Technische Universität München

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

IN2P3 School of Statistics, Autrans, June 2, 2016

Daniel Greenwald, Technische Universität München

The BAT men: Frederik Beaujean, Allen Caldwell, Daniel Greenwald, Stefan Kluth, Kevin Kröninger, Oliver Schulz

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.)

Requirements and solutions

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 $p_0(\vec{\lambda})$
- def. & calc. observables $f(\vec{\lambda})$

IN YOUR CODE

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

Usage of MCMC in Bayesian inference

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

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

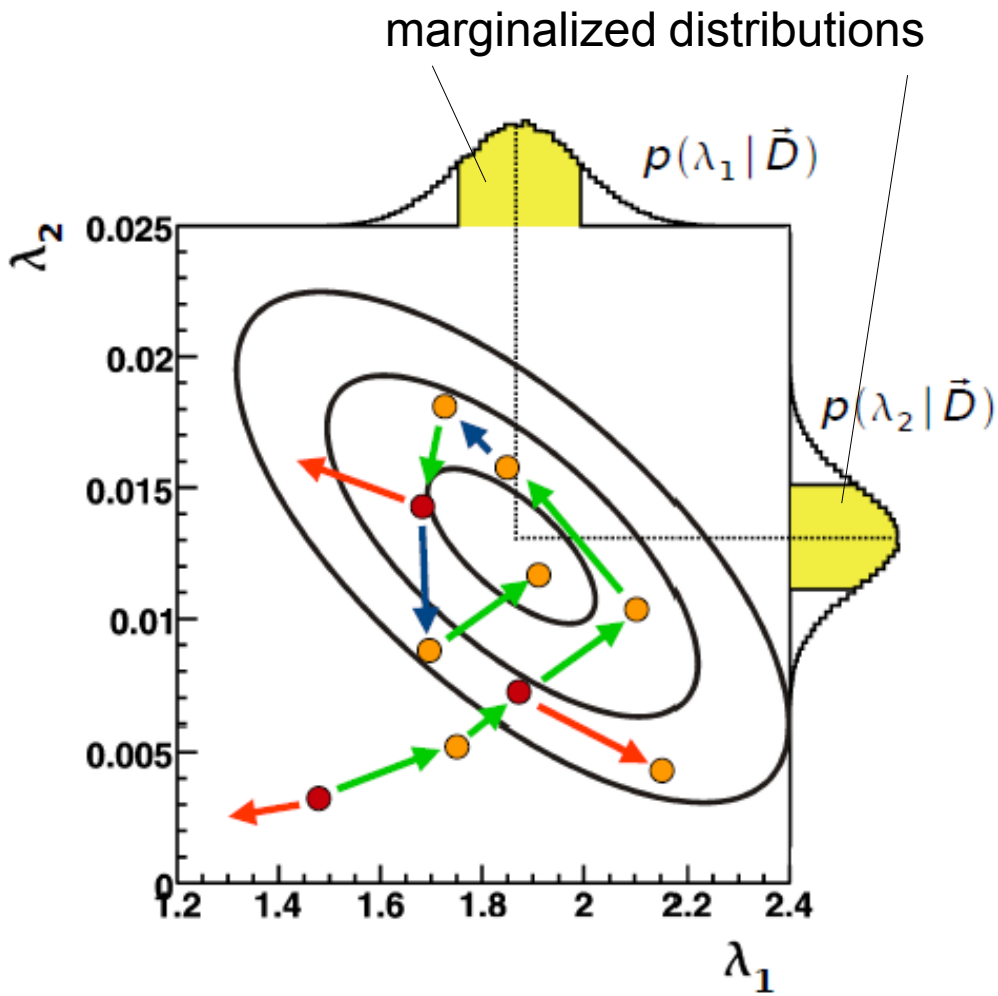
Marginalization of posterior:

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

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

Distributed random according to prior (default)

or center of each dimension

or user-defined

Step 2: Burn-in phase (Prerun)

Use multiple chains (by default 4)

Run until convergence is reached and chains are efficient

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

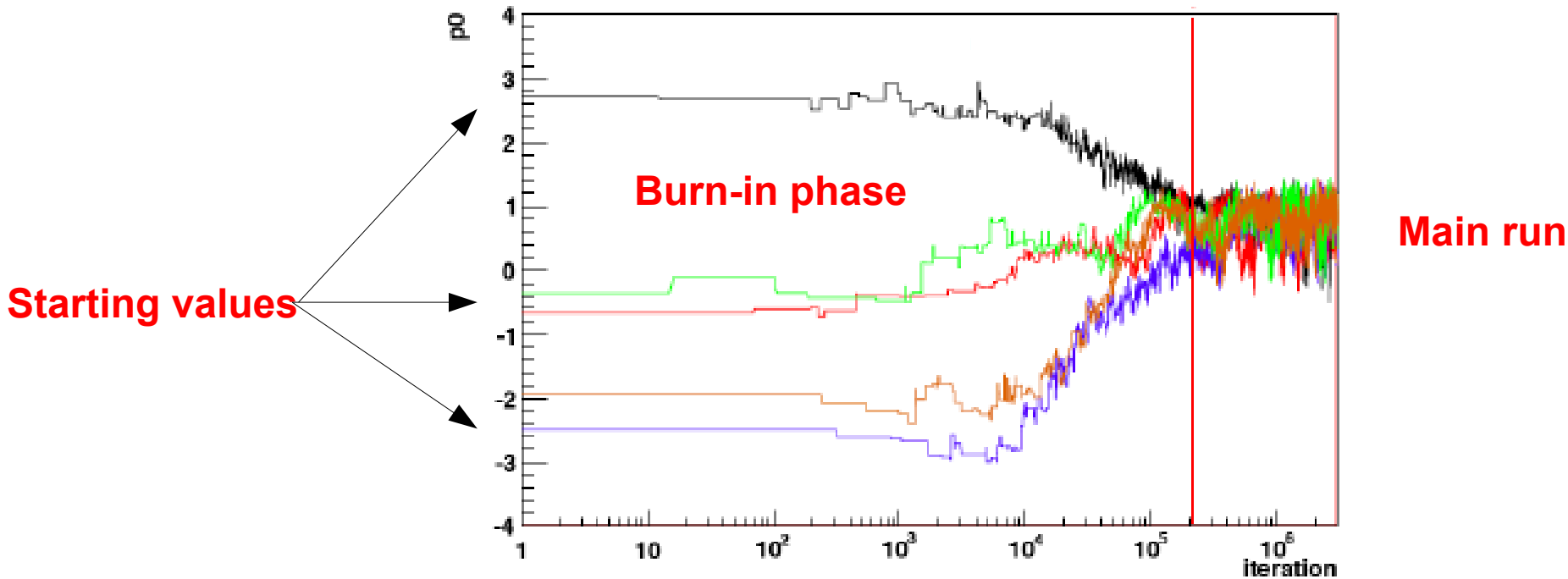
Adjust covariance matrix for multi-dimensional proposal function

Step 3: Main run

Fix scale and covariance of proposal function to that obtained from prerun
(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.)



