RooStats
Lecture and Tutorials

Lorenzo Moneta (CERN)

Terascale Alliance School and Workshop, Bonn, 20-22 August 2012

Outline

- Introduction to RooStats
- Model building with RooFit
  - brief introduction to RooFit
    - slides from W. Verkerke, NIKHEF), but more material available at http://indico.in2p3.fr/getFile.py/access?contribId=15&resId=0&materialId=slides&confId=750
- RooStats Statistic Calculators
- Tutorials on model building and basic RooStats functionality
- Hypothesis tests in RooStats
- Hypothesis test inversion
  - Frequentist Limit calculators (CLs)
- Tutorials on CL_s limits and discovery significance

RooStats Project

- Collaborative project to provide and consolidate advanced statistical tools needed by LHC experiments
- Joint contribution from ATLAS, CMS ROOT and RooFit
  - developments over sighted by ATLAS and CMS statistics committees
  - initiated from previous code developed in ATLAS and CMS
  - current contributors: K. Cranmer, G. Lewis, S. Kreiss (ATLAS), G. Schott, G. Kukartsev (CMS), G. Bucur, L . Moneta (ROOT), W. Verkerke (RooFit & ATLAS) , A. Lazzaro (OpenLab)
  - and contributors also from: K. Belasco (ATLAS), A. De Cosa, M. Pelliccioni, D. Piparo, G. Petrucciani S. Schmitz, Wolf (CMS)

What is RooStats ?

- Common framework for statistical calculations
  - work on arbitrary models and datasets
    - factorize modeling from statistical calculations
  - implement most accepted techniques
    - frequentists, Bayesian and likelihood based tools
  - possible to easy compare different statistical methods
  - provide utility for combinations of results
  - using same tools across experiments facilitates the combinations of results

Statistical Applications

- Problems addressed by RooFit/RooStats:
  - point estimation: determine the best estimate of a parameter
  - estimation of confidence (credible) intervals
    - lower/upper limits
    - multi-dimensional contours or just a lower/upper limit
  - hypothesis tests:
    - evaluation of p-value for one or multiple hypotheses (discovery significance)
- Analysis combination:
  - performed at analysis level: full information available to treat correlations

RooStats Technology

- Built on top of RooFit
  - generic and convenient description of models (probability density function or likelihood functions)
  - provides *workspace* (RooWorkspace)
    - container for model and data and can be written to disk
    - inputs to all RooStats statistical tools
    - convenient for sharing models (e.g digital publishing of results)
  - easily generation of models (workspace factory and HistFactory tool)
  - tools for combinations of model (e.g. simultaneous pdf)
- Use of ROOT core libraries:
  - minimization (e.g. Minuit), numerical integration, etc...
  - additional tools provided when needed (e.g. Markov-Chain MC)

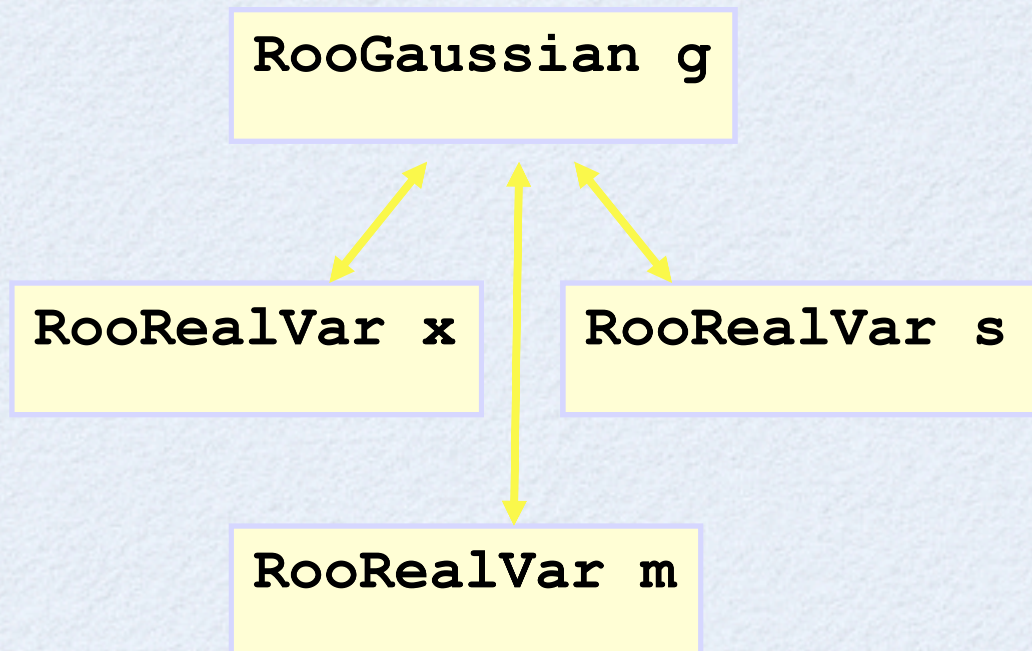
RooFit Modeling

Mathematical concepts are represented as C++ objects

Mathematical concept		RooFit class
variable	x	`RooRealVar`
function	$f(x)$	`RooAbsReal`
PDF	$f(x)$	`RooAbsPdf`
space point	$\vec{x}$	`RooArgSet`
integral	$\int_{x_{\text{min}}}^{x_{\text{max}}} f(x)dx$	`RooRealIntegral`
list of space points		`RooAbsData`

RooFit Modeling

Example: Gaussian pdf *Gaus(x,m,s)*



RooFit code:

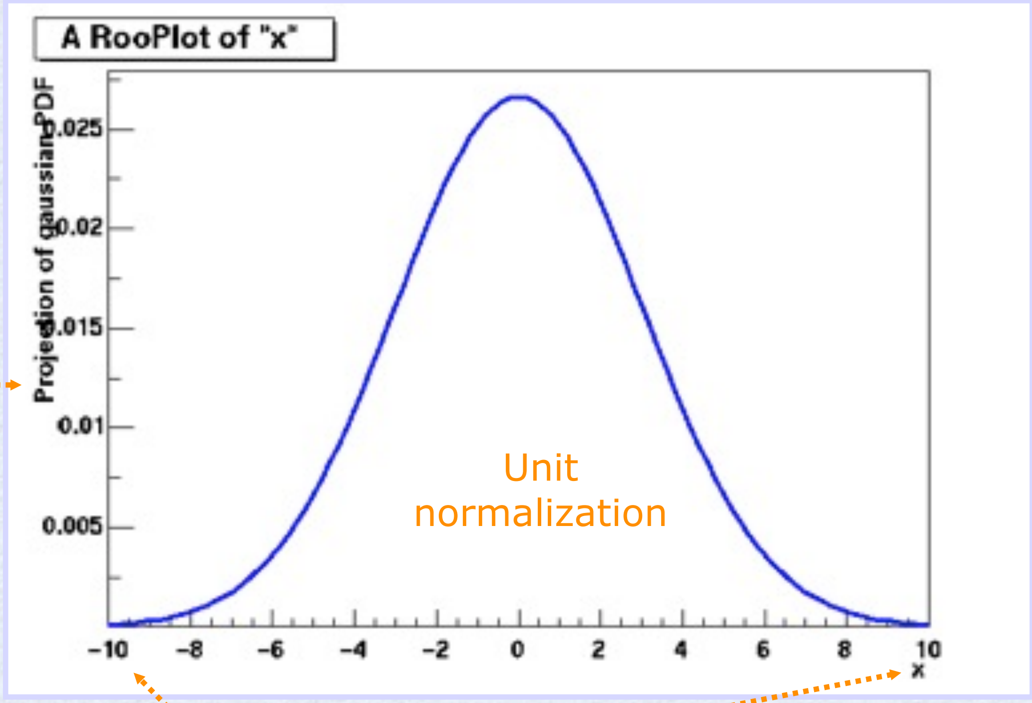
```
RooRealVar x("x","x",2,-10,10)
RooRealVar s("s","s",3) ;
RooRealVar m("m","m",0) ;
RooGaussian g("g","g",x,m,s)
```

- Represent relations between variables and functions as client/server links between objects

RooFit Functionality

- pdf visualization

```
RooAbsPdf * pdf = w.pdf("g");
RooPlot * xframe = x->frame();
pdf->plotOn(xframe);
xframe->Draw();
```



Axis label from **gauss** title

A **RooPlot** is an empty frame capable of holding anything plotted versus it variable

Plot range taken from limits of **x**

RooFit Functionality

- Toy MC generation from any pdf

Generate 10000 events from Gaussian p.d.f and show distribution

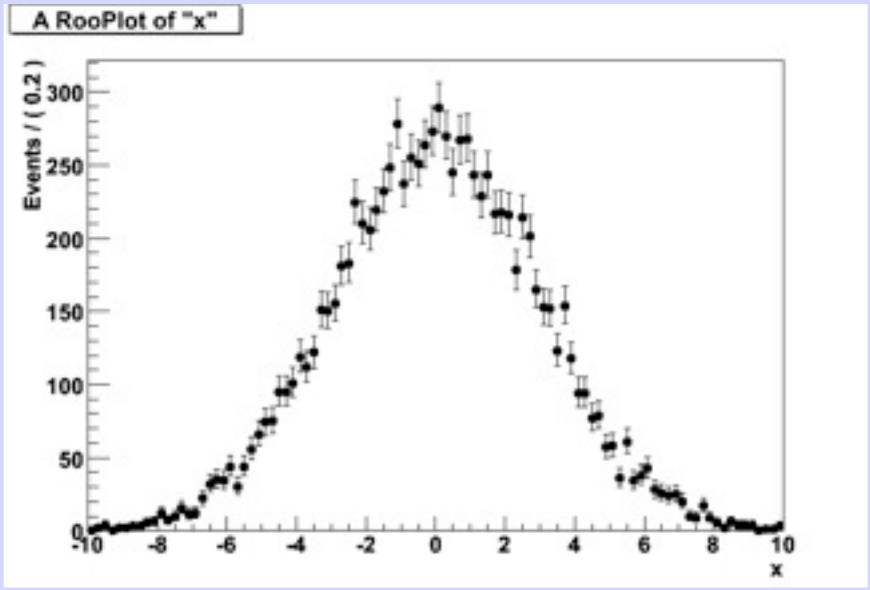
```
RooAbsPdf * pdf = w.pdf(“g”);
RooRealVar * x = w.var(“x”);
RooDataSet * data = pdf->generate(*x,10000);
```

- data visualization

```
RooPlot * xframe = x->frame();
data->plotOn(xframe);
xframe->Draw();
```

Note that dataset is ***unbinned***
(vector of data points, x, values)

Binning into histogram is performed in `data->plotOn()` call



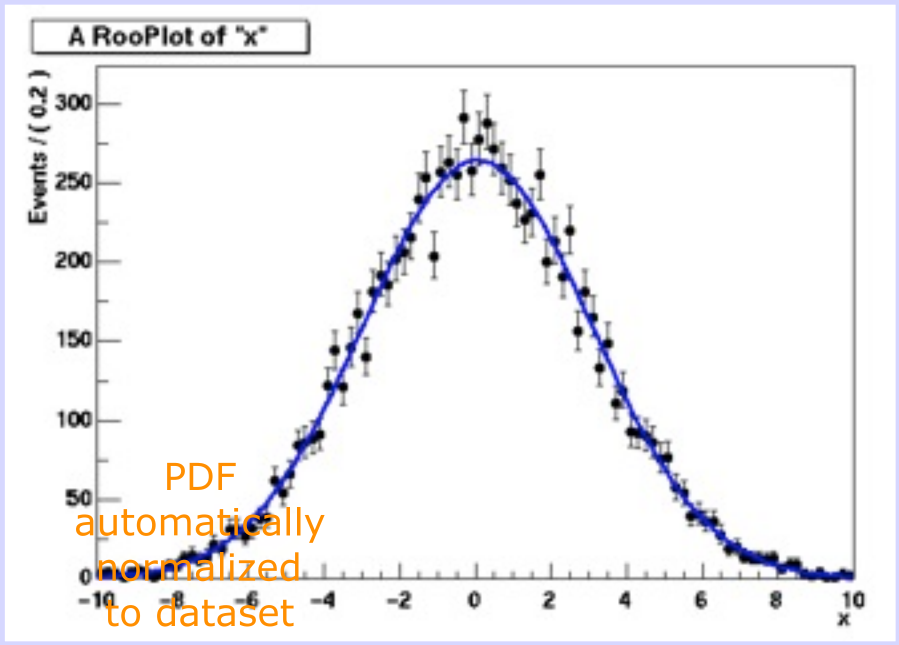
RooFit Functionality

- Fit of model to data
  - e.g. unbinned maximum likelihood fit

```
pdf = pdf->fitTo(data);
//parameters will have now fitted values
w->var(“m”)->Print();
w->var(“s”)->Print();
```

- data and pdf visualization after fit

```
RooAbsPdf * pdf = w.pdf(“g”);
RooPlot * xframe = x->frame();
data->plotOn(xframe);
pdf->plotOn(xframe);
xframe->Draw();
```



RooFit Workspace

- `RooWorkspace` class: container for all objected created:
  - full model configuration
    - PDF and parameter/observables descriptions
    - uncertainty/shape of nuisance parameters
  - (multiple) data sets
- Maintain a complete description of all the model
  - possibility to save entire model in a ROOT file
- Combination of results joining workspaces in a single one
- All information is available for further analysis
  - common format for combining and sharing physics results

```
RooWorkspace workspace(“Example_workspace”);
workspace.import(*data);
workspace.import(*pdf);
workspace.defineSet(“obs”,”x”);
workspace.defineSet(“poi”,”mu”);
workspace.importClassCode();
workspace.writeToFile(“myWorkspace”)
```

RooFit Factory

```
RooRealVar x("x","x",2,-10,10)
RooRealVar s("s","s",3) ;
RooRealVar m("m","m",0) ;
RooGaussian g("g","g",x,m,s)
```

The workspace provides a factory method to auto-generates objects from a math-like language
(the p.d.f is made with 1 line of code instead of 4)

```
RooWorkspace w;
w.factory("Gaussian::g(x[2,-10,10],m[0],s[3])")
```

In the tutorial we will work using the workspace factory to build models

Using the workspace

- Workspace
  - A generic container class for all RooFit objects of your project
  - Helps to organize analysis projects
- Creating a workspace

```
RooWorkspace w("w") ;
```

- Putting variables and function into a workspace
  - When importing a function or pdf, all its components (variables) are automatically imported too

```
RooRealVar x("x","x",-10,10) ;
RooRealVar mean("mean","mean",5) ;
RooRealVar sigma("sigma","sigma",3)  ;
RooGaussian f("f","f",x,mean,sigma) ;

// imports f,x,mean and sigma
w.import(f) ;
```

Using the workspace

- Looking into a workspace

```
w.Print() ;

variables
---------
(mean,sigma,x)

p.d.f.s
-------
RooGaussian::f[ x=x mean=mean sigma=sigma ] = 0.249352
```

- Getting variables and functions out of a workspace

```
// Variety of accessors available
RooRealVar * x = w.var("x");
RooAbsPdf * f = w.pdf("f");
```

- Writing workspace and contents to file

```
w.writeToFile("wspace.root") ;
```

Factory syntax

- Rule #1 – Create a variable

```
x[-10,10]    // Create variable with given range
x[5,-10,10]  // Create variable with initial value and range
x[5]         // Create initially constant variable
```

- Rule #2 – Create a function or pdf object

```
ClassName::Objectname(arg1,[arg2],...)
```

  - Leading ‘Roo’ in class name can be omitted
  - Arguments are names of objects that already exist in the workspace
  - Named objects must be of correct type, if not factory issues error
  - Set and List arguments can be constructed with brackets {}

```
Gaussian::g(x,mean,sigma)
         → RooGaussian(“g”,”g”,x,mean,sigma)

Polynomial::p(x,{a0,a1})
              → RooPolynomial(“p”,”p”,x”,RooArgList(a0,a1));
```

Factory syntax

- Rule #3 – Each creation expression returns the name of the object created
  - Allows to create input arguments to functions ‘in place’ rather than in advance

```
Gaussian::g(x[-10,10],mean[-10,10],sigma[3])
    →  x[-10,10]
       mean[-10,10]
       sigma[3]
       Gaussian::g(x,mean,sigma)
```

- Miscellaneous points
  - You can always use numeric literals where values or functions are expected

```
Gaussian::g(x[-10,10],0,3)
```

  - It is not required to give component objects a name, e.g.

```
SUM::model(0.5*Gaussian(x[-10,10],0,3),Uniform(x))  ;
```

Factory syntax – using expressions

- Customized p.d.f from interpreted expressions

```
w.factory("EXPR::mypdf('sqrt(a*x)+b',x,a,b)") ;
```

- Customized class, compiled and linked on the fly

```
w.factory("CEXPR::mypdf('sqrt(a*x)+b',x,a,b)") ;
```

- re-parametrization of variables (making functions)

```
w.factory("expr::w('(1-D)/2',D[0,1])") ;
```

  - note using expr (builds a function, a RooAbsReal)
  - instead of EXPR (builds a pdf, a RooAbsPdf)

This usage of upper vs lower case applies also for other factory commands (SUM, PROD,....)

