
SPE Templates V2 Technote

Extracting and fitting procedure

Spencer N. Axani - Jan. 23rd, 2020



Introduction

The following tech note describes all the changes to the MC processing implemented to include the *SPE Templates v2*. I've tried to make it as simple as possible to get the useful information to the reader, but there's a lot to cover. We have a 21 page paper that goes through the procedure and results it in detail (see https://wiki.icecube.wisc.edu/index.php/SPE_Templates_Paper). This document will likely be useful for people trying to implement the SPE Templates in their MC, understand terminology, and how to rerun my code.

P.S. The MEOWS analysis used the SPE Templates V2.

The following projects were modified and currently included in COMBO:

1. DOM Launcher:

- Accounts for the discriminator change. (0.93 scaling factor applied to discriminator). This sets the discriminator to 0.2325PE.
- Looks to the GCD file for the SPE charge distribution.

2. Dataclasses:

- Understands the new SPE templates GCD file entries.
- Scales the SPE Template in the GCD file by the **residual**.

3. CLSim:

- Applies the compensation factor. Same manner that the RDE is applied.

4. GCD File:

- Has entries in the I3Calibration object that describe the shape and the charge scaling factor, as well as the compensation factor.

5. Vuvuzela:

- Vuvuzela now looks into the GCD file for the noise parameters.

6. WaveDeform:

- Changed to allow for the `apply_spe_correction` option to be used also with MC. When set to True, the charge extracted by WaveDeform will be scaled by an amount found in the GCD file. This is what distinguishes Pass1 v Pass2.

To minimize the uncertainty on the fit, for a given DOM, we take all the pulses from 2011-2017 and fit to that. This is what is called the **AVG** distributions. These are what will be used for MC. The Gaussian Mean will eventually be used for calibrating the DOMs gain settings. This is a yearly dependent value.

A useful file for people looking into correlations, or seeking further information about the DOMs can be found in this JSON file. It contains all the information I've collected about the DOMs over the years:

```
/data/ana/SterileNeutrino/IC86/HighEnergy/SPE_Templates/SPE_harvesting/scripts/jobs/DOM_properties.json
```

How to use the SPE Templates

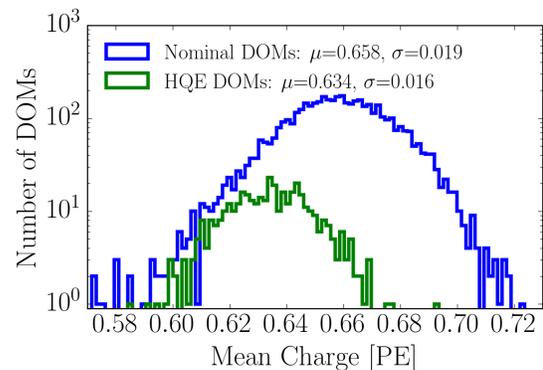
1. Use the trunk of COMBO for Detector level. This requires py2v3.1.1.
2. Use my GCD file. They will have {Pass1,Pass2,Pass3} in the label indicating which pass you want to simulate. i.g. if you are going to compare to Pass2 data, use the Pass2 GCD file for simulation.
3. Use the updated icerec so that you can set WaveDeform apply_spe_corection to True in L1:

```
tray.AddSegment(OnlineFilter, "OnlineFilter",
    decode=False, simulation=True,
    vemcal_enabled=False,
    alert_followup_omit_GCD_diff = True,
    needs_wavedeform_spe_corr = True,
    **online_kwargs
)
```

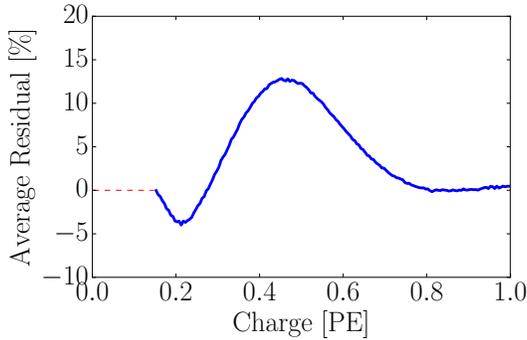
Key words and overview of modifications

SPE Templates: The GCD file now contains the description of the individual DOM single photoelectron charge distribution. This is represented as the sum of two exponentials and a gaussian. DOM Launcher assigns a charge to the photoelectron by sampling from the SPE charge distribution. A single DOM's charge distribution is now described by the shape in terms of $\text{exp1}+\text{exp2}+\text{Gaussian}$, a global residual correction (see below), a compensating factor to scale the DOM efficiency, and a charge scaling factor to mimic the Pass2/Pass3 shifts.

Compensation Factor: Since the SPE templates change the average amount of charge per photoelectron, this shifts the DOM efficiency. To compensate for the DOM efficiency shift, we scale the number of photons produced by the average charge of the TA0003 distribution by the average charge of the SPE templates. It turns out to be ~ 1.3 . CLSIM generates roughly 30% more photons (internally), in the same what that it handles the HQE DOMs. Note: HQE DOMs have a noticeably different shape than the NQE DOMS, resulting in their mean charge being 3.5% lower than the NQE DOMS (Figure below).



The compensation factor is set to 3.5% higher (effectively 3.5% higher RDE) so that their total charge collected remains the same after implementing the SPE Templates.

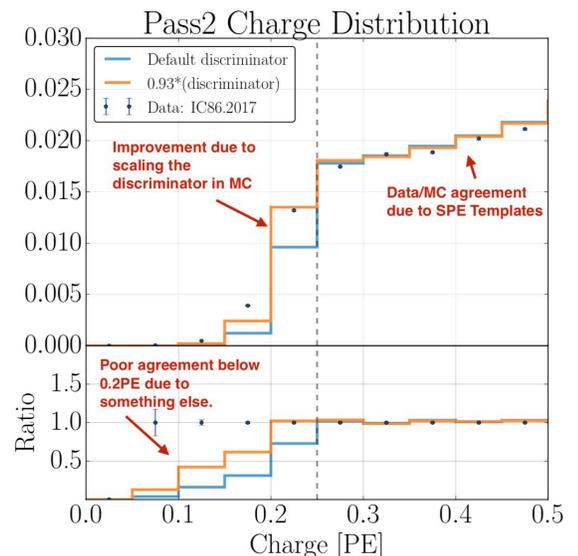


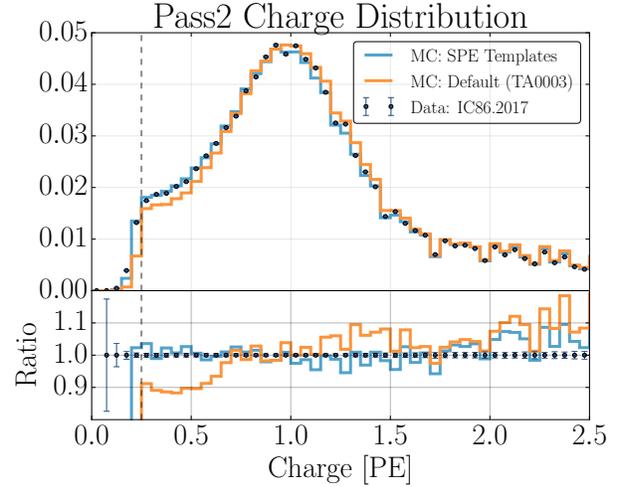
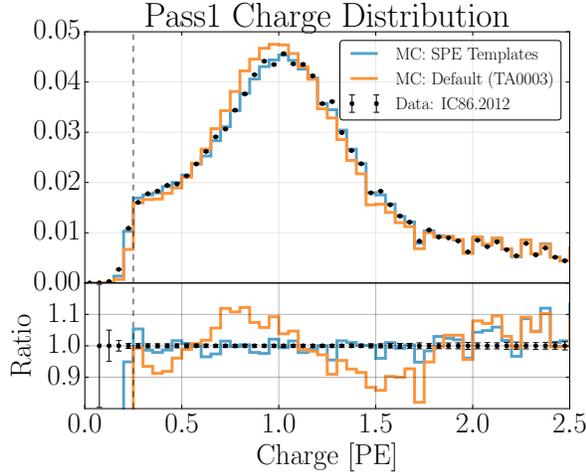
Residual Correction: The functional form of the SPE Template is assumed to be Exp+Exp+Gausn. This likely isn't the exact shape of the distribution in nature. After we extract the values from each fit, we calculate the residual (how far off the fit is from the actual distribution in percentage points). The average residual is used as a correction to all of the charge distributions. The plot on the left shows that the Exp+Exp+Gausn distributions underestimate the

measured distribution by ~13% at 0.45PE. This correction is applied in dataclasses.

