Advanced Statistics Course – Part III

W. Verkerke (NIKHEF)

Nuisance parameters: what are they and what do you do with them?

- 1 – Definition of nuisance parameters
  - A nuisance parameter is any parameter of the model that is not a parameter-of-interest (for physics).
    - Example: for Higgs discovery N(higgs) is of interest, everything else is nuisance
- 2 – Introduction of nuisance parameters in Likelihood
  - Sometimes nuisance parameter arise naturally in the likelihood.
  - Systematic uncertainties always introduce nuisance parameters, but explicit parameterization not always obvious (e.g. how to parameterize effect of Pythia-vs-Herwig?)
- 3 – Treatment of nuisance parameters in inference
  - Each of the three main classes of constructing intervals (Bayesian, likelihood ratio, Neyman confidence intervals) has a different way to incorporate the uncertainty on the nuisance parameters in the parameters of interest.

Systematic uncertainties

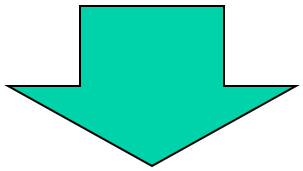
- Common source of systematic uncertainty in particle physics are
- Detector systematics
  - Jet energy scale uncertainty
  - Flavor tagging efficient / mistag rate uncertainty
  - Jet-to electron fake rates (reconstruction mistakes)
- Theory systematics
  - Parton showering model uncertainty (e.g. Pythia vs Herwig)
  - Cross-section uncertainties
  - Parton densitity function uncertainties
- Simulation statistics
  - Finite size of simulated event samples model signal and background distributions

Systematic uncertainties – the naïve approach

- Naïve approach to incorporating systematic uncertainties in a ***measurement***
- Measure parameter of interest (e.g ML fit using $L(x|\mu)$)
- Example: include a 5% Jet-Energy-Scale systematic
  - Construct alternate model $L_{JESup}(x|\mu)$ that is built with signal and background models reflecting a JES that is moved up by 5% and use that measure μ_{JESup}
  - Construct alternate model $L_{JESdn}(x|\mu)$ that is built with signal and background models reflecting a JES that is down up by 5% and use that to measure μ_{JESdn}
  - Calculate uncertainty due to JES as $\boldsymbol{\sigma(\mu)_{JES}} = (\mu_{JESup} - \mu_{JESdn})/2$
- Repeat for each systematic uncertainty, then add all uncertainties in quadrature
  - $\sigma(\mu)2 = \sigma(\mu)_{stat}^2 + \sigma(\mu)_{systA}^2 + \sigma(\mu)_{statB}^2 \ldots$

Issues with the naïve approach

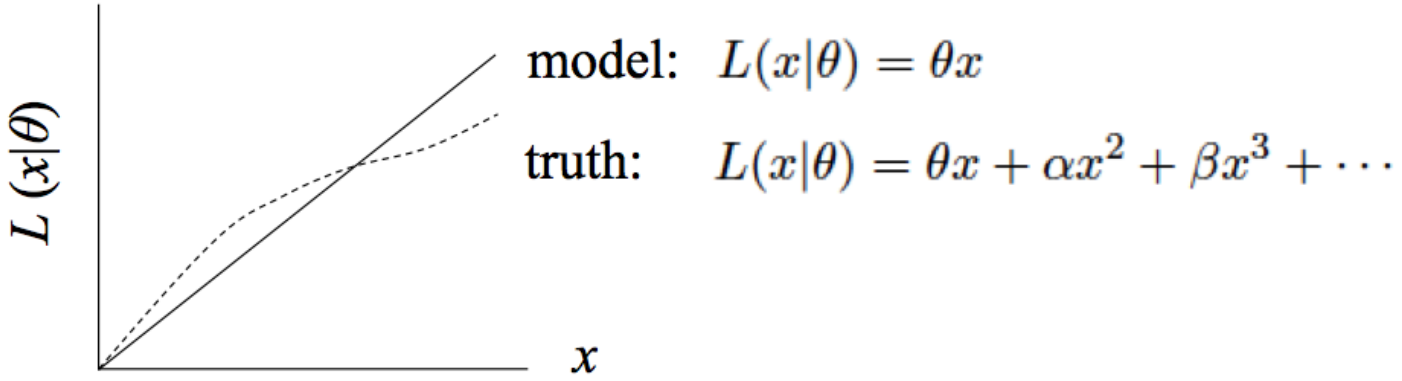
- All systematic uncertainties are treated as uncorrelated
- It only works for measurements, not for hypothesis testing or limit setting



- To incorporate effect of systematic uncertainties in hypothesis testing, must include it in the likelihood

What are nuisance parameters?

- In general, our mode of the data is not perfect

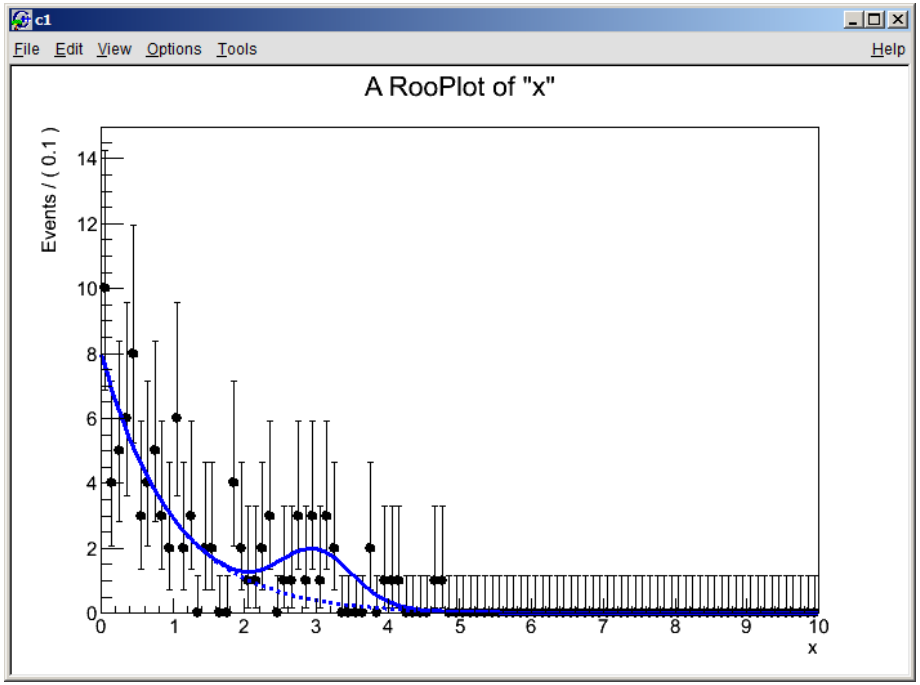


- Can improve modeling by including additional adjustable parameters $L(x|\theta) \rightarrow L(x|\theta, \nu)$
- Nuisance parameters can be used to model systematic uncertainties: Some point in the parameter space of the enlarged model should be "true"
- Presence of nuisance parameters decreases the sensitivity of the analysis of the parameter(s) of interest

Introducing nuisance parameter in the likelihood

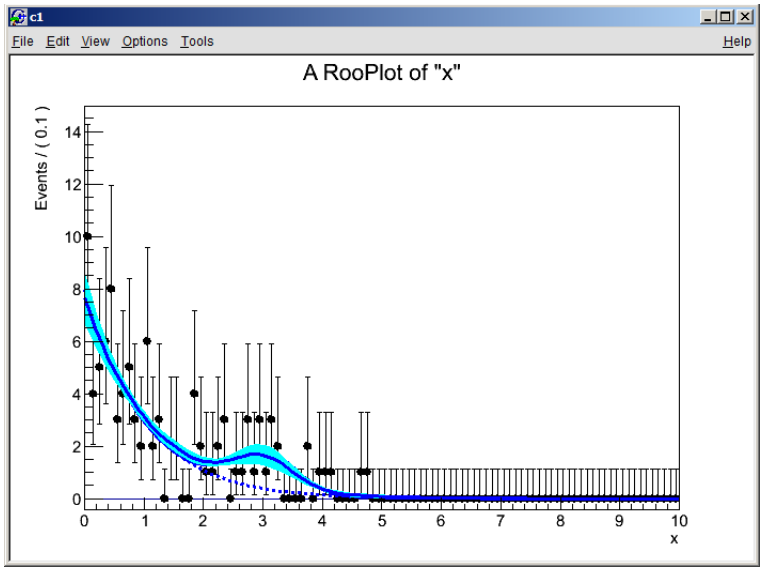
Introducing nuisance parameters in the likelihood

- Any parameter in a likelihood (or probability density function) that is not a parameter-of-interest is a nuisance parameter
- Example: Gaussian+Exponential model from yesterdays exercises: parameter-of-interest is signal yield
- If slope of exponential, width or mean are introduced as floating parameters
  these are in fact
  'nuisance parameters'
  - Their introduction makes the model more correct (assuming true values of width, slope and mean were not known)
  - Their introduction allows to model to assume more correct values for these NPs, but weakens the sensitivity to the POI



Subsidiary measurements

- In the preceding example, the likelihood model had – with sufficient statistics – sensitivity to constrain the nuisance parameters from the data



Error band visualize approx 68% uncertainty on model due to measured uncertainty on slope

- This is not always possible → Introduce a 'subsidiary measurement' in the likelihood that describes external knowledge

A simple example: a counting experiment

- Lets revisit the Poisson counting experiment

$$P(n \mid s+b) = \frac{(s+b)^n}{n!} e^{-(s+b)}$$

- In yesterdays limit setting exercise we have assumed b is known and fixed → L(s) has no nuisance parameters
  - b is not a parameter in the statistical sense, as it is fixed
- Now lets assume b is not perfectly known → introduce it as nuisance parameter. Probability function P(N|s+b) is the same, but likelihood is now L(s,**b**)
- Note that in this case we cannot constrain b and s from the same data: increasing b by one and decreasing s by one results in the same likelihood: $\mathrm{cov}[\hat{s},\hat{b}] = -1$

Introducing a subsidiary measurement

- Suppose we can measure the background rate in a control region:

Probability model for signal region

$$P(n \mid s+b) = \frac{(s+b)^n}{n!} e^{-(s+b)}$$

Probability model for control region

$$P(n_{ctl} \mid \tau b) = \frac{(\tau b)^n}{n_{ctl}!} e^{-\tau b}$$

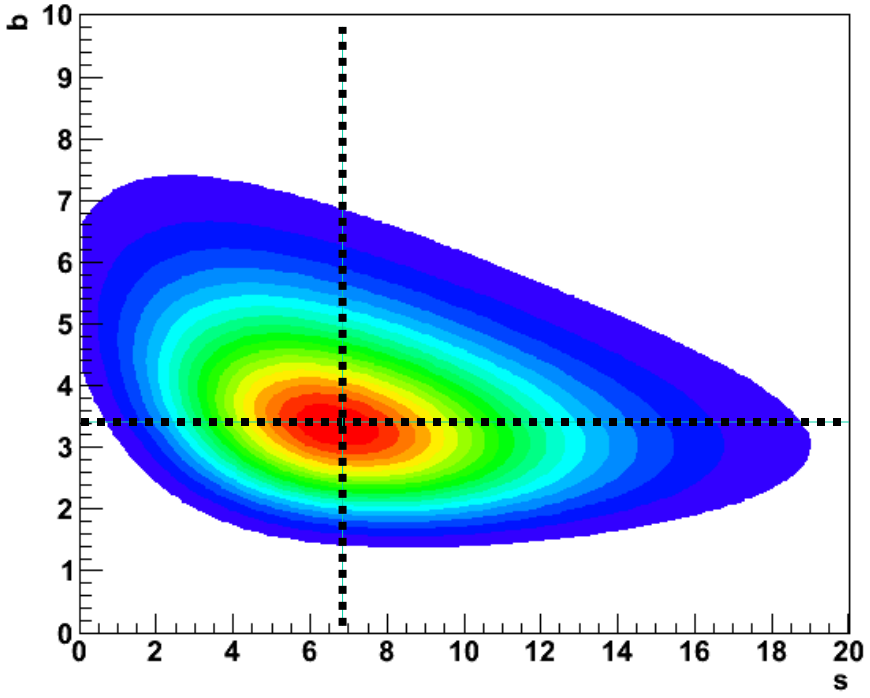
- If signal region and control region have the same size then τ=1, otherwise the τ factor scales b such that it respresents the estimate of b for the signal region
  - The scale factor τ is assumed to be known and fixed
- Now we have two measurement *N* and N_{ctl}, can write joint model $P(n,n_{ctl}|s,b)$

$$P(n \mid s+b) = \frac{(s+b)^n}{n!} e^{-(s+b)} \cdot \frac{(\tau b)^n}{n_{ctl}!} e^{-\tau b}$$

Joint model can measure both s and b

- Model: Poisson(N|s+b)Poisson(N_{ctl}|τ·b), τ=3 (exact)
- Visualization of Likelihood for N=10,N_{ctl}=10

  L(s,b) = Poisson(10|s+b)Poisson(10|3·b)



Note on the Poisson counting model

- Model: Poisson(N|s+b)Poisson(N_{ctl}|τ·b), τ=3 is the simplest model for discovery and limit setting for new physics
- In the (HEP) statistics literature it is called the "on/off" problem
- Hypothetical case for "Exotic" discovery
  - Define fancy selection sensitive to Exotic physics (of arbitrary complexity). Calculate predicted event yields in data and simulation:
  - Simulation for SM – Predicts 3 events
  - Simulation for SM+Exo– Predicts 3+6 events → 9 events in total
  - Observed event count in data: 8 events
- How do you conclude (or not) that you've discovered something?
  - You expect 9 events (with NP), you see 8, looks promising
- Can solve this with the on/off model.
- However: estimate of background and its uncertainty is crucial.
  - For realistic scenario unlikely to be a simple counting estimate, but also need to account for effect of detector/theory systematics

Generalizing subsidiary measurements

- Q: How to formulate a likelihood if b is estimated from theory
  - Estimate for b is 5 ± 2 from simulation, the uncertainty originates from a theory cross-section uncertainty

- A: Can formulate a subsidiary measurement $\tilde{b}=5 \pm 2$ that results in a constraint on nuisance parameter b

$$L(N,\tilde{b}\,|\,s,b) = Poisson(N\,|\,s+b)Gaussian(\tilde{b}\,|\,b,\sigma(\tilde{b})=2)$$

- Given an observation of N=10 from the data our full observation (including information from the subsidiary measurement) is now

$$(N,\tilde{b}) = (10,2)$$

- Goal achieved: theory syst. uncertainty included in likelihood as nuisance parameters
- Note similarity in formulation to original on/off problem

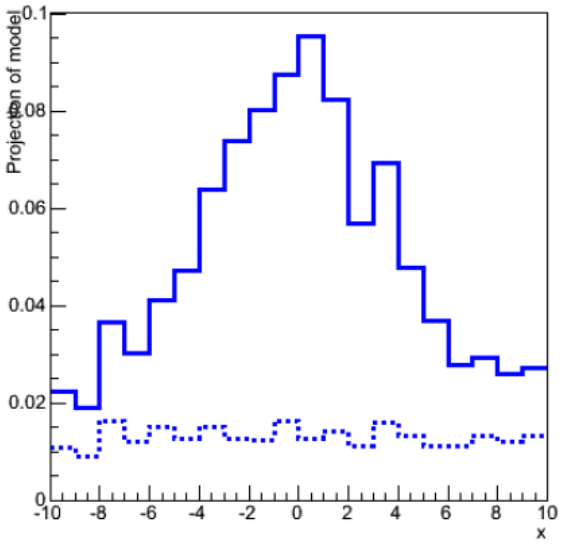
$$L(N,Nctl\,|\,s,b) = Poisson(N\,|\,s+b)Poisson(Nctl\,|\,\tau\cdot b)$$

Nuisance parameters affecting distributions

- Goal is to model all systematic uncertainties as nuisance parameters. Theory uncertainty in counting was easy
- How to model systematic uncertainties that affect distributions?
- Example: How model effect of a Jet-Energy-Scale uncertainty affecting an invariant mass distribution?
- Preamble:
  - Most LHC physics analysis using shape information construct complex observables sensitive to the parameter-of-interest for which no simple analytical model exist to describe these
  - For the signal component this is sometimes still possible, but background shapes are generally very difficult
  - Solution: Represent signal and background distributions with histograms obtained from the full physics/detector simulation chain of experiments.

Template models

- Probability models based on such histograms are commonly called 'template models'
  - Example template model with a signal and background component



- Advantage: no effort needed to construct analytical shape to describe signal and background components
- Issue 1: If simulation statistics are low compare to data this introduces a (systematic) uncertainty on its own
- Issue 2: Rigid model → Need to introduce nuisance parameters that allow sufficient flexibility to so that it can cover shape of 'true' (but unknown) model
- Note that both issues also occur in practice for parametric models that are based on physics/detector simulation samples!

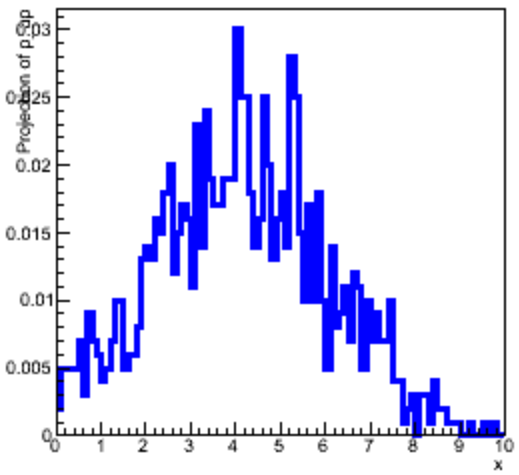
Introducing flexibility in template models

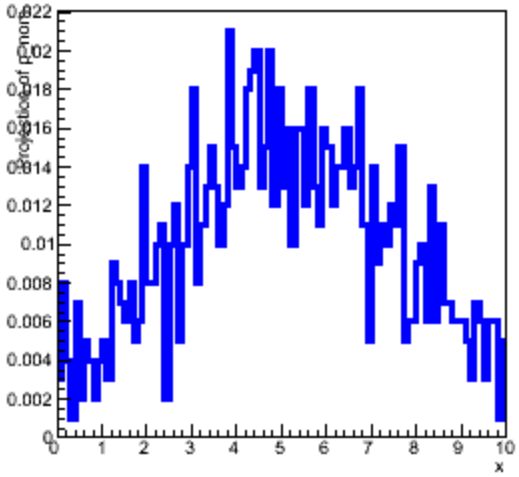
- Suppose we have three template for signal representing
  - The signal distribution at the nominal Jet-Energy Scale
  - The signal distribution at JES = nominal * 105%
  - The signal distribution at JES = nominal * 95%
- How can we make a flexible model $L(x|\theta_{JES})$ from this?