Factory syntax: p.d.f. composition

- Additions of PDF (using fractions)

```
SUM::name(frac1*PDF1,PDFN)
```

$$S(x) = fF(x) + (1-f)G(x)$$

```
SUM::name(frac1*PDF1,frac2*PDF2,...,PDFN)
```

  - Note that last PDF does not have an associated fraction

- PDF additions (using expected events instead of fractions)

```
SUM::name(Nsig*SigPDF,Nbkg*BkgPDF)
```

  - the resulting model will be extended
    - the likelihood will contain a Poisson term depending on the total number of expected events (Nsig+Nbkg)

- Uncorrelated product of PDF

```
w.factory("Gaussian::gx(x[-5,5],mx[2],sx[1])") ;
w.factory("Gaussian::gy(y[-5,5],my[-2],sy[3])") ;

w.factory("PROD::gxy(gx,gy)") ;
```

Constructing joint pdfs

- Operator class SIMUL to construct joint models at the pdf level
  - need a discrete observable (category) to label the channels

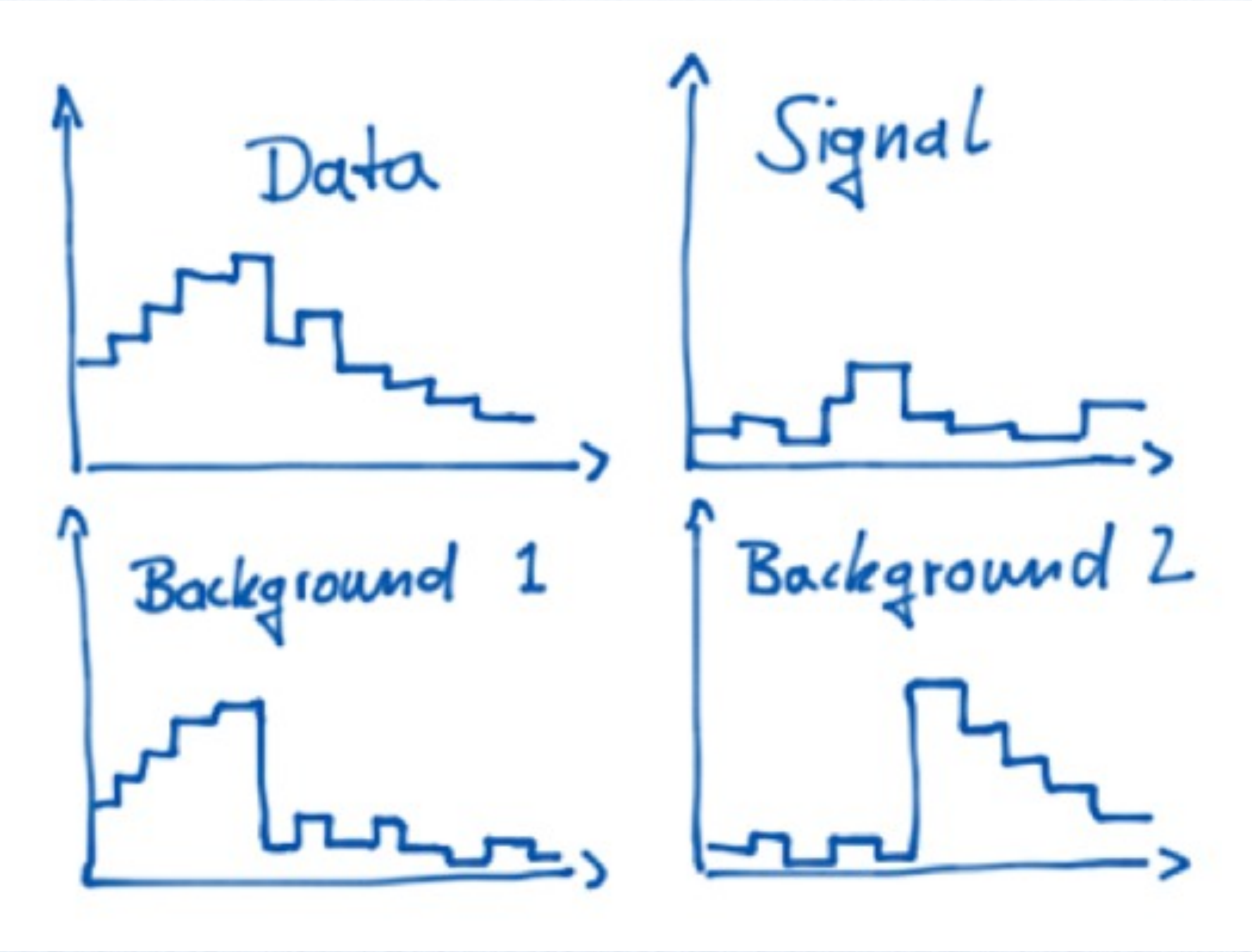
```
// Pdfs for channels 'A' and 'B'
w.factory("Gaussian::pdfA(x[-10,10],mean[-10,10],sigma[3])") ;
w.factory("Uniform::pdfB(x)") ;

// Create discrete observable to label channels
w.factory("index[A,B]") ;

// Create joint pdf (RooSimultaneous)
w.factory("SIMUL::joint(index,A=pdfA,B=pdfB)") ;
```

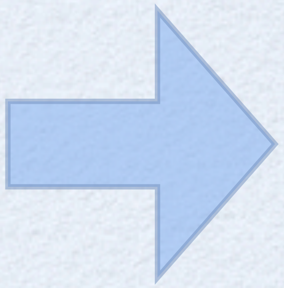
- Can also construct joint datasets
  - contains observables ("x") and category ("index")

```
RooDataSet *dataA, *dataB ;
RooDataSet dataAB("dataAB","dataAB",
                  RooArgSet(*w.var("x"),*w.cat("index")),
                  Index(*w.cat("index"),
                  Import("A",*dataA),Import("B",*dataB)) ;
```

Model Building with HistFactory

- Tool to build models from input histograms





RooFit Workspace

HistFactory

- Tool available in ROOT (in roofit/histfactory)
- Generalization of number counting models

$$\mathcal{P}(n_b|\mu) = \text{Pois}(n_{\text{tot}}|\mu S + B) \left[\prod_{b\in\text{bins}} \frac{\mu\nu_b^{\text{sig}} + \nu_b^{\text{bkg}}}{\mu S + B} \right]$$

where n_b is the data histogram

In general HistFactory produces model of this form

$$\mathcal{P}(n_{cb}, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_{c\in\text{channels}} \prod_{b\in\text{bins}} \text{Pois}(n_{cb}|\nu_{cb}) \cdot G(L_0|\lambda, \Delta_L) \cdot \prod_{p\in\mathbb{S}+\mathbf{\Gamma}} P_p(a_p|\alpha_p)$$

luminosity constraint

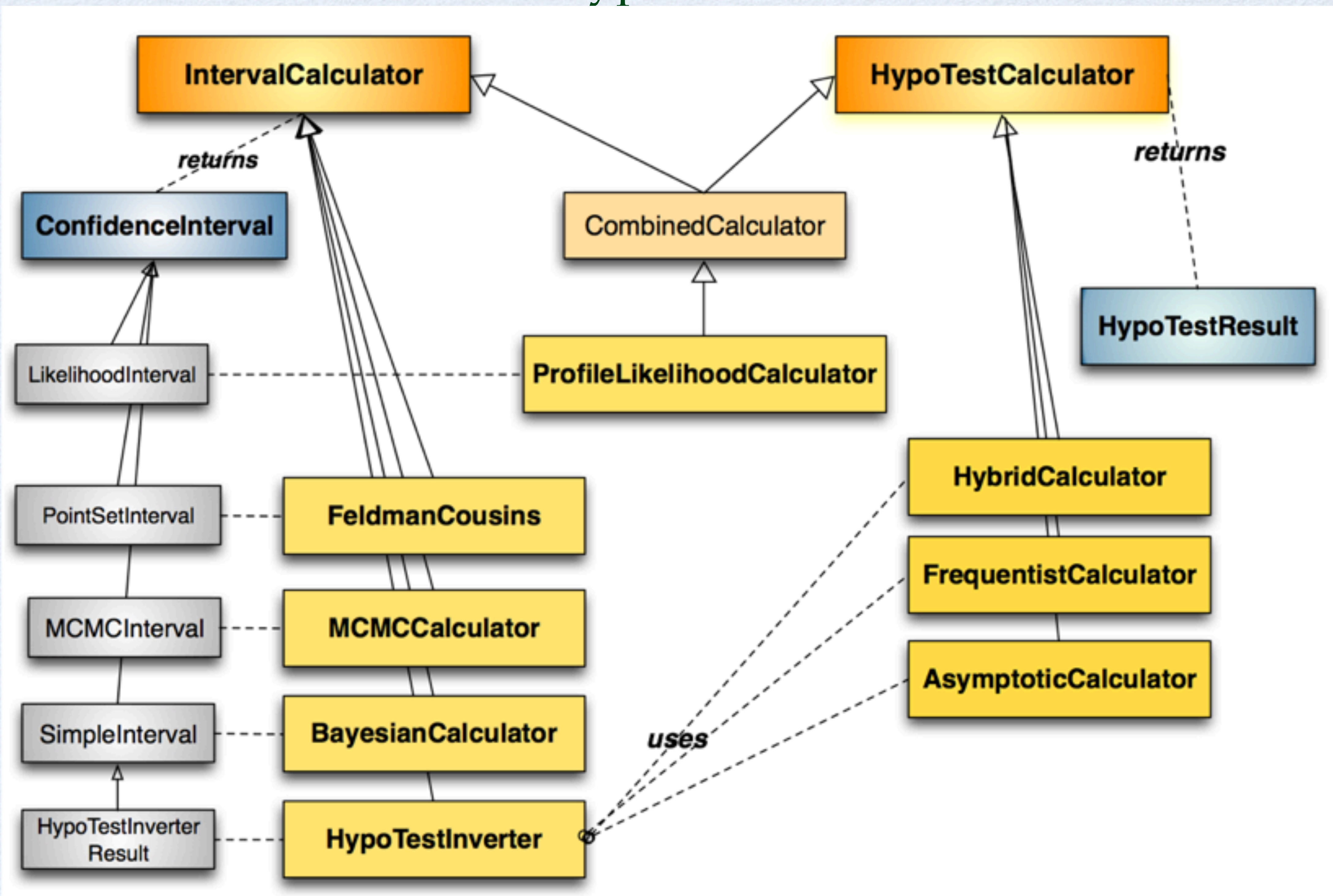
parameter constraint

HistFactory can be configured with XML files or directly in C++/Python (New in 5.34)

see also HistFactory User Guide (https://twiki.cern.ch/twiki/pub/RooStats/WebHome/HistFactoryLikelihood.pdf)

RooStats Design

- C++ interfaces and classes mapping to real statistical concepts
  - interval estimation or hypothesis tests



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)
};
```

ConfInterval 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

   };
}
```

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 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
};
```

HypoTestResult

```
class HypoTestResult : public TNamed {

   public:
      ........

      // 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 !fBackgroundIsAlt ? NullPValue() : AlternatePValue(); }

      // Convert  AlternatePValue into a "confidence level"
      virtual Double_t CLsplusb() const { return !fBackgroundIsAlt ? AlternatePValue() : NullPValue(); }

      // 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() ); }

      // return sampling distribution of test statistic for the null
      SamplingDistribution* GetNullDistribution(void) const  { return fNullDistr; }

      // return sampling distribution of test statistic for the alternate
      SamplingDistribution* GetAltDistribution(void) const  { return fAltDistr; }


      // return data value of test statistic
      Double_t GetTestStatisticData(void) const { return fTestStatisticData; }

      .....

private:
      ......
};
```

Main RooStats Calculator classes

- ProfileLikelihood calculator
  - interval estimation using asymptotic properties of the likelihood function
- Bayesian calculators
  - interval estimation using Bayes theorem

  `BayesianCalculator` (analytical or adaptive numerical integration)

  `MCMCCalculator` (Markov-Chain Monte Carlo)
- HybridCalculator, FrequentistCalculator
  - frequentist hypothesis test calculators using toy data (difference in treatment of nuisance parameters)
- AsymptoticCalculator
  - hypothesis tests using asymptotic properties of likelihood function
- HypoTestInverter
  - invert hypothesis test results (from Asympototic, Hybrid or FrequentistCalculator) to estimate an interval
  - main tools used for limits at LHC (limits using CLs procedure)
- NeymanConstruction and FeldmanCousins
  - frequentist interval calculators

ModelConfig Class

- `ModelConfig` class input to all Roostats calculators
  - contains a reference to the RooFit workspace class
  - provides the workspace meta information needed to run RooStats calculators
    - pdf of the model stored in the workspace
    - what are observables (needed for toy generations)
    - what are the parameters of interest and the nuisance parameters
    - global observables (from auxiliary measurements) for frequentist calculators
    - prior pdf for the Bayesian tools
  - ModelConfig can be imported in workspace for storage and later retrieval

Building ModelConfig Class

- ModelConfig must be built after having the workspace
- Set names for all the components which are present in the workspace

```
//specify components of model for statistical tools
ModelConfig  modelConfig("G(x|mu,1)");
modelConfig.SetWorkspace(workspace);
//set components using the name of ws objects
modelConfig.SetPdf( "normal");
modelConfig.SetParameterOfInterest("poi");
modelConfig.SetObservables("obs");
```

- Alternatively ModelConfig can be used to import the components directly into the workspace

