Software chapter
COMPASS thesis

Armando Brandonisio

August 10, 2016

Contents

Chapter 1

Software development

SiPMs spectrum is theoretically composed by gaussian peaks that follow Poisson distribution as discussed in Cap (..).
In this chapter I show basics for a simple Monte Carlo model and its application for SiPM spectrum simulation. Furthermore I created an analysis software that simplifies and automatizes physical results extracted by SiPMs spectra. I wrote all scripts and code using Python language.

1.1 ~~Monte Carlo simulation~~

~~Monte Carlo method solves a problem by generating suitable random numbers and observing that fraction of the numbers obeying some property or properties. The method is useful for obtaining numerical solutions to problems which are too complicated to solve analytically. In physics-related problems, Monte Carlo methods are quite useful for simulating systems with many coupled degrees of freedom, such as fluids, disordered materials, strongly coupled solids, and cellular structures.~~
In my work I used Monte Carlo method to simulate a SiPM output spectra in the ideal case: no *cross-talk* and no *afterpulses*. The expected spectrum is a gaussians sequence which centroids correspond to each photo-electron value. This result would be a convergence test for fitting processes and physical properties interpretation.
First I simulated a uniform distribution and than a gaussian spectrum to emulate a detected X-Ray photo-peak.
I performed all analysis data and simulation using `Python v.3.5.2` and the following libraries:

- `Numpy v.1.10.4`
- `Numba v.0.23.1`
- `LmFit v.0.93`
- `Pandas v.0.18.1`
- `Scipy v.0.17.0`

- `Matplotlib v.1.5.1`
- `accelerate_cudalib v.2.0`
- `Stoner v.0.5`

1.1.1 Discrete uniform distribution

The discrete uniform distribution has *pdf*

$$f(x;a,b) = \frac{1}{b-a+1}, \quad x \in \{a,...,b\} \tag{1.1}$$

where $a, b \in \mathbb{Z}, b \geq a$ are parameters. The discrete uniform distribution is used as a model for choosing a random element from $\{a,...,b\}$ such that each element is equally likely to be drawn.
There is a easy way to compute a uniform distribution using `Python` language: `Numpy` random libraries. A dedicated routine to compute uniform distributions is `numpy.random.uniform`[1].
Generate random numbers could be a hard work for CPU computers, than I setted *bins* and *events* such that total time computation was $\sim 2s$[2]. I chose 500 bins with 10^5 events. I tested the same program with `accelerate_cudalib` library too that compute code with Nvidia GPU cores (CUDA). The same result was achieved in $\sim 0.1s$[3]. This could be a start point for more realistic simulations.
I interpreted bins as collected charge in a range of 0-40 e^-, and rate as counts per second (normed to 0.5). The result in Fig 1.1.

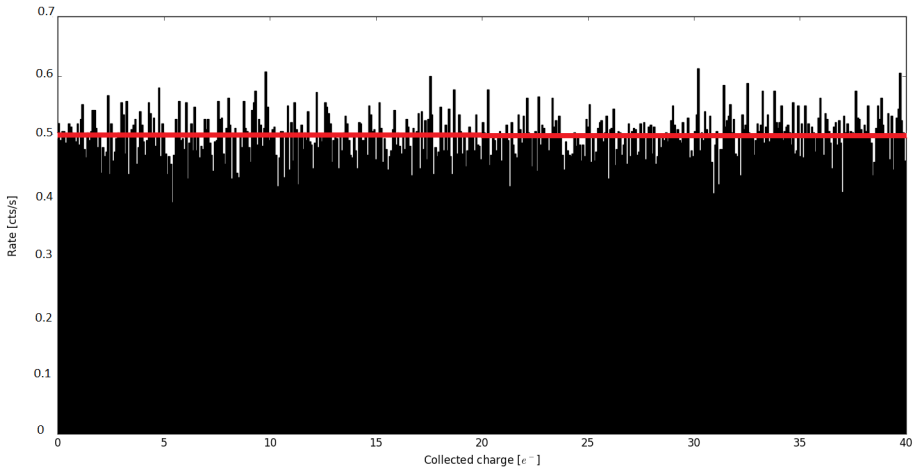

Figure 1.1: Discrete uniform distribution

[1] np.random.uniform:http://docs.scipy.org/doc/numpy/reference/generated/numpy.random.uniform.html
[2] Tested with Intel Core i7-5500U (Broadwell) 2.4 GHz up to 3.0 GHz
[3] Tested with Nvidia 920M and CUDA toolkit 7.5

