CAFAna Tutorial

Gavin S. Davies

Iowa State University

Introduction

- The **CAFAna** framework provides a collection of classes allowing easy plotting for analyses based on CAF files
- Extends from simple studies to fitting of oscillated spectra
  - Equally applicable to the ν_μ and ν_e analyses

- Chris Backhouse is the main architect of this framework – he and Gareth Kafka put together a tutorial which can be found in doc-db 10127
- Chris also put together a very concise tech note in doc-db 9222
- Also documentation inline in doxygen
  - Lives in the "ana" namespace: http://nusoft.fnal.gov/nova/novasoft/doxygen/html/namespaceana.html
  - Excellent example of good documentation!!

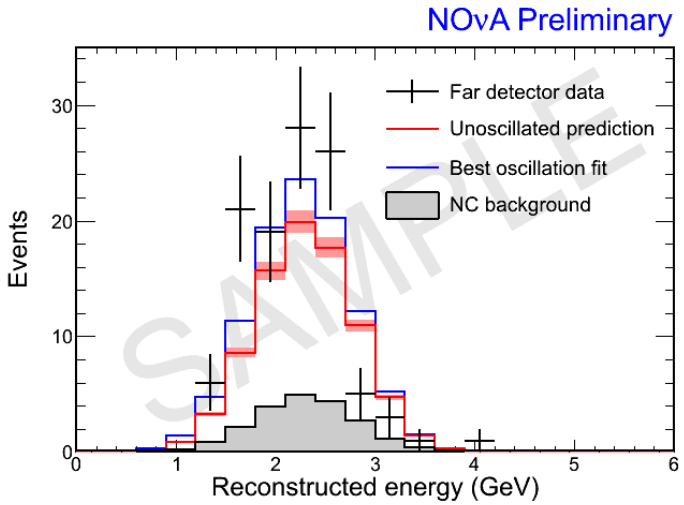
- Will hold your hand through the already existing demo macros (plus one additional one), but certainly won't cover everything.
- The framework really is very powerful

