Software 2
Hands-on:

Counting models with NPs in RooFit

Limits & Intervals with RooStats

•Wouter Verkerke, NIKHEF

Goals of this hands-on session

1. Learn basics of RooFit model building
   - Learn to use to workspace factory to quickly specify models
   - Focus on counting models this morning – these are quick and easy
2. Learn basics of RooStats limit & interval calculators
   - You can run these on the counting models your built in part one

- We only have 90 minutes – so tailored approach to that
  - First some introductory slides to familiarize you with the syntax of RooFit model building and RooFit model usage
  - 10 prepared macros that are fully functional and execute progressively complex tasks
  - You start from a functional working point – goal of your exercise time is to understand what they do and how they do it and work on some extensions and modifications of the macros

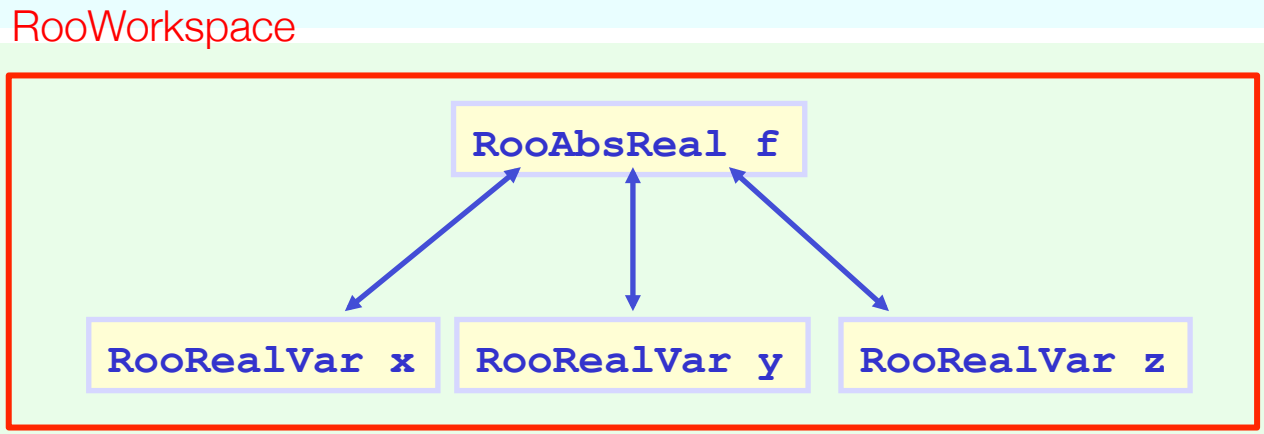
Populating a workspace the easy way – "the factory"

- Creating many objects can be tedious: The workspace factory allows to fill a workspace with pdfs and variables using a simplified scripting language