Simulated spectra have a density bin of $\sim$ 12.5 $bins/e^{-}$. The real spectrum acquired with LCT5/1 (+3V), Ortec474 (gain:4x2, int: 50μs) and MCA dynamic range of 0-5V divided in 4096 channels, has a density bin of $\sim$ 40 $bins/e^{-}$, where bins are MCA channels.
To generate gaussian spectra I used `numpy.random.normal` [4] routine. As discussed in Chapter (..) SiPM spectra follow Poissonian distribution, and then gaussian curves for each photo-electron value (N-PE) depends on $\sigma_i\sqrt{N}$: An example of gaussians sequence is reported in Fig 1.2. I used `numpy.concatenate`[5] routine to sum events of different distributions on the same bin, in order to get the correct analytical convolution.

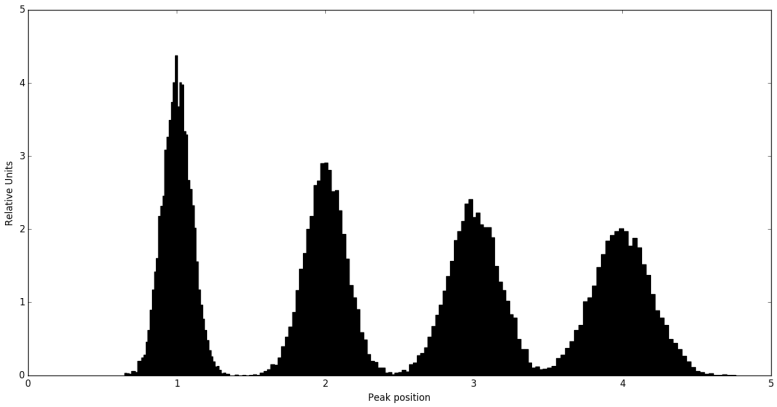

Figure 1.2: Convolved gaussian distributions.

I replicated uniform distribution showed in fig 1.1 with SiPM-like distribution, and the results is plotted in fig 1.3. The resolution for each PE is $\sim$ 10%.

First N-Pe values are well resolved from each other (quantized spectrum), but for $N > 10$ the convolution effect makes single N-PE peak unresolved and for this the spectrum becomes discrete uniform distribution with the same value of fig 1.1 (0.5 cts/s).

1.1.2 Gaussian distribution

The normal or standard Gaussian distribution has pdf:

$$f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} = N(\mu,\sigma) \tag{1.2}$$

or for a non-normed distribution:

$$f(x;\mu,\sigma^2,A) = Ae^{-\frac{(x-\mu)^2}{\sigma^2}} = G(\mu,\sigma,A) \tag{1.3}$$

[4] numpy.random.normal http://docs.scipy.org/doc/numpy/reference/generated/numpy.random.normal.html

[5] numpy.concatenate http://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html