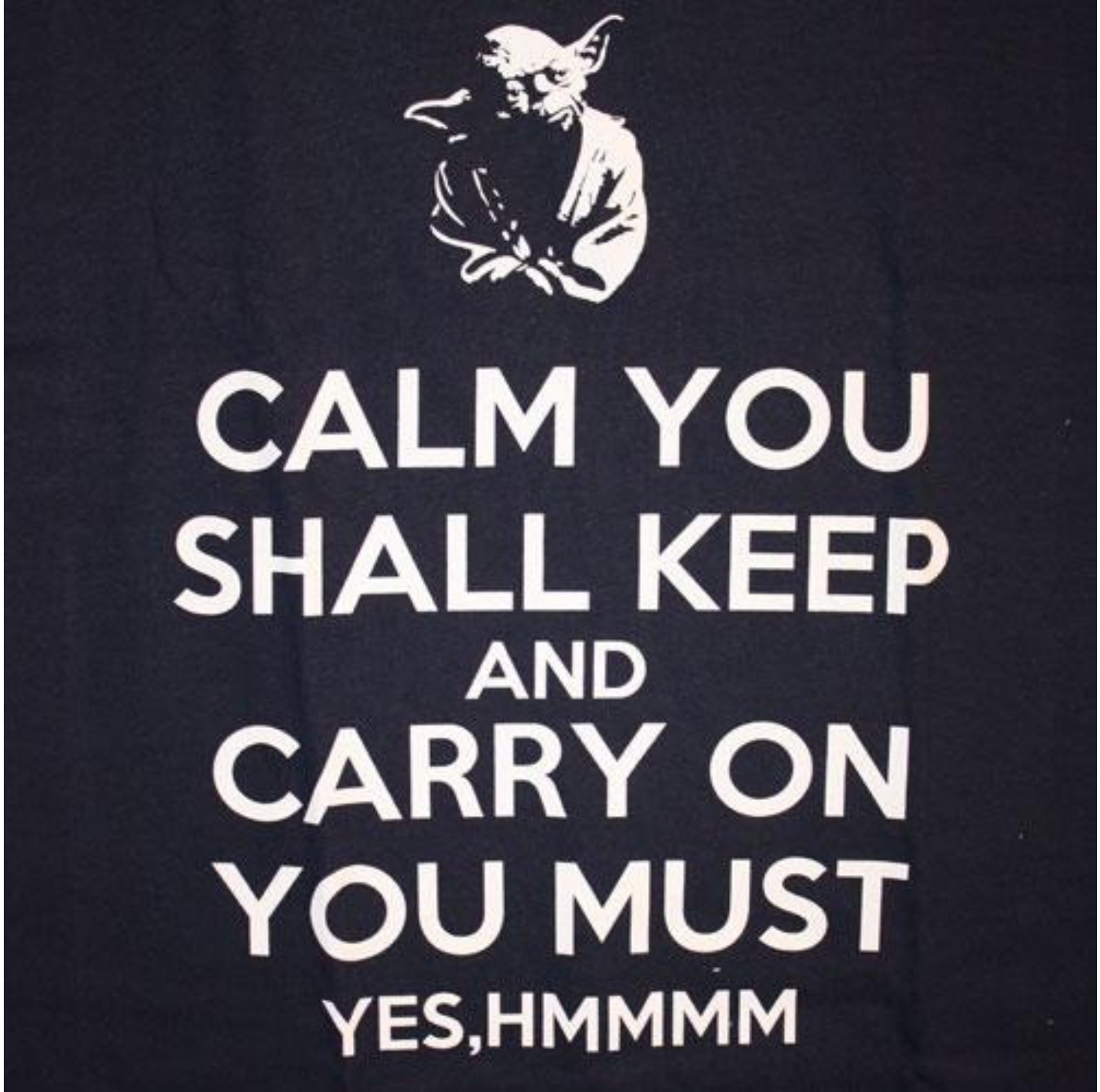
Philosophy

- Unified tools for different analysis groups – work from the same framework
  - Don't want propagation of incompatible stuff or wheel reinventions
- CAFs are independent of ART framework
  - Heavy reconstruction lifting happens in ART via the nova executable and ART module interfacing
  - CAFAna is just for assembling the final analysis from simple ROOT TTree-based CAF files
- Building blocks for putting together an analysis macro
  - Not some monolithic program that constrains you
  - Highly extensible
- Speed is very important
  - Histogram based
  - Encapsulation
  - Regular ROOT macros will get you only so far before becoming unbearably slow and cumbersome!
- Make reasonable plots by default
  - Standardises plot making and styles also
  - https://cdcvs.fnal.gov/redmine/projects/novaart/wiki/NOvA_plot_style

Philosophy

- Unified tools for different analysis groups – work from the same framework
  - Don't want propagation of incompatible stuff or wheel reinventions
- CAFs are independent of ART framework
  - Heavy reconstruction lifting happens in ART via the nova executable and ART module interfacing
  - CAFAna is just for assembling the final analysis from simple ROOT TTree-based CAF files
- Building blocks for putting together an analysis macro
  - Not some monolithic program that constrains you
  - Highly extensible
- Speed is very important
  - Histogram based
  - Encapsulation
  - Regular ROOT macros will get you only so far before becoming unbearably slow and cumbersome!
- Make reasonable plots by default
  - Standardises plot making and styles also
  - https://cdcvs.fnal.gov/redmine/projects/novaart/wiki/NOvA_plot_style




CALM YOU
SHALL KEEP
AND
CARRY ON
YOU MUST
YES,HMMMM

cafe

% setup_nova –b maxopt
[*TOP TIP 1 – Runs about 2x faster if you use the optimised build*]
[*TOP TIP 2 – Good practice to setup the nova release the CAF files were produced in*]

```
% cafe --help
** NOvA Common Analysis Format Executor **
usage: cafe [-h] [-b] [-q] [-bq] [--gdb | --valgrind]
            script.C [args [args ...]]

positional arguments:
  script.C              the root script to run. Append a + to force recompilation
  args                  optional arguments passed to script main function

optional arguments:
  -h, --help            show this help message and exit
  -b, --batch           batch mode, no graphics output
  -q, --quit            quit at end of job
  -bq                   shorthand for -b -q
  --gdb                 run root under gdb
  --valgrind            run root under valgrind
```

Alternatively you may specify a single .root file to open it in ROOT
libraries loaded.

Spectrum Class

- A Spectrum consists of a histogram representing the number of events per bin, and a POT holding the exposure that those event counts represent
  - No more explicit handling of POT which eliminates mistakes
- http://nusoft.fnal.gov/nova/novasoft/doxygen/html/classana_1_1Spectrum.html

- For plotting, a Spectrum can be converted into a TH1, scaled to some particular exposure
- For a data/MC comparison you would plot the data at its own exposure, and the MC scaled to the data's POT

[1D] Spectrum (std::string label, const Binning &bins, SpectrumLoaderBase &loader, const Var &var, const Cut &cut=kNoCut, const Var &wei=kUnweighted)