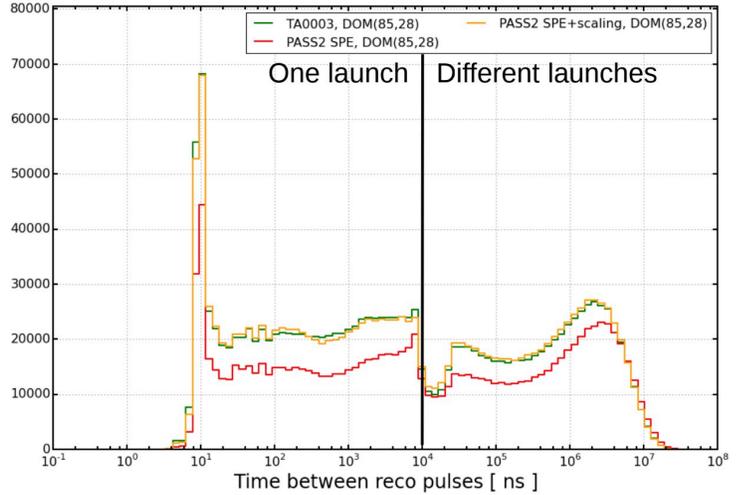
MeanATWDCharge and MeanFADCCharge: These entries in the I3Calibration object in the GCD file determine the scaling factor used by WaveDeform to scale the charge. It allows you to scale the charge identically to that used in Pass1/Pass2/Pass3. Simply set these values to the values used in Data, and it will be properly scaled as well in MC. For example, Pass2 scaled the charge of DOM (1,1) from the ATWD by 1.032, if you want to simulate Pass2, then MeanATWDCharge is set to 1.032. The labelling in the GCD file tells you what scaling factor I've applied. So if you use the GCD file with the word "Pass2" in it, it means that the value for MeanATWDCharge is set to 1.032 for DOM(1,1).

Discriminator setting: In developing the SPE Templates, the measured charge distributions were found to have data/MC disagreement near the discriminator. Originally it was set to 0.25PE, however, PE refers to the quantity that is output from WaveDeform. The discriminator triggers on the waveform voltage. By simulating several sets, a value of $0.93 \times (\text{discriminator threshold})$ was found to produce the best results. This was found independently in LE/Osc and in calibration by Martin Rongen. The discriminator modification is implemented in DOMLauncher.





Vuvuzela noise update: The SPE Templates require an update to the noise. To reproduce the timing distribution of the noise, the noise is scaled by the area above the discriminator compared to the area above the discriminator of the TA0003 distribution. A comparison between the noise before and after the scaling is shown in red and yellow below, while the noise model used before the SPE templates is shown in green.



Introduction

The single photoelectron (SPE) templates represent the probability density function for the charge distributions on all in-ice DOMs. The functional form used to describe the distribution is the sum of two exponentials and a Gaussian, $\text{Exp}_1 + \text{Exp}_2 + \text{Gauss.}$, explicitly:

$$f(q)_{\text{SPE}} = \frac{P_{e1}}{w_1} \cdot e^{-q/w_1} + \frac{P_{e2}}{w_2} \cdot e^{-q/w_2} + \frac{1 - P_{e1} - P_{e2}}{\sigma \sqrt{\pi/2} \cdot \text{Erfc}[-\mu/(\sigma\sqrt{2})]} \cdot e^{-\frac{(q-\mu)^2}{2\sigma^2}}$$

Where charge, q , is the quantity calculated in WaveDeform. WaveDeform deconvolve the digitized waveform into a pulse series (pulse times and charges). This is accomplished by fitting SPE pulse templates (templates which are dependent on the version of the AC coupling used in the DOM) to the digitized waveforms via a χ^2 . The charge associated with a fitted pulse is determined by dividing the the area of the fitted SPE pulse template by the load resistance (and further scaling it by the gain of the PMT). WaveDeform is capable of fitting multiple SPE pulse templates to multi-PE pulses, and deconvolving them into their separate photoelectrons. Since it may occasionally deconvolving a non-multiPE pulse, we must **always** sum up the pulses from WaveDeform to make a comparison between MC and Data (i.e. we need to reassemble pulses that may have been accidentally split).

For this study, all frames (events) that pass the MinBias filter are then passed through WaveDeform. This filter works by randomly selecting HLC events, at a rate of 1 / 1000 triggers. The MinBias filter saves the full digitized waveform from the ATWD and fADC. This allows us to rerun WaveDeform with different settings (this will be useful later). This study will also use the BeaconLaunch dataset. This dataset represents forced time windows. That is, it is not triggered by a pulse, rather, it is triggered by a time. Normally, a random force trigger will readout the ADC baseline, and it is rare that it record actual pulse. Since these frames are primarily baseline fluctuations, we can use this dataset to analysis the accidental pulse reconstruction associated with ADC fluctuations (referred to as noise).

This tech note will begin with a set of condensed instructions for those familiar in the procedure already. Then, I'll provide much more informative description of every step. We use a modern simulation Metaproject (e.g. V06-00-03), and recent updates to WaveDeform mean that we need to use the current trunk versions (<http://code.icecube.wisc.edu/svn/projects/wavedeform/trunk/>). Later, this modified version of WaveDeform will be added to the release. Changing the meta project involves simply replacing the hashtag Metaproject identifier at the top of the processing scripts. WaveDeform is only used in the harvesting.py script.

Condensed instructions

The "inlocation" variable will be defined as the location of the GitHub project.

The processing files are located here :

`inlocation + /SPE_harvesting/scripts/jobs/`

The output from these files are sent here:

`inlocation +/SPE_harvesting/`

I typically have a "setup-X.py" file that generates the dagman to run X.py script on the cluster. You will have to modify the setup-X.py script so that it generates dagmans in your scratch/ dir. You will also have to copy the .submit scripts for the dagman to your location.

Essentially there are four steps. We first **harvest** the pulses from the MinBias and BeaconLaunch data (you won't need the BeaconLaunch data, but it's automatically generated). Then, we histogram all the found pulses in a season together into a common JSON file. Finally, we fit the distributions using the **convolutional fitter** and make a **GCD file** from the results.

Harvesting pulses

1. Run **setup-extract_i3.py**, by passing it the season name. Like IC86.2011. It will try to find the PFFilt files of that season, and only extract the frames that pass the MinBias filter.
2. The events that pass the MinBias filter are now in an i3 file. Send the output .i3 file of (1) through **harvest.py**. You can run setup-harvest.py but you have to specify what you want this set of pulses to be called in the "**identifier**" variable (like "**OWD_50ns**") and the WaveDeform parameters (you will always want to use **923**). This file will run WaveDeform on every frame. It will find the pulses of interest (pulses that pass the pulse selection). The output of harvest.py will be JSON files that contain the individual pulses extracted sorted by each DOM and digitizer. The name of this harvest set will be called **OWD_50ns_923**, and a folder will be generated for the season of interest with that name.
3. Merge the all the files in (2), by running merge_harvest.py. You need to specify the name from the previous step (**OWD_50ns_923**). During this merging process, the individual charges are binned. We use a 1000 bins from 0 to 5PE for the ATWD and 200 bins over the same range for the SLC.