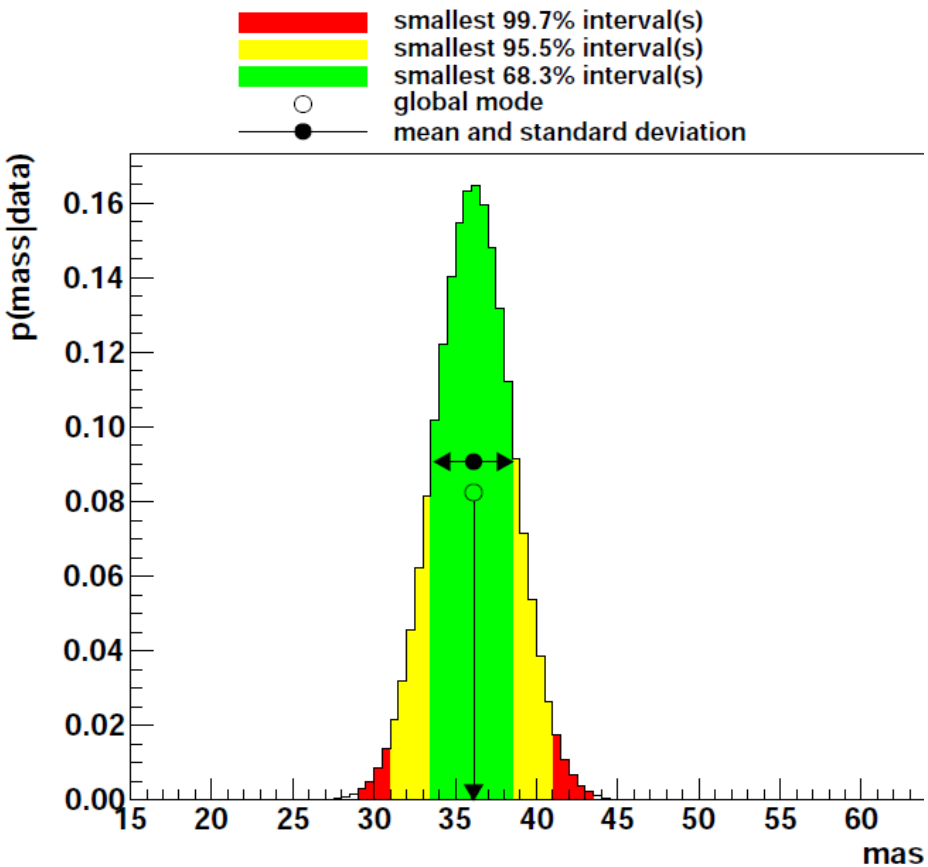
Output

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

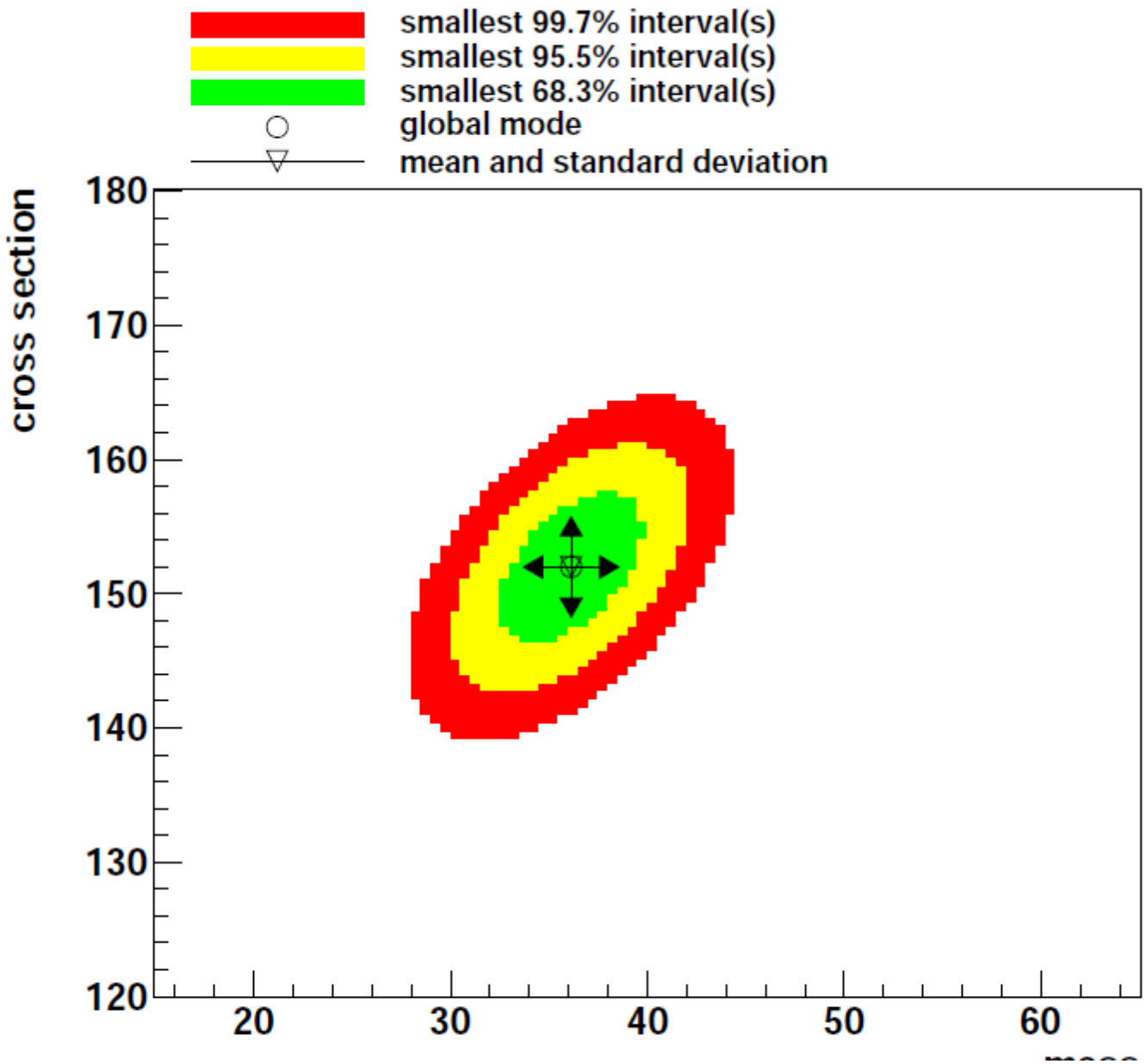
Default text output:

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

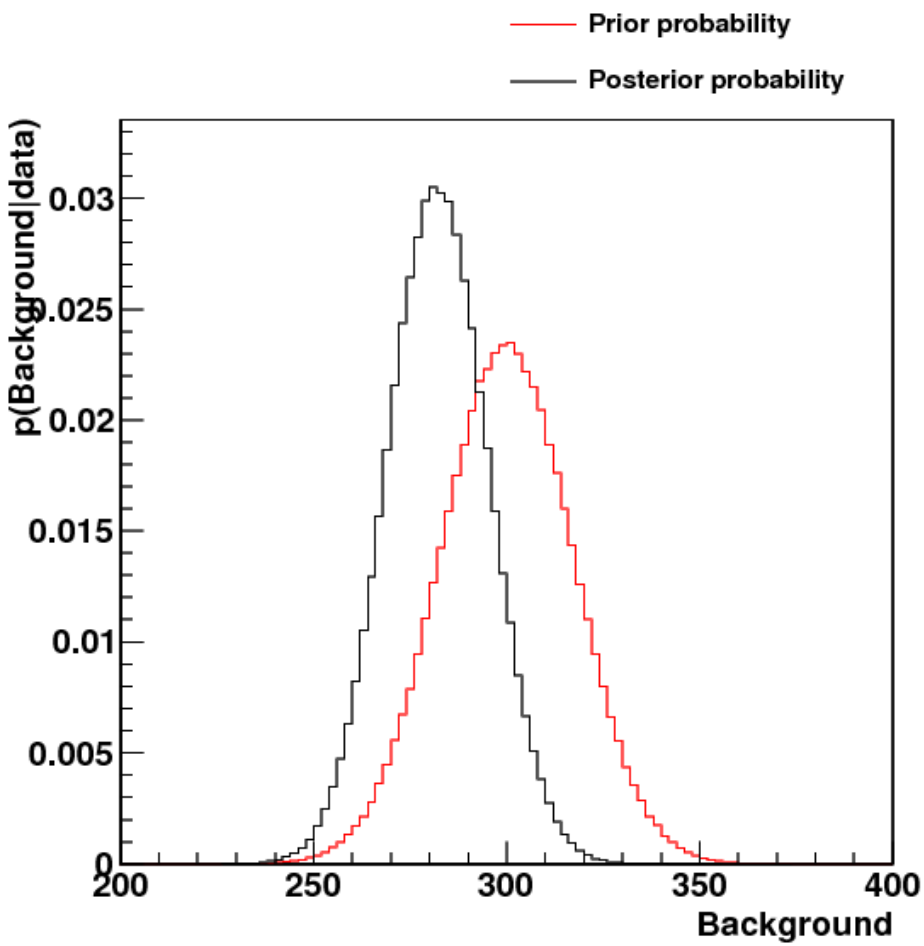
Marginalizations:
Projection of posterior in
one or two parameter dimensions

