# Turn-Key Cross-Sections

Thomas Britton

# Overview

- Cross sections have essentially three components
  - Yields
  - Acceptance
  - Flux
- The lowest hanging fruit to have the biggest impact is a generic framework for fitting mass distributions and easily calculating yields

# Generic Fitting tool

- LineShape_Library

```
LineShape_Library sig("sig", "sig", BValues,PiPlus,PiMinus);
 sig.CreateComponent("rho",JBREITWIGNER,5.3,1, .757, .15,true);
 sig.CreateComponent("omega",JBREITWIGNER,5.3,1, .783, .008,false);
sig.CreateComponent("sigma",NONRESONANT,5.3,1, 1);
```

Instantiated with a vector of values to fit (currently just a vector of one thing...mass) and the two daughters via the ParticleType.h enum list

Then you can simply add components through the CreateComponent function.

Each component takes in some parameter guesses and generates other parameters needed. Currently this is the scale parameter, the L in the decay, initial mass guess, initial width guess, and whether or not you are normalizing to this state

# Generic Fitting tool cont

- The library is fitting the intensity as a PDF. Thus, each component has a relative normalization and there exists an overall normalization
- $Y = |c_A * A + c_B * B \ldots|^2$
  - Where the A's and B's are the models

```
enum {POLYNOMIAL=0, BREITWIGNER, JBREITWIGNER,NONRESONANT};
```

Currently the above are implemented, but it is easy to add more

# The Fitting

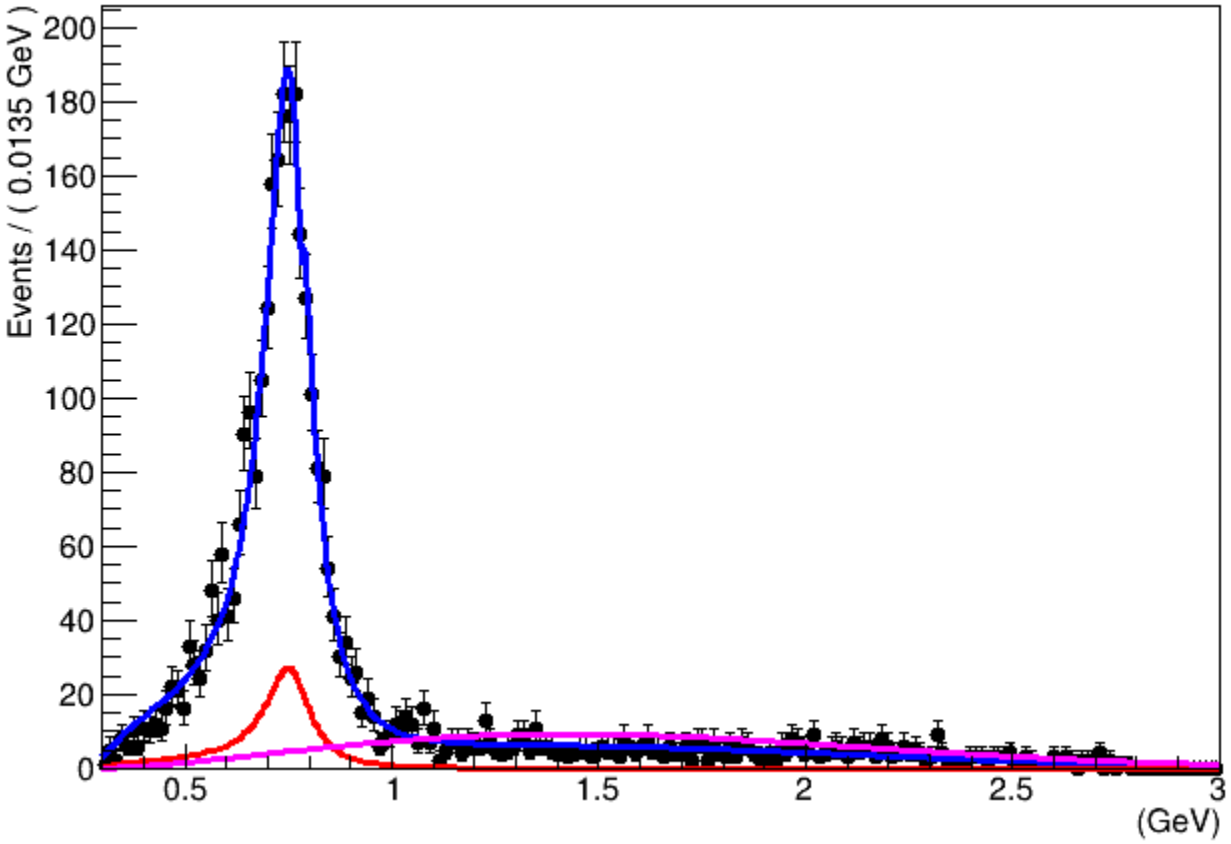
```
RooRealVar nevents("nevt", "N_{evt}", 1e3,0,1e8);
nevents.setError(10.0);
 nevents.setConstant(false);
 RooAddPdf model("model", "G+poly", RooArgList(sig), RooArgList(nevents));
```