Math

$$\text{Gauss}(x,\mu,\sigma)$$

RooFit diagram

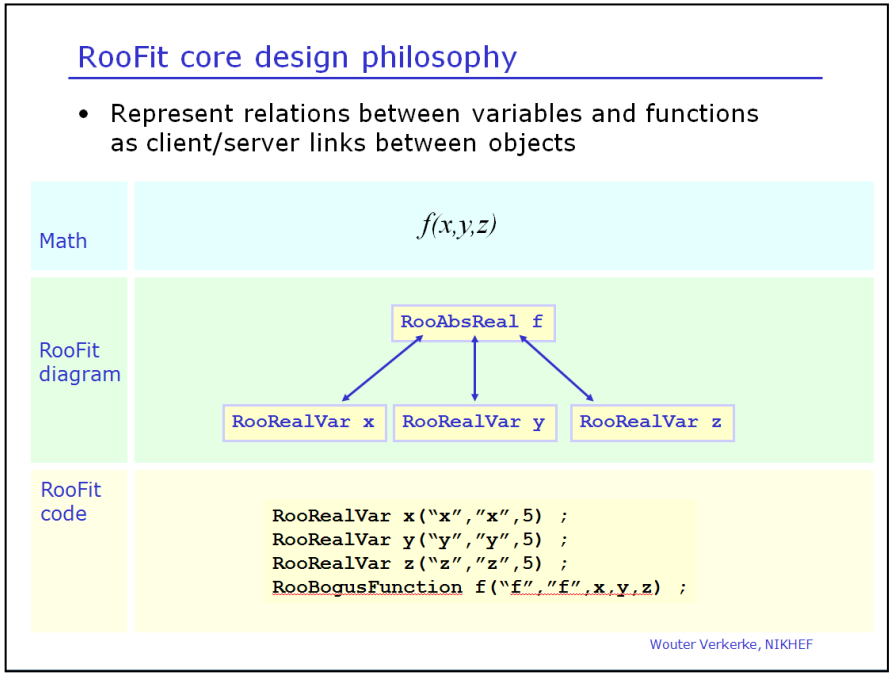


RooFit code

```
RooWorkspace w("w") ;
w.factory("RooGaussian::g(x[-10,10],m[-10,10],z[3,0.1,10])");
```

Factory and Workspace

- *One C++ object per math symbol* provides ultimate level of control over each objects functionality, but results in lengthy user code for even simple macros
- Solution: add factory that auto-generates objects from a math-like language




```
Gaussian::f(x[-10,10],mean[5],sigma[3])
```

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

Factory and Workspace

- This is *not* the same as reinventing Mathematica!
  String constructs an expression in terms of C++ objects, rather than being the expression
  - Objects can be tailored after construction through object pointers
  - For example: tune parameters and algorithms of numeric integration to be used with a given object
- Implementation in RooFit:
  Factory makes objects, Workspace owns them

```
RooWorkspace w("w") ;
w.factory("Gaussian::f(x[-10,10],mean[5],sigma[3])") ;

w.Print("t") ;

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

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

Accessing the workspace contents

- Workspace contents can be obtained through accessor methods

```
// retrieve a probability density function
RooAbsPdf* g = w.pdf("g") ;

// retrieve a regular function
RooAbsReal* f = w.function("f") ;

// retrieving a variable
RooRealVar* x = w.var("x") ;

// retrieving a dataset
RooAbsData* data = w.data("observed") ;
```

Factory language

- The factory language has a 1-to-1 mapping to the constructor syntax of RooFit classes
  - With a few handy shortcuts for variables

- Creating variables

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

- Creating pdfs (and functions)

```
Gaussian::g(x,mean,sigma) → RooGaussian("g","g",x,mean,sigma)
Polynomial::p(x,{a0,a1}) → RooPolynomial("p","p",x",RooArgList(a0,a1));
```

  - Can always omit leading 'Roo'
  - Curly brackets translate to set or list argument (depending on context)

Factory language

- Composite expression are created by nesting statements
  - No limit to recursive nesting

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

- You can also use numeric constants whenever an unnamed constant is needed

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

- Names of nested function objects are optional
  - SUM syntax explained later

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

Factory language

- *Interpreted function expressions* allow to customize existing probability density functions

```
// construct Nexp=mu*S+B (a function)
expr::Nexp('mu*S+B',mu[0,5],S[50],B[50])

