The Bayesian Analysis Toolkit

Daniel Kollár – CERN

Frederik Beaujean, Allen Caldwell – Max-Planck-Institute for Physics, Munich
Kevin Kröninger – University of Göttingen

School of Statistics 2010

Autrans, France, May 21, 2010

Introduction

BAT → Software package for solving of statistical problems using Bayesian approach

Bayes' formula for parameter estimation

$$p(\vec{\lambda} \mid \vec{D}) = \frac{p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})}{\int p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})\, d\vec{\lambda}}$$

Motivation:

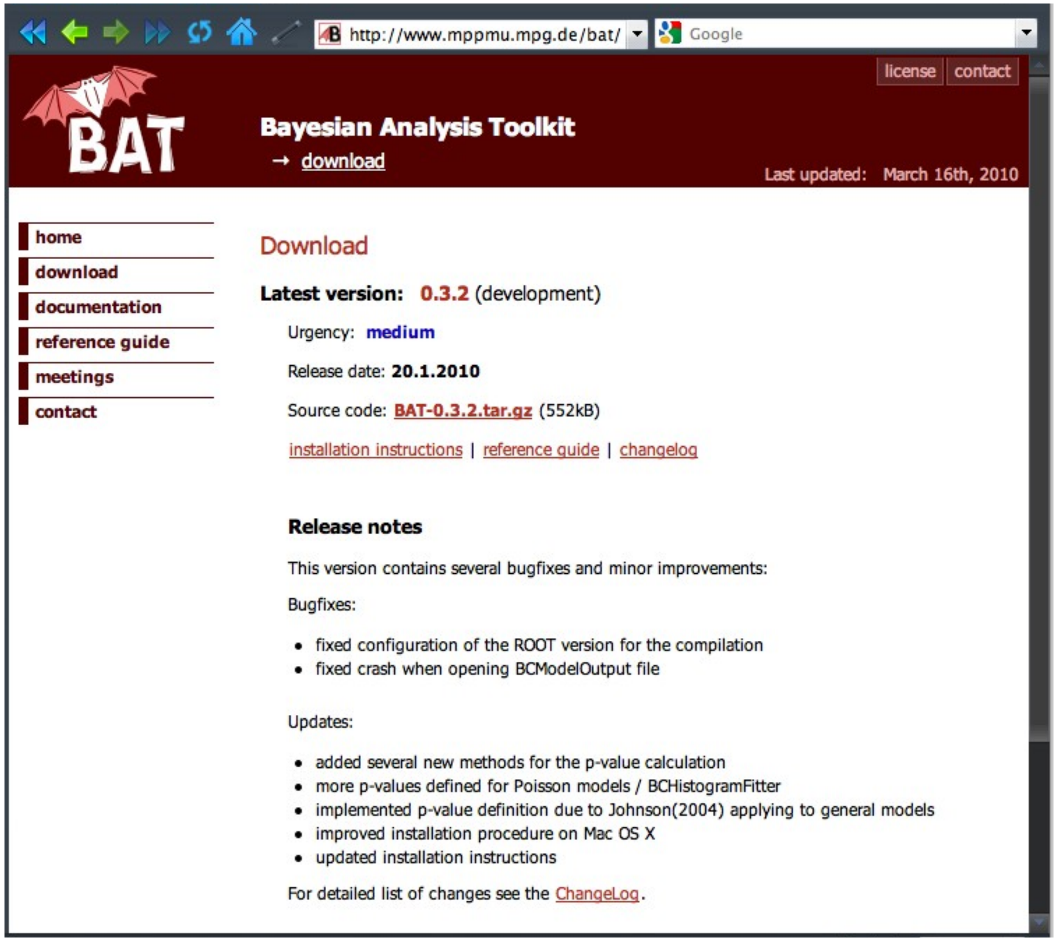
- many of us have done Bayesian analyses in HEP always having to implement the numerical algorithms and tools by ourselves → generally non-trivial
- create a package/toolkit to take care of that

The idea behind BAT

- Merge common parts of every Bayesian analysis into a software package
- Provide flexible environment to phrase arbitrary problems
- Provide a set of well tested/tuned numerical algorithms and tools
- C++ based framework (flexible, modular)
- Interfaces to ROOT, Cuba, Minuit, user defined, ...

BAT availability

- can be downloaded from: http://www.mppmu.mpg.de/bat
- BAT comes in form of shared library
- depends of the ROOT I/O functionality
- BAT contains at the moment 15 classes which provide:
  - main infrastructure
  - algorithms
  - output and logging
  - extension classes to solve specific (frequent) fitting problems
- a set of well documented examples is included in BAT distribution
  - good starting point
- "Introduction to BAT" document



Bayesian Analysis Toolkit
→ download
Last updated: March 16th, 2010

Download

Latest version: 0.3.2 (development)

Urgency: **medium**

Release date: **20.1.2010**

Source code: BAT-0.3.2.tar.gz (552kB)

installation instructions | reference guide | changelog

Release notes

This version contains several bugfixes and minor improvements:

Bugfixes:

- fixed configuration of the ROOT version for the compilation
- fixed crash when opening BCModelOutput file

Updates:

- added several new methods for the p-value calculation
- more p-values defined for Poisson models / BCHistogramFitter
- implemented p-value definition due to Johnson(2004) applying to general models
- improved installation procedure on Mac OS X
- updated installation instructions

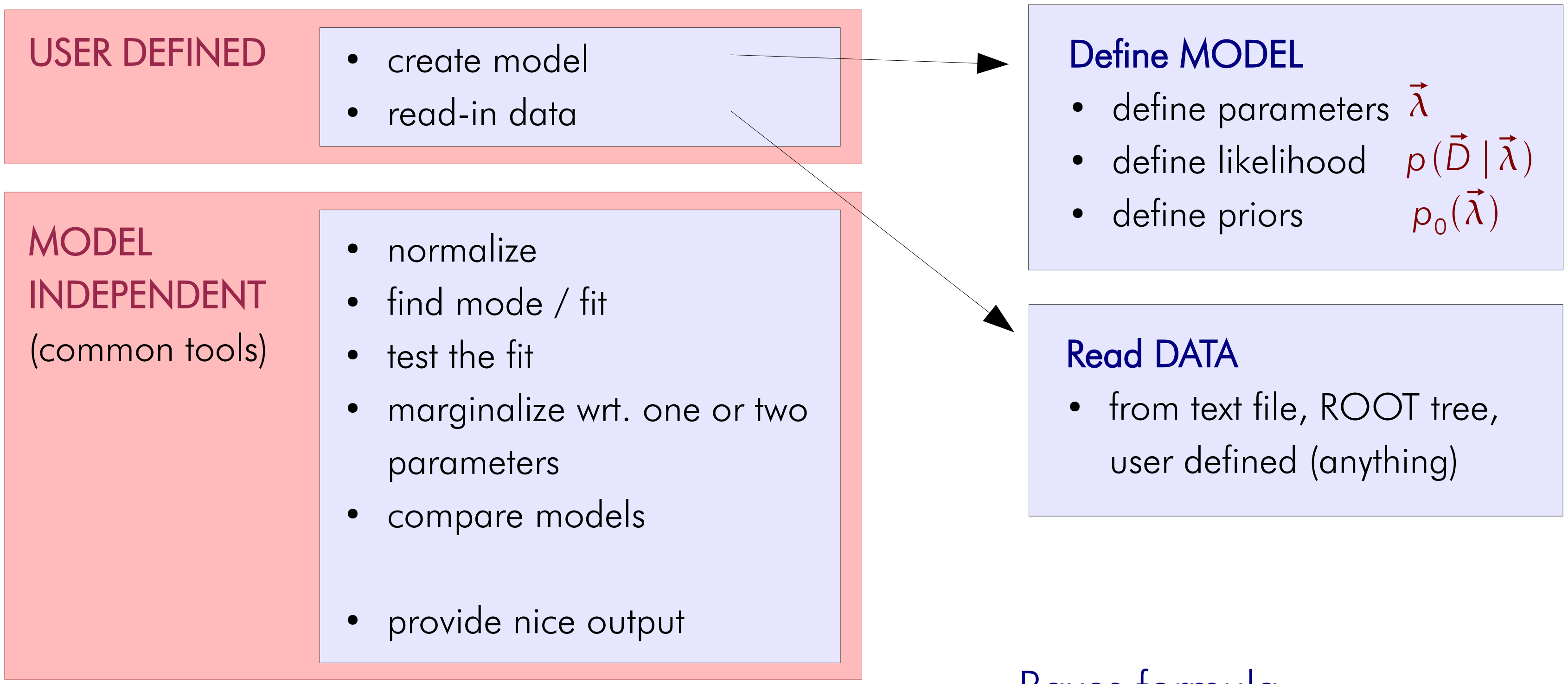
For detailed list of changes see the ChangeLog.

- the BAT paper: *Computer Physics Communications* **180** (2009) 2197-2209

The idea

Separate the common parts from the rest

- case specific: the model and the data
- common tools: all the rest



USER DEFINED

- create model
- read-in data

MODEL INDEPENDENT (common tools)

- normalize
- find mode / fit
- test the fit
- marginalize wrt. one or two parameters
- compare models
- provide nice output

Define MODEL

- define parameters $\vec{\lambda}$
- define likelihood $p(\vec{D}\,|\,\vec{\lambda})$
- define priors $p_0(\vec{\lambda})$

Read DATA

- from text file, ROOT tree, user defined (anything)

Bayes formula

$$p(\vec{\lambda}\,|\,\vec{D}) = \frac{p(\vec{D}\,|\,\vec{\lambda})\; p_0(\vec{\lambda})}{\int p(\vec{D}\,|\,\vec{\lambda})\; p_0(\vec{\lambda})\; d\vec{\lambda}}$$

Common tools

- Posterior mapping $\rightarrow$ Marginalization
  - Markov Chain Monte Carlo (MCMC)
    - key tool in the package
    - lot of emphasis put on efficiency, performance and validation

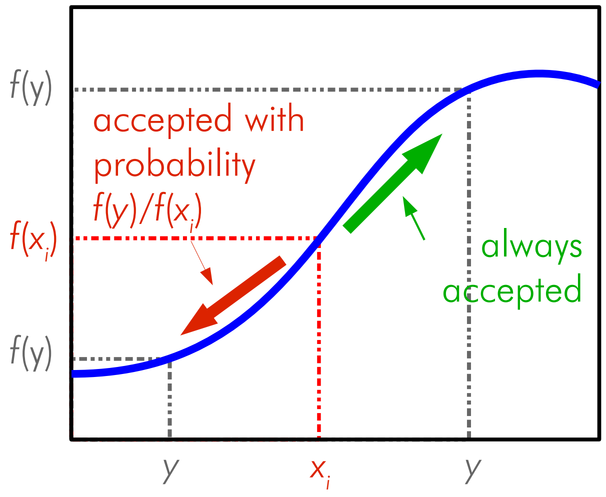
- Integration
  - Monte Carlo (sampled mean), Cuba (Vegas, …)
- Maximization
  - Monte Carlo, MCMC, Minuit, Simulated Annealing
- Model testing
  - Posterior comparison, K-factors, p-value calculation
- User interface
  - simple model definition
  - standard output: text output, plots, ROOT histograms and trees, …

Markov Chain Monte Carlo (MCMC)

- generally it is very difficult to obtain the full posterior PDF
  - number of parameters can be large
  - different input data will result in a different posterior
- also the visualization of the PDF in more than 3 dimensions is rather impractical and hard to understand
- usually one looks at marginalized posterior wrt. one, two or three parameters
  - a projection of the posterior onto one (two, three) parameter
  - integrating all the other parameters out $$p(\lambda_i \mid \vec{D}) = \int p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})\, d\vec{\lambda}_{j \neq i}$$
  - still numerically difficult
- the Markov Chain Monte Carlo revolutionized the area of Bayesian analysis
  - Metropolis algorithm

Metropolis algorithm

- In BAT implemented Metropolis algorithm
- Map positive function ***f*(*x*)** by random walk towards higher probabilities
- Algorithm:
  - Start at some randomly chosen x_i
  - Randomly generate y around x_i
  - If $f(y) \geq f(x_i)$, set $x_{i+1} = y$
  - If $f(y) < f(x_i)$, set $x_{i+1} = y$ with probability $p = \frac{f(y)}{f(x_i)}$
  - If y not accepted, stay where you are, i.e., set $x_{i+1} = x_i$
  - Start over



- For each step fill the histogram with x_{i+1}
- For infinite number of steps the distribution in the histogram converges to ***f*(*x*)** except for the normalization