[2D] Spectrum (std::string label, SpectrumLoaderBase &loader, const Binning &binsx, const Var &varx, const Binning &binsy, const Var &vary, const Cut &cut=kNoCut, const Var &wei=kUnweighted)

```
// Spectrum to be filled from the loader
Spectrum len("Track length (cm)", bins, loader, kTrackLen, kIsNumuCC);
```

SpectrumLoader

- Reads in the StandardRecord objects from one or more CAF files and fills spectra with their contents
- http://nusoft.fnal.gov/nova/novasoft/doxygen/html/classana_1_1SpectrumLoader.html
- Only activates the branches necessary (speedy)
- Only scans through a file once (more speed)

```
// Environment variables and wildcards work
const std::string fname = "$NOVA_DATA/mc/S13-06-18/genie/fd/fd_r*_s?0_*fhc*nonswap*caf.root";

SpectrumLoader loader(fname);  // create loader for each 'data' type
....
// Once all spectra are registered, loop through input files,extracting variable values and filling histograms. Total POT is accumulated and used to set spectra POT
loader.Go()
```

Cut and Var

- When filling a histogram, it's necessary to know what to fill it with
- Var is a function that is given a StandardRecord object and returns a number
- Cut is similar, but returns a boolean indicating whether or not to use an event

- Pass a Var, and optionally a Cut, when registering a Spectrum
- Generally useful Cut or Var are found in CAFAna/Cuts.h and CAFAna/Vars.h
- Cut objects work with the usual logical operators so it's easy to combine them

Binning

- Easy way to keep binning options portable
- A few are predefined:
  - kNumuEnergyBinning, 100 bins equally spaced between 0 and 10
  - kNueEnergyBinning, 40 bins equally spaced between 0 and 10
  - kTrueEnergyBins, 100 bins unequally spaced and smaller at lower values
- To define new binning:

```
const Binning bins = Binning::Simple(nbins, min, max);
```

- Also has ability to create more complicated bins from vector of edges

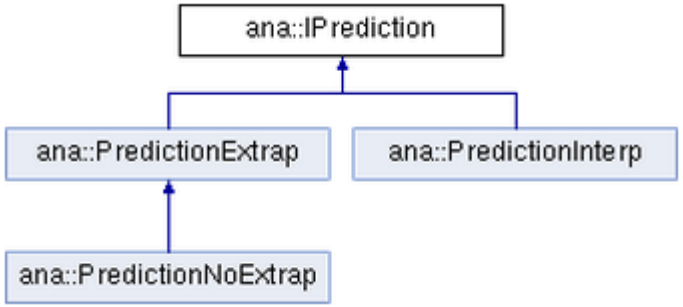
OscillatableSpectrum

- An OscillatableSpectrum is similar to a Spectrum, but with a second dimension representing true energy
- With the use of an OscCalculator this allows a 1D spectrum to be produced at any oscillation parameters
  - OscCalculatorPMNSOpt is the fastest and most precise oscillation calculator
- OscillatedSpectrum::Oscillate() --> Operates by weighting each bin of the histogram by the calculated oscillation probability, and then summing the totals onto the user axis, yielding a one-dimensional Spectrum
- Specialisation of ReweightableSpectrum, the more general class where the second axis is customisable

- Constructed in same fashion as a Spectrum

```
// With true energy axis too
OscillatableSpectrum sosc("Number of hits in slice", bins, loader, kNHit, kIsNumuCC);
// Make a calculator. This is the fastest variant
osc::OscCalculatorPMNSOpt calc;
// Can oscillate a spectrum (numu->numu)
Spectrum soscd = sosc.Oscillated(&calc, 14, 14);
```

Prediction



- In order to do an oscillation analysis, you need a method to produce an oscillated prediction to compare to the data
- The most basic prediction is PredictionNoExtrap
- It holds OscillatableSpectrum objects corresponding to each true CC component (ve/vμ, neutrino/antineutrino, appearance/survival) and a Spectrum representing the NC component
- When asked to predict the spectrum, it simply oscillates all of these components returns the sum

```
// All interaction types
PredictionNoExtrap pred(loader, loaderSwap, kNullLoader, "Number of hits in slice", bins, kNHit);
```

- A more complex Prediction would be used to represent an FD prediction extrapolated from ND data. Beyond the scope of the tutorial

DataMC Comparisons

// Overlay MC spectrum with data spectrum, POT normalised

DataMCComparison(const Spectrum & data, const Spectrum mc)

// Overlay MC spectrum with data spectrum, area normalised

DataMCComparisonAreaNormalized(const Spectrum & data, const Spectrum mc)

// Plot MC broken down into flavour compenents, overlayed with data

DataMCComparisonComponents(const Spectrum & data, const Spectrum mc)

…and much more!

