Advanced analysis / RooFit / HistFactory tutorial

Aaron Armbruster

CERN

September 17, 2017

What’s covered here

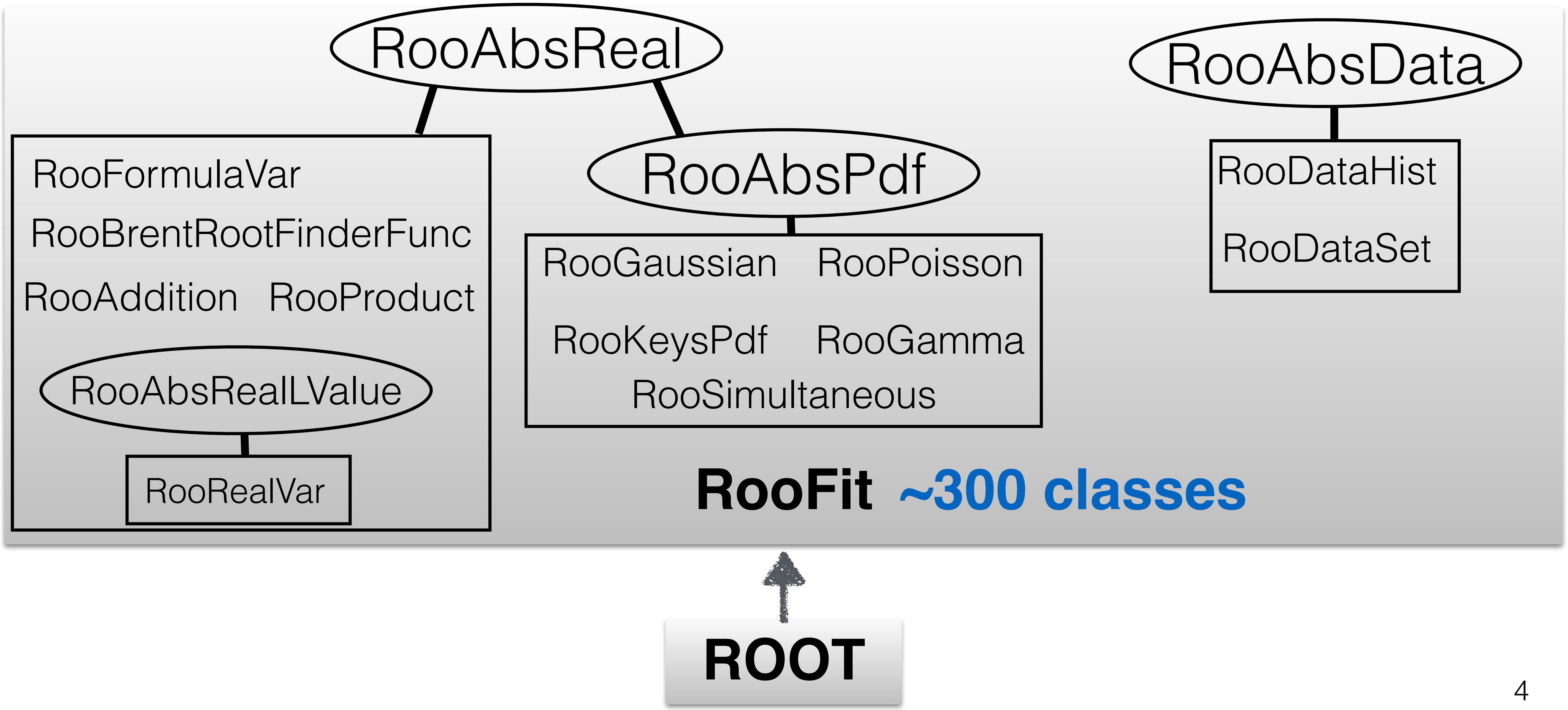
- Setting up the tutorial
- What is RooFit?
- Brief statistics review
- Practical examples
    - Simple number counting experiment with systematics
    - Fitting signal on top of background (binned shape fit)
    - Profiled unfolding
    - Reparametrization
    - Combinations
    - Compatibility tests
    - Dealing with covariance matrices
- Other tools
- Debugging and common problems

Setting up

- Tutorial is on git: **> git clone https://:@gitlab.cern.ch:8443/armbrust/RooFit-tutorial.git**
- Latest version also on afs:
  - **/afs/cern.ch/work/a/armbrust/RooFit-tutorial**
- There is a large 3.4GB ntuple file that you don't necessarily need (histograms made from it are included in the git repository), but I'll keep it available here for posterity:
  - **/afs/cern.ch/work/a/armbrust/tree.root**
- Setup script setup.sh for running on lxplus:
  - **> source setup_lxplus.sh**
- The tutorial should also be able to be ran locally on your laptop with a reasonably recent ROOT version (5.34.X) after running setup.sh
  - **> source setup.sh**
- On Mac, for some reason ROOT < 5.34.21 didn't want to work for me, but your mileage may vary
- That's all!

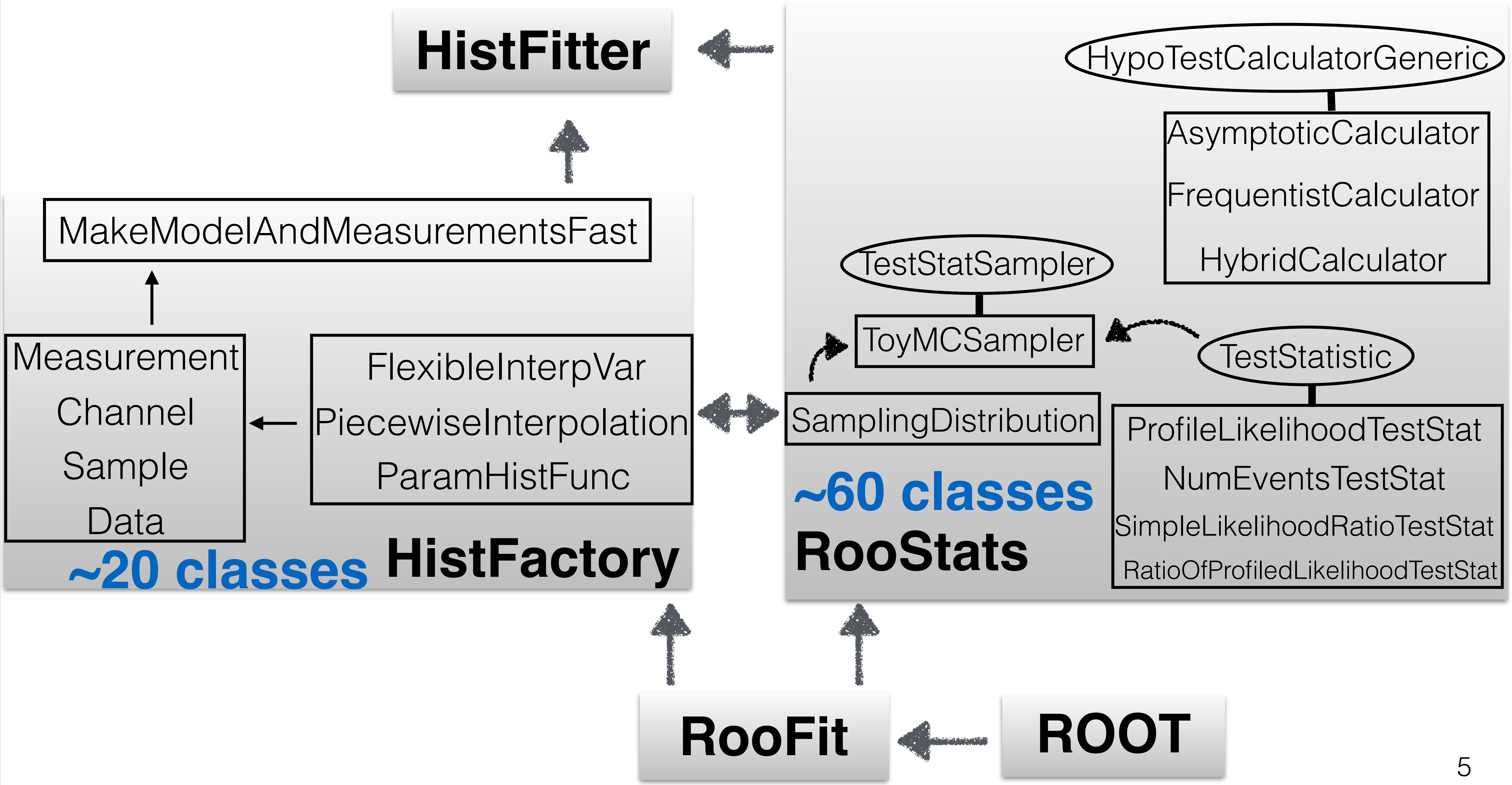
What is RooFit?

- **RooFit** is an object oriented modelling framework written in c++, built on top of **ROOT**
- Heavily uses abstraction to be able to create high level models, which can later be built upon by even higher level packages (RooStats, HistFactory, etc) that implement concrete statistical tools
- Somewhat high learning curve
    - Though almost all of the time the common design themes are followed, occasionally a class has unexpected behavior that makes debugging difficult!



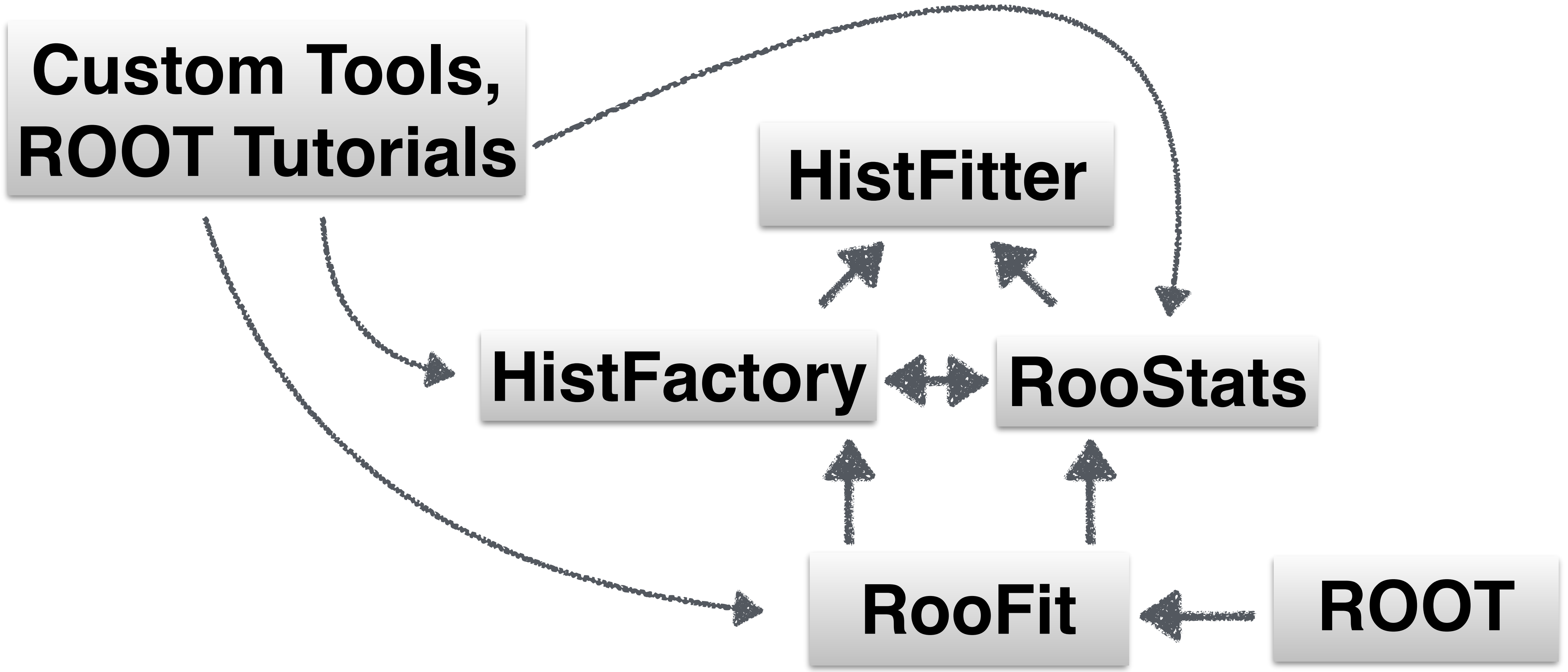
What is RooFit?

- **HistFactory** and **RooStats** are built on top of **RooFit**
- **HistFactory** is a class-based package that builds a likelihood and data out of input histograms, including systematics, which imported into a RooWorkspace and written to a TFile
- **RooStats** is a collection of tools used to perform statistical calculations
- **HistFitter** is a python wrapper that interfaces the **HistFactory** output with **RooStats** tools



What is RooFit?

- Many custom tools developed by different analysis groups that combine elements of RooFit, HistFactory, and RooStats to build statistical models of the analysis and do statistics
    - Many times rederived from previous analysis' tools and adapted to new analysis
    - Most of the time not "officially" supported
    - Usually similar themes and functionality in all of these
    - However, in the hands of an expert, custom built tools lead to a very flexible framework for doing statistics for your specific analysis, which is why they are so widely used compared to more official tools
- ROOT tutorials contain not only examples for RooFit, HistFactory, and RooStats, but also a nice set of top-level tools you can use out-of-the-box (or with minimal tuning) for your analysis



Brief statistics review: Likelihood formalism

$$\mathcal{L}(\mu, \theta) = \mathcal{L}_{data}(N|\vec{\mu}, \theta) \times \mathcal{A}(\tilde{\theta}|\theta)$$

- Elements of a likelihood:
  - **L_{data}** is the portion of the likelihood representing *your* data "N"
  - **A** represents some auxiliary measurement already done for you, compressed into a single term such as a unit Gaussian or Poisson
    - Could be JES, lepton efficiency, scale, resolution, some PDF EV, etc
  - **N** is eg the number of observed events after selections, or the observed p^T_{ll} distribution for Z's, etc
  - **μ** is the set of parameters of interest (POIs) that you ultimately wish to measure, eg an integrated or differential cross-section or a SM parameter
  - **θ** is the set of nuisance parameters (NPs) corresponding to the auxiliary measurements **A**, and represents **new** measurements in your phase space of data. **θ** are most commonly used to implement systematics or background normalizations
  - **~θ** is the set of **old** measurements, and is normally fixed at a single value, except in toy MC
- Today I'll focus on binned likelihoods, which are just products of Poisson probabilities for each bin: $L_{data} = \Pi_i\, P(N_i \mid E_i(\mu, \theta))$

Brief statistics review: Test statistics

- A test statistic is any function of the data, which has an obtainable probability distribution
- The Neyman-Pearson lemma states that, in the absence of systematics, the ratio of likelihoods between two hypotheses is the most powerful statistical test to separate those hypotheses:
    - $\Lambda(x) = L(x \mid H_1) / L(x \mid H_2)$
- Strictly speaking, this proof doesn't hold in the presence of systematics, but this is the motivation for the **profile likelihood ratio** used in modern HEP

$$\lambda(\mu) = \frac{\mathcal{L}(\mu, \hat{\hat{\theta}}_\mu)}{\mathcal{L}(\hat{\mu}, \hat{\theta})}$$

- Here, the numerator is the likelihood maximized at a fixed point of the POIs (aka conditional likelihood), and the denominator is the likelihood maximized over all paramaters (unconditional likelihood)
- The one-sided profile likelihood test statistic t_μ is simply the -2 log of this

$$t_\mu = -2 \ln \lambda(\mu)$$

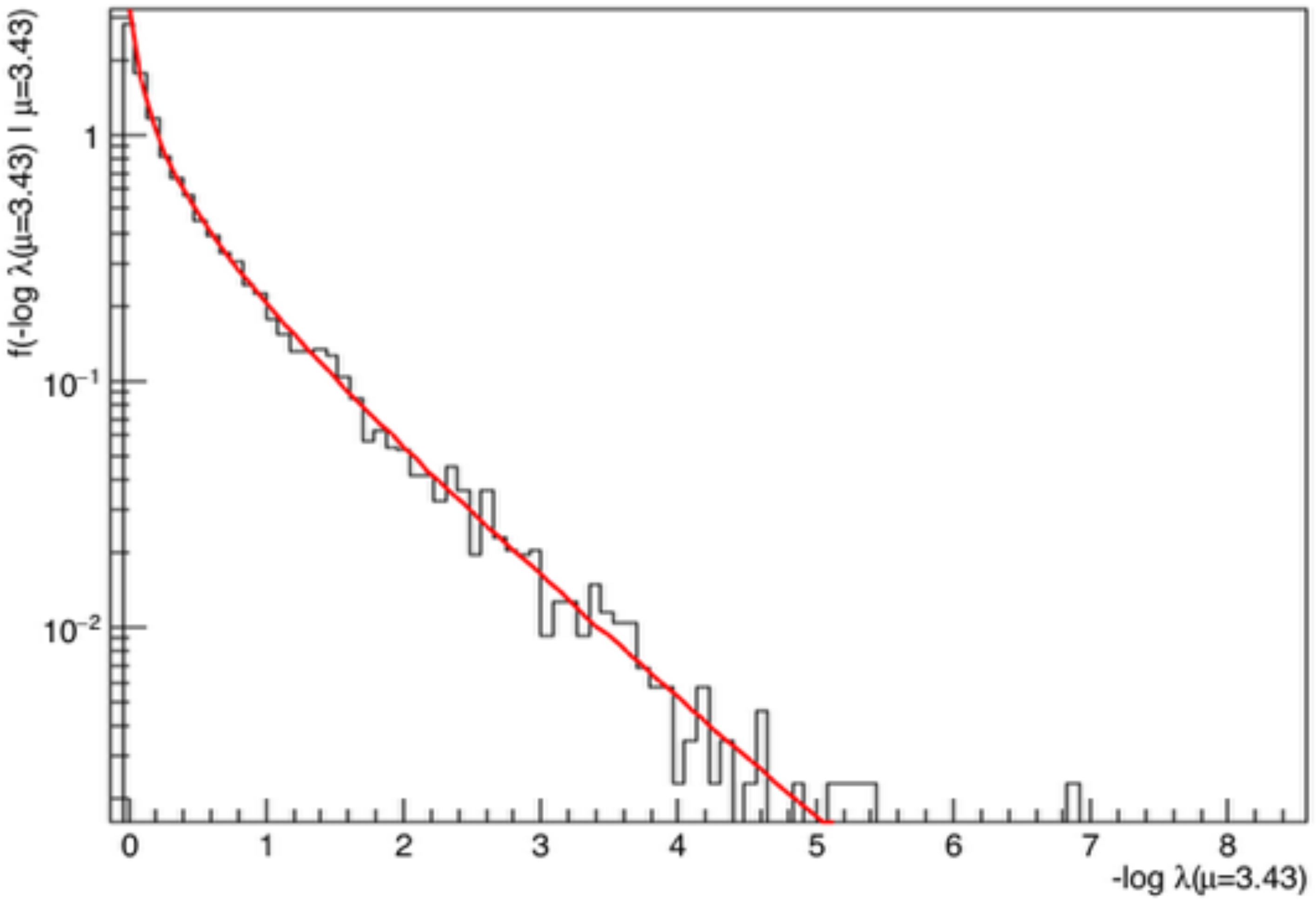
- Less trivial forms of this exist and are used in searches, but for today this is the only thing we'll use. See arxiv.org/abs/1007.1727 for more details

Brief statistics review: Sampling distributions

$$t_\mu = -2 \ln \lambda(\mu)$$

- The sampling distribution of a test statistic is simply it's probability distribution under pseudo-experiments, ie under Poisson variations of N observed in each bin of your distribution
- Wilk's theorem states that, for a single POI μ, the distribution of t_μ asymptotically (when N is large) approaches a χ^2 with one degree of freedom

$$f(t_\mu|\mu) = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{t_\mu}} e^{-t_\mu/2}$$



- Integrating this, tail probability for an observed value of t_μ is

$$p_\mu = 1 - F(t_\mu|\mu) = 2\left(1 - \Phi\left(\sqrt{t_\mu}\right)\right)$$

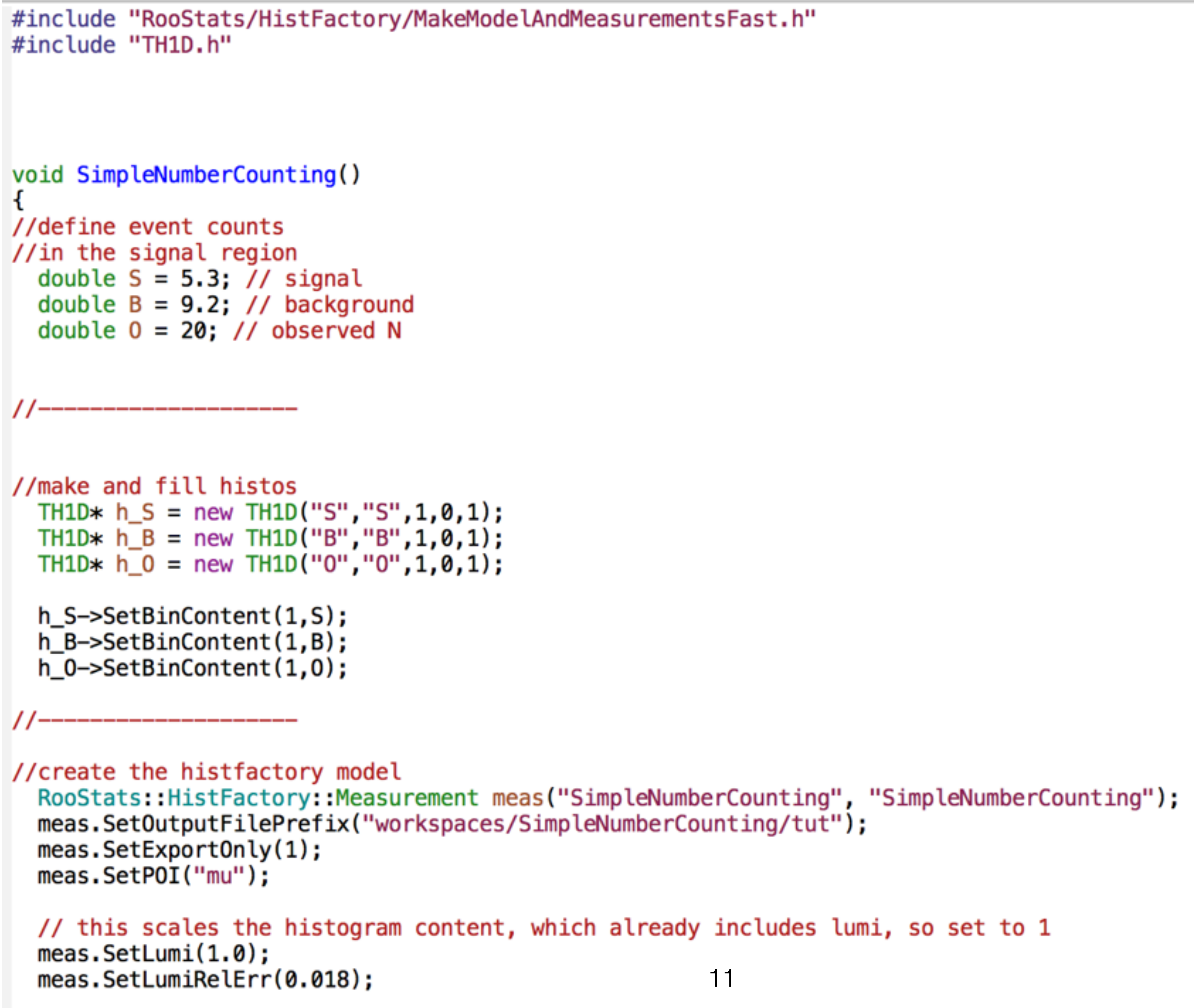
- Notably, $p_\mu = 0.32$ (ie, 68% CL or 1σ) corresponds to $\text{sqrt}(t_\mu) = 1$, and in general N σ corresponds to $\text{sqrt}(t_\mu) = N$

- Important feature: $f(t_\mu)$ doesn't depend explicitely on the parameters of the model μ, θ, only on the observed value t_μ itself!

Practical example: Simple number counting experiment

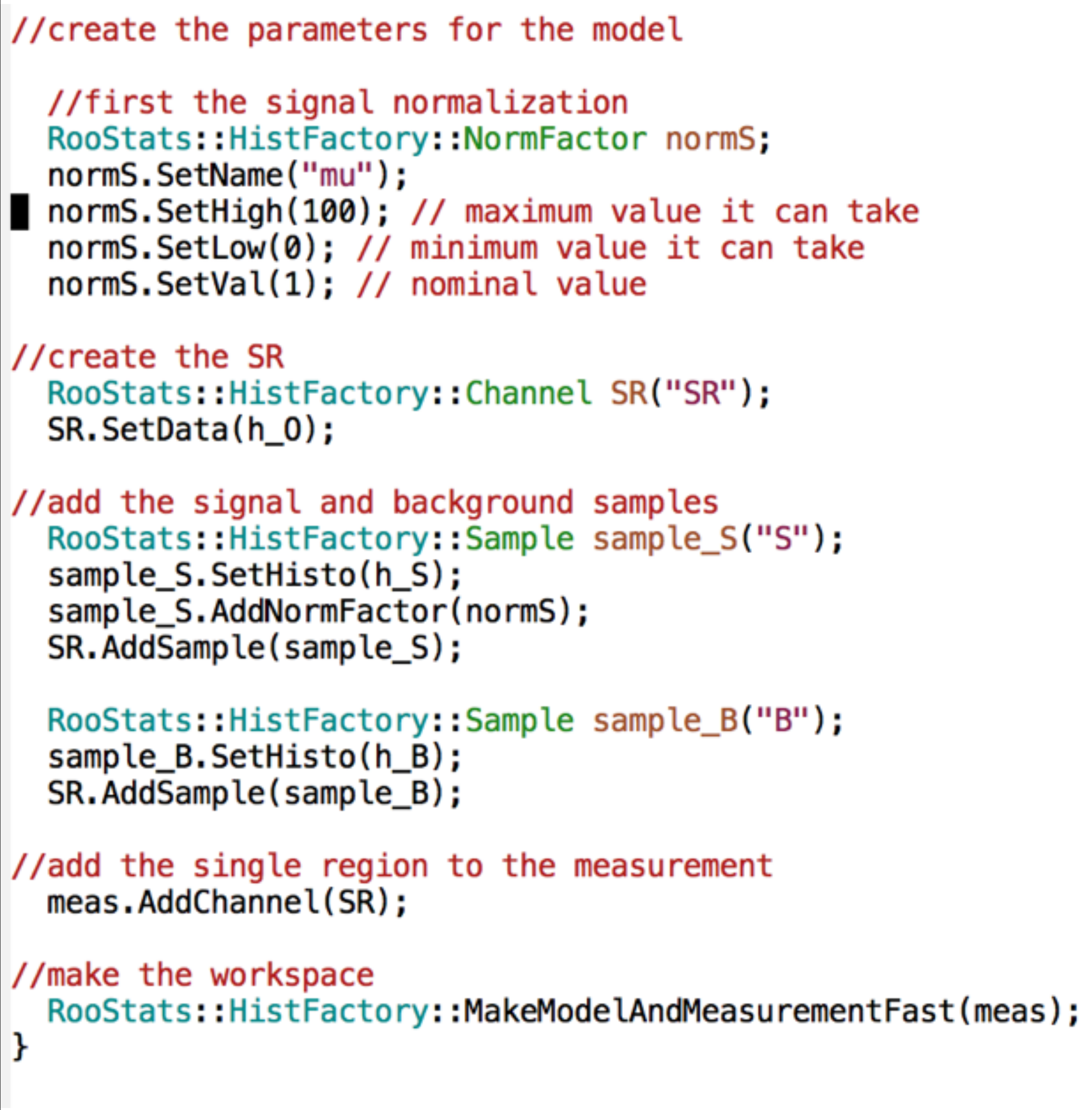
- Exercise:
  - Look through SimpleNumberCounting.C
  - Run it: **> root -b -q ex/SimpleNumberCounting.C**
  - Open file and look through workspace
- SimpleNumberCounting.C builds a HistFactory model for a single bin counting experiment:
- $L = P(N \mid \mu * S + B)$

SimpleNumberCounting.C

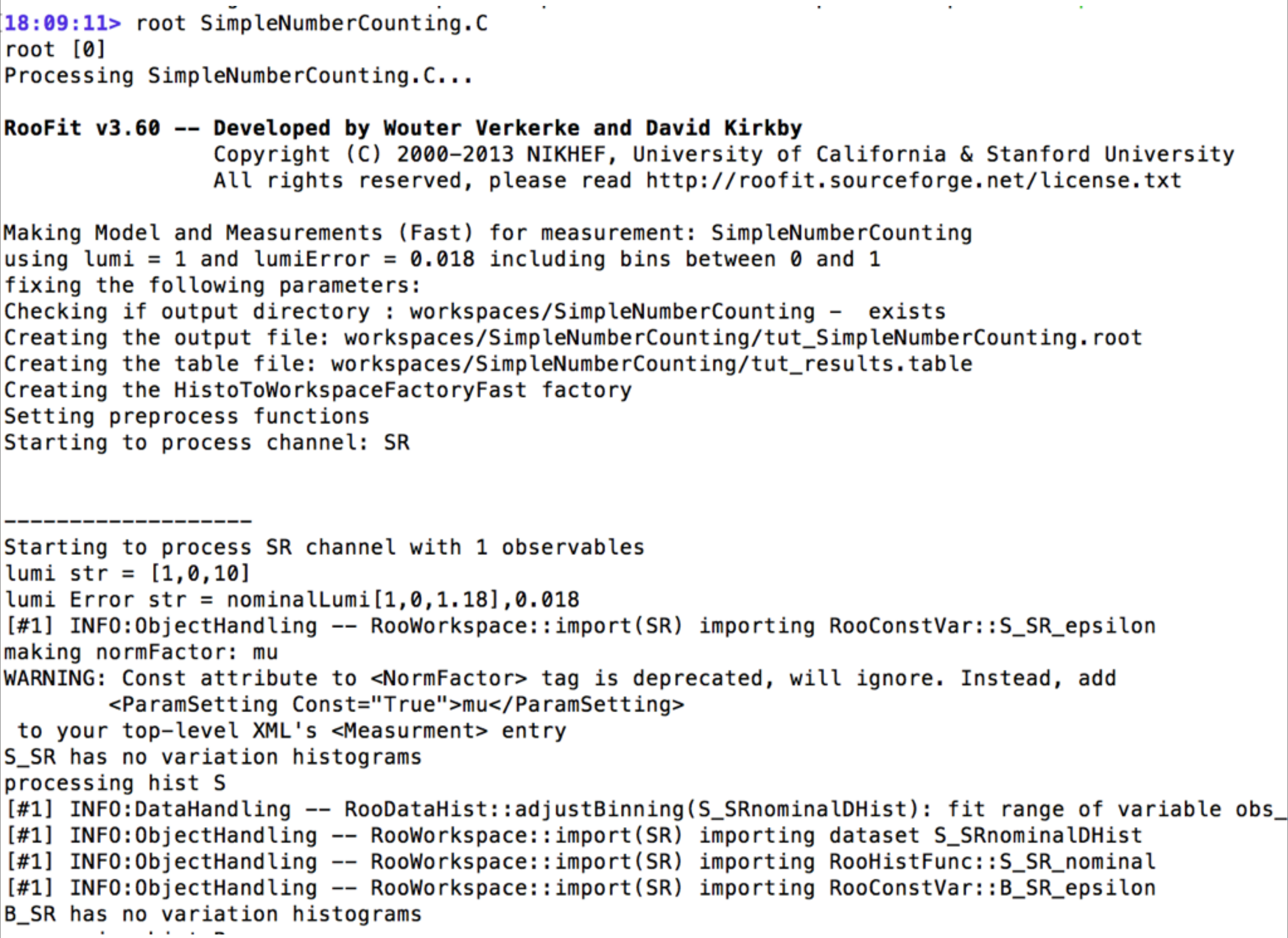


```
#include "RooStats/HistFactory/MakeModelAndMeasurementsFast.h"
#include "TH1D.h"




void SimpleNumberCounting()
{
//define event counts
//in the signal region
  double S = 5.3; // signal
  double B = 9.2; // background
  double O = 20; // observed N


//--------------------


//make and fill histos
  TH1D* h_S = new TH1D("S","S",1,0,1);
  TH1D* h_B = new TH1D("B","B",1,0,1);
  TH1D* h_O = new TH1D("O","O",1,0,1);

  h_S->SetBinContent(1,S);
  h_B->SetBinContent(1,B);
  h_O->SetBinContent(1,O);

//--------------------

//create the histfactory model
  RooStats::HistFactory::Measurement meas("SimpleNumberCounting", "SimpleNumberCounting");
  meas.SetOutputFilePrefix("workspaces/SimpleNumberCounting/tut");
  meas.SetExportOnly(1);
  meas.SetPOI("mu");

  // this scales the histogram content, which already includes lumi, so set to 1
  meas.SetLumi(1.0);
  meas.SetLumiRelErr(0.018);
```