```
// set and to import into workspace
modelConfig.SetPdf( *pdf);
```

- Some tools (Bayesian) require to specify prior pdf

```
//Bayesian tools would also need a prior
modelConfig.SetPriorPdf( "prior");
```

- ModelConfig can be imported in workspace to be then stored in a file

```
//can import modelConfig into workspace too
workspace.import(*modelConfig);
```

Profile Likelihood Calculator

- Method based on properties of the likelihood function
- Profile likelihood function:

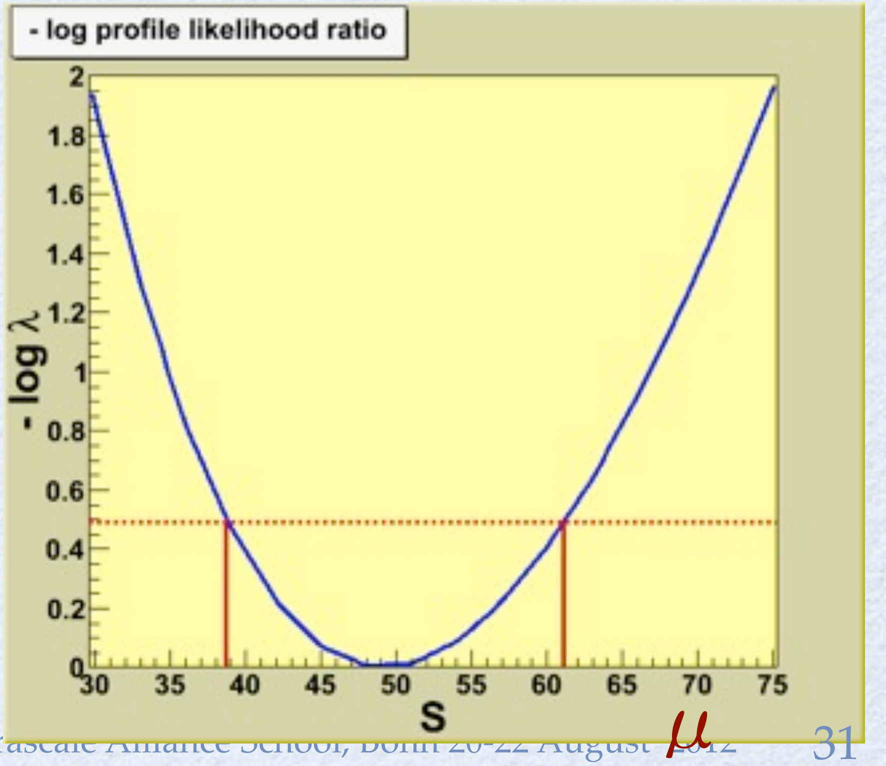
$$\lambda(\mu) = \frac{L(x|\mu, \hat{\hat{\nu}})}{L(x|\hat{\mu}, \hat{\nu})}$$

→ maximize w.r.t nuisance parameters ν and fix POI μ

→ maximize w.r.t. all parameters

λ is a function of only the parameter of interest μ

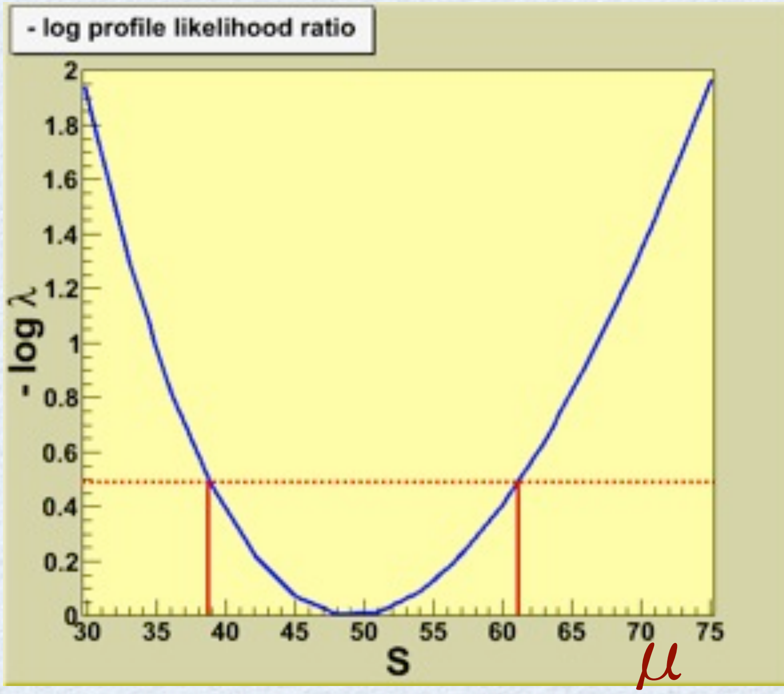
- Uses asymptotic properties of λ based on Wilks' theorem:
- from a Taylor expansion of logλ around the minimum:
  - ➔ -2logλ is a parabola (λ is a gaussian function)
  - ➔ interval on μ from logλ values



- Method of MINUIT/MINOS
  - lower/upper limits for 1D
  - contours for 2 parameters

Usage of Profile Likelihood Calculator

```
// create the class using data and model
ProfileLikelihoodCalculator plc(*data, *model);

// set the confidence level
plc.SetConfidenceLevel(0.683);

// compute the interval
LikelihoodInterval* interval = plc.GetInterval();
double lowerLimit = interval->LowerLimit(*mu);
double upperLimit = interval->UpperLimit(*mu);

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



- For one-dimensional intervals:
  - 68% CL (1 σ) interval : $\Delta\log\boldsymbol{\lambda} = \mathbf{0.5}$
  - 95% CL interval : $\Delta\log\boldsymbol{\lambda} = 1.96$
- **`LikelihoodIntervalPlot`** can plot the 2D contours

Hypothesis Test with Profile Likelihood

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

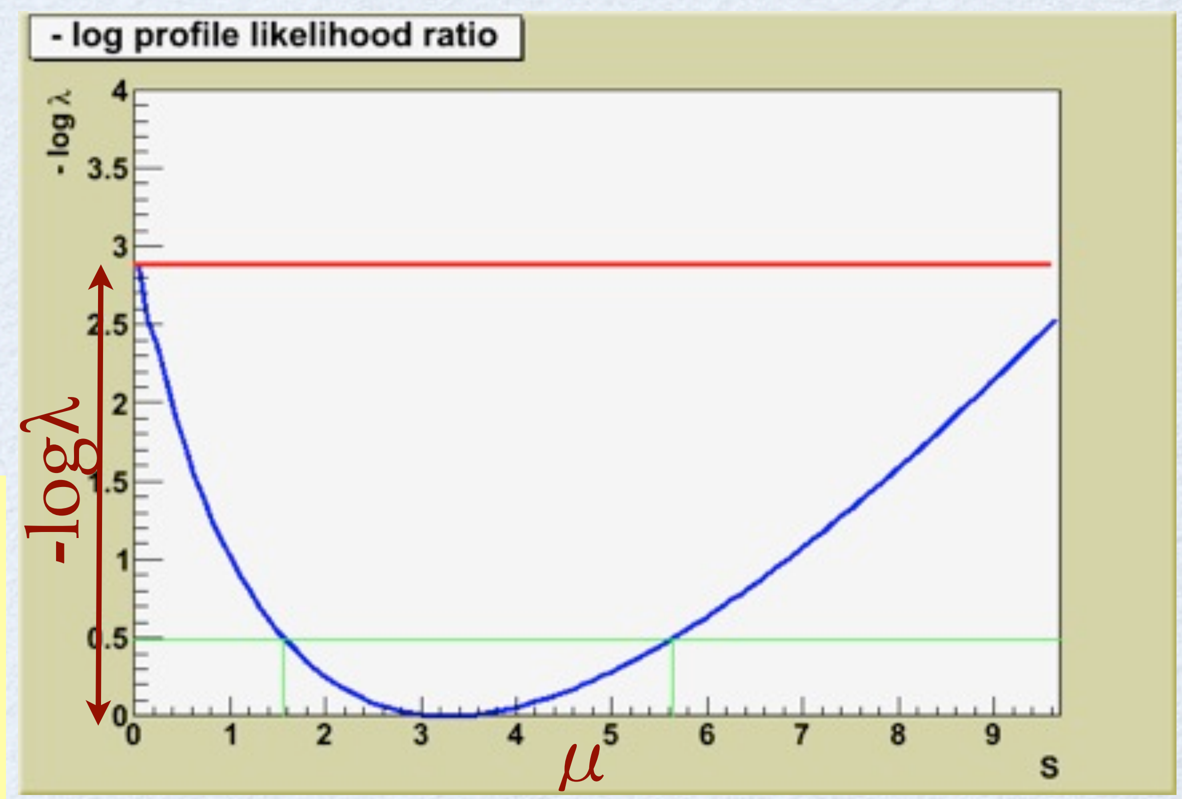
$$\lambda(\mu) = \frac{L(x|\mu,\hat{\hat{\nu}})}{L(x|\hat{\mu},\hat{\nu})}$$

- Null hypothesis (H_0): $\mu = \mu_0$
- Alternate hypothesis (H_1): $\mu \neq \mu_0$

- Distribution of -2logλ is asymptotically a χ^2 distribution under H_0
- p-value and significance can then be obtained from the -2logλ ratio

significance: $n_\sigma = \sqrt{-2\log\lambda}$

```
// set value of POI  to zero
// one can also use model.SetSnapshot(*mu)
S->setVal(0);
plc.SetNullParameters(*mu);
HypoTestResult* hypotest =plc.GetHypoTest();
double alpha = hypotest->NullPValue();
double significance = hypotest->Significance()
```



Bayesian Analysis in RooStats

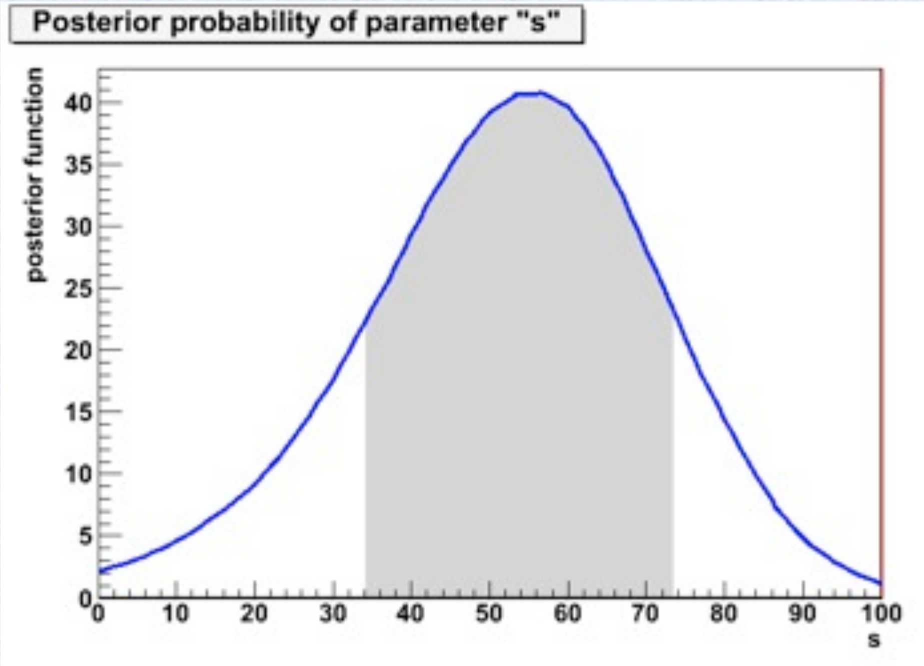
- `RooStats` provides classes for
  - marginalize posterior and estimate credible interval

posterior probability — likelihood function — prior probability — nuisance parameters marginalization

$$\underbrace{P(\mu|x)}_{\text{POI} \quad \text{data}} = \frac{\int L(x|\mu,\nu)\Pi(\mu,\nu)d\nu}{\underbrace{\iint L(x|\mu,\nu)\Pi(\mu,\nu)d\mu d\nu}_{\text{normalisation term}}}$$

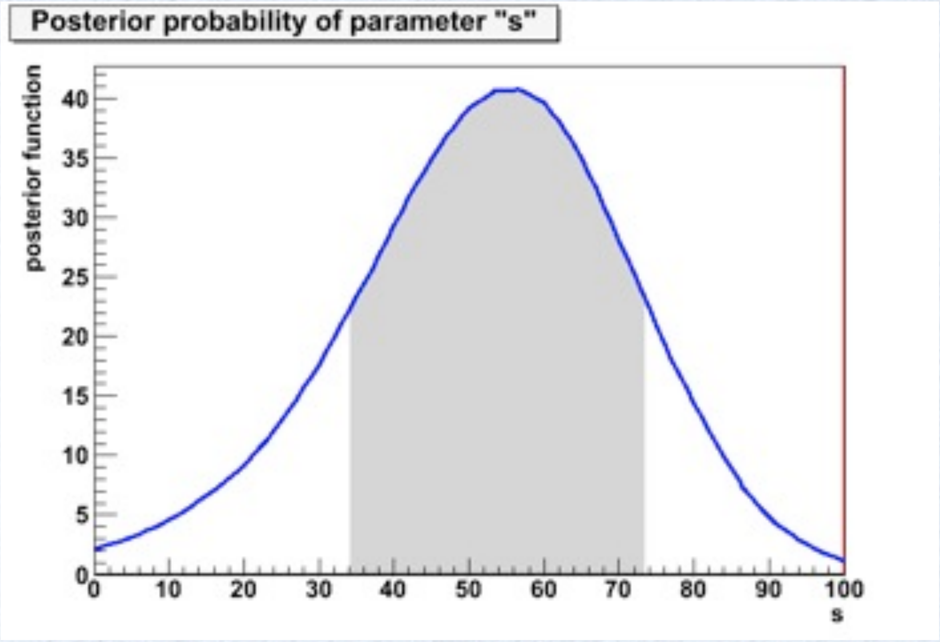
Bayesian Theorem

  - support for different integration algorithms:
    - adaptive (numerical)
    - MC integration
    - Markov-Chain
  - can work with models with many parameters (e.g few hundreds)



Bayesian Classes

- **`BayesianCalculator`** class
  - posterior and interval estimation using numerical integration
  - working only for one parameter of interest but can integrate many nuisance parameters
  - support for different integration algorithms, using **`BayesianCalculator::SetIntegrationType`**
    - adaptive numerical (default type), working only for few nuisances (< 10)
    - Monte Carlo integration (PLAIN, MISER, VEGAS)
    - TOYMC : sampling toys from nuisance pdf's (requires not-uniform nuisance pdf but can work with many parameters)
  - can compute central interval or one-sided interval (upper limit) or a shortest interval (SetCentralInterval)
  - provide plot of posterior and interval



Example: 68% CL central interval

```
BayesianCalculator bc(data, model);
bc.SetConfidenceLevel(0.683);
bc.SetLeftSideTailFraction(0.5);
bc.SetIntegrationType("ADAPTIVE");
SimpleInterval* interval = bc.GetInterval();
double lowerLimit = interval->LowerLimit();
double upperLimit = interval->UpperLimit();
RooPlot * plot = bc.GetPosteriorPlot();
plot->Draw();
```

