Оценка параметров статистической модели средствами пакетов RooStats и HistFactory на примере байесовского анализа

П. Мандрик

ФГБУ ГНЦ ИФВЭ

НИЦ “Курчатовский институт”

12.05.2017

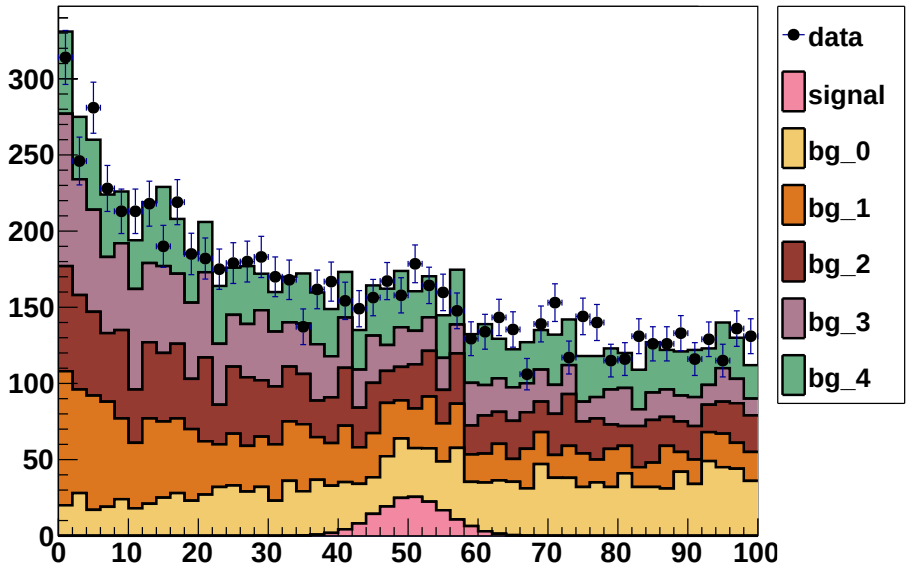
Постановка задачи

Имееется:

- Экспериментально измеренное дискретное распределения числа событий по какой-либо переменной - гистограмма **data**
- Гистограммы для фоновых процессов, полученные из Монте-Карло - **bg_1**, **bg_2**, **bg_3**, ...
- Монте-Карло гистограмма для сигнального процесса - **signal**
- Теоретическая модель, предсказывающая зависимость вида гистограммы для сигнального процесса от параметра модели

Требуется:

- Оценить величину параметра теоретической модели (одностороннее ограничение, интервал и т.д.) на основе экспериментального измерения, с учётом имеющихся систематических и статистических ошибок



Алгоритм решения:

1. Задание статистической модели измерения
2. Применение какого-либо метода оценки параметров модели (байесовский анализ, частотный подход, CLs метод и др.)

Инструменты:

1. **HistFactory** - набор c++ классов для создания основанных на гистограммах статистических моделей (параметрических функций распределения плотности вероятности) в формате библиотеки **RooFit** для дальнейшего использования в **RooStats**. Для задания моделей могут быть использованы конфигурационные xml-файлы и c++ или python интерфейс.

   https://cds.cern.ch/record/1456844/files/CERN-OPEN-2012-016.pdf ⇒

2. **RooStats** - набор инструментов для статистического анализа, созданный на основе библиотеки **RooFit** и распространяемый вместе с пакетом **ROOT**.

Плюсы RooStats + HistFactory:

- Известен, открыт, бесплатен, широко распростаранён, рекомендован к использованию крупных экспериментах (**CMS**,**ATLAS** и т.д.)
- Использует инструменты из **ROOT**
- Поддержка **ROOT**-команды, много презентаций
- Содержит широкий набор методов для статистического анализа
- Высококонфигурироемое символьное задание модели без необходимости написания дополнительного **c++** кода

Минусы:

