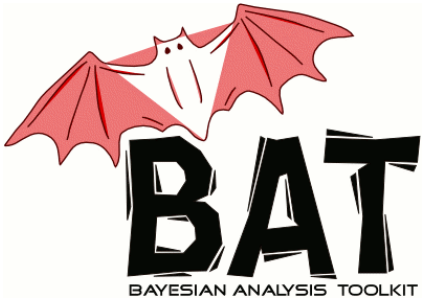


Technische Universität München

The Bayesian Analysis Toolkit (BAT) – a complex MCMC application

IN2P3 School of Statistics, Autrans, May 29, 2014

Daniel Greenwald, Technische Universität München

The BAT men: Frederik Beaujean,
Allen Caldwell, Daniel Greenwald,
Daniel Kollar, Kevin Kröninger

Aims

Provide a flexible and modular framework for statistical models
in the context of a Bayesian interpretation

Provide a set of (mostly numerical) methods
to solve data-analysis problems
(parameter estimation, limit setting, model comparison, goodness-of-fit tests, etc.)

Scope

Developed in experimental-particle-physics community
(explains choice of C^{++} and ROOT dependence)

Extended to other fields of research
(phenomenology, medicine, astroparticle physics, etc.)

Phrase arbitrary models and use data sets

- C++ library based on ROOT
- Models implemented as base classes
- Easy to interface to any existing code

(interesting for complex fitting: fits of CKM parameters, SM parameters, …)

Perform data analysis tasks

- Graphical output via ROOT core functionality
- Point estimation done using Minuit and Simulated Annealing
- Interval estimation and uncertainty propagation done using MCMC
- Model comparison via Bayes factors or evidence calculation using interface to Cuba
  (Cuba is a collection of integration methods, e.g., VEGAS)

Implementation

COMMON METHODS

- Normalize
- Find mode / fit
- Test the fit
- **Marginalize w/r/t several parameters**
- Compare models
- Provide nice output

Usage of MCMC in Bayesian inference

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

USER DEFINED

Read DATA

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

interface to user-defined software

Define MODEL

define parameters $\vec{\lambda}$

define likelihood $p(\vec{D} \mid \vec{\lambda})$

define priors $p_0(\vec{\lambda})$

IN YOUR CODE

```
BCModel::LogLikelihood(const std::vector<double> & parameters)
BCModel::LogAPrioriProbability(const std::vector<double> & parameters)
```

Usage of MCMC in Bayesian inference

Use MCMC to sample the posterior probability, i.e.

$$f(\vec{\lambda}) = p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})$$

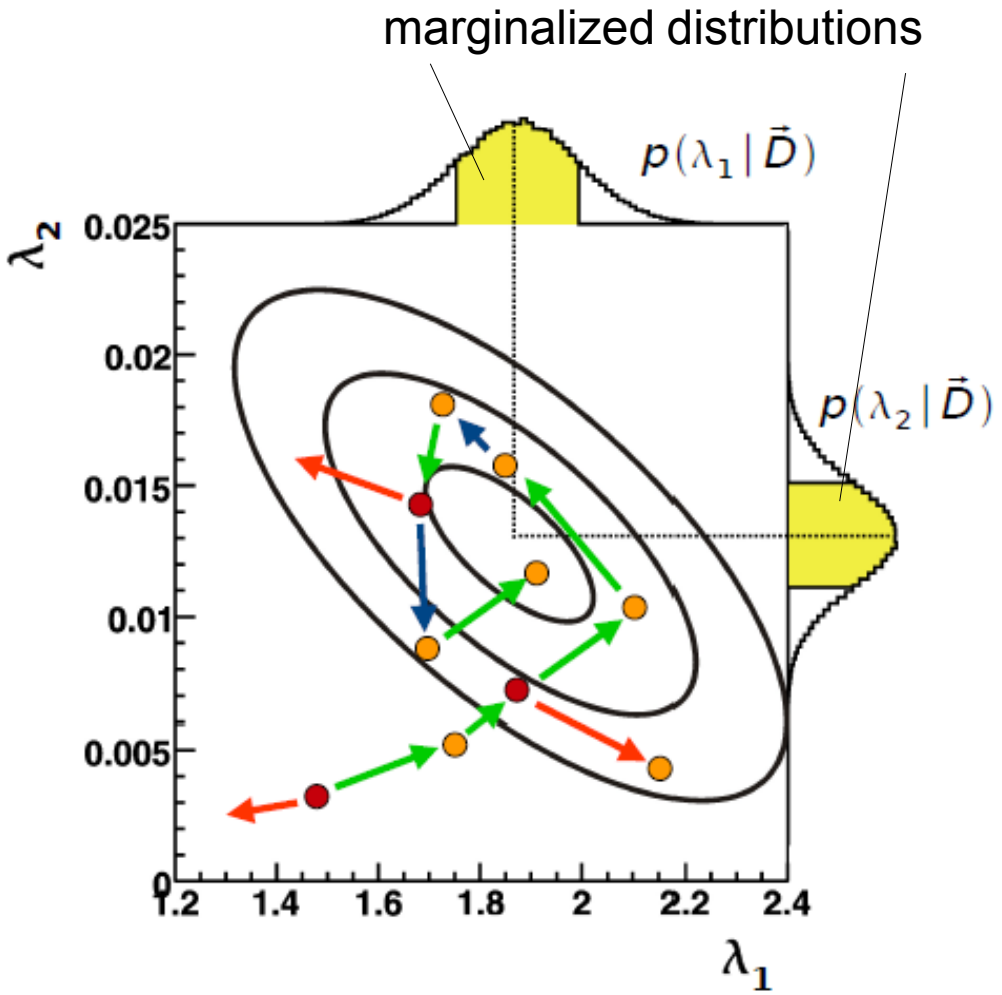
Marginalization of posterior:

$$p(\lambda_i \mid \vec{D}) = \int p(\vec{D} \mid \vec{\lambda})\, p_0(\vec{\lambda})\, d\vec{\lambda}_{j \neq i}$$

Fill a histogram with just one coordinate while sampling

Uncertainty propagation:
calculate any function of the parameters while sampling

Point estimation:
find mode while sampling



Step 1: Starting values

Either random within parameter space (default)

or center of each dimension

or user-defined

Step 2: Burn-in phase

Use multiple chains (by default 5)

Run until convergence is reached and chains are efficient

Convergence is reached if inter- and intra-chain variances are equal
(Gelman and Rubin criterion [Gelman & Rubin, StatSci 7, 1992])

Chains are efficient if the efficiency is between 15% and 50%

Run in sequences to adjust the width of the proposal functions:

efficiency > 50% → increase width
efficiency < 15% → decrease width

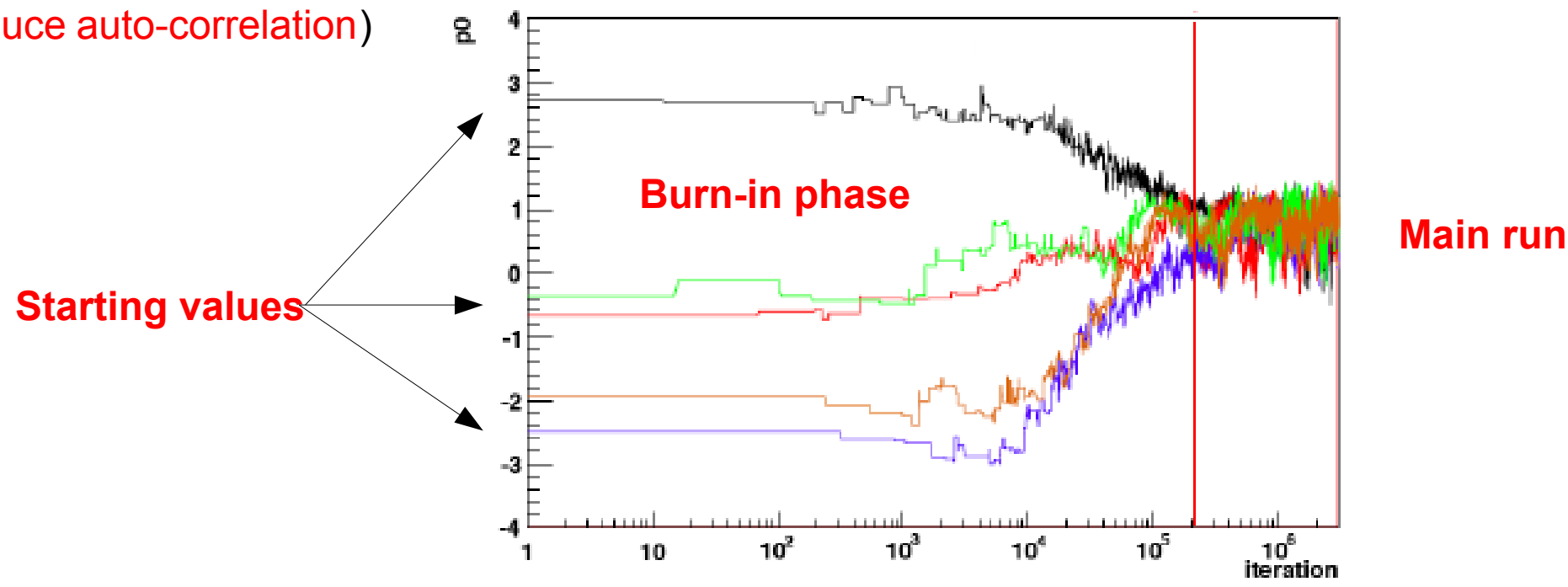
Step 3: Main run

Fix width of proposal function to that obtained from efficiency optimization
(always fixed during the main run)

Run for a specified number of iterations

Perform analysis-specific calculations
(fill marginalized histograms, uncertainty propagation, fill ROOT tree, etc.)

Store information of every n^{th} iteration
(reduce auto-correlation)

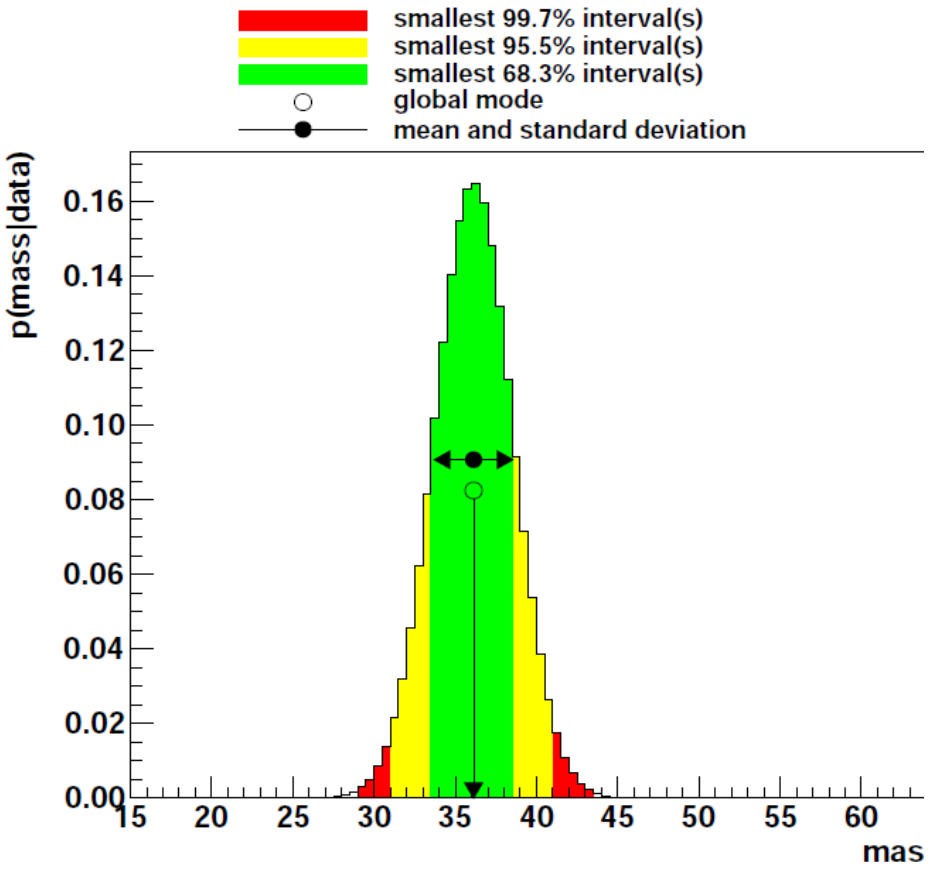


Full (correlated) information in
Markov Chain written as ROOT TTree

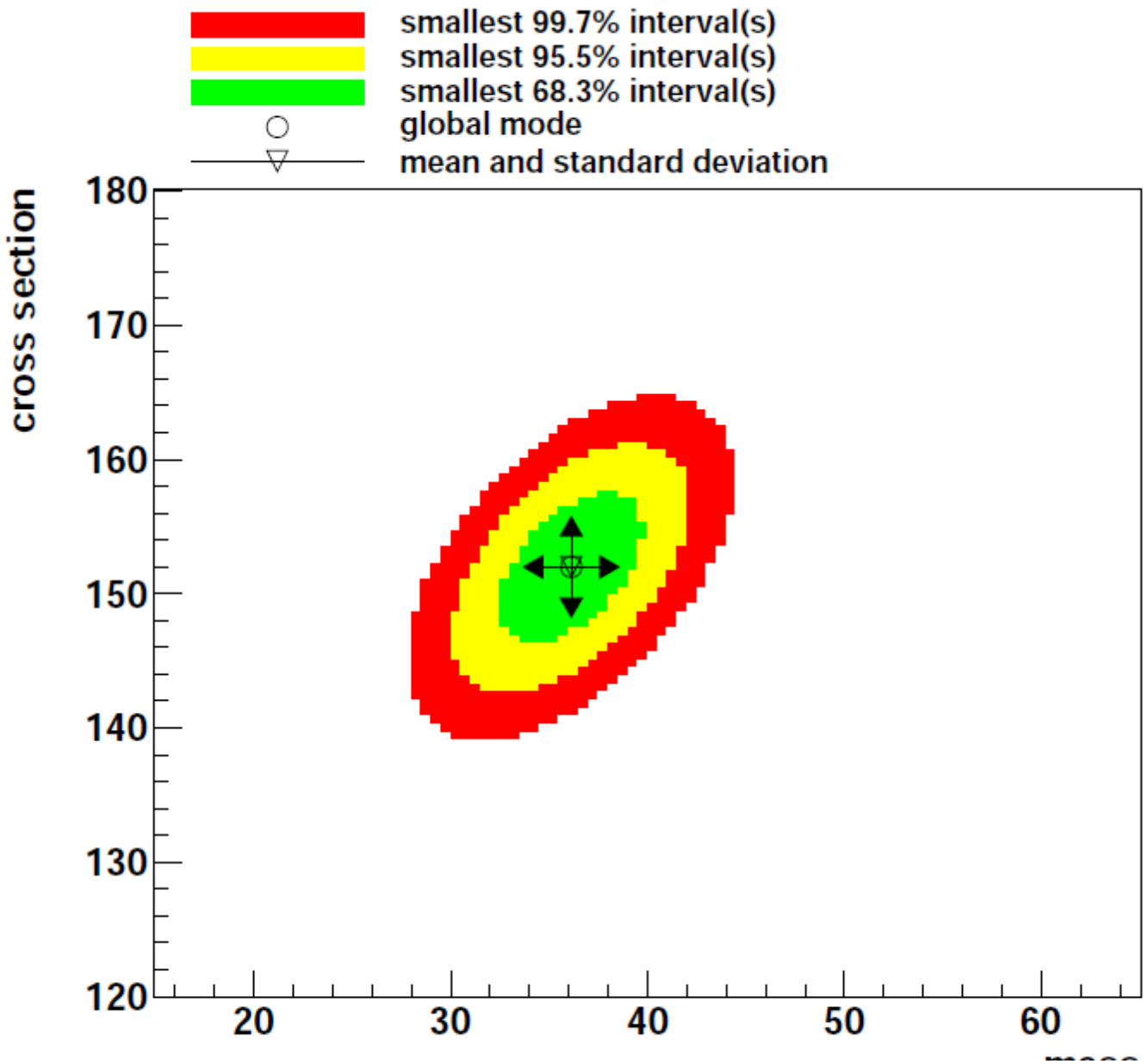
Default text output:

- Mean and standard deviation
- Median and central interval
- Mode and smallest intervals
- Important quantiles
- Global mode

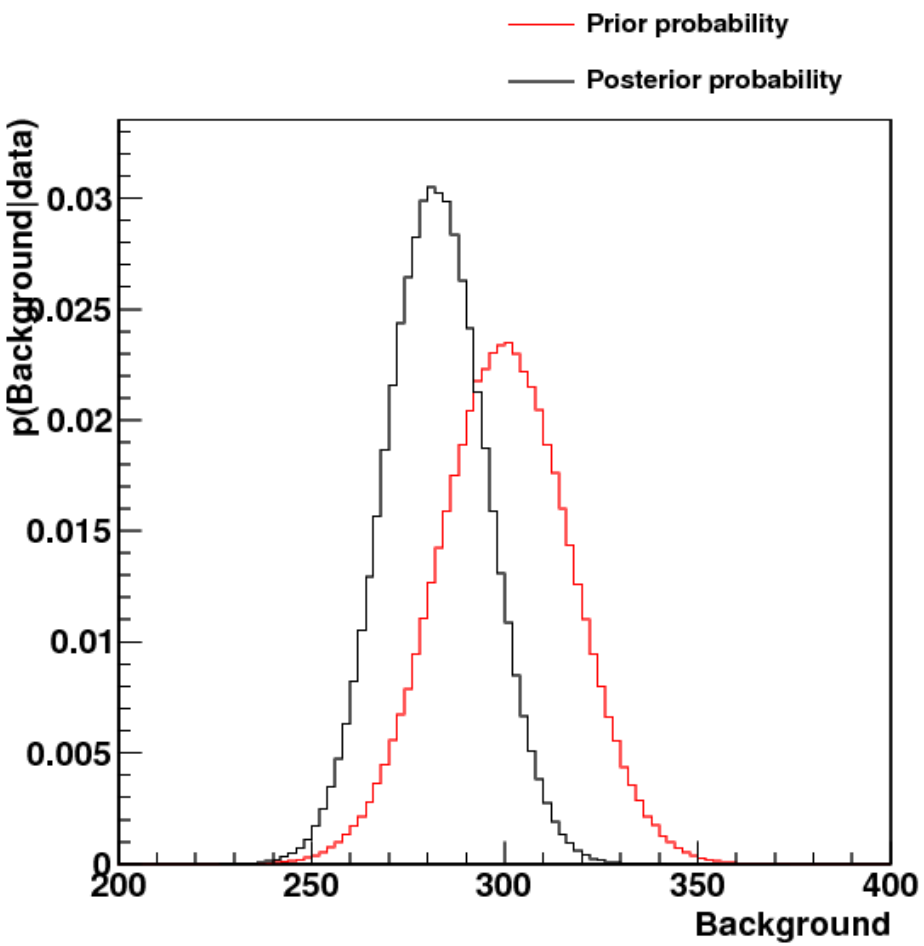
Marginalizations:
Projection of posterior in
one or two parameter dimensions

