Introduction to the HistFitter framework

Max Baak (CERN), Geert-Jan Besjes (Nijmegen/Nikhef), Jeanette Lorenz (LMU/Excellence Cluster Universe), Sophio Pataraia (Bergische Universitaet Wuppertal)

+ many other people

30 March 2015

Overview

- HistFitter overview
  - Introduction & strategy
- HistFitter tutorial
  - Running a fit & visualization
  - Calculating limits

HistFitter introduction

Introduction

- **HistFitter** is a statistical tool/framework used in (almost) all ATLAS SUSY analyses since 2012 for fitting, interpretation and presentation of fit results + few Exotics and Higgs examples

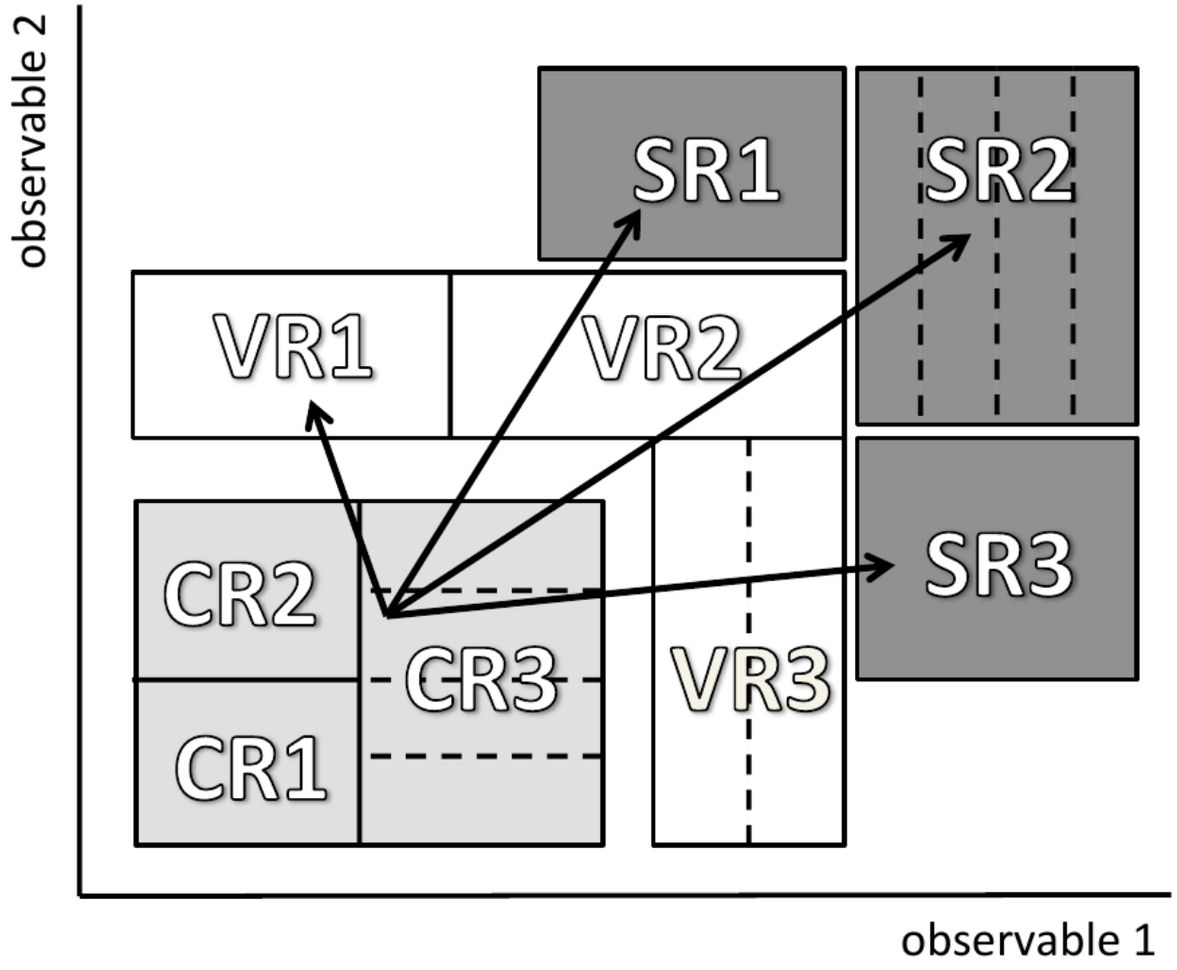
- **HistFitter** is:
  - built on top of RooFit/HistFactory and RooStats
  - consists of Python part for configuration and C++ part for CPU-intensive calculations

- Why HistFitter?
  - **HistFitter** extends RooFit/HistFactory and RooStats in four key areas:
    - Programmable framework: performing complete analysis (steps 0-4 from last talk) from a simple configuration file
    - Analysis strategy: common physics analysis strategy concepts, such as control/signal/validation regions, woven into the fabric of HistFitter design
    - Bookkeeping: can keep track of numerous data models, from histogram production until final statistical tests → handy when working with large collections of signal hypotheses (*signal grids*)
    - Presentation and interpretation: multiple methods are provided to determine statistical significance of signal hypotheses, and produce publication-quality tables and plot summarizing the fit results (*step 4*)

Data analysis strategy

- Particle physics analyze large data samples for measurements of known physics or to search for new physics
- Data interpretation relies on using external - simulation, Monte Carlo (MC) - predictions for backgrounds and signal
- HistFitter configures and builds parametric models from these predictions

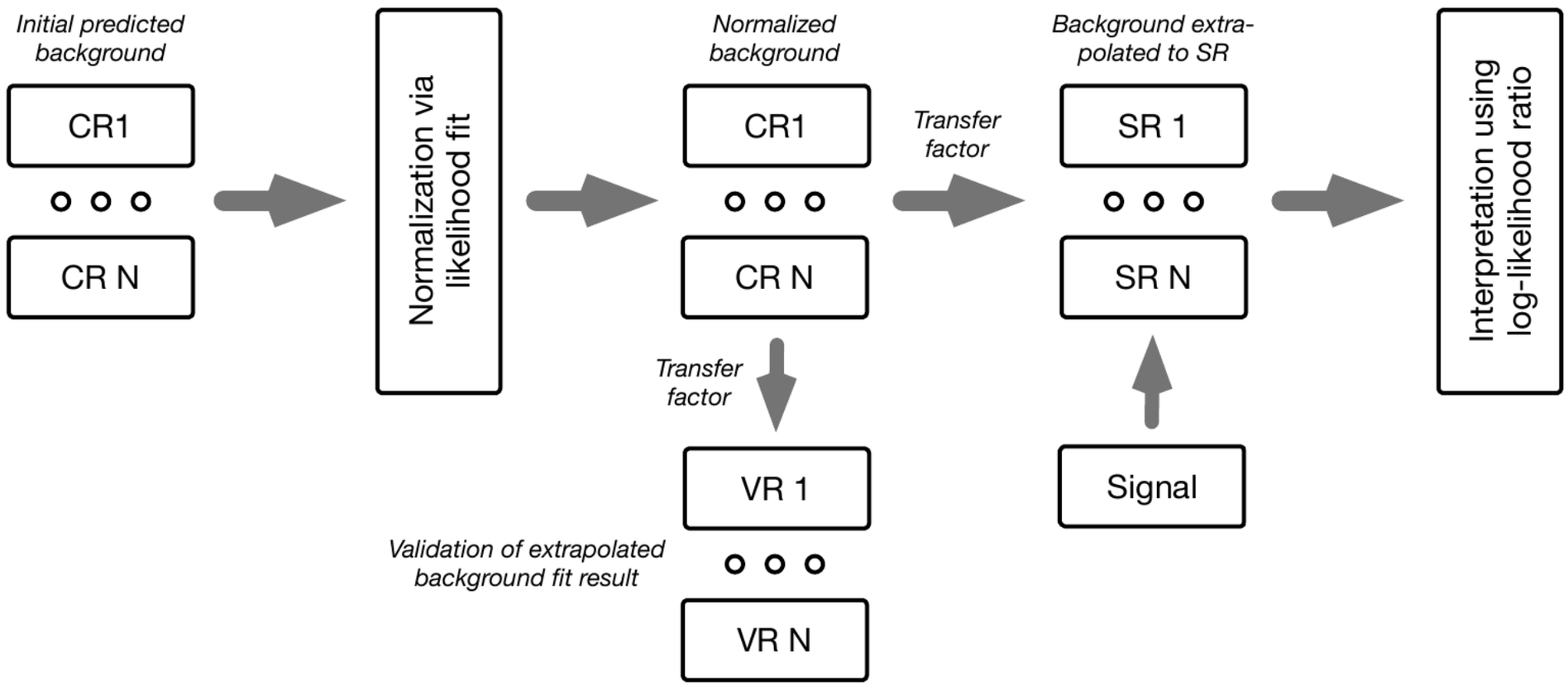
- Typically one defines several phase space regions to study a specific phenomenon
  - Definition depends on the purpose:
    - **Signal region:** signal-rich region (SR)
    - **Control region:** background-rich region (CR), fit simulated backgrounds to data
    - **Validation region:** validation of extrapolation (VR)
- Concepts of CR/SR/VR woven into the fabric of HistFitter



Analysis strategy flow

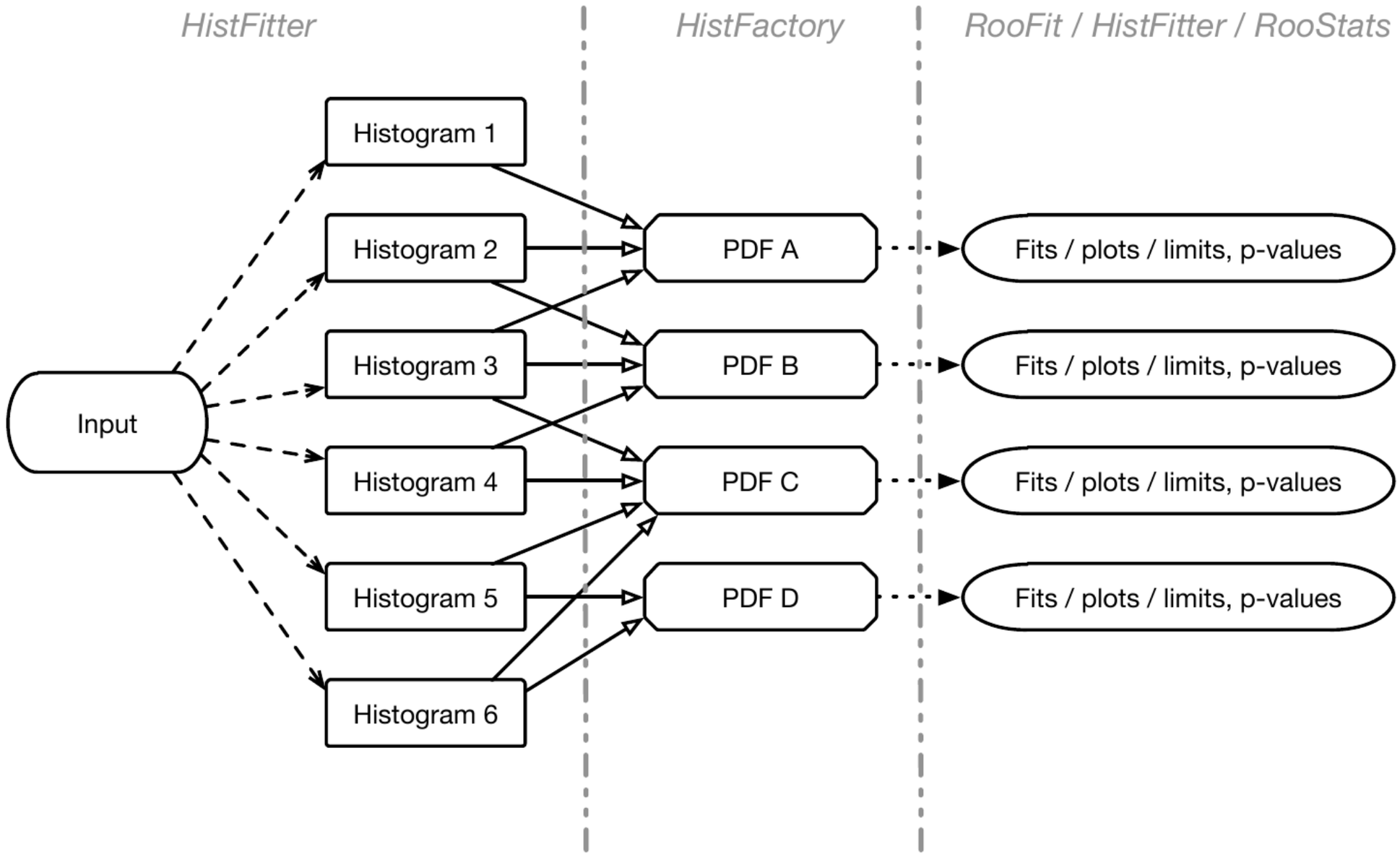
- Each CR/VR/SR modeled by a separate PDF, combined in a simultaneous fit
- Parameters shared in all regions → consistent background/signal prediction and systematics
  - Sharing user-defined