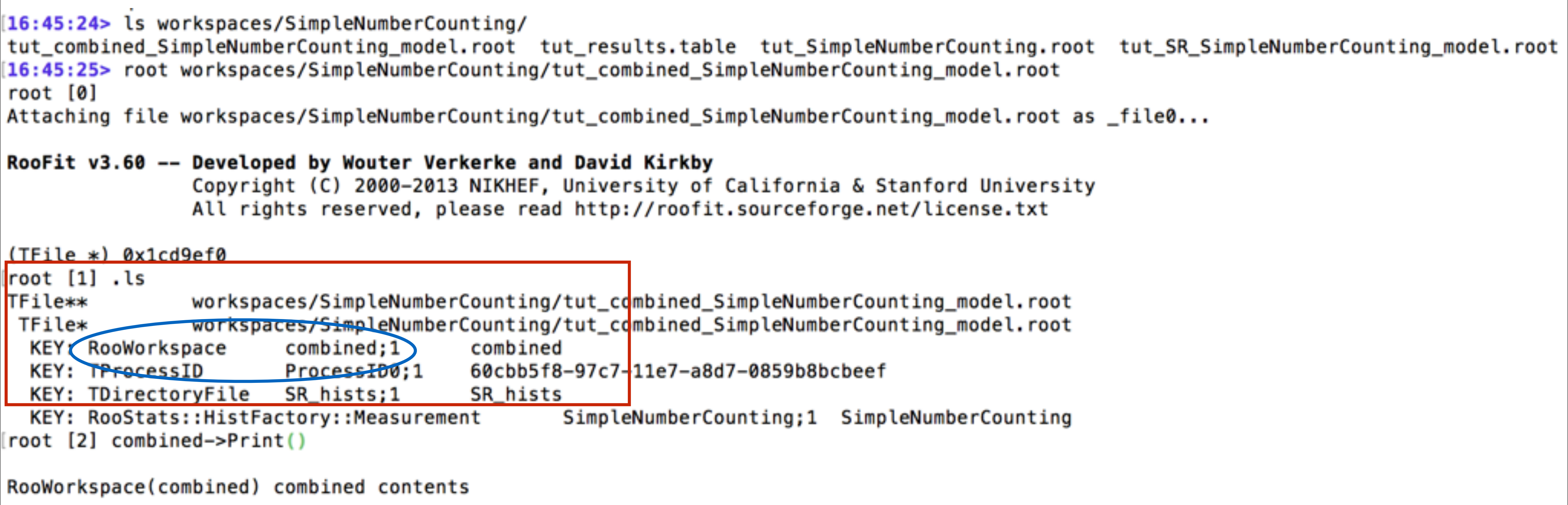
SimpleNumberCounting.C



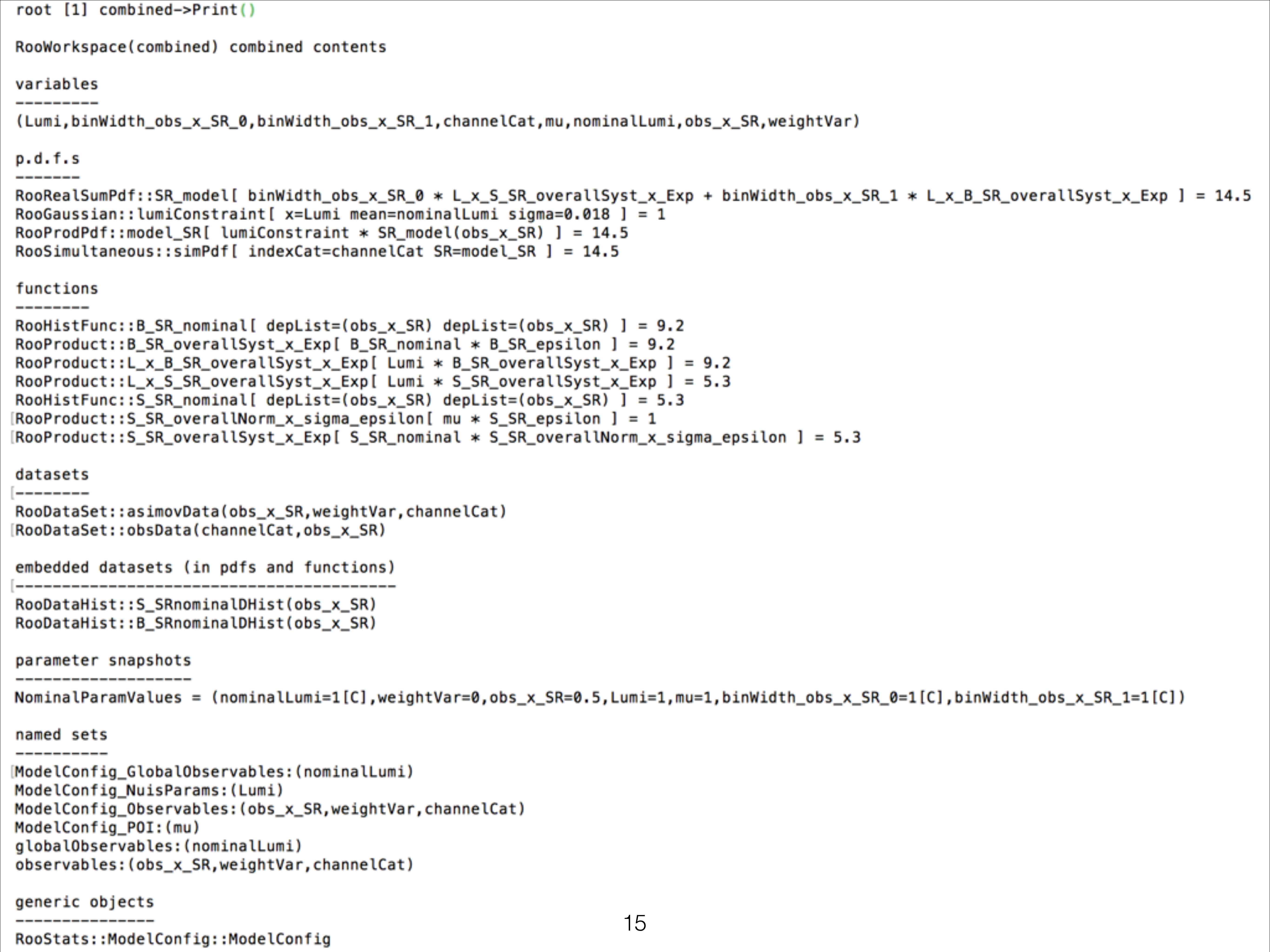
```
//create the parameters for the model

  //first the signal normalization
  RooStats::HistFactory::NormFactor normS;
  normS.SetName("mu");
  normS.SetHigh(100); // maximum value it can take
  normS.SetLow(0); // minimum value it can take
  normS.SetVal(1); // nominal value

//create the SR
  RooStats::HistFactory::Channel SR("SR");
  SR.SetData(h_O);

//add the signal and background samples
  RooStats::HistFactory::Sample sample_S("S");
  sample_S.SetHisto(h_S);
  sample_S.AddNormFactor(normS);
  SR.AddSample(sample_S);

  RooStats::HistFactory::Sample sample_B("B");
  sample_B.SetHisto(h_B);
  SR.AddSample(sample_B);

//add the single region to the measurement
  meas.AddChannel(SR);

//make the workspace
  RooStats::HistFactory::MakeModelAndMeasurementFast(meas);
}
```



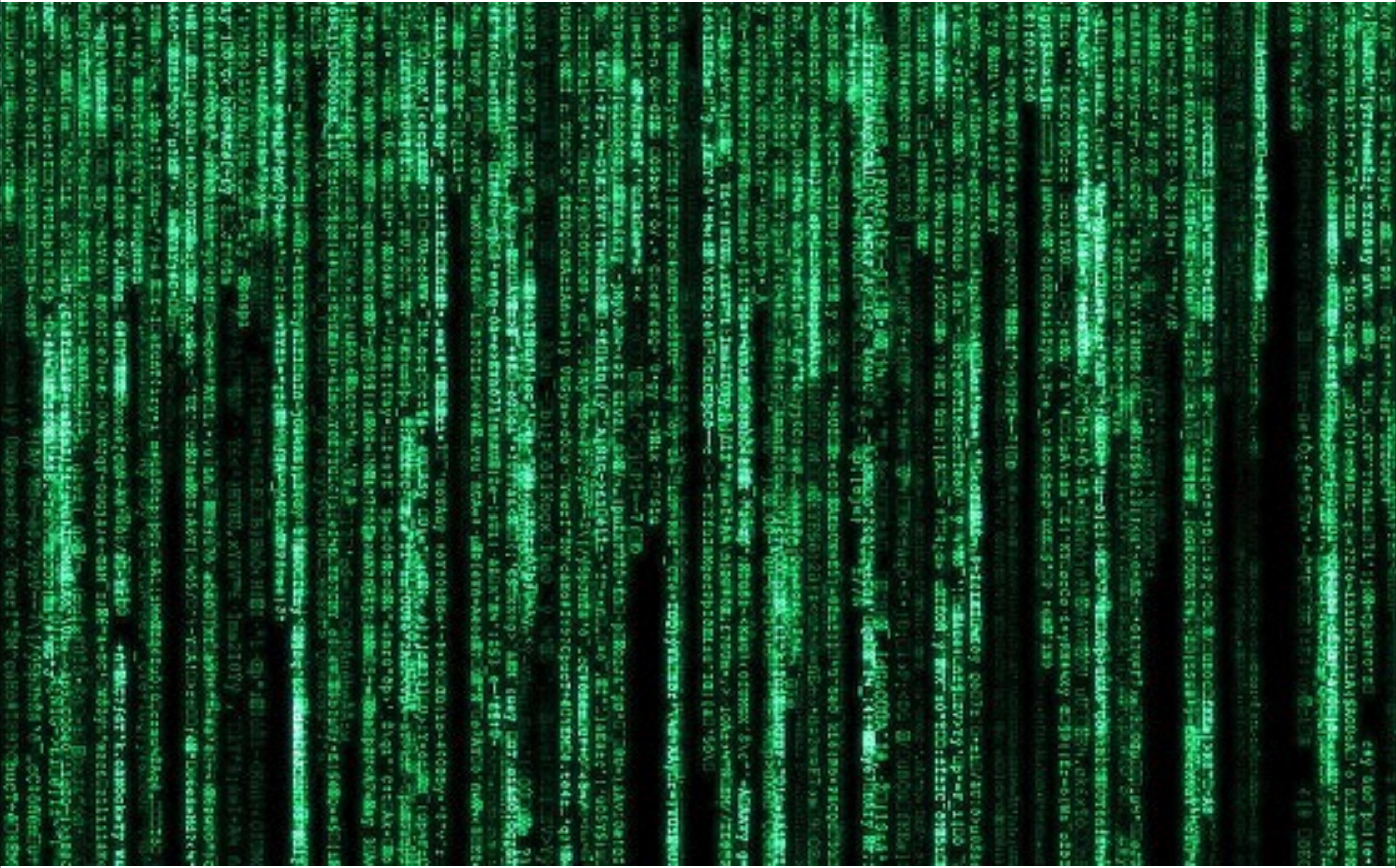
```
18:09:11> root SimpleNumberCounting.C
root [0]
Processing SimpleNumberCounting.C...

RooFit v3.60 -- Developed by Wouter Verkerke and David Kirkby
                Copyright (C) 2000-2013 NIKHEF, University of California & Stanford University
                All rights reserved, please read http://roofit.sourceforge.net/license.txt

Making Model and Measurements (Fast) for measurement: SimpleNumberCounting
using lumi = 1 and lumiError = 0.018 including bins between 0 and 1
fixing the following parameters:
Checking if output directory : workspaces/SimpleNumberCounting -  exists
Creating the output file: workspaces/SimpleNumberCounting/tut_SimpleNumberCounting.root
Creating the table file: workspaces/SimpleNumberCounting/tut_results.table
Creating the HistoToWorkspaceFactoryFast factory
Setting preprocess functions
Starting to process channel: SR



-------------------
Starting to process SR channel with 1 observables
lumi str = [1,0,10]
lumi Error str = nominalLumi[1,0,1.18],0.018
[#1] INFO:ObjectHandling -- RooWorkspace::import(SR) importing RooConstVar::S_SR_epsilon
making normFactor: mu
WARNING: Const attribute to <NormFactor> tag is deprecated, will ignore. Instead, add
        <ParamSetting Const="True">mu</ParamSetting>
 to your top-level XML's <Measurment> entry
S_SR has no variation histograms
processing hist S
[#1] INFO:DataHandling -- RooDataHist::adjustBinning(S_SRnominalDHist): fit range of variable obs_
[#1] INFO:ObjectHandling -- RooWorkspace::import(SR) importing dataset S_SRnominalDHist
[#1] INFO:ObjectHandling -- RooWorkspace::import(SR) importing RooHistFunc::S_SR_nominal
[#1] INFO:ObjectHandling -- RooWorkspace::import(SR) importing RooConstVar::B_SR_epsilon
B_SR has no variation histograms
```

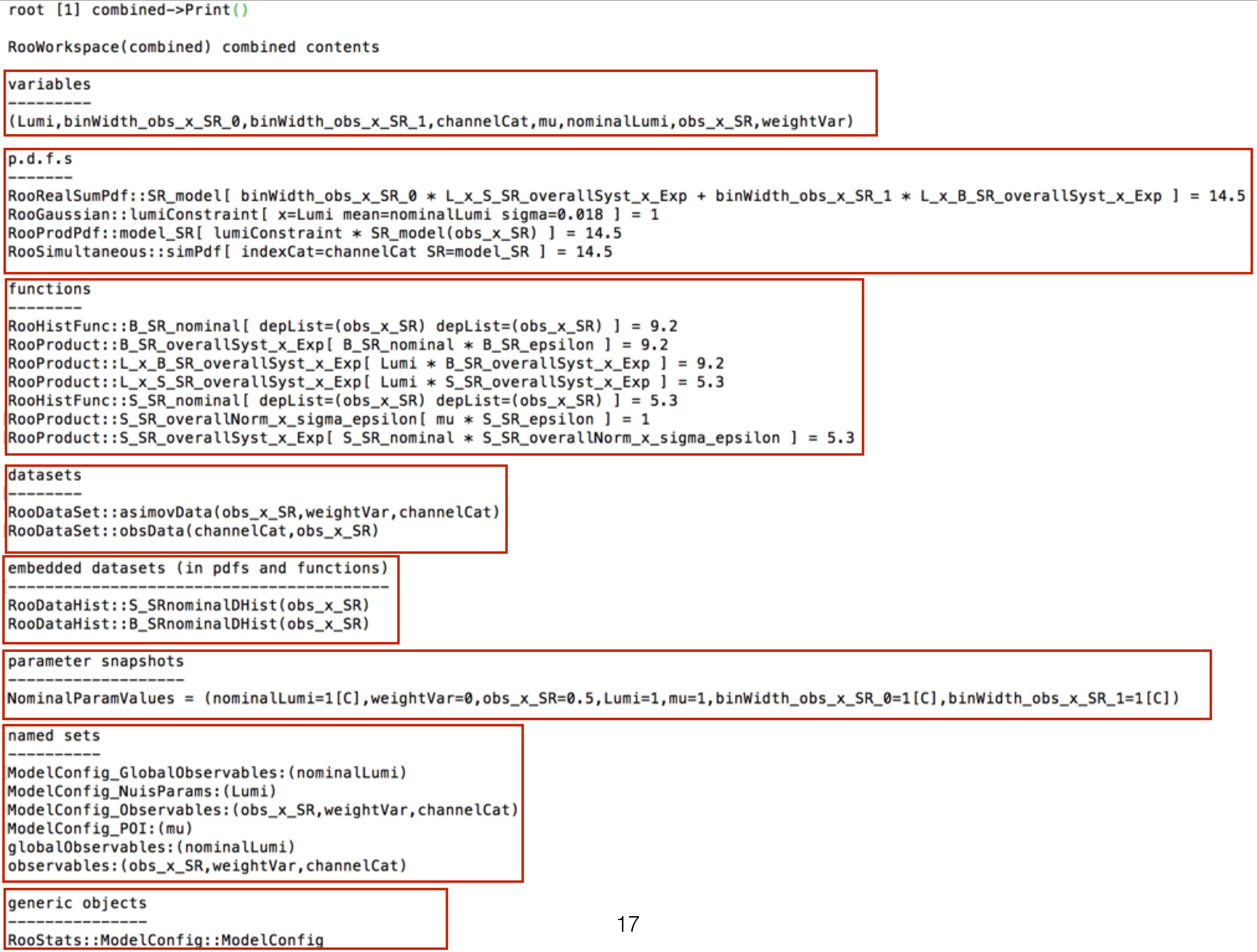


```
16:45:24> ls workspaces/SimpleNumberCounting/
tut_combined_SimpleNumberCounting_model.root  tut_results.table  tut_SimpleNumberCounting.root  tut_SR_SimpleNumberCounting_model.root
16:45:25> root workspaces/SimpleNumberCounting/tut_combined_SimpleNumberCounting_model.root
root [0]
Attaching file workspaces/SimpleNumberCounting/tut_combined_SimpleNumberCounting_model.root as _file0...

RooFit v3.60 -- Developed by Wouter Verkerke and David Kirkby
                Copyright (C) 2000-2013 NIKHEF, University of California & Stanford University
                All rights reserved, please read http://roofit.sourceforge.net/license.txt

(TFile *) 0x1cd9ef0
root [1] .ls
TFile**         workspaces/SimpleNumberCounting/tut_combined_SimpleNumberCounting_model.root
 TFile*         workspaces/SimpleNumberCounting/tut_combined_SimpleNumberCounting_model.root
  KEY: RooWorkspace     combined;1      combined
  KEY: TProcessID       ProcessID0;1    60cbb5f8-97c7-11e7-a8d7-0859b8bcbeef
  KEY: TDirectoryFile   SR_hists;1      SR_hists
  KEY: RooStats::HistFactory::Measurement       SimpleNumberCounting;1  SimpleNumberCounting
root [2] combined->Print()

RooWorkspace(combined) combined contents
```

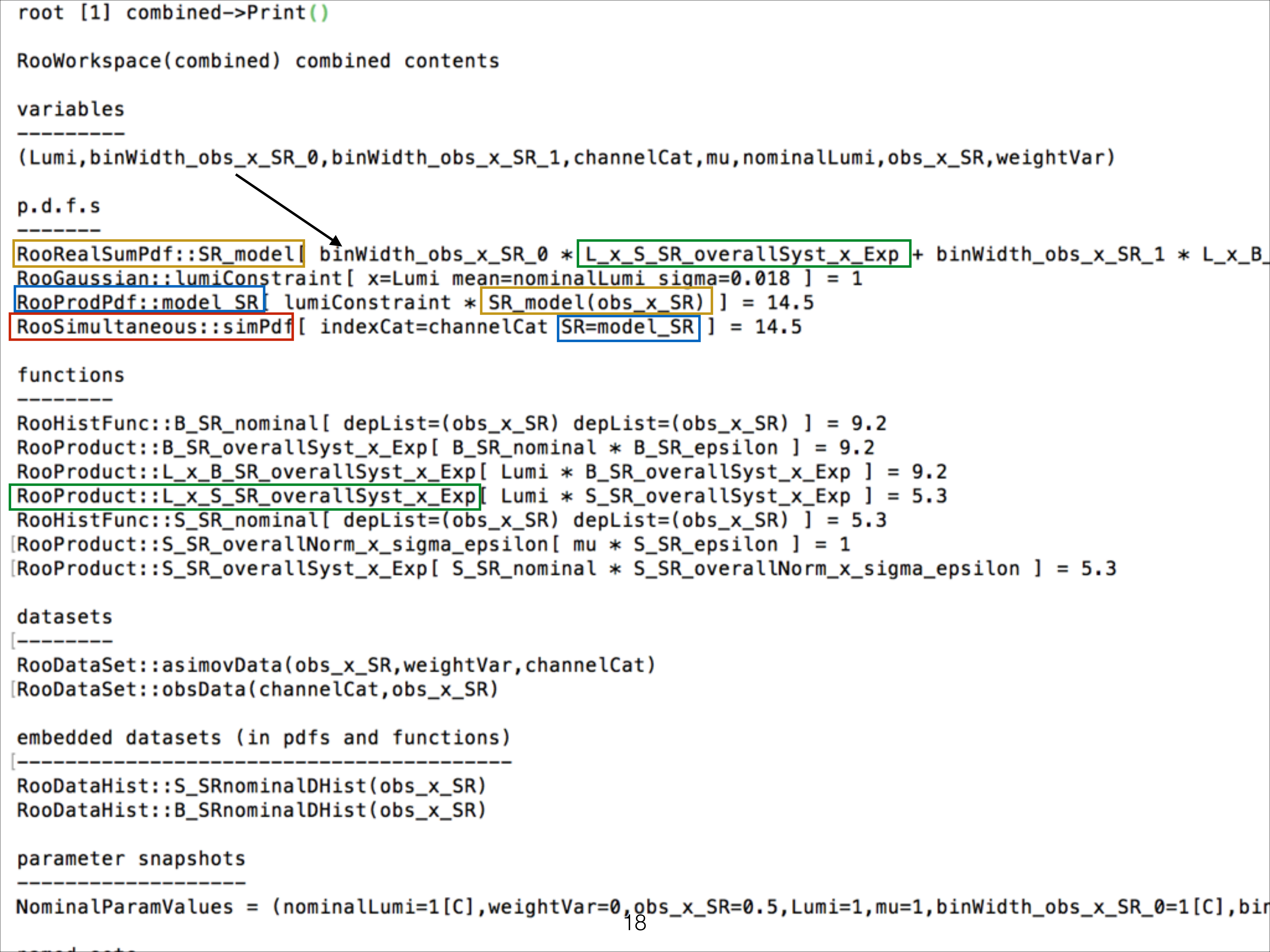


```
root [1] combined->Print()

RooWorkspace(combined) combined contents

variables
---------
(Lumi,binWidth_obs_x_SR_0,binWidth_obs_x_SR_1,channelCat,mu,nominalLumi,obs_x_SR,weightVar)

p.d.f.s
-------
RooRealSumPdf::SR_model[ binWidth_obs_x_SR_0 * L_x_S_SR_overallSyst_x_Exp + binWidth_obs_x_SR_1 * L_x_B_SR_overallSyst_x_Exp ] = 14.5
RooGaussian::lumiConstraint[ x=Lumi mean=nominalLumi sigma=0.018 ] = 1
RooProdPdf::model_SR[ lumiConstraint * SR_model(obs_x_SR) ] = 14.5
RooSimultaneous::simPdf[ indexCat=channelCat SR=model_SR ] = 14.5

functions
---------
RooHistFunc::B_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 9.2
RooProduct::B_SR_overallSyst_x_Exp[ B_SR_nominal * B_SR_epsilon ] = 9.2
RooProduct::L_x_B_SR_overallSyst_x_Exp[ Lumi * B_SR_overallSyst_x_Exp ] = 9.2
RooProduct::L_x_S_SR_overallSyst_x_Exp[ Lumi * S_SR_overallSyst_x_Exp ] = 5.3
RooHistFunc::S_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 5.3
RooProduct::S_SR_overallNorm_x_sigma_epsilon[ mu * S_SR_epsilon ] = 1
RooProduct::S_SR_overallSyst_x_Exp[ S_SR_nominal * S_SR_overallNorm_x_sigma_epsilon ] = 5.3

datasets
--------
RooDataSet::asimovData(obs_x_SR,weightVar,channelCat)
RooDataSet::obsData(channelCat,obs_x_SR)

embedded datasets (in pdfs and functions)
-----------------------------------------
RooDataHist::S_SRnominalDHist(obs_x_SR)
RooDataHist::B_SRnominalDHist(obs_x_SR)

parameter snapshots
-------------------
NominalParamValues = (nominalLumi=1[C],weightVar=0,obs_x_SR=0.5,Lumi=1,mu=1,binWidth_obs_x_SR_0=1[C],binWidth_obs_x_SR_1=1[C])

named sets
----------
ModelConfig_GlobalObservables:(nominalLumi)
ModelConfig_NuisParams:(Lumi)
ModelConfig_Observables:(obs_x_SR,weightVar,channelCat)
ModelConfig_POI:(mu)
globalObservables:(nominalLumi)
observables:(obs_x_SR,weightVar,channelCat)

generic objects
---------------
RooStats::ModelConfig::ModelConfig
```

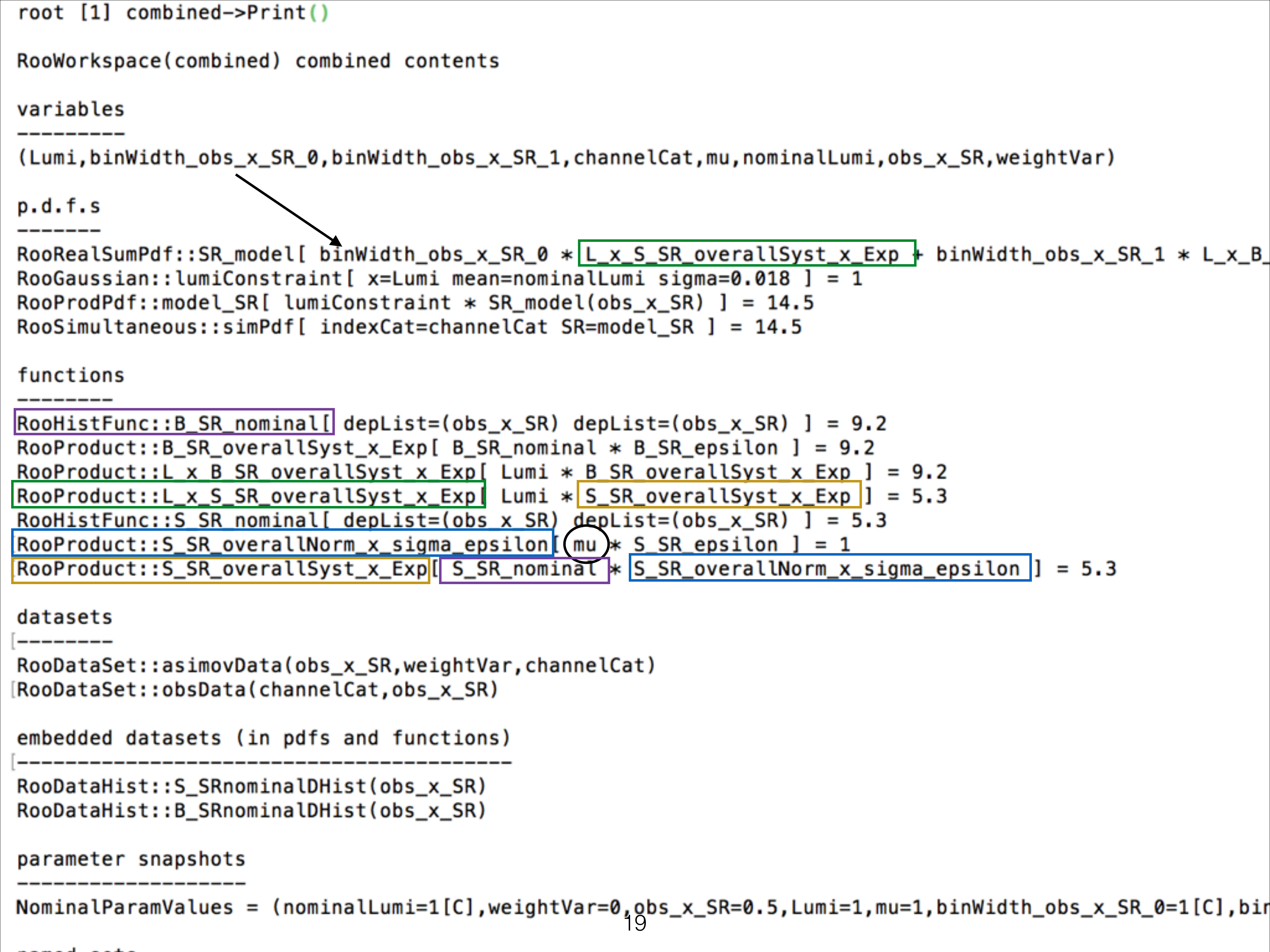




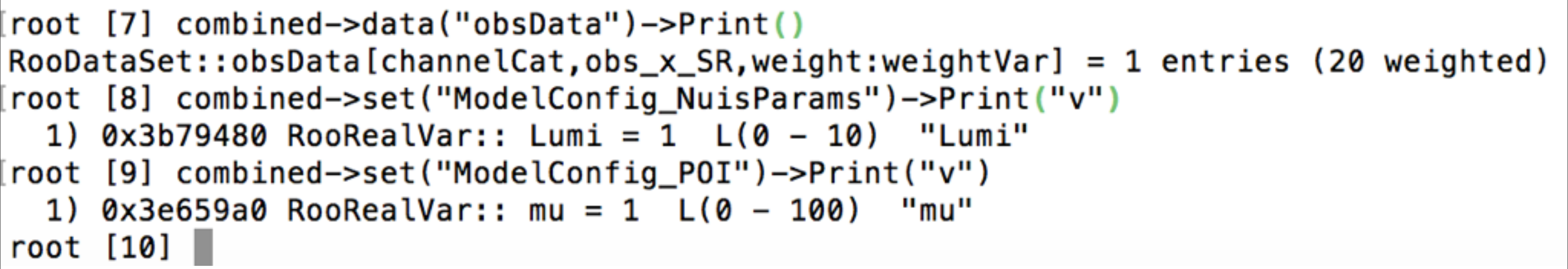
```
root [1] combined->Print()

RooWorkspace(combined) combined contents

variables
---------
(Lumi,binWidth_obs_x_SR_0,binWidth_obs_x_SR_1,channelCat,mu,nominalLumi,obs_x_SR,weightVar)

p.d.f.s
-------
RooRealSumPdf::SR_model[ binWidth_obs_x_SR_0 * L_x_S_SR_overallSyst_x_Exp + binWidth_obs_x_SR_1 * L_x_B_SR_overallSyst_x_Exp ] = 14.5
RooGaussian::lumiConstraint[ x=Lumi mean=nominalLumi sigma=0.018 ] = 1
RooProdPdf::model_SR[ lumiConstraint * SR_model(obs_x_SR) ] = 14.5
RooSimultaneous::simPdf[ indexCat=channelCat SR=model_SR ] = 14.5

functions
--------
RooHistFunc::B_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 9.2
RooProduct::B_SR_overallSyst_x_Exp[ B_SR_nominal * B_SR_epsilon ] = 9.2
RooProduct::L_x_B_SR_overallSyst_x_Exp[ Lumi * B_SR_overallSyst_x_Exp ] = 9.2
RooProduct::L_x_S_SR_overallSyst_x_Exp[ Lumi * S_SR_overallSyst_x_Exp ] = 5.3
RooHistFunc::S_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 5.3
RooProduct::S_SR_overallNorm_x_sigma_epsilon[ mu * S_SR_epsilon ] = 1
RooProduct::S_SR_overallSyst_x_Exp[ S_SR_nominal * S_SR_overallNorm_x_sigma_epsilon ] = 5.3

datasets
--------
RooDataSet::asimovData(obs_x_SR,weightVar,channelCat)
RooDataSet::obsData(channelCat,obs_x_SR)

embedded datasets (in pdfs and functions)
-----------------------------------------
RooDataHist::S_SRnominalDHist(obs_x_SR)
RooDataHist::B_SRnominalDHist(obs_x_SR)

parameter snapshots
-------------------
NominalParamValues = (nominalLumi=1[C],weightVar=0,obs_x_SR=0.5,Lumi=1,mu=1,binWidth_obs_x_SR_0=1[C],binWidth_obs_x_SR_1=1[C])

named sets
----------
ModelConfig_GlobalObservables:(nominalLumi)
ModelConfig_NuisParams:(Lumi)
ModelConfig_Observables:(obs_x_SR,weightVar,channelCat)
ModelConfig_POI:(mu)
globalObservables:(nominalLumi)
observables:(obs_x_SR,weightVar,channelCat)

generic objects
---------------
RooStats::ModelConfig::ModelConfig
```