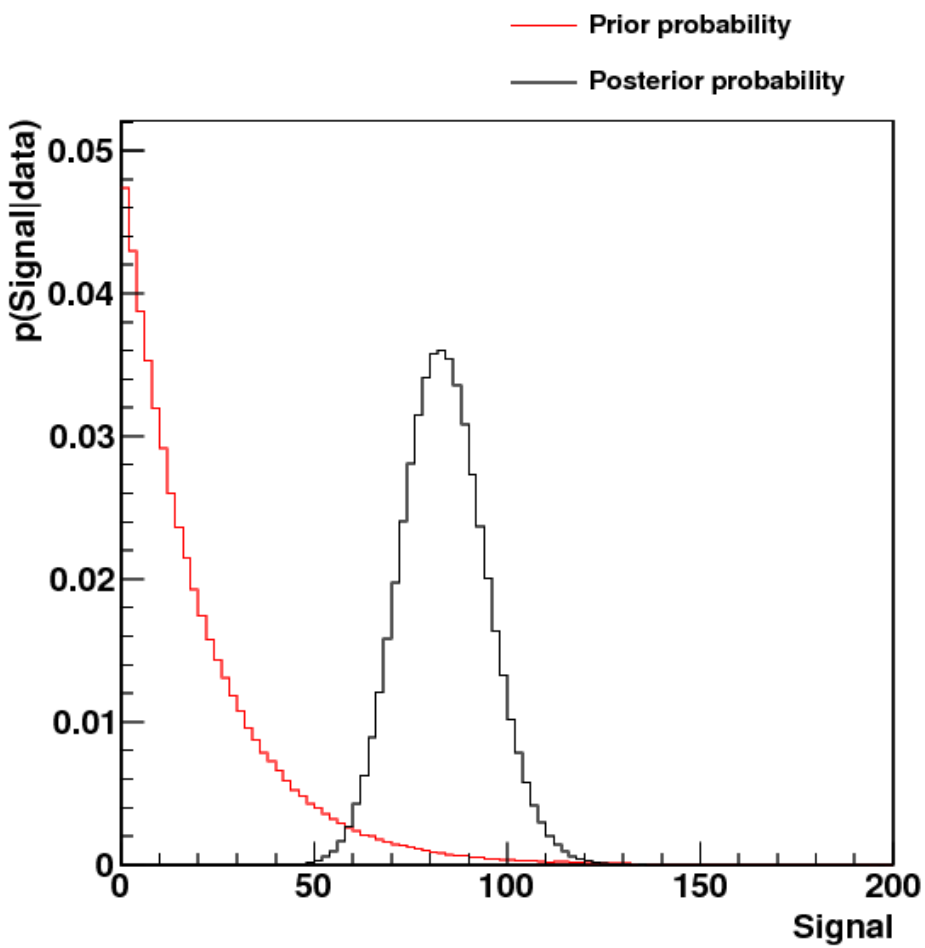


Output – 2D Marginalized Distribution

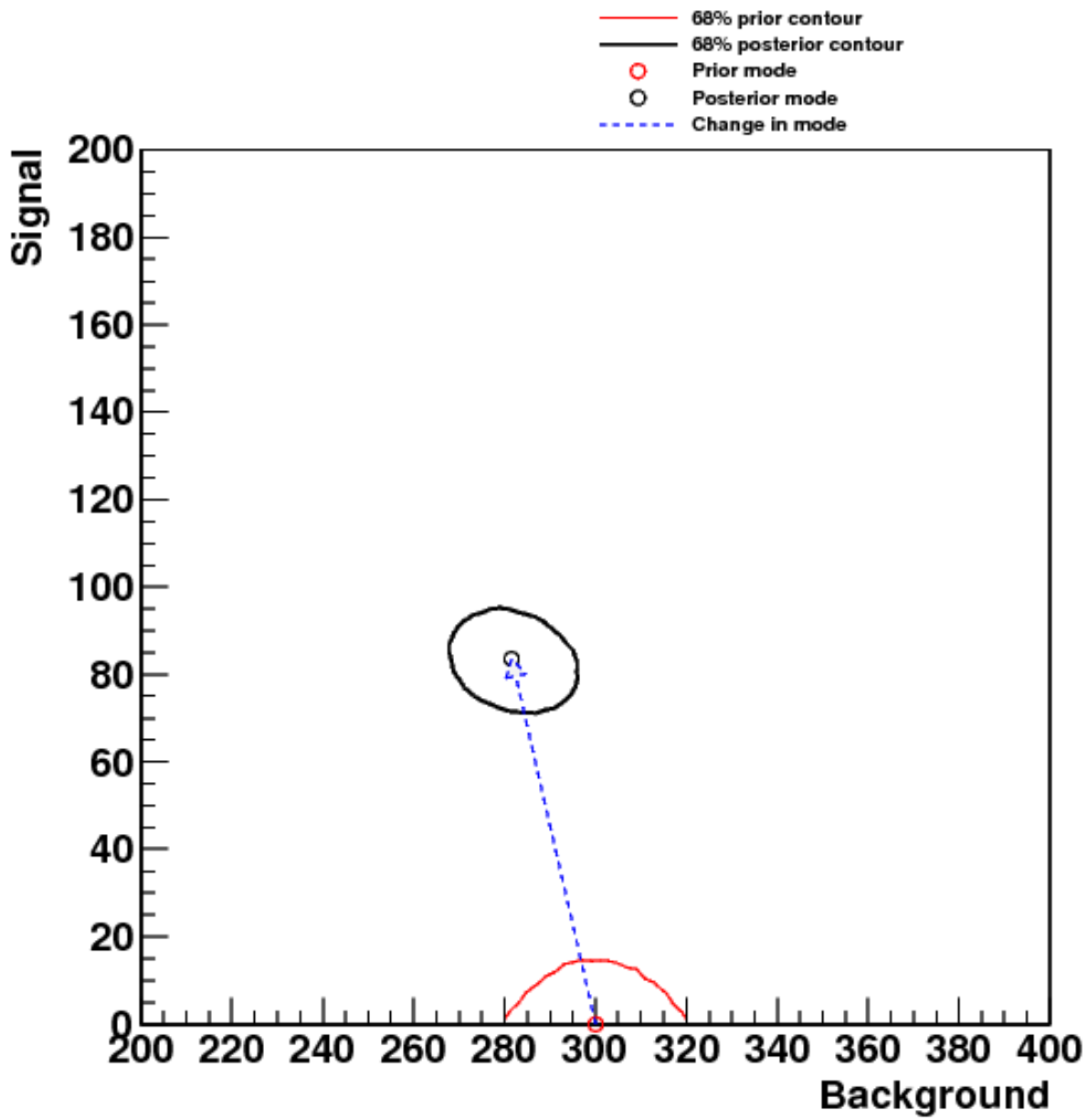


Output – Update of Knowledge

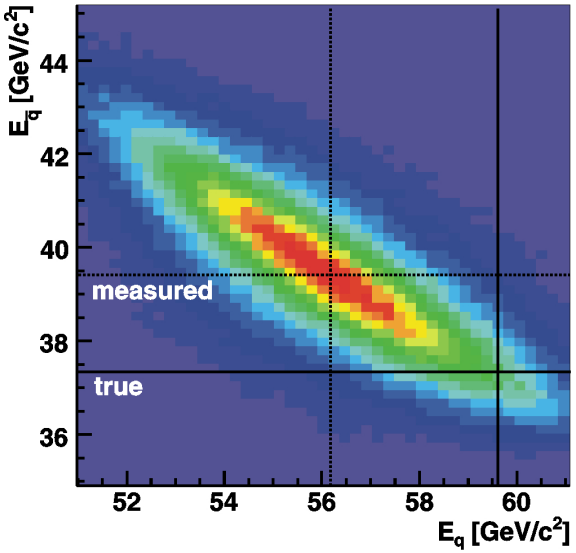


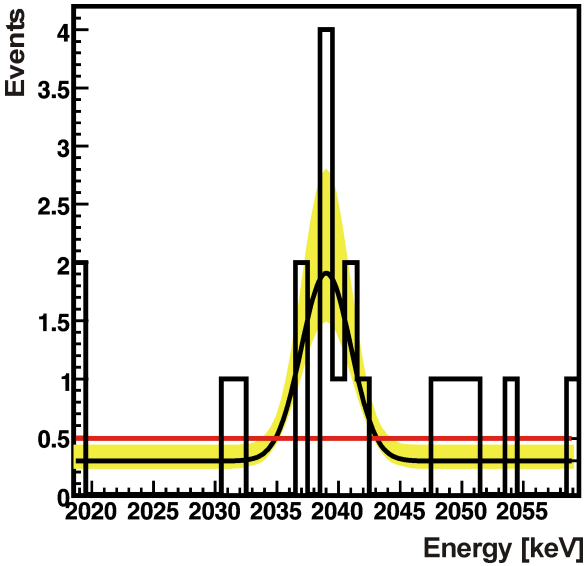


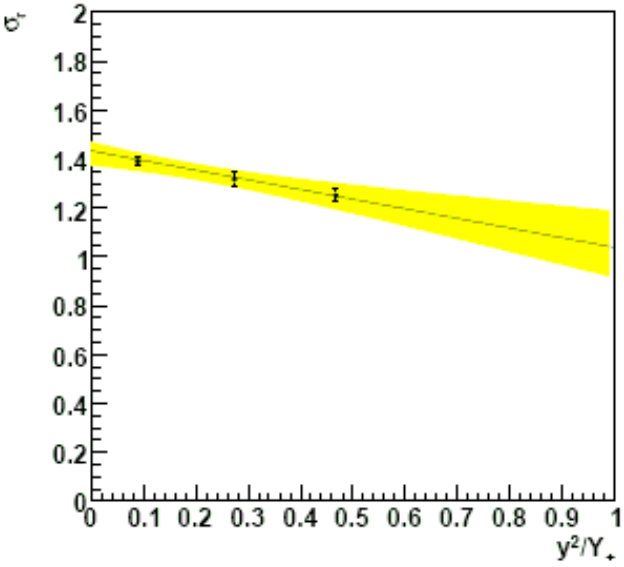
Output – Update of Knowledge



Example-Use Cases







Quentin Buat,*Search for extra dimensions in the diphoton final state with ATLAS* [arXiv:1201.4748]

ATLAS collaboration, *Search for excited leptons in proton-proton collisions at sqrt(s) = 7 TeV with the ATLAS detector* [arXiv:1201.3293]

I. Abt *et al.*, *Measurement of the temperature dependence of pulse lengths in an n-type germanium detector,* Eur. Phys. J. Appl. Phys.56:10104,2011 [arXiv:1112.5033]

ATLAS collaboration, *Search for Extra Dimensions using diphoton events in 7 TeV proton-proton collisions with the ATLAS detector* [arXiv:1112.2194]

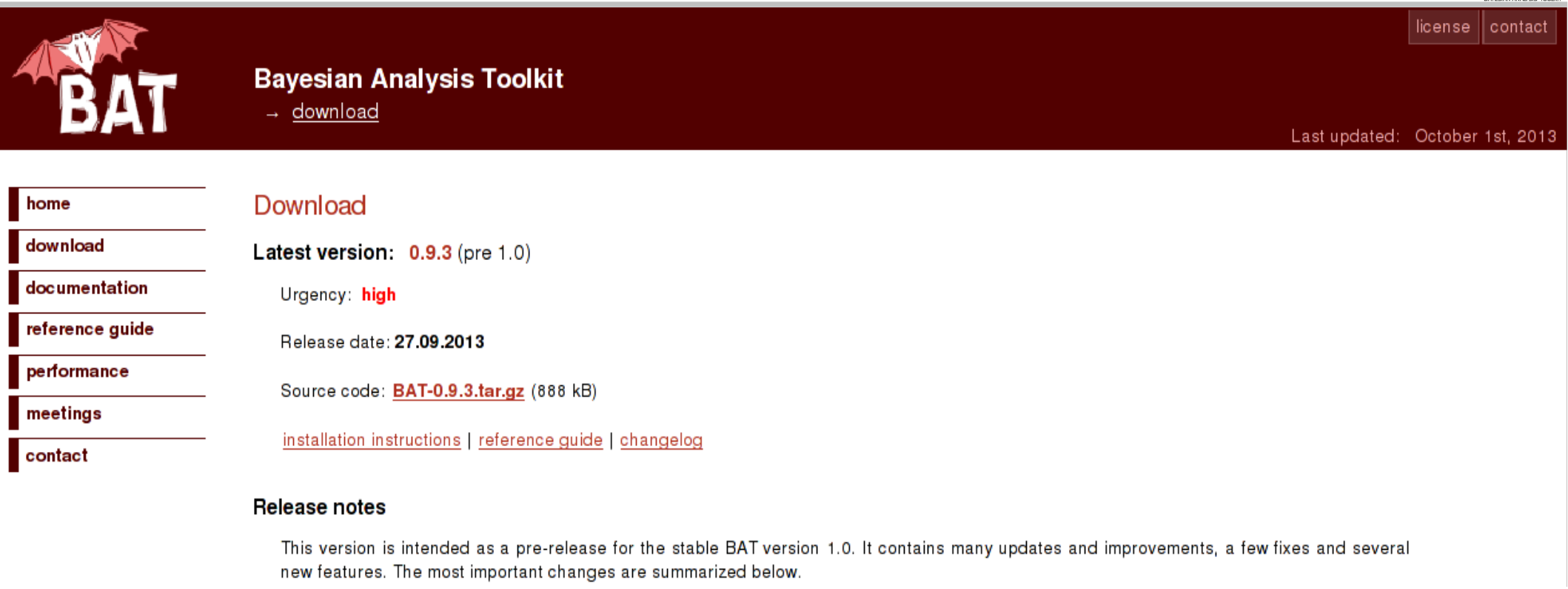
ATLAS collaboration, *A measurement of the ratio of the W and Z cross sections with exactly one associated jet in pp collisions at sqrt(s) = 7 TeV with ATLAS*, Phys.Lett.B708:221-240,2012 [arXiv:1108.4908]