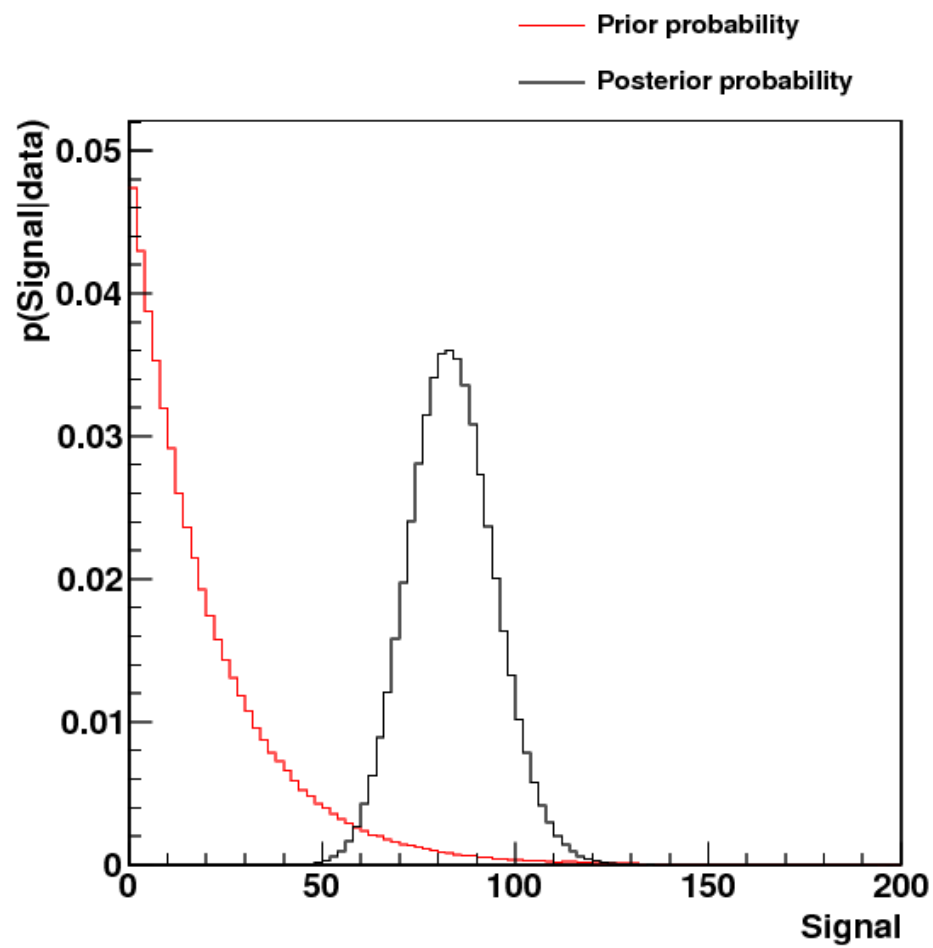


Output – 2D Marginalized Distribution

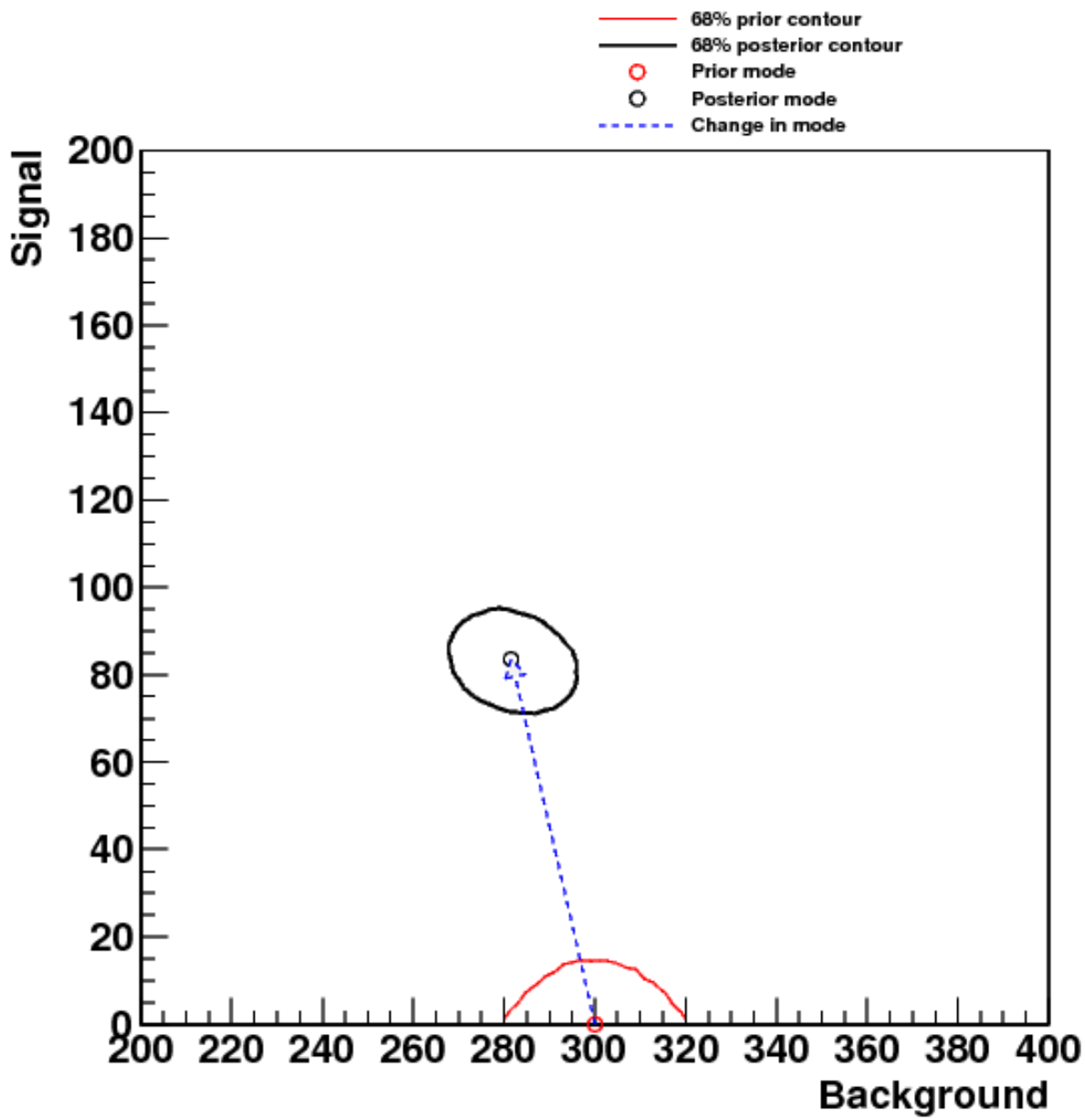


Output – Update of Knowledge

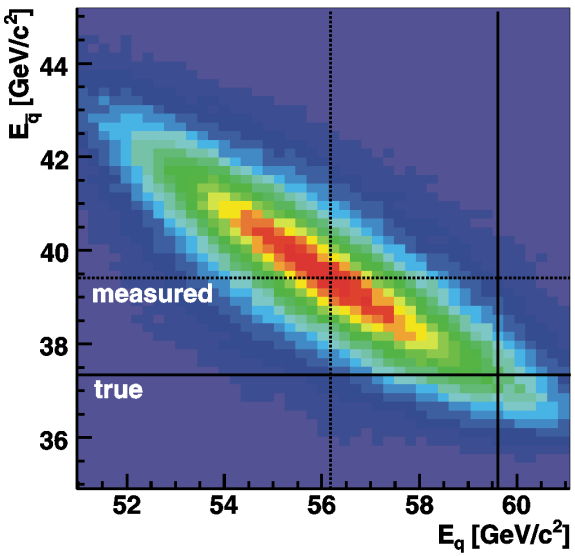


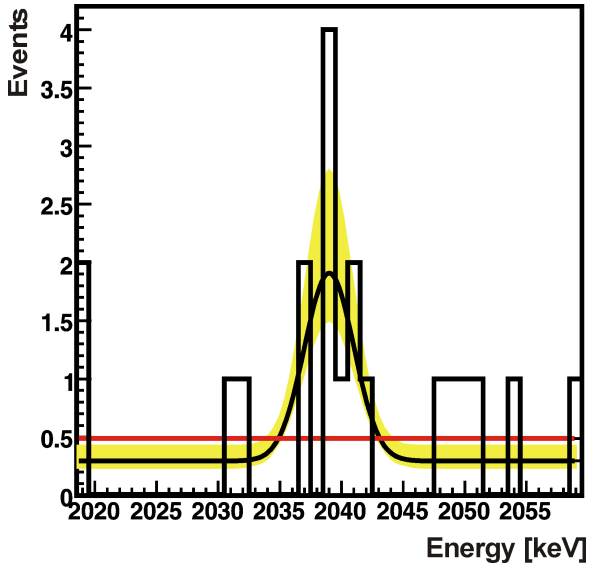


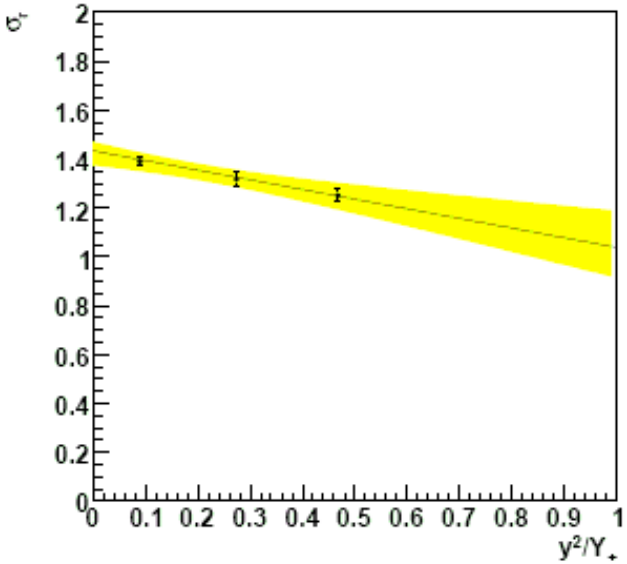
Output – Update of Knowledge



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]

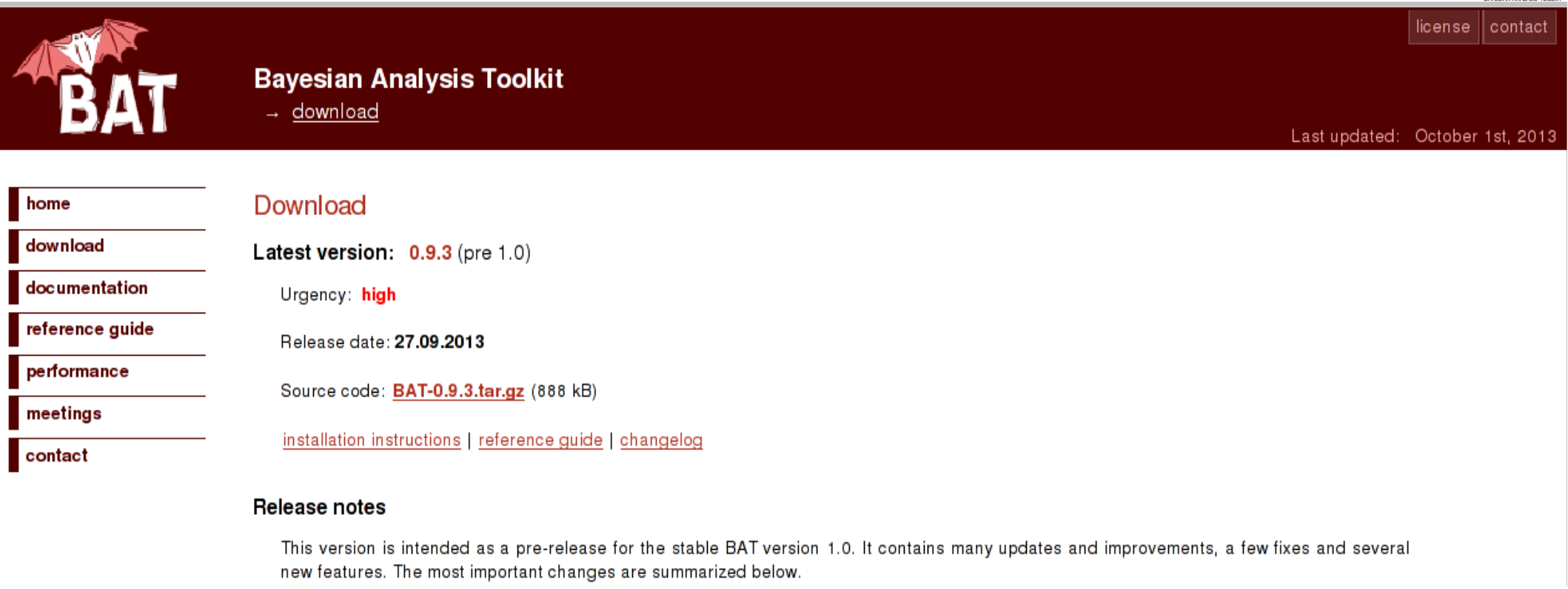
ATLAS collaboration, *A measurement of the ratio of the W and Z cross sections with exactly one associated jet in pp collisions at sqrt(s) = 7 TeV with ATLAS*, Phys.Lett.B708:221-240,2012 [arXiv:1108.4908]

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]

Using the BAT USB stick

Using the BAT USB stick

More information found in README's on stick

Linux users: (easiest method)

1. Copy `BAT-VM/bat-vm.proot-fs.tar.gz` to your computer and unzip/unpack it to `/some/place/`

2. In the terminal, from a directory other than `/some/place/bat-vm-proot`, run: `bash /some/place/bat-vm-proot/proot-shell`

3. Continue working now in that terminal.

All others:

1. Run your favorite virtual machine from those available in `BAT-VM` and import the relevant file.

1a. If you don't have a virtual machine handler installed:
Install VirtualBox from `Software/VirtualBox`.

2a. Run VirtualBox, in the GUI: “File > Import Appliance” and
import `BAT-VM/bat-vm.ova`
Login with username “vagrant”, password “vagrant”

A Pedagogical Example

Are you ready?

Create a new project:

run BAT's create-project script:

```
bat-project [ProjectName] [ModelName]
```

Throughout the tutorial, my project name will be Tuto

And my model will by TutMod

Enter newly created directory (same as project name)

```
cd Tuto
```

and run

```
make
./runTuto
```

If you have no compile errors or c++ errors, you are ready to go!
(BAT will issue a runtime error saying your model is empty. This is OK!)

A counting experiment:

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

Our parameters:

$B \equiv$ expectation for number of background events

$S \equiv$ number of signal events

$N \equiv$ observed number of events

Later

$\varepsilon \equiv$ detection efficiency

$S_{obs} \equiv \varepsilon * S$

Outline of Exercise

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: Multiple runs

Exercise 7: Choice of Priors (optional)

Be careful: if you copy and paste from this document, the quotation marks around strings may not be the same ones as C++ wants!

Exercise 1 – Precisely Known Background

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(string name, double min, double max, string latex_name, string units)

   Q – What should the ranges be?

   B. Define Priors for *B* and *S.*

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

   GetParameter(string name).Fix(double value);

   GetParameter(string name).SetPriorConstant();

   GetParameter(string name).SetPrior(BCPrior* prior);

   BCGaussianPrior::BCGaussianPrior(double mean, double sigma);

Exercise 1 – Precisely Known Background (cont'd)

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 to set model name:

   TutMod m(string name);

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

5. Try outputting ROOT TTree:

   ::WriteMarkovChain(string filename, string options);

   Before running MarginalizeAll(...). Options = "RECREATE"

Exercise 1 – Precisely Known Background – Solution

1. In TutMod.cxx

```
TutMod::TutMod()
   : BCModel()
{
      // Define Parameters
      AddParameter("B", 0, 20, "#nu_{B}", "[events]");
      AddParameter("S", 0, 20, "#nu_{S}", "[events]");
      // Set Priors
      GetParameter("B").Fix(10);
      GetParameter("S").SetPriorConstant();

}
```

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);
}
```

Exercise 1 – Precisely Known Background – Solution (cont'd)

3. In runTuto.cxx name the model:

```
TutMod m("myTutoMod");
```

5. Turn on ROOT output:

```
m.WriteMarkovChain(m.GetSafeName() + "_mcmc.root", "RECREATE");
```

Recompile: `make` Rerun: `./runTuto`

Open ROOT file: `root [file]`

Play with data:

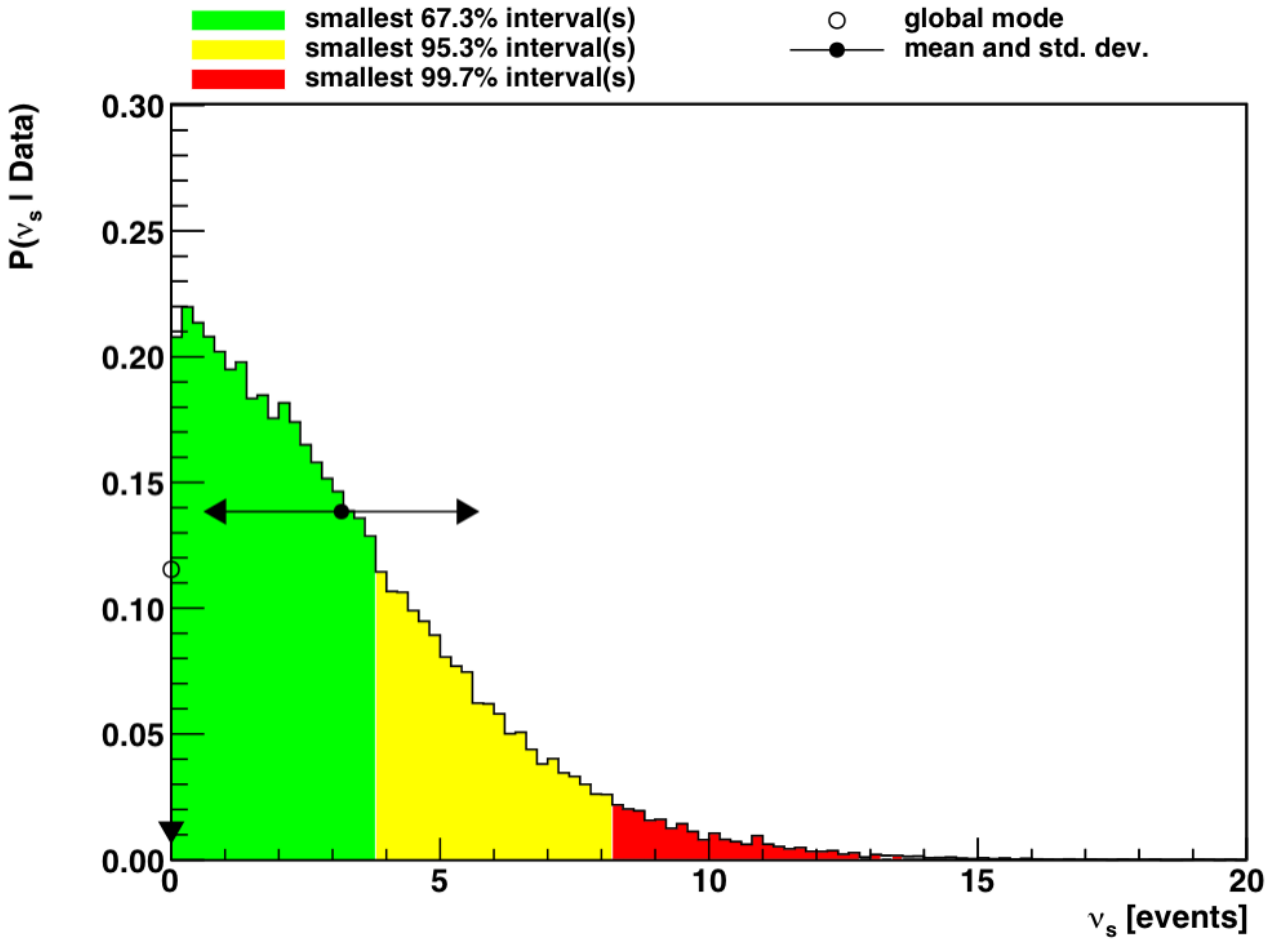
```
myTutoMod_mcmc->Draw("B:S", "Phase>0", "colz")
```

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: 8.1

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

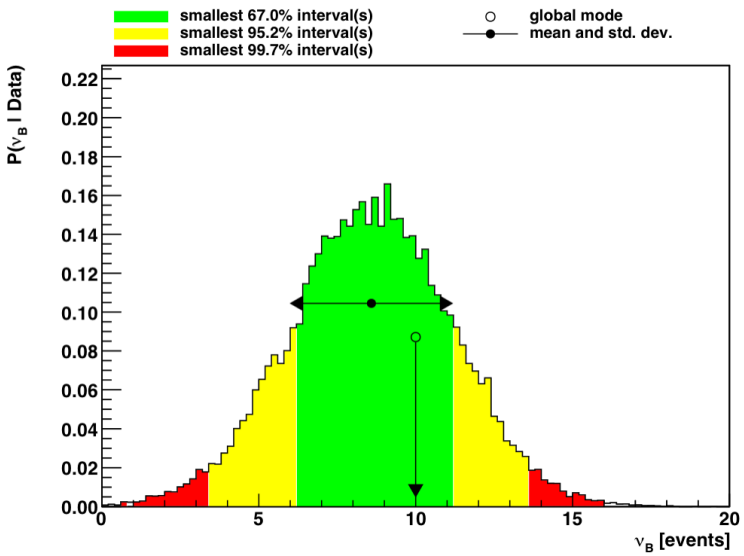
   Q – What is the prior for B?

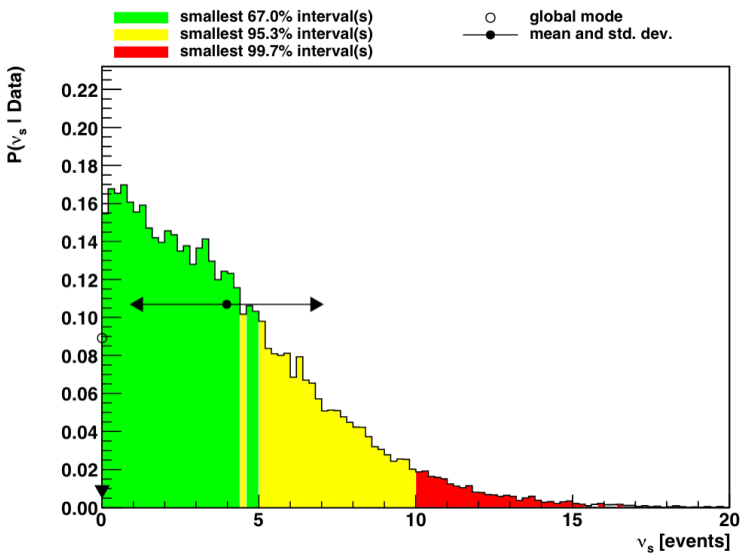
2. Compile & run.

Exercise 2 – Uncertain Background – Solution

1. #include <BAT/BCGaussianPrior.h>

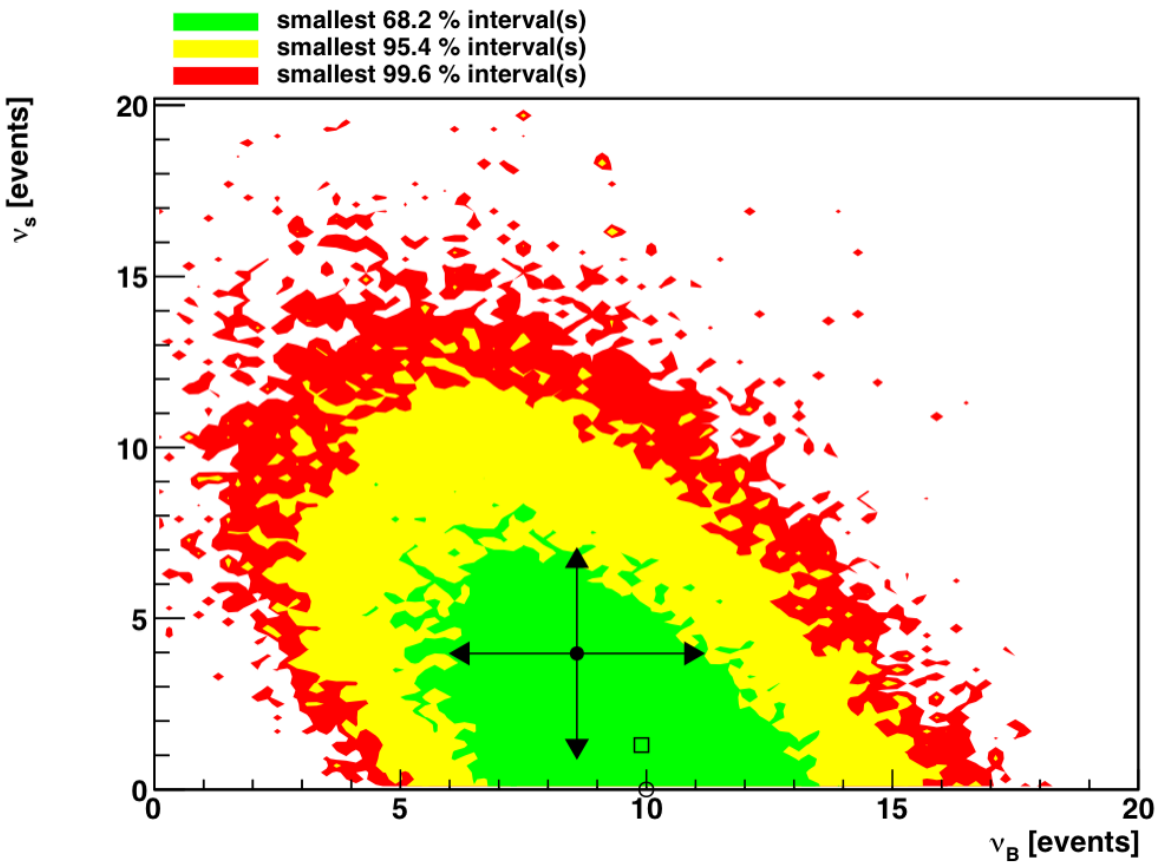