- При проведении анализа прийдётся работать с **ROOT**, **RooFit**, **RooStats**, **c++**
- Функция правдоподобия из **HistFactory** менее оптимизирована в сравнении с другими специализированными пакетами (**theta**, **CombinedLimit** и д.р.), большее время вычислений
- Плохо работает как “чёрный ящик”

Создание статистической модели в HistFactory

Рассмотрим последовательный процесс задания статистической модели с помощью c++ интерфейса (использовалась версия ROOT v6-06-00):

```
// define model
RooStats::HistFactory::Measurement myModel("myModelName");

// define data channel
RooStats::HistFactory::Channel myChannel("myChannelName");
myChannel.SetData(dataHistName, inputFileName, path);
```

Measurement - базовый класс для задания модели.
Channel - отдельный регион/гистограмма данных.
HistFactory позволяет задавать многоканальные модели с общими для разных каналов параметрами.

Добавим в данный канал Монте-Карло гистограмму процесса:

```
RooStats::HistFactory::Sample signalSample("sampleName",
    sampleHistName, inputFileName, path);
myChannel.AddSample( signalSample );
```

Аналогично добавляются гистограммы других процессов. Будем для простоты рассматривать один сигнальный и один фоновый процесс, тогда функция правдоподобия будет иметь вид:

$$\mathcal{L} = \prod_{i}^{N_{bins}} \mathcal{P}(d_i|s_i + b_i)$$

где d_i, s_i, b_i - значения в i-том интервале гистограммы данных, сигнального и фонового процесса соотвественно.

$\mathcal{P}(d_i|s_i + b_i) = \frac{(s_i+b_i)_i^d e^{-(s_i+b_i)}}{d_i!}$ - распределение Пуассона.

Введём в модель параметр (θ), как нормализующий фактор:

```
RooStats::HistFactory::NormFactor myParameter;
myParameter.SetName( "theta" );
// set parameter default value and range of definition
myParameter.SetVal ( 1.0 );
myParameter.SetLow ( 0.5 );
myParameter.SetHigh( 1.5 );

// add parameter to signal sample
signalSample.AddNormFactor( myParameter );
```

Теперь статистическая модель включает параметр θ:

$$\mathcal{L} = \prod_{i}^{N_{bins}} \mathcal{P}(d_i|\theta \cdot s_i + b_i)$$

Систематические ошибки данных

Помимо информативных параметров, значения которых требуется оценить при анализе, в модель часто включают неинформативные параметры для учёта имеющихся систематических ошибок. Пусть нормировки фонового процесса известна с систематической ошибкой (сечение, светимость и т.д.). Учтем это введя новый параметр (ν):

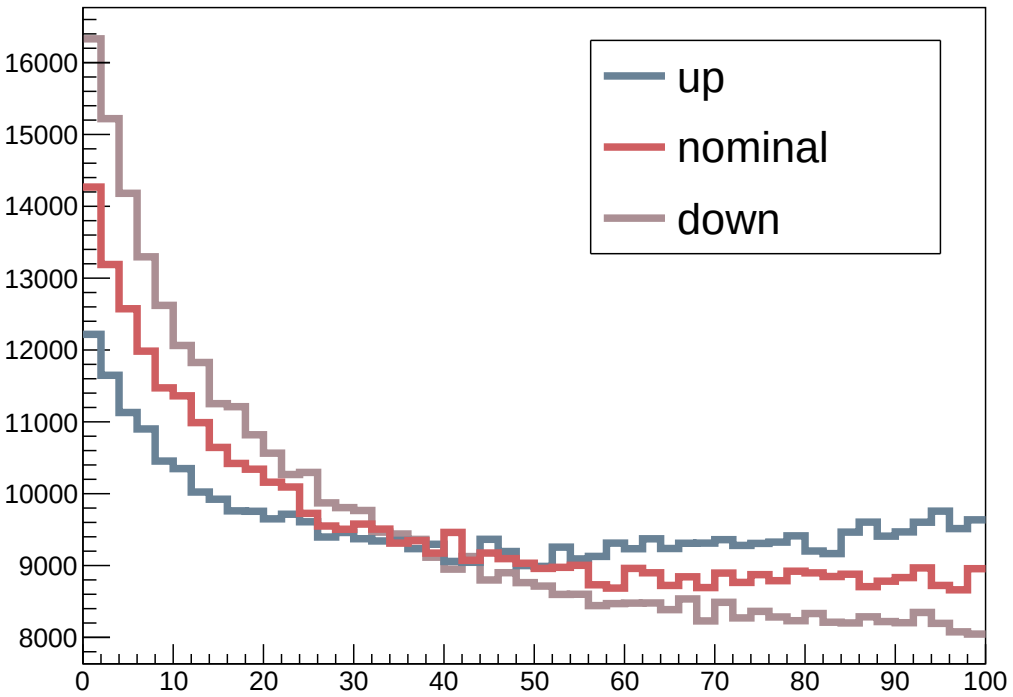
```
RooStats::HistFactory::NormFactor sigmaBg;
sigmaBg.SetName( "nu" );
sigmaBg.SetVal ( 1.0 );
sigmaBg.SetLow ( sigmaBg_max );
sigmaBg.SetHigh( sigmaBg_min );

// add parameter to signal sample
backgroundSample.AddNormFactor( sigmaBg );
```

