INTT Weekly Meeting

Joseph Bertaux

Purdue University

June 19, 2024

Overview

- OnlMon progress
  - Martin implemented a new API handles to retrieve all BCOs from a Packet
  - Hitmap and hitrates options are now properly normalized
  - Changes were applied yesterday, but we haven't had beam to check
- Revisting hot channel classification
  - Using survey geometry to normalize by positions
  - Identify cold channel cutoff with local minimum in hitrate PDF
  - Identify hot channel cutoff when the tail reaches the aforementioned local minimum
- QA
  - Plots produced from calibration modules can be used for QA database
  - Calibration modules themselves can have QA functionality
  - Eventual automation of database modification with scripts and SQL
- HV SQL table
  - In the daq database of opc0
  - Logging MPOD values since April 24

Last time

- Mentioned software change I have been meaning to implement for some time
- Ultimately fell through–Chris thought it was too drastic
- Mostly agree with his decision and will revisit this after physics running (October)

HV Database

- `psql` table in the `daq` database
- accessible on `opc0`
- table is `intt_mpodlog`
  - time `timestamp`
  - ip `varchar(32)`
  - mpod_channel `smallint`
  - status `varchar(12)`
  - voltage `real`
  - current `real`

HV Database (cont'd)

- Updated every 2 minutes due to a cron job periodically run by John Haggerty
- The cron job runs `home/phnxrc/INTT/josephb/HV/pycodes/mpodlog.sh`
  - Wrapper shell script that configures a `venv`
  - runs a python script
  - implementation is similar to the HV GUI
  - values obtained via `snmpwalk`
- High voltage currents/status can be monitored via Grafana
  - Nicety but I think implementing a Grafana monitor is low priority
  - But it could make a nice project for a student who wants to learn Grafana/SQL

Hot/Cold Channel classification

- The fundamental shape is not described by a Gaussian or Poisson
  - Peak is too far left of the median and mean
  - The hot "tail" is too fat
- Fits using a Gaussian or Poisson will fail in several ways:
  - The mean will not correspond to the peak location, but closer to the mean
  - The variance will correspond to the IQR, and fail to predict the slow decay in the tail
  - Underestimating the variance leads to masking a higher portion of the detector
  - The fit can fail to converge normally and is subject to histogram binning

Hot/Cold Channel classification (cont'd)

