

BASIC GOALS

- modular, object-oriented framework (no monolithic files)
- ROOT-based
- reasonably fast (pretty much should be IO-bound)
- do as much “offline” analysis as can be done in real-time
- make all cross-checks we can think of
- provide a GUI for users which is easy to maintain (ROOT plots, C++)

class `TreePlanter` reads in raw files and writes out ROOT files

ROOT file contains TTree's of class (not struct!) `HjetEvent`

Can make cuts on data to be written to the tree

I propose making some ROOT files which have some loose cuts (basically within the banana). This makes later analysis MUCH faster since most data is background. One can always run with no cuts as well (raw data never gets deleted, of course).

Similar to what PHENIX does:

class **FileRunner** takes care of all opening and scanning of ROOT files. It interleaves data analysis and file reading with multiple threads. This is transparent to the user.

The user writes subclasses of the class **BatchModule**, and registers them with the **FileRunner**. The data files can also be passed over multiple times in a single run. Multiple files can be run over at a time, but the modules will be told when they are moving to a new file.

User only has to implement this class

```
class BatchModule
{
public:
    BatchModule(){};
    virtual ~BatchModule(){};

    //called at beginning of run
    virtual int Init() = 0;
    //called at beginning of a new file
    virtual int InitFile(const char* fname) = 0;
    //called when a file is closed
    virtual int EndFile() = 0;
    //called for every event
    virtual int processEvent(HjetEvent* event, int strip) = 0;
    //called at end of run
    virtual int End() = 0;
};
```

I wrote a data fitting package for the following reasons:

- provide custom fits like 1-D curve to 2-D data (banana, for instance)
- speed and improvements
 - multi-threaded...this is 2011 and my desktop has 32 cores
 - analytic calculation of gradient and hessian: also improves the result!
- automatic ignoring of outliers in data (hot channels not a problem now)
- it is fun to do!

It is quite general so I put it on google code:

<http://code.google.com/p/fitnewton/>

Together, the 3 modules

ElasticCutter

MissingMassPlotter

Asymm

- Perform banana fit (“T0” cross-check)
- Determine angle of each strip (beam position cross-check)
 - includes correction for magnetic field : I want to compare to the actual field value
- Cut on missing mass squared
- Determine background level from hjet tail (is it polarized?)
- Calculate polarization for blue and yellow beams

SOME TECHNICAL STUFF

JET SVN repository seems to be down: I will call ITD

Should any of this go into the general cnipol repository?

All of my packages use the ubiquitous `pkgconfig` to keep track of library and header paths. So there is only one environment variable to worry about.

I would like to bring at least a plotting GUI to the shifters sometime this run.

The DAQ GUI/code maybe should be put into a more maintainable form, but DAQ code needs to be treated very carefully. Does it need to be maintained, or is it pretty much good forever?

I demonstrate a basic GUI for those locally present. Since the Carbon/HJet have similar plots/cross-checks to show, is it worth making a GUI framework which would work for both, with an API to add plots? Or should there just be a standalone program? The former is more work on my part, but if others want to use such a thing in the future it could be worth it.

The GUIs used now are thousands of lines of perl/tk, all in one file. This is just not maintainable. If we want people in the future to be able to modify the interface that the shifter sees, the code should be made more modular.