The output of the pulse harvesting is a single JSON per season which includes the binned charges for all DOMs, for ATWD0 and FADC (SLC), for both the MinBias and for the BeaconLaunch.

Fitting and making the GCD file

1. Run `setup-convolutional_fitter.py`. You need to specify the name of this fit set in the “description” variable, as well as the name of the pulse set above in the “identifier” variable. You will fit one string at a time by passing the string number (1-86) and digitizer (ATWD or SLC) to the convolutional fitter (`convolutional_fitter.py`). It will produce a JSON file per string, in a directory called “pdf,” along with plots of all the fits. I start the fit for ATWD at bin 30, and bin 14 for the SLC. The convolutional fitter requires the identifier name, which just specifies the file merged in step (3) above.
2. The output will be a JSON file that contains the fit parameters for all the DOM on the desired string and digitizer. These now need to be merged using the `merge_convolutional_fitter.py` script. The output is a fantastic file that contains all the fit information (plus all other information that I have collected on all the DOMs — it’s a really useful file). It will also produce a file with the “_lite” name which contains the minimum amount of information for the SPE Templates.
3. Pass the fitted JSON file to `make_GCD_SPE.py`. You must specify what kind of GCD you want. Do you want the Gaussian mean to be at unity? Do you want to simply produce the TA0003 distribution, or do you want to use the SPE Templates?

The not at all condensed instructions

Extracting pulses

Since we are interested in single photoelectrons, we are using the digitized waveforms from ATWD0 are used. We cannot use the trigger pulse, since this would limit us to the discriminator threshold (0.25PE) therefore we devised a method to extract pulses after the

window is open. This allows us to collect charges down to the limit of which WaveDeform will try to extract pulses. The pulse selection is described in this section.

The pulse selection is the method used to extract candidate, unbiased, single photoelectron pulses from data, while minimizing the multi-PE contamination. It avoids collecting afterpulses, rejects late pulses from the trigger, reassembles late pulses, accounts for the discriminator threshold, reduces the effect of droop and baseline undershoot, and gives sufficient statistics to perform a season-to-season measurement. An illustrative diagram of the pulse selection is shown in the left side of Fig. 1, while a description of the procedure is detailed below.

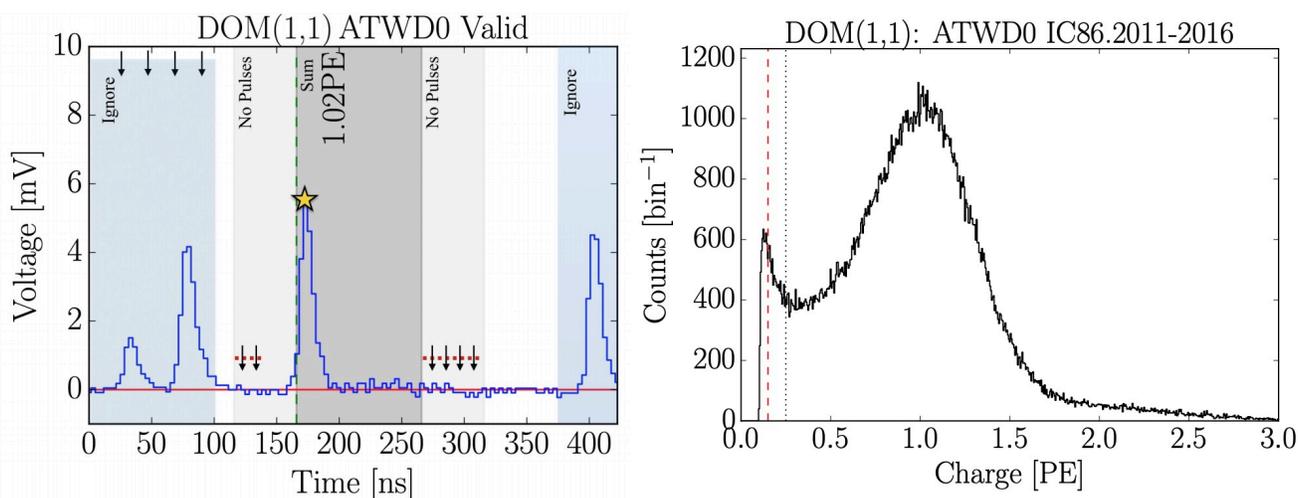


Fig. 1: Left: an illustration of the pulse selection. Set the ‘pdf’ option in SPE_harvest.py to make these plots for every pulse. Right: the extracted charge distribution for DOM 1,1. You could make this plot after JSON files have been merged.

In order to trigger a DOM, the input to the front-end amplifiers must exceed the discriminator threshold. To avoid the selection bias of the discriminator trigger, we ignore the trigger pulse as well as the entire first 100~ns of the time window. Ignoring the first 100~ns has the added benefit of also removing late pulses that could be attributed to the triggering pulse. To ensure we are not accepting afterpulses into the selection, we also enforce that the pulse of interest (POI) is within the first 375~ns of the ATWD time window. In the vicinity of the POI, we check that WaveDeform did not reconstruct any pulses up to 50~ns prior to the POI, or 100-150~ns after the POI (the light-gray region of Fig.~\ref{fig::pulse_selection} Left).

This later constraint is to reduce the probability of accidentally splitting a late pulse in the summation window.

Restrictions are put on the full ATWD waveforms as well, such as ensuring that the trigger pulse does not exceed 10~mV (to reduce the effect of the subsequent baseline undershoot due to the AC coupling or other artifacts from large pulses) as well as a global constraint that the time window cannot contain any pulses that exceeds 20~mV.

If a pulse is reconstructed between 100 and 375~ns after the time window is opened and the voltage criteria is met, it is accepted as a candidate photoelectron and several checks are performed on the waveform prior-to and after the pulse. The first check is to ensure that the waveform is near the baseline just prior to the rising edge of the POI. This is accomplished by ensuring that the waveform does not exceed 1~mV, 50 to 20~ns prior to the POI, and eliminates cases where the POI is late pulse. We also ensure the waveform returns to the baseline by checking that no ADC measurement exceeds 1~mV, 50 to 100~ns after the POI (these constraints are illustrated as the red-dotted lines and black arrows in Fig.~\ref{fig::pulse_selection} Left).

If all the above criteria are met, we sum the reconstructed charges from the POI time (given by WaveDeform) to +50~ns (the dark gray area of in Fig.~\ref{fig::pulse_selection} Left). This ensures that any nearby pulses are either fully separated or fully added (in-case WaveDeform incorrectly split the pulse, and to reassemble late pulses). The 50~ns summation also means that the pulse selection we will occasionally be accepting multi-PE (MPE) events.