First build a model out of your instance of the LineShape_Library class and the overall normalization (the strings don't matter much right now)

```
model.fitTo(*OOTsubsubset,RooFit::Extended(),RooFit::SumW2Error(kFALSE),RooFit::NumCPU(4));
```

Then call fitTo on your RooData Set

# The Fitting Example




Note the component normalization are incorrect and currently just guide the eye

# Helper functions

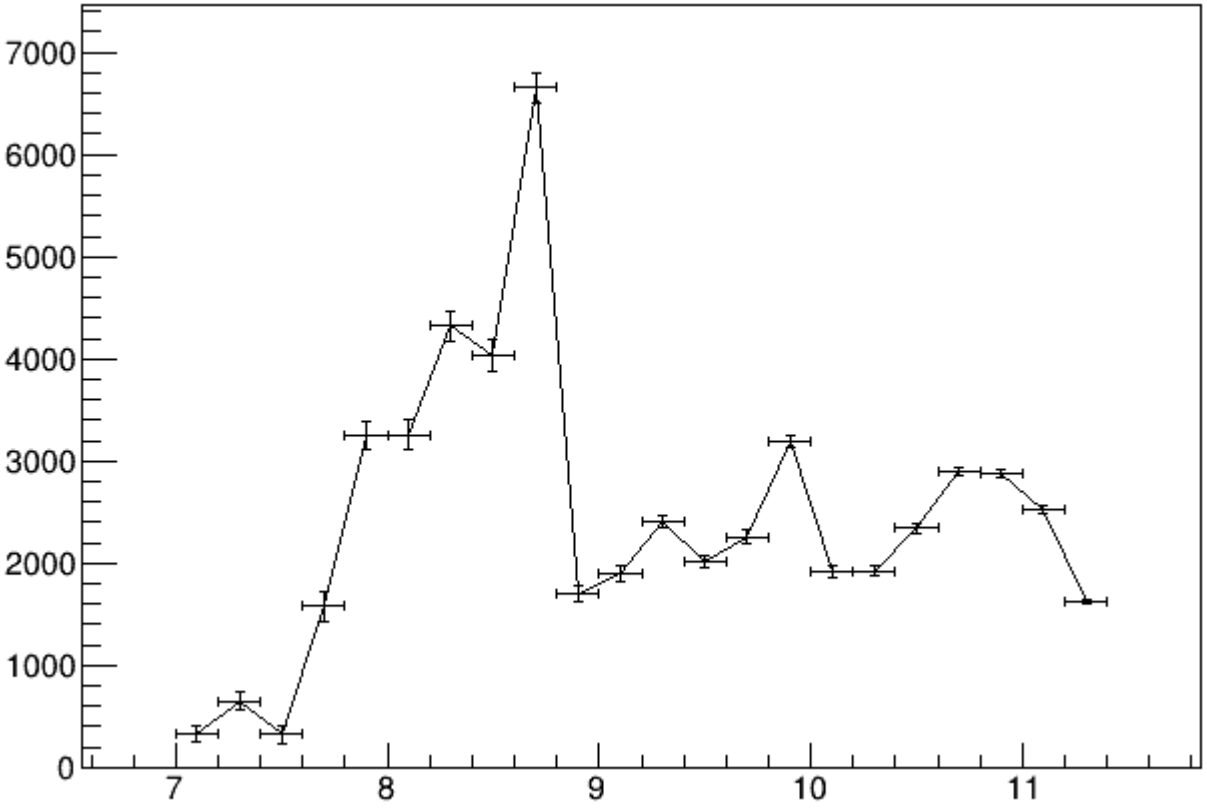
Many helper functions allow you to get single components,  parameters from those components and even easily compute the yields as seen below

```
double rho_yield=nevents.getVal()*sig.GetIntegral("rho",.5,1.1);
double omega_yield=nevents.getVal()*sig.GetIntegral("omega");
double sigma_yield=nevents.getVal()*sig.GetIntegral("sigma",.5,1.1);
```

