

Broad structure of the code:

### Part#1: Generating correlations $S_{mn}$

class treeProcessor →

- takes care of opening the .root file containing the eventbuilt tree and drawing things from it. Also maintains correlation histograms and saves them.
- everything SPS+SABRE related happens here, gainmatching, identifying channels, all else
- in the scheme of analysis, an object of this class is asked to do the following:
  - open the ROOT file corresponding to the right run number
  - be ready to look through all its 'events' in coincidence when asked: we only care about coincident MMM ring+wedge for a single MMM petal of SABRE('board' from here on, numbered 0 to 4)
  - have ready the channelmap, have ready the ability to zero in on a given SABRE detector's data when asked for it
  - jump through the data when asked for by a LOOP (more on that below) and report all events that have a particular (ring,wedge) pair fire with non zero energy in both, and single multiplicity
  - keep filling the ring, wedge energies in a scatterplot named 'scatter' as you go along

THE LOOP:

- lives in main.cpp
- Asks the treeProcessor to jump through the eventbuilder output as described above, for a particular board's particular (Ring,Wedge) pair
- For each coincident, 'good' (ring,wedge) event found by the treeProcessor, send the ring, wedge energies ( $A_m$ ,  $A_n$ ) to the likelihoodNet.

class likelihoodNet

- The likelihoodNet class keeps an array lkGrid. The x position of each point in lkGrid (the array index) corresponds to a particular value of the slope  $S_{mn}$ . What's stored in each array position is the likelihood that that slope is the real E correlation between the (ring,wedge) pair at hand
- The lkGrid, hence, is the probability distribution for each likelihood in the range chosen. We start out with the prior that all likelihoods in the range  $S_{min}$ ,  $S_{max}$  are equally likely - and see how as we step through data that assumption evolves. Eventually, we expect to have a peaked distribution that gives us a maximum, and a measure of variance.
- NOTE: likelihood based calculations are meaningful even when data is sparse. The paper assumes the slope to have the same distribution as that of a random variable that's the ratio of two gaussian distributed random variables  $A_m$  and  $A_n$  - the Cauchy-Lorentz distribution.
- Robust linear fits to data are also obtainable by using the option ROB=0.x in ROOT fits. We do both of these here.

$S = \text{getGridXAt}(i);$

`lkGrid[i] *= (1.0/(w*w+log(backE/(frontE*S))*log(backE/(frontE*S))));`

- The above two lines perform the bayesian propagation described in eqs 7.8,9 in Reese(2015). We then normalize the distribution as we step forward (divide every lkGrid element by 1/sum of the grid)
- After we've stepped through every instance where there's energies in the (ring, wedge) pair chosen - likelihoodNet can then be integrated over by summing the normalized distribution. When it crosses the 16% and 84% threshold, we get the uncertainties about the median. 50% point is the median - we avoid mean in the interest of robustness. Between 16 and 84 percentile lies 68% of the data - which gives us our  $S_{mn}$  with error.
- The executable here is called 'findCorrelations' and takes as argument \$runID \$inputfile \$r \$w
- ./correlateAllMMMs.sh writes the output the executable to text files titled **outlist.0, outlist.1 ... outlist.4** that live in the directory **runXXX/det0, runXXX/det1** etc respectively
- These are formatted clearly enough so as to be readable, it only makes the program in #3 sweat a little extra to clean up the formatting and extract the slope and error from it.
- Lastly, the output from doing all these correlations is also written as png files to the folder that looks like **runXXX/det[0-4]/test\_detector[0-4]\_rng\$r\_wdg\$w.png** - this way one can verify that the slopes have reasonable agreement with correlations seen.

## Part#2: Find the global scaling factors we will need, from Am-241 data

- What's this?
- When we perform gainmatching this way, we ultimately get all factors  $s_n$  such that we can convert ADC readout  $A_n$  on channel  $n$  to energy  $E$  as
 
$$E = s_n A_n$$
- Thing to note here is that all  $s_n$ 's could be scaled by the same factor  $\alpha$  and nothing would change - the calibration still holds.
- It helps us out quite a bit if we set this  $\alpha$  such that  $1\text{keV} = 1\text{channel}$  holds for SABRE data. In this step, we basically scour some raw ROOT data stored with Am-241 peaks in it, that can help us find this scale factors.
- calibrator.cpp has the main() function inside it, since it's so trivial. I decided to write c++ for this just so the entire project is consistent
- Basically, we read a tree, identify the back-most SABRE rings, and scale the E peak we see such that it falls on 5486 channels. We get 5 factors out of running this program, that are stored in ./etc/global\_gain\_scalefactors.dat
- The executable is called 'calibrator' and takes just one argument, the raw ROOT file with Am-241 data

Part#3: Use the two parts above, and generate a simple gainmatch file that is ready to use

- Basically, at this stage, we have all the  $S_{mn}$  values ready from part#1. We perform the chi2 minimization described in eq 5 and 6 in Reese(2015) now.
- Gives us a series of  $s_m$  values for all fronts, backs, all detector segments we've used in #1 – up to a multiplicative constant. We can scale up each board's  $s_n$  values by its own ' $\alpha$ ' by reading from the output txt file in #2 above
- All of this action is in src/fitter/fitter.cpp. Another cpp file which already has a main() in it, since the use-case is so simple. The executable is called 'findGainMatchFactors' which takes as argument the run# and the board#
- The shell script which wraps it, basically calls it for all 5 detectors/boards.
- The neat trick is that it reads outlist.0, outlist.1 ... outlist.4 – as well as the global\_gain\_scalefactors.dat to then create  $s_n$  such that  $1ch = 1keV$  holds for all 5 detectors.
- The final output is temporarily written to a textfile called "outputFile". But at the end of the analysis, it gets moved to the folder runXXX with all other data as gainmatch.runXXX – a simple text file with two columns, ready for use with GWM\_EventBuilder