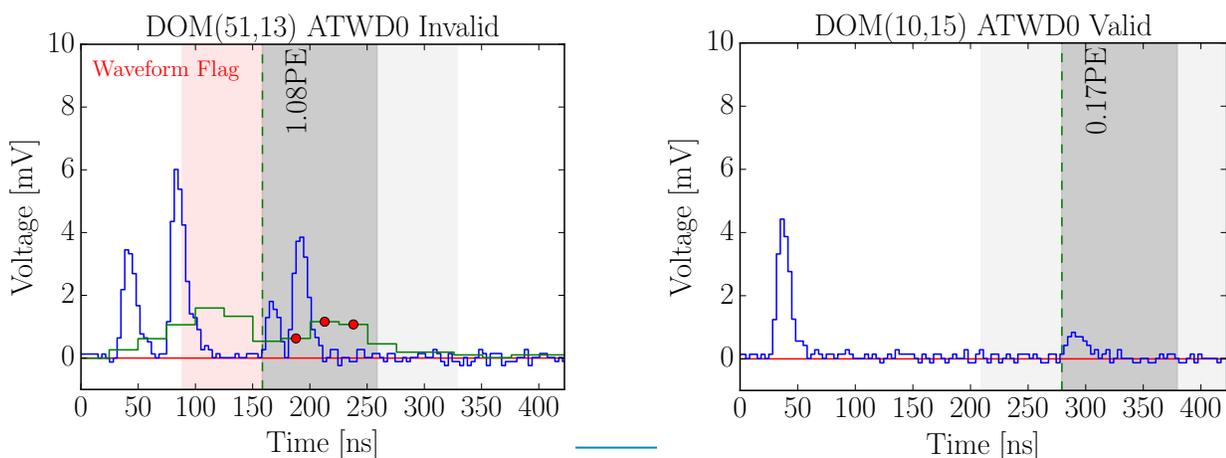


Fig. 3 (left) shows a pulse that we reject. The reason for this is that there is another pulse too close to the one of interest. Specifically there is a waveform flag, indicating that the voltage before or after the pulse was too high. You can also see the fADC in green, and the calculated SLC pulse with the red dots. The pulse of interest is the smaller pulse in the grey time window, followed by what looks like a late pulse. The sum of these two pulses is found to be 1.08PE (sum of the two output from WaveDeform). Fig. 3 (right) shows a sub discriminator threshold pulse (0.17PE).

How to extract Pulses

We first run the PFFilt files through `extract_i3.py`, which simply removes all frame from these files (`/data/exp/IceCube/'+year+'/filtered/PFFilt/'+folder+'/*bz2'`) that do not pass the MinBias filter.

These files are then sent to `harvest.py` script. This script requires:

1. MinBias files. From `extract_i3.py` but only contains frames that pass the MinBias filter. Example files:

`inlocation + /SPE_harvesting/IC86.2017/i3/2017_10/PFFilt_MinBias_IC86.2017_Y2017_M10_D14_Run130125.i3.bz2`

2. The correct GCD file for the PFFilt file:

`/data/exp/IceCube/2017/filtered/level2/1014/Run00130125/Level2_IC86.2017_data_Run00130125_1014_68_344_GCD.i3.zst`

3. The processing script `harvest.py` requires several options:

1. Infile - the name and location of the MinBias file
2. Outfile - the name and location of the output file
3. GCDfile - the name and location of the gcd file
4. Pdf - This is the desired location to save the waveform of the pulse. If not specified, the pulse will not be plotted. Setting this option, say to `"/data/user/saxani"` like the one shown in Fig. 3.
5. NFrames - The number of frames to process. Set to 0 to process the full file.
6. Skip_beacons - set to string `"True"` to skip the beacon pulses and only look at the non-beacon pulses. Otherwise, it will run the BeaconLaunches first, then look through the non-BeaconLaunches.

7. Tolerance - this is a parameter passed to WaveDeform. It defines a metric of how good the SPE pulse fit should be. The default value, and that used for processing normal IceCube data is 9. I set this to 6 in order for WaveDeform to reconstruct smaller pulses.

8. Noise threshold. This is a parameter passed to WaveDeform. All ADC values below this will be set to zero. Default value for IceCube data is 2.

9. Basis threshold. This is also a parameter passed to WaveDeform. It defines the minimum size of bin to consider fitting. Default value for IceCube is 3.

I run `setup-harvest.py`, which generates the dagman used to submit all my jobs. In this script, I define the IceCube season and identifier name — a string which defines essentially the name of the output folder. The values for the Tolerance, Noise, and Basis Threshold are

added to the identifier. For example, for the year 2011, and an identifier NewWaveDeform, the data from harvest.py is sent to this folder:

```
invocation + /SPE_harvesting/IC86.2011/923_NewWaveDeform
```

The data in this folder is all the pulses that pass the pulse selection, organized by month, by DOM, in JSON files. It is also separated into pulses extracted from ATWD0 and from calculated SLC pulses. The file should be pretty self-explanatory. The next step is to merge all the files in a given IceCube season. This is done with the merge_harvest.py script.

Again, this can be set up using the setup-merge_harvest.py script. In this script, I specify the identifier, or 923_NewWaveDeform (in the previous example). It then generates a dagman that runs jobs to merge all the JSON files from the seasons selected in this script. The merge_harvest.py script also histograms all the ATWD charges into 1000 bins from 0 to 5PE (The SLC binning is more coarse, we use 200 bins from 0 to 5PE). The merged files are sent to, for example:

```
inlocation + /SPE_harvesting/Final_charge_distributions/IC86.2011_923_NewWaveDeform.json
```

Suppose you want the histogram of the charge distribution for DOM (1,1) from IC86.2011 from the WaveDeform with the settings 923, then open the file above with a JSON reader like so:

```
import json

infile = inlocation + '/SPE_harvesting/Final_charge_distributions/IC86.2011_923_NewWaveDeform.json'
with open(infile) as data_file:
    data = json.load(data_file)

charge_dist = data['1,1']['MinBias']['hist']['ATWD']['binContents']
number_of_bins = data['1,1']['MinBias']['hist']['ATWD']['nbins']
min_charge = data['1,1']['MinBias']['hist']['xmin']
max_charge = data['1,1']['MinBias']['hist']['xmax']
```

You could also look at SLC rather than ATWD, or BeaconLaunch rather than MinBias, for any in-ice DOM. There aren't many BeaconLaunch pulses, since the BeaconLaunch is a forced time window, which means that a pulse will either be accidental or noise. Fig. 4 shows the charge distribution for summing over all DOMs, over all seasons, for the MinBias and the BeaconLaunches. These are subsequently divided into the set of data with WaveDeform set to 923 and 623. The 623 set is simply used to try and determine the Low charge region.

Fig. 4 shows that BeaconLaunch data in red, and the MinBias in black. The BeaconLaunch data was scaled by 28.4, which is the ratio of the MinBias lifetime to the BeaconLaunch live time. The noise is determined by subtracting the shape of the MinBias data by the BeaconLaunch data. We see when we modify WavDeform (Fig. 4 right), the amount of noise increases (to a maximum of 1.7% of the data), but allows us to examine the distribution down to approximately 0.10PE. We can verify the factor 28.4 by plotting the low charge region for different settings on WaveDeform. After subtracting the background from the MinBias data, we should see rough agreement between settings. This is shown in Fig. 5 left. If the scaling factor was too large, one would see something like Fig. 5 right.

