



RooStats (Part 2)

Lorenzo Moneta
(CERN, PH-SFT)



Introduction



- ❖ New project to create advanced statistical tools needed by the LHC experiments
- ❖ Joint contribution ATLAS, CMS, ROOT and RooFit
 - ◆ core developers:
 - ◆ K.Cranmer (ATLAS), G. Schott (CMS), W. Verkerke (RooFit), L. Moneta (ROOT)
 - ◆ other contributors :
 - ◆ D. Piparo, M. Pelliccioni (CMS), A. Lazzaro (ATLAS)
- ❖ Developments monitored by ATLAS and CMS statistic committees
 - ◆ implement the accepted methods
- ❖ Included in ROOT since version 5.22



RooStats Project



❖ Goals:

- ❖ standardize interface for major statistical procedures
- ❖ can work on an arbitrary model and dataset and handle many parameters of interest and nuisance parameters
- ❖ implement most accepted techniques from Frequentist, Bayesian, and Likelihood-based approaches
- ❖ provide utilities to perform combined measurements

❖ Design:

- ❖ base on RooFit to construct models
 - ❖ all statistical methods start from description of probability density function or likelihood function
- ❖ build series of tools that perform statistical procedures on RooFit models



RooStats Structure



❖ RooFit (data modeling)

- ❖ Data modeling language (pdfs and likelihoods).
Scales to arbitrary complexity
- ❖ Support for efficient integration, toy MC generation
- ❖ Workspace
 - ❖ Persistent container for data models
 - ❖ Completely self-contained (including custom code)
 - ❖ Complete introspection and access to components
- ❖ Workspace factory provides easy scripting language to populate the workspace

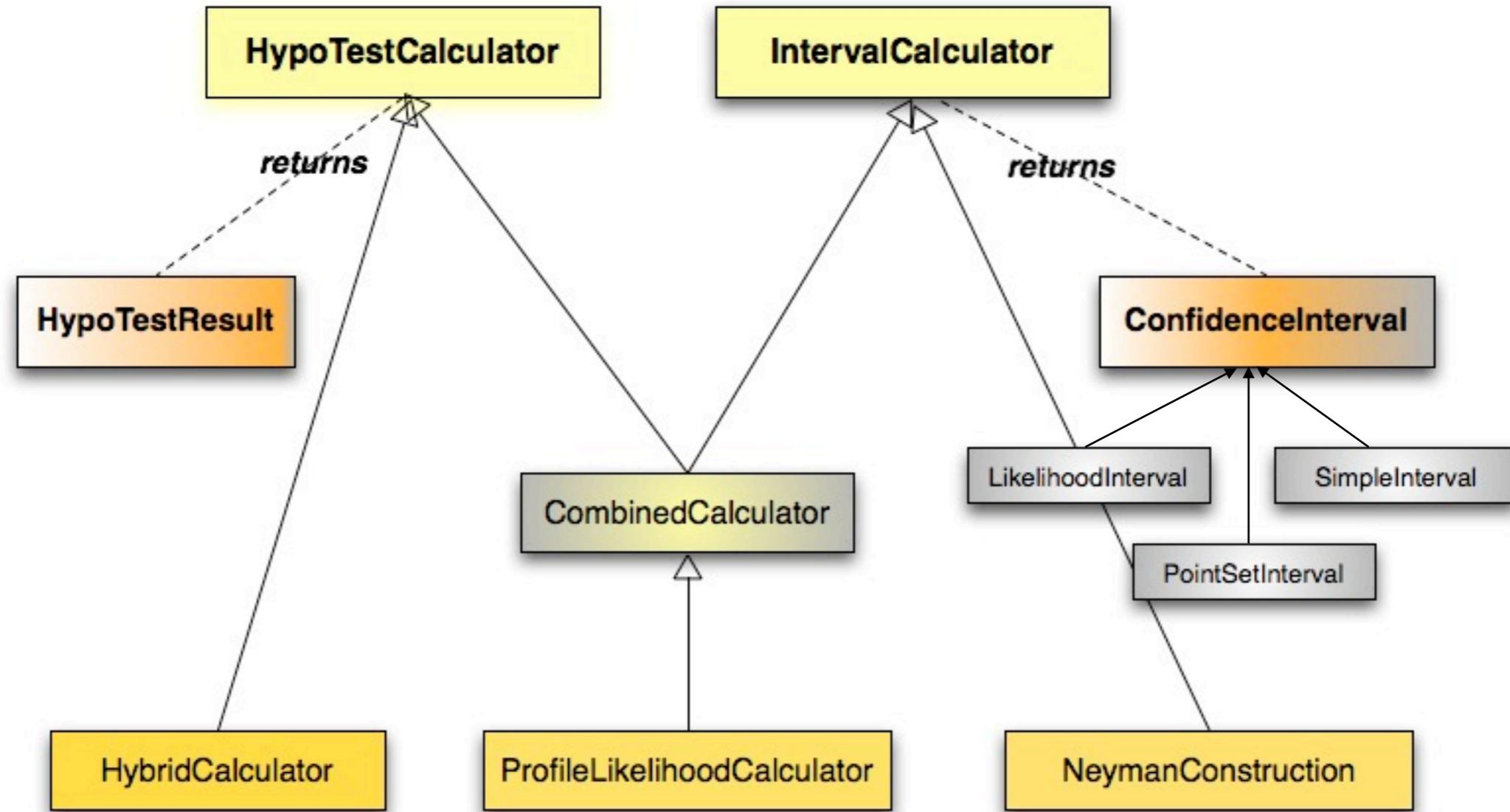
❖ RooStats

- ❖ statistical methods using frequentists or bayesian statistics for calculating interval, limits and performing hypothesis tests
- ❖ Utilities for combinations and to construct pdf's corresponding to standard number counting problems

RooStats Interfaces



- ❖ Interfaces to statistical tools
 - ❖ direct mapping to statistical concepts





IntervalCalculator interface



```
class IntervalCalculator {  
  
public:  
  
    virtual ~IntervalCalculator() {}  
  
    // Main interface to get a ConfInterval, pure virtual  
    virtual ConfInterval* GetInterval() const = 0;  
  
    // Get the size of the test (eg. rate of Type I error)  
    virtual Double_t Size() const = 0;  
  
    // Get the Confidence level for the test  
    virtual Double_t ConfidenceLevel() const = 0;  
  
    // Set the DataSet ( add to the the workspace if not already there ?)  
    virtual void SetData(RooAbsData&) = 0;  
  
    // Set the Model  
    virtual void SetModel(const ModelConfig & /* model */) = 0;  
  
    // set the size of the test (rate of Type I error) ( e.g. 0.05 for a 95% Confidence Interval)  
    virtual void SetTestSize(Double_t size) = 0;  
  
    // set the confidence level for the interval (e.g. 0.95 for a 95% Confidence Interval)  
    virtual void SetConfidenceLevel(Double_t cl) = 0;  
  
protected:  
    ClassDef(IntervalCalculator,1) // Interface for tools setting limits (producing confidence intervals)  
};
```



HypoTestCalculator



```
class HypoTestCalculator {  
  
public:  
  
    virtual ~HypoTestCalculator() {}  
  
    // main interface to get a HypoTestResult, pure virtual  
    virtual HypoTestResult* GetHypoTest() const = 0;  
  
    // Set a common model for both the null and alternate, add to the workspace if not already there  
    virtual void SetCommonModel(const ModelConfig& model) {  
        SetNullModel(model);  
        SetAlternateModel(model);  
    }  
  
    // Set the model for the null hypothesis  
    virtual void SetNullModel(const ModelConfig& model) = 0;  
    // Set the model for the alternate hypothesis  
    virtual void SetAlternateModel(const ModelConfig& model) = 0;  
    // Set the DataSet  
    virtual void SetData(RooAbsData& data) = 0;  
  
protected:  
    ClassDef(HypoTestCalculator,1) // Interface for tools doing hypothesis tests  
};
```



Conflnterval Interface



```
namespace RooStats {

    class ConfInterval : public TNamed {

        public:

            // constructor given name and title
            explicit ConfInterval(const char* name = 0) : TNamed(name,name) {}

            // destructor
            virtual ~ConfInterval() {}

            // check if given point is in the interval
            virtual Bool_t IsInInterval(const RooArgSet&) const = 0;

            // used to set confidence level. Keep pure virtual
            virtual void SetConfidenceLevel(Double_t cl) = 0;

            // return confidence level
            virtual Double_t ConfidenceLevel() const = 0;

            // return list of parameters of interest defining this interval (return a new cloned list)
            virtual RooArgSet* GetParameters() const = 0;

            // check if parameters are correct (i.e. they are the POI of this interval)
            virtual Bool_t CheckParameters(const RooArgSet&) const = 0;

        protected:

            ClassDef(ConfInterval,1) // Interface for Confidence Intervals

    };
}
```



HypoTestResult Interface



```
namespace RooStats {  
  
    class HypoTestResult : public TNamed {  
  
    public:  
  
        // default constructor  
        explicit HypoTestResult(const char* name = 0);  
  
        // constructor from name, null and alternate p values  
        HypoTestResult(const char* name, Double_t nullp, Double_t altp);  
  
        // destructor  
        virtual ~HypoTestResult();  
  
        // Return p-value for null hypothesis  
        virtual Double_t NullPValue() const {return fNullPValue;}  
  
        // Return p-value for alternate hypothesis  
        virtual Double_t AlternatePValue() const {return fAlternatePValue;}  
  
        // Convert NullPValue into a "confidence level"  
        virtual Double_t CLb() const {return 1.-NullPValue();}  
  
        // Convert AlternatePValue into a "confidence level"  
        virtual Double_t CLsplusb() const {return AlternatePValue();}  
  
        // CLs is simply CLs+b/CLb (not a method, but a quantity)  
        virtual Double_t CLs() const;  
  
        // familiar name for the Null p-value in terms of 1-sided Gaussian significance  
        virtual Double_t Significance() const {return RooStats::PValueToSignificance( NullPValue() ); }  
  
    protected:  
  
        mutable Double_t fNullPValue; // p-value for the null hypothesis (small number means disfavored)  
        mutable Double_t fAlternatePValue; // p-value for the alternate hypothesis (small number means disfavored)  
  
        ClassDef(HypoTestResult,1) // Base class to represent results of a hypothesis test  
    };  
}
```



RooStats Calculator classes

- ❖ Profile Likelihood calculator
 - ❖ interval estimation and hypothesis testing
- ❖ Bayesian calculators
 - ❖ BayesianCalculator
 - ❖ analytical or adaptive numerical integration (only 1 dim)
 - ❖ MCMCCalculator (Markov-Chain Monte Carlo)
- ❖ HybridCalculator
 - ❖ frequentist hypothesis test calculator with Cousins-Highland integration for nuisance parameters
- ❖ Neyman construction
 - ❖ interval calculator (returns a PointSetInterval)
- ❖ FeldmanCousins
 - ❖ Neyman construction configured with the Likelihood ratio ordering rule

Profile Likelihood Calculator



- ❖ Method based on properties of the likelihood function

- ❖ Profile likelihood function:

$$\ell = \frac{L(x|\theta_{r0}, \hat{\theta}_s)}{L(x|\hat{\theta}_r, \hat{\theta}_s)}$$

minimize w.r.t nuisance parameters and fix POI
minimize w.r.t. all parameters

ℓ is a function of only the parameter of interest

- ❖ Uses asymptotic properties based on Wilks' theorem:

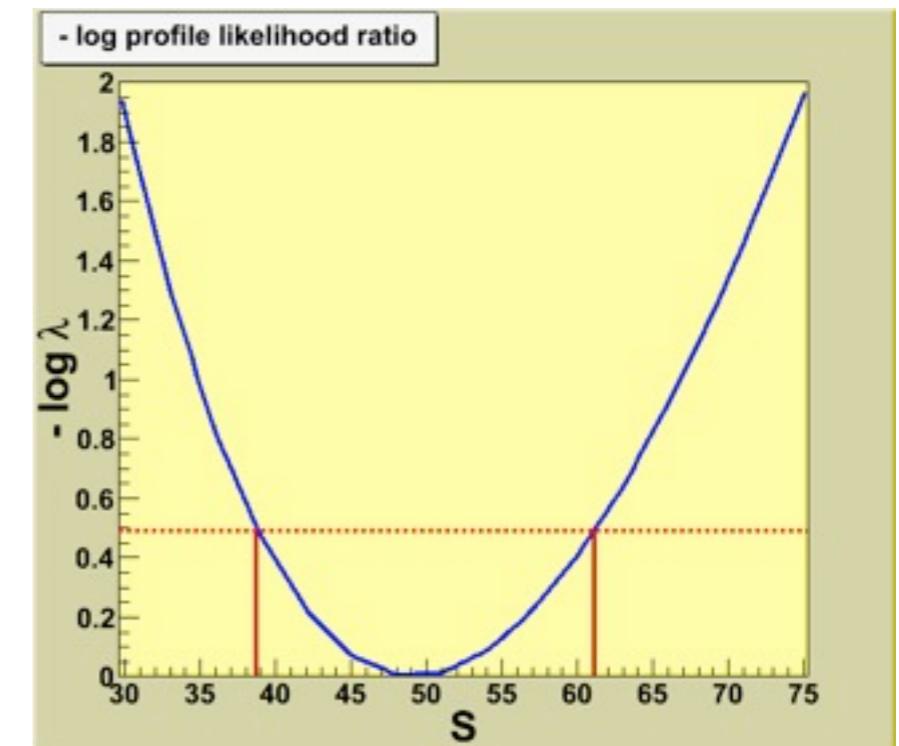
- ❖ ℓ is a gaussian function

- ❖ e.g. $-2\log\ell$ is a parabolic function

- ❖ Method of MINUIT/MINOS

- ❖ lower/upper limits for 1D

- ❖ contours for 2 parameters



Profile Likelihood Calculator (2)



- Profile Likelihood can be used for hypothesis tests using the properties of the likelihood ratio:

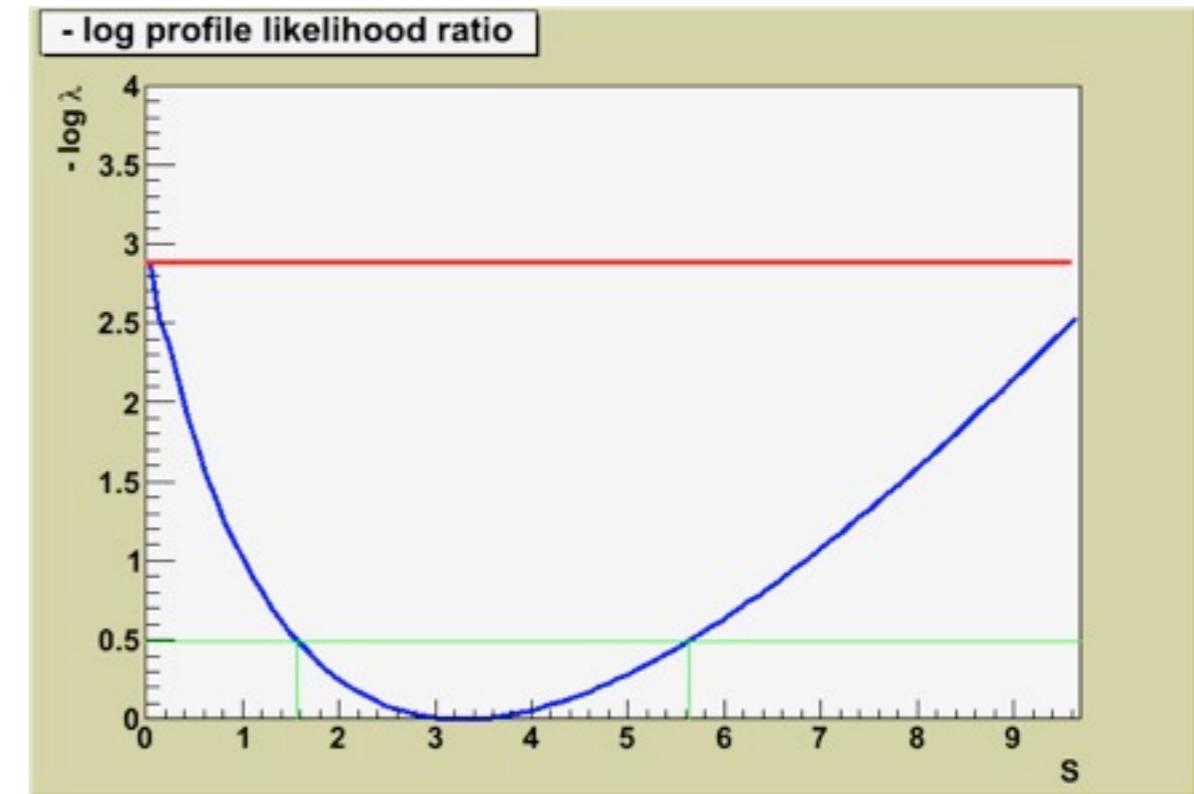
$$\ell = \frac{L(x|\theta_{r0}, \hat{\theta}_s)}{L(x|\hat{\theta}_r, \hat{\theta}_s)}$$

- Null hypothesis (H_0): $\theta = \theta_0$
- Alternate hypothesis (H_1): $\theta \neq \theta_0$

- Due to Wilk theorem, distribution of $-2\log\lambda$ is asymptotically a χ^2 distribution under H_0

◆ p-value and significance are then obtained from the $-2\log\lambda$ ratio

◆ signif. = $\sqrt{(2 * \Delta\log\lambda)}$



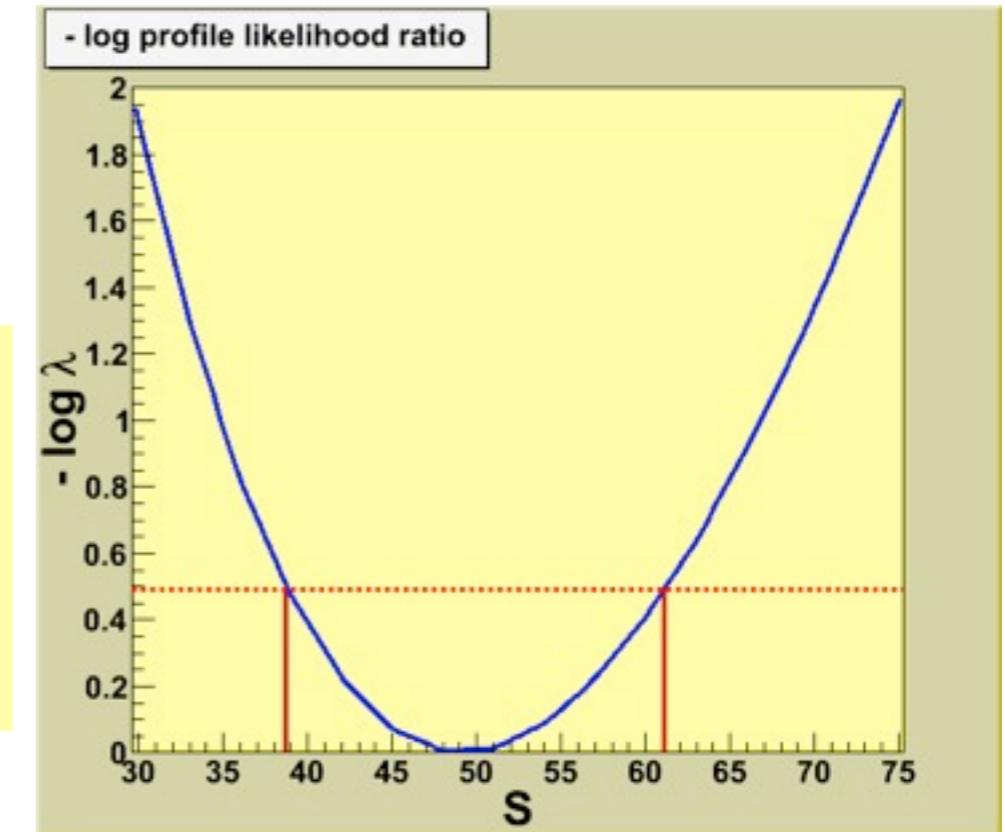


❖ RooStats Code Example

❖ 68% (1 sigma) Interval estimation

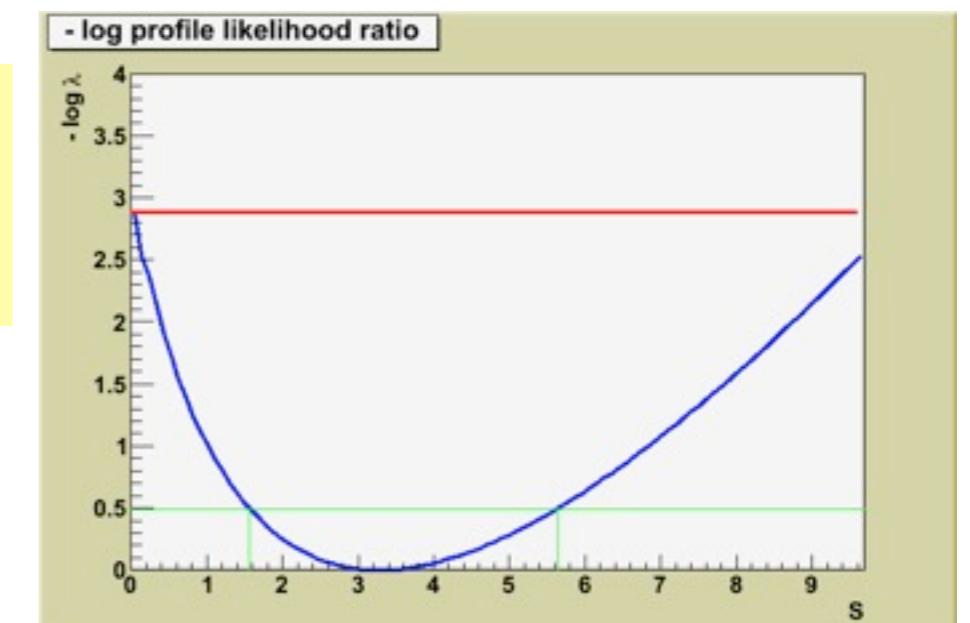
```
ProfileLikelihoodCalculator plc(*data, *model, *POI);
plc.SetTestSize(0.32);
LikelihoodInterval* interval = plc.GetInterval();
double lowerLimit = interval->LowerLimit(*S);
double upperLimit = interval->UpperLimit(*S);

LikelihoodIntervalPlot plot(interval);
plot.Draw();
```



❖ Significance of discovery (S non zero)

```
S->setVal(0); // set value of S to zero
plc.SetNullParameters(RooArgSet(*S));
HypoTestResult* hypotest = plc.GetHypoTest();
double significance = hypotest->Significance();
```



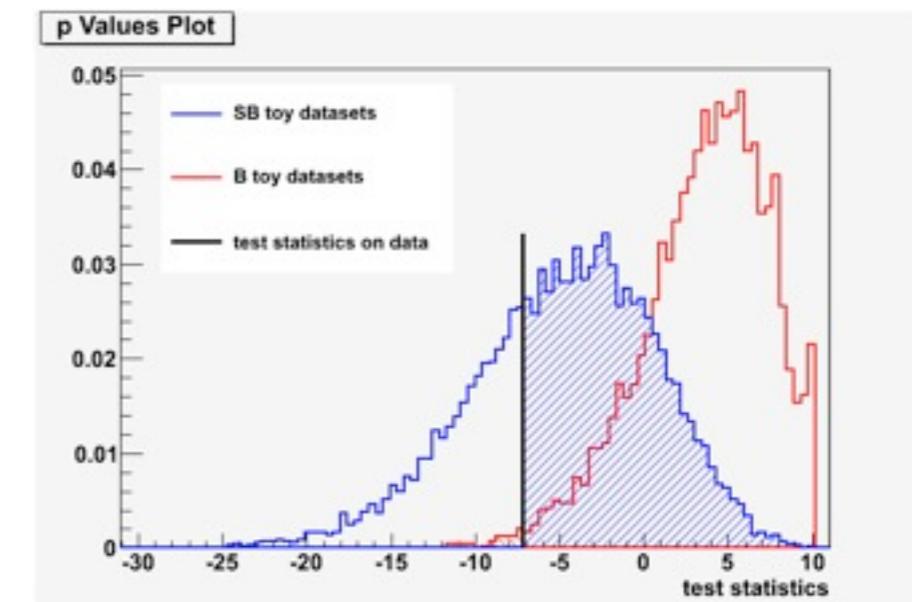


Hybrid Calculator

- ❖ Hypothesis test calculator (method used at LEP)
- ❖ uses by default the likelihood ratio as test statistics Q

$$Q(m_H) = \frac{L(s+b)}{L(b)}$$

- ❖ optionally one can use the number of events or the profile likelihood ratio
- ❖ generate toy MC experiments to obtain the distribution of Q
- ❖ From the empirical distribution obtained estimate the p-values:
 - ❖ CL_{sb} and CL_b
 - ❖ CL_s is obtained as the ratio CL_{sb} / CL_b
- ❖ Frequentist method
- ❖ Cousins-Highland integration over the nuisance parameters (Bayesian)
 - ❖ toys are generated with nuisance parameters smeared according to their distribution





