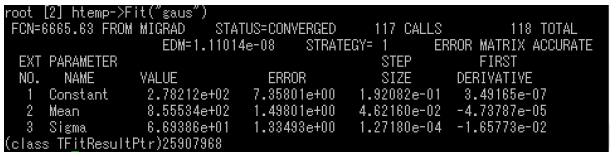



図 18: フィットの結果

ガウス分布の場合は、$(gaus) = (Constant)\times\exp(-\frac{(x-(Mean))^2}{2(Sigma)^2})$ で定義されている。それぞれの出力されるパラメータは関数によって異なるため、ほかの関数を用いる際は一度確認しておくとよい。また、htemp->Fit(”gaus”,””,””,min,max) とすることでフィットする範囲を制限することができる。

あらかじめ定義されている関数以外の関数を用いたい場合は自分で定義することができる。例えば直線でフィットしたい場合には、

```
TF1 *f1 = new TF1(''f1'',''[0] + [1] * x'',min,max);
```

とすることでまず直線の関数を定義することができる。この関数の名前は f1 になっているので、あとは先ほどガウス分布でフィットしたときと同様に、

```
htemp->Fit(''f1'','''',''',min,max);
```

とすればよい。

フィットする際にある程度のパラメータを指定したい場合がある。その際には f1-¿SetParameter(0,5) とすれば 0 番目のパラメータの初期値を 5 に設定でき、f1-¿SetParameters(5,10) とすればまとめて複数のパラメータの初期値を設定できる。

マクロで行う場合には基本的には同じだが、フィットした結果をマクロ内で用いたい場合は値として取り出す必要がある。

```
Double_t p0 = f1->GetParameter(0);
```

とすると、p0 にフィットした後の 0 番目のパラメータの値を代入することができる。

### 5.4.5 グラフの描き方

基本的な使い方として、テキストファイルからグラフを描くやり方と配列や vector[6]を用いる方法がある。まずはテキストファイルでのグラフの書き方から説明する。

例として data.dat というファイルに

```
1.00 3936
0.50 3007
0.10 2249
-0.10 1836
-0.50 1097
-1.00 146
```

と書かれている場合、簡単にグラフを描くことができ

[5]htemp の部分はその時のヒストグラムの名前に応じて適宜変える
[6]vector が何かわからない場合は c++の勉強が先に必要なため、c++ vector 等で検索して使い方を勉強したほうがよいでしょう

```
TGraph *tg1 = new TGraph(''data.dat'');
tg1->Draw(''AP'');
```

とするだけでグラフを描くことができる。ただし、この方法を用いるためには結果をまず dat ファイルとして出力する必要があるため、あまり使うことはないだろう。

配列や vector を用いる場合、まずは配列や vector にデータを詰める必要があり、その部分は省略する。すでに値が詰まっていると仮定すると、配列の場合は

```
TGraph *tg1 = new TGraph(6,x,y);
tg1->Draw(''AP'');
```

とすると、6 個のデータを含むグラフを描くことができる。しかし、測定データによってグラフが何点あるかは異なる場合もあり、vector を用いたほうが様々な場合に柔軟に用いることができ便利である。

vector を用いた場合は

```
TGraph *tg1 = new TGraph(x.size(),&(x.at(0)),&(y.at(0)));
tg1->Draw(''AP'');
```

c++の知識がある人は、これを見てやってることは同じだとわかると思う。

また、グラフにエラーバーをつけたい場合も多いと思う。その場合は TGraph ではなく TGraphErrors を用いればよい。使い方はほとんど同じで、

```
TGraph *tg1 = new TGraph(6,x,x_error,y,y_error);
tg1->Draw(''AP'');
```

の順番に引数がなっており、その通りに TGraph の時と同様に配列や vector で与えればよい。

最後にグラフのフィットについてであるが、ヒストグラムの場合と全く同じであり例えば直線でフィットしたい場合には

```
TF1 *f1 = new TF1(''f1'',''[0] + [1] * x'',min,max);
tg1->Fit(''f1'',''''',''''',min,max);
```

とすればグラフを直線でフィットすることができる。

### 5.4.6 結果の保存

コマンドラインで結果のヒストグラムやグラフを保存する場合には、左上の File から Save As を選ぶと名前を付けて保存することができる。マクロで実行している場合には、ヒストグラムやグラフを描く前に

```
TCanvas *c1 = new TCanvas(''c1'',''c1'',1600,900);
```

としてキャンバスを作っておき、ヒストグラムやグラフを Draw した後に

```
c1->SaveAs(''save name'');
```

とすることで保存することができる。保存する際にきれいにヒストグラムやグラフを整えた後、保存したい人も多いと思う。root はヒストグラムやグラフそれぞれに様々な修飾等が存在し、それをここで説明しているときりがないためまとめ 6 にそれらが書かれているリンクを載せておく。

また、ヒストグラムを保存する際に画像としてでなく root ファイルにヒストグラムのまま保存することができる。実際に触ってみないとそのメリットは感じにくいかもしれないが、root ファイルの状態で保存することで、保存した後にタイトルをつけたり bin 幅を変えるといった加工が簡単に行える。そのため、最終的にスライド等に用いる時以外は root ファイルで保存しておいたほうが便利かもしれない。具体的なやり方が書かれているサイトのリンクをここに載せておく。[7]なお、ヒストグラム等を保存する際は測定データの root ファイルとは別に保存用の root ファイルを作るようにしてください。測定データを上書きしてしまうリスクを避けるために、測定データは読み込み専用で開く習慣をつけるためである。