$$\mathcal{L} = \prod_{i}^{N_{bins}} \mathcal{P}(d_i | s_i + \nu \cdot b_i)$$

Далее для ν может быть также добавлено априорное распределение.

Систематическая ошибка может менять форму гистограммы (Pile-Up, b-tagging и другие). Такая ошибка учитывается в модели, как параметр интерполяции между номинальной гистограммой и гистограммами, отвечающими возможной вариации формы гистограммы:



“nominal” - гистограмма, полученая при номинальном значении известного с систематической ошибкой параметра
“plus”, **“minus”** - гистограммы, полученные при смещённом в пределах ошибки его значении

Добавим в **HistFactory** параметр α_{sys}, имеющий область определения $[-5, 5]$ и распределённый по Гауссу с шириной $1.$ и средним $0.$:

```
backgroundSample.AddHistoSys("sys",
  sysHistNameUp   , inputFileNameUp,   path,
  sysHistNameDown, inputFileNamedown, path);
```

Что в терминах статистической модели будет означать:

$$\mathcal{L} = \prod_{i}^{N_{bins}} \mathcal{P}(d_i | s_i + I(\alpha_{sys}, b_i, b_i^+, b_i^-)) \cdot \mathcal{G}(\alpha_{sys}|0, 1)$$

Есть несколько видов возможной интерполирующей функции. По умолчанию используется линейная интерполяция:

$$I(\alpha_{sys}, b_i, b_i^+, b_i^-) = \begin{cases} b_i + \alpha_{sys} \cdot (b_i^+ - b_i), & \alpha \geq 0 \\ b_i + \alpha_{sys} \cdot (b_i - b_i^-), & \alpha < 0 \end{cases}$$

Систематические ошибки, связанные с конечностью статистики МК

Используется метод **Барлоу-Бистон "light"** - вводится один эффективный параметр γ_i на i-тый интервал гистограммы:

$$\mathcal{L} = \prod_{i}^{N_{bins}} \mathcal{P}(d_i|\gamma_i) \cdot \mathcal{G}(s_i + b_i|\gamma_i, \sigma_{stat})$$

где σ_{stat} - суммарная статистическая ошибка всех МК-гистограмм. Если суммарная относительная статистическая ошибка меньше 5%, то HistFactory дополнительный параметр не вводит.

```
// Activate Statistical errors for signal and background
signalSample.ActivateStatError();
backgroundSample.ActivateStatError();
```