// construct a Poisson probability model describing
// the distribution of Nobs given Nexp
Poisson::p(Nobs[0,1000],Nexp) ;
```

- Generally:
  types starting with UPPER-CASE are Probability Density Functions,
  types starting with lower-case are regular functions

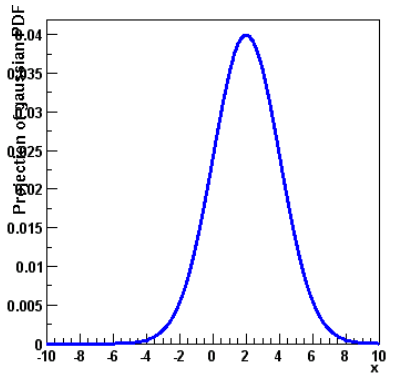
Model building – (Re)using standard components

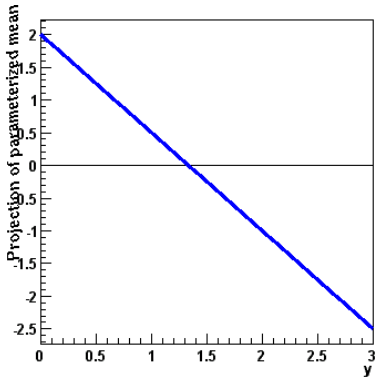
- List of frequently used pdfs and their factory spec

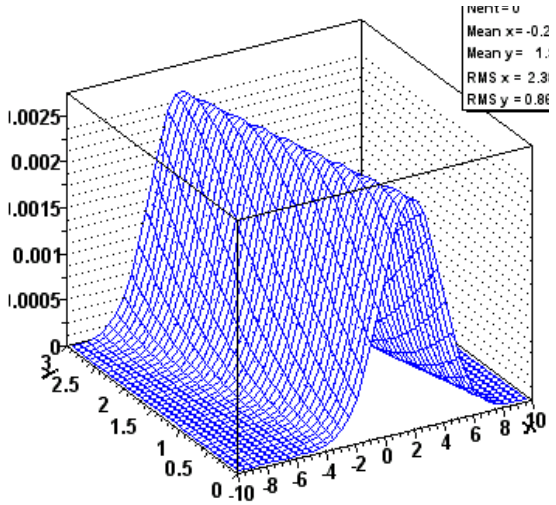
Gaussian	`Gaussian::g(x,mean,sigma)`
Breit-Wigner	`BreitWigner::bw(x,mean,gamma)`
Landau	`Landau::l(x,mean,sigma)`
Exponential	`Exponential::e(x,alpha)`
Polynomial	`Polynomial::p(x,{a0,a1,a2})`
Chebychev	`Chebychev::p(x,{a0,a1,a2})`
Kernel Estimation	`KeysPdf::k(x,dataSet)`
Poisson	`Poisson::p(x,mu)`
Voigtian (=BW⊗G)	`Voigtian::v(x,mean,gamma,sigma)`

Model building – (Re)using standard components

- Library p.d.f.s can be adjusted on the fly.
  - Just plug in *any function expression* you like as input variable
  - Works universally, even for classes you write yourself







g(x;m,s) $m(y;a_0,a_1)$ $g(x,y;a_0,a_1,s)$

```
RooPolyVar  m("m",y,RooArgList(a0,a1)) ;
RooGaussian g("g","gauss",x,m,s) ;
```

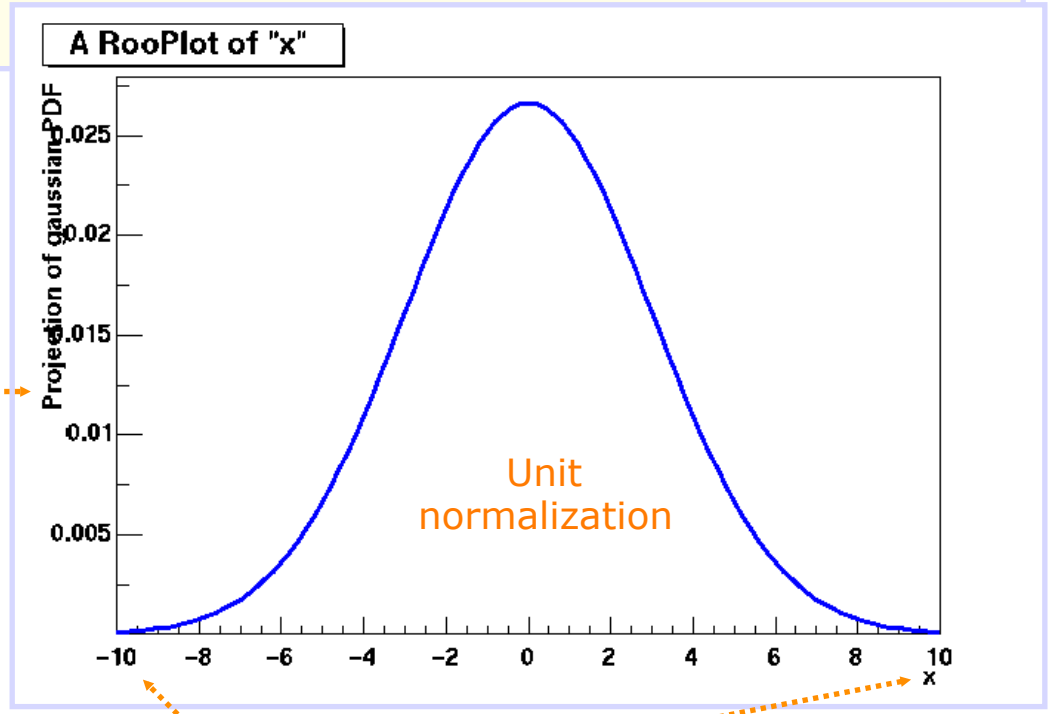
- Maximum flexibility of library shapes keeps library small

Basics – Creating and plotting a Gaussian p.d.f

Setup gaussian PDF and plot