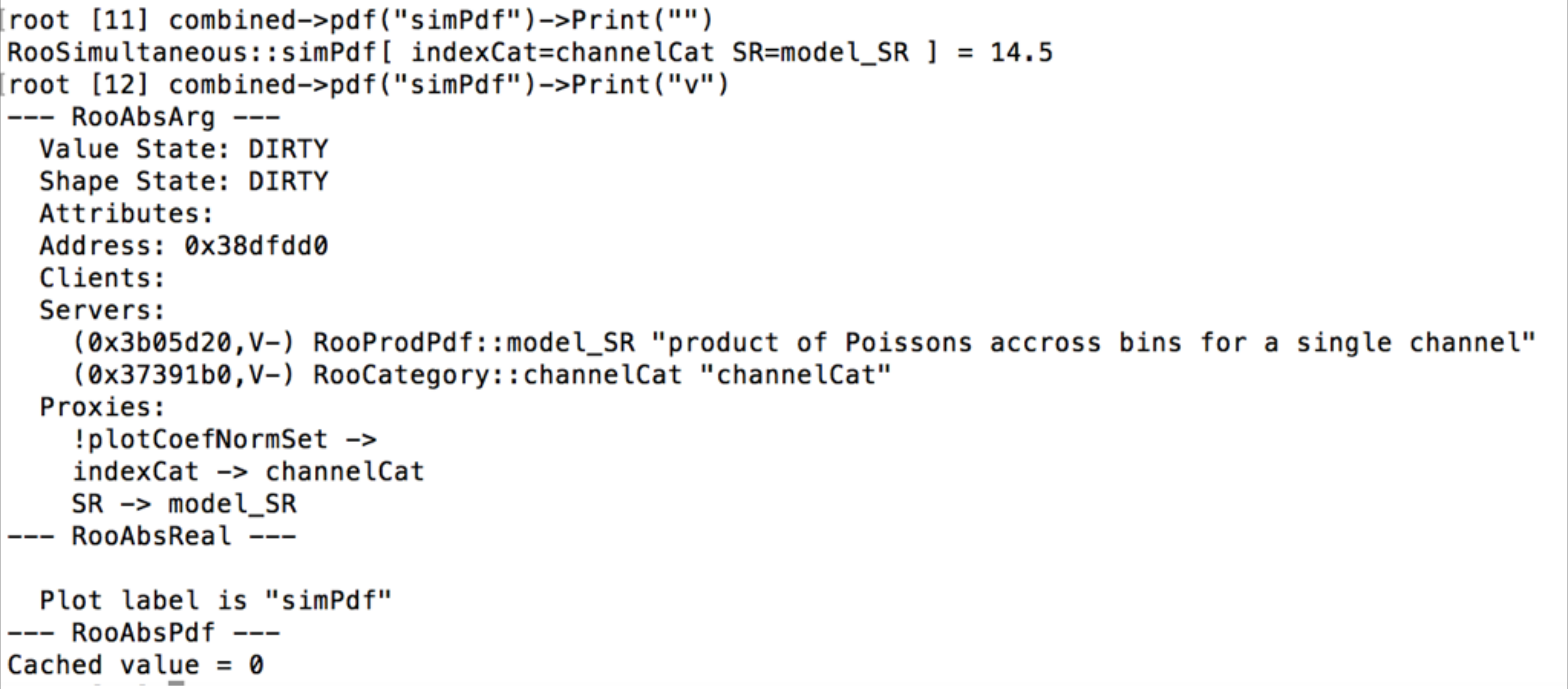
```
root [1] combined->Print()

RooWorkspace(combined) combined contents

variables
---------
(Lumi,binWidth_obs_x_SR_0,binWidth_obs_x_SR_1,channelCat,mu,nominalLumi,obs_x_SR,weightVar)

p.d.f.s
-------
RooRealSumPdf::SR_model[ binWidth_obs_x_SR_0 * L_x_S_SR_overallSyst_x_Exp + binWidth_obs_x_SR_1 * L_x_B_
RooGaussian::lumiConstraint[ x=Lumi mean=nominalLumi sigma=0.018 ] = 1
RooProdPdf::model_SR[ lumiConstraint * SR_model(obs_x_SR) ] = 14.5
RooSimultaneous::simPdf[ indexCat=channelCat SR=model_SR ] = 14.5

functions
--------
RooHistFunc::B_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 9.2
RooProduct::B_SR_overallSyst_x_Exp[ B_SR_nominal * B_SR_epsilon ] = 9.2
RooProduct::L_x_B_SR_overallSyst_x_Exp[ Lumi * B_SR_overallSyst_x_Exp ] = 9.2
RooProduct::L_x_S_SR_overallSyst_x_Exp[ Lumi * S_SR_overallSyst_x_Exp ] = 5.3
RooHistFunc::S_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 5.3
RooProduct::S_SR_overallNorm_x_sigma_epsilon[ mu * S_SR_epsilon ] = 1
RooProduct::S_SR_overallSyst_x_Exp[ S_SR_nominal * S_SR_overallNorm_x_sigma_epsilon ] = 5.3

datasets
--------
RooDataSet::asimovData(obs_x_SR,weightVar,channelCat)
RooDataSet::obsData(channelCat,obs_x_SR)

embedded datasets (in pdfs and functions)
-----------------------------------------
RooDataHist::S_SRnominalDHist(obs_x_SR)
RooDataHist::B_SRnominalDHist(obs_x_SR)

parameter snapshots
-------------------
NominalParamValues = (nominalLumi=1[C],weightVar=0,obs_x_SR=0.5,Lumi=1,mu=1,binWidth_obs_x_SR_0=1[C],bin

named sets
```



```
root [1] combined->Print()

RooWorkspace(combined) combined contents

variables
---------
(Lumi,binWidth_obs_x_SR_0,binWidth_obs_x_SR_1,channelCat,mu,nominalLumi,obs_x_SR,weightVar)

p.d.f.s
-------
RooRealSumPdf::SR_model[ binWidth_obs_x_SR_0 * L_x_S_SR_overallSyst_x_Exp + binWidth_obs_x_SR_1 * L_x_B_
RooGaussian::lumiConstraint[ x=Lumi mean=nominalLumi sigma=0.018 ] = 1
RooProdPdf::model_SR[ lumiConstraint * SR_model(obs_x_SR) ] = 14.5
RooSimultaneous::simPdf[ indexCat=channelCat SR=model_SR ] = 14.5

functions
--------
RooHistFunc::B_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 9.2
RooProduct::B_SR_overallSyst_x_Exp[ B_SR_nominal * B_SR_epsilon ] = 9.2
RooProduct::L_x_B_SR_overallSyst_x_Exp[ Lumi * B_SR_overallSyst_x_Exp ] = 9.2
RooProduct::L_x_S_SR_overallSyst_x_Exp[ Lumi * S_SR_overallSyst_x_Exp ] = 5.3
RooHistFunc::S_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 5.3
RooProduct::S_SR_overallNorm_x_sigma_epsilon[ mu * S_SR_epsilon ] = 1
RooProduct::S_SR_overallSyst_x_Exp[ S_SR_nominal * S_SR_overallNorm_x_sigma_epsilon ] = 5.3

datasets
--------
RooDataSet::asimovData(obs_x_SR,weightVar,channelCat)
RooDataSet::obsData(channelCat,obs_x_SR)

embedded datasets (in pdfs and functions)
-----------------------------------------
RooDataHist::S_SRnominalDHist(obs_x_SR)
RooDataHist::B_SRnominalDHist(obs_x_SR)

parameter snapshots
-------------------
NominalParamValues = (nominalLumi=1[C],weightVar=0,obs_x_SR=0.5,Lumi=1,mu=1,binWidth_obs_x_SR_0=1[C],bi
RooHistFunc::B_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 9.2
RooProduct::B_SR_overallSyst_x_Exp[ B_SR_nominal * B_SR_epsilon ] = 9.2
RooProduct::L_x_B_SR_overallSyst_x_Exp[ Lumi * B_SR_overallSyst_x_Exp ] = 9.2
RooProduct::L_x_S_SR_overallSyst_x_Exp[ Lumi * S_SR_overallSyst_x_Exp ] = 5.3
RooHistFunc::S_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 5.3
RooProduct::S_SR_overallNorm_x_sigma_epsilon[ mu * S_SR_epsilon ] = 1
RooProduct::S_SR_overallSyst_x_Exp[ S_SR_nominal * S_SR_overallNorm_x_sigma_epsilon ] = 5.

datasets
--------
RooDataSet::asimovData(obs_x_SR,weightVar,channelCat)
RooDataSet::obsData(channelCat,obs_x_SR)

embedded datasets (in pdfs and functions)
-----------------------------------------
RooDataHist::S_SRnominalDHist(obs_x_SR)
RooDataHist::B_SRnominalDHist(obs_x_SR)

parameter snapshots
-------------------
NominalParamValues = (nominalLumi=1[C],weightVar=0,obs_x_SR=0.5,Lumi=1,mu=1,binWidth_obs_x_

named sets
----------
ModelConfig_GlobalObservables:(nominalLumi)
ModelConfig_NuisParams:(Lumi)
ModelConfig_Observables:(obs_x_SR,weightVar,channelCat)
ModelConfig_POI:(mu)
globalObservables:(nominalLumi)
observables:(obs_x_SR,weightVar,channelCat)

generic objects
---------------
RooStats::ModelConfig::ModelConfig
```



```
[root [7] combined->data("obsData")->Print()
RooDataSet::obsData[channelCat,obs_x_SR,weight:weightVar] = 1 entries (20 weighted)
[root [8] combined->set("ModelConfig_NuisParams")->Print("v")
  1) 0x3b79480 RooRealVar:: Lumi = 1  L(0 - 10)  "Lumi"
[root [9] combined->set("ModelConfig_POI")->Print("v")
  1) 0x3e659a0 RooRealVar:: mu = 1  L(0 - 100)  "mu"
root [10] 
```



```
[root [11] combined->pdf("simPdf")->Print("")
RooSimultaneous::simPdf[ indexCat=channelCat SR=model_SR ] = 14.5
[root [12] combined->pdf("simPdf")->Print("v")
--- RooAbsArg ---
  Value State: DIRTY
  Shape State: DIRTY
  Attributes:
  Address: 0x38dfdd0
  Clients:
  Servers:
    (0x3b05d20,V-) RooProdPdf::model_SR "product of Poissons accross bins for a single channel"
    (0x37391b0,V-) RooCategory::channelCat "channelCat"
  Proxies:
    !plotCoefNormSet ->
    indexCat -> channelCat
    SR -> model_SR
--- RooAbsReal ---

  Plot label is "simPdf"
--- RooAbsPdf ---
Cached value = 0
```

Flickr: morrosv7

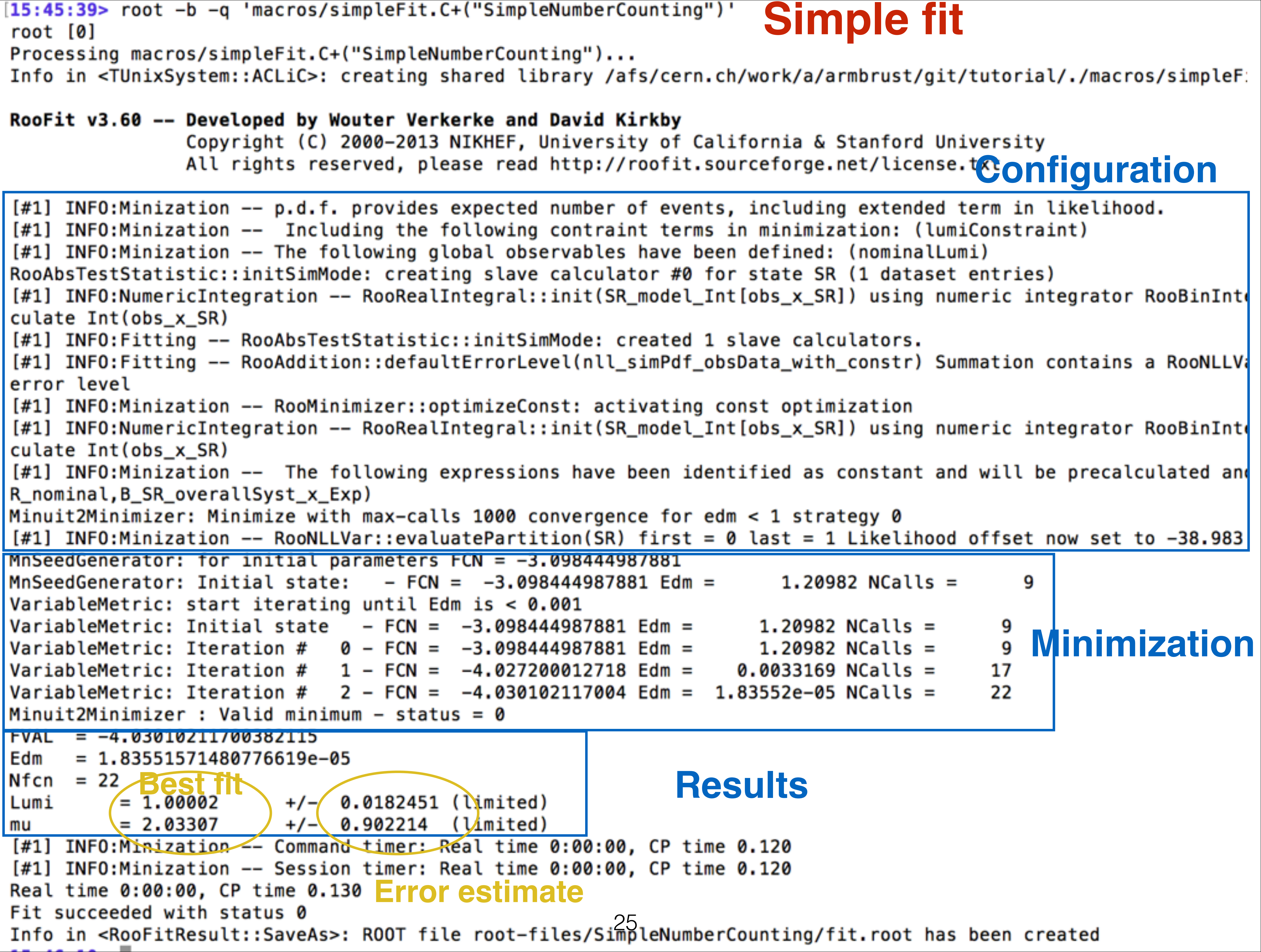
Practical example: Simple fit

- Exercise:
  - Look through macros/simpleFit.C
  - Run it on the workspace we just made:
    - **> root -b -q ‘macros/simpleFit.C(“SimpleNumberCounting”)’**
  - Examine output
- simpleFit.C will run a single unconditional fit of the likelihood, and print the maximum likelihood estimators μ hat and θ hat: $\mathcal{L}(\hat{\mu}, \hat{\theta})$

Practical example: Simple fit

```
void simpleFit(string version, int strategy=0)
{
  //Open the file, grab the workspace
  TFile* f = new TFile(Form("workspaces/%s/tut_combined_%s_model.root",version.c_str(),version.c_str()));
  RooWorkspace* w = (RooWorkspace*)f->Get("combined");

  //Grab the ModelConfig and the data
  ModelConfig* mc = (ModelConfig*)w->obj("ModelConfig");
  RooDataSet* data = (RooDataSet*)w->data("obsData");


  //Configure MINUIT
  ROOT::Math::MinimizerOptions::SetDefaultMinimizer("Minuit2");
  ROOT::Math::MinimizerOptions::SetDefaultStrategy(strategy);
  ROOT::Math::MinimizerOptions::SetDefaultPrintLevel(1);

  //Put the NPs and POIs into one set to send to the NLL
  RooArgSet params(*mc->GetNuisanceParameters(),*mc->GetParametersOfInterest());

  //Build the NLL
  RooNLLVar* nll = (RooNLLVar*)mc->GetPdf()->createNLL(*data, Constrain(params),
                                                       GlobalObservables(*mc->GetGlobalObservables()),
                                                       RooFit::Offset(1));

  //Do the minimization and save the RooFitResult

  //Definition in macros/minimize.C:
  //RooFitResult* minimize(RooAbsReal* fcn, bool save=0, int retry_mode=3, int* ret_status=NULL)
  RooFitResult* result = minimize(nll, true, 0);

  //Save to file
  system(Form("mkdir -vp root-files/%s",version.c_str()));
  result->SetName("result");
  result->SaveAs(Form("root-files/%s/fit.root",version.c_str()));
}
```

Simple fit



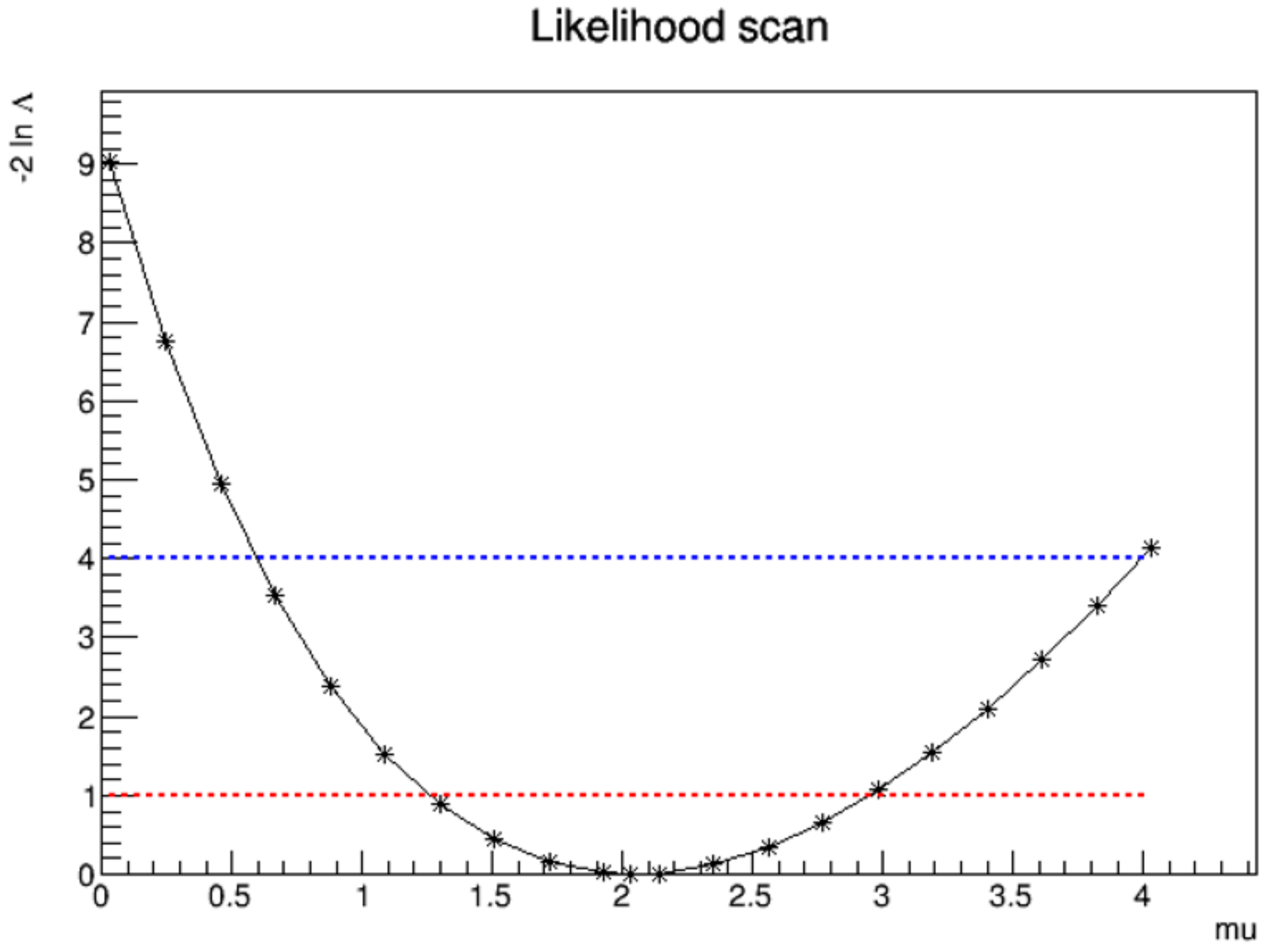
```
[15:45:39> root -b -q 'macros/simpleFit.C+("SimpleNumberCounting")'
root [0]
Processing macros/simpleFit.C+("SimpleNumberCounting")...
Info in <TUnixSystem::ACLiC>: creating shared library /afs/cern.ch/work/a/armbrust/git/tutorial/./macros/simpleF

RooFit v3.60 -- Developed by Wouter Verkerke and David Kirkby
                Copyright (C) 2000-2013 NIKHEF, University of California & Stanford University
                All rights reserved, please read http://roofit.sourceforge.net/license.txt

[#1] INFO:Minization -- p.d.f. provides expected number of events, including extended term in likelihood.
[#1] INFO:Minization --  Including the following contraint terms in minimization: (lumiConstraint)
[#1] INFO:Minization -- The following global observables have been defined: (nominalLumi)
RooAbsTestStatistic::initSimMode: creating slave calculator #0 for state SR (1 dataset entries)
[#1] INFO:NumericIntegration -- RooRealIntegral::init(SR_model_Int[obs_x_SR]) using numeric integrator RooBinInte
culate Int(obs_x_SR)
[#1] INFO:Fitting -- RooAbsTestStatistic::initSimMode: created 1 slave calculators.
[#1] INFO:Fitting -- RooAddition::defaultErrorLevel(nll_simPdf_obsData_with_constr) Summation contains a RooNLLVa
error level
[#1] INFO:Minization -- RooMinimizer::optimizeConst: activating const optimization
[#1] INFO:NumericIntegration -- RooRealIntegral::init(SR_model_Int[obs_x_SR]) using numeric integrator RooBinInte
culate Int(obs_x_SR)
[#1] INFO:Minization --  The following expressions have been identified as constant and will be precalculated and
R_nominal,B_SR_overallSyst_x_Exp)
Minuit2Minimizer: Minimize with max-calls 1000 convergence for edm < 1 strategy 0
[#1] INFO:Minization -- RooNLLVar::evaluatePartition(SR) first = 0 last = 1 Likelihood offset now set to -38.983
MnSeedGenerator: for initial parameters FCN = -3.098444987881
MnSeedGenerator: Initial state:   - FCN =  -3.098444987881 Edm =      1.20982 NCalls =      9
VariableMetric: start iterating until Edm is < 0.001
VariableMetric: Initial state   - FCN =  -3.098444987881 Edm =      1.20982 NCalls =      9
VariableMetric: Iteration #   0 - FCN =  -3.098444987881 Edm =      1.20982 NCalls =      9
VariableMetric: Iteration #   1 - FCN =  -4.027200012718 Edm =    0.0033169 NCalls =     17
VariableMetric: Iteration #   2 - FCN =  -4.030102117004 Edm =  1.83552e-05 NCalls =     22
Minuit2Minimizer : Valid minimum - status = 0
FVAL  = -4.03010211700382115
Edm   = 1.83551571480776619e-05
Nfcn  = 22
Lumi       = 1.00002       +/-   0.0182451  (limited)
mu         = 2.03307       +/-   0.902214   (limited)
[#1] INFO:Minization -- Command timer: Real time 0:00:00, CP time 0.120
[#1] INFO:Minization -- Session timer: Real time 0:00:00, CP time 0.120
Real time 0:00:00, CP time 0.130
Fit succeeded with status 0
Info in <RooFitResult::SaveAs>: ROOT file root-files/SimpleNumberCounting/fit.root has been created
```

Likelihood scan

- Exercise:
  - Look through macros/scanLikelihood.C
  - Run it: **> root ‘macros/scanLikelihood.C(“SimpleNumberCounting”,”mu”,2,20)’**
  - Run over the Lumi nuisance parameter:
    - **> root ‘macros/scanLikelihood.C(“SimpleNumberCounting”,”Lumi”,0.04,20)’**
  - Examine plot
- scanLikelihood.C will compute the test statistic t_α at various points in the specified parameter α (not necessarily μ)

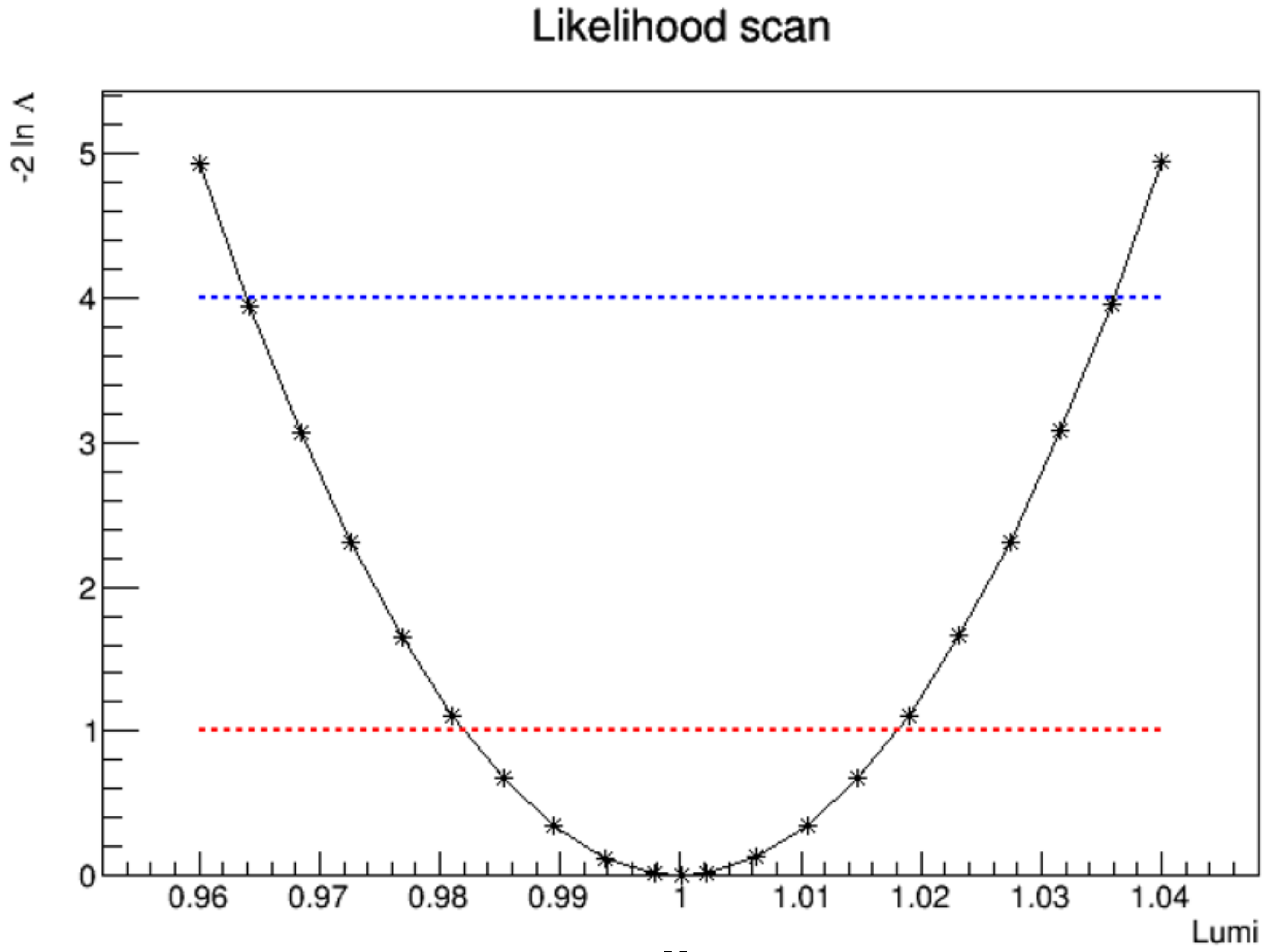
Likelihood scan

```
16:06:27> root 'macros/scanLikelihood.C+("SimpleNumberCounting","mu",2,20)'
root [0]
Processing macros/scanLikelihood.C+("SimpleNumberCounting","mu",2,20)...
```



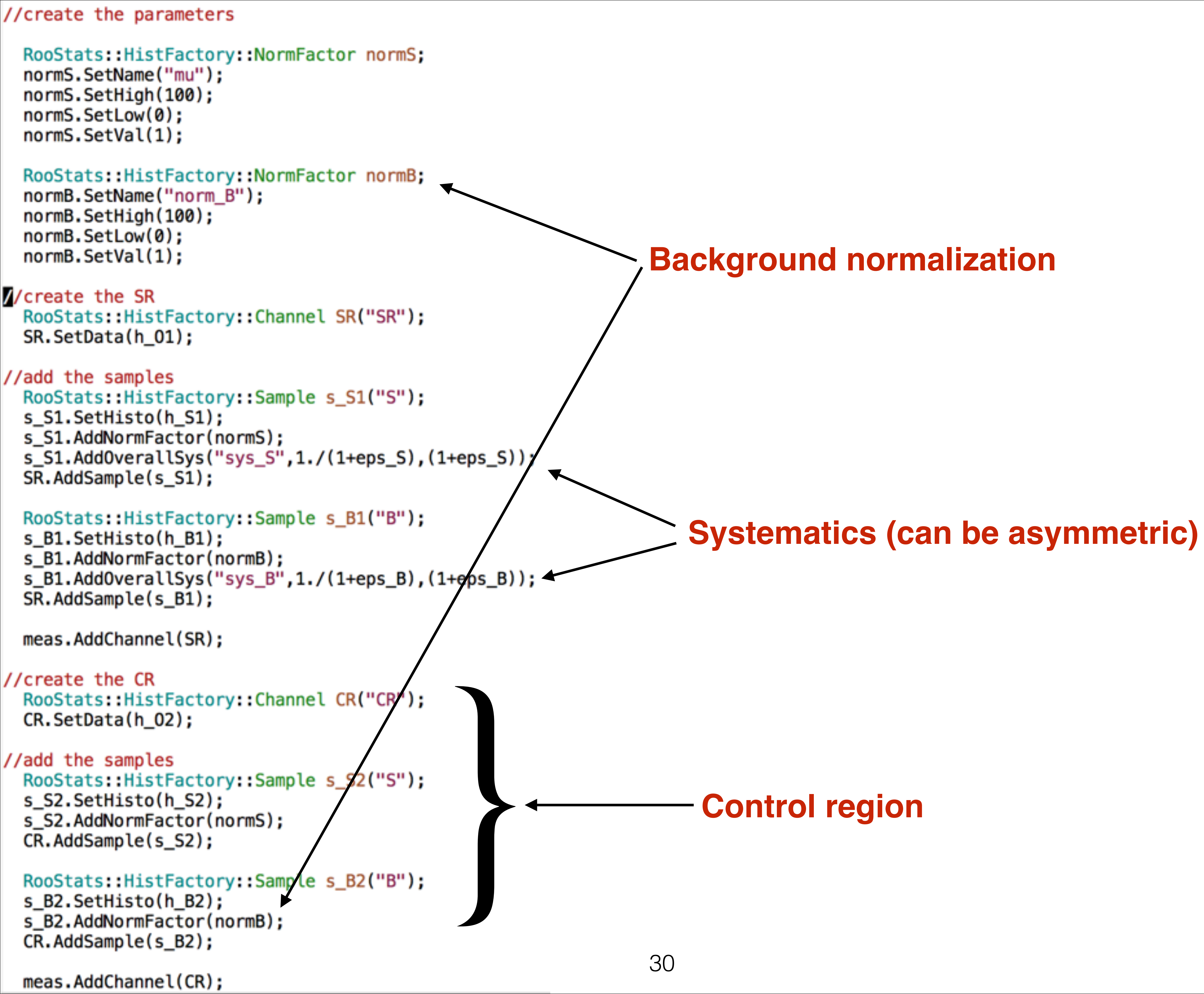
Likelihood scan

```
[16:13:06> root 'macros/scanLikelihood.C+("SimpleNumberCounting","Lumi",0.04,20)'
root [0]
Processing macros/scanLikelihood.C+("SimpleNumberCounting","Lumi",0.04,20)...
```