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

$$\rho = \frac{L(\vec{\alpha}') \cdot P(\vec{\alpha}')}{L(\vec{\alpha}) \cdot P(\vec{\alpha})} \cdot \frac{Q(\vec{\alpha}|\vec{\alpha}')}{Q(\vec{\alpha}'|\vec{\alpha})}$$

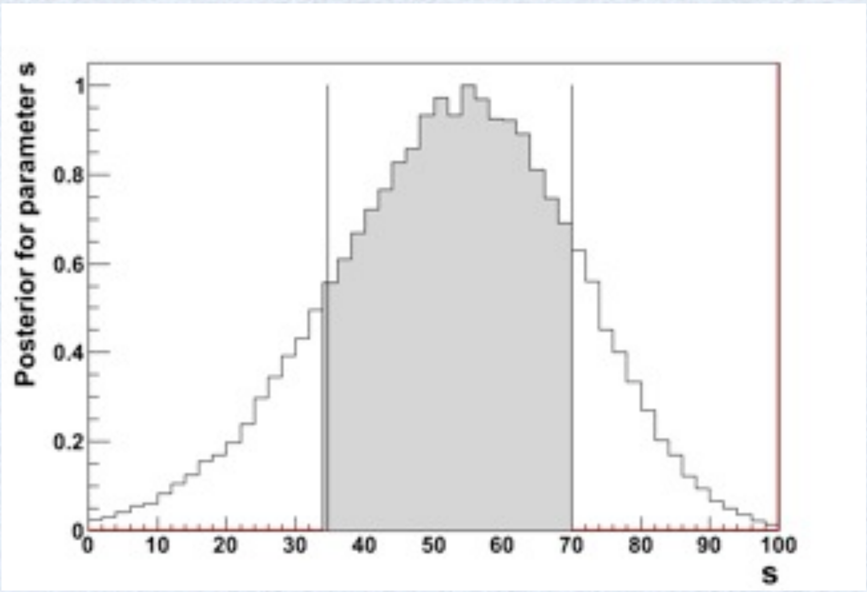
  - (appending original point $\vec{\alpha}$ with complementary probability)
- RooStats works with any $L(\vec{\alpha})$, $P(\vec{\alpha})$
- ~~Since last week~~: can use any RooFit PDF as proposal function $Q(\vec{\alpha}'|\vec{\alpha})$

Work done primarily by Kevin Belasco, a Princeton undergraduate I'm working with.

MCMC Calculator

- **`MCMCCalculator`** class
  - integration using Markov-Chain Monte Carlo (Metropolis Hastings algorithm)
  - can deal with more than one parameter of interest
  - can work with many nuisance parameters
    - e.g. used in Higgs combination with more than 300 nuisances
  - possible to specify ProposalFunction
    - multivariate Gaussian from fit result
    - Sequential proposal
  - can visualize posterior and also the chain result

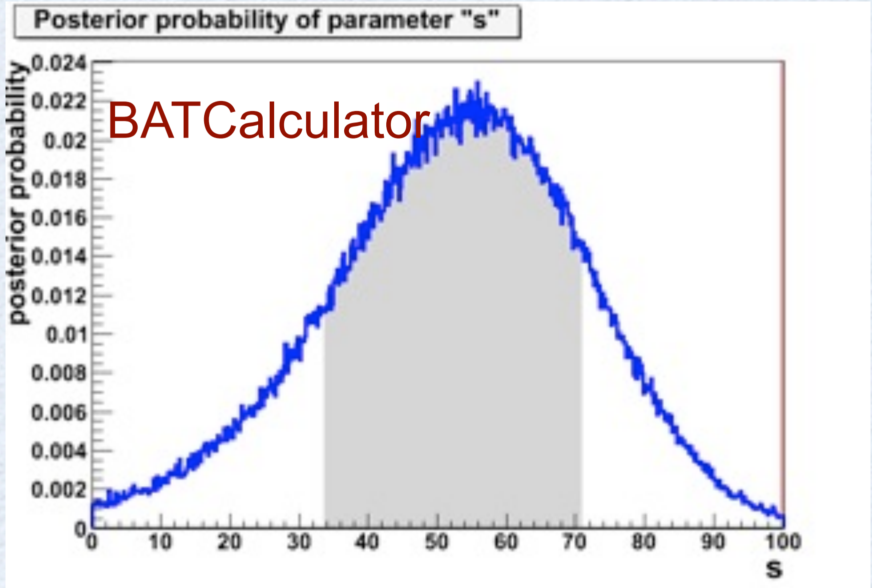
MCMCCalculator



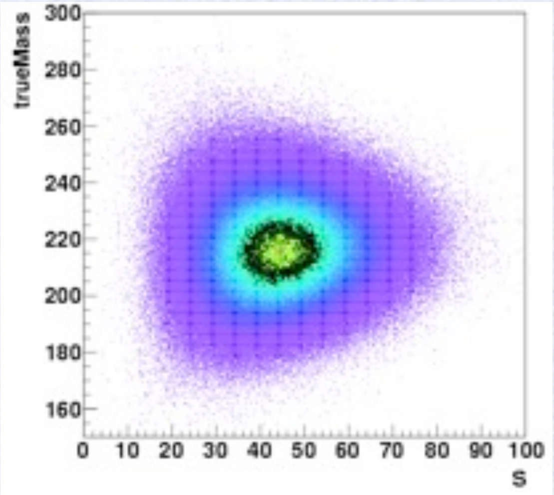

```
MCMCCalculator mc(data, model);
mc.SetConfidenceLevel(0.683);
mc.SetLeftSideTailFraction(0.5);
SequentialProposal sp(0.1);
mc.SetProposalFunction(sp);
mc.SetNumIters(1000000);
mc.SetNumBurnInSteps(50);
MCInterval* interval = bc.GetInterval();
RooRealVar * s = (RooRealVar*)
model.GetParametersOfInterest()->find("s");
double lowerLimit = interval->LowerLimit(*s);
double upperLimit = interval->UpperLimit(*s);
MCMCIntervalPlot plot(*interval);
plot.Draw();
```

BAT Calculator

- **`BATCalculator`** class
  - developed by S. Schmitz & G. Schott
  - provided by the BAT package (not part of Roostats)
    A. Caldwell, D. Kollar, K. Kröninger, Comp. Physics Comm. 180 (2009) 2197
    see also http://www.mppmu.mpg.de/bat/
  - valuable alternative for cross-checks
  - various options for controlling the Markov chain
  - similar interface as other RooStats Bayesian calculator
    - but requires to load first libBAT to use it




```
gSystem->Load("libBAT");
BatCalculator bc(data, model);
batc->SetnMCMC(500000);
MCInterval* interval = bc.GetInterval();
```



BATCalculator for a 2-dim problem

RooStats Lecture and Tutorials Part2

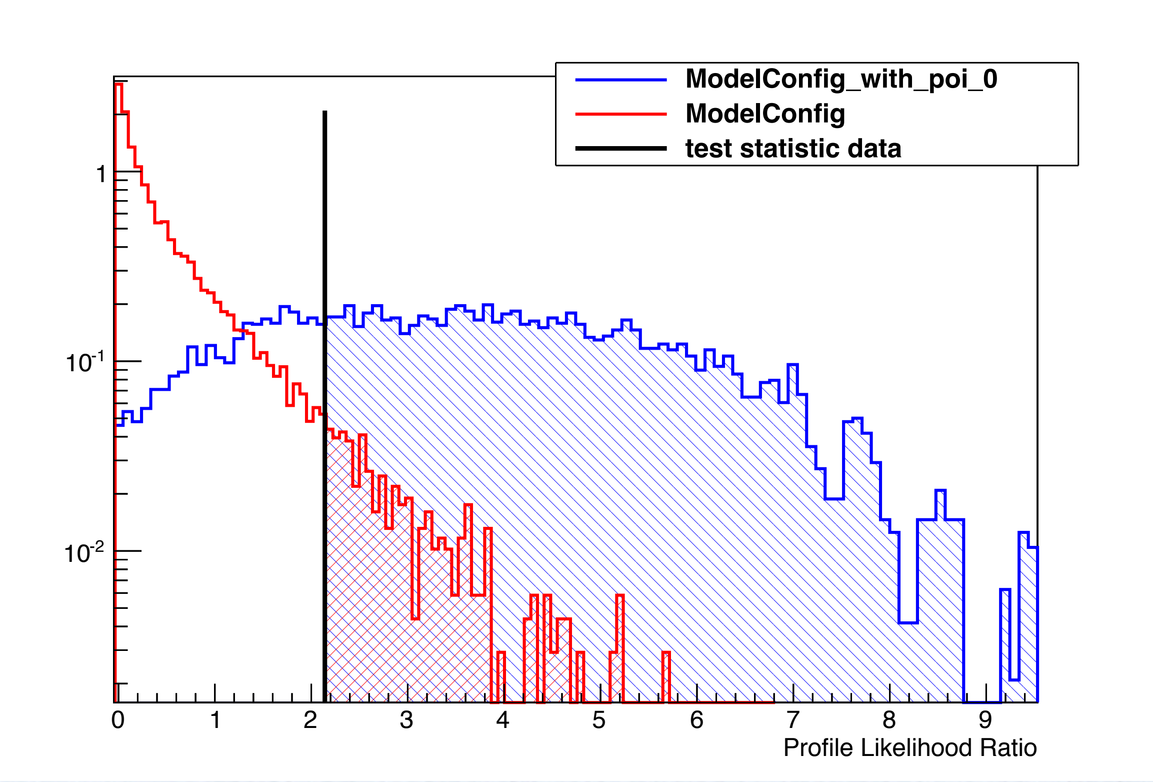
- Hypothesis tests in RooStats using toys and asymptotic formulae
- Hypothesis test inversion
  - Limit and interval calculators
    - CLs, Feldman-Cousins

Frequentist Hypothesis Tests

- Ingredients:
  - Null Hypothesis: the hypothesis being tested (e.g. $\theta = \theta_0$), assumed to be true and one tries to reject it
  - Alternate Hypothesis: the competitive hypothesis (e.g. $\theta \neq \theta_0$)
  - w is the critical region, a subspace of all possible data:
    - size of test : $\alpha = P(X \in w \mid H_0)$
    - power of test : $1- \beta = P(X \in w \mid H_1)$
  - Test statistics: a function of the data, t(X) ,used for defining the critical region in multidimensional data: $X \in w \rightarrow t(X) \in w_t$

RooStats Hypothesis Test

- Define null and alternate model using ModelConfig
  - can use ModelConfig::SetSnapshot(const RooArgSet &) to define parameter values for the null in case of a common model (e.g. $\mu = 0$ for the B model)
- Select test statistics to use
- Select calculator
  - Use toys or asymptotic formula to get sampling distribution of test statistics
  - FrequentistCalculator or HybridCalculator have different treatment of nuisance parameters



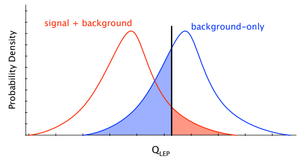
Test Statistics

- Test statistics maps multidimensional space in one, in a way relevant to the hypothesis being tested

RooStats has the three common test statistics used in the field (and more)

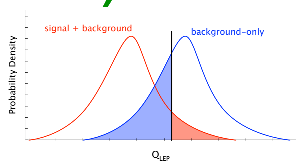
- simple likelihood ratio (used at LEP, nuisance parameters fixed)

$$Q_{LEP} = L_{s+b}(\mu = 1)/L_b(\mu = 0)$$



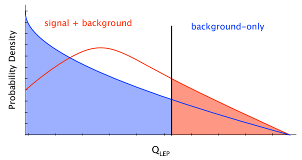
- ratio of profiled likelihoods (used commonly at Tevatron)

$$Q_{TEV} = L_{s+b}(\mu = 1, \hat{\nu})/L_b(\mu = 0, \hat{\hat{\nu}}')$$



- profile likelihood ratio (related to Wilks's theorem)

$$\lambda(\mu) = L_{s+b}(\mu, \hat{\hat{\nu}})/L_{s+b}(\hat{\mu}, \hat{\nu})$$



- preferred choice is profile likelihood ratio which has known asymptotic distribution

FrequentistCalculator

- Generate toys using nuisance parameter at their conditional ML estimate ($\theta = \hat{\theta}_{\mu}$) by fitting them to the observed data
- Treat constraint terms in the likelihood (e.g. systematic errors) as auxiliary measurements
  - introduce global observables which will be varied (tossed) for each pseudo-experiment
  - $L = \text{Poisson}(n_{obs} \mid \mu + b)\ \text{Gaussian}(b_0 \mid b, \sigma_b)$
    - b_0 is a global observables, varied for each toys but it needs to be considered constant when fitting
    - n_{obs} is the observable which is part of the data set
    - μ is the parameter of interest (poi)
    - b is the nuisance parameter

HybridCalculator

- Nuisance parameters are integrated using their pdf (the constraint term) which is interpreted as a Bayesian prior
  - integration is done by generating for each toys different nuisance parameters values
  - need to have a pdf for the nuisance parameters (often it can be derived automatically from the model)

$$L = \text{Poisson}(\, n_{obs} \mid \mu + b)\, \text{Gaussian}(\, b \mid b_0, \sigma_b)$$

$$\Downarrow$$

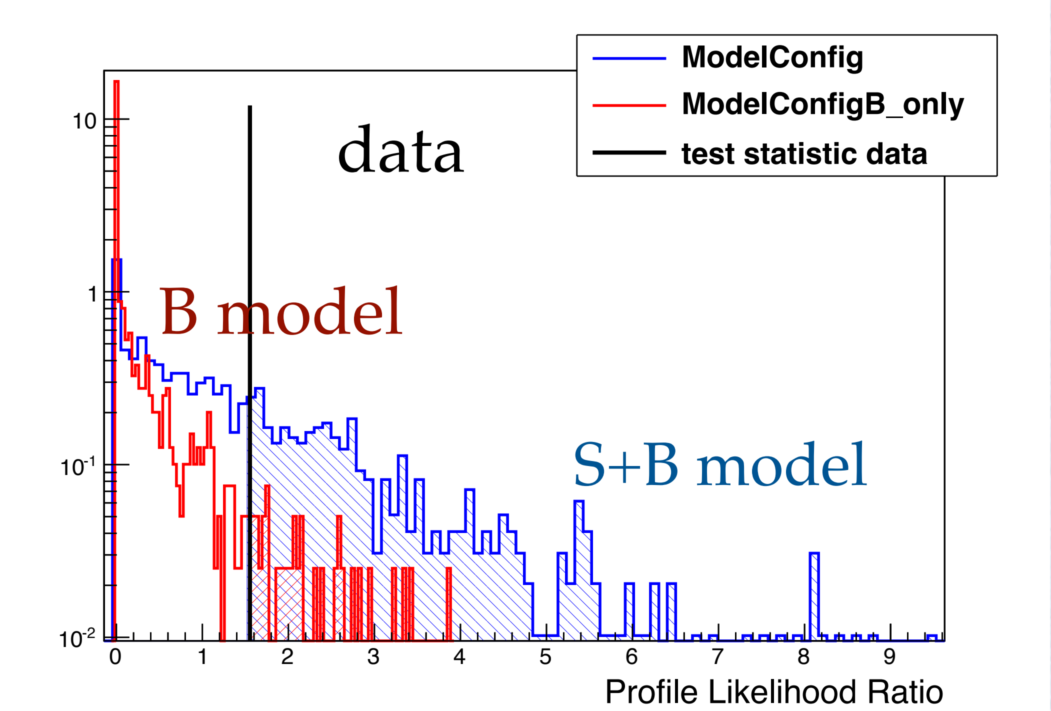
$$L = \int \text{Poisson}(\, n_{obs} \mid \mu + b)\, \text{Gaussian}(\, b \mid b_0, \sigma_b)\, db$$

