MadTools/StatTools

W. Quayle

University of Wisconsin-Madison

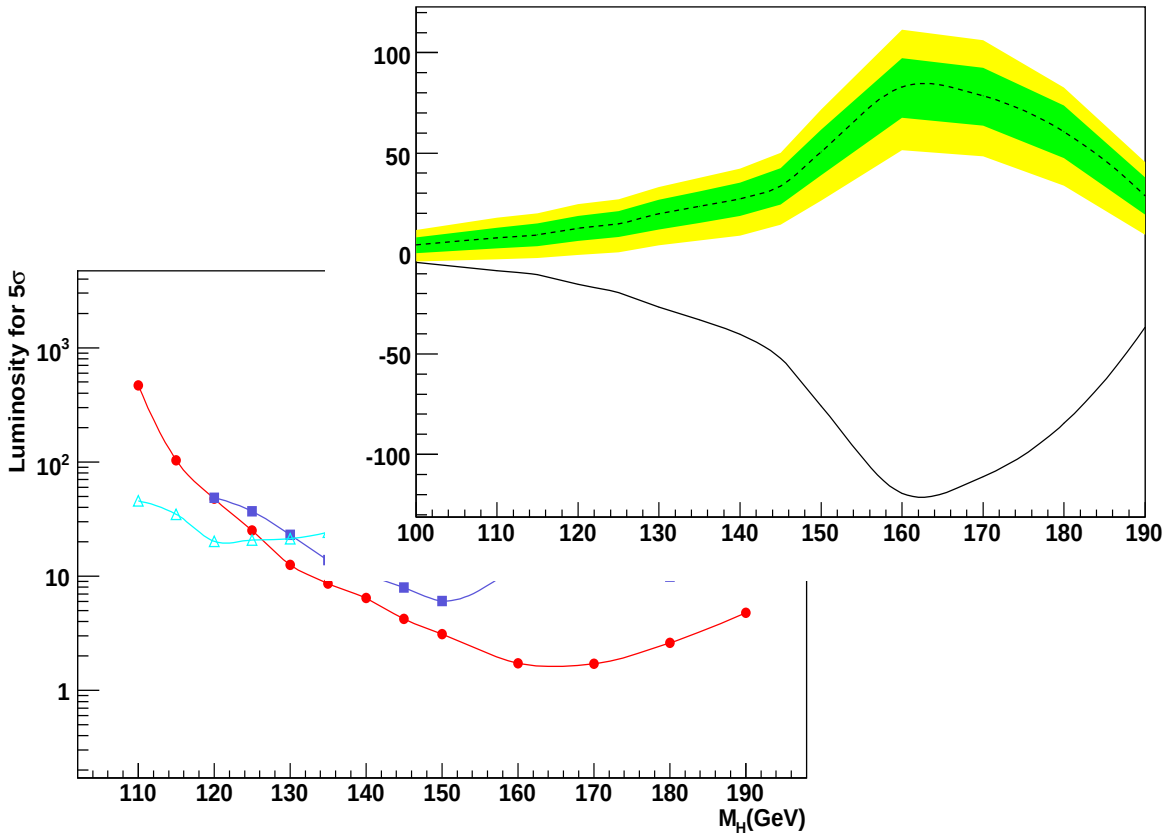
Root 2007 Workshop

28 March 2007

Introduction

- Statistical analysis is a vital part of any search for new physics
  - Shared tools are desirable

- In this talk:
  - Overview of existing features and recent developments
  - Emphasis on software issues (API, etc.)



All plots are examples only; no official results

Goals and Status

- Aim to provide a unified interface to a variety of statistical treatments
  - Scripts that use statistical information can then be written in a way that is independent of the formulation
- Already implemented and public:
  - Toy MC computation similar to TLimit
  - FFT-based algorithms similar to CLFFT
- Currently under development:
  - Toy MC computation for fit-based searches (compatible with hand-coded direct calls to Minuit as well as RooFit)
- Not started yet, but under consideration:
  - Interface to "CL_s with fits" limit calculator from Tom Junk

Command-line Interfaces

- The publicly-available code comes with simple-minded command-line and xml-driven utilities that can compute combined significances for number-counting searches

MadTools/run> ./significance -s 50 -b 10 --precision 50
(...output suppressed...)
Total significance combined with Likelihood Ratio: 10.677

- These features are documented in a talk in an ATLAS Physics Analysis Tools meeting, so I won't go into detail about them in the present talk.

http://indico.cern.ch/conferenceDisplay.py?confId=10328

Programming Interface (1)

