Simulating zero suppression (and comments on code design)

DUNE 35-ton Simulation and Reconstruction

David Adams

BNL

November 4, 2015

Introduction

There is the possibility of continuous data taking for 35 ton

- This should allow us to collect a much larger sample of near-horizontal muons
  - Although we will quickly get many muons, I don't think we will ever get enough close to the beam direction
- A similar comment can be made for pi0s

Continuous data taking requires online zero suppression

- What factor do we need? 50?
- Online zero suppression algorithm was described by James Russell at the last collaboration meeting:
  - https://indico.fnal.gov/conferenceOtherViews.py?view=standard&confId=10100
- It has five parameters: two thresholds and three channel counts
- Useful to study the data reduction and affect on performance in MC
- So I have implemented the algorithm in software
  - Class ZeroSuppress35t

ZeroSuppress35t

Zero suppression is in a dedicated repository

- https://github.com/dladams/dunezs
- Class ZeroSuppress35t implements the suppression
  - Method filter(…) zeroes out filtered ticks
- Class ZeroSuppress35tService wraps this up in an art service

How to use this

- Easy to apply this in analysis after reading raw data
  - Replaces values in filtered bins with zero (or any other value)
- Should also be easy to add this to DUNE MC production
  - Need to add a parameter to the raw data producer and insert a line to call the filter if that parameter is set
    - Mods would be only in dunetpc
    - Change can be such that old fcl recovers the old behavior
  - Let me know if I should do this

ZeroSuppress35t.h

```
// ZeroSuppress35t.h
…
class ZeroSuppress35t {
public

  typedef unsigned int Index;
  typedef double Signal;
  typedef std::vector<Signal> SignalArray;

  // Ctor from the five parameters that characterize the algorithm.
  ZeroSuppress35t(Signal tl, Signal td, Index nl, Index nd, Index nt, Signal azero =0);

  // Return the value assigned to suppressed channels.
  Signal zero() const;

  // Apply the ZS filter to an array of signals.
  // Filtered-out signals are replaced with the value zero().
  // Returns nonzero for error.
  int filter(SignalArray& sigs) const;

  // Display the signal parameters.
  std::ostream& print(std::ostream& out =std::cout, std::string prefix ="  ") const;

…
};
```

DUNE code design

This work gave me the opportunity to study some DUNE code

- I make a few comments (could be many more)
- Goal is ***not*** to trigger a massive code rewrite
- But some things to keep in mind when writing new code
  - And adiabatically updating existing components

Is now the time for such a discussion?

- Data from the 35-ton detector is imminent
- But after that we will want to analyze that data, then work on the next design report, then protoDUNE, then …, then FD data taking, …
- Never a good time ➔ always a good time to have some discussion
  - Please give me 5-10 min today
  - After general discussion on goals, I will cover two topics
- Try to agree on goals and policies for new code and updates
  - So that 5-10 years from now we end up with robust code

DUNE code design (2)

Goals for code design

- Facilitate contributions from a wide community
  - For fairness and to ensure the best ideas find their way into our code
  - Make it easy to understand: structure and documentation
    - FCL naming and structure is an important component
    - Brief comments in code headers describing purpose, technique and interface
  - Fine granularity so an individual can contribute in one place without disturbing others
  - Modularity so it is possible to plug in alternative algorithms without disturbing existing code
- Robustness
  - For production running and physics analysis
    - And to protect from contributions from the above wide community
  - Unit and high-level testing are part of this
- Flexibility
  - Make it easy to support new platforms and frameworks
    - E.g. can same code be used for studies in Root on my desktop or laptop

Class granularity and configuration

Limit the scope of any one class

- Only one complex action (e.g. ZS) in a class
  - Call this an action class
  - Make a new class for other actions (e.g. compression) or for alternative implementations (different coding or different approach)
- Class which carries out a sequence of actions does not have the code for each but calls a different object for each action
- Configuration naturally follows the same structure
  - Rather than a long list of parameters, the high-level configuration carries a list of actions and each of those actions has a list of parameters

Class configuration

- Action may be characterized by many configuration parameters
  - I.e. parameters that may change between jobs but are fixed for one job
- Prefer to specify these parameters in the ctor of the action class
  - Rather than pass them as arguments each time action is invoked
  - This makes it easy to plug in a new action configuration

Framework modularity

Code has three levels of framework dependence

- Utilities have no dependence on art or fcl
  - E.g. class with ctor taking parameters and method to carry out action
- Services are configured from fcl
  - May make use of art callbacks but, if not
  - Can easily be used outside of art
    - See https://github.com/dladams/art_extensions
  - Restricted to use a singleton (no instances)
    - I would like to lift this restriction—call these tools
    - Tool could have multiple named instances
- Producers have strict art interface and are only used there

Action class should be a service (or tool)

- So it can be used with its configuration from a producer in art
- And can be used with configuration outside art
- May want to put code in a utility and add a service wrapper to allow use outside art without fcl
  - Facilitates later migration to a different configuration language

Conclusions

Continuous running with 35-ton detector

- Can provide much larger sample of horizontal muons and pi0s
- Requires zero suppression that significantly reduces data rate

ZeroSuppress35t

- Class that simulates planned zero suppression algorithm
- Available in dunezs repository
- Has five parameters
- Plan to use this to optimize parameters for data taking

Code design

- Goal to make it easy for a wide range of DUNE collaborators to use and contribute to the DUNE SW
- Some policies suggested
- dunezs follows these policies