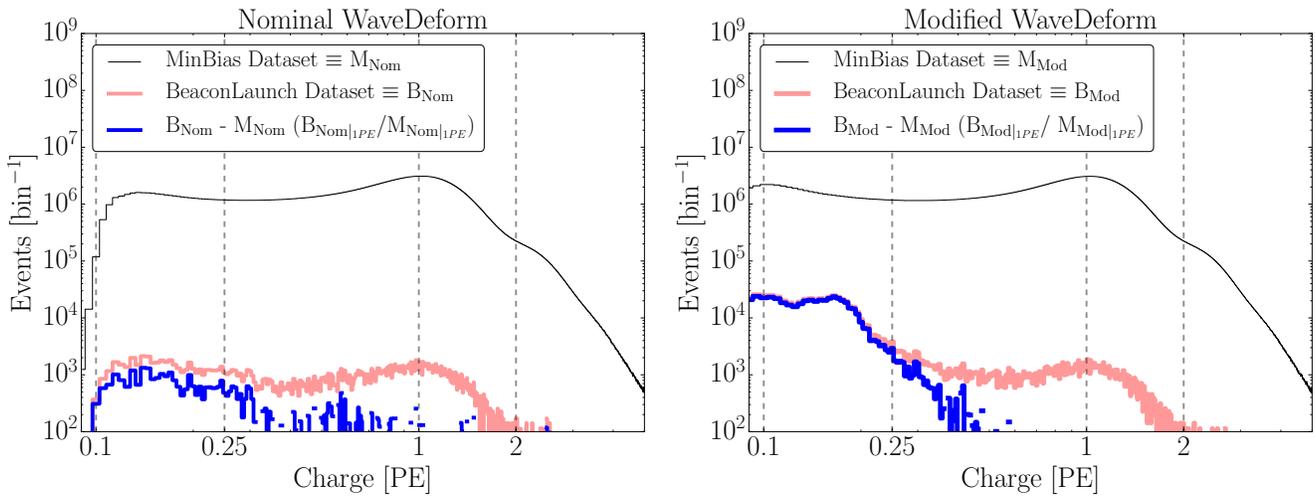
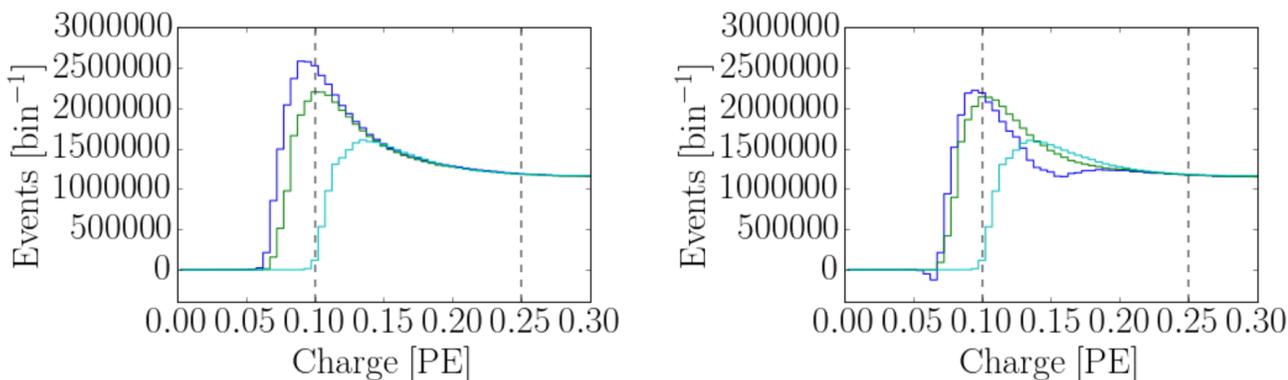


Fig. 4. Left: The normal 923 settings for WaveDeform. Right: setting WaveDeform to 623. We only use the BeaconLaunch data to describe the low-PE region. This region (sub 0.25PE) really only effects the DOM efficiency, since if you randomly sample from a distribution with a large area below the discriminator, that photon will just be thrown away (which is what the DOM efficiency is). It's important to describe the data as accurately as possible, and therefore we do the best job that we can, but in reality, the shape in the region where there is no data is fully compensated for with the DOM efficiency.

Fig. 5. Using three different settings for WaveDeform 923 in light blue, 723 in green, and 623 in dark blue. Left: scaling factor between MinBias and BeaconLaunch set to 28.4. Right: scaling factor set to 100. You can see that with the improper scaling factor, the curves do not agree.



Convolutional Fitting

As of now, we have the charge distributions for all seasons, ATWD, SLC, per DOM, for two settings of WaveDeform (923 and 623). Now, we use the convolutional fitter to fit the background subtracted data. This is done using the `convolutional_fitter.py` script located here:

`inlocation + /SPE_harvesting/scripts/jobs/convolutional_fitter.py`

This file takes a few arguments.

1. Identifier - for example 923_NewWaveDeform.
2. Starting bin - this is the bin you wish to start fitting at. ATWD 923 I set to 30 (0.15PE), SLC 923 I set to 14. For the 623 set, we can lower the starting bin to 20 (0.1PE).
3. year — this is the IceCube season. It selects the year to fit, as well used for the titles on the plots, as well as the name of the outfile.
4. Pdf location — if this is set to a directory, the plot of every fit will be saved to that directory.
5. `set_exp` — if this is set to the string 'True' then it will use the default shape of the EXP1, ie. 6.9 amplitude and 0.032 for the exponential decay. If set to False, you should be using the 623 set, and it will try to determine the shape of the exp1.
6. DOM — if specified, a string like 1,1, it will only fit that single DOM.
7. String — if specified, it will only fit that string. Example, set to 1 to only fit string 1. When I submit jobs to the cluster to fit, I fit one string per job.

8. digitizer — You can specify either ATWD or SLC. This means that it will fit the ATWD or the SLC distribution from the input file.

An example fit for DOM 1,1 with the year set to AVG (sum over 2011-2016 data), the set_exp to True, the starting bin is 30, ATWD, is shown in Fig. 6. In this figure, you can see that the exponential EXP1 is set to the default value of 6.9 and 0.027 decay width. The other components of the fit: EXP2 and Gaus are shown in blue, along with the 2PE component, determined by the convolutional fitter. The purple distribution shows the Full (all years, all DOMs) BeaconLaunch dataset, scaled by 28.4, then scaled again by a factor of 30 such that it is visible. The charge data from the pulse selection is shown in the black histogram, while the full convolutional fit is shown in the black line. The fit values, are all shown in the description on the side, along with the uncertainty. The large uncertainty in the EXP1 component is due to the fact that we are fitting the 923 set, which can't say much about the low-PE region. The text in the plot also shows the number of entries, the validity (whether the fit succeeded), the peak-to-valley ratio, the reduced chi2 of the distribution and fit, and the mean charge of the fit.

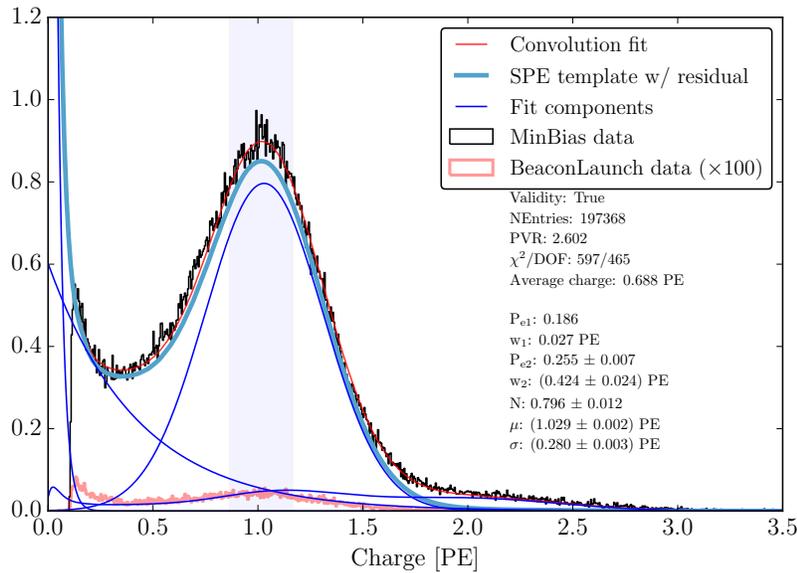


Fig. 6. An example output of the convolutional fitter for the ATWD, DOM 1,1 of all the years data summed together. The SPE Template is the red distribution. The semi-opaque region around the peak is where we prioritize the fitter. This is accomplished by minimizing the uncertainty in the data around the peak. The scipy fitting code has an option called sigma which allows you to specify the uncertainty.