```
// Build Gaussian PDF
w.factory("Gaussian::gauss(x[-10,10],mean[-10,10],sigma[3,1,10]")

// Plot PDF
RooPlot* frame = w.var("x")->frame() ;
w.pdf("gauss")->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`

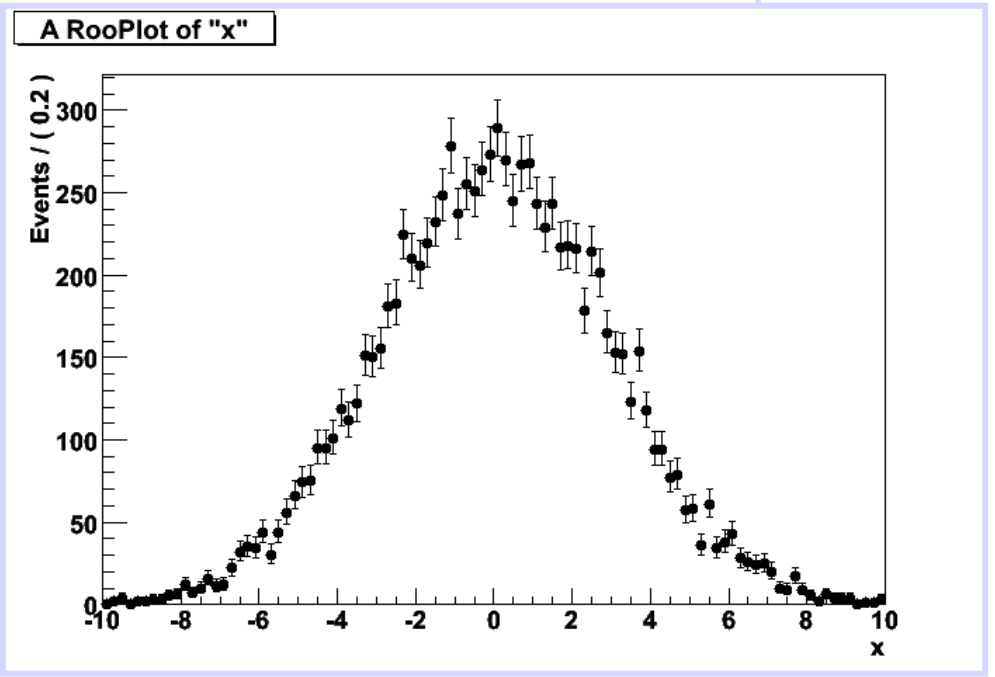
Basics – Generating toy MC events

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

```
// Generate an unbinned toy MC set
RooDataSet* data = w.pdf("gauss")->generate(w::x,10000) ;

// Generate an binned toy MC set
RooDataHist* data =
             w.pdf("gauss")->generateBinned(w::x,10000) ;

// Plot PDF
RooPlot* xframe =
  w.var("x")->frame() ;
data->plotOn(xframe) ;
xframe->Draw() ;
```



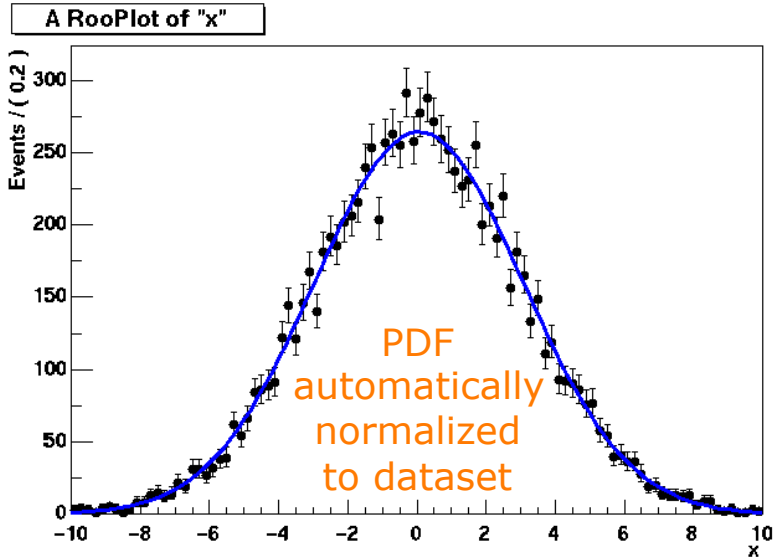
Can generate both binned and unbinned datasets