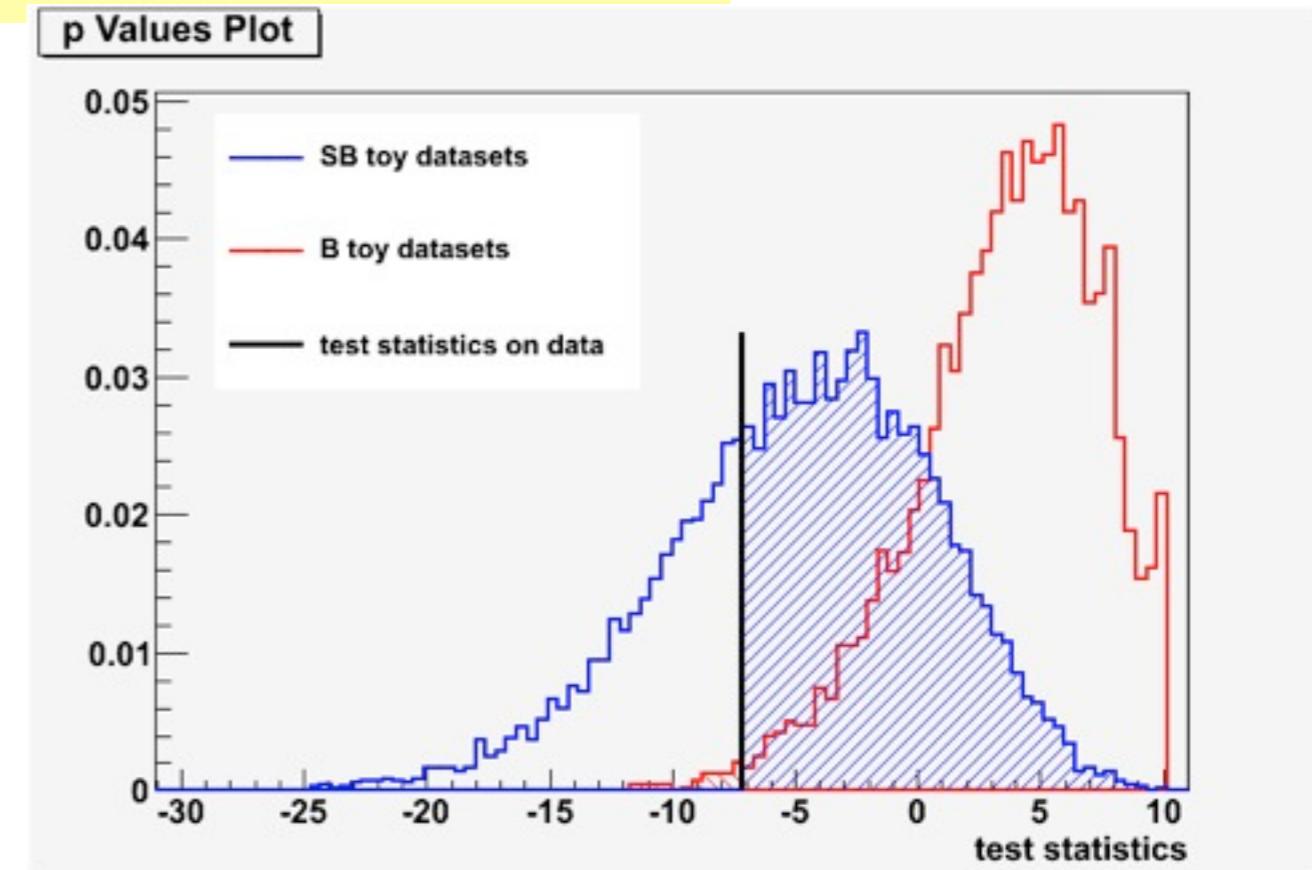
Hybrid Calculator



```
HybridCalculator * hc=new HybridCalculator(*data,*modelFull,*modelBkg)
hc->SetNumber0fToys(1000)
// if using systematics
hc->SetNuisancePdf(*priorNuisance);

HybridResult* result=hc->GetHypoTest();
HybridPlot* plot=result->GetPlot("hcPlot","p Values Plot",100);
plot->Draw();

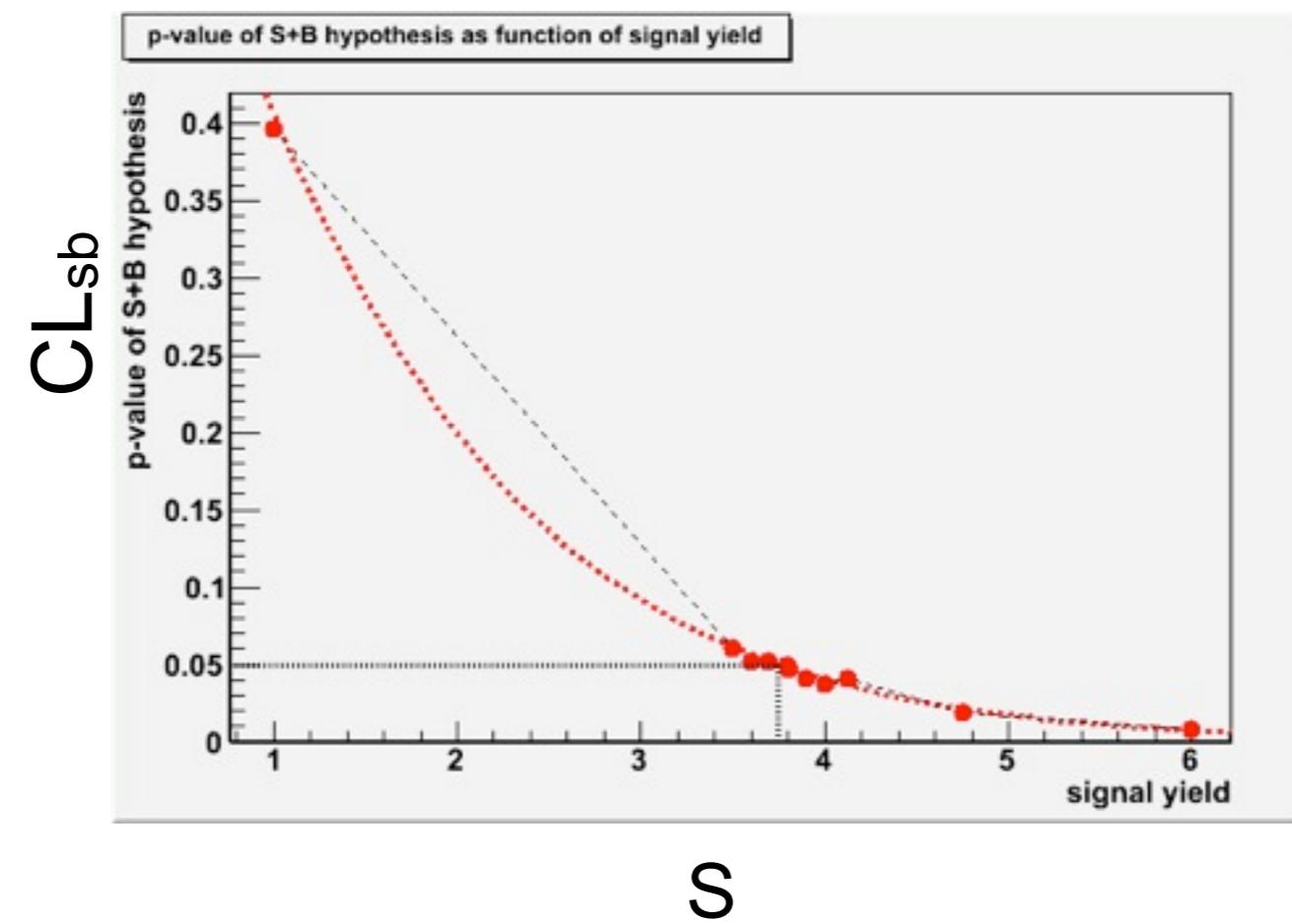
// retrieve CLsb, CLb and CLs
double clsb_data = hcresult->CLsplusb();
double clb_data = hcresult->CLb();
double cls_data = hcresult->CLs();
double data_significance = hcresult->Significance();
```



Hypothesis Inverter



- ❖ Class to invert result from an hypothesis test calculator and compute a limit (RooStats::HypoTestInverter)
 - ❖ working on Hybrid calculator
 - ❖ run many runs of the calculator scanning the POI
 - ❖ obtain an upper limit from the CL curve

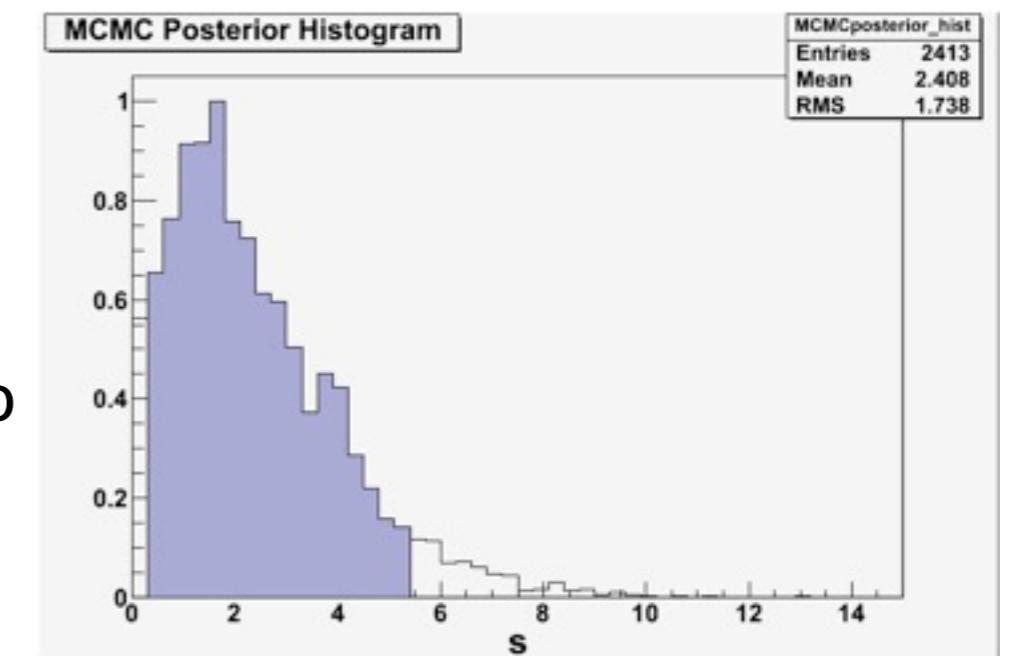
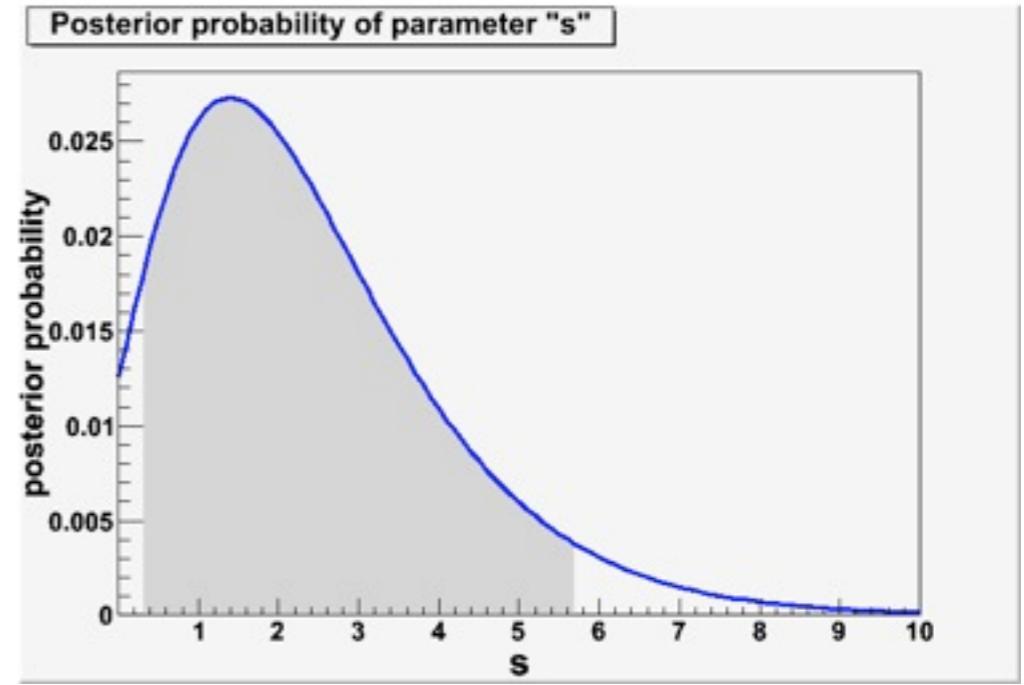