- TDistribution: a class for low-level data
  - In the public version, this only uses binned data, but the development version can deal with unbinned data too

```
TFile f("some_file.root");
TH1* some_hist=(TH1*)f.Get("some_hist");
TDistribution dist(some_hist,(int)ID,xsec);
```

- THypothesis: collection of TDistributions

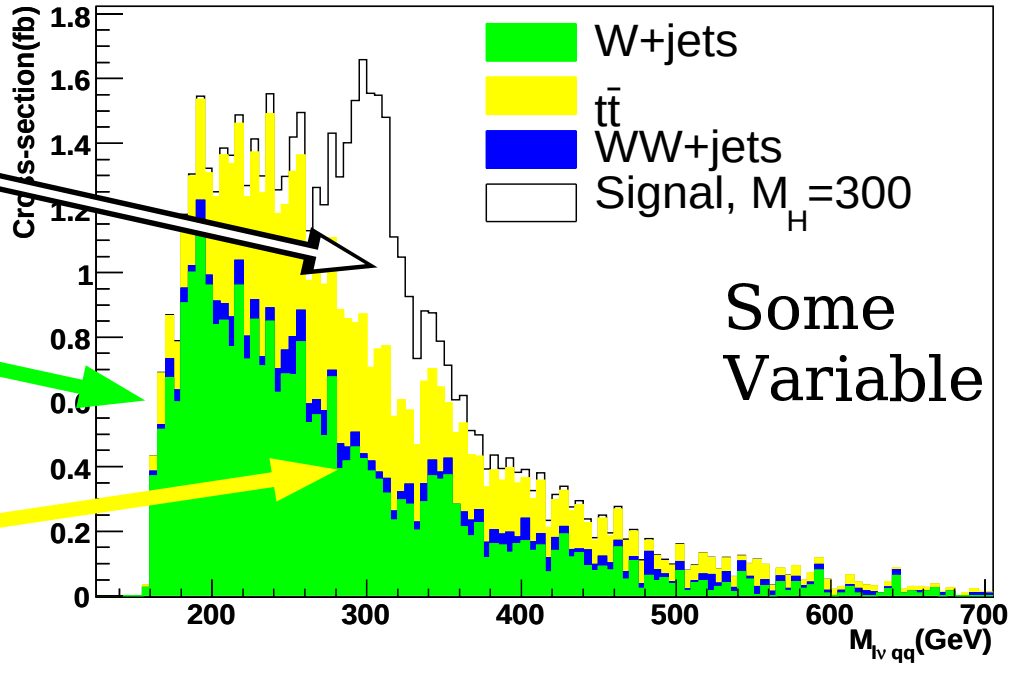
```
THypothesis hyp;
hyp.AddProcess(sig_dist,true,"some signal");
hyp.AddProcess(bg_dist_1,false,"some BG");
hyp.AddProcess(bg_dist_2,false,"other BG");
```

Programming Interface (2)

Signal corresponds to one TDistribution

Background 1 corresponds to another TDistribution

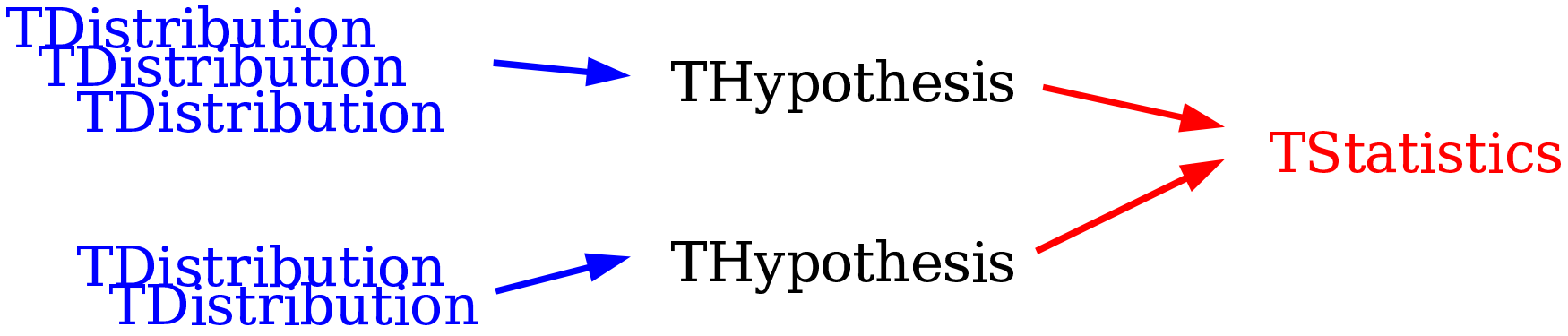
Background 2 is another one



- A THypothesis represents the expectations for several processes in some region of phase space
  - Signal and background are labelled as such, so a THypothesis is used, e.g., to generate toy MC in S+B and BG-only hypothesis

Programming Interface (3)

- TStatistics: an abstract base class for confidence level calculations
- Typical program flow:
  - Create TDistributions
  - Assemble a THypothesis object for each channel you want to combine
  - Pass them to a TStatistics instance and compute significances, etc.