The output of the `convolutional_fitter.py` is a JSON file that contains all the fit information. I perform the fitting on the cluster using the `setup-convolutional_fitter.py` script. Which will fit each string, per job. Then I run the `merge_convolution_fitter.py` script located:

inlocation + /SPE_harvesting/scripts/jobs/merge-convolutional_fitter.py

In the merging file above, it goes through the strings, and merges all the JSON files together. There's often 1 string that might fail on the cluster. If you re-run it, it will succeed. This merging script also populates the final JSON file with all the information related to each DOM. Have a look at this example output file:

`inlocation + /SPE_harvesting/SPE_fits/Fits_923_NewWaveDeform/IC86.2011_923_NewWaveDeform.json`

In this file, you can find a ton of information, such as:

ATWDFitParams: an array of the components of the fit.

ATWDFitParams_err: the error of the fit values

ATWD_fit: The information related to the output of the convolutional fitter for the ATWD.

Absorption: This is the Spice3.2 absorption value at the depth of the DOM.

AnodeSens: The sensitivity of the anode. As Chris Wendt, I'm not sure what the units are.

BlueSens: DOM sensitivity to blue light?

CalDate: I retrieve the HV and Gain values of every DOM at one point per year to compare. These are the dates of the measure HV and Gain settings.

CathSens: DOM cathode sensitivity?

DOMid: the id of the DOM

DeadDOM: if the DOM is labelled as dead in the GCD file, I set this to 1.

DeepCore: boolean if the DOM is part of DeepCore.

DeploymentDay: the MJD date that the DOM was deployed in-ice

DeploymentSeason: the month and year of the deployment.

Depth: The position of the DOM. Distance from top of the ice.

Gain: I record the gain once per year. The measure time is given in the CalDate.

GainAVG: simply the average of the Gain value above.

HQE: if the DOM is HQE, set to 1.

HV: I record the HV of the DOM once per year. The time is given in CalDate.

HV1e7: this is HV setting to get 1E7 given by Hamamatsu? or lab?

HVAvgInIce: The average of the HV values above.

HVSlope: the slope of the HV values above.

MBid: the id of the main board.

MJD: The modified julian date of the deployment.

MeanCharge: the calculated mean charge of the ATWD fit.

NEntries: the number of entries in the ATWD fit.

NickName: the DOM nickname

OM: the optical module number.

Order: the DOM deployment order?

PMTSerial: the serial number of the PMT. We are missing some of these.

PVRInIce: This is my measurement of the PVR.

PVRLab: This is Hamamatsu's measurement of the PVR.

QE: set to 0 if dead. Set to 1 if standard quantum efficiency, set to 2 if HQE.

RDE: This should be 1 for standard and 1.35 for HQE.

RelSens: relative sensitivity? Ask Chris.

SLC_Fit: the fit values from the SLC.

Scattering: the Spice3.2 scattering values at the depth of the DOM (not taking into account position, just depth).

String: The string that the DOM is on.

TemperatureCalc: Temperature calculated from the depth fit.

TemperatureInIce: the measured onboard DOM temperature sensor.

Toroid: the version of the AC coupling used on the DOM. 1 for old, 2 for new. 0 for dead.

Valid: is the fit valid? 1 for true.

Velocity: the velocity of the DOM.

VelocityGradient: the calculated velocity relative to the DOMs below and above it. So 0.01m/yr means that this DOM is moving 1cm per year relative to the DOM above and below it.

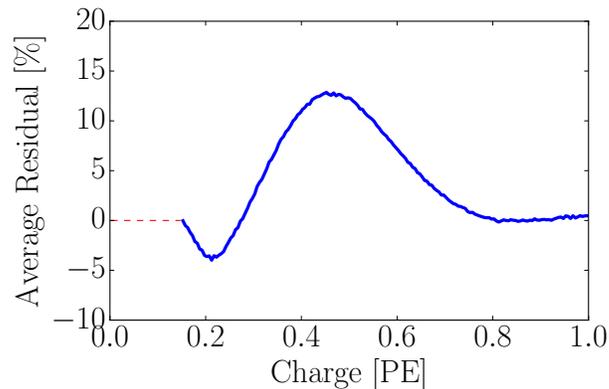
X, Y, Z: the position of the DOM.

The residual can be calculated with the script:

inlocation + /SPE_harvesting/scripts/jobs/make_Residual.py

If you make dramatic changes to the fit, then the residual will likely change. The residual is described as a scaling factor as a function of charge. The distribution is shown

below. In dataclasses, where the charge distribution functional form is defined, there is a long array called **yData** and **xData**, it's simply (1+ the curve below).



Making a GCD file with the SPE Templates

The JSON file that is output from the `merge_convolutional_fitter.py` script contains the values of the parameters in Eq. 1. These values represent the SPE Templates. We now need to insert them into the GCD file. There is a script to do this called `inject_spe_templates.py`.

```
inlocation + /SPE_harvesting/scripts/jobs/inject_SPE_templates.py
```

This file takes several arguments:

1. **Identifier** - the name of the set of convolutional fits.
2. **description** - This just appends a description to the GCD file output name
3. **season** —This is the IceCube season you want to make the GCD for. You will need to modify the script to take in the correct GCD file for you. For MC, it finds the appropriate GCD file to input the SPE_Templates.
4. **scale factor** — set to 1 for not charge scaling. This scales the measured distribution by some factor. For example, suppose you want a Pass2 GCD. DOM 1,1 had its charge scaled by 1.037. If we want to further scale this number by 1%, we can put 0.99 in as a scaling factor.

The options above allow you to specify two things: the scaling factor (ie. how to stretch the distribution), and the shape of the charge distribution (SPE Templates, or default TA0003).

This script creates a field in the `domCal` called `combined_spe_charge_distribution`. This holds the fit values for the SPE template associated with that DOM. There is one caveat to this, it was found that the HQE DOMs in IceCube have a measurable different shape than the standard DOMs. This caused the average charge of their distributions to be 3.5% lower than

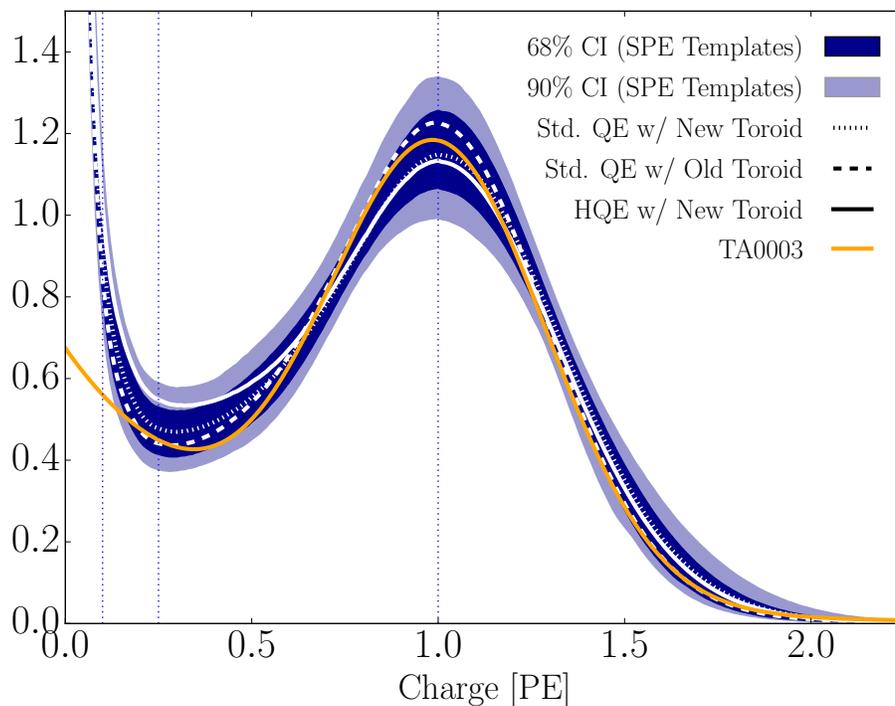
the standard DOMs. In simulation, this would look like a 3.5% change in DOM efficiency for the HQE DOMs. In order to take this into account, we scale the compensation factor by 3.5% (the actual value is calculated in this script).