Basics – ML fit of p.d.f to *unbinned* data

```
// ML fit of gauss to data
w.pdf("gauss")->fitTo(*data)
(MINUIT printout omitted)

// Parameters if gauss now
// reflect fitted values
w.var("mean")->Print()
RooRealVar::mean = 0.0172335 +/- 0.0299542
w.var("sigma")->Print()
RooRealVar::sigma = 2.98094  +/- 0.0217306

// Plot fitted PDF and toy data overlaid
RooPlot* xframe = w.var("x")->frame() ;
data->plotOn(xframe) ;
w.pdf("gauss")->plotOn(xframe) ;
```



Basics – ML fit of p.d.f to *unbinned* data

- Can also choose to save full detail of fit

```
RooFitResult* r = w::gauss.fitTo(*data,Save()) ;

r->Print() ;
  RooFitResult: minimized FCN value: 25055.6,
                estimated distance to minimum: 7.27598e-08
                coviarance matrix quality:
                Full, accurate covariance matrix

    Floating Parameter    FinalValue +/-  Error
  --------------------  --------------------------
                  mean    1.7233e-02 +/-  3.00e-02
                 sigma    2.9809e+00 +/-  2.17e-02


r->correlationMatrix().Print() ;

2x2 matrix is as follows

     |      0    |      1    |
-------------------------------
   0 |          1   0.0005869
   1 |  0.0005869           1
```

Further practical information

- Most useful (in my experience) tutorial macros
  - In every ROOT installation, in directory $ROOTSYS/tutorials/roofit you will find 86 tutorial macros each demonstrating one important task or feature of RooFit

```
rf101_basics.C
rf102_dataimport.C
rf103_interprfuncs.C
rf104_classfactory.C
rf105_funcbinding.C
rf106_plotdecoration.C
rf107_plotstyles.C
rf108_plotbinning.C
rf109_chi2residpull.C
rf110_normintegration.C
rf111_derivatives.C
rf201_composite.C
rf202_extendedmlfit.C
rf203_ranges.C
rf204_extrangefit.C
rf205_compplot.C
rf206_treevistools.C
rf207_comptools.C
rf208_convolution.C
rf209_anaconv.C
rf210_angularconv.C
rf211_paramconv.C
rf301_composition.C
rf302_utilfuncs.C
rf303_conditional.C
rf304_uncorrprod.C
rf305_condcorrprod.C
rf306_condpereventerrors.C
rf307_fullpereventerrors.C
rf308_normintegration2d.C
rf309_ndimplot.C
rf310_sliceplot.C
rf311_rangeplot.C
rf312_multirangefit.C
rf313_paramranges.C
rf314_paramfitrange.C
rf315_projectpdf.C
rf316_llratioplot.C
rf401_importttreethx.C
rf402_datahandling.C
rf403_weightedevts.C
rf404_categories.C
rf405_realtocatfuncs.C
rf406_cattocatfuncs.C
rf407_latextables.C
rf501_simultaneouspdf.C
rf502_wspacewrite.C
rf503_wspaceread.C
rf504_simwstool.C
rf505_asciicfg.C
rf506_msgservice.C
rf507_debugtools.C
rf508_listsetmanip.C
rf509_wsinteractive.C
rf510_wsnamedsets.C
rf511_wsfactory_basic.C
rf512_wsfactory_oper.C
rf513_wsfactory_tools.C
rf601_intminuit.C
rf602_chi2fit.C
rf603_multicpu.C
rf604_constraints.C
rf605_profilell.C
rf606_nllerrorhandling.C
rf607_fitresult.C
rf608_fitresultaspdf.C
rf609_xychi2fit.C
rf610_visualerror.C
rf701_efficiencyfit.C
rf702_efficiencyfit_2D.C
rf703_effpdfprod.C
rf704_amplitudefit.C
rf705_linearmorph.C
rf706_histpdf.C
rf707_kernelestimation.C
rf708_bphysics.C
rf709_momentmorph.C
rf709_momentmorph1.C
rf710_momentmorph2.C
rf801_mcstudy.C
rf802_mcstudy_addons.C
rf803_mcstudy_addons2.C
rf804_mcstudy_constr.C
rf901_numintconfig.C
rf902_numgenconfig.C
rf903_numintcache.C
```