[7] https://www-he.scphys.kyoto-u.ac.jp/member/n.kamo/wiki/doku.php?id=study:software:root:io の Write ・オブジェクトをファイルに書き込む、root ファイルを保存するを参考にしてもらえばよい。

## 5.5 Poisson 統計

Poisson 分布とはめったに起こらない事象を大量に測定した場合にこの分布に従うことが多い。式で表すと、

$$P(n) = \frac{\lambda^n e^{-\lambda}}{n!} \tag{1}$$

で、これは平均 λ 起こる事象が n 回起こるときの確率を表す。今回の実験の場合はそれぞれの測定での光子数の分布は Poisson 分布に従うと思われる。

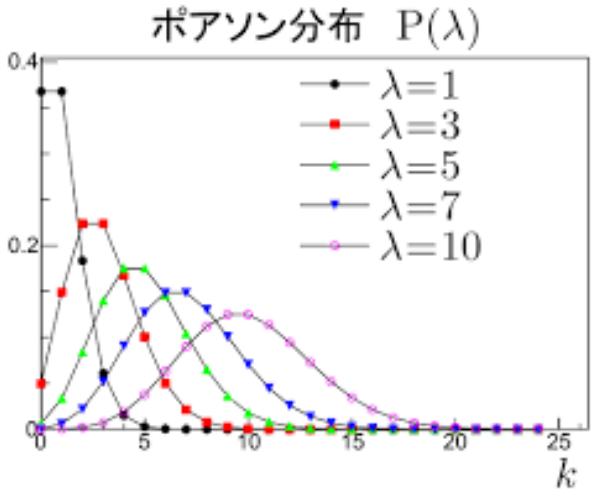



図 19: poisson 分布の図

λ が大きい場合は Poison 分布は Gauss 分布 (正規分布) に従う。今回の実験の場合では、それぞれの光子数に対応するピークは Gauss 分布に従うと思われる。

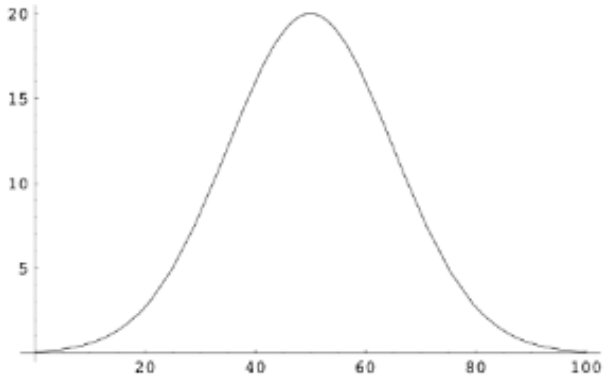



図 20: gauss 分布の図

## 5.6 干渉縞の再現

今回の測定データは最初 adc の値として保存されており、この値自体に物理的な意味はない。そのため、まずはこの値から光子数に変換する必要がある。やり方としてはそれぞれのピークが光子数に対応してい

るため、それぞれをガウス分布でフィットして中心値を求める。その adc 値と光子数の関係を直線近似することで横軸を adc から光子数に変換することができる。

そのあと、変換した後のヒストグラムから MPPC の位置ごとの平均光子数を求めてプロットすることで干渉縞を再現することができる。平均光子数については様々な求め方があるが、簡単なものについては光子数で重みづけした平均、もう少しちゃんと求める場合には上で述べたように光子数の分布は Poisson 分布に従うため、全体を Poisson 分布でフィットすることで求めるほうがよい。

また、演習 3 で光子数を減らした場合はフィットで平均光子数を求めることは難しい。その場合は平均光子数ではなく 1 光子以上のイベント数で考えるとよい。平均光子数そのものではないが、光子数が少ない場合には平均光子数に比例して 1 光子以上のイベント数も増加するため、それでも干渉縞を観測することができる。[8]

# 6 まとめ

今回説明できたのはごく基本的な部分のみであり、今回の実験の解析についてもこれ以上の知識が必要になることがあると思う。c++の知識でわからないことがあれば

`http://www7b.biglobe.ne.jp/~robe/cpphtml/` の第 1 部を見ればある程度の情報はのっていると思われる。もし、これにのっていない知識が必要になれば別途調べるか質問してください。

ROOT についての情報は基本的なものは

`http://www-ppl.s.chiba-u.jp/~hiroshi/ref/ROOT_text.pdf`

`https://www.quark.kj.yamagata-u.ac.jp/~miyachi/ROOT/root.pdf`

、途中で話した修飾については

`http://atlas.kek.jp/comp/ROOT-commands.html`

を見ればある程度一覧になっている。もし、そこにのっていない知識が必要になったときは別途調べるか、

`https://root.cern.ch/guides/users-guide`

の User's Guide を見ればわかると思うが、User's Guide は英語の上膨大であるため、適宜調べるのがよいと思う。

また、あまり全てを書きすぎると参加している皆さんが考えることがなくなってしまうため、あえて説明をしていない部分もある。なので、この資料を読んでわからないことがあれば遠慮せずにどんどん聞いてください。

# 参考文献

[1] https://www.hamamatsu.com/resources/pdf/ssd/03_handbook.pdf

[8] 物理的な説明をすると、1 光子の干渉と仮定すると明線の部分では光子の存在確率が大きいため、光子を観測するイベント数が増加する。光子数が増加すると光子数 0 となるイベントは暗線であっても減少するため、この考え方で解析することは難しくなる。