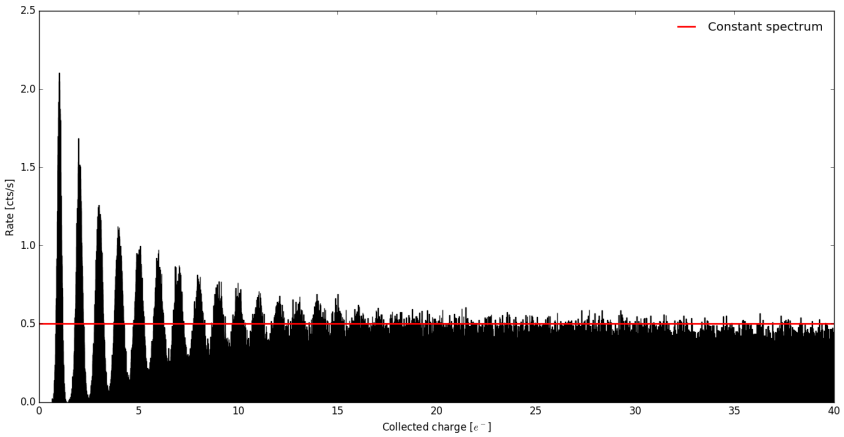

Figure 1.3: Discrete uniform distribution simulated with SiPM criterion.

The aim of this section is to explain how I simulated a gaussian distribution seen from a SiPM.
First I created a normed gaussian centered in $14e^-$ and $\sim$ 78% of resolution as laboratory measurements reported in Chap(Instrumental), that I called N_T. (fig 1.4).

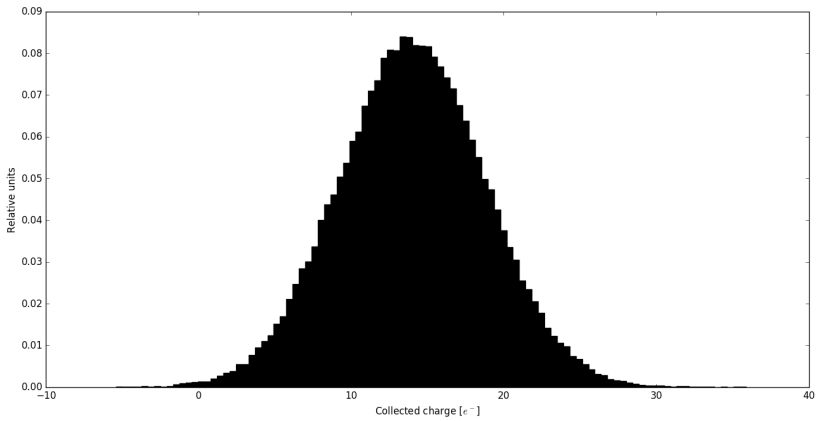

Figure 1.4: Gaussian normed distribution N_T.

I calculated each SiPM n-PE distribution (G_N as follows:

$$G_n = N(n, \sigma\sqrt{n}) \cdot N_T(x = n) \tag{1.4}$$

where $N(n, \sigma\sqrt{N}$ is a normed gaussian centered in n and with required σ variation, and $N_T(x = n)$ is the starting distribution computed in $x = n$. The result of this step is showed in fig 1.5.

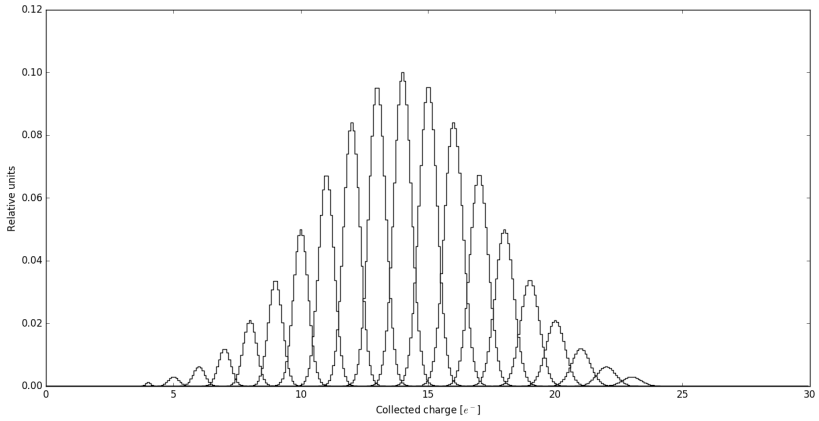

Figure 1.5: n-PE distribution for a gaussian function.