Example Hypo Test

- Define the models
  - N.B for discovery significance null is B model and alt is S+B

```
// create first HypoTest calculator (data, alt model , null model)
FrequentistCalculator fcalc(*data, *sbModel, *bModel);

// create the test statistics
ProfileLikelihoodTestStat profll(*sbModel->GetPdf());
// use one-sided profile likelihood for discovery tests
profll.SetOneSidedDiscovery(true);

// configure  ToyMCSampler and set the test statistics
ToyMCSampler *toymcs = (ToyMCSampler*)fcalc.GetTestStatSampler();
toymcs->SetTestStatistic(&profll);

fcalc.SetToys(1000,1000);  // set number of toys for (null, alt)

// run the test
HypoTestResult * r = fcalc.GetHypoTest();
r->Print();

// plot test statistic distributions
HypoTestPlot * plot = new HypoTestPlot(*r);
plot->Draw();
```

```
Results HypoTestCalculator_result:
 - Null p-value = 0.034 +/- 0.00573097
 - Significance = 1.82501 sigma
 - Number of Alt toys: 1000
 - Number of Null toys: 1000
```



AsymptoticCalculator

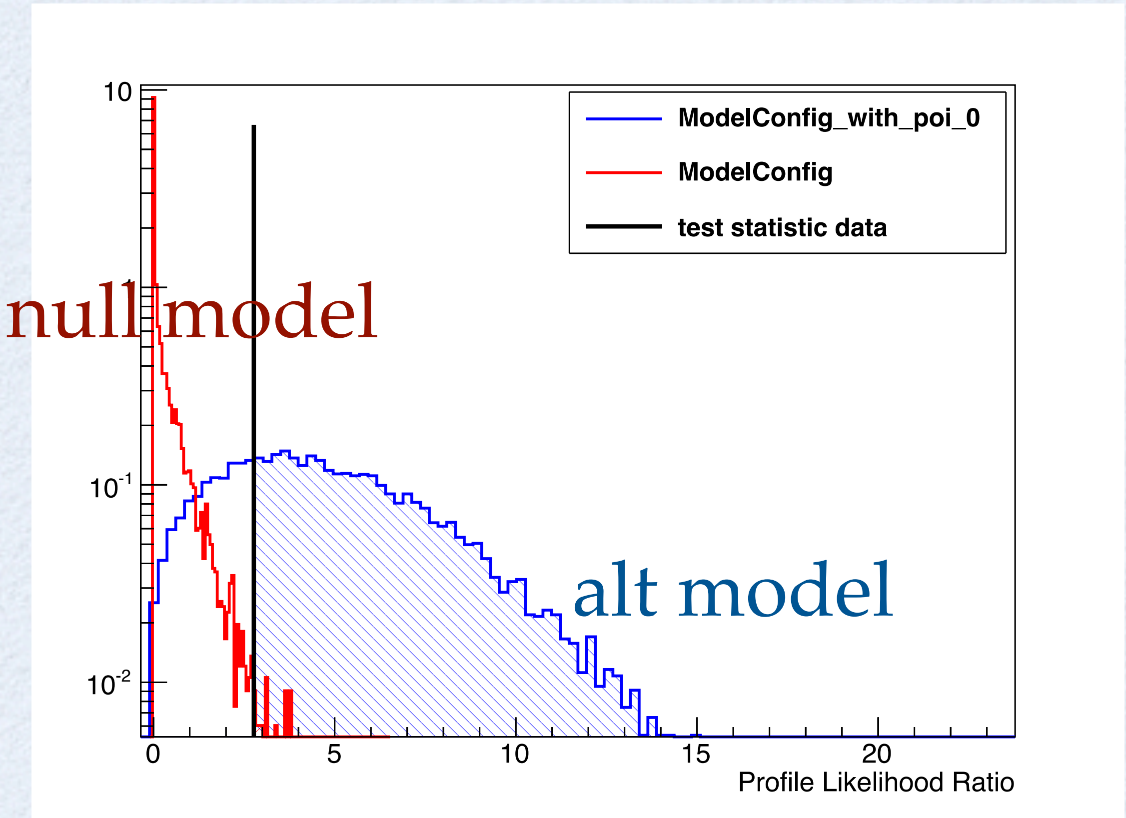
- Use the asymptotic formula for the test statistic distributions
- one-sided profile likelihood test statistic:
  - null model ($\mu = \mu_{\mathrm{TEST}}$)
    - half χ^2 distribution
  - alt model ($\mu \neq \mu_{\mathrm{TEST}}$)
    - non-central χ^2
    - use Asimov data to get the non centrality parameter $\Lambda = (\mu\text{-}\mu_{\mathrm{TEST}})/\sigma$
- p-values for null and alternate can be obtained without generating toys

$$\lambda(\mu) = \frac{L(x|\mu,\hat{\hat{\nu}})}{L(x|\hat{\mu},\hat{\nu})}$$

$\lambda(\mu) = 0$ for

$\hat{\mu} < 0$ (discovery)

$\hat{\mu} < \mu_{\mathrm{TEST}}$ (limits)



➡ see Cowan, Cranmer, Gross, Vitells, arXiv:1007.1727,EPJC 71 (2011) 1-1

Inversion of Hypothesis Tests

- one-to-one mapping between hypothesis tests and confidence intervals

Table 20.1 Relationships between hypothesis testing and interval estimation

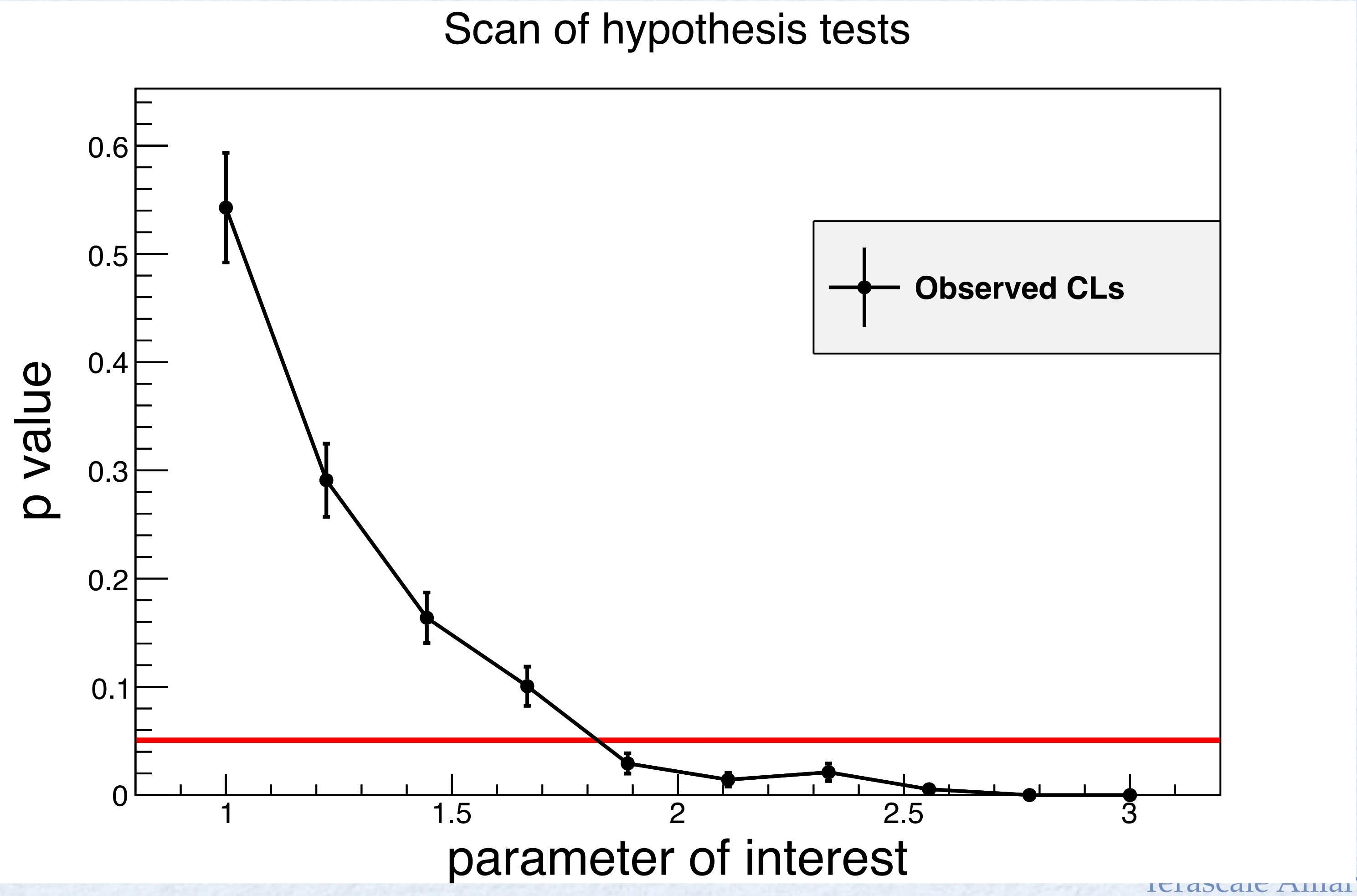
Property of test	Property of corresponding confidence interval
Size $= \alpha$	Confidence coefficient $= 1 - \alpha$
Power $=$ probability of rejecting a false value of $\theta = 1 - \beta$	Probability of not covering a false value of $\theta = 1 - \beta$
Most powerful	Uniformly most accurate
$\longleftarrow \left\{ \begin{matrix} \textit{Unbiased} \\ 1 - \beta \geq \alpha \end{matrix} \right\} \longrightarrow$	
Equal-tails test $\alpha_1 = \alpha_2 = \frac{1}{2}\alpha$	Central interval

from G. Feldman visiting Harvard statistics department

They explained that in statistical theory there is a one-to-one correspondence between a hypothesis test and a confidence interval. (The confidence interval is a hypothesis test for each value in the interval.) The Neyman-Pearson Theorem states that the likelihood ratio gives the most powerful hypothesis test. Therefore, it must be the standard method of constructing a confidence interval.

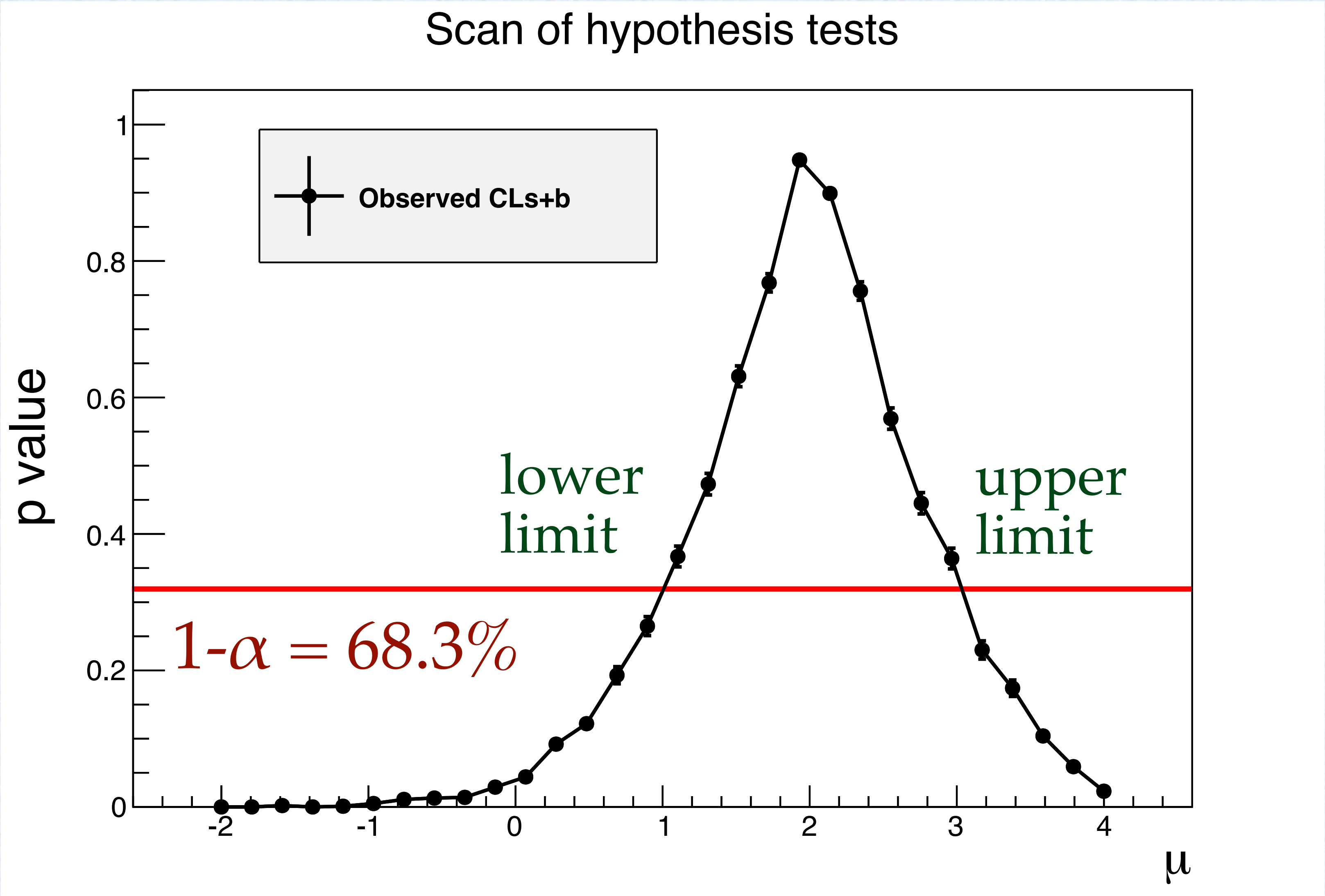
Hypothesis Test Inversion

- Performing an hypothesis test at each value of the parameter
- Interval van be derived by inverting the p-value curve, function of the parameter of interest (μ)
  - value of μ which has p-value α (e.g. 0.05), is the upper limit of 1-α confidence interval (e.g. 95%)



Hypothesis Test Inversion

- use one-sided test for upper limits (e.g. one-side profile likelihood test statistics)
- use two-sided test for a 2-sided interval



Example: 1-σ interval for a Gaussian measurement

HypoTestInverter class

- Input:
  - Hypothesis Test calculator (e.g. FrequentistCalculator)
    - possible to customize test statistic, number of toys, etc..
    - N.B: null model is S+B, alternate is B only model
- Interval calculator class
  - scan given interval of μ and perform hypothesis tests
  - compute upper/lower limit from scan result
    - can use $CL_s = CL_{s+b} / CL_b$ for the p-value
  - store in result (HypoTestInverterResult) also all the hypothesis test results for each scanned μ value
    - possible to merge later results
- Can compute expected limits and bands