   GetParameter("B").SetPrior(new BCGaussianPrior(10, 3));





90% upper limit on signal: 8.2

95% upper limit on signal: 9.8



Exercise 3 – Update of Knowledge

AIM: Check update of knowledge

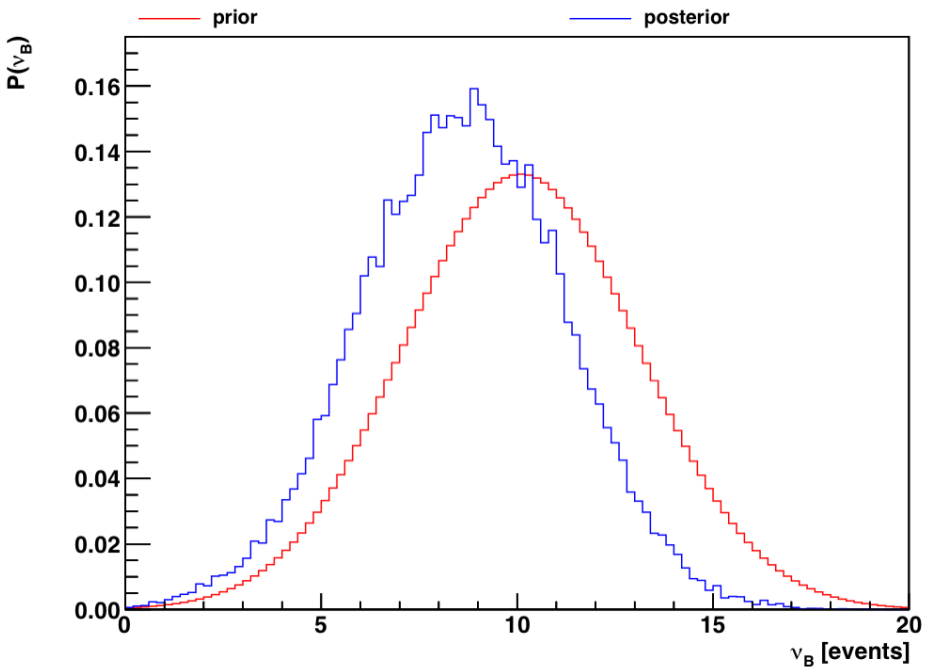
Steps:

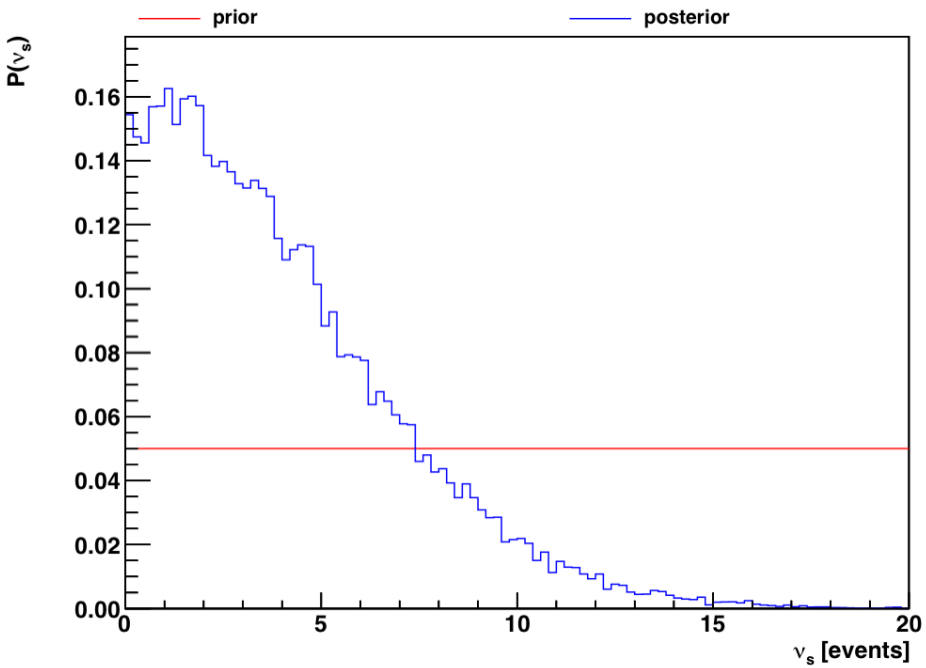
1. Print knowledge update.

Exercise 3 – Update of Knowledge – Solution

1. In runTutMod.cxx uncomment line

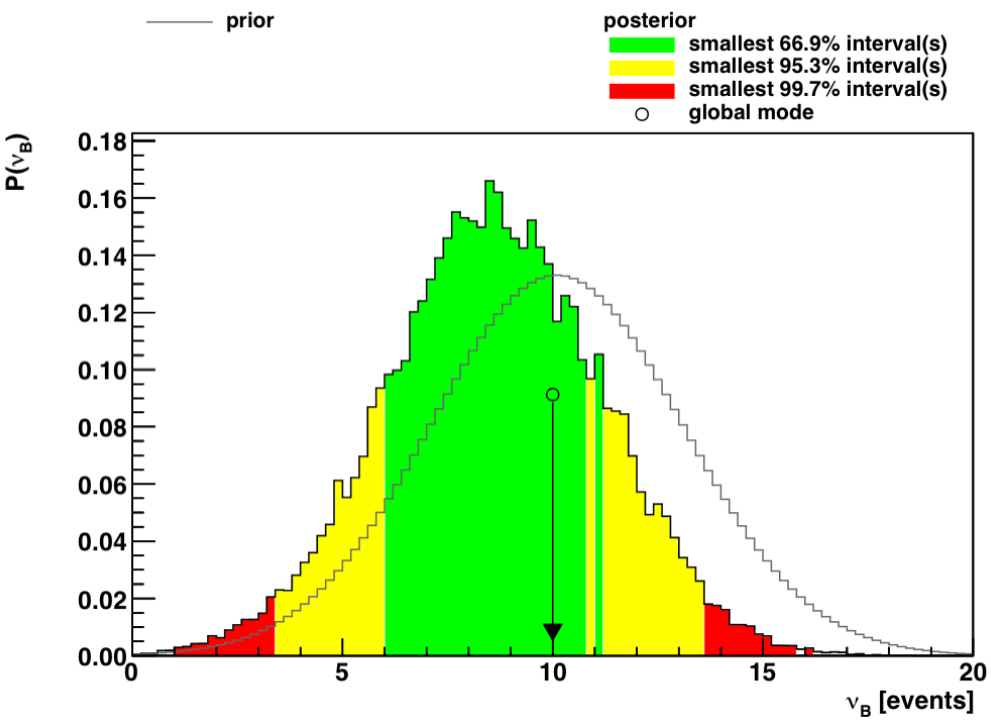
```
m.PrintKnowledgeUpdatePlots(m.GetSafeName() + "_update.pdf");
```

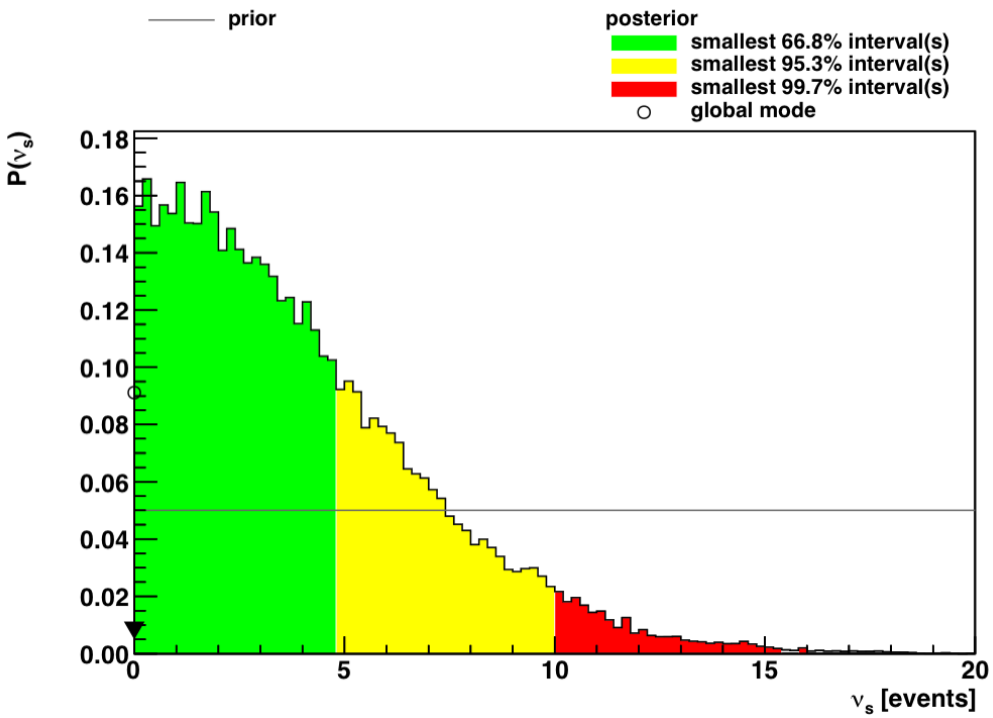




1'. Now try modifying drawing style:

```
m.SetKnowledgeUpdateDrawingStyle(BCAux::kKnowledgeUpdateDetailedPosterior);
m.PrintKnowledgeUpdatePlots(m.GetSafeName() + "_update.pdf");
```





Try:BCAux::kKnowledgeUpdateDetailedPrior

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*?

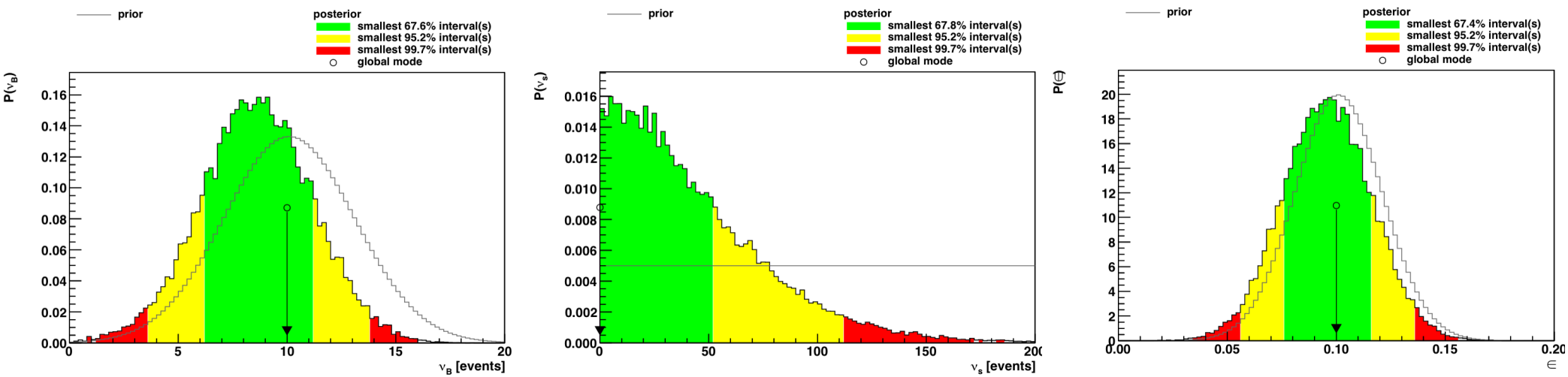
Exercise 4 – Signal Detection Efficiency – Solution

1. In constructor ...