The equivalent gaussian distribution for a SiPM is calculated by:

$$G_{tot} = \sum_n N(n, \sigma\sqrt{n}) \cdot N_T(x = n) \tag{1.5}$$

and showed in fig 1.6.

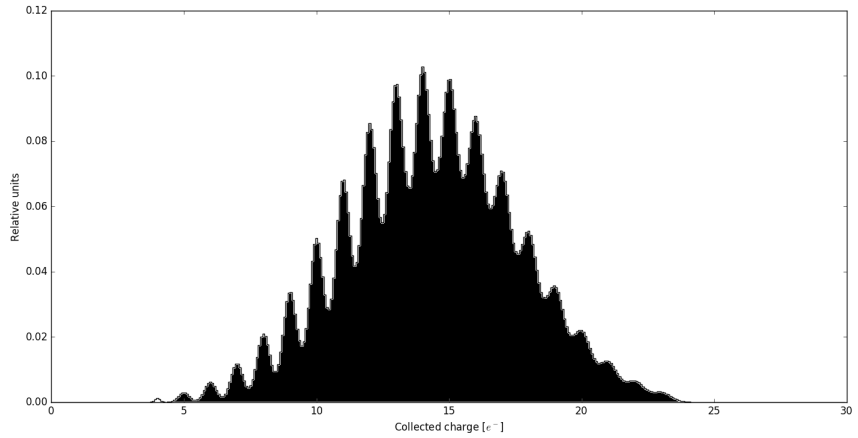

Figure 1.6: n-PE distribution for a gaussian function.

1.2 Fit strategy

After simulating ideal SiPM spectra, the fundamental step is to study a fit strategy that converges to original spectra calculated in fig. 1.1 and fig. 1.4.

Basically we have a mixture of gaussians to fit. Fitting process is not stable when there are a lot of functions due to large number of degrees of freedom. For this is fundamental to fix some parameters or choose initial values. In my work I found very useful to choose gaussian peaks as fit initial values, because are intrinsic properties of n-PE distribution and are well distinguished form each other until are resolved. This process is explained in Par 1.2.1. Than I performed a multi-gaussian fit (Par 1.2.2) from which I obtained information on single n-PE FWHM (par. 1.2.3), area of each one and their linearity in position calibrated in charge (par. 1.2.4).
This process makes possible to extract physical spectrum (or information) from SiPM characteristic output signal. From this one starts the last algorithm that extrapolate information about resolution, total area and collected charge of a X-ray photo-peak (par. 1.2.5). The whole logical process is schematized and showed in Fig 1.7

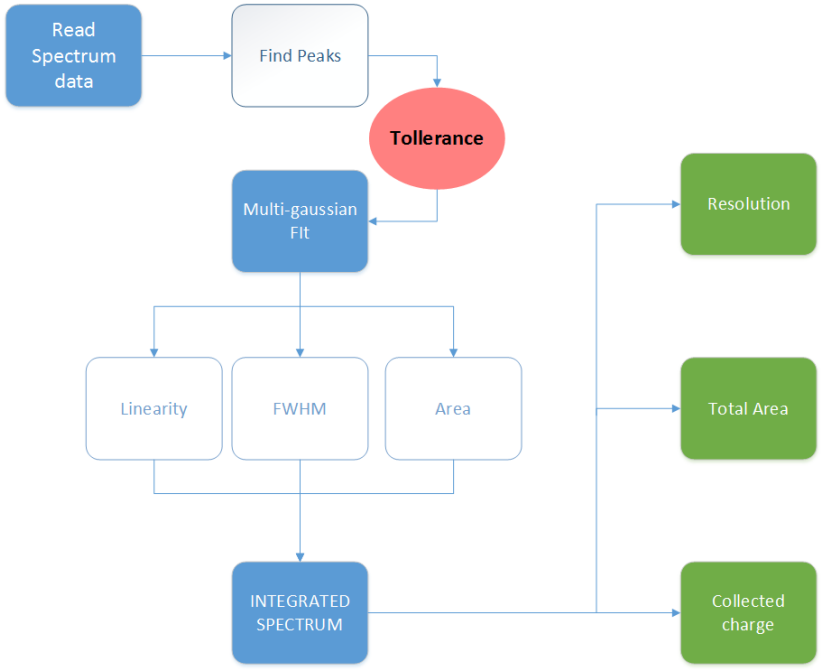

Figure 1.7: Software logical scheme.