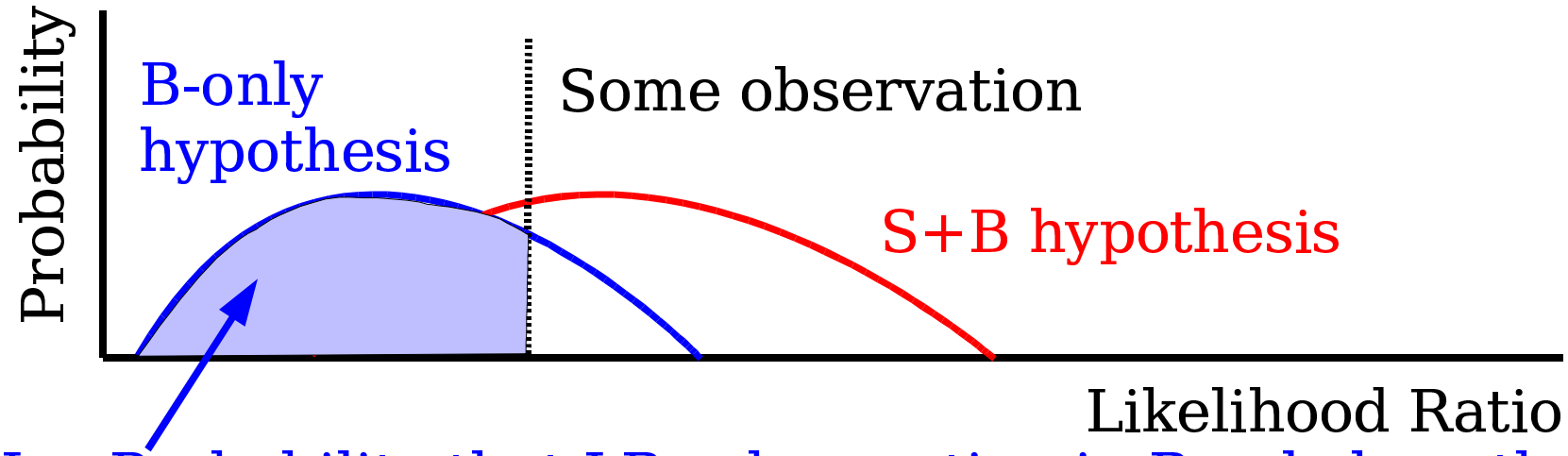
An Example Script

```
int main(){
  TFile f("some_file.root");
  Double_t lumi(30);
  //suppose that these histograms are
  //normalized to cross-section
  TH1* hist12=(TH1*)f.Get("h12");
  TH1* hist22=(TH1*)f.Get("h22");
  TH1* hist32=(TH1*)f.Get("h32");
  TDistribution dist12(hist12,1,hist12->Integral());
  TDistribution dist22(hist22,2,hist22->Integral());
  TDistribution dist32(hist22,3,hist32->Integral());
  THypothesis hyp;
  hyp.AddProcess(dist12,true,"signal");
  hyp.AddProcess(dist22,false,"BG1");
  hyp.AddProcess(dist32,false,"BG2");
  TStatistics* stat=new TLikelihoodRatioFFT();
  stat->AddChannel(hyp);
  stat->GetTestStatDist(lumi,(int)pow(2.,19.));
  cout << "sig=" << stat->Significance() <<endl;
}
```

Important Methods (1)

- TStatistics::AddChannel(THypothesis&)
  - Just what it sounds like
  - You can combine multiple channels by making multiple calls to this method
- TStatistics::Configure(TiXmlElement*)
  - Some derived classes – especially fit-based implementations – need configuration information. XML is a good way to do this.
- TStatistics::GetTestStatDist(double,int)
  - Base class stores the probability distribution of the test statistic as an array. Implementations of this pure virtual function compute it.