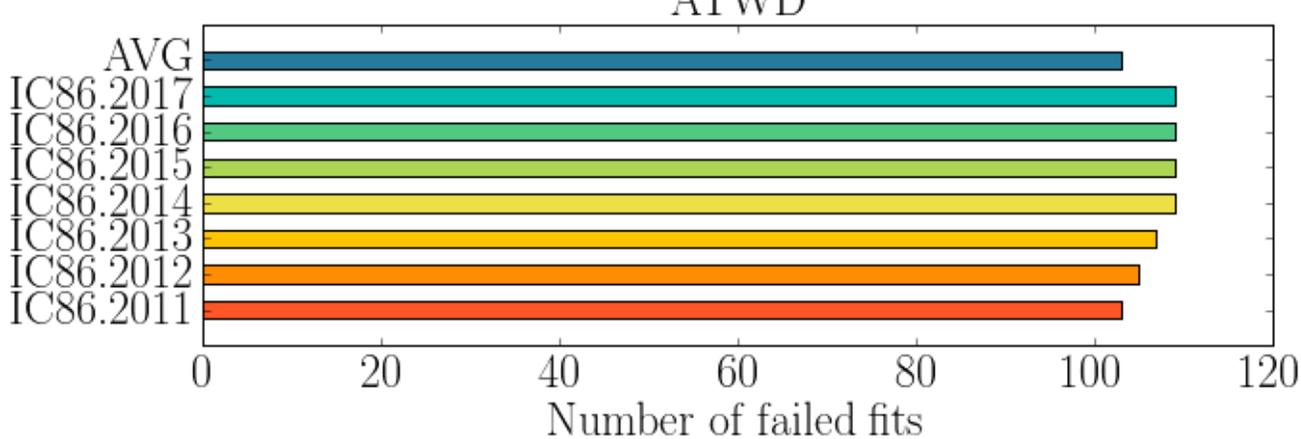
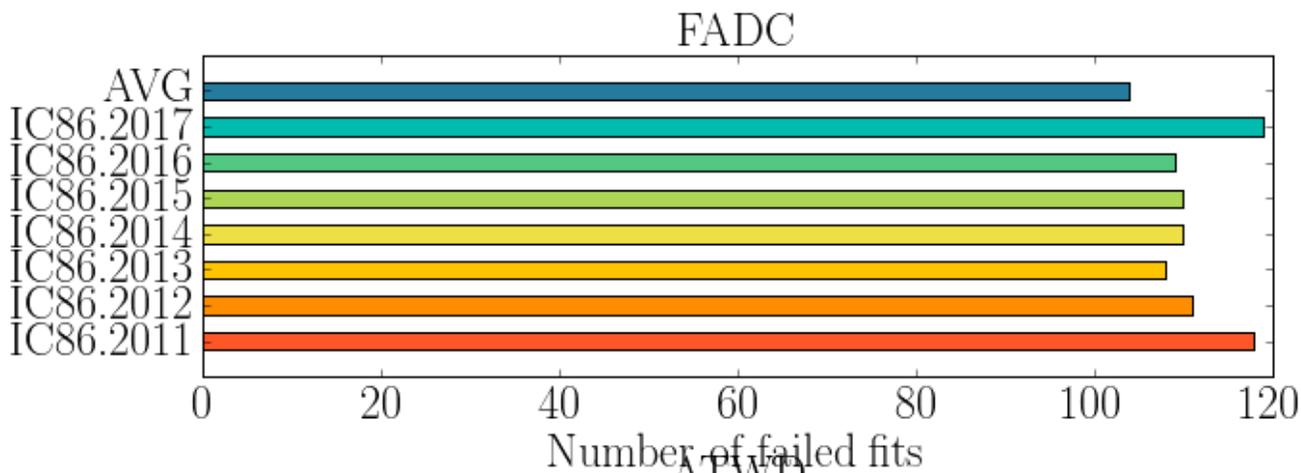
This file also updates the noise in the GCD file according to the SPE templates. We then want vuvuzela to use the values in the GCD file.

This GCD is now ready to be used for Simulation. If using it for data, make sure that you have the correct initial GCD file. The name of the GCD specifies what it is. You want to use GCD files that say "AVG" to indicate that they were fit using all the data. You also want to watch out for "Pass1/Pass2/Pass3". They indicate what the produced MC should be compared against.

Results from fits

The failed ATWD fits are almost entirely dead DOMs. I won't describe each plot, although they should hopefully be self-explanatory. If not, feel free to send me an email at saxani@mit.edu.





CombinedSPEChargeDistribution :