- Analysis flow:
  - Backgrounds normalized to data in a fit of control regions
  - Extrapolate to validation/signal regions using transfer factors (ratio of events between CR and SR/VR)
  - If good agreement in VR, unblind the SR
  - If no excess, add signal prediction and interpret/set limits



Processing sequence

- Based on user-defined configuration file, processing sequence of HistFitter split in three stages



step-0
Histogram production

step-1
PDF construction
Workspace building

step-2/3
Analysis of models

Model construction

- Models constructed using HistFactory from input histograms
- General form of the constructed likelihood:

$$L(\boldsymbol{n}, \boldsymbol{\theta}^0 | \mu_{\mathrm{sig}}, \boldsymbol{b}, \boldsymbol{\theta}) \quad = \quad P_{\mathrm{SR}} \times P_{\mathrm{CR}} \times C_{\mathrm{syst}}$$

  - P = Poisson measurements of number of observed events in CR/SR (VR)
  - C = Constraint terms for systematic uncertainties, auxiliary measurements
  - Likelihood depends on number of observed events in all regions (n), predictions for various background processes (b), the nuisance parameter (θ) parametrizing the systematic uncertainties with their central value (θ^0) and signal strength (μ_{SIG})

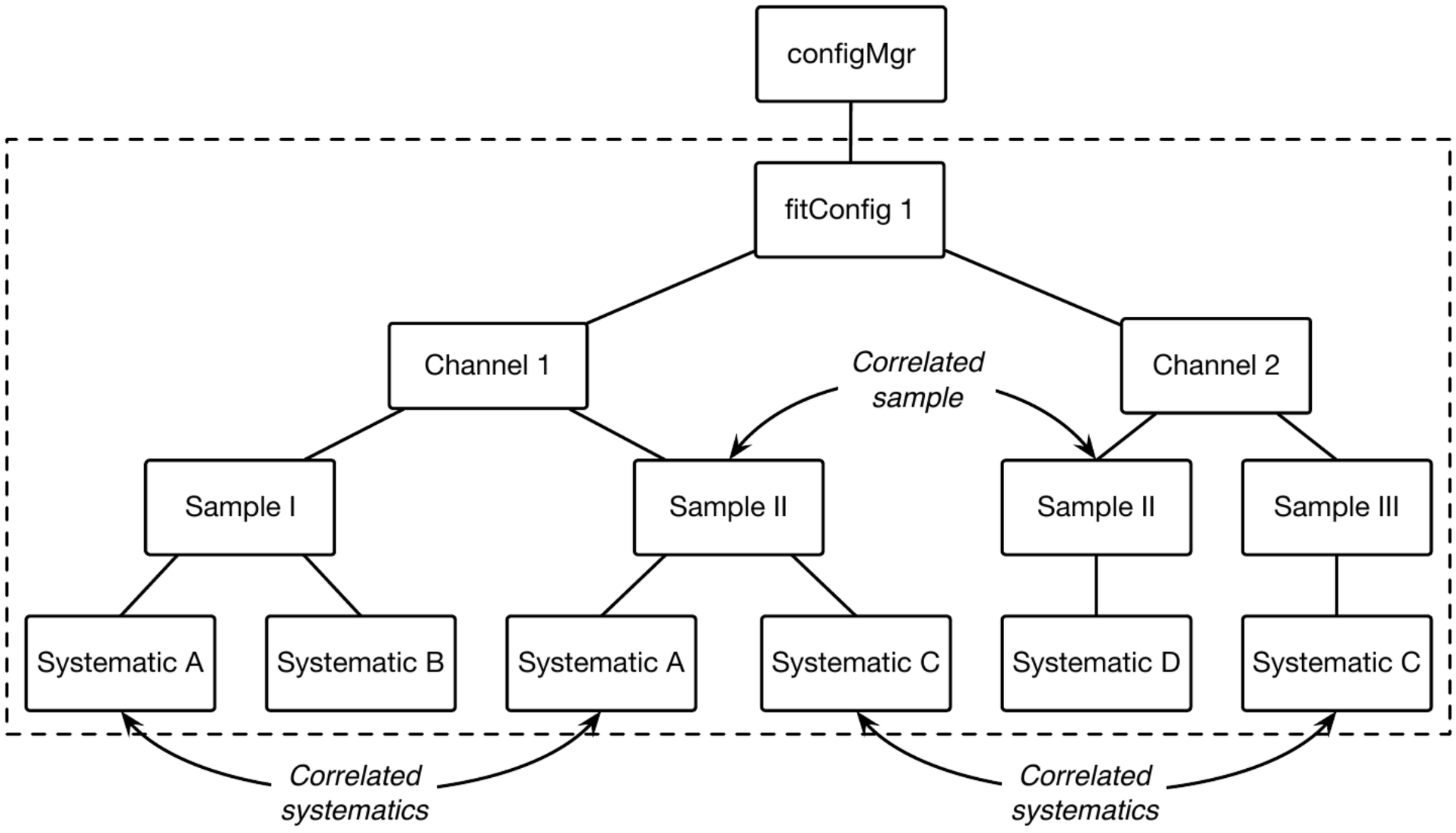
- Likelihood has multiple building blocks:
  - Control/validation/signal regions: called `channel` in HistFitter (HistFactory)
  - Signal and background processes: called `sample` in HistFitter (HistFactory)
  - Uncertainties: called `systematic` in HistFitter (HistFactory)
    - Including statistical/theory/experimental uncertainties

- HistFitter is designed to build and manipulate PDFs of nearly arbitrary complexity
- Bookkeeping/configuration machinery realized through a user-defined Python configuration file

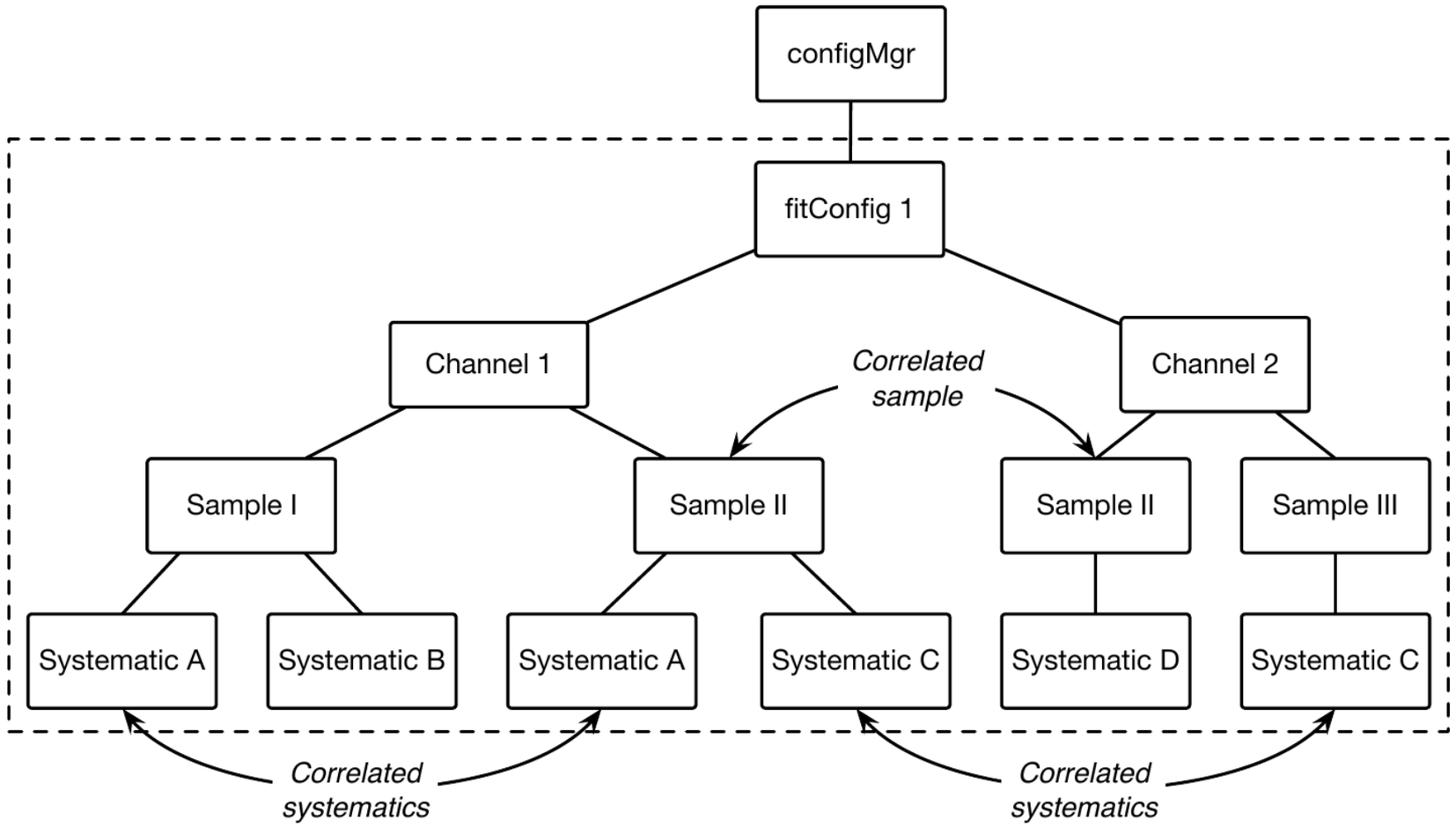
- Configuration manager (`configManager`) highest level (singleton) object in Python and C++
- Manages `fitConfig` objects that contain PDF and meta-data

Fit configuration

- `fitConfig` objects summarize `channels, samples` and `systematics` together with corresponding input histograms

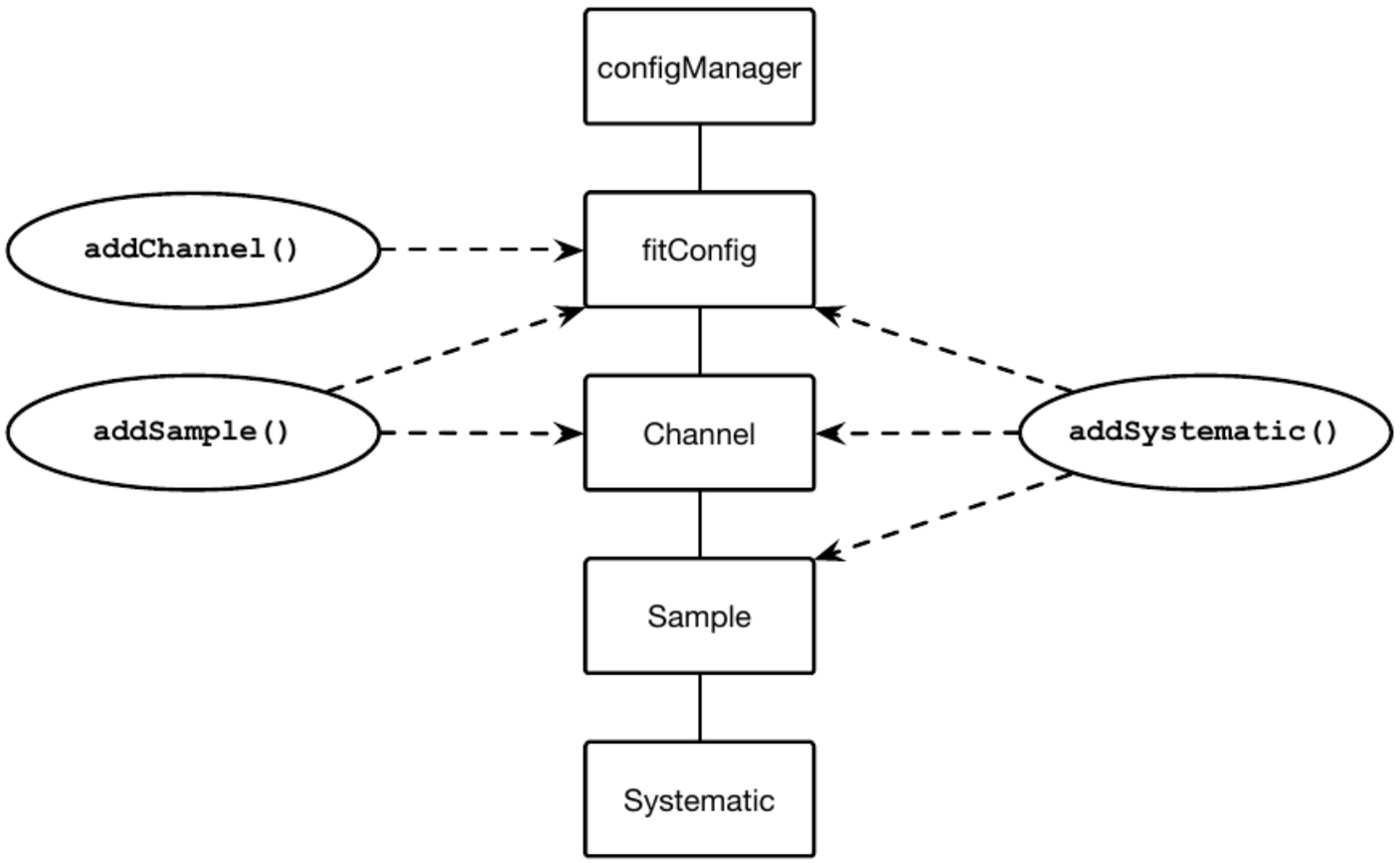


Fit configuration properties



- `fitConfig`: can be cloned/extended (see next slide)
- `channels:` either single-bin or multi-bin (shape), property as CR/VR/SR
- `samples:` input from TTree, TH1 or raw (hard-coded) floats, correlated between channels
- `systematics`: provided as ±1σ variation of nominal histogram; input from TTree, TH1 or raw floats; can be correlated between samples and/or channels; many types available extended from HistFactory base types (see later); trickle-down mechanism

Trickle-down mechanism



`Channels` are added to a `fitConfig`.