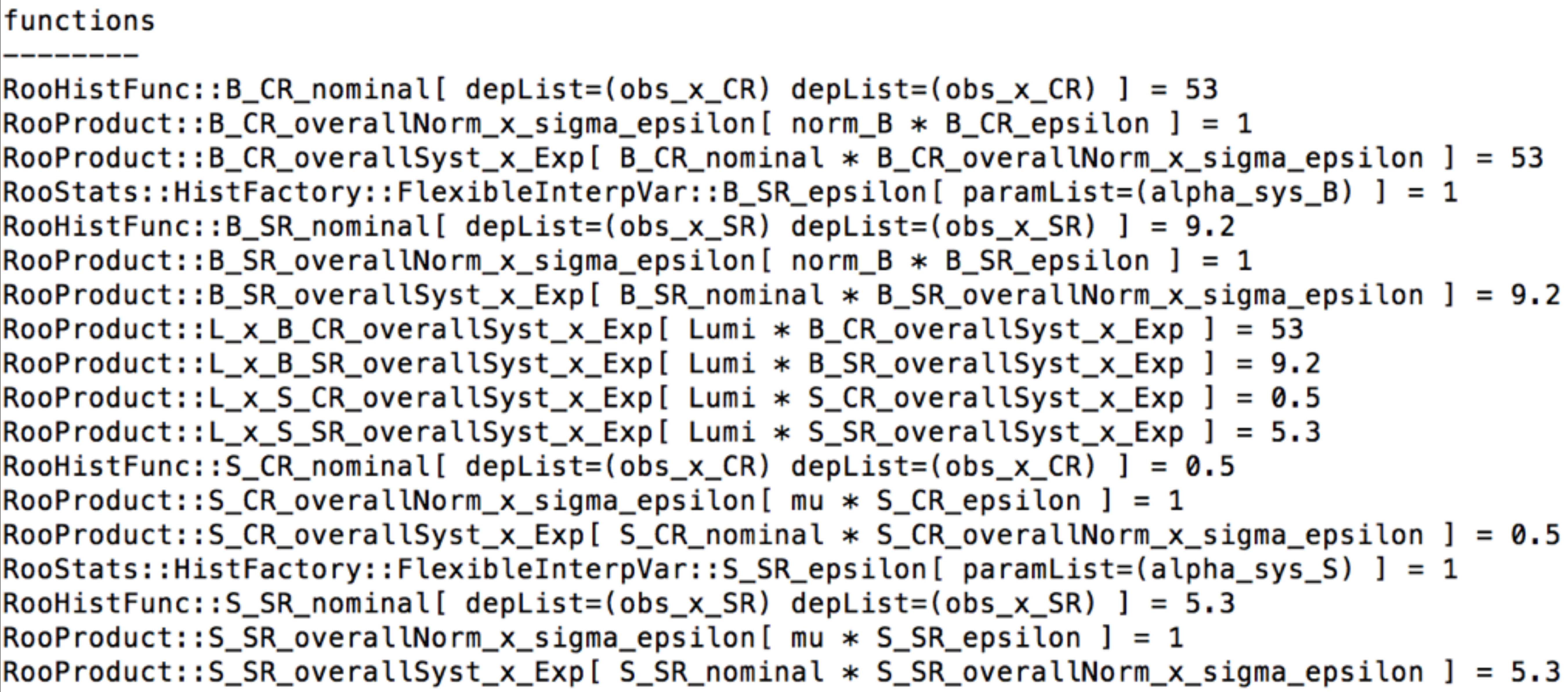
Practical example: On/Off problem, with systematics

- Exercise:
    - Look through ex/OnOff.C
    - Run it: **> root -b -q ex/OnOff.C**
    - Open the workspace and look through
    - Make likelihood scans of all parameters
- OnOff.C adds three additional elements: a background control-region, a corresponding background normalization parameter, and systematic uncertainties
- $L = P(N_{SR} \mid \mu * S_{SR}(\theta) + \mu_B * B_{SR}(\theta)) * P(N_{CR} \mid \mu * S_{CR}(\theta) + \mu_B * B_{CR}(\theta)) * G(\theta, 0, 1)$

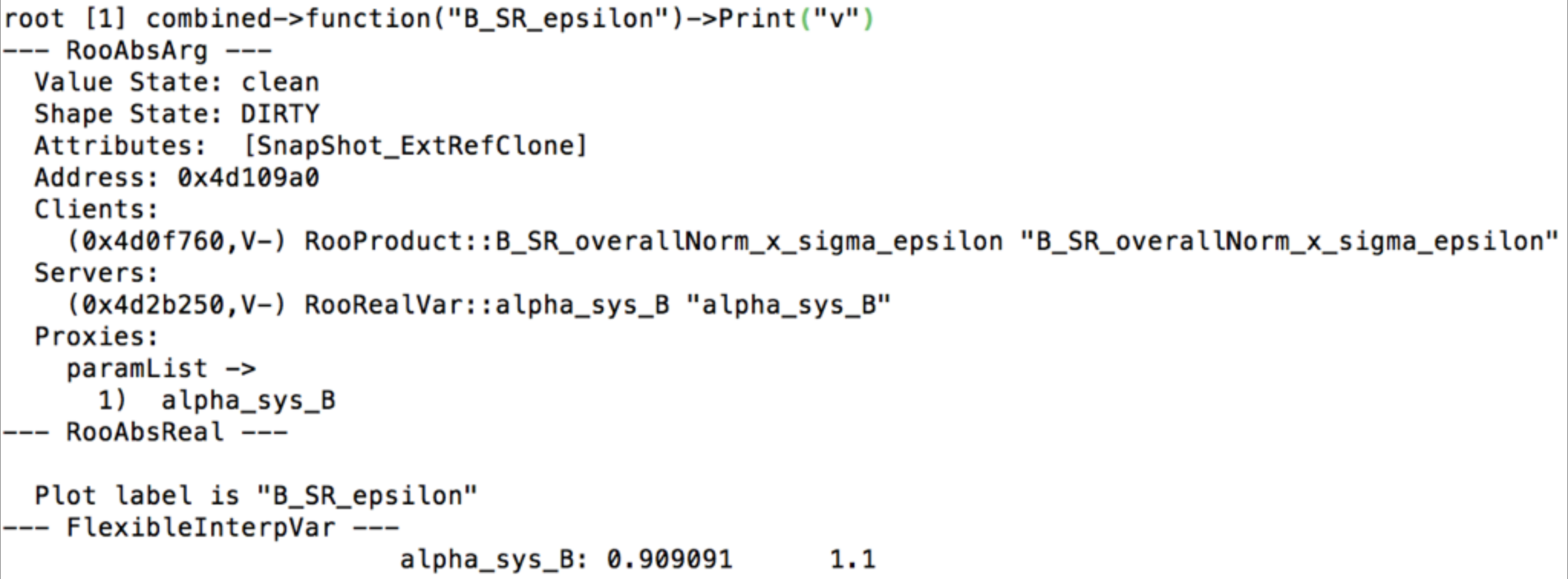


```
//create the parameters

  RooStats::HistFactory::NormFactor normS;
  normS.SetName("mu");
  normS.SetHigh(100);
  normS.SetLow(0);
  normS.SetVal(1);

  RooStats::HistFactory::NormFactor normB;
  normB.SetName("norm_B");
  normB.SetHigh(100);
  normB.SetLow(0);
  normB.SetVal(1);

//create the SR
  RooStats::HistFactory::Channel SR("SR");
  SR.SetData(h_01);

//add the samples
  RooStats::HistFactory::Sample s_S1("S");
  s_S1.SetHisto(h_S1);
  s_S1.AddNormFactor(normS);
  s_S1.AddOverallSys("sys_S",1./(1+eps_S),(1+eps_S));
  SR.AddSample(s_S1);

  RooStats::HistFactory::Sample s_B1("B");
  s_B1.SetHisto(h_B1);
  s_B1.AddNormFactor(normB);
  s_B1.AddOverallSys("sys_B",1./(1+eps_B),(1+eps_B));
  SR.AddSample(s_B1);

  meas.AddChannel(SR);

//create the CR
  RooStats::HistFactory::Channel CR("CR");
  CR.SetData(h_02);

//add the samples
  RooStats::HistFactory::Sample s_S2("S");
  s_S2.SetHisto(h_S2);
  s_S2.AddNormFactor(normS);
  CR.AddSample(s_S2);

  RooStats::HistFactory::Sample s_B2("B");
  s_B2.SetHisto(h_B2);
  s_B2.AddNormFactor(normB);
  CR.AddSample(s_B2);

  meas.AddChannel(CR);
```

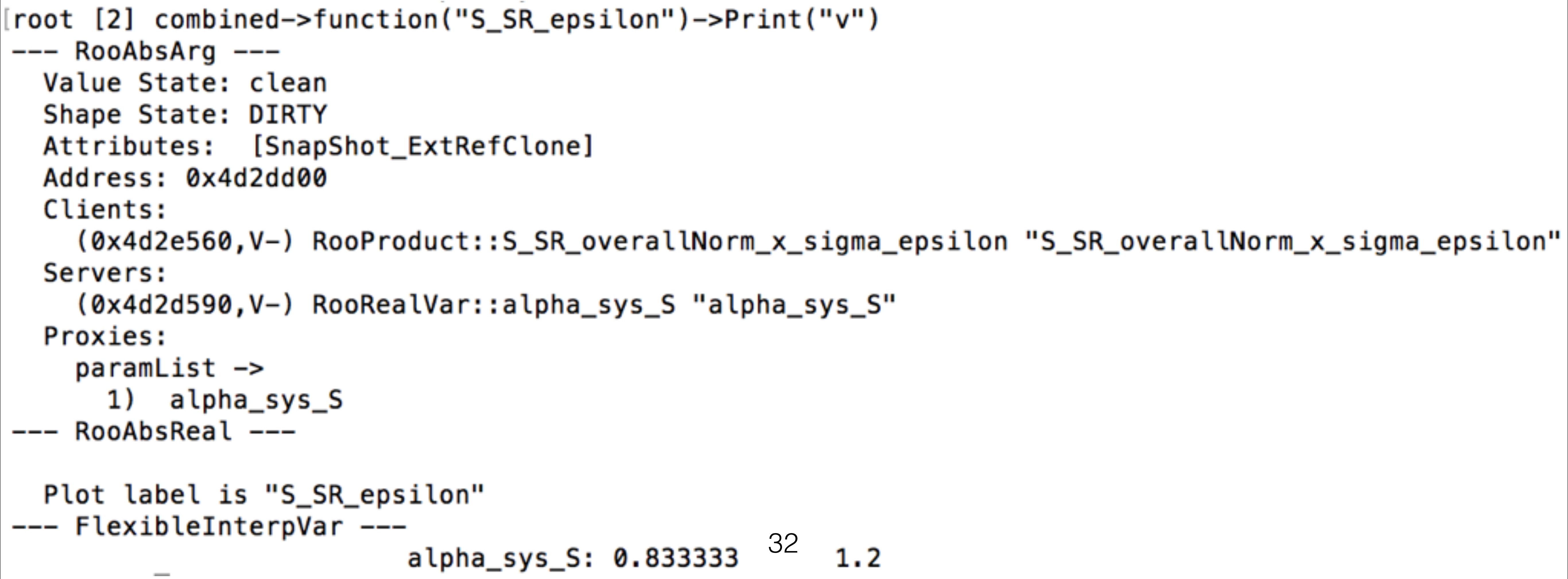
Practical example: On/Off problem, with systematics



```
functions
---------
RooHistFunc::B_CR_nominal[ depList=(obs_x_CR) depList=(obs_x_CR) ] = 53
RooProduct::B_CR_overallNorm_x_sigma_epsilon[ norm_B * B_CR_epsilon ] = 1
RooProduct::B_CR_overallSyst_x_Exp[ B_CR_nominal * B_CR_overallNorm_x_sigma_epsilon ] = 53
RooStats::HistFactory::FlexibleInterpVar::B_SR_epsilon[ paramList=(alpha_sys_B) ] = 1
RooHistFunc::B_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 9.2
RooProduct::B_SR_overallNorm_x_sigma_epsilon[ norm_B * B_SR_epsilon ] = 1
RooProduct::B_SR_overallSyst_x_Exp[ B_SR_nominal * B_SR_overallNorm_x_sigma_epsilon ] = 9.2
RooProduct::L_x_B_CR_overallSyst_x_Exp[ Lumi * B_CR_overallSyst_x_Exp ] = 53
RooProduct::L_x_B_SR_overallSyst_x_Exp[ Lumi * B_SR_overallSyst_x_Exp ] = 9.2
RooProduct::L_x_S_CR_overallSyst_x_Exp[ Lumi * S_CR_overallSyst_x_Exp ] = 0.5
RooProduct::L_x_S_SR_overallSyst_x_Exp[ Lumi * S_SR_overallSyst_x_Exp ] = 5.3
RooHistFunc::S_CR_nominal[ depList=(obs_x_CR) depList=(obs_x_CR) ] = 0.5
RooProduct::S_CR_overallNorm_x_sigma_epsilon[ mu * S_CR_epsilon ] = 1
RooProduct::S_CR_overallSyst_x_Exp[ S_CR_nominal * S_CR_overallNorm_x_sigma_epsilon ] = 0.5
RooStats::HistFactory::FlexibleInterpVar::S_SR_epsilon[ paramList=(alpha_sys_S) ] = 1
RooHistFunc::S_SR_nominal[ depList=(obs_x_SR) depList=(obs_x_SR) ] = 5.3
RooProduct::S_SR_overallNorm_x_sigma_epsilon[ mu * S_SR_epsilon ] = 1
RooProduct::S_SR_overallSyst_x_Exp[ S_SR_nominal * S_SR_overallNorm_x_sigma_epsilon ] = 5.3
```

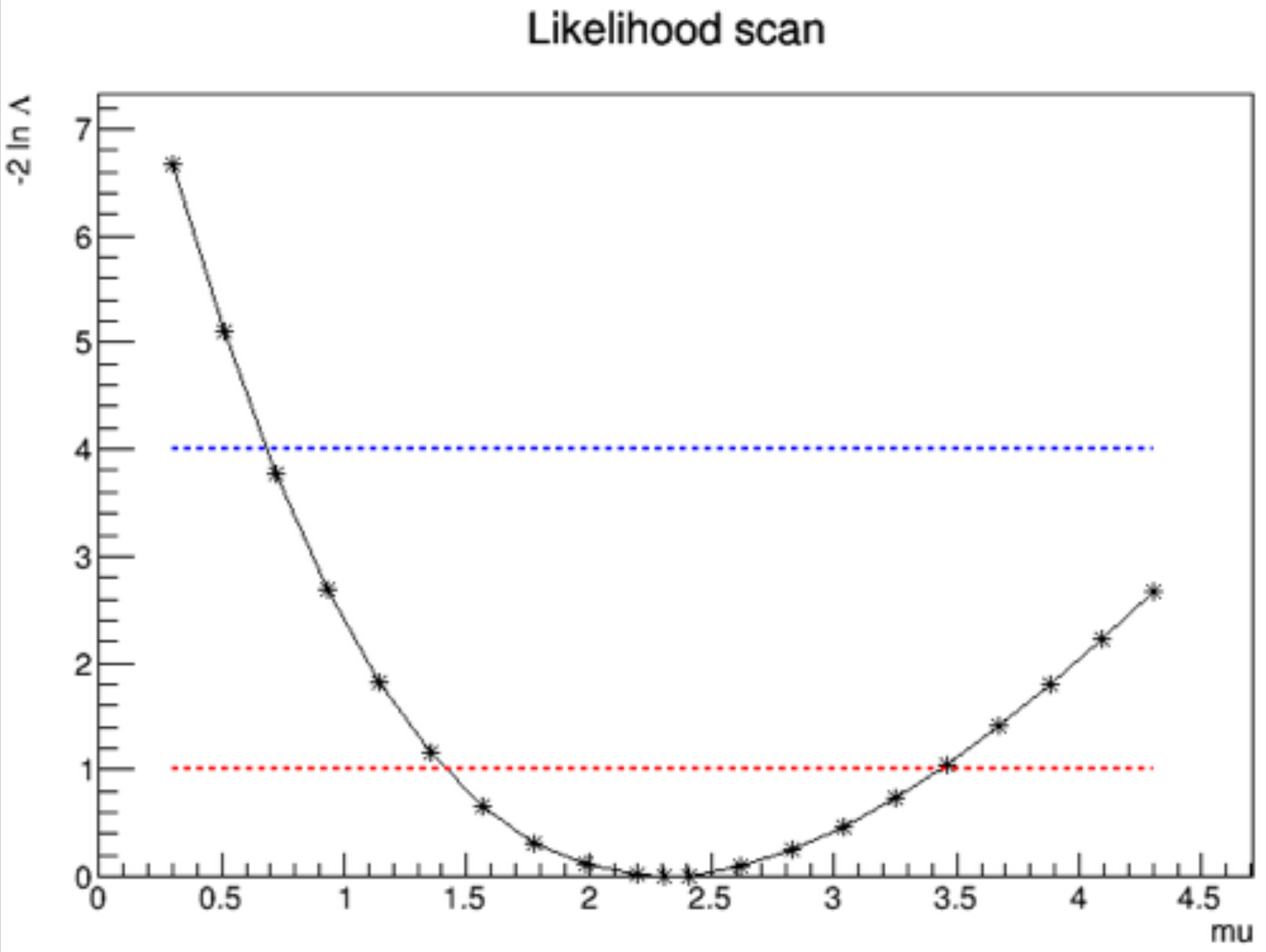


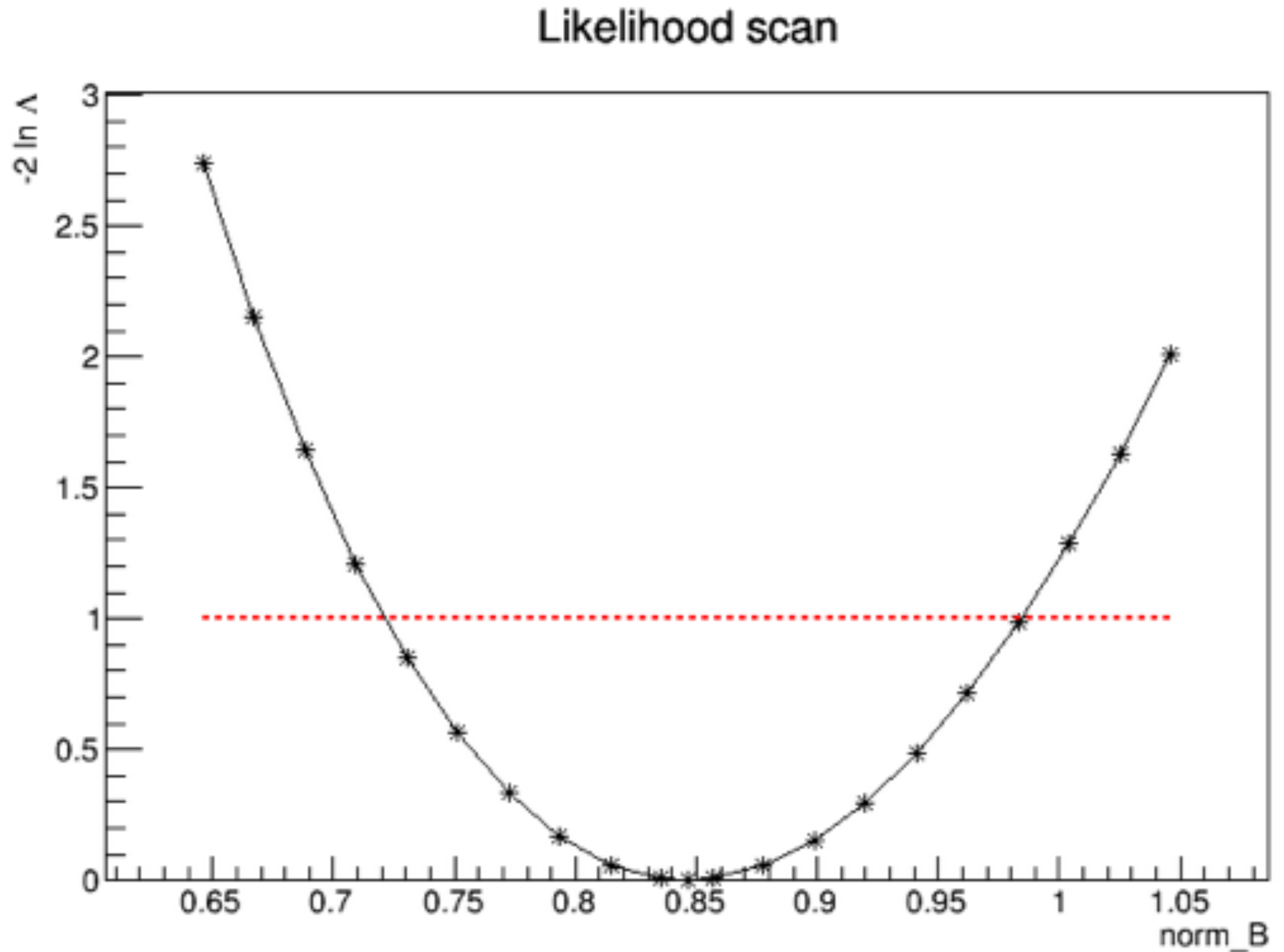
```
root [1] combined->function("B_SR_epsilon")->Print("v")
--- RooAbsArg ---
  Value State: clean
  Shape State: DIRTY
  Attributes:  [SnapShot_ExtRefClone]
  Address: 0x4d109a0
  Clients:
    (0x4d0f760,V-) RooProduct::B_SR_overallNorm_x_sigma_epsilon "B_SR_overallNorm_x_sigma_epsilon"
  Servers:
    (0x4d2b250,V-) RooRealVar::alpha_sys_B "alpha_sys_B"
  Proxies:
    paramList ->
      1)  alpha_sys_B
--- RooAbsReal ---

  Plot label is "B_SR_epsilon"
--- FlexibleInterpVar ---
                         alpha_sys_B: 0.909091      1.1
```

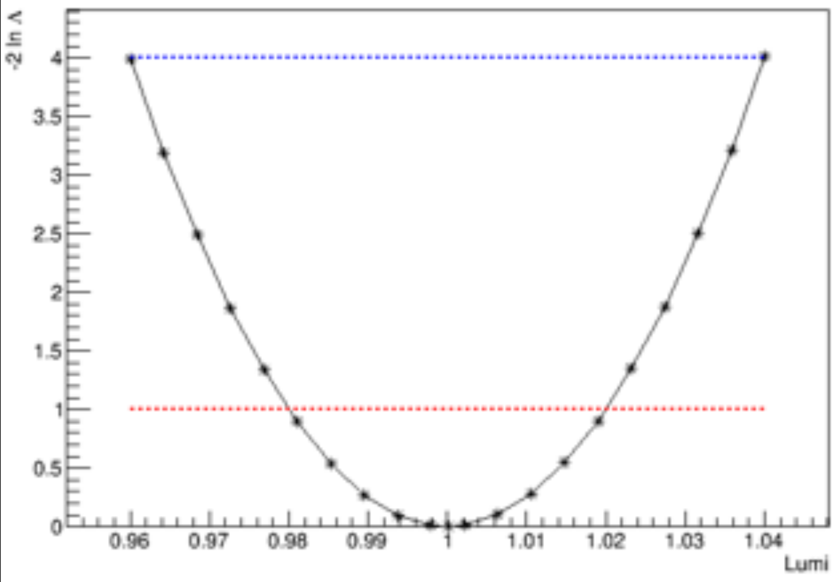


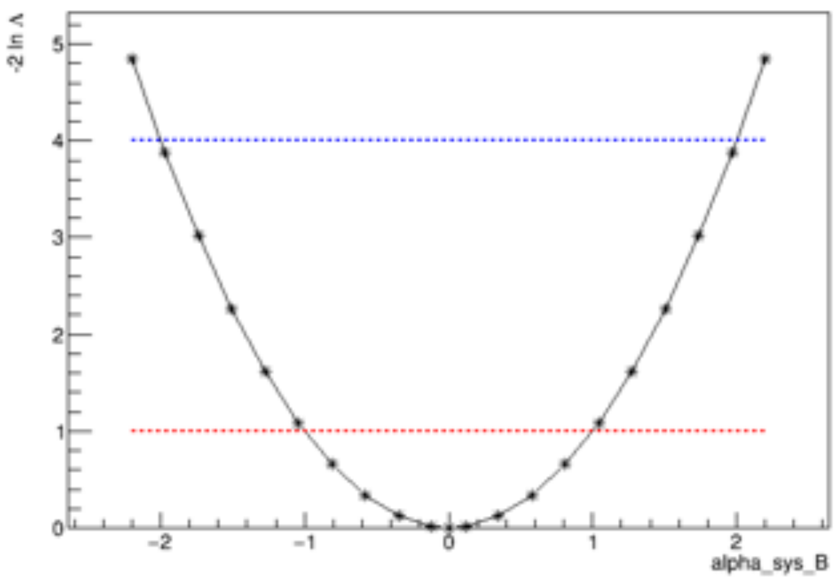
```
root [2] combined->function("S_SR_epsilon")->Print("v")
--- RooAbsArg ---
  Value State: clean
  Shape State: DIRTY
  Attributes:  [SnapShot_ExtRefClone]
  Address: 0x4d2dd00
  Clients:
    (0x4d2e560,V-) RooProduct::S_SR_overallNorm_x_sigma_epsilon "S_SR_overallNorm_x_sigma_epsilon"
  Servers:
    (0x4d2d590,V-) RooRealVar::alpha_sys_S "alpha_sys_S"
  Proxies:
    paramList ->
      1)  alpha_sys_S
--- RooAbsReal ---

  Plot label is "S_SR_epsilon"
--- FlexibleInterpVar ---
                         alpha_sys_S: 0.833333      1.2
```

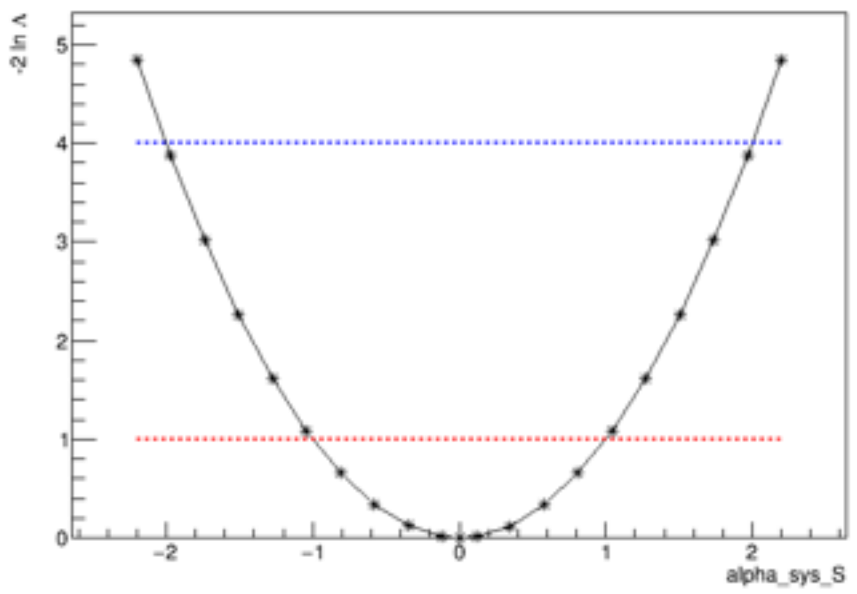
```
Lumi        = 1.00005       +/-  0.0194965 (limited)
alpha_sys_B        = -0.000109141         +/-  0.969303  (limited)
alpha_sys_S        = -0.000202487         +/-  0.941561  (limited)
mu          = 2.3042        +/-  0.96781   (limited)
norm_B      = 0.846142      +/-  0.131918  (limited)
```