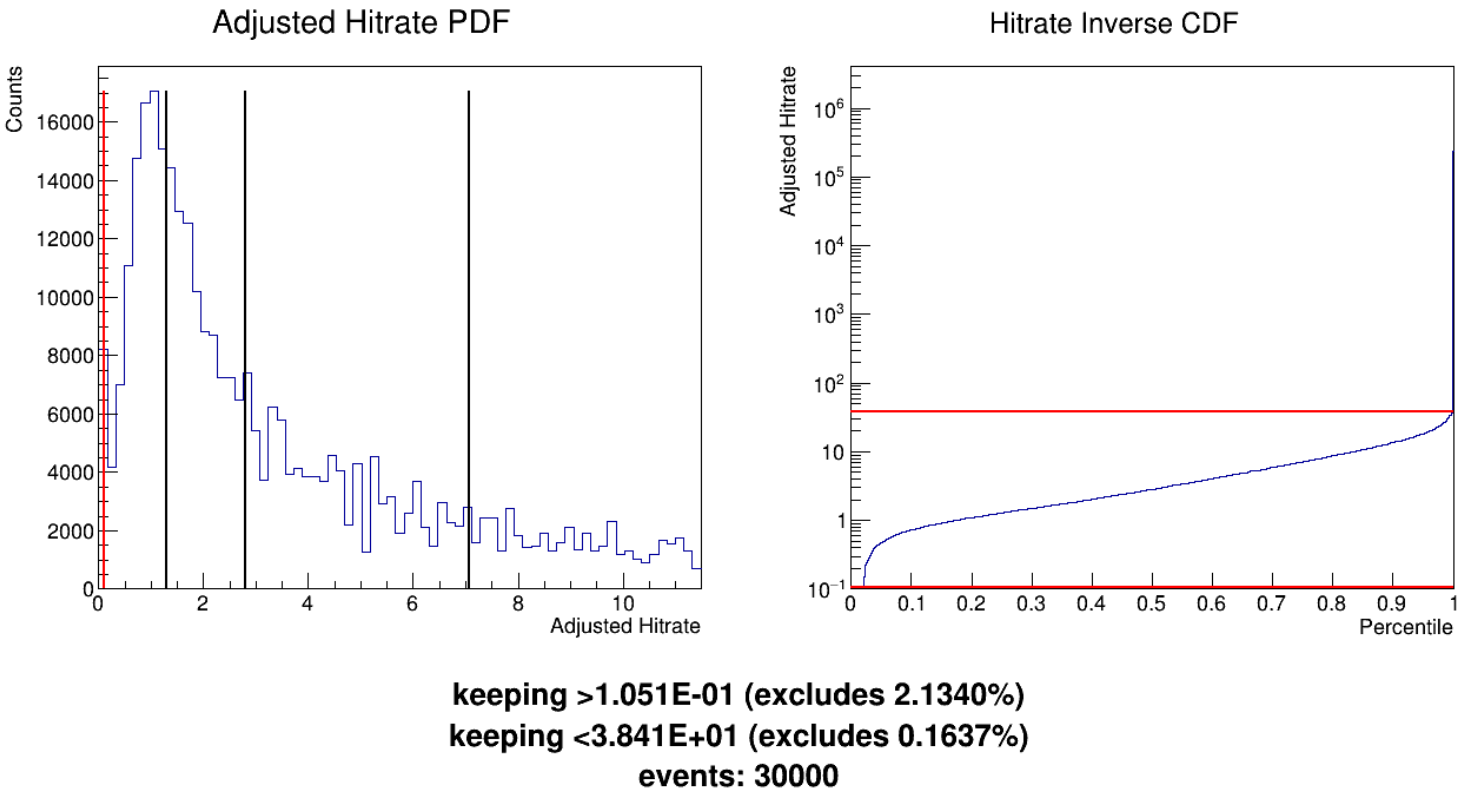


Figure: Quartiles are drawn as vertical black lines on the hitrate pdf. The peak lies outside the IQR; Gaussian fits find the mean close to the medain value, with sigma commensurate with the IQR.

Hitrate Adjustment

- Hitrate normalization:

$$h' = \frac{h \left|\left|\vec{r}\right|\right|^2}{L\left(\hat{n} \cdot \hat{r}\right)} \tag{1}$$

  - h' is the adjusted hitrate
  - h is the raw hitrate (hits per event)
  - $\vec{r}$ is the displacement vector from an origin to the strip position
  - $\hat{n}$ is the sensor unit normal
  - L is strip length, either 1.6cm for Type A or 2.0cm for Type B
- This can be understood by solving for h
- A given hitrate should be proportional to the flux one expects geometrically
- Note that $\vec{r}$ is obtained using survey geometry assuming a vertex position $\vec{0}$
- All channels can be compared in a common distribution

Algorithm

Get the inverse cdf, starting with a hitmap:

```
// m_hitmap is filled over many process_event calls

// get pdf up to normalization
std::map<double, double> hitrate_pdf;
for(auto const& [channel, count] : m_hitmap) {
        // count is the number of hits seen by channel
        double h = count / m_num_evts;
        double h_adjust = adjust_hitrate(channel, h);
        ++hitrate_pdf[h_adjust];
}

// get inverse cdf
double total = 0;
double NUM_CHANNELS = 8 * 14 * 26 * 128; // number of INTT channels
std::map<double, double> hitrate_inv_cdf;
for(auto const& [hitrate, count] : hitrate_pdf) {
        total += count;
        double fraction = total / NUM_CHANNELS;
        hitrate_inv_cdf[fraction] = hitrate;
}
```

Algorithm (Cont'd)

Find local minima using the inverse cdf

```
double min_hitrate = 0.0, max_hitrate = 0.0; // Our cold/hot cut values
double best_min_ration = 0.0, best_max_ratio = 0.0;
double prev_fraction = 0.0, prev_hitrate = 0.0;
for(auto const& [fraction, hitrate] : hitrate_inv_cdf) {
        // Division is safe since the smallest fraction is >0
        double ratio = (hitrate - prev_hitrate) / (fraction - prev_fration);

        // search for local min left of median
        if(best_min_ratio < ratio && fraction < 0.5) {
                best_min_ratio = ratio;
                min_hitrate = hitrate;
        }

        // search for max right of median
        if(best_max_ratio < ratio && 0.5 < fraction) {
                best_max_ratio = ratio;
                max_hitrate = hitrate;
        }

        // ratio can become large as the tail becomes sparse
        // stop if we've found a best_max_ratio as large as best_min_ratio
        if(best_min_ratio < best_max_ratio) {
                break;
        }
}

// min_hitrate is now a suitable cold cut
// max_hitrate is now a suitable hot cut
```