Подробнее https://ghl.web.cern.ch/ghl/html/BarlowBeeston.html ⇒

Составные параметры

Приведём пример добавления в статистическую модель более сложной параметрической зависимости от R и L:

$$\mathcal{L} = \prod_{i}^{N_{bins}} \mathcal{P}\left(d_i | \frac{L^2}{\sqrt{L^2+R^2}} \cdot s_i^L + \frac{R^2}{\sqrt{L^2+R^2}} \cdot s_i^R\right)$$

```
// add parameters as usual
RooStats::HistFactory::NormFactor parL;
parL.SetName( "parL" );
signalSampleL.AddNormFactor( parL );

RooStats::HistFactory::NormFactor parR;
parR.SetName( "parR" );
signalSampleR.AddNormFactor( parR );

// override W1, W2 parameters with custome functions
model.AddPreprocessFunction("parL", "L*L/sqrt(L*L+R*R)",
            "L[1,0,2],R[1,0,2]");
model.AddPreprocessFunction("parR", "R*R/sqrt(L*L+R*R)",
            "L[1,0,2],R[1,0,2]");
```

Дальнейшее использование Histfactory модели в RooStats

После того, как модель задана, она импортируется в **RooWorkspace**, который можно продолжить использовать или сохранить в **.root**-файл:

```
// build histsfactory model and import into RooWorkspace
myModel.CollectHistograms();
RooStats::HistFactory::HistoToWorkspaceFactoryFast factory;
RooWorkspace* myWorkspace = factory.MakeCombinedModel(myModel);
```

В созданном **RooWorkspace** есть **ModelConfig**, хранящий информацию о модели и используемый **RooStats**-алгоритмами:

```
// get histfactory modelconfig
RooStats::ModelConfig* myModelConfig = (RooStats::ModelConfig*)
    myWorkspace->obj("ModelConfig");
// set some custom options
myModelConfig->SetParametersOfInterest( myPois );
myModelConfig->SetNuisanceParameters( myNuisances );
myModelConfig->SetPriorPdf( myPrior );
```

Рассмотрим пример добавления неинформативного параметра и априорного распределения для него:

```
// define nuisance list
RooArgList myNuisances("myNuisancesList");
// get my parameters from workspace and add to list
myNuisances.add( *myWorkspace->factory( "myParName" ) );

// define priors list
RooArgList myPriors("myPriorsList");
// produce gaussian prior with mean = 1.0 and width = 0.5
RooAbsPdf * myGausPrior = workspace->factory("Gaussian::
    prior_myParName(myParName,1.0,0.5)");
myPriors.add( myGausPrior );

// define final prior as a production of priors in list
RooProdPdf myPrior("myPrior", "myPrior", myPriors);
```

Схожим образом вводится информативнык параметры. Процедура обработки информативных и неинформативных параметров зависит от выбранного статистического метода и его реализации. Формально в Байесовском анализе параметры не различаются.

Байесовский анализ

В пакете **RooStats** реализован ряд методов для постановки ограничений на параметры моделей. Далее мы рассмотрим работу с полученной из **HistFactory** моделью на примере байесовского анализа.

Байесовский анализ основан на применении формулы Байеса:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Которая для измерения **data** и набора параметров модели $\{\theta_i\}$ примет вид* :

$$\mathcal{P}(\theta_1, \theta_2, \theta_3, ...|data) \propto \mathcal{P}(data|\theta_1, \theta_2, \theta_3, ...) \prod_i P(\theta_i)$$

$$\mathcal{P}(\theta_j) = \int \mathcal{P}(\theta_1, \theta_2, \theta_3, ...|data) \prod_{i \neq j} d\theta_i$$

*если априорное распределение модели факторизуется по параметрам

Плюсы Байесовского анализа в практическом анализе:

- Меньшая сложность вычислений, что означает возможность проведения анализа без необходимости приближений, например, применения минимизации по неинформативным параметрам
- Отсутствуют методологические сложности с числом размерностей параметров - можно ставить двухмерные ограничения и т.д.