- **Experiment** class: Takes an oscillation calculator representing some set of oscillation parameters, and returns a χ^2 (or log-likelihood)
- SingleSampleExperiment is constructed with a Prediction object and an observed data Spectrum. The χ^2 function calls Predict() on the Prediction to get an expected spectrum and then calculates the log-likelihood with the data

- **Fitter** class: Takes an Experiment and one or two oscillation parameters to determine, drives MINUIT fit

- **Surface** class: Used to calulate a χ^2 surface as a function of two parameters and to draw confidence intervals.
  - Performs a scan over the parameters, recording χ^2 value at each pint in the 2D space. Also performs minimisation

- Honourable class mentions:
  - **XSec** , **Decomp** etc

Final Word

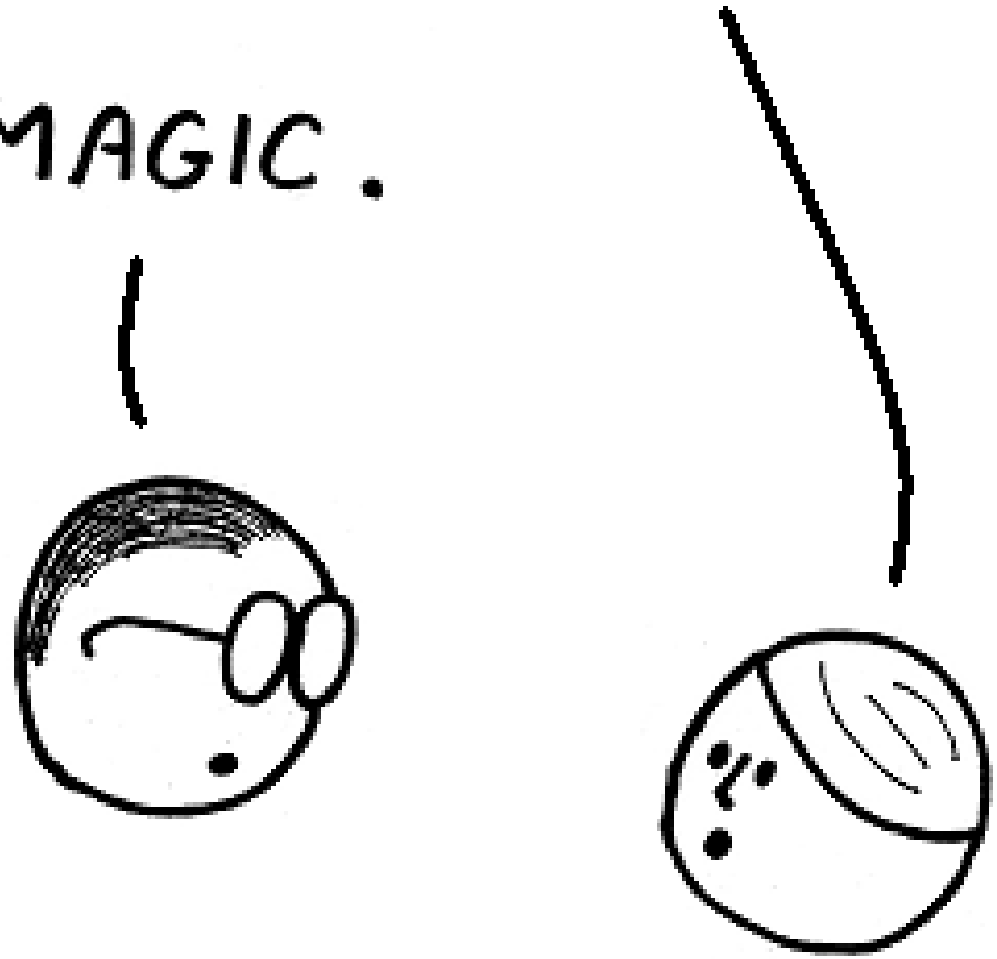
- **Example macros:**
  - **CAFAna/tute/demo{0-6}.C for overview of featured classes**
  - **CAFAna/nue/nue_ana_basic.C for basic full ν_e analysis**
  - **CAFAna/numu/numu_ana_basic.C for basic full ν_μ analysis**
  - **CAFAna/test/demo_xsec.C for cross-section features**
  - **CAFAna/test/fartonearextrap.C for full functionality of extrapolation**

- I echo Gareth's disclaimer:

- At any point, the tech notes, tutorials or macros may become out of date. The same applies to the wiki. Don't be afraid to update them! Doxygen is your friend.

- We hope you enjoyed today's tutorial sessions and learnt a lot!

HOW DOES COMPUTER PROGRAMMING WORK?



金玄浩