1.2.1 Peak-finder

Peak finding is a tricky and often required task in experimental data analysis. When a functional form is known, it is possible to fit the data to this functional form. However, often a more numerical approach is required. The `AnalyseFile.peaks()`[6] provides a relatively simple and effective method for doing this.

[6] http://pythonhosted.org/Stoner/classesStoner.Analysis.AnalyseFile.peaks.html

The algorithm used is to differentiate the data with a Savitsky-Golay filter[7], which in effect fits a polynomial locally over the data. Zero crossing values in the derivative are located and then the second derivative is found for these points and are used to identify peaks and troughs in the data.
In this package there is a `significance` parameter too. The `significance` parameter controls which peaks and troughs are returned. If `significance` is a float, then only peaks and troughs whose second derivatives are larger than `significance` are returned. If `significance` is an integer, then maximum second derivative in the data is divided by the supplied `significance` and used as the threshold on which peaks and troughs to return. The `width` and `poly` keywords are used to control the order of polynomial and the width of the window used for calculating the derivative. A lower order of polynomial and wider width will make the algorithm less sensitive to narrow peaks.
I integrated `AnalyseFile.peaks()` in my own algorithm that i called `find_peak` .
This script is equipped of an optimized version of `significance` parameter that I named `tollerace`. This function allows to improve robustness of peak-search for a multi-gaussian spectrum avoiding probability of revealing statistical fluctuations as peaks and it is based on *peak-to-valley* ratio.
An example of script core is reported as follows:

```
d, x, y = Data                          #Acquisition data
error = sqrt(rate)                      #Counts error
a = #peak-to-valley ratio in %

for i in range(1, n+1)
    min(n) = min(y[n+1] - y[n])
    max(n) = max(y[n+1] - y[n])
    tollerance = min(n)/max(n)

def find_peak(n, rate, peaks)
    if tollerance > a
       d.peaks(width=8,poly=4,significance=tollerance)
       returns peaks
    else peaks == nan
         quit = True
d.find_peak()
```

While `tollerance` is less then a certain value, the script detects all peaks over the threshold, otherwise it stops to find other peaks. In fig 1.8 is reported a comparison between high and low `tollerance` value.