Important Methods (2)



CL_b = Probability that LR<observation in B-only hypothesis
= Erf("number of sigmas"/√2)

- TStatistics::Significance() returns the expected "number of sigmas"
- More generally, the methods FractionOfDistGreaterThan() and FractionOfDistLessThan() can be used to obtain other interesting p-values

Available in Public Version (1)

- The public code implements three TStatistics derived classes, all based on a "LEP-style" Likelihood Ratio:

$$LR(obs) = \frac{L_{s+b}(obs)}{L_b(obs)}$$

- For a number-counting experiment, the likelihood L_X(obs=N events) is just the Poisson prob. to see n events with X(assumed to be well-known) expected

- One can take advantage of information about the shape of a variable by multiplying the likelihood ratios for the bins of a histogram:

$$LR(hist) = \prod_{i=1}^{Num.Bins} LR(bin\ \ i)$$

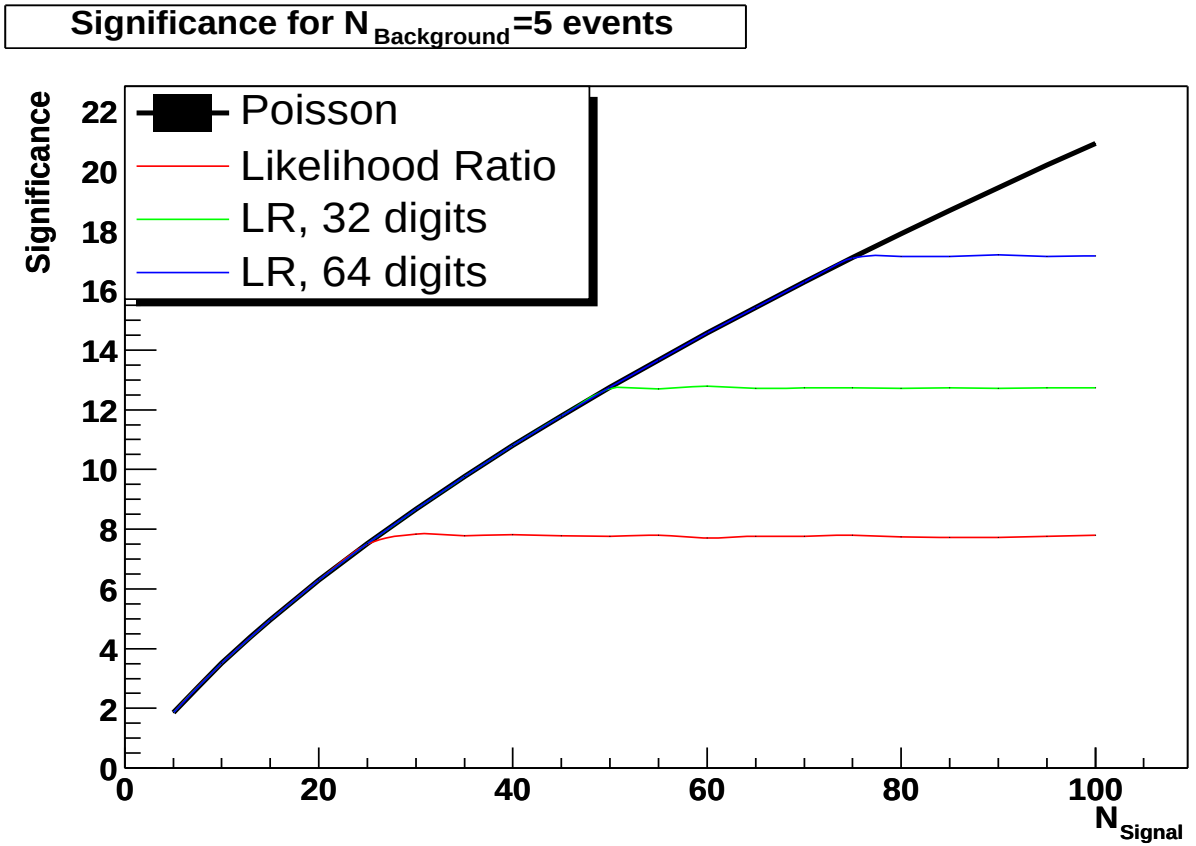
- The classes in the public version assume that the discriminating variable shape is well-known

Available in Public Version (2)

- TStatistics derived classes in public version
  - TLikelihoodRatioToyMC: Toy Monte Carlo algorithm similar to TLimit
  - TLikelihoodRatioFFT: Fast Fourier Transform based calculation similar to CLFFT
  - TLikelihoodRatioFFT_cln: like TLikelihoodRatioFFT, but using high-precision floating point numbers to suppress rounding errors