ZEUS collaboration, *Search for single-top production in ep collisions at HERA*, Phys.Lett.B708:27-36,2012 [arXiv:1111.3901]

CMS collaboration, *Search for a W' boson decaying to a muon and a neutrino in pp collisions at sqrt(s) = 7 TeV*, Phys.Lett.B701:160-179,2011 [arXiv:1103.0030]

ZEUS collaboration, *Measurement of the Longitudinal Proton Structure Function at HERA,* Phys.Lett.B682:8-22,2009 [arXiv:0904.1092]

Contact



Web page: http://www.mppmu.mpg.de/bat/

EMail: bat@mppmu.mpg.de

Paper on BAT

A. Caldwell, D. Kollar, K. Kröninger, *BAT - The Bayesian Analysis Toolkit.* Comp. Phys. Comm. 180 (2009) 2197-2209 [arXiv:0808.2552]

Installing BAT

ROOT (required), http://root.cern.ch

Download the precompiled binary for your system,
before attempting compilation yourself!

After installation edit environment variables

bash, zsh (~/.bashrc, ~/.zshenv)

```
export ROOTSYS=/path/to/root
export PATH=$ROOTSYS/bin:$PATH
export CPATH=$ROOTSYS/include:$CPATH
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
```

csh, etc. – change export X=Y to setenv X Y for above lines

TEST:

open new terminal (or source env-variable script)

run root-config --version

BAT: https://github.com/bat/bat ("Download ZIP" button on right-hand side)
(Or: https://www.mppmu.mpg.de/bat/)

Unzip, enter directory, and run ./autogen.sh

```
./configure --with-rootsys=`root-config --prefix`
            [ --prefix=/desired/path/to/installation ]

make          (~ 1 minute)

make install
```

After installation edit environment variables (if --prefix=...)

bash, zsh (~/.bashrc, ~/.zshenv)

```
export BATINSTALLDIR=/path/to/bat
export CPATH=$BATINSTALLDIR/include:$CPATH
export LD_LIBRARY_PATH=$BATINSTALLDIR/lib:$LD_LIBRARY_PATH
```

csh, etc. export X=Y → setenv X Y

TEST: open new terminal (or source env-variable script)
run ls -R $BATINSTALLDIR, to see headers (.h) and libraries (.so)

A Pedagogical Example

Do you have …

BAT (and therefore also ROOT) installed?

Environment variables properly set (BATINSTALLDIR)?

Create a new project:

run BAT's Create-Project script (found in the “tools” directory):

```
CreateProject.sh [ProjectName]
```

(Throughout the tutorial, my project name will be “TutMod”)

Enter newly created directory (same as project name) and run

```
make
./runTutMod
```

If you have no errors, you are ready to go!

A counting experiment:

Search for a signal (S) in the presence of a background (B)

Our parameters:

B ≡ expectation for number of background events

S ≡ number of signal events

N ≡ observed number of events

Later

ε ≡ detection efficiency

S_{obs} ≡ S / ε

Exercise 1: Precisely Known Background

Exercise 2: Uncertain Background

Exercise 3: Update of Knowledge

Exercise 4: Signal Detection Efficiency

Exercise 5: Propagation of Uncertainty

Exercise 6: Choice of Priors (optional)

Consult the BAT reference guide throughout:

http://www.mppmu.mpg.de/bat/docs/refman/html-0.9.3/annotated.html

(NB – Your-Model : BCModel : BCIntegrate : BCEngineMCMC)

AIM: Given a measured number of events, $N = 10$,
and a precisely known background expectation $B = 10$,
learn about number of measured signal events $S = ??$

Steps:

1. In TutMod::DefineParameters(),

   A. Add parameters for *B* and *S* using
   ::AddParameter(const char * name, double min, double max)

   Q – What should the ranges be?

   B. Define Priors for *B* and *S.*
   Consult reference guide of BCModel for predefined prior choices

   Q – What is the prior for *B*? What prior should we use for *S*?

2. Define likelihood $L(N \mid B, S)$
   in TutMod::LogLikelihood(const std::vector<double> &parameters)
   according to Poisson distribution
   Use BCMath::LogPoisson(double measured, double expected)

   Q – What is the expected number of events?

3. Modify runTutMod.cxx for analysis

   ::SetMarginalizationMethod(BCIntegrate kMargMetropolis)

   ::MarginalizeAll()

   And output

   ::PrintAllMarginalized(const char * filename)

   ::PrintResults(const char * filename)

4. Compile (make) and run (./runTutMod)!

1. In TutMod.cxx

```
void TutMod::DefineParameters() {
    // Define Parameters
    AddParameter("B",0,20);
    AddParameter("S",0,20);
    // Set Priors
    SetPriorDelta("B",10);
    SetPriorConstant("S");
}
```

Comment out TutMod::LogAPrioriProbability(...) in TutMod.cxx AND TutMod.h

2. In TutMod.cxx

```
double TutMod::LogLikelihood(const std::vector<double> &parameters) {
    double N = 10;
    double B = parameters[0];
    double S = parameters[1];
    return BCMath::LogPoisson(N,B+S);
}
```

3. In runTutMod.cxx uncomment lines

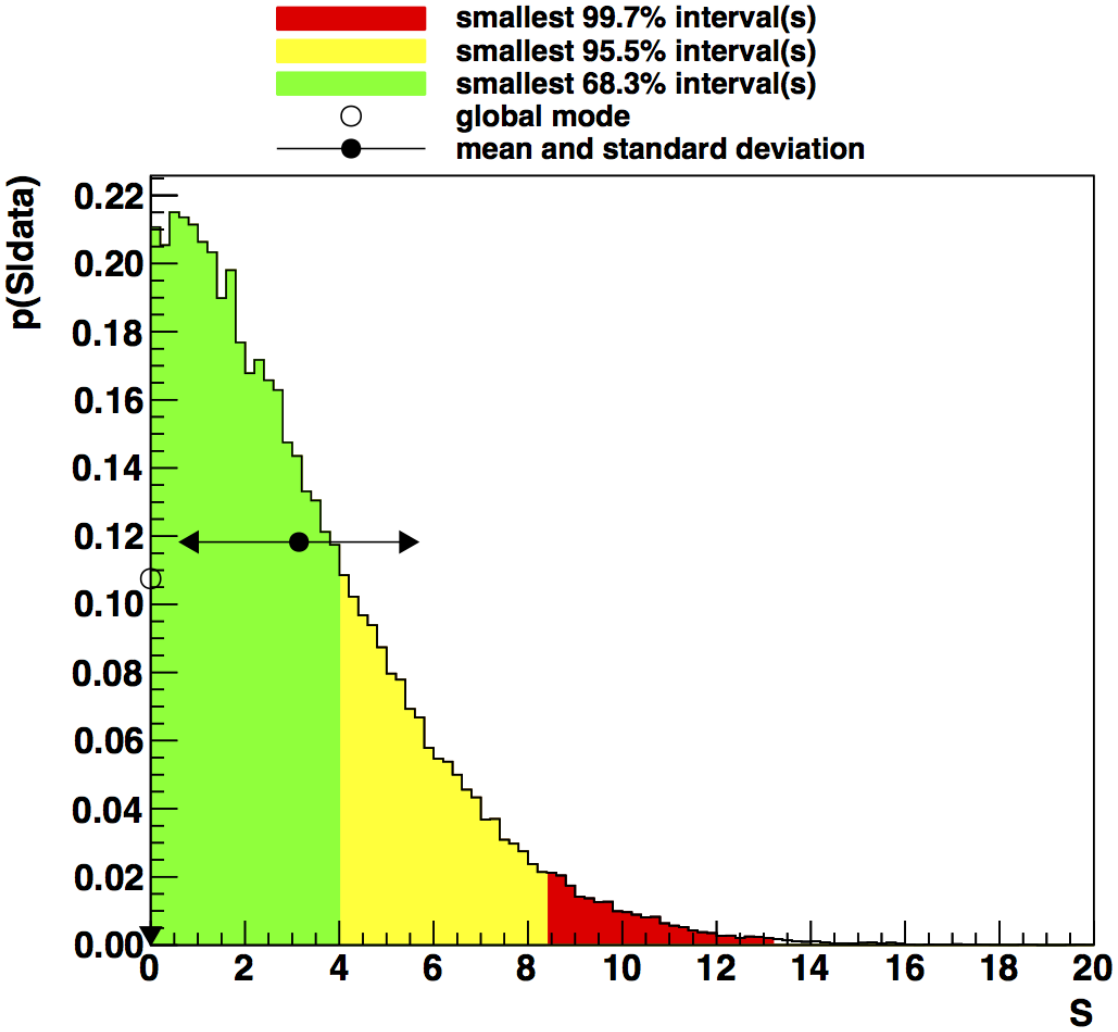
```
m->SetMarginalizationMethod(BCIntegrate::kMargMetropolis);
m->MarginalizeAll();
m->PrintAllMarginalized("TutMod_plots.pdf");
m->PrintResults("TutMod_results.txt");
```