MCMC: an example

- mapping an arbitrary function:
  $f(x) = x^4 \sin^2 x$

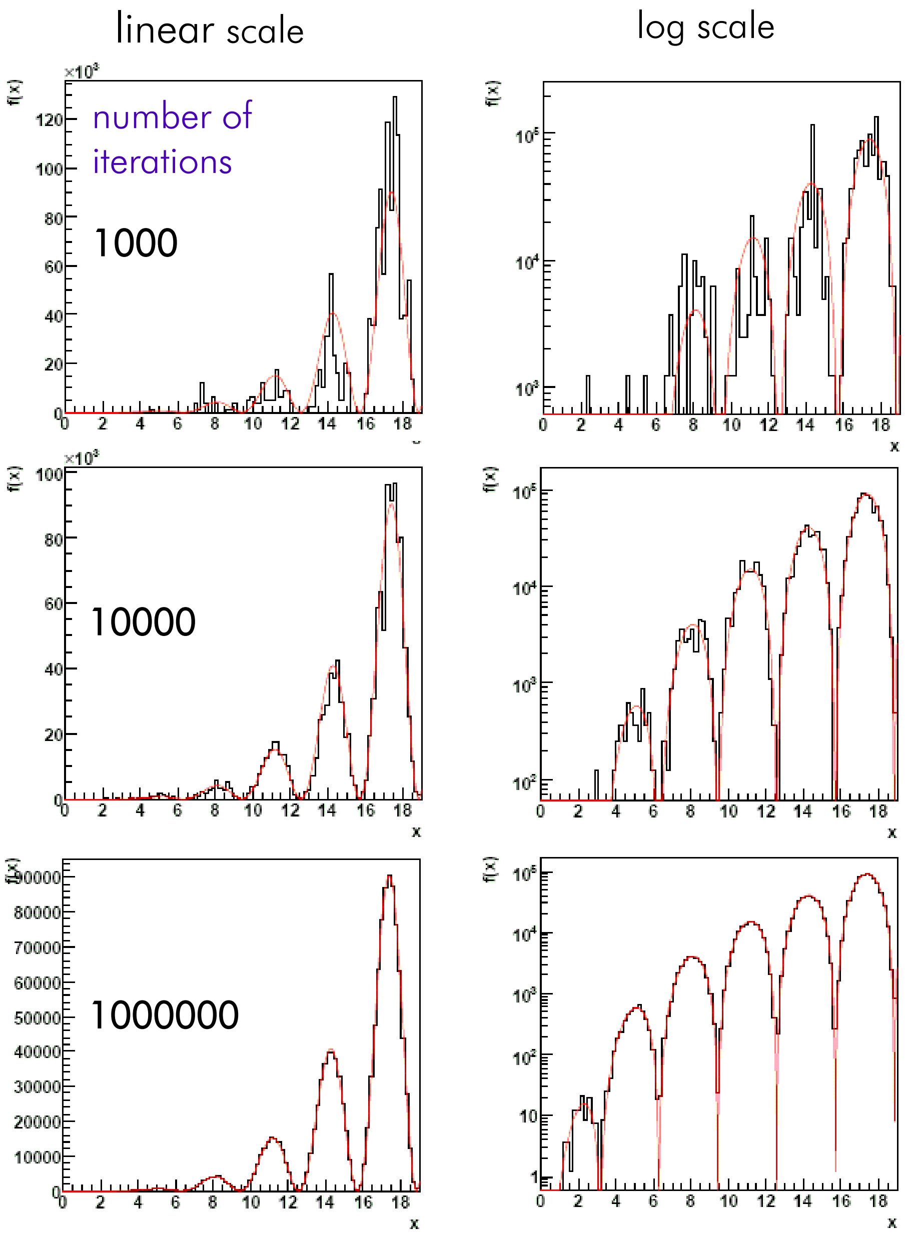
- distribution sampled by MCMC in this case quickly converges towards the underlying distribution

- mapping of complicated shapes with multiple minima and maxima

Note:

- MCMC has to become stationary to sample from underlying distribution

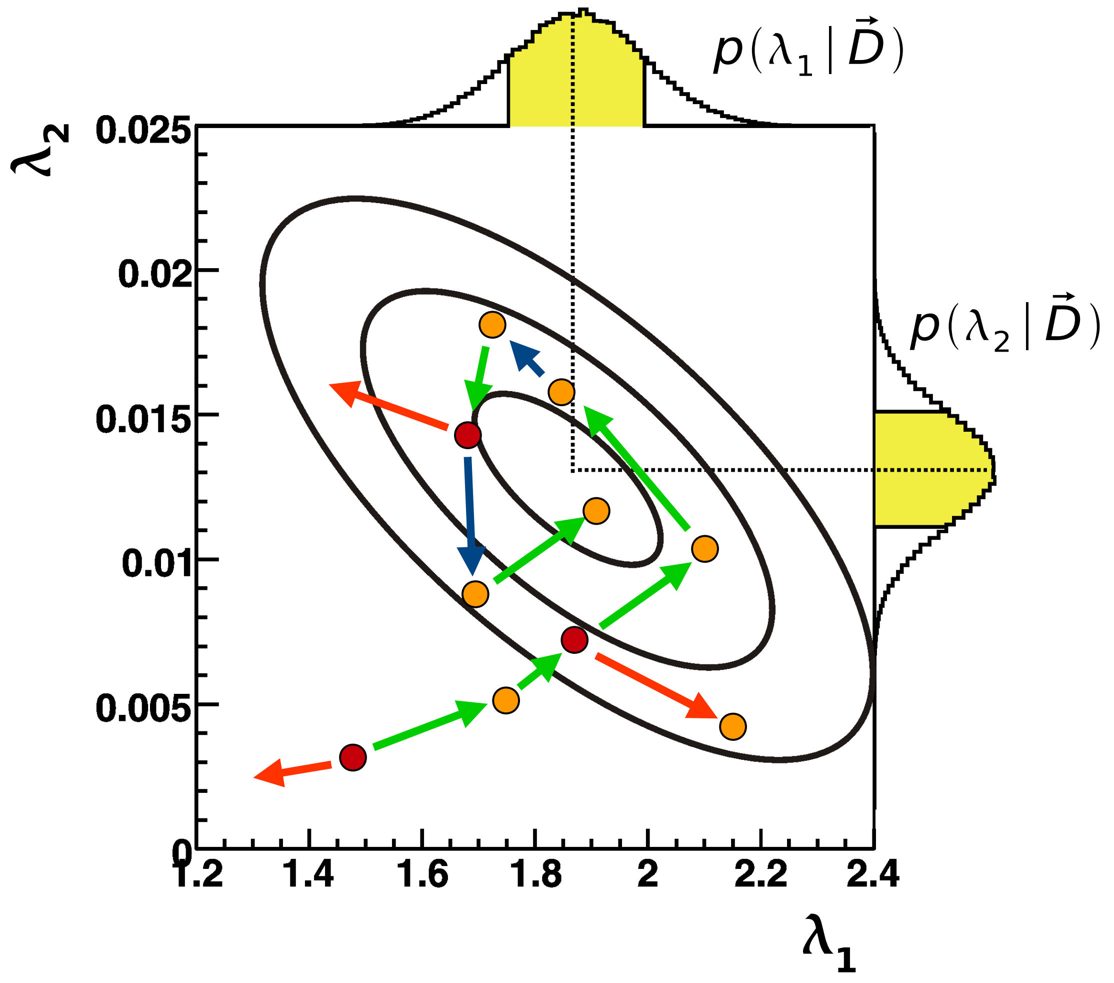
- in general the convergence is a non-trivial problem



Scanning parameter space with MCMC

- In Bayesian analysis use MCMC to scan parameter space of $\vec{\lambda}$
- $f(\vec{\lambda}) = p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})$
- MCMC converges towards underlying distribution
  - Determining of the overall probability distribution of the parameters $p(\vec{\lambda} \mid \vec{D})$
- Marginalize wrt. individual parameters while walking → obtain $p(\lambda_i \mid \vec{D}) = \int p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})\, d\vec{\lambda}_{j \neq i}$
- Find maximum (mode)
- Uncertainty propagation

$$p(\vec{\lambda} \mid \vec{D}) = \frac{p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})}{\int p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})\, d\vec{\lambda}}$$



Some details of MCMC implementation

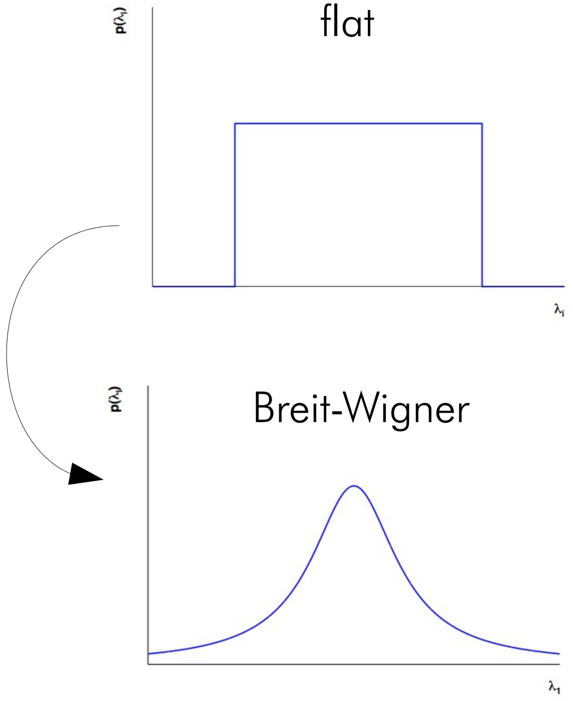
Running several chains in parallel (default is 5)

- Start at random locations in allowed parameter space
- Initialize chains by doing a pre-run to achieve convergence
  - Convergence defined using *r*-value (*Gelman & Rubin, StatSci 7, 1992*)
    - Essentially a ratio of the mean of the variances and the variance of the mean values of the chains for each parameter
    - Convergence criterion $|r-1| < 0.1$
- Steps in parameter space done consecutively for each parameter and chain
- Proposal function for new steps is a product of Breit-Wigner functions with varying widths
- The efficiency for accepting new point is evaluated for each parameter and chain over last min(npar*1000,10000) iterations and the widths are adjusted for all parameters to increase the performance
  - If efficiency > 50%, increase the width
  - If efficiency < 15%, decrease the width
- use MCMC only after pre-run has ended, convergence was reached and all parameter proposal widths have been adjusted

Most parameters can be set by the user

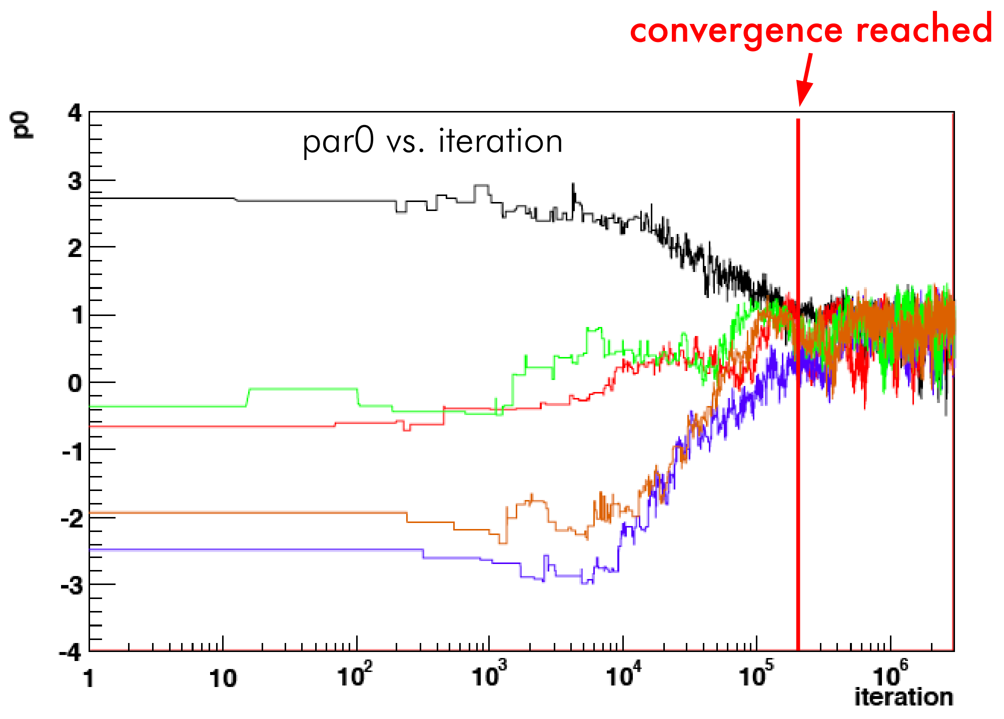
MCMC proposal function

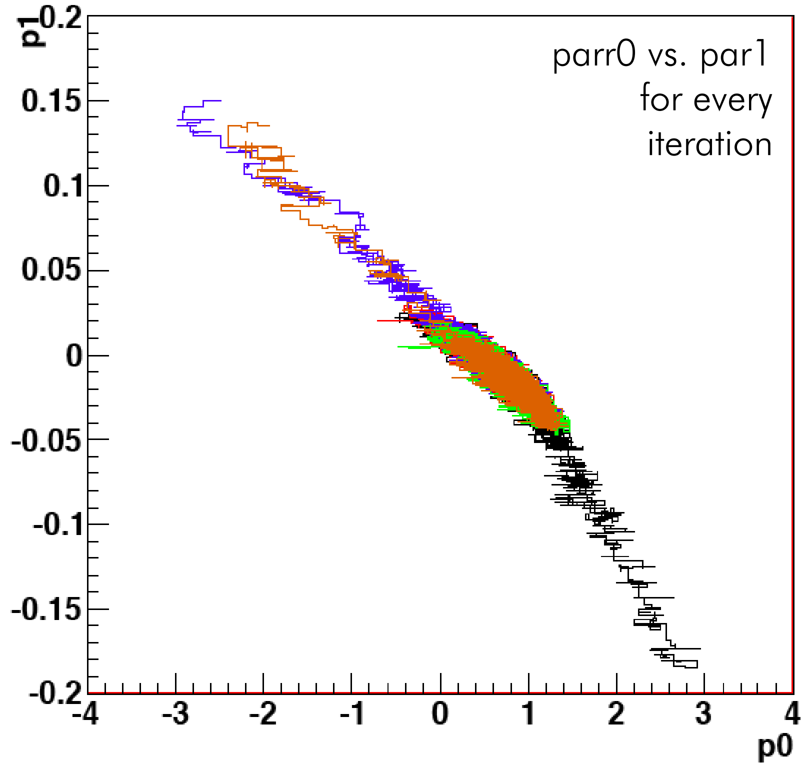
- by default using independent proposal functions for each parameter, i.e., new MCMC point for each parameter generated separately
- tested several different **proposal functions** on multi-peak distributions with deep valleys of probability density between them
  - **old default: flat distribution**
    - adjusting step size to achieve high efficiency
    - convergence problems
  - **new default: Breit-Wigner distribution**
    - adjusting width to achieve high efficiency
    - best convergence performance on all tested examples
  - Gaussian distribution
    - tails too low, convergence not very good
- it is possible to overload the proposal function with user defined function
  - either separately for every parameter
  - or with a single correlated function