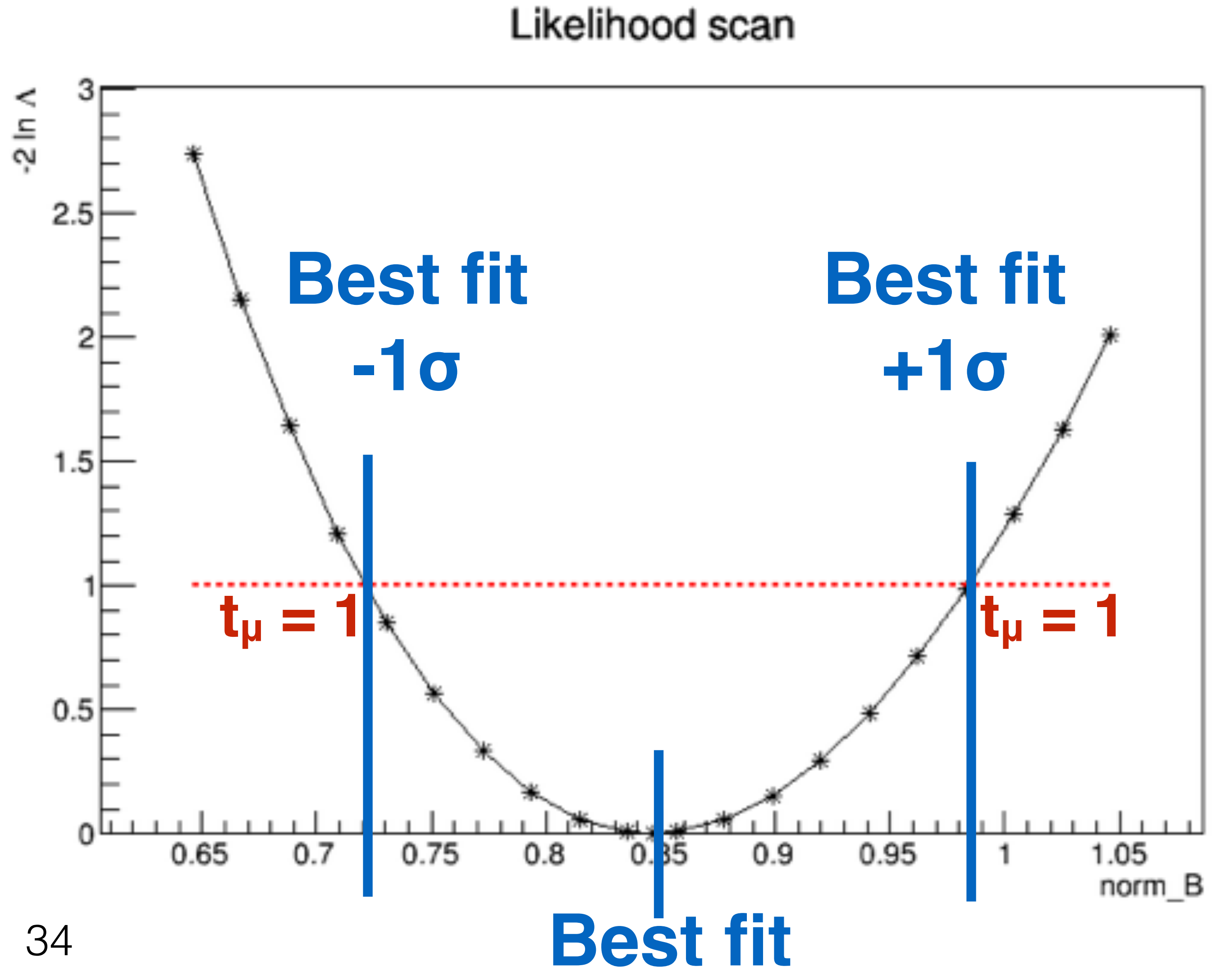






Minos uncertainties

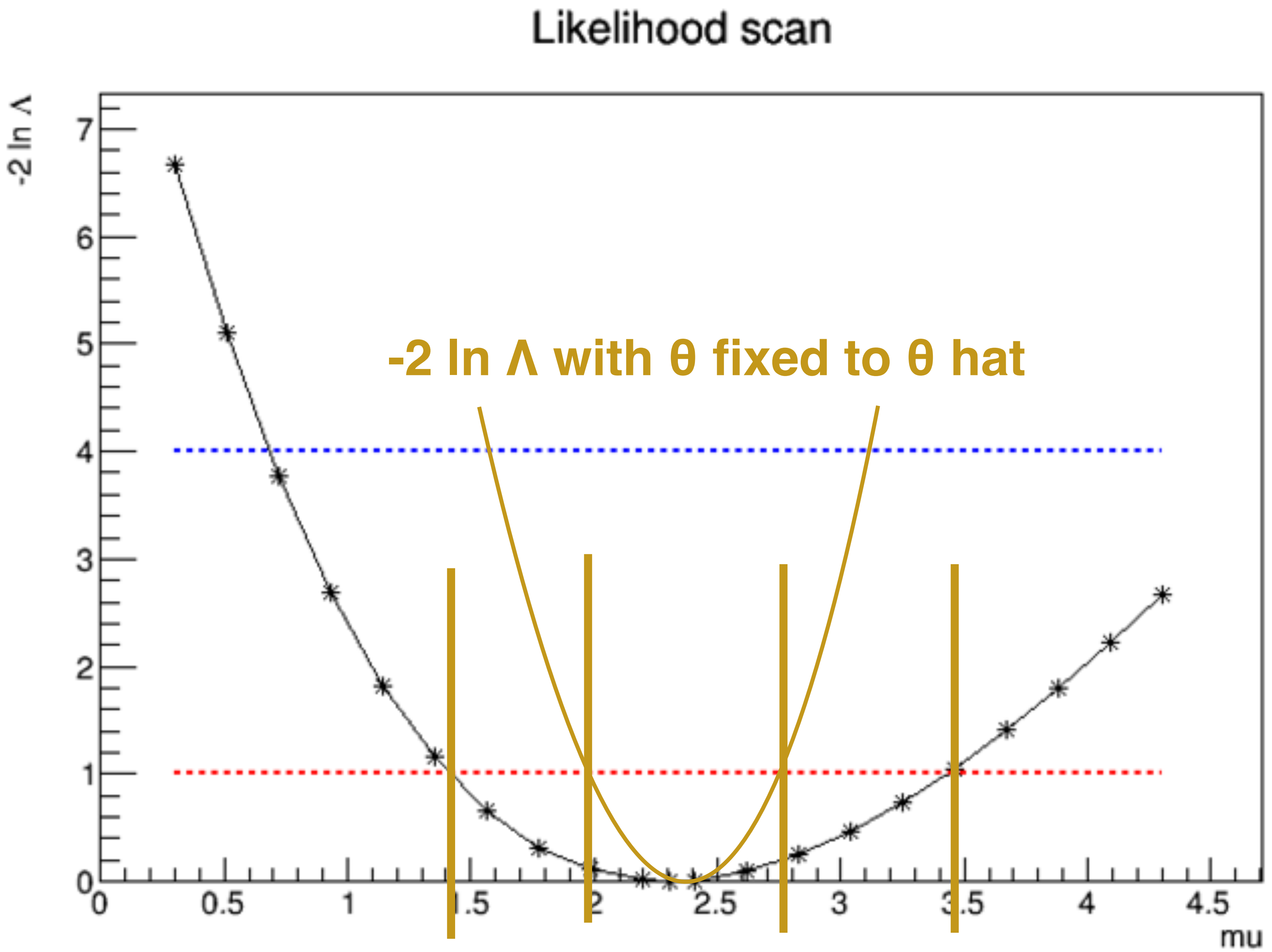
- Uncertainties shown in the MINUIT printout are approximate hessian errors
- These are computed from derivatives using finite differences around the minimum of the likelihood, and in general depend on the minimization path that MINUIT takes
- They are generally lower than the true uncertainty on the parameter, and not very reliable without some validation
- Recall: The 1σ point for a parameter μ corresponds to the value of μ when $t_\mu = 1$
- A "**Minos**" algorithm performs a likelihood scan to numerically find the crossing $t_\mu = 1$
- The helper function macros/findSigma.C



Error breakdowns

- The uncertainty σ can be broken down into quadrature components based on contributions from individual fit parameters, and therefore individual sources or groups of uncertainties
- The uncertainty σ computed when one or more nuisance parameters are fixed to their best fit value will be smaller than the total uncertainty

- $\sigma^2_{tot} = \sigma^2_{data} + \Sigma_{sys}\, \sigma^2_{sys}$
- Each component σ_{sys} can be computed by finding σtot, then the σ having fixed “sys”, and taking the quadrature difference:
- $\sigma^2_{sys} = \sigma^2_{tot} - \sigma^2(\theta_{sys}\text{-fixed})$

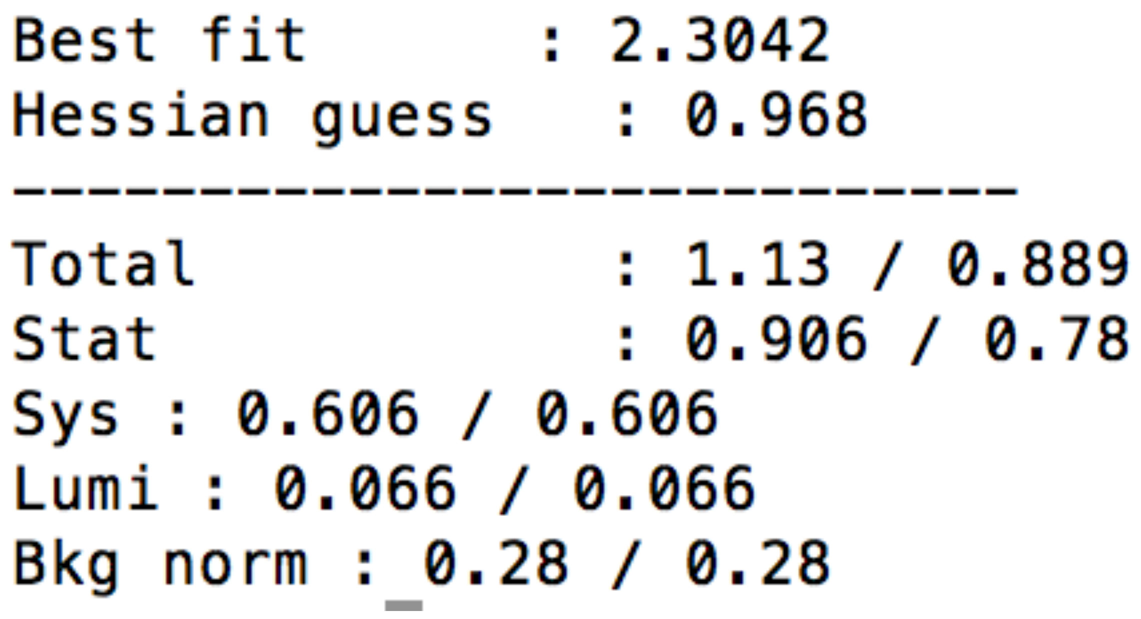


Practical example: Parameter uncertainty with breakdown

- Exercise:
  - Open macros/getErrorWithBreakdown.C
  - Run it on μ from the OnOff model:
    - **> root -b -q ‘macros/getErrorWithBreakdown.C(“OnOff”,”mu”)’**
  - Examine output

Practical example: Parameter uncertainty with breakdown

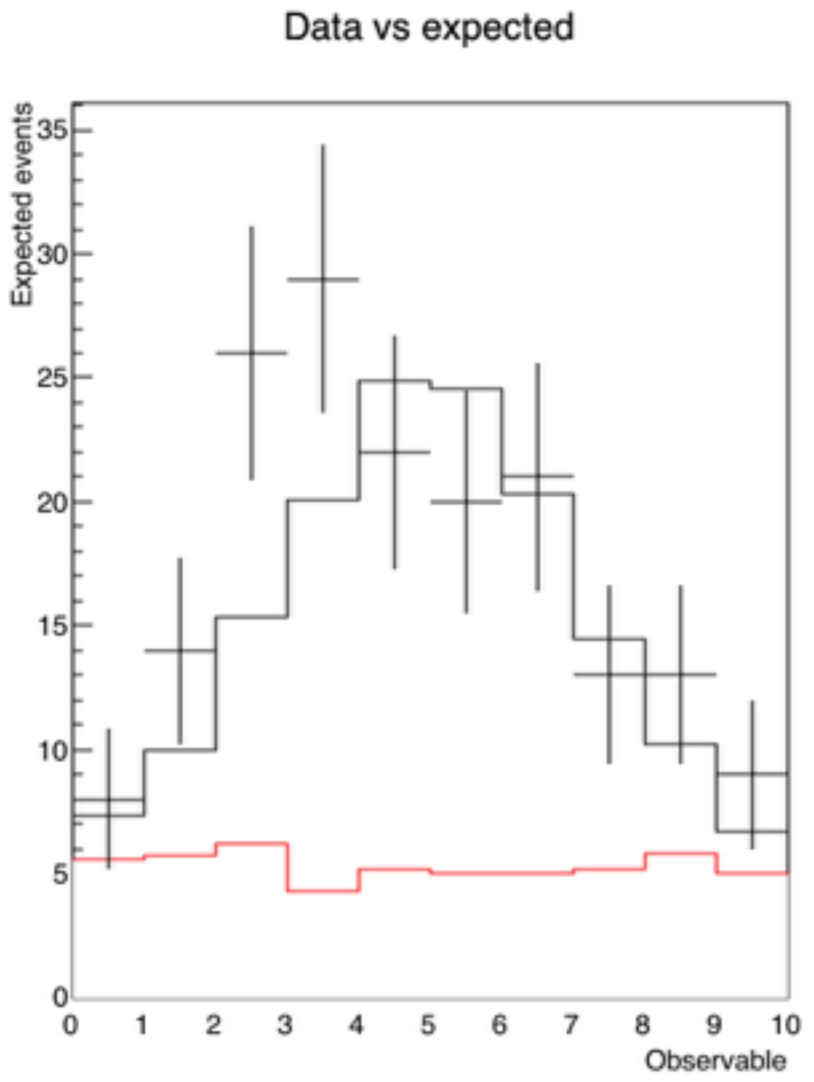
- Output when running over mu: **> root -b -q 'macros/getErrorWithBreakdown.C("OnOff","mu")'**

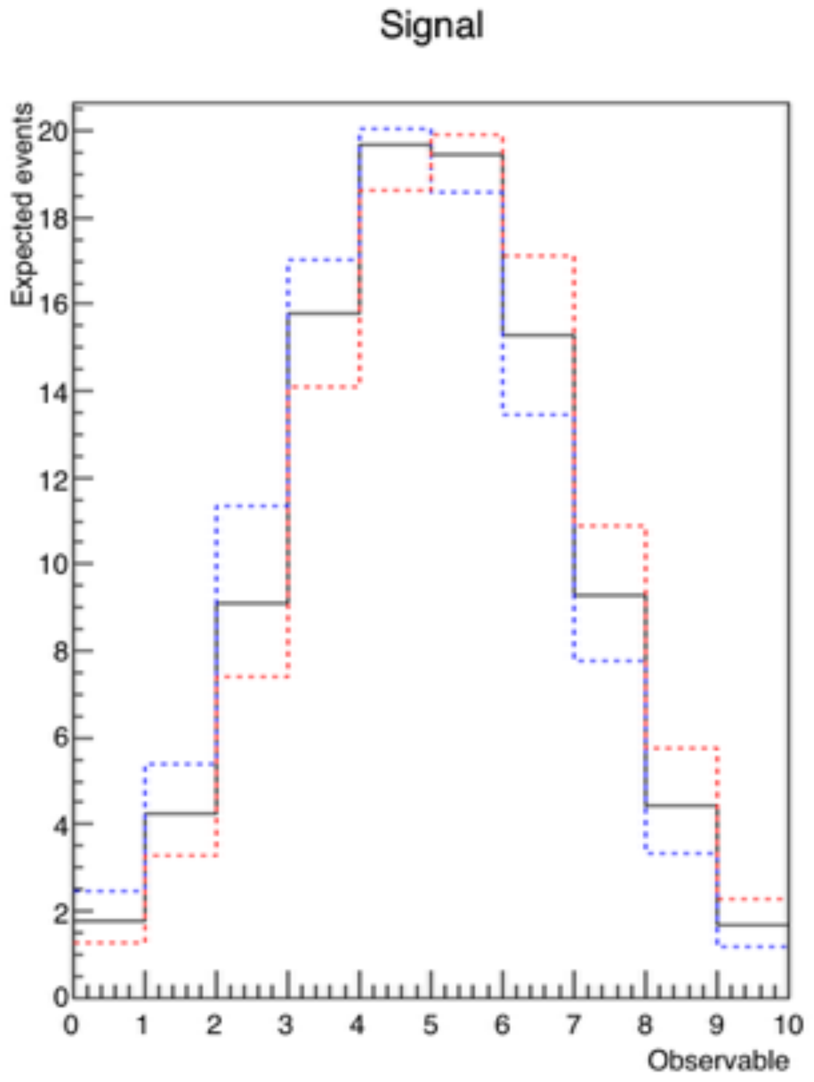


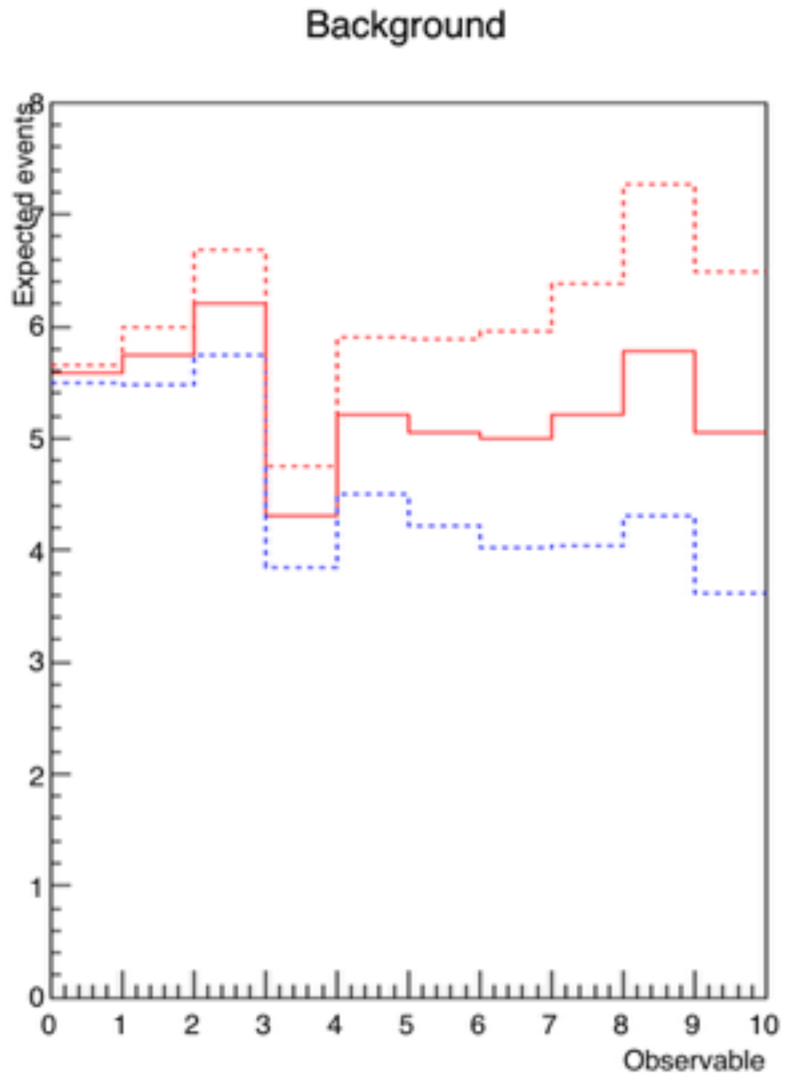
```
Best fit        : 2.3042
Hessian guess     : 0.968
---------------------------
Total             : 1.13 / 0.889
Stat              : 0.906 / 0.78
Sys : 0.606 / 0.606
Lumi : 0.066 / 0.066
Bkg norm : 0.28 / 0.28
```

Practical example: Fitting a distribution

- Exercise:
  - Run ex/ShapeFit_makeHists.C to make histogram inputs for a shape fit:
    - **> root ex/ShapeFit_makeHists.C**
  - Open ex/ShapeFit.C and run it:
    - **> root ex/ShapeFit.C**
- This exercise introduces two main elements: Multiple bins (shape), and shape systematics

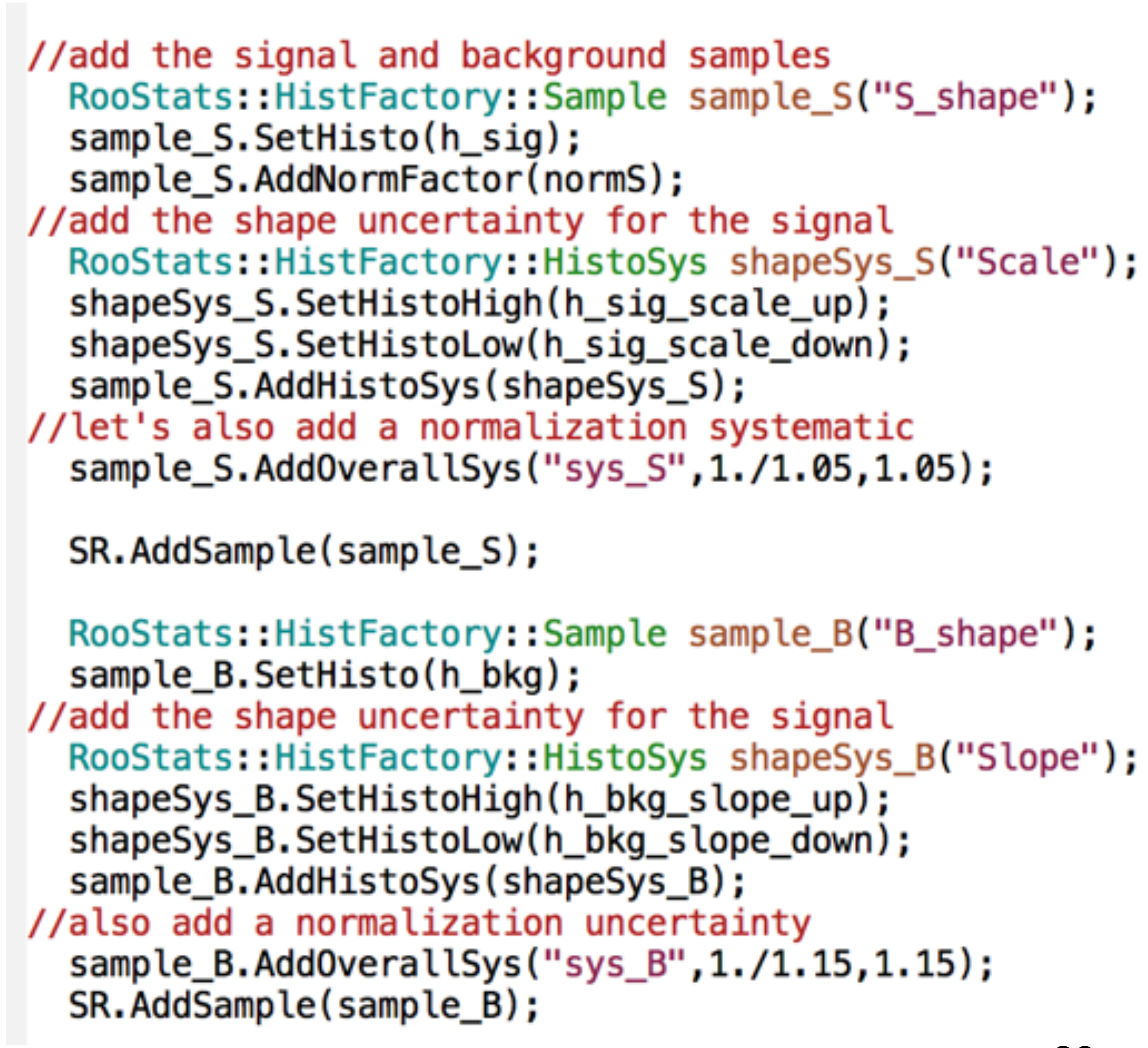






Practical example: Fitting a distribution

- Exercise:
    - Run ex/ShapeFit_makeHists.C to make histogram inputs for a shape fit:
        - **> root ex/ShapeFit_makeHists.C**
    - Open ex/ShapeFit.C and run it:
        - **> root ex/ShapeFit.C**
- This exercise introduces two main elements: Multiple bins (shape), and shape systematics

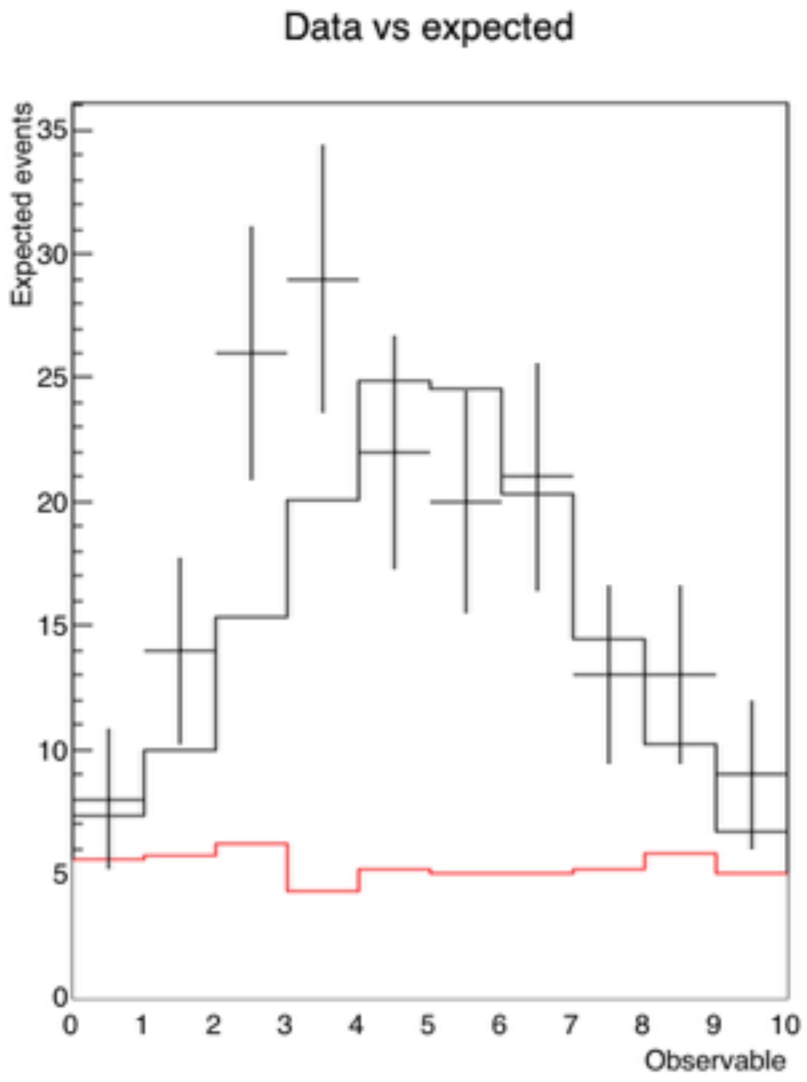


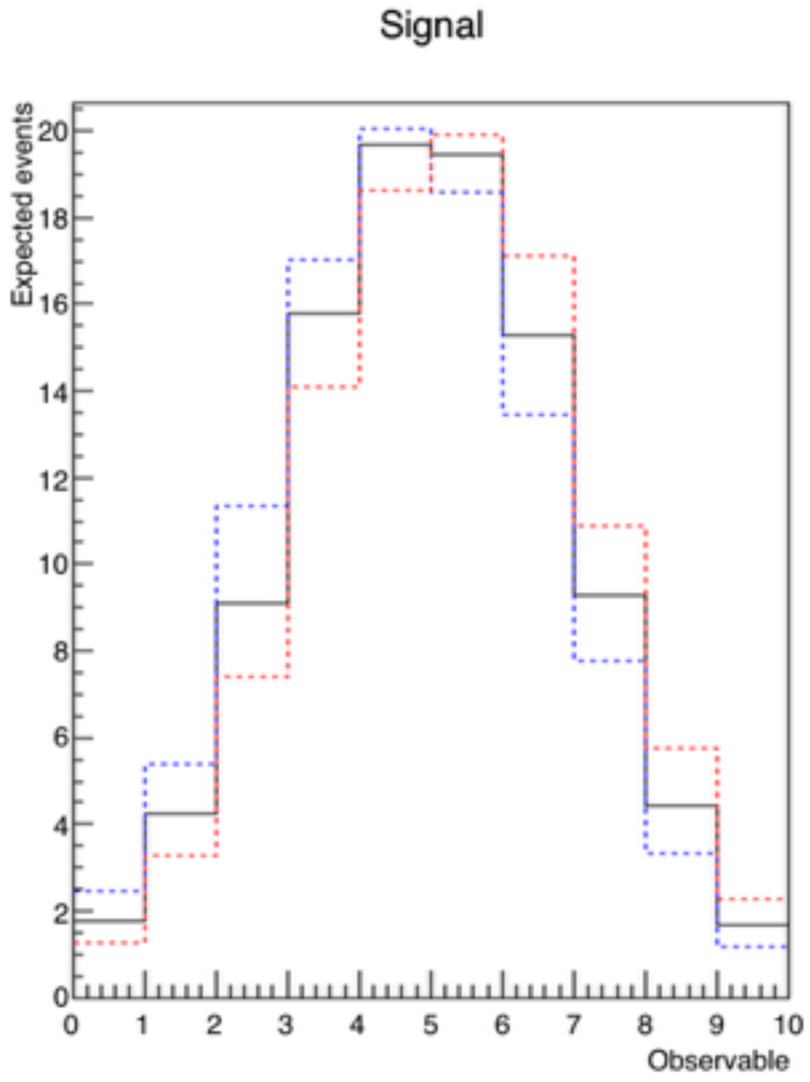
```
//add the signal and background samples
  RooStats::HistFactory::Sample sample_S("S_shape");
  sample_S.SetHisto(h_sig);
  sample_S.AddNormFactor(normS);
//add the shape uncertainty for the signal
  RooStats::HistFactory::HistoSys shapeSys_S("Scale");
  shapeSys_S.SetHistoHigh(h_sig_scale_up);
  shapeSys_S.SetHistoLow(h_sig_scale_down);
  sample_S.AddHistoSys(shapeSys_S);
//let's also add a normalization systematic
  sample_S.AddOverallSys("sys_S",1./1.05,1.05);

  SR.AddSample(sample_S);

  RooStats::HistFactory::Sample sample_B("B_shape");
  sample_B.SetHisto(h_bkg);
//add the shape uncertainty for the signal
  RooStats::HistFactory::HistoSys shapeSys_B("Slope");
  shapeSys_B.SetHistoHigh(h_bkg_slope_up);
  shapeSys_B.SetHistoLow(h_bkg_slope_down);
  sample_B.AddHistoSys(shapeSys_B);
//also add a normalization uncertainty
  sample_B.AddOverallSys("sys_B",1./1.15,1.15);
  SR.AddSample(sample_B);
```

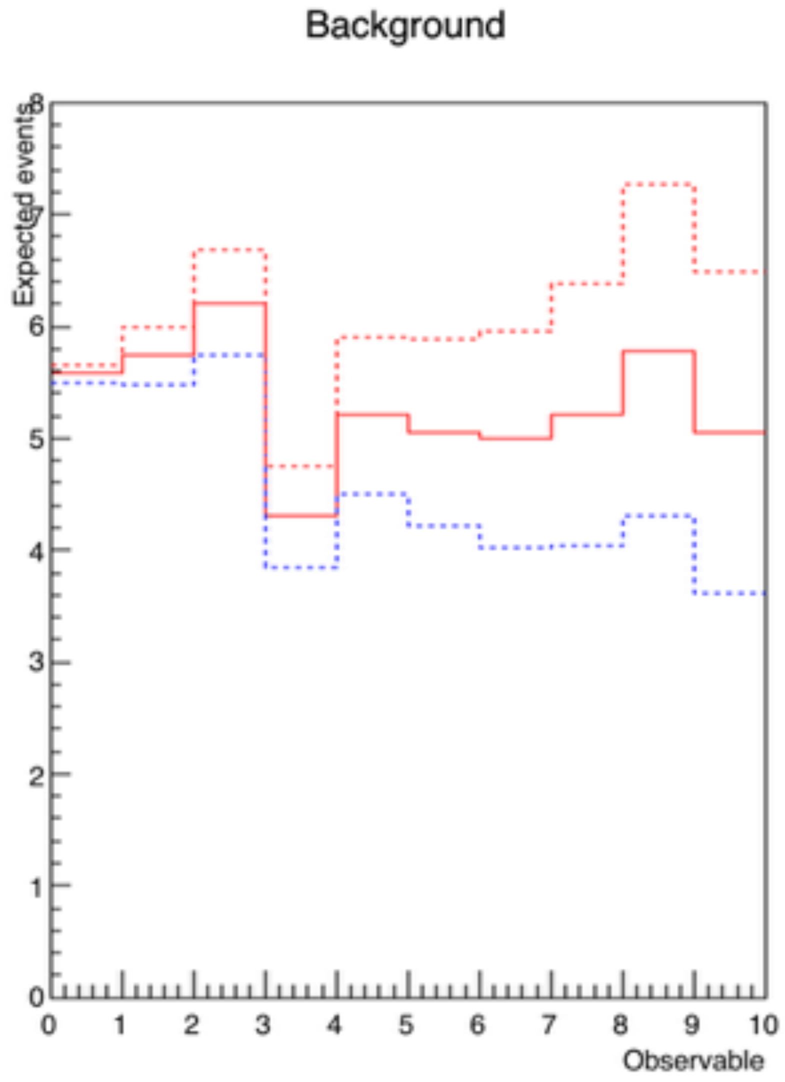
Shape systematics

Nuisance parameter pulls/constraints

- The data has an interesting feature (by pure TRandom chance!): The distribution seems systematically shifted to the left
  - This is something that could delay an analysis by months, but is really just Poisson statistics
  - It nicely serves as an example for studying NP pulls
- The best fit value of a NP θ may not be centered at zero due to fluctuations of the data
- Just as with POIs, you can find the best fit value and uncertainty on NPs, which pre-fit were unit Gaussian