Samples can be added to either a `fitConfig` or a `channel`. If adding to `fitConfig` also added to all depending `channels`.

Similar for **systematics**. Can be added to `fitConfig` and then propagated to all `channels` and `samples`. Or added to a `channel` and then propagated to all depending `samples`. Or just added to a specific `sample`.

⇒ A complicated PDF can be described by few lines of code.

Common fit strategies

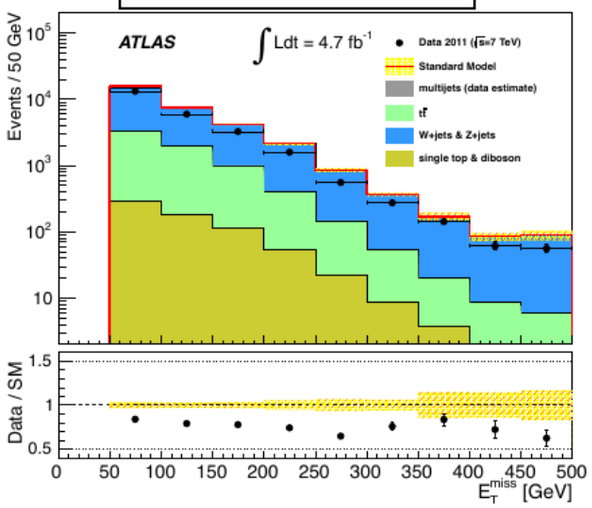
- **Background-only fit**: estimate background yields in validation/signal regions; including *only* CRs in the fit to data; no signal component included in fit configuration
- **Model-dependent signal fit**: set exclusion limit on a specific signal model; possible use of multi-binned (or multi-SR) shape fit for a robust signal estimation - aka ***exclusion fit***
- **Model-*in*dependent signal fit**: to obtain model-independent upper limits on number of BSM events beyond background prediction; only usable with one single-bin SR (otherwise not model-independent) - aka ***discovery fit***

Fit setup	*Background-only fit*	*Model-dependent signal fit*	*Model-independent signal fit*
Samples used	backgrounds	backgrounds + signal	backgrounds + dummy signal
Fit regions	CR(s)	CR(s) + SR(s)	CR(s) + SR

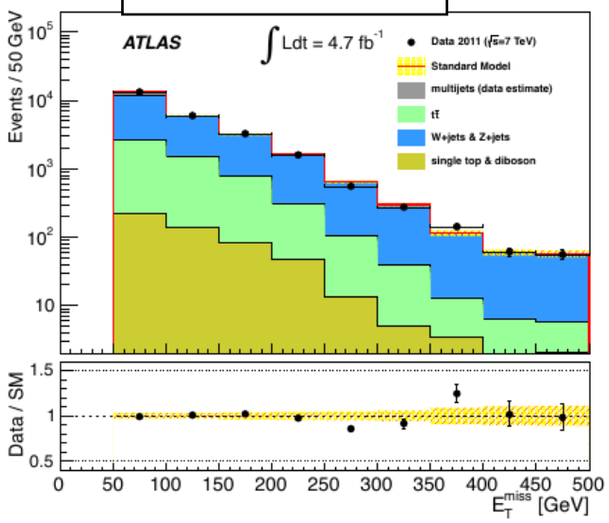
Presentation of results

- HistFitter includes a collection of tools (scripts/functions) to present/understand fit results

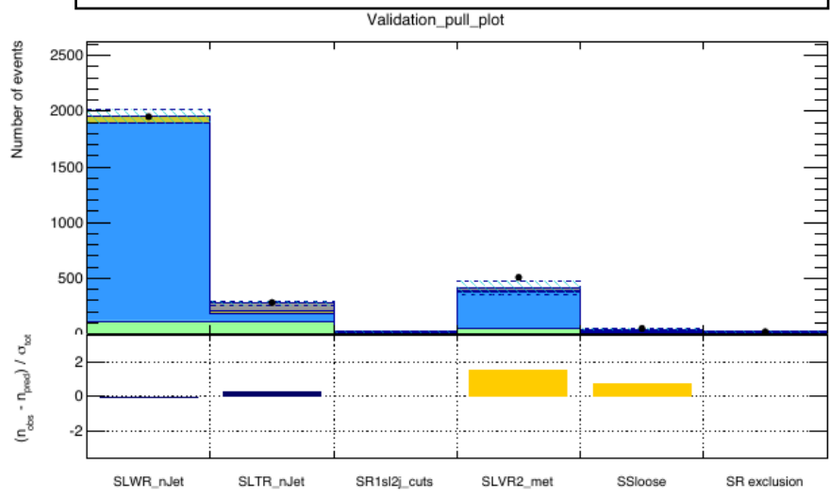
Before Fit



After Fit



After Fit VRs pull plot



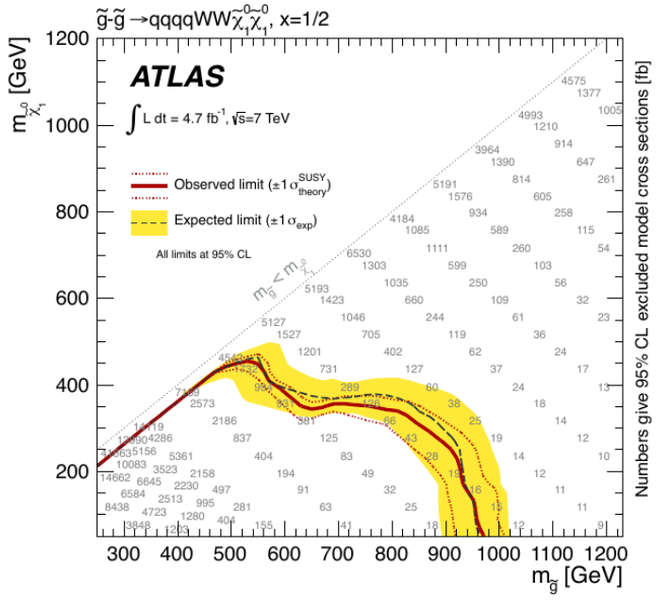
Yields Table

Signal Region	SR1	SR2
Observed events	16	19
Fitted bkg events	19.54 ± 3.93	20.47 ± 5.14
Fitted Top events	4.02 ± 0.96	4.32 ± 1.04
Fitted V+jets events	9.89 ± 1.86	10.47 ± 1.91
Fitted other background events	1.14 ± 0.15	1.19 ± 0.16
Fitted QCD events	4.49 ± 2.72	4.49 ± 4.24
MC exp. SM events	24.85	26.32
MC exp. Top events	8.42	9.11
MC exp. V+jets events	10.82	11.55
MC exp. other background events	1.13	1.17
Data-driven exp. QCD events	4.49	4.49

Systematics Table

Uncertainty of channel	SR1	SR2
Total background expectation	19.54	20.47
Total statistical ($\sqrt{N_{\text{exp}}}$)	$\pm$4.42	$\pm$4.52
Total background systematic	$\pm$3.93 [20.14%]	$\pm$5.14 [25.09%]
QCD background	$\pm$2.66	$\pm$4.20
Statistical uncertainties	$\pm$2.54	$\pm$1.86
Jet Energy Scale	$\pm$1.15	$\pm$1.17
Top yield	$\pm$0.82	$\pm$0.88
Renormalization scale (Top)	$\pm$0.34	$\pm$0.39
V+jets yields	$\pm$0.28	$\pm$0.29
Renormalization scale (V+jets)	$\pm$0.14	$\pm$0.03

Exclusion contour with upper limits



Model-independent upper limits

Signal channel	$\langle\sigma_{\text{vis}}\rangle^{95}_{\text{obs}}$[fb]	S^{95}_{obs}	S^{95}_{exp}	$p(s=0)$
SR3b	0.19	3.9	$4.4^{+1.7}_{-0.6}$	0.50
SR0b	0.80	16.3	$8.9^{+3.6}_{-2.0}$	0.03

HistFitter & documentation

- HistFitter paper on arXiv: http://arxiv.org/abs/1410.1280
- HistFitter webpage with doxgen documentation: http://cern.ch/histfitter
- ACAT 2014 talk on HistFitter: https://indico.cern.ch/event/258092/session/8/contribution/39

HistFitter tutorial

Running HistFitter

- `HistFitter.py <options> <configuration_file>`

- **-t:** Create histograms in all regions used for all backgrounds, signal, data from TTrees
- **-w**: Build workspaces from histograms
- **-f:** Fit
- **-D**: various drawing options, to be discussed later
- **-L:** log level {VERBOSE,DEBUG,INFO,WARNING,ERROR,FATAL,ALWAYS}
- **-m PARAM**: run Minos for asymmetric error calculation
  - optionally give parameter names comma separated; for all parameters use ‘ALL’ or ‘all’
- **-l**: Calculate upper limit
- **-p**: Calculate the CLs value for a specific signal model (for exclusion)
- **-i**: interactive mode, keeps you in python command line, but shows plots on your screen

- To see all options run: `HistFitter.py –help`

- **In this tutorial we will also study more advanced options**

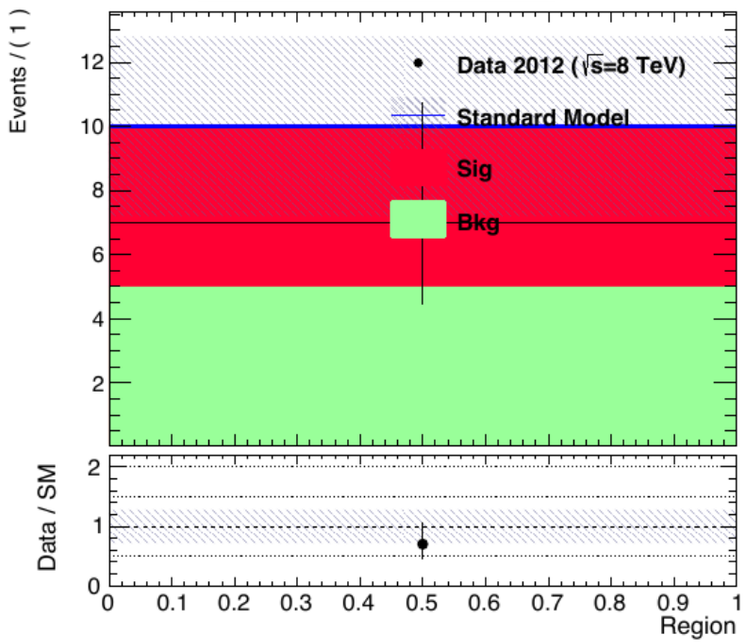
Simple example

- Simple example with one region with one bin:

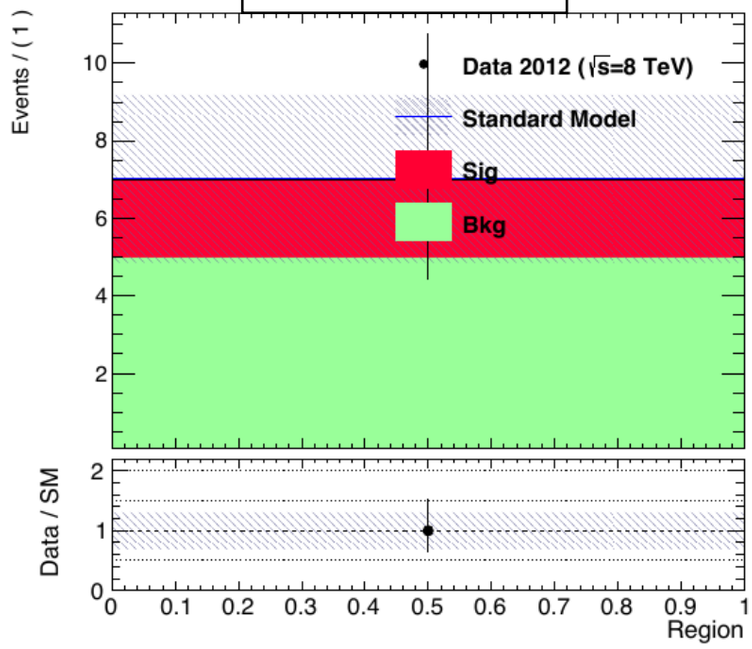
```
HistFitter.py -w -f -D "before,after,corrMatrix" -i
analysis/tutorial/MyUserAnalysis.py
```