Bayesian Calculator



- ❖ BayesianCalculator class
 - ❖ posterior and interval estimation using numerical integration
 - ❖ current implementation works only for one parameter of interest

```
BayesianCalculator bc(*data, *modelPdf, *POI);
bc.SetTestSize(0.05);
SimpleInterval* interval = bc.GetInterval();
double lowerLimit = interval->LowerLimit(*S);
double upperLimit = interval->UpperLimit(*S);
(bcCalc.GetPosteriorPlot())->Draw();
```



- ❖ MCMCCalculator
 - ❖ integration using Markov-Chain Monte Carlo
 - ❖ possible to specify ProposalFunction
 - ❖ can visualize posterior and also the chain

Markov Chain Monte Carlo



Markov Chain Monte Carlo (MCMC) is a nice technique which will produce a sampling of a parameter space which is proportional to a posterior

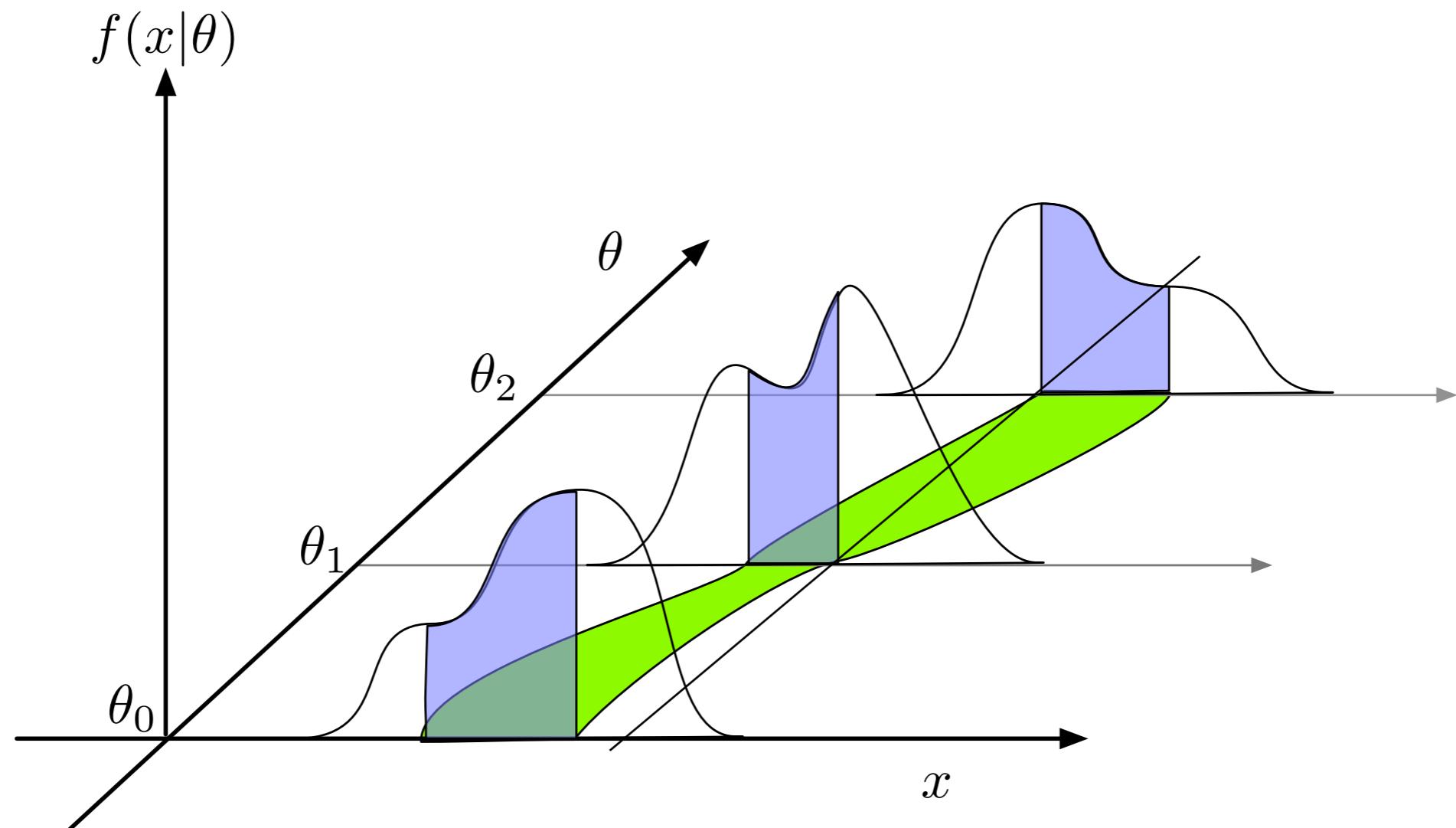
- it works well in high dimensional problems
- Metropolis–Hastings Algorithm: generates a sequence of points $\{\vec{\alpha}^{(t)}\}$
 - Given the likelihood function $L(\vec{\alpha})$ & prior $P(\vec{\alpha})$, the posterior is proportional to $L(\vec{\alpha}) \cdot P(\vec{\alpha})$
 - propose a point $\vec{\alpha}'$ to be added to the chain according to a proposal density $Q(\vec{\alpha}'|\vec{\alpha})$ that depends only on current point $\vec{\alpha}$
 - if posterior is higher at $\vec{\alpha}'$ than at $\vec{\alpha}$, then add new point to chain
 - else: add $\vec{\alpha}'$ to the chain with probability
$$\rho = \frac{L(\vec{\alpha}') \cdot P(\vec{\alpha}')}{L(\vec{\alpha}) \cdot P(\vec{\alpha})} \cdot \frac{Q(\vec{\alpha}|\vec{\alpha}')}{Q(\vec{\alpha}'|\vec{\alpha})}$$
(appending original point $\vec{\alpha}$ with complementary probability)
- RooStats works with any $L(\vec{\alpha})$ $P(\vec{\alpha})$
- Can use any RooFit PDF as proposal function $Q(\vec{\alpha}'|\vec{\alpha})$

Work done primarily by Kevin Belasco (Princeton)

Neyman Construction Review



- ◆ Treat each point in parameter space θ independently
- ◆ For each point, need distribution of some test statistic x
- ◆ Choose an **ordering rule** that selects a specific $1 - \alpha$ region
- ◆ Confidence Interval is **set** of parameter points where data in acceptance region (eg. intersects confidence belt)



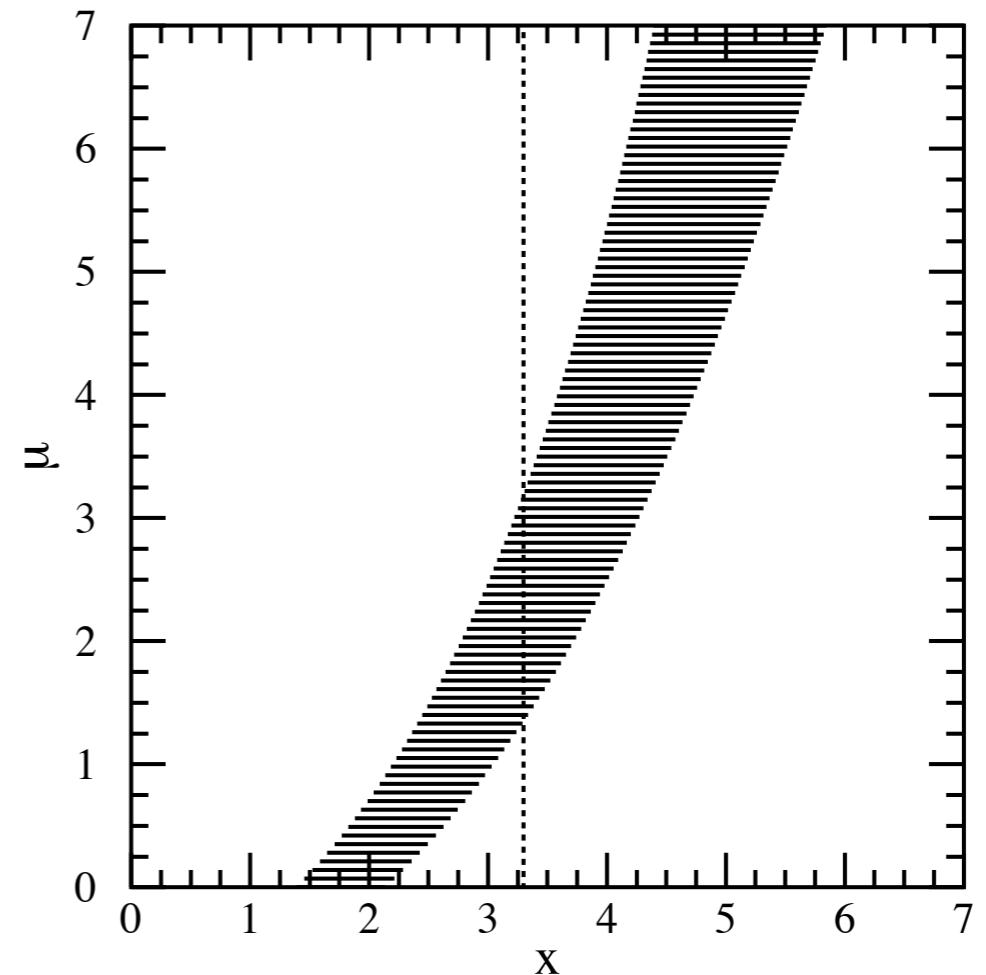
Feldman-Cousins: a special case



Feldman & Cousins “Unified Approach” looks like this:

Neyman Construction

- For each μ : find region R_μ with probability $1 - \alpha$
- Confidence Interval includes all μ consistent with observation at x_0



Ordering Rule specifies what region

F-C ordering rule is the Likelihood Ratio

$$R_\mu = \left\{ x \mid \frac{L(x|\mu)}{L(x|\mu_{\text{best}})} > k_\alpha \right\}$$