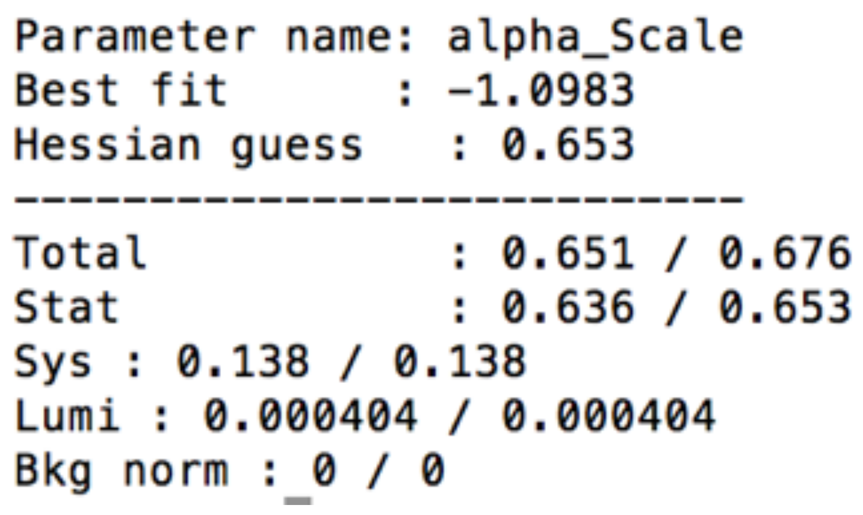




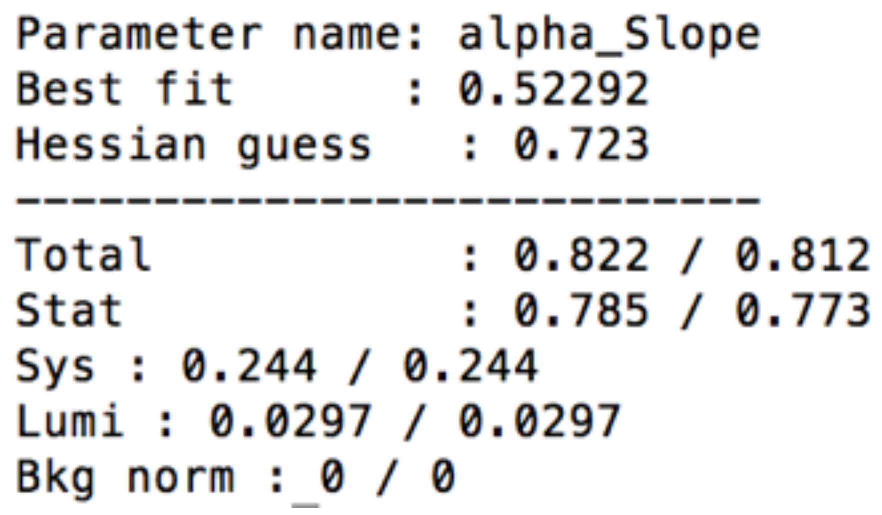


Nuisance parameter pulls/constraints

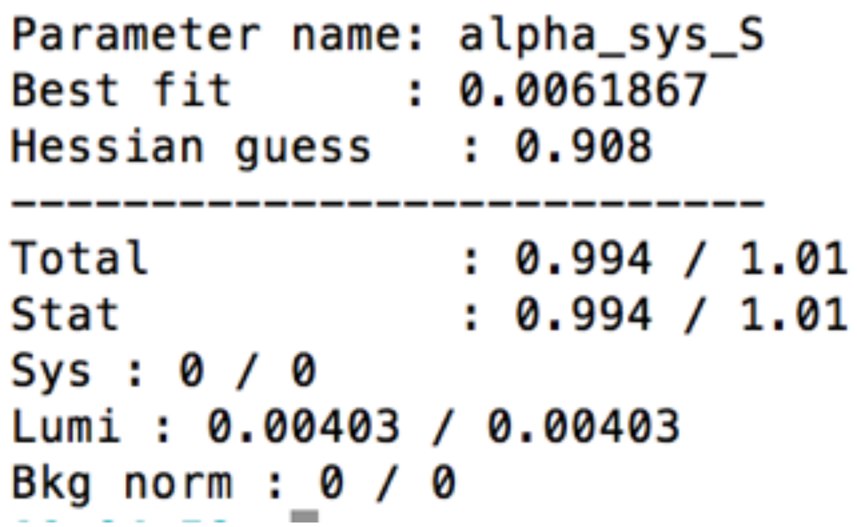
- Exercise:
    - Run getErrorWithBreakdown.C over various model parameters and examine output:
        - **> root -b -q ‘macros/getErrorWithBreakdown.C(“ShapeFit”,”alpha_Scale”)’**
        - **> root -b -q ‘macros/getErrorWithBreakdown.C(“ShapeFit”,”alpha_Slope”)’**
        - **> root -b -q ‘macros/getErrorWithBreakdown.C(“ShapeFit”,”alpha_sys_S”)’**
        - **> root -b -q ‘macros/getErrorWithBreakdown.C(“ShapeFit”,”alpha_sys_B”)’**



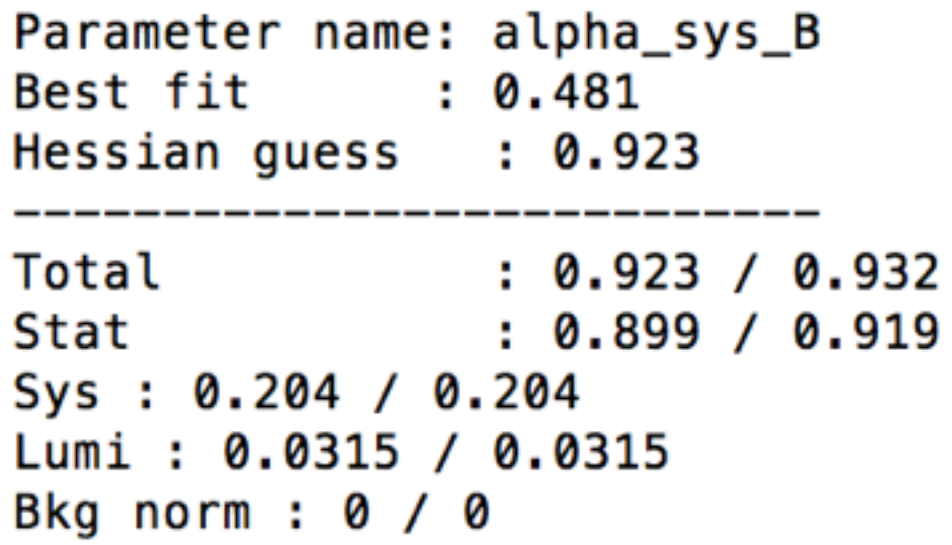
```
Parameter name: alpha_Scale
Best fit       : -1.0983
Hessian guess    : 0.653
---------------------------
Total            : 0.651 / 0.676
Stat             : 0.636 / 0.653
Sys : 0.138 / 0.138
Lumi : 0.000404 / 0.000404
Bkg norm : 0 / 0
```



```
Parameter name: alpha_Slope
Best fit       : 0.52292
Hessian guess    : 0.723
---------------------------
Total            : 0.822 / 0.812
Stat             : 0.785 / 0.773
Sys : 0.244 / 0.244
Lumi : 0.0297 / 0.0297
Bkg norm : 0 / 0
```

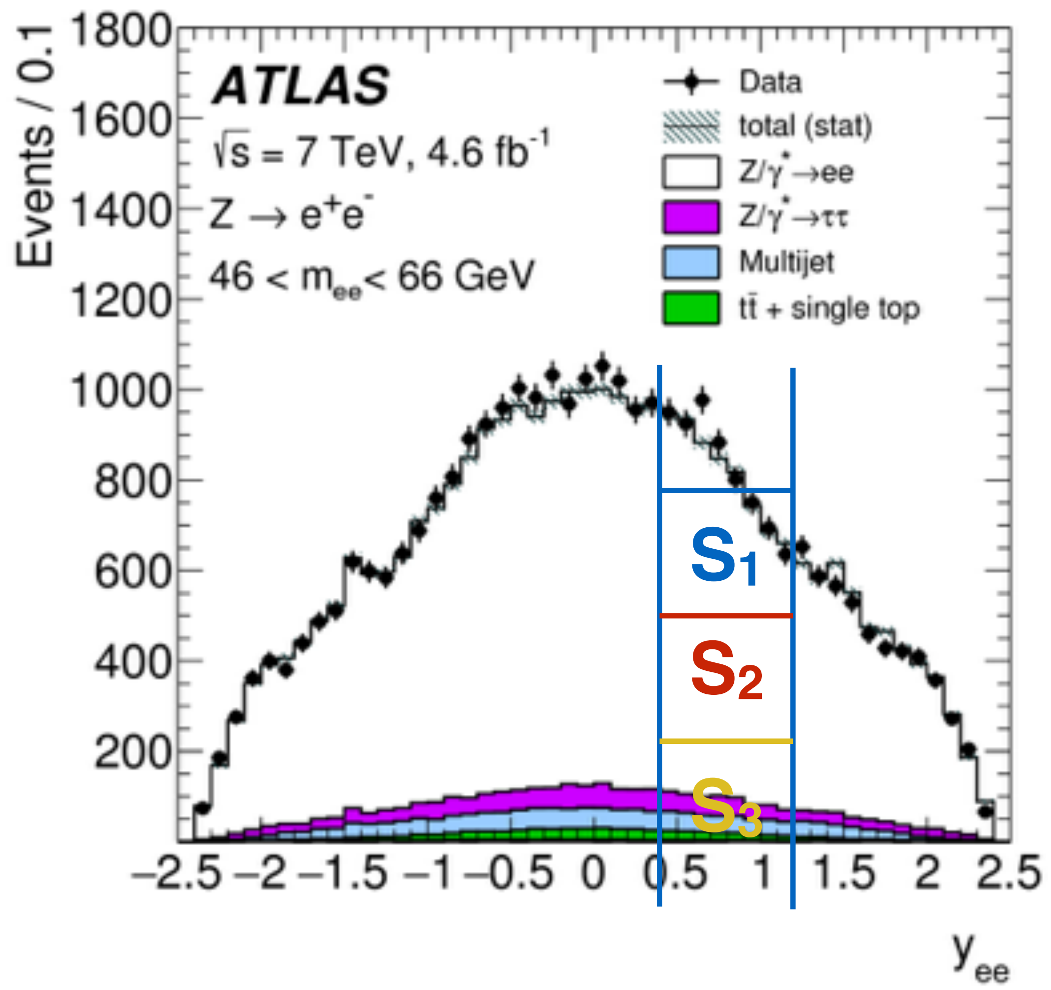


```
Parameter name: alpha_sys_S
Best fit       : 0.0061867
Hessian guess    : 0.908
---------------------------
Total            : 0.994 / 1.01
Stat             : 0.994 / 1.01
Sys : 0 / 0
Lumi : 0.00403 / 0.00403
Bkg norm : 0 / 0
```

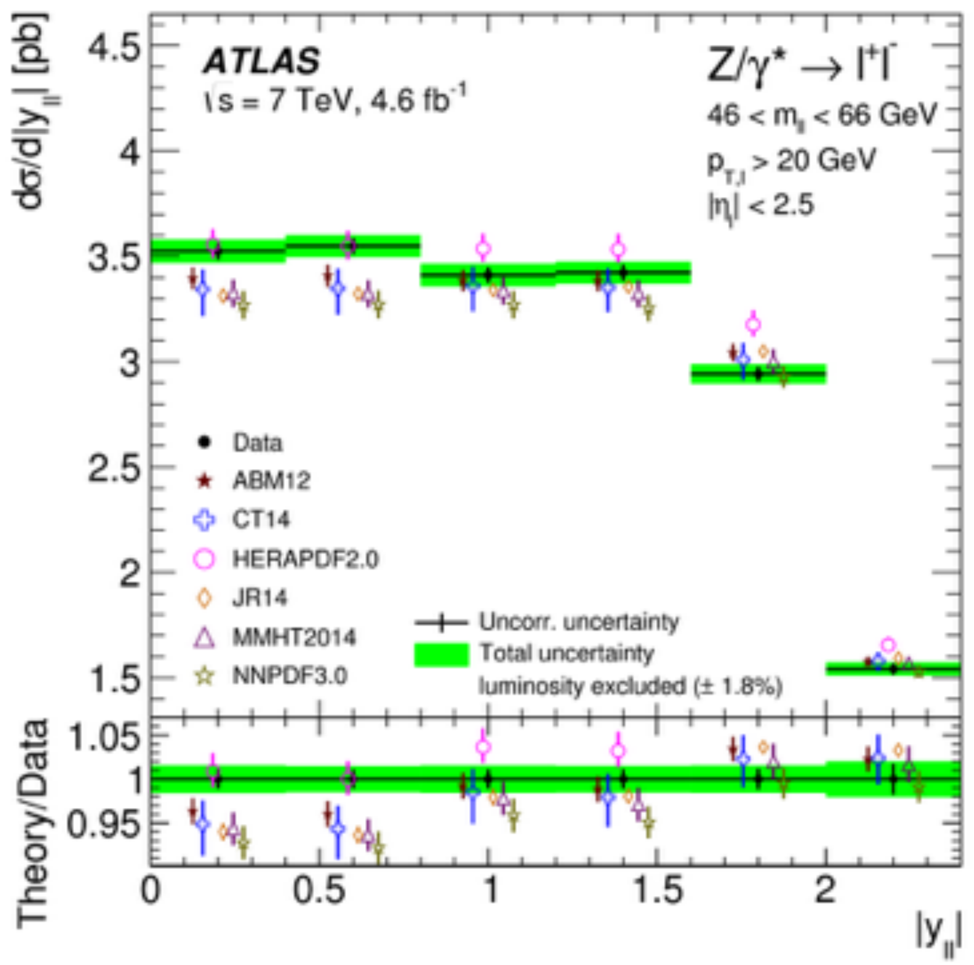


```
Parameter name: alpha_sys_B
Best fit       : 0.481
Hessian guess    : 0.923
---------------------------
Total            : 0.923 / 0.932
Stat             : 0.899 / 0.919
Sys : 0.204 / 0.204
Lumi : 0.0315 / 0.0315
Bkg norm : 0 / 0
```

Matrix Inversion



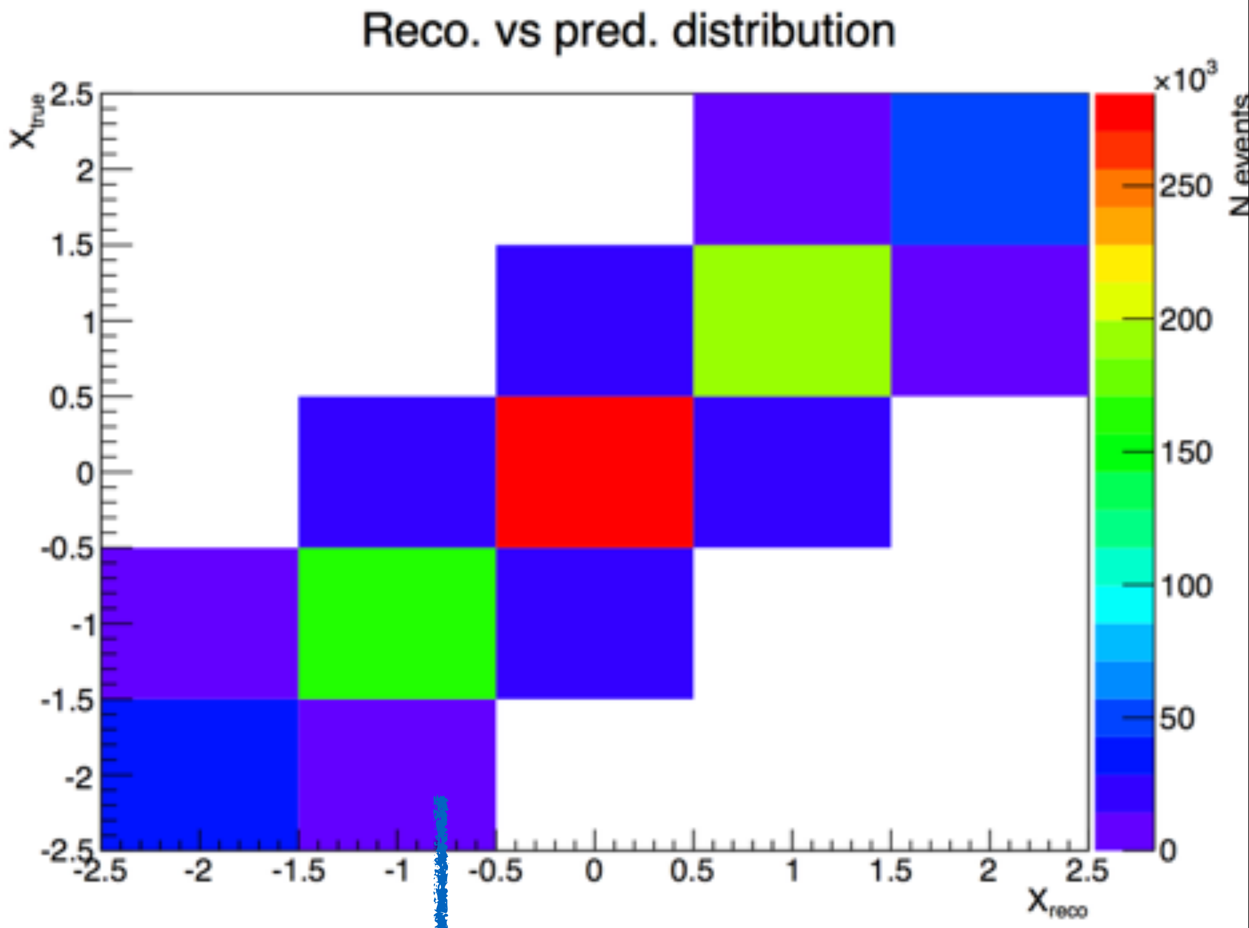
$R^{-1} * (N - B)$



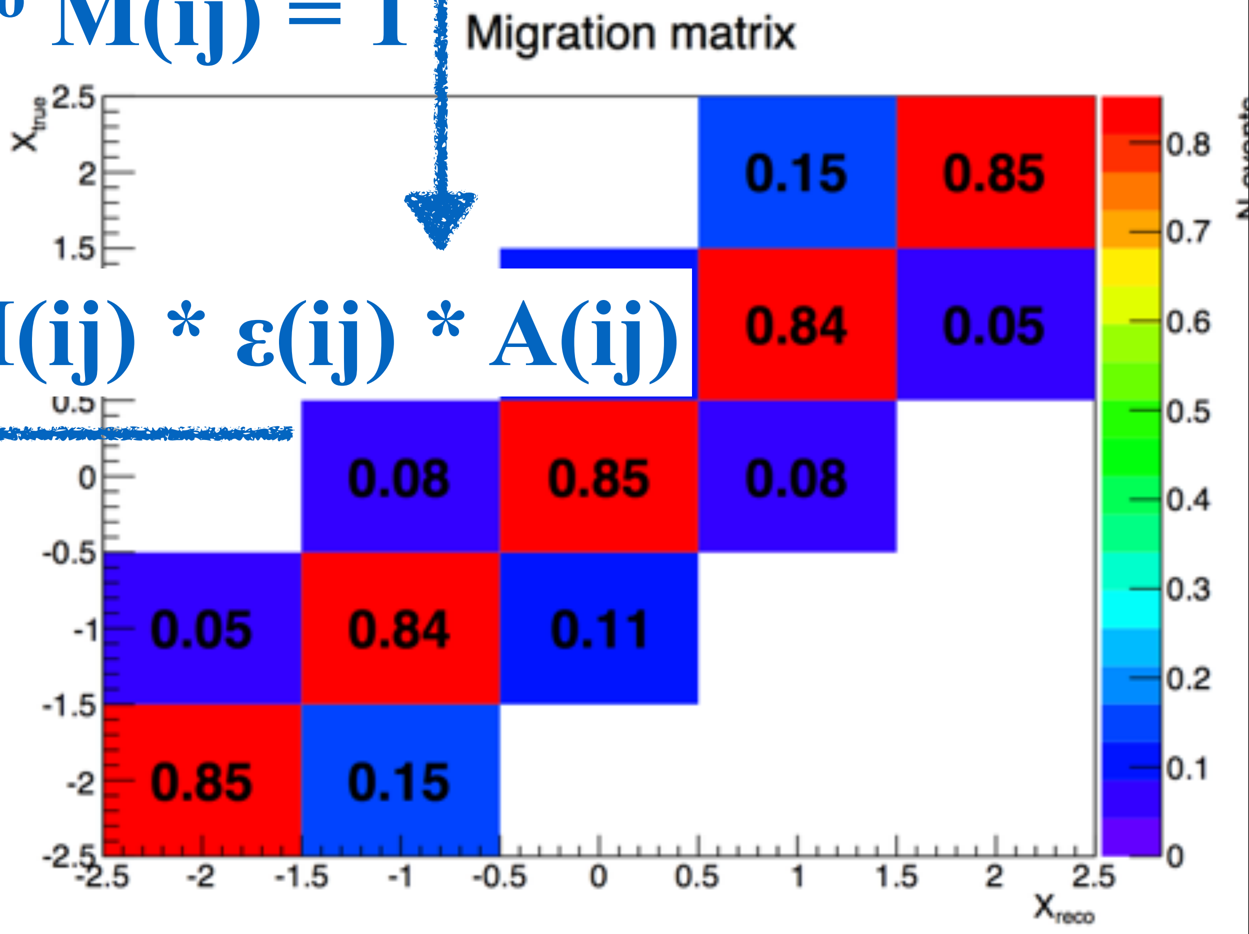
- Matrix inversion is the most basic form of unfolding
- A binned reconstructed distribution can be inverted to obtain an estimate of the true underlying distribution
- Reconstructed distribution is comprised of different truth contributions, and some background:
  - N(i) = B(i) + Σj R(ij) * S(j)
  - N(i) is i'th bin of the reconstructed distribution
  - B(i) is the estimated background
  - S(j) is the underlying distribution of interest, in bins of truth "j"
  - R(ij) is the response matrix, not to be confused with the migration matrix!
- Formula is inverted (equivalently, data is unfolded) to estimate the true distribution S(j). Response matrix corrects for experimental migrations, acceptance, and efficiency.
  - **$S = R^{-1} * (N - B)$**

Response matrix

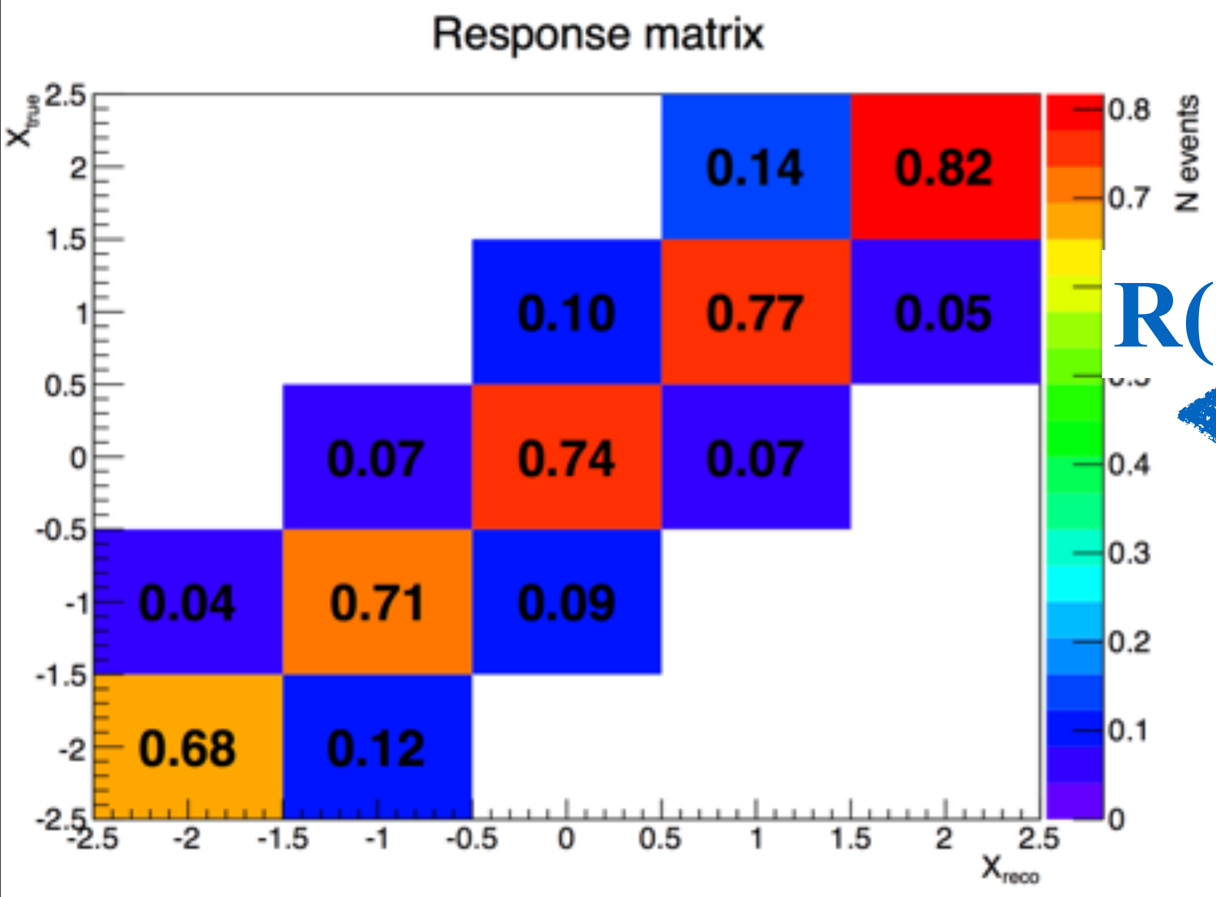
- "Response matrix" is the migration matrix, after correcting for efficiency, possibly acceptance!)
- Generally estimated from MC, the response matrix is subject to theoretical, experimental, and MC stat uncertainties
- For matrix inversion, this is all you need to unfold the distributions (but still need unc.)



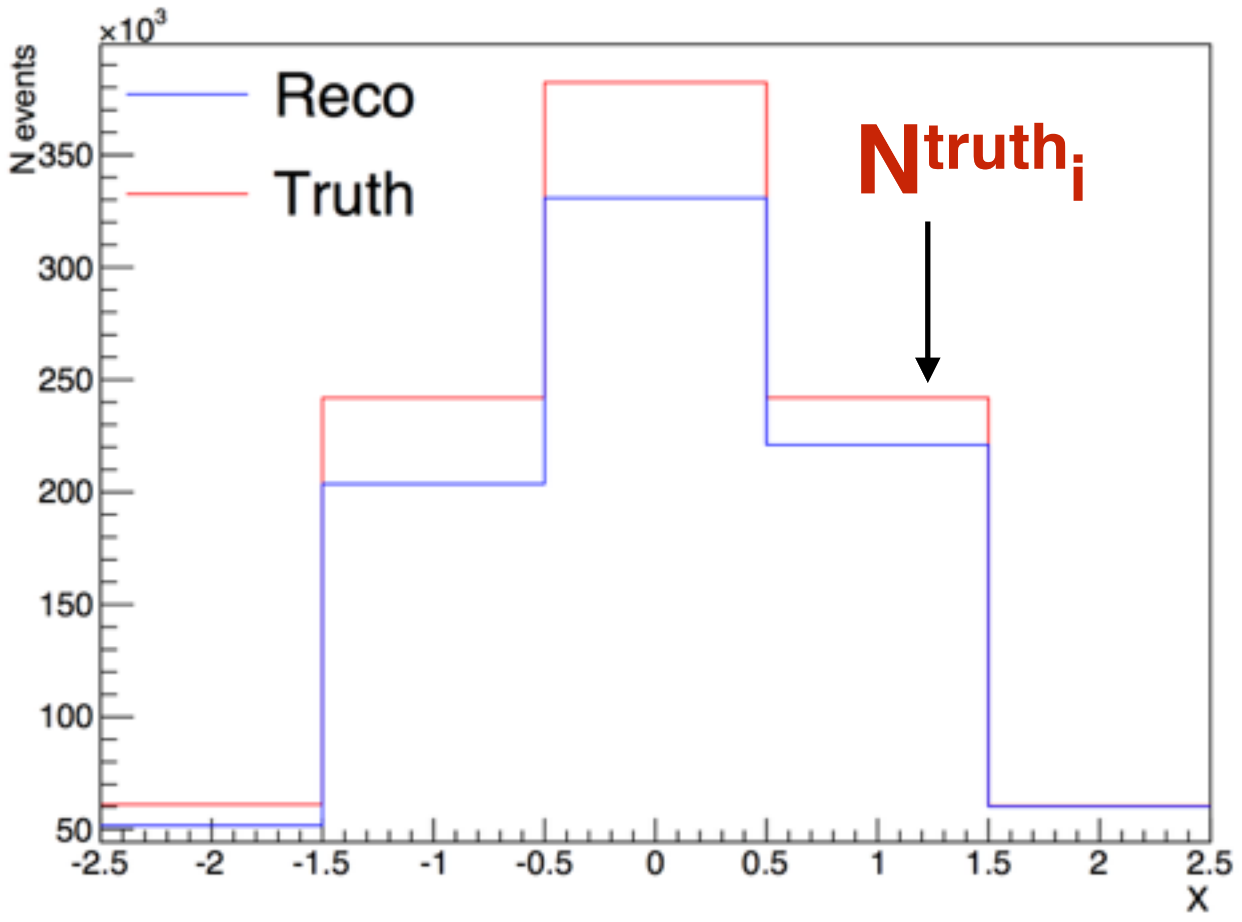
Σ_i^{reco} M(ij) = 1

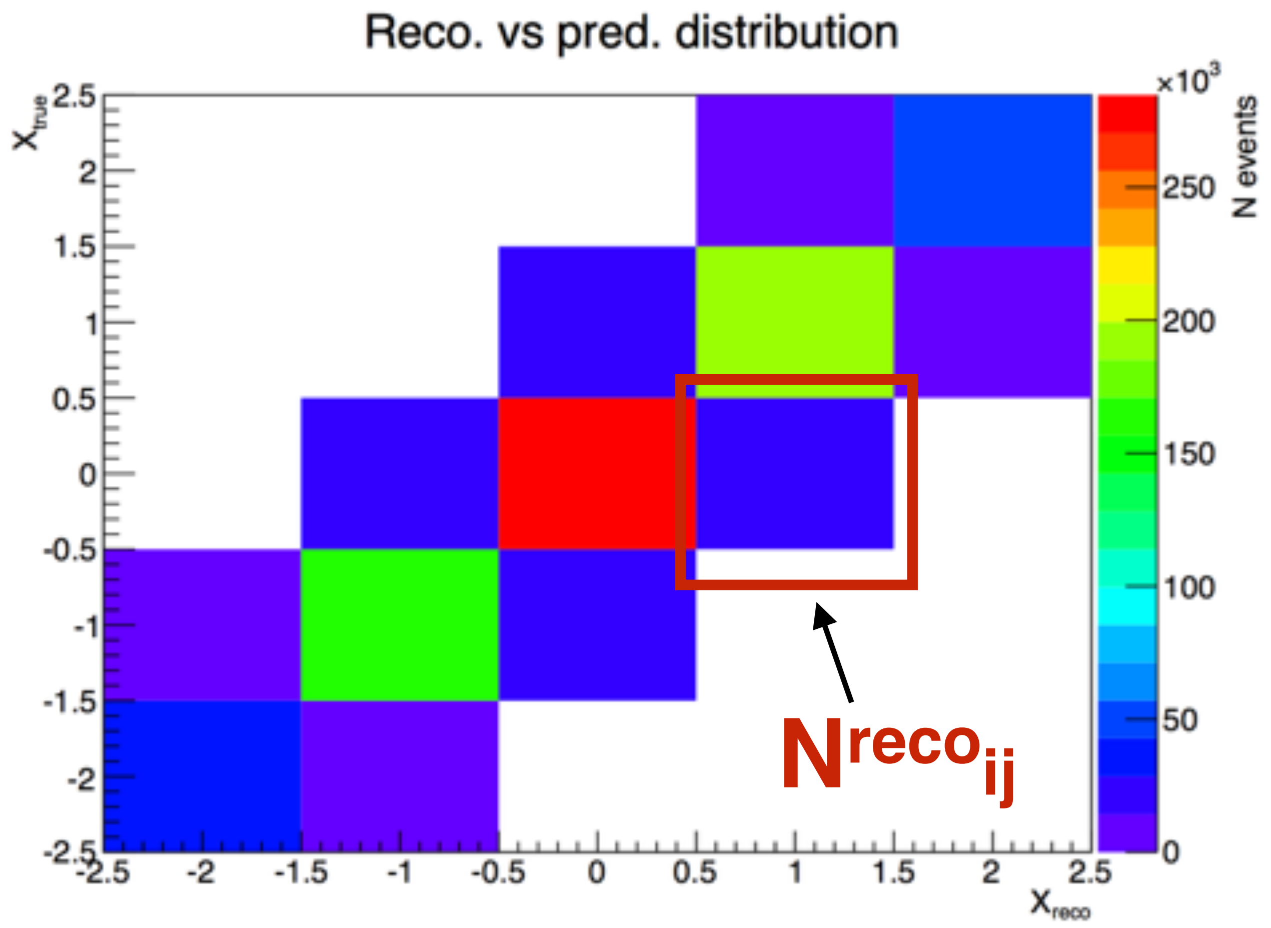


R(ij) = M(ij) * ε(ij) * A(ij)