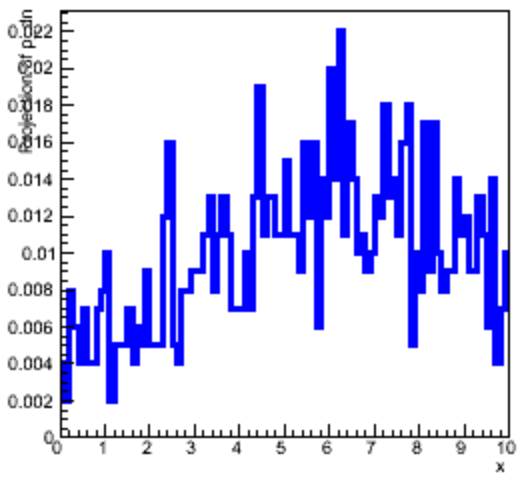
distribution for JES at -1σ

nominal distribution

distribution for JES at +1σ



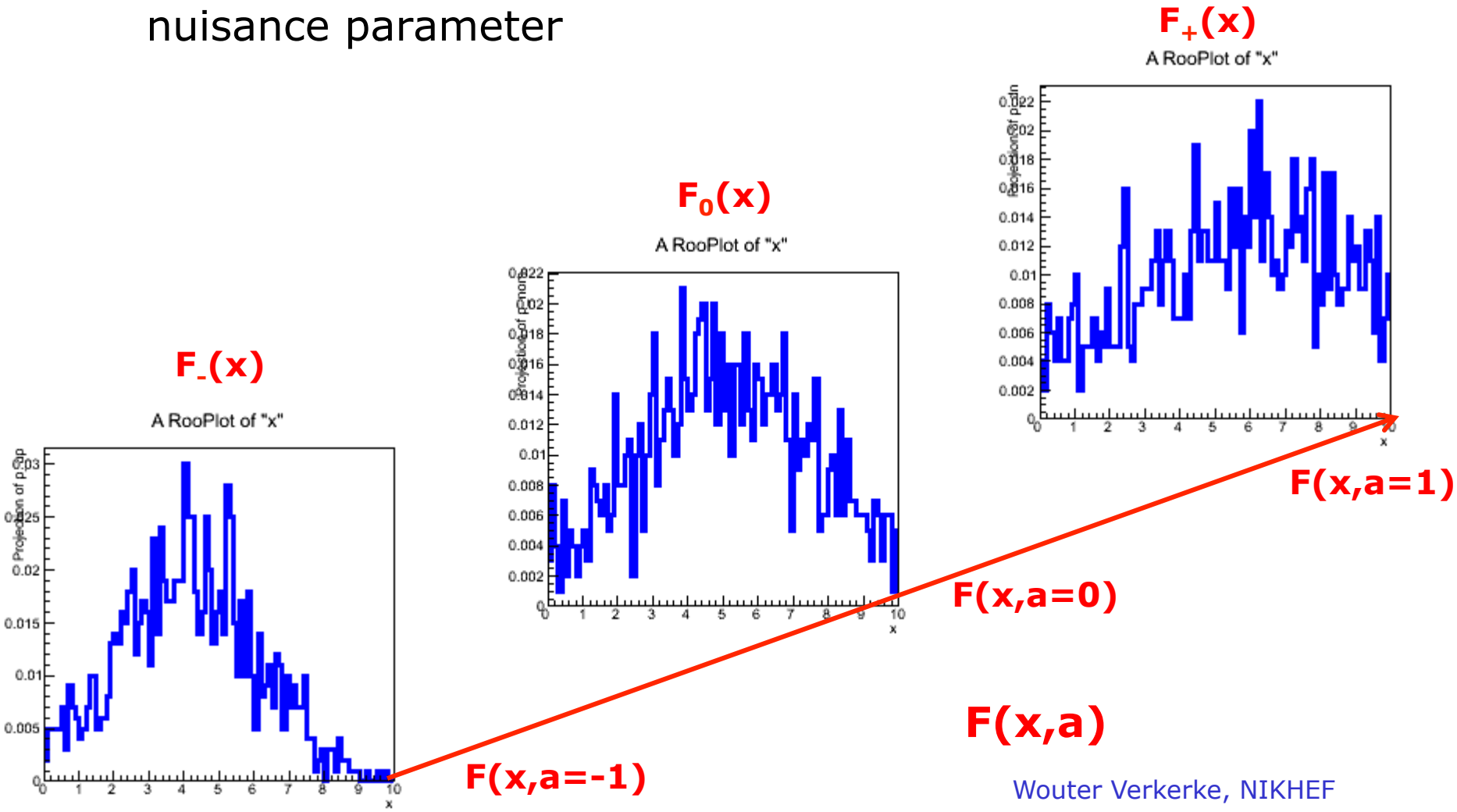




$F_-(x)$ $F_0(x)$ $F_+(x)$

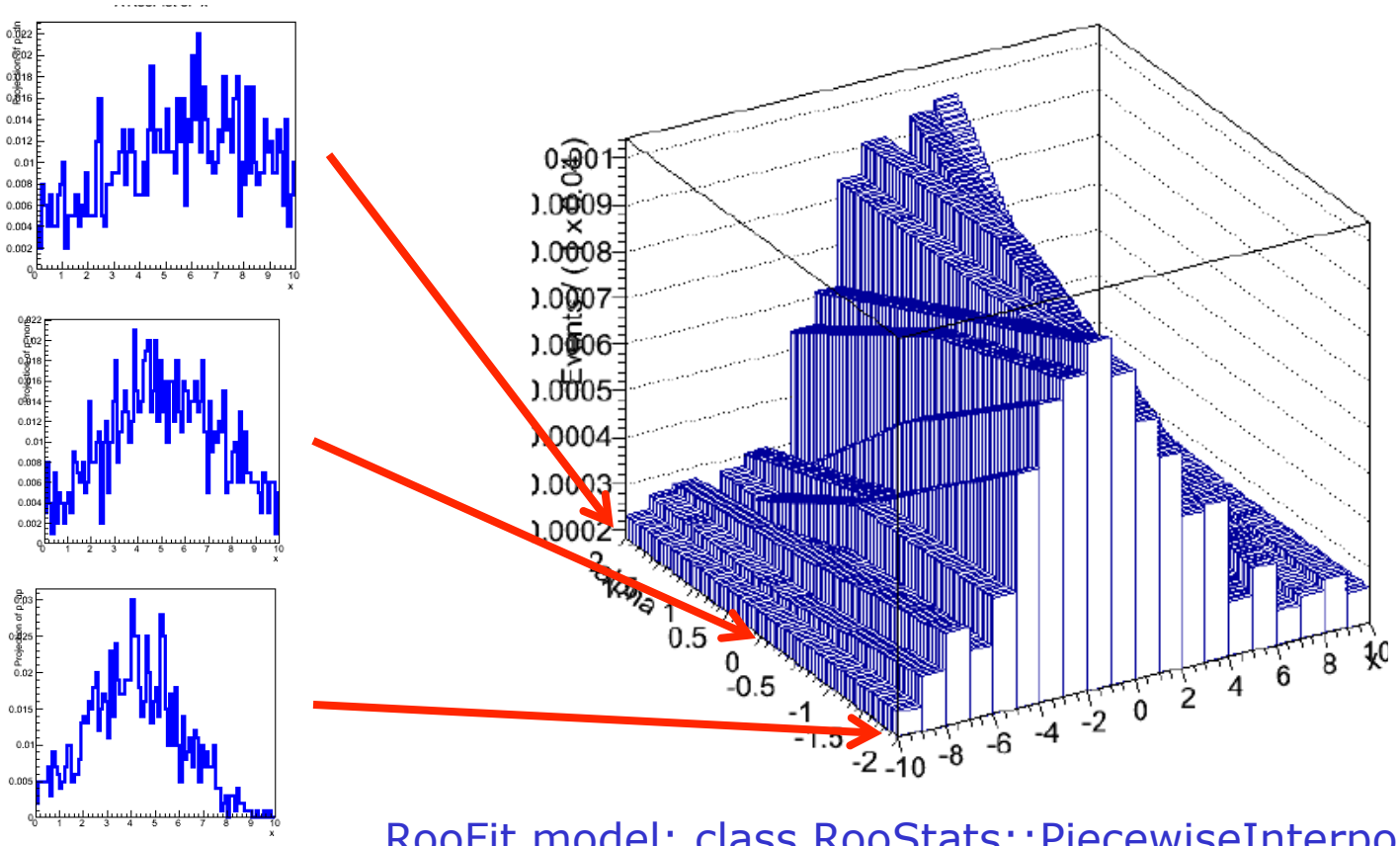
Solution: template morphing

- Construct a model that interpolates (bin-by-bin) between the histograms introducing a newly introduced nuisance parameter



Simplest version: vertical interpolation

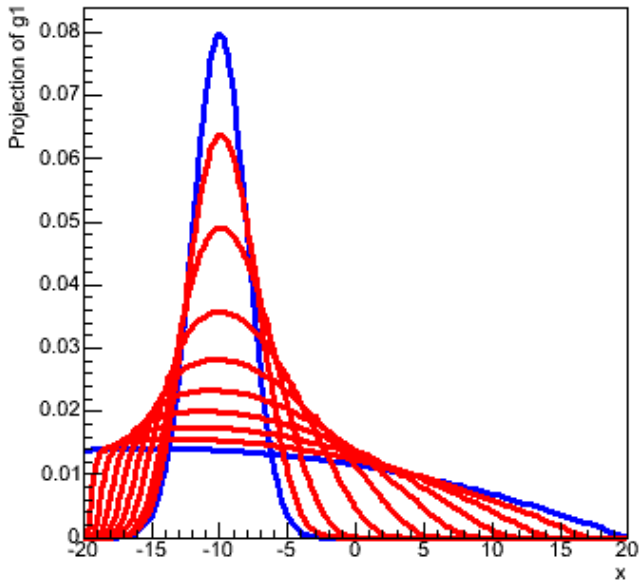
- Note template morphing is not a uniquely defined problem. Several practical solutions exist.
- The simplest (conceptually and practically) is vertical interpolation bin-by-bin

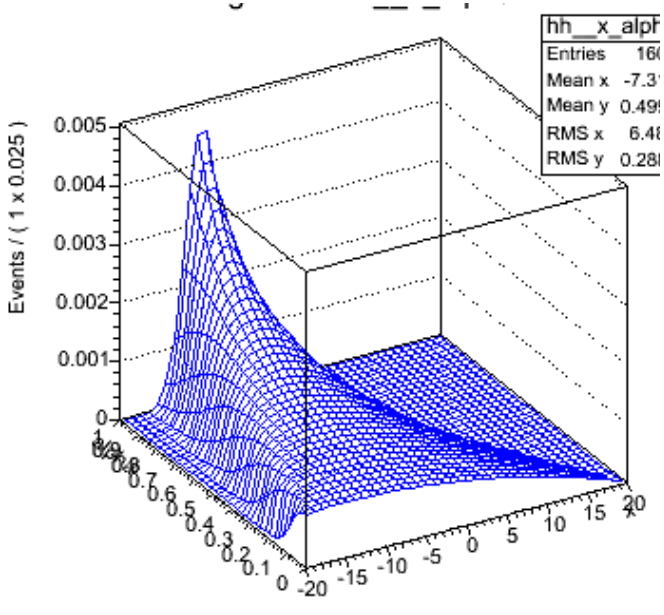


RooFit model: class RooStats::PiecewiseInterpolation

Other morphing variations

- 'Horizontal morphing': interpolation cumulative distributions rather than pdfs $F(x) = \int_0^x f(x')dx'$
  - More suitable for shifting (narrow) distributions
  - Computationally expensive

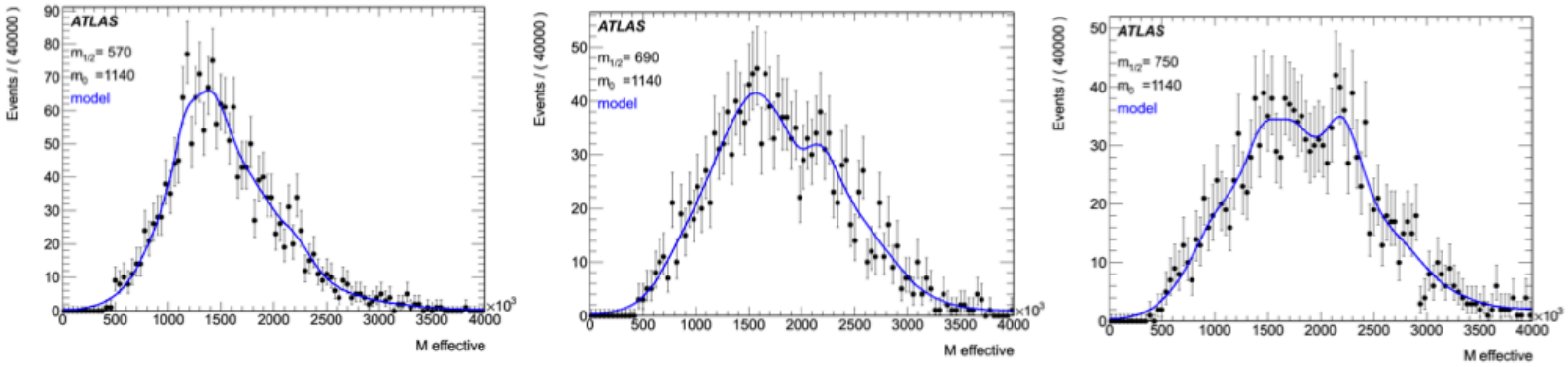




RooFit model: class RooLinearMorph

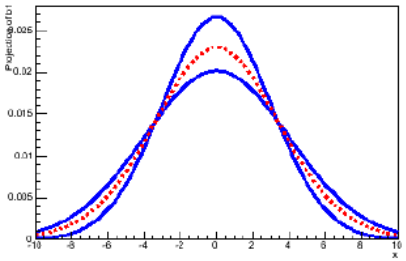
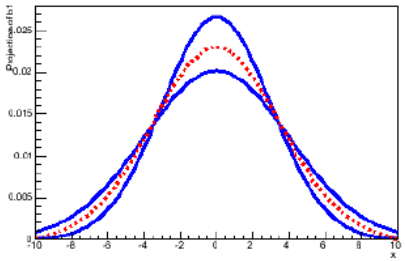
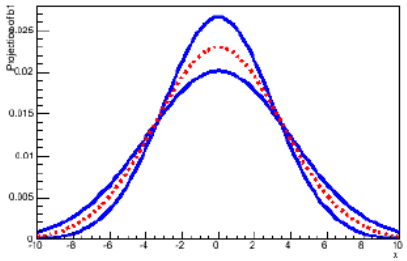
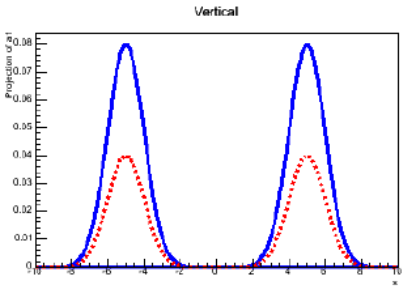
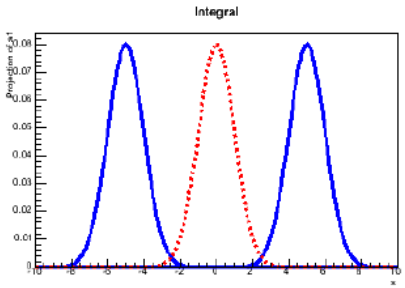
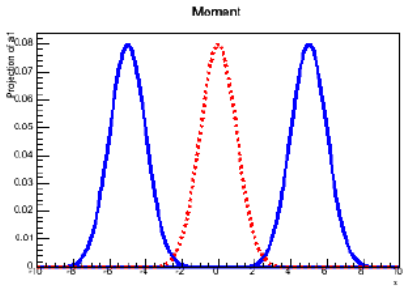
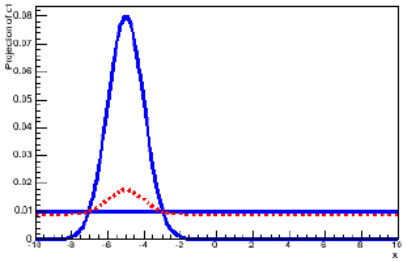
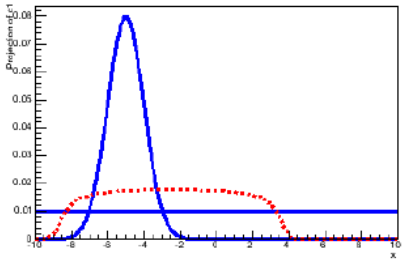
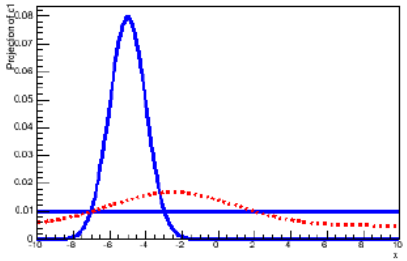
Other morphing variations

- 'Moment morphing': transform models with linear transformation in observables that adjust 1st and 2nd moment



RooFit model: class RooMomentMorph

Comparison of morphing algorithms

	Vertical Morphing	Horizontal Morphing	Moment Morphing
Gaussian varying width			
Gaussian varying mean			
Gaussian to Uniform (this is conceptually undefined!)			