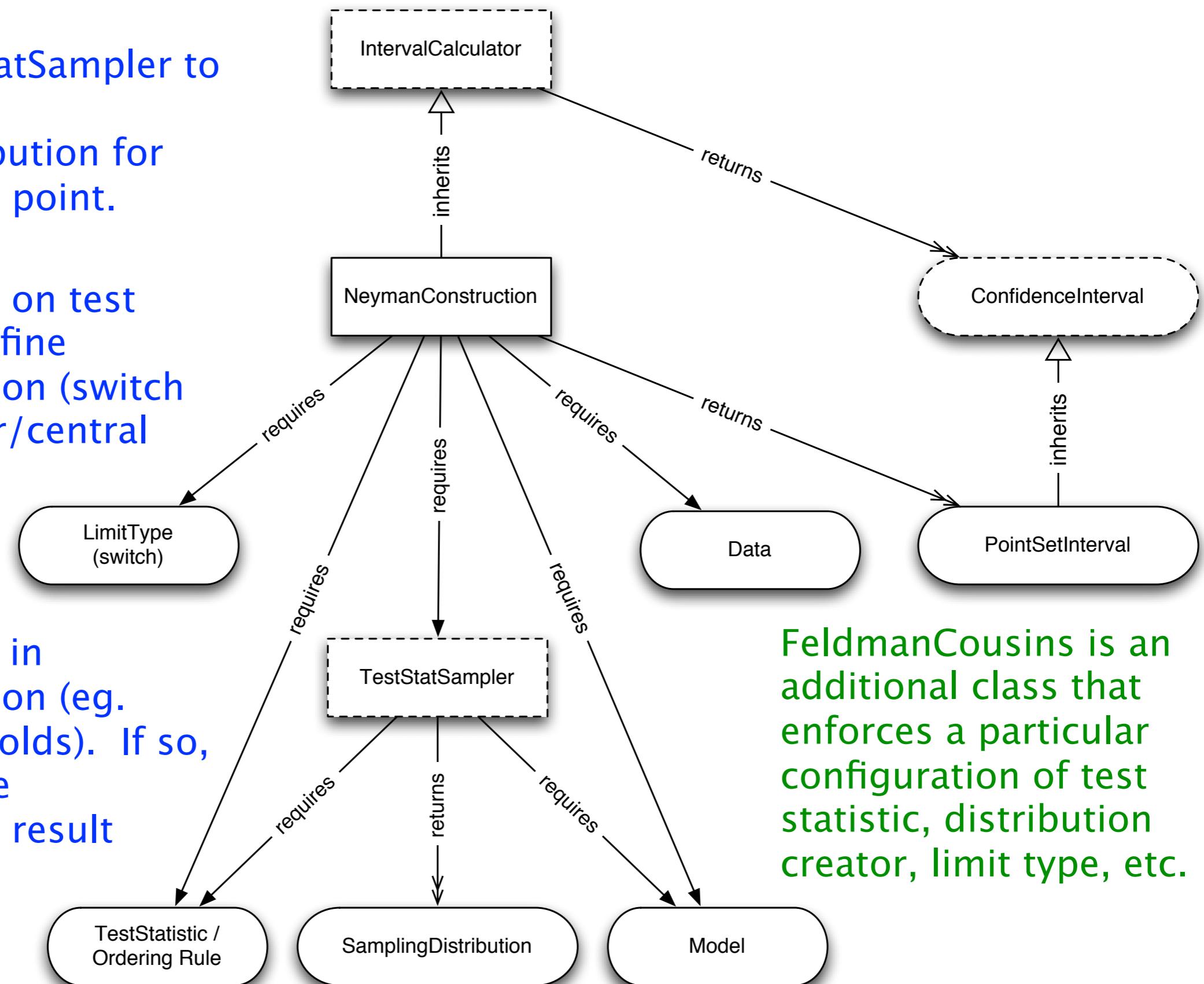
The F-C ordering rule follows naturally from Neyman-Pearson Lemma

NeymanConstruction Implementation



It uses a `TestStatSampler` to generate a `SamplingDistribution` for each parameter point.

Find thresholds on test statistic that define acceptance region (switch for upper/lower/central limits)



Check if data is in acceptance region (eg. between thresholds). If so, add point to the `PointSetInterval` result

`FeldmanCousins` is an additional class that enforces a particular configuration of test statistic, distribution creator, limit type, etc.



Neutrino Oscillation Example

Kyle coded up neutrino oscillation experiment based on description in Feldman & Cousins's original paper.

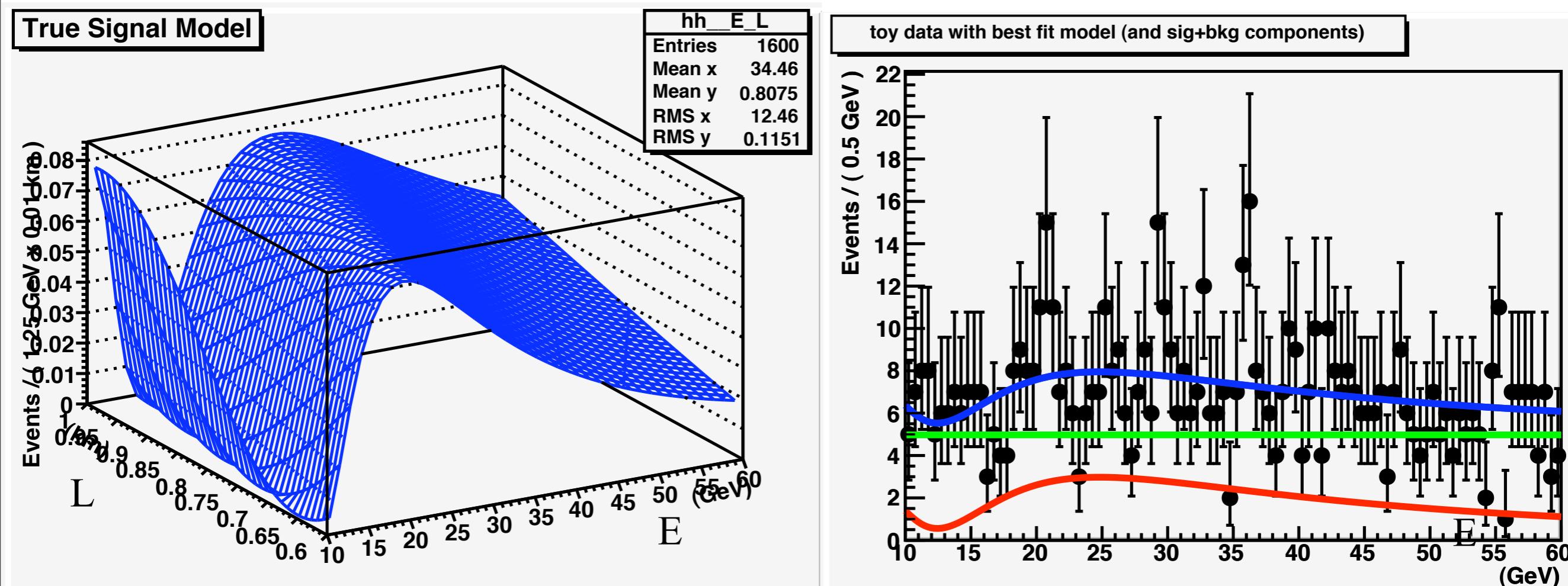
Generate toy data at same true parameters and compare RooStats with results in paper

$$P(\nu_\mu \rightarrow \nu_e) = \sin^2(2\theta) \sin^2\left(\frac{1.27\Delta m^2 L}{E}\right), \quad (5.3)$$

where P is the probability for a ν_μ to transform into a ν_e , L is the distance in km between the creation of the neutrino from meson decay and its interaction in the detector, E is the neutrino energy in GeV, and $\Delta m^2 = |m_1^2 - m_2^2|$ in $(\text{eV}/c^2)^2$.

To demonstrate how this works in practice, and how it compares to alternative approaches that have been used, we consider a toy model of a typical neutrino oscillation experiment. The toy model is defined by the following parameters: Mesons are assumed to decay to neutrinos uniformly in a region 600 m to 1000 m from the detector. The expected background from conventional ν_e interactions and misidentified ν_μ interactions is assumed to be 100 events in each of 5 energy bins which span the region from 10 to 60 GeV. We assume that the ν_μ flux is such that if $P(\nu_\mu \rightarrow \nu_e) = 0.01$ averaged over any bin, then that bin would have an expected additional contribution of 100 events due to $\nu_\mu \rightarrow \nu_e$ oscillations.

http://root.cern.ch/root/html/tutorials/roostats/rs401d_FeldmanCousins.C.html



Example Feldman-Cousins

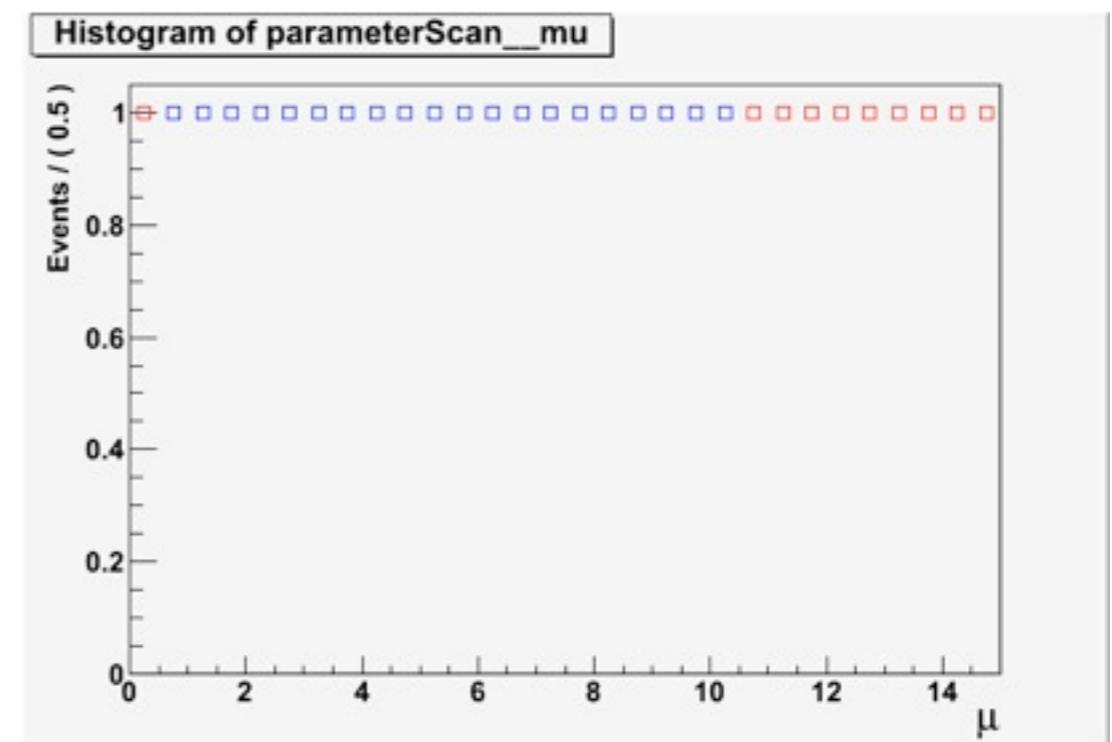


❖ FeldmanCousins class

- ❖ configure like the other calculator class

```
RooStats::FeldmanCousins fc;  
// set the distribution creator, which encodes the test statistic  
fc.SetPdf(pois);  
fc.SetParameters(parameters);  
fc.SetTestSize(.05); // set size of test  
fc.SetData(*data);  
fc.UseAdaptiveSampling(true);  
// number counting analysis: dataset always has 1 entry with N events observed  
fc.FluctuateNumDataEntries(false);  
fc.SetNBins(30); // number of points to test per parameter
```

- ❖ SetNBins(n) specify the number of points to scan in the parameter of interest
- ❖ class returns a ConflInterval object
 - ❖ can only test if a point is inside or not



Working Plan for today



- ❖ Working today on simple Poisson (number counting) and gaussian model (gaussian peak over a flat background)
- ❖ Build model using RooFit (factory class from Workspace)
 - ❖ Workspace used to manage all the pdf's and parameters
- ❖ Run the RooStats calculator classes:
 - ❖ Likelihood method
 - ❖ with and without inclusion of systematic in the nuisance parameters
 - ❖ Hybrid calculator class
- ❖ If still time we will try to use the BayesianCalculator and the MCMCCalculator
- ❖ Tomorrow exercises using Neyman construction (Feldman Cousins)



Hands-on Session

RooStats Exercises

Getting Started



★ A few advice words:

★ Install version 5.25.04 locally

★ setting up

 ★ . root_installation_directory/bin/thisroot.sh (or .csh)

★ recommend to compile your macros and run with:

 ★ .x macro.C++; or .L macro.C+ then macro();

 ★ working in Python (with PyROOT) is not fully supported yet

★ include at beginning of macro:

```
using namespace RooFit;  
using namespace RooStats
```

★ avoid running a macro multiple times in the same ROOT session

★ If not working locally:

 ★ setting ROOT 5.25.04 using the CERN AFS installation:

 ★ on SLC4:

 ★ /afs/cern.ch/sw/lcg/app/releases/ROOT/5.25.04/sl4_amd64_gcc34/root/bin/thisroot.csh
 or .sh

Getting Help



- ❖ RooStats documentation:
 - ❖ Class reference doc (could be improved but still useful):
 - ❖ http://root.cern.ch/root/html/doc/ROOFIT_ROOSTATS_Index.html
 - ❖ Some tutorials available as example in
 - ❖ \$ROOTSYS/tutorials/roostats
 - ❖ Wiki page from last tutorials session
 - ❖ <https://twiki.cern.ch/twiki/bin/view/RooStats/TutorialsOctober2009>
 - ❖ be careful: some of the example code is not valid anymore due to some changes in latest RooStats release
 - ❖ Ask help to tutors, we are few people around here to help you
 - ❖ Can start working with a couple of macro which generate the data
 - ❖ download from agenda page or from
 - ❖ http://www.cern.ch/moneta/temp/exercises_1.tar.gz

Exercise 1: Model Generation



- ★ Generate first a Poisson Model (using RooFit Workspace and factory method)

- ★ build model of $n_{obs} = S + B$ with a $\text{Poisson}(S+B)$. $\langle S \rangle = 2$, $B = 1$ fixed

```
// Use a RooWorkspace to store the PDF models, parameters, etc ...
RooWorkspace myWS("myWS");

// Generate the model: POisson with S+B - signal is the parameter of interest
myWS.factory("S[2,0,20]"); // default value 2 and range [0,20]
myWS.factory("B[1]"); // B fixed at value 1
// define the observable: the number of observed events (min < 0 to include nobs =0)
myWS.factory("nobs[-0.001,20]"); // arbitrary range [0,20] but with 0 included

// model is a Poisson distribution for S+B
myWS.factory("sum:splusb(S,B)");
myWS.factory("Poisson::model(nobs,splusb)");
```

- ★ generate one event (i.e. one experiment) -> get nobs

```
// set generator seed (use 0 for getting different events each time, or a fixed number)
(RooRandom::randomGenerator())->SetSeed(0); // 0 to generate always different event

// generate 1 event to get the nobs value
RooAbsData* data = myWS.pdf("model")->generate(*myWS.var("nobs"),1,Name("data"));
double nobs = ((RooRealVar *) data->get(0)->first())->getVal();

std::cout << "number of observed events = " << nobs << std::endl;
```

- ★ if you prefer not to type can use macro **PoissonModel.C** for model generation

Exercise 1: Likelihood interval



- ★ Use Likelihood method to find interval limits and significance
 - ★ create Profile Likelihood calculator from data, model and parameter of interest (see doc http://root.cern.ch/root/htmldoc/RooStats_ProfileLikelihoodCalculator.html)

```
// first use some convenience variables to avoid too much typing
RooRealVar * S = myWS.var("S"); // the parameter of interest
RooArgSet poi(*S); // define the parameters of interest set
RooRealVar * B = myWS.var("B"); // the nuisance parameter
RooArgSet nuispar(*B); // define the nuisance parameter set
RooAbsPdf * model = myWS.pdf("model"); // define the model pdf

// create Profile Likelihood Calculator (with 68% CL, can be set also later)
ProfileLikelihoodCalculator plc(*data,*model,poi,0.68);
```

- ★ get the interval and its limit

```
// find the interval limits
LikelihoodInterval * interval = plc.GetInterval();
double lowerLimit = interval->LowerLimit(*S);
double upperLimit = interval->UpperLimit(*S);

std::cout << " PROFILE LIKELIHOOD results:\n";
std::cout << "68% CL interval: [ " << lowerLimit << " ; " << upperLimit << " ]\n";
```

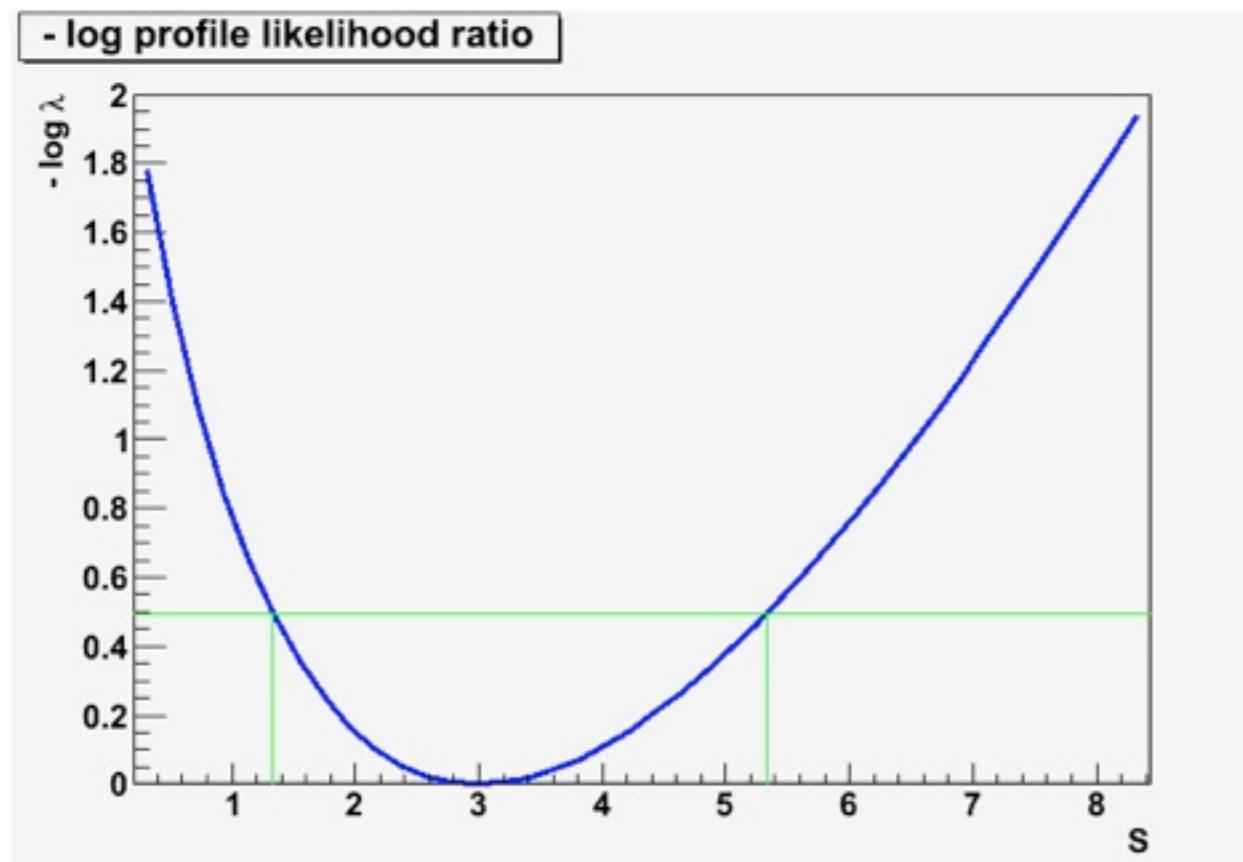
- ★ plot the result

```
// plot the result
LikelihoodIntervalPlot lplot(interval);
lplot.Draw();
```

Exercise 1: Likelihood result



- Example: plot obtained for nobs = 4 (with random seed = 999)



68% CL interval:
[1.32625 , 5.33143]

significance = 2.25633

- Test S=0 vs S not 0 hypothesis using the profile likelihood ratio:

```
// Create a copy of the POI parameters and set the S value to zero
RooArgSet nullparams;
nullparams.addClone(*myWS.var("S"));      // deep copy of parameter of interest
((RooRealVar *) (nullparams.first()))->setVal(0);    // set S = 0
plc.SetNullParameters(nullparams);        // set the new set in the calculator
//perform hypothesis test
HypoTestResult* plcResult = plc.GetHypoTest();
double significance = plcResult->Significance();
std::cout << "Likelihood significance for S = " << significance << std::endl;
```

Exercise 1: Systematics



- ❖ Add systematics in nuisance parameters
 - ❖ Gaussian Background B : 1 ± 0.5 (Gaussian model)
 - ❖ change code as below:

```
/// myWS.factory("B[1]"); // B fixed at value 1  
myWS.factory("B[1,0,20]"); // B variable
```

- ❖ specify the full model Poisson(S+B) × Gaussian(B)

```
// model is now a Poisson(S+B) x Gaussian(B)  
myWS.factory("Poisson::modelSB(nobs,splusb)");  
// Gaussian distribution for background (uncertainty of 30%)  
myWS.factory("Gaussian::bkgPdf(B,1,0.5)"); // B gaussian  $1 \pm 0.5$   
  
myWS.factory("PROD::model(modelSB,bkgPdf)");
```

- ❖ Run Profile Likelihood calculator as before
 - ❖ you will get larger interval and a smaller significance value due to the effect of systematics

68% CL interval: [1.24004 ; 5.37717]
significance = 1.95652

Exercise 2: Gaussian Model



- ❖ Generate Gaussian signal over flat background
- ❖ systematics in sigma of mass and background
- macro : GaussianModel.C (only model generation)

```

RooWorkspace myWS( "myWS" );

// Observable
myWS.factory("mass[0,500]"); // range [0,500]

// Signal and background distribution of the observable
myWS.factory("Gaussian::sigPdf(mass,200,sigSigma[0,100])");
myWS.factory("Uniform::bkgPdf(mass)");
myWS.factory("SUM::modelNoSyst(S[5,0,30]*sigPdf,B[10,0,100]*bkgPdf") ;
// Background only pdf
myWS.factory("ExtendPdf::modelBkg(bkgPdf,B)");
// Prior for signal
myWS.factory("Uniform::priorPOI(S)");
// Priors for nuisance parameters (signal + backg)
myWS.factory("Gaussian::prior_sigSigma(sigSigma,50,5)");
myWS.factory("Gaussian::prior_B(B,10,3)");
myWS.factory("PROD::priorNuisance(prior_sigSigma,prior_B)");
//full model including systematics
myWS.factory("PROD::model(modelNoSyst,priorNuisance)");

// generate data (unbinned with Poisson fluctuations on total N = S+B)
RooAbsData * data = myWS.pdf("model")->generate(*myWS.var("mass"),Extended(),Name("data"));