- All these classes can apply a Cousins-Highland smearing to take account of systematic error on BG normalization

Available in Public Version (3)



- The high-precision floating point FFT class allows the calculation of large significances
  - Handy for significance plots with large dynamic range: no approximations

Newer Features Still Under Development

Fit-based Search Channels (1)

- It is difficult to work on statistics at LHC without also working on fits
  - In-situ BG determination is vital in many situations
- Very good fitting toolkits are already available
  - See, e.g., talks from L. Moneta and W. Verkerke
- Not everyone who does fits uses the same tools
  - One would like to reuse as much code as possible when performing a combined fit
    - Make as few assumptions as possible about the inner workings of fitters contributed by colleagues
- Goal: design an interface so that most fitters can be incorporated into a combined fit with minimal changes to the core logic of the individual fitter

Fit-based Search Channels (2)

- Use TStatistics et al. to allow users to:
  - Combine fit-based searches built using different frameworks (e.g. hand-coded vs. RooFit)
    - Wrap functions that any fitter is likely to have
  - Combine fit-based searches with number-counting and “LEP-style” CL calculations (maybe)
- Current status and plans
  - Several working examples of fitters
    - hand-coded binned Maximum Likelihood fits
    - a RooFit-based multi-region unbinned Maximum Likelihood fit with a penalty term
    - a combined fitter which can combine the hand coded fits and “ought to work” with the RooFit one
  - Need to try incorporating a few fitters written by other people to make sure interface is practical

How it Works (1)

- Introduce a generic interface (TBackgroundFitter) to fit-related tasks that have to be performed for a combination
  - More about this class later
- Extend TDistribution to accommodate unbinned (TTree) data
  - This is done via composition rather than inheritance because the interface to access binned and unbinned data is inherently different
  - Methods IsHist() and IsTreeData() are used to identify which kind of information is present
  - Most code that needs to know these details goes into a TBackgroundFitter derived class

How it Works (2)

- Class TLikelihoodRatioToyMC_fits (derived from TStatistics) handles high-level operations by talking to the TBackgroundFitter interface
  - Generate toy Monte Carlo outcomes
  - Run fits on toy outcomes
  - Save Likelihood Ratio, best fit parameters, etc. in a TTree
  - Produce new ntuples to store toy results or use existing files to obtain the probability distribution for the likelihood ratio (needs testing)

Methods in TBackgroundFitter

- virtual void Initialize(...)=0
  - Don't have to do anything here, but most fitters will do something, e.g. setting up PDFs
- virtual double GetLikelihood(...)=0
  - Compute the function to be maximized by MINUIT given a set of parameters.
- virtual double GetLikelihoodRatio(...)=0
  - Compute test stat. given a (pseudo-)observation and fit parameters for S+B and BG-only hyp.
- virtual double DoMinimization(...)=0
  - Perform any fits necessary to compute the test statistic for an observation, and return its value
- virtual vector<TDistribution> Toss(...)
  - Generate a toy Monte Carlo outcome

Methods in TBackgroundFitter

- There are a number of trivial bookkeeping functions that are called by routines that use TBackgroundFitter
  - const char* GetType()
  - TBackgroundFitter* Clone()
  - GetParNames_SB() and GetParNames_BG()
  - HasLimits_SB() and HasLimits_BG()
  - Functions to get upper and lower param. limits
  - Functions to match parameters from one fitter with parameters from another
    - Only needed if you do combined fits where you constrain, e.g., M_H to be the same in all channels
  - Functions to read out the best-fit parameters for a particular outcome

TCombinedBackgroundFitter

- With the interface standardized, it is straightforward to write a combined fitter
  - Initialize(): call the Initialize() method for all channels to be combined
  - GetLikelihood(): Add up the output of individual channels.
    - This function is virtual in case adding is no good
  - GetLikelihoodRatio(): Add up the output of individual channels
  - DoMinimization(): minimize -1 times the output of GetLikelihood
- I expect all this code to evolve a bit before it is released.
  - Example: will probably add a virtual function to perform process-specific convergence tests

Summary

- The MadTools/StatTools package implements a unified interface to several algorithms to calculate confidence levels
  - A version of the code can be found here: http://www-wisconsin.cern.ch/physics/software.html
- Recent developments aim to expand this software to include confidence level calculations for combined fits
  - Work on new features is ongoing, and code will be made public when it is ready