- Creates the workspace
- Runs the fit
- Plots before/after fit regions and correlation matrix
- Keeps you in interactive mode

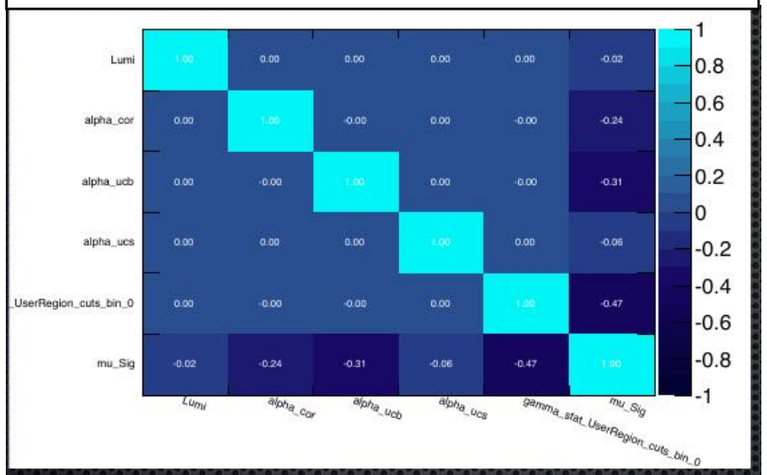
Before Fit



After Fit



correlations matrix



-D corrMatrix

Config file explained - I

- Define a configManager and setup a `fitConfig` ana named SPlusB
- `from configManager import configMgr`
  `ana = configMgr.addFitConfig("SPlusB")`

- Add one channel/region to the `fitConfig`
- `chan = ana.addChannel("cuts",["UserRegion"],1,0.5,1.5)`

- One defines the region/channel in cutsDict (as one would in ROOT for TTree call)
- Here include all:
- `configMgr.cutsDict["UserRegion"] = "1."`

- Channels can also be binned (shape-fit)
- `chan = ana.addChannel("myObs", ["mySelection"], nBins, varLow, varHigh)`

Config file explained - II

- Define samples: bkgSample, sigSample and dataSample

- # Define samples

```
bkgSample = Sample("Bkg",kGreen-9) # define a background sample with color KGreen-9 if plotting
bkgSample.setStatConfig(True) #This sample gets statistical uncertainties
bkgSample.buildHisto([nbkg],"UserRegion","cuts") #Build histograms from numbers defined by the user
bkgSample.buildStatErrors([nbkgErr],"UserRegion","cuts")

sigSample = Sample("Sig",kPink) #A signal sample with color kPink
sigSample.setNormFactor("mu_Sig",1.,0.,100.) # This samples receives a normalization parameter
sigSample.setStatConfig(True) #This sample gets statistical uncertainties
sigSample.setNormByTheory() # and uncertainties due to the luminosity are added
sigSample.buildHisto([nsig],"UserRegion","cuts")
sigSample.buildStatErrors([nsigErr],"UserRegion","cuts")

dataSample = Sample("Data",kBlack) #Data sample
dataSample.setData()
dataSample.buildHisto([ndata],"UserRegion","cuts")

# add all samples to the fitconfig object and thus to all channels
ana.addSamples([bkgSample,sigSample,dataSample])
```

Config file explained - III

- Add systematics to signal/background samples
- Correlating systematics happens by giving them the same name

- # Set uncorrelated systematics for bkg and signal (1 +- relative uncertainties)

```
ucb = Systematic("ucb", configMgr.weights, 1.2,0.8, "user","userOverallSys")
ucs = Systematic("ucs", configMgr.weights, 1.1,0.9, "user","userOverallSys")
```

correlated systematic between background and signal (1 +- relative uncertainties)

```
corb = Systematic("cor",configMgr.weights, [1.1],[0.9], "user","userHistoSys")
cors = Systematic("cor",configMgr.weights, [1.15],[0.85],
"user","userHistoSys")
```

```
bkgSample.addSystematic(corb)
bkgSample.addSystematic(ucb)
sigSample.addSystematic(cors)
sigSample.addSystematic(ucs)
```

Table production

- **YieldsTable.py** produces customizable tables of yields before/after fit
- Example: `YieldsTable.py -s Top,WZ,BG,QCD -c SLWR_nJet,SLTR_nJet -w results/MyConfigExample/BkgOnly_combined_NormalMeasurement_model_afterFit.root -o MyYieldsTable.tex`

table.results.yields channel	SLWR_nJet	SLTR_nJet	SR1sl2j	SS_metmeff2Jet
Observed events	1794	269	25	26
Fitted bkg events	1800.73 ± 39.91	262.45 ± 11.47	28.53 ± 5.26	31.74 ± 8.50
Fitted Top events	117.20 ± 11.42	113.20 ± 12.53	6.17 ± 1.12	6.65 ± 1.26
Fitted WZ events	1629.37 ± 42.19	69.75 ± 6.63	13.95 ± 2.03	14.57 ± 1.98
Fitted BG events	43.49 ± 1.90	23.19 ± 1.94	0.96 ± 0.32	1.00 ± 0.32
Fitted QCD events	10.64 ± 0.51	56.30 ± 13.65	7.44 ± 3.75	9.52 ± 7.54
MC exp. SM events	1921.26	261.96	32.04	35.35
MC exp. Top events	165.16	153.98	8.75	9.38
MC exp. WZ events	1647.04	66.30	15.26	15.82
MC exp. BG events	40.96	25.03	0.59	0.63
data-driven exp. QCD events	68.06	16.64	7.44	9.52

- **SysTable.py** produces customizable tables of systematic breakdown per region (or sample)
- Example: `SysTable.py -w results/MyConfigExample/BkgOnly_combined_NormalMeasurement_model_afterFit.root -c SR1sl2j -o systable_SR1sl2j.tex`

Uncertainty of channel	SR1sl2j
Total background expectation	28.53
Total statistical ($\sqrt{N_{\text{exp}}}$)	±5.34
Total background systematic	±5.26 [18.43%]
gamma_stat_SR1sl2j_cuts_bin_0	±3.63
alpha_QCDNorm_SR1sl2j	±3.63
alpha_JES	±0.93
mu_Top	±0.65
alpha_KtScaleTop	±0.52
alpha_KtScaleWZ	±0.37
mu_WZ	±0.36

Signal model hypothesis test

- Once you have unblinded your SR, one can calculate the CLs/p-value on specific signal models using the exclusion fit (aka model-dependent fit setup)
- As simple in HistFitter as calling:

```
HistFitter.py -p analysis/tutorial/MyUserAnalysis.py
```

- Will calculate:
  - CLs_observed = taking N observed events as data in all regions
  - CLs_expected = taking N expected events as data in all regions
  - CLs_expected ±1sigma experimental uncertainty = N expected as data, ±1sigma fit results
    - yellow band next slide
  - CLs_observed ±1sigma signal theory uncertainty = N observed as data, ±1sigma signal theory
    - need to set the name of the signal theory uncertainty systematic as `Systematic("SigXSec", ...)`
    - red-dotted lines next slide

- Setting calculator and test statistic type can be set in configManager (see backup):

```
## setting the parameters of the hypothesis test
#configMgr.nTOYs=5000
configMgr.calculatorType=2 # 2=asymptotic calculator, 0=frequentist calculator
configMgr.testStatType=3   # 3=one-sided profile likelihood test statistic (LHC default)
configMgr.nPoints=20       # number of values scanned of signal-strength for upper-limit
determination of signal strength.
```

- Result of ‘-p’ stored in a ROOT file with ‘hypotest’ in the name:

```
results/MySimpleChannelAnalysis_fixSigXSecNominal_hypotest.root
```

Contour plot explained

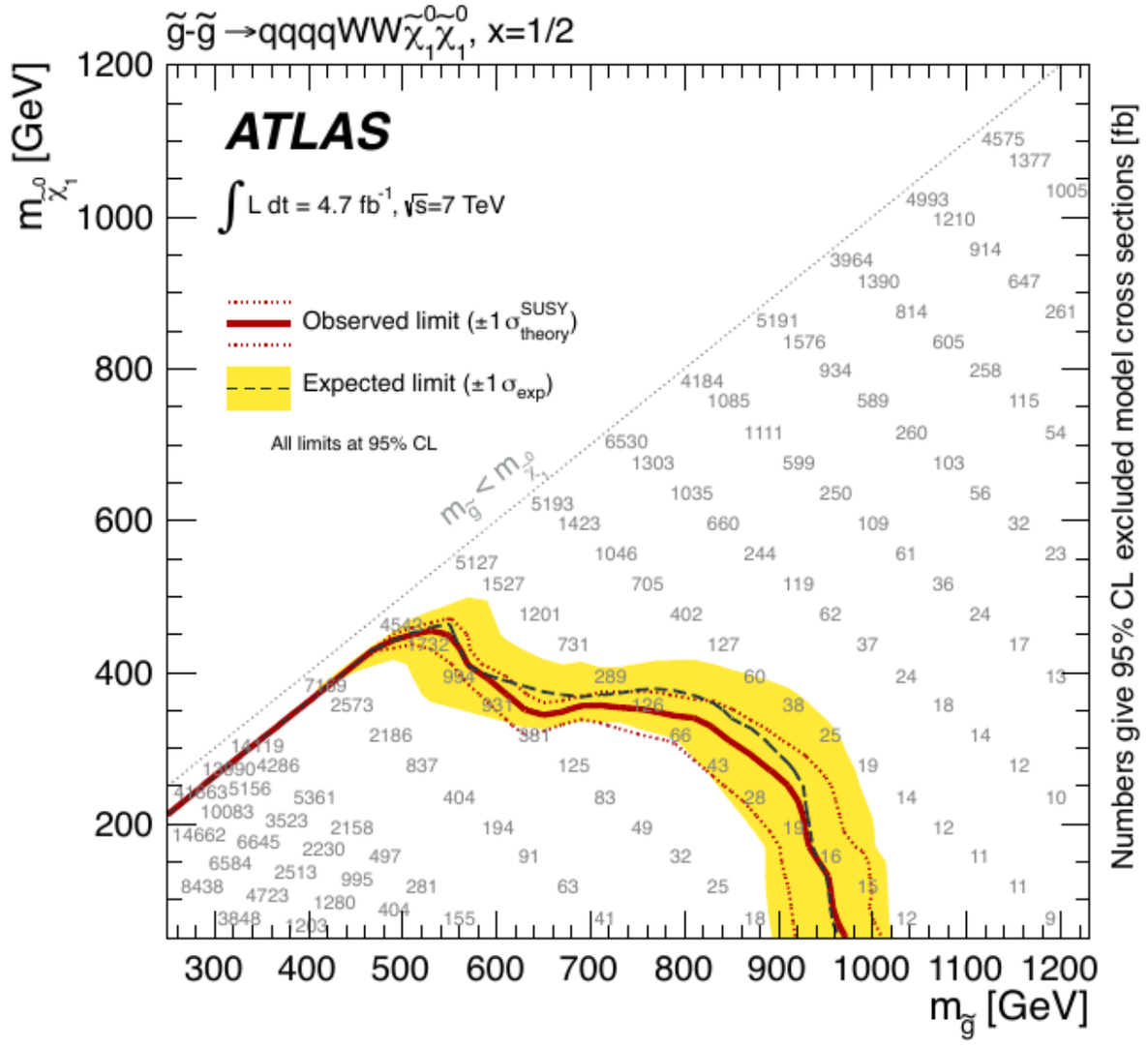
Description of limit lines

The model limits should be computed using the HistFitter package. We present the following limits:

1. **Observed limit** (thick solid dark-red line): all uncertainties are included in the fit as nuisance parameters, with the exception of the theoretical signal uncertainties (PDF, scales).
2. **Expected limit** (less thick long-dashed dark-blue line): all uncertainties are included in the fit as nuisance parameters, with the exception of the theoretical signal uncertainties (PDF, scales).

We present the following uncertainty bands:

- **±1σ lines around observed limit** (1) with style "thin dark-red dotted": re-run limit calculation (1) while increasing or decreasing the signal cross section by the theoretical signal uncertainties (PDF, scales).
- **±1σ band around expected limit** (2) with style "yellow band": the band contours are the ±1σ results of the fit (2).