```

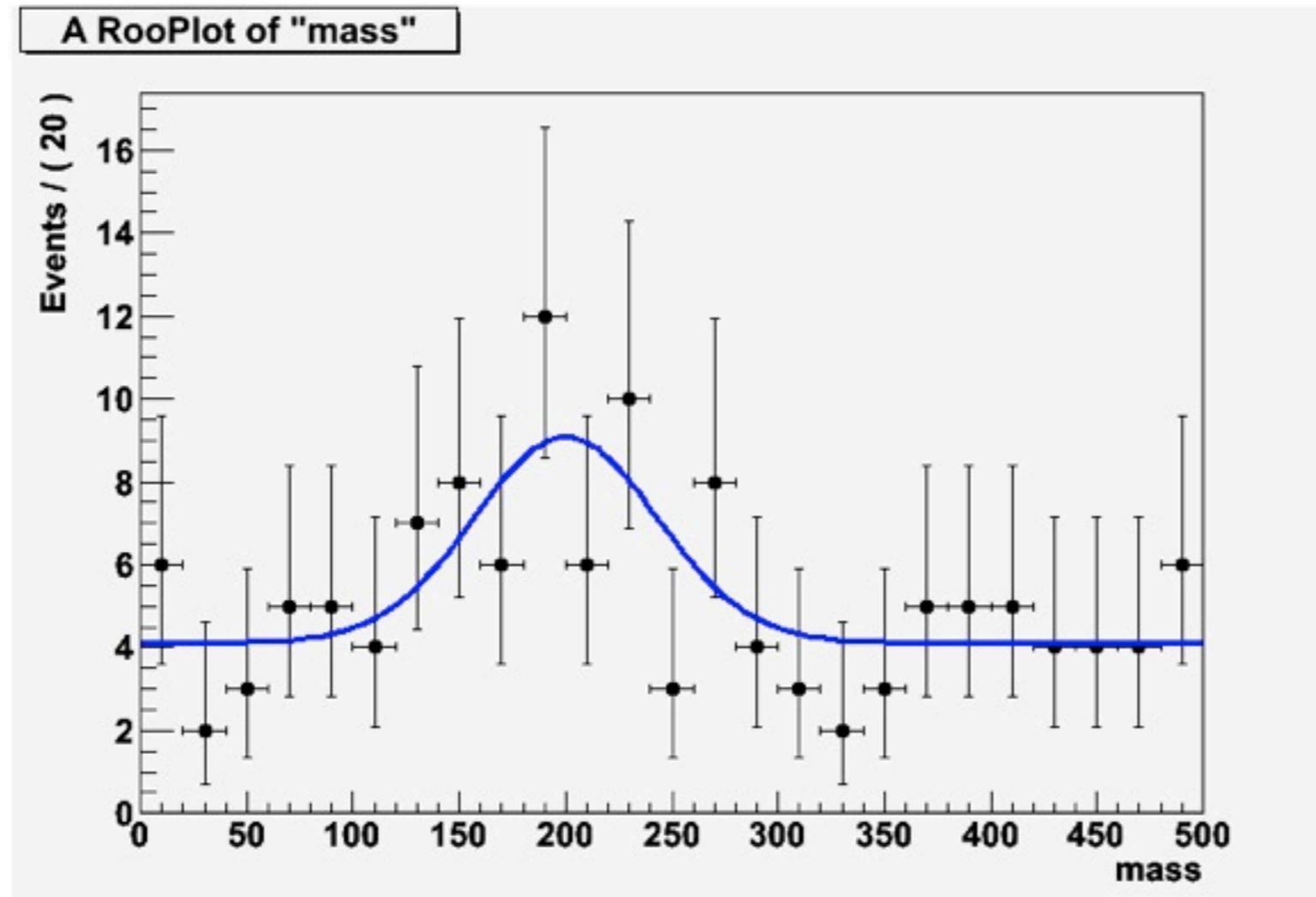
Exercise 2: Plot Data



★plot the data

```
//plot the generated data
RooPlot* plot = myWS.var("mass")->frame();
data->plotOn(plot);
```

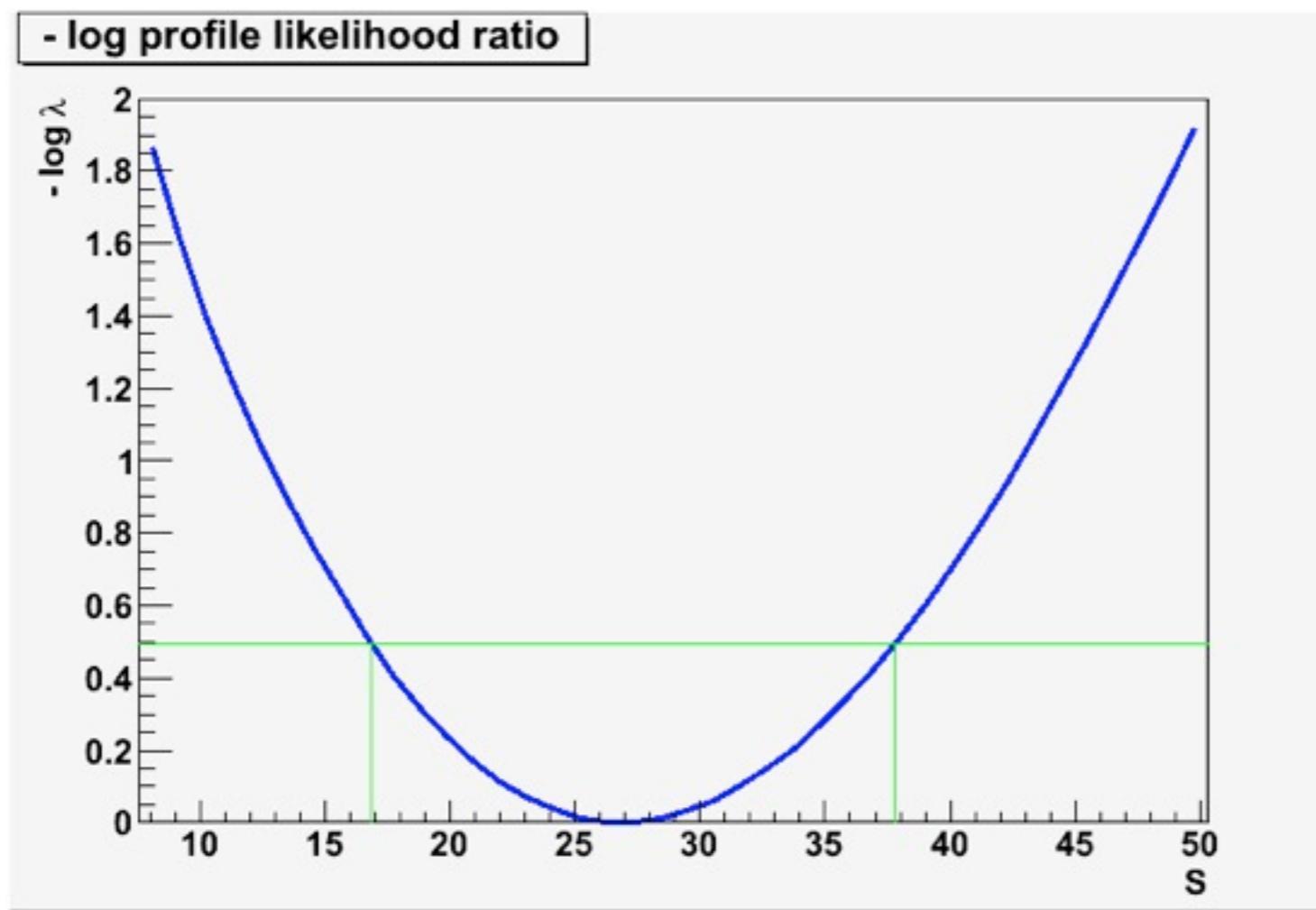
★result obtained (with seed = 1)



Exercise 2: Likelihood Result



- run then the Profile Likelihood calculator as before in the Poisson case (Exercise 1)



PROFILE LIKELIHOOD result: 68% CL interval: [16.8412 , 37.7899]

SIGNIFICANCE = 3.36367 sigma

Exercise 3: HybridCalculator



- ❖ Run HybridCalculator to perform an hypothesis test
 - ❖ test B only hypothesis vs S+B
 - ❖ set first value of all parameters S, B, sigSigma (e.g. initial or best fit values)
- ❖ For the HybridCalculator we need the background pdf, the pdf of the nuisance parameters (see “priorNuisance” in slide 8) and the nuisance parameters set
 - ❖ see http://root.cern.ch/root/html/doc/RooStats_HybridCalculator.html

```
RooAbsPdf * nuisPdf = myWS.pdf("priorNuisance");
RooAbsPdf * modelBkg = myWS.pdf("modelBkg");
HybridCalculator * hc=new HybridCalculator(*data,*model,*modelBkg, &nuisPar, nuisPdf);
hc->SetNumberOfToys(5000);
// default is 1 likelihood ratio, 0 is number of events, 2 profile likelihood ratio
hc->SetTestStatistic(1);
```

- ❖ do the hypothesis test running the toys

```
HybridResult* hcresult=hc->GetHypoTest();
double clsb_data = hcresult->CLsplusb();
double clb_data = hcresult->CLb();
double cls_data = hcresult->CLs();
double data_significance = hcresult->significance();
```

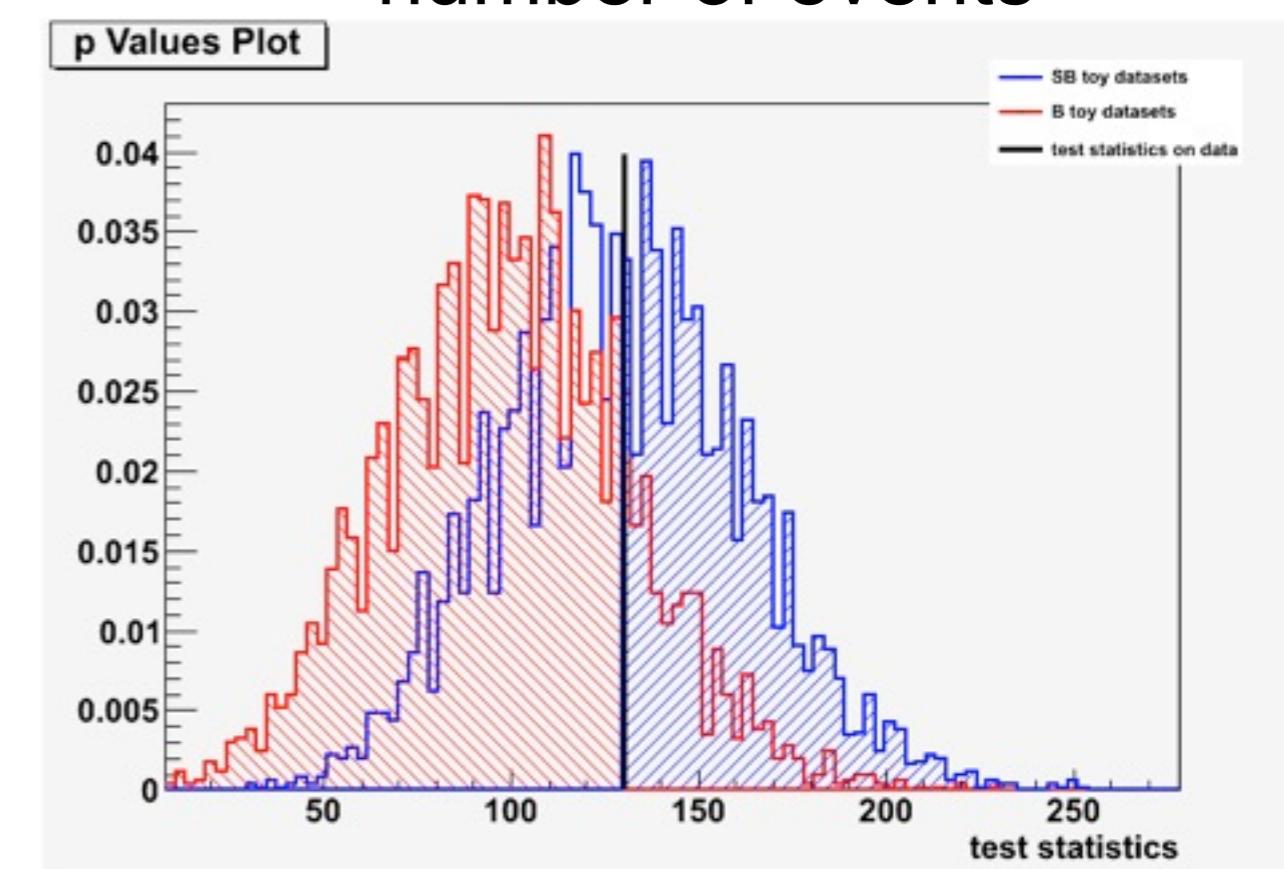
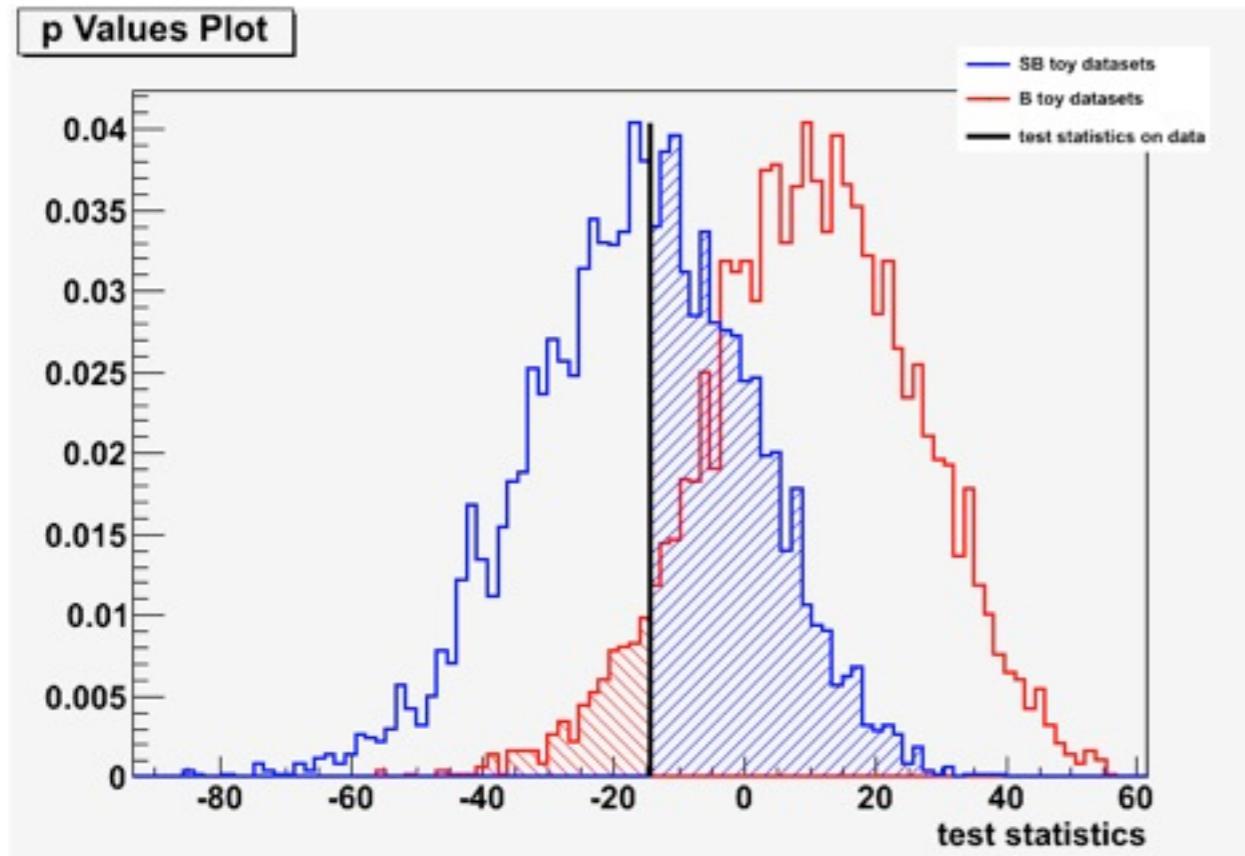
- ❖ plot the result (test statistic distributions

```
// plot the result: distribution of test statistic for null and alternate hypothesis
HybridPlot* hcPlot=hcresult->GetPlot("hcPlot","p Values Plot",100);
hcPlot->Draw();
```