- Class reference on ROOT website
  - Most useful for syntax of key classes like RooAbsPdf, RooWorkspace etc…

Exercise 1

- Macro ex01_expore_Poisson.C
  - Construct a Poisson probability model P(N|μS+B) with S,B fixed
  - Fits model to 25 observed event → returns fitted value of μ
  - Alternatively, explicitly constructs the likelihood L(25|μ) and visualizes that
  - Also visualized on the same frame is L(μ)/L(μ-hat)

- Questions & explorations
  - Do you understand the difference between the likelihood and the likelihood ratio curve
  - How does the interval defined by the rise of the likelihood ratio by half a unit compare to the MINOS error
  - Can you construct the 95% interval from the plot?

Exercise 2

- Macro ex02_build_Poisson.C
  - Construct the same Poisson model as ex01
  - Constructs a RooStats ModelConfig object that describes a uniquely defined statistical problem definition
  - Saves the model to a workspace in a ROOT file

- Questions & explorations
  - Open the workspace in a clean root session after you ran macro ex02, retrieve the Poisson model and the observed data from it and rerun the fit on the command line
  - Plot the Poisson distribution on a frame (see code in Ex01)
  - *Change the value of mu and plot the Poisson distribution for that new mu value on the same frame. Note that you will have to recall frame->Draw() after each addition to visualize it on the canvas.*
  - *You can change the color and plot style by adding a RooFit::LineColor(int color) argument to the plotOn() call. See tutorial macro rf107 for details*

Exercise 3

- Macro ex03_roostats_plr_interval.C
  - Opens the model.root file, retrieves the workspace and from that the ModelConfig object (unique statistical problem definition) and the observed data
  - Instantiates a RooStats Profile Likelihood Ratio calculator and lets it calculate the profile likelihood ratio interval on the above problem
  - Reports the results on the command line
- Questions & explorations
  - How does the profile likelihood calculator result compare to your manual investigation of the likelihood ratio curve in Ex01?
  - Calculate the same type of interval at difference confidence levels, e.g. 65% and 95%.

Exercise 4

- Macro ex04_roostats_bayes_interval.C
  - Opens the model.root file, retrieves the workspace and from that the ModelConfig object (unique statistical problem definition) and the observed data
  - Instantiates a RooStats Bayesian calculator and lets it calculate the Bayesian credible interval on the above problem
  - Reports the results on the command line
- Questions & explorations
  - How does the Bayesian 90% interval compare with flat prior to the Frequentist Profile Likelihood Ratio interval?
  - Run the macro for some different interval shapes: e.g. upper limit, or shorttest interval
  - Explore what happens for various choices of priors, e.g. 1/sqrt(mu), or flat prior for mu>0 only?

  Note that this Bayesian calculator uses a simple numeric integration engine, it may emit warnings about numeric precision if pushed to perform complex integrations.

Exercise 5

- Macro ex05_build_PoissonPoisson.C
  - Constructs the classic statistical model known as 'on/off':
    A Poisson model for the signal region measuring mu*S+B
    A Poisson model for the control region measuring tau*B
  - Here tau is a scale factor for the size of the control region, e.g.
    if tau=3 then a count of 30 in the control region will predict
    a background rate of 10 in the signal region with a relative
    error of 10/sqrt(30).
  - Constructs a RooStats ModelConfig and saves everything to a workspace
    on file.
- Questions and explorations
  - Do you understand the observed uncertainty on the fitted background
    rate in the SR? (In terms of the given numbers, N_CR=200, tau=10,
    N_SR=25)?
  - Run the RooStats PLR and Bayesian calculators on this model
  - *Can you reproduce the 'standard candle' result N_SR=178, N_CR=100,*
    tau=1 of in the course and confirm that it's signficance is exactly 5 sigma?
    To do so, plot a scan of the profile likelihood ratio of this problem (see Ex01
    on how to do that), and look at the value of the PLR for mu=0

Exercise 6

- Macro ex06_build_PoissonPoissonGlobs.C
  - Builds the same probability model as ex05, but uses the notion of global observables in the construction
  - Global observables are purely technical construction that 'hard-wire' certain observables in the model itself, rather than having them appear in the dataset, e.g.