Constructing a joint model for a shape systematic

- Remember joint model for theory systematic

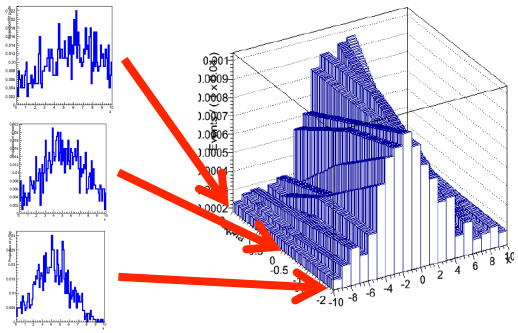
$$L(N,\tilde{y}\,|\,s,b) = Poisson(N\,|\,s+b)Gaussian(\tilde{b}\,|\,b,\sigma(\tilde{b})=2)$$

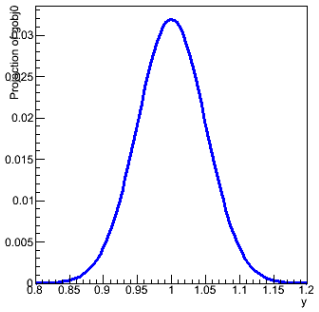
- Now we can put together a similar form for a shape systematic

Measured JES

JES NP

$$L(\vec{x},\tilde{\theta}_{JES}\,|\,s,\theta_{JES}) = Morph(\vec{x}\,|\,\theta_{JES})\cdot Gauss(\tilde{\theta}_{JES}\,|\,\theta_{JES},\sigma(\tilde{\theta}_{JES}))$$

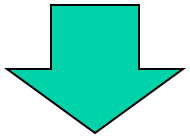




Nuisance parameters in pull form

- Nuisance parameters in morphing are conventionally formulated in pull form

$$L(\vec{x},\tilde{y}\,|\,s,\theta_{JES}) = Morph(\vec{x}\,|\,\theta_{JES})\cdot Gauss(\tilde{y}\,|\,\theta_{JES},\sigma(\theta_{JES}))$$



$$L(\vec{x},\tilde{y}\,|\,s,\alpha_{JES}) = Morph(\vec{x}\,|\,\alpha_{JES})\cdot Gauss(\tilde{\alpha}_{JES}=0\,|\,\frac{(\alpha_{JES}-\hat{\theta}_{JES})}{\sigma(\hat{\theta}_{JES})},1)$$

- So that
  - The subsidiary measurement is always zero
  - α=0 corresponds to the nominal configuration (θ=θhat)
  - α=+1 corresponds to 'one sigma up' (θ=θhat+σ(θ))
  - α=-1 corresponds to 'one sigma down' (θ=θhat-σ(θ))

Symmetry between subsidiary and mean measurement

- Note again that main measurement and subsidiary measurement are treated symmetrically in the likelihood

$$L(\vec{x},\tilde{y}\,|\,s,\alpha_{JES}) = Morph(\vec{x}\,|\,\alpha_{JES})\cdot Gauss(\tilde{y}=0\,|\,\frac{(\alpha_{JES}-\hat{\theta}_{JES})}{\sigma(\hat{\theta}_{JES})},1)$$

- This also means that both main measurement and the main measurement can (in principle) constrain α_{JES}!
  - I.e. with sufficient statistics the main measurement can (in principle) provide a strongly constraint on a systematic uncertainty than subsidiary measurement
  - This is in principle OK, but you should be extra careful examining the precise assumption made in the main model to confirm that this improved inference is legitimate

Choosing distributions for subsidiary measurements

- What are good choices for the distribution of the subsidiary measurements that model your systematic uncertainties
- Have see two cases so far:
  - On/Off problem: a Poisson model

$$L(N, Nctl \mid s, b) = Poisson(N \mid s + b)\, Poisson(Nctl \mid \tau \cdot b)$$

  - Morphing shape systematic: a Gaussian model

$$L(\vec{x}, \tilde{\theta}_{JES} \mid s, \theta_{JES}) = Morph(\vec{x} \mid \theta_{JES}) \cdot Gauss(\tilde{\theta}_{JES} \mid \theta_{JES}, \sigma(\tilde{\theta}_{JES}))$$

- What are good choices?
  - If the subsidiary measurement describes (the equivalent) of a counting experiment, Poisson is good choice
  - If subsidiary measurement was a high-statistics measurement, or a addition of many systematic effects in quadrature, a Gaussian can be a good choice (Central Limit Theorem)

Choosing distributions for subsidiary measurements

- For theory uncertainties → Not always clear what to do
- Warning on large uncertainties from subsidiary measurements on background rates
  - If a nuisance parameter is a background rate and it has a large uncertainty, a Gaussian shape may allow for a negative background rate:
  - Example $\tilde{b} = 5 \pm 3$
  - 68% Gaussian interval only contains values with b>0, but 95% Gaussian interval also includes b=1
  - Consider to use a Poisson shape (always positive), or alternatively a logNormal distribution

Gamma and logNormal distributions

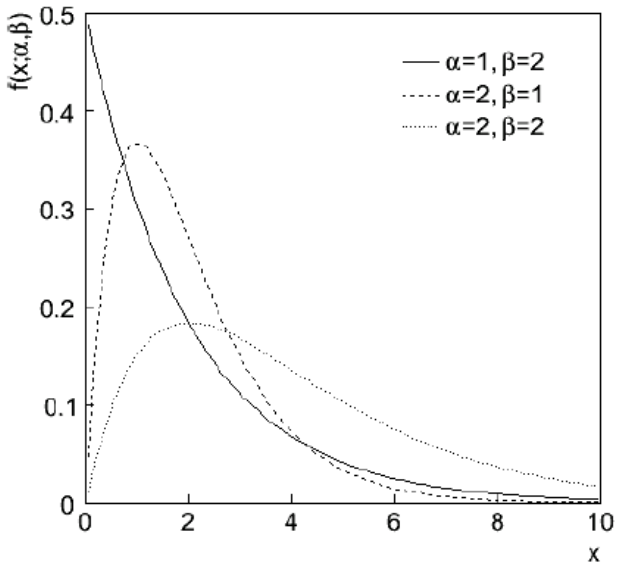
Gamma distribution

=distribution of μ resulting from a Poisson measurement L(N|μ)

$$f(x;\alpha,\beta) = \frac{1}{\Gamma(\alpha)\beta^{\alpha}} x^{\alpha-1} e^{-x/\beta}$$

$$E[x] = \alpha\beta$$

$$V[x] = \alpha\beta^2$$

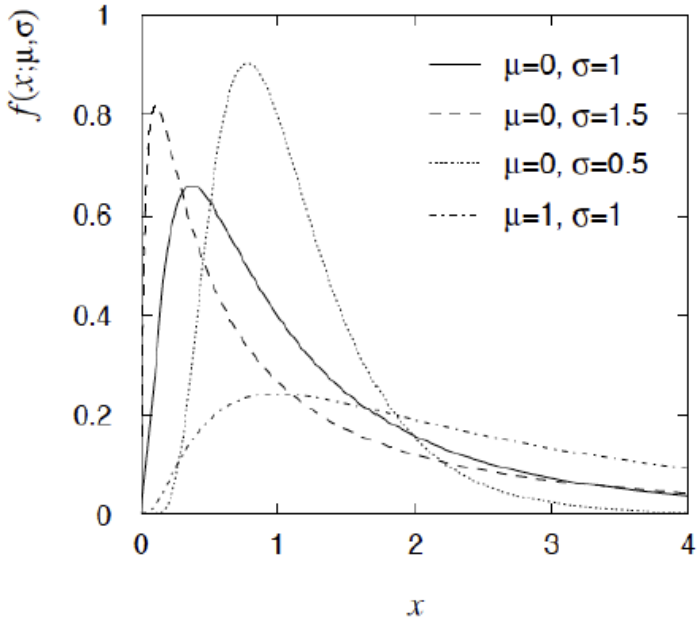


logNormal distribution

$$f(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{1}{x} \exp\left(\frac{-(\log x - \mu)^2}{2\sigma^2}\right)$$

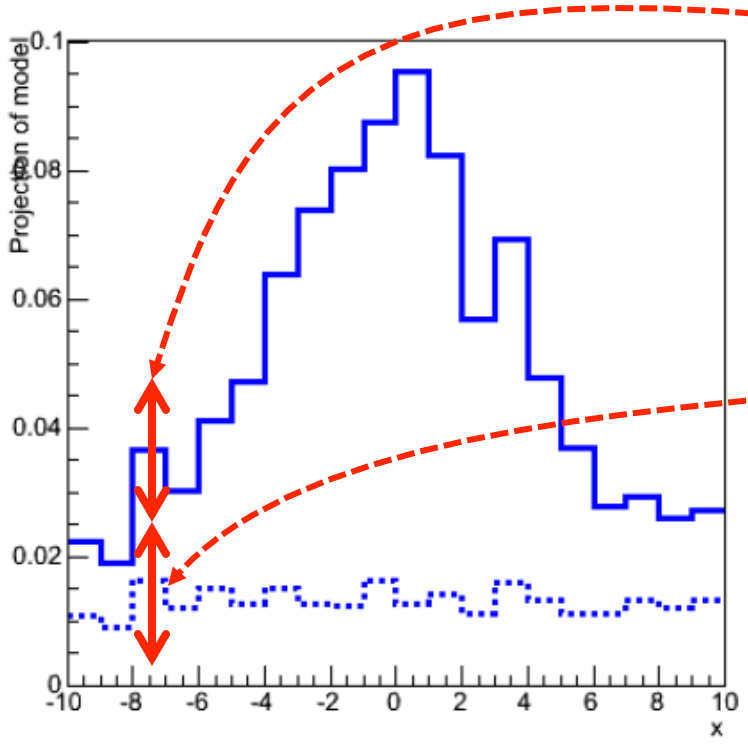
$$E[x] = \exp(\mu + \tfrac{1}{2}\sigma^2)$$

$$V[x] = \exp(2\mu + \sigma^2)[\exp(\sigma^2) - 1]$$



Nuisance parameters for template statistics?

- When template statistics are comparable (or smaller) to that of the observed data, additional source of uncertainty → Introduce these as nuisance parameters
- In principle: for every bin of every template one nuisance parameter. Subsidiary measurement for each



$$L(\tilde{y}_{0,s},\ldots,\tilde{y}_{n,s}) = \prod_{i=0}^{n} P_s(\tilde{y}_{i,s} \mid \theta_{i,s}) P_b(\tilde{y}_{i,b} \mid \theta_{i,b})$$

Signal simulation event count for bin i

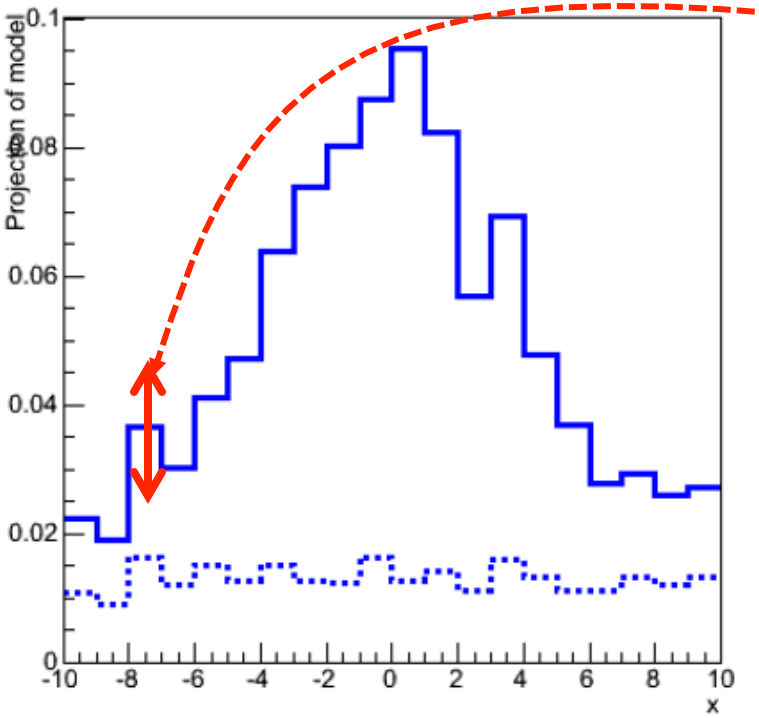
Background simulation event count for bin i

Nuisance parameters for template statistics?

- With 10-100 bins per template and multiple templates, could end up with hundreds of nuisance parameters for template statistics
  → Numeric implementation of full statistical treatment becomes intractable
- Solution 1 – 'Beeston Barlow'.
  - Beeston and Barlow have shown that minimization of likelihood for template statistics can be factorized for each bin so that they can be performed recursively.
  - Instead minimizing likelihood w.r.t. $N_{templ}*N_{bin} + N_{other}$ parameters, do N_{bin} minimizations of N_{templ} parameters, plus one minimization of N_{other}
  - NB: This procedure is implemented in ROOT class TFractionFitter

Nuisance parameters for template statistics?

- Solution 2 – 'Beeston Barlow light'
  - Approximate Beeston-Barlow solution (which is exact) by modeling only the effect of the total statistics in each bin



$$L(\tilde{y}_0,\ldots,\tilde{y}_n) = \prod_{i=0}^{n} P(\tilde{y}_i \mid \theta_i)$$

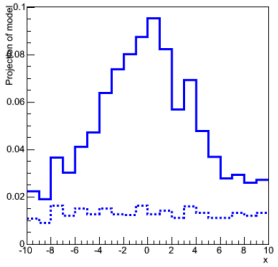
Signal+Backrgound simulation event count for bin i

  - Requires N_{bin} minimization of one parameter
    → Can be solved analytically

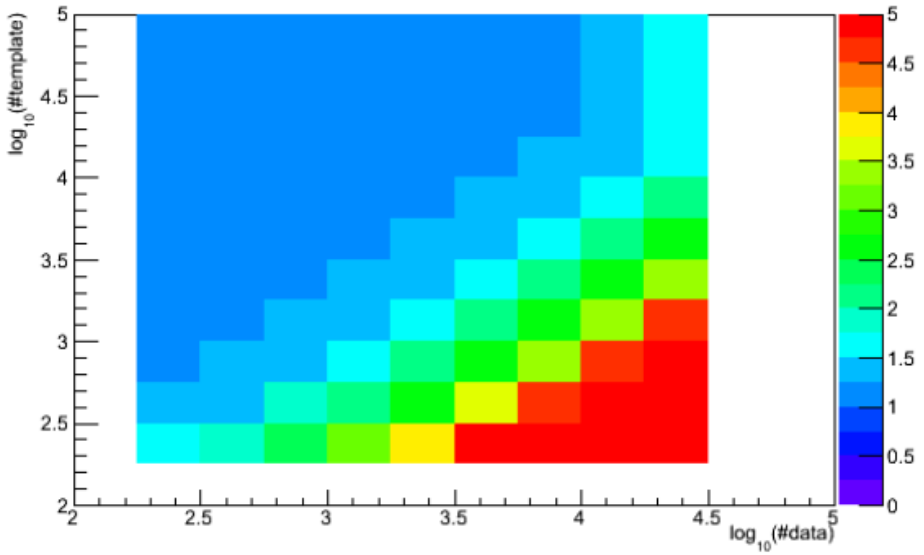
$$\frac{-\ln \mathcal{L}_i}{\partial \beta_i} = 0 \quad \Rightarrow \quad \beta_i^2 + (\mu_i \sigma_i^2 - 1)\beta_i - n_i \sigma_i^2 = 0$$

The effect of template statistics

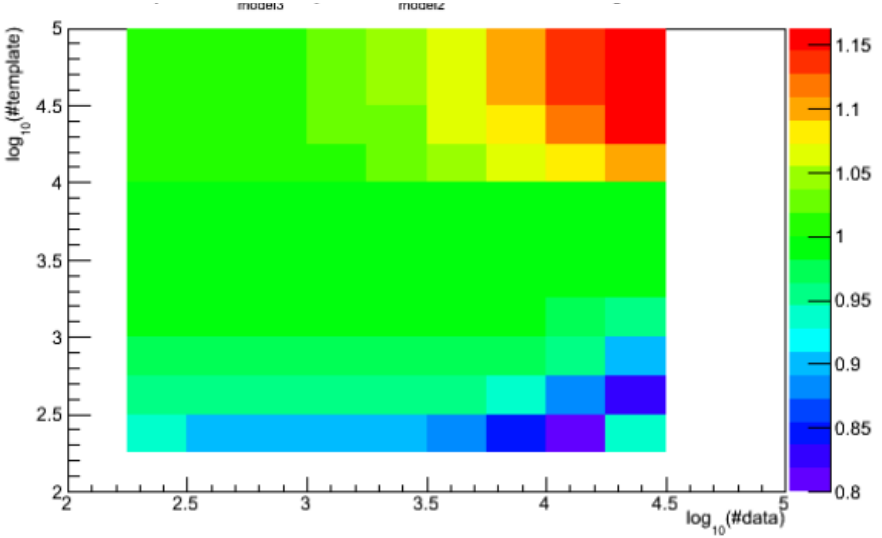
- Roughly speaking the effect of template statistics becomes important when $N_{templ} < 10x\ N_{data}$



$$\log(\sigma / \sigma_{BB})$$



$$\log(\sigma_{BB-light} / \sigma_{BB})$$



Tools to build parameterized likelihood models

- Note that RooStats comes with 'histfactory' a tool to construct parameterized template models from an XML specification and a set of input histograms

Example Channel

- config/example_channel.xml

zween — 112×33

ssh | bash | bash | ...