Additional fun:

Rerun with different measured numbers of events.

Try N < B (for example, N=5).

Solution 1 – Precisely Known Background – Output



90% upper limit on signal: 6.7

95% upper limit on signal: 8.2

Exercise 2 – Uncertain Background

AIM: Given a measured number of events, $N = 10$,
and an uncertain background expectation $B = 10 \pm 3$,
learn about number of measured signal events $S = ??$

Steps:

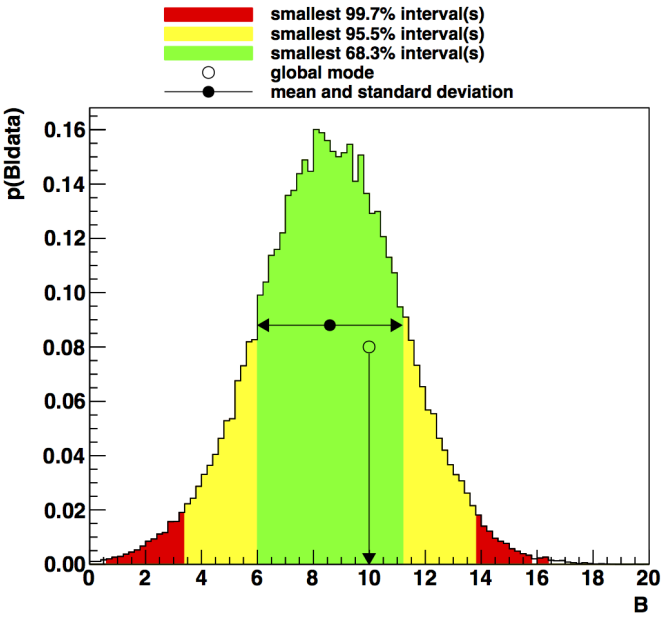
1. Modify background prior to incorporate uncertainty

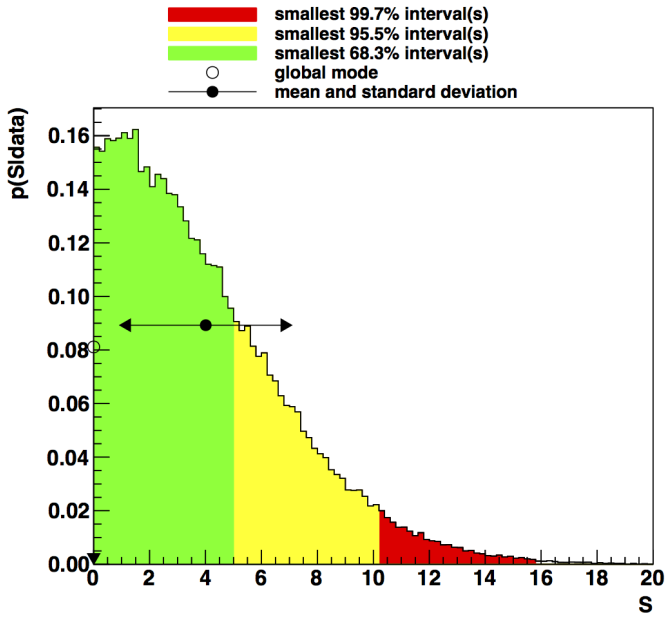
   Q – What is the prior for B?

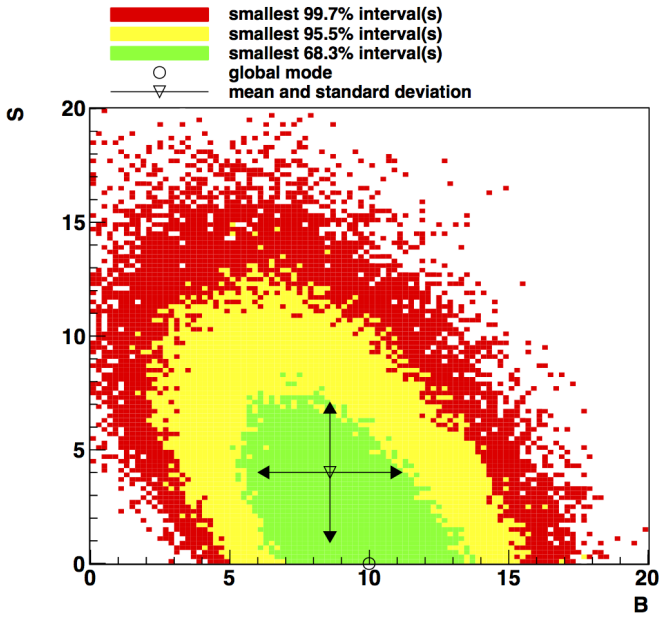
2. Compile & run.

Exercise 2 – Uncertain Background – Solution

1. SetPriorGauss(“B”, 10, 3);







90% upper limit on signal: 8.3

95% upper limit on signal: 10.3

Exercise 3 – Update of Knowledge

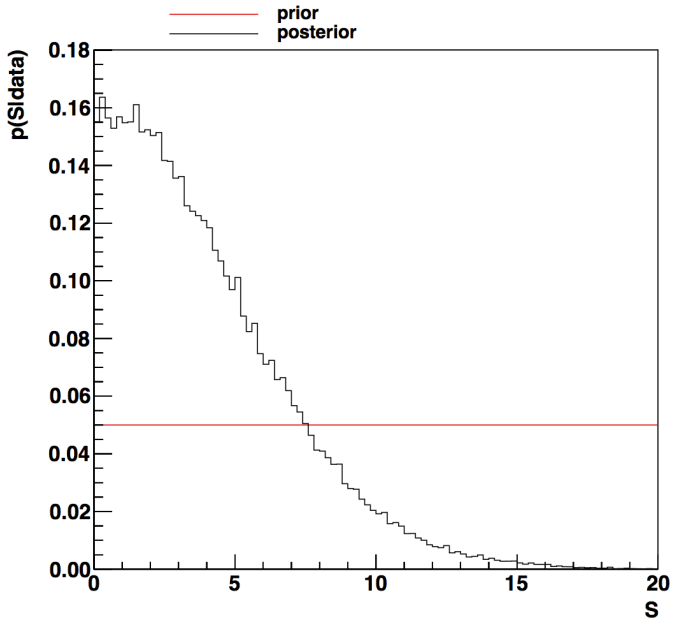
AIM: Check update of knowledge

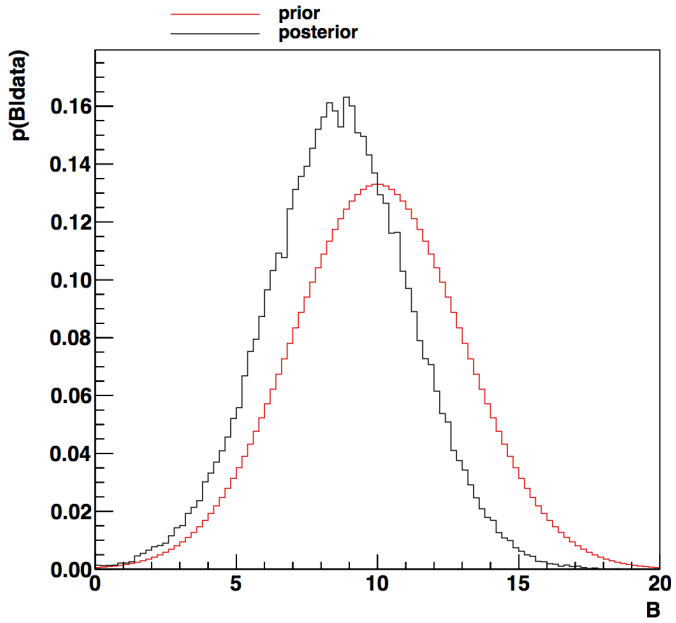
Steps:

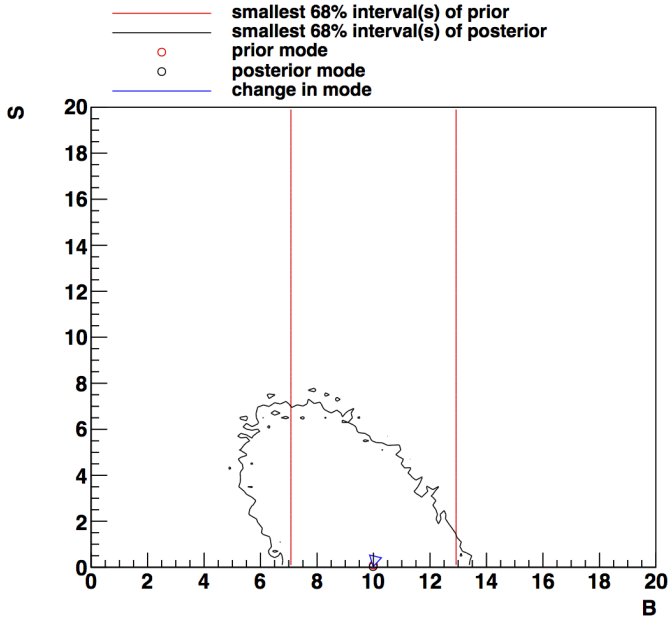
1. Use BCSummaryTool to print knowledge update.

1. In runTutMod.cxx uncomment line

```
summary->PrintKnowledgeUpdatePlots("TutMod_update.pdf");
```







Exercise 4 – Signal Detection Efficiency

AIM: Add efficiency of detection into our model, $\varepsilon = (10 \pm 2)\ \%$

Steps:

1. Add parameter for efficiency and set its prior. Change range for *S*.

   Q – What is the range for the efficiency? What is its prior?

   Q – What should the new range for *S* be?

2. Change likelihood calculation.

   Q – How must you change the expected number of events?

Q – What is the 95% upper limit on the true *S*?

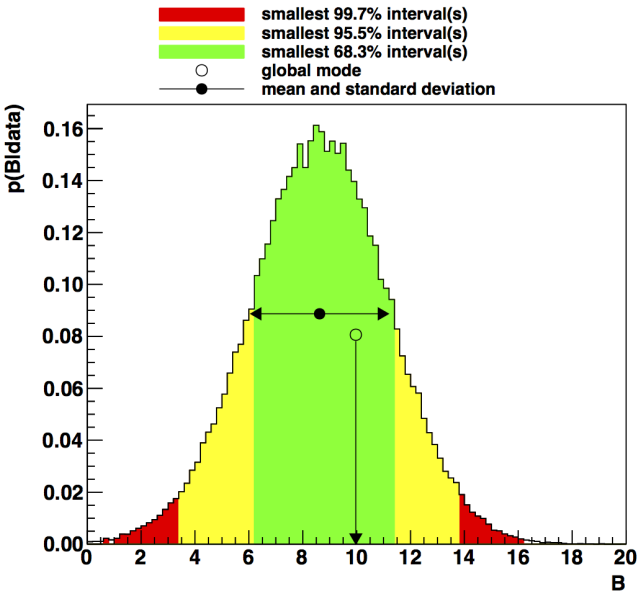
Exercise 4 – Signal Detection Efficiency – Solution

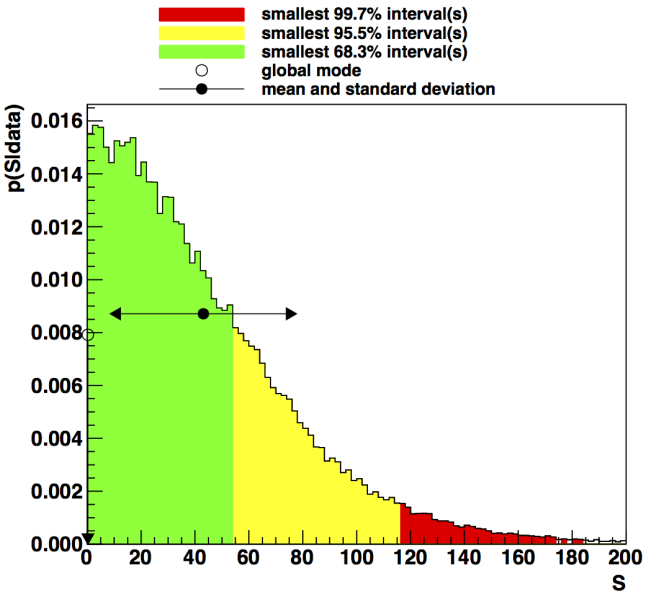
1. In DefineParameters() ...

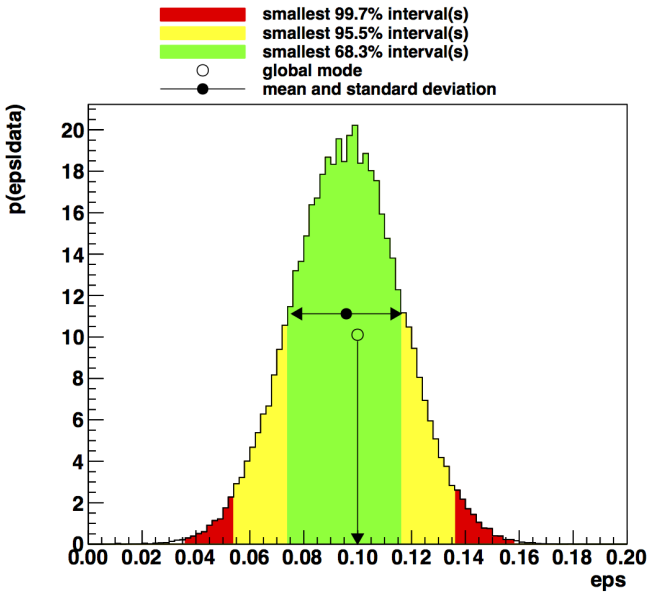
```
AddParameter(“S”, 0, 200);
AddParameter("eps", 0, 0.20);
SetPriorGauss(“eps”, 0.10, 0.02);
```

2. In LogLikelihood(std::vector<double> &parameters) …

```
double eps = parameters[2];
return BCMath::LogPoisson(N,B+S*eps);
```







95% upper limit on true *S* = 115

AIM: Calculate the number of observed signal events and store its distribution. $S_{obs} = \varepsilon S$

Steps:

1. Add calculation of observed number of events.

   Add ROOT TH1D for storing histogram S_{obs}.

2. Use BCH1D to draw histogram.

   Q – What is the 95% upper limit on the observed S?