Analysis of Markov Chain

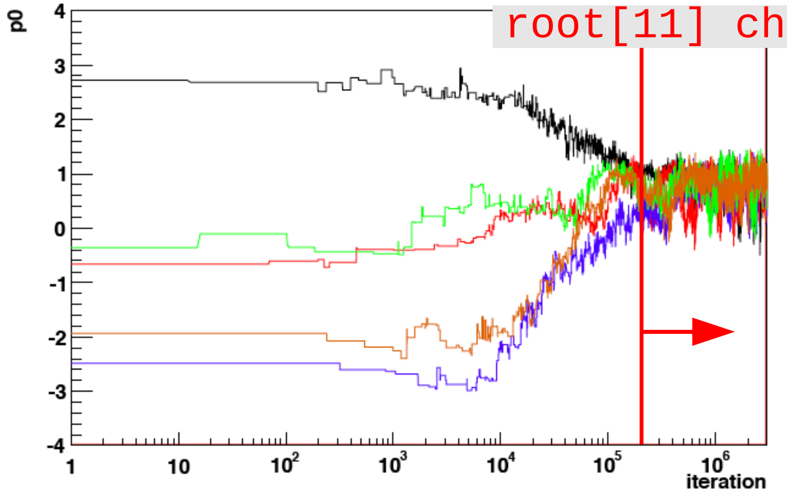
- danger of non-convergence still remains
- the full chain(s) can be stored for further analysis and parameter tuning as ROOT TTree(s)
  - allows direct usage of standard ROOT tools for analysis
- Markov Chain contains the complete information about the posterior (except for the normalization)

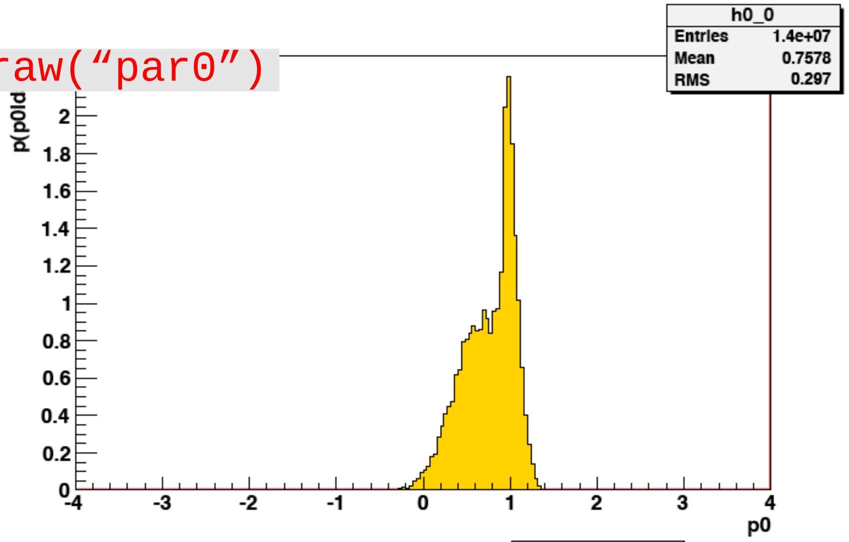




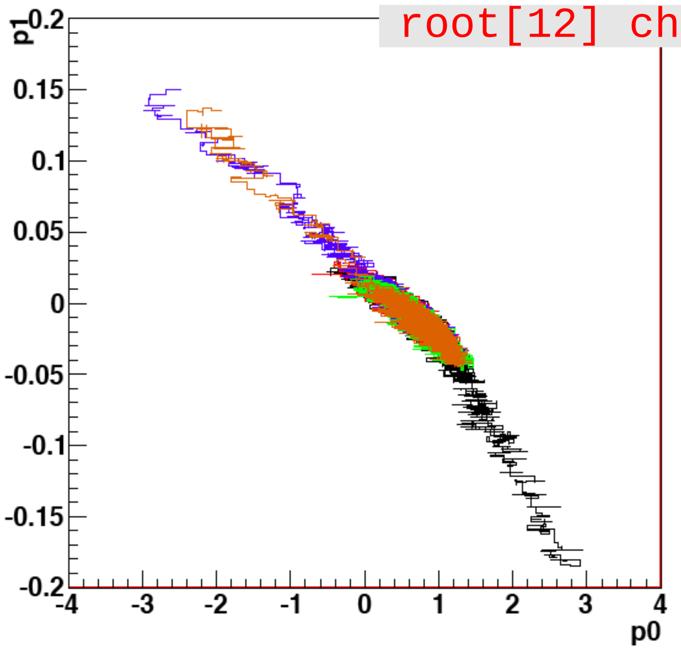
Obtaining marginalized distributions from TTree

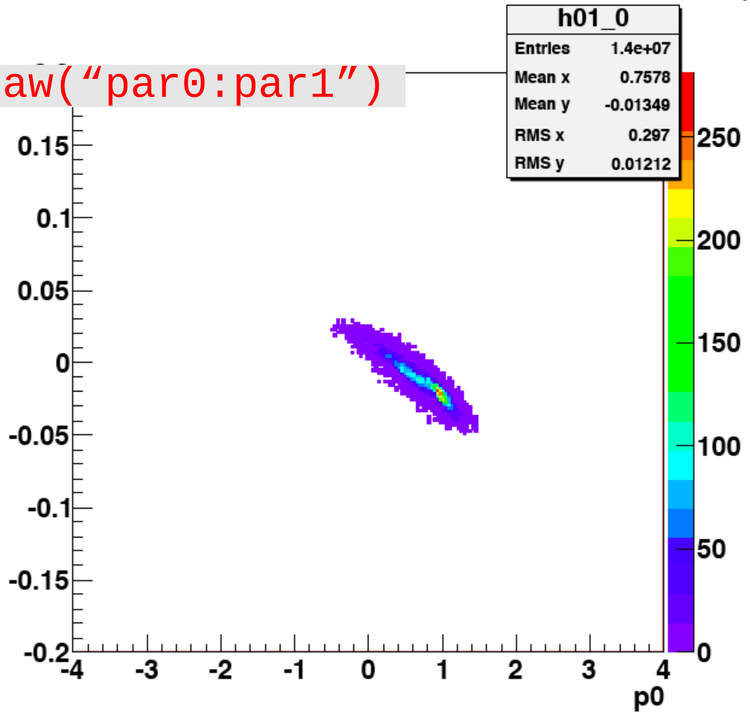
```
root[11] chain0 -> Draw("par0")
```





```
root[12] chain0 -> Draw("par0:par1")
```

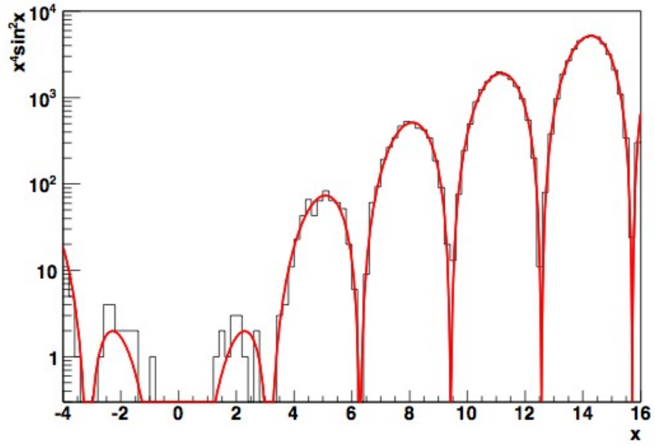


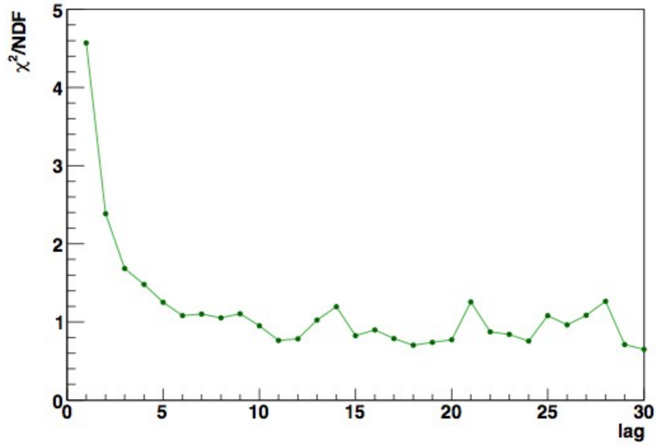


MCMC lag

- due to sampling nature of the MCMC, there is an autocorrelation between samples
  - newly generated point to some extent depends on the previous point, especially if step-size/range for generating new point is small
  - autocorrelation can be removed by "thinning" the sample – introducing a lag N
    - only every N-th sample from MCMC is used to generate the distribution
    - cost for better description of the mapped PDF is high – number of iterations has to be increased by factor N to reach the same statistical power
  - lag can be set in BAT (default is 1, i.e., no lag)
    - all calculations and filling of the distributions are only performed every N-th iteration
  - testing on various distributions shows improved MCMC description when using lag

MCMC mapping of function **$x^4\sin^2x$** overlayed with the function itself





χ^2 difference between the function **$x^4\sin^2x$** and the distribution generated by MCMC as a function of lag

Markov Chain in BAT

What's done within one run of the MCMC in BAT?

- for every iteration the histograms for all 1D and 2D marginalized distributions are filled (TH1Ds and TH2Ds)
  - large number of histograms: $Nparam\,(Nparam+1)\,/\,2$
    (e.g. for $Nparam=50$ there are 1275 histograms in total)
  - it is possible to switch off filling of individual distributions
- any function of parameters can be evaluated for every iteration
  - most natural candidate is uncertainty propagation
- since we're scanning parameter space, location of maximum can be found
  - not very efficient for maximization (minimization)
  - mode found in MCMC is generally an excellent starting point for other minimization algorithms
  - Minuit or Simulated Annealing can be used directly from BAT

BAT How-To

- BAT analysis can be run in compiled mode as well as from within an interactive ROOT session (ROOT macro)
  - examples distributed together with BAT show both types of use

- user model is defined by creating a model class that inherits from the base model class BCModel
  - define parameters and their ranges
  - define the **LogLikelihood()** method
  - define the **LogAPrioriProbability()** method
- in the main program use the functionality of BCModel to perform the analysis
- script **CreateProject.sh** from the BAT distribution creates several source files with a skeleton of a new model definition, main program and an appropriate Makefile

- in addition, several predefined models in BAT remove the need to define a model for frequent analysis problems: Fast fitters

Simple GraphFitter example

Simple ROOT macro:

```
   TGraphErrors * graph = ...;



   TF1 * f1 = new TF1(“f1”,”[0]+[1]*x”, 0., 100.);
   f1->SetParLimits(0, -20., 30.);
   f1->SetParLimits(1, .5, 1.5);

// BAT fitting bellow
   BCGraphFitter * gf = new BCGraphFitter(graph, f1);

   gf->Fit();


   gf->DrawFit("", true);


   gf->PrintAllMarginalized (“file.ps”);
   gf->PrintResults();
```

Data to fit are in a TGraphErrors object (uncertainties are necessary)

Function to fit is defined via TF1 object

Parameter ranges have to be specified !