Unfolding via likelihood

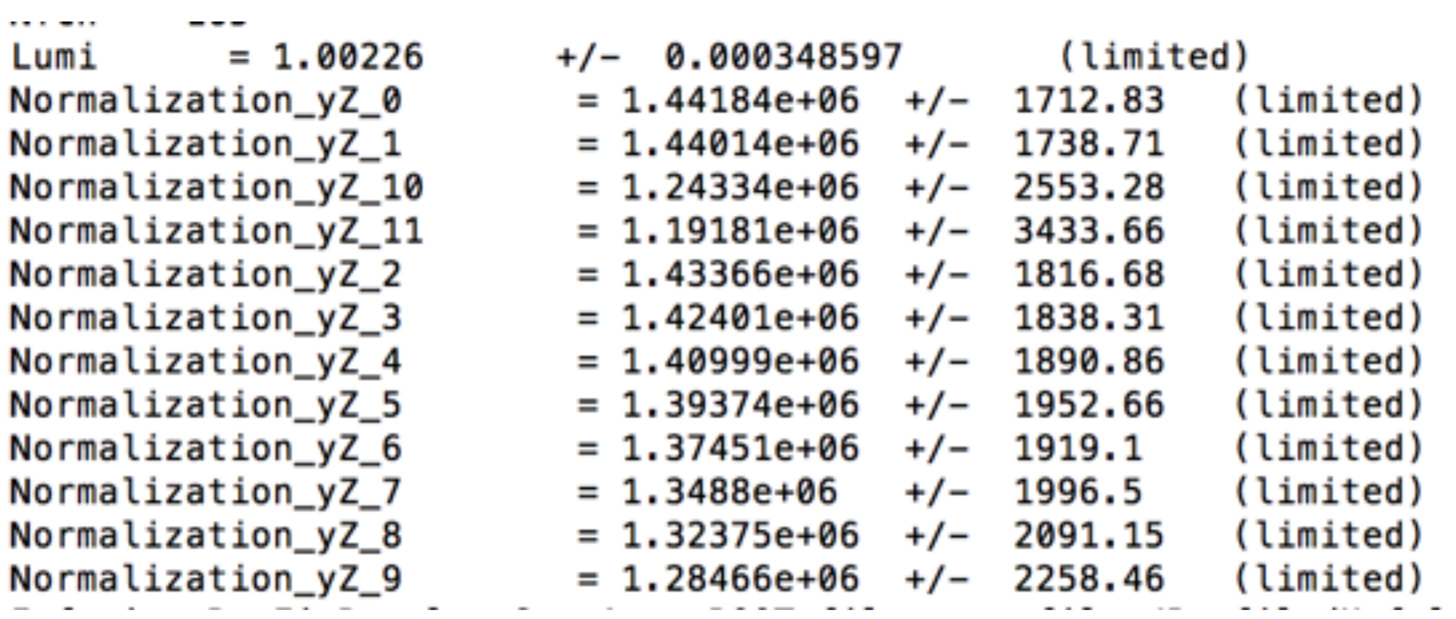




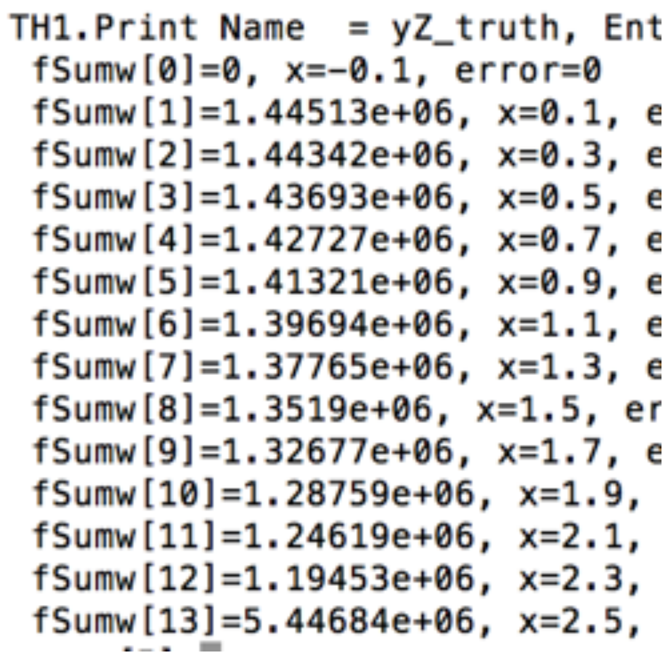
- Response matrix R_{ij} can be written in terms the number of reco and truth events in a given analysis bin: Rij = $N^{reco}_{ij} / N^{truth}_i$
- N^{truth}_i is the number of expected truth events in truth bin 'i' of your fiducial definition
- N^{reco}_{ij} is the number of expected reconstructed events in reco bin 'j', AND truth bin 'i' of your fiducial definition
- Binned likelihood can be written out in terms of the response matrix Rij, integrated lumi L_{int}, background B, and non-fiducial signal S^{nonfid}, and finally "unfolded" cross-section parameters σ_i
- **$L(\sigma) = \Pi_j P(N_j \mid L_{int} * \Sigma_i R_{ij}\sigma_i + B_j + S^{non\text{-}fid}_j)$**
- Non-fiducial signal $S^{non\text{-}fid}$ is technically treated as a background, representing the signal contribution outside of your truth fiducial region but passing your reco selections

Practical example: Profiled unfolding

- Exercise:
  - Open ex/ProfiledUnfolding_makeHists.C
  - Either run it, or use hists already created in hists/ProfiledUnfolding.root
    - **> root -b -q ex/ProfiledUnfolding_makeHists.C**
  - Examine output
  - Open ex/ProfiledUnfolding.C (workspace building code)
  - Run it: **> root -b -q ex/ProfiledUnfolding.C**
  - Run a simple fit: **> root -b -q 'macros/simpleFit.C("ProfiledUnfolding")'**
- Open histograms and print the truth distribution to check the closure of the fit:
  - > **root hists/ProfiledUnfolding.root**
  - > **yZ_truth->Print("all")**



```
Lumi        = 1.00226       +/-  0.000348597       (limited)
Normalization_yZ_0          = 1.44184e+06  +/-  1712.83   (limited)
Normalization_yZ_1          = 1.44014e+06  +/-  1738.71   (limited)
Normalization_yZ_10         = 1.24334e+06  +/-  2553.28   (limited)
Normalization_yZ_11         = 1.19181e+06  +/-  3433.66   (limited)
Normalization_yZ_2          = 1.43366e+06  +/-  1816.68   (limited)
Normalization_yZ_3          = 1.42401e+06  +/-  1838.31   (limited)
Normalization_yZ_4          = 1.40999e+06  +/-  1890.86   (limited)
Normalization_yZ_5          = 1.39374e+06  +/-  1952.66   (limited)
Normalization_yZ_6          = 1.37451e+06  +/-  1919.1    (limited)
Normalization_yZ_7          = 1.3488e+06   +/-  1996.5    (limited)
Normalization_yZ_8          = 1.32375e+06  +/-  2091.15   (limited)
Normalization_yZ_9          = 1.28466e+06  +/-  2258.46   (limited)
```



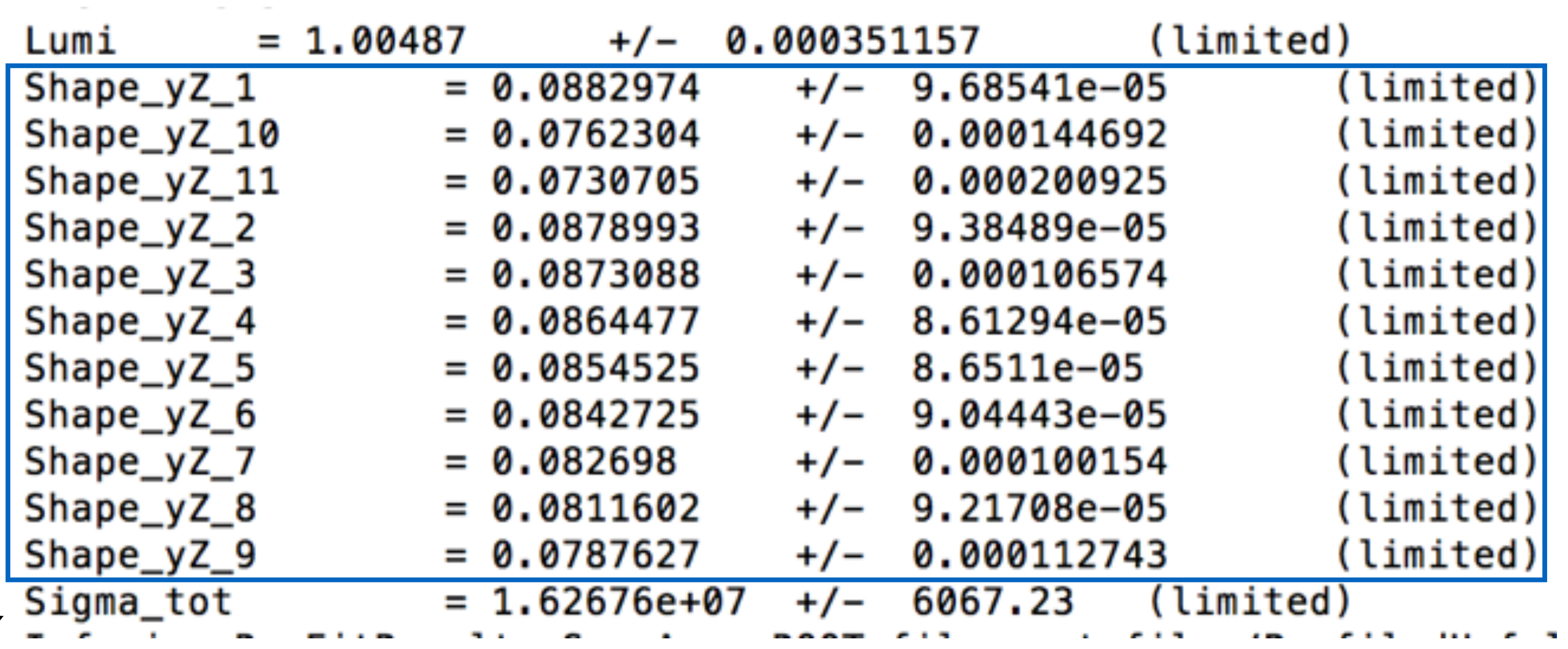
```
TH1.Print Name  = yZ_truth, Ent
 fSumw[0]=0, x=-0.1, error=0
 fSumw[1]=1.44513e+06, x=0.1, e
 fSumw[2]=1.44342e+06, x=0.3, e
 fSumw[3]=1.43693e+06, x=0.5, e
 fSumw[4]=1.42727e+06, x=0.7, e
 fSumw[5]=1.41321e+06, x=0.9, e
 fSumw[6]=1.39694e+06, x=1.1, e
 fSumw[7]=1.37765e+06, x=1.3, e
 fSumw[8]=1.3519e+06, x=1.5, er
 fSumw[9]=1.32677e+06, x=1.7, e
 fSumw[10]=1.28759e+06, x=1.9,
 fSumw[11]=1.24619e+06, x=2.1,
 fSumw[12]=1.19453e+06, x=2.3,
 fSumw[13]=5.44684e+06, x=2.5,
```

Reparametrization

- Often you'll want to re-express your baseline measurements in terms of something else
  - From absolute differential measurement to shape differential: $d\sigma/dX \to 1/\sigma * (d\sigma/dX)$
  - From measurement of a distribution to a SM parameter: $A_{FB} \to \sin^2\theta_W$
  - From measurements in individual channels to 'delta' parameters for compatibility tests: $(d\sigma/dX)_{ee}, (d\sigma/dX)_{\mu\mu} \to \Delta(d\sigma / dX) = (d\sigma/dX)_{ee} - (d\sigma/dX)_{\mu\mu}$
- This requires reparametrizing the likelihood from one set of parameters to a completely new set
- This is straightforward in RooFit, as long as you can easily write an expression for the old parameters in terms of the new ones
- For an absolute to shape differential measurement, this means, for example, writing:
  - Diff_X_i = Shape_X_i * Sigma_tot
- But wait! We started with N differential parameters, and ended up with N+1, since we picked up Sigma_tot as well
- There's a degeneracy with one of the shape parameters that has to be solved
- Instead of involving one of the new parameters, say, Shape_X_0, we can use the constraint Σi Shape_X_i = 1 to get rid of it. This means in the case of Diff_X_0, we instead rewrite as
  - Diff_X_0 = (1 - $\Sigma_{i!=0}$ Shape_X_i) * Sigma_tot

Practical example: Reparametrization

- Exercise:
  - Open macros/parametrizeShape.C
  - Run it on the ProfiledUnfolding example:
    - **> root -b -q ‘macros/parametrizeShape.C(“ProfiledUnfolding”)’**
  - Run a simple fit to examine the new parameters:
    - **> root -b -q ‘macros/simpleFit.C(“ProfiledUnfolding_Shape”)’**



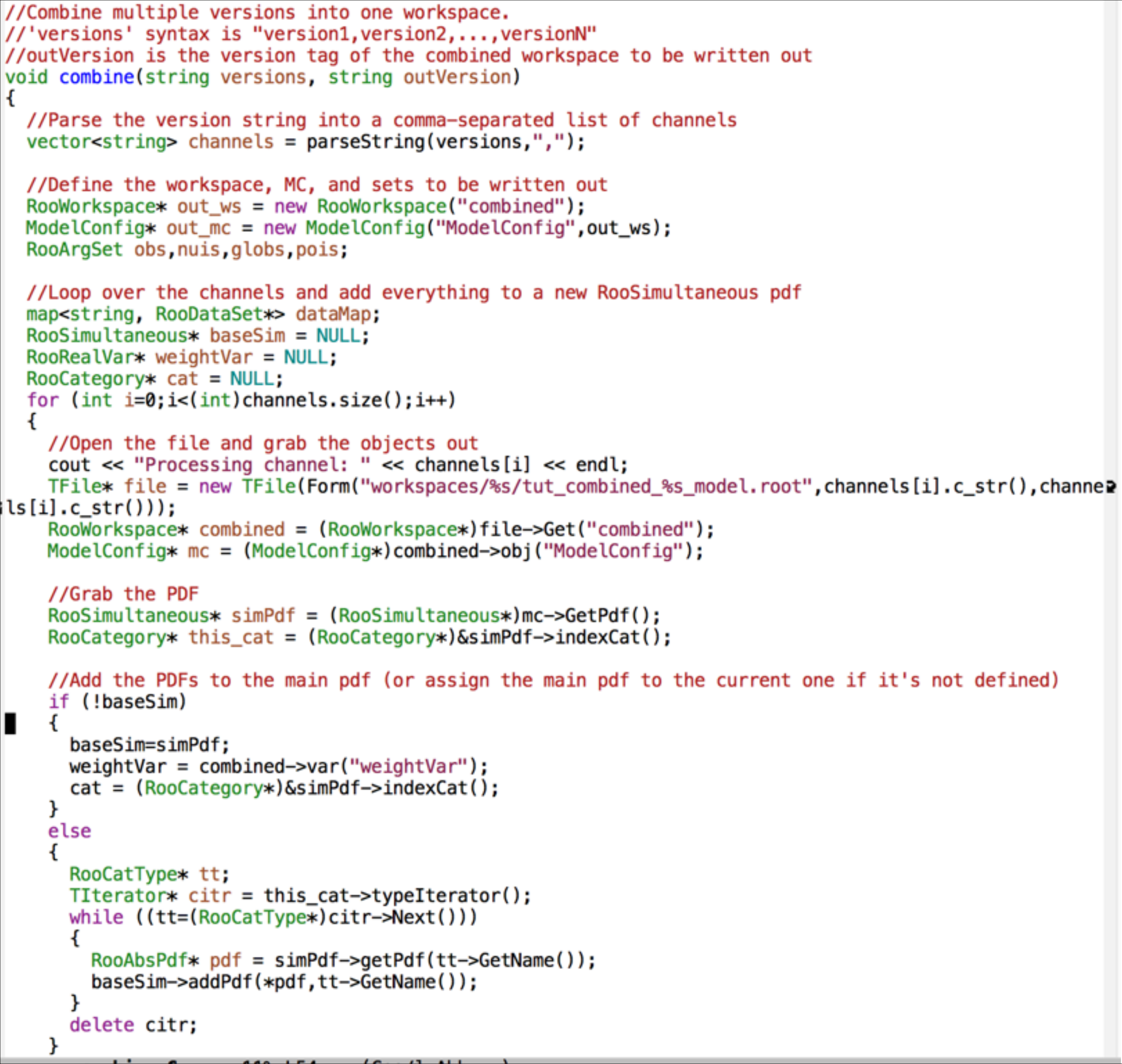
```
Lumi        = 1.00487       +/-   0.000351157        (limited)
Shape_yZ_1          = 0.0882974    +/-   9.68541e-05         (limited)
Shape_yZ_10         = 0.0762304    +/-   0.000144692         (limited)
Shape_yZ_11         = 0.0730705    +/-   0.000200925         (limited)
Shape_yZ_2          = 0.0878993    +/-   9.38489e-05         (limited)
Shape_yZ_3          = 0.0873088    +/-   0.000106574         (limited)
Shape_yZ_4          = 0.0864477    +/-   8.61294e-05         (limited)
Shape_yZ_5          = 0.0854525    +/-   8.6511e-05          (limited)
Shape_yZ_6          = 0.0842725    +/-   9.04443e-05         (limited)
Shape_yZ_7          = 0.082698     +/-   0.000100154         (limited)
Shape_yZ_8          = 0.0811602    +/-   9.21708e-05         (limited)
Shape_yZ_9          = 0.0787627    +/-   0.000112743         (limited)
Sigma_tot           = 1.62676e+07  +/-   6067.23    (limited)
```

New shape parameters

Total cross-section (summed over yZ)

Practical example: Combinations

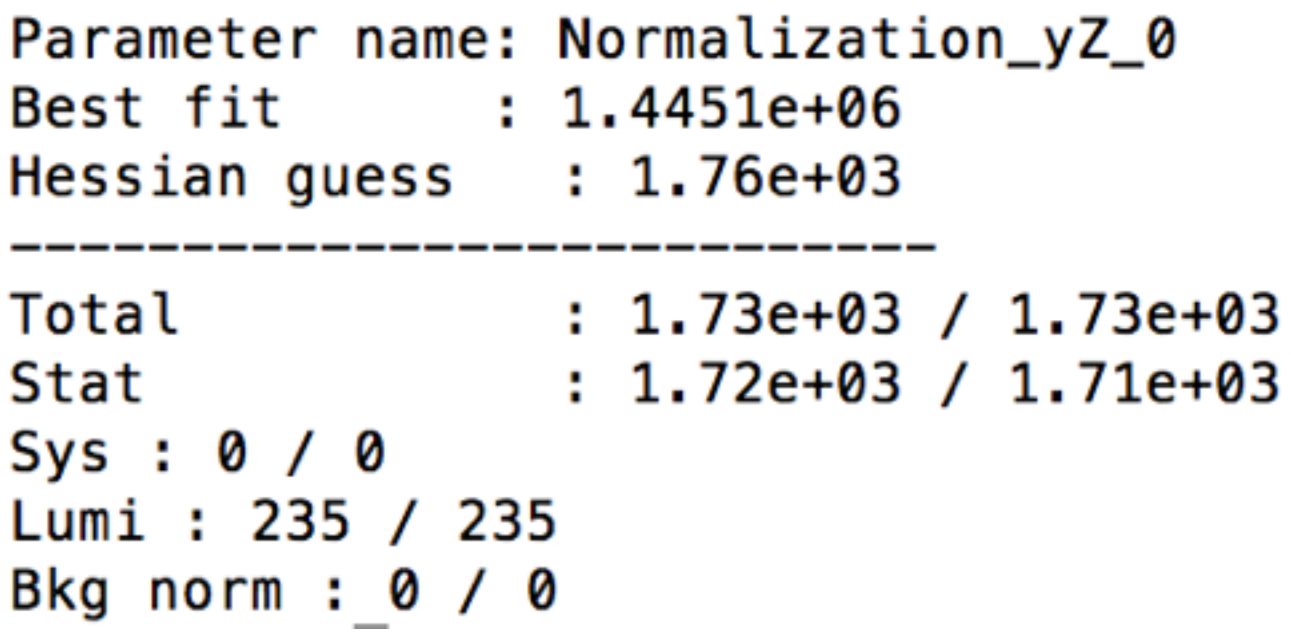
- A basic combination of channels can be trivial, if the names of the parameters in the model are identical
- If you have to do some parameter renaming or something more complicated, best to use something like [WorkspaceCombiner]() (used extensively in HComb, for instance)
- But if you design your likelihood to be combinable from the beginning, it's rather trivial likelihood multiplication
- Exercise:
    - Make a workspace similar to the ProfiledUnfolding model, but from a different channel (histograms already prepared)
        - **> root -b -q ex/ProfiledUnfolding_mumu.C**
    - Open and examine macros/combine.C
    - Run the combination on the first and old channels
        - **> root -b -q 'macros/ combine.C("ProfiledUnfolding,ProfiledUnfolding_mumu","ProfiledUnfolding_cb")'**
    - Compare the uncertainties on the individual and combined models for one parameter
        - **> root -b -q 'macros/getErrorWithBreakdown.C("ProfiledUnfolding")'**
        - **> root -b -q 'macros/getErrorWithBreakdown.C("ProfiledUnfolding_mumu")'**
        - **> root -b -q 'macros/getErrorWithBreakdown.C("ProfiledUnfolding_cb")'**



```
//Combine multiple versions into one workspace.
//'versions' syntax is "version1,version2,...,versionN"
//outVersion is the version tag of the combined workspace to be written out
void combine(string versions, string outVersion)
{
  //Parse the version string into a comma-separated list of channels
  vector<string> channels = parseString(versions,",");

  //Define the workspace, MC, and sets to be written out
  RooWorkspace* out_ws = new RooWorkspace("combined");
  ModelConfig* out_mc = new ModelConfig("ModelConfig",out_ws);
  RooArgSet obs,nuis,globs,pois;

  //Loop over the channels and add everything to a new RooSimultaneous pdf
  map<string, RooDataSet*> dataMap;
  RooSimultaneous* baseSim = NULL;
  RooRealVar* weightVar = NULL;
  RooCategory* cat = NULL;
  for (int i=0;i<(int)channels.size();i++)
  {
    //Open the file and grab the objects out
    cout << "Processing channel: " << channels[i] << endl;
    TFile* file = new TFile(Form("workspaces/%s/tut_combined_%s_model.root",channels[i].c_str(),channe
ls[i].c_str()));
    RooWorkspace* combined = (RooWorkspace*)file->Get("combined");
    ModelConfig* mc = (ModelConfig*)combined->obj("ModelConfig");

    //Grab the PDF
    RooSimultaneous* simPdf = (RooSimultaneous*)mc->GetPdf();
    RooCategory* this_cat = (RooCategory*)&simPdf->indexCat();

    //Add the PDFs to the main pdf (or assign the main pdf to the current one if it's not defined)
    if (!baseSim)
    {
      baseSim=simPdf;
      weightVar = combined->var("weightVar");
      cat = (RooCategory*)&simPdf->indexCat();
    }
    else
    {
      RooCatType* tt;
      TIterator* citr = this_cat->typeIterator();
      while ((tt=(RooCatType*)citr->Next()))
      {
        RooAbsPdf* pdf = simPdf->getPdf(tt->GetName());
        baseSim->addPdf(*pdf,tt->GetName());
      }
      delete citr;
    }
    //Add the old sets to the main one
    obs.add(*mc->GetObservables(),true);
    nuis.add(*mc->GetNuisanceParameters(),true);
    globs.add(*mc->GetGlobalObservables(),true);
    pois.add(*mc->GetParametersOfInterest(),true);

    //Add the old data to the new one
    RooDataSet* data = (RooDataSet*)combined->data("obsData");
    TList* datalist = data->split(*this_cat, true);
    TIterator* dataItr = datalist->MakeIterator();
    RooAbsData* ds;
    while ((ds = (RooAbsData*)dataItr->Next()))
    {
      string typeName(ds->GetName());
      cout << "Adding dataset to map: " << ds->GetName() << endl;
      dataMap[string(ds->GetName())] = (RooDataSet*)ds;
    }
  }

  //Make the combined data
  RooDataSet* data = new RooDataSet("obsData","obsData",RooArgSet(obs,*(RooAbsArg*)weightVar),Index(*cat),Import(dataMap),WeightVar(*weightVar));
  data->Print();

  //Import everything to the workspace
  out_ws->import(*baseSim,Silence());
  out_ws->import(*data);

  obs.sort();
  nuis.sort();
  globs.sort();
  pois.sort();

  //Set everything in the MC and add to the workspace
  out_mc->SetPdf(*baseSim);
  out_mc->SetObservables(obs);
  out_mc->SetNuisanceParameters(nuis);
  out_mc->SetGlobalObservables(globs);
  out_mc->SetParametersOfInterest(pois);
  out_ws->import(*out_mc);

  //Write the workspace out
  system(Form("mkdir -vp workspaces/%s",outVersion.c_str()));
  out_ws->writeToFile(Form("workspaces/%s/tut_combined_%s_model.root",outVersion.c_str(),outVersion.c_str()),true);
}
```

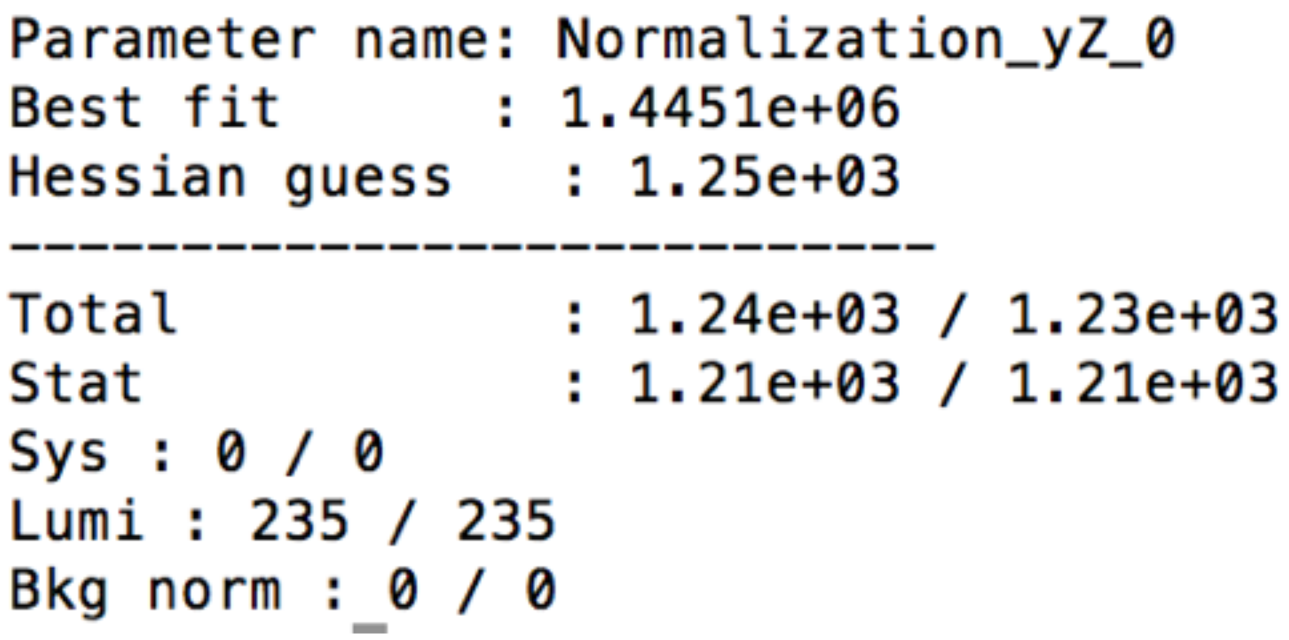
Practical example: Combinations

Each individual channel:



```
Parameter name: Normalization_yZ_0
Best fit       : 1.4451e+06
Hessian guess    : 1.76e+03
---------------------------
Total            : 1.73e+03 / 1.73e+03
Stat             : 1.72e+03 / 1.71e+03
Sys : 0 / 0
Lumi : 235 / 235
Bkg norm : 0 / 0
```

Combined (~1/sqrt(2) smaller uncertainty):



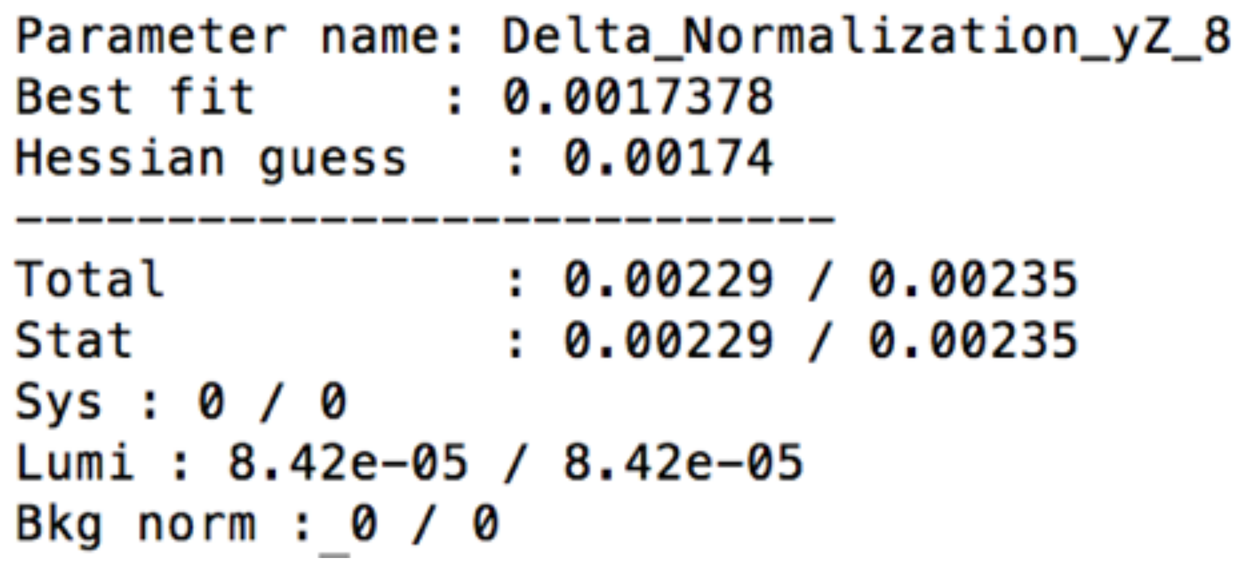
```
Parameter name: Normalization_yZ_0
Best fit       : 1.4451e+06
Hessian guess    : 1.25e+03
---------------------------
Total            : 1.24e+03 / 1.23e+03
Stat             : 1.21e+03 / 1.21e+03
Sys : 0 / 0
Lumi : 235 / 235
Bkg norm : 0 / 0
```

Compatibility tests

- To test the compatibility between two channels or two measurements, you can use a specific style of reparametrization mentioned before, ie something of the form $\Delta(d\sigma / dX) = [(d\sigma/dX)_{ee} - (d\sigma/dX)_{\mu\mu}] / (d\sigma/dX)_{ee}$
- This means adding new parameters that should represent for two different channels 'chan1' and 'chan2' the relative difference in the POIs
    - Delta_POI_X = (POI_X_chan1 - POI_X_chan2) / POI_X_chan1
- As usual, invert this to write the old parameters in terms of the new, simply as
    - POI_X_chan1 = POI_X_chan2 / (1 - Delta_POI_X)
- In this case the POI in channel 2 doesn't change, despite those in channel 1 changing
- This also means that the reparametrization only has to be applied in channel 1