Exercise 5 – Propagation of Uncertainty – Solution (hard stuff)

1. In TutMod.h add... At top: #include <TH1D.h>

   And inside class:

```
TH1D * MyHistogram;
void MCMCUserIterationInterface();
```

In TutMod.cxx add.... In constructor:

```
MyHistogram = new TH1D("S","";S_{obs};P(S_{obs}|data)",100, 0, 20);
```

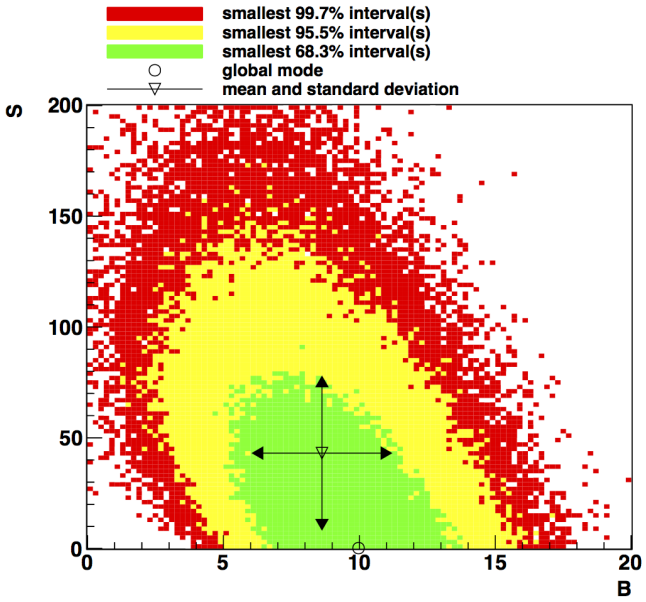
and add function:

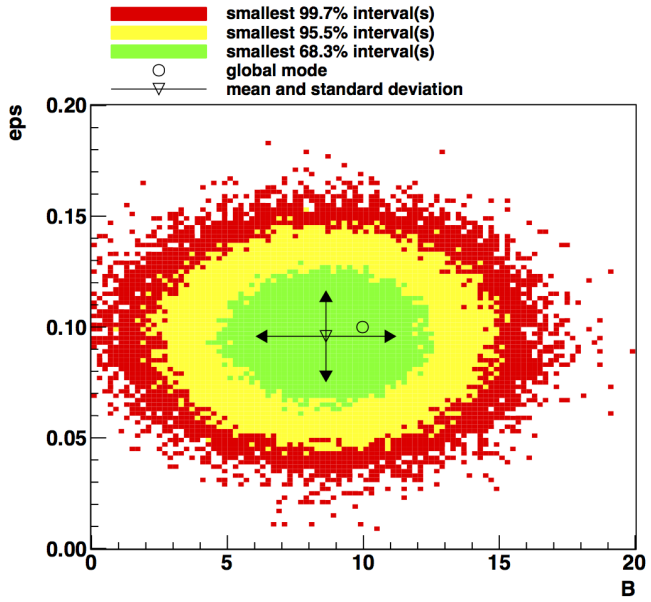
```
void TutMod::MCMCUserIterationInterface() {
   for (int i=0; i<MCMCGetNChains(); ++i) {
         double S      = fMCMCx.at(i*GetNParameters() + 1);
         double eps    = fMCMCx.at(i*GetNParameters() + 2);
         MyHistogram->Fill(S*eps);
   }
}
```

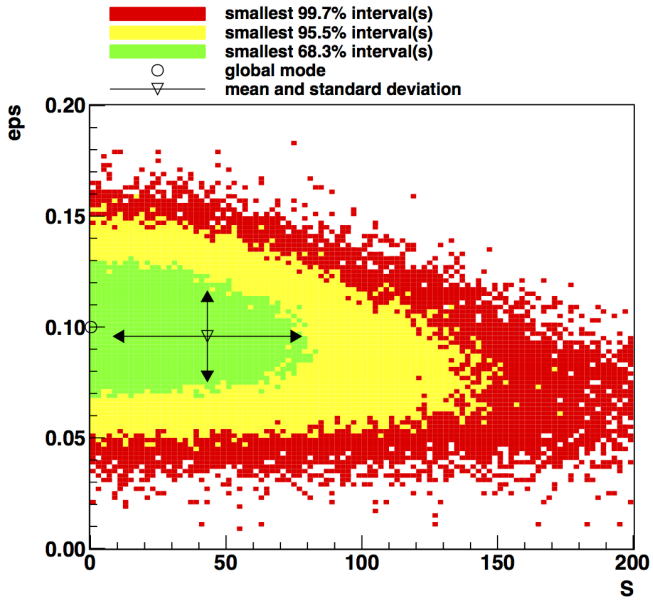
2. In runTutMod.cxx add (after printing results)...

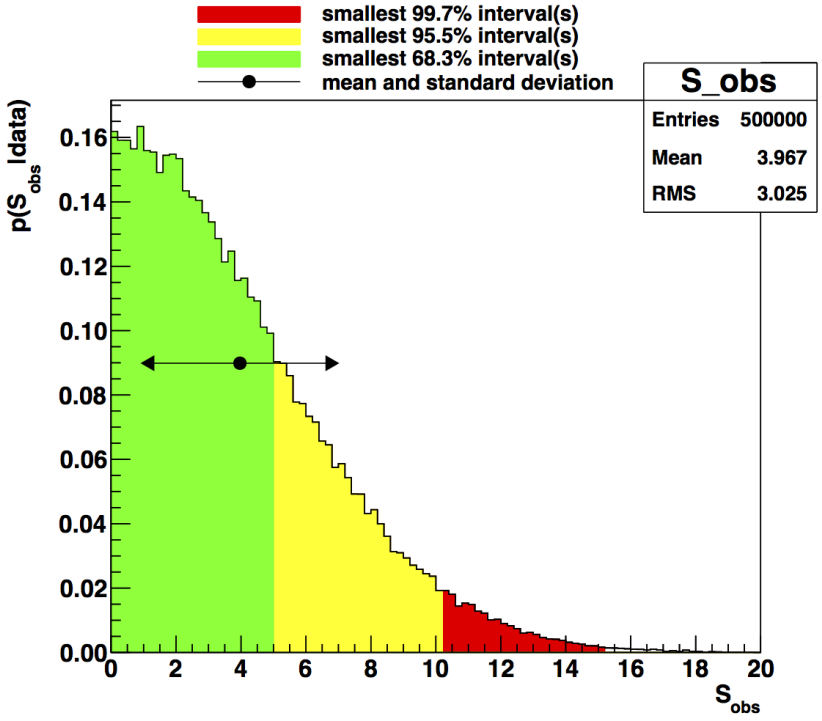
```
BCH1D * MyS = new BCH1D(m->MyHistogram);
MyS -> Print("TutMod_ObsS_plot.pdf");
```

Exercise 5 – Propagation of Uncertainty – Solution (cont'd)









Repeat your analysis with different priors

e.g. an exponential one,

a Gaussian one,

or a Jeffreys prior

How do the limits on the true and observed signal counts change?

Uncomment TutMod::LogAPrioriProbability(...) and go wild!

Further practice!

Additional (more complicated) tutorials can be found at

https://www.mppmu.mpg.de/bat/?page=tutorials

Installing BAT with Cuba

CUBA (optional), http://www.feynarts.de/cuba, latest version (3.2)

Download source file, extract, enter directory

```
./configure CFLAGS='-fPIC -O3 -fomit-frame-pointer -ffast-math -Wall'
            [ --prefix=/desired/path/to/installation ]

make lib        (~10 seconds)

make install
```

After installation edit environment variables:

bash, zsh (~/.bashrc, ~/.zshenv)

```
export CUBADIR=/path/to/cuba
export CPATH=$CUBADIR/include:$CPATH
export LD_LIBRARY_PATH=$CUBADIR/lib:$LD_LIBRARY_PATH
```

csh, etc. export X=Y → setenv X Y

TEST: open new terminal (or source env-variable script)
run ls -R $CUBADIR, and you should see cuba.h and libcuba.a

BAT, http://mppmu.mpg.de/bat, latest version (0.9.3)

Download source, extract, enter directory

```
./configure --with-rootsys=`root-config --prefix`
            [ --with-cuba --with-cuba-include-dir=$CUBADIR/include
            --with-cuba-lib-dir=$CUBADIR/lib ]
            [ --prefix=/desired/path/to/installation ]

make        (~ 1 minute)

make install
```

After installation edit environment variables (if --prefix=...)

bash, zsh (~/.bashrc, ~/.zshenv)

```
export BATINSTALLDIR=/path/to/bat
export CPATH=$BATINSTALLDIR/include:$CPATH
export LD_LIBRARY_PATH=$BATINSTALLDIR/lib:$LD_LIBRARY_PATH
```

csh, etc. export X=Y → setenv X Y

TEST: open new terminal (or source env-variable script)
run ls -R $BATINSTALLDIR, to see headers (.h) and libraries (.so)