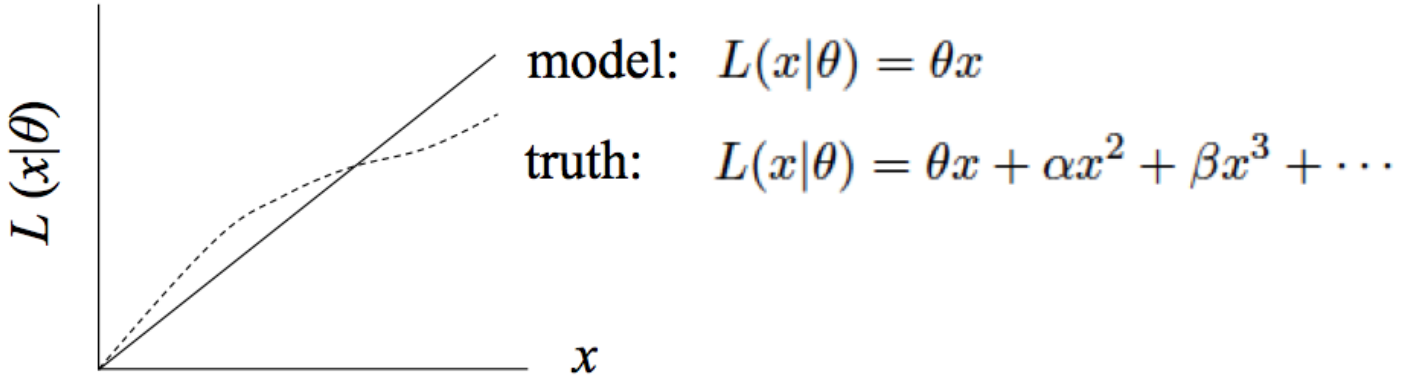
```
<!--
    Single channel configuration exampple.
    The top level configuration XML is example.xml

    NormalizedByTheory should be "True" (not "TRUE" or "true") for all non-data-driven backgrounds.

    If you comment or remove the <Data> tag then it will use the expected data.

    Histogram inputs should be in pb and in top-level xml the lumi should be in 1/pb
    (The important thing is that they match... fb and 1/fb is also ok)

    Note: Config.dtd needs to be accessible.  It can be found in ROOT release area.
    The file system path is relative to location of this XML file, not the executable.
-->

<!DOCTYPE Channel  SYSTEM 'HistFactorySchema.dtd'>

  <Channel Name="channel1" InputFile="./data/example.root" HistoName="">
    <Data HistoName="data" HistoPath="" />
    <Sample Name="signal" HistoPath="" HistoName="signal">
      <OverallSys Name="syst1" High="1.05" Low="0.95"/>
      <NormFactor Name="SigXsecOverSM" Val="1" Low="0." High="3." Const="True" />
    </Sample>
    <Sample Name="background1" HistoPath="" NormalizeByTheory="True" HistoName="background1">
      <OverallSys Name="syst2" Low="0.95" High="1.05"/>
    </Sample>
    <Sample Name="background2" HistoPath="" NormalizeByTheory="True" HistoName="background2">
      <OverallSys Name="syst3" Low="0.95" High="1.05"/>
      <!-- <HistoSys Name="syst4" HistoPathHigh="" HistoPathLow="histForSyst4"/>-->
    </Sample>
  </Channel>
~
"example_channel.xml" 31L, 1424C                                    12,5          All
```

file we just looked at

names of the histograms from previous page

What are nuisance parameters?

- In general, our mode of the data is not perfect



- Can improve modeling by including additional adjustable parameters $L(x|\theta) \rightarrow L(x|\theta, \nu)$
- Nuisance parameters can be used to model systematic uncertainties: **Some point in the parameter space of the enlarged model should be "true"**
- Presence of nuisance parameters decreases the sensitivity of the analysis of the parameter(s) of interest

Summary on nuisance parameters

- Systematic uncertainties can only be properly accounted for in hypothesis testing if their effect is modeling in the likelihood
- Nuisance parameters introduce flexibility in models so that they can (in theory) assume the true distribution for some value of these parameters
- All systematic uncertainties can (in principle) be represented by nuisance parameters
  - Rate systematics can be introduced as a multiplicative factor
  - Shape systematics can be implemented with template morphing techniques
  - MC statistics systematics can be implemented with the Beeston-Barlow (light) techniques
- For many nuisance parameters it is common to introduce a *subsidiary measurement* that constrains the systematic uncertainty according to its specification

Treatment of nuisance parameters in limits

Hypothesis testing with nuisance parameters

- Yesterday we covered frequentist hypothesis testing and interval calculation using likelihood ratios based on a likelihood with a single parameter (of interest) L(μ)
  - Result is p-value on hypothesis with given μ value, or
  - Result is a confidence interval [μ-,μ+] with values of μ for which p-value is at or above a certain level (the confidence level)
- How do you do this with a likelihood L(μ,θ) where θ is a nuisance parameter?
  - With a test statistics q_μ, we calculate p-value for hypothesis θ as

$$p_\mu = \int_{q_{\mu,obs}}^{\infty} f(q_\mu \mid \mu, \theta) dq_\mu$$

- But what values of μ do we use for $f(q_\mu|\mu,\theta)$? Fundamentally, we want to reject θ only if p<α for ***all*** θ
  → Exact confidence interval

Hypothesis testing with nuisance parameters

- The goal is that the parameter of interest should be covered at the stated confidence **for every value of the nuisance parameter**
- if there is *any value* of the nuisance parameter which makes the data consistent with the parameter of interest, that parameter point should be considered:
  - e.g. don't claim discovery if any background scenario is compatible with data
- But: technically very challenging and significant problems with over-coverage
  - Example: how broadly should 'any background scenario' be defined? Should we include background scenarios that are clearly incompatible with the observed data?

Example of over-coverage

- The 1958 thought expt of David R. Cox focused the issue:
  - Your procedure for weighing an object consists of flipping a coin to decide whether to use a weighing machine with a 10% error or one with a 1% error; and then measuring the weight.
- Then "surely" the error you quote for your measurement should reflect which weighing machine you actually used, and not the average error of the "whole space" of all measurements!
- But this is not how the classical frequentist confidence interval works!
  - Suppose weight=100, coin='1% error' Can you exclude weight=90 at 95% C.L?
  - No: because for 'coin=10% error' weight=90 cannot be excluded at 95% C.L.
- Solution: conditioning on observed data will make result more relevant (at expense of exact frequentist coverage)
  - Restricting whole space of probabilities to 'coin=1% error' only if that is observed allows to exclude weight=90 at 95% C.L.

The profile likelihood construct as compromise

- For LHC the following prescription is used:

  Given L(μ,θ)

  perform hypothesis test for each value of μ,

  assuming values of nuisance parameter(s) θ that best fit the data under the hypothesis μ

- Introduce the following notation

$$\hat{\hat{\theta}}(\mu)$$

M.L. estimate of θ for a *given* value of μ (i.e. a conditional ML estimate)

- The resulting confidence interval will have exact coverage for the points $(\mu, \hat{\hat{\theta}}(\mu))$
  - Elsewhere it may overcover or undercover (but this can be checked)

The profile likelihood ratio

- With this prescription we can construct the profile likelihood ratio as test statistic

Likelihood for given μ

Maximum Likelihood for given μ

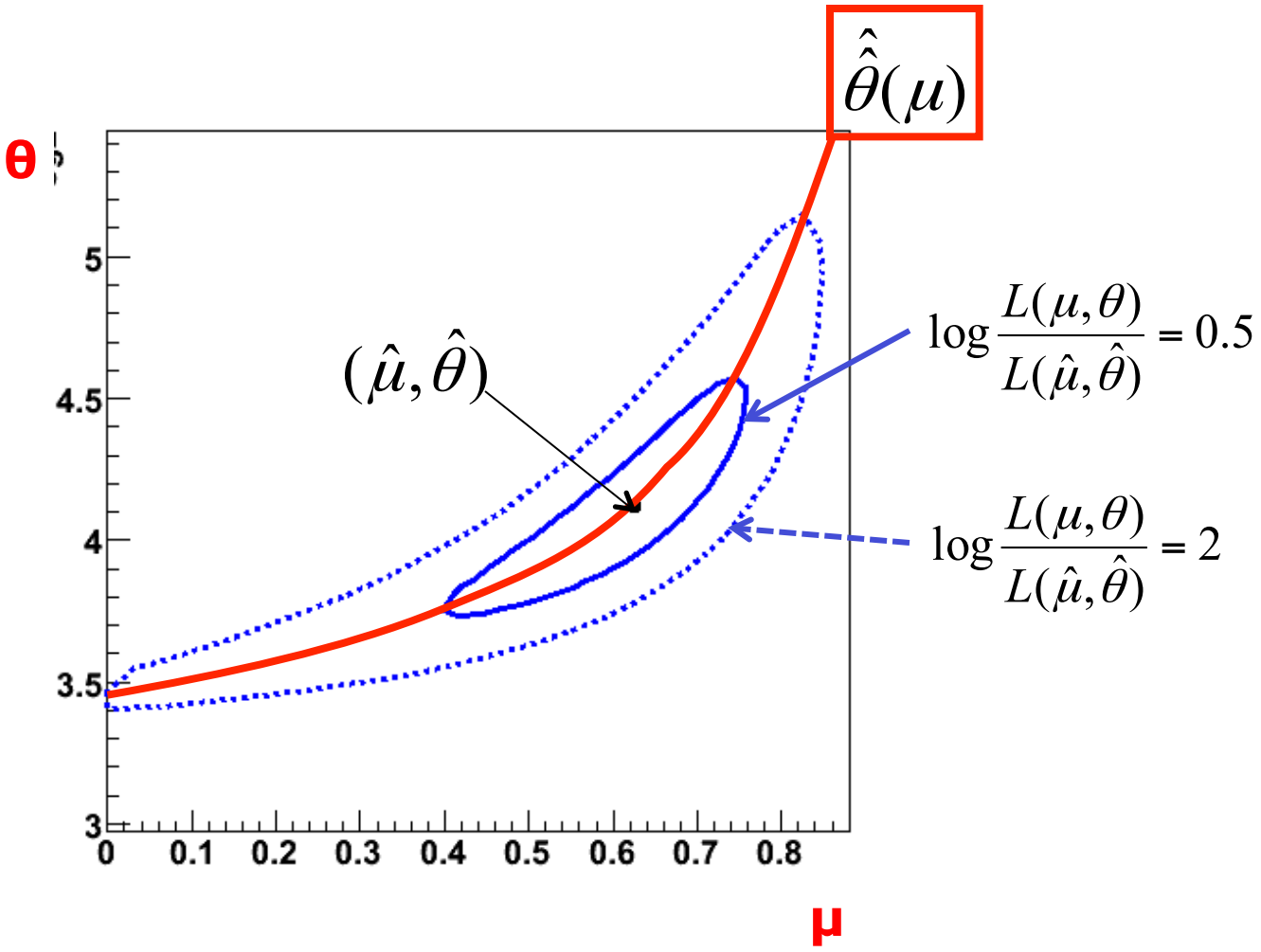
$$\lambda(\mu) = \frac{L(\mu)}{L(\hat{\mu})} \Rightarrow \lambda(\mu) = \frac{L(\mu, \hat{\hat{\theta}}(\mu))}{L(\hat{\mu}, \hat{\theta})}$$

Maximum Likelihood

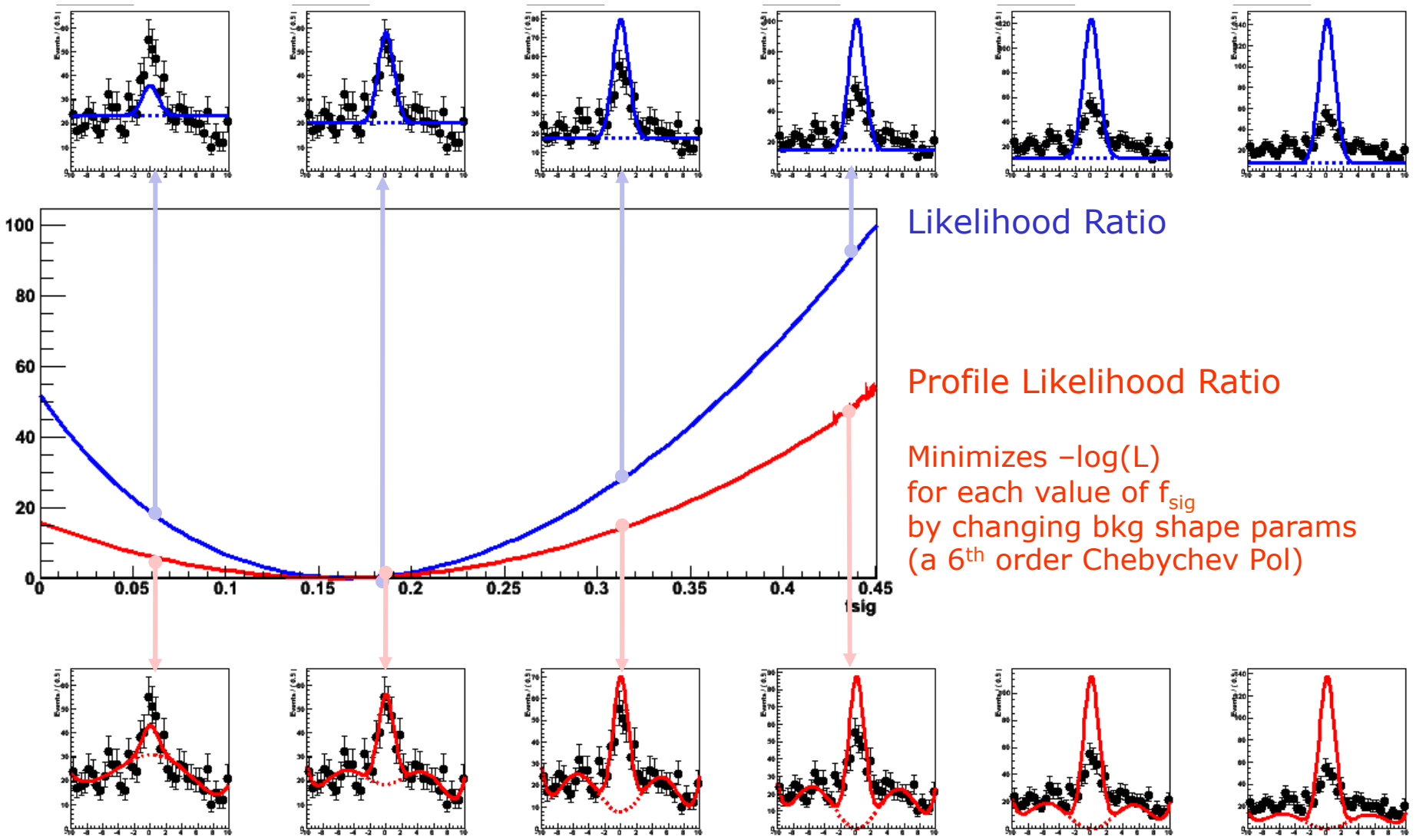
Maximum Likelihood

- NB: value profile likelihood ratio does *not* depend on θ

Profiling illustration with one nuisance parameter



Dealing with nuisance parameters in Likelihood ratio intervals



Likelihood Ratio

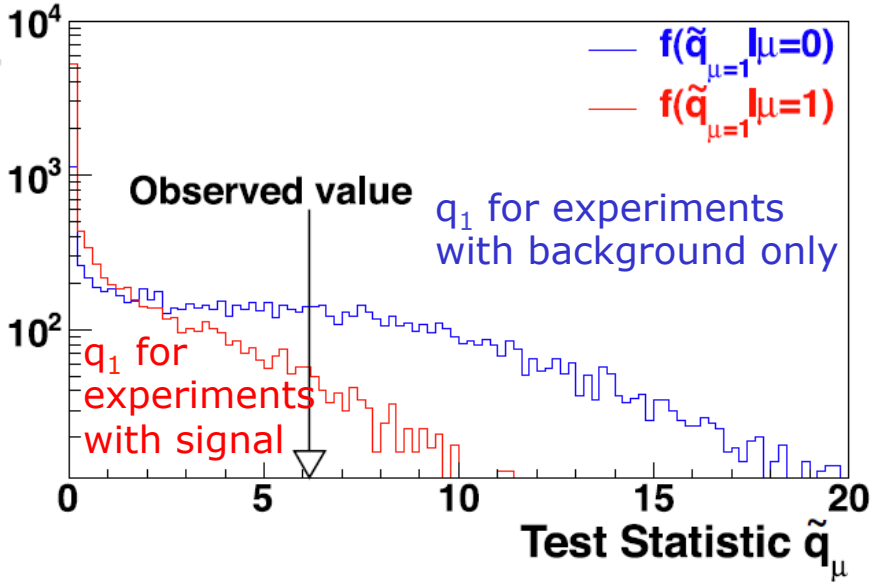
Profile Likelihood Ratio

Minimizes –log(L)
for each value of f_{sig}
by changing bkg shape params
(a 6^{th} order Chebychev Pol)

Setting limits with t_1 using *profile* likelihood ratio

- Value of t_1 from data is now the 'measurement'
- Distribution of $t_1 = f(t_1|\mu=1)$ *not* calculable
  → But can obtain distribution from toy MC approach
  → Asymptotic form exists for N→∞

$$t_1 = -2\ln\frac{L(data\,|\,\mu=1,\hat{\hat{\theta}}(\mu=1))}{L(data\,|\,\hat{\mu},\hat{\theta})}$$