    $\text{Data}(N_{SR}=25, M_{CR}=200) \leftrightarrow P(N_{SR}, N_{CR})$
    $\text{Data}(N_{SR}=25) \leftrightarrow P(N_{SR}, N_{CR}=200)$

  - While mathematically equivalent, global observables, are often used for convenience so they don't need to be carried in all datasets.
  - This is particularly true for models with many unit-Gaussian subsidiary measurements where all global observable values are always zero
- Questions and explorations
  - Look at macro ex06 and see how it differs from macro ex05 in how the problem is formulated

Exercise 7

- Macro ex07_build_PoissonGaussGlobs.C
  - This macro builds a variant of the model of ex06 – it changes the control region model that measured the background B from a Poisson to a Gaussian.
  - It also maps the physics effect (the magnitude of the uncertainty) in a response function encoded in the signal region probability model in terms of a nuisance parameter alpha, and reduces the subsidiary measurement of alpha to a unit Gaussian.
  - Writes probability model and RooStats ModelConfig to output file
- Questions and explorations
  - Identify the piece of code that encodes the response function of the systematic uncertainty.
  - Modify the response function such that magnitude of the systematic uncertainty is doubled and rerun
  - Analyze the model of ex07 with the RooStats PLR and Bayesian calculators

Exercise 8

- Macro ex08_roostats_cls_limit.C
  - Implements the RooStats hypothesis test inverter limit calculator. This is the most general limit calculator.
  - In this macro the calculator is configured to use the profile likelihood ratio test statistic for the limit calculation, *and to assume its known asymptotic distributions*
  - In the final limit calculation the CLS technique is enabled, which is designed to always return non-empty intervals in the range [0,X]
- Questions and explorations
  - Run the calculator first on several of the models built so far (Single Poisson, Two Poissons, Poisson/Gaussian)
  - Understand the working of the calculator by identifying its pieces
    1) An explicit alternative hypothesis is constructed from the workspace (ModelConfig for b-only hypothesis). This is needed for the calculation of CLS and for the calculation of expected limits under the B-only hypothesis
    2) Set up an (asymptotic) calculator that can calculate the p-value of the data under both hypothesis (B-only, and S+B(for a given value of mu)
    3) Configure the Inverter, the tool that will vary mu in such a way that CLS == p-value(S+B)/(1-p-value(B) corresponds to the desired confidence level. The value of mu for which this is true is then reported as the CLS upper limit

Exercise 9

- Macro ex09_roostats_cls_limit_toys.C
  - Implements the RooStats hypothesis test inverter limit calculator. This is the most general limit calculator.
  - In this macro the calculator is configured to use the profile likelihood ratio test statistic for the limit calculation, but no assumptions are made on the distributions, instead these are calculated from distribution of ensembles of toy MC data
  - In the final limit calculation the CLS technique is enabled, which is designed to always return non-empty intervals in the range [0,X]
- Questions and explorations
  - Run the calculator first on one of the model (NB: this calculation takes significantly more time than the asymptotic one0
  - Understand the working of the calculator by identifying its pieces
    1) An explicit alternative hypothesis is constructed from the workspace (ModelConfig for b-only hypothesis). This is needed for the calculation of CLS and for the calculation of expected limits under the B-only hypothesis
    2) Set up an a generic calculator for p-values from test statistic distribution that can calculate the p-value of the data under both hypothesis (B-only, and S+B(for a given value of mu). This generic Frequentist calculator will use ensembles of toy data to obtain the distributions
    3) Configure the generic calculator to use the pne-sided Profile Likelihood Ratio test statistic
    4) Configure the Inverter, the tool that will vary mu in such a way that CLS == p-value(S+B)/(1-p-value(B) corresponds to the desired confidence level. The value of mu for which this is true is then reported as the CLS upper limit