Define BCGraphFitter, assume gaussian uncertainties

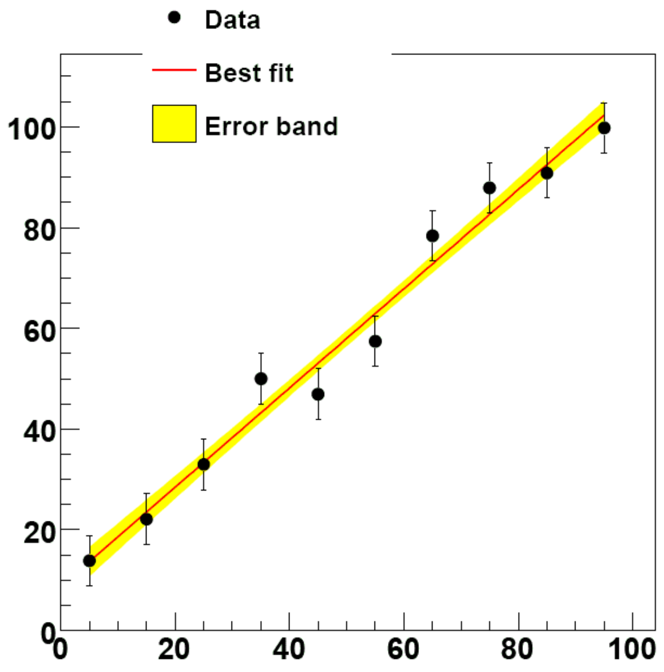
Fit the TGraphErrors with TF1 function using BAT

Draw the data, the best fit and the error band

Print all Marginalized distributions

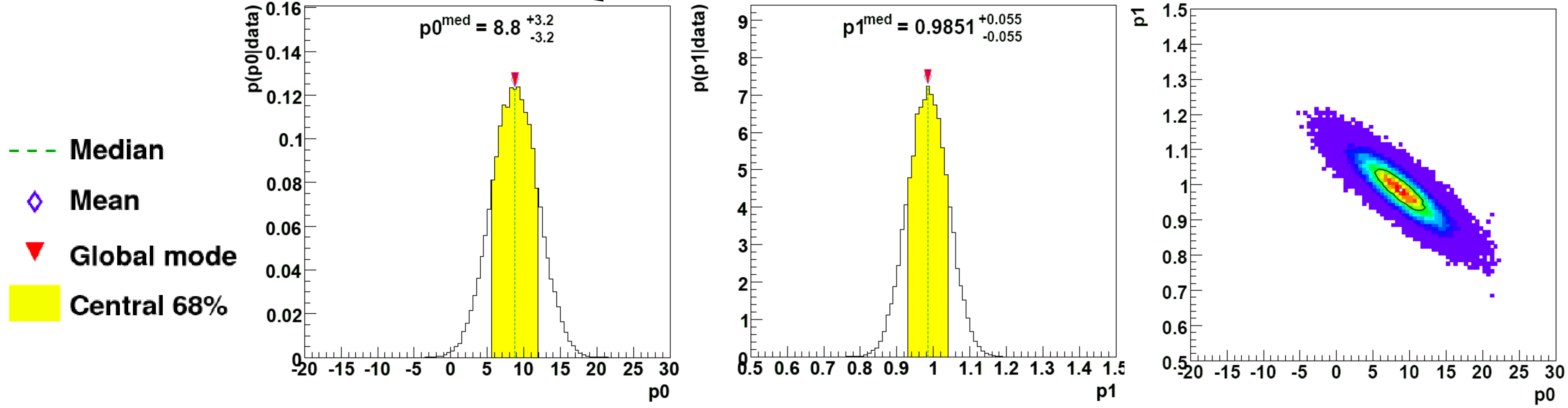
Print summary of results to text file

Graphical output



Fit and the error band representing the central 68%probability interval

Marginalized posterior probability densities



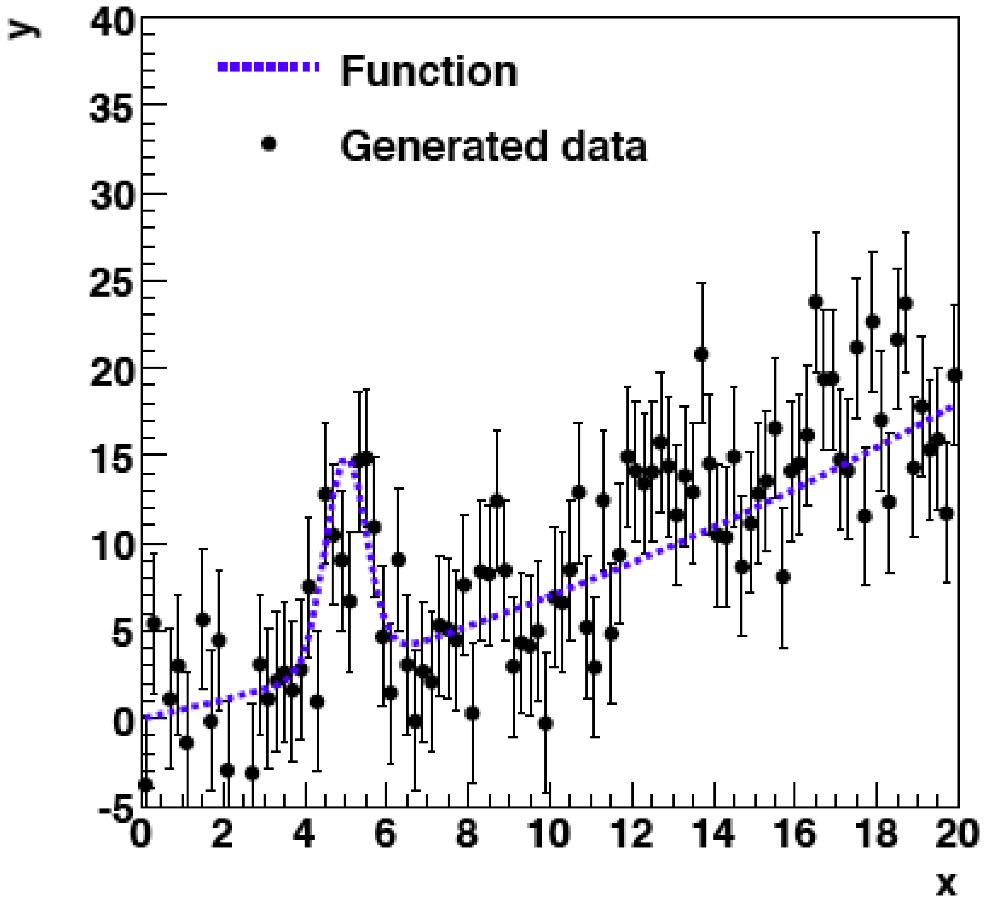
results.txt

```
-------------------------------------------------------
 Summary
-------------------------------------------------------

Model summary
==============
 Model: GraphFitter with f1
 Number of parameters: 2
 List of Parameters and ranges:
   (0) Parameter "p0": (-20, 30)
   (1) Parameter "p1": (0.5, 1.5)

 Results of the marginalization
=================================
 List of parameters and properties of the marginalized
 distributions:
  (0) Parameter "p0":
      Mean +- sqrt(V):              8.756 +- 3.152
      Median +- central 68% interval: 8.77 +  3.115 - 3.138
      (Marginalized) mode:           8.75
       5% quantile:                3.519
      10% quantile:                 4.693
      16% quantile:                 5.632
      84% quantile:                 12.01
      90% quantile:                 12.77
      95% quantile:                 13.91
      Smallest interval(s) containing 68% and local modes:
       (5.5, 12.5) (local mode at 8.75 with rel. height 1; rel. area 0.7)

  (1) Parameter "p1":
      ...
```

```
   ...
Results of the optimization
=============================
 Optimization algorithm used: Minuit
 List of parameters and global mode:
   (0) Parameter "p0": 8.779 +- 3.165
   (1) Parameter "p1": 0.9847 +- 0.05493

Results of the model test
===========================
p-value at global mode: 0.6913

Status of the MCMC
====================
 Convergence reached:                    yes
 Number of iterations until convergence: 6001
 Number of chains:                      5
 Number of iterations per chain:         100000
 Average efficiencies:
    (0) Parameter "p0": 22.62%
    (1) Parameter "p1": 20.22%


-------------------------------------------------------
Notation:
 Mean        : mean value of the marg. pdf
 Median      : maximum of the marg. pdf
 Marg. mode  : most probable value of the marg. pdf
 V          : Variance of the marg. pdf
 Quantiles   : most commonly used quantiles
-------------------------------------------------------
```

Another GraphFitter example

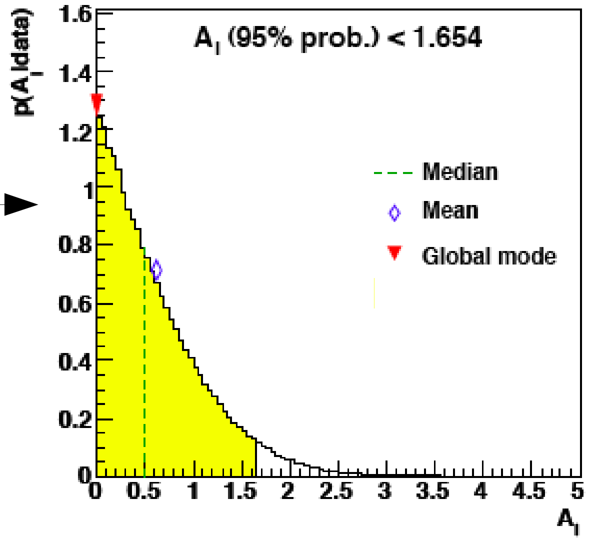
- Fit data set using:
  - I. 2nd order polynomial (no peak)
  - II. gaussian peak + constant
  - III. gaussian peak + straight line
  - IV. gaussian peak + 2nd order pol.

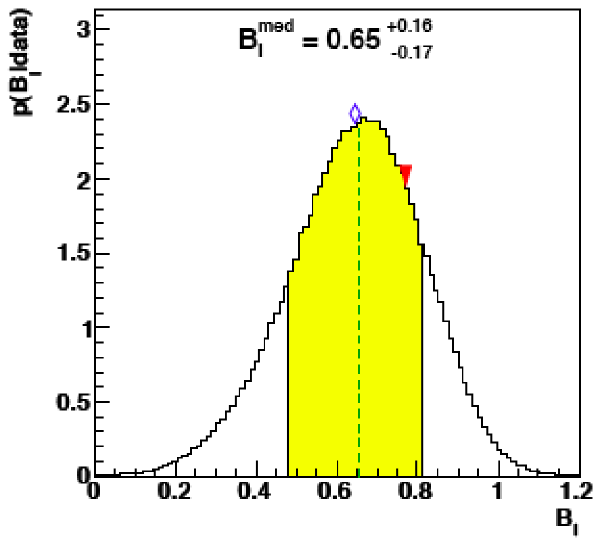


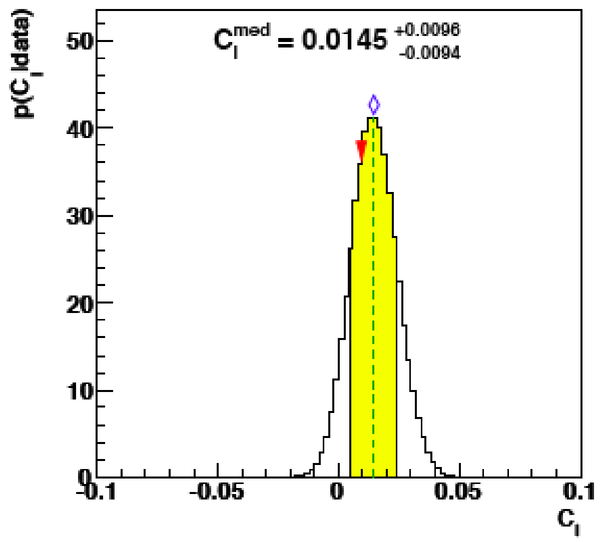
- Assume flat a priori probabilities in certain ranges of parameters, i.e. $p_0(\vec{\lambda}) = \text{const.}$
- Search for peak in range from 2. to 18. with maximum sigma of 4.
- Data were generated as gaussian peak + 2nd order polynomial

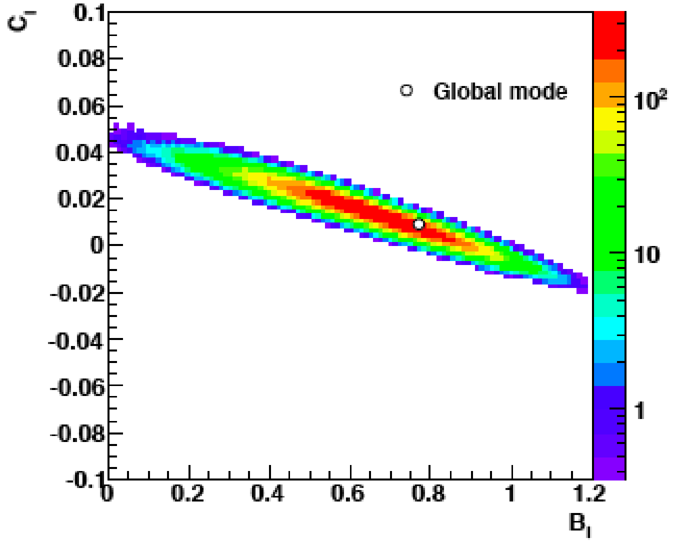
Marginalized posterior pdf
2nd order polynomial