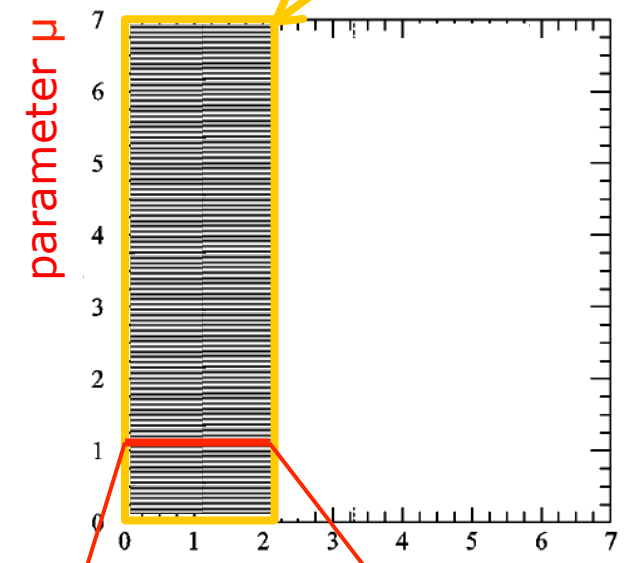


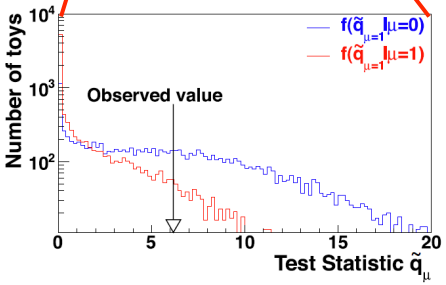
- Limit on μ (w/o CL_S) : Find t_μ for which $\int_0^{t_\mu^{obs}} f(t_\mu\,|\,\mu)=\alpha\%$
- P-value of bkg similarly obtained from t_0 $\int_{t_0^{obs}}^{\infty} f(t_0\,|\,\mu=0)=\alpha\%$

PLR Confidence interval vs MINOS

Confidence belt now range in LR

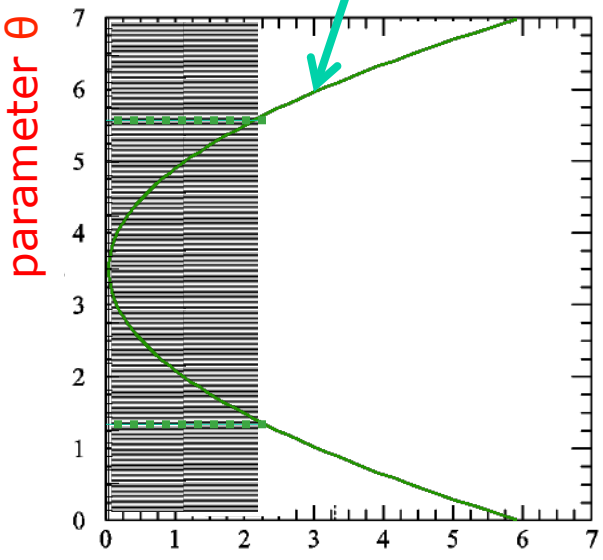
$t_\mu(x,\mu)$



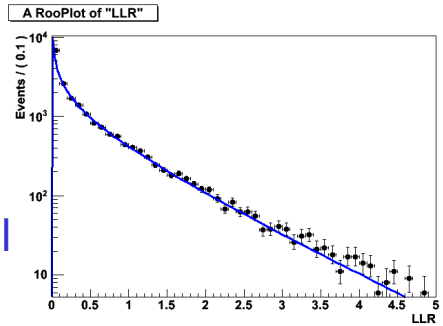


Measurement = $t_\mu(x_{obs},\mu)$ is now a function of μ

$t_\mu(x,\mu)$

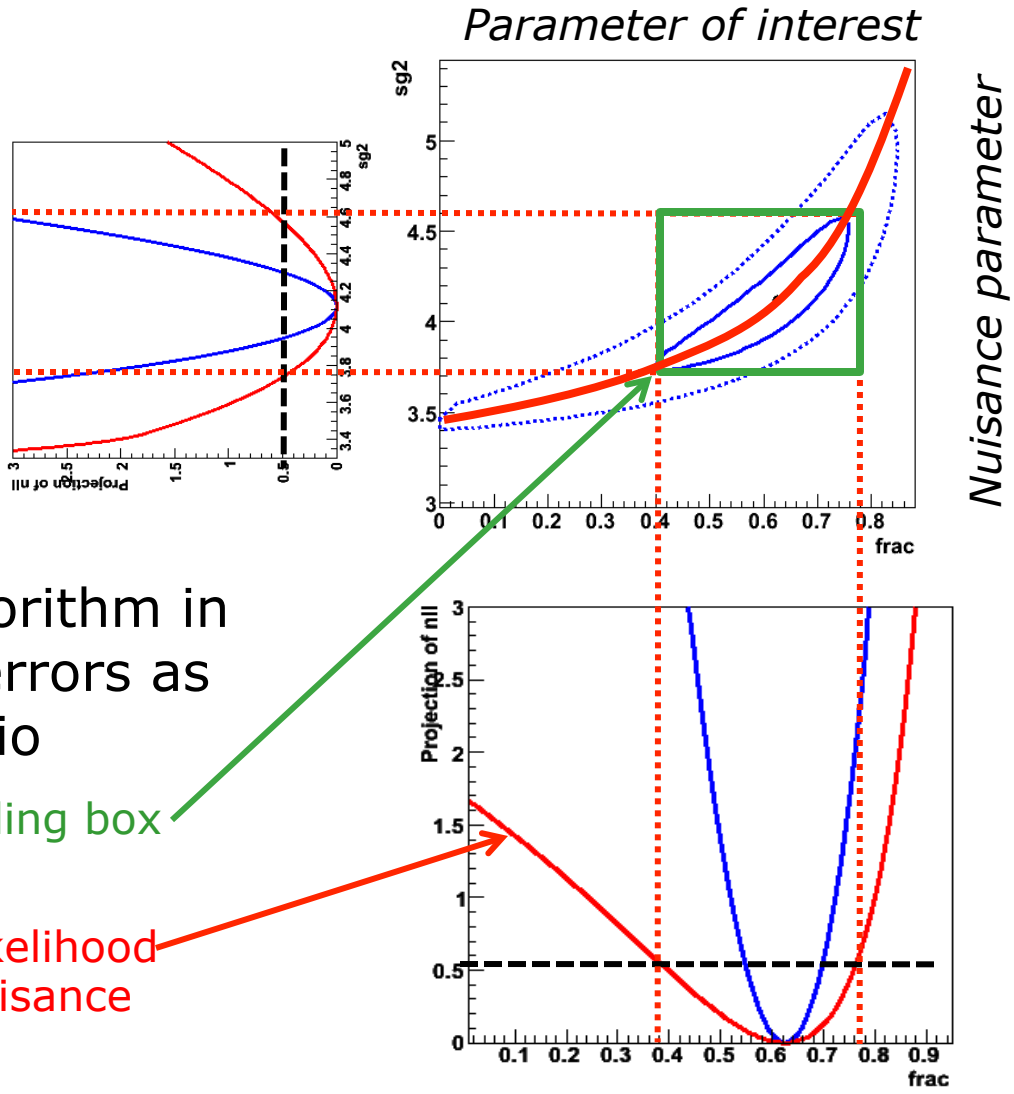


Asymptotically, distribution is identical for all μ



Asymptotically, confidence interval is profile likelihood ratio interval

Link between MINOS errors and profile likelihood



- Note that MINOS algorithm in MINUIT gives same errors as Profile Likelihood Ratio
  - MINOS errors is bounding box around λ(s) contour
  - Profile Likelihood = Likelihood minimized w.r.t. all nuisance parameters

Summary on NPs in frequentist intervals

- Exact confidence intervals are difficult with nuisance parameters
  - Interval should cover for any value of nuisance parameters
  - Technically difficult and significant overcoverage common
- LHC solution → Guaranteed coverage at *measured* values of nuisance parameters only
- Technically replace likelihood ratio with profile likelihood ratio
  - Computationally more intensive (need to minimize likelihood w.r.t all nuisance parameters for each evaluation of the test statistic), but still very tractable
- Asymptotically confidence intervals constructed with profile likelihood ratio test statistics correspond to (MINOS) likelihood ratio intervals

Bayesian treatment of nuisance parameters

Dealing with nuisance parameters in Bayesian intervals

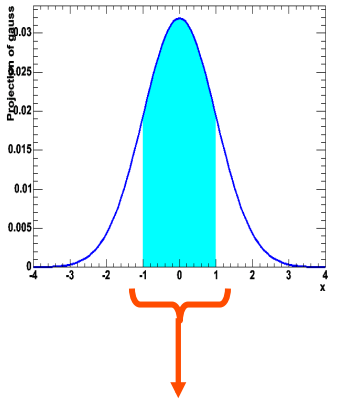
- For comparison, will briefly discuss Bayesian treatment of nuisance parameters
- Reminder: definition of Bayesian intervals

$$P(\mu) \propto L(x_0|\mu)\, \pi(\mu),$$

where:

- $P(\mu)$ = posterior pdf for μ, given the results of this experiment
- $L(x_0|\mu)$ = Likelihood $L(\mu)$ from the experiment
- $\pi(\mu)$ = prior pdf for μ,

$p(\mu|x_0)$



Area that integrates X% of posterior

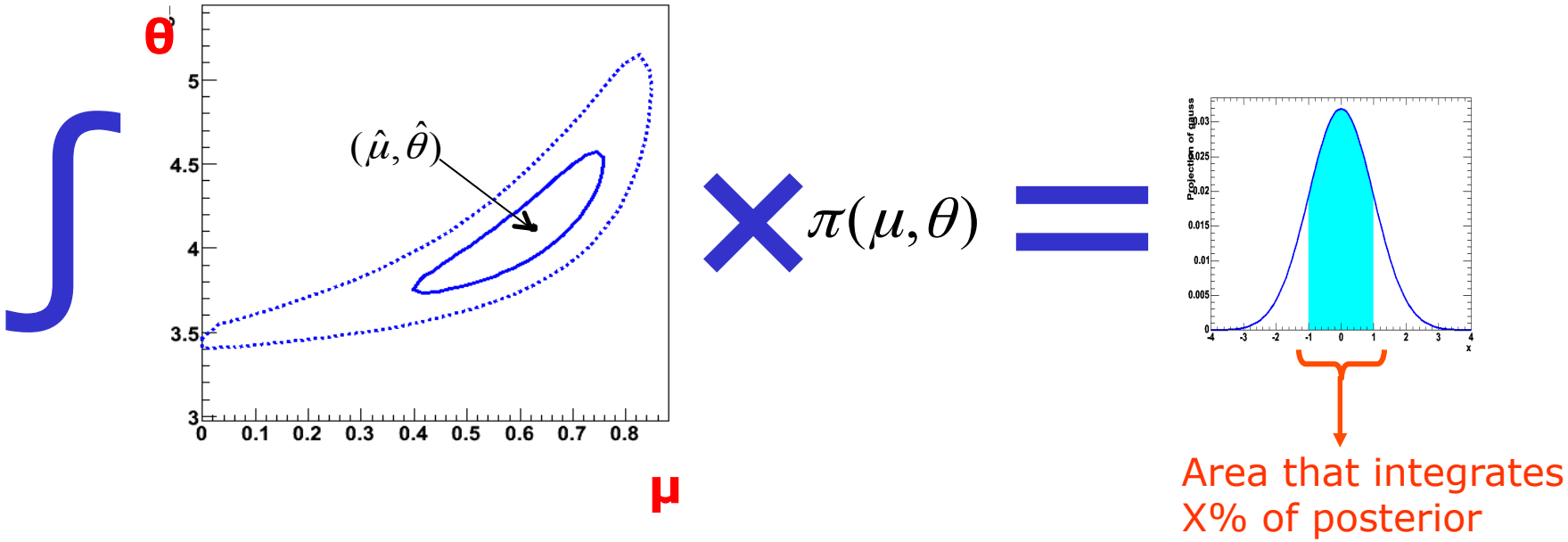
- If you have nuisance parameters $\boldsymbol{\theta}$, equation becomes
- $P(\mu,\boldsymbol{\theta}) \propto L(x_0|\mu,\boldsymbol{\theta})\, \pi(\mu)\, \pi(\boldsymbol{\theta})$

Dealing with nuisance parameters in Bayesian intervals

- Elimination of nuisance parameters in Bayesian interval: Integrate over the full subspace of all nuisance parameters;

$$P(\mu) = \int \left(L(\mu, \vec{\theta}) \pi(\mu) \pi(\vec{\theta}) \right) d\vec{\theta}$$

- You are left with posterior pdf for μ



Dealing with nuisance parameters in Bayesian intervals

- Choosing priors – generally a difficult issue. For nuisance parameter representing systematic uncertainties can exploit information from subsidiary measurement
  - Reminder: frequentist procedure treats main and subsidiary measurement on equal footing, e.g.

$$L(N, N_{ctl} \mid s, b) = Poisson(N \mid s+b)\, Poisson(N_{ctl} \mid \tau \cdot b)$$
$$L(\vec{x}, \tilde{\theta}_{JES} \mid s, \theta_{JES}) = Morph(\vec{x} \mid \theta_{JES}) \cdot Gauss(\tilde{\theta}_{JES} \mid \theta_{JES}, \sigma(\tilde{\theta}_{JES}))$$

- Equivalent Bayesian formulation

$$L(N, Nctl \mid s, b) = Poisson(N \mid s+b)$$
$$L(\vec{x}, \tilde{\theta}_{JES} \mid s, \theta_{JES}) = Morph(\vec{x} \mid \theta_{JES})$$

$$\pi(b) = Gamma(b \mid N_{CTL})$$
$$\pi(\theta_{JES}) = Gauss(\theta_{JES}, \tilde{\theta}_{JES}, \sigma(\tilde{\theta}_{JES}))$$

  - Note: The posterior of a Poisson is a Gamma

- Subsidiary measurement ***not*** treated on equal footing with main measurement: once the prior is formulated, the fact that it was a measurement is 'forgotten', i.e. data is N, not (N,Nctl)

Hybrid frequentist Bayesian approach

- Use marginalized likelihood to described the data

$$L_m(\mu) = \int \left(L(\mu, \vec{\theta}) \pi(\vec{\theta}) \right) d\vec{\theta}$$

- Then construct test statistic based on $L_m(\mu)$ to construct a confidence interval
  - It does not represent what the data distribution would be if we really repeated the experiment, since the subsidiary measurements $\theta\sim$ are not repeated
  - But it does effectively incorporate the uncertainty due to θ in the model
- Procedure sometimes referred to as 'Cousins-Highland' or 'Z_N'

How much do answers differ between methods?

A Prototype Problem

BROOKHAVEN NATIONAL LABORATORY

What is significance Z of an observation $x = 178$ events in a signal like region, if my expected background $b = 100$ with a 10% uncertainty?

- if you use the ATLAS TDR formula $Z_{5'}$=5.5
- if you use Cousins-Highland Z_N=5.0

The question seems simple enough, but it is not actually well-posed

- what do I mean by 10% background uncertainty?

Typically, we consider an auxiliary measurement y used to estimate background (Type I systematic)

- eg: a sideband counting experiment where backgrou in sideband is a factor τ bigger than in signal region

$$L_P(x, y|\mu, b) = Pois(x|\mu + b) \cdot Pois(y|\tau b).$$

These slide discuss the earlier shown problem:

$\text{Poisson}(N_{sig}|s+b) \cdot \text{Poisson}(N_{ctl}|\tau \cdot b)$

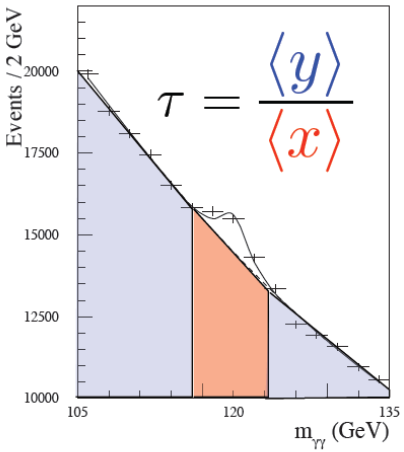
NB: This is one of the very few problems with nuisance parameters with can be *exactly* calculation

Example Sideband Measurement

BROOKHAVEN NATIONAL LABORATORY

Sideband measurement used to extrapolate / interpolate the background rate in signal-like region

For now ignore uncertainty in extrapolation.



$$L_P(x, y|\mu, b) = Pois(x|\mu + b) \cdot Pois(y|\tau b).$$

Recent comparisons results from PhyStat 2007

Comparison of Methods for Prototype Problem

BROOKHAVEN NATIONAL LABORATORY

In my contribution to PhyStat2005, I considered this problem and compared the coverage for several methods

- See Linnemann's PhyStat03 paper

Major results:

- Cousins-Highland result (Z_N) badly under-covers (only 4.2σ)!
  - rate of Type I error is 110 times higher than stated!
  - much less luminosity required

- Profile Likelihood Ratio (MINUIT/ MINOS) works great out to 5σ!

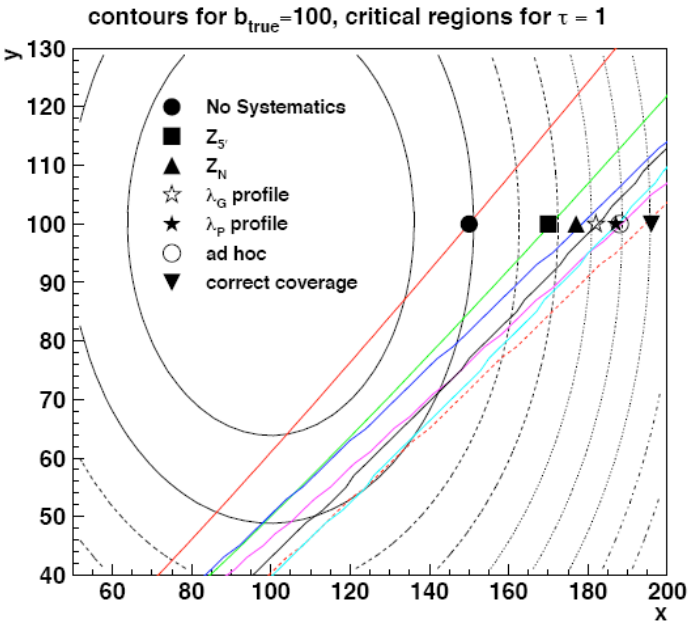


Figure 7. A comparison of the various methods critical boundary $x_{crit}(y)$ (see text). The concentric ovals represent contours of L_G from Eq. 15.