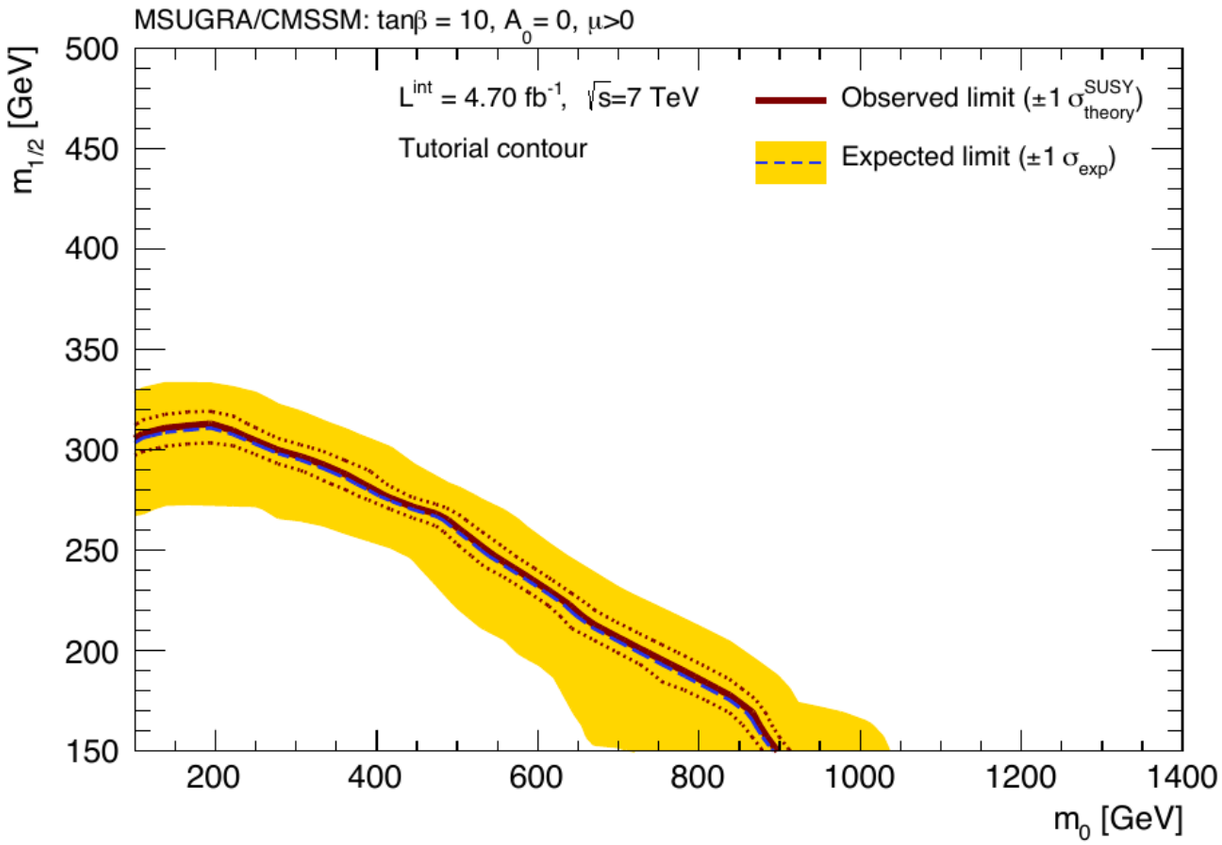


Contour plot production

- Typically a grid of signal model points with varying signal parameters (m_H or m_{gluino}) get processed to produce an exclusion contour
- Five steps to produce (Part 5 of tutorial):

1. run hypothesis tests over all grid points (results saved in multiple *hypotest* files)
2. merge all the output root files into one using `hadd` (if stored in a separate files)
3. transform this set of hypothesis tests into a plain-text file: `makelistfiles.C`
4. create TH2D(s) from the ascii data in this list file: `makecontourhists.C`
5. plot TH2D(s) to draw contour lines and cosmetics: `makecontourplots.C`

- at the requested CLs level, typically 95% CL, CLs<0.05



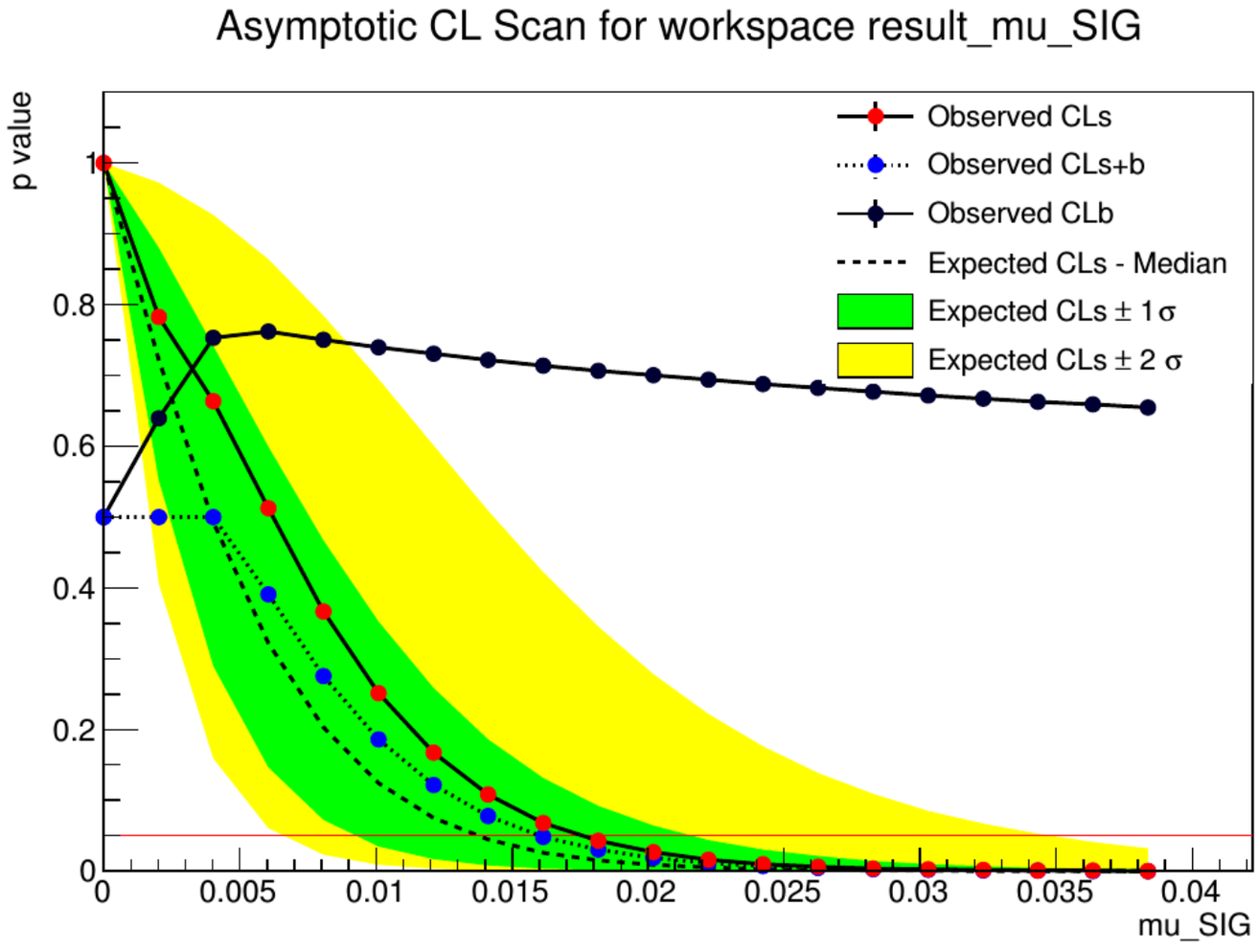
Signal strength upper limit

- Once you have unblinded your SR, one can set upper limits on specific signal models using the exclusion fit (aka model-dependent fit setup)
- As simple in HistFitter as calling:

```
HistFitter.py -l analysis/tutorial/MyUserAnalysis.py
```

  - Technicalities similar to '-p'

- Hypothesis test ***inversion***:
  - find the value of mu_SIG for which CLs below 0.05 (or other required value)
    - instead of calculating the p-value for the specific signal
  - run the hypothesis test for increasing values of signal strength mu_SIG
    - scan range determined automatically
    - upper limit on cross section = nominal cross section × upper limit on signal strength (grey numbers in contour plots, run for each signal grid point)



Model-independent upper limit

- Calculate the upper limit on the number of BSM physics events that we exclude in our SR
  - Typically used by theorists to check their favorite BSM model, that we have not looked at
- Requires the model-independent fit setup - aka discovery fit
  - ‘dummy signal’ = exactly one event in signal region (none in CRs)
  - upper limit on this ‘dummy signal’ = upper limit on BSM number of events
- Use the **UpperLimitTable.py** script:

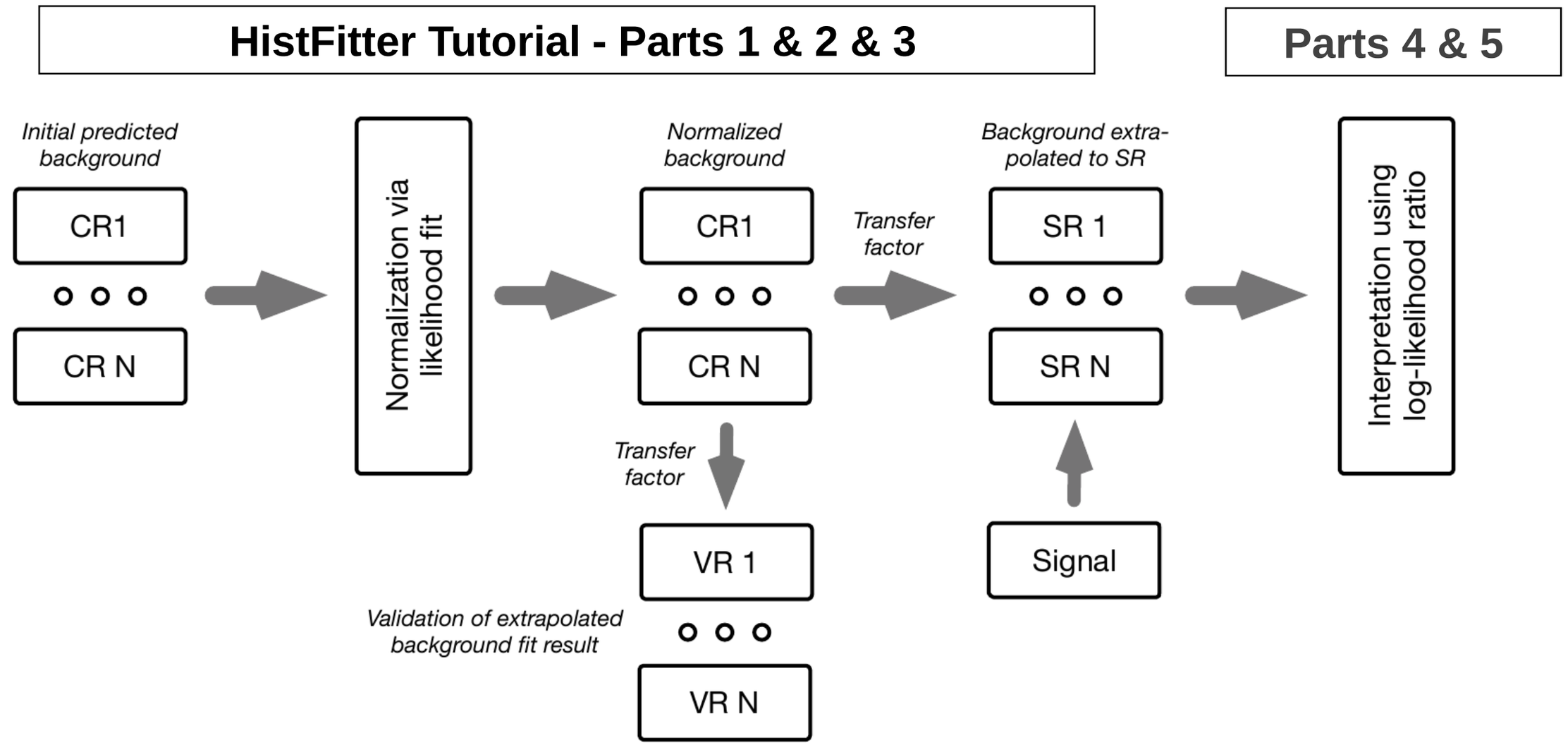
```
UpperLimitTable.py -c SS -w
results/MyUpperLimitAnalysis_SS/SPlusB_combined_NormalMeasurement_model.root -
l 4.713 -n 1000
```

- Results in LaTeX table:

Signal channel	$\langle \epsilon\sigma \rangle^{95}_{\text{obs}}$[fb]	S^{95}_{obs}	S^{95}_{exp}	CL_B	$p(s=0)$
SS	1.73	8.2	$6.1^{+2.3}_{-1.3}$	0.80	0.21

  - ⟨σvis⟩95_obs : 95% CL upper limits on the visible cross section obs
  - S95_obs :95% CL upper limits on the number of signal events obs
  - S95_exp : 95% CL upper limit on the number of signal events, given the expected number (and ±1σ excursions on the expectation) of background events
  - CLB: the confidence level observed for the background-only hypothesis
  - p(s = 0): discovery p-value - the probability, capped at 0.5, that a background-only experiment is more signal-like than the observed number of events in a signal region

HistFitter - tutorial



- **This morning:** Get started with HistFitter, Part 1 & 2 & 3
- **Afternoon**: Part 4 & 5

- **Tommorrow:** Additional and more advanced features + discussion session to answer your questions!