Better bundling and more simplification will follow with development

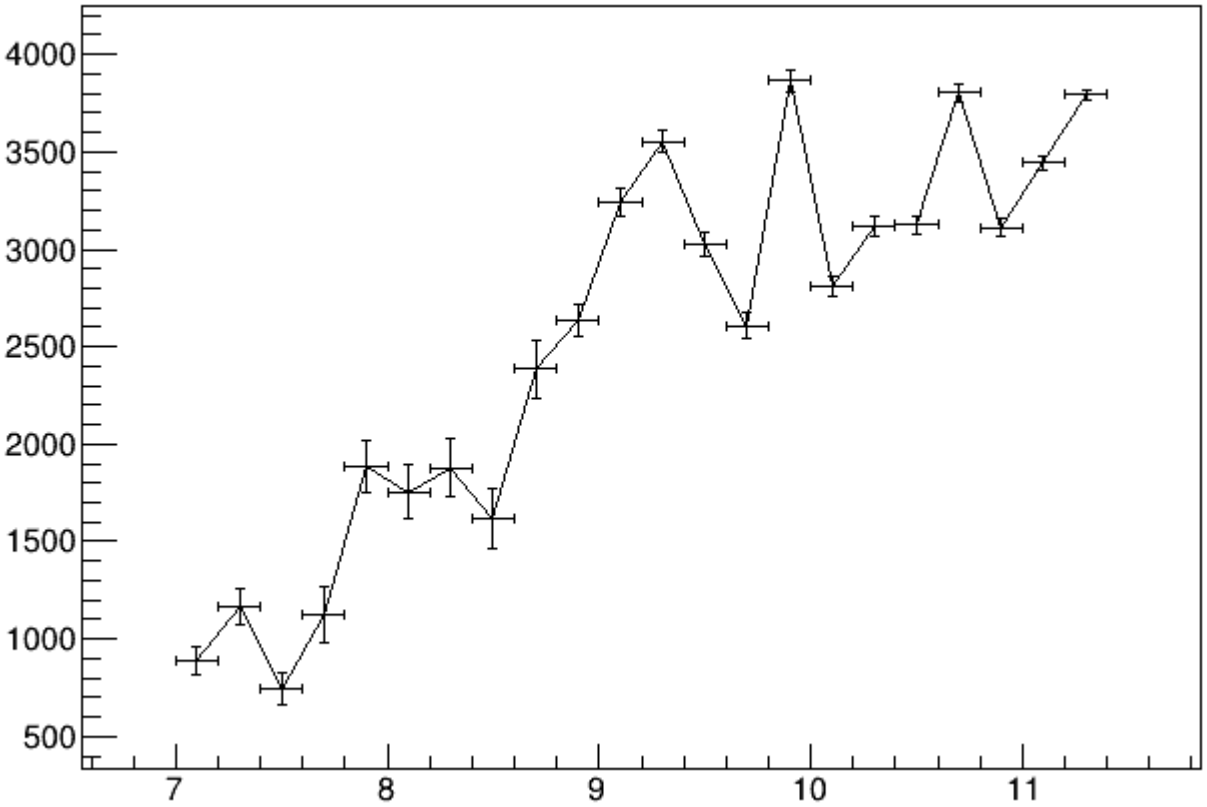
# Initial results

Accidental subtracted Yields of rho



# Initial results

"Cross-Section"  (no efficiency included and the flux may not be correct)



# Future work

- Incorporate more models (eg Flatte) for use
- Bundle helper functions better
  - And add more functions useful for computation of cross-sections
- Develop an interface:
  - You provide a model, a couple of RooDataSets and a couple of histograms and it will produce canonical plots with a cross-section
- Hope to have something presentable in the next couple of weeks on rho