The Flux Reader Package

Gareth Kafka
Harvard University
Beam Working Group
4/3/14

Warning and Disclaimer

- This talk is describing the features of and how to use a coding framework
- It is completely devoid of plots and/or pretty things to look at (I'm sorry about that)
- For those in the room, I have a demo script that I will run at the end of the talk which will contain many plots
- For those out of the room, I point to the location the file will be at the end of the slides, but you will have to open and peruse the file yourselves (again, sorry about that)

Flux Reader

- FluxReader is a macro that reads flux files, and outputs their content in histograms
- Histograms are created per neutrino flavor, decay parent, applied cross section, and detector location
- FluxReader can generate plots for any information in the flux files, and can perform event by event calculations
- External weights can be applied
- All of these options are user configurable

Procedure (Outline?)

- A Parameters object is created, setting up the neutrino flavors, decay parents, cross sections, and detectors
- This object is given to a FluxReader object
- The FluxReader object is given an input file wildcard string and a set of histograms to create
- The FluxReader object is given an output file location and creates the histograms
- Limited post analysis can be performed by a Combiner object, which takes the FluxReader output file as input

Parameters

- Parameters class determines what neutrino flavors, decay parents, cross sections, and detectors to use
- The user can add or remove most of the parameters using the built in Add or Remove functions with the appropriate input
  - No AddNuFlav or RemoveDetector
- The sensitivity to the decay parent sign can be set, but neutrinos and anti-neutrinos are always separate
- The object can no longer be configured after it is passed to a FluxReader

FluxReader

- FluxReader takes a Parameters pointer in its constructor, and is configured by its AddSpectra and InputFiles member functions
- InputFiles takes a string, and two optional arguments
  - The string is a path name for input flux files, and can contain wildcard characters
  - First optional argument is the max number of files to run over
  - Second optional argument is the number of files to skip past

FluxReader

- AddSpectra takes many arguments, notably an axis label, vector of bin edges, Var object, and Weight object
  - Up to 3 sets of the label, edges, and Vars can be given
  - The Var determines both what is plotted on an axis and how it is calculated
  - The Weight determines how to weight each event
  - AddSpectra also takes a string 'title,' which will be common to all histograms of this type
  - A set of external weights is an optional argument
  - Each Spectra can have a different external weight, but the Weight object defines how to apply them

FluxReader

- FluxReader Bins function creates a vector of bin edges from a number of bins, min and max
- Once the input files are spectra are set, the ReadFlux function is called with a TDirectory pointer, which will be the location for the output
- ReadFlux structures the output directory with a folder for each detector, a folder for each type of spectra in each detector folder, and fills these subfolders with the appropriate histograms

Combiner

- A Combiner object takes as its input, the output from FluxReader
- Combiner reads the file and re-extracts the set of Parameters used to create it
- CombineNuFlavs function combines all neutrino flavors into one plot holding decay parent, cross section, and detectors constant, and retaining the original plots
- CombineParents performs similar function
- CombineAll calls the above functions, then creates a plot of all neutrinos for each detector and cross section

Running the Code

- A normal <scriptname>.C file can be created, which will configure and run everything
- The script needs to begin with

```
#ifdef __CINT__
void <scriptname>()
{
  std::cout << "Sorry, you must run in compiled mode."
  << std::endl;
}
#else
```

and end with

```
#endif
```

Running the code

- To run <scriptname.C>, run at the command line:
- …> root runbeammats.C

```
{ // This is the entire body of runbeammats.C
  // This lets root know where to look for files, and it
  // is only printed on multiple lines here for clarity
  gSystem->SetIncludePath(
    "-I$SRT_PRIVATE_CONTEXT/include
     -I$SRT_PUBLIC_CONTEXT/include
     -I$ROOTSYS/include");

  // This library must be loaded before running
  gSystem->Load("libNovaBeamMat.so");

  // Compile the script
  gROOT->ProcessLine(".L <scriptname>.C+");
  <scriptname>(); // Run the script
}
```

dk2nu Transition

- The code will be changed to run over dk2nu files
- It will generally be optimized to run over these files, and this may cause some of the functionality to become obsolete
- This is a goal for the near future

What's Cooking?

- Committing to a general NuMI-X repository
- Create a new Spectra type that can put a different detector on each axis
- Transfer external cross section code to FluxReader, and add target nucleus as a member of the Detector class
- Toying with the option to continue running on old files by specifying a flux file tree name
- Some of the internal FluxReader functions inherit from the old NovaBeamMat; these should be audited to make sure they are performing properly
- Any other suggestions?

Acknowledgement

- The Var and Weight classes are very similar to the CAFAna Var and Cut classes
- The handling of wildcards in the FluxReader::InputFiles function comes directly from CAFAna/Utilities.cxx
- For both of the above points, thanks to Chris Backhouse who developed the CAFAna framework

Demo Script

- I've written a demo script to show case as much functionality as possible
- /nova/app/users/gkafka/test_svn_development/NovaBeamMat/DemoFluxReader.C
- It's output will be in the same folder
- /nova/app/users/gkafka/test_svn_development/NovaBeamMat/demooutput.root