Exercise 3: Hybrid Result



- ❖ Result obtained (remember to set initial values correctly)
 - ❖ 5000 toys, initial parameter values, test statistic:
likelihood ratio



HYBRID CALCULATOR result:

CL_b: 0.9322
 CL_s: 0.505471
 CL_sb: 0.4712
 significance: 1.49238

HYBRID CALCULATOR result:

CL_b: 0.836
 CL_s: 0.614593
 CL_sb: 0.5138
 significance: 0.97815

Exercise 4: BayesianCalculator



- ❖ Run BayesianCalculator for interval in parameter of interest
 - ❖ class works only for 1D, can use MCMCCalculator for multi-dimensional problems
- ❖ Build from data, model and prior for p.o.i.
 - ❖ see http://root.cern.ch/root/html/doc/RooStats_BayesianCalculator.html
- ❖ Need first to define a prior for p.o.i.
 - ❖ Example in case of Poisson model define uniform prior

```
myWS.factory("Uniform::priorPOI(s)");
BayesianCalculator bcalc(*data,*model,poi,*priorPOI,&nuisPar);
```

- ❖ normally would get interval by doing

```
bcalc.SetConfidenceLevel(0.32);
SimpleInterval * bInterval = bcalc.GetInterval();
lowerLimit = bInterval->LowerLimit(); upperLimit = bInterval->UpperLimit();
```

- ❖ but there is a bug in 5.25.04 (sorry!). Workaround, use posterior pdf and ROOT class TF1 (32% and 68% quantiles are the interval limits)

```
TF1 * f1 = posteriorPdf->asTF(poi);
double q[2]; double pval[2];
pval[0] = 0.32; pval[1] = 0.68;
f1->GetQuantiles(2,q,pval);
f1->DrawClone(); // for drawing the posterior pdf
std::cout << "68% CL interval: [ " << q[0] << " - " << q[1] << " ] \n";
```

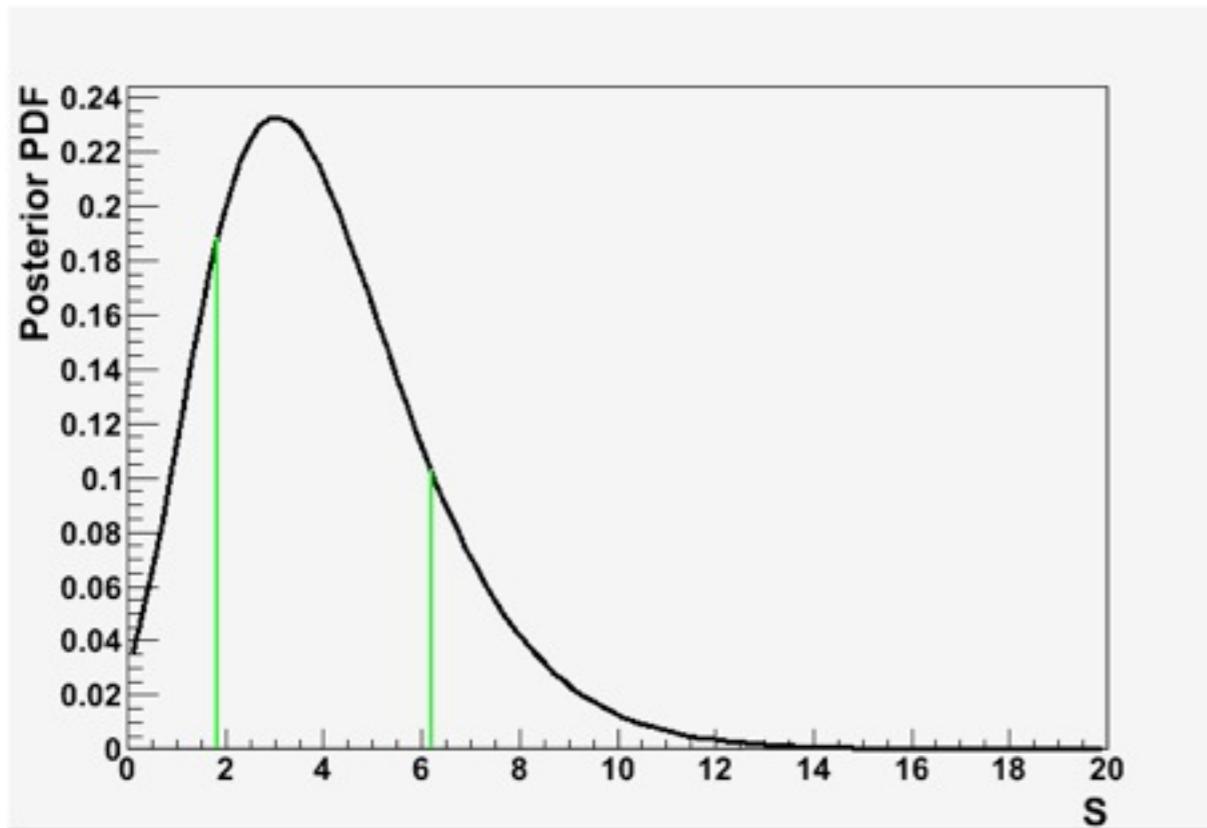
Exercise 4: Result



- ★ Can draw the posterior and its interval (when bug will be fixed)

```
RooPlot * bplot = bcalc.GetPosteriorPlot();
bplot->Draw();
```

- ★ Draw for the moment the TF1 with interval limit



68% CL interval: [1.82826 - 6.19458]

Exercise 4b: MCMCCalculator



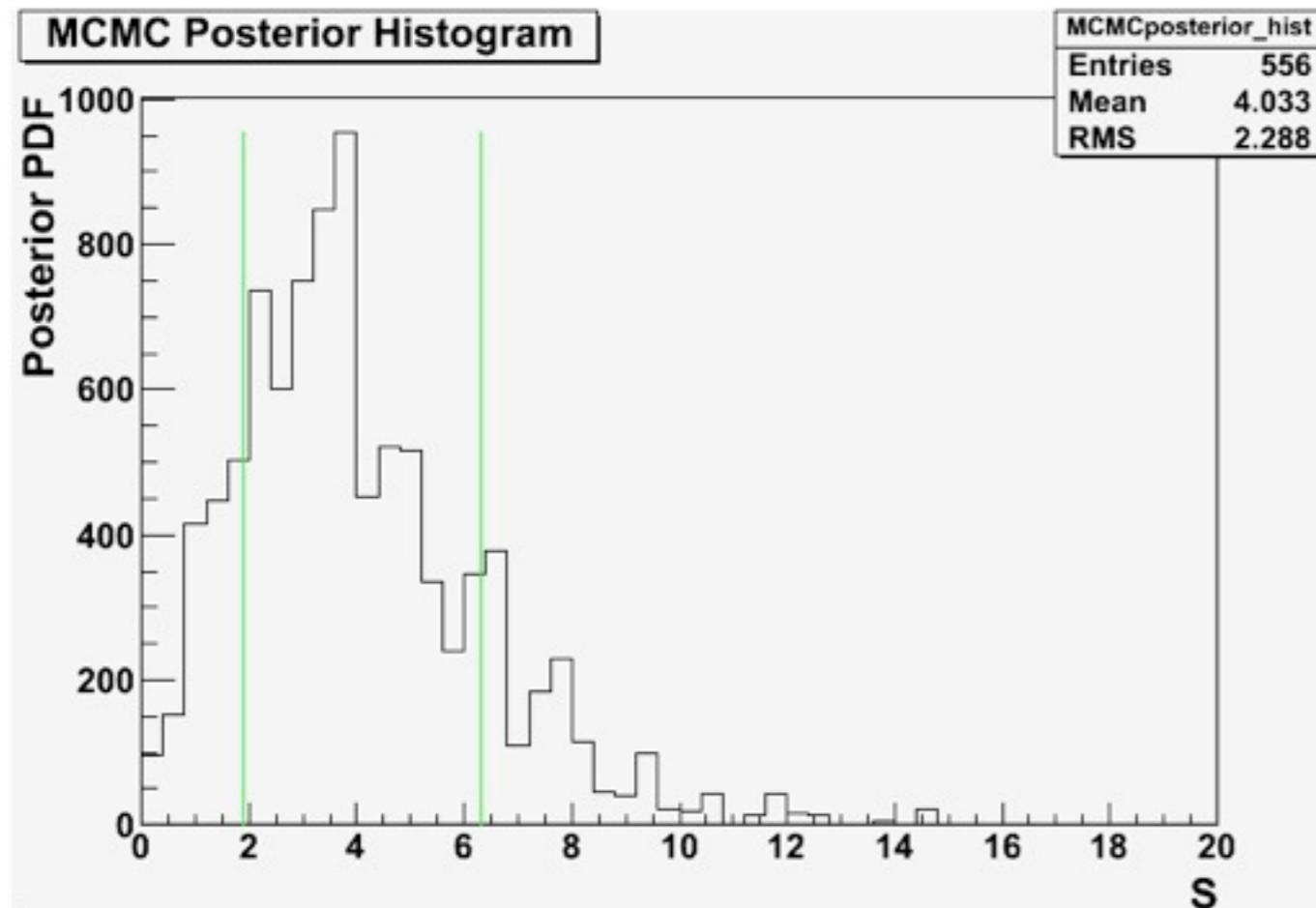
- Run same example as before but using the MCMC calculator

see http://root.cern.ch/root/html/RooStats_MCMCCalculator.html

```
MCMCCalculator mccalc(*data,*model,poi,*priorPOI);
mccalc.SetConfidenceLevel(0.32);
MCMCInterval * mcInterval = mcCalc.GetInterval();
```

- Also for a bug in LowerLimit/UpperLimit use posterior histogram as before

```
TH1* posterior = mcInterval->GetPosteriorHist();
posterior->GetQuantiles(2,q,p);
```



68% CL interval:
[1.90416 - 6.30061]

with default MC parameters

```
mccalc.SetNumIter(100000);
mccalc.SetNumBins(50);
```