ANALYSIS NOTE OF Q-FACTORS

Lawrence Ng
Department of Physics
Florida State University
Tallhassee, FL 32304
ln16@my.fsu.edu

ABSTRACT

The Q-factors technique is a method to separate signal and background by weighing each event by a quality factor. This quality factor is the probability that an event originated from the signal distribution and can be used as an event weight. This technique was first introduced in *Multivariate side-band subtraction using probabilistic event weights* [Williams et al., 2009]. An implementation of this technique is developed for the analysis of the reaction $\gamma p \to \eta\pi^0 p \to 4\gamma p$ at GlueX.

1 Introduction

It is often the case that some signal of interest sits on top of some background. Sideband subtraction can be used to statistically subtract out the background contribution underneath the signal but assumes the kinematics of the background regions are identical to that of the signal region. Also, yields in the sideband regions shrink for higher dimensional sidebands which enlarge statistical uncertainties.

The Q-factors technique is an alternative approach to do sideband subtraction. The phase space of a given reaction is labeled by some set of coordinates. A subset of these coordinates are chosen as reference coordinates where the signal and background distribution is known. In the space defined by the non-reference coordinates, k nearest neighbors are determined for each event. An unbinned maximum likelihood (MLE) fit is performed on the reference coordinates of the k nearest neighbors. The Q-factor is defined to be the signal fraction/probability from the resulting fit. This value becomes the weight assigned to a given event and can be used in subsequent analyses such as partial wave analysis (PWA).

2 Implementation

Q-factors is a computationally expensive technique requiring one to search for nearest neighbors and performing an unbinned MLE fit for each event. The current implementation is written in C++ with a python driver script. Unbinned MLE fits are done in RooFit which is currently not thread-safe. This requires one to spawn multiple processes instead which trades memory usage for increased speed. Separate ROOT processes are not entirely independent and will not run at full CPU usage unless compiled. Nearest neighbors are determined by using a priority queue with maximum size equal to k. For significantly better performance BatchMode can be used in RooFit's `fitTo` command but requires ROOT v6.23 or greater.

The `main` program does most the heavy lifting which can be largely ignored by the user. All important features are pulled into the `run.py` which acts as the driver script. The probability density functions (PDF) which are used to fit the data must be set in `configPDFs.h` along with some functions to do stuff like filling the data-set, re-initializing the fit, and fitting. `main` pulls some extra code from some other files which are located in the auxiliary folder. These extra codes allow us to standardize the inputs for distance calculation, sort our neighbors (`helperFuncs.h`), and plot the individual fit results (`drawPlot` program). The results of the `drawPlot` program is saved into a ROOT file where the TCanvas is saved of the plotted fit results. There files are located in the `histograms` folder. `convertROOTtoPNG.C` can be used to convert all the root files to another file format, i.e. png or pdf. This was done to allow future compatibility with

multi-threading. An extra set of codes is used to make the final weighted histograms which are set in the `makePlots` programs. Figure. 1 details the general structure of the code and their goals.

The vast majority of hyper-parameters are set at the top of `run.py`. Usage of these flags are detailed at the top of the script. These flags determine basic operations like how many neighbors we should choose, how many processes to spawn, and what is the input ROOT file to be analyzed. The standard flags should be changed to match your input. Advanced flags can be left as is.

Standard flags:

- **rooFileLocs**: What file we will analyze and what tree to look for. Also need a tag to save the data to so that we don't overwrite other runs
- **accWeight**: the branch to look at to get the accidental subtraction weights. empty string to not use accidental weights
- **sbWeight**: the branch to look at to get the sideband subtraction weights. Wont be used in q-factors calculation, used only as final comparison
- **varStringBase**: semicolon separated branch names to get phase space variables for distance calcuation
- **discrimVars**: semicolon separated branch names to get discriminating/reference variables
- **nProcess**: how many processes to spawn
- **kDim**: number of neighbors
- **nentries**: how many combos we want to run over. Set to -1 to run over all. This should be much significantly larger than kDim or we might get errors
- **numberEventsToSavePerProcess**: how many event level fit histograms (root files) we want to save. Set to -1 to save all histograms