95% limit $\longrightarrow$



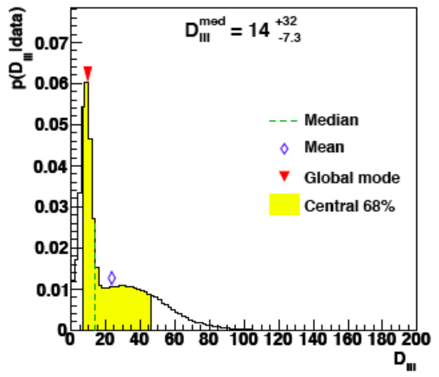


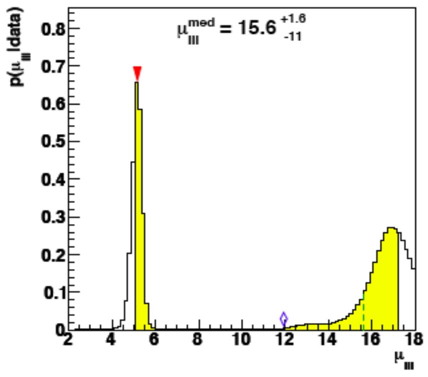


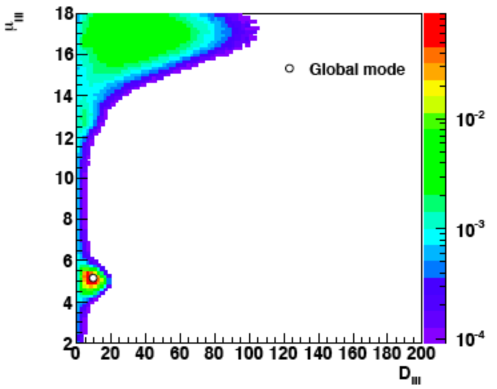


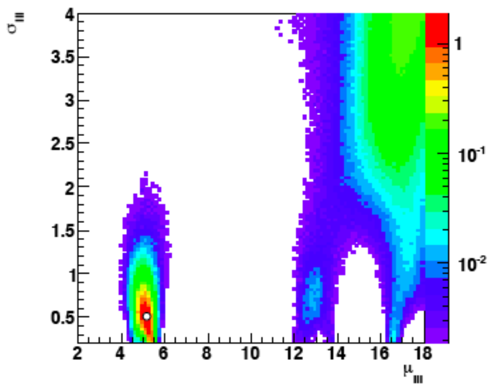
Marginalizad posterior pdf

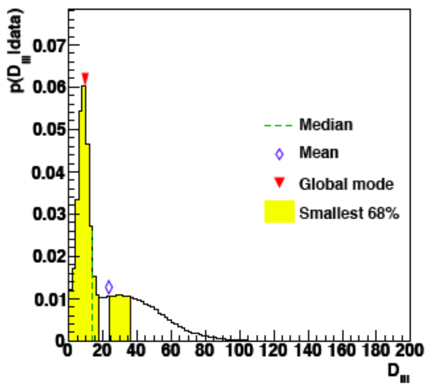
peak + straight line

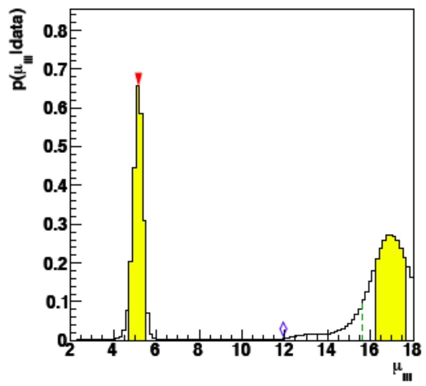










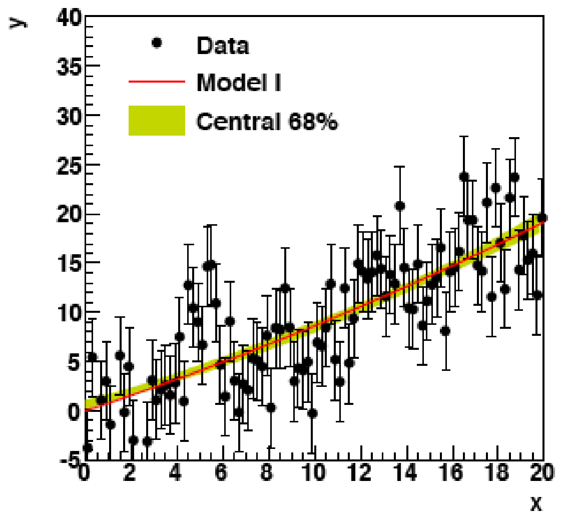


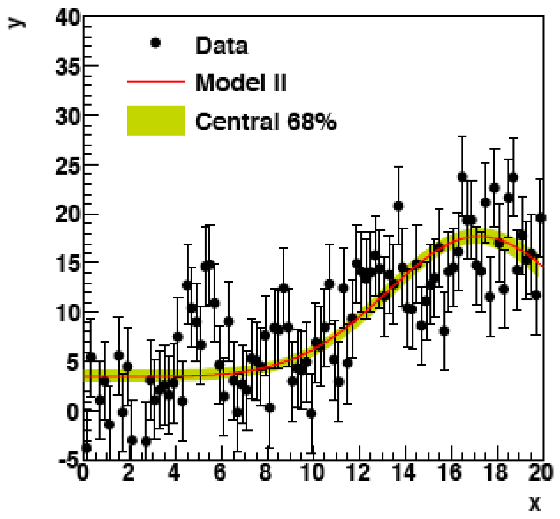
- Central interval not always optimal
- Multiple maxima in parameter space
- MCMC follows probability distributions with complicated shapes

- Optionally calculate smallest interval

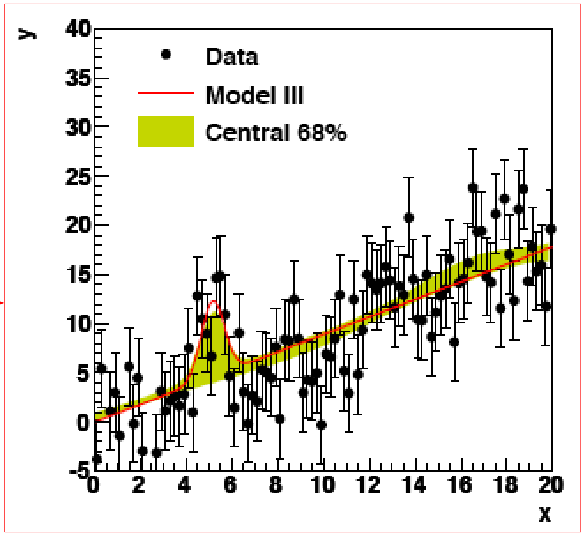
Fits + Uncertainty bands

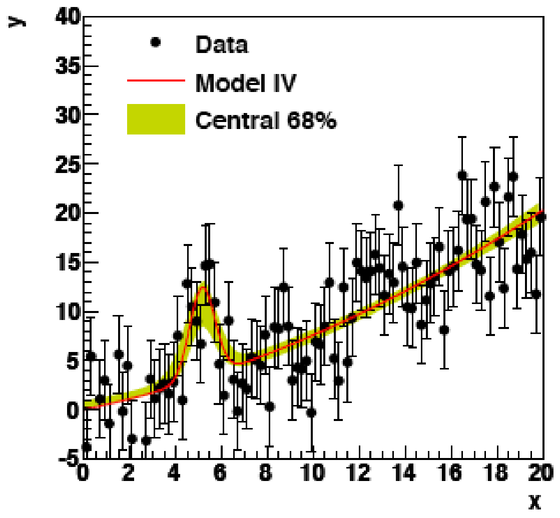
- Uncertainty band calculated during Markov Chain run
  - calculate $f(x)$ at many different x using λ sampled according to posterior
  - fill 2D histogram in (x,y)
  - after run look at distribution of y at given x and calculate central 68% interval





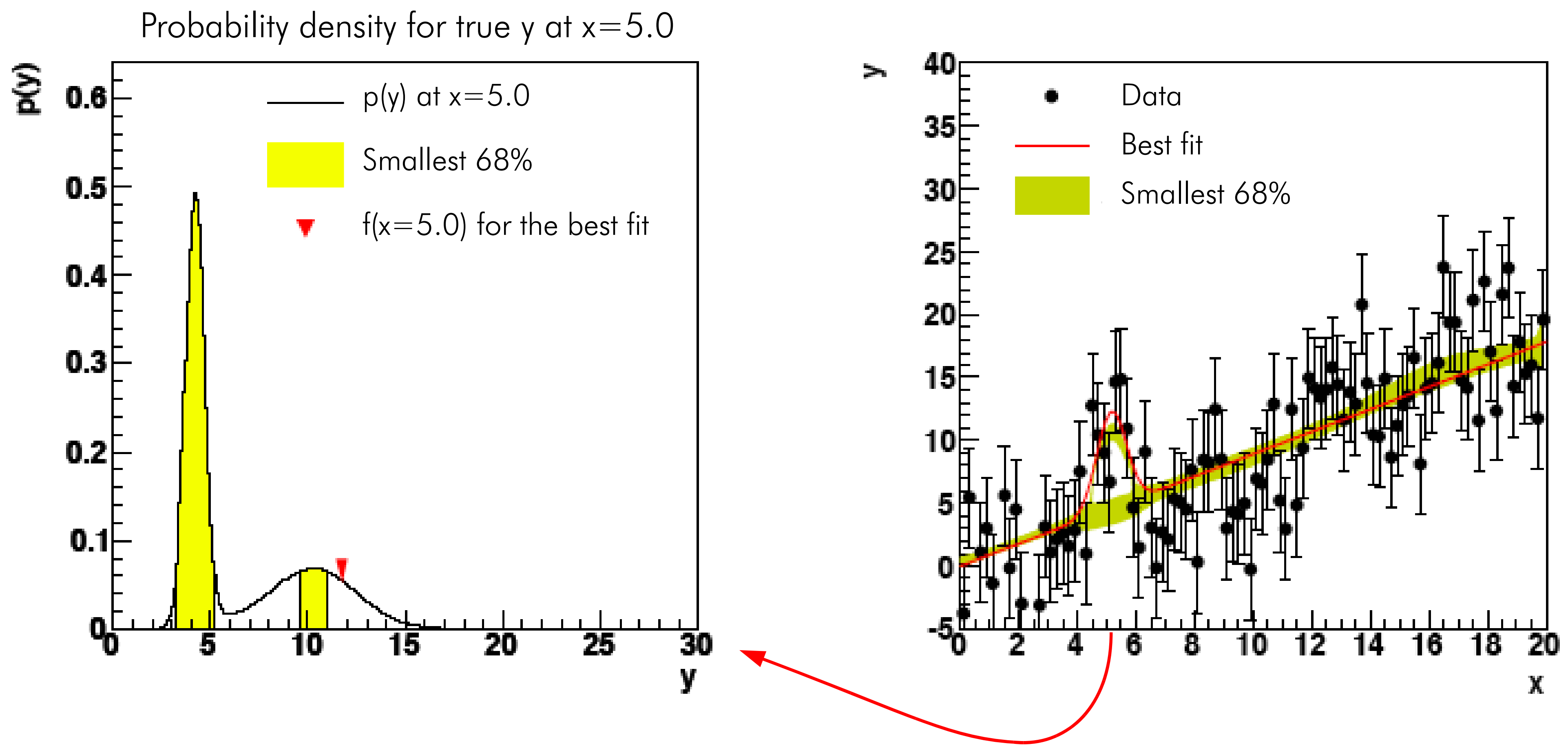
- Fit can lie outside of the central 68% interval
- Again, for multimodal distributions central 68% not optimal





Uncertainty multi-band

- Calculating f(x) at many x for every set of parameters sampled in the Markov chain ⇔ uncertainty propagation
- Can lead to multiple uncertainty bands



HistogramFitter example

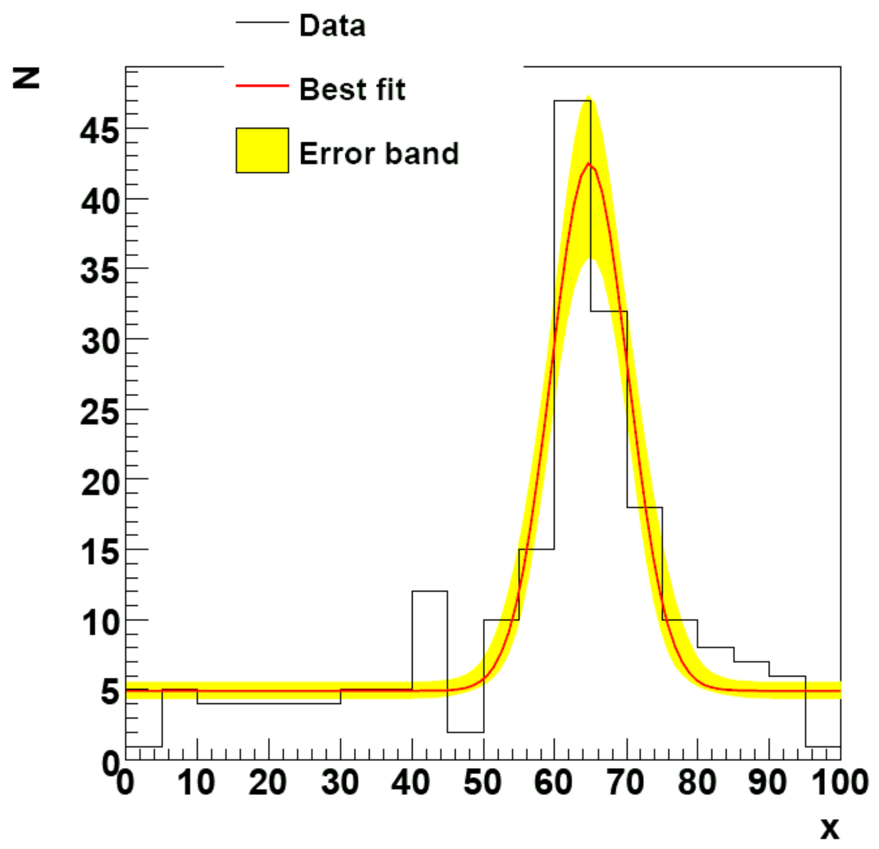
Simple ROOT macro:

- takes histogram as an input
- uses poissonian uncertainties

```
TH1D * histo = ...;
TF1 * f1 = ...;

BCHistogramFitter * hf =
     new BCHistogramFitter(histo, f1);
hf -> Fit();
hf -> DrawFit("", true);
...
```

define data histogram

define fit function and parameter ranges



Use whenever you want to fit a histogram.

