Simulation Processor How-To

Michael Brown

March 21, 2016

What's needed for running:

1. **Set of Trigger Functions**
2. **Set of values for nPE/keV used for simulating width of peaks**
3. **Position Map (currently in position_maps directory and called from there)**

The trigger functions are part of the git repository and the one for Xe period -1 pertains to a chunk of Xe runs from 2010. I've put a conditional statement which chooses an appropriate trigger function for each of the geometries depending on what is passed to the SimulationProcessor.

The nominal set of values for nPE/keV are 1e-1 for each PMT and they are in the nPE_keV directory. These can be changed in this file.

The default position map to be used is from 2011/2012, but they don't look terribly different for each geometry. This map is in the position_maps directory, and it is read in via a function in the posMapReader.h code.

Simulations

We want the name of every simulation output .root file to be of the same format. For the time being, the code is expecting 1 file called xuan_analyzed.root. The distinction between each type of simulation lies in the location of the file.

There should be a directory dedicated solely to holding your simulations somewhere outside the git repo, or at least not tracked. You need to set the path to this directory in your environment variables to CALTECH_SIMS. So if you're using bash, you would have

export CALTECH_SIMS="path/to/your/sims/"

Within this directory there should be a directory called **2010/, 2011-2012/, 2012-2013/.** Within each of these, there should be: **Sn113/, Ce137/, Bi207/, and Beta/.** Furthermore, you should have two directories within Beta/, namely **base/, and fierz/.** The base/ directory holds simulations with no fierz term in the event generator, and the fierz/ directory contains simulations coming from events with a fierz term introduced.

Using Simulation Processor on its own

First you need to run "make" in the little_b/ directory to compile the code. At this point you can type ./SimulationProcessor and the usage command will be printed to screen.

The processor is looking for the user to provide:

- **source** (Sn113, Ce137, Bi207, Beta, Beta_fierz)
- **geometry** (2010, 2011/2012, 2012/2013)
- **numEvents** (Number of events to be run through the processor)
- **boolean doLinCorr** (Tells the code whether or not you will be supplying linearity corrections)

If doLinCorr is true:

- **P0 -** constant offset
- **P1 -** correction to the linear term
- **P2 -** correction to the quadratic term
- **P3 -** correction to the cubic term

There is a hidden last parameter that can be passed telling the processor the index of the parameter set that was passed to it. This is really only applicable when running from the python script, since the parameters are read in from a file.

The general process that occurs within SimulationProcessor.cpp is a source and geometry are supplied with a number of events and the twiddle parameters. The processor then loads position maps, trigger functions, and the nPE/keV values pertaining to the geometry chosen. The events are read in from the proper location depending on the source and geometry. On an event-by-event basis until the requested number of events are reached, the processor takes the simulated event's quenched energy, E_Q, samples a gaussian for each PMT centered on E_Q with a sigma related to the nPE/keV (and with the position dependence folded in) for that PMT to effectively smear the E_Q spectrum, and then applies the linearity twiddle to create an $E_Q^{'}$. The trigger threshold function is then applied for each detector to deem whether or not we have a trigger for this event. At this point, event type reconstruction occurs and the proper histograms or tree is filled (depending on whether it is a source or a beta simulation). If we have a source run, the peaks are fit and checked against the error envelope for the appropriate geometry which is stored in SimulationProcessor.hh.

Using SimulationProcessorManager.py

By running "./SimulationProcessorManager.py --help" you can see all the options which can be passed to the script. I'll explain below what each option does.

- **--makeLinCorrFile** - This will generate a file with parameters spanning whatever space is hardcoded in the makeLinearityParamFile function. The pertinent information for changing the parameter ranges can be found in paramDeltaRanges and paramNSteps.
- **--runSourcesBaseline** - This will run all the sources (Sn, Ce, Bi) through SimulationProcessor with no linearity twiddles
- **--runSourcesLinCorr -** Runs each source through the processor for every parameter set that is contained in parameters.dat (which can be generated using --makeLinCorrFile). This will produce a file which holds all the "good" parameter sets for that source simulation.
- **--makeGoodParamFile** - This cross-checks the files made for each of the sources to create a master list of good parameters. It will not overwrite an old parameter file, as they will just be numbered sequentially if one already exists.
- --**runBetaBaseline -** Runs both the fierz and non-fierz simulations through the processor without any linearity twiddles
- --**runBetaLinCorr -** Runs both the fierz and non-fierz simulations through the processor for all of the parameter combinations which are contained in the "good" parameter file. You can designate the parameter file with one of the following flags.
- **--nEvents=NEVENTS** where this is the number of events you want run through the processor for whatever flags have been given.
- --**geometry=GEOMETRY** where this is 2010, 2011/2012, 2012/2013
- --**betaParamFile=fileNum** where this is the number of the parameter file you want the parameters taken from for applying the twiddles to the beta decay simulations.