Advanced flags:

- **standardizationType**: {range,std} what type of standardization to apply when normalizing the phase space variables for distance calculation
- **redistributeBkgSigFits**: should we do the 3 different fits where there is 100% bkg, 50/50, 100% signal initilizations.
- **nRndRepSubset**: size of the random subset of potential neighbors. If <=0 or > nentries then we will not consider random subsets
- **doKRandomNeighbors**: should we use k random neighbors as a test instead of doing k nearest neighbors?
- **nBS**: number of times we should bootstrap the set of neighbors to calculate q-factors with. Used to extract an error on the q-factors. 0 = no bootstrapping
- **runTag**: 3 folders are outputs of this set of programs fitResults/diagnosticPlots/histograms. Append a runTag to the names - i.e. fitResults_newTag
- **seedShift**: in case we dont want to save the same q-value histogram we can choose another random seed
- **saveBShistsAlso**: should we save every bootstrapped histogram also?
- **alwaysSaveTheseEvents**: A histogram of the fit will always be saved for these semicolon separated events listed in a string, i.e. "1;32"
- **saveBranchOfNeighbors**: Should we save a branch containing all the neighbors per event. Size increase (4Bytes per int)*kDim*nentries
- **saveMemUsage**: should we output the memory usage into the logs file?
- **saveEventLevelProcessSpeed**: include info on process speed into processLogX.log files
- **emailWhenFinished**: we can send an email when the code is finished, no email sent if empty string
- **runBatch**: Not ready - (default=0) 0=run on a single computer, 1=submit to condor for batch process

The user must set `configPDFs.h` to their requirements. Templates for 1D and 2D PDFs are in `auxilliary/pdfTemplates/`. It is here where the PDFs are defined, how they will be initialized, how to extract the q-factor, how to insert data into the data-set to be fitted, how to get the fit parameters, how to draw the individual fits, and how to perform the fit. Including all the functionality here in the fitManager class allows one to totally ignore the operations of the `main` program.

After executing `main` there will new ROOT files in the `logs` folder. These ROOT files will contain only the information of the extracted Q-factor among some other quality measures. To merge these results into the original input tree `mergeQresults.C` is used producing a final ROOT file/tree containing the full analyzed results.