Algorithm (Cont'd)

- Effectively, we are finding a local minimum in the pdf
- We do this by finding maximum values of the slope of the inverse cdf
- Since the pdf tail can become arbitrarily sparse, we match the value found on the left
- The algorithm is very stable and can be used for *pp* and *AuAu* runs without modification
- Pooling of all channels increases statistics, and reduces the number of events needed for convergence
  - *pp* converge in about 30k events or less
  - *AuAu* can converge in as few as 1k events
- Typically
  - 3%-4% of the detector is cold
  - 0.1%-0.3% of the detector is hot
  - but this is just the trend over the runs I analyzed while designing and debugging
- Examples follow on subsequent slides

Run 41021

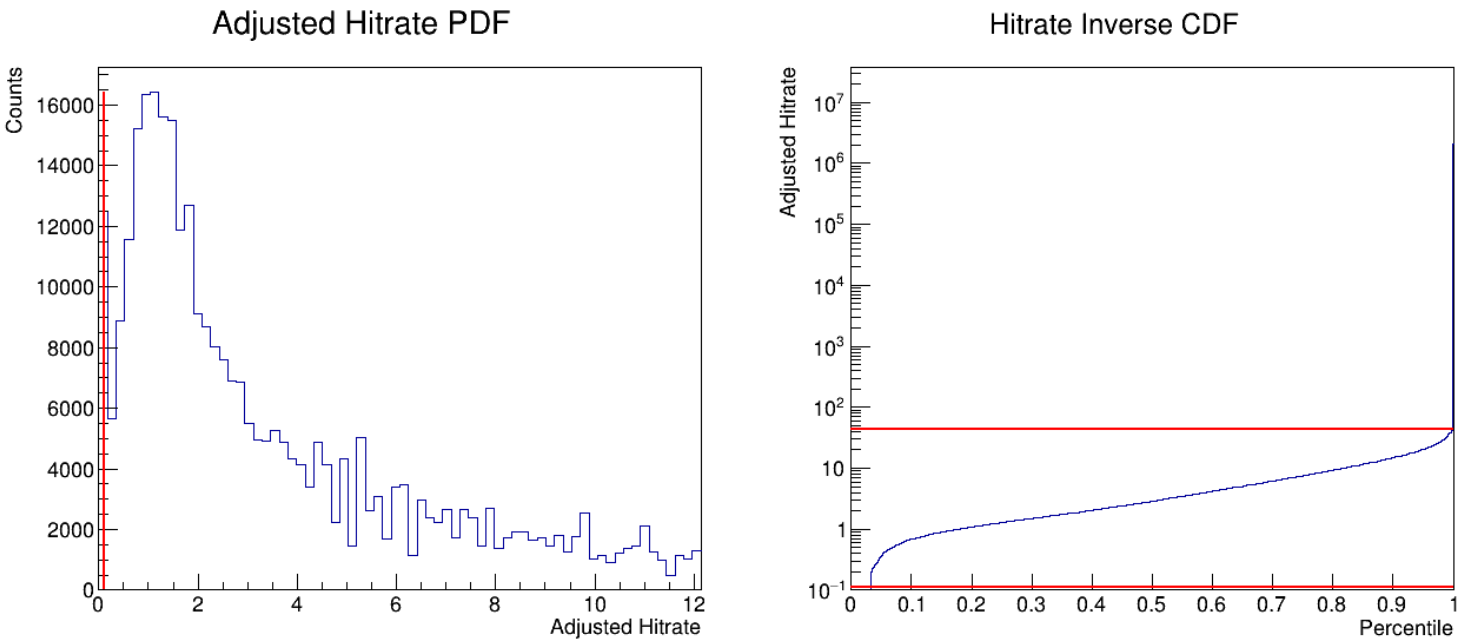
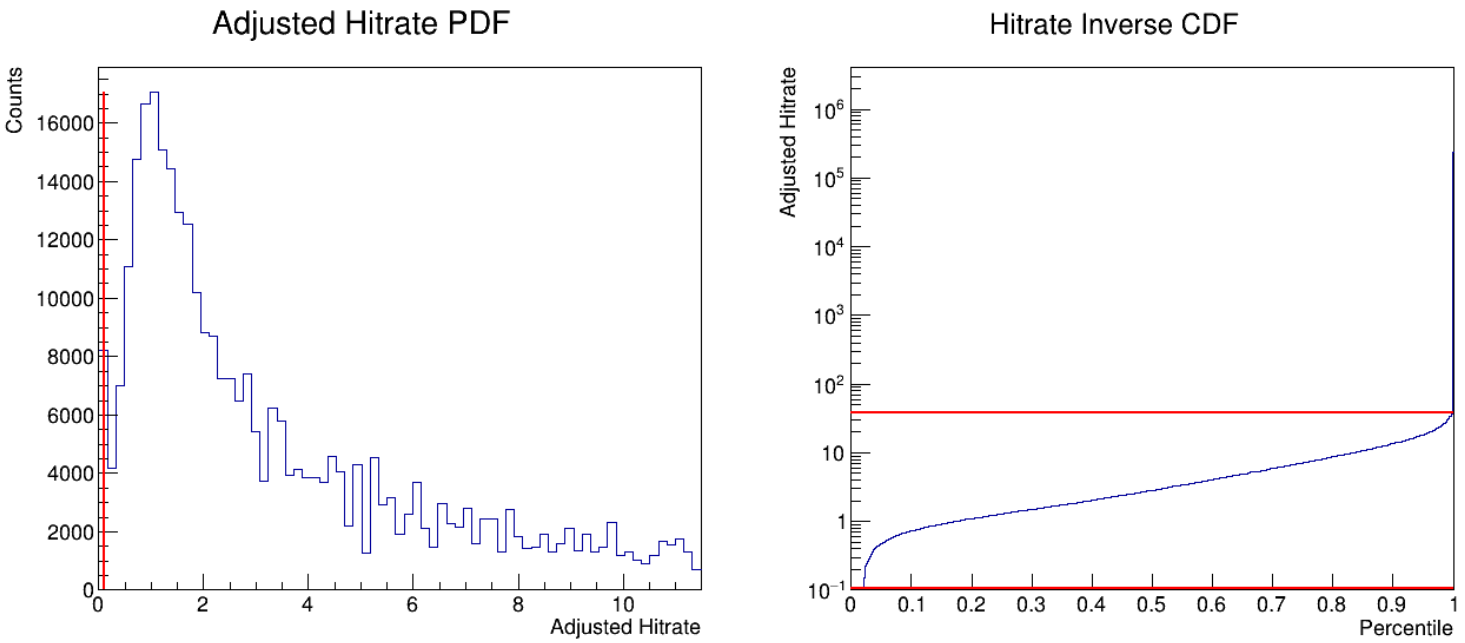