Method	L_G ($Z\sigma$)	L_P ($Z\sigma$)	$x_{crit}(y = 100)$
No Syst	3.0	3.1	150
$Z_{5'}$	4.1	4.1	171
Z_N (Sec. 4.1)	4.2	4.2	178
ad hoc	4.6	4.7	188
$Z_\Gamma = Z_{Bi}$	4.9	5.0	185
profile λ_P	5.0	5.0	185
profile λ_G	4.7	4.7	~182

Exact solution

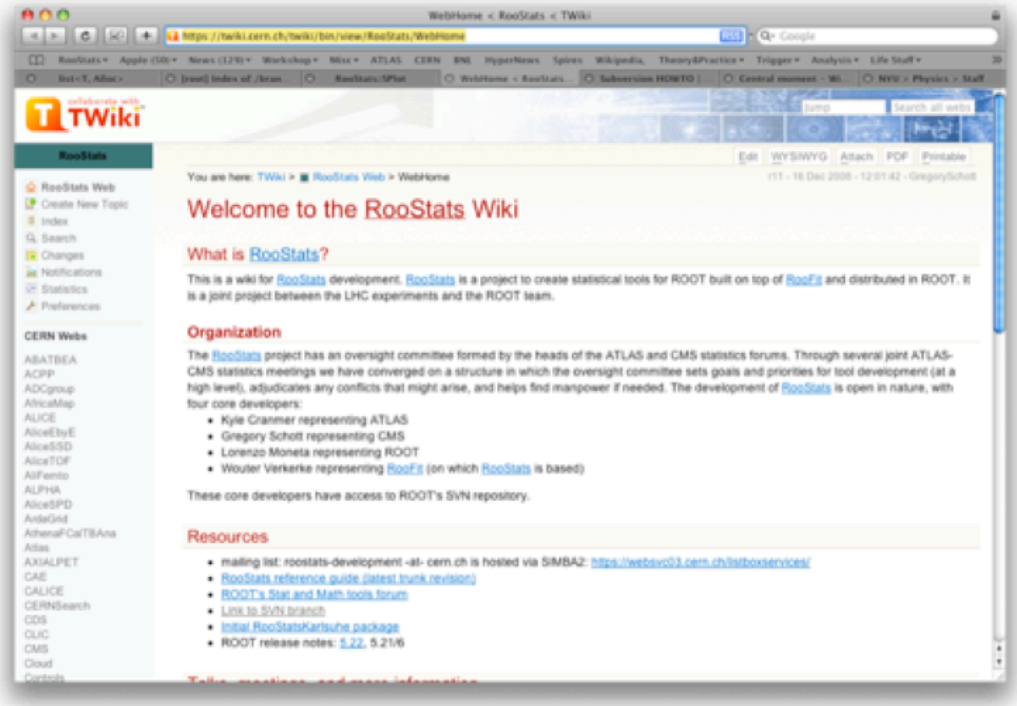
RooStats - Software for hypothesis testing and limit calculation

RooStats Project – Overview

- Goals:
  - Standardize interface for major statistical procedures so that they can work on an arbitrary RooFit model & 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:
  - Essentially all methods start with the basic probability density function or likelihood function. *Building a good model is the hard part*. Want to re-use it for multiple methods → Use RooFit to construct models
  - Build series of tools that perform statistical procedures on RooFit models

The RooStats project



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

Release notes:
http://root.cern.ch/root/v524/Version524.news.html#roofit

Code documentation:
http://root.cern.ch/root/html/ROOFIT_ROOSTATS_Index.html

There is also a category in ROOT's Savannah bug tracking system

Core developers

- K. Cranmer (ATLAS)
- Gregory Schott (CMS)
- Wouter Verkerke (RooFit)
- Lorenzo Moneta (ROOT)
- open project, you are welcome to join. Contributions from
  - Max Baak, Kevin Belasco, Danilo Piparo, Giacinto Piacquadio, Maurizio Pierini, George H. Lewis, Alfio Lazzaro, Mario Pelliccioni, Matthias Wolf
  - Included since ROOT v5.22
- Example macros in
  - $ROOTSYS/tutorials/roostats

Documentation

- code doc. via ROOT
- users manual currently 32 pages, linked from Wiki page

RooFit/RooStats Project – 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** (limits, interval calculators & utilities)
  - Profile Likelihood calculator
  - Neyman construction (FC)
  - Bayesian calculator (BAT & native MCMC)
  - Utilities (combinations, construct pdfs corresponding to standard number counting problems)

RooStats Project – Example

- Create a model - Example

$$Poisson(x \mid s \cdot r_s + b \cdot r_b) \cdot Gauss(r_s,1,0.05) \cdot Gauss(r_b,1,0.1)$$

Create workspace with above model (using factory)

```
RooWorkspace* w = new RooWorkspace("w");
w->factory("Poisson::P(obs[150,0,300],
                       sum::n(s[50,0,120]*ratioSigEff[1.,0,2.],
                              b[100,0,300]*ratioBkgEff[1.,0.,2.]))");
w->factory("PROD::PC(P, Gaussian::sigCon(ratioSigEff,1,0.05),
                        Gaussian::bkgCon(ratioBkgEff,1,0.1))");
```

Contents of workspace from above operation

```
RooWorkspace(w) w contents

variables
---------
(b,obs,ratioBkgEff,ratioSigEff,s)

p.d.f.s
-------
RooProdPdf::PC[ P * sigCon * bkgCon ] = 0.0325554
  RooPoisson::P[ x=obs mean=countingModel_2 ] = 0.0325554
    RooAddition::n[ s * ratioSigEff + b * ratioBkgEff ] = 150
  RooGaussian::sigCon[ x=ratioSigEff mean=1 sigma=0.05 ] = 1
  RooGaussian::bkgCon[ x=ratioBkgEff mean=1 sigma=0.1 ] = 1
```

RooStats Project – Example

- Confidence intervals calculated with model
  - Profile likelihood

```
ProfileLikelihoodCalculator plc;
plc.SetPdf(w::PC);
plc.SetData(data); // contains [obs=160]
plc.SetParameters(w::s);
plc.SetTestSize(.1);
ConfInterval* lrint = plc.GetInterval(); // that was easy.
```

  - Feldman Cousins

```
FeldmanCousins fc;
fc.SetPdf(w::PC);
fc.SetData(data); fc.SetParameters(w::s);
fc.UseAdaptiveSampling(true);
fc.FluctuateNumDataEntries(false);
fc.SetNBins(100); // number of points to test per parameter
fc.SetTestSize(.1);
ConfInterval* fcint = fc.GetInterval(); // that was easy.
```

  - Bayesian (MCMC)

```
UniformProposal up;
MCMCCalculator mc;
mc.SetPdf(w::PC);
mc.SetData(data);  mc.SetParameters(s);
mc.SetProposalFunction(up);
mc.SetNumIters(100000); // steps in the chain
mc.SetTestSize(.1); // 90% CL
mc.SetNumBins(50); // used in posterior histogram
mc.SetNumBurnInSteps(40);
ConfInterval* mcmcint = mc.GetInterval();
```

Interfacing RooFit to RooStats → ModelConfig

- RooFit provides a very flexible toolkit for model building, but a RooFit pdf does not unambiguously specify a statistical model
  - No distinction is made between parameters and observables and conditional onservables
  - No distinction is made between nuisance parameters and parameters of interest
  - (Bayesian) Priors are not part of any model
- Add a new class that specifies all these items: RooStats::ModelConfig.
- An instance of class ModelConfig can be stored inside a workspace so that a workspace with a ModelConfig inside presents a complete and self-documenting description of a statistical model that can be analyzed by RooStats tools

The ModelConfig and Workspace Interface

```
namespace RooStats {
 class ModelConfig : public TNamed {
  /// specify meaning of variables: observable, parameter, etc.
   const RooArgSet * GetObservables() ;
   const RooArgSet * GetParametersOfInterest() ;
   const RooArgSet * GetNuisanceParameters() ;
   const RooArgSet * GetConditionalObservables() ;

   /// specify the objective pdf and the Bayesian prior separately
   RooAbsPdf * GetPdf() ;
   RooAbsPdf * GetPriorPdf() ;

   /// specify value of parameters for a particular hypothesis
   const RooArgSet *    GetSnapshot () ;

   /// get the associated workspace
   const RooWorkspace * GetWS() ;

   /// corresponding Set methods
   virtual void SetPdf(RooAbsPdf& pdf) ;
   ...
```

```
class RooWorkspace : public TNamed {
// import functions
Bool_t import(const RooAbsArg& arg, ....) ;

// use a low-level factory to create and edit objects in the workspace
RooAbsArg*          factory(const char* expr) ;

// Accessor functions
RooAbsPdf*          pdf(const char* name) ;
RooAbsData*         data(const char* name) ;
RooRealVar*         var(const char* name) ;
const RooArgSet*    set(const char* name) ;

// Write this workspace to a ROOT file
writeToFile(const char* fileName, Bool_t recreate=kTRUE) ;

// import class code for custom classes (eg. custom PDFs and functions)
autoImportClassCode(Bool_t flag) ;
```

An example with ModelConfig

Here we show use of the Workspace factory to create a model, and use of ModelConfig to specify what we will need for the statistical tools

Create a new workspace

Create a the pdf G(x|mu,1) and the variables x, mu, sigma using the factory syntax

Create a new ModelConfig

Define parameter sets for observables and parameters of interest

```
// make a simple model via the workspace factory
RooWorkspace* wspace = new RooWorkspace();
wspace->factory("Gaussian::normal(x[-10,10],mu[-1,1],sigma[1])");
wspace->defineSet("poi","mu");
wspace->defineSet("obs","x");

// specify components of model for statistical tools
ModelConfig* modelConfig = new ModelConfig("G(x|mu,1)");
modelConfig->SetWorkspace(*wspace);
modelConfig->SetPdf( *wspace->pdf("normal") );
modelConfig->SetParametersOfInterest( *wspace->set("poi") );
modelConfig->SetObservables( *wspace->set("obs") );

// create a toy dataset
RooDataSet* data = wspace->pdf("normal")->generate(*wspace->set("obs"),100);
```

Specify workspace that holds pdf, parameters of interest, observables, ...

... and we generate a toy dataset with 100 measurements of the observables (x)
(note, the data is NOT part of the ModelConfig)

RooStats: interfaces for statistical test and results

We try to keep a clean separation between the tools and the results

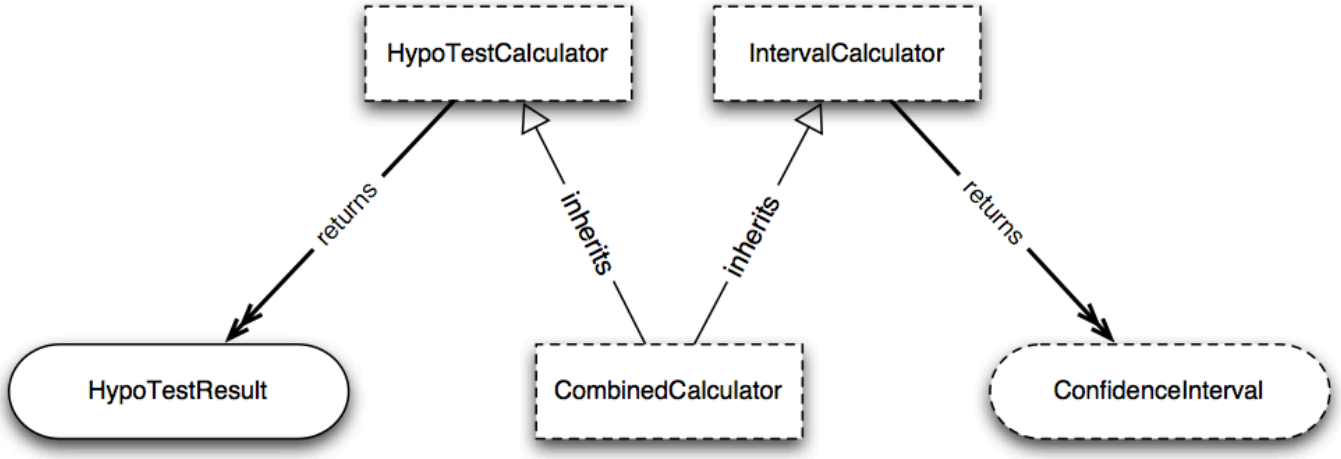
- results in general can be saved into a ROOT file

Two main classes of tools: Hypothesis Tests & Confidence Intervals

- each has an abstract interface for tools and results

Let's think about the interfaces,

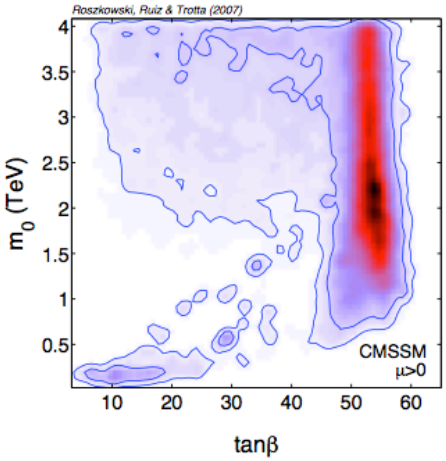
- what do they need to know?
- what do they provide?



Confidence Intervals

ConfInterval is the interface for confidence intervals in RooStats.

- There are many types of intervals: from a simple range [a,b] in 1-D to a complicated, disconnected region in multiple dimensions.
  - So the common interface is simply to ask the interval if a given point **IsInInterval**.
  - The Interval also knows what **ConfidenceLevel** it was constructed at and the space of parameters for which it was constructed.



- Any tool inheriting from **IntervalCalculator** can return a ConfInterval.

```
namespace RooStats {
   class ConfInterval : public TNamed {

      // 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;
   }
}
```

Tools that calculate confidence intervals

IntervalCalculator is the interface for a tools which produce ConfIntervals.

- After configuring the calculator, one only needs to ask **GetInterval**, which will return a ConfInterval pointer (the user takes ownership of new interval).
- assumes that any interval calculator can be configured by specifying the:
  - single model configuration (one model over a space of parameters of interest),
  - data set,
  - confidence level

```
namespace RooStats {
   class 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;

   };
}
```

Hypothesis tests results

HypoTestResult is the class that holds the result of a Hypothesis Test

- Very simple class:
  - stores p-value for the null and alternate hypotheses as calculated by a **HypoTestCalculator**
  - equivalently, Gaussian significance, CLb and CLs+b
  - (note can calculate CLs = CLs+b / CLb, which is used within physics, but is not a probability)
- Note, HypoTestResult is a concrete class, not an interface.

```
namespace RooStats {
   class HypoTestResult : public TNamed {

      // 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;
   };
}
```

Hypothesis test calculators

HypoTestCalculator is the interface tools that produce **HypoTestResults**

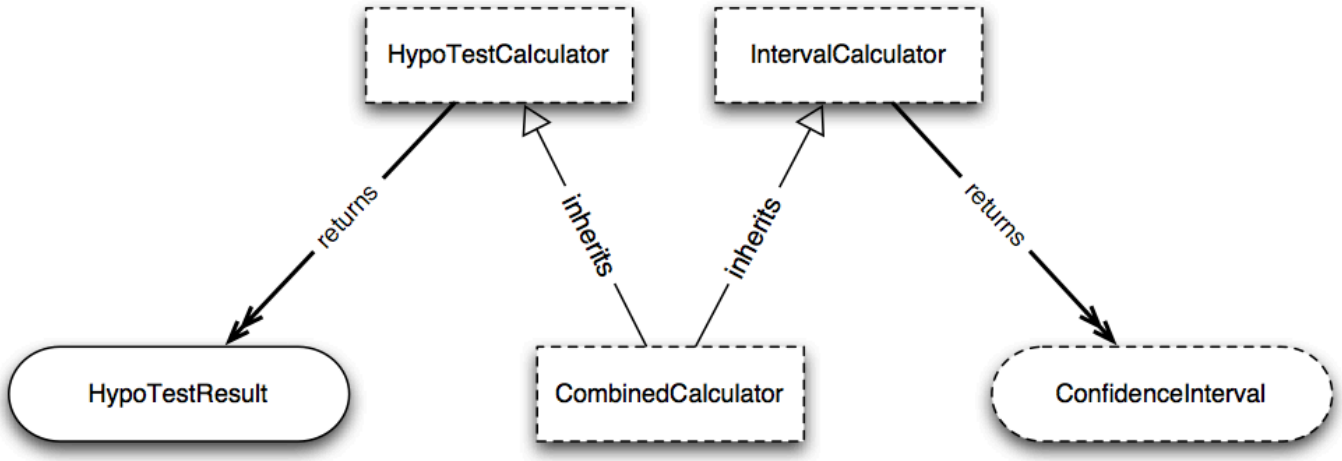
- The interface currently assumes that any interval calculator can be configured by specifying:
  - a model configuration for the null,
  - a model configuration for the alternate (often a totally different PDF),
  - a data set,
- After configuring the calculator, one simply calls **GetHypoTest**, which will return a HypoTestResult pointer (the user takes ownership of new object).

```
namespace RooStats {
   class HypoTestCalculator {

      // main interface to get a HypoTestResult, pure virtual
      virtual HypoTestResult* GetHypoTest() const = 0;

      // 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;

      // Set a common model for both the null and alternate
      virtual void SetCommonModel(const ModelConfig& model) ;
   }
}
```

Combined calculators (of hypo tests and intervals)

CombinedCalculator is the interface tools that can do both hypothesis tests and produce confidence intervals

- In this case, the null and alternate models share the same pdf, and are specified by specific settings for the parameters of interest