```
AddParameter("S", 0, 200);
AddParameter("eps", 0, 0.20, "#epsilon");
GetParameter("eps").SetPrior(new BCGaussianPrior(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 = 111

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

Steps:

1. Add Observable for observed number of events

```
AddObservable(string name, double min, double max, string latex, string units)
```

2. Add function

```
void CalculateObservables(const vector<double>& p);
```

to calculate observable.

Note: you can set observable with:

```
GetObservable([int index]) = [value];
```

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

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

1. In constructor:

```
AddObservable("obsS", 0, 20, "#nu_{S}^{obs}", "[events]");
```

2. In TutMod.h add:

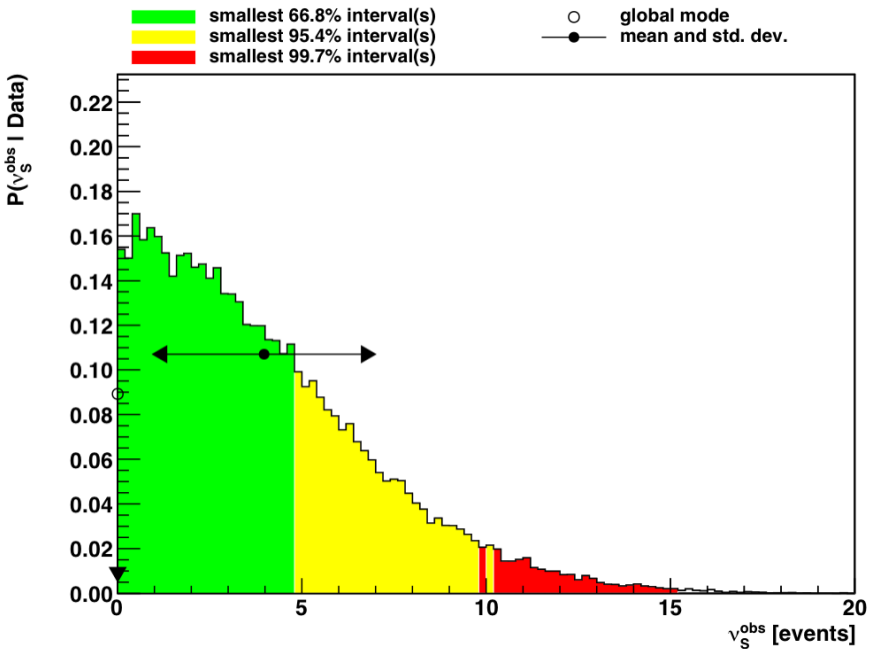
```
void CalculateObservables(const std::vector<double>& p);
```

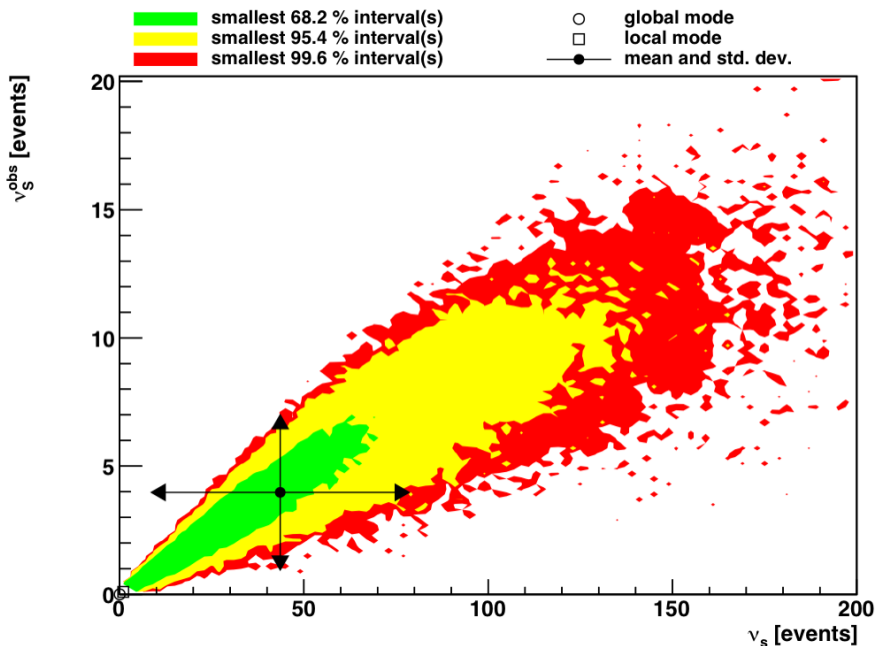
In TutMod.cxx add:

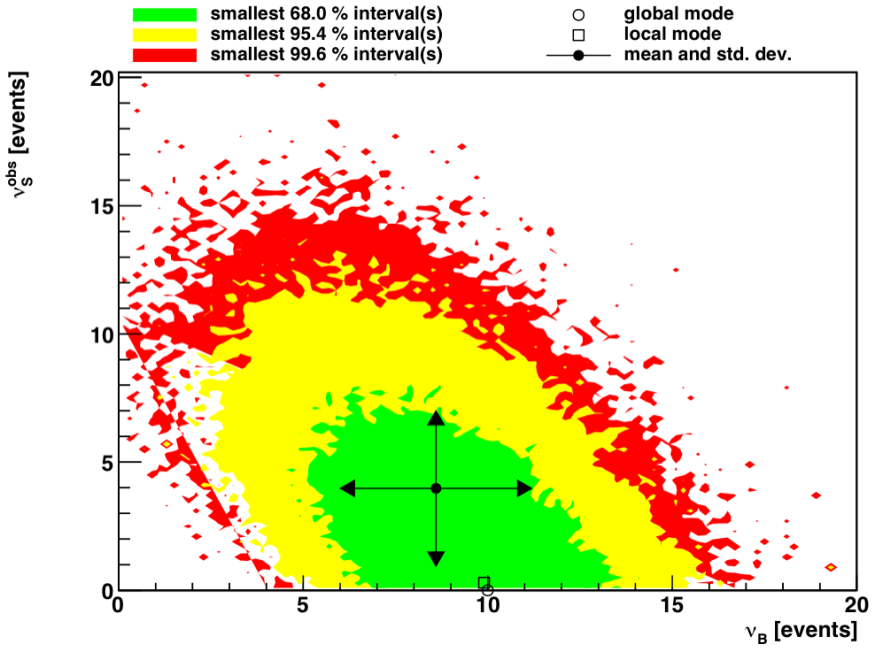
```
void TutMod::CalculateObservables(const std::vector<double>& p)
{
    double S    = p[1];
    double eps = p[2];
    GetObservable(0) = eps * S;
}
```

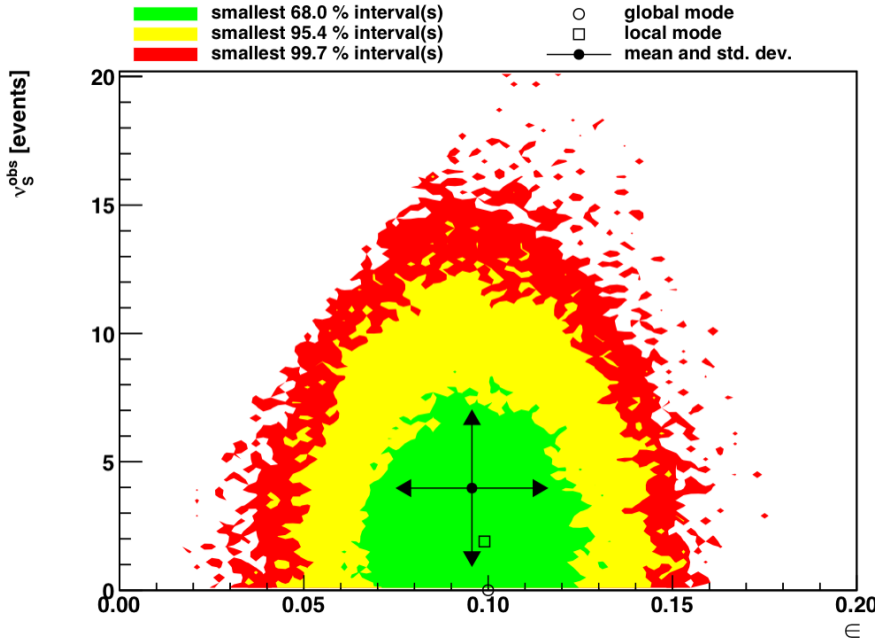
Compile & Run

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









95% Upper limit = 9.8 events

Exercise 6 – Multiple runs

Try the problem again now with a dedicate background-only measurement:

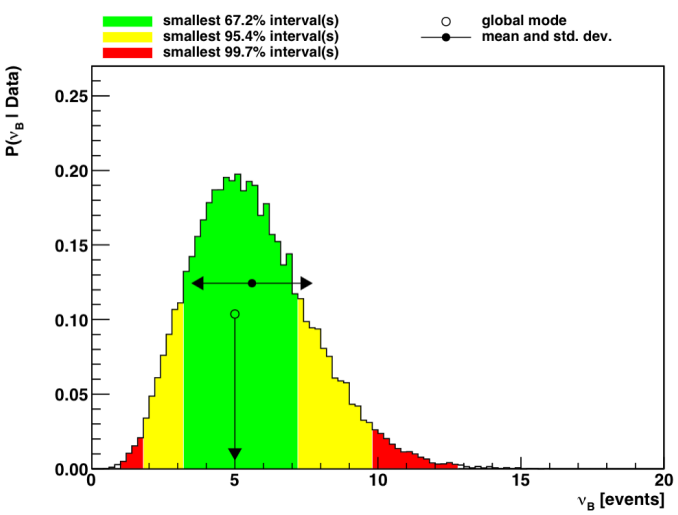
1. Set the background prior to flat.
2. Add a second measurement and adjust the prior accordingly.

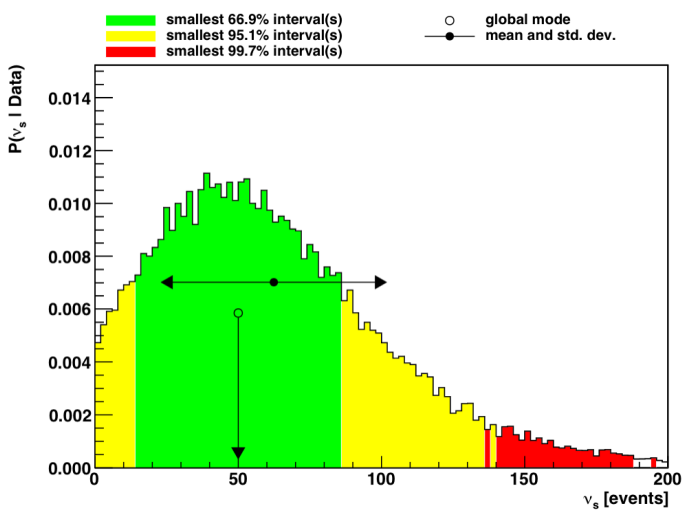
Exercise 6 – Multiple runs

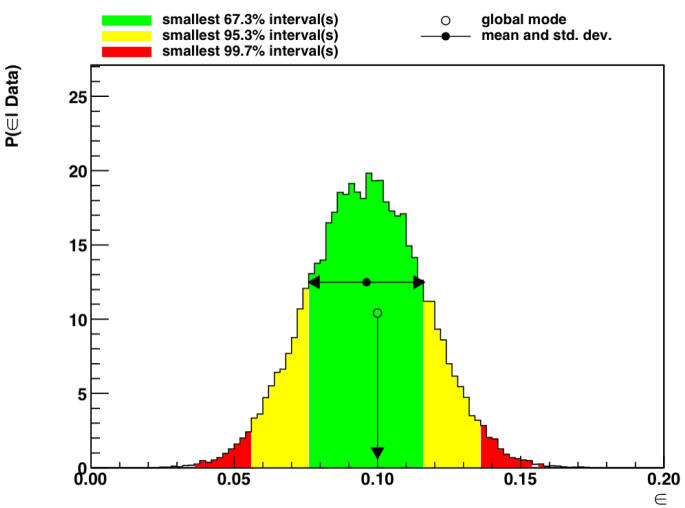
1. GetParameter("B").SetPriorConstant();

2. double TutMod::LogLikelihood(const std::vector<double>& parameters)

```
{
    double N = 10;
    double Nb = 5;
    double B = parameters[0];
    double S = parameters[1];
    double epsS = parameters[2];
    return BCMath::LogPoisson(Nb, B) + BCMath::LogPoisson(N, B + S * eps);
}
```







Exercise 6 – Choice of Priors

Repeat your analysis with different priors

e.g. an exponential one,

a Gaussian one,

a Split Gaussian one

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 inside the BAT software directory in examples