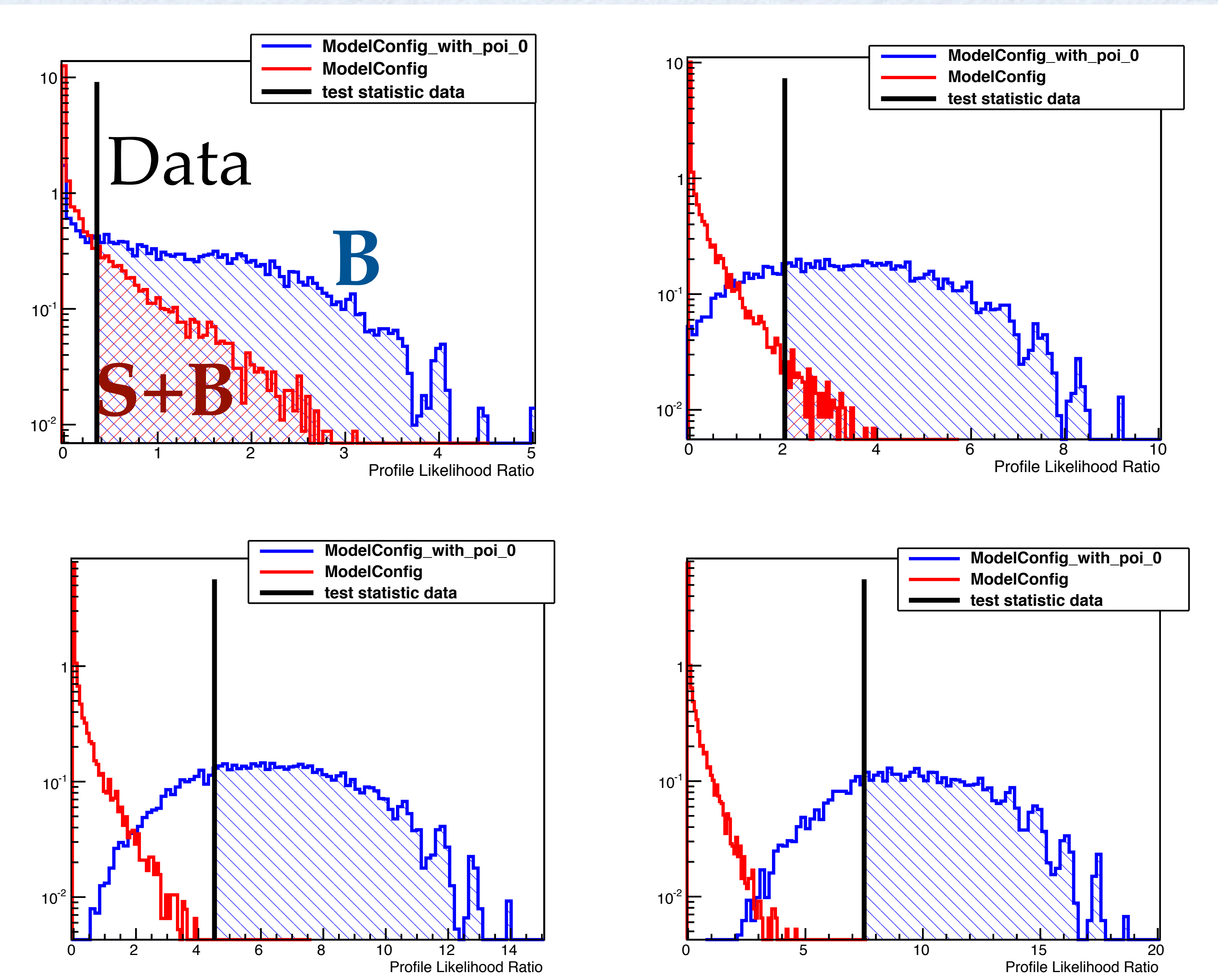
HypoTestInverter

- **HypoTestInverter** class in RooStats

```
// create first HypoTest calculator (N.B null is s+b model)
FrequentistCalculator fc(*data, *bModel, *sbModel);

HypoTestInverter calc(*fc);
calc.UseCLs(true);

// configure  ToyMCSampler and set the test statistics
ToyMCSampler *toymcs = (ToyMCSampler*)fc.GetTestStatSampler();

ProfileLikelihoodTestStat profll(*sbModel->GetPdf());
// for CLs (bounded intervals) use one-sided profile likelihood
profll.SetOneSided(true);
toymcs->SetTestStatistic(&profll);

// configure and run the scan
calc.SetFixedScan(npoints,poimin,poimax);
HypoTestInverterResult * r = calc.GetInterval();

// get result and plot it
double upperLimit = r->UpperLimit();
double expectedLimit = r->GetExpectedUpperLimit(0);

HypoTestInverterPlot *plot = new HypoTestInverterPlot("hi","",r);
plot->Draw();
```

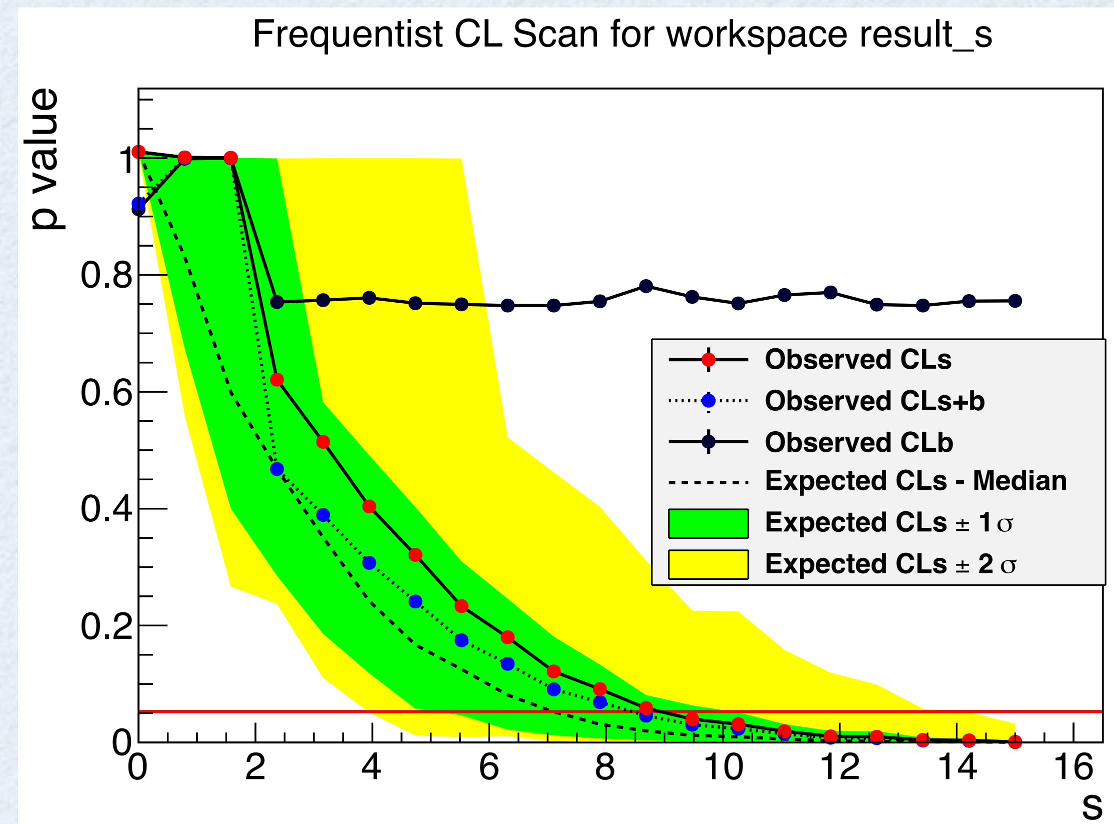
Running the HypoTestInverter

Hypothesis test results for each scanned point



p-value, CL_{s+b} (or CL_b) is integral of S+B (or B) test statistic distribution from data value

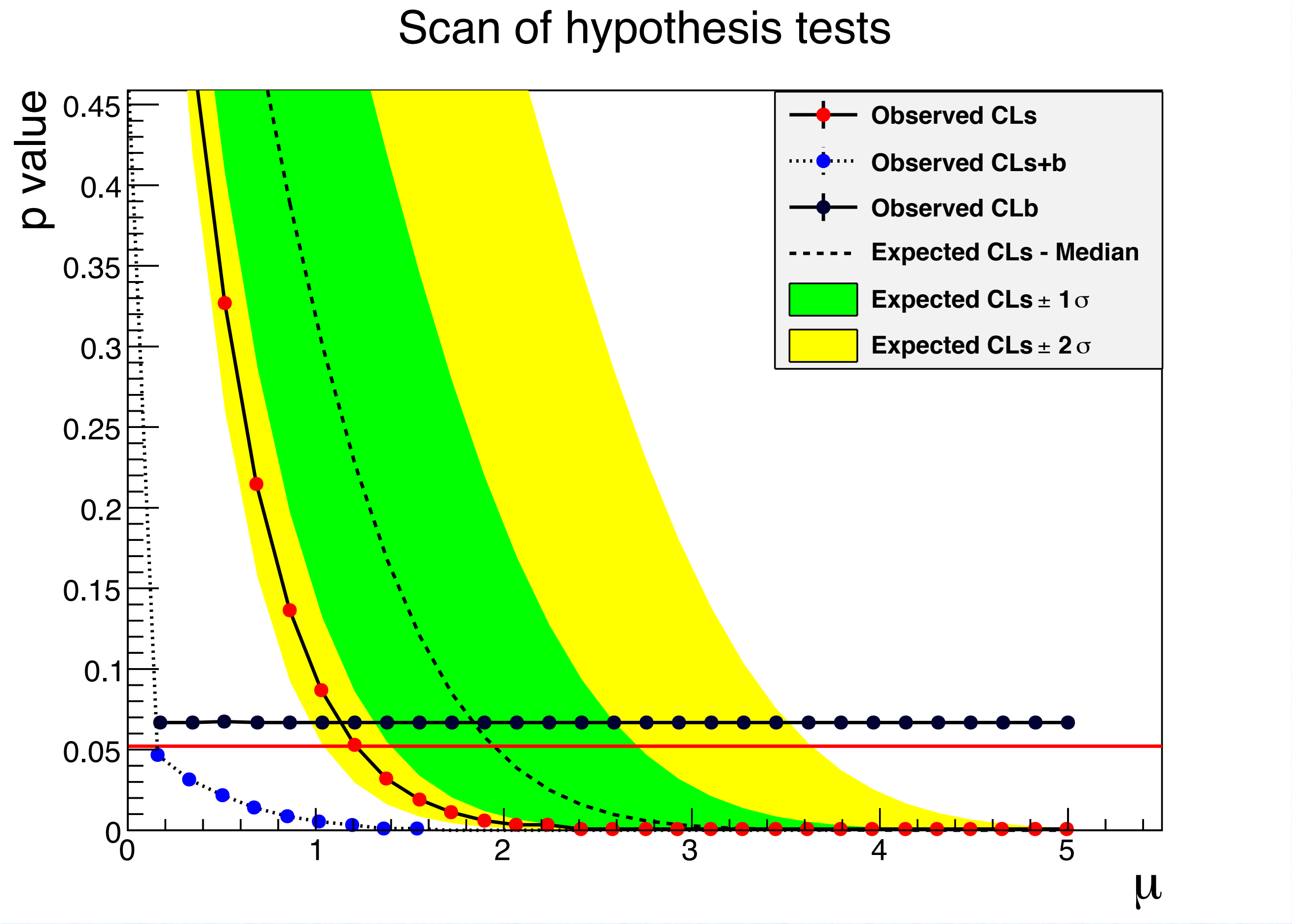
Scan result

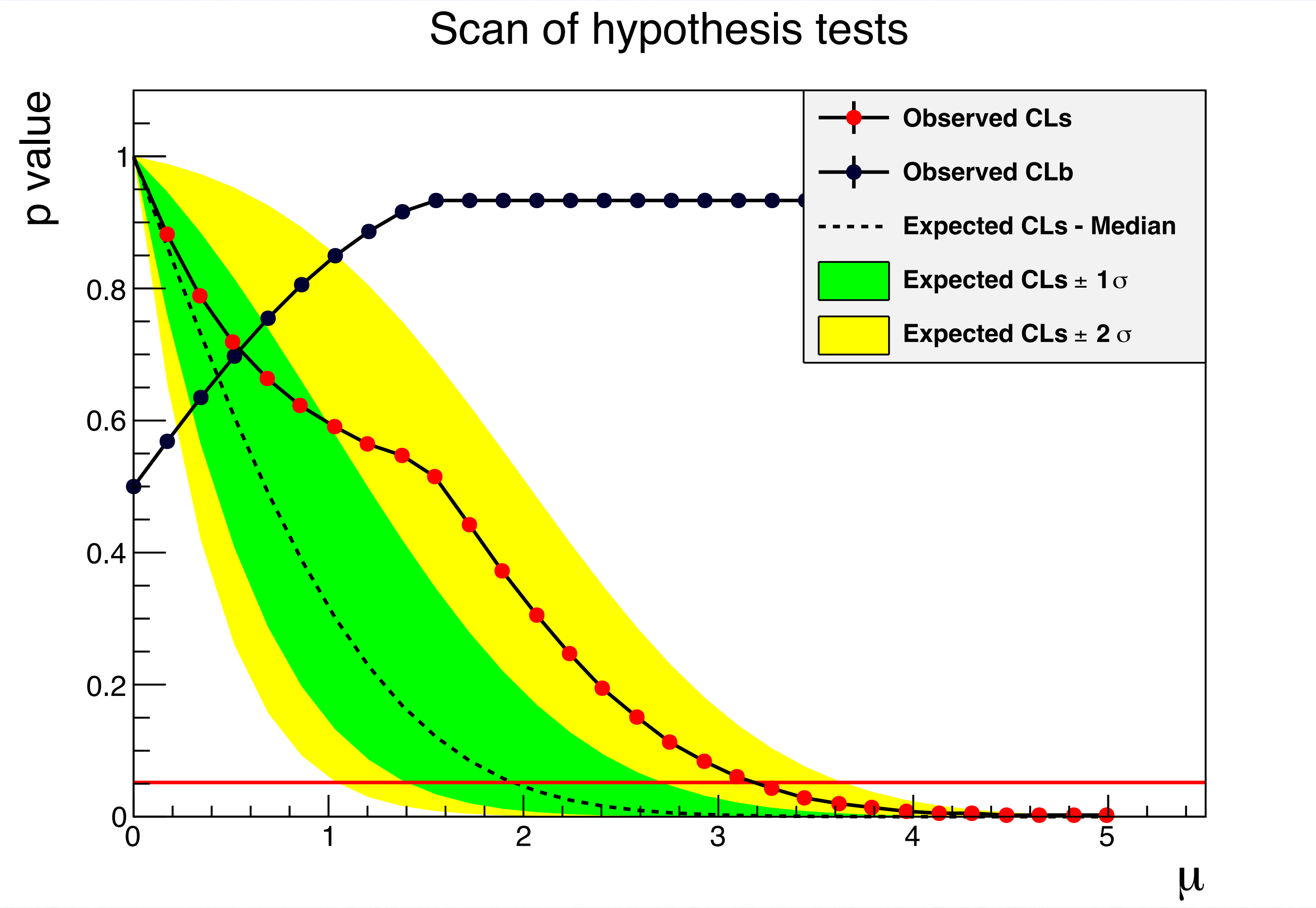


Expected limit and bands are obtained by replacing data test statistic value with quantiles of the B test stat. distribution

Example of Scan

- 95% CL limit on a Gaussian measurement:
  - Gauss(x,μ,1), with μ≥0





deficit, observation x = -1.5

excess, observation x = 1.5

use CL_s as p-value to avoid setting limits which are too good

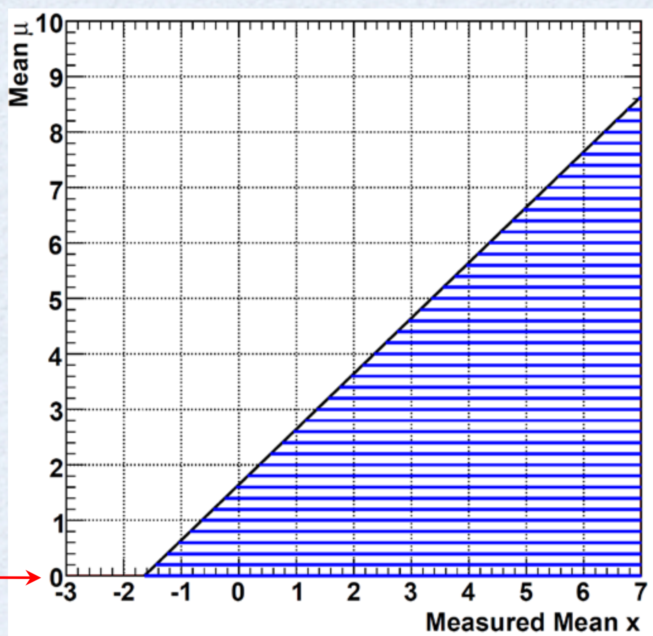
Limits on bounded measurements

from Bob Cousins:

Downward fluctuations in searches for excesses

Classic example: Upper limit on mean μ of Gaussian based on measurement *x* (in units of σ).