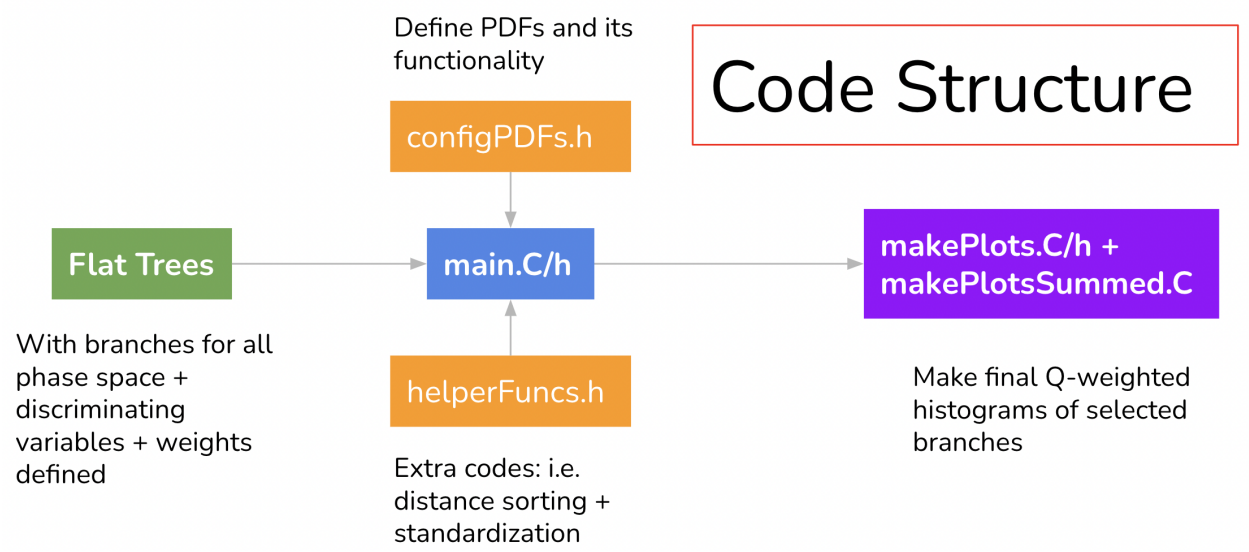

Figure 1: Code structure. The `main` program creates a `logs` folder and a `histograms`. The `logs` folder contains log outputs and also the results ROOT file. The `histograms` folder contains all the event level fits. `makePlots` program dumps all its diagnostic plots into `diagnosticPlots` folder. There diagnostic plots are created from select branches with weighting.

The data can then be read into the `makePlots` set of programs to draw the final histograms. It is recommended to include the name of the sideband subtraction weights into **sbWeight** flag of `run.py`. This allows one to compare the results of the Q-factors to that of the sideband subtraction technique. To create a set of weighted histograms is quite simple. The following code will load the "Meta" and "Mpi0" branches and make the following Q-value weighted + sideband subtracted histograms: the 1D Meta histogram and the 2D Mpi0 vs Meta plot. These "diagnostic" plots are located in the `diagnosticPlots` folder.

```
vector<string> histsToMake={"Meta","Mpi0;Meta"};
```

2.1 Accidental Subtraction

An additional branch can be named in `run.py` to include weights to be used during the likelihood fitting step for an event. This allows one to accidentally subtract the nearest neighbors' reference distribution to make the extracted Q-factor weight independent of the accidental subtraction weight such that we are able to multiply the two weights together in the end. Accidental subtraction removes the contribution of multiple in-time tagged beam photons. Without accidentally subtracting the fit distributions the resulting Q-factor would have knowledge of the accidental beam photons and thus sort of "double" count their contribution.

2.2 Future considerations

The original authors have provided a way to extract statistical uncertainties. Alternatively, one might use a technique like bootstrapping to resample the data and extract error bars. Both are very computationally expensive techniques but the implementation of bootstrapping is quite straightforward. Another feature to consider is that since the processes are already independent it is quite simple to write a batch submission script for the HTCondor system. Both these features are implemented but have not been studied in detail.

3 Case study: $\gamma p \to \eta\pi^0 p \to 4\gamma p$

Mapping the light meson spectrum is important in understanding the confinement of quarks and gluons inside hadrons. A particularly interesting system is that of the hybrid mesons which is a meson system with explicit gluonic degrees of freedom. Lattice QCD predicts their existence with the lightest exotic hybrid, dubbed the π_1, having a mass under 2

GeV. Recent analyses like the coupled channel analysis of $\eta\pi$ and $\eta'\pi$ COMPASS data by the Joint Physics Analysis Center (JPAC) have determined the mass of the π_1 to be around 1.6 GeV.

The primary goal of GlueX is to map the light meson spectrum with a focus on exotic hybrids. Measuring the properties of the π_1 is an important first step in future searches. Observation of the π_1 in photoproduction is also important as previous experiments mainly used a pion beam.

The reaction of interest is $\gamma p \to \eta\pi^0 p \to 4\gamma p$. Simply by chance, multiple combinations of the final state photons can pass all event selections. This type of background should be mostly flat in the 2D distribution of $M(\eta)$ vs $M(\pi)$. Other processes like $\gamma p \to \omega p \to 3\gamma p$ and $\gamma p \to b_1\ p \to \omega\pi^0 p \to 5\gamma p$ can also leak in where one photon is gained and another is lost respectively.

References

M Williams, M Bellis, and C A Meyer. Multivariate side-band subtraction using probabilistic event weights. *Journal of Instrumentation*, 4(10):P10003–P10003, Oct 2009. ISSN 1748-0221. doi:10.1088/1748-0221/4/10/p10003. URL `http://dx.doi.org/10.1088/1748-0221/4/10/P10003`.