HistFitter tutorial start up for ATLAS

- Setup described on the HistFitter Tutorial twiki:
  https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/HistFitterTutorial#Part_0_Setting_up_HistFitter

- Check out the package

```
svn co
svn+ssh://$USER@svn.cern.ch/reps/atlasphys/Physics/SUSY/Analyses/HistFitter/tags/HistFitter-00-00-46 HistFitterTutorial
cd HistFitterTutorial
```

- Setup ROOT, setup HistFitter, compile the package, link the files needed for the tutorial

```
setupATLAS
localSetupROOT
source setup.sh
cd src
make
cd ..
ln -s $ALRB_TutorialData/samples samples
```

HistFitter tutorial start up for non-ATLAS

- Unfortunately a little bit more complicated as we cannot use ATLAS protected webpages or software.

- **Thus:**

  A public version is available on the HistFitter webpage:
  http://histfitter.web.cern.ch/histfitter/Software/Install/index.html

  We use HistFitter-2.0.tar.gz for this tutorial.

- **Installation instructions:**
  - Untar the HistFitter package
  - Setup ROOT (if not already done)
  - Go the HistFitter directory cd HistFitter-2.0
  - Run the HistFitter setup script source setup.sh
  - Go to the src/ directory and compile the C++ side of HistFitter cd src && make
  - Go back to the main HistFitter directory

There is also the possibility to use the Root version from afs. Check setup_afs.sh for this. (This is described at the beginning of the tutorial.)

Input data for non-ATLAS

- **Unfortunately, non-ATLAS members are not allowed to use the original input data we used when creating this tutorial, as it was derived at some point from some ATLAS Monte Carlo (and we have thus a copyright problem here).**

- Kindly, the ATLAS collaboration allowed to use this data nevertheless for this tutorial, but only under the condition that

  Every participant deletes the Monte Carlo data provided immediately after the tutorial.

Nevertheless, we are very happy to have obtained this permission from the ATLAS collaboration, as otherwise we would have needed to rewrite our tutorial completely.

In the future, we may want to ask our CheckMate friends to tell us on how to simulate some fake ATLAS data in the most efficient way to be used for this tutorial. ;-)

Input data

- Download the input data from:

- https://dmz-sv-owncloud.physik.uni-muenchen.de/public.php?service=files&t=49acd8b7754fffe166a905d063150adb

- ATLAS members just obtain the input data as described in the tutorial.

Tutorial location

- **ATLAS members:**
  - **Monday:** https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/HistFitterTutorial
  - **Tuesday:** https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/HistFitterAdvancedTutorial

- **Non-ATLAS members:**
  - **Monday:** https://twiki.cern.ch/twiki/bin/view/Main/HistFitterTutorialOutsideAtlas
  - **Tuesday:** https://twiki.cern.ch/twiki/bin/view/Main/HistFitterAdvancedTutorialOutsideAtlas

Differences between both tutorials minor. Mostly some few internal ATLAS things removed for the non-ATLAS tutorial.

HistFitter questions

Questions answered

- How to read fit output?

- Hypothesis test/upper limit fails, what to do?

- What are the different types of systematics?
- What are ‘norm’ systematics?
- Which systematic should I use?
  - Flow chart for what systematic to use

- How can I use asymmetric errors?

- Why should I use shape-fits?

- How do I make pretty plots in HistFitter?

Systematics - INTERLUDE

- HistFitter extends basic HistFactory systematic types

Basic systematic methods in HistFactory	
`overallSys`	uncertainty of the global normalization, not affecting the shape
`histoSys`	correlated uncertainty of shape and normalization
`shapeSys`	uncertainty of statistical nature applied to a sum of samples, bin by bin
Additional systematic methods in HistFitter	
`overallNormSys`	overallSys constrained to conserve total event count in a list of region(s)
`normHistoSys`	histoSys constrained to conserve total event count in a list of region(s)
`normHistoSysOneSide`	one-sided normHistoSys uncertainty built from tree-based or weight-based inputs
`normHistoSysOneSideSym`	symmetrized normHistoSysOneSide
`overallHistoSys`	factorized normalization shape and uncertainty, described with `overallSys` and `histoSys` respectively
`overallNormHistoSys`	overallHistoSys in which the shape uncertainty is modeled with a normHistoSys and the global normalization uncertainty is modeled with an overallSys
`shapeStat`	shapeSys applied to an individual sample

Table 1: Sub-set of the systematic methods available in HistFitter. The methods are specified by a string argument containing a combination of basic HistFactory methods and optional HistFitter keywords: `norm`, `OneSide` and/or `Sym`. Systematic objects can be built with Tree-based, weight-based, Float or histogram input methods in all cases.

- Do **not** correlate systematics of different types: histoSys and normHistoSys should **not** be correlated
- **What systematic to use?** See guidelines in next slides

‘norm’ systematics

- **Transfer factors** are used to extrapolate from CR to SR

$$N_p(\mathrm{SR},\mathrm{est.}) = N_p(\mathrm{CR},\mathrm{obs.}) \times \left[\frac{\mathrm{MC}_p(\mathrm{SR},\mathrm{raw})}{\mathrm{MC}_p(\mathrm{CR},\mathrm{raw})}\right] = \mu_p \times \mathrm{MC}_p(\mathrm{SR},\mathrm{raw})\,,$$

  - Np(CR,obs) = number of observed events in CR
  - Np(CR/SR,raw) = number of expected MC events in CR/SR
  - The ratio in [...]-brackets is called the ***transfer factor*** **(TF)**

- An important feature of TFs is that systematic uncertainties on the background estimates can be cancelled out in the extrapolation; a virtue of using ratio of MC estimates
- Total uncertainty on the background estimate in SR is then a combination of the statistical uncertainties on the normalization in CR (smaller stat. uncert.) and residual systematic of extrapolation (shape)

- Any systematics in HistFitter with ‘norm’ in the name are uncertainties on the transfer factors (residual systematic of extrapolation, shape of extrapolation)
  - Systematics without ‘norm’ are uncertainties on event counts
- **Caution:** Care must be taken when applying ‘norm’ type systematics, as it should only be applied to samples that carry a normalization factor **μ** in the fit. Otherwise the uncertainty on the total event count or normalization is not taken into account for this sample, which can lead to an underestimation of the sample uncertainty in the signal region.
  - The correlation between the systematic alpha_X and **μ** gets removed

‘norm’ systematics implementation

Example: `overallNormaHistoSys`

Implementation:

Define the systematic uncertainty as usual, e.g.:

```
Systematic("JES","_NoSys","_JESup","_JESdown","tree","overallNormHistoSys")
```

and set regions in which the systematic uncertainties will be normalized with respect to the nominal expectation:

```
mySample.setNormRegions([region1,region2,...])
```

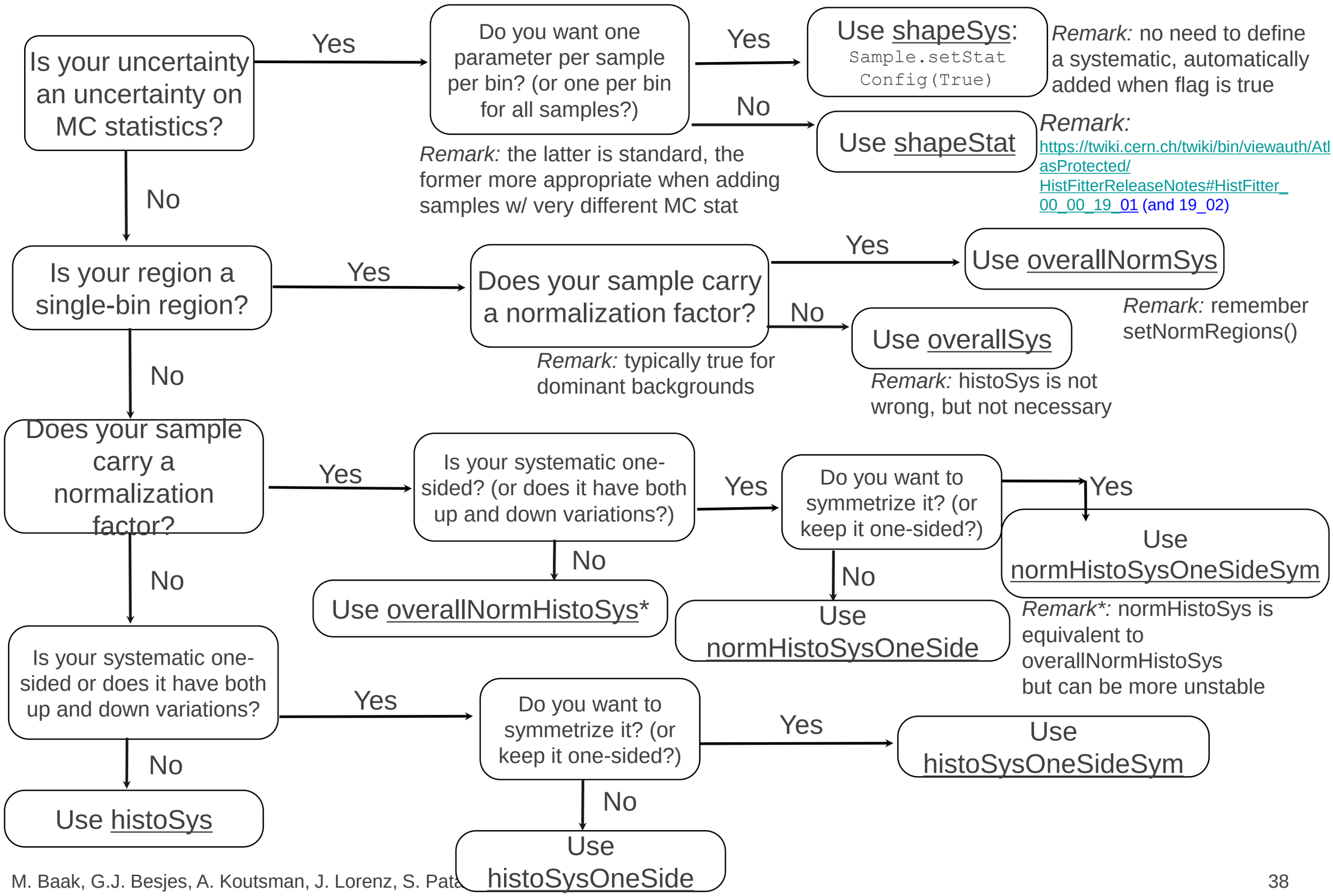
Usually the control regions are chosen for this normalization.

How does it work?

1. Temporary histograms are built containing the sum of the yields in all normalization regions (this is done for the nominal, the up and the down histogram)
2. First normalization step: Up and down temporary histograms are scaled to the yield in the nominal temporary histogram
3. The obtained scaling factor is applied to the original up and down histograms
4. If the systematics type contains an ’overall’ in the name - second normalization step: Up and down histograms are scaled to the yields in the nominal histogram. The resulting histograms are used with an `histoSys` type, the scaling factor with a `overallSys` type.