EfficiencyFitter example

Simple ROOT macro:

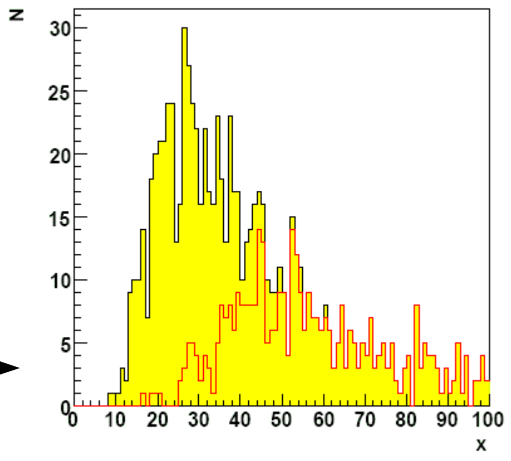
- takes two histograms as an input:
  - histogram with the whole set of events/entries
  - histogram with subset of events/entries
- uses binomial uncertainties

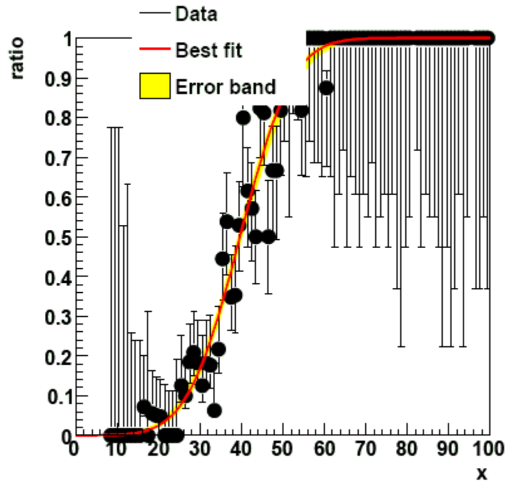
```
TH1D * hfull = ...;
TH1D * hsub = ...;
TF1 * f1 = ...;

BCEfficiencyFitter * ef = new BCEfficiencyFitter(hfull, hsub, f1);
ef -> Fit();
ef -> DrawFit("", true);
...
```

Use whenever you want to fit an efficiency.

Typical example:

→ fitting the trigger efficiency



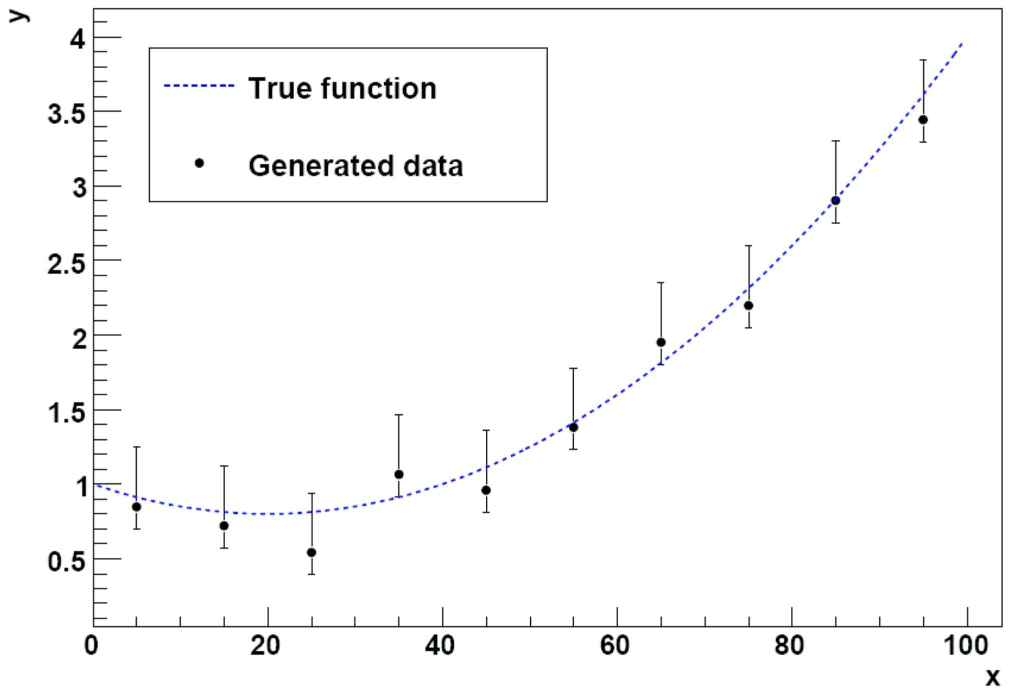


A few points to fast fitters

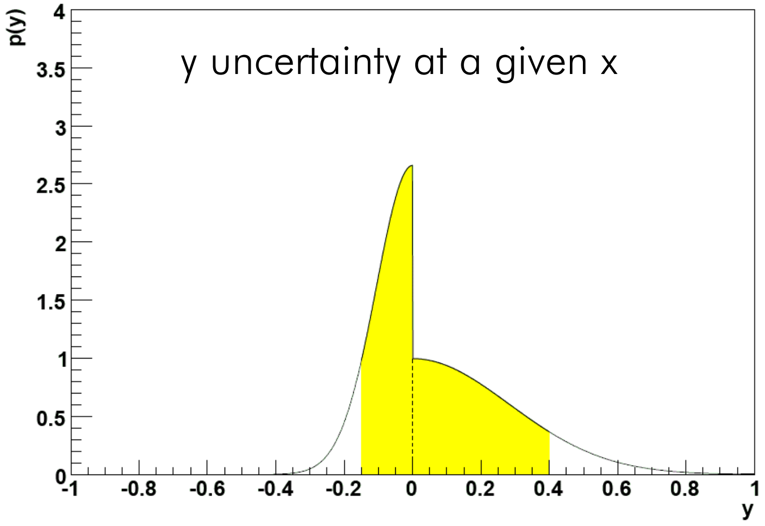
- **by default the priors for all parameters are flat !**
  - to set a different than flat prior one has to at the moment create a new class which inherits from a fast fitter class and overload the LogAPrioriProbability() method
  - in the next release it will be possible to set a prior for individual parameter as a TF1 object

- all fast fitters inherit from **BCModel** so they can use all it's functionality to also do a more sophisticated analysis
  - normalize
  - compare models
  - use fancy plotting
  - ...

Example with explicit model definition



- Data generated according to 2nd order polynomial
- Fit using straight line and 2nd order polynomial
- assuming 2 half gaussians for the description of the asymmetric errors
- **JUST TO ILLUSTRATE THE MODEL DEFINITION!**



Source code: Model definition

USER MODEL EXAMPLE – **2nd order polynomial** (model class)

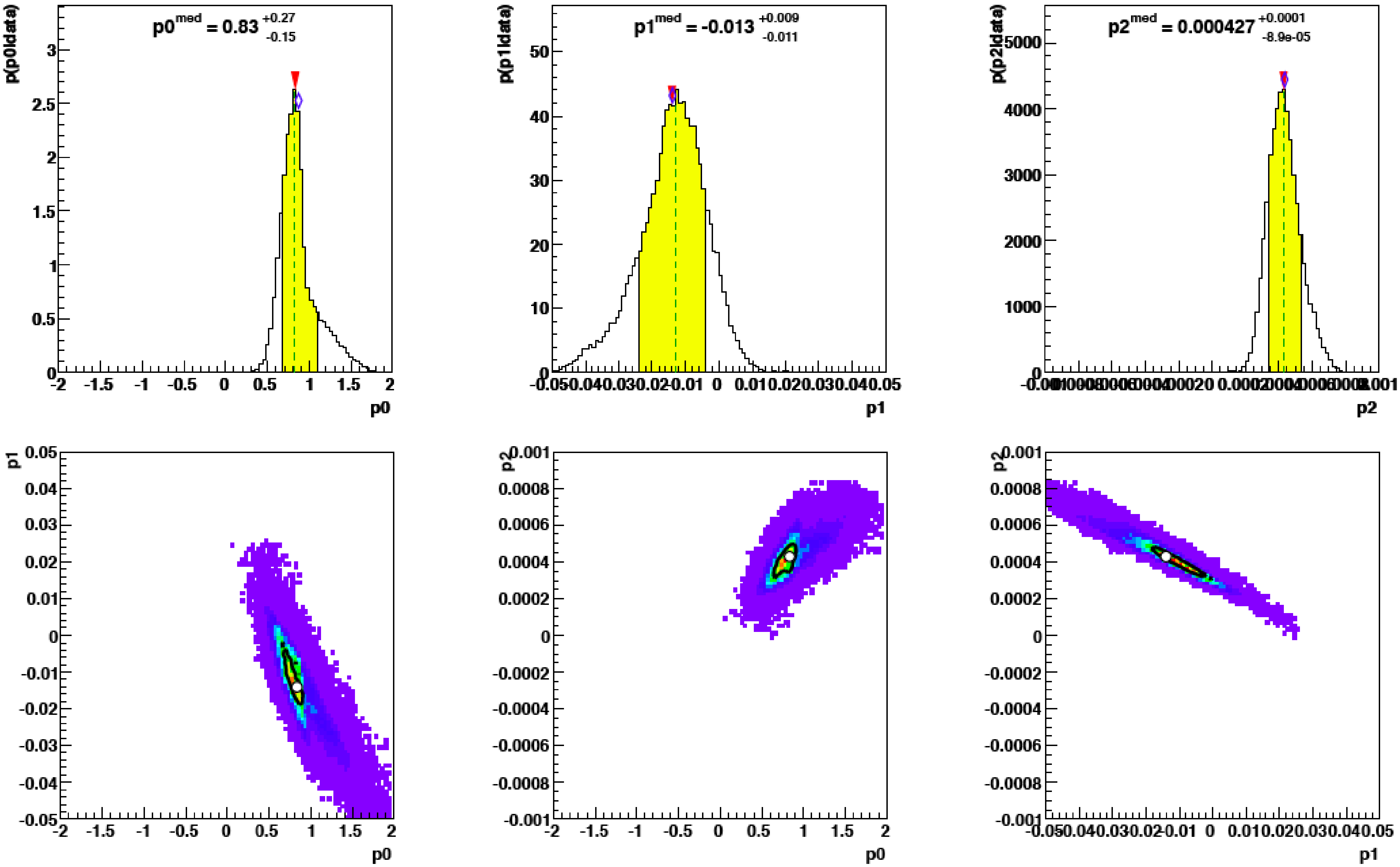
```
void ModelPol2::DefineParameters() {    // define parameters of the model
   this -> AddParameter(“A”,  0.,   5.);  // index 0
   this -> AddParameter(“B”,  0.,   1.2); // index 1
   this -> AddParameter(“C”, -0.1., 0.1); // index 2
}                                           // fit function is  f(x) = A + Bx + Cx^2

double ModelPol2::LogLikelihood(vector <double> params) { // define likelihood
   double lprob = 0.;
   double A = params[0], B = params[1], C = params[2];
   for(int i=0; i<this -> GetNDataPoints(); i++) {  // loop over all data points
      BCDataPoint * data = this -> GetDataPoint(i);
      double        x = data -> GetValue(0);
      double        y = data -> GetValue(1);
      double yerrdown = data -> GetValue(2);  // asymmetric errors on all points
      double   yerrup = data -> GetValue(3);

      double yexp     = A + x*B + x*x*C;  // calculate expectation value

      double yerr     = (y>yexp) ? yerrdown : yerrup; // decide which uncertainty is applicable

      lprob += BCMath::LogGaus(y, yexp, yerr, true);
   }
   return lprob;
}

double ModelPol2::LogAPrioriProbability(vector <double> params) { // define prior
   return 0.;  // flat prior probability for all parameters in their range; !!! not normalized !!!
}
```

Source code: Main program

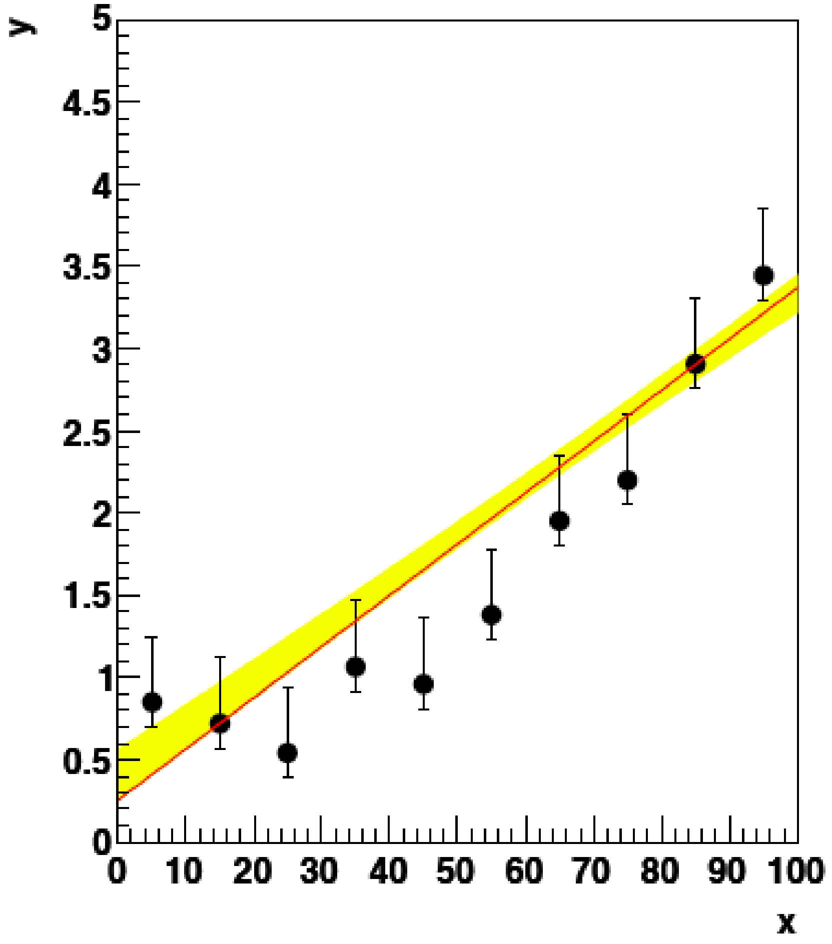
USER MODEL EXAMPLE – **2nd order polynomial** (simple main program)