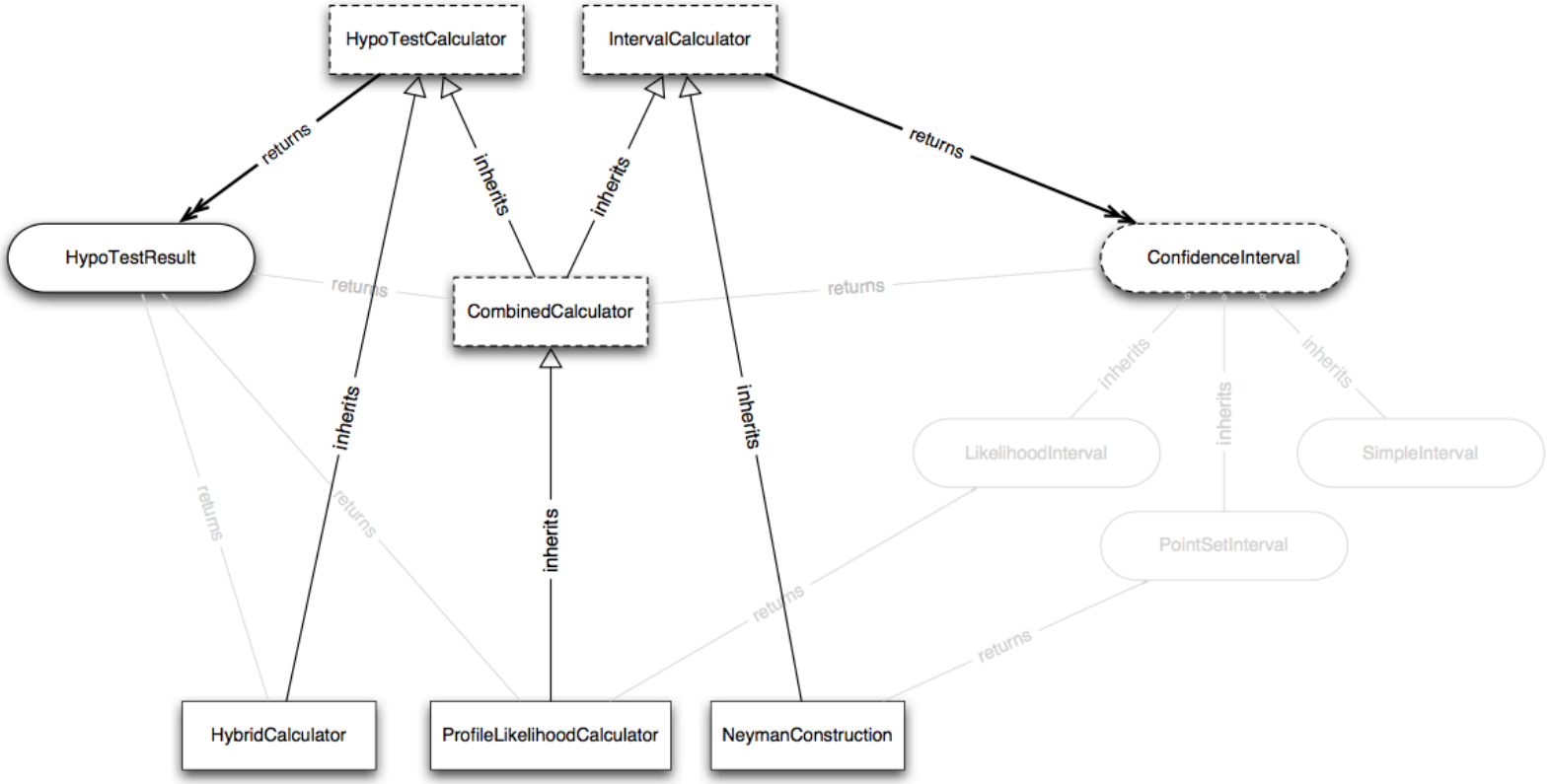


Concrete implementations of calculators

HybridCalculator is an example of a HypoTestCalculator, it returns a HypoTestResult

ProfileLikelihoodCalculator is an example of a CombinedCalculator, it can return either a HypoTestResult or a ConfInterval

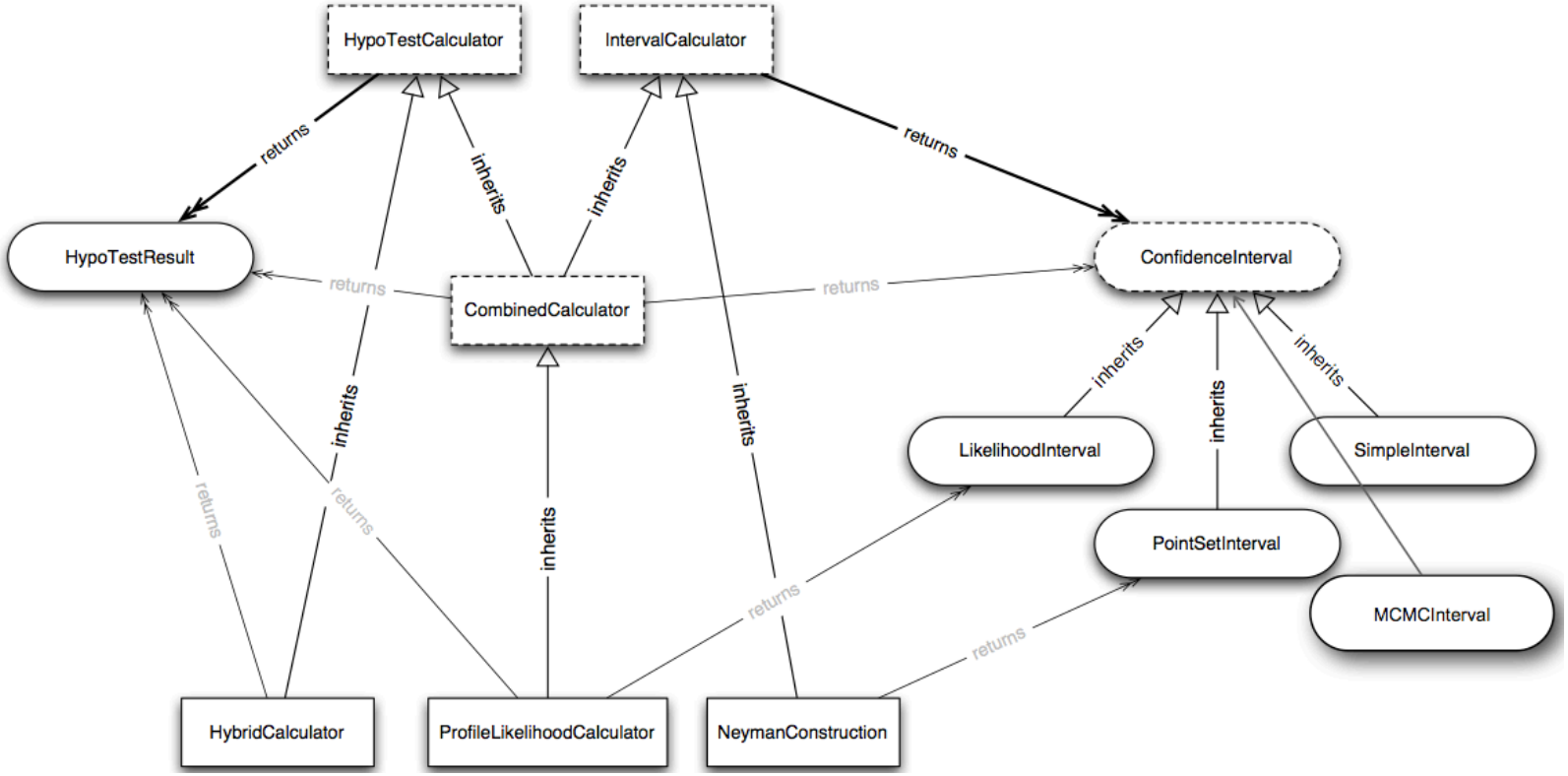
NeymanConstruction is an example of an IntervalCalculator, it returns a ConfInterval



Concrete implementations of intervals

The **ProfileLikelihoodCalculator** returns a **LikelihoodInterval**, which is a concrete implementation of **ConfInterval** (based on contours of likelihood function)

The **NeymanConstruction** returns a **PointSetInterval**, which is a different concrete implementation (based on scanning points in the parameter space)



List of calculator tools

HypoTestCalculators

- HybridCalculator
  - hybrid Bayes-frequentist calculation (marginalize nuisance parameters)
- ProfileLikelihoodCalculator
  - the method of MINUIT/MINOS, based on Wilks's theorem

IntervalCalculators

- ProfileLikelihoodCalculator
  - the method of MINUIT/MINOS, based on Wilks's theorem
- NeymanConstruction
  - general purpose Neyman Construction class, highly configurable: choice of TestStatistic, TestStatSampler (defines ensemble/conditioning), integration boundary (upper, lower, central limits), and parameter points to scan
- FeldmanCousins
  - specific configuration of NeymanConstruction for Feldman-Cousins (generalized for nuisance parameters)
- MCMCCalculator
  - Bayesian Markov Chain Monte Carlo (Metropolis Hastings), proposal function is highly customizable
- BayesianCalculator
  - Bayesian posterior calculated via numeric integration routines, currently only supports one parameter
- HypoTestInverter
  - adapter any HypoTestCalculator and forms an IntervalCalculator

Putting it together in RooStats

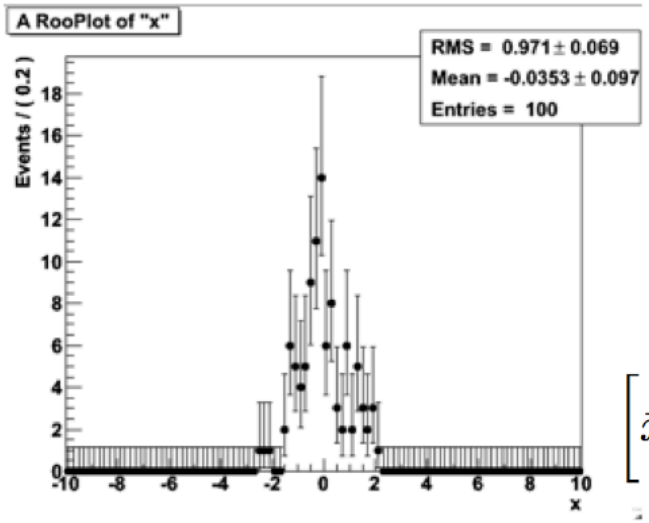
An example problem

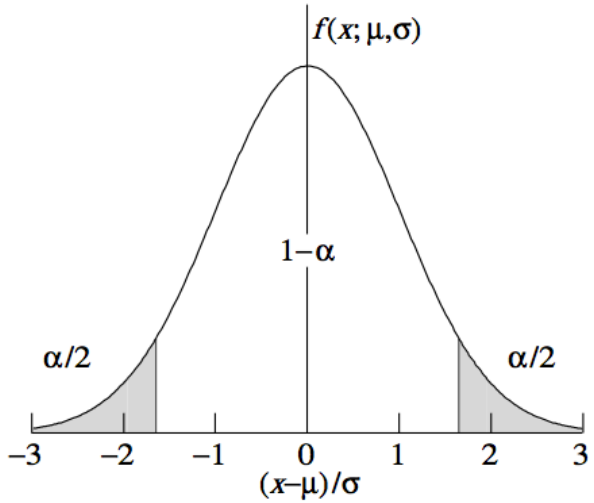
Let's consider an example where we know the answer:

- 95% CL interval on the mean of a Gaussian with sigma=1
- generate a toy dataset with N = 100
- Look up in PDG what we should expect
- $\sigma = 1,\ \delta = 1.96,\ N = 100$

Table 32.1: Area of the tails α outside $\pm\delta$ from the mean of a Gaussian distribution.

α	δ	α	δ
0.3173	1σ	0.2	1.28σ
4.55×10^{-2}	2σ	0.1	1.64σ
2.7×10^{-3}	3σ	0.05	1.96σ
6.3×10^{-5}	4σ	0.01	2.58σ
5.7×10^{-7}	5σ	0.001	3.29σ
2.0×10^{-9}	6σ	10^{-4}	3.89σ





$$\left[\bar{x} - \frac{\delta}{\sqrt{N}}, \bar{x} + \frac{\delta}{\sqrt{N}}\right] \quad \bar{x} = -0.035$$

expected interval → $[-0.23, 0.16]$

Creating the model

Here we show use of the Workspace factory to create a model, and use of ModelConfig to specify what we will need for the statistical tools

Create a new workspace

Create a the pdf G(x|mu,1) and the variables x, mu, sigma using the factory syntax

Create a new ModelConfig

Define parameter sets for observables and parameters of interest

```
// make a simple model via the workspace factory
RooWorkspace* wspace = new RooWorkspace();
wspace->factory("Gaussian::normal(x[-10,10],mu[-1,1],sigma[1]))");
wspace->defineSet("poi","mu");
wspace->defineSet("obs","x");

// specify components of model for statistical tools
ModelConfig* modelConfig = new ModelConfig("G(x|mu,1)");
modelConfig->SetWorkspace(*wspace);
modelConfig->SetPdf( *wspace->pdf("normal") );
modelConfig->SetParametersOfInterest( *wspace->set("poi") );
modelConfig->SetObservables( *wspace->set("obs") );

// create a toy dataset
RooDataSet* data = wspace->pdf("normal")->generate(*wspace->set("obs"),100);
```

Specify workspace that holds pdf, parameters of interest, observables, ...

... and we generate a toy dataset with 100 measurements of the observables (x)

Using the profile likelihood calculator

Once one has the model and the data, creating the interval is quite easy!

Create a **ProfileLikelihoodCalculator**, constructor needs data and the model config

Want a 95% CL

```
// set confidence level
double confidenceLevel = 0.95;

// example use profile likelihood calculator
ProfileLikelihoodCalculator plc(*data, *modelConfig);
plc.SetConfidenceLevel( confidenceLevel);
LikelihoodInterval* plInt = plc.GetInterval();
```

Obtain the resulting interval

(Note, here we know the return type is a **LikelihoodInterval**, in general one could use the interface **ConfInterval.** Note, you take ownership of `plInt`.)

Use the interval

```
cout << "plc interval is [ " <<
  plInt->LowerLimit(*mu) << ", " <<
  plInt->UpperLimit(*mu) << "]" << endl;
mu->setVal(0);
cout << "is mu=0 in the interval? " << plInt->IsInInterval(*mu) << endl;
```

```
expected interval is [-0.231277, 0.160716]
plc interval is      [-0.231277, 0.160716]
is mu=0 in the interval? 1
```

!

Example using the Feldman Cousins calculator

```
// example use of Feldman-Cousins
FeldmanCousins fc(*data, *modelConfig);
fc.SetConfidenceLevel( confidenceLevel);

// special options
fc.SetNBins(200); // number of points to test per parameter
fc.UseAdaptiveSampling(true); // make it go faster

PointSetInterval* interval = fc.GetInterval();



std::cout << "fc interval is ["<<
  interval->LowerLimit(*mu) << " , "  <<
  interval->UpperLimit(*mu) << "]" << endl;
```

```
expected interval is [-0.231277, 0.160716]
plc interval is      [-0.231277, 0.160716]
fc interval is       [-0.215   , 0.165   ]
Real time 0:00:36, CP time 34.900
```

By default, the FeldmanCousins calculator only samples 10 points per parameter, (pretty fast)

- modify this with SetNBins

The default number of samples is 50/(type I error rate)

- For 95% CL, 1000 toys/point
- for 200 parameter points, that's 200,000 generations of datasets and MINUIT minimizations! About 15 min.

The AdaptiveSampling algorithm will use fewer toys for obvious points and more near boundary

- reduces it to 2 min!
- will explain algorithm later on

Note F-C interval is consistent within step-size for the upper limit, off by one step in lower limit... likely a statistical fluctuation in ToyMC.

Example using BayesianCalculator

```
// example use of BayesianCalculator
// now we also need to specify a prior in the ModelConfig
wspace->factory("Uniform::prior(mu)");
modelConfig->SetPriorPdf(*wspace->pdf("prior"));

// example usage of BayesianCalculator
BayesianCalculator bc(*data, *modelConfig);
bc.SetConfidenceLevel( confidenceLevel);
SimpleInterval* bcInt = bc.GetInterval();
```

The BayesianCalculator requires an additional ingredient beyond what is currently in our ModelConfig

- it needs a prior for mu

Once that is specified in the ModelConfig, use of the BayesianCalculator is the same as the other tools

- currently this tool is restricted to one parameter, waiting for some additional support in underlying RooFit integration infrastructure.

```
expected interval is [-0.231277, 0.160716]
plc interval is      [-0.231277, 0.160716]
fc interval is       [-0.215   , 0.165   ]
bc interval is       [-0.232274, 0.159713]
```

Note bayesian interval is based on numerical integration instead of ToyMC. Accuracy of answer depends on accuracy specified in numerical integration, can be configured.

Example using Markov Chain Bayesian Calculator

```
// example use of MCMCInterval
MCMCCalculator mc(*data, *modelConfig);
mc.SetConfidenceLevel( confidenceLevel);

// special options
mc.SetNumBins(1000);     // bins used internally for representing posterior
mc.SetNumBurnInSteps(500); // first N steps to be ignored as burn-in
mc.SetNumIters(100000);

MCMCInterval* mcInt = mc.GetInterval();
```

The MCMCCalculator requires the same ingredients as the Bayesian Calculator, but uses Markov Chain Monte Carlo to calculate the posterior

This class works is very configurable, and will be described in more detail later.