Figure: Run 41021 (pp)

Run 42212



keeping >1.051E-01 (excludes 2.1340%)
keeping <3.841E+01 (excludes 0.1637%)
events: 30000

Figure: Run 42212 (pp)

Run 20444

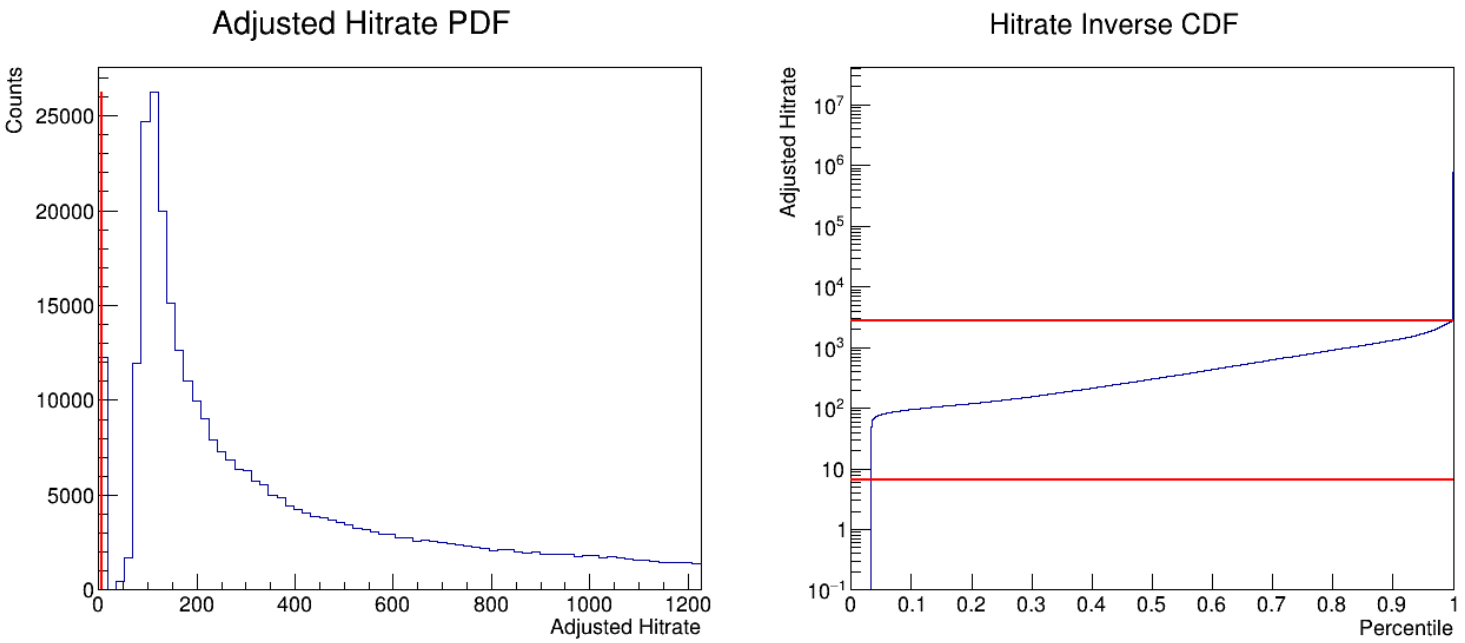
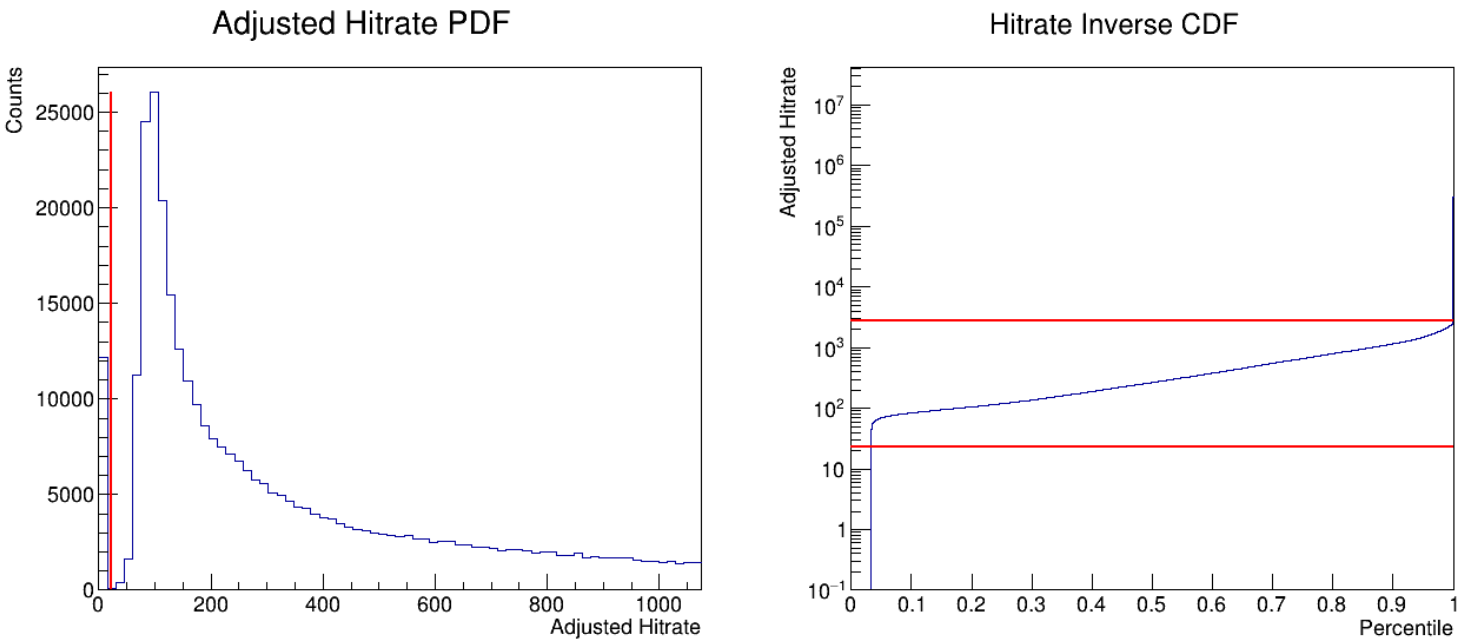