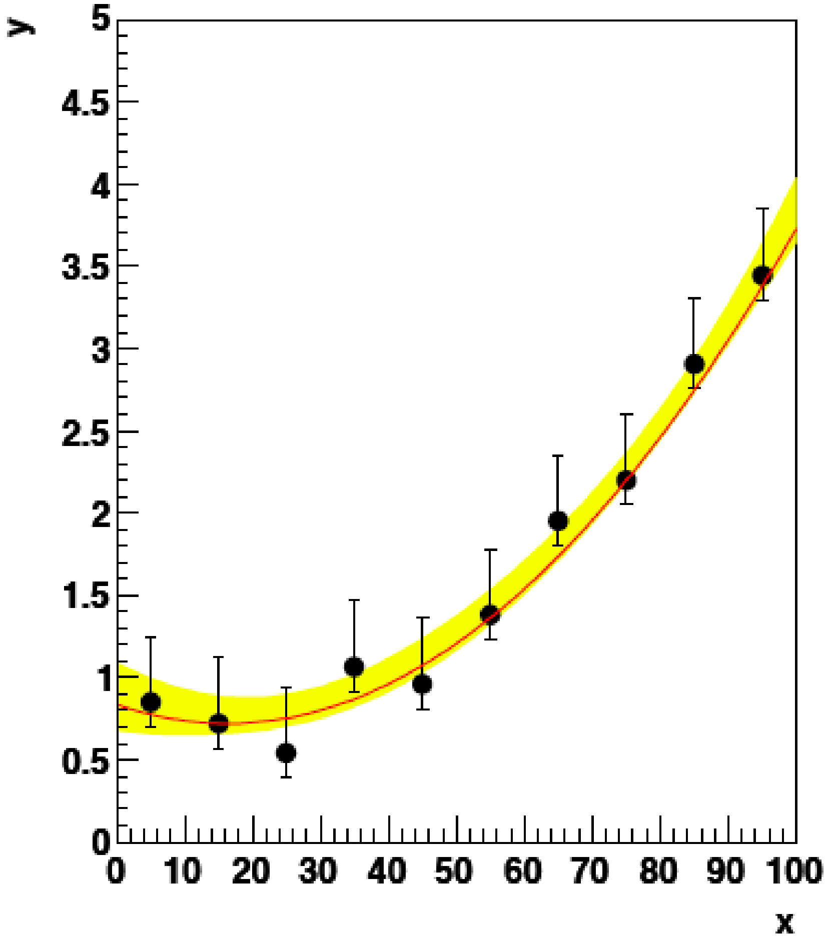
```
int main()
{
   ModelPol2 * mymodel = new ModelPol2(“2Dpol”);  // create model object

   DataSet * mydata = new DataSet(“measurement1”);  // create data object
   mydata->ReadDataFromFileTxT(“measurement1.dat”,4); // read in data, 4 columns: x,y,erup,erdn
   mymodel->SetDataSet(mydata);  // assign data to model

   // mymodel->Normalize();  // integrate to get the normalization

   mymodel->MarginalizeAll();  // marginalization
   mymodel->PrintAllMarginalized(“mymodel_all.ps”);

   mymodel->FindModeMinuit( mymodel -> GetBestFitParameters() ); // Mode finding using Minuit

   BCModelOutput * myout = new BCModelOutput(mymodel,”mymodel.root”);
   myout->WriteMarginalizedDistributions();
   myout->WriteErrorBand();
   myout->Close();

   mymodel->PrintResults();

   // add more things to do

   return 0;
}
```

1D and 2D marginalized distributions



Example with asymmetric errors
Fit + error band





Interface to RooStats

- class **BCRooInterface** implemented by Gregory Schott from RooStats team
  - no MCMC available in RooStats at that time
- allows to run complete BAT analysis starting from RooFit workspace
  - contains definitions of parameters and their ranges, likelihood, prior, data
  - full definition of a model in BAT

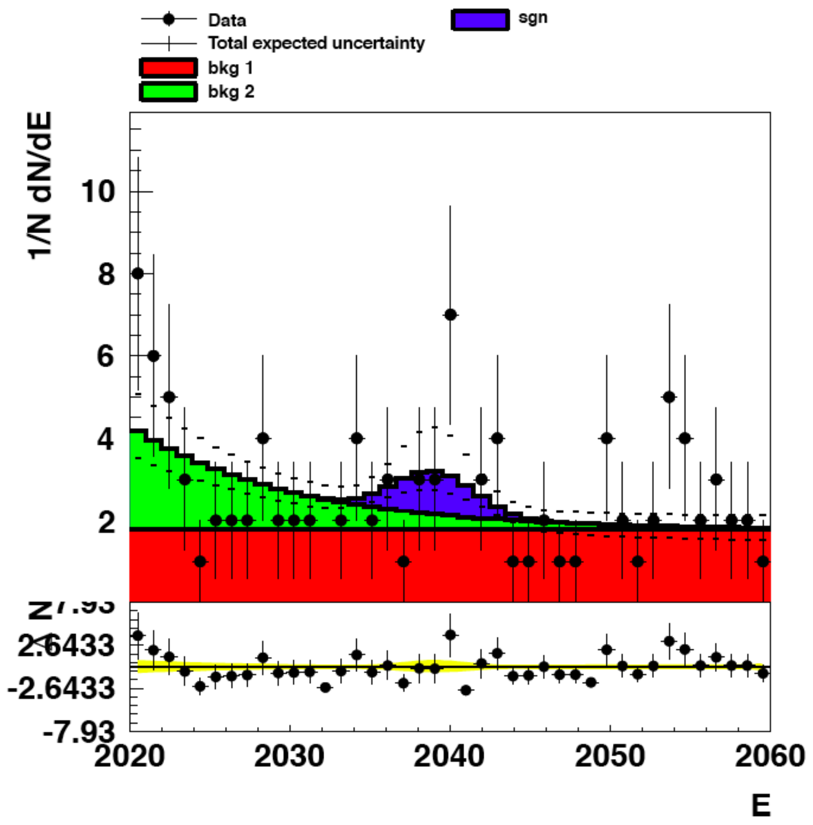
- work is ongoing on implementation of a complete interface to RooStats
  - allowing to call BAT from within RooStats during run-time
  - providing results in RooStats format
  - more or less transparent for RooStats users

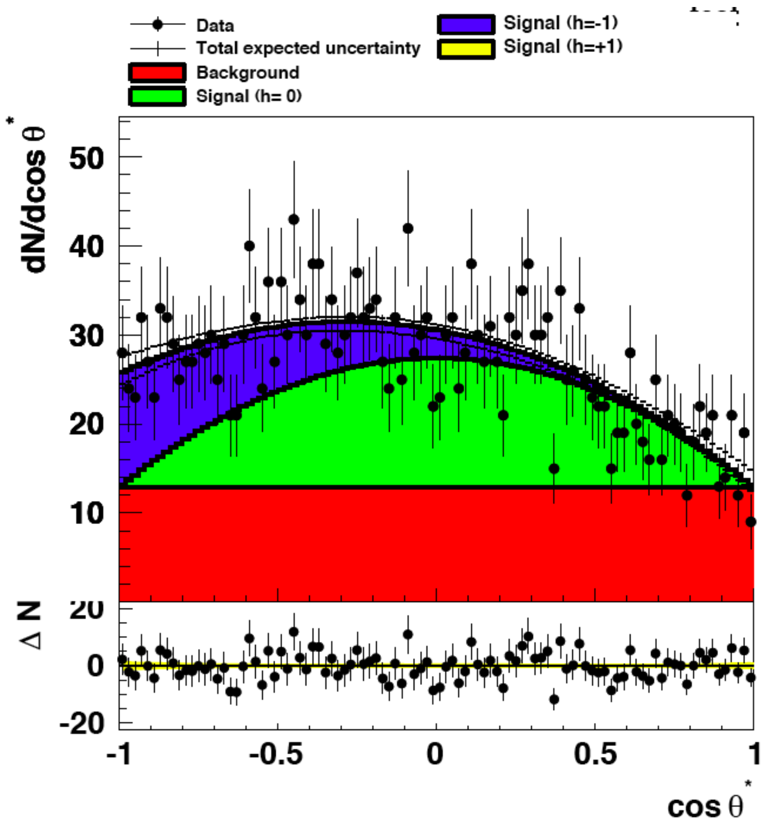
Next BAT release

- **current BAT version is 0.3.2**
- **next version 0.4 to show up within several weeks**
  - many updates planned
    - easy setup of priors without need of overloading LogAPrioriProbability() (especially useful for fast fitters)
    - new fast fitter class – TemplateFitter
    - new tool to provide better summary of results
    - some performance improvements
    - …

Template Fitter

- fit a set of templates to a histogram
    - e.g. a signal peak on top of a multi-component background
    - assuming that the shapes of all components are known
    - estimate the size of all contributions



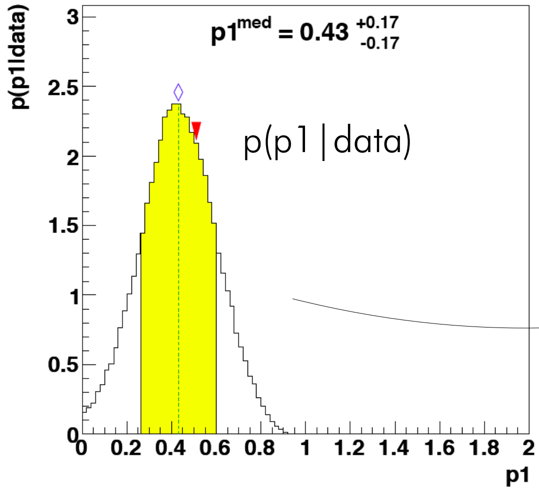


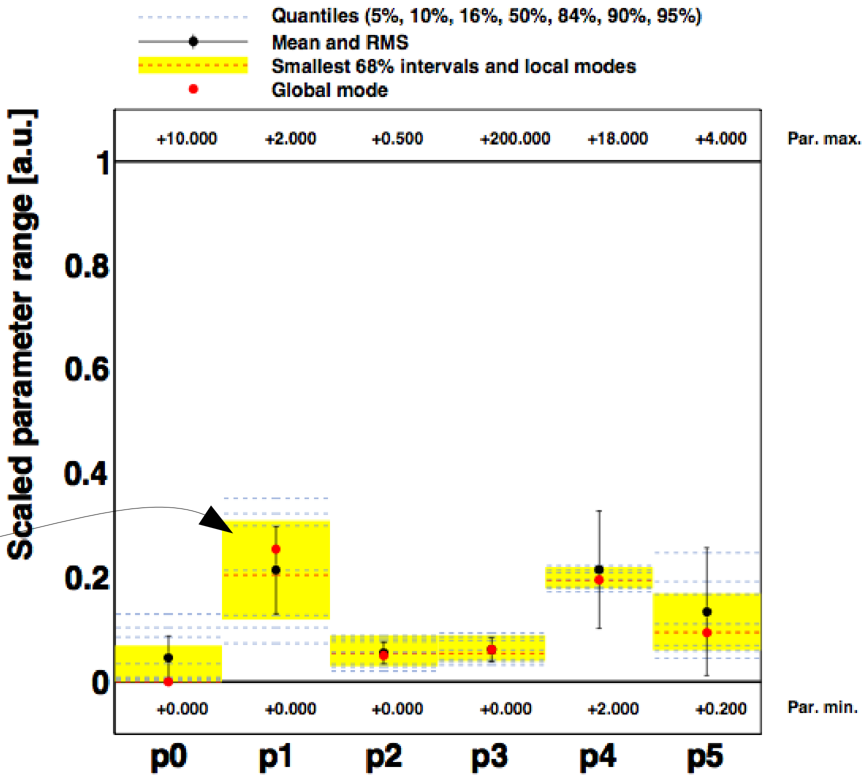
SummaryTool

- tool providing summary plots for easy overview of results without having to go through all the plots of marginalized distributions
  - particularly useful for models with large number of parameters
- easy to use:

```
SummaryTool * summaryM1 = new SummaryTool();
summaryM1->SetModel(model1);
summaryM1->PrintParameterPlot("par_M1.eps");
summaryM1->PrintCorrelationPlot("corr_M1.eps");
...
```