[7] https://en.wikipedia.org/wiki/Savitzky-Golay_filter

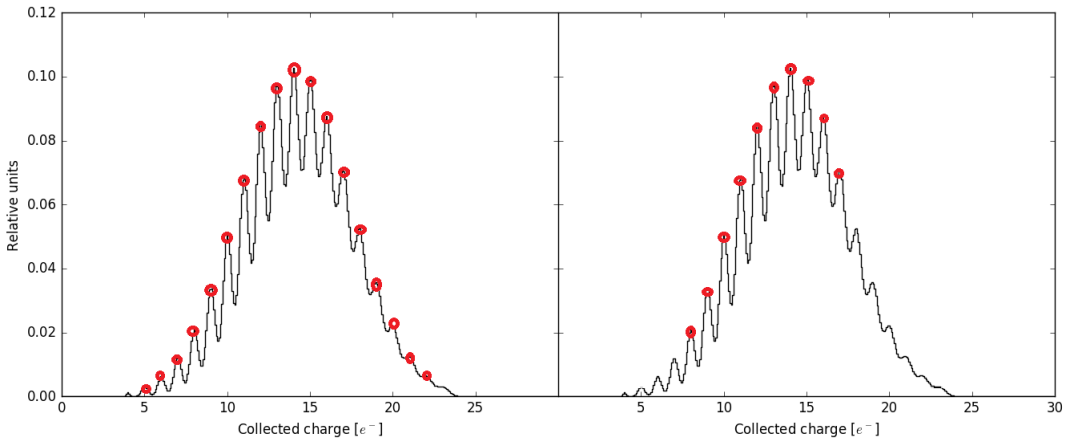

Figure 1.8: *Left:* 5% tollerance value. *Right:* 50% tollerance value

1.2.2 Multi-gaussian fit

Previous paragraph gives fit initial values to starting analysis. A Multi-gaussian fit, with more than 20 gaussians, is a really challenge test for common fitting tools convergence. Peaks-positions values make fit works only if `tollerance` value is appropriate. Choosing right `tollerance` parameter is not a lucky process, but strongly depends on 'purity' and symmetry of spectrum. For SiPM ideal spectra tollerance(τ) is $\sim$ 5% for a smoothly fit convergence. For LCT5/1 spectra @+3V $\tau \sim 30 - 35\%$ due to small distortions (afterpulses, pile-up, etc.) and for really distorted signals like LCT4/10 $\tau > 65\%$. This could be a threshold for a not-analyzable SiPM spectra.
The fitting script, that I called `mg_fit`, has a code core showed as follows:

```
def mg_fit(x, *params):
    y = np.zeros_like(x)
    for i in range(0, len(params), 3):
        ctr = params[i]
        amp = params[peaks[i]]
        wid = params[i+2]
        y = y + amp * np.exp( -((x - ctr)/wid)**2)
    return y
```

that is a simple iteration using `lmfit` libraries for fitting models and statistical tools, and `find_peak` results to force convergence. I analyzed constant and gaussian simulated spectra from previous paragraphs. Results in Fig. 1.9 and Fig. 1.10.

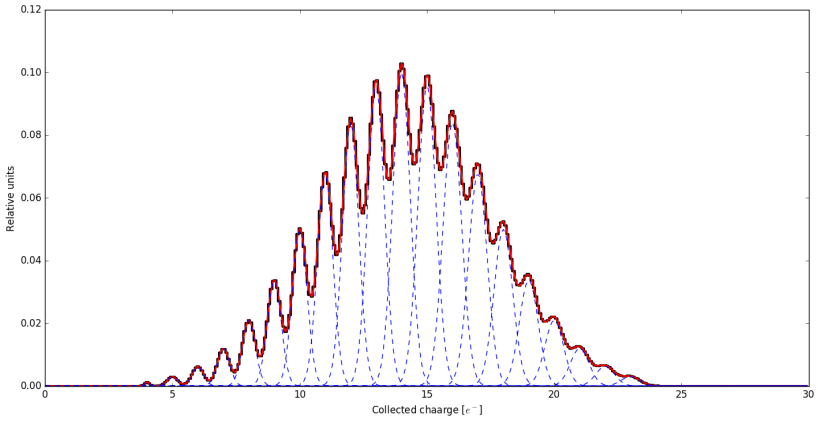

Figure 1.9: Fit of gaussian distribution spectrum

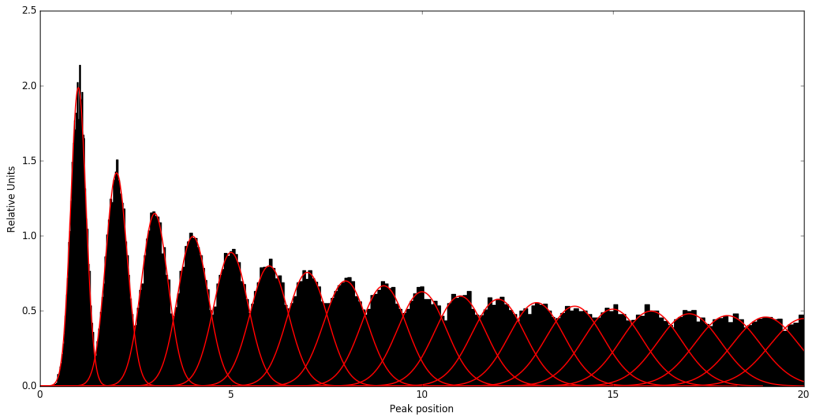

Figure 1.10: Fit of constant distribution spectrum.