μ=0



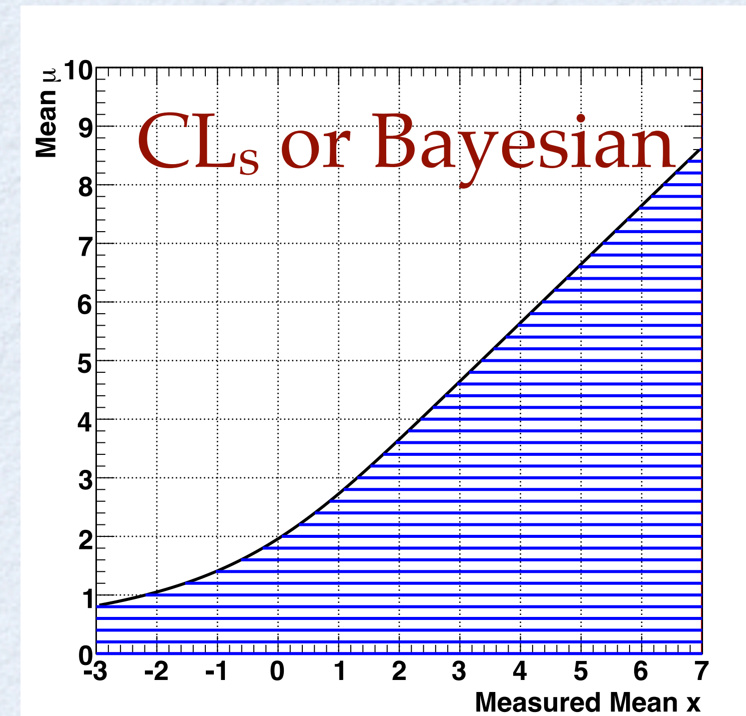
If $\mu \geq 0$ in model, as measured x becomes increasingly negative, standard classical upper limit becomes small and then null.

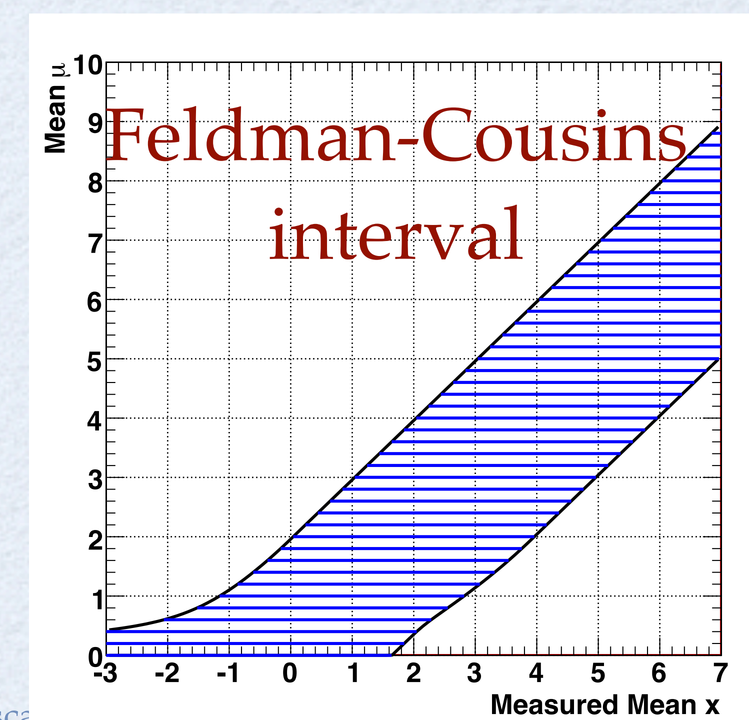
Issue acute 15-25 years ago in expts to measure ν_e mass in (tritium β decay): several measured $m_\nu^2 < 0$.

Frequentist 1-sided 95% C.L. Upper Limits, based on α = 1 – C.L. = 5% (called CL_{sb} at LEP).

For *x* < –1.64 σ the confidence interval is the *null* set!

Bob Cousins, CMSDAS, 1/2012

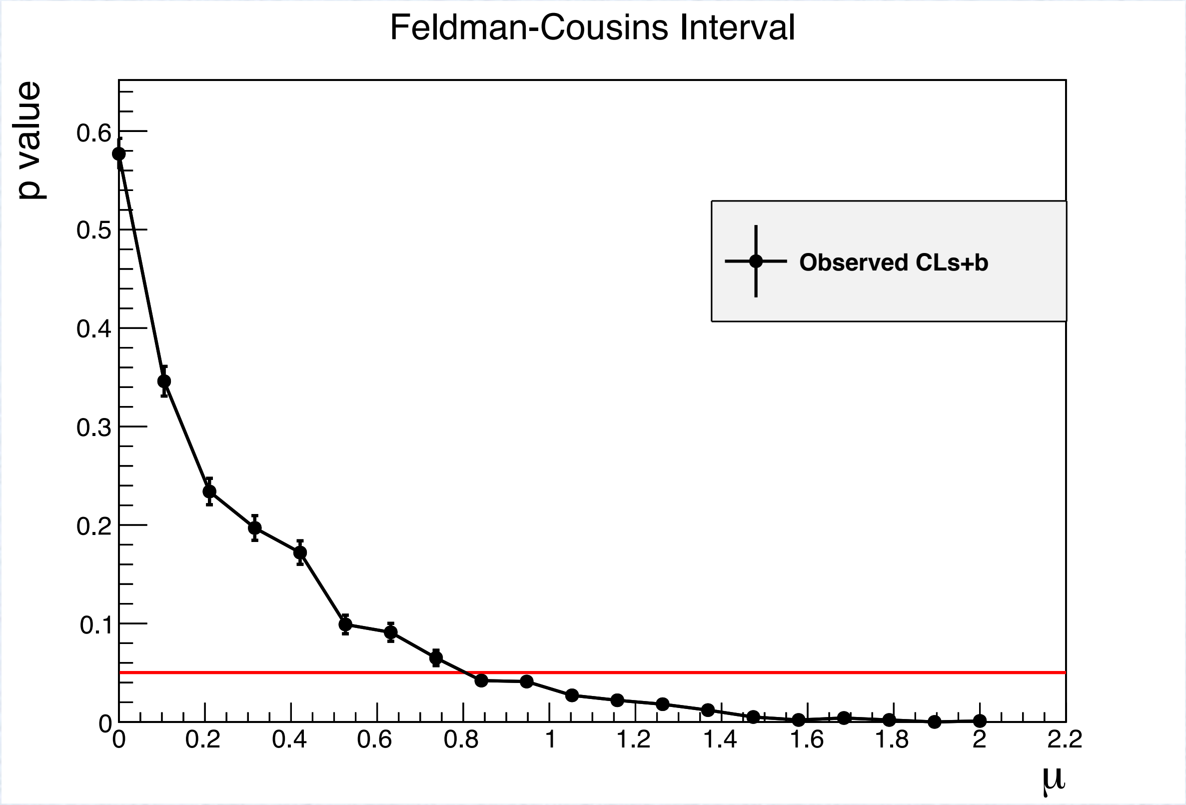




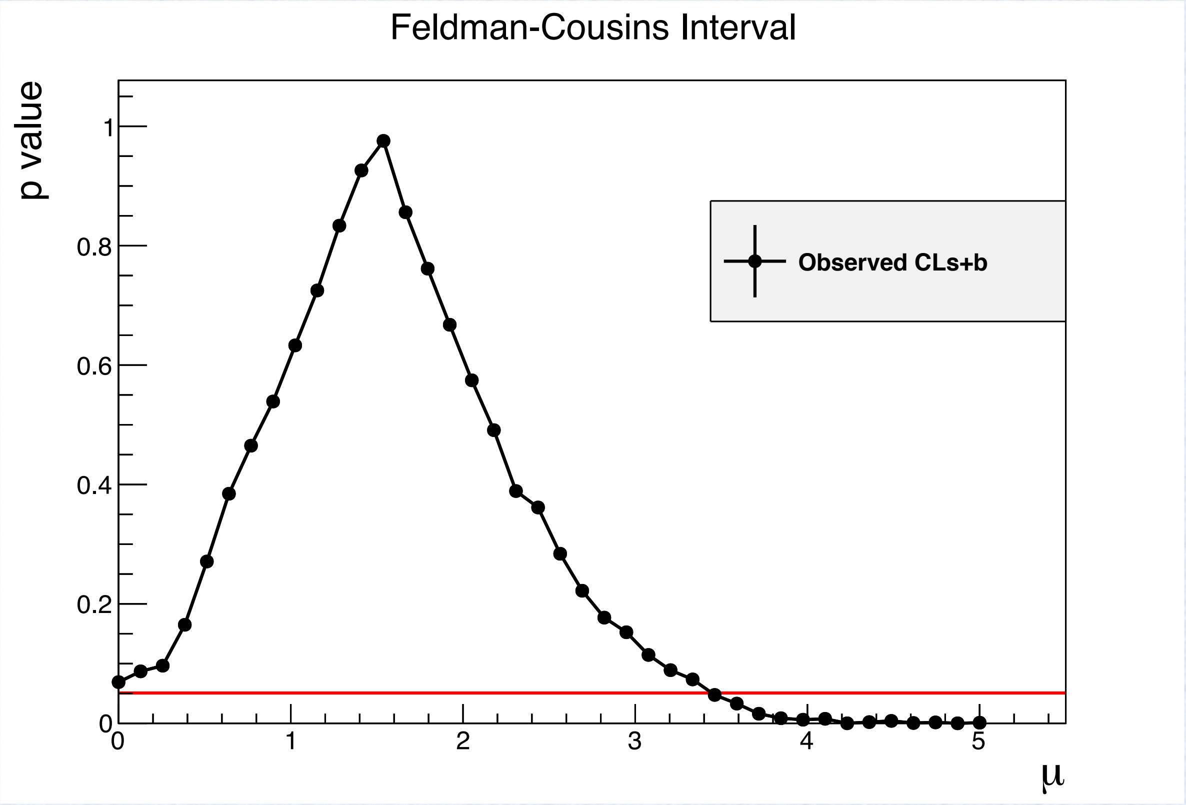
Feldman-Cousins intervals

- HypoTestInverter class can compute also a Feldman-Cousins interval
  - need to use FrequentistCalculator and CL_{s+b} as p-value
  - use the 2-sided profile likelihood test statistic

$$\lambda(\mu) = \frac{L(x|\mu, \hat{\hat{\nu}})}{L(x|\hat{\mu}, \hat{\nu})}$$



observation x = -1.5



observation x = 1.5

Feldman-Cousins Interval

from Kyle Cranmer:

A different way to picture Feldman-Cousins

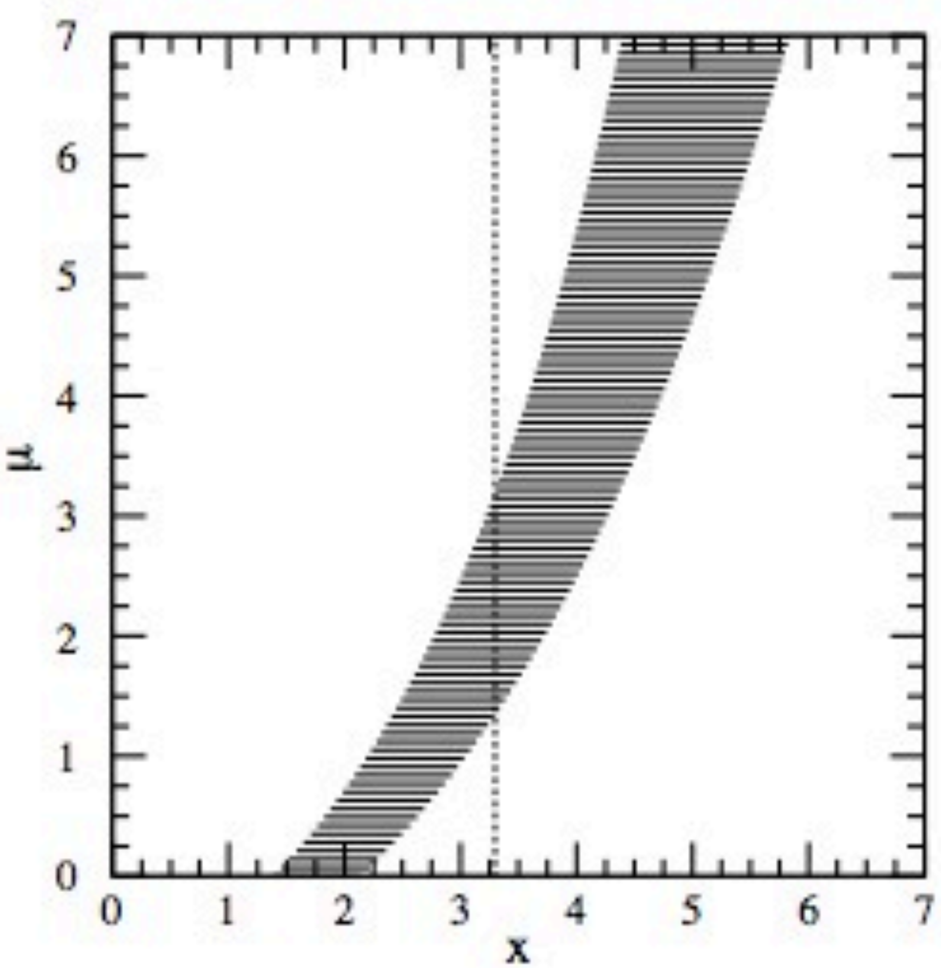
CENTER FOR COSMOLOGY AND PARTICLE PHYSICS