- If you do not set normalization regions (setNormRegions), the systematic will be used without being renormalized

Systematics ‘guidelines’



Understanding the fit result

- **mu**_ = signal strength per sample (normalization factor), unconstrained in the fit
- **alpha**_ = constrained parameter on systematic uncertainty (input alpha_ = 0 ± 1)
  - value of alpha_ represents preferred mean value of Gaussian (input value = 0)
  - error of alpha_ represents preferred gamma value of Gaussian (input value = 1) in units of input sigma
- *example*: after fit alpha_ = 0.1 ± 0.7 means
  preferred central value 0.1
  preferred uncertainty from data 0.7 x input uncertainty (30% profiled, not taking into account the correlations with other parameters)
- **gamma_stat**_ = constrained parameter (Poisson), bin-by-bin uncertainty from MC statistics (mainly for propagating errors, as no information in bin to constrain)
  - value of gamma_stat_ represents width of Poisson (conversely to alpha_)
  - error of gamma_stat_ represents error on width (conversely to alpha_)
- *example*: after fit gamma_ = 0.99 ± 0.05
  preferred sigma is 0.99 x input uncertainty

```
RooFitResult: minimized FCN value: -195178, estimated distance to minimum: 0.0201107
covariance matrix quality: Full, accurate covariance matrix    Floating Parameter    FinalValue +/-  Error
--------------------  --------------------------
                alpha_BT    1.6742e-01 +/-  4.50e-01
                alpha_JR   -4.0678e-01 +/-  5.00e-01
              alpha_JR3T   -4.0670e-02 +/-  8.79e-01
              alpha_JR4T   -6.8573e-02 +/-  8.91e-01
          alpha_KtScaleTop    4.1254e-01 +/-  5.48e-01
          alpha_KtScaleWZ    1.5158e-01 +/-  6.50e-01
                alpha_LE    6.5777e-04 +/-  9.91e-01
               alpha_LES    8.0288e-03 +/-  9.34e-01
               alpha_LRI    9.1592e-03 +/-  9.90e-01
              alpha_LRI3T    2.1968e-02 +/-  9.93e-01
              alpha_LRI4T   -1.7169e-02 +/-  9.93e-01
               alpha_LRM   -1.9373e-03 +/-  9.91e-01
              alpha_LRM3T    6.7347e-03 +/-  1.03e+00
              alpha_LRM4T   -2.0694e-03 +/-  9.99e-01
                alpha_MC    2.5970e-03 +/-  9.89e-01
               alpha_MC3T   -7.5278e-02 +/-  1.05e+00
               alpha_MC4T   -4.0237e-02 +/-  1.01e+00
                alpha_MP    1.1924e-01 +/-  6.49e-01
               alpha_MP3T   -9.7037e-02 +/-  9.91e-01
               alpha_MP4T   -1.5838e-01 +/-  8.81e-01
                alpha_PU   -6.5987e-01 +/-  9.15e-01
          alpha_PtMinTop3T   -1.3355e-02 +/-  1.03e+00
          alpha_PtMinTop4T   -2.2688e-02 +/-  1.05e+00
          alpha_PtMinTopC   -1.2472e-01 +/-  8.77e-01
          alpha_PtMinWZ3T   -2.9269e-02 +/-  1.06e+00
          alpha_PtMinWZ4T   -4.6418e-03 +/-  1.01e+00
          alpha_PtMinWZC    6.9847e-02 +/-  7.28e-01
alpha_QCDNorm_SR3jT_cuts   -5.2945e-02 +/-  9.64e-01
alpha_QCDNorm_SR4jT_cuts   -2.2599e-02 +/-  9.69e-01
 alpha_QCDNorm_TR_nJet    1.2121e-01 +/-  9.18e-01
alpha_QCDNorm_WR_nJet   -1.7881e-01 +/-  8.08e-01
                alpha_TE    3.6969e-04 +/-  9.89e-01
gamma_stat_SR3jT_cuts_bin_0    9.6901e-01 +/-  2.22e-01
gamma_stat_SR4jT_cuts_bin_0    9.8631e-01 +/-  1.58e-01
 gamma_stat_TR_nJet_bin_6    9.8506e-01 +/-  1.02e-01
gamma_stat_WR_nJet_bin_6    1.0177e+00 +/-  1.09e-01
                mu_SR3jT    9.2656e-04 +/-  3.18e+00
                mu_SR4jT    4.7156e-05 +/-  5.81e+00
                   mu_Top    9.9618e-01 +/-  1.06e-01
                    mu_WZ    8.8805e-01 +/-  1.07e-01
```

Asymmetric errors - Minos

- Run the fit as usual, but adding -m “ALL” (or -m “all” or a comma-separated list):
  `HistFitter.py -t -w -f -m all -F bkg --fitname Validation python/OneHardLepton_ForCombination_notower.py`

Further fitting options: Using Minos

Last year we encountered various problems with instable fits and with an artificial large profiling. A detailed summary of all problems was given by Max: `https://indico.cern.ch/event/245778/contribution/3/material/slides/0.pdf`

Main conclusions:

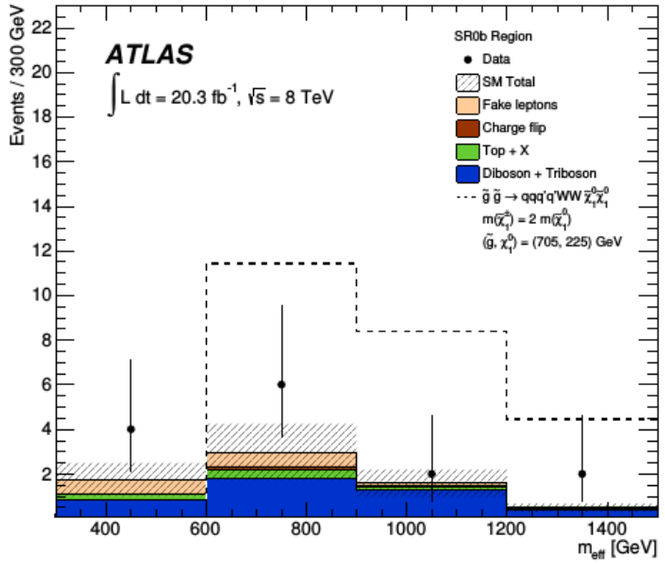
- If observing profiling check the fit by using Minos, the profiling may be artificial and point to an incorrect error given back by the Hesse calculation
  - Activate Minos by using -m PARAM, where PARAM is a comma separated list of all parameters for which the Minos calculation should be performed. Usually taking ’ALL’
- We are using Minuit 2 now, instead of Minuit. This seems to be more stable, but gives the same result.
- In addition the interpolation method between up and down systematic variations was changed to “polynomial interpolation + linear extrapolation”, which avoids kinks in the profile log-likelihood curve. More details in Max’ slides.

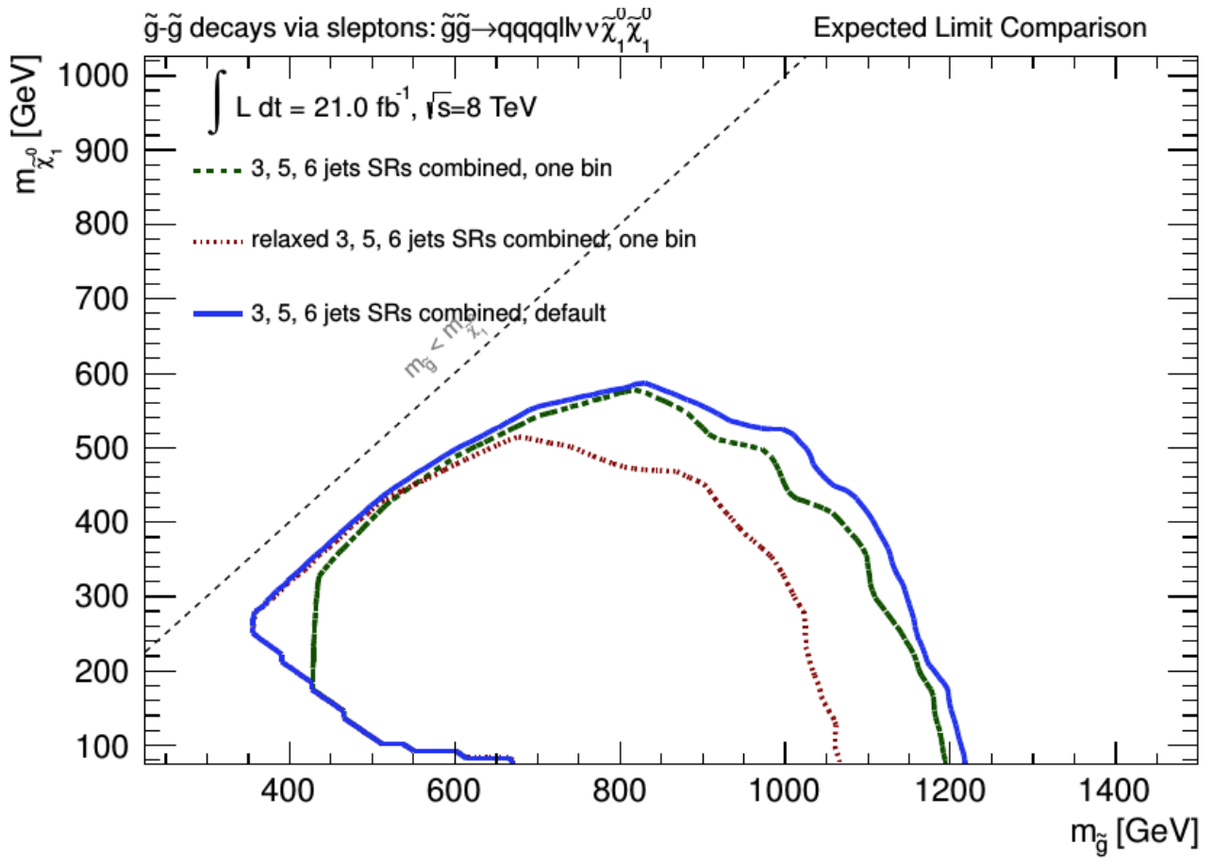
Some (anti-)profiling may still occur. If so: recommendation to check the profile log-likelihood curves.

Power of shape fits - combining SRs

- Many analyses now doing shape fits or combining exclusive SRs to boost performance or be more ‘general’

- Examples from SUSY 1-lepton searches
- Right-bottom: power of shape fits (Simplified 2-step Model)
  - using shape fits, multi-bin SRs (top right), while extending the exclusion reach (keeping the reach in worst case)





Backup

Test statistic/ calculator types

- Setting calculator and test statistic type can be set in configManager:

```
## setting the parameters of the hypothesis test
#configMgr.nTOYs=5000
configMgr.calculatorType=2 # 2=asymptotic calculator, 0=frequentist calculator
configMgr.testStatType=3   # 3=one-sided profile likelihood test statistic (LHC
default)
configMgr.nPoints=20       # number of values scanned of signal-strength for upper-
limit determination of signal strength.
```

Available methods:

- calculatortype = 0 Freq calculator (= limit using toy experiments, using a fully Frequentist approach)
- type = 1 Hybrid calculator (= limit using toy experiments, using a Bayesian-Frequentist hybrid approach)
- type = 2 Asymptotic calculator (= limit using asymptotic formula)
- type = 3 Asymptotic calculator using nominal Asimov data sets (not using fitted parameter values but nominal ones)

The calculators typically used in ATLAS are 0, or 2.
When choosing the Frequentist or Hybrid calculator, one needs to specify the number of toy experiments.

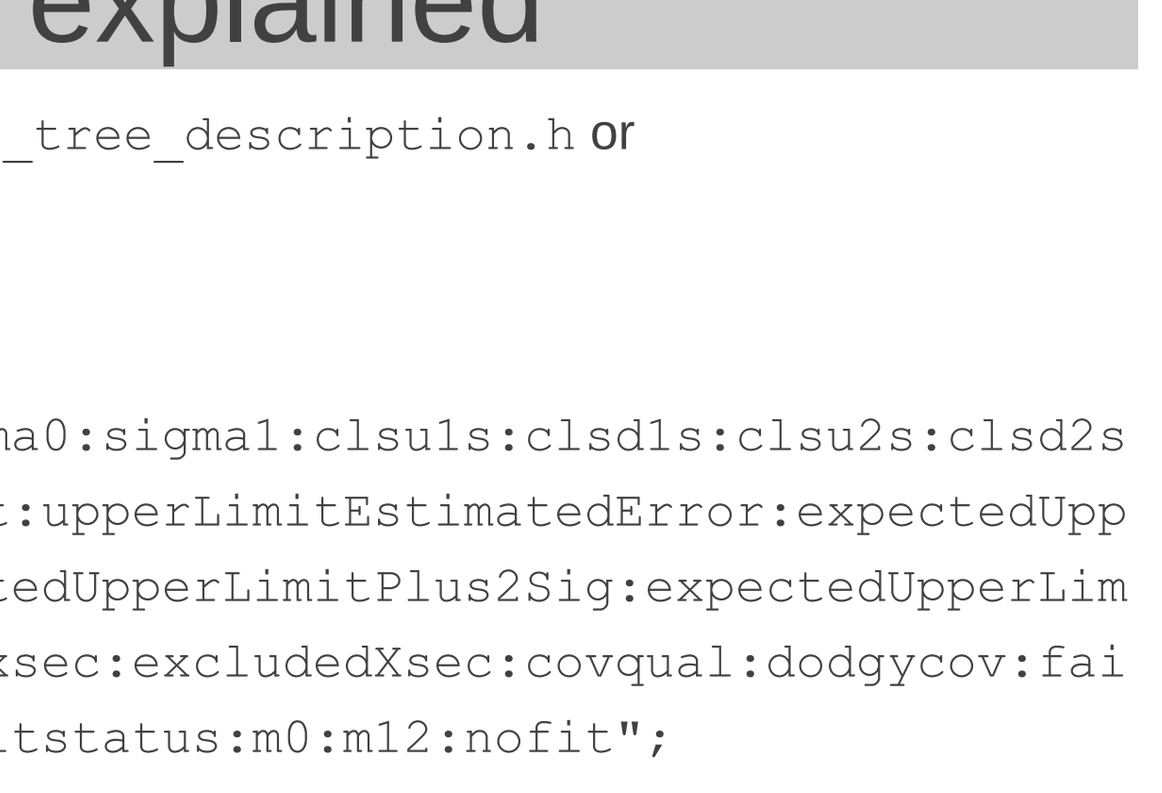
- testStatType = 0 LEP
- = 1 Tevatron
- = 2 Profile Likelihood
- = 3 Profile Likelihood one sided (i.e. = 0 if mu $<$ mu_hat)