1.2.3 Study of $FWHM/\sqrt{N}$ linearity

SiPM poissonian distribution is a predictable feature of detected spectra. One of the most important proof for poissonian distribution test is the $\sigma_{PE_n}/\sqrt{n}$ linearity, or the equivalent $FWHM_{PE_n}/\sqrt{n}$. When this fraction is not constant (on a certain uncertainty), the analyzed spectrum could be affected by noisy factors as electronic instability or pile-up. This feature is easy to detect because comes from `mg_fit` results. I created a secondary script called `fwhm_lin` that saves FWHM values in a independent array which divides it for $\sqrt{n}$. I tested this script on previous results. Gaussian and constant simulated spectra have the same statistical 'birth' than I can choice the first one as representative for both. `fwhm_lin` results are showed in fig 1.11

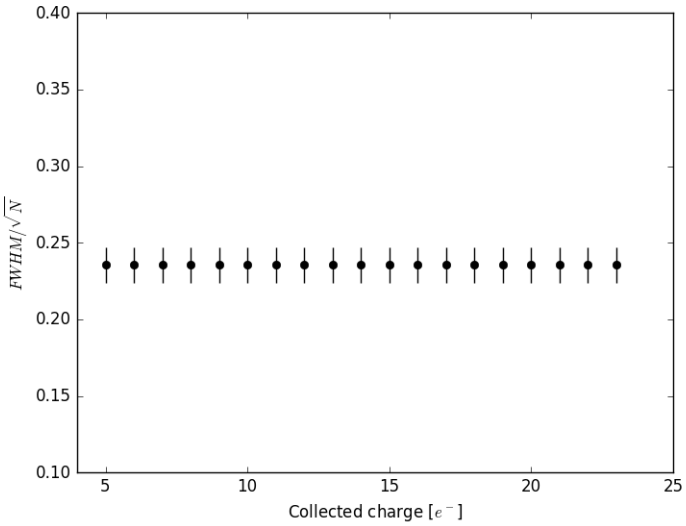

Figure 1.11: `fwhm_lin` results for $FWHM/\sqrt{N}$ linearity.

The results is obviously constant for an ideal spectrum and the value is ~ 0.23. This is a SiPM intrinsic number that is strictly linked to PE resolution since:

$$Res = \frac{FWHM}{n} \tag{1.6}$$

therefore

$$\frac{FWHM}{\sqrt{n}} = Res \cdot \sqrt{n} \tag{1.7}$$

Thus ratio lower values are better for a good single PE resolution and for a higher resolved PE peaks.

1.2.4 Linearity of n-PE position

Another fundamental SiPM feature is the n-PE position linearity. For a calibrated spectrum (in charge), difference from n-PE peaks should be $\sim 1e^-$. I created a simple script called `position_diff` that saves n-PE centroid values from `mg_fit` in a independent array and makes the following operation:

$$PE_{n+1} - PE_n \tag{1.8}$$

Results from gaussian spectrum is showed in fig 1.12.
A non-linearity result it could be caused by a wrong `tollerance` value or a possible signal saturation.