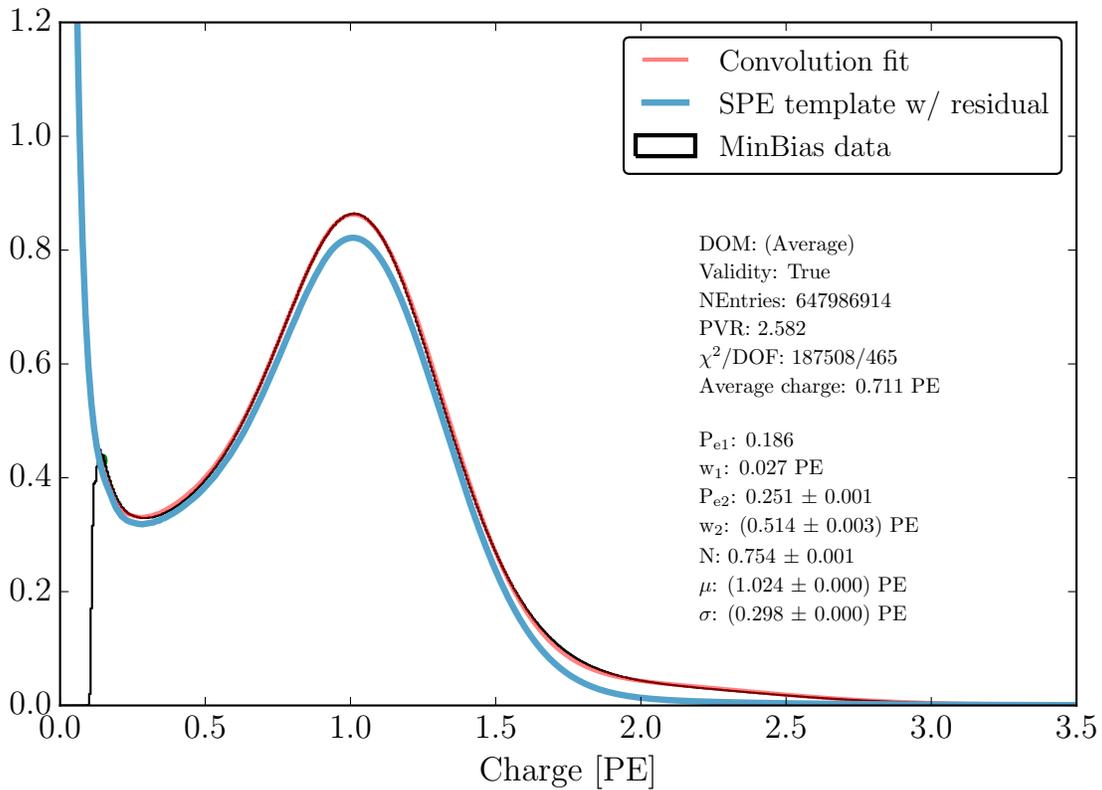
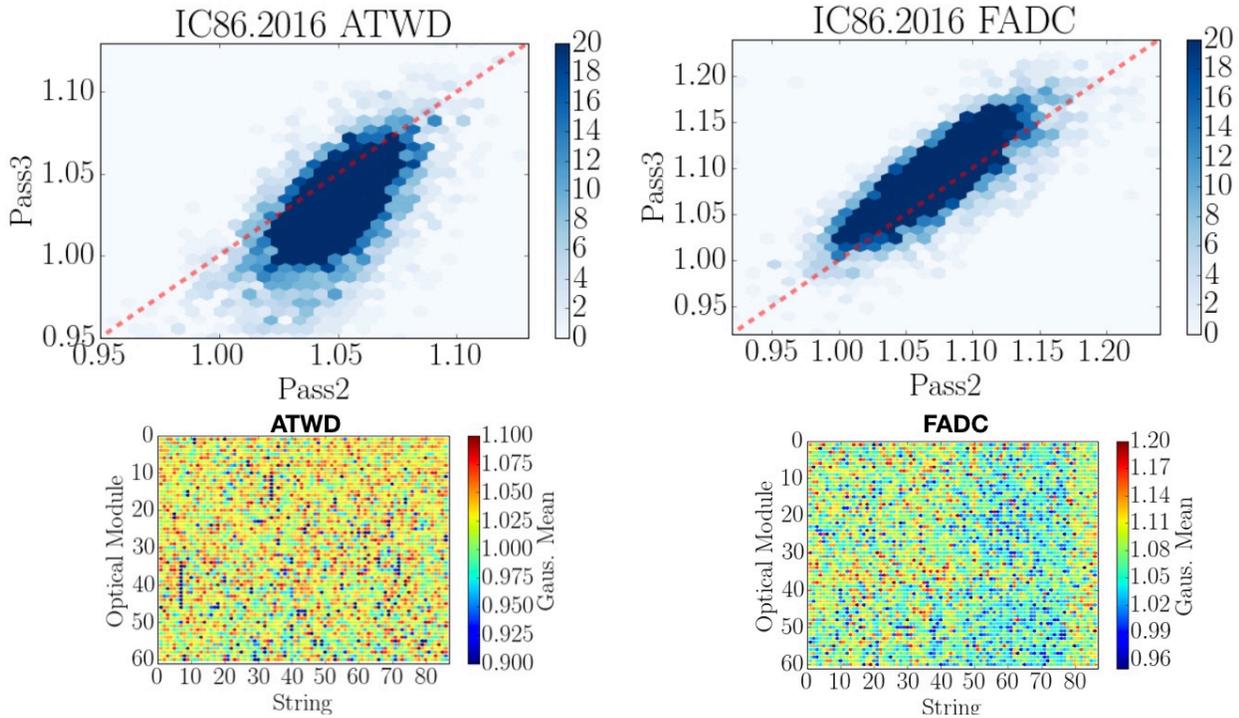
[SPEChargeDistribution ::

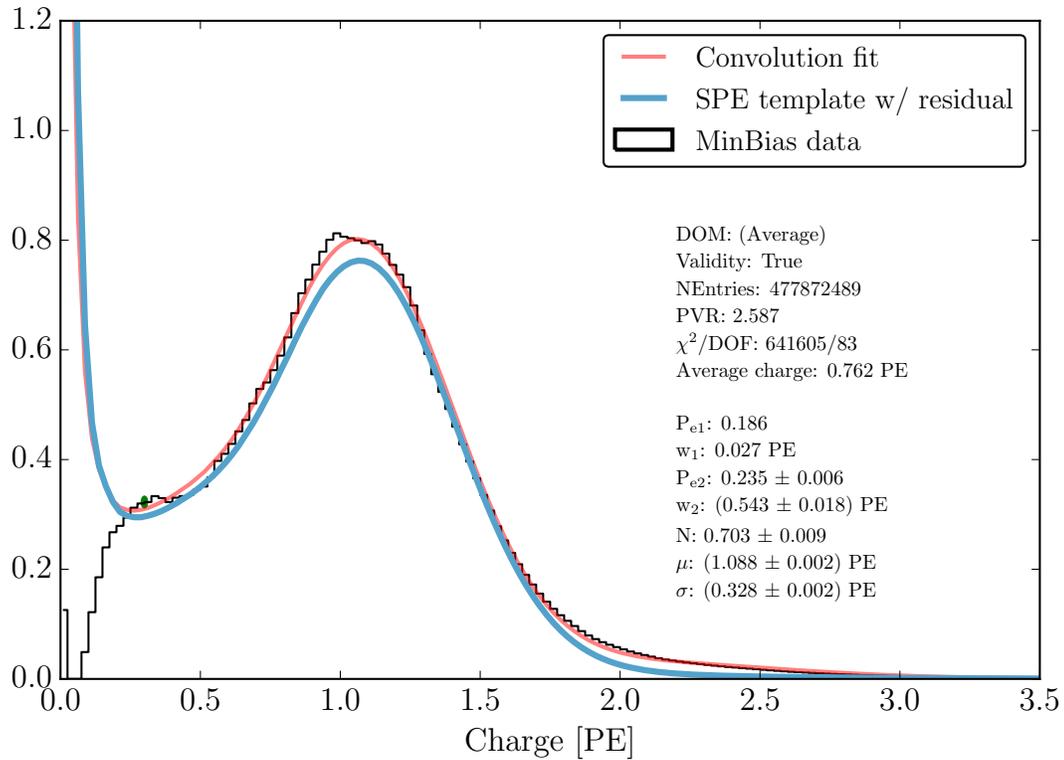
- Exp1 Amp. : 6.9
- Exp1 Width : 0.027
- Exp2 Amp. : 0.598297
- Exp2 Width : 0.394905
- Gaussian Amp. : 0.804938
- Gaussian Mean : 1.0245
- Gaussian Width : 0.286234
- Compensation Factor : 1.26608
- SLC Gaussian Mean : 1.12746

] MeanATWDCharge : 1.04566
 MeanFADCCCharge : 1.11749

Scale me +/- 1% for systematic variation.

- These describe the shape of the charge distributions coming out of the detector.
- Compensation factor is like an RDE. It just scales the DOM efficiency so that IceCubers can use 0.96 as a DOM efficiency.
- These tell you what scaling factor you want to apply.
 - MeanATWDCharge = 1.0 for PASS1*
 - MeanFADCCCharge = 1.0 for PASS1*
 - Gaussian Mean = MeanATWDCharge for PASS3*
 - SLC Gaussian Mean = MeanFADCCCharge for PASS3*
 - Gaussian Mean != MeanATWDCharge for PASS2*
 - SLC Gaussian Mean != MeanFADCCCharge for PASS2*





The following plots show various extracted properties as a function of PMT id. The IDs are separated into an identifier (like TA, MA, AA...) and number (1,2,3, ...)

We can sort the groups of PMT identifiers by their number to look for correlations. Assuming that the ID number is sequentially assigned to the PMT during manufacturing, observed changes in the following extracted quantities over the ID number are likely due to manufacturing. First of, the location of the ID identifiers are:

The majority of the standard QE DOMs have a TA identifier. The MA corresponds primarily to the HQE DOMs. The vertical lines in the plots below indicate DOMs which are instrumented with the old toroids.