Минусы:

- Как правило, априорные распределения известны не для всех параметров и в модель включаются неинформативные априорные распределения

В **RooStats** есть два класса для проведения Байесовского анализа:

- **MCMCCalculator** использует метод марковских цепей Монте-Карло
- **BayesianCalculator** использует численное интегрирование, предоставляемые пакетом **ROOT**

У **MCMCCalculator** есть несколько параметров:

- используемая **ProposalFunction** - функция для выбора следующей точки в марковской цепи.
  В **RooStats** обычно используется **SequentialProposal**, которая на каждом шаге сдвигает в пространстве параметров одну переменную:

```
val += RooRandom::gaussian() * len * fDivisor;
```

  где len - длина интервала определения параметра, $fDivisor$ - параметр $\Rightarrow$ нужно настраивать оптимальные значения вручную.

```
// setup MCMC proposal function
RooStats::SequentialProposal myProposal(0.01);

// setup MCMC calculator
RooStats::MCMCCalculator myCalculator(*myWorkspace->data("obsData
    "), *myModelConfig);
myCalculator.SetProposalFunction(myProposal);
```

Так же можно попробовать **ProposalHelper**.

- **NumIters** - длина цепи
- **NumBurnInSteps** - невзвешанное число точек, которые будут исключены для уменьшения влияния выбора начальной точки

```
myCalculator.SetNumIters( myChainLenght );
myCalculator.SetNumBurnInSteps(0);
// actually we dont use this options but should set them anyway
myCalculator.SetConfidenceLevel(0.5);
myCalculator.SetLeftSideTailFraction(0.5);
```

Запуск построения цепи:

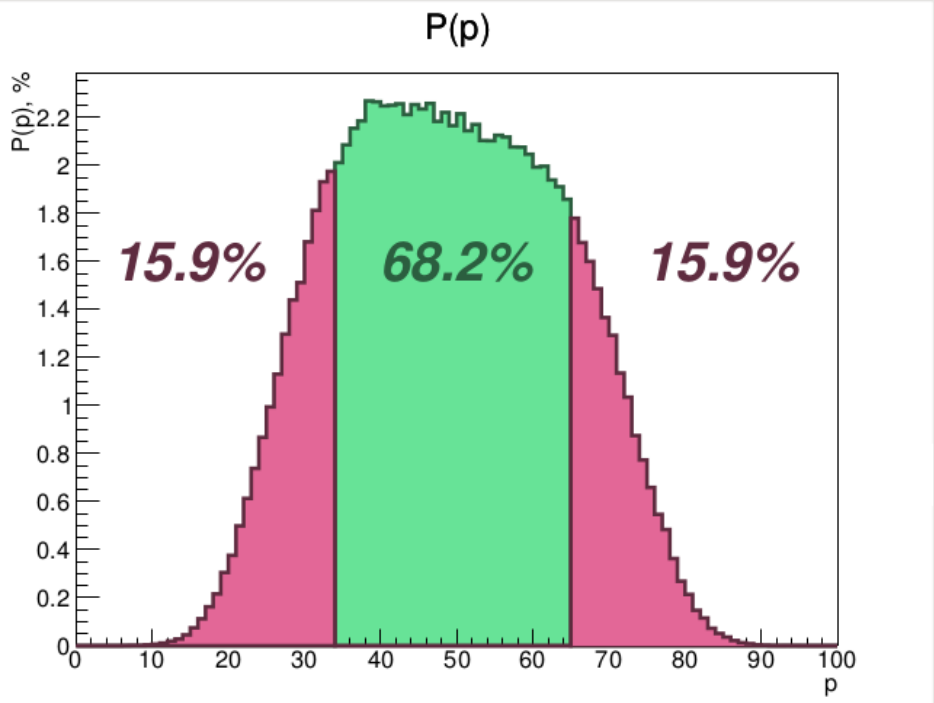
```
// produce chain
RooStats::MCMCInterval * mcInt = mc.GetInterval();
const RooStats::MarkovChain* myChain = mcInt->GetChain();
```

Пример записи результата в гистограмму:

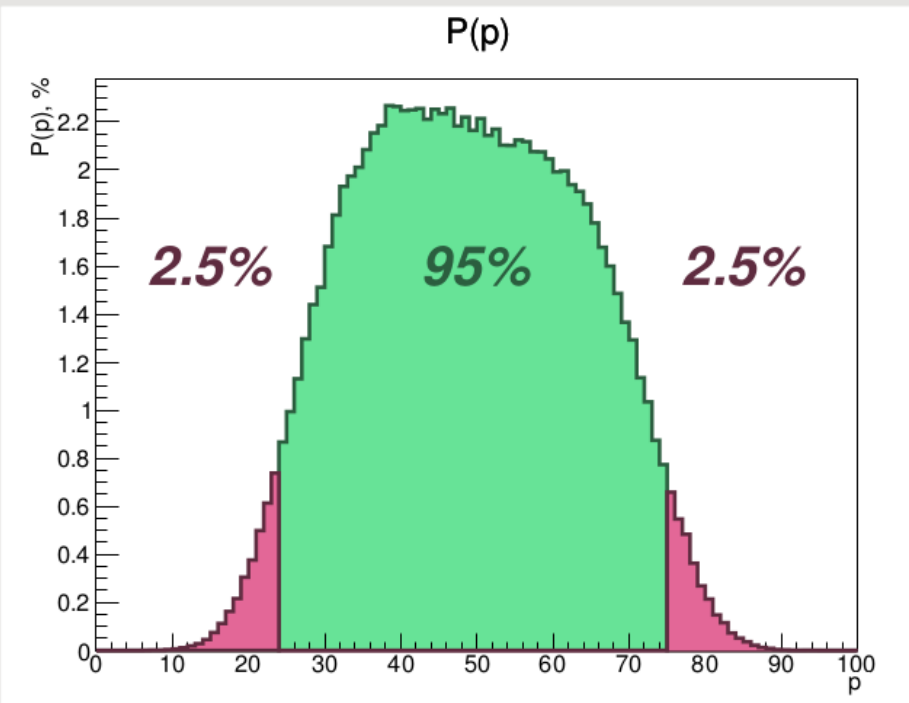
```
for(int i = 0; i < myChain->Size(); ++i){
  const RooArgSet * argset = chain->Get( i );
  double val = argset->getRealValue("myParName");
  double weight = chain->Weight(i);
  myParHist->Fill(val, weight);
}
```

Таким образом результатом баесовского статистического анализа будет является набор гистограмм, соответствующих функциям плотности распределения вероятности параметров. По ним могут быть найдены верхние ограничения, двухсторонние интервалы и т.д.:

интервал 1σ:



95% интервал:



Ожидаемое ограничение

Другой обычной задачей является нахождение “ожидаемого” ограничения - ограничения на параметр модели в предположении верности альтернативной модели, как правило, допускающей наличие только фона. Для этого из альтернативной модели генерируется набор Монте-Карло гистограмм:

```
// Generate toy histogram over "obs_x_myData"
RooDataHist * toyHist = modelPDF.generateBinned( *workspace->
    factory("obs_x_myData"));
// Example of per bin access
for(int i = 1; i <= nBins; i++){
  toyHist->get(i);
  cout << i << "␣" << toyHist->weight() << endl;
}
```

При этом используются текущие значения прочих параметров модели ($\rightarrow$ нужно разыгрывать их отдельно, если необходимо). Из Монте-Карло гистограммы обычным способом находится ограничение. Распределение ограничения по сгенерированному набору используется для нахождения среднего “ожидаемого” ограничения и его $1-2\sigma$ отклонений.

Заключение

- Постановка ограничений на параметры теоретических моделей является распространённой задачей в физике высоких энергий
- Увеличение доступных вычислительных ресурсов подтолкнуло развитие статистических методов и програмного обеспечения, применение в анализе основанных на MK-статистике сложных многоканальных моделей больших размерностей по параметрам
- RooStats, HistFactory и баесовский формализм являются одним из возможных вариантов для анализа
- Пример задания модели и нахождения ограничений:
  https://github.com/pmandrik/Other/blob/master/e00_histfactory_bayesian_mcmc.C
  $\Rightarrow$