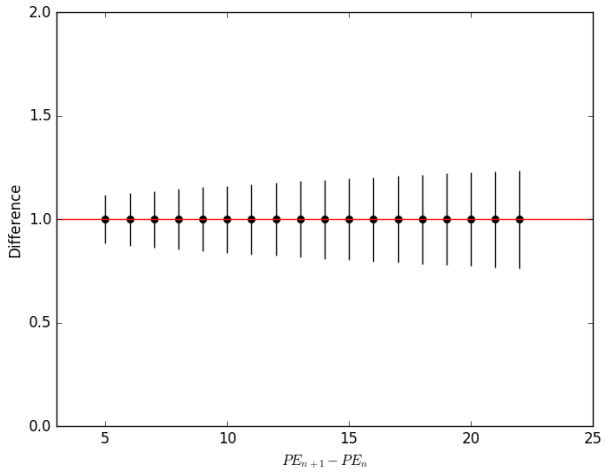

Figure 1.12: `position_diff` results for gaussian spectrum.

1.2.5 Fit of integrated area

After making sure that `fwhm_lin` and `position_diff` have accorded results to SiPM physics, the integrated spectrum should be right for analysis and physical interpretation.
Integrated final area is the result of integrated single n-PE areas. I wrote a script called `fit_area` that makes the following operation:

$$Area_n = A_n \cdot \sigma_n \cdot \sqrt{2\pi} \tag{1.9}$$

where A_n is the n-peak amplitude. In the same script there is a fit function too that can choose to linear or gaussian fit for the final spectrum. I tested this script on the simulated gaussian spectrum showed in fig 1.6. The result in fig 1.13.

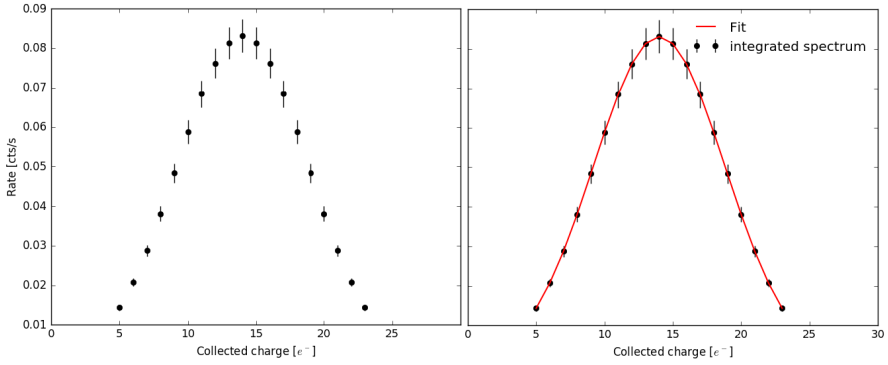

Figure 1.13: `fit_area` results for gaussian spectrum.

The gaussian result could be a photo-peak detection (as in my case) and the fit results give us information as peak position , resolution and counts per second.
The final step of testing software is to compare the integrated spectrum with the original gaussian showed in fig 1.4. Compared values are reported in fig 1.14 that makes 1-σ accordance between simulation and integration.

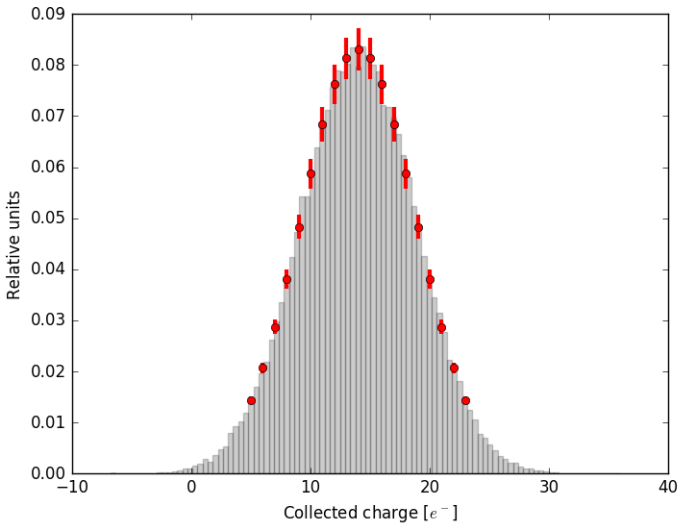

Figure 1.14: Comparison between simulated and SiPM post-processed spectrum