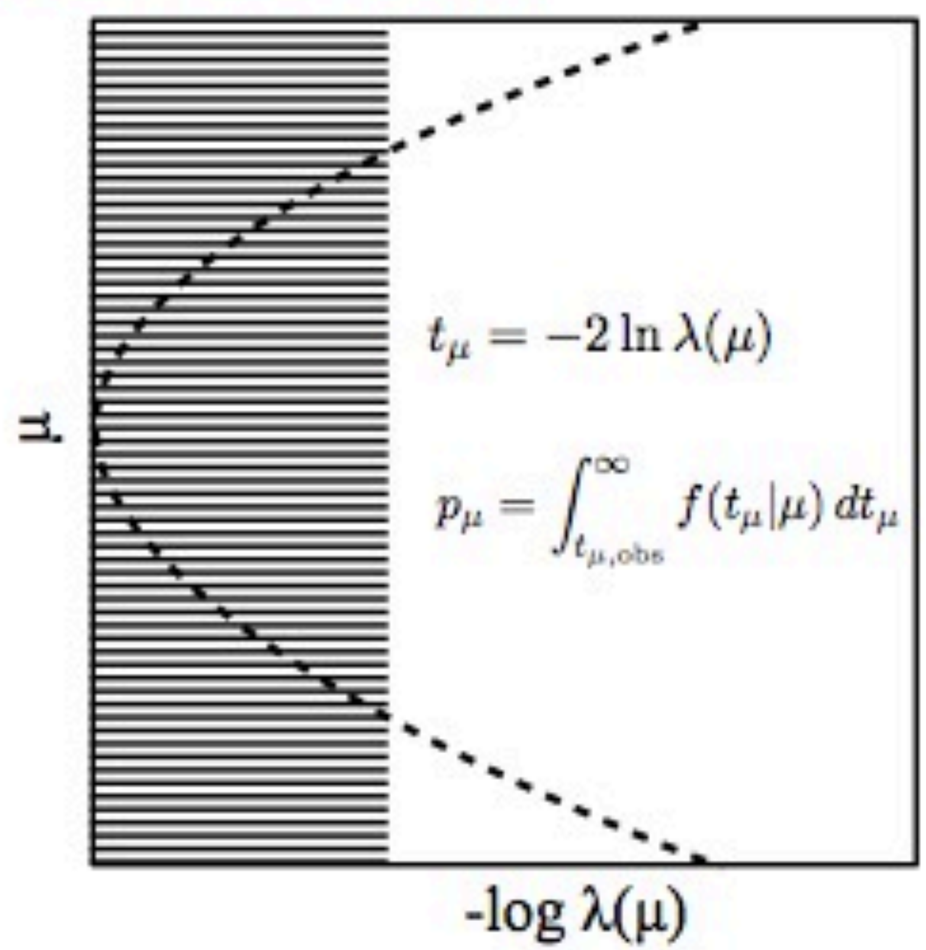
Most people think of plot on left when thinking of Feldman-Cousins

- bars are regions "ordered by" $R = P(n|\mu)/P(n|\mu_{\text{best}})$, with $\int_{x_1}^{x_2} P(x|\mu)dx = \alpha.$

But this picture doesn't generalize well to many measured quantities.

- Instead, just use R as the test statistic... and R is $\lambda(\mu)$

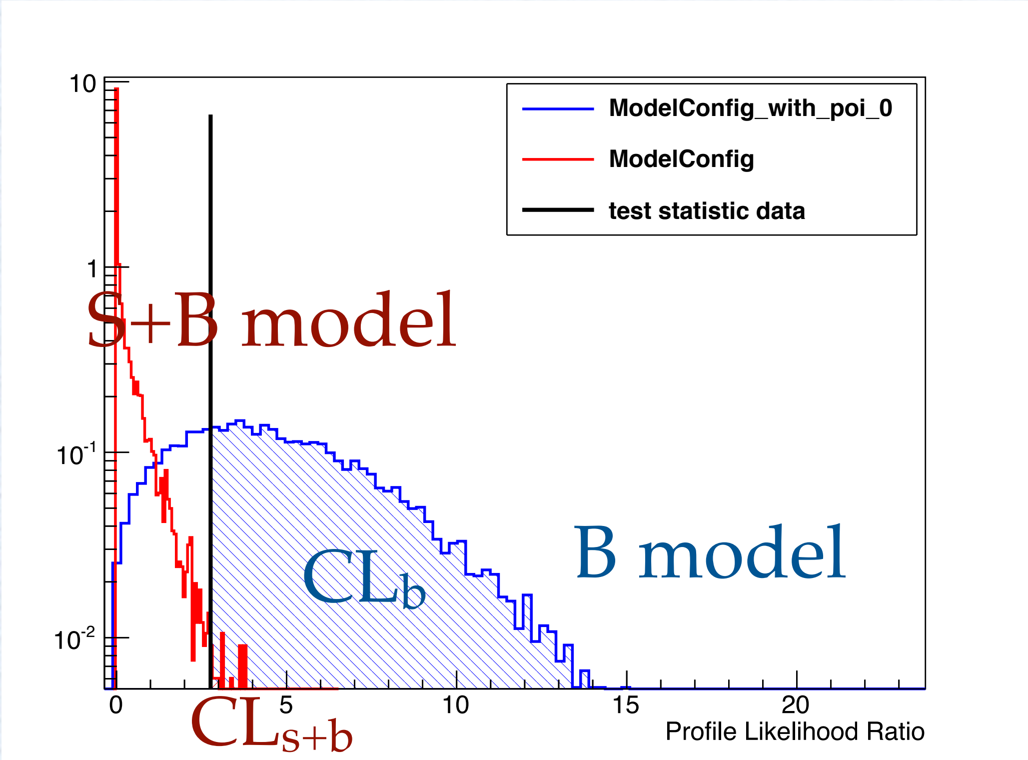


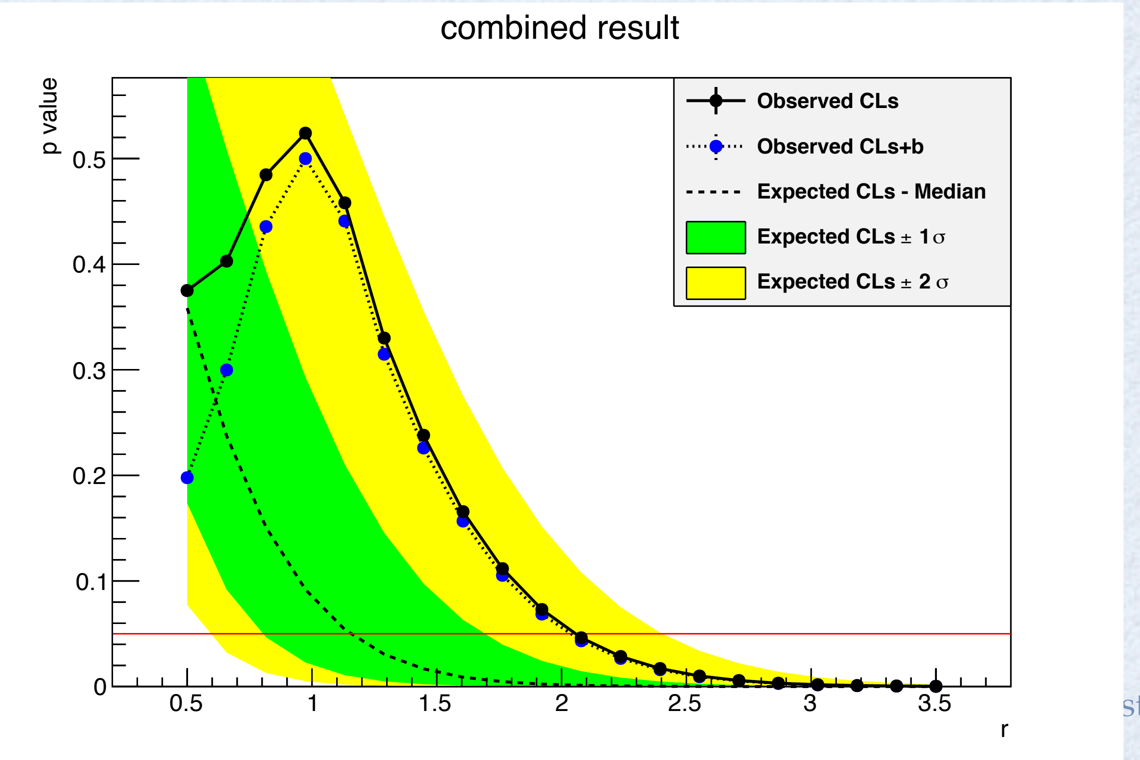


Asymptotic Limits

- **AsymptoticCalculator** class for HypoTestInverter
  - use the asymptotic formula for the test statistic distributions
    - χ^2 approximation for the profile likelihood ratio
      - see G. Cowan *et al.*, arXiv:1007.1727,EPJC 71 (2011) 1-1
  - p-values CL_{s+b} (null) and CL_b (alt) obtained without generating toys
  - also expected limits from the alt distribution

```
// create first HypoTest calculator (N.B null is s+b model)
AsymptoticCalculator ac(*data, *bModel, *sbModel);

HypoTestInverter calc(*ac);
// run inverter same as using other calculators
........
```





StandardHypoTestInvDemo.C

- Standard ROOT macro to run the Hypothesis Test inversion.
- Inputs to the macro:
  - workspace file, workspace name
  - name of S+B model (null) and for B model (alt)
    - if no B model is given, use S+B model with poi = 0
  - data set name
  - calculator type: frequentist (= 0), hybrid (=1), or asymptotic (=2)
  - test statistics
- options:
  - use CL_s or CL_{s+b} for computing limit
  - number of points to scan and min, max of interval

load the macro after having created the workspace and saved in file SPlusBExpoModel.root

```
root[] .L StandardHypoTestInvDemo.C
```

run for CLs (with frequentist calculator (type = 0) and one-side PL test statistics (type = 3) scan 10 points in [0,100]

```
root[] StandardHypoTestInvDemo("SPlusBExpoModel.root","w","ModelConfig","","data",0,3, true, 10, 0, 100)
```

run for Asymptotic CLs (scan 20 points in [0,100])

```
root[] StandardHypoTestInvDemo(SPlusBExpoModel.root","w","ModelConfig","","data",2,3, true, 20, 0, 100)
```

run for Feldman-Cousins (scan 10 points in [0,100])

```
root[] StandardHypoTestInvDemo(SPlusBExpoModel.root","w","ModelConfig","","data",0,2, false, 10, 0, 15)
```

RooStats Exercises

documented in a Twiki page

see

https://twiki.cern.ch/twiki/bin/view/RooStats/RooStatsTutorialsAugust2012

Getting Started

- all RooStats classes are in a namespace
- recommended to add at beginning of macro:
  - using namespace RooStats
- this will also load automatically the RooStats library
- note that RooStats methods start with upper case letter while RooFit start with lower case
- RooStats calculator are quite verbose, useful to suppress many info messages"

```
RooMsgService::instance().setGlobalKillBelow(RooFit::WARNING) ;
```

Roostats reference guide:

http://root.cern.ch/root/htmldoc/ROOSTATS_Index.html

RooStats tutorial macros:

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

Exercises

- Code snippets showing how to build the different models and run the RooStats calculator is available in the Twiki page
- Observable-based analysis:
  - Gaussian-distributed signal event over exponential-distributed background.
  - We assume that the location and width of the signal are known.
- Poisson model:
  - you observe n events in data while expecting b background event. We consider some systematic uncertainties in the background model value, b. We express this uncertainty as Gaussian.
    - Modify the model then using Gamma constraint (on/off problem)
    - add additional uncertainty (efficiency) as Log-normal

Exercises

- Compute on the two models, first
  - 68% CL 2-sided confidence interval and optionally significance from the (profiled-) likelihood ratio
  - 68% credible interval using the BayesianCalculator and/or the MCMCCalculator and compare the results
  - Optionally compute the 95% CL upper limits
  - Modify input models by adding extra systematics (e.g. efficiency in Poisson model) and/or using different constraint type (e.g. Log-normal)
- Later
  - 95% CL upper-limit with the inverter method and CLs, first using the frequentist calculator (with toys) then using the asymptotic calculator.
  - A significance test using the FrequentistCalculator or the AsymptoticCalculator

see Twiki page: https://twiki.cern.ch/twiki/bin/view/RooStats/RooStatsTutorialsAugust2012

RooStats Tutorials

- RooStats provides standard tutorials taking all as input workspace, ModelConfig and data set names
- StandardProfileLikelihoodDemo.C

  run ProfileLikelihoodCalculator - get interval and produce plot

  ```
  root[]StandardProfileLikelihoodDemo("ws.root","w","ModelConfig","data")
  ```

- StandardBayesianNumericalDemo.C

  run Bayesiancalculator: get a credible interval and produce plot of posterior function

  ```
  root[]StandardBayesianNumericalDemo("ws.root","w","ModelConfig","data")
  ```

- StandardBayesianMCMCDemo.C

  run bayesian MCMCCalculator: get a credible interval and produce plot of posterior function

  ```
  root[]StandardBayesianMCMCDemo("ws.root","w","ModelConfig","data")
  ```

Documentation and References

- RooStats TWiki: https://twiki.cern.ch/twiki/bin/view/RooStats/WebHome
- RooStats users guide (still under development, to be completed)
  - http://root.cern.ch/viewcvs/branches/dev/roostats/roofit/roostats/doc/usersguide/RooStats_UsersGuide.pdf
- Paper: ACAT 2010 proceedings: http://arxiv.org/abs/1009.1003
- ROOT reference guide: http://root.cern.ch/root/htmldoc/ROOSTATS_Index.html
- RooFit and RooStats tutorial macros: http://root.cern.ch/root/html/tutorials
- RooFit's users guide: http://root.cern.ch/drupal/content/users-guide
- RooStats tutorials:
  - see links in RooStats Twiki page
  - e.g. tutorials at Desy school of statistics: https://indico.desy.de/conferenceOtherViews.py?view=standard&confId=5065
- RooStats user support:
  - Request support via ROOT talk forum: http://root.cern.ch/phpBB2/viewforum.php?f=15 (questions on statistical concepts accepted)
  - Submit bugs to ROOT Savannah: https://savannah.cern.ch/bugs/?func=additem&group=savroot
- Contacts for statistical questions:
  - ATLAS statistics forum: hn-atlas-physics-Statistics@cern.ch (Cowan, Gross, Read et al)
    - TWiki: https://twiki.cern.ch/twiki/bin/view/AtlasProtected/StatisticsTools
  - CMS statistics committee: (Cousins, Demortier, Lyons et al)
- Statistics
  - See various statistic lectures by G. Cowan, K. Cranmer or B. Cousins