Practical example: Compatibility tests

- Exercise: Reparametrize workspaces to include delta parameters, and measure the level of compatibility between two channels
    - Open macros/parametrizeDelta.C and look through
    - Run it on the “ee” channel (version = “ProfiledUnfolding”). This writes out a workspace with version name “ProfiledUnfolding_Delta”.
        - **> root -b -q ‘macros/parametrizeShape.C(“ProfiledUnfolding”)’**
    - Combine the parametrized “ee” with the normal “mumu”
        - **> root -b -q ‘macros/combine.C(“ProfiledUnfolding_Delta,ProfiledUnfolding_mumu”,”ProfiledUnfolding_Delta_cb”)’**
    - Run getErrorWithBreakdown on some of the new delta parameters
        - **> root -b -q ‘macros/getErrorWithBreakdown.C(“ProfiledUnfolding_Delta_cb”,”Delta_Normalization_yZ_0”)’**



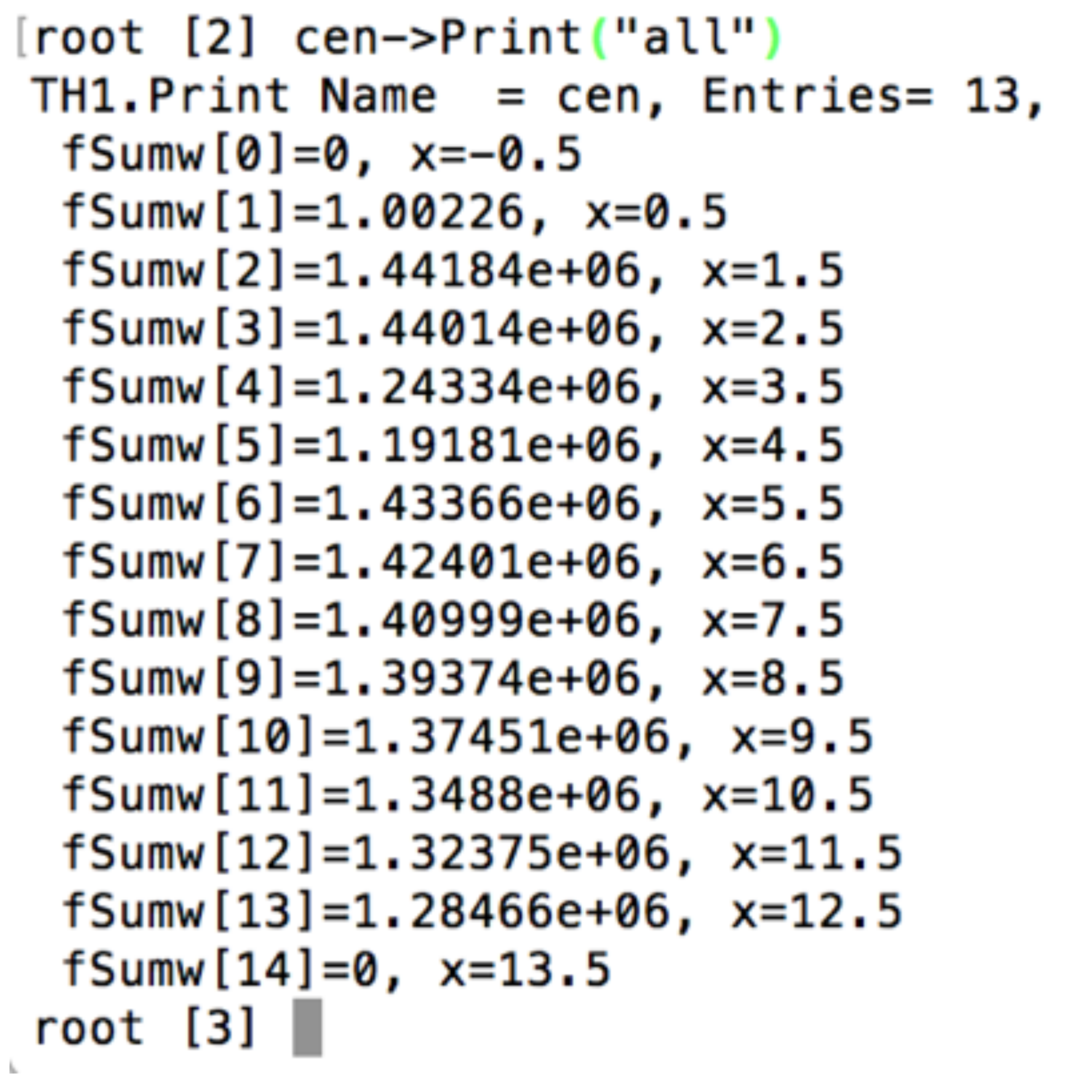
```
Parameter name: Delta_Normalization_yZ_8
Best fit       : 0.0017378
Hessian guess    : 0.00174
---------------------------
Total            : 0.00229 / 0.00235
Stat             : 0.00229 / 0.00235
Sys : 0 / 0
Lumi : 8.42e-05 / 8.42e-05
Bkg norm :_0 / 0
```

Covariance matrices

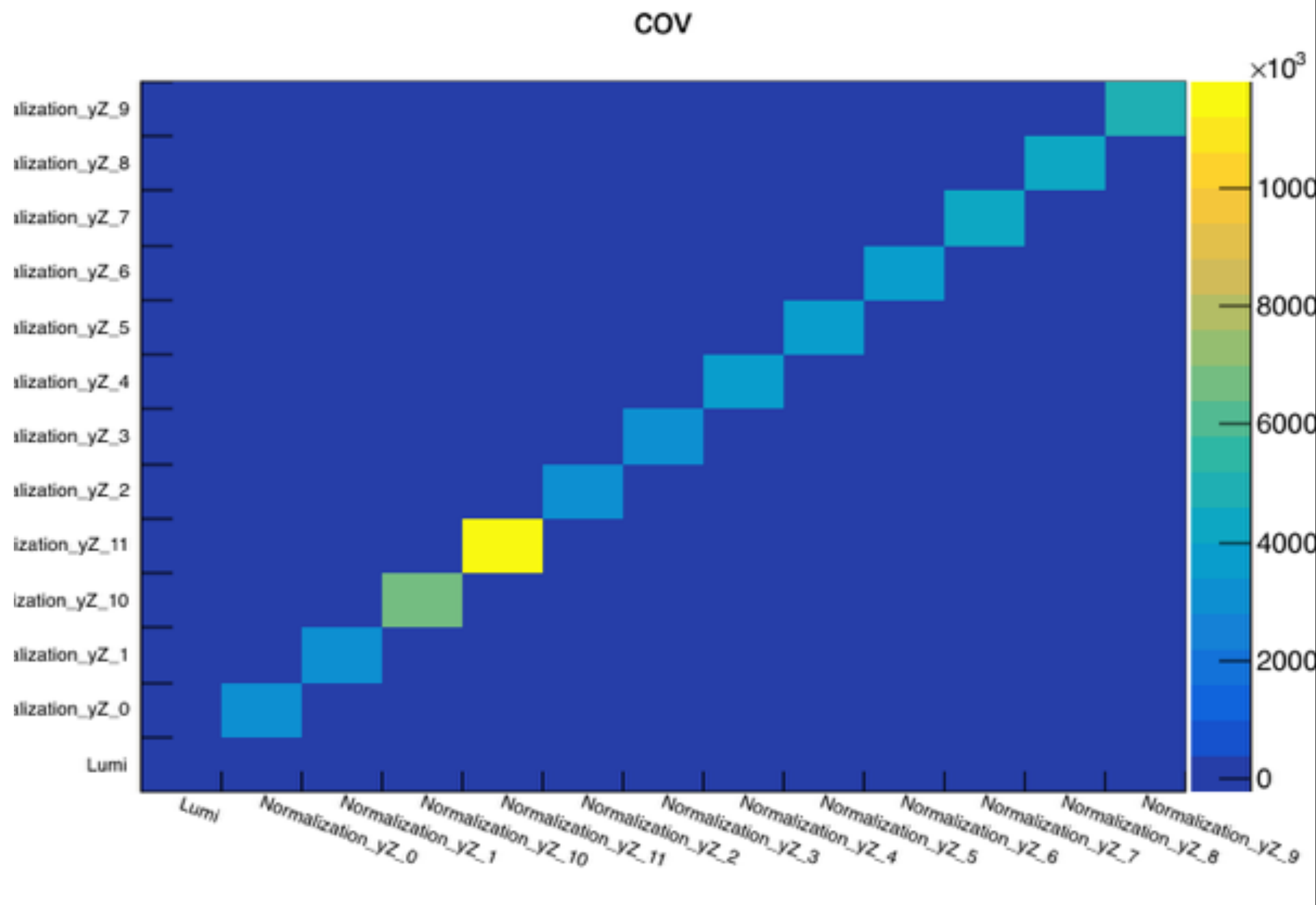
- Problem: Not everyone uses RooFit! How can I share my result with someone if I can't pass them my workspace with the data and the likelihood?
- Solution: In principle most measurements can be fully described by their central values and the full covariance matrix with all parameters
- This is easy to extract with RooFit
- With each MINUIT minimization, a RooFitResult can be saved and returned which contains all of the necessary information
- However, the quality of the covariance matrix depends on the fitting strategy you use in MINUIT
- In "strategy 0", the hessian matrix (the inverse of the covariance matrix) is computed along the minimization path, and is generally not very robust
- In "strategy 1", at the end of the fit the quality of the hessian is checked, and if it's better than some specified criteria nothing is done, otherwise an explicit recalculation of the hessian is performed at the minimum
- In "strategy 2", the hessian is always explicitly recalculated at the minimum
- If you want a robust covariance matrix, opt for strategy 2
- More discussion on the difference between MINUIT strategies here:
    - **https://twiki.cern.ch/twiki/bin/view/AtlasProtected/MinuitStrategyChoice**

Practical example: Making a covariance matrix

- Exercise:
  - Run simpleFit.C with strategy 2 (second argument of macro) for the ProfiledUnfolding example
    - **> root -b -q 'simpleFit.C("ProfiledUnfolding",2)'**
    - This saves a RooFitResult to root-files/ProfiledUnfolding/fit.root
  - Open and look through ex/MakeCovarianceMatrix.C
  - Run it: **> root 'ex/MakeCovarianceMatrix.C("ProfiledUnfolding")'**
  - This also writes a histogram of the central values of the measurement under "cen"



```
root [2] cen->Print("all")
TH1.Print Name  = cen, Entries= 13,
 fSumw[0]=0, x=-0.5
 fSumw[1]=1.00226, x=0.5
 fSumw[2]=1.44184e+06, x=1.5
 fSumw[3]=1.44014e+06, x=2.5
 fSumw[4]=1.24334e+06, x=3.5
 fSumw[5]=1.19181e+06, x=4.5
 fSumw[6]=1.43366e+06, x=5.5
 fSumw[7]=1.42401e+06, x=6.5
 fSumw[8]=1.40999e+06, x=7.5
 fSumw[9]=1.39374e+06, x=8.5
 fSumw[10]=1.37451e+06, x=9.5
 fSumw[11]=1.3488e+06, x=10.5
 fSumw[12]=1.32375e+06, x=11.5
 fSumw[13]=1.28466e+06, x=12.5
 fSumw[14]=0, x=13.5
root [3]
```



```
void MakeCovarianceMatrix(string version)
{
  //Open the RooFitResult of the specified version
  TFile f(Form("root-files/%s/fit.root",version.c_str()));
  RooFitResult* rfr = (RooFitResult*)f.Get("result");

  //Grab the covariance matrix
  TMatrixDSym covMat = rfr->covarianceMatrix();
  RooArgList pars = rfr->floatParsFinal();

  //Fill the histogram. That's all!
  TFile* file = new TFile(Form("root-files/%s/cov.root",version.c_str()),"recreate");
  TH2D* covHist = new TH2D("cov","cov",pars.getSize(),0,pars.getSize(),
                            pars.getSize(),0,pars.getSize());

  //Also save a histogram of the central values
  TH1D* cen = new TH1D("cen","cen",pars.getSize(),0,pars.getSize());
  for (int i=0;i<pars.getSize();i++)
  {
    covHist->GetXaxis()->SetBinLabel(i+1,pars.at(i)->GetName());
    covHist->GetYaxis()->SetBinLabel(i+1,pars.at(i)->GetName());
    for (int j=0;j<pars.getSize();j++)
    {
      covHist->SetBinContent(i+1,j+1,covMat[i][j]);
    }
    cen->GetXaxis()->SetBinLabel(i+1,pars.at(i)->GetName());
    cen->SetBinContent(i+1,((RooRealVar*)pars.at(i))->getVal());
  }


  //Write to file
  file->Write();

  covHist->Draw("colz");
}
```

Building a likelihood from a given covariance matrix

- Problem: Not everyone uses RooFit! How can I use someone elses results if they don't give me a workspace with the likelihood and data?
- Solution: Given the covariance and central value of the measurement, you can easily build a RooFit likelihood with the RooMultiVarGaussian class
- A multi-variate Gaussian is just a Gaussian representation of N measurements **μ**, along with their covariance matrix **Σ**, and observables **x**

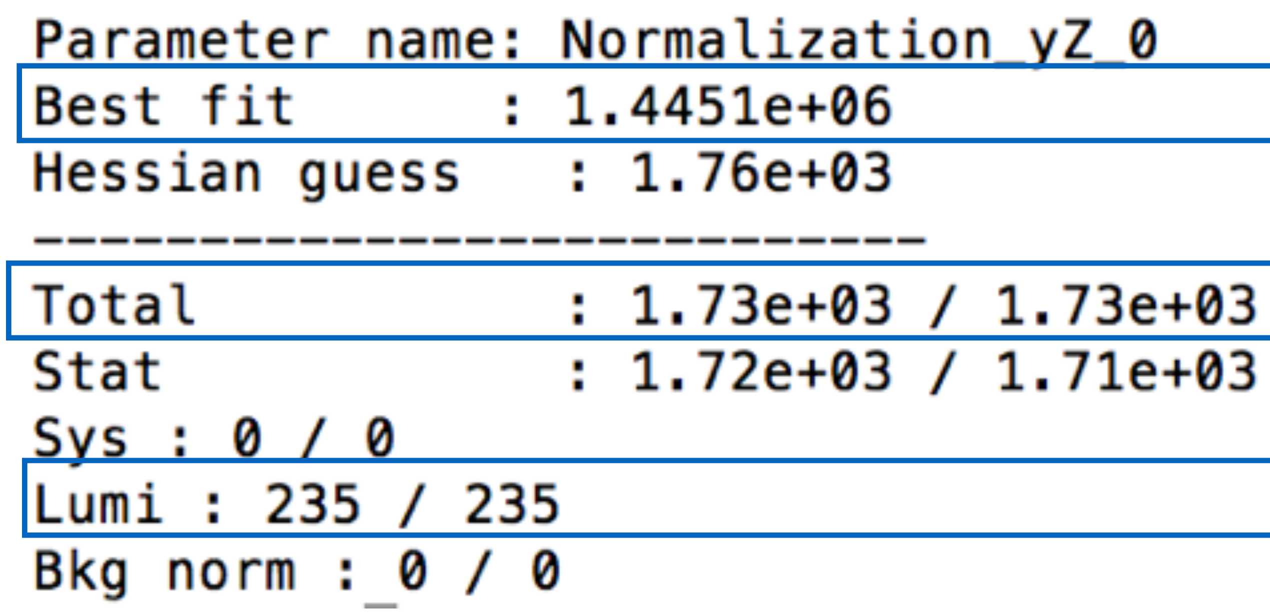
$$f_{\mathbf{x}}(x_1,\ldots,x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}$$

- **x** is like your POIs, **μ** is like your data, and **Σ** is exactly what you extracted in the last example

Practical example: Building a likelihood from a given covariance matrix

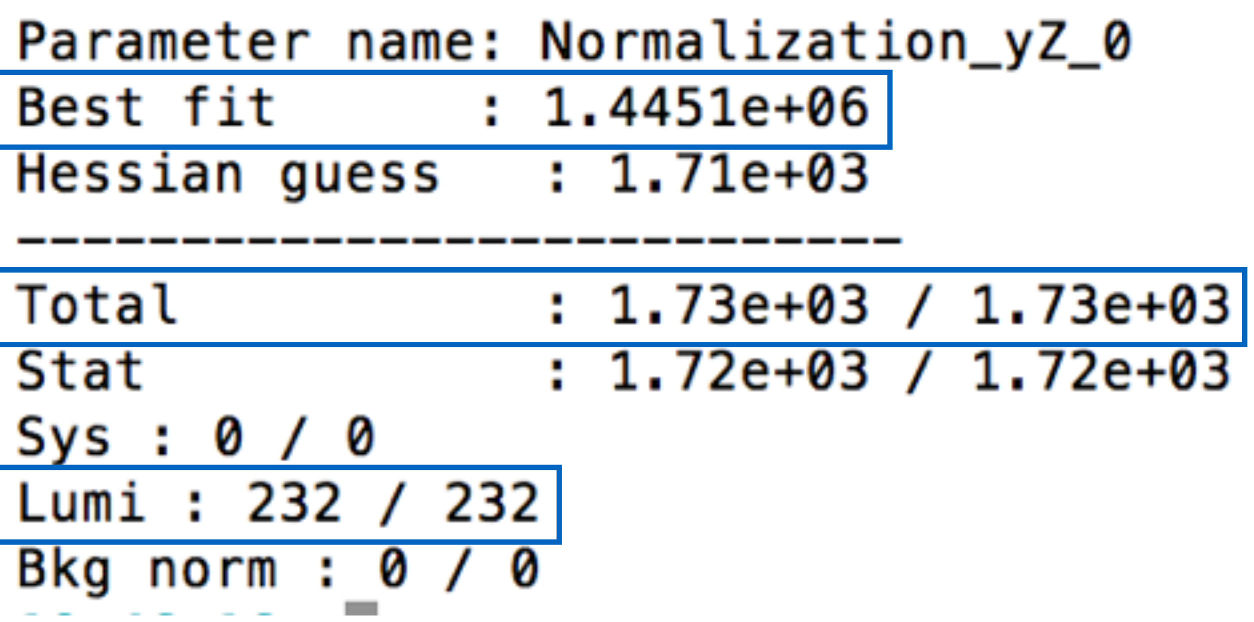
- Exercise: Take the covariance matrix you built last example and turn it back into a likelihood with data
  - Open ex/MultiVarGaussian.C and look around
  - Run it over the covariance matrix from the yZ measurement you just made
    - **> root -b -q 'ex/MultiVarGaussian.C("ProfiledUnfolding")'**
  - This makes a workspace with version name "ProfiledUnfolding_MVG"
  - Run getErrorWithBreakdown over the exact version and the MVG you just made to see how they compare
- **> root -b -q 'macros/getErrorWithBreakdown.C("ProfiledUnfolding","Normalization_yZ_0")'**
- **> root -b -q 'macros/getErrorWithBreakdown.C("ProfiledUnfolding_MVG","Normalization_yZ_0")'**

From "ProfiledUnfolding":



```
Parameter name: Normalization_yZ_0
Best fit        : 1.4451e+06
Hessian guess   : 1.76e+03
---------------------------
Total           : 1.73e+03 / 1.73e+03
Stat            : 1.72e+03 / 1.71e+03
Sys : 0 / 0
Lumi : 235 / 235
Bkg norm : 0 / 0
```

From "ProfiledUnfolding_MVG":



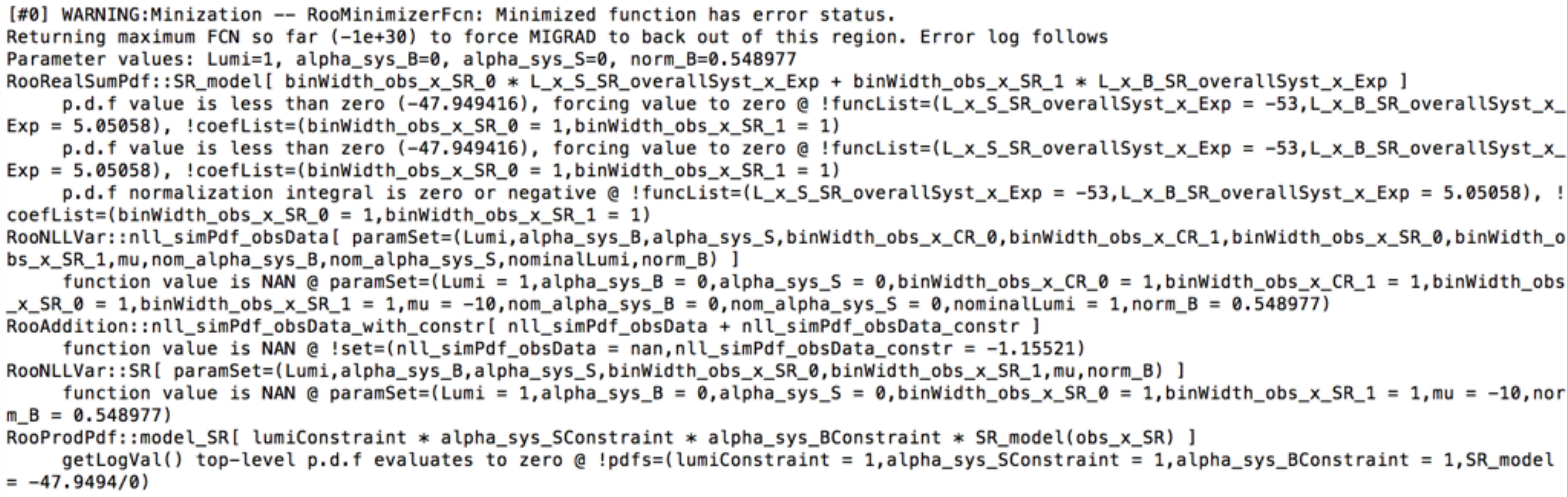
```
Parameter name: Normalization_yZ_0
Best fit        : 1.4451e+06
Hessian guess   : 1.71e+03
---------------------------
Total           : 1.73e+03 / 1.73e+03
Stat            : 1.72e+03 / 1.72e+03
Sys : 0 / 0
Lumi : 232 / 232
Bkg norm : 0 / 0
```

Other tools

- There are many other tools for other things you might want to show that I haven't covered, for example
  - Running toys
  - Computing upper limits
  - Computing the statistical significance of the bkg-only hypothesis
  - Plotting PDFs and data from the workspace
  - Computing the chi2 of the fit wrt data
  - Bayesian statistics
  - Building unbinned likelihoods
  - The list goes on…
- Many examples of these in the built-in RooFit and RooStats tutorials under $ROOTSYS/roofit and $ROOTSYS/roostats
- Many tools available on the Stats forum and HComb TWiki pages
  - **https://twiki.cern.ch/twiki/bin/view/AtlasProtected/StatisticsTools**
  - **https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/HiggsProperties**

Debugging and common problems

- If you get stuck, there are mailing lists for further help and support
  - General statistics questions: hn-atlas-physics-Statistics@cern.ch
  - RooFit/RooStats support: atlas-phys-stat-root@cern.ch
- HistFactory manual: **https://twiki.cern.ch/twiki/bin/view/RooStats/HistFactory**
- The most common problems are when RooFit starts to spit out error messages during a fit, like



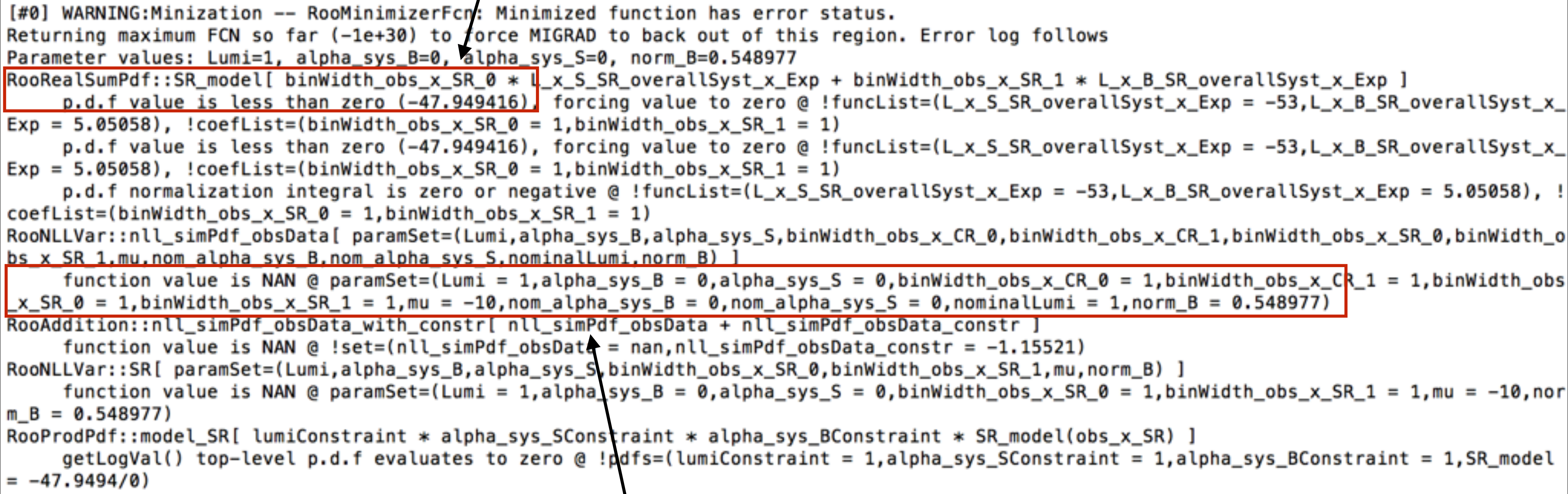
```
[#0] WARNING:Minization -- RooMinimizerFcn: Minimized function has error status.
Returning maximum FCN so far (-1e+30) to force MIGRAD to back out of this region. Error log follows
Parameter values: Lumi=1, alpha_sys_B=0, alpha_sys_S=0, norm_B=0.548977
RooRealSumPdf::SR_model[ binWidth_obs_x_SR_0 * L_x_S_SR_overallSyst_x_Exp + binWidth_obs_x_SR_1 * L_x_B_SR_overallSyst_x_Exp ]
     p.d.f value is less than zero (-47.949416), forcing value to zero @ !funcList=(L_x_S_SR_overallSyst_x_Exp = -53,L_x_B_SR_overallSyst_x_
Exp = 5.05058), !coefList=(binWidth_obs_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1)
     p.d.f value is less than zero (-47.949416), forcing value to zero @ !funcList=(L_x_S_SR_overallSyst_x_Exp = -53,L_x_B_SR_overallSyst_x_
Exp = 5.05058), !coefList=(binWidth_obs_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1)
     p.d.f normalization integral is zero or negative @ !funcList=(L_x_S_SR_overallSyst_x_Exp = -53,L_x_B_SR_overallSyst_x_Exp = 5.05058), !
coefList=(binWidth_obs_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1)
RooNLLVar::nll_simPdf_obsData[ paramSet=(Lumi,alpha_sys_B,alpha_sys_S,binWidth_obs_x_CR_0,binWidth_obs_x_CR_1,binWidth_obs_x_SR_0,binWidth_o
bs_x_SR_1,mu,nom_alpha_sys_B,nom_alpha_sys_S,nominalLumi,norm_B) ]
     function value is NAN @ paramSet=(Lumi = 1,alpha_sys_B = 0,alpha_sys_S = 0,binWidth_obs_x_CR_0 = 1,binWidth_obs_x_CR_1 = 1,binWidth_obs
_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1,mu = -10,nom_alpha_sys_B = 0,nom_alpha_sys_S = 0,nominalLumi = 1,norm_B = 0.548977)
RooAddition::nll_simPdf_obsData_with_constr[ nll_simPdf_obsData + nll_simPdf_obsData_constr ]
     function value is NAN @ !set=(nll_simPdf_obsData = nan,nll_simPdf_obsData_constr = -1.15521)
RooNLLVar::SR[ paramSet=(Lumi,alpha_sys_B,alpha_sys_S,binWidth_obs_x_SR_0,binWidth_obs_x_SR_1,mu,norm_B) ]
     function value is NAN @ paramSet=(Lumi = 1,alpha_sys_B = 0,alpha_sys_S = 0,binWidth_obs_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1,mu = -10,nor
m_B = 0.548977)
RooProdPdf::model_SR[ lumiConstraint * alpha_sys_SConstraint * alpha_sys_BConstraint * SR_model(obs_x_SR) ]
     getLogVal() top-level p.d.f evaluates to zero @ !pdfs=(lumiConstraint = 1,alpha_sys_SConstraint = 1,alpha_sys_BConstraint = 1,SR_model
= -47.9494/0)
```

- This means that MINUIT is trying to probe an unphysical region of the likelihood, ie trying to compute $P(N \mid S+B \leq 0)$
  - This can result from a histogram with expected <= 0 but observed > 0 for some bins
  - This can happen when the above happens under a systematic variation, despite nominal being OK
  - This can happen when the initial values of the parameters are in this unphysical region
- The error message, through difficult to parse, gives you some hints about the problem

Debugging and common problems

- The error message, through difficult to parse, gives you some hints about the problem

Here it shows which channel is giving you the negative PDF



```
[#0] WARNING:Minization -- RooMinimizerFcn: Minimized function has error status.
Returning maximum FCN so far (-1e+30) to force MIGRAD to back out of this region. Error log follows
Parameter values: Lumi=1, alpha_sys_B=0, alpha_sys_S=0, norm_B=0.548977
RooRealSumPdf::SR_model[ binWidth_obs_x_SR_0 * L_x_S_SR_overallSyst_x_Exp + binWidth_obs_x_SR_1 * L_x_B_SR_overallSyst_x_Exp ]
     p.d.f value is less than zero (-47.949416), forcing value to zero @ !funcList=(L_x_S_SR_overallSyst_x_Exp = -53,L_x_B_SR_overallSyst_x_
Exp = 5.05058), !coefList=(binWidth_obs_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1)
     p.d.f value is less than zero (-47.949416), forcing value to zero @ !funcList=(L_x_S_SR_overallSyst_x_Exp = -53,L_x_B_SR_overallSyst_x_
Exp = 5.05058), !coefList=(binWidth_obs_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1)
     p.d.f normalization integral is zero or negative @ !funcList=(L_x_S_SR_overallSyst_x_Exp = -53,L_x_B_SR_overallSyst_x_Exp = 5.05058), !
coefList=(binWidth_obs_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1)
RooNLLVar::nll_simPdf_obsData[ paramSet=(Lumi,alpha_sys_B,alpha_sys_S,binWidth_obs_x_CR_0,binWidth_obs_x_CR_1,binWidth_obs_x_SR_0,binWidth_o
bs_x_SR_1,mu,nom_alpha_sys_B,nom_alpha_sys_S,nominalLumi,norm_B) ]
     function value is NAN @ paramSet=(Lumi = 1,alpha_sys_B = 0,alpha_sys_S = 0,binWidth_obs_x_CR_0 = 1,binWidth_obs_x_CR_1 = 1,binWidth_obs
_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1,mu = -10,nom_alpha_sys_B = 0,nom_alpha_sys_S = 0,nominalLumi = 1,norm_B = 0.548977)
RooAddition::nll_simPdf_obsData_with_constr[ nll_simPdf_obsData + nll_simPdf_obsData_constr ]
     function value is NAN @ !set=(nll_simPdf_obsData = nan,nll_simPdf_obsData_constr = -1.15521)
RooNLLVar::SR[ paramSet=(Lumi,alpha_sys_B,alpha_sys_S,binWidth_obs_x_SR_0,binWidth_obs_x_SR_1,mu,norm_B) ]
     function value is NAN @ paramSet=(Lumi = 1,alpha_sys_B = 0,alpha_sys_S = 0,binWidth_obs_x_SR_0 = 1,binWidth_obs_x_SR_1 = 1,mu = -10,nor
m_B = 0.548977)
RooProdPdf::model_SR[ lumiConstraint * alpha_sys_SConstraint * alpha_sys_BConstraint * SR_model(obs_x_SR) ]
     getLogVal() top-level p.d.f evaluates to zero @ !pdfs=(lumiConstraint = 1,alpha_sys_SConstraint = 1,alpha_sys_BConstraint = 1,SR_model
= -47.9494/0)
```

Here it shows the value of all parameters where the PDF is negative

- If you encounter a problem like this, remember that you can open the workspace and work on the command line to explore and debug, set parameter values, evaluate pdfs, etc
- If you're up to the challenge, digging into the RooFit source code, adding more verbose print statements, and playing around can be very informative, and not as difficult as you might think!