Figure: Run 20444 (*AuAu*)

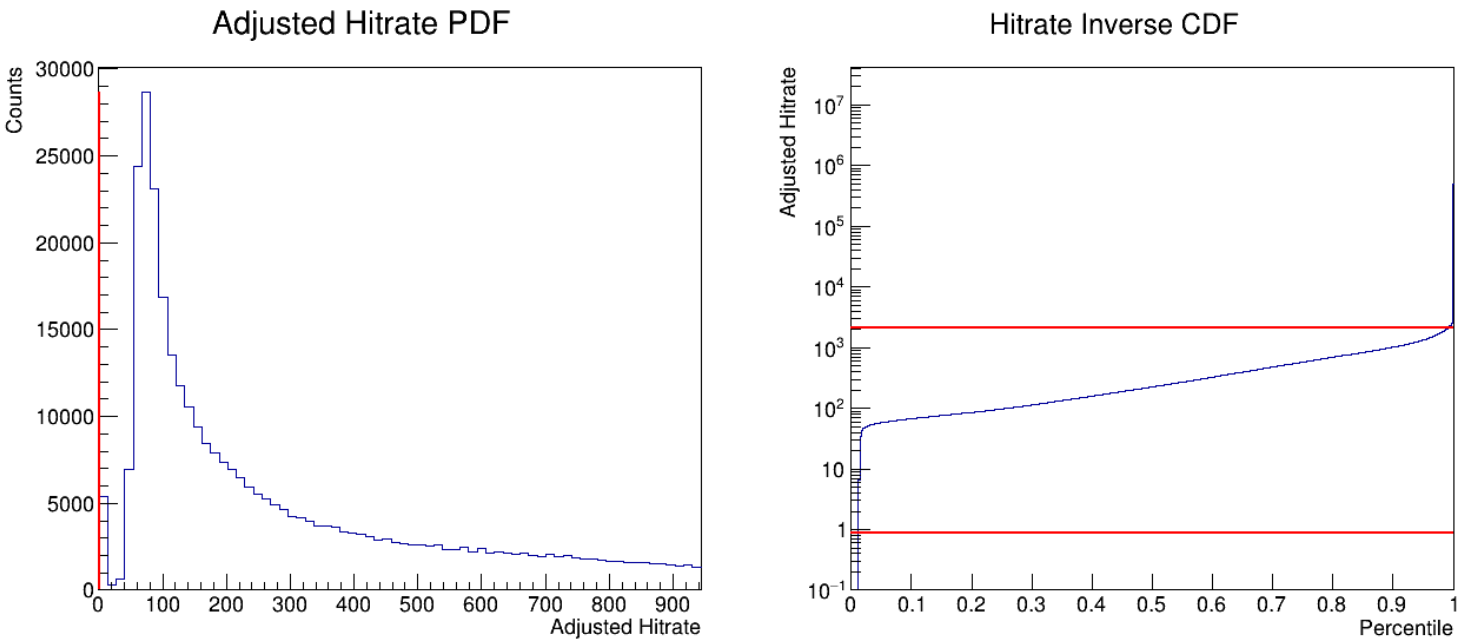
Run 20446



keeping >2.313E+01 (excludes 3.2817%)
keeping <2.699E+03 (excludes 0.0612%)
events: 5000

Figure: Run 20446 (*AuAu*)

Run 20869



keeping >8.771E-01 (excludes 1.3613%)
keeping <2.068E+03 (excludes 0.7968%)
events: 5000

Figure: Run 20869 (*AuAu*)

Natural extension to QA

- The plots for debugging are suitable for QA output
  - The same module/macro can be used
- Chris' students implemented a QA database
  - The natural conclusion of QA should be to issue SQL commands
  - Classify the run as "Good" or "Bad" based on module output
  - Issue SQL statements upon completion
- For the hitmap, an easy metric is the total portion of the detector that is excluded
  - For example, $> 5\%$ total excluded is "Bad"
  - Otherwise, it is "Good" unless other modules detect problems

Future Work

- Implement QA modules for other things
  - BCO timing is another natural choice
  - Genki has many such modules, but they are not committed to coresoftware or macros (that I know of)
  - For each QA module, there should be a concrete conclusion of run goodness
- Continue to work on Online Monitoring
- Misc. INTT work as it arises