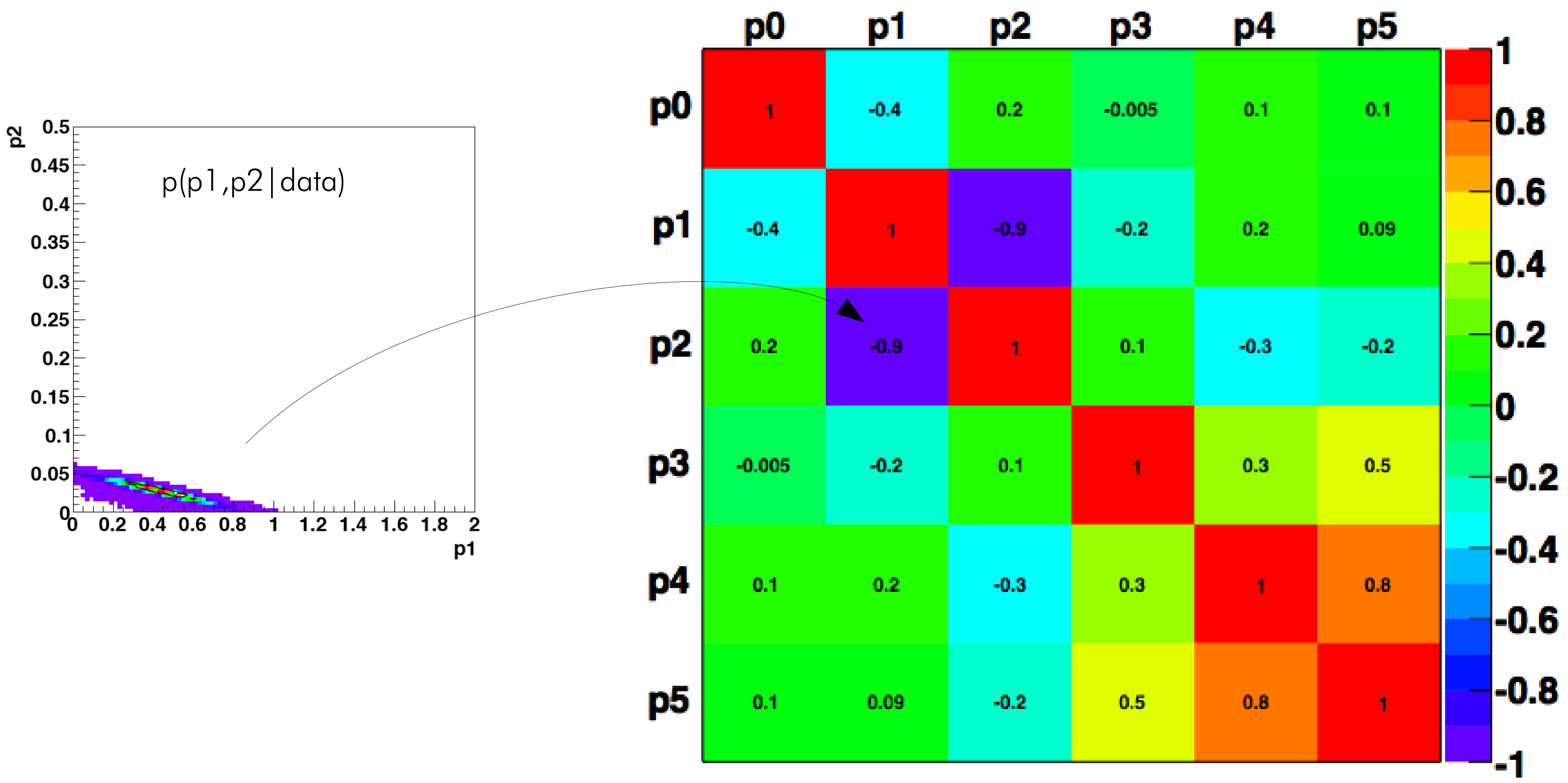
- summary of single parameter posterior PDFs





SummaryTool – Correlation Matrix

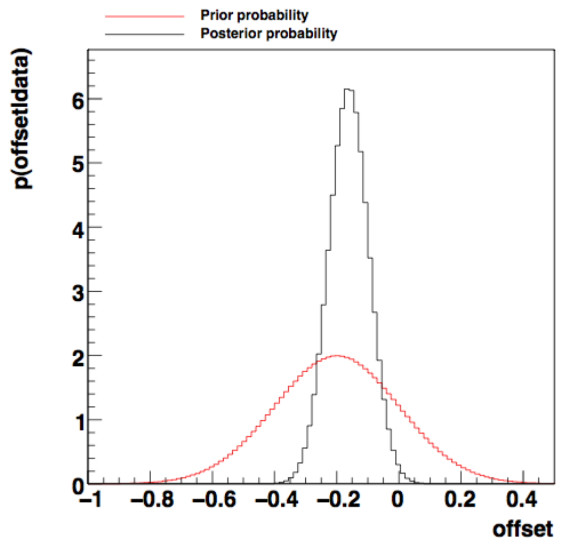
- summarizes Npar*(Npar-1)/2 posterior distributions marginalized wrt. combination of any two parameters of the model (for Npar=50 it's ~ 1000 distributions) into a single plot

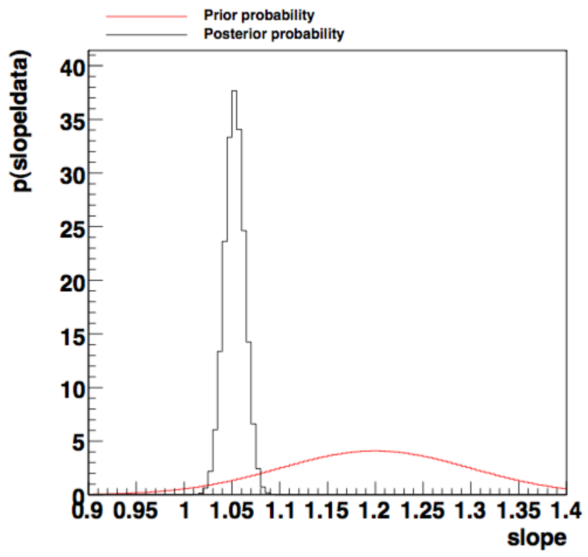


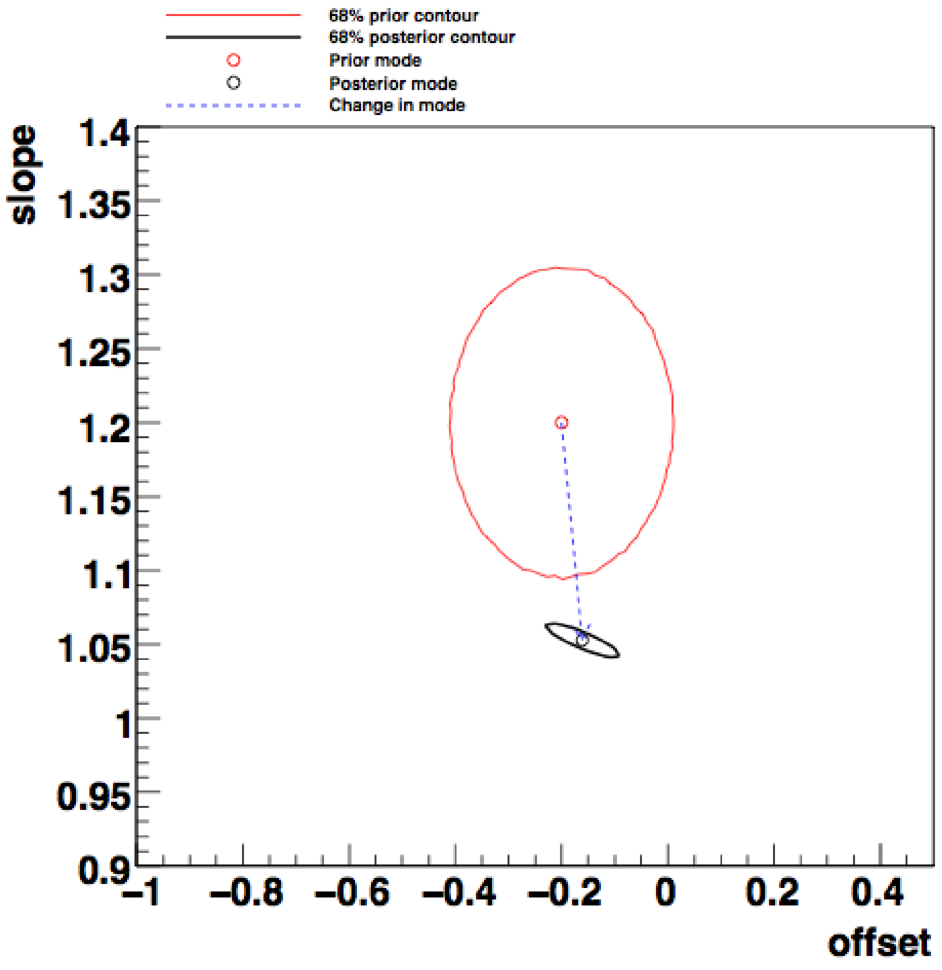
	p0	p1	p2	p3	p4	p5
p0	1	-0.4	0.2	-0.005	0.1	0.1
p1	-0.4	1	-0.9	-0.2	0.2	0.09
p2	0.2	-0.9	1	0.1	-0.3	-0.2
p3	-0.005	-0.2	0.1	1	0.3	0.5
p4	0.1	0.2	-0.3	0.3	1	0.8
p5	0.1	0.09	-0.2	0.5	0.8	1

SummaryTool – Knowledge update

- comparison of prior and posterior for 1D and 2D distributions



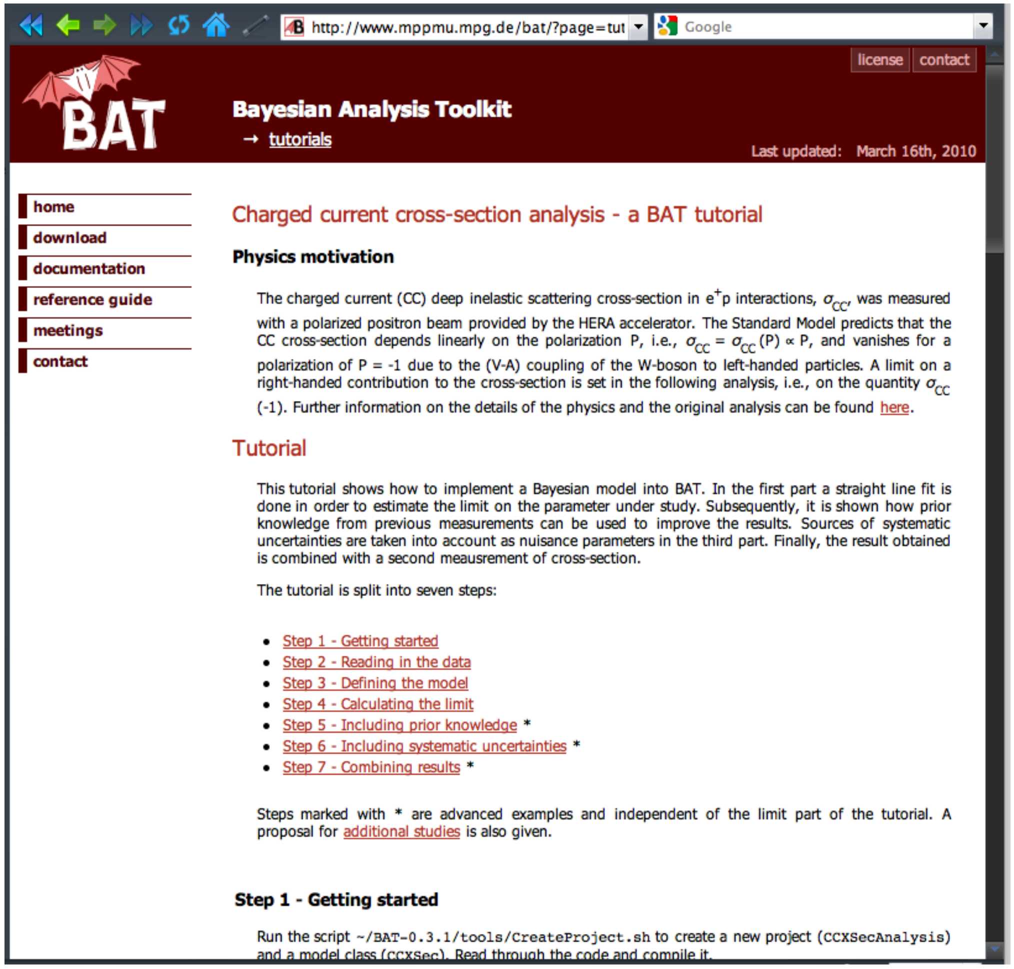




BAT

Tutorials

- tutorials section was recently added to the BAT webpage
  - can be found under: Documentation $\longrightarrow$ Tutorials
- several tutorials available
- show basic information on how to
  - set up a model
  - calculate limits
  - define prior
  - include systematic uncertainties
  - etc.
- tutorials have form of exercises with solutions
- more tutorials will come



Bayesian Analysis Toolkit
→ tutorials
Last updated: March 16th, 2010

Charged current cross-section analysis - a BAT tutorial

Physics motivation

The charged current (CC) deep inelastic scattering cross-section in e^+p interactions, σ_{CC}, was measured with a polarized positron beam provided by the HERA accelerator. The Standard Model predicts that the CC cross-section depends linearly on the polarization P, i.e., $\sigma_{CC} = \sigma_{CC}(P) \propto P$, and vanishes for a polarization of P = -1 due to the (V-A) coupling of the W-boson to left-handed particles. A limit on a right-handed contribution to the cross-section is set in the following analysis, i.e., on the quantity σ_{CC}(-1). Further information on the details of the physics and the original analysis can be found here.

Tutorial

This tutorial shows how to implement a Bayesian model into BAT. In the first part a straight line fit is done in order to estimate the limit on the parameter under study. Subsequently, it is shown how prior knowledge from previous measurements can be used to improve the results. Sources of systematic uncertainties are taken into account as nuisance parameters in the third part. Finally, the result obtained is combined with a second meausrement of cross-section.

The tutorial is split into seven steps:

- Step 1 - Getting started
- Step 2 - Reading in the data
- Step 3 - Defining the model
- Step 4 - Calculating the limit
- Step 5 - Including prior knowledge *
- Step 6 - Including systematic uncertainties *
- Step 7 - Combining results *

Steps marked with * are advanced examples and independent of the limit part of the tutorial. A proposal for additional studies is also given.

Step 1 - Getting started

Run the script `~/BAT-0.3.1/tools/CreateProject.sh` to create a new project (`CCXSecAnalysis`) and a model class (`CCXSec`). Read through the code and compile it.

Future plans

- we're starting to be limited by the current structure of the code especially when trying to implement new algorithms
  - we're preparing reworked BAT with new internal structure which should allow much easier extensibility
  - slowed down by the required maintenance of the current BAT + user requests
  - not too much manpower until now, but this will change soon
- many things to implement
  - objective priors, reference priors
  - different sampling algorithms
  - extended proposal functions
- more predefined models for frequent problems for quick use
- feature requests are accepted
  - you can also contribute yourself
  - if you make an analysis using BAT and it's addressing a general problem, it might be worth to consider making it available as a predefined class for easy use for everyone

Visit http://www.mppmu.mpg.de/bat for more info, updates, documentation, examples, etc.