Summary list explained

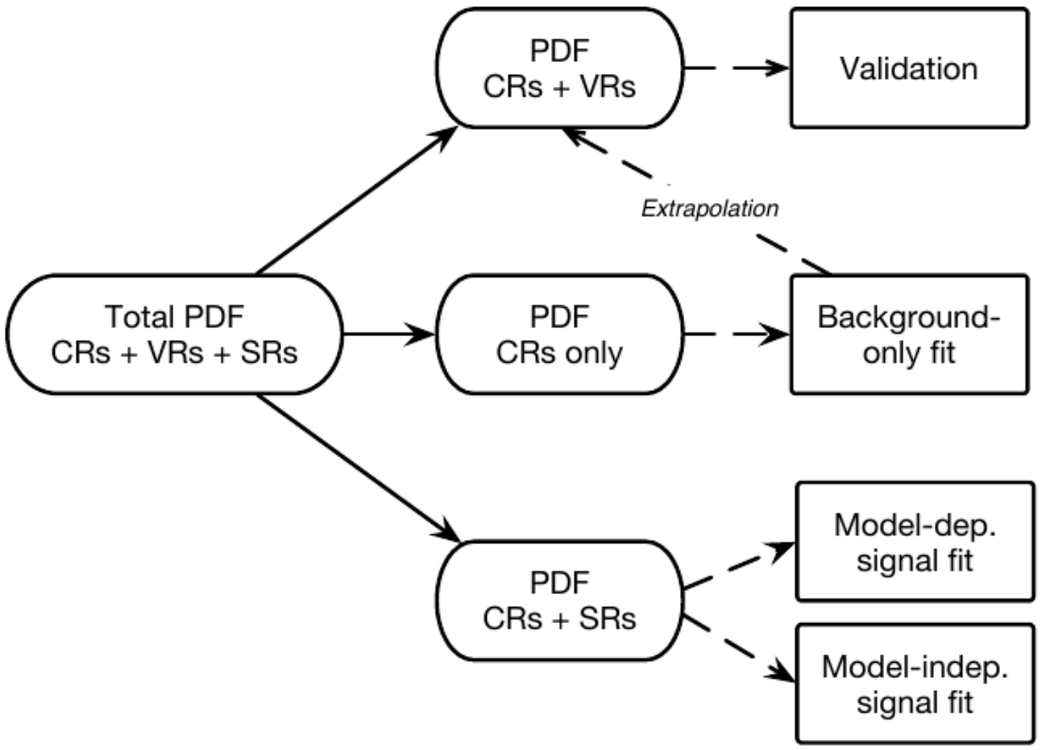
- Your summary list description is in `summary_harvest_tree_description.h` or `summary_harvest_tree_description.py`:

```
const char* description =
"p0:p1:CLs:mode:nexp:seed:CLsexp:fID:sigma0:sigma1:clsu1s:clsd1s:clsu2s:clsd2s
:p0exp:p0u1s:p0d1s:p0u2s:p0d2s:upperLimit:upperLimitEstimatedError:expectedUpp
erLimit:expectedUpperLimitPlus1Sig:expectedUpperLimitPlus2Sig:expectedUpperLim
itMinus1Sig:expectedUpperLimitMinus2Sig:xsec:excludedXsec:covqual:dodgycov:fai
ledcov:failedfit:failedp0:failedstatus:fitstatus:m0:m12:nofit";
```

- Clearly m0 and m12 are used as parameters of the model (other/more parameter possible)
- CLs_observed = `CLs`
- CLs_expected = `CLsexp`
- CLs_expected ±1sigma experimental uncertainty = `clsu1s` / `clsd1s` (yellow band)
- CLs_observed ±1sigma signal theory uncertainty = `CLs` from up/down variation files (red dashed lines) - see tutorial Part5



Extrapolation/error propagation



Extrapolation into validation and signal regions:

- Deconstruction of full likelihood containing CRs and VRs/SRs into smaller likelihood only containing CRs for use in the background-only fit.
- Incorporation of fitted parameters after background-only fit into full likelihood.
- Evaluation of the extrapolated uncertainty in validation/signal regions through standard error propagation.

Extrapolation into signal and validation regions particularly rigorous in HistFitter due to use of `RooExpandedFitResult` class:

- Standard `RooFitResult` contains only the parameters used in the background-only fit.
- Using instead the `RooExpandedFitResult` class allows to extrapolate **all** parameters, such that a correct evaluation of the uncertainties through error propagation is possible.

Region/fit type definition

More advanced fit 1: explicit use of control, validation and signal regions

Every channel in a config file should be specified as either control, validation or signal region:

- `yourFitConfig.setSignalChannels([SR1,SR2,...])`
- `yourFitConfig.setBkgConstrainChannels([CR1,CR2,...])`
- `yourFitConfig.setValidationChannels([VR1,VR2,...])`

Using this, different fit strategies can be separated:

- **Background-only fit:** Only the CRs are used to constrain the fit parameters. Any potential signal contribution is neglected everywhere. Only regions set in `setBkgConstrainChannels` are used to constrain the backgrounds. The fit result may be extrapolated to regions defined in `setValidationChannels` (signal regions also need to be included here, if an extrapolation the signal regions is wished)
- **Exclusion fit:** CRs and SRs are used in the fit. The potential signal contribution is taken into account as predicted by the tested model in all the regions. This option considers only the signal and control regions. No validation regions may be set.
- **Discovery fit:** Both CRs and SRs are used in the fit. The signal is independently considered in each SR, but is neglected in the CRs. In this way, one total background prediction in the signal regions is achieved. This background prediction is conservative since any signal contribution in the CRs is attributed to background and thus yields a possible overestimation of the background in the SRs. Dedicated SRs (and signal samples) need to be used, which are one bin counting-experiments. A 'dummy' signal model that is a single bin histogram with the bin value at 1 is used. (The discovery fit should not have any model dependence.) Validation regions may not be used in this fit configuration.

The fit type can be defined by the option -F {excl,disc,bkg}. This sets the variable myFitType to FitType.Exclusion, FitType.Discovery, FitType.Background, respectively, which can be propagated to the configuration file.

RooStats HistFactory tool

- Builds PDFs from histograms using XML configuration
  - Supports multiple regions (=channels)
  - Systematics defined in a variety of ways

$$P(n_{cb}, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_c \prod_b \text{Pois}(n_{cb} \mid \nu_{cb}) \text{G}(L_0 \mid \lambda, \Delta_L) \prod_p \text{P}_P(a_p \mid \alpha_p)$$

$$\nu_{cb}(\phi_p, \alpha_p, \gamma_b) = \lambda_{cs} \gamma_{cb} \phi_{cs} \eta_{cs}(\vec{\alpha}) \sigma_{csb}(\vec{\alpha})$$

Lumi

ShapeSys
MC stats
ShapeFactor

NormFactor

OverallSys

HistoSys

- Central disadvantage: PDFs cannot be contructed in *flexible* way
- Solution: HistFitter acts as flexible wrapper to build and analyze PDFs.

- https://twiki.cern.ch/twiki/pub/RooStats/WebHome/HistFactoryLikelihood.pdf

Blinding & selection optimization

- Some people are starting to use HistFitter for optimization of signal selection.
- Need to be careful not to include the data under investigation.

- New option: blinding of data.

- Possibility to blind CRs and/or SR.
  - If blinded, MC expectation is chosen for data.

- Select in configuration file with:
  - configMgr.blindSR = True
  - configMgr.blindCR = True

ROOSTATS example

- **RooStats** project/tools suite delivers a series of tools that can calculate intervals and perform hypothesis tests using a variety of statistical techniques

An example of a custom RooStats 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

‘exclusion’ vs ‘discovery’

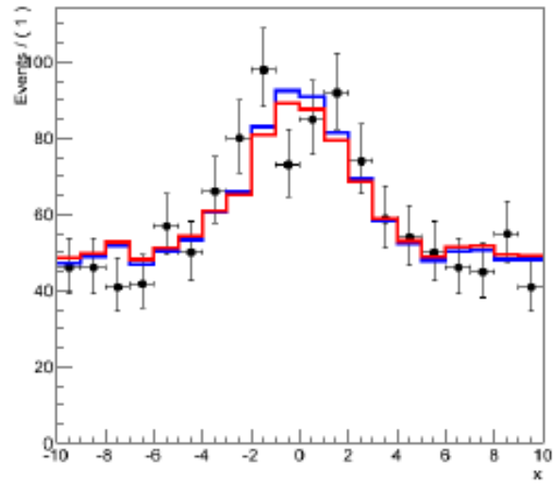
‘exclusion’

‘likelihood assuming μ signal strength’

$$\tilde{q}_\mu = -2\ln\frac{L(data\,|\,\mu,\hat{\hat{\theta}}_\mu)}{L(data\,|\,\hat{\mu},\hat{\theta})}$$

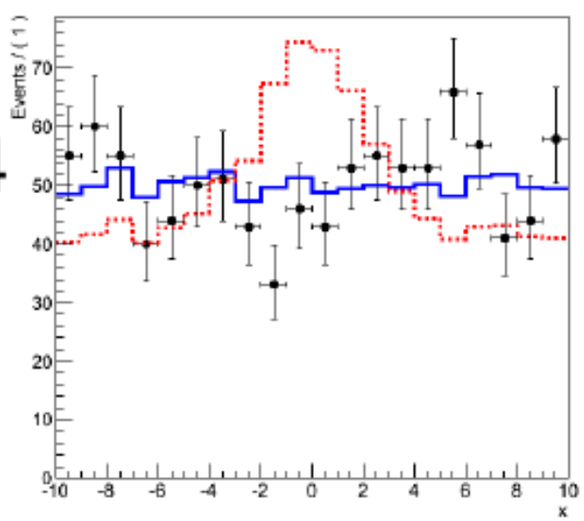
‘likelihood of best fit’

$\tilde{q}_1 = 0.77$



simulated data with signal+bkg

$\tilde{q}_1 = 52.34$



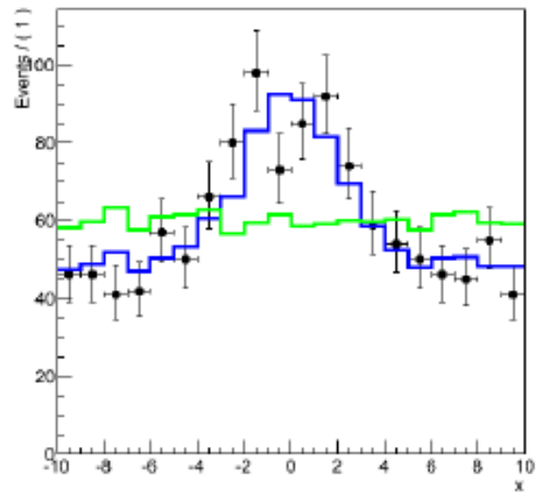
simulated data with bkg only

‘discovery’

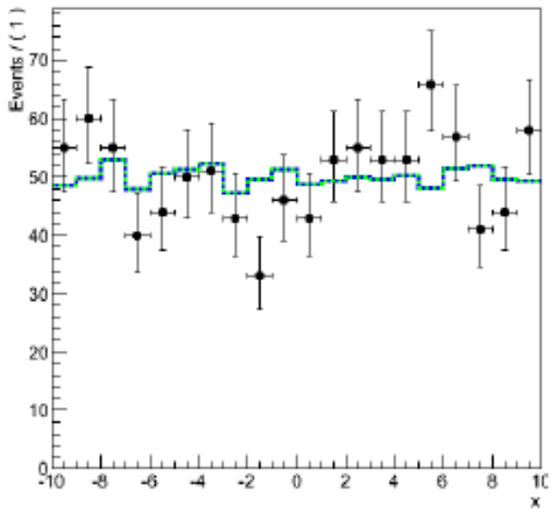
‘likelihood assuming background only’

$$\tilde{q}_0 = -2\ln\frac{L(data\,|\,\mu=0,\hat{\hat{\theta}}_0)}{L(data\,|\,\hat{\mu},\hat{\theta})}$$

‘likelihood of best fit’



$\tilde{q}_0 = 34.7$



$\tilde{q}_0 = 0$

Pull plot production

- To validate the extrapolation from CR to SR, one typically looks at the VRs
- Within HistFitter one can produce a ***pull*** plot, that summarizes the agreement between data and extrapolated prediction in all the VRs

$$\chi = \frac{n_{\text{obs}} - n_{\text{pred}}}{\sigma_{\text{tot}}}$$

$$\sigma_{\text{tot}} = \sqrt{\sigma^2_{\text{pred}} + \sigma^2_{\text{stat, exp}}}$$

- In tutorial: https://twiki.cern.ch/twiki/bin/view/AtlasProtected/HistFitterTutorial#Validation_pull_plot