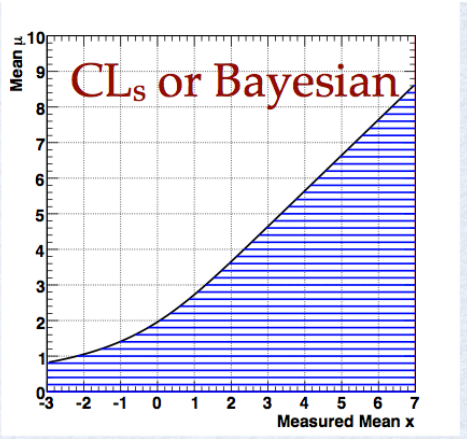
- here we set the binning resolution for the parameter of interest, the number of "burn in" steps, the number of iterations to run the MCMC

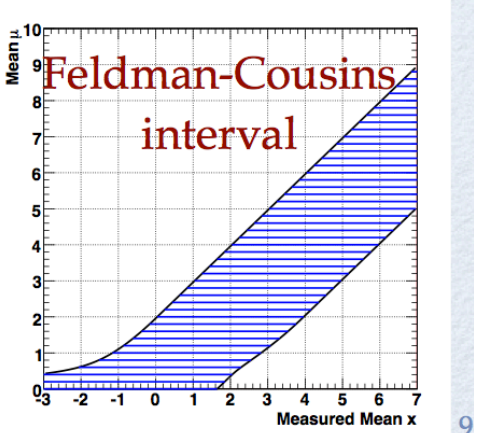
```
expected interval is [-0.231277, 0.160716]
plc interval is      [-0.231277, 0.160716]
fc interval is       [-0.215   , 0.165   ]
bc interval is       [-0.232274, 0.159713]
mc interval is       [-0.227   , 0.157   ]
```

Hypothesis test inversion

Generic interval calculation

- RooStats provide a flexible framework to do Neyman construction in a modular way in software
- Classes that construct test statistic from a Workspace
  - ProfileLikelihoodRatio (LHC)
  - RatioOfProfiledLikelihoods (Tevatron)
  - SimpleLikelihoodRatio
  - (your own)
- Universal test statistic sampler class that maps the confidence belt
  - With support for PROOF(lite) parallelization
- A Class to invert hypothesis tests in belt into a confidence interval
- A standard script that drives the whole procedure





An example driver script

Tool to calculate p-values for a given hypothesis

$$\int_{q_{\mu,obs}}^{\infty} f(q_\mu \mid \mu')dq_\mu$$

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

// 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);

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

// 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();
```

$f(q_\mu \mid \mu')$

Tool to construct test statistic distribution

$q_\mu(\mu')$

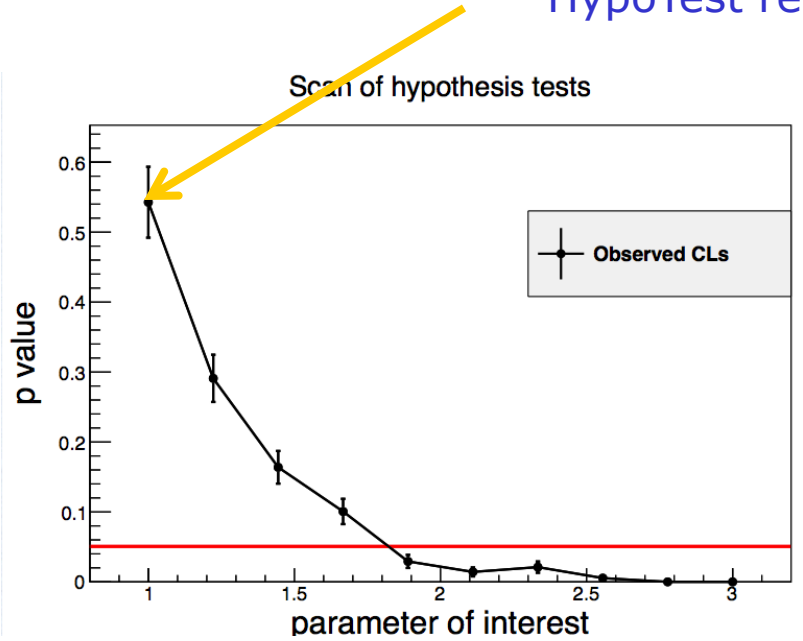
The test statistic to be used for the calculation of p-values

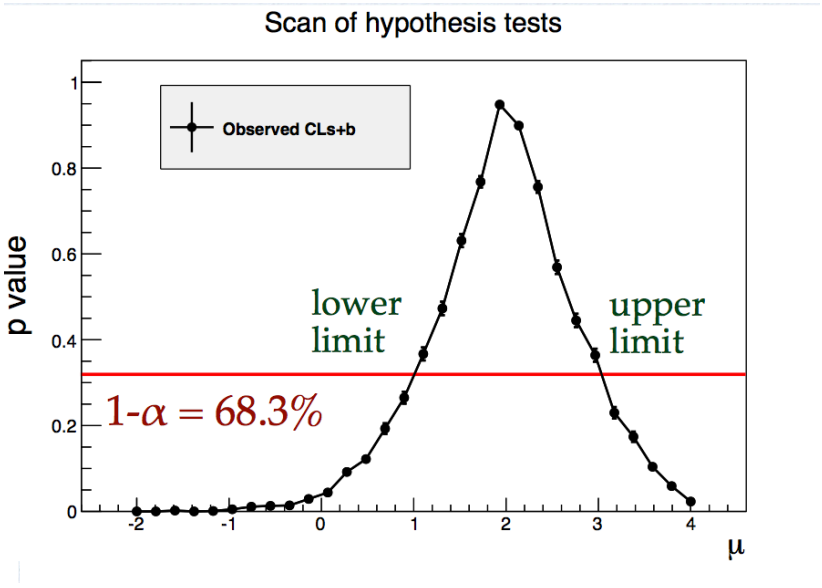
Tool to construct interval from hypo test results

Example output of hypothesis test inversion

- Hypothesis test calculator computes p-value for each value of μ

HypoTest result (p-value) at given μ (here μ=1)



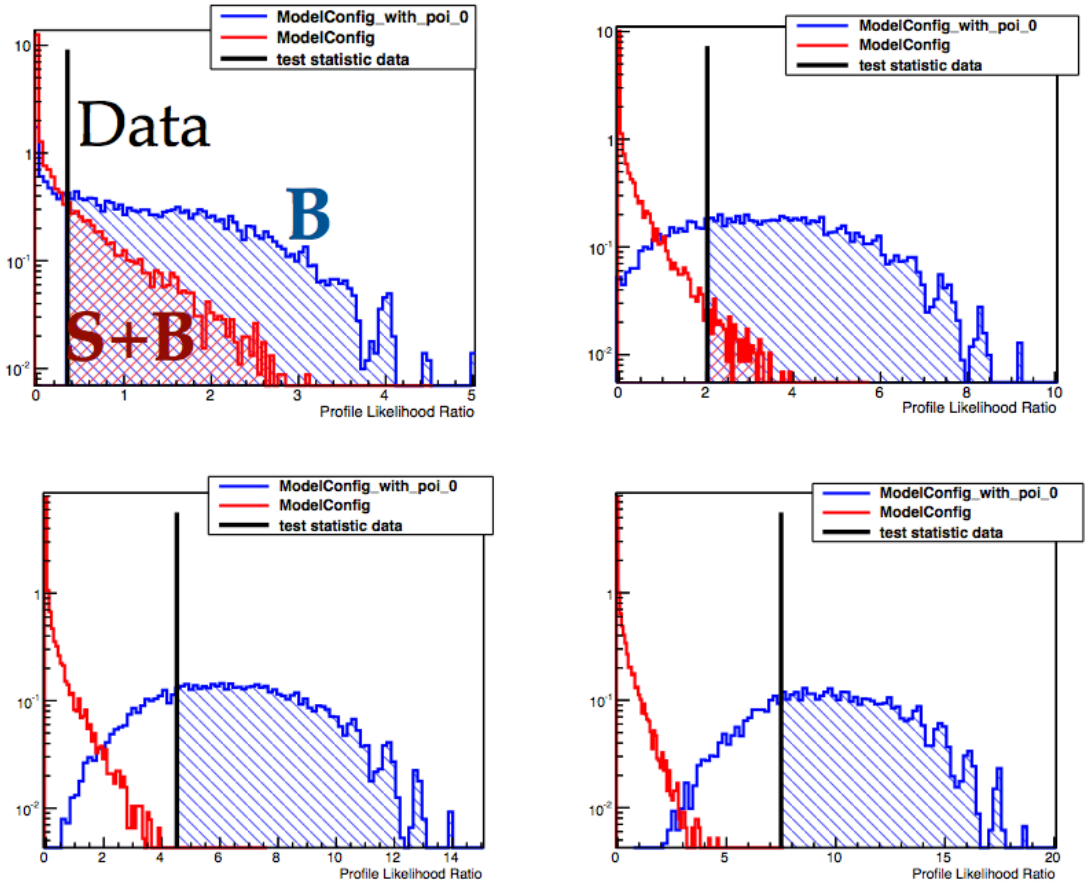


One-sided in interval (upper limit) at 95% C.L.

Two-sided in interval at 68% C.L.

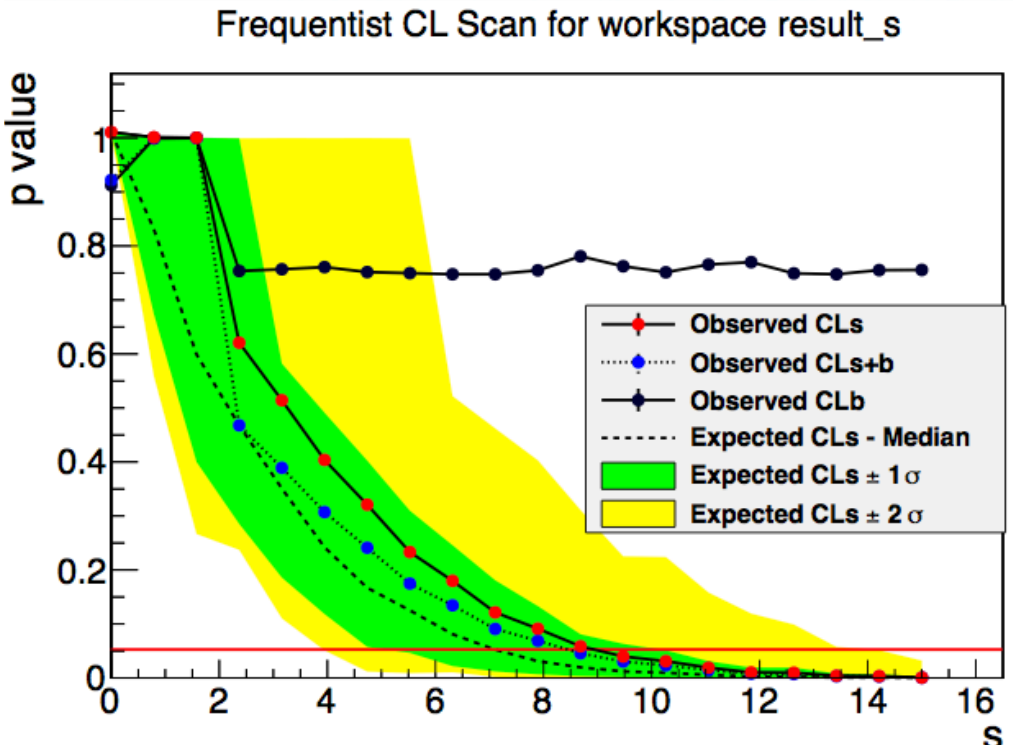
Detailed output also possible

- Example: distributions of test statistic q_μ for toys generated with $\mu_{gen}=0$ and $\mu_{gen}=\mu$ for each scanned point of μ



Additional useful information CL_S, expected limits

- Can also show in p-value vs μ scan plots
  - Observed CL_S (= $p_\mu/(1-p_0)$)
  - Observed p_0
  - Expected CL_S (median and 1,2σ bands)



The 'standard' driver script

- Input information needed
  - Input workspace (file name and workspace name)
  - Name of ModelConfig object to be used in workspace
    - Specifies S+B model, B model (if not S+B with μ=0), POI, nuisance params etc
  - Name of observed dataset in workspace
- Statistics options
  - Calculator type (Frequentist, Hybrid, Asymptotic)
  - Test statistic (ProfileLR [LHC], RatioOfPLR [TeV], LR [LEP])
  - Use CL_S technique (yes/no)
- Technical options
  - Range of POI to scan
  - Fixed number of steps (for nice plots),
    or -1 for adaptive sampling (for precise and fast limit calculations)

```
load the macro after having create the workspace using given macro (e.g. 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)
```

Exercises

Exercises A – Specifying a ModelConfig

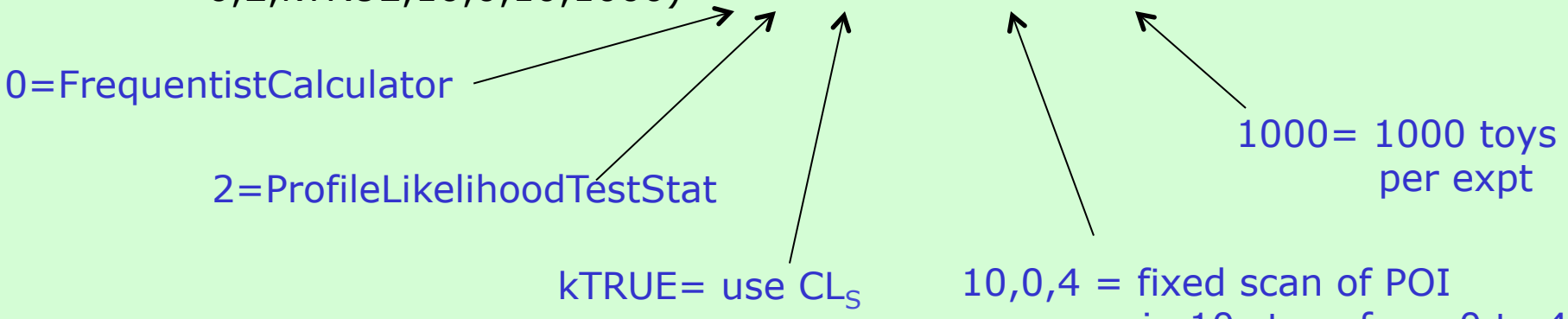
- To make generic RooFit pdfs suitable for statistical analysis the following must be defined
  - The pdf f(x|mu,theta)
  - The parameter-of-interest (mu)
  - The nuisance parameters (theta)
  - If only one pdf is specified it is assumed to be for the signal hypothesis, and the background hypothesis is assumed to that pdf with mu=0
- Copy mod3/ex3A.C as starting point
  - The file contains a slight modification of ex2C (Gaussian+Exponential), now as extended pdf and with somewhat different parameters
  - Fix parameters 'mean' and 'tau' (using var->setConstant(kTRUE)) so that the model effectively has no nuisance parameters
  - Create a ModelConfig object [ModelConfig mc("name",wspacePtr)] ;
  - Configure it by calling its methods SetPdf(RooAbsPdf*) and SetObservables (),SetParametersOfInterest(),SetNuisanceParameters()
    - The latter three all take a RooArgSet as argument. If you need to specify only one element, you can optionally omit the set. If you need to pass an empty set, simply pass RooArgSet().
  - Import the ModelConfig object mc also in the workspace (use import()) and write the workspace to a ROOT file named "splusbmodel.root"
- Copy the completed ex3A.C to ex3A_np.C
  - In this copy remove the lines that fix mean and tau, and instead define these as nuisance parameters in ModelConfig
  - Write this configuration of the workspace into file "splusbmodel_np.root"

Exercise B – Using RooStats standard tools

- Copy mod3/StandardProfileLikelihoodDemo.C
  - NB You can also find this file in $ROOTSYS/tutorials/roostats
  - Run 'root –l StandardProfileLikelihoodDemo.C ("splusbmodel.root","w")' which will visualize the profile likelihood scan and the 95% interval associated with the profile likelihood ratio
  - Repeat for splusbmodel_np.root. What is the effect of the nuisance parameters?
- Copy mod3/StandardTestStatDistributionDemo.C
  - NB You can also find this file in $ROOTSYS/tutorials/roostats
  - Run as the above macro, passing the file name and workspace name for both root files
  - This macro will show the distribution of the test statistic q_mu_hat, with mu set to the fitted POI value and overlay the distribution of q_mu_hat generated from the pdf with mu=mu_hat. Overlaid is the asymptotic expectation of this distribution (a half chi-square distribution)

Exercise B - continued

- Copy mod3/StandardHypoTestInvDemo.C
  - NB You can also find this file in $ROOTSYS/tutorials/roostats
  - This is the single most powerful and useful macro of roostats that runs the full frequent limit setting procedure for *any* pdf stored in a workspace with a modelconfig.
- Iteration 1 – Limit from Hypothesis test inversion based on Neyman construction with standard profile likelihood test statistic (t_mu)
  - Run `root –l `StandardHypoTestInvDemo ("splusbmodel_np.root","w", "ModelConfig","","obsData", 0,2,kTRUE,10,0,10,1000)



Exercise B - continued

  - It will take about 5 minutes. If you have a multi-core host you can speed this up: edit the macro and set useProof to true and set nworkers to the number of cores
  - You will get output plots with the test statistic distributions for each sampled point of the POI (with toys for signal and background in each). The scan information is summarized in another plot that shows the p-values (p(s+b),p(b) and CLS) as function of the POI, and the expected limits with one and 2 sigma bands.

- Iteration 2 - Rerun with the one-sided LHC test statistic
  - Choose 3=OneSidedProfileLikelihoodTestStat (as the 2nd integer argument).
  - This this give a stronger exclusion?
- Iteration 3 – Adaptive scanning of pvalues-vs-POI curve for improved precision
  - Set the number of scan points to -1 and rerun

Exercise C – A template model

- Copy mod3/ex3C.C
  - This file contains a template likelihood model of the form f(x|alpha)g(y|alpha) where f(x|alpha) is a template morphing model - the 'main measurement' and g() is a subsidiary measurement that represents the result of an external measurement on alpha (here modeled as a Gaussian)
- Visualize the model
  - To understand the structure of this model, visualize the main measurement pdf 'main_measurement' as function of both x and alpha:
    - TH2* hh_pdf = pdf->createHistogram("x,alpha")->Draw("lego") ;
- Prepare the model
  - Write a ModelConfig object for this pdf, include it in the workspace and write it to a file "npmodel.root". Note that the model config for such a pdf needs one extra specification: the "subsidiary observables" must be specified (in this case 'y'). Use the method SetGlobalObservables() to do this.
- Calculate limits
  